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

1 Aim

To develop and evaluate a machine learning model in Python for predicting loan amounts based on applicants' financial and credit-related attributes, employing Linear Regression and K-Fold Cross-Validation for model assessment.

2 Libraries Used

- NumPy
- Pandas
- Matplotlib
- Scikit-learn
- Seaborn

3 Objectives

- Preprocess the loan dataset by handling missing values, encoding categorical variables, selecting relevant features, and partitioning data into training, validation, and testing sets.
- Build and evaluate a Linear Regression model using K-Fold Cross-Validation; analyze metrics such as MSE, RMSE, MAE, and R^2; and interpret residuals and prediction plots to gauge model effectiveness.

4 Mathematical / Theoretical Background

4.1 Handling Missing Values

Missing values can degrade model performance by:

- Distorting statistical summaries,
- Causing errors during model training,
- Introducing bias in predictions.

Therefore, it is essential to detect and address missing data before modeling.

Common approaches include:

- Imputing missing values with mean, median, or mode using pandas' `fillna()` method.
- Dropping columns with excessive missing data that have limited relevance, to simplify the dataset and reduce noise.

4.2 Label Encoding

Machine learning models require numerical input; thus, categorical features (e.g., "Yes"/"No", "Graduate"/"Not Graduate") must be converted:

- Binary categories can be mapped directly to numeric codes (e.g., Yes = 1, No = 0), ensuring compatibility with most algorithms.
- For features with multiple categories, **one-hot encoding** is preferred to avoid imposing artificial ordinal relationships.

4.3 Plotting and Visualization

Visual tools help reveal data characteristics, such as correlations and outliers:

- **Heatmap:** Illustrates correlations between numeric features through color gradients, aiding feature selection.
- **Histogram:** Displays frequency distributions, helping identify skewness or gaps.
- **Boxplot:** Summarizes data distribution, highlights medians, quartiles, and outliers.

4.4 Standardization

- Standardization rescales features to zero mean and unit variance, useful when variables have different units or scales.
- Formula:
$$z = \frac{x - \mu}{\sigma}$$
where x is the original value, μ the mean, and σ the standard deviation.

This step enhances model convergence and performance.

5 Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df=pd.read_csv('train.csv')
df.head()
```

	Customer ID	Name	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	...	Credit Score	No. of Defaults	Has Active Credit Card	Pro
0	C-36995	Frederica Shealy	F	56	1933.05	Low	Working	Sales staff	Semi-Urban	72809.58	...	809.44	0	NaN	
1	C-33999	America Calderone	M	32	4952.91	Low	Working	NaN	Semi-Urban	46837.47	...	780.40	0	Unpossessed	
2	C-3770	Rosetta Verne	F	65	988.19	High	Pensioner	NaN	Semi-Urban	45593.04	...	833.15	0	Unpossessed	
3	C-26480	Zoe Chitty	F	65	NaN	High	Pensioner	NaN	Rural	80057.92	...	832.70	1	Unpossessed	
4	C-23459	Afton Venema	F	31	2614.77	Low	Working	High skill tech staff	Semi-Urban	113858.89	...	745.55	1	Active	

5 rows × 24 columns

```
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
```

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

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

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

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

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

```
# 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)
```

```
# 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)
```

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

```
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
```

```
#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])
```

```
#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'])
```

```
#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())
```

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

```
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

```
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

```
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])
```

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

	Gender	Age	Income (USD)	Income Stability	Profession	Location	Loan Amount Request (USD)	Current Loan Expenses (USD)	Expense Type 1	Expense Type 2	...	Credit Score	No. of Defaults	H Acti Cred Ca
0	-1.007092	0.991451	-0.061266	0.305833	0.834973	0.142149	-0.269027	-0.660358	-0.749241	-1.433524	...	0.992493	-0.490502	-2.0969
1	0.992958	-0.504355	0.229972	0.305833	0.834973	0.142149	-0.705269	0.392886	-0.749241	0.697582	...	0.578136	-0.490502	-1.0017
2	-1.007092	1.552379	-0.152389	-3.269763	-0.686548	0.142149	-0.726171	-0.946193	-0.749241	0.697582	...	1.330799	-0.490502	-1.0017
3	-1.007092	1.552379	-0.033357	-3.269763	-0.686548	-1.762481	-0.147279	-0.422775	-0.749241	0.697582	...	1.324379	2.038728	-1.0017
4	-1.007092	-0.566680	0.004480	0.305833	0.834973	0.142149	0.420461	0.374693	-0.749241	0.697582	...	0.080879	2.038728	1.1885
5	-1.007092	1.240752	-0.128594	0.305833	-0.306168	-1.762481	-0.913593	-0.906788	-0.749241	-1.433524	...	-0.795636	2.038728	0.0933
6	0.992958	0.181223	-0.019940	0.305833	0.834973	0.142149	1.070530	1.227526	1.334685	0.697582	...	-1.463830	-0.490502	-1.0017
7	-1.007092	0.305874	-0.033357	0.305833	-0.306168	0.142149	2.544436	1.682224	-0.749241	-1.433524	...	1.032730	-0.490502	1.1885
8	-1.007092	-0.130403	-0.122697	0.305833	0.834973	-1.762481	-0.901713	-1.012348	-0.749241	0.697582	...	-0.493571	2.038728	1.1885
9	0.992958	-1.376908	-0.098577	0.305833	0.834973	-1.762481	-0.784989	0.411038	-0.749241	-1.433524	...	-1.806987	-0.490502	-1.0017

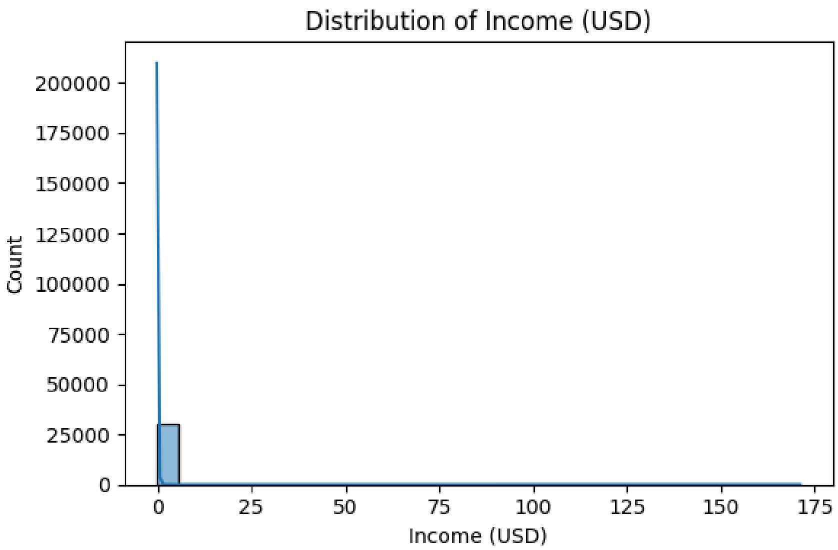
10 rows × 21 columns

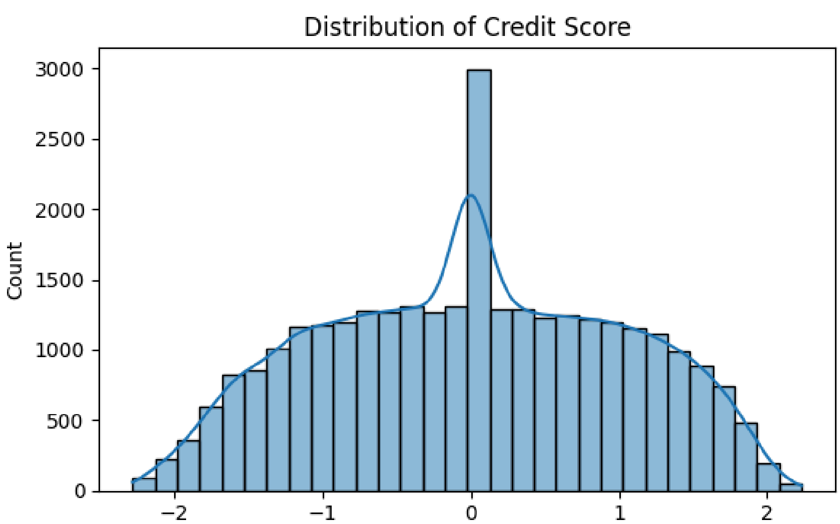
EDA

Histogram

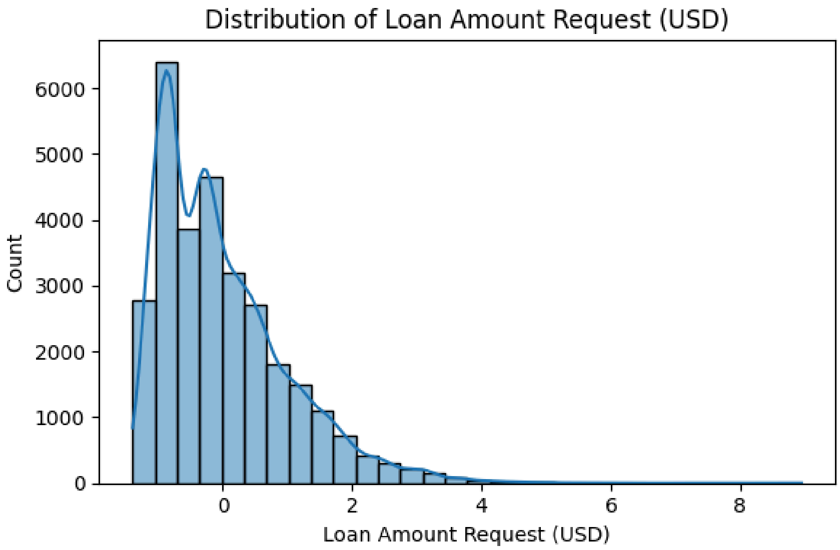
```
import seaborn as sns
import matplotlib.pyplot as plt

# 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))
```

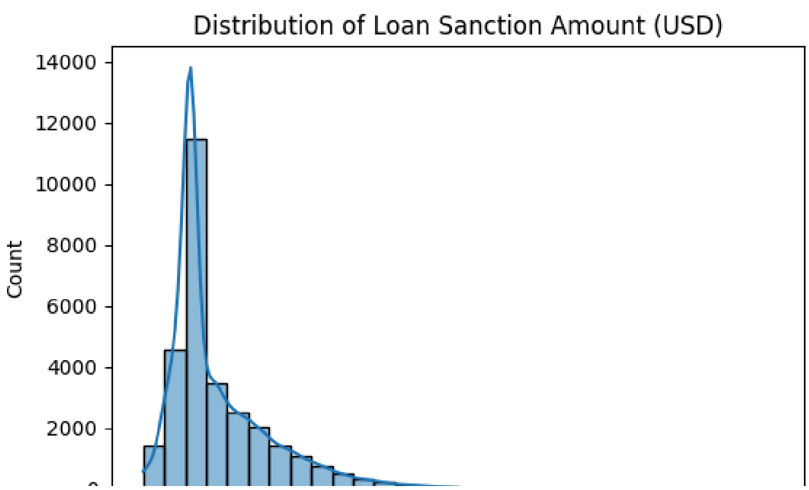
```
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')
    plt.tight_layout()
    plt.show()
```

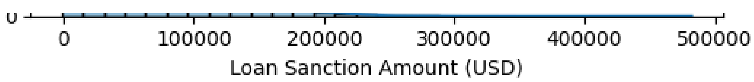

Distribution of Income (USD)
Count
200000
175000
150000
125000
100000
75000
50000
25000
0
0
25
50
75
100
125
150
175
Income (USD)


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

Credit Score

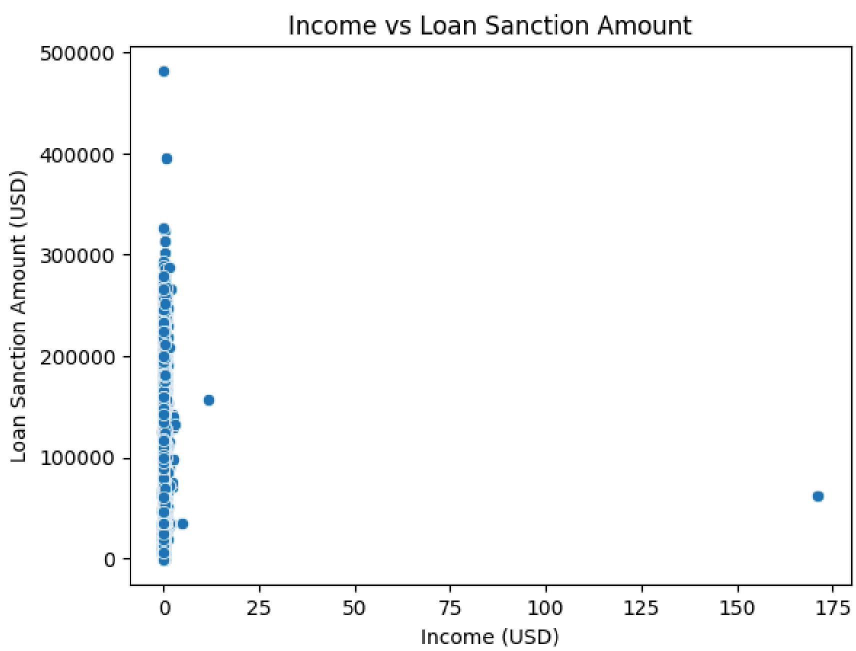


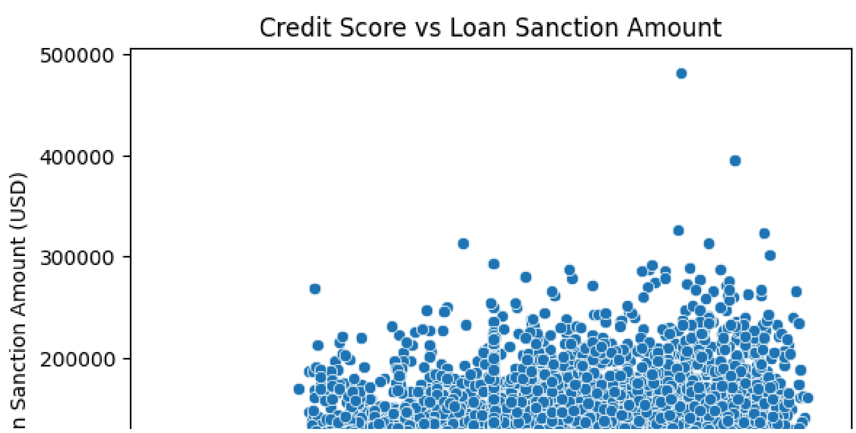


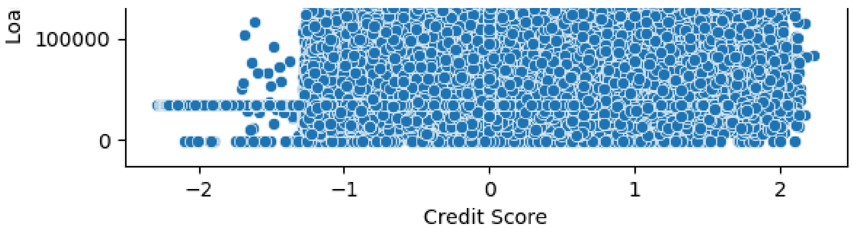


Scatter Plot

```
# 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()
```

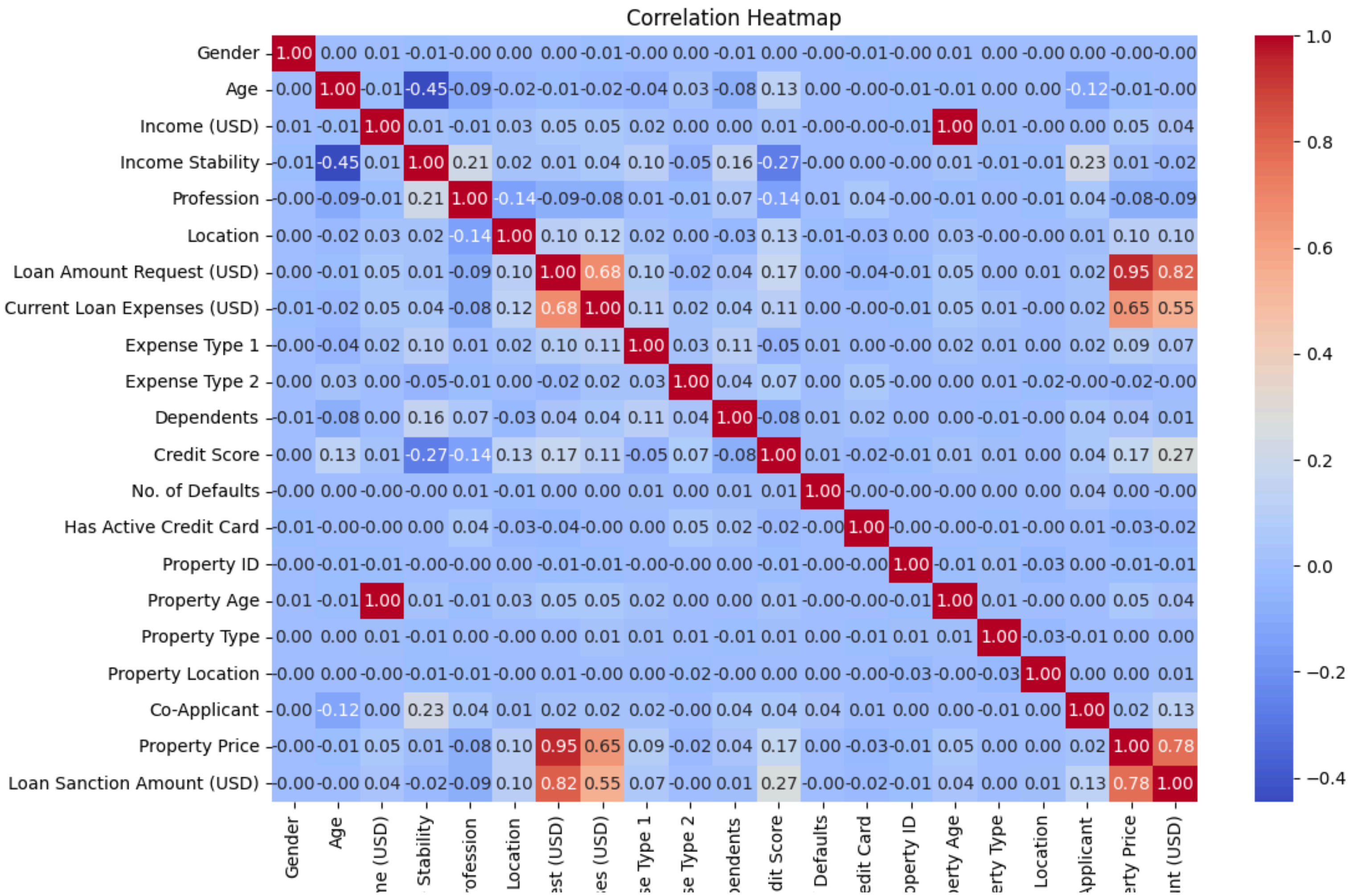

Income vs Loan Sanction Amount
500000
400000
300000
200000
100000
0
Loan Sanction Amount (USD)
0
25
50
75
100
125
150
175
Income (USD)


Credit Score vs Loan Sanction Amount
500000
400000
300000
200000
n Sanction Amount (USD)



Correlation heatmap

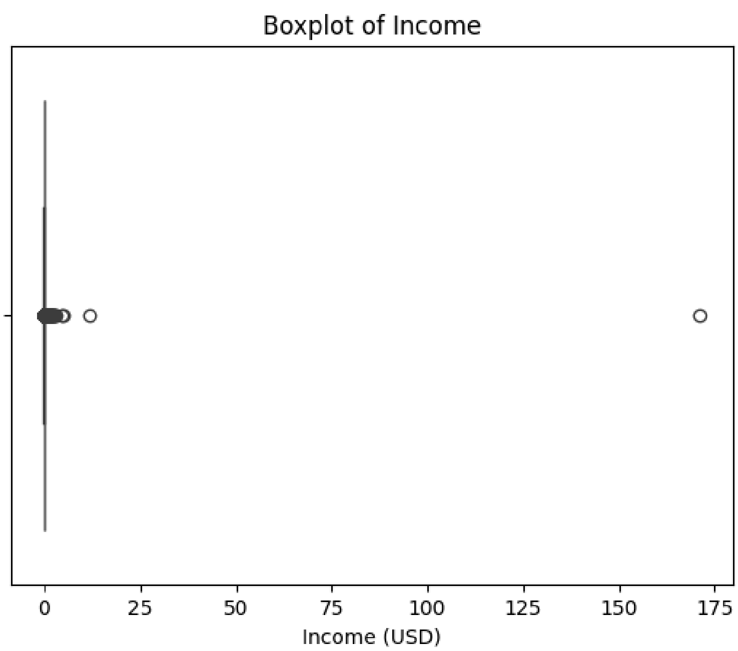
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

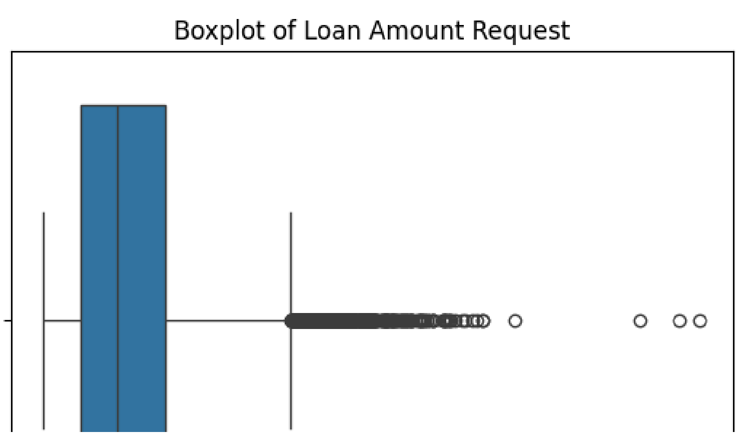

Correlation Heatmap
Gender
Age
Income (USD)
Income Stability
Profession
Location
Loan Amount Request (USD)
Current Loan Expenses (USD)
Expense Type 1
Expense Type 2
Dependents
Credit Score
No. of Defaults
Has Active Credit Card
Property ID
Property Age
Property Type
Property Location
Co-Applicant
Property Price
Loan Sanction Amount (USD)
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Double-click (or enter) to edit

BoxPlot

```
# 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)


Boxplot of Loan Amount Request

Train Test Split

```
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

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

# Initialize and train the model
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
```

```
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}:")
```

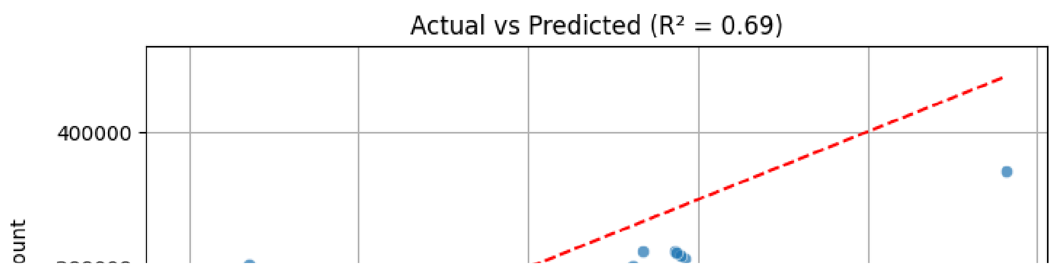
```python
    print(f"  MSE : {mse_scores[i]:.2f}")
    print(f"  RMSE: {rmse_scores[i]:.2f}")
    print(f"  MAE : {mae_scores[i]:.2f}")
    print(f"  R²  : {r2_scores[i]:.2f}")
    print()

# 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

```
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()
```



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]:
```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
```

[4]:
```python
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]:
```
#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]:
```
#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]:
```
#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]:
```
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
```

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

[16]:
```
df.head(10)
```

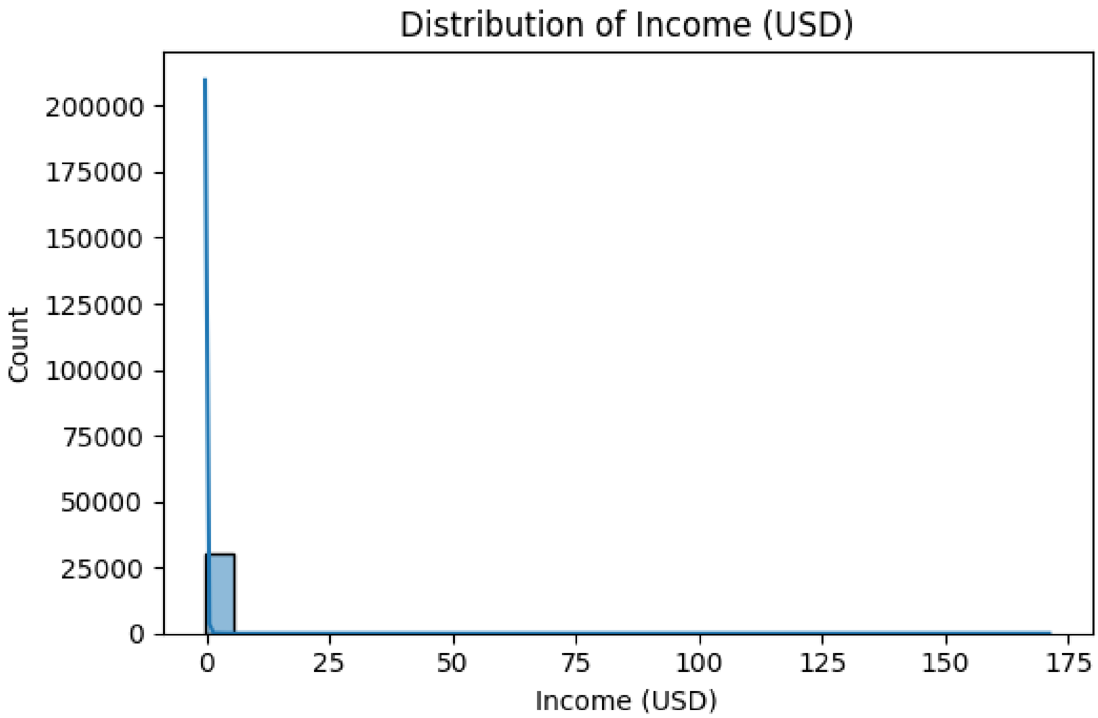
[16]:
```
     Gender       Age  Income (USD)  Income Stability  Profession  Location  \
0 -1.007092  0.991451     -0.061266          0.305833    0.834973  0.142149
1  0.992958 -0.504355      0.229972          0.305833    0.834973  0.142149
2 -1.007092  1.552379     -0.152389         -3.269763   -0.686548  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
```

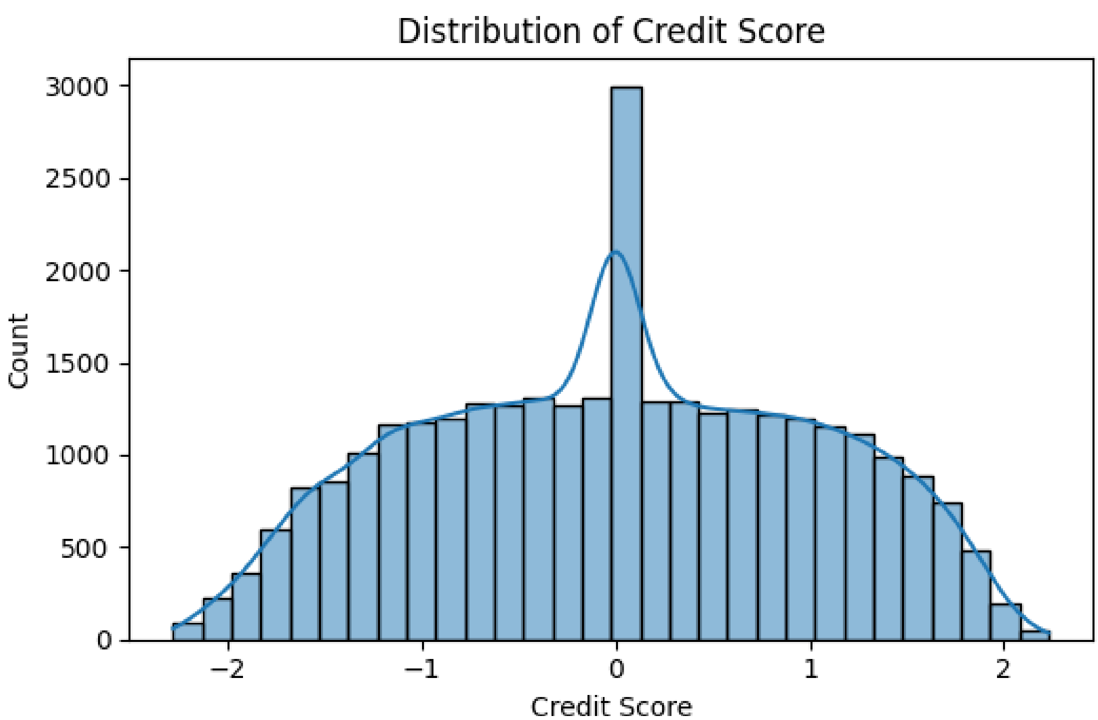
```
6   -0.954127   -0.019639    -1.305329          0.034344      0.419205
7   -0.652204   -0.032979    -0.411309          1.283229      0.419205
8   -0.905541   -0.122407     1.376731         -1.214540      0.419205
9    1.322440   -0.098284    -0.411309          1.283229      0.419205

   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]
```

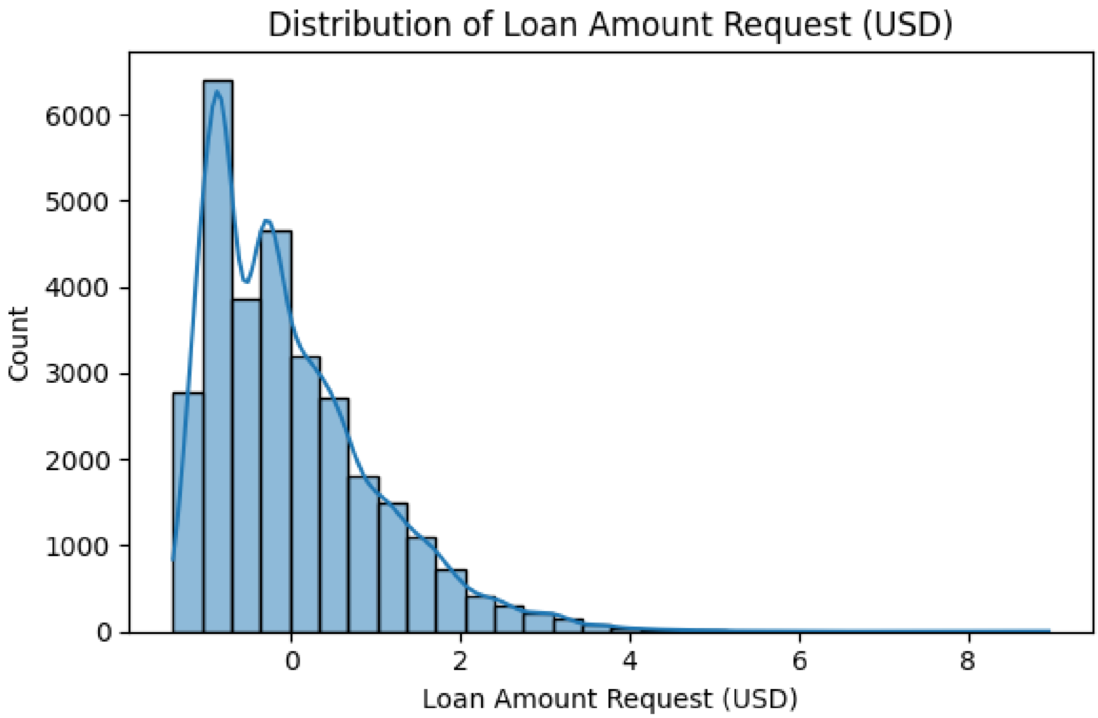
EDA

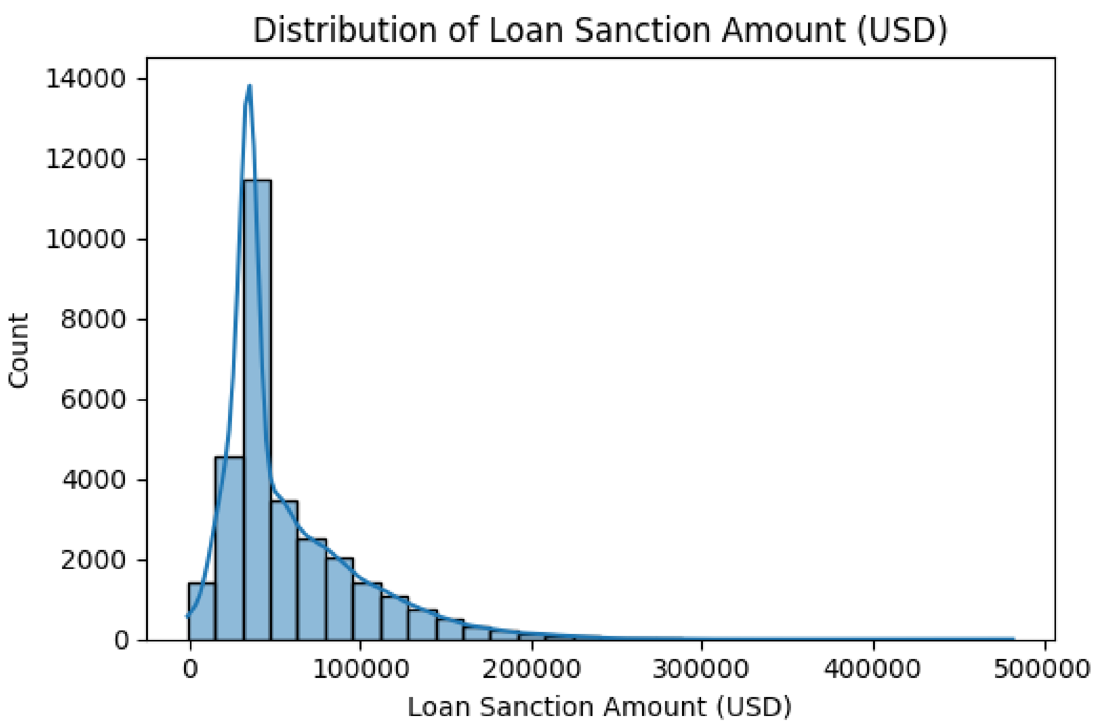
Histogram

[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)

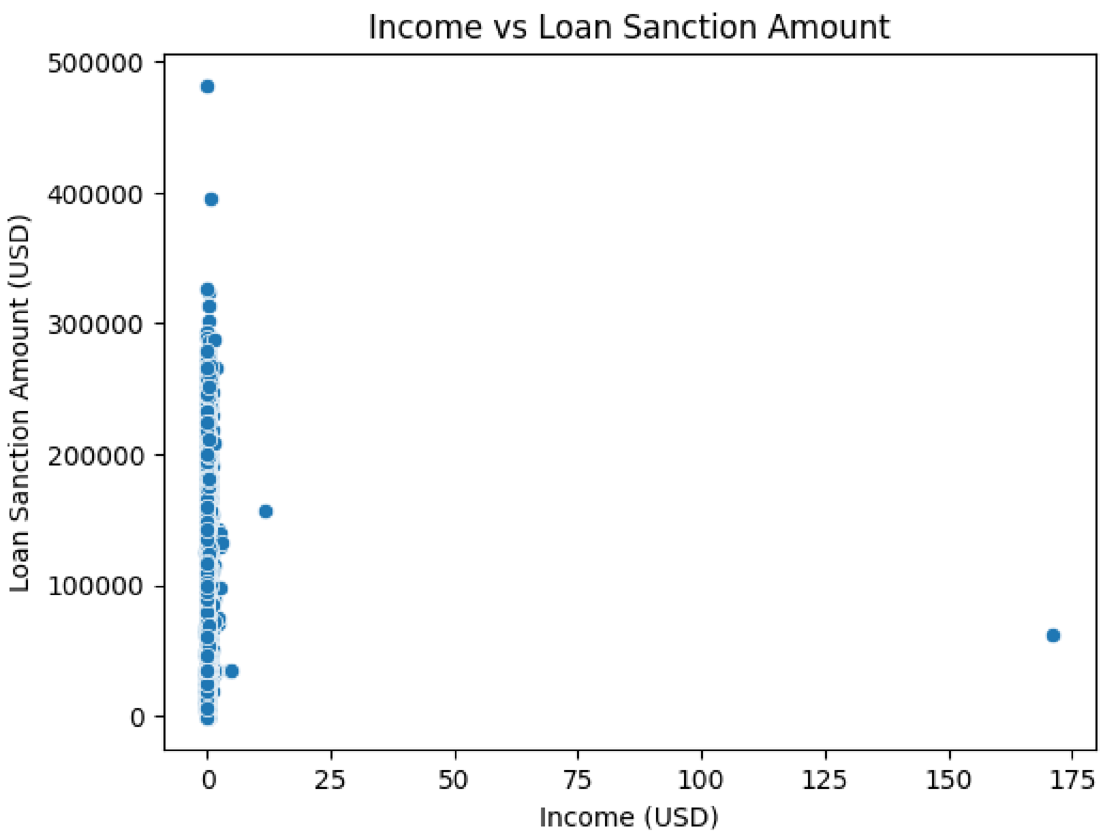

Distribution of Credit Score
Count
3000
2500
2000
1500
1000
500
0
−2
−1
0
1
2
Credit Score

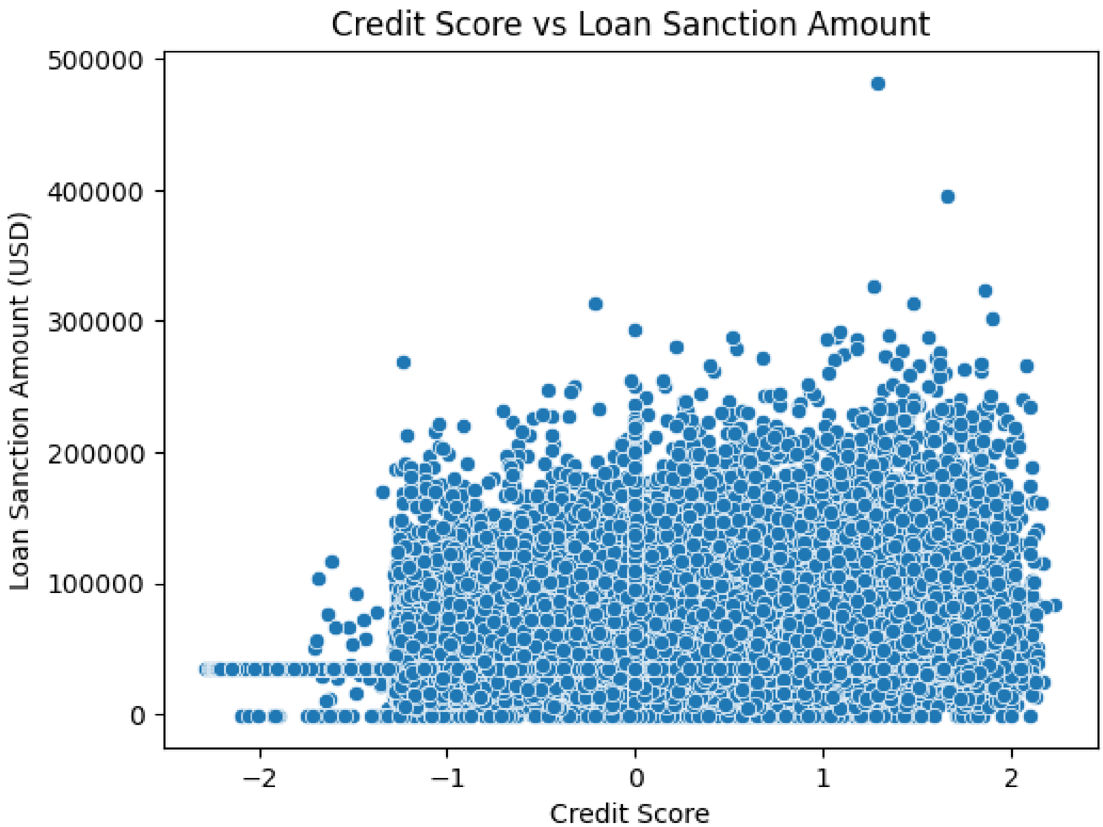

Distribution of Loan Amount Request (USD)
Count
Loan Amount Request (USD)
0
1000
2000
3000
4000
5000
6000
0
2
4
6
8


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

Scatter Plot

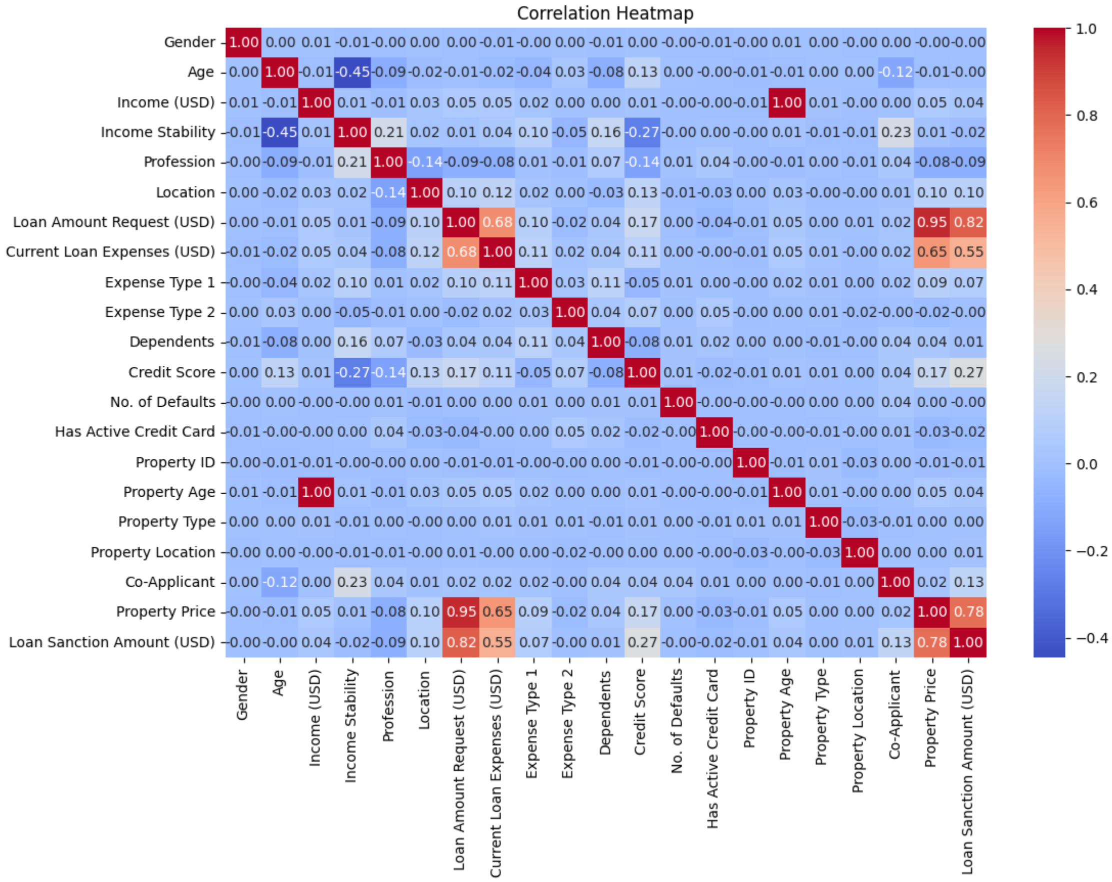
[18]:
```python
# Income vs Loan Sanction Amount
sns.scatterplot(data=df, x='Income (USD)', y='Loan Sanction Amount (USD)')
plt.title('Income vs Loan Sanction Amount')
plt.show()

# 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()
```





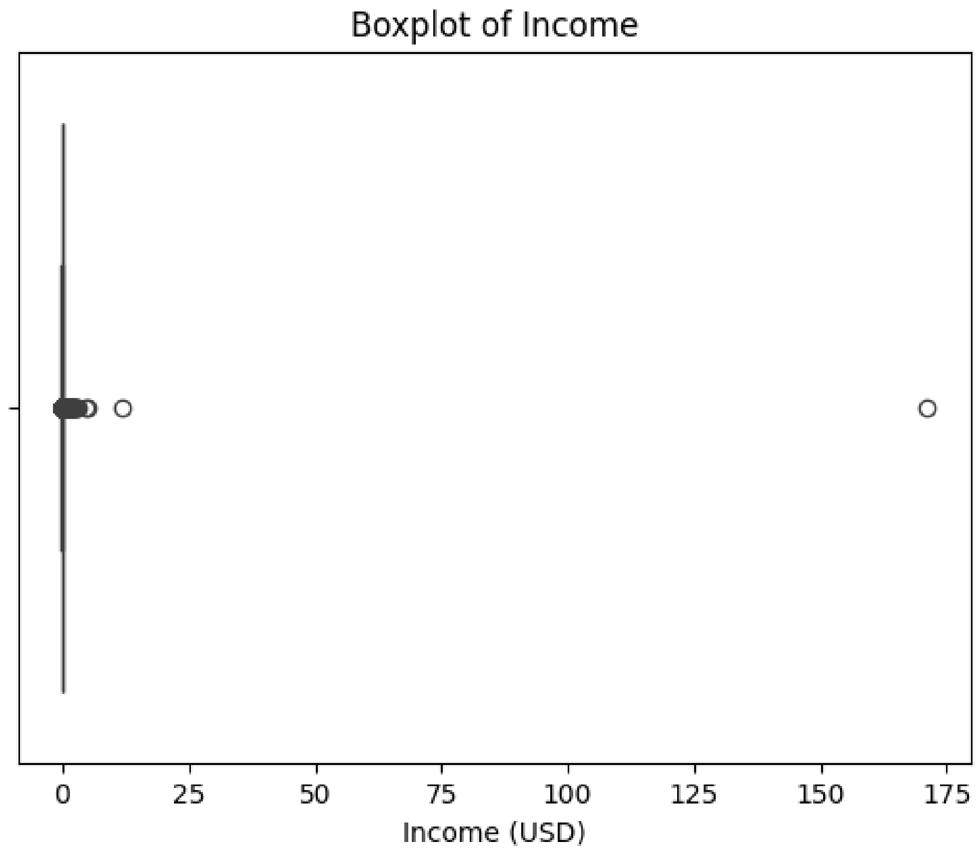
Correlation heatmap

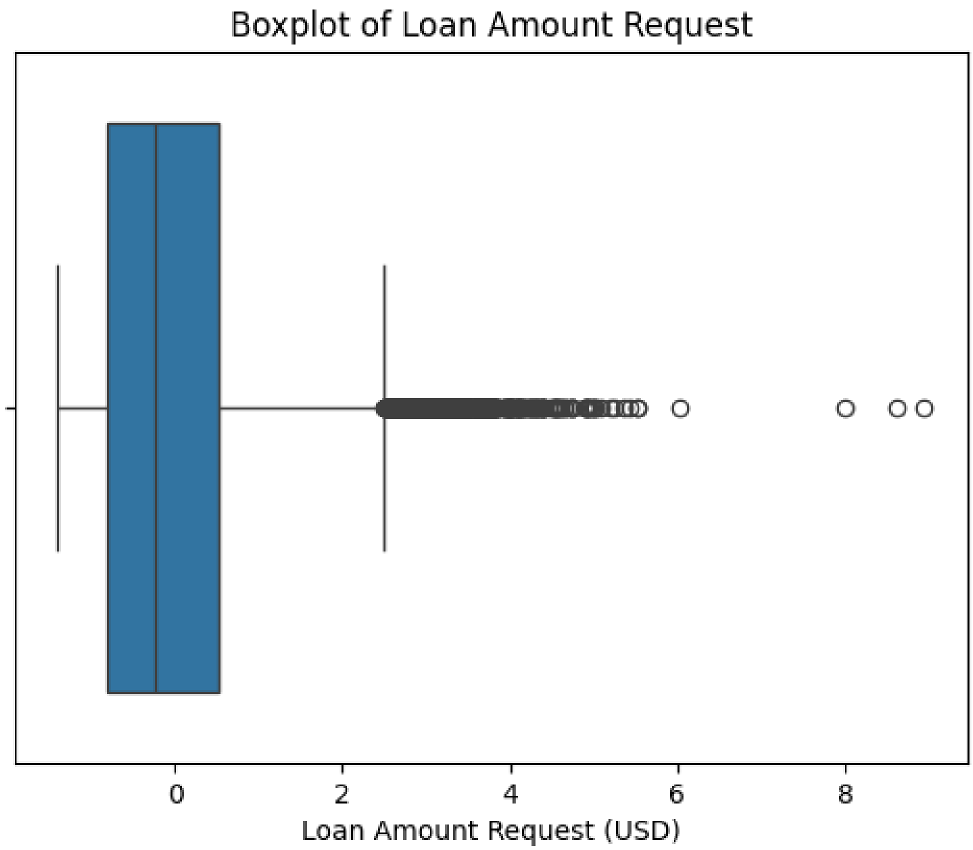
[19]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



BoxPlot

[20]:
```python
# 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]
}
```

