In [1]:

```
import pandas as pd

# Load the CSV file
file_path = 'dataset_traffic_accident_prediction1.csv'  # Use this exact name
df = pd.read_csv(file_path)

# Show the shape of the dataset
print("Shape of the dataset:", df.shape)

# Display column names
print("Columns:\n", df.columns.tolist())

# Preview the first 5 rows
df.head()
```

```
Shape of the dataset: (840, 14)
Columns:
 ['Weather', 'Road_Type', 'Time_of_Day', 'Traffic_Density', 'Speed_Limit', 'Number_o
f_Vehicles', 'Driver_Alcohol', 'Accident_Severity', 'Road_Condition', 'Vehicle_Typ
e', 'Driver_Age', 'Driver_Experience', 'Road_Light_Condition', 'Accident']
```

Out[1]:

	Weather	Road_Type	Time_of_Day	Traffic_Density	Speed_Limit	Number_of_Vehicles	Dr
0	Rainy	City Road	Morning	1.0	100.0	5.0	
1	Clear	Rural Road	Night	NaN	120.0	3.0	
2	Rainy	Highway	Evening	1.0	60.0	4.0	
3	Clear	City Road	Afternoon	2.0	60.0	3.0	
4	Rainy	Highway	Morning	1.0	195.0	11.0	

In [5]:

```
# Check for missing values
print("Missing values per column:\n", df.isnull().sum())

# Drop columns with too many missing values (threshold optional)
df = df.dropna(thresh=0.5*len(df), axis=1)

# Replace this:
df.fillna(method='ffill', inplace=True)

# ✅ With this:
df.ffill(inplace=True)


# Verify
print("Remaining missing values:", df.isnull().sum().sum())
```

```
Missing values per column:
 Weather                 0
Road_Type               0
Time_of_Day             0
Traffic_Density         0
Speed_Limit             0
Number_of_Vehicles      0
Driver_Alcohol          0
Accident_Severity       1
Road_Condition          0
Vehicle_Type            0
Driver_Age              0
Driver_Experience       0
Road_Light_Condition    0
Accident                0
dtype: int64
Remaining missing values: 1
```

```
C:\Users\SHANGAR\AppData\Local\Temp\ipykernel_13180\550429416.py:8: FutureWarning: D
ataFrame.fillna with 'method' is deprecated and will raise in a future version. Use
obj.ffill() or obj.bfill() instead.
  df.fillna(method='ffill', inplace=True)
```

In [4]:

```
# Check for missing values
print("Missing values per column:\n", df.isnull().sum())

# Drop columns with too many missing values (threshold optional)
df = df.dropna(thresh=0.5*len(df), axis=1)

# Fill missing values
df.fillna(method='ffill', inplace=True)  # Forward fill as default
# OR: df.fillna(df.mean(numeric_only=True), inplace=True)  # Numeric only

# Verify
print("Remaining missing values:", df.isnull().sum().sum())
```

```
Missing values per column:
 Weather                 0
Road_Type               0
Time_of_Day             0
Traffic_Density         0
Speed_Limit             0
Number_of_Vehicles      0
Driver_Alcohol          0
Accident_Severity       1
Road_Condition          0
Vehicle_Type            0
Driver_Age              0
Driver_Experience       0
Road_Light_Condition    0
Accident                0
dtype: int64
Remaining missing values: 1
```

```
C:\Users\SHANGAR\AppData\Local\Temp\ipykernel_13180\124349902.py:8: FutureWarning: D
ataFrame.fillna with 'method' is deprecated and will raise in a future version. Use
obj.ffill() or obj.bfill() instead.
  df.fillna(method='ffill', inplace=True)  # Forward fill as default
```

In [6]:

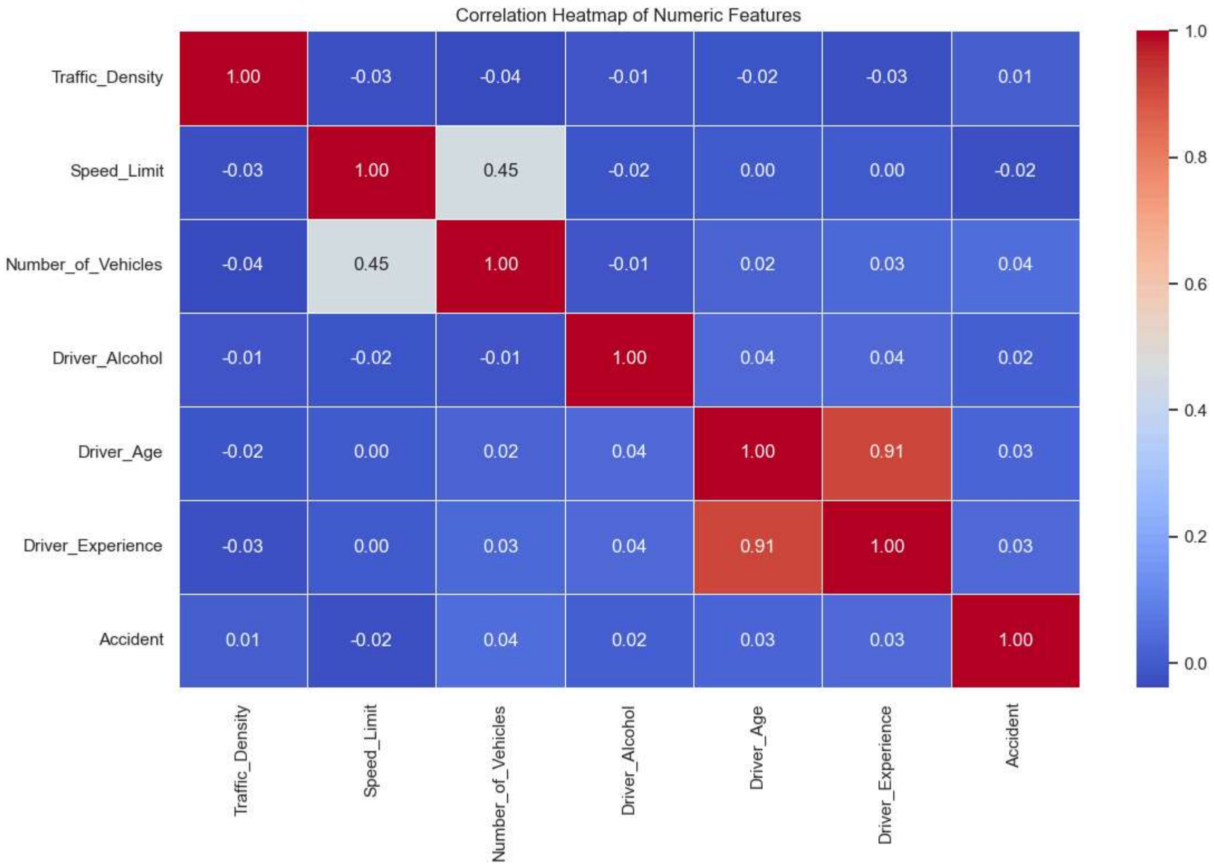
```
import pandas as pd

# Load dataset
df = pd.read_csv('dataset_traffic_accident_prediction1.csv')

# Step 1: Check missing values
print("🔍 Missing values before handling:\n")
print(df.isnull().sum())

# Step 2: Drop columns with too many missing values (optional)
# Drops columns where more than 50% of values are missing
df = df.dropna(thresh=0.5 * len(df), axis=1)

# Step 3: Forward fill remaining missing values (recommended method)
df.ffill(inplace=True)

# OPTIONAL: You could also backward fill if needed
# df.bfill(inplace=True)

# Step 4: Check again
print("\n✅ Missing values after handling:\n")
print(df.isnull().sum().sum())  # Should be 0 if all handled
```

```
🔍 Missing values before handling:

Weather                 42
Road_Type               42
Time_of_Day             42
Traffic_Density         42
Speed_Limit             42
Number_of_Vehicles      42
Driver_Alcohol          42
Accident_Severity       42
Road_Condition          42
Vehicle_Type            42
Driver_Age              42
Driver_Experience       42
Road_Light_Condition    42
Accident                42
dtype: int64

✅ Missing values after handling:

1
```

In [11]:

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

# Load the cleaned dataset
df = pd.read_csv('dataset_traffic_accident_prediction1.csv')

# Optional: Fill or drop missing values before plotting if not already done
df.ffill(inplace=True)

# Set visual style
```

```
sns.set(style='whitegrid')

# Compute correlation matrix
correlation_matrix = df[numeric_cols].corr()

# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=
plt.title('Correlation Heatmap of Numeric Features')
plt.tight_layout()
plt.show()
```



	Traffic_Density	Speed_Limit	Number_of_Vehicles	Driver_Alcohol	Driver_Age	Driver_Experience	Accident
Traffic_Density	1.00	-0.03	-0.04	-0.01	-0.02	-0.03	0.01
Speed_Limit	-0.03	1.00	0.45	-0.02	0.00	0.00	-0.02
Number_of_Vehicles	-0.04	0.45	1.00	-0.01	0.02	0.03	0.04
Driver_Alcohol	-0.01	-0.02	-0.01	1.00	0.04	0.04	0.02
Driver_Age	-0.02	0.00	0.02	0.04	1.00	0.91	0.03
Driver_Experience	-0.03	0.00	0.03	0.04	0.91	1.00	0.03
Accident	0.01	-0.02	0.04	0.02	0.03	0.03	1.00

In [ ]:

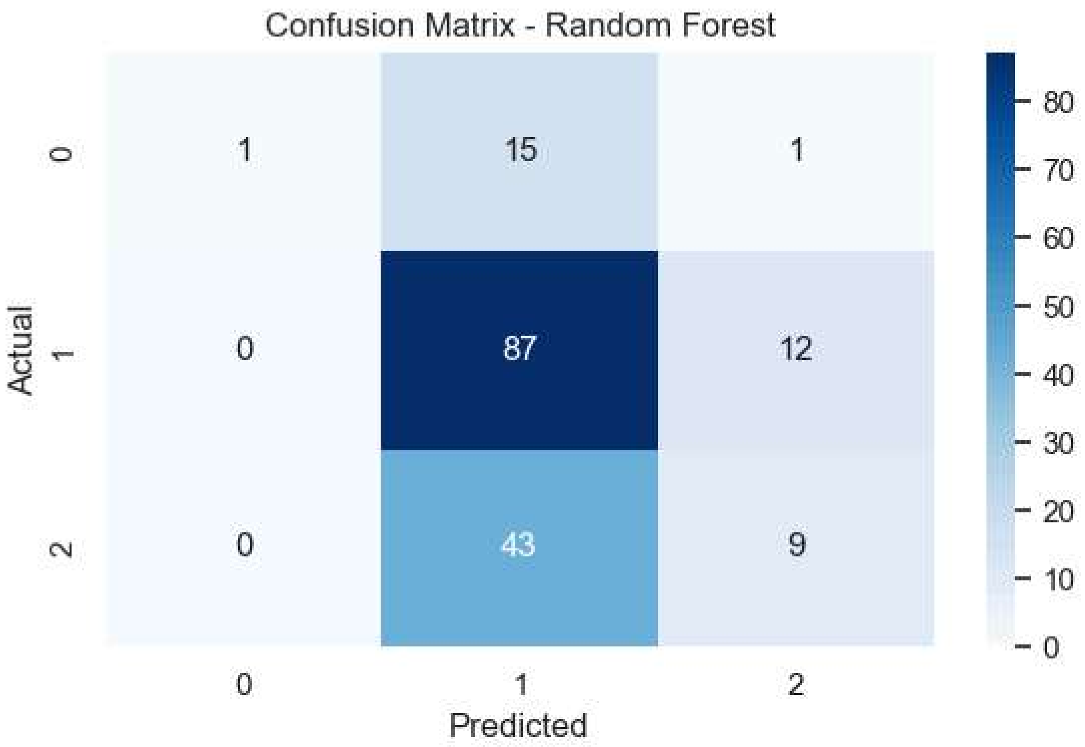
In [17]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load and prepare the dataset
df = pd.read_csv('dataset_traffic_accident_prediction1.csv')
df.ffill(inplace=True)  # Handle missing values

# Example: encode target variable (adjust column name if needed)
if df['Accident_Severity'].dtype == 'object':
    le = LabelEncoder()
    df['Accident_Severity'] = le.fit_transform(df['Accident_Severity'])

# Optional: One-hot encode other categorical features
df = pd.get_dummies(df, drop_first=True)

# Define features (X) and target (y)
X = df.drop('Accident_Severity', axis=1)
y = df['Accident_Severity']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


# 3. XGBoost (Advanced)
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_s
xgb_model.fit(X_train, y_train)
xgb_preds = xgb_model.predict(X_test)
print("XGBoost Accuracy:", accuracy_score(y_test, xgb_preds))
print(classification_report(y_test, xgb_preds))
```

