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: 25/07/2025**

Experiment 2 : Loan Amount Prediction using Linear Regression

Aim:

To predict the loan amount using Linear Regression using historical data and measure the model's performance

Libraries used:

- Numpy
- Pandas
- Matplotlib
- Seaborn
- sklearn

Mathematical Description

In this experiment, Linear Regression is used to predict the loan sanction amount based on several input features.
The mathematical model for Linear Regression is represented as:

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

where:

- y is the dependent variable (Loan Sanction Amount),
- $x_1, x_2, \ldots, x_n$ are independent variables (features such as Age, Income, Credit Score, etc.),
- β_0 is the intercept,
- $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients of the features,
- ε is the error term representing noise or unexplained variance.

The coefficients β are estimated by minimizing the Residual Sum of Squares (RSS), defined as:

$$RSS = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} \left(y_i - \left(\beta_0 + \sum_{j=1}^{n} \beta_j x_{ij} \right) \right)^2$$

where m is the number of observations. The model is evaluated using the following metrics:

- **Mean Absolute Error (MAE)** – average absolute difference between actual and predicted values,
- **Mean Squared Error (MSE)** – average squared difference between actual and predicted values,
- **Root Mean Squared Error (RMSE)** – square root of MSE, providing error in original units,
- **R-squared (R^2)** – proportion of variance explained by the model,
- **Adjusted** R^2 – adjusted for number of predictors to prevent overfitting.

Objectives :

Apply Linear Regression to predict the loan amount sanctioned to users using the dataset provided, visualize and interpret the results to gain insights into the model performance

Importing Required Libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, KFold, learning_curve
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

Loading the dataset:

```
"' Load data "'
df = pd.read_csv("C:/Users/Kavi/Downloads/train.csv/train.csv")
df.head()
```

Output:

	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	Property ID	Property Age	Property Type	Property Location	Co-Applicant	Property Price	Loan Sanction Amount (USD)
0	C-36995	Frederica Shealy	F	56	1933.05	Low	Working	Sales staff	Semi-Urban	72809.58	...	809.44	0	NaN	746	1933.05	4	Rural	1	119933.46	54607.18
1	C-33999	America Calderone	M	32	4952.91	Low	Working	NaN	Semi-Urban	46837.47	...	780.40	0	Unpossessed	608	4952.91	2	Rural	1	54791.00	37469.98
2	C-3770	Rosetta Verne	F	65	988.19	High	Pensioner	NaN	Semi-Urban	45593.04	...	833.15	0	Unpossessed	546	988.19	2	Urban	0	72440.58	36474.43
3	C-26480	Zoe Chitty	F	65	NaN	High	Pensioner	NaN	Rural	80057.92	...	832.70	1	Unpossessed	890	NaN	2	Semi-Urban	1	121441.51	56040.54
4	C-23459	Afton Venema	F	31	2614.77	Low	Working	High skill tech staff	Semi-Urban	113858.89	...	745.55	1	Active	715	2614.77	4	Semi-Urban	1	208567.91	74008.28

5 rows × 24 columns

Figure 1: Displaying first 5 rows

Exploratory Data Analysis:

- **Descriptive statistics**:

  print(df.info())
  print(df.describe())

```
<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
...
 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
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

None
```

	Age	Income (USD)	Loan Amount Request (USD)	Current Loan Expenses (USD)	Dependents	Credit Score	No. of Defaults
count	30000.000000	2.542400e+04	30000.000000	29828.000000	27507.000000	28297.000000	30000.000000
mean	40.092300	2.630574e+03	88826.333855	400.936876	2.253027	739.885381	0.193933
std	16.045129	1.126272e+04	59536.949605	242.545375	0.951162	72.163846	0.395384
min	18.000000	3.777000e+02	6048.240000	-999.000000	1.000000	580.000000	0.000000
25%	25.000000	1.650457e+03	41177.755000	247.667500	2.000000	681.880000	0.000000
50%	40.000000	2.222435e+03	75128.075000	375.205000	2.000000	739.820000	0.000000
75%	55.000000	3.090593e+03	119964.605000	521.292500	3.000000	799.120000	0.000000
max	65.000000	1.777460e+06	621497.820000	3840.880000	14.000000	896.260000	1.000000

