Phase 2

Energy consumption for public transport optimization

Data Collection:

Gather historical data on bus or train arrivals, including timestamps, routes, stops, and real-time traffic conditions. Public APIs or historical data from transit agencies can be valuable sources

```
# You can use APIs or import historical data from a CSV file

Import pandas as pd

# Example using pandas to load data from a CSV file

Data = pd.read_csv('public_transport_data.csv')
```

Data Preprocessing:

Clean and preprocess the data. This may involve handling missing values, encoding categorical features, and normalizing numerical data.

```
# Handle missing values (assuming 'df' is your DataFrame)

Df.dropna(inplace=True)


# Encode categorical features (assuming 'categorical_columns' is a list of categorical column names)

Df = pd.get_dummies(df, columns=categorical_columns)


# Normalize numerical data (e.g., using Min-Max scaling)

From sklearn.preprocessing import MinMaxScaler

Scaler = MinMaxScaler()

Df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```

Feature Engineering:

Extract relevant features that can impact arrival times, such as time of day, day of the week, weather conditions, and traffic congestion. Create a target variable (arrival time delay) by calculating the difference between scheduled and actual arrival times.

```
# Calculate arrival time delay as the target variable

Df['arrival_time_delay'] = df['actual_arrival_time'] – df['scheduled_arrival_time']


# Extract relevant features (e.g., time of day, day of the week)

Df['hour_of_day'] = df['timestamp'].dt.hour
```

```
Df[‘day_of_week’] = df[‘timestamp’].dt.dayofweek
```

Data Splitting

Divide the dataset into training and testing sets. Typically, you’d reserve a portion of the data for model evaluation.

```
 From sklearn.model_selection import train_test_split


X = df.drop([‘arrival_time_delay’], axis=1)  # Features

Y = df[‘arrival_time_delay’]  # Target variable


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Choose Machine Learning Algorithms:

Select machine learning algorithms suitable for regression tasks. Common choices include Linear Regression, Decision Trees, Random Forests, or more advanced methods like Gradient Boosting or Neural Networks

```
# Import the machine learning algorithm you want to use (e.g., Linear Regression)

From sklearn.linear_model import LinearRegression
```

Model Training:

Train the chosen models on the training dataset. Experiment with different algorithms and hyperparameters to find the best-performing model.

```
# Create and train the chosen model (e.g., Linear Regression)

Model = LinearRegression()

Model.fit(X_train, y_train)
```

Model Evaluation

Evaluate model performance using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE) on the testing dataset. Cross-validation can also be helpful for robust evaluation.

```
From sklearn.metrics import mean_absolute_error


# Make predictions on the test data

Y_pred = model.predict(X_test)


# Evaluate model performance using Mean Absolute Error (MAE)

Mae = mean_absolute_error(y_test, y_pred)

Print(f”Mean Absolute Error: {mae}”)
```