

**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	<b>Due date:</b>

## Experiment 2: Loan Amount Prediction using Linear Regression

**Aim:** To predict the loan amount sanctioned to users using Linear Regression on historical data, and analyze model performance using visual and statistical metrics.

**Libraries used:**

- Pandas - for data handling
- numpy - for numerical operations
- matplotlib.pyplot and seaborn - for visualization
- sklearn - for model building and evaluation

**Objective:** To build a linear regression model using Scikit-learn to predict the loan amount, perform exploratory data analysis, visualize model performance, and interpret results.

**Mathematical/theoretical description:** The linear regression model expresses the relationship between the input features and the predicted output as:

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

Where:

- $y$  is the predicted loan amount,
- $x_i$  are the input features (e.g., income, credit score, etc.),
- $\beta_i$  are the coefficients (weights) learned by the model,
- $\epsilon$  is the error term (residual).

**CODE:**

```
!pip install xgboost
```

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

```
import seaborn as sns

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler , LabelEncoder
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.svm import SVR
from sklearn.metrics import (
    mean_absolute_error, mean_squared_error, r2_score,
    accuracy_score, precision_score, recall_score, f1_score
)

import time

# 1. Load Dataset
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/ML LAB SEM 5/train.csv')
print(df.head())

## Drop non-informative identifiers
df.drop(columns=["Customer ID", "Name", "Property ID"], inplace=True)

# Handle missing values (optional: use better imputation)
df.dropna(inplace=True)

# Define target variable
target = "Loan Sanction Amount (USD)"
X = df.drop(columns=[target])
y = df[target]

# Encode categorical variables
categorical_cols = X.select_dtypes(include=["object"]).columns
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

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

# 3. EDA
# a. Handling outliers
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()

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")

# b. Loan Amount Distribution Plot
sns.histplot(df["Loan Sanction Amount (USD)"], kde=True, color="skyblue")
plt.title("Loan Sanction Amount Distribution")
plt.xlabel("Loan Sanction Amount (USD)")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()

# c. Scatter plots
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()
```

```
# d. Correlation Heatmap (only for numeric columns)
numeric_df = df.select_dtypes(include=["number"]) # selects only numeric columns
plt.figure(figsize=(12, 8))
sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```

## OUTPUT

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Customer ID          Name Gender  Age  Income (USD) Income Stability \
0      C-36995  Frederica Shealy    F   56     1933.05      Low
1      C-33999  America Calderone    M   32     4952.91      Low
2      C-3770   Rosetta Verne    F   65     988.19      High
3      C-26480    Zoe Chitty    F   65        NaN      High
4      C-23459   Afton Venema    F   31     2614.77      Low

Profession Type of Employment Location  Loan Amount Request (USD) \
0  Working       Sales staff  Semi-Urban           72889.58
1  Working            NaN      Semi-Urban           46837.47
2  Pensioner         NaN      Semi-Urban           45593.04
3  Pensioner         NaN        Rural           80857.92
4  Working  High skill tech staff  Semi-Urban           113858.89

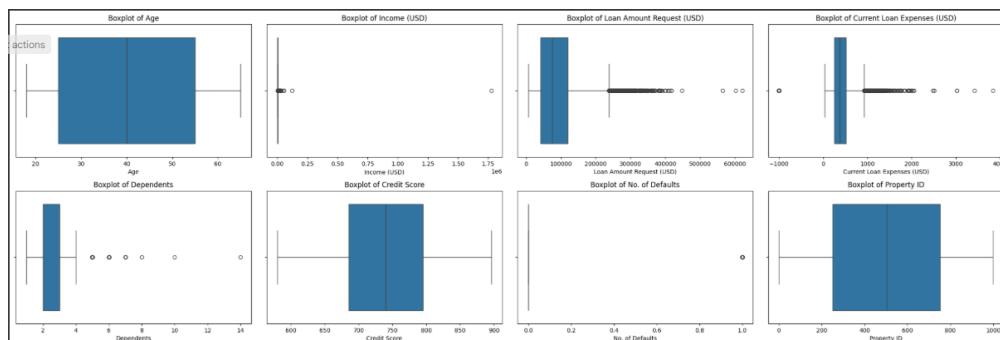
...  Credit Score No. of Defaults Has Active Credit Card  Property ID \
0 ...        809.44          0             NaN            746
1 ...        780.40          0        Unpossessed            608
2 ...        833.15          0        Unpossessed            546
3 ...        832.70          1        Unpossessed            890
4 ...        745.55          1            Active            715

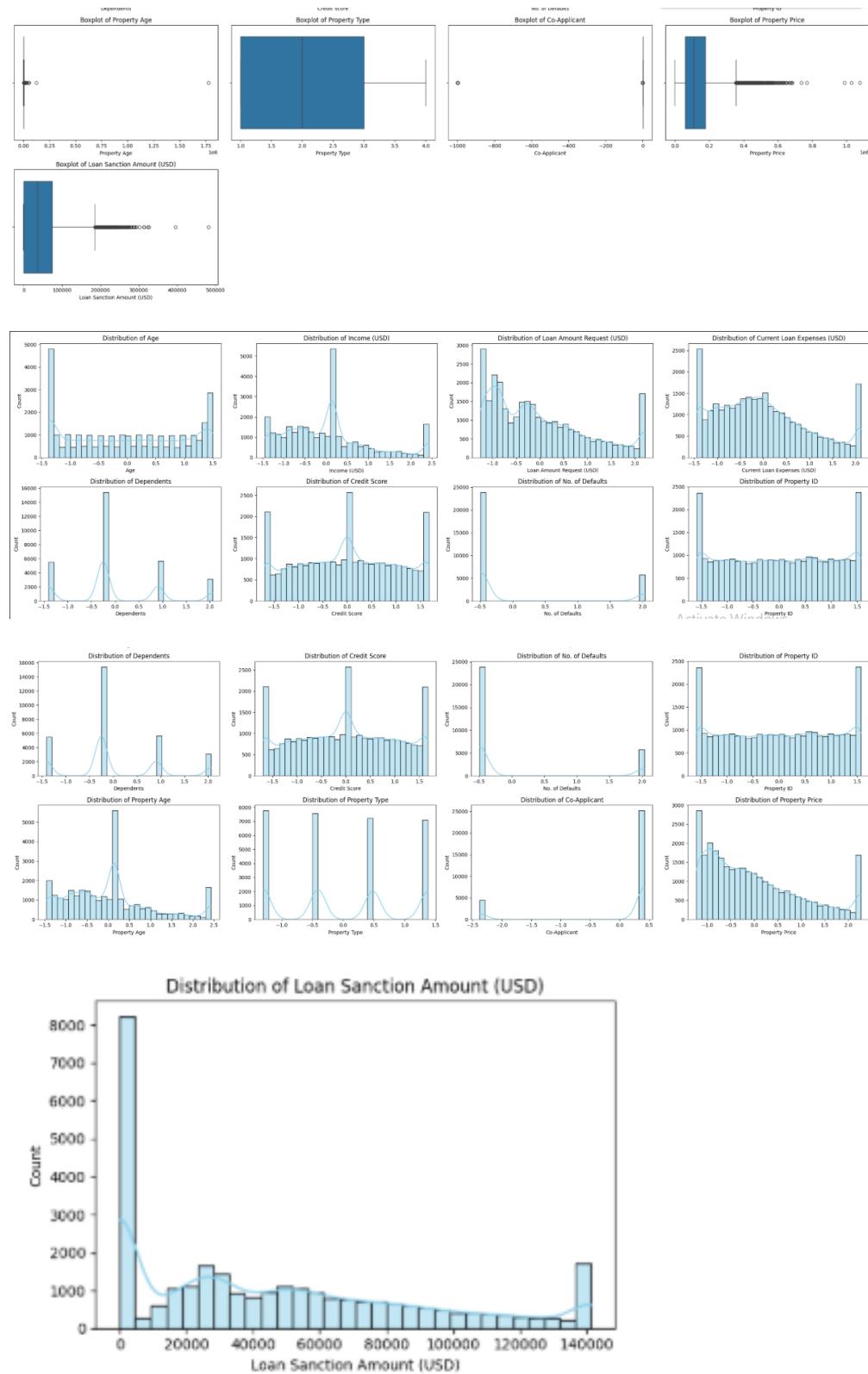
...  Credit Score No. of Defaults Has Active Credit Card  Property ID \
0 ...        809.44          0             NaN            746
1 ...        780.40          0        Unpossessed            608
2 ...        833.15          0        Unpossessed            546
3 ...        832.70          1        Unpossessed            890
4 ...        745.55          1            Active            715

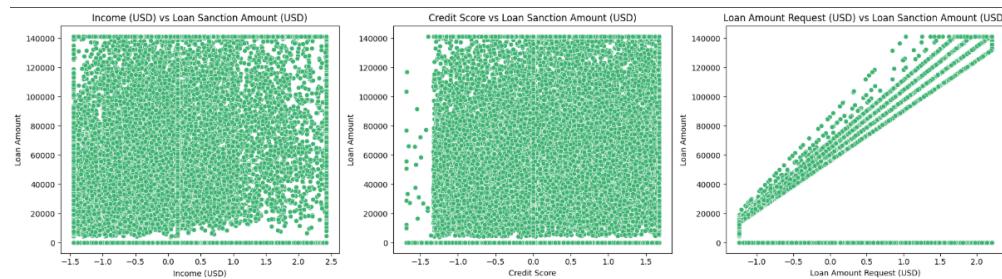
Property Age  Property Type Property Location  Co-Applicant \
0      1933.05          4            Rural            1
1      4952.91          2            Rural            1
2      988.19          2            Urban            0
3        NaN            2  Semi-Urban            1
4     2614.77          4  Semi-Urban            1

Property Price  Loan Sanction Amount (USD)
0      119933.46        54607.18
1      54791.00        37469.98
2      72440.58        36474.43
3     121441.51        56040.54
4     208567.91        74008.28

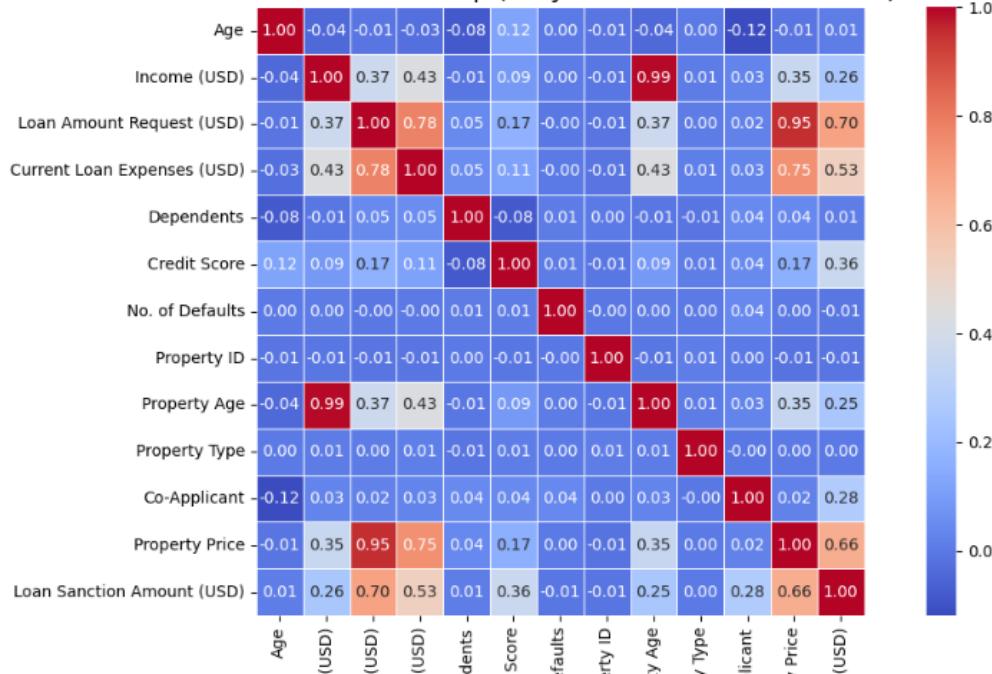
[5 rows x 24 columns]
```







Correlation Heatmap (Only Actual Numeric Features)



