

```
from PIL import Image  
  
# Open an image file and display it  
img = Image.open('P1.png')  
display(img)
```



What is regrssion task?

A regression task is a type of machine learning problem where the goal is to predict a continuous numeric value rather than a discrete class label.

- Decision Tree Regression (DTR)
- Random Forest Regression (RFR)
- Support Vector Regression (SVR)
- k-Nearest Neighbors Regression (kNNR)
- Ridge Regression
- Lasso Regression

Common Regression Evaluation Metrics

1. Mean Absolute Error (MAE)

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Explanation:

Measures the average absolute difference between predicted and actual values.
All errors contribute equally.

2. Mean Squared Error (MSE)

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

Explanation:

Calculates the average of squared differences between predicted and actual values.
Penalizes larger errors more heavily due to squaring.

3. Root Mean Squared Error (RMSE)

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

Explanation:

Square root of MSE.
Has the same unit as the target variable and is more interpretable in real-world contexts.

4. Coefficient of Determination (R²)

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

Where:

- y_i : actual value
- $\hat{y}_i$: predicted value

- $(\bar{y})$: mean of actual values

Explanation:

Indicates how well the model explains the variance in the data.

- $(R^2 = 1)$: perfect fit
- $(R^2 = 0)$: model explains no variance
- $(R^2 < 0)$: model performs worse than predicting the mean

Import necessary libraries

```
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

```
data = pd.read_csv("WineQT.csv")
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar
0	7.4	0.70	0.00	1.9
1	7.8	0.88	0.00	2.6
2	7.8	0.76	0.04	2.3
3	11.2	0.28	0.56	1.9
4	7.4	0.70	0.00	1.9

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	11.0	34.0	0.9978	3.51	0.56
1	25.0	67.0	0.9968	3.20	0.68
2	15.0	54.0	0.9970	3.26	0.65

3	17.0	60.0	0.9980	3.16	0.58
4	11.0	34.0	0.9978	3.51	0.56

	alcohol	quality	Id
0	9.4	5	0
1	9.8	5	1
2	9.8	5	2
3	9.8	6	3
4	9.4	5	4

data.describe()

	fixed acidity	volatile acidity	citric acid	residual sugar \
count	1143.000000	1143.000000	1143.000000	1143.000000
mean	8.311111	0.531339	0.268364	2.532152
std	1.747595	0.179633	0.196686	1.355917
min	4.600000	0.120000	0.000000	0.900000
25%	7.100000	0.392500	0.090000	1.900000
50%	7.900000	0.520000	0.250000	2.200000
75%	9.100000	0.640000	0.420000	2.600000
max	15.900000	1.580000	1.000000	15.500000

	chlorides	free sulfur dioxide	total sulfur dioxide
density \			
count	1143.000000	1143.000000	1143.000000
1143.000000			
mean	0.086933	15.615486	45.914698
0.996730			
std	0.047267	10.250486	32.782130
0.001925			
min	0.012000	1.000000	6.000000
0.990070			
25%	0.070000	7.000000	21.000000
0.995570			
50%	0.079000	13.000000	37.000000
0.996680			
75%	0.090000	21.000000	61.000000
0.997845			
max	0.611000	68.000000	289.000000
1.003690			

	pH	sulphates	alcohol	quality	Id
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000
mean	3.311015	0.657708	10.442111	5.657043	804.969379
std	0.156664	0.170399	1.082196	0.805824	463.997116

min	2.740000	0.330000	8.400000	3.000000	0.000000
25%	3.205000	0.550000	9.500000	5.000000	411.000000
50%	3.310000	0.620000	10.200000	6.000000	794.000000
75%	3.400000	0.730000	11.100000	6.000000	1209.500000
max	4.010000	2.000000	14.900000	8.000000	1597.000000

```
data.isnull().sum()
```

```
fixed acidity      0
volatile acidity   0
citric acid        0
residual sugar     0
chlorides          0
free sulfur dioxide 0
total sulfur dioxide 0
density           0
pH               0
sulphates         0
alcohol           0
quality           0
Id                0
dtype: int64
```

Define features and target variable

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

```
y = data['quality']
```

```
X
```

```

      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides \
0              7.4              0.700          0.00              1.9
0.076
1              7.8              0.880          0.00              2.6
0.098
2              7.8              0.760          0.04              2.3
0.092
3             11.2              0.280          0.56              1.9
0.075
4              7.4              0.700          0.00              1.9
0.076
...              ...              ...          ...              ...
...
1138           6.3              0.510          0.13              2.3
0.076
```

1139	6.8	0.620	0.08	1.9
0.068				
1140	6.2	0.600	0.08	2.0
0.090				
1141	5.9	0.550	0.10	2.2
0.062				
1142	5.9	0.645	0.12	2.0
0.075				

	free sulfur dioxide	total sulfur dioxide	density	pH
sulphates \				
0	11.0	34.0	0.99780	3.51
0.56				
1	25.0	67.0	0.99680	3.20
0.68				
2	15.0	54.0	0.99700	3.26
0.65				
3	17.0	60.0	0.99800	3.16
0.58				
4	11.0	34.0	0.99780	3.51
0.56				
...	...	...	...	...
...				
1138	29.0	40.0	0.99574	3.42
0.75				
1139	28.0	38.0	0.99651	3.42
0.82				
1140	32.0	44.0	0.99490	3.45
0.58				
1141	39.0	51.0	0.99512	3.52
0.76				
1142	32.0	44.0	0.99547	3.57
0.71				

	alcohol
0	9.4
1	9.8
2	9.8
3	9.8
4	9.4
...	...
1138	11.0
1139	9.5
1140	10.5
1141	11.2
1142	10.2

[1143 rows x 11 columns]

Scaling features (X)

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

X

```
array([[ -0.52157961,  0.93933222, -1.36502663, ...,  1.27069495,
        -0.57365783, -0.96338181],
       [ -0.29259344,  1.94181282, -1.36502663, ..., -0.70892755,
         0.1308811 , -0.59360107],
       [ -0.29259344,  1.27349242, -1.16156762, ..., -0.32577481,
        -0.04525363, -0.59360107],
       ...,
       [ -1.20853813,  0.38239855, -0.9581086 , ...,  0.88754221,
        -0.45623467,  0.05351522],
       [ -1.38027776,  0.10393172, -0.8563791 , ...,  1.33455374,
         0.60057372,  0.70063152],
       [ -1.38027776,  0.6330187 , -0.75464959, ...,  1.65384769,
         0.30701583, -0.22382033]], shape=(1143, 11))
```

Splitting data

```
# 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_state=42)
```

Evaluation models

```
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 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

```

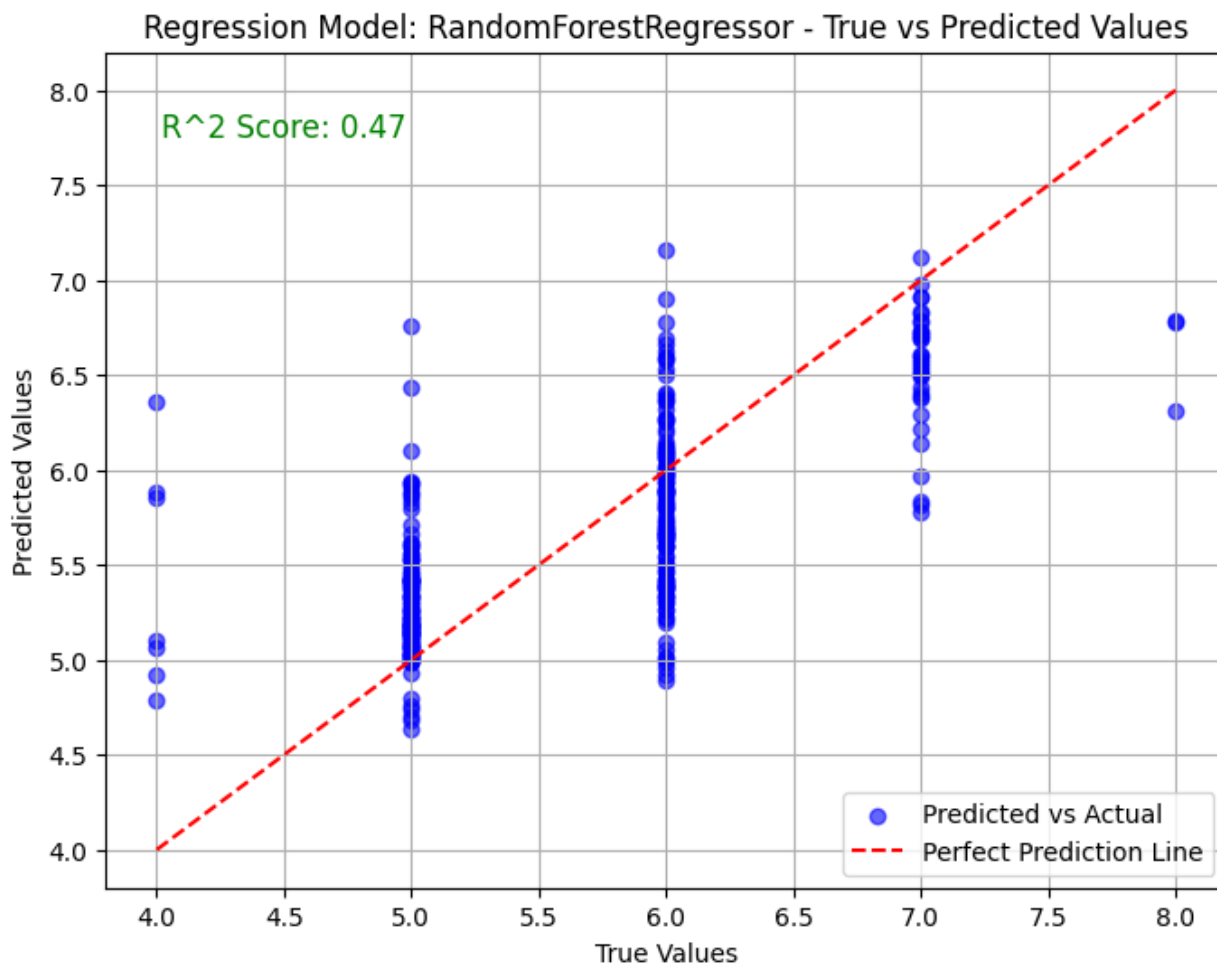
RandomForestRegressor

```

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

```



```
{'Mean Absolute Error (MAE)': 0.4141258741258741,
'Mean Squared Error (MSE)': 0.3020489510489511,
'Root Mean Squared Error (RMSE)': np.float64(0.549589802533627),
'R^2 Score': 0.4714644133062359}
```

y_pred

```
array([5.26, 5.67, 5.41, 5.05, 6.06, 6.79, 5.27, 5.12, 5.69, 5.03,
6.71,
6.25, 5.65, 6.01, 5.33, 5.02, 5.56, 5.62, 5.15, 6.9 , 6.66,
6.13,
5.04, 6.79, 5.26, 5.2 , 6.53, 6.1 , 5.42, 5.85, 5.38, 5.54,
6.78,
5.66, 5.38, 5.22, 5.67, 6.84, 6.41, 5.94, 6.59, 5.71, 6.26,
5.18,
5.1 , 4.99, 5.37, 5.8 , 4.99, 5.97, 6.57, 5.36, 5.51, 6.69,
5.52,
6.55, 5.64, 6.36, 5.99, 5.14, 5.02, 5.98, 5.91, 6.69, 6.4 ,
5.64,
6.58, 5.07, 5.25, 5.4 , 6.5 , 5.38, 5.39, 5.6 , 5.3 , 5.14,
```

```

5.47,
5.27,
5.94,
6.83,
,
5.06,
5.33,
5.22,
6.41,
5.04,
6.53,
6.29,
6.63,
5.64,
7.16,
,
5.89,
5.39,
,
6.78])

```

```

y_test

```

```

158      5
1081     6
291      5
538      6
367      6
...
813      7
213      5

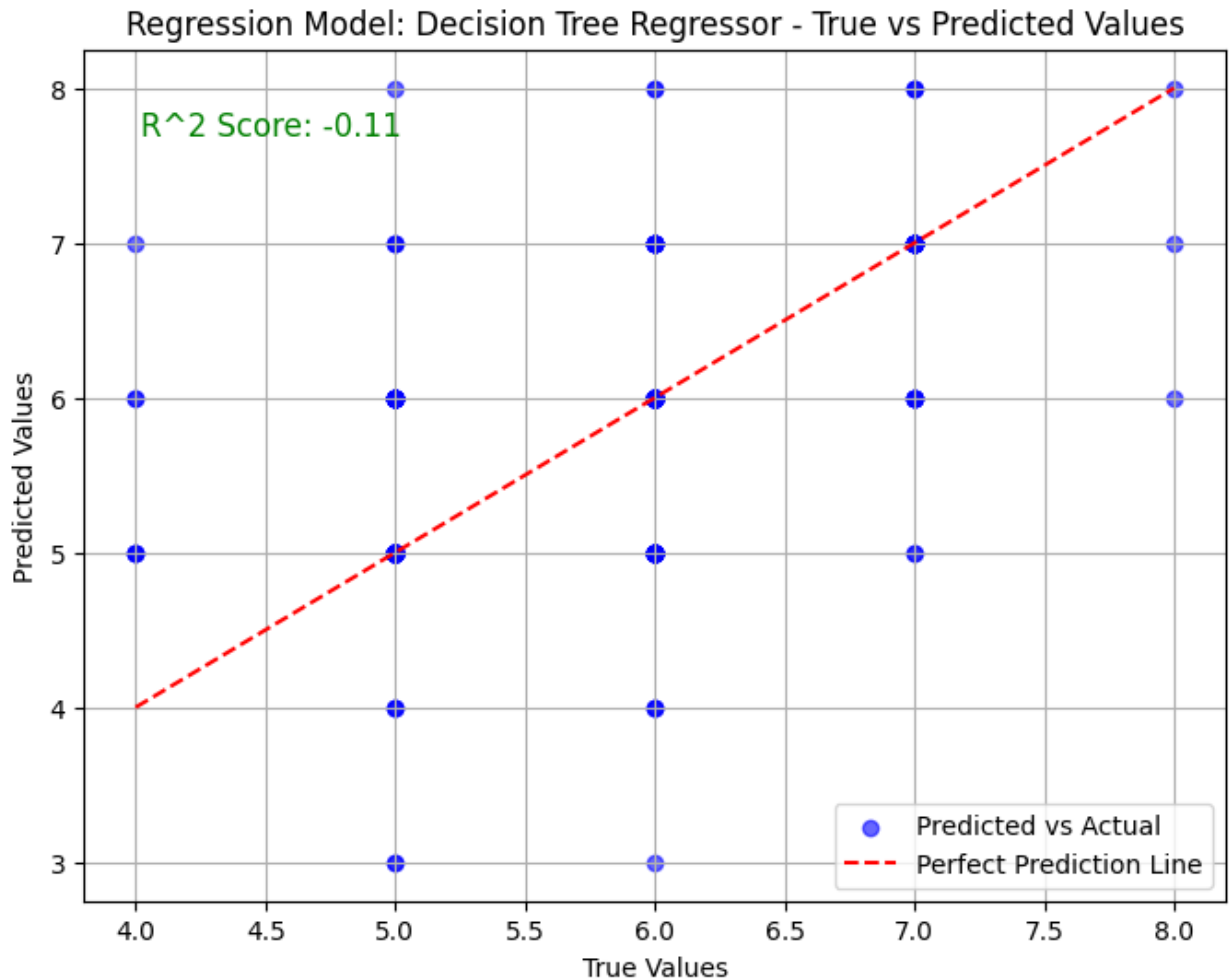
```

```
120      4
1026     6
667      7
Name: quality, Length: 286, dtype: int64
```

