CSE 422 Lab Project Report

Topic: Student Exam score Prediction

Team members

Khandaker Ziaur Rahman	**24141125**
Sougata Chowdhury	**22201074**

Group No. - 15

Section - 15

Table of Contents

1. **Introduction**

The topic of "StudentPerformanceFactors" aims to analyze factors influencing students' academic success. Education shapes futures, and understanding these contributors can help educators and parents create supportive learning environments. Despite reforms, many students struggle to reach their potential.

This project identifies key factors like study habits, parental involvement, and school quality, uncovering actionable strategies to enhance student outcomes. By addressing root causes of poor performance, we aim to foster personalized education and equitable opportunities for success.

2. Dataset Description

Source: Kaggle.com

Link: https://www.kaggle.com/datasets/lainguyn123/student-performance-factors/data

Dataset Description:

The dataset comprises 20 features, which include a mix of quantitative and categorical data. The features are as follows: Hours_Studied, Attendance, Parental_Involvement, Access_to_Resources, Extracurricular_Activities, Sleep_Hours, Previous_Scores, Motivation_Level, Internet_Access, Tutoring_Sessions, Family_Income, Teacher_Quality, School_Type, Peer_Influence, Physical_Activity, Learning_Disabilities, Parental_Education_Level, Distance_from_Home, Gender, Exam_Score.

Classification or Regression Problem:

This is a regression problem since the target variable, Exam_Score, is a continuous numerical value. The primary goal is to predict the Exam_Score based on the input features, which aligns with the objectives of a regression model.

Number of Data Points:

The dataset contains N rows and 20 columns (replace N with the actual number after running the analysis). Each row represents a unique observation of student performance.

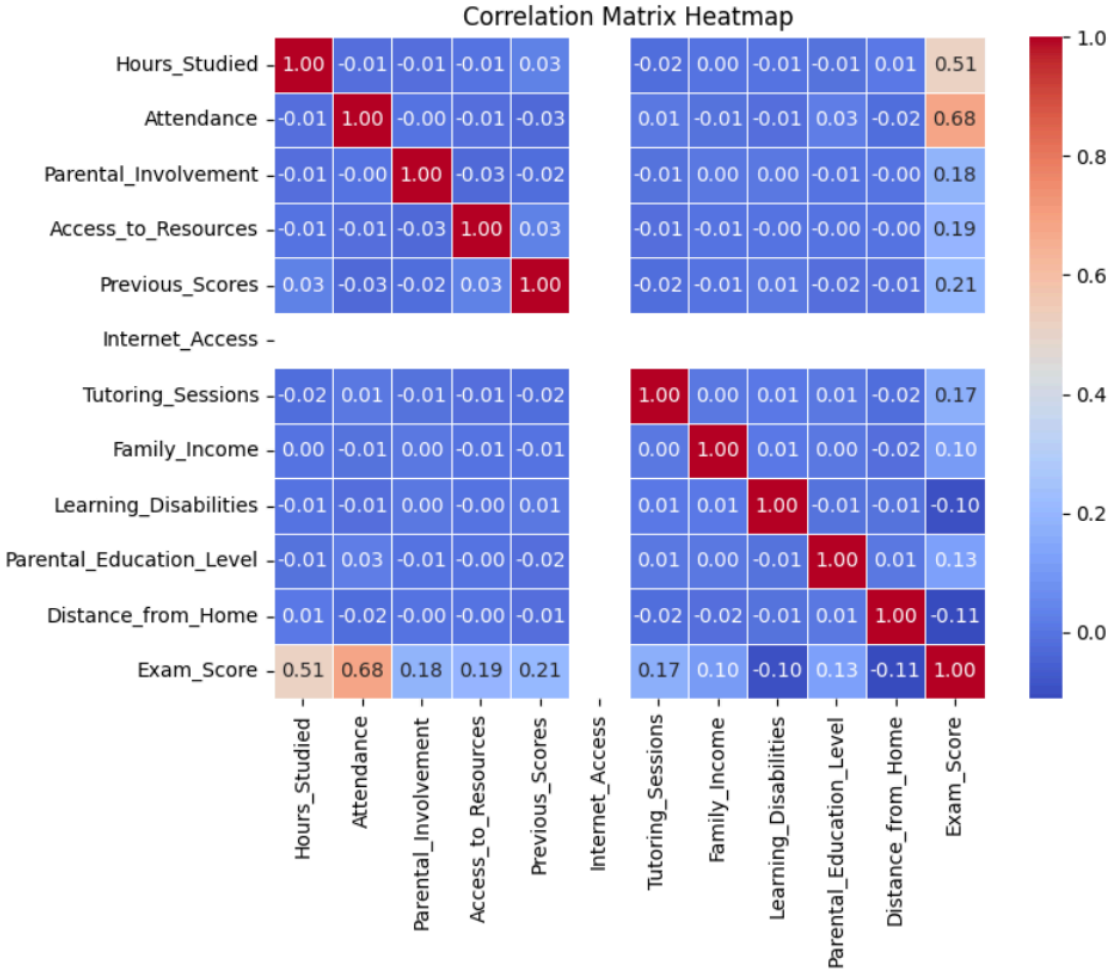
Types of Features:

- Quantitative Features: These include numerical data such as Hours_Studied, Attendance, Sleep_Hours, Previous_Scores, Motivation_Level, Family_Income, Distance_from_Home, and Exam_Score.

- Categorical Features: These include features such as Parental_Involvement, Internet_Access, School_Type, Gender, and Learning_Disabilities.

Heatmap:

A heatmap was generated to visualize the correlation of all features.



Imbalanced Dataset:

Since this is a regression problem and imbalance only occurs in classification problems, we did not have to handle such issues.

3. Dataset Preprocessing

Faults:

- Classified data were unsuitable for training regression models.

- Presence of outliers in the Exam_Score dataset that could skew statistical analyses and affect model accuracy.

- Some features in the dataset were not very useful or did not affect the Exam_Score.

Solutions:

- We assigned numeric values to classified data. For example, for School_Type, we assigned 0 for public and 1 for private.

```
print(df.isnull().sum())
print(df["Motivation_Level"].unique())


df["Parental_Involvement"] = df["Parental_Involvement"].map({"Low": 0, "Medium": 1, "High": 2})
df["Access_to_Resources"] = df["Access_to_Resources"].map({"Low": 0, "Medium": 1, "High": 2})
df["Extracurricular_Activities"] = df["Extracurricular_Activities"].map({"Yes": 1, "No": 0})
df["Motivation_Level"] = df["Motivation_Level"].map({"Low": 0, "Medium": 1, "High": 2})
df["Internet_Access"] = df["Internet_Access"].map({"Yes": 1, "No": 0})
df["Teacher_Quality"] = df["Teacher_Quality"].map({"Low": 0, "Medium": 1, "High": 2})
df["School_Type"] = df["School_Type"].map({"Public": 0, "Private": 1})
df["Peer_Influence"] = df["Peer_Influence"].map({"Positive": 0, "Negative": 1, "Neutral": 2})
df["Learning_Disabilities"] = df["Learning_Disabilities"].map({"Yes": 1, "No": 0})
df["Parental_Education_Level"] = df["Parental_Education_Level"].map({"High School": 0, "College": 1, "Postgraduate": 2})
df["Distance_from_Home"] = df["Distance_from_Home"].map({"Near": 0, "Moderate": 1, "Far": 2})
df["Gender"] = df["Gender"].map({"Male": 1, "Female": 0})
df["Family_Income"] = df["Family_Income"].map({"Low": 0, "Medium": 1, "High": 2})
```

- Outliers were handled using the Z-score method. Values beyond three standard deviations from the mean were removed, resulting in a cleaner dataset.

```python
# Removing outliers from dataset based on the Z-score method

z = np.abs((df - df.mean()) / df.std())
print(z)
threshold = 3
result = z < threshold
filtered = result.all(axis = 1)
df = df[filtered]
print(df)