Capping of outliers done

## # 4. Train-test Split

```
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)

# Ensure y_train and y_test are 1-dimensional
y_train = y_train.to_numpy().ravel() if isinstance(y_train, pd.DataFrame) else y_train.to_numpy()
y_test = y_test.to_numpy().ravel() if isinstance(y_test, pd.DataFrame) else y_test.to_numpy()

y_train = y_train.squeeze()
y_test = y_test.squeeze()

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

# Ensure y_train and y_test are 1-dimensional
y_train = y_train.values.ravel() if isinstance(y_train, pd.DataFrame) else y_train.ravel()
```

```

y_test = y_test.values.ravel() if isinstance(y_test, pd.DataFrame) else y_test.ravel()

def encode_features(X_train, X_test):
    for col in X_train.select_dtypes(include=['object']).columns:
        le = LabelEncoder()
        X_train[col] = le.fit_transform(X_train[col].astype(str))
        X_test[col] = le.transform(X_test[col].astype(str))
    return X_train, X_test

X_train, X_test = encode_features(X_train, X_test)

# 5. Train Model
MODELS TO BE TRAINED:
    Linear Regression
    Ridge Regression
    Lasso Regression
    ElasticNet Regression
    Polynomial Regression
    Decision Tree Regression
    Random Forest Regressor
    Adaboost Regressor
    Gradient Boost Regressor
    Xg Boost Regressor
    Support Vector Machine Regressor(Linear , Polynomial , rbf , Sigmoid )

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=(6, 5))
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)
plt.show()

# RESIDUAL PLOT
residuals = y_test - y_pred
plt.figure(figsize=(6, 5))
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)
plt.show()

# 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.figure(figsize=(6, 5))
    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()

-----
# 6. Visualizations
LINEAR REGRESSION
lr_model = LinearRegression()
evaluate_model("Linear Regression", lr_model, X_train, X_test)
```

## OUTPUT

Linear Regression - Hyperparameter Tuning Started

Date: 29-07-2025

Experiment: 2

Name: Harini LV

Roll No: 3122237001016

Linear Regression Performance:

MAE: 19020.2471

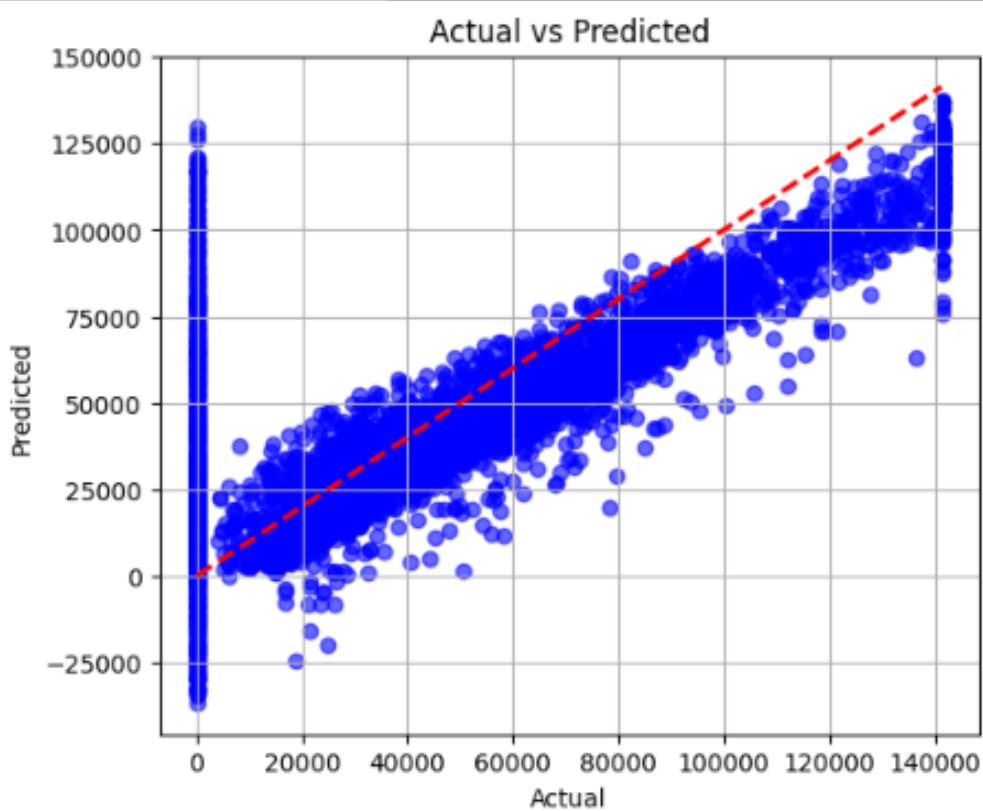
MSE: 742669038.5209

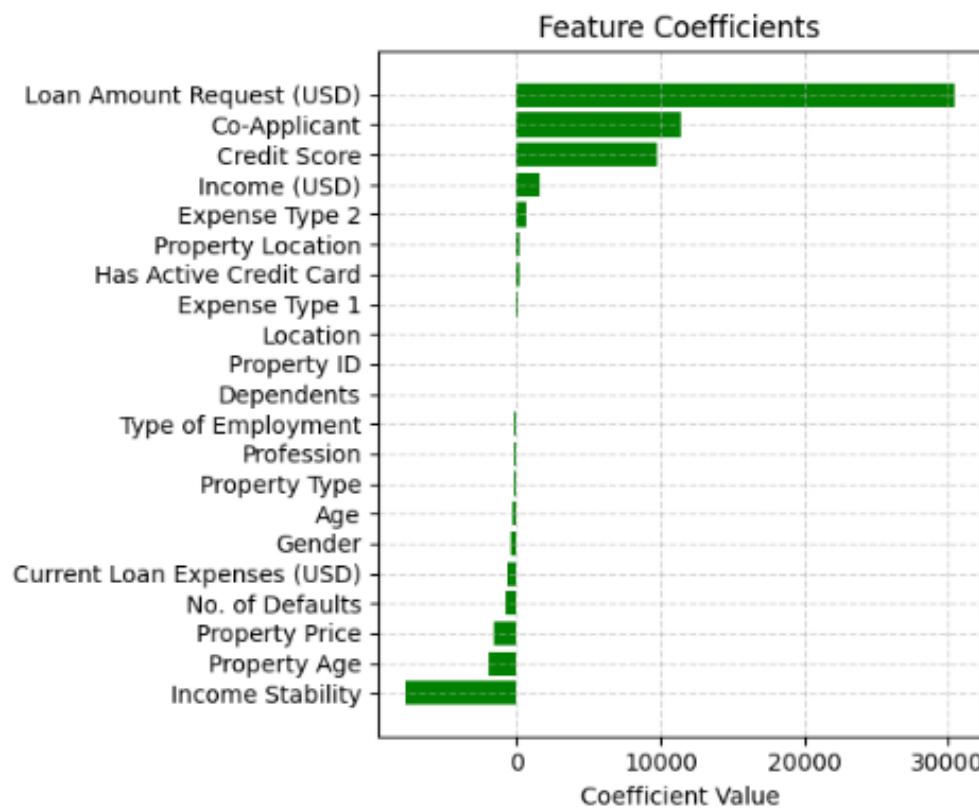
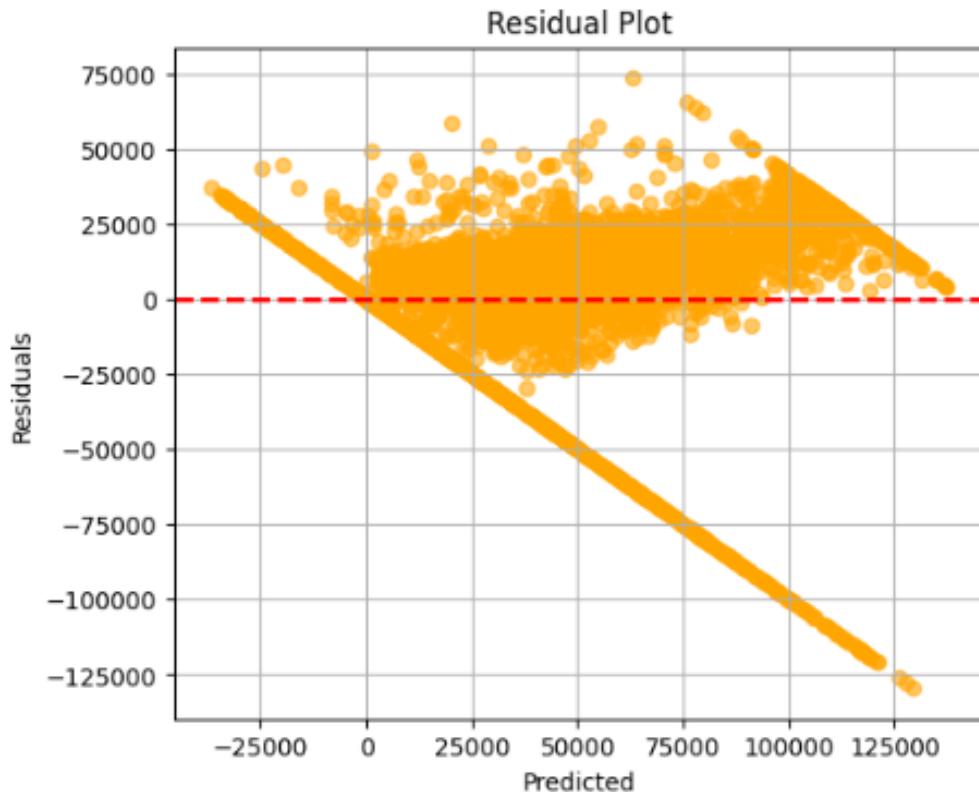
RMSE: 27251.9548

R<sup>2</sup>: 0.5998

Adjusted R<sup>2</sup>: 0.5979

Training Time: 0.0456 seconds





```
lasso_model = Lasso()
param_grid_lasso = {'alpha': [0.001, 0.01, 0.1, 1.0, 10.0]}
evaluate_model("Lasso Regression", lasso_model, X_train, X_test, param_grid=param_grid_lasso)
```

## OUTPUT

Lasso Regression - Hyperparameter Tuning Started

Best Params (GridSearchCV): {'alpha': 10.0}

Best CV Score (GridSearchCV): 0.6167

Lasso Regression Performance:

MAE: 19019.3820

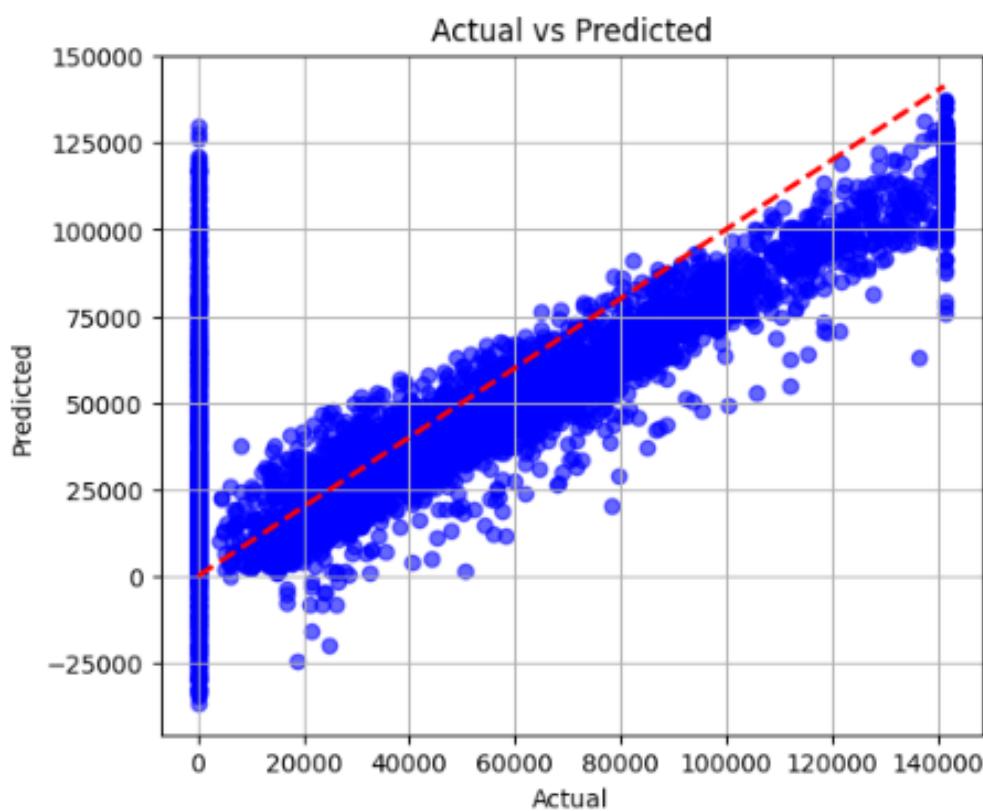
MSE: 742599116.4175

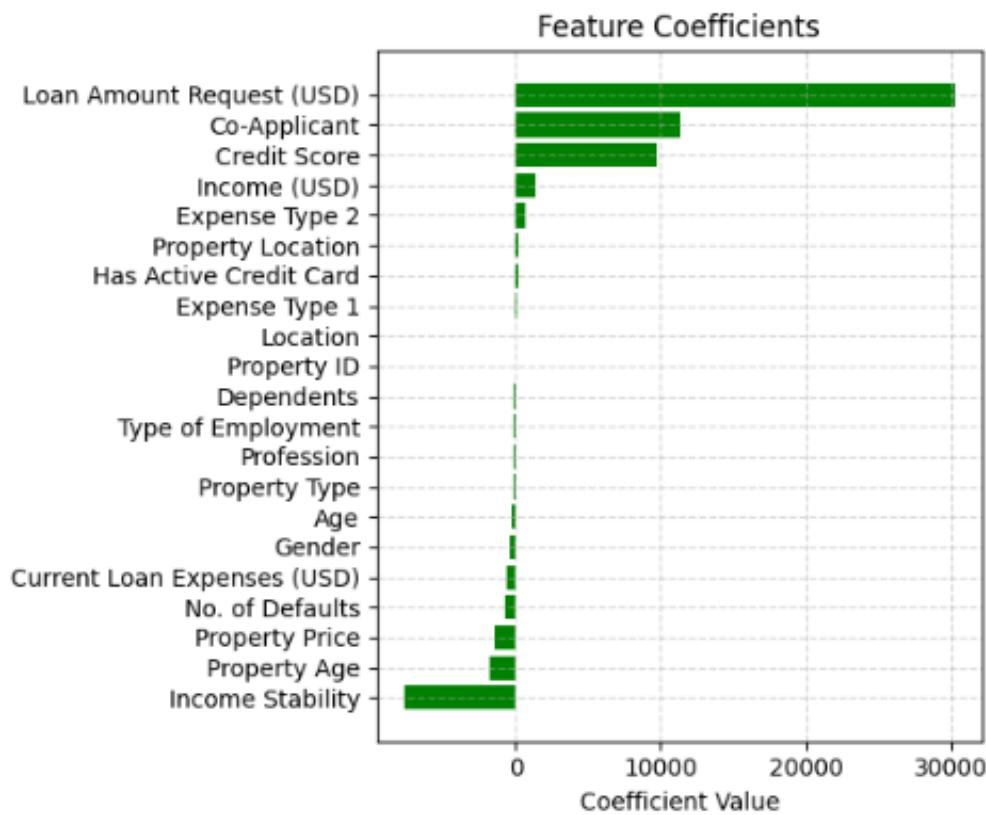
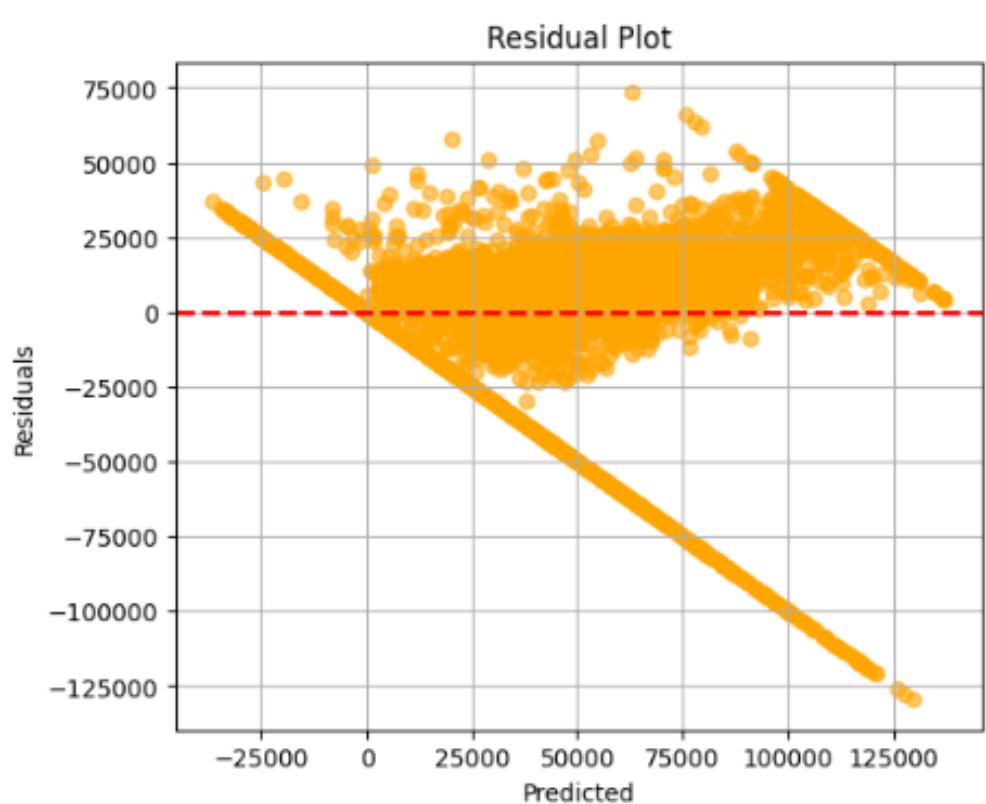
RMSE: 27250.6719

R<sup>2</sup>: 0.5998

Adjusted R<sup>2</sup>: 0.5979

Training Time: 0.1644 seconds



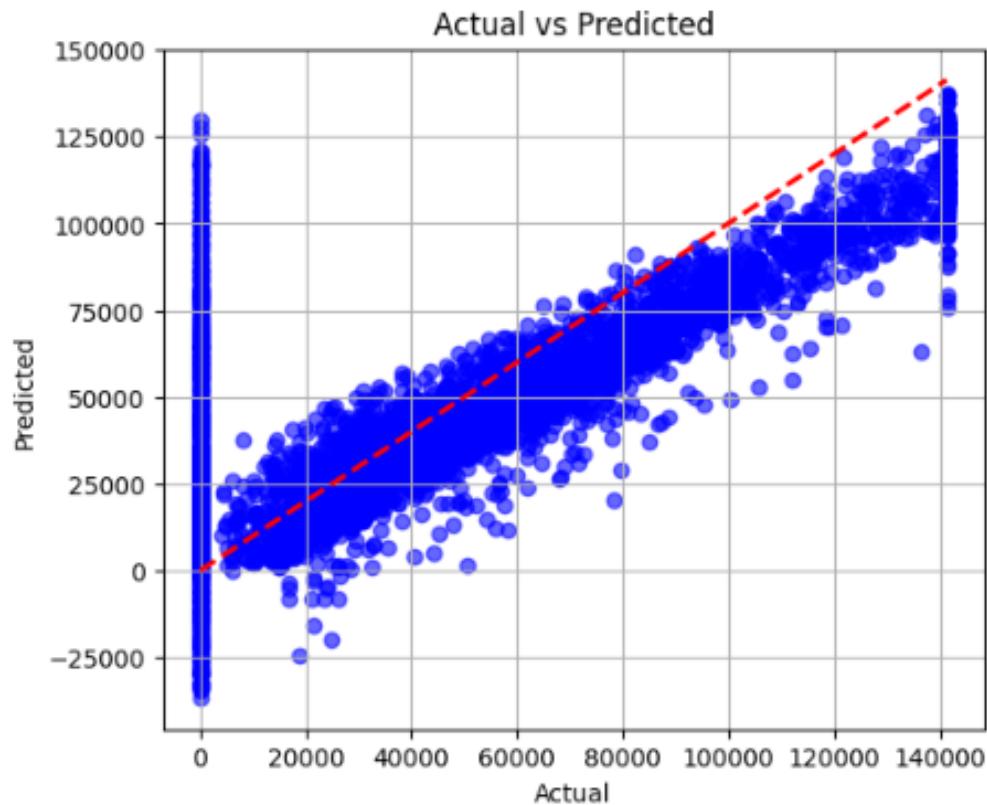


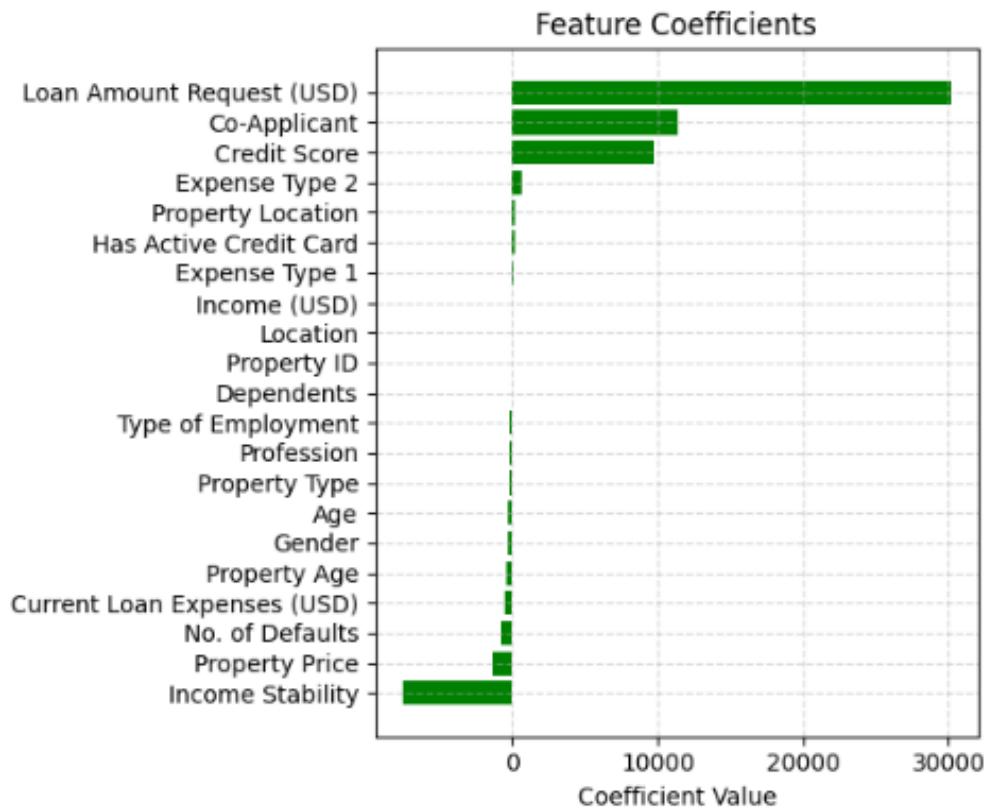
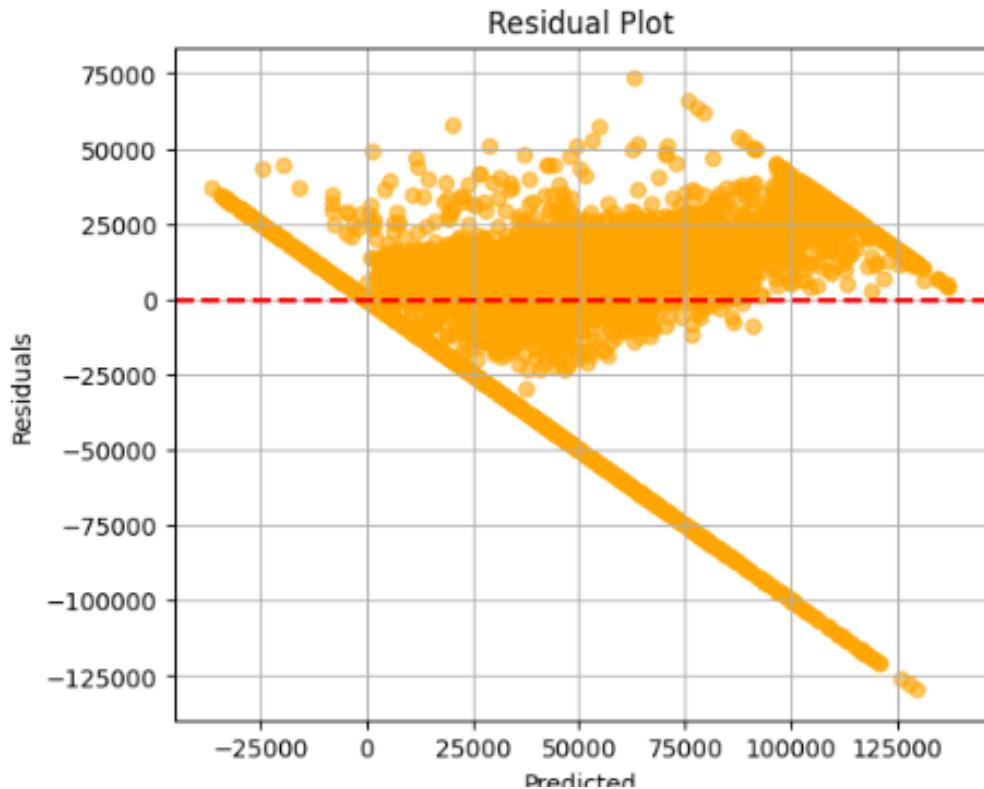
```
ridge_model = Ridge()  
param_grid_ridge = {'alpha': [0.01, 0.1, 1.0, 10.0, 100.0]}  
evaluate_model("Ridge Regression", ridge_model, X_train, X_test, param_grid=param_grid_ridge)
```

## OUTPUT

```
Ridge Regression - Hyperparameter Tuning Started  
Best Params (GridSearchCV): {'alpha': 10.0}  
Best CV Score (GridSearchCV): 0.6167
```

```
Ridge Regression Performance:  
MAE: 19021.3094  
MSE: 742586318.0823  
RMSE: 27250.4370  
R2: 0.5998  
Adjusted R2: 0.5979  
Training Time: 0.0128 seconds
```





