Cairo University

Faculty of Computers and Artificial Intelligence

SCS492-Selected Topics in Software Engineering-2 [Machine Learning]

Assignment - 1

Supervised By:

DR.Hanaa Mobarez

TA.Abdalrahman Roshdi

Prepared & Implemented By:

Mootaz Medhat Ezzat Abdelwahab (20206074)

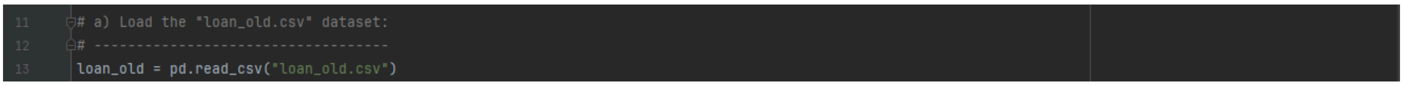
➢ Documentation of Code Execution and Results:

- ➢ a) Load the "loan_old.csv" dataset
- ➢ h) Load "loan_new.csv" dataset
- ➢ b) Perform analysis on the dataset:
  - • i) check whether there are missing values in "loan_old.csv" dataset

    check whether there are missing values in "loan_new.csv" dataset
  - • ii) check the type of each column (categorical or numerical)
  - • iii) check whether numerical columns have the same scale
  - • iv) visualize a pairplot between numerical columns
- ➢ c) Preprocess the data:
  - • i) remove records with missing values from the original "loan_old" DataFrame directly and returns None

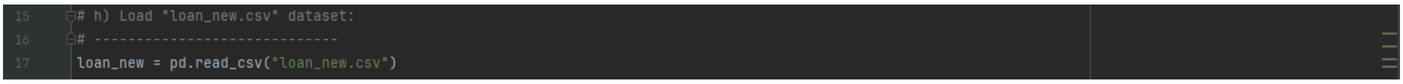
    remove records with missing values from the original "loan_new" DataFrame directly and returns None
  - • ii) separate features and targets
  - • iii) shuffle and split the data into training and testing sets using train_test_split() function from scikit-learn, it randomly shuffles the data before splitting it into training and testing sets
  - • iv) encode categorical features into numerical labels
  - • v) encode categorical targets into numerical labels
  - • vi) standardize the features using the mean and standard deviation
- ➢ d) Fit linear regression model
- ➢ e) Evaluate linear regression model
- ➢ f) Fit logistic regression model from scratch
- ➢ g) Function (from scratch) to calculate the accuracy of the logistic regression model
- ➢ j) Predict using models
- ➢ Print predictions for loan amounts and loan status for new data

➢ a) Load the "loan_old.csv" dataset:



```
# a) Load the "loan_old.csv" dataset:
# ---------------------------------
loan_old = pd.read_csv("loan_old.csv")
```

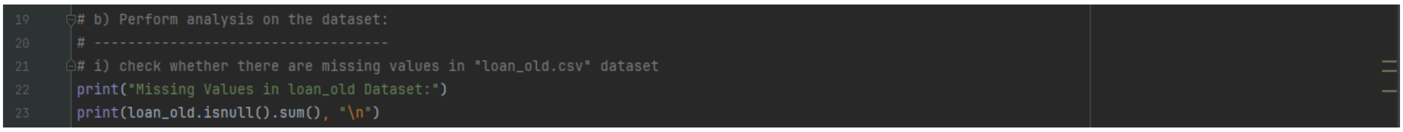
➢ h) Load "loan_new.csv" dataset:



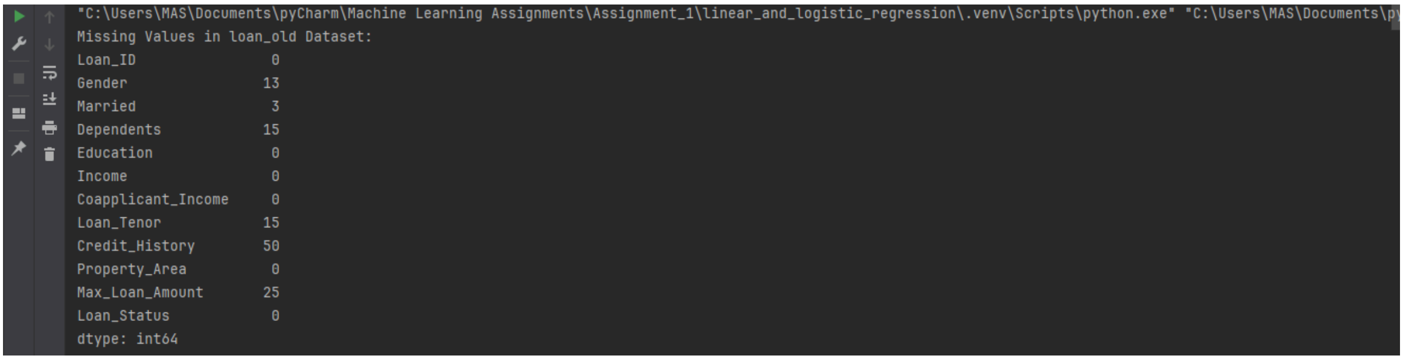
```
# h) Load "loan_new.csv" dataset:
# ----------------------------
loan_new = pd.read_csv("loan_new.csv")
```

