In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import QuantileTransformer
from sklearn.ensemble import RandomForestRegressor
```

In [ ]:
```
# Load the dataset
df = pd.read_csv(r'C:\Users\Orakul\Documents\dumps\Life Expectancy Data.csv')
```

In [ ]:
```
# Display the first few rows of the dataset
(df.head()).T
```

Out[ ]:

	0	1	2	3	4
Country	Afghanistan	Afghanistan	Afghanistan	Afghanistan	Afghanistan
Year	2015	2014	2013	2012	2011
Status	Developing	Developing	Developing	Developing	Developing
Life expectancy	65.0	59.9	59.9	59.5	59.2
Adult Mortality	263.0	271.0	268.0	272.0	275.0
infant deaths	62	64	66	69	71
Alcohol	0.01	0.01	0.01	0.01	0.01
percentage expenditure	71.279624	73.523582	73.219243	78.184215	7.097109
Hepatitis B	65.0	62.0	64.0	67.0	68.0
Measles	1154	492	430	2787	3013
BMI	19.1	18.6	18.1	17.6	17.2
under-five deaths	83	86	89	93	97
Polio	6.0	58.0	62.0	67.0	68.0
Total expenditure	8.16	8.18	8.13	8.52	7.87
Diphtheria	65.0	62.0	64.0	67.0	68.0
HIV/AIDS	0.1	0.1	0.1	0.1	0.1
GDP	584.25921	612.696514	631.744976	669.959	63.537231
Population	33736494.0	327582.0	31731688.0	3696958.0	2978599.0
thinness 1-19 years	17.2	17.5	17.7	17.9	18.2
thinness 5-9 years	17.3	17.5	17.7	18.0	18.2
Income composition of resources	0.479	0.476	0.47	0.463	0.454
Schooling	10.1	10.0	9.9	9.8	9.5

In [ ]:
```
# Standardize the column names

df.columns = df.columns.str.lower().str.strip().str.replace(' ', '_')

df.columns
```

Out[ ]:
```
Index(['country', 'year', 'status', 'life_expectancy', 'adult_mortality',
       'infant_deaths', 'alcohol', 'percentage_expenditure', 'hepatitis_b',
       'measles', 'bmi', 'under-five_deaths', 'polio', 'total_expenditure',
       'diphtheria', 'hiv/aids', 'gdp', 'population', 'thinness__1-19_years',
       'thinness_5-9_years', 'income_composition_of_resources', 'schooling'],
      dtype='object')
```

In [ ]:
```
# Dropping all columns except gdp, percentage_expenditure, total_expenditure, incom
columns_to_keep = ['gdp', 'percentage_expenditure', 'total_expenditure', 'income_co

df = df[columns_to_keep]
pd.DataFrame(df.columns)
```

Out[ ]:

	0
0	gdp
1	percentage_expenditure
2	total_expenditure
3	income_composition_of_resources
4	schooling
5	life_expectancy

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

Out[ ]:

	gdp	percentage_expenditure	total_expenditure	income_composition_of_resources
0	584.259210	71.279624	8.16	0.479
1	612.696514	73.523582	8.18	0.476
2	631.744976	73.219243	8.13	0.470
3	669.959000	78.184215	8.52	0.463
4	63.537231	7.097109	7.87	0.454
5	553.328940	79.679367	9.20	0.448
6	445.893298	56.762217	9.42	0.434
7	373.361116	25.873925	8.33	0.433
8	369.835796	10.910156	6.73	0.415
9	272.563770	17.171518	7.43	0.405

In [ ]:
```
# Checking the data types

pd.DataFrame(df.dtypes)
```

Out[ ]:

	0
gdp	float64
percentage_expenditure	float64
total_expenditure	float64
income_composition_of_resources	float64
schooling	float64
life_expectancy	float64

In [ ]:
```
# Checking For Missing Values

pd.DataFrame(df.isnull().sum())
```

Out[ ]:

	0
gdp	448
percentage_expenditure	0
total_expenditure	226
income_composition_of_resources	167
schooling	163
life_expectancy	10

In [ ]:
```
# Checking the Data Types of all the Misssing Values
missing_values = df.isnull().sum()
missing_values.dtype
```

Out[ ]:
```
dtype('int64')
```

In [ ]:
```
# Filling All the Missing Values With zeros

df = df.fillna(0)
```

In [ ]:
```
# Verifying the filling

pd.DataFrame(df.isnull().sum())
```

Out[ ]:

	0
gdp	0
percentage_expenditure	0
total_expenditure	0
income_composition_of_resources	0
schooling	0
life_expectancy	0

In [ ]:
```
# Checking For Duplicate

df.duplicated().sum()
```

Out[ ]: 7

In [ ]:
```
# Checking the rows of the duplicates

duplicate_rows = df[df.duplicated()]

duplicate_rows
```

Out[ ]:

	gdp	percentage_expenditure	total_expenditure	income_composition_of_resources	sc
714	0.0	0.0	0.0	0.0	
715	0.0	0.0	0.0	0.0	
2381	0.0	0.0	0.0	0.0	
2386	0.0	0.0	0.0	0.0	
2417	0.0	0.0	0.0	0.0	
2419	0.0	0.0	0.0	0.0	
2421	0.0	0.0	0.0	0.0	

In [ ]:
```
# Dropping Duplicates

df = df.drop_duplicates()
```

In [ ]:
```
# Checking For Duplicate

df.duplicated().sum()
```

Out[ ]: 0

In [ ]:
```
# Checking the correlation

```

```
correlation_matrix = df.corr()

# Display correlation matrix
pd.DataFrame(correlation_matrix)
```

Out[ ]:

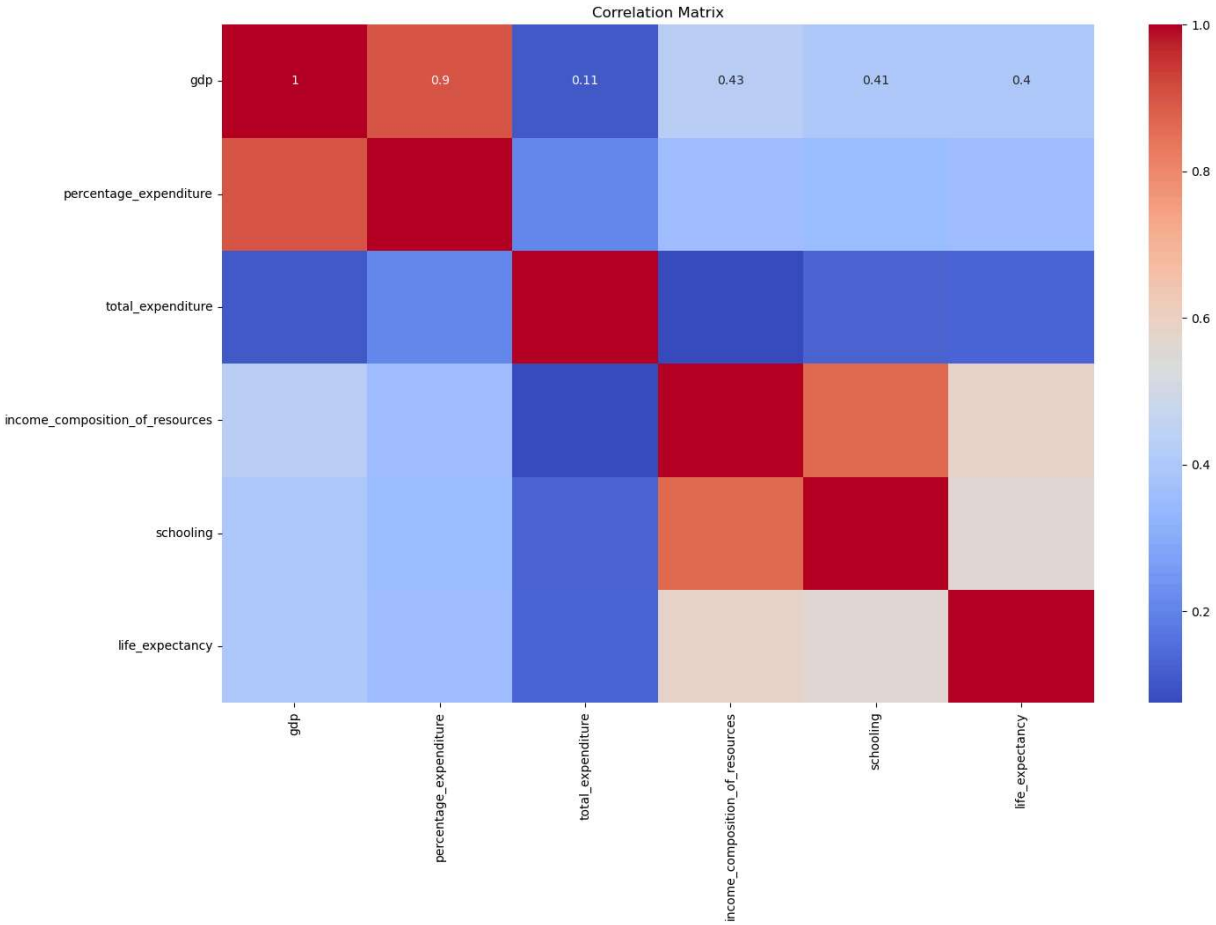
	gdp	percentage_expenditure	total_expenditure	inco
gdp	1.000000	0.901661	0.105674	
percentage_expenditure	0.901661	1.000000	0.203024	
total_expenditure	0.105674	0.203024	1.000000	
income_composition_of_resources	0.425788	0.362955	0.074843	
schooling	0.406120	0.355665	0.128218	
life_expectancy	0.403698	0.358242	0.134982	

In [ ]:

```
# Visualize the correlation matrix

correlation_matrix = np.corrcoef(df, rowvar=False)

# Create a heatmap using seaborn
plt.figure(figsize=(16, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', xticklabels=df.columns
plt.title('Correlation Matrix')
plt.show()
```



Based on the correlations provided:

1. **GDP and Percentage Expenditure**: There is a very strong positive correlation of 0.90 between GDP and percentage expenditure. This suggests that countries with higher GDP tend to allocate a larger percentage of their GDP towards various expenditures, possibly including healthcare, infrastructure, and social services. For business intelligence and economic planning, this correlation indicates that economic growth could drive increased investment across multiple sectors.

2. **Income Composition of Resources and Schooling**: There is a strong positive correlation of 0.87 between income composition of resources and schooling. This implies that countries with higher income composition also tend to have higher levels of education (schooling). This correlation is crucial for understanding human development and economic disparities across different regions.

3. **Schooling and Life Expectancy**: The correlation of 0.56 between schooling and life expectancy suggests that education plays a significant role in determining life expectancy. Countries with better education systems tend to have populations with longer life expectancies, indicating the importance of education in promoting public health and well-being.

Suggestions and Recommendations:

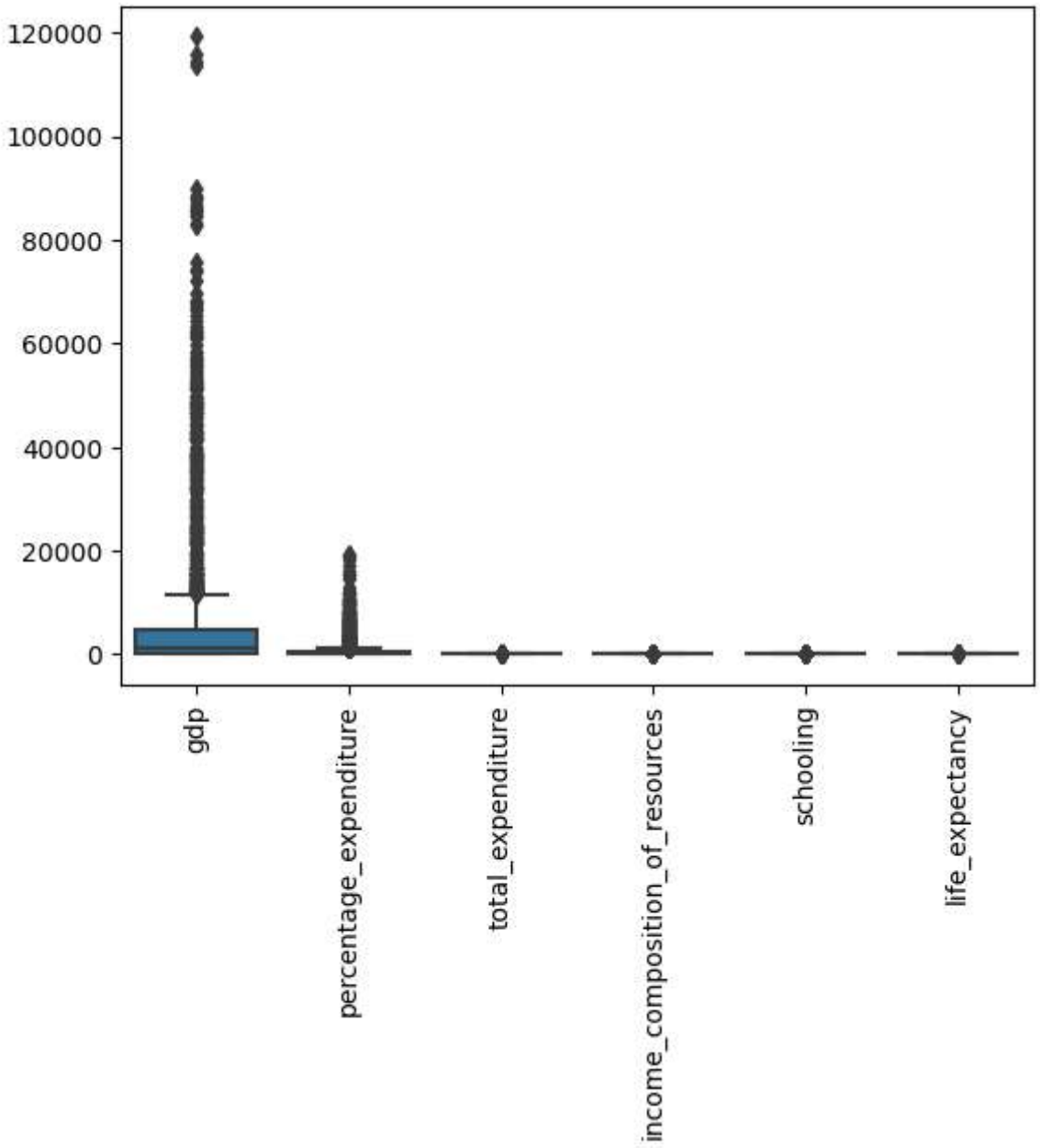
1. **Promoting Economic Growth**: Foster policies that stimulate economic growth, as higher GDP correlates strongly with increased expenditure across various sectors. This could involve incentivizing investments, improving infrastructure, and supporting entrepreneurship.

2. **Investment in Education**: Prioritize investments in education to enhance human capital development. Education not only correlates with income levels but also significantly impacts social mobility and quality of life indicators like life expectancy.

3. **Enhancing Income Composition**: Focus on improving income composition through equitable economic policies, job creation, and skills development programs. A balanced income distribution can contribute to overall economic stability and social cohesion.

By leveraging these correlations and recommendations, stakeholders in business intelligence and economic policy-making can better formulate strategies to promote sustainable development, improve human capital, and foster inclusive economic growth.

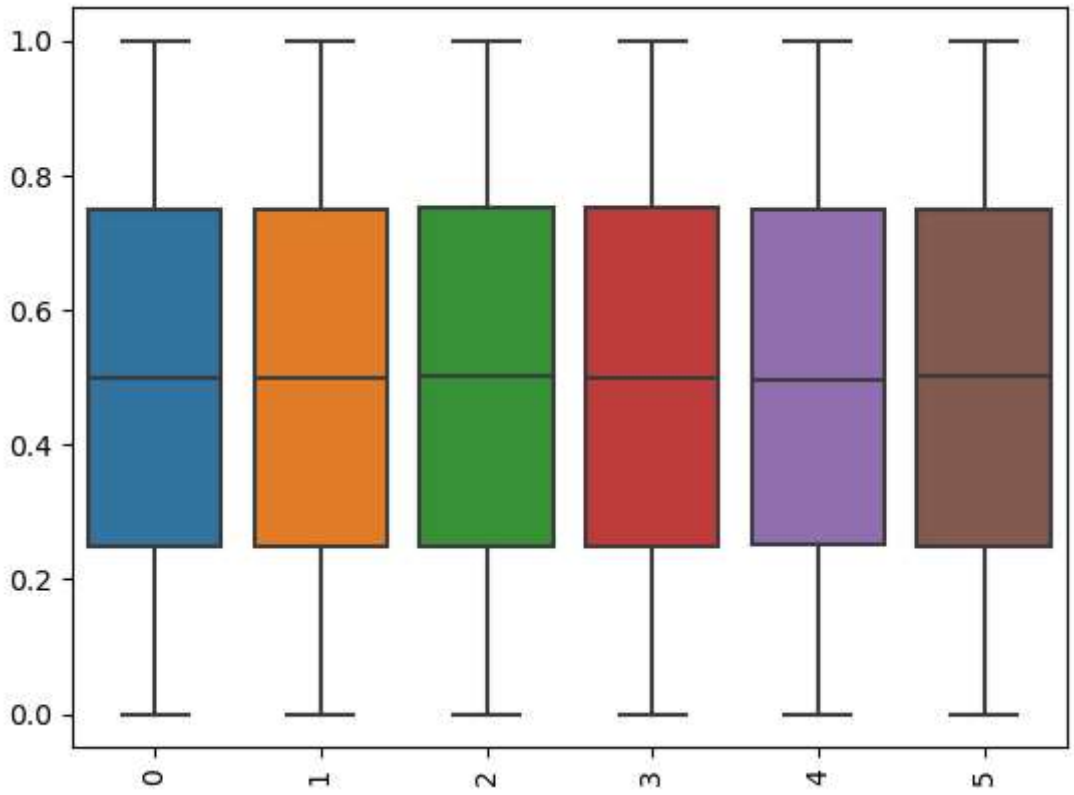
In [ ]:
```
# Checking For Outliers
sns.boxplot(data=df)
plt.xticks(rotation=90)
plt.show()
```



In [ ]:
```
# Initialize the QuantileTransformer to handle the outliers as seen from the the bo
quantile_transformer = QuantileTransformer(output_distribution='uniform', random_st

# Apply the transformation
transformed_data  = quantile_transformer.fit_transform(df)
```

In [ ]:
```
# Verifying the effect of QuantileTransformer
sns.boxplot(data=transformed_data)
plt.xticks(rotation=90)
plt.show()
```



In [ ]:
```
# Convert the transformed data back to a DataFrame, preserving the original column
df_transformed = pd.DataFrame(transformed_data, columns=df.columns)
df_transformed
```

Out[ ]:

	gdp	percentage_expenditure	total_expenditure	income_composition_of_resource
0	0.402399	0.511982	0.821880	0.26826
1	0.406413	0.516781	0.825325	0.26326
2	0.412429	0.516198	0.819820	0.25625
3	0.422755	0.524442	0.868869	0.24424
4	0.190563	0.265817	0.804285	0.23123
...	...	...	...	.
2926	0.357637	0.000000	0.730230	0.17149
2927	0.356877	0.000000	0.652152	0.18318
2928	0.187582	0.000000	0.653654	0.19319
2929	0.391426	0.000000	0.594595	0.19319
2930	0.390768	0.000000	0.724725	0.20070

2931 rows × 6 columns

In [ ]:
```
# Identify predictors and target variable
predictors = df_transformed.drop(columns=['life_expectancy'])
target = df_transformed['life_expectancy']
```

In [ ]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(predictors, target, test_size=0
```

In [ ]:
```
# Initializing and training a Random Forest Regressor model for pricing strategy

model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

In [ ]:
```
# Predict on training data
y_pred_train = model.predict(X_train)

pd.DataFrame(y_pred_train)
```

Out[ ]:

	0
0	0.251041
1	0.165826
2	0.896842
3	0.711802
4	0.811972
...	...
2339	0.916166
2340	0.119891
2341	0.209510
2342	0.832497
2343	0.166807

2344 rows × 1 columns

In [ ]:
```
# Calculate bias
bias = np.mean((y_pred_train - y_train)**2)

# Calculate variance
y_pred_mean = np.mean(y_pred_train)
variance = np.mean((y_pred_train - y_pred_mean)**2)
```

In [ ]:
```
print(f'New Bias: {bias:.2f}')
print(f'New Variance: {variance:.2f}')
```

```
New Bias: 0.00
New Variance: 0.07
```

Bias:

Definition: Bias measures how far off the predictions are from the actual values on average. A bias of 0.00 indicates that, on average, the model's predictions match the actual values without systematic overestimation or underestimation. Interpretation: A lower bias generally suggests that the model is fitting the data well and is not biased towards particular outcomes. Variance:

Definition: Variance measures the variability of model predictions for a given data point. A variance of 0.07 indicates that the model's predictions can vary by this amount around the mean prediction. Interpretation: Higher variance can indicate that the model is sensitive to small fluctuations in the training data, possibly overfitting and performing inconsistently on new data. Importance for Model Evaluation: Bias and variance trade-off: Balancing bias and variance is crucial for achieving a model that generalizes well to new data. High bias indicates underfitting, where the model is too simplistic to capture the underlying patterns in the data. High variance suggests overfitting, where the model is too complex and captures noise in the training data, leading to poor performance on unseen data.

Optimal model performance: Ideally, you want to minimize both bias and variance to achieve a model that accurately predicts the target variable without being too simple or too complex.

In [ ]:
```
# Make predictions on test
y_pred_test = model.predict(X_test)
pd.DataFrame(y_pred_test)
```

Out[ ]:

	0
0	0.547718
1	0.319049
2	0.554439
3	0.602708
4	0.924224
...	...
582	0.519615
583	0.494791
584	0.675571
585	0.095417
586	0.785626

587 rows × 1 columns

In [ ]:
```
# Checking Test Scores metrics
mse = mean_squared_error(y_test, y_pred_test)
mae = mean_absolute_error(y_test, y_pred_test)
rmse = sqrt(mse)
r2 = r2_score(y_test, y_pred_test)

# Print metrics
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R²): {r2:.2f}")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[29], line 4
      2 mse = mean_squared_error(y_test, y_pred_test)
      3 mae = mean_absolute_error(y_test, y_pred_test)
----> 4 rmse = sqrt(mse)
      5 r2 = r2_score(y_test, y_pred_test)
      7 # Print metrics

NameError: name 'sqrt' is not defined
```

Mean Squared Error (MSE):

Definition: MSE measures the average squared difference between predicted values and actual values. It quantifies the average magnitude of error produced by the model. Interpretation: An MSE of 0.02 means that, on average, the squared difference between predicted and actual values is 0.02. Lower values indicate better model performance, with 0 being the perfect score. Mean Absolute Error (MAE):

Definition: MAE measures the average absolute difference between predicted values and actual values. It provides a more straightforward interpretation of error than MSE since it doesn't square the errors. Interpretation: An MAE of 0.08 means that, on average, the model's predictions differ from the actual values by 0.08 units. Like MSE, lower values indicate better model performance. Root Mean Squared Error (RMSE):

Definition: RMSE is the square root of MSE, providing a measure of the prediction error in the same units as the target variable. It is more sensitive to outliers than MAE. Interpretation: An RMSE of 0.13 means that, on average, the model's predictions differ from the actual values by 0.13 units. Again, lower values indicate better model performance. R-squared (R^2):

Definition: R^2 measures the proportion of the variance in the dependent variable (target) that is predictable from the independent variables (features). It ranges from 0 to 1, with 1 indicating perfect prediction. Interpretation: An R^2 of 0.81 means that 81% of the variance in the target variable is explained by the model. Higher values of R^2 indicate that the model fits the data well and explains a large proportion of the variability in the target variable. Practical Application: Performance Evaluation: These metrics collectively provide insights into how well your model is performing. They help you understand the accuracy, precision, and goodness of fit of your predictive model relative to the actual data.

In [ ]:
```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(model, predictors, target, cv=5, scoring='r2')
print(f'Cross-Validation R² Scores: {cv_scores}')
print(f'Mean Cross-Validation R² Score: {cv_scores.mean()}')
```

Definition: R^2 (R-squared) score is a statistical measure that indicates how well the model predictions approximate the real data points. It ranges from 0 to 1, where 1 indicates a perfect fit.

Individual Scores:

The cross-validation R^2 scores are as follows: [0.71438987, 0.76626888, 0.76725213, 0.61410414, 0.58258238]. Each score represents the R^2 value obtained from one fold (subset) of the cross-validation process. Interpretation:

A score closer to 1 suggests that the model explains a large proportion of the variance in the target variable and is a good fit. Lower scores indicate that the model's predictions do not explain as much of the variance in the target variable. Mean Cross-Validation R^2 Score: Definition: The mean cross-validation R^2 score is the average of all R^2 scores obtained from the cross-validation process.

Calculation:

Mean Cross-Validation R^2 Score: 0.6889194816652501 This value is derived by averaging the individual R^2 scores obtained from each fold of cross-validation. Significance:

The mean R^2 score provides a consolidated measure of how well the model generalizes to new data. It indicates the overall predictive power of the model across different subsets of the data. Higher mean R^2 scores generally indicate that the model is robust and performs consistently well across different samples of the dataset. Practical Use: Model Evaluation: Use cross-validation R^2 scores to evaluate and compare models. A higher mean R^2 score suggests a better-performing model with higher predictive accuracy.

In [ ]: