UCS2612 Machine Learning Laboratory

Assignment on Linear Regression with User Defined Functions

Name : **Mega V** Roll No : **3122 21 5001 051**

Aim

Write the python code from scratch to implement Linear Regression without using Scikit-learn library or built in functions.

CODE and Output

Loading The Dataset

```
import numpy as np
import pandas as pd
import scipy as sl
import matplotlib.pyplot as mat
```

```
train_df=pd.read_csv("train.csv")
test_df=pd.read_csv("test.csv")
```

```
Printing The Train and Test Dataset
```

```
print("\n\nThe Shapes Of the Datase : ",train_df.shape)
```

```
The Shapes Of the Datase :  (30000, 24)
```

```
print("\n\nThe Type of The Attribues in the Dataset\n\n",train_df.dtypes)
```

```
The Type of The Attribues in the Dataset

 Customer ID                    object
Name                            object
Gender                          object
Age                              int64
Income (USD)                   float64
Income Stability                object
Profession                      object
Type of Employment              object
Location                        object
Loan Amount Request (USD)      float64
Current Loan Expenses (USD)    float64
Expense Type 1                  object
Expense Type 2                  object
Dependents                     float64
Credit Score                   float64
No. of Defaults                  int64
Has Active Credit Card          object
Property ID                      int64
Property Age                   float64
Property Type                    int64
Property Location               object
Co-Applicant                     int64
Property Price                 float64
Loan Sanction Amount (USD)     float64
dtype: object
```

Pre-Processing the data (Handling missing values, Encoding, Normalization, Standardization).

```
print("\n\nThe Number of Missing Values in Each Attributes in Training
Dataset\n\n",train_df.isnull().sum())
```

```
The Number of Missing Values in Each Attributes in Training Dataset

 Customer ID                     0
Name                             0
Gender                          53
Age                              0
```

```
Income (USD)                    4576
Income Stability                1683
Profession                         0
Type of Employment              7270
Location                           0
Loan Amount Request (USD)          0
Current Loan Expenses (USD)      172
Expense Type 1                     0
Expense Type 2                     0
Dependents                      2493
Credit Score                    1703
No. of Defaults                    0
Has Active Credit Card          1566
Property ID                        0
Property Age                    4850
Property Type                      0
Property Location                356
Co-Applicant                       0
Property Price                     0
Loan Sanction Amount (USD)       340
dtype: int64
```

```
print("\n\nThe Number of Missing Values in Each Attributes in Training
Dataset\n\n",test_df.isnull().sum())
```

```
The Number of Missing Values in Each Attributes in Training Dataset

 Customer ID                     0
Name                             0
Gender                          31
Age                              0
Income (USD)                   750
Income Stability               813
Profession                       0
Type of Employment            4689
Location                         0
Loan Amount Request (USD)        0
Current Loan Expenses (USD)     83
Expense Type 1                   0
Expense Type 2                   0
Dependents                    1142
Credit Score                   743
```

```
No. of Defaults                0
Has Active Credit Card      1076
Property ID                    0
Property Age                 892
Property Type                  0
Property Location            160
Co-Applicant                  77
Property Price               168
dtype: int64
```

```
The Gender is the categorical features. So handling the error fill the Most
occured gender in that Missing values
```

```
most_frequent_train_gender = train_df['Gender'].mode()[0]
train_df['Gender'] = train_df['Gender'].fillna(most_frequent_train_gender)

most_frequent_test_gender = test_df['Gender'].mode()[0]
test_df['Gender'] = test_df['Gender'].fillna(most_frequent_test_gender)

print("The Most Occurred Gender in Traing Data : ",most_frequent_train_gender)
print("The Most Occurred Gender in Test Data   : ",most_frequent_test_gender)
```

```
The Most Occurred Gender in Traing Data :  M
The Most Occurred Gender in Test Data   :  F
```

```
The Income (USD) attribute has the numerical values. So handling the error
fill the median of Income attribute in that Missing values
```

```
train_median=train_df['Income (USD)'].median()
test_median=test_df['Income (USD)'].median()

print("The Median of Income (USD) in Traing Data : ",train_median)
print("The Median of Income (USD) in Test Data   : ",test_median)

train_df['Income (USD)']=train_df['Income (USD)'].fillna(train_median)
test_df['Income (USD)']=test_df['Income (USD)'].fillna(test_median)
```

```
The Median of Income (USD) in Traing Data :  2222.435
The Median of Income (USD) in Test Data   :  2224.59
```

```
The Income Stability is the categorical features. So handling the error fill the Most occured in that Missing values
```

```
train_Income_stabilty = train_df['Income Stability'].mode()[0]
train_df['Income Stability'] = train_df['Income Stability'].fillna(train_Income_stabilty)

test_Income_stabilty = test_df['Income Stability'].mode()[0]
test_df['Income Stability'] = test_df['Income Stability'].fillna(test_Income_stabilty)

print("The Most Occurred Gender in Traing Data : ",train_Income_stabilty)
print("The Most Occurred Gender in Test Data   : ",test_Income_stabilty)
```

```
The Most Occurred Gender in Traing Data :  Low
The Most Occurred Gender in Test Data   :  Low
```

```
The Type Of Employment is categorical feature. It has the more number of missing values. So Unging the probabilistic imputation we can fill the Missing Values.
```

