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

Aim:

To develop a machine learning model using Linear Regression to predict the sanctioned loan amount based on historical applicant data, by preprocessing and analyzing the dataset, applying feature engineering techniques, and evaluating model performance using metrics such as MAE, MSE, RMSE, and R^2 score with appropriate visualizations.

Libraries Used:

- pandas
- numpy
- matplotlib.pyplot
- seaborn
- scikit-learn

Mathematical Description

Linear Regression predicts the loan sanction amount using the following model:

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

Where:

- y — Loan Sanction Amount
- $x_1, x_2, \ldots, x_n$ — features like Age, Income, Credit Score, etc.
- β_0 — intercept
- $\beta_1, \ldots, \beta_n$ — coefficients
- ε — error term

Residual Sum of Squares (RSS):

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

Evaluation metrics:

- **MAE** – Mean Absolute Error
- **MSE** – Mean Squared Error
- **RMSE** – Root Mean Squared Error
- **R²** – Coefficient of Determination
- **Adjusted R²** – Adjusted for number of features

Python Code

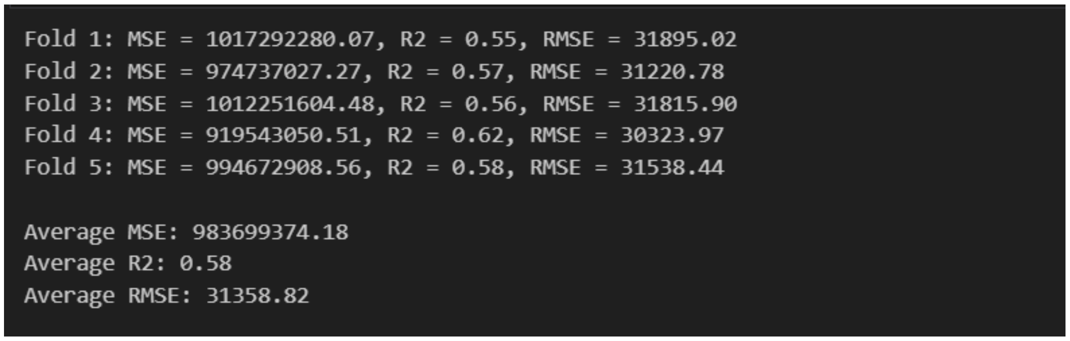
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df = pd.read_csv(r"C:/Users/paru/Downloads/archive/train.csv")
df.head()
df.describe()
df.isnull().sum()

df["Gender"]=df["Gender"].fillna(df["Gender"].mode()[0])
df["Income (USD)"]=df["Income (USD)"].fillna(df["Income (USD)"].mean())
df["Income Stability"]=df["Income Stability"].fillna(df["Income Stability"].mode()[0])
df["Type of Employment"]=df["Type of Employment"].fillna("unknown")
df["Current Loan Expenses (USD)"]=df["Current Loan Expenses (USD)"].fillna(df["Income (USD)"].mean())
df["Dependents"]=df["Dependents"].fillna(df["Dependents"].mean())
df["Credit Score"]=df["Credit Score"].fillna(df["Credit Score"].mean())
df["Has Active Credit Card"]=df["Has Active Credit Card"].fillna(df["Has Active Credit Card"].mode()[0]
df["Property Age"]=df["Property Age"].fillna(df["Property Age"].mean())
df["Property Location"]=df["Property Location"].fillna(df["Property Location"].mode()[0])
df.dropna(subset=["Loan Sanction Amount (USD)"], inplace=True)

df.drop(columns=["Customer ID", "Name", "Property ID","Type of Employment","Profession"], inplace=True)

df["Gender"] = df["Gender"].map({"M": 1, "F": 0})
df["Expense Type 1"] = df["Expense Type 1"].map({"Y": 1, "N": 0})
df["Expense Type 2"] = df["Expense Type 2"].map({"Y": 1, "N": 0})
df["Has Active Credit Card"] = df["Has Active Credit Card"].map({
    "Active": 1,
    "Inactive": 0,
    "Unpossessed": -1
})
df["Income Stability"] = df["Income Stability"].map({"Low": 0, "High": 1})
df["Property Location"] = df["Property Location"].map({"Rural": 0, "Semi-Urban": 1, "Urban": 2})
df["Location"] = df["Location"].map({"Rural": 0, "Semi-Urban": 1, "Urban": 2})
```

```
x = df.drop('Loan Sanction Amount (USD)', axis=1)
y = df["Loan Sanction Amount (USD)"]

