Sri Sivasubramaniya Nadar College of Engineering, Chennai
(An autonomous Institution affiliated to Anna University)

Degree & Branch	B.E. Computer Science & Engineering	Semester	V
Subject Code & Name	ICS1512 & Machine Learning Algorithms Laboratory		
Academic year	2025-2026 (Odd)	Batch:2023-2028	**Due date:**

Experiment 2 : Loan Amount Prediction using Linear Regression

Git Hub:https://github.com/Vignesh-0013/Machine_Learning

1 Aim:

To develop and evaluate a machine learning model using Python to predict loan amounts based on applicant financial and credit-related features, utilizing Linear Regression and K-Fold Cross-Validation for performance assessment.

2 Libraries used:

- Numpy
- Pandas
- Matplotlib
- Scikit-learn
- Seaborn

3 Objective:

- To preprocess the loan dataset by handling missing values, encoding categorical variables, selecting relevant features, and splitting the data for training, validation, and testing purposes.
- To build and evaluate a Linear Regression model using K-Fold Cross-Validation, analyze performance metrics such as MSE, RMSE, MAE, and R^2, and interpret the results through residual and prediction plots to assess model effectiveness.

4 Mathematical/theoretical description of the algorithm/objective performed:

4.1 Handling Missing Values

Missing values can negatively impact the performance of machine learning models by:

- Distorting statistical summaries
- Causing errors in training algorithms
- Leading to biased predictions

So, it's crucial to detect and properly handle them before modeling.

There are several ways to handle missing values:

- Missing values in a dataset can be handled using imputation techniques such as replacing them with the mean, median, or mode of the respective feature by using fillna() method in pandas
- If a column contains a large number of missing values and does not contribute significantly to the prediction task, it can be dropped to simplify the dataset. Removing such irrelevant or incomplete features helps reduce noise and improve model efficiency.

4.2 Label encoding:

To train machine learning models, all input features must be in numeric format. Hence, categorical variables (like "Yes"/"No", "Graduate"/"Not Graduate") need to be converted into numbers. This process is essential for enabling algorithms to interpret and process the data correctly.

- Categorical values can be directly replaced with numeric codes, such as mapping "Yes" to 1 and "No" to 0. This is useful when the categories are binary or have no specific order. It ensures compatibility with machine learning models that require numerical input.
- If a categorical feature has more than two values, simple replacement may introduce unintended ordinal relationships. In such cases, **one-hot encoding** is preferred, where each category becomes a separate binary column. This prevents the model from assuming any order or ranking between the categories.

4.3 Plotting:

To better understand the patterns and relationships within the dataset, various data visualization techniques are used. These plots help in identifying correlations, distributions, and outliers effectively.

- The **heatmap()** function visualizes the correlation between numerical features using a colored matrix. Darker shades typically represent stronger correlations, helping identify redundant or related features. It's a useful tool for understanding feature interdependencies before model building.

- A **histogram** displays the frequency distribution of a numeric variable, showing how data is spread across ranges. It helps detect skewness, modality, and presence of outliers or missing value gaps. This is often used as a first step in understanding individual feature behavior.

- A **box plot** (or whisker plot) shows the spread and central tendency of a feature using quartiles. It clearly identifies the median, interquartile range (IQR), and outliers. This makes it an excellent tool for spotting extreme values and data symmetry.

4.4 Standardization:

- Standardization is a feature scaling technique that **transforms data to have a mean of 0 and a standard deviation of 1**. It is especially useful when features have different units or scales, ensuring all variables contribute equally to the model. Many machine learning algorithms, like logistic regression and KNN, perform better when data is standardized.

- The formula used for standardization is:

$$z = \frac{x-\mu}{\sigma}$$

- where x is the original value, μ is the mean, and σ is the standard deviation. This process centers the data around zero and makes it easier for models to converge efficiently.

The preprocessing steps involved handling missing values, encoding categorical variables, visualizing data using heatmaps, histograms, and boxplots, and addressing outliers. Additionally, feature standardization was applied to bring all variables to a common scale, ensuring better model performance and stability.

5 Code:

loan_amount_prediction

July 29, 2025

[2]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```python
df=pd.read_csv('train.csv')
df.head()
```

[3]:
```
  Customer ID                Name Gender  Age  Income (USD) Income Stability  \
0     C-36995   Frederica Shealy      F   56       1933.05              Low
1     C-33999  America Calderone      M   32       4952.91              Low
2      C-3770      Rosetta Verne      F   65        988.19             High
3     C-26480         Zoe Chitty      F   65           NaN             High
4     C-23459      Afton Venema       F   31       2614.77              Low

  Profession    Type of Employment   Location  Loan Amount Request (USD)  \
0    Working           Sales staff  Semi-Urban                   72809.58
1    Working                   NaN  Semi-Urban                   46837.47
2  Pensioner                   NaN  Semi-Urban                   45593.04
3  Pensioner                   NaN       Rural                   80057.92
4    Working  High skill tech staff  Semi-Urban                  113858.89

  ... Credit Score No. of Defaults Has Active Credit Card  Property ID  \
0 ...       809.44               0                    NaN          746
1 ...       780.40               0            Unpossessed          608
2 ...       833.15               0            Unpossessed          546
3 ...       832.70               1            Unpossessed          890
4 ...       745.55               1                 Active          715

  Property Age  Property Type Property Location  Co-Applicant  \
0      1933.05              4             Rural             1
1      4952.91              2             Rural             1
2       988.19              2             Urban             0
3          NaN              2        Semi-Urban             1
4      2614.77              4        Semi-Urban             1

  Property Price  Loan Sanction Amount (USD)
0      119933.46                    54607.18
```

```
1         54791.00                        37469.98
2         72440.58                        36474.43
3        121441.51                        56040.54
4        208567.91                        74008.28

[5 rows x 24 columns]
```

[4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Customer ID                 30000 non-null  object
 1   Name                        30000 non-null  object
 2   Gender                      29947 non-null  object
 3   Age                         30000 non-null  int64
 4   Income (USD)                25424 non-null  float64
 5   Income Stability            28317 non-null  object
 6   Profession                  30000 non-null  object
 7   Type of Employment          22730 non-null  object
 8   Location                    30000 non-null  object
 9   Loan Amount Request (USD)   30000 non-null  float64
 10  Current Loan Expenses (USD) 29828 non-null  float64
 11  Expense Type 1              30000 non-null  object
 12  Expense Type 2              30000 non-null  object
 13  Dependents                  27507 non-null  float64
 14  Credit Score                28297 non-null  float64
 15  No. of Defaults             30000 non-null  int64
 16  Has Active Credit Card      28434 non-null  object
 17  Property ID                 30000 non-null  int64
 18  Property Age                25150 non-null  float64
 19  Property Type               30000 non-null  int64
 20  Property Location           29644 non-null  object
 21  Co-Applicant                30000 non-null  int64
 22  Property Price              30000 non-null  float64
 23  Loan Sanction Amount (USD)  29660 non-null  float64
dtypes: float64(8), int64(5), object(11)
memory usage: 5.5+ MB
```

[5]:
```python
df['Co-Applicant'].unique()
```

[5]:
```
array([   1,    0, -999])
```

[6]:
```python
df['Has Active Credit Card'].unique()
```

[6]:
```
array([nan, 'Unpossessed', 'Active', 'Inactive'], dtype=object)
```

[7] :
```python
#Removing unnecessary columns
df = df.drop(columns=["Customer ID", "Name"])
```

[8] :
```python
# Replace -999 with NaN
df['Co-Applicant'] = df['Co-Applicant'].replace(-999, np.nan)

# Option 1: Impute missing values (e.g., assume no co-applicant)
df['Co-Applicant'] = df['Co-Applicant'].fillna(0)
```

[9] :
```python
# Fill NaN with 'Unknown'
df['Has Active Credit Card'] = df['Has Active Credit Card'].fillna('Unknown')

# Optional: Encode as ordinal
credit_card_map = {
    'Unpossessed': 0,
    'Inactive': 1,
    'Active': 2,
    'Unknown': -1
}
df['Has Active Credit Card'] = df['Has Active Credit Card'].map(credit_card_map)
```

[10] :
```python
df.isnull().sum()
```

[10]:
```
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            0
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
```

[11] :

```
#Filling null values
df['Gender']=df['Gender'].fillna(df['Gender'].mode()[0])
df['Income (USD)']=df['Income (USD)'].fillna(df['Income (USD)'].median())
df['Income Stability']=df['Income Stability'].fillna(df['Income Stability'].
↪mode()[0])
```

[12] :

```
#Dropping this column due to presence of more null values and may categories
df['Type of Employment'].unique()
df=df.drop(columns=['Type of Employment'])
```

[13] :

```
#Current Loan Expenses (USD) — Numeric → fill with median
df['Current Loan Expenses (USD)'] = df['Current Loan Expenses (USD)'].
↪fillna(df['Current Loan Expenses (USD)'].median())

#Dependents — Numeric → fill with mode (likely a small integer like 1 or 2)
df['Dependents'] = df['Dependents'].fillna(df['Dependents'].mode()[0])

#Credit Score — Numeric → fill with median
df['Credit Score'] = df['Credit Score'].fillna(df['Credit Score'].median())

#Property Age — Numeric → fill with median
df['Property Age'] = df['Property Age'].fillna(df['Property Age'].median())

#Property Location — Categorical → fill with mode
df['Property Location'] = df['Property Location'].fillna(df['Property␣
↪Location'].mode()[0])

# Loan Sanction Amount (USD) - Numeric → fill with median
df['Loan Sanction Amount (USD)'] = df['Loan Sanction Amount (USD)'].
↪fillna(df['Loan Sanction Amount (USD)'].median())

df['Loan Sanction Amount (USD)'] = df['Loan Sanction Amount (USD)'].replace(0,␣
↪df['Loan Sanction Amount (USD)'].median())
```

[14] :

```
df.isnull().sum()
```

[14] :

```
Gender                          0
Age                             0
Income (USD)                    0
Income Stability                0
Profession                      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
```

Encoding of variables with values

[15]:
```python
from sklearn.preprocessing import LabelEncoder

# List of categorical columns
cat_cols = [
    'Gender', 'Income Stability', 'Profession',
    'Expense Type 1', 'Expense Type 2',
    'Has Active Credit Card', 'Property Type', 'Property Location','Location'
]

# Create a label encoder instance
le = LabelEncoder()

# Apply label encoding to each column
for col in cat_cols:
    df[col] = le.fit_transform(df[col])
```