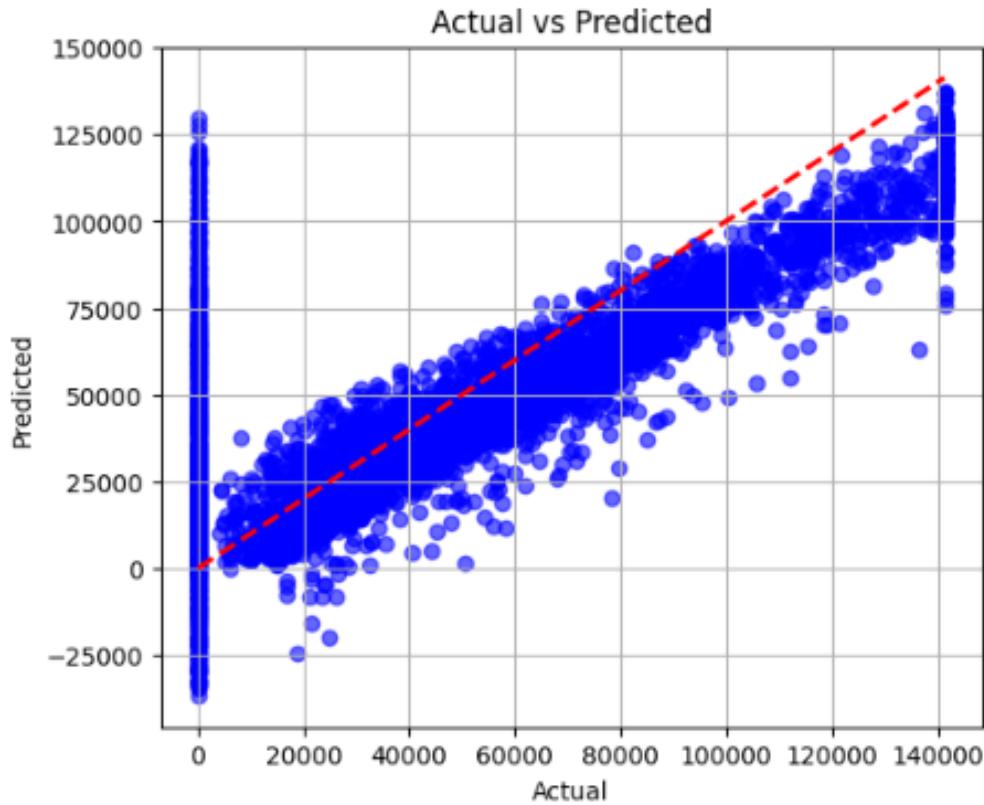
```
elastic_net_model = ElasticNet()
param_grid_elastic = {'alpha': [0.01, 0.1, 1.0], 'l1_ratio': [0.1, 0.5, 0.9]}
evaluate_model("ElasticNet Regression", elastic_net_model, X_train, X_test, param_grid=param_g
```

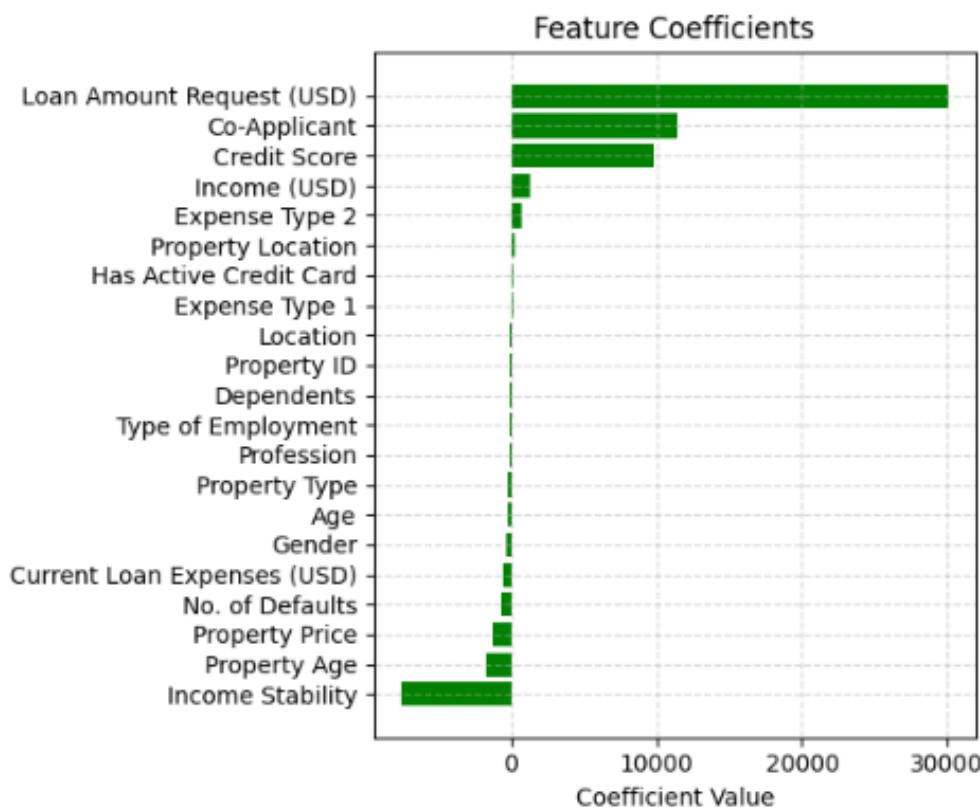
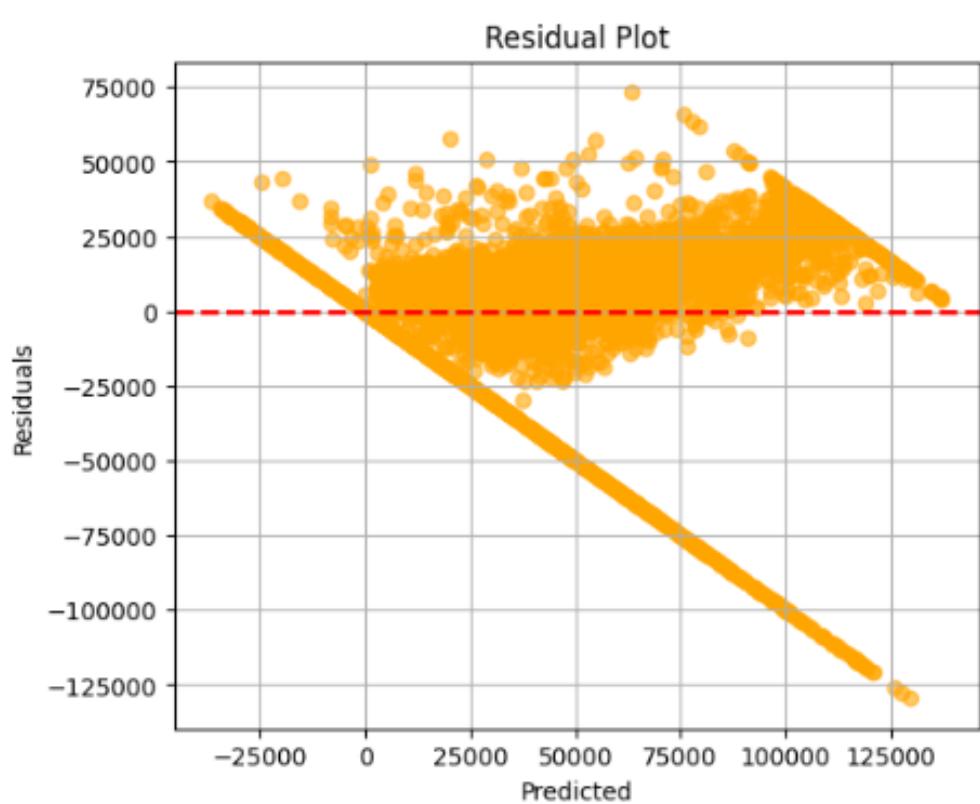
## OUTPUT

```
ElasticNet Regression - Hyperparameter Tuning Started
Best Params (GridSearchCV): {'alpha': 0.01, 'l1_ratio': 0.9}
Best CV Score (GridSearchCV): 0.6167
```

ElasticNet Regression Performance:

MAE: 19022.4970  
MSE: 742506150.4232  
RMSE: 27248.9660  
 $R^2$ : 0.5998  
Adjusted  $R^2$ : 0.5979  
Training Time: 0.5872 seconds





```
poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
poly_model = LinearRegression()
print("\nPolynomial Regression (Degree 2) - Evaluation")
poly_model.fit(X_train_poly, y_train)
y_pred_poly = poly_model.predict(X_test_poly)

mae_poly = mean_absolute_error(y_test, y_pred_poly)
mse_poly = mean_squared_error(y_test, y_pred_poly)
rmse_poly = np.sqrt(mse_poly)
r2_poly = r2_score(y_test, y_pred_poly)
adj_r2_poly = 1 - (1 - r2_poly) * (len(y_test) - 1) / (len(y_test) - X_test_poly.shape[1] - 1)

print("Polynomial Regression Performance:")
print(f"MAE: {mae_poly:.4f}")
print(f"MSE: {mse_poly:.4f}")
print(f"RMSE: {rmse_poly:.4f}")
print(f"R2: {r2_poly:.4f}")
print(f"Adjusted R2: {adj_r2_poly:.4f}")
```

## OUTPUT

```
Polynomial Regression (Degree 2) - Evaluation
Polynomial Regression Performance:
MAE: 15338.2460
MSE: 624255927.7910
RMSE: 24985.1141
R2: 0.6636
Adjusted R2: 0.6434
```

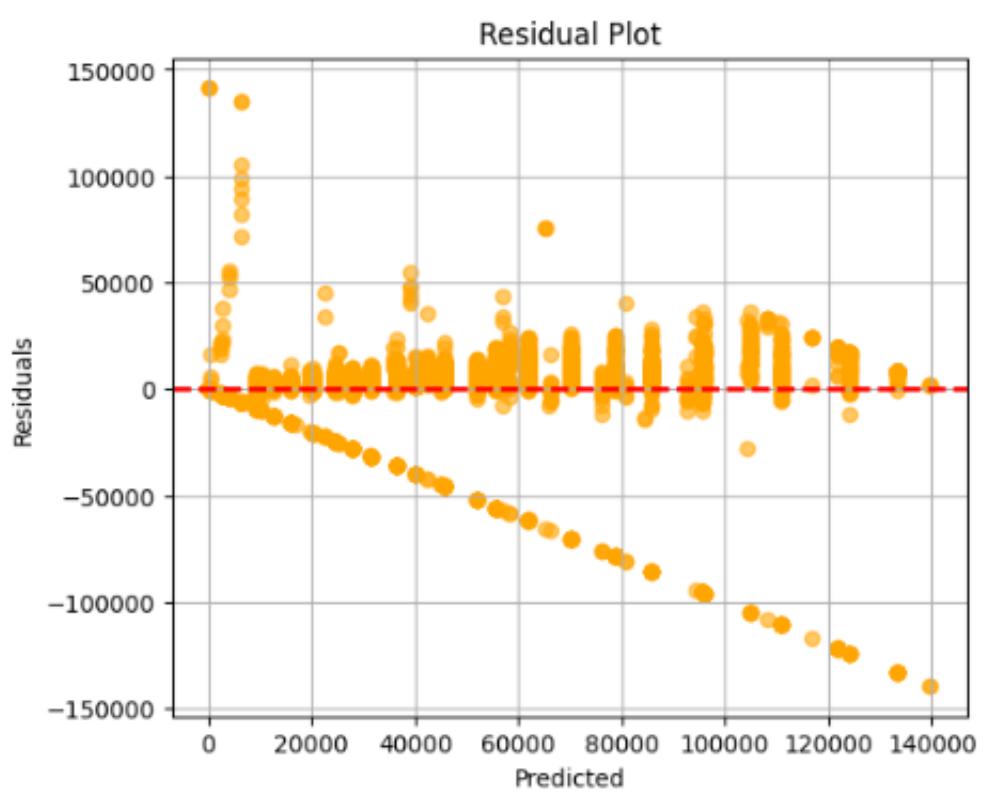
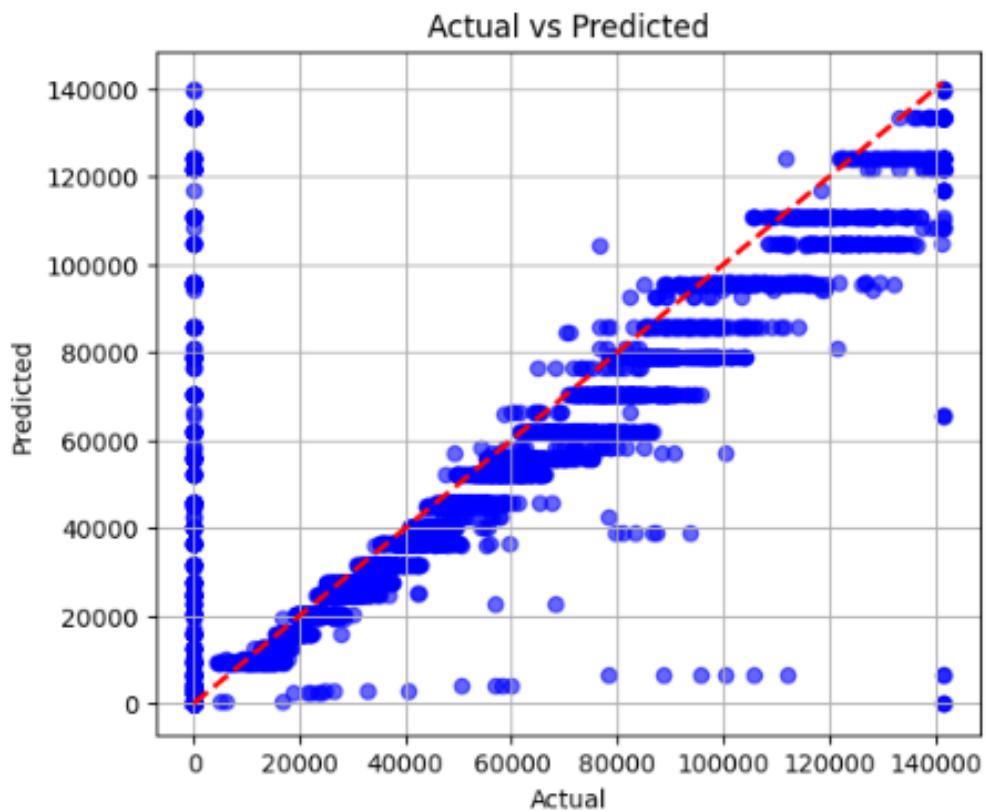
## DECISION TREE REGRESSOR

```
dt_model = DecisionTreeRegressor(random_state=42)
param_grid_dt = {'max_depth': [3, 5, 7, 10], 'min_samples_split': [2, 5, 10]}
evaluate_model("Decision Tree Regressor", dt_model, X_train, X_test, param_grid=param_grid_dt)
```

