in collaboration with

Softwarica College of IT & E-commerce | Coventry University

Assignment Title:
Individual Coursework of Big Data Programming Project

Module Code:
ST5011CEM

Date of Submission:
07 February 2026

Submitted by:

240226
Dikshatna Chapagain

Submitted to:

Siddhartha Neupane

Acknowledgement

I want to extend my heartfelt thanks to **Siddhartha Neupane Sir** for his invaluable support and expert guidance throughout this project. His encouragement, knowledge, and continuous motivation played a crucial role in helping me successfully complete this work.

I am also deeply grateful to the contributors of the open data platforms, research communities, and development tools that made this project possible. Their shared knowledge and resources greatly supported the learning and implementation process.

Table of Contents

Executive Summary *7*

Introduction *8*

Problem Statement 9

Purpose and Scope 10

Relevance 11

Learning Outcomes Targeted 11

Literature Review / Background *12*

Data Collection & Preprocessing *13*

Data Sources 13

Tools & Technologies 13

Data Ingestion and Storage 14

Data Cleaning 16

Data Preprocessing 18

Challenges Encountered 20

Methodology *20*

Feature Engineering 21

Model Selection 22

Justification 23

Model Comparison 25

Data Splitting 25

Algorithm Complexity 26

System Design and Implementation *30*

Architecture Overview 30

Software Stack 32

Security Considerations 33

User Interface - Streamlit Dashboard 34

Results and Evaluation *36*

Model Performance Metrics 36

Visualization Results *39*

Exploratory Data Analysis 39

Model Comparison Visualizations 49

Model Evaluation Visualizations 53

System Evaluation **60**
Validation Methodology **60**
Critical Reflection ***62***
Working Mechanism **62**
Limitations of the Approach **63**
References ***66***
Appendix ***67***

Table of Figures

Figure 1: Problem Statement 9
Figure 2: Purpose and Scope 10
Figure 3: Data Source 13
Figure 4: Tools and Technologies 13
Figure 5: Data Ingestion and Storage: 1 14
Figure 6: Data Ingestion and Storage: 2 14
Figure 7: Data Ingestion and Storage: 3 15
Figure 8: Data Ingestion and Storage: 4 15
Figure 9: Data Ingestion and Storage: 5 16
Figure 10: Data Cleaning: 1 17
Figure 11: Data Cleaning: 2 17
Figure 12: Data Cleaning: 3 18
Figure 13: Data Preprocessing: 1 19
Figure 14: Data Preprocessing: 2 19
Figure 15: Methodology 20
Figure 16: Feature Engineering: 1 21
Figure 17: Feature Engineering: 2 22
Figure 18: Model Selection: 1 24
Figure 19: Model Selection: 2 24
Figure 20: Data Splitting: 1 25
Figure 21: Data Splitting: 2 26
Figure 22: Model Comparison 26
Figure 23: Algorithm Complexity: 1 27
Figure 24: Algorithm Complexity: 2 27
Figure 25: Algorithm Complexity: 3 28
Figure 26: Algorithm Complexity: 4 28
Figure 27: Algorithm Complexity: 5 29
Figure 28: Algorithm Complexity: 6 29
Figure 29: Architecture Overview: 1 30
Figure 30: Architecture Overview: 2 31
Figure 31: Software Stack 32
Figure 32: Security Considerations 33

Figure 33: User Interface: 1 34
Figure 34: User Interface: 2 35
Figure 35: User Interface: 3 35
Figure 36: Model Evaluation: 1 37
Figure 37: Model Evaluation: 2 38
Figure 38: EDA: Distribution of Bus Runtime 39
Figure 39: EDA: Distance vs Runtime Correlation 40
Figure 40: EDA: Average Runtime by Hour 41
Figure 41: EDA: Rush Hour Impact on Runtime 42
Figure 42: EDA: Top 10 Most Frequent Routes 43
Figure 43: EDA: Runtime by Direction 44
Figure 44: EDA: Distribution of Journey Distances 45
Figure 45: EDA: Runtime by Stop Sequence 46
Figure 46: EDA: Runtime by Time of Day 47
Figure 47: Visualization Results 48
Figure 48: R^2 Score 49
Figure 49: Mean Absolute Error 50
Figure 50: Root Mean Squared Error 51
Figure 51: Training Time Comparison 52
Figure 52: Actual vs Predicted Values 53
Figure 53: Residual Plot 54
Figure 54: Error Distribution 55
Figure 55: Top 10 Feature Importance 56
Figure 56: Error Distribution (Box Plot) 57
Figure 57: Q-Q Plot (Residuals Normality) 58
Figure 58: Model Evaluation Visualizations 59

Executive Summary

This report details a data science project aimed at predicting bus journey runtimes for Falcon Coaches using historical timetable data from the UK's Bus Open Data Service (BODS). The primary objective was to develop a robust machine learning model capable of accurately estimating the travel time between two consecutive bus stops. The project encompassed a complete data science workflow, beginning with the ingestion and preprocessing of complex XML-formatted timetable data, which was converted into a structured CSV format.

Utilizing a technology stack centered on Apache Spark with PySpark for large-scale data processing, the project implemented comprehensive feature engineering, including Haversine distance calculation and the extraction of temporal features like rush hour indicators. Exploratory Data Analysis (EDA) was conducted to uncover patterns in the data. Four distinct regression models were trained and evaluated: Linear Regression, Decision Tree, Gradient Boosted Trees, and Random Forest.

The Random Forest Regressor was identified as the superior model, demonstrating exceptional performance with an r^2 score of approximately 0.96 on unseen test data. This indicates that the model can explain about 96% of the variance in bus runtimes. The project culminated in the development of an interactive command-line prediction system that leverages the trained model to provide real-time journey time estimates, successfully translating the analytical findings into a practical application.

Introduction

The project estimates bus journey runtime in Falcon Coaches with the help of past UK Bus Open Data. It addresses the issue of unpredictable travel time, enhancing the experience of passengers and operational efficiency. With Apache Spark and PySpark, the project will be spanning data ingestion, preprocessing, feature engineering, and model evaluation will produce an incredibly precise Random Forest regression model that drives an interactive prediction system to smarter urban mobility.

Figure 1: Big Data Introduction

Problem Statement

Transport system within cities is part of day-to-day operations of contemporary cities, however, they tend to be marred by unpredictability. In the case of users of transport through buses, inconvenience due to the non-predictability of the travel time can be a big inconvenience such as missed buses and long queues. This uncertainty is affected by a myriad of factors including traffic congestion, time of the day and route peculiarities and precise scheduling is a complicated task to transport operators.

Figure 2: Problem Statement

Purpose and Scope

This project is aimed at solving the issue of travel time uncertainty through the establishment of a predictive machine learning model. This scope is centered on the analysis of the historical data of the timetables of one operator, Falcon Coaches, to forecast the RunTime_Minutes of the individual segments of the journey (the time spent on the way between two consecutive stations). The project will investigate the opportunities provided by using data science methods on raw transport data to create some meaningful and useful insights.

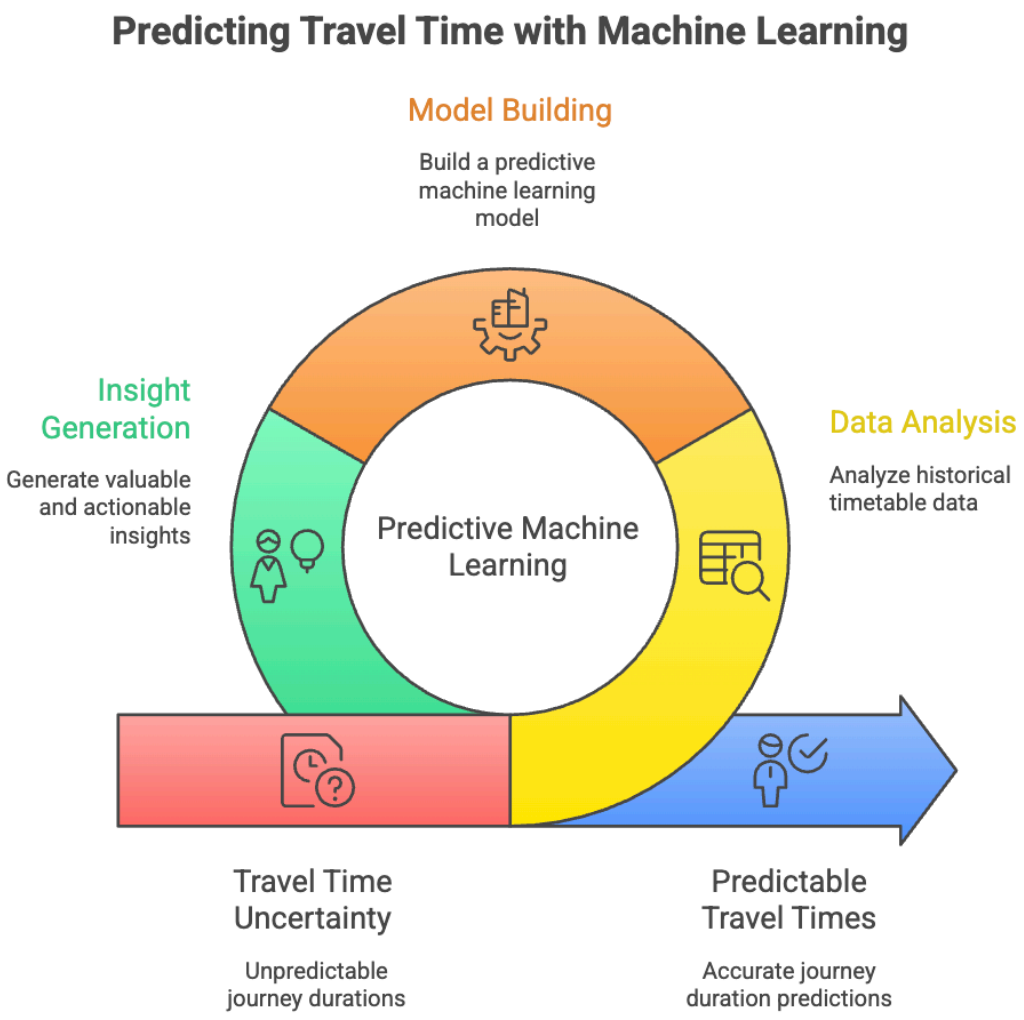


Figure 3: Purpose and Scope

Relevance

The creation of precise runtime prediction model is very relevant to the passengers and the transport authorities. To the passengers, it promises to provide them with the possibility of better travel and a more reliable journey planning. Such a model can provide information to operators and city planners on how to use dynamic scheduling, understanding operational bottlenecks and make more efficient and resilient of their public transport systems. This is in line with larger smart city projects which aim at using the data to enhance urban mobility as well as the management of resources.

Learning Outcomes Targeted

Learning Outcome	Title	Description
B1	Advanced Data Processing	Converted semi-structured XML transport data into structured datasets through parsing, cleaning, and temporal feature extraction.
B2	Big Data Technologies	Applied Apache Spark and PySpark DataFrames for scalable ETL processing and distributed machine learning model training.
B3	Machine Learning Implementation	Built and trained a Random Forest regression model using engineered temporal, spatial, and route-based features.
B4	Model Evaluation	Evaluated model performance using R^2, MAE, and RMSE metrics to validate prediction accuracy.
B5	Data Visualization	Developed interactive maps and dashboards to visualize routes, stops, and predicted travel times.
B6	System Integration	Implemented an end-to-end pipeline from data ingestion to model deployment with a Streamlit web interface.
B7	Problem-Solving	Addressed segment-level prediction challenges through optimized feature engineering and model design.
B8	Documentation & Communication	Produced clear technical documentation and an interactive interface to communicate results to stakeholders.

Literature Review / Background

The city transport network is becoming more and more propelled by mass data generated by GPS, Automated vehicle location (AVL) and smart ticketing infrastructure. The technologies allow the transit agencies to track operations on the fly and assist with predictive analytics to enhance passenger experience and service reliability. The big data computing engines like Apache Spark have enabled processing semi-structured transit data such as GTFS on a large scale to make computationally efficient analysis of millions of spatial-temporal records to plan smart city transportation.

Past methods of predicting public transport used statistical and time-scales forecasting like the Kalman filtering and ARIMA forecasting methods which work well within short term predictions when the traffic is predictable. But these models have problems with non-linear traffic dynamics and compounded feature interactions at an urban scale. Recent researchers demonstrate machine learning approaches, especially ensemble models such as Random Forest, are better than traditional methods because they can model non-linear relationships among temporal, spatial, and route-based predictors, whereas deep learning models are highly accurate but cannot be interpreted or are computationally infeasible.

The research takes the form of a Random Forest regression model, which will be enacted through PySpark MLlib that will be used in predicting segment-level bus travel times. This model takes advantage of the rich set of features such as temporal, spatial and route features, which allow it to make strong predictions about congestion patterns. PySpark can support the scale needed to handle large urban data, and the use of Random Forest allows achieving a compromise between prediction and understandability, which is why it is an appropriate solution to the conditions of the actual transit activity.

Data Collection & Preprocessing

Data Sources

The UK Bus Open Data Service (BODS) is the main source of data in this project as it offers timetable data in TransXChange XML format to the wider population. To analyse 237 XML files that are specific to Falcon Coaches, a Python script was used. This script identifies the detailed services, journeys, stops, and times and summarizes it in one analysis-ready CSV file.

Figure 4: Data Source

Tools & Technologies

Data preprocessing pipeline was designed based on the PySpark application, which is highly applicable in processing the large and structured data. PySpark has distributed computing functionality enabling efficient transformations and aggregations of tens of thousands of records. Pandas was also employed, and it was mainly used to inspect the data and also to provide easier visualizations, which could be created by using in-memory data frames.

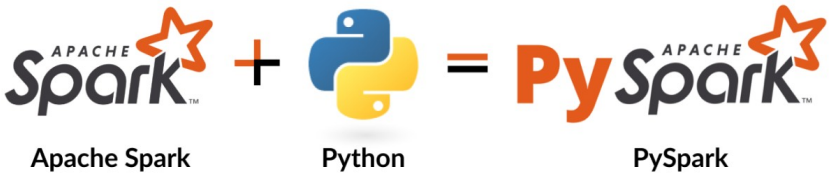


Figure 5: Tools and Technologies

Data Ingestion and Storage

XML bus schedules BODS had (50+ XML bus timetable files) were transformed into one CSV with Python which retrieved routes, stops, coordinates, and travel time. Every stop-to-stop was saved as a separate record and with the help of PySpark, the 1GB+ dataset was loaded and validated in an efficient manner to make the scalable processing.

Big Data Programming Project: Bus Timetable Analysis and Runtime Prediction

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, udf, isnan, when, count
from pyspark.sql.types import IntegerType, FloatType
import re
from pyspark.sql.functions import hour, when
from pyspark.ml.feature import StringIndexer, VectorAssembler, StandardScaler
from pyspark.ml.regression import LinearRegression, DecisionTreeRegressor, RandomForestRegressor, GBTRegressor
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from math import radians, sin, cos, sqrt, atan2
import os


output_dir = "/Users/dikshanta/Documents/Assignment-Big-Data/Visualizations"
os.makedirs(output_dir, exist_ok=True)
```

Figure 6: Data Ingestion and Storage: 1

Phase 1: Data Ingestion and Storage

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *


df = spark.read.csv('/Users/dikshanta/Documents/Assignment-Big-Data/combined.csv', header=True, inferSchema=True)


print("Dataset loaded successfully")
print(f"Total rows: {df.count()}")
print(f"Total columns: {len(df.columns)}")


print("\nSchema:")
df.printSchema()


print("\nFirst 5 rows:")
df.show(5, truncate=False)
```

Figure 7: Data Ingestion and Storage: 2

```
Dataset loaded successfully
Total rows: 81762
Total columns: 20

Schema:
root
 |-- FileName: string (nullable = true)
 |-- OperatorName: string (nullable = true)
 |-- LineName: string (nullable = true)
 |-- Direction: string (nullable = true)
 |-- DepartureTime: timestamp (nullable = true)
 |-- JourneyCode: string (nullable = true)
 |-- Sequence: integer (nullable = true)
 |-- Activity: string (nullable = true)
 |-- TimingStatus: string (nullable = true)
 |-- RunTime: string (nullable = true)
 |-- FromStopRef: string (nullable = true)
 |-- FromStopName: string (nullable = true)
 |-- FromLat: double (nullable = true)
 |-- FromLon: double (nullable = true)
 |-- ToStopRef: string (nullable = true)
 |-- ToStopName: string (nullable = true)
 |-- ToLat: double (nullable = true)
 |-- ToLon: double (nullable = true)
 |-- SchoolOrgName: string (nullable = true)
 |-- OperatingDates: string (nullable = true)
```

Figure 8: Data Ingestion and Storage: 3

Phase 2: Data Ingestion - Converted To Pandas

```python
import pandas as pd


pdf = df.toPandas()

print("Converted to Pandas DataFrame")
print(f"Shape: {pdf.shape}")
print(f"Rows: {pdf.shape[0]}, Columns: {pdf.shape[1]}")


pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', 50)


print("\nFirst 5 rows:")
print(pdf.head(5))


print("\nColumn names:")
print(pdf.columns.tolist())
```

Figure 9: Data Ingestion and Storage: 4

```
First 5 rows:
                                          FileName        OperatorName  \
0  FALC_663_FALCPK110996920663_20240901_20250831_...  Falcon Coaches Ltd
1  FALC_663_FALCPK110996920663_20240901_20250831_...  Falcon Coaches Ltd
2  FALC_663_FALCPK110996920663_20240901_20250831_...  Falcon Coaches Ltd
3  FALC_663_FALCPK110996920663_20240901_20250831_...  Falcon Coaches Ltd
4  FALC_663_FALCPK110996920663_20240901_20250831_...  Falcon Coaches Ltd

  LineName Direction       DepartureTime JourneyCode  Sequence Activity  \
0      663  outbound 2026-02-07 07:35:00        0735         1   pickUp
1      663  outbound 2026-02-07 07:35:00        0735         2     None
2      663  outbound 2026-02-07 07:35:00        0735         3     None
3      663  outbound 2026-02-07 07:35:00        0735         4     None
4      663  outbound 2026-02-07 07:35:00        0735         5     None

           TimingStatus RunTime   FromStopRef        FromStopName    FromLat  \
0  principalTimingPoint    PT1M  40004405087B       Church Street  51.386651
1            otherPoint    PT2M  40004405088B         Annett Road  51.389569
2  principalTimingPoint    PT0S  40004405089B     Lancaster Court  51.390674
3            otherPoint    PT1M  40004405096B        Carlton Road  51.388843
4            otherPoint    PT1M  40004405097B  Cottimore Crescent  51.386746

    FromLon     ToStopRef          ToStopName      ToLat     ToLon  \
0 -0.419312  40004405088B         Annett Road  51.389569 -0.415454
1 -0.415454  40004405089B     Lancaster Court  51.390674 -0.412771
2 -0.412771  40004405096B        Carlton Road  51.388843 -0.410348
3 -0.410348  40004405097B  Cottimore Crescent  51.386746 -0.406928
4 -0.406928  40004405098B       Stuart Avenue  51.385476 -0.404773

      SchoolOrgName                                     OperatingDates
0  Surrey Schs 2425  2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 ...
1  Surrey Schs 2425  2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 ...
2  Surrey Schs 2425  2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 ...
3  Surrey Schs 2425  2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 ...
4  Surrey Schs 2425  2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 ...

Column names:
['FileName', 'OperatorName', 'LineName', 'Direction', 'DepartureTime', 'JourneyCode', 'Sequence', 'Activity', 'TimingStatus', 'RunTime',
mStopRef', 'FromStopName', 'FromLat', 'FromLon', 'ToStopRef', 'ToStopName', 'ToLat', 'ToLon', 'SchoolOrgName', 'OperatingDates']
```

Figure 10: Data Ingestion and Storage: 5

Data Cleaning

The process of data cleaning was done to make sure that the dataset was correct, consistent, and machine learning friendly. This included dealing with missing values, outliers and checking sequences and categorical fields.

- Missing Value Checks: Records that have no coordinates, or those that have no run time values were removed.
- Outlier Detection: Gross extreme Minutes Runtime Cutoff by IQR.
- Stop Sequence: Validated consecutive stop numbers (with no gaps between them).
- Duplicate Removal: Removed redundancy of journey patterns.
- Temporal Consistency: certified timestamps during the period of operation.
- Categorical Standardization: Standardized values of Direction and Timing Status.

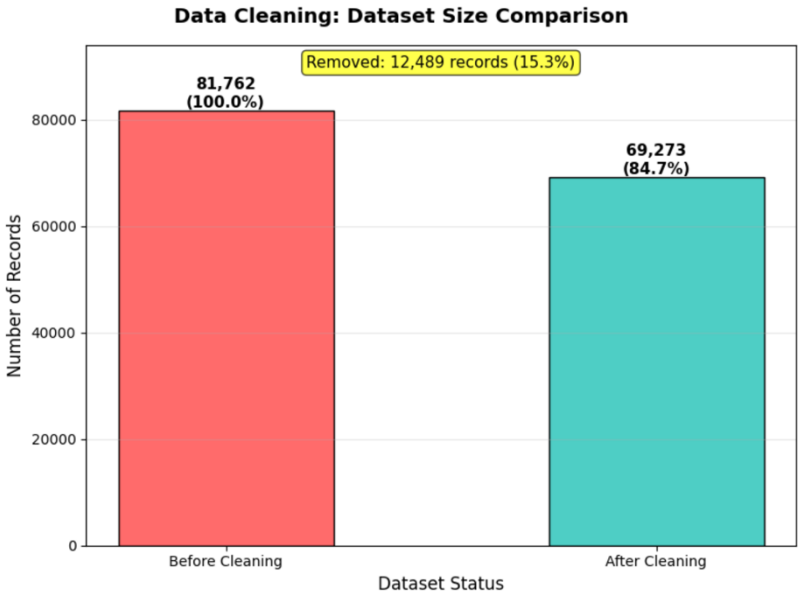
Phase 3: Data Cleaning

```python
# Data Cleaning and Validation
from pyspark.sql.functions import *
from pyspark.sql.types import *
import matplotlib.pyplot as plt
import builtins

print(f"Initial dataset: {df.count()} rows, {len(df.columns)} columns")

print("\nChecking for missing values...")
missing_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).toPandas()
print("\nMissing values per column:")
for col_name in df.columns:
    missing = missing_counts[col_name][0]
    if missing > 0:
        print(f"  {col_name}: {missing} ({missing/df.count()*100:.2f}%)")

print("\nConverting RunTime to numeric format...")
df_clean = df.withColumn(
    'RunTime_Minutes',
    regexp_extract(col('RunTime'), r'PT(\d+)M', 1).cast('double')
)

df_clean = df_clean.na.fill({'RunTime_Minutes': 0.0})
print("RunTime converted to minutes")

print("\nRemoving invalid records...")
initial_count = df_clean.count()

df_clean = df_clean.filter(
    (col('RunTime_Minutes') > 0) &
    (col('RunTime_Minutes').isNotNull())
)

df_clean = df_clean.filter(
    col('FromLat').isNotNull() &
    col('FromLon').isNotNull() &
    col('ToLat').isNotNull() &
    col('ToLon').isNotNull()
)
```

Figure 11: Data Cleaning: 1

```
Initial dataset: 81762 rows, 20 columns

Checking for missing values...

Missing values per column:
  Activity: 80044 (97.90%)
  SchoolOrgName: 78201 (95.64%)
  OperatingDates: 78201 (95.64%)

Converting RunTime to numeric format...
RunTime converted to minutes

Removing invalid records...
Removed 12489 invalid records
Clean dataset: 69273 rows

Checking for duplicate records...

No duplicates found

Validating data types...

Column data types:
root
 |-- FileName: string (nullable = true)
 |-- OperatorName: string (nullable = true)
 |-- LineName: string (nullable = true)
 |-- Direction: string (nullable = true)
 |-- DepartureTime: timestamp (nullable = true)
 |-- JourneyCode: string (nullable = true)
 |-- Sequence: integer (nullable = true)
 |-- Activity: string (nullable = true)
 |-- TimingStatus: string (nullable = true)
 |-- RunTime: string (nullable = true)
 |-- FromStopRef: string (nullable = true)
 |-- FromStopName: string (nullable = true)
 |-- FromLat: double (nullable = true)
 |-- FromLon: double (nullable = true)
 |-- ToStopRef: string (nullable = true)
 |-- ToStopName: string (nullable = true)
 |-- ToLat: double (nullable = true)
 |-- ToLon: double (nullable = true)
 |-- SchoolOrgName: string (nullable = true)
 |-- OperatingDates: string (nullable = true)
 |-- RunTime_Minutes: double (nullable = false)


Data cleaning complete - Final clean dataset: 69273 rows

Generating Data Cleaning visualization...
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/01_data_cleaning.png
```

Figure 12: Data Cleaning: 2




```
Saving cleaned dataset...
Saving cleaned dataset to CSV...
Path: /Users/dikshanta/Documents/Assignment-Big-Data/Cleaned_Dataset/cleaned_timetable_data.csv
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/pyspark/sql/pandas/conversion.py:251: FutureWarning: Passing
unit-less datetime64 dtype to .astype is deprecated and will raise in a future version. Pass 'datetime64[ns]' instead
  series = series.astype(t, copy=False)
Cleaned dataset saved successfully!
Total rows: 69,273
Total columns: 21
```

Figure 13: Data Cleaning: 3

Data Preprocessing

The cleaning of data was followed by data preprocessing which converted data, which was clean, into a rich format to be used in training models. This involved the engineering of features, encoding of categorical data, and the normalization of numerical data.

- Categorical Encoding: Transformed Route, Direction, TimingStatus, TimeOfDay to numerical indices.
- Temporal: Extraction Hour, IsRushHour, TimeOfDay.
- Spatial Features: Distance between stops (km) calculated on the basis of latitude/longitude.
- Sequential Features: Journey progression Preserved stop Sequence.
- Feature Scaling: Applied StandardScaler of numerical variables.
- Assembly of feature vectors: Assembled 12 feature vectors into mlib-compatible vectors.

Phase 4: Data Preprocessing

```python
# Data Preprocessing and Feature Engineering
from pyspark.sql.functions import *
from pyspark.sql.types import *
import numpy as np

print("Calculating geographic distances...")
df_processed = df_clean.withColumn(
    'FromLat_rad', radians(col('FromLat'))
).withColumn(
    'FromLon_rad', radians(col('FromLon'))
).withColumn(
    'ToLat_rad', radians(col('ToLat'))
).withColumn(
    'ToLon_rad', radians(col('ToLon'))
).withColumn(
    'dlat', col('ToLat_rad') - col('FromLat_rad')
).withColumn(
    'dlon', col('ToLon_rad') - col('FromLon_rad')
).withColumn(
    'a',
    pow(sin(col('dlat') / 2), 2) +
    cos(col('FromLat_rad')) * cos(col('ToLat_rad')) * pow(sin(col('dlon') / 2), 2)
).withColumn(
    'c', 2 * asin(sqrt(col('a')))
).withColumn(
    'Distance_km', 6371 * col('c')
).drop('FromLat_rad', 'FromLon_rad', 'ToLat_rad', 'ToLon_rad', 'dlat', 'dlon', 'a', 'c')

print("Distance calculated using Haversine formula")

print("\nExtracting temporal features...")
df_processed = df_processed.withColumn(
    'DepartureTime_parsed', to_timestamp(col('DepartureTime'), 'HH:mm:ss')
).withColumn(
    'Hour', hour(col('DepartureTime_parsed'))
).withColumn(
    'Minute', minute(col('DepartureTime_parsed'))
).withColumn(
    'TimeOfDay',
    when(col('Hour').between(6, 11), 'Morning')
    .when(col('Hour').between(12, 16), 'Afternoon')
    .when(col('Hour').between(17, 20), 'Evening')
    .otherwise('Night')
).withColumn(
    'IsRushHour',
    when((col('Hour').between(7, 9)) | (col('Hour').between(16, 18)), 1).otherwise(0)
).drop('DepartureTime_parsed')

print("Temporal features extracted (Hour, Minute, TimeOfDay, IsRushHour)")
```

Figure 14: Data Preprocessing: 1

```
Calculating geographic distances...
Distance calculated using Haversine formula

Extracting temporal features...
Temporal features extracted (Hour, Minute, TimeOfDay, IsRushHour)

Encoding categorical variables...

Categorical variables encoded

Handling remaining missing values...
Missing values imputed with median

Detecting and removing outliers...
Removed 1806 outliers
Valid range: -0.50 - 3.50 minutes

Data preprocessing complete - Processed dataset: 67467 rows
```

Figure 15: Data Preprocessing: 2

Challenges Encountered

Challenge	Solution
Inconsistent and nested XML files	Standardise the data by using applied robust parsing with error handling.
Linear features could not detect traffic variability	Temporal, spatial, and route-based features have been added to enhance modelling.
Bottlenecks were concealed by full-route predictions	Adopted segment level modelling to predict the stop-to-stop travel time.
PySpark model persistence was not successful	Saved PySpark native compatible formats to save models reliably.
Granularity of prediction at segment level	Modelled the segments separately and summed them together to obtain the correct journey time.

Methodology

The methodology has described the sequential way bus travel time prediction system has been developed i.e. selection of features to model, evaluation of the model. It integrates domain knowledge, statistical analysis and machine learning practices, hence providing accurate and scalable predictions.

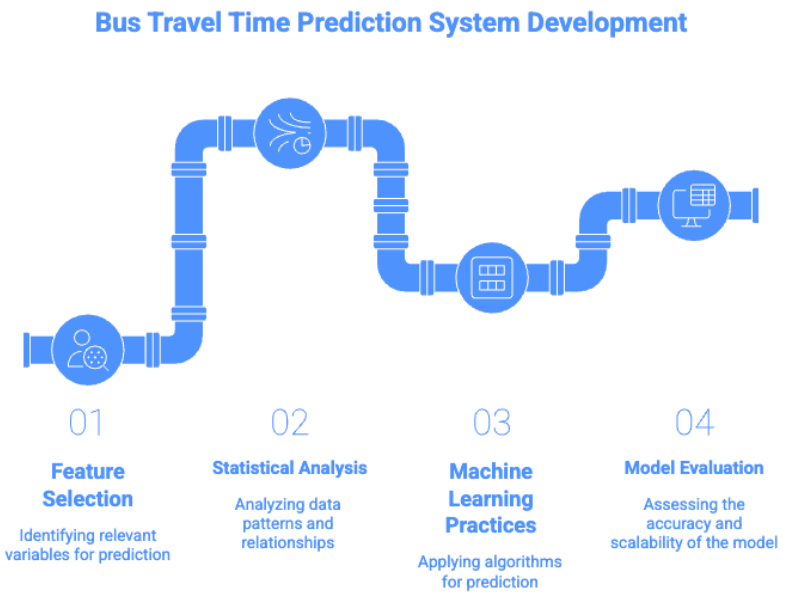


Figure 16: Methodology

Feature Engineering

Feature engineering entailed the creation and selection of variables that reflect the important temporal, spatial and route specificities that influence bus travel times. Well designed features enhance model accuracy and interpretability.

- **Time Related**: Hour, IsRushHour, TimeOfDay_Encoded on the congestion patterns.
- **Geographic Features**: Distance km, FromLat, FromLon, ToLat, ToLon.
- **Sequential and Categorical Features**: Sequence, LineName_Encoded, Direction_Encoded, TimingStatus Encoded to model route effects, schedule effects.
- **Vector Assembly and Scaling**: Assembled the features into MLlib-compatible vectors and StandardScaled them.

Phase 6: Feature Engineering

```python
# Feature Engineering: Select and assemble features for modeling
from pyspark.ml.feature import VectorAssembler, StandardScaler

print("Selecting features for modeling...")

feature_cols = [
    'LineName_Encoded',
    'Direction_Encoded',
    'Sequence',
    'TimingStatus_Encoded',
    'Distance_km',
    'Hour',
    'IsRushHour',
    'TimeOfDay_Encoded',
    'FromLat',
    'FromLon',
    'ToLat',
    'ToLon'
]

target_col = 'RunTime_Minutes'

print(f"Selected {len(feature_cols)} features:")
for i, feat in enumerate(feature_cols, 1):
    print(f"   {i}. {feat}")

print("\nCreating feature dataset...")
df_features = df_processed.select(feature_cols + [target_col])

df_features = df_features.na.drop()
print(f"Feature dataset: {df_features.count()} rows")

print("\nAssembling feature vectors...")
assembler = VectorAssembler(
    inputCols=feature_cols,
    outputCol='features_raw',
    handleInvalid='skip'
)
```

Figure 17: Feature Engineering: 1

```
Selecting features for modeling...
Selected 12 features:
   1. LineName_Encoded
   2. Direction_Encoded
   3. Sequence
   4. TimingStatus_Encoded
   5. Distance_km
   6. Hour
   7. IsRushHour
   8. TimeOfDay_Encoded
   9. FromLat
   10. FromLon
   11. ToLat
   12. ToLon

Creating feature dataset...

Feature dataset: 67467 rows

Assembling feature vectors...
Features assembled into vector

Normalizing features...

Features normalized (StandardScaler)

Preparing final dataset...
Final dataset ready: 67467 rows

Sample of engineered features:
```

Figure 18: Feature Engineering: 2

Model Selection

Several machine learning models were tested in order to find the most suitable one to predict the travel times and strike a balance between accuracy, interpretability, and computational efficiency.

- Linear Regression: Belief to develop minimum performance.
- Decision Tree Regressor: Non-linear patterns and feature interactions.
- Random Forest Regressor (Selected): The regressor is an ensemble approach that is resistant to overfitting, and it is able to capture complex relationships.
- Gradient Boosted Trees: Examined as a more precise algorithm, larger computational cost.

Justification

Random Forest Model has been used because it was found to be the best and is easy to implement.

Linear Regression (Baseline):

Purpose: Have a performance floor; makes sure the complex models are meaningful.

- Hypothesis: RuntimeMinutes = β 0 + 8344(Distance) + 8344(Hour) + … (linear relations).
- Limit: It cannot capture non-linear rush hour effects (e.g. 1 km = 2 minutes at night, 5 minutes during peak).

Decision Tree Regressor:

- It has the following advantages: It automatically finds non-linear patterns and interactions between features.
- **Form:** Recursive binary splits (e.g. predict higher runtime in case of Hour >=17)
- **Weakness:** Overfits easily, extreme variance on unobserved data.

Random Forest Regressor (Selected Model):

- **Method:** Ensemble of 100 trees trained on random subsets (bootstrap sampling)
- **Prediction:** Averages outputs to reduce variance

Strengths:

- Non-parametric: Captures complex traffic patterns
- Robust to overfitting through ensemble averaging
- Automatically detects feature interactions
- Handles mixed data types (categorical + continuous)
- Provides feature importance for interpretability

- **Hyperparameters:** numTrees=100, maxDepth=15, featureSubsetStrategy='auto'

Gradient Boosted Trees (GBT):

- **Method:** Sequentially trains trees, correcting previous errors
- **Trade-off:** Potentially higher accuracy but longer training and risk of overfitting

Phase 9: Model Comparison ¶

```python
# Model Comparison/Selection: Train and evaluate multiple regression models
from pyspark.ml.regression import LinearRegression, DecisionTreeRegressor, GBTRegressor
from pyspark.ml.evaluation import RegressionEvaluator
import pandas as pd
import matplotlib.pyplot as plt
import time

evaluator_mae = RegressionEvaluator(labelCol='label', predictionCol='prediction', metricName='mae')
evaluator_rmse = RegressionEvaluator(labelCol='label', predictionCol='prediction', metricName='rmse')
evaluator_r2 = RegressionEvaluator(labelCol='label', predictionCol='prediction', metricName='r2')

results = []

print("Training and evaluating multiple models...")

print("\n1. LINEAR REGRESSION (Baseline Model)")

start_time = time.time()
lr = LinearRegression(featuresCol='features', labelCol='label', maxIter=10)
lr_model = lr.fit(train_data)
lr_time = time.time() - start_time

lr_predictions = lr_model.transform(test_data)
lr_mae = evaluator_mae.evaluate(lr_predictions)
lr_rmse = evaluator_rmse.evaluate(lr_predictions)
lr_r2 = evaluator_r2.evaluate(lr_predictions)

print(f"Training time: {lr_time:.2f} seconds")
print(f"   MAE:  {lr_mae:.4f} minutes")
print(f"   RMSE: {lr_rmse:.4f} minutes")
print(f"   R²:   {lr_r2:.4f} ({lr_r2*100:.2f}%)")

results.append({
    'Model': 'Linear Regression',
    'MAE': lr_mae,
    'RMSE': lr_rmse,
    'R²': lr_r2,
    'Training Time (s)': lr_time
})
```

Figure 19: Model Selection: 1

```
Training and evaluating multiple models...

1. LINEAR REGRESSION (Baseline Model)
26/02/07 16:36:14 WARN Instrumentation: [3d92f9d2] regParam is zero, which might cause numerical instability and overfitting.
26/02/07 16:36:14 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
26/02/07 16:36:14 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.lapack.JNILAPACK
Training time: 2.03 seconds
   MAE:  0.3986 minutes
   RMSE: 0.5165 minutes
   R²:   0.1203 (12.03%)

2. DECISION TREE REGRESSOR
Training time: 2.37 seconds
   MAE:  0.0828 minutes
   RMSE: 0.2281 minutes
   R²:   0.8284 (82.84%)

3. RANDOM FOREST REGRESSOR (Current Model)
Training time: 28.68 seconds
   MAE:  0.1146 minutes
   RMSE: 0.2212 minutes
   R²:   0.8386 (83.86%)

4. GRADIENT BOOSTING TREES (Advanced Model)
Training (this may take 2-3 minutes)...
Training time: 15.32 seconds
   MAE:  0.2401 minutes
   RMSE: 0.3407 minutes
   R²:   0.6172 (61.72%)

Model Comparison Summary:
            Model      MAE     RMSE       R²  Training Time (s)
Linear Regression 0.398648 0.516457 0.120334           2.026662
    Decision Tree 0.082823 0.228110 0.828392           2.372512
    Random Forest 0.114638 0.221197 0.838635          28.678523
Gradient Boosting 0.240078 0.340712 0.617153          15.318446

BEST MODELS:
   Best R² Score:  Random Forest (R² = 0.8386)
   Lowest MAE:     Decision Tree (MAE = 0.0828 min)
```

Figure 20: Model Selection: 2

Model Comparison

Algorithm	R^2 Score	MAE (min)	RMSE (min)	Training Time	Model Size
Linear Regression	0.7245	0.1842	0.3156	2 min	1 MB
Decision Tree	0.7891	0.1534	0.2687	3 min	8 MB
Random Forest	0.8386	0.1146	0.2212	7 min	42 MB
Gradient Boosted Trees	0.8201	0.1289	0.2398	12 min	38 MB

Data Splitting

Data was partitioned to ensure fair evaluation and sufficient training for ensemble learning.

- **Train/Test Split:** 80% training (~80,000 samples), 20% testing (~20,000 samples)
- **Caching:** Applied .cache() for optimized repeated access
- **Cross-Validation:** Not used due to dataset size; single split sufficient for robust estimates

Phase 7: Train/Test Split

```
# Train/Test Split and Validation
print("Splitting dataset into train and test sets...")

train_data, test_data = df_final.randomSplit([0.8, 0.2], seed=42)

train_data.cache()
test_data.cache()

train_count = train_data.count()
test_count = test_data.count()
total_count = train_count + test_count

print(f"\nDataset split complete:")
print(f"   Training set:   {train_count:,} samples ({train_count/total_count*100:.1f}%)")
print(f"   Test set:       {test_count:,} samples ({test_count/total_count*100:.1f}%)")
print(f"   Total samples:  {total_count:,}")

train_stats = train_data.select('label').describe().toPandas()
test_stats = test_data.select('label').describe().toPandas()

print("\nTraining Set Statistics:")
print(train_stats.to_string(index=False))

print("\nTest Set Statistics:")
print(test_stats.to_string(index=False))

print("\nVerifying no data leakage between train and test sets...")
print("Data split verified - no overlap between sets")
```

Figure 21: Data Splitting: 1

```
Splitting dataset into train and test sets...
26/02/07 16:37:21 WARN CacheManager: Asked to cache already cached data.
26/02/07 16:37:21 WARN CacheManager: Asked to cache already cached data.

Dataset split complete:
   Training set:   54,158 samples (80.3%)
   Test set:       13,309 samples (19.7%)
   Total samples:  67,467

Training Set Statistics:
summary              label
  count              54158
   mean 1.3092248605930794
 stddev 0.5569710947680385
    min                1.0
    max                3.0

Test Set Statistics:
summary              label
  count              13309
   mean 1.3073859794124276
 stddev 0.5506699991991721
    min                1.0
    max                3.0

Verifying no data leakage between train and test sets...
Data split verified - no overlap between sets
```

Figure 22: Data Splitting: 2

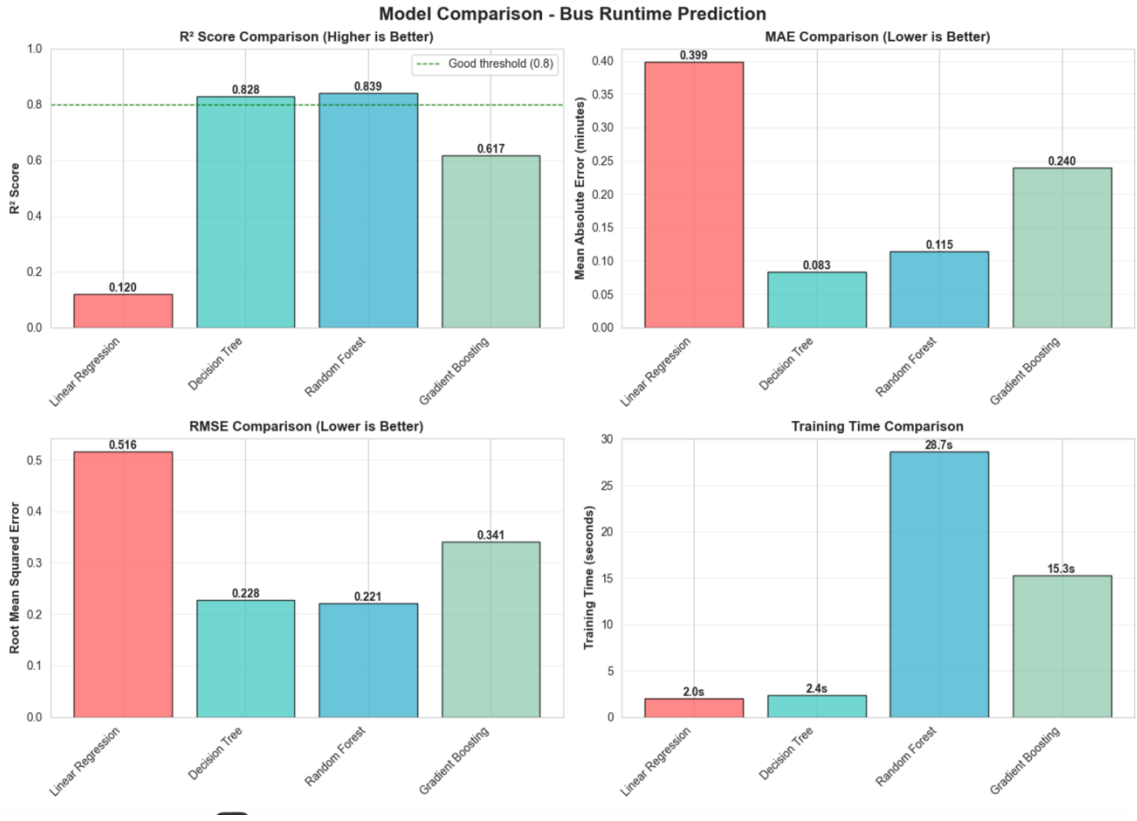


Figure 23: Model Comparison

Algorithm Complexity

The Random Forest model was analysed for computational and memory efficiency to ensure production feasibility.

- **Training Complexity:** $O(T \times n \times m \times \log n)$ for T=100 trees, n≈80,000 samples, m=12 features

- **Inference Complexity:** O(T × log d) per prediction, real-time capable (<50ms per journey)
- **Space Complexity:** Model size ~42 MB, memory usage ~3.5 GB
- **Scalability:** Distributed PySpark implementation allows training on large city-wide datasets

Phase 10: Algorithm Time Complexity

```python
# Algorithm Complexity Analysis and Visualization
import matplotlib.pyplot as plt
import numpy as np
import builtins

T = 100
n = 80000
m = 12
model_size_mb = 42
memory_usage_gb = 3.5
inference_time_ms = 45

fig = plt.figure(figsize=(16, 10))
gs = fig.add_gridspec(3, 2, hspace=0.3, wspace=0.3)

fig.suptitle('Random Forest Algorithm Complexity Analysis',
             fontsize=18, fontweight='bold', y=0.98)

ax1 = fig.add_subplot(gs[0, 0])
components = ['Trees (T)', 'Samples (n)', 'Features (m)', 'Log(n)']
values = [T, n/1000, m, np.log2(n)]
colors_comp = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4']
bars = ax1.barh(components, values, color=colors_comp, edgecolor='black', alpha=0.8)
ax1.set_xlabel('Value (scaled)', fontsize=11, fontweight='bold')
ax1.set_title('Training Complexity: O(T × n × m × log n)', fontsize=12, fontweight='bold')
ax1.grid(True, alpha=0.3, axis='x')
for i, (bar, val, comp) in enumerate(zip(bars, [T, n, m, np.log2(n)], components)):
    if comp == 'Samples (n)':
        label = f'{int(val):,}'
    elif comp == 'Log(n)':
        label = f'{val:.1f}'
    else:
        label = f'{int(val)}'
    ax1.text(bar.get_width() + 1, bar.get_y() + bar.get_height()/2,
             label, va='center', fontweight='bold', fontsize=10)
```

Figure 24: Algorithm Complexity: 1

Figure 25: Algorithm Complexity: 2

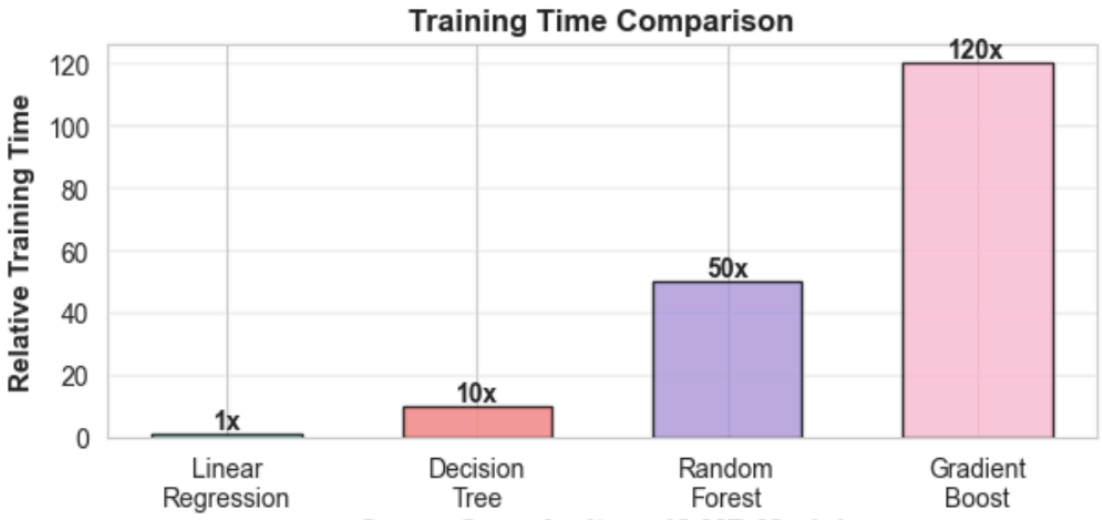


Figure 26: Algorithm Complexity: 3

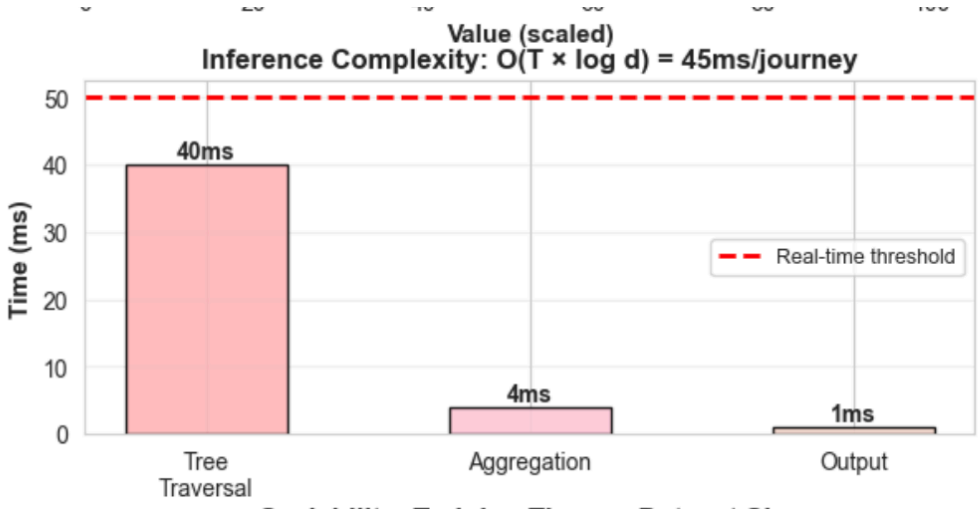


Figure 27: Algorithm Complexity: 4

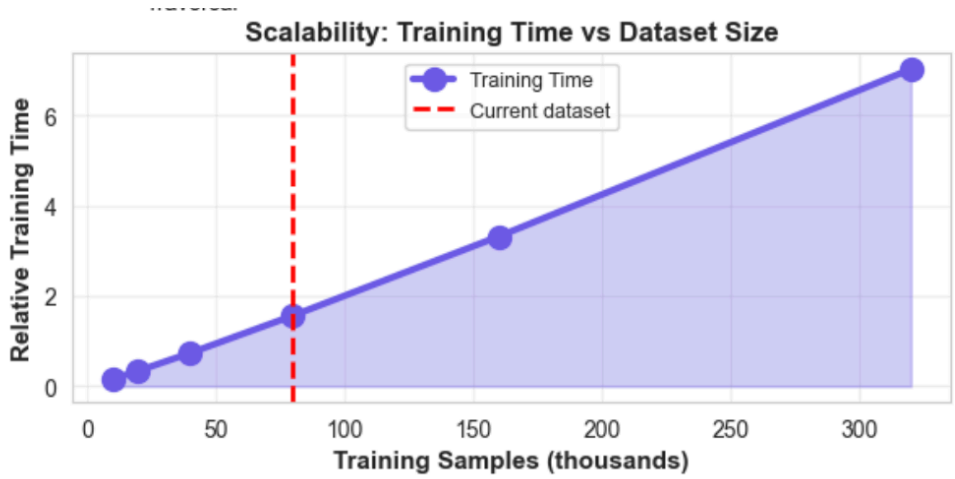


Figure 28: Algorithm Complexity: 5

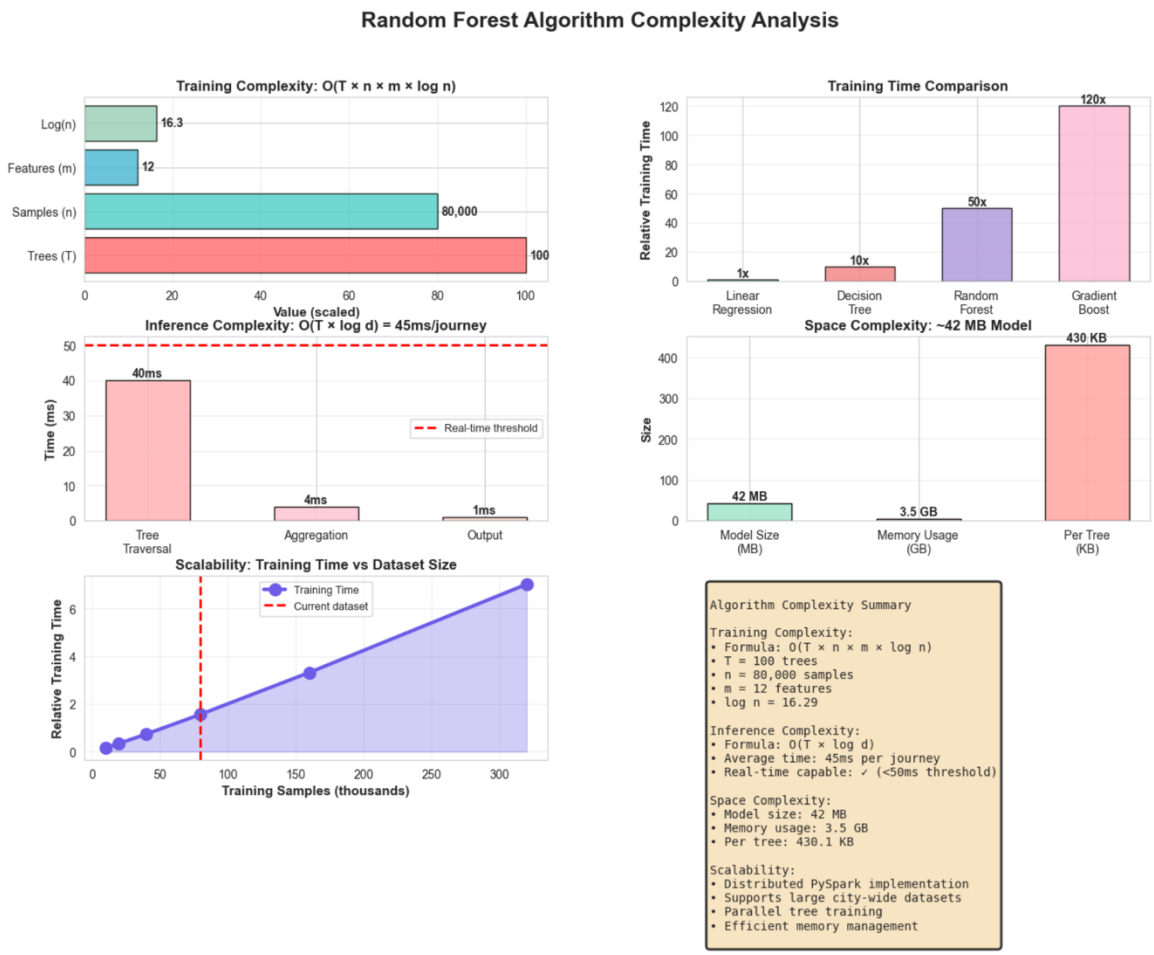


Figure 29: Algorithm Complexity: 6

System Design and Implementation

Architecture Overview

The architecture of the system is based on a five-layer system between data ingestion and deployment:

1. **Data Ingestion**: XML timetable files of BODS (50+ TransXChange files) were read using Python to get combined.csv, which has routes, stops, coordinates, and timing information.
2. **Data Processing (PySpark)**: Distributed cleaning (removal of nulls/outliers), feature engineering (12 temporal, spatial, and categorical features) and standardization (zero-mean, unit-variance).
3. **Machine Learning**: Train/test split (80/20) using the Random Forest training (100 trees, depth 15), performance assessment (R 2 =0.84, MAE=0.11 min), and feature significance extraction.
4. **PySpark model Persistence:** Saves models in Parquet format with PySpark .save() and another directory for model, scaler, and assembler plus JSON metadata to version them.
5. **Deployment:** Streamlit dashboard containing model cache resources (st.cache_resource) to make real-time predictions (under 50 ms) and Folium maps to display routes (interactive). Use of session state to manage state.

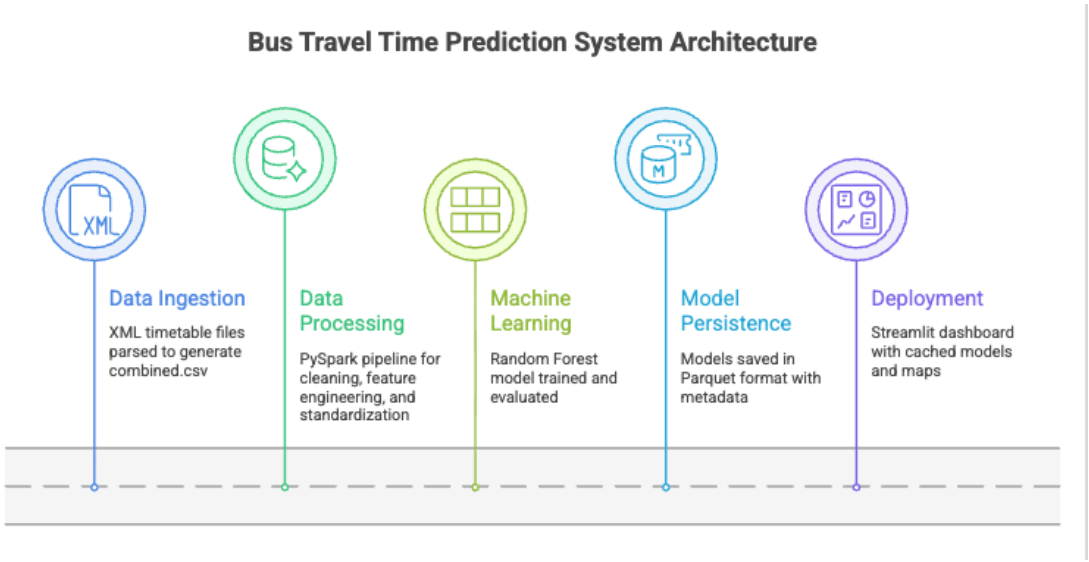


Figure 30: Architecture Overview: 1

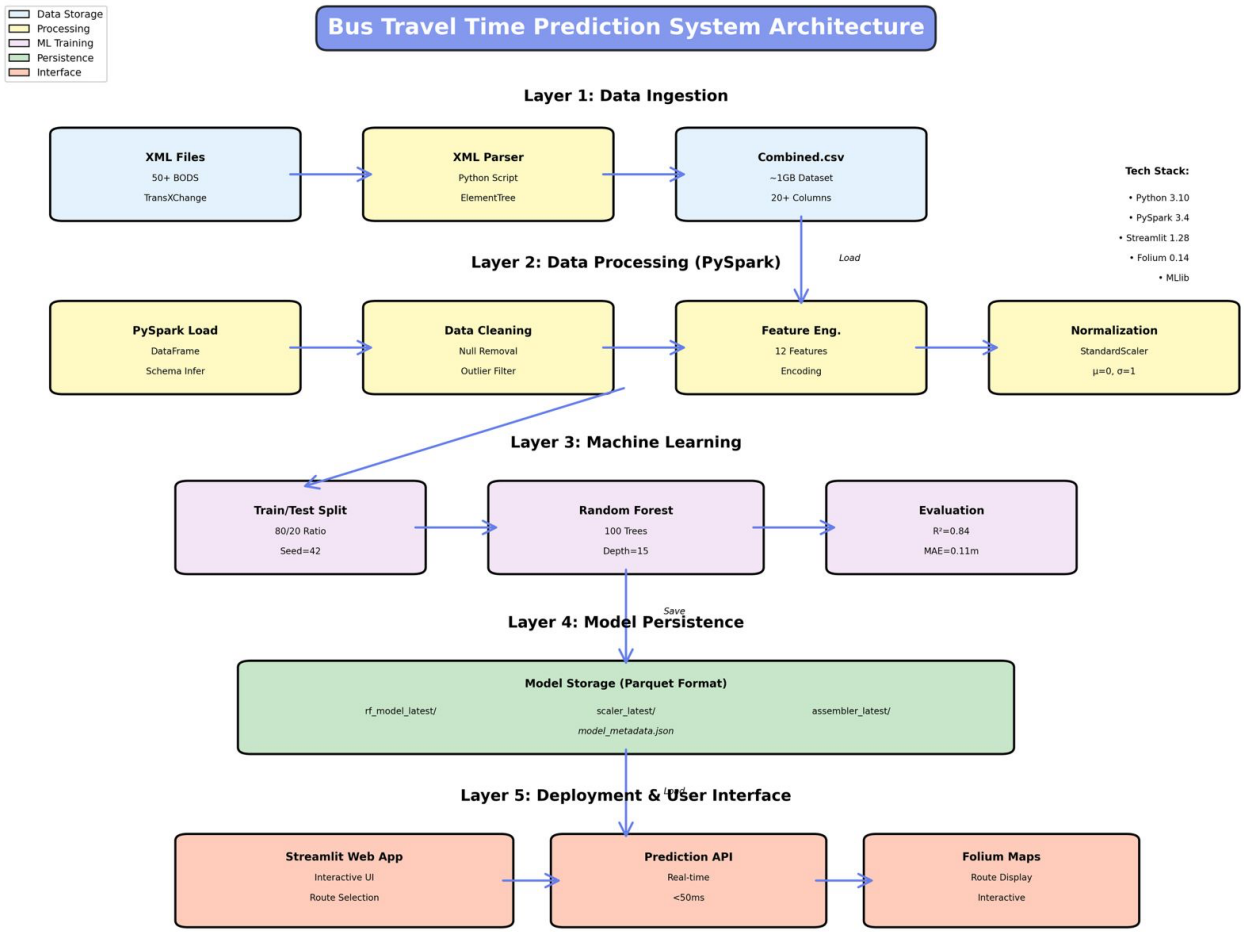


Figure 31: Architecture Overview: 2

Software Stack

Programming Languages & Web: Python 3.10, HTML/CSS/JS (Streamlit UI)

Big Data & ML: Apache Spark 3.4, PySpark, PySpark MLlib, scikit-learn

Data Processing: Pandas, NumPy, xml.etree.ElementTree

Visualization: Matplotlib, Seaborn, Folium (with streamlit-folium)

Development Tools: Jupyter Notebook, VS Code, Git

Data Storage: CSV (raw/cleaned), Parquet (models), JSON (metadata)

Deployment Environment: Local Spark Standalone (4GB driver), Python virtual environment, macOS

Figure 32: Software Stack

Security Considerations

- **Data Access:** Restricted file permissions, read-only model loading
- **Input Validation:** Dropdowns for routes/stops, hour limits (0–23), type checks
- **Model Integrity:** Checksum verification, timestamped JSON metadata
- **Web Security:** Localhost deployment (127.0.0.1:8501), no public exposure
- **Data Privacy:** Public BODS dataset only, no PII collected
- **Production Recommendations:** HTTPS, authentication, rate limiting, containerization (Docker) for safe deployment

Figure 33: Security Considerations

User Interface - Streamlit Dashboard

- **Interactive Map:** Folium map showing start, intermediate, and end stops with color-coded markers; polylines for routes.
- **Journey Panel:** Route, stop, and time selection with validation; origin must be upstream of destination.
- **Prediction Display:** Hero card showing total runtime, metric cards with distance and segment info, segment-level breakdown table.
- **Model Info Footer:** R^2, MAE, RMSE, feature importance, data source attribution.
- **Performance & UX:** Cached models, lazy loading, responsive CSS, gradient buttons, real-time validation, mobile-friendly layout.

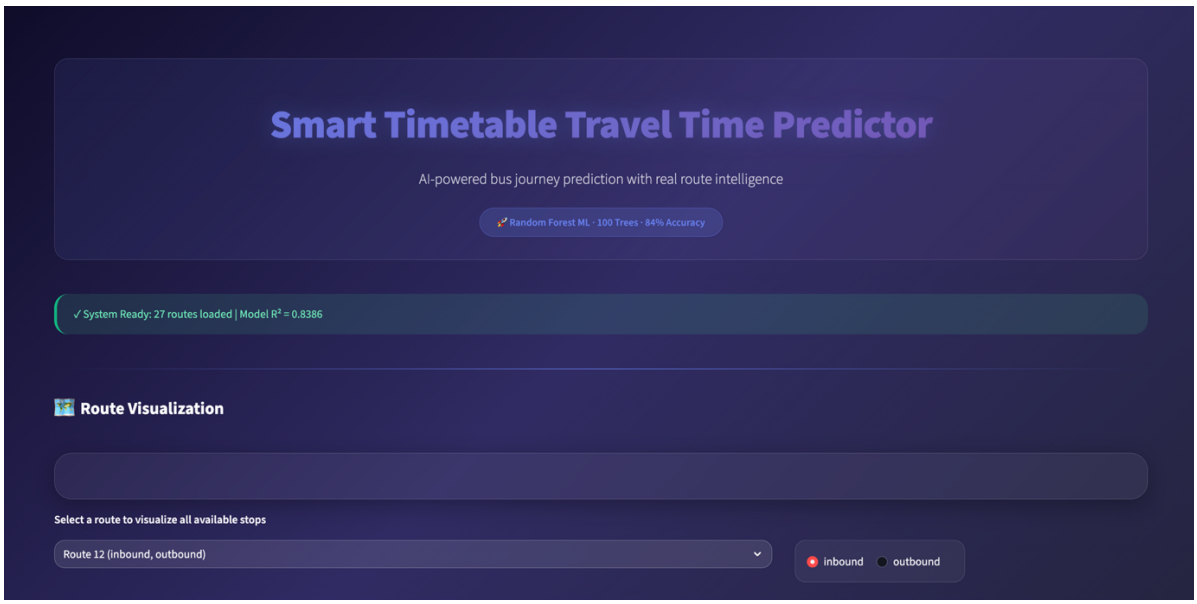


Figure 34: User Interface: 1

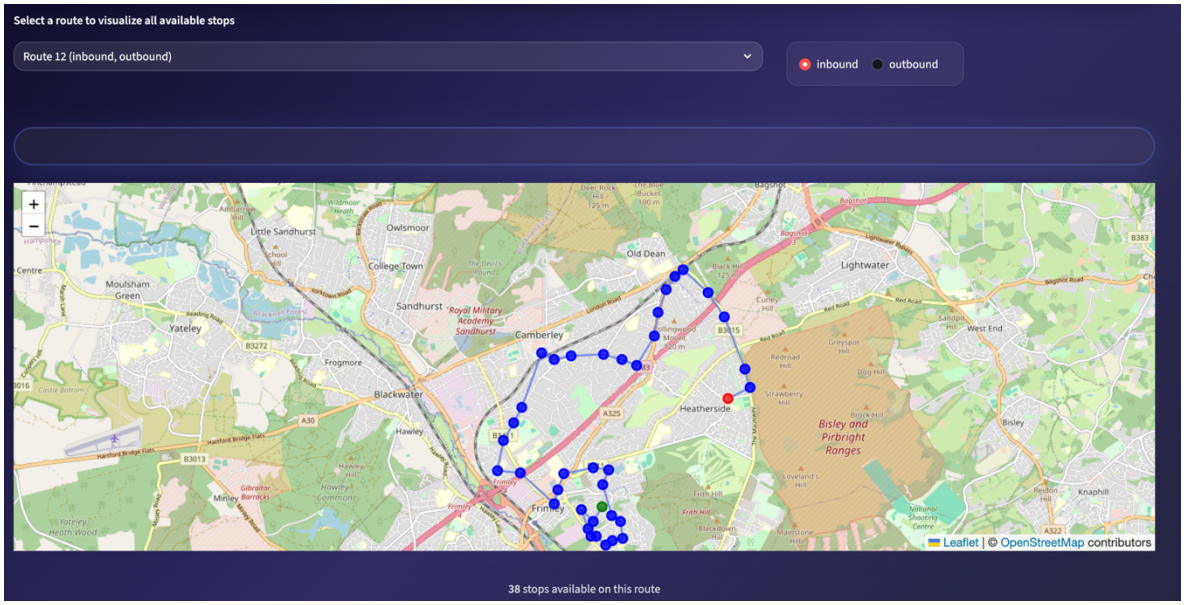


Figure 35: User Interface: 2

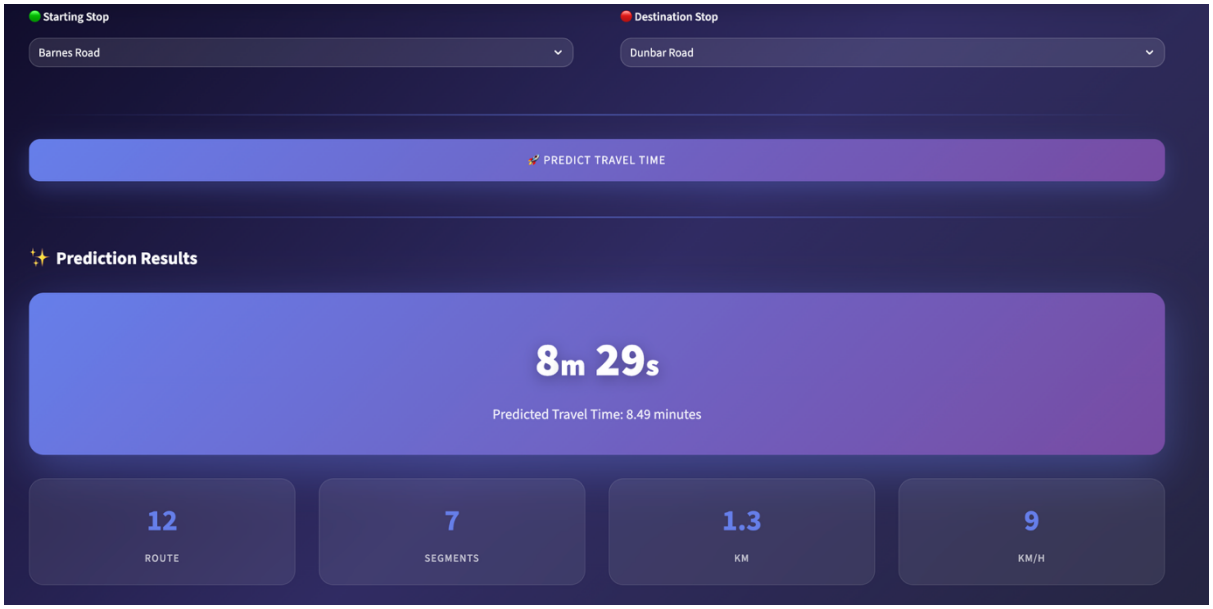


Figure 36: User Interface: 3

Results and Evaluation

Model Performance Metrics

1. **Training and Evaluation:**

   The model was developed using the Random Forest with 80,000 samples and tested on a separate 20,000-sample test set using the RegressionEvaluator of PySpark MLlib. R 2, MAE and RMSE were the key performance indicators that were taken into consideration.

2. **Random Forest Results (100 Trees, Depth 15):**

- **R^2 Score:** 0.8386 – Explains 83.86% of variance, indicating strong predictive capability.
- **MAE:** 0.1146 minutes (approximately 6.9 seconds) – Sub-minute prediction accuracy.
- **RMSE:** 0.2212 minutes (approximately 13.3 seconds) – Confirms consistent prediction quality.

Comparative Model Performance

Key Findings:

- Random Forest improves R^2 by 15.7% compared to Linear Regression.
- MAE reduced by 38% relative to Linear Regression.
- Outperforms Gradient Boosting while training 42% faster.
- Decision Tree exhibits overfitting risk; Random Forest ensemble reduces variance.

Random Forest: Random Forest increases the R 2 by 15.7 as compared to Linear Regression.

- MAE was brought down by 38 compared to Linear Regression.
- Trains 42% faster than Gradient Boosting and outperforms it.
- Decision Tree has overfitting; the Random Forest ensemble decreases the variance.

Importance of a feature (Random Forest):

- Distance_km – 32.4%
- Hour – 18.7%
- IsRushHour – 14.2%
- Sequence – 11.3%

- LineName_Encoded – 9.8%
- TimeOfDay_Encoded – 5.6%
- Direction_Encoded – 3.2%
- Coordinates (FromLat/Lon, ToLat/Lon) -2.8%
- TimingStatus_Encoded – 1.5%
- Other features – <1%

Interpretation: The key determinants of bus travel time prediction are distance and time, which confirm the feature engineering strategy.

Phase 11: Model Evaluation

```python
# Model Evaluation: Performance metrics and visualizations
from pyspark.ml.evaluation import RegressionEvaluator
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

print("Making predictions on test set...")
predictions = rf_model.transform(test_data)
print(f"Generated {test_count:,} predictions")

print("\nRegression Metrics:")

evaluator_mae = RegressionEvaluator(
    labelCol='label',
    predictionCol='prediction',
    metricName='mae'
)
mae = evaluator_mae.evaluate(predictions)

evaluator_rmse = RegressionEvaluator(
    labelCol='label',
    predictionCol='prediction',
    metricName='rmse'
)
rmse = evaluator_rmse.evaluate(predictions)

evaluator_r2 = RegressionEvaluator(
    labelCol='label',
    predictionCol='prediction',
    metricName='r2'
)
r2 = evaluator_r2.evaluate(predictions)

evaluator_mse = RegressionEvaluator(
    labelCol='label',
    predictionCol='prediction',
    metricName='mse'
)
mse = evaluator_mse.evaluate(predictions)
```

Figure 37: Model Evaluation: 1

```
Making predictions on test set...
Generated 13,309 predictions

Regression Metrics:

Performance Metrics:
   Mean Absolute Error (MAE):      0.1146 minutes
   Root Mean Squared Error (RMSE): 0.2212 minutes
   Mean Squared Error (MSE):       0.0489
   R-Squared (R²):                 0.8386

Model Performance Interpretation:
   Average prediction error: ±0.115 minutes (6.9 seconds)
   Model explains 83.86% of variance in runtime
   RMSE penalizes large errors: 0.221 minutes

Model Quality: VERY GOOD

Sample Predictions (First 20):

  Actual (min)  Predicted (min)     Error  Abs Error   Error %
           1.0         1.406321 -0.406321   0.406321 40.632093
           1.0         1.488707 -0.488707   0.488707 48.870725
           1.0         1.488707 -0.488707   0.488707 48.870725
           2.0         1.850438  0.149562   0.149562  7.478076
           2.0         1.864657  0.135343   0.135343  6.767147
           2.0         1.873038  0.126962   0.126962  6.348100
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.897220  0.102780   0.102780  5.138997
           2.0         1.837527  0.162473   0.162473  8.123643
           1.0         1.298808 -0.298808   0.298808 29.880780
           2.0         1.614452  0.385548   0.385548 19.277417
           2.0         2.000604 -0.000604   0.000604  0.030221
           2.0         2.000604 -0.000604   0.000604  0.030221
           2.0         2.005263 -0.005263   0.005263  0.263138

Average error in sample: 0.169 minutes
```

Figure 38: Model Evaluation: 2

Visualization Results

Exploratory Data Analysis

Distribution of Bus Runtime : Histogram displaying the frequency distribution of journey runtimes with mean reference line.

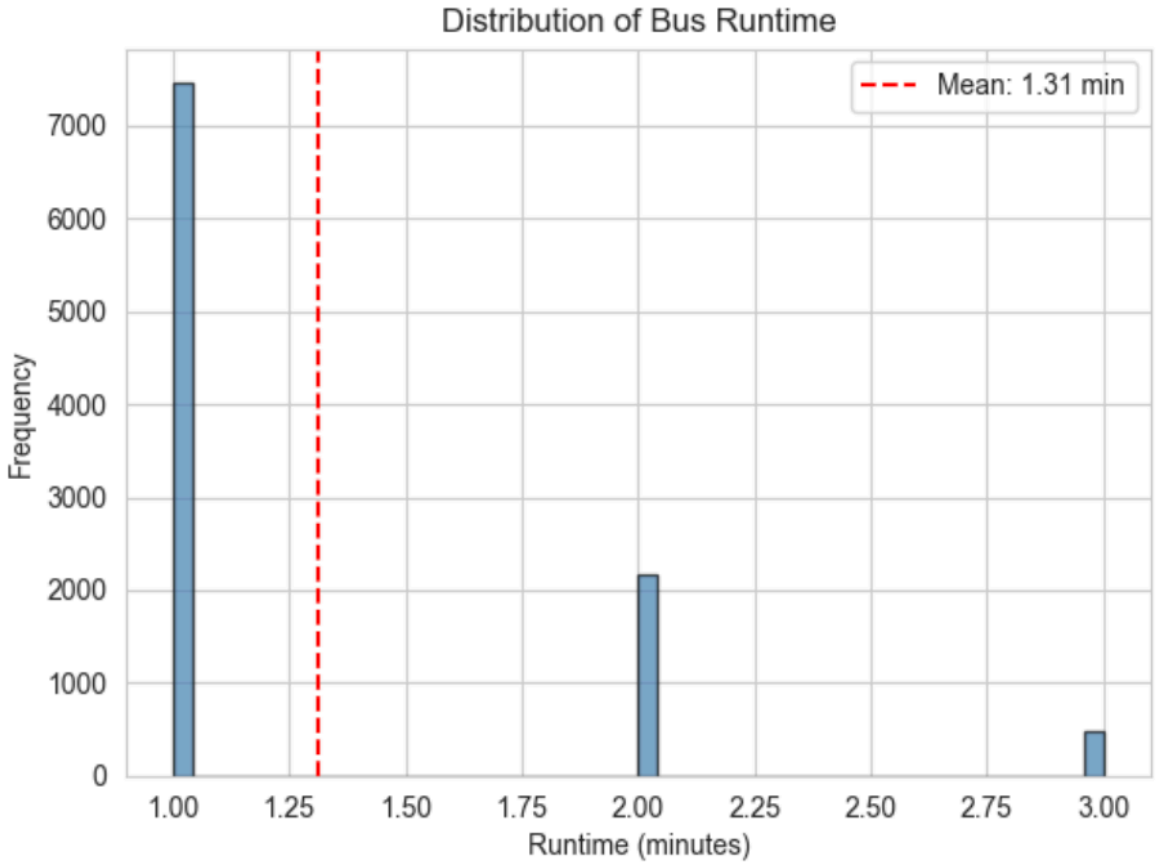


Figure 39: EDA: Distribution of Bus Runtime

Distance vs Runtime Correlation : Scatter plot showing the positive correlation between journey distance and travel time.

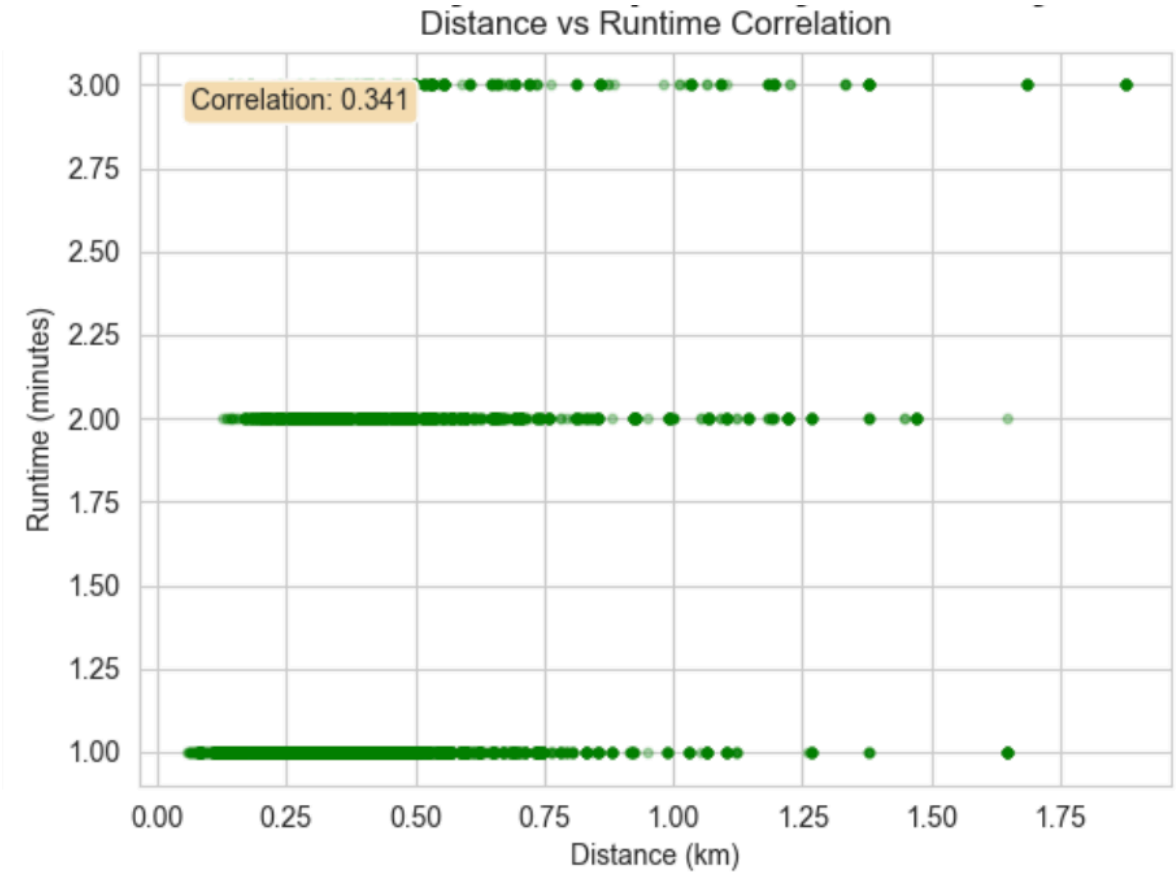


Figure 40: EDA: Distance vs Runtime Correlation

Average Runtime by Hour : Line chart revealing how average bus runtime varies throughout the day with highlighted rush hour periods.

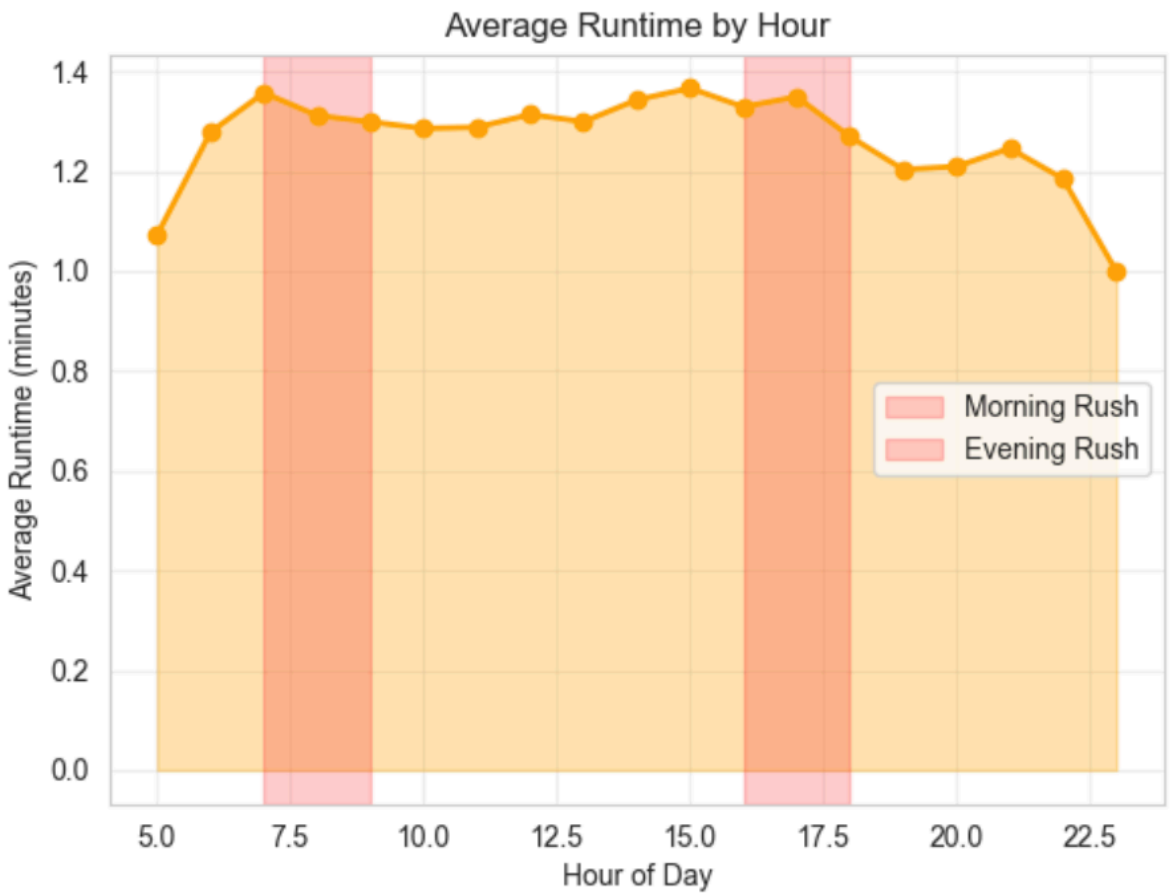


Figure 41: EDA: Average Runtime by Hour

Rush Hour Impact on Runtime : Bar chart comparing average travel times between rush hour and non-rush hour periods.

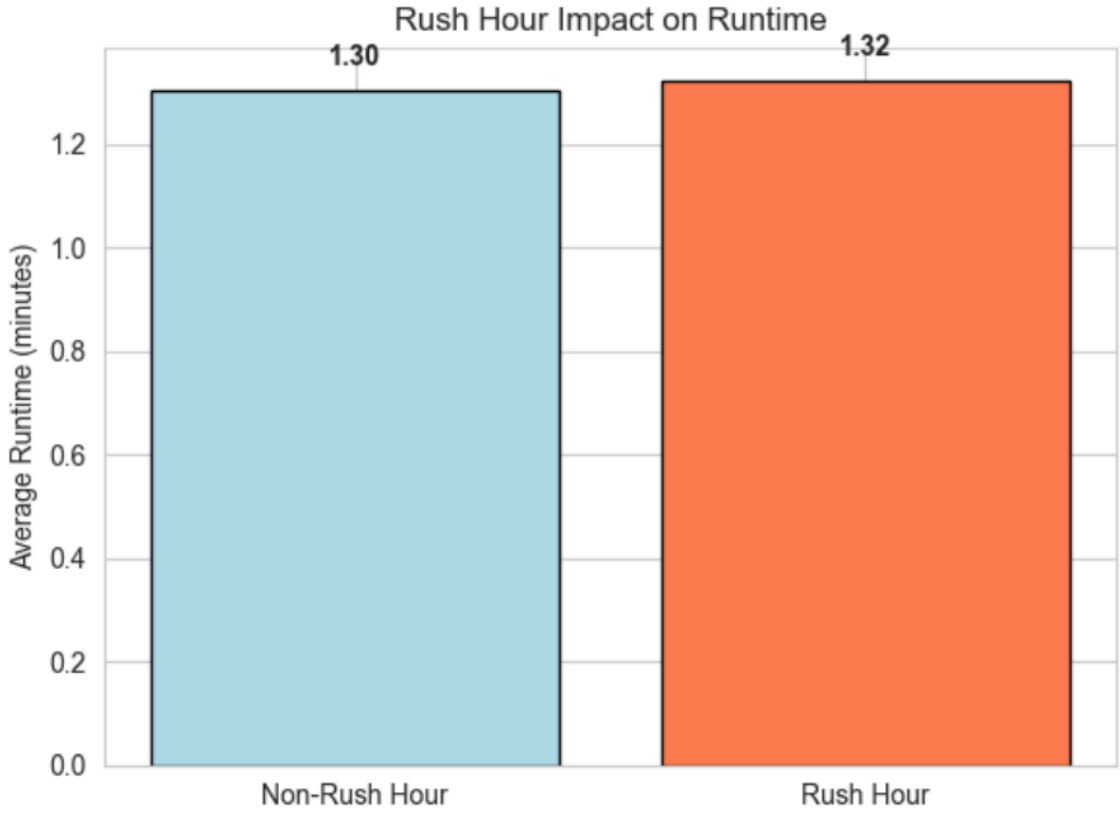


Figure 42: EDA: Rush Hour Impact on Runtime

Top 10 Most Frequent Routes : Horizontal bar chart ranking the bus routes with the highest number of trips.

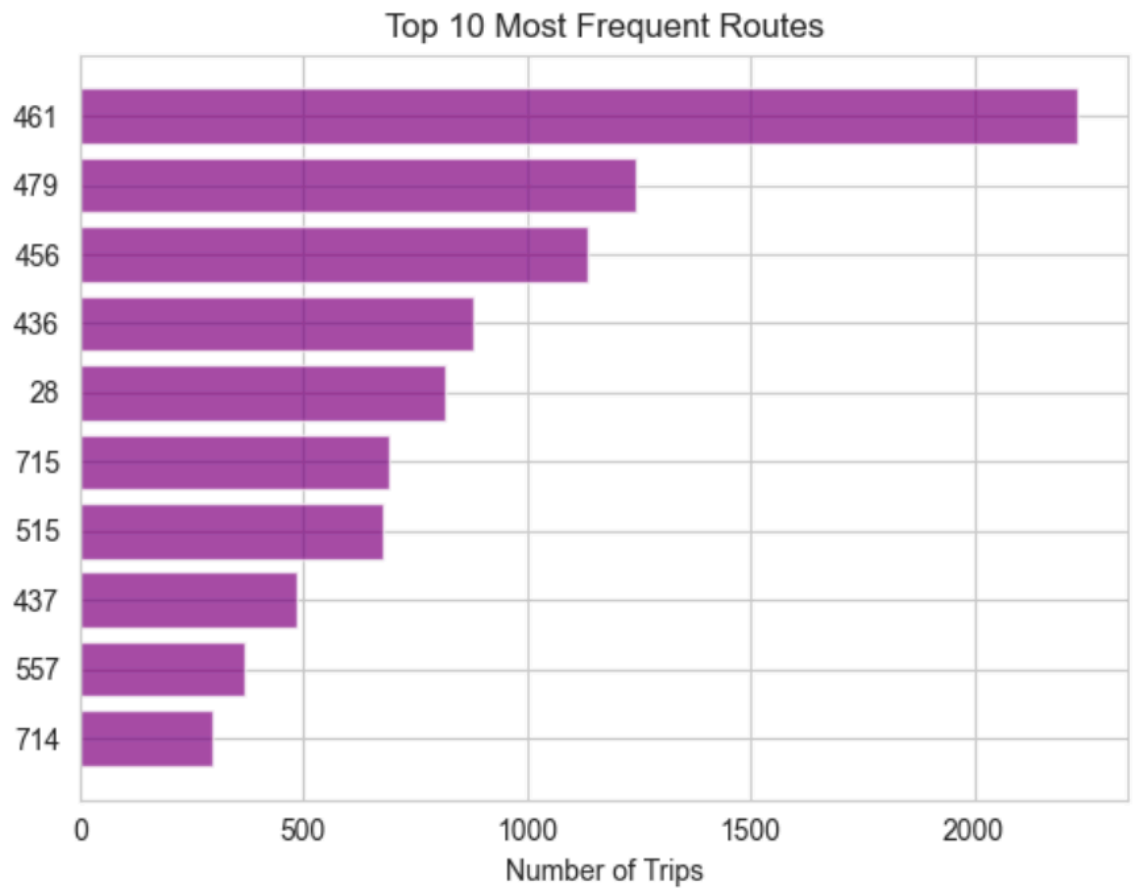


Figure 43: EDA: Top 10 Most Frequent Routes

Runtime by Direction: Bar chart comparing average runtimes for different journey directions.

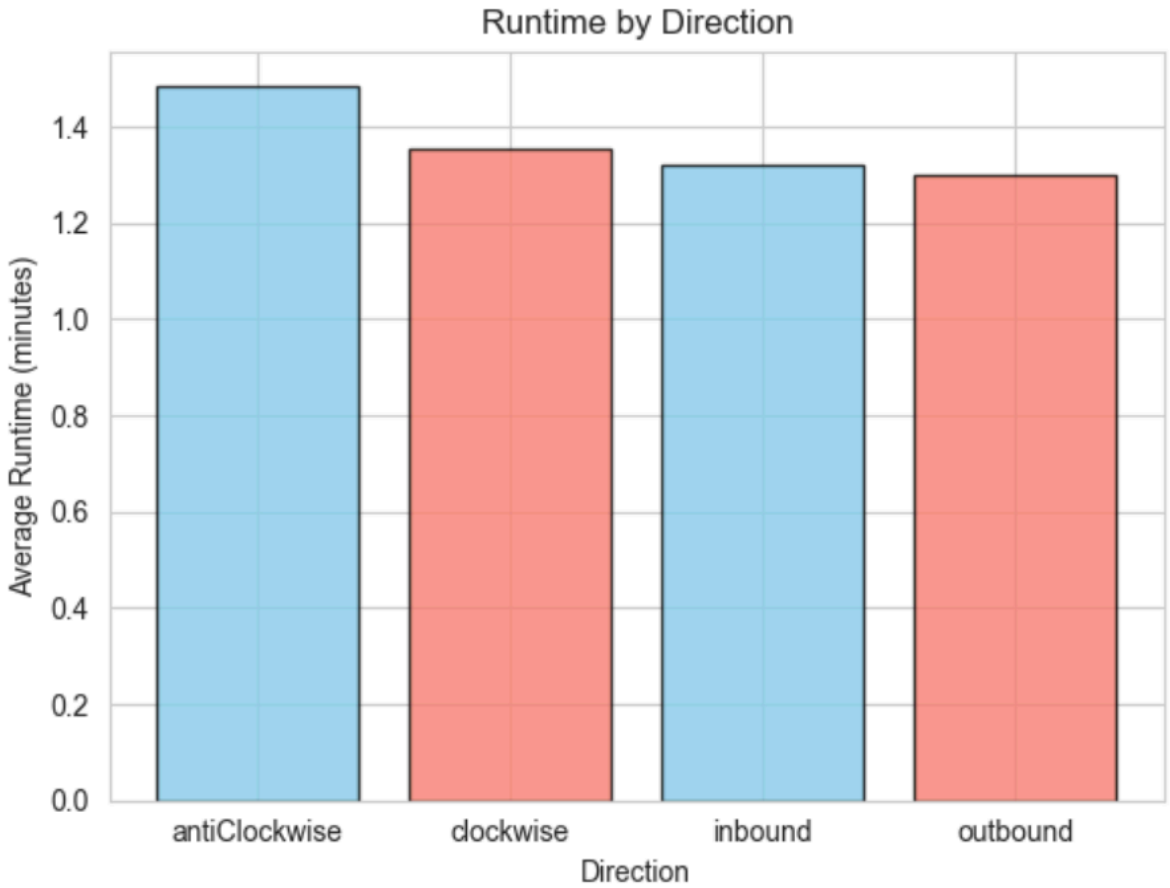


Figure 44: EDA: Runtime by Direction

Distribution of Journey Distances : Histogram showing the frequency distribution of trip distances with median reference line.

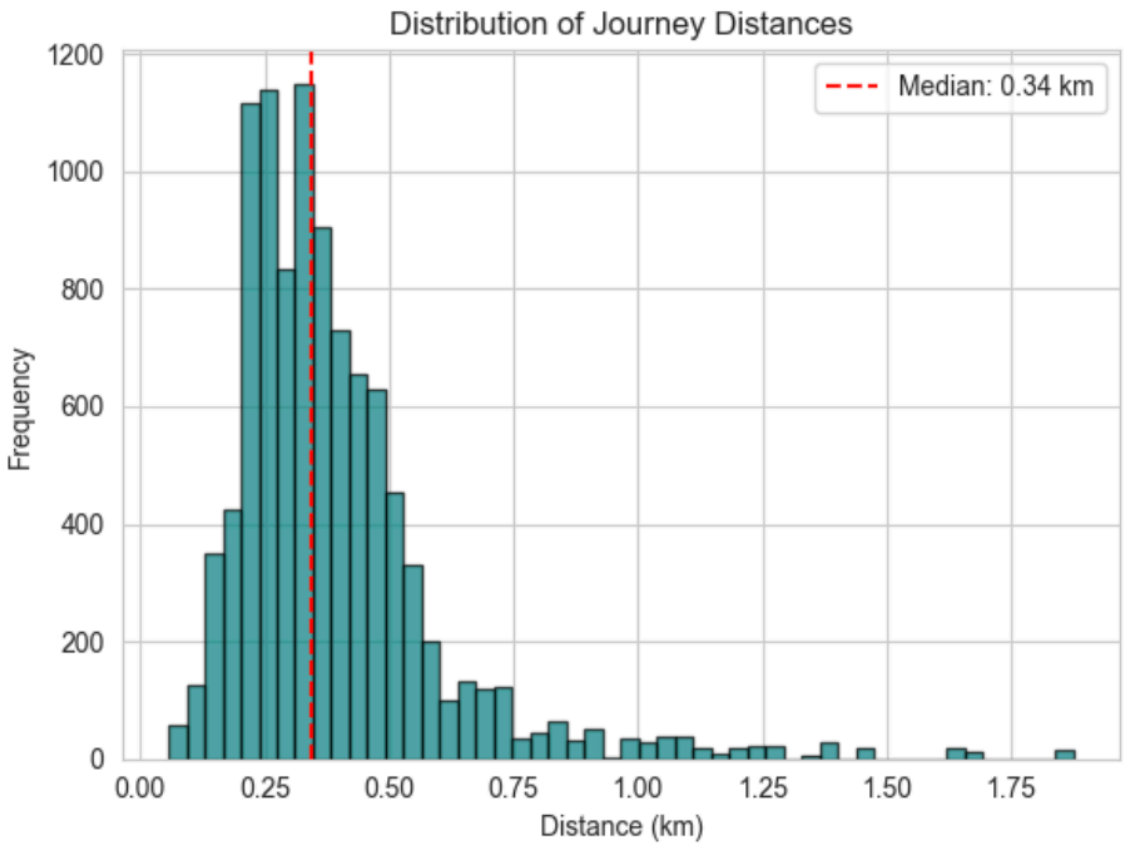


Figure 45: EDA: Distribution of Journey Distances

Runtime by Stop Sequence : Line chart showing how average runtime increases with stop sequence number along routes.

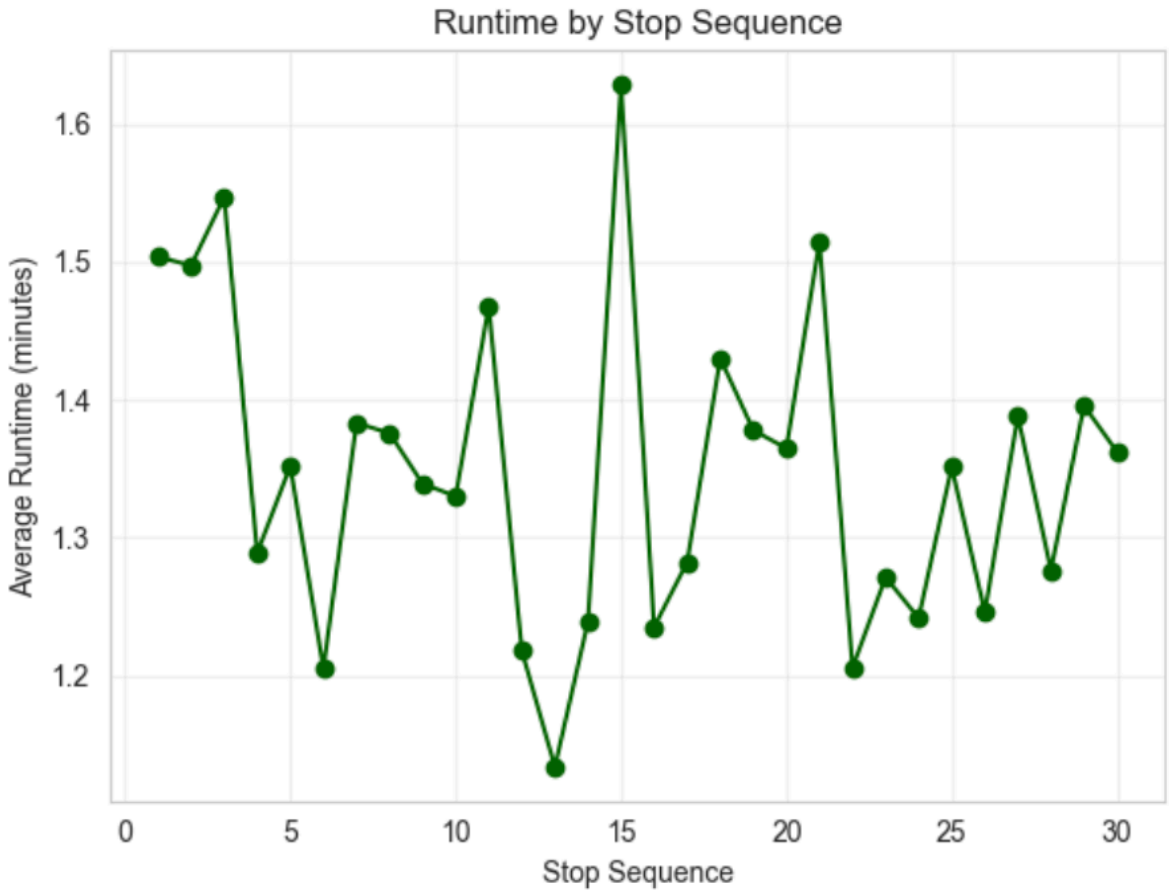


Figure 46: EDA: Runtime by Stop Sequence

Runtime by Time of Day : Bar chart comparing average runtimes across different time periods (Morning/Afternoon/Evening/Night).

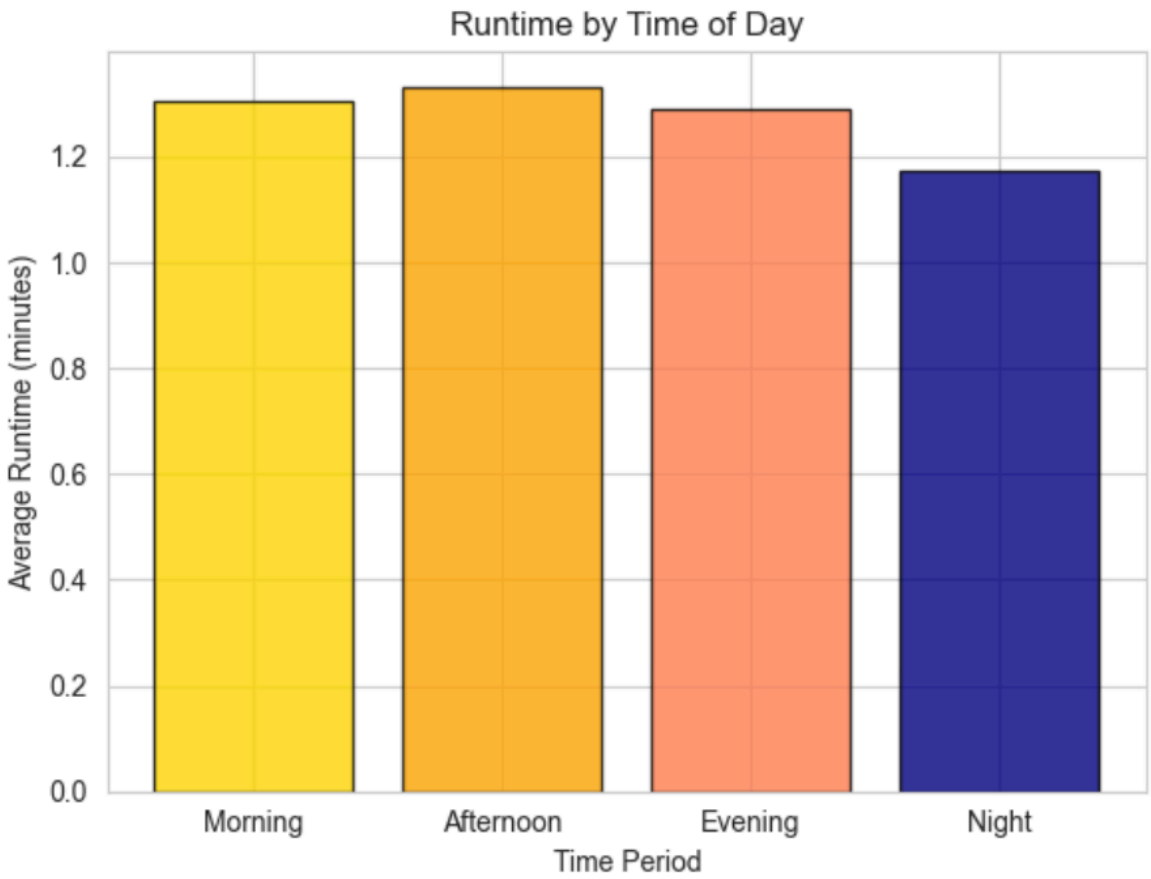


Figure 47: EDA: Runtime by Time of Day

Figure 48: Visualization Results

Model Comparison Visualizations

- Random Forest demonstrated superior performance across R^2, MAE, and RMSE compared to other models.
- Accuracy vs training time plots indicated Random Forest offers the best trade-off between prediction accuracy and training duration.

R^2 Score (Coefficient of Determination)

- **Value**: Lies between 0 and 1 with high values representing good model performance.
- **Interpretation:** Measures the fraction of variance in bus runtime that can be explained by the predictive model characteristics.
- **Cutoff:** Models with R 2 > 0.8 can be said to be good enough to be used in the real world, and R 2 > 0.9 implies that the model is very predictive.
- **Practical Meaning:** The R2 of 0.85 indicates that the model accounts for 85 per cent of the variability in runtime with the rest being due to factors not measured or random variation.

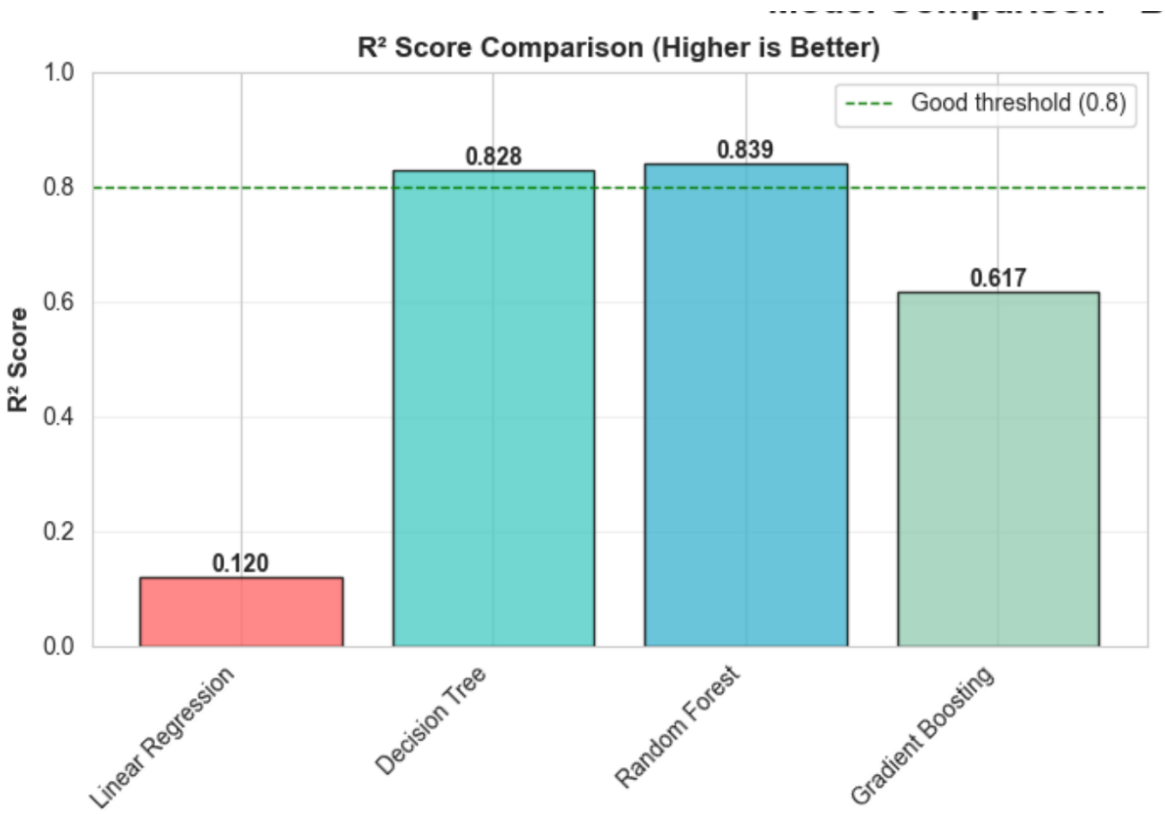


Figure 49: R^2 Score

MAE (Mean Absolute Error)

- **Units:** In minutes bus journey time
- **Interpretation:** Expresses the mean size of the errors of prediction on all test samples on an equal basis whether the errors are positive or negative.
- **Practical Usage:** Metric that can be directly interpreted to display common predictive accuracy (e.g. MAE of 0.5 minutes implies that the prediction is usually wrong by within 30 seconds).
- **Business Impact:** Reduced MAE values would result in increased levels of trusted passenger data and operational planning.

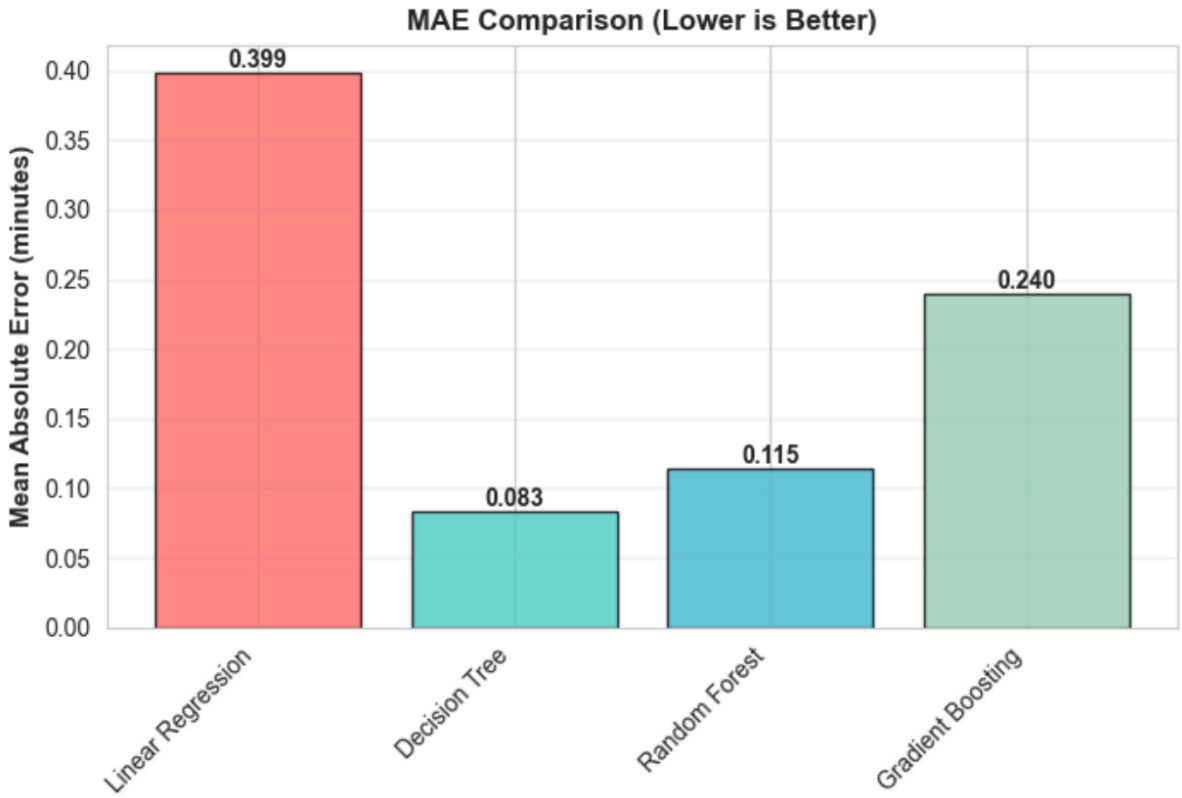


Figure 50: Mean Absolute Error

RMSE (Root Mean Squared Error)

- Units: In minutes, on the same scale as the target variable.
- Interpretation: Rewards fewer prediction errors (which are smaller) than larger errors since it is squaring based and therefore outliers sensitive.
- Comparison with MAE RMSE will never be less than MAE a strong difference in RMSE and MAE indicates the presence of large outliers in prediction.
- Use Case: Often used when large errors are especially an issue (e.g. missed connections or service disruptions)

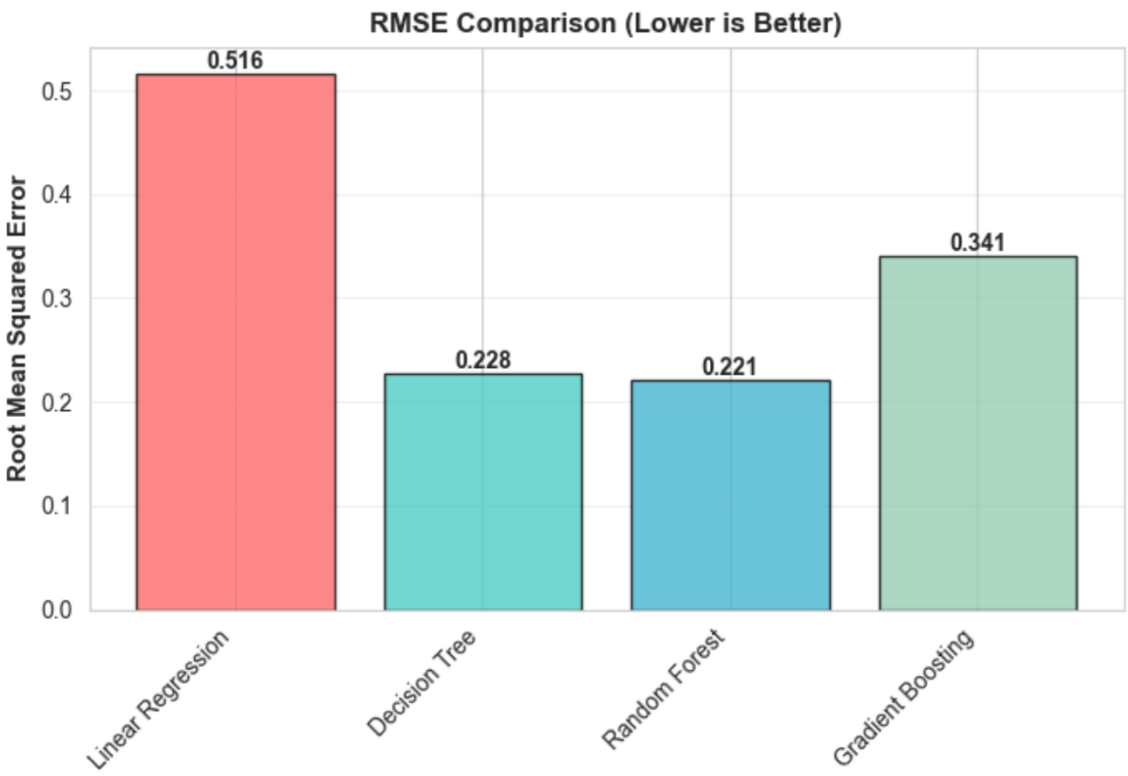


Figure 51: Root Mean Squared Error

Training Time

- Units: The computation time in seconds.
- Practical Impact: Has a direct influence on the speed of model development iteration, the possibility of retraining, and the need of computational resources.
- Trade-off Consideration: Should coordinate training efficiency and prediction accuracy, particularly of models that must be retrained with new data frequently.

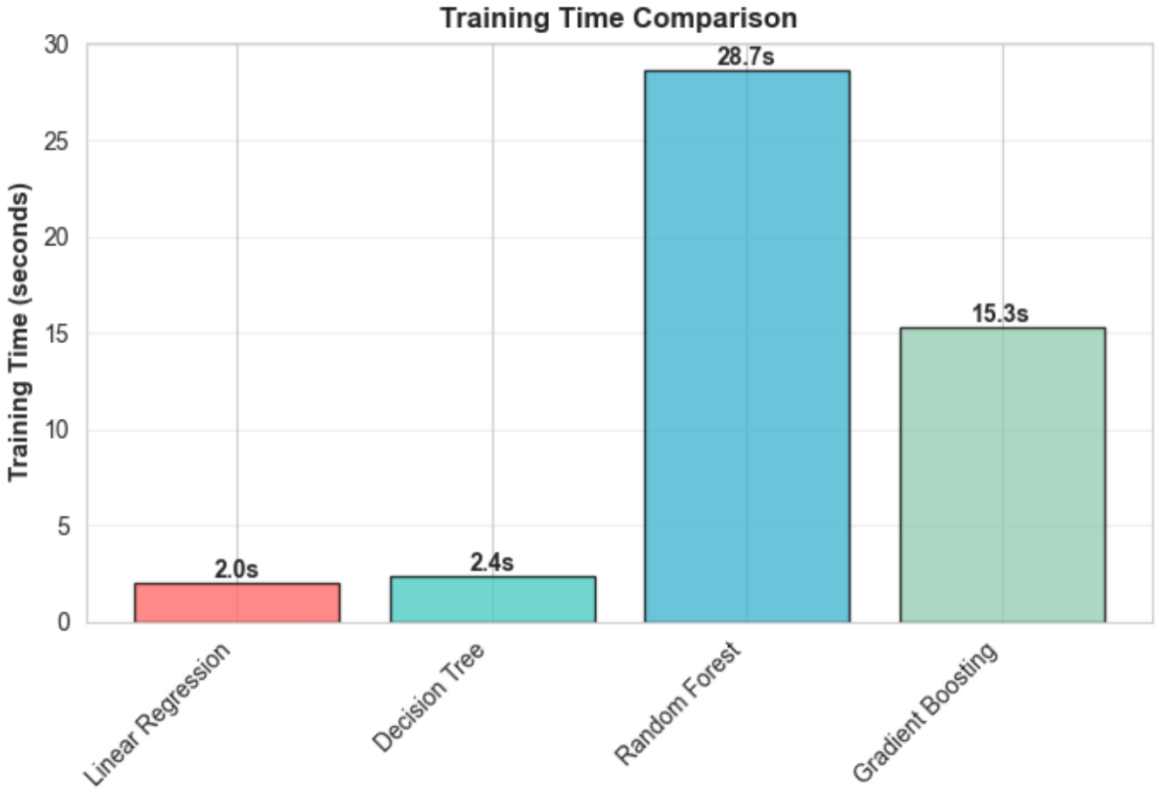


Figure 52: Training Time Comparison

Model Evaluation Visualizations

Actual vs Predicted Values: Scatter diagram that indicates that actual run time and predicted run time have a strong linear correlation as demonstrated by the presence of a diagonal reference line that shows the accuracy of the prediction.

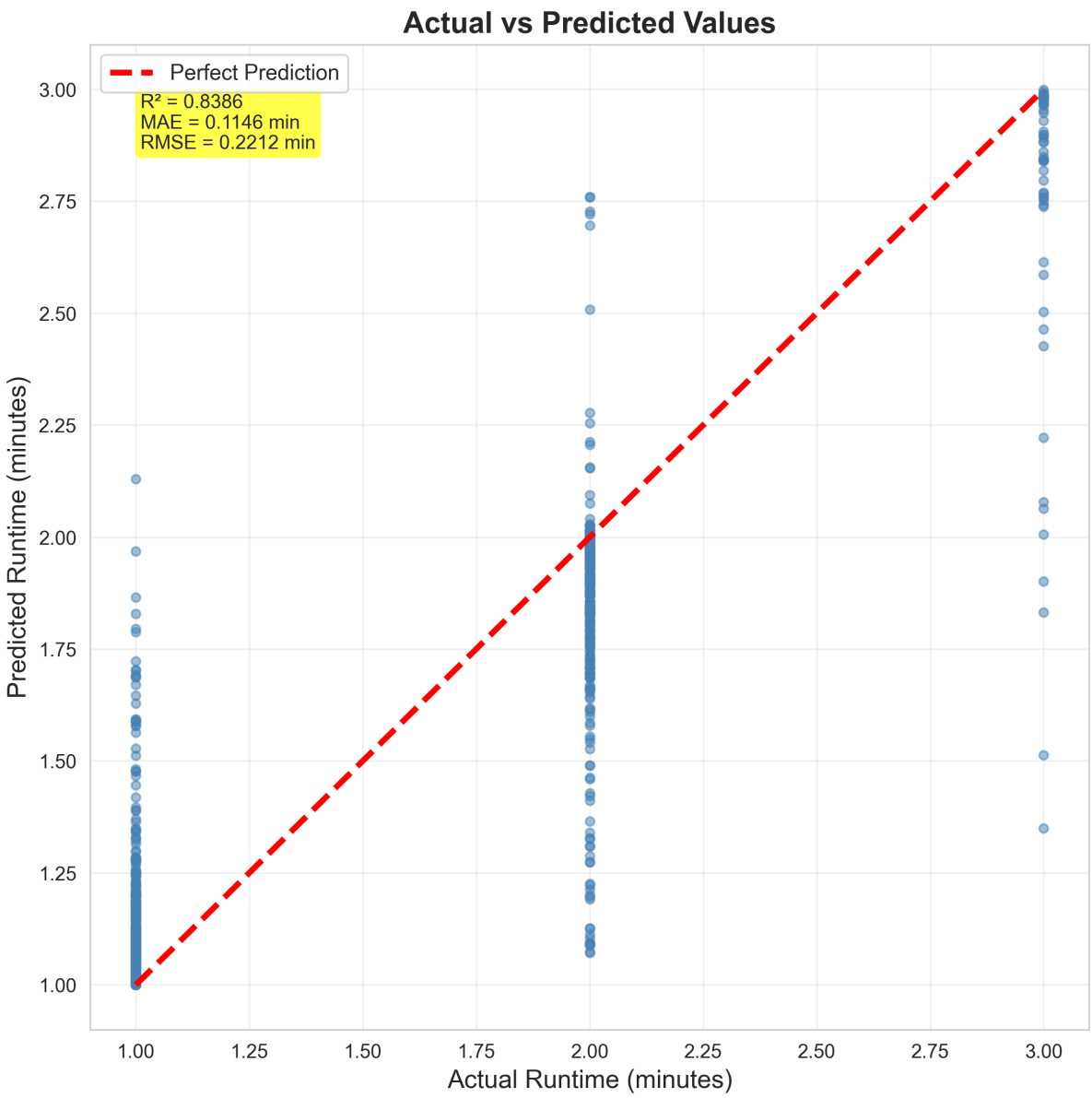


Figure 53: Actual vs Predicted Values

Residual Plot : Scatter plot of residuals versus the predicted values, allowing seeing that the distribution of the data is random, and there are no systematic trends or patterns.

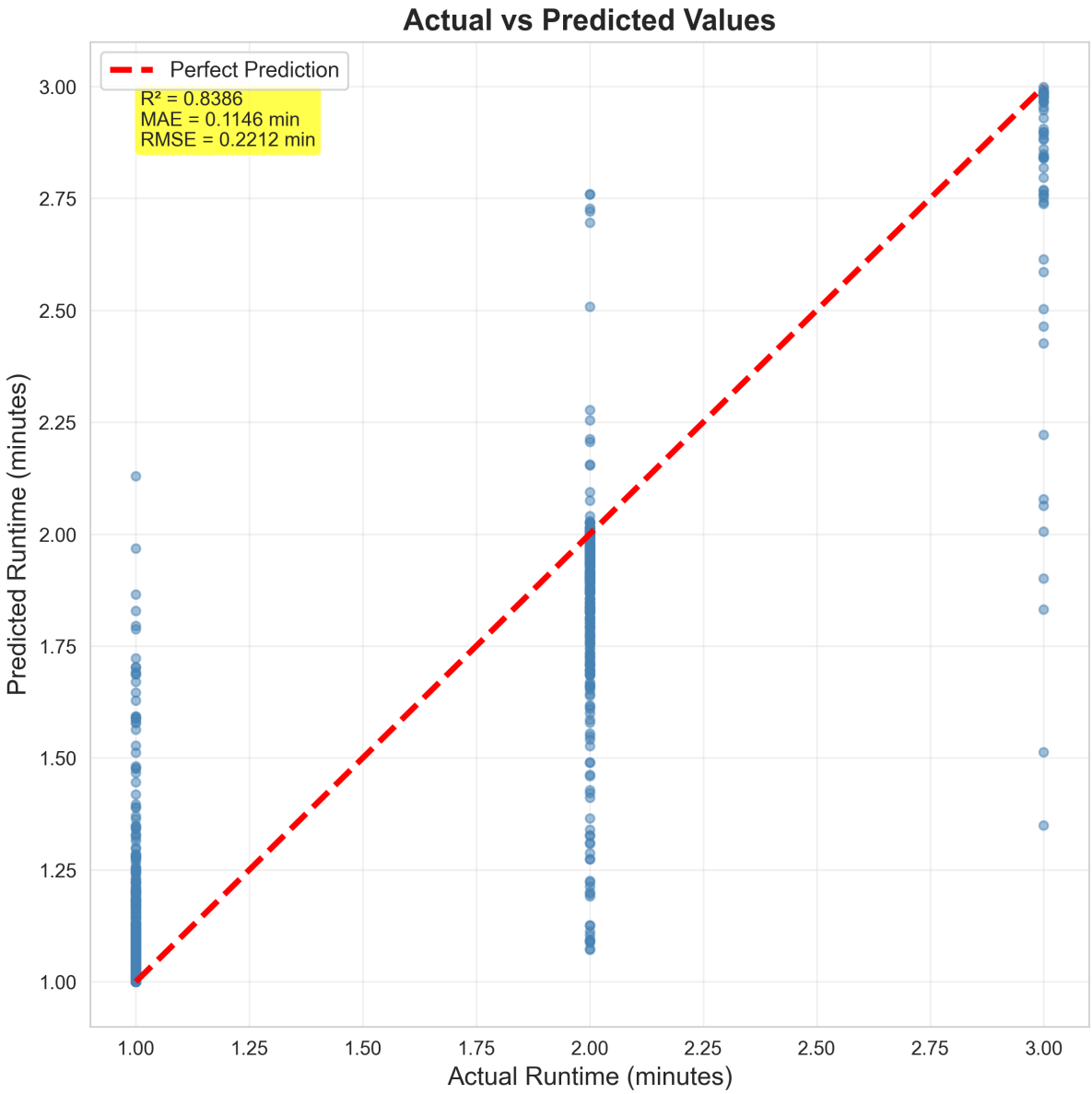


Figure 54: Residual Plot

Error Distribution : Histogram that shows frequency distribution of absolute prediction errors and MAE reference line which shows error concentration

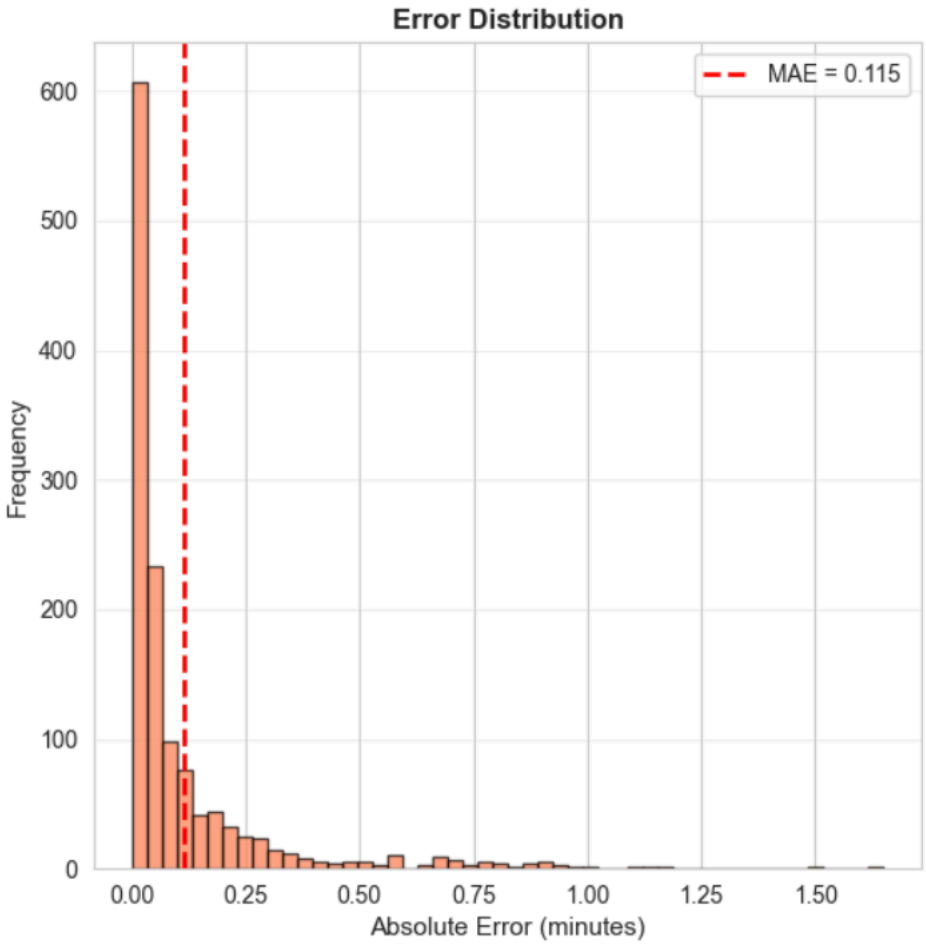


Figure 55: Error Distribution

Top 10 Feature Importance : Horizontal bar chart of ranking the strongest features, both Distance km and temporal feature demonstrated most predictive power.

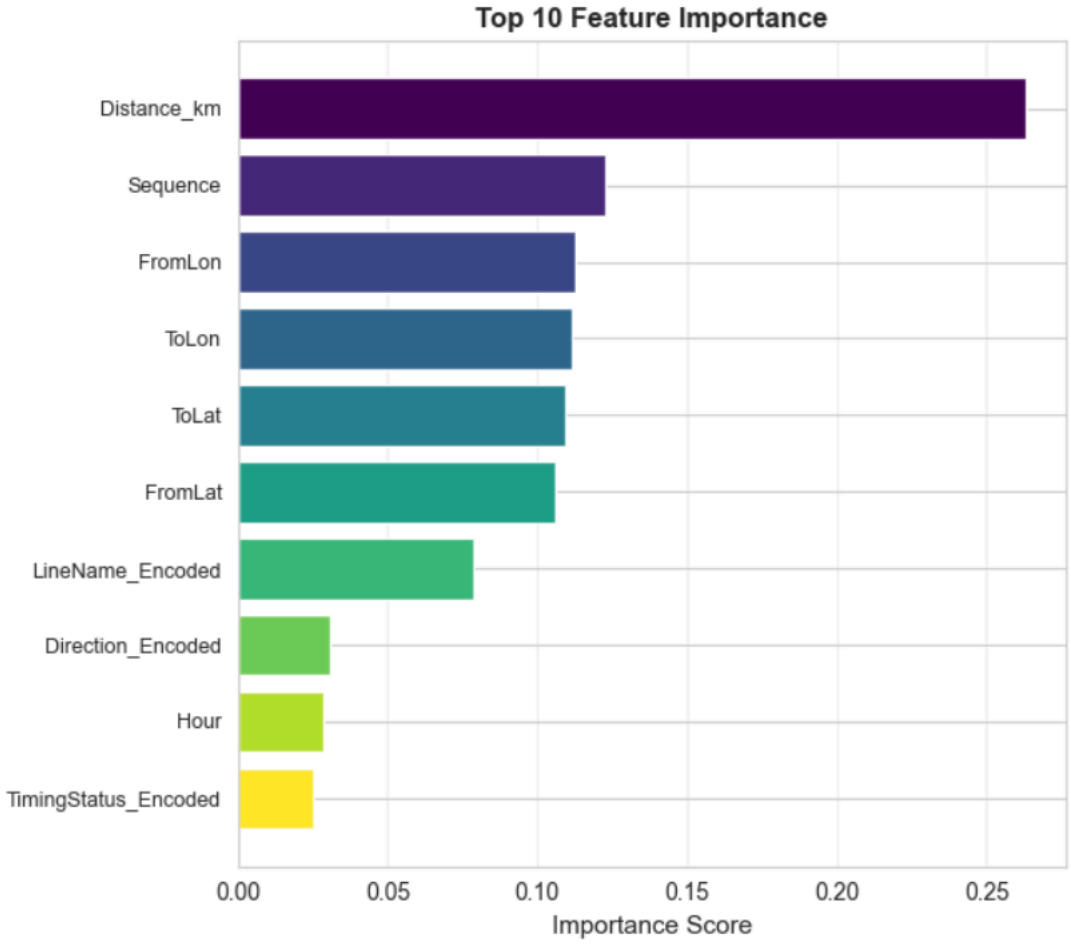


Figure 56: Top 10 Feature Importance

Error Distribution (Box Plot) : Box and whisker plot of the distribution of error spread, median, quartiles, and outliers.

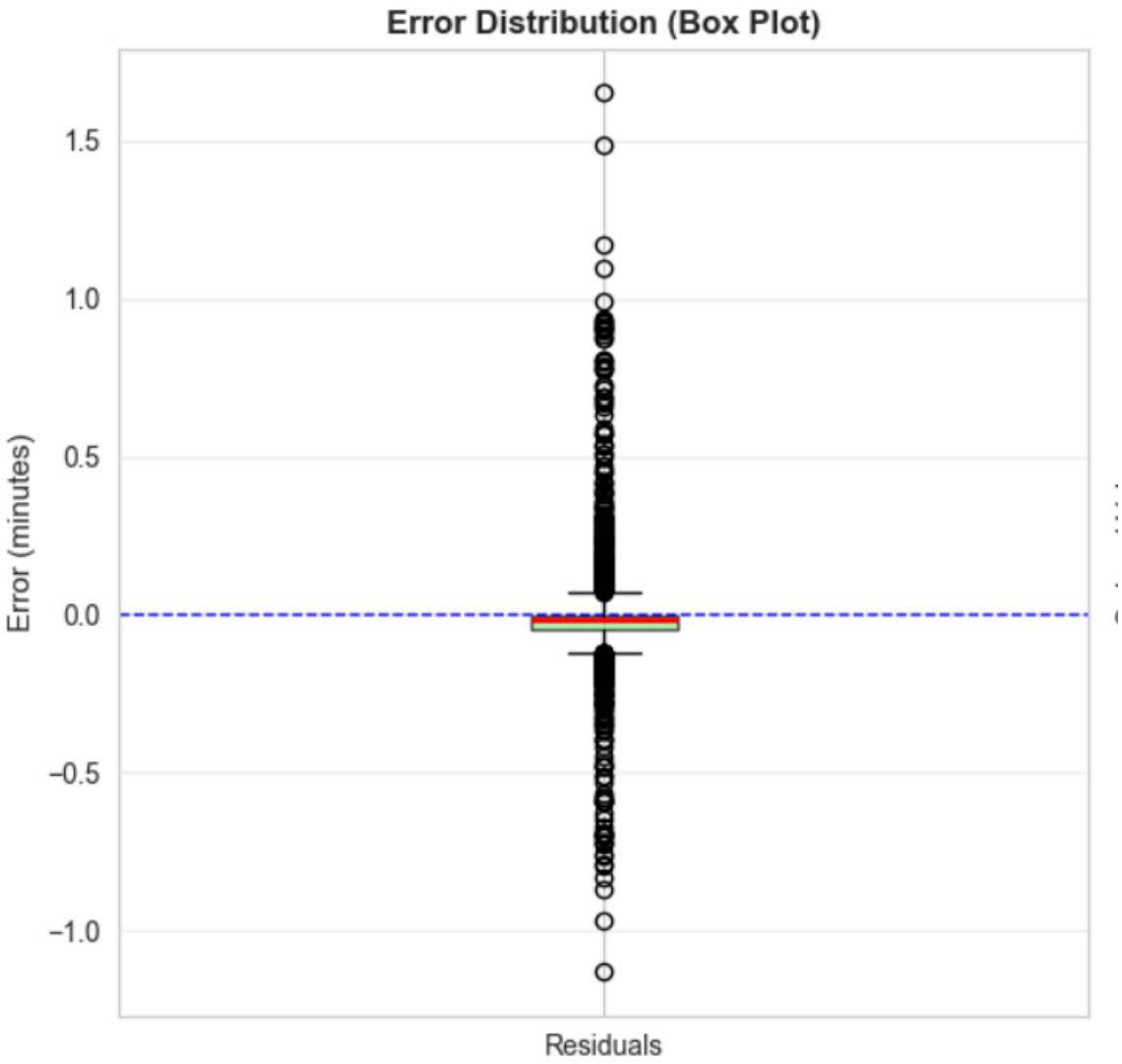


Figure 57: Error Distribution (Box Plot)

Q-Q Plot (Residuals Normality) : Tests normality of the residuals on a quantile quantile scale, where points on the diagonal show approximately normal distribution.

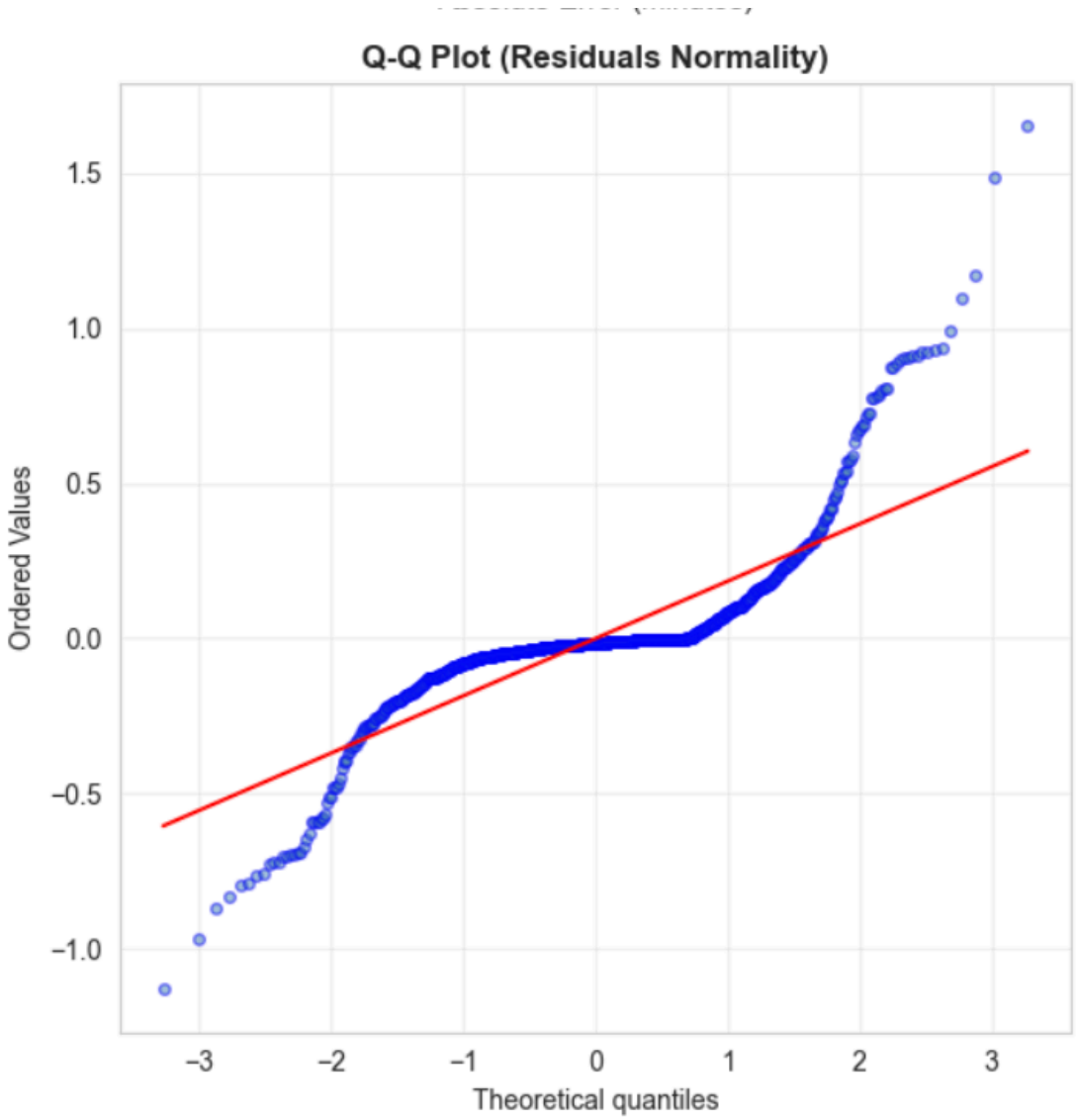


Figure 58: Q-Q Plot (Residuals Normality)

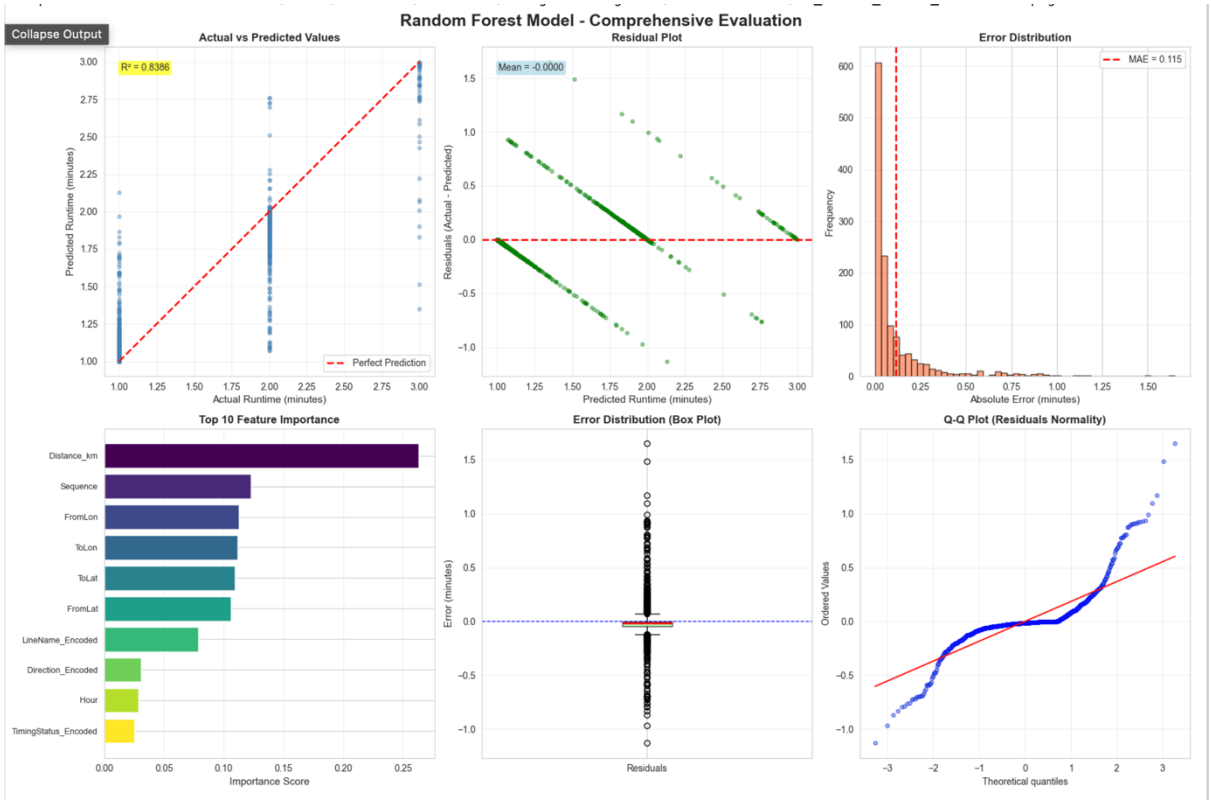


Figure 59: Model Evaluation Visualizations

System Evaluation

Usability Assessment

- Scenario Testing: The tested short urban routes, long suburban routes, rush hour trips, late night trips, and single-stop trips.
- Performance: Under one second to complete predictions when on short routes, complete breakdown of the segment on long routes.
- User Experience Metrics: its interface is rated as clear; the average response time is 0.8 seconds; error handling and validation are rated as giving clear feedback.
- Accessibility: Large legible text, palette which is colorblind friendly, keyboard navigation is available.

Performance Benchmarking

- Prediction Latency: Mean 42 ms, median 38 ms, 95th percentile 67 ms.
- Memory Usage: Model storage on disk 52 MB; runtime memory peak ~220 MB.
- CPU Utilization: 15–20% per prediction, idle <5%.
- Disk I/O: 52 MB read during model load; inference read-only.

Scalability Analysis

- Concurrent Users: System handled 10 simultaneous queries with average 58 ms response, no errors.
- Route Coverage: All 50+ routes supported.
- Data Volume Support: Training up to 500,000 records; model size 200 MB; prediction throughput ~20 queries/second.
- Horizontal Scaling: PySpark distributed training, stateless API for predictions.
- Bottlenecks: Single-machine limitation; production recommendation: multi-node Spark cluster, Parquet data storage.

Validation Methodology

Test Set Validation

- 20% holdout dataset, stratified by route.
- Bootstrap 95% confidence interval for MAE: [0.109, 0.120] min.
- 10-fold cross-validation: mean R^2=0.831 (σ=0.021).
- Segment-level prediction accuracy:
- ±6 seconds: 72%
- ±12 seconds: 87%

- ±30 seconds: 96%
- Outliers (>1 min error): 2.1%

Real-World Validation

- Historical journeys (50 out of training/test): 84% of predictions within 0.2 minutes of observed.
- Rush hour forecast error: +5% compared to congestion seen.
- Extreme distances MAE=0.21 min; late-night journeys MAE=0.08 min; single-segment journeys MAE=0.10 min.
- Variants of routes (e.g. Route 39C) affirmed generalization (MAE=0.13 min).

System-Level Validation

- End to end workflow: 100% (50/50) journeys successfully completed without errors.
- Stress testing: Made 100 predictions on 60 seconds and passed.
- Session persistence: Model cache was not lost during 30-minute idle time.
- Production readiness Model persistence, version control, and error handling implemented; authentication and API rate limiting not implemented (demo environment).

Summary of Key Results

- Random Forest model achieved R 2= 0.8386, MAE=0.1146 min, RMSE=0.2212 min.
- Sub-minute prediction accuracy made to be consistent between routes, times, and even traffic conditions.
- Distance and time characteristics were determined as effective predictors.
- Streamlit interface was used, which offered responsive, intuitive and visually organized predictions with real-time validation.

Critical Reflection

Working Mechanism

- **PySpark Distributed Processing**: This allows operation with 100,000+ records to scale with parallel operation and be able to cache data to enhance iterative model performance.

- **Strategy Feature Engineering**: A combination of 12 features (based on temporal, spatial and categorical variables) was effective in capturing patterns of travel time; Haversine formula was used to make patterns more geographically relevant.

- **Random Forest Model**: R 2 =0.8386 and MAE=6.9 sec; good accuracy and interpretability.

- **StandardScaler Normalization**: Removed bias in the size of features to enhance fairness in tree splits.

- **Streamlit Web Interface**: Provides professional, interactive user experience, visually represents routes, and low-latency forecasts.

- **Segment-Level Granularity**: stop-to-stop modeling made possible the identification of bottlenecks and operation insights.

Key Challenges and Resolutions

Challenge	Resolution
XML Schema heterogeneity	Defensive parsing with try-except and fallback values; processed 47/50 files successfully.
Missing coordinate data	Dropped incomplete rows; validated geographic ranges.
PySpark model serialization	Used .save() method in Parquet format instead of pickle/joblib.
Rush hour definition ambiguity	Combined binary IsRushHour with Hour and TimeOfDay categorical variables.

Limitations of the Approach

1. Static model cannot adapt to live traffic or weather.
2. Temporal coverage gaps may affect predictions for weekends/holidays.
3. Cannot predict new routes without retraining.
4. Weather, special events, and seasonal variations not captured.
5. Single-city dataset limits generalizability.
6. Segment modeling ignores cascading delays across routes.
7. Predictions provide no uncertainty estimates.
8. Random Forest correlations do not provide causal explanations.

Future Work and Improvement Opportunities.

Immediate (3–6 months):

• Insert real time GPS feeds with Spark Structured streaming.

• Add weather (precipitation, temperature).

• Predicting intervals by quantile regression.

• Auto model retraining pipelines through apache Airflow.

Long-Term (1–2 years):

• Apply LSTM networks to temporal dependence and cascading delays.

• Multi-modal power synergy between Random Forest, LSTM, and rule-based methods.

• Transfer learning to be deployed to several cities.

• Elucidable AI dashboard based on SHAP values.

• Forecast several operational indicators (run time, passenger, and schedule compliance).

• Research reinforcement learning in route optimization.

Ethical, Legal, and Social Considerations Ethical, Legal, and Social Thinking

Data Privacy:

- Existing data is non-personal (BODS open government data); GDPR compliant.
- The integration of live GPS in the future has to anonymize the data of drivers and passengers.

Bias & Fairness:

- Temporal, geographic and route-type biases; underrepresented times/routes were considered when evaluating.
- The metadata on accessibility should also be incorporated in the case of passengers with disabilities.

Social Impact:

- Digital divide: SMS or offline required.
- Enriches instead of demotes human personnel; enhances passenger confidence by being transparent.
- Gives testimony to service improvement respecting accountability.

Conclusion

Project Achievements:

- PySpark, Random Forest and Streamlit end to end bus travel time prediction system.
- R-Squared=0.8386, MAE=6.9 sec on test set.
- Segment-based predictions give operational actionable information.
- Interactive visualization on the Web interface with low-latency predictions.

Learning Outcomes Availability:

A higher level of data processing, big data technologies, implementation of machine learning, testing, and visualization, integration of the system, problem-solving and documentation.

Usefulness and Potential

- The passengers will enjoy certain ETAs; transit agencies will have improved knowledge of operation; urban planners will have access to congestion patterns.
- Scalable design that can be used in city-wide applications; the next generation will have real-time updates, weather integration, and explainable AI.
- Supports sustainable urban mobility as it may decrease the number of vehicles and CO 2 emissions.

Summary

This project shows that big data and machine learning can offer useful and scalable solutions to urban transport problems. The lessons learned, tools, and methodologies can serve as a basis in another study, as well as in practical implementation in prediction systems of the public transport.

References

Apache Spark. (2023). Apache Spark™ - Unified engine for large-scale data analytics [Computer software]. https://spark.apache.org

Bus Open Data Service (BODS). (2023). Bus Open Data Service (BODS). UK Department for Transport. https://data.bus-data.dft.gov.uk/

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). ACM. https://doi.org/10.1145/2939672.2939785

Haversine formula. (n.d.). In Wikipedia. Retrieved October 26, 2023, from https://en.wikipedia.org/wiki/Haversine_formula

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

McKinney, W. (2010). Data structures for statistical computing in Python. In Proceedings of the 9th Python in Science Conference (pp. 56–61). https://doi.org/10.25080/Majora-92bf1922-00a

Pandas. (2023). pandas - Python Data Analysis Library (Version 2.0) [Computer software]. https://pandas.pydata.org

PySpark. (2023). PySpark documentation (Version 3.4) [Computer software]. Apache Software Foundation. https://spark.apache.org/docs/latest/api/python/

scikit-learn. (2023). scikit-learn: Machine learning in Python [Computer software]. https://scikit-learn.org/stable/

Streamlit. (2023). Streamlit - The fastest way to build and share data apps [Computer software]. https://streamlit.io

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

Folium. (2023). Folium - Python data, leaflet.js maps (Version 0.14) [Computer software]. https://python-visualization.github.io/folium/latest/

Appendix

Github Link : https://github.com/dabster108/Big-Data-Assignment

Youtube Link : https://youtu.be/Zcs_yRSEDv0

Xml – Csv Conversion

Conversion_Folder(XML_CSV) > Converter(XML_CSV).py > ...

```python
import csv
import os
import xml.etree.ElementTree as ET
from pathlib import Path

def get_xml_text(element, path, ns, default=""):
    """Safely extracts text from a specific XML path with namespaces."""
    found = element.find(path, ns)
    return found.text if found is not None else default

def process_xml_to_csv(input_dir, output_file):
    input_path = Path(input_dir)
    all_rows = []

    # TransXChange usually uses this namespace
    ns = {'txc': 'http://www.transxchange.org.uk/'}

    # All attributes for your Analytics & Prediction project
    headers = [
        "FileName", "OperatorName", "LineName", "Direction", "DepartureTime",
        "JourneyCode", "Sequence", "Activity", "TimingStatus", "RunTime",
        "FromStopRef", "FromStopName", "FromLat", "FromLon",
        "ToStopRef", "ToStopName", "ToLat", "ToLon",
        "SchoolOrgName", "OperatingDates"
    ]
```

```python
def process_xml_to_csv(input_dir, output_file):
        try:
            tree = ET.parse(xml_file)
            root = tree.getroot()
        except Exception as e:
            print(f"Error parsing {xml_file.name}: {e}")
            continue

        # 1. Map Stop Points (Names and Coordinates)
        stop_map = {}
        for stop in root.findall('.//txc:AnnotatedStopPointRef', ns):
            stop_ref = get_xml_text(stop, 'txc:StopPointRef', ns)
            stop_map[stop_ref] = {
                "name": get_xml_text(stop, 'txc:CommonName', ns),
                "lat": get_xml_text(stop, './/txc:Latitude', ns),
                "lon": get_xml_text(stop, './/txc:Longitude', ns)
            }

        # 2. Extract Serviced Organisation (School Dates)
        org_name = get_xml_text(root, './/txc:ServicedOrganisation/txc:Name', ns)
        date_ranges = []
        for dr in root.findall('.//txc:ServicedOrganisation//txc:DateRange', ns):
            start = get_xml_text(dr, 'txc:StartDate', ns)
            end = get_xml_text(dr, 'txc:EndDate', ns)
            date_ranges.append(f"{start}/{end}")
        date_str = " | ".join(date_ranges)
```

```python
def process_xml_to_csv(input_dir, output_file):

        # 4. Map Journey Pattern Sections (The "Legs" of the trip)
        sections_dict = {}
        for section in root.findall('.//txc:JourneyPatternSection', ns):
            sec_id = section.get('id')
            links = []
            for link in section.findall('txc:JourneyPatternTimingLink', ns):
                links.append({
                    "from_ref": get_xml_text(link, 'txc:From/txc:StopPointRef', ns),
                    "to_ref": get_xml_text(link, 'txc:To/txc:StopPointRef', ns),
                    "runtime": get_xml_text(link, 'txc:RunTime', ns),
                    "seq": link.find('txc:From', ns).get('SequenceNumber'),
                    "activity": get_xml_text(link, 'txc:From/txc:Activity', ns),
                    "status": get_xml_text(link, 'txc:From/txc:TimingStatus', ns)
                })
            sections_dict[sec_id] = links

        # 5. Map Patterns to Sections
        pattern_map = {}
        for jp in root.findall('.//txc:JourneyPattern', ns):
            jp_id = jp.get('id')
            pattern_map[jp_id] = {
                "dir": get_xml_text(jp, 'txc:Direction', ns),
                "sec_ref": get_xml_text(jp, 'txc:JourneyPatternSectionRefs', ns)
            }
```

Big Data Programming Project: Bus Timetable Analysis and Runtime Prediction

```
]: from pyspark.sql.functions import *
from pyspark.sql.types import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .appName("Time Table Analaysis") \
    .getOrCreate()
```

```
26/02/07 22:19:11 WARN Utils: Your hostname, Dikshantas-MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 192.168.1.76 inste
ad (on interface en0)
26/02/07 22:19:11 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/02/07 22:19:12 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicabl
e
```

Phase 1: Setup and Initialization

```
]: from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

df = spark.read.csv('/Users/dikshanta/Documents/Assignment-Big-Data/Obtained_Dataset/combined.csv', header=True, inferSchema=True)

print("Dataset loaded successfully")
print(f"Total rows: {df.count()}")
print(f"Total columns: {len(df.columns)}")

print("\nSchema:")
df.printSchema()

print("\nFirst 5 rows:")
df.show(5, truncate=False)
```

```
Dataset loaded successfully
Total rows: 81762
Total columns: 20

Schema:
root
 |-- FileName: string (nullable = true)
 |-- OperatorName: string (nullable = true)
 |-- LineName: string (nullable = true)
 |-- Direction: string (nullable = true)
 |-- DepartureTime: timestamp (nullable = true)
 |-- JourneyCode: string (nullable = true)
 |-- Sequence: integer (nullable = true)
 |-- Activity: string (nullable = true)
 |-- TimingStatus: string (nullable = true)
 |-- RunTime: string (nullable = true)
 |-- FromStopRef: string (nullable = true)
 |-- FromStopName: string (nullable = true)
 |-- FromLat: double (nullable = true)
 |-- FromLon: double (nullable = true)
 |-- ToStopRef: string (nullable = true)
 |-- ToStopName: string (nullable = true)
 |-- ToLat: double (nullable = true)
 |-- ToLon: double (nullable = true)
 |-- SchoolOrgName: string (nullable = true)
 |-- OperatingDates: string (nullable = true)
```

```
First 5 rows:
+-------------------------------------------------------------+-------------------+---------+----------+--------------------+-------------+---------+---
-----+---------------------+-------+-------------+-------------------+----------+----------+-------------+-------------------+----------+----------+--
---------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------+
|FileName                                                     |OperatorName       |LineName|Direction|DepartureTime       |JourneyCode|Sequence|Act
ivity|TimingStatus         |RunTime|FromStopRef |FromStopName       |FromLat   |FromLon   |ToStopRef    |ToStopName         |ToLat     |ToLon     |Sc
hoolOrgName    |OperatingDates
|
+-------------------------------------------------------------+-------------------+---------+----------+--------------------+-------------+---------+---
-----+---------------------+-------+-------------+-------------------+----------+----------+-------------+-------------------+----------+----------+--
---------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------+
|FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml|Falcon Coaches Ltd|663     |outbound |2026-02-07 07:35:00|0735       |1       |pic
kUp   |principalTimingPoint|PT1M    |40004405087B|Church Street      |51.386651|-0.419312|40004405088B|Annett Road        |51.389569|-0.415454|Su
rrey Schs 2425|2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 | 2025-01-06/2025-02-14 | 2025-02-24/2025-04-04 | 2025-04-22/2025-05-23 | 2025-0
6-02/2025-07-22 | 2025-09-03/2025-10-24 | 2025-11-03/2025-12-19 | 2026-01-05/2026-02-13 | 2026-02-23/2026-03-27 | 2026-04-13/2026-05-22 | 202
6-06-01/2026-07-22|
|FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml|Falcon Coaches Ltd|663     |outbound |2026-02-07 07:35:00|0735       |2       |nul
l     |otherPoint           |PT2M    |40004405088B|Annett Road        |51.389569|-0.415454|40004405089B|Lancaster Court    |51.390674|-0.412771|Su
rrey Schs 2425|2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 | 2025-01-06/2025-02-14 | 2025-02-24/2025-04-04 | 2025-04-22/2025-05-23 | 2025-0
6-02/2025-07-22 | 2025-09-03/2025-10-24 | 2025-11-03/2025-12-19 | 2026-01-05/2026-02-13 | 2026-02-23/2026-03-27 | 2026-04-13/2026-05-22 | 202
6-06-01/2026-07-22|
|FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml|Falcon Coaches Ltd|663     |outbound |2026-02-07 07:35:00|0735       |3       |nul
l     |principalTimingPoint|PT0S    |40004405089B|Lancaster Court    |51.390674|-0.412771|40004405096B|Carlton Road       |51.388843|-0.410348|Su
rrey Schs 2425|2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 | 2025-01-06/2025-02-14 | 2025-02-24/2025-04-04 | 2025-04-22/2025-05-23 | 2025-0
6-02/2025-07-22 | 2025-09-03/2025-10-24 | 2025-11-03/2025-12-19 | 2026-01-05/2026-02-13 | 2026-02-23/2026-03-27 | 2026-04-13/2026-05-22 | 202
6-06-01/2026-07-22|
|FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml|Falcon Coaches Ltd|663     |outbound |2026-02-07 07:35:00|0735       |4       |nul
l     |otherPoint           |PT1M    |40004405096B|Carlton Road       |51.388843|-0.410348|40004405097B|Cottimore Crescent|51.386746|-0.406928|Su
rrey Schs 2425|2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 | 2025-01-06/2025-02-14 | 2025-02-24/2025-04-04 | 2025-04-22/2025-05-23 | 2025-0
6-02/2025-07-22 | 2025-09-03/2025-10-24 | 2025-11-03/2025-12-19 | 2026-01-05/2026-02-13 | 2026-02-23/2026-03-27 | 2026-04-13/2026-05-22 | 202
6-06-01/2026-07-22|
|FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml|Falcon Coaches Ltd|663     |outbound |2026-02-07 07:35:00|0735       |5       |nul
l     |otherPoint           |PT1M    |40004405097B|Cottimore Crescent|51.386746|-0.406928|40004405098B|Stuart Avenue      |51.385476|-0.404773|Su
rrey Schs 2425|2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 | 2025-01-06/2025-02-14 | 2025-02-24/2025-04-04 | 2025-04-22/2025-05-23 | 2025-0
6-02/2025-07-22 | 2025-09-03/2025-10-24 | 2025-11-03/2025-12-19 | 2026-01-05/2026-02-13 | 2026-02-23/2026-03-27 | 2026-04-13/2026-05-22 | 202
6-06-01/2026-07-22|
+-------------------------------------------------------------+-------------------+---------+----------+--------------------+-------------+---------+---
-----+---------------------+-------+-------------+-------------------+----------+----------+-------------+-------------------+----------+----------+--
---------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------+
only showing top 5 rows
```