➢ b) Perform analysis on the dataset:

- **i) check whether there are missing values in "loan_old.csv" dataset**

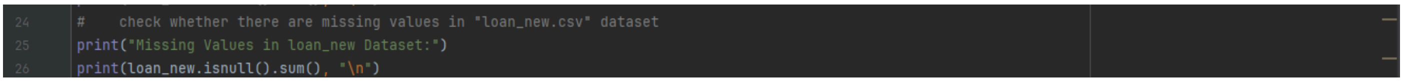


```
# b) Perform analysis on the dataset:
# ---------------------------------
# i) check whether there are missing values in "loan_old.csv" dataset
print("Missing Values in loan_old Dataset:")
print(loan_old.isnull().sum(), "\n")
```

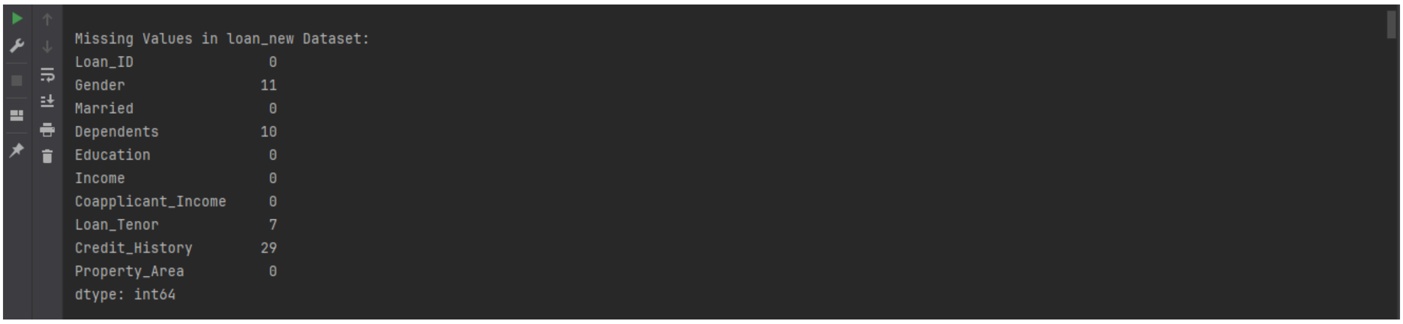


```
"C:\Users\MAS\Documents\pyCharm\Machine Learning Assignments\Assignment_1\linear_and_logistic_regression\.venv\Scripts\python.exe" "C:\Users\MAS\Documents\py
Missing Values in loan_old Dataset:
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Income                0
Coapplicant_Income    0
Loan_Tenor           15
Credit_History       50
Property_Area         0
Max_Loan_Amount      25
Loan_Status           0
dtype: int64
```

- **i) check whether there are missing values in "loan_new.csv" dataset**

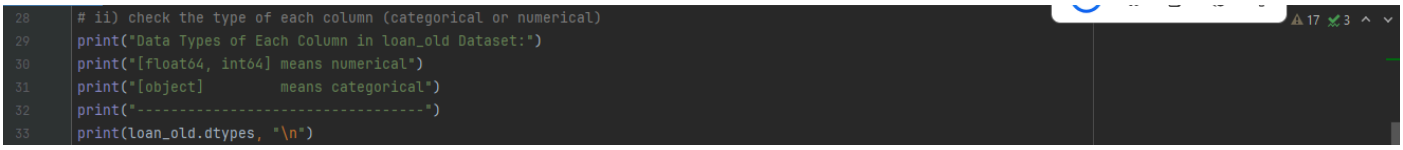


```
#    check whether there are missing values in "loan_new.csv" dataset
print("Missing Values in loan_new Dataset:")
print(loan_new.isnull().sum(), "\n")
```

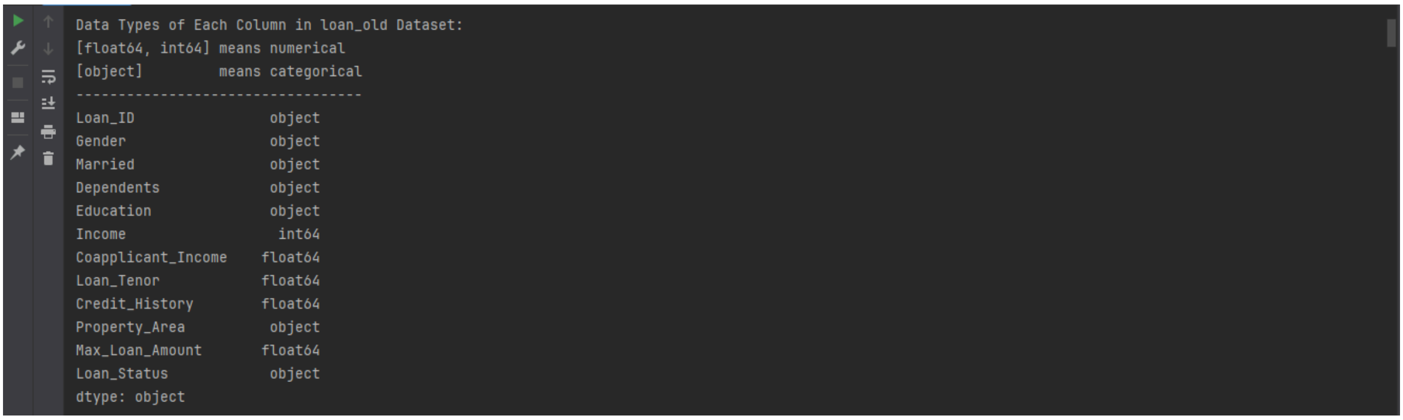


```
Missing Values in loan_new Dataset:
Loan_ID               0
Gender               11
Married               0
Dependents           10
Education             0
Income                0
Coapplicant_Income    0
Loan_Tenor            7
Credit_History       29
Property_Area         0
dtype: int64
```

- **ii) check the type of each column (categorical or numerical)**

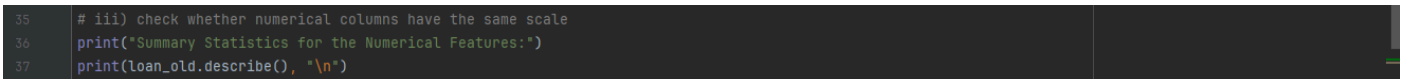


```
# ii) check the type of each column (categorical or numerical)
print("Data Types of Each Column in loan_old Dataset:")
print("[float64, int64] means numerical")
print("[object]         means categorical")
print("---------------------------------")
print(loan_old.dtypes, "\n")
```

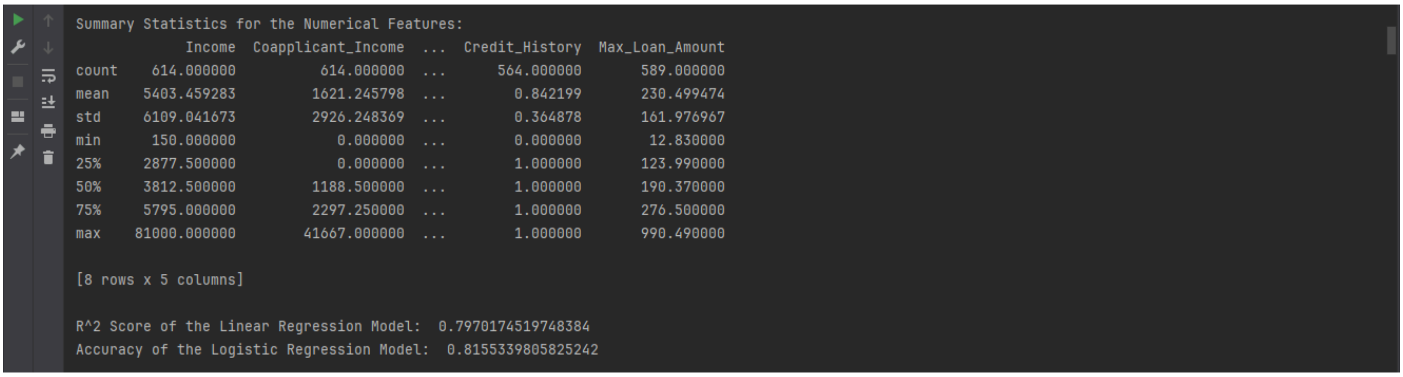


```
Data Types of Each Column in loan_old Dataset:
[float64, int64] means numerical
[object]         means categorical
---------------------------------
Loan_ID                 object
Gender                  object
Married                 object
Dependents              object
Education               object
Income                   int64
Coapplicant_Income     float64
Loan_Tenor             float64
Credit_History         float64
Property_Area           object
Max_Loan_Amount        float64
Loan_Status             object
dtype: object
```

- **iii) check whether numerical columns have the same scale**

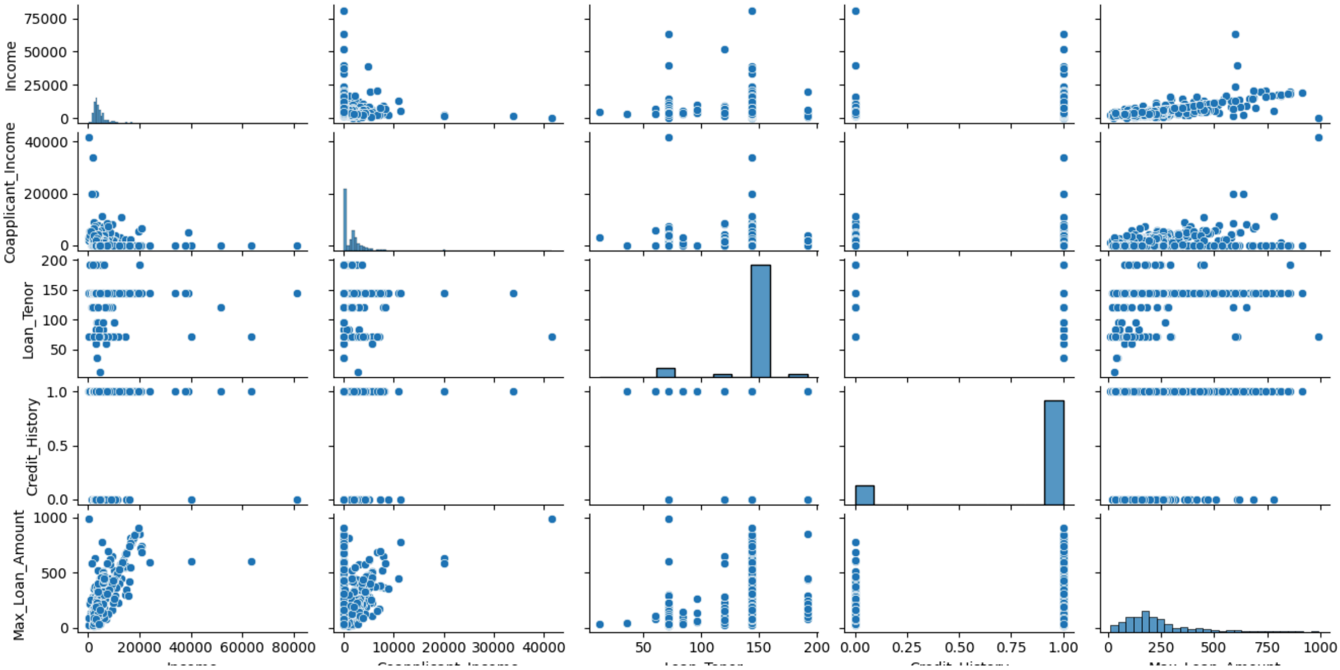


```
# iii) check whether numerical columns have the same scale
print("Summary Statistics for the Numerical Features:")
print(loan_old.describe(), "\n")
```



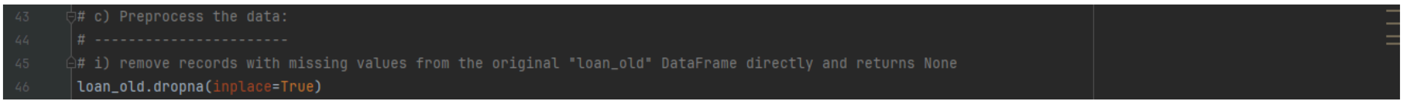
```
Summary Statistics for the Numerical Features:
             Income  Coapplicant_Income  ...  Credit_History  Max_Loan_Amount
count    614.000000          614.000000  ...      564.000000       589.000000
mean    5403.459283         1621.245798  ...        0.842199       230.499474
std     6109.041673         2926.248369  ...        0.364878       161.976967
min      150.000000            0.000000  ...        0.000000        12.830000
25%     2877.500000            0.000000  ...        1.000000       123.990000
50%     3812.500000         1188.500000  ...        1.000000       190.370000
75%     5795.000000         2297.250000  ...        1.000000       276.500000
max    81000.000000        41667.000000  ...        1.000000       990.490000

[8 rows x 5 columns]

R^2 Score of the Linear Regression Model:  0.7970174519748384
Accuracy of the Logistic Regression Model:  0.8155339805825242
```

- **iv) visualize a pairplot between numerical columns**



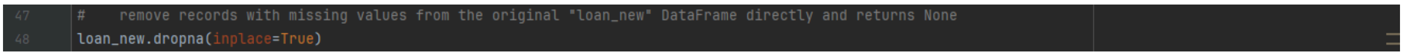
➢ c) Preprocess the data:

- **i) remove records with missing values from the original "loan_old" DataFrame directly and returns None**



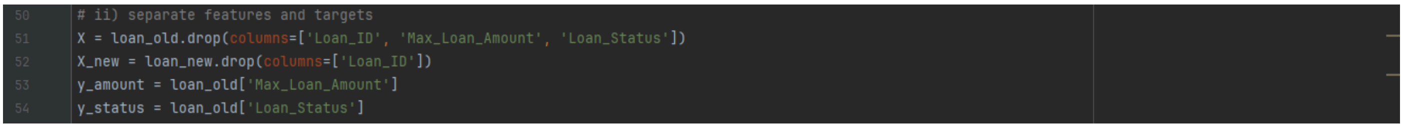
```
# c) Preprocess the data:
# ----------------------
# i) remove records with missing values from the original "loan_old" DataFrame directly and returns None
loan_old.dropna(inplace=True)
```

- **i) remove records with missing values from the original "loan_new" DataFrame directly and returns None**



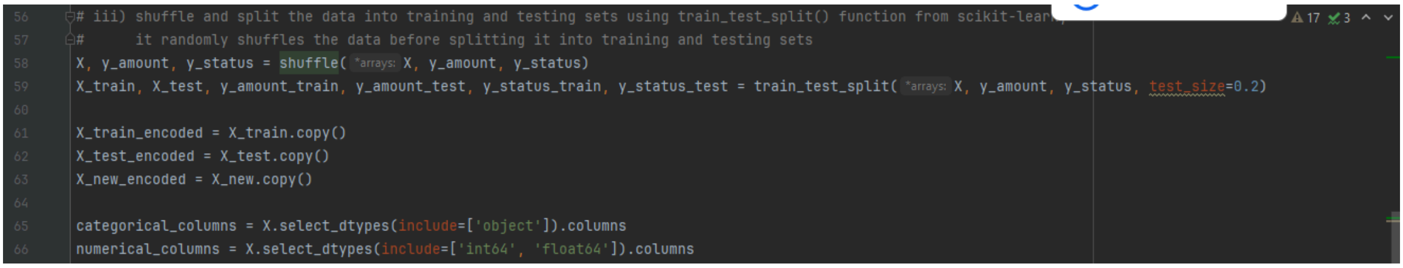
```
#    remove records with missing values from the original "loan_new" DataFrame directly and returns None
loan_new.dropna(inplace=True)
```

- **ii) separate features and targets**



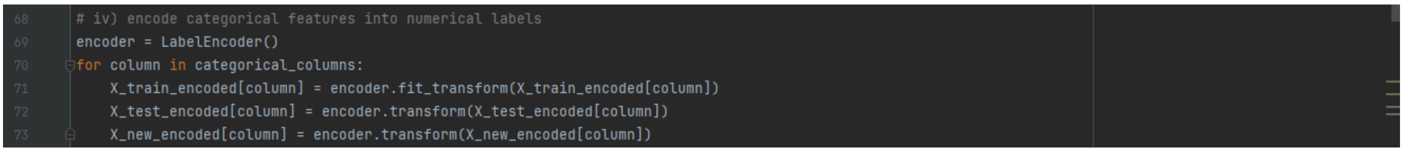
```
# ii) separate features and targets
X = loan_old.drop(columns=['Loan_ID', 'Max_Loan_Amount', 'Loan_Status'])
X_new = loan_new.drop(columns=['Loan_ID'])
y_amount = loan_old['Max_Loan_Amount']
y_status = loan_old['Loan_Status']
```

- **iii) shuffle and split the data into training and testing sets using train_test_split() function from scikit-learn, it randomly shuffles the data before splitting it into training and testing sets**



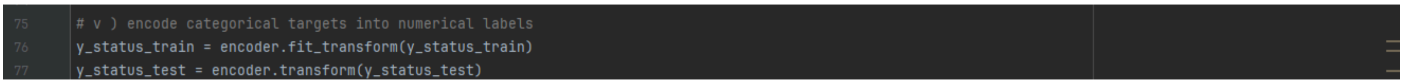
```
# iii) shuffle and split the data into training and testing sets using train_test_split() function from scikit-lear
#      it randomly shuffles the data before splitting it into training and testing sets
X, y_amount, y_status = shuffle( *arrays: X, y_amount, y_status)
X_train, X_test, y_amount_train, y_amount_test, y_status_train, y_status_test = train_test_split( *arrays: X, y_amount, y_status, test_size=0.2)

X_train_encoded = X_train.copy()
X_test_encoded = X_test.copy()
X_new_encoded = X_new.copy()

categorical_columns = X.select_dtypes(include=['object']).columns
numerical_columns = X.select_dtypes(include=['int64', 'float64']).columns
```

- **iv) encode categorical features into numerical labels**



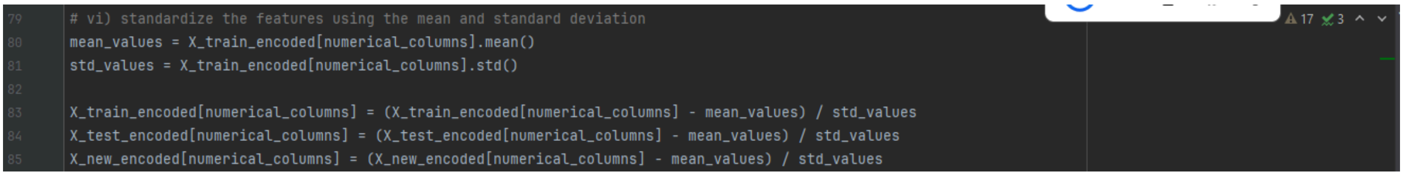
```
# iv) encode categorical features into numerical labels
encoder = LabelEncoder()
for column in categorical_columns:
    X_train_encoded[column] = encoder.fit_transform(X_train_encoded[column])
    X_test_encoded[column] = encoder.transform(X_test_encoded[column])
    X_new_encoded[column] = encoder.transform(X_new_encoded[column])
```

- **v) encode categorical targets into numerical labels**



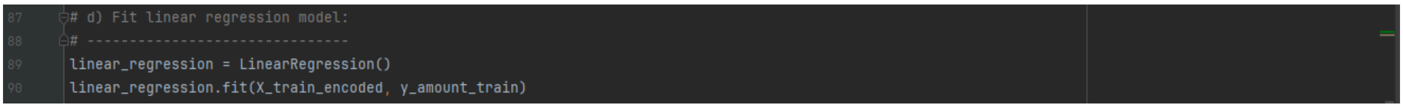
```
# v ) encode categorical targets into numerical labels
y_status_train = encoder.fit_transform(y_status_train)
y_status_test = encoder.transform(y_status_test)
```

- **vi) standardize the features using the mean and standard deviation**



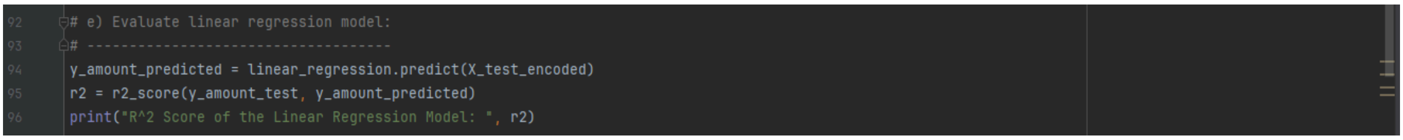
```
# vi) standardize the features using the mean and standard deviation
mean_values = X_train_encoded[numerical_columns].mean()
std_values = X_train_encoded[numerical_columns].std()

X_train_encoded[numerical_columns] = (X_train_encoded[numerical_columns] - mean_values) / std_values
X_test_encoded[numerical_columns] = (X_test_encoded[numerical_columns] - mean_values) / std_values
X_new_encoded[numerical_columns] = (X_new_encoded[numerical_columns] - mean_values) / std_values
```

➢ d) Fit linear regression model:



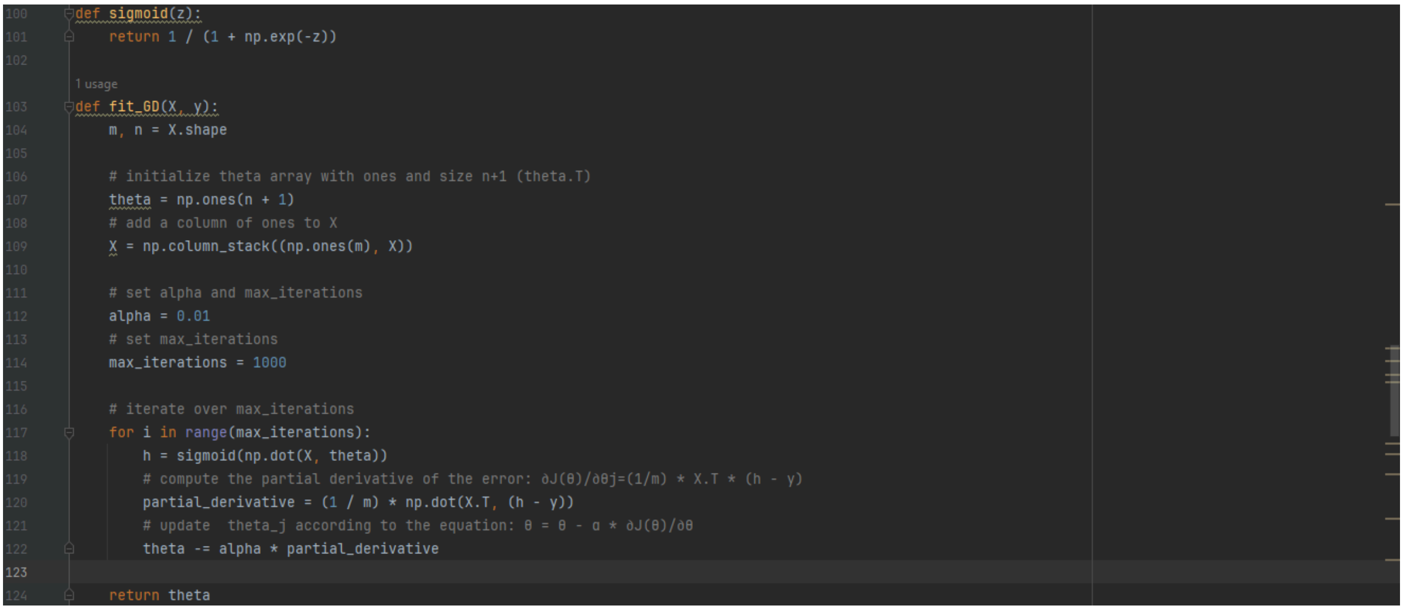
```
# d) Fit linear regression model:
# -------------------------------
linear_regression = LinearRegression()
linear_regression.fit(X_train_encoded, y_amount_train)
```

➢ e) Evaluate linear regression model:

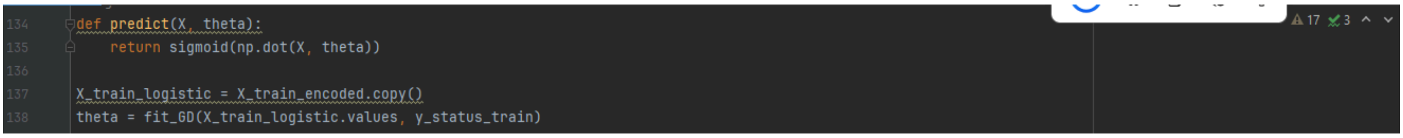


```
# e) Evaluate linear regression model:
# -----------------------------------
y_amount_predicted = linear_regression.predict(X_test_encoded)
r2 = r2_score(y_amount_test, y_amount_predicted)
print("R^2 Score of the Linear Regression Model: ", r2)
```

➢ f) Fit logistic regression model from scratch:

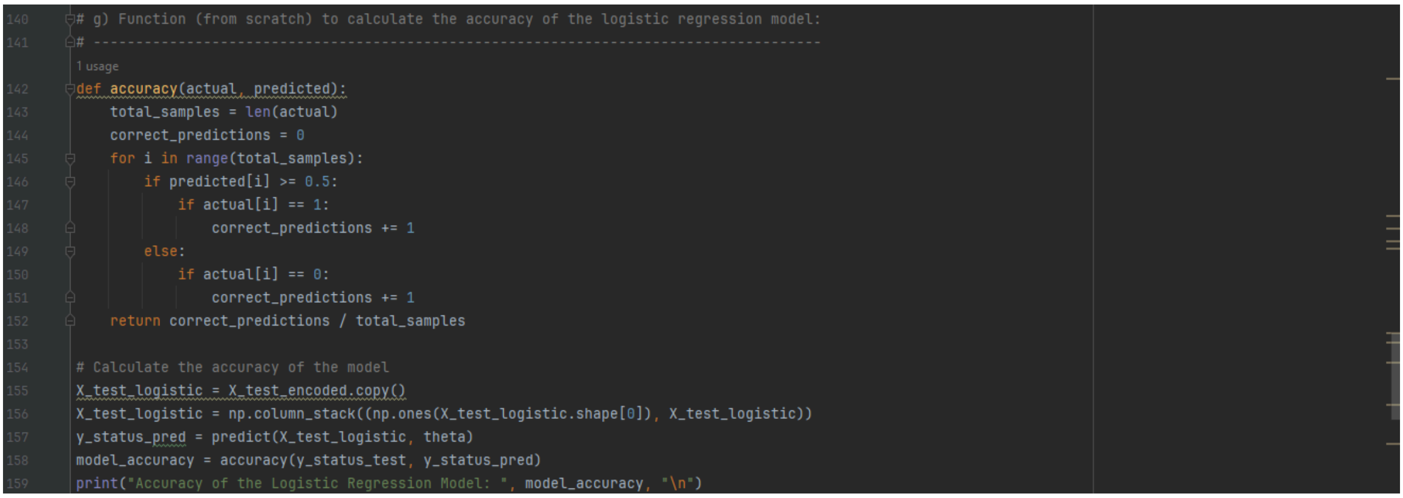


```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

1 usage
def fit_GD(X, y):
    m, n = X.shape

    # initialize theta array with ones and size n+1 (theta.T)
    theta = np.ones(n + 1)
    # add a column of ones to X
    X = np.column_stack((np.ones(m), X))

    # set alpha and max_iterations
    alpha = 0.01
    # set max_iterations
    max_iterations = 1000

    # iterate over max_iterations
    for i in range(max_iterations):
        h = sigmoid(np.dot(X, theta))
        # compute the partial derivative of the error: ∂J(θ)/∂θj=(1/m) * X.T * (h - y)
        partial_derivative = (1 / m) * np.dot(X.T, (h - y))
        # update  theta_j according to the equation: θ = θ - α * ∂J(θ)/∂θ
        theta -= alpha * partial_derivative

    return theta
```



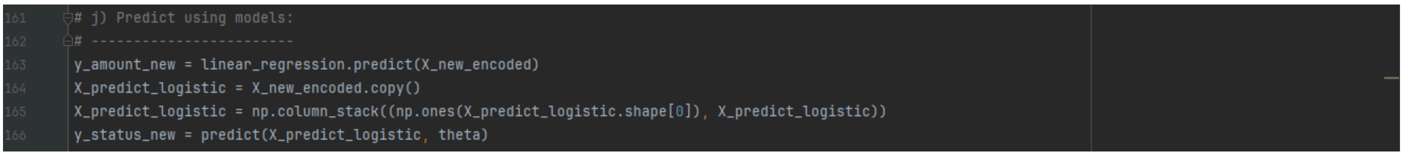
```
def predict(X, theta):
    return sigmoid(np.dot(X, theta))

X_train_logistic = X_train_encoded.copy()
theta = fit_GD(X_train_logistic.values, y_status_train)
```

➢ g) Function (from scratch) to calculate the accuracy of the logistic regression model:



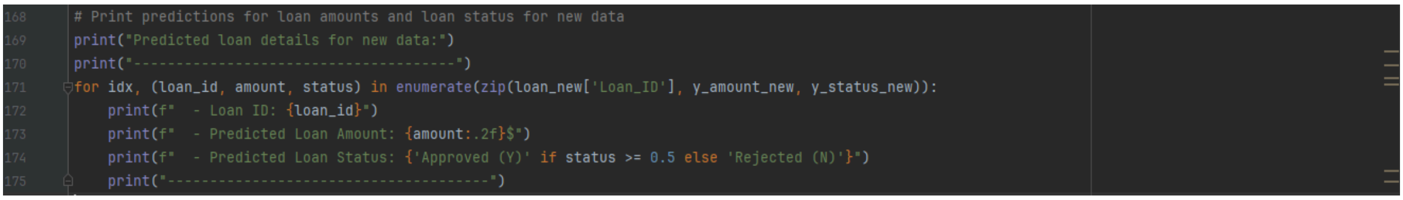
```
# g) Function (from scratch) to calculate the accuracy of the logistic regression model:
# -------------------------------------------------------------------------------------
1 usage
def accuracy(actual, predicted):
    total_samples = len(actual)
    correct_predictions = 0
    for i in range(total_samples):
        if predicted[i] >= 0.5:
            if actual[i] == 1:
                correct_predictions += 1
        else:
            if actual[i] == 0:
                correct_predictions += 1
    return correct_predictions / total_samples

# Calculate the accuracy of the model
X_test_logistic = X_test_encoded.copy()
X_test_logistic = np.column_stack((np.ones(X_test_logistic.shape[0]), X_test_logistic))
y_status_pred = predict(X_test_logistic, theta)
model_accuracy = accuracy(y_status_test, y_status_pred)
print("Accuracy of the Logistic Regression Model: ", model_accuracy, "\n")
```