Standardization of Features

[16]:
```python
from sklearn.preprocessing import StandardScaler

# Identify numeric columns (excluding categorical and target)
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()

# Optionally exclude target column (e.g., 'Loan Sanction Amount (USD)')
numeric_cols.remove('Loan Sanction Amount (USD)')

# Initialize scaler
scaler = StandardScaler()

# Fit and transform numeric features
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

[17]:
```python
df.head(10)
```

[17]:

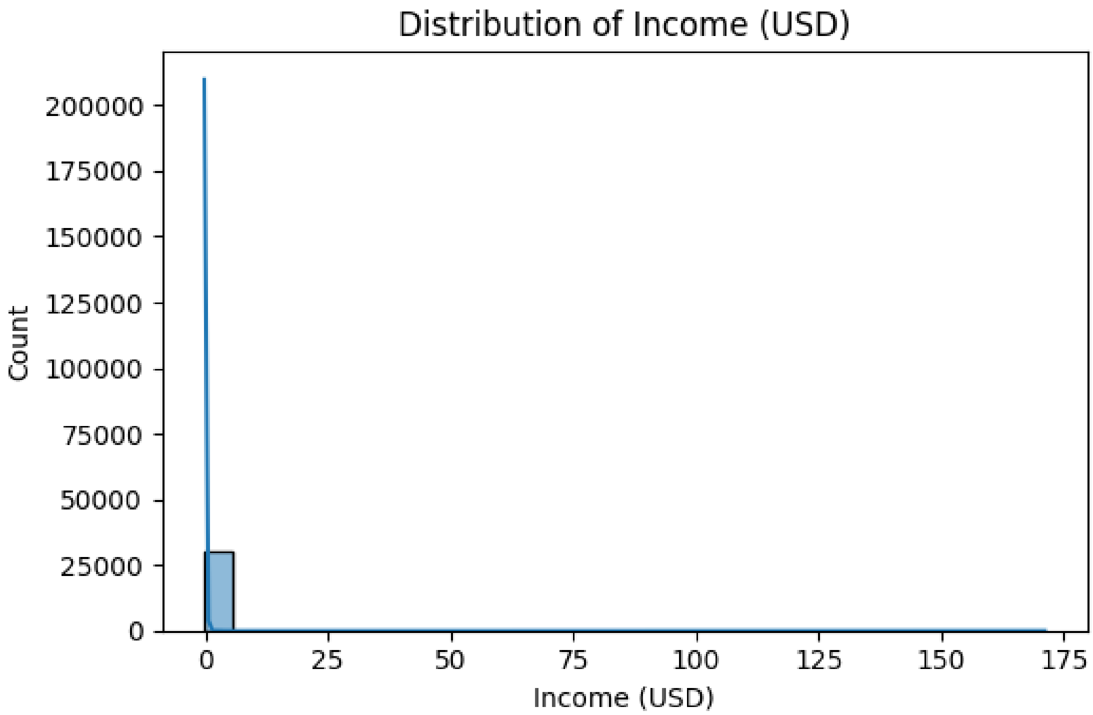
```
     Gender       Age  Income (USD)  Income Stability  Profession  Location  \
0 -1.007092  0.991451     -0.061266          0.305833    0.834973  0.142149
1  0.992958 -0.504355      0.229972          0.305833    0.834973  0.142149
2 -1.007092  1.552379     -0.152389         -3.269763   -0.686548  0.142149
3 -1.007092  1.552379     -0.033357         -3.269763   -0.686548 -1.762481
4 -1.007092 -0.566680      0.004480          0.305833    0.834973  0.142149
5 -1.007092  1.240752     -0.128594          0.305833   -0.306168 -1.762481
6  0.992958  0.181223     -0.019940          0.305833    0.834973  0.142149
7 -1.007092  0.305874     -0.033357          0.305833   -0.306168  0.142149
8 -1.007092 -0.130403     -0.122697          0.305833    0.834973 -1.762481
9  0.992958 -1.376908     -0.098577          0.305833    0.834973 -1.762481

   Loan Amount Request (USD)  Current Loan Expenses (USD)  Expense Type 1  \
0                  -0.269027                    -0.660358       -0.749241
1                  -0.705269                     0.392886       -0.749241
2                  -0.726171                    -0.946193       -0.749241
3                  -0.147279                    -0.422775       -0.749241
4                   0.420461                     0.374693       -0.749241
5                  -0.913593                    -0.906788       -0.749241
6                   1.070530                     1.227526        1.334685
7                   2.544436                     1.682224       -0.749241
8                  -0.901713                    -1.012348       -0.749241
9                  -0.784989                     0.411038       -0.749241

   Expense Type 2  ...  Credit Score  No. of Defaults  Has Active Credit Card  \
0       -1.433524  ...      0.992493        -0.490502               -2.096903
1        0.697582  ...      0.578136        -0.490502               -1.001762
2        0.697582  ...      1.330799        -0.490502               -1.001762
3        0.697582  ...      1.324379         2.038728               -1.001762
4        0.697582  ...      0.080879         2.038728                1.188520
5       -1.433524  ...     -0.795636         2.038728                0.093379
6        0.697582  ...     -1.463830        -0.490502               -1.001762
7       -1.433524  ...      1.032730        -0.490502                1.188520
8        0.697582  ...     -0.493571         2.038728                1.188520
9       -1.433524  ...     -1.806987        -0.490502               -1.001762

   Property ID  Property Age  Property Type  Property Location  Co-Applicant  \
0     0.846998     -0.060969       1.376731          -1.214540      0.419205
1     0.368086      0.230298      -0.411309          -1.214540      0.419205
2     0.152923     -0.152102      -0.411309           1.283229     -2.385467
3     1.346732     -0.032979      -0.411309           0.034344      0.419205
4     0.739417      0.004783       1.376731           0.034344      0.419205
5    -0.037948     -0.128305      -0.411309          -1.214540      0.419205
6    -0.954127     -0.019639      -1.305329           0.034344      0.419205
7    -0.652204     -0.032979      -0.411309           1.283229      0.419205
8    -0.905541     -0.122407       1.376731          -1.214540      0.419205
9     1.322440     -0.098284      -0.411309           1.283229      0.419205
```

```
   Property  Price    Loan Sanction Amount (USD)
0         -0.126419                     54607.180
1         -0.822772                     37469.980
2         -0.634103                     36474.430
3         -0.110298                     56040.540
4          0.821057                     74008.280
5         -0.947245                     22382.570
6          0.954495                     35209.395
7          2.878533                    168218.240
8         -0.821570                     22842.290
9         -0.681642                     35209.395

[10 rows x 21 columns]
```

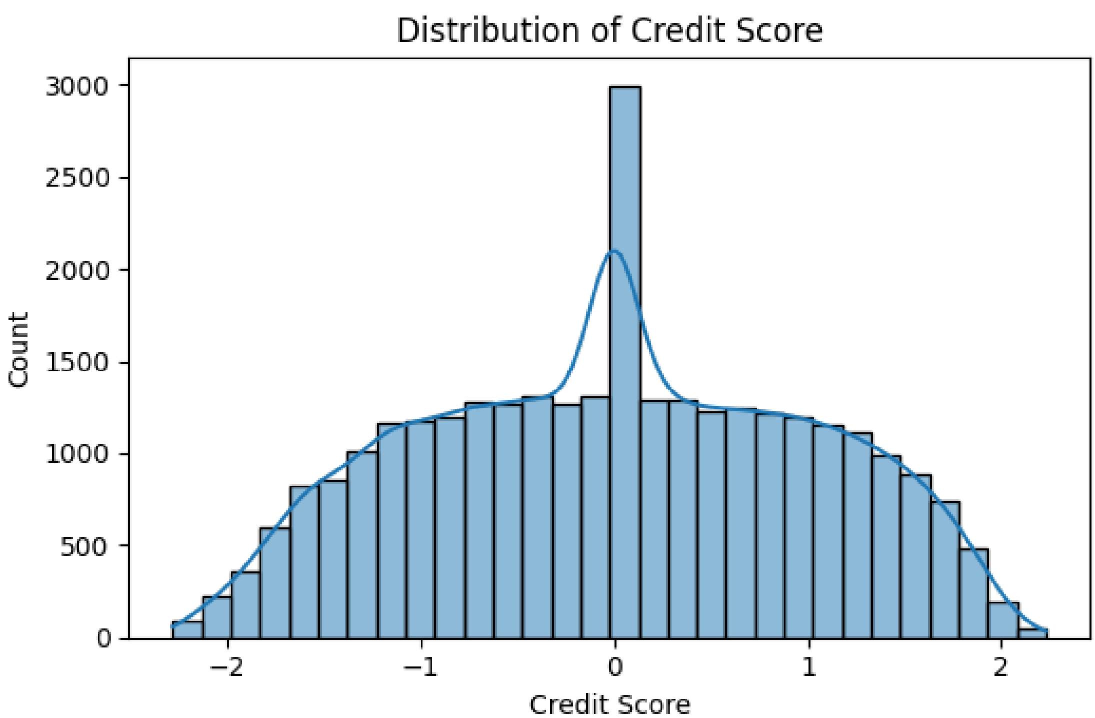
EDA

Histogram

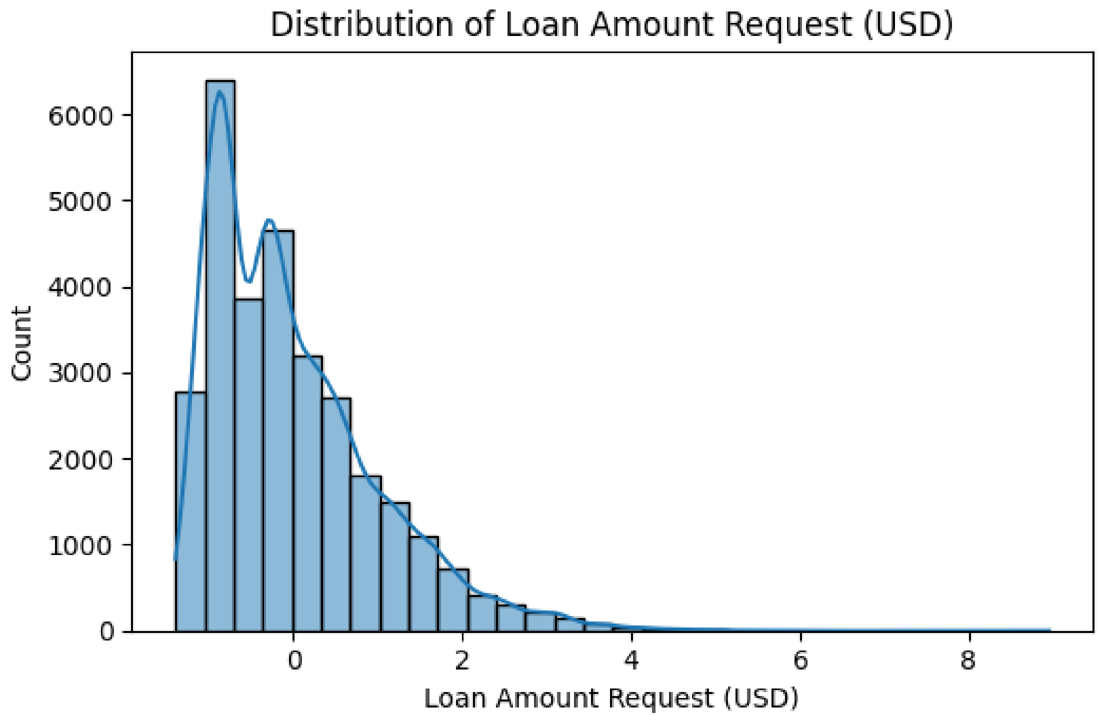
[18]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot distributions for selected numeric columns
cols_to_plot = ['Income (USD)', 'Credit Score', 'Loan Amount Request (USD)',␣
↪'Loan Sanction Amount (USD)']

for col in cols_to_plot:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')
    plt.tight_layout()
    plt.show()
```

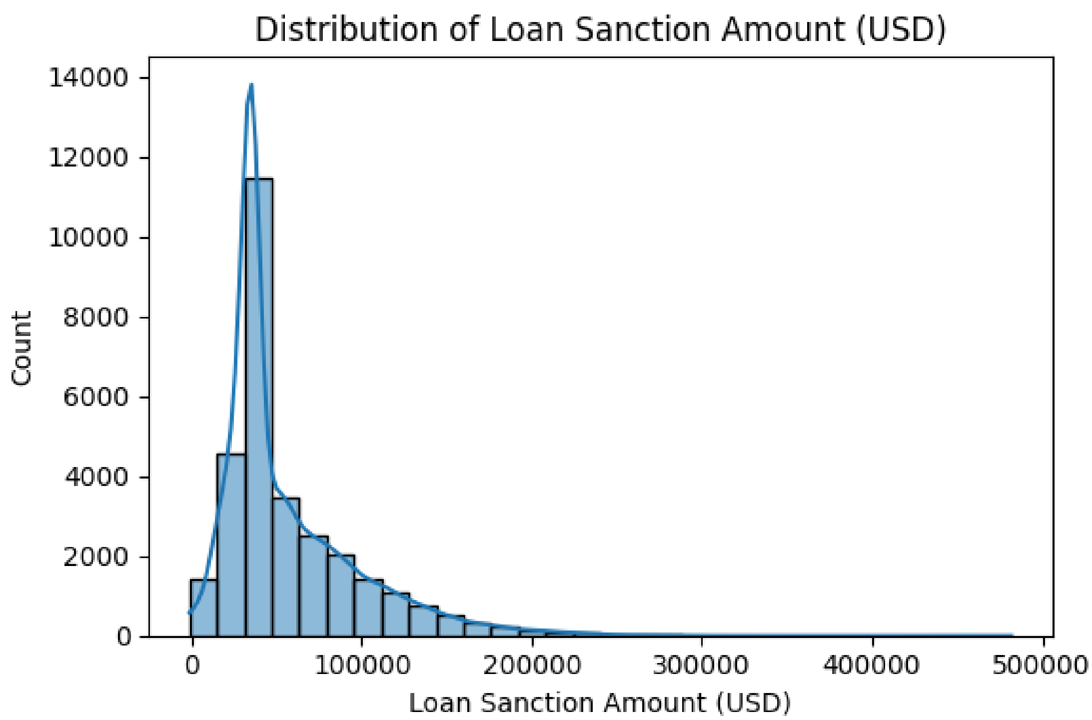


Distribution of Income (USD)
200000
175000
150000
125000
100000
75000
50000
25000
0
Count
0
25
50
75
100
125
150
175
Income (USD)



