Machine Learning Project Documentation

Prepared By:
Samer Wael Elbehidy

Here is where My Documentation begins

Implementations & Data Load:

You Must Run The Following Code in case to Run any Model From The Models Below

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, ShuffleSplit
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,accuracy_score, classification_report,confusion_matrix
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv(r"C:\Users\samer\Desktop\Battery_RUL.csv")
data.head()
```

✓ 0.0s

	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Voltage Charg. (V)	Time at 4.15V (s)	Time constant current (s)	Charging time (s)	RUL
0	1.0	2595.30	1151.488500	3.670	3.211	5460.001	6755.01	10777.82	1112
1	2.0	7408.64	1172.512500	4.246	3.220	5508.992	6762.02	10500.35	1111
2	3.0	7393.76	1112.992000	4.249	3.224	5508.993	6762.02	10420.38	1110
3	4.0	7385.50	1080.320667	4.250	3.225	5502.016	6762.02	10322.81	1109
4	6.0	65022.75	29813.487000	4.290	3.398	5480.992	53213.54	56699.65	1107

AGENDA

You Can Choose the model you want to run from these Different Models (Regression And Classification):

Content:

1. KNN-Regression
2. KNN-Classification
3. Linear Regression
4. Logistic Regression
5. Decision Tree Regression
6. Decision Tree Classification
7. Support Vector Regression
8. Support Vector Classification
9. Random Forest Regression
10. Random Forest Classification

KNN Regression :

K-Nearest Neighbors (KNN) Regression is a non-parametric and lazy supervised learning algorithm used for both classification and regression tasks. In KNN regression, the algorithm predicts the target variable by averaging the values of its k-nearest neighbors

```
from sklearn.neighbors import KNeighborsRegressor  # Import KNeighborsRegressor

# Data preprocessing... (Preprocessing steps here):

# Split the data into features (X) and target variable (y)
y = data['RUL']
X = data.drop(['RUL'], axis=1)

# Split the 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=900)

# Dropping all the NaN values
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print("NAN values have been dropped successfully \n")

# Check if there are NaN values in the features
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())
```

KNN Regression :

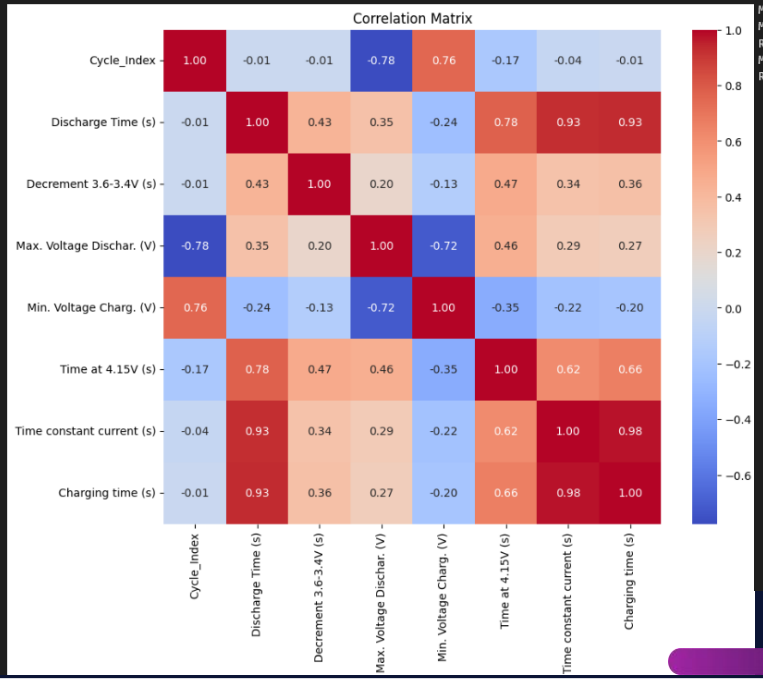
```
# Scale features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize KNN Regressor and fit the model
knn_regressor = KNeighborsRegressor(n_neighbors=3)
knn_regressor.fit(X_train_scaled, y_train)

# Draw a correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X_train_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

# Apply cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=42)
cv_scores = cross_val_score(knn_regressor, X_train_scaled, y_train, cv=cv, scoring='r2')  # Use 'r2' for regression

# Print cross-validation scores
print("Cross-Validation R^2 Scores:", cv_scores)
print("Mean R^2: {:.2f}".format(cv_scores.mean()))

# Make predictions
y_pred = knn_regressor.predict(X_test_scaled)
```

KNN Regression :

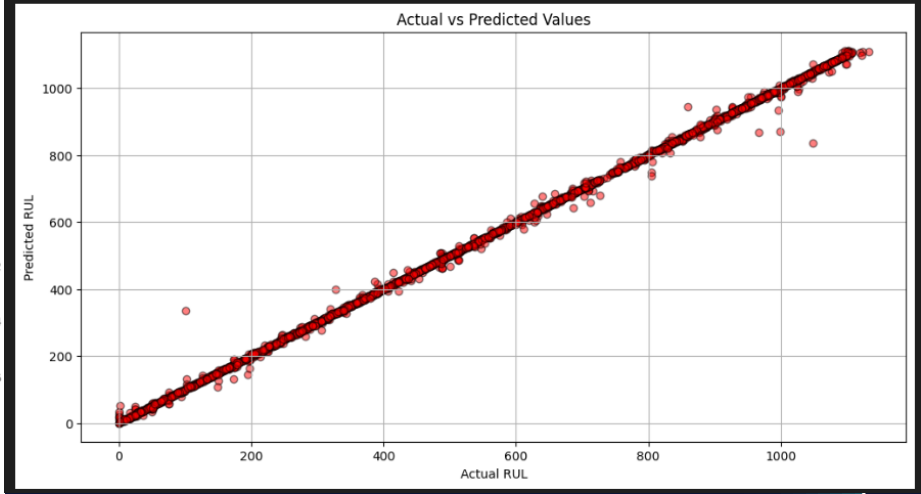
```
# Evaluate the model
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Absolute Error: {mae}")
print(f"Root Mean Squared Error: {rmse}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R^2 Score: {r2:.2f}")

# Scatter plot for actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, alpha=0.5, c="red", edgecolors='k')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual RUL')
plt.ylabel('Predicted RUL')
plt.grid(True)
plt.show()
```

KNN Regression Results :

```
NAN values have been dropped successfully

NaN values in y_train: 0
NaN values in y_test: 0
NaN values in X_train: 0
NaN values in X_test: 0
```




```
Cross-Validation R^2 Scores: [0.99933774 0.99713302 0.99823544 0.99939216 0.99892179]
Mean R^2: 1.00
Mean Absolute Error: 2.870229007633587
Root Mean Squared Error: 8.848309668400189
Mean Squared Error: 78.29
R^2 Score: 1.00
```



KNN Classification :

K-Nearest Neighbors (KNN) Classifier is a simple and intuitive machine learning algorithm used for classification tasks. It belongs to the family of lazy or instance-based learning algorithms.

```
from sklearn.neighbors import KNeighborsClassifier  # Import KNeighborsClassifier

# Data preprocessing... (Preprocessing steps here):

# Define bins for classification
bins = [0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, float('inf')]
# Create labels for each bin
labels = ['0-100', '100-200', '200-300', '300-400', '400-500', '500-600', '600-700', '700-800', '800-900', '900-1000', '1000-1100', '1100-inf']

# Add a new column 'RUL_bin' with the bin labels
data['RUL_bin'] = pd.cut(data['RUL'], bins=bins, labels=labels)

# Split the data into features (X) and target variable (y)
y = data['RUL_bin']
X = data.drop(['RUL', 'RUL_bin'], axis=1)

# Split the 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=900)

# Check for NaN values in the data
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

# Drop NaN values in the data
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print ("NAN values has been dropped successfully \n")

# Check if there are other NaN values in the data
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())
```

KNN Classification :

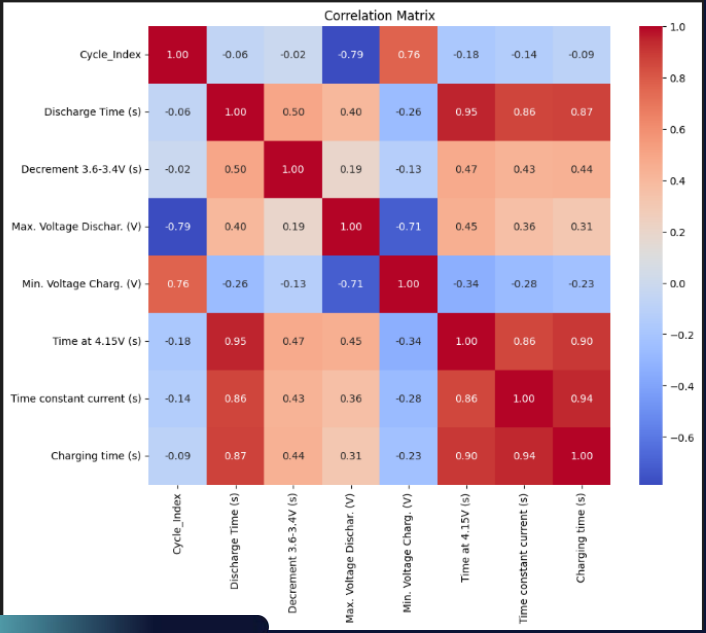
```
# Drop NaN values in the data
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print ("NAN values has been dropped successfully \n")

# Check if there are other NaN values in the data
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

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

# Initialize KNN Classifier and fit the model
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train_scaled, y_train)

# draw a correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X_train_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

# Apply cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=42)
cv_scores = cross_val_score(knn_classifier, X_train_scaled, y_train, cv=cv, scoring='accuracy')

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy: {:.2%}".format(cv_scores.mean()))
```

KNN Classification :

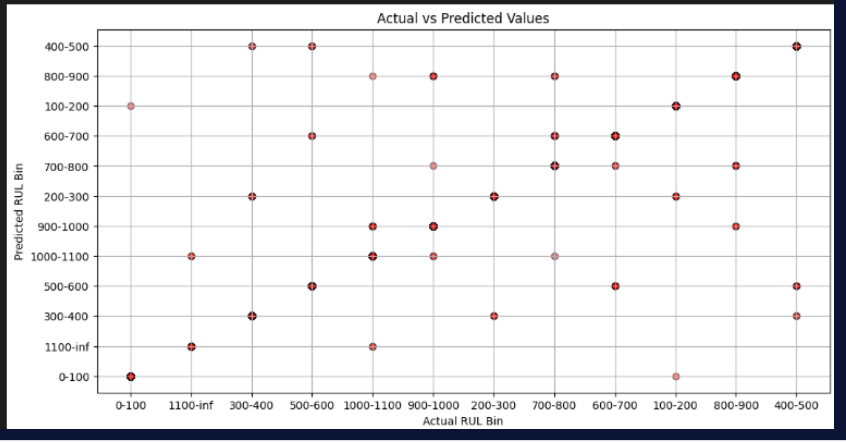
```
# Make predictions
y_pred = knn_classifier.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2%}")

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Scatter plot for actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, alpha=0.5, c="red", edgecolors='k')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual RUL Bin')
plt.ylabel('Predicted RUL Bin')
plt.grid(True)
plt.show()

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Plot confusion matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

KNN Classification results:

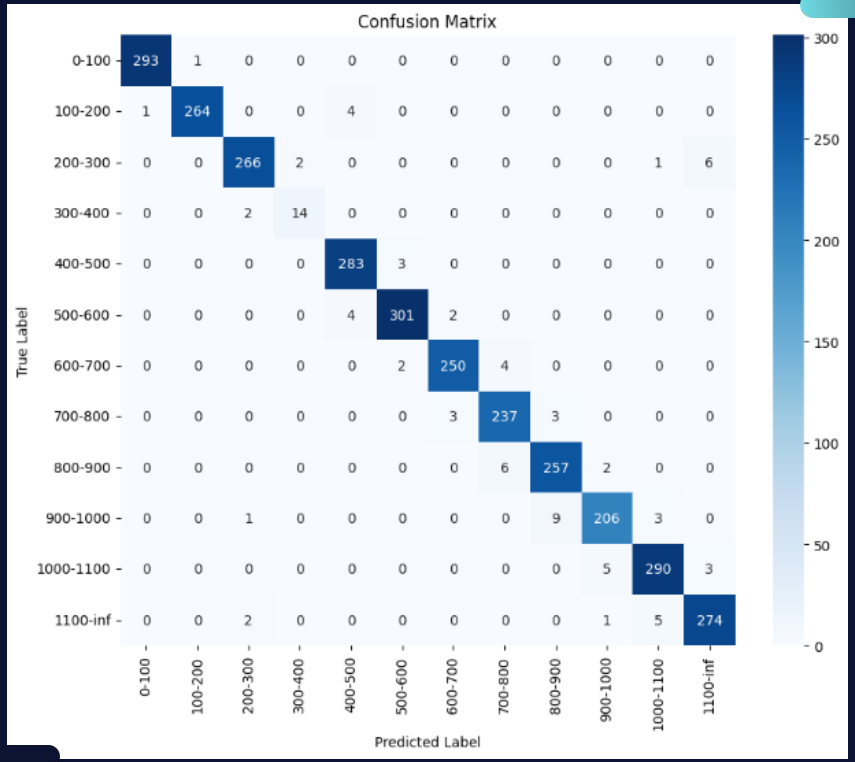
```
NaN values in y_train: 11
NaN values in y_test: 3
NaN values in X_train: 0
NaN values in X_test: 0
NAN values has been dropped successfully

NaN values in y_train: 0
NaN values in y_test: 0
NaN values in X_train: 0
NaN values in X_test: 0
```




```
Cross-Validation Scores: [0.96760797 0.97591362 0.97508306 0.9717608  0.97342193]
Mean Accuracy: 97.28%
Accuracy: 97.51%
Classification Report:
              precision    recall  f1-score   support

       0-100       1.00      1.00      1.00       294
     100-200       1.00      0.98      0.99       269
    1000-1100      0.98      0.97      0.97       275
    1100-inf       0.88      0.88      0.88        16
     200-300       0.97      0.99      0.98       286
     300-400       0.98      0.98      0.98       307
     400-500       0.98      0.98      0.98       256
     500-600       0.96      0.98      0.97       243
     600-700       0.96      0.97      0.96       265
     700-800       0.96      0.94      0.95       219
     800-900       0.97      0.97      0.97       298
    900-1000       0.97      0.97      0.97       282

    accuracy                           0.98      3010
   macro avg       0.97      0.97      0.97      3010
weighted avg       0.98      0.98      0.98      3010
```



KNN Classification results:



Linear Regression :

Linear Regression is a fundamental statistical and machine learning technique used for predicting a continuous target variable based on one or more predictor variables.

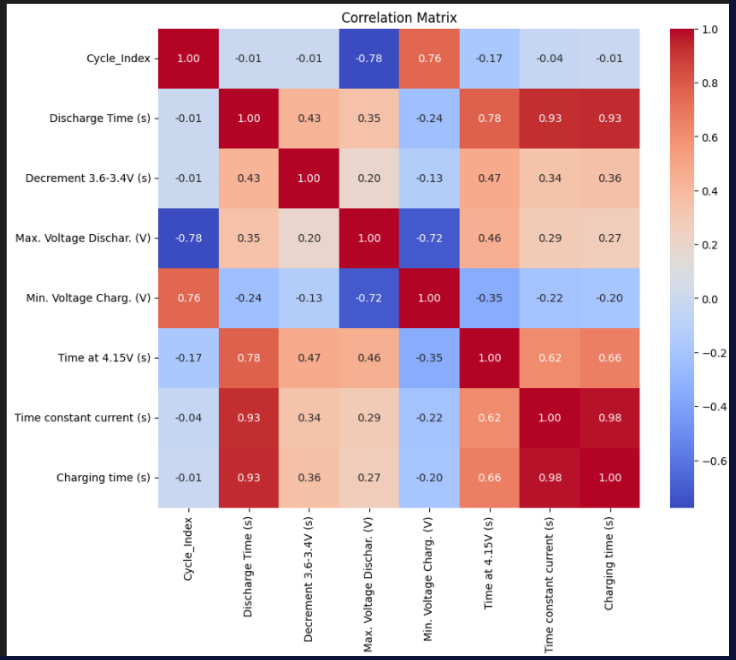
```python
from sklearn.linear_model import LinearRegression  # Import LinearRegression

# Data preprocessing... (Preprocessing steps here):

# Split the data into features (X) and target variable (y)
y = data['RUL']
X = data.drop(['RUL'], axis=1)

# Split the 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=900)

# Dropping all the NaN values
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print("NAN values have been dropped successfully \n")

# Check if there are NaN values in the features
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

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

# Initialize Linear Regression and fit the model
linear_regressor = LinearRegression()
linear_regressor.fit(X_train_scaled, y_train)

# draw a correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X_train_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

Linear Regression :

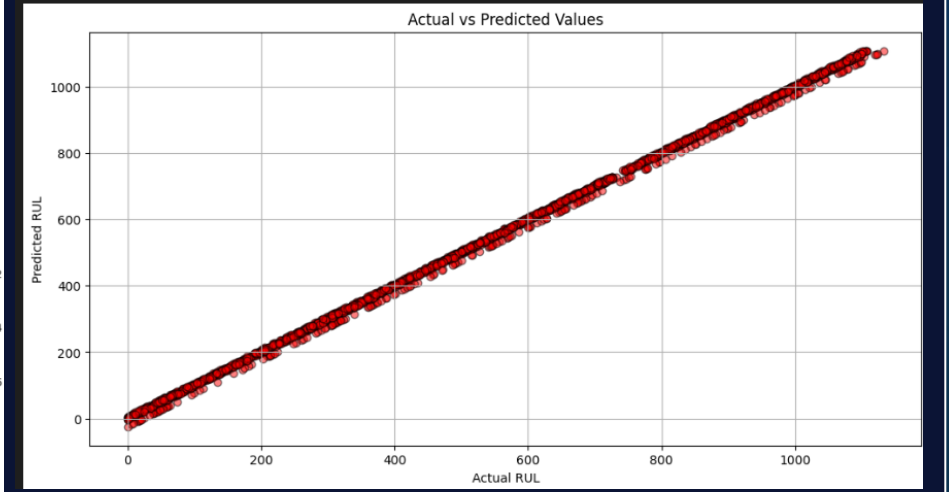
```
# Apply cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=42)
cv_scores = cross_val_score(linear_regressor, X_train_scaled, y_train, cv=cv, scoring='r2')

# Print cross-validation scores
print("Cross-Validation R^2 Scores:", cv_scores)
print("Mean R^2: {:.2f}".format(cv_scores.mean()))

# Make predictions
y_pred = linear_regressor.predict(X_test_scaled)

# Evaluate the model
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Absolute Error: {mae}")
print(f"Root Mean Squared Error: {rmse}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R^2 Score: {r2:.2f}")