Decision Tree Regressor

```
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}")
```



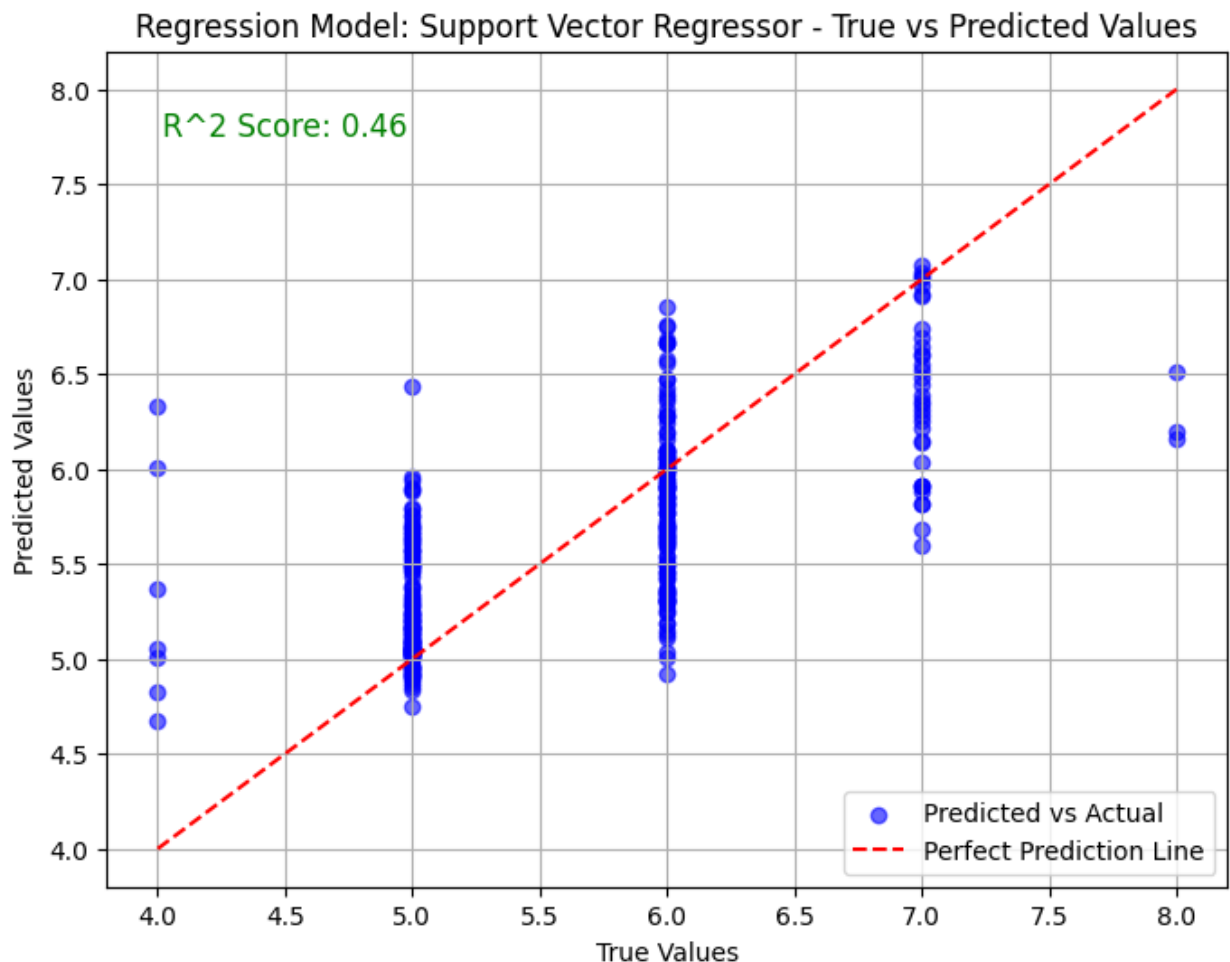
Mean Absolute Error (MAE): 0.4650
Mean Squared Error (MSE): 0.6329
Root Mean Squared Error (RMSE): 0.7955
R² Score: -0.1074

Support Vector Regressor (SVR)

```
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}")
```

