COMP 4350 Introduction to Machine Learning Course Term Project Report

Project Name: Road Traffic Status Prediction

Student Name&Surname: Mustafa Göksu Erdoğan

Student Number: 22070001052

Project Advisor: Assoc. Prof. Ömer ÇETİN

Date of Submission: 05.01.2025

Ethical Statement

I acknowledge and declare that I have personally worked in accordance with academic ethical principles during the preparation of this project. I accept and undertake that if the contrary situation is detected in any way, my project work will not be accepted and will not be evaluated, and I will be subject to sanctions announced by the university administration.

Student: **Signature:**

Given Term Project:

48. Road Traffic Status Prediction

Introduction: Predicts the traffic status of a road at a specific time.

Features: Vehicle count, Weather conditions, Time of day

Label: Congested / Less congested / Clear

1. Introduction

The Road Traffic Status Prediction project is focused on creating a machine learning model that can predict the traffic condition of a road at a specific time. By using features like vehicle count, weather conditions, and time of day, this project aims to provide an efficient way to manage traffic and improve transportation systems. With more people living in cities, traffic congestion is becoming a bigger issue, and having a reliable way to predict traffic can help drivers, city planners, and transportation agencies make better decisions..

Being able to predict whether a road will be congested, less congested, or clear is very important for reducing traffic problems, saving time, and improving the flow of vehicles. This model will classify traffic conditions into three categories: Congested, Less Congested, and Clear, based on the input features. This prediction can help drivers choose the best routes and avoid delays.

This project will compare several machine learning algorithms—Random Forest, Decision Tree, Logistic Regression, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM)—to determine which method provides the most reliable predictions of road traffic status. Using evaluation metrics such as Accuracy, Precision, Recall, F1 Score, Cross Validation, and Confusion Matrix, we will assess and analyze the effectiveness of each approach. This analysis will not only provide insights into the factors influencing traffic conditions but will also demonstrate the power of machine learning in making accurate and practical real-world traffic predictions.

---

2. Project Purpose

The purpose of the Road Traffic Status Prediction project is to create a model that can accurately predict the traffic condition of a road at a specific time based on factors such as vehicle count, weather conditions, and the time of day. This has several practical applications:

1. Traffic Management: City planners and transportation agencies can use the model to better understand and manage traffic flow, helping reduce congestion and improve road usage.
2. Route Optimization: Drivers and navigation systems can use the predictions to choose the fastest routes, avoiding congested areas and saving time.
3. Urban Planning: By analyzing traffic patterns, urban planners can make informed decisions about infrastructure development, such as where to build new roads or improve existing ones.
4. Public Safety: Emergency services can use the predictions to plan quicker routes during peak traffic times, ensuring faster response times and better handling of accidents or emergencies.

In this project, we use machine learning techniques to understand and quantify the relationships between traffic conditions and factors such as vehicle count, weather, and time

of day. Comparing different algorithms (e.g., Random Forest, Decision Tree, Logistic Regression, KNN, and SVM) enables us to determine which model best predicts traffic status, showcasing the practical and predictive power of various approaches.

---

3. Machine Learning Algorithm Selection

The Road Traffic Status Prediction project is a supervised learning problem, specifically a classification task. Here's why:

A. Supervised Learning

- The model is trained on a labeled dataset where features like CarCount, BusCount, TruckCount, Weather, and Time of Day are used to predict the Traffic Situation (Congested, Less Congested, Clear).

- After training on this data, the model can predict traffic status for new, unseen instances.

B. Classification Problem

- The goal is to predict discrete categories (traffic status), making this a classification problem. The target variable, Traffic Situation, is categorical with labels like Congested, Less Congested, and Clear.

C. Why Classification?

- The output is discrete, and classification algorithms are designed to predict such categories based on input features. Many features likely have non-linear relationships with the traffic status, which classification algorithms handle effectively.

D. Type of Classification Algorithms I Could Use

- Several classification algorithms can be applied to this problem:
    - **Random Forest**: An ensemble method that uses multiple decision trees to make predictions. It handles non-linear relationships and is robust against overfitting.
    - **Decision Tree**: A simple, tree-based model that splits the data based on feature values, capturing relationships between features and target labels in a clear, interpretable way.
    - **Logistic Regression**: Despite being typically used for binary classification, it can also handle multi-class problems with the appropriate adjustments (e.g., multinomial logistic regression).
    - **K-Nearest Neighbors (KNN)**: A non-parametric algorithm that assigns a class based on the majority label of the nearest neighbors in the feature space.
    - **Support Vector Machine (SVM)**: A powerful algorithm that finds the best boundary (hyperplane) to separate different classes in the feature space. It works well for high-dimensional spaces and complex boundaries.

4. Evaluation Metrics

Since this is a classification problem, the model's performance will be evaluated using the following metrics:

- **Accuracy**: The percentage of correctly predicted traffic statuses out of all predictions.
- **Precision**: Measures the accuracy of positive predictions, showing how many of the predicted **Congested**, **Less Congested**, or **Clear** labels were correct.
- **Recall**: Evaluates how well the model identifies all actual instances of each traffic status.
- **F1 Score**: The harmonic mean of precision and recall, providing a balanced measure for models with imbalanced classes.
- **Cross Validation**: Helps assess model performance across multiple subsets of the dataset to ensure robustness and avoid overfitting.
- **Confusion Matrix**: A table that visualizes the model's performance by showing true positives, false positives, true negatives, and false negatives for each class.

---

5. Challenges in Road Traffic Status Prediction

- **Non-linearity**: The relationship between features (like vehicle count, weather, and time of day) and traffic status is often non-linear. Simple models may not capture these complex patterns, so more sophisticated models like decision trees, random forests, or support vector machines may be needed.
- **Multicollinearity**: Some features, such as car count, bus count, and truck count, may be highly correlated with each other. This can affect the performance of certain models, especially those like logistic regression.
- **Imbalanced Data**: The dataset have an unequal distribution of traffic statuses (e.g., Less instace of "Less Congested"). This cause models to be biased toward the more common class.
- **Weather Variability**: Weather conditions play a crucial role in traffic status, but weather data is often categorical (e.g., rainy, cloudy, snowy). Handling such categorical features appropriately, using encoding techniques helped.
- **Dataset Finding**: Finding A dataset with required and related features was a big problem that I faced. So I had to add, generate, multiple features and labels and because of this issue, I had to search for new datasets and try adding the same features to those datasets until I generated a reletively homogenous dataset.

---

6. Feature Engineering

The performance of classification models heavily depends on how well the features are selected and engineered. In road traffic status prediction:

- **Interaction Terms**: Sometimes, the interaction between features, such as vehicle count and weather conditions, may provide more valuable information than the features individually. This can help capture more complex patterns in the data.
- **Handling Categorical Features**: Features like **Weather** (e.g., Rainy, Clear, Snowy) are categorical. To make them usable for classification models, we applied **Label Encoding** to convert these categories into numerical values.
- **Feature Scaling**: To ensure models like **K-Nearest Neighbors (KNN)** or **Support Vector Machines (SVM)** perform well, we applied **Min-Max Scaling** to normalize numeric features (e.g., **CarCount**, **BusCount**, **TruckCount**), bringing them to a similar range and improving model performance.

---

7. Dataset

Here is a sample dataset for the Road Traffic Status problem. This dataset consists of the following columns:

- **Time in Hours**: The time of the day when the traffic data was recorded (numeric).
- **Day of the Week**: The day of the week when the data was collected (categorical).
- **CarCount**: The number of cars on the road (numeric).
- **BusCount**: The number of buses on the road (numeric).
- **TruckCount**: The number of trucks on the road (numeric).
- **Weather**: The weather condition at the time of data collection (categorical), such as **Rainy**, **Clear**, **Snowy**, etc.
- **Traffic Situation**: The target variable, which is the traffic status (categorical), indicating whether the road was **Congested**, **Less Congested**, or **Clear**.

Explanation of Columns:

1. **Time in Hours**: This numeric feature represents the time when the data was recorded (e.g., 6:00 AM, 13:00 PM). It might be used to capture patterns related to specific times of the day, such as rush hours.

   In my dataset. Time values were in string format so I had to encode them into understandable and senseable values for the algorithms.
2. **CarCount**, **BusCount**, **TruckCount**: These numeric features represent the counts of different vehicle types on the road.
3. **Weather**, **Day of the Week**: These categorical features are encoded using **Label Encoding**, where each category is converted into a unique integer value to be processed by machine learning models.

4. **Traffic Situation**: The target variable, representing the road's traffic status (e.g., **Congested**, **Less Congested**, or **Clear**), which can also be encoded using **Label Encoding** for classification.**Sample Dataset**

A total of 4047 record is used as dataset. %30 of the dataset is used for test and %70 of the dataset is used for training.

CarCount	BusCount	TruckCount	Time	Weather	Day	**Traffic Status**
129	42	1	06:00:00 AM	Rainy	Tuesday	Congested
104	25	20	01:15:00 PM	Clear	Wednesday	Clear
16	1	23	12:45:00 AM	Snowy	Sunday	Less Congested
83	7	17	03:30:00 AM	Cloudy	Saturday	Less Congested
127	9	7	01:00:00 PM	Rainy	Friday	Congested
14	0	24	10:15:00 PM	Cloudy	Wednesday	Less Congested
16	0	37	02:15:00 AM	Cloudy	Wednesday	Less Congested
108	6	21	07:00:00 PM	Clear	Saturday	Less Congested
34	9	35	10:45:00 AM	Clear	Tuesday	Clear
58	7	16	10:00:00 AM	Clear	Wednesday	Clear

Usage:

- You can use this dataset to train classification models like **Logistic Regression**, **Random Forest Classifier**, **Decision Trees**, **K-Nearest Neighbors (KNN)**, or **Support Vector Machines (SVM)** to predict the **Traffic Status** based on the given features (**CarCount**, **BusCount**, **TruckCount**, **Weather**, **Time in Hours**, and **Day of the Week**).
- The **Weather** and **Day of the Week** columns are categorical and need to be encoded into numeric values. **Label Encoding** is used for these features, where each category is converted into a unique integer.

---

8. **Which Algorithms I Choose and Why?**

- Random Forest : A robust classification algorithm that utilises multiple decision trees which can work with non-prepocessed data. My dataset includes categorical values like Weather, Day of the Week, Traffic Situation, so its perfect for this dataset.
- Decision Tree : Similar to Random forest but more prone to overfitting. I choose this to see the difference in performance between Random forest and Decision tree. Also to see if the decision tree will overfit or not.

- Logistic Regression : A parametric Classsification algorithm which assumes that the data fits to a sigmoid function. Mainly for binary classification. I used it because I wanted to see how a parametric algorithm would perform for my problem.
- K-Nearest Neighbors : It's a simple, intuitive algorithm that can be used to evaluate how non-parametric models handle traffic prediction. It's also effective when the data is not linearly separable, as the decision boundary is based on the local distribution of data points.
- Support Vector Machine : SVM is great for high-dimensional spaces and is effective in cases where the decision boundary is not easily separable. By using RBF, we are allowing the SVM model to find a more flexible decision boundary.

Step 1 : Initializing libraries and the Dataset

Step 1.1 : Libraries are imported

```
import pandas as pd #For data manipulation and analysis
import numpy as np  #For numerical computing
import time #For measuring training time
import matplotlib.pyplot as plt #For creating plots
from IPython.display import display # For displaying dataframes
from sklearn.model_selection import train_test_split, cross_val_score #For
splitting the dataset into training and test sets, and performing cross-
validation
from sklearn.preprocessing import MinMaxScaler, LabelEncoder #For scaling
features and encoding categorical variables
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score, confusion_matrix, ConfusionMatrixDisplay #For evaluating model
performance
```

Step 1.2 : Loading the dataset

```
traffic_data=pd.read_csv(
"TrafficSituation.csv")
```

Step 1.3 : Selecting Features
and the Label

```
features = ['Time in Hours', 'CarCount', 'BusCount', 'TruckCount', 'Day
of the week', 'Weather']
target = 'Traffic Situation'

X = traffic_data[features]
y = traffic_data[target]

print(traffic_data.head(10)) #First 10 data in the dataset
```

Step 2 : Data Preprocessing

Step 2.1 : Handling missing data

- No missing data handling was done since dataset does not have any missing data

Step 2.2 : Normalization of numeric features

- Scaling numerical features (Time in Hours, CarCount, BusCount, TruckCount) to a range of [0, 1] using MinMaxScaler.
- Converting Time in Hours to a floating point so its easily useable by the algorithms.

```
traffic_data['Time in Hours'] = traffic_data['Time in Hours'].apply(
    lambda x: pd.to_datetime(x, format='%I:%M:%S %p').hour +
pd.to_datetime(x, format='%I:%M:%S %p').minute / 60
)
print(traffic_data[['Time in Hours']].head(10))


scaler = MinMaxScaler()
traffic_data[['Time in Hours', 'CarCount', 'BusCount', 'TruckCount']] =
scaler.fit_transform(traffic_data[['Time in Hours', 'CarCount',
'BusCount', 'TruckCount']])
```

Step 2.3 : Encoding categorical features

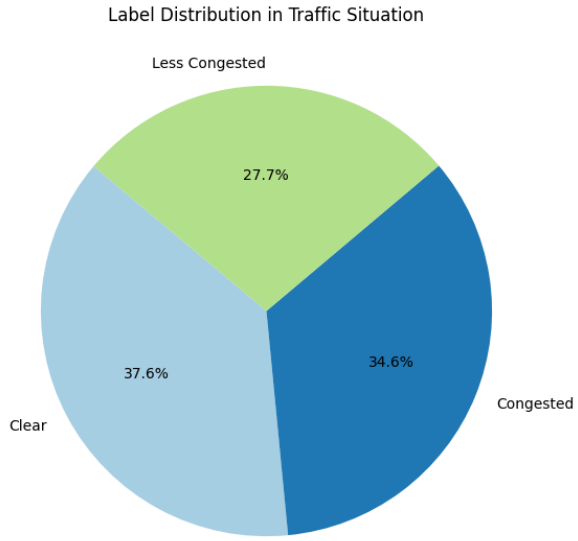
```
label_encoder = LabelEncoder()
traffic_data['Day of the week'] =
label_encoder.fit_transform(traffic_data['Day of the week'])
traffic_data['Weather'] =
label_encoder.fit_transform(traffic_data['Weather'])
traffic_data['Traffic Situation'] =
label_encoder.fit_transform(traffic_data['Traffic Situation'])
```