# Scatter plot for actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, alpha=0.5, c="red", edgecolors='k')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual RUL')
plt.ylabel('Predicted RUL')
plt.grid(True)
plt.show()
```

Linear Regression results:

```
NAN values have been dropped successfully

NaN values in y_train: 0
NaN values in y_test: 0
NaN values in X_train: 0
NaN values in X_test: 0
```




```
Cross-Validation R^2 Scores: [0.99949358 0.99939785 0.99946899 0.99957554 0.99953931]
Mean R^2: 1.00
Mean Absolute Error: 4.415000255561129
Root Mean Squared Error: 6.9648977077278404
Mean Squared Error: 48.51
R^2 Score: 1.00
```



Logistic Regression :

Logistic Regression is a statistical and machine learning model used for binary and multi-class classification tasks. Despite its name, it is primarily employed for classification rather than regression.

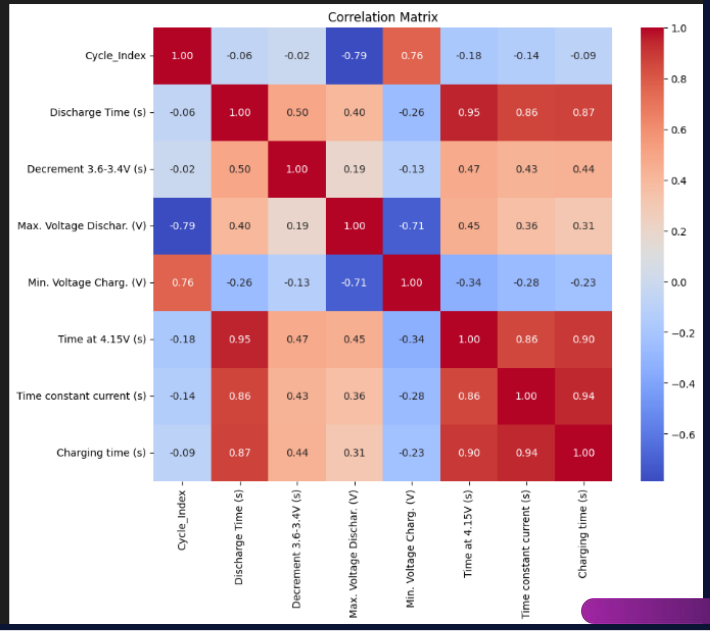
```python
from sklearn.linear_model import LogisticRegression  # Import LogisticRegression

# Data preprocessing... (Preprocessing steps here):

# Define bins for classification
bins = [0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, float('inf')]
# Create labels for each bin
labels = ['0-100', '100-200', '200-300', '300-400', '400-500', '500-600', '600-700', '700-800', '800-900', '900-1000', '1000-1100', '1100-inf']

# Add a new column 'RUL_bin' with the bin labels
data['RUL_bin'] = pd.cut(data['RUL'], bins=bins, labels=labels)

# Split the data into features (X) and target variable (y)
y = data['RUL_bin']
X = data.drop(['RUL', 'RUL_bin'], axis=1)

# Split the 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=900)

# Check for NaN values in the data
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

# Drop NaN values in the data
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print("NAN values have been dropped successfully \n")

# Check if there are NaN values in the features
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

# Scale features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Logistic Regression :

```
# Initialize Logistic Regression and fit the model
logistic_regressor = LogisticRegression(max_iter=1000, random_state=42)
logistic_regressor.fit(X_train_scaled, y_train)

# Draw a correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X_train_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

# Apply cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=42)
cv_scores = cross_val_score(logistic_regressor, X_train_scaled, y_train, cv=cv, scoring='accuracy')

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy: {:.2%}".format(cv_scores.mean()))

# Make predictions
y_pred = logistic_regressor.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2%}")

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Scatter plot for actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, alpha=0.5, c="red", edgecolors='k')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual RUL Bin')
plt.ylabel('Predicted RUL Bin')
plt.grid(True)
plt.show()

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Plot confusion matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

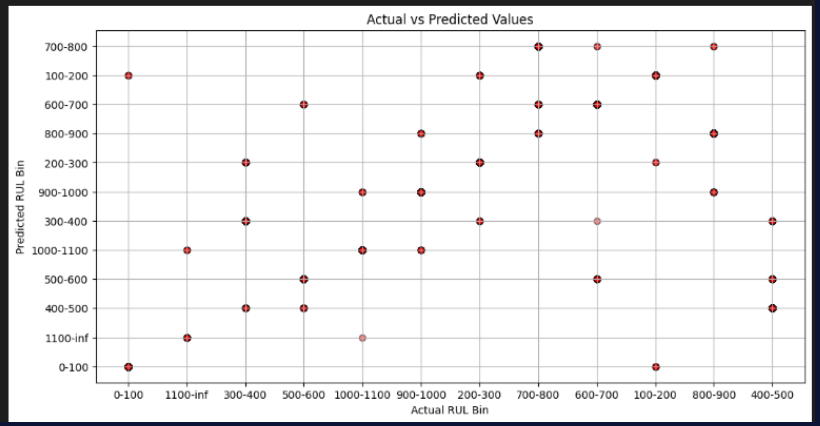
Logistic Regression results:

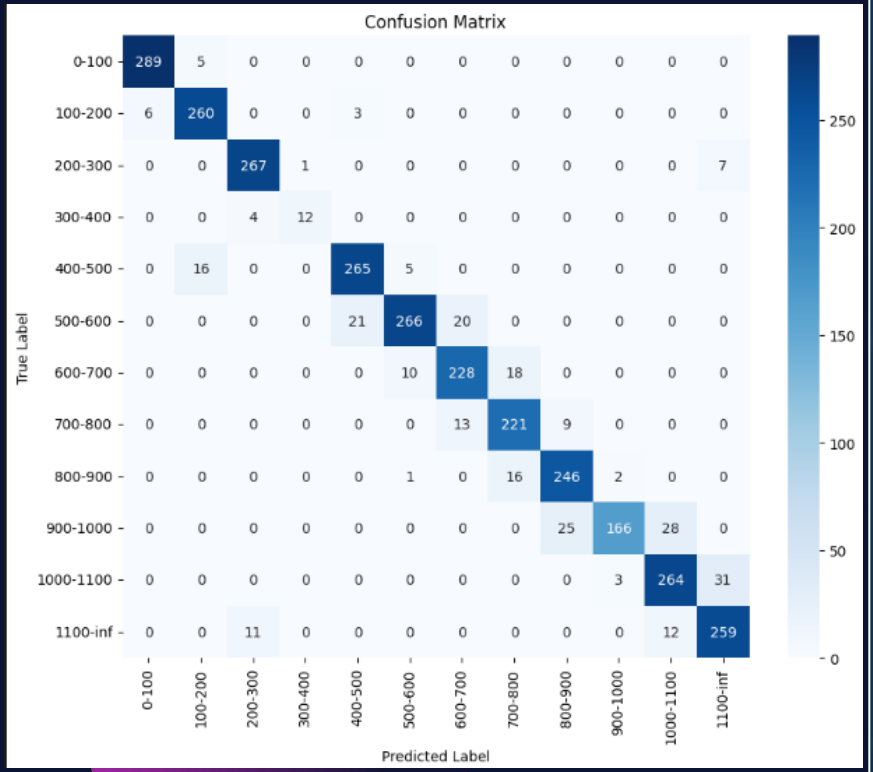
```
NaN values in y_train: 11
NaN values in y_test: 3
NaN values in X_train: 0
NaN values in X_test: 0
NAN values have been dropped successfully

NaN values in y_train: 0
NaN values in y_test: 0
NaN values in X_train: 0
NaN values in X_test: 0
```



Logistic Regression results:

```
Cross-Validation Scores: [0.89784053 0.90614618 0.90614618 0.91694352 0.90282392]
Mean Accuracy: 90.60%
Accuracy: 91.13%
Classification Report:
              precision    recall  f1-score   support

       0-100       0.98      0.98      0.98       294
     100-200       0.93      0.97      0.95       269
    1000-1100      0.95      0.97      0.96       275
    1100-inf       0.92      0.75      0.83        16
     200-300       0.92      0.93      0.92       286
     300-400       0.94      0.87      0.90       307
     400-500       0.87      0.89      0.88       256
     500-600       0.87      0.91      0.89       243
     600-700       0.88      0.93      0.90       265
     700-800       0.97      0.76      0.85       219
     800-900       0.87      0.89      0.88       298
    900-1000       0.87      0.92      0.89       282

    accuracy                           0.91      3010
   macro avg       0.91      0.90      0.90      3010
weighted avg       0.91      0.91      0.91      3010
```





Decision Tree Regression :

Decision Tree Regression is a machine learning algorithm used for predicting continuous numerical values. Unlike its classification counterpart, decision trees for regression tasks aim to predict a real-valued output rather than a categorical one.

```python
from sklearn.tree import DecisionTreeRegressor, plot_tree  # Import DecisionTreeRegressor and plot_tree

# Data preprocessing... (Preprocessing steps here):

# Split the data into features (X) and target variable (y)
y = data['RUL']
X = data.drop(['RUL'], axis=1)

# Split the 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=900)

# Dropping all the NaN values
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print("NAN values have been dropped successfully \n")

# Check if there are NaN values in the features
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

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

# Draw a correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X_train_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

# Initialize Decision Tree Regressor and fit the model
decision_tree_regressor = DecisionTreeRegressor(random_state=42)
decision_tree_regressor.fit(X_train_scaled, y_train)
```

Decision Tree Regression :

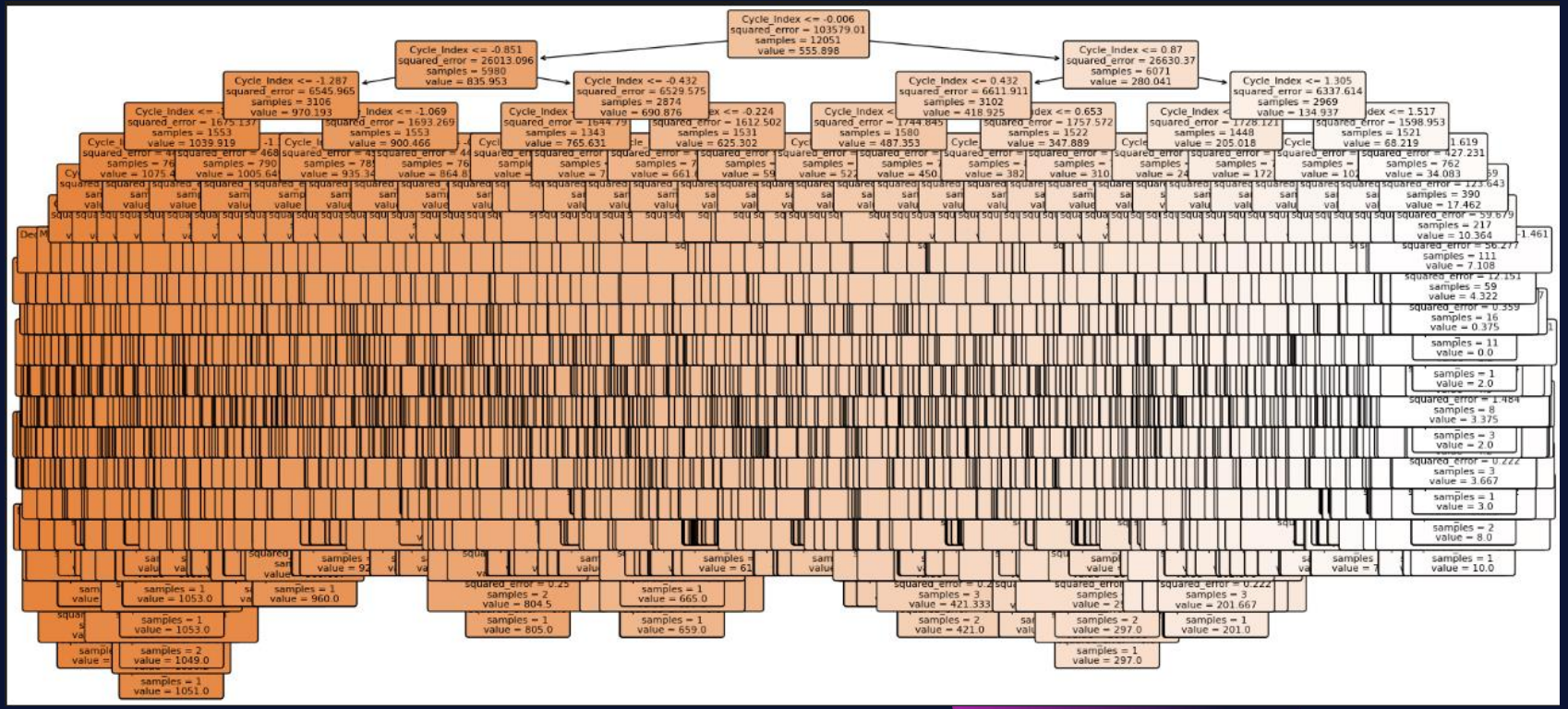
```python
# Plot the decision tree
plt.figure(figsize=(20, 10))
plot_tree(decision_tree_regressor, filled=True, feature_names=X.columns, rounded=True, fontsize=8)
plt.show()

# Apply cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=42)
cv_scores = cross_val_score(decision_tree_regressor, X_train_scaled, y_train, cv=cv, scoring='r2')

# Print cross-validation scores
print("Cross-Validation R^2 Scores:", cv_scores)
print("Mean R^2: {:.2f}".format(cv_scores.mean()))

# Make predictions
y_pred = decision_tree_regressor.predict(X_test_scaled)

# Evaluate the model
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Absolute Error: {mae}")
print(f"Root Mean Squared Error: {rmse}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R^2 Score: {r2:.2f}")

# Scatter plot for actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, alpha=0.5, c="red", edgecolors='k')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual RUL')
plt.ylabel('Predicted RUL')
plt.grid(True)
plt.show()
```

```
(variable) y_train: Any
```

Decision Tree Regression results:

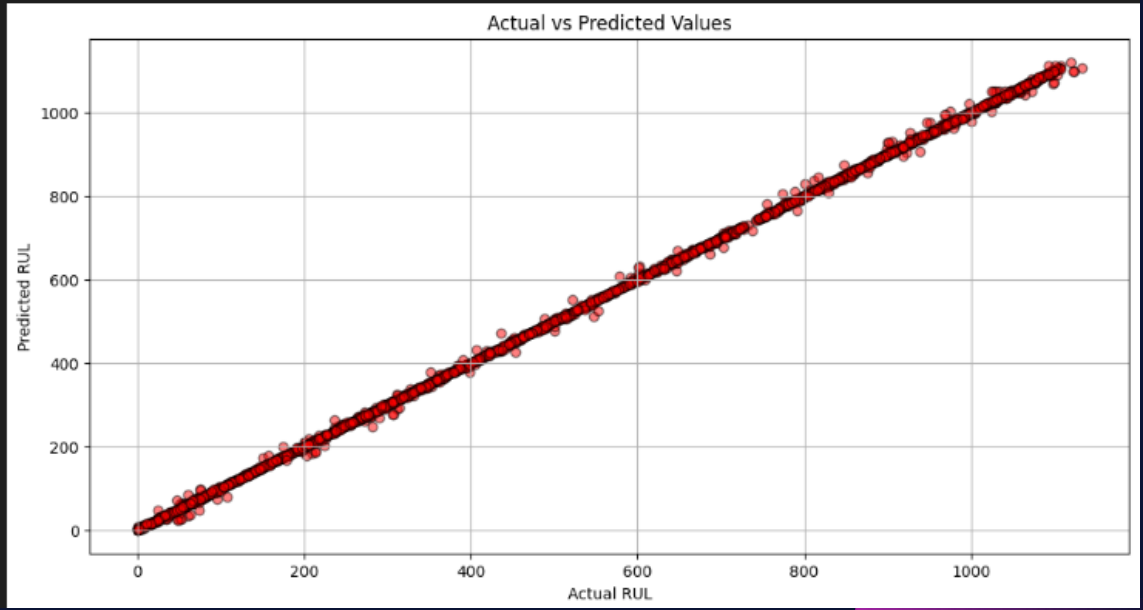
```
NAN values have been dropped successfully

NaN values in y_train: 0
NaN values in y_test: 0
NaN values in X_train: 0
NaN values in X_test: 0
```

Decision Tree Regression results:



Decision Tree Regression results:

```
Cross-Validation R^2 Scores: [0.99972548 0.99972792 0.99979577 0.99974943 0.99976697]
Mean R^2: 1.00
Mean Absolute Error: 2.128775307002987
Root Mean Squared Error: 4.877252005224797
Mean Squared Error: 23.79
R^2 Score: 1.00
```



Decision Tree Classification :

Decision Tree Classification is a machine learning algorithm used for predicting categorical outcomes or class labels. It is widely employed in various domains for its simplicity and interpretability.

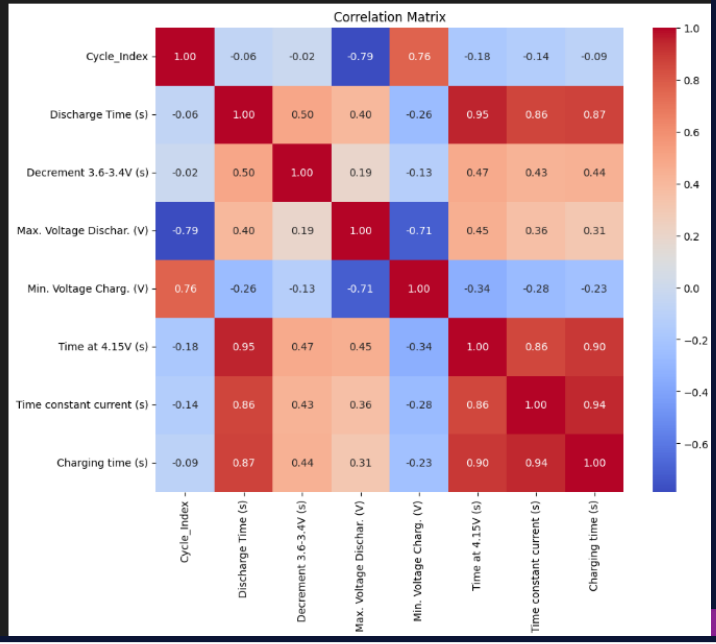
```
from sklearn.tree import DecisionTreeClassifier, plot_tree  # Import DecisionTreeClassifier and plot_tree

# Data preprocessing... (Preprocessing steps here):

# Define bins for classification
bins = [0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, float('inf')]
# Create labels for each bin
labels = ['0-100', '100-200', '200-300', '300-400', '400-500', '500-600', '600-700', '700-800', '800-900', '900-1000', '1000-1100', '1100-inf']

# Add a new column 'RUL_bin' with the bin labels
data['RUL_bin'] = pd.cut(data['RUL'], bins=bins, labels=labels)

# Split the data into features (X) and target variable (y)
y = data['RUL_bin']
X = data.drop(['RUL', 'RUL_bin'], axis=1)

# Split the 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=900)

# Check for NaN values in the data
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

# Drop NaN values in the target variable
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print("NaN values have been dropped successfully \n")

# Check if there are NaN values in the features
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

# Scale features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Decision Tree Classification :

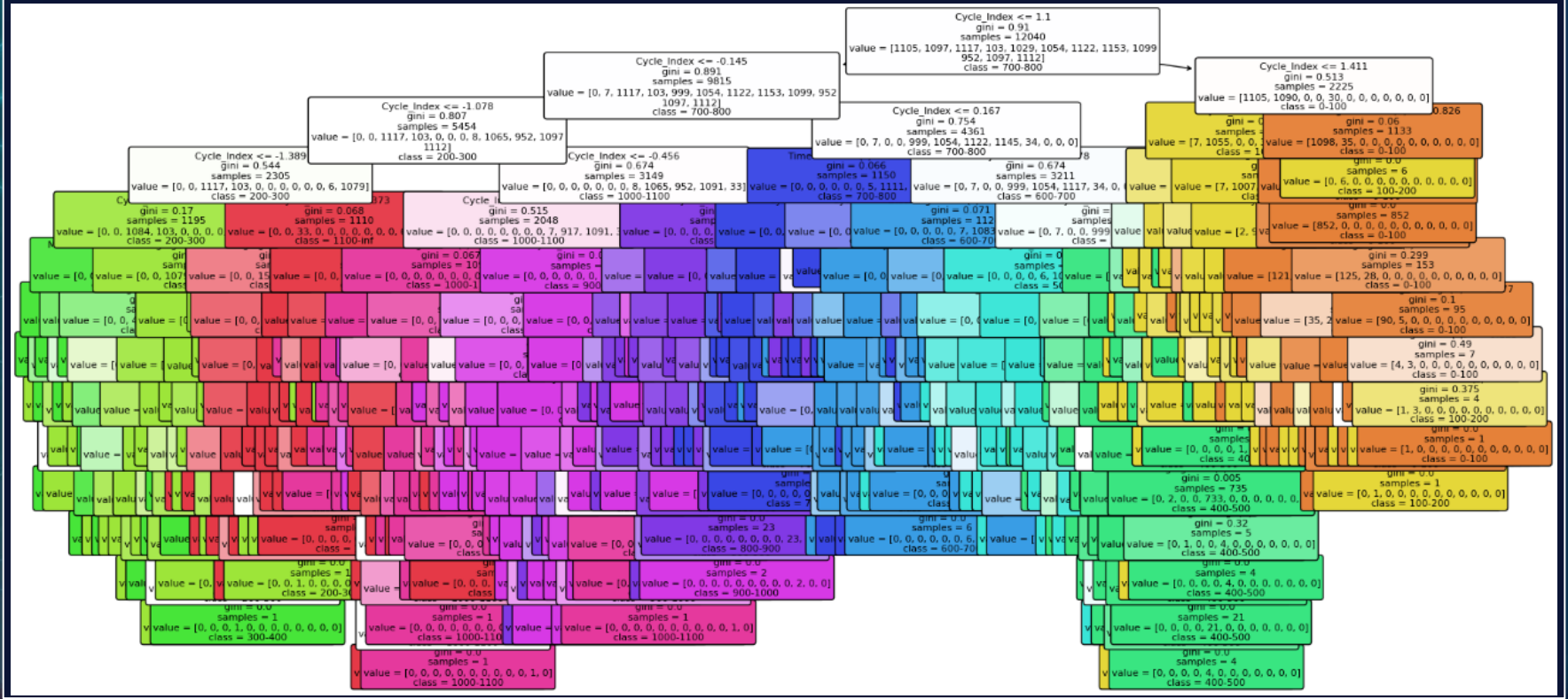
```
# draw a correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X_train_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

# Initialize Decision Tree Classifier and fit the model
decision_tree_classifier = DecisionTreeClassifier(random_state=42)
decision_tree_classifier.fit(X_train_scaled, y_train)

# Plot the decision tree
plt.figure(figsize=(20, 10))
plot_tree(decision_tree_classifier, filled=True, feature_names=X.columns, class_names=labels, rounded=True, fontsize=8)
plt.show()

# Apply cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=42)
cv_scores = cross_val_score(decision_tree_classifier, X_train_scaled, y_train, cv=cv, scoring='accuracy')

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy: {:.2%}".format(cv_scores.mean()))

# Make predictions
y_pred = decision_tree_classifier.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2%}")

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Plot confusion matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Decision Tree Classification results:

```
NaN values in y_train: 11
NaN values in y_test: 3
NaN values in X_train: 0
NaN values in X_test: 0
NaN values have been dropped successfully

NaN values in y_train: 0
NaN values in y_test: 0
NaN values in X_train: 0
NaN values in X_test: 0
```

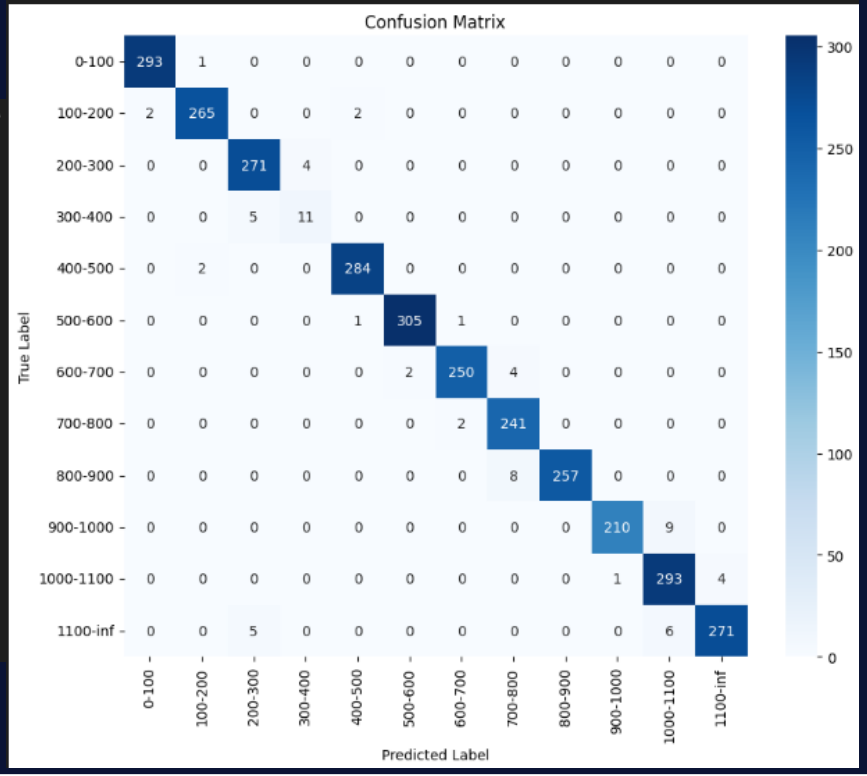


Decision Tree Classification results:



Decision Tree Classification results:

```
Cross-Validation Scores: [0.98255814 0.98671096 0.98255814 0.97757475 0.9858804
Mean Accuracy: 98.31%
Accuracy: 98.04%
Classification Report:
              precision    recall  f1-score   support

       0-100       0.99      1.00      0.99       294
     100-200       0.99      0.99      0.99       269
    1000-1100      0.96      0.99      0.97       275
    1100-inf       0.73      0.69      0.71        16
     200-300       0.99      0.99      0.99       286
     300-400       0.99      0.99      0.99       307
     400-500       0.99      0.98      0.98       256
     500-600       0.95      0.99      0.97       243
     600-700       1.00      0.97      0.98       265
     700-800       1.00      0.96      0.98       219
     800-900       0.95      0.98      0.97       298
    900-1000       0.99      0.96      0.97       282

    accuracy                           0.98      3010
   macro avg       0.96      0.96      0.96      3010
weighted avg       0.98      0.98      0.98      3010
```



Support Vector Regression :

Support Vector Regression is a supervised machine learning algorithm used for regression tasks. It belongs to the family of Support Vector Machines (SVM) and is particularly effective in capturing complex relationships in data.

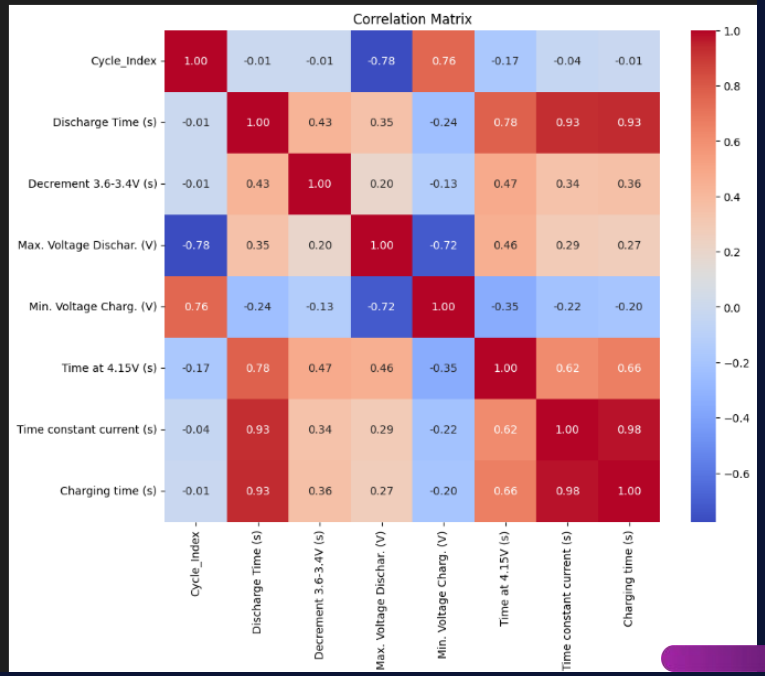
```python
from sklearn.svm import SVR  # Import SVR

