Name: Ayush Maurya Class: D15C/ Batch B Roll No: 34

Experiment 5

Aim: Perform Regression Analysis using **Scipy** and **Scikit-learn**.

Problem Statement:

1. **Perform Logistic Regression** to determine the relationship between variables.
2. **Apply a Regression Model** technique to predict the data based on the given dataset.

Dataset Description:

The dataset contains more than **1 lakh instances**, fulfilling the requirement for **Big Data analysis**. The following are the key features used in the regression models:

- **Cycle_Index**: Index of the cycle during battery charging/discharging.
- **Discharge Time (s)**: Time taken for battery discharge.
- **Decrement 3.6-3.4V (s)**: Time decrements between specific voltage levels.
- **Max. Voltage Discharge. (V)**: Maximum voltage during discharge.
- **Min. Voltage Charge. (V)**: Minimum voltage during charge.
- **Time at 4.15V (s)**: Time spent at a specific voltage level.
- **Time constant current (s)**: Duration of constant current phase.
- **Charging time (s)**: Total time taken for charging.
- **RUL (Remaining Useful Life)**: Predicted remaining life of the battery.

The dataset is preprocessed to remove missing values and scale numerical features before regression analysis.

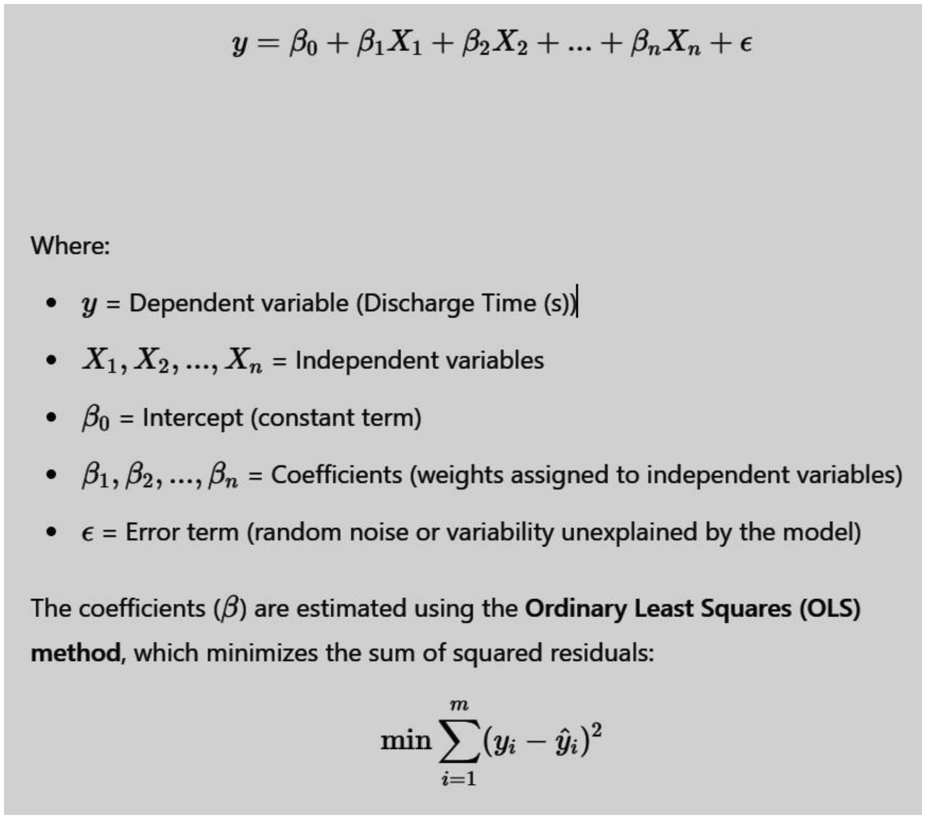
Theory & Mathematical Background

1. Linear Regression:

Linear Regression is a fundamental supervised learning algorithm that models the relationship between a dependent variable (**y**) and one or more independent variables (**X**) by fitting a linear equation. It is widely used for predictive modeling where the goal is to establish a linear relationship between input and output variables.

