

In [51]:



Import necessary libraries

```
In [38]: import pandas as pd
import numpy as np
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
import seaborn as sns
```

Reading dataset

```
In [39]: data = pd.read_csv("WineQT.csv")
data.head()
```

Out[39]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	5.0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	5.0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	5.0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	5.0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	5.0

In [40]: `data.describe()`

Out[40]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	quality
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45.017347	0.997545	3.387143	0.616579	5.041129
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.074964	0.001818	0.109605	0.185481	0.715854
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.996000	2.940000	0.450000	3.000000
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000	0.997000	3.190000	0.550000	4.000000
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000	0.997500	3.300000	0.600000	5.000000
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000	0.998000	3.490000	0.680000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289.000000	1.000000	4.010000	0.950000	9.000000

Define features and target variable

In [41]: `X = data.drop(['quality', 'Id'], axis=1)`
`y = data['quality']`

Scaling features (X)

In [42]: `from sklearn.preprocessing import StandardScaler`
`scaler = StandardScaler()`
`X = scaler.fit_transform(X)`

Splitting data

In [43]: `# Split data into training and testing sets`
`X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_s`

Evaluation models

```
In [44]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

def evaluate_model(model_name, y_true, y_pred):
    """
    Evaluate a regression model using MAE, MSE, RMSE, and R^2 score.
    Additionally, plot y_true vs. y_pred with a reference line and display model na

    Parameters:
        model_name (str): Name of the regression model.
        y_true (array-like): True target values.
        y_pred (array-like): Predicted target values.

    Returns:
        dict: Dictionary containing MAE, MSE, RMSE, and R^2 score.
    """
    # Calculate metrics
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)

    # Print the evaluation results
    results = {
        "Mean Absolute Error (MAE)": mae,
        "Mean Squared Error (MSE)": mse,
        "Root Mean Squared Error (RMSE)": rmse,
        "R^2 Score": r2
    }

    # Plotting
    plt.figure(figsize=(8, 6))
    plt.scatter(y_true, y_pred, alpha=0.6, color='blue', label='Predicted vs Actual')
    plt.plot([min(y_true), max(y_true)], [min(y_true), max(y_true)], 'r--', label='True Values')
    plt.xlabel('True Values')
    plt.ylabel('Predicted Values')
    plt.title(f'Regression Model: {model_name} - True vs Predicted Values')
    plt.text(0.05, 0.9, f'R^2 Score: {r2:.2f}', transform=plt.gca().transAxes, font
    plt.legend()
    plt.grid(True)
    plt.show()

    return results
```