	Property ID	Property Age	Property Type	Co-Applicant	Property Price	Loan Sanction Amount (USD)
count	30000.000000	2.515000e+04	30000.000000	30000.000000	3.000000e+04	29660.000000
mean	501.934700	2.631119e+03	2.460067	-4.743867	1.317597e+05	47649.342208
std	288.158086	1.132268e+04	1.118562	74.614593	9.354955e+04	48221.146686
min	1.000000	3.777000e+02	1.000000	-999.000000	-9.990000e+02	-999.000000
25%	251.000000	1.650450e+03	1.000000	1.000000	6.057216e+04	0.000000
50%	504.000000	2.223250e+03	2.000000	1.000000	1.099936e+05	35209.395000
75%	751.000000	3.091408e+03	3.000000	1.000000	1.788807e+05	74261.250000
max	999.000000	1.777460e+06	4.000000	1.000000	1.077967e+06	481907.320000

Figure 2: Descriptive Statistics

Descriptive statistics summarize and organize the characteristics of a dataset, providing insights into its **central tendency, dispersion, and distribution**. Key metrics such as **mean, median, mode, standard deviation, minimum, and maximum values** offer a foundational understanding of each feature's behavior. These statistics are essential for identifying data quality issues, such as **skewness, outliers, or missing values**, and guide appropriate preprocessing steps before advanced modeling.

- **Visualizing numerical features distribution:**

  df.drop(['Customer ID', 'Name', 'Property ID'], axis=1, inplace=True)
  numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
  df[numeric_cols].hist(figsize=(15, 10), bins=30)
  plt.suptitle("Initial Distribution of Numerical Features (Raw Data)")
  plt.show()