➢ j) Predict using models:

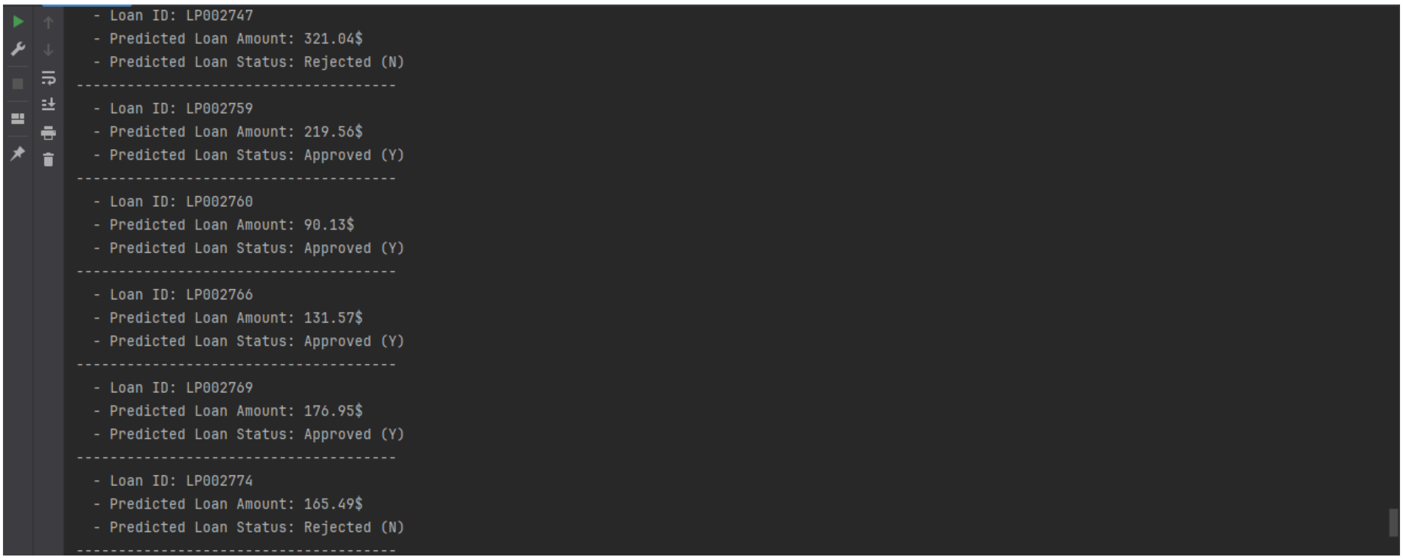


```
# j) Predict using models:
# -----------------------
y_amount_new = linear_regression.predict(X_new_encoded)
X_predict_logistic = X_new_encoded.copy()
X_predict_logistic = np.column_stack((np.ones(X_predict_logistic.shape[0]), X_predict_logistic))
y_status_new = predict(X_predict_logistic, theta)
```

➢ Print predictions for loan amounts and loan status for new data



```
# Print predictions for loan amounts and loan status for new data
print("Predicted loan details for new data:")
print("-------------------------------------")
for idx, (loan_id, amount, status) in enumerate(zip(loan_new['Loan_ID'], y_amount_new, y_status_new)):
    print(f"  - Loan ID: {loan_id}")
    print(f"  - Predicted Loan Amount: {amount:.2f}$")
    print(f"  - Predicted Loan Status: {'Approved (Y)' if status >= 0.5 else 'Rejected (N)'}")
    print("-------------------------------------")
```



```
  - Loan ID: LP002747
  - Predicted Loan Amount: 321.04$
  - Predicted Loan Status: Rejected (N)
-------------------------------------
  - Loan ID: LP002759
  - Predicted Loan Amount: 219.56$
  - Predicted Loan Status: Approved (Y)
-------------------------------------
  - Loan ID: LP002760
  - Predicted Loan Amount: 90.13$
  - Predicted Loan Status: Approved (Y)
-------------------------------------
  - Loan ID: LP002766
  - Predicted Loan Amount: 131.57$
  - Predicted Loan Status: Approved (Y)
-------------------------------------
  - Loan ID: LP002769
  - Predicted Loan Amount: 176.95$
  - Predicted Loan Status: Approved (Y)
-------------------------------------
  - Loan ID: LP002774
  - Predicted Loan Amount: 165.49$
  - Predicted Loan Status: Rejected (N)
-------------------------------------
```