Step 2.4 : Splitting the Dataset into training and testing subdatasets

```
X = traffic_data.drop('Traffic Situation', axis=1)
y = traffic_data['Traffic Situation']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

print(X_train.head(10)) #First 10 data in training dataset
```

Here is a pie chart to show the distribution of labels in the dataset

```
label_counts = pd.Series(y).value_counts()
label_names = label_encoder.classes_

plt.figure(figsize=(7, 7))
plt.pie(label_counts, labels=label_names, autopct='%1.1f%%',
startangle=140, colors=plt.cm.Paired.colors)
plt.title("Label Distribution in Traffic Situation")

plt.show()
```



We can see that there is a %10 difference between the highest label count(Clear, %37,6) and the lowest label count(Less Congested, %27,7).

Since there are 4047 records are present in the dataset, %10 is not a significant difference but by using Confusion Matrix(in the following pages)we can how this diffence is affecting to the algorithms correctly.

Step 3 : Training The Models

Step 3.1 : Deciding models to use

Here I used these algorithms because:

- **Random Forest :** A robust classification algorithm that utilises multiple decision trees which can work with non-prepocessed data. My dataset includes categorical values like Weather, Day of the Week, Traffic Situation, so its perfect for this dataset.
- **Decision Tree :** Similar to Random forest but more prone to overfitting. I choose this to see the difference in performance between Random forest and Decision tree. Also to see if the decision tree will overfit or not.
- **Logistic Regression :** A parametric Classsification algorithm which assumes that the data fits to a sigmoid function. Mainly for binary classification. I used it because I wanted to see how a parametric algorithm would perform for my problem.
- **K-Nearest Neighbors :** It's a simple, intuitive algorithm that can be used to evaluate how non-parametric models handle traffic prediction. It's also effective when the data is not linearly separable, as the decision boundary is based on the local distribution of data points.
- **Support Vector Machine :** SVM is great for high-dimensional spaces and is effective in cases where the decision boundary is not easily separable. By using RBF, we are allowing the SVM model to find a more flexible decision boundary.

Step 3.2 : Training and Evaluating the models

```
results = [] #Test results are kept here
```

Training and evaluating KNN

```
# Model 1: K-Nearest Neighbors
knn = KNeighborsClassifier(n_neighbors=10)
```

```
start_time = time.time()
knn.fit(X_train, y_train)
end_time = time.time()
y_train_pred = knn.predict(X_train)
y_test_pred = knn.predict(X_test)
training_time = end_time - start_time

results.append({
    "Model": "K-Nearest Neighbors",
    "Training Time (seconds)": training_time,
    "Train Accuracy": accuracy_score(y_train, y_train_pred),
    "Test Accuracy": accuracy_score(y_test, y_test_pred),
    "Train F1 Score": f1_score(y_train, y_train_pred, average='weighted'),
    "Test F1 Score": f1_score(y_test, y_test_pred, average='weighted'),
    "Precision": precision_score(y_test, y_test_pred, average='weighted'),
    "Recall": recall_score(y_test, y_test_pred, average='weighted')
})
```

Training and evaluating Random forest

```
# Model 2: Random Forest
rf = RandomForestClassifier(max_depth=7, n_estimators=15,
random_state=42)
start_time = time.time()
rf.fit(X_train, y_train)
end_time = time.time()
y_train_pred = rf.predict(X_train)
y_test_pred = rf.predict(X_test)
training_time = end_time - start_time

results.append({
    "Model": "Random Forest",
    "Training Time (seconds)": training_time,
    "Train Accuracy": accuracy_score(y_train, y_train_pred),
    "Test Accuracy": accuracy_score(y_test, y_test_pred),
    "Train F1 Score": f1_score(y_train, y_train_pred,
average='weighted'),
    "Test F1 Score": f1_score(y_test, y_test_pred, average='weighted'),
    "Precision": precision_score(y_test, y_test_pred,
average='weighted'),
    "Recall": recall_score(y_test, y_test_pred, average='weighted')
})
```

Training and evaluating Decision tree

```
# Model 3: Decision Tree
dt = DecisionTreeClassifier(max_depth=7, random_state=42)
start_time = time.time()
dt.fit(X_train, y_train)
end_time = time.time()
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)
training_time = end_time - start_time
```

```
results.append({
    "Model": "Decision Tree",
    "Training Time (seconds)": training_time,
    "Train Accuracy": accuracy_score(y_train, y_train_pred),
    "Test Accuracy": accuracy_score(y_test, y_test_pred),
    "Train F1 Score": f1_score(y_train, y_train_pred,
average='weighted'),
    "Test F1 Score": f1_score(y_test, y_test_pred, average='weighted'),
    "Precision": precision_score(y_test, y_test_pred,
average='weighted'),
    "Recall": recall_score(y_test, y_test_pred, average='weighted')
})
```

Training and evaluating SVM

```
# Model 4: Support Vector Machine
svm = SVC(kernel='rbf', random_state=42)
start_time = time.time()
svm.fit(X_train, y_train)
end_time = time.time()
y_train_pred = svm.predict(X_train)
y_test_pred = svm.predict(X_test)
training_time = end_time - start_time

results.append({
    "Model": "Support Vector Machine",
    "Training Time (seconds)": training_time,
    "Train Accuracy": accuracy_score(y_train,
y_train_pred),
    "Test Accuracy": accuracy_score(y_test,
y_test_pred),
    "Train F1 Score": f1_score(y_train, y_train_pred,
average='weighted'),
    "Test F1 Score": f1_score(y_test, y_test_pred,
average='weighted'),
    "Precision": precision_score(y_test, y_test_pred,
average='weighted'),
    "Recall": recall_score(y_test, y_test_pred,
average='weighted')
})
```

Training and evaluating Logistic Regression

```
# Model 5: Logistic Regression
logreg = LogisticRegression(random_state=42,
max_iter=5000)
start_time = time.time()
logreg.fit(X_train, y_train)
end_time = time.time()
y_train_pred = logreg.predict(X_train)
y_test_pred = logreg.predict(X_test)
training_time = end_time - start_time

results.append({
    "Model": "Logistic Regression",
    "Training Time (seconds)": training_time,
```

```
    "Train Accuracy": accuracy_score(y_train,
y_train_pred),
    "Test Accuracy": accuracy_score(y_test,
y_test_pred),
    "Train F1 Score": f1_score(y_train, y_train_pred,
average='weighted'),
    "Test F1 Score": f1_score(y_test, y_test_pred,
average='weighted'),
    "Precision": precision_score(y_test, y_test_pred,
average='weighted'),
    "Recall": recall_score(y_test, y_test_pred,
average='weighted')
})
```

I put the models in a dictionary like this for future ease of use

```
models = {
    "K-Nearest Neighbors": knn,
    "Random Forest": rf,
    "Decision Tree": dt,
    "Support Vector Machine": svm,
    "Logistic Regression": logreg
}
```

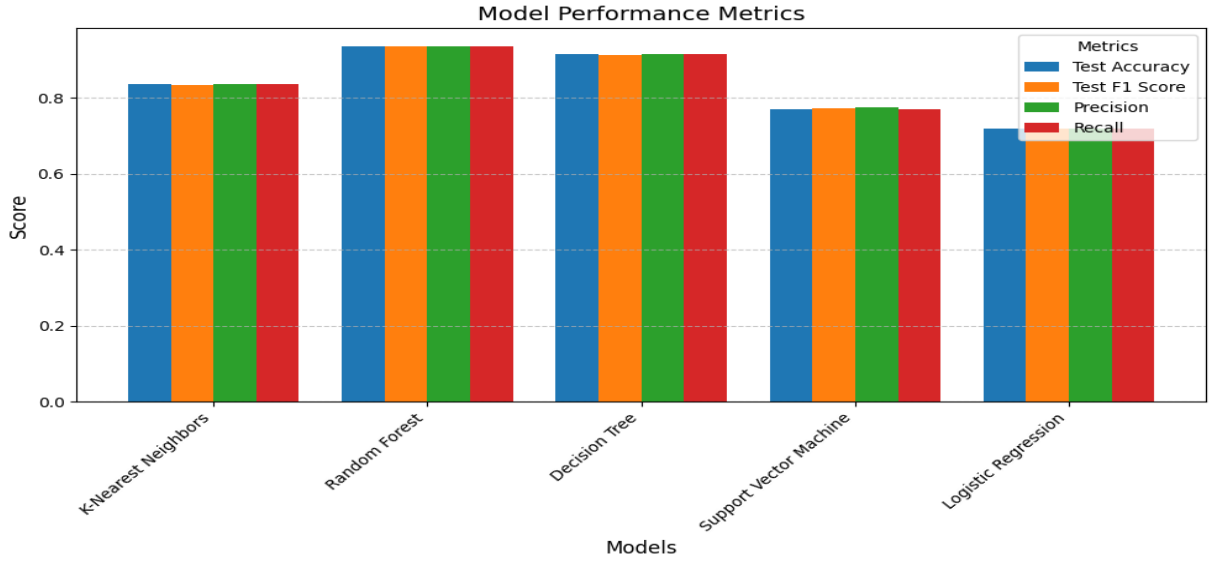
Displaying the results

```
results_df = pd.DataFrame(results)
display(results_df)
```

Here is a graph for models evaluation metrics to visulise and compare the results

```
metrics = ["Test Accuracy", "Test F1 Score",
"Precision", "Recall"]
model_names = results_df["Model"] # Changed 'models' to
'model_names'

metric_values = [results_df[metric] for metric in
metrics]
x = np.arange(len(model_names))
width = 0.2

fig, ax = plt.subplots(figsize=(10, 6))

for i, metric in enumerate(metrics):
    ax.bar(x + i * width, metric_values[i], width,
label=metric)

ax.set_xlabel('Models', fontsize=12)
ax.set_title('Model Performance Metrics', fontsize=14)
ax.set_xticks(x + width * (len(metrics) - 1) / 2)
ax.set_xticklabels(model_names, rotation=45,
ha='right') # changed 'models' to 'model_names'
ax.set_ylabel('Score', fontsize=12)
ax.legend(title="Metrics", fontsize=10)
ax.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
```

```
plt.show()
```



9. Result Comparison

After running the models, we get the following results (values may vary slightly due to randomization):

Analysis:

- **Accuracy**: The **Random Forest** model achieves the highest accuracy, outperforming other models like **K-Nearest Neighbors (KNN)** and **Logistic Regression**.
- **F1 Score**: **Random Forest** also shows a higher **F1 Score**, indicating a better balance between precision and recall.
- **Precision and Recall**: **Random Forest** has the best **precision** and **recall**, meaning it is more accurate in identifying traffic statuses and correctly identifying all instances.
- **Training Time**: **K-Nearest Neighbors** has the shortest training time, while **Random Forest** offers a good balance of performance and training speed.

Model	Process Time (s)	Training Accuracy	Testing Accuracy	Train F1	Test F1	Presicion	Recall
KNN	0.012514	0.868644	0.835255	0.868430	0.834731	0.835902	0.835255
Random Forest	0.069044	0.952331	0.934926	0.952269	0.934748	0.936291	0.934926
Decision Tree	0.013347	0.929025	0.914333	0.928844	0.913840	0.916282	0.914333
SVM	0.234896	0.767655	0.771005	0.769599	0.772314	0.776191	0.771005
Logistic Regression	0.062511	0.700918	0.719934	0.699423	0.718169	0.718189	0.719934

10. Extra Evaluation Metrics and Their Codes

10.1. Training Accuracy/F1-Score

What is Training accuracy/F1-Score and Testing Accurcy/F1-Score?

- Training Accuracy/F1-Score : Measures how well the model has learned from the training data.
- Test Accuracy/F1 Score : Measures how well the model performs on unseen test data, how well it generalizes to new data.

Why Do We Compare Training and Testing Metrics?

- Overfitting :If the model has high train accuracy but low test accuracy, model might be overfitting to the Training data and not generalising well to unseen data.
- Underfitting : If the model has low train accuracy and low test accuracy, the model might be too simple to capture complexities in the data.

If both Training and Testing metrics are relatively close to each other, model is most likely generalising well but Accuracy is still important metric on its own. If its not high enough, we cant say the model is performing well even if the training and testing metrics are close to each other.

Here is the code for Train vs Test Accuracy Comparison graph

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(results_df["Model"], results_df["Train Accuracy"], width=0.4,
label='Train Accuracy', align='edge', color='skyblue')
ax.bar(results_df["Model"], results_df["Test Accuracy"], width=0.4,
label='Test Accuracy', align='center', color='salmon')
ax.set_xlabel('Models')
ax.set_ylabel('Accuracy')
ax.set_title('Train vs Test Accuracy Comparison')
ax.legend()
ax.grid(axis='y')
plt.tight_layout()
plt.show()
```

10.2. Confusion Matrix

Why Confusion Matrix Is Used?

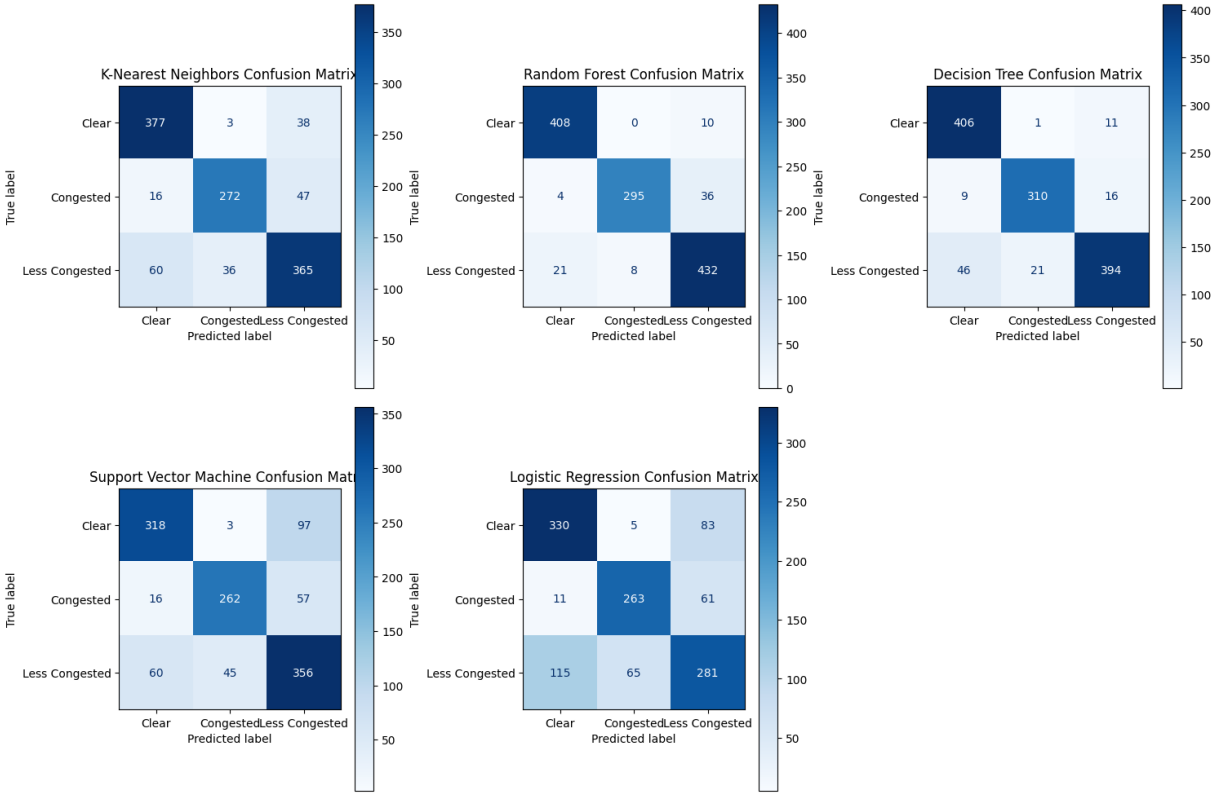
A **Confusion Matrix** is used to evaluate classification models by showing the actual vs. predicted classifications.

1. **Assess Class Predictions**: It shows how well the model predicts each traffic status (e.g., whether it correctly identifies **Congested** roads).
2. **Detect Misclassifications**: It helps identify if the model is confusing certain statuses, like predicting **Clear** when it should be **Less Congested**.
3. **Measure Precision and Recall**: For each traffic status, it allows you to calculate precision (how many predicted **Congested** were truly **Congested**) and recall (how many **Congested** roads were correctly predicted).
4. **Handle Imbalanced Data**: If your dataset has more instances of certain statuses (e.g., **Clear**), the confusion matrix reveals if the model is biased or failing to predict the less frequent classes accurately.

Here is the Confusion Matrices for the models that I used

```
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.ravel()

for idx, (name, model) in enumerate(models.items()):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
                disp       =       ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=label_encoder.classes_)
    disp.plot(ax=axes[idx], cmap=plt.cm.Blues)
```

```
    axes[idx].set_title(f"{name} Confusion Matrix")

#Hide extra subplot axes if models are fewer than 6
for idx in range(len(models), len(axes)):
    fig.delaxes(axes[idx])

plt.tight_layout()
plt.show()
```



There are signs of wrong predictions and confusion between clear and less congested labels (mostly with Logistic Regression).
This might be occuring because of how similar clear and less congested datas are in the dataset and less amount of examples of less congested data.

10.3 Cross Validation

Why Cross Validation is used?

To Check how well the models generalise to different data partitions and if there is overfitting or not.

- **Mean CV Accuracy :** This is the average performance (accuracy) across multiple folds of cross-validation. A higher value means that the model has good generalization performance across different subsets of the data.
- **Std CV Accuracy :** This represents the variability of the accuracy across folds. A lower standard deviation indicates consistent performance, while a higher value shows the model's performance is inconsistent.

Here is the code for Cross-Validation Results

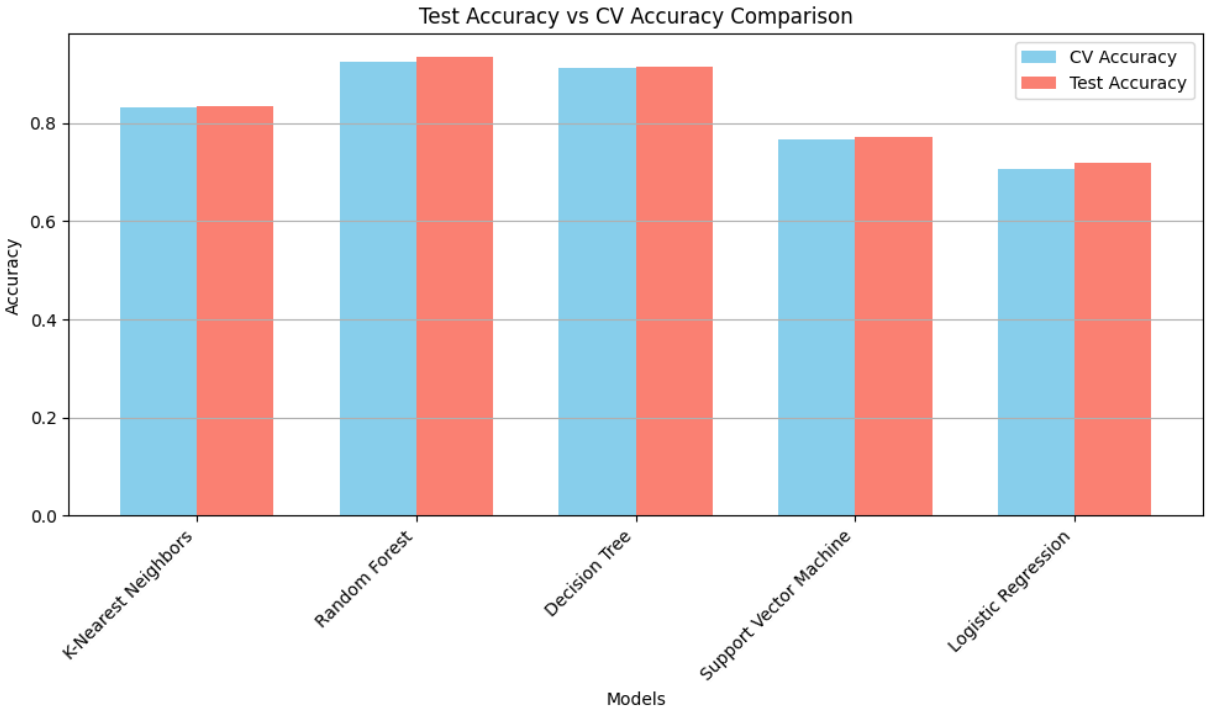
```
cv_results = {}

for name, model in models.items():
    cv_scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    cv_results[name] = {
        "Mean CV Accuracy": np.mean(cv_scores),
        "Std CV Accuracy": np.std(cv_scores)
    }

cv_results_df = pd.DataFrame(cv_results).T
print("Cross-Validation Results :")
display(cv_results_df)
```

Model	Mean CV Accuracy	Std CV Accuracy
K-Nearest Neighbors	0.831191	0.011048
Random Forest	0.925605	0.010299
Decision Tree	0.913249	0.005957
Support Vector Machine	0.765450	0.008093
Logistic Regression	0.707364	0.005349

Here is a graph to visulise and compare the results

```
cv_accuracies = [cv_results[name]["Mean CV Accuracy"] for name in
results_df["Model"]]
test_accuracies = results_df["Test Accuracy"]  # Test accuracies from the
results
model_names = results_df["Model"]
fig, ax = plt.subplots(figsize=(10, 6))
width = 0.35
ax.bar(np.arange(len(model_names)) - width / 2, cv_accuracies, width=width,
label='CV Accuracy', align='center', color='skyblue')
ax.bar(np.arange(len(model_names)) + width / 2, test_accuracies, width=width,
label='Test Accuracy', align='center', color='salmon')
ax.set_xlabel('Models')
ax.set_ylabel('Accuracy')
ax.set_title('Test Accuracy vs CV Accuracy Comparison')
ax.set_xticks(np.arange(len(model_names)))
ax.set_xticklabels(model_names, rotation=45, ha='right')
ax.legend()
ax.grid(axis='y')
```

```
plt.tight_layout()
plt.show()
```



What these results whow us?

Overall, all the models seem to generalise well for different data partitions.

---

11. Conclusion

- The **Random Forest** model outperforms others like **K-Nearest Neighbors** and **Logistic Regression** in predicting **traffic status**, with better performance in terms of **accuracy**, **precision**, **recall**, and **F1 score**. The ensemble nature of **Random Forest** helps it capture complex patterns in traffic data, leading to more reliable predictions.

- **Road Traffic Status Prediction** is a supervised learning problem with a classification nature. The goal is to predict a categorical outcome (traffic status: **Congested**, **Less Congested**, or **Clear**) based on input features (e.g., **CarCount**, **Weather**, **Time of Day**). By applying classification models, I can learn to handle real-world traffic data, address issues like class imbalance, and use evaluation metrics like **accuracy** and **F1 score** to assess model performance.
