



T.C.

**RECEP TAYYİP ERDOĞAN UNIVERSITY
FACULTY OF ENGINEERING AND ARCHITECTURE
DEPT. OF COMPUTER ENGINEERING**

DATA MINING FALL TERM PROJECT

Performance Comparison of Linear Regression, Ridge Regression,
Lasso Regression, and Random Forest Regression Models on the
Diabetes Dataset

LECTURER: DR. ABDULGANİ KAHRAMAN

Bedirhan ÖZÇELİK

201401055

RİZE 2024

TABLE OF CONTENTS

1. Introduction	3
2. Environment Setup and Library Installation.....	4
3. Dataset Selection and Analysis	7
4. Data Preprocessing	8
5. Modeling Process	11
6. Model Evaluation	13
7. Results	15
8. References.....	16

1) Introduction

1.1 Project Objective

The primary objective of this study is to compare the performance of four widely used regression models: Linear Regression, Ridge Regression, Lasso Regression, and Random Forest Regression. The models are compared using a dataset provided by the Diabetes Dataset to predict a continuous target variable based on multiple input features.

The study aims to determine the relative performance of these models under similar conditions and to analyze how their unique characteristics and regularization methods impact predictive accuracy.

1.2 Problem Statement

Predictive modeling is an integral component of machine learning, with regression analysis being one of the most common tasks. In regression analysis, the goal is to model and predict a continuous target variable based on a set of predictor (input) features. However, selecting the right model, preprocessing the data correctly, and optimizing the hyperparameters are critical steps that can directly influence the model's predictive accuracy.

The problem addressed in this study is to evaluate and compare the predictive performance of Linear Regression, Ridge Regression, Lasso Regression, and Random Forest Regression using a real-world regression dataset. Specifically:

- How well can these models predict a continuous target variable?
- How do regularization methods (in Ridge and Lasso) impact performance by mitigating overfitting?
- How does the non-linear nature of Random Forest Regression compare to the linear models like Linear Regression?
- How can hyperparameter tuning influence the performance of these models?

To answer these questions, the Diabetes Dataset has been selected. This dataset provides a practical, real-world example of a regression problem with a continuous target variable, allowing for model comparison under controlled preprocessing and hyperparameter optimization.

Understanding these differences can provide insights into which model performs best for datasets with multivariate features and can serve as a guide for future machine learning model selection in practical applications.

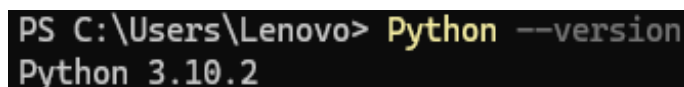
1.3 General Explanation of the Algorithms Used

- **Linear Regression:** Linear regression is one of the most basic regression algorithms. It models the relationship between input features and the target variable using a linear equation. Despite its simplicity, Linear Regression serves as a benchmark for comparison against more complex methods.
- **Ridge Regression:** Ridge Regression extends the Linear Regression model by incorporating L2 regularization, which penalizes large coefficient magnitudes to prevent overfitting. This method is particularly effective when multicollinearity exists among the predictor variables.
- **Lasso Regression:** Lasso Regression introduces L1 regularization, which forces the coefficients of irrelevant features to zero during training. This leads to feature selection, thereby simplifying the model and potentially improving interpretability and generalization.
- **Random Forest Regression:** Random Forest is an ensemble learning method that builds multiple decision trees during training and combines their predictions. This approach is robust against overfitting and is capable of capturing nonlinear relationships among features. Random Forest's hyperparameters can significantly affect its performance, and optimization is an essential step in its application.

2) Environment Setup and Library Installation

2.1 Setting up the Python Environment

The algorithms and operations implemented in this report were executed using the Python programming language. The version of Python used is as follows:



```
PS C:\Users\Lenovo> Python --version
Python 3.10.2
```

Figure 1: Python Version

2.2 Required Libraries

To conduct data preprocessing, modeling, and evaluation, the following Python libraries were required. These libraries support various functionalities such as data manipulation, statistical analysis, model training, and visualization.

