Assignment Report – Machine Learning

Name: Kumar Naman

Student ID: 25340053

Course: Machine Learning

Date: 19/10/2025

Dataset ID: 19-19–19

I) Introduction

This study compares **Lasso** and **Ridge Regression**, two regularized extensions of linear regression designed to prevent overfitting and improve generalization. A synthetic dataset with two input variables was expanded into polynomial features (up to degree 5) to model nonlinear relationships. Lasso applies an **L1 penalty**, driving some coefficients to zero for feature selection, while Ridge uses an **L2 penalty**, shrinking all coefficients smoothly for stability. Models were trained over a range of C values (inverse of regularization strength) and evaluated using **5-fold cross-validation** to identify the optimal balance between bias and variance for accurate and generalizable predictions.

II) Data Visualization and Exploration

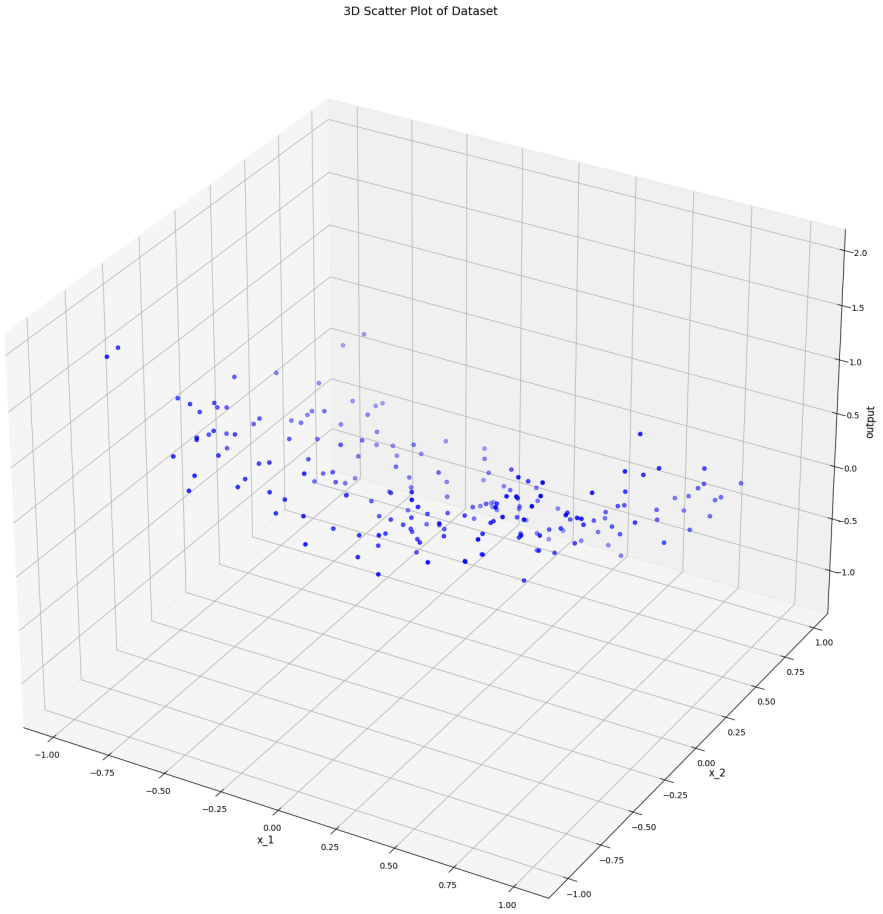
This section gives us a basic idea of the dataset provided. There is a 3D graph plot giving us a high level overview on the type of data and the relationship between the input dataset and the output data.

II.I) Dataset Identifier and Description

- The dataset belongs to the unique dataset ID 19-19–19 which was obtained from the first line of the file provided.
- The dataset contains 199 samples with two input features (X_1, X_2) and a target variable.

II.II) 3D Scatter Plot of Input and Output

Figure 1 shows the spatial distribution of training samples in the two-dimensional feature space.



II.III) Observations and Insights

- The graph above shows us the output lies relatively on a slanted plane with some points lying outside this plane.
- On further investigation, we found that R^2 value on a normal linear regression gives us a value of about 0.76 which implies a moderate level of accuracy, suggesting that all points do not ideally form a slanted plane.

```
X = df[['x_1', 'x_2']]
y = df['output']

model = LinearRegression()
model.fit(X, y)

y_pred = model.predict(X)
r2 = r2_score(y, y_pred)
print("R² =", r2)
```

✓ 0.0s

```
R² = 0.762529817686938
```

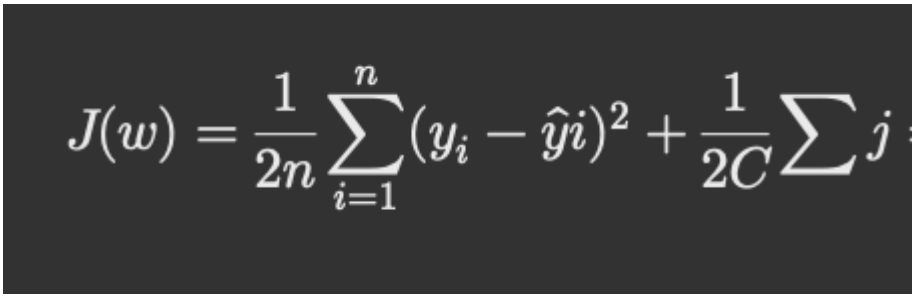
III) Regularization - Ridge and Lasso

III.I Lasso Regression

Lasso Regression (Least Absolute Shrinkage and Selection Operator) is a type of linear regression that introduces L1 regularization to improve model generalization and interpretability. It modifies the standard least squares cost function by adding a penalty term proportional to the absolute values of the regression coefficients.

III.I.I Cost Function in Logistic Regression

The cost function of Lasso Regression is expressed as:



The above equations lead us to conclude that when C decreases, the penalty increases, forcing some weights to become exactly zero. This leads to feature selection and model sparsity, making Lasso Regression effective for high-dimensional or noisy datasets.

By selectively shrinking and eliminating less important features, Lasso improves generalization and maintains predictive efficiency while enhancing model interpretability.

III.I.II Lasso Regression using skLearn

The Lasso regression model was trained using sklearn's Lasso class which handles the mathematics under the hood and provides us with the parameter values and interesting insights. We trained the dataset for various values of C, this led us to draw interesting insights on how C plays a role in Lasso Regression. Sklearn uses alpha = 1/2C.

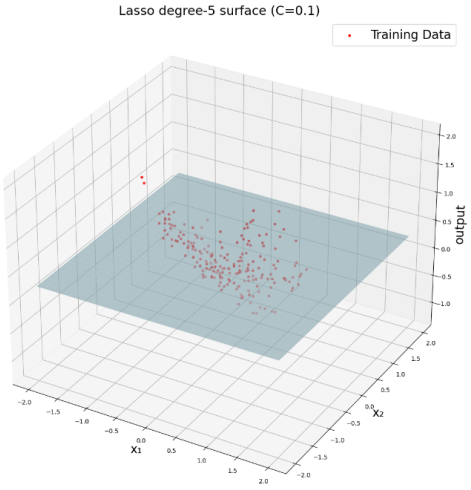
Trained Model Parameters:

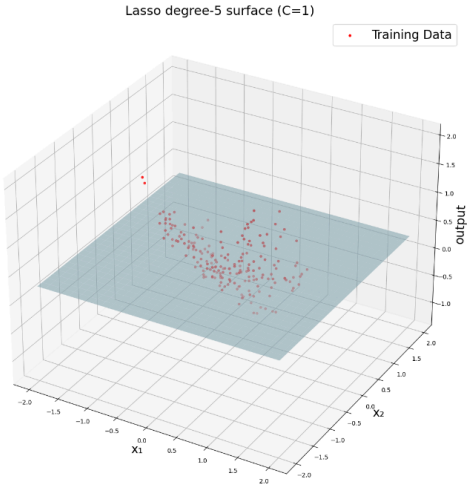
C	Non-zero Coefficients	Key Coefficients	Intercept	Interpretation

