```
import pandas as pd
import numpy as np
```

```
data = pd.read_csv("/content/sample_data/AMHM-ITAI-01.csv")
```

```
data.head(10)
```

| | Gender | Parental Education Level | Lunch Type | Test Preparation Course | Study Time | Absences | Math Score | Reading Score | Writing Score |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Female | Some College | Standard | Completed | 1 | 4 | 83 | 92 | 91 |
| **1** | Male | Some College | Standard | NaN | 1 | 6 | 94 | 97 | 93 |
| **2** | Female | Bachelor | Free/Reduced | NaN | 5 | 8 | 94 | 65 | 87 |
| **3** | Female | Some College | Free/Reduced | Completed | 3 | 4 | 95 | 91 | 90 |
| **4** | Female | Some College | Free/Reduced | Completed | 4 | 0 | 77 | 80 | 67 |
| **5** | Male | Bachelor | Standard | NaN | 3 | 9 | 98 | 75 | 98 |
| **6** | Female | Bachelor | Standard | NaN | 1 | 9 | 91 | 80 | 85 |
| **7** | Female | High School | Free/Reduced | Completed | 1 | 0 | 83 | 70 | 93 |
| **8** | Female | Master | Free/Reduced | Completed | 5 | 1 | 82 | 96 | 62 |
| **9** | Male | Master | Free/Reduced | Completed | 6 | 5 | 91 | 95 | 71 |

```
data['Test Preparation Course'].value_counts()
```

| Test Preparation Course | count |
|---|---|
| **Completed** | 58 |

**dtype:** int64

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Gender                    100 non-null    object
 1   Parental Education Level  100 non-null    object
 2   Lunch Type                100 non-null    object
 3   Test Preparation Course   58 non-null     object
 4   Study Time                100 non-null    int64
 5   Absences                  100 non-null    int64
 6   Math Score                100 non-null    int64
 7   Reading Score             100 non-null    int64
 8   Writing Score             100 non-null    int64
dtypes: int64(5), object(4)
memory usage: 7.2+ KB
```

```
data['Test Preparation Course'].fillna("Not Completed", inplace=True)
```

```
<ipython-input-68-cd69be179c18>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  data['Test Preparation Course'].fillna("Not Completed", inplace=True)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Gender                    100 non-null    object
```

```
 1   Parental Education Level  100 non-null    object
 2   Lunch Type                100 non-null    object
 3   Test Preparation Course   100 non-null    object
 4   Study Time                100 non-null    int64
 5   Absences                  100 non-null    int64
 6   Math Score                100 non-null    int64
 7   Reading Score             100 non-null    int64
 8   Writing Score             100 non-null    int64
dtypes: int64(5), object(4)
memory usage: 7.2+ KB
```

```
data.head()
```

| | Gender | Parental Education Level | Lunch Type | Test Preparation Course | Study Time | Absences | Math Score | Reading Score | Writing Score |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Female | Some College | Standard | Completed | 1 | 4 | 83 | 92 | 91 |
| **1** | Male | Some College | Standard | Not Completed | 1 | 6 | 94 | 97 | 93 |
| **2** | Female | Bachelor | Free/Reduced | Not Completed | 5 | 8 | 94 | 65 | 87 |
| **3** | Female | Some College | Free/Reduced | Completed | 3 | 4 | 95 | 91 | 90 |

```
from sklearn.preprocessing import LabelEncoder


le = LabelEncoder()
joblib.dump(le, 'Label_encoder.pkl')
data['Gender'] = le.fit_transform(data['Gender'])
data['Lunch Type'] = le.fit_transform(data['Lunch Type'])
data['Test Preparation Course'] = le.fit_transform(data['Test Preparation Course'])
data['Parental Education Level'] = le.fit_transform(data['Parental Education Level'])


l = ['Gender', 'Lunch Type','Test Preparation Course', 'Parental Education Level' ]
for i in l:
  data[i] = le.fit_transform(data[i])
```

