TRAFFIC MANAGEMENT USING AI AND ML

BY BATCH 13

CONTEXT:

INTRODUCTION

Traffic management is a critical aspect of urban planning, aiming to ensure the safe, efficient, and smooth flow of traffic. With increasing urbanization and the complexity of modern transportation systems, traditional traffic management approaches are often insufficient to address the growing challenges. This is where **Artificial Intelligence (AI)** and **Machine Learning (ML)** come into play, offering innovative solutions to optimize traffic flow, reduce congestion, and improve safety.

AI and ML enable traffic management systems to process and analyze vast amounts of real-time data from sources like sensors, cameras, GPS, and traffic apps. By leveraging these technologies, traffic control centers can predict traffic patterns, optimize signal timings, detect incidents, and even guide autonomous vehicles.

AIM

The primary aim of using AI and ML in traffic management is to **optimize traffic flow**, **reduce congestion**, **enhance safety**, and **improve the overall efficiency** of transportation systems through intelligent, data-driven solutions.

OBJECTIVES:

Real-time Traffic Optimization: Use AI to dynamically adjust traffic signals and routes based on live traffic data.

Predictive Traffic Analytics: Leverage ML algorithms to forecast traffic congestion and incidents, enabling proactive management.

Incident Detection and Response: Automate the identification of accidents or road disruptions for quicker response times.

Enhancing Road Safety: Use AI to reduce accidents and improve safety for all road users, including pedestrians and cyclists.

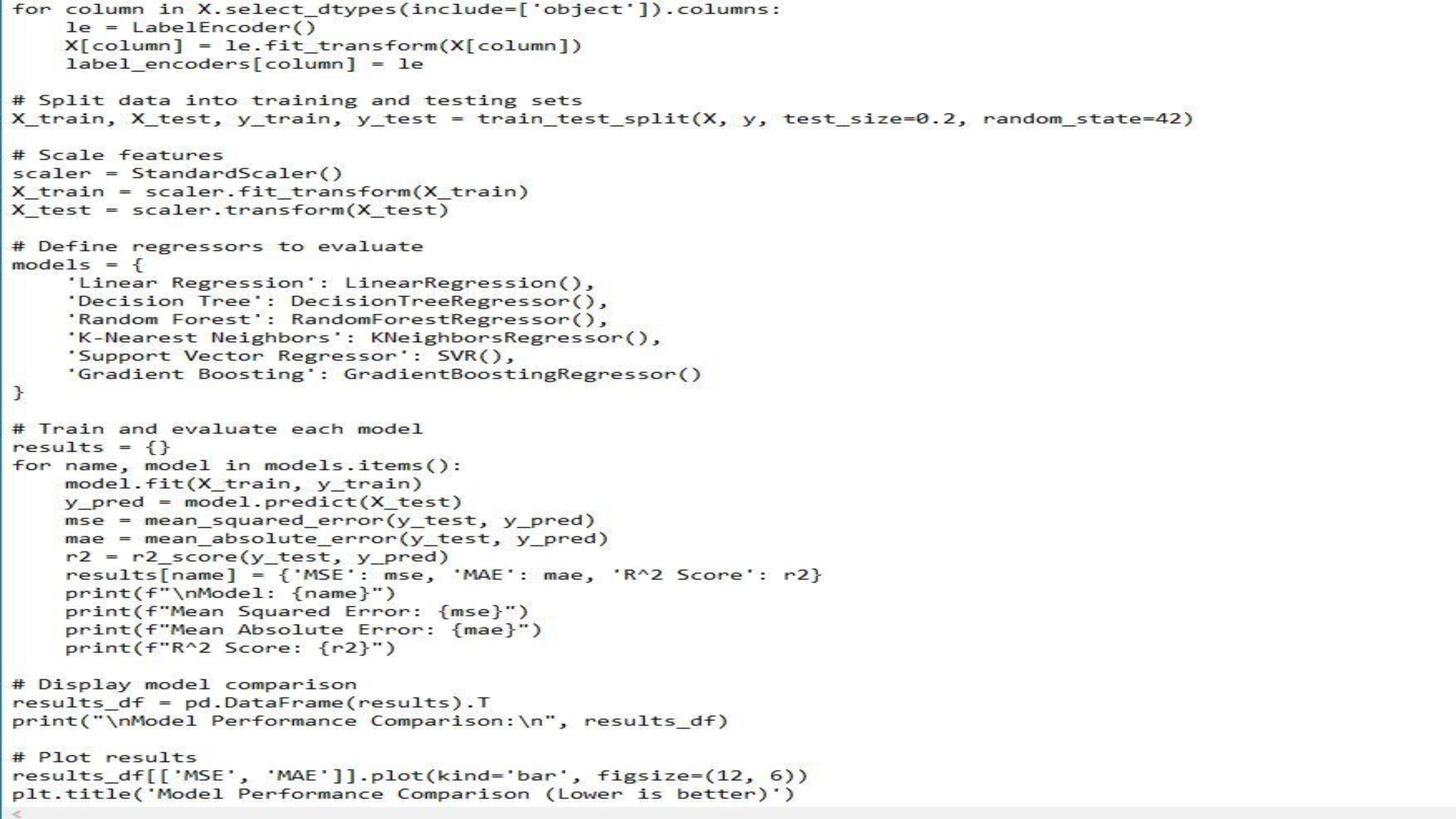
Environmental Impact Reduction: Minimize fuel consumption and emissions by optimizing traffic flow and reducing idle times.

Support Autonomous Vehicles: Integrate AI systems to ensure safe interaction between autonomous and non-autonomous vehicles on the road.

FUNCTIONALITIES OF AI AND ML IN TRAFFIC MANAGEMENT:

1. **Adaptive Traffic Signal Control**: AI adjusts traffic light timings in real-time based on traffic flow to minimize congestion and reduce wait times.
2. **Traffic Prediction**: ML algorithms analyze historical and real-time data to predict traffic patterns, allowing for proactive congestion management and route optimization.
3. **Incident Detection and Response**: AI monitors surveillance cameras and sensors to quickly detect accidents, road obstructions, or other incidents and trigger immediate alerts for faster response.
4. **Automated Traffic Monitoring**: AI systems process data from cameras, sensors, and IoT devices to continuously monitor traffic conditions and ensure smooth traffic flow.
5. **Vehicle and Pedestrian Safety**: AI-driven systems improve safety by detecting potential collisions or unsafe behavior, alerting drivers or pedestrians in real time.
6. **Route Optimization**: AI and ML recommend optimal routes for drivers by analyzing traffic conditions, accidents, and road closures to reduce travel time.
7. **Smart Parking Management**: AI-based systems help find available parking spaces by analyzing traffic and parking data, guiding drivers to open spots.
8. **Traffic Flow Simulation**: ML models simulate and evaluate different traffic scenarios to design more efficient road systems and predict the outcomes of infrastructure changes

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
import warnings

warnings.filterwarnings('ignore')

# Load dataset
df = pd.read_csv('traffic_data.csv')  # Replace with your dataset

# Inspect dataset
print("Dataset shape:", df.shape)
print(df.head())
print(df.info())

# Check for missing data
missing_data = df.isnull().sum()
print("Missing values in each column:\n", missing_data)

# Basic data cleaning (e.g., fill or drop missing values)
df = df.dropna()  # Drop rows with missing data, you can also fill them if necessary

# Convert datetime column if applicable
# Assuming 'date_time' is a column in the dataset
df['date_time'] = pd.to_datetime(df['date_time'])  # Replace with the correct column name
df['hour'] = df['date_time'].dt.hour
df['day_of_week'] = df['date_time'].dt.dayofweek

# Define feature and target variable
# Replace 'traffic_volume' with the actual target variable name
X = df.drop(['traffic_volume', 'date_time'], axis=1)  # Drop target variable and any unneeded columns
y = df['traffic_volume']

# Encode categorical variables if any
```

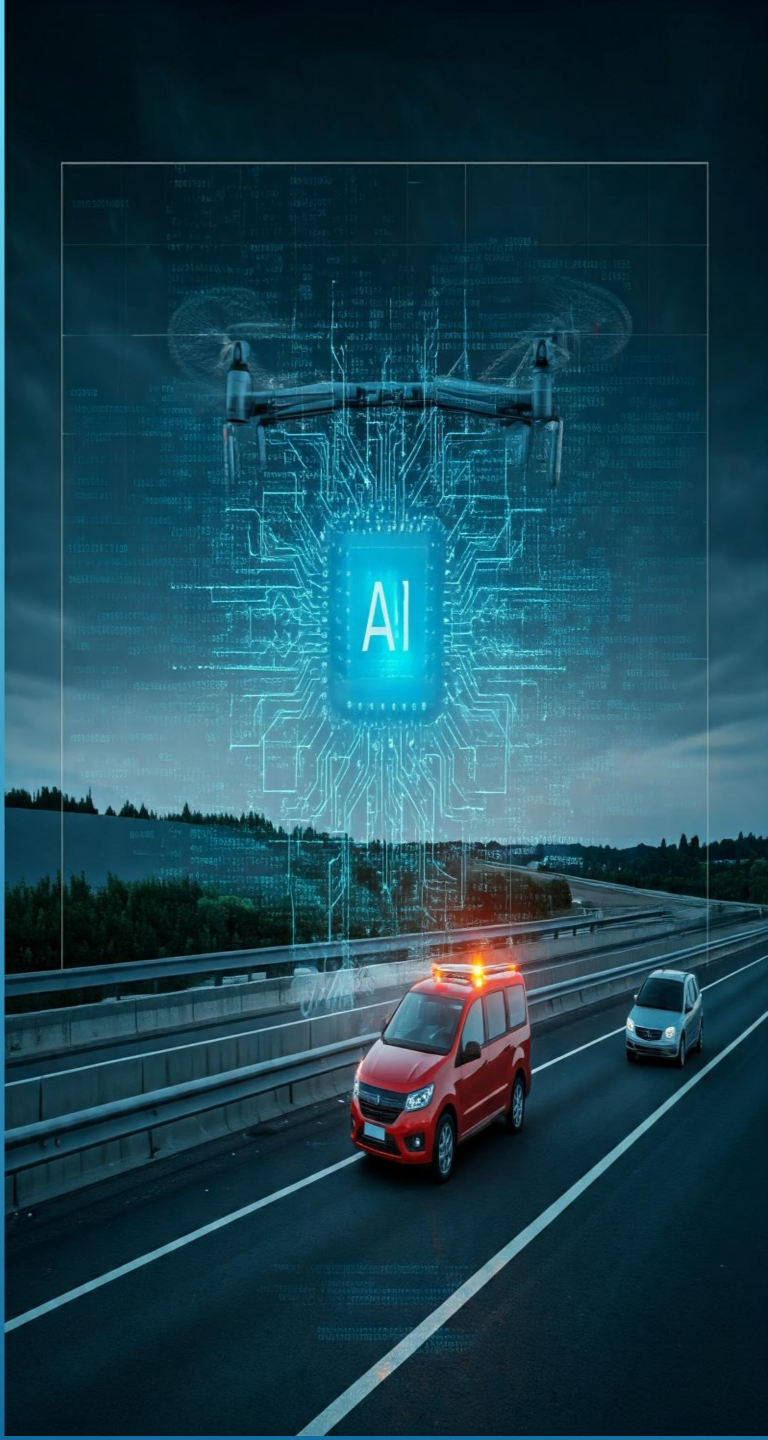


```
for column in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[column] = le.fit_transform(X[column])
    label_encoders[column] = le

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define regressors to evaluate
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
    'K-Nearest Neighbors': KNeighborsRegressor(),
    'Support Vector Regressor': SVR(),
    'Gradient Boosting': GradientBoostingRegressor()
}

# Train and evaluate each model
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results[name] = {'MSE': mse, 'MAE': mae, 'R^2 Score': r2}
    print(f"\nModel: {name}")
    print(f"Mean Squared Error: {mse}")
    print(f"Mean Absolute Error: {mae}")
    print(f"R^2 Score: {r2}")

# Display model comparison
results_df = pd.DataFrame(results).T
print("\nModel Performance Comparison:\n", results_df)

# Plot results
results_df[['MSE', 'MAE']].plot(kind='bar', figsize=(12, 6))
plt.title('Model Performance Comparison (Lower is better)')
```

```python
plt.ylabel('Error')
plt.show()

# Hyperparameter tuning example (Random Forest)
param_dist = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}
random_search = RandomizedSearchCV(RandomForestRegressor(), param_distributions=param_dist, n_iter=10, scoring='neg_mean_squared_error', cv=3, random_state=42)
random_search.fit(X_train, y_train)

print("\nBest parameters found for Random Forest:", random_search.best_params_)
print("Best MSE score:", -random_search.best_score_)

# Final model evaluation
final_model = random_search.best_estimator_
y_final_pred = final_model.predict(X_test)

# Calculate final metrics
final_mse = mean_squared_error(y_test, y_final_pred)
final_mae = mean_absolute_error(y_test, y_final_pred)
final_r2 = r2_score(y_test, y_final_pred)

print("\nFinal Model Performance:")
print("Mean Squared Error:", final_mse)
print("Mean Absolute Error:", final_mae)
print("R^2 Score:", final_r2)

# Feature Importance (for Random Forest or Gradient Boosting models)
if hasattr(final_model, 'feature_importances_'):
    feature_importance = final_model.feature_importances_
    feature_names = X.columns
    sorted_idx = np.argsort(feature_importance)[::-1]
    plt.figure(figsize=(10, 6))
    plt.barh(feature_names[sorted_idx], feature_importance[sorted_idx], align='center')
    plt.xlabel("Feature Importance")
    plt.title("Feature Importance of the Final Model")
    plt.show()
```

Explanation of the Code

1. **Data Loading and Exploration**:
   - Load the traffic data from a CSV file and check for missing values or anomalies in the dataset.
2. **Feature Engineering**:
   - Extract time-based features like hour and day_of_week, which can be important for predicting traffic volume.
3. **Preprocessing**:
   - Label encode categorical features and standardize the numerical features.
4. **Model Training and Evaluation**:
   - Train multiple regression models to predict traffic volume, including Linear Regression, Decision Tree, Random Forest, K-Nearest Neighbors, Support Vector Regressor, and Gradient Boosting.
   - Evaluate each model based on Mean Squared Error (MSE), Mean Absolute Error (MAE) and R^2 Score.
5. **Visualization**:
   - Display model performance comparison through bar plots for error metrics.
6. **Hyperparameter Tuning**:
   - Use RandomizedSearchCV to find the best parameters for a Random Forest model to improve performance.
7. **Feature Importance**:
   - Display feature importance for models that support it (e.g., Random Forest or Gradient Boosting) to understand which features contribute most to the predictions.



CONCLUSION:The Traffic Management Project has successfully addressed several critical transportation challenges, improving overall traffic flow, safety, and sustainability within the targeted area. Through a comprehensive analysis and the implementation of a variety of strategies, including infrastructure upgrades, advanced traffic control systems, and public awareness campaigns, the project has demonstrated substantial positive outcomes.