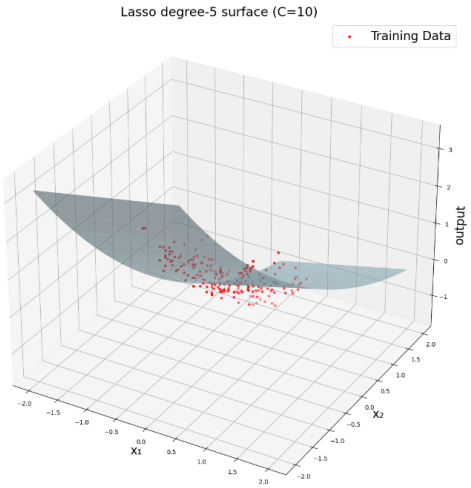
0.1	0	NA	0.2761	Very strong regularization. All weights shrink to zero. The model outputs only the intercept, meaning no feature contributes to prediction.
1	0	NA	0.2761	Same as 0.1
10	2	x_2: -0.8684, x_1^2: 0.3364	0.1954	Moderate regularization. Only x_2 and x_1^2 remain active
100	4	x_2 :-0.9920 x_1^2 :0.8699 x_1 x_2^2 :-0.0051 x_1^4 :0.0308	0.0305	Regularization weakens. Two more features survive: x_1x_2^2 and x_1^4. The model becomes slightly more complex and expressive.
1000	12	x_1 :0.0248 x_2 :-1.0291 x_1^2 :0.7563 x_2^2 :0.0599 x_1^2 x_2 :-0.0003 x_1 x_2^2 :-0.2085 x_1^4 :0.0880 x_1^3 x_2 :0.0863 x_1^2 x_2^2 :0.3635 x_2^4 :-0.1570 x_1^5 :0.1030 x_2^5 :0.0407	0.0307	Weak regularization. Several polynomial terms survive, including higher-order interactions. The model fits more complex nonlinear relationships between x_1 and x_2.

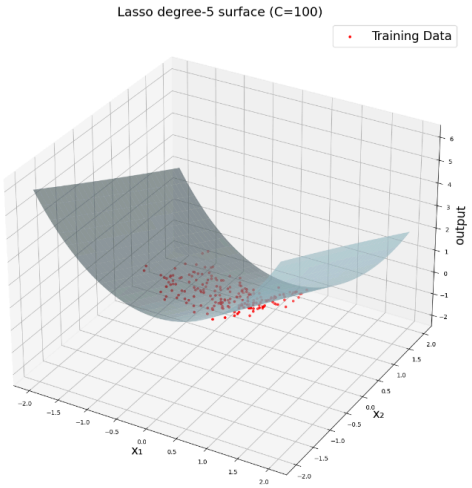
Conclusion: By adding polynomial features up to degree 5 and varying C, Lasso initially sets all coefficients to zero under strong regularization but gradually activates the most predictive terms as C increases, demonstrating its ability to both regularize the model and automatically select the most relevant nonlinear features.

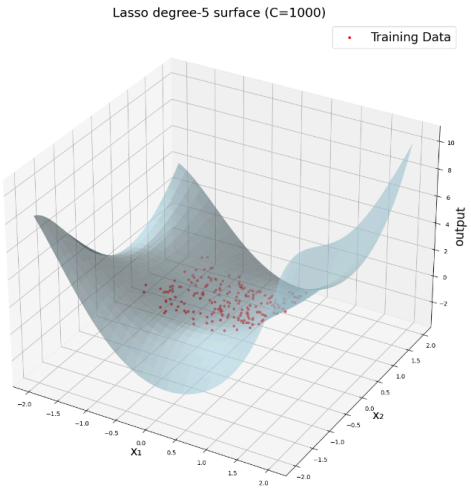
III.I.III Predictions for the target variable for all C's











Interpretation of the surfaces:

As C increases, the Lasso surfaces gradually change from flat to more curved, showing how the model starts including higher-order polynomial terms (up to

x_1^5 and x_2^5) and becomes more complex. The flatter surfaces at low C represent underfitting, while the more curved ones at high C show overfitting—illustrating the bias–variance tradeoff where low C means high bias and low variance, and high C means low bias and high variance.

III.II Ridge Regression

Ridge Regression is a regularized form of linear regression that addresses overfitting and multicollinearity by adding an **L2 penalty** to the cost function. This penalty constrains the magnitude of the model weights, preventing them from growing too large when features are highly correlated or when the dataset is noisy.

