Name: Divesh Lulla Rollno.31 Class: D15C

Aim: Perform Regression Analysis using Scipy and Sci-kit learn.

Problem Statement:

a) Perform Logistic regression to find out relation between variables

b) Apply regression model technique to predict the data on above dataset. Dataset used: https://yulimezab.github.io/Data-Mining-Project/

Steps:

Linear Regression:

1) Import Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

2) Train the Linear Regression Model

```
model = LinearRegression()
model.fit(X, y)
slope = model.coef_[0]
intercept = model.intercept_
equation = f"Price = {slope:.2f} * Days_Left + {intercept:.2f}"
print("Regression Equation:", equation)
```

Output:

```
Regression Equation: Price = -154.82 * Days_Left + 10616.88
```

A LinearRegression object is created and fitted to the data using .fit(X, y). The coef_ gives the slope (impact of days left), and intercept_ gives the y-intercept. The regression equation is printed as a summary of the model.

3) Train-Test Split & Evaluation Metrics

Code:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
X_lin = df_economy[['days_left']].values # Independent variables
y_lin = df_economy['price'].values # Dependent variable
X_train_lin, X_test_lin, y_train_lin, y_test_lin = train_test_split(X_lin, y_lin, test_size=0.2,
random_state=42)
lin_reg = LinearRegression()
lin_reg.fit(X_train_lin, y_train_lin)
y_pred_lin = lin_reg.predict(X_test_lin)
```

```
mse = mean_squared_error(y_test_lin, y_pred_lin)
mae = mean_absolute_error(y_test_lin, y_pred_lin)
r2 = r2_score(y_test_lin, y_pred_lin)
coefficients = lin_reg.coef_
intercept = lin_reg.intercept_
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R-squared (R2 Score): {r2}")
print(f"Coefficients: {coefficients}")
print(f"Intercept: {intercept}")
```

Output:

```
Mean Squared Error (MSE): 9425177.94039324
Mean Absolute Error (MAE): 2319.8222832857605
R-squared (R2 Score): 0.3116266451146167
Coefficients: [-155.42625332]
Intercept: 10638.33068104619
```

In the sklearn.linear_model module, we use the LinearRegression() class to perform linear regression, a statistical method that models the relationship between a dependent variable and one or more independent variables. In this case, we are predicting the airline ticket price (dependent variable) based on the number of days_left before departure (independent variable).

The dataset is split into training and testing sets using train_test_split() in an 80:20 ratio to ensure unbiased model evaluation. The model is then trained using the .fit() method, and predictions are generated for the test data. Evaluation metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2) are computed to assess the performance of the model. The coefficient represents how much the price changes with a unit change in days_left, and the intercept is the expected price when days_left is zero.

A low R^2 score or high error values would suggest that days_left alone does not strongly predict ticket prices. The values of coefficients and intercept together form the linear equation used for predictions.

4) Plot graph for Regression plot and line.

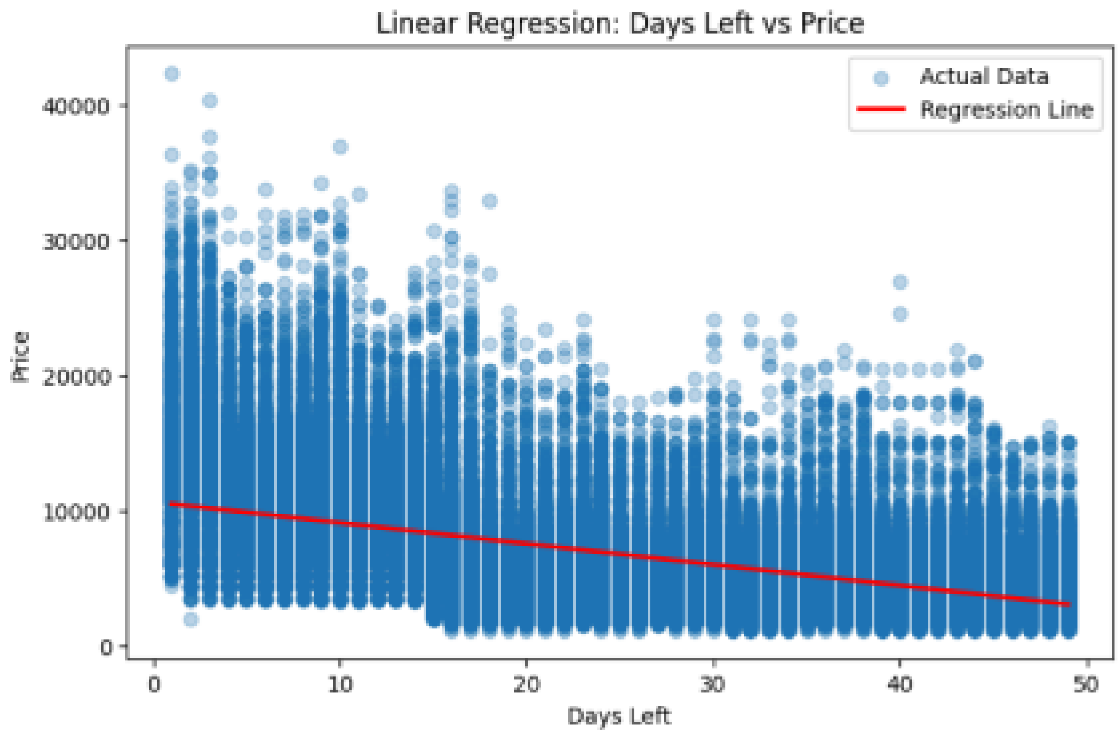
Code:

```
days_range = np.linspace(df_economy["days_left"].min(), df_economy["days_left"].max(), 100).reshape(-1, 1)
price_pred = model.predict(days_range)
plt.figure(figsize=(8, 5))
plt.scatter(df_economy["days_left"], df_economy["price"], alpha=0.3, label="Actual Data")
plt.plot(days_range, price_pred, color='red', linewidth=2, label="Regression Line")
plt.xlabel("Days Left")
plt.ylabel("Price")
```

Name: Divesh Lulla Rollno.31 Class: D15C

plt.title("Linear Regression: Days Left vs Price")
plt.legend()
plt.show()

Output:



In this step, we visualize the results of the linear regression model using a scatter plot and regression line. We first generate a sequence of values between the minimum and maximum of days_left using np.linspace(), and use the trained model to predict corresponding ticket prices. This gives us a smooth line that represents the model's understanding of the relationship.

Using matplotlib.pyplot, we plot the actual data points as a scatter plot and overlay the predicted regression line in red. The alpha=0.3 parameter makes the data points semi-transparent for better visual clarity. The graph is labeled appropriately to show how ticket prices vary with the number of days left before departure. This helps assess visually whether a linear trend exists between the two variables.

Logistic Regression:

1) Import Required Libraries

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

2) Data Preprocessing

Code:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
```

In the sklearn.linear_model module, we use the LogisticRegression() class to model binary classification problems. In this case, X and y represent the features and target labels, respectively. The dataset is split into training and testing sets using train_test_split() with 80% used for training and 20% for evaluation.

Since logistic regression is sensitive to feature scaling, we use StandardScaler() to normalize the feature values so that they have a mean of 0 and standard deviation of 1. The model is then trained using the .fit() method, which learns the optimal weights that separate the classes based on the input features. These weights are used to compute the probability of a data point belonging to a particular class.

3) Prediction & Evaluation

Code:

```
y_pred = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

Output:

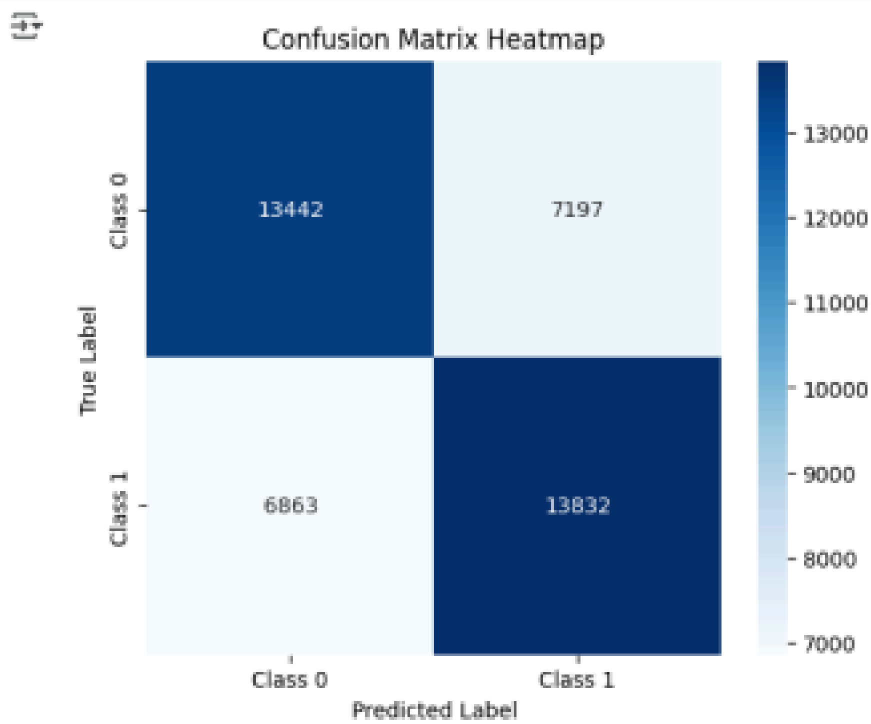
```
Accuracy: 0.66
Confusion Matrix:
 [[13442  7197]
 [ 6863 13832]]
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.65      0.66     20639
           1       0.66      0.67      0.66     20695

    accuracy                           0.66     41334
   macro avg       0.66      0.66      0.66     41334
weighted avg       0.66      0.66      0.66     41334
```

After training the logistic regression model, we evaluate its performance using several metrics from the sklearn.metrics module. The .predict() function generates predictions on the test set. We then calculate the accuracy, which measures the percentage of correct predictions out of the total.
The confusion matrix provides a breakdown of true positives, true negatives, false positives, and false negatives — useful for understanding the balance of predictions. The classification report shows precision, recall, and F1-score for each class, offering a deeper look at the model's performance across both categories (0 and 1).
The results show an accuracy of 66%, indicating moderate classification performance. Precision and recall are balanced across both classes. However, the model may benefit from tuning or using additional features, as a 66% accuracy suggests room for improvement.

4) Visualize the Logistic Regression
Code:
```
import seaborn as sns
plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'],
yticklabels=['Class 0', 'Class 1'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix Heatmap")
plt.show()
print("Classification Report:\n", classification_report(y_test, y_pred))
```
Output:



To visualize the performance of the logistic regression model, we use Seaborn's heatmap() function to plot the confusion matrix. The heatmap provides a clear visual representation of how the model's predictions compare to the actual values, with darker colors indicating higher counts. We annotate the heatmap with actual values using annot=True and format them as integers (fmt='d'). The color map Blues adds clarity to the visual. Axes are labeled to distinguish between predicted and true classes, and custom tick labels (Class 0, Class 1) improve interpretability. The plot helps quickly identify where the model is performing well or struggling.
Below the heatmap, the classification report is printed again for reference, showing metrics like precision, recall, and F1-score for each class.

Conclusion:
In Logistic Regression, we used the price_category to predict the values in the test split of the dataset. After running the regression model using python library, the accuracy of the model comes out to be 66% which means that few values were not predicted correctly. To support this, the heatmap of the confusion matrix shows that there are false positives and false negatives in the trained model.
In case of linear regression, the 'days_left' numeric column is used to predict the 'price' in the testing set. Once the dataset is splitted, the training model is used and regression model is applied on it. The performance parameters such as MSE (9425177) and R-squared score (0.331) indicate that there is a great difference between the predicted and actual values which are also visible in the scatterplot.