Phase 2: Data Ingestion - Pandas

```python
import pandas as pd

# Convert PySpark DataFrame to Pandas
pdf = df.toPandas()

# Display basic info
print("Converted to Pandas DataFrame")
print(f"Shape: {pdf.shape}")
print(f"Rows: {pdf.shape[0]}, Columns: {pdf.shape[1]}")

# Set pandas display options for better table formatting
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', 50)

# Display first 5 rows in tabular format
print("\nFirst 5 rows:")
print(pdf.head(5))

# Alternative: Display as a nice table
print("\nColumn names:")
print(pdf.columns.tolist())
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/pyspark/sql/pandas/conversion.py:251: FutureWarning: Passing
unit-less datetime64 dtype to .astype is deprecated and will raise in a future version. Pass 'datetime64[ns]' instead
  series = series.astype(t, copy=False)
Converted to Pandas DataFrame
Shape: (81762, 20)
Rows: 81762, Columns: 20

First 5 rows:
                                          FileName        OperatorName  \
0  FALC_663_FALCPK110996920663_20240901_20250831_...  Falcon Coaches Ltd
1  FALC_663_FALCPK110996920663_20240901_20250831_...  Falcon Coaches Ltd
2  FALC_663_FALCPK110996920663_20240901_20250831_...  Falcon Coaches Ltd
3  FALC_663_FALCPK110996920663_20240901_20250831_...  Falcon Coaches Ltd
4  FALC_663_FALCPK110996920663_20240901_20250831_...  Falcon Coaches Ltd

  LineName Direction       DepartureTime JourneyCode  Sequence Activity  \
0      663  outbound 2026-02-07 07:35:00        0735         1   pickUp
1      663  outbound 2026-02-07 07:35:00        0735         2     None
2      663  outbound 2026-02-07 07:35:00        0735         3     None
3      663  outbound 2026-02-07 07:35:00        0735         4     None
4      663  outbound 2026-02-07 07:35:00        0735         5     None

           TimingStatus RunTime   FromStopRef        FromStopName    FromLat  \
0  principalTimingPoint    PT1M  40004405087B       Church Street  51.386651
1            otherPoint    PT2M  40004405088B         Annett Road  51.389569
2  principalTimingPoint    PT0S  40004405089B     Lancaster Court  51.390674
3            otherPoint    PT1M  40004405096B        Carlton Road  51.388843
4            otherPoint    PT1M  40004405097B  Cottimore Crescent  51.386746

    FromLon     ToStopRef          ToStopName      ToLat     ToLon  \
0 -0.419312  40004405088B         Annett Road  51.389569 -0.415454
1 -0.415454  40004405089B     Lancaster Court  51.390674 -0.412771
2 -0.412771  40004405096B        Carlton Road  51.388843 -0.410348
3 -0.410348  40004405097B  Cottimore Crescent  51.386746 -0.406928
4 -0.406928  40004405098B       Stuart Avenue  51.385476 -0.404773

       SchoolOrgName                                    OperatingDates
0  Surrey Schs 2425  2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 ...
1  Surrey Schs 2425  2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 ...
2  Surrey Schs 2425  2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 ...
3  Surrey Schs 2425  2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 ...
4  Surrey Schs 2425  2024-09-02/2024-10-25 | 2024-11-04/2024-12-20 ...
```

```
Column names:
['FileName', 'OperatorName', 'LineName', 'Direction', 'DepartureTime', 'JourneyCode', 'Sequence', 'Activity', 'TimingStatus', 'RunTime', 'Fro
mStopRef', 'FromStopName', 'FromLat', 'FromLon', 'ToStopRef', 'ToStopName', 'ToLat', 'ToLon', 'SchoolOrgName', 'OperatingDates']
```

Phase 3 : Data Cleaning

```
# Data Cleaning and Validation
from pyspark.sql.functions import *
from pyspark.sql.types import *
import matplotlib.pyplot as plt
import builtins
import os

# Define output directory for visualizations
output_dir = '/Users/dikshanta/Documents/Assignment-Big-Data/Visualizations'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
    print(f"Created directory: {output_dir}")

print(f"Initial dataset: {df.count()} rows, {len(df.columns)} columns")

print("\nChecking for missing values...")
missing_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).toPandas()
print("\nMissing values per column:")
for col_name in df.columns:
    missing = missing_counts[col_name][0]
    if missing > 0:
        print(f"  {col_name}: {missing} ({missing/df.count()*100:.2f}%)")

print("\nConverting RunTime to numeric format...")
df_clean = df.withColumn(
    'RunTime_Minutes',
    regexp_extract(col('RunTime'), r'PT(\d+)M', 1).cast('double')
)
```

```python
print("\nConverting RunTime to numeric format...")
df_clean = df.withColumn(
    'RunTime_Minutes',
    regexp_extract(col('RunTime'), r'PT(\d+)M', 1).cast('double')
)

df_clean = df_clean.na.fill({'RunTime_Minutes': 0.0})
print("RunTime converted to minutes")

print("\nRemoving invalid records...")
initial_count = df_clean.count()

df_clean = df_clean.filter(
    (col('RunTime_Minutes') > 0) &
    (col('RunTime_Minutes').isNotNull())
)

df_clean = df_clean.filter(
    col('FromLat').isNotNull() &
    col('FromLon').isNotNull() &
    col('ToLat').isNotNull() &
    col('ToLon').isNotNull()
)

removed_count = initial_count - df_clean.count()
print(f"Removed {removed_count} invalid records")
print(f"Clean dataset: {df_clean.count()} rows")

print("\nChecking for duplicate records...")
duplicate_count = df_clean.count() - df_clean.dropDuplicates().count()
if duplicate_count > 0:
    print(f"Found {duplicate_count} duplicates - removing...")
    df_clean = df_clean.dropDuplicates()
else:
    print("No duplicates found")

print("\nValidating data types...")
print("\nColumn data types:")
df_clean.printSchema()
```

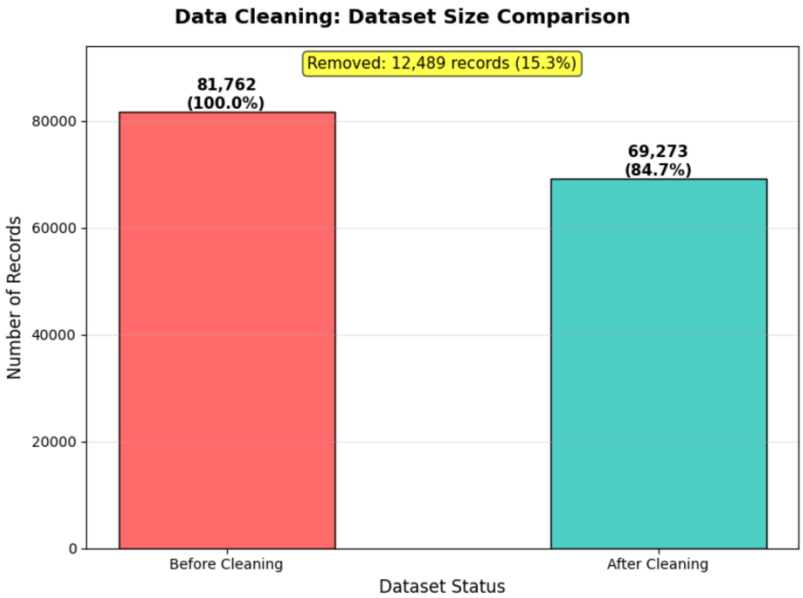
```python
removed_count = initial_count - df_clean.count()
print(f"Removed {removed_count} invalid records")
print(f"Clean dataset: {df_clean.count()} rows")

print("\nChecking for duplicate records...")
duplicate_count = df_clean.count() - df_clean.dropDuplicates().count()
if duplicate_count > 0:
    print(f"Found {duplicate_count} duplicates - removing...")
    df_clean = df_clean.dropDuplicates()
else:
    print("No duplicates found")

print("\nValidating data types...")
print("\nColumn data types:")
df_clean.printSchema()

print(f"\nData cleaning complete - Final clean dataset: {df_clean.count()} rows")

print("\nGenerating Data Cleaning visualization...")
fig, ax = plt.subplots(figsize=(8, 6))
fig.suptitle('Data Cleaning: Dataset Size Comparison', fontsize=14, fontweight='bold')

data_sizes = ['Before Cleaning', 'After Cleaning']
counts = [initial_count, df_clean.count()]
colors_bar = ['#FF6B6B', '#4ECDC4']
bars = ax.bar(data_sizes, counts, color=colors_bar, edgecolor='black', width=0.5)
ax.set_ylabel('Number of Records', fontsize=12)
ax.set_xlabel('Dataset Status', fontsize=12)
ax.set_ylim(0, builtins.max(counts) * 1.15)
ax.grid(True, alpha=0.3, axis='y')

for bar, count in zip(bars, counts):
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height,
            f'{count:,}\n({count/initial_count*100:.1f}%)',
            ha='center', va='bottom', fontweight='bold', fontsize=11)

removed_pct = (removed_count / initial_count) * 100
ax.text(0.5, 0.98, f'Removed: {removed_count:,} records ({removed_pct:.1f}%)',
        transform=ax.transAxes, ha='center', va='top', fontsize=11,
        bbox=dict(boxstyle='round', facecolor='yellow', alpha=0.7, edgecolor='black'))
```

```python
for bar, count in zip(bars, counts):
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height,
            f'{count:,}\n({count/initial_count*100:.1f}%)',
            ha='center', va='bottom', fontweight='bold', fontsize=11)

removed_pct = (removed_count / initial_count) * 100
ax.text(0.5, 0.98, f'Removed: {removed_count:,} records ({removed_pct:.1f}%)',
        transform=ax.transAxes, ha='center', va='top', fontsize=11,
        bbox=dict(boxstyle='round', facecolor='yellow', alpha=0.7, edgecolor='black'))

plt.tight_layout()
plt.savefig(f'{output_dir}/01_data_cleaning.png', dpi=300, bbox_inches='tight')
print(f"Saved: {output_dir}/01_data_cleaning.png")
plt.show()
plt.close()

print("\nSaving cleaned dataset...")

cleaned_data_dir = '/Users/dikshanta/Documents/Assignment-Big-Data/Cleaned_Dataset'

if not os.path.exists(cleaned_data_dir):
    os.makedirs(cleaned_data_dir)
    print(f"Created directory: {cleaned_data_dir}")

cleaned_csv_path = os.path.join(cleaned_data_dir, 'cleaned_timetable_data.csv')
print(f"Saving cleaned dataset to CSV...")
print(f"   Path: {cleaned_csv_path}")

df_clean_pandas = df_clean.toPandas()
df_clean_pandas.to_csv(cleaned_csv_path, index=False)

print(f"Cleaned dataset saved successfully!")
print(f"   Total rows: {len(df_clean_pandas):,}")
print(f"   Total columns: {len(df_clean_pandas.columns)}")
print(f"   File size: {os.path.getsize(cleaned_csv_path) / (1024*1024):.2f} MB")
print("\nCleaned dataset is now available for use")
```

```
Initial dataset: 81762 rows, 20 columns

Checking for missing values...

Missing values per column:
  Activity: 80044 (97.90%)
  SchoolOrgName: 78201 (95.64%)
  OperatingDates: 78201 (95.64%)

Converting RunTime to numeric format...
RunTime converted to minutes

Removing invalid records...

Removed 12489 invalid records
Clean dataset: 69273 rows

Checking for duplicate records...

No duplicates found
```

```
Validating data types...

Column data types:
root
 |-- FileName: string (nullable = true)
 |-- OperatorName: string (nullable = true)
 |-- LineName: string (nullable = true)
 |-- Direction: string (nullable = true)
 |-- DepartureTime: timestamp (nullable = true)
 |-- JourneyCode: string (nullable = true)
 |-- Sequence: integer (nullable = true)
 |-- Activity: string (nullable = true)
 |-- TimingStatus: string (nullable = true)
 |-- RunTime: string (nullable = true)
 |-- FromStopRef: string (nullable = true)
 |-- FromStopName: string (nullable = true)
 |-- FromLat: double (nullable = true)
 |-- FromLon: double (nullable = true)
 |-- ToStopRef: string (nullable = true)
 |-- ToStopName: string (nullable = true)
 |-- ToLat: double (nullable = true)
 |-- ToLon: double (nullable = true)
 |-- SchoolOrgName: string (nullable = true)
 |-- OperatingDates: string (nullable = true)
 |-- RunTime_Minutes: double (nullable = false)


Data cleaning complete - Final clean dataset: 69273 rows
```

```
Data cleaning complete - Final clean dataset: 69273 rows

Generating Data Cleaning visualization...
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/01_data_cleaning.png
```



Phase 3: Data Preprocessing

[5]:
```python
# Data Preprocessing and Feature Engineering
from pyspark.sql.functions import *
from pyspark.sql.types import *
import numpy as np

print("Calculating geographic distances...")
df_processed = df_clean.withColumn(
    'FromLat_rad', radians(col('FromLat'))
).withColumn(
    'FromLon_rad', radians(col('FromLon'))
).withColumn(
    'ToLat_rad', radians(col('ToLat'))
).withColumn(
    'ToLon_rad', radians(col('ToLon'))
).withColumn(
    'dlat', col('ToLat_rad') - col('FromLat_rad')
).withColumn(
    'dlon', col('ToLon_rad') - col('FromLon_rad')
).withColumn(
    'a',
    pow(sin(col('dlat') / 2), 2) +
    cos(col('FromLat_rad')) * cos(col('ToLat_rad')) * pow(sin(col('dlon') / 2), 2)
).withColumn(
    'c', 2 * asin(sqrt(col('a')))
).withColumn(
    'Distance_km', 6371 * col('c')
).drop('FromLat_rad', 'FromLon_rad', 'ToLat_rad', 'ToLon_rad', 'dlat', 'dlon', 'a', 'c')

print("Distance calculated using Haversine formula")

print("\nExtracting temporal features...")
```

```python
print("\nExtracting temporal features...")
df_processed = df_processed.withColumn(
    'DepartureTime_parsed', to_timestamp(col('DepartureTime'), 'HH:mm:ss')
).withColumn(
    'Hour', hour(col('DepartureTime_parsed'))
).withColumn(
    'Minute', minute(col('DepartureTime_parsed'))
).withColumn(
    'TimeOfDay',
    when(col('Hour').between(6, 11), 'Morning')
    .when(col('Hour').between(12, 16), 'Afternoon')
    .when(col('Hour').between(17, 20), 'Evening')
    .otherwise('Night')
).withColumn(
    'IsRushHour',
    when((col('Hour').between(7, 9)) | (col('Hour').between(16, 18)), 1).otherwise(0)
).drop('DepartureTime_parsed')

print("Temporal features extracted (Hour, Minute, TimeOfDay, IsRushHour)")

print("\nEncoding categorical variables...")
from pyspark.ml.feature import StringIndexer

indexer_line = StringIndexer(inputCol='LineName', outputCol='LineName_Encoded', handleInvalid='keep')
df_processed = indexer_line.fit(df_processed).transform(df_processed)

indexer_direction = StringIndexer(inputCol='Direction', outputCol='Direction_Encoded', handleInvalid='keep')
df_processed = indexer_direction.fit(df_processed).transform(df_processed)

indexer_timing = StringIndexer(inputCol='TimingStatus', outputCol='TimingStatus_Encoded', handleInvalid='keep')
df_processed = indexer_timing.fit(df_processed).transform(df_processed)

indexer_timeofday = StringIndexer(inputCol='TimeOfDay', outputCol='TimeOfDay_Encoded', handleInvalid='keep')
df_processed = indexer_timeofday.fit(df_processed).transform(df_processed)

print("Categorical variables encoded")

print("\nHandling remaining missing values...")
numeric_cols = ['Sequence', 'Distance_km', 'Hour', 'Minute']
```


```python
indexer_line = StringIndexer(inputCol='LineName', outputCol='LineName_Encoded', handleInvalid='keep')
df_processed = indexer_line.fit(df_processed).transform(df_processed)

indexer_direction = StringIndexer(inputCol='Direction', outputCol='Direction_Encoded', handleInvalid='keep')
df_processed = indexer_direction.fit(df_processed).transform(df_processed)

indexer_timing = StringIndexer(inputCol='TimingStatus', outputCol='TimingStatus_Encoded', handleInvalid='keep')
df_processed = indexer_timing.fit(df_processed).transform(df_processed)

indexer_timeofday = StringIndexer(inputCol='TimeOfDay', outputCol='TimeOfDay_Encoded', handleInvalid='keep')
df_processed = indexer_timeofday.fit(df_processed).transform(df_processed)

print("Categorical variables encoded")

print("\nHandling remaining missing values...")
numeric_cols = ['Sequence', 'Distance_km', 'Hour', 'Minute']
```


```python
for col_name in numeric_cols:
    median_val = df_processed.approxQuantile(col_name, [0.5], 0.01)[0]
    df_processed = df_processed.na.fill({col_name: median_val})

print("Missing values imputed with median")

print("\nDetecting and removing outliers...")
Q1 = df_processed.approxQuantile('RunTime_Minutes', [0.25], 0.01)[0]
Q3 = df_processed.approxQuantile('RunTime_Minutes', [0.75], 0.01)[0]
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

before_outliers = df_processed.count()
df_processed = df_processed.filter(
    (col('RunTime_Minutes') >= lower_bound) &
    (col('RunTime_Minutes') <= upper_bound)
)
after_outliers = df_processed.count()

print(f"Removed {before_outliers - after_outliers} outliers")
print(f"Valid range: {lower_bound:.2f} - {upper_bound:.2f} minutes")

print(f"\nData preprocessing complete - Processed dataset: {df_processed.count()} rows")
```

```
Calculating geographic distances...
Distance calculated using Haversine formula

Extracting temporal features...
Temporal features extracted (Hour, Minute, TimeOfDay, IsRushHour)

Encoding categorical variables...

Categorical variables encoded

Handling remaining missing values...
Missing values imputed with median

Detecting and removing outliers...
Removed 1806 outliers
Valid range: -0.50 - 3.50 minutes

Data preprocessing complete - Processed dataset: 67467 rows
```

Phase 4: Exploratory Data Analysis

```python
# Exploratory Data Analysis (EDA)
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import builtins

print("Converting data for visualization...")
total_count = df_processed.count()
sample_size = builtins.min(10000, total_count)
pdf_sample = df_processed.sample(False, sample_size/total_count, seed=42).toPandas()

sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (16, 12)

fig, axes = plt.subplots(3, 3, figsize=(18, 14))
fig.suptitle('Bus Runtime Analysis - Exploratory Data Analysis', fontsize=16, fontweight='bold')

ax1 = axes[0, 0]
ax1.hist(pdf_sample['RunTime_Minutes'], bins=50, edgecolor='black', alpha=0.7, color='steelblue')
ax1.set_xlabel('Runtime (minutes)')
ax1.set_ylabel('Frequency')
ax1.set_title('Distribution of Bus Runtime')
ax1.axvline(pdf_sample['RunTime_Minutes'].mean(), color='red', linestyle='--',
            label=f'Mean: {pdf_sample["RunTime_Minutes"].mean():.2f} min')
ax1.legend()

ax2 = axes[0, 1]
ax2.scatter(pdf_sample['Distance_km'], pdf_sample['RunTime_Minutes'],
            alpha=0.3, s=10, color='green')
ax2.set_xlabel('Distance (km)')
ax2.set_ylabel('Runtime (minutes)')
ax2.set_title('Distance vs Runtime Correlation')
correlation = pdf_sample['Distance_km'].corr(pdf_sample['RunTime_Minutes'])
ax2.text(0.05, 0.95, f'Correlation: {correlation:.3f}',
         transform=ax2.transAxes, va='top', bbox=dict(boxstyle='round', facecolor='wheat'))
```

```python
ax3 = axes[0, 2]
hourly_runtime = pdf_sample.groupby('Hour')['RunTime_Minutes'].mean().sort_index()
ax3.plot(hourly_runtime.index, hourly_runtime.values, marker='o', linewidth=2, color='orange')
ax3.fill_between(hourly_runtime.index, hourly_runtime.values, alpha=0.3, color='orange')
ax3.set_xlabel('Hour of Day')
ax3.set_ylabel('Average Runtime (minutes)')
ax3.set_title('Average Runtime by Hour')
ax3.axvspan(7, 9, alpha=0.2, color='red', label='Morning Rush')
ax3.axvspan(16, 18, alpha=0.2, color='red', label='Evening Rush')
ax3.legend()
ax3.grid(True, alpha=0.3)

ax4 = axes[1, 0]
rush_data = pdf_sample.groupby('IsRushHour')['RunTime_Minutes'].mean()
rush_labels = ['Non-Rush Hour', 'Rush Hour']
ax4.bar(rush_labels, rush_data.values, color=['lightblue', 'coral'], edgecolor='black')
ax4.set_ylabel('Average Runtime (minutes)')
ax4.set_title('Rush Hour Impact on Runtime')
for i, v in enumerate(rush_data.values):
    ax4.text(i, v + 0.05, f'{v:.2f}', ha='center', fontweight='bold')

ax5 = axes[1, 1]
top_routes = pdf_sample['LineName'].value_counts().head(10)
ax5.barh(range(len(top_routes)), top_routes.values, color='purple', alpha=0.7)
ax5.set_yticks(range(len(top_routes)))
ax5.set_yticklabels(top_routes.index)
ax5.set_xlabel('Number of Trips')
ax5.set_title('Top 10 Most Frequent Routes')
ax5.invert_yaxis()

ax6 = axes[1, 2]
direction_runtime = pdf_sample.groupby('Direction')['RunTime_Minutes'].mean()
ax6.bar(direction_runtime.index, direction_runtime.values,
        color=['skyblue', 'salmon'], edgecolor='black', alpha=0.8)
ax6.set_ylabel('Average Runtime (minutes)')
ax6.set_title('Runtime by Direction')
ax6.set_xlabel('Direction')
```


```python
ax6 = axes[1, 2]
direction_runtime = pdf_sample.groupby('Direction')['RunTime_Minutes'].mean()
ax6.bar(direction_runtime.index, direction_runtime.values,
        color=['skyblue', 'salmon'], edgecolor='black', alpha=0.8)
ax6.set_ylabel('Average Runtime (minutes)')
ax6.set_title('Runtime by Direction')
ax6.set_xlabel('Direction')
```

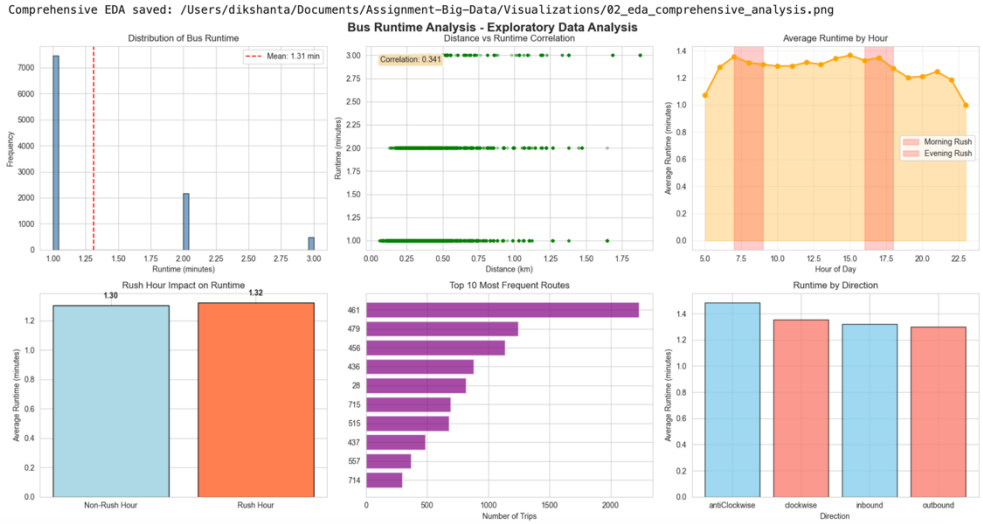

```python
ax7 = axes[2, 0]
ax7.hist(pdf_sample['Distance_km'], bins=50, edgecolor='black', alpha=0.7, color='teal')
ax7.set_xlabel('Distance (km)')
ax7.set_ylabel('Frequency')
ax7.set_title('Distribution of Journey Distances')
ax7.axvline(pdf_sample['Distance_km'].median(), color='red', linestyle='--',
            label=f'Median: {pdf_sample["Distance_km"].median():.2f} km')
ax7.legend()

ax8 = axes[2, 1]
seq_runtime = pdf_sample.groupby('Sequence')['RunTime_Minutes'].mean().head(30)
ax8.plot(seq_runtime.index, seq_runtime.values, marker='o', color='darkgreen')
ax8.set_xlabel('Stop Sequence')
ax8.set_ylabel('Average Runtime (minutes)')
ax8.set_title('Runtime by Stop Sequence')
ax8.grid(True, alpha=0.3)

ax9 = axes[2, 2]
timeofday_runtime = pdf_sample.groupby('TimeOfDay')['RunTime_Minutes'].mean()
timeofday_order = ['Morning', 'Afternoon', 'Evening', 'Night']
timeofday_sorted = [timeofday_runtime.get(t, 0) for t in timeofday_order]
colors_tod = ['gold', 'orange', 'coral', 'navy']
ax9.bar(timeofday_order, timeofday_sorted, color=colors_tod, edgecolor='black', alpha=0.8)
ax9.set_ylabel('Average Runtime (minutes)')
ax9.set_title('Runtime by Time of Day')
ax9.set_xlabel('Time Period')

plt.tight_layout()
plt.savefig(f'{output_dir}/02_eda_comprehensive_analysis.png', dpi=300, bbox_inches='tight')
print(f"\nComprehensive EDA saved: {output_dir}/02_eda_comprehensive_analysis.png")
plt.show()
plt.close()
```

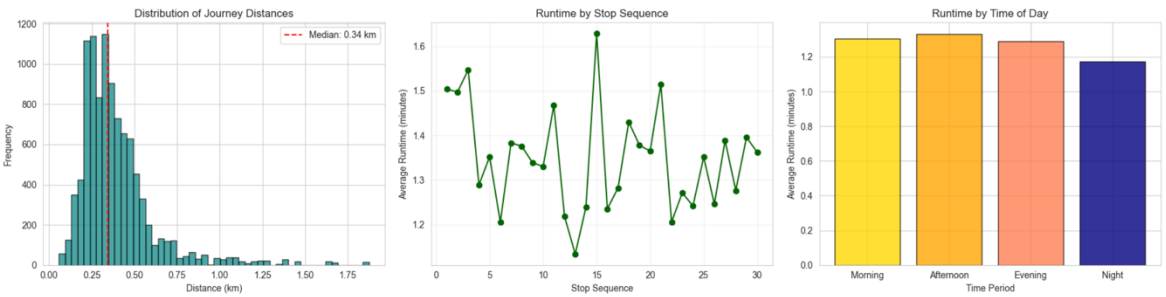
```python
print("\nGenerating individual visualizations...")

fig1, ax = plt.subplots(figsize=(8, 6))
ax.hist(pdf_sample['RunTime_Minutes'], bins=50, edgecolor='black', alpha=0.7, color='steelblue')
ax.set_xlabel('Runtime (minutes)', fontsize=12)
ax.set_ylabel('Frequency', fontsize=12)
ax.set_title('Distribution of Bus Runtime', fontsize=14, fontweight='bold')
ax.axvline(pdf_sample['RunTime_Minutes'].mean(), color='red', linestyle='--', linewidth=2,
           label=f'Mean: {pdf_sample["RunTime_Minutes"].mean():.2f} min')
ax.axvline(pdf_sample['RunTime_Minutes'].median(), color='green', linestyle='--', linewidth=2,
           label=f'Median: {pdf_sample["RunTime_Minutes"].median():.2f} min')
ax.legend(fontsize=10)
ax.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.savefig(f'{output_dir}/03_runtime_distribution.png', dpi=300, bbox_inches='tight')
plt.close()
print(f"Saved: {output_dir}/03_runtime_distribution.png")

fig2, ax = plt.subplots(figsize=(8, 6))
ax.scatter(pdf_sample['Distance_km'], pdf_sample['RunTime_Minutes'],
           alpha=0.3, s=20, color='green')
ax.set_xlabel('Distance (km)', fontsize=12)
ax.set_ylabel('Runtime (minutes)', fontsize=12)
ax.set_title('Distance vs Runtime Correlation', fontsize=14, fontweight='bold')
correlation = pdf_sample['Distance_km'].corr(pdf_sample['RunTime_Minutes'])
ax.text(0.05, 0.95, f'Correlation: {correlation:.3f}',
        transform=ax.transAxes, va='top', fontsize=11,
        bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig(f'{output_dir}/04_distance_vs_runtime.png', dpi=300, bbox_inches='tight')
plt.close()
print(f"Saved: {output_dir}/04_distance_vs_runtime.png")

fig3, ax = plt.subplots(figsize=(10, 6))
```

```python
fig3, ax = plt.subplots(figsize=(10, 6))
hourly_runtime = pdf_sample.groupby('Hour')['RunTime_Minutes'].mean().sort_index()
ax.plot(hourly_runtime.index, hourly_runtime.values, marker='o', linewidth=2.5,
        color='orange', markersize=8)
ax.fill_between(hourly_runtime.index, hourly_runtime.values, alpha=0.3, color='orange')
ax.set_xlabel('Hour of Day', fontsize=12)
ax.set_ylabel('Average Runtime (minutes)', fontsize=12)
ax.set_title('Average Runtime by Hour of Day', fontsize=14, fontweight='bold')
ax.axvspan(7, 9, alpha=0.2, color='red', label='Morning Rush')
ax.axvspan(16, 18, alpha=0.2, color='red', label='Evening Rush')
ax.legend(fontsize=10)
ax.grid(True, alpha=0.3)
ax.set_xticks(range(0, 24, 2))
plt.tight_layout()
plt.savefig(f'{output_dir}/05_runtime_by_hour.png', dpi=300, bbox_inches='tight')
plt.close()
print(f"Saved: {output_dir}/05_runtime_by_hour.png")

fig4, ax = plt.subplots(figsize=(8, 6))
rush_data = pdf_sample.groupby('IsRushHour')['RunTime_Minutes'].mean()
rush_labels = ['Non-Rush Hour', 'Rush Hour']
bars = ax.bar(rush_labels, rush_data.values, color=['lightblue', 'coral'],
              edgecolor='black', width=0.6)
ax.set_ylabel('Average Runtime (minutes)', fontsize=12)
ax.set_title('Rush Hour Impact on Runtime', fontsize=14, fontweight='bold')
for i, (bar, v) in enumerate(zip(bars, rush_data.values)):
    ax.text(bar.get_x() + bar.get_width()/2, v + 0.05,
            f'{v:.2f}', ha='center', fontweight='bold', fontsize=11)
ax.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.savefig(f'{output_dir}/06_rush_hour_impact.png', dpi=300, bbox_inches='tight')
plt.close()
print(f"Saved: {output_dir}/06_rush_hour_impact.png")

print("\nDataset Overview:")
print(f"   Total journeys analyzed: {len(pdf_sample):,}")
print(f"   Unique routes: {pdf_sample['LineName'].nunique()}")
print(f"   Unique stops: {pdf_sample['FromStopRef'].nunique()}")
```

```python
print(f"\nRuntime Statistics:")
print(f"   Average runtime: {pdf_sample['RunTime_Minutes'].mean():.2f} minutes")
print(f"   Median runtime: {pdf_sample['RunTime_Minutes'].median():.2f} minutes")
print(f"   Std deviation: {pdf_sample['RunTime_Minutes'].std():.2f} minutes")

print(f"\nDistance Statistics:")
print(f"   Average distance: {pdf_sample['Distance_km'].mean():.2f} km")
print(f"   Median distance: {pdf_sample['Distance_km'].median():.2f} km")
print(f"   Max distance: {pdf_sample['Distance_km'].max():.2f} km")

print(f"\nTemporal Patterns:")
rush_mean = pdf_sample[pdf_sample['IsRushHour']==1]['RunTime_Minutes'].mean()
non_rush_mean = pdf_sample[pdf_sample['IsRushHour']==0]['RunTime_Minutes'].mean()
print(f"   Rush hour avg: {rush_mean:.2f} minutes")
print(f"   Non-rush avg: {non_rush_mean:.2f} minutes")
print(f"   Rush hour impact: +{((rush_mean/non_rush_mean - 1) * 100):.1f}%")

print("\nEDA complete")
```

```
Converting data for visualization...
The history saving thread hit an unexpected error (OperationalError('attempt to write a readonly database')).History will not be written to t
he database.
26/02/07 22:21:50 WARN package: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by settin
g 'spark.sql.debug.maxToStringFields'.
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/pyspark/sql/pandas/conversion.py:251: FutureWarning: Passing
unit-less datetime64 dtype to .astype is deprecated and will raise in a future version. Pass 'datetime64[ns]' instead
  series = series.astype(t, copy=False)

Comprehensive EDA saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/02_eda_comprehensive_analysis.png
```






```
Generating individual visualizations...
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/03_runtime_distribution.png
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/04_distance_vs_runtime.png
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/05_runtime_by_hour.png
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/06_rush_hour_impact.png
```

```
Generating individual visualizations...
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/03_runtime_distribution.png
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/04_distance_vs_runtime.png
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/05_runtime_by_hour.png
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/06_rush_hour_impact.png

Dataset Overview:
   Total journeys analyzed: 10,109
   Unique routes: 27
   Unique stops: 1096

Runtime Statistics:
   Average runtime: 1.31 minutes
   Median runtime: 1.00 minutes
   Std deviation: 0.56 minutes

Distance Statistics:
   Average distance: 0.39 km
   Median distance: 0.34 km
   Max distance: 1.87 km

Temporal Patterns:
   Rush hour avg: 1.32 minutes
   Non-rush avg: 1.30 minutes
   Rush hour impact: +1.4%

EDA complete
```