III.II.I Cost Function

The cost function for ridge regression:

$$J(w) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}i)^2 + \frac{1}{2C}\sum j$$

Unlike Lasso, Ridge does **not** drive coefficients exactly to zero. Instead, it **shrinks all weights smoothly** toward zero, distributing the effect across correlated features. This helps stabilize the model and improves generalization to unseen data.

Ridge Regression is particularly effective when many predictors contribute to the target variable or when features are correlated, as it reduces variance without completely eliminating any input feature.

III.II.II Ridge Regression using skLearn

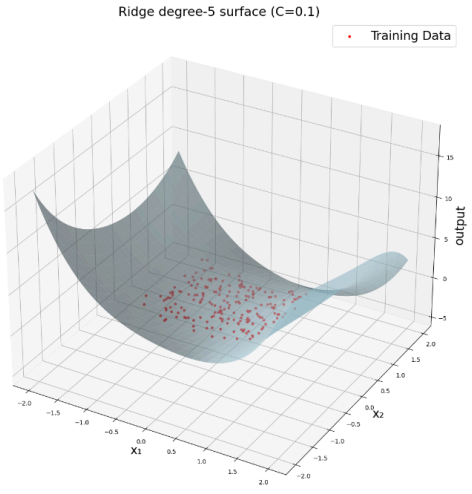
The ridge regression model was trained using sklearn's Ridge class which handles the mathematics under the hood and provides us with the parameter values and interesting insights. We trained the dataset for various values of C, this led us to draw interesting insights on how C plays a role in Ridge Regression. Sklearn uses alpha = 1/2C.

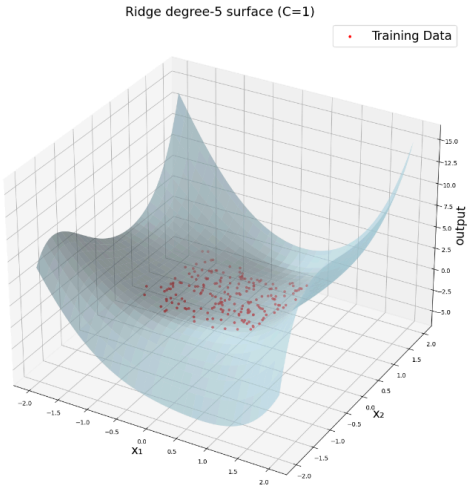
C	**Key Coefficients**	**Intercept**	**Interpretation**

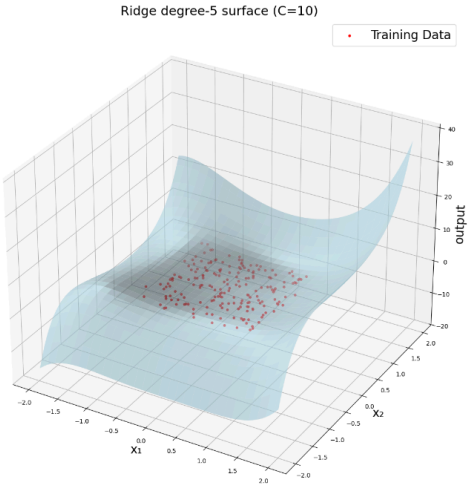
0.1	x_1 :0.0265 x_2 :-0.7831 x_1^2 :0.4728 x_1 x_2 :0.0259 x_2^2 :0.0035 x_1^3 :0.0235 x_1^2 x_2 :-0.1100 x_1 x_2^2 :-0.0748 x_2^3 :-0.2200 x_1^4 :0.3351 x_1^3 x_2 :0.0015 x_1^2 x_2^2 :0.2036 x_1 x_2^3 :0.0007 x_2^4 :-0.0482 x_1^5 :0.0309 x_1^4 x_2 :-0.0293 x_1^3 x_2^2 :-0.0507 x_1^2 x_2^3 :0.0427 x_1 x_2^4 :-0.0717 x_2^5 :-0.0136	0.0852	Ridge applies strong regularization, forcing all coefficients to small magnitudes. The surface is nearly flat and predictions are dominated by the intercept
1	x_1 :0.0212 x_2 :-0.9228 x_1^2 :0.6280 x_1 x_2 :-0.0048 x_2^2 :0.0964 x_1^3 :0.0083 x_1^2 x_2 :-0.1181 x_1 x_2^2 :-0.1361 x_2^3 :-0.2694 x_1^4 :0.2357 x_1^3 x_2 :0.1035 x_1^2 x_2^2 :0.3513 x_1 x_2^3 :0.0117 x_2^4 :-0.1917 x_1^5 :0.0964 x_1^4 x_2 :-0.0117 x_1^3 x_2^2 :-0.0547 x_1^2 x_2^3 :0.1532 x_1 x_2^4 :-0.0523 x_2^5 :0.1974	0.0413	Similar to 0.1
10	x_1 :0.0860 x_2 :-0.8430 x_1^2 :0.7198 x_1 x_2 :-0.0513 x_2^2 :0.1565 x_1^3 :-0.2377 x_1^2 x_2 :-0.2521 x_1 x_2^2 :-0.2731 x_2^3 :-0.6414 x_1^4 :0.1026 x_1^3 x_2 :0.1898 x_1^2 x_2^2 :0.4343 x_1 x_2^3 :0.0226 x_2^4 :-0.2708 x_1^5 :0.3390 x_1^4 x_2 :0.0338 x_1^3 x_2^2 :0.0011 x_1^2 x_2^3 :0.3305 x_1 x_2^4 :0.0524 x_2^5 :0.5146	0.0245	The penalty weakens, and coefficients grow in magnitude, especially for x_2 and x_1^2. Ridge captures moderate nonlinearity without overfitting.
100	x_1 :0.1475 x_2 :-0.7729 x_1^2 :0.7354 x_1 x_2 :-0.0725 x_2^2 :0.1572 x_1^3 :-0.4608 x_1^2 x_2 :-0.3713 x_1 x_2^2 :-0.3665 x_2^3 :-0.8895 x_1^4 :0.0731 x_1^3 x_2 :0.2249 x_1^2 x_2^2 :0.4588 x_1 x_2^3 :0.0328 x_2^4 :-0.2717 x_1^5 :0.5266 x_1^4 x_2 :0.1002 x_1^3 x_2^2 :0.0878 x_1^2 x_2^3 :0.4458 x_1 x_2^4 :0.1056 x_2^5 :0.7076	0.0226	Ridge coefficients continue to increase smoothly. More polynomial interactions (like x_1x_2^2, x_1^4) start contributing, and the fitted surface becomes more expressive.
1000	x_1 :0.1584 x_2 :-0.7610 x_1^2 :0.7368 x_1 x_2 :-0.0760 x_2^2 :0.1565 x_1^3 :-0.5003 x_1^2 x_2 :-0.3926 x_1 x_2^2 :-0.3824 x_2^3 :-0.9299 x_1^4 :0.0696 x_1^3 x_2 :0.2305 x_1^2 x_2^2 :0.4625 x_1 x_2^3 :0.0346 x_2^4 :-0.2709 x_1^5 :0.5592 x_1^4 x_2 :0.1130 x_1^3 x_2^2 :0.1042 x_1^2 x_2^3 :0.4651 x_1 x_2^4 :0.1138 x_2^5 :0.7387 x_1^3 x_2 :0.0863 x_1^2 x_2^2 :0.3635 x_2^4 :-0.1570 x_1^5 :0.1030 x_2^5 :0.0407	0.0225	Ridge regularization becomes weak, allowing most coefficients to reach significant magnitudes, especially for higher-order and interaction terms.

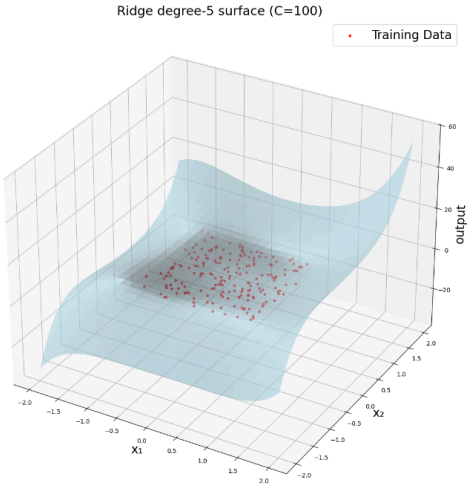
Conclusion: Ridge Regression shows a gradual transition from a smooth, flat surface to a complex, flexible one as C increases. All polynomial terms contribute in varying degrees, making Ridge robust and stable. In comparison, Lasso evolves from complete sparsity to selective complexity, emphasizing interpretability through automatic feature elimination.

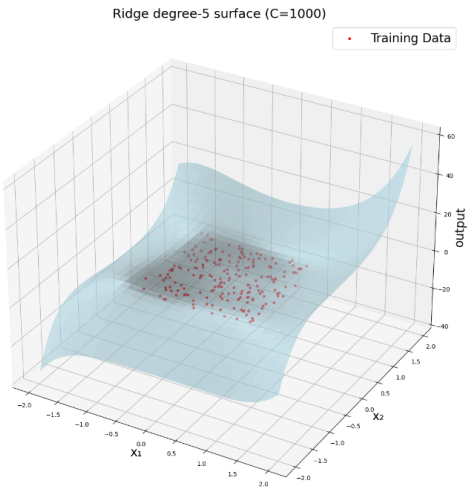
III.II.III Predictions for the target variable for all C's











Ridge Regression transitions gradually from a simple, smooth plane to a detailed nonlinear surface as C increases. Unlike Lasso, which introduces abrupt changes by setting coefficients to zero, Ridge maintains all features with reduced

magnitudes, resulting in continuous and stable surfaces that illustrate its smooth regularization effect.

III.III Conclusion

- **Underfitting** occurs when a model is too simple to capture the true patterns in the data. It has high bias and low variance, producing flat or overly smooth predictions that fail to follow the data trends.
- **Overfitting** occurs when a model is too complex and fits the noise in the training data rather than the underlying relationship. It has low bias but high variance, producing highly curved or irregular prediction surfaces that do not generalize well to new data.

The parameter C directly controls the trade-off between under- and overfitting:

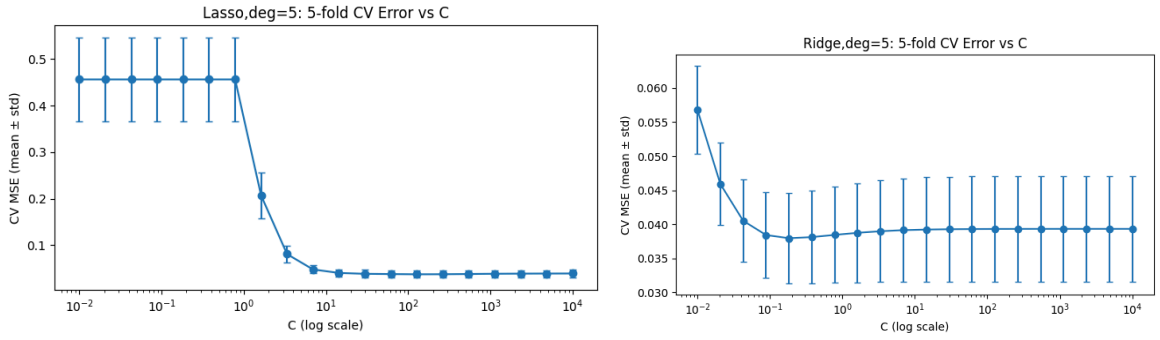
- **Small** C leads to strong regularization(high bias, underfitting).
- **Large** C leads to weak regularization(high variance, overfitting).

By tuning C, we manage model complexity,finding a value that captures essential patterns without fitting random fluctuations in the data.

IV. Cross Validation

Cross-validation is a technique used to evaluate how well a model generalizes to unseen data. It splits the dataset into several folds, trains on some, and tests on the rest. Repeating this across all folds gives an average performance score, helping choose parameters, like C that balance bias and variance effectively.

We split the dataset into 5 and ran CV to figure out accurate values for C ranging from 0.01 to 10000. The range of C values from **0.01 to 10000** was chosen to cover the full spectrum of regularization effects,from very strong to very weak penaltyallowing clear observation of the bias–variance trade-off.The below figure shows us the mean and standard deviation of the CVs for various different C values for both regressions.



Lasso Regression

From the cross-validation curve, the mean squared error (MSE) decreases sharply as C increases from around 0.1 to 10, then stabilizes beyond that point. The error plateau indicates that increasing C further no longer improves generalization but may increase the risk of overfitting.

Recommended C: approximately 10

Reasoning:At C < 1, the strong regularization forces most coefficients to zero, leading to underfitting and high error.At C approx 10, the model includes the most relevant polynomial features, achieving the lowest cross-validation error.For C > 10, the CV error remains nearly constant, showing that additional flexibility provides no real benefit.Thus, **C = 10 achieves the best balance between bias reduction and variance control**, maximizing generalization.

Ridge Regression

The Ridge cross-validation plot shows that MSE decreases quickly up to about C = 0.1 and then stays nearly constant for higher values, indicating minimal change in performance once the model has sufficient flexibility.

Recommended C: approximately 1

Reasoning:For C < 0.1, the penalty is too strong, over-smoothing the model and underfitting the data.Between C = 0.1 and C = 1, the model reaches its minimum cross-validation error, providing the best fit without overfitting.Larger C values (>10) yield no meaningful improvement because Ridge Regression distributes the effect across all coefficients smoothly, maintaining stability.

Conclusion

However, we can see that the Lasso regression provides us a much more stable model for the given dataset, whereas the Ridge regression model does not stabilise for any value of C.

V. APPENDIX

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

from sklearn.linear_model import  LinearRegression, Lasso, Ridge

from sklearn.metrics import r2_score

from sklearn.preprocessing import PolynomialFeatures, StandardScaler

from sklearn.pipeline import make_pipeline



df = []

with open('week3.php', 'r') as f:

   i = f.readlines()

   print(i[0])

   i = i[1:]

   data = [line.strip() for line in i]

   data = [d.split(",") for d in data]

   df = pd.DataFrame(data, columns=['x_1','x_2','output'])

df = df.astype(float)

# For the graph

fig = plt.figure(figsize=(20, 20))  # make it large enough for legibility

ax = fig.add_subplot(111, projection='3d')



# Plot the 3D scatter

ax.scatter(df['x_1'], df['x_2'], df['output'], color='blue', marker='o')
```

```
# Axis labels and title with larger font

ax.set_xlabel(df.columns[0], fontsize=12)

ax.set_ylabel(df.columns[1], fontsize=12)

ax.set_zlabel(df.columns[2], fontsize=12)

ax.set_title("3D Scatter Plot of Dataset", fontsize=14, pad=25)



plt.show()

# To predict if plane or curve

X = df[['x_1', 'x_2']]

y = df['output']



model = LinearRegression()

model.fit(X, y)



y_pred = model.predict(X)

r2 = r2_score(y, y_pred)

print("R² =", r2)

input_poly = df[['x_1', 'x_2']]

output_poly = df['output']



poly = PolynomialFeatures(degree=5, include_bias=False)

input_poly = poly.fit_transform(input_poly)

feature_names = poly.get_feature_names_out(['x_1', 'x_2'])

print("Polynomial feature names:", feature_names)



C_values = [0.1, 1, 10, 100, 1000]
```

```
nonzero_counts = []

nonzero_ridge_counts = []



for C in C_values:

   alpha = 1 / (2 * C)

   lasso = Lasso(alpha=alpha, max_iter=100000, random_state=42)

   ridge = Ridge(alpha=alpha)

   lasso.fit(input_poly, output_poly)

   ridge.fit(input_poly, output_poly)

   # Count non-zero coefficients

   nonzero_lasso = np.sum(np.abs(lasso.coef_) > 1e-6)

   nonzero_ridge = np.sum(np.abs(ridge.coef_) > 1e-6)

   nonzero_counts.append(nonzero_lasso)

   nonzero_ridge_counts.append(nonzero_ridge)




   # Print model details

   print(f"Lasso:-\n C = {C} ;alpha = {alpha:.5f}")

   print(f"Intercept: {lasso.intercept_:.4f}")

   for fname, coef in zip(feature_names, lasso.coef_):

       if abs(coef) > 1e-6:

           print(f"{fname:15s}:{coef:.4f}")

   print(f"Number of non-zero coefficients: {nonzero_lasso}")

   print(f"\nRidge:-\n C = {C} ;alpha = {alpha:.5f}")

   print(f"Intercept: {ridge.intercept_:.4f}")

   for fname, coef in zip(feature_names, ridge.coef_):
```

```
        if abs(coef) > 1e-6:

            print(f"{fname:15s}:{coef:.4f}")

    print(f"Number of non-zero coefficients: {nonzero_ridge}\n")

fig, axes = plt.subplots(1, 2, figsize=(10, 4), sharey=True)



# --- Lasso plot ---

axes[0].plot(C_values, nonzero_counts, marker='o', color='blue')

axes[0].set_xscale('log')

axes[0].set_xlabel("C")

axes[0].set_ylabel("Number of non-zero coefficients")

axes[0].set_title("Lasso Complexity vs C")

axes[0].grid(True, linestyle='--', alpha=0.6)



# --- Ridge plot ---

axes[1].plot(C_values, nonzero_ridge_counts, marker='s', color='green')

axes[1].set_xscale('log')

axes[1].set_xlabel("C")

axes[1].set_title("Ridge Complexity vs C")

axes[1].grid(True, linestyle='--', alpha=0.6)



plt.tight_layout()

plt.show()

poly = PolynomialFeatures(degree=5, include_bias=False)



def predict_surface(C, model, grid_n=60, pad_factor=0.5):

    alpha = 1 / (2 * C)
```

```
    model = model(alpha=alpha, max_iter=200000, random_state=42)
    X_poly = poly.fit_transform(X)
    model.fit(X_poly, y)

    # Define extended grid
    x1_min, x1_max = df["x_1"].min(), df["x_1"].max()
    x2_min, x2_max = df["x_2"].min(), df["x_2"].max()
    r1, r2 = x1_max - x1_min, x2_max - x2_min

    gx = np.linspace(x1_min - pad_factor * r1, x1_max + pad_factor * r1, grid_n)
    gy = np.linspace(x2_min - pad_factor * r2, x2_max + pad_factor * r2, grid_n)
    GX, GY = np.meshgrid(gx, gy)

    X_grid = np.c_[GX.ravel(), GY.ravel()]
    Z = model.predict(poly.transform(X_grid)).reshape(GX.shape)
    return GX, GY, Z

# Define C values to visualize
C_values = [0.1, 1, 10, 100, 1000]

# Create combined figure with 5 subplots
fig = plt.figure(figsize=(35,35))
for i, C in enumerate(C_values):
    GX, GY, Z = predict_surface(C, Ridge)  # or Ridge
    ax = fig.add_subplot(3, 2, i + 1, projection='3d')
    ax.plot_surface(GX, GY, Z, alpha=0.6, color='lightblue', linewidth=0)
```

```
    ax.scatter(df["x_1"], df["x_2"], df["output"], s=12, color='red',label='Training
Data')

    ax.set_xlabel("x₁", fontsize=20)

    ax.set_ylabel("x₂", fontsize=20)

    ax.legend(fontsize=20)

    ax.set_zlabel("output", fontsize=20)

    ax.set_title(f"Ridge degree-5 surface (C={C})", fontsize=21)



plt.tight_layout()

plt.show()



X = df[['x_1','x_2']].to_numpy(dtype=float)

y = df['output'].to_numpy(dtype=float)

def cv_graph(model, model_name):

    mean_mse, std_mse = [], []

    C_grid = np.logspace(-2, 4, 20)

    for C in C_grid:

        alpha = 1.0/(2.0*C)

        pipe = make_pipeline(

            poly,

            StandardScaler(with_mean=True, with_std=True),

            model(alpha=alpha, max_iter=200000, random_state=42)

        )

        scores = cross_val_score(pipe, X, y, cv=5, scoring='neg_mean_squared_error')

        mean_mse.append(abs(scores).mean())

        std_mse.append(scores.std())
```

```
    plt.figure(figsize=(7,4))
    plt.errorbar(C_grid, mean_mse, yerr=std_mse, fmt='-o', capsize=3)
    plt.xscale('log')
    plt.xlabel('C (log scale)')
    plt.ylabel('CV MSE (mean ± std)')
    plt.title(f'{model_name},deg=5: 5-fold CV Error vs C')
    plt.tight_layout()
    plt.show()
    mean_mse = np.array(mean_mse)
    std_mse = np.array(std_mse)
cv_graph(Ridge, "Ridge")
cv_graph(Lasso, "Lasso")
```