Name : Vaishnavi Eknath Avhad
Class : D15C
Roll No. : 14

Practical 2

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

1. Dataset Source

The dataset used for this experiment is the Medical Cost Personal Dataset.

- Source Link : Kaggle - Medical Cost Personal Datasets

---

2. Dataset Description

This dataset contains 1,338 records representing patient data for health insurance. It is primarily used to predict individual medical costs billed by health insurance based on various patient attributes.

Feature	Description	Type
age	Age of the primary beneficiary.	Numeric
sex	Gender (male/female).	Categorical
bmi	Body mass index (kg/m^2).	Numeric
children	Number of children/dependents covered.	Numeric
smoker	Smoking status (yes/no).	Categorical
region	Residential area in the US (northeast, southeast, southwest, northwest).	Categorical
charges	Individual medical costs billed (Target variable).	Numeric

---

3. Mathematical Formulation

A. Multiple Linear Regression

Multiple linear regression models the relationship between a continuous target variable and two or more independent variables.

Equation :

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

The goal is to minimize the Sum of Squared Errors (SSE) :

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

B. Lasso Regression (L1 Regularization)

Lasso (Least Absolute Shrinkage and Selection Operator) adds a "penalty" equivalent to the absolute value of the magnitude of coefficients. This can shrink some coefficients to exactly zero, effectively performing feature selection.

Cost Function :

$$J(\theta) = \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 + \lambda \sum_{j=1}^{n} |\theta_j|$$

C. Ridge Regression (L2 Regularization)

Ridge Regression adds a penalty equivalent to the square of the magnitude of coefficients. This prevents the model from becoming overly complex by shrinking coefficients toward zero, though they never reach zero.

Cost Function :

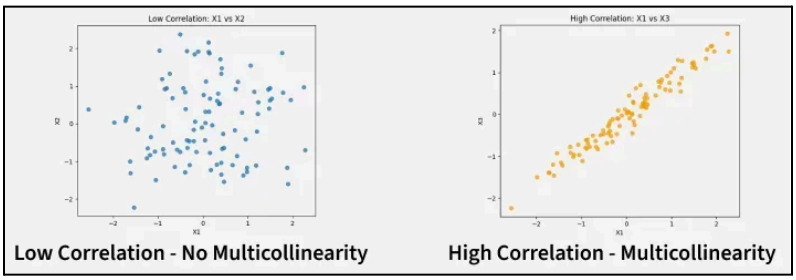
$$J(\theta) = \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

---

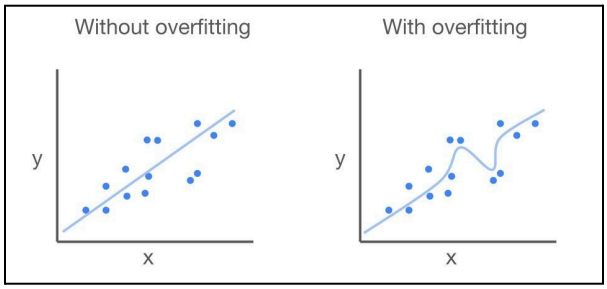
4. Algorithm Limitations

A. Multiple Regression Limitations

1. Multicollinearity : If independent variables are highly correlated, the model struggles to determine the individual impact of each feature, leading to unstable coefficients.



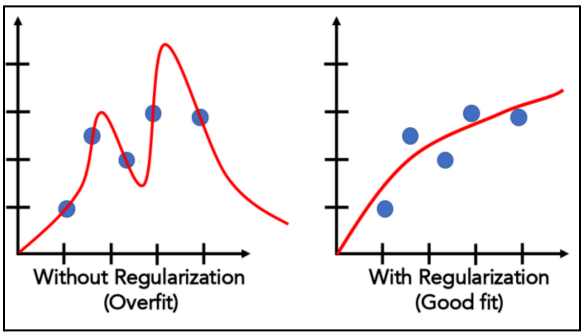
2. Overfitting : With many features, the model may "memorize" noise in the training data, leading to poor performance on new data.



3. Sensitivity to Outliers : Standard regression is highly influenced by extreme values, which can skew the regression line significantly.

