Experiment 05

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 dataset.

Theory:

Regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. It helps predict outcomes by fitting a line or curve to observed data points.

- **Logistic Regression**
    - Used for classification (binary or multi-class).
    - Estimates the probability of class membership using a sigmoid function.
    - Class labels are determined by applying a threshold (typically 0.5).
    - Evaluated with metrics such as accuracy, confusion matrix, precision, recall, and F1-score.
- **Linear Regression**
    - Used for predicting continuous outcomes.
    - Fits a line (or hyperplane) that minimizes the mean squared error (MSE).
    - Performance measured by MSE and the coefficient of determination (R2R^2R2 Score).

The formula for logistic regression is:

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n)}}$$

Where:

- p is the probability of the positive class (1).
- b_0 is the intercept.
- $b_1, b_2, \ldots, b_n$ are the coefficients of the independent variables $x_1, x_2, \ldots, x_n$.
- e is the base of the natural logarithm.

FEATURE ENGINEERING

- **Custom Target Creation:**
    - *Purpose:* Design a new target variable that exhibits a strong linear relationship with chosen predictors.
    - *Explanation:* For instance, we create a "NewLoanScore" as a linear combination of Income, LoanAmount, and CreditScore.
- **Noise Addition:**
    - *Purpose:* Introduce variability to simulate real-world conditions.
    - *Explanation:* Adding normally distributed noise to the custom target ensures the model does not perform perfectly, reflecting realistic data challenges.
- **Inclusion of Extra (Irrelevant) Features:**
    - *Purpose:* Test the model's robustness and demonstrate the impact of redundant information.
    - *Explanation:* Adding a random noise feature to the predictor set shows how irrelevant variables can lower performance metrics like R2R^2R2.
- **Impact on Model Performance:**
    - *Observation:* Feature engineering methods such as noise addition and extra features alter the bias-variance trade-off.
    - *Explanation:* These modifications help illustrate how controlled complexity and randomness affect metrics (e.g., MSE and R2R^2R2).

MODEL EVALUATION & METRICS

- **Evaluation for Classification (Logistic Regression):**
    - *Key Metrics:* Accuracy, confusion matrix, precision, recall, and F1-score.
    - *Purpose:* Assess how well the model distinguishes between classes and identifies misclassification errors.
- **Evaluation for Regression (Linear Regression):**
    - *Key Metrics:* Mean Squared Error (MSE) and the coefficient of determination (R2R^2R2 Score).
    - *Purpose:* MSE measures the average squared difference between predicted and actual values, while R2R^2R2 indicates the proportion of variance explained by the model.

DATA DESCRIPTION

- **Dataset Overview:**
    - Contains 255,347 entries and 18 columns.

- ○ Mix of numerical (e.g., Age, Income, LoanAmount) and categorical (e.g., Education, EmploymentType) features.
- **Key Features:**
  - *Numerical Variables:* Age, Income, LoanAmount, CreditScore, MonthsEmployed, NumCreditLines, InterestRate, LoanTerm, DTIRatio.
  - *Categorical Variables:* Education, EmploymentType, MaritalStatus, HasMortgage, HasDependents, LoanPurpose, HasCoSigner, etc.
- **Data Quality & Preprocessing:**
  - No missing values detected.
  - Categorical variables encoded using methods like LabelEncoder.
  - Numerical features standardized using StandardScaler.

Implementation:

Logistic Regression

1. **Data Preparation:**

```
[2] data = pd.read_csv('Loan_default.csv')
    print("Dataset shape:", data.shape)
    data.head()
```

```
Dataset shape: (255347, 18)
```

	LoanID	Age	Income	LoanAmount	CreditScore	MonthsEmployed	NumCreditLines	InterestRate	LoanTerm	DTIRatio	Education	EmploymentType	MaritalStatus	HasMortgage	HasDepen
0	I38PQUQS96	56	85994	50587	520	80	4	15.23	36	0.44	Bachelor's	Full-time	Divorced	Yes	
1	HPSK72WA7R	69	50432	124440	458	15	1	4.81	60	0.68	Master's	Full-time	Married	No	
2	C1OZ6DPJ8Y	46	84208	129188	451	26	3	21.17	24	0.31	Master's	Unemployed	Divorced	Yes	
3	V2KKSFM3UN	32	31713	44799	743	0	3	7.07	24	0.23	High School	Full-time	Married	No	
4	EY08JDHTZP	60	20437	9139	633	8	4	6.51	48	0.73	Bachelor's	Unemployed	Divorced	No	

```python
numeric_features = ['Age', 'Income', 'LoanAmount', 'CreditScore',
                    'MonthsEmployed', 'NumCreditLines', 'InterestRate',
                    'LoanTerm', 'DTIRatio']
categorical_features = ['Education', 'EmploymentType']

df = data.copy()

# Encode the categorical features using LabelEncoder
from sklearn.preprocessing import LabelEncoder

# Encode each categorical feature
for col in categorical_features:
    le = LabelEncoder()
    df[col + '_encoded'] = le.fit_transform(df[col])

features = numeric_features + [col + '_encoded' for col in categorical_features]
target = 'Default'
```

2. **Data Splitting & Scaling:**
   Split the dataset into training and testing subsets (e.g., 70/30 split) and apply feature scaling (using StandardScaler) to both sets.

```python
X = df[features]
y = df[target]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Fit on training data and transform both train and test sets
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("X_train shape:", X_train_scaled.shape)
print("X_test shape:", X_test_scaled.shape)
```

```
X_train shape: (178742, 11)
X_test shape: (76605, 11)
```

```python
# Initialize the Logistic Regression model (increase max_iter if needed)
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = log_reg.predict(X_test_scaled)
```

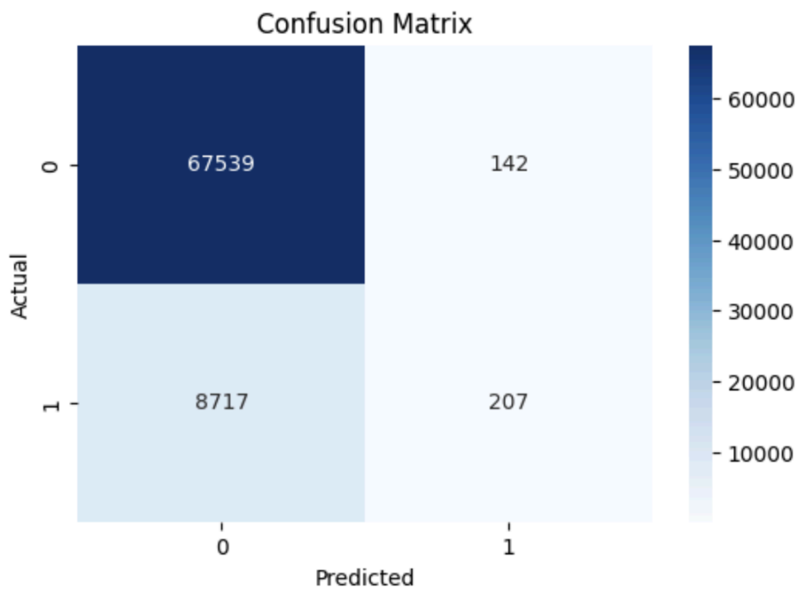
3. **Model Training:**
   Train the logistic regression model on the scaled training data.

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = log_reg.predict(X_test_scaled)
```

4. **Model Evaluation:**
   Evaluate the model using metrics such as accuracy, confusion matrix, precision, recall, and F1-score. Generate visualizations (e.g., confusion matrix heatmap) to assess performance.

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

# Calculate and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.884354807127472
Confusion Matrix:
 [[67539   142]
 [ 8717   207]]
Classification Report:
               precision    recall  f1-score   support

           0       0.89      1.00      0.94     67681
           1       0.59      0.02      0.04      8924

    accuracy                           0.88     76605
   macro avg       0.74      0.51      0.49     76605
weighted avg       0.85      0.88      0.83     76605
```



Linear Regression Implementation (with Feature Engineering)

1. **Custom Target Creation:**
   Create a new target variable (e.g., "NewLoanScore") as a linear combination of selected predictors (like Income, LoanAmount, CreditScore) and add normally distributed noise to simulate real-world variability.

```
noise = np.random.normal(0, 15000, len(df))  # Adjust the standard deviation as needed
df['NewLoanScore'] = 0.5 * df['Income'] + 2 * df['LoanAmount'] - 3 * df['CreditScore'] + noise
df.head()

df['ExtraNF'] = np.random.uniform(0, 1, len(df))
```

2. **Data Preparation & Splitting:**
   Process the dataset similarly—encode categorical variables, standardize numeric features—and then split the data into training and testing sets.

```
X = df[['Income', 'LoanAmount', 'CreditScore','ExtraNF']]
y = df['NewLoanScore']
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3. **Model Training:**
   Train the linear regression model using the engineered target variable on the training data.

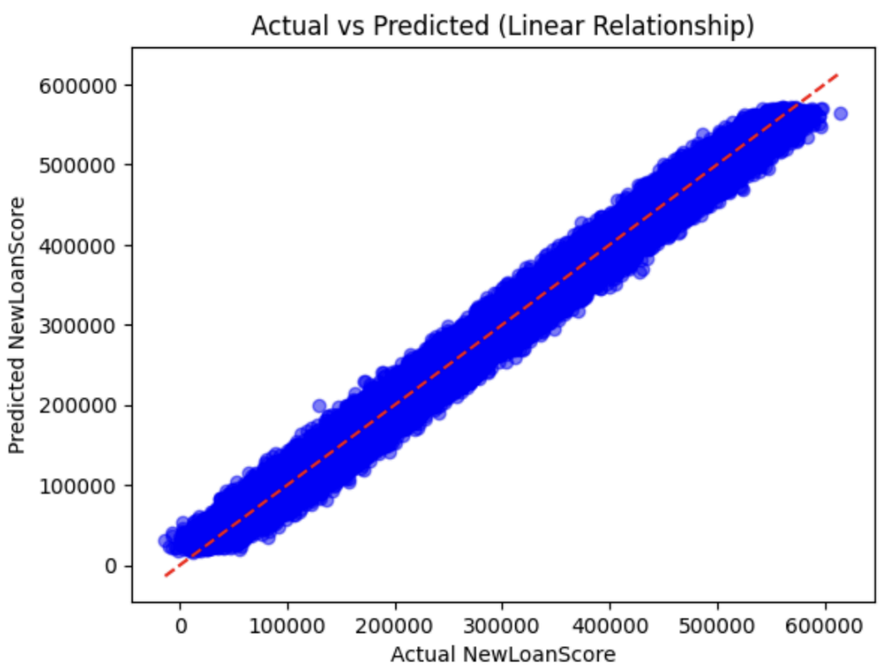
```
# Fit model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
```

4. **Model Evaluation:**
   Evaluate the regression model by calculating the Mean Squared Error (MSE) and the R2R^2R2 Score, and visualize the relationship between actual and predicted values (e.g., scatter plot with a reference line).

```python
# Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Linear Regression Mean Squared Error: {mse}")
print(f"Linear Regression R^2 Score: {r2}")
```

```
Linear Regression Mean Squared Error: 224664554.19170806
Linear Regression R^2 Score: 0.9891091055812746
```

```python
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='dashed')
plt.xlabel("Actual NewLoanScore")
plt.ylabel("Predicted NewLoanScore")
plt.title("Actual vs Predicted (Linear Relationship)")
plt.show()
```



Conclusion:

The logistic regression model achieved about 88.4% accuracy on the test set, with the confusion matrix highlighting lower performance for the minority class. The linear regression model initially produced near-perfect results, but after adding noise and an extra irrelevant feature, its performance became more realistic—with an R^2 score of around 0.989 and a higher Mean Squared Error. These outcomes clearly demonstrate the impact of noise and feature engineering on model performance. Screenshots of key outputs (e.g., confusion matrix, scatter plots, and metric summaries) are included in the detailed implementation section.