Princess Nourah University

Department of Computer And Information Sciences

Masters of Data Science *First Term 2023*

Breast Cancer

Data Management and Visualization

This report is submitted in full fulfillment of the Data Management and Visualization Project

Academic year: 2023/2024

Author:

Sawsan Daban 445009481

Alaa Alsharekh 445009444

Supervisor:

Dr. Romana Aziz

ACKNOWLEDGEMENTS

We express our deep gratitude to the Almighty Allah for giving us an opportunity to successfully finish modeling and simulating and write this report.

We would like to express our gratitude and appreciation to Princess Nourah Bint Abdulrahman University for giving us the opportunity to continue learning and achieve new things in our life, and for their remarkable services they gave us.

We would like to express our gratitude to Dr. Romana Aziz for her kindness and unequivocal support.

We would like to thank our friends for helping us realize what We are missing from our life. We thank them for their interesting discussions and help. Without them, We would not be here.

Finally, We would like to thank our family for their unwavering support throughout our lives, pushing us to always do our best.

EXECUTIVE SUMMARY

This report includes information about the problem statement of the project, goals and objectives, data preparation, and data visualization.

Contents

List of Figures

List of Tables

None

Introduction

1.1 Background

Our Breast Cancer dataset appears to represent a sample of individuals, likely patients, from a medical context. These individuals are described based on various characteristics related to medical conditions.

Each row in the dataset represents an observation of an individual patient. There are multiple observations, each corresponding to a different patient in the dataset.

1.2 Problem Statement

The dataset comprises information about individuals diagnosed with breast cancer. It includes details such as age, menopausal status, tumor size, number of nodes, positivity or negativity of node-caps, degree of malignancy, affected breast, affected quadrant within the breast, receipt of irradiation treatment, and recurrence status of the cancer.

1.3 Goals And Objectives

- Acquire a comprehensive understanding of data preparation for analysis.
- Interpret data using graphical representations.
- Perform univariate and bivariate analysis using suitable data visualization techniques.

Data Preparation

2.1 Data Summary

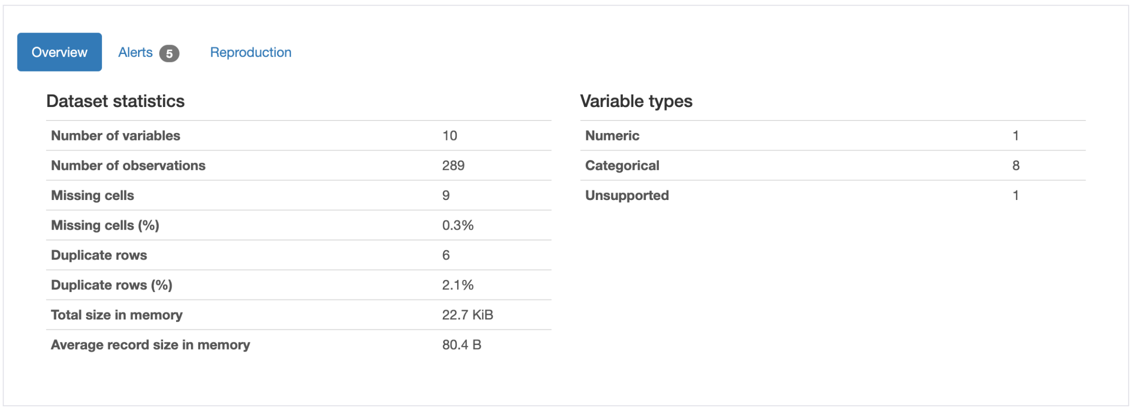
The dataset comprises several numerical features that provide valuable insights into the characteristics of individuals in the study. Firstly, the "Age" variable represents the age of the subjects, offering a continuous numerical measure. Moving on to the "Tumor-size" feature, despite its apparent string format denoting intervals ('15-19', '35-39', etc.), it fundamentally represents numerical data, allowing for a quantitative assessment of tumor sizes. Lastly, the "Nodes" variable contributes discrete numerical values, indicating the count of nodes associated with each individual. Together, these numerical attributes form a foundation for quantitative analysis and interpretation in the dataset.

The dataset also incorporates several categorical features that play a pivotal role in characterizing individuals within the study. Firstly, the "Menopause" variable classifies individuals based on their menopausal status, with categories including 'premeno' and 'ge40'. Moving on to the "Degree-of-malignance," this categorical feature reflects the degree of malignancy, with categories such as '3', '1', and '2'. The "Breast" variable categorizes individuals based on the side of the breast affected, distinguishing between 'right' and 'left'. Similarly, the "Breast-quad" feature categorizes the quadrant of the breast affected, with categories like 'left_up' and 'central'. The "Irradiation" variable is binary, indicating whether irradiation was administered, with categories 'yes' or 'no'. Finally, the "Recurrence" variable serves as a categorical indicator of events, with potential values being 'recurrence-events' or 'no-recurrence-events'. Together, these categorical attributes offer a comprehensive understanding of qualitative aspects in the dataset.

2.1.1 Quality Report

- *Overview* [1]

  The provided summary offers a concise overview of the existing dataset and highlights certain noteworthy aspects. According to the description, our dataset comprises 10 variables and encompasses a total of 289 observations. Among these variables, 8 fall under the categorical category, while 1 is designated as a numerical variable in addition to unsupported variable due to unknown type .2.1 Additionally, the summary points out the presence of duplicate rows and an unnamed column as part of the dataset's characteristics. 2.2



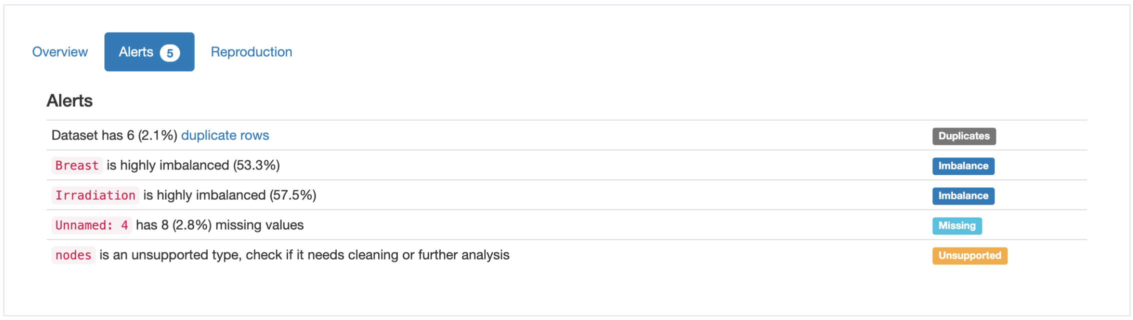
Dataset statistics

Number of variables	10
Number of observations	289
Missing cells	9
Missing cells (%)	0.3%
Duplicate rows	6
Duplicate rows (%)	2.1%
Total size in memory	22.7 KiB
Average record size in memory	80.4 B

Variable types

Numeric	1
Categorical	8
Unsupported	1

Figure 2.1: Quality report for row data



Alerts

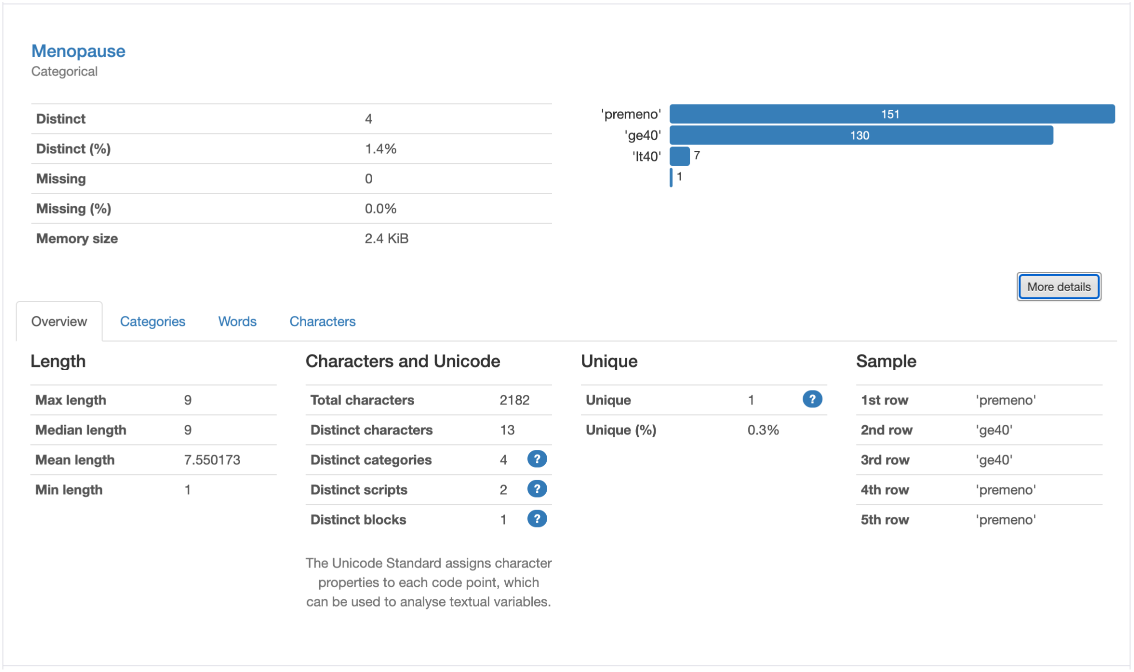
Dataset has 6 (2.1%) duplicate rows	Duplicates
`Breast` is highly imbalanced (53.3%)	Imbalance
`Irradiation` is highly imbalanced (57.5%)	Imbalance
`Unnamed: 4` has 8 (2.8%) missing values	Missing
`nodes` is an unsupported type, check if it needs cleaning or further analysis	Unsupported

Figure 2.2: Quality report showing current data set issues

[1]For more information, please check YData Profiling

- *V*ariables

  As an additional feature in the quality report, a comprehensive description of each variable has been provided in the report as shown in the example below. 2.3



Menopause
Categorical

Distinct	4
Distinct (%)	1.4%
Missing	0
Missing (%)	0.0%
Memory size	2.4 KiB

Length

Max length	9
Median length	9
Mean length	7.550173
Min length	1

Characters and Unicode

Total characters	2182
Distinct characters	13
Distinct categories	4
Distinct scripts	2
Distinct blocks	1

The Unicode Standard assigns character properties to each code point, which can be used to analyse textual variables.

Unique

Unique	1
Unique (%)	0.3%

Sample

1st row	'premeno'
2nd row	'ge40'
3rd row	'ge40'
4th row	'premeno'
5th row	'premeno'

Figure 2.3: Quality report for Menopause feature

After conducting a quality report, the identified challenges and issues within the dataset have been addressed. This process involves implementing various data preprocessing techniques to enhance the quality and reliability of the data. By handling missing values, duplicates, outliers, and other inconsistencies, analysts can create a more accurate dataset for analyses. This, in turn, enables informed decision-making and contributes to the overall reliability of the results obtained from the data.

2.2 Data Quality Control

Every set of data needs to undergo a form of quality control, which includes verifying for errors and ensuring uniform consistency, this might include :

1. Data consistency
2. Error checking
3. Zero versus null

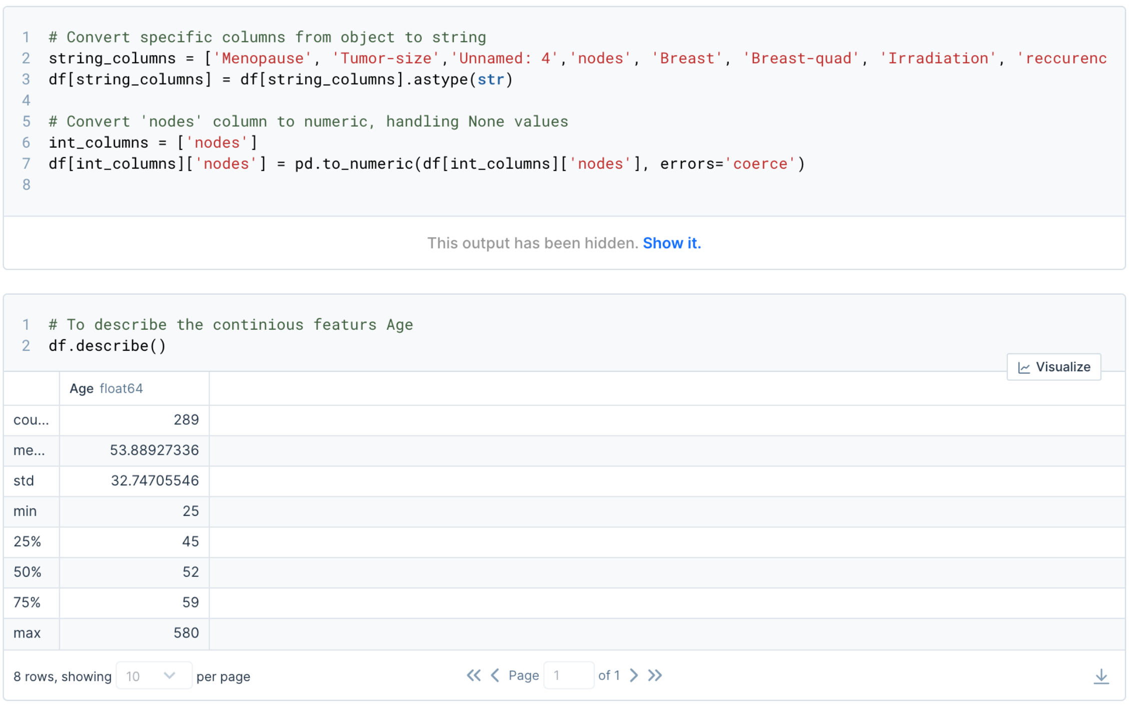
2.2.1 Data consistency

Ensuring data consistency involves modifying data types, examining the dataset for duplicate entries, confirming the presence of column names, and validating the conventions used for column naming.

- *Data type*

  Converting the data type is an efficient pre-processing to handle the data in a better and more optimized way. This process ensures that each variable is assigned the most suitable data type.

  During this step, the focus was on addressing unsupported variable types highlighted in the previous data quality report 2.2, with the aim of converting them into numeric types.



```python
# Convert specific columns from object to string
string_columns = ['Menopause', 'Tumor-size','Unnamed: 4','nodes', 'Breast', 'Breast-quad', 'Irradiation', 'reccurenc
df[string_columns] = df[string_columns].astype(str)

# Convert 'nodes' column to numeric, handling None values
int_columns = ['nodes']
df[int_columns]['nodes'] = pd.to_numeric(df[int_columns]['nodes'], errors='coerce')

```

This output has been hidden. Show it.

```python
# To describe the continious featurs Age
df.describe()
```

Visualize

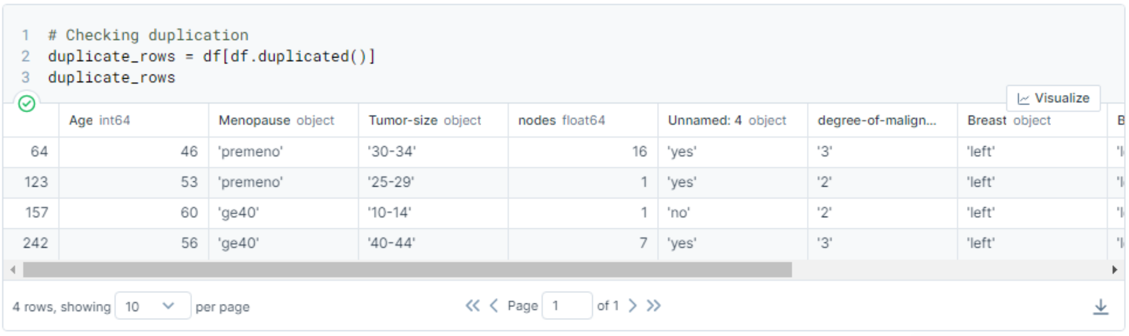
	Age float64
cou...	289
me...	53.88927336
std	32.74705546
min	25
25%	45
50%	52
75%	59
max	580

8 rows, showing 10 per page Page 1 of 1

Figure 2.4: Modify data types

- *Duplication*

  Initially, the dataset contained four duplicate rows. These were addressed by removing the duplicates.

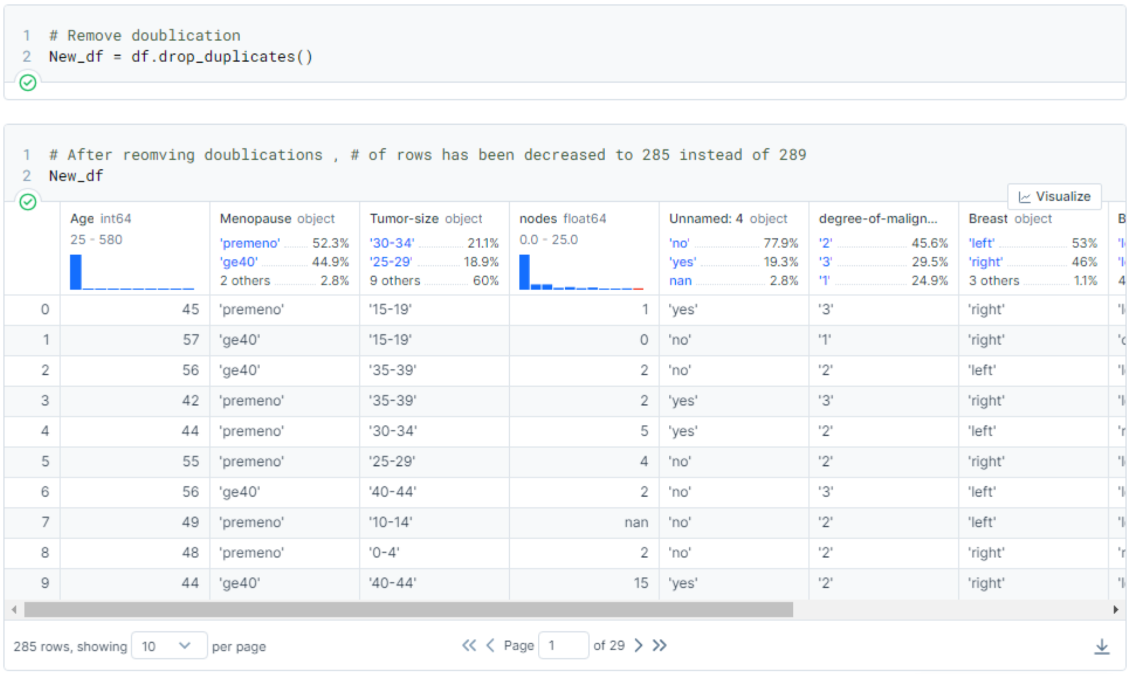


```python
# Checking duplication
duplicate_rows = df[df.duplicated()]
duplicate_rows
```

	Age int64	Menopause object	Tumor-size object	nodes float64	Unnamed: 4 object	degree-of-malign...	Breast object	B
64	46	'premeno'	'30-34'	16	'yes'	'3'	'left'	'l
123	53	'premeno'	'25-29'	1	'yes'	'2'	'left'	'l
157	60	'ge40'	'10-14'	1	'no'	'2'	'left'	'l
242	56	'ge40'	'40-44'	7	'yes'	'3'	'left'	'l

4 rows, showing 10 per page Page 1 of 1

Figure 2.5: Check for duplication



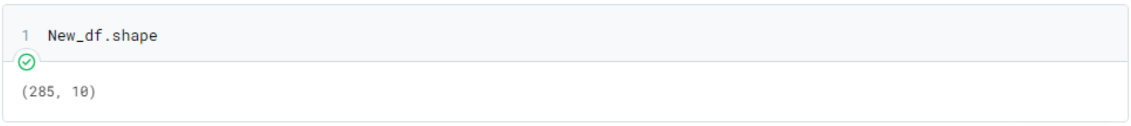
```python
# Remove doublication
New_df = df.drop_duplicates()
```

```python
# After reomving doublications , # of rows has been decreased to 285 instead of 289
New_df
```

	Age int64	Menopause object	Tumor-size object	nodes float64	Unnamed: 4 object	degree-of-malign...	Breast object	B
	25 - 580	'premeno' 52.3% 'ge40' 44.9% 2 others 2.8%	'30-34' 21.1% '25-29' 18.9% 9 others 60%	0.0 - 25.0	'no' 77.9% 'yes' 19.3% nan 2.8%	'2' 45.6% '3' 29.5% '1' 24.9%	'left' 53% 'right' 46% 3 others 1.1%	'l 'l 4
0	45	'premeno'	'15-19'	1	'yes'	'3'	'right'	'l
1	57	'ge40'	'15-19'	0	'no'	'1'	'right'	'c
2	56	'ge40'	'35-39'	2	'no'	'2'	'left'	'l
3	42	'premeno'	'35-39'	2	'yes'	'3'	'right'	'l
4	44	'premeno'	'30-34'	5	'yes'	'2'	'left'	'r
5	55	'premeno'	'25-29'	4	'no'	'2'	'right'	'l
6	56	'ge40'	'40-44'	2	'no'	'3'	'left'	'l
7	49	'premeno'	'10-14'	nan	'no'	'2'	'left'	'l
8	48	'premeno'	'0-4'	2	'no'	'2'	'right'	'r
9	44	'ge40'	'40-44'	15	'yes'	'2'	'right'	'l

285 rows, showing 10 per page Page 1 of 29

Figure 2.6: Remove duplication



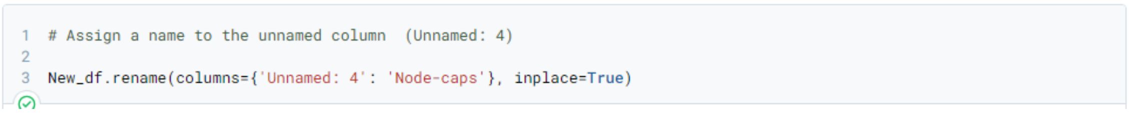
```python
New_df.shape
```

```
(285, 10)
```

Figure 2.7: Check the data shape after removing duplication

- *Missing Column Names*

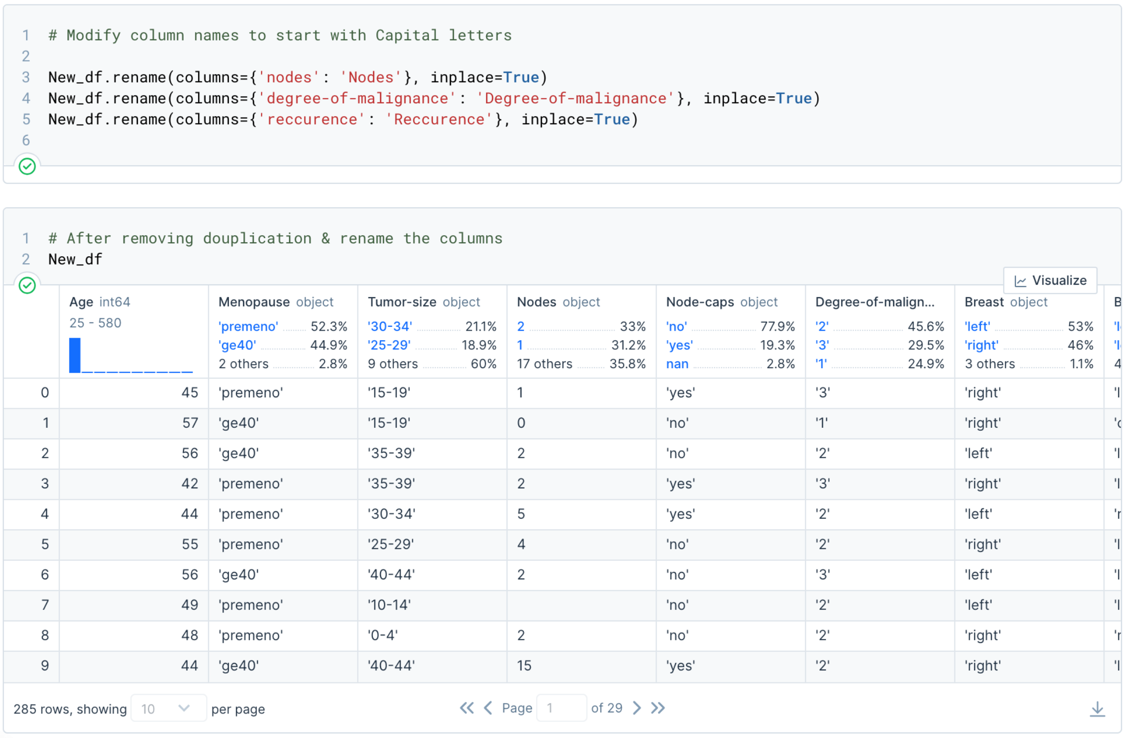
  It was noticed that one column was unnamed. In light of understanding the problem statement and the values associated with this feature, the decision was made to rename it to Node-caps.



```
# Assign a name to the unnamed column  (Unnamed: 4)

New_df.rename(columns={'Unnamed: 4': 'Node-caps'}, inplace=True)
```

Figure 2.8: Assign a name to the unnamed column

An additional step has been added, by unifying the column to start with capital letters.



```
# Modify column names to start with Capital letters

New_df.rename(columns={'nodes': 'Nodes'}, inplace=True)
New_df.rename(columns={'degree-of-malignance': 'Degree-of-malignance'}, inplace=True)
New_df.rename(columns={'reccurence': 'Reccurence'}, inplace=True)

```

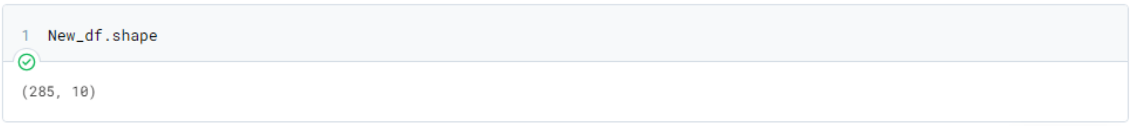
```
# After removing douplication & rename the columns
New_df
```

Visualize

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...	Breast object	B
	25 - 580	'premeno' 52.3% 'ge40' 44.9% 2 others 2.8%	'30-34' 21.1% '25-29' 18.9% 9 others 60%	2 33% 1 31.2% 17 others 35.8%	'no' 77.9% 'yes' 19.3% nan 2.8%	'2' 45.6% '3' 29.5% '1' 24.9%	'left' 53% 'right' 46% 3 others 1.1%	'l 'l 4
0	45	'premeno'	'15-19'	1	'yes'	'3'	'right'	'l
1	57	'ge40'	'15-19'	0	'no'	'1'	'right'	'c
2	56	'ge40'	'35-39'	2	'no'	'2'	'left'	'l
3	42	'premeno'	'35-39'	2	'yes'	'3'	'right'	'l
4	44	'premeno'	'30-34'	5	'yes'	'2'	'left'	'r
5	55	'premeno'	'25-29'	4	'no'	'2'	'right'	'l
6	56	'ge40'	'40-44'	2	'no'	'3'	'left'	'l
7	49	'premeno'	'10-14'		'no'	'2'	'left'	'l
8	48	'premeno'	'0-4'	2	'no'	'2'	'right'	'r
9	44	'ge40'	'40-44'	15	'yes'	'2'	'right'	'l

285 rows, showing 10 per page   Page 1 of 29

Figure 2.9: Modify column names to start with Capital letters



```
New_df.shape
```

```
(285, 10)
```

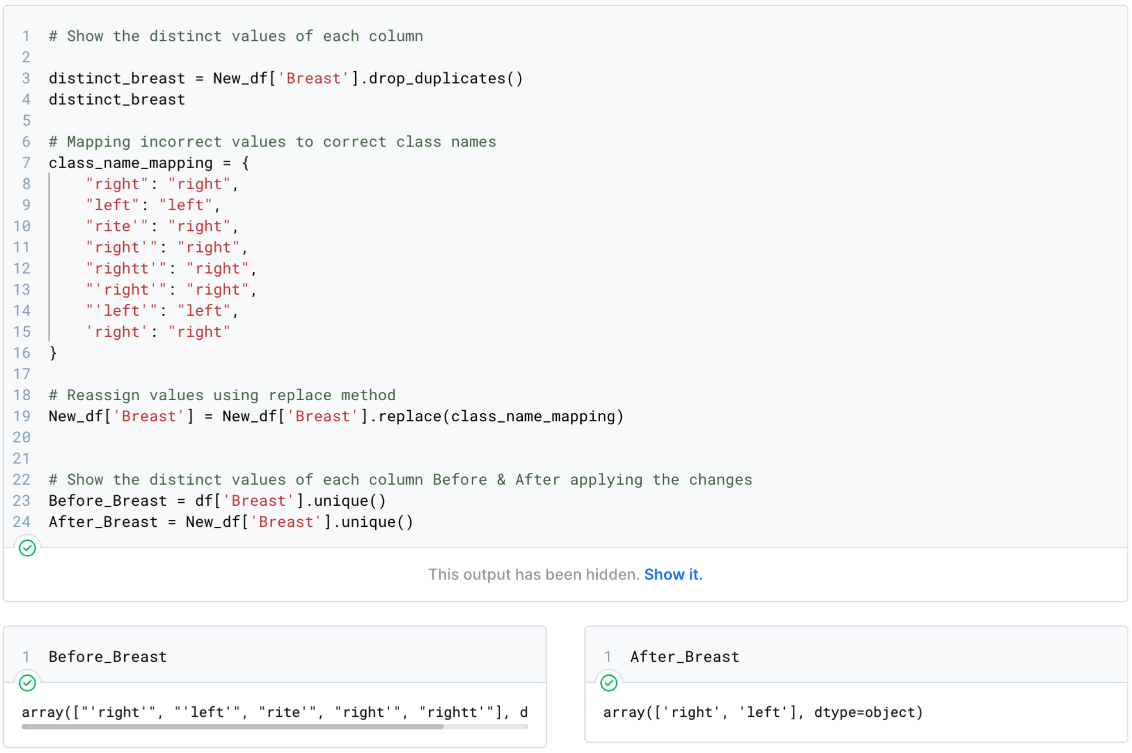
Figure 2.10: Dataset after removing duplicates

2.2.2 Error Checking

In error checking, a verification process is implemented to examine feature values and make necessary adjustments.

- *Typos* [2]

  As observed, certain features exhibit inconsistent values, such as Breast, Recurrence, and Irradiation. To address this issue, a unification of values is performed.



```python
# Show the distinct values of each column

distinct_breast = New_df['Breast'].drop_duplicates()
distinct_breast

# Mapping incorrect values to correct class names
class_name_mapping = {
    "right": "right",
    "left": "left",
    "rite'": "right",
    "right'": "right",
    "rightt'": "right",
    "'right'": "right",
    "'left'": "left",
    'right': "right"
}

# Reassign values using replace method
New_df['Breast'] = New_df['Breast'].replace(class_name_mapping)


# Show the distinct values of each column Before & After applying the changes
Before_Breast = df['Breast'].unique()
After_Breast = New_df['Breast'].unique()
```

This output has been hidden. Show it.

```python
Before_Breast
```

```
array(["'right'", "'left'", "rite'", "right'", "rightt'"], d
```

```python
After_Breast
```

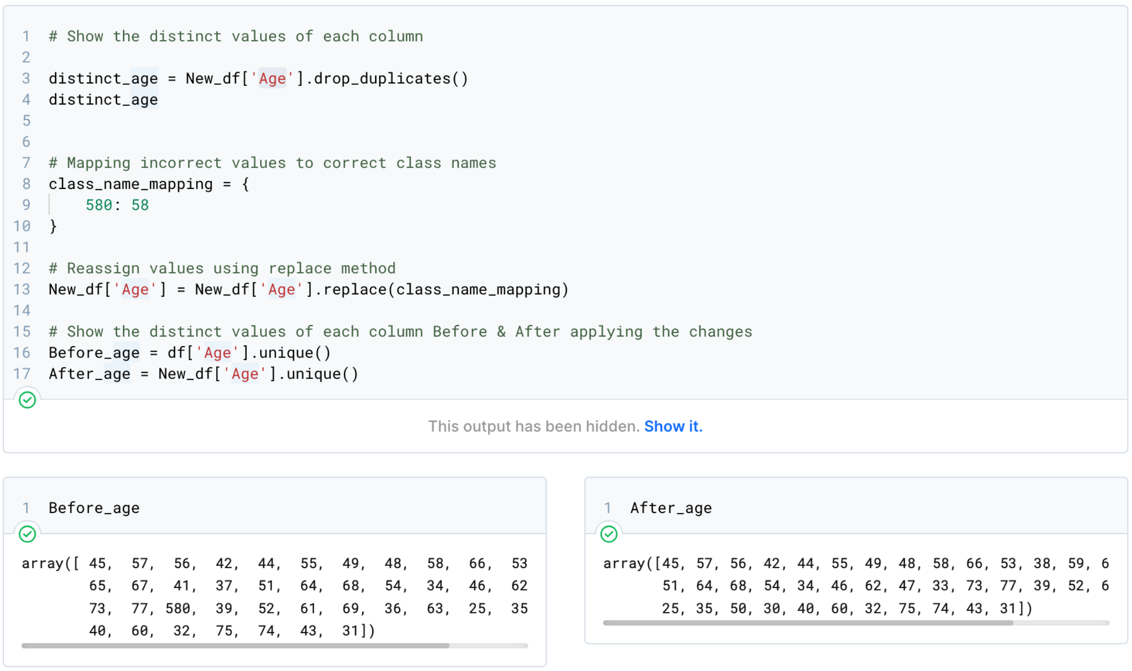
```
array(['right', 'left'], dtype=object)
```

Figure 2.11: Error checking - typos

[2]The remaining features are added in the Appendices

- *O*utliers

  The average Age is approximately 52. Excluding the outlier with the value 580, the maximum Age is 77, and the minimum Age is 25. The data anomaly associated with the value 580 has been identified and corrected as shown below. 2.12



```python
# Show the distinct values of each column

distinct_age = New_df['Age'].drop_duplicates()
distinct_age


# Mapping incorrect values to correct class names
class_name_mapping = {
    580: 58
}

# Reassign values using replace method
New_df['Age'] = New_df['Age'].replace(class_name_mapping)

# Show the distinct values of each column Before & After applying the changes
Before_age = df['Age'].unique()
After_age = New_df['Age'].unique()
```

This output has been hidden. **Show it.**

```python
Before_age
```

```
array([ 45,  57,  56,  42,  44,  55,  49,  48,  58,  66,  53
        65,  67,  41,  37,  51,  64,  68,  54,  34,  46,  62
        73,  77, 580,  39,  52,  61,  69,  36,  63,  25,  35
        40,  60,  32,  75,  74,  43,  31])
```

```python
After_age
```

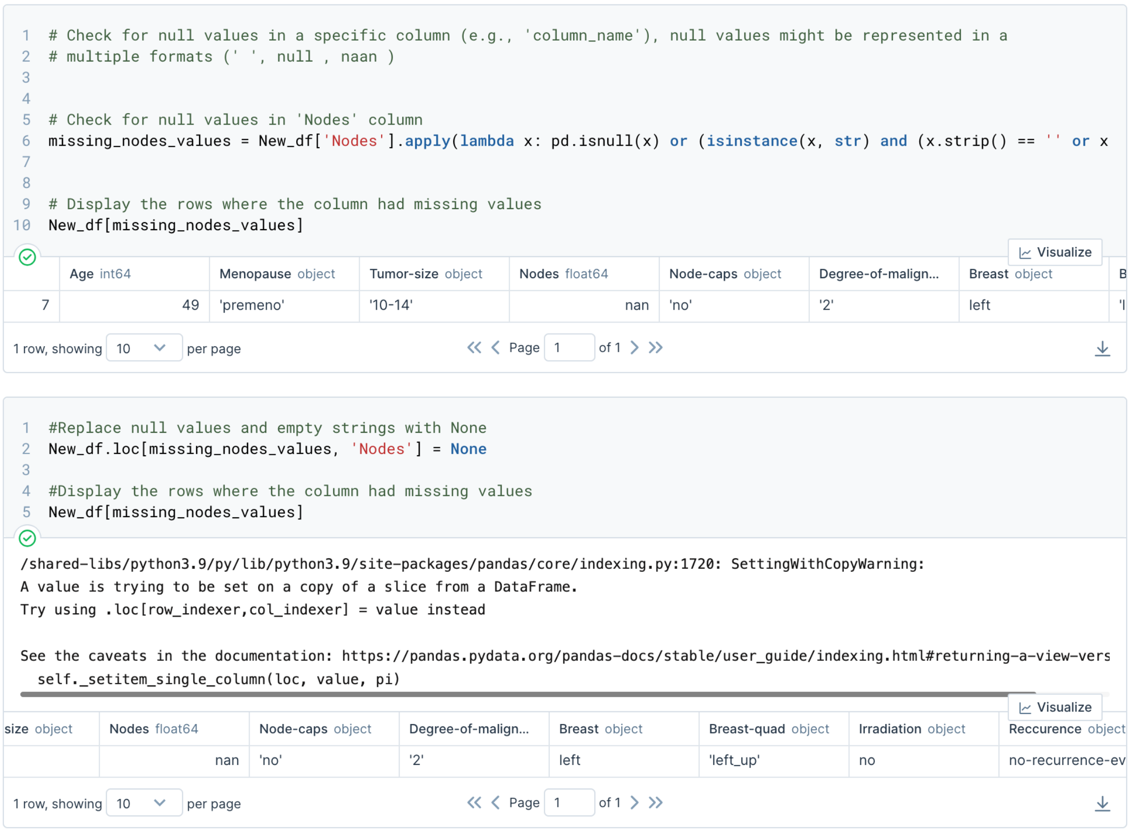
```
array([45, 57, 56, 42, 44, 55, 49, 48, 58, 66, 53, 38, 59, 6
       51, 64, 68, 54, 34, 46, 62, 47, 33, 73, 77, 39, 52, 6
       25, 35, 50, 30, 40, 60, 32, 75, 74, 43, 31])
```

Figure 2.12: Error checking - typos

2.2.3 Zero versus null

- *Null values* [3]

  One way for addressing null values involves examining various formats for representing null and then standardizing and replacing them with the term None.



```
# Check for null values in a specific column (e.g., 'column_name'), null values might be represented in a
# multiple formats (' ', null , naan )


# Check for null values in 'Nodes' column
missing_nodes_values = New_df['Nodes'].apply(lambda x: pd.isnull(x) or (isinstance(x, str) and (x.strip() == '' or x


# Display the rows where the column had missing values
New_df[missing_nodes_values]
```

Visualize

	Age int64	Menopause object	Tumor-size object	Nodes float64	Node-caps object	Degree-of-malign...	Breast object	B
7	49	'premeno'	'10-14'	nan	'no'	'2'	left	'l

1 row, showing 10 per page Page 1 of 1

```
#Replace null values and empty strings with None
New_df.loc[missing_nodes_values, 'Nodes'] = None

#Display the rows where the column had missing values
New_df[missing_nodes_values]
```

```
/shared-libs/python3.9/py/lib/python3.9/site-packages/pandas/core/indexing.py:1720: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers
  self._setitem_single_column(loc, value, pi)
```

Visualize

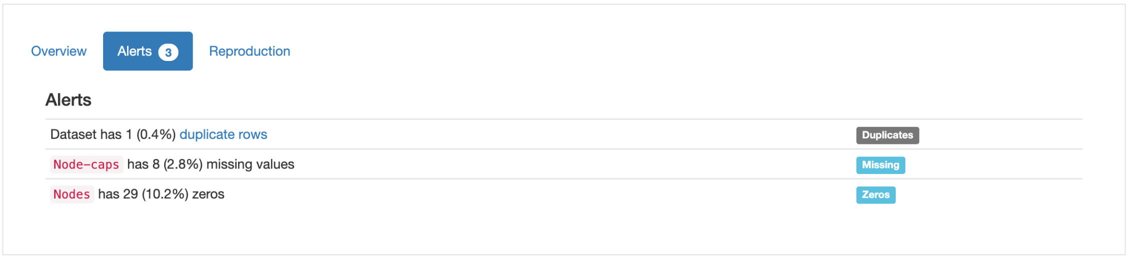
size object	Nodes float64	Node-caps object	Degree-of-malign...	Breast object	Breast-quad object	Irradiation object	Reccurence object
	nan	'no'	'2'	left	'left_up'	no	no-recurrence-ev

1 row, showing 10 per page Page 1 of 1

Figure 2.13: Handle null values

[3]The remaining features are added in the Appendices

2.2.4 Validation step

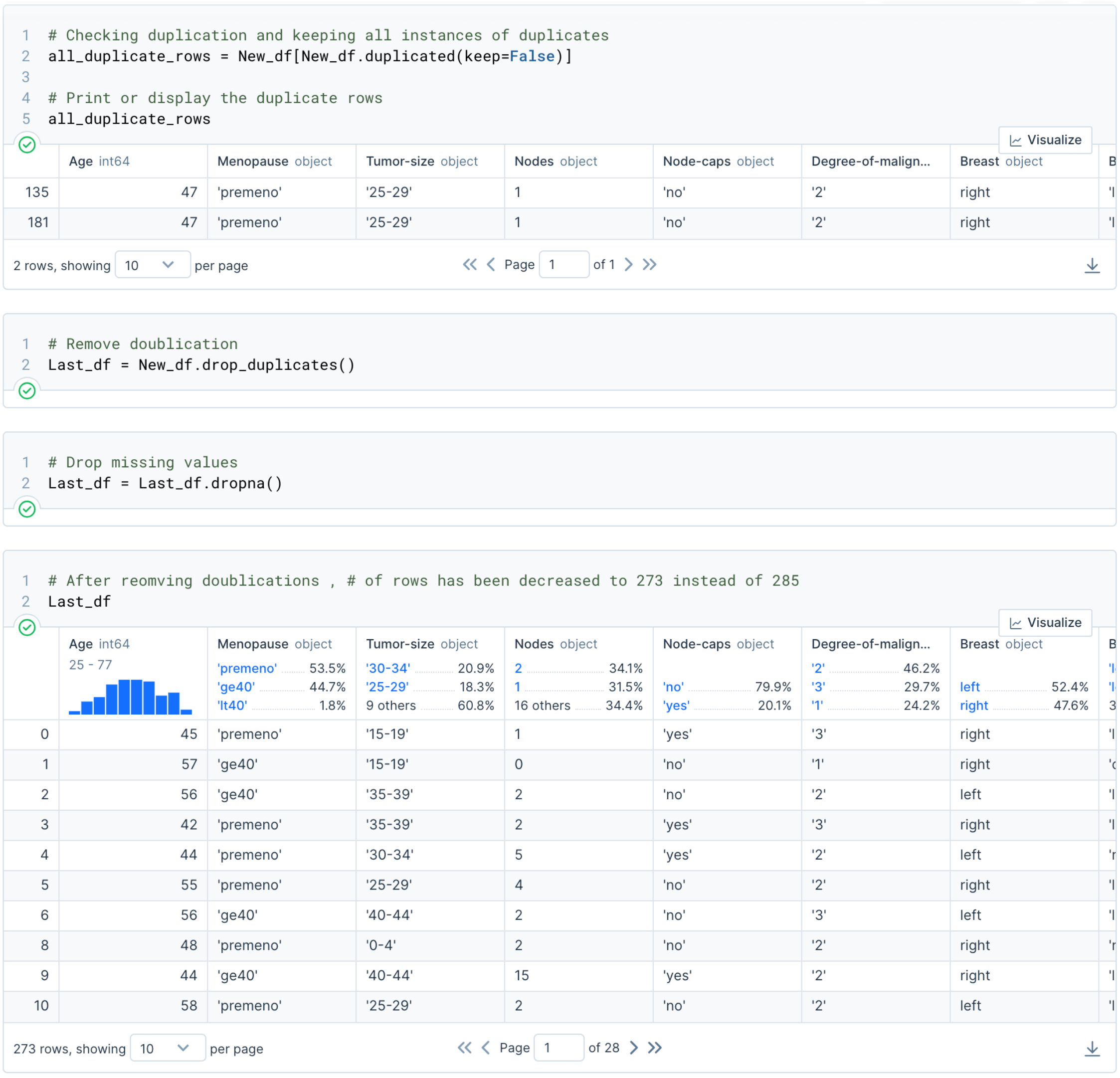


Alerts

Dataset has 1 (0.4%) duplicate rows	Duplicates
`Node-caps` has 8 (2.8%) missing values	Missing
`Nodes` has 29 (10.2%) zeros	Zeros

Figure 2.14: Validation step

Based on the report outcomes following the implementation of the necessary checks, it appears that additional duplicate rows have surfaced. Furthermore, we have identified an increased number of missing values in the dataset.



```
# Checking duplication and keeping all instances of duplicates
all_duplicate_rows = New_df[New_df.duplicated(keep=False)]

# Print or display the duplicate rows
all_duplicate_rows
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...	Breast object	B
135	47	'premeno'	'25-29'	1	'no'	'2'	right	'l
181	47	'premeno'	'25-29'	1	'no'	'2'	right	'l

2 rows, showing 10 per page Page 1 of 1

```
# Remove doublication
Last_df = New_df.drop_duplicates()
```

```
# Drop missing values
Last_df = Last_df.dropna()
```

```
# After reomving doublications , # of rows has been decreased to 273 instead of 285
Last_df
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...	Breast object	B
	25 - 77	'premeno' 53.5% 'ge40' 44.7% 'lt40' 1.8%	'30-34' 20.9% '25-29' 18.3% 9 others 60.8%	2 34.1% 1 31.5% 16 others 34.4%	'no' 79.9% 'yes' 20.1%	'2' 46.2% '3' 29.7% '1' 24.2%	left 52.4% right 47.6%	'l 'l 3
0	45	'premeno'	'15-19'	1	'yes'	'3'	right	'l
1	57	'ge40'	'15-19'	0	'no'	'1'	right	'c
2	56	'ge40'	'35-39'	2	'no'	'2'	left	'l
3	42	'premeno'	'35-39'	2	'yes'	'3'	right	'l
4	44	'premeno'	'30-34'	5	'yes'	'2'	left	'r
5	55	'premeno'	'25-29'	4	'no'	'2'	right	'l
6	56	'ge40'	'40-44'	2	'no'	'3'	left	'l
8	48	'premeno'	'0-4'	2	'no'	'2'	right	'r
9	44	'ge40'	'40-44'	15	'yes'	'2'	right	'l
10	58	'premeno'	'25-29'	2	'no'	'2'	left	'l

273 rows, showing 10 per page Page 1 of 28

Figure 2.15: Resolving dataset after validation

Univariate Bivariate Analysis

3.1 Univariate Analysis

In data analysis, the initial step involves looking at each important variable individually. Univariate graphs show the distribution of data for a single variable, which could be a category like Menopause or a quantity like Age.

3.1.1 Qualitative variable

- *Menopause* [1]

  The graph indicates that the occurrence of symptoms is least frequent in premenopausal women, followed by menopausal women, and then postmenopausal women. This implies that the transition through menopause itself is linked to an increase in symptoms, and the frequency of symptoms remains heightened even after a woman has entered the postmenopausal stage.

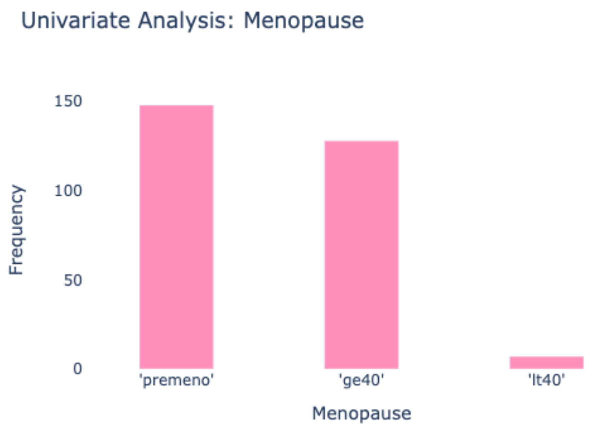


Figure 3.1: Univariate Analysis - Menopause

[1]The remaining univatiate features are added in the Appendices

- *Breast*

  The graph reveals that breast pain is more prevalent in the left breast compared to the right breast.

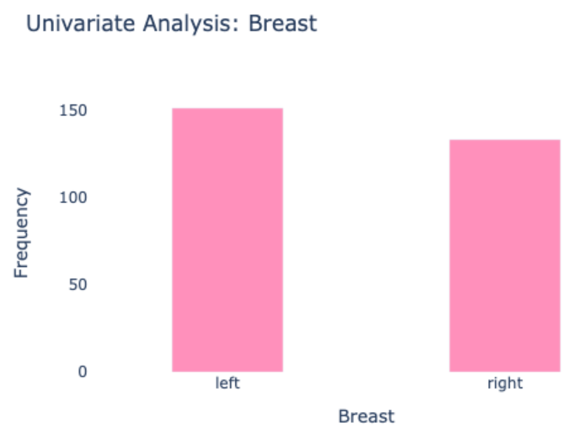


Figure 3.2: Univariate Analysis - Breast

3.1.2 Quantitative variable

- *Age* [2]

  The graph indicates a right-skewed age distribution, implying a higher concentration of individuals in younger age groups compared to older ones. The median age stands at 50, signifying that half of the sample is below 50, and the other half is above 50.

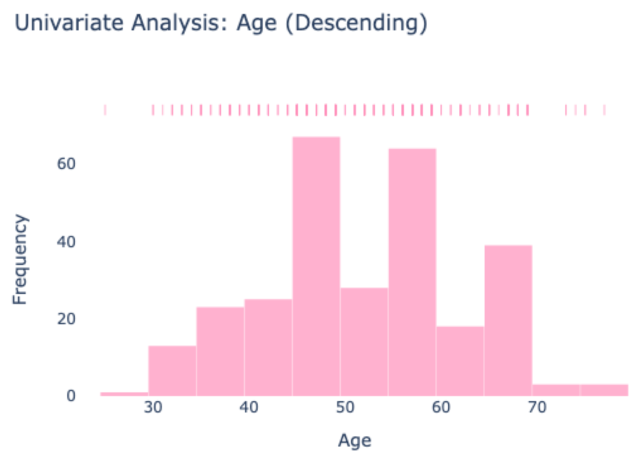


Figure 3.3: Univariate Analysis - Age

[2]The remaining univatiate features are added in the Appendices

- *Nodes*

  This suggests that the highest number of nodes is concentrated between 0 and 5.

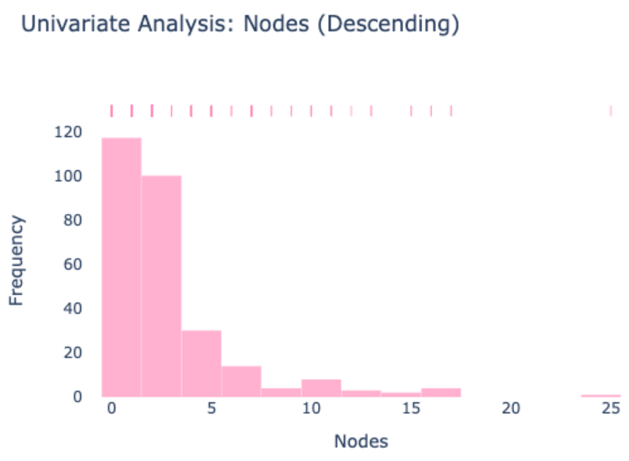


Figure 3.4: Univariate Analysis - Nodes

3.2 Bivariate Analysis

In research, a key question is understanding the relationship between two things, *A and B*. Bivariate graphs show this relationship by representing two variables together. The type of graph used depends on whether the variables are categories or quantities.

3.2.1 Categorical vs Quantitative

- *Menopause vs Age* [3]

  The provided graph illustrates the distribution of women in a sample based on their menopausal status at different ages. It reveals an increasing proportion of menopausal women with age. Specifically, at the age of 40, approximately 20% of women are in the menopausal stage, and by the age of 50, this proportion rises to around 50%. As women reach the age of 60, nearly all have undergone menopause.

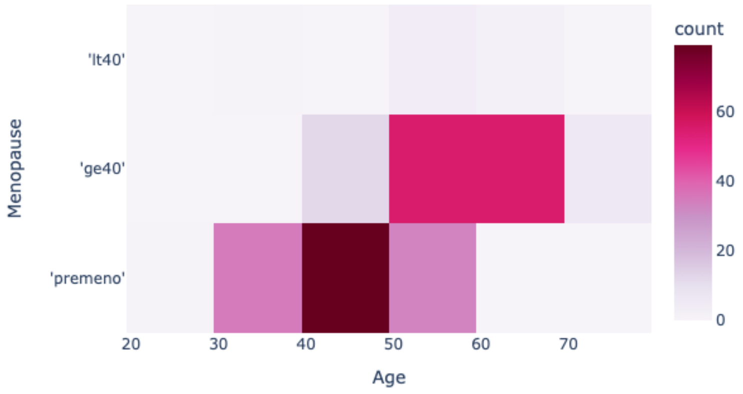


Figure 3.5: Bivariate Analysis - Menopause vs Age

[3]The remaining bivariate features are added in the Appendices

- *Menopause vs Tumor-size-encoded*

  The bivariate analysis comparing menopause and tumor-size-encoded reveals a negative correlation between the two variables. This implies that menopausal women are less likely to have large tumors compared to women who are not in the menopausal stage.

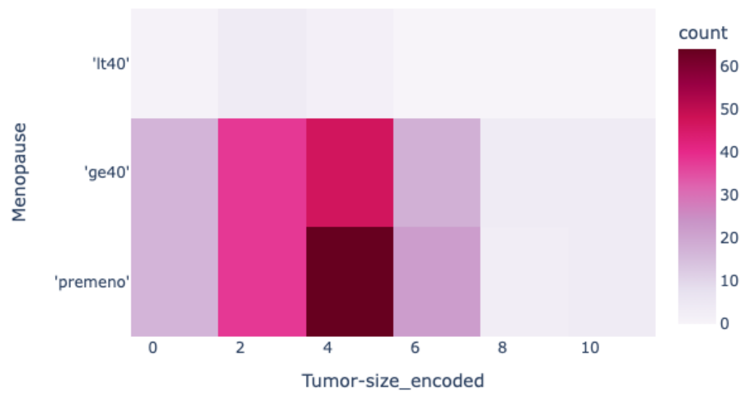


Figure 3.6: Bivariate Analysis - Menopause vs Tumor-size-encoded

3.2.2 Quantitative vs Quantitative

- *Age vs Nodes*

  The bivariate analysis indicates that there might be an association between age and an increased risk of having more nodes.

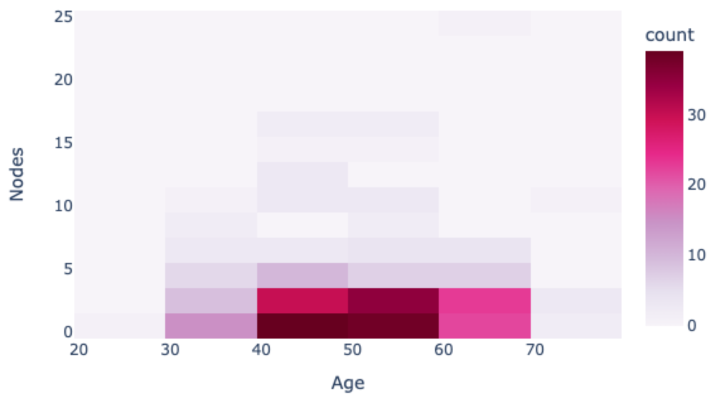


Figure 3.7: Bivariate Analysis - Age vs Node

- *Age vs Tumor-size$_e$ncoded*

  The bivariate analysis reveals a positive correlation between the two variables, indicating that older individuals are more likely to have larger tumors compared to younger individuals.

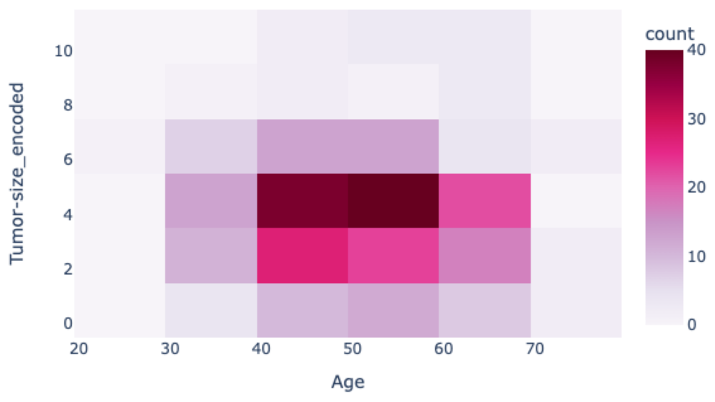


Figure 3.8: Bivariate Analysis - Age vs Tumor-size$_e$ncoded

Appendices

A Full Report

Data Management and Visualization Project

Sawsan Daban - Alaa AlSharekh

```
# Import Libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
! pip install openpyxl

# To read the data
RawNormal = 'Raw Normal.xlsx'
df = pd.read_excel(RawNormal)
```

```
# Display the current data frame as df
df
```

	Age int64	Menopause object	Tumor-size object	nodes object	Unnamed: 4 object	degree-of-malign...
	25 - 580	'premeno' 52.2% 'ge40' 45% 2 others 2.8%	'30-34' 21.1% '25-29' 19% 9 others 59.9%	2 32.5% 1 31.5% 17 others 36%	'no' 77.2% 'yes' 20.1% Missing 2.8%	'2' 45.7 '3' 29.8 '1' 24.6
0	45	'premeno'	'15-19'	1	'yes'	'3'
1	57	'ge40'	'15-19'	0	'no'	'1'
2	56	'ge40'	'35-39'	2	'no'	'2'
3	42	'premeno'	'35-39'	2	'yes'	'3'
4	44	'premeno'	'30-34'	5	'yes'	'2'
5	55	'premeno'	'25-29'	4	'no'	'2'
6	56	'ge40'	'40-44'	2	'no'	'3'
7	49	'premeno'	'10-14'		'no'	'2'
8	48	'premeno'	'0-4'	2	'no'	'2'
9	44	'ge40'	'40-44'	15	'yes'	'2'

289 rows, showing 10 per page   Page 1 of 29

Data Preparation

Data summary

Population: The dataset appears to represent a sample of individuals, likely patients, from a medical context. These individuals are described based on various characteristics related to medical conditions.

Observations: Each row in the dataset represents an observation of an individual patient. The dataset has multiple observations, each corresponding to a different patient.

Dataset

Quantitative features: 1. Age: Numerical data representing the age of individuals. 2. Tumor-size: Although it appears as a string ('15-19', '35-39', etc.), it represents numerical intervals. 3. Nodes: Numerical data representing the number of nodes. It is a discrete numerical variable.

Categorical features: 1. Menopause: Categorical data representing the menopausal status of individuals ('premeno', 'ge40'). 2. Degree-of-malignance: Categorical data representing the degree of malignancy ('3', '1', '2'). 3. Breast: Categorical data indicating the side of the breast ('right' or 'left'). 4. Breast-quad: Categorical data representing the quadrant of the breast ('left_up', 'central', etc.). 5. Irradiation: Categorical data indicating whether irradiation was done ('yes' or 'no'). 6. Recurrence: Categorical data indicating the occurrence of events ('recurrence-events' or 'no-recurrence-events').

Quality report

```
# Initial # of Columns vs Rows
df.shape
```

```
(289, 10)
```

```
# General Info about the features of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 289 entries, 0 to 288
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Age                   289 non-null    int64 
 1   Menopause             289 non-null    object
 2   Tumor-size            289 non-null    object
 3   nodes                 289 non-null    object
 4   Unnamed: 4            281 non-null    object
 5   degree-of-malignance  289 non-null    object
 6   Breast                289 non-null    object
 7   Breast-quad           288 non-null    object
 8   Irradiation           289 non-null    object
 9   reccurence            289 non-null    object
dtypes: int64(1), object(9)
memory usage: 22.7+ KB
```

```
# Data Profiling EDA
!pip install ydata-profiling
```

```
from ydata_profiling import ProfileReport

# Generate the data profiling report
report1 = ProfileReport(df, title='Raw Normal Profiling - report1 ')
report1.to_file("Raw Normal Profiling - report1.html")
```

```
!pip install ipywidgets==8.1.1
```

```
! pip install --upgrade pip
```

```
report1
```

Overview

Dataset statistics

Number of variables	10
Number of observations	289
Missing cells	9
Missing cells (%)	0.3%
Duplicate rows	6
Duplicate rows (%)	2.1%
Total size in memory	22.7 KiB
Average record size in memory	80.4 B

Variable types

Numeric	1
Categorical	8

The provided summary offers a concise overview of the existing dataset and highlights certain noteworthy aspects. According to the description, our dataset comprises 10 variables and encompasses a total of 289 observations. Among these variables, 8 fall under the categorical category, while 1 is designated as numerical variables & nodes with unknown variable type. Additionally, the summary points out the presence of duplicate rows and an unnamed column as part of the dataset's characteristics.

Please find below the full report: https://msc-science-in-computing-2023.github.io/DMV-Reports/report1/

Data quality control

Every set of data needs to undergo a form of quality control, which includes verifying for errors and ensuring uniform consistency, this might include : 1. Data consistency 2. Error checking 3. Zero versus null

Data Consistency

Data type

```
# Convert specific columns from object to string
string_columns = ['Menopause', 'Tumor-size','Unnamed: 4','nodes', 'Breast', 'Breast-quad', 'Irradiat
df[string_columns] = df[string_columns].astype(str)

# Convert 'nodes' column to numeric, handling None values
int_columns = ['nodes']
df[int_columns]['nodes'] = pd.to_numeric(df[int_columns]['nodes'], errors='coerce')
```

```
# To describe the continious featurs Age
df.describe()
```

	Age float64
cou...	289
me...	53.88927336
std	32.74705546
min	25
25%	45

50%	52
75%	59
max	580

8 rows, showing 10 per page Page 1 of 1

Duplications

Initially, the dataset has 4 duplicate rows. We handle them by removing duplicates.

```
# Checking duplication
duplicate_rows = df[df.duplicated()]
duplicate_rows
```

	Age int64	Menopause object	Tumor-size object	nodes object	Unnamed: 4 object	degree-of-malign...
64	46	'premeno'	'30-34'	16	'yes'	'3'
123	53	'premeno'	'25-29'	1	'yes'	'2'
157	60	'ge40'	'10-14'	1	'no'	'2'
242	56	'ge40'	'40-44'	7	'yes'	'3'

4 rows, showing 10 per page Page 1 of 1

```
# Remove doublication
New_df = df.drop_duplicates()
```

```
# After reomving doublications , # of rows has been decreased to 285 instead of 289
New_df
```

	Age int64	Menopause object	Tumor-size object	nodes object	Unnamed: 4 object	degree-of-malign...
	25 - 580	'premeno' 52.3% 'ge40' 44.9% 2 others 2.8%	'30-34' 21.1% '25-29' 18.9% 9 others 60%	2 33% 1 31.2% 17 others 35.8%	'no' 77.9% 'yes' 19.3% nan 2.8%	'2' 45.6 '3' 29.5 '1' 24.9
0	45	'premeno'	'15-19'	1	'yes'	'3'
1	57	'ge40'	'15-19'	0	'no'	'1'
2	56	'ge40'	'35-39'	2	'no'	'2'
3	42	'premeno'	'35-39'	2	'yes'	'3'
4	44	'premeno'	'30-34'	5	'yes'	'2'
5	55	'premeno'	'25-29'	4	'no'	'2'
6	56	'ge40'	'40-44'	2	'no'	'3'
7	49	'premeno'	'10-14'		'no'	'2'
8	48	'premeno'	'0-4'	2	'no'	'2'
9	44	'ge40'	'40-44'	15	'yes'	'2'

285 rows, showing 10 per page Page 1 of 29

Dataset size has been decreased from 289 to 285.

```
New_df.shape
```

```
(285, 10)
```

Missing Column Name

As observed, it was noted that one column lacked a name. In response to understanding the problem statement and the values associated with this feature, a decision was made to rename it to Node-caps.

```
# Assign a name to the unnamed column  (Unnamed: 4)

New_df.rename(columns={'Unnamed: 4': 'Node-caps'}, inplace=True)
```

An additional step has been added, by unifying the column to start with capital letters.

```
# Modify column names to start with Capital letters

New_df.rename(columns={'nodes': 'Nodes'}, inplace=True)
New_df.rename(columns={'degree-of-malignance': 'Degree-of-malignance'}, inplace=True)
New_df.rename(columns={'reccurence': 'Reccurence'}, inplace=True)
```

```
# After removing douplication & rename the columns
New_df
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
	25 - 580	'premeno' 52.3% 'ge40' 44.9% 2 others 2.8%	'30-34' 21.1% '25-29' 18.9% 9 others 60%	2 33% 1 31.2% 17 others 35.8%	'no' 77.9% 'yes' 19.3% nan 2.8%	'2' 45.6 '3' 29.5 '1' 24.9
0	45	'premeno'	'15-19'	1	'yes'	'3'
1	57	'ge40'	'15-19'	0	'no'	'1'
2	56	'ge40'	'35-39'	2	'no'	'2'
3	42	'premeno'	'35-39'	2	'yes'	'3'
4	44	'premeno'	'30-34'	5	'yes'	'2'
5	55	'premeno'	'25-29'	4	'no'	'2'
6	56	'ge40'	'40-44'	2	'no'	'3'
7	49	'premeno'	'10-14'		'no'	'2'
8	48	'premeno'	'0-4'	2	'no'	'2'
9	44	'ge40'	'40-44'	15	'yes'	'2'

285 rows, showing 10 per page   Page 1 of 29

Error Checking

Typos

To standardize the values, it was observed that certain entries contained typos.

```
# Show the distinct values of each column

distinct_breast = New_df['Breast'].drop_duplicates()
distinct_breast

# Mapping incorrect values to correct class names
class_name_mapping = {
    "right": "right",
    "left": "left",
    "rite'": "right",
    "right'": "right",
    "rightt'": "right",
    "'right'": "right",
    "'left'": "left",
    'right': "right"
}

# Reassign values using replace method
New_df['Breast'] = New_df['Breast'].replace(class_name_mapping)


# Show the distinct values of each column Before & After applying the changes
Before_Breast = df['Breast'].unique()
After_Breast = New_df['Breast'].unique()
```

```
Before_Breast
```

```
array(["'right'", "'left'", "rite'", "right'", "ri
```

```
After_Breast
```

```
array(['right', 'left'], dtype=object)
```

```
# Show the distinct values of each column

distinct_Reccurence = New_df['Reccurence'].drop_duplicates()
distinct_Reccurence

# Mapping incorrect values to correct class names
class_name_mapping = {
    "recurrence-events": "recurrence-events",
    "no-recurrence-events": "no-recurrence-events",
    "'recurrence-events'" : "recurrence-events",
    "'no-recurrence-events'" : "no-recurrence-events",
    "R E'": "recurrence-events"
}
```

```python
# Reassign values using replace method
New_df['Reccurence'] = New_df['Reccurence'].replace(class_name_mapping)

# Show the distinct values of each column Before & After applying the changes
Before_Reccurence = df['reccurence'].unique()
After_Reccurence = New_df['Reccurence'].unique()
```

```
Before_Reccurence
```

```
array(["'recurrence-events'", "'no-recurrence-even
      dtype=object)
```

```
After_Reccurence
```

```
array(['recurrence-events', 'no-recurrence-events'
```

```python
# Show the distinct values of each column

distinct_Irradiation = New_df['Irradiation'].drop_duplicates()
distinct_Irradiation


# Mapping incorrect values to correct class names
class_name_mapping = {
    "no": "no",
    "N": "no",
    "yes": "yes",
    "'yes'" : "yes",
    "'no'": "no"
}

# Reassign values using replace method
New_df['Irradiation'] = New_df['Irradiation'].replace(class_name_mapping)

# Show the distinct values of each column Before & After applying the changes
Before_Irradiation = df['Irradiation'].unique()
After_Irradiation = New_df['Irradiation'].unique()
```

```
Before_Irradiation
```

```
array(["'no'", "'yes'", ' ', 'N'], dtype=object)
```

```
After_Irradiation
```

```
array(['no', 'yes', ' '], dtype=object)
```

Outlier

The average Age is approximately 52. Excluding the outlier with the value 580, the maximum Age is 77, and the minimum Age is 25. The data anomaly associated with the value 580 has been identified and corrected as shown below.

```python
# Show the distinct values of each column
distinct_age = New_df['Age'].drop_duplicates()
distinct_age


# Mapping incorrect values to correct class names
class_name_mapping = {
    580: 58
}

# Reassign values using replace method
New_df['Age'] = New_df['Age'].replace(class_name_mapping)

# Show the distinct values of each column Before & After applying the changes
Before_age = df['Age'].unique()
After_age = New_df['Age'].unique()
```

```
Before_age
```

```
array([ 45,  57,  56,  42,  44,  55,  49,  48,  58
        65,  67,  41,  37,  51,  64,  68,  54,  34
        73,  77, 580,  39,  52,  61,  69,  36,  63
        40,  60,  32,  75,  74,  43,  31])
```

```
After_age
```

```
array([45, 57, 56, 42, 44, 55, 49, 48, 58, 66, 53,
       51, 64, 68, 54, 34, 46, 62, 47, 33, 73, 77,
       25, 35, 50, 30, 40, 60, 32, 75, 74, 43, 31]
```

Validation step

```python
from ydata_profiling import ProfileReport
```

```
# Generate the data profiling report to varify Data Consistency & Error Checking
report2 = ProfileReport(New_df, title='Raw Normal Profiling - report2')
report2.to_file("Raw Normal Profiling - report2.html")
```

```
report2
```

Overview

Dataset statistics

Number of variables	10
Number of observations	285
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	1
Duplicate rows (%)	0.4%
Total size in memory	24.5 KiB
Average record size in memory	88.0 B

Variable types

Numeric	1

Please find below the full report: https://msc-science-in-computing-2023.github.io/DMV-Reports/report2/

Zero versus null

Null Values

Some techniques to handle Null values:

1. Removing Rows with Null Values. 2. Filling Null Values with a Specific Value like 'NA', 'Unknown' .. etc

```
# Check for null values in a specific column (e.g., 'column_name'), null values might be represented
# multiple formats (' ', null , naan )


# Check for null values in 'Menopause' column
missing_menopause_values = New_df['Menopause'].apply(lambda x: pd.isnull(x) or (isinstance(x, str) a


# Display the rows where the column had missing values
New_df[missing_menopause_values]
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
36	46		'10-14'	0	'no'	'2'

1 row, showing 10 per page   Page 1 of 1

```
# Replace null values and empty strings with None
New_df.loc[missing_menopause_values, 'Menopause'] = None

# Display the rows where the column had missing values
New_df[missing_menopause_values]
```

```
/shared-libs/python3.9/py/lib/python3.9/site-packages/pandas/core/indexing.py:1720: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
  self._setitem_single_column(loc, value, pi)
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
36	46	None	'10-14'	0	'no'	'2'

1 row, showing 10 per page   Page 1 of 1

```
New_df['Menopause'].unique()
```

```
array(["'premeno'", "'ge40'", "'lt40'", None], dtype=object)
```

```
# Check for null values in a specific column (e.g., 'column_name'), null values might be represented
# multiple formats (' ', null , naan )


# Check for null values in 'Nodes' column
missing_nodes_values = New_df['Nodes'].apply(lambda x: pd.isnull(x) or (isinstance(x, str) and (x.st


# Display the rows where the column had missing values
New_df[missing_nodes_values]
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
7	49	'premeno'	'10-14'		'no'	'2'

1 row, showing 10 per page   Page 1 of 1

```
#Replace null values and empty strings with None
New_df.loc[missing_nodes_values, 'Nodes'] = None

#Display the rows where the column had missing values
New_df[missing_nodes_values]
```

```
/shared-libs/python3.9/py/lib/python3.9/site-packages/pandas/core/indexing.py:1720: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
  self._setitem_single_column(loc, value, pi)
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
7	49	'premeno'	'10-14'	None	'no'	'2'

1 row, showing 10 per page   Page 1 of 1

```
New_df['Nodes'].unique()
```

```
array(['1', '0', '2', '5', '4', None, '15', '7', '3', '17', '10', '16',
       '8', '6', '11', '9', '25', '13', '12'], dtype=object)
```

```
# Check for null values in a specific column (e.g., 'column_name'), null values might be represented
# multiple formats (' ', null , naan )


# Check for missing values in the 'Node-caps' column
missing_nodescaps_values = New_df['Node-caps'].apply(lambda x: pd.isnull(x) or (isinstance(x, str) a

# Print the rows where the column has missing values
New_df[missing_nodescaps_values]
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
20	56	'lt40'	'20-24'	2	nan	'1'
31	68	'ge40'	'25-29'	3	nan	'1'

50	73	'ge40'	'15-19'	10	nan	'1'
54	48	'premeno'	'25-29'	1	nan	'2'
72	61	'ge40'	'25-29'	5	nan	'1'
93	51	'lt40'	'20-24'	0	nan	'1'
151	50	'ge40'	'30-34'	9	nan	'3'
267	57	'ge40'	'30-34'	11	nan	'3'

8 rows, showing 10 per page Page 1 of 1

```
# Replace null values and empty strings with None
New_df.loc[missing_nodescaps_values, 'Node-caps'] = None

# Display the rows where the column had missing values
New_df[missing_nodescaps_values]
```

```
/shared-libs/python3.9/py/lib/python3.9/site-packages/pandas/core/indexing.py:1720: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
  self._setitem_single_column(loc, value, pi)
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
20	56	'lt40'	'20-24'	2	None	'1'
31	68	'ge40'	'25-29'	3	None	'1'
50	73	'ge40'	'15-19'	10	None	'1'
54	48	'premeno'	'25-29'	1	None	'2'
72	61	'ge40'	'25-29'	5	None	'1'
93	51	'lt40'	'20-24'	0	None	'1'
151	50	'ge40'	'30-34'	9	None	'3'
267	57	'ge40'	'30-34'	11	None	'3'

8 rows, showing 10 per page Page 1 of 1

```
New_df['Node-caps'].unique()
```

```
array(["'yes'", "'no'", None], dtype=object)
```

```
# Check for null values in a specific column (e.g., 'column_name'), null values might be represented
# multiple formats (' ', null , naan )


# Check for missing values in the 'Breast-quad' column
missing_breastquad_values = New_df['Breast-quad'].apply(lambda x: pd.isnull(x) or (isinstance(x, str)

# Print the rows where the column has missing values
New_df[missing_breastquad_values]
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
243	59	'ge40'	'30-34'	1	'no'	'3'

1 row, showing 10 per page Page 1 of 1

```
# Replace null values and empty strings with None
New_df.loc[missing_breastquad_values, 'Breast-quad'] = None

# Display the rows where the column had missing values
New_df[missing_breastquad_values]
```

```
/shared-libs/python3.9/py/lib/python3.9/site-packages/pandas/core/indexing.py:1720: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
  self._setitem_single_column(loc, value, pi)
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
243	59	'ge40'	'30-34'	1	'no'	'3'

1 row, showing 10 per page Page 1 of 1

```
New_df['Breast-quad'].unique()
```

```
array(["'left_up'", "'central'", "'left_low'", "'right_up'",
       "'right_low'", None], dtype=object)
```

```
# Check for null values in a specific column (e.g., 'column_name'), null values might be represented
# multiple formats (' ', null , naan )


# Check for missing values in the 'Irradiation' column
missing_irradiation_values = New_df['Irradiation'].apply(lambda x: pd.isnull(x) or (isinstance(x, st

# Print the rows where the column has missing values
New_df[missing_irradiation_values]
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
93	51	'lt40'	'20-24'	0	None	'1'

1 row, showing 10 per page Page 1 of 1

```
# Replace null values and empty strings with None
New_df.loc[missing_irradiation_values, 'Irradiation'] = None

# Display the rows where the column had missing values
New_df[missing_irradiation_values]
```

```
/shared-libs/python3.9/py/lib/python3.9/site-packages/pandas/core/indexing.py:1720: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
  self._setitem_single_column(loc, value, pi)
```

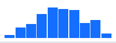
	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
93	51	'lt40'	'20-24'	0	None	'1'

1 row, showing 10 per page Page 1 of 1

```
New_df['Irradiation'].unique()
```

```
array(['no', 'yes', None], dtype=object)
```

```
New_df
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
	25 - 77 	'premeno' 52.3% 2 others 47.4% Missing 0.4%	'30-34' 21.1% '25-29' 18.9% 9 others 60%	2 33% 17 others 66.7% Missing 0.4%	'no' 77.9% 'yes' 19.3% Missing 2.8%	'2' 45.6 '3' 29.5 '1' 24.9
0	45	'premeno'	'15-19'	1	'yes'	'3'
1	57	'ge40'	'15-19'	0	'no'	'1'
2	56	'ge40'	'35-39'	2	'no'	'2'
3	42	'premeno'	'35-39'	2	'yes'	'3'
4	44	'premeno'	'30-34'	5	'yes'	'2'
5	55	'premeno'	'25-29'	4	'no'	'2'
6	56	'ge40'	'40-44'	2	'no'	'3'
7	49	'premeno'	'10-14'	None	'no'	'2'
8	48	'premeno'	'0-4'	2	'no'	'2'
9	44	'ge40'	'40-44'	15	'yes'	'2'

285 rows, showing 10 per page Page 1 of 29

Validation step

```python
from ydata_profiling import ProfileReport

# Generate the data profiling report to varify Data Consistency & Error Checking
report3 = ProfileReport(New_df, title='Raw Normal Profiling - report3')
report3.to_file("Raw Normal Profiling - report3.html")
```

```python
report3
```

Overview

Dataset statistics

Number of variables	10
Number of observations	285
Missing cells	12
Missing cells (%)	0.4%
Duplicate rows	1
Duplicate rows (%)	0.4%
Total size in memory	32.6 KiB
Average record size in memory	117.1 B

Variable types

Numeric	1

Please find below the full report: https://msc-science-in-computing-2023.github.io/DMV-Reports/report3/

Based on the report outcomes following the implementation of the necessary checks, it appears that additional duplicate rows have surfaced. Furthermore, we have identified an increased number of missing values in the dataset.

```python
# Checking duplication and keeping all instances of duplicates
all_duplicate_rows = New_df[New_df.duplicated(keep=False)]

# Print or display the duplicate rows
all_duplicate_rows
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
135	47	'premeno'	'25-29'	1	'no'	'2'
181	47	'premeno'	'25-29'	1	'no'	'2'

2 rows, showing 10 per page   Page 1 of 1

```python
# Remove doublication
Last_df = New_df.drop_duplicates()
```

```python
# Drop missing values
Last_df = Last_df.dropna()
```

```python
# After reomving doublications , # of rows has been decreased to 273 instead of 285
Last_df
```

	Age int64	Menopause object	Tumor-size object	Nodes object	Node-caps object	Degree-of-malign...
	25 - 77	'premeno' 53.5% 'ge40' 44.7%	'30-34' 20.9% '25-29' 18.3%	2 34.1% 1 31.5%	'no' 79.9%	'2' 46.2 '3' 29.7

		'lt40' 1.8%	9 others 60.8%	16 others 34.4%	'yes' 20.1%	'1' 24.2
0	45	'premeno'	'15-19'	1	'yes'	'3'
1	57	'ge40'	'15-19'	0	'no'	'1'
2	56	'ge40'	'35-39'	2	'no'	'2'
3	42	'premeno'	'35-39'	2	'yes'	'3'
4	44	'premeno'	'30-34'	5	'yes'	'2'
5	55	'premeno'	'25-29'	4	'no'	'2'
6	56	'ge40'	'40-44'	2	'no'	'3'
8	48	'premeno'	'0-4'	2	'no'	'2'
9	44	'ge40'	'40-44'	15	'yes'	'2'
10	58	'premeno'	'25-29'	2	'no'	'2'

273 rows, showing 10 per page   Page 1 of 28

```
Last_df.shape
```

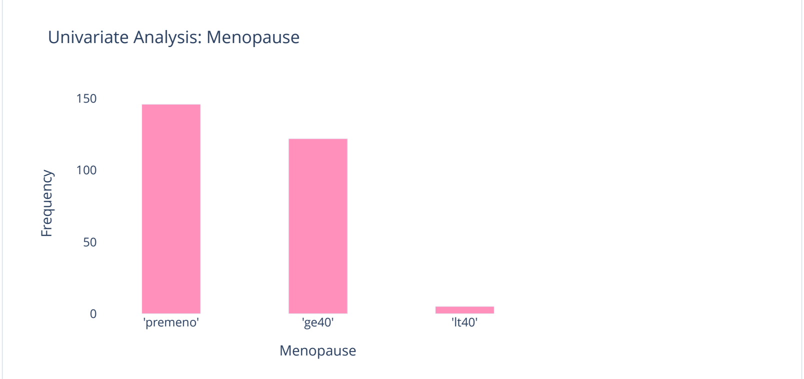
```
(273, 10)
```

Univariate & Bivariate analysis

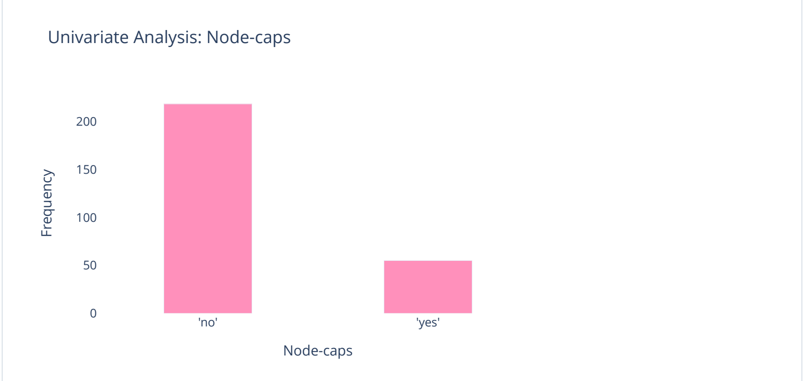
Univariate Analysis

Qualitative variable

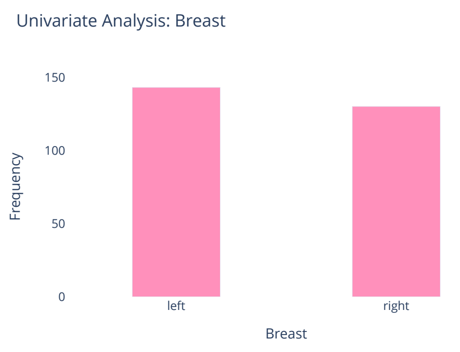
```
import plotly.graph_objects as go

# Specify the size of the graph
graph_width = 600
graph_height = 400

# Select the column for analysis
column = 'Menopause'

# Calculate the count of each category, including None values
column_counts = Last_df[column].value_counts(dropna=False).sort_index()

# Sort the values in descending order
column_counts_sorted = column_counts.sort_values(ascending=False)

# Define the custom bar color
custom_color = 'rgba(255, 144, 187, 1)'

# Create a bar chart
fig = go.Figure([go.Bar(x=column_counts_sorted.index, y=column_counts_sorted.values, marker_color=cu

# Update layout for better visualization and adjust the size
fig.update_layout(title_text=f'Univariate Analysis: {column}',
                  xaxis_title=column,
                  yaxis_title='Frequency',
                  paper_bgcolor='white',  # Set background color to white
                  plot_bgcolor='white',  # Set plot background color to white
                  bargap=0.2,  # Adjust the gap between bars
                  width=graph_width,
                  height=graph_height)

# Show the plot
fig.show()
```



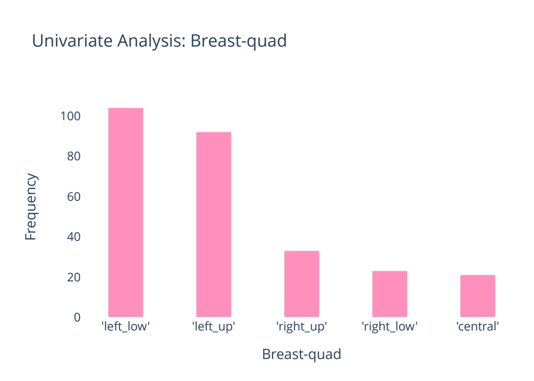
```
import plotly.graph_objects as go

# Specify the size of the graph
graph_width = 600
graph_height = 400

# Select the column for analysis
column = 'Node-caps'

# Calculate the count of each category, including None values
column_counts = Last_df[column].value_counts(dropna=False).sort_index()

# Sort the values in descending order
column_counts_sorted = column_counts.sort_values(ascending=False)

# Define the custom bar color
custom_color = 'rgba(255, 144, 187, 1)'

# Create a bar chart
fig = go.Figure([go.Bar(x=column_counts_sorted.index, y=column_counts_sorted.values, marker_color=cu

# Update layout for better visualization and adjust the size
fig.update_layout(title_text=f'Univariate Analysis: {column}',
                  xaxis_title=column,
                  yaxis_title='Frequency',
                  paper_bgcolor='white',  # Set background color to white
                  plot_bgcolor='white',  # Set plot background color to white
                  bargap=0.2,  # Adjust the gap between bars
                  width=graph_width,
                  height=graph_height)

# Show the plot
fig.show()
```



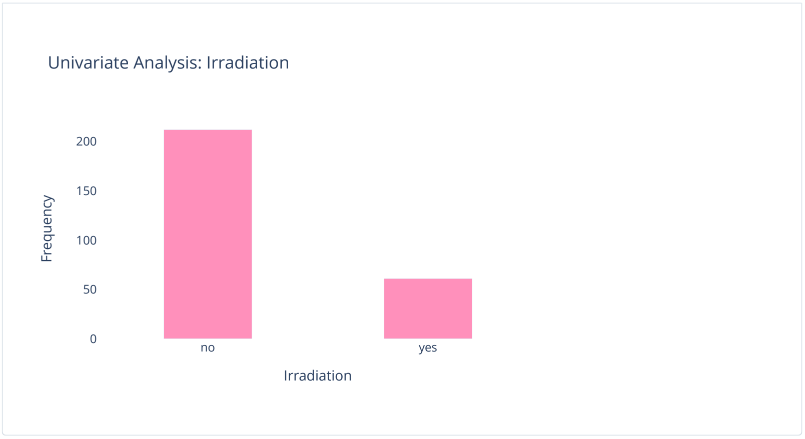
```
import plotly.graph_objects as go

# Specify the size of the graph
graph_width = 600
graph_height = 400

# Select the column for analysis
column = 'Breast'

# Calculate the count of each category, including None values
column_counts = Last_df[column].value_counts(dropna=False).sort_index()

# Sort the values in descending order
column_counts_sorted = column_counts.sort_values(ascending=False)

# Define the custom bar color
custom_color = 'rgba(255, 144, 187, 1)'

# Create a bar chart
fig = go.Figure([go.Bar(x=column_counts_sorted.index, y=column_counts_sorted.values, marker_color=cus

# Update layout for better visualization and adjust the size
fig.update_layout(title_text=f'Univariate Analysis: {column}',
                  xaxis_title=column,
                  yaxis_title='Frequency',
                  paper_bgcolor='white',  # Set background color to white
                  plot_bgcolor='white',  # Set plot background color to white
                  bargap=0.2,  # Adjust the gap between bars
                  width=graph_width,
                  height=graph_height)

# Show the plot
fig.show()
```



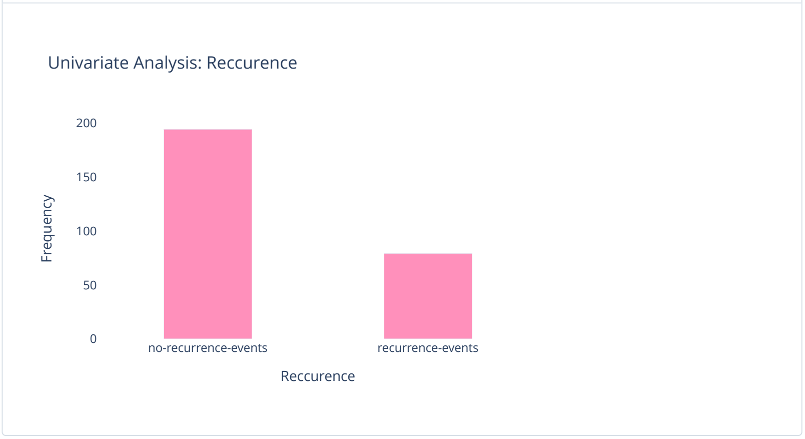
```
import plotly.graph_objects as go

# Specify the size of the graph
graph_width = 600
graph_height = 400

# Select the column for analysis
column = 'Breast-quad'

# Calculate the count of each category, including None values
column_counts = Last_df[column].value_counts(dropna=False).sort_index()

# Sort the values in descending order
column_counts_sorted = column_counts.sort_values(ascending=False)

# Define the custom bar color
custom_color = 'rgba(255, 144, 187, 1)'

# Create a bar chart
fig = go.Figure([go.Bar(x=column_counts_sorted.index, y=column_counts_sorted.values, marker_color=cus

# Update layout for better visualization and adjust the size
fig.update_layout(title_text=f'Univariate Analysis: {column}',
                  xaxis_title=column,
                  yaxis_title='Frequency',
                  paper_bgcolor='white',  # Set background color to white
                  plot_bgcolor='white',  # Set plot background color to white
                  bargap=0.2,  # Adjust the gap between bars
                  width=graph_width,
                  height=graph_height)

# Show the plot
fig.show()
```



```
import plotly.graph_objects as go

# Specify the size of the graph
graph_width = 600
graph_height = 400

# Select the column for analysis
column = 'Irradiation'

# Calculate the count of each category, including None values
column_counts = Last_df[column].value_counts(dropna=False).sort_index()

# Sort the values in descending order
column_counts_sorted = column_counts.sort_values(ascending=False)

# Define the custom bar color
custom_color = 'rgba(255, 144, 187, 1)'

# Create a bar chart
fig = go.Figure([go.Bar(x=column_counts_sorted.index, y=column_counts_sorted.values, marker_color=cu

# Update layout for better visualization and adjust the size
fig.update_layout(title_text=f'Univariate Analysis: {column}',
                  xaxis_title=column,
                  yaxis_title='Frequency',
                  paper_bgcolor='white',  # Set background color to white
                  plot_bgcolor='white',  # Set plot background color to white
                  bargap=0.2,  # Adjust the gap between bars
                  width=graph_width,
                  height=graph_height)

# Show the plot
fig.show()
```

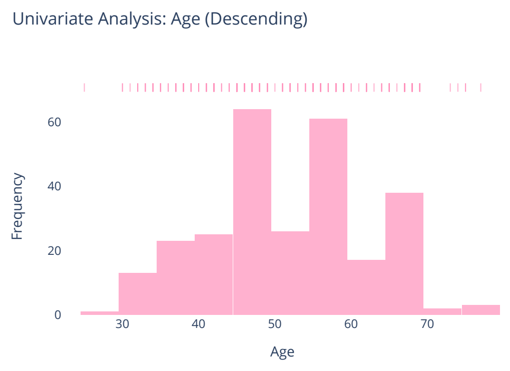


```python
import plotly.graph_objects as go

# Specify the size of the graph
graph_width = 600
graph_height = 400

# Select the column for analysis
column = 'Reccurence'

# Calculate the count of each category, including None values
column_counts = Last_df[column].value_counts(dropna=False).sort_index()

# Sort the values in descending order
column_counts_sorted = column_counts.sort_values(ascending=False)

# Define the custom bar color
custom_color = 'rgba(255, 144, 187, 1)'

# Create a bar chart
fig = go.Figure([go.Bar(x=column_counts_sorted.index, y=column_counts_sorted.values, marker_color=cus

# Update layout for better visualization and adjust the size
fig.update_layout(title_text=f'Univariate Analysis: {column}',
                  xaxis_title=column,
                  yaxis_title='Frequency',
                  paper_bgcolor='white',  # Set background color to white
                  plot_bgcolor='white',  # Set plot background color to white
                  bargap=0.2,  # Adjust the gap between bars
                  width=graph_width,
                  height=graph_height)

# Show the plot
fig.show()
```

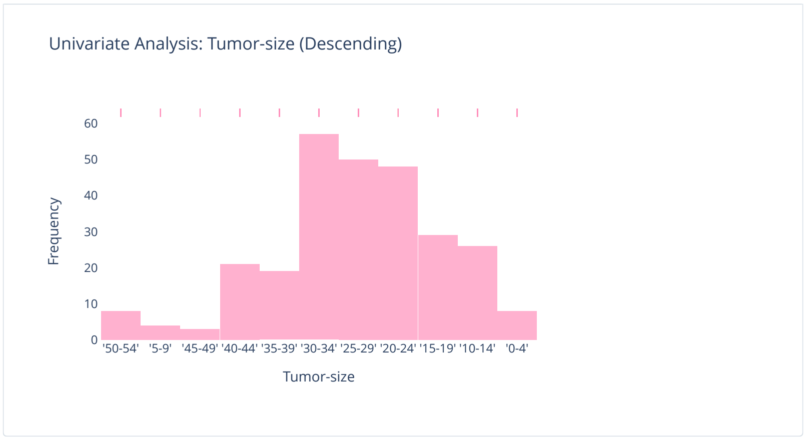


Quantitative variable

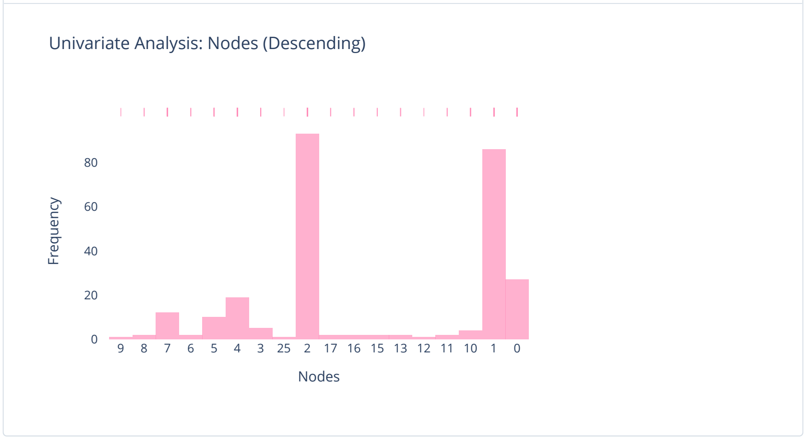
```
import plotly.express as px

# Specify the size of the graph
graph_width = 600
graph_height = 400

# Select the column for analysis
column = 'Age'

# Sort the column in descending order
sorted_df = Last_df.sort_values(by=column, ascending=False)

# Create a histogram for the sorted column
fig = px.histogram(sorted_df, x=column, nbins=20, marginal='rug', color_discrete_sequence=['rgba(255

# Update layout for better visualization and adjust the size
fig.update_layout(title_text=f'Univariate Analysis: {column} (Descending)',
                  xaxis_title=column,
                  yaxis_title='Frequency',
                  paper_bgcolor='white',  # Set background color to white
                  plot_bgcolor='white',  # Set plot background color to white
                  bargap=0.01,  # Adjust the gap between bars
                  width=graph_width,
                  height=graph_height)

# Show the plot
fig.show()
```



```
import plotly.express as px

# Specify the size of the graph
graph_width = 600
graph_height = 400

# Select the column for analysis
column = 'Tumor-size'

# Sort the column in descending order
sorted_df = Last_df.sort_values(by=column, ascending=False)

# Create a histogram for the sorted column
fig = px.histogram(sorted_df, x=column, nbins=20, marginal='rug', color_discrete_sequence=['rgba(255

# Update layout for better visualization and adjust the size
fig.update_layout(title_text=f'Univariate Analysis: {column} (Descending)',
                  xaxis_title=column,
                  yaxis_title='Frequency',
                  paper_bgcolor='white',  # Set background color to white
                  plot_bgcolor='white',  # Set plot background color to white
                  bargap=0.01,  # Adjust the gap between bars
                  width=graph_width,
                  height=graph_height)

# Show the plot
fig.show()
```

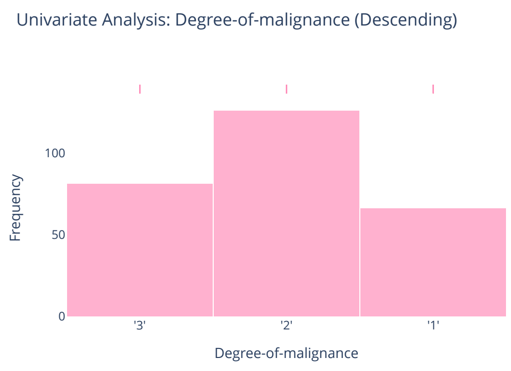


```
import plotly.express as px

# Specify the size of the graph
graph_width = 600
graph_height = 400

# Select the column for analysis
column = 'Nodes'

# Sort the column in descending order
sorted_df = Last_df.sort_values(by=column, ascending=False)

# Create a histogram for the sorted column
fig = px.histogram(sorted_df, x=column, nbins=20, marginal='rug', color_discrete_sequence=['rgba(255

# Update layout for better visualization and adjust the size
fig.update_layout(title_text=f'Univariate Analysis: {column} (Descending)',
                  xaxis_title=column,
                  yaxis_title='Frequency',
                  paper_bgcolor='white',  # Set background color to white
                  plot_bgcolor='white',  # Set plot background color to white
                  bargap=0.01,  # Adjust the gap between bars
                  width=graph_width,
                  height=graph_height)

# Show the plot
fig.show()
```



```
import plotly.express as px

# Specify the size of the graph
graph_width = 600
graph_height = 400

# Select the column for analysis
```

```
column = 'Degree-of-malignance'

# Sort the column in descending order
sorted_df = Last_df.sort_values(by=column, ascending=False)

# Create a histogram for the sorted column
fig = px.histogram(sorted_df, x=column, nbins=20, marginal='rug', color_discrete_sequence=['rgba(255

# Update layout for better visualization and adjust the size
fig.update_layout(title_text=f'Univariate Analysis: {column} (Descending)',
                  xaxis_title=column,
                  yaxis_title='Frequency',
                  paper_bgcolor='white',  # Set background color to white
                  plot_bgcolor='white',  # Set plot background color to white
                  bargap=0.01,  # Adjust the gap between bars
                  width=graph_width,
                  height=graph_height)

# Show the plot
fig.show()
```



Bivariate Analysis

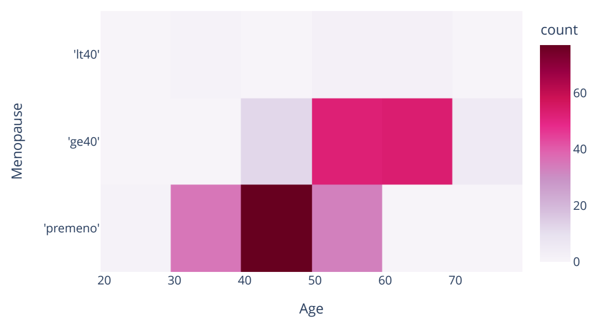
We had to encode Tumer-size to a single definable numbers instead of an interval

```
from sklearn.preprocessing import LabelEncoder
```

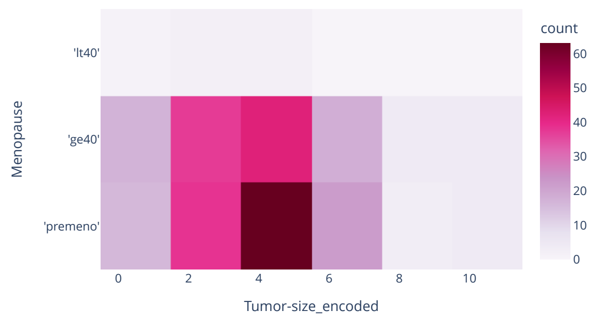
```
# Create a copy to avoid SettingWithCopyWarning
Last_df = Last_df.copy()

# Encode 'Tumor-size' using label encoding
label_encoder = LabelEncoder()
Last_df['Tumor-size_encoded'] = label_encoder.fit_transform(Last_df['Tumor-size'])
```

Qualitative vs Quantitative

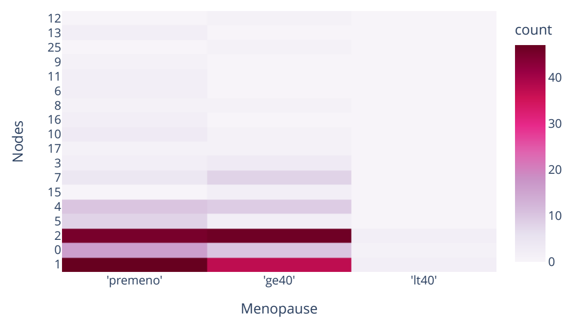
```
import plotly.express as px

# Using "PuRd" color scale with size 600x400
fig_purd = px.density_heatmap(
    Last_df, x="Age", y="Menopause",
    color_continuous_scale="PuRd",
    width=600, height=400
)

# Show the plots
fig_purd.show()
```



```python
import plotly.express as px
from sklearn.preprocessing import import LabelEncoder

# Using "PuRd" color scale with size 600x400
fig_purd = px.density_heatmap(
    Last_df, x="Tumor-size_encoded", y="Menopause",
    color_continuous_scale="PuRd",
    width=600, height=400
)

# Show the plots
fig_purd.show()
```

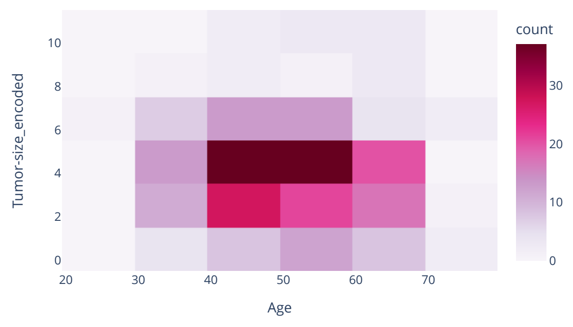


```python
import plotly.express as px

# Using "PuRd" color scale with size 600x400
fig_purd = px.density_heatmap(
    Last_df, x="Menopause", y="Nodes",
    color_continuous_scale="PuRd",
    width=600, height=400
)

# Show the plots
fig_purd.show()
```



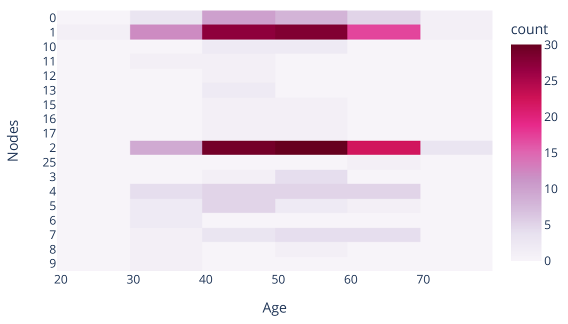
Qualitative vs Quantitative

```python
import plotly.express as px

# Assuming Last_df is your DataFrame
# Replace "Last_df" with the actual name of your DataFrame

# Sort the values of the 'Tumor-size_encoded' column in descending order
Last_df_sorted = Last_df.sort_values(by='Tumor-size_encoded', ascending=False)

# Using "PuRd" color scale with size 600x400
fig_purd = px.density_heatmap(
    Last_df_sorted, x="Age", y="Tumor-size_encoded",
    color_continuous_scale="PuRd",
    width=600, height=400
)

# Show the plot
fig_purd.show()
```

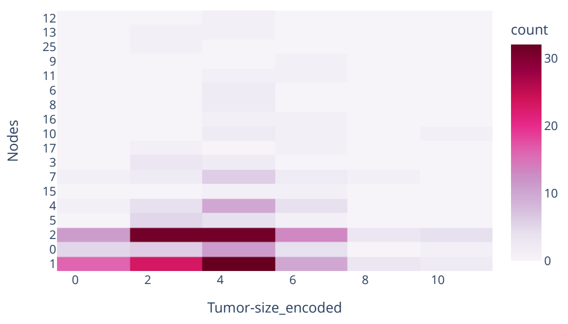


```python
import plotly.express as px

# Sort the values of the 'Tumor-size' column in descending order
Last_df_sorted = Last_df.sort_values(by=['Nodes'], ascending=False, na_position='first')

# Using "PuRd" color scale with size 600x400
fig_purd = px.density_heatmap(
    Last_df_sorted, x="Age", y="Nodes",
    color_continuous_scale="PuRd",
    width=600, height=400
)
```

```
# Show the plots
fig_purd.show()
```



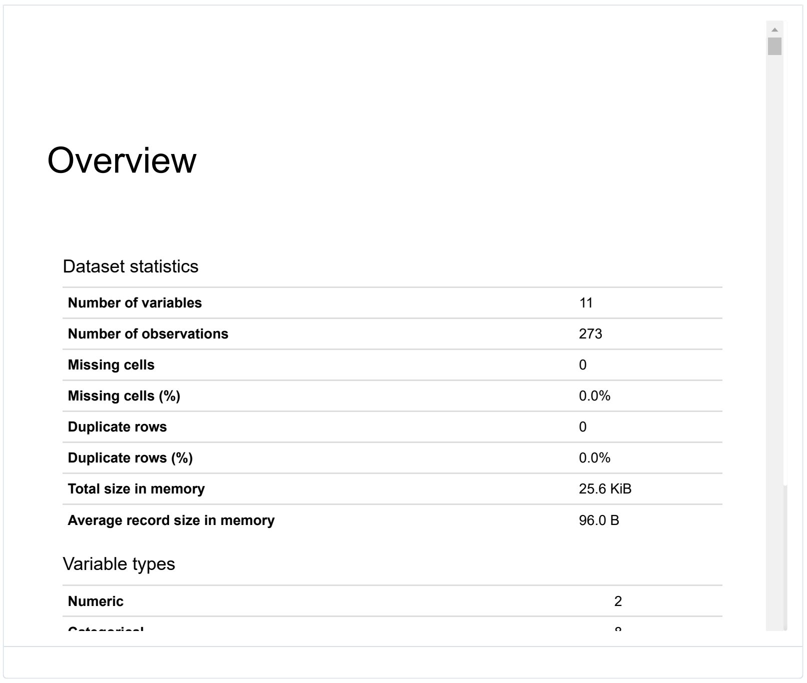
```
import plotly.express as px

# Using "PuRd" color scale with size 600x400
fig_purd = px.density_heatmap(
    Last_df, x="Tumor-size_encoded", y="Nodes",
    color_continuous_scale="PuRd",
    width=600, height=400
)

# Show the plots
fig_purd.show()
```



Validation step

```
from ydata_profiling import ProfileReport

# Generate the data profiling report to varify Data Consistency & Error Checking
report4 = ProfileReport(Last_df, title='Raw Normal Profiling - report4')
report4.to_file("Raw Normal Profiling - report4.html")
```

```
report4
```



Overview

Dataset statistics

Number of variables	11
Number of observations	273
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	25.6 KiB
Average record size in memory	96.0 B

Variable types

Numeric	2

Please find below the full report: https://msc-science-in-computing-2023.github.io/DMV-Reports/report4/