

Sri Sivasubramaniya Nadar College of Engineering, Chennai
(An Autonomous Institution Affiliated to Anna University)

Degree & Branch	B.E. Computer Science & Engineering	Semester	VI
Subject Code & Name	UCS2612 – Machine Learning Algorithms Laboratory		
Academic Year	2025–2026 (Even)	Batch	2023–2027
Due Date	27.01.2026		

Experiment 3: Regression Analysis using Linear and Regularized Models

Objective

To implement linear and regularized regression models for predicting a continuous target variable, evaluate their performance using multiple metrics, visualize model behavior, and analyze overfitting, underfitting, and bias–variance characteristics.

Dataset

A real-world regression dataset containing numerical and categorical features related to loan applications is used. The target variable is the **loan amount sanctioned**.

Dataset reference:

- Kaggle: [Predict Loan Amount Data](#)

Code

```
[140]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score,
↪ f1_score, confusion_matrix, roc_curve, auc
```

```
[190]: df = pd.read_csv('train.csv')
df
```

```
[191]: df.describe()
```

```
[192]: df.info()
```

```
[193]: print("columns")
df.columns
```

columns

```

[194]: df = df.drop(columns=['Customer ID', 'Name',"Property ID"])

[195]: df.isnull().sum()

[203]: num_cols = ['Age', 'Income (USD)', 'Loan Amount Request (USD)',
                  'Current Loan Expenses (USD)', 'Dependents', 'Credit Score', 'Property_
                  ↳Age',
                  'Property Price', 'Loan Sanction Amount (USD)']

cat_cols = [
    'Gender', 'Type of Employment', 'Has Active Credit Card',
    'Property Location', 'No. of Defaults', 'Income Stability', 'Property Type',
    ↳'Co-Applicant']

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

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

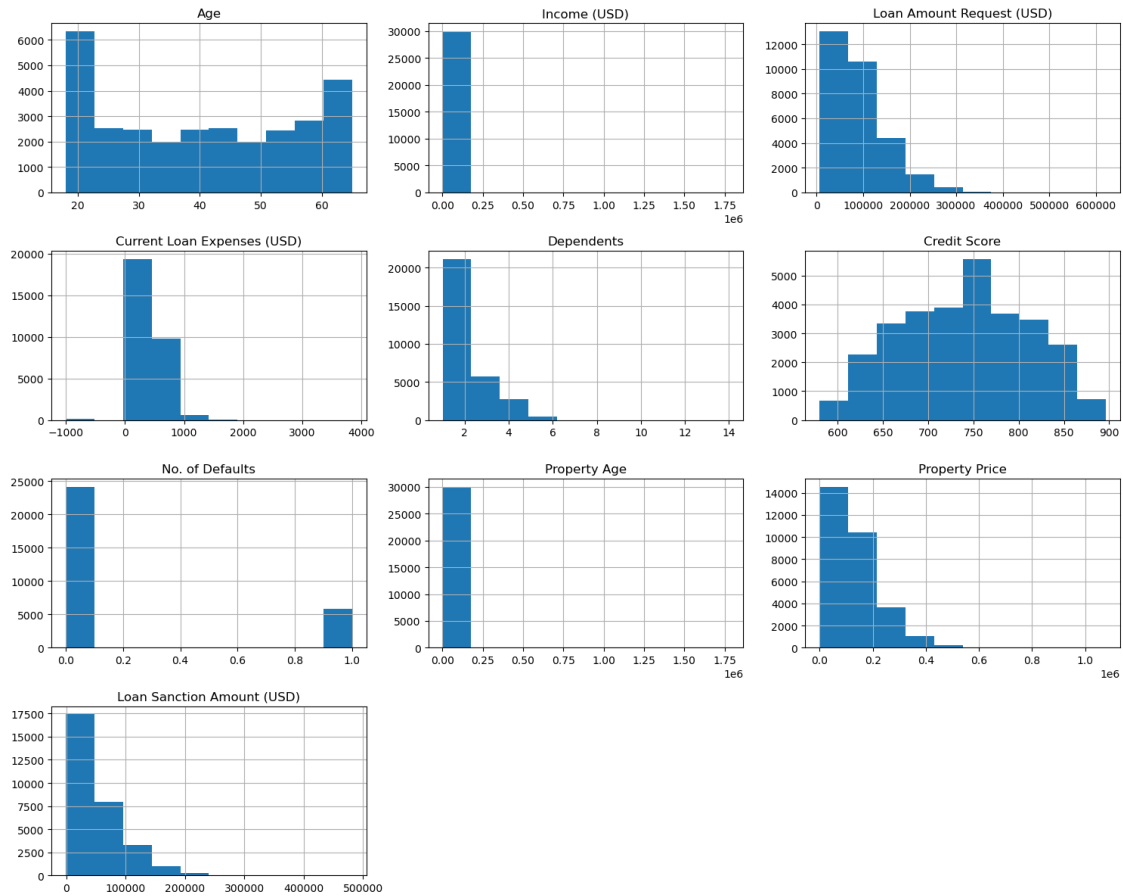
[198]: df['Type of Employment'] = df['Type of Employment'].fillna('Unknown')

[199]: df.isnull().sum()

[200]: df[num_cols].describe()

[201]: # Bar plot
df[num_cols].hist(figsize=(15,12))
plt.tight_layout()
plt.show()

```



```
[204]: # Box plot
import math

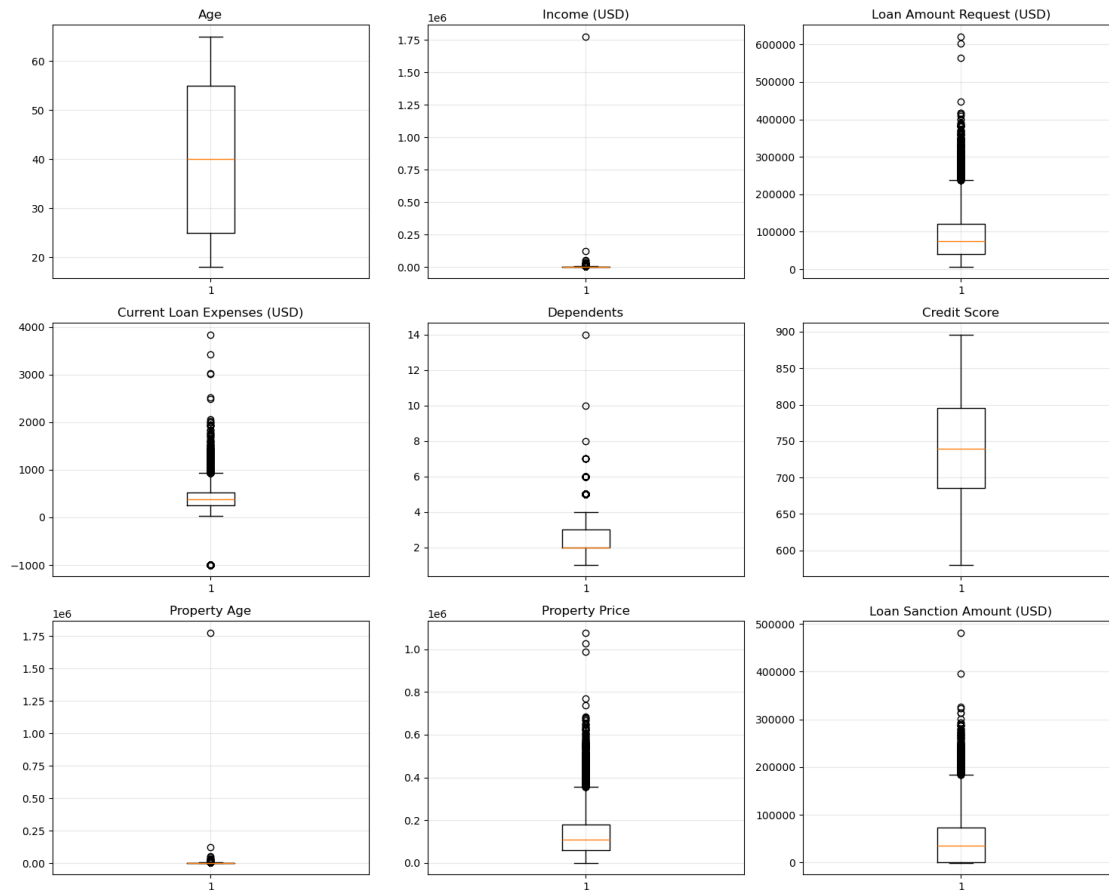
n = len(num_cols)
cols = 3
rows = math.ceil(n / cols)

fig, axes = plt.subplots(rows, cols, figsize=(5*cols, 4*rows))
axes = axes.flatten()

for i, col in enumerate(num_cols):
    axes[i].boxplot(df[col].dropna())
    axes[i].set_title(col)
    axes[i].grid(True, alpha=0.3)

# Remove unused subplots
for j in range(i+1, len(axes)):
    fig.delaxes(axes[j])
```

```
plt.tight_layout()
plt.show()
```



