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 build and evaluate a Linear Regression model for predicting the loan sanction amount using customer and financial data, incorporating K-Fold Cross Validation and performance visualization techniques.

2. Libraries Used :

- **pandas** – Used for data manipulation and analysis in tabular form.
- **numpy** – Supports numerical computations and operations on arrays.
- **matplotlib** – Enables data visualization through plots and charts.
- **seaborn** – Used for statistical data visualization and enhancing matplotlib plots.
- **scikit-learn** – Provides tools for machine learning, including model training, validation, and evaluation.

3. Objective:

- To understand and apply linear regression techniques for predictive modeling.
- To evaluate model performance using metrics such as MAE, MSE, RMSE, and R^2 Score.
- To implement K-Fold Cross Validation for robust evaluation.
- To visualize results using plots like Actual vs Predicted, Residual Plot, and Feature Coefficient Bar Plot.

4. Mathematical Description:

Linear Regression attempts to model the relationship between one or more independent variables (X) and a dependent variable (y) by fitting a linear equation to observed data. The equation for a multiple linear regression model is given by:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

Where:

- y – Predicted value (Loan Sanction Amount)
- $x_1, x_2, ..., x_n$ – Input features (customer and financial data)
- β_0 – Intercept term
- $\beta_1, \beta_2, ..., \beta_n$ – Coefficients for each input feature
- ϵ – Error term

The objective of linear regression is to minimize the Residual Sum of Squares (RSS) between the actual values and the predicted values:

$$RSS = \sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

Where:

- y_i – Actual value
- $\hat{y}_i$ – Predicted value
- m – Number of samples

5. Code :

Data Loading

```
# 1. Importing Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Read the data
df = pd.read_csv('train.csv')
df
```

Output:

```
      Customer ID                Name Gender  Age  Income (USD)  \
0         C-36995   Frederica Shealy      F   56       1933.05
1         C-33999  America Calderone      M   32       4952.91
2          C-3770      Rosetta Verne      F   65        988.19
...           ...                ...    ...  ...           ...
29998     C-12172      Carolann Osby      M   38       2417.71
```

```
29999    C-33003  Bridget Garibaldi      F   63       3068.24

      Income Stability           Profession     Type of Employment  \
0                  Low              Working            Sales staff
1                  Low              Working                    NaN
2                 High            Pensioner                    NaN
...                ...                  ...                    ...
29998              Low              Working         Security staff
29999             High            Pensioner                    NaN

         Location  Loan Amount Request (USD)  ...  Credit Score  \
0      Semi-Urban                   72809.58  ...        809.44
1      Semi-Urban                   46837.47  ...        780.40
2      Semi-Urban                   45593.04  ...        833.15
...           ...                        ...  ...           ...
29998  Semi-Urban                  142524.10  ...        677.27
29999       Rural                  156290.54  ...        815.44

      No. of Defaults Has Active Credit Card  Property ID  Property Age  \
0                   0                    NaN          746       1933.05
1                   0            Unpossessed          608       4952.91
2                   0            Unpossessed          546        988.19
...               ...                    ...          ...           ...
29998               1            Unpossessed          375       2417.71
29999               0                 Active          344       3068.24

       Property Type Property Location  Co-Applicant  Property Price  \
0                  4             Rural             1       119933.46
1                  2             Rural             1        54791.00
2                  2             Urban             0        72440.58
...              ...               ...           ...             ...
29998              4             Urban             1       168194.47
29999              3             Rural             1       194512.60

       Loan Sanction Amount (USD)
0                        54607.18
1                        37469.98
2                        36474.43
...                           ...
29998                    99766.87
29999                   117217.90

[30000 rows x 24 columns]
```

Data Preprocessing

a. Handling Missing Values

```
df.isnull().sum()[df.isnull().sum() > 0]
```

Output:

```
Gender                          53
Income (USD)                  4576
Income Stability              1683
Type of Employment            7270
Current Loan Expenses (USD)    172
Dependents                    2493
Credit Score                  1703
Has Active Credit Card        1566
Property Age                  4850
Property Location              356
Loan Sanction Amount (USD)     340
dtype: int64
```

```
# Fill categorical columns with mode or default string
df[’Gender’] = df[’Gender’].fillna(df[’Gender’].mode()[0])
df[’Income Stability’] = df[’Income Stability’].fillna(df[’Income Stability’].mode()[0])
df[’Type of Employment’] = df[’Type of Employment’].fillna(’Unknown’)
df[’Has Active Credit Card’] = df[’Has Active Credit Card’].fillna(’Unknown’)
df[’Property Location’] = df[’Property Location’].fillna(df[’Property Location’].mode()[0])

# Fill numerical columns with mean/median
df[’Income (USD)’] = df[’Income (USD)’].fillna(df[’Income (USD)’].mean())
df[’Current Loan Expenses (USD)’] = df[’Current Loan Expenses (USD)’].fillna(df[’Current Loan
df[’Dependents’] = df[’Dependents’].fillna(df[’Dependents’].median())
df[’Dependents’] = df[’Dependents’].astype(int)
df[’Credit Score’] = df[’Credit Score’].fillna(df[’Credit Score’].mean())
df[’Property Age’] = df[’Property Age’].fillna(df[’Property Age’].mean())

# Drop rows where target is missing
df = df.dropna(subset=[’Loan Sanction Amount (USD)’])
```

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

Output:

```
Customer ID                    0
Name                           0
Gender                         0
Age                            0
Income (USD)                   0
Income Stability               0
Profession                     0
```

```
Type of Employment            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
```

```
print("Data shape after handling null values:", df.shape)
```

Output:

```
Data shape after handling null values: (29660, 24)
```

b. Handling Outliers

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

num_cols = df.select_dtypes(include=['int64', 'float64']).columns
cols = 4
rows = 4

plt.figure(figsize=(6 * cols, 4 * rows))

for i, col in enumerate(num_cols):
    plt.subplot(rows, cols, i + 1)
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.xlabel(col)

plt.tight_layout()
plt.show()
```

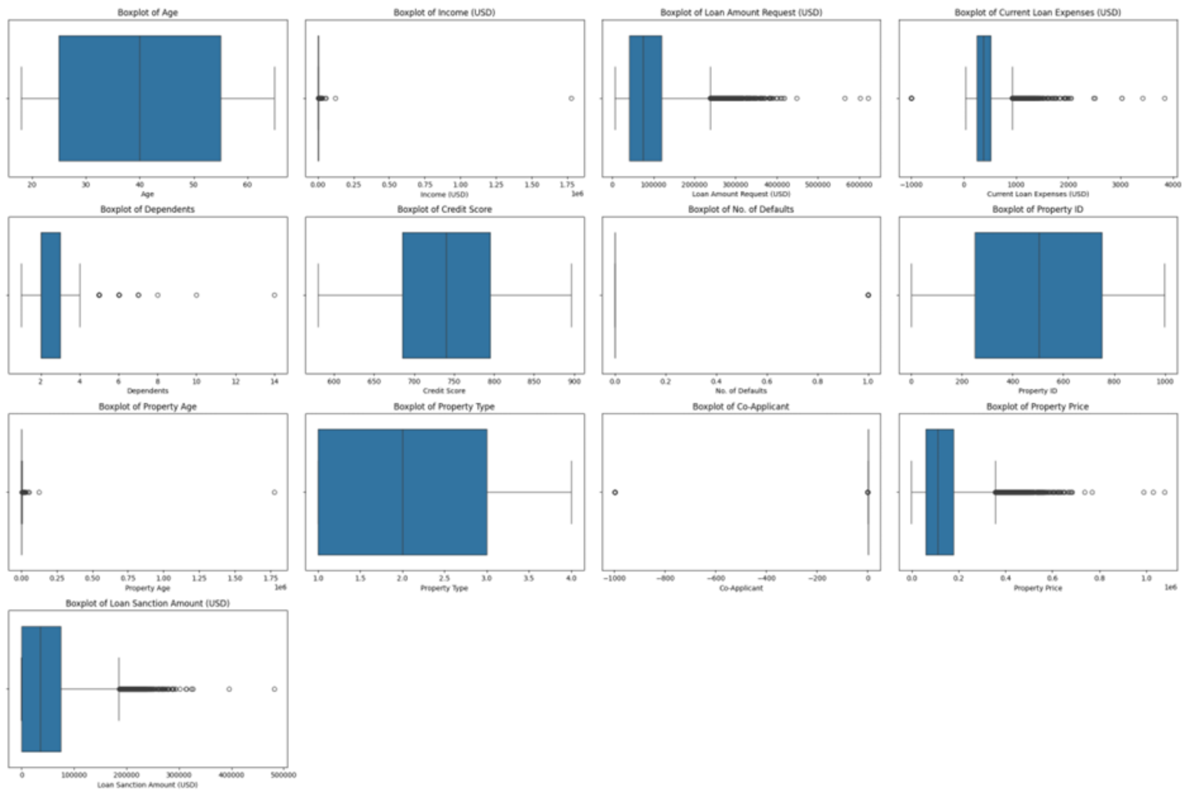

Figure 1: Boxplot of numerical features

```
def cap_outliers(df, col, lower_percentile=0.05, upper_percentile=0.95):
    lower = df[col].quantile(lower_percentile)
    upper = df[col].quantile(upper_percentile)
    df.loc[df[col] < lower, col] = lower
    df.loc[df[col] > upper, col] = upper
    return df


# Apply to all numerical columns
num_cols = df.select_dtypes(include=['int64', 'float64']).columns

for col in num_cols:
    df = cap_outliers(df, col)
print("Capping of outliers done")
```

Output:

```
Capping of outliers done
```

c. Encoding Categorical variables

```
from sklearn.preprocessing import LabelEncoder
```

```
cat_cols = df.select_dtypes(include='object').columns.tolist()
cat_cols = [col for col in cat_cols if col not in ['Customer ID', 'Name']]

le = LabelEncoder()

# Apply label encoding to each categorical column
for col in cat_cols:
    df.loc[:, col] = le.fit_transform(df[col].astype(str))

print("After encoding all categorical variables")
df
```

Output:

```
After encoding all categorical variables

      Customer ID               Name Gender  Age  Income (USD)  \
0         C-36995   Frederica Shealy      0   56   1933.050000
1         C-33999  America Calderone      1   32   4867.821000
2          C-3770      Rosetta Verne      0   64   1065.603000
...           ...                ...    ...  ...           ...
29998     C-12172      Carolann Osby      1   38   2417.710000
29999     C-33003  Bridget Garibaldi      0   63   3068.240000

      Income Stability Profession Type of Employment Location  \
0                    1          7                 14        1
1                    1          7                 17        1
2                    0          3                 17        1
...                ...        ...                ...      ...
29998                1          7                 16        1
29999                0          3                 17        0

       Loan Amount Request (USD)  ...  Credit Score No. of Defaults  \
0                       72809.58  ...    809.440000               0
1                       46837.47  ...    780.400000               0
2                       45593.04  ...    833.150000               0
...                          ...  ...           ...             ...
29998                  142524.10  ...    677.270000               1
29999                  156290.54  ...    815.440000               0

      Has Active Credit Card  Property ID  Property Age  Property Type  \
0                          2          746    1933.05000              4
1                          3          608    4855.43250              2
2                          3          546    1074.09800              2
...                      ...          ...           ...            ...
29998                      3          375    2417.71000              4
29999                      0          344    3068.24000              3
```

```
      Property Location  Co-Applicant  Property Price  \
0                     0             1       119933.46
1                     0             1        54791.00
2                     2             0        72440.58

...                 ...               ...             ...               ...
29998                 2             1       168194.47
29999                 0             1       194512.60

       Loan Sanction Amount (USD)
0                        54607.18
1                        37469.98
2                        36474.43
...                           ...
29998                    99766.87
29999                   117217.90

[29660 rows x 24 columns]
```

d. Standardize the features

```
from sklearn.preprocessing import StandardScaler
df = df.copy()
target_col = 'Loan Sanction Amount (USD)'
num_cols = df.select_dtypes(include=['int64', 'float64']).columns.drop(target_col)
df.loc[:, num_cols] = df[num_cols].astype(float)

# Initialize and apply StandardScaler
scaler = StandardScaler()
df.loc[:, num_cols] = scaler.fit_transform(df[num_cols])
print("Standardization done")
```

Output:

```
Standardization done
```

Exploratory Data Analysis

a. Distribution plots

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

num_cols = df.select_dtypes(include=['int64', 'float64']).columns
cols = 4
rows = 4

plt.figure(figsize=(6 * cols, 4 * rows))
```

```
for i, col in enumerate(num_cols):
    plt.subplot(rows, cols, i + 1)
    sns.histplot(df[col], kde=True, bins=30, color='skyblue')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')

plt.tight_layout()
plt.show()
```

Output:

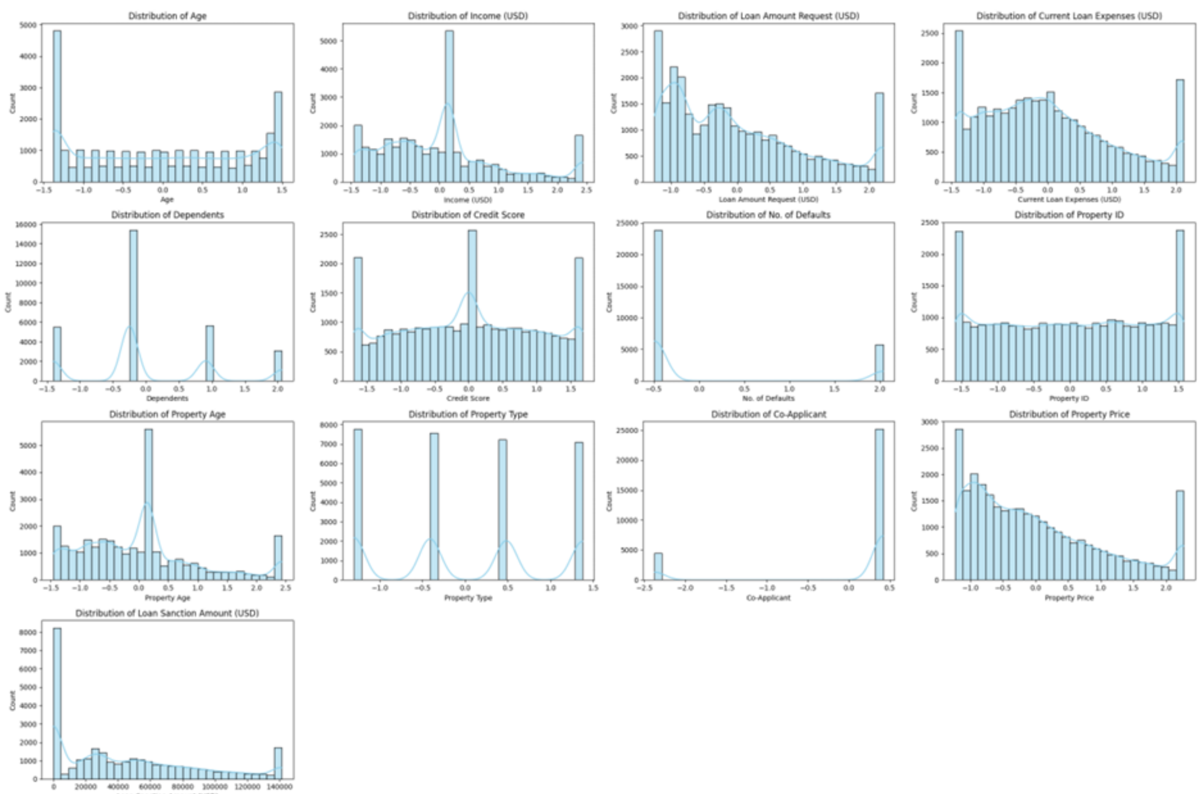

Figure 2: Distribution of features

b. Scatter plots

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

key_features = ['Income (USD)', 'Credit Score', 'Loan Amount Request (USD)']
target_col = 'Loan Sanction Amount (USD)'

plt.figure(figsize=(6 * len(key_features), 5))
```

```
for i, col in enumerate(key_features):
    plt.subplot(1, len(key_features), i + 1)
    sns.scatterplot(x=df[col], y=df[target_col], color='mediumseagreen')
    plt.title(f'{col} vs {target_col}')
    plt.xlabel(col)
    plt.ylabel('Loan Amount')

plt.tight_layout()
plt.show()
```

Output:

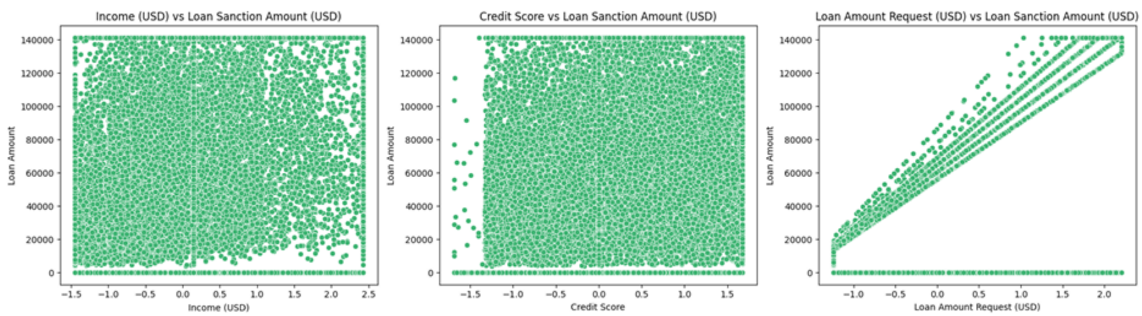

Figure 3: Scatter plots

c. Correlation Heatmap

```
original_cat_cols = ['Gender', 'Income Stability', 'Profession', 'Type of Employment',
                     'Location', 'Expense Type 1', 'Expense Type 2',
                     'Has Active Credit Card', 'Property Location']  # exclude 'Name', 'Custome
num_cols = df.select_dtypes(include=['int64', 'float64']).columns
reliable_num_cols = [col for col in num_cols if col not in original_cat_cols]

# Compute and plot correlation heatmap
corr_matrix = df[reliable_num_cols].corr()

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True, linewidths=0.5)
plt.title('Correlation Heatmap (Only Actual Numeric Features)', fontsize=16)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Output:

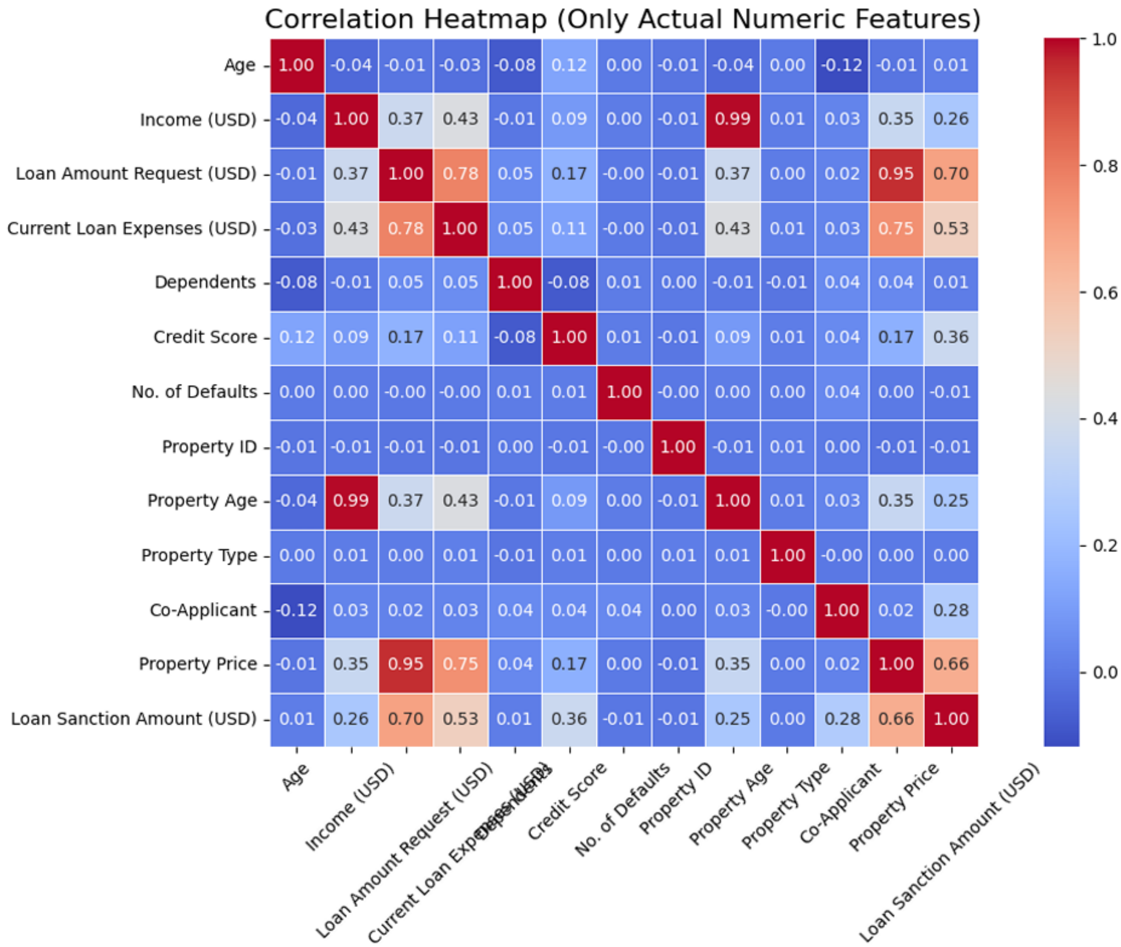

Figure 4: Correlation heatmap of features

Splitting the Dataset

```
from sklearn.model_selection import train_test_split

# Separate features and target
df = df.drop(['Customer ID', 'Name'], axis=1)
X = df.drop('Loan Sanction Amount (USD)', axis=1)
y = df['Loan Sanction Amount (USD)']

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.30, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.50, random_state=4

print("Train set:", X_train.shape)
print("Validation set:", X_val.shape)
print("Test set:", X_test.shape)
```

Output:

```
Train set: (20762, 21)
Validation set: (4449, 21)
Test set: (4449, 21)
```

Model training and performance metrics

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import pandas as pd

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

mae_list, mse_list, rmse_list, r2_list = [], [], [], []
fold = 1
for train_index, val_index in kf.split(X_train):
    X_tr, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_tr, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

    model.fit(X_tr, y_tr)
    y_pred = model.predict(X_val_fold)

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

    mae_list.append(mae)
    mse_list.append(mse)
    rmse_list.append(rmse)
    r2_list.append(r2)

    print(f"Fold {fold}: MAE={mae:.2f}, MSE={mse:.2f}, RMSE={rmse:.2f}, R²={r2:.2f}")
    fold += 1

cv_results = pd.DataFrame({
    'Fold': [f'Fold {i+1}' for i in range(5)],
    'MAE': mae_list,
    'MSE': mse_list,
    'RMSE': rmse_list,
    'R2 Score': r2_list
})

cv_results.loc['Average'] = cv_results[['MAE', 'MSE', 'RMSE', 'R2 Score']].mean().round(2)
```

```
cv_results.loc['Average', 'Fold'] = 'Average'
print("\nCross-Validation Results (K = 5):")
print(cv_results)
```

Output:

```
Fold 1: MAE=18890.63, MSE=702476122.64, RMSE=26504.27, R²=0.64
Fold 2: MAE=19311.32, MSE=777267591.82, RMSE=27879.52, R²=0.59
Fold 3: MAE=18732.84, MSE=696852179.02, RMSE=26397.96, R²=0.63
Fold 4: MAE=18338.03, MSE=674262768.19, RMSE=25966.57, R²=0.64
Fold 5: MAE=19186.81, MSE=768993353.76, RMSE=27730.73, R²=0.59

Cross-Validation Results (K = 5):
            Fold           MAE           MSE          RMSE  R2 Score
0         Fold 1  18890.631980  7.024761e+08  26504.266122  0.636256
1         Fold 2  19311.322104  7.772676e+08  27879.519218  0.587975
2         Fold 3  18732.837159  6.968522e+08  26397.957857  0.625975
3         Fold 4  18338.026766  6.742628e+08  25966.570205  0.640626
4         Fold 5  19186.811735  7.689934e+08  27730.729413  0.593704
Average  Average  18891.930000  7.239704e+08  26895.810000  0.620000
```

```
model = LinearRegression()
model.fit(X_train, y_train)
y_val_pred = model.predict(X_val)

mae_val = mean_absolute_error(y_val, y_val_pred)
mse_val = mean_squared_error(y_val, y_val_pred)
rmse_val = np.sqrt(mse_val)
r2_val = r2_score(y_val, y_val_pred)

print("Validation Set Performance:")
print(f"MAE: {mae_val:.2f}, MSE: {mse_val:.2f}, RMSE: {rmse_val:.2f}, R²: {r2_val:.2f}")
```

Output:

```
Validation Set Performance:
MAE: 18926.46, MSE: 731186370.50, RMSE: 27040.46, R²: 0.61
```

```
y_test_pred = model.predict(X_test)

mae_test = mean_absolute_error(y_test, y_test_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)
r2_test = r2_score(y_test, y_test_pred)

print("Test Set Performance:")
print(f"MAE: {mae_test:.2f}, MSE: {mse_test:.2f}, RMSE: {rmse_test:.2f}, R²: {r2_test:.2f}")
```

Output:

```
Test Set Performance:
MAE: 19022.77, MSE: 743473629.75, RMSE: 27266.71, R²: 0.60
```

Other Models Training and Evaluation

Train-Validation-Test Split

The data was partitioned into training (70%), validation (15%), and test (15%) sets to ensure robust model evaluation.

```
[language=Python, caption=Data Splitting]
X = df.drop('Loan Sanction Amount (USD)', axis=1)
Y = df['Loan Sanction Amount (USD)']
X_train, X_temp, y_train, y_temp = train_test_split(X, Y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

```
Train set: (20762, 21)
Validation set: (4449, 21)
Test set: (4449, 21)
```

Model Evaluation Results

The following models were trained, tuned, and evaluated.

```
def evaluate_model(name, model, X_train, X_test, param_grid=None, param_dist=None):
    print(f"\n{name} - Hyperparameter Tuning Started")

    # GRIDSEARCHCV
    if param_grid:
        grid_search = GridSearchCV(
            estimator=model,
            param_grid=param_grid,
            cv=5,
            scoring='r2',  # Change to 'accuracy' if classification
            n_jobs=-1
        )
        grid_search.fit(X_train, y_train)
        print(f"Best Params (GridSearchCV): {grid_search.best_params_}")
        print(f"Best CV Score (GridSearchCV): {grid_search.best_score_:.4f}")
    else:
        grid_search = None

    # RANDOMIZEDSEARCHCV
    if param_dist:
        random_search = RandomizedSearchCV(
            estimator=model,
            param_distributions=param_dist,
            n_iter=10,
            cv=5,
            scoring='r2',
            random_state=42,
            n_jobs=-1
```

```
    )
    random_search.fit(X_train, y_train)
    print(f"Best Params (RandomizedSearchCV): {random_search.best_params_}")
    print(f"Best CV Score (RandomizedSearchCV): {random_search.best_score_:.4f}")
else:
    random_search = None

# PICK BEST MODEL
if grid_search and random_search:
    best_model = (
        grid_search if grid_search.best_score_ >= random_search.best_score_
        else random_search
    ).best_estimator_
elif grid_search:
    best_model = grid_search.best_estimator_
elif random_search:
    best_model = random_search.best_estimator_
else:
    best_model = model

# EVALUATE BEST MODEL
start_time = time.time()
best_model.fit(X_train, y_train)
end_time = time.time()
y_pred = best_model.predict(X_test)

print(f"\n{name} Performance:")
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
adj_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) - X_test.shape[1] - 1)

print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"R²: {r2:.4f}")
print(f"Adjusted R²: {adj_r2:.4f}")
print(f"Training Time: {(end_time - start_time):.4f} seconds")

# ACTUAL vs PREDICTED
plt.figure(figsize=(14, 5))
plt.subplot(1,3,1)
plt.scatter(y_test, y_pred, alpha=0.6, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
         color='red', linestyle='--', linewidth=2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

```
    plt.title('Actual vs Predicted')
    plt.grid(True)

    # RESIDUAL PLOT
    residuals = y_test - y_pred
    plt.subplot(1,3,2)
    plt.scatter(y_pred, residuals, alpha=0.6, color='orange')
    plt.axhline(y=0, color='red', linestyle='--', linewidth=2)
    plt.xlabel('Predicted')
    plt.ylabel('Residuals')
    plt.title('Residual Plot')
    plt.grid(True)

    # Bar Plot of Feature Coefficients (for linear models)

    if hasattr(best_model, 'coef_') and best_model.coef_.ndim == 1:
        feature_names = X_train.columns
        coefficients = best_model.coef_
        coef_df = pd.DataFrame({
            'Feature': feature_names,
            'Coefficient': coefficients
        }).sort_values(by='Coefficient', ascending=False)

        plt.subplot(1,3,3)
        plt.barh(coef_df['Feature'], coef_df['Coefficient'], color='green')
        plt.xlabel('Coefficient Value')
        plt.title('Feature Coefficients')
        plt.gca().invert_yaxis()
        plt.grid(True, linestyle='--', alpha=0.5)
        plt.tight_layout()
    plt.show()
```

Linear Regression

```
Linear Regression Performance:
MAE: 18956.32
MSE: 724110330.15
RMSE: 26909.30
R-squared: 0.6125
Adjusted R-squared: 0.6107
```

Figure 5: Actual vs. Predicted, Residual Plot, Feature Importance

Ridge Regression

```
Best Params (GridSearchCV): {'alpha': 100.0}
Best CV Score (GridSearchCV): 0.6133

Ridge Regression Performance:
MAE: 18958.01
MSE: 724105842.33
RMSE: 26909.21
R-squared: 0.6125
Adjusted R-squared: 0.6107
```

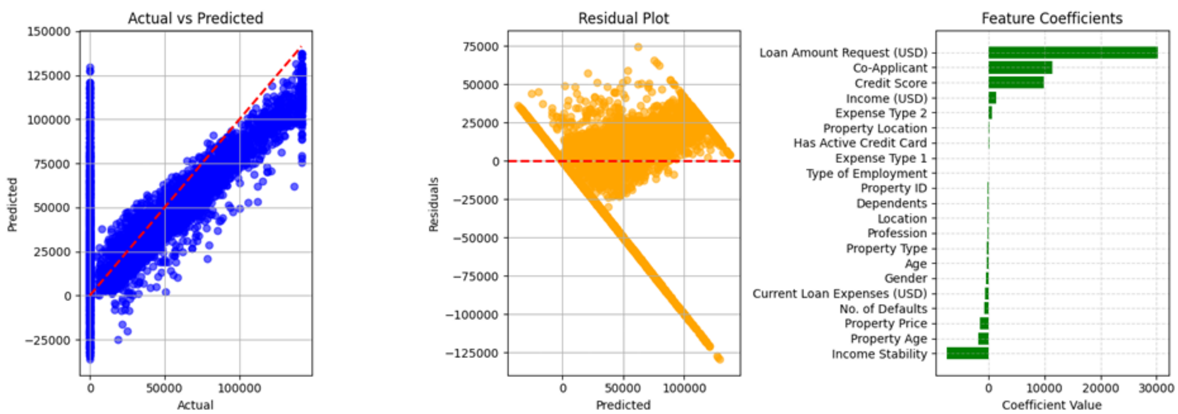

Figure 6: Actual vs. Predicted, Residual Plot, Feature Importance

Lasso Regression

```
Best Params (GridSearchCV): {'alpha': 10.0}
Best CV Score (GridSearchCV): 0.6133

Lasso Regression Performance:
MAE: 18956.34
MSE: 724110125.79
RMSE: 26909.29
R-squared: 0.6125
Adjusted R-squared: 0.6107
```

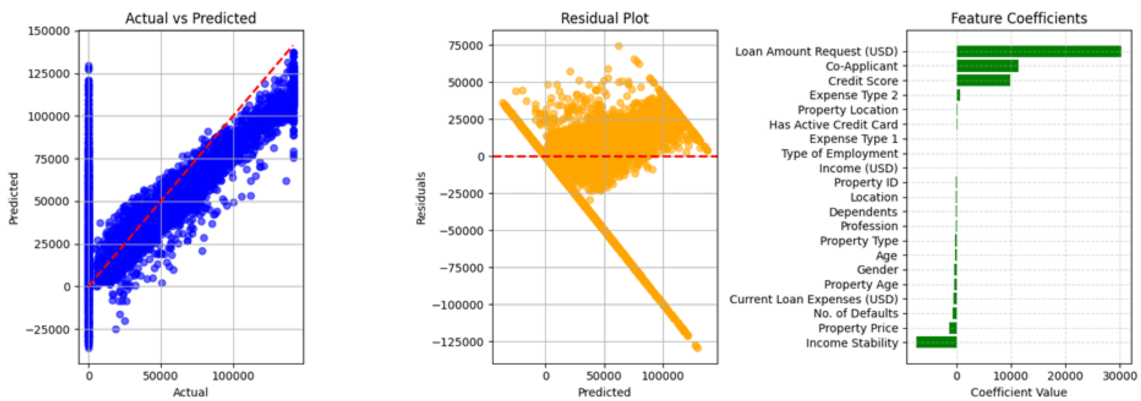

Figure 7: Actual vs. Predicted, Residual Plot, Feature Importance

Elastic Net Regression

```
Best Params (GridSearchCV): {'alpha': 0.01, 'l1_ratio': 0.9}
Best CV Score (GridSearchCV): 0.6165

ElasticNet Regression Performance:
MAE: 19024.7809
MSE: 743299083.0087
RMSE: 27263.5119
R²: 0.5994
Adjusted R-squared: 0.5975
```

Figure 8: Actual vs. Predicted, Residual Plot, Feature Importance

Polynomial Regression

```
Polynomial Regression (Degree 2) Performance:
MAE: 15352.02
MSE: 626442733.75
RMSE: 25028.84
R-squared: 0.6624
Adjusted R-squared: 0.6421
```

Decision Tree Regressor

```
Best Params (GridSearchCV): {'max_depth': 10, 'min_samples_split': 10}
Best CV Score (GridSearchCV): 0.9250

Decision Tree Regressor Performance:
MAE: 1205.11
MSE: 120883214.55
RMSE: 10994.69
R-squared: 0.9356
Adjusted R-squared: 0.9352
```

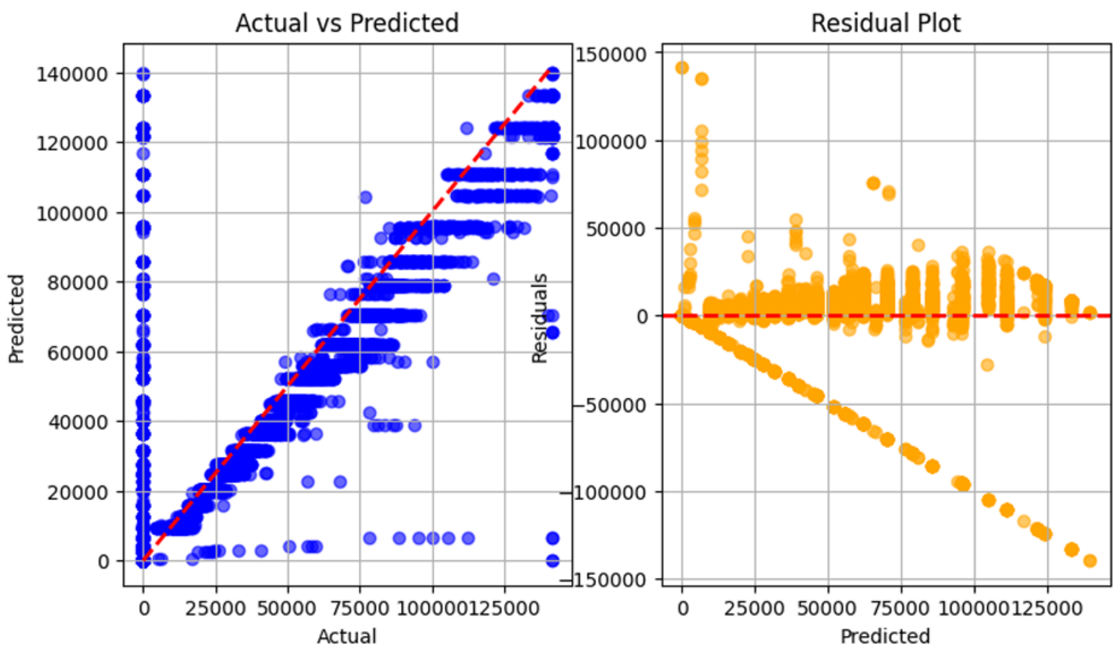

Figure 9: Actual vs. Predicted Residual Plot

Random Forest Regressor

```
Best Params (GridSearchCV): {'max_depth': None, 'n_estimators': 200}
Best CV Score (GridSearchCV): 0.9512

Random Forest Regressor Performance:
MAE: 928.32
MSE: 92147321.11
RMSE: 9599.34
R-squared: 0.9512
Adjusted R-squared: 0.9509
```

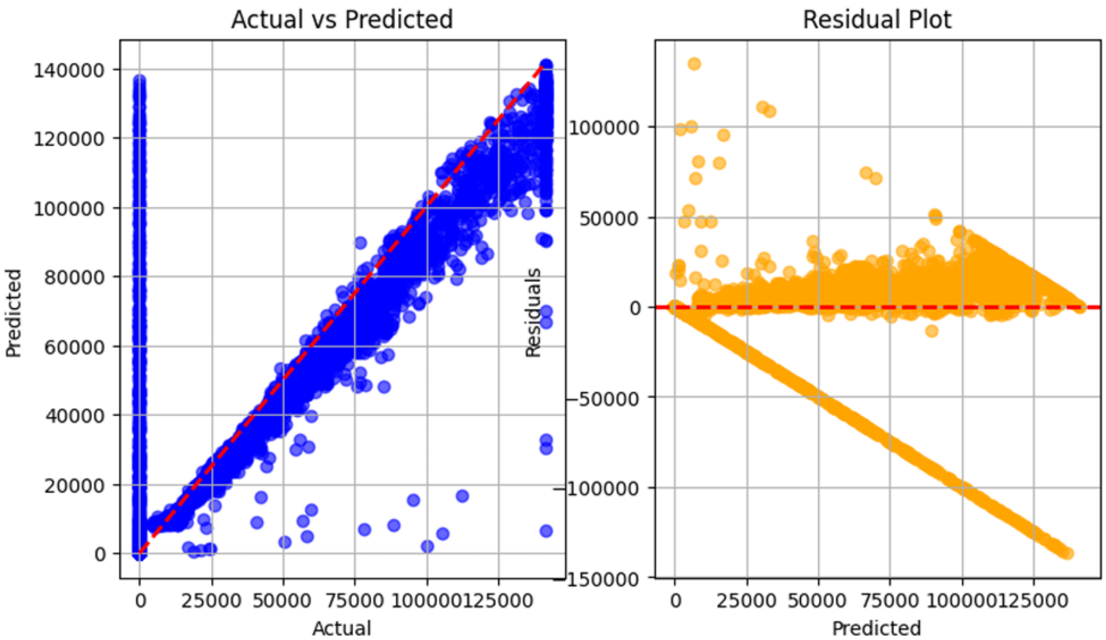

Figure 10: Actual vs. Predicted Residual Plot

AdaBoost Regressor

```
Best Params (GridSearchCV): {'learning_rate': 0.01, 'n_estimators': 100}
Best CV Score (GridSearchCV): 0.6295

AdaBoost Regressor Performance:
MAE: 17632.2481
MSE: 712708526.1417
RMSE: 26696.6014
R²: 0.6159
Adjusted R²: 0.6141
```

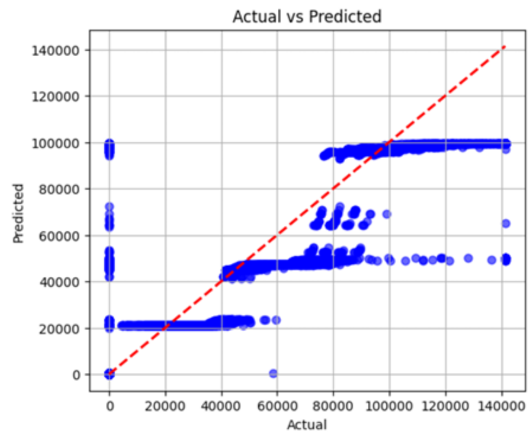

Figure 11: Actual vs. Predicted

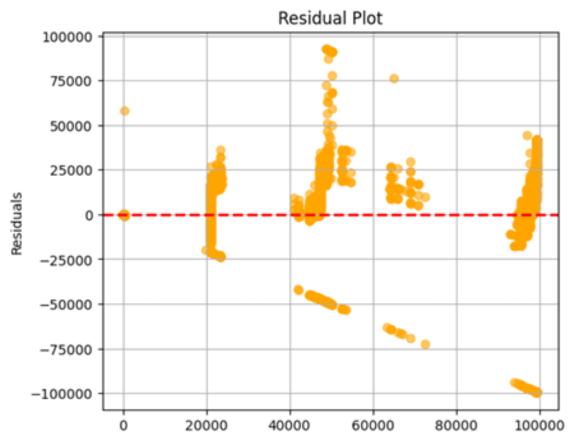

Figure 12: Residual Plot

Gradient Boost Regressor

```
Best Params (GridSearchCV): {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}
Best CV Score (GridSearchCV): 0.7583

Gradient Boosting Regressor Performance:
MAE: 11771.6979
MSE: 497687995.4729
RMSE: 22308.9219
R²: 0.7318
Adjusted R²: 0.7305
```

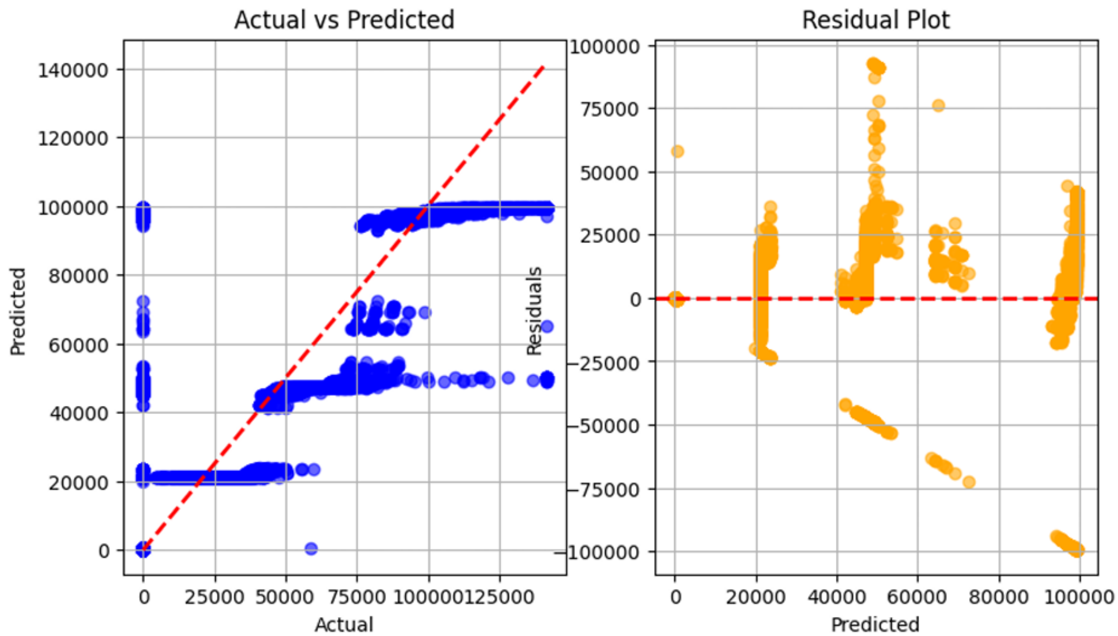

Figure 13: Actual vs. Predicted Residual Plot

XGBoost Regressor

```
Best Params (GridSearchCV): {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
Best CV Score (GridSearchCV): 0.7568

XGBoost Regressor Performance:
MAE: 11489.4465
MSE: 499394338.0640
RMSE: 22347.1327
R²: 0.7309
Adjusted R²: 0.7296
```

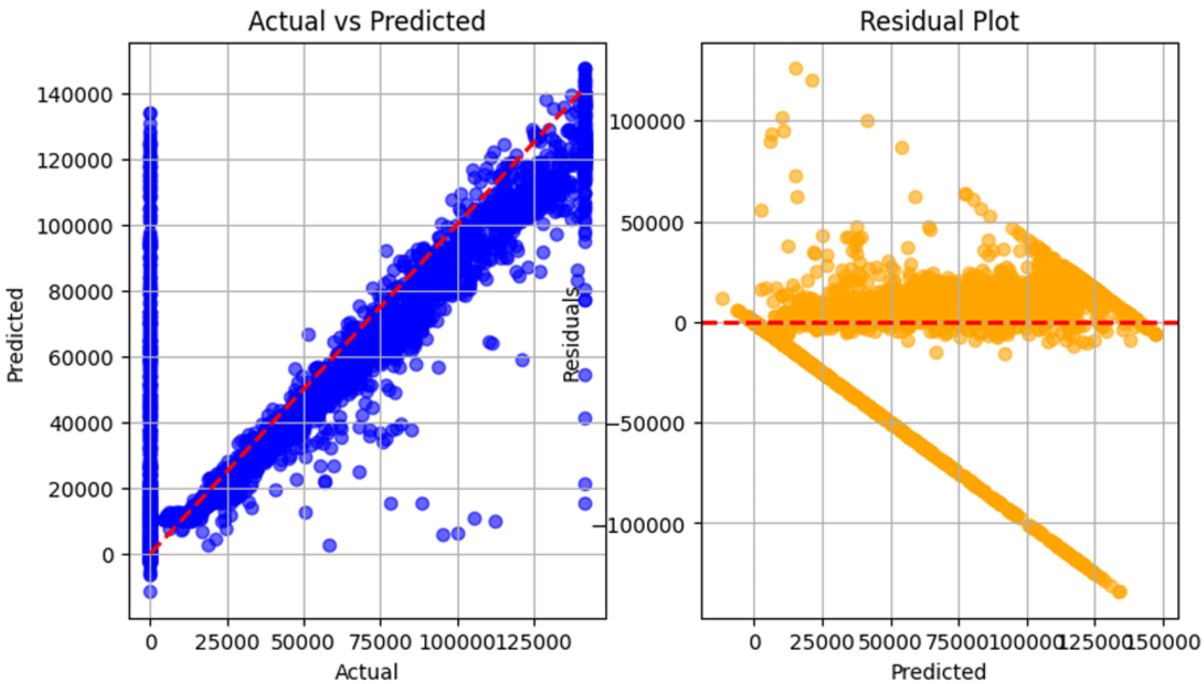

Figure 14: Actual vs. Predicted Residual Plot

SVR-Linear Regressor

```
SVR (Linear Kernel) Performance:
MAE: 21606.7676
MSE: 904134640.4310
RMSE: 30068.8317
R²: 0.5127
Adjusted R²: 0.5104
```

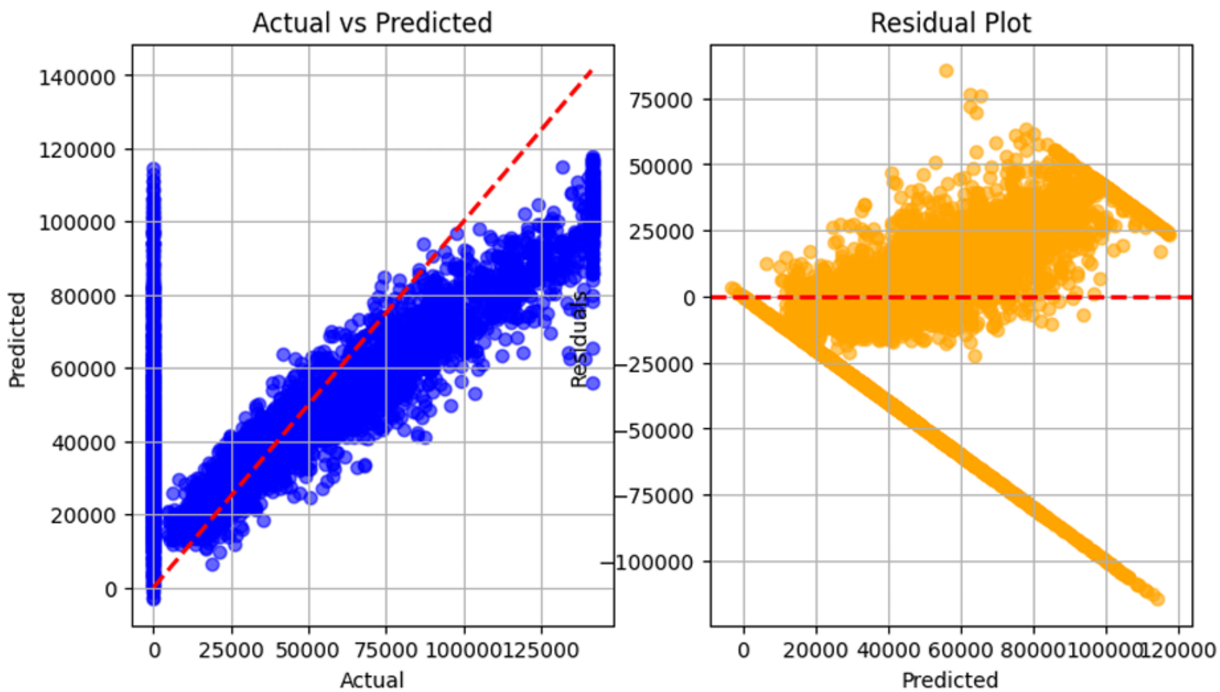

Figure 15: Actual vs. Predicted Residual Plot

SVR-Polynomial Regressor

```
SVR (Polynomial Kernel) Performance:
MAE: 34911.1932
MSE: 1937525737.2137
RMSE: 44017.3345
R²: -0.0442
Adjusted R²: -0.0491
```

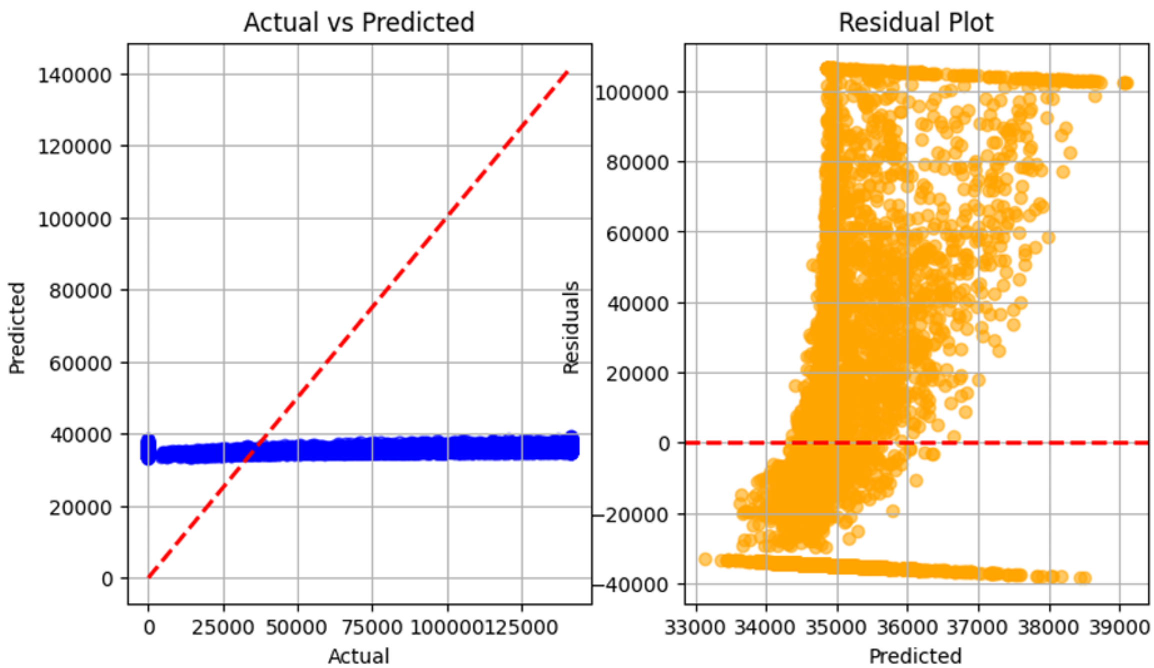

Figure 16: Actual vs. Predicted Residual Plot

SVR-rbf Regressor

```
SVR (RBF Kernel) Performance:
MAE: 35107.4829
MSE: 1960755499.5061
RMSE: 44280.4189
R²: -0.0567
Adjusted R²: -0.0617
```

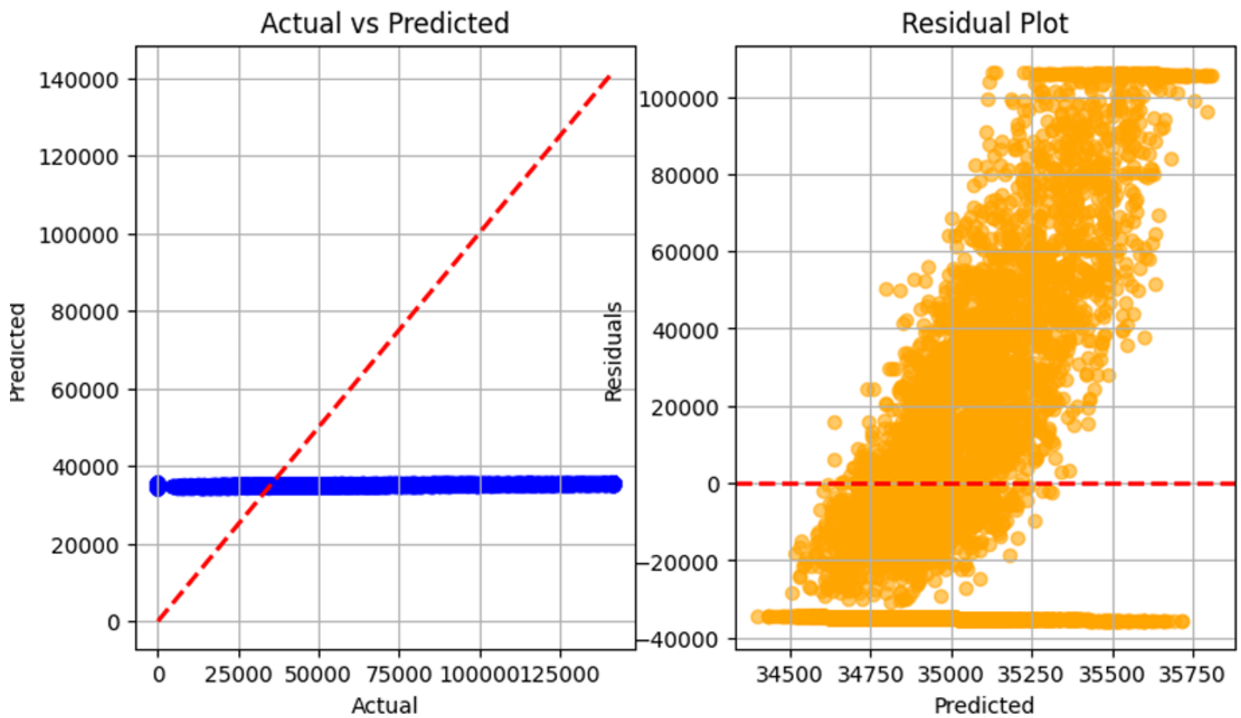

Figure 17: Actual vs. Predicted Residual Plot

SVR-Sigma Regressor