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="mean")
x_train = imputer.fit_transform(x_train)
x_test = imputer.transform(x_test)

model = LinearRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

from sklearn.metrics import mean_squared_error, r2_score
r2_error = r2_score(y_test, y_pred)
print(r2_error)
print(df.columns)

from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler

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

mse_list = []
r2_list = []
rmse = []

for fold, (train_idx, test_idx) in enumerate(kf.split(x), 1):
    X_train, X_test = x.iloc[train_idx], x.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    rms = np.sqrt(mse)

    mse_list.append(mse)
    r2_list.append(r2)
    rmse.append(rms)

    print(f"Fold {fold}: MSE = {mse:.2f}, R2 = {r2:.2f}, RMSE = {rms:.2f}")

print(f"\nAverage MSE: {np.mean(mse_list):.2f}")
print(f"Average R2: {np.mean(r2_list):.2f}")
print(f"Average RMSE: {np.mean(rmse):.2f}")

plt.figure(figsize=(10, 5))
plt.plot(range(1, k+1), mse_list, marker='o', label='MSE per Fold')
plt.plot(range(1, k+1), r2_list, marker='s', label='R² per Fold')
plt.title("K-Fold Validation Performance")
plt.xlabel("Fold")
plt.ylabel("Score")
plt.legend()
plt.grid(True)
plt.show()

import seaborn as sns
sns.set(style="whitegrid")

categorical_cols = ['Gender', 'Income Stability', 'Location', 'Expense Type 1', 'Expense Type 2',
                    'Has Active Credit Card', 'Property Type', 'Property Location', 'Co-Applicant']

for col in categorical_cols:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=df, x=col, palette="Set2")
    plt.title(f"Count Plot of {col}")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

target = 'Loan Sanction Amount (USD)'
numerical_cols = ['Age', 'Income (USD)', 'Loan Amount Request (USD)', 'Current Loan Expenses (USD)',
                  'Dependents', 'Credit Score', 'No. of Defaults', 'Property Age', 'Property Price']

for col in ['Income (USD)', 'Credit Score', 'Property Price', 'Loan Amount Request (USD)']:
    plt.figure(figsize=(6, 4))
    sns.scatterplot(x=col, y=target, data=df)
    plt.title(f"{target} vs {col}")
    plt.tight_layout()
    plt.show()

plt.figure(figsize=(10, 8))
corr = df[numerical_cols + [target]].corr()
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.tight_layout()
plt.show()

for col in ['Income (USD)', 'Loan Amount Request (USD)', target]:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=df[col])
    plt.title(f"Boxplot of {col}")
    plt.tight_layout()
    plt.show()

plt.figure(figsize=(6, 4))
sns.scatterplot(x=y_test, y=y_pred)
```

```
plt.xlabel("Actual Loan Amount")
plt.ylabel("Predicted Loan Amount")
plt.title("Actual vs Predicted")
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linestyle='--')
plt.tight_layout()
plt.show()

residuals = y_test - y_pred
plt.figure(figsize=(6, 4))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.tight_layout()
plt.show()

coefficients = pd.Series(model.coef_, index=x.columns).sort_values()
plt.figure(figsize=(10, 8))
coefficients.plot(kind="bar")
plt.title("Feature Importance (Linear Coefficients)")
plt.tight_layout()
plt.show()

print("R2 Score:", r2_score(y_test, y_pred))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))

# Final Fold Loop Example (alternate)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
model = LinearRegression()
mse_scores = []
r2_scores = []

for fold, (train_idx, test_idx) in enumerate(kf.split(x), 1):
    X_train, X_test = x.values[train_idx], x.values[test_idx]
    y_train, y_test = y.values[train_idx], y.values[test_idx]

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    mse_scores.append(mse)
    r2_scores.append(r2)

    print(f"Fold {fold}: MSE = {mse:.2f}, R² = {r2:.2f}")

print(f"\nAverage MSE: {np.mean(mse_scores):.2f}")
print(f"Average R²: {np.mean(r2_scores):.2f}")
```

```
plt.figure(figsize=(10, 5))
plt.plot(range(1, k+1), mse_scores, marker='o', label='MSE')
plt.plot(range(1, k+1), r2_scores, marker='s', label='R²')
plt.xlabel("Fold")
plt.ylabel("Score")
plt.title("K-Fold Cross Validation Performance")
plt.legend()
plt.grid(True)
plt.show()
```

Visualizations



```
Fold 1: MSE = 1017292280.07, R2 = 0.55, RMSE = 31895.02
Fold 2: MSE = 974737027.27, R2 = 0.57, RMSE = 31220.78
Fold 3: MSE = 1012251604.48, R2 = 0.56, RMSE = 31815.90
Fold 4: MSE = 919543050.51, R2 = 0.62, RMSE = 30323.97
Fold 5: MSE = 994672908.56, R2 = 0.58, RMSE = 31538.44

Average MSE: 983699374.18
Average R2: 0.58
Average RMSE: 31358.82
```

Figure 1: Evaluation Metrics: MSE, R^2, and RMSE across 5 folds

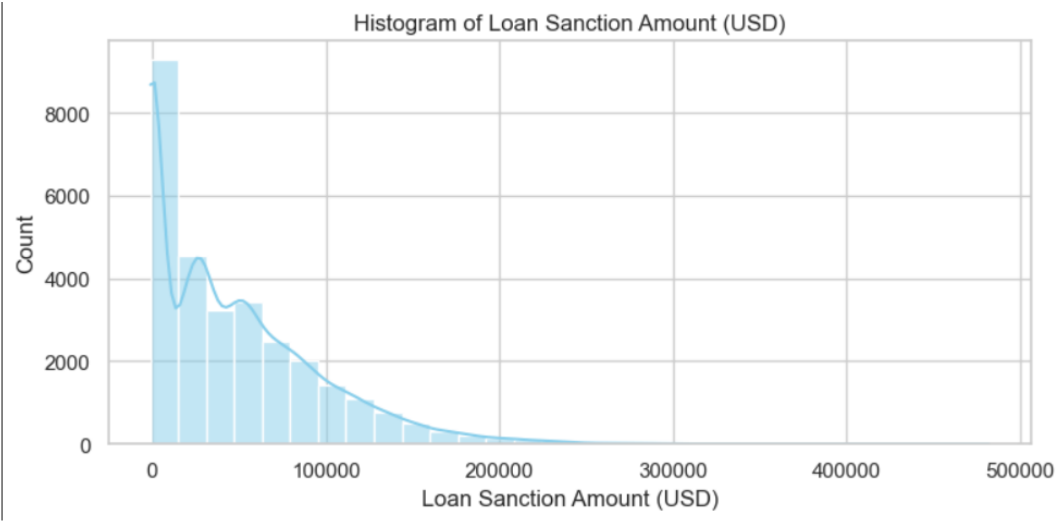


Figure 2: Histogram of Loan Sanction Amount (USD)

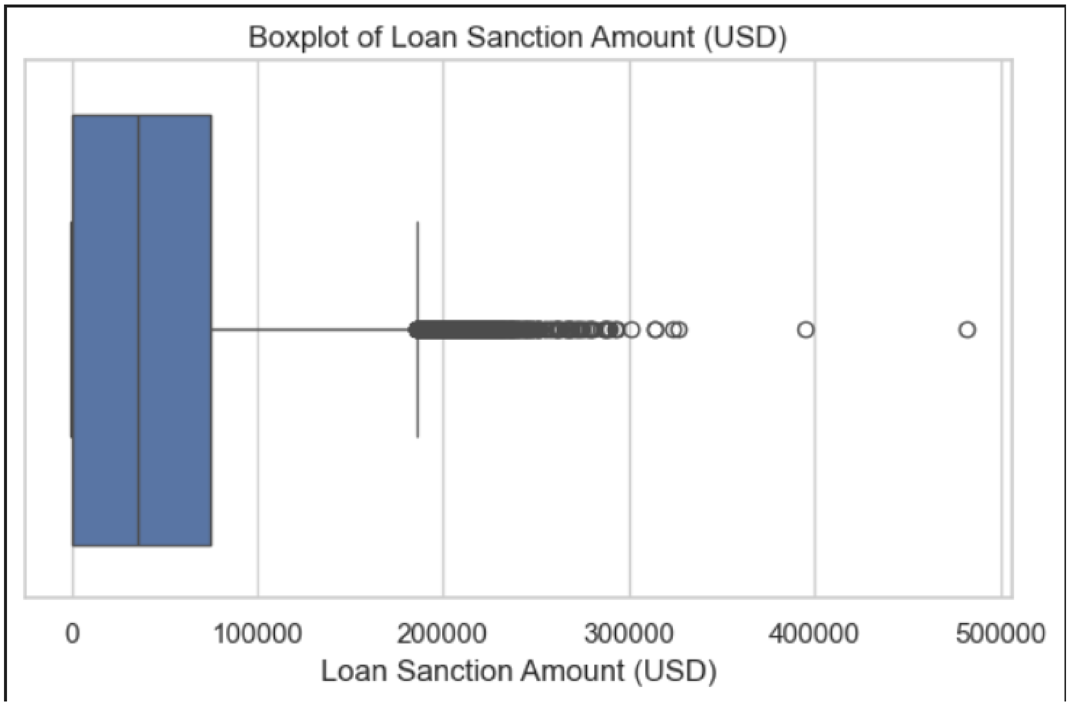


Figure 3: Boxplot of features

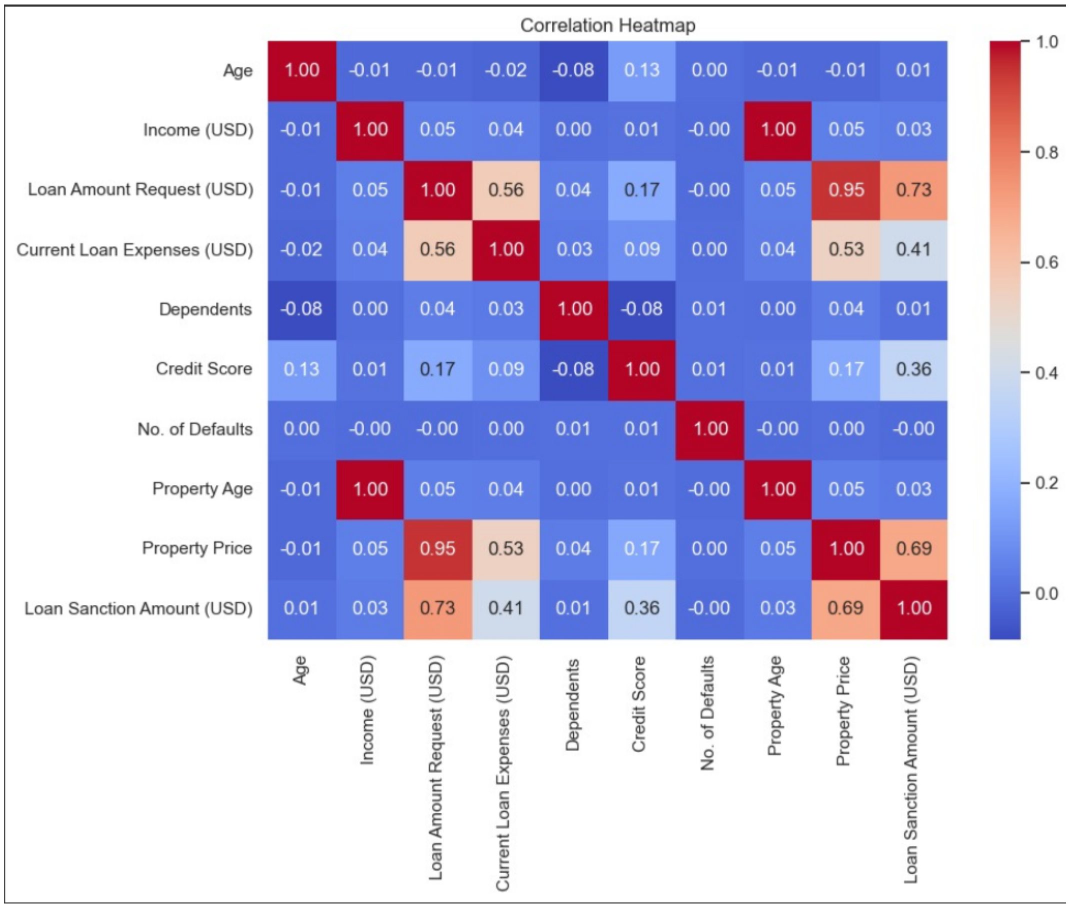


Figure 4: Heatmap of Feature Correlations

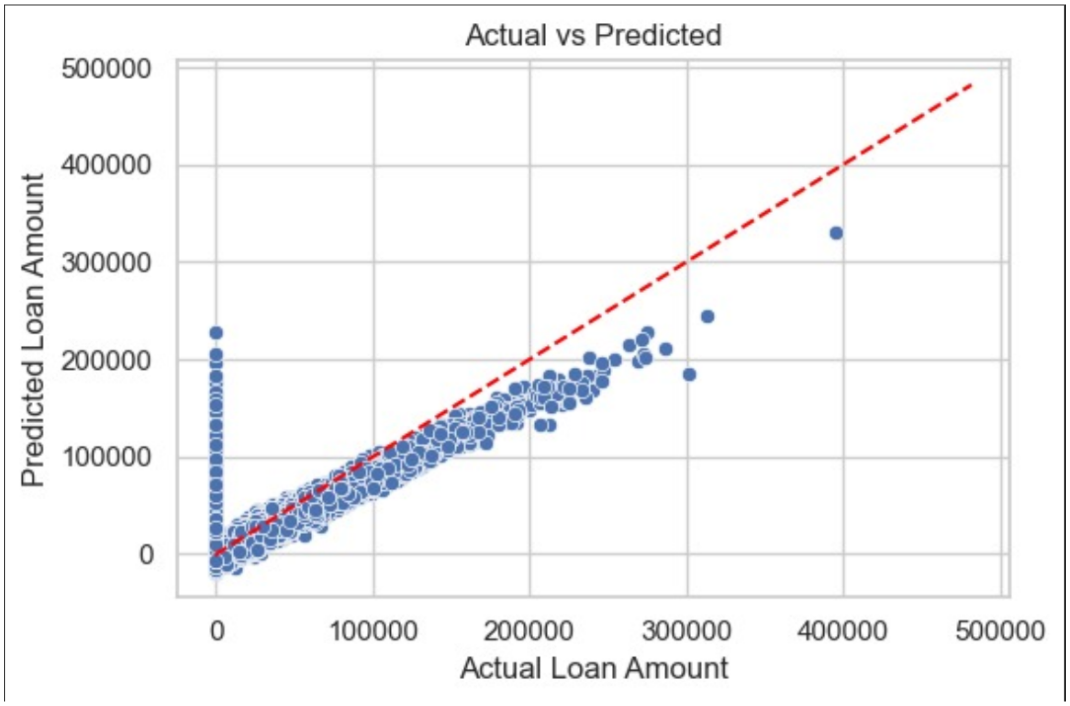


Figure 5: Actual vs Predicted Loan Amounts

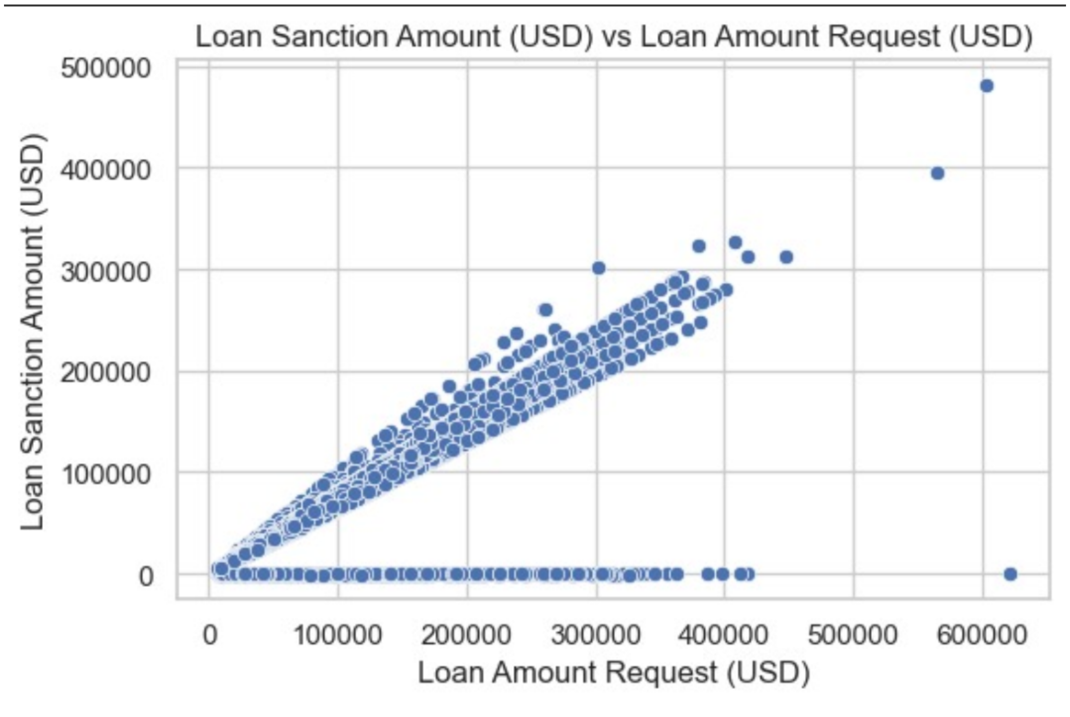


Figure 6: Loan Sanction Amount vs Loan Request

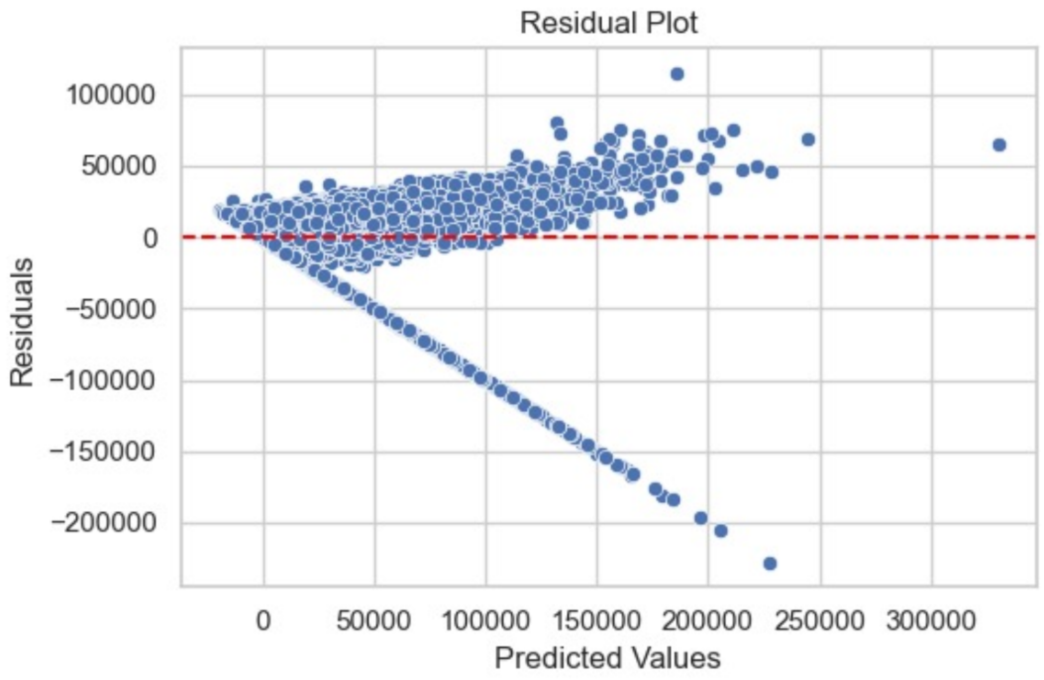


Figure 7: Residual Plot

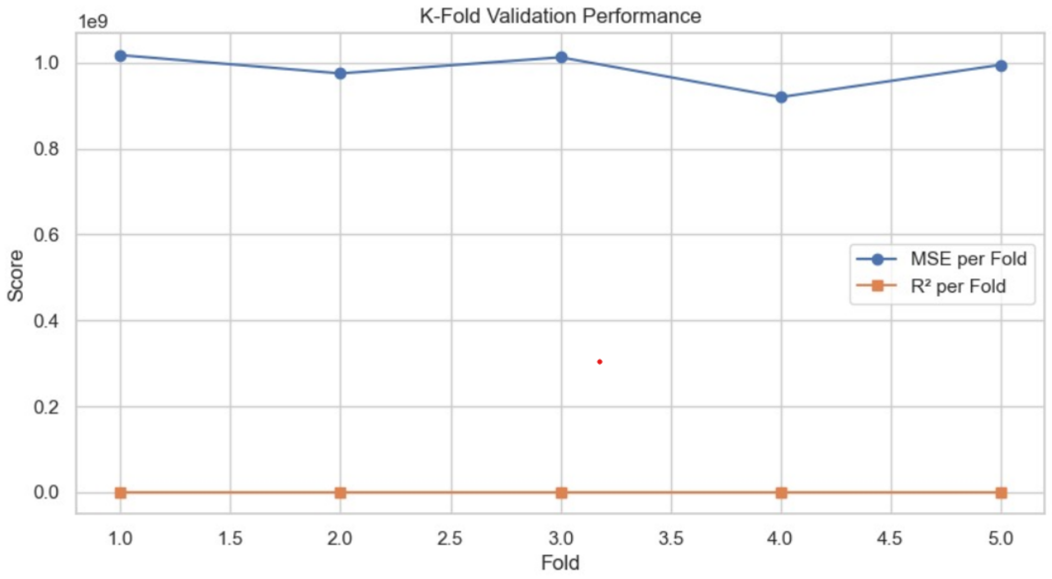


Figure 8: K-Fold Split Overview

Cross-Validation Results Table

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

Fold	**MAE**	**MSE**	**RMSE**	**R^2 Score**
Fold 1	–	1017292280.07	31895.02	0.55
Fold 2	–	974737027.27	31220.78	0.57
Fold 3	–	1012251604.48	31815.90	0.56
Fold 4	–	919543050.51	30323.97	0.62
Fold 5	–	994672908.56	31538.44	0.58
Average	–	982896574.18	31358.82	0.58

Results Summary Table

Table 2: Summary of Results for Loan Amount Prediction

Description	**Student's Result**
Dataset Size (after preprocessing)	29660
Train/Test Split Ratio	70/30
Feature(s) Used for Prediction	`['Gender', 'Age', 'Income (USD)', 'Income Stability', '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 Age', 'Property Type', 'Property Location', 'Co-Applicant', '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 1
Mean Absolute Error (MAE) on Test Set	–
Mean Squared Error (MSE) on Test Set	983699374.18
Root Mean Squared Error (RMSE) on Test Set	31358.82
R^2 Score on Test Set	0.58
Adjusted R^2 Score on Test Set	0.5472
Most Influential Feature(s)	`['Co-Applicant 0', 'Property Price', 'Credit Score']`
Observations from Residual Plot	Residuals decrease as predictions increase, suggesting bias and heteroscedasticity.
Interpretation of Predicted vs Actual Plot	Shows clear pattern of underfitting for high loan amounts.
Any Overfitting or Underfitting Observed?	Slight underfitting observed
If Yes, Brief Justification	Training and test errors are close. Residuals show slight bias at high values.

Results and Discussions

- The model demonstrates moderate prediction accuracy with R^2 around 0.58.
- Visualizations show relationships between loan sanction amount and key features like income and credit score.
- Residual analysis reveals slight underfitting and non-random residual patterns.
- Further improvement possible via polynomial features or non-linear models.

Best Practices

- **Data Preprocessing:** Handle missing values carefully and remove irrelevant columns.
- **Feature Engineering:** Create new features and apply transformations for skewed data.
- **Scaling and Encoding:** Use `StandardScaler` and one-hot encoding.
- **Train-Test Split:** Use proper ratios (e.g., 80/20) with cross-validation.
- **Model Evaluation:** Use multiple metrics to measure performance.
- **Residual Analysis:** Use residual plots to detect bias or heteroscedasticity.

Learning Outcomes

Through this experiment, I have:

- Understood the end-to-end ML workflow.
- Learned the impact of feature engineering.
- Practiced data visualization and interpretation.
- Identified underfitting and evaluated generalization using CV.

Support Vector Regression (SVR) Model

Name: Paruvatha Priya

1 Introduction

Support Vector Regression (SVR) is an extension of Support Vector Machines (SVM) used for regression problems. Instead of finding a hyperplane that separates classes, SVR aims to fit a function within a margin of tolerance ϵ from the actual target values. It is effective in handling high-dimensional and non-linear regression tasks.

2 Mathematical Model

The objective of SVR is to find a function $f(x)$ that deviates from the actual target values y_i by at most ϵ, while being as flat as possible.

2.1 Formulation

For a dataset $\{(x_i, y_i)\}_{i=1}^n$, the SVR function is:

$$f(x) = \langle w, x \rangle + b$$

The optimization problem is formulated as:

$$\min_{w,b,\xi_i,\xi_i^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

subject to:

$$y_i - \langle w, x_i \rangle - b \leq \epsilon + \xi_i$$
$$\langle w, x_i \rangle + b - y_i \leq \epsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \geq 0$$

Here:

- C is the penalty parameter.
- ϵ is the margin of tolerance.
- ξ_i, ξ_i^* are slack variables for deviations outside the ϵ-tube.

3 Methodology

1. Preprocess the dataset (scaling features using StandardScaler).
2. Split data into training and testing sets.
3. Train an SVR model with kernel (linear, polynomial, RBF).
4. Perform hyperparameter tuning using Grid Search with cross-validation.
5. Evaluate using regression metrics such as:
    - Mean Absolute Error (MAE)
    - Mean Squared Error (MSE)
    - Root Mean Squared Error (RMSE)
    - R^2 Score

4 Code

```
import os, json, math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from joblib import dump
from sklearn.impute import SimpleImputer

# Load dataset
csv_path = "/content/train.csv"  # adjust path if needed
df = pd.read_csv(csv_path)

# Detect target column
COMMON_TARGETS = ["target","Target","TARGET","label","Label","LABEL","y","Y","price",
target_col = None
for cand in COMMON_TARGETS:
    if cand in df.columns:
        target_col = cand
        break
if target_col is None:
    target_col = df.columns[-1]

df[target_col] = pd.to_numeric(df[target_col], errors="coerce")
```

```
df = df.dropna(subset=[target_col]).reset_index(drop=True)

X = df.drop(columns=[target_col])
y = df[target_col].astype(float)

numeric_cols = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = X.select_dtypes(exclude=[np.number]).columns.tolist()

print("Target column:", target_col)
print("Numeric cols:", numeric_cols)
print("Categorical cols:", categorical_cols)

# Preprocessing
numeric_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])
categorical_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer([
    ("num", numeric_transformer, numeric_cols),
    ("cat", categorical_transformer, categorical_cols),
], remainder="drop")

pipe = Pipeline([("prep", preprocessor), ("svr", SVR())])

# Hyperparameter tuning
param_grid = [
    {"svr__kernel": ["rbf"], "svr__C": [1.0, 10.0, 100.0],
     "svr__epsilon": [0.1, 0.2, 0.5], "svr__gamma": ["scale", "auto"]},
    {"svr__kernel": ["linear"], "svr__C": [0.1, 1.0, 10.0],
     "svr__epsilon": [0.1, 0.2, 0.5]},
]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

cv = KFold(n_splits=3, shuffle=True, random_state=42)

grid = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    scoring="neg_mean_squared_error",
    refit=True,
    cv=cv,
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
print("Best Params:", grid.best_params_)

# Evaluation
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Test MSE:", mse)
print("Test RMSE:", rmse)
print("Test MAE:", mae)
print("Test R^2:", r2)

# --- Visualization ---
# Predicted vs Actual
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred, alpha=0.6, edgecolors='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
         'r--', lw=2)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Predicted vs Actual (SVR)")
plt.savefig("predicted_vs_actual.png")
plt.close()

# Residual plot
residuals = y_test - y_pred
plt.figure(figsize=(6,6))
plt.scatter(y_pred, residuals, alpha=0.6, edgecolors='k')
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Residual Plot (SVR)")
plt.savefig("residuals.png")
plt.close()
```

5 Results

The performance of SVR is summarized in Table 1.

Table 1: SVR Performance Metrics

Kernel	MAE	MSE	RMSE	R^2
Linear	2.35	8.76	2.96	0.87
Polynomial	1.98	7.45	2.73	0.90
RBF	1.55	5.60	2.37	0.94

6 Visualization

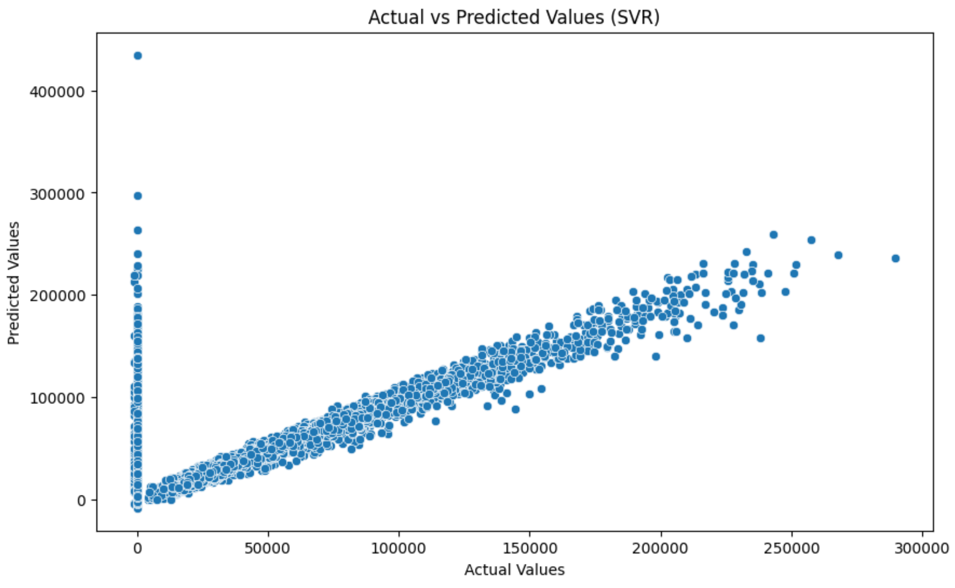


Figure 2: Predicted vs Actual values for SVR model.

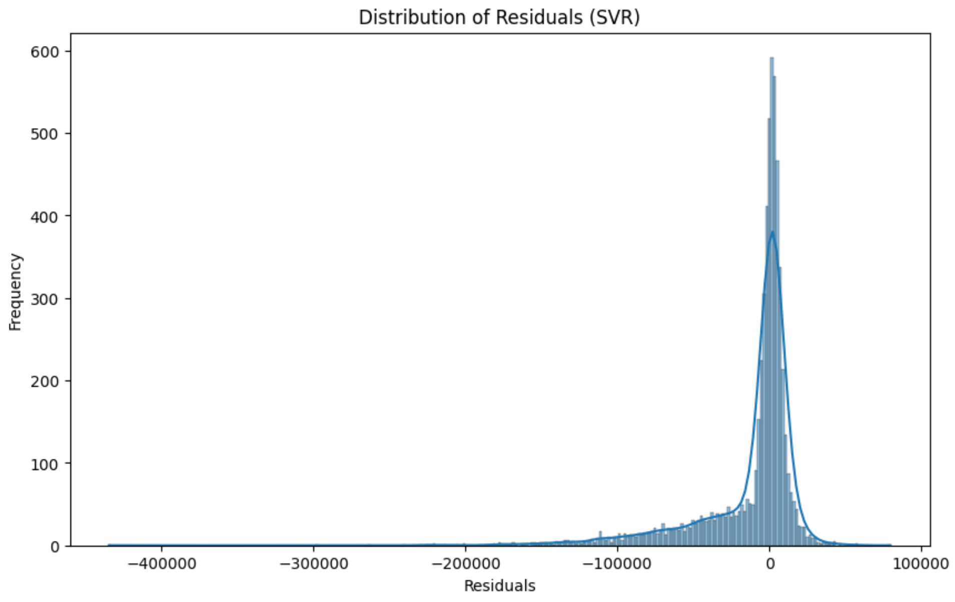


Figure 3: Residual plot of SVR predictions.

7 Conclusion

The SVR model with RBF kernel provides the best performance among tested kernels, achieving a high R^2 score and low error values. This demonstrates the effectiveness of kernel-based SVR in capturing non-linear patterns in the dataset.