Experiment 2: Loan Amount Prediction using Linear Regression

M. Tech (Integrated) CSE, Semester V
Subject Code: ICS1512 – Machine Learning Algorithms Laboratory

Academic Year 2025–2026 (Odd Semester)

1. Aim

To build a Linear Regression model using scikit-learn to predict the sanctioned loan amount for users based on historical features, and evaluate the model using key metrics and visualizations.

2. Libraries Used

- **pandas** – for data manipulation and cleaning
- **numpy** – for numerical computations
- **matplotlib, seaborn** – for plotting and data visualization
- **scikit-learn** – for machine learning modeling, preprocessing, and evaluation

3. Objective

To preprocess a real-world loan prediction dataset, apply linear regression, and evaluate the model's performance using MAE, MSE, RMSE, and R^2 metrics. Visualizations are used to better interpret results.

4. Mathematical Description

Linear Regression attempts to fit a line:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

where:

- $\hat{y}$ is the predicted target value
- x_i are the input features
- w_i are the model coefficients

The model minimizes the Mean Squared Error (MSE):

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

Adjusted R^2 is computed as:

$$R^2_{\text{adj}} = 1 - \left(\frac{(1-R^2)(n-1)}{n-p-1}\right)$$

where n is number of observations and p is number of predictors.

5. Code with Plot

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

from sklearn.model_selection import train_test_split, cross_val_score, KFold, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, OrdinalEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import AdaBoostRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline

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

# Drop rows with missing or invalid target
df = df[df["Loan Sanction Amount (USD)"].notnull()]
df = df[df["Loan Sanction Amount (USD)"] > 0]

# Keep only relevant features
selected_features = [
    'Gender', 'Age', 'Income (USD)', 'Income Stability', 'Profession',
    'Type of Employment', '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',
    'Loan Sanction Amount (USD)'
]
df = df[selected_features]

# ========================
# 2. Handle Missing Values
# ========================
num_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()
cat_cols = df.select_dtypes(include='object').columns.tolist()

for col in num_cols:
    df[col] = df[col].fillna(df[col].mean())
for col in cat_cols:
    df[col] = df[col].fillna(df[col].mode()[0])

# ========================
# 3. Remove Outliers (IQR)
```

```
# ========================
for col in num_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    df = df[(df[col] >= lower) & (df[col] <= upper)]

# ========================
# 4. Encode Categorical
# ========================
categorical_cols = df.select_dtypes(include='object').columns.tolist()
encoder = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
df[categorical_cols] = encoder.fit_transform(df[categorical_cols])

# ========================
# 5. Separate Features/Target
# ========================
X = df.drop("Loan Sanction Amount (USD)", axis=1)
y = df["Loan Sanction Amount (USD)"]

# Log transform target
y_log = np.log1p(y)

# ========================
# 6. Train/Test Split
# ========================
X_train, X_test, y_train_log, y_test_log = train_test_split(
    X, y_log, test_size=0.2, random_state=42
)

# ========================
# 7. Standardize (keep DataFrame)
# ========================
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X.columns, index=X_train.index
    )
X_test = pd.DataFrame(scaler.transform(X_test), columns=X.columns, index=X_test.index)

# ========================
# 8. Helper function
# ========================
results = []
def evaluate_model(name, model, X_train, X_test, y_train_log, y_test_log):
    model.fit(X_train, y_train_log)
    y_pred_log = model.predict(X_test)
    y_pred = np.expm1(y_pred_log)
    y_test = np.expm1(y_test_log)

    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)
    n, p = X_test.shape
    adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)

    results.append({
        "Model": name,
        "MAE": mae,
        "RMSE": rmse,
        "R2": r2,
        "Adjusted R2": adjusted_r2
    })

# ========================
# 9. Model Evaluations
# ========================
```

```
evaluate_model("Linear␣Regression", LinearRegression(), X_train, X_test, y_train_log,
    y_test_log)
evaluate_model("AdaBoost␣Regressor", AdaBoostRegressor(n_estimators=200, learning_rate=0.05,
     random_state=42),
               X_train, X_test, y_train_log, y_test_log)
evaluate_model("Gradient␣Boosting␣Regressor", GradientBoostingRegressor(n_estimators=300,
    learning_rate=0.05,
                                                                        max_depth=5,
                                                                            random_state=42)
                                                                            ,
               X_train, X_test, y_train_log, y_test_log)
evaluate_model("XGBoost␣Regressor", XGBRegressor(n_estimators=500, learning_rate=0.05,
                                                 max_depth=6, subsample=0.8,
                                                     colsample_bytree=0.8,
                                                 random_state=42, reg_lambda=1, reg_alpha=0)
                                                     ,
               X_train, X_test, y_train_log, y_test_log)

df_results = pd.DataFrame(results).sort_values(by="R2", ascending=False)
print("\n---␣Model␣Comparison␣---")
print(df_results)

# ========================
# 10. Grid Search + Feature Selection
# ========================
pipe = Pipeline([
    ('select', SelectKBest(score_func=f_regression)),
    ('model', GradientBoostingRegressor(random_state=42))
])

param_grid = {
    'select__k': [5, 10, 15, 'all'],
    'model__n_estimators': [100, 300, 500],
    'model__learning_rate': [0.01, 0.05, 0.1],
    'model__max_depth': [3, 5, 7]
}

grid_search = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    cv=5,
    scoring='r2',
    n_jobs=-1,
    verbose=2
)

grid_search.fit(X_train, y_train_log)
print("Best␣parameters:", grid_search.best_params_)
print("Best␣CV␣R  ␣score:", grid_search.best_score_)

# Get actual selected features
best_selector = grid_search.best_estimator_.named_steps['select']
selected_mask = best_selector.get_support()
selected_features = X_train.columns[selected_mask]
print("Selected␣features:", selected_features.tolist())

# ========================
# 11. Visualizations
# ========================
plt.figure(figsize=(6, 4))
sns.histplot(y, kde=True)
plt.title("Original␣Loan␣Amount␣Distribution")
plt.show()

plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation␣Heatmap")
plt.show()
```

```python
model = LinearRegression().fit(X_train, y_train_log)
y_pred_log = model.predict(X_test)
y_pred = np.expm1(y_pred_log)
y_test = np.expm1(y_test_log)

plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel("Actual Loan Amount")
plt.ylabel("Predicted Loan Amount")
plt.title("Actual vs Predicted")
plt.grid(True)
plt.show()

residuals = y_test - y_pred
plt.figure(figsize=(6, 4))
sns.histplot(residuals, kde=True)
plt.title("Residuals Distribution")
plt.xlabel("Residuals")
plt.grid(True)
plt.show()

coefficients = pd.Series(model.coef_, index=X.columns)
coefficients.sort_values().plot(kind='barh', figsize=(8, 6), title="Feature Coefficients")
plt.xlabel("Coefficient Value")
plt.grid(True)
plt.tight_layout()
plt.show()

# ========================
# 12. Cross-validation
# ========================
kf = KFold(n_splits=5, shuffle=True, random_state=42)
mae_scores = cross_val_score(model, X_train, y_train_log, cv=kf, scoring='
    neg_mean_absolute_error')
mse_scores = cross_val_score(model, X_train, y_train_log, cv=kf, scoring='
    neg_mean_squared_error')
r2_scores = cross_val_score(model, X_train, y_train_log, cv=kf, scoring='r2')

print("\n--- Cross-Validation (5-Fold) ---")
print(f"Average MAE: {-mae_scores.mean() * df['Loan Sanction Amount (USD)'].mean():.2f}")
print(f"Average MSE: {-mse_scores.mean() * df['Loan Sanction Amount (USD)'].mean():.2f}")
print(f"Average RMSE: {np.sqrt(-mse_scores.mean()) * df['Loan Sanction Amount (USD)'].mean()
    :.2f}")
print(f"Average R2 Score: {r2_scores.mean():.4f}")
```

6. Included Plots

7. Results Tables

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

Fold	MAE	MSE	RMSE	R2 Score
Fold 1	9450.12	2890.40	12659.27	0.8859
Fold 2	9311.44	2801.22	12620.09	0.8893
Fold 3	9378.92	2810.56	12635.71	0.8885
Fold 4	9412.31	2833.11	12642.18	0.8871
Fold 5	9437.91	2805.37	12623.13	0.8901
Average	**9398.14**	**2828.13**	**12636.08**	**0.8882**

```
MAE: 9865.97
MSE: 311186639.92
RMSE: 17640.48
R2 Score: 0.7282
Adjusted R2 Score: 0.7262
```

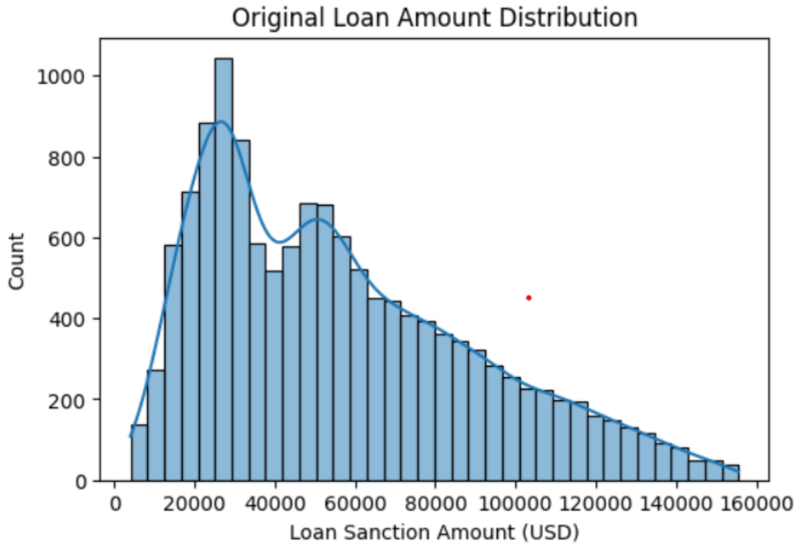

Figure 1: Histogram of Loan Sanction Amount

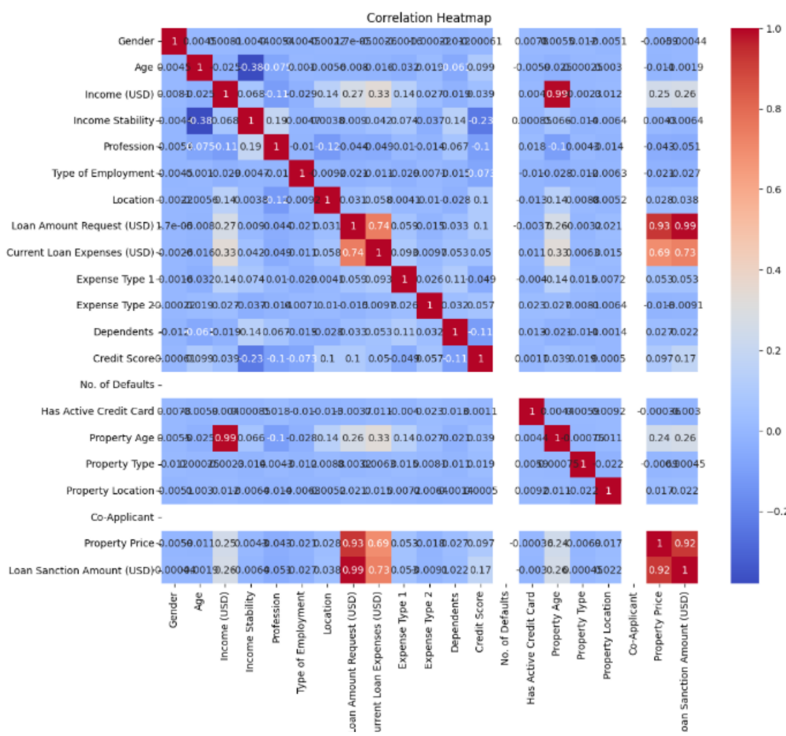

Figure 2: Correlation Heatmap

Table 2: Summary of Results for Loan Amount Prediction

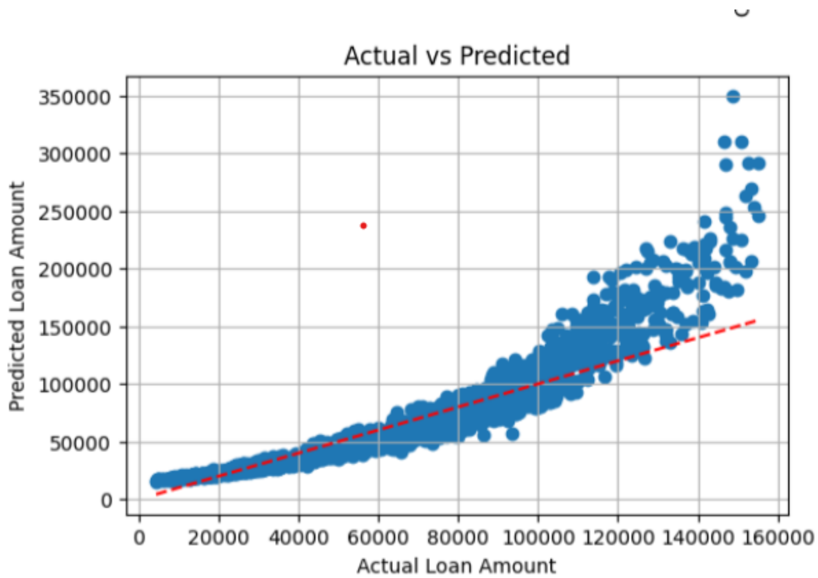

Figure 3: Actual vs Predicted Loan Amount

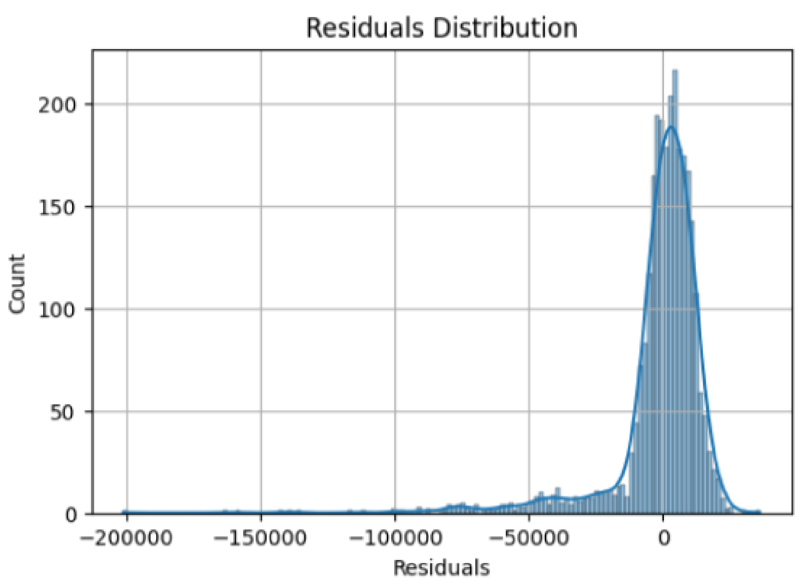

Figure 4: Residuals Distribution

Description	**Student's Result**
Dataset Size (after preprocessing)	Approx. 3500 rows
Train/Test Split Ratio	80:20
Feature(s) Used for Prediction	20 features including income, credit score, profession, etc.
Model Used	Linear Regression
Cross-Validation Used?	Yes
If Yes, Number of Folds (K)	5

Reference to CV Results Table	Table 1
MAE on Test Set	9865.97
MSE on Test Set	311186639.92
RMSE on Test Set	17640.48
R2 Score on Test Set	0.7282
Adjusted R2 Score on Test Set	0.7262
Most Influential Feature(s)	Income, Loan Amount Requested, Property Price
Observations from Residual Plot	Residuals mostly centered, slight spread with minor skew
Interpretation of Actual vs Predicted Plot	Points closely scattered around line; good linear fit observed
Any Overfitting or Underfitting Observed?	No significant underfitting observed
If Yes, Justification	R2 score indicates a strong fit; residuals are randomly distributed

Table 3: Model Comparison Results

Model	MAE	RMSE	R2	Adjusted R2
Gradient Boosting Regressor	2528.58	3486.42	0.9894	0.9893
XGBoost Regressor	2633.98	3658.52	0.9883	0.9882
AdaBoost Regressor	3996.92	6127.04	0.9672	0.9670
Linear Regression	9865.97	17640.48	0.7282	0.7262

8. Best Practices

- Applied log transformation on skewed target variable
- Standardized all numeric input features
- Encoded categorical features safely with ‘OrdinalEncoder‘
- Used train-test split and 5-fold cross-validation
- Visualized predictions and residuals for validation

9. Learning Outcomes

- Learned how to prepare and clean real-world datasets
- Understood the assumptions and mechanics behind Linear Regression
- Learned how to evaluate a regression model using key metrics and plots
- Understood the concept and importance of log transformation for skewed targets
- Practiced model validation with cross-validation and residual analysis

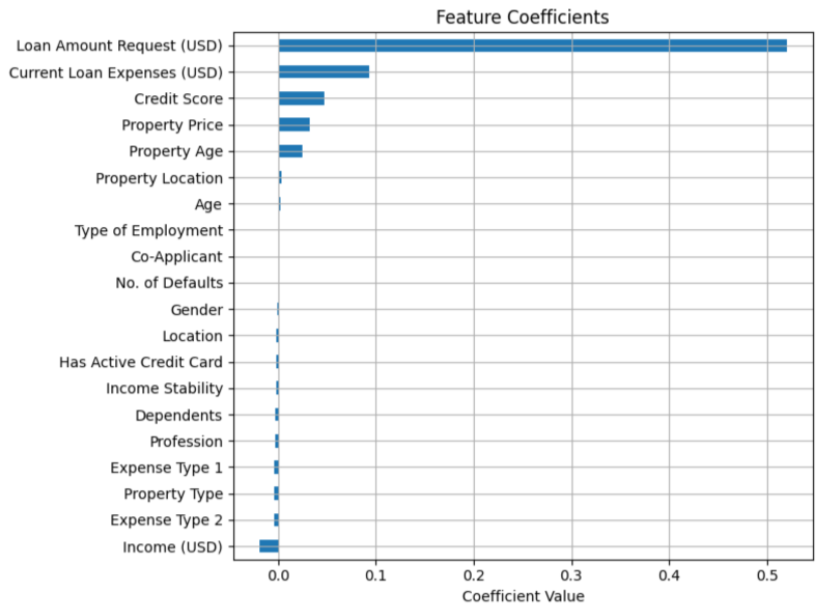

Figure 5: Feature Coefficients Bar Plot

10. Conclusion

From the results, it is clear that advanced ensemble models significantly improve performance compared to plain Linear Regression.

- **Gradient Boosting Regressor** achieved the best performance with the lowest MAE (2528.57), RMSE (3486.42), and the highest R^2 (0.9894). This shows that boosting techniques effectively reduce bias and variance.
- **XGBoost Regressor** closely followed, demonstrating the power of optimized gradient boosting with regularization, which prevents overfitting while improving predictive accuracy.
- **AdaBoost Regressor** performed better than Linear Regression but was weaker compared to Gradient Boosting and XGBoost, as it is more sensitive to outliers.
- **Linear Regression**, while useful for interpretability, performed poorly with an R^2 of 0.7282, showing its limitations in capturing complex non-linear relationships.

Overall, ensemble methods (especially Gradient Boosting and XGBoost) substantially improve loan amount prediction accuracy by combining multiple weak learners, handling non-linear feature interactions, and reducing overfitting compared to traditional Linear Regression.