Practical-5

AIM: Implementation of simple linear regression to predict weight by taking height as input using a student- created dataset (either randomly generated or collected from classmates). Implement multiple linear regression, polynomial regression, lasso, and ridge regression on the winequalityred.csv dataset. Part 1: Predict Weight Using Height

1. Import Libraries:

Import all necessary libraries like Pandas, NumPy, Matplotlib, Seaborn, and

scikitlearn in one cell. import pandas as pd import numpy as np import

matplotlib.pyplot as plt import seaborn as sns

2. Load the Height-Weight Dataset:

o Use the given link to load the dataset into your environment. o Link:

weight-height.csv

df=pd.read_csv('/content/drive/MyDrive/dataset/weight-height.csv')

3. Check Data Information: o Use info(), describe() ,etc. methods to get a

summary of the dataset (mean, min, max values) and also check data types.

o Use head() to display the first few rows of the data.

df.info() df.describe()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Gender  10000 non-null  object
 1   Height  10000 non-null  float64
 2   Weight  10000 non-null  float64
dtypes: float64(2), object(1)
memory usage: 234.5+ KB
```

	Height	Weight
count	10000.000000	10000.000000
mean	66.367560	161.440357
std	3.847528	32.108439
min	54.263133	64.700127
25%	63.505620	135.818051
50%	66.318070	161.212928
75%	69.174262	187.169525
max	78.998742	269.989699

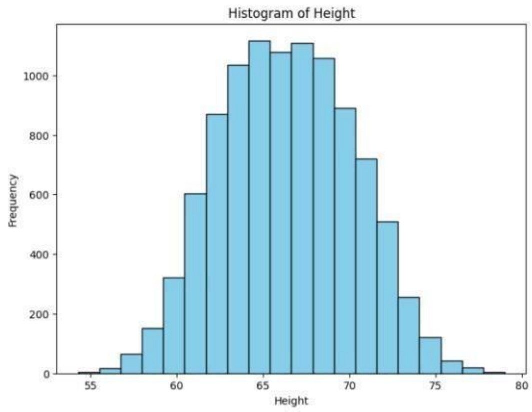
df.head()

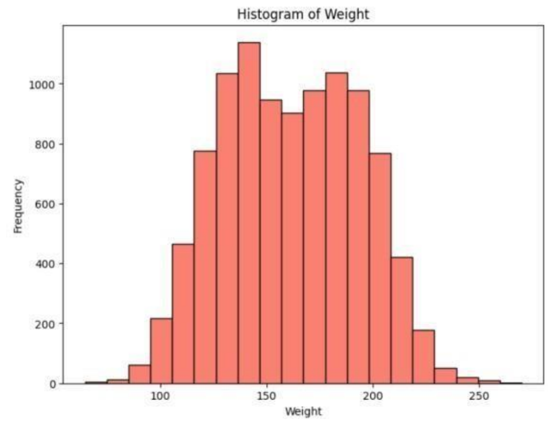
	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

4. Histogram and Skewness:

o Plot histograms for the Height and Weight columns. o Calculate and display the skewness for both columns to check data distribution.

plt.figure(figsize=(8, 6)) plt.hist(df['Height'], bins=20, color='skyblue', edgecolor='black') plt.xlabel('Height') plt.ylabel('Frequency') plt.title('Histogram of Height') plt.show() plt.figure(figsize=(8, 6)) plt.hist(df['Weight'], bins=20, color='salmon', edgecolor='black') plt.xlabel('Weight') plt.ylabel('Frequency') plt.title('Histogram of Weight') plt.show()





skewness_height = df['Height'].skew() skewness_weight = df['Weight'].skew() print(f"Skewness of Height: {skewness_height}") print(f"Skewness of Weight: {skewness_weight}")

```
Skewness of Height: 0.04936908937689031
Skewness of Weight: 0.03295450444592437
```

5. Check for Missing Values: o Use isnull().sum() to identify any missing values.

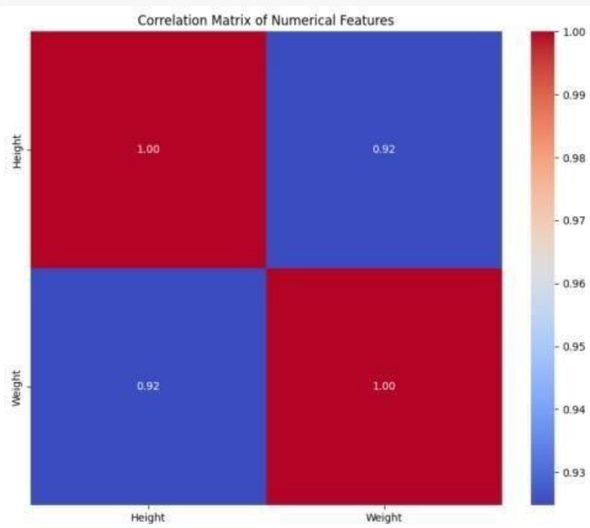
Df.isnull().sum()

	0
Gender	0
Height	0
Weight	0

dtype: int64

6. Correlation Matrix: o Select the numerical columns and use a heatmap to display the correlation matrix.

```
numerical_df = df.select_dtypes(include=np.number)  correlation_matrix =
numerical_df.corr()
plt.figure(figsize=(10,  8))      sns.heatmap(correlation_matrix,  annot=True,
cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')  plt.show()
```



7. Split Data into X and Y:

o Set Height as X and Weight as Y.

o Split the dataset into training and testing sets, and display the shapes (number of

rows and columns) of both training and testing sets.

```
X = df['Height']
y = df['Weight']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) print(X_train.shape,
y_train.shape)  print(X_test.shape, y_test.shape)
```

```
(8000,) (8000,)
(2000,) (2000,)
```

8. **Simple Linear Regression: o Train a simple linear regression model using the training data. o Check the model's accuracy, coefficient, and intercept.**

```
from sklearn.linear_model import LinearRegression  from
sklearn.metrics import mean_squared_error, r2_score model =
LinearRegression()
X_train_reshaped = X_train.values.reshape(-1, 1)
model.fit(X_train_reshaped, y_train)  X_test_reshaped =
X_test.values.reshape(-1, 1) y_pred =
model.predict(X_test_reshaped)  mse =
mean_squared_error(y_test, y_pred)  r2 = r2_score(y_test, y_pred)
print("Model Coefficients:", model.coef_) print("Model
Intercept:", model.intercept_)  print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Model Coefficients: [7.70218561]
Model Intercept: -349.7878205824451
Mean Squared Error: 149.00350418448127
R-squared: 0.8577317777038499
```

9. **Predict and Evaluate: o Predict Weight using the test set, and calculate the root mean squared error**

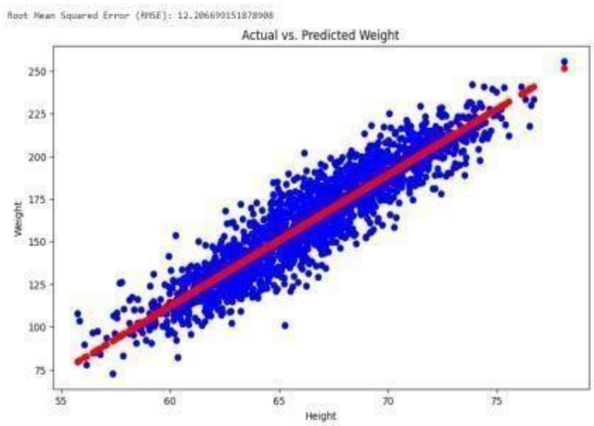
(RMSE). o Plot the predicted and actual values for comparison. rmse

= np.sqrt(mse) print("Root Mean Squared Error

(RMSE):", rmse)

plt.figure(figsize=(10, 6))