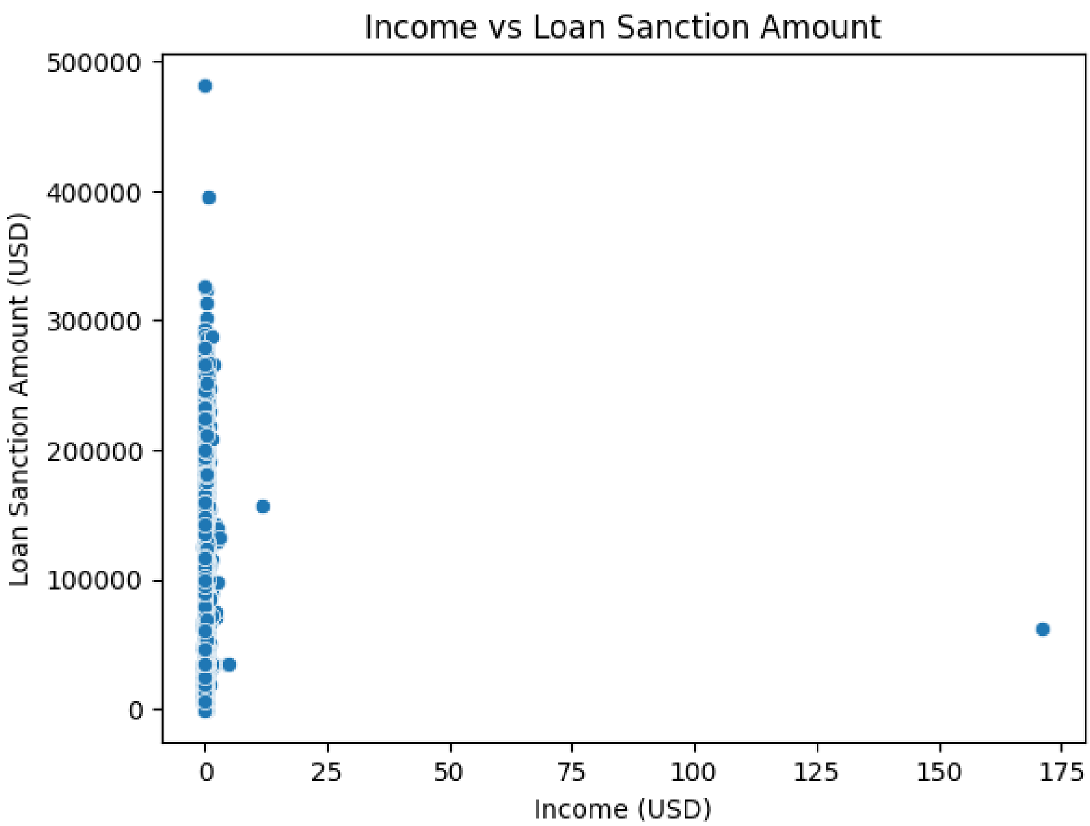
Distribution of Credit Score
3000
2500
2000
1500
1000
500
0
Count
−2
−1
0
1
2
Credit Score

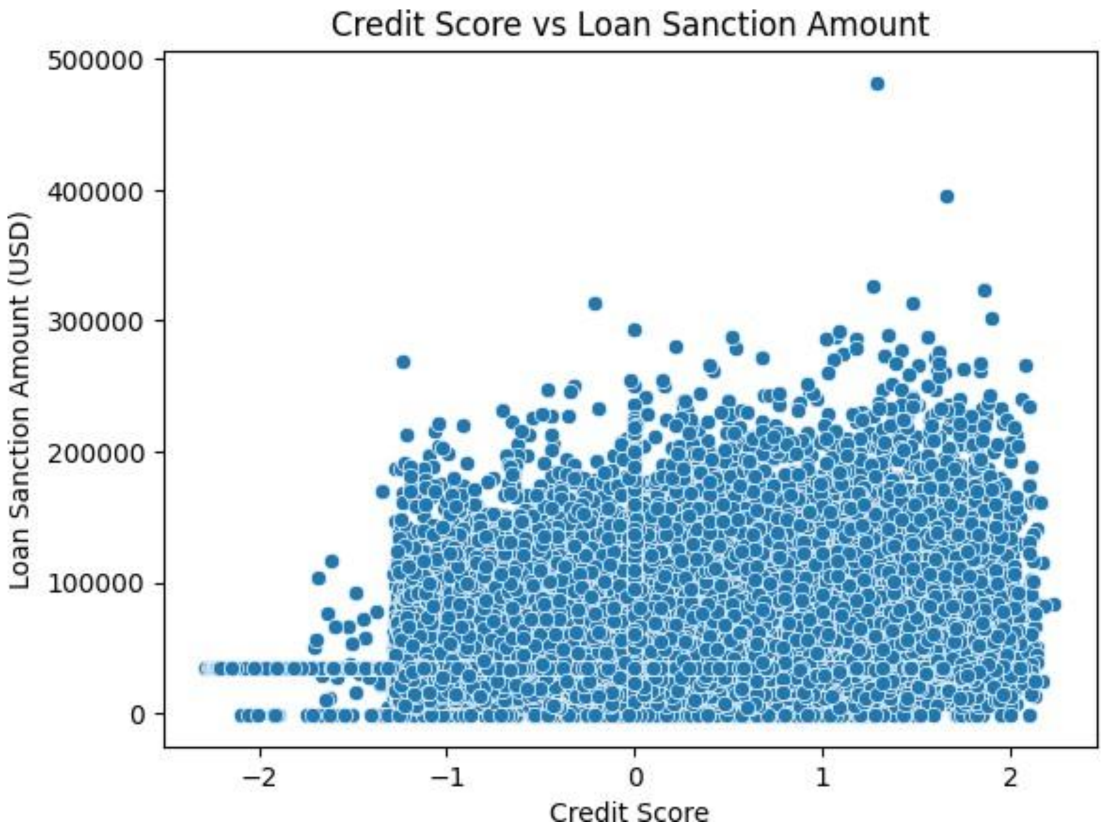

Distribution of Loan Amount Request (USD)
6000
5000
4000
3000
2000
1000
0
Count
0
2
4
6
8
Loan Amount Request (USD)


Distribution of Loan Sanction Amount (USD)
14000
12000
10000
8000
6000
4000
2000
0
Count
0
100000
200000
300000
400000
500000
Loan Sanction Amount (USD)

Scatter Plot

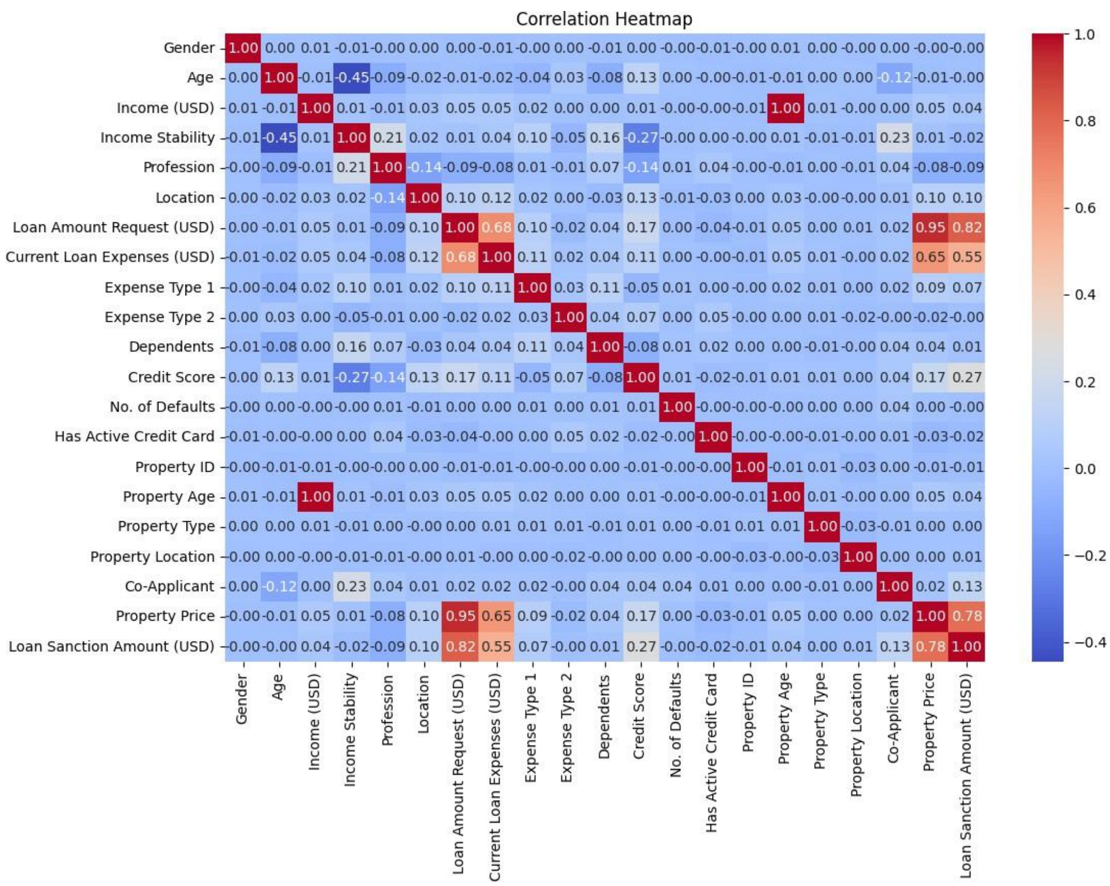
[19]:
```python
# Income vs Loan Sanction Amount
sns.scatterplot(data=df, x='Income (USD)', y='Loan Sanction Amount (USD)')
plt.title('Income vs Loan Sanction Amount')
plt.show()

# Credit Score vs Loan Sanction Amount
sns.scatterplot(data=df, x='Credit Score', y='Loan Sanction Amount (USD)')
plt.title('Credit Score vs Loan Sanction Amount')
plt.show()
```





