

```
In [29]: # import necessary libraries
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_s

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

```
In [30]: df = pd.read_csv('student_productivity_distraction_dataset_20000.csv')
```

```
In [31]: df.head(10)
```

```
Out[31]:
```

	student_id	age	gender	study_hours_per_day	sleep_hours	phone_usage_hour
0	1	23	Female	4.35	3.63	3.3
1	2	20	Male	6.14	6.58	5.4
2	3	29	Female	4.98	3.26	4.8
3	4	27	Female	3.19	4.58	10.0
4	5	24	Male	7.67	6.21	3.0
5	6	29	Other	7.18	3.52	4.0
6	7	21	Female	9.06	6.36	11.4
7	8	23	Female	6.37	4.86	3.3
8	9	26	Male	4.19	4.87	9.6
9	10	19	Female	7.28	9.56	2.1

```
In [32]: len(df)
```

```
Out[32]: 20000
```

```
In [33]: df.loc[0]
```

```
Out[33]: student_id      1
         age            23
         gender        Female
         study_hours_per_day  4.35
         sleep_hours    3.63
         phone_usage_hours  3.38
         social_media_hours  2.73
         youtube_hours   1.83
         gaming_hours    5.26
         breaks_per_day   6
         coffee_intake_mg  347
         exercise_minutes 111
         assignments_completed  2
         attendance_percentage 57.21
         stress_level     10
         focus_score      57
         final_grade     81.87
         productivity_score 33.78
         Name: 0, dtype: object
```

```
In [34]: #In this step we are checking if any missing values occurs in our data so
         df.isnull().sum()
```

```
Out[34]: student_id      0
         age            0
         gender        0
         study_hours_per_day  0
         sleep_hours    0
         phone_usage_hours  0
         social_media_hours  0
         youtube_hours   0
         gaming_hours    0
         breaks_per_day   0
         coffee_intake_mg  0
         exercise_minutes  0
         assignments_completed  0
         attendance_percentage  0
         stress_level     0
         focus_score      0
         final_grade     0
         productivity_score  0
         dtype: int64
```

```
In [35]: #Handling the missing values
         missingValuesPercentage = df.isnull().mean() * 100

         #Filtering the columns based on the percentage of the missing values of e
         columnsWithMissing30 = missingValuesPercentage[missingValuesPercentage >

         #Printing the missing values that is greater than 30 percent.
         print('Column with more than 30 percent missing values :')
         if columnsWithMissing30.empty:
             print("No missing value detected")
         else:
             print(columnsWithMissing30)
```

Column with more than 30 percent missing values :
No missing value detected

```
In [36]: #Hence no missing values were detected, we will find in this step the out
import numpy as np
import pandas as pd

data = df.copy()

# Function to detect outliers using Z-score
def detect_outliers_zscore(column, threshold=3):
    mean = np.mean(column)
    std_dev = np.std(column)
    z_scores = (column - mean) / std_dev
    return np.abs(z_scores) > threshold

# We select only numeric columns to apply
numeric_cols = data.select_dtypes(include=np.number).columns

# Apply Z-score only on numeric columns
for col in numeric_cols:
    data[f'{col}_Outlier'] = detect_outliers_zscore(data[col])

# Show rows that contain any outlier
outliers_only = data[data.filter(like='_Outlier').any(axis=1)]

outliers_only
```

```
Out[36]:
```

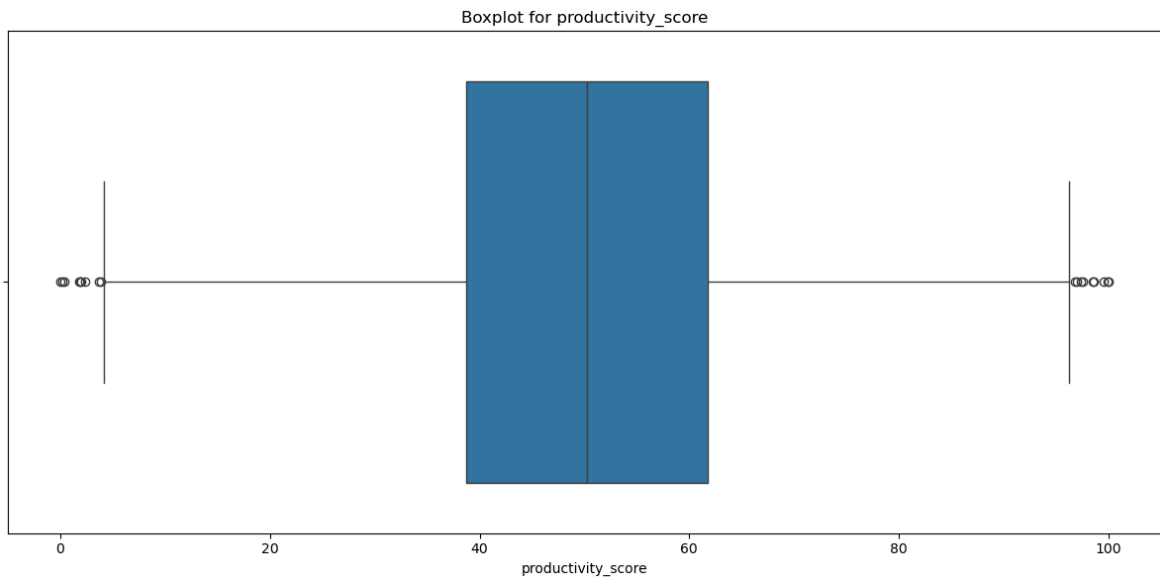
	student_id	age	gender	study_hours_per_day	sleep_hours	phone_usage_
61	62	21	Female	9.06	9.27	
1073	1074	21	Female	9.86	8.50	
2106	2107	23	Male	9.64	7.59	
4533	4534	20	Female	0.67	3.73	
5008	5009	23	Female	9.76	9.81	
6236	6237	17	Female	0.65	3.06	
8901	8902	20	Male	0.51	3.76	
10597	10598	21	Male	1.16	3.34	
16228	16229	18	Male	1.15	3.20	
19125	19126	23	Male	9.83	9.31	

10 rows x 35 columns

```
In [37]: plt.figure(figsize=(12, 6))

#The only column that has outliers is the productivity_score column so we
for i, col in enumerate(df.columns[17:18]): # Only plotting the feature
    sns.boxplot(x=df[col])
    plt.title(f"Boxplot for {col}")
```

```
plt.tight_layout()
plt.show()
```



In [38]: *#Since the data frame has only 10 rows that contains outliers we will rep*

data = df.copy() # Create a copy to avoid modifying the original datafra

```
def replace_outliers_with_null(data, threshold=3):
```

Boolean DataFrame indicating if a value is an outlier

```
outliers = pd.DataFrame(False, index=df.index, columns=df.columns)
```

Looping through each numeric column

```
for column in data.columns:
```

```
    if data[column].dtype in ['float64', 'int64']: # Applicable on n
```

```
        mean = data[column].mean()
```

```
        std_dev = data[column].std()
```

```
        z_scores = (data[column] - mean) / std_dev
```

Mark True when abs(z_score) > threshold

```
        outliers[column] = abs(z_scores) > threshold
```

Replace outliers with null values

```
        data.loc[outliers[column], column] = np.nan
```

```
    return data
```

#This variable will hold the data after replacing the outliers with null

```
cleaned_data = replace_outliers_with_null(data, threshold=3)
```

DataFrame that contains rows with NaN values after outlier replacement

```
rows_with_nan = cleaned_data[cleaned_data.isna().any(axis=1)].copy()
```

```
rows_with_nan
```

```
print("Original Data length:")
```

```
print(len(data))
```

```
print("\ncleaned Data length:")
```

```
print(len(cleaned_data))
```

```
print("\nData after replacing outliers with null values:")
cleaned_data.head()
```

Original Data length:
20000

cleaned Data length:
20000

Data after replacing outliers with null values:

Out [38]:

	student_id	age	gender	study_hours_per_day	sleep_hours	phone_usage_hou
0	1.0	23.0	Female	4.35	3.63	3.3
1	2.0	20.0	Male	6.14	6.58	5.4
2	3.0	29.0	Female	4.98	3.26	4.8
3	4.0	27.0	Female	3.19	4.58	10.0
4	5.0	24.0	Male	7.67	6.21	3.0

In [39]: `cleaned_data.isnull().sum()`

Out [39]:

student_id	0
age	0
gender	0
study_hours_per_day	0
sleep_hours	0
phone_usage_hours	0
social_media_hours	0
youtube_hours	0
gaming_hours	0
breaks_per_day	0
coffee_intake_mg	0
exercise_minutes	0
assignments_completed	0
attendance_percentage	0
stress_level	0
focus_score	0
final_grade	0
productivity_score	9
dtype:	int64

In [40]: *#I wanted to print the rows that has a null values*
`rows_with_null = cleaned_data[cleaned_data.isnull().any(axis=1)]`
`print(len(rows_with_null))`
`rows_with_null`

9

Out [40]:

	student_id	age	gender	study_hours_per_day	sleep_hours	phone_usage
61	62.0	21.0	Female	9.06	9.27	
1073	1074.0	21.0	Female	9.86	8.50	
4533	4534.0	20.0	Female	0.67	3.73	
5008	5009.0	23.0	Female	9.76	9.81	
6236	6237.0	17.0	Female	0.65	3.06	
8901	8902.0	20.0	Male	0.51	3.76	
10597	10598.0	21.0	Male	1.16	3.34	
16228	16229.0	18.0	Male	1.15	3.20	
19125	19126.0	23.0	Male	9.83	9.31	

```
In [41]: # Replace null values in numeric columns with the column mean
cleaned_data = cleaned_data.fillna(cleaned_data.mean(numeric_only=True))

print("Missing values after replacement:")
print(cleaned_data.isnull().sum())
```

Missing values after replacement:

```
student_id      0
age             0
gender          0
study_hours_per_day  0
sleep_hours     0
phone_usage_hours  0
social_media_hours  0
youtube_hours   0
gaming_hours    0
breaks_per_day  0
coffee_intake_mg  0
exercise_minutes  0
assignments_completed  0
attendance_percentage  0
stress_level     0
focus_score     0
final_grade     0
productivity_score  0
dtype: int64
```

```
In [42]: cleaned_data = cleaned_data.drop('student_id', axis=1)
```

```
In [43]: cleaned_data.head()
```

Out [43]:

	age	gender	study_hours_per_day	sleep_hours	phone_usage_hours	social_m
0	23.0	Female	4.35	3.63	3.38	
1	20.0	Male	6.14	6.58	5.48	
2	29.0	Female	4.98	3.26	4.83	
3	27.0	Female	3.19	4.58	10.06	
4	24.0	Male	7.67	6.21	3.02	

```
In [93]: # Define target column (as STRING, not data)
target_column = 'productivity_score'

# Features (X)
X = cleaned_data.drop(columns=[target_column])

# Target (y)
y = cleaned_data[target_column]
```

```
In [94]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X[numeric_features])
print(X)

[[-0.00198413 -0.32979401 -1.42274794 ...  1.57706791 -0.36897771
  0.67143244]
 [-0.80096225  0.32282195  0.03064516 ...  1.57706791 -0.7654961
 -0.5419797 ]
 [ 1.59597211 -0.10010236 -1.60503792 ... -1.21343102 -1.31070888
  0.92314168]
 ...
 [ 0.53066795  1.4968015  -1.077875  ...  1.22825554  0.22579987
  0.65465182]
 [-0.26831017 -1.64596362 -1.02368068 ... -0.16699392 -0.5672369
 -0.99679227]
 [-1.06728829 -0.58865285 -0.08267023 ...  1.22825554  0.02754068
  0.14081778]]
```

```
In [95]: X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.30,
    random_state=42
)
```

```
In [101... import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_sc
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 name and R^2 score on the plot.

    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='Perfect Prediction Line')`

`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, fontsize=12, color='green')`

`plt.legend()`

`plt.grid(True)`

`plt.show()`

return results

In [102... `rf_regressor = RandomForestRegressor(n_estimators=100,`
`random_state=42)`

Train the model

`rf_regressor.fit(X_train, y_train)`

Make predictions

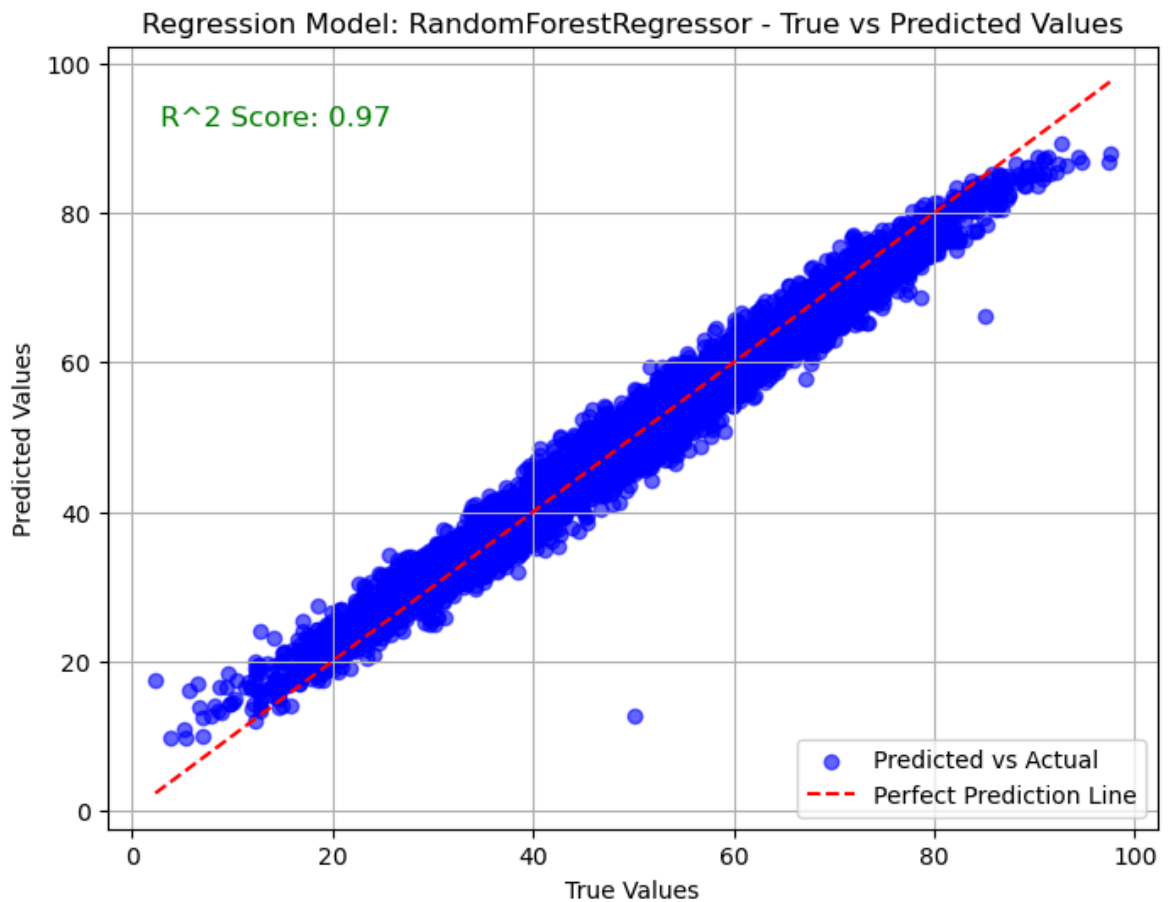
`y_pred = rf_regressor.predict(X_test)`

Evaluate the model

`evaluation_results = evaluate_model('RandomForestRegressor', y_test,`
`y_pred)`

Print the evaluation results

`evaluation_results`



```
Out[102...] {'Mean Absolute Error (MAE)': 2.068541718925183,
             'Mean Squared Error (MSE)': 7.1632593283975305,
             'Root Mean Squared Error (RMSE)': np.float64(2.6764265968633496),
             'R^2 Score': 0.9725911626775411}
```

```
In [105...] from sklearn.tree import DecisionTreeRegressor

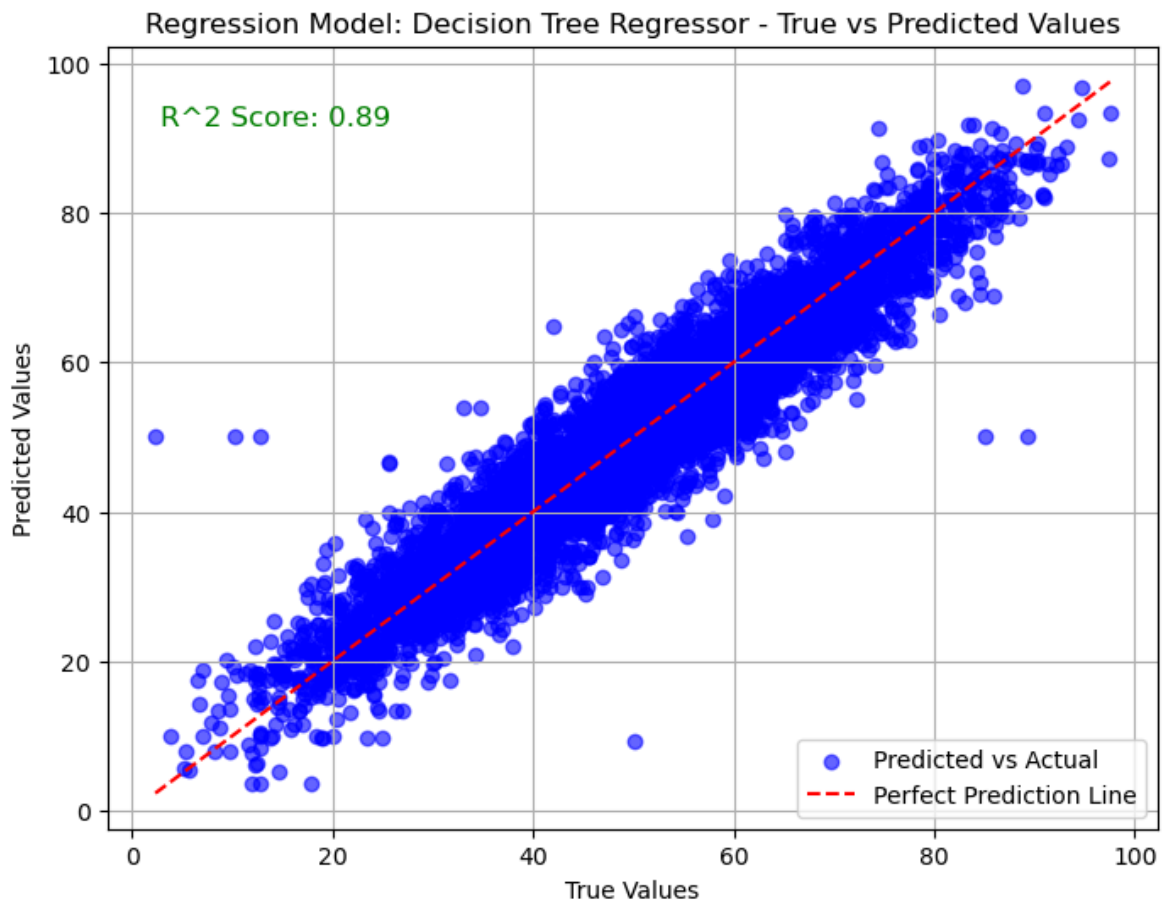
DT = DecisionTreeRegressor(random_state=42)

# Train
DT.fit(X_train, y_train)

# Predict
y_pred = DT.predict(X_test)

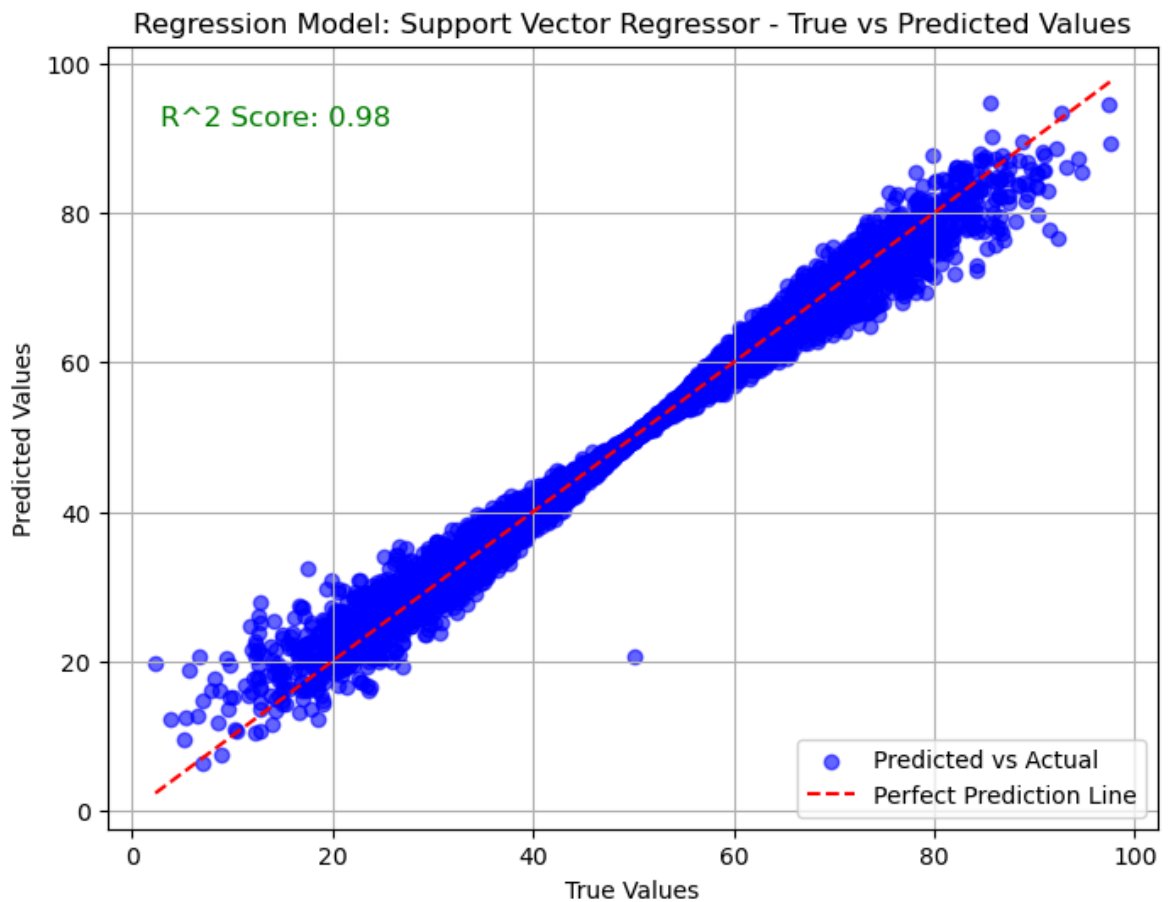
# Evaluate
evaluation_results = evaluate_model('Decision Tree Regressor', y_test, y_

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



Mean Absolute Error (MAE): 4.0718
 Mean Squared Error (MSE): 27.6320
 Root Mean Squared Error (RMSE): 5.2566
 R^2 Score: 0.8943

```
In [106... 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): 1.4307
 Mean Squared Error (MSE): 5.0696
 Root Mean Squared Error (RMSE): 2.2516
 R² Score: 0.9806

```
In [108... from sklearn.ensemble import GradientBoostingRegressor

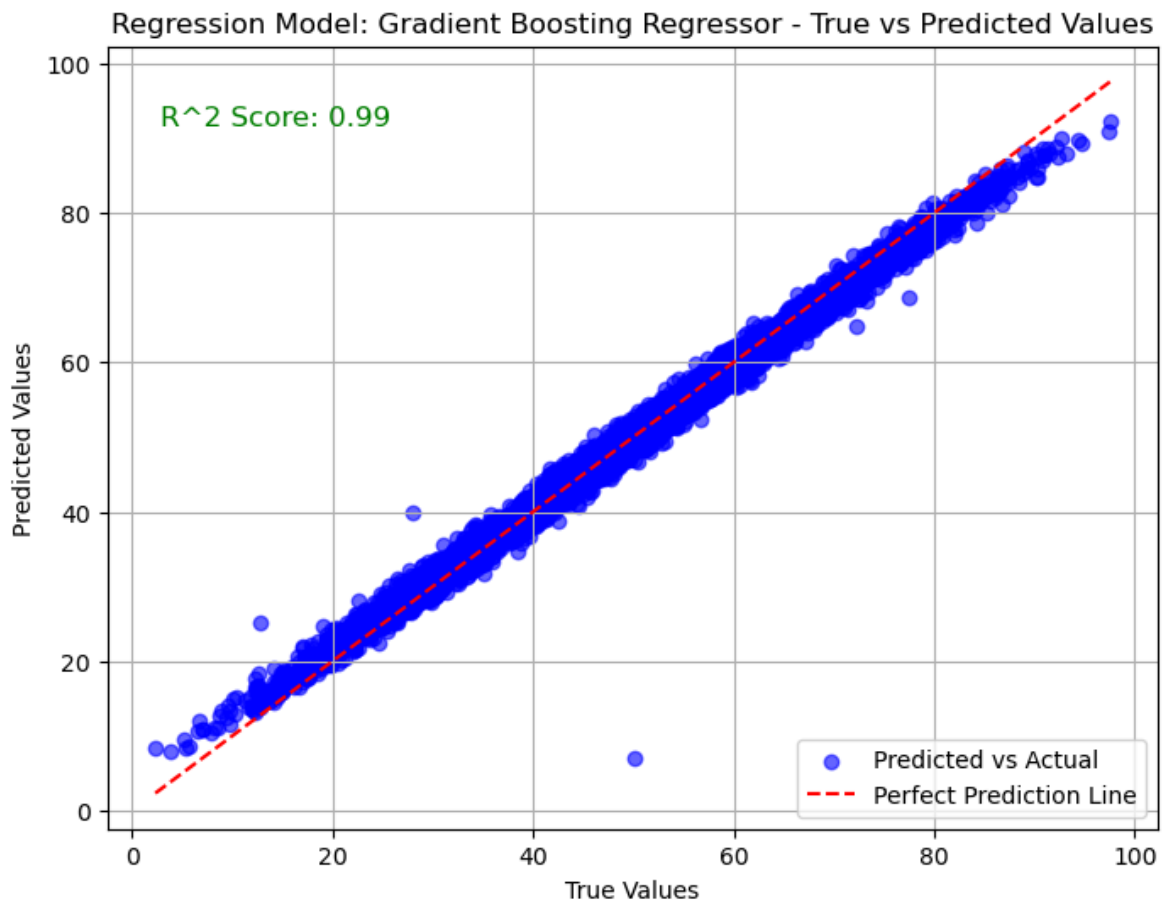
GBR = GradientBoostingRegressor(random_state=42)

# Train
GBR.fit(X_train, y_train)

# Predict
y_pred = GBR.predict(X_test)

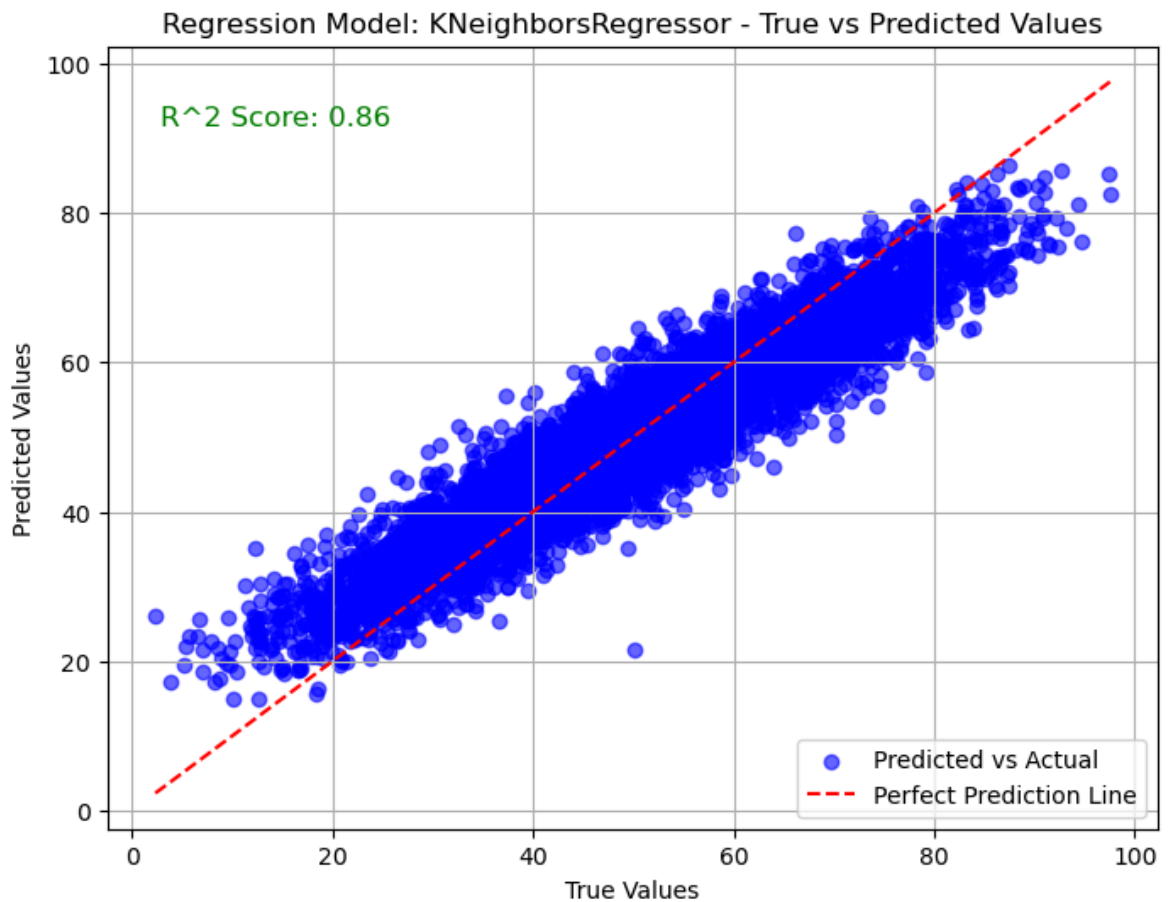
# Evaluate
evaluation_results = evaluate_model('Gradient Boosting Regressor', y_test

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



Mean Absolute Error (MAE): 1.1817
 Mean Squared Error (MSE): 2.6041
 Root Mean Squared Error (RMSE): 1.6137
 R^2 Score: 0.9900

```
In [110... from sklearn.neighbors import KNeighborsRegressor
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}")
```

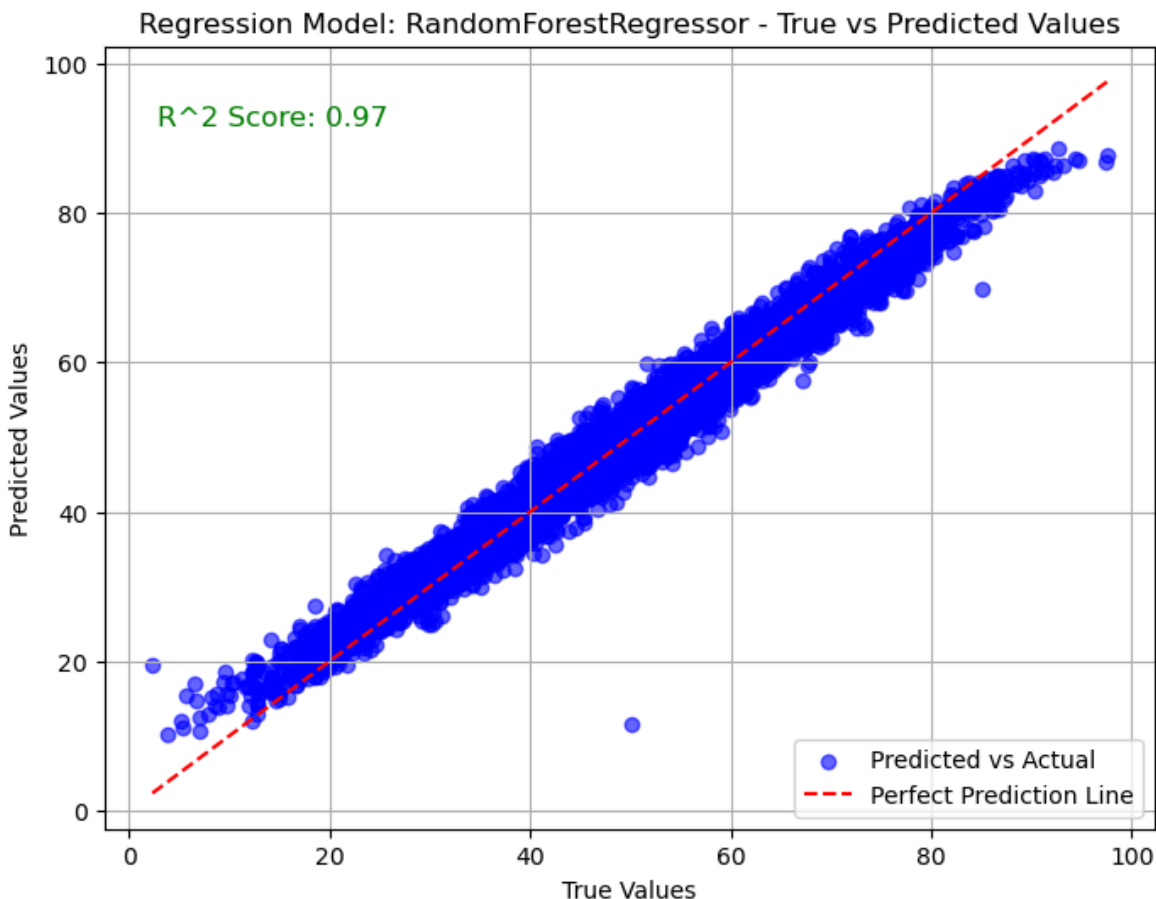


Mean Absolute Error (MAE): 4.8079
 Mean Squared Error (MSE): 36.3351
 Root Mean Squared Error (RMSE): 6.0279
 R^2 Score: 0.8610

```
In [112]: from sklearn.model_selection import train_test_split, GridSearchCV
rf = RandomForestRegressor(random_state=42)
# Define parameter grid (try all combinations)
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
# Grid Search
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    verbose=2
)
grid_search.fit(X_train, y_train)
# Evaluate best model
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', 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}")
```

```
print("Best Parameters:", grid_search.best_params_)
```

Fitting 3 folds for each of 24 candidates, totalling 72 fits



Mean Absolute Error (MAE): 2.0609

Mean Squared Error (MSE): 7.1129

Root Mean Squared Error (RMSE): 2.6670

R² Score: 0.9728

Best Parameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}

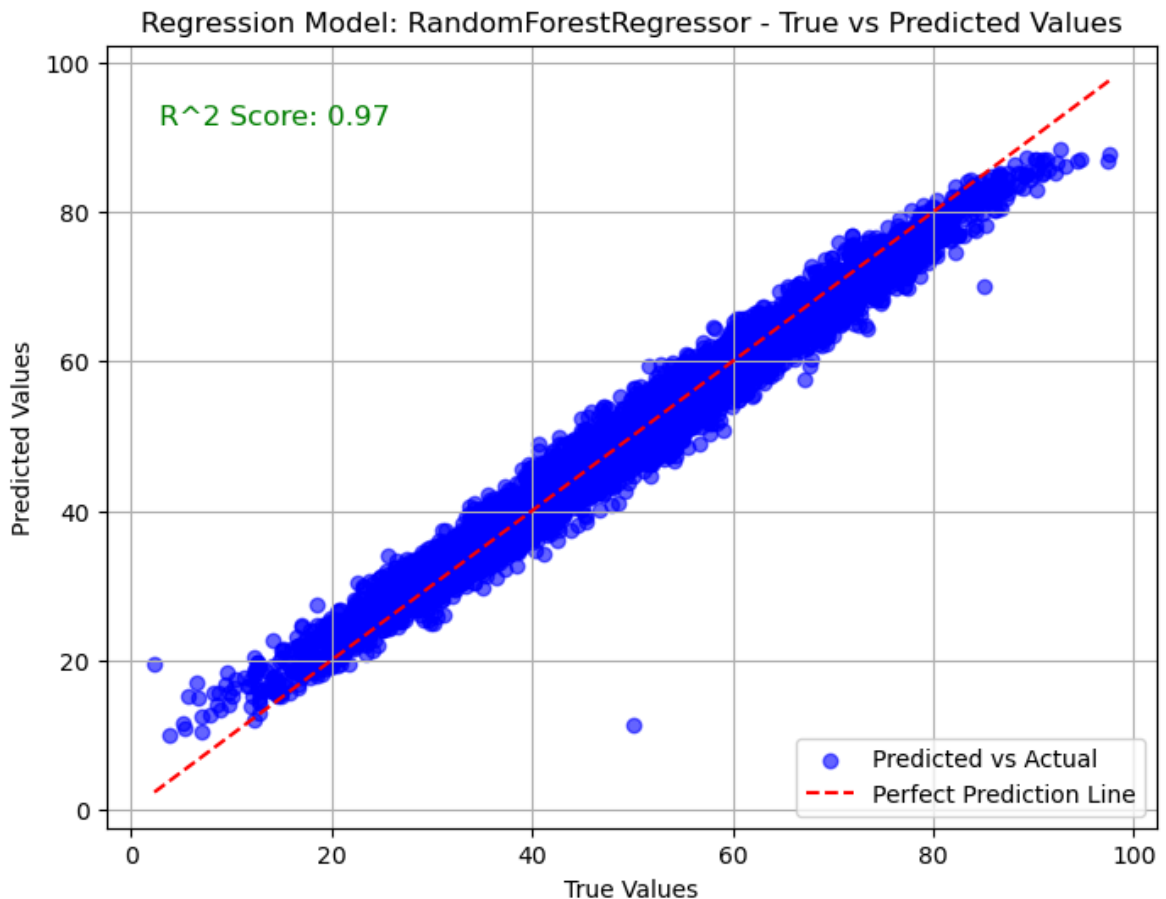
```
In [113... from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
# Define model
rf = RandomForestRegressor(random_state=42)
# Define parameter distributions (sample randomly)
param_distributions = {
    'n_estimators': randint(100, 500),
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': randint(2, 10),
    'min_samples_leaf': randint(1, 5)
}
# Random Search
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_distributions,
    n_iter=20, # number of random combinations to try
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    random_state=42,
    verbose=2
)
```

```

random_search.fit(X_train, y_train)
# Evaluate best model
best_rf_random = random_search.best_estimator_
y_pred = best_rf_random.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', 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}")
print("Best Parameters:", grid_search.best_params_)

```

Fitting 3 folds for each of 20 candidates, totalling 60 fits



Mean Absolute Error (MAE): 2.0547

Mean Squared Error (MSE): 7.0831

Root Mean Squared Error (RMSE): 2.6614

R² Score: 0.9729

Best Parameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}

In [114...

```

SVM = SVR()
param_grid = {
    'C': [1, 10],
    'kernel': ['linear', 'rbf']
}
grid_search = GridSearchCV(
    SVM,
    param_grid,
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    verbose=2
)

```

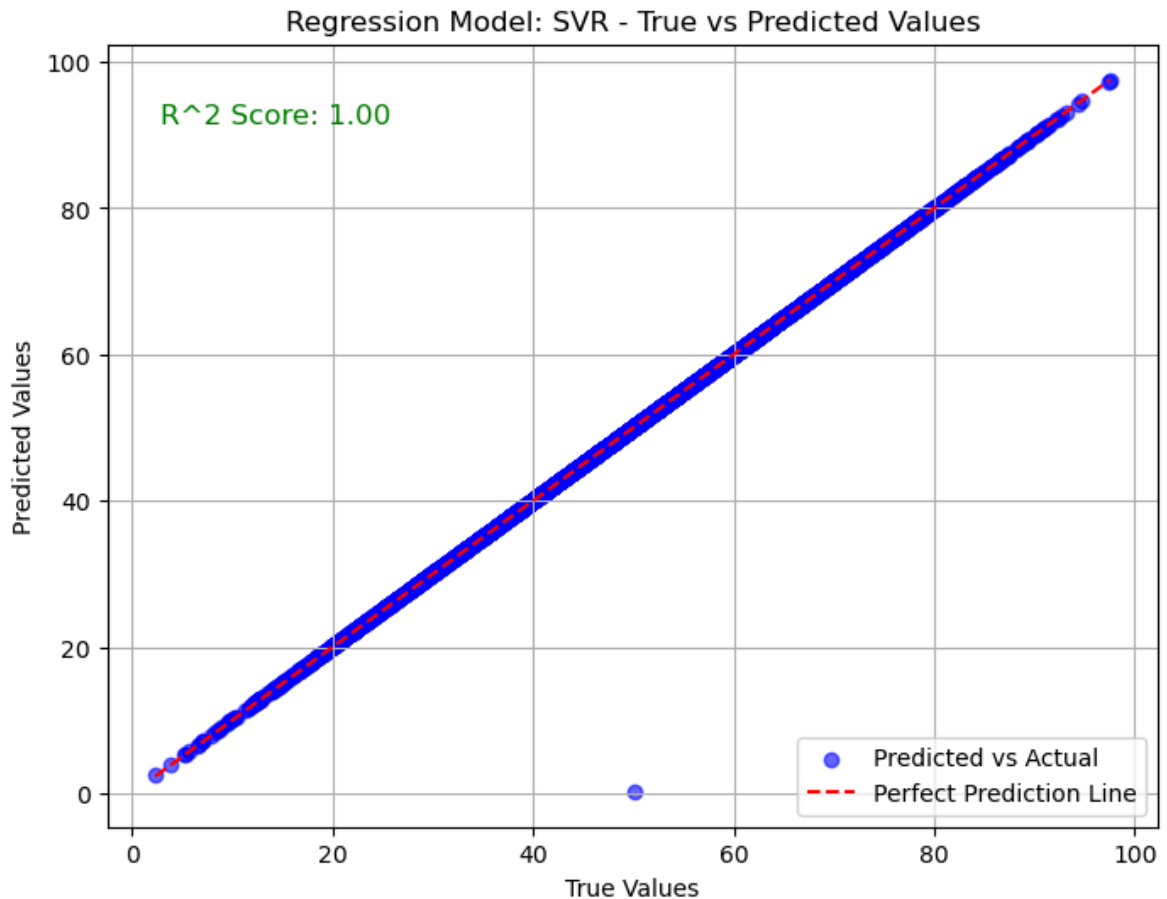
```

grid_search.fit(X_train, y_train)
print("Best Parameters (Grid Search):", grid_search.best_params_)
y_pred = grid_search.best_estimator_.predict(X_test)
evaluation_results = evaluate_model('SVR', 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}")

```

Fitting 3 folds for each of 4 candidates, totalling 12 fits

Best Parameters (Grid Search): {'C': 10, 'kernel': 'linear'}



Mean Absolute Error (MAE): 0.0375

Mean Squared Error (MSE): 0.4162

Root Mean Squared Error (RMSE): 0.6451

R² Score: 0.9984

[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 2.8s

[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 5.9s

[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time= 3.0s

[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time= 6.4s

[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 11.2s

[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time= 12.0s

[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 14.8s

[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time= 12.2s

[CV] END max_depth=20, min_samples_leaf=4, min_samples_split=6, n_estimators=120; total time= 6.8s

[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=4, n_estimators=314; total time= 19.4s

[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=3, n_estimators=443; total time= 23.1s

[CV] END max_depth=30, min_samples_leaf=2, min_samples_split=7, n_estimators=352; total time= 20.2s

[CV] END max_depth=30, min_samples_leaf=3, min_samples_split=7, n_estimators=274; total time= 15.4s

[CV] END max_depth=None, min_samples_leaf=3, min_samples_split=3, n_estimators=487; total time= 27.8s

[CV] END max_depth=30, min_samples_leaf=3, min_samples_split=5, n_estimators=363; total time= 18.1s

[CV] ENDC=10, kernel=linear; total time= 0.6s

[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 2.8s

[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 6.3s

[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time= 6.4s

[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time= 5.8s

[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 11.3s

[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time= 11.7s

[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 13.5s

[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time= 6.2s

[CV] END max_depth=20, min_samples_leaf=4, min_samples_split=6, n_estimators=370; total time= 20.6s

[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=3, n_estimators=443; total time= 23.3s

[CV] END max_depth=30, min_samples_leaf=4, min_samples_split=6, n_estimators=260; total time= 14.4s

[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estimators=158; total time= 10.8s

```

[CV] END max_depth=30, min_samples_leaf=3, min_samples_split=7, n_estimators=274; total time= 15.7s
[CV] END max_depth=None, min_samples_leaf=3, min_samples_split=8, n_estimators=120; total time= 6.8s
[CV] END max_depth=None, min_samples_leaf=4, min_samples_split=7, n_estimators=341; total time= 18.5s
[CV] END max_depth=30, min_samples_leaf=3, min_samples_split=5, n_estimators=363; total time= 18.7s
[CV] END .....C=1, kernel=linear; total time= 0.3s
[CV] END .....C=10, kernel=linear; total time= 0.5s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time= 2.8s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 6.3s
[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time= 6.2s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 11.2s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time= 11.6s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 15.0s
[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time= 6.4s
[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time= 11.2s
[CV] END max_depth=20, min_samples_leaf=4, min_samples_split=6, n_estimators=370; total time= 20.3s
[CV] END max_depth=30, min_samples_leaf=4, min_samples_split=9, n_estimators=230; total time= 12.4s
[CV] END max_depth=30, min_samples_leaf=2, min_samples_split=7, n_estimators=485; total time= 27.9s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=9, n_estimators=287; total time= 16.2s
[CV] END max_depth=10, min_samples_leaf=3, min_samples_split=5, n_estimators=154; total time= 7.9s
[CV] END max_depth=None, min_samples_leaf=3, min_samples_split=8, n_estimators=120; total time= 6.9s
[CV] END max_depth=None, min_samples_leaf=4, min_samples_split=7, n_estimators=341; total time= 18.4s
[CV] END max_depth=30, min_samples_leaf=3, min_samples_split=5, n_estimators=363; total time= 18.5s
[CV] END .....C=1, kernel=rbf; total time= 6.4s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 5.7s
[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time= 6.5s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 11.7s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time= 11.2s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 7.8s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time= 6.8s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 13.0s

```

```

[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time= 10.3s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=4, n_estimators=314; total time= 19.5s
[CV] END max_depth=30, min_samples_leaf=4, min_samples_split=9, n_estimators=230; total time= 12.4s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=3, n_estimators=443; total time= 23.0s
[CV] END max_depth=30, min_samples_leaf=2, min_samples_split=7, n_estimators=352; total time= 20.2s
[CV] END max_depth=10, min_samples_leaf=3, min_samples_split=5, n_estimators=154; total time= 8.0s
[CV] END max_depth=30, min_samples_leaf=4, min_samples_split=2, n_estimators=230; total time= 12.7s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=3, n_estimators=152; total time= 7.9s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=5, n_estimators=466; total time= 22.3s
[CV] END .....C=1, kernel=rbf; total time= 6.5s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 5.6s
[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time= 3.2s
[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time= 3.1s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 5.9s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time= 5.7s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time= 5.2s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time= 11.9s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 14.8s
[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time= 12.7s
[CV] END max_depth=20, min_samples_leaf=4, min_samples_split=6, n_estimators=120; total time= 6.8s
[CV] END max_depth=20, min_samples_leaf=3, min_samples_split=9, n_estimators=472; total time= 25.9s
[CV] END max_depth=30, min_samples_leaf=2, min_samples_split=7, n_estimators=485; total time= 27.9s
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estimators=158; total time= 10.8s
[CV] END max_depth=10, min_samples_leaf=3, min_samples_split=5, n_estimators=154; total time= 7.8s
[CV] END max_depth=30, min_samples_leaf=4, min_samples_split=2, n_estimators=230; total time= 12.7s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=3, n_estimators=152; total time= 8.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=3, n_estimators=152; total time= 8.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=5, n_estimators=466; total time= 21.4s
[CV] END .....C=1, kernel=rbf; total time= 6.6s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 5.8s

```

```

[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time= 6.5s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 11.5s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time= 5.3s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time= 5.7s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time= 11.9s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 12.9s
[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time= 6.2s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=4, n_estimators=314; total time= 19.4s
[CV] END max_depth=20, min_samples_leaf=3, min_samples_split=9, n_estimators=472; total time= 25.9s
[CV] END max_depth=30, min_samples_leaf=4, min_samples_split=6, n_estimators=260; total time= 14.4s
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estimators=158; total time= 10.6s
[CV] END max_depth=30, min_samples_leaf=3, min_samples_split=7, n_estimators=274; total time= 15.4s
[CV] END max_depth=None, min_samples_leaf=3, min_samples_split=3, n_estimators=487; total time= 28.1s
[CV] END .....C=1, kernel=linear; total time= 0.3s
[CV] END .....C=10, kernel=rbf; total time= 14.1s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 2.9s
[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time= 2.8s
[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time= 6.4s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 5.9s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 11.2s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time= 6.1s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 7.7s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time= 6.7s
[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time= 6.5s
[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time= 6.1s
[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time= 10.3s
[CV] END max_depth=20, min_samples_leaf=4, min_samples_split=6, n_estimators=120; total time= 6.9s
[CV] END max_depth=20, min_samples_leaf=3, min_samples_split=9, n_estimators=472; total time= 26.2s
[CV] END max_depth=30, min_samples_leaf=4, min_samples_split=6, n_estimators=260; total time= 14.1s
[CV] END max_depth=30, min_samples_leaf=2, min_samples_split=7, n_estimators=352; total time= 20.5s

```

```

[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=9, n_estimators=287; total time= 16.2s
[CV] END max_depth=None, min_samples_leaf=3, min_samples_split=8, n_estimators=120; total time= 7.1s
[CV] END max_depth=None, min_samples_leaf=4, min_samples_split=7, n_estimators=341; total time= 18.6s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=5, n_estimators=466; total time= 21.1s
[CV] END .....C=10, kernel=linear; total time= 0.6s
[CV] END .....C=10, kernel=rbf; total time= 14.2s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time= 2.8s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time= 2.8s
[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time= 3.3s
[CV] END max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time= 3.1s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 5.9s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time= 5.6s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time= 5.4s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time= 5.8s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 7.9s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time= 6.8s
[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time= 6.5s
[CV] END max_depth=None, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time= 12.0s
[CV] END max_depth=20, min_samples_leaf=4, min_samples_split=6, n_estimators=370; total time= 20.5s
[CV] END max_depth=30, min_samples_leaf=4, min_samples_split=9, n_estimators=230; total time= 12.5s
[CV] END max_depth=30, min_samples_leaf=2, min_samples_split=7, n_estimators=485; total time= 28.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=9, n_estimators=287; total time= 16.1s
[CV] END max_depth=30, min_samples_leaf=4, min_samples_split=2, n_estimators=230; total time= 12.8s
[CV] END max_depth=None, min_samples_leaf=3, min_samples_split=3, n_estimators=487; total time= 27.4s
[CV] END .....C=1, kernel=linear; total time= 0.3s
[CV] END .....C=10, kernel=rbf; total time= 14.8s

```

```

In [120.. from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Count available features
n_features = X_train.shape[1]

```

```
select_kbest_pipeline = Pipeline(steps=[
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('model', RandomForestRegressor(random_state=42))
])

select_kbest_param_grid = {
    'feature_selection__k': [5, min(10, n_features), 'all'],
    'model__n_estimators': [100, 200],
    'model__max_depth': [5, 10, None]
}

select_kbest_grid = GridSearchCV(
    select_kbest_pipeline,
    select_kbest_param_grid,
    cv=3,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

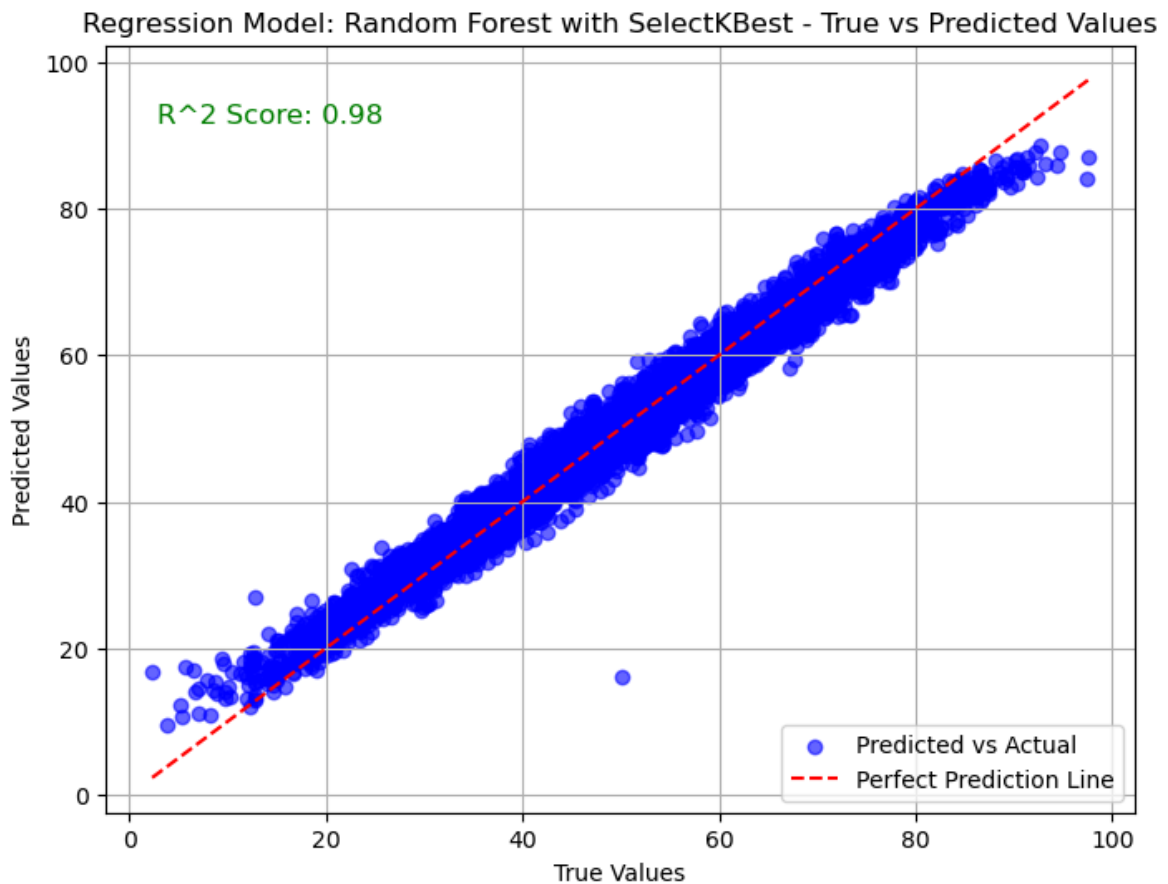
select_kbest_grid.fit(X_train, y_train)

y_pred_kbest = select_kbest_grid.best_estimator_.predict(X_test)

kbest_result = evaluate_model(
    'Random Forest with SelectKBest',
    y_test,
    y_pred_kbest
)

kbest_result['Feature Selection Method'] = 'SelectKBest'
kbest_result['Best Parameters'] = select_kbest_grid.best_params_

kbest_result
```



```
Out[120...] {'Mean Absolute Error (MAE)': 1.9199566872129963,
             'Mean Squared Error (MSE)': 6.208767593106233,
             'Root Mean Squared Error (RMSE)': np.float64(2.49173987268058),
             'R^2 Score': 0.9762433421532328,
             'Feature Selection Method': 'SelectKBest',
             'Best Parameters': {'feature_selection_k': 10,
                                'model__max_depth': None,
                                'model__n_estimators': 200}}
```

```
In [142...] feature_names = cleaned_data.drop(columns=['productivity_score']).select_
            include=['int64', 'float64']
            ).columns

            selected_features_kbest = np.array(feature_names)[select_kbest.get_support

            print("Selected Features using SelectKBest:")
            print(selected_features_kbest)
```

```
Selected Features using SelectKBest:
['age' 'study_hours_per_day' 'sleep_hours' 'phone_usage_hours'
 'breaks_per_day' 'coffee_intake_mg' 'attendance_percentage'
 'stress_level' 'focus_score' 'final_grade']
```

```
In [136...] # =====
# Requirement 4: Feature Selection Method 2 - SelectFromModel
# =====

from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestRegressor

# Use Random Forest to identify important features
selector_model = SelectFromModel(
    RandomForestRegressor(n_estimators=100, random_state=42)
```

```

)

X_train_model_selected = selector_model.fit_transform(X_train, y_train)
X_test_model_selected = selector_model.transform(X_test)

# Train model on selected features
rf_model_selected = RandomForestRegressor(random_state=42)
rf_model_selected.fit(X_train_model_selected, y_train)

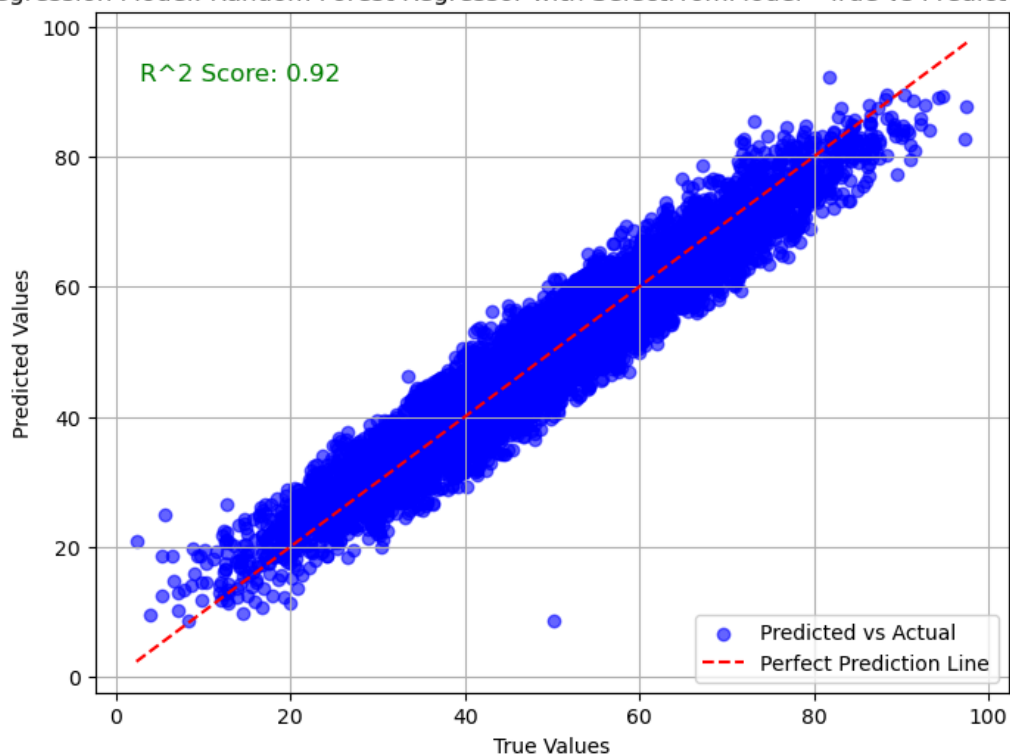
# Predict
y_pred_model_selected = rf_model_selected.predict(X_test_model_selected)

# Evaluate
model_selected_results = evaluate_model(
    'Random Forest Regressor with SelectFromModel',
    y_test,
    y_pred_model_selected
)

model_selected_results

```

Regression Model: Random Forest Regressor with SelectFromModel - True vs Predicted Values



```

Out[136...] {'Mean Absolute Error (MAE)': 3.6951855462233008,
'Mean Squared Error (MSE)': 20.830924059931387,
'Root Mean Squared Error (RMSE)': np.float64(4.564090715567712),
'R^2 Score': 0.9202944661556905}

```

```

In [140...] # =====
# Show selected features from SelectFromModel
# =====

feature_names = cleaned_data.drop(columns=['productivity_score']).select_
    include=['int64', 'float64']
).columns

selected_features_model = np.array(feature_names)[selector_model.get_supp

```

```
print("Selected Features using SelectFromModel:")
print(selected_features_model)
```

Selected Features using SelectFromModel:

```
['study_hours_per_day' 'sleep_hours' 'phone_usage_hours' 'focus_score']
```

```
In [143... feature_selection_results = pd.DataFrame([
    {
        'Method': 'SelectKBest',
        'MAE': kbest_result['Mean Absolute Error (MAE)'],
        'MSE': kbest_result['Mean Squared Error (MSE)'],
        'RMSE': kbest_result['Root Mean Squared Error (RMSE)'],
        'R2 Score': kbest_result['R^2 Score']
    },
    {
        'Method': 'SelectFromModel',
        'MAE': model_selected_results['Mean Absolute Error (MAE)'],
        'MSE': model_selected_results['Mean Squared Error (MSE)'],
        'RMSE': model_selected_results['Root Mean Squared Error (RMSE)'],
        'R2 Score': model_selected_results['R^2 Score']
    }
])

feature_selection_results
```

```
Out [143...

```

	Method	MAE	MSE	RMSE	R2 Score
0	SelectKBest	1.919957	6.208768	2.491740	0.976243
1	SelectFromModel	3.695186	20.830924	4.564091	0.920294

In []: