LAB 2
Subham Beura
CE 7th Sem
B521060

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
```

```
lab2 = pd.read_csv("lab1_dataset.csv")
lab2.head(
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

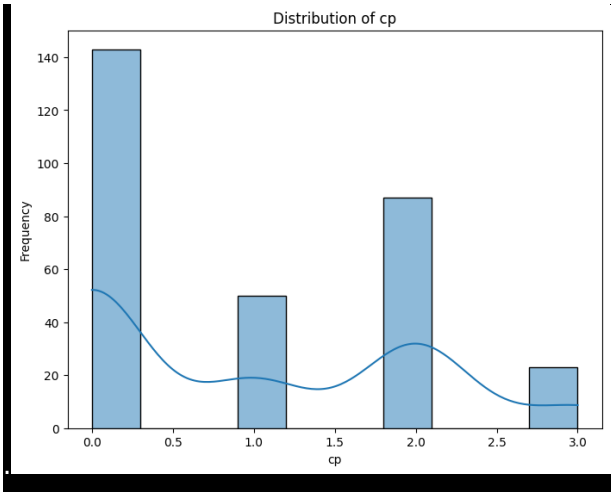
```
print(lab2.isnull().sum()
```

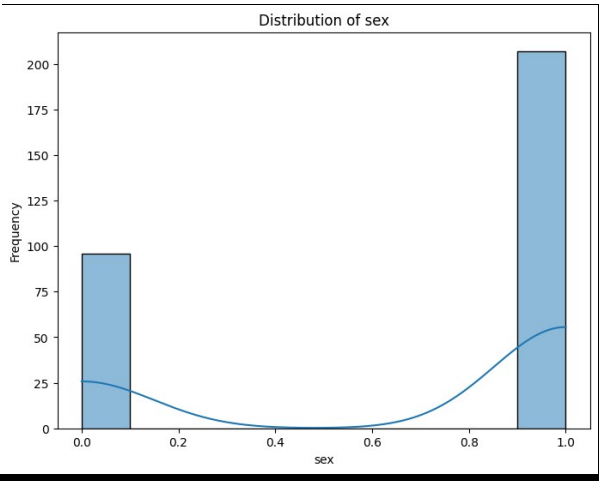
```
age       0
sex       0
cp        0
trestbps  0
chol      0
fbs       0
restecg   0
thalach   0
exang     0
oldpeak   0
slope     0
ca        0
thal      0
target    0
dtype: int64
```

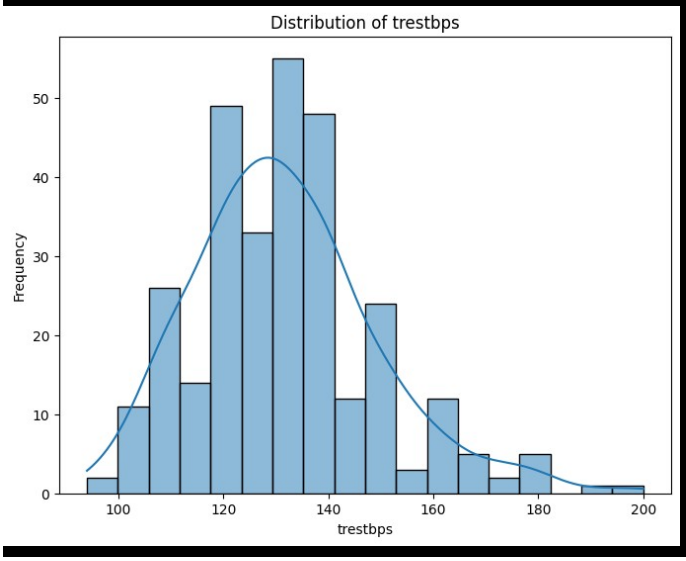
The dataset does not contain any null values across all columns, so no handling of missing values is necessary

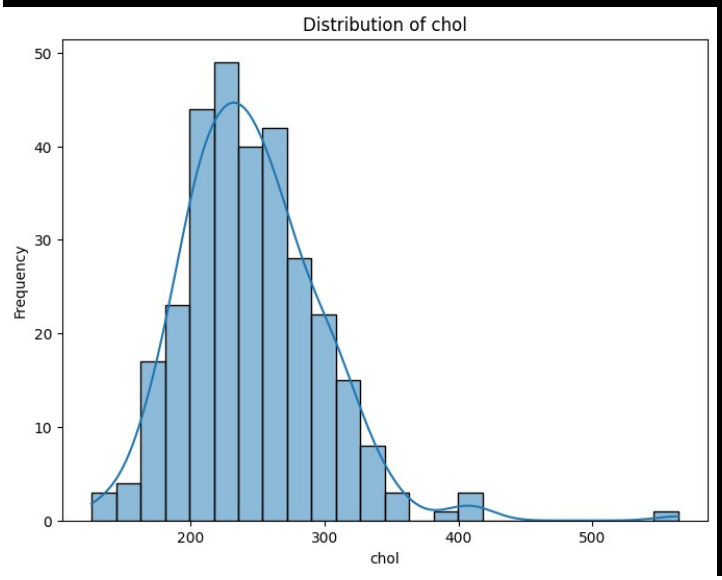
```
from scipy.stats import shapiro
```

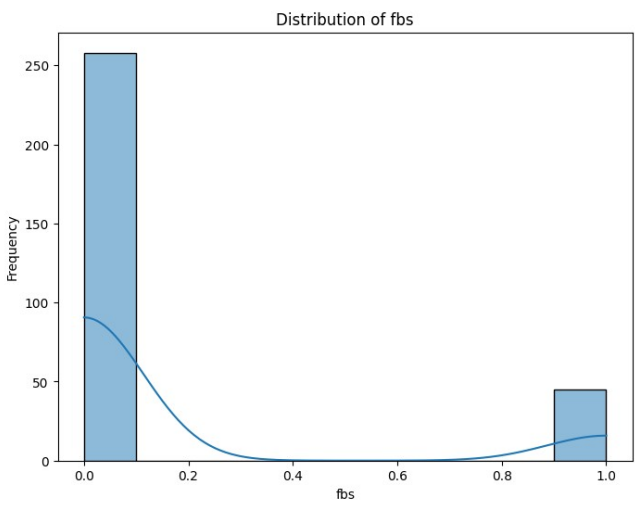
```
def check_normal_distribution(data,columns):
 results = {}
 for column in columns:
   plt.figure(figsize=(8, 6))
   sns.histplot(data[column], kde=True)
   plt.title(f'Distribution of {column}')
   plt.xlabel(column)
   plt.ylabel('Frequency')
   plt.show()
   stat,p_value = shapiro(data[column])
   results[column] = {'stat':stat,'p_value':p_value}
   if p_value > 0.05:
     results[column]['result'] = 'normal'
   else:
     results[column]['result'] = 'not normal'
   print(f"\n{column}: stat={stat}, p_value={p_value},
result={results[column]['result']}\n")
 return results
numeric_columns =
lab2.select_dtypes(include=['int64','float64']).columns
normality_results = check_normal_distribution(lab2,numeric_columns)
```

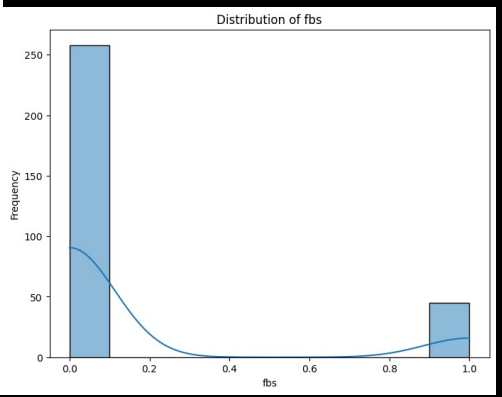

Distribution of cp
Frequency
cp
0
20
40
60
80
100
120
140
0.0
0.5
1.0
1.5
2.0
2.5
3.0

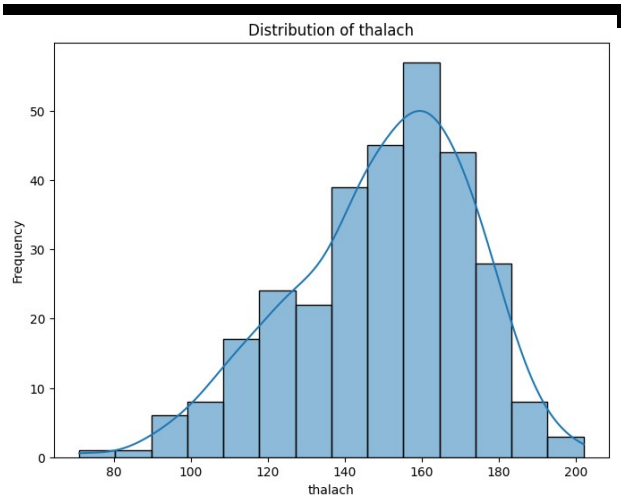

Distribution of sex
Frequency
sex
0
25
50
75
100
125
150
175
200
0.0
0.2
0.4
0.6
0.8
1.0

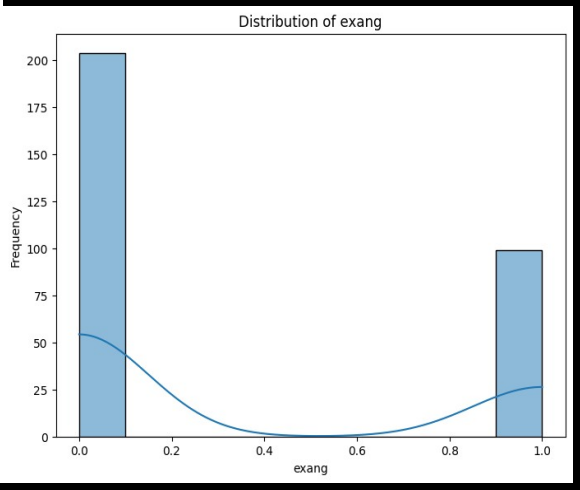

Distribution of trestbps
Frequency
trestbps
0
10
20
30
40
50
100
120
140
160
180
200

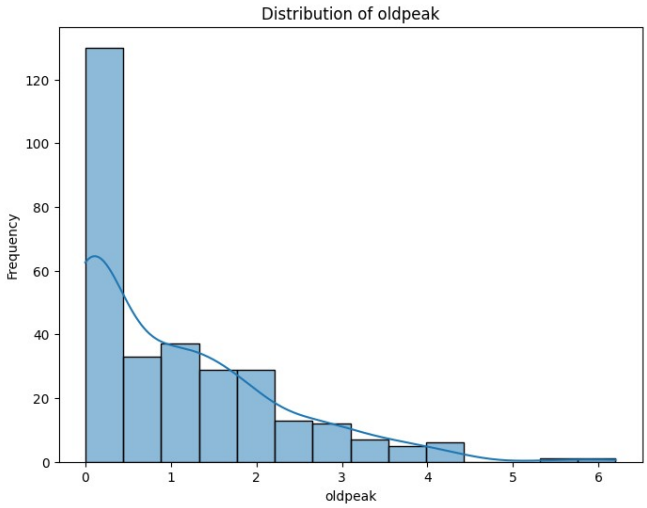

Distribution of chol
Frequency
chol
0
10
20
30
40
50
200
300
400
500

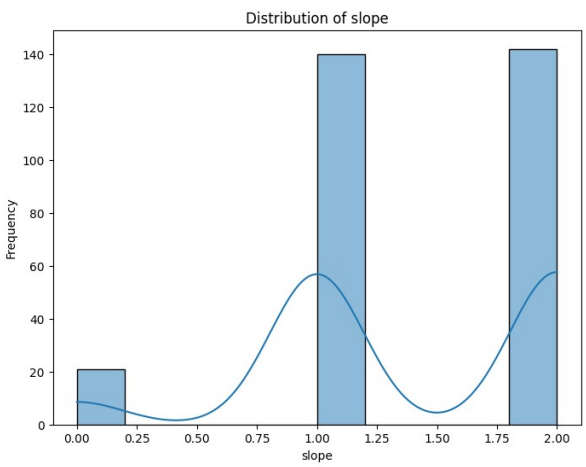

Distribution of fbs
Frequency
fbs
0
50
100
150
200
250
0.0
0.2
0.4
0.6
0.8
1.0

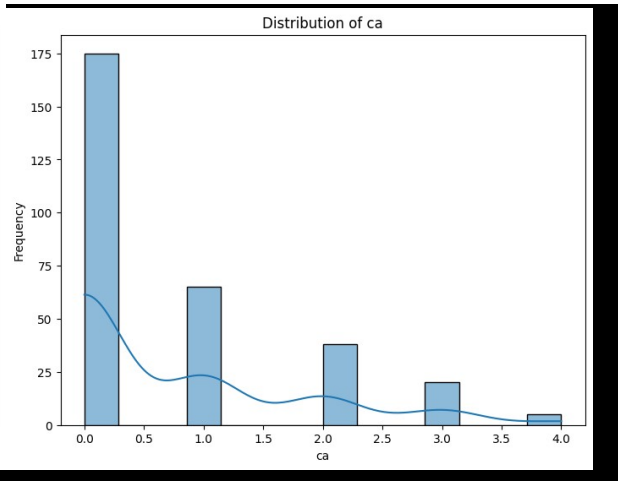

Distribution of fbs
Frequency
fbs
0
50
100
150
200
250
0.0
0.2
0.4
0.6
0.8
1.0

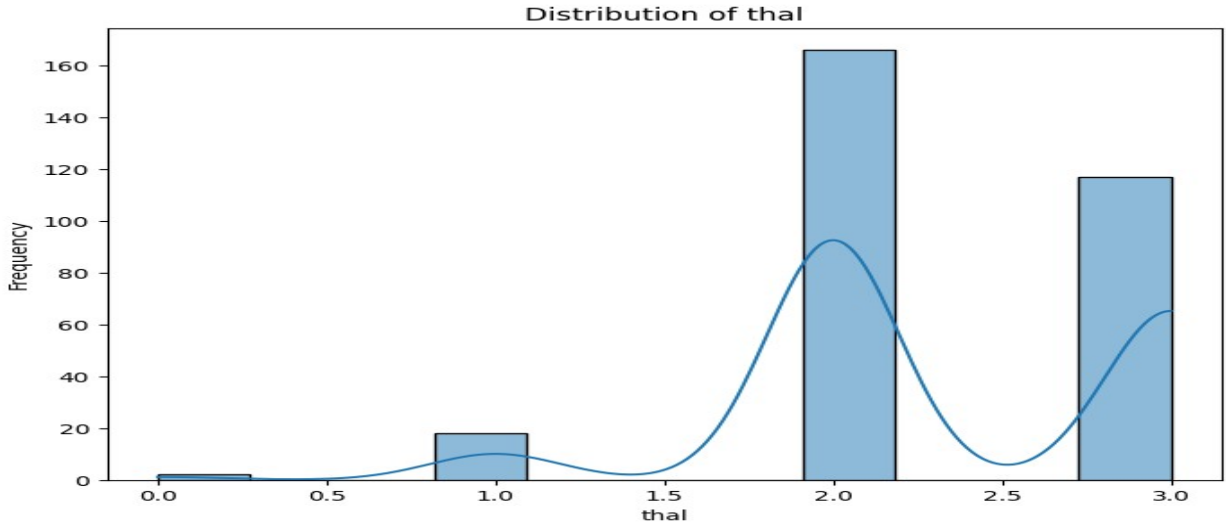

Distribution of thalach
Frequency
thalach
0
10
20
30
40
50
80
100
120
140
160
180
200

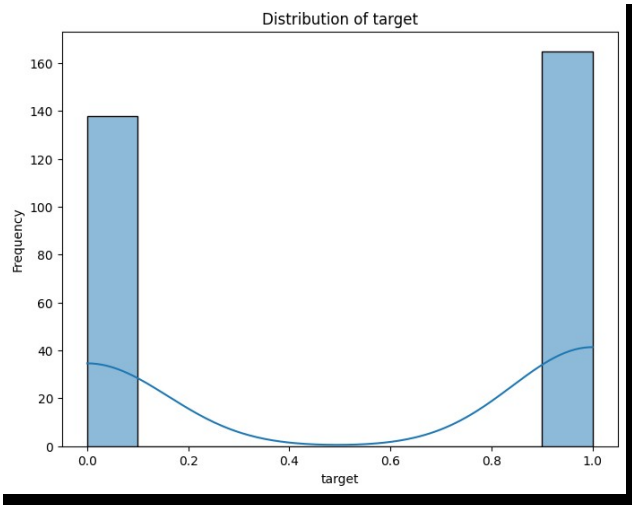

Distribution of exang
Frequency
exang
0
25
50
75
100
125
150
175
200
0.0
0.2
0.4
0.6
0.8
1.0

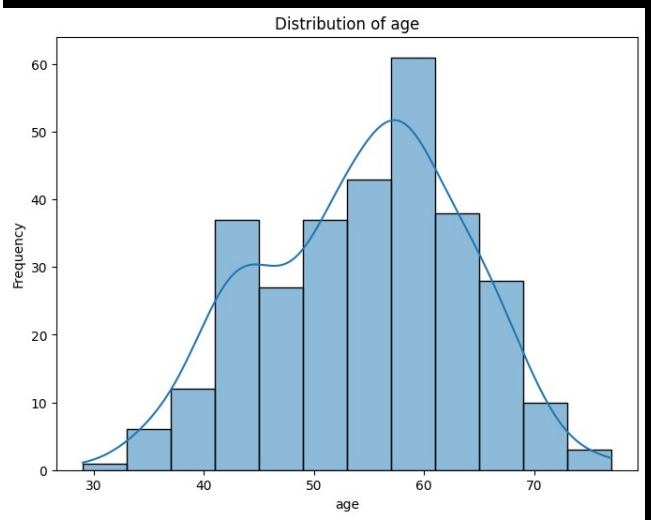

Distribution of oldpeak
Frequency
oldpeak
0
20
40
60
80
100
120
0
1
2
3
4
5
6


Distribution of slope
Frequency
slope
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
20
40
60
80
100
120
140


Distribution of ca
Frequency
ca
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
25
50
75
100
125
150
175


Distribution of thal
Frequency
thal
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100
120
140
160





Shapiro-Wilk Test Results for Normality

Feature	Shapiro-Wilk statistic	p-value
age	0.986	0.0058
sex	0.586	≈ 0
cp	0.790	≈ 0
trestbps	0.966	≈ 0
chol	0.947	≈ 0
fbs	0.424	≈ 0
restecg	0.679	≈ 0
thalac	0.976	≈ 0

h

exang	0.591	≈ 0
oldpeak	0.844	≈ 0
slope	0.745	≈ 0
ca	0.728	≈ 0
thal	0.751	≈ 0
target	0.634	≈ 0

Interpretation of p-values:

1. p-value < 0.05: Indicates that the null hypothesis of normality is rejected, meaning the feature does not follow a normal distribution.
2. p-value ≥ 0.05: Fails to reject the null hypothesis, suggesting the feature might follow a normal distribution.

Conclusion: Based on the p-values, all features have p-values less than 0.05, indicating that none of the features follow a normal distribution. The histogram plots further confirm this, showing various distributions that are not normally distributed.

	age	sex	cp	trestbps	chol	fbs	restecg	\
0	0.952197	0.681005	1.973123	0.763956	-0.256334	2.394438	-1.005832	
1	-1.915313	0.681005	1.002577	-0.092738	0.072199	-0.417635	0.898962	
2	-1.474158	-1.468418	0.032031	-0.092738	-0.816773	-0.417635	-1.005832	
3	0.180175	0.681005	0.032031	-0.663867	-0.198357	-0.417635	0.898962	
4	0.290464	-1.468418	-0.938515	-0.663867	2.082050	-0.417635	0.898962	

	thalach	exang	oldpeak	slope	ca	thal	target
0	0.015443	-0.696631	1.087338	-2.274579	-0.714429	-2.148873	0.914529
1	1.633471	-0.696631	2.122573	-2.274579	-0.714429	-0.512922	0.914529
2	0.977514	-0.696631	0.310912	0.976352	-0.714429	-0.512922	0.914529
3	1.239897	-0.696631	-0.206705	0.976352	-0.714429	-0.512922	0.914529
4	0.583939	1.435481	-0.379244	0.976352	-0.714429	-0.512922	0.914529

Understanding Data Standardization

1. Importance of Standardization

- **Equal Contribution:** Standardization ensures that all features contribute equally to the analysis, regardless of their original scale.

- **Algorithm Efficiency:** Many machine learning algorithms, such as those using gradient descent (e.g., linear regression, logistic regression), benefit from standardized data as it speeds up convergence.
- **Comparability:** Features with different scales can distort model predictions. Standardization prevents this by putting all features on a common scale.

2. Interpreting Standardized Values

- **Positive Values:** A positive standardized value indicates that the original feature value is above the mean of that feature. The greater the positive value, the further away from the mean it is.
- **Negative Values:** A negative standardized value indicates that the original feature value is below the mean of that feature. The more negative the value, the further below the mean it is.
- **Values Close to Zero:** Values close to zero mean that the original feature value is close to the mean.

Key Takeaway: Each value has been transformed such that the features now have a mean of 0 and a standard deviation of 1. This transformation is useful for algorithms sensitive to the scale of the input data, ensuring that all features contribute equally.

```
from sklearn.preprocessing import MinMaxScaler
numeric_columns = lab2.select_dtypes(include=['int64',
'float64']).columns
scaler = MinMaxScaler()
lab2[numeric_columns] = scaler.fit_transform(lab2[numeric_columns])
print(lab2.head())
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	\
0	0.708333	1.0	1.000000	0.481132	0.244292	1.0	0.0	0.603053	0.0	
1	0.166667	1.0	0.666667	0.339623	0.283105	0.0	0.5	0.885496	0.0	
2	0.250000	0.0	0.333333	0.339623	0.178082	0.0	0.0	0.770992	0.0	
3	0.562500	1.0	0.333333	0.245283	0.251142	0.0	0.5	0.816794	0.0	
4	0.583333	0.0	0.000000	0.245283	0.520548	0.0	0.5	0.702290	1.0	

	oldpeak	slope	ca	thal	target
0	0.370968	0.0	0.0	0.333333	1.0
1	0.564516	0.0	0.0	0.666667	1.0
2	0.225806	1.0	0.0	0.666667	1.0
3	0.129032	1.0	0.0	0.666667	1.0
4	0.096774	1.0	0.0	0.666667	1.0

Understanding Data Normalization

1. Use of Normalization

- **Non-Gaussian Distribution:**
  - Normalization is useful when the data does not follow a Gaussian (normal) distribution.
  - It's beneficial when you need all features to be on the same scale but do not want to make assumptions about the distribution.
- **Distance-Based Algorithms:**
  - Normalization is particularly beneficial for algorithms that compute distances between data points, such as:
    - K-Nearest Neighbors (KNN)
    - Neural networks
  - It ensures that no feature dominates due to its scale.

Key Benefit: By normalizing, all numeric features in the dataset will be scaled to the range [0, 1]. This makes them suitable for various machine learning algorithms that require data to be on the same scale..

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
X = lab2.drop('target', axis=1)
y = lab2['target']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42, stratify=y)
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)
print("Original training set class distribution:")
print(y_train.value_counts())
print("\nBalanced training set class distribution:")
print(pd.Series(y_train_balanced).value_counts()
```

Original training set class distribution:
target
1.0 115
0.0 97
Name: count, dtype: int64

Balanced training set class distribution:
target
1.0 115
0.0 115
Name: count, dtype: int64

Class Distribution Analysis and Dataset Balancing

Original Class Distribution

- Class 1.0: 115 instances
- Class 0.0: 97 instances

Balanced Class Distribution

- Class 1.0: 115 instances
- Class 0.0: 115 instances

Impact of Balancing

Balancing the dataset typically improves the performance of classification models by:

1. Preventing bias towards the majority class
2. Ensuring equal representation of all classes

Balancing Techniques

Common methods include:

- Oversampling
- Undersampling
- Synthetic data generation (e.g., SMOTE)

Best Practice

It's recommended to assess how these balancing adjustments affect the model's performance metrics..

Subham Beura
B521060
CE 7th Sem