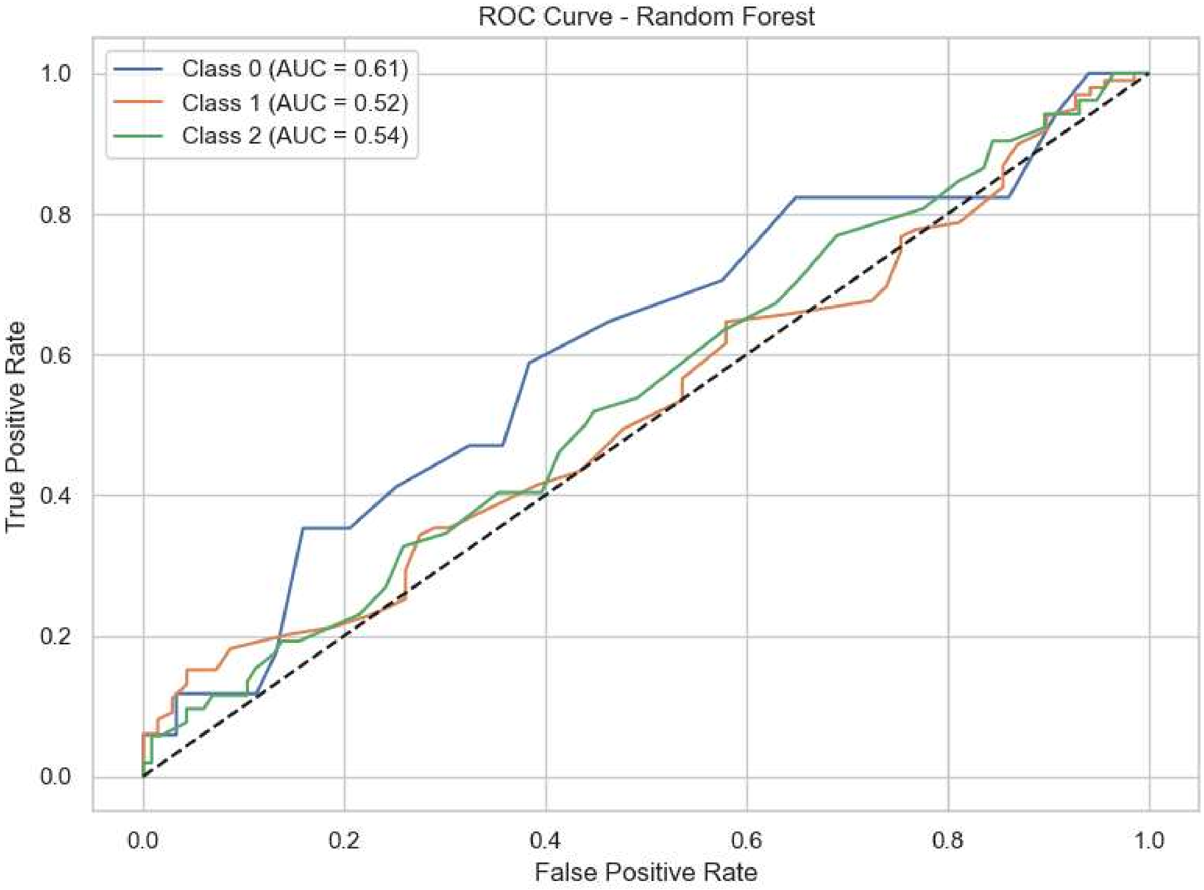
```
C:\Users\SHANGAR\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning:
[12:05:08] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
XGBoost Accuracy: 0.5059523809523809
              precision    recall  f1-score   support

           0       0.25      0.06      0.10        17
           1       0.58      0.74      0.65        99
           2       0.28      0.21      0.24        52

    accuracy                           0.51       168
   macro avg       0.37      0.34      0.33       168
weighted avg       0.46      0.51      0.47       168
```

In [19]:

```
# Assumes models and predictions are already made from previous steps
# y_test: true labels
# log_preds, rf_preds, xgb_preds: predictions from different models
from sklearn.metrics import classification_report

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion matrix
cm = confusion_matrix(y_test, rf_preds)

# Plot heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()
```



In [20]:
```
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.preprocessing import label_binarize

# If multiclass, binarize the output
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])  # Adjust based on your targ
rf_preds_proba = rf_model.predict_proba(X_test)

# Plot ROC curve for each class
plt.figure(figsize=(8, 6))
for i in range(y_test_bin.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], rf_preds_proba[:, i])
    auc = roc_auc_score(y_test_bin[:, i], rf_preds_proba[:, i])
    plt.plot(fpr, tpr, label=f'Class {i} (AUC = {auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Random line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```


ROC Curve - Random Forest
Class 0 (AUC = 0.61)
Class 1 (AUC = 0.52)
Class 2 (AUC = 0.54)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0