```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

Load the Dataset

```
file_path = '/content/Multiple Classification - EV Battery Faults Dataset.xlsx'
df = pd.read_excel(file_path)
```

Step 1: Exploratory Data Analysis (EDA)

```
print("Dataset Info:")
print(df.info())
print("\nFirst 5 Rows:")
print(df.head())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1152 entries, 0 to 1151
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   SoC          1152 non-null   float64
 1   Temperature  1152 non-null   float64
 2   Voltage      1152 non-null   float64
 3   Label        1152 non-null   int64
dtypes: float64(3), int64(1)
memory usage: 36.1 KB
None

First 5 Rows:
          SoC  Temperature   Voltage  Label
0  100.000000   298.150000  4.014300      0
1   99.173138   298.849283  3.916820      0
2   98.346276   299.665201  3.887562      0
3   97.519413   300.497825  3.877287      0
4   96.692551   301.327592  3.870545      0
```

Check for missing values

```
# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())
```

```
Missing Values:
SoC            0
Temperature    0
Voltage        0
Label          0
dtype: int64
```

Summary statistics

```
# Summary statistics
print("\nSummary Statistics:")
print(df.describe())
```

```
Summary Statistics:
               SoC   Temperature      Voltage        Label
count  1152.000000   1152.000000  1152.000000  1152.000000
mean   -577.695941   7790.772745     2.298061     0.947917
std    1222.210370  14325.882567     1.588660     0.825777
min   -4966.525811    298.150000     0.000000     0.000000
25%    -545.261868    332.411216     0.000000     0.000000
50%      22.886109    372.348398     3.167955     1.000000
75%      61.173425   8162.870735     3.522845     2.000000
```

```
max      100.000000  58394.697430     4.019340     2.000000
```

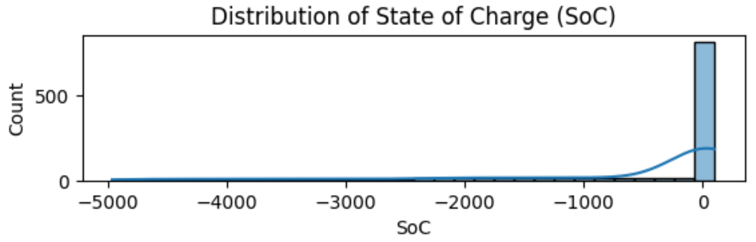
Distribution of features

```
# Distribution of features
plt.figure(figsize=(15, 10))
```

```
<Figure size 1500x1000 with 0 Axes>
<Figure size 1500x1000 with 0 Axes>
```

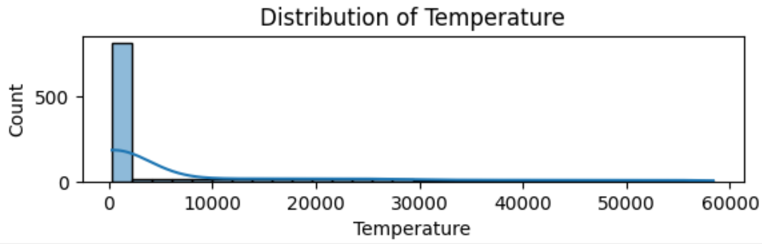
```
# Distribution of SoC
plt.subplot(3, 1, 1)
sns.histplot(df['SoC'], kde=True, bins=30)
plt.title('Distribution of State of Charge (SoC)')
```

```
Text(0.5, 1.0, 'Distribution of State of Charge (SoC)')
```



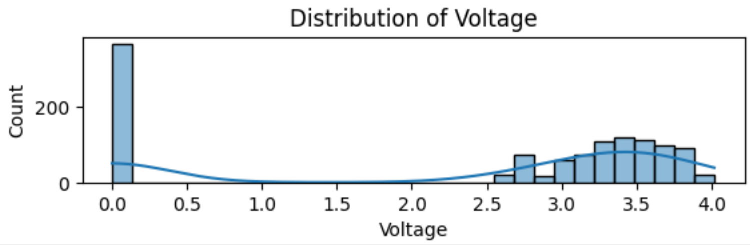

```
# Distribution of Temperature
plt.subplot(3, 1, 2)
sns.histplot(df['Temperature'], kde=True, bins=30)
plt.title('Distribution of Temperature')
```

```
Text(0.5, 1.0, 'Distribution of Temperature')
```




```
# Distribution of Voltage
plt.subplot(3, 1, 3)
sns.histplot(df['Voltage'], kde=True, bins=30)
plt.title('Distribution of Voltage')
```

```
Text(0.5, 1.0, 'Distribution of Voltage')
```



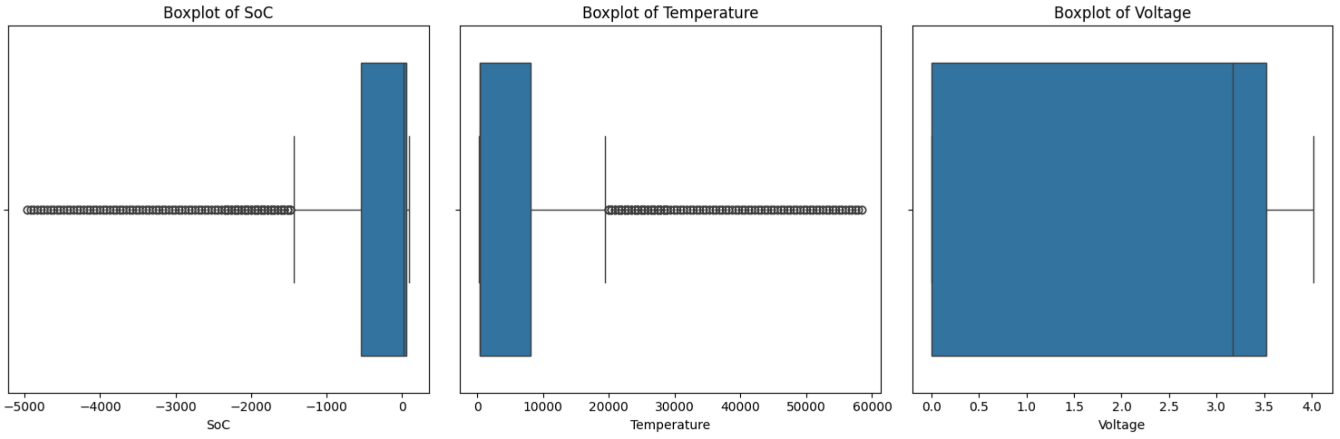
Checking Outliers

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

```
<Figure size 640x480 with 0 Axes>
```

```
# Boxplots to detect outliers
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.boxplot(x=df['SoC'])
plt.title('Boxplot of SoC')

plt.subplot(1, 3, 2)
```

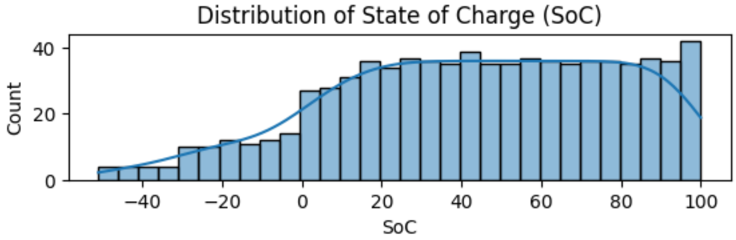
```
sns.boxplot(x=df['Temperature'])
plt.title('Boxplot of Temperature')

plt.subplot(1, 3, 3)
sns.boxplot(x=df['Voltage'])
plt.title('Boxplot of Voltage')

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



--- Outlier Handling ---

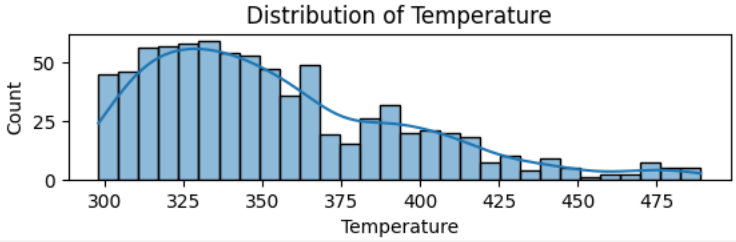
```
# Z-Score Method
def handle_outliers_zscore(df, columns, threshold=3):
    for col in columns:
        z = np.abs(stats.zscore(df[col]))
        df = df[(z < threshold)]
    return df

# IQR Method
def handle_outliers_iqr(df, columns, multiplier=1.5):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - multiplier * IQR
        upper_bound = Q3 + multiplier * IQR
        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df

# Columns to handle outliers
columns_to_handle = ['SoC', 'Temperature', 'Voltage']

# Apply Z-Score method
df_zscore = handle_outliers_zscore(df.copy(), columns_to_handle)

# Apply IQR method
df_iqr = handle_outliers_iqr(df.copy(), columns_to_handle)

# Choose which dataframe to use for further analysis
# For now, let's use df_iqr, but you can switch to df_zscore
df = df_iqr.copy()


# Distribution of SoC
plt.subplot(3, 1, 1)
sns.histplot(df['SoC'], kde=True, bins=30)
plt.title('Distribution of State of Charge (SoC)')
```

```
Text(0.5, 1.0, 'Distribution of State of Charge (SoC)')
```



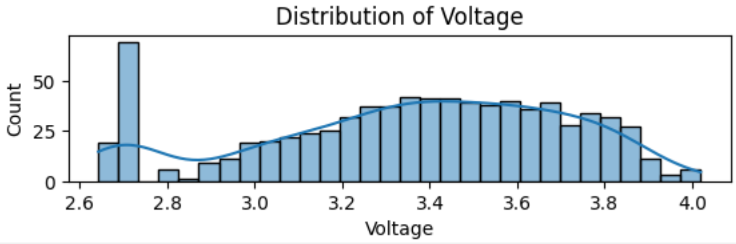

```
# Distribution of Temperature
plt.subplot(3, 1, 2)
sns.histplot(df['Temperature'], kde=True, bins=30)
plt.title('Distribution of Temperature')
```

```
Text(0.5, 1.0, 'Distribution of Temperature')
```




```
# Distribution of Voltage
plt.subplot(3, 1, 3)
sns.histplot(df['Voltage'], kde=True, bins=30)
plt.title('Distribution of Voltage')
```

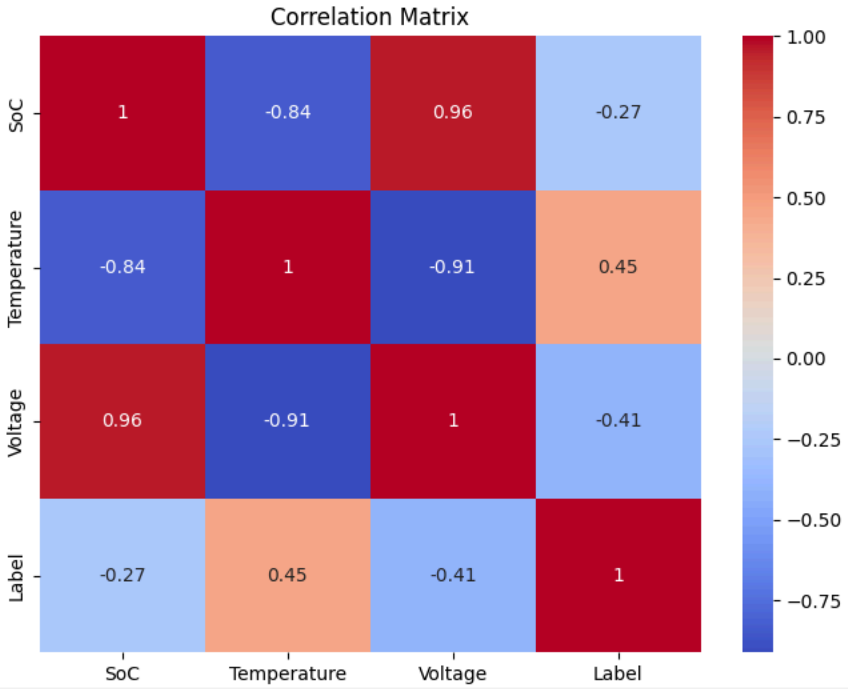
```
Text(0.5, 1.0, 'Distribution of Voltage')
```



Correlation Matrix

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

# Correlation matrix
plt.figure(figsize=(8, 6))
corr_matrix = df[['SoC', 'Temperature', 'Voltage', 'Label']].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



Step 2: Define Fire Risk Thresholds

```
temp_threshold = 350  # Temperature threshold in Kelvin
voltage_low_threshold = 3.0  # Low voltage threshold
voltage_high_threshold = 4.2  # High voltage threshold
soc_low_threshold = 10  # Low SoC threshold
soc_high_threshold = 90  # High SoC threshold
```

Step 3: Analyze the Dataset and Add Fire Risk Column

```
df['Fire Risk'] = 0  # Initialize Fire Risk column
```

Conditions for Fire Risk

```
fire_conditions = (
    (df['Temperature'] > temp_threshold) |
    (df['Voltage'] < voltage_low_threshold) |
    (df['Voltage'] > voltage_high_threshold) |
    (df['SoC'] < soc_low_threshold) |
    (df['SoC'] > soc_high_threshold) |
    (df['Label'].isin([1, 2]))  # Over-discharge or short circuit
)
```

Step 4: Display Fire Risk Analysis

```
# Set Fire Risk = 1 if any condition is met
df.loc[fire_conditions, 'Fire Risk'] = 1

print("\nFire Risk Analysis:")
print(df[['SoC', 'Temperature', 'Voltage', 'Label', 'Fire Risk']].head(20))
```

```
Fire Risk Analysis:
          SoC  Temperature   Voltage  Label  Fire Risk
0  100.000000   298.150000  4.014300      0          1
1   99.173138   298.849283  3.916820      0          1
2   98.346276   299.665201  3.887562      0          1
3   97.519413   300.497825  3.877287      0          1
4   96.692551   301.327592  3.870545      0          1
5   95.865689   302.139618  3.866940      0          1
6   95.038827   302.930224  3.863284      0          1
7   94.211964   303.701198  3.859399      0          1
8   93.385102   304.454840  3.855101      0          1
```

```
9    92.558240   305.192803  3.850402      0          1
10   91.731378   305.916264  3.845349      0          1
11   90.904515   306.626008  3.840066      0          1
12   90.077653   307.321088  3.834922      0          1
13   89.250791   308.000494  3.829458      0          0
14   88.423929   308.663776  3.823774      0          0
15   87.597066   309.311021  3.817903      0          0
16   86.770204   309.942668  3.811862      0          0
17   85.943342   310.559266  3.805665      0          0
18   85.116480   311.161540  3.799278      0          0
19   84.289617   311.751322  3.792692      0          0
```

Step 5: Train Model to Predict Fire Risk

```
# Separate features and target
X = df[['SoC', 'Temperature', 'Voltage']]
y = df['Fire Risk']


# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


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

Define a Function to Evaluate Models

```
def evaluate_model(model, X_train, y_train, X_test, y_test):
    # Cross-Validation
    cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    print(f"Cross-Validation Accuracy: {np.mean(cv_scores):.4f} (±{np.std(cv_scores):.4f})")

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

    # Test the model
    y_pred = model.predict(X_test)
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print(f"Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print("-" * 60)


# Algorithms to evaluate
models = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Support Vector Machine": SVC(),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42)
}


# Evaluate each model
for name, model in models.items():
    print(f"Evaluating {name}:")
    evaluate_model(model, X_train, y_train, X_test, y_test)
```

```
Evaluating Logistic Regression:
Cross-Validation Accuracy: 0.8698 (±0.0185)
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.96      0.89        53
           1       0.98      0.90      0.94       105

    accuracy                           0.92       158
   macro avg       0.90      0.93      0.91       158
weighted avg       0.93      0.92      0.92       158

Confusion Matrix:
[[51  2]
 [11 94]]
Test Accuracy: 0.9177
------------------------------------------------------------
Evaluating Random Forest:
Cross-Validation Accuracy: 0.9365 (±0.0100)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.98      0.94        53
           1       0.99      0.94      0.97       105

    accuracy                           0.96       158
   macro avg       0.94      0.96      0.95       158
weighted avg       0.96      0.96      0.96       158

Confusion Matrix:
[[52  1]
 [ 6 99]]
Test Accuracy: 0.9557
------------------------------------------------------------
Evaluating Support Vector Machine:
Cross-Validation Accuracy: 0.9032 (±0.0232)
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.96      0.93        53
           1       0.98      0.94      0.96       105

    accuracy                           0.95       158
   macro avg       0.94      0.95      0.94       158
weighted avg       0.95      0.95      0.95       158

Confusion Matrix:
[[51  2]
 [ 6 99]]
Test Accuracy: 0.9494
------------------------------------------------------------
Evaluating Gradient Boosting:
Cross-Validation Accuracy: 0.9302 (±0.0169)
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.98      0.90        53
```

```
# Hyperparameter Tuning with GridSearchCV
# Example: Tuning Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

GridSearchCV

best_estimator_: RandomForestClassifier

RandomForestClassifier

```
# Best parameters and model
print("Best Parameters for Random Forest:")
print(grid_search.best_params_)

best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
print("Classification Report for Best Model:")
print(classification_report(y_test, y_pred_best))
print("Confusion Matrix for Best Model:")
print(confusion_matrix(y_test, y_pred_best))
print(f"Test Accuracy for Best Model: {accuracy_score(y_test, y_pred_best):.4f}")
```

```
Best Parameters for Random Forest:
{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 50}
Classification Report for Best Model:
              precision    recall  f1-score   support

           0       0.90      0.98      0.94        53
           1       0.99      0.94      0.97       105

    accuracy                           0.96       158
   macro avg       0.94      0.96      0.95       158
weighted avg       0.96      0.96      0.96       158

Confusion Matrix for Best Model:
[[52  1]
 [ 6 99]]
```

```
Test Accuracy for Best Model: 0.9557
```

```
# Predict Fire Risk for New Data
new_data = np.array([[95, 400, 3.5]])  # Example input (SoC, Temperature, Voltage)
new_data_scaled = scaler.transform(new_data)
predicted_fire_risk = best_model.predict(new_data_scaled)

# Convert prediction to a meaningful message
if predicted_fire_risk[0] == 1:
    print('\nPredicted Fire Risk: Fire Detected')
else:
    print('\nPredicted Fire Risk: Fire Not Detected')
```

```
Predicted Fire Risk: Fire Detected
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Star
  warnings.warn(
```

```
# Predict Fire Risk for New Data
new_data = np.array([[95, 200, 3.5]])  # Example input (SoC, Temperature, Voltage)
new_data_scaled = scaler.transform(new_data)
predicted_fire_risk = best_model.predict(new_data_scaled)

# Convert prediction to a meaningful message
if predicted_fire_risk[0] == 1:
    print('\nPredicted Fire Risk: Fire Detected')
else:
    print('\nPredicted Fire Risk: Fire Not Detected')
```

```
Predicted Fire Risk: Fire Detected
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Star
  warnings.warn(
```

```
# Predict Fire Risk for New Data
new_data = np.array([[20, 100, 3.5]])  # Example input (SoC, Temperature, Voltage)
new_data_scaled = scaler.transform(new_data)
predicted_fire_risk = best_model.predict(new_data_scaled)

# Convert prediction to a meaningful message
if predicted_fire_risk[0] == 1:
    print('\nPredicted Fire Risk: Fire Detected')
else:
    print('\nPredicted Fire Risk: Fire Not Detected')
```

```
Predicted Fire Risk: Fire Not Detected
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Star
  warnings.warn(
```

Conclusion:

This code performs fire risk prediction for EV batteries based on features like SoC, Temperature, and Voltage. It begins with exploratory data