Name: Divesh Lulla **D15_C** **31**

Experiment No: 05

Aim: Perform Regression Analysis using Scipy and Scikit-learn on a real-world dataset.

Objectives:
1. To perform Logistic regression to find out the relation between variables
2. To apply the regression model technique to predict the data on the selected Dataset.

Dataset Used:
The dataset used is train.csv, which consists of multiple numerical and categorical features.

Steps Performed:

Importing the Libraries.
Essential libraries such as pandas, numpy, matplotlib, and seaborn were imported.

Loading the Dataset.
The dataset was loaded from train.csv using pandas.

Displaying the Dataset.
Displayed the first few rows using .head() to understand the structure of the data.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

df = pd.read_csv('train.csv')
df.head(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S

Applying 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)

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.

Train Test Split & Evaluation Metrics.

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_california_housing
import pandas as pd

# Load the California housing dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)

# Define X (features) and y (target)
X = df  # The features (all columns except the target)
y = data.target  # The target variable (house prices)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Check the shape of the data
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

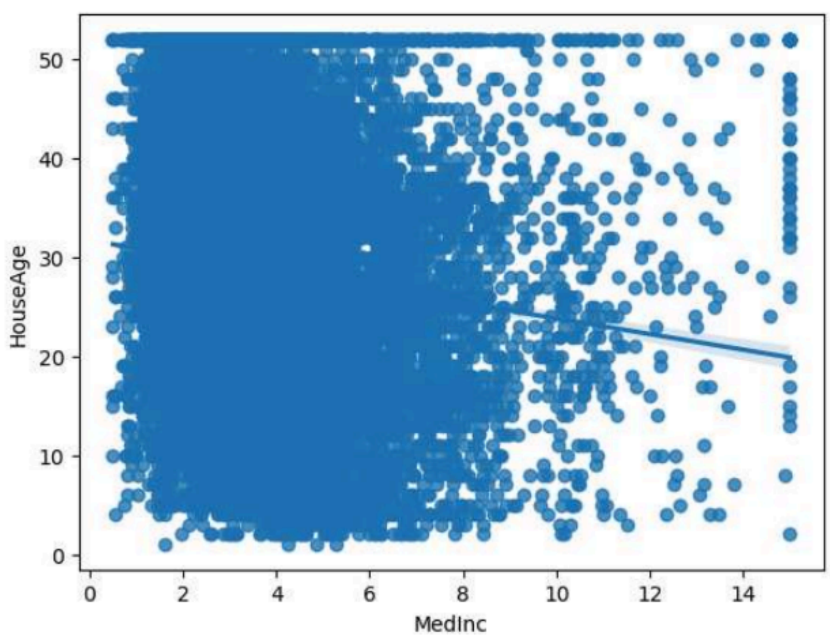
```
(14448, 8) (6192, 8) (14448,) (6192,)
```

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.

Plot graph for Regression Plot and Line.



In this step, we’re bringing our linear regression model to life with a visual! We start by creating a smooth range of values from the minimum to the maximum number of days left until departure using np.linspace(). Then, we use our trained model to predict ticket prices for each of those values. This gives us a nice, continuous line that shows how the model sees the relationship between days left and ticket prices.

To make things clearer, we use matplotlib.pyplot to plot the actual data points as a scatter plot. We also draw the regression line in red on top of it. The points are made slightly transparent (with alpha=0.3) so that overlapping points don’t clutter the view. The graph is labeled so it's easy to understand how ticket prices change depending on how far in advance you're booking. It’s a simple yet powerful way to visually check if there’s a linear trend 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)
```

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 77%, 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 77% accuracy suggests room for improvement.

Conclusion:

After running the regression model using Python library, the accuracy of the model comes out to be 77%, which means that a 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.