# Data preprocessing... (Preprocessing steps here):

# Split the data into features (X) and target variable (y)
y = data['RUL']
X = data.drop(['RUL'], axis=1)

# Split the 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=900)

# Dropping all the NaN values
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print("NAN values have been dropped successfully \n")

# Check if there are NaN values in the features
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

# Scale features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Support Vector Regression :

```python
# Draw a correlation matrix before fitting the SVR
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X_train_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

# Initialize Support Vector Regressor (SVR) and fit the model
svr_regressor = SVR()
svr_regressor.fit(X_train_scaled, y_train)

# Apply cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=42)
cv_scores = cross_val_score(svr_regressor, X_train_scaled, y_train, cv=cv, scoring='r2')  # Use 'r2' for regression

# Print cross-validation scores
print("Cross-Validation R^2 Scores:", cv_scores)
print("Mean R^2: {:.2f}".format(cv_scores.mean()))

# Make predictions
y_pred = svr_regressor.predict(X_test_scaled)

# Evaluate the model
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Absolute Error: {mae}")
print(f"Root Mean Squared Error: {rmse}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R^2 Score: {r2:.2f}")

# Scatter plot for actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, alpha=0.5, c="red", edgecolors='k')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual RUL')
plt.ylabel('Predicted RUL')
plt.grid(True)
plt.show()
```

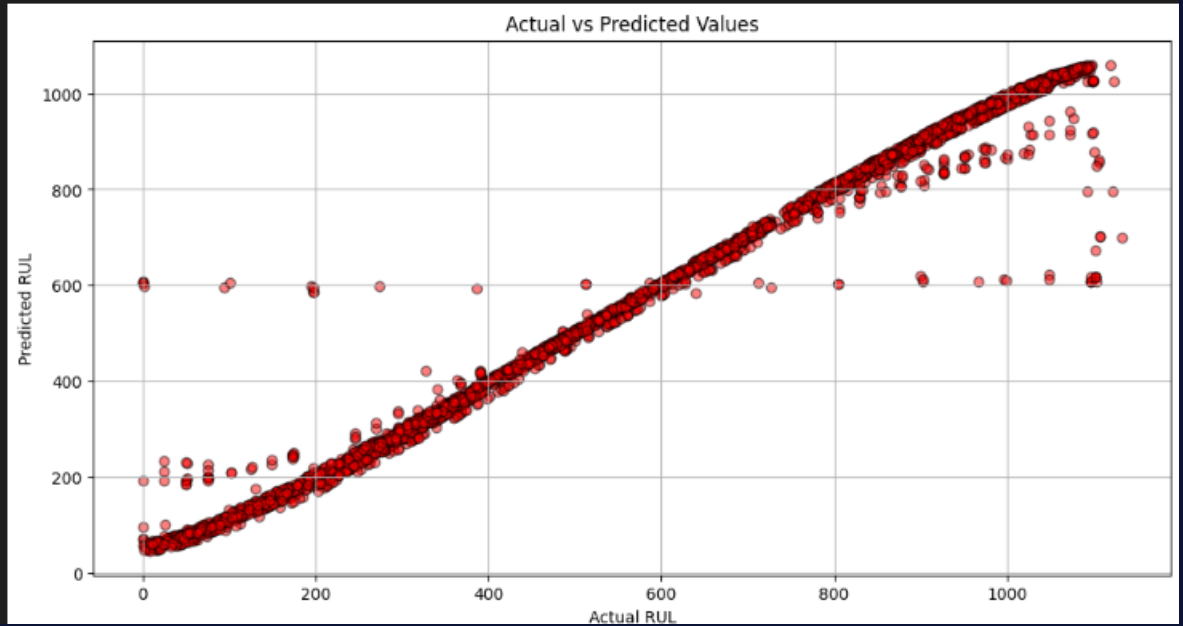
Support Vector Regression results :

```
NAN values have been dropped successfully

NaN values in y_train: 0
NaN values in y_test: 0
NaN values in X_train: 0
NaN values in X_test: 0
```



Support Vector Regression results :

```
Cross-Validation R^2 Scores: [0.97417263 0.96535508 0.98039194 0.97363283 0.9786913 ]
Mean R^2: 0.97
Mean Absolute Error: 19.736672974608304
Root Mean Squared Error: 54.79687239227916
Mean Squared Error: 3002.70
R^2 Score: 0.97
```



Support Vector Classification :

Support Vector Classification, commonly known as Support Vector Machine (SVM), is a powerful machine learning algorithm used for classification tasks.

```
from sklearn.svm import SVC  # Import SVC for classification

# Data preprocessing... (Preprocessing steps here):

# Define bins for classification
bins = [0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, float('inf')]
# Create labels for each bin
labels = ['0-100', '100-200', '200-300', '300-400', '400-500', '500-600', '600-700', '700-800', '800-900', '900-1000', '1000-1100', '1100-inf']

# Add a new column 'RUL_bin' with the bin labels
data['RUL_bin'] = pd.cut(data['RUL'], bins=bins, labels=labels)

# Split the data into features (X) and target variable (y)
y = data['RUL_bin']
X = data.drop(['RUL', 'RUL_bin'], axis=1)

# Split the 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=900)

# Check for NaN values in the target variable (y_train and y_test)
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

# Drop NaN values in the target variable
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print("NAN values have been dropped successfully \n")

# Check if there are NaN values in the features
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())
```

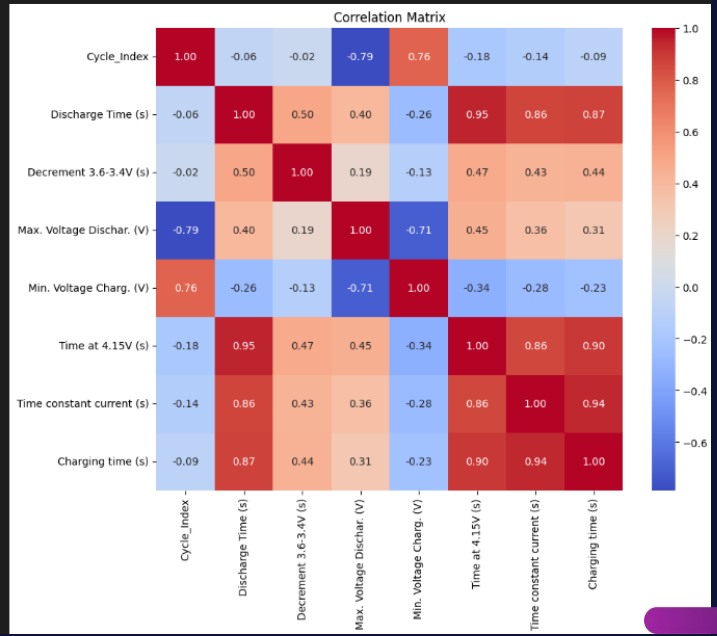
Support Vector Classification :

```python
# Scale features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize Support Vector Classifier (SVC) and fit the model
svc_classifier = SVC()
svc_classifier.fit(X_train_scaled, y_train)

# draw a correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X_train_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm',
plt.title('Correlation Matrix')
plt.show()

# Apply cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=42)
cv_scores = cross_val_score(svc_classifier, X_train_scaled, y_train, cv=cv, scoring='accuracy')

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy: {:.2%}".format(cv_scores.mean()))

# Make predictions
y_pred = svc_classifier.predict(X_test_scaled)
```

```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2%}")

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Scatter plot for actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, alpha=0.5, c="red", edgecolors='k')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual RUL Bin')
plt.ylabel('Predicted RUL Bin')
plt.grid(True)
plt.show()

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Plot confusion matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

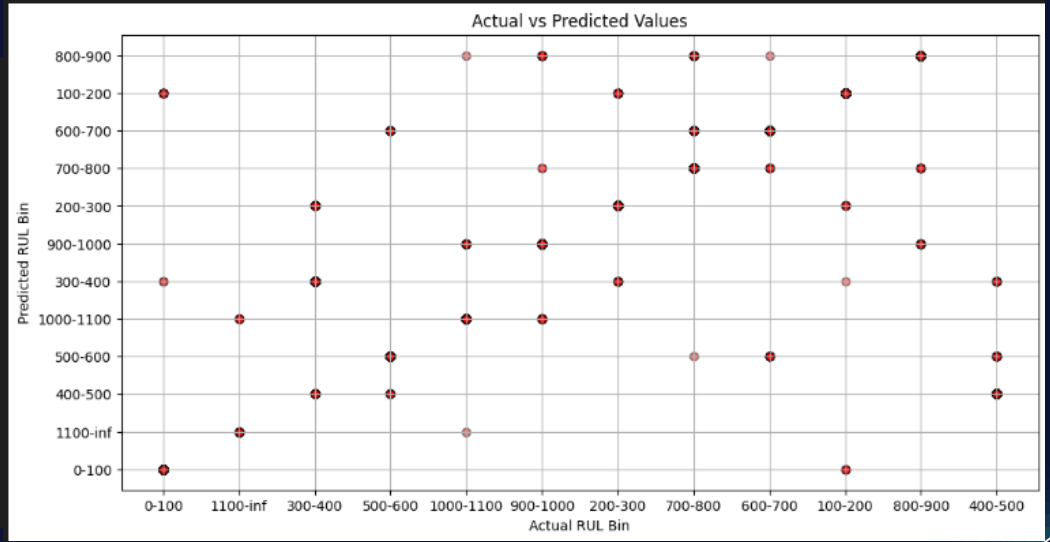
Support Vector Classification results :

```
NaN values in y_train: 11
NaN values in y_test: 3
NaN values in X_train: 0
NaN values in X_test: 0
NAN values have been dropped successfully

NaN values in y_train: 0
NaN values in y_test: 0
NaN values in X_train: 0
NaN values in X_test: 0
```

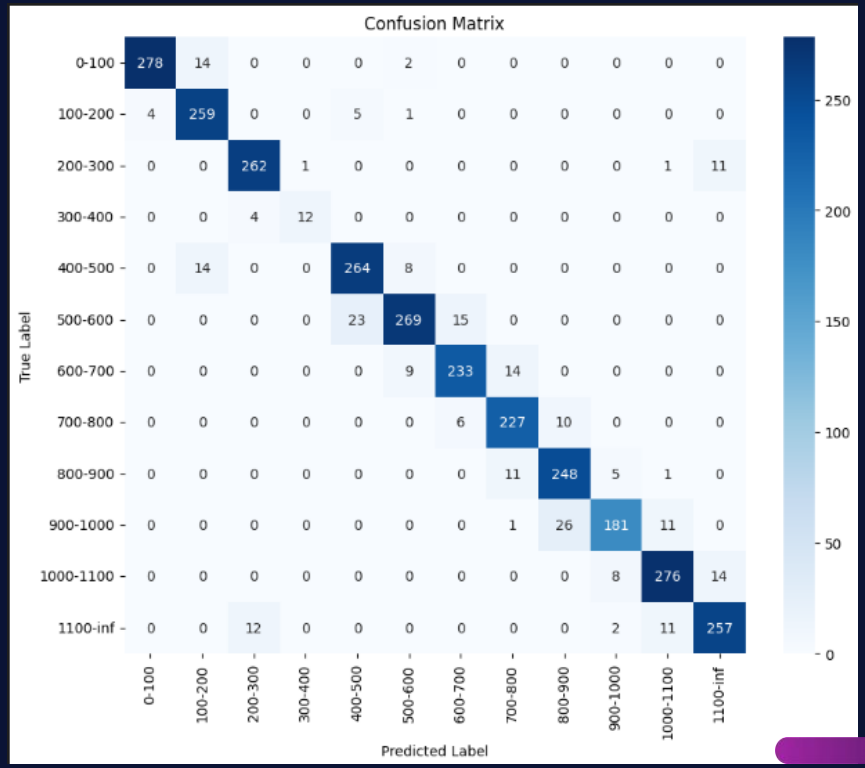


Support Vector Classification results :

```
Cross-Validation Scores: [0.90780731 0.9244186  0.92192691 0.92192691 0.91528239]
Mean Accuracy: 91.83%
Accuracy: 91.89%
Classification Report:
              precision    recall  f1-score   support

       0-100       0.99      0.95      0.97       294
     100-200       0.90      0.96      0.93       269
    1000-1100      0.94      0.95      0.95       275
    1100-inf       0.92      0.75      0.83        16
     200-300       0.90      0.92      0.91       286
     300-400       0.93      0.88      0.90       307
     400-500       0.92      0.91      0.91       256
     500-600       0.90      0.93      0.92       243
     600-700       0.87      0.94      0.90       265
     700-800       0.92      0.83      0.87       219
     800-900       0.92      0.93      0.92       298
    900-1000       0.91      0.91      0.91       282

    accuracy                           0.92      3010
   macro avg       0.92      0.90      0.91      3010
weighted avg       0.92      0.92      0.92      3010
```



Support Vector Classification results :



Random Forest Regression :

Random Forest Regression is an ensemble learning algorithm used for making predictions in regression tasks. It builds multiple decision trees and combines their outputs to provide a more accurate and robust prediction.

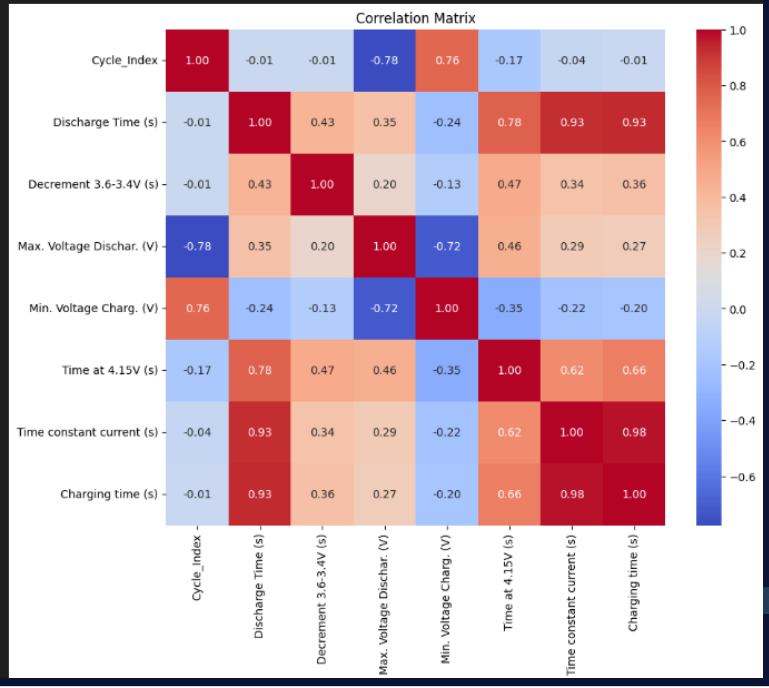
```python
from sklearn.ensemble import RandomForestRegressor  # Import RandomForestRegressor

# Data preprocessing... (Preprocessing steps here):

# Split the data into features (X) and target variable (y)
y = data['RUL']
X = data.drop(['RUL'], axis=1)

# Split the 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=900)

# Dropping all the NaN values
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print("NAN values have been dropped successfully \n")

# Check if there are NaN values in the features
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

# Scale features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Random Forest Regression :

```
# Draw a correlation matrix before fitting the Random Forest Regressor
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X_train_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

# Initialize Random Forest Regressor and fit the model
random_forest_regressor = RandomForestRegressor(random_state=42)
random_forest_regressor.fit(X_train_scaled, y_train)

# Apply cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=42)
cv_scores = cross_val_score(random_forest_regressor, X_train_scaled, y_train, cv=cv, scoring='r2')

# Print cross-validation scores
print("Cross-Validation R^2 Scores:", cv_scores)
print("Mean R^2: {:.2f}".format(cv_scores.mean()))

# Make predictions
y_pred = random_forest_regressor.predict(X_test_scaled)

# Evaluate the model
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Absolute Error: {mae}")
print(f"Root Mean Squared Error: {rmse}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R^2 Score: {r2:.2f}")

# Scatter plot for actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, alpha=0.5, c="red", edgecolors='k')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual RUL')
plt.ylabel('Predicted RUL')
plt.grid(True)
plt.show()
```

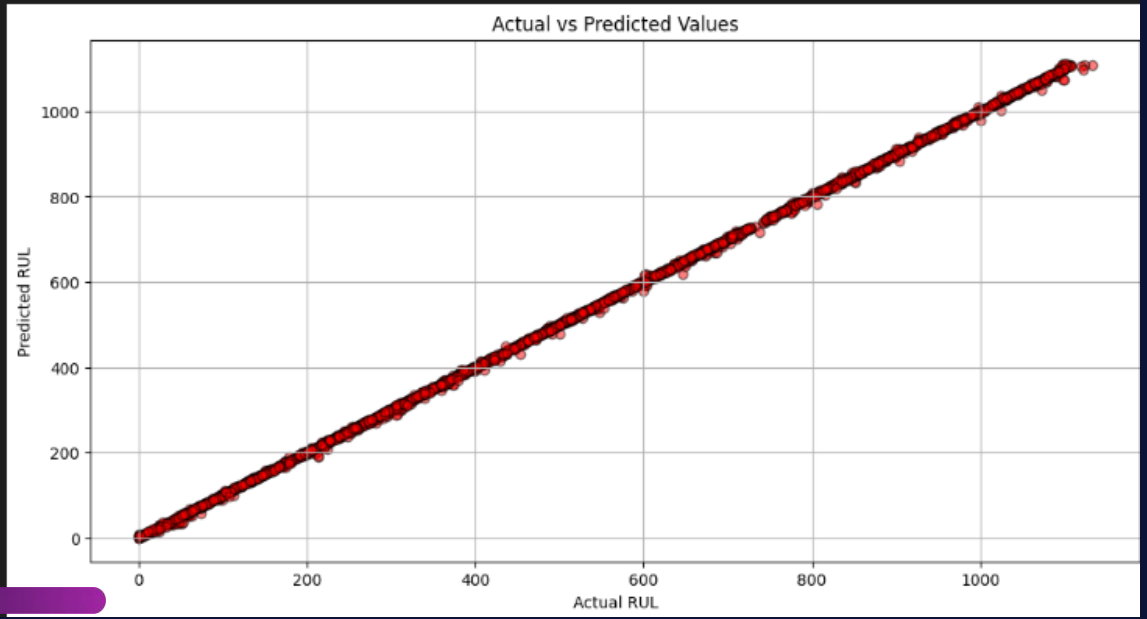
Random Forest Regression results:

```
NAN values have been dropped successfully

NaN values in y_train: 0
NaN values in y_test: 0
NaN values in X_train: 0
NaN values in X_test: 0
```



Random Forest Regression results:

```
Cross-Validation R^2 Scores: [0.99985702 0.99983745 0.99984962 0.99988944 0.99987624]
Mean R^2: 1.00
Mean Absolute Error: 1.9283770328576166
Root Mean Squared Error: 3.443044822563677
Mean Squared Error: 11.85
R^2 Score: 1.00
```



Random Forest Classification :

Random Forest Classification is a go-to choice for many practitioners due to its ability to improve accuracy and handle complex datasets. It is particularly useful when interpretability of individual trees is not a primary concern, and the focus is on achieving robust and accurate predictions.

```
from sklearn.ensemble import RandomForestClassifier  # Import RandomForestClassifier

# Data preprocessing... (Preprocessing steps here):

# Define bins for classification
bins = [0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, float('inf')]
# Create labels for each bin
labels = ['0-100', '100-200', '200-300', '300-400', '400-500', '500-600', '600-700', '700-800', '800-900', '900-1000', '1000-1100', '1100-inf']

# Add a new column 'RUL_bin' with the bin labels
data['RUL_bin'] = pd.cut(data['RUL'], bins=bins, labels=labels)

# Split the data into features (X) and target variable (y)
y = data['RUL_bin']
X = data.drop(['RUL', 'RUL_bin'], axis=1)

# Split the 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=900)

# Check for NaN values in the target variable (y_train and y_test)
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())
```

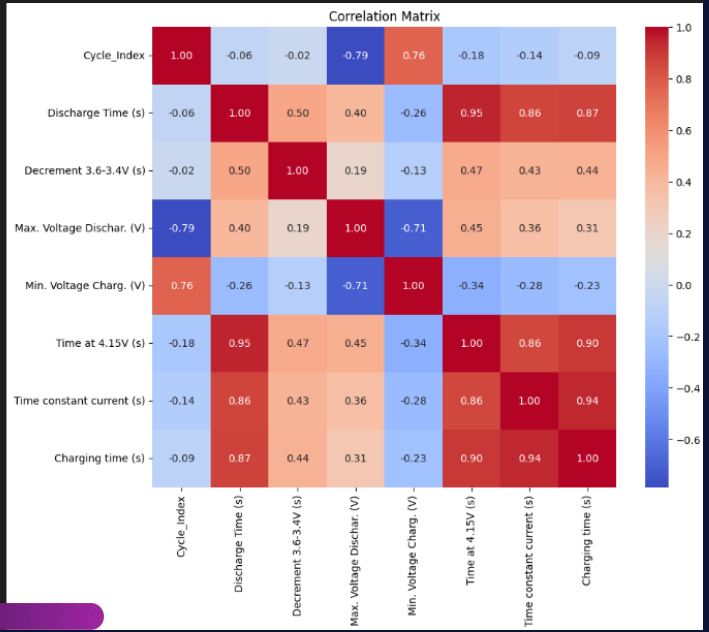
Random Forest Classification :

```python
# Drop NaN values in the target variable
X_train = X_train[~y_train.isnull()]
y_train = y_train.dropna()
X_test = X_test[~y_test.isnull()]
y_test = y_test.dropna()
print("NaN values have been dropped successfully \n")

# Check if there are NaN values in the features
print("NaN values in y_train:", y_train.isnull().sum())
print("NaN values in y_test:", y_test.isnull().sum())
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

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

# Initialize Random Forest Classifier and fit the model
random_forest_classifier = RandomForestClassifier(random_state=42)
random_forest_classifier.fit(X_train_scaled, y_train)

# draw a correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X_train_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

# Apply cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=42)
cv_scores = cross_val_score(random_forest_classifier, X_train_scaled, y_train, cv=cv, scoring='accuracy')

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy: {:.2%}".format(cv_scores.mean()))
```

```python
# Make predictions
y_pred = random_forest_classifier.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2%}")

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Scatter plot for actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, alpha=0.5, c="red", edgecolors='k')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual RUL Bin')
plt.ylabel('Predicted RUL Bin')
plt.grid(True)
plt.show()

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Plot confusion matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

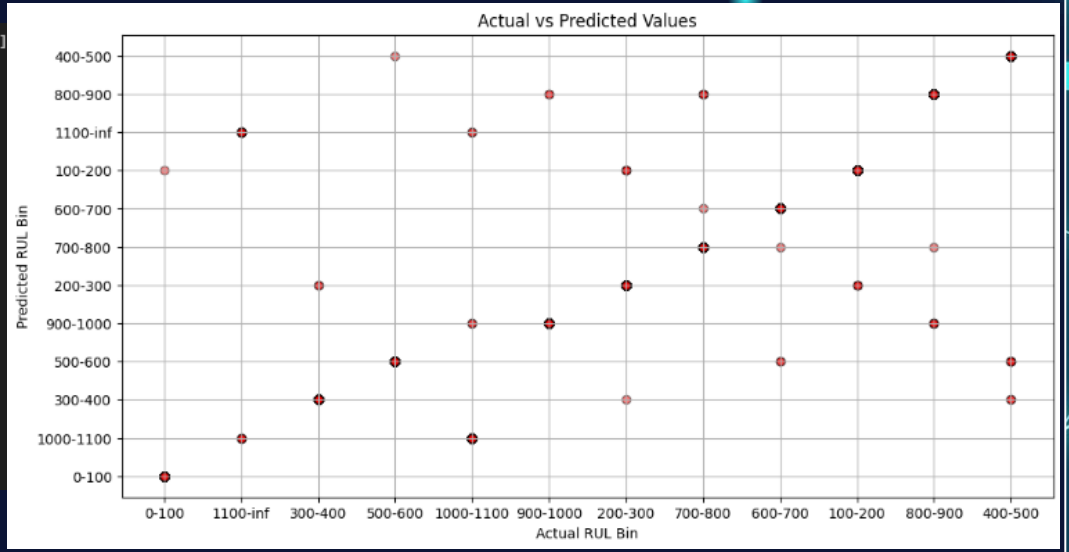
Random Forest Classification results:

```
NaN values in y_train: 11
NaN values in y_test: 3
NaN values in X_train: 0
NaN values in X_test: 0
NaN values have been dropped successfully

NaN values in y_train: 0
NaN values in y_test: 0
NaN values in X_train: 0
NaN values in X_test: 0
```

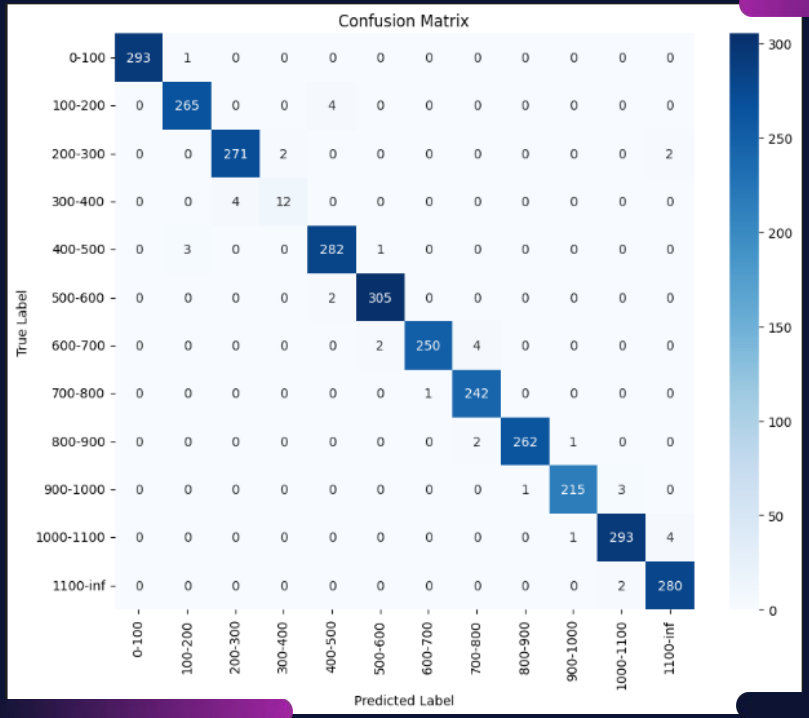


Random Forest Classification results:

```
Cross-Validation Scores: [0.98255814 0.98920266 0.9833887  0.9833887  0.98754153]
Mean Accuracy: 98.52%
Accuracy: 98.67%
Classification Report:
              precision    recall  f1-score   support

       0-100       1.00      1.00      1.00       294
     100-200       0.99      0.99      0.99       269
    1000-1100      0.99      0.99      0.99       275
    1100-inf       0.86      0.75      0.80        16
     200-300       0.98      0.99      0.98       286
     300-400       0.99      0.99      0.99       307
     400-500       1.00      0.98      0.99       256
     500-600       0.98      1.00      0.99       243
     600-700       1.00      0.99      0.99       265
     700-800       0.99      0.98      0.99       219
     800-900       0.98      0.98      0.98       298
     900-1000      0.98      0.99      0.99       282

    accuracy                           0.99      3010
   macro avg       0.98      0.97      0.97      3010
weighted avg       0.99      0.99      0.99      3010
```



Random Forest Classification results:



Classificcation Accuracy Comparison :

KNN
97.51%

01

02

Logistic Regression
91.13%

Decision Tree
98.04%

03

04

Support Vector
91.89%

Random Forrest
98.67%

05

Thanks!

Do you have any questions?

Samer.wael.2003@gmail.com
+201554563448
Samer Wael