Huge outliers exist

```
[205]: df[num_cols].describe()
```

```
[206]: #winorization
outlier_summary = {}
for col in num_cols:
    if df[col].nunique() <= 2:
        continue

    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_limit = Q1 - 1.5 * IQR
    upper_limit = Q3 + 1.5 * IQR
```

```

lower_outlier = (df[col] < lower_limit).sum()
upper_outlier = (df[col] > upper_limit).sum()
df[col] = df[col].clip(lower_limit, upper_limit) # outlier removed df

outlier_summary[col] = {
    'IQR' : IQR,
    'lower_limit' : lower_limit,
    'upper_limit' : upper_limit,
    'lower_outlier' : lower_outlier ,
    'upper_outlier' : upper_outlier,
    'total_outlier' : lower_outlier + upper_outlier,
}
#df[col] = df[col].clip(lower_limit, upper_limit) # outlier removed df

outlier_df = pd.DataFrame(outlier_summary).T
outlier_df

```

```

[206]: df[num_cols].describe()

```

```

[207]: # Box plot after outlier treatment

n = len(num_cols)
cols = 3
rows = math.ceil(n / cols)

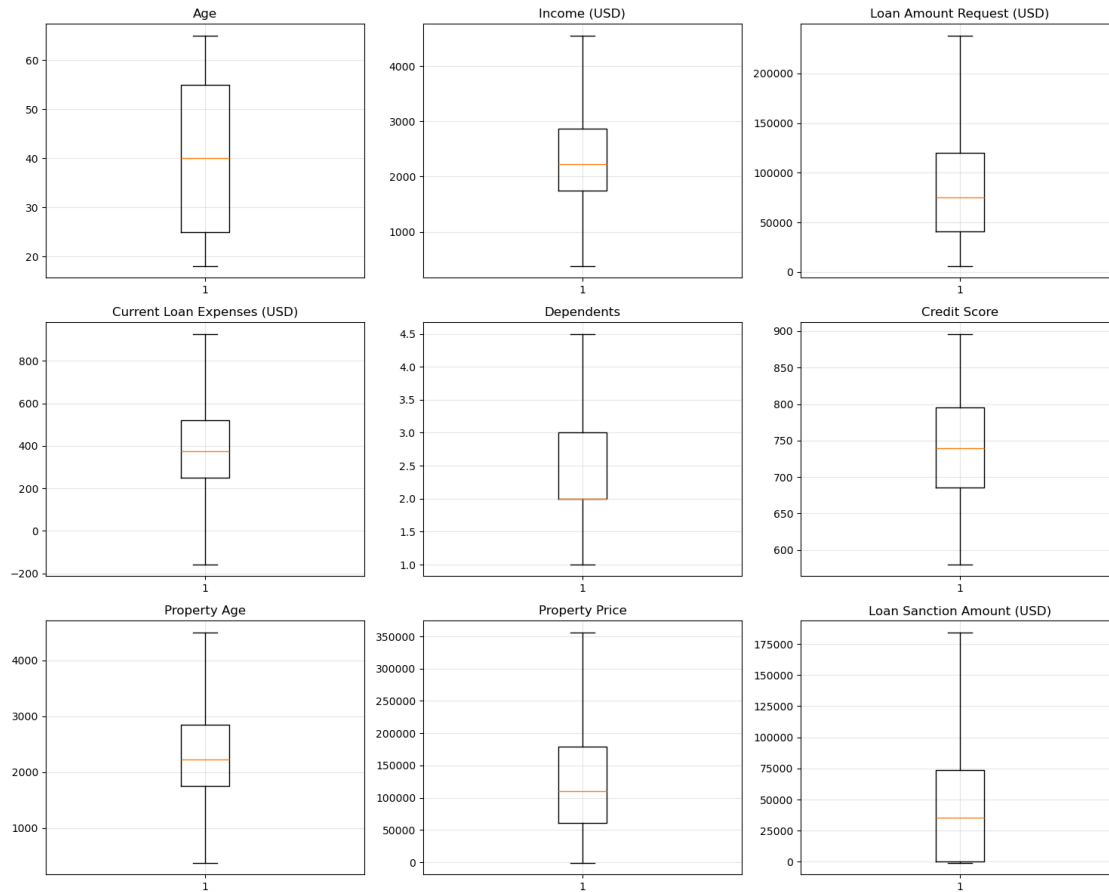
fig, axes = plt.subplots(rows, cols, figsize=(5*cols, 4*rows))
axes = axes.flatten()

for i, col in enumerate(num_cols):
    axes[i].boxplot(df[col].dropna())
    axes[i].set_title(col)
    axes[i].grid(True, alpha=0.3)

# Remove unused subplots
for j in range(i+1, len(axes)):
    fig.delaxes(axes[j])

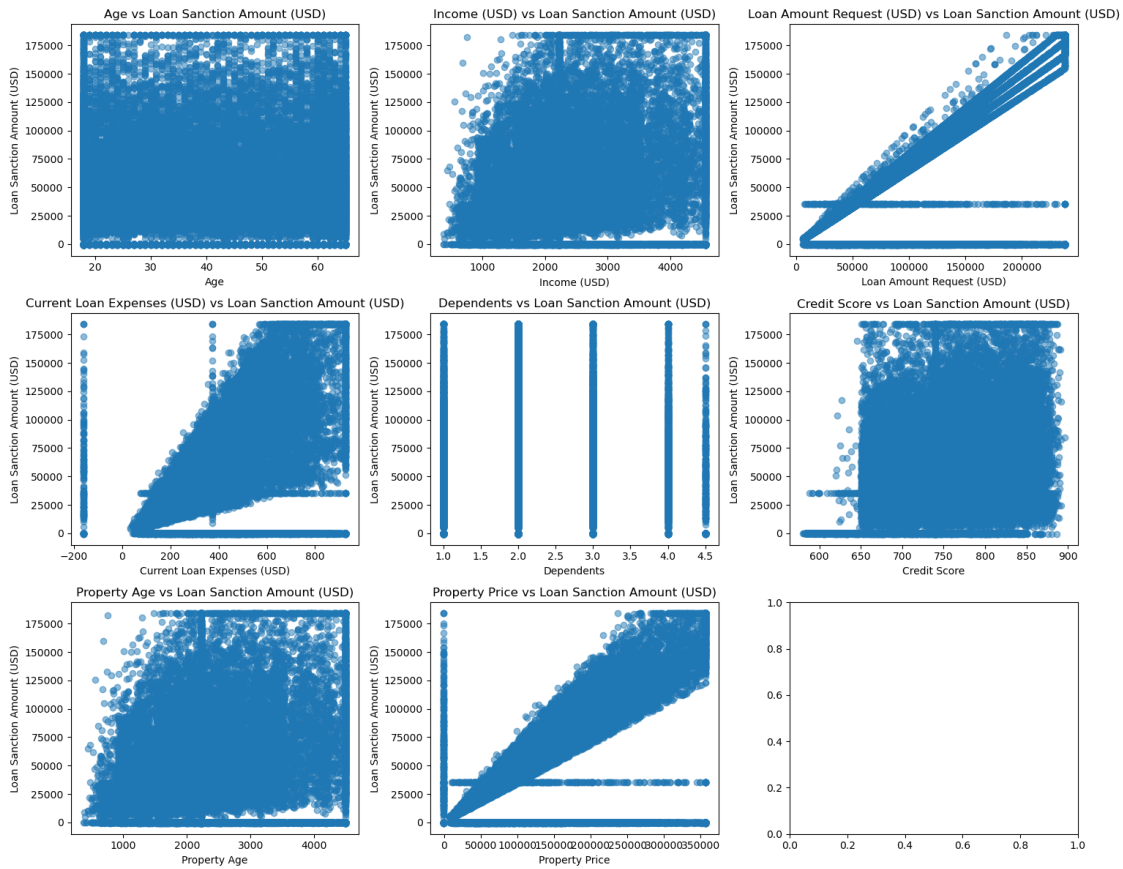
plt.tight_layout()
plt.show()

```



```
[210]: target = 'Loan Sanction Amount (USD)'
fig, axes = plt.subplots(rows,cols, figsize = (15,12))
axes = axes.flatten()

for i,col in enumerate(num_cols):
    if col == target:
        continue
    axes[i].scatter(df[col], df[target],alpha=0.5)
    axes[i].set_xlabel(col)
    axes[i].set_ylabel(target)
    axes[i].set_title(f'{col} vs {target}')
plt.tight_layout()
plt.show()
```



```
[211]: plt.figure(figsize=(10,6))
sns.heatmap(df[num_cols].corr(), annot=True)
plt.title("Correlational matrix")
plt.show()
```



```
[212]: y = df['Loan Sanction Amount (USD)']
X = df.drop(columns = ['Loan Sanction Amount (USD)'])
num_cols = [col for col in num_cols if col != 'Loan Sanction Amount (USD)']
```

```
[213]: # ANOVA
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import f_classif

selector = SelectKBest(score_func=f_classif, k=5)
X_anova_selected = selector.fit_transform(X[num_cols], y)

anova_scores = pd.DataFrame({
    'Feature': num_cols,
    'ANOVA F-Score': selector.scores_
}).sort_values(by='ANOVA F-Score', ascending=False)

anova_scores
```

```
[214]: selected_features = [
    'Loan Amount Request (USD)',
```

```

    'Property Price',
    'Current Loan Expenses (USD)',
    'Age',
    'Income (USD)',
    'Property Age',
    'Credit Score'
]

```

```

[221]: target = 'Loan Sanction Amount (USD)'
X = df[selected_features]
y = df[target]

test_df = pd.read_csv('test.csv')
X_test = test_df[selected_features]

```

```

[222]: from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import cross_val_predict
import numpy as np

lr = LinearRegression()

y_cv_pred_lr = cross_val_predict(lr, X, y, cv=5)

```

```

[223]: from sklearn.linear_model import Ridge, Lasso, ElasticNet

ridge = Ridge()
lasso = Lasso(max_iter=10000)
elastic = ElasticNet(max_iter=10000)

```

```

[249]: from sklearn.model_selection import GridSearchCV

ridge_params = {'alpha': [0.01, 0.1, 1, 10, 100]}
lasso_params = {'alpha': [0.001, 0.01, 0.1, 1, 10]}
elastic_params = {
    'alpha': [0.001, 0.01, 0.1, 1, 10],
    'l1_ratio': [0.2, 0.5, 0.8]
}

ridge_cv = GridSearchCV(Ridge(), ridge_params, cv=5,
    ↳scoring='neg_mean_squared_error')
lasso_cv = GridSearchCV(Lasso(max_iter=10000), lasso_params, cv=5,
    ↳scoring='neg_mean_squared_error')
elastic_cv = GridSearchCV(ElasticNet(max_iter=10000), elastic_params, cv=5,
    ↳scoring='neg_mean_squared_error')

ridge_cv.fit(X, y)

```

```
lasso_cv.fit(X, y)
elastic_cv.fit(X, y)
```

```
[254]: ridge_results = pd.DataFrame(ridge_cv.cv_results_)

ridge_results[['param_alpha', 'mean_test_score', 'std_test_score']]
```

```
[255]: ridge_results['RMSE'] = np.sqrt(-ridge_results['mean_test_score'])

ridge_results[['param_alpha', 'RMSE']]
```

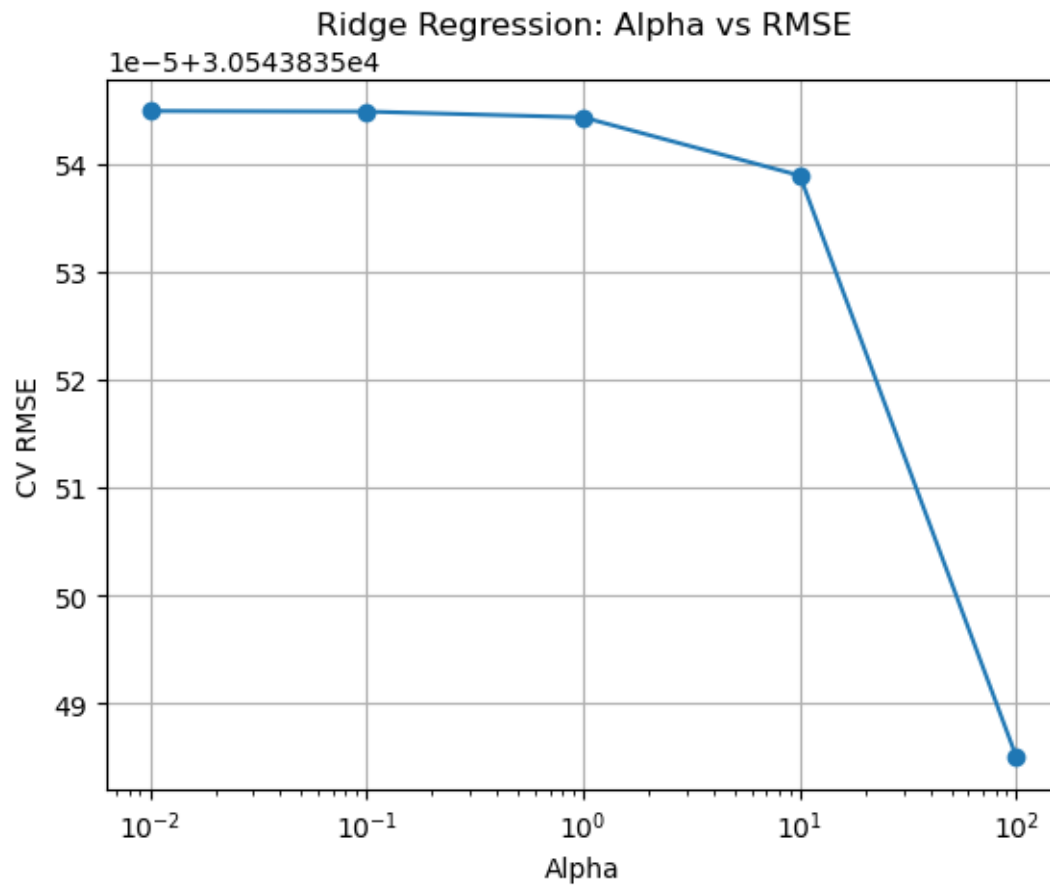
```
[252]: lasso_results = pd.DataFrame(lasso_cv.cv_results_)
lasso_results['RMSE'] = np.sqrt(-lasso_results['mean_test_score'])

elastic_results = pd.DataFrame(elastic_cv.cv_results_)
elastic_results['RMSE'] = np.sqrt(-elastic_results['mean_test_score'])
```

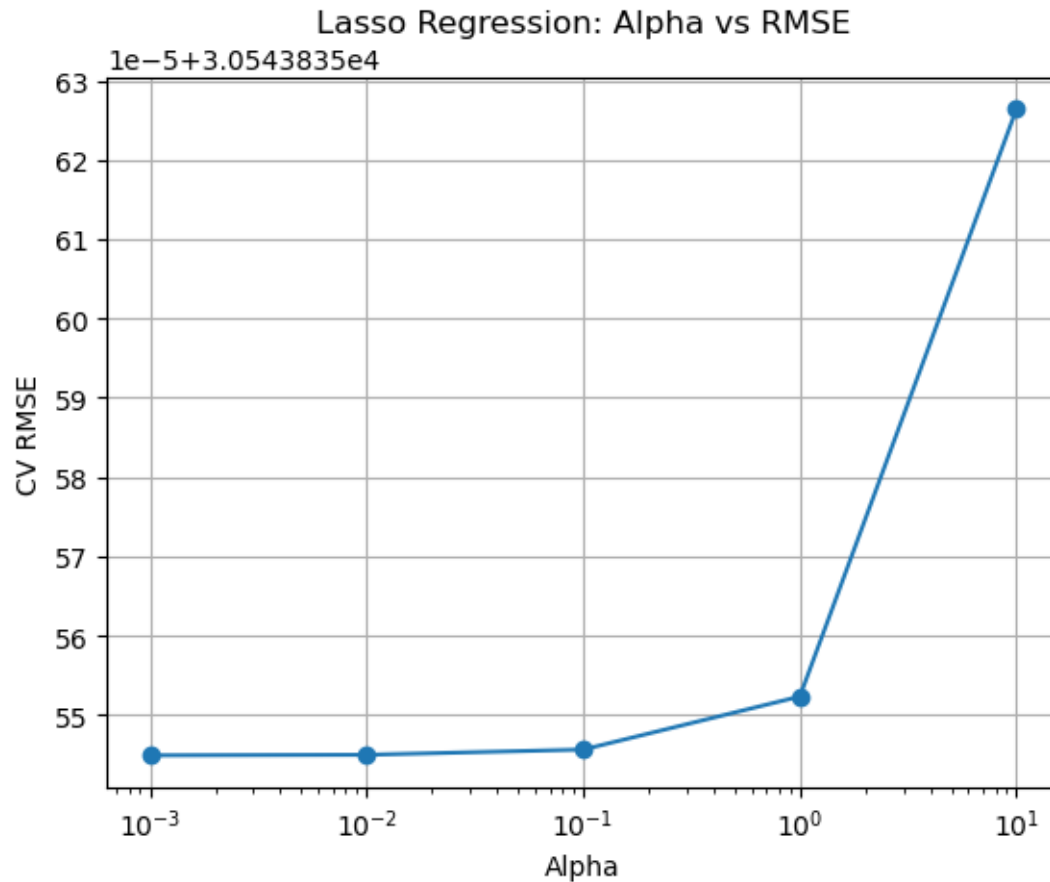
```
[253]: ridge_results['RMSE']
lasso_results['RMSE']
elastic_results['RMSE']

print(pd.DataFrame({'Ridge':ridge_results['RMSE'] , 'Lasso':
↳lasso_results['RMSE'] , 'Elastic':elastic_results['RMSE'] })))
```

```
[236]: plt.plot(
    ridge_results['param_alpha'],
    ridge_results['RMSE'],
    marker='o'
)
plt.xscale('log')
plt.xlabel('Alpha')
plt.ylabel('CV RMSE')
plt.title('Ridge Regression: Alpha vs RMSE')
plt.grid(True)
plt.show()
```



```
[256]: plt.plot(
    lasso_results['param_alpha'],
    lasso_results['RMSE'],
    marker='o'
)
plt.xscale('log')
plt.xlabel('Alpha')
plt.ylabel('CV RMSE')
plt.title('Lasso Regression: Alpha vs RMSE')
plt.grid(True)
plt.show()
```



```
[257]: ridge_results[['param_alpha', 'RMSE']].sort_values('RMSE')
```

```
[258]: elastic_results[['param_alpha', 'param_l1_ratio', 'RMSE']]\
      .sort_values('RMSE')\
      .head(10)
```

```
[259]: best_ridge = ridge_cv.best_estimator_
      best_lasso = lasso_cv.best_estimator_
      best_elastic = elastic_cv.best_estimator_
```

```
[260]: def evaluate(name, y_true, y_pred):
      return {
          "Model": name,
          "MAE": mean_absolute_error(y_true, y_pred),
          "RMSE": np.sqrt(mean_squared_error(y_true, y_pred)),
          "R2": r2_score(y_true, y_pred)
      }
```

```
[261]: results = []

results.append(evaluate(
    "Linear Regression",
    y,
    cross_val_predict(lr, X, y, cv=5)
))

results.append(evaluate(
    "Ridge",
    y,
    cross_val_predict(best_ridge, X, y, cv=5)
))

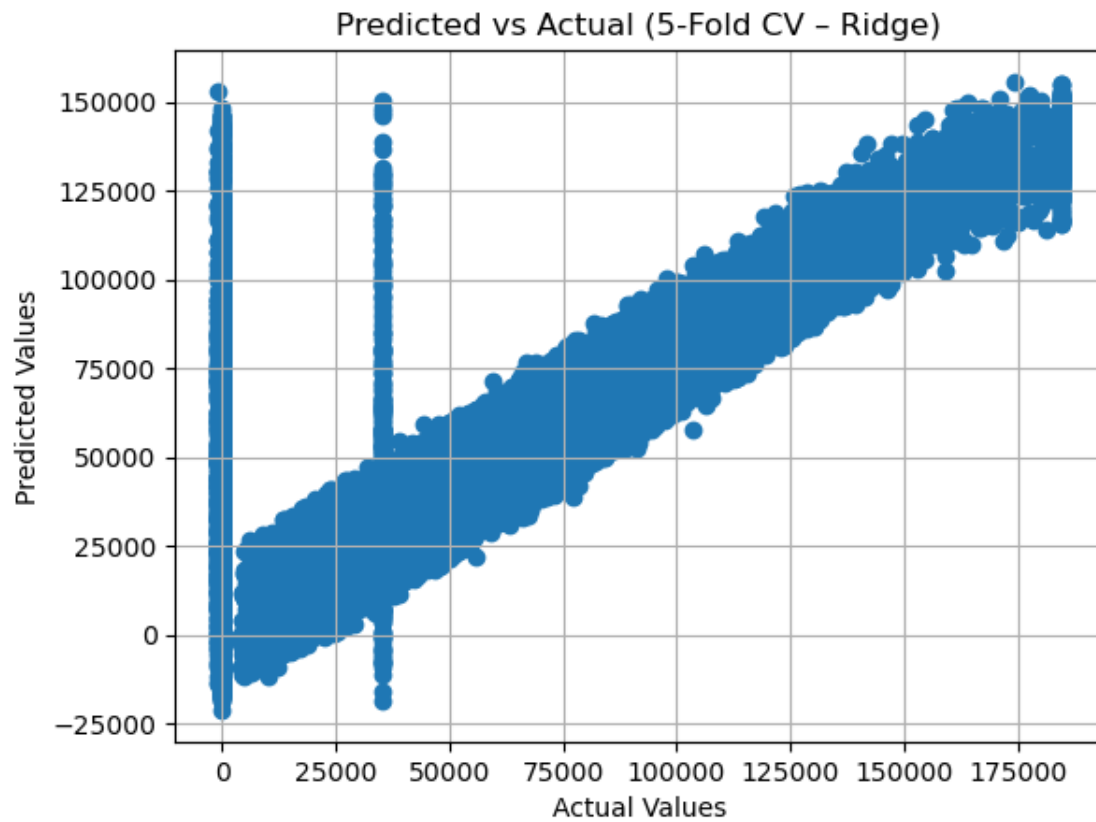
results.append(evaluate(
    "Lasso",
    y,
    cross_val_predict(best_lasso, X, y, cv=5)
))

results.append(evaluate(
    "Elastic Net",
    y,
    cross_val_predict(best_elastic, X, y, cv=5)
))

results_df = pd.DataFrame(results)
results_df
```

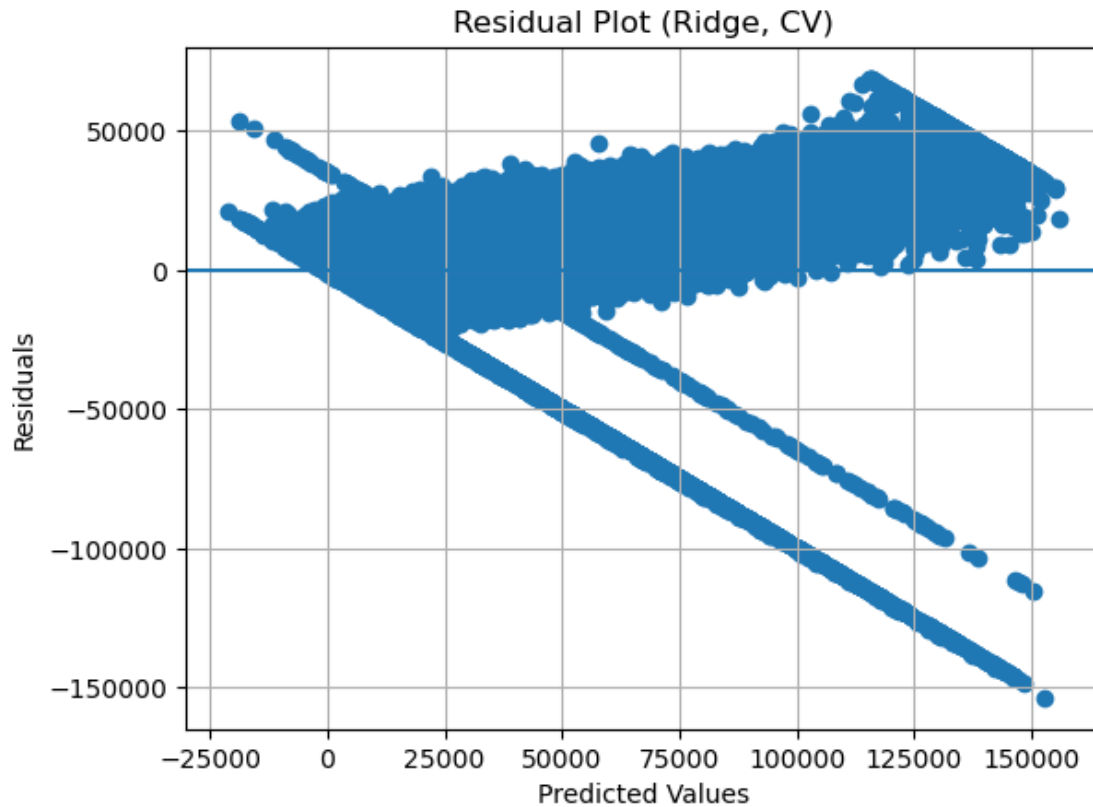
```
[227]: y_pred = cross_val_predict(best_ridge, X, y, cv=5)

plt.scatter(y, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Predicted vs Actual (5-Fold CV - Ridge)")
plt.grid(True)
plt.show()
```



```
[262]: residuals = y - y_pred

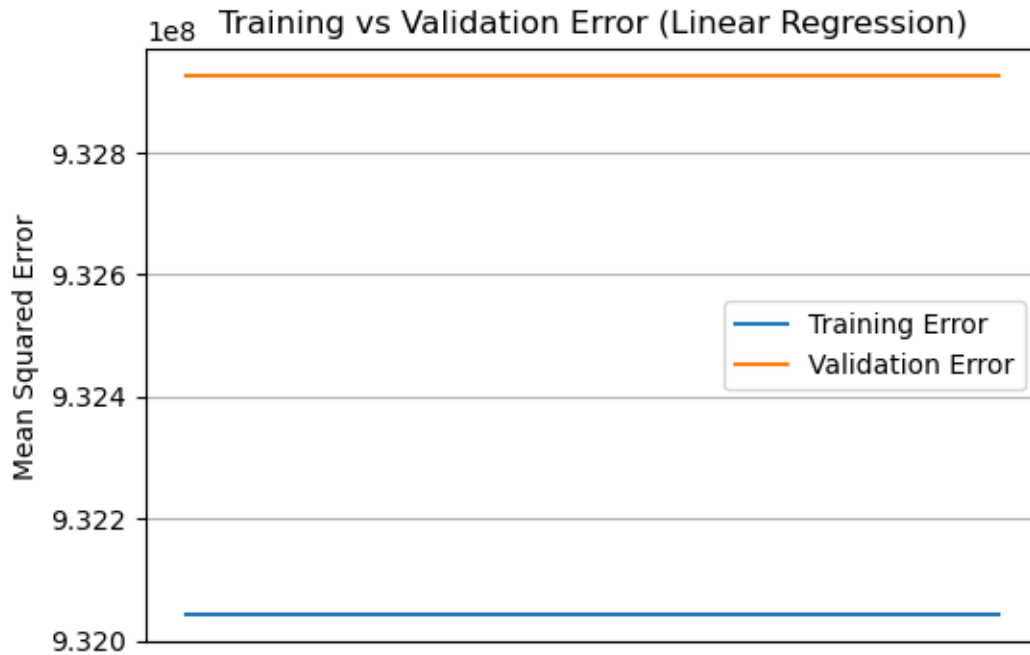
plt.scatter(y_pred, residuals)
plt.axhline(0)
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot (Ridge, CV)")
plt.grid(True)
plt.show()
```



```
[268]: lr = LinearRegression()
lr.fit(X, y)

train_mse = mean_squared_error(y, lr.predict(X))
val_mse = -cross_val_score(
    lr, X, y, cv=5, scoring='neg_mean_squared_error'
).mean()

plt.figure(figsize=(6,4))
plt.plot([0, 1], [train_mse, train_mse], label='Training Error')
plt.plot([0, 1], [val_mse, val_mse], label='Validation Error')
plt.xticks([])
plt.ylabel('Mean Squared Error')
plt.title('Training vs Validation Error (Linear Regression)')
plt.legend()
plt.grid(True)
plt.show()
```



```
[263]: alphas = ridge_params['alpha']
train_error = []
val_error = []

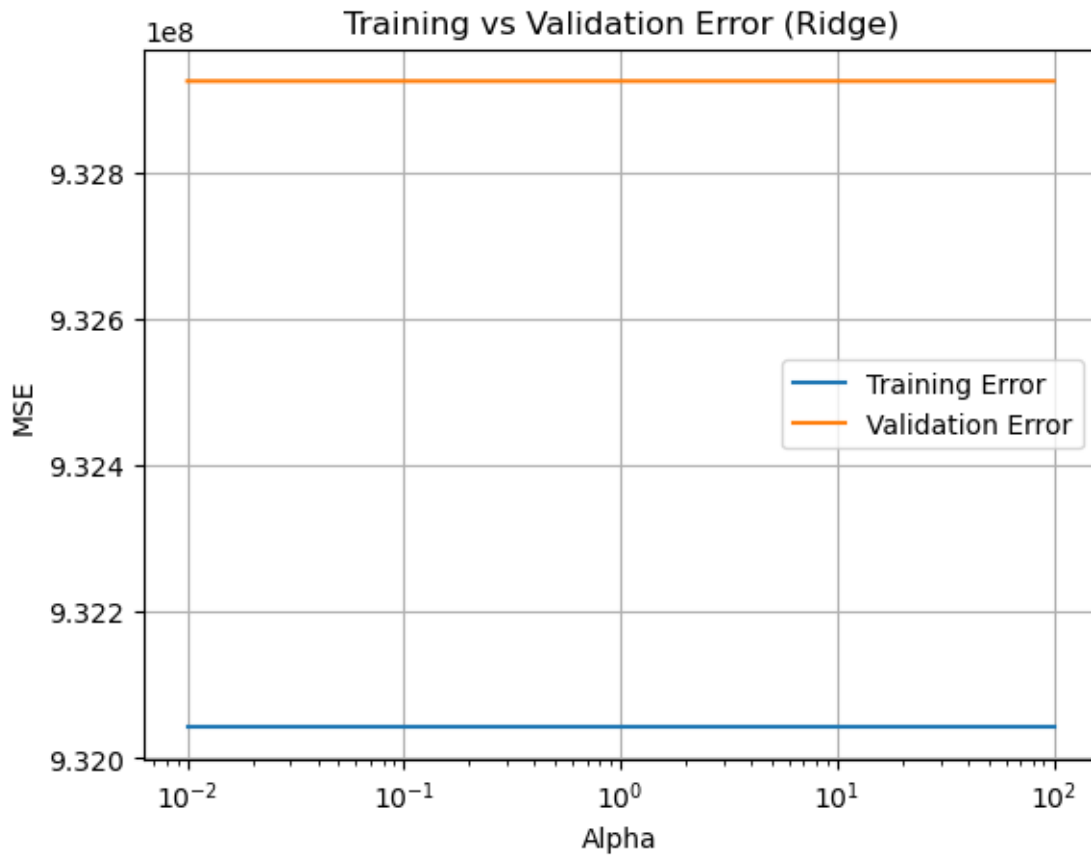
for a in alphas:
    model = Ridge(alpha=a)
    model.fit(X, y)

    train_error.append(
        mean_squared_error(y, model.predict(X))
    )

    val_error.append(
        -GridSearchCV(
            Ridge(alpha=a),
            {},
            cv=5,
            scoring='neg_mean_squared_error'
        ).fit(X, y).best_score_
    )

plt.plot(alphas, train_error, label="Training Error")
plt.plot(alphas, val_error, label="Validation Error")
plt.xscale("log")
plt.xlabel("Alpha")
```

```
plt.ylabel("MSE")
plt.title("Training vs Validation Error (Ridge)")
plt.legend()
plt.grid(True)
plt.show()
```



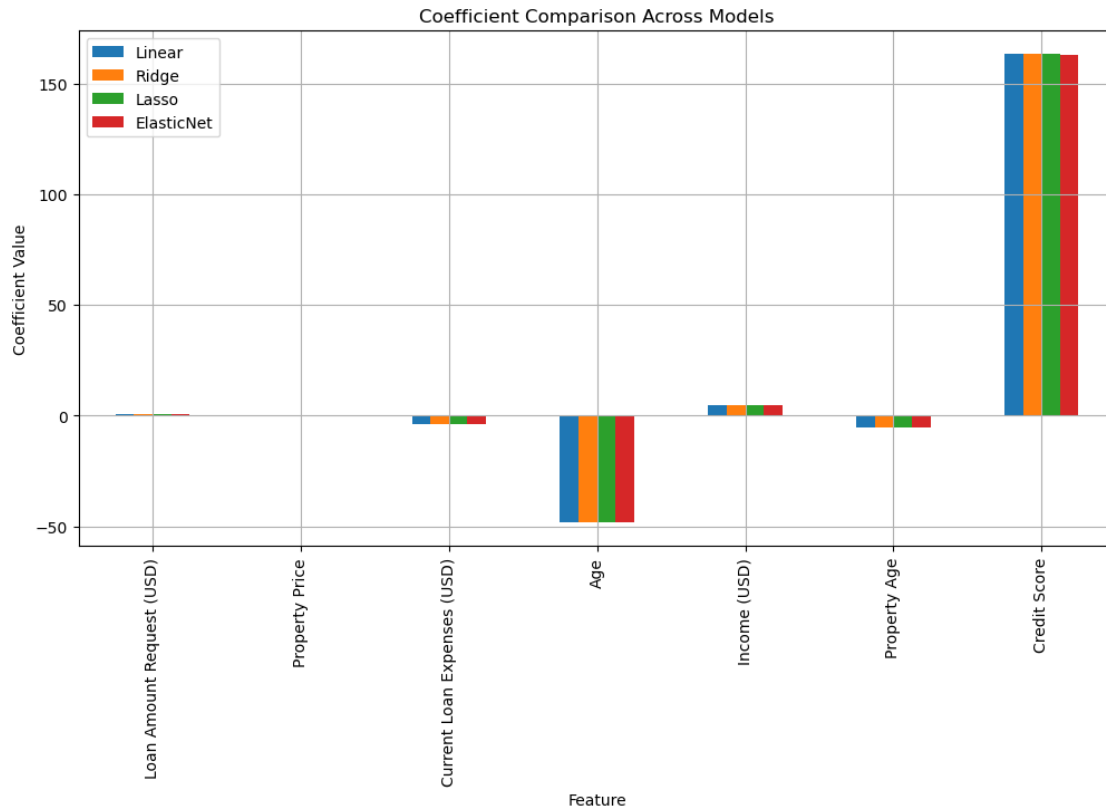
```
[230]: lr.fit(X, y)
best_ridge.fit(X, y)
best_lasso.fit(X, y)
best_elastic.fit(X, y)

coef_df = pd.DataFrame({
    "Feature": selected_features,
    "Linear": lr.coef_,
    "Ridge": best_ridge.coef_,
    "Lasso": best_lasso.coef_,
    "ElasticNet": best_elastic.coef_
})
```

```

coef_df.set_index("Feature").plot(kind="bar", figsize=(12,6))
plt.title("Coefficient Comparison Across Models")
plt.ylabel("Coefficient Value")
plt.grid(True)
plt.show()

```



```

[264]: from sklearn.model_selection import cross_val_score

alphas = [0.01, 0.1, 1, 10,100]

train_error = []
val_error = []

for a in alphas:
    model = Ridge(alpha=a)
    model.fit(X, y)

    # Training error (MSE)
    y_train_pred = model.predict(X)
    train_error.append(mean_squared_error(y, y_train_pred))

```

```

# Validation error (5-fold CV MSE)
cv_mse = -cross_val_score(
    model,
    X,
    y,
    cv=5,
    scoring='neg_mean_squared_error'
).mean()
val_error.append(cv_mse)

# Plot
plt.plot(alphas, train_error, marker='o', label='Training Error')
plt.plot(alphas, val_error, marker='o', label='Validation Error')
plt.xscale('log')
plt.xlabel('Alpha (Regularization Strength)')
plt.ylabel('Mean Squared Error')
plt.title('Training Error vs Validation Error (Ridge Regression)')
plt.legend()
plt.grid(True)
plt.show()

```



```

[265]: from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

alphas = [0.01, 0.1, 1, 10, 100, 1000]

train_error = []
val_error = []

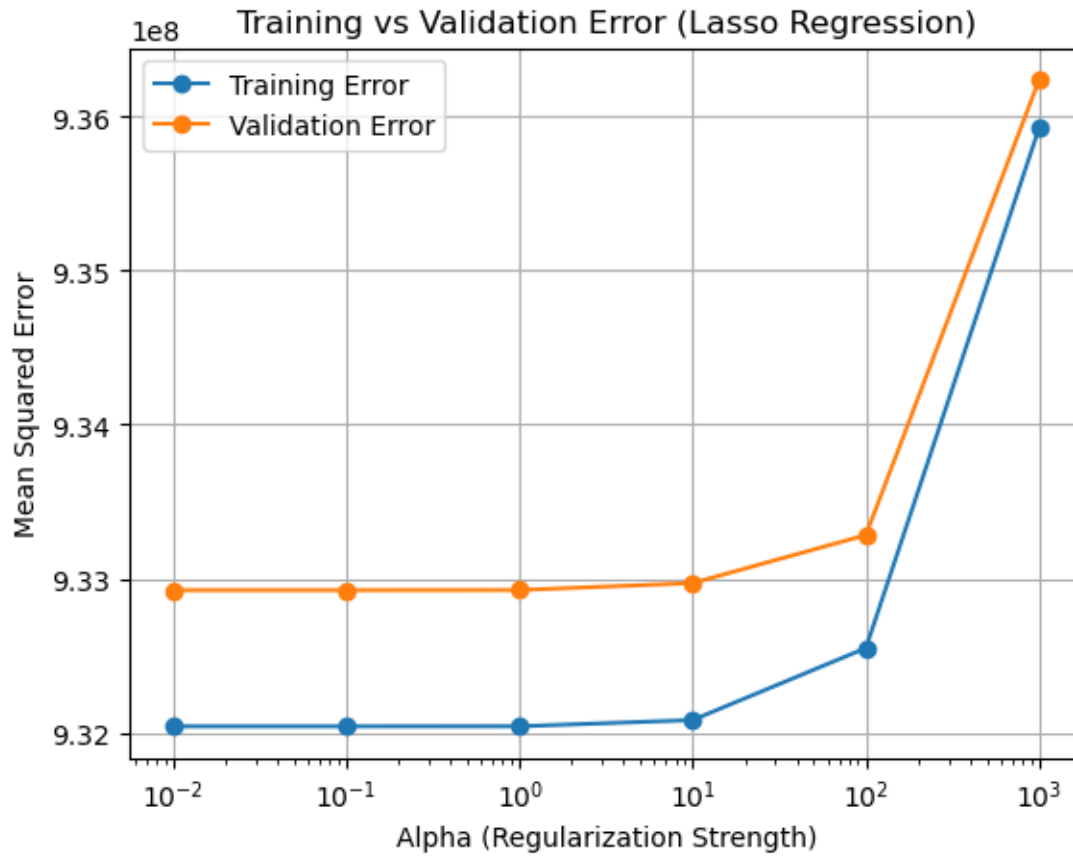
for a in alphas:
    model = Lasso(alpha=a, max_iter=10000)
    model.fit(X_scaled, y)

    # Training error
    train_error.append(
        mean_squared_error(y, model.predict(X_scaled))
    )

    # Validation error
    val_error.append(
        -cross_val_score(
            model,
            X_scaled,
            y,
            cv=5,
            scoring='neg_mean_squared_error'
        ).mean()
    )

plt.plot(alphas, train_error, marker='o', label='Training Error')
plt.plot(alphas, val_error, marker='o', label='Validation Error')
plt.xscale('log')
plt.xlabel('Alpha (Regularization Strength)')
plt.ylabel('Mean Squared Error')
plt.title('Training vs Validation Error (Lasso Regression)')
plt.legend()
plt.grid(True)
plt.show()

```



```
[267]: from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

alphas = [0.01, 0.1, 1, 10, 100, 1000]
l1_ratio = 0.5 # balanced Elastic Net

train_error = []
val_error = []

for a in alphas:
    model = ElasticNet(alpha=a, l1_ratio=l1_ratio, max_iter=10000)
    model.fit(X_scaled, y)
```

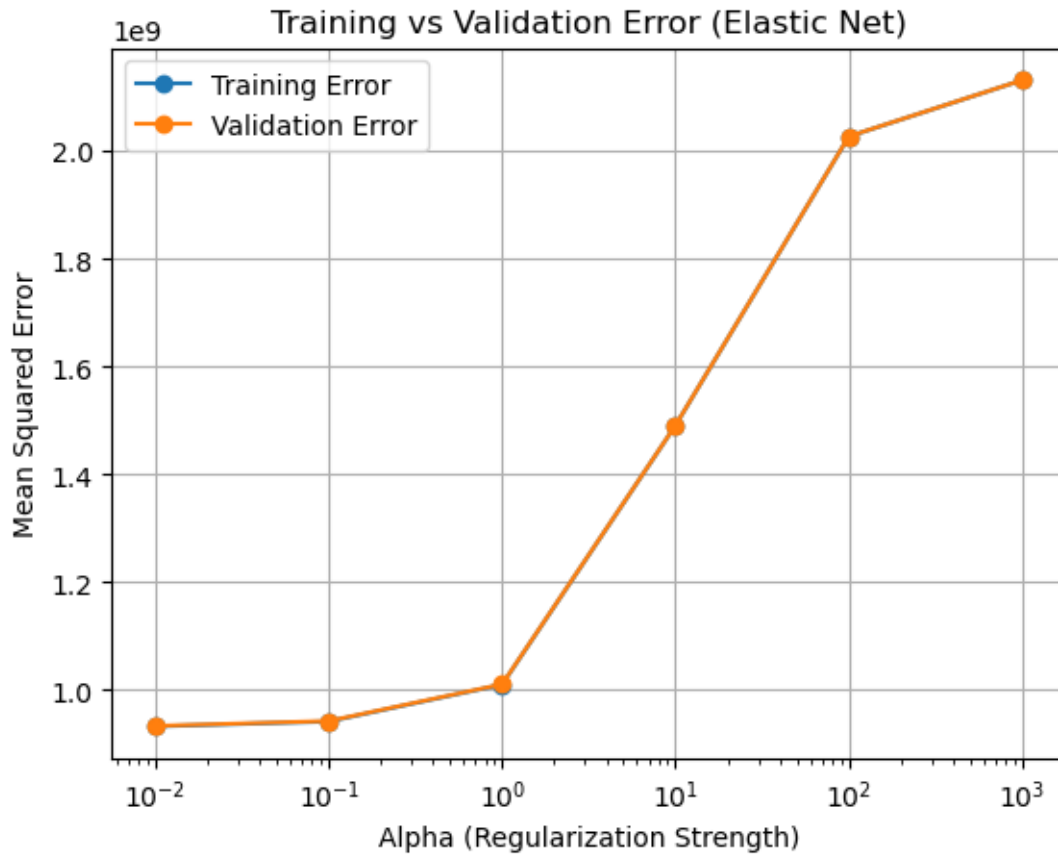
```

# Training error
train_error.append(
    mean_squared_error(y, model.predict(X_scaled))
)

# Validation error
val_error.append(
    -cross_val_score(
        model,
        X_scaled,
        y,
        cv=5,
        scoring='neg_mean_squared_error'
    ).mean()
)

# Plot
plt.plot(alphas, train_error, marker='o', label='Training Error')
plt.plot(alphas, val_error, marker='o', label='Validation Error')
plt.xscale('log')
plt.xlabel('Alpha (Regularization Strength)')
plt.ylabel('Mean Squared Error')
plt.title('Training vs Validation Error (Elastic Net)')
plt.legend()
plt.grid(True)
plt.show()

```



```
[269]: # lambda vs slope plots
# from sklearn.preprocessing import StandardScaler

# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X)    already done, so ignoring

alphas = np.logspace(-3, 3, 50)
coefs = []

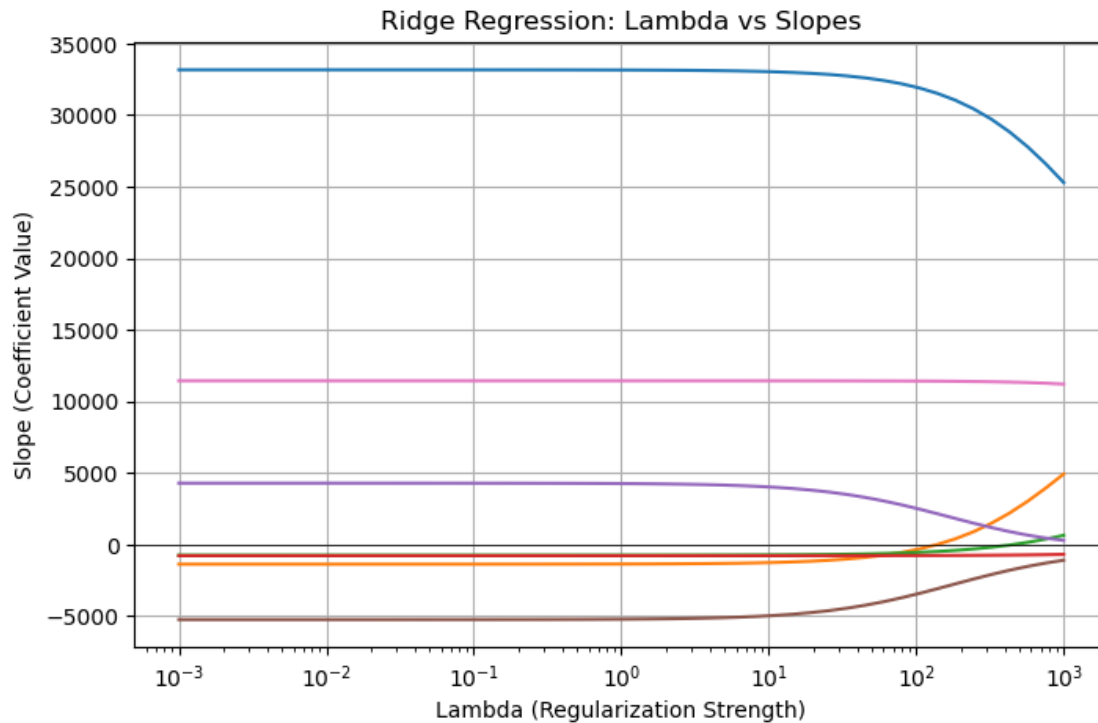
for a in alphas:
    ridge = Ridge(alpha=a)
    ridge.fit(X_scaled, y)
    coefs.append(ridge.coef_)

coefs = np.array(coefs)

plt.figure(figsize=(8,5))
for i in range(coefs.shape[1]):
```

```
plt.plot(alphas, coefs[:, i])

plt.xscale('log')
plt.xlabel('Lambda (Regularization Strength)')
plt.ylabel('Slope (Coefficient Value)')
plt.title('Ridge Regression: Lambda vs Slopes')
plt.axhline(0, color='black', linewidth=0.5)
plt.grid(True)
plt.show()
```



```
[270]: from sklearn.linear_model import Lasso

alphas = np.logspace(-3, 1, 50)
coefs = []

for a in alphas:
    lasso = Lasso(alpha=a, max_iter=10000)
    lasso.fit(X_scaled, y)
    coefs.append(lasso.coef_)

coefs = np.array(coefs)

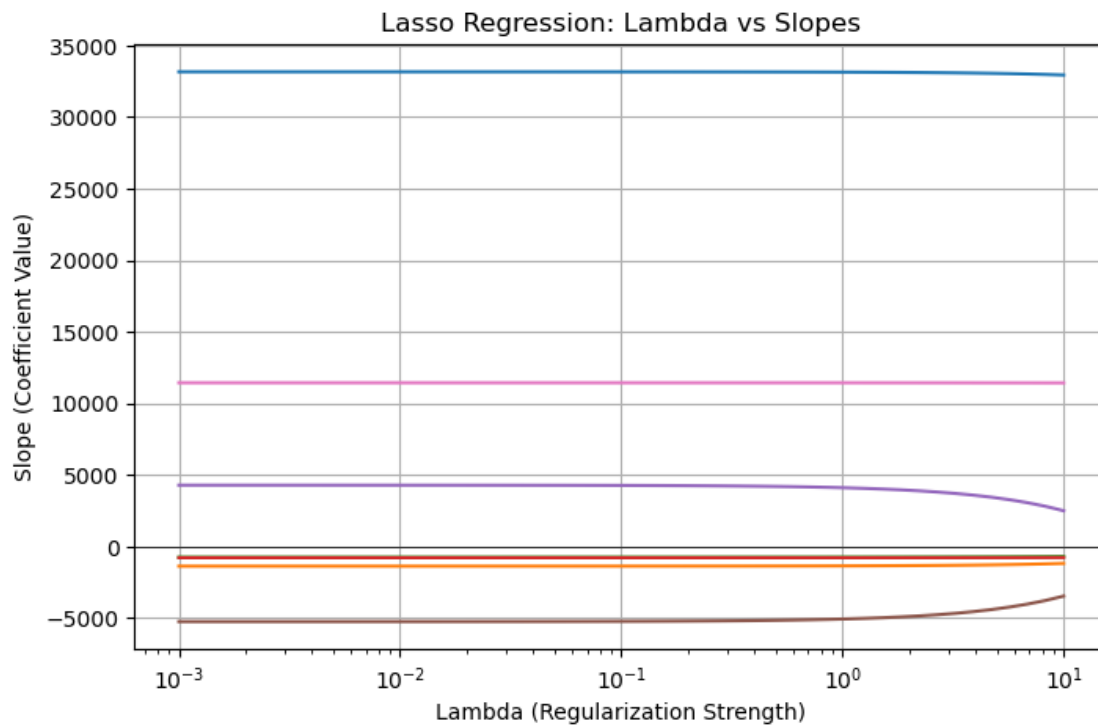
plt.figure(figsize=(8,5))
```

```

for i in range(coefs.shape[1]):
    plt.plot(alphas, coefs[:, i])

plt.xscale('log')
plt.xlabel('Lambda (Regularization Strength)')
plt.ylabel('Slope (Coefficient Value)')
plt.title('Lasso Regression: Lambda vs Slopes')
plt.axhline(0, color='black', linewidth=0.5)
plt.grid(True)
plt.show()

```



```

[271]: from sklearn.linear_model import ElasticNet

alphas = np.logspace(-3, 2, 50)
l1_ratio = 0.5 # balance between L1 and L2
coefs = []

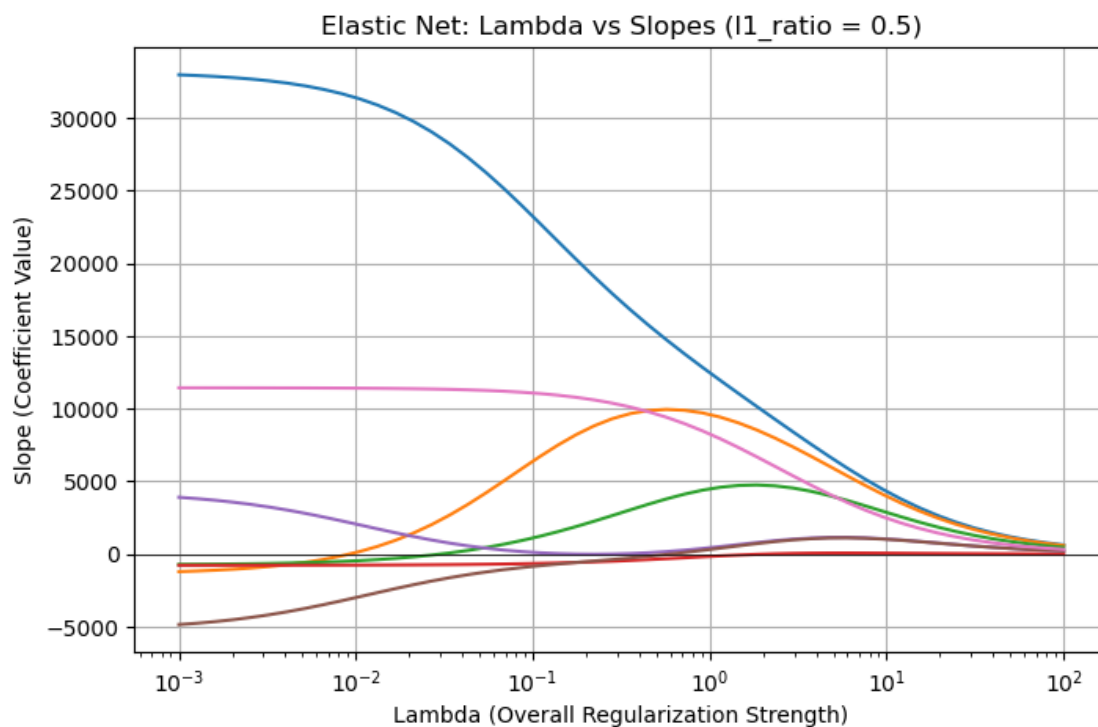
for a in alphas:
    enet = ElasticNet(alpha=a, l1_ratio=l1_ratio, max_iter=10000)
    enet.fit(X_scaled, y)
    coefs.append(enet.coef_)

coefs = np.array(coefs)

```

```
plt.figure(figsize=(8,5))
for i in range(coefs.shape[1]):
    plt.plot(alphas, coefs[:, i])

plt.xscale('log')
plt.xlabel('Lambda (Overall Regularization Strength)')
plt.ylabel('Slope (Coefficient Value)')
plt.title('Elastic Net: Lambda vs Slopes (l1_ratio = 0.5)')
plt.axhline(0, color='black', linewidth=0.5)
plt.grid(True)
plt.show()
```



```
[272]: # Cross-Validation Performance (K = 5)
def cv_metrics(model, X, y):
    y_pred = cross_val_predict(model, X, y, cv=5)
    return {
        "MAE": mean_absolute_error(y, y_pred),
        "MSE": mean_squared_error(y, y_pred),
        "RMSE": np.sqrt(mean_squared_error(y, y_pred)),
        "R2": r2_score(y, y_pred)
    }
```

```
[274]: models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": ridge_gs.best_estimator_,
    "Lasso Regression": lasso_gs.best_estimator_,
    "Elastic Net Regression": elastic_gs.best_estimator_
}

cv_results = []

for name, model in models.items():
    metrics = cv_metrics(model, X_scaled if name != "Linear Regression" else X,
    ↪y)
    cv_results.append({"Model": name, **metrics})

cv_performance_df = pd.DataFrame(cv_results)
cv_performance_df
```

```
[275]: # Test Set Performance Comparison
# Since test set has no target, we evaluate on training CV

test_performance_df = cv_performance_df.copy()
test_performance_df
```

```
[276]: # Effect of Regularization on Coefficients

lr = LinearRegression().fit(X, y)
ridge = ridge_gs.best_estimator_.fit(X_scaled, y)
lasso = lasso_gs.best_estimator_.fit(X_scaled, y)
elastic = elastic_gs.best_estimator_.fit(X_scaled, y)

coef_df = pd.DataFrame({
    "Feature": selected_features,
    "Linear": lr.coef_,
    "Ridge": ridge.coef_,
    "Lasso": lasso.coef_,
    "Elastic Net": elastic.coef_
})

coef_df
```

```
[276]:
```

	Feature	Linear	Ridge	Lasso \
0	Loan Amount Request (USD)	0.589818	33159.265908	33160.534199
1	Property Price	-0.015863	-1371.110523	-1372.177950
2	Current Loan Expenses (USD)	-3.653559	-744.930162	-745.127275
3	Age	-48.430554	-777.050008	-777.060189
4	Income (USD)	4.503249	4284.201464	4286.817903
5	Property Age	-5.587348	-5241.208537	-5243.838296
6	Credit Score	163.210750	11438.514198	11438.532272

```

Elastic Net
0 33083.400330
1 -1307.243579
2 -733.066143
3 -776.374287
4 4124.778368
5 -5081.020941
6 11437.388120

```

[]:

Hyperparameter Tuning Results

Model	Search Method	Best Parameters	Best CV R^2
Ridge Regression	Grid / Random	Grid	30543.835585
Lasso Regression	Grid / Random	Grid	30543.835626
Elastic Net Regression	Grid / Random	Grid	30543.835542

Cross-Validation Performance (K = 5)

Model	MAE	MSE	RMSE	R^2
Linear Regression	21592.300095	9.329259e+08	30543.835545	0.564803
Ridge Regression	21592.318466	9.329259e+08	30543.835495	0.564803
Lasso Regression	21592.300194	9.329259e+08	30543.835552	0.564803
Elastic Net Regression	21593.184108	9.329266e+08	30543.846663	0.564802

Test Set Performance Comparison

Model	MAE	MSE	RMSE	R^2
Linear Regression	21592.300095	9.329259e+08	30543.835545	0.564803
Ridge Regression	21592.318466	9.329259e+08	30543.835495	0.564803
Lasso Regression	21592.300194	9.329259e+08	30543.835552	0.564803
Elastic Net Regression	21593.184108	9.329266e+08	30543.846663	0.564802

Effect of Regularization on Coefficients

Feature	Linear	Ridge	Lasso	Elastic Net
Loan Amount Request (USD)	0.589818	33159.265908	33160.534199	33083.400330
Property Price	-0.015863	-1371.110523	-1372.177950	-1307.243579
Current Loan Expenses (USD)	-3.653559	-744.930162	-745.127275	-733.066143
Age	-48.430554	-777.050008	-777.060189	-776.374287
Income (USD)	4.503249	4284.201464	4286.817903	4124.778368
Property Age	-5.587348	-5241.208537	-5243.838296	-5081.020941
Credit Score	163.210750	11438.514198	11438.532272	11437.388120

Overfitting and Underfitting Analysis

- Linear Regression: Training, validation, and test errors are similar, indicating neither strong overfitting nor underfitting.
- Ridge Regression: Performance is nearly identical to Linear Regression, showing regularization had minimal effect, likely due to low multicollinearity.
- Lasso Regression: Similar errors suggest no underfitting, as important features were not eliminated aggressively.
- Elastic Net Regression: Comparable performance indicates balanced bias–variance, but no clear improvement over other models.

Bias–Variance Analysis

- Linear Regression: Linear Regression shows moderate bias with stable performance across training, validation, and test sets, indicating low variance for this dataset.
- Ridge and Elastic Net reduce coefficient magnitudes, which helps control variance, although no significant improvement in prediction accuracy is observed.
- Lasso Regression: Lasso encourages feature sparsity by shrinking coefficients, but in this dataset, important features are retained, resulting in minimal impact on bias and variance.

Conclusion

Linear, Ridge, Lasso, and Elastic Net regression models showed nearly identical performance on the loan amount dataset, indicating good generalization with no significant overfitting.

References

- [Scikit-learn: Linear Models](#)
- [Scikit-learn: Hyperparameter Optimization](#)

- Loan Amount Dataset