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

Theory:
Regression analysis is a statistical technique used to model and analyze relationships between variables. It is commonly used for predicting numerical outcomes and identifying trends. There are different types of regression, with Linear Regression being the most fundamental.
Regression analysis helps in understanding:

- The relationship between dependent and independent variables.
- The impact of independent variables on the dependent variable.
- Predicting values based on trends in data.

Mathematically, it follows the equation:

$$y = \beta_0 + \beta_1 x + \epsilon$$

Where:

y is the dependent variable (what we predict).

x is the independent variable (the predictor).

β_0 is the intercept (constant term).

β_1 is the coefficient (slope).

ϵ is the error term (random noise).

There can also exist more than one independent variable, in such a case, regression follows the equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

Types of Regression:

- Linear Regression (Simple & Multiple)
- Polynomial Regression (Non-linear relationship)
- Logistic Regression (Classification problems)
- Ridge & Lasso Regression (Regularization techniques)
- Support Vector Regression (SVR)
- Decision Tree & Random Forest Regression

Steps:

a. **Linear Regression**

1. **Load the dataset**

```
import pandas as pd
import numpy as np
df = pd.read_csv("/content/bankdataset.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 192114 entries, 0 to 192113
Data columns (total 5 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Date               192114 non-null  object
 1   Domain             192114 non-null  object
 2   Location           192114 non-null  object
 3   Value              192113 non-null  float64
 4   Transaction_count  192113 non-null  float64
dtypes: float64(2), object(3)
memory usage: 7.3+ MB
```

The dataset consists of **192,114 entries** with **5 columns**, capturing various aspects of financial transactions across different domains and locations. It contains a mix of categorical and numerical variables. The **categorical** attributes include **Date, Domain, and Location**, which provide insights into the time, type of transaction, and geographical information. The **numerical** attributes include **Value** and **Transaction_count**, representing the financial value of the transactions and the number of transactions recorded. This dataset can be used to analyze patterns, identify trends across domains and locations, and perform regression analysis to predict future transaction values or counts.

2. **Perform all the necessary preprocessing steps**
    a. Handling missing values
    b. Drop unnecessary columns
    c. Data transformation

```
# 1. Handling missing values
# Drop rows with missing values
df.dropna(inplace=True)

# 2. Drop unnecessary columns (if any)
# Assuming all columns are necessary for now; modify as needed

# 3. Data Transformation

# a) Convert 'Date' to datetime format (handle mixed formats)
df['Date'] = pd.to_datetime(df['Date'], format='mixed', dayfirst=True)

# Extract useful features from the date
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day

# Drop the original Date column if not needed
df.drop('Date', axis=1, inplace=True)

# b) Encode categorical variables using one-hot encoding
df = pd.get_dummies(df, columns=['Domain', 'Location'], drop_first=True)

# c) Normalize numerical columns ('Value' and 'Transaction_count')
scaler = MinMaxScaler()
df[['Value', 'Transaction_count']] = scaler.fit_transform(df[['Value', 'Transaction_count']])

# Display the transformed dataset
print(df.head())
```

```
      Value  Transaction_count  Year  Month  Day  Domain_INTERNATIONAL  \
0  0.074272           0.713222  2022      1    1                 False
1  0.607426           0.614991  2022      1    1                 False
2  0.540487           0.546089  2022      1    1                 False
3  0.077654           0.767691  2022      1    1                  True
4  0.351008           0.520950  2022      1    1                 False

   Domain_INVESTMENTS  Domain_MEDICAL  Domain_PUBLIC  Domain_RESTRAUNT  ...  \
0               False           False          False              True  ...
```

Convert categorical columns to binary data.

3. **Split data and train the model**

```
# Define Features and Target
X = df[['Value']]  # Independent Variable
y = df['Transaction_count']  # Dependent Variable

# Train-Test Split (75%-25%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Apply Linear Regression
model = LinearRegression()
model.fit(X_train, y_train)

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

Split data into 75% training and 25% testing datasets.

Regression is sensitive to sudden changes in ranges of independent variables, to avoid this we normalize our numerical columns.
Then we train our model, and use the model to predict the testing dataset.

4. **Evaluation**

```
Mean Squared Error: 0.08337233857631968
R-squared Score: -1.229404256064548e-05
Coefficient (Slope): 0.0014923215201184602, Intercept: 0.49894002234129436
```

The model's **R^2 score is very close to zero**, indicating that it **fails to explain** the relationship between `Value` and `Transaction_count`. This suggests that **linear regression may not be the right fit** for your data, and **more features** or a **different model** (like **Decision Trees** or **Random Forest**) could improve accuracy.

b. Logistic regression :

Its a supervised classification algorithm used to predict binary (0/1) or multi-class outcomes. Despite its name, it is actually a classification model, not a regression model

- Logistic Regression estimates the probability **P(Y=1 | X)** using the **sigmoid (logistic) function**. The output is a probability value between **0 and 1**, which is then classified using a **threshold** (e.g., ≥0.5 as class 1, <0.5 as class 0).

1. Splits the selected feature dataset into training (75%) and testing (25%) set

```
# Define features (X) and target (y)
X = df.drop('Transaction_count', axis=1)
y = (df['Transaction_count'] > 0.5).astype(int)  # Binary target for logistic regression

# Split dataset 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)
```

```
# Initialize and train logistic regression model
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)

# Make predictions
y_pred = log_reg.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.49942590527768926
Confusion Matrix:
 [[82244 68497]
 [82348 68255]]
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.55      0.52    150741
           1       0.50      0.45      0.48    150603

    accuracy                           0.50    301344
   macro avg       0.50      0.50      0.50    301344
weighted avg       0.50      0.50      0.50    301344
```

Output indicates that the Logistic Regression model performed poorly. The model achieved an accuracy of 49.94%, which is close to random guessing (50%), suggesting it lacks predictive power. The confusion matrix reveals that the model correctly classified 82,244 instances as negative and 68,255 as positive, but misclassified 68,497 negative cases and 82,348 positive cases. This high misclassification rate indicates the model struggles to distinguish between the two classes effectively.

The classification report further supports this conclusion, showing a precision of 0.50 for both classes, meaning only half of the positive predictions were correct. The recall values of 0.55 for class 0 and 0.45 for class 1 indicate that the model misses a significant portion of the actual cases. Additionally, the F1-score of 0.50 reflects a poor balance between precision and recall across both classes

Conclusion : Both the linear and logistic regression models show **poor performance**. The linear model has a **near-zero R-squared**, indicating it cannot explain the target variable. The logistic model's **49.94% accuracy** is close to random guessing, with weak precision, recall, and F1-scores. **Improvement** requires better **feature selection**, handling **class imbalance**, or using **advanced models** like Random Forest or XGBoost.

Conclusion:

Both the linear and logistic regression models show **poor performance**. The linear model has a **near-zero R-squared**, indicating it cannot explain the target variable. The logistic model's **49.94% accuracy** is close to random guessing, with weak precision, recall, and F1-scores. **Improvement** requires better **feature selection**, handling **class imbalance**, or using **advanced models** like Random Forest or XGBoost.