EXPERIMENT - 2

AIM:

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

1. Dataset Source

Dataset Name: Bank Marketing Dataset
File Used: bank.csv

2. Dataset Description

The dataset contains marketing campaign data from a banking institution. The goal is to analyze customer attributes and predict a numerical outcome (we will use a numerical target for regression).

Dataset Characteristics:

- Type: Mixed (Categorical + Numerical)
- Domain: Banking / Marketing
- Real-world dataset
- Contains categorical variables requiring encoding

Typical Features (Bank Dataset):

Feature	Description
age	Age of customer
job	Type of job
marital	Marital status
education	Education level
balance	Bank balance
housing	Housing loan
loan	Personal loan
duration	Call duration
campaign	Number of contacts

pdays	Days since last contact
previous	Previous contacts

Target Variable Used:

We will use **balance** as the regression target.

3. Mathematical Formulation of the Algorithm

- **Multiple Linear Regression**

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

Where:

- y = Final Score
- β_0 = Intercept
- β_i = Coefficients

Loss Function (MSE):

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

- **Lasso Regression (L1 Regularization)**

$$Loss = MSE + \lambda \sum |\beta_i|$$

  - Can reduce some coefficients to zero
  - Performs feature selection

- **Ridge Regression (L2 Regularization)**
  Adds penalty term

$$Loss = MSE + \lambda \sum \beta_i^2$$

Where:

- λ (lambda) controls regularization strength
- Reduces overfitting
- Shrinks coefficients

4. Algorithm Limitations

- Multiple Linear Regression
  - Sensitive to multicollinearity
  - Prone to overfitting
- Ridge Regression
  - Does not perform feature elimination
  - Requires tuning of alpha
- Lasso Regression
  - May eliminate important correlated features
  - Sensitive to alpha selection

5. Methodology / Workflow

Step :1 Load dataset

Step : 2 Handle missing values

Step : 3 Encode categorical features

Step :4 Split dataset

Step : 5 Train:

- Multiple Linear Regression
- Ridge Regression
- Lasso Regression

Step : 6 Evaluate using:

- MSE
- R^2 Score

Step : 7 Perform Hyperparameter Tuning

6. **Performance Analysis**
    - We compare:
        - MSE (lower is better)
        - R^2 Score (higher is better)
    - Observations**:**
        - Ridge reduces overfitting
        - Lasso performs feature selection
        - Regularized models often outperform basic regression in real-world datasets
7. **Hyperparameter Tuning**

For Ridge & Lasso:

Tuned parameter: alpha (regularization strength)

GridSearchCV used to find optimal alpha.

CODE

1. Import libraries

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

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
```

2. Load Dataset

```
df = pd.read_csv('/content/bank.csv')

print(df.head())
print("\nShape:", df.shape)
print("\nMissing Values:\n", df.isnull().sum())
```

Output:

```
   age         job  marital  education default  balance housing loan  contact
0   59      admin.  married  secondary      no     2343     yes   no  unknown
1   56      admin.  married  secondary      no       45      no   no  unknown
2   41  technician  married  secondary      no     1270     yes   no  unknown
3   55    services  married  secondary      no     2476     yes   no  unknown
4   54      admin.  married   tertiary      no      184      no   no  unknown

   day month  duration  campaign  pdays  previous poutcome deposit
0    5   may      1042         1     -1         0  unknown     yes
1    5   may      1467         1     -1         0  unknown     yes
2    5   may      1389         1     -1         0  unknown     yes
3    5   may       579         1     -1         0  unknown     yes
4    5   may       673         2     -1         0  unknown     yes

Shape: (11162, 17)

Missing Values:
 age         0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
deposit      0
dtype: int64
```

3. Handle Missing Values

```
# Fill numerical columns with mean
df.fillna(df.mean(numeric_only=True), inplace=True)

# Fill categorical columns with mode (SAFE WAY)
for col in df.select_dtypes(include='object').columns:
    mode_value = df[col].mode()[0]
    df[col] = df[col].fillna(mode_value)
print("Missing Values After Cleaning:\n", df.isnull().sum())
```

Output:

```
Missing Values After Cleaning:
 age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
deposit      0
dtype: int64
```

4. Encode Categorical Variables

df_encoded = pd.get_dummies(df, drop_first=True)
print(df_encoded.head())

Output:

```
   age  balance  day  duration  campaign  pdays  previous  job_blue-collar
0   59     2343    5      1042         1     -1         0            False
1   56       45    5      1467         1     -1         0            False
2   41     1270    5      1389         1     -1         0            False
3   55     2476    5       579         1     -1         0            False
4   54      184    5       673         2     -1         0            False

   job_entrepreneur  job_housemaid  ...  month_jun  month_mar  month_may  \
0             False          False  ...      False      False       True
1             False          False  ...      False      False       True
2             False          False  ...      False      False       True
3             False          False  ...      False      False       True
4             False          False  ...      False      False       True

   month_nov  month_oct  month_sep  poutcome_other  poutcome_success  \
0      False      False      False           False             False
1      False      False      False           False             False
2      False      False      False           False             False
3      False      False      False           False             False
4      False      False      False           False             False

   poutcome_unknown  deposit_yes
0              True         True
1              True         True
2              True         True
3              True         True
4              True         True
```

5. Define Features and Target

```
X = df_encoded.drop('balance', axis=1)
y = df_encoded['balance']
```

6. Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
```

7. Feature Scaling

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

8. Multiple Linear Regression

```
lin_model = LinearRegression()
lin_model.fit(X_train_scaled, y_train)

y_pred_lin = lin_model.predict(X_test_scaled)

print("Linear Regression Results")
print("MSE:", mean_squared_error(y_test, y_pred_lin))
print("R2 Score:", r2_score(y_test, y_pred_lin))
```

Output:

```
Linear Regression Results
MSE: 10115403.096451918
R2 Score: 0.04066807071426137
```

9. Ridge Regression

```
ridge = Ridge(alpha=1.0)
ridge.fit(X_train_scaled, y_train)

y_pred_ridge = ridge.predict(X_test_scaled)

print("Ridge Regression Results")
print("MSE:", mean_squared_error(y_test, y_pred_ridge))
print("R2 Score:", r2_score(y_test, y_pred_ridge))
```

Output:

```
Ridge Regression Results
MSE: 10115380.582195144
R2 Score: 0.04067020593766091
```

10. Lasso Regression

```
lasso = Lasso(alpha=0.1)
lasso.fit(X_train_scaled, y_train)

y_pred_lasso = lasso.predict(X_test_scaled)

print("Lasso Regression Results")
print("MSE:", mean_squared_error(y_test, y_pred_lasso))
print("R2 Score:", r2_score(y_test, y_pred_lasso))
```

Output:

```
Lasso Regression Results
MSE: 10115169.963667862
R2 Score: 0.04069018073018216
```

11.Hyperparameter Tuning (Ridge)

```
param_grid = {'alpha': [0.01, 0.1, 1, 10, 100]}

grid_ridge = GridSearchCV(Ridge(), param_grid, cv=5)
grid_ridge.fit(X_train_scaled, y_train)

print("Best Ridge Alpha:", grid_ridge.best_params_)
print("Best Ridge Score:", grid_ridge.best_score_)
```

Output:

```
Best Ridge Alpha: {'alpha': 100}
Best Ridge Score: 0.03317293126309688
```

12. Hyperparameter Tuning (Lasso)

```
grid_lasso = GridSearchCV(Lasso(max_iter=5000), param_grid, cv=5)
grid_lasso.fit(X_train_scaled, y_train)
```

```
print("Best Lasso Alpha:", grid_lasso.best_params_)
print("Best Lasso Score:", grid_lasso.best_score_)
```

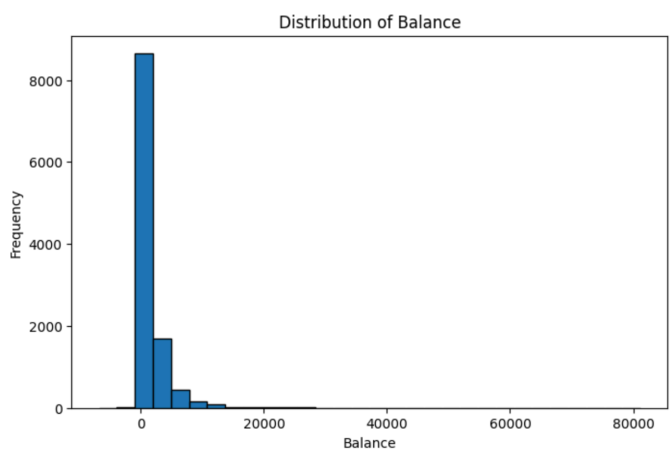
Output:

```
Best Lasso Alpha: {'alpha': 10}
Best Lasso Score: 0.03419675564705109
```