```
plt.scatter(X_test, y_test, color='blue', label='Actual Weight')
plt.scatter(X_test, y_pred, color='red', label='Predicted
Weight')    plt.xlabel('Height')    plt.ylabel('Weight')
plt.title('Actual vs. Predicted Weight')
plt.show()
```



10. Model Performance Metrics:

o Compare the adjusted R-squared, mean absolute error, mean squared error, and R-squared using built-in methods.

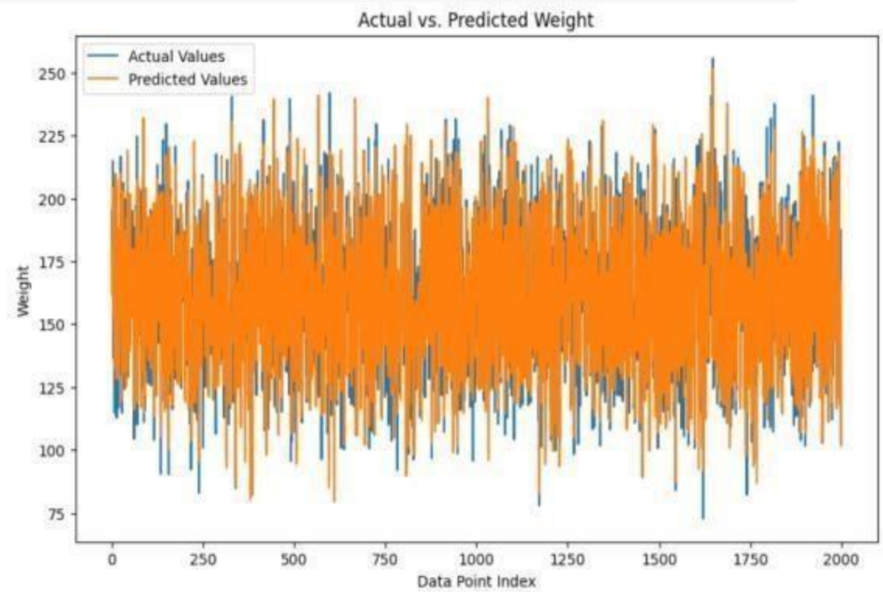
```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)   n = len(y_test)
p = 1  adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
print("Adjusted Rsquared:", adjusted_r2)
print("\nModel Evaluation Metrics:")
print("------------------------")  print("R-squared:",
r2)
print("Adjusted R-squared:", adjusted_r2)
print("Mean Squared Error:", mse)   print("Root
Mean Squared Error (RMSE):", rmse)  print("Mean
Absolute Error:", mae)
```

```
Mean Absolute Error: 9.691933801884572
Adjusted R-squared: 0.8576605723873854

Model Evaluation Metrics:
------------------------
R-squared: 0.8577317777038499
Adjusted R-squared: 0.8576605723873854
Mean Squared Error: 149.00350418448127
Root Mean Squared Error (RMSE): 12.206699151878908
Mean Absolute Error: 9.691933801884572
```

11. Plot Actual vs Predicted Errors: o Draw a graph to show the difference between actual and predicted values.

```
plt.figure(figsize=(10, 6))  plt.plot(y_test.values, label='Actual Values')
                    plt.plot(y_pred,  label='Predicted  Values')
plt.xlabel('Data Point Index')  plt.ylabel('Weight')
plt.title('Actual vs. Predicted Weight')  plt.legend()   plt.show()
```



Part 2: Wine Quality Prediction Using Multiple Regressions

12. Load the Wine Quality Dataset:

o winequality-red.csv df1=pd.read_csv('/content/drive/MyDrive/dataset/winequality-red.csv', delimiter=';')

13. Check Dataset Information:

o Use head(), info(), and describe() to explore the dataset (similar to part 1).

df1.head()

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

df1.info() df1.describe()

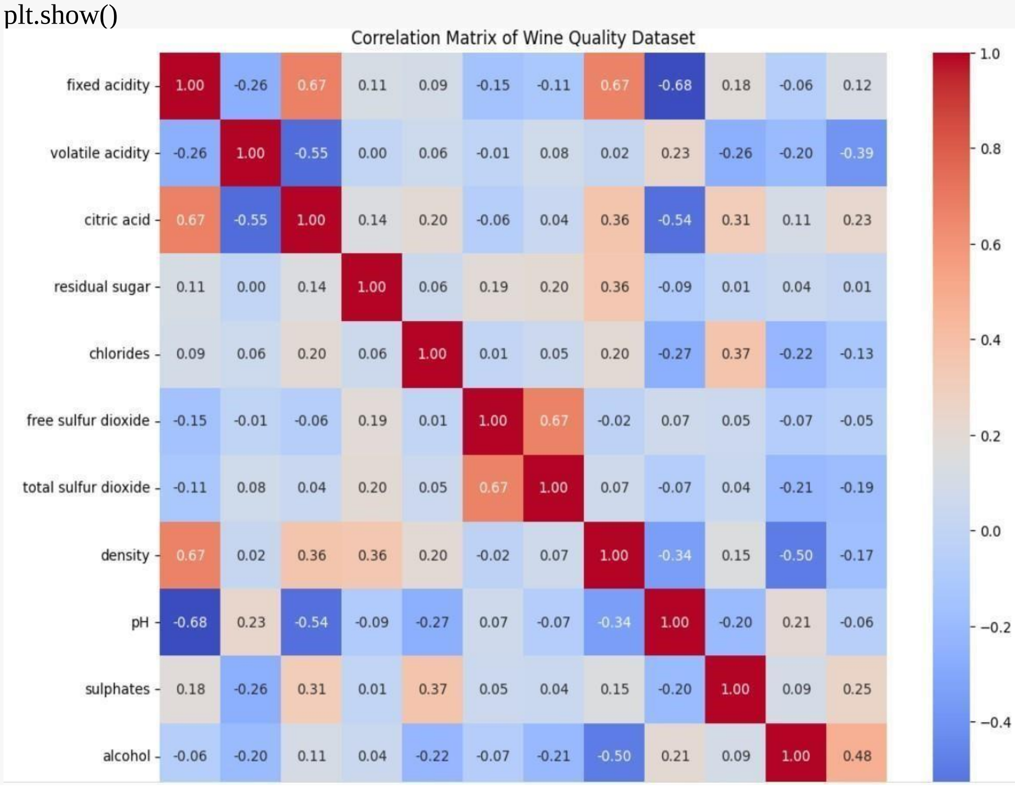
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.636023
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.807569
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

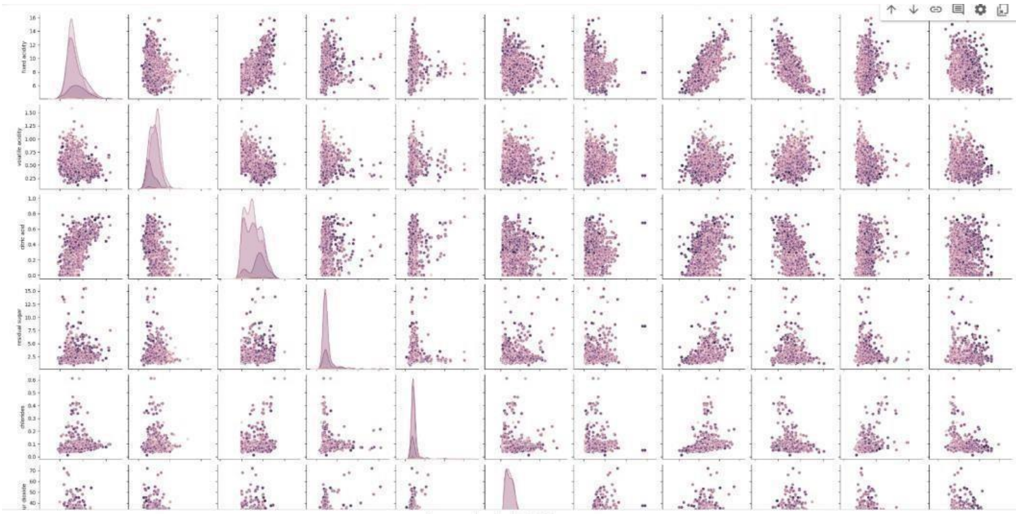
14. Correlation Matrix and Heatmap:

o Calculate the correlation between columns and visualize it with a heatmap.

```
correlation_matrix = df1.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Wine Quality Dataset')
plt.show()
```



15. Pairplot: o Create a pairplot of the dataset, using the quality column as the hue. sns.pairplot(df1, hue='quality') plt.show()



16. Split Data into X and Y: o Set X as all columns except quality, and Y as the quality column. o Split the data into training and testing sets.

```
X = df1.drop('quality', axis=1)  y
= df1['quality']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

17. Linear Regression Model: o Train a linear regression model on the training data (without polynomial features). o Check the model's score and calculate the RMSE.

```
n_features = X_train.shape[1]  desired_columns
= 10
X_train_reshaped = X_train.values.reshape(X_train.shape[0], -1)
X_test_reshaped = X_test.values.reshape(X_test.shape[0], -1)
```

o Create a dataframe to display the actual and predicted values side by side.

```
import pandas as pd  import numpy as np  from
sklearn.linear_model import LinearRegression  from
sklearn.model_selection import train_test_split from
sklearn.metrics import mean_squared_error, r2_score  model
= LinearRegression()  model.fit(X_train, y_train)  y_pred =
model.predict(X_test)   mse = mean_squared_error(y_test,
y_pred)  rmse = np.sqrt(mse)  r2 = r2_score(y_test, y_pred)
print("Model Score (Rsquared):", r2)
print("Root Mean Squared Error (RMSE):", rmse)
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})  print(comparison_df)
```

```
Model Score (R-squared): 0.40318034127962254
Root Mean Squared Error (RMSE): 0.6245199307980126
      Actual  Predicted
803        6   5.346664
124        5   5.056313
350        6   5.664470
682        5   5.464515
1326       6   5.725185
...      ...        ...
1259       6   5.688153
1295       5   5.232255
1155       5   5.280535
963        6   6.272466
704        4   5.197072

[320 rows x 2 columns]
```

18. Polynomial Regression:

o Use polynomial features with degree 2 to transform X, then split the dataset. o Train a new linear regression model on the transformed data and compare the RMSE and score with the previous model.

```
from sklearn.preprocessing import PolynomialFeatures poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train_reshaped)
X_test_poly = poly.transform(X_test_reshaped) model_poly = LinearRegression() model_poly.fit(X_train_poly, y_train)
y_pred_poly = model_poly.predict(X_test_poly) mse_poly = mean_squared_error(y_test, y_pred_poly) rmse_poly = np.sqrt(mse_poly) r2_poly = r2_score(y_test, y_pred_poly)
print(r2_poly) print(rmse_poly) print("\nModel Comparison:") print("-----------------") print("Linear Regression:") print("R-squared:", r2) print("RMSE:", rmse)
print("Polynomial Regression:") print("R-squared:", r2_poly)
print("RMSE:", rmse_poly)
```

```
0.22018748469533955
0.7138711704261158

Model Comparison:
-----------------
Linear Regression:
R-squared: 0.40318034127962254
RMSE: 0.6245199307980126
Polynomial Regression:
R-squared: 0.22018748469533955
RMSE: 0.7138711704261158
```

19. Lasso Regression:

o Train a Lasso regression model on the original dataset. o Display the model's score, RMSE, weights, and intercept.

```
from sklearn.linear_model import Lasso
X_train_reshaped =aX_train.values.reshape(-1, 1) if X_train.ndim == 1 else X_train
X_test_reshaped = X_test.values.reshape(-1, 1) if X_test.ndim == 1 else X_test
lasso_model = Lasso(alpha=1.0)  lasso_model.fit(X_train_reshaped, y_train)
y_pred_lasso = lasso_model.predict(X_test_reshaped)  lasso_score =
lasso_model.score(X_test_reshaped, y_test)  lasso_rmse =
np.sqrt(mean_squared_error(y_test, y_pred_lasso))  print("Lasso Regression
Results:")  print("------------------------")   print(f"Model Score (R-squared):
{lasso_score}")  print(f"RMSE: {lasso_rmse}")  print(f"Weights:
{lasso_model.coef_}")  print(f"Intercept:
{lasso_model.intercept_}")
```

```
Lasso Regression Results:
------------------------
Model Score (R-squared): 0.009014670905063582
RMSE: 0.8047451268061234
Weights: [ 0.         -0.          0.          0.         -0.          0.
 -0.00397837 -0.         -0.          0.          0.        ]
Intercept: 5.809544459115708
```

20. Ridge Regression: o Train a Ridge regression model on the original dataset and check the model's score, RMSE, weights, and intercept.

```
from sklearn.linear_model import Ridge
X_train_reshaped = X_train.values.reshape(-1, 1) if X_train.ndim == 1 else X_train
X_test_reshaped = X_test.values.reshape(-1, 1) if X_test.ndim == 1 else X_test
ridge_model = Ridge(alpha=1.0) # You can adjust the alpha parameter
ridge_model.fit(X_train_reshaped, y_train) y_pred_ridge =
ridge_model.predict(X_test_reshaped) ridge_score = ridge_model.score(X_test_reshaped,
y_test) ridge_rmse =
np.sqrt(mean_squared_error(y_test, y_pred_ridge)) print("Ridge Regression
Results:") print("------------------------")  print(f"Model Score (R-squared):
{ridge_score}") print(f"RMSE: {ridge_rmse}") print(f"Weights:
{ridge_model.coef_}") print(f"Intercept:
{ridge_model.intercept_}")
```

```
Ridge Regression Results:
------------------------
Model Score (R-squared): 0.3987064350775855
RMSE: 0.6268563375138516
Weights: [ 1.96137645e-02 -1.02152929e+00 -1.64270261e-01  6.23679545e-04
 -1.22677375e+00  5.68588748e-03 -3.56053037e-03 -1.11526006e-02
 -3.76222673e-01  7.46955766e-01  2.97591067e-01]
Intercept: 3.893865480452895
```