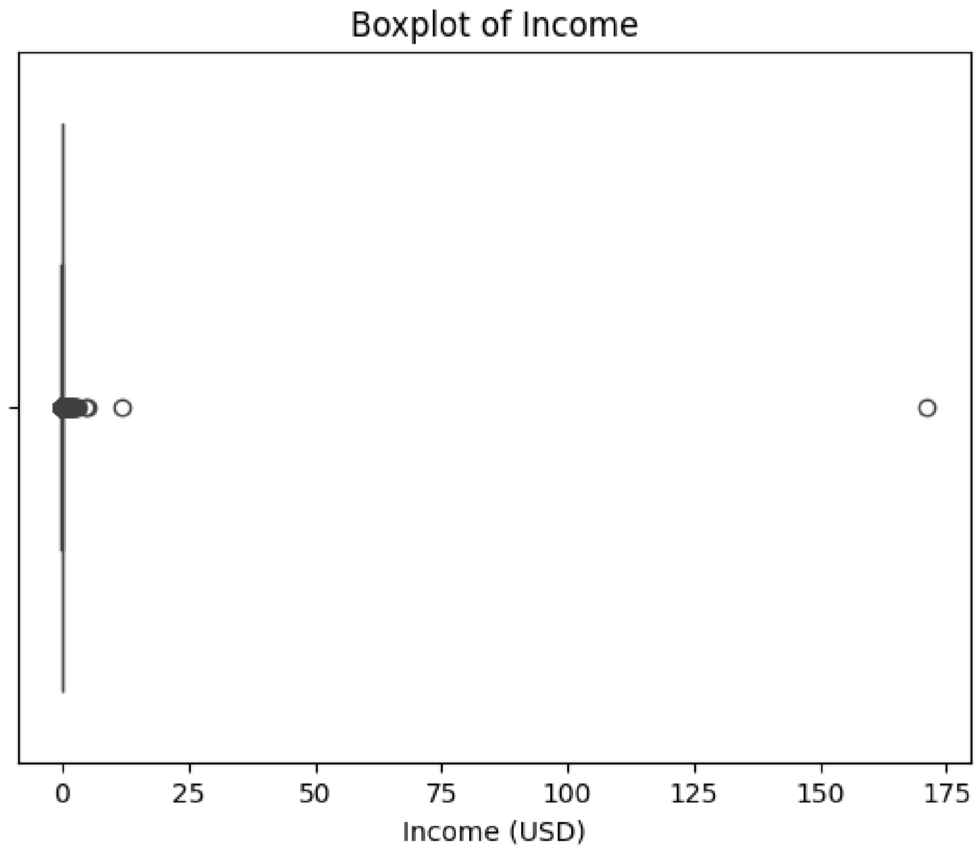
Correlation heatmap

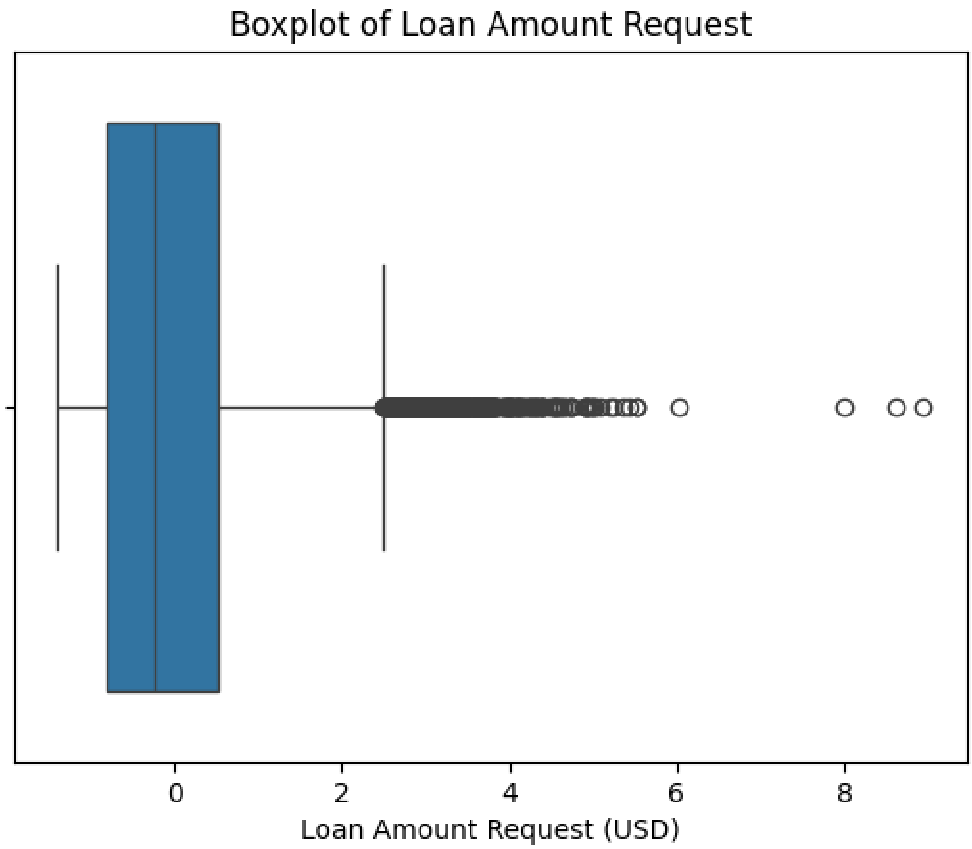
[20] :
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



BoxPlot

[21] :
```
# Boxplot for Income
sns.boxplot(x=df['Income (USD)'])
plt.title('Boxplot of Income')
plt.show()

# Boxplot for Loan Amount Request
sns.boxplot(x=df['Loan Amount Request (USD)'])
plt.title('Boxplot of Loan Amount Request')
plt.show()
```


Boxplot of Income
0
25
50
75
100
125
150
175
Income (USD)



Train Test Split

[22] :
```
from sklearn.model_selection import train_test_split

# Define target variable
target = 'Loan Sanction Amount (USD)'

# Define feature columns
X = df.drop(columns=[target])
y = df[target]

# Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 random_state=42)
```

Model Training

[23] :
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error

# Initialize and train the model
```

```python
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)


# Evaluation Metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print results
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
Mean Squared Error (MSE): 527445271.77
Root Mean Squared Error (RMSE): 22966.18
Mean Absolute Error (MAE): 13803.42
R² Score: 0.69
```

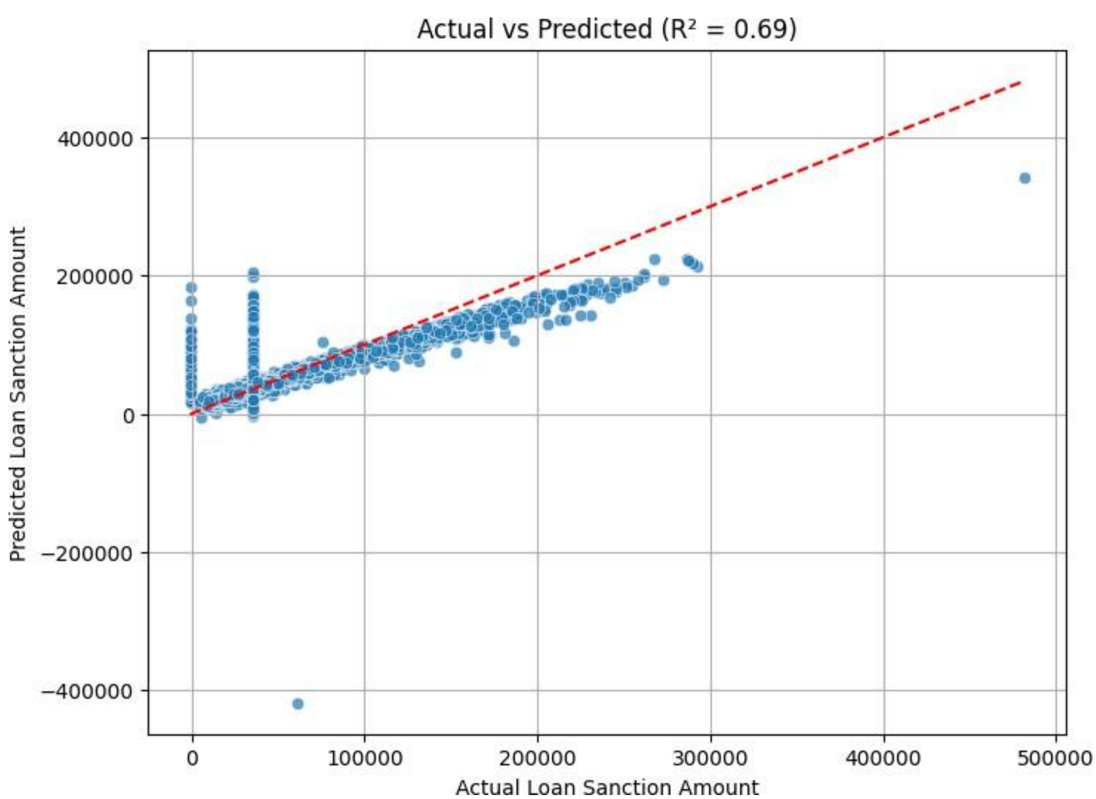
[24] :
```python
from sklearn.model_selection import KFold, cross_val_score
import numpy as np

# Define K-Fold with 5 splits
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Custom scoring functions
mse_scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',␣
↪cv=kf)
mae_scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error',␣
↪cv=kf)
r2_scores = cross_val_score(model, X, y, scoring='r2', cv=kf)

# Convert negative MSE/MAE to positive
mse_scores = -mse_scores
mae_scores = -mae_scores
rmse_scores = np.sqrt(mse_scores)

# Print metrics per fold
print("Fold-wise Metrics:")
for i in range(len(mse_scores)):
    print(f"Fold {i+1}:")
    print(f"  MSE : {mse_scores[i]:.2f}")
```

```python
    print(f"    RMSE: {rmse_scores[i]:.2f}")
    print(f"    MAE : {mae_scores[i]:.2f}")
    print(f"    R²  : {r2_scores[i]:.2f}")
    print()

# Print average performance
print("Average Metrics Across Folds:")
print(f"Average MSE : {mse_scores.mean():.2f}")
print(f"Average RMSE: {rmse_scores.mean():.2f}")
print(f"Average MAE : {mae_scores.mean():.2f}")
print(f"Average R²  : {r2_scores.mean():.2f}")
```

```
Fold-wise Metrics:
Fold 1:
  MSE : 527445271.77
  RMSE: 22966.18
  MAE : 13803.42
  R²  : 0.69

Fold 2:
  MSE : 493351608.13
  RMSE: 22211.52
  MAE : 13779.90
  R²  : 0.70

Fold 3:
  MSE : 544801753.47
  RMSE: 23340.99
  MAE : 14030.71
  R²  : 0.67

Fold 4:
  MSE : 513654615.80
  RMSE: 22663.95
  MAE : 14044.93
  R²  : 0.70

Fold 5:
  MSE : 440761214.18
  RMSE: 20994.31
  MAE : 13347.16
  R²  : 0.73

Average Metrics Across Folds:
Average MSE : 504002892.67
Average RMSE: 22435.39
Average MAE : 13801.23
Average R²  : 0.70
```

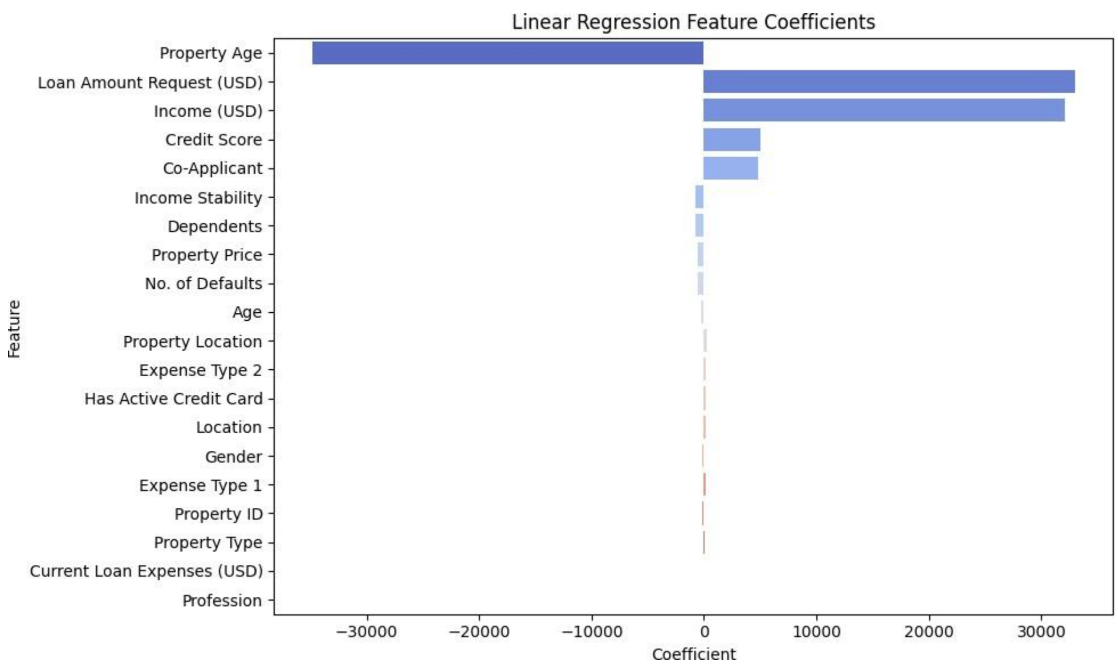
Actual vs Predicted values

[25] :
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import r2_score

# Predict values
y_pred = model.predict(X_test)

# Plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
↪color='red', linestyle='--')
plt.xlabel("Actual Loan Sanction Amount")
plt.ylabel("Predicted Loan Sanction Amount")
plt.title(f"Actual vs Predicted (R² = {r2_score(y_test, y_pred):.2f})")
plt.grid(True)
plt.show()
```



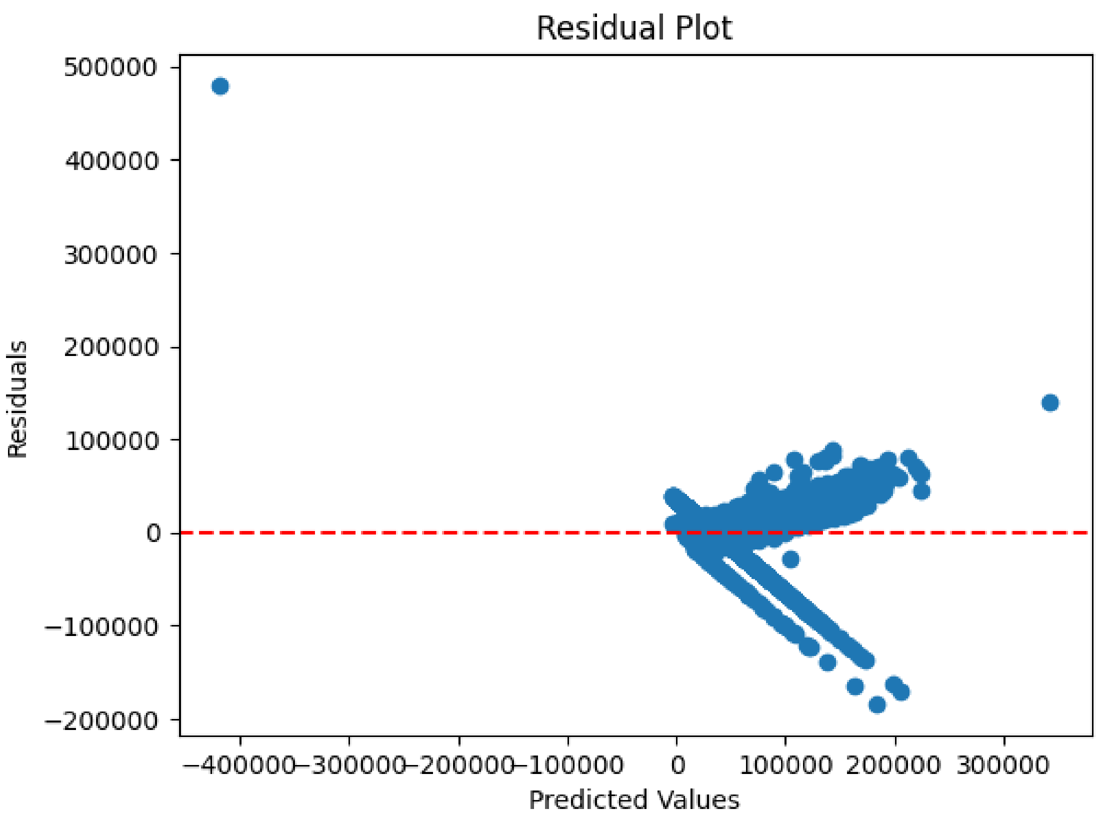
[26] :
```
# Create a DataFrame for coefficients
coeff_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Coefficient': model.coef_
}).sort_values(by='Coefficient', key=abs, ascending=False)

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Coefficient', y='Feature', data=coeff_df, palette='coolwarm')
plt.title("Linear Regression Feature Coefficients")
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_7636/3484898736.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Coefficient', y='Feature', data=coeff_df, palette='coolwarm')
```



[27] :
```
# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Predicted Values")
```

```python
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```



[28] :

```python
from sklearn.linear_model import LassoCV

lasso = LassoCV(cv=5, random_state=42)
lasso.fit(X_train, y_train)

# Get features with non-zero coefficients
lasso_features = X_train.columns[lasso.coef_ != 0]
print(" Features selected by Lasso:\n", lasso_features)
```

```
 Features selected by Lasso:
Index(['Gender', 'Age', 'Income Stability', 'Location',
       'Loan Amount Request (USD)', 'Expense Type 2', 'Dependents',
       'Credit Score', 'No. of Defaults', 'Has Active Credit Card',
       'Property ID', 'Property Location', 'Co-Applicant'],
      dtype='object')
```

6 Included Plots:

- **Heatmap**: Displays the correlation between features to identify strong positive or negative linear relationships.
- **Boxplot**: Visualizes the spread, median, and potential outliers in numerical variables, useful for detecting skewness or extreme values.
- **Scatter Plot**: Shows the relationship between two numerical variables, helping to identify trends or clusters.
- **Histogram**: Illustrates the frequency distribution of a variable, showing how data points are spread across intervals.

7 Best Practices Followed

- **Consistent Data Preprocessing:** Ensured all features were cleaned, encoded, and standardized uniformly before feeding into the model. This helps in improving model performance and generalizability.
- **Model Validation with Cross-Validation:** Used 5-fold cross-validation to evaluate the model's robustness across different subsets of the data, reducing the chances of overfitting or bias due to a specific data split.

8 Learning Outcomes

- **End-to-End Workflow Understanding:** Gained hands-on experience with the full machine learning pipeline including EDA, preprocessing, feature selection, training, evaluation, and visualization.
- **Interpretation of Model Performance:** Learned how to use statistical metrics like MAE, MSE, RMSE, and R^2 Score to evaluate regression models and interpret residual plots for diagnosing model fit.

9 SVR Code:

loan_amount_prediction_svm

September 3, 2025

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

[2]:
```
df=pd.read_csv('train.csv')
df.head()
```

[2]:
```
  Customer ID                Name Gender  Age  Income (USD) Income Stability  \
0     C-36995    Frederica Shealy      F   56       1933.05              Low
1     C-33999  America Calderone       M   32       4952.91              Low
2      C-3770        Rosetta Verne      F   65        988.19             High
3     C-26480          Zoe Chitty      F   65           NaN             High
4     C-23459        Afton Venema      F   31       2614.77              Low

  Profession        Type of Employment    Location  Loan Amount Request (USD)  \
0    Working               Sales staff  Semi-Urban                   72809.58
1    Working                       NaN  Semi-Urban                   46837.47
2  Pensioner                       NaN  Semi-Urban                   45593.04
3  Pensioner                       NaN       Rural                   80057.92
4    Working  High skill tech staff  Semi-Urban                  113858.89

   ... Credit Score No. of Defaults Has Active Credit Card  Property ID  \
0  ...       809.44               0                    NaN          746
1  ...       780.40               0            Unpossessed          608
2  ...       833.15               0            Unpossessed          546
3  ...       832.70               1            Unpossessed          890
4  ...       745.55               1                 Active          715

  Property Age  Property Type Property Location  Co-Applicant  \
0      1933.05              4             Rural             1
1      4952.91              2             Rural             1
2       988.19              2             Urban             0
3          NaN              2        Semi-Urban             1
4      2614.77              4        Semi-Urban             1
```

```
    Property Price   Loan Sanction Amount (USD)
0       119933.46                     54607.18
1        54791.00                     37469.98
2        72440.58                     36474.43
3       121441.51                     56040.54
4       208567.91                     74008.28

[5 rows x 24 columns]
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Customer ID                 30000 non-null  object
 1   Name                        30000 non-null  object
 2   Gender                      29947 non-null  object
 3   Age                         30000 non-null  int64
 4   Income (USD)                25424 non-null  float64
 5   Income Stability            28317 non-null  object
 6   Profession                  30000 non-null  object
 7   Type of Employment          22730 non-null  object
 8   Location                    30000 non-null  object
 9   Loan Amount Request (USD)   30000 non-null  float64
 10  Current Loan Expenses (USD) 29828 non-null  float64
 11  Expense Type 1              30000 non-null  object
 12  Expense Type 2              30000 non-null  object
 13  Dependents                  27507 non-null  float64
 14  Credit Score                28297 non-null  float64
 15  No. of Defaults             30000 non-null  int64
 16  Has Active Credit Card      28434 non-null  object
 17  Property ID                 30000 non-null  int64
 18  Property Age                25150 non-null  float64
 19  Property Type               30000 non-null  int64
 20  Property Location           29644 non-null  object
 21  Co-Applicant                30000 non-null  int64
 22  Property Price              30000 non-null  float64
 23  Loan Sanction Amount (USD)  29660 non-null  float64
dtypes: float64(8), int64(5), object(11)
memory usage: 5.5+ MB
```

[4]:
```
df['Co-Applicant'].unique()
```

[4]:
```
array([   1,    0, -999])
```

[5]:
```
df['Has Active Credit Card'].unique()
```

[5]:
```
array([nan, 'Unpossessed', 'Active', 'Inactive'], dtype=object)
```

[6]:
```
#Removing unnecessary columns
df = df.drop(columns=["Customer ID", "Name"])
```

[7]:
```
# Replace -999 with NaN
df['Co-Applicant'] = df['Co-Applicant'].replace(-999, np.nan)

# Option 1: Impute missing values (e.g., assume no co-applicant)
df['Co-Applicant'] = df['Co-Applicant'].fillna(0)
```

[8]:
```
# Fill NaN with 'Unknown'
df['Has Active Credit Card'] = df['Has Active Credit Card'].fillna('Unknown')

# Optional: Encode as ordinal
credit_card_map = {
    'Unpossessed': 0,
    'Inactive': 1,
    'Active': 2,
    'Unknown': -1
}
df['Has Active Credit Card'] = df['Has Active Credit Card'].map(credit_card_map)
```

[9]:
```
df.isnull().sum()
```

[9]:
```
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           0
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
```

[10]:
```python
#Filling null values
df['Gender']=df['Gender'].fillna(df['Gender'].mode()[0])
df['Income (USD)']=df['Income (USD)'].fillna(df['Income (USD)'].median())
df['Income Stability']=df['Income Stability'].fillna(df['Income Stability'].
 ↪mode()[0])
```

[11]:
```python
#Dropping this column due to presence of more null values and may categories
df['Type of Employment'].unique()
df=df.drop(columns=['Type of Employment'])
```

[12]:
```python
#Current Loan Expenses (USD) — Numeric → fill with median
df['Current Loan Expenses (USD)'] = df['Current Loan Expenses (USD)'].
 ↪fillna(df['Current Loan Expenses (USD)'].median())

#Dependents — Numeric → fill with mode (likely a small integer like 1 or 2)
df['Dependents'] = df['Dependents'].fillna(df['Dependents'].mode()[0])