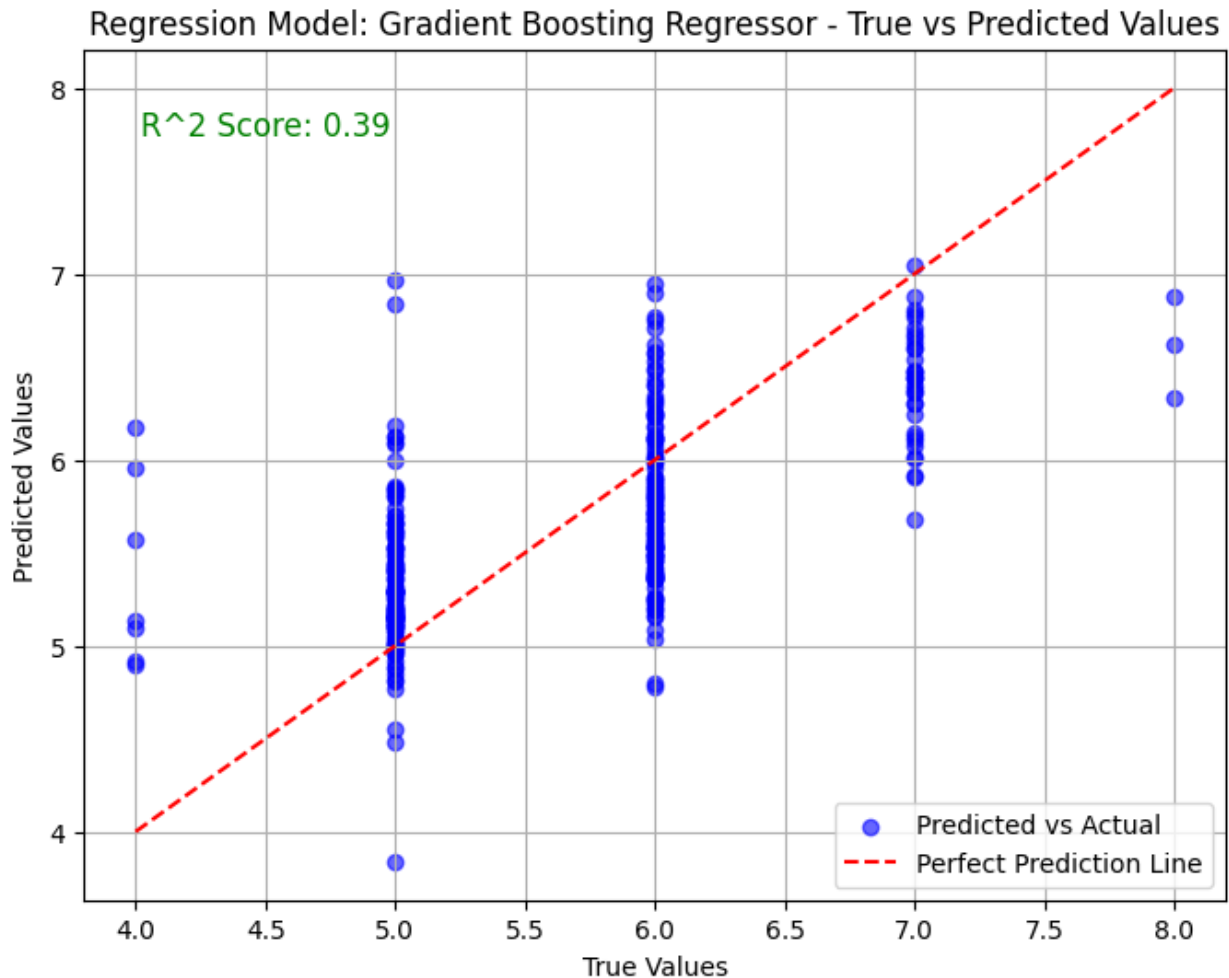


Mean Absolute Error (MAE): 0.4091
Mean Squared Error (MSE): 0.3098
Root Mean Squared Error (RMSE): 0.5566
R² Score: 0.4579

Gradient Boosting Regressor

```
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
R^2 Score: 0.3922
```

KNeighborsRegressor

```
knn = KNeighborsRegressor()

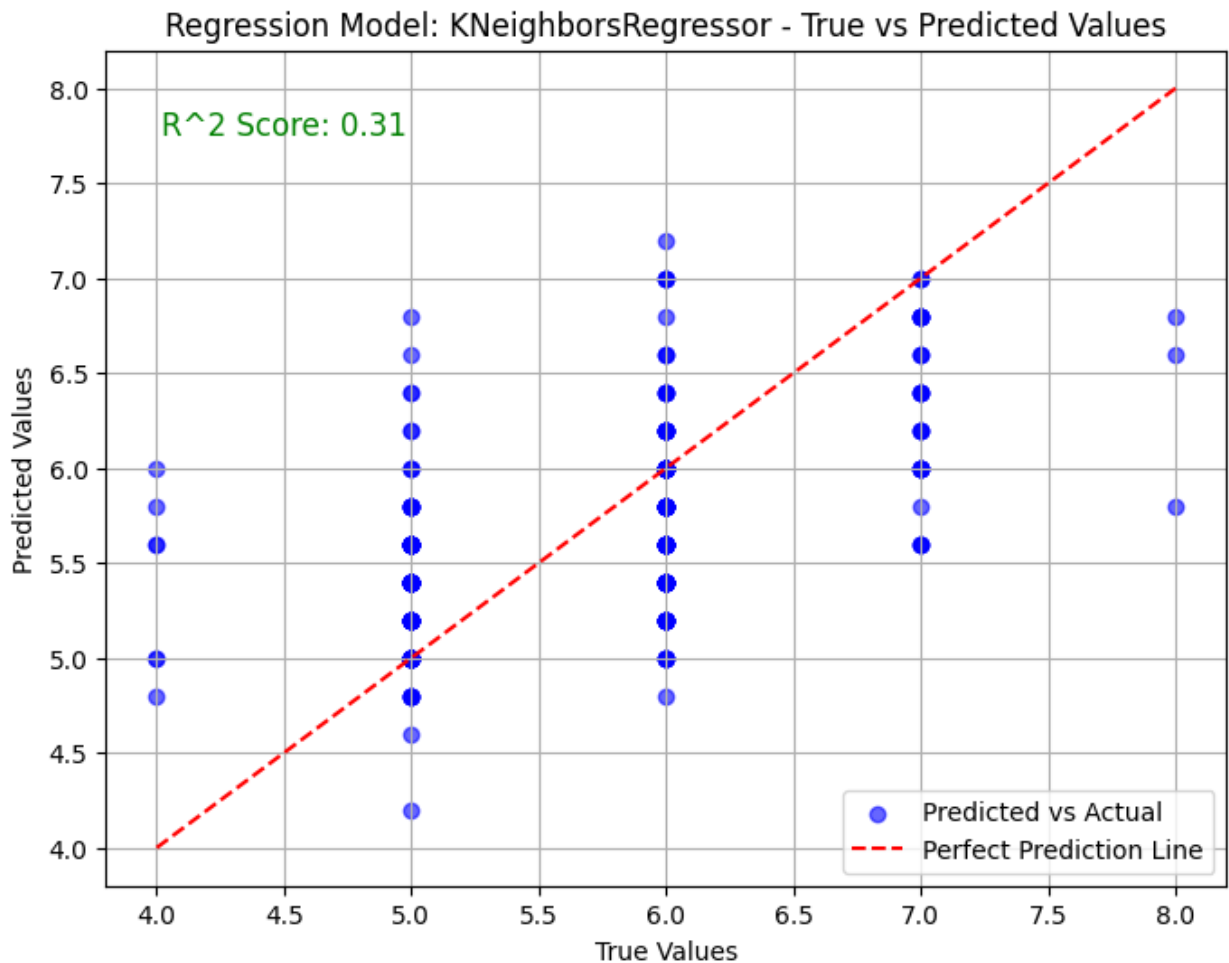
knn.fit(X_train, y_train)
# Make predictions
y_pred = knn.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('KNeighborsRegressor', y_test,
```

```

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): 0.4734
Mean Squared Error (MSE): 0.3971
Root Mean Squared Error (RMSE): 0.6301
R^2 Score: 0.3052

