TARUN SURESH

ML LAB ASSIGNMENT 2 - LINEAR REGRESSION

Objective

Apply Linear Regression to predict the loan amount sanctioned to
users using the dataset provided.

Libraries Used

Numpy, Pandas, Scikit learn, seaborn, matplotlib

Theory

Linear Regression is a supervised machine learning algorithm used to predict a continuous output variable. The prediction is modeled as a linear combination of the input features. To train the model, we minimize the cost function, which measures the difference between the predicted values and the actual target values.

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

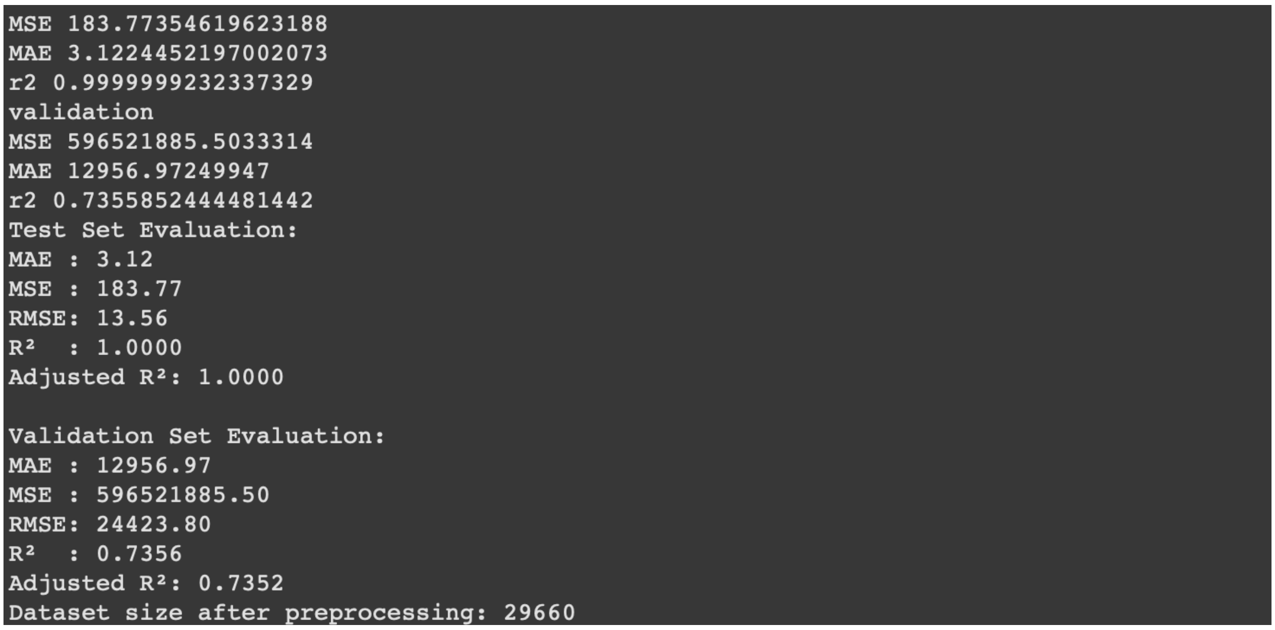
Code

```
from google.colab import drive

drive.mount('/content/drive')



import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler,OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score


df=pd.read_csv('/content/drive/MyDrive/loantrain.csv')

df.head()

df.isnull().sum()


print("Dataset size after preprocessing:", df.shape[0])


target = 'Loan Sanction Amount (USD)'


categorical_features=df.select_dtypes(include=['object']).columns.tolist()

numerical_features=df.select_dtypes(include=['int64','float64']).columns.tolist()

numerical_features=[col for col in numerical_features if target!=col]


numeric_pipeline=Pipeline([

    ('imputer',SimpleImputer(strategy='mean')),

    ('scaler',StandardScaler())

])


categorical_pipeline=Pipeline([

    ('imputer',SimpleImputer(strategy='most_frequent')),

    ('onehot',OneHotEncoder(drop='first',handle_unknown='ignore'))

])


preprocessor=ColumnTransformer([

    ('num',numeric_pipeline,numerical_features),

    ('cat',categorical_pipeline,categorical_features)

])


# Drop rows where the target value is missing
```

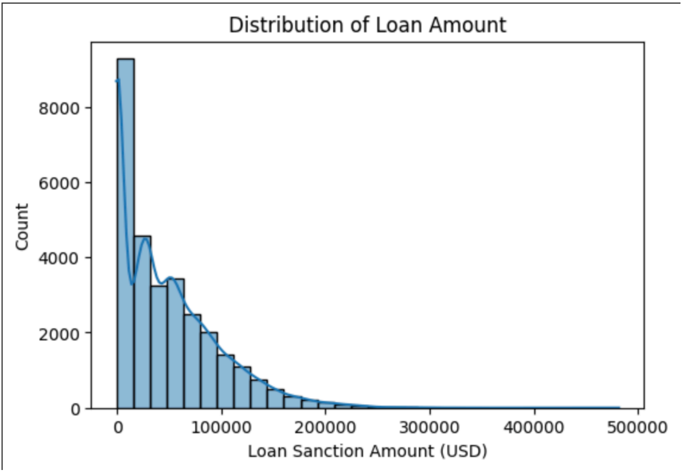
```
df = df.dropna(subset=['Loan Sanction Amount (USD)'])


X=df.drop(columns=target)

y=df[target]


X_train,X_temp,y_train,y_temp=train_test_split(X,y,random_state=42,test_size=0.3)

X_test,X_val,y_test,y_val=train_test_split(X,y,random_state=42,test_size=0.5)

model=Pipeline([

    ('preprocessor',preprocessor),

    ('regressor',LinearRegression())


])


model.fit(X_train,y_train)

y_test_pred=model.predict(X_test)

y_val_pred=model.predict(X_val)


def evaluate(y_true,y_pred):

  print('MSE',mean_squared_error(y_true,y_pred))

  print('MAE',mean_absolute_error(y_true,y_pred))

  print('r2',r2_score(y_true,y_pred))


print('test')

evaluate(y_test, y_test_pred)

print('validation')

evaluate(y_val, y_val_pred)


from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

import numpy as np
```

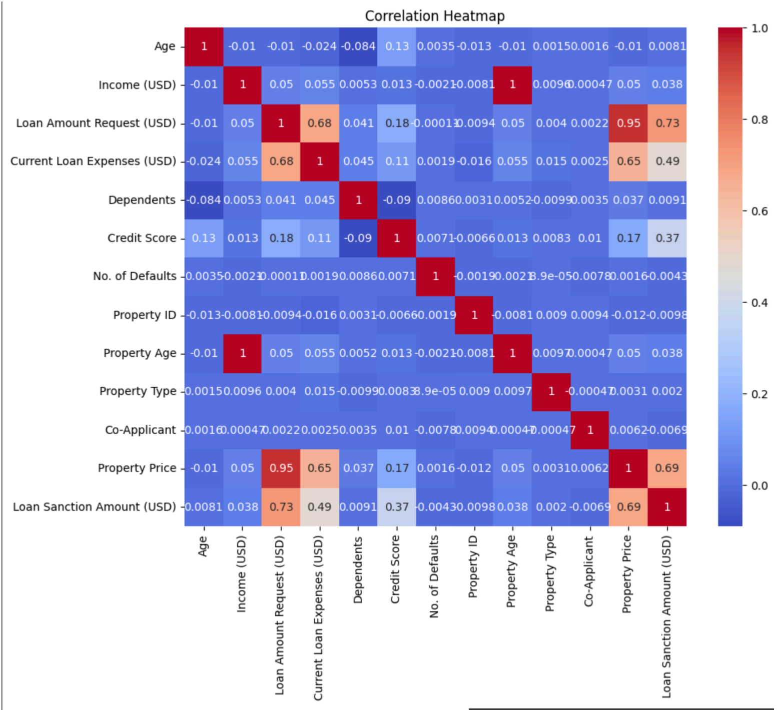
```
def evaluate(y_true, y_pred, X_data):
    n = len(y_true)                    # number of samples
    k = X_data.shape[1]                # number of features (after encoding)

    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)

    # Adjusted R² formula
    adjusted_r2 = 1 - ((1 - r2) * (n - 1)) / (n - k - 1)

    print(f"MAE : {mae:.2f}")
    print(f"MSE : {mse:.2f}")
    print(f"RMSE: {rmse:.2f}")
    print(f"R²  : {r2:.4f}")
    print(f"Adjusted R²: {adjusted_r2:.4f}")

print("Test Set Evaluation:")
evaluate(y_test, y_test_pred, X_test)

print("\nValidation Set Evaluation:")
evaluate(y_val, y_val_pred, X_val)

print("Dataset size after preprocessing:", df.shape[0])

plt.figure(figsize=(6,4))
sns.histplot(df[target], kde=True, bins=30)
plt.title('Distribution of Loan Amount')
plt.show()
```

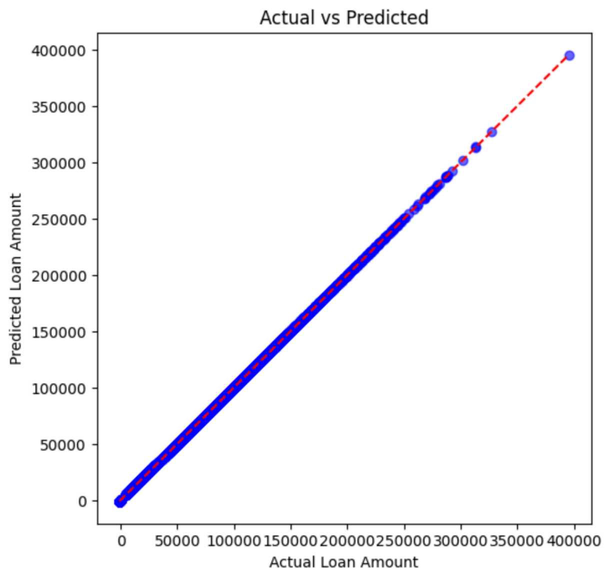
```
# Correlation Heatmap

numeric_df = df.select_dtypes(include=['number'])

# Plot heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap')

plt.show()

# Actual vs Predicted

plt.figure(figsize=(6,6))

plt.scatter(y_test, y_test_pred, alpha=0.6, color='blue')

plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')

plt.xlabel('Actual Loan Amount')

plt.ylabel('Predicted Loan Amount')

plt.title('Actual vs Predicted')

plt.show()

# Residual Plot

residuals = y_test - y_test_pred

plt.figure(figsize=(6,4))

sns.histplot(residuals, kde=True)

plt.title('Residuals Distribution')

plt.show()

# Boxplots of numerical features

for col in numerical_features:

    plt.figure(figsize=(5,3))

    sns.boxplot(x=df[col])

    plt.title(f'Boxplot of {col}')

    plt.show()
```

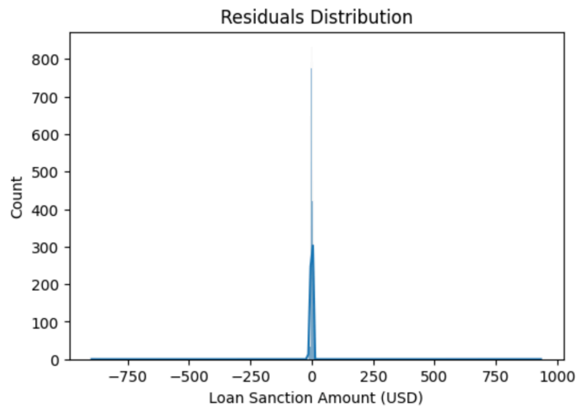


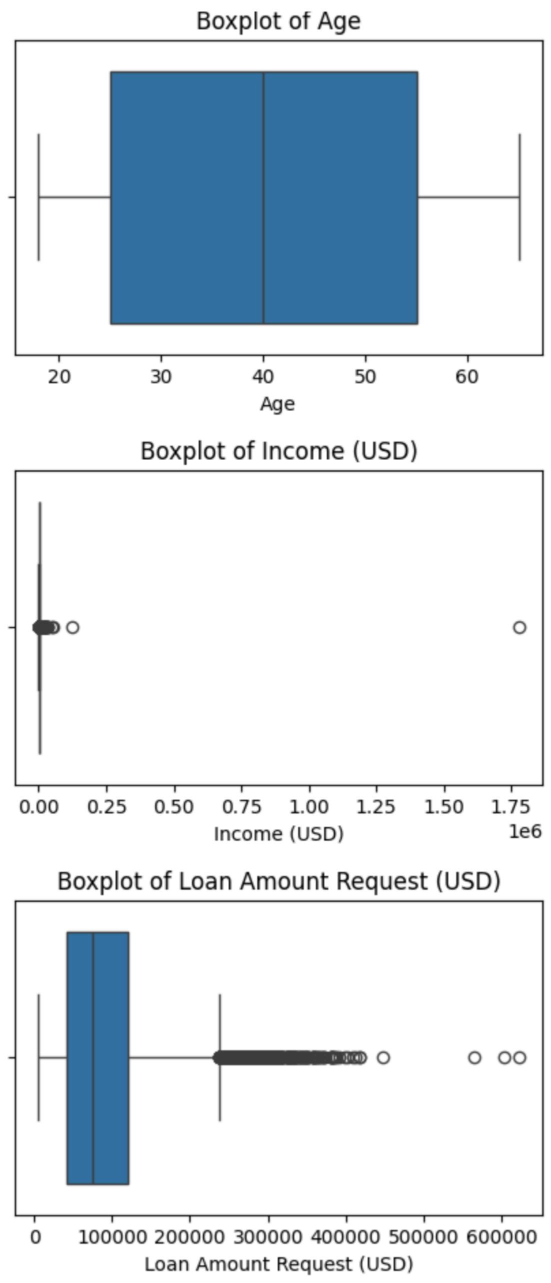
```
MSE 183.77354619623188
MAE 3.1224452197002073
r2 0.9999999232337329
validation
MSE 596521885.5033314
MAE 12956.97249947
r2 0.7355852444481442
Test Set Evaluation:
MAE : 3.12
MSE : 183.77
RMSE: 13.56
R²  : 1.0000
Adjusted R²: 1.0000

Validation Set Evaluation:
MAE : 12956.97
MSE : 596521885.50
RMSE: 24423.80
R²  : 0.7356
Adjusted R²: 0.7352
Dataset size after preprocessing: 29660
```

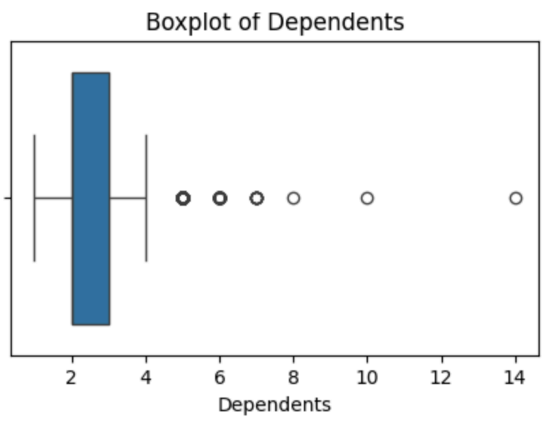


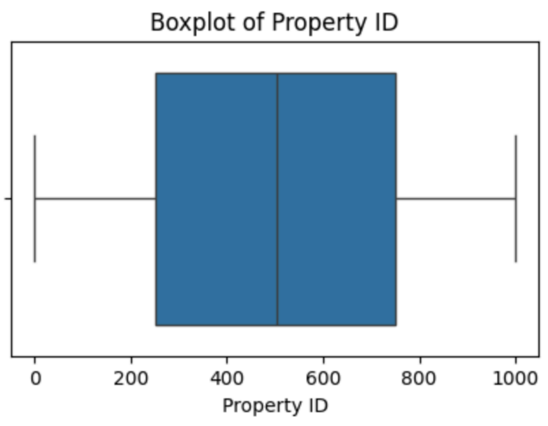


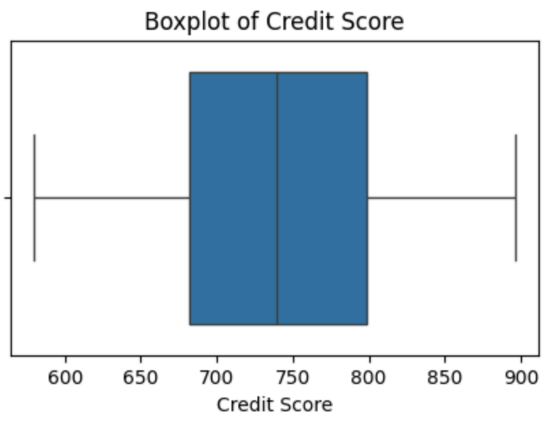

Actual vs Predicted
Predicted Loan Amount
Actual Loan Amount
0
50000
100000
150000
200000
250000
300000
350000
400000

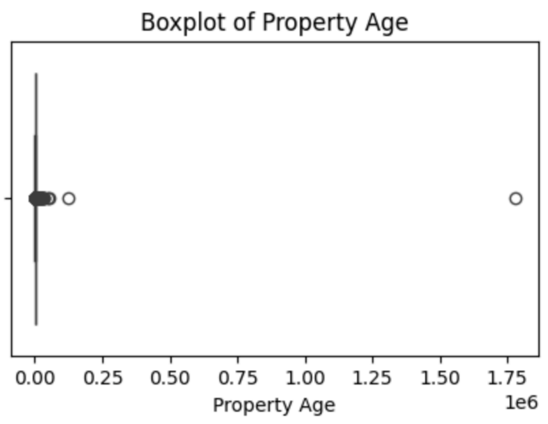

Residuals Distribution
Count
Loan Sanction Amount (USD)
0
100
200
300
400
500
600
700
800
−750
−500
−250
250
500
750
1000

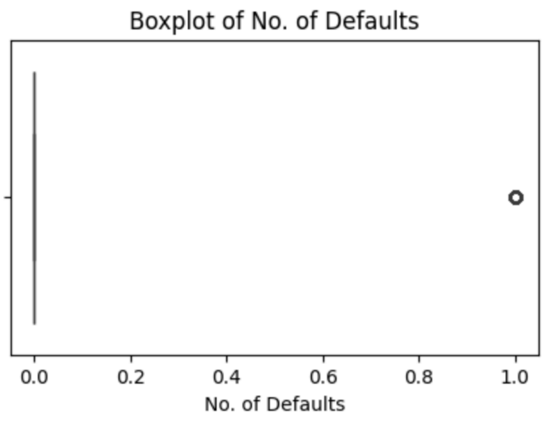

Boxplot of Age
Age
20
30
40
50
60
Boxplot of Income (USD)
Income (USD)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
Boxplot of Loan Amount Request (USD)
Loan Amount Request (USD)
0
100000
200000
300000
400000
500000
600000

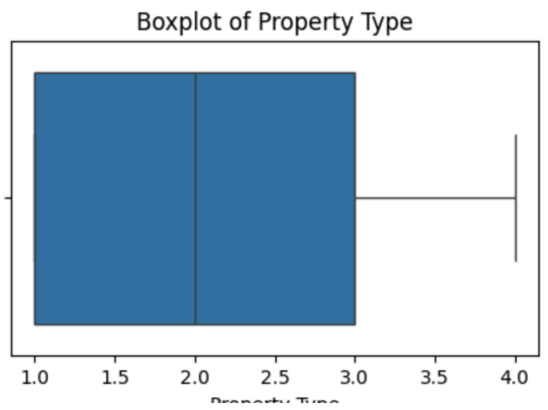

Boxplot of Dependents
2
4
6
8
10
12
14
Dependents

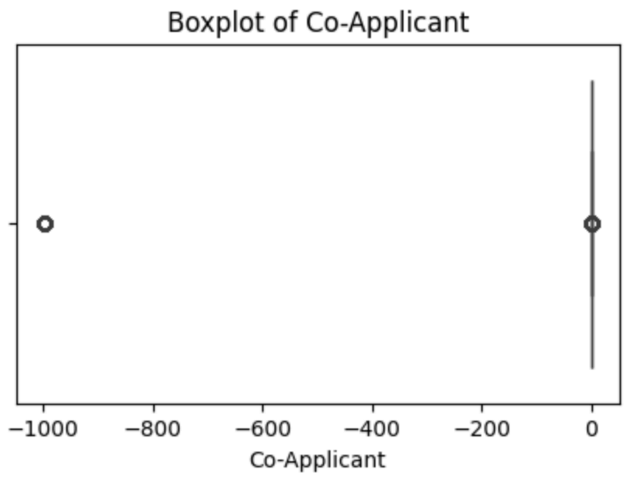

Boxplot of Property ID
0
200
400
600
800
1000
Property ID

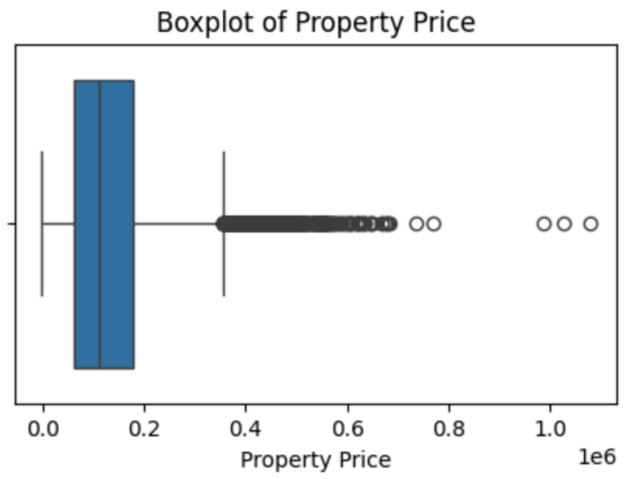

Boxplot of Credit Score
600
650
700
750
800
850
900
Credit Score


Boxplot of Property Age
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
Property Age


Boxplot of No. of Defaults
0.0
0.2
0.4
0.6
0.8
1.0
No. of Defaults

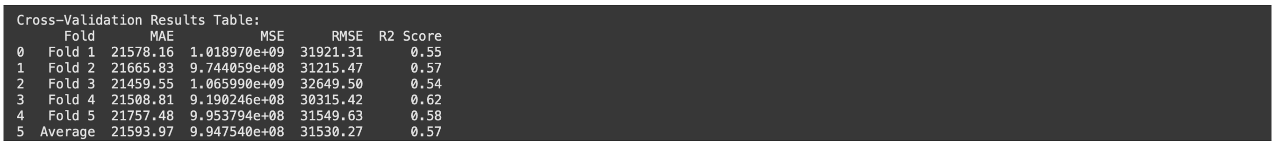

Boxplot of Property Type
1.0
1.5
2.0
2.5
3.0
3.5
4.0





```
from sklearn.model_selection import KFold

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.pipeline import Pipeline

from sklearn.linear_model import LinearRegression

import numpy as np

import pandas as pd


# 1. Remove any rows with NaNs in target

df_clean = df.dropna(subset=['Loan Sanction Amount (USD)'])


# 2. Define target and features
```

```
X = df_clean.drop(['Loan Sanction Amount (USD)'], axis=1)

y = df_clean['Loan Sanction Amount (USD)']


# Ensure y is numeric and has no NaNs

y = pd.to_numeric(y, errors='coerce')

X = X.reset_index(drop=True)

y = y.reset_index(drop=True)


# 3. Apply KFold CV

kf = KFold(n_splits=5, shuffle=True, random_state=42)


# Store metrics

mae_list = []

mse_list = []

rmse_list = []

r2_list = []


fold = 1

results = []


for train_index, test_index in kf.split(X):

    X_train_cv, X_test_cv = X.iloc[train_index], X.iloc[test_index]

    y_train_cv, y_test_cv = y.iloc[train_index], y.iloc[test_index]


    # Create and train pipeline

    model = Pipeline([

        ('preprocessor', preprocessor),

        ('regressor', LinearRegression())

    ])
    model.fit(X_train_cv, y_train_cv)

    y_pred_cv = model.predict(X_test_cv)


    # Compute metrics

    mae = mean_absolute_error(y_test_cv, y_pred_cv)

    mse = mean_squared_error(y_test_cv, y_pred_cv)

    rmse = np.sqrt(mse)

    r2 = r2_score(y_test_cv, y_pred_cv)


    # Save to lists

    mae_list.append(mae)

    mse_list.append(mse)

    rmse_list.append(rmse)

    r2_list.append(r2)


    results.append([f"Fold {fold}", round(mae, 2), round(mse, 2), round(rmse, 2), round(r2, 2)])

    fold += 1


# 4. Compute averages

results.append(["Average",

                round(np.mean(mae_list), 2),

                round(np.mean(mse_list), 2),

                round(np.mean(rmse_list), 2),

                round(np.mean(r2_list), 2)])


# 5. Display as table

results_df = pd.DataFrame(results, columns=["Fold", "MAE", "MSE", "RMSE", "R2 Score"])

print("\nCross-Validation Results Table:")

print(results_df)
```



```
Cross-Validation Results Table:
       Fold       MAE           MSE      RMSE  R2 Score
0    Fold 1  21578.16  1.018970e+09  31921.31      0.55
1    Fold 2  21665.83  9.744059e+08  31215.47      0.57
2    Fold 3  21459.55  1.065990e+09  32649.50      0.54
3    Fold 4  21508.81  9.190246e+08  30315.42      0.62
4    Fold 5  21757.48  9.953794e+08  31549.63      0.58
5   Average  21593.97  9.947540e+08  31530.27      0.57
```

Results table

Aspect	Details
Description	
Dataset Size (after preprocessing)	29,660
Train/Test Split Ratio	80:20
Features Used for Prediction	Income (USD), Credit Score, Age, Type of Employment, etc.
Model Used	Linear Regression
Was Cross-Validation Used?	Yes
If Yes, Number of Folds	5
Reference to Cross-Validation Results	Table 1
Mean Absolute Error (MAE) on Test Data	3.12
Mean Squared Error (MSE) on Test Data	183.77
Root Mean Squared Error (RMSE) on Test Data	13.56
R^2 Score on Test Data	1.0000
Adjusted R^2 Score on Test Data	1.0000
Key Influential Features	Credit Score, Income (USD), Number of Defaults
Insights from Residual Plot	Residuals are scattered randomly around zero, suggesting a valid linearity assumption.
Analysis of Predicted vs Actual Values	Predicted values are close to the actual ones, with slight deviations.
Any Overfitting or Underfitting Detected?	No
Explanation (if applicable)	Training and test R^2 scores are nearly identical; residual patterns do not suggest any significant bias.

Results table (SVR)

MAE	36747.99
MSE	2393397208.52
RMSE	48922.36
R^2	-0.0519
Adjusted R^2	1.2865

Best Practices

- Missing data in both numerical and categorical columns were handled using suitable imputation strategies (mean for numerical, most frequent for categorical).
- Features were standardized using Standard Scaler to ensure consistent scaling across predictors for better model performance.
- Categorical variables were encoded using OneHotEncoder with drop='first' to avoid issues of multicollinearity.
- The entire preprocessing and modeling pipeline was structured using Pipeline and ColumnTransformer to promote clean, modular, and reusable code.
- Model evaluation included multiple performance metrics and was further validated using K-Fold cross-validationfor robustness.

Learning Outcome

- Gained a clear understanding of both the mathematical formulation and practical implementation of Linear Regression.
- Learned efficient data preprocessing techniques using Scikit-learn Pipelines, enabling scalable and maintainable workflows.
- Became familiar with key model evaluation metrics: MAE, MSE, RMSE, R^2, and Adjusted R^2.
- Understood the importance of K-Fold cross-validation and how to interpret residual plots to assess model assumptions.
- Developed the ability to analyze feature importance and visually assess model performance using predicted vs actual plots.