```
data.head(10)
```

| | Gender | Parental Education Level | Lunch Type | Test Preparation Course | Study Time | Absences | Math Score | Reading Score | Writing Score |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 4 | 1 | 0 | 1 | 4 | 83 | 92 | 91 |
| **1** | 1 | 4 | 1 | 1 | 1 | 6 | 94 | 97 | 93 |
| **2** | 0 | 1 | 0 | 1 | 5 | 8 | 94 | 65 | 87 |
| **3** | 0 | 4 | 0 | 0 | 3 | 4 | 95 | 91 | 90 |
| **4** | 0 | 4 | 0 | 0 | 4 | 0 | 77 | 80 | 67 |
| **5** | 1 | 1 | 1 | 1 | 3 | 9 | 98 | 75 | 98 |
| **6** | 0 | 1 | 1 | 1 | 1 | 9 | 91 | 80 | 85 |
| **7** | 0 | 2 | 0 | 0 | 1 | 0 | 83 | 70 | 93 |
| **8** | 0 | 3 | 0 | 0 | 5 | 1 | 82 | 96 | 62 |

```
data['Parental Education Level'].value_counts()
```

| Parental Education Level | count |
|---|---|
| 2 | 24 |
| 1 | 22 |
| 3 | 20 |
| 0 | 19 |
| 4 | 15 |

dtype: int64

```
data.corr()
```

| | Gender | Parental Education Level | Lunch Type | Test Preparation Course | Study Time | Absences | Math Score | Reading Score | Writing Score |
|---|---|---|---|---|---|---|---|---|---|
| **Gender** | 1.000000 | -0.127196 | 0.125809 | 0.101226 | 0.120680 | 0.149392 | 0.014248 | -0.078102 | 0.158505 |
| **Parental Education Level** | -0.127196 | 1.000000 | -0.063189 | -0.088329 | -0.099894 | -0.042461 | -0.009186 | 0.135249 | -0.170049 |
| **Lunch Type** | 0.125809 | -0.063189 | 1.000000 | 0.155729 | 0.065908 | 0.050477 | -0.030741 | 0.090363 | 0.222691 |
| **Test Preparation Course** | 0.101226 | -0.088329 | 0.155729 | 1.000000 | -0.088149 | 0.017832 | 0.011625 | -0.061919 | 0.091675 |
| **Study Time** | 0.120680 | -0.099894 | 0.065908 | -0.088149 | 1.000000 | 0.110486 | -0.038316 | 0.017448 | -0.102173 |
| **Absences** | 0.149392 | -0.042461 | 0.050477 | 0.017832 | 0.110486 | 1.000000 | 0.014253 | -0.091829 | 0.058257 |
| **Math Score** | 0.014248 | -0.009186 | -0.030741 | 0.011625 | -0.038316 | 0.014253 | 1.000000 | 0.078773 | -0.040784 |
| **Reading Score** | -0.078102 | 0.135249 | 0.090363 | -0.061919 | 0.017448 | -0.091829 | 0.078773 | 1.000000 | -0.000772 |
| **Writing Score** | 0.158505 | -0.170049 | 0.222691 | 0.091675 | -0.102173 | 0.058257 | -0.040784 | -0.000772 | 1.000000 |

```
from sklearn.preprocessing import MinMaxScaler
import joblib
scaler = MinMaxScaler()
joblib.dump(scaler, 'Feature_scaler.pkl')
data[['Study Time', 'Absences']] = scaler.fit_transform(data[['Study Time', 'Absences']])
```

```
data.head()
```

| | Gender | Parental Education Level | Lunch Type | Test Preparation Course | Study Time | Absences | Math Score | Reading Score | Writing Score |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 4 | 1 | 0 | 0.000000 | 0.4 | 83 | 92 | 91 |
| **1** | 1 | 4 | 1 | 1 | 0.000000 | 0.6 | 94 | 97 | 93 |
| **2** | 0 | 1 | 0 | 1 | 0.444444 | 0.8 | 94 | 65 | 87 |
| **3** | 0 | 4 | 0 | 0 | 0.222222 | 0.4 | 95 | 91 | 90 |

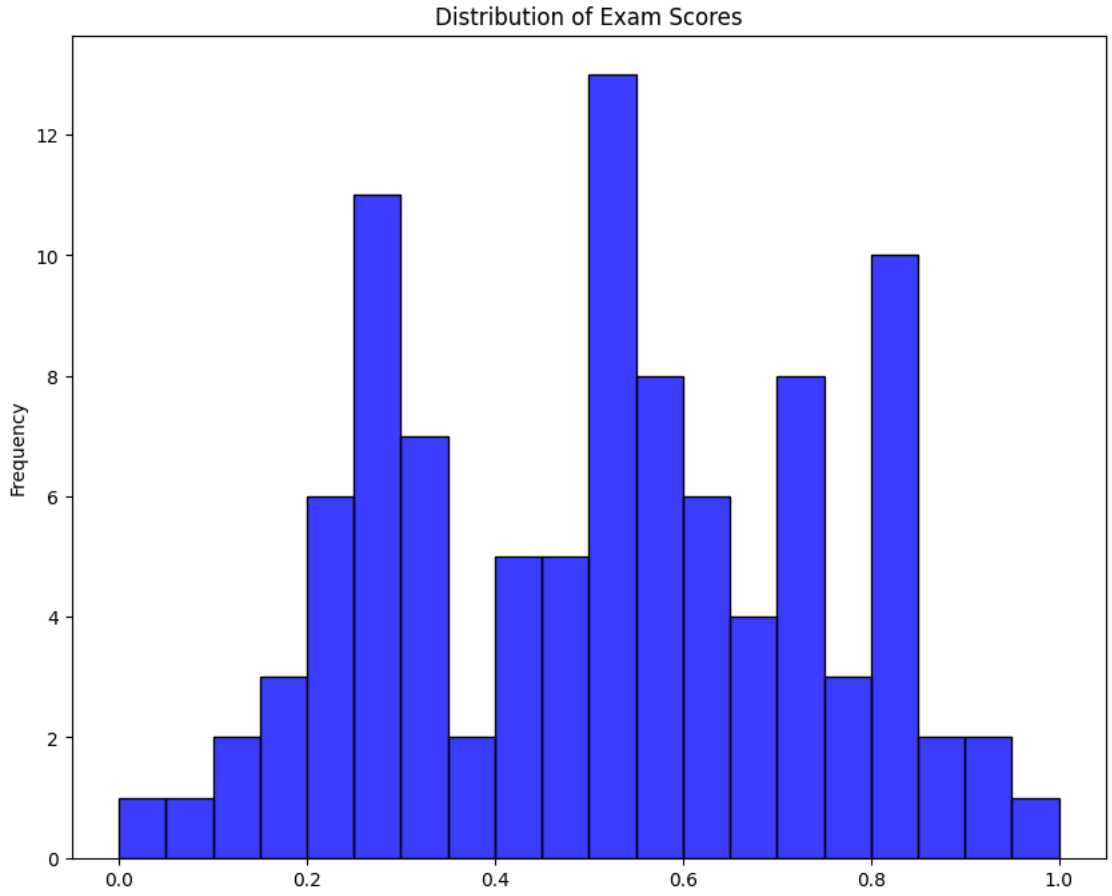
```
data.describe()
```

| | Gender | Parental Education Level | Lunch Type | Test Preparation Course | Study Time | Absences | Math Score | Reading Score | Writing Score |
|---|---|---|---|---|---|---|---|---|---|
| count | 100.000000 | 100.000000 | 100.000000 | 100.000000 | 100.000000 | 100.000000 | 100.000000 | 100.000000 | 100.000000 |
| mean | 0.560000 | 1.900000 | 0.480000 | 0.420000 | 0.497778 | 0.479000 | 80.890000 | 79.620000 | 79.410000 |
| std | 0.498888 | 1.337116 | 0.502117 | 0.496045 | 0.337785 | 0.322019 | 11.595798 | 11.852443 | 11.501511 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 60.000000 | 60.000000 | 60.000000 |
| 25% | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.222222 | 0.200000 | 71.000000 | 69.750000 | 70.000000 |
| 50% | 1.000000 | 2.000000 | 0.000000 | 0.000000 | 0.500000 | 0.400000 | 81.000000 | 79.500000 | 80.000000 |
| 75% | 1.000000 | 3.000000 | 1.000000 | 1.000000 | 0.777778 | 0.800000 | 90.250000 | 90.250000 | 89.000000 |

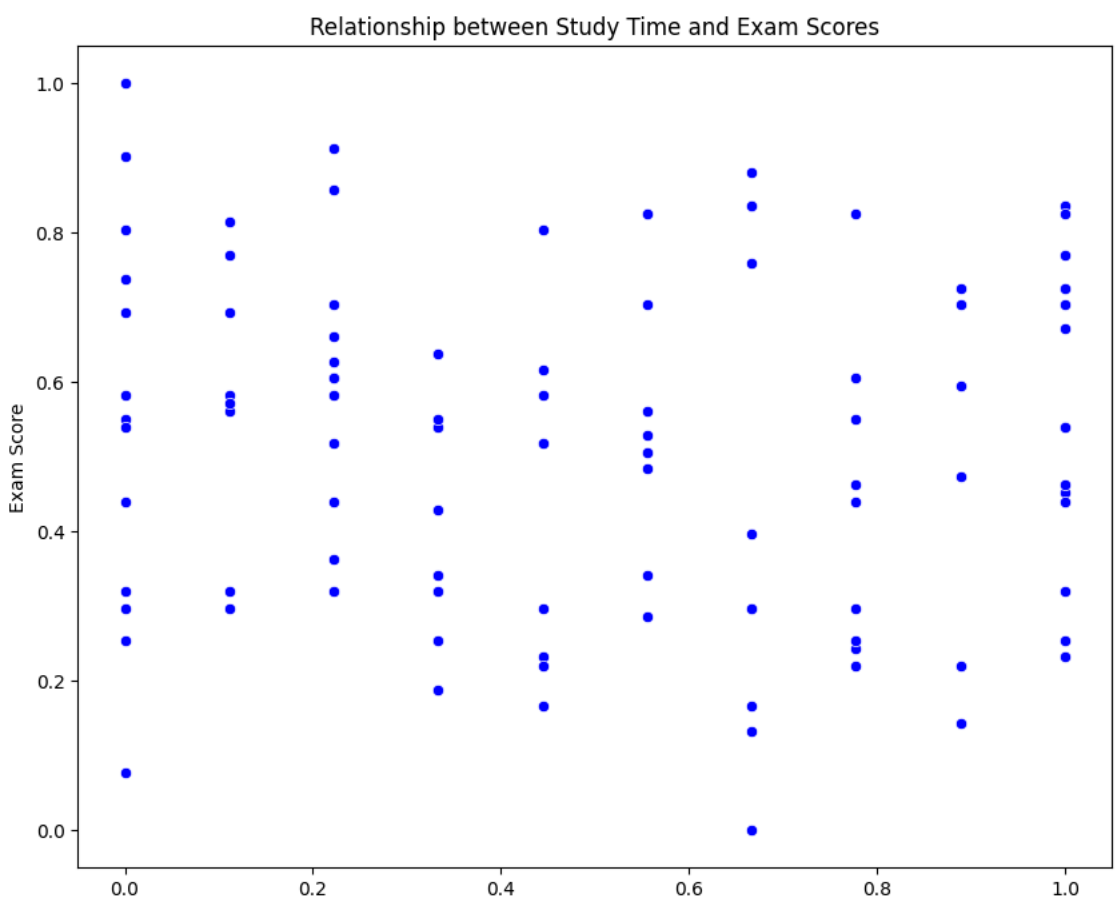
```
data['exam_scores'] = data['Math Score'] + data['Reading Score'] + data['Writing Score']
scaler = MinMaxScaler()
data['exam_scores'] = scaler.fit_transform(data[['exam_scores']])
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