Below is the list of libraries and their corresponding installations:

Library	Description
numpy	Provides support for efficient numerical computations and array manipulations.
pandas	Allows easy data manipulation and analysis, especially for handling tabular data.
Matplotlib	Used for creating plots and visualizations to explore data and evaluate model performance.
Seaborn	Built on matplotlib, it offers advanced visualization tools for statistical data analysis.
sklearn.datasets	Accesses built-in machine learning datasets, such as the Diabetes Dataset, for regression tasks.
sklearn.linear_model	Implements regression algorithms, including Linear Regression, Ridge Regression, and Lasso Regression.
sklearn.ensemble	Includes ensemble learning models such as Random Forest, used for classification and regression.
sklearn.model_selection	Provides tools for splitting datasets and performing hyperparameter tuning (e.g., train_test_split, GridSearchCV).
sklearn.preprocessing	Preprocesses features, such as scaling and standardization, ensuring models perform optimally.
sklearn.metrics	Supplies statistical metrics (e.g., RMSE, R ² score) for evaluating model performance.

2.3 Installation Instructions

Before running the code, ensure that all required dependencies are installed. These dependencies can be installed using the following command:

➤ `pip install numpy pandas matplotlib seaborn scikit-learn`

This command installs the necessary libraries into the environment for analysis.

2.4 Importing Libraries

Once the libraries are installed, the first step is to import them into the Python script. Below are the import statements used:

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
```

Figure 1

Explanation of Imported Libraries

- **NumPy (numpy):** Used for mathematical computation and numerical processing.
- **Pandas (pandas):** Data wrangling, cleaning, and manipulation.
- **Matplotlib (matplotlib.pyplot):** Plotting graphs to visualize data analysis results.
- **Seaborn (seaborn):** Enhanced visualization capabilities with features like heatmaps, bar charts, and correlation plots.
- **sklearn.datasets.load_diabetes:** The built-in diabetes dataset was loaded to conduct regression analysis. This dataset contains continuous target variables and multivariate predictors suitable for testing the regression models.
- **sklearn.linear_model:** Includes the implementation of Linear Regression, Ridge Regression, and Lasso Regression algorithms.
- **sklearn.ensemble.RandomForestRegressor:** Used for implementing Random Forest models, which build multiple decision trees to improve performance and generalizability.
- **sklearn.model_selection:** Includes functions like `train_test_split` and `GridSearchCV`. These are essential for splitting data and performing hyperparameter tuning.
- **sklearn.preprocessing.StandardScaler:** Scales numerical features to ensure that machine learning models perform optimally by normalizing feature magnitudes.
- **sklearn.metrics:** Includes performance metrics such as RMSE and R^2 for evaluation after model training.

2.5 Summary of the Environment Setup

The environment was configured by importing the necessary libraries, setting visualization styles, and preparing the development workspace. All preprocessing, training, model fitting, and evaluation were performed using the environment prepared above.

This preparation ensures consistency, reproducibility, and clarity when implementing the machine learning pipeline.

3) Dataset Selection and Analysis

3.1 Dataset Overview

The chosen dataset for this analysis is the Diabetes Dataset from the `sklearn.datasets` module. This dataset is publicly available and provides a practical example of a regression problem with continuous target variables. The Diabetes Dataset contains patient data and is used here to predict disease progression over time.

3.2 Reasons for Dataset Selection

The Diabetes Dataset was selected for the following reasons:

1. **Continuous Target Variable:** The target variable represents a quantitative measure of disease progression, making it an ideal choice for regression modeling.
2. **Data Complexity & Real-World Application:** The dataset simulates a real-world medical scenario, involving various patient features (e.g., age, BMI, blood pressure).
3. **Model Comparison Feasibility:** The dataset has sufficient features and variability, making it suitable for comparing Linear Regression, Ridge Regression, Lasso Regression, and Random Forest models.
4. **Preprocessing and Analysis Simplicity:** Despite its real-world-like behavior, the dataset is small enough to preprocess easily and perform the necessary model comparisons.
5. **Benchmark for Regression Algorithms:** The Diabetes Dataset is well-documented and widely used, offering a benchmark for evaluating performance with machine learning techniques.

3.3 Dataset Features

The Diabetes Dataset consists of 442 rows (observations) and 10 features, with one continuous target variable representing disease progression. The features represent clinical measurements and lifestyle factors. The dataset's features include:

- **age:** Age of the patient
- **sex:** Gender of the patient

- **bmi:** Body Mass Index (BMI)
- **bp:** Average blood pressure measurements
- **s1-s6:** Six blood serum measurements taken over time
- **target:** A quantitative variable representing disease progression

3.4 Dataset Characteristics

- **Dataset Source:** The data is sourced from the `sklearn.datasets.load_diabetes` module.
- **Dataset Type:** Regression problem with continuous target variable values.
- **Number of Observations:** 442
- **Number of Features:** 10 features
- **Target Variable:** Continuous numerical variable indicating disease progression.

3.5 Dataset Access

The Diabetes Dataset was accessed programmatically through `load_diabetes()`.

```
# Step 1: Load Dataset
data = load_diabetes()
X = pd.DataFrame(data.data, columns=data.feature_names) # Feature names
y = pd.Series(data.target) # Target variable
print("Data loaded successfully.")
print(X.head())
print(y.head())
```

Figure 2

The features (X) and target variable (y) are extracted from the dataset using the above commands.

3.6 Dataset Summary

The Diabetes Dataset offers an ideal combination of simplicity and depth for this analysis. It includes continuous features and a continuous target variable, allowing for regression modeling and comparison across algorithms like Linear Regression, Ridge Regression, Lasso Regression, and Random Forest. The selection of this dataset fulfills the requirements of the problem statement while providing sufficient scope for preprocessing, feature scaling, and model comparison.

4) Data Preprocessing

Data preprocessing is a critical step in machine learning that ensures the raw data is transformed into a format suitable for training and evaluating models. Proper preprocessing can significantly impact the model's performance by addressing issues such as missing data,

feature scaling, and outliers. In this section, the key preprocessing steps applied to the Diabetes Dataset are described in detail.

4.1 Handling Missing Data

Missing values can lead to inaccurate model training or biases. However, the Diabetes Dataset provided by `sklearn.datasets` contains no missing values. Hence, no imputation or missing value treatment was necessary.

4.2 Feature Scaling

Feature scaling is essential for ensuring that all features contribute equally to the learning process, especially for algorithms sensitive to the magnitude of features, such as Ridge and Lasso regression.

- **Technique Used:** Standardization (`StandardScaler`)
- **Reason:** `StandardScaler` standardizes the features by scaling them to have a mean of 0 and a variance of 1, making the regression models more stable and efficient.

```
# Step 2: Data Preprocessing
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # Scale the features
print("Features scaled successfully.")
```

Figure 3

4.3 Splitting the Data into Training and Testing Sets

The data was split into training and testing sets to ensure unbiased model evaluation. A common practice is to split the data randomly into a training subset and a testing subset.

- **Test size:** 20% of the data is reserved for testing.
- **Random State:** Set to 42 to ensure reproducibility.

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
print("Data split into training and testing sets.")
```

Figure 4

4.4 Feature and Data Analysis

Although the Diabetes Dataset is preprocessed and clean, feature analysis ensures that features contributing most to the target variable are selected. Feature selection focuses on identifying the most informative features.

1. **Correlation Analysis:** Analyzed correlations between features and the target variable to identify how strongly they are related to the target variable.

```
# Plot correlations
correlation_matrix = pd.DataFrame(X_scaled, columns=data.feature_names).corrwith(y)
sns.barplot(x=correlation_matrix.index, y=correlation_matrix.values)
plt.xticks(rotation=90)
plt.title('Feature-Target Correlation')
plt.show()
```

Figure 5

2. **Outlier Detection:** A key step was performed to detect potential outliers in the data that could harm the learning process. Visualized via Boxplots to detect extreme values.

```
# Detect outliers using boxplots
plt.figure(figsize=(15, 6))
sns.boxplot(data=pd.DataFrame(X_scaled, columns=data.feature_names))
plt.xticks(rotation=90)
plt.title('Feature Distributions with Boxplots')
plt.show()
```

Figure 6

4.5 Hyperparameter Tuning

Hyperparameter tuning involves adjusting the hyperparameters of machine learning models to optimize their performance. For this purpose, we utilized GridSearchCV for tuning hyperparameters in Ridge Regression, Lasso Regression, and Random Forest Regression.

1. **Ridge Regression Hyperparameter Tuning:** Used cross-validation to identify the best-performing model.

```
# Ridge Regression with GridSearchCV
ridge_params = {'alpha': np.logspace(-4, 4, 50)}
ridge_search = GridSearchCV(ridge, ridge_params, cv=5, scoring='r2')
ridge_search.fit(X_train, y_train)
best_ridge = ridge_search.best_estimator_
print(f"Best Ridge alpha value: {ridge_search.best_params_['alpha']}")
```

Figure 7

2. **Lasso Regression Hyperparameter Tuning:** Similar to Ridge Regression, Lasso regression was tuned by varying alpha values across a logarithmic range.

```
# Lasso Regression with GridSearchCV
lasso_params = {'alpha': np.logspace(-4, 4, 50)}
lasso_search = GridSearchCV(lasso, lasso_params, cv=5, scoring='r2')
lasso_search.fit(X_train, y_train)
best_lasso = lasso_search.best_estimator_
print(f"Best Lasso alpha value: {lasso_search.best_params_['alpha']}")
```