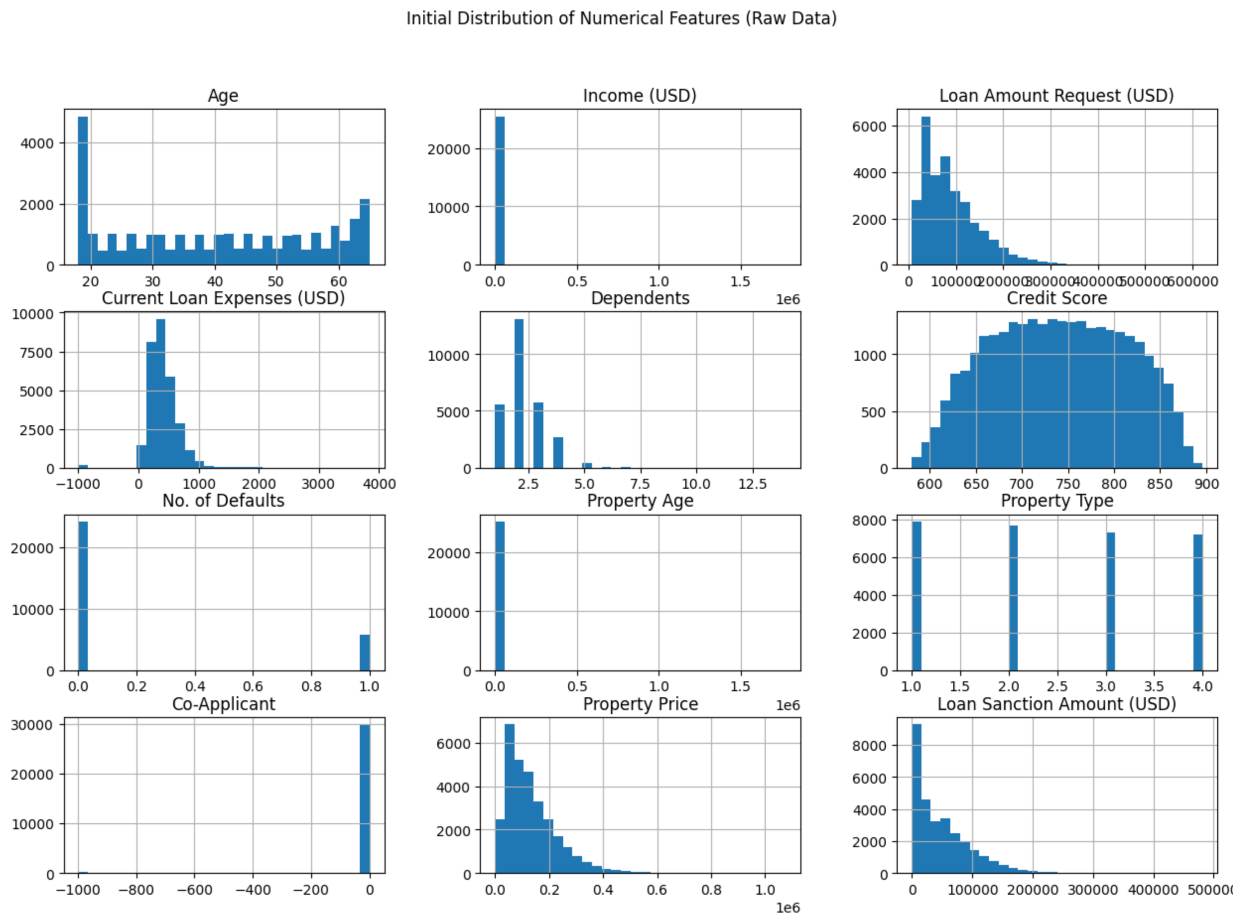


Figure 3: Numerical features distribution

Summary:

- Age: Bimodal, possible outliers.
- Income: Strong right skew, heavy outliers.
- Loan Amount Request/Sanction: Right-skewed, lower values dominate.
- Loan Expenses: Nearly normal, mild skew.
- Dependents: Discrete, mode at 0.
- Credit Score: Bell-shaped, well-centered.
- Defaults / Co-Applicant: Binary, mostly 0s.
- Property Age: Right-skewed, most are new.
- Property Type: Categorical (4 types).
- Property Price: Right-skewed, wide range.