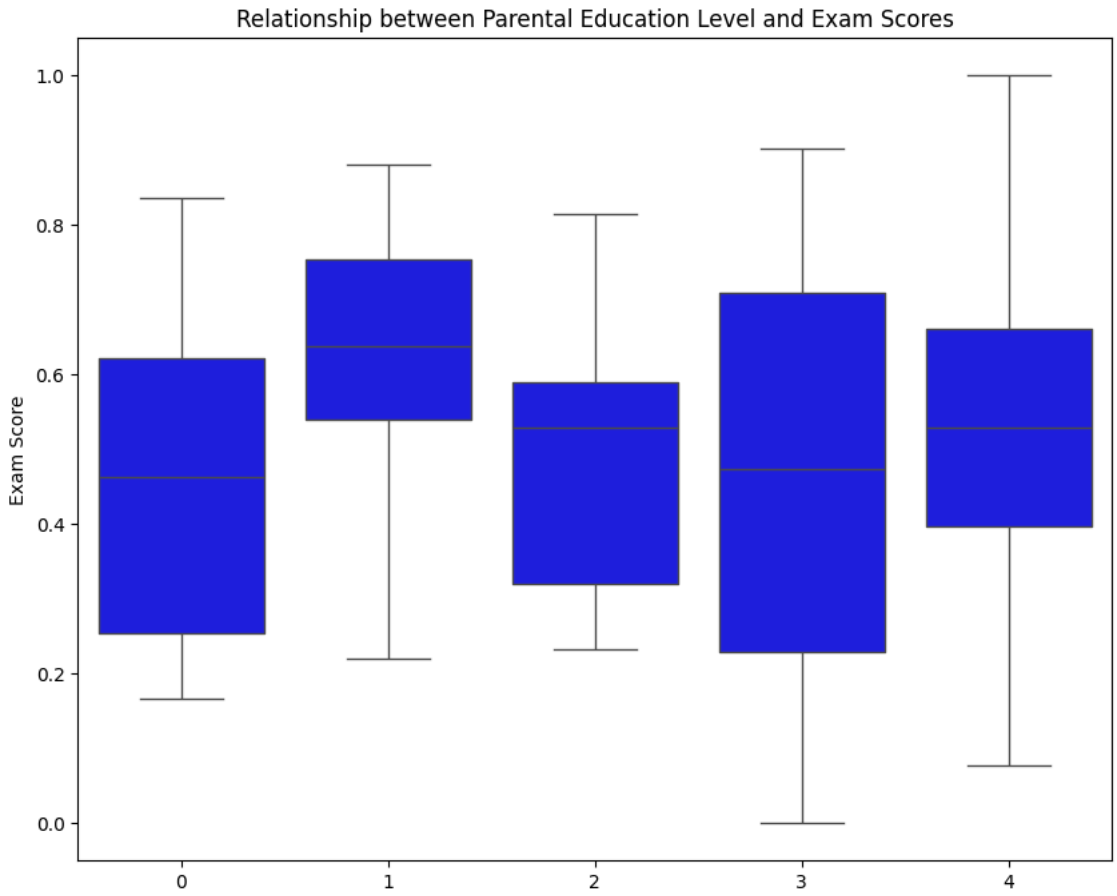
```
plt.figure(figsize=(10, 8))
sns.histplot(data['exam_scores'], kde=False, bins=20, color='blue')
plt.title('Distribution of Exam Scores')
plt.xlabel('Exam Score')
plt.ylabel('Frequency')
plt.show()
```



