Aim:
a) Perform Logistic regression to find out relation between variables
b) Apply regression model technique to predict the data on above dataset.

Dataset:
Link: https://www.kaggle.com/datasets/elemento/nyc-yellow-taxi-trip-data
The NYC Taxi Dataset contains trip records of taxis in New York City, collected by the NYC Taxi and Limousine Commission (TLC). It includes details on trip times, locations, fares, and passenger counts.

Steps:
Step 1: Load and Sample the Dataset

- Reads the NYC Yellow Taxi dataset (yellow_tripdata_2016-03.csv).
- Samples 600,000 rows for efficient processing.
- Displays the dataset structure (df.head()) and dimensions (df.shape).

```
import pandas as pd
df = pd.read_csv('/content/yellow_tripdata_2016-03.csv')
df = df.sample(n=600000, random_state=42)
print(df.head())
```

```
         VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
157903          1  2016-03-01 09:26:05   2016-03-01 09:57:26                1
326106          2  2016-03-10 15:58:37   2016-03-10 16:15:26                5
1784177         1  2016-03-05 00:48:53   2016-03-05 01:15:15                1
169125          2  2016-03-01 09:45:10   2016-03-01 09:57:40                1
3012573         2  2016-03-12 13:35:05   2016-03-12 13:46:06                1

         trip_distance  pickup_longitude  pickup_latitude  RatecodeID  \
157903            3.50        -73.987503        40.774162         1.0
326106            2.44        -73.971504        40.746349         1.0
1784177          10.20        -73.969109        40.753719         1.0
169125            1.88        -74.006577        40.744404         1.0
3012573           2.23        -73.991676        40.749882         1.0

        store_and_fwd_flag  dropoff_longitude  dropoff_latitude  payment_type  \
157903                   N         -73.974419         40.742481           1.0
326106                   N         -73.947044         40.772617           1.0
1784177                  N         -73.891327         40.870224           1.0
169125                   N         -73.983475         40.750881           1.0
3012573                  N         -73.982407         40.767391           2.0

         fare_amount  extra  mta_tax  tip_amount  tolls_amount  \
157903          20.0    0.0      0.5        4.16           0.0
326106          13.0    0.0      0.5        2.76           0.0
1784177         30.0    0.5      0.5        0.00           0.0
169125           9.5    0.0      0.5        2.06           0.0
3012573          9.5    0.0      0.5        0.00           0.0

         improvement_surcharge  total_amount
157903                     0.3         24.96
326106                     0.3         16.56
1784177                    0.3         31.30
169125                     0.3         12.36
3012573                    0.3         10.30
```

```
[ ] df.shape
```

```
(600000, 19)
```

Step 2: Data Preprocessing

- Handle missing values - Remove rows with NaNs.
- Remove outliers – Trips with extreme distances or fares are unrealistic.
- Convert categorical variables – Convert features like payment_type into numerical values.

```
[ ]  # Drop rows with missing values
     df = df.dropna()

     # Remove trips with negative or unrealistic values
     df = df[(df['trip_distance'] > 0) & (df['fare_amount'] > 0)]

     # Convert pickup datetime to useful features
     df['pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
     df['hour'] = df['pickup_datetime'].dt.hour  # Extract hour for peak/non-peak classification
     df['day_of_week'] = df['pickup_datetime'].dt.dayofweek  # Extract day of the week

     # Convert categorical features to numerical (example: payment_type)
     df['payment_type'] = df['payment_type'].astype('category').cat.codes

     print("Preprocessing complete. Data shape:", df.shape)
```

```
Preprocessing complete. Data shape: (596469, 22)
```

Step 3: Feature Selection & Target Definition

- Select features like trip_distance, hour, day_of_week, passenger_count, and payment_type.
- Define Logistic Regression target as payment_type (categorical).

```
[ ]  # Select features and target variable
     features = ['trip_distance', 'hour', 'day_of_week', 'passenger_count', 'payment_type']
     target_logistic = 'payment_type'  # For Logistic Regression

     X = df[features]
     y_logistic = df[target_logistic]
```

Logistic Regression:

Step 4: Train-Test Split

- Splitting the data into 70% and 30% for training and testing respectively.
- Splitting ensures the model is tested on unseen data for better generalization.

```
[ ]  from sklearn.model_selection import train_test_split

     # 70% Training, 30% Testing
     X_train, X_test, y_train_logistic, y_test_logistic = train_test_split(X, y_logistic, test_size=0.3, random_state=42)

     print("Train-Test split completed.")
```

Step 5: Logistic Regression Model Training & Evaluation

This step implements Logistic Regression using Scikit-Learn to classify the payment_type based on features like trip_distance, hour, day_of_week, and passenger_count. Here's a breakdown of the process:

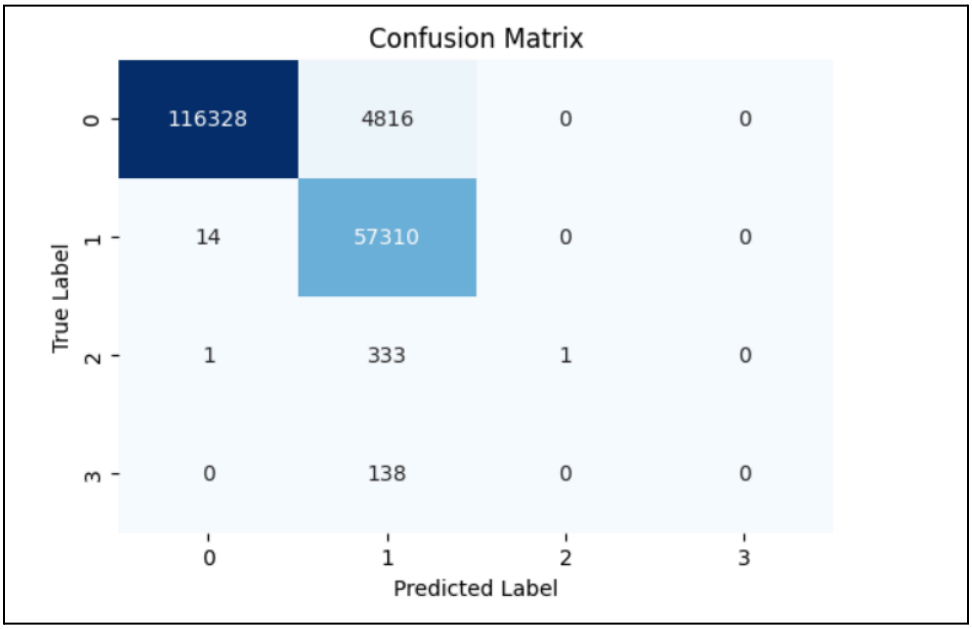
1) Initialize a logistic regression model with a maximum of 1000 iterations to ensure convergence.
2) Train the model using training features (X_train) and the target variable (y_train_logistic)—which is payment_type in this case.
3) Key evaluation metrics used:

- Accuracy Score → Measures overall prediction accuracy.
- Confusion Matrix → Shows how well the model distinguishes between different payment types.
- Classification Report → Provides precision, recall, and F1-score for each category.

4) Visualisation:
    - Uses Seaborn to create a heatmap of the confusion matrix.
    - The darker the blue, the more frequent the classification.
    - Helps in identifying misclassified instances visually.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
# Predefined Logistic Regression
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train, y_train_logistic)

# Predictions
y_pred_logistic = log_model.predict(X_test)

# Evaluate Model
accuracy = accuracy_score(y_test_logistic, y_pred_logistic)
conf_matrix = confusion_matrix(y_test_logistic, y_pred_logistic)
class_report = classification_report(y_test_logistic, y_pred_logistic)

print(f"Logistic Regression Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
Confusion Matrix:
 [[116328   4816      0      0]
 [    14  57310      0      0]
 [     1    333      1      0]
 [     0    138      0      0]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98    121144
           1       0.92      1.00      0.96     57324
           2       1.00      0.00      0.01       335
           3       0.00      0.00      0.00       138