RandomForestRegressor

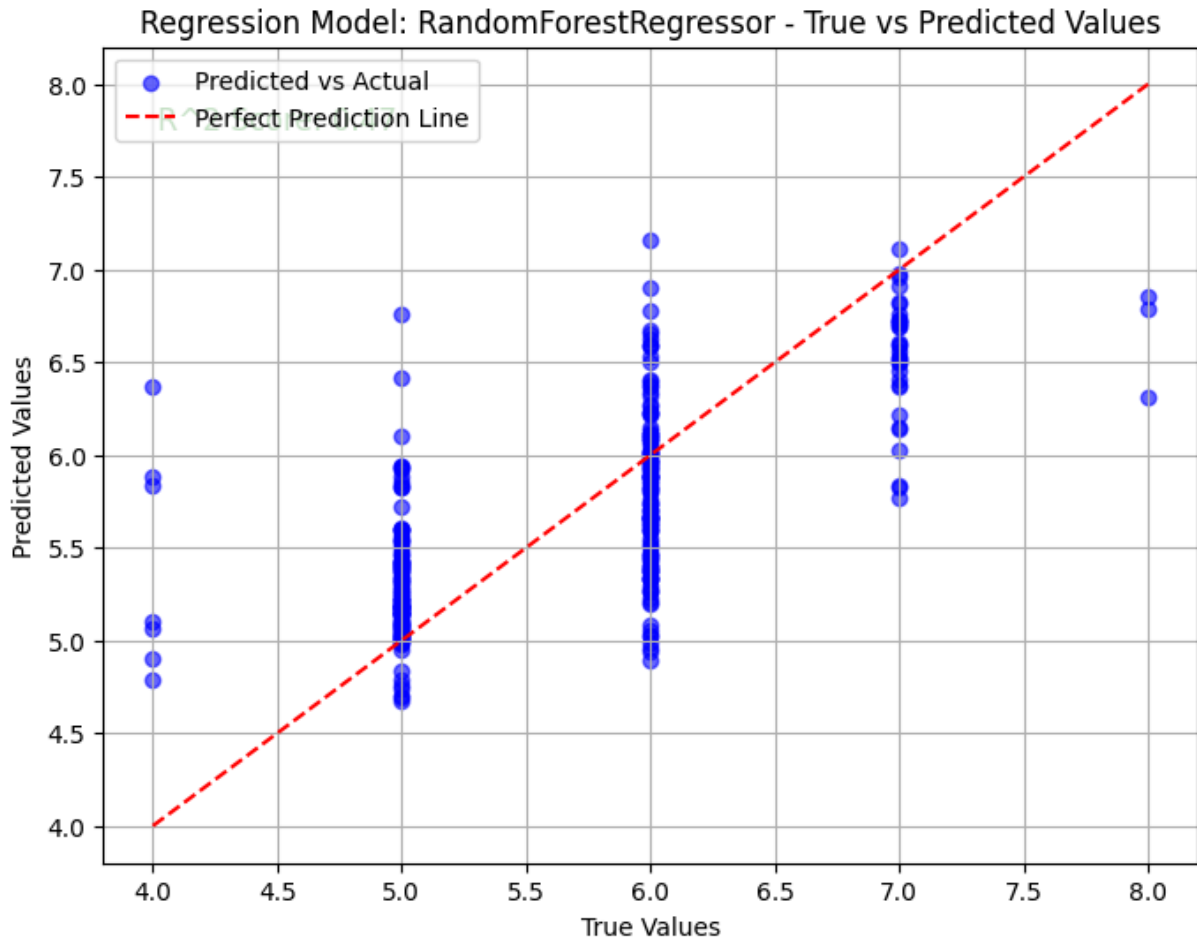
```
In [45]: rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
# Train the model
rf_regressor.fit(X_train, y_train)
```

```

# Make predictions
y_pred = rf_regressor.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestRegressor', y_test, y_pred)

# Print the evaluation results
evaluation_results

```



```

Out[45]: {'Mean Absolute Error (MAE)': 0.41363636363636364,
'Mean Squared Error (MSE)': 0.30136993006993007,
'Root Mean Squared Error (RMSE)': 0.5489717024309451,
'R^2 Score': 0.47265258744250727}

```

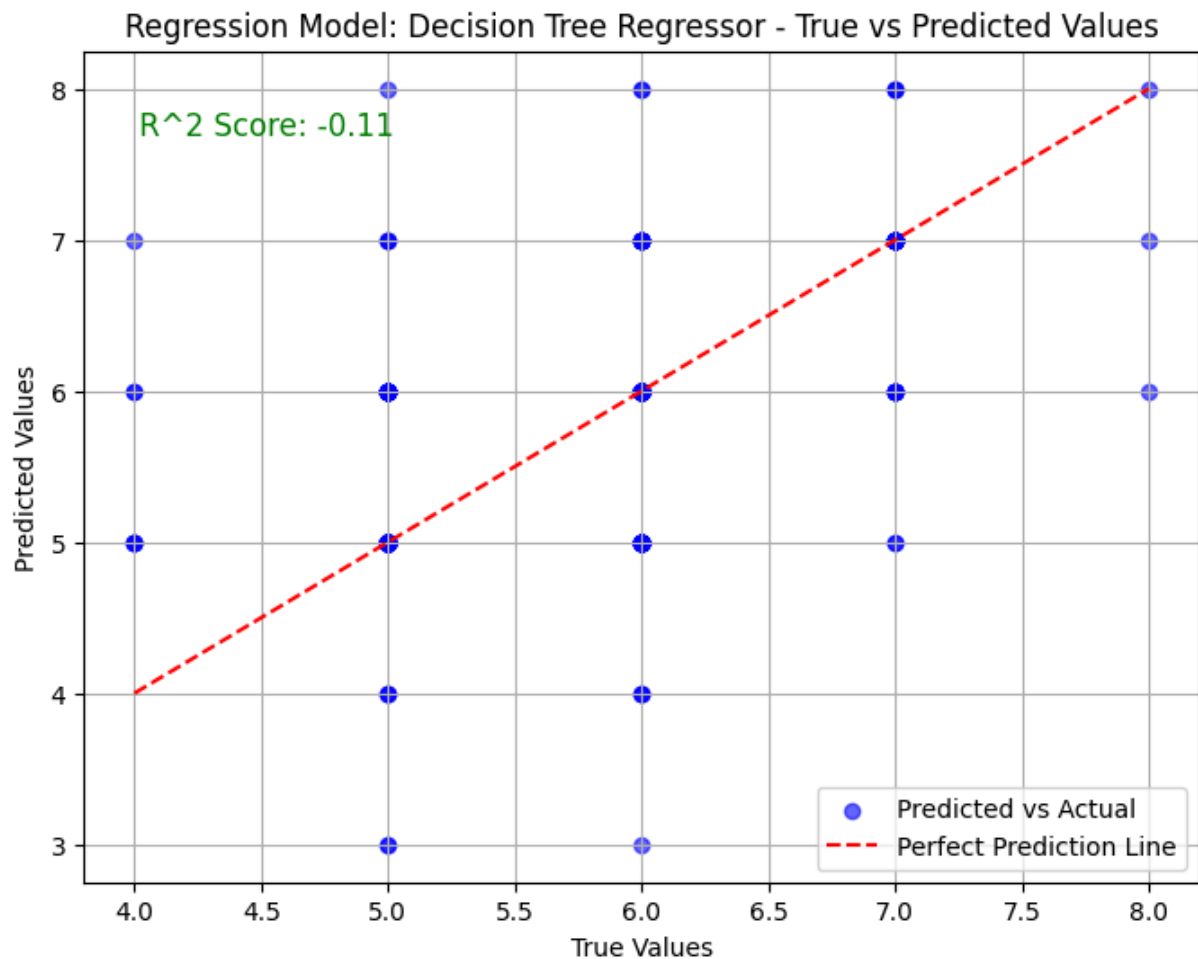
Decision Tree Regressor

```

In [46]: DT = DecisionTreeRegressor(random_state=42)
# Train the model
DT.fit(X_train, y_train)
# Make predictions
y_pred = DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Decision Tree Regressor', y_test, y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")

```

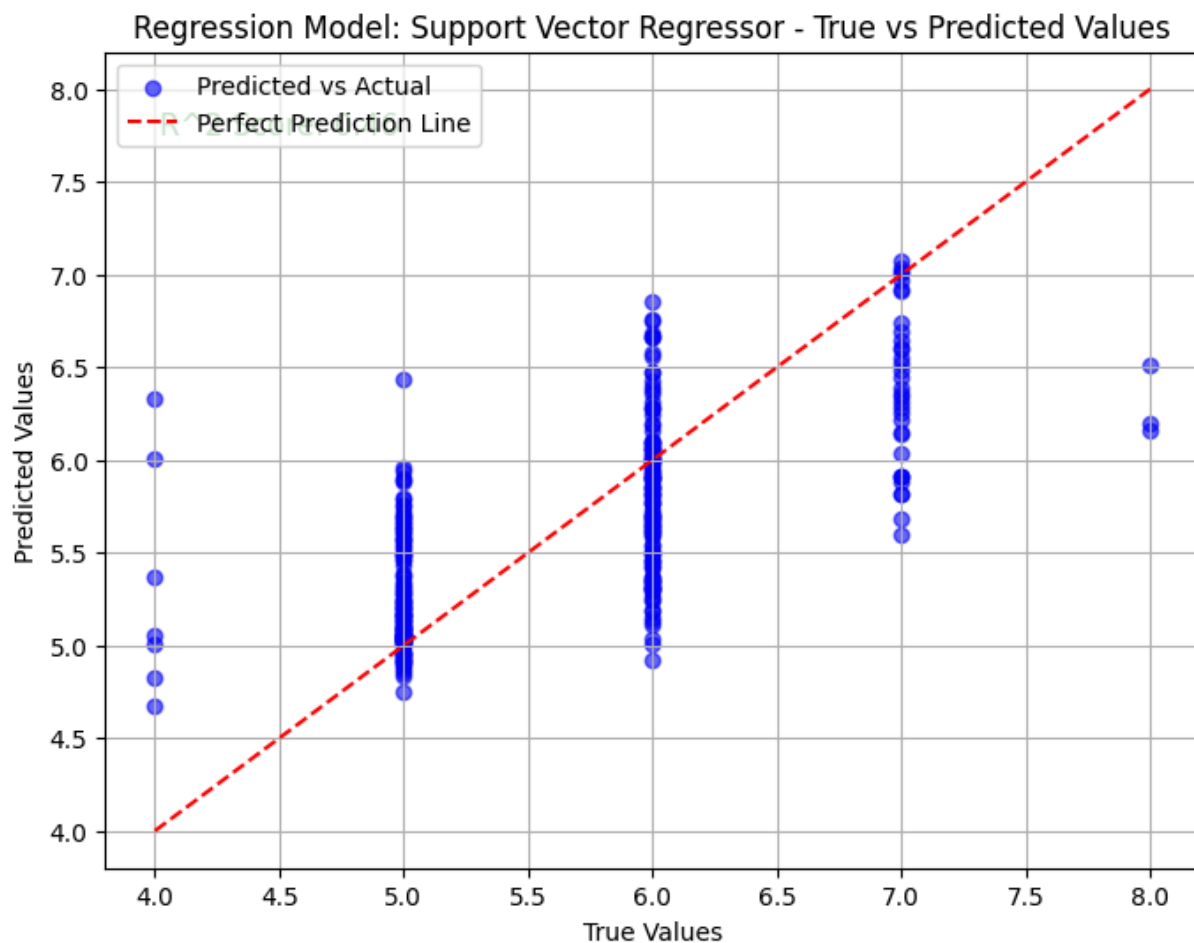


Mean Absolute Error (MAE): 0.4650
Mean Squared Error (MSE): 0.6329
Root Mean Squared Error (RMSE): 0.7955
 R^2 Score: -0.1074

Support Vector Regressor (SVR)

```
In [47]: SVM = SVR()
# Train the model
SVM.fit(X_train, y_train)
# Make predictions
y_pred = SVM.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Regressor', y_test, y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")
```

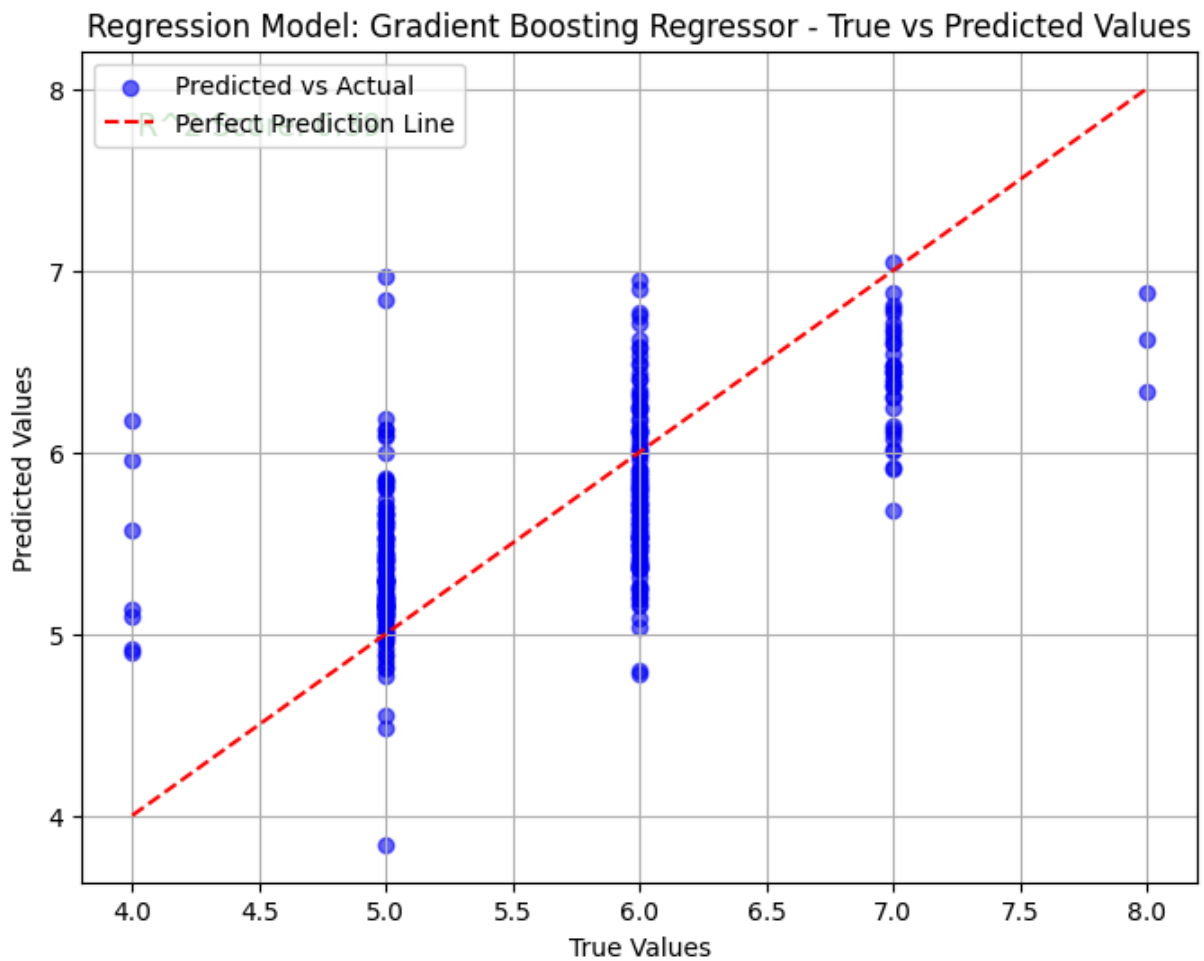


Mean Absolute Error (MAE): 0.4091
Mean Squared Error (MSE): 0.3098
Root Mean Squared Error (RMSE): 0.5566
R² Score: 0.4579

Gradient Boosting Regressor

```
In [48]: GBR = GradientBoostingRegressor(random_state=42)
# Train the model
GBR.fit(X_train, y_train)
# Make predictions
y_pred = GBR.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Gradient Boosting Regressor', y_test, y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")
```



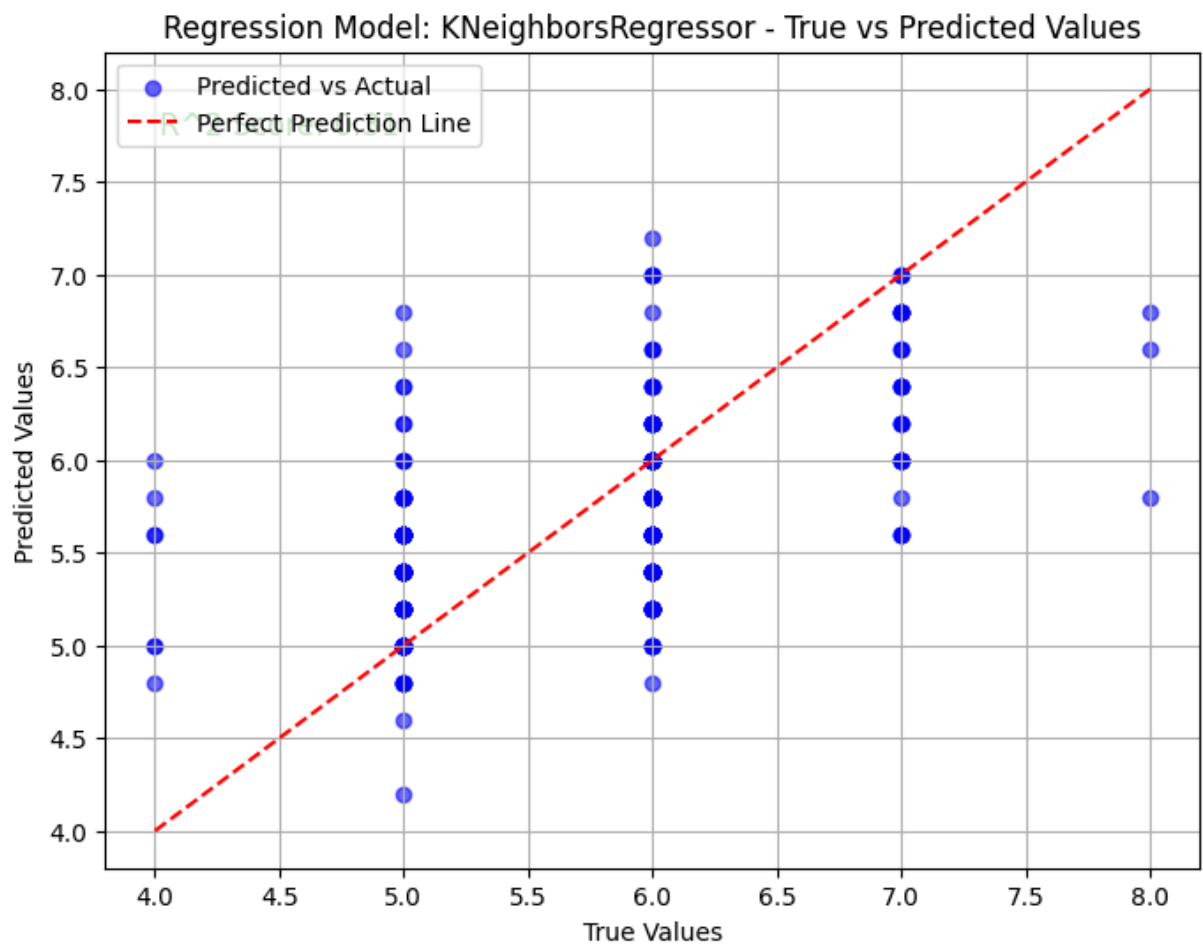
Mean Absolute Error (MAE): 0.4636
Mean Squared Error (MSE): 0.3474
Root Mean Squared Error (RMSE): 0.5894
R² Score: 0.3922

KNeighborsRegressor

```
In [49]: knn = KNeighborsRegressor()

knn.fit(X_train, y_train)
# Make predictions
y_pred = knn.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('KNeighborsRegressor', y_test, y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")
```



Mean Absolute Error (MAE): 0.4734
Mean Squared Error (MSE): 0.3971
Root Mean Squared Error (RMSE): 0.6301
R² Score: 0.3052

In []: