Feature Scaling

Presented by

Thorung Boonkaew 6510545454

Present to

Kitsana Waiyamai, Ph.D.

2/2023

01219367 Data Analytics

Department of Computer Engineering, Faculty of Engineering

Introduction

Feature scaling is a pivotal preprocessing step in machine learning, crucial for standardizing or normalizing the range of independent variables or features within datasets. By ensuring that all features contribute equally to the learning algorithm, feature scaling mitigates biases and enhances model performance. This process is indispensable across various machine learning tasks, such as classification, regression, clustering, and dimensionality reduction. In classification, for example, it enables algorithms to discern patterns more accurately, leading to precise predictions. Likewise, in regression, feature scaling facilitates convergence, aiding optimization algorithms in finding optimal solutions efficiently. Moreover, feature scaling enhances interpretability by ensuring that coefficients and weights assigned to each feature are comparable, empowering stakeholders to make informed decisions. In essence, feature scaling lays the foundation for robust and accurate machine learning models, fostering fairness, efficiency, and interpretability in predictive analytics.

Contents

Meaning and Importance of Feature Scaling

What is Feature Scaling?

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values. (Gupta_OMG, n.d.)

Importance of Feature Scaling

In machine learning, feature scaling is employed for several purposes:

- Scaling guarantees that all features are on a comparable scale and have comparable ranges. This process is known as feature normalization. This is significant because the magnitude of the features has an impact on many machine learning techniques. Larger scale features may dominate the learning process and have an excessive impact on the outcomes. You can avoid this problem and make sure that each feature contributes equally to the learning process by scaling the features.
- Algorithm performance improvement: When the features are scaled, several machines learning methods, including gradient descent-based algorithms, distance-based algorithms (such k-nearest neighbors), and support vector machines, perform better or converge more quickly. The algorithm's performance can be enhanced by scaling the features, which can hasten the convergence of the algorithm to the ideal outcome.
- Preventing numerical instability: Numerical instability can be prevented by avoiding significant scale disparities between features. Examples include distance calculations or matrix operations, where having features with radically differing scales can result in numerical overflow or underflow problems. Stable computations are ensured, and these issues are mitigated by scaling the features.
- Scaling features makes ensuring that each characteristic is given the same consideration during the learning process. Without scaling, bigger scale features could dominate learning, producing skewed outcomes. This bias is removed through scaling, which also guarantees that each feature contributes fairly to model predictions. (Gupta_OMG, n.d.)

Feature Scaling Techniques

Feature scaling encompasses a variety of techniques aimed at transforming the range of features within a dataset to a standardized form. By harmonizing the scales of features, these techniques enable machine learning algorithms to operate more effectively and efficiently. Let's delve into some of the most commonly employed feature scaling techniques:

Standardization (Z-score Scaling)

Standardization, also known as Z-score normalization, is a method for rescaling values by subtracting the mean and dividing by the standard deviation. This process transforms features to have a mean of 0 and a standard deviation of 1, making it crucial for many machine learning algorithms. Standardization ensures that features are centered around 0, which is essential for algorithms like logistic regression, SVMs, and neural networks to converge efficiently. Unlike tree-based models, which are insensitive to feature scaling, algorithms like K-Nearest Neighbors and clustering methods heavily rely on standardized features for accurate results.

$$x' = \frac{x - \bar{x}}{\sigma}$$

Where x is the original feature vector, $\bar{x}$ = average(x) is the mean of that feature vector, and σ is its standard deviation.

Min-Max Scaling

Min-Max Scaling, also known as normalization, rescales features to a fixed range, typically between 0 and 1. This technique subtracts the minimum value from each feature and then divides it by the range (maximum value minus minimum value). Min-Max Scaling is particularly useful when the distribution of features is not Gaussian and when the algorithm relies on distance measures, such as K-nearest neighbors (KNN) or support vector machines (SVM).

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Where x is the original feature vector, x' is the normalized value.

Max Absolute Scaling

Max Absolute Scaling scales features by dividing each feature by its maximum absolute value. This technique preserves the sign of the data while scaling it to the range [-1, 1]. Max Absolute Scaling is useful when the presence of outliers is expected and when preserving the sign of the data is important.

$$x_{scaled} = \frac{x}{max(|x|)}$$

Robust Scaling

Robust Scaling, as the name suggests, is robust to outliers and extreme values in the dataset. Instead of using the mean and standard deviation like standardization, robust scaling employs the median and the interquartile range (IQR). It subtracts the median from each feature and divides by the IQR. This makes robust scaling particularly useful for datasets with outliers, as it reduces their impact on the scaling process.

$$x_{rs} = \frac{x_i - Q_2(x)}{Q_3(x) - Q_1(x)}$$

where Q1(x) is the first quartile of the attribute x, Q2(x) is the median, and Q3(x) is the third quartile.

Mean Normalization

Mean Normalization centers the data around zero by subtracting the mean of each feature from its values. This technique does not scale the range of features to a specific interval but ensures that the mean of each feature is zero. Mean normalization is useful when the data distribution is not Gaussian and when preserving the relative distance between data points is important.

$$x' = \frac{x - \bar{x}}{\max(x) - \min(x)}$$

Where x is the original feature vector, $\bar{x}$ = average(x) is the mean of that feature vector.

Unit Vector Scaling

Unit Vector Scaling, also known as vector normalization, scales each feature vector to have a unit norm. This technique is particularly useful when the magnitude of features is important, but their direction is irrelevant. By scaling each feature vector to have a length of 1, unit vector scaling ensures that features contribute equally to the model regardless of their original magnitudes.

$$x' = \frac{x}{\|x\|}$$

$\|x\|$ Is the Euclidean length of the Feature Vector.

The advantages and disadvantages of each common feature scaling technique

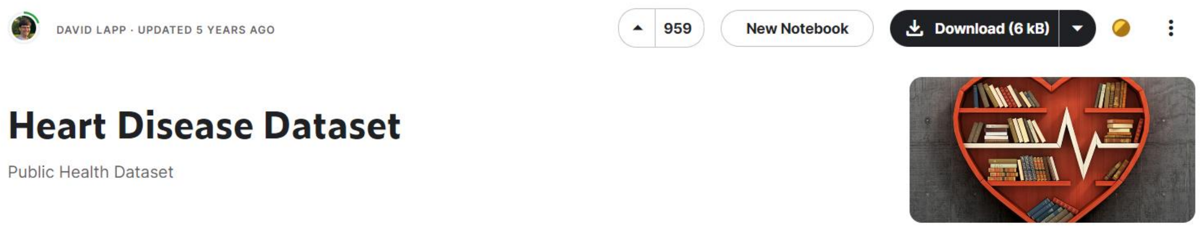
techniques	Advantages	Disadvantages
Standardization	• Robust to outliers: Less sensitive to outliers compared to Min-Max Scaling, making it more reliable in the presence of extreme values. • Preserves distribution shape: Does not constrain values to a specific range, allowing the original distribution shape to be retained.	• Unbounded values: Does not restrict values to a predefined range, which may not be suitable for algorithms expecting features within a specific range. • Assumption of Gaussian distribution: Assumes a Gaussian distribution, which may not hold true for all datasets.
Min-Max Scaling	• Enhances interpretability: Scaled features are bounded between 0 and 1, facilitating easier interpretation. • Preserves relationships: Maintains the relative distances between data points, ensuring that the original data relationships are preserved.	• Susceptible to outliers: Outliers can significantly influence the scaling process, particularly when the range of values is small. • Potential loss of precision: Precision may be compromised when dealing with datasets featuring a wide range of values.
Max Absolute Scaling	• Preservation of data sign: Scales features while retaining their sign, ensuring that directional information is retained. • Simplified implementation: Dividing by the maximum	• Lack of data standardization: Features are not centered around zero, which may impact the performance of algorithms relying on standardized data.

	absolute value provides a straightforward scaling mechanism.	• Sensitivity to outliers: Outliers with large absolute values can disproportionately influence the scaling process, potentially distorting the scaled data distribution.
Robust Scaling	• Resilience to outliers: Utilizes robust statistics such as the median and interquartile range, rendering it less susceptible to the influence of outliers. • Applicability to skewed distributions: Suitable for datasets with skewed distributions, as it does not impose distribution assumptions.	• Reduced interpretability: Scaling based on median and IQR may obscure the interpretability of the features. • Limited adoption: Not as widely utilized as Min-Max Scaling or Standardization, potentially limiting its applicability in practice.
Mean Normalization	• Centers data around zero: Useful when the mean of features holds significance, facilitating a standardized reference point. • Maintains relative distances: Ensures that the relative distances between data points remain unchanged post-scaling.	• Lack of value bounding: Does not restrict values to a specific interval, potentially posing challenges for algorithms expecting features within predefined ranges. • Vulnerability to outliers: Outliers can skew the mean and compromise the effectiveness of mean normalization.
Unit Vector Scaling	• Retention of feature direction: Scales features to have unit norm, preserving their directionality and ensuring consistent contribution to the model. • Equal feature contribution: Ensures that each feature contributes equally to the model, irrespective of its original magnitude.	• Limited applicability: May not be suitable for all algorithms, particularly those where feature magnitude is of lesser significance. • Outlier susceptibility: Outliers can still exert influence on feature directionality, potentially affecting model performance.

Methodology Outline

In this study, we aim to compare the performance of different feature scaling techniques using a dataset containing various features with different scales. Our evaluation will focus on the performance of logistic regression models trained on scaled data using different feature scaling techniques.

The dataset utilized for this comparison study is “Heart Disease Dataset” from (LAPP, 2019)


DAVID LAPP · UPDATED 5 YEARS AGO

959 New Notebook Download (6 kB)

Heart Disease Dataset

Public Health Dataset

Dataset Description:

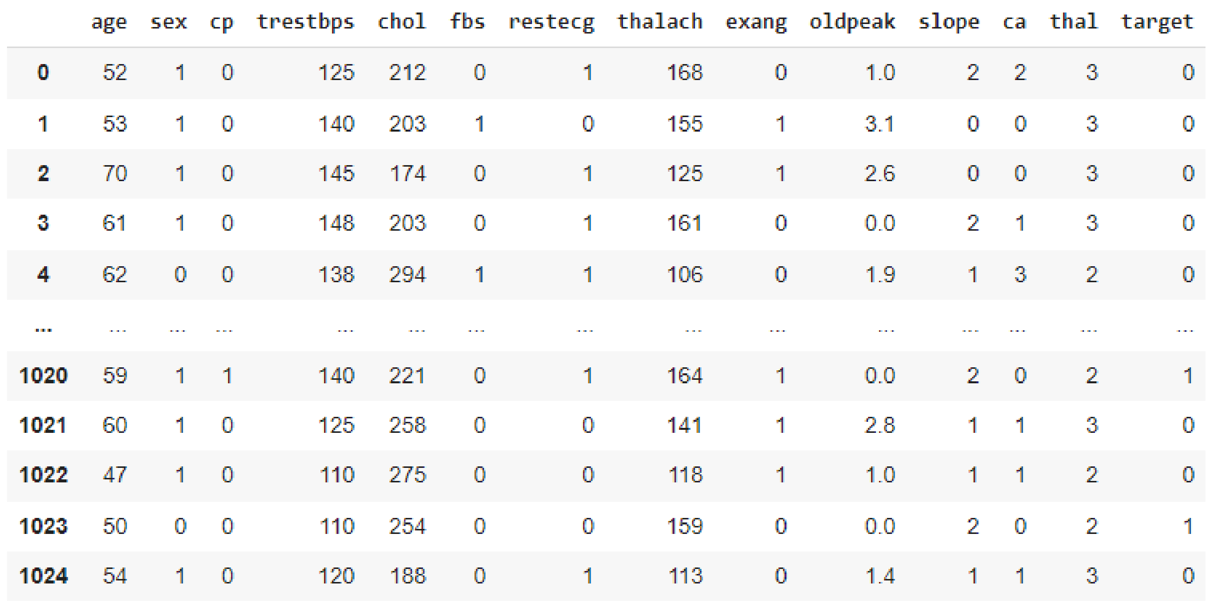
This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease.

Attribute Information:

1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholestoral in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)
8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
11. the slope of the peak exercise ST segment
12. number of major vessels (0-3) colored by flourosopy
13. thal: 0 = normal; 1 = fixed defect; 2 = reversable defect

The names and social security numbers of the patients were recently removed from the database, replaced with dummy values.

Dataset before pre-processed:



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

Logistic Regression:

Logistic regression is a statistical method used in regression analysis to estimate the parameters of a logistic model for binary classification tasks. It involves predicting the probability of an event occurring based on a given set of independent variables. The logistic function, with its distinctive S-shaped curve, transforms a linear combination of input features into a probability value between 0 and 1, making it suitable for binary classification tasks like spam email detection or disease diagnosis. (Logistic regression, n.d.)

Logistic regression offers a balance between simplicity, interpretability, and performance, making it a suitable choice for evaluating the impact of feature scaling techniques in this study.

Evaluation Metrics:

To assess the effectiveness of each feature scaling technique, we employ logistic regression as our chosen classification algorithm. Logistic regression is well-suited for binary classification tasks and provides insights into the predictive performance of our models. Specifically, we utilize the `classification_report` function to generate a comprehensive summary of key evaluation metrics, including precision, recall, F1-score, and support, for each class in the dataset.

Feature Scaling Techniques:

We will evaluate the following feature scaling techniques:

1. Min-Max Scaling (Normalization)
2. Standardization (Z-score Scaling)
3. Robust Scaling
4. Mean Normalization
5. Unit Vector Scaling
6. Max Absolute Scaling

Methodology:

1. Dataset Exploration and Preparation: Before applying feature scaling techniques, the dataset will undergo preprocessing steps, including handling missing values, encoding categorical variables, and splitting the data into training and testing sets.
2. Feature Scaling: Each feature scaling technique will be applied to the training and testing sets, ensuring that features are standardized or normalized according to the chosen method.
3. Model Training: A machine learning model (In this study, we use logistic regression) will be trained on the scaled training data.
4. Model Evaluation: The trained model will be evaluated on the scaled testing data, and the classification report will be generated to assess the performance of each feature scaling technique.

Results and Analysis:

The classification report generated for each feature scaling technique will be analyzed to identify patterns and trends in model performance. Key metrics such as precision, recall, and F1-score will be compared across techniques to determine the effectiveness of each method in improving model performance.

Future Work:

Future study could explore additional feature scaling techniques and evaluate their performance on different datasets. Additionally, the impact of feature scaling on various machine learning algorithms and tasks could be investigated to further refine best practices in data preprocessing and model training.

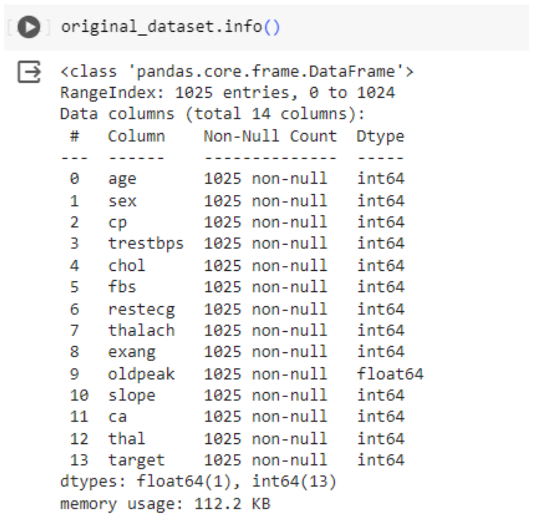
Practical Implementation

In this section, we will demonstrate the implementation of feature scaling techniques using Python and popular libraries such as NumPy, pandas, and scikit-learn. The objective is to provide a comprehensive guide with step-by-step instructions, code snippets, and visualizations to illustrate the effects of feature scaling on model performance.

Dataset Exploration and Preparation

Before applying feature scaling techniques, it's crucial to explore and understand the dataset. This involves loading the dataset, examining its structure, and identifying any preprocessing steps required. We will perform exploratory data analysis (EDA) to gain insights into the correlation, and characteristics of the features.

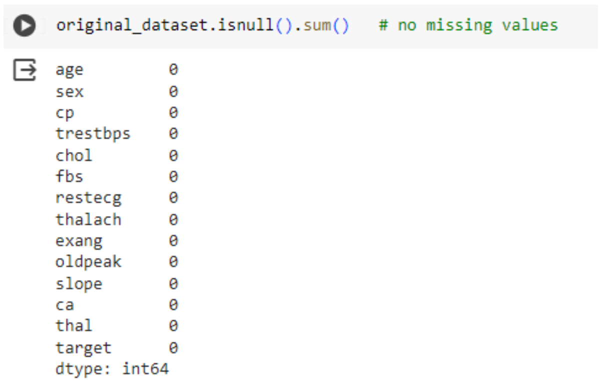
- Explore characteristics of the features.



```
original_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

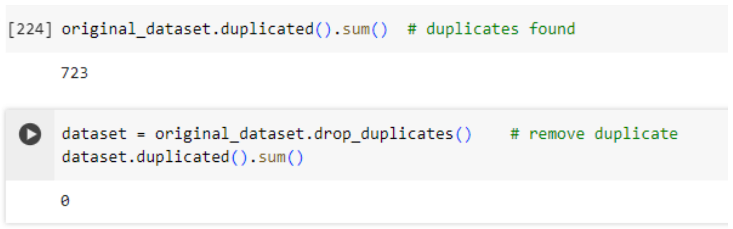
- Find the missing values.



```
original_dataset.isnull().sum()   # no missing values
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

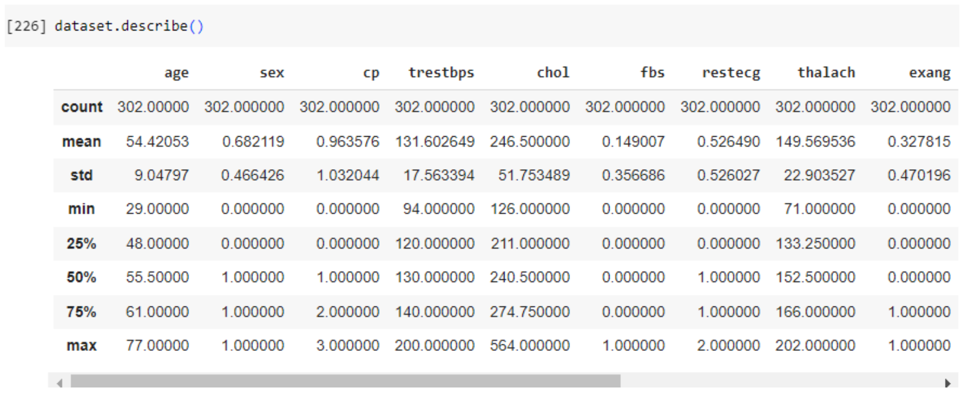
- Remove duplicated data.



```
[224] original_dataset.duplicated().sum()  # duplicates found

723
```

```
dataset = original_dataset.drop_duplicates()    # remove duplicate
dataset.duplicated().sum()

0
```

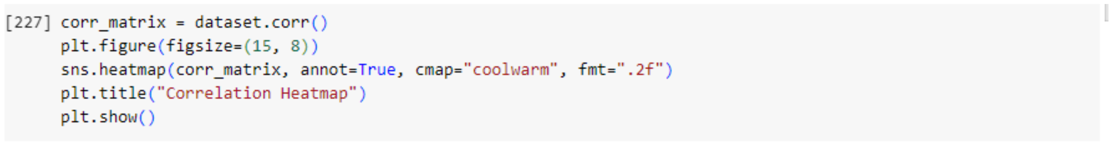
- Summary statistics



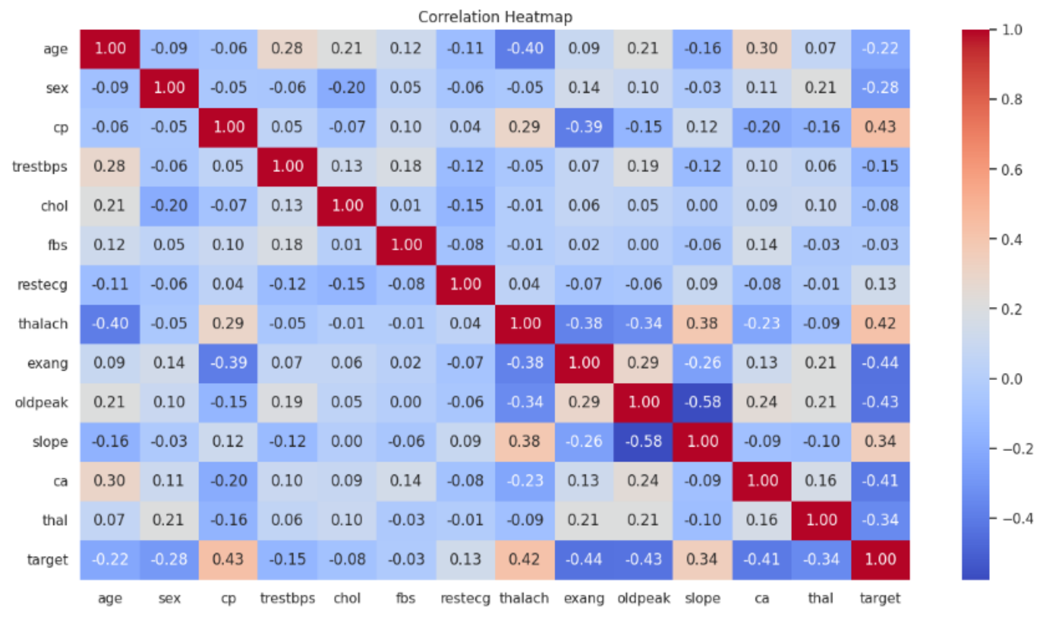
```
[226] dataset.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang
count	302.00000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000
mean	54.42053	0.682119	0.963576	131.602649	246.500000	0.149007	0.526490	149.569536	0.327815
std	9.04797	0.466426	1.032044	17.563394	51.753489	0.356686	0.526027	22.903527	0.470196
min	29.00000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000
25%	48.00000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.250000	0.000000
50%	55.50000	1.000000	1.000000	130.000000	240.500000	0.000000	1.000000	152.500000	0.000000
75%	61.00000	1.000000	2.000000	140.000000	274.750000	0.000000	1.000000	166.000000	1.000000
max	77.00000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000

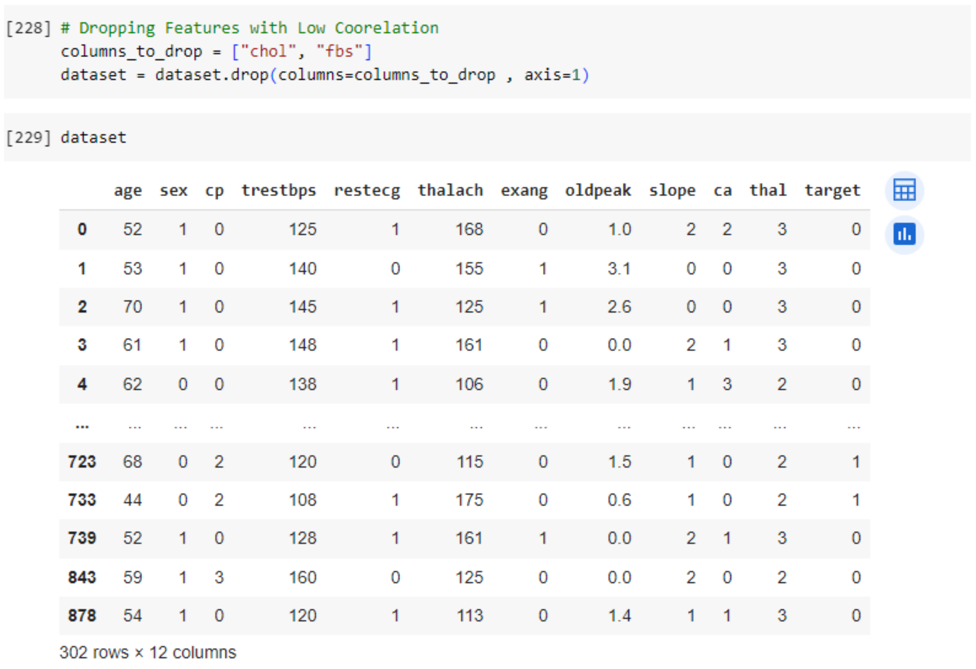
- Feature selection by removing the features that have low correlation with target.



```
[227] corr_matrix = dataset.corr()
      plt.figure(figsize=(15, 8))
      sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f")
      plt.title("Correlation Heatmap")
      plt.show()
```



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
age	1.00	-0.09	-0.06	0.28	0.21	0.12	-0.11	-0.40	0.09	0.21	-0.16	0.30	0.07	-0.22
sex	-0.09	1.00	-0.05	-0.06	-0.20	0.05	-0.06	-0.05	0.14	0.10	-0.03	0.11	0.21	-0.28
cp	-0.06	-0.05	1.00	0.05	-0.07	0.10	0.04	0.29	-0.39	-0.15	0.12	-0.20	-0.16	0.43
trestbps	0.28	-0.06	0.05	1.00	0.13	0.18	-0.12	-0.05	0.07	0.19	-0.12	0.10	0.06	-0.15
chol	0.21	-0.20	-0.07	0.13	1.00	0.01	-0.15	-0.01	0.06	0.05	0.00	0.09	0.10	-0.08
fbs	0.12	0.05	0.10	0.18	0.01	1.00	-0.08	-0.01	0.02	0.00	-0.06	0.14	-0.03	-0.03
restecg	-0.11	-0.06	0.04	-0.12	-0.15	-0.08	1.00	0.04	-0.07	-0.06	0.09	-0.08	-0.01	0.13
thalach	-0.40	-0.05	0.29	-0.05	-0.01	-0.01	0.04	1.00	-0.38	-0.34	0.38	-0.23	-0.09	0.42
exang	0.09	0.14	-0.39	0.07	0.06	0.02	-0.07	-0.38	1.00	0.29	-0.26	0.13	0.21	-0.44
oldpeak	0.21	0.10	-0.15	0.19	0.05	0.00	-0.06	-0.34	0.29	1.00	-0.58	0.24	0.21	-0.43
slope	-0.16	-0.03	0.12	-0.12	0.00	-0.06	0.09	0.38	-0.26	-0.58	1.00	-0.09	-0.10	0.34
ca	0.30	0.11	-0.20	0.10	0.09	0.14	-0.08	-0.23	0.13	0.24	-0.09	1.00	0.16	-0.41
thal	0.07	0.21	-0.16	0.06	0.10	-0.03	-0.01	-0.09	0.21	0.21	-0.10	0.16	1.00	-0.34
target	-0.22	-0.28	0.43	-0.15	-0.08	-0.03	0.13	0.42	-0.44	-0.43	0.34	-0.41	-0.34	1.00



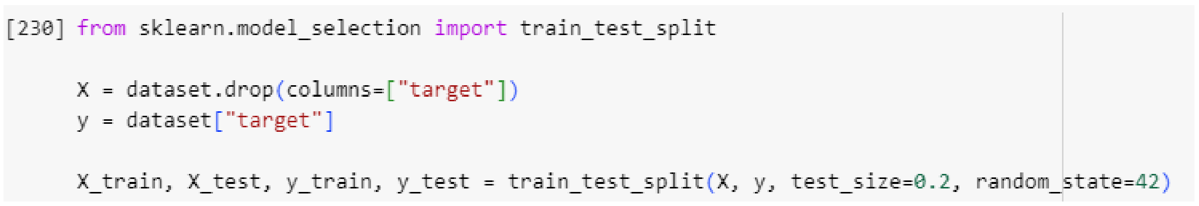
```
[228] # Dropping Features with Low Coorelation
      columns_to_drop = ["chol", "fbs"]
      dataset = dataset.drop(columns=columns_to_drop , axis=1)
```

```
[229] dataset
```

	age	sex	cp	trestbps	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	1	168	0	1.0	2	2	3	0
1	53	1	0	140	0	155	1	3.1	0	0	3	0
2	70	1	0	145	1	125	1	2.6	0	0	3	0
3	61	1	0	148	1	161	0	0.0	2	1	3	0
4	62	0	0	138	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...
723	68	0	2	120	0	115	0	1.5	1	0	2	1
733	44	0	2	108	1	175	0	0.6	1	0	2	1
739	52	1	0	128	1	161	1	0.0	2	1	3	0
843	59	1	3	160	0	125	0	0.0	2	0	2	0
878	54	1	0	120	1	113	0	1.4	1	1	3	0

302 rows × 12 columns

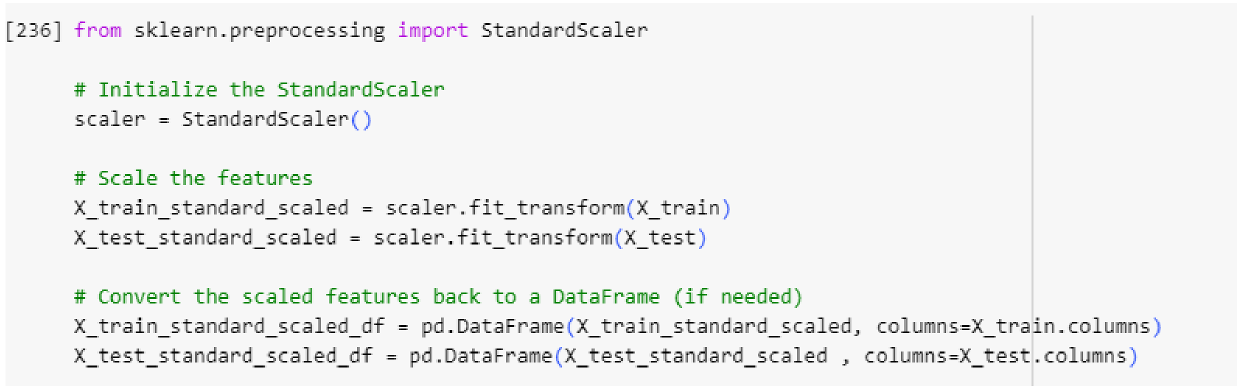
- Splitting the dataset into the Training set and Test set. 20% of the data will be used for testing, and the remaining 80% will be used for training.



```
[230] from sklearn.model_selection import train_test_split

      X = dataset.drop(columns=["target"])
      y = dataset["target"]

      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

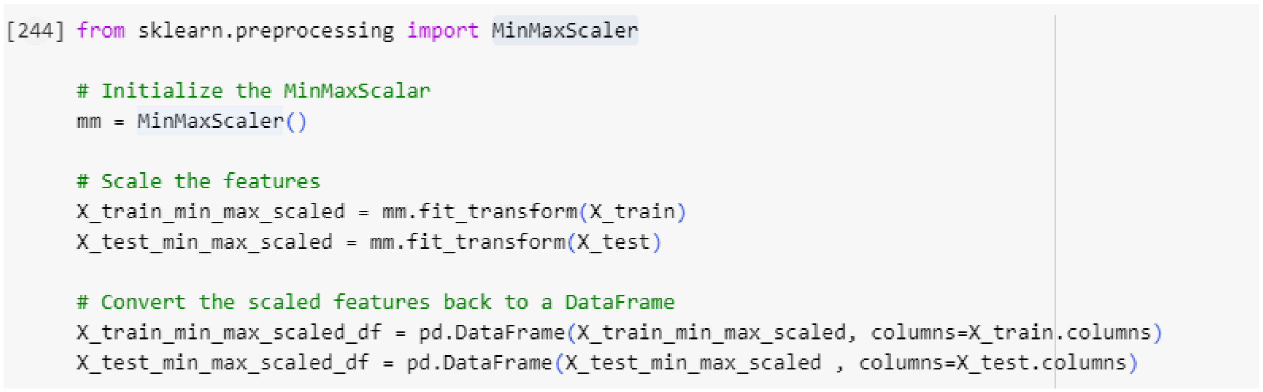
Feature Scaling Techniques

- Standardization: Use StandardScaler to scale X_train and X_test.



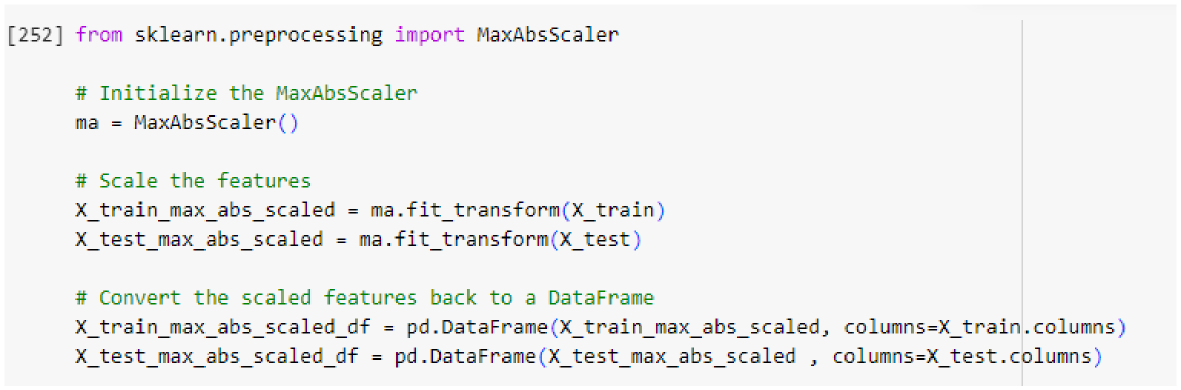
```
[236] from sklearn.preprocessing import StandardScaler

      # Initialize the StandardScaler
      scaler = StandardScaler()

      # Scale the features
      X_train_standard_scaled = scaler.fit_transform(X_train)
      X_test_standard_scaled = scaler.fit_transform(X_test)

      # Convert the scaled features back to a DataFrame (if needed)
      X_train_standard_scaled_df = pd.DataFrame(X_train_standard_scaled, columns=X_train.columns)
      X_test_standard_scaled_df = pd.DataFrame(X_test_standard_scaled , columns=X_test.columns)
```

- Min-Max Scaling: Use MinMaxScaler to scale X_train and X_test.



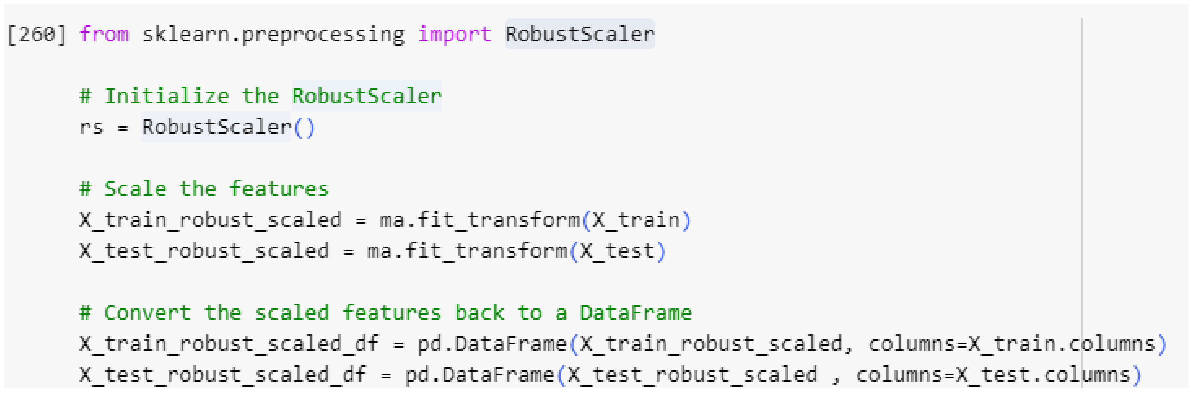
```
[244] from sklearn.preprocessing import MinMaxScaler

      # Initialize the MinMaxScalar
      mm = MinMaxScaler()

      # Scale the features
      X_train_min_max_scaled = mm.fit_transform(X_train)
      X_test_min_max_scaled = mm.fit_transform(X_test)

      # Convert the scaled features back to a DataFrame
      X_train_min_max_scaled_df = pd.DataFrame(X_train_min_max_scaled, columns=X_train.columns)
      X_test_min_max_scaled_df = pd.DataFrame(X_test_min_max_scaled , columns=X_test.columns)
```

- Max Absolute Scaling: Use MaxAbsScaler to scale X_train and X_test.



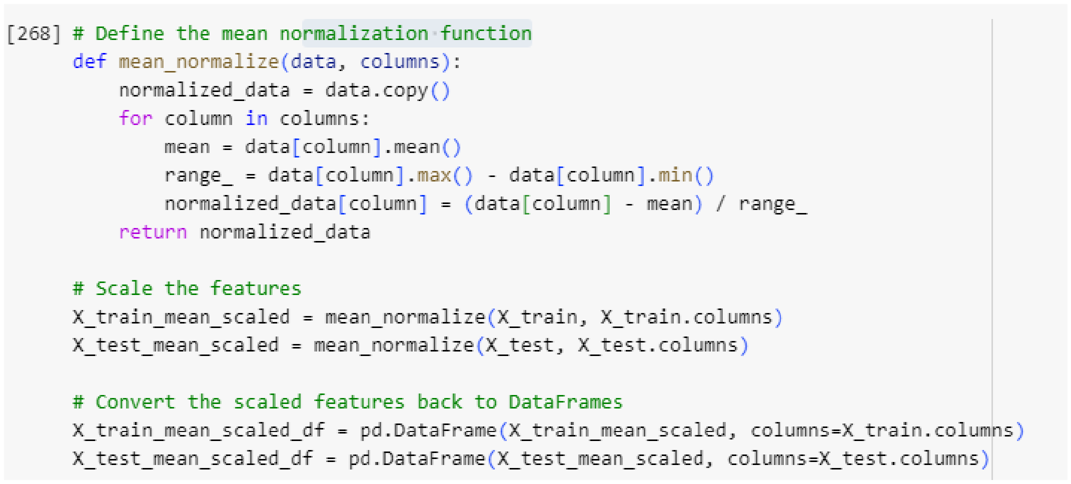
```
[252] from sklearn.preprocessing import MaxAbsScaler

      # Initialize the MaxAbsScaler
      ma = MaxAbsScaler()

      # Scale the features
      X_train_max_abs_scaled = ma.fit_transform(X_train)
      X_test_max_abs_scaled = ma.fit_transform(X_test)

      # Convert the scaled features back to a DataFrame
      X_train_max_abs_scaled_df = pd.DataFrame(X_train_max_abs_scaled, columns=X_train.columns)
      X_test_max_abs_scaled_df = pd.DataFrame(X_test_max_abs_scaled , columns=X_test.columns)
```

- Robust Scaling: Use RobustScaler to scale X_train and X_test.



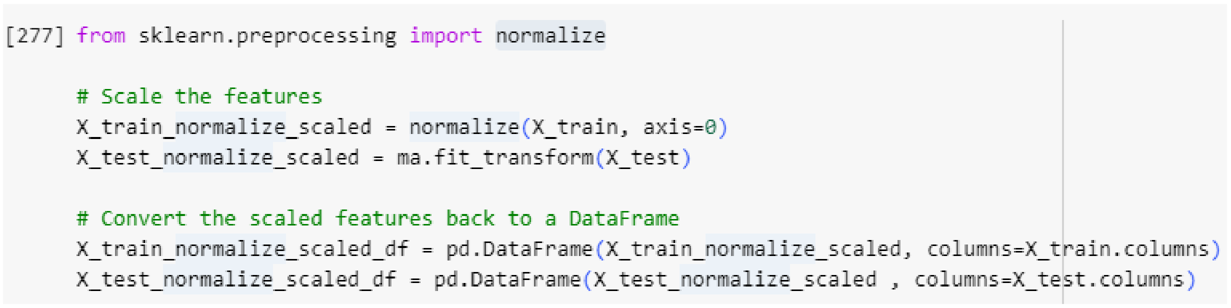
```
[260] from sklearn.preprocessing import RobustScaler

      # Initialize the RobustScaler
      rs = RobustScaler()

      # Scale the features
      X_train_robust_scaled = ma.fit_transform(X_train)
      X_test_robust_scaled = ma.fit_transform(X_test)

      # Convert the scaled features back to a DataFrame
      X_train_robust_scaled_df = pd.DataFrame(X_train_robust_scaled, columns=X_train.columns)
      X_test_robust_scaled_df = pd.DataFrame(X_test_robust_scaled , columns=X_test.columns)
```

- Mean Normalization: Because `sklearn.preprocessing` does not have Mean Normalization module, we have to create a new function to normalize data.



```
[268] # Define the mean normalization function
      def mean_normalize(data, columns):
          normalized_data = data.copy()
          for column in columns:
              mean = data[column].mean()
              range_ = data[column].max() - data[column].min()
              normalized_data[column] = (data[column] - mean) / range_
          return normalized_data

      # Scale the features
      X_train_mean_scaled = mean_normalize(X_train, X_train.columns)
      X_test_mean_scaled = mean_normalize(X_test, X_test.columns)

      # Convert the scaled features back to DataFrames
      X_train_mean_scaled_df = pd.DataFrame(X_train_mean_scaled, columns=X_train.columns)
      X_test_mean_scaled_df = pd.DataFrame(X_test_mean_scaled, columns=X_test.columns)
```

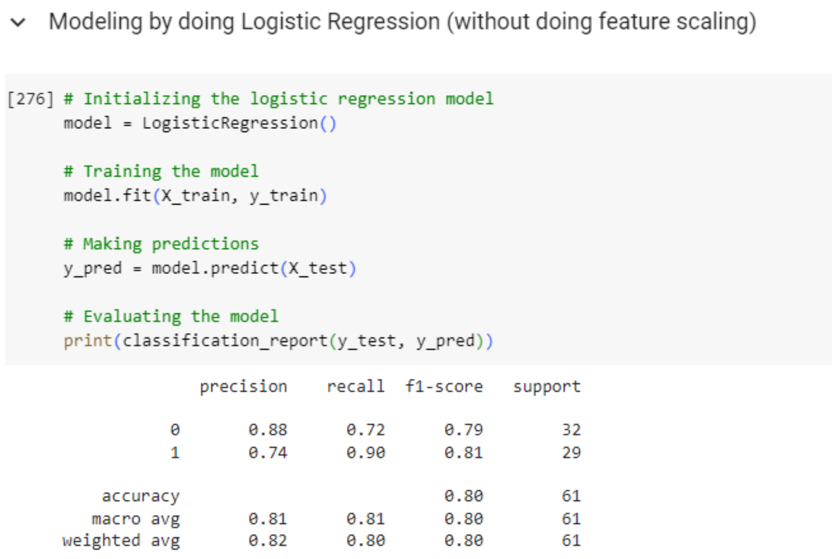
- Unit vector scaling: Use normalize to scale X_train and X_test.



```
[277] from sklearn.preprocessing import normalize

      # Scale the features
      X_train_normalize_scaled = normalize(X_train, axis=0)
      X_test_normalize_scaled = ma.fit_transform(X_test)

      # Convert the scaled features back to a DataFrame
      X_train_normalize_scaled_df = pd.DataFrame(X_train_normalize_scaled, columns=X_train.columns)
      X_test_normalize_scaled_df = pd.DataFrame(X_test_normalize_scaled , columns=X_test.columns)
```

Model Training and Evaluation

- Modeling without doing feature scaling.

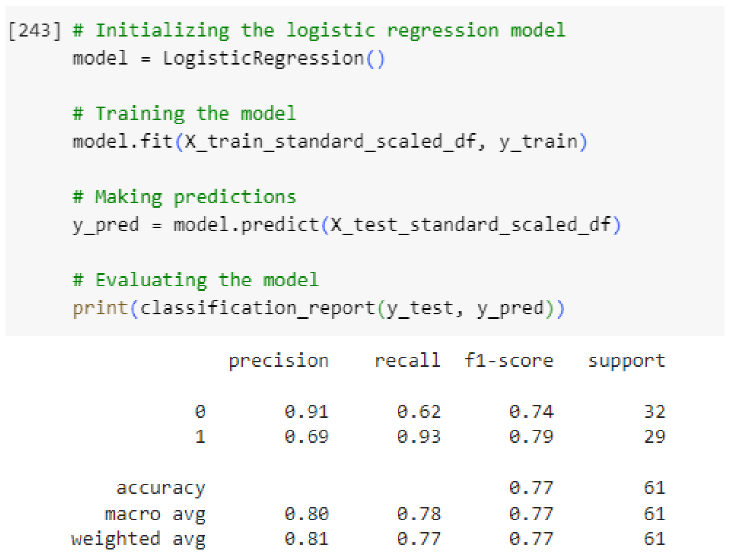


Modeling by doing Logistic Regression (without doing feature scaling)

```
[276] # Initializing the logistic regression model
      model = LogisticRegression()

      # Training the model
      model.fit(X_train, y_train)

      # Making predictions
      y_pred = model.predict(X_test)

      # Evaluating the model
      print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.72      0.79        32
           1       0.74      0.90      0.81        29

    accuracy                           0.80        61
   macro avg       0.81      0.81      0.80        61
weighted avg       0.82      0.80      0.80        61
```

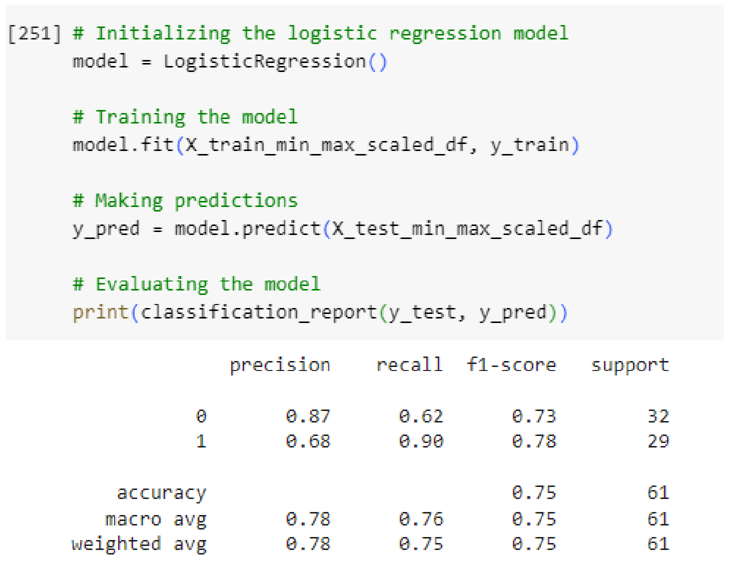
- Modeling with Standardization.



```
[243] # Initializing the logistic regression model
      model = LogisticRegression()

      # Training the model
      model.fit(X_train_standard_scaled_df, y_train)

      # Making predictions
      y_pred = model.predict(X_test_standard_scaled_df)

      # Evaluating the model
      print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.62      0.74        32
           1       0.69      0.93      0.79        29

    accuracy                           0.77        61
   macro avg       0.80      0.78      0.77        61
weighted avg       0.81      0.77      0.77        61
```

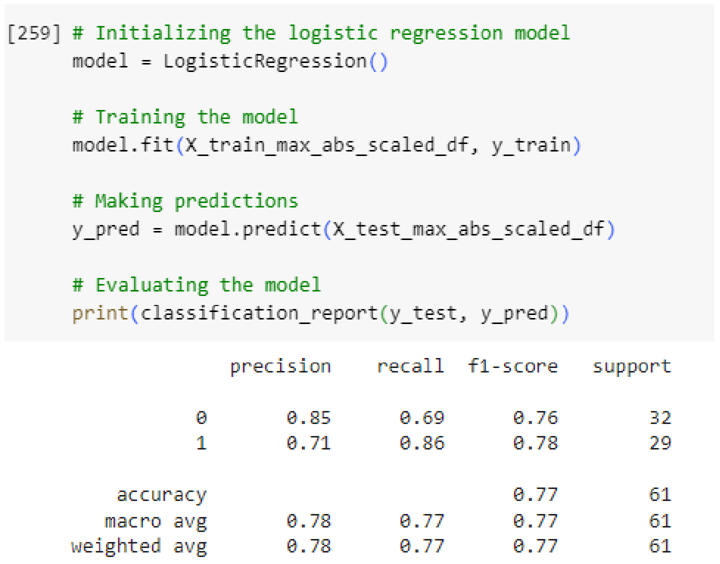
- Modeling with Min-Max Scaling



```
[251] # Initializing the logistic regression model
      model = LogisticRegression()

      # Training the model
      model.fit(X_train_min_max_scaled_df, y_train)

      # Making predictions
      y_pred = model.predict(X_test_min_max_scaled_df)

      # Evaluating the model
      print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.62      0.73        32
           1       0.68      0.90      0.78        29

    accuracy                           0.75        61
   macro avg       0.78      0.76      0.75        61
weighted avg       0.78      0.75      0.75        61
```

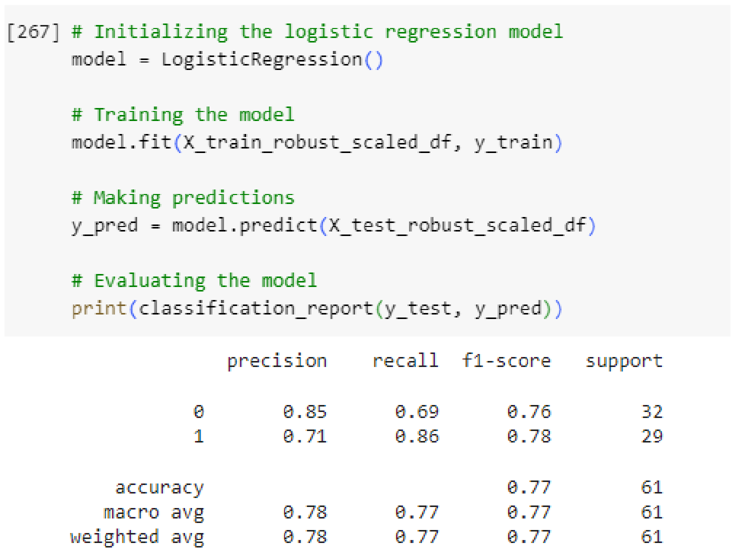
- Modeling with Max Absolute Scaling



```
[259] # Initializing the logistic regression model
      model = LogisticRegression()

      # Training the model
      model.fit(X_train_max_abs_scaled_df, y_train)

      # Making predictions
      y_pred = model.predict(X_test_max_abs_scaled_df)

      # Evaluating the model
      print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.69      0.76        32
           1       0.71      0.86      0.78        29

    accuracy                           0.77        61
   macro avg       0.78      0.77      0.77        61
weighted avg       0.78      0.77      0.77        61
```

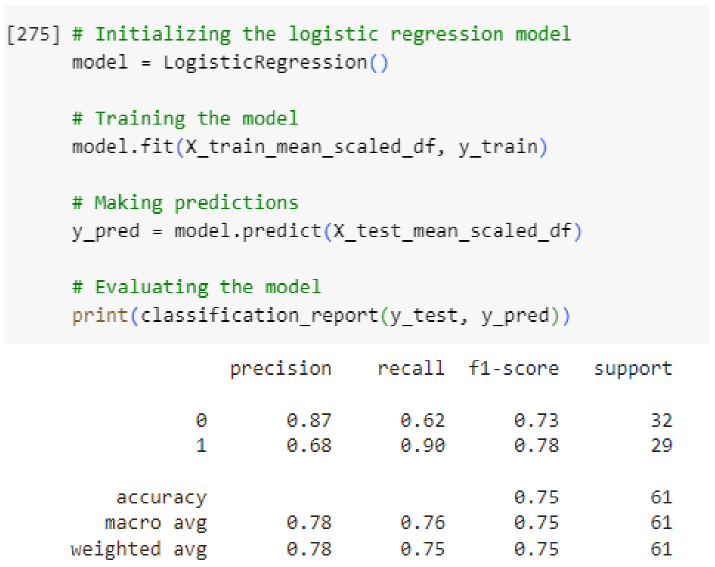
- Modeling with Robust Scaling



```
[267] # Initializing the logistic regression model
      model = LogisticRegression()

      # Training the model
      model.fit(X_train_robust_scaled_df, y_train)

      # Making predictions
      y_pred = model.predict(X_test_robust_scaled_df)

      # Evaluating the model
      print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.69      0.76        32
           1       0.71      0.86      0.78        29

    accuracy                           0.77        61
   macro avg       0.78      0.77      0.77        61
weighted avg       0.78      0.77      0.77        61
```

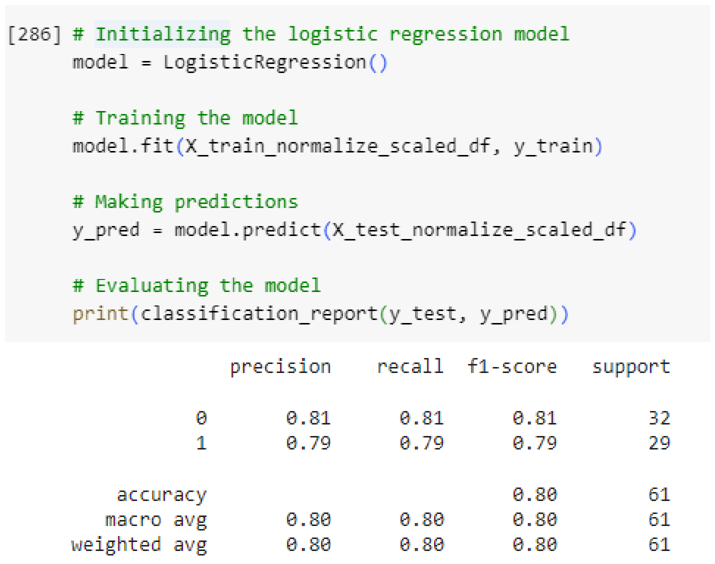
- Modeling with Mean Normalization



```
[275] # Initializing the logistic regression model
      model = LogisticRegression()

      # Training the model
      model.fit(X_train_mean_scaled_df, y_train)

      # Making predictions
      y_pred = model.predict(X_test_mean_scaled_df)

      # Evaluating the model
      print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.62      0.73        32
           1       0.68      0.90      0.78        29

    accuracy                           0.75        61
   macro avg       0.78      0.76      0.75        61
weighted avg       0.78      0.75      0.75        61
```

- Modeling with Unit vector scaling



```
[286] # Initializing the logistic regression model
      model = LogisticRegression()

      # Training the model
      model.fit(X_train_normalize_scaled_df, y_train)

      # Making predictions
      y_pred = model.predict(X_test_normalize_scaled_df)

      # Evaluating the model
      print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.81      0.81        32
           1       0.79      0.79      0.79        29

    accuracy                           0.80        61
   macro avg       0.80      0.80      0.80        61
weighted avg       0.80      0.80      0.80        61
```

Interpretation and Discussion

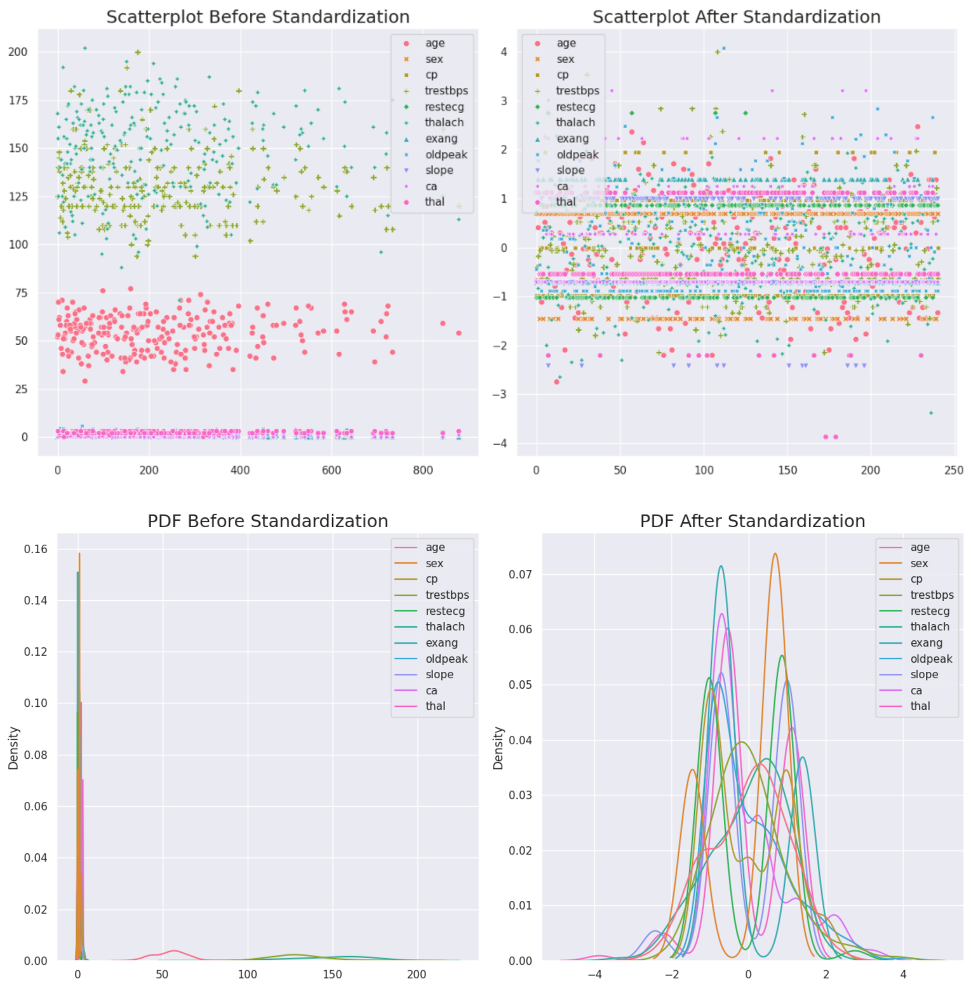
In this section, we interpret the results obtained from our experiments and discuss the implications of each feature scaling technique on model performance.

Comparative Analysis:

- Visualization for showing the distribution of features before and after scaling. (Since the distribution of training and test datasets are similar, these visualizations below are the comparison between X_train before and after scaling)

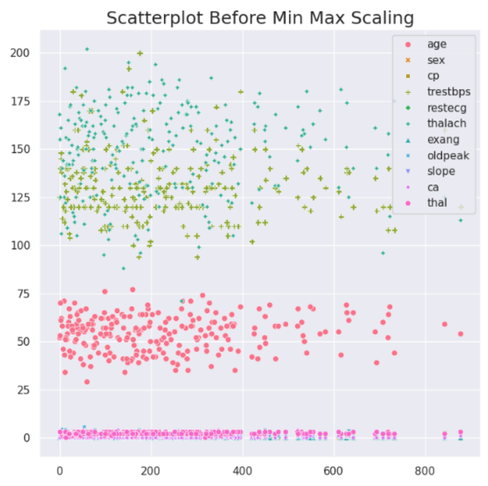
Before Scaling - The features appear to have a wide range of distributions. For instance, features like "age" and "thal" seem to have a relatively uniform distribution, while "trestbps" (resting blood pressure) might have a skewed distribution.

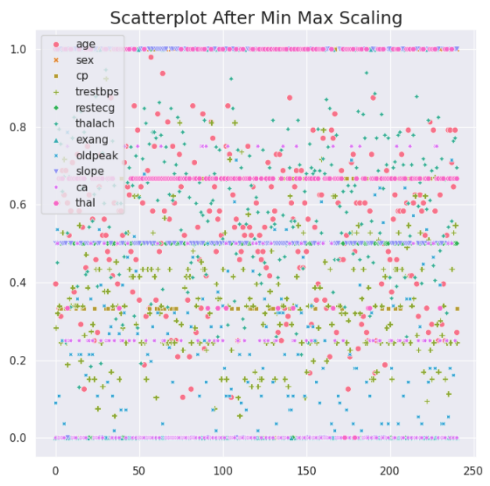
- Standardization

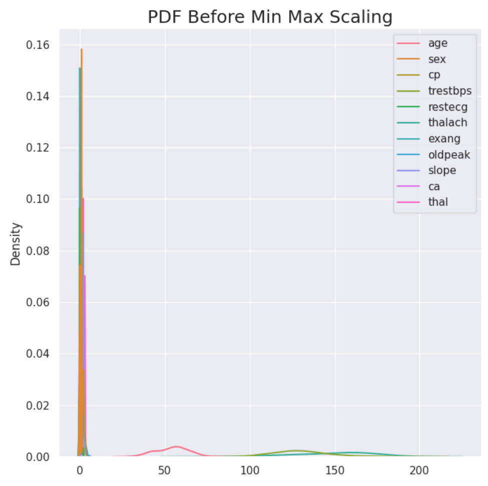


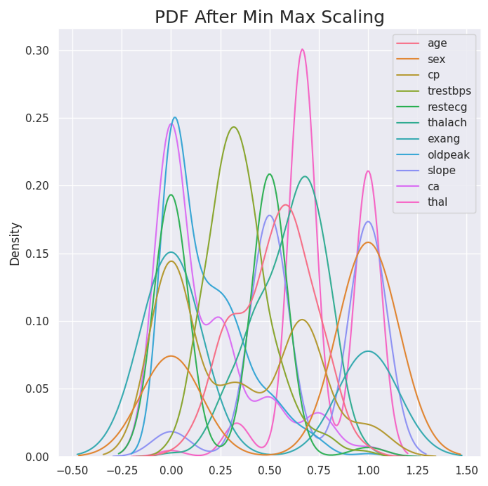
After standardization, the distributions of all features are compressed and centered around a zero mean with a standard deviation of one. This is because standardization transforms the data such that it has a mean of zero and a unit variance.

- Min-Max Scaling



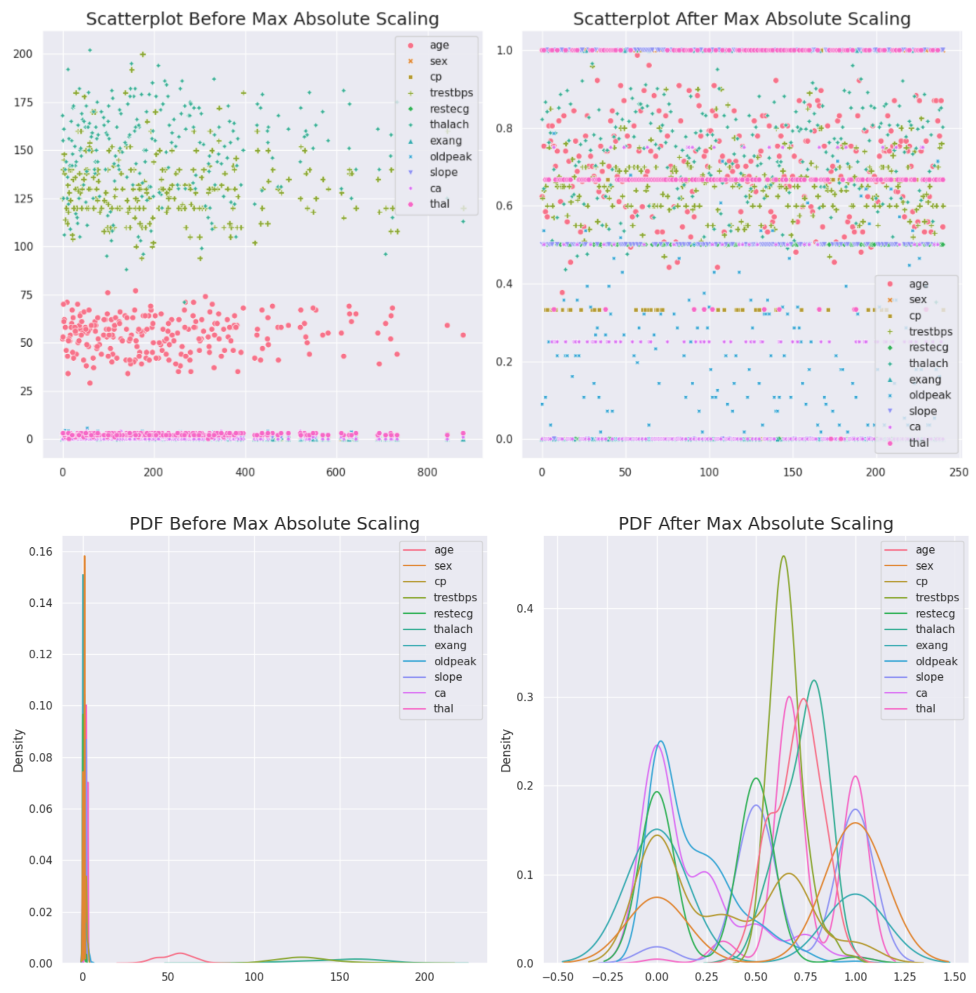






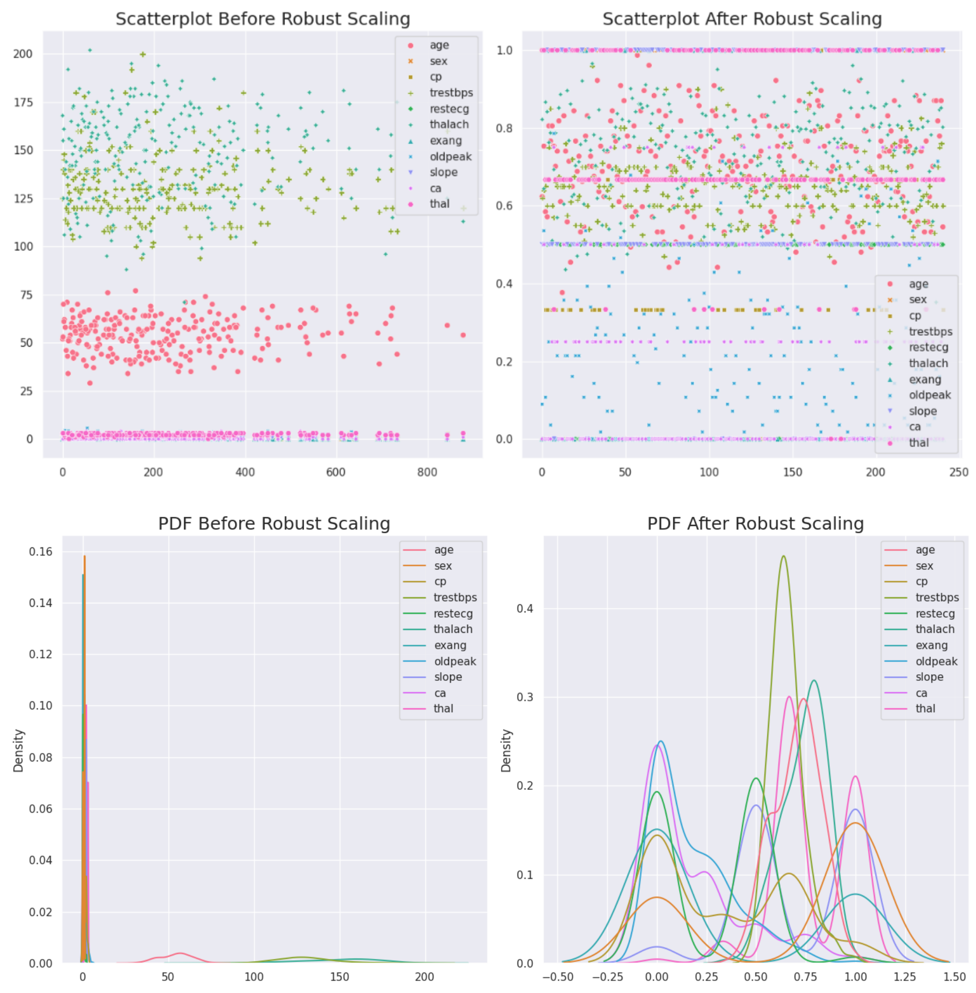
After Min-Max scaling, the distributions of all features are compressed and centered around a value of 0.5 on the x-axis. This is because Min-Max Scaling scales each feature to fit within a defined range (typically 0 to 1 in this case).

- Max Absolute Scaling



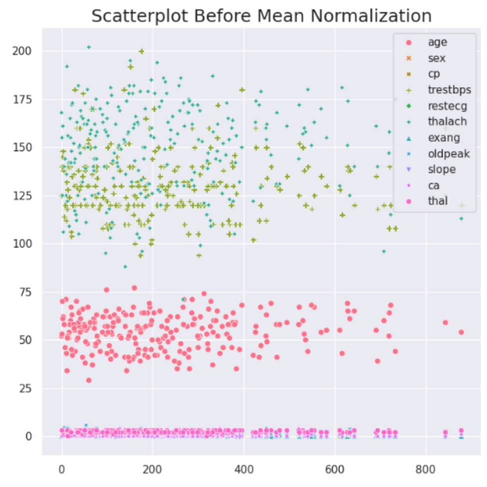
After Max Absolute scaling, the distributions of all features are likely compressed towards the center (around 0 on the x-axis) because max absolute scaling scales each feature based on its own absolute maximum value.

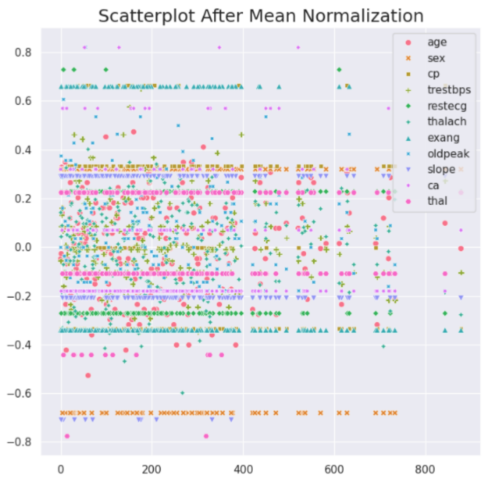
- Robust Scaling

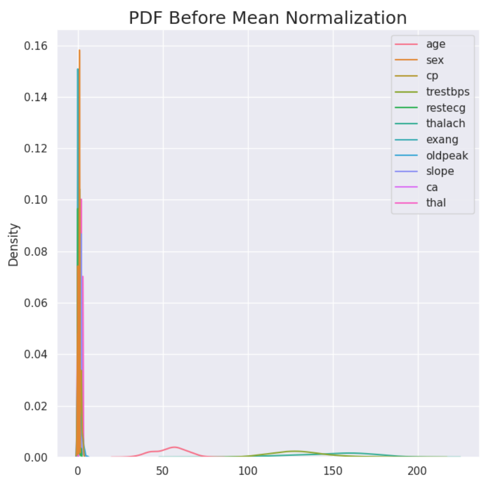


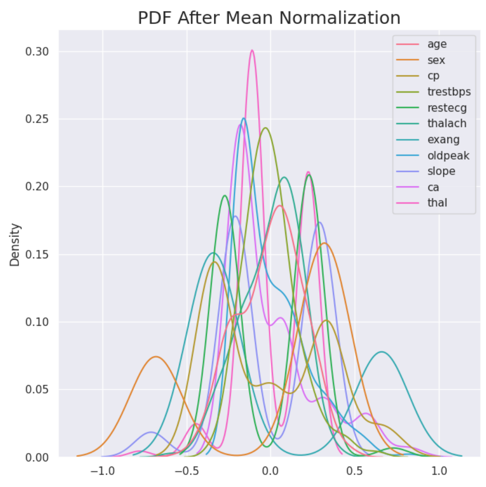
After Robust scaling, the distributions of all features are likely compressed towards the center (around 0 on the x-axis) because robust scaling is similar to standardization and centers the data around a median of 0 with a standard deviation of 1.

- Mean Normalization



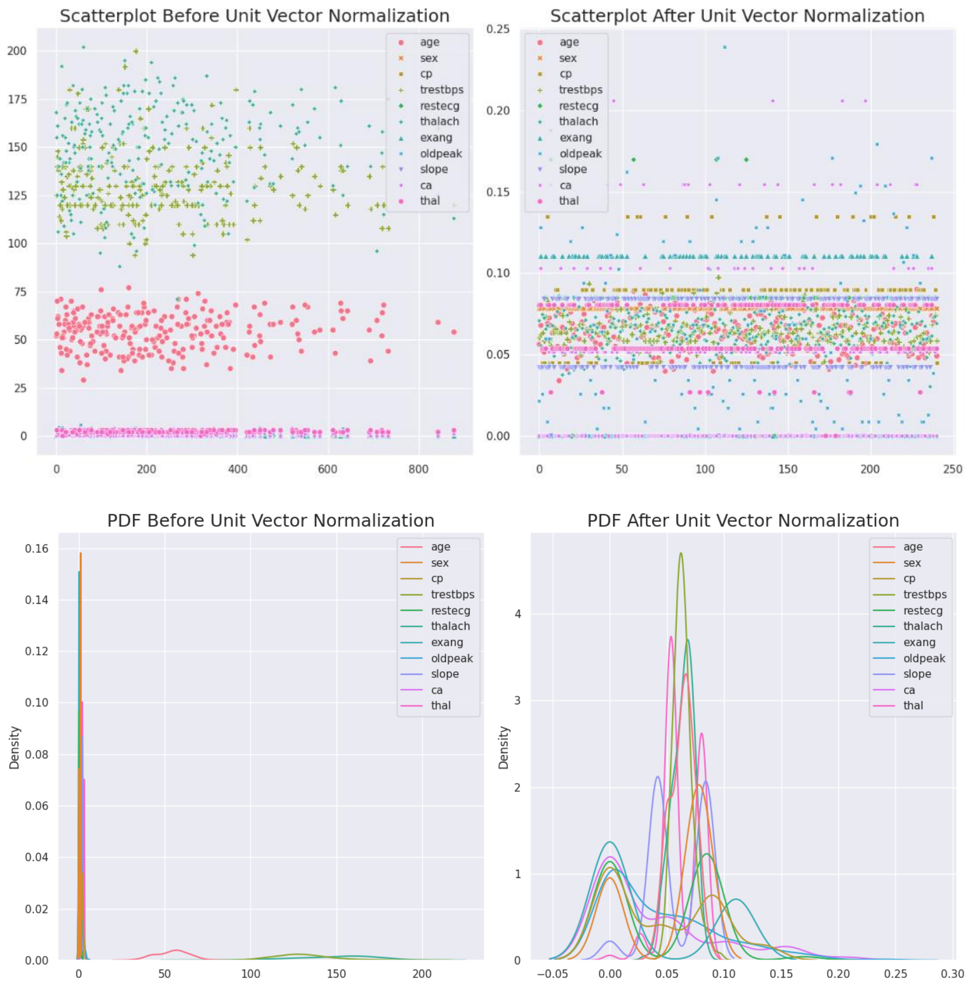






After Mean Normalization, the distributions of all features are compressed and centered around a value of 0 on the x-axis. This is because standardization transforms each feature to have a mean of zero and a unit variance.

- Unit Vector Normalization



After Unit Vector Normalization, the distribution of features is compressed into a tighter range around 0, with a peak at around 0.05 to 0.1. This is because normalization has scaled the data points to have a magnitude of 1.

- Classification Performance

Techniques	Accuracy	Precision	Recall	F1-Score
Without Feature Scaling	0.80	0.82	0.81	0.80
Standardization	0.77	0.81	0.78	0.77
Min-Max Scaling	0.75	0.78	0.76	0.75
Max Absolute Scaling	0.77	0.78	0.77	0.77
Robust Scaling	0.77	0.78	0.77	0.77
Mean Normalization	0.75	0.78	0.76	0.75
Unit Vector Normalization	0.80	0.80	0.80	0.80

Overall Observations:

- All scaling techniques resulted in a slight decrease in the overall accuracy compared to the non-scaled model (80%). The accuracy ranged from 75% (min-max scaling, mean normalization) to 77% (standardization, max absolute scaling, robust scaling).
- The F1-score also showed a minor decrease across all scaling methods compared to the non-scaled model (0.80). The F1-score ranged from 0.75 (min-max scaling, mean normalization) to 0.79 (standardization, max absolute scaling, robust scaling).
- The effect of scaling on precision and recall for each class varied. In some cases, scaling improved the metric for one class at the expense of the other. For example, standardization increased recall for class 1 (0.93) but decreased it for class 0 (0.62).

Discussion:

From the classification performance, Standardization, Min-Max Scaling, and Mean Normalization do not significantly improve model performance compared to the unscaled model. Max Absolute Scaling and Robust Scaling maintain performance similar to the unscaled model, making them suitable alternatives for preserving model accuracy. Unit Vector Normalization provides consistent performance with the unscaled model, indicating its effectiveness in maintaining model stability.

Analyze the advantages and disadvantages:

We analyze the advantages and disadvantages of each technique based on the classification report metrics, including precision, recall, and F1-score.

Techniques	Advantages	Disadvantages
Without Feature Scaling	The model achieved relatively high precision and recall for both classes without feature scaling, indicating that the original feature values were already conducive to model performance.	The model's performance could potentially be further improved with feature scaling, especially in scenarios where features have different scales or distributions.
Standardization	Standardization ensures that features have a mean of 0 and a standard deviation of 1, making it robust to outliers and suitable for algorithms assuming Gaussian distributions.	In this study, standardization led to a decrease in precision for class 0, suggesting that it may not be the optimal scaling technique for this specific dataset and model.
Min-Max Scaling	Min-Max Scaling bounds feature values between 0 and 1, preserving relationships between data points and making it	Although Min-Max Scaling improved recall for class 1, it led to a decrease in precision for both classes, indicating a trade-off between precision and recall.

	suitable for algorithms relying on distance measures.	
Max Absolute Scaling	Max Absolute Scaling preserves the sign of the data while scaling it, making it useful when the sign is significant.	Similar to Min-Max Scaling, Max Absolute Scaling resulted in a decrease in precision for both classes, indicating potential trade-offs between precision and recall.
Robust Scaling	Robust Scaling utilizes robust statistics such as the median and interquartile range, making it resilient to outliers and suitable for datasets with skewed distributions.	While Robust Scaling maintained precision and recall for class 1, it led to a decrease in precision for class 0, suggesting potential limitations in preserving class boundaries.
Mean Normalization	Mean Normalization centers data around zero, preserving relative distances between data points and ensuring equal contribution from each feature.	Similar to other scaling techniques, Mean Normalization resulted in a decrease in precision for both classes, indicating potential trade-offs between precision and recall.
Unit Vector Scaling	Unit Vector Normalization scales each feature vector to have a unit norm, ensuring equal contribution from each feature regardless of magnitude.	While Unit Vector Normalization maintained precision and recall for both classes, it did not significantly improve model performance compared to other techniques.

Conclusion:

The choice of feature scaling technique depends on the characteristics of the dataset and the requirements of the model. While some techniques may offer marginal improvements, others may provide stable performance without scaling. Further experimentation and validation are necessary to determine the most suitable technique for a given task.

For this dataset, Max Absolute Scaling, Robust Scaling, and Unit Vector Normalization are viable options for maintaining model performance.

Limitations:

- This analysis only considered a single dataset and may not generalize to all classification problems.
- Other factors beyond scaling, such as hyperparameter tuning or feature selection, could also influence the model's performance.

Conclusion

Feature scaling is a critical preprocessing step in machine learning that improves model convergence, performance, stability, and interpretability. Understanding and appropriately applying feature scaling techniques can significantly impact the success of machine learning models across various domains.

In summary, this report provides an in-depth analysis of feature scaling, its importance, common techniques, practical implementation, and recommendations for best practices in machine learning workflows.

Reference

DEY, A. (2021). *Complete guide to Feature Scaling*. Retrieved from Kaggle: https://www.kaggle.com/code/aimack/complete-guide-to-feature-scaling

Feature scaling. (2024, February 16). Retrieved from wikipedia: https://en.wikipedia.org/wiki/Feature_scaling

Gupta_OMG, M. (n.d.). *Feature Engineering: Scaling, Normalization, and Standardization*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/ml-feature-scaling-part-2/

LAPP, D. (2019). *Heart Disease Dataset*. Retrieved from Kaggle: https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset/data?select=heart.csv

Logistic regression. (n.d.). Retrieved from wikipedia: https://en.wikipedia.org/wiki/Logistic_regression

Preprocessing data. (n.d.). Retrieved from scikit-learn: https://scikit-learn.org/stable/modules/preprocessing.html

Vashisht, R. (2021, January 06). *When to perform a Feature Scaling?* Retrieved from Atoti: https://www.atoti.io/articles/when-to-perform-a-feature-scaling/

Yadav, P. (2023, November 9). *Difference between Standardization and Normalization*. Retrieved from medium: https://medium.com/@pawan329/difference-between-standardization-and-normalization-52c2b2313193