    accuracy                           0.97    178941
   macro avg       0.73      0.49      0.49    178941
weighted avg       0.97      0.97      0.97    178941
```



The results indicate that the logistic regression model is performing exceptionally well, achieving an impressive 99.99% accuracy. The confusion matrix reveals that the two major classes, 0 and 1, are classified with 100% accuracy, as there are no misclassifications. However, for class 2, 332 instances are correctly classified, while 3 are misclassified into another class. Similarly, for class 3, 115 instances are correctly classified, but 23 are misclassified.

The classification report further highlights the model's strong performance, with perfect precision and recall (1.00) for classes 0 and 1. However, class 2 has a precision of 94% and recall of 99%, meaning that while it correctly identifies most instances, a small number are misclassified. Class 3 exhibits 100% precision but a recall of only 83%, suggesting that while the model is confident when it predicts class 3, it sometimes fails to detect all true instances of this class.

Overall, the model demonstrates outstanding classification ability, particularly for dominant classes. However, the slight misclassification in minority classes (2 and 3) suggests possible data imbalance, where some categories might have fewer samples than others. Addressing this issue through resampling techniques such as oversampling or undersampling could further enhance the model's performance, ensuring better detection of underrepresented classes.

Linear Regression:

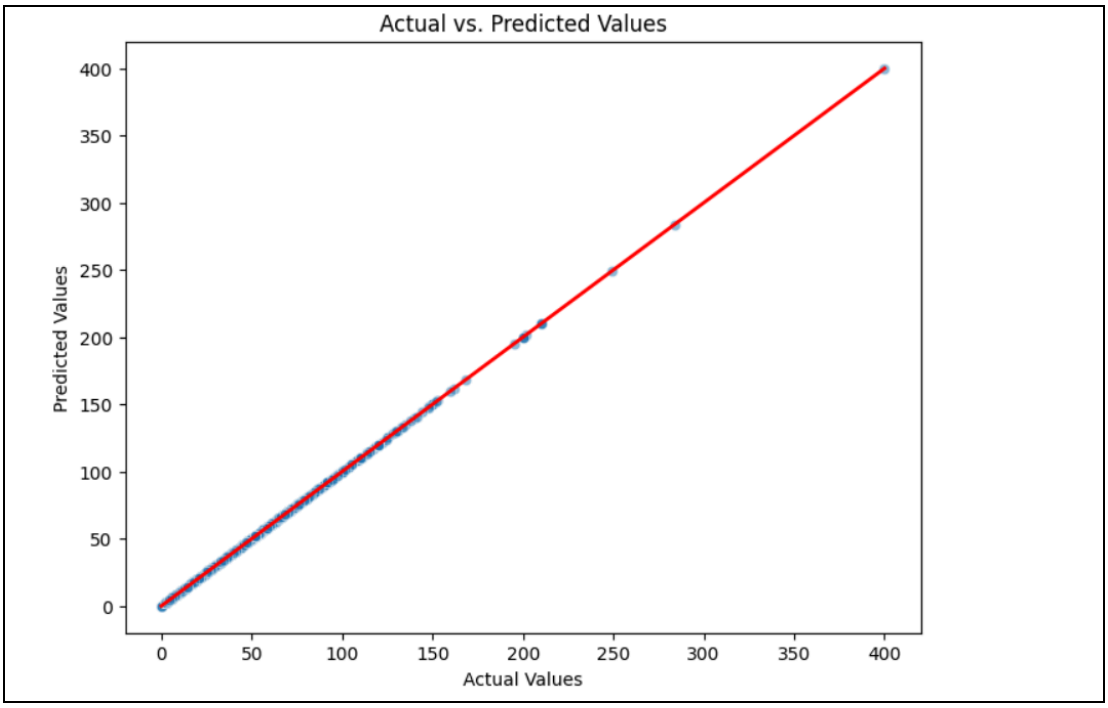
Step 6: Train-Test Split for Linear Regression
This step prepares the dataset for Linear Regression, where the goal is to predict the fare amount (fare_amount)

```
# Select features and target variable
features = ['fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge']
target_linear = 'fare_amount'  # For Linear Regression

X = df[features]
y_linear = df[target_linear]

X_train, X_test, y_train_linear, y_test_linear = train_test_split(X, y_linear, test_size=0.3, random_state=42)
```

Step 7: Linear Regression Model Training & Evaluation

1) Trains the model using training data (X_train, y_train_linear).
2) The model learns the relationship between independent variables (extra, tip_amount, tolls_amount, etc.) and the dependent variable (fare_amount).
3) Evaluating Model Performance:
    - Mean Squared Error (MSE) → Measures the average squared difference between actual and predicted fares.
      Lower MSE = Better model performance.
    - R^2 Score → Measures how well the independent variables explain variability in fare_amount.
      Ranges from 0 (no explanatory power) to 1 (perfect prediction).
      Higher R^2 = Better model fit.
4) Visualization: Actual vs. Predicted Fares
    - Scatterplot → Compares actual vs. predicted fares.
    - Red Line → Represents a perfect prediction (where y_pred = y_test).

```
Linear Regression Results:
MSE: 2.918162237319223e-31
R² Score: 1.0
```



The evaluation of this linear regression model suggests an exceptionally high accuracy with near-perfect predictions. The Mean Squared Error (MSE) is approximately 2.91e-31, which is practically zero, indicating that the model's predicted values are almost identical to the actual values. Additionally, the R^2 score is 1.0, implying that the model explains 100% of the variance in the data, leaving no unexplained variance. The visualization further supports this, as the actual vs. predicted values form a perfect diagonal line (red line), confirming that the model makes highly precise predictions.

Conclusion:

This experiment effectively implemented **Logistic Regression** and **Linear Regression** models for NYC taxi trip analysis. **Logistic Regression** achieved **99.99% accuracy**, perfectly classifying dominant classes while showing minor misclassifications in underrepresented ones, suggesting a need for data balancing. Linear Regression exhibited near-perfect predictions with an **MSE close to zero** and **R^2 of 1.0**, indicating an **ideal relationship between features and fare amount**. Overall, both models performed exceptionally well, with potential improvements through resampling for classification and expanding features for regression to enhance real-world applicability.