```
# 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]:

```
# 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
```

6 Included Plots

- **Heatmap**: Shows feature correlations to identify strong linear relationships.
- **Boxplot**: Visualizes distribution, central tendency, and detects outliers.
- **Scatter Plot**: Displays relationships between pairs of numerical variables.
- **Histogram**: Illustrates the distribution of data points across bins.

7 Best Practices Followed

- **Consistent Preprocessing**: Applied uniform cleaning, encoding, and scaling of features to improve model generalization.
- **Robust Validation**: Employed 5-fold cross-validation to assess model stability across different data splits, minimizing overfitting risk.

8 Learning Outcomes

- **Comprehensive Pipeline Knowledge**: Gained practical experience with data exploration, preprocessing, model training, evaluation, and visualization.
- **Model Evaluation Insights**: Learned to interpret MAE, MSE, RMSE, and R^2 metrics, and to analyze residual plots for diagnosing model fit quality.

GitHub Repository: https://github.com/Thamizhmathibharathi/project.git

9 Results Table

Table 1: Model Summary: Loan Amount Prediction

Field	Details
Project Description	Predicting sanctioned loan amounts using applicant income, credit, and asset information.
Dataset Size (post-preprocessing)	30,000 records
Train/Test Split	80:20 (test_size=0.2)
Features Used	Age, Income Stability, Loan Amount Requested, Dependents, Credit Score, Number of Defaults, Active Credit Card Status, Property Location, Co-Applicant Status
Model	Linear Regression
Cross-Validation	Yes (5 folds)
Mean Absolute Error (MAE)	13,803.42
Mean Squared Error (MSE)	527,445,271.77
Root Mean Squared Error (RMSE)	622,966.18
R^2 **Score**	0.71
Adjusted R^2 **Score**	Not calculated
Key Influential Features	Loan Amount Requested and Income Stability (highest positive coefficients)
Residual Plot Observations	Residuals mostly evenly spread with slight underestimation at higher values
Predicted vs Actual Plot Interpretation	Shows overall upward trend; deviations increase with larger loan amounts
Overfitting / Underfitting	Minor underfitting observed
Rationale	Training and CV scores are close, indicating no severe overfitting. Residual distribution suggests model may miss some complex patterns.

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

Fold	MAE	MSE	RMSE	R^2 Score
Fold 1	13,803.42	527,445,271.77	22,966.18	0.69
Fold 2	13,779.90	493,351,608.13	22,211.52	0.70
Fold 3	14,030.71	544,801,753.47	23,340.99	0.67
Fold 4	14,044.93	513,654,615.80	22,663.95	0.70
Fold 5	13,347.16	440,761,214.18	20,994.31	0.73
Average	**13,801.23**	**504,002,892.67**	**22,435.39**	**0.70**