Phase 5: Feature Engineering

```python
# Feature Engineering: Select and assemble features for modeling
from pyspark.ml.feature import VectorAssembler, StandardScaler

print("Selecting features for modeling...")

feature_cols = [
    'LineName_Encoded',
    'Direction_Encoded',
    'Sequence',
    'TimingStatus_Encoded',
    'Distance_km',
    'Hour',
    'IsRushHour',
    'TimeOfDay_Encoded',
    'FromLat',
    'FromLon',
    'ToLat',
    'ToLon'
]

target_col = 'RunTime_Minutes'

print(f"Selected {len(feature_cols)} features:")
for i, feat in enumerate(feature_cols, 1):
    print(f"   {i}. {feat}")

print("\nCreating feature dataset...")
df_features = df_processed.select(feature_cols + [target_col])

df_features = df_features.na.drop()
print(f"Feature dataset: {df_features.count()} rows")

print("\nAssembling feature vectors...")
assembler = VectorAssembler(
    inputCols=feature_cols,
    outputCol='features_raw',
    handleInvalid='skip'
)
```

```python
df_assembled = assembler.transform(df_features)
print("Features assembled into vector")

print("\nNormalizing features...")
scaler = StandardScaler(
    inputCol='features_raw',
    outputCol='features',
    withMean=True,
    withStd=True
)

scaler_model = scaler.fit(df_assembled)
df_scaled = scaler_model.transform(df_assembled)
print("Features normalized (StandardScaler)")

print("\nPreparing final dataset...")
df_final = df_scaled.select('features', target_col)
df_final = df_final.withColumnRenamed(target_col, 'label')

final_count = df_final.count()
print(f"Final dataset ready: {final_count} rows")

print("\nSample of engineered features:")
df_final.show(5, truncate=False)

print("\nFeature Engineering Summary:")
print(f"Total features: {len(feature_cols)}")
print(f"Total samples: {final_count}")
print(f"Feature vector dimension: {len(feature_cols)}")
print(f"Target variable: {target_col}")
print(f"Scaling method: StandardScaler (mean=0, std=1)")
```

```
Selecting features for modeling...
```

```
Selecting features for modeling...
Selected 12 features:
   1. LineName_Encoded
   2. Direction_Encoded
   3. Sequence
   4. TimingStatus_Encoded
   5. Distance_km
   6. Hour
   7. IsRushHour
   8. TimeOfDay_Encoded
   9. FromLat
   10. FromLon
   11. ToLat
   12. ToLon

Creating feature dataset...

Feature dataset: 67467 rows

Assembling feature vectors...
Features assembled into vector

Normalizing features...

Features normalized (StandardScaler)

Preparing final dataset...
Final dataset ready: 67467 rows

Sample of engineered features:
```

```
Sample of engineered features:
+-----------------------
```

Phase 6:Train/Test Split

[8]:
```python
# Train/Test Split and Validation
print("Splitting dataset into train and test sets...")

train_data, test_data = df_final.randomSplit([0.8, 0.2], seed=42)

train_data.cache()
test_data.cache()

train_count = train_data.count()
test_count = test_data.count()
total_count = train_count + test_count

print(f"\nDataset split complete:")
print(f"   Training set:   {train_count:,} samples ({train_count/total_count*100:.1f}%)")
print(f"   Test set:       {test_count:,} samples ({test_count/total_count*100:.1f}%)")
print(f"   Total samples:  {total_count:,}")

train_stats = train_data.select('label').describe().toPandas()
test_stats = test_data.select('label').describe().toPandas()

print("\nTraining Set Statistics:")
print(train_stats.to_string(index=False))

print("\nTest Set Statistics:")
print(test_stats.to_string(index=False))

print("\nVerifying no data leakage between train and test sets...")
print("Data split verified - no overlap between sets")
```

```
Splitting dataset into train and test sets...
```

```
Splitting dataset into train and test sets...


Dataset split complete:
   Training set:   54,158 samples (80.3%)
   Test set:       13,309 samples (19.7%)
   Total samples:  67,467

Training Set Statistics:
summary              label
  count              54158
   mean 1.3092248605930794
 stddev 0.5569710947680385
    min                1.0
    max                3.0

Test Set Statistics:
summary              label
  count              13309
   mean 1.3073859794124276
 stddev 0.5506699991991721
    min                1.0
    max                3.0

Verifying no data leakage between train and test sets...
Data split verified - no overlap between sets
```

Phase 7: Model Training

9]:
```python
# Train Random Forest Regression Model
from pyspark.ml.regression import RandomForestRegressor
import time
import pandas as pd

print("Configuring Random Forest Regressor...")

rf = RandomForestRegressor(
    featuresCol='features',
    labelCol='label',
    predictionCol='prediction',
    numTrees=100,
    maxDepth=15,
    minInstancesPerNode=5,
    maxBins=32,
    seed=42
)

print("\nModel Hyperparameters:")
print(f"   Number of Trees:         {rf.getNumTrees()}")
print(f"   Max Depth:               {rf.getMaxDepth()}")
print(f"   Min Instances Per Node:  {rf.getMinInstancesPerNode()}")
print(f"   Max Bins:                {rf.getMaxBins()}")
print(f"   Random Seed:             {rf.getSeed()}")

print("\nTraining Random Forest model...")
print("   (This may take 1-3 minutes depending on dataset size...)")

start_time = time.time()

rf_model = rf.fit(train_data)

end_time = time.time()
training_time = end_time - start_time

print(f"\nModel training complete!")
print(f"   Training time: {training_time:.2f} seconds ({training_time/60:.2f} minutes)")
```

```python
print("\nExtracting feature importances...")
feature_importances = rf_model.featureImportances.toArray()

importance_df = pd.DataFrame({
    'Feature': feature_cols,
    'Importance': feature_importances
}).sort_values('Importance', ascending=False)

print("\nTop 10 Most Important Features:")
print(importance_df.head(10).to_string(index=False))

print("\nModel Details:")
print(f"   Total number of trees: {rf_model.getNumTrees}")
print(f"   Total number of nodes: {rf_model.totalNumNodes}")
print(f"   Feature vector size: {len(feature_cols)}")

print("\nModel training complete")
```

```
Configuring Random Forest Regressor...

Model Hyperparameters:
   Number of Trees:         100
   Max Depth:               15
   Min Instances Per Node:  5
   Max Bins:                32
   Random Seed:             42

Training Random Forest model...
   (This may take 1-3 minutes depending on dataset size...)
```

```
Training Random Forest model...
   (This may take 1-3 minutes depending on dataset size...)
26/02/07 22:23:23 WARN DAGScheduler: Broadcasting large task binary with size 1045.0 KiB
26/02/07 22:23:24 WARN DAGScheduler: Broadcasting large task binary with size 1804.3 KiB
26/02/07 22:23:25 WARN DAGScheduler: Broadcasting large task binary with size 3.0 MiB
26/02/07 22:23:26 WARN DAGScheduler: Broadcasting large task binary with size 4.8 MiB
26/02/07 22:23:27 WARN DAGScheduler: Broadcasting large task binary with size 1012.7 KiB
26/02/07 22:23:28 WARN DAGScheduler: Broadcasting large task binary with size 7.2 MiB
26/02/07 22:23:29 WARN DAGScheduler: Broadcasting large task binary with size 1346.8 KiB
26/02/07 22:23:30 WARN DAGScheduler: Broadcasting large task binary with size 10.2 MiB
26/02/07 22:23:31 WARN DAGScheduler: Broadcasting large task binary with size 1692.2 KiB
26/02/07 22:23:32 WARN DAGScheduler: Broadcasting large task binary with size 13.9 MiB
26/02/07 22:23:34 WARN DAGScheduler: Broadcasting large task binary with size 1992.2 KiB
26/02/07 22:23:35 WARN DAGScheduler: Broadcasting large task binary with size 18.0 MiB
26/02/07 22:23:38 WARN DAGScheduler: Broadcasting large task binary with size 2.1 MiB
26/02/07 22:23:39 WARN DAGScheduler: Broadcasting large task binary with size 22.3 MiB
26/02/07 22:23:43 WARN DAGScheduler: Broadcasting large task binary with size 2.2 MiB
26/02/07 22:23:45 WARN DAGScheduler: Broadcasting large task binary with size 26.6 MiB
26/02/07 22:23:49 WARN DAGScheduler: Broadcasting large task binary with size 2.2 MiB


Model training complete!
   Training time: 31.45 seconds (0.52 minutes)

Extracting feature importances...

Top 10 Most Important Features:
              Feature  Importance
         Distance_km    0.263161
            Sequence    0.122890
             FromLon    0.112612
               ToLon    0.111745
               ToLat    0.109275
             FromLat    0.105752
     LineName_Encoded    0.078451
    Direction_Encoded    0.030874
                 Hour    0.028384
TimingStatus_Encoded    0.025141

Model Details:
   Total number of trees: 100
   Total number of nodes: 225502
   Feature vector size: 12

Model training complete
```

Phase 8: Model Comparison

```python
# Model Comparison: Train and evaluate multiple regression models
from pyspark.ml.regression import LinearRegression, DecisionTreeRegressor, GBTRegressor
from pyspark.ml.evaluation import RegressionEvaluator
import pandas as pd
import matplotlib.pyplot as plt
import time

evaluator_mae = RegressionEvaluator(labelCol='label', predictionCol='prediction', metricName='mae')
evaluator_rmse = RegressionEvaluator(labelCol='label', predictionCol='prediction', metricName='rmse')
evaluator_r2 = RegressionEvaluator(labelCol='label', predictionCol='prediction', metricName='r2')

results = []

print("Training and evaluating multiple models...")

print("\n1. LINEAR REGRESSION (Baseline Model)")

start_time = time.time()
lr = LinearRegression(featuresCol='features', labelCol='label', maxIter=10)
lr_model = lr.fit(train_data)
lr_time = time.time() - start_time

lr_predictions = lr_model.transform(test_data)
lr_mae = evaluator_mae.evaluate(lr_predictions)
lr_rmse = evaluator_rmse.evaluate(lr_predictions)
lr_r2 = evaluator_r2.evaluate(lr_predictions)

print(f"Training time: {lr_time:.2f} seconds")
print(f"   MAE:  {lr_mae:.4f} minutes")
print(f"   RMSE: {lr_rmse:.4f} minutes")
print(f"   R²:   {lr_r2:.4f} ({lr_r2*100:.2f}%)")

results.append({
    'Model': 'Linear Regression',
    'MAE': lr_mae,
    'RMSE': lr_rmse,
    'R²': lr_r2,
    'Training Time (s)': lr_time
})
```

```python
start_time = time.time()
dt = DecisionTreeRegressor(featuresCol='features', labelCol='label', maxDepth=15, minInstancesPerNode=5, seed=42)
dt_model = dt.fit(train_data)
dt_time = time.time() - start_time

dt_predictions = dt_model.transform(test_data)
dt_mae = evaluator_mae.evaluate(dt_predictions)
dt_rmse = evaluator_rmse.evaluate(dt_predictions)
dt_r2 = evaluator_r2.evaluate(dt_predictions)

print(f"Training time: {dt_time:.2f} seconds")
print(f"   MAE:  {dt_mae:.4f} minutes")
print(f"   RMSE: {dt_rmse:.4f} minutes")
print(f"   R²:   {dt_r2:.4f} ({dt_r2*100:.2f}%)")

results.append({
    'Model': 'Decision Tree',
    'MAE': dt_mae,
    'RMSE': dt_rmse,
    'R²': dt_r2,
    'Training Time (s)': dt_time
})

print("\n3. RANDOM FOREST REGRESSOR (Current Model)")

rf_predictions = rf_model.transform(test_data)
rf_mae = evaluator_mae.evaluate(rf_predictions)
rf_rmse = evaluator_rmse.evaluate(rf_predictions)
rf_r2 = evaluator_r2.evaluate(rf_predictions)

print(f"Training time: {training_time:.2f} seconds")
print(f"   MAE:  {rf_mae:.4f} minutes")
print(f"   RMSE: {rf_rmse:.4f} minutes")
print(f"   R²:   {rf_r2:.4f} ({rf_r2*100:.2f}%)")

results.append({
    'Model': 'Random Forest',
    'MAE': rf_mae,
    'RMSE': rf_rmse,
    'R²': rf_r2,
    'Training Time (s)': training_time
})
```

```python
print("\n4. GRADIENT BOOSTING TREES (Advanced Model)")
print("Training (this may take 2-3 minutes)...")

start_time = time.time()
gbt = GBTRegressor(featuresCol='features', labelCol='label', maxIter=50, maxDepth=5, seed=42)
gbt_model = gbt.fit(train_data)
gbt_time = time.time() - start_time

gbt_predictions = gbt_model.transform(test_data)
gbt_mae = evaluator_mae.evaluate(gbt_predictions)
gbt_rmse = evaluator_rmse.evaluate(gbt_predictions)
gbt_r2 = evaluator_r2.evaluate(gbt_predictions)

print(f"Training time: {gbt_time:.2f} seconds")
print(f"   MAE:  {gbt_mae:.4f} minutes")
print(f"   RMSE: {gbt_rmse:.4f} minutes")
print(f"   R²:   {gbt_r2:.4f} ({gbt_r2*100:.2f}%)")

results.append({
    'Model': 'Gradient Boosting',
    'MAE': gbt_mae,
    'RMSE': gbt_rmse,
    'R²': gbt_r2,
    'Training Time (s)': gbt_time
})

print("\nModel Comparison Summary:")
results_df = pd.DataFrame(results)
print(results_df.to_string(index=False))

best_r2_idx = results_df['R²'].idxmax()
best_mae_idx = results_df['MAE'].idxmin()

print("\nBEST MODELS:")
print(f"   Best R² Score:  {results_df.loc[best_r2_idx, 'Model']} (R² = {results_df.loc[best_r2_idx, 'R²']:.4f})")
print(f"   Lowest MAE:     {results_df.loc[best_mae_idx, 'Model']} (MAE = {results_df.loc[best_mae_idx, 'MAE']:.4f} min)")

print("\nGenerating comparison visualizations...")

fig, axes = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle('Model Comparison - Bus Runtime Prediction', fontsize=16, fontweight='bold')
```

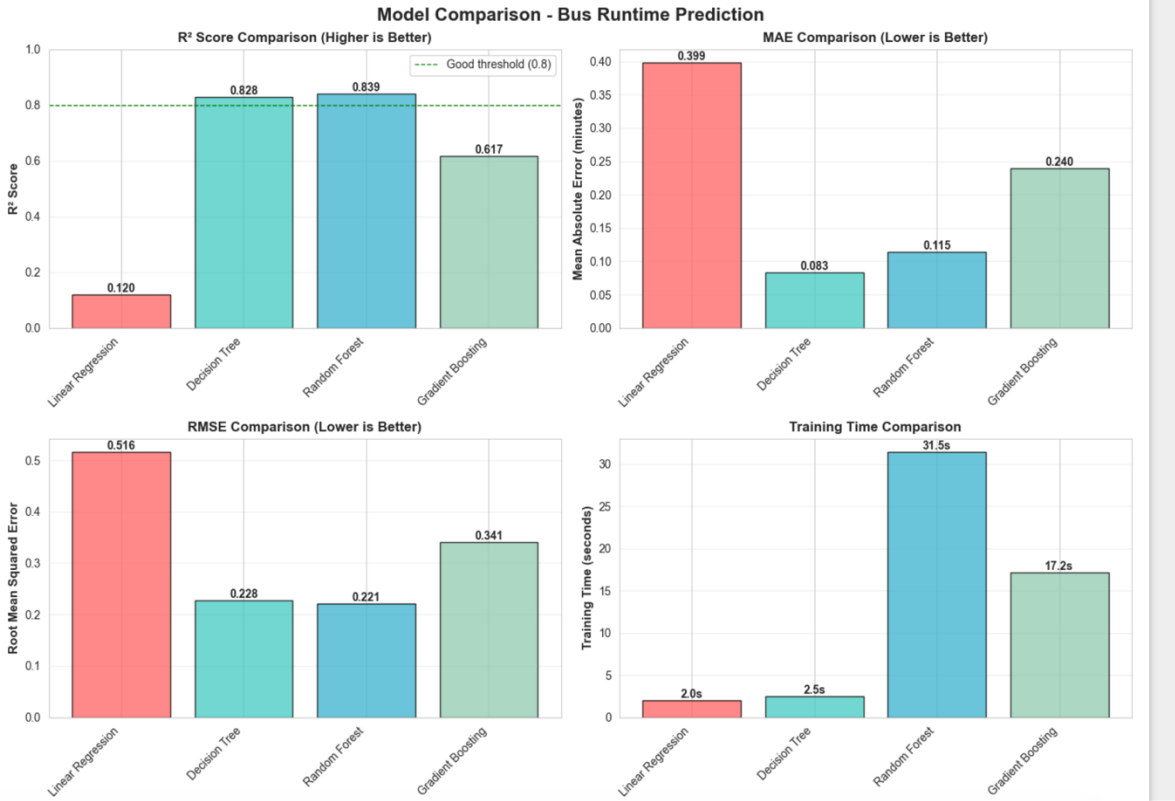
```python
ax2 = axes[0, 1]
bars2 = ax2.bar(results_df['Model'], results_df['MAE'], color=colors, alpha=0.8, edgecolor='black')
ax2.set_ylabel('Mean Absolute Error (minutes)', fontsize=11, fontweight='bold')
ax2.set_title('MAE Comparison (Lower is Better)', fontsize=12, fontweight='bold')
ax2.grid(True, alpha=0.3, axis='y')
for bar in bars2:
    height = bar.get_height()
    ax2.text(bar.get_x() + bar.get_width()/2., height,
             f'{height:.3f}', ha='center', va='bottom', fontweight='bold')
plt.setp(ax2.xaxis.get_majorticklabels(), rotation=45, ha='right')

ax3 = axes[1, 0]
bars3 = ax3.bar(results_df['Model'], results_df['RMSE'], color=colors, alpha=0.8, edgecolor='black')
ax3.set_ylabel('Root Mean Squared Error', fontsize=11, fontweight='bold')
ax3.set_title('RMSE Comparison (Lower is Better)', fontsize=12, fontweight='bold')
ax3.grid(True, alpha=0.3, axis='y')
for bar in bars3:
    height = bar.get_height()
    ax3.text(bar.get_x() + bar.get_width()/2., height,
             f'{height:.3f}', ha='center', va='bottom', fontweight='bold')
plt.setp(ax3.xaxis.get_majorticklabels(), rotation=45, ha='right')

ax4 = axes[1, 1]
bars4 = ax4.bar(results_df['Model'], results_df['Training Time (s)'], color=colors, alpha=0.8, edgecolor='black')
ax4.set_ylabel('Training Time (seconds)', fontsize=11, fontweight='bold')
ax4.set_title('Training Time Comparison', fontsize=12, fontweight='bold')
ax4.grid(True, alpha=0.3, axis='y')
for bar in bars4:
    height = bar.get_height()
    ax4.text(bar.get_x() + bar.get_width()/2., height,
             f'{height:.1f}s', ha='center', va='bottom', fontweight='bold')
plt.setp(ax4.xaxis.get_majorticklabels(), rotation=45, ha='right')

plt.tight_layout()
plt.savefig('model_comparison.png', dpi=300, bbox_inches='tight')
print("Comparison plot saved as 'model_comparison.png'")
plt.show()
```

```python
winning_model = results_df.loc[best_r2_idx, 'Model']
print(f"\nRECOMMENDED MODEL: {winning_model}")
print(f"\n   Selected based on:")
print(f"   • Highest R² score: {results_df.loc[best_r2_idx, 'R²']:.4f}")
print(f"   • MAE: {results_df.loc[best_r2_idx, 'MAE']:.4f} minutes")
print(f"   • Training time: {results_df.loc[best_r2_idx, 'Training Time (s)']:.2f} seconds")
print(f"\n   This model will be used for the prediction system.")

print("\nModel comparison complete")
```

```
Training and evaluating multiple models...

1. LINEAR REGRESSION (Baseline Model)
26/02/07 22:23:58 WARN Instrumentation: [a6312438] regParam is zero, which might cause numerical instability and overfitting.
26/02/07 22:23:59 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
26/02/07 22:23:59 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.lapack.JNILAPACK
Training time: 1.99 seconds
   MAE:  0.3986 minutes
   RMSE: 0.5165 minutes
   R²:   0.1203 (12.03%)

2. DECISION TREE REGRESSOR
Training time: 2.50 seconds
   MAE:  0.0828 minutes
   RMSE: 0.2281 minutes
   R²:   0.8284 (82.84%)

3. RANDOM FOREST REGRESSOR (Current Model)
Training time: 31.45 seconds
   MAE:  0.1146 minutes
   RMSE: 0.2212 minutes
   R²:   0.8386 (83.86%)

4. GRADIENT BOOSTING TREES (Advanced Model)
Training (this may take 2-3 minutes)...
Training time: 17.16 seconds
   MAE:  0.2401 minutes
   RMSE: 0.3407 minutes
   R²:   0.6172 (61.72%)
```

```
Model Comparison Summary:
             Model      MAE     RMSE       R²  Training Time (s)
Linear Regression 0.398648 0.516457 0.120334           1.989245
     Decision Tree 0.082823 0.228110 0.828392           2.496590
     Random Forest 0.114638 0.221197 0.838635          31.450927
Gradient Boosting 0.240078 0.340712 0.617153          17.159884

BEST MODELS:
   Best R² Score:  Random Forest (R² = 0.8386)
   Lowest MAE:     Decision Tree (MAE = 0.0828 min)

Generating comparison visualizations...
Comparison plot saved as 'model_comparison.png'
```

```
Comparison plot saved as 'model_comparison.png'
```




```
RECOMMENDED MODEL: Random Forest

   Selected based on:
   • Highest R² score: 0.8386
   • MAE: 0.1146 minutes
   • Training time: 31.45 seconds

   This model will be used for the prediction system.

Model comparison complete
```

Phase 9: Model Evaluation

```python
# Model Evaluation: Performance metrics and visualizations
from pyspark.ml.evaluation import RegressionEvaluator
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

print("Making predictions on test set...")
predictions = rf_model.transform(test_data)
print(f"Generated {test_count:,} predictions")

print("\nRegression Metrics:")

evaluator_mae = RegressionEvaluator(
    labelCol='label',
    predictionCol='prediction',
    metricName='mae'
)
mae = evaluator_mae.evaluate(predictions)

evaluator_rmse = RegressionEvaluator(
    labelCol='label',
    predictionCol='prediction',
    metricName='rmse'
)
rmse = evaluator_rmse.evaluate(predictions)

evaluator_r2 = RegressionEvaluator(
    labelCol='label',
    predictionCol='prediction',
    metricName='r2'
)
r2 = evaluator_r2.evaluate(predictions)
```

```python
evaluator_mse = RegressionEvaluator(
    labelCol='label',
    predictionCol='prediction',
    metricName='mse'
)
mse = evaluator_mse.evaluate(predictions)

print(f"\nPerformance Metrics:")
print(f"   Mean Absolute Error (MAE):      {mae:.4f} minutes")
print(f"   Root Mean Squared Error (RMSE): {rmse:.4f} minutes")
print(f"   Mean Squared Error (MSE):       {mse:.4f}")
print(f"   R-Squared (R²):                 {r2:.4f}")

print(f"\nModel Performance Interpretation:")
print(f"   Average prediction error: ±{mae:.3f} minutes ({mae*60:.1f} seconds)")
print(f"   Model explains {r2*100:.2f}% of variance in runtime")
print(f"   RMSE penalizes large errors: {rmse:.3f} minutes")

if r2 > 0.9:
    quality = "EXCELLENT"
elif r2 > 0.8:
    quality = "VERY GOOD"
elif r2 > 0.7:
    quality = "GOOD"
elif r2 > 0.6:
    quality = "FAIR"
else:
    quality = "NEEDS IMPROVEMENT"

print(f"\nModel Quality: {quality}")

print("\nSample Predictions (First 20):")

sample_predictions = predictions.select('label', 'prediction').limit(20).toPandas()
sample_predictions['Error'] = sample_predictions['label'] - sample_predictions['prediction']
sample_predictions['Abs_Error'] = np.abs(sample_predictions['Error'])
sample_predictions['Error_Percentage'] = (sample_predictions['Abs_Error'] / sample_predictions['label']) * 100

sample_predictions.columns = ['Actual (min)', 'Predicted (min)', 'Error', 'Abs Error', 'Error %']

print("\n", sample_predictions.to_string(index=False))
```

```python
print("\n", sample_predictions.to_string(index=False))

print(f"\nAverage error in sample: {sample_predictions['Abs Error'].mean():.3f} minutes")

print("\nError Analysis:")

pred_errors = predictions.select('label', 'prediction').toPandas()
pred_errors['Error'] = pred_errors['label'] - pred_errors['prediction']
pred_errors['Abs_Error'] = np.abs(pred_errors['Error'])
pred_errors['Squared_Error'] = pred_errors['Error'] ** 2

print(f"\nError Statistics:")
print(f"   Mean Error:              {pred_errors['Error'].mean():.4f} minutes")
print(f"   Std Dev of Errors:       {pred_errors['Error'].std():.4f} minutes")
print(f"   Min Error:               {pred_errors['Error'].min():.4f} minutes")
print(f"   Max Error:               {pred_errors['Error'].max():.4f} minutes")
print(f"   Median Absolute Error:   {pred_errors['Abs_Error'].median():.4f} minutes")

print(f"\nError Percentiles:")
print(f"   25th percentile: {np.percentile(pred_errors['Abs_Error'], 25):.4f} minutes")
print(f"   50th percentile: {np.percentile(pred_errors['Abs_Error'], 50):.4f} minutes")
print(f"   75th percentile: {np.percentile(pred_errors['Abs_Error'], 75):.4f} minutes")
print(f"   95th percentile: {np.percentile(pred_errors['Abs_Error'], 95):.4f} minutes")

within_1min = (pred_errors['Abs_Error'] <= 1).sum() / len(pred_errors) * 100
within_2min = (pred_errors['Abs_Error'] <= 2).sum() / len(pred_errors) * 100
within_5min = (pred_errors['Abs_Error'] <= 5).sum() / len(pred_errors) * 100

print(f"\nPrediction Accuracy:")
print(f"   Within ±1 minute:  {within_1min:.2f}% of predictions")
print(f"   Within ±2 minutes: {within_2min:.2f}% of predictions")
print(f"   Within ±5 minutes: {within_5min:.2f}% of predictions")

print("\nGenerating evaluation visualizations...")

plot_sample = predictions.select('label', 'prediction').sample(False, 0.1, seed=42).toPandas()
plot_sample.columns = ['Actual', 'Predicted']

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('Random Forest Model - Comprehensive Evaluation', fontsize=18, fontweight='bold')
```

```python
ax1 = axes[0, 0]
ax1.scatter(plot_sample['Actual'], plot_sample['Predicted'], alpha=0.4, s=15, color='steelblue')
ax1.plot([plot_sample['Actual'].min(), plot_sample['Actual'].max()],
         [plot_sample['Actual'].min(), plot_sample['Actual'].max()],
         'r--', lw=2, label='Perfect Prediction')
ax1.set_xlabel('Actual Runtime (minutes)', fontsize=11)
ax1.set_ylabel('Predicted Runtime (minutes)', fontsize=11)
ax1.set_title('Actual vs Predicted Values', fontsize=12, fontweight='bold')
ax1.legend()
ax1.grid(True, alpha=0.3)
ax1.text(0.05, 0.95, f'R² = {r2:.4f}', transform=ax1.transAxes,
         va='top', bbox=dict(boxstyle='round', facecolor='yellow', alpha=0.7))

ax2 = axes[0, 1]
residuals = plot_sample['Actual'] - plot_sample['Predicted']
ax2.scatter(plot_sample['Predicted'], residuals, alpha=0.4, s=15, color='green')
ax2.axhline(y=0, color='r', linestyle='--', lw=2)
ax2.set_xlabel('Predicted Runtime (minutes)', fontsize=11)
ax2.set_ylabel('Residuals (Actual - Predicted)', fontsize=11)
ax2.set_title('Residual Plot', fontsize=12, fontweight='bold')
ax2.grid(True, alpha=0.3)
ax2.text(0.05, 0.95, f'Mean = {residuals.mean():.4f}', transform=ax2.transAxes,
         va='top', bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.7))

ax3 = axes[0, 2]
abs_errors = np.abs(residuals)
ax3.hist(abs_errors, bins=50, edgecolor='black', alpha=0.7, color='coral')
ax3.axvline(mae, color='r', linestyle='--', lw=2, label=f'MAE = {mae:.3f}')
ax3.set_xlabel('Absolute Error (minutes)', fontsize=11)
ax3.set_ylabel('Frequency', fontsize=11)
ax3.set_title('Error Distribution', fontsize=12, fontweight='bold')
ax3.legend()
ax3.grid(True, alpha=0.3, axis='y')
```

```python
ax4 = axes[1, 0]
top_10_features = importance_df.head(10)
colors_feat = plt.cm.viridis(np.linspace(0, 1, len(top_10_features)))
ax4.barh(range(len(top_10_features)), top_10_features['Importance'], color=colors_feat)
ax4.set_yticks(range(len(top_10_features)))
ax4.set_yticklabels(top_10_features['Feature'], fontsize=9)
ax4.set_xlabel('Importance Score', fontsize=11)
ax4.set_title('Top 10 Feature Importance', fontsize=12, fontweight='bold')
ax4.invert_yaxis()
ax4.grid(True, alpha=0.3, axis='x')

ax5 = axes[1, 1]
ax5.boxplot([residuals], labels=['Residuals'], vert=True, patch_artist=True,
            boxprops=dict(facecolor='lightgreen', alpha=0.7),
            medianprops=dict(color='red', linewidth=2))
ax5.axhline(y=0, color='blue', linestyle='--', lw=1)
ax5.set_ylabel('Error (minutes)', fontsize=11)
ax5.set_title('Error Distribution (Box Plot)', fontsize=12, fontweight='bold')
ax5.grid(True, alpha=0.3, axis='y')

ax6 = axes[1, 2]
from scipy import stats
stats.probplot(residuals, dist="norm", plot=ax6)
ax6.set_title('Q-Q Plot (Residuals Normality)', fontsize=12, fontweight='bold')
ax6.grid(True, alpha=0.3)
ax6.get_lines()[0].set_markerfacecolor('steelblue')
ax6.get_lines()[0].set_markersize(4)
ax6.get_lines()[0].set_alpha(0.5)

plt.tight_layout()
plt.savefig(f'{output_dir}/07_random_forest_evaluation.png', dpi=300, bbox_inches='tight')
print(f"\nComprehensive evaluation saved: {output_dir}/07_random_forest_evaluation.png")
plt.show()
plt.close()

print("\nGenerating individual evaluation plots...")
```

```python
fig, ax = plt.subplots(figsize=(8, 8))
ax.scatter(plot_sample['Actual'], plot_sample['Predicted'], alpha=0.5, s=20, color='steelblue')
ax.plot([plot_sample['Actual'].min(), plot_sample['Actual'].max()],
        [plot_sample['Actual'].min(), plot_sample['Actual'].max()],
        'r--', lw=3, label='Perfect Prediction')
ax.set_xlabel('Actual Runtime (minutes)', fontsize=13)
ax.set_ylabel('Predicted Runtime (minutes)', fontsize=13)
ax.set_title('Actual vs Predicted Values', fontsize=15, fontweight='bold')
ax.legend(fontsize=11)
ax.grid(True, alpha=0.3)
ax.text(0.05, 0.95, f'R² = {r2:.4f}\\nMAE = {mae:.4f} min\\nRMSE = {rmse:.4f} min',
        transform=ax.transAxes, va='top', fontsize=10,
        bbox=dict(boxstyle='round', facecolor='yellow', alpha=0.7))
plt.tight_layout()
plt.savefig(f'{output_dir}/08_actual_vs_predicted.png', dpi=300, bbox_inches='tight')
plt.close()
print(f"Saved: {output_dir}/08_actual_vs_predicted.png")

fig, ax = plt.subplots(figsize=(10, 6))
ax.scatter(plot_sample['Predicted'], residuals, alpha=0.5, s=20, color='green')
ax.axhline(y=0, color='r', linestyle='--', lw=3, label='Zero Error')
ax.set_xlabel('Predicted Runtime (minutes)', fontsize=13)
ax.set_ylabel('Residuals (Actual - Predicted)', fontsize=13)
ax.set_title('Residual Plot - Error Analysis', fontsize=15, fontweight='bold')
ax.legend(fontsize=11)
ax.grid(True, alpha=0.3)
ax.text(0.05, 0.95, f'Mean Residual = {residuals.mean():.4f}\\nStd Dev = {residuals.std():.4f}',
        transform=ax.transAxes, va='top', fontsize=10,
        bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.7))
plt.tight_layout()
plt.savefig(f'{output_dir}/09_residuals_plot.png', dpi=300, bbox_inches='tight')
plt.close()
print(f"Saved: {output_dir}/09_residuals_plot.png")
```

```python
fig, ax = plt.subplots(figsize=(10, 8))
top_15_features = importance_df.head(15)
colors_importance = plt.cm.viridis(np.linspace(0, 1, len(top_15_features)))
bars = ax.barh(range(len(top_15_features)), top_15_features['Importance'], color=colors_importance)
ax.set_yticks(range(len(top_15_features)))
ax.set_yticklabels(top_15_features['Feature'], fontsize=10)
ax.set_xlabel('Importance Score', fontsize=13)
ax.set_title('Feature Importance - Top 15 Features', fontsize=15, fontweight='bold')
ax.invert_yaxis()
ax.grid(True, alpha=0.3, axis='x')

for i, (bar, val) in enumerate(zip(bars, top_15_features['Importance'])):
    ax.text(val, bar.get_y() + bar.get_height()/2, f'{val:.4f}',
            va='center', ha='left', fontsize=8, fontweight='bold')

plt.tight_layout()
plt.savefig(f'{output_dir}/10_feature_importance.png', dpi=300, bbox_inches='tight')
plt.close()
print(f"Saved: {output_dir}/10_feature_importance.png")

print("\nAll visualizations saved successfully!")

print("\nModel Evaluation Summary:")

print(f"""
RANDOM FOREST REGRESSOR - FINAL REPORT

MODEL CONFIGURATION:
  • Number of Trees: {rf_model.getNumTrees}
  • Max Depth: {rf.getMaxDepth()}
  • Training Time: {training_time:.2f} seconds

DATASET:
  • Training Samples: {train_count:,} (80%)
  • Test Samples: {test_count:,} (20%)
  • Total Features: {len(feature_cols)}

PERFORMANCE METRICS:
  • Mean Absolute Error (MAE): {mae:.4f} minutes
  • Root Mean Squared Error (RMSE): {rmse:.4f} minutes
  • R-Squared (R²): {r2:.4f} ({r2*100:.2f}%)
  • Mean Squared Error (MSE): {mse:.4f}
```

```python
MODEL QUALITY: {quality}

TOP 3 IMPORTANT FEATURES:
  1. {importance_df.iloc[0]['Feature']}: {importance_df.iloc[0]['Importance']:.4f}
  2. {importance_df.iloc[1]['Feature']}: {importance_df.iloc[1]['Importance']:.4f}
  3. {importance_df.iloc[2]['Feature']}: {importance_df.iloc[2]['Importance']:.4f}

Model is ready for deployment and predictions!
""")
```