```

Hybridparamets optimization

RandomForestRegressor with grid search

```

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)

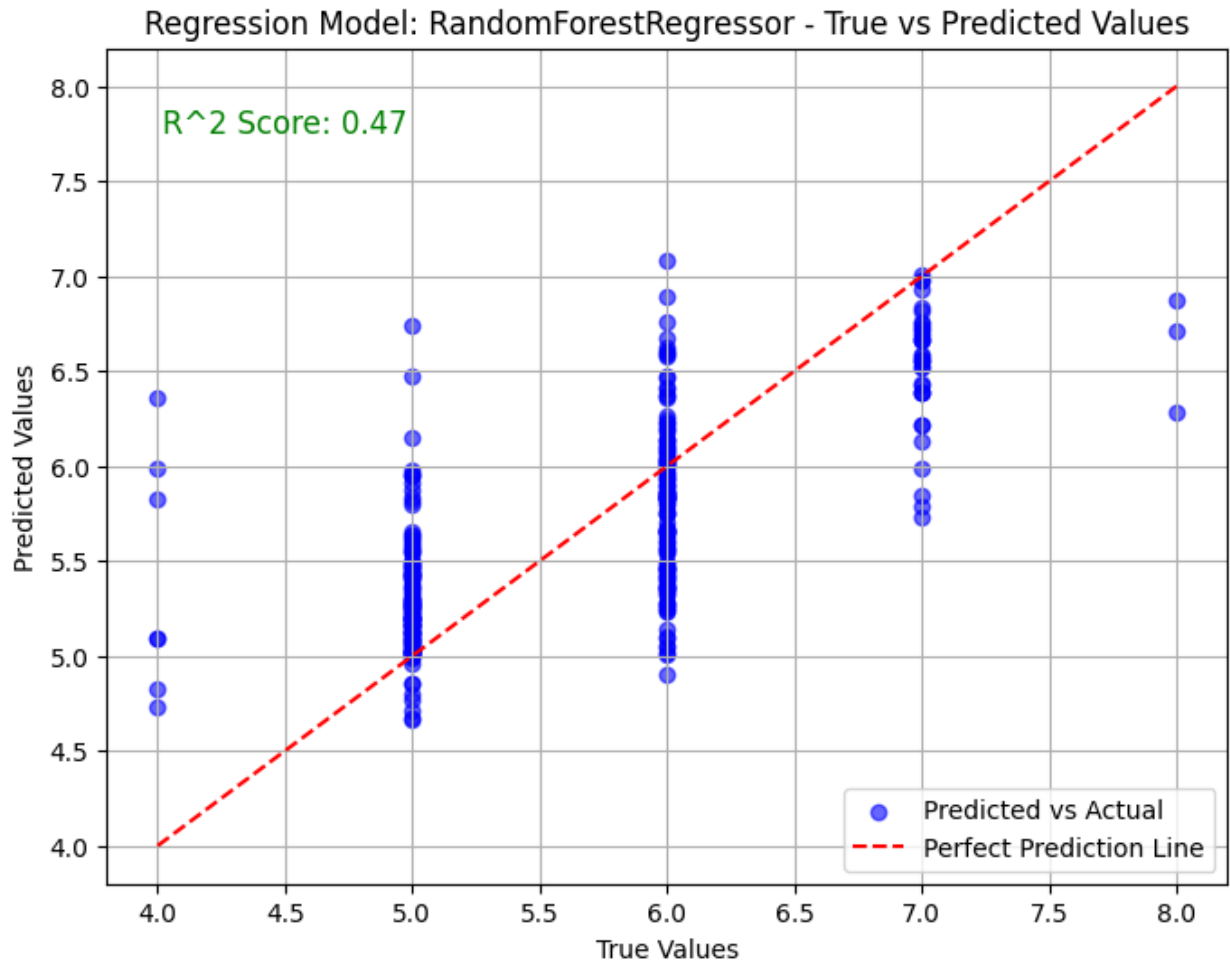
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)

# 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): 0.4154
Mean Squared Error (MSE): 0.3026
Root Mean Squared Error (RMSE): 0.5501
 R^2 Score: 0.4705
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

RandomForestRegressor with RandomizedSearchCV

```
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)

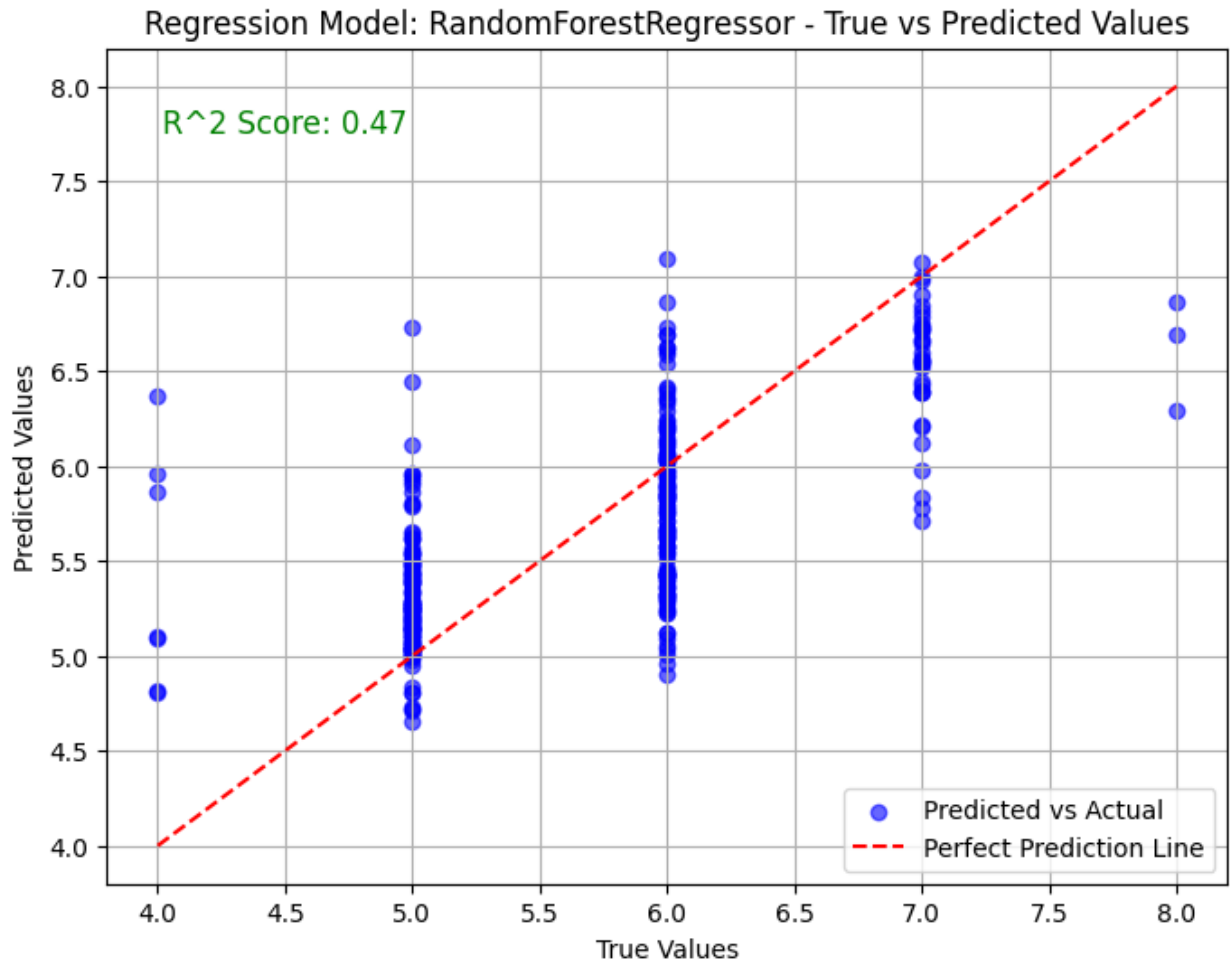
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)

# 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): 0.4159
Mean Squared Error (MSE): 0.3034
Root Mean Squared Error (RMSE): 0.5508
R² Score: 0.4691
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

SVM grid search

```
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)

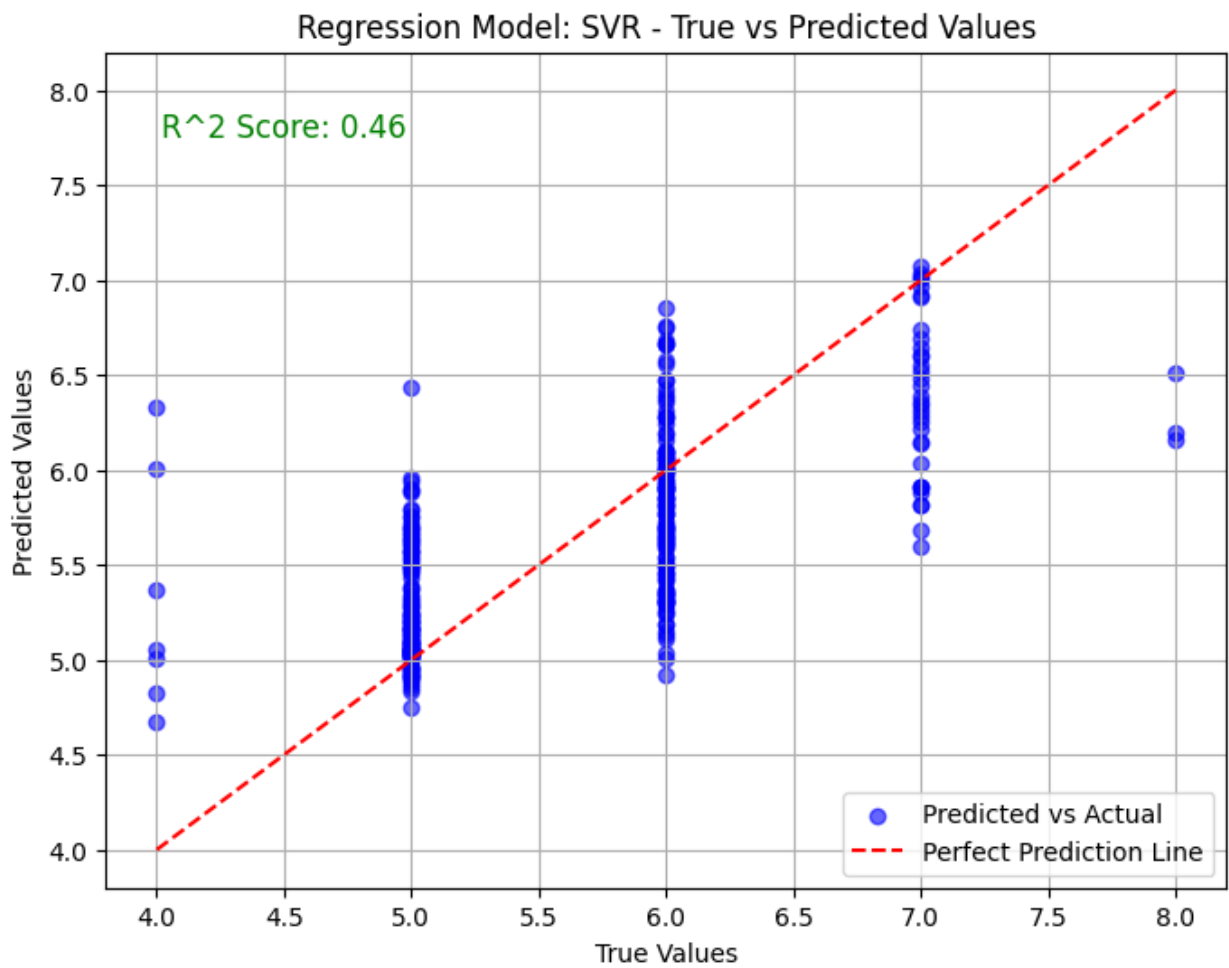
print("Best Parameters (Grid Search):", grid_search.best_params_)
y_pred = grid_search.best_estimator_.predict(X_test)

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': 1, 'kernel': 'rbf'}

```



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

Multi-task

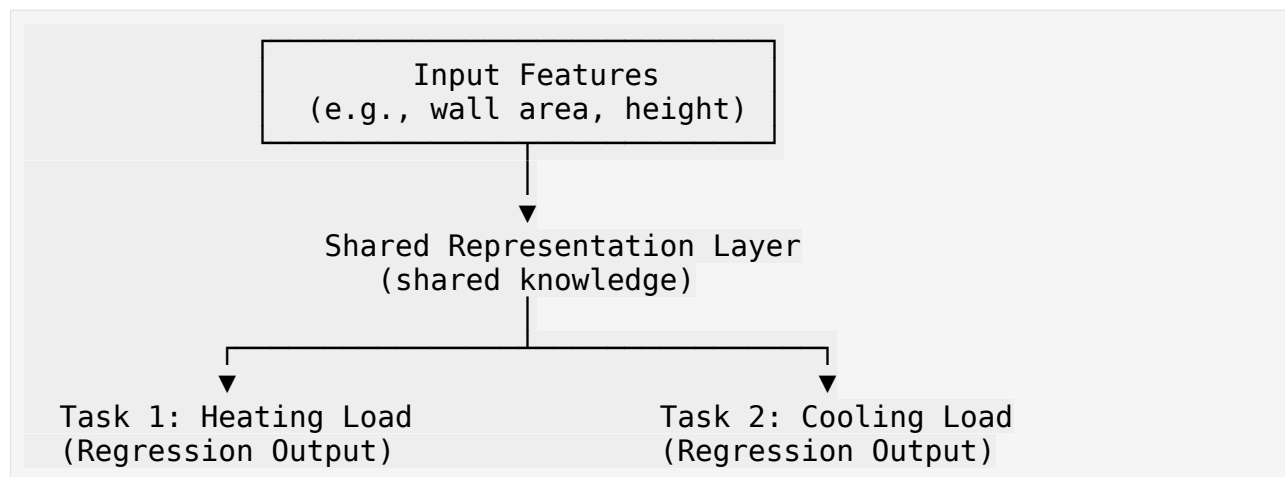
What is Multi-Task Learning (MTL)?

Multi-task learning is a machine learning approach where a single model is trained to perform multiple related tasks simultaneously. The idea is that shared learning between related tasks helps the model generalize better and learn useful representations.

Example Energy Efficiency Dataset

Kaggle Link

<https://www.kaggle.com/code/imjielingchen/building-energy-efficiency-prediction/input>



The dataset contains 768 samples of residential buildings with 8 features and 2 targets. All the buildings have the same volume but different surface areas and dimensions. The materials used for each of the 18 elements are the same for all building forms.

Specifically:

X1 Relative Compactness

X2 Surface Area

X3 Wall Area

X4 Roof Area

X5 Overall Height

X6 Orientation

X7 Glazing Area. Three types of glazing areas, which are expressed as percentages of the floor area: 10%, 25%, and 40%.

X8 Glazing Area Distribution. (1) uniform: with 25% glazing on each side, (2) north: 55% on the north side and 15% on each of the other sides, (3) east: 55% on the east side and 15% on each of the other sides, (4) south: 55% on the south side and 15% on each of the other sides, and (5) west: 55% on the west side and 15% on each of the other sides.

y1 Heating Load

y2 Cooling Load

Libraries

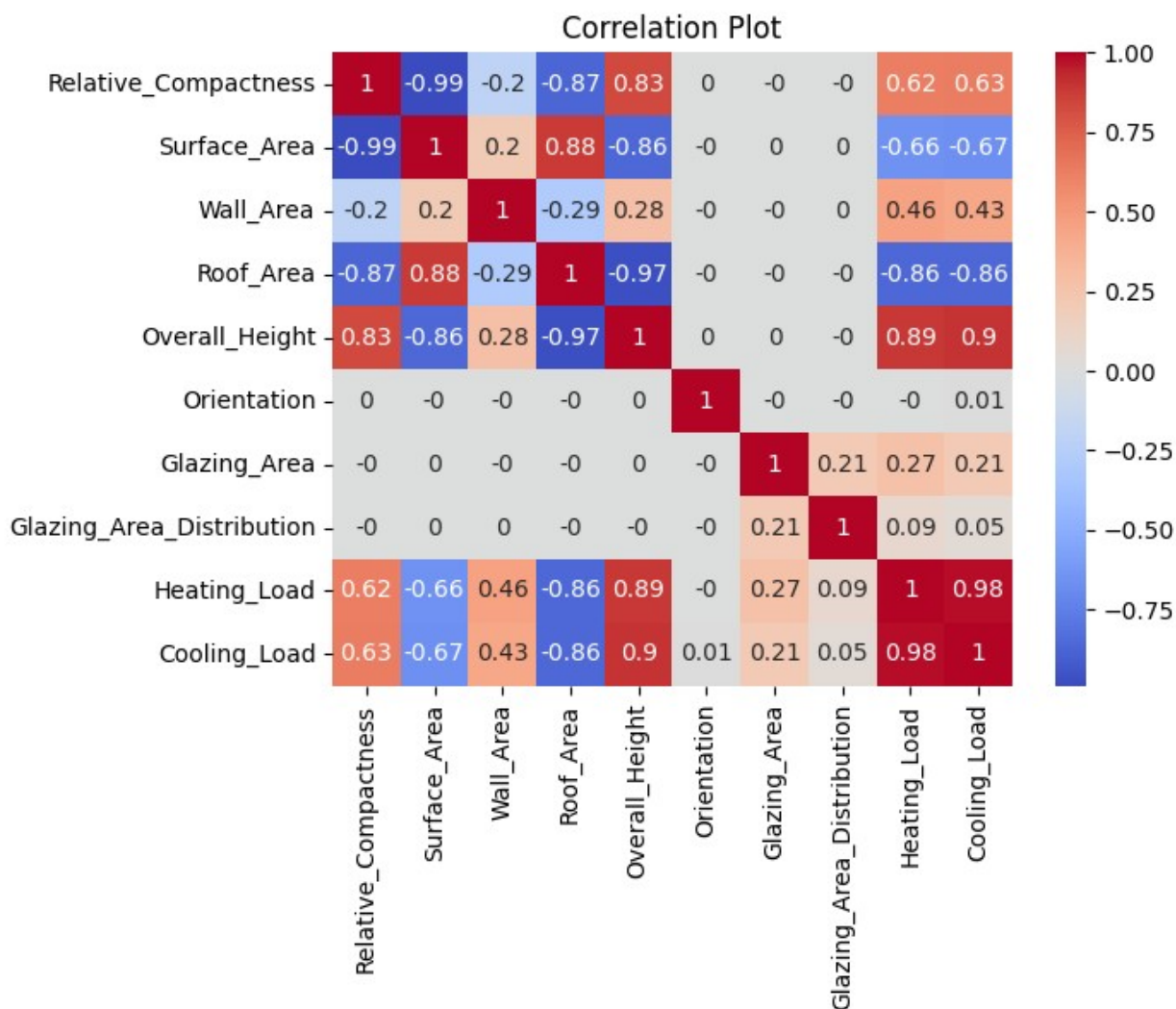
```
# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import numpy as np
import matplotlib.pyplot as plt
```

```
df=pd.read_csv("energy_efficiency_data.csv")
```

```
df.isnull().sum()
```

Relative_Compactness	0
Surface_Area	0
Wall_Area	0
Roof_Area	0
Overall_Height	0
Orientation	0
Glazing_Area	0
Glazing_Area_Distribution	0
Heating_Load	0
Cooling_Load	0
dtype:	int64

```
corr = df.corr().round(2)
sns.heatmap(corr, cmap='coolwarm', annot=True)
plt.title('Correlation Plot')
plt.show()
```



```
df.columns
```

```
Index(['Relative_Compactness', 'Surface_Area', 'Wall_Area',
      'Roof_Area',
      'Overall_Height', 'Orientation', 'Glazing_Area',
      'Glazing_Area_Distribution', 'Heating_Load', 'Cooling_Load'],
      dtype='object')
```

```
X = df.drop(['Heating_Load', 'Cooling_Load'], axis=1)
y = df[['Heating_Load', 'Cooling_Load']]
```

```
# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Multi-task using RandomForestRegressor

```
# Define multi-output RFR model
rfr = RandomForestRegressor(n_estimators=200, random_state=42)
multi_rfr = MultiOutputRegressor(rfr)

# Train
multi_rfr.fit(X_train, y_train)

# Predict
y_pred = multi_rfr.predict(X_test)

# Evaluate
mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y.shape[1])]

# Display results
for i, col in enumerate(y.columns):
    print(f"\nTarget: {col}")
    print(f"MAE: {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R²: {r2[i]:.3f}")

# ----- SCATTER PLOTS -----
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, col in enumerate(y.columns):
    axes[i].scatter(y_test.iloc[:, i], y_pred[:, i], alpha=0.7,
edgecolors='k')
    axes[i].plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
[y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
'r--', lw=2)
    axes[i].set_xlabel('Actual Values')
    axes[i].set_ylabel('Predicted Values')
    axes[i].set_title(f"{col} - Actual vs Predicted (DTR)")
    axes[i].grid(True)

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

```
Target: Heating_Load
MAE: 0.355
RMSE: 0.493
```

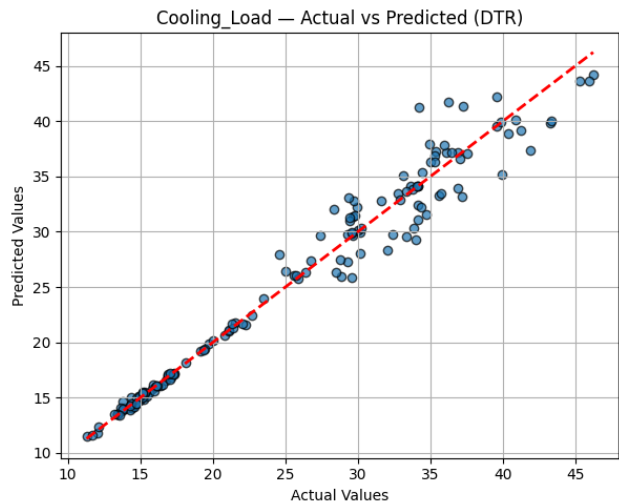
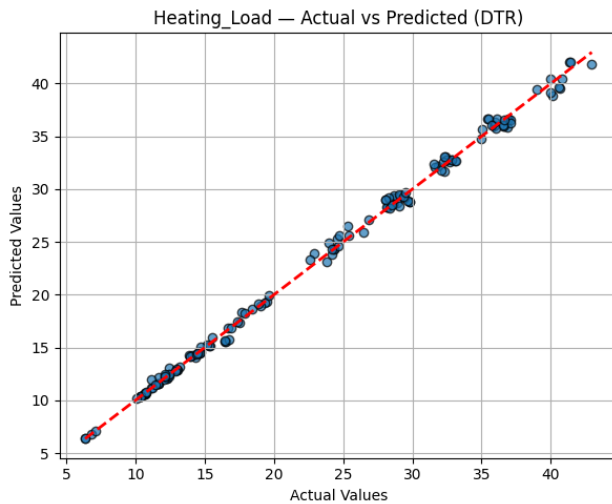
R^2 : 0.998

Target: Cooling_Load

MAE: 1.074

RMSE: 1.740

R^2 : 0.967



Multi-task using DecisionTreeRegressor

```
# Define multi-output Decision Tree Regressor
dtr = DecisionTreeRegressor(random_state=42)
multi_dtr = MultiOutputRegressor(dtr)

# Train model
multi_dtr.fit(X_train, y_train)

# Predict
y_pred = multi_dtr.predict(X_test)

# Evaluate performance
mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y.shape[1])]

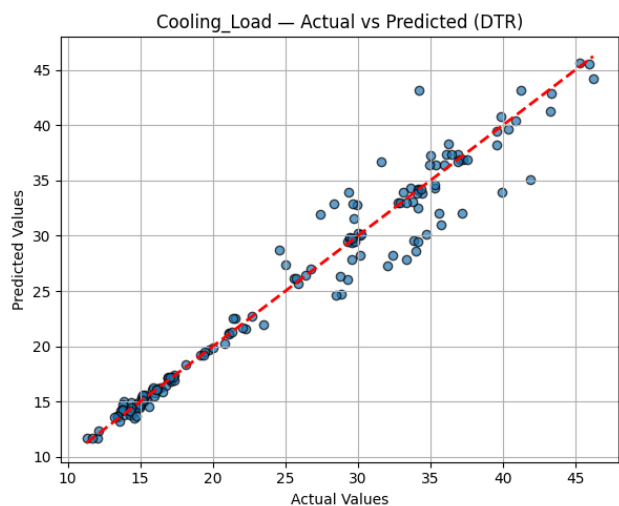
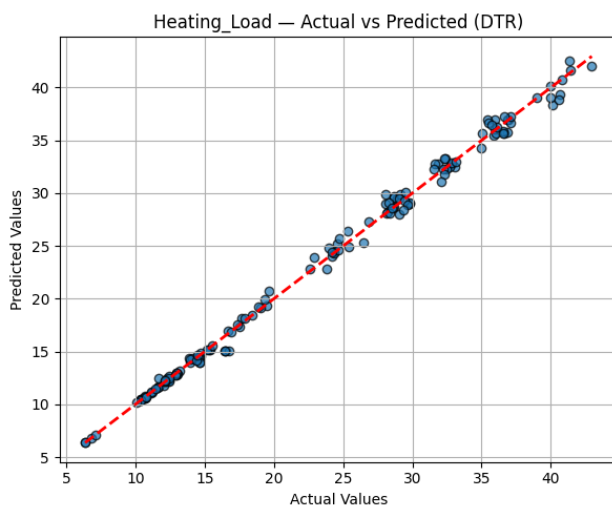
# Display metrics
for i, col in enumerate(y.columns):
    print(f"\nTarget: {col}")
    print(f"MAE: {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f" $R^2$ : {r2[i]:.3f}")
```

```
# ----- SCATTER PLOTS -----
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, col in enumerate(y.columns):
    axes[i].scatter(y_test.iloc[:, i], y_pred[:, i], alpha=0.7,
edgecolors='k')
    axes[i].plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
                [y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
                'r--', lw=2)
    axes[i].set_xlabel('Actual Values')
    axes[i].set_ylabel('Predicted Values')
    axes[i].set_title(f"{col} — Actual vs Predicted (DTR)")
    axes[i].grid(True)

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

Target: Heating_Load
MAE: 0.425
RMSE: 0.621
 R^2 : 0.996

Target: Cooling_Load
MAE: 1.155
RMSE: 2.017
 R^2 : 0.956



Multi-task using SVR

```
svr = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)
multi_svr = MultiOutputRegressor(svr)

# Train model
```

```

multi_svr.fit(X_train, y_train)

# Predict
y_pred = multi_svr.predict(X_test)

# Evaluate
mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y.shape[1])]

# Display metrics
for i, col in enumerate(y.columns):
    print(f"\nTarget: {col}")
    print(f"MAE: {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R²: {r2[i]:.3f}")

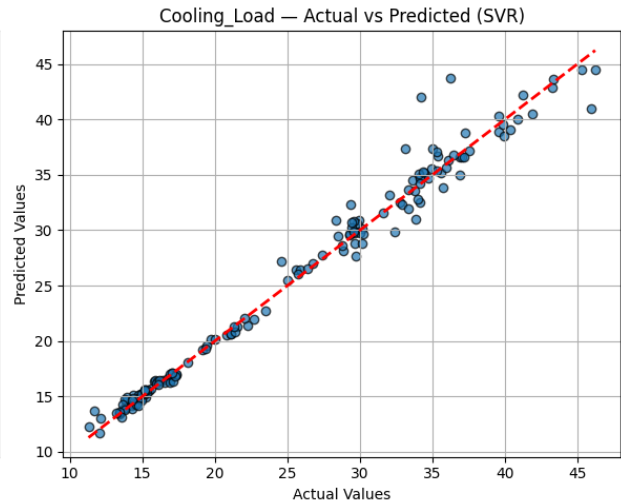
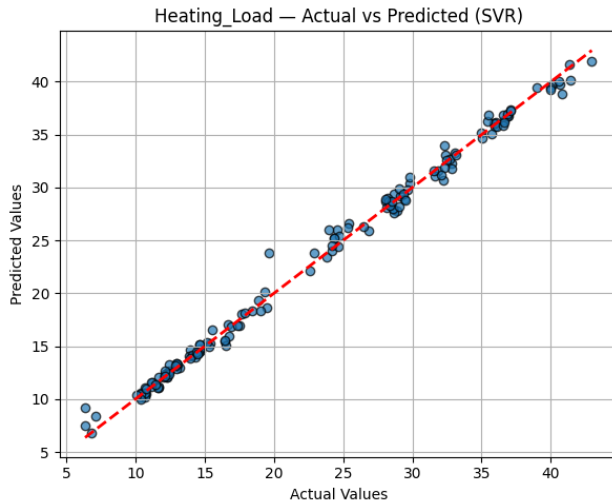
# ----- SCATTER PLOTS -----
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, col in enumerate(y.columns):
    axes[i].scatter(y_test.iloc[:, i], y_pred[:, i], alpha=0.7,
edgecolors='k')
    axes[i].plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
[y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
'r--', lw=2)
    axes[i].set_xlabel('Actual Values')
    axes[i].set_ylabel('Predicted Values')
    axes[i].set_title(f"{col} - Actual vs Predicted (SVR)")
    axes[i].grid(True)

plt.tight_layout()
plt.show()

```

Target: Heating_Load
MAE: 0.526
RMSE: 0.752
R²: 0.995

Target: Cooling_Load
MAE: 0.765
RMSE: 1.335
R²: 0.981



Multi-task KNeighborsRegressor

```
# Define and train KNN model
knn = KNeighborsRegressor(n_neighbors=5, weights='distance') # You
can change to 'uniform' if needed
knn.fit(X_train, y_train)

# Predict
y_pred = knn.predict(X_test)

# Evaluate performance
mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y.shape[1])]

# Display metrics
for i, col in enumerate(y.columns):
    print(f"\nTarget: {col}")
    print(f"MAE: {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R²: {r2[i]:.3f}")

# ----- SCATTER PLOTS -----
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, col in enumerate(y.columns):
    axes[i].scatter(y_test.iloc[:, i], y_pred[:, i], alpha=0.7,
edgecolors='k')
    axes[i].plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
[y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
'r--', lw=2)
    axes[i].set_xlabel('Actual Values')
    axes[i].set_ylabel('Predicted Values')
```

```
axes[i].set_title(f"{col} — Actual vs Predicted (KNN)")  
axes[i].grid(True)
```

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

Target: Heating_Load

MAE: 2.131

RMSE: 2.735

R^2 : 0.928

Target: Cooling_Load

MAE: 1.787

RMSE: 2.240

R^2 : 0.946