Mathematical Representation:

The equation for multiple linear regression is:



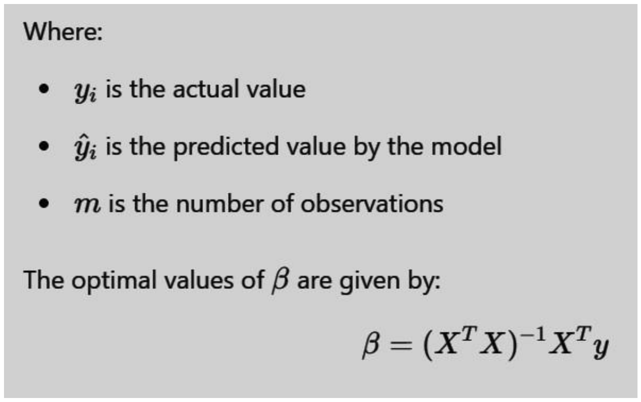
$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon$$

Where:

- y = Dependent variable (Discharge Time (s))
- $X_1, X_2, \ldots, X_n$ = Independent variables
- β_0 = Intercept (constant term)
- $\beta_1, \beta_2, \ldots, \beta_n$ = Coefficients (weights assigned to independent variables)
- ϵ = Error term (random noise or variability unexplained by the model)

The coefficients (β) are estimated using the **Ordinary Least Squares (OLS) method**, which minimizes the sum of squared residuals:

$$\min \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$



Where:

- y_i is the actual value
- $\hat{y}_i$ is the predicted value by the model
- m is the number of observations

The optimal values of β are given by:

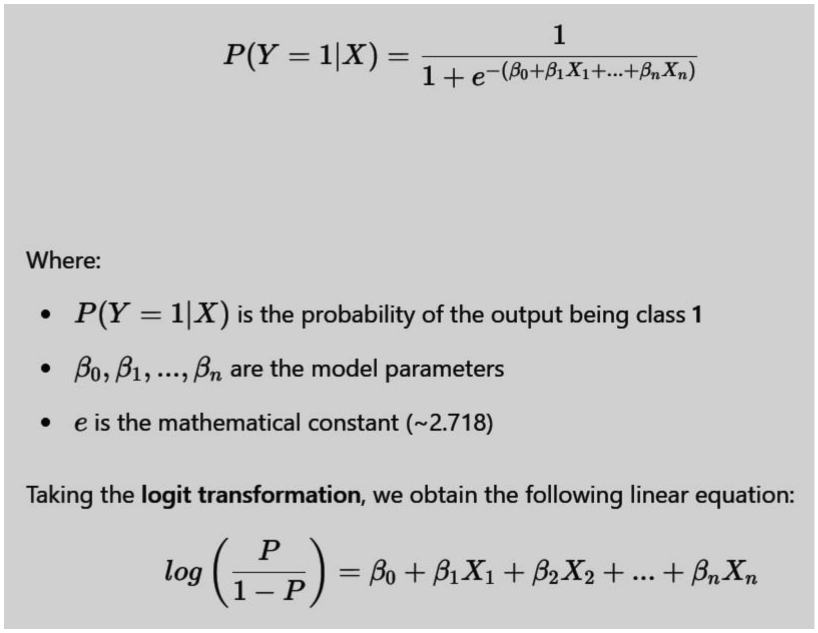
$$\beta = (X^T X)^{-1} X^T y$$

2. Logistic Regression:

Logistic Regression is used when the dependent variable is **binary** (classification problem). Instead of predicting a continuous value, it predicts the probability that a given input belongs to a specific category (0 or 1).

Mathematical Representation:

Instead of a linear function, logistic regression uses the **sigmoid function** (logistic function) to map the predictions between **0 and 1**:



$$P(Y=1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+...+\beta_n X_n)}}$$