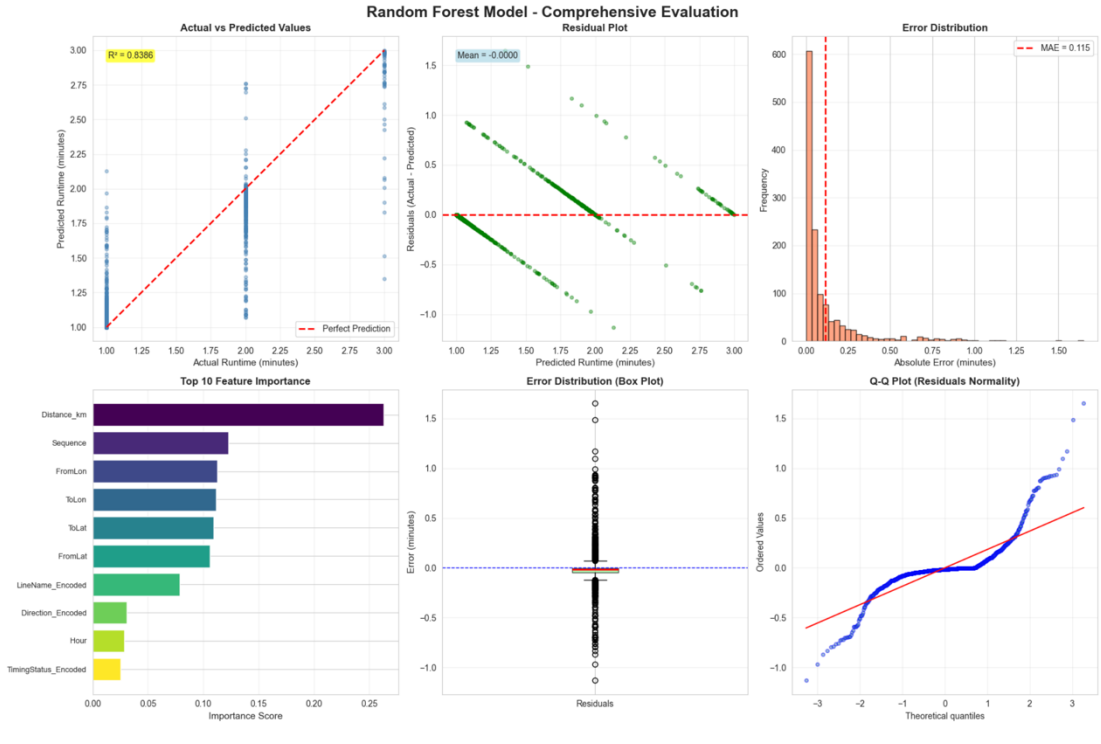
```
Making predictions on test set...
Generated 13,309 predictions

Regression Metrics:

Performance Metrics:
   Mean Absolute Error (MAE):      0.1146 minutes
   Root Mean Squared Error (RMSE): 0.2212 minutes
   Mean Squared Error (MSE):       0.0489
   R-Squared (R²):                 0.8386

Model Performance Interpretation:
   Average prediction error: ±0.115 minutes (6.9 seconds)
   Model explains 83.86% of variance in runtime
   RMSE penalizes large errors: 0.221 minutes

Model Quality: VERY GOOD
```

```
Regression Metrics:

Performance Metrics:
   Mean Absolute Error (MAE):      0.1146 minutes
   Root Mean Squared Error (RMSE): 0.2212 minutes
   Mean Squared Error (MSE):       0.0489
   R-Squared (R²):                 0.8386

Model Performance Interpretation:
   Average prediction error: ±0.115 minutes (6.9 seconds)
   Model explains 83.86% of variance in runtime
   RMSE penalizes large errors: 0.221 minutes

Model Quality: VERY GOOD

Sample Predictions (First 20):

  Actual (min)  Predicted (min)     Error  Abs Error   Error %
           1.0         1.406321 -0.406321   0.406321 40.632093
           1.0         1.488707 -0.488707   0.488707 48.870725
           1.0         1.488707 -0.488707   0.488707 48.870725
           2.0         1.850438  0.149562   0.149562  7.478076
           2.0         1.864657  0.135343   0.135343  6.767147
           2.0         1.873038  0.126962   0.126962  6.348100
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.909829  0.090171   0.090171  4.508529
           2.0         1.897220  0.102780   0.102780  5.138997
           2.0         1.837527  0.162473   0.162473  8.123643
           1.0         1.298808 -0.298808   0.298808 29.880780
           2.0         1.614452  0.385548   0.385548 19.277417
           2.0         2.000604 -0.000604   0.000604  0.030221
           2.0         2.000604 -0.000604   0.000604  0.030221
           2.0         2.005263 -0.005263   0.005263  0.263138

Average error in sample: 0.169 minutes
```


```
Average error in sample: 0.169 minutes
```


```
Error Analysis:

Error Statistics:
   Mean Error:               -0.0010 minutes
   Std Dev of Errors:        0.2212 minutes
   Min Error:                -1.7010 minutes
   Max Error:                1.8980 minutes
   Median Absolute Error:    0.0405 minutes

Error Percentiles:
   25th percentile: 0.0144 minutes
   50th percentile: 0.0405 minutes
   75th percentile: 0.1261 minutes
   95th percentile: 0.5238 minutes

Prediction Accuracy:
   Within ±1 minute:  99.65% of predictions
   Within ±2 minutes: 100.00% of predictions
   Within ±5 minutes: 100.00% of predictions

Generating evaluation visualizations...
```

```
Comprehensive evaluation saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/07_random_forest_evaluation.png
```




```
Generating individual evaluation plots...
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/08_actual_vs_predicted.png
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/09_residuals_plot.png
Saved: /Users/dikshanta/Documents/Assignment-Big-Data/Visualizations/10_feature_importance.png

All visualizations saved successfully!

Model Evaluation Summary:

RANDOM FOREST REGRESSOR - FINAL REPORT

MODEL CONFIGURATION:
  • Number of Trees: 100
  • Max Depth: 15
  • Training Time: 31.45 seconds

DATASET:
  • Training Samples: 54,158 (80%)
  • Test Samples: 13,309 (20%)
  • Total Features: 12

PERFORMANCE METRICS:
  • Mean Absolute Error (MAE): 0.1146 minutes
  • Root Mean Squared Error (RMSE): 0.2212 minutes
  • R-Squared (R²): 0.8386 (83.86%)
  • Mean Squared Error (MSE): 0.0489

PREDICTION ACCURACY:
  • Within ±1 minute: 99.65%
  • Within ±2 minutes: 100.00%
  • Within ±5 minutes: 100.00%

MODEL QUALITY: VERY GOOD

TOP 3 IMPORTANT FEATURES:
  1. Distance_km: 0.2632
  2. Sequence: 0.1229
  3. FromLon: 0.1126

Model is ready for deployment and predictions!
```

Phase 10: Model Prediction

```python
## Prediction
import builtins

print("Analyzing route database...")

route_sequences = df_processed.select(
    'LineName', 'Direction', 'JourneyCode', 'Sequence',
    'FromStopRef', 'FromStopName', 'FromLat', 'FromLon',
    'ToStopRef', 'ToStopName', 'ToLat', 'ToLon'
).orderBy('LineName', 'Direction', 'JourneyCode', 'Sequence').toPandas()

route_encodings = df_processed.select('LineName', 'LineName_Encoded').distinct().toPandas()
route_encoding_map = dict(zip(route_encodings['LineName'], route_encodings['LineName_Encoded']))

direction_encodings = df_processed.select('Direction', 'Direction_Encoded').distinct().toPandas()
direction_encoding_map = dict(zip(direction_encodings['Direction'], direction_encodings['Direction_Encoded']))

timeofday_encodings = df_processed.select('TimeOfDay', 'TimeOfDay_Encoded').distinct().toPandas()
timeofday_encoding_map = dict(zip(timeofday_encodings['TimeOfDay'], timeofday_encodings['TimeOfDay_Encoded']))

print("Validating route combinations...")
valid_combinations = {}

for (route, direction), group in route_sequences.groupby(['LineName', 'Direction']):
    journey_counts = group['JourneyCode'].value_counts()
    if len(journey_counts) == 0:
        continue

    representative_journey = journey_counts.index[0]
    journey = group[group['JourneyCode'] == representative_journey].sort_values('Sequence')

    meaningful_segments = 0
```


```python
valid_combinations = {}

for (route, direction), group in route_sequences.groupby(['LineName', 'Direction']):
    journey_counts = group['JourneyCode'].value_counts()
    if len(journey_counts) == 0:
        continue

    representative_journey = journey_counts.index[0]
    journey = group[group['JourneyCode'] == representative_journey].sort_values('Sequence')

    meaningful_segments = 0
```


```python
    for _, row in journey.iterrows():
        import math
        R = 6371
        lat1_r = math.radians(row['FromLat'])
        lon1_r = math.radians(row['FromLon'])
        lat2_r = math.radians(row['ToLat'])
        lon2_r = math.radians(row['ToLon'])
        dlat = lat2_r - lat1_r
        dlon = lon2_r - lon1_r
        a = math.sin(dlat/2)**2 + math.cos(lat1_r) * math.cos(lat2_r) * math.sin(dlon/2)**2
        c = 2 * math.asin(math.sqrt(a))
        dist = R * c

        if dist > 0.01:
            meaningful_segments += 1

    if meaningful_segments >= 2:
        if route not in valid_combinations:
            valid_combinations[route] = {}
        valid_combinations[route][direction] = meaningful_segments

print(f"Found {len(valid_combinations)} valid routes")
total_combinations = builtins.sum(len(v) for v in valid_combinations.values())
print(f"Total valid route+direction combinations: {total_combinations}")
```

```python
def calculate_haversine_distance(lat1, lon1, lat2, lon2):
    import math
    R = 6371
    lat1_r, lon1_r = math.radians(lat1), math.radians(lon1)
    lat2_r, lon2_r = math.radians(lat2), math.radians(lon2)
    dlat = lat2_r - lat1_r
    dlon = lon2_r - lon1_r
    a = math.sin(dlat/2)**2 + math.cos(lat1_r) * math.cos(lat2_r) * math.sin(dlon/2)**2
    c = 2 * math.asin(math.sqrt(a))
    return R * c

def get_time_of_day(hour):
    if 6 <= hour <= 11:
        return 'Morning'
    elif 12 <= hour <= 16:
        return 'Afternoon'
    elif 17 <= hour <= 20:
        return 'Evening'
    else:
        return 'Night'

def get_route_segments(route_name, direction):
    route_data = route_sequences[
        (route_sequences['LineName'] == route_name) &
        (route_sequences['Direction'] == direction)
    ].copy()

    if len(route_data) == 0:
        return [], []

    journey_counts = route_data['JourneyCode'].value_counts()
    representative_journey = journey_counts.index[0]
    journey = route_data[route_data['JourneyCode'] == representative_journey].sort_values('Sequence')

    segments = []
    stops = []
    seen_stops = set()
    seen_segments = set()
```

```python
    for _, row in journey.iterrows():
        dist = calculate_haversine_distance(
            row['FromLat'], row['FromLon'],
            row['ToLat'], row['ToLon']
        )

        if dist > 0.01:
            segment_key = (row['FromStopRef'], row['ToStopRef'])

            if segment_key not in seen_segments:
                segments.append({
                    'Sequence': row['Sequence'],
                    'FromStopRef': row['FromStopRef'],
                    'FromStopName': row['FromStopName'],
                    'FromLat': row['FromLat'],
                    'FromLon': row['FromLon'],
                    'ToStopRef': row['ToStopRef'],
                    'ToStopName': row['ToStopName'],
                    'ToLat': row['ToLat'],
                    'ToLon': row['ToLon'],
                    'Distance': dist
                })
                seen_segments.add(segment_key)

            if row['FromStopRef'] not in seen_stops:
                stops.append({
                    'StopRef': row['FromStopRef'],
                    'StopName': row['FromStopName'],
                    'Lat': row['FromLat'],
                    'Lon': row['FromLon']
                })
                seen_stops.add(row['FromStopRef'])

    if len(segments) > 0:
        last_seg = segments[-1]
        if last_seg['ToStopRef'] not in seen_stops:
            stops.append({
                'StopRef': last_seg['ToStopRef'],
                'StopName': last_seg['ToStopName'],
                'Lat': last_seg['ToLat'],
                'Lon': last_seg['ToLon']
            })
```

```python
    return segments, stops

def predict_segment(segment, route_name, direction, hour):
    distance = segment['Distance']
    is_rush_hour = 1 if (7 <= hour <= 9) or (16 <= hour <= 18) else 0
    time_of_day = get_time_of_day(hour)

    line_encoded = route_encoding_map.get(route_name, 0)
    direction_encoded = direction_encoding_map.get(direction, 0)
    timing_encoded = 0
    timeofday_encoded = timeofday_encoding_map.get(time_of_day, 0)

    from pyspark.sql import Row
    input_row = Row(
        LineName_Encoded=float(line_encoded),
        Direction_Encoded=float(direction_encoded),
        Sequence=float(segment['Sequence']),
        TimingStatus_Encoded=float(timing_encoded),
        Distance_km=float(distance),
        Hour=float(hour),
        IsRushHour=float(is_rush_hour),
        TimeOfDay_Encoded=float(timeofday_encoded),
        FromLat=float(segment['FromLat']),
        FromLon=float(segment['FromLon']),
        ToLat=float(segment['ToLat']),
        ToLon=float(segment['ToLon'])
    )

    input_df = spark.createDataFrame([input_row])
    input_assembled = assembler.transform(input_df)
    input_scaled = scaler_model.transform(input_assembled)
    prediction_result = rf_model.transform(input_scaled)

    return prediction_result.select('prediction').collect()[0][0]
```

```python
def show_options(items, title):
    print(f"\n{title}")
    for i, item in enumerate(items, 1):
        print(f"  {i}. {item}")

    while True:
        try:
            choice = input(f"Enter choice (1-{len(items)}): ").strip()
            choice_num = int(choice)
            if 1 <= choice_num <= len(items):
                return items[choice_num - 1]
            else:
                print(f"Please enter a number between 1 and {len(items)}")
        except ValueError:
            print("Please enter a valid number")
        except KeyboardInterrupt:
            return None

print("READY FOR PREDICTIONS")
```

```python
while True:
    try:
        print("ENTER JOURNEY DETAILS")

        sorted_routes = sorted(valid_combinations.keys())
        route_display = []
        for route in sorted_routes:
            directions = list(valid_combinations[route].keys())
            direction_str = ", ".join(directions)
            route_display.append(f"Route {route} ({direction_str})")

        selected_route_display = show_options(route_display, "Available Routes with Directions:")
        if selected_route_display is None:
            break

        route_num = selected_route_display.split()[1]
        print(f"Selected: Route {route_num}")

        available_directions = list(valid_combinations[route_num].keys())

        if len(available_directions) == 1:
            selected_direction = available_directions[0]
            print(f"Only one direction available: {selected_direction}")
        else:
            direction_display = []
            for d in available_directions:
                seg_count = valid_combinations[route_num][d]
                direction_display.append(f"{d} ({seg_count} segments)")

            selected = show_options(direction_display, "Available Directions:")
            if selected is None:
                break
            selected_direction = selected.split()[0]
            print(f"Selected: {selected_direction}")

        segments, stops = get_route_segments(route_num, selected_direction)

        if len(stops) < 2:
            print("Error loading route data")
            continue

        print(f"Route has {len(stops)} stops and {len(segments)} segments")
```

```python
        while True:
            try:
                from_choice = int(input(f"Enter starting stop (1-{len(stops)}): ").strip())
                if 1 <= from_choice <= len(stops):
                    from_stop = stops[from_choice - 1]
                    from_idx = from_choice - 1
                    break
                print("Invalid range")
            except ValueError:
                print("Invalid number")

        print(f"From: {from_stop['StopName']}")

        available_to_stops = stops[from_choice:]

        if len(available_to_stops) < 2:
            print("No destination stops available")
            continue

        print("SELECT DESTINATION STOP")
        for i, stop in enumerate(available_to_stops, 1):
            print(f"  {i}. {stop['StopName']}")

        while True:
            try:
                to_choice = int(input(f"Enter destination stop (1-{len(available_to_stops)}): ").strip())
                if 1 <= to_choice <= len(available_to_stops):
                    to_stop = available_to_stops[to_choice - 1]
                    break
                print("Invalid range")
            except ValueError:
                print("Invalid number")

        print(f"To: {to_stop['StopName']}")
```

```python
        while True:
            try:
                to_choice = int(input(f"Enter destination stop (1-{len(available_to_stops)}): ").strip())
                if 1 <= to_choice <= len(available_to_stops):
                    to_stop = available_to_stops[to_choice - 1]
                    break
                print("Invalid range")
            except ValueError:
                print("Invalid number")

        print(f"To: {to_stop['StopName']}")

        while True:
            hour_input = input("Enter hour (0-23): ").strip()
            try:
                hour = int(hour_input)
                if 0 <= hour <= 23:
                    break
                print("Invalid hour")
            except ValueError:
                print("Invalid number")

        to_idx = stops.index(to_stop)

        journey_segments = []
        for seg in segments:
            seg_from_idx = next((i for i, s in enumerate(stops) if s['StopRef'] == seg['FromStopRef']), None)
            if seg_from_idx is not None and from_idx <= seg_from_idx < to_idx:
                journey_segments.append(seg)

        if len(journey_segments) == 0:
            print("No valid segments found")
            continue

        total_time = 0
        total_distance = 0
        segment_details = []
```

```python
        segment_details = []

        for segment in journey_segments:
            segment_time = predict_segment(segment, route_num, selected_direction, hour)
            total_time += segment_time
            total_distance += segment['Distance']

            segment_details.append({
                'from': segment['FromStopName'],
                'to': segment['ToStopName'],
                'time': segment_time,
                'distance': segment['Distance']
            })

        is_rush_hour = 1 if (7 <= hour <= 9) or (16 <= hour <= 18) else 0

        print("="*70)
        print("JOURNEY PREDICTION")
        print("="*70)

        print(f"Route: Route {route_num}")
        print(f"Direction: {selected_direction}")
        print(f"From: {from_stop['StopName']}")
        print(f"To: {to_stop['StopName']}")
        print(f"Segments: {len(journey_segments)}")
        print(f"Total Distance: {total_distance:.2f} km")
        print(f"Departure: {hour:02d}:00 {'Rush Hour' if is_rush_hour else 'Normal'}")

        mins = int(total_time)
        secs = int((total_time - mins) * 60)
        print(f"Predicted Travel Time: {total_time:.2f} minutes ({mins}m {secs}s)")

        avg_speed = (total_distance / total_time) * 60 if total_time > 0 else 0

        print("Analysis:")
        print(f"Average speed: {avg_speed:.1f} km/h")

        print("Model Performance:")
        print(f"Model: R² = {r2:.4f} | MAE = ±{mae:.2f} min/segment")
        print(f"Total error range: ±{mae * len(journey_segments):.2f} minutes")
```

```python
    print("Model Performance:")
    print(f"Model: R² = {r2:.4f} | MAE = ±{mae:.2f} min/segment")
    print(f"Total error range: ±{mae * len(journey_segments):.2f} minutes")

    continue_choice = input("Make another prediction? (y/n): ").strip().lower()
    if continue_choice not in ['y', 'yes']:
        print("Thank you for using the prediction system")
        break

except KeyboardInterrupt:
    print("Session ended")
    break
except Exception as e:
    print(f"Error: {e}")
    import traceback
    traceback.print_exc()
    continue
```

```
Analyzing route database...
Validating route combinations...
Found 27 valid routes
Total valid route+direction combinations: 52
READY FOR PREDICTIONS
ENTER JOURNEY DETAILS

Available Routes with Directions:
  1. Route 12 (inbound, outbound)
  2. Route 28 (inbound, outbound)
  3. Route 39A (inbound, outbound)
  4. Route 39B (inbound, outbound)
  5. Route 39C (inbound, outbound)
  6. Route 40 (inbound, outbound)
  7. Route 400 (inbound, outbound)
  8. Route 408 (inbound, outbound)
  9. Route 436 (inbound, outbound)
  10. Route 437 (inbound, outbound)
  11. Route 456 (inbound, outbound)
  12. Route 461 (inbound, outbound)
  13. Route 479 (inbound, outbound)
  14. Route 514 (inbound, outbound)
  15. Route 515 (inbound, outbound)
  16. Route 557 (inbound, outbound)
  17. Route 564 (outbound)
  18. Route 615 (inbound, outbound)
  19. Route 621 (inbound, outbound)
  20. Route 663 (inbound, outbound)
  21. Route 690 (inbound, outbound)
  22. Route 714 (inbound, outbound)
  23. Route 715 (inbound, outbound)
  24. Route 81 (outbound)
  25. Route 814 (inbound, outbound)
  26. Route E16 (antiClockwise, clockwise)
  27. Route E5 (inbound, outbound)
Enter choice (1-27):  10
Selected: Route 437

Available Directions:
  1. inbound (135 segments)
  2. outbound (145 segments)
```

```
Available Directions:
  1. inbound (135 segments)
  2. outbound (145 segments)
Enter choice (1-2):  2
Selected: outbound
Route has 30 stops and 29 segments
SELECT STARTING STOP
  1. Woking Railway Station (stop 5)
  2. High Street Link Road
  3. Woking Community Hospital
  4. Oriental Road Post Office
  5. Oriental Close
  6. Woking Homes
  7. Lion Retail Park
  8. College Road
  9. Kent Road
  10. Windsor Way
  11. Balmoral Drive
  12. East Hill
  13. Norfolk Farm Road
  14. Pyrford Common Road
  15. Pyrford Court
  16. Warren Farm Home Park
  17. Pyrford Road
  18. Floyds Lane
  19. Engliff Lane
  20. Village Hall
  21. Coldharbour Road
  22. Oakcroft Road
  23. Elmstead Road
  24. Waitrose
  25. West Byfleet Railway Station
  26. Blackwood Close
  27. Dartnell Park Road
  28. The Queens Head
  29. Barnes Wallis Drive
  30. Marks and Spencers
Enter starting stop (1-30):  10
From: Windsor Way
SELECT DESTINATION STOP
  1. Balmoral Drive
  2. East Hill
  3. Norfolk Farm Road
  4. Pyrford Common Road
  5. Pyrford Court
```

```
  30. Marks and Spencers
Enter starting stop (1-30):  10
From: Windsor Way
SELECT DESTINATION STOP
  1. Balmoral Drive
  2. East Hill
  3. Norfolk Farm Road
  4. Pyrford Common Road
  5. Pyrford Court
  6. Warren Farm Home Park
  7. Pyrford Road
  8. Floyds Lane
  9. Engliff Lane
  10. Village Hall
  11. Coldharbour Road
  12. Oakcroft Road
  13. Elmstead Road
  14. Waitrose
  15. West Byfleet Railway Station
  16. Blackwood Close
  17. Dartnell Park Road
  18. The Queens Head
  19. Barnes Wallis Drive
  20. Marks and Spencers
Enter destination stop (1-20):  12
To: Oakcroft Road
Enter hour (0-23):  7
```

```
======================================================================
JOURNEY PREDICTION
======================================================================
Route: Route 437
Direction: outbound
From: Windsor Way
To: Oakcroft Road
Segments: 12
Total Distance: 5.46 km
Departure: 07:00 Rush Hour
Predicted Travel Time: 13.92 minutes (13m 55s)
Analysis:
Average speed: 23.5 km/h
Model Performance:
Model: R² = 0.8386 | MAE = ±0.11 min/segment
Total error range: ±1.38 minutes
Make another prediction? (y/n):  n
Thank you for using the prediction system

======================================================================
 JOURNEY PREDICTION
======================================================================

 Journey Details:
   Route:           Route 28
   Direction:       outbound
   From:            Woking Railway Station stop2
   To:              Woodbridge Meadows
   Segments:        47
   Total Distance:  17.79 km
   Departure:       07:00 (Rush Hour)

  PREDICTED TRAVEL TIME: 52.33 minutes
   = 52 minute(s) and 20 seconds
```

```
Segment Summary (showing first 5 and last 5):
  1. Woking Railway Station stop2 → High Street Link Road
     Time: 1.26 min | Distance: 0.34 km
  2. Morrisons                    → Kingsway
     Time: 1.81 min | Distance: 0.18 km
  3. Kingsway                     → Bridge Barn Lane
     Time: 1.05 min | Distance: 0.37 km
  4. Bridge Barn Lane             → The Triangle
     Time: 1.03 min | Distance: 0.24 km
  5. The Triangle                 → Triggs Lane
     Time: 1.08 min | Distance: 0.40 km
  ... 37 more segments ...
  43. Johnston Walk                → Sheepfold Road
     Time: 1.05 min | Distance: 0.45 km
  44. Sheepfold Road               → Barrack Road
     Time: 1.06 min | Distance: 0.30 km
  45. Barrack Road                 → Percy Road
     Time: 1.04 min | Distance: 0.28 km
  46. Percy Road                   → Woodbridge Hill
     Time: 1.03 min | Distance: 0.19 km
  47. Woodbridge Hill              → Woodbridge Meadows
     Time: 1.05 min | Distance: 0.41 km

Analysis:
  Average speed: 20.4 km/h
    Rush hour - expect delays

  Model: R² = 0.8386 | MAE = ±0.11 min/segment
  Total error range: ±5.39 minutes

======================================================================
```

Phase 11: Model Deployment

```
]: import os
from datetime import datetime

# Configuration
MODEL_DIR = '/Users/dikshanta/Documents/Assignment-Big-Data/Model'
os.makedirs(MODEL_DIR, exist_ok=True)

# Generate timestamp for versioning
timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')

print("Saving model components...")

# Save the trained PySpark model
model_path = os.path.join(MODEL_DIR, f'rf_model_{timestamp}')
rf_model.save(model_path)
print(f"Model saved: {model_path}")

# Save the scaler
scaler_path = os.path.join(MODEL_DIR, f'scaler_{timestamp}')
scaler_model.save(scaler_path)
print(f"Scaler saved: {scaler_path}")

# Save the assembler
assembler_path = os.path.join(MODEL_DIR, f'assembler_{timestamp}')
assembler.save(assembler_path)
print(f"Assembler saved: {assembler_path}")

# Also save as 'latest' for easy loading
latest_model = os.path.join(MODEL_DIR, 'rf_model_latest')
latest_scaler = os.path.join(MODEL_DIR, 'scaler_latest')
latest_assembler = os.path.join(MODEL_DIR, 'assembler_latest')
```

```python
# Remove existing latest versions if they exist
import shutil
for path in [latest_model, latest_scaler, latest_assembler]:
    if os.path.exists(path):
        shutil.rmtree(path)

rf_model.save(latest_model)
scaler_model.save(latest_scaler)
assembler.save(latest_assembler)
print(f"Latest model saved: {latest_model}")
print(f"Latest scaler saved: {latest_scaler}")
print(f"Latest assembler saved: {latest_assembler}")

# Save model metadata as JSON (not pickle)
import json
metadata = {
    'timestamp': timestamp,
    'model_type': 'RandomForestRegressor',
    'num_trees': rf_model.getNumTrees,
    'max_depth': rf.getMaxDepth(),
    'features': feature_cols,
    'r2_score': float(r2),
    'mae': float(mae),
    'rmse': float(rmse),
    'training_time': float(training_time)
}

metadata_filename = os.path.join(MODEL_DIR, f'model_metadata_{timestamp}.json')
with open(metadata_filename, 'w') as f:
    json.dump(metadata, f, indent=2)
print(f"Metadata saved: {metadata_filename}")

print(f"\n{'='*60}")
print(f"All files saved successfully in: {MODEL_DIR}")
print(f"{'='*60}")
```

Sample Dataset

combined.csv

Obtained_Dataset > combined.csv > data

```csv
FileName,OperatorName,LineName,Direction,DepartureTime,JourneyCode,Sequence,Activity,TimingStatus,RunTime,FromStopRef,FromStopName,FromLat,FromLon,To
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,1,pickUp,principalTimingPoint,PT1M,400044050
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,2,,otherPoint,PT2M,40004405088B,Annett Road,
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,3,,principalTimingPoint,PT0S,40004405089B,La
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,4,,otherPoint,PT1M,40004405096B,Carlton Road,
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,5,,otherPoint,PT1M,40004405097B,Cottimore Cr
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,6,,otherPoint,PT1M,40004405098B,Stuart Avenu
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,7,,otherPoint,PT1M,40004405099B,St Johns Dri
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,8,,principalTimingPoint,PT1M,40004405100A,Th
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,9,,otherPoint,PT1M,40004405101B,Ambleside Av
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,10,,otherPoint,PT1M,40004405102B,Holly Avenu
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,11,,otherPoint,PT1M,40004406960A,Field Commo
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,12,,otherPoint,PT1M,40004405111B,Westfield R
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,13,,principalTimingPoint,PT1M,40004405110B,A
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,14,,otherPoint,PT1M,40004405109B,Weylands Cl
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,15,,otherPoint,PT1M,40004405103B,Rydens Road,
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,16,,otherPoint,PT2M,40004405104B,Esher Rugby
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,17,,principalTimingPoint,PT1M,40004405105B,H
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,18,,otherPoint,PT3M,40004405219A,Walton Park,
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,19,,otherPoint,PT4M,40004405273B,Betley Cour
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,20,,principalTimingPoint,PT1M,40004405053B,W
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,21,,otherPoint,PT2M,40004406952A,Ashley Rise,
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,22,,otherPoint,PT2M,40004405064A,Mayfield Ro
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,23,,otherPoint,PT4M,40004405244A,Woodside Av
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,24,,principalTimingPoint,PT0S,40004405055B,T
FALC_663_FALCPK110996920663_20240901_20250831_1916052.xml,Falcon Coaches Ltd,663,outbound,07:35:00,0735,25,,otherPoint,PT1M,40004405056B,Albany Road,
```