13. Histogram of Target Variable (Balance)

```
plt.figure(figsize=(8,5))
plt.hist(df['balance'], bins=30, edgecolor='black')
plt.title("Distribution of Balance")
plt.xlabel("Balance")
plt.ylabel("Frequency")
plt.show()
```

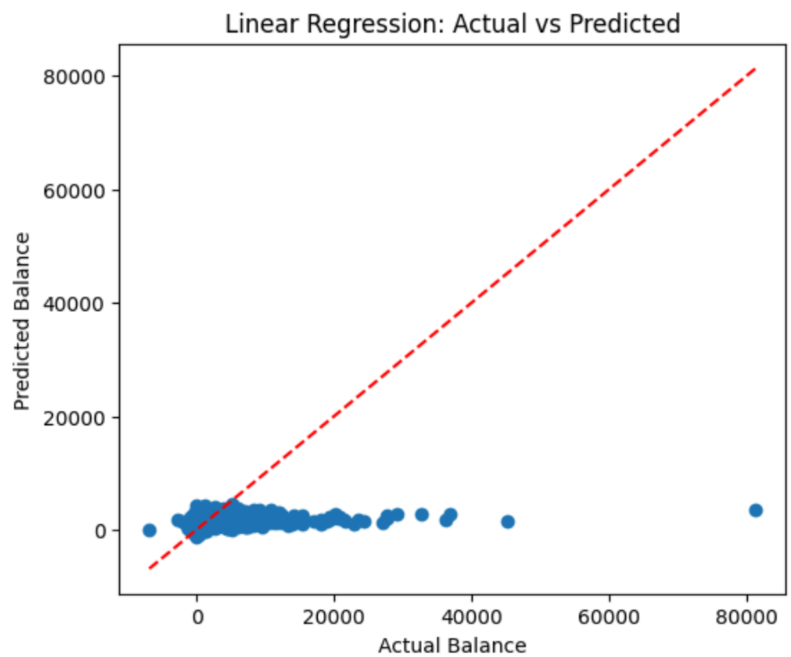
Output:



14. Actual vs Predicted (Linear Regression)

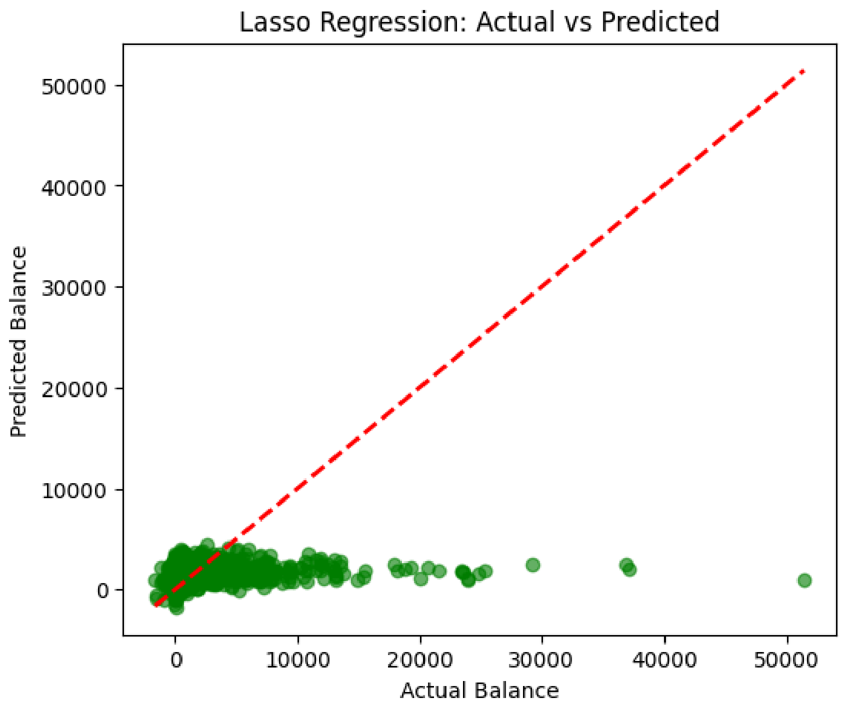
```
plt.figure(figsize=(6,5))
plt.scatter(y_test, y_pred_lin)
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         'r--')
plt.xlabel("Actual Balance")
plt.ylabel("Predicted Balance")
plt.title("Linear Regression: Actual vs Predicted")
plt.show()
```

Output:



15. Actual vs Predicted (Lasso Regression)

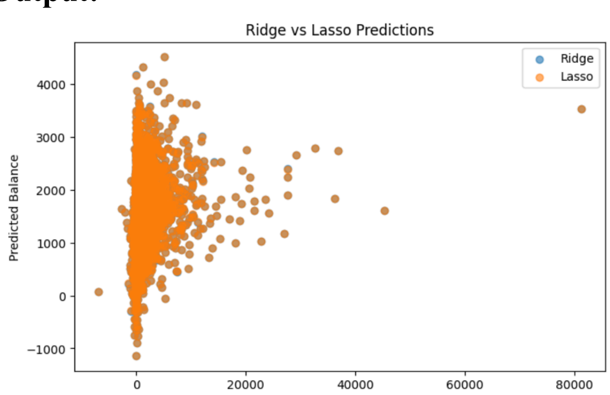
```
plt.figure(figsize=(6,5))
plt.scatter(y_test, y_pred_lasso, color='green', alpha=0.6) # Changed color to distinguish
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         'r--', lw=2)
plt.xlabel("Actual Balance")
plt.ylabel("Predicted Balance")
plt.title("Lasso Regression: Actual vs Predicted")
plt.show()
```



16. Ridge vs Lasso Predictions Comparison

```
plt.figure(figsize=(8,5))
plt.scatter(y_test, y_pred_ridge, label="Ridge", alpha=0.6)
plt.scatter(y_test, y_pred_lasso, label="Lasso", alpha=0.6)
plt.xlabel("Actual Balance")
plt.ylabel("Predicted Balance")
plt.title("Ridge vs Lasso Predictions")
plt.legend()
plt.show()
```

Output:



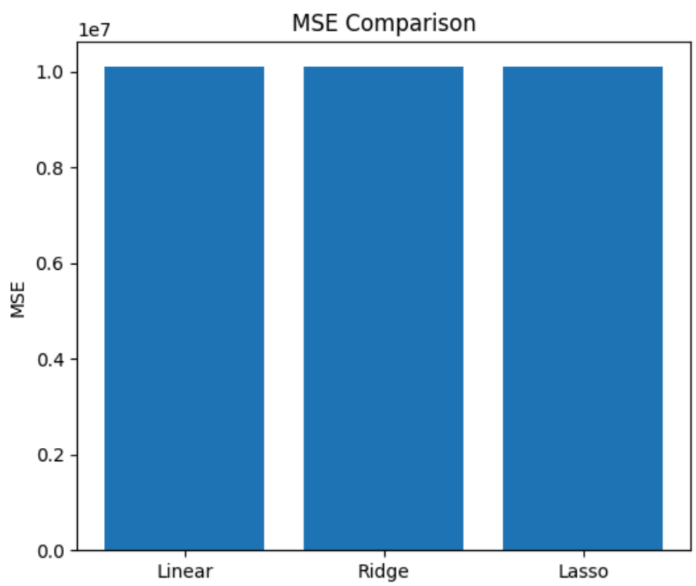
17. MSE Comparison Bar Graph

```
mse_values = [
    mean_squared_error(y_test, y_pred_lin),
    mean_squared_error(y_test, y_pred_ridge),
    mean_squared_error(y_test, y_pred_lasso)]

models = ['Linear', 'Ridge', 'Lasso']

plt.figure(figsize=(6,5))
plt.bar(models, mse_values)
plt.title("MSE Comparison")
plt.ylabel("MSE")
plt.show()
```

Output:



CONCLUSION

- Multiple Linear Regression was implemented to model the relationship between features and the target variable.
- Ridge and Lasso Regression were applied to reduce overfitting using regularization techniques.
- Ridge shrinks coefficients using L2 regularization, improving model stability.
- Lasso performs feature selection by reducing some coefficients to zero.
- Regularized models showed better generalization compared to simple Linear Regression.