```

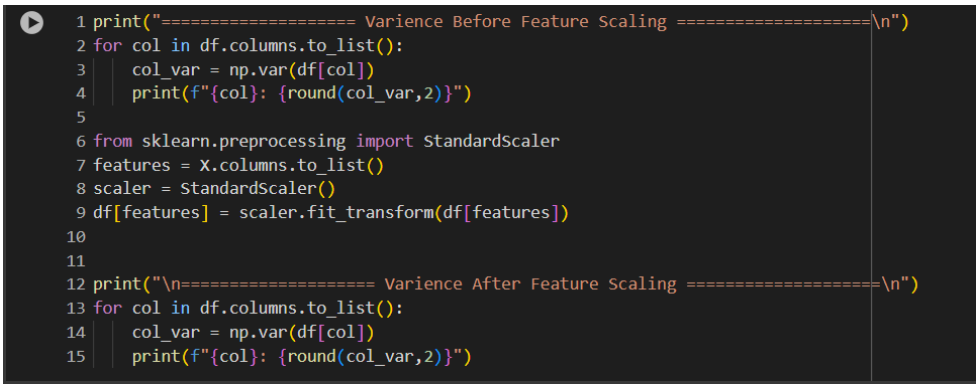
- Features that were not useful were removed before training the models.

We did so by removing columns with correlation less than 0.1, this ensured us better results as the information used to train the models are now much more relevant and useful to the model to yield better accuracy.

```python
correlation = df.corr()["Exam_Score"]
print(correlation)

low_correlation_columns = correlation[correlation.abs() <= 0.1].index

print(low_correlation_columns)
df = df.drop(columns = low_correlation_columns)
print(df.columns)



```

4. Feature Scaling

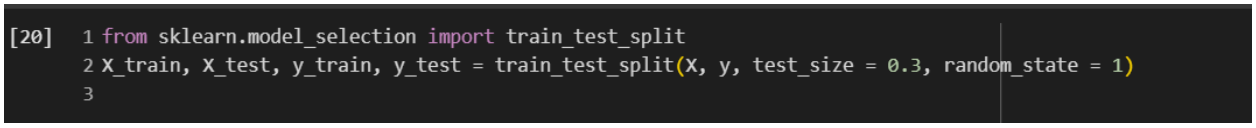
We applied Standard Scaling using StandardScaler to scale features. This transforms the data so that all features contribute equally to model training, especially when they have different ranges or units.



```
print("==================== Varience Before Feature Scaling ====================\n")
for col in df.columns.to_list():
    col_var = np.var(df[col])
    print(f"{col}: {round(col_var,2)}")

from sklearn.preprocessing import StandardScaler
features = X.columns.to_list()
scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])


print("\n==================== Varience After Feature Scaling ====================\n")
for col in df.columns.to_list():
    col_var = np.var(df[col])
    print(f"{col}: {round(col_var,2)}")
```

5. Data Splitting

We split the dataset in a 70:30 ratio as instructed. Seventy percent of the data was used for training, and 30% for testing. The data split was random, not stratified.

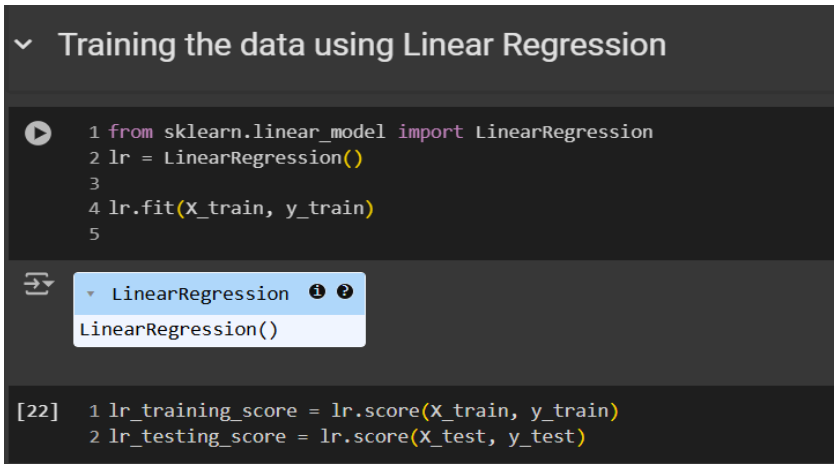


```
[20]  from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)
```

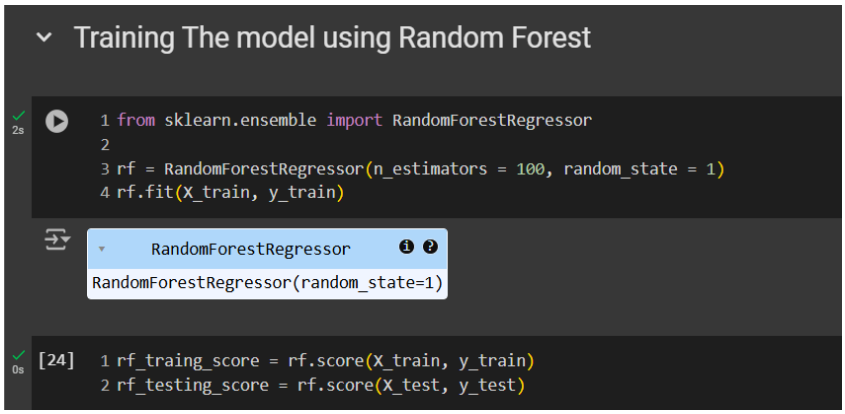
6. Model Training and Testing

We trained six models using this dataset. The models used are:

- Linear Regression



- Random Forest



- Decision Tree

Training The model using Decision Tree Regression

```
from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor(max_depth = 5, random_state = 1)
dtr.fit(X_train, y_train)
```

```
DecisionTreeRegressor(max_depth=5, random_state=1)
```

```
dtr_training_score = dtr.score(X_train, y_train)
dtr_testing_score = dtr.score(X_test, y_test)
```

- Gradient Boosting

Training the model using Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3)
gbr.fit(X_train, y_train)
```

```
GradientBoostingRegressor()
```

```
gbr_training_score = gbr.score(X_train, y_train)
gbr_testing_score = gbr.score(X_test, y_test)
```

- MLP Regressor

Training the model using MLP Regressor

```
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor(hidden_layer_sizes=(50,), max_iter=1000, random_state=42)
mlp.fit(X_train, y_train)

```

```
MLPRegressor(hidden_layer_sizes=(50,), max_iter=1000, random_state=42)
```

```
mlp_training_score = mlp.score(X_train, y_train)
mlp_testing_score = mlp.score(X_test, y_test)
```

- Lasso Regressor

Training the model using Lasso Regression

[31]
```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
```

```
Lasso
Lasso(alpha=0.1)
```

[32]
```
lasso_training_score = lasso.score(X_train, y_train)
lasso_testing_score = lasso.score(X_test, y_test)
```

7. Model Comparison Analysis

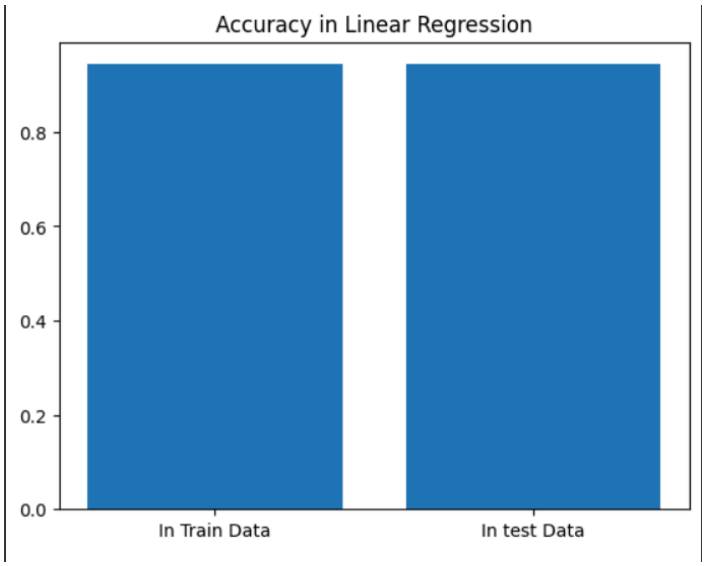
The performance of these models was evaluated using metrics such as RMSE, R^2 Score, and Mean Absolute Error. Detailed results are as follows:

Model Comparison and Analysis

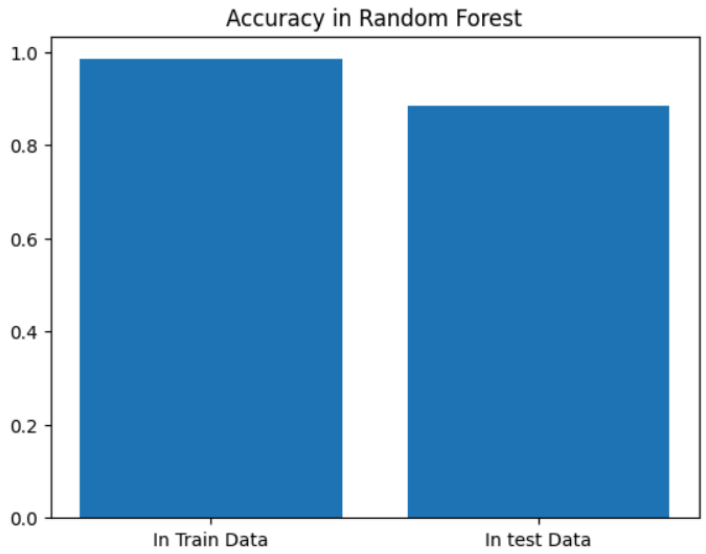
```
accuracy_train = [lr_training_score, rf_traing_score, dtr_training_score, gbr_training_score, mlp_training_score, lasso_training_score]
accuracy_test = [lr_testing_score, rf_testing_score, dtr_testing_score, gbr_testing_score, mlp_testing_score, lasso_testing_score]
label =  ["Linear Regression", "Random Forest", "Decision Tree ", "Gradient Boosting", "MLP Regressor", "Lasso Regressor"]

accuracy_show = pd.DataFrame({
    "Labels": label,
    "Accuracy in Trained Data": accuracy_train,
    "Accuracy in Test Data": accuracy_test
})

print(accuracy_show)
```

	Labels	Accuracy in Trained Data	Accuracy in Test Data
0	Linear Regression	0.943547	0.943103
1	Random Forest	0.984151	0.885262
2	Decision Tree	0.736358	0.680487
3	Gradient Boosting	0.940631	0.921983
4	MLP Regressor	0.941646	0.941605
5	Lasso Regressor	0.925217	0.924673

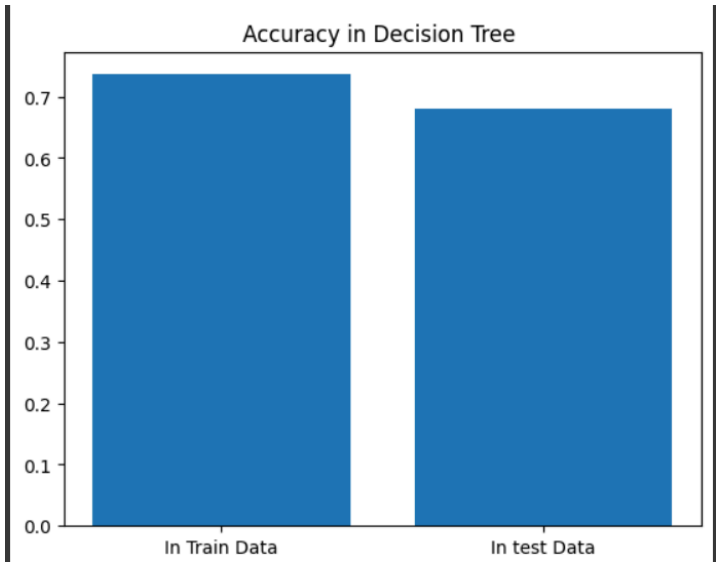
- Linear Regression:



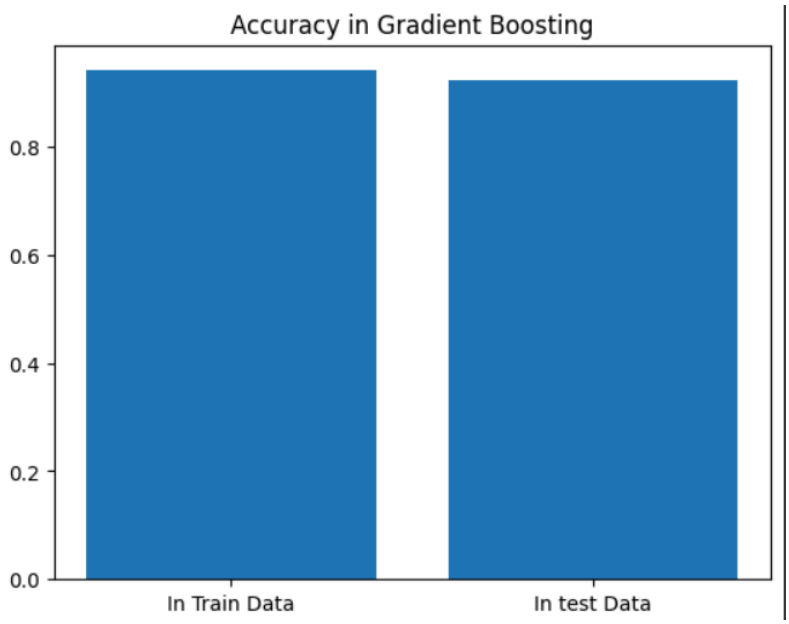
- Random Forest:



- Decision Tree:

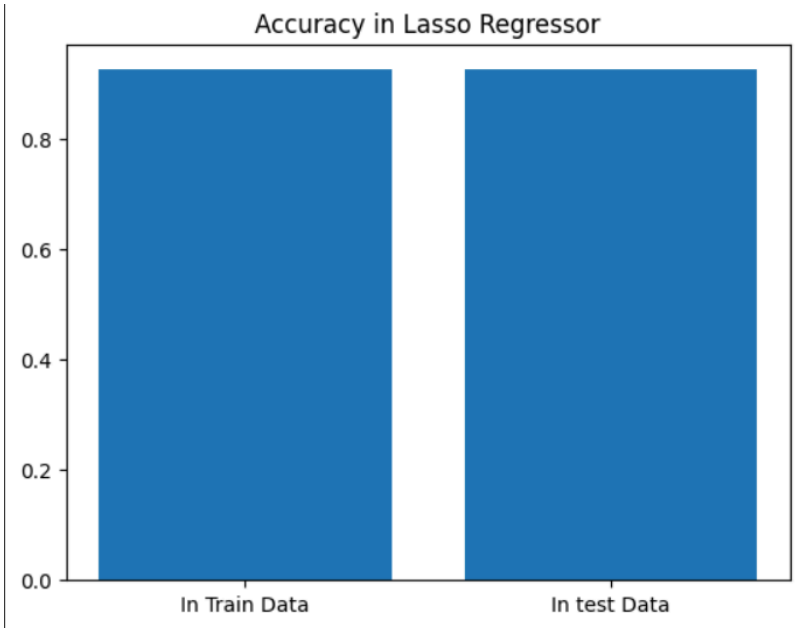


- Gradient Boosting:

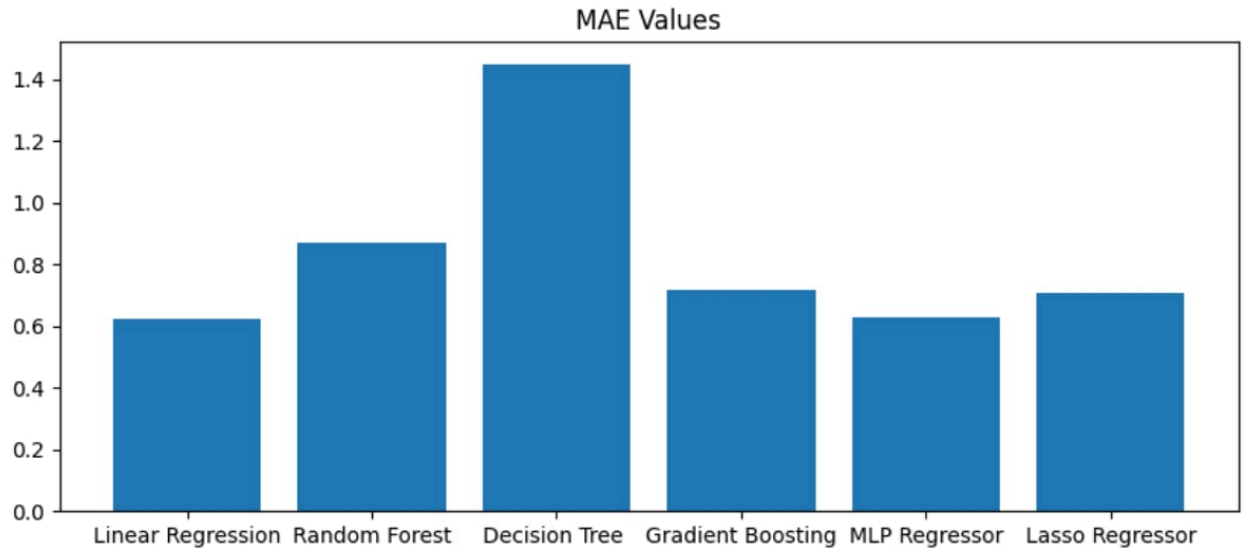


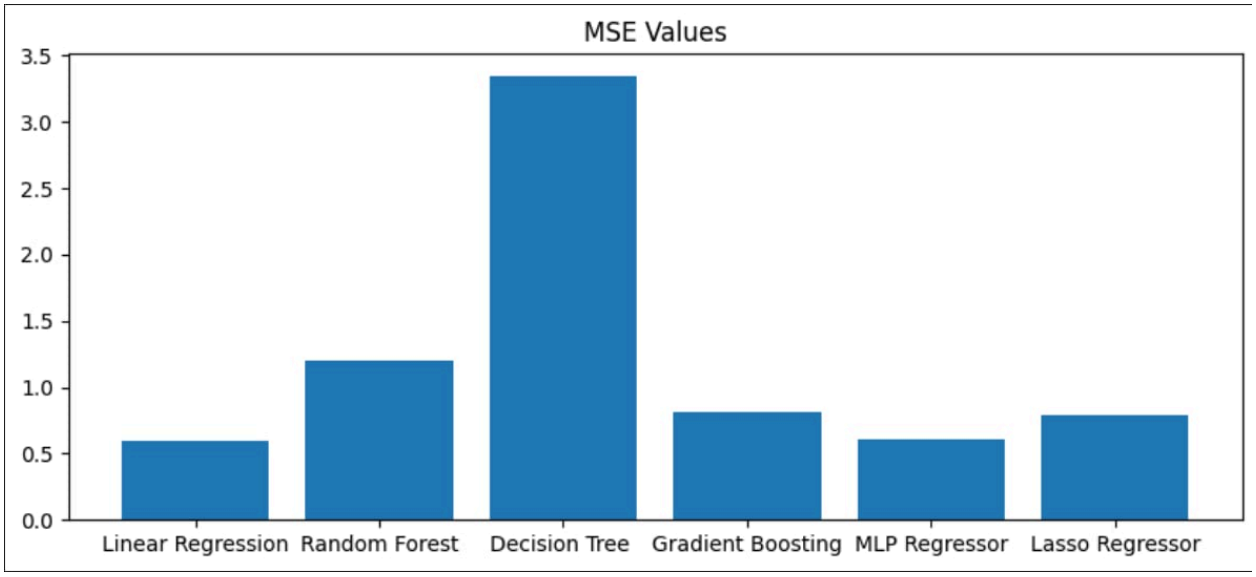
- MLP Regressor:

- Lasso Regressor:



MSC and MAE model comparison:





8. Conclusion

This project successfully analyzed the "StudentPerformanceFactors" dataset to identify key predictors of academic success. By addressing issues such as outliers and feature scaling, the dataset was optimized for regression modeling. The correlation analysis provided valuable insights into the relationships between features and the target variable, Exam_Score. Preprocessing steps like feature encoding, outlier removal, and standard scaling improved model reliability and accuracy. These steps allowed us to achieve highly accurate results. The findings highlight the potential for data-driven strategies to enhance student performance and guide personalized educational interventions.