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier

features_for_imputation = ['Income (USD)', 'Loan Amount Request (USD)','Current Loan Expenses (USD)']

target_feature = 'Type of Employment'
features_for_modeling = ['Income (USD)', 'Loan Amount Request (USD)','Current Loan Expenses (USD)']

data_with_missing = train_df[train_df[target_feature].isnull()]
data_without_missing = train_df.dropna(subset=[target_feature])

classifier = RandomForestClassifier()
```

```
classifier.fit(data_without_missing[features_for_modeling],
data_without_missing[target_feature])

predicted_values =
classifier.predict(data_with_missing[features_for_modeling])
print(predicted_values)
probabilistic_imputation = np.random.choice(predicted_values,
size=len(data_with_missing))
print(probabilistic_imputation)

train_df.loc[train_df[target_feature].isnull(), target_feature] =
probabilistic_imputation
```

```
['Managers' 'Laborers' 'Laborers' ... 'Sales staff' 'Laborers' 'Drivers']
['Laborers' 'Laborers' 'Laborers' ... 'Laborers' 'Sales staff' 'Laborers']
```

```
features_for_imputation = ['Income (USD)', 'Loan Amount Request
(USD)','Current Loan Expenses (USD)']

target_feature = 'Type of Employment'
features_for_modeling = ['Income (USD)', 'Loan Amount Request (USD)','Current
Loan Expenses (USD)']

data_with_missing = test_df[test_df[target_feature].isnull()]
data_without_missing = test_df.dropna(subset=[target_feature])

classifier = RandomForestClassifier()
classifier.fit(data_without_missing[features_for_modeling],
data_without_missing[target_feature])

predicted_values =
classifier.predict(data_with_missing[features_for_modeling])
print(predicted_values)
probabilistic_imputation = np.random.choice(predicted_values,
size=len(data_with_missing))
print(probabilistic_imputation)

test_df.loc[test_df[target_feature].isnull(), target_feature] =
probabilistic_imputation
```

```
['Laborers' 'Laborers' 'Laborers' ... 'Laborers' 'Sales staff' 'Laborers']
['Laborers' 'Laborers' 'Sales staff' ... 'Laborers' 'Laborers' 'Managers']
```

The Current Loan Expenses (USD) attribute has the numerical values. So handling the error fill the median of Income attribute in that Missing values

```
train_median=train_df['Current Loan Expenses (USD)'].median()
test_median=test_df['Current Loan Expenses (USD)'].median()

print("The Median of Income (USD) in Traing Data : ",train_median)
print("The Median of Income (USD) in Test Data   : ",test_median)

train_df['Current Loan Expenses (USD)']=train_df['Current Loan Expenses
(USD)'].fillna(train_median)
test_df['Current Loan Expenses (USD)']=test_df['Current Loan Expenses
(USD)'].fillna(test_median)

test_median=test_df['Property Price'].median()
test_df['Property Price']=test_df['Property Price'].fillna(test_median)
```

```
The Median of Income (USD) in Traing Data :  375.205
The Median of Income (USD) in Test Data   :  374.0
```

The Dependents is the categorical features. So handling the error fill the Most occured in that Missing values

```
train_Dependents = train_df['Dependents'].mode()[0]
train_df['Dependents'] = train_df['Dependents'].fillna(train_Dependents)

test_Dependents = test_df['Dependents'].mode()[0]
test_df['Dependents'] = test_df['Dependents'].fillna(test_Dependents)

print("The Most Occurred Gender in Traing Data : ",train_Dependents)
print("The Most Occurred Gender in Test Data   : ",test_Dependents)
```

```
The Most Occurred Gender in Traing Data :  2.0
The Most Occurred Gender in Test Data   :  2.0
```

The Credit Score attribute has the numerical values. So handling the error fill the median of Income attribute in that Missing values

