Importing Libraries

In [1]:

```
# Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler, RobustScaler, FunctionTransformer
from sklearn.metrics import accuracy_score
import numpy as np
```

Loading Data

In [60]:

```
#Loading the data
df = pd.read_csv(r"C:\Users\mahey\Downloads\wine-data.csv")
df.head()
```

Out[60]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

In [4]:

```
# Displaying dataset information
df.info()

# Checking for null values in the dataset
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6463 entries, 0 to 6462
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         6463 non-null   float64
 1   volatile acidity      6463 non-null   float64
 2   citric acid           6463 non-null   float64
 3   residual sugar        6463 non-null   float64
 4   chlorides             6463 non-null   float64
 5   free sulfur dioxide   6463 non-null   float64
 6   total sulfur dioxide  6463 non-null   float64
 7   density               6463 non-null   float64
 8   pH                    6463 non-null   float64
 9   sulphates             6463 non-null   float64
 10  alcohol               6463 non-null   float64
 11  quality               6463 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 606.0 KB
```

Out[4]:

```
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
dtype: int64
```

Initial Model Building and Evaluation

In [54]:

```
# Preparing the feature matrix (X) and target vector (y)
X = df.drop('quality', axis=1)  # Drop the target variable to create feature matrix
y = df['quality']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the initial Logistic Regression model
initial_model = LogisticRegression(max_iter=10000)
initial_model.fit(X_train, y_train)

# Make predictions and calculate accuracy
initial_predictions = initial_model.predict(X_test)
bottom_line_accuracy = accuracy_score(y_test, initial_predictions)

print(f"Bottom-line Accuracy: {bottom_line_accuracy}")
```

```
Bottom-line Accuracy: 0.5382830626450116
```

Feature Transformations and Model Evaluation

In [59]:

```
df_scaled = X.copy()
col_names = ['free sulfur dioxide']
features = df_scaled[col_names]
scaler = MinMaxScaler()
df_scaled[col_names] = scaler.fit_transform(features.values)
X_train, X_test, y_train, y_test = train_test_split(df_scaled, y, test_size=0.2, random_state=42)

# Build the initial Logistic Regression model
min_model = LogisticRegression(max_iter=10000)
min_model.fit(X_train, y_train)

# Make predictions and calculate accuracy
initial_predictions = min_model.predict(X_test)
bottom_line_accuracy = accuracy_score(y_test, initial_predictions)

print(f"min-max_model Accuracy: {bottom_line_accuracy}")
```

```
min-max_model Accuracy: 0.5320959010054138
```

```
C:\Users\mahey\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [18]:

```
df_scaled = X.copy()

# Scaling the 'alcohol' feature using StandardScaler
scaler = StandardScaler()
df_scaled['alcohol'] = scaler.fit_transform(df_scaled[['alcohol']])

# Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df_scaled, y, test_size=0.2, random_state=42)

# Building the Logistic Regression model with increased number of iterations
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Making predictions and calculating accuracy
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)

# Printing the accuracy
print(f"Standard Scaler Accuracy: {accuracy}")
```

```
Standard Scaler Accuracy: 0.5390564578499614
```

In [20]:

```
df_scaled = X.copy()

# Scaling the 'total sulfur dioxide' feature using MaxAbsScaler
scaler = MaxAbsScaler()
df_scaled['total sulfur dioxide'] = scaler.fit_transform(df_scaled[['total sulfur dioxide']])

# Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df_scaled, y, test_size=0.2, random_state=42)

# Building the Logistic Regression model with increased number of iterations
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Making predictions and calculating accuracy
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)

# Printing the accuracy
print(f"MaxAbsScaler Accuracy: {accuracy}")
```

```
MaxAbsScaler Accuracy: 0.5375096674400619
```

In [22]:

```
df_scaled = X.copy()

# Scaling the 'total sulfur dioxide' feature using RobustScaler
scaler = RobustScaler()
df_scaled['total sulfur dioxide'] = scaler.fit_transform(df_scaled[['total sulfur dioxide']])

# Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df_scaled, y, test_size=0.2, random_state=42)

# Building the Logistic Regression model with increased number of iterations
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Making predictions and calculating accuracy
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)

# Printing the accuracy with the correct label
print(f"RobustScaler Accuracy: {accuracy}")
```

```
RobustScaler Accuracy: 0.5367362722351121
```

In [39]:

```
# df_scaled = X.copy()
col_names = ['total sulfur dioxide']
features = df_scaled[col_names]
scaler = StandardScaler()
df_scaled[col_names] = scaler.fit_transform(features.values)
X_train, X_test, y_train, y_test = train_test_split(df_scaled, y, test_size=0.2, random_state=42)

# Build the initial Logistic Regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Make predictions and calculate accuracy
initial_predictions = model.predict(X_test)
bottom_line_accuracy = accuracy_score(y_test, initial_predictions)

print(f"Standard Scaler Accuracy: {bottom_line_accuracy}")
```

```
Standard Scaler Accuracy: 0.5328692962103635
```

Feature Selection and Model Evaluation

In [24]:

```
feature_selection_model=X.copy()
feature_selection_model = feature_selection_model.drop(feature_selection_model.columns[[0,1,2,3,4,5,6,7]], axis=1)
feature_selection_model.head()
X_train, X_test, y_train, y_test = train_test_split(feature_selection_model, y, test_size=0.2, random_state=42)

# Build the initial Logistic Regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Make predictions and calculate accuracy
initial_predictions = model.predict(X_test)
bottom_line_accuracy = accuracy_score(y_test, initial_predictions)

print(f"feature_selection_model_1 Accuracy: {bottom_line_accuracy}")
```

```
feature_selection_model_1 Accuracy: 0.5204949729311679
```

In [26]:

```
feature_selection_model=X.copy()
feature_selection_model = feature_selection_model.drop(feature_selection_model.columns[[3,4,5,6,7,8,9,10]], axis=1)
feature_selection_model.head()
X_train, X_test, y_train, y_test = train_test_split(feature_selection_model, y, test_size=0.2, random_state=42)

# Build the initial Logistic Regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Make predictions and calculate accuracy
initial_predictions = model.predict(X_test)
bottom_line_accuracy = accuracy_score(y_test, initial_predictions)

print(f"feature_selection_model_2 Accuracy: {bottom_line_accuracy}")
```

```
feature_selection_model_2 Accuracy: 0.44934261407579273
```

In [27]:

```
feature_selection_model=X.copy()
feature_selection_model = feature_selection_model.drop(feature_selection_model.columns[[0,3,4,5,6,7,9,10]], axis=1)
feature_selection_model.head()
X_train, X_test, y_train, y_test = train_test_split(feature_selection_model, y, test_size=0.2, random_state=42)

# Build the initial Logistic Regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Make predictions and calculate accuracy
initial_predictions = model.predict(X_test)
bottom_line_accuracy = accuracy_score(y_test, initial_predictions)

print(f"feature_selection_model_3 Accuracy: {bottom_line_accuracy}")
```

```
feature_selection_model_3 Accuracy: 0.4508894044856922
```

In [28]:

```
feature_selection_model=X.copy()
feature_selection_model = feature_selection_model.drop(feature_selection_model.columns[[0,1,2,4,5,7,8,9]], axis=1)
feature_selection_model.head()
X_train, X_test, y_train, y_test = train_test_split(feature_selection_model, y, test_size=0.2, random_state=42)

# Build the initial Logistic Regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Make predictions and calculate accuracy
initial_predictions = model.predict(X_test)
bottom_line_accuracy = accuracy_score(y_test, initial_predictions)

print(f"feature_selection_model_4 Accuracy: {bottom_line_accuracy}")
```

```
feature_selection_model_4 Accuracy: 0.5150812064965197
```

In [30]:

```
feature_selection_model=X.copy()
feature_selection_model = feature_selection_model.drop(feature_selection_model.columns[[0,1,2,4,5,6,7,9]], axis=1)
feature_selection_model.head()
X_train, X_test, y_train, y_test = train_test_split(feature_selection_model, y, test_size=0.2, random_state=42)

# Build the initial Logistic Regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Make predictions and calculate accuracy
initial_predictions = model.predict(X_test)
bottom_line_accuracy = accuracy_score(y_test, initial_predictions)

print(f"feature_selection_model_5 Accuracy: {bottom_line_accuracy}")
```

```
feature_selection_model_5 Accuracy: 0.5104408352668214
```

Conclusion

Overview of Model Performances: Bottom-line Model:

The bottom-line model, which uses the original dataset without any modifications, achieved an accuracy of 0.5383. This serves as our baseline for comparison.

Data Transformation Models:

The models using Standard Scaler, MaxAbsScaler, RobustScaler, and MinMaxScaler showed accuracies ranging from 0.5321 to 0.5391. Notably, the Standard Scaler applied to a different feature achieved the highest accuracy among these (0.5391), slightly surpassing the bottom-line model. This suggests that standardization may offer a marginal benefit over the original data scaling for this specific dataset. The other scaling techniques (MaxAbsScaler, RobustScaler, and MinMaxScaler) resulted in accuracies marginally lower than the bottom-line model, indicating a negligible or no substantial improvement over the untransformed data.

Feature Selection Models:

The feature selection models showed a decrease in accuracy compared to the bottom-line model, with accuracies ranging from 0.4493 to 0.5205. The most significant drop in accuracy was observed in feature_selection_model_2 (0.4493), suggesting that the features removed in this model were likely critical for predicting wine quality. The highest accuracy in feature selection models was feature_selection_model_1 (0.5205), which, while lower than the bottom-line accuracy, implies that certain features might have a more significant impact on the prediction outcome than others.

Conclusions and Insights:

Data Transformation Impact: The minimal impact of scaling techniques on model accuracy indicates that the original feature scales in this dataset are already quite suitable for logistic regression modeling. The slight improvement with Standard Scaler suggests that some features might benefit from normalization, but the overall benefit is limited.

Feature Selection Impact: The general decrease in accuracy with feature selection models emphasizes the importance of the features in predicting wine quality. Removing even a small number of features can lead to significant information loss, negatively affecting the model's performance.

Optimal Data Preparation Strategy: For this dataset, using all features with minimal transformation seems to be the most effective strategy. While certain scaling methods offer a marginal increase in accuracy, the overall benefit compared to the computational cost and complexity is limited