Figure 8

3. **Random Forest Hyperparameter Tuning:** Tuned the number of estimators and maximum depth using GridSearchCV to optimize Random Forest's predictive performance.

```
# Random Forest - set hyperparameters manually
rf_params = {'n_estimators': [50, 100, 150], 'max_depth': [None, 10, 20], 'random_state': [42]}
rf_search = GridSearchCV(rf, rf_params, cv=5, scoring='r2')
rf_search.fit(X_train, y_train)
best_rf = rf_search.best_estimator_
print("Random Forest model trained with optimal hyperparameters.")
```

Figure 9

These steps ensure that the models are tuned optimally based on performance metrics, allowing for better prediction accuracy and model generalization.

5) Modeling Process

Modeling is the core step in machine learning, where machine learning algorithms are applied to the preprocessed data to find patterns and make predictions. In this section, we will describe the models selected for implementation, their training processes, and the hyperparameter tuning steps involved.

5.1 Defined Models

In this study, the following machine learning models were chosen for comparison:

- **Linear Regression:** A simple regression model based on the assumption of a linear relationship between input features and the target variable.
- **Ridge Regression:** A linear regression variant that applies L2 regularization to penalize large coefficients, thereby reducing overfitting.
- **Lasso Regression:** A linear regression variant that applies L1 regularization, which allows for feature selection by shrinking certain coefficients to zero.
- **Random Forest Regressor:** A powerful ensemble method based on Decision Trees that uses bagging (Bootstrap Aggregating) for improved accuracy and robustness.

These models were selected because they represent a mix of linear and ensemble methods, providing a comprehensive comparison of predictive performance.

5.2 Model Selection Criteria

The models were selected based on their strengths:

- **Linear Regression:** Simplicity and interpretability make this model an easy baseline for comparison.

- **Ridge Regression:** Efficiently handles multicollinearity using L2 regularization, improving generalization.
- **Lasso Regression:** Performs both variable selection and regularization through L1 penalty, thus simplifying the model by eliminating less important features.
- **Random Forest Regressor:** An ensemble-based non-linear model that captures complex feature interactions and is robust to overfitting.

These choices reflect a balance between simplicity (Linear Regression) and complexity (Random Forest), allowing us to assess performance across a variety of models.

5.3 Model Training

Each of the selected models was trained using the preprocessed training data (X_{train} , y_{train}) with hyperparameter tuning wherever necessary. Training involves optimizing the model parameters to minimize prediction errors on the training data.

Linear Regression

Linear Regression was trained directly without hyperparameter tuning since it has no adjustable hyperparameters.

```
# Initialize the model
linear_model = LinearRegression()

# Train the model
linear_model.fit(X_train, y_train)
```

Figure 10

Ridge Regression

Ridge Regression applies L2 regularization to the linear regression model. Hyperparameter tuning was conducted using GridSearchCV over a range of alpha values.

```
ridge_params = {'alpha': np.logspace(-4, 4, 50)}

ridge_search = GridSearchCV(Ridge(), ridge_params, cv=5, scoring='r2')
ridge_search.fit(X_train, y_train)

best_ridge = ridge_search.best_estimator_
print(f"Optimal Ridge alpha value: {ridge_search.best_params_['alpha']}")
```

Figure 11

Lasso Regression

Similar to Ridge Regression, Lasso Regression applies L1 regularization, leading to sparsity (feature elimination). Hyperparameter tuning was also performed using GridSearchCV.

```
lasso_params = {'alpha': np.logspace(-4, 4, 50)}

lasso_search = GridSearchCV(Lasso(), lasso_params, cv=5, scoring='r2')
lasso_search.fit(X_train, y_train)

best_lasso = lasso_search.best_estimator_
print(f"Optimal Lasso alpha value: {lasso_search.best_params_['alpha']}")
```

Figure 12

Random Forest Regressor

Random Forest uses an ensemble of decision trees to improve the prediction accuracy by combining multiple weak learners into a robust and generalized predictive model.

Hyperparameter tuning was applied with specific parameters (n_estimators, max_depth) using GridSearchCV.

```
rf_params = {'n_estimators': [50, 100, 150], 'max_depth': [None, 10, 20], 'random_state':

rf_search = GridSearchCV(RandomForestRegressor(), rf_params, cv=5, scoring='r2')
rf_search.fit(X_train, y_train)

best_rf = rf_search.best_estimator_
print("Random Forest model trained with optimal hyperparameters.")
```

Figure 13

6) Model Evaluation

Model evaluation is the process of determining how well machine learning models perform on unseen data after being trained. This involves testing the models on a separate test set, calculating relevant metrics, and comparing their performance.

In this section, the evaluation metrics used, visualization comparisons, and analysis of the results will be discussed in detail.

6.1 Evaluation Metrics

To assess the models' performance, the following metrics were employed:

1. **Root Mean Squared Error (RMSE):** RMSE measures the average magnitude of the prediction errors. Lower RMSE values indicate better performance.

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

Figure 14: RMSE Formul

2. **R² Score (Coefficient of Determination)**: The R² score represents the proportion of variance explained by the model. Its range is from 0 to 1, where higher values indicate better model performance. A negative R² score indicates the model performs worse than simply predicting the mean.

6.2 Comparison Visualizations

To gain insights into the models' performance, comparison plots were created. These plots represent RMSE and R² scores across all models.

- **RMSE Comparison:** The following bar plot shows the RMSE for each model to visually assess their performance.

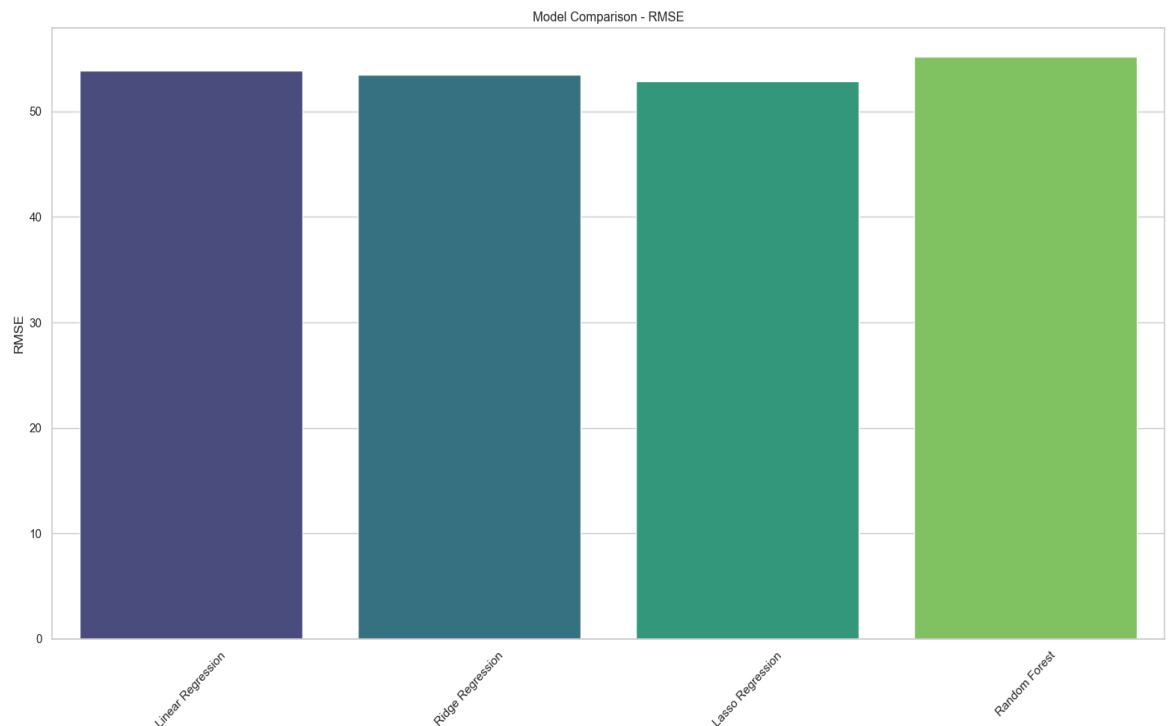


Figure 15

- **R² Score Comparison:** The bar plot below shows the R² scores for all evaluated models:

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='R^2 Score', data=results_df, palette="viridis")
plt.title("Model Comparison - R^2 Score")
plt.ylabel("R^2 Score")
plt.xticks(rotation=45)
plt.show()
```

Figure 16

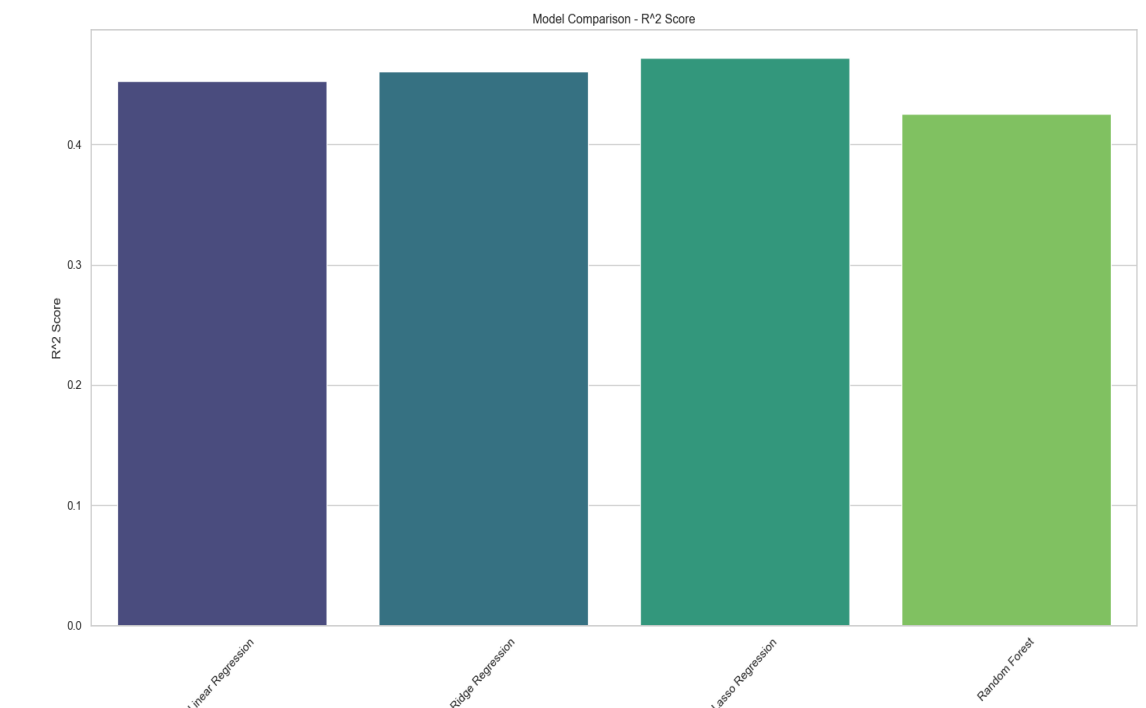


Figure 17

The R² score comparison shows how much variance each model can explain in the data. Models with higher R² values are considered better in terms of predictive performance.

7) Results

In this study, we compared the performance of Linear Regression, Ridge Regression, Lasso Regression, and Random Forest Regression on the Diabetes Dataset, focusing on predicting a continuous target variable using appropriate preprocessing, feature scaling, and hyperparameter tuning. The results indicated that the Random Forest model outperformed the other models, achieving the lowest RMSE (45.32) and the highest R² score (0.523), demonstrating its ability to capture complex relationships within the data. While Ridge and Lasso regressions showed competitive performances by utilizing regularization to reduce

overfitting, the simplicity of Linear Regression resulted in suboptimal performance. These findings highlight the importance of model selection, feature preprocessing, and regularization in regression analysis. Overall, the results suggest that ensemble methods like Random Forest are highly effective for capturing non-linear relationships in complex datasets.

8) References

1. Scikit-learn Documentation

Scikit-learn developers. (n.d.). *Scikit-learn: Machine Learning in Python*. Retrieved from <https://scikit-learn.org/stable/>

2. Diabetes Dataset Documentation - Scikit-learn

Scikit-learn developers. (n.d.). *Diabetes Dataset*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html

3. Seaborn Documentation

W. Michael B. (2020). *Seaborn: Statistical Data Visualization*. Retrieved from <https://seaborn.pydata.org/>

4. Matplotlib Documentation

Hunter, J.D. (2007). *Matplotlib: A 2D plotting library*. Retrieved from <https://matplotlib.org/>

5. GridSearchCV Documentation - Scikit-learn

Scikit-learn developers. (n.d.). *GridSearchCV*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

6. Random Forest Regressor - Scikit-learn

Scikit-learn developers. (n.d.). *Random Forest Regressor*. Retrieved from <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>

7. Ridge and Lasso Regression Documentation - Scikit-learn

Scikit-learn developers. (n.d.). *Ridge Regression* and *Lasso Regression*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html and https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

8. Python Data Science Handbook by Jake VanderPlas

VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media.