Sri Sivasubramaniya Nadar College of Engineering, Chennai

(An autonomous Institution affiliated to Anna University)

Degree & Branch	M.Tech Computer Science & Engineering Integrated (5 Years)	**Semester** V
Subject Code & Name	ICS1512 & Machine Learning Algorithms Laboratory	
Academic year	2025–2026 (Odd)	**Batch:** 2023–2028

Name: I.S.Rajesh **Register No.: 3122237001042**

Experiment #2: Loan Amount Prediction using Linear Regression

Aim:
To learn working of Linear Regression on real-world datasets like loan amount prediction in detail.

Libraries used:
`numpy, pandas, sklearn, matplotlib`

Objective:
Apply Linear Regression to predict loan amount sanctioned to users using provided dataset. Use K-Fold Cross Validation technique to validate the model after effective splitting.

1 Mathematical Description

Linear Regression

Given data points $(x_1, y_1), \ldots, (x_n, y_n)$, Linear Regression fits the model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \varepsilon$$

The model parameters β_j are chosen to minimize the Residual Sum of Squares (RSS):

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

Ridge Regression

Ridge regression adds L2 regularization (penalty on coefficients):

$$\text{Loss}_{\text{Ridge}} = \sum (y_i - \hat{y}_i)^2 + \lambda \sum \beta_j^2$$

Lasso Regression

Lasso regression applies L1 regularization:

$$\text{Loss}_{\text{Lasso}} = \sum (y_i - \hat{y}_i)^2 + \lambda \sum |\beta_j|$$

ElasticNet Regression

ElasticNet combines both L1 and L2 penalties:

$$\text{Loss}_{\text{ElasticNet}} = \sum (y_i - \hat{y}_i)^2 + \lambda_1 \sum |\beta_j| + \lambda_2 \sum \beta_j^2$$

K-Fold Cross-Validation

Cross-validation helps estimate a model's generalization to unseen data. In K-Fold CV:

- The dataset is divided into K approximately equal folds.
- The model is trained on $K - 1$ folds and validated on the remaining fold.
- This process repeats K times, each time with a different fold as the validation set.
- Final performance is calculated by averaging the metrics across the K folds.

Mathematically, if E_k is the evaluation metric (e.g., RMSE) from the k^{th} fold, then the overall metric is:

$$E_{\text{avg}} = \frac{1}{K} \sum_{k=1}^{K} E_k$$

Python Codes for Linear Regression Model

Listing 1: Import libraries

```python
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, root_mean_squared_error
    ↪ , mean_absolute_error, r2_score
```

Listing 2: Numpy

```
# Load Dataset
df = pd.read_csv("train.csv")

# Drop rows with missing target
df = df.dropna(subset=['Loan␣Sanction␣Amount␣(USD)'])

# Fill missing values
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
    ↪ .tolist()
numerical_cols.remove('Loan␣Sanction␣Amount␣(USD)')

categorical_cols = df.select_dtypes(include='object').columns.tolist()

for col in numerical_cols:
    df[col] = df[col].fillna(df[col].median())

# Simultaneously encode categorical variables

for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])

# Target-Guided Ordinal Encoding for Categorical Columns
for col in categorical_cols:
    ordering = df.groupby(col)["Loan␣Sanction␣Amount␣(USD)"].mean().
        ↪ sort_values().index
    ordinal_map = {key: idx for idx, key in enumerate(ordering)}
    df[col] = df[col].map(ordinal_map)

# EDA

# 1. Distribution of Loan Amount
plt.figure(figsize=(6, 4))
sns.histplot(df['Loan␣Sanction␣Amount␣(USD)'], kde=True, color='blue')
plt.title("Loan␣Amount␣Sanctioned␣-␣Distribution")
plt.show()

# 2. Boxplots for numerical features
for col in ['Income␣(USD)', 'Credit␣Score', 'Loan␣Amount␣Request␣(USD)'
    ↪ , 'Property␣Price']:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=df[col])
    plt.title(f"Boxplot␣of␣{col}")
    plt.show()

# 3. Correlation Heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation␣Matrix")
plt.show()

# 4. Scatter plots
plt.figure(figsize=(6, 4))
sns.scatterplot(x=df['Income␣(USD)'], y=df['Loan␣Sanction␣Amount␣(USD)'
    ↪ ])
plt.title("Income␣vs␣Loan␣Sanction␣Amount")
plt.show()
plt.figure(figsize=(6, 4))
sns.scatterplot(x=df[’Credit␣Score’], y=df[’Loan␣Sanction␣Amount␣(USD)’
   ↪ ])
plt.title("Credit␣Score␣vs␣Loan␣Sanction␣Amount")
plt.show()

# Feature Engineering
df[’DebtToIncomeRatio’] = df[’Current␣Loan␣Expenses␣(USD)’] / (df[’
   ↪ Income␣(USD)’] + 1)
df[’LoanToPropertyRatio’] = df[’Loan␣Amount␣Request␣(USD)’] / (df[’
   ↪ Property␣Price’] + 1)
df[’TotalFinancialBurden’] = df[’Current␣Loan␣Expenses␣(USD)’] + df[’
   ↪ Loan␣Amount␣Request␣(USD)’]
df.drop([’Customer␣ID’, ’Name’, ’Gender’, ’Property␣ID’], axis=1,
   ↪ inplace=True)

df.to_csv(’updated_train.csv’, index=False)

# Dataset size (after prepreocessing)
print(df.shape)

# Define features and target
X = df.drop(’Loan␣Sanction␣Amount␣(USD)’, axis=1)
y = df[’Loan␣Sanction␣Amount␣(USD)’]

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting: Train (60%), Validation (20%), Test (20%)
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y,
   ↪ test_size=0.4, 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 Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)

# Evaluating Model on Test and Validation Sets (Without Performance
   ↪ Metrics)

def plot_actual_vs_predicted(y_true, y_pred, title):
    plt.figure(figsize=(6, 4))
    plt.scatter(y_true, y_pred, alpha=0.5, edgecolor=’k’)
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()
       ↪ ], ’r--’)
    plt.xlabel("Actual␣Loan␣Amount")
    plt.ylabel("Predicted␣Loan␣Amount")
    plt.title(f"Actual␣vs␣Predicted␣-␣{title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

```
def plot_residuals(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    plt.scatter(y_pred, residuals, alpha=0.5, edgecolor=’k’)
    plt.axhline(y=0, color=’red’, linestyle=’--’)
    plt.xlabel("Predicted␣Loan␣Amount")
    plt.ylabel("Residuals")
    plt.title(f"Residual␣Plot␣-␣{title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residual_distribution(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    sns.histplot(residuals, kde=True, color=’skyblue’)
    plt.title(f"Residual␣Distribution␣-␣{title}")
    plt.xlabel("Residuals")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# For Validation Set
plot_actual_vs_predicted(y_val, y_val_pred, "Validation␣Set")
plot_residuals(y_val, y_val_pred, "Validation␣Set")
plot_residual_distribution(y_val, y_val_pred, "Validation␣Set")

# For Test Set
plot_actual_vs_predicted(y_test, y_test_pred, "Test␣Set")
plot_residuals(y_test, y_test_pred, "Test␣Set")
plot_residual_distribution(y_test, y_test_pred, "Test␣Set")

# Performance Metrics
mse = mean_squared_error(y_test, y_test_pred)
rmse = root_mean_squared_error(y_test, y_test_pred)
mae = mean_absolute_error(y_test, y_test_pred)
r2 = r2_score(y_test, y_test_pred)
n, p = X_test.shape
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)

print("Mean␣Squared␣Error:", mse)
print("Root␣Mean␣Squared␣Error:", rmse)
print("Mean␣Absolute␣Error:", mae)
print("R2␣Score:", r2)
print("Adjusted␣R2␣Score:", adjusted_r2)

# Visualization of Results

# Plot: Actual vs Predicted
plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_test_pred, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], ’r--’)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual␣vs␣Predicted␣Loan␣Amount")
plt.show()
```

```
# Plot: Residuals
residuals = y_test - y_test_pred
plt.figure(figsize=(6, 4))
sns.histplot(residuals, kde=True, color='orange')
plt.title("Residuals␣Distribution")
plt.show()

# Plot: Feature Coefficients
coefficients = model.coef_
feature_names = X.columns
plt.figure(figsize=(10, 6))
sns.barplot(x=coefficients, y=feature_names)
plt.title("Linear␣Regression␣Coefficients")
plt.xlabel("Coefficient␣Value")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()

# Performance Metrics for Validation to compare with Test set and
    ↪ conclude on Overfitting or Underfitting
mse = mean_squared_error(y_val, y_val_pred)
rmse = root_mean_squared_error(y_val, y_val_pred)
mae = mean_absolute_error(y_val, y_val_pred)
r2 = r2_score(y_val, y_val_pred)
n, p = X_test.shape
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)

print("Mean␣Squared␣Error:", mse)
print("Root␣Mean␣Squared␣Error:", rmse)
print("Mean␣Absolute␣Error:", mae)
print("R2␣Score:", r2)
print("Adjusted␣R2␣Score:", adjusted_r2)
```

Output for Linear Regression Model

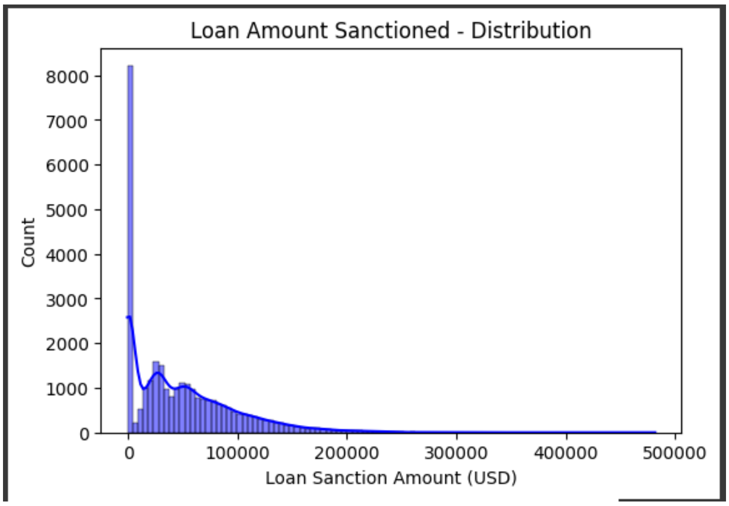


Figure 1: Exploratory Data Analysis 1

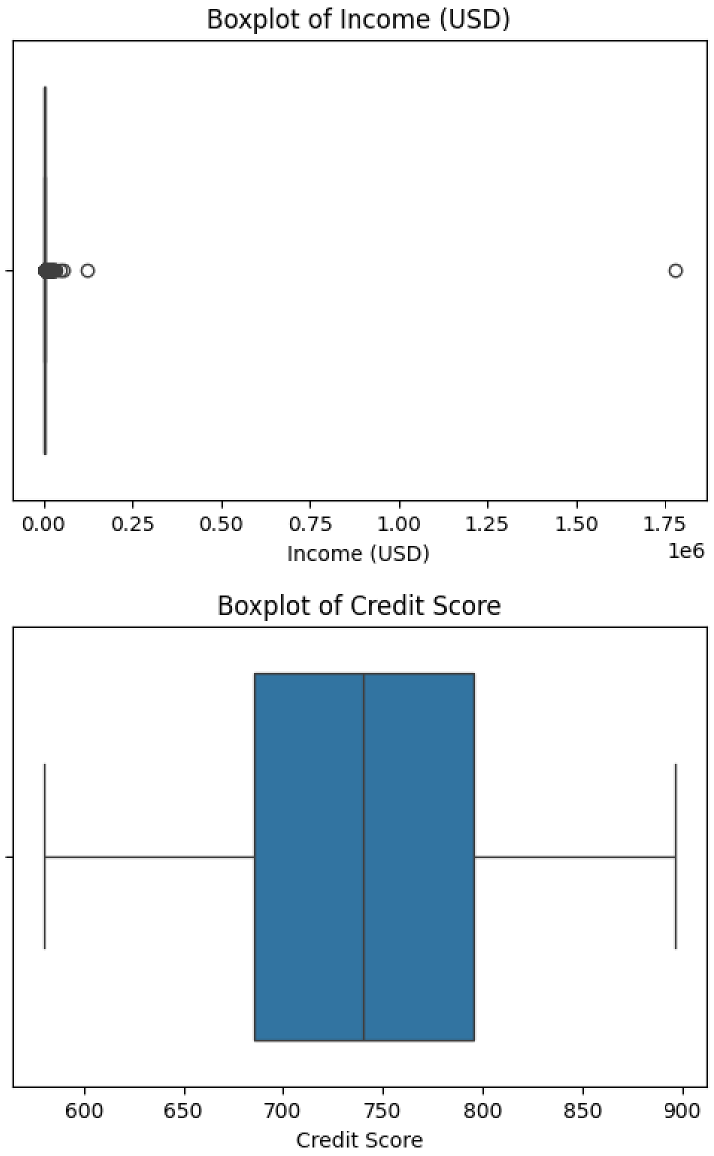


Figure 2: Exploratory Data Analysis 2

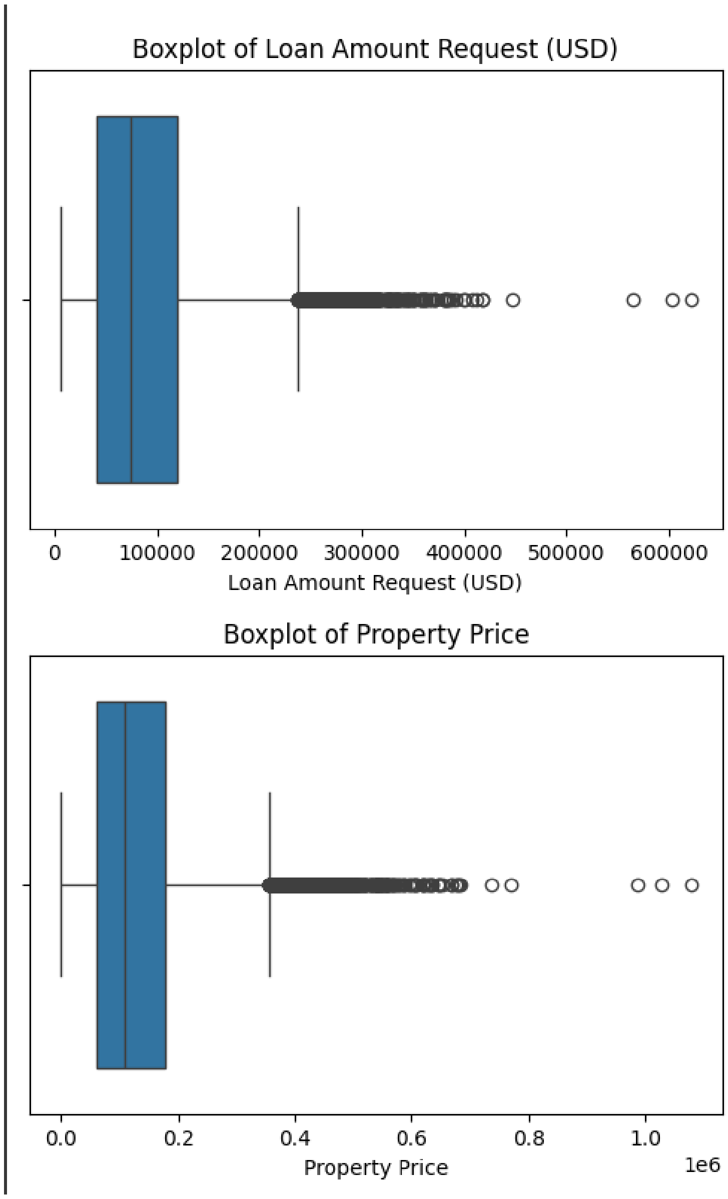


Figure 3: Exploratory Data Analysis 3

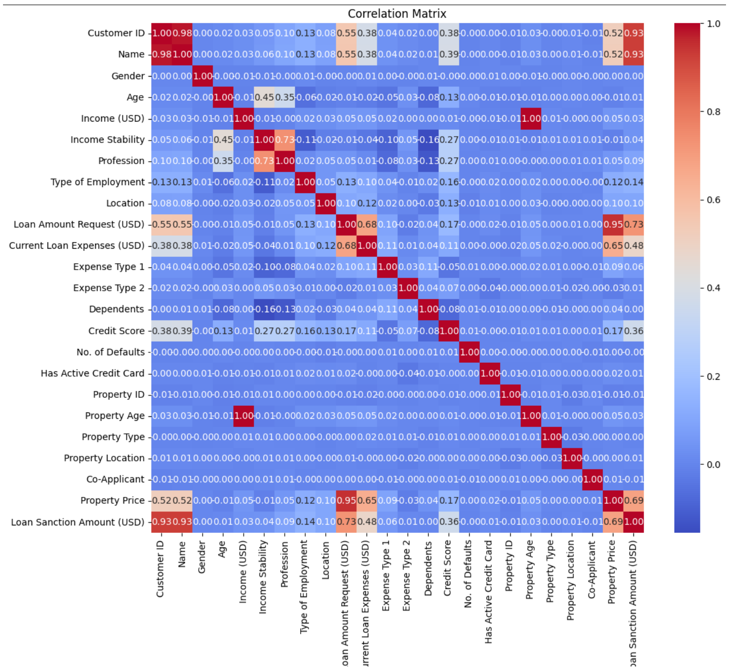


Figure 4: Exploratory Data Analysis 4

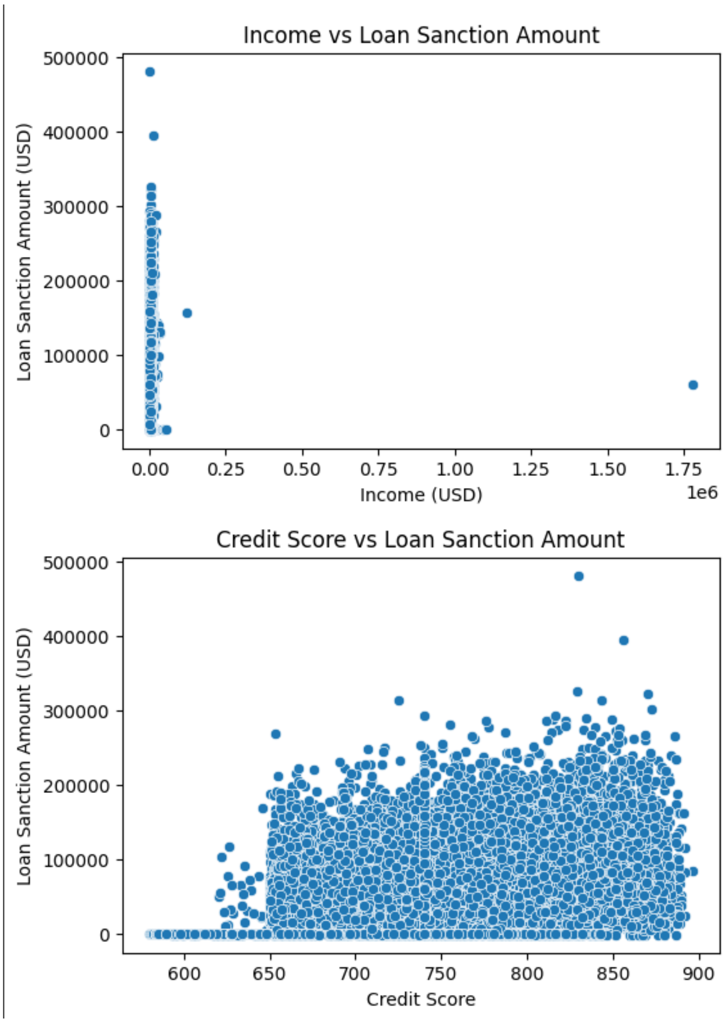


Figure 5: Exploratory Data Analysis 5

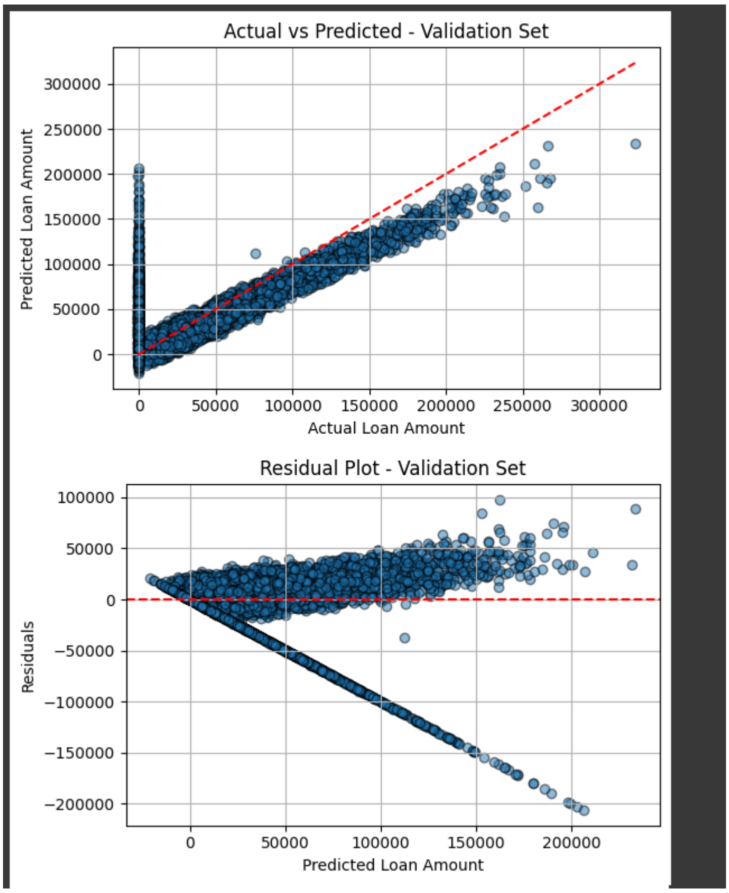


Figure 6: Evaluation without Metrics (Test vs Validation) 1

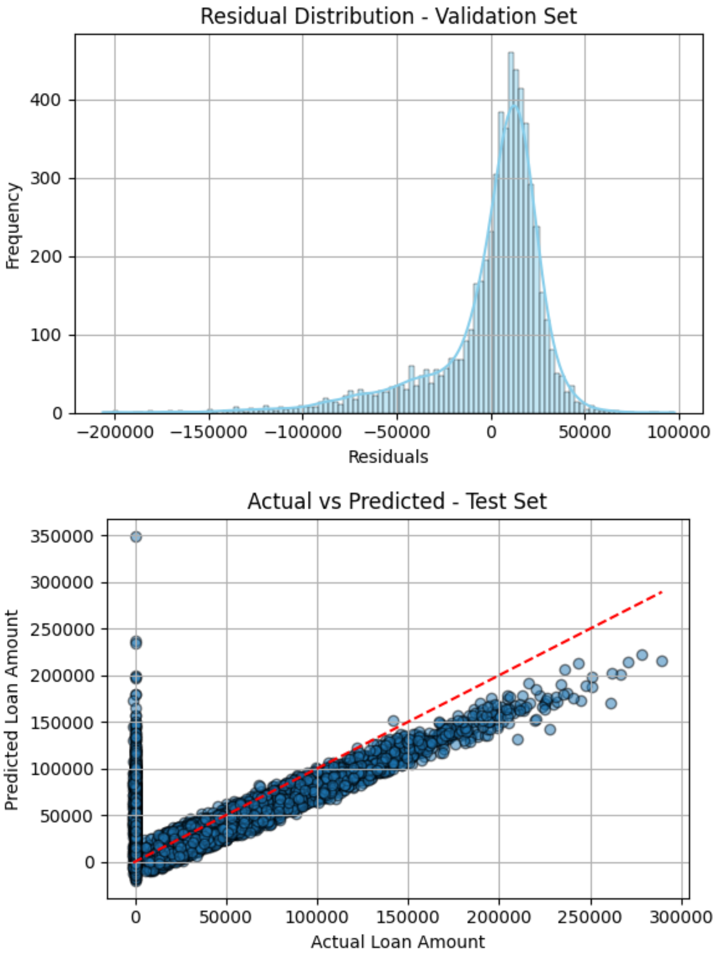


Figure 7: Evaluation without Metrics (Test vs Validation) 2

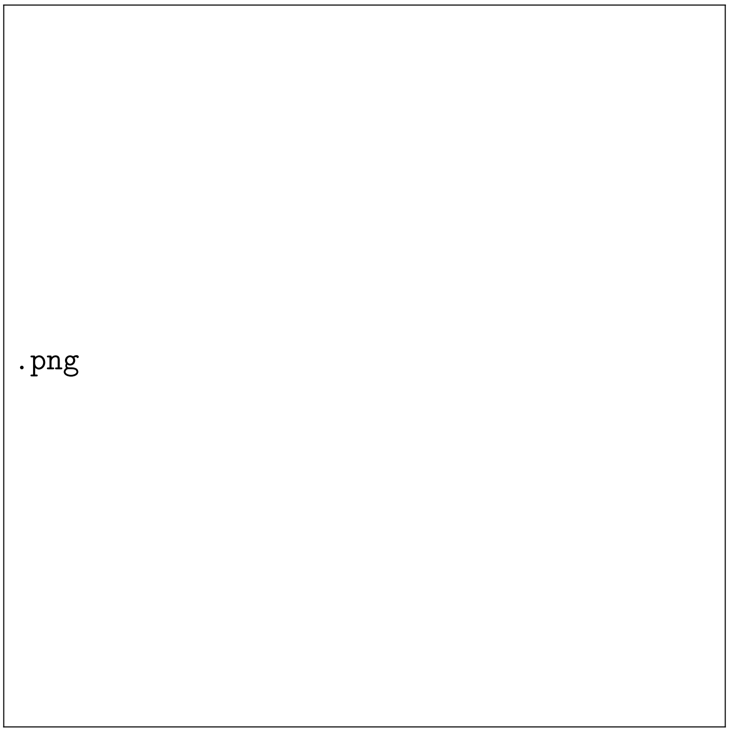
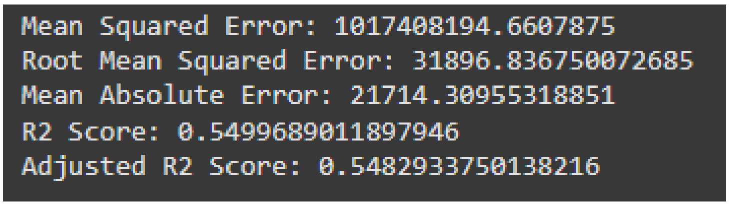


Figure 8: Evaluation without Metrics (Test vs Validation) 3



```
Mean Squared Error: 1017408194.6607875
Root Mean Squared Error: 31896.836750072685
Mean Absolute Error: 21714.30955318851
R2 Score: 0.5499689011897946
Adjusted R2 Score: 0.5482933750138216
```

Figure 9: Test Metrics

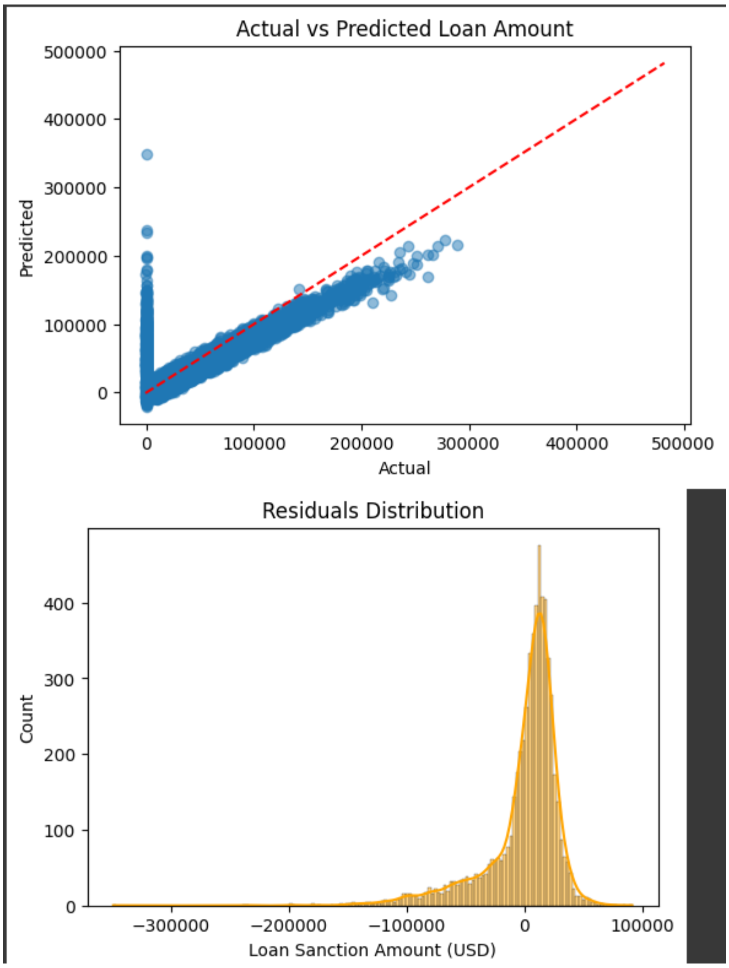


Figure 10: Result Visualization

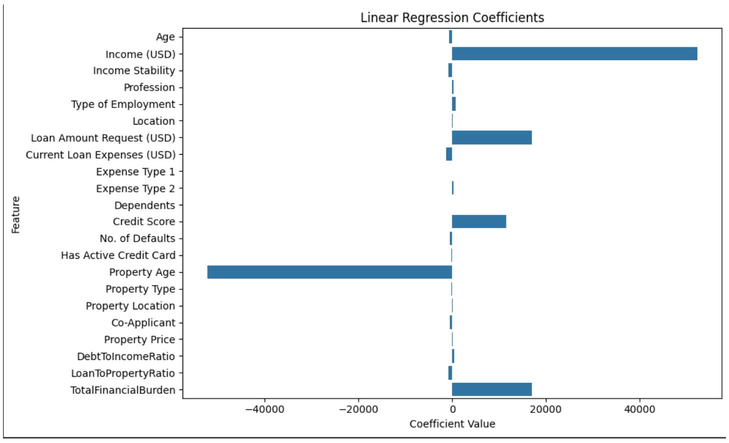
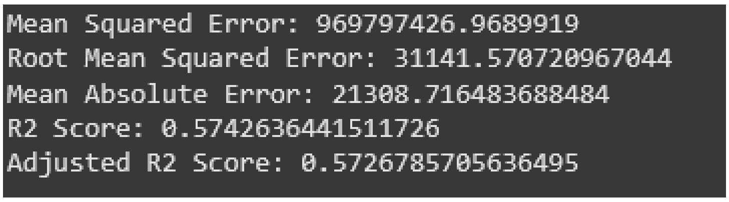


Figure 11: Feature Coefficient Comparison



```
Mean Squared Error: 969797426.9689919
Root Mean Squared Error: 31141.570720967044
Mean Absolute Error: 21308.716483688484
R2 Score: 0.5742636441511726
Adjusted R2 Score: 0.5726785705636495
```

Figure 12: Validation Metrics

Python Codes for K-Fold Cross Validation

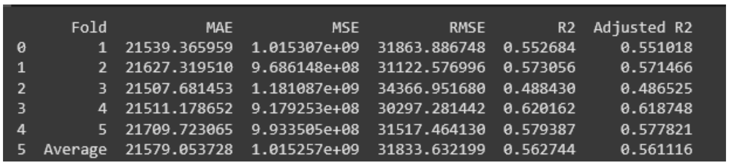
Listing 3: Import libraries

```
# Import Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, root_mean_squared_error
    ↪ , mean_absolute_error, r2_score
```

Listing 4: K-Fold Cross Validation

```
# Load the updated dataset
df = pd.read_csv("updated_train.csv")

# Define the target column
target_column = 'Loan␣Sanction␣Amount␣(USD)'

# Separate features and target
X = df.drop(columns=[target_column])
y = df[target_column]

# Standardize the feature values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Initialize 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Store evaluation metrics
results = []

# Run cross-validation
for fold, (train_index, val_index) in enumerate(kf.split(X_scaled), 1):
    X_train, X_val = X_scaled[train_index], X_scaled[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]

    # Train the model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Predict on validation set
    y_pred = model.predict(X_val)

    # Calculate metrics
    mse = mean_squared_error(y_val, y_pred)
    rmse = root_mean_squared_error(y_val, y_pred)
    mae = mean_absolute_error(y_val, y_pred)
    r2 = r2_score(y_val, y_pred)
    adj_r2 = 1 - (1 - r2) * (len(y_val) - 1) / (len(y_val) - X_val.
       ↪ shape[1] - 1)

    results.append({
        "Fold": fold,
        "MAE": mae,
        "MSE": mse,
        "RMSE": rmse,
        "R2": r2,
        "Adjusted␣R2": adj_r2
    })

cv_results_df = pd.DataFrame(results)

average_row = {
    "Fold": "Average",
    "MAE": cv_results_df["MAE"].mean(),
    "MSE": cv_results_df["MSE"].mean(),
    "RMSE": cv_results_df["RMSE"].mean(),
    "R2": cv_results_df["R2"].mean(),
    "Adjusted␣R2": cv_results_df["Adjusted␣R2"].mean()
}

cv_results_df = pd.concat([cv_results_df, pd.DataFrame([average_row])],
   ↪ ignore_index=True)

# Display results

# Display results
print(cv_results_df)
```

Output for K-Fold Cross Validation



```
       Fold           MAE           MSE          RMSE        R2  Adjusted R2
0         1  21539.365959  1.015307e+09  31863.886748  0.552684     0.551018
1         2  21627.319510  9.686148e+08  31122.576996  0.573056     0.571466
2         3  21507.681453  1.181087e+09  34366.951680  0.488430     0.486525
3         4  21511.178652  9.179253e+08  30297.281442  0.620162     0.618748
4         5  21709.723065  9.933505e+08  31517.464130  0.579387     0.577821
5   Average  21579.053728  1.015257e+09  31833.632199  0.562744     0.561116
```

Figure 13: K-Fold Cross Validation Metrics

Python Codes for Linear Regression Variations

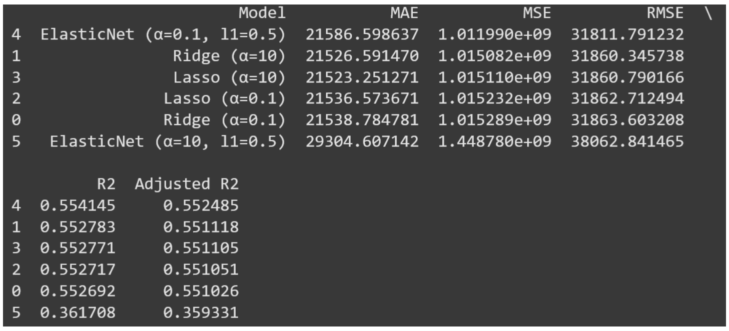
Listing 5: Import libraries

```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error, root_mean_squared_error
    ↪ , mean_absolute_error, r2_score
```

Listing 6: Variations codes

```
import pandas as pd
import numpy as np
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error,
    ↪ r2_score

# Load dataset
df = pd.read_csv("updated_train.csv")

# Target and Features
target_column = ’Loan␣Sanction␣Amount␣(USD)’
X = df.drop(columns=[target_column])
y = df[target_column]

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
    ↪ test_size=0.2, random_state=42)

# Define models with hyperparameter variations
models = {
    "Ridge␣(alpha=0.1)": Ridge(alpha=0.1),
    "Ridge␣(alpha=10)": Ridge(alpha=10),
    "Lasso␣(alpha=0.1)": Lasso(alpha=0.1, max_iter=10000),
    "Lasso␣(alpha=10)": Lasso(alpha=10, max_iter=10000),
    "ElasticNet␣(alpha=0.1,␣l1=0.5)": ElasticNet(alpha=0.1, l1_ratio
        ↪ =0.5, max_iter=10000),
    "ElasticNet␣(alpha=10,␣l1=0.5)": ElasticNet(alpha=10, l1_ratio=0.5,
        ↪  max_iter=10000)
}

# Evaluate and store results
results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    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)
    adj_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) - X_test.
        ↪ shape[1] - 1)
    results.append({
        "Model": name,
        "MAE": mae,
        "MSE": mse,
        "RMSE": rmse,
        "R2": r2,
        "Adjusted␣R2": adj_r2
    })

# Display results
results_df = pd.DataFrame(results).sort_values(by="R2", ascending=False
    ↪ )
print(results_df)
```

Output for Linear Regression Variations



```
                        Model           MAE           MSE          RMSE  \
4  ElasticNet (α=0.1, l1=0.5)  21586.598637  1.011990e+09  31811.791232
1                Ridge (α=10)  21526.591470  1.015082e+09  31860.345738
3                Lasso (α=10)  21523.251271  1.015110e+09  31860.790166
2               Lasso (α=0.1)  21536.573671  1.015232e+09  31862.712494
0               Ridge (α=0.1)  21538.784781  1.015289e+09  31863.603208
5   ElasticNet (α=10, l1=0.5)  29304.607142  1.448780e+09  38062.841465

         R2  Adjusted R2
4  0.554145     0.552485
1  0.552783     0.551118
3  0.552771     0.551105
2  0.552717     0.551051
0  0.552692     0.551026
5  0.361708     0.359331
```

Figure 14: Variations Metrics

Results Tables

Table 1: Cross-Validation Results (K = 5)

Fold	**MAE**	**MSE**	**RMSE**	**R^2 Score**
Fold 1	21539.37	1.0153e+09	31863.89	0.552684
Fold 2	21627.32	9.6861e+08	31122.58	0.573856
Fold 3	21507.68	1.1811e+09	34366.95	0.488430
Fold 4	21511.18	9.1793e+08	30297.28	0.620162
Fold 5	21709.72	9.9336e+08	31517.46	0.579387
Average	21579.05	1.0153e+09	31833.63	0.562744

Table 2: Summary of Results for Loan Amount Prediction

Description	**Student's Result**
Dataset Size (after preprocessing)	29660 rows, 23 columns
Train/Test Split Ratio	60% Test, 20% Train, 20% Validation
Feature(s) Used for Prediction	Age, Income, Property, Profession, Type of Employment, Location, Expenses, Credit Score, Defaults, Dependents, DebttoIncomeRatio, LoantoPropertyRatio, FinancialBurden
Model Used	Linear Regression
Cross-Validation Used? (Yes/No)	Yes
If Yes, Number of Folds (K)	5
Reference to CV Results Table	Table 1
Mean Absolute Error (MAE) on Test Set	[Enter if separate test set used]
Mean Squared Error (MSE) on Test Set	[Enter if separate test set used]
Root Mean Squared Error (RMSE) on Test Set	[Enter if separate test set used]
R^2 Score on Test Set	[Enter if separate test set used]
Adjusted R^2 Score on Test Set	[Enter if applicable]
Most Influential Feature(s)	Income, Credit Score, Property Age, FinancialBurden
Observations from Residual Plot	Residuals are mostly near 0, with some randomly scattered in $$-35000, 10000$$
Interpretation of Predicted vs Actual Plot	Most points cluster along y=x line with some near it, and some near each axes, indicating a good fit for the most part
Any Overfitting or Underfitting Observed?	No major signs
Brief Justification for above (e.g., training vs test error, residual patterns)	Train and Validation metrics are good and very cose to each other, and plots show good fit

Learning Outcomes

- Learned to preprocess datasets using feature scaling and handling missing values.
- Applied multiple ML models and understood when to use each.
- Practiced evaluating regression with MSE and R^2, and classification with accuracy/confusion matrix.
- Gained practical experience using popular Python libraries in machine learning workflows.

Best Practices

- Normalize or standardize features when necessary.
- Split data into train, validation, and test sets.
- Use K-Fold Cross Validation to get more reliable performance estimates.
- Visualize residuals and feature importance to interpret model behavior.

Learning Outcomes

- Understood and applied linear regression for regression tasks.
- Evaluated model performance using MAE, MSE, RMSE, R^2.
- Gained experience with cross-validation and model interpretation.
- Learned about regularization and comparison with Ridge, Lasso, ElasticNet.