- **Removing unneeded columns and plotting correlation heatmap:**

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

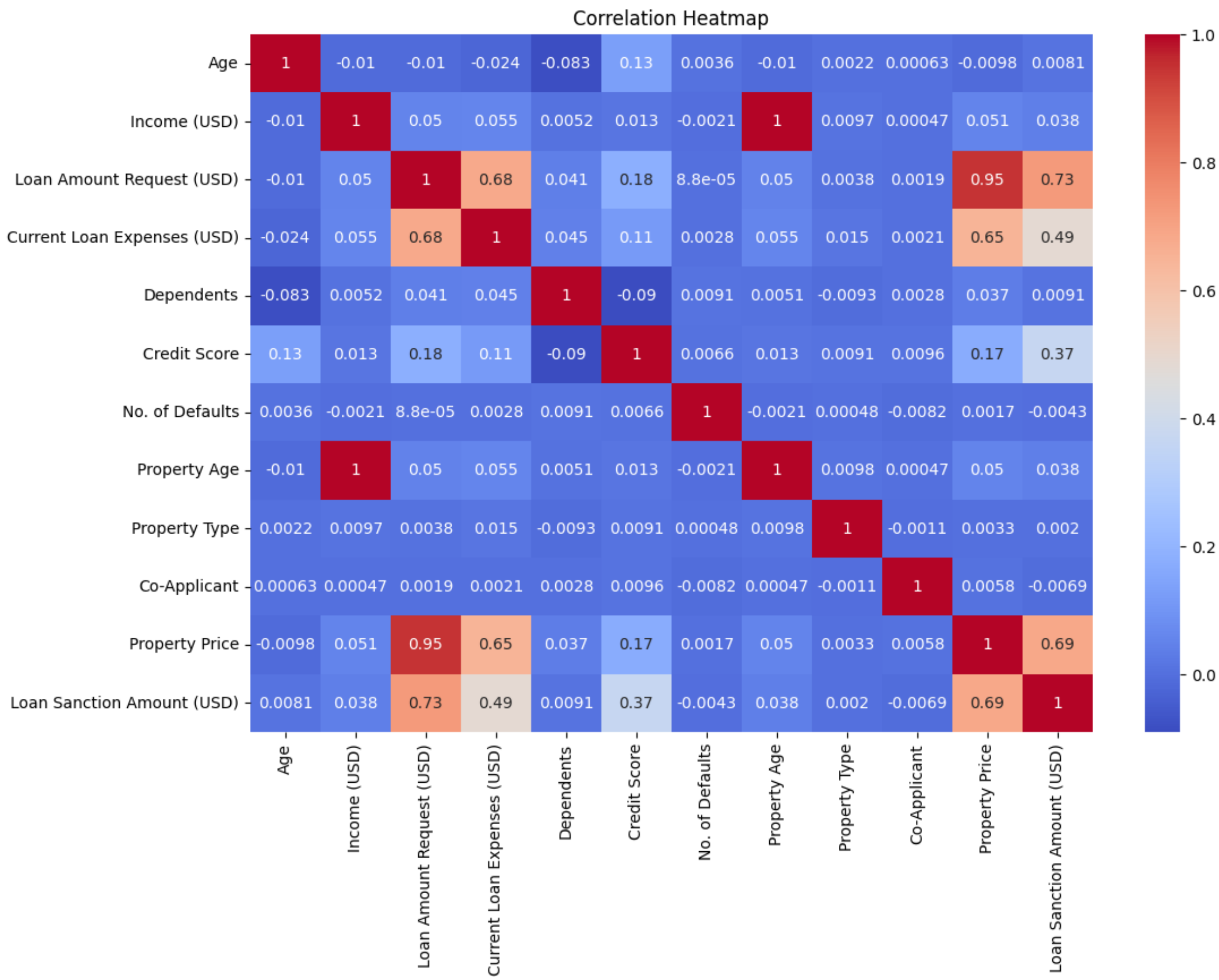


Figure 4: Correlation Heatmap

Highly Correlated features with Target: Loan Amount Request, Property price, Loan Expenses
Moderate Correlation: Current Loan Expenses with Credit Score
Negligibly Correlated features with Target : Age, Income, Dependents, Property Age, Co-Applicant, No.of Defaults

- **Outlier Detection and Removal**

```
”’ Boxplots for outlier detection ”’
num_cols = df.select_dtypes(include=[’int64’,’float64’]).columns
for col in num_cols:
    plt.figure(figsize=(6, 2))
    sns.boxplot(data=df, x=col)
    plt.title(f”Boxplot for col”)
    plt.show()



”’ Outlier removal using IQR method ”’
Q1 = df[num_cols].quantile(0.25)
Q3 = df[num_cols].quantile(0.75)
IQR = Q3 - Q1
df = df[~((df[num_cols] <(Q1 - 1.5 * IQR)) |(df[num_cols] > (Q3+1.5*IQR))).any(axis = 1)]
```

- **Data Cleaning and Preprocessing:**

```
”’ Recompute column types after dropping ”’
num_cols = df.select_dtypes(include=[’int64’, ’float64’]).columns.tolist()
cat_cols = df.select_dtypes(include=’object’).columns.tolist()
ordinal_cols = [’Income Stability’, ’Location’, ’Has Active Credit Card’,’Expense Type 1’,’Expense Type 2’]
cat_cols = [col for col in cat_cols if col not in ordinal_cols]

num_cols.remove(’Loan Sanction Amount (USD)’)
df = df[df[’Loan Sanction Amount (USD)’] > 0].reset_index(drop=True)
X = df.drop(’Loan Sanction Amount (USD)’, axis=1)
y = df[’Loan Sanction Amount (USD)’]

numeric_transformer = Pipeline(steps=[(’imputer’, SimpleImputer(strategy=’mean’)),(’scaler’, StandardScaler())])

categorical_transformer = Pipeline(steps=[(’imputer’, SimpleImputer(strategy=’most_frequent’)),(’encode
OneHotEncoder(handle_unknown=’ignore’))])

ordinal_transformer = Pipeline(steps=[(’imputer’, SimpleImputer(strategy=’most_frequent’)),
(’ordinal’, OrdinalEncoder())])

”’ Update categorical transformer”’
preprocessor = ColumnTransformer(transformers=[(’num’, numeric_transformer, num_cols),(’ord’,
ordinal_transformer, ordinal_cols), (’cat’, categorical_transformer, list(set(cat_cols) - set(ordinal_cols)))])
```

- **Learning Curve : Shows model performance with increasing data**

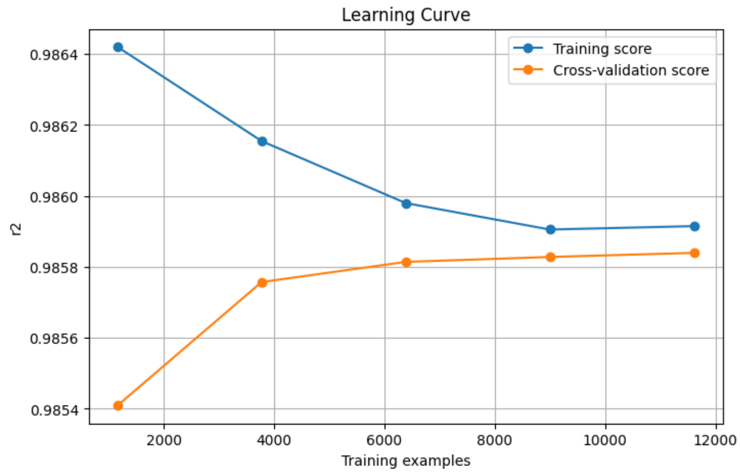


Figure 5: Training vs Cross Validation

Low Bias : Small gap b/w training and validation as the number oof data increases = not much underfitting
Low Variance : Training and validation scores shrinks as data increases = Not overly sensitive to training data
Model Generalization : Both lines converge around 0.985 R^2 = Good Generalization

- **K-Fold Cross Validation:**

```
"' Define the full pipeline "'
pipeline = Pipeline(steps=[('preprocessor', preprocessor),('regressor', LinearRegression())])

"' K-Fold CV "'
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
scores = []
mse_scores = []
mae_scores = []
rmse_scores = []
for train_idx, val_idx in kfold.split(X):
    X_train_fold, X_val_fold = X.iloc[train_idx], X.iloc[val_idx]
    y_train_fold, y_val_fold = y.iloc[train_idx], y.iloc[val_idx]

    pipeline.fit(X_train_fold, y_train_fold)
    y_pred = pipeline.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)
    print(f'MAE : {mae:.2f} , MSE : {mse:.2f} , RMSE : {rmse:.2f} , R2 : {r2:.2f}')
    scores.append(r2)
    mae_scores.append(r2)
    mse_scores.append(r2)
    rmse_scores.append(r2)

print(f"Average R² across 5 folds: np.mean(scores):.4f")
print(f"Average MAE across 5 folds: np.mean(scores):.4f")
print(f"Average MSE across 5 folds: np.mean(scores):.4f")
print(f"Average RMSE across 5 folds: np.mean(scores):.4f")
```

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

Fold	**MAE**	**MSE**	**RMSE**	**R^2 Score**
Fold 1	3129.42	18551597.737	4307.16	0.99
Fold 2	3079.05	18188836.99	4264.84	0.99
Fold 3	3115.78	18223159.98	4268.86	0.99
Fold 4	3152.64	18691139.54	4323.33	0.99
Fold 5	3173.19	19185272.34	4380.10	0.99
Average	3130.0141	18568001.3171	4308.8556	0.9858

- **Train/Validate/Test:**

```
# Step 1: Train/Test Split (80% train, 20% test)
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 2: Further split training into Train/Validation (75% train, 25% val of the 80%)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42)
# Final Split: 60% train, 20% val, 20% test

# Step 3: Fit the pipeline on training data
pipeline.fit(X_train, y_train)

# Step 4: Predict on validation data
y_val_pred = pipeline.predict(X_val)

# Step 5: Validation metrics
val_mse = mean_squared_error(y_val, y_val_pred)
val_mae = mean_absolute_error(y_val, y_val_pred)
val_rmse = np.sqrt(val_mse)
val_r2 = r2_score(y_val, y_val_pred)

print("----- Validation Metrics -----")
print(f"Validation MAE: {val_mae:.2f}")
print(f"Validation RMSE: {val_rmse:.2f}")
print(f"Validation R²: {val_r2:.4f}")

# Step 6: Final training on combined train+val for final test eval (optional)
# pipeline.fit(X_temp, y_temp)  # Uncomment if you want final model retrained on more data

# Step 7: Predict on test data
y_test_pred = pipeline.predict(X_test)

# Step 8: Test metrics
test_mse = mean_squared_error(y_test, y_test_pred)
test_mae = mean_absolute_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)
test_r2 = r2_score(y_test, y_test_pred)

print("\n----- Test Metrics -----")
print(f"Test MAE: {test_mae:.2f}")
print(f"Test RMSE: {test_rmse:.2f}")
print(f"Test R²: {test_r2:.4f}")
```

Figure 6: Training, Validating and Testing

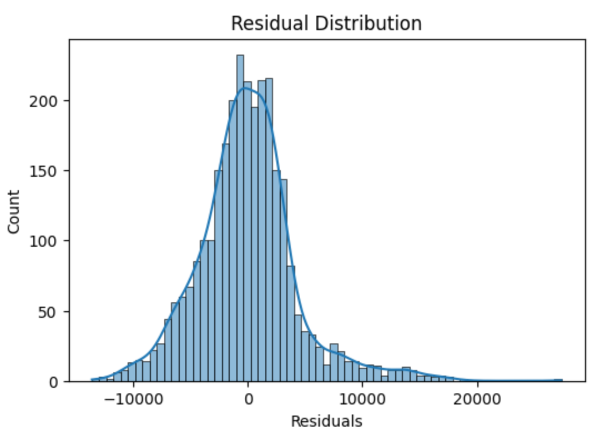
Phase	MAE	MSE	RMSE	R^2 Score
Validation	3148.68	18439475.88	4294.12	0.9862
Testing	3126.74	18565888.59	4308.82	0.9855

Table 2: Validation/ Test Metrics - Linear Regression

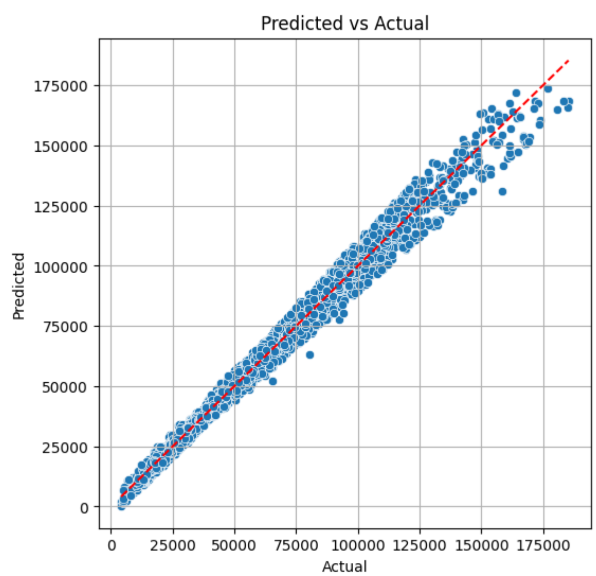
- **Residual Plot and Model Performance assessment:**

```
”’ Residual Plot ”’
residuals = y_test - y_test_pred
plt.figure(figsize=(6,4))
sns.histplot(residuals, kde=True)
plt.title(”Residual Distribution”)
plt.xlabel(”Residuals”)
plt.show()

”’ Predicted vs Actual ”’
plt.figure(figsize=(6,6))
sns.scatterplot(x=y_test, y=y_test_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], ’r–’)
plt.title(”Predicted vs Actual”)
plt.xlabel(”Actual”)
plt.ylabel(”Predicted”)
plt.grid()
plt.show()
```



(a) Residual Distribution Graph



(b) Best Fit Line

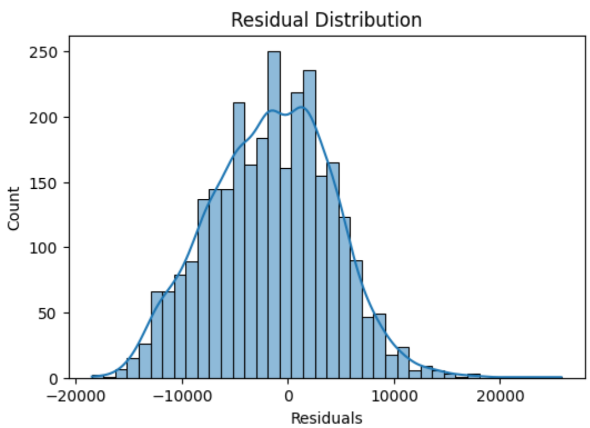
Figure 7: Model Performance Assessment

The validation/test metrics and the residuals of same loan amount prediction dataset by other regressors (AdaBoost Regressor, Gradient Boost Regressor, XGBoost Regressor, Support Vector Regressor are as follows:)

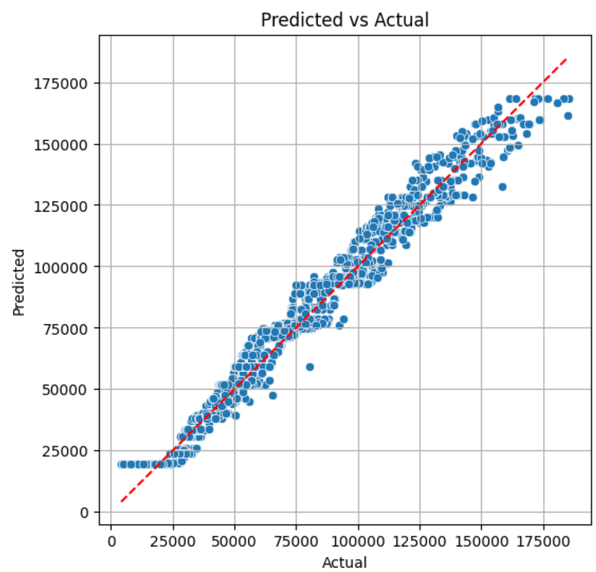
AdaBoost Regressor

Phase	MAE	MSE	RMSE	R^2 Score
Validation	4772.95	35867082.69	5988.56	0.9732
Testing	4789.11	35599892.19	5966.56	0.9723

Table 3: Validation/ Test Metrics - AdaBoost Regression



(a) Residual Distribution Graph



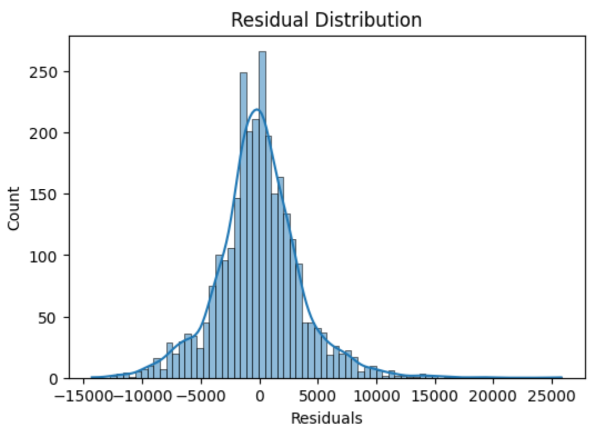
(b) Best Fit Line

Figure 8: Model Performance Assessment

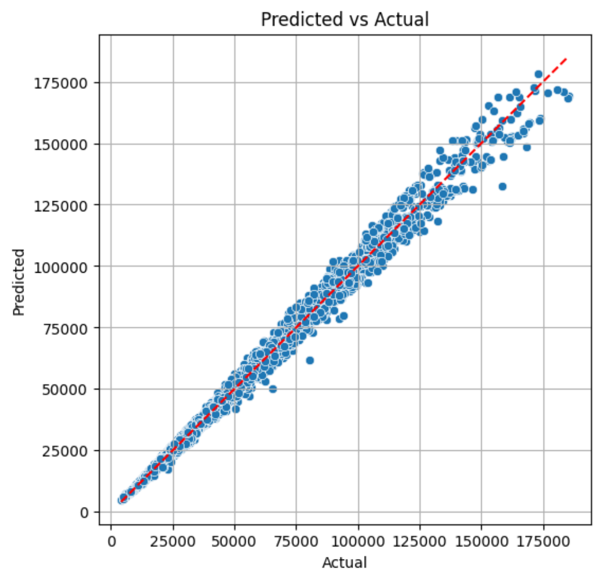
Gradient Boosting Regressor

Phase	MAE	MSE	RMSE	R^2 Score
Validation	2657.37	13552250.28	3681.34	0.9899
Testing	2639.97	13381796.98	3658.11	0.9896

Table 4: Validation/ Test Metrics - Gradient Boosting Regression



(a) Residual Distribution Graph



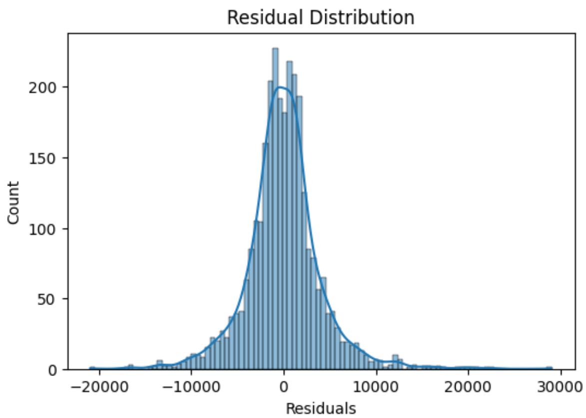
(b) Best Fit Line

Figure 9: Model Performance Assessment

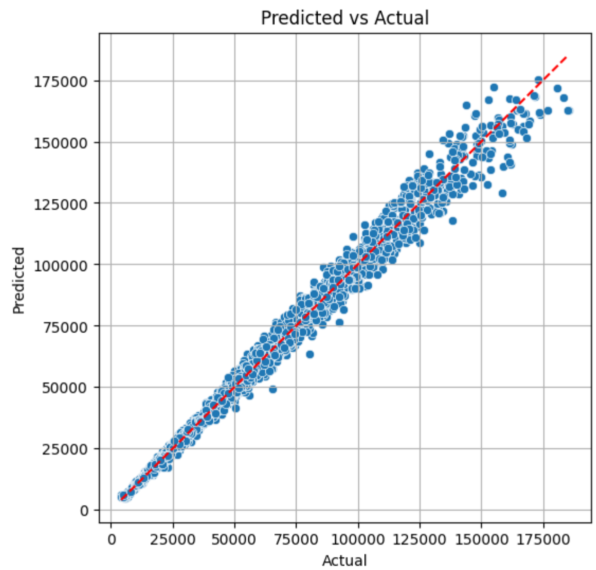
XGBoost Regressor

Phase	MAE	MSE	RMSE	R^2 Score
Validation	2801.77	16043427.58	4005.42	0.9880
Testing	2807.66	16114177.94	4014.25	0.9874

Table 5: Validation/ Test Metrics - XGBoost Regression



(a) Residual Distribution Graph



(b) Best Fit Line

Figure 10: Model Performance Assessment

INTERPRETATION:

- **Residual Distribution:** Looks reasonably tight and Gaussian-ish(though slightly skewed) = **Errors are small and almost normally distributed**
- **Predicted vs Actual:** Points are closely scattered around the best fit line. Visible spread at high values but general prediction trend is tight = **Good generalization with minor underfitting** at high targets

Table 6: Summary of Results for Loan Amount Prediction

Description	**Result**
Dataset Size (after preprocessing)	14,516
Train/Validation/Test Split Ratio	60/20/20
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, Loan Sanction Amount (USD)`
Model Used	Linear Regression
Cross-Validation Used?	Yes
If Yes, Number of Folds (K)	5
Reference to CV Results Table	Table 1
Mean Absolute Error (MAE) on Test Set	3126.74
Mean Squared Error (MSE) on Test Set	18565888.59
Root Mean Squared Error (RMSE) on Test Set	4308.82
R^2 Score on Test Set	0.9855
Most Influential Feature(s)	`Income (USD), Loan Amount Request (USD), Property price, Loan Expenses, Credit Score`
Observations from Residual Plot	Residuals appear tight and Gaussian-like (centered at 0, slight right tail), suggesting errors are small and normally distributed.
Interpretation of Predicted vs Actual Plot	Data points are tightly clustered around the best-fit line, indicating good generalization with slight underfitting.
Any Overfitting or Underfitting Observed?	Minor underfitting at higher loan amounts.
Justification	Spread in residuals increases at higher target values.

Best Practices

- **Data Preprocessing:** Handle missing values carefully (drop or impute), and remove irrelevant columns such as IDs and names that do not contribute to prediction and removing the outliers.
- **Feature Engineering:** Create new meaningful features (e.g., total income) and apply transformations (e.g., log transformation for skewed data) to improve model performance.
- **Scaling and Encoding:** Use scaling (e.g., `StandardScaler`) for numerical features and one-hot encoding for categorical features to prepare data for linear regression.
- **Train-Validate-Test Split:** Use proper splits (e.g., 60/20/20) and consider cross-validation to ensure the model generalizes well and to prevent overfitting.
- **Model Evaluation:** Use multiple metrics (MAE, MSE, RMSE, R^2) to assess different aspects of model performance.
- **Residual Analysis:** Analyze residual plots to detect model bias or heteroscedasticity and decide if further feature engineering or alternative models are needed.

Learning Outcomes

- Able to identify and handling missing values, outliers and further scaling operations
- Able to interpret the variable distributions and relationships between the target and remaining features
- Able to train and evaluate linear regression model using CV and multiple performance metrics
- Able to assess model behaviour using residual and predicted vs actual plots to detect overfitting/underfitting patterns