Assignment 1: Linear and Logistic Regression

Names and IDs:

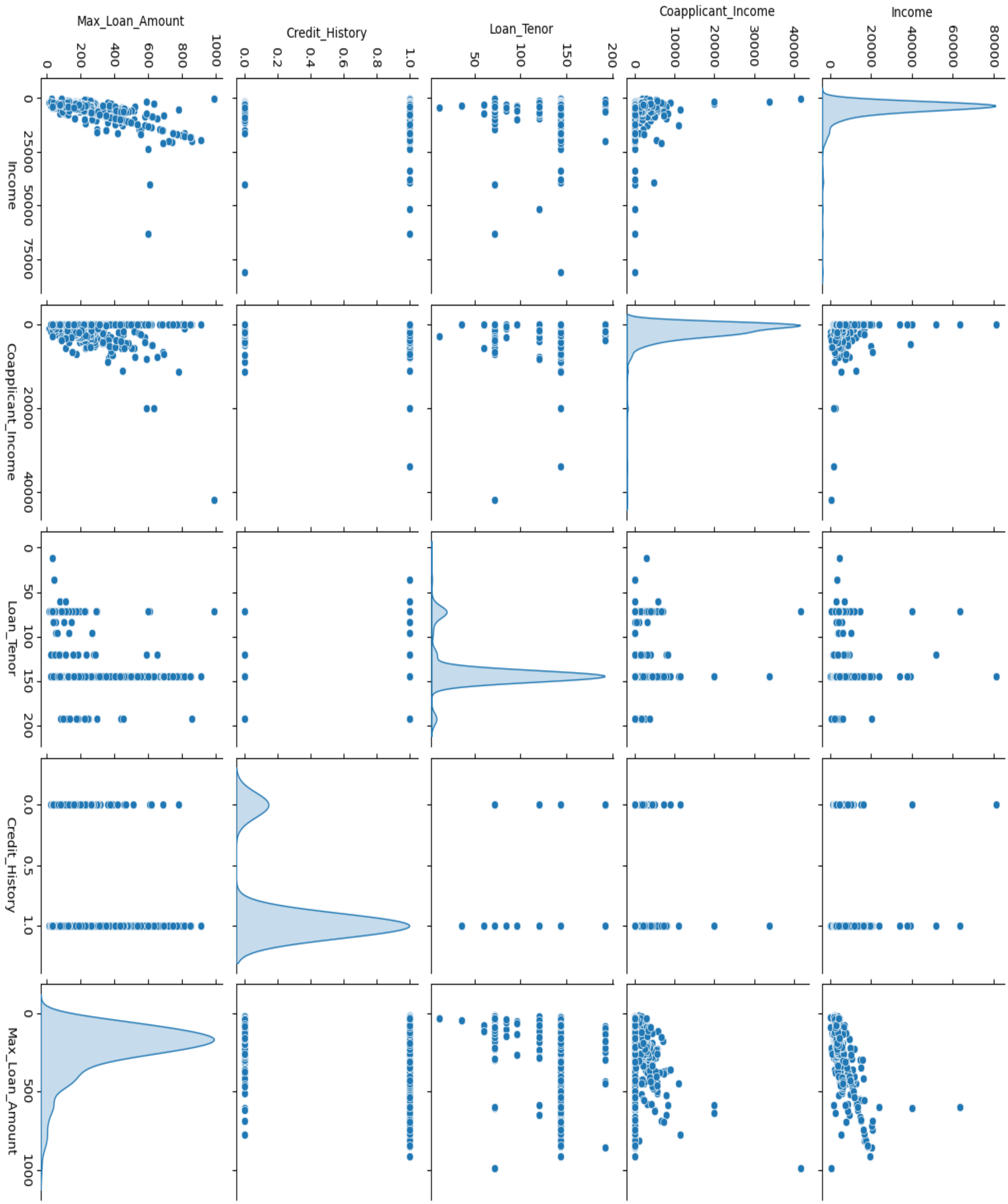
Amr Mohamed	20206049
Maryam Khamis	20206160
Mazen Ayman	20206057
Mariam Abdelazim	20206150

Code with screenshots of the output of each part:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,
LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Load the "loan_old.csv" dataset
loan_old_df = pd.read_csv("loan_old.csv")

# a) Perform analysis on the dataset
# i) Check for missing values
missing_values = loan_old_df.isnull().sum()
print("Missing values:\n", missing_values)

# ii) Check the type of each feature
print("\nData types:")
print(loan_old_df.dtypes)
```

```
Missing values:
 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

Data types:
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
```

```
# iv) Visualize pairplot between numerical columns
# KDE plots give a smoother estimate of the distribution of
each variable
numerical_columns =
loan_old_df.select_dtypes(include=[np.number]).columns
plt.figure(figsize=(10, 6))
sns.pairplot(loan_old_df[numerical_columns],
diag_kind='kde')
plt.show() # to save as png ---> plt.savefig('pairplot.png')
```



Income
Coapplicant_Income
Loan_Tenor
Credit_History
Max_Loan_Amount

```python
# iii) Check whether numerical features have the same scale
print("\nNumerical features scale:")
print(loan_old_df.describe())

# iv) Visualize pairplot between numerical columns
# KDE plots give a smoother estimate of the distribution of each variable
numerical_columns = loan_old_df.select_dtypes(include=[np.number]).columns
plt.figure(figsize=(10, 6))
sns.pairplot(loan_old_df[numerical_columns], diag_kind='kde')
plt.show() # to save as png ---> plt.savefig('pairplot.png')

# c) Preprocess the data
# i) Remove records containing missing values
loan_old_df.dropna(inplace=True)

# ii) Separate features and targets
X = loan_old_df.drop(columns=['Max_Loan_Amount', 'Loan_Status'])
y_amount = loan_old_df['Max_Loan_Amount']
y_status = loan_old_df['Loan_Status']

# iii) Shuffle and split into training and testing sets
# 20% of the data will be used for testing, and the remaining 80% will be used
for training.
X_train, X_test, y_amount_train, y_amount_test, y_status_train, y_status_test = \
    train_test_split(X, y_amount, y_status, test_size=0.2, random_state=40)

# iv) Categorical features encoding
encoder = LabelEncoder()

# Concatenate training and test sets for encoding
X_combined = pd.concat([X_train, X_test], axis=0)
X_combined_encoded = X_combined.apply(encoder.fit_transform)

# Split back into training and test sets
X_train_encoded = X_combined_encoded[:len(X_train)]
X_test_encoded = X_combined_encoded[len(X_train):]

# Concatenate encoded features with target variables
X_train_encoded = pd.concat([X_train_encoded, y_amount_train, y_status_train],
axis=1)

# v) Numerical features standardization
scaler = StandardScaler()
```

```python
# Fit the scaler on the training data excluding Max_Loan_Amount and Loan_Status
columns
X_train_scaled =
scaler.fit_transform(X_train_encoded.drop(columns=['Max_Loan_Amount',
'Loan_Status']))
X_test_scaled = scaler.transform(X_test_encoded)

# d) Fit a linear regression model
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train_scaled, y_amount_train)

# e) Evaluate the linear regression model
y_amount_pred = linear_reg_model.predict(X_test_scaled)
r2 = r2_score(y_amount_test, y_amount_pred)
print("\nLinear Regression R2 Score:", r2)

# f) Fit a logistic regression model (from scratch)
class LogisticRegressionGD:
    def __init__(self, eta=0.01, n_iter=1000, random_state=40):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        np.random.seed(self.random_state)
        self.weights = np.random.rand(X.shape[1])
        self.costs = []

        for _ in range(self.n_iter):
            net_input = self.net_input(X)
            output = self.activation(net_input)
            errors = y - output
            self.weights += self.eta * X.T.dot(errors)
            cost = self.loss(output, y)
            self.costs.append(cost)

    def net_input(self, X):
        return np.dot(X, self.weights)

    def activation(self, z):
        return 1 / (1 + np.exp(-np.clip(z, -250, 250)))

    def loss(self, output, y):
        return -y.dot(np.log(output)) - ((1 - y).dot(np.log(1 - output)))
w
```

```
    def predict(self, X):
        return np.where(self.activation(self.net_input(X)) >= 0.5, 1, 0)

# g) Calculate the accuracy of the model (from scratch)
def accuracy(y_true, y_pred):
    return np.mean(y_true == y_pred)

# Encode the target variables for logistic regression
y_status_encoder = LabelEncoder()
y_status_train_encoded = y_status_encoder.fit_transform(y_status_train)
y_status_test_encoded = y_status_encoder.transform(y_status_test)

# Fit the logistic regression model
logistic_reg_model = LogisticRegressionGD()
logistic_reg_model.fit(X_train_scaled, y_status_train_encoded)

# Calculate the accuracy of the model
y_status_pred = logistic_reg_model.predict(X_test_scaled)
accuracy_scratch = accuracy(y_status_test_encoded, y_status_pred)
print("Logistic Regression Accuracy (from scratch):", accuracy_scratch)
```

```
Numerical features scale:
              Income  Coapplicant_Income  Loan_Tenor  Credit_History  Max_Loan_Amount
count     614.000000          614.000000  599.000000      564.000000       589.000000
mean     5403.459283         1621.245798  137.689482        0.842199       230.499474
std      6109.041673         2926.248369   23.366294        0.364878       161.976967
min       150.000000            0.000000   12.000000        0.000000        12.830000
25%      2877.500000            0.000000  144.000000        1.000000       123.990000
50%      3812.500000         1188.500000  144.000000        1.000000       190.370000
75%      5795.000000         2297.250000  144.000000        1.000000       276.500000
max     81000.000000        41667.000000  192.000000        1.000000       990.490000

Linear Regression R2 Score: 0.6682035132047686
Logistic Regression Accuracy (from scratch): 0.8349514563106796
```

```python
# h) Load the "loan_new.csv" dataset
loan_new_df = pd.read_csv("loan_new.csv")

# i) Perform the same preprocessing on it (except shuffling and splitting)

# 1) Separate features
X_new = loan_new_df.copy()

# 2) Categorical features encoding using the same encoder trained on the old dataset
X_new_encoded = X_new.apply(lambda x: encoder.transform(x) if x.name in encoder.classes_ else x)

# 3) Numerical features standardization
X_new_scaled = scaler.transform(X_new_encoded)

# j) Use your models on this data to predict the Max_Loan_Amount and Loan_Status

# Predict Max_Loan_Amount using linear regression model
max_loan_amount_pred = linear_reg_model.predict(X_new_scaled)

# Predict Loan_Status using logistic regression model (from scratch)
loan_status_pred = logistic_reg_model.predict(X_new_scaled)

# Convert encoded Loan_Status back to original labels
loan_status_pred_labels = y_status_encoder.inverse_transform(loan_status_pred)

# Output predictions
predictions_df = pd.DataFrame({
    'Max_Loan_Amount_Predicted': max_loan_amount_pred,
    'Loan_Status_Predicted': loan_status_pred_labels
})

print("Predictions for the loan_new dataset:")
print(predictions_df)
```