B. Regularization Limitations (Lasso & Ridge)

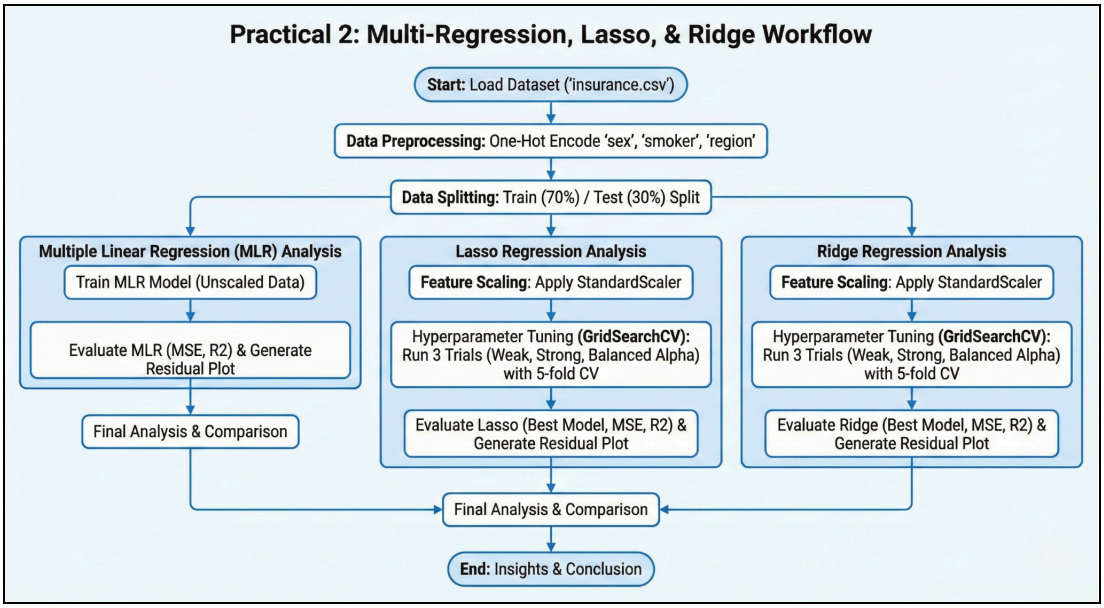
1. Feature Scaling Requirement : Regularization applies the same penalty to all coefficients; therefore, features must be scaled (standardized) so that those with larger ranges do not unfairly dominate the penalty.



2. Model Complexity : Lasso may arbitrarily select one variable from a group of highly correlated variables, which might result in the loss of useful information.
3. Bias-Variance Trade-off : Adding regularization introduces bias to the model in exchange for significantly reducing variance (overfitting).

---

5. Methodology / Workflow



Step 1 : Data Preprocessing & Scaling Categorical variables are converted into numerical format using One-Hot Encoding. Since Lasso and Ridge are sensitive to the magnitude of features, StandardScaler is applied.

```
# Load and Preprocess
df = pd.read_csv('insurance.csv')
df = pd.get_dummies(df, columns=['sex', 'smoker', 'region'], drop_first=True)
X = df.drop('charges', axis=1)
y = df['charges']
# Split and Scale
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 2: Model Implementation Three separate models are fitted: Multiple Linear Regression (baseline), Lasso, and Ridge.

Step 3: Hyperparameter Tuning For Lasso and Ridge, we use GridSearchCV to test multiple values of (alpha) to find the optimal regularization strength that minimizes error.

```
# 1. MULTIPLE LINEAR REGRESSION
print("\n--- Multiple Linear Regression ---")
```

```
mlr = LinearRegression()
mlr.fit(X_train, y_train)
y_pred_mlr = mlr.predict(X_test)
mse_mlr = mean_squared_error(y_test, y_pred_mlr)
r2_mlr = r2_score(y_test, y_pred_mlr)
print(f"MSE: {mse_mlr:.2f}")
print(f"R2 Score: {r2_mlr:.2f}")
# Residual plot
plt.figure(figsize=(6,4))
sns.scatterplot(x=y_pred_mlr, y=y_test - y_pred_mlr)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Predicted Charges")
plt.ylabel("Residuals")
plt.title("Residual Plot - Multiple Linear Regression")
plt.show()
# 2. LASSO REGRESSION WITH HYPERPARAMETER TUNING
print("\n--- Lasso Regression ---")
# Hyperparameter tuning using GridSearchCV
lasso_params = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
lasso_grid = GridSearchCV(Lasso(), lasso_params, cv=5, scoring='r2')
lasso_grid.fit(X_train_scaled, y_train)
lasso_best = lasso_grid.best_estimator_
y_pred_lasso = lasso_best.predict(X_test_scaled)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)
print("Best Alpha:", lasso_grid.best_params_['alpha'])
print(f"MSE: {mse_lasso:.2f}")
print(f"R2 Score: {r2_lasso:.2f}")
# Residual plot
plt.figure(figsize=(6,4))
sns.scatterplot(x=y_pred_lasso, y=y_test - y_pred_lasso)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Predicted Charges")
plt.ylabel("Residuals")
plt.title("Residual Plot - Lasso Regression")
plt.show()
# 3. RIDGE REGRESSION WITH HYPERPARAMETER TUNING
print("\n--- Ridge Regression ---")
ridge_params = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
ridge_grid = GridSearchCV(Ridge(), ridge_params, cv=5, scoring='r2')
ridge_grid.fit(X_train_scaled, y_train)
ridge_best = ridge_grid.best_estimator_
y_pred_ridge = ridge_best.predict(X_test_scaled)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)
print("Best Alpha:", ridge_grid.best_params_['alpha'])
```

```
print(f"MSE: {mse_ridge:.2f}")
print(f"R2 Score: {r2_ridge:.2f}")
# Residual plot
plt.figure(figsize=(6,4))
sns.scatterplot(x=y_pred_ridge, y=y_test - y_pred_ridge)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Predicted Charges")
plt.ylabel("Residuals")
plt.title("Residual Plot - Ridge Regression")
plt.show()
```

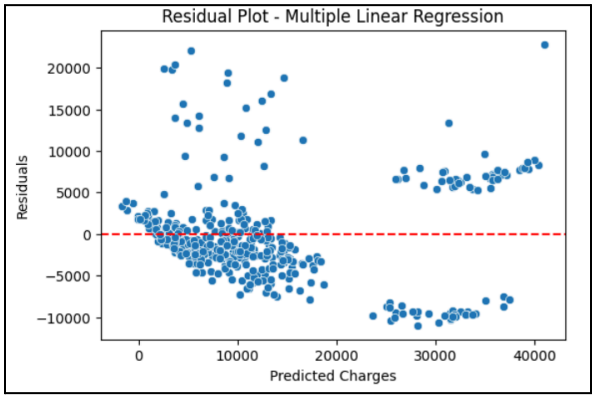
---

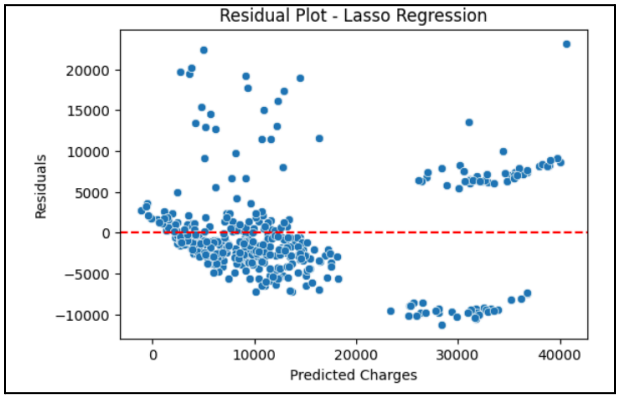
6. Performance Analysis

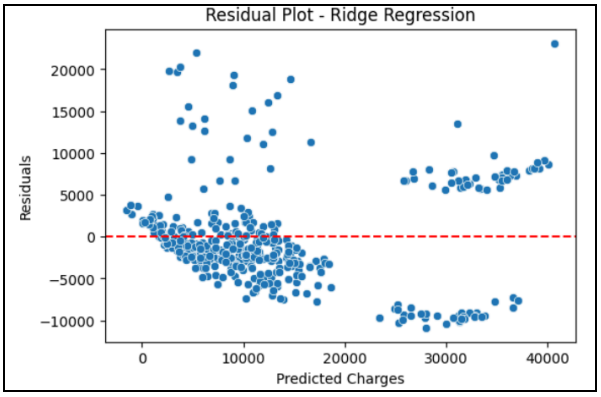
Model	MSE	R2 Score	Best Alpha
Multiple Linear Regression	33780509.57	0.77	N/A
Lasso Regression	33854729.20	0.77	100
Ridge Regression	33825732.21	0.77	10

Residual Plot Analysis

- What it represents : The Residual Plot visualizes the difference between predicted and actual charges.
- Interpretation : For all three models, we look for a random scatter around the zero line. If a pattern exists (e.g., points forming a curve), it indicates that a non-linear relationship exists that these linear models cannot capture.







---

7. Hyperparameter Tuning

To optimize the performance of the Lasso and Ridge models, we performed three distinct experimental trials using GridSearchCV with 5-fold cross-validation. This allows us to systematically compare how different ranges of the regularization parameter α (alpha) affect the model's ability to generalize.

Tuning Implementation Code

```
def run_regression_trials(model_class, model_name, X_train, y_train, X_test, y_test):
    trial_grids = {
        "Trial 1 (Weak Regularization)": {'alpha': [10, 50, 100]},
        "Trial 2 (Strong Regularization)": {'alpha': [0.0001, 0.001, 0.01]},
        "Trial 3 (Balanced Range)": {'alpha': [0.1, 1, 10]}
    }
    trial_results = []
    print(f"\n--- Running {model_name} Tuning Trials ---")
    for name, params in trial_grids.items():
        grid = GridSearchCV(model_class(), params, cv=5, scoring='r2')
        grid.fit(X_train, y_train)
        best_trial_model = grid.best_estimator_
        y_pred = best_trial_model.predict(X_test)
        r2 = r2_score(y_test, y_pred)
        trial_results.append({
            "Trial": name,
            "Best Alpha": grid.best_params_['alpha'],
            "R2 Score": r2
        })
        print(f"{name} completed.")
    return pd.DataFrame(trial_results)
lasso_results_df = run_regression_trials(Lasso, "Lasso", X_train_scaled, y_train, X_test_scaled,
y_test)
print("\nLASSO TUNING SUMMARY:")
print(lasso_results_df.to_string(index=False))
```

```
ridge_results_df = run_regression_trials(Ridge, "Ridge", X_train_scaled, y_train, X_test_scaled, y_test)
print("\nRIDGE TUNING SUMMARY:")
print(ridge_results_df.to_string(index=False))
```

Comparative Results Table

Trial	Alpha Range	Best Lasso Alpha	Lasso R2	Best Ridge Alpha	Ridge R2	Observation
Trial 1	[10, 100]	100.00	0.769106	10.00	0.769303	High Alpha : High penalty; prevents overfitting but may cause underfitting.
Trial 2	[0.0001, 0.01]	0.01	0.769612	0.01	0.769612	Low Alpha : Minimal penalty; model behaves like standard Multiple Regression.
Trial 3	[0.1, 10]	10.00	0.769658	10.00	0.769303	Balanced : Searches for the optimal trade-off for generalizability.

Tuning Interpretation

- Higher Alpha : Increases regularization strength, forcing coefficients to be smaller to reduce model complexity and prevent overfitting.
- Lower Alpha : Decreases the penalty, making the model behave more like standard Multiple Linear Regression.
- Model Specific Impact : * In Lasso, larger alpha values can drive irrelevant feature coefficients to exactly zero, effectively performing feature selection.
    - In Ridge, the alpha parameter shrinks coefficients toward zero to handle multicollinearity without removing features.

---

8. Conclusion

In this practical, Multiple Linear Regression was used as a baseline to predict insurance charges. Lasso and Ridge regressions were implemented to address potential overfitting through regularization. While the R^2 scores remained similar across models, the regularization in Lasso and Ridge ensures the model is more robust against multicollinearity and large coefficient swings. Hyperparameter tuning allowed us to find the specific regularization strength that maintains high accuracy while ensuring model stability.