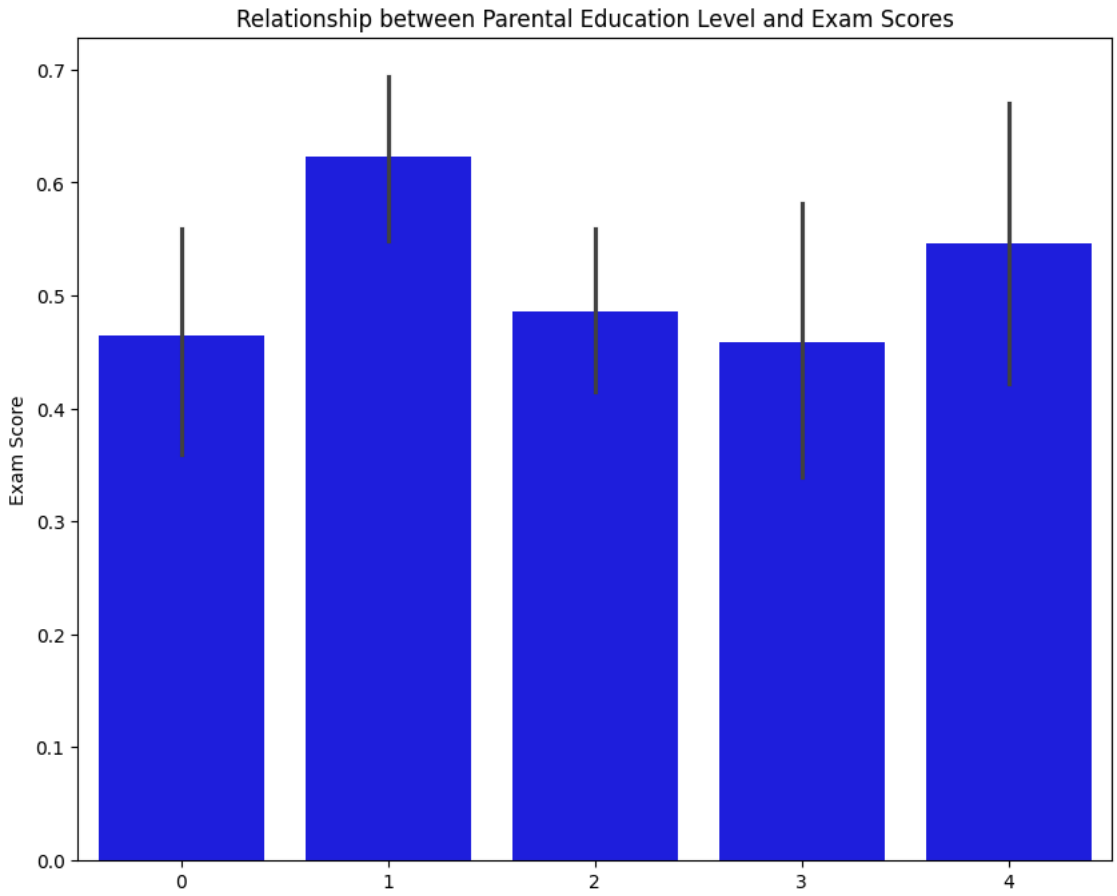
```
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Study Time', y='exam_scores', data=data, color='blue')
plt.title('Relationship between Study Time and Exam Scores')
plt.xlabel('Study Time')
plt.ylabel('Exam Score')
plt.show()
```




```
plt.figure(figsize=(10, 8))
sns.boxplot(x='Parental Education Level', y='exam_scores', data=data, color='blue')
plt.title('Relationship between Parental Education Level and Exam Scores')
plt.xlabel('Parental Education Level')
plt.ylabel('Exam Score')
plt.show()
```




```
plt.figure(figsize=(10, 8))
sns.barplot(x='Parental Education Level', y='exam_scores', data=data, color='blue')
plt.title('Relationship between Parental Education Level and Exam Scores')
plt.xlabel('Parental Education Level')
plt.ylabel('Exam Score')
plt.show()
```




```
from sklearn.model_selection import train_test_split


x = data.drop(['exam_scores', 'Math Score', 'Reading Score', 'Writing Score'], axis=1)
y = data['exam_scores']
target_scaler = MinMaxScaler()
joblib.dump(target_scaler, 'Target_scaler.pkl')
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
x
```

| | Gender | Parental Education Level | Lunch Type | Test Preparation Course | Study Time | Absences |
|---|---|---|---|---|---|---|
| **0** | 0 | 4 | 1 | 0 | 0.000000 | 0.4 |
| **1** | 1 | 4 | 1 | 1 | 0.000000 | 0.6 |
| **2** | 0 | 1 | 0 | 1 | 0.444444 | 0.8 |
| **3** | 0 | 4 | 0 | 0 | 0.222222 | 0.4 |
| **4** | 0 | 4 | 0 | 0 | 0.333333 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **95** | 1 | 2 | 1 | 1 | 1.000000 | 0.8 |
| **96** | 1 | 0 | 1 | 1 | 0.666667 | 0.3 |
| **97** | 1 | 0 | 0 | 0 | 1.000000 | 0.9 |
| **98** | 1 | 2 | 1 | 1 | 0.444444 | 0.4 |
| **99** | 0 | 3 | 1 | 0 | 1.000000 | 0.8 |

100 rows × 6 columns

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor


lr = LinearRegression()
lr.fit(x_train, y_train)
```

```
dt = DecisionTreeRegressor()
dt.fit(x_train, y_train)
rf = RandomForestRegressor()
rf.fit(x_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error


y_pred_lr = lr.predict(x_test)
y_pred_dt = dt.predict(x_test)
y_pred_rf = rf.predict(x_test)


print("Linear regssion MSE: ", mean_squared_error(y_test, y_pred_lr))
print("Descion Tree MSE: ", mean_squared_error(y_test, y_pred_dt))
print("Random Forest MSE: ", mean_squared_error(y_test, y_pred_rf))
```

```
Linear regssion MSE:  0.05418884766835943
Descion Tree MSE:  0.07836010143702452
Random Forest MSE:  0.06190760563638631
```

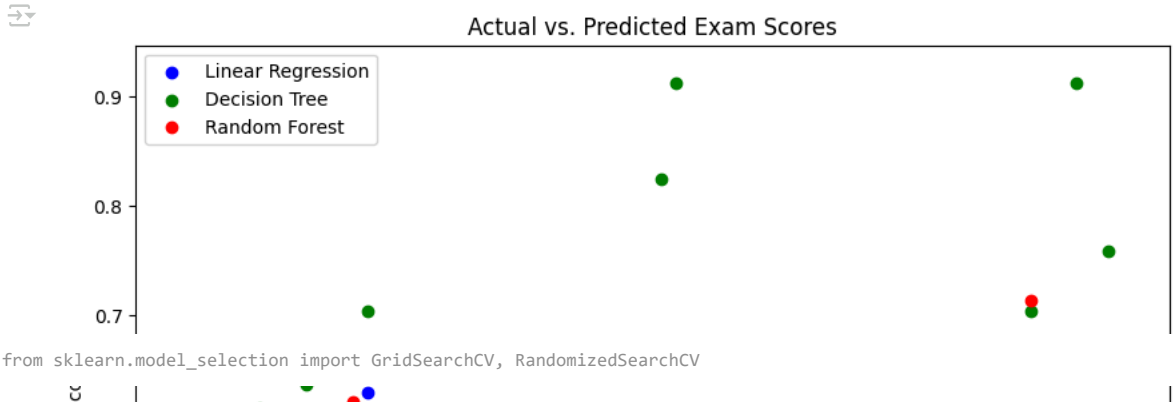
```
print("Linear Regression Accuracy: ", r2_score(y_test, y_pred_lr))
print("Decision Tree Accuracy: ", r2_score(y_test, y_pred_dt))
print("Random Forest Accuracy: ", r2_score(y_test, y_pred_rf))
```

```
Linear Regression Accuracy:  -0.12092685998759611
Decision Tree Accuracy:  -0.6209228694006781
Random Forest Accuracy:  -0.28059371957938173
```

```
print("Mean Absolute Error Linear regression:", mean_absolute_error(y_test, y_pred_lr))
print("Mean Absolute Error Descion tree:", mean_absolute_error(y_test, y_pred_dt))
print("Mean Absolute Error random forest:", mean_absolute_error(y_test, y_pred_rf))
```

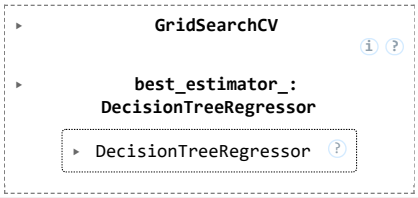
```
Mean Absolute Error Linear regression: 0.19843935000805174
Mean Absolute Error Descion tree: 0.2373626373626374
Mean Absolute Error random forest: 0.20728554421768708
```

```
plt.figure(figsize=(10, 8))
plt.scatter(y_test, y_pred_lr, color='blue', label='Linear Regression')
plt.scatter(y_test, y_pred_dt, color='green', label='Decision Tree')
plt.scatter(y_test, y_pred_rf, color='red', label='Random Forest')
plt.xlabel('Actual Exam Scores')
plt.ylabel('Predicted Exam Scores')
plt.title('Actual vs. Predicted Exam Scores')
plt.legend()
plt.show()
```




```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

```
# using gridsearchcv
dt_param = {'max_depth': [3, 5, 10, None], 'max_leaf_nodes': [2,5,7]}
grid_model = GridSearchCV(DecisionTreeRegressor(), dt_param, cv=5)
grid_model.fit(x_train, y_train)
```




```
print("best paramters is: ", grid_model.best_params_)
```

```
best paramters is:  {'max_depth': 5, 'max_leaf_nodes': 2}
```

```
y_predict_grid = grid_model.best_estimator_.predict(x_test)
print("Mean Absoulte error for Grisearch is:", mean_absolute_error(y_test,y_predict_grid ))
```

```
Mean Absoulte error for Grisearch is: 0.19962225274725276
```

```
rf_params = {'n_estimators': [50, 100, 200], 'max_depth': [3, 5,10]}
random_model = RandomizedSearchCV(RandomForestRegressor(),rf_params, scoring = 'neg_mean_absolute_error', random_state=42)
random_model.fit(x_train, y_train)
joblib.dump(random_model, 'Random_model.pkl')
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:317: UserWarning: The total space of parameters 9 is smaller
  warnings.warn(
['Random_model.pkl']
```

```
print('Randomized bes paramters is:', random_model.best_params_)
```

```
Randomized bes paramters is: {'n_estimators': 50, 'max_depth': 5}
```

```
y_predict_rf = random_model.best_estimator_.predict(x_test)
```