#Credit Score — Numeric → fill with median
df['Credit Score'] = df['Credit Score'].fillna(df['Credit Score'].median())

#Property Age — Numeric → fill with median
df['Property Age'] = df['Property Age'].fillna(df['Property Age'].median())

#Property Location — Categorical → fill with mode
df['Property Location'] = df['Property Location'].fillna(df['Property␣
 ↪Location'].mode()[0])

# Loan Sanction Amount (USD) - Numeric → fill with median
df['Loan Sanction Amount (USD)'] = df['Loan Sanction Amount (USD)'].
 ↪fillna(df['Loan Sanction Amount (USD)'].median())

df['Loan Sanction Amount (USD)'] = df['Loan Sanction Amount (USD)'].replace(0,␣
 ↪df['Loan Sanction Amount (USD)'].median())
```

[13]:
```python
df.isnull().sum()
```

[13]:
```
Gender                        0
Age                           0
Income (USD)                  0
Income Stability              0
Profession                    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
```

Encoding of variables with values

[14]:
```python
from sklearn.preprocessing import LabelEncoder

# List of categorical columns
cat_cols = [
    'Gender', 'Income Stability', 'Profession',
    'Expense Type 1', 'Expense Type 2',
    'Has Active Credit Card', 'Property Type', 'Property Location','Location'
]

# Create a label encoder instance
le = LabelEncoder()

# Apply label encoding to each column
for col in cat_cols:
    df[col] = le.fit_transform(df[col])
```

Standardization of Features

[15]:
```python
from sklearn.preprocessing import StandardScaler

# Identify numeric columns (excluding categorical and target)
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()

# Optionally exclude target column (e.g., 'Loan Sanction Amount (USD)')
numeric_cols.remove('Loan Sanction Amount (USD)')

# Initialize scaler
scaler = StandardScaler()

# Fit and transform numeric features
```

```python
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

[16]:

```python
df.head(10)
```

[16]:

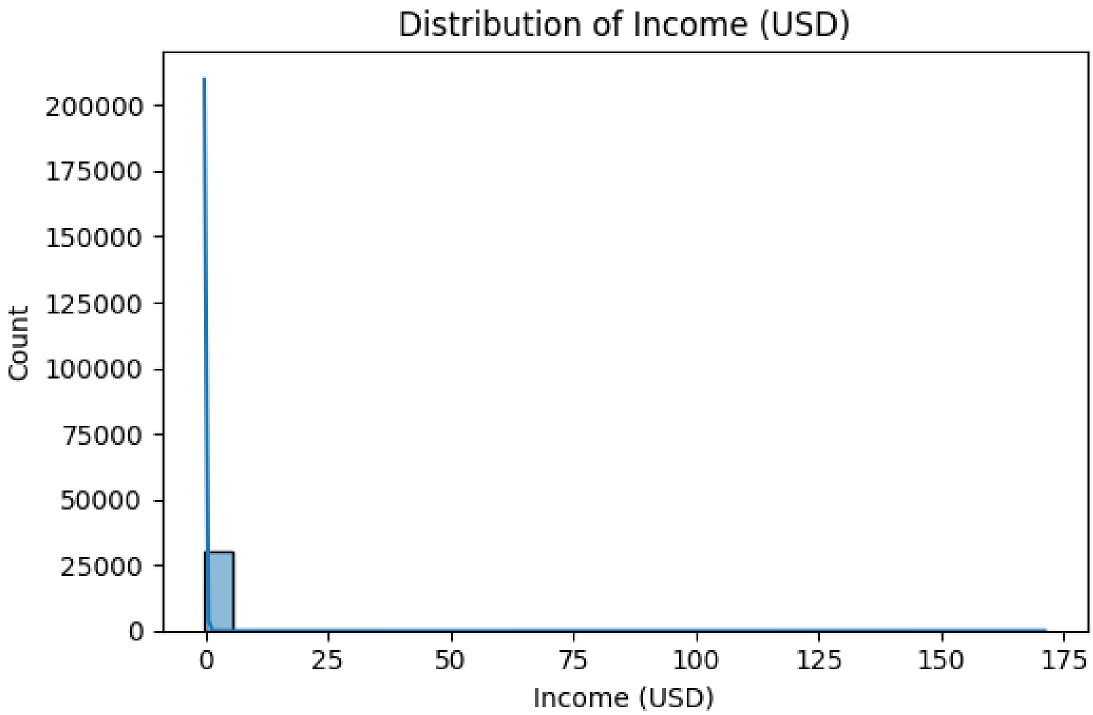
```
     Gender       Age  Income (USD)  Income Stability  Profession  Location  \
0 -1.007092  0.991451     -0.061266          0.305833    0.834973  0.142149
1  0.992958 -0.504355      0.229972          0.305833    0.834973  0.142149
2 -1.007092  1.552379     -0.152389         -3.269763   -0.686548  0.142149
3 -1.007092  1.552379     -0.033357         -3.269763   -0.686548 -1.762481
4 -1.007092 -0.566680      0.004480          0.305833    0.834973  0.142149
5 -1.007092  1.240752     -0.128594          0.305833   -0.306168 -1.762481
6  0.992958  0.181223     -0.019940          0.305833    0.834973  0.142149
7 -1.007092  0.305874     -0.033357          0.305833   -0.306168  0.142149
8 -1.007092 -0.130403     -0.122697          0.305833    0.834973 -1.762481
9  0.992958 -1.376908     -0.098577          0.305833    0.834973 -1.762481

   Loan Amount Request (USD)  Current Loan Expenses (USD)  Expense Type 1  \
0                  -0.269027                    -0.660358       -0.749241
1                  -0.705269                     0.392886       -0.749241
2                  -0.726171                    -0.946193       -0.749241
3                  -0.147279                    -0.422775       -0.749241
4                   0.420461                     0.374693       -0.749241
5                  -0.913593                    -0.906788       -0.749241
6                   1.070530                     1.227526        1.334685
7                   2.544436                     1.682224       -0.749241
8                  -0.901713                    -1.012348       -0.749241
9                  -0.784989                     0.411038       -0.749241

   Expense Type 2  ...  Credit Score  No. of Defaults  Has Active Credit Card  \
0       -1.433524  ...      0.992493        -0.490502               -2.096903
1        0.697582  ...      0.578136        -0.490502               -1.001762
2        0.697582  ...      1.330799        -0.490502               -1.001762
3        0.697582  ...      1.324379         2.038728               -1.001762
4        0.697582  ...      0.080879         2.038728                1.188520
5       -1.433524  ...     -0.795636         2.038728                0.093379
6        0.697582  ...     -1.463830        -0.490502               -1.001762
7       -1.433524  ...      1.032730        -0.490502                1.188520
8        0.697582  ...     -0.493571         2.038728                1.188520
9       -1.433524  ...     -1.806987        -0.490502               -1.001762

   Property ID  Property Age  Property Type  Property Location  Co-Applicant  \
0     0.846998     -0.060969       1.376731          -1.214540      0.419205
1     0.368086      0.230298      -0.411309          -1.214540      0.419205
2     0.152923     -0.152102      -0.411309           1.283229     -2.385467
3     1.346732     -0.032979      -0.411309           0.034344      0.419205
4     0.739417      0.004783       1.376731           0.034344      0.419205
5    -0.037948     -0.128305      -0.411309          -1.214540      0.419205
```

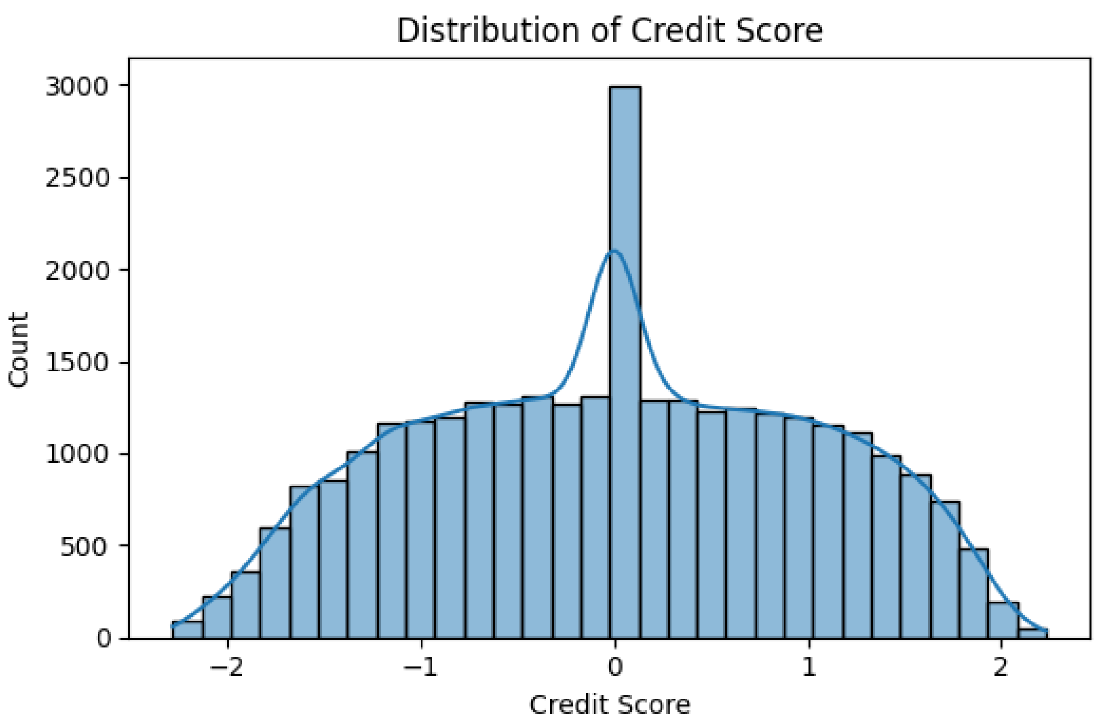
```
6   -0.954127   -0.019639   -1.305329        0.034344    0.419205
7   -0.652204   -0.032979   -0.411309        1.283229    0.419205
8   -0.905541   -0.122407    1.376731       -1.214540    0.419205
9    1.322440   -0.098284   -0.411309        1.283229    0.419205

   Property Price  Loan Sanction Amount (USD)
0       -0.126419                  54607.180
1       -0.822772                  37469.980
2       -0.634103                  36474.430
3       -0.110298                  56040.540
4        0.821057                  74008.280
5       -0.947245                  22382.570
6        0.954495                  35209.395
7        2.878533                 168218.240
8       -0.821570                  22842.290
9       -0.681642                  35209.395

[10 rows x 21 columns]
```

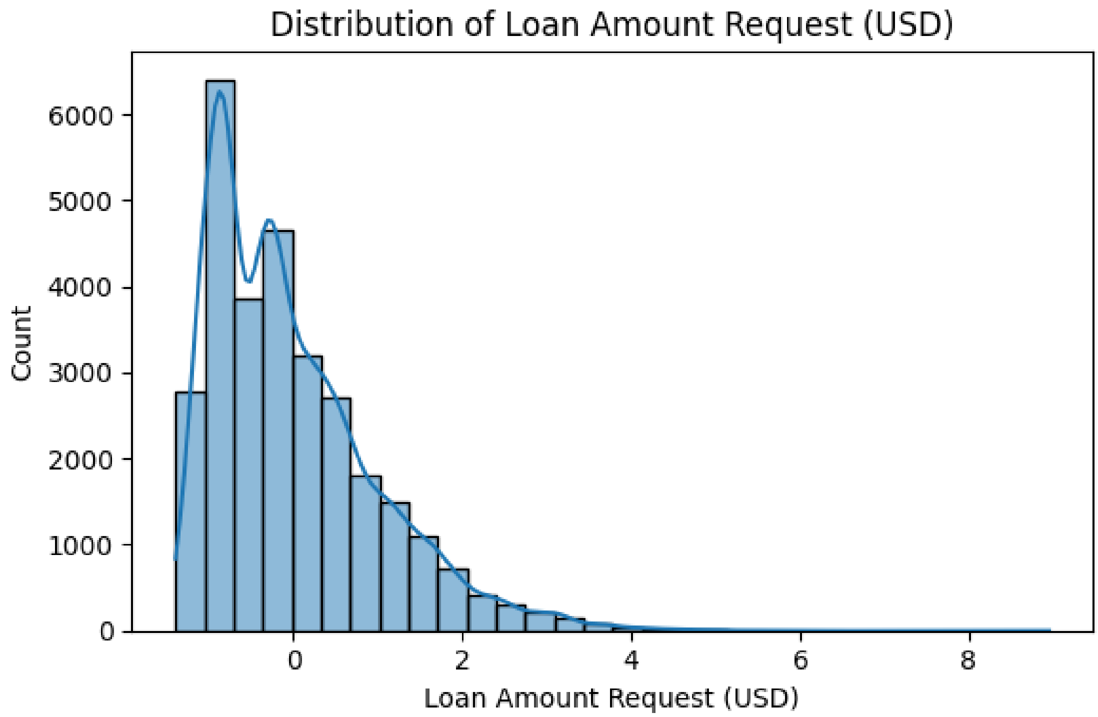
EDA

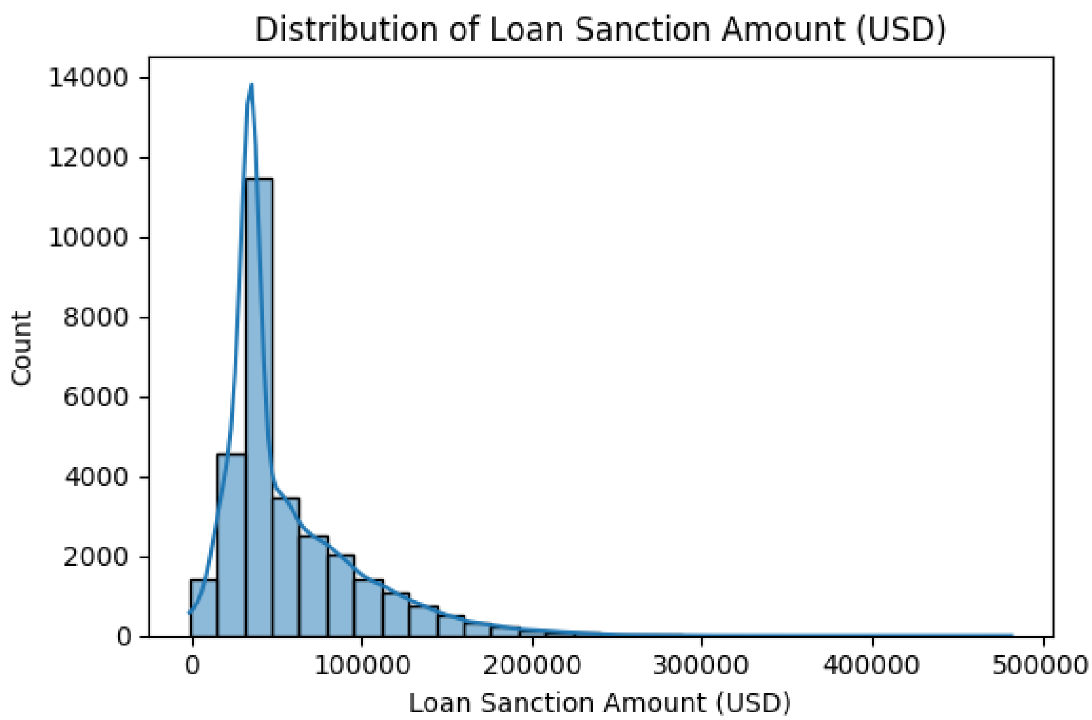
Histogram

[17] :
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Plot distributions for selected numeric columns
cols_to_plot = ['Income (USD)', 'Credit Score', 'Loan Amount Request (USD)',␣
↪'Loan Sanction Amount (USD)']

for col in cols_to_plot:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')
    plt.tight_layout()
    plt.show()
```


Distribution of Income (USD)
Count
200000
175000
150000
125000
100000
75000
50000
25000
0
0
25
50
75
100
125
150
175
Income (USD)

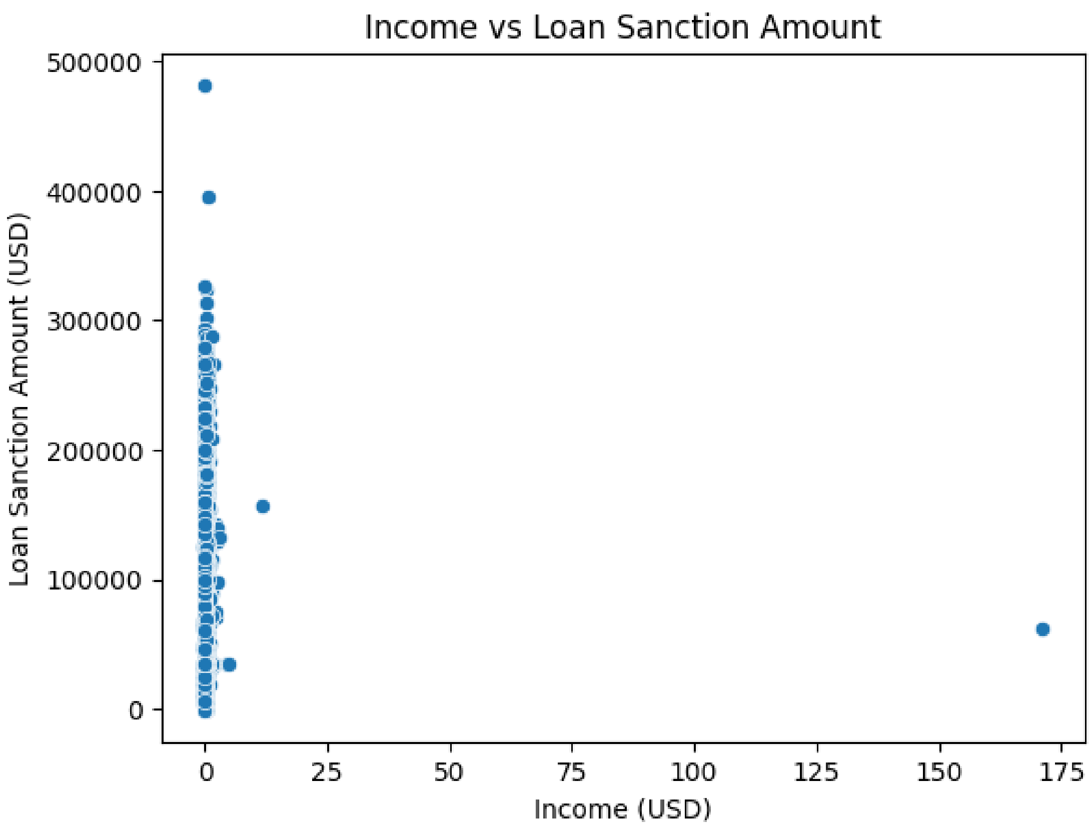

Distribution of Credit Score
Count
3000
2500
2000
1500
1000
500
0
−2
−1
0
1
2
Credit Score

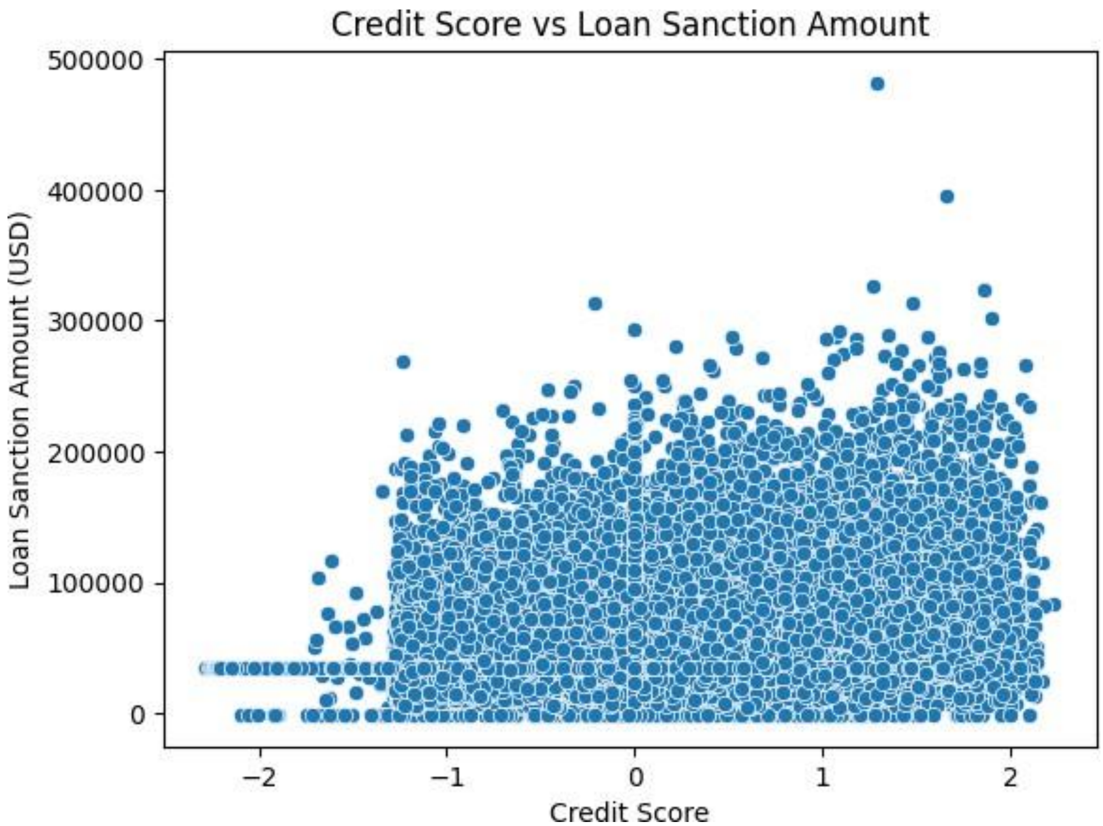

Distribution of Loan Amount Request (USD)
Count
0
1000
2000
3000
4000
5000
6000
0
2
4
6
8
Loan Amount Request (USD)


Distribution of Loan Sanction Amount (USD)
Count
0
2000
4000
6000
8000
10000
12000
14000
0
100000
200000
300000
400000
500000
Loan Sanction Amount (USD)

Scatter Plot

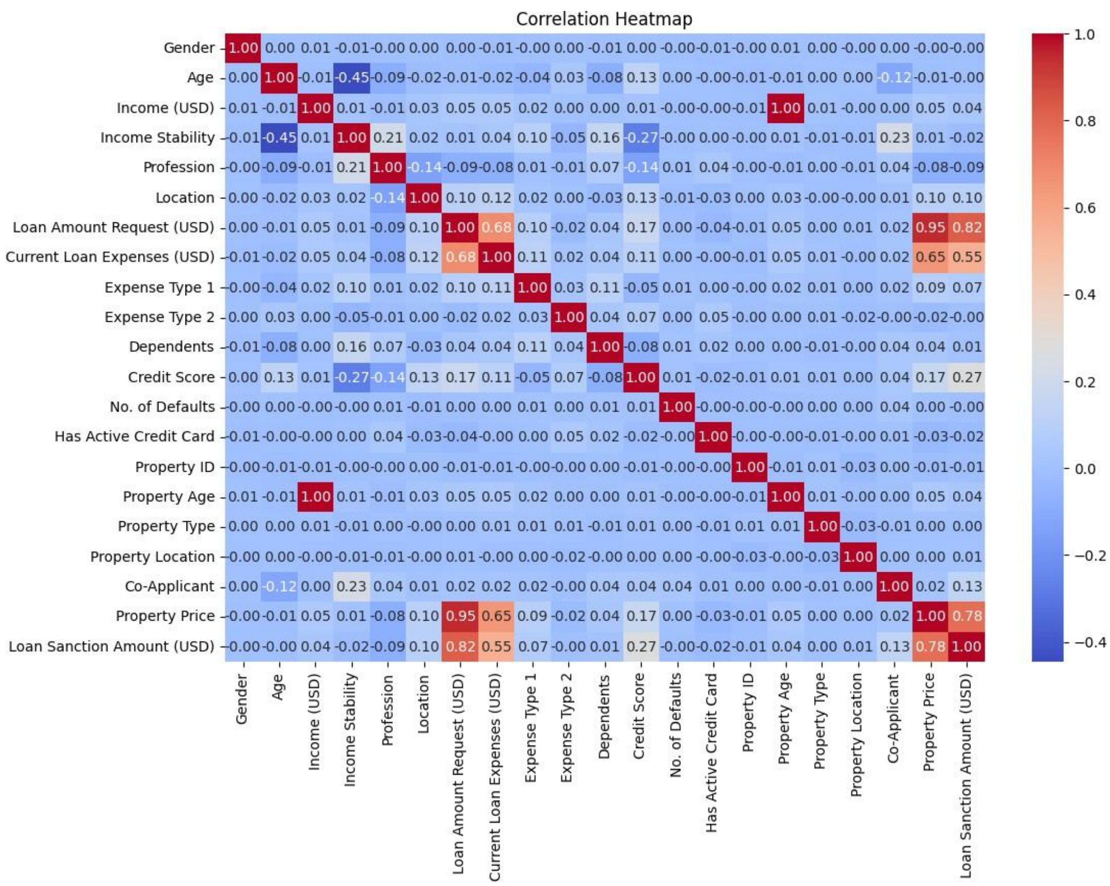
[18]:
```
# Income vs Loan Sanction Amount
sns.scatterplot(data=df, x='Income (USD)', y='Loan Sanction Amount (USD)')
plt.title('Income vs Loan Sanction Amount')
plt.show()

