**Name:Bhushan Mukund Kor** **Academic Year:2024-2025**

**Division: D15C** **Roll No: 28**

# Experiment No 5

**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 the above dataset.

**Dataset used:** https://yulimezab.github.io/Data-Mining-Project/

**Theory:** Regression analysis is a statistical method used to examine the relationship between a dependent variable and one or more independent variables. Using libraries like **Scipy** and **Scikit-learn**, we can implement both linear and logistic regression models to predict outcomes or classify data. **Linear regression** predicts continuous values, while **logistic regression** handles binary classification problems. These models are trained on historical data and evaluated using metrics such as $R^2$, MSE, and accuracy.

**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) Training 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 initialized and trained on the dataset using .fit(X, y). The coef_ attribute provides the slope, indicating the effect of days left on price, while intercept_ gives the y-intercept. These values are combined to form and display the regression equation summarizing the model.

**3) Training and Testing and 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
```

The LinearRegression() class from the sklearn.linear_model module is used to implement linear regression, which estimates the relationship between a dependent variable and one or more independent variables. In this case, it predicts airline ticket prices (dependent variable) using the number of days left before departure (independent variable).

To ensure fair evaluation, the dataset is split into training and testing sets in an 80:20 ratio using train_test_split(). The model is trained using the .fit() method, and predictions are made on the test set. Performance is evaluated using metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared ($R^2$). The coefficient indicates how price changes with each additional day left, and the intercept represents the predicted price when no days are left.

If the $R^2$ score is low or the error values are high, it suggests that days_left alone is not a strong predictor of ticket prices. The model's linear equation is derived from the slope and intercept values.

**4)Regression Line and Plot Graph**

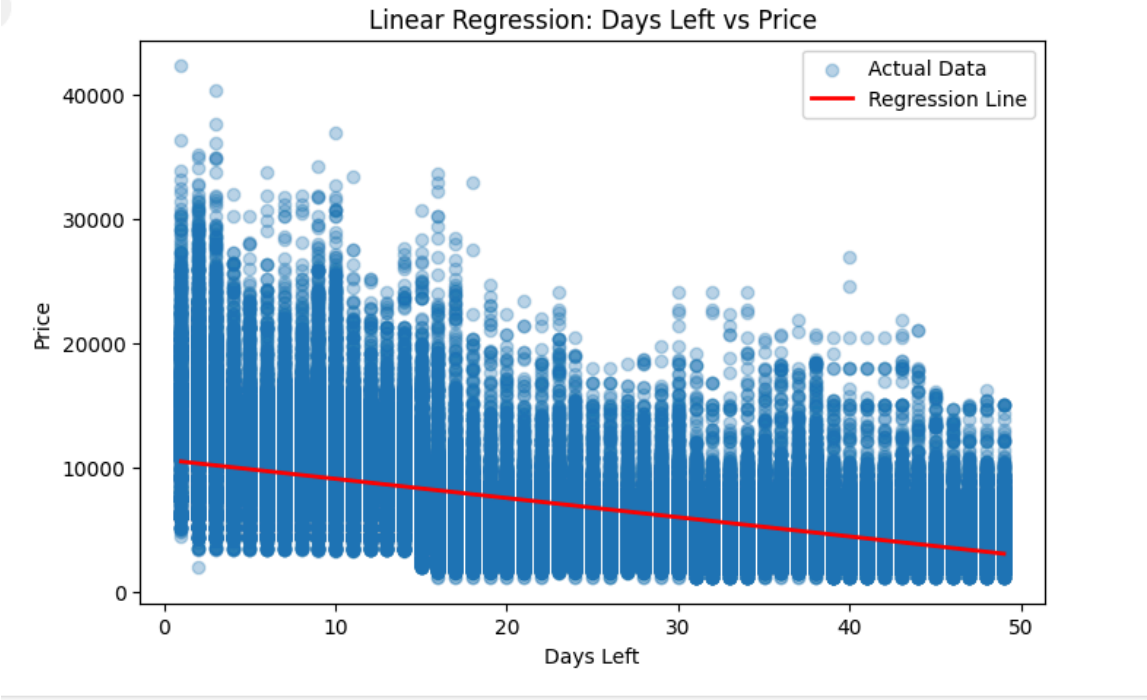
**Code:**

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

**Name:Bhushan Mukund Kor** **Academic Year:2024-2025**

**Division: D15C** **Roll No: 28**

**Output:**




In this step, we visualize the output of the linear regression model through a scatter plot and a regression line. A range of values between the minimum and maximum of days_left is generated using np.linspace(), and the trained model predicts the corresponding prices, forming a smooth regression line.

With matplotlib.pyplot, the actual data is displayed as a scatter plot, and the predicted values are shown as a red line. The alpha=0.3 setting makes the data points semi-transparent to enhance visibility. Proper labeling illustrates the relationship between ticket price and days left, helping to visually determine if a linear pattern exists.

**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)
```

The LogisticRegression() class from the sklearn.linear_model module is used to handle binary classification tasks. Here, X represents the input features and y represents the target labels. The data is divided into training and testing sets using train_test_split(), with 80% allocated for training and 20% for testing.
Because logistic regression is sensitive to the scale of input features, StandardScaler() is used to standardize the data, ensuring a mean of 0 and a standard deviation of 1. The model is then trained using the .fit() method, which determines the optimal weights to differentiate between the classes. These learned weights are used to estimate the probability of each data point belonging to a specific 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
```

Once the logistic regression model is trained, its performance is assessed using various metrics from the sklearn.metrics module. Predictions on the test data are made using the .predict() method. Accuracy is then calculated to determine the percentage of correct predictions out of all predictions.

The confusion matrix breaks down the results into true positives, true negatives, false positives, and false negatives, helping to evaluate prediction balance. The classification report provides detailed insights into each class through precision, recall, and F1-score. With an accuracy of 66%, the model shows moderate performance, but the results suggest it could be improved through tuning or by incorporating more relevant features.

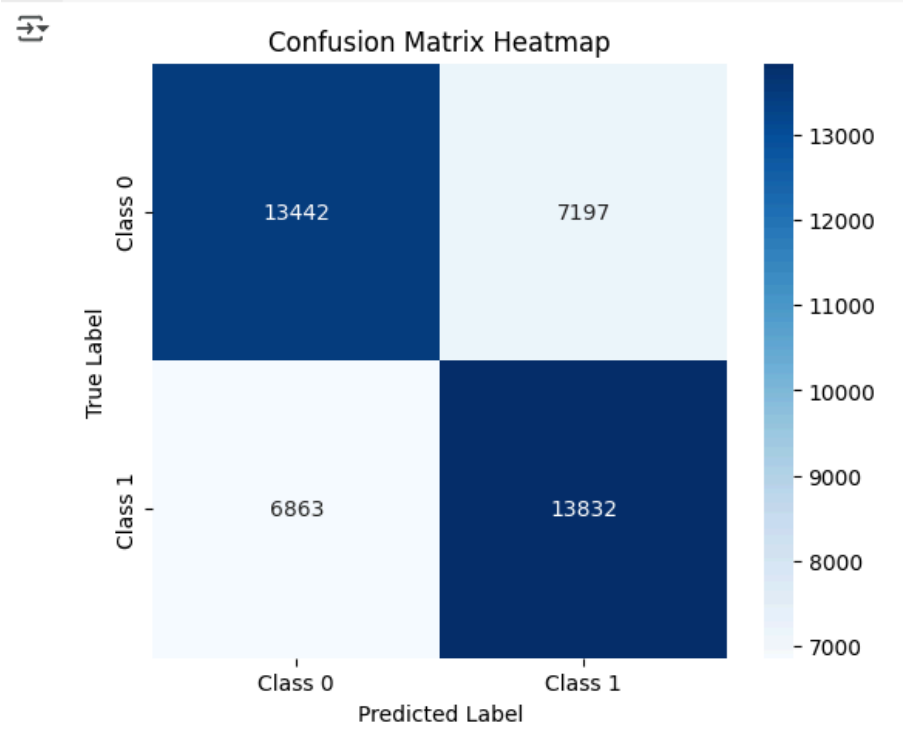
**4) Visualize the Logistic Regression**

**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 how well the logistic regression model performs, Seaborn's heatmap() function is used to display the confusion matrix. This heatmap clearly shows the comparison between actual and predicted values, with darker shades representing higher counts.

Annotations are added using annot=True and formatted as integers with fmt='d', while the 'Blues' color map enhances visual clarity. The axes are labeled to distinguish predicted versus actual classes, and custom tick labels (Class 0, Class 1) make the chart easier to interpret. This visual helps quickly spot areas where the model performs well or needs improvement.

Below the heatmap, the classification report is displayed again, summarizing key metrics like precision, recall, and F1-score for each class.

**Conclusion:**

In the Logistic Regression model, the price_category variable was used to predict outcomes in the test portion of the dataset. The model achieved an accuracy of 66%, indicating that some predictions were incorrect. This is further confirmed by the confusion matrix heatmap, which reveals the presence of false positives and false negatives.

For Linear Regression, the numerical feature days_left was used to predict the price. After splitting the dataset, the model was trained and applied to the test set. Evaluation metrics like MSE (9425177) and R-squared (0.331) show a significant gap between predicted and actual values, which is also evident from the scatter plot visualization.