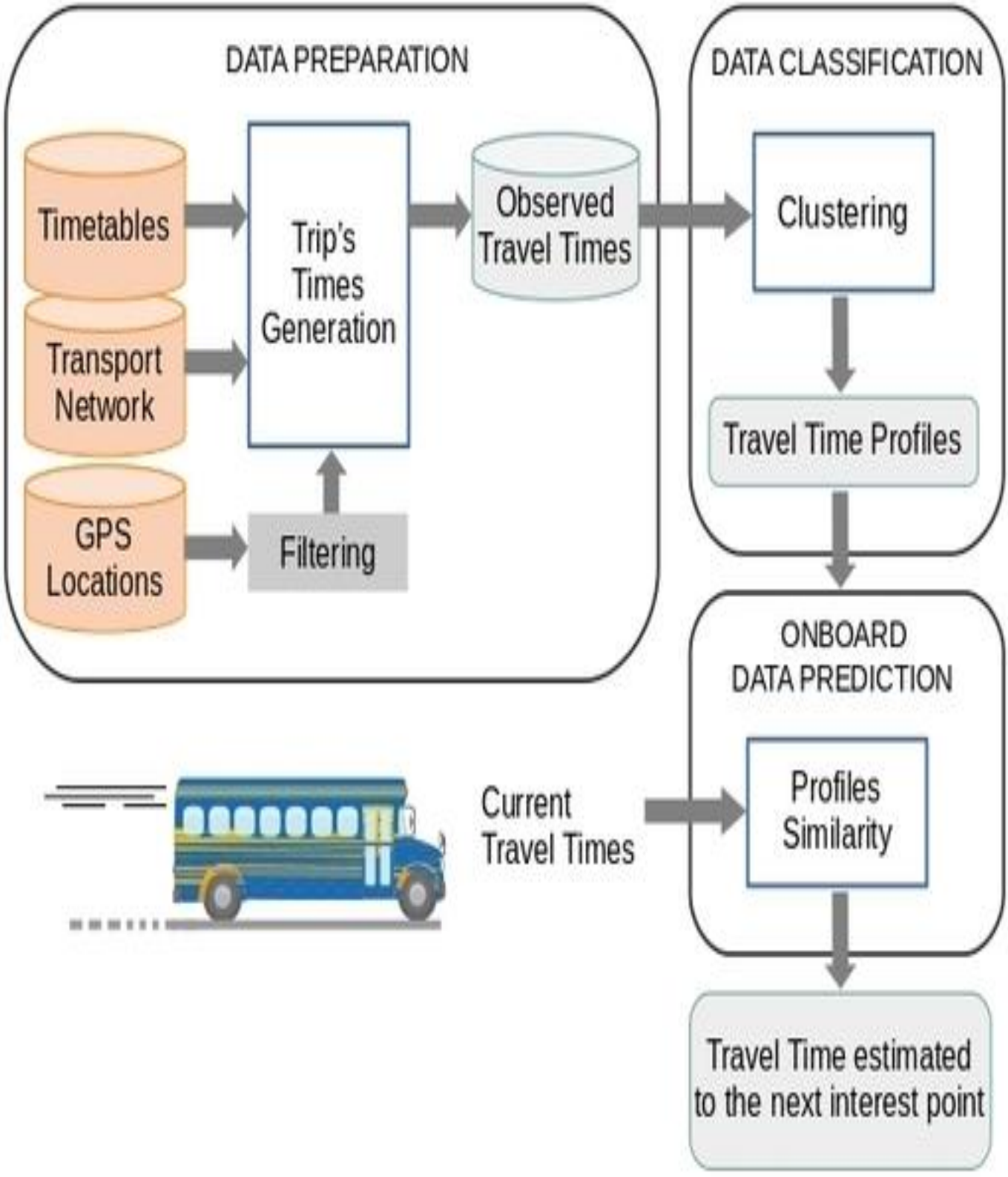
PUBLIC TRANSPORT OPTIMIZATION

PHASE 2 - INNOVATION


DATA PREPARATION
Timetables
Transport Network
GPS Locations
Trip's Times Generation
Filtering
Observed Travel Times
DATA CLASSIFICATION
Clustering
Travel Time Profiles
ONBOARD DATA PREDICTION
Current Travel Times
Profiles Similarity
Travel Time estimated to the next interest point

INTRODUCTION

Public transportation is a vital part of urban life, but delays and unpredictable traffic conditions can make commuting a challenge. Machine learning can help alleviate these issues by predicting arrival times and providing real-time traffic updates to commuters. In this document, we present an approach to address these problems using machine learning.

PROBLEM STATEMENT

The main goal is to predict the estimated arrival time of public transport vehicles (e.g., buses, trams, subways) at various stops along a route, while also considering real-time traffic conditions.

DATA COLLECTION

Historical Transport Data:

Collect historical data on the movement of public transport vehicles, including timestamps, GPS coordinates, and passenger load at different stops.

Traffic Data:

Acquire real-time traffic data from sources like GPS devices, traffic cameras, or third-party APIs, which includes traffic congestion, road closures, and accidents.

DATA PREPROCESSING

Data Cleaning:

Clean the collected data by removing duplicates, handling missing values, and ensuring data consistency.

Feature Engineering:

Create relevant features such as time of day, day of the week, and holiday indicators.

Data Fusion:

Merge transport, traffic, and weather data into a unified dataset for modeling.

MACHINE LEARNING MODELS

Regression Models:

Use regression models (e.g., linear regression, decision trees, or random forests) to predict arrival times based on historical data.

Traffic Prediction:

Utilize machine learning models like convolutional neural networks (CNNs) or recurrent neural networks (RNNs) to predict traffic conditions.

EVALUATION METRICS

To assess the accuracy and performance of the models, consider metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE).

ALGORITHM

Step 1: Data Collection
Gather historical data on routes, including start times, end times, weather conditions, traffic congestion levels, distance, and actual arrival times. Ensure that the dataset is well-structured and includes relevant features.

Step 2: Data Preprocessing
Clean and preprocess the data. This involves handling missing values, encoding categorical variables, and scaling/normalizing numerical features. Ensure that the data is in a suitable format for modeling.

Step 3: Split the data into training and test sets
Split the dataset into training and testing sets, preserving the temporal order. The training set is used to train the model, while the testing set is used to evaluate its performance.

Step 4: Create the random forest regression model
This can be done using a variety of machine learning libraries, such as scikit-learn in Python.

Step 5: Model Training
Train the random forest regression model using the training data. The model will learn patterns and relationships in the data to make predictions.

Step 6: Model Evaluation
Evaluate the model's performance on the testing data using appropriate regression metrics. Common metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE).

Step 7: Evaluate the performance of the model on the test set
This will give you an idea of how well the model will generalize to new data.

Step 8: Deploy the model to production
Once you are satisfied with the performance of the model, you can deploy it to production so that it can be used to predict arrival times in real time.

PYTHON CODE

Data Preprocessing

Preprocess the data by handling missing values, encoding categorical variables, and splitting it into training and testing sets.

Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load your dataset (replace 'data.csv' with a sample dataset)
data = pd.read_csv('data.csv')

# Handle missing values if any
data.fillna(0, inplace=True)

# Encode categorical variables if any
label_encoder = LabelEncoder()
data['categorical_feature'] =
label_encoder.fit_transform(data['categorical_feature'])

# Split the data into training and testing sets
X = data.drop('arrival_time_accuracy', axis=1)
y = data['arrival_time_accuracy']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Model Training (Random Forest Regression)

Train a Random Forest Regression model using the training data.

Code

```
from sklearn.ensemble import RandomForestRegressor

# Define the Random Forest Regressor
rf_regressor = RandomForestRegressor()

# Train the model
rf_regressor.fit(X_train, y_train)
```

Model Evaluation

Evaluate the model's performance on the testing data.

Code

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Make predictions
y_pred = rf_regressor.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print(f'Mean Absolute Error: {mae}')
print(f'Root Mean Squared Error: {rmse}')
print(f'R-squared: {r2}')
```

Feature Importance Analysis

Random Forest provides feature importance scores, allowing you to understand which features contribute the most to the predictions.

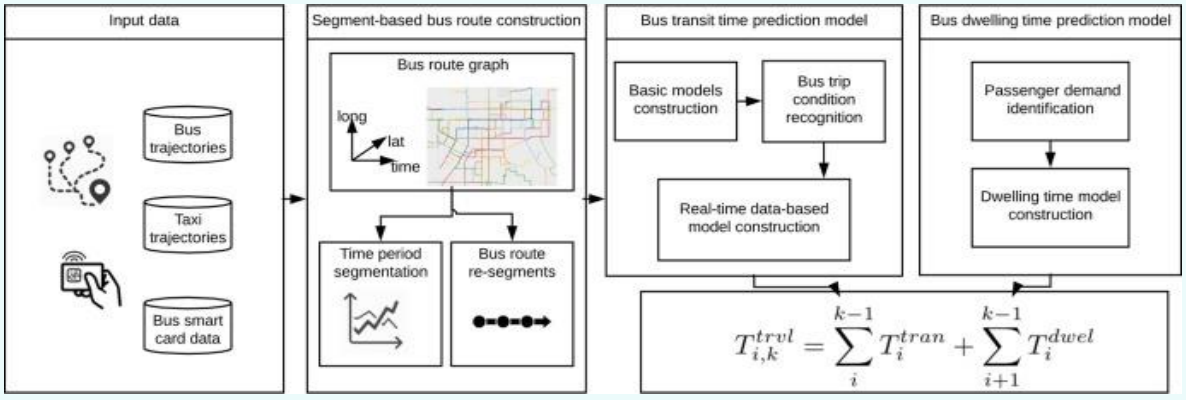
Code

```
import matplotlib.pyplot as plt
import seaborn as sns

# Get feature importances
feature_importances = rf_regressor.feature_importances_

# Create a DataFrame to visualize feature importances
importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance':
eature_importances})
importance_df = importance_df.sort_values(by='Importance',
scending=False)

 # Plot feature importances
 plt.figure(figsize=(10, 6))
 sns.barplot(x='Importance', y='Feature', data=importance_df)
 plt.title('Feature Importances')
 plt.show()
```



CONCLUSION

This document outlined the steps to build a Random Forest Regression model for predicting arrival time accuracy based on historical data and traffic conditions. By following these steps, we can effectively prepare the data, preprocess it, train a Random Forest Regression model, evaluate its performance, and analyze feature importance. The use of Random Forest Regression provides a robust and interpretable solution for this prediction task, offering valuable insights into the factors influencing arrival time accuracy. However, it's essential to customize the code to your specific dataset and continuously refine the model to maintain its accuracy. This predictive model can significantly benefit Public transportation and travelers by improving route planning, enhancing the travel experience, and optimizing resource allocation.