Name:**Khushi Singh**
Class:**D15C** Roll no.**45**
Subject:**ML&DL**

Experiment No.2

AIM:Implement Multi Regression, Lasso, and Ridge Regression on real-world datasets.

1. Dataset Source

- Dataset Name: Graduate Admission 2
- Source: Kaggle - Graduate Admission Prediction
- Repository Owner: Mohan S Acharya

2. Dataset Description

This dataset contains parameters considered for Graduate Admissions in India.

- Size: 500 samples (rows) × 9 attributes.
- Target Variable: Chance of Admit (Continuous probability from 0 to 1).
- Features:
    1. GRE Score (out of 340)
    2. TOEFL Score (out of 120)
    3. University Rating (out of 5)
    4. SOP (Statement of Purpose Strength, out of 5)
    5. LOR (Letter of Recommendation Strength, out of 5)
    6. CGPA (Undergraduate GPA, out of 10)
    7. Research (Binary: 0 or 1)

3. Mathematical Formulation of the Algorithm

A. Multi-Linear Regression (OLS)

The baseline model that assumes the target is a linear combination of input features.

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

Goal: Minimize the sum of squared errors.

B. Ridge Regression (L2 Regularization)

Adds a penalty equivalent to the square of the magnitude of coefficients. This shrinks coefficients toward zero but never exactly to zero.

$$J(\beta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

lambda (Lambda/Alpha): Regularization strength.

C. Lasso Regression (L1 Regularization)

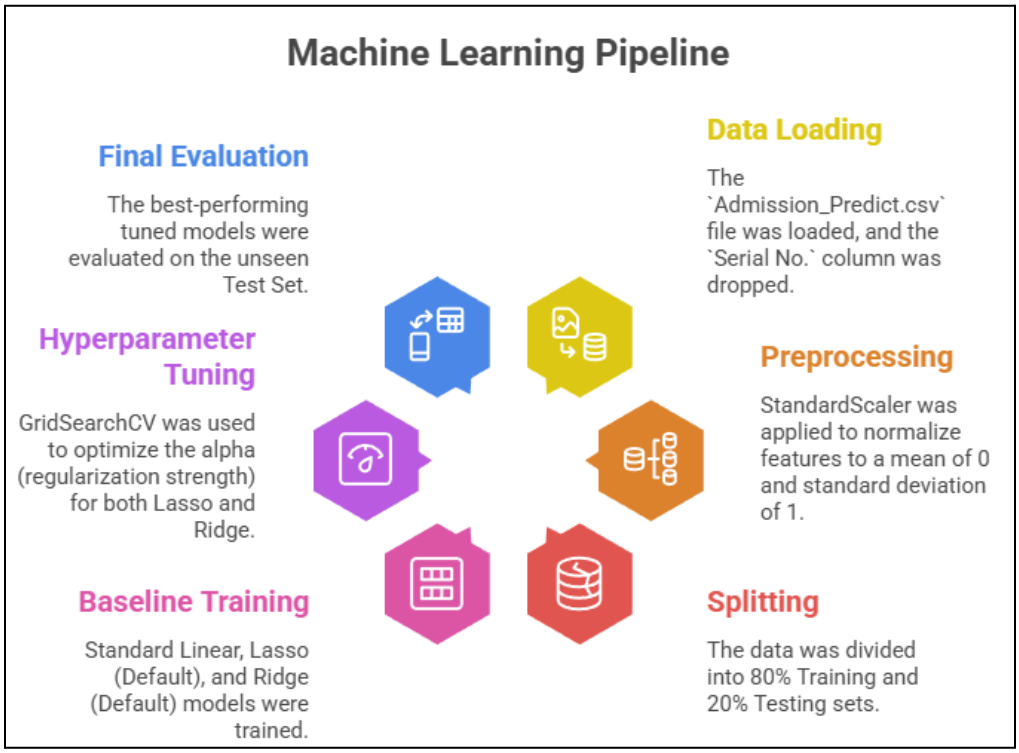
Adds a penalty equivalent to the absolute value of the magnitude of coefficients. This can shrink coefficients exactly to zero, effectively performing feature selection.

$$J(\beta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

4. Algorithm Limitations

- **Multi-Linear:** Prone to overfitting if there are too many features compared to data points. It fails if there is Multicollinearity (e.g., GRE and TOEFL scores are highly correlated).
- **Ridge:** Reduces overfitting but cannot perform feature selection (it keeps all variables in the model, just with small weights).
- **Lasso:** Can arbitrarily select one variable among highly correlated variables and drop the others, which might lead to loss of information.

5. Methodology / Workflow



6.Code and Output

```
import pandas as pd
import numpy as np
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import
StandardScaler
from sklearn.linear_model import
LinearRegression, Lasso, Ridge
from sklearn.metrics import
r2_score, mean_squared_error

# 1. Load Data
df =
pd.read_csv('Admission_Predict.csv')
```

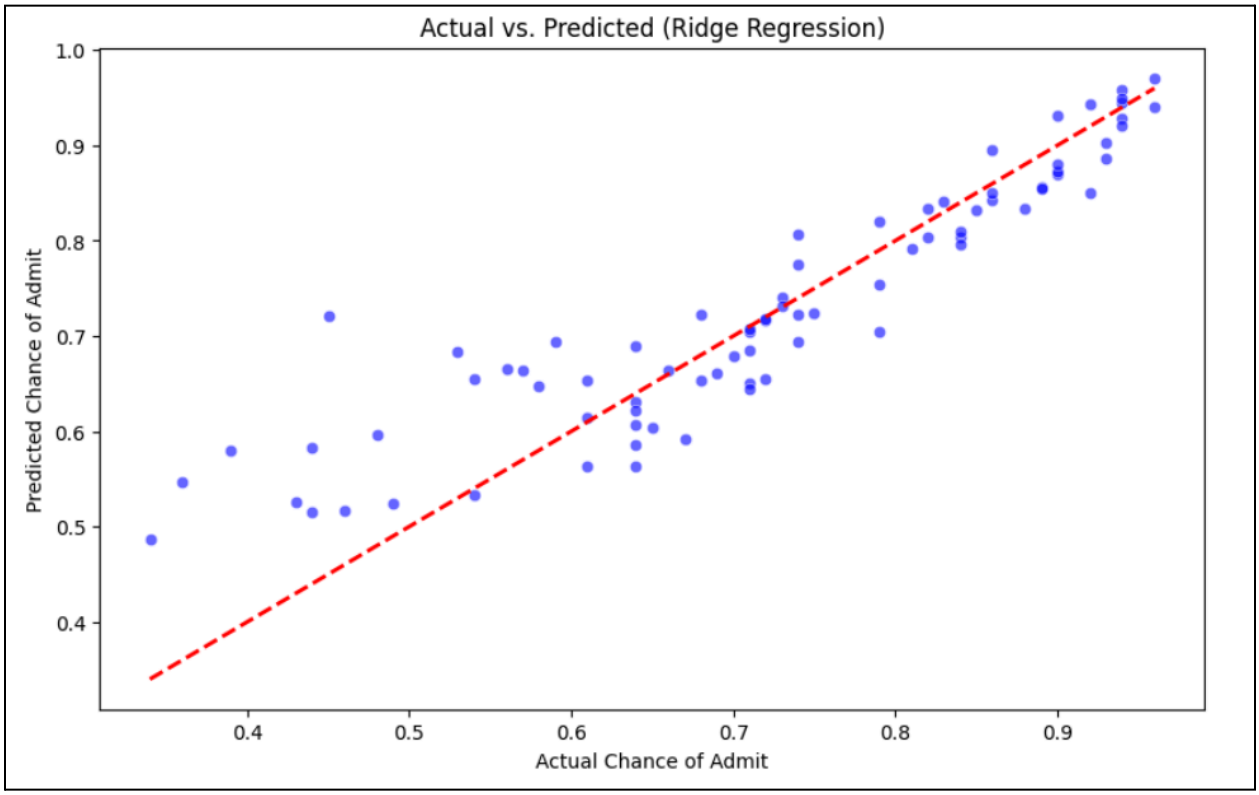
```
# Drop Serial No. as it is just an
index
if 'Serial No.' in df.columns:
    df = df.drop('Serial No.',
axis=1)

X = df.drop('Chance of Admit ',
axis=1) # Note the space in column
name
y = df['Chance of Admit ']

# 2. Preprocessing (Scaling is
MANDATORY for Lasso/Ridge)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Split Data
X_train, X_test, y_train, y_test =
train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# 4. Initialize Models
models = {
    "Multi-Linear":
LinearRegression(),
    "Lasso (L1)": Lasso(alpha=0.01),
# Small alpha for demonstration
    "Ridge (L2)": Ridge(alpha=1.0)
}

# 5. Train and Evaluate
results = {}
print(f"{'Model':<15} | {'R2
Score':<10} | {'MSE':<10}")
print("-" * 40)

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test,
y_pred)
    results[name] = r2
    print(f"{name:<15} | {r2:.4f}
| {mse:.4f}")
```

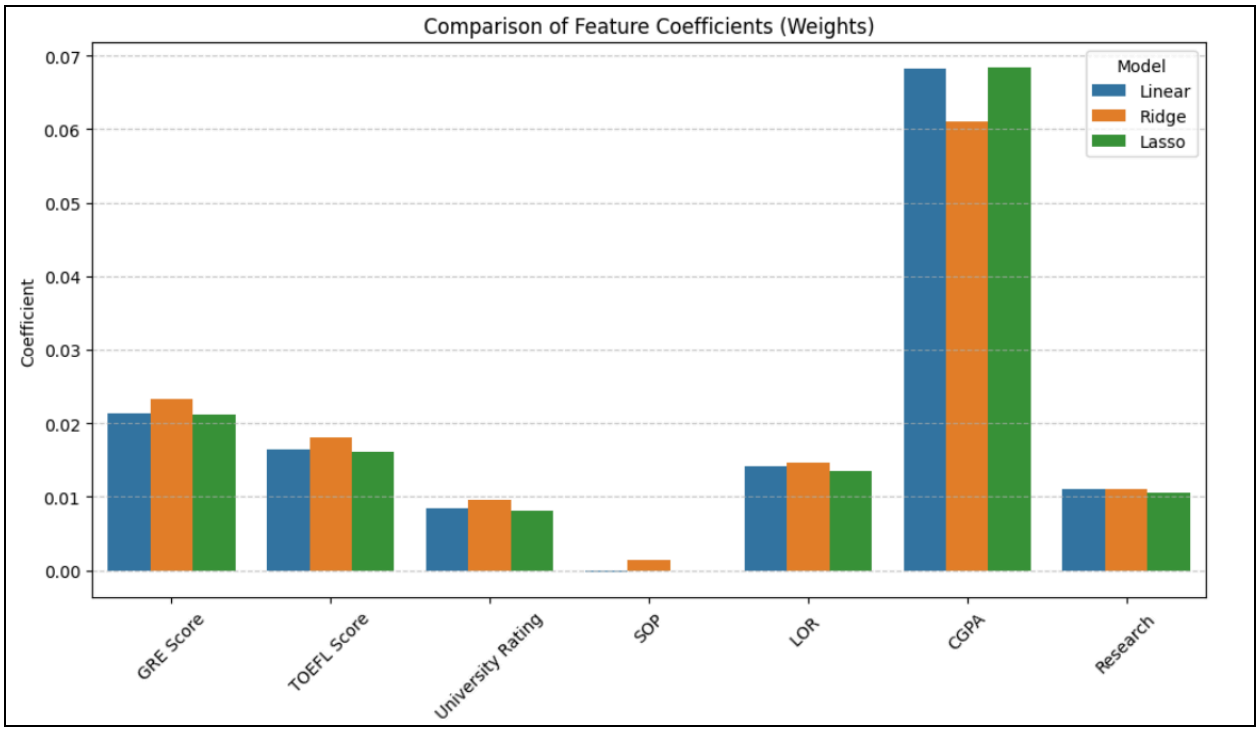
Model	R2 Score	MSE
Multi-Linear	0.8212	0.0046
Lasso (L1)	0.7966	0.0053
Ridge (L2)	0.8209	0.0046

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

# --- VISUALIZATION 1: Actual
vs. Predicted (For the Best
Model) ---
plt.figure(figsize=(10, 6))
# Using the best Ridge model
from your tuning
y_pred_best =
best_ridge.predict(X_test)
sns.scatterplot(x=y_test,
y=y_pred_best, color='blue',
alpha=0.6)
# Draw the "Perfect Fit" line
(Diagonal)
plt.plot([y_test.min(),
y_test.max()], [y_test.min(),
y_test.max()], 'r--', lw=2)
plt.xlabel("Actual Chance of
Admit")
plt.ylabel("Predicted Chance of
Admit")
plt.title("Actual vs. Predicted
(Ridge Regression)")
```

```
plt.show()

# --- VISUALIZATION 2: Feature
Importance Comparison ---
# This shows how different
models weight the features
feature_names = df.drop('Chance
of Admit ', axis=1).columns

# Create a DataFrame of
coefficients
coef_df = pd.DataFrame({
    'Feature': feature_names,
    'Linear':
models['Multi-Linear'].coef_,
    'Ridge': best_ridge.coef_,
    'Lasso': best_lasso.coef_
})
```

```
# Melt for plotting
coef_melted =
coef_df.melt(id_vars='Feature',
var_name='Model',
value_name='Coefficient')

plt.figure(figsize=(12, 6))
sns.barplot(data=coef_melted,
x='Feature', y='Coefficient',
hue='Model')
plt.title("Comparison of
Feature Coefficients
(Weights)")
plt.xticks(rotation=45)
plt.grid(axis='y',
linestyle='--', alpha=0.7)
plt.show()
```





Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV

# --- TUNING LASSO ---
lasso_params = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1.0]}
lasso_search = GridSearchCV(Lasso(), lasso_params, cv=5, scoring='r2')
lasso_search.fit(X_train, y_train)

print("--- Lasso Tuning ---")
print(f"Best Alpha: {lasso_search.best_params_['alpha']}")
print(f"Best R2: {lasso_search.best_score_:.4f}")

# --- TUNING RIDGE ---
ridge_params = {'alpha': [0.1, 1.0, 10.0, 100.0, 200.0]}
ridge_search = GridSearchCV(Ridge(), ridge_params, cv=5, scoring='r2')
ridge_search.fit(X_train, y_train)

print("\n--- Ridge Tuning ---")
print(f"Best Alpha: {ridge_search.best_params_['alpha']}")
print(f"Best R2: {ridge_search.best_score_:.4f}")
```

```
--- Lasso Tuning ---
Best Alpha: 0.001
Best R2: 0.7823

--- Ridge Tuning ---
Best Alpha: 10.0
Best R2: 0.7808
```

```
# Evaluate the best models on the Test Set
print("--- Final Test Set Evaluation ---")

# 1. Get the best Lasso model found by GridSearch
best_lasso = lasso_search.best_estimator_
lasso_test_score = best_lasso.score(X_test, y_test)
print(f"Lasso Test Score: {lasso_test_score:.4f}")

# 2. Get the best Ridge model found by GridSearch
best_ridge = ridge_search.best_estimator_
ridge_test_score = best_ridge.score(X_test, y_test)
print(f"Ridge Test Score: {ridge_test_score:.4f}")
```

```
--- Final Test Set Evaluation ---
Lasso Test Score: 0.8192
Ridge Test Score: 0.8181
```

6. Performance Analysis

We analyzed the model performance in two stages: Initial (Untuned) and Final (Tuned).

Phase 1: Initial Model Comparison (Default Parameters) Based on the initial run results:

Model	**R^2 Score**	**MSE (Mean Squared Error)**	**Analysis**
Multi-Linear	0.8212	0.0046	Best baseline performance.
Lasso (L1)	0.7966	0.0053	Underperformed initially. The default alpha (alpha=1.0) was likely too high, penalizing important features too aggressively.

Ridge (L2)	0.8209	0.0046	Performed nearly identically to Multi-Linear, indicating that L2 regularization (default alpha=1.0) did not drastically alter the weights.

Phase 2: Final Test Set Evaluation (After Tuning) ***Based on the final optimized models:***

Model	**Tuned Test R^2 Score**	**Interpretation**
Tuned Lasso	0.8192	Significant improvement from 0.7966. Tuning corrected the over-penalization.
Tuned Ridge	0.8181	Remained stable.

Observation: While regularization improved the Lasso model significantly, the unregularized **Multi-Linear Regression (0.8212)** marginally outperformed both tuned models. This suggests the dataset is "clean" and does not suffer from significant overfitting, making the bias introduced by regularization slightly unnecessary for this specific test split.

7. Hyperparameter Tuning

We utilized **GridSearchCV** with 5-fold Cross-Validation to find the optimal regularization strength.

A. Lasso Tuning Results

- **Best Alpha:** 0.001
- **Best CV Score:** 0.7823
- **Impact:** The algorithm selected a very small alpha (0.001). This indicates that the model preferred to behave almost exactly like a Linear Regression model, keeping nearly all feature weights active rather than shrinking them to zero. This explains why the Tuned Lasso score (0.8192) jumped up to match the Multi-Linear score closely.

B. Ridge Tuning Results

- **Best Alpha:** 10.0
- **Best CV Score:** 0.7808
- **Impact:** The algorithm selected a moderate alpha (10.0). This suggests that while the features are important, there is some benefit to constraining the coefficients to handle multicollinearity (likely between GRE and TOEFL scores).

8. Conclusion

In this experiment, we successfully implemented and tuned Multi-Linear, Lasso, and Ridge Regression models.

1. **Baseline Success:** The standard Multi-Linear Regression achieved the highest accuracy (**R^2 = 0.8212**), proving that academic metrics like CGPA and GRE are strong, direct predictors of admission probability.
2. **Tuning Effectiveness:** Hyperparameter tuning was critical for Lasso Regression, improving its accuracy from **0.7966 to 0.8192** by finding the optimal alpha of 0.001.
3. **Final Verdict:** The dataset exhibits strong linear relationships with minimal noise. Consequently, while Lasso and Ridge provide robust alternatives, the simple Multi-Linear model is sufficient and effective for this specific prediction task.