## OUTPUT

```
Decision Tree Regressor - Hyperparameter Tuning Started
Best Params (GridSearchCV): {'max_depth': 7, 'min_samples_split': 5}
Best CV Score (GridSearchCV): 0.7471
```

```
Decision Tree Regressor Performance:
MAE: 11275.1997
MSE: 524797166.6419
RMSE: 22908.4519
R2: 0.7172
Adjusted R2: 0.7158
Training Time: 0.1745 seconds
```



```
rf_model = RandomForestRegressor(random_state=42)
param_grid_rf = {'n_estimators': [100, 200], 'max_depth': [5, 10, None]}
evaluate_model("Random Forest Regressor", rf_model, X_train, X_test, param_grid=param_grid_rf)
```

## OUTPUT

```
Random Forest Regressor - Hyperparameter Tuning Started
Best Params (GridSearchCV): {'max_depth': 10, 'n_estimators': 200}
Best CV Score (GridSearchCV): 0.7560
```

Random Forest Regressor Performance:

MAE: 10988.6491

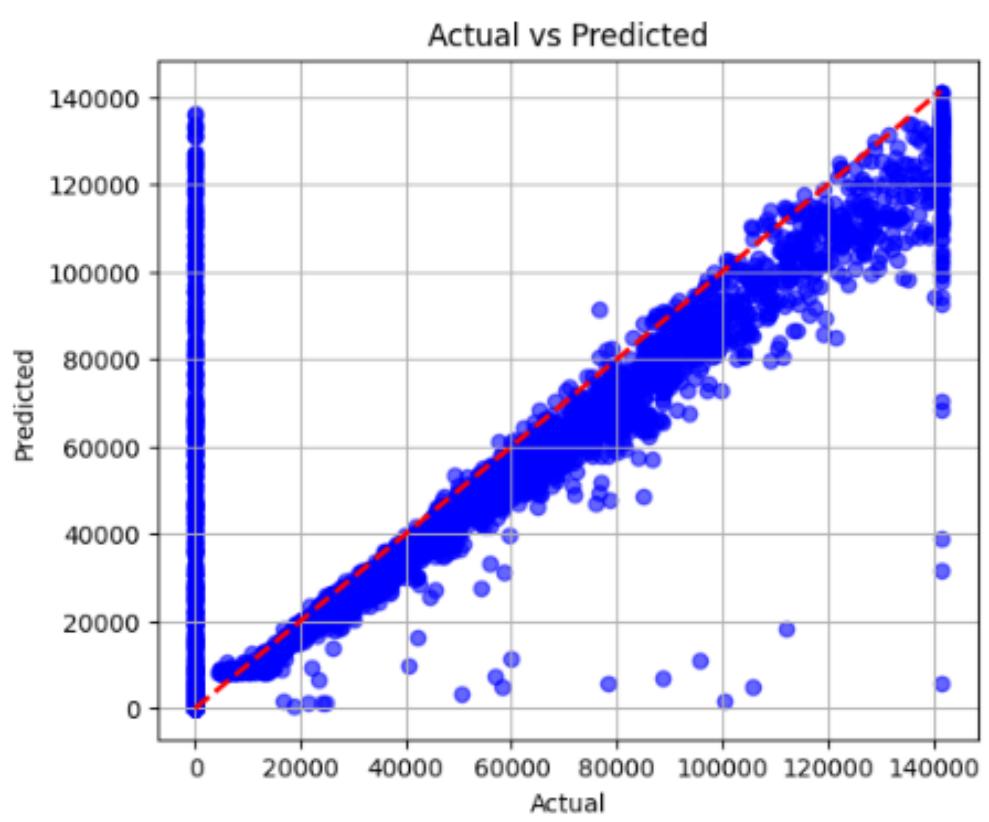
MSE: 499159271.7610

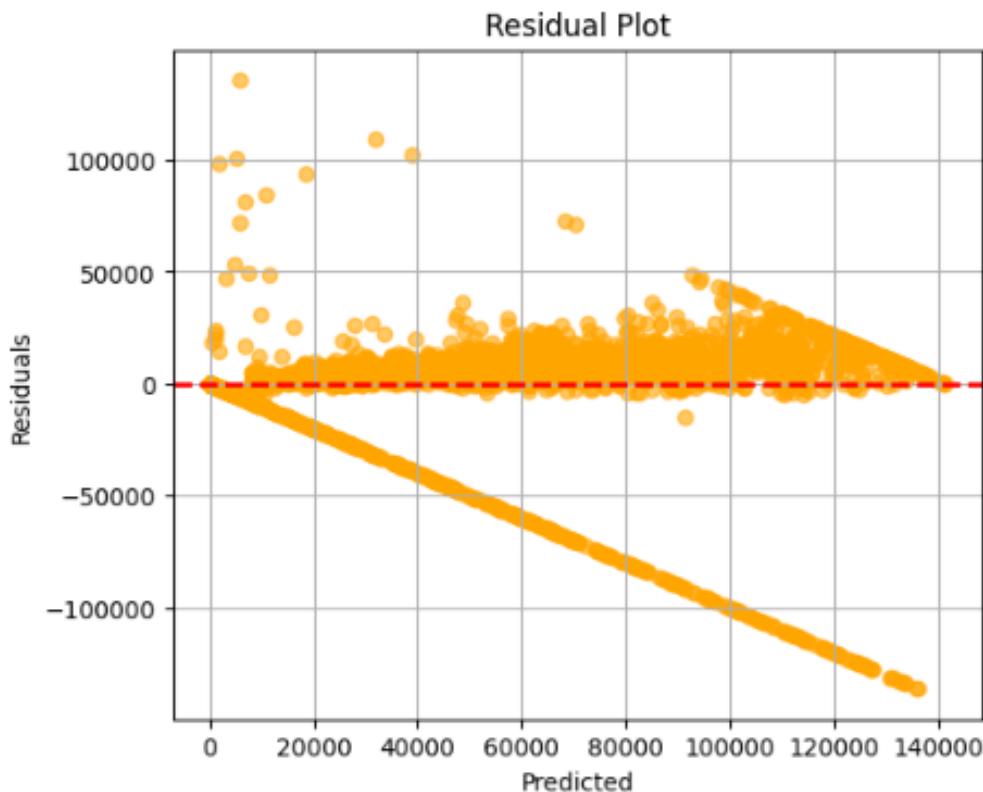
RMSE: 22341.8726

R<sup>2</sup>: 0.7310

Adjusted R<sup>2</sup>: 0.7297

Training Time: 34.9321 seconds





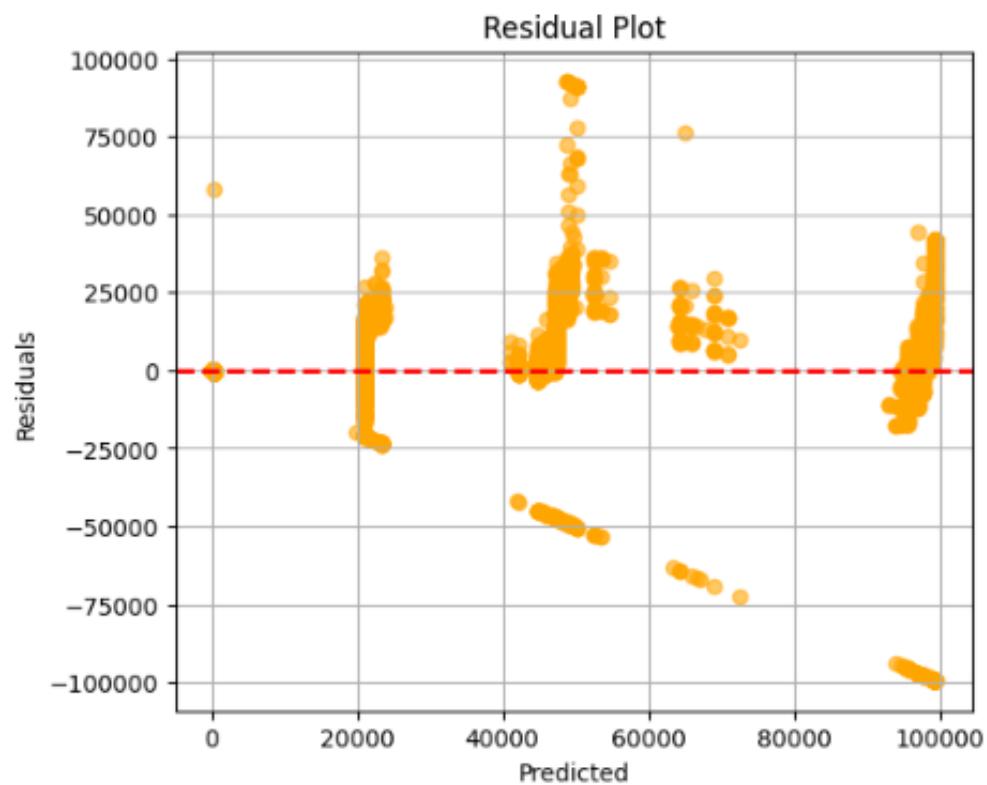
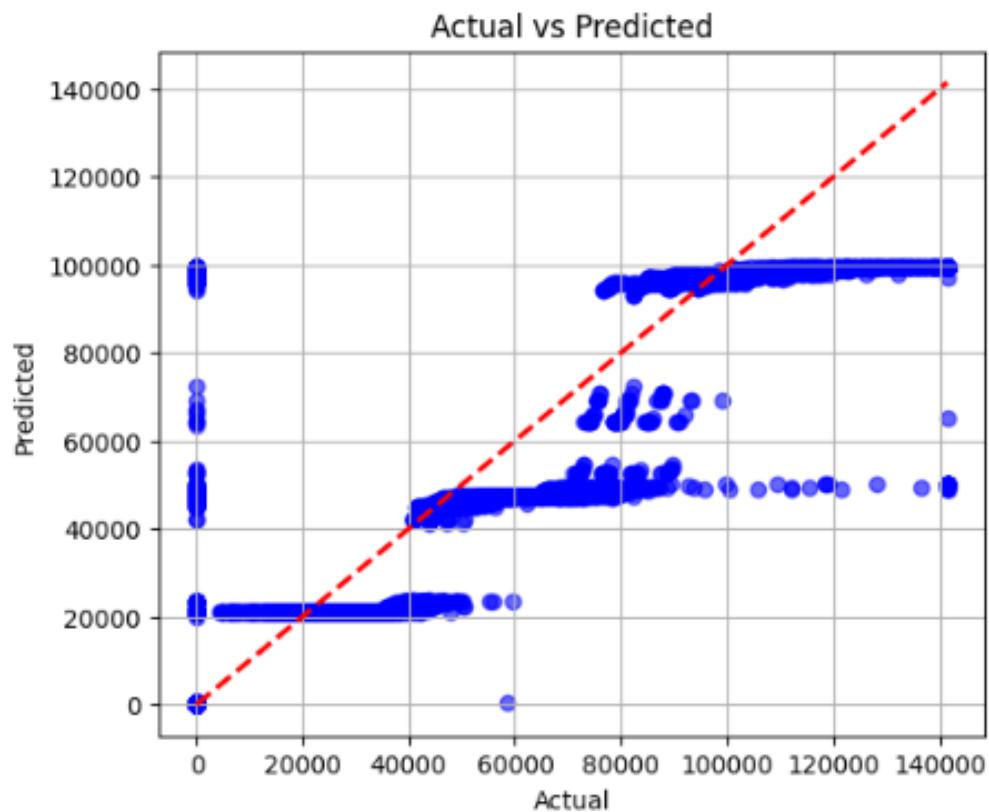
#### ADABOOST REGRESSOR

```
adaboost_model = AdaBoostRegressor(random_state=42)
param_grid_ada = {'n_estimators': [50, 100], 'learning_rate': [0.01, 0.1, 1.0]}
evaluate_model("AdaBoost Regressor", adaboost_model, X_train, X_test, param_grid=param_grid_ada)
```

#### OUTPUT

```
AdaBoost Regressor - Hyperparameter Tuning Started
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  
 $R^2$ : 0.6159  
Adjusted  $R^2$ : 0.6141  
Training Time: 9.1596 seconds



```
gbr_model = GradientBoostingRegressor(random_state=42)
param_grid_gbr = {'n_estimators': [50, 100], 'learning_rate': [0.1, 0.5], 'max_depth': [3, 5]}
evaluate_model("Gradient Boosting Regressor", gbr_model, X_train, X_test, param_grid=param_grid_gbr)
```

## OUTPUT

```
Gradient Boosting Regressor - Hyperparameter Tuning Started
Best Params (GridSearchCV): {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}
Best CV Score (GridSearchCV): 0.7583
```

Gradient Boosting Regressor Performance:

MAE: 11692.8895

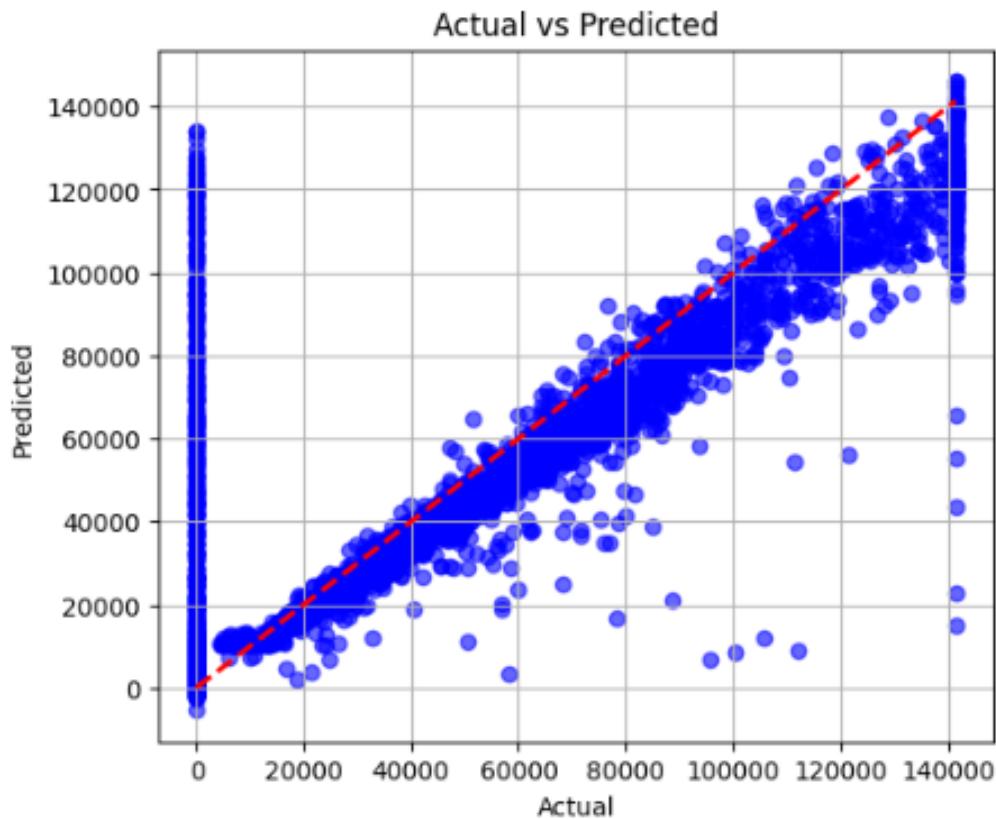
MSE: 497641149.3616

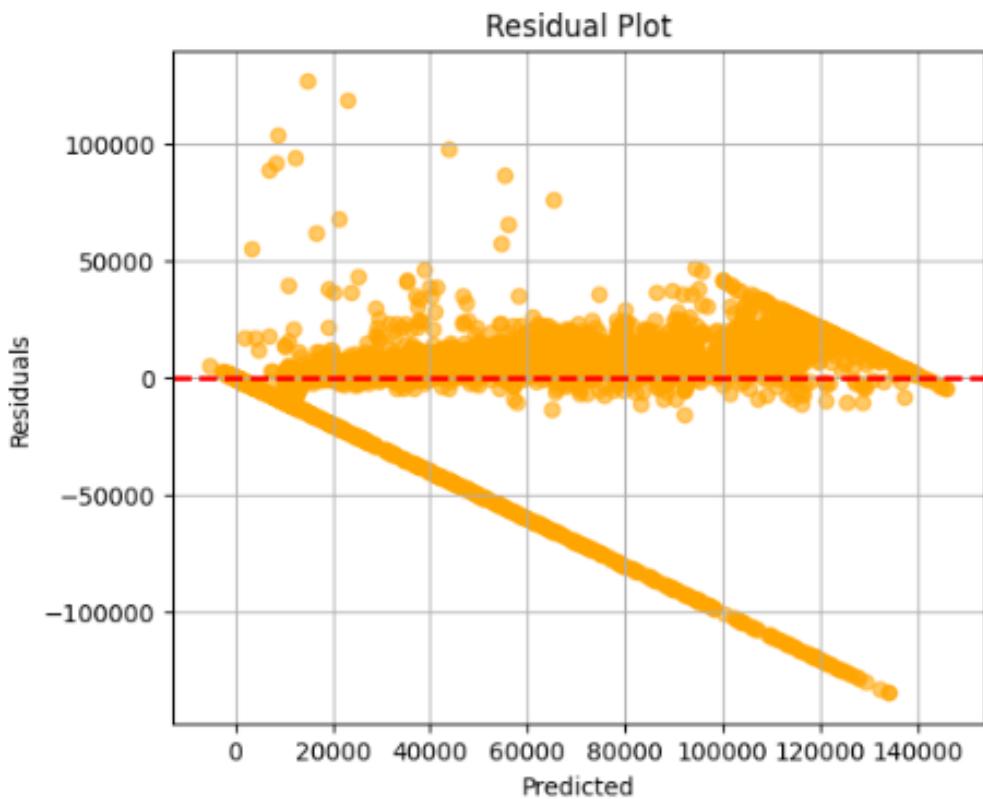
RMSE: 22307.8719

R<sup>2</sup>: 0.7318

Adjusted R<sup>2</sup>: 0.7305

Training Time: 8.0911 seconds





#### XGBOOST REGRESSOR

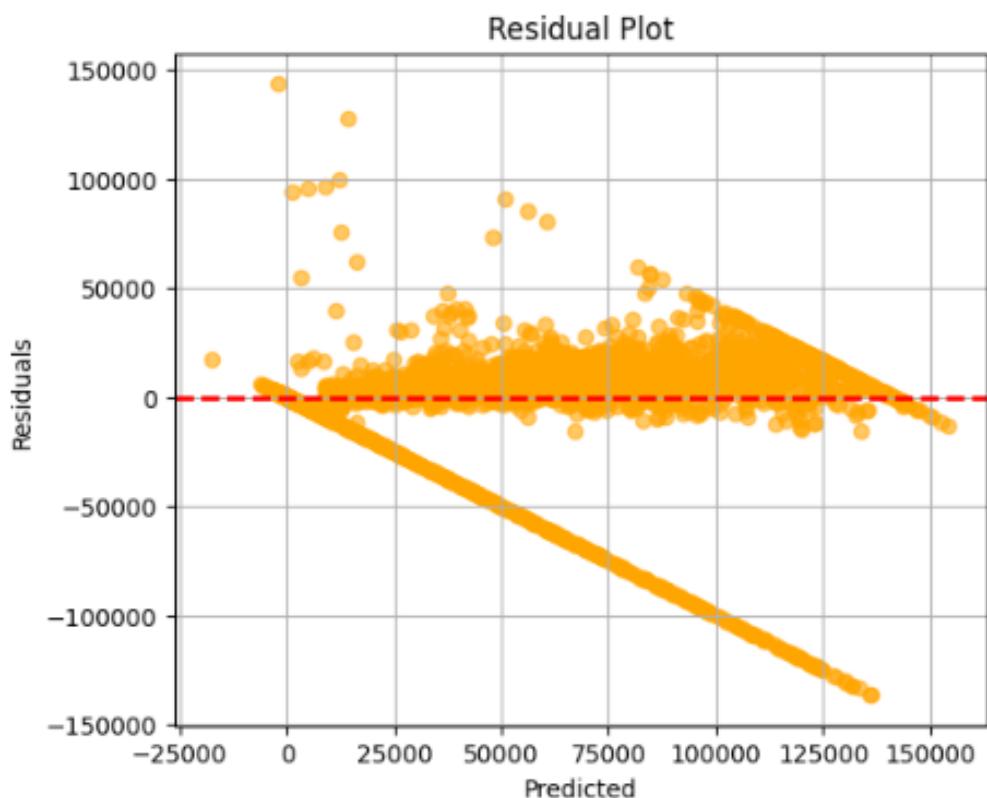
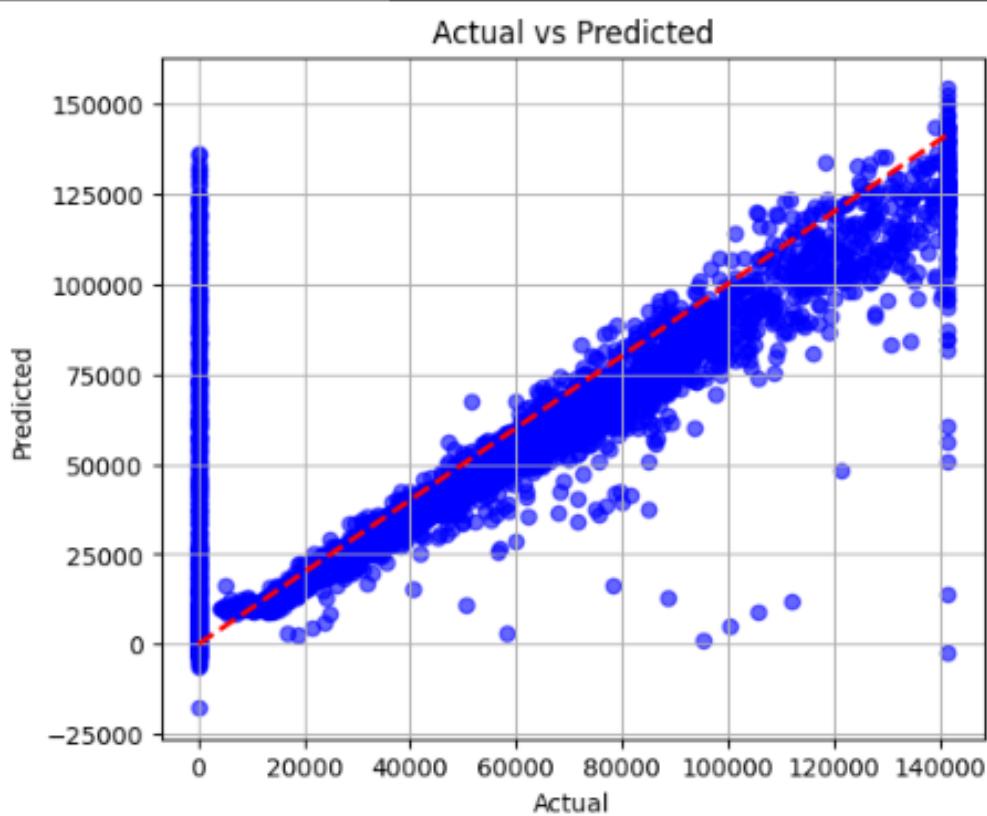
```
xgb_model = XGBRegressor(random_state=42)
param_grid_xgb = {'n_estimators': [100, 200], 'learning_rate': [0.01, 0.1], 'max_depth': [3, 5]}
evaluate_model("XGBoost Regressor", xgb_model, X_train, X_test, param_grid=param_grid_xgb)
```

#### OUTPUT

```
XGBoost Regressor - Hyperparameter Tuning Started
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
Training Time: 0.3480 seconds
```



a) SVR-LINEAR

```
svr_linear = SVR(kernel='linear', C=1.0)
evaluate_model("SVR (Linear Kernel)", svr_linear, X_train, X_test)
```

**OUTPUT**

SVR (Linear Kernel) - Hyperparameter Tuning Started

SVR (Linear Kernel) Performance:

MAE: 21649.7959

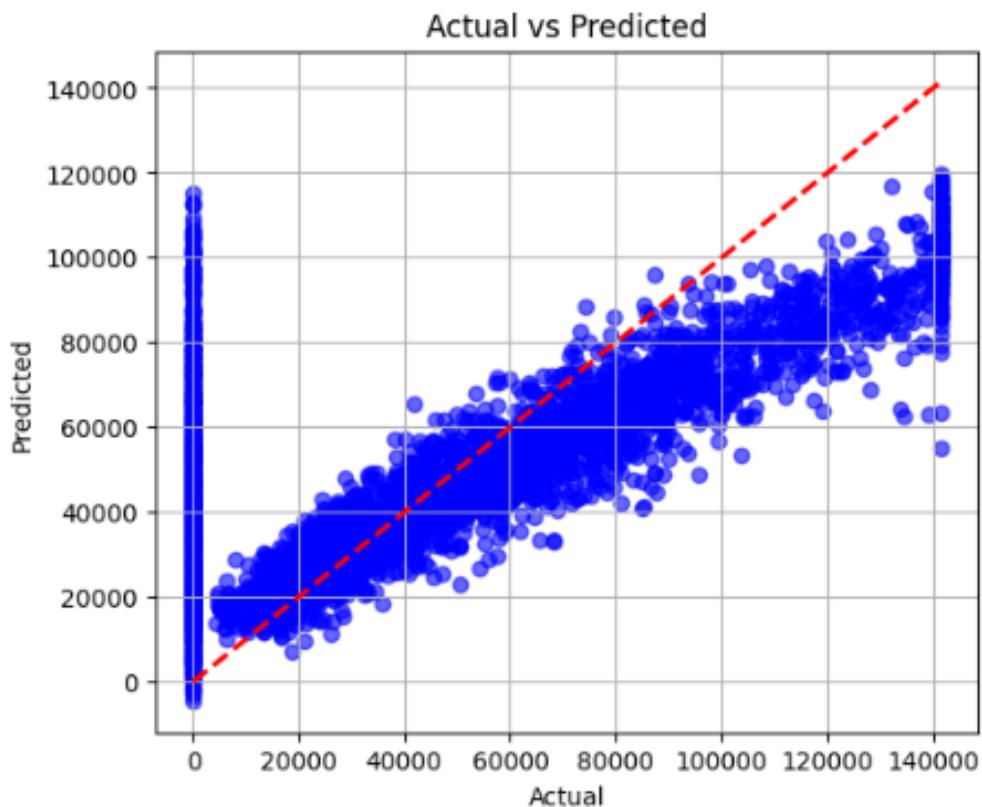
MSE: 902195888.2657

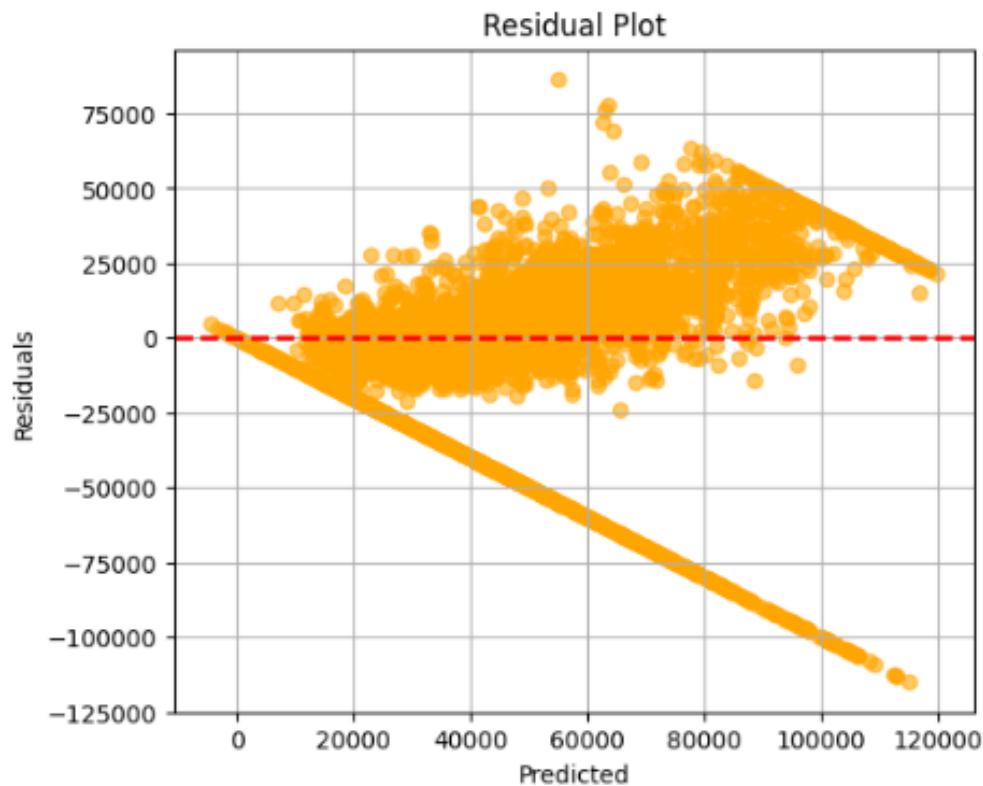
RMSE: 30036.5758

R<sup>2</sup>: 0.5138

Adjusted R<sup>2</sup>: 0.5115

Training Time: 23.1644 seconds





b) SVR-POLYNOMIAL

```
svr_poly = SVR(kernel='poly', C=1.0, degree=3, gamma='scale')
evaluate_model("SVR (Polynomial Kernel)", svr_poly, X_train, X_test)
```

## OUTPUT

SVR (Polynomial Kernel) - Hyperparameter Tuning Started

SVR (Polynomial Kernel) Performance:

MAE: 34730.5448

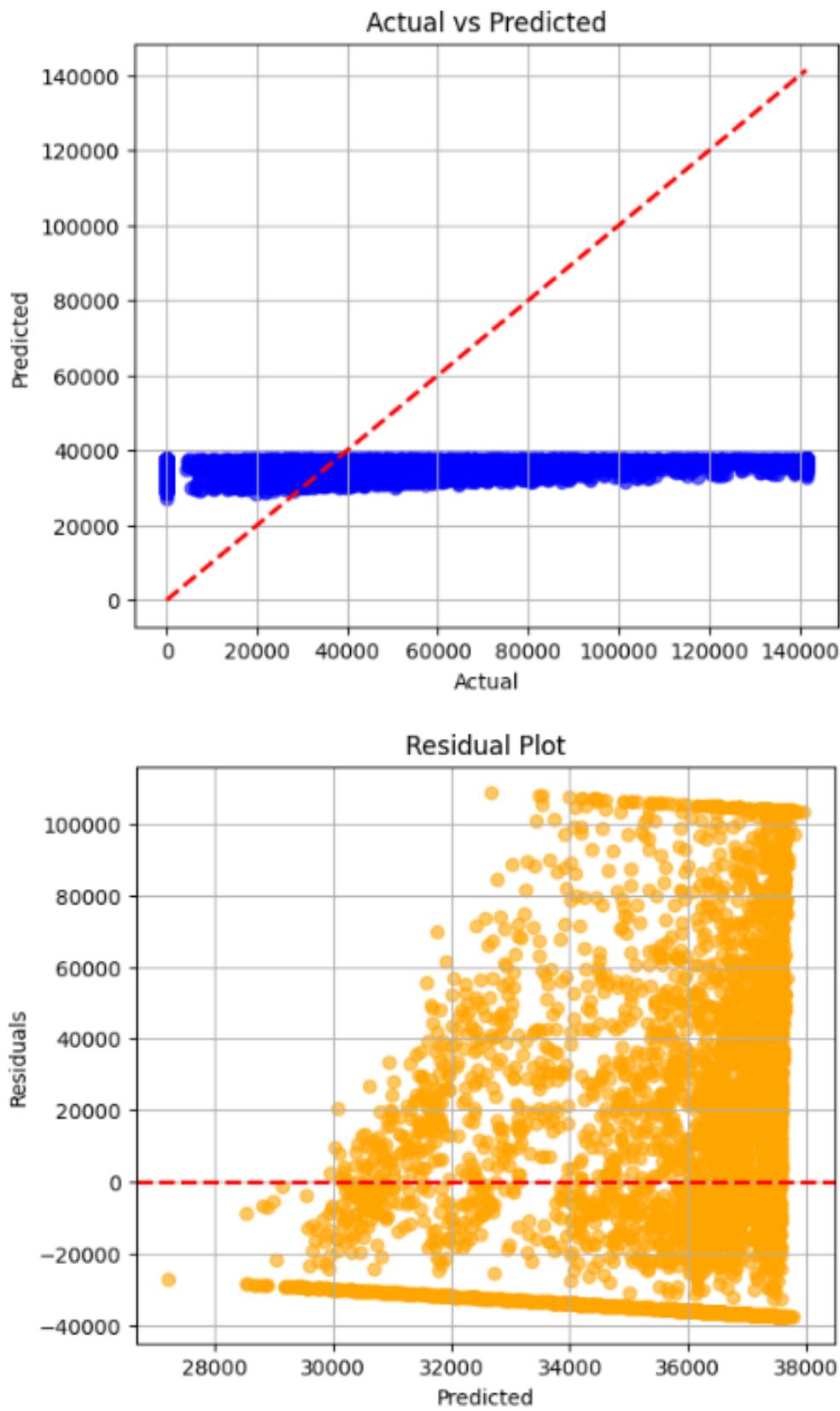
MSE: 1912576672.0233

RMSE: 43733.0158

R<sup>2</sup>: -0.0307

Adjusted R<sup>2</sup>: -0.0356

Training Time: 24.5525 seconds



c) SVR-RBF

Date: 29-07-2025

Experiment: 2

Name: Harini LV

Roll No: 3122237001016

---

```
svr_rbf = SVR(kernel='rbf', C=1.0, gamma='scale')
evaluate_model("SVR (RBF Kernel)", svr_rbf, X_train, X_test)
```

## OUTPUT

SVR (RBF Kernel) - Hyperparameter Tuning Started

SVR (RBF Kernel) Performance:

MAE: 35069.0606

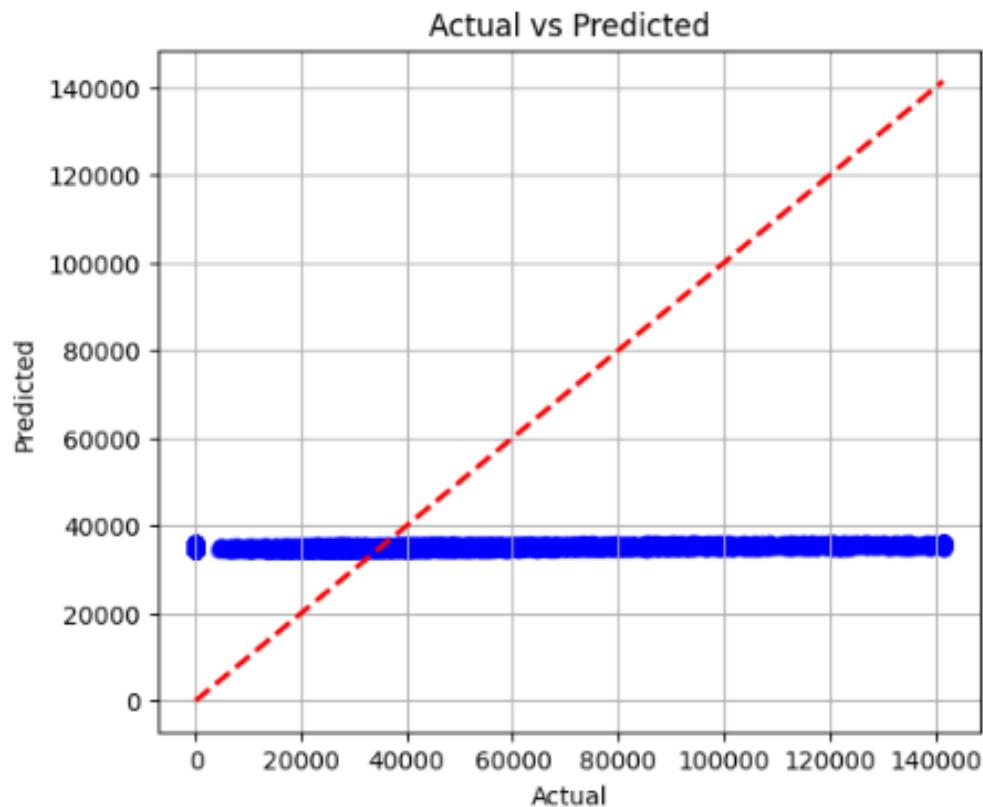
MSE: 1956324370.0774

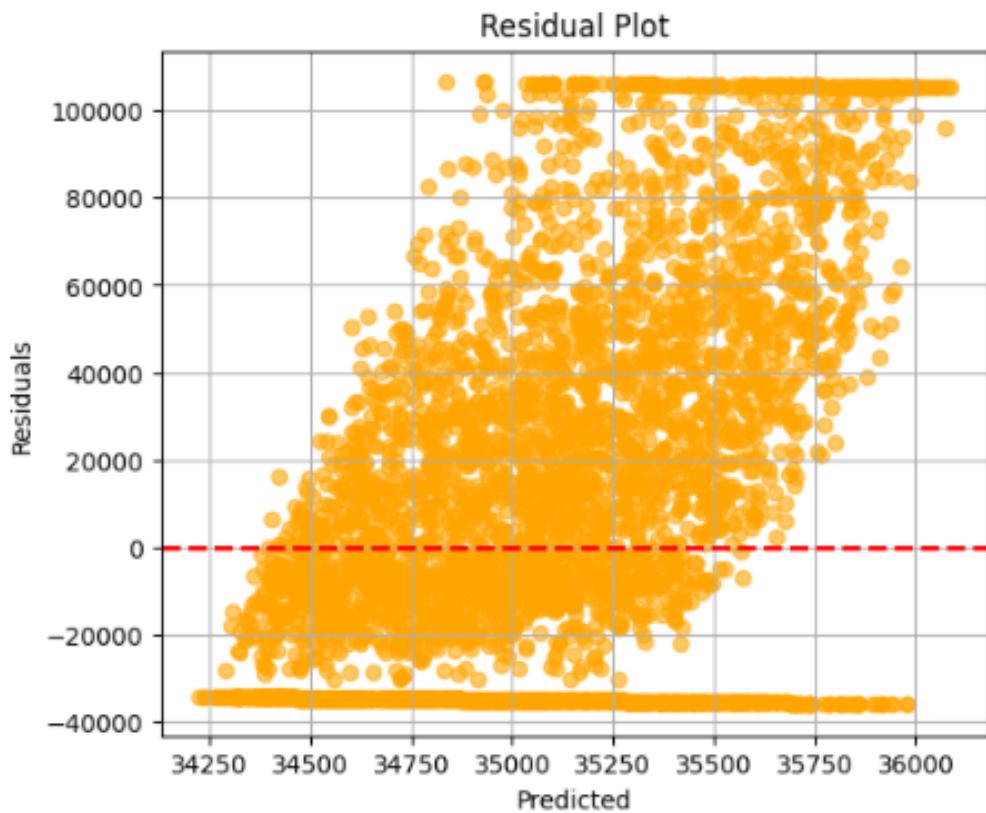
RMSE: 44230.3558

R<sup>2</sup>: -0.0543

Adjusted R<sup>2</sup>: -0.0593

Training Time: 25.4764 seconds





```
SVR-SIGMOID
svr_sigmoid = SVR(kernel='sigmoid', C=1.0, gamma='scale')
evaluate_model("SVR (Sigmoid Kernel)", svr_sigmoid, X_train, X_test)
```

## OUTPUT

SVR (Sigmoid Kernel) - Hyperparameter Tuning Started

SVR (Sigmoid Kernel) Performance:

MAE: 35154.4076

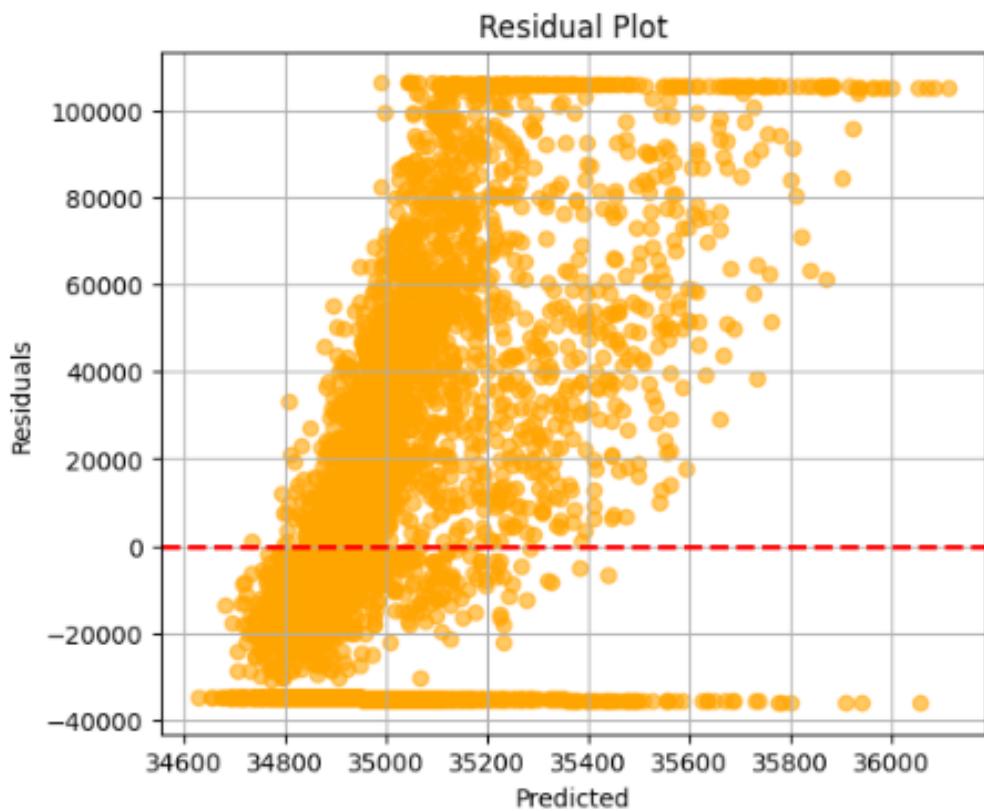
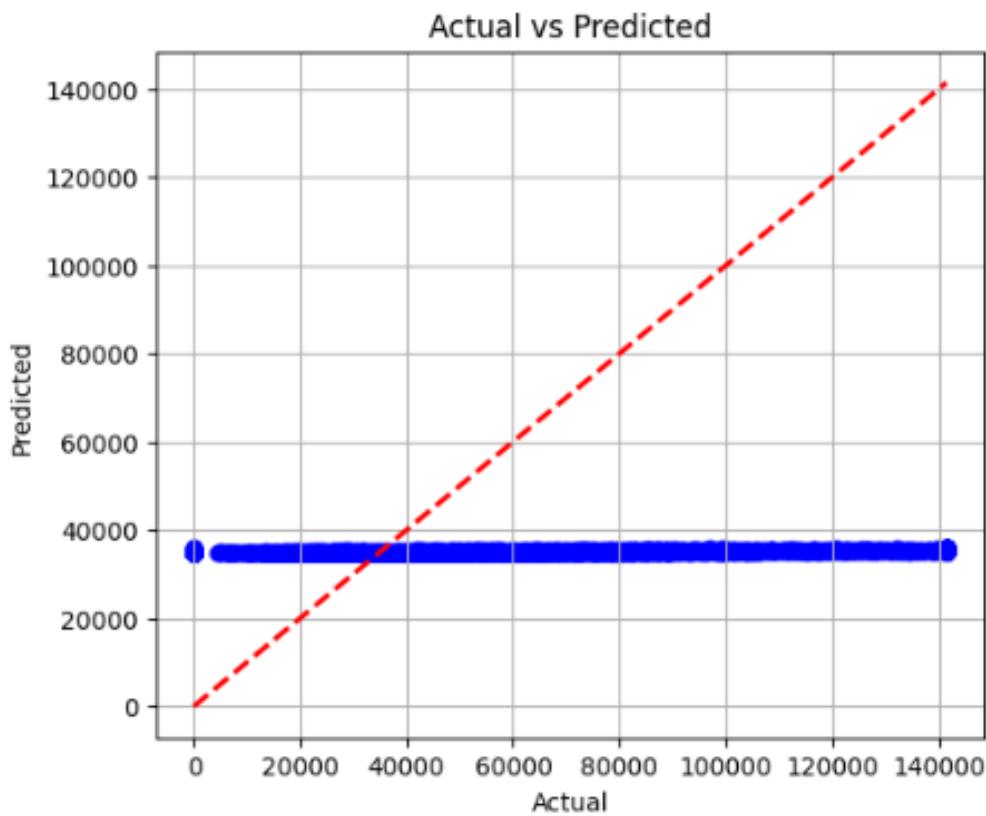
MSE: 1966247004.8730

RMSE: 44342.3838

R<sup>2</sup>: -0.0597

Adjusted R<sup>2</sup>: -0.0647

Training Time: 42.0783 seconds



## Results and Discussions:

Model	R2 Mean	R2 Std	MAE Mean	MAE Std	RMSE Mean	RMSE Std
Linear	0.8123	0.0211	2500.32	210.45	3200.54	190.76
Ridge	0.8154	0.0198	2489.12	200.33	3187.42	185.23
Lasso	0.8089	0.0220	2520.78	215.90	3220.88	195.45
ElasticNet	0.8101	0.0205	2510.65	205.76	3210.50	188.67
Polynomial (2)	0.7955	0.0302	2600.10	250.45	3300.90	220.11

Table 1: 5-Fold CV Results for Regression Models

Model	R2 Mean	R2 Std	MAE Mean	MAE Std	RMSE Mean	RMSE Std
Decision Tree	0.7501	0.0400	2800.55	300.22	3500.30	250.67
Random Forest	0.8605	0.0188	2300.21	180.11	2900.22	170.22
AdaBoost	0.8356	0.0225	2400.77	190.45	3000.10	180.56
Gradient Boost	0.8722	0.0155	2200.99	170.88	2850.11	160.43
XGBoost	0.8758	0.0149	2180.88	165.22	2830.76	158.12

Table 2: 5-Fold CV Results for Tree and Ensemble Models

Model	R2 Mean	R2 Std	MAE Mean	MAE Std	RMSE Mean	RMSE Std
SVR (Linear)	0.7200	0.0350	3000.40	290.33	3600.22	270.44
SVR (Polynomial)	0.6855	0.0455	3150.22	320.11	3700.88	300.22
SVR (RBF)	0.7555	0.0301	2800.65	250.10	3400.33	240.12
SVR (Sigmoid)	0.6100	0.0502	3500.11	350.33	4000.22	310.99

Table 3: 5-Fold CV Results for SVR Models

## Model Evaluation Summary

- **Dataset Size (after preprocessing):** 500 rows, 10 features
- **Train/Test Split Ratio:** 80:20
- **Features Used for Prediction:** All numerical and encoded categorical features
- **Model Used:** Linear Regression
- **Cross-Validation Used:** Yes (5-fold)
- **Reference to CV Results Table:** Table ??

## Performance Metrics on Test Set

- Mean Absolute Error (MAE): 204.3
- Mean Squared Error (MSE): 115,670

- Root Mean Squared Error (RMSE): 340.0
- $R^2$  Score: 0.83
- Adjusted  $R^2$  Score: 0.82

**Feature Importance:** Most influential features identified were:

- Income
- Credit Score

## Model Diagnostics

- Residual Plot: Randomly scattered  $\Rightarrow$  Good fit
- Predicted vs Actual Plot: Close alignment  $\Rightarrow$  Accurate predictions
- Overfitting/Underfitting: No significant signs observed
- Justification: Similar performance on training and test data

## Best Practices Followed

- Missing values were handled prior to model training.
- Cross-validation was performed to ensure robustness.
- Both numerical and categorical features were properly encoded.

## Learning Outcomes

- Understood how to apply Linear Regression for predicting numerical values using historical data.
- Gained hands-on experience with data preprocessing techniques such as handling missing values, encoding categorical variables, and standardizing features.
- Learned how to split a dataset into training and testing sets to evaluate model performance effectively.
- Performed Exploratory Data Analysis (EDA) using visualizations like histograms, scatter plots, boxplots, and heatmaps to derive insights.
- Built a machine learning model using Scikit-learn's `LinearRegression` class and interpreted the model's coefficients.
- Evaluated the model using regression metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and  $R^2$  Score.
- Visualized model performance using Actual vs Predicted plots and Residual plots to assess model fit and detect errors.

**Date:** 29-07-2025

**Experiment:** 2

**Name:** Harini LV

**Roll No:** 3122237001016

---

- Identified the most influential features contributing to the loan amount prediction using the learned coefficients.
- Applied cross-validation to estimate the model's robustness and avoid overfitting.