```
train_median=train_df['Credit Score'].median()
test_median=test_df['Credit Score'].median()

print("The Median of Income (USD) in Traing Data : ",train_median)
print("The Median of Income (USD) in Test Data   : ",test_median)

train_df['Credit Score']=train_df['Credit Score'].fillna(train_median)
test_df['Credit Score']=test_df['Credit Score'].fillna(test_median)
```

```
The Median of Income (USD) in Traing Data :  739.82
The Median of Income (USD) in Test Data   :  739.3
```

```
The Has Active Credit Card is the categorical features. So handling the error
fill the Most occured in that Missing values
```

```
features_for_imputation = ['Dependents','Current Loan Expenses (USD)','Credit
Score']

target_feature = 'Has Active Credit Card'
features_for_modeling = ['Dependents','Current Loan Expenses (USD)','Credit
Score']

data_with_missing = test_df[test_df[target_feature].isnull()]
data_without_missing = test_df.dropna(subset=[target_feature])

classifier = RandomForestClassifier()
classifier.fit(data_without_missing[features_for_modeling],
data_without_missing[target_feature])

predicted_values =
classifier.predict(data_with_missing[features_for_modeling])
print(predicted_values)
probabilistic_imputation = np.random.choice(predicted_values,
size=len(data_with_missing))
print(probabilistic_imputation)

test_df.loc[test_df[target_feature].isnull(), target_feature] =
probabilistic_imputation
```

```
['Inactive' 'Unpossessed' 'Unpossessed' ... 'Active' 'Active'
 'Unpossessed']
['Inactive' 'Unpossessed' 'Unpossessed' ... 'Unpossessed' 'Active'
 'Unpossessed']
```

```
features_for_imputation = ['Dependents','Income (USD)', 'Loan Amount Request
(USD)','Current Loan Expenses (USD)']

target_feature = 'Has Active Credit Card'
features_for_modeling = ['Dependents','Income (USD)', 'Loan Amount Request
(USD)','Current Loan Expenses (USD)']

data_with_missing = train_df[train_df[target_feature].isnull()]
data_without_missing = train_df.dropna(subset=[target_feature])

classifier = RandomForestClassifier()
classifier.fit(data_without_missing[features_for_modeling],
data_without_missing[target_feature])

predicted_values =
classifier.predict(data_with_missing[features_for_modeling])
print(predicted_values)
probabilistic_imputation = np.random.choice(predicted_values,
size=len(data_with_missing))
print(probabilistic_imputation)

train_df.loc[train_df[target_feature].isnull(), target_feature] =
probabilistic_imputation
```

```
['Unpossessed' 'Unpossessed' 'Inactive' ... 'Active' 'Inactive' 'Active']
['Inactive' 'Active' 'Active' ... 'Active' 'Active' 'Inactive']
```

```
The Property Age and Loan Sanction Amount (USD) attribute has the numerical
values. So handling the error fill the median of Income attribute in that
Missing values
```

```
train_median=train_df['Property Age'].median()
test_median=test_df['Property Age'].median()
```

```
print("The Median of Income (USD) in Traing Data : ",train_median)
print("The Median of Income (USD) in Test Data   : ",test_median)

train_df['Property Age']=train_df['Property Age'].fillna(train_median)
test_df['Property Age']=test_df['Property Age'].fillna(test_median)

train_median = train_df['Loan Sanction Amount (USD)'].median()

print("The Median of Loan Sanction Amount (USD) in Training Data: ",
train_median)

train_df['Loan Sanction Amount (USD)'] = train_df['Loan Sanction Amount
(USD)'].fillna(train_median)
```

```
The Median of Income (USD) in Traing Data :  2223.25
The Median of Income (USD) in Test Data   :  2220.6049999999996
The Median of Loan Sanction Amount (USD) in Training Data:  35209.395000000004
```

```
The Income Stability is the categorical features. So handling the error fill
the Most occured in that Missing values
```

```
train_Property_Location = train_df['Property Location'].mode()[0]
train_df['Property Location'] = train_df['Property
Location'].fillna(train_Property_Location )
test_Property_Location  = test_df['Property Location'].mode()[0]
test_df['Property Location'] = test_df['Property
Location'].fillna(test_Property_Location )
print("The Most Occurred Gender in Traing Data : ",train_Property_Location )
print("The Most Occurred Gender in Test Data   : ",test_Property_Location )
```

```
The Most Occurred Gender in Traing Data :  Semi-Urban
The Most Occurred Gender in Test Data   :  Semi-Urban
```

```
print("\n\nThe Number of Missing Values in Each Attributes in Training
Dataset\n\n",train_df.isnull().sum())
```

```
The Number of Missing Values in Each Attributes in Training Dataset

 Customer ID                    0
Name                            0
Gender                          0
Age                             0
Income (USD)                    0
Income Stability                0
Profession                      0
Type of Employment              0
Location                        0
Loan Amount Request (USD)       0
Current Loan Expenses (USD)     0
Expense Type 1                  0
Expense Type 2                  0
Dependents                      0
Credit Score                    0
No. of Defaults                 0
Has Active Credit Card          0
Property ID                     0
Property Age                    0
Property Type                   0
Property Location               0
Co-Applicant                    0
Property Price                  0
Loan Sanction Amount (USD)      0
dtype: int64
```

```
from sklearn.preprocessing import MinMaxScaler

columns_to_normalize = ['Loan Amount Request (USD)', 'Income (USD)', 'Property
Price']

scaler = MinMaxScaler()

X_sample_normalized = train_df.copy()
X_sample_normalized[columns_to_normalize] =
scaler.fit_transform(X_sample_normalized[columns_to_normalize])
train_df=X_sample_normalized
```

```
test_df = test_df.drop(columns=['Name', 'Customer ID'])
train_df = train_df.drop(columns=['Name', 'Customer ID'])
```

Selecting The Featurs from Existing featurs

```
from sklearn.feature_selection import SelectKBest, f_regression

numerical_columns = train_df.select_dtypes(include=['float64',
'int64']).columns

categorical_columns = train_df.select_dtypes(include=['object']).columns

encoded_categorical = pd.get_dummies(train_df[categorical_columns])

sample_data_encoded = pd.concat([train_df[numerical_columns],
encoded_categorical], axis=1)

X_sample = sample_data_encoded.drop(columns=['Loan Sanction Amount (USD)'])
y_sample = sample_data_encoded['Loan Sanction Amount (USD)']

selector = SelectKBest(score_func=f_regression, k=15)
selector.fit(X_sample, y_sample)

selected_features = X_sample.columns[selector.get_support()]

print("Selected Features:")
print(selected_features)
```

```
Selected Features:
Index(['Loan Amount Request (USD)', 'Current Loan Expenses (USD)',
       'Credit Score', 'Property Price', 'Income Stability_Low',
       'Profession_Commercial associate', 'Profession_Pensioner',
       'Profession_Working', 'Type of Employment_Accountants',
       'Type of Employment_Laborers', 'Type of Employment_Managers',
       'Location_Semi-Urban', 'Location_Urban', 'Expense Type 1_N',
       'Expense Type 1_Y'],
      dtype='object')
```

Linear Regression Model

Linear regression Model Fit

```
def linear_regression_Fit(X, y):

    X = np.column_stack((np.ones(len(X)), X))

    coefficients = np.linalg.inv(X.T @ X) @ X.T @ y

    intercept = coefficients[0]
    coefficients = coefficients[1:]

    return coefficients, intercept
```

Linear regression Model Testing

```
def linear_regression_testing(x,y,coefficients,intercept):
    predictions=[]
    for i in range(len(x)):
        amount=0
        for j in range(len(coefficients)):
            amount+=coefficients[j]*x.iloc[i,j]
        amount+=intercept
        amount=amount*y[i]
        predictions.append(amount)
    return predictions
```

Calculating The Mean Squared Error and R^2 Score For The Linear regression Model

```
def mean_squared_Error(actual,predicted):
    sum=0
    for i in range(len(actual)):
        sum+=((predicted[i]-actual.iloc[i])*(predicted[i]-actual.iloc[i]))
    mse=sum/len(actual)
    return mse

def R2_Error(actual,predicted):
    sum1 = 0
```

```
    sum2 = 0
    mean_actual = actual.mean()
    for i in range(len(actual)):
        sum1 += ((actual.iloc[i] - predicted[i]) ** 2)
        sum2 += ((actual.iloc[i] - mean_actual) ** 2)
    r_squared = 1 - (sum1.sum() / sum2.sum())
    return r_squared
```

Spliting the Training and testing dataset

```
X=df.drop(columns={"Loan Sanction Amount (USD)"})
y=df["Loan Sanction Amount (USD)"]
```

```
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state
=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(22500, 11)
(7500, 11)
(22500,)
(7500,)
```

Model Training For Dataset

```
coefficients,intercept=linear_regression_Fit(x_train,y_train)
print("\n\nThe Slope or Coeifficients Values \n",coefficients)
print("\n\nThe Intercept Value               :",intercept)
```

```
The Slope or Coeifficients Values
 [ 3.31288069e+02 -5.91885164e+01 -3.20365699e-01  5.72616256e-01
 -1.74797222e+00 -1.44398045e+03  1.65547984e+02 -9.36858533e+02
  2.50499732e+02 -6.11143667e+00 -1.31792009e-02]


The Intercept Value               : -117553.02261896536
```

```
predictions=linear_regression_testing(x_test,y_label,coefficients,intercept)
print(y_test)
print(predictions)
```

```
Calculate the Mean Squared Error and R^2 Score
```

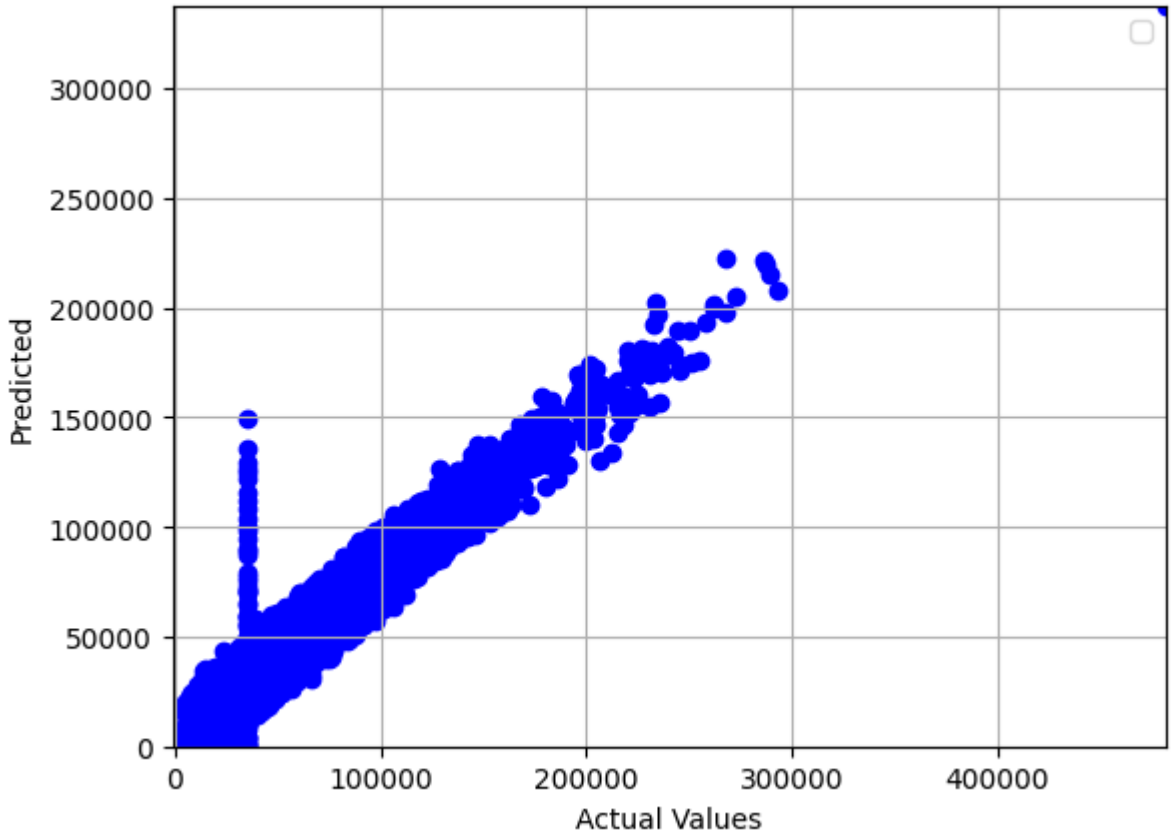
```
print("\nMean Squared Error : ",mean_squared_Error(y_test,predictions))
print("R^2 Score          : ",R2_Error(y_test,predictions))
print("Accuracy is        : ",R2_Error(y_test,predictions)*100)
```

```
Mean Squared Error :  335774685.32100934
R^2 Score          :  0.8521605786409783
Accuracy is        :  85.21605786409782
```

Represent the results using graphs

```
import matplotlib.pyplot as plt

plt.scatter(y_test, y_pred, color='blue')
plt.title('Actual vs Predicted Loan Sanction Amount')
plt.xlabel('Actual Amount (USD)')
plt.ylabel('Predicted Amount (USD)')
plt.show()
```



Learning Outcome :

1) Implement linear regression using user-defined methods to grasp fundamental regression concepts.
2) Prepare and preprocess data for linear regression analysis, including feature scaling and handling missing values.
3) Understand the mathematical foundation of linear regression, including the ordinary least squares (OLS) method for parameter estimation.
4) Explore model evaluation techniques such as R-squared, mean squared error (MSE), and adjusted R-squared.
5) Interpret regression coefficients
6) Apply linear regression to real-world datasets for tasks such as predicting loan amount