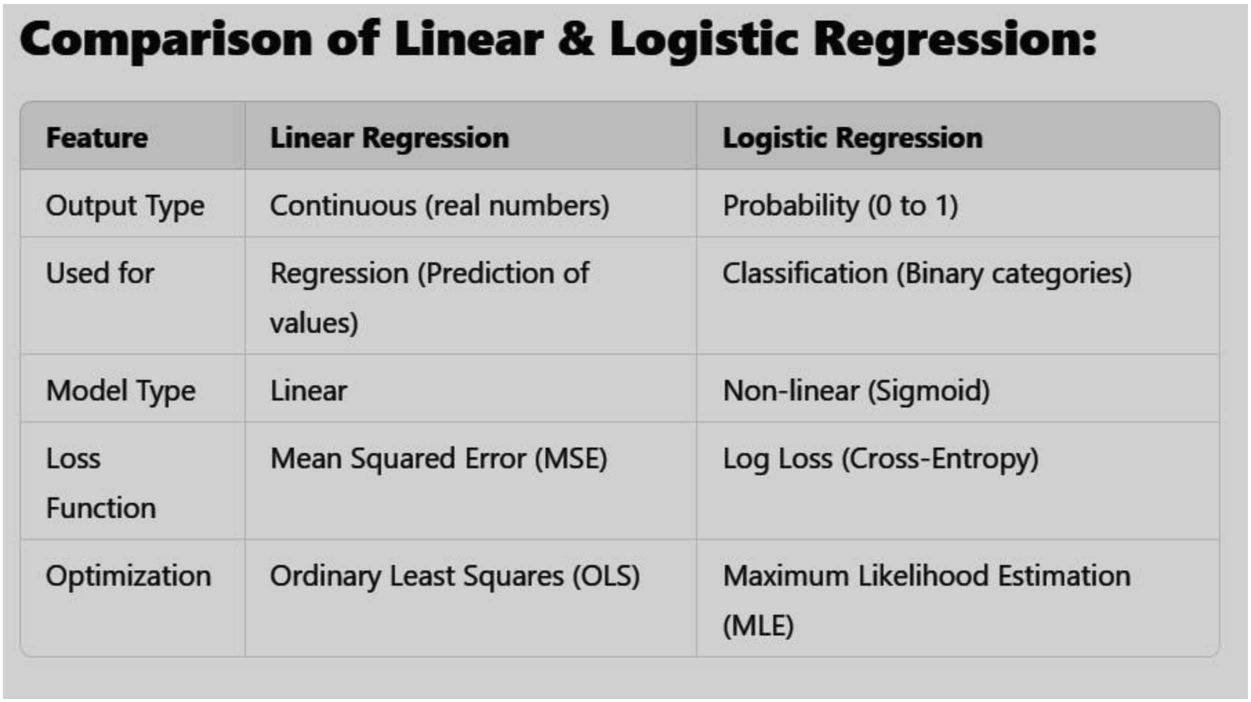
Where:

- $P(Y=1|X)$ is the probability of the output being class **1**
- $\beta_0, \beta_1, ..., \beta_n$ are the model parameters
- e is the mathematical constant (~2.718)

Taking the **logit transformation**, we obtain the following linear equation:

$$log\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$$

The model is trained using **Maximum Likelihood Estimation (MLE)**, which finds the parameters that maximize the likelihood of the observed data.

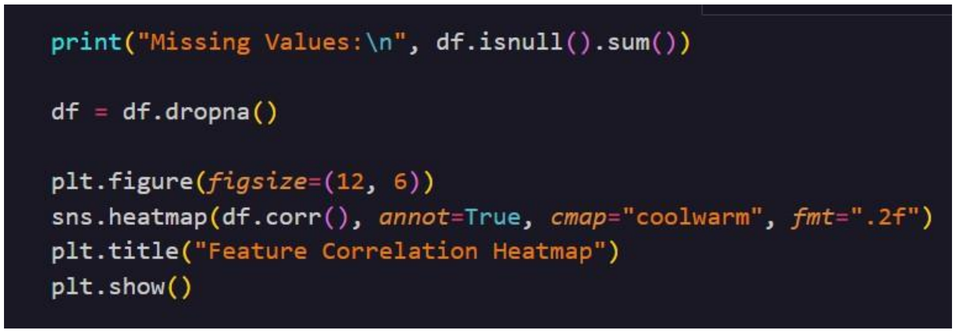


Comparison of Linear & Logistic Regression:

Feature	**Linear Regression**	**Logistic Regression**
Output Type	Continuous (real numbers)	Probability (0 to 1)
Used for	Regression (Prediction of values)	Classification (Binary categories)
Model Type	Linear	Non-linear (Sigmoid)
Loss Function	Mean Squared Error (MSE)	Log Loss (Cross-Entropy)
Optimization	Ordinary Least Squares (OLS)	Maximum Likelihood Estimation (MLE)

Output:

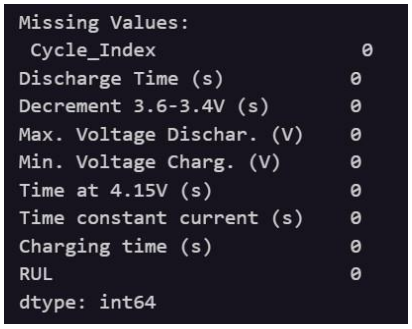
1]The heatmap shows correlations between features, highlighting strong positive relationships (e.g., Charging Time & Discharge Time: 0.94) and negative correlations (e.g., Cycle Index & RUL: -1.00). It confirms that battery aging affects discharge characteristics and RUL.



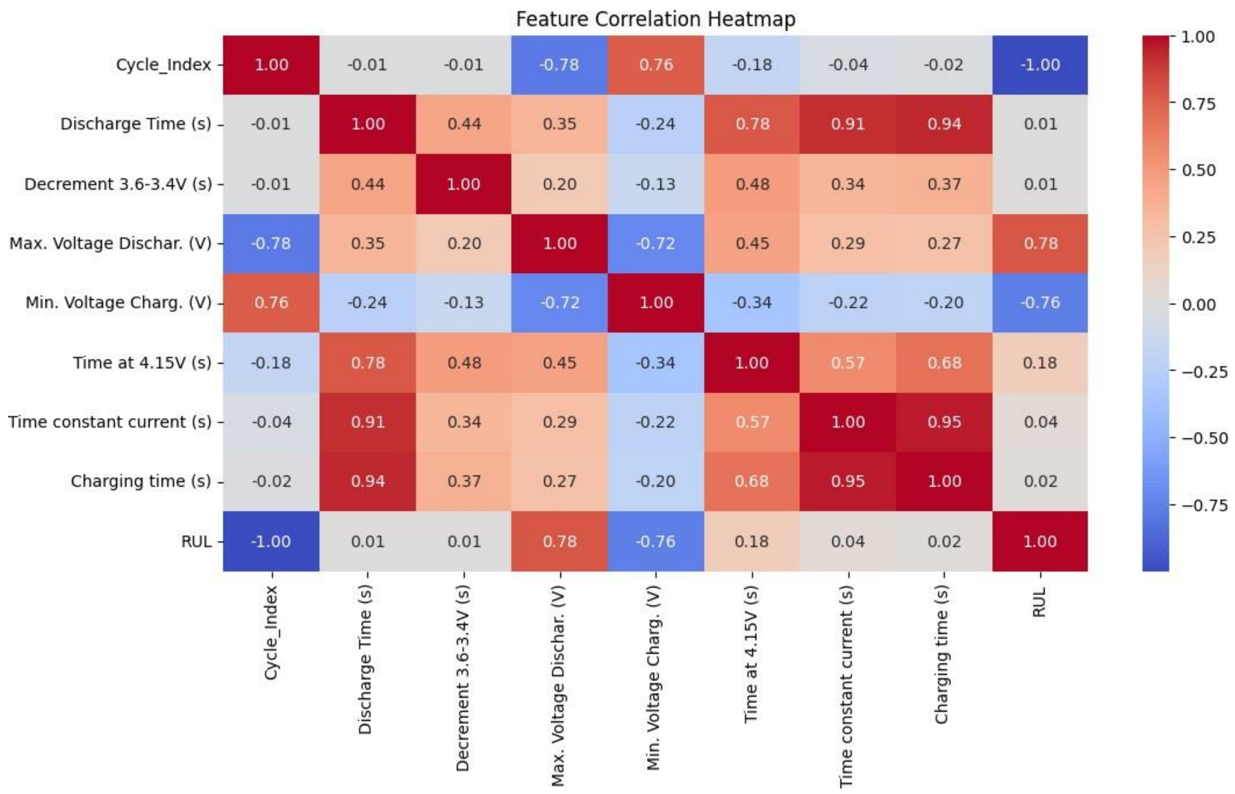
```python
print("Missing Values:\n", df.isnull().sum())

df = df.dropna()

plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```

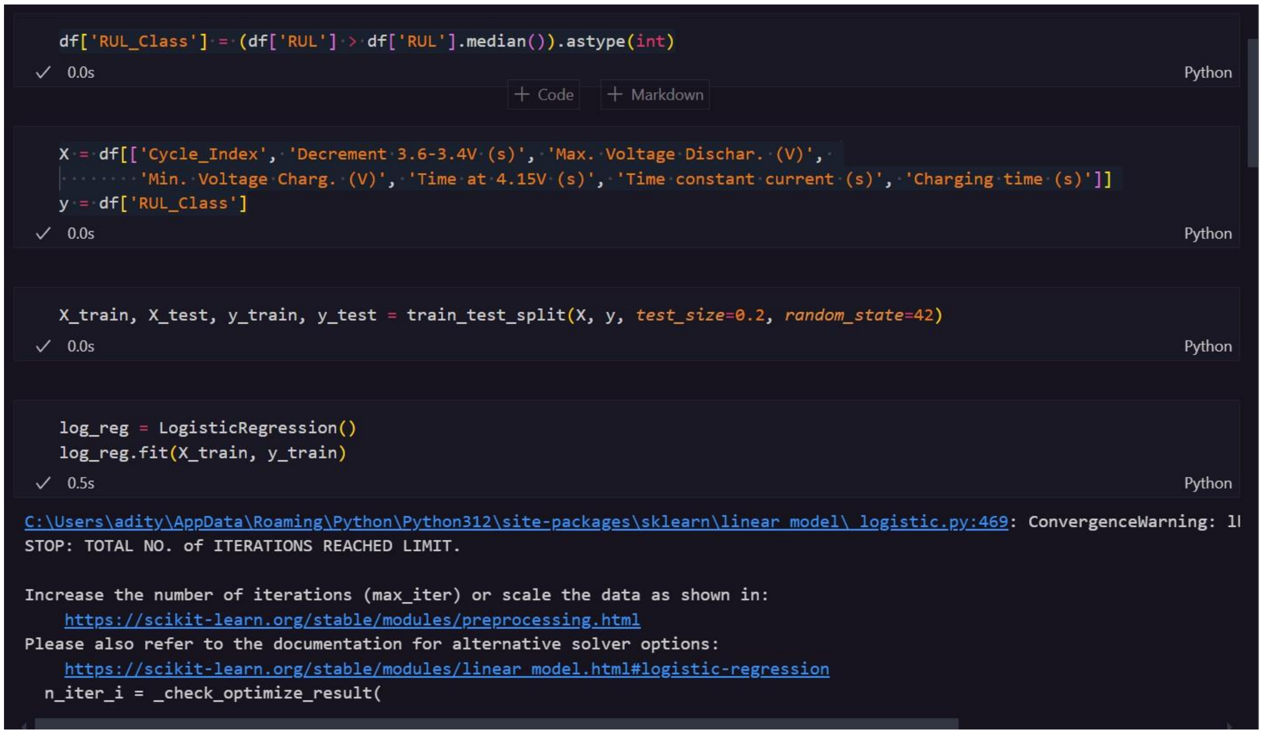


```
Missing Values:
 Cycle_Index                  0
Discharge Time (s)           0
Decrement 3.6-3.4V (s)       0
Max. Voltage Dischar. (V)    0
Min. Voltage Charg. (V)      0
Time at 4.15V (s)            0
Time constant current (s)    0
Charging time (s)            0
RUL                          0
dtype: int64
```



	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
Cycle_Index	1.00	-0.01	-0.01	-0.78	0.76	-0.18	-0.04	-0.02	-1.00
Discharge Time (s)	-0.01	1.00	0.44	0.35	-0.24	0.78	0.91	0.94	0.01
Decrement 3.6-3.4V (s)	-0.01	0.44	1.00	0.20	-0.13	0.48	0.34	0.37	0.01
Max. Voltage Dischar. (V)	-0.78	0.35	0.20	1.00	-0.72	0.45	0.29	0.27	0.78
Min. Voltage Charg. (V)	0.76	-0.24	-0.13	-0.72	1.00	-0.34	-0.22	-0.20	-0.76
Time at 4.15V (s)	-0.18	0.78	0.48	0.45	-0.34	1.00	0.57	0.68	0.18
Time constant current (s)	-0.04	0.91	0.34	0.29	-0.22	0.57	1.00	0.95	0.04
Charging time (s)	-0.02	0.94	0.37	0.27	-0.20	0.68	0.95	1.00	0.02
RUL	-1.00	0.01	0.01	0.78	-0.76	0.18	0.04	0.02	1.00

2]This script performs a binary classification task on battery life data using logistic regression. First, it creates a new target variable, `RUL_Class`, by comparing the `RUL` values to their median, converting the problem into a classification task where values above the median are labeled as `1` and below as `0`. Next, it selects relevant sensor features (`Cycle_Index`, voltage measurements, and charging times) as input (`X`) and assigns the newly created `RUL_Class` as the target variable (`y`). The dataset is then split into training (80%) and testing (20%) sets using `train_test_split`, ensuring reproducibility with `random_state=42`. A logistic regression model is initialized and trained on the training data. However, a `ConvergenceWarning` occurs, indicating that the model did not converge within the default number of iterations. To resolve this, one could increase `max_iter`, scale the data using standardization techniques, or try alternative solvers like `saga` or `lbfgs`.



```
df['RUL_Class'] = (df['RUL'] > df['RUL'].median()).astype(int)
✓ 0.0s                                                             Python
```

```
X = df[['Cycle_Index', '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)']]
y = df['RUL_Class']
✓ 0.0s                                                             Python
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
✓ 0.0s                                                             Python
```

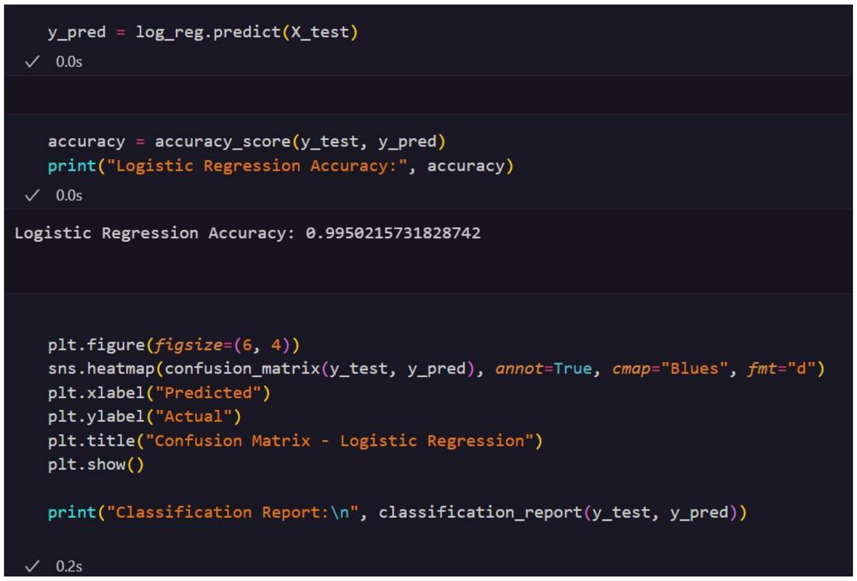
```
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
✓ 0.5s                                                             Python
```

```
C:\Users\adity\AppData\Roaming\Python\Python312\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lb
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

3]This script evaluates the performance of the trained logistic regression model. It first predicts the target labels (`y_pred`) using the test dataset (`X_test`). Then, it calculates and prints the model's accuracy using `accuracy_score`, which shows an extremely high accuracy of ~99.5%. To visualize performance, a confusion matrix is plotted using Seaborn's heatmap, displaying the actual vs. predicted values. The matrix helps in identifying misclassifications. Finally, a classification report is printed, which provides precision, recall, and F1-score for each class, giving deeper insights into model performance. The high accuracy suggests possible overfitting, especially if the dataset is imbalanced.

This confusion matrix represents the performance of the logistic regression model. The matrix shows that out of 1536 actual class `0` samples, 1530 were correctly classified, and 6 were misclassified as class `1`. Similarly, out of 1477 actual class `1` samples, 1468 were correctly classified, and 9 were misclassified as class `0`. The model demonstrates high accuracy with minimal misclassifications, indicating strong performance. However, the extremely low error rate suggests the dataset might be highly separable or imbalanced, possibly leading to overfitting.

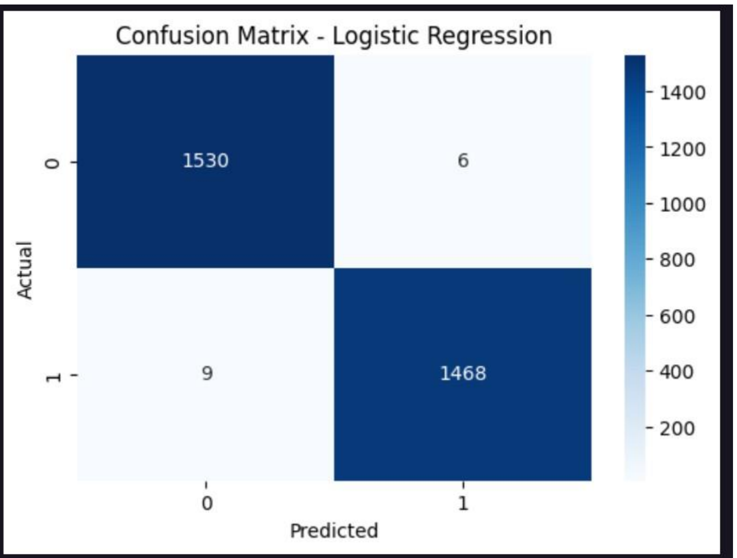


```
y_pred = log_reg.predict(X_test)
✓ 0.0s

accuracy = accuracy_score(y_test, y_pred)
print("Logistic Regression Accuracy:", accuracy)
✓ 0.0s
Logistic Regression Accuracy: 0.9950215731828742

plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Logistic Regression")
plt.show()

print("Classification Report:\n", classification_report(y_test, y_pred))
✓ 0.2s
```

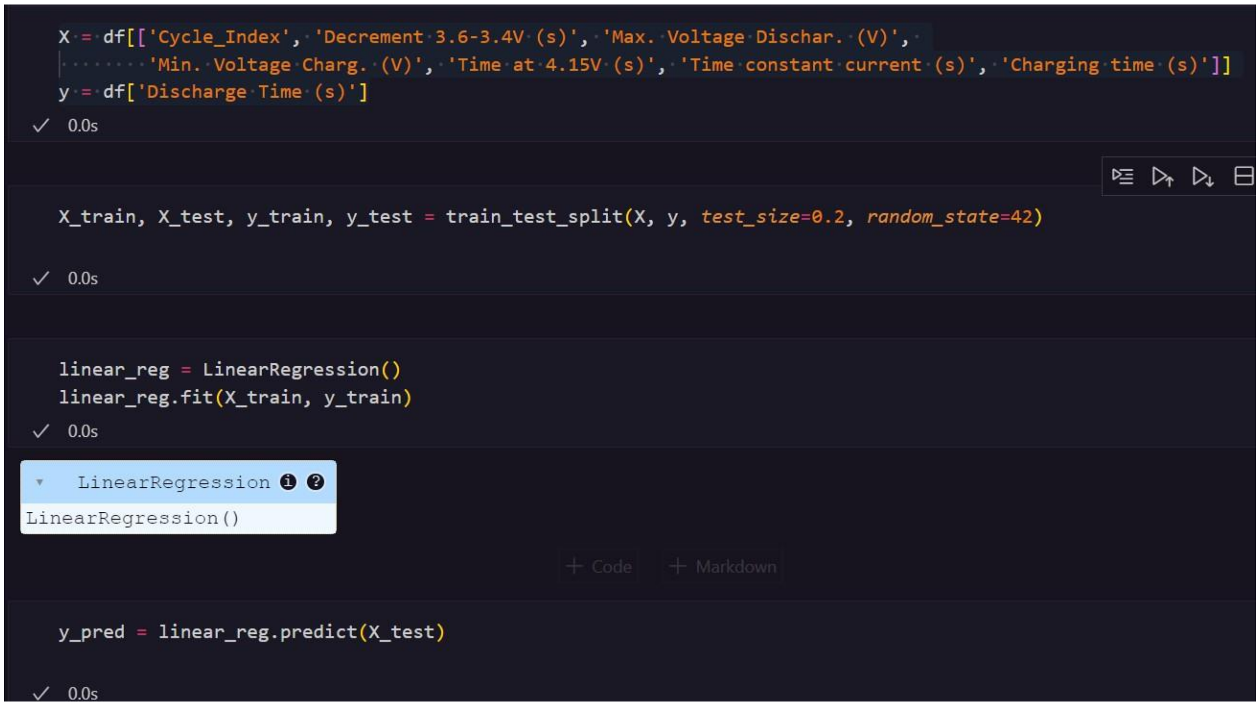


```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      1536
           1       1.00      0.99      0.99      1477

    accuracy                           1.00      3013
   macro avg       1.00      1.00      1.00      3013
weighted avg       1.00      1.00      1.00      3013
```

4]For linear regression, to predict battery discharge time using various features. You've split the dataset into training and testing sets, trained a `LinearRegression` model, and made predictions. The next step is to evaluate the model using metrics like Mean Squared Error (MSE) and R² score to determine how well it performs.



```
X = df[['Cycle_Index', '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)']]
y = df['Discharge Time (s)']
```

✓ 0.0s

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

✓ 0.0s

```
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
```

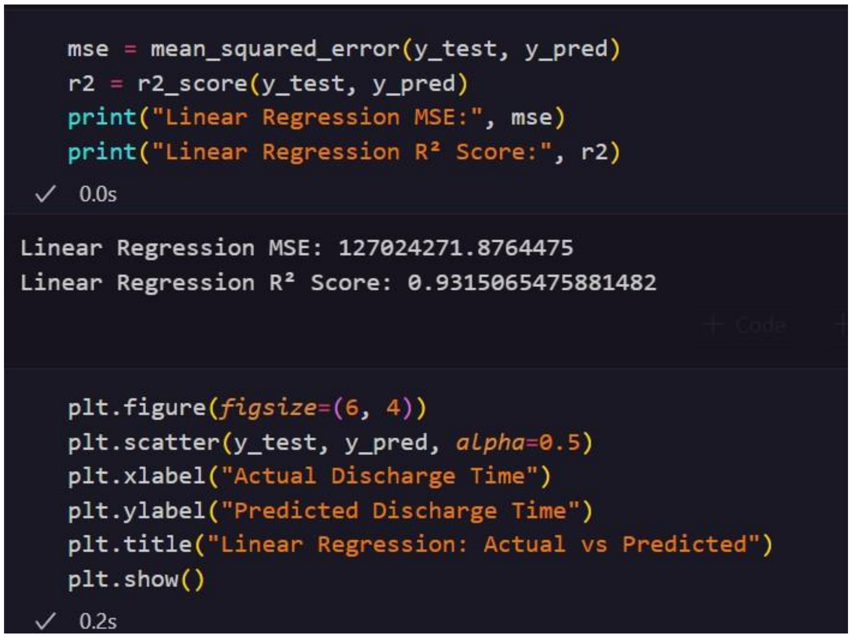
✓ 0.0s

```
LinearRegression
LinearRegression()
```

```
y_pred = linear_reg.predict(X_test)
```

✓ 0.0s

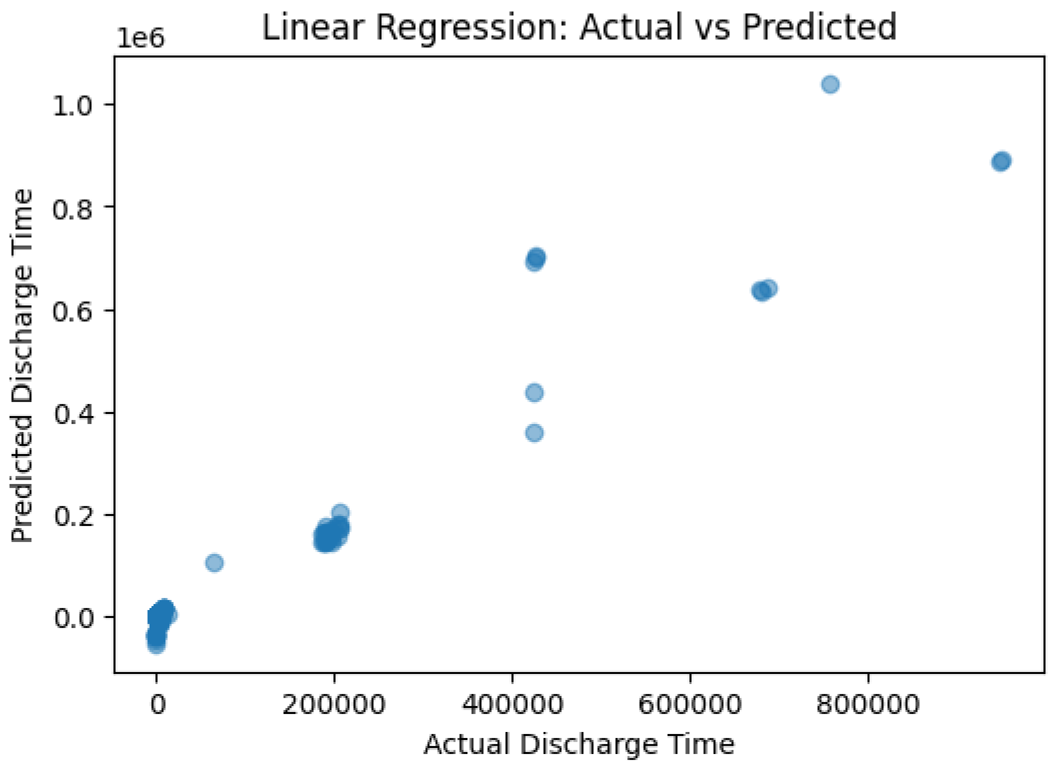
5]Your linear regression model seems to be performing well, with an R² score of approximately 0.93, indicating that the model explains 93% of the variance in the discharge time. However, the Mean Squared Error (MSE) value is quite large, which suggests that while the model fits well overall, there might be some high-magnitude errors in individual predictions. Your scatter plot of actual vs. predicted discharge time shows a generally upward trend, which indicates that your linear regression model is capturing the overall pattern of the data. However, there are some potential issues to consider. The spread of points suggests that for higher actual discharge times, the model's predictions might be slightly off, possibly underestimating or overestimating in some cases. This could be due to outliers, heteroscedasticity, or the linear model's inability to capture complex relationships in the data. We might consider feature scaling, polynomial regression, or trying a different model like Random Forest or Gradient Boosting to improve performance.



```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Linear Regression MSE:", mse)
print("Linear Regression R² Score:", r2)
```
✓ 0.0s

```
Linear Regression MSE: 127024271.8764475
Linear Regression R² Score: 0.9315065475881482
```

```
plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual Discharge Time")
plt.ylabel("Predicted Discharge Time")
plt.title("Linear Regression: Actual vs Predicted")
plt.show()
```
✓ 0.2s



Conclusion:

Linear regression is useful for predicting continuous values, as seen in the discharge time prediction. It showed a strong R² score (~0.93), indicating a good fit, but potential outliers and non-linearity could impact accuracy. Logistic regression, used for classification, helps predict categorical outcomes but isn't suitable for continuous data. Comparing both, I learned that linear regression works best for numerical trends, while logistic regression is for classification tasks. Innovation lies in optimizing models—feature engineering, non-linear models, or ensemble methods improve accuracy. o improve accuracy, we can handle outliers, apply feature engineering, or choose advanced models like Random Forest, which can capture complex patterns better and reduce overfitting for improved predictions.

Understanding these differences enhances real-world predictive capabilities and decision-making in machine learning applications.