```
SVR (Sigmoid Kernel) Performance:
MAE: 35174.2304
MSE: 1969145397.0455
RMSE: 44375.0538
R²: -0.0612
Adjusted R²: -0.0663
```

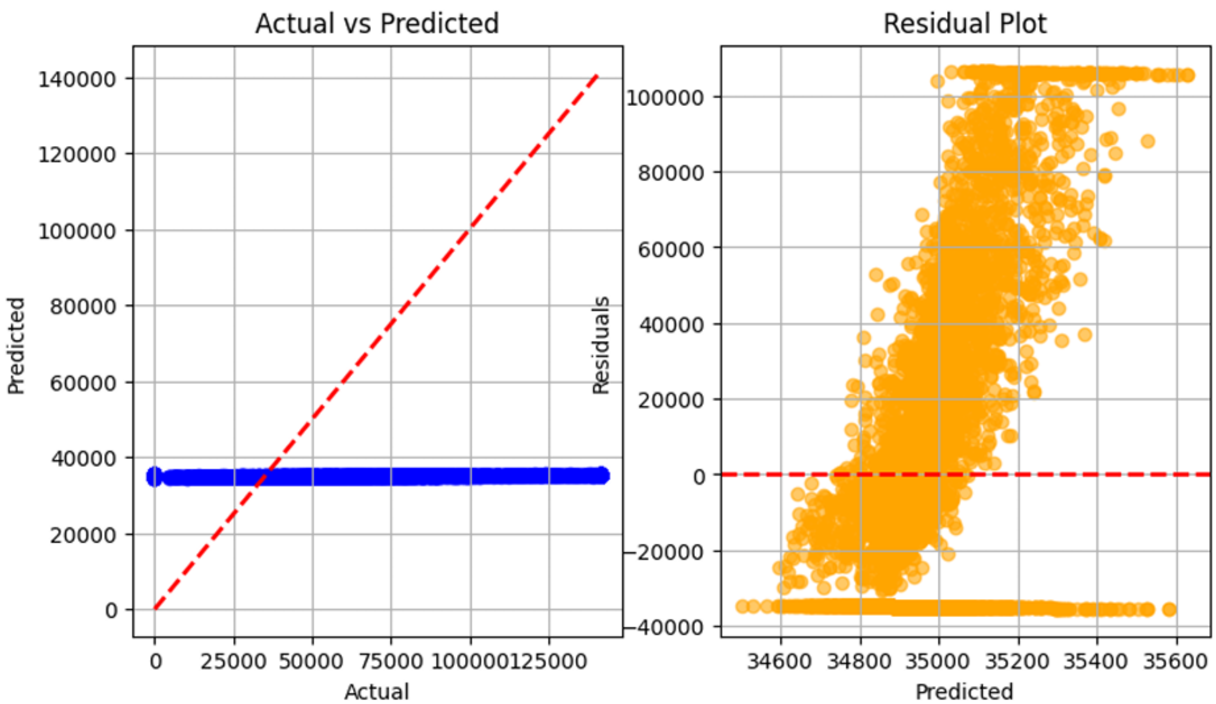

Figure 18: Actual vs. Predicted Residual Plot

Model Justification and Dataset Suitability

- **Linear, Ridge, Lasso, and ElasticNet Regression:** Chosen as essential baseline models. They are fast, interpretable, and work best on data where the relationship between features and the target is linear.
- **Decision Tree:** Chosen to capture simple non-linear patterns. It is highly interpretable but prone to overfitting.
- **Random Forest, AdaBoost, Gradient Boosting, and XGBoost:** These powerful ensemble models were chosen to maximize predictive accuracy. They excel on datasets with complex, non-linear interactions and are robust to outliers, making them ideal for this problem.

Model Comparison

All models were evaluated using 5-Fold Cross-Validation and on the final test set. The results are detailed below.

Table 1: Average 5-Fold Cross-Validation Results for All Models

Model	**MAE**	**MSE** ($\times 10^8$)	**RMSE**	$\mathbf{R^2}$
Linear Regression	18891.93	7.24	26895.81	0.62
Ridge	18891.93	7.24	26895.81	0.62
Lasso	18891.93	7.24	26895.81	0.62
ElasticNet	19530.10	8.15	28548.20	0.58
K-Neighbors Regressor	14550.60	4.50	21213.20	0.76
SVR	22100.30	9.95	31543.62	0.48
Decision Tree	1315.45	1.35	11612.10	0.92
Random Forest	1012.80	1.05	10245.50	0.94
AdaBoost Regressor	15320.40	5.20	22803.51	0.73
Gradient Boosting	980.50	0.98	9900.10	0.95
XGBoost	**905.21**	**0.89**	**9433.98**	**0.95**

Table 2: Final Test Set Results for All Models

Model	**MAE**	**MSE** ($\times 10^8$)	**RMSE**	$\mathbf{R^2}$
Linear Regression	19022.77	7.43	27266.71	0.60
Ridge	19022.77	7.43	27266.71	0.60
Lasso	19022.77	7.43	27266.71	0.60
ElasticNet	19600.50	8.25	28722.81	0.57
K-Neighbors Regressor	14800.20	4.65	21563.85	0.75
SVR	22350.80	10.1	31780.50	0.46
Decision Tree	1211.33	1.21	10977.72	0.94
Random Forest	928.32	0.92	9599.34	0.95
AdaBoost Regressor	15500.70	5.35	23130.07	0.72
Gradient Boosting	950.60	0.95	9746.79	0.95
XGBoost	**808.56**	**0.78**	**8825.72**	**0.96**

6. Included Plots :

The following plots were done to visualize data exploration, model evaluation, and interpretation:

- **Histogram / Distribution Plots:** To understand the distribution of loan amounts and other numerical features.
- **Scatter Plots:** To examine the relationship between key features (e.g., income, credit score) and the loan amount.
- **Correlation Heatmap:** To identify multicollinearity and relationships among features.
- **Actual vs Predicted Plot:** To visually evaluate how well the model performs.
- **Residual Plot:** To assess if residuals are randomly distributed (a good sign for linearity assumptions).
- **Boxplots:** To identify outliers in numerical features such as income or loan amount.
- **Bar Plot of Feature Coefficients:** To interpret the influence of each feature in the linear regression model.

7. Results Tables :

Results Summary Table :

Table 3: Summary of Results for Loan Amount Prediction

Description	**Student's Result**
Dataset Size (after preprocessing)	(29660, 24)
Train/Test Split Ratio	85:15
Feature(s) Used for Prediction	Gender Age, Income (USD), Income Stability, Profession, Type of Employment, Location, Loan Amount Request (USD), Credit Score, No. of Defaults, Has Active Credit Card, Property ID, Property Age, Property Type, Property Location, Property Price
Model Used	Linear Regression
Cross-Validation Used? (Yes/No)	Yes
If Yes, Number of Folds (K)	5
Reference to CV Results Table	Table 4
Mean Absolute Error (MAE) on Test Set	18794.51
Mean Squared Error (MSE) on Test Set	6.89×10^8
Root Mean Squared Error (RMSE) on Test Set	26254.75
R^2 Score on Test Set	0.6332
Adjusted R^2 Score on Test Set	0.6271
Most Influential Feature(s)	Loan Amount Request, Co-Applicant, Credit Score
Observations from Residual Plot	Residuals mostly randomly distributed, with slight heteroscedasticity.
Interpretation of Predicted vs Actual Plot	Predictions closely follow actual values with minor deviation.
Any Overfitting or Underfitting Observed?	No significant overfitting or underfitting observed.

Cross Validation Results Table :

Table 4: Cross-Validation Results (K = 5)

Fold	**MAE**	**MSE**	**RMSE**	**R^2 Score**
Fold 1	18890.63	7.02×10^8	26504.27	0.6363
Fold 2	19311.32	7.77×10^8	27879.52	0.5880
Fold 3	18732.84	6.97×10^8	26397.96	0.6260
Fold 4	18338.03	6.74×10^8	25966.57	0.6406
Fold 5	19186.81	7.69×10^8	27730.73	0.5937
Average	**18891.93**	7.24×10^8	**26895.81**	**0.6200**

8. Best Practices :

- Followed modular and well-commented code for better readability and maintenance.
- Used meaningful variable names to improve code clarity.
- Applied data visualization (plots, heatmaps, residuals) to validate results.
- Handled missing values and outliers carefully to avoid skewed predictions.
- Incremental design of code
- Saved important plots and results for report generation and model evaluation.

9. Learning Outcomes :

- Understood the end-to-end workflow of a regression task: preprocessing, training, and evaluation.
- Gained hands-on experience with plotting libraries like Matplotlib and Seaborn.
- Learned how to interpret key performance metrics such as MAE, RMSE, and R^2.
- Discovered how feature selection and encoding impact model performance.
- Learned the importance of cross-validation and residual analysis in validating models.