# Credit Score vs Loan Sanction Amount
sns.scatterplot(data=df, x='Credit Score', y='Loan Sanction Amount (USD)')
plt.title('Credit Score vs Loan Sanction Amount')
plt.show()
```





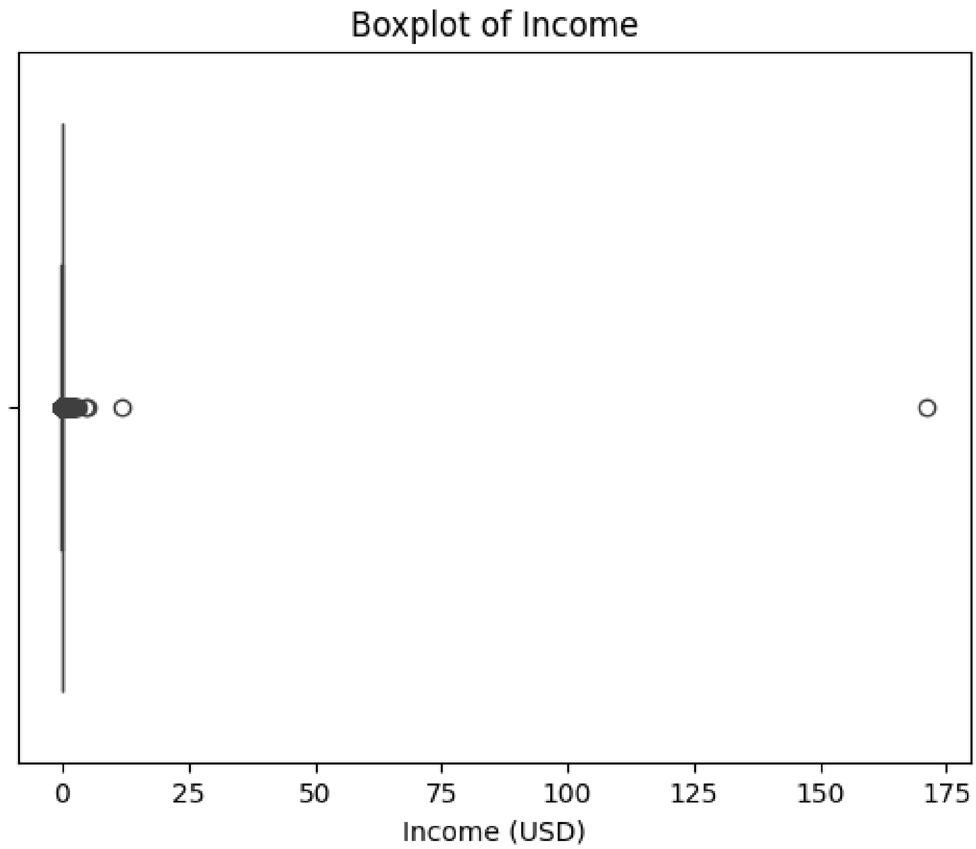
Correlation heatmap

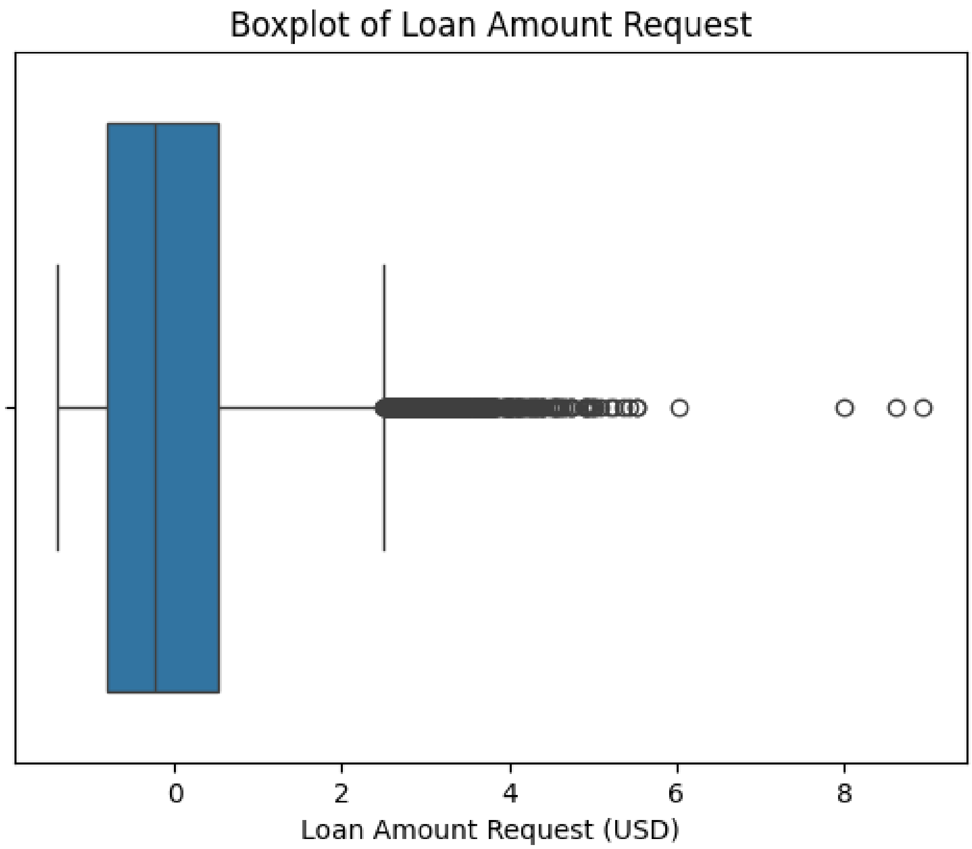
[19] :
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



BoxPlot

[20] :
```
# Boxplot for Income
sns.boxplot(x=df['Income (USD)'])
plt.title('Boxplot of Income')
plt.show()

# Boxplot for Loan Amount Request
sns.boxplot(x=df['Loan Amount Request (USD)'])
plt.title('Boxplot of Loan Amount Request')
plt.show()
```


Boxplot of Income
0
25
50
75
100
125
150
175
Income (USD)



[21]:
```
# Define target variable
target = 'Loan Sanction Amount (USD)'

# Define feature columns
X = df.drop(columns=[target])
y = df[target]

# Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

0.0.1 Grid Search

[23]:
```
# Define the parameter grid
param_grid = {
    'kernel': ['linear', 'rbf'],
    'C': [0.1, 1, 10],
    'gamma': ['scale', 0.1]
}
```

```python
# Split into train (80%) and test (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize SVR model
svr = SVR()

# Grid Search with 5-fold cross-validation
grid_search = GridSearchCV(
    estimator=svr,
    param_grid=param_grid,
    scoring='r2',
    cv=5,
    n_jobs=-1,
    verbose=0
)

# Fit on training set
grid_search.fit(X_train, y_train)

# Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best CV Score (R²):", grid_search.best_score_)
```

```
Best Parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Best CV Score (R²): 0.6543903124032083
```

[25]:

```python
# Define target variable
target = 'Loan Sanction Amount (USD)'

# Define feature columns
X = df.drop(columns=[target])
y = df[target]

# Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train SVR with best parameters on the training set
best_svr = grid_search.best_estimator_
best_svr.fit(X_train, y_train)

# Predictions on test set
y_pred = best_svr.predict(X_test)

# Evaluation metrics
```

```python
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nModel Evaluation on Test Set:")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"R² Score: {r2:.4f}")
```

```
Model Evaluation on Test Set:
Mean Squared Error (MSE): 552578763.6120
Mean Absolute Error (MAE): 11951.5667
R² Score: 0.6708
```

GitHub Link

10 Results Table

Table 1: Model Summary: Loan Amount Prediction

Field	Answer
Description Student's Result	Predicting sanctioned loan amount based on applicant's income, credit, and asset details.
Dataset Size (after preprocessing)	30000 records
Train/Test Split Ratio	80:20 (i.e., (test_size=0.2))
Feature(s) Used for Prediction	Age, Income Stability, Loan Amount Request (USD), Dependents, Credit Score, No. of Defaults, Has Active Credit Card, Property Location, Co-Applicant
Model Used	Linear Regression
Cross-Validation Used? (Yes/No)	Yes
If Yes, Number of Folds (K)	5
Reference to CV Results Table	See Table 2
Mean Absolute Error (MAE)	13803.42
Mean Squared Error (MSE)	527445271.77
Root Mean Squared Error (RMSE)	622966.18
R^2 Score	0.71
Adjusted R^2 Score	Not calculated
Most Influential Feature(s)	Loan amount request and Income Stability (features with higher positive coefficient)
Observations from Residual Plot	Residuals are fairly spread with slight underestimation for higher values
Interpretation of Predicted vs Actual Plot	Follows an upward trend, but deviations increase with higher amounts
Any Overfitting or Underfitting Observed?	Slight underfitting
Justification	Training and cross-validation scores are similar. There is no strong pattern, but large errors (residuals) on both ends suggest the model isn't fitting complex relationships well.

Table 2: Cross-Validation Results (K = 5)

Fold	MAE	MSE	RMSE	R^2 Score
Fold 1	13803.42	527445271.77	22966.18	0.69
Fold 2	13779.90	493351608.13	22211.52	0.70
Fold 3	14030.71	544801753.47	23340.99	0.67
Fold 4	14044.93	513654615.80	22663.95	0.70
Fold 5	13347.16	440761214.18	20994.31	0.73
Average	**13801.23**	**504002892.67**	**22435.39**	**0.70**

Best Practices

- Handle missing values carefully to avoid data loss
- Scale numeric features to improve model performance
- Split dataset into train/test/validation sets for fair evaluation
- Use visualization to interpret model behavior

Learning Outcomes

- Learned how to preprocess and clean data for ML
- Implemented Linear Regression using Scikit-learn
- Evaluated model performance using key metrics
- Visualized insights using Matplotlib