Name: Shrey Pendurkar
Class: D15C
Batch: B
Roll No: 38

MLDL - Experiment 2

Aim

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

Dataset Source

- Dataset: Student Performance Dataset
- Kaggle: https://www.kaggle.com/datasets/uciml/student-alcohol-consumption

Used `student-mat.csv` (Math course) as the main file.

Dataset Description

The Student Performance dataset originates from a secondary school in Portugal and combines academic, demographic, and social information about students. It is widely used for educational data mining and regression tasks such as predicting student grades and analyzing the impact of socio-economic factors on academic performance. Unlike artificially clean datasets, this one contains noisy, correlated features and mixed data types (categorical + numerical), making it suitable to test regularized linear models such as Lasso and Ridge.

Key characteristics (for `student-mat.csv`):

- Number of instances: 395 students
- Number of attributes: 33 (30 input features + 3 grade columns)
- File type: CSV (Comma Separated Values)
- Task: Predict the final grade `G3` (0–20) based on student background and behavior.
- Example features:
    - Demographics: `age`, `sex`, `address`, `famsize`, `Pstatus`.

- Parental education: `Medu`, `Fedu`.
- Study and failure history: `studytime`, `failures`, `absences`.
- Support and activities: `schoolsup`, `famsup`, `paid`, `activities`, `internet`, `romantic`.
- Behavior / alcohol: `goout`, `Dalc` (weekday alcohol), `Walc` (weekend alcohol).
- Academic indicators: `G1`, `G2` (first and second period grades).

In all three models, we predict `G3` from all available features after appropriate preprocessing.

---

1. Multiple Linear Regression – Student Performance

Theory

Multiple Linear Regression is used to predict a continuous target based on several independent variables. Here, the final math grade $G3$ is modeled as a linear combination of demographic, behavioral, and academic features.

Mathematical formulation:

$$G3 = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \varepsilon$$

Where:

- $G3$: Dependent variable (final grade).
- $x_1, \ldots, x_n$: Independent variables (e.g., `G1`, `G2`, `studytime`, `Dalc`, `Walc`, `absences`, etc.).
- β_0: Intercept (expected grade when all predictors are zero).
- β_i: Regression coefficients; β_i represents the change in `G3` for a one-unit change in x_i, assuming all other predictors remain constant.
- ε: Error term capturing unexplained variation.

Assumptions (Applied to grades):

- Linearity between predictors and grade.
- No severe multicollinearity among predictors (though `G1` and `G2` are correlated in practice).
- Homoscedasticity of residuals (constant variance).
- Approximately normal residuals.

Limitations on this dataset

1. Multicollinearity
   Features like the partial grades `G1` and `G2` are highly correlated with each other and with `G3`. When predictors are strongly correlated, coefficients become unstable and hard to interpret, and small changes in the data can produce large swings in estimated β values.
2. Curse of Dimensionality / Overfitting
   The dataset has many encoded categorical variables relative to the number of students. With too many predictors, the model may memorize noise in the training data, giving low training error but poor test performance.
3. Assumption of Additivity (no interactions)
   The linear model assumes the effect of each feature is independent and additive. In reality, the effect of alcohol (`Dalc`, `Walc`) on grades might depend on study time or parental education, but such interaction effects are not captured unless we add explicit interaction terms.

Workflow – Multiple Regression

1. Data Collection
   - Load `student-mat.csv` into a Pandas DataFrame.
2. Data Preprocessing
   - Identify categorical variables (e.g., `school`, `sex`, `address`, `famsize`, `schoolsup`, `famsup`, `internet`, `romantic`).
   - Apply one-hot encoding to convert categorical columns into numerical form (`get_dummies`, `drop_first=True`).
   - Handle any missing values if present (mean imputation for numeric columns).
3. Train–Test Split
   - Split into 80% training and 20% testing sets using `train_test_split` to evaluate generalization and avoid overfitting.
4. Model Training & Prediction
   - Train a Multiple Linear Regression (OLS) model on the training set.

    - Predict `G3` for the test set.
5. Model Evaluation
    - Compute RMSE to measure average prediction error in grade points.
    - Compute R² to measure the proportion of variance in `G3` explained by the predictors.
6. Conclusion
    - If $R2$ is high and RMSE is around 2–3 grade points, the model is capturing most of the variation; remaining error suggests scope for regularized models (Lasso and Ridge).

Performance Analysis

The performance of the Multiple Linear Regression model on the Student Performance (Math) dataset is evaluated using Root Mean Squared Error (RMSE) and the R² score.

1. Root Mean Squared Error (RMSE)
   The model achieves an RMSE of 2.38 grade points (≈ 2.3784). This means that, on average, the predicted final grade `G3` differs from the actual grade by about 2.4 points on the 0–20 scale. In the context of school marks, this error corresponds roughly to less than half a letter grade, indicating that the model's predictions are reasonably close to true student performance, though individual predictions can still deviate by a couple of points.
2. R² Score (Coefficient of Determination)
   The R² score of 0.7241 indicates that the model explains approximately 72% of the variance in students' final math grades. This shows that the combination of demographic, behavioral, and academic features (especially prior grades `G1` and `G2`) captures most of the factors that drive final performance. The remaining 28% of unexplained variance likely arises from unmeasured influences (such as teacher quality, short-term events, or motivation) and from non-linear relationships or interactions that a purely linear, additive model cannot fully represent.

Hyperparameter Tuning

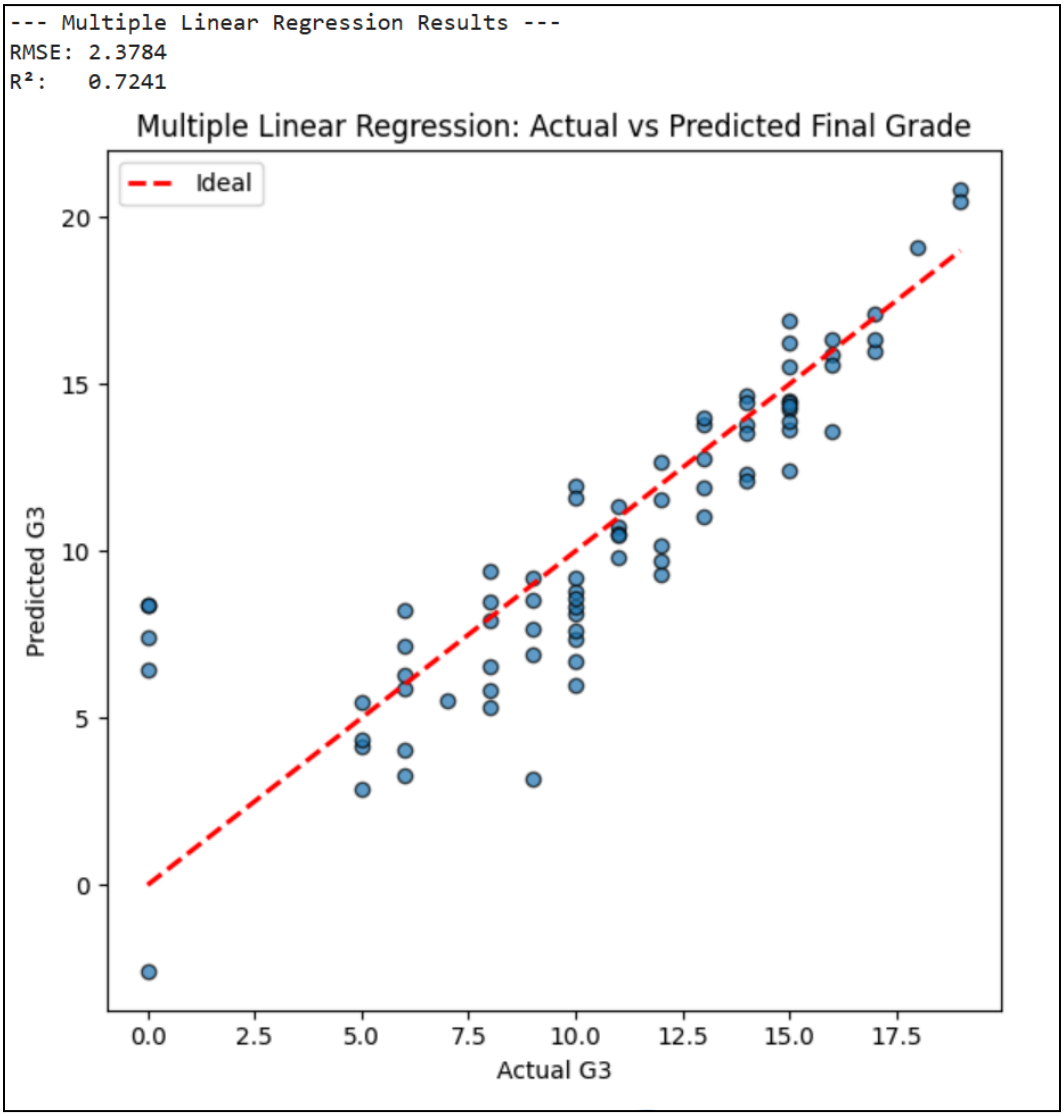
As in your life-expectancy reference, standard OLS regression has no hyperparameters to tune: the coefficients are uniquely determined by minimizing squared error. To introduce tunable regularization, we move to Lasso and Ridge.

Code and Output - Multiple Regression

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

from sklearn.model_selection import
train_test_split
from sklearn.linear_model import
LinearRegression
from sklearn.metrics import
mean_squared_error, r2_score

# 1. Data Collection
df = pd.read_csv('student-mat.csv')

target_col = 'G3'

# 2. Data Preprocessing
categorical_cols =
df.select_dtypes(include=['object'])
.columns.tolist()
categorical_cols = [c for c in
categorical_cols if c != target_col]

df_encoded = pd.get_dummies(df,
columns=categorical_cols,
drop_first=True)

X =
df_encoded.drop(columns=[target_col]
)
y = df_encoded[target_col]

# 3. Train–Test Split
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2,
random_state=42
)

# 4. Model Training
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# 5. Prediction & Evaluation
y_pred = lin_reg.predict(X_test)

rmse =
np.sqrt(mean_squared_error(y_test,
y_pred))
r2 = r2_score(y_test, y_pred)

print("\n--- Multiple Linear
Regression Results ---")
print(f"RMSE: {rmse:.4f}")
print(f"R²:   {r2:.4f}")
```

```
# 6. Visualization: Actual vs Predicted Final Grade
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred, alpha=0.7, edgecolor='k')
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         'r--', linewidth=2, label='Ideal')

plt.xlabel('Actual G3')
plt.ylabel('Predicted G3')
plt.title('Multiple Linear Regression: Actual vs Predicted Final Grade')
plt.legend()
plt.tight_layout()
plt.show()
```

```
--- Multiple Linear Regression Results ---
RMSE: 2.3784
R²:   0.7241
```



2. Lasso Regression – Student Performance

Theory

Lasso Regression (Least Absolute Shrinkage and Selection Operator) is a linear regression method with L1 regularization. It adds a penalty proportional to the absolute values of the coefficients, shrinking some of them exactly to zero. This provides both overfitting control and automatic feature selection, which is useful when there are many categorical dummies and correlated predictors in the student dataset.

Objective function:

$$\text{Cost} = \sum_{i=1}^{N} (G3_i - \hat{G3}_i)^2 + \alpha \sum_{j=1}^{n} |\beta_j|$$

- α controls the strength of regularization.
- Larger $\alpha \rightarrow$ stronger penalty $\rightarrow$ more coefficients driven to zero (sparser model).

Advantages:

- Reduces model variance and overfitting.
- Selects a subset of the most informative student attributes (e.g., `G1`, `G2`, key behavior variables), simplifying interpretation.

Limitations to this dataset

1. Handling Correlated Variables
   When features are highly correlated (e.g., `G1` and `G2`), Lasso tends to pick one and shrink the other to zero arbitrarily, which may obscure the fact that both are important.
2. p > n Limit
   If the number of predictors exceeds the number of samples, Lasso can keep at most `n` variables with non-zero coefficients, limiting its ability to reflect all relevant predictors in extremely high-dimensional settings.

Workflow – Lasso Regression

1. Data Collection

- Load `student-mat.csv` into a DataFrame.

2. Data Preprocessing
   - One-hot encode all categorical columns.
   - Separate `X` (all features except `G3`) and `y` (`G3`).
3. Train–Test Split & Scaling
   - Split into 80% train / 20% test.
   - Apply StandardScaler to standardize features, as Lasso is sensitive to scale.
4. Model Training & Prediction
   - Use LassoCV (Lasso with cross-validation) to automatically select the best α over a log-spaced range.
   - Train Lasso on the scaled training data and predict on the scaled test data.
5. Model Evaluation & Feature Selection
   - Compute RMSE and R^2.
   - Count non-zero coefficients to see how many features were retained.
   - Examine the largest positive and negative coefficients to understand which student factors most strongly influence grades.
6. Conclusion
   - Lasso should achieve comparable RMSE/R^2 to simple multiple regression but with a smaller, more interpretable set of predictors.

Performance Analysis

The performance of the implemented Lasso Regression model on the Student Performance (Math) dataset is evaluated using RMSE, R^2, and the number of non-zero coefficients.

1. RMSE Analysis
   The Lasso model attains an RMSE of 2.17 grade points (≈ 2.1732), which is lower than the Multiple Linear Regression RMSE of 2.38. This means that, on average, Lasso's predictions of the final grade `G3` are closer to the true values, reducing the typical error by about 0.2 grade points. On a 0–20 grading scale, this improvement is meaningful, indicating that regularization helps the model generalize better to unseen students.
2. R^2 Score Analysis
   The R^2 score of 0.7697 shows that Lasso explains about 77% of the variance in final math grades, an improvement over the 72% explained by the unregularized

model. This indicates that Lasso captures the underlying relationships between student attributes and final performance more effectively, balancing bias and variance to obtain a stronger overall fit.

3. Feature Selection and Model Sparsity
   Out of 41 encoded features, only 17 retain non-zero coefficients, meaning Lasso has automatically discarded more than half of the predictors as non-informative or redundant. This sparsity both reduces the risk of overfitting and simplifies interpretation: the remaining 17 features can be viewed as the most influential factors for predicting `G3` (typically including `G1`, `G2`, and a subset of behavioral and support variables). Overall, Lasso provides better accuracy and a more compact model compared with standard Multiple Regression.

Hyperparameter Tuning

Hyperparameter tuned: Alpha (α) – L1 regularization strength, analogous to your life-expectancy document.

- Smaller α: weaker regularization, more features retained.
- Larger α: stronger regularization, more coefficients shrunk to zero.

Using LassoCV:

- Evaluate a logarithmic range of α values (e.g., 50 values from 0.0001 to 10) with k-fold cross-validation.
- Choose the α that minimizes the cross-validated error; you can report this value as your optimal hyperparameter.

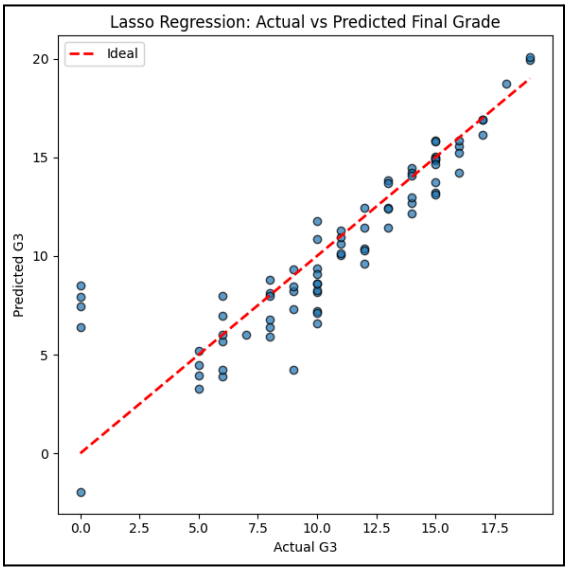
Code and Output - Lasso Regression

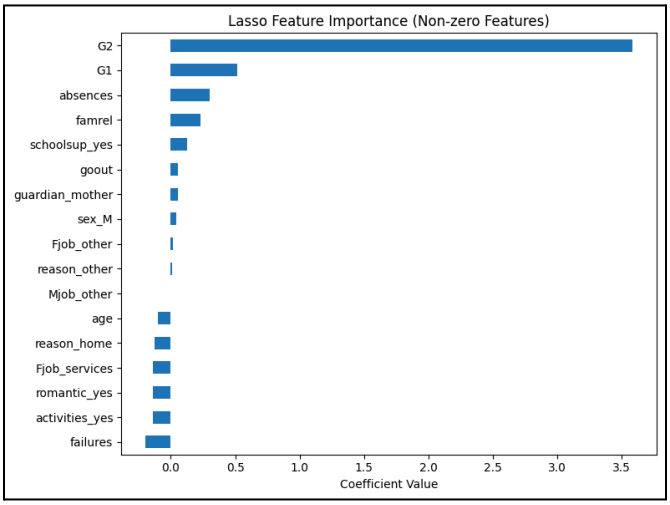
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import
StandardScaler
from sklearn.linear_model import
LassoCV
from sklearn.metrics import
mean_squared_error, r2_score
```

```
# 1. Data Collection
df = pd.read_csv('student-mat.csv')

target_col = 'G3'

# 2. Data Preprocessing
categorical_cols =
df.select_dtypes(include=['object'])
.columns.tolist()
categorical_cols = [c for c in
categorical_cols if c != target_col]

df_encoded = pd.get_dummies(df,
columns=categorical_cols,
drop_first=True)

X =
df_encoded.drop(columns=[target_col]
)
y = df_encoded[target_col]

feature_names = X.columns

# 3. Train–Test Split
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2,
random_state=42
)

# 4. Scaling
scaler = StandardScaler()
X_train_scaled =
scaler.fit_transform(X_train)
X_test_scaled =
scaler.transform(X_test)

# 5. Lasso with Cross-Validation
alphas = np.logspace(-4, 1, 50)

lasso_cv = LassoCV(alphas=alphas,
cv=5, random_state=42,
max_iter=10000)
lasso_cv.fit(X_train_scaled,
y_train)

print("\nBest alpha (Lasso):",
lasso_cv.alpha_)

# 6. Prediction & Evaluation
y_pred =
lasso_cv.predict(X_test_scaled)

rmse =
np.sqrt(mean_squared_error(y_test,
y_pred))
r2 = r2_score(y_test, y_pred)

print("\n--- Lasso Regression
Results ---")
print(f"RMSE: {rmse:.4f}")
print(f"R²:   {r2:.4f}")

non_zero = np.sum(lasso_cv.coef_ !=
0)
print(f"Non-zero coefficients:
{non_zero} / {len(lasso_cv.coef_)}")

# 7. Visualization 1: Actual vs
Predicted Final Grade
```

```
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred,
alpha=0.7, edgecolor='k')
plt.plot([y_test.min(),
y_test.max()],
         [y_test.min(),
y_test.max()],
         'r--', linewidth=2,
label='Ideal')

plt.xlabel('Actual G3')
plt.ylabel('Predicted G3')
plt.title('Lasso Regression: Actual
vs Predicted Final Grade')
plt.legend()
plt.tight_layout()
plt.show()
```

```
# 8. Visualization 2: Lasso Feature
Importance (Non-zero Coefficients)
coef_series =
pd.Series(lasso_cv.coef_,
index=feature_names)
coef_nonzero =
coef_series[coef_series !=
0].sort_values()

plt.figure(figsize=(8, 6))
coef_nonzero.plot(kind='barh')
plt.xlabel('Coefficient Value')
plt.title('Lasso Feature Importance
(Non-zero Features)')
plt.tight_layout()
plt.show()
```





---

3. Ridge Regression – Student Performance

Theory

Ridge Regression is a linear regression model with L2 regularization. It adds a penalty proportional to the square of the coefficients to the loss function, which is especially helpful when predictors are correlated, as in this student dataset.

Objective function:

$$\text{Cost} = \sum_{i=1}^{N} (G3_i - \hat{G3}_i)^2 + \alpha \sum_{j=1}^{n} \beta_j^2$$

- α controls the strength of L2 penalty.
- Coefficients are shrunk towards zero but never become exactly zero, so all predictors remain in the model.

Advantages:

- Reduces the impact of multicollinearity (e.g., between `G1`, `G2`, and behavior variables).
- Produces more stable coefficient estimates and often better generalization.

Limitations

1. No Feature Selection
   Ridge cannot "turn off" unimportant features completely; it only shrinks them. With many predictors, the final model can still be dense and harder to interpret than a Lasso model.
2. Bias–Variance Tradeoff
   If α is too large, coefficients are over-shrunk, leading to underfitting and poor predictive performance. Selecting α carefully is crucial.

Workflow – Ridge Regression

1. Data Collection & Preprocessing
   - Same as Lasso: one-hot encode categoricals, set `G3` as target.
2. Train–Test Split & Scaling
   - 80/20 split.
   - Standardize all features using `StandardScaler`.
3. Model Training & Hyperparameter Selection

- Use RidgeCV with k-fold cross-validation to choose the best α from a log-spaced range (e.g., 0.01–100).

4. Prediction & Evaluation
    - Predict `G3` on the test set.
    - Compute RMSE and R^2.
    - Optionally inspect the largest coefficients to understand influential factors; note that all coefficients remain non-zero but may be heavily shrunk.
5. Conclusion
    - Ridge typically yields similar or slightly smoother performance compared with Multiple Regression, with more stable coefficients in presence of correlated predictors.

Performance Analysis

The performance of the Ridge Regression model on the Student Performance (Math) dataset is evaluated using RMSE and R^2, with the regularization strength controlled by the hyperparameter alpha.

1. RMSE Analysis
   The Ridge model achieves an RMSE of 2.37 grade points (≈ 2.3667). This implies that, on average, its predictions for the final grade `G3` deviate from the actual grades by about 2.4 points on the 0–20 scale. The error is very similar to that of the Multiple Linear Regression model (2.38), indicating that Ridge provides comparable predictive accuracy while introducing regularization to stabilize coefficients in the presence of correlated features such as `G1` and `G2`.
2. R^2 Score Analysis
   With an R^2 score of 0.7268, the Ridge model explains roughly 72.7% of the variance in students' final math grades. This is slightly higher than the 72.4% from plain Multiple Regression but below the 76.9% achieved by Lasso. The result shows that Ridge successfully captures most of the linear relationships between student attributes and final performance, offering a modest improvement in robustness over OLS without significantly changing overall fit quality.
3. Effect of Regularization (Alpha)
   The selected alpha ≈ 4.94 corresponds to a moderate L2 penalty, which shrinks all coefficients towards zero but keeps every feature in the model. This helps mitigate multicollinearity and reduces the variance of coefficient estimates, yielding a more stable model than pure OLS. However, because Ridge does not

perform feature selection, it remains less sparse and slightly less accurate than the Lasso model, which both improves R^2 and removes irrelevant predictors.

Hyperparameter Tuning

Hyperparameter tuned: Alpha (α) – L2 regularization strength, exactly like in your life-expectancy document.

- Smaller α: coefficients close to OLS (low bias, higher variance).
- Larger α: stronger shrinkage (higher bias, lower variance) to combat multicollinearity.

Using RidgeCV:

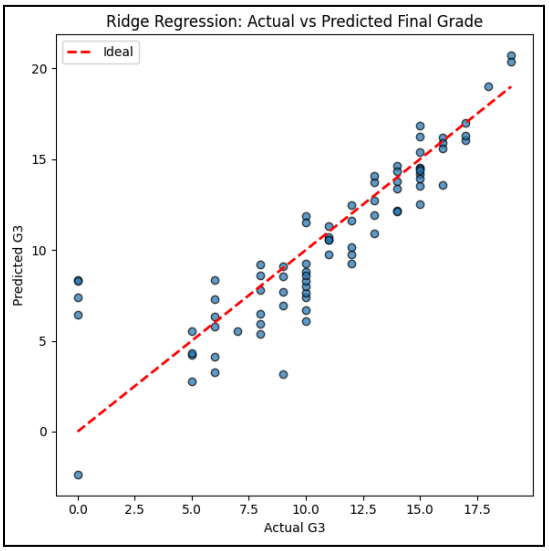
- Define a log-spaced set of α values (e.g., 50 values from 0.01 to 100).
- Cross-validate to pick the α giving the best average validation score; report it as the optimal value.

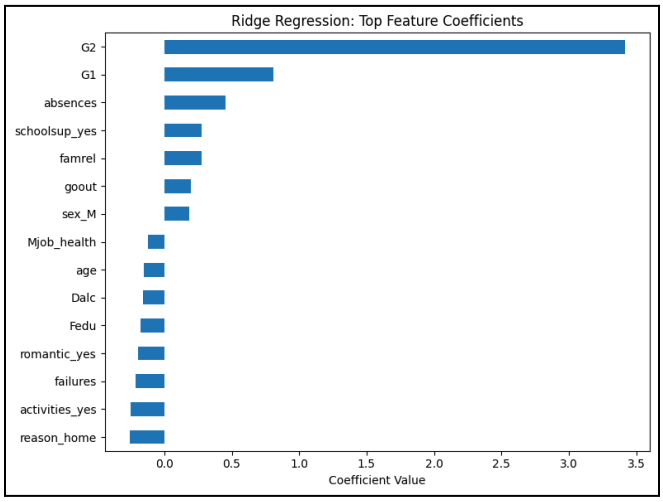
Code and Output – Ridge Regression

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

from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import
StandardScaler
from sklearn.linear_model import
RidgeCV
from sklearn.metrics import
mean_squared_error, r2_score

# 1. Data Collection
df = pd.read_csv('student-mat.csv')

target_col = 'G3'

# 2. Data Preprocessing
categorical_cols =
df.select_dtypes(include=['object'])
.columns.tolist()
categorical_cols = [c for c in
categorical_cols if c != target_col]

df_encoded = pd.get_dummies(df,
columns=categorical_cols,
drop_first=True)
```

```
X =
df_encoded.drop(columns=[target_col]
)
y = df_encoded[target_col]

feature_names = X.columns

# 3. Train–Test Split
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2,
random_state=42
)

# 4. Scaling
scaler = StandardScaler()
X_train_scaled =
scaler.fit_transform(X_train)
X_test_scaled =
scaler.transform(X_test)

# 5. Ridge with Cross-Validation
alphas = np.logspace(-2, 2, 50)

ridge_cv = RidgeCV(alphas=alphas,
cv=5)
ridge_cv.fit(X_train_scaled,
y_train)

print("\nBest alpha (Ridge):",
ridge_cv.alpha_)

# 6. Prediction & Evaluation
y_pred =
ridge_cv.predict(X_test_scaled)
rmse =
np.sqrt(mean_squared_error(y_test,
y_pred))
r2 = r2_score(y_test, y_pred)

print("\n--- Ridge Regression
Results ---")
print(f"RMSE: {rmse:.4f}")
print(f"R²:   {r2:.4f}")

# 7. Visualization 1: Actual vs
Predicted Final Grade
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred,
alpha=0.7, edgecolor='k')
plt.plot([y_test.min(),
y_test.max()],
         [y_test.min(),
y_test.max()],
         'r--', linewidth=2,
label='Ideal')

plt.xlabel('Actual G3')
plt.ylabel('Predicted G3')
plt.title('Ridge Regression: Actual
vs Predicted Final Grade')
plt.legend()
plt.tight_layout()
plt.show()

# 8. Visualization 2: Top Ridge
Coefficients
coef_series =
pd.Series(ridge_cv.coef_,
index=feature_names)
```

```
coef_sorted =
coef_series.sort_values(key=lambda
s: s.abs(), ascending=False)

top_n = 15
plt.figure(figsize=(8, 6))
coef_sorted.head(top_n).sort_values(
).plot(kind='barh')
plt.xlabel('Coefficient Value')
plt.title('Ridge Regression: Top
Feature Coefficients')
plt.tight_layout()
plt.show()
```





Conclusion

The practical demonstrates that all three models can predict final math grades reasonably well, but regularization clearly helps. Multiple Linear Regression gives a strong baseline (about 72% variance explained), while Ridge adds stability to the coefficients without changing accuracy much. Lasso performs best overall, slightly reducing RMSE, increasing R^2 to around 77%, and automatically selecting a smaller set of important features, making the final model both more accurate and easier to interpret.