21. Stepwise Regression (LassoLars): o Train a Stepwise regression amodel and check the score, RMSE, weights, and intercept.

```
import statsmodels.api as sm
X_train_sm = sm.add_constant(X_train) X_test_sm
= sm.add_constant(X_test)
stepwise_model = sm.OLS(y_train, X_train_sm).fit()
print(stepwise_model.summary())
y_pred_stepwise = stepwise_model.predict(X_test_sm)
stepwise_score
= stepwise_model.rsquared
stepwise_rmse = np.sqrt(mean_squared_error(y_test,
y_pred_stepwise)) print("Stepwise Regression Results:")
print("------------------------") print(f"Model Score (R-squared):
{stepwise_score}") print(f"RMSE:
{stepwise_rmse}") print(f"Weights:
{stepwise_model.params}") print(f"Intercept:
{stepwise_model.params['const']}")
```

```
                          coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                  14.3551     23.705      0.606      0.545     -32.150      60.860
fixed acidity           0.0231      0.029      0.801      0.423      -0.033       0.080
volatile acidity       -1.0013      0.137     -7.283      0.000      -1.271      -0.732
citric acid            -0.1408      0.168     -0.839      0.402      -0.470       0.188
residual sugar          0.0066      0.017      0.391      0.696      -0.026       0.039
chlorides              -1.8065      0.457     -3.949      0.000      -2.704      -0.909
free sulfur dioxide     0.0056      0.002      2.252      0.025       0.001       0.011
total sulfur dioxide   -0.0036      0.001     -4.419      0.000      -0.005      -0.002
density               -10.3516     24.192     -0.428      0.669     -57.812      37.109
pH                     -0.3937      0.215     -1.830      0.067      -0.816       0.028
sulphates               0.8412      0.126      6.651      0.000       0.593       1.089
alcohol                 0.2819      0.030      9.465      0.000       0.223       0.340
==============================================================================
Omnibus:                       28.708   Durbin-Watson:                   2.003
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               46.050
Skew:                          -0.192   Prob(JB):                     1.00e-10
Kurtosis:                       3.847   Cond. No.                     1.12e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.12e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
Stepwise Regression Results:
------------------------
Model Score (R-squared): 0.34799261935298575
RMSE: 0.6245199307980335
Weights: const                  14.355105
fixed acidity            0.023085
volatile acidity        -1.001304
citric acid             -0.140821
residual sugar           0.006564
chlorides               -1.806503
free sulfur dioxide      0.005627
total sulfur dioxide    -0.003644
density                -10.351594
pH                      -0.393688
sulphates                0.841172
alcohol                  0.281890
```

22. Bayesian Regression (Combination of Ridge and Lasso): o Train a Bayesian Ridge regression model and evaluate the score, RMSE, weights, and intercept.

```
from sklearn.linear_model import BayesianRidge bayesian_ridge_model
= BayesianRidge()  bayesian_ridge_model.fit(X_train_reshaped,
y_train) y_pred_bayesian =
bayesian_ridge_model.predict(X_test_reshaped)    bayesian_score   =
bayesian_ridge_model.score(X_test_reshaped, y_test) bayesian_rmse =
np.sqrt(mean_squared_error(y_test, y_pred_bayesian)) print("Bayesian
Ridge Regression Results:")  print("------------------------")
print(f"Model Score (R-squared): {bayesian_score}") print(f"RMSE:
{bayesian_rmse}")  print(f"Weights: {bayesian_ridge_model.coef_}")
```

```
Bayesian Ridge Regression Results:
------------------------
Model Score (R-squared): 0.3976681363057878
RMSE: 0.6273973240399033
Weights: [ 2.06310753e-02 -1.01898939e+00 -1.63663095e-01  2.55978485e-04
 -1.11598062e+00  5.69587774e-03 -3.54540924e-03 -9.16860682e-03
 -3.61820193e-01  7.30247914e-01  2.98760717e-01]
Intercept: 3.823329283198931
```

Name: DHRUV SHERE
Enrollment No: 23012022021
Batch: 5IT-B-2