PRINCESS NOURAH UNIVERSITY

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

Masters of Data Science *First Term 2023*

Breast Cancer

Machine Learning

This report is submitted in full fulfillment of the Machine Learning Project

Academic year: 2023/2024

Author:

Sawsan Daban 445009481

Alaa Alsharekh 445009444

Supervisor:

Prof. Dr. Seham Bas. Meshoul

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 *Prof.* Dr. Seham Bas. Meshoul and *Prof.* Dr. Mohamed Batouche 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

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

- Gain a good understanding of the ML methods.
- Select and apply the suitable ML methods to solve the problem.
- Evaluate the different models, interpret results and select the best model.
- Master the use of the related tools.

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.

2.2 Learning Tasks

- Binary Classification - Recurrence Prediction:
  - Task: Predict whether a patient will experience a recurrence of breast cancer (binary outcome: recurrence or no recurrence).
  - Target Variable: Recurrence column.
- Multiclass Classification - Cancer Severity Prediction:
  - Task: Predict the severity level of breast cancer based on the degree-of-malignance.
  - Target Variable: degree-of-malignance (assuming it has multiple classes).
- Regression - Age Prediction:
  - Task: Predict the age of a patient based on other available features.
  - Target Variable: Age.
- Categorical Classification - Breast Type Prediction:
  - Task: Predict the type of breast involved (left or right).
  - Target Variable: Breast column.
- Categorical Classification - Menopause Prediction:
  - Task: Predict whether a patient is in menopause or not.
  - Target Variable: Menopause column.
- Clustering - Patient Segmentation:
  - Task: Cluster patients based on their features to identify subgroups with similar characteristics.
- Association Rule Mining - Patterns in Treatment:
  - Task: Identify patterns or associations between different features and the type of treatment received (Irradiation).

2.3 Machine Learning Models

For predicting disease recurrence (a binary classification task), several machine learning models can be applied. The choice of model often depends on various factors like the size of the dataset, the complexity of relationships, interpretability, and computational efficiency. Here are some suitable ML models for predicting disease recurrence:

- Logistic Regression:
  - Suitable for binary classification tasks.
  - Interpretable and provides probabilities.
  - Works well with linearly separable data.
- Decision Trees and Random Forests:
  - Effective for classification tasks.
  - Can handle nonlinear relationships and interactions between features.
  - Random Forests reduce overfitting and increase accuracy by combining multiple decision trees.
- Support Vector Machines (SVM):
  - Effective in high-dimensional spaces.
  - Works well with both linear and nonlinear data.
  - Finds the best separation boundary (hyperplane) between classes.
- Neural Networks:
  - Deep learning models suitable for complex, nonlinear relationships.
  - Requires more data and computational power but can capture intricate patterns.
- Naive Bayes:
  - Simple and efficient for binary classification.
  - Assumes independence between features (which might not hold true in all cases).
- K-Nearest Neighbors (KNN):

- Non-parametric and instance-based method.
- Predicts based on the majority class among its nearest neighbors.

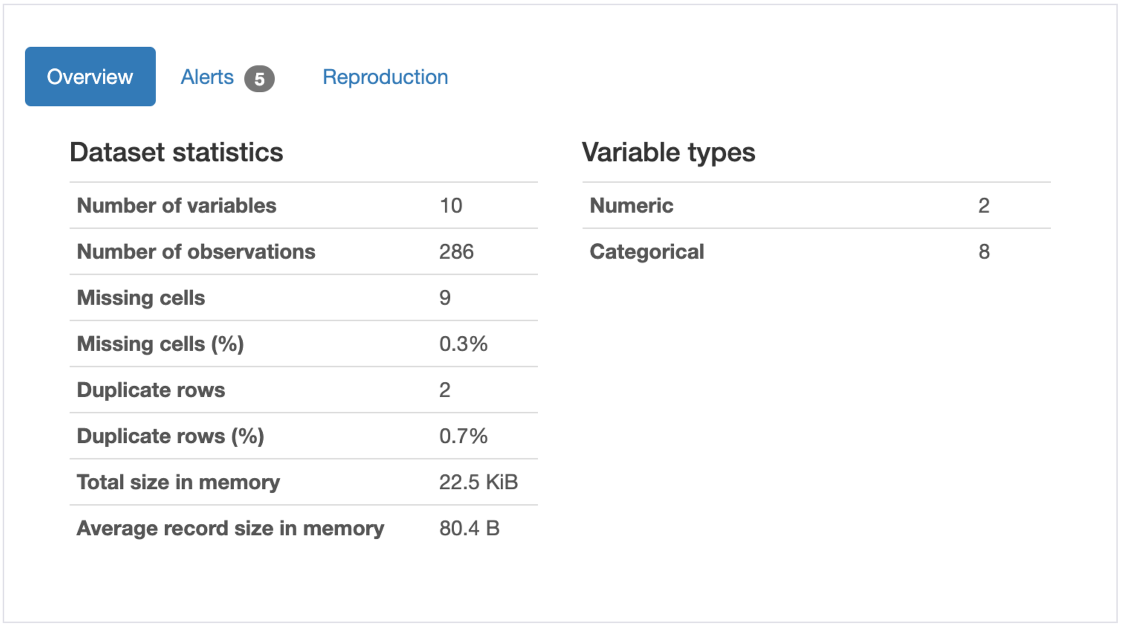
When choosing a model, considerations include the dataset size, feature importance, interpretability, computational resources, and the trade-off between accuracy and model complexity. Additionally, techniques such as cross-validation, hyperparameter tuning, and feature selection can enhance model performance.

2.4 Preparing a Dataset for Machine Learning

2.4.1 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 286 observations. Among these variables, 8 fall under the categorical category, while 2 are designated as a numerical variables .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

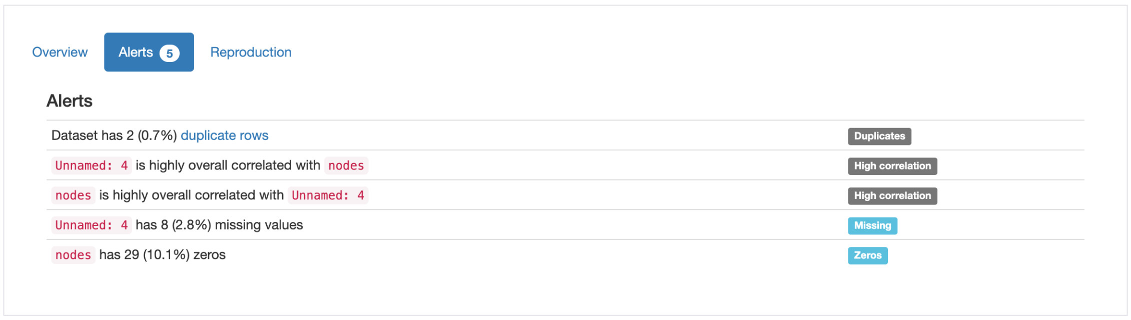
Number of variables	10
Number of observations	286
Missing cells	9
Missing cells (%)	0.3%
Duplicate rows	2
Duplicate rows (%)	0.7%
Total size in memory	22.5 KiB
Average record size in memory	80.4 B

Variable types

Numeric	2
Categorical	8

Figure 2.1: Quality report for row data

[1]For more information, please check YData Profiling



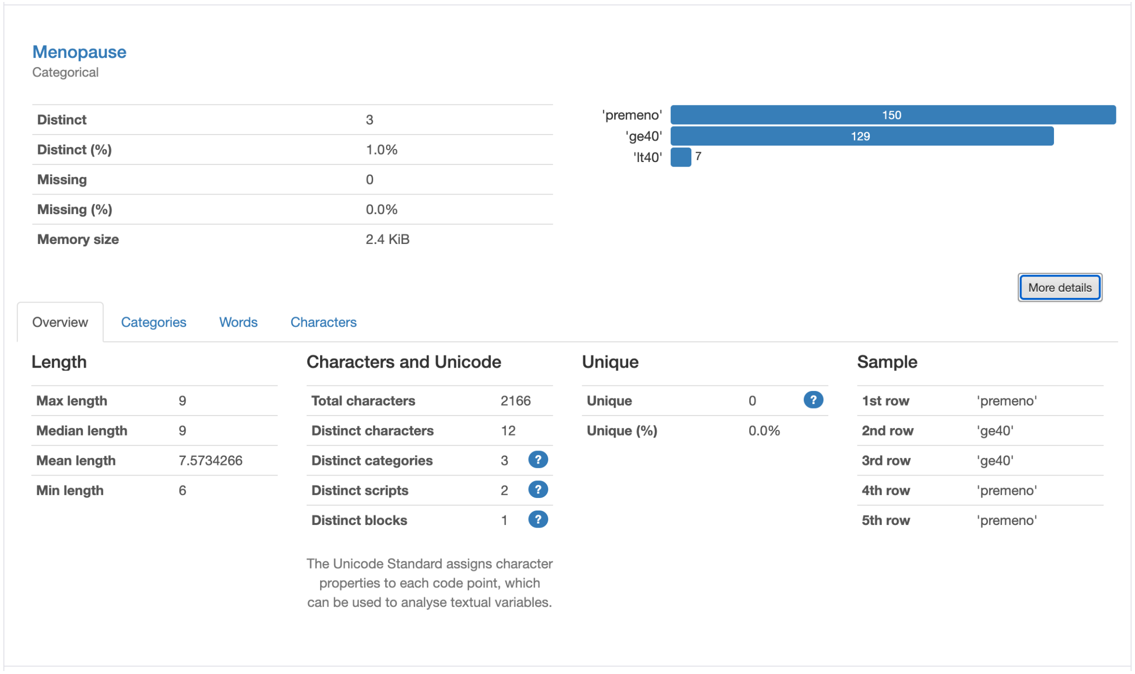
Alerts

Dataset has 2 (0.7%) duplicate rows	Duplicates
`Unnamed: 4` is highly overall correlated with `nodes`	High correlation
`nodes` is highly overall correlated with `Unnamed: 4`	High correlation
`Unnamed: 4` has 8 (2.8%) missing values	Missing
`nodes` has 29 (10.1%) zeros	Zeros

Figure 2.2: Quality report showing current data set issues

- *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	3
Distinct (%)	1.0%
Missing	0
Missing (%)	0.0%
Memory size	2.4 KiB

Length

Max length	9
Median length	9
Mean length	7.5734266
Min length	6

Characters and Unicode

Total characters	2166
Distinct characters	12
Distinct categories	3
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	0
Unique (%)	0.0%

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.

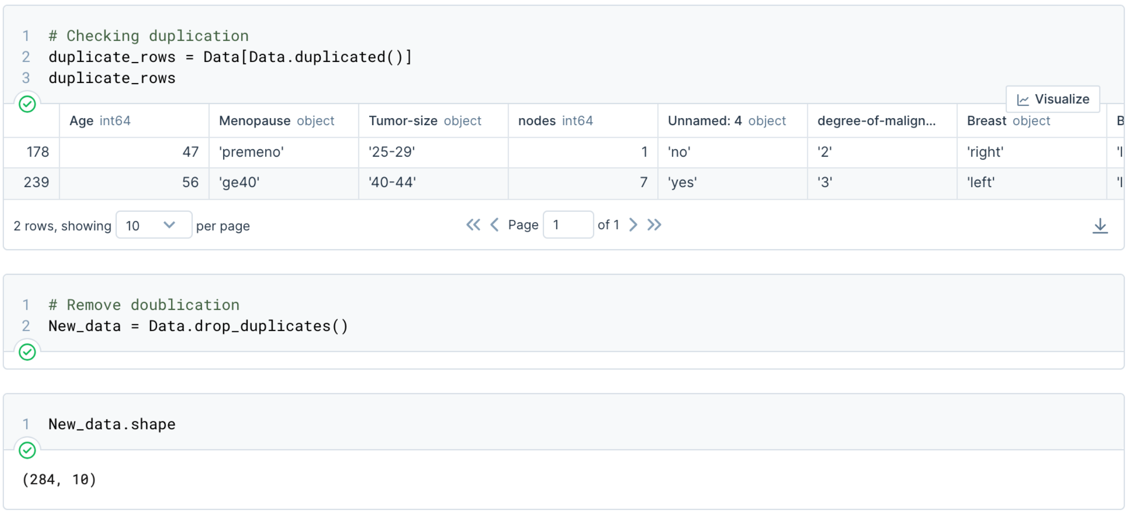
2.5 Apply Quality Controls

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 cleaning: handling duplication
2. Data cleaning: missing column name
3. Data cleaning: unified missing values
4. Data cleaning: Checking outlier
5. Data cleaning: Removing None values
6. Data cleaning: encoding categorical variables
7. Data cleaning: imbalanced data

2.5.1 Data cleaning: handling duplication

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.



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

	Age int64	Menopause object	Tumor-size object	nodes int64	Unnamed: 4 object	degree-of-malign...	Breast object	B
178	47	'premeno'	'25-29'	1	'no'	'2'	'right'	'l
239	56	'ge40'	'40-44'	7	'yes'	'3'	'left'	'l

2 rows, showing 10 per page Page 1 of 1

```python
# Remove doublication
New_data = Data.drop_duplicates()
```

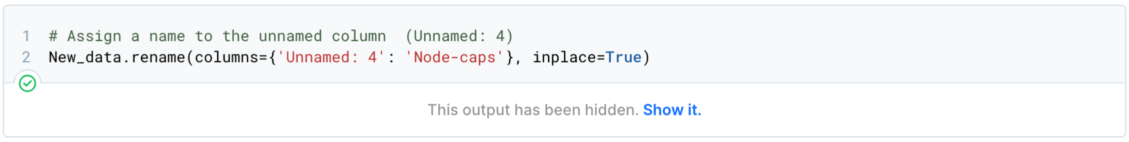
```python
New_data.shape
```

```
(284, 10)
```

Figure 2.4: Handling duplication

2.5.2 Data cleaning: missing column name

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.

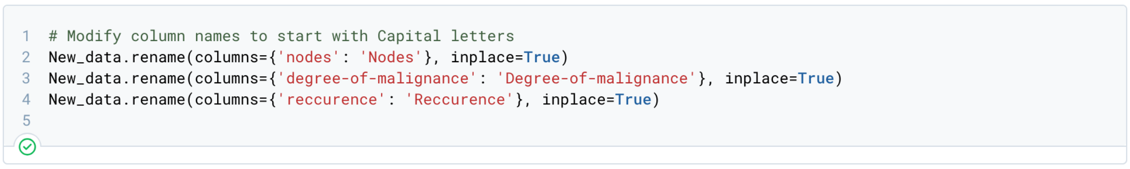


```python
# Assign a name to the unnamed column  (Unnamed: 4)
New_data.rename(columns={'Unnamed: 4': 'Node-caps'}, inplace=True)
```

This output has been hidden. Show it.

Figure 2.5: Assign a name to the unnamed column

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



```python
# Modify column names to start with Capital letters
New_data.rename(columns={'nodes': 'Nodes'}, inplace=True)
New_data.rename(columns={'degree-of-malignance': 'Degree-of-malignance'}, inplace=True)
New_data.rename(columns={'reccurence': 'Reccurence'}, inplace=True)

```

Figure 2.6: Modify column names to start with Capital letters

2.5.3 Data cleaning: unified missing values

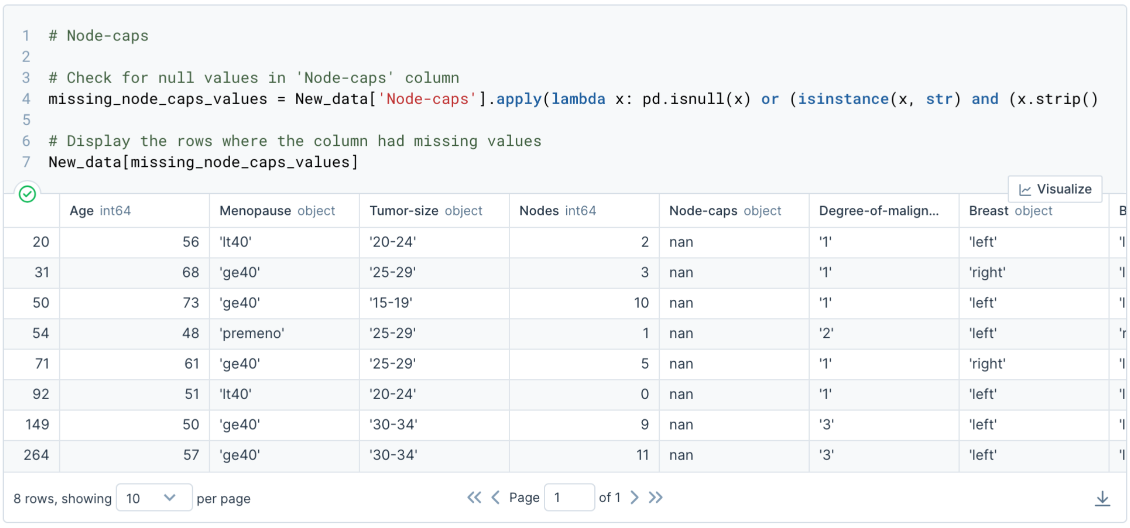
One way for addressing null values involves examining various formats for representing null and then standardizing and replacing them with the term None.[2]

Some techniques to handle Null values:

1. Removing Rows with Null Values.

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

Based on the report, there are a total of 8 missing values in Node-caps out of 284 observations which represent only 0.3%.

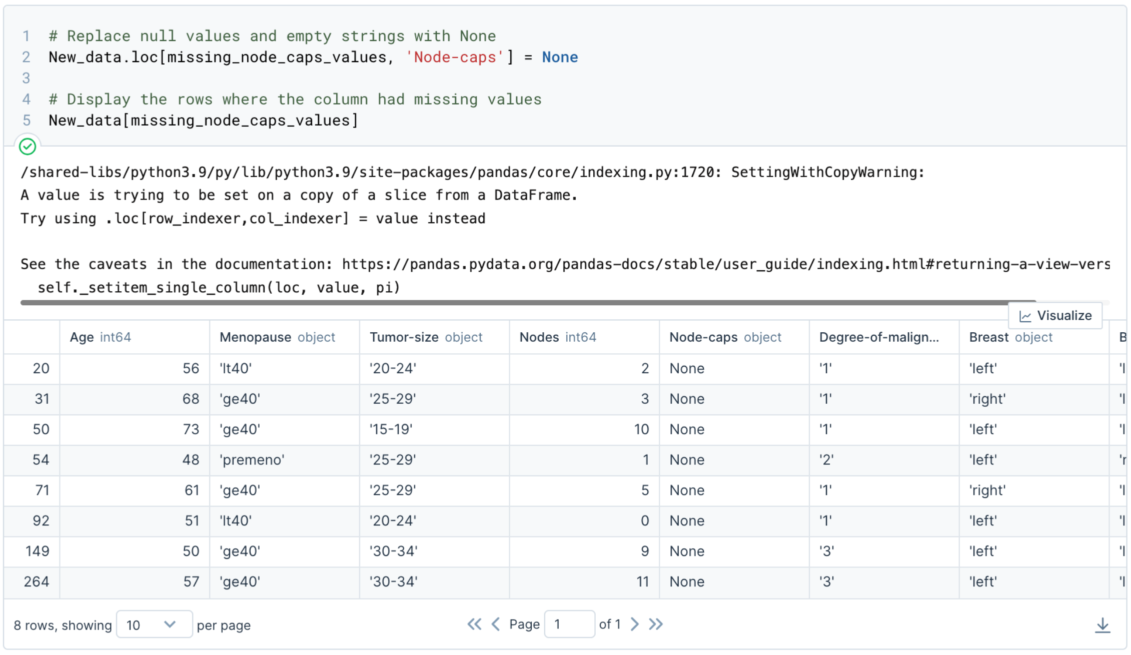


```
# Node-caps

# Check for null values in 'Node-caps' column
missing_node_caps_values = New_data['Node-caps'].apply(lambda x: pd.isnull(x) or (isinstance(x, str) and (x.strip()

# Display the rows where the column had missing values
New_data[missing_node_caps_values]
```

	Age int64	Menopause object	Tumor-size object	Nodes int64	Node-caps object	Degree-of-malign...	Breast object	B
20	56	'lt40'	'20-24'	2	nan	'1'	'left'	'l
31	68	'ge40'	'25-29'	3	nan	'1'	'right'	'l
50	73	'ge40'	'15-19'	10	nan	'1'	'left'	'l
54	48	'premeno'	'25-29'	1	nan	'2'	'left'	'r
71	61	'ge40'	'25-29'	5	nan	'1'	'right'	'l
92	51	'lt40'	'20-24'	0	nan	'1'	'left'	'l
149	50	'ge40'	'30-34'	9	nan	'3'	'left'	'l
264	57	'ge40'	'30-34'	11	nan	'3'	'left'	'l

Figure 2.7: Identify null values within features

[2]Full report added in the appendices



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

# Display the rows where the column had missing values
New_data[missing_node_caps_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)
```

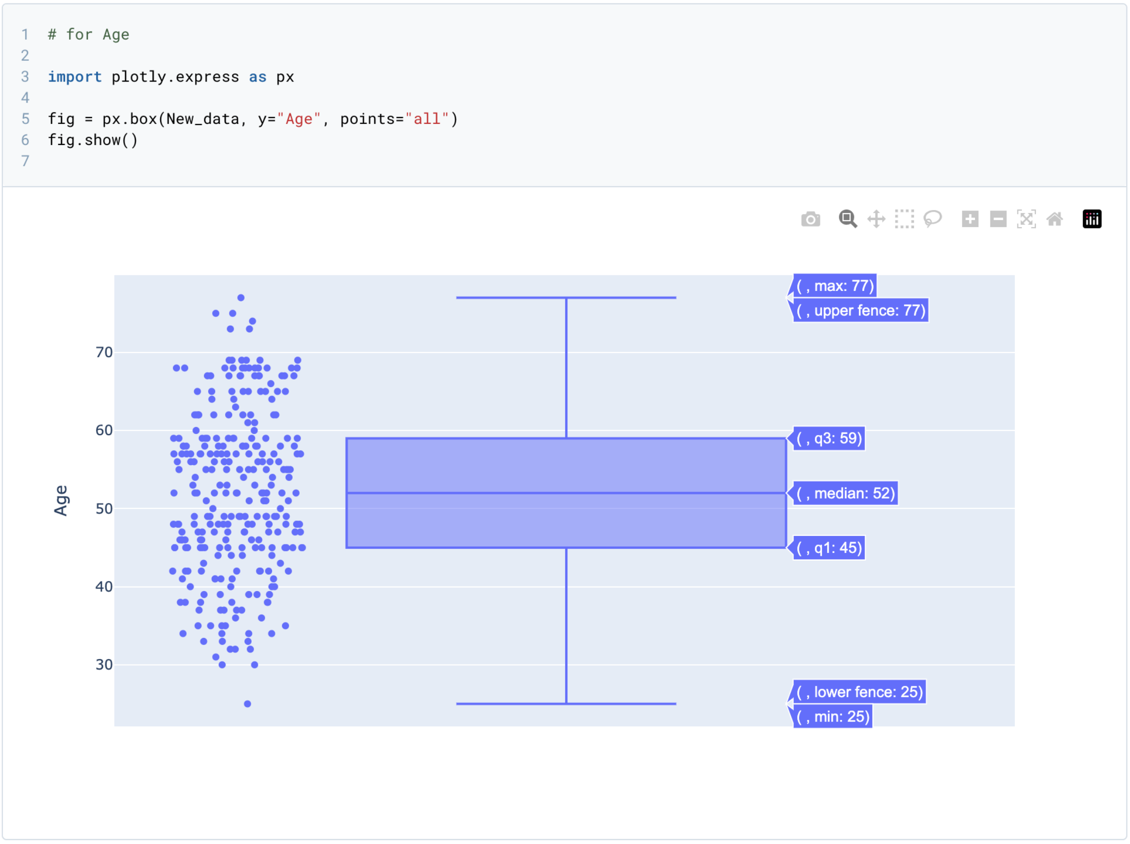
	Age int64	Menopause object	Tumor-size object	Nodes int64	Node-caps object	Degree-of-malign...	Breast object	B
20	56	'lt40'	'20-24'	2	None	'1'	'left'	'l
31	68	'ge40'	'25-29'	3	None	'1'	'right'	'l
50	73	'ge40'	'15-19'	10	None	'1'	'left'	'l
54	48	'premeno'	'25-29'	1	None	'2'	'left'	'r
71	61	'ge40'	'25-29'	5	None	'1'	'right'	'l
92	51	'lt40'	'20-24'	0	None	'1'	'left'	'l
149	50	'ge40'	'30-34'	9	None	'3'	'left'	'l
264	57	'ge40'	'30-34'	11	None	'3'	'left'	'l

8 rows, showing 10 per page · Page 1 of 1

Figure 2.8: Handle null values within features

2.5.4 Data cleaning: checking outlier

The box plot indicates that the highest age is 77, while the lowest age is 25, signifying the absence of any outliers.

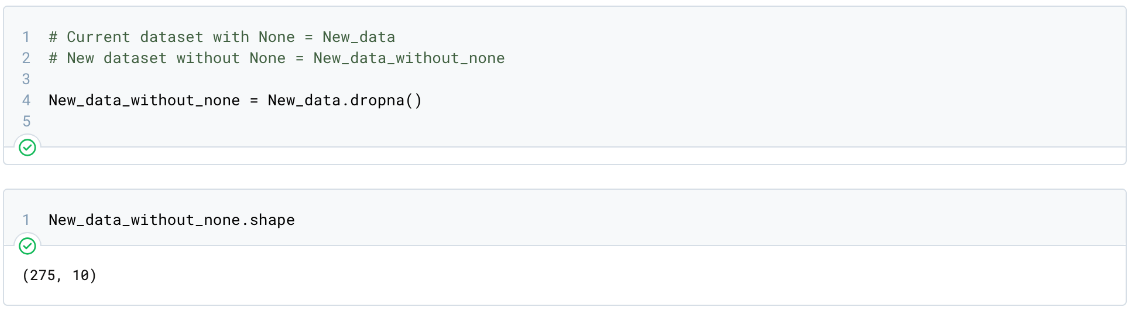


```python
# for Age

import plotly.express as px

fig = px.box(New_data, y="Age", points="all")
fig.show()

```

Figure 2.9: Checking outliers

2.5.5 Data cleaning: removing None values

Based on the report outcomes following the implementation of the necessary checks, it appears that additional missing values have surfaced.



```
# Current dataset with None = New_data
# New dataset without None = New_data_without_none

New_data_without_none = New_data.dropna()

```

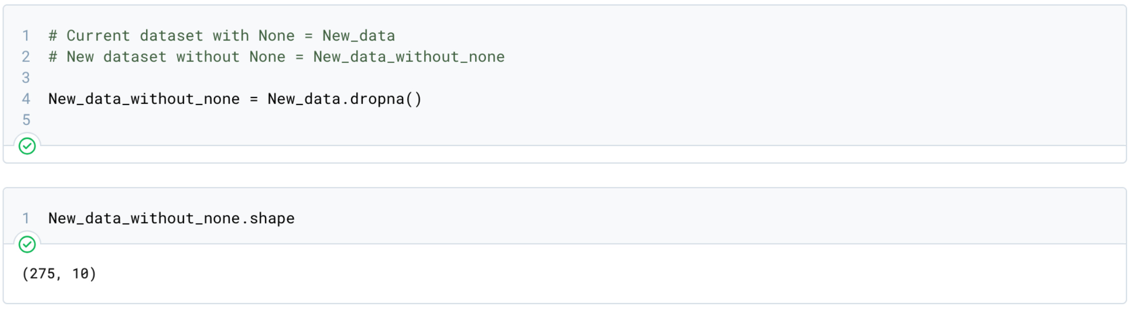
```
New_data_without_none.shape
```

```
(275, 10)
```

Figure 2.10: Remove null values

2.5.6 Data cleaning: encoding categorical variables

Encoding converts categorical variables into numerical representations, allowing the algorithms to process and learn from the data.



```
# Current dataset with None = New_data
# New dataset without None = New_data_without_none

New_data_without_none = New_data.dropna()

```

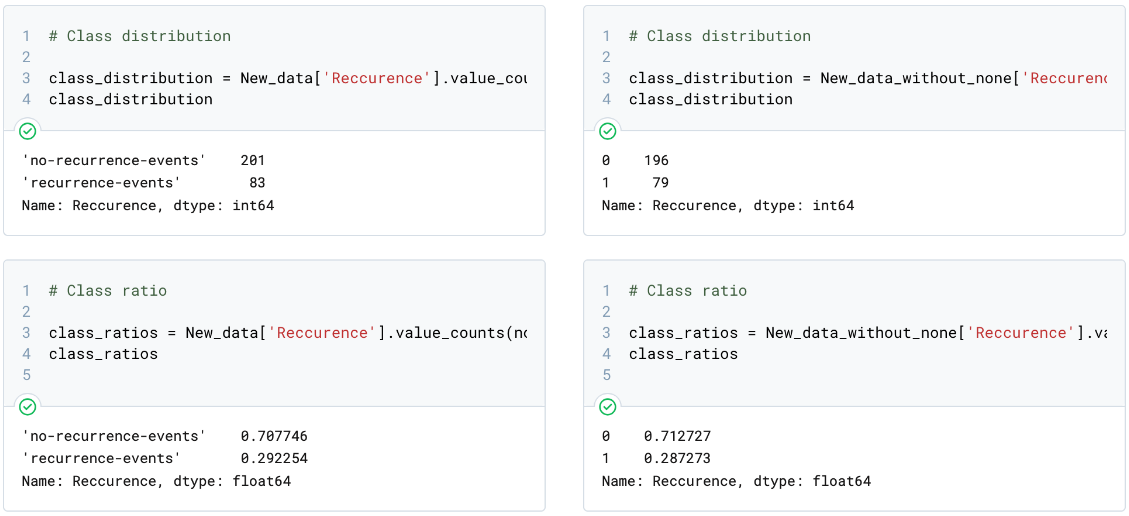
```
New_data_without_none.shape
```

```
(275, 10)
```

Figure 2.11: Encoding categorical variables

2.5.7 Data cleaning: imbalanced data

Several techniques can be employed to assess whether the target variable is imbalanced or not.



```
# Class distribution

class_distribution = New_data['Reccurence'].value_cou
class_distribution
```

```
'no-recurrence-events'    201
'recurrence-events'        83
Name: Reccurence, dtype: int64
```

```
# Class distribution

class_distribution = New_data_without_none['Reccurenc
class_distribution
```

```
0    196
1     79
Name: Reccurence, dtype: int64
```

```
# Class ratio

class_ratios = New_data['Reccurence'].value_counts(nc
class_ratios

```

```
'no-recurrence-events'    0.707746
'recurrence-events'       0.292254
Name: Reccurence, dtype: float64
```

```
# Class ratio

class_ratios = New_data_without_none['Reccurence'].va
class_ratios

```

```
0    0.712727
1    0.287273
Name: Reccurence, dtype: float64
```

Figure 2.12: Handling imbalanced data techniques

Machine Leaning Models

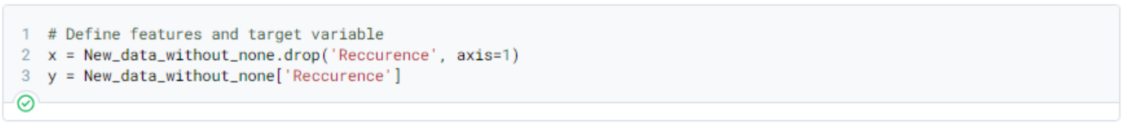
3.1 Naïve Bayes Classifier

Naïve Bayes is a probabilistic classifier based on Bayes' theorem. It assumes independence between features given the class.

- Implementation: Train the Naïve Bayes model using the dataset.
- Evaluation: Assess model performance using accuracy, precision, recall, and F1-score for both hold-out sampling and K-fold cross-validation.

3.1.1 Data Splitting

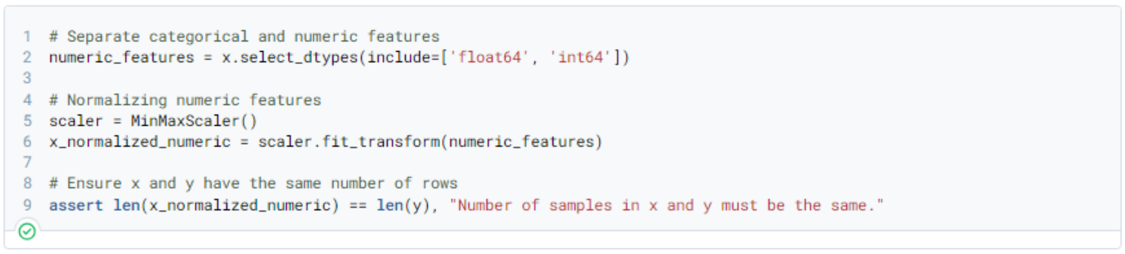
Train-Test Split: Divide the dataset into training and testing sets to train the model on one subset and validate its performance on another.



```
# Define features and target variable
x = New_data_without_none.drop('Reccurence', axis=1)
y = New_data_without_none['Reccurence']
```

Figure 3.1: Define Features and Target Variable

Before we split the data, we normalize and scale the numeric variables as shown in figure 3.3.

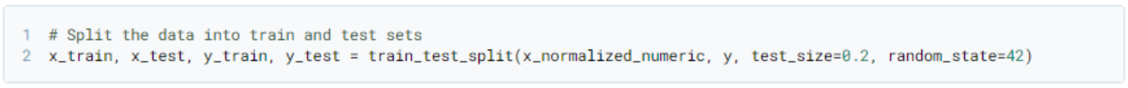


```
# Separate categorical and numeric features
numeric_features = x.select_dtypes(include=['float64', 'int64'])

# Normalizing numeric features
scaler = MinMaxScaler()
x_normalized_numeric = scaler.fit_transform(numeric_features)

# Ensure x and y have the same number of rows
assert len(x_normalized_numeric) == len(y), "Number of samples in x and y must be the same."
```

Figure 3.2: Split The Data Into Train and Test Sets

3.1.2 Feature Scaling/Normalization

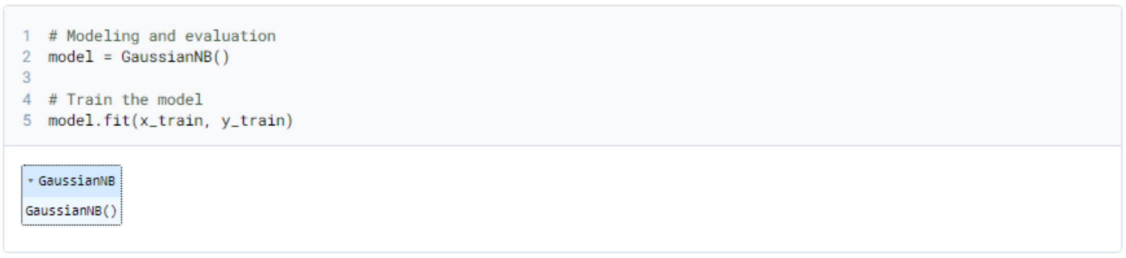
Scale Numerical Features: Standardize or normalize numerical features to a similar scale to avoid bias in models that rely on distance measures.



```
# Split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x_normalized_numeric, y, test_size=0.2, random_state=42)
```

Figure 3.3: Normalize and Scale The Data

3.1.3 Initialize the model



```
# Modeling and evaluation
model = GaussianNB()

# Train the model
model.fit(x_train, y_train)
```

```
▸ GaussianNB
GaussianNB()
```

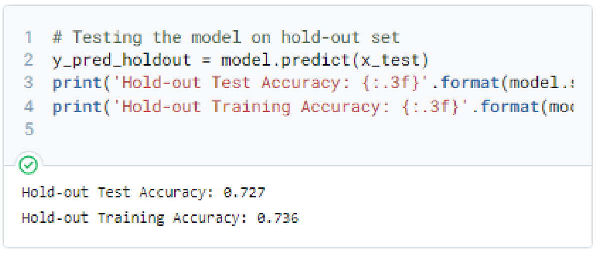
Figure 3.4: Initialize Naïve Bayes Classifier

3.1.4 Cross-Validation

Use cross-validation techniques to assess model performance robustly and avoid overfitting.

Hold-out Sampling

Split the dataset into training and testing sets, train the models on the training set, and evaluate their performance on the unseen testing set.



```
# Testing the model on hold-out set
y_pred_holdout = model.predict(x_test)
print('Hold-out Test Accuracy: {:.3f}'.format(model.
print('Hold-out Training Accuracy: {:.3f}'.format(mo

```

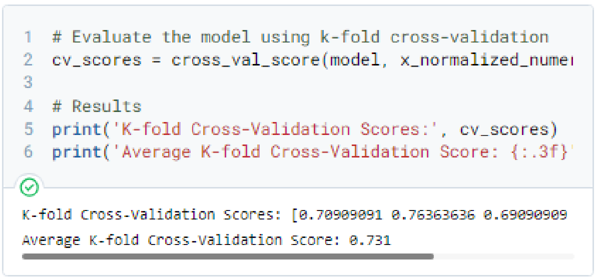
```
Hold-out Test Accuracy: 0.727
Hold-out Training Accuracy: 0.736
```

Figure 3.5: Evaluate Using Hold-Out Sampling

This is an indicator of the accuracy of the model on the test set that was not seen during training. In this case, our model correctly predicted the target variable for approximately 72.7% of the samples. In addition to how well the model performs on the training data. The model achieved an accuracy of approximately 73.6% on the training set.

K-fold Cross-Validation

Divide the dataset into k subsets (folds), train the models on k-1 folds, and validate on the remaining fold. This process is repeated k times, and the performance is averaged.



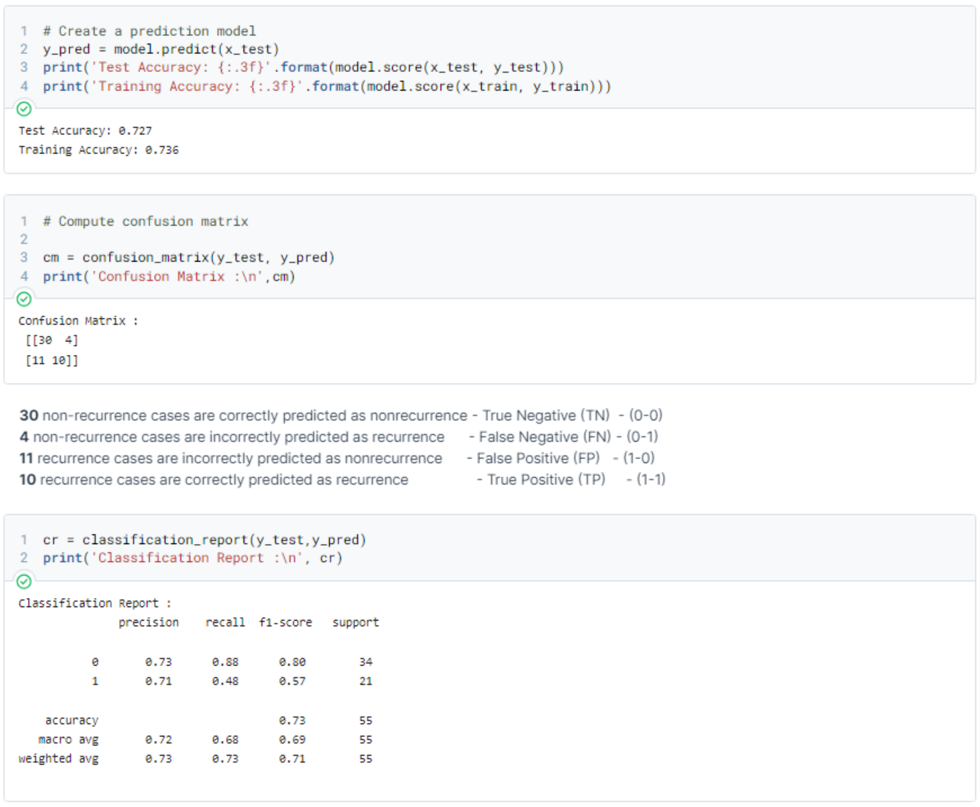
```
# Evaluate the model using k-fold cross-validation
cv_scores = cross_val_score(model, x_normalized_nume

# Results
print('K-fold Cross-Validation Scores:', cv_scores)
print('Average K-fold Cross-Validation Score: {:.3f}'
```

```
K-fold Cross-Validation Scores: [0.70909091 0.76363636 0.69090909
Average K-fold Cross-Validation Score: 0.731
```

Figure 3.6: Evaluate Using K-Fold Cross-Validation

The k-fold cross-validation scores represent the accuracy of your model across different folds (splits) of the dataset. The scores for each fold are as follows: [0.709, 0.764, 0.691, 0.800, 0.691]. The average k-fold cross-validation score is calculated as 0.731, representing the overall model performance across the different folds.

3.1.5 Testing The Model



```
# Create a prediction model
y_pred = model.predict(x_test)
print('Test Accuracy: {:.3f}'.format(model.score(x_test, y_test)))
print('Training Accuracy: {:.3f}'.format(model.score(x_train, y_train)))
```

```
Test Accuracy: 0.727
Training Accuracy: 0.736
```

```
# Compute confusion matrix

cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[30  4]
 [11 10]]
```

30 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0)
4 non-recurrence cases are incorrectly predicted as recurrence - False Negative (FN) - (0-1)
11 recurrence cases are incorrectly predicted as nonrecurrence - False Positive (FP) - (1-0)
10 recurrence cases are correctly predicted as recurrence - True Positive (TP) - (1-1)

```
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

```
Classification Report :
               precision    recall  f1-score   support

           0       0.73      0.88      0.80        34
           1       0.71      0.48      0.57        21

    accuracy                           0.73        55
   macro avg       0.72      0.68      0.69        55
weighted avg       0.73      0.73      0.71        55
```

Figure 3.7: Create A Prediction Model

3.2 K-Nearest Neighbors (KNN)

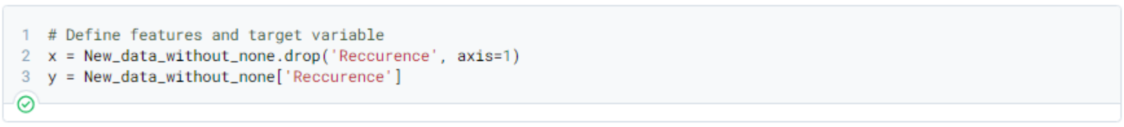
KNN is an instance-based learning algorithm that classifies new instances based on the majority class of its k-nearest neighbors.

- Implementation: Train the KNN model using different values of k and evaluate.
- Evaluation: Measure accuracy, precision, recall, and F1-score using hold-out sampling and K-fold cross-validation.

3.2.1 KNN Classification

3.2.2 Data Splitting

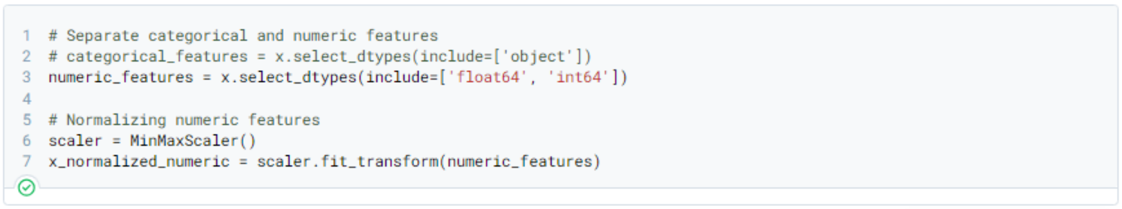
Train-Test Split: Divide the dataset into training and testing sets to train the model on one subset and validate its performance on another.



```
# Define features and target variable
x = New_data_without_none.drop('Reccurence', axis=1)
y = New_data_without_none['Reccurence']
```

Figure 3.8: Define Features and Target Variable

Before we split the data, we normalize and scale the numeric variables as shown in figure 3.34.

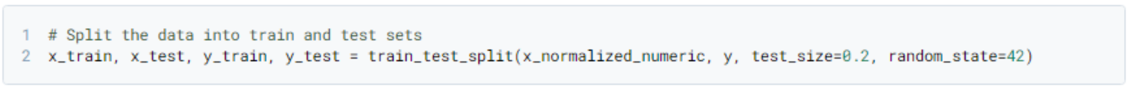


```
# Separate categorical and numeric features
# categorical_features = x.select_dtypes(include=['object'])
numeric_features = x.select_dtypes(include=['float64', 'int64'])

# Normalizing numeric features
scaler = MinMaxScaler()
x_normalized_numeric = scaler.fit_transform(numeric_features)
```

Figure 3.9: Split The Data Into Train and Test Sets

3.2.3 Feature Scaling/Normalization

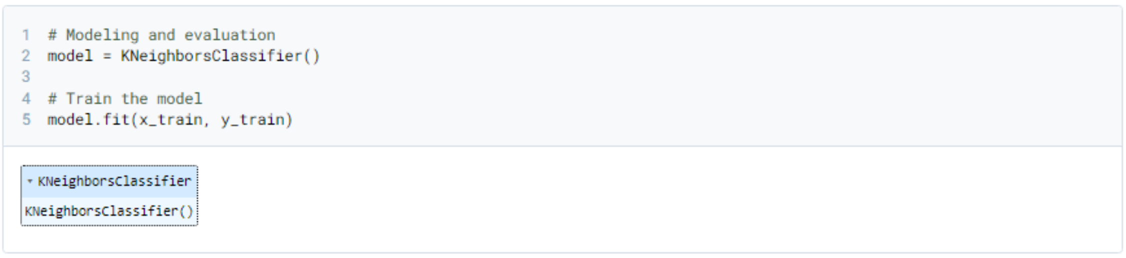
Scale Numerical Features: Standardize or normalize numerical features to a similar scale to avoid bias in models that rely on distance measures.



```
# Split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x_normalized_numeric, y, test_size=0.2, random_state=42)
```

Figure 3.10: Normalize and Scale The Data

3.2.4 Initialize the model



```
# Modeling and evaluation
model = KNeighborsClassifier()

# Train the model
model.fit(x_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

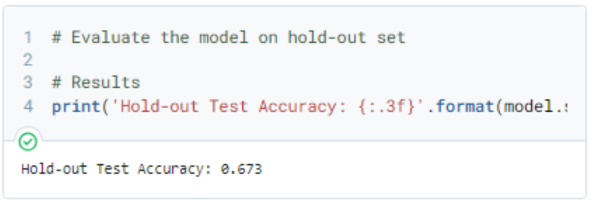
Figure 3.11: Initialize K-Nearest Neighbors Classifier

3.2.5 Cross-Validation

Use cross-validation techniques to assess model performance robustly and avoid overfitting.

Hold-out Sampling

Split the dataset into training and testing sets, train the models on the training set, and evaluate their performance on the unseen testing set.



```
# Evaluate the model on hold-out set

# Results
print('Hold-out Test Accuracy: {:.3f}'.format(model.
```

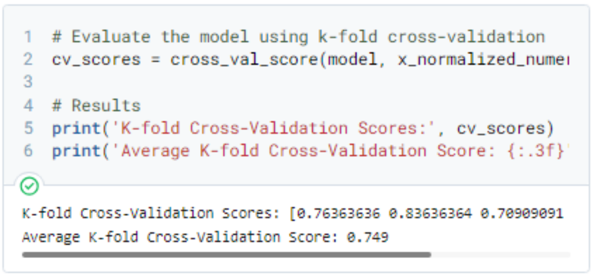
```
Hold-out Test Accuracy: 0.673
```

Figure 3.12: Evaluate Using Hold-Out Sampling

This metric indicates the accuracy of the KNN model on the test set that was not seen during training. In this case, the model correctly predicted the target variable for approximately 67.3% of the samples.

K-fold Cross-Validation

Divide the dataset into k subsets (folds), train the models on k-1 folds, and validate on the remaining fold. This process is repeated k times, and the performance is averaged.



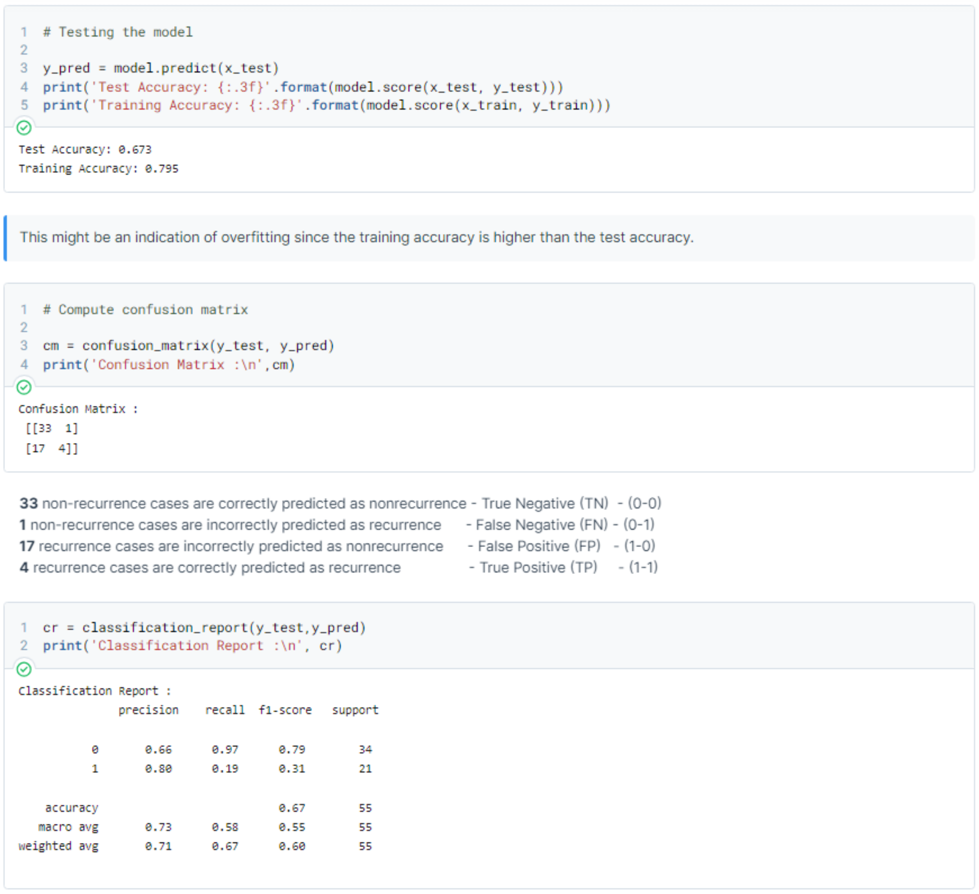
```
# Evaluate the model using k-fold cross-validation
cv_scores = cross_val_score(model, x_normalized_nume

# Results
print('K-fold Cross-Validation Scores:', cv_scores)
print('Average K-fold Cross-Validation Score: {:.3f}
```

```
K-fold Cross-Validation Scores: [0.76363636 0.83636364 0.70909091
Average K-fold Cross-Validation Score: 0.749
```

Figure 3.13: Evaluate Using K-Fold Cross-Validation

The k-fold cross-validation scores represent the accuracy of the model across different folds (splits) of the dataset. The scores for each fold are as follows: [0.764, 0.836, 0.709, 0.727, 0.709]. The average k-fold cross-validation score is calculated as 0.749, representing the overall model performance across the different folds.

3.2.6 Testing The Model



```python
# Testing the model

y_pred = model.predict(x_test)
print('Test Accuracy: {:.3f}'.format(model.score(x_test, y_test)))
print('Training Accuracy: {:.3f}'.format(model.score(x_train, y_train)))
```

```
Test Accuracy: 0.673
Training Accuracy: 0.795
```

This might be an indication of overfitting since the training accuracy is higher than the test accuracy.

```python
# Compute confusion matrix

cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[33  1]
 [17  4]]
```

33 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0)
1 non-recurrence cases are incorrectly predicted as recurrence - False Negative (FN) - (0-1)
17 recurrence cases are incorrectly predicted as nonrecurrence - False Positive (FP) - (1-0)
4 recurrence cases are correctly predicted as recurrence - True Positive (TP) - (1-1)

```python
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

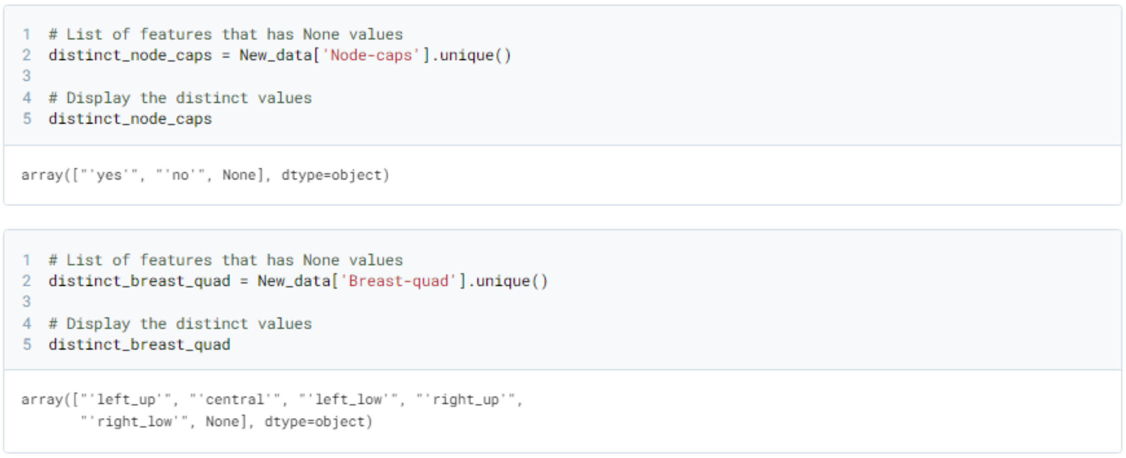
```
Classification Report :
               precision    recall  f1-score   support

           0       0.66      0.97      0.79        34
           1       0.80      0.19      0.31        21

    accuracy                           0.67        55
   macro avg       0.73      0.58      0.55        55
weighted avg       0.71      0.67      0.60        55
```

Figure 3.14: Create A Prediction Model

3.2.7 KNN Imputation

An effective approach to data imputing is to use a model to predict the missing values.

3.2.8 None Value Features



```python
# List of features that has None values
distinct_node_caps = New_data['Node-caps'].unique()

# Display the distinct values
distinct_node_caps
```

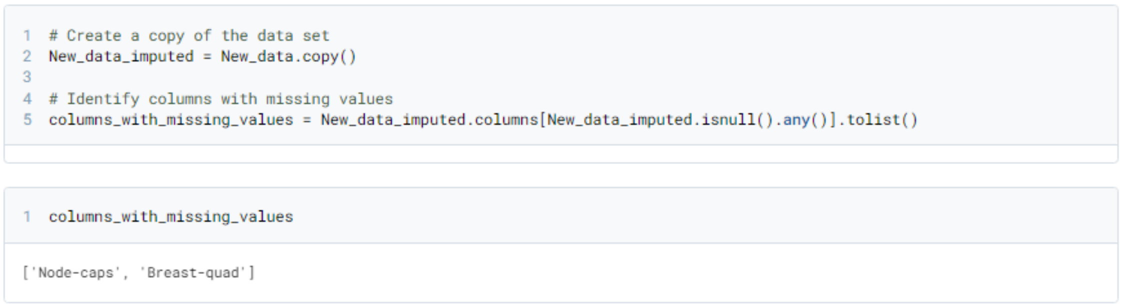
```
array(["'yes'", "'no'", None], dtype=object)
```

```python
# List of features that has None values
distinct_breast_quad = New_data['Breast-quad'].unique()

# Display the distinct values
distinct_breast_quad
```

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

Figure 3.15: Listing The Features That Have None Values (Node-caps and Breast-quad)

3.2.9 New Dataset



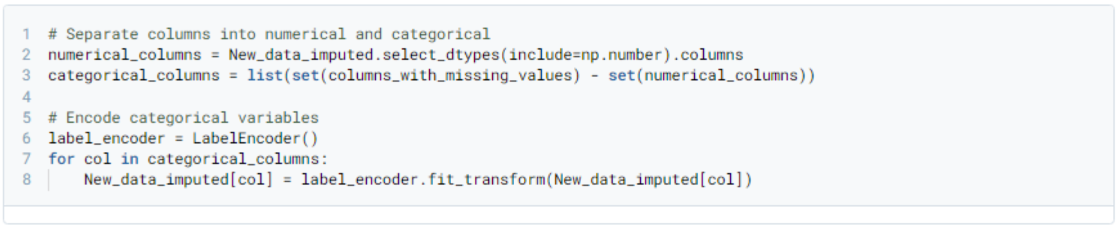
```python
# Create a copy of the data set
New_data_imputed = New_data.copy()

# Identify columns with missing values
columns_with_missing_values = New_data_imputed.columns[New_data_imputed.isnull().any()].tolist()
```

```python
columns_with_missing_values
```

```
['Node-caps', 'Breast-quad']
```

Figure 3.16: Create A Copy Of The Dataset

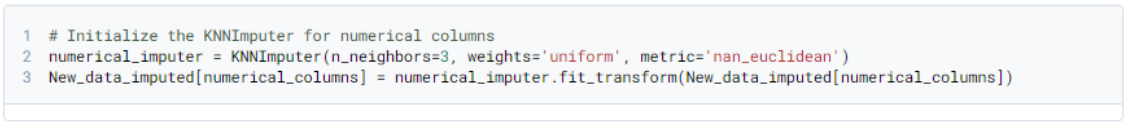
3.2.10 Define The Imputer



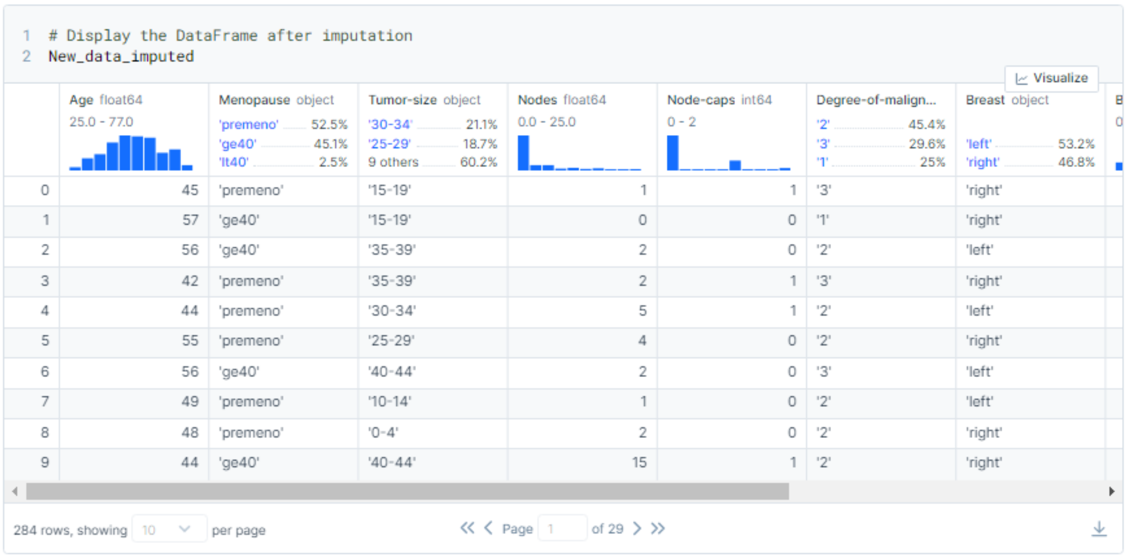
```python
# Separate columns into numerical and categorical
numerical_columns = New_data_imputed.select_dtypes(include=np.number).columns
categorical_columns = list(set(columns_with_missing_values) - set(numerical_columns))

# Encode categorical variables
label_encoder = LabelEncoder()
for col in categorical_columns:
    New_data_imputed[col] = label_encoder.fit_transform(New_data_imputed[col])
```

Figure 3.17: Define The Imputer and Fit To The Dataset

3.2.11 Initialize The Imputer



```python
# Initialize the KNNImputer for numerical columns
numerical_imputer = KNNImputer(n_neighbors=3, weights='uniform', metric='nan_euclidean')
New_data_imputed[numerical_columns] = numerical_imputer.fit_transform(New_data_imputed[numerical_columns])
```



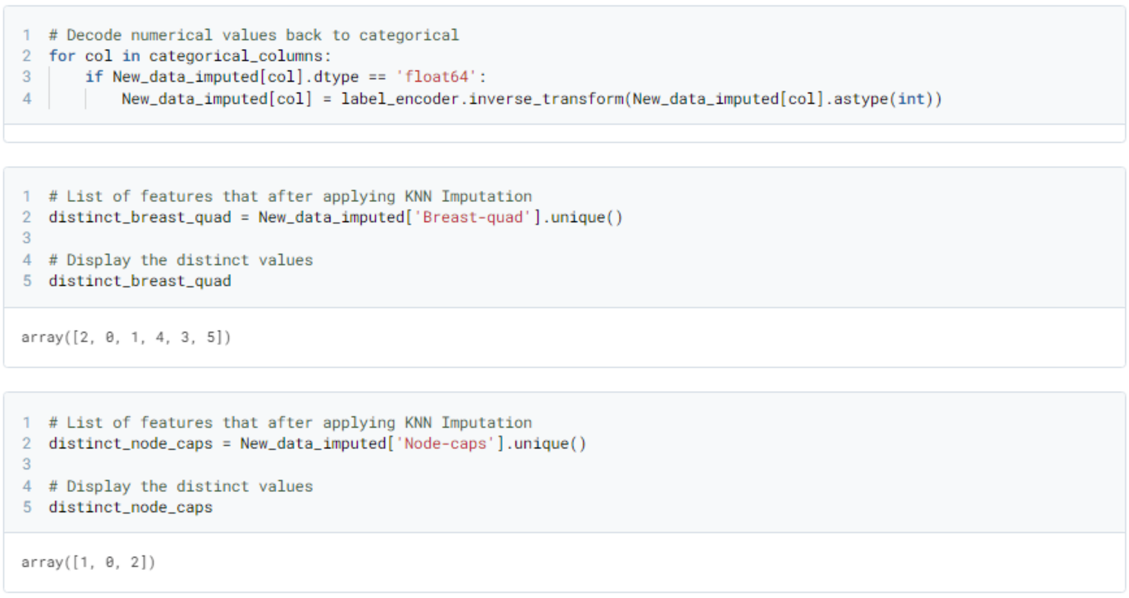
```python
# Display the DataFrame after imputation
New_data_imputed
```

	Age float64	Menopause object	Tumor-size object	Nodes float64	Node-caps int64	Degree-of-malign...	Breast object
	25.0 - 77.0	'premeno' 52.5% 'ge40' 45.1% 'lt40' 2.5%	'30-34' 21.1% '25-29' 18.7% 9 others 60.2%	0.0 - 25.0	0 - 2	'2' 45.4% '3' 29.6% '1' 25%	'left' 53.2% 'right' 46.8%
0	45	'premeno'	'15-19'	1	1	'3'	'right'
1	57	'ge40'	'15-19'	0	0	'1'	'right'
2	56	'ge40'	'35-39'	2	0	'2'	'left'
3	42	'premeno'	'35-39'	2	1	'3'	'right'
4	44	'premeno'	'30-34'	5	1	'2'	'left'
5	55	'premeno'	'25-29'	4	0	'2'	'right'
6	56	'ge40'	'40-44'	2	0	'3'	'left'
7	49	'premeno'	'10-14'	1	0	'2'	'left'
8	48	'premeno'	'0-4'	2	0	'2'	'right'
9	44	'ge40'	'40-44'	15	1	'2'	'right'

284 rows, showing 10 per page   Page 1 of 29

Figure 3.18: Initialize The Imputer - 1

The KNN imputation process helps fill missing values in the dataset, providing a more complete dataset for analysis. This imputed dataset (New_data_imputed) can now be used for further exploration and modeling. It's essential to assess the impact of imputation on the overall analysis and adjust the modeling process accordingly.



```
# Decode numerical values back to categorical
for col in categorical_columns:
    if New_data_imputed[col].dtype == 'float64':
        New_data_imputed[col] = label_encoder.inverse_transform(New_data_imputed[col].astype(int))
```

```
# List of features that after applying KNN Imputation
distinct_breast_quad = New_data_imputed['Breast-quad'].unique()

# Display the distinct values
distinct_breast_quad
```

```
array([2, 0, 1, 4, 3, 5])
```

```
# List of features that after applying KNN Imputation
distinct_node_caps = New_data_imputed['Node-caps'].unique()

# Display the distinct values
distinct_node_caps
```

```
array([1, 0, 2])
```

Figure 3.19: Initialize The Imputer - 2

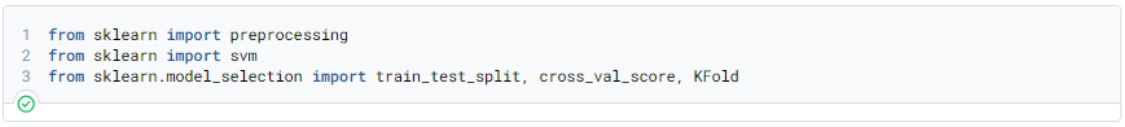
3.3 Support Vector Machine (SVM)

SVM constructs a hyperplane to separate classes with the maximum margin, often using kernel tricks for non-linear separation.

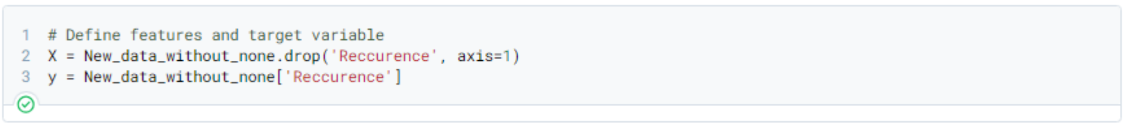
- Implementation: Train SVM with different kernels (linear, polynomial, RBF) and tune hyperparameters.
- Evaluation: Assess accuracy, precision, recall, and F1-score using hold-out sampling and K-fold cross-validation.

3.3.1 Data Splitting

Train-Test Split: Divide the dataset into training and testing sets to train the model on one subset and validate its performance on another.



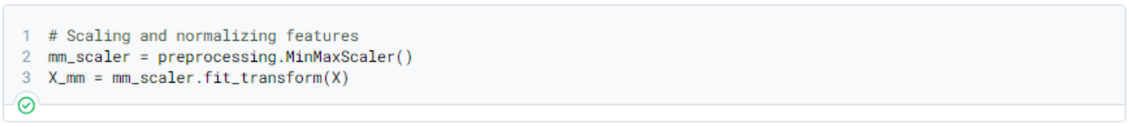
```python
from sklearn import preprocessing
from sklearn import svm
from sklearn.model_selection import train_test_split, cross_val_score, KFold
```



```python
# Define features and target variable
X = New_data_without_none.drop('Reccurence', axis=1)
y = New_data_without_none['Reccurence']
```

Figure 3.20: Define Features and Target Variable

Before we split the data, we normalize and scale the numeric variables as shown in figure 3.22.

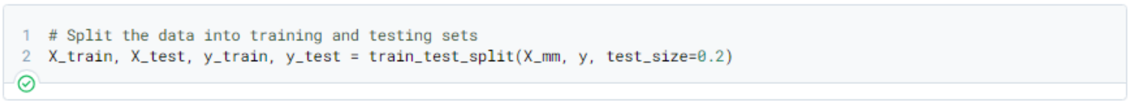


```python
# Scaling and normalizing features
mm_scaler = preprocessing.MinMaxScaler()
X_mm = mm_scaler.fit_transform(X)
```

Figure 3.21: Split The Data Into Train and Test Sets

3.3.2 Feature Scaling/Normalization

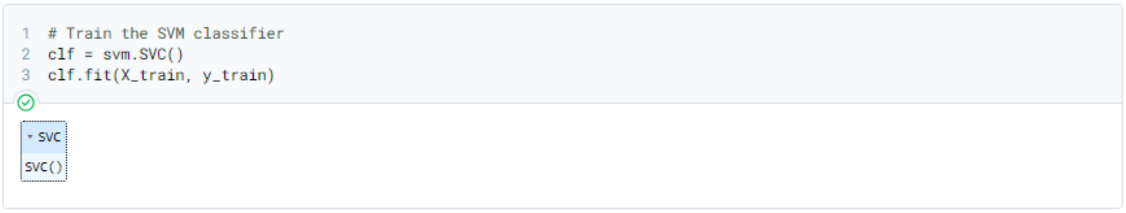
Scale Numerical Features: Standardize or normalize numerical features to a similar scale to avoid bias in models that rely on distance measures.



```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_mm, y, test_size=0.2)
```

Figure 3.22: Normalize and Scale The Data

3.3.3 Initialize the model



```
# Train the SVM classifier
clf = svm.SVC()
clf.fit(X_train, y_train)
```

```
SVC
SVC()
```

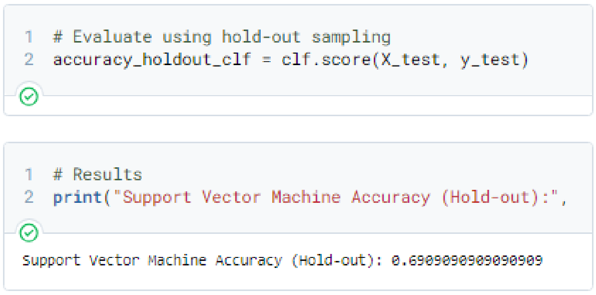
Figure 3.23: Initialize Support Vector Machine

3.3.4 Cross-Validation

Use cross-validation techniques to assess model performance robustly and avoid overfitting.

Hold-out Sampling

Split the dataset into training and testing sets, train the models on the training set, and evaluate their performance on the unseen testing set.



```
# Evaluate using hold-out sampling
accuracy_holdout_clf = clf.score(X_test, y_test)
```

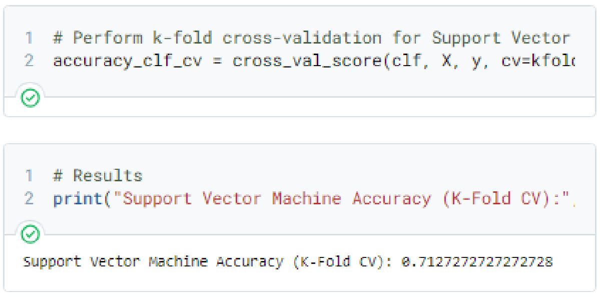
```
# Results
print("Support Vector Machine Accuracy (Hold-out):",
```

```
Support Vector Machine Accuracy (Hold-out): 0.6909090909090909
```

Figure 3.24: Evaluate Using Hold-Out Sampling

K-fold Cross-Validation

Divide the dataset into k subsets (folds), train the models on k-1 folds, and validate on the remaining fold. This process is repeated k times, and the performance is averaged.



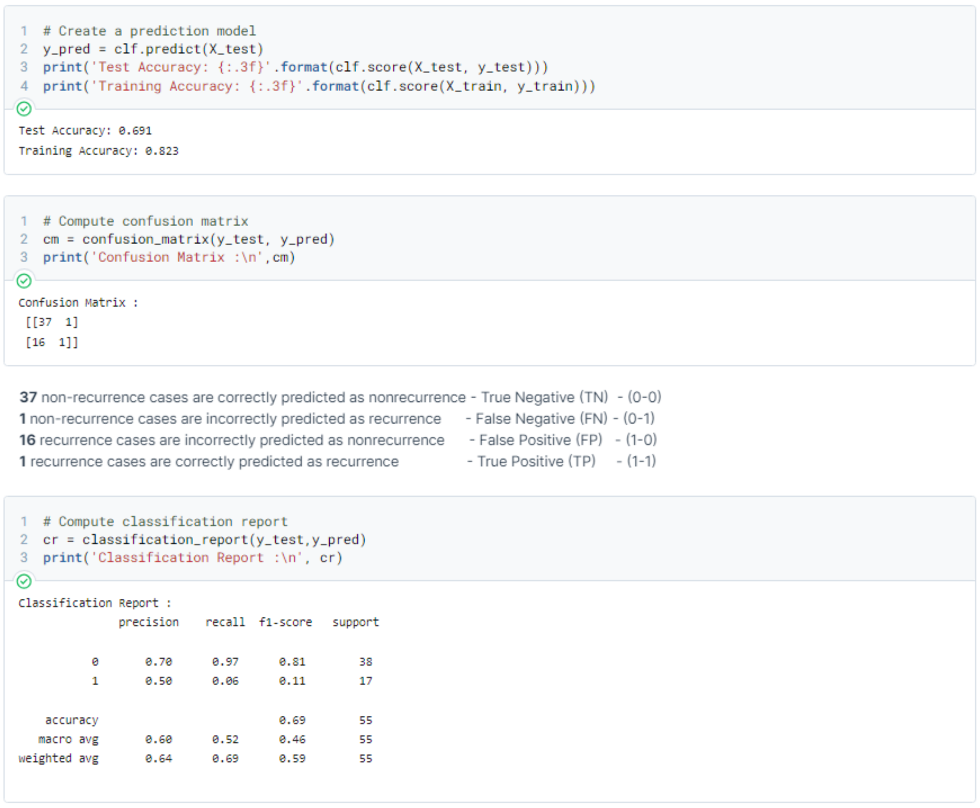
```
# Perform k-fold cross-validation for Support Vector
accuracy_clf_cv = cross_val_score(clf, X, y, cv=kfol
```

```
# Results
print("Support Vector Machine Accuracy (K-Fold CV):",
```

```
Support Vector Machine Accuracy (K-Fold CV): 0.7127272727272728
```

Figure 3.25: Evaluate Using K-Fold Cross-Validation

3.3.5 Testing The Model



```
# Create a prediction model
y_pred = clf.predict(X_test)
print('Test Accuracy: {:.3f}'.format(clf.score(X_test, y_test)))
print('Training Accuracy: {:.3f}'.format(clf.score(X_train, y_train)))
```

```
Test Accuracy: 0.691
Training Accuracy: 0.823
```

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[37  1]
 [16  1]]
```

37 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0)
1 non-recurrence cases are incorrectly predicted as recurrence - False Negative (FN) - (0-1)
16 recurrence cases are incorrectly predicted as nonrecurrence - False Positive (FP) - (1-0)
1 recurrence cases are correctly predicted as recurrence - True Positive (TP) - (1-1)

```
# Compute classification report
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.70      0.97      0.81        38
           1       0.50      0.06      0.11        17

    accuracy                           0.69        55
   macro avg       0.60      0.52      0.46        55
weighted avg       0.64      0.69      0.59        55
```

Figure 3.26: Create A Prediction Model

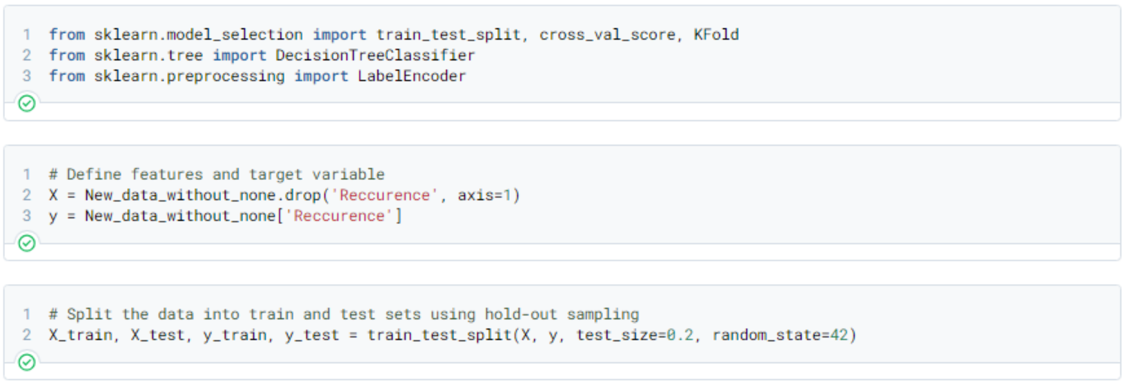
3.4 Decision Tree Classifier

Decision trees partition data into subsets based on features to create a tree-like model for classification.

- Implementation: Train decision trees and evaluate using different tree depths or pruning techniques.
- Evaluation: Measure accuracy, precision, recall, and F1-score with hold-out sampling and K-fold cross-validation.

3.4.1 Data Splitting

Train-Test Split: Divide the dataset into training and testing sets to train the model on one subset and validate its performance on another.



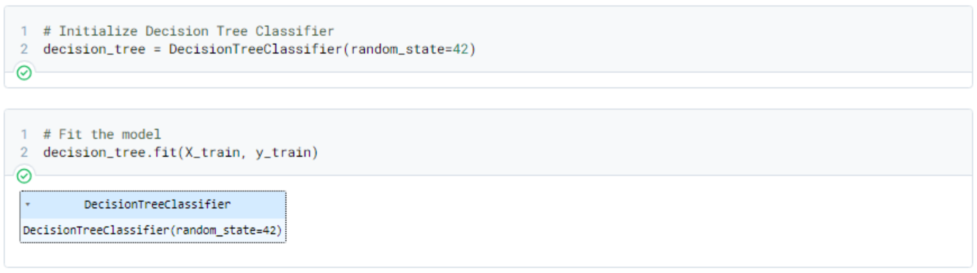
```
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

```
# Define features and target variable
X = New_data_without_none.drop('Reccurence', axis=1)
y = New_data_without_none['Reccurence']
```

```
# Split the data into train and test sets using hold-out sampling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 3.27: Define Features and Target Variable

3.4.2 Initialize the model



```
# Initialize Decision Tree Classifier
decision_tree = DecisionTreeClassifier(random_state=42)
```

```
# Fit the model
decision_tree.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

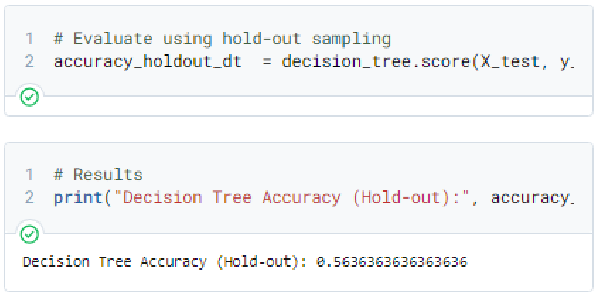
Figure 3.28: Initialize Decision Tree Classifier

3.4.3 Cross-Validation

Use cross-validation techniques to assess model performance robustly and avoid overfitting.

Hold-out Sampling

Split the dataset into training and testing sets, train the models on the training set, and evaluate their performance on the unseen testing set.



```
# Evaluate using hold-out sampling
accuracy_holdout_dt  = decision_tree.score(X_test, y
```

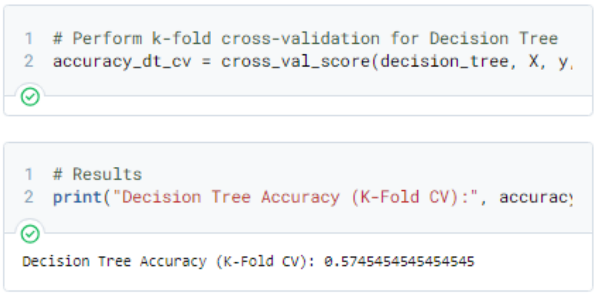
```
# Results
print("Decision Tree Accuracy (Hold-out):", accuracy
```

```
Decision Tree Accuracy (Hold-out): 0.5636363636363636
```

Figure 3.29: Evaluate Using Hold-Out Sampling

K-fold Cross-Validation

Divide the dataset into k subsets (folds), train the models on k-1 folds, and validate on the remaining fold. This process is repeated k times, and the performance is averaged.



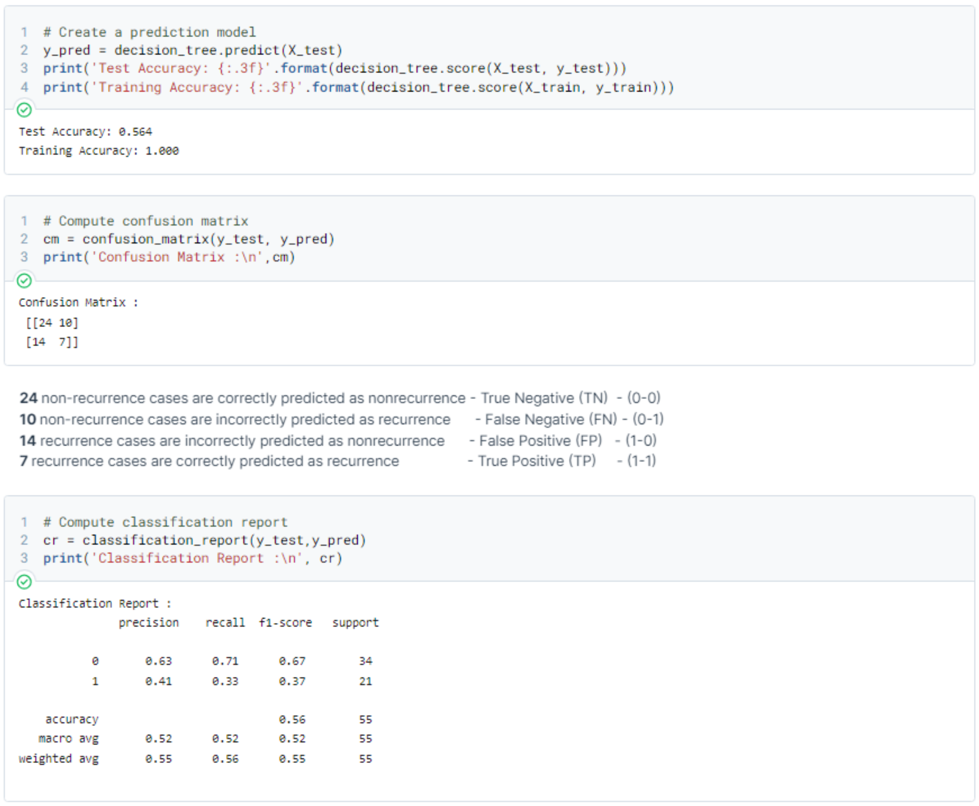
```
# Perform k-fold cross-validation for Decision Tree
accuracy_dt_cv = cross_val_score(decision_tree, X, y
```

```
# Results
print("Decision Tree Accuracy (K-Fold CV):", accurac
```

```
Decision Tree Accuracy (K-Fold CV): 0.5745454545454545
```

Figure 3.30: Evaluate Using K-Fold Cross-Validation

3.4.4 Testing The Model



```python
# Create a prediction model
y_pred = decision_tree.predict(X_test)
print('Test Accuracy: {:.3f}'.format(decision_tree.score(X_test, y_test)))
print('Training Accuracy: {:.3f}'.format(decision_tree.score(X_train, y_train)))
```

```
Test Accuracy: 0.564
Training Accuracy: 1.000
```

```python
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[24 10]
 [14  7]]
```

24 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0)
10 non-recurrence cases are incorrectly predicted as recurrence - False Negative (FN) - (0-1)
14 recurrence cases are incorrectly predicted as nonrecurrence - False Positive (FP) - (1-0)
7 recurrence cases are correctly predicted as recurrence - True Positive (TP) - (1-1)

```python
# Compute classification report
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

```
Classification Report :
               precision    recall  f1-score   support

           0       0.63      0.71      0.67        34
           1       0.41      0.33      0.37        21

    accuracy                           0.56        55
   macro avg       0.52      0.52      0.52        55
weighted avg       0.55      0.56      0.55        55
```

Figure 3.31: Create A Prediction Model

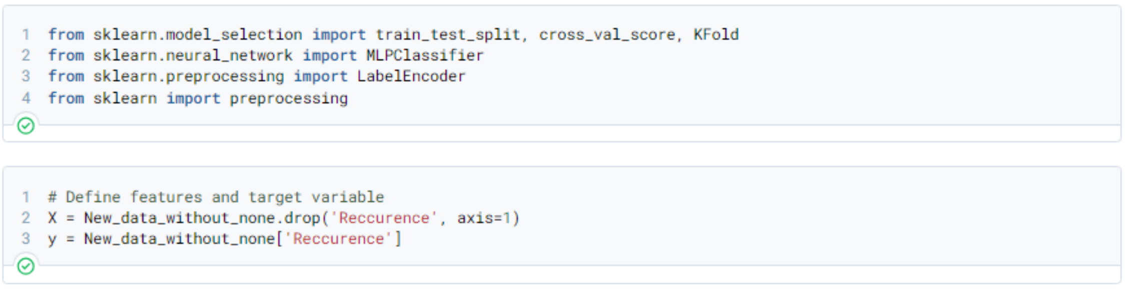
3.5 Artificial Neural Network (ANN)

ANN is a network of interconnected nodes inspired by the human brain, capable of learning complex patterns.

- Implementation: Design and train a neural network with multiple layers, neurons, and activation functions.
- Evaluation: Assess accuracy, precision, recall, and F1-score using hold-out sampling and K-fold cross-validation.

3.5.1 Data Splitting

Train-Test Split: Divide the dataset into training and testing sets to train the model on one subset and validate its performance on another.

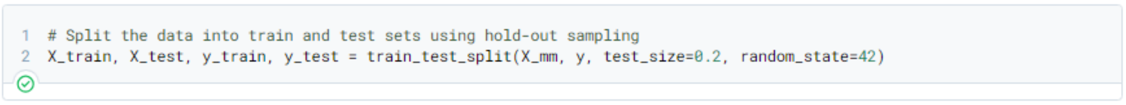


```
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
```

```
# Define features and target variable
X = New_data_without_none.drop('Reccurence', axis=1)
y = New_data_without_none['Reccurence']
```

Figure 3.32: Define Features and Target Variable

Before we split the data, we normalize and scale the numeric variables as shown in figure 3.34.



```
# Split the data into train and test sets using hold-out sampling
X_train, X_test, y_train, y_test = train_test_split(X_mm, y, test_size=0.2, random_state=42)
```

Figure 3.33: Split The Data Into Train and Test Sets

3.5.2 Feature Scaling/Normalization

Scale Numerical Features: Standardize or normalize numerical features to a similar scale to avoid bias in models that rely on distance measures.

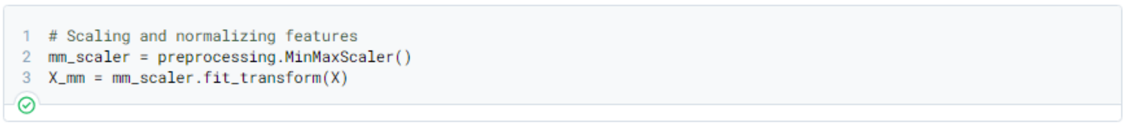
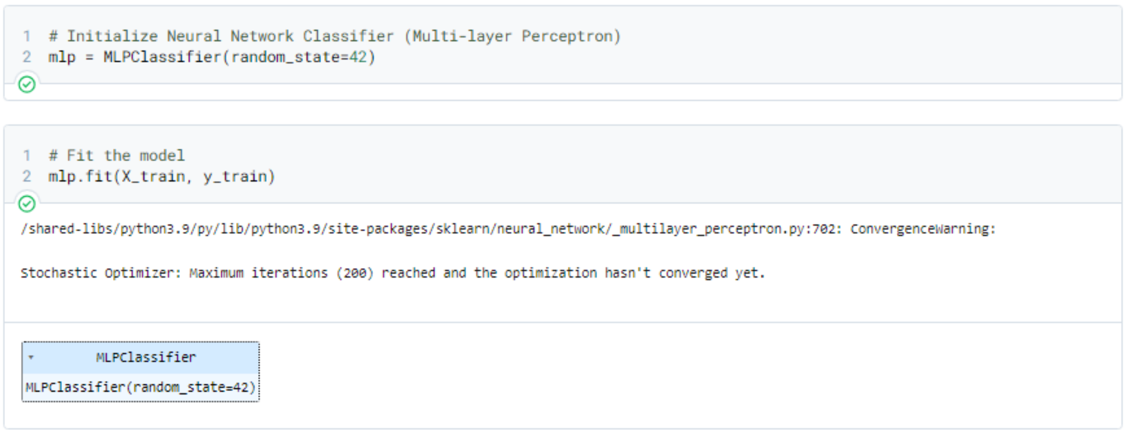

```python
# Scaling and normalizing features
mm_scaler = preprocessing.MinMaxScaler()
X_mm = mm_scaler.fit_transform(X)
```

Figure 3.34: Normalize and Scale The Data

3.5.3 Initialize the model


```python
# Initialize Neural Network Classifier (Multi-layer Perceptron)
mlp = MLPClassifier(random_state=42)
```

```python
# Fit the model
mlp.fit(X_train, y_train)
```

```
/shared-libs/python3.9/py/lib/python3.9/site-packages/sklearn/neural_network/_multilayer_perceptron.py:702: ConvergenceWarning:

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
```

```
MLPClassifier
MLPClassifier(random_state=42)
```

Figure 3.35: Initialize Neural Network Classifier

3.5.4 Cross-Validation

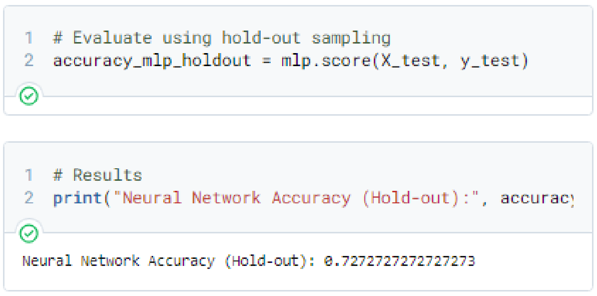
Use cross-validation techniques to assess model performance robustly and avoid overfitting.

Hold-out Sampling

Split the dataset into training and testing sets, train the models on the training set, and evaluate their performance on the unseen testing set.

K-fold Cross-Validation

Divide the dataset into k subsets (folds), train the models on k-1 folds, and validate on the remaining fold. This process is repeated k times, and the performance is averaged.

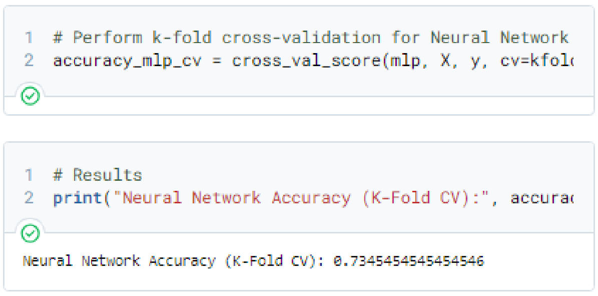


```
# Evaluate using hold-out sampling
accuracy_mlp_holdout = mlp.score(X_test, y_test)
```

```
# Results
print("Neural Network Accuracy (Hold-out):", accurac
```

```
Neural Network Accuracy (Hold-out): 0.7272727272727273
```

Figure 3.36: Evaluate Using Hold-Out Sampling



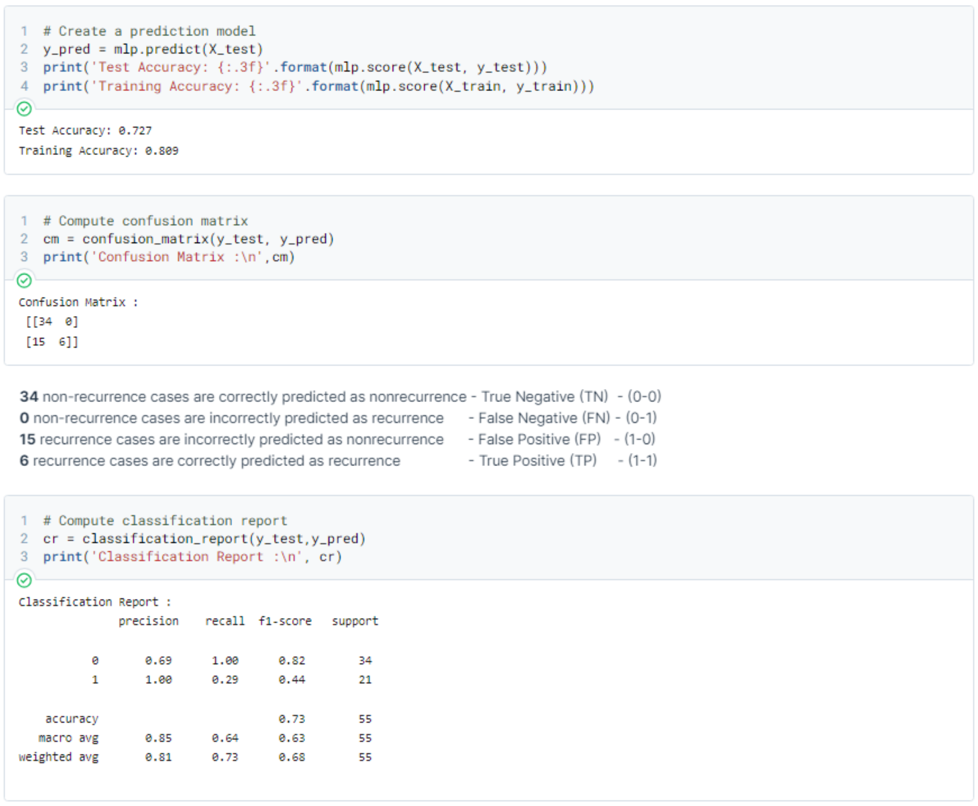
```
# Perform k-fold cross-validation for Neural Network
accuracy_mlp_cv = cross_val_score(mlp, X, y, cv=kfol
```

```
# Results
print("Neural Network Accuracy (K-Fold CV):", accura
```

```
Neural Network Accuracy (K-Fold CV): 0.7345454545454546
```

Figure 3.37: Evaluate Using K-Fold Cross-Validation

3.5.5 Testing The Model



```
# Create a prediction model
y_pred = mlp.predict(X_test)
print('Test Accuracy: {:.3f}'.format(mlp.score(X_test, y_test)))
print('Training Accuracy: {:.3f}'.format(mlp.score(X_train, y_train)))
```

```
Test Accuracy: 0.727
Training Accuracy: 0.809
```

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[34  0]
 [15  6]]
```

34 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0)
0 non-recurrence cases are incorrectly predicted as recurrence - False Negative (FN) - (0-1)
15 recurrence cases are incorrectly predicted as nonrecurrence - False Positive (FP) - (1-0)
6 recurrence cases are correctly predicted as recurrence - True Positive (TP) - (1-1)

```
# Compute classification report
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

```
Classification Report :
               precision    recall  f1-score   support

           0       0.69      1.00      0.82        34
           1       1.00      0.29      0.44        21

    accuracy                           0.73        55
   macro avg       0.85      0.64      0.63        55
weighted avg       0.81      0.73      0.68        55
```

Figure 3.38: Create A Prediction Model

3.6 Discussion

- Naive Bayes performed well with both hold-out sampling and K-fold cross-validation, suggesting that it is a robust model for this dataset. This is likely because the Naive Bayes algorithm is based on the assumption that the features are independent of each other. This assumption is often violated in real-world datasets, but it may be approximately true for the dataset used in this study.

- The KNN model has a higher average K-fold cross-validation score 0.749 compared to GNB 0.731, indicating better generalization The GNB model has a slightly higher hold-out test accuracy 0.727 compared to KNN 0.673. KNN is a sensitive algorithm that can be affected by outliers and noise in the data.

- SVM exhibited good performance with hold-out sampling, indicating robust performance, but had a lower K-fold cross-validation score. This suggests that SVM may be prone to overfitting.

- Decision Tree's performance was relatively low, suggesting that it may not be well-suited for this dataset. This is likely because decision trees are not well-suited for handling complex relationships between features.

- ANN demonstrated promising results with both hold-out sampling and K-fold cross-validation, indicating that it is a capable model for this task. ANNs are powerful algorithms that can learn complex relationships between features.

Overall, SVM and ANN appear to be the most promising models for this dataset.

3.6.1 Conclusion

SVM shows the highest hold-out accuracy, but there's a notable discrepancy between hold-out and cross-validation results, suggesting potential overfitting.

Naïve Bayes and ANN exhibit relatively consistent performances across test and cross-validation sets, indicating good generalization.

Decision tree's low accuracy on both test and cross-validation sets might indicate overfitting or insufficient model complexity for the data.

KNN shows moderate performance but might benefit from tuning hyperparameters or scaling features.

Appendices

A Full Report

Machine Learning Project

Sawsan Daban - Alaa AlSharekh

Learning tasks

- **Binary Classification - Recurrence Prediction:**

Task: Predict whether a patient will experience a recurrence of breast cancer (binary outcome: recurrence or r recurrence). Target Variable: Recurrence column.

- **Multiclass Classification - Cancer Severity Prediction:**

Task: Predict the severity level of breast cancer based on the degree-of-malignance. Target Variable: degree-malignance (assuming it has multiple classes).

- **Regression - Age Prediction:**

Task: Predict the age of a patient based on other available features. Target Variable: Age.

- **Categorical Classification - Breast Type Prediction:**

Task: Predict the type of breast involved (left or right). Target Variable: Breast column.

- **Categorical Classification - Menopause Prediction:**

Task: Predict whether a patient is in menopause or not. Target Variable: Menopause column.

- **Clustering - Patient Segmentation:**

Task: Cluster patients based on their features to identify subgroups with similar characteristics.

- **Association Rule Mining - Patterns in Treatment:**

Task: Identify patterns or associations between different features and the type of treatment received (Irradiati

Machine Learning Model

For predicting disease recurrence (a binary classification task), several machine learning models can be applie choice of model often depends on various factors like the size of the dataset, the complexity of relationships, interpretability, and computational efficiency. Here are some suitable ML models for predicting disease recurre

- **Logistic Regression:**

Suitable for binary classification tasks. Interpretable and provides probabilities. Works well with linearly separa

- **Decision Trees and Random Forests:**

Effective for classification tasks. Can handle nonlinear relationships and interactions between features. Rando reduce overfitting and increase accuracy by combining multiple decision trees.

- **Support Vector Machines (SVM):**

Effective in high-dimensional spaces. Works well with both linear and nonlinear data. Finds the best separatio (hyperplane) between classes.

- **Neural Networks:**

Deep learning models suitable for complex, nonlinear relationships. Requires more data and computational po capture intricate patterns.

- **Naive Bayes:**

Simple and efficient for binary classification. Assumes independence between features (which might not hold cases).

- **K-Nearest Neighbors (KNN):**

Non-parametric and instance-based method. Predicts based on the majority class among its nearest neighbo

When choosing a model, considerations include the dataset size, feature importance, interpretability, comput resources, and the trade-off between accuracy and model complexity. Additionally, techniques such as cross- hyperparameter tuning, and feature selection can enhance model performance.

Preparing a dataset for machine learning

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.

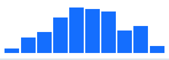
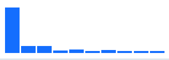
Observations: Each row in the dataset repres observation of an individual patient. The datas multiple observations, each corresponding to patient.

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

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

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

# To read the data
Data = pd.read_excel('Disease-Reccurence DCS 873.xlsx')
```

```
# Display the current data frame
Data
```

	Age int64	Menopause object	Tumor-size object	nodes int64	Unnamed: 4 object	deg
	25 - 77 	'premeno' 52.4% 'ge40' 45.1% 'lt40' 2.4%	'30-34' 21% '25-29' 18.9% 9 others 60.1%	0 - 25 	'no' 77.6% 'yes' 19.6% Missing 2.8%	'2' '3' '1'
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'	1	'no'	'2'
8	48	'premeno'	'0-4'	2	'no'	'2'
9	44	'ge40'	'40-44'	15	'yes'	'2'

286 rows, showing 10 per page   Page 1 of 29

Quality report

Data Exploration and Understanding (EDA)

EDA plays a critical role in the machine learning workflow. It helps us dig into the data, uncover patterns, and u
how different things relate to each other. This exploration is crucial before we dive into building machine learn
as it gives us the necessary insights for better decision-making.

```
Data.shape
```

```
(286, 10)
```

```
# Info about the dataset
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 286 entries, 0 to 285
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 286 non-null    int64
 1   Menopause           286 non-null    object
 2   Tumor-size          286 non-null    object
 3   nodes               286 non-null    int64
 4   Unnamed: 4          278 non-null    object
 5   degree-of-malignance  286 non-null    object
 6   Breast              286 non-null    object
 7   Breast-quad         285 non-null    object
 8   Irradiation         286 non-null    object
 9   reccurence          286 non-null    object
dtypes: int64(2), object(8)
memory usage: 22.5+ KB
```

```
# To describe the continious featurs Age , nodes
Data.describe()
```

	Age float64	nodes float64
cou...	286	286
me...	52.07342657	2.818181818
std	10.43332348	3.421575227
min	25	0
25%	45	1
50%	52	2
75%	59	3
max	77	25

8 rows, showing 10 per page   Page 1 of 1

```python
# Data Profiling EDA

!pip install ydata-profiling
!pip install ipywidgets==8.1.1
!pip install --upgrade pip
```

```python
from ydata_profiling import ProfileReport

# Generate the data profiling report
report1 = ProfileReport(Data, title='Disease-Reccurence - report1 ')
report1.to_file("Disease-Reccurence - report1.html")
```

```python
report1
```

Overview

Dataset statistics

Number of variables	10
Number of observations	286
Missing cells	9
Missing cells (%)	0.3%
Duplicate rows	2
Duplicate rows (%)	0.7%
Total size in memory	22.5 KiB
Average record size in memory	80.4 B

Variable types

Numeric	2
Categorical	8

The provided summary offers a concise overview of the existing dataset and highlights certain noteworthy as
According to the description, our dataset comprises 10 variables and encompasses a total of 286 observation
these variables, 8 fall under the categorical category, while 2 are designated as numerical variables. Additiona
summary points out the presence of duplicate rows and an unnamed column as part of the dataset's characte

> ***Please find below the full report:*** https://msc-science-in-computing-2023.github.io/ML-Reports1

Apply quality controls

Data cleaning: handling duplication

2 duplicate rows were identified.

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

	Age int64	Menopause object	Tumor-size object	nodes int64	Unnamed: 4 object	degi
178	47	'premeno'	'25-29'	1	'no'	'2'
239	56	'ge40'	'40-44'	7	'yes'	'3'

2 rows, showing 10 per page Page 1 of 1

```
# Remove doublication
New_data = Data.drop_duplicates()
```

```
New_data.shape
```

```
(284, 10)
```

After removing duplications, # of rows has been decreased to **284** instead of **286**

Data cleaning: missing column name

Based on the provided problem statement, we have identified the column with missing information.

```
# Assign a name to the unnamed column  (Unnamed: 4)
New_data.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_data.rename(columns={'nodes': 'Nodes'}, inplace=True)
New_data.rename(columns={'degree-of-malignance': 'Degree-of-malignance'}, inplace=True)
New_data.rename(columns={'reccurence': 'Reccurence'}, inplace=True)
```

Data cleaning: unified missing 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

Based on the report, there are a total of **8** missing values in Node-caps out of **284** observations which represe
0.3%.

```
# Node-caps

# Check for null values in 'Node-caps' column
missing_node_caps_values = New_data['Node-caps'].apply(lambda x: pd.isnull(x) or (isinstan

# Display the rows where the column had missing values
New_data[missing_node_caps_values]
```

	Age int64	Menopause object	Tumor-size object	Nodes int64	Node-caps object	Deg
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'
71	61	'ge40'	'25-29'	5	nan	'1'
92	51	'lt40'	'20-24'	0	nan	'1'
149	50	'ge40'	'30-34'	9	nan	'3'
264	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_data.loc[missing_node_caps_values, 'Node-caps'] = None

# Display the rows where the column had missing values
New_data[missing_node_caps_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
  self._setitem_single_column(loc, value, pi)
```

	Age int64	Menopause object	Tumor-size object	Nodes int64	Node-caps object	Deg
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'
71	61	'ge40'	'25-29'	5	None	'1'
92	51	'lt40'	'20-24'	0	None	'1'
149	50	'ge40'	'30-34'	9	None	'3'
264	57	'ge40'	'30-34'	11	None	'3'

8 rows, showing 10 per page Page 1 of 1

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

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

```
# Breast-quad

# Check for null values in 'Breast-quad' column
missing_breast_quad_values = New_data['Breast-quad'].apply(lambda x: pd.isnull(x) or (isin

# Display the rows where the column had missing values
New_data[missing_breast_quad_values]
```

	Age int64	Menopause object	Tumor-size object	Nodes int64	Node-caps object	Deg
240	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_data.loc[missing_breast_quad_values, 'Breast-quad'] = None

# Display the rows where the column had missing values
New_data[missing_breast_quad_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
  self._setitem_single_column(loc, value, pi)
```

	Age int64	Menopause object	Tumor-size object	Nodes int64	Node-caps object	Deg

240	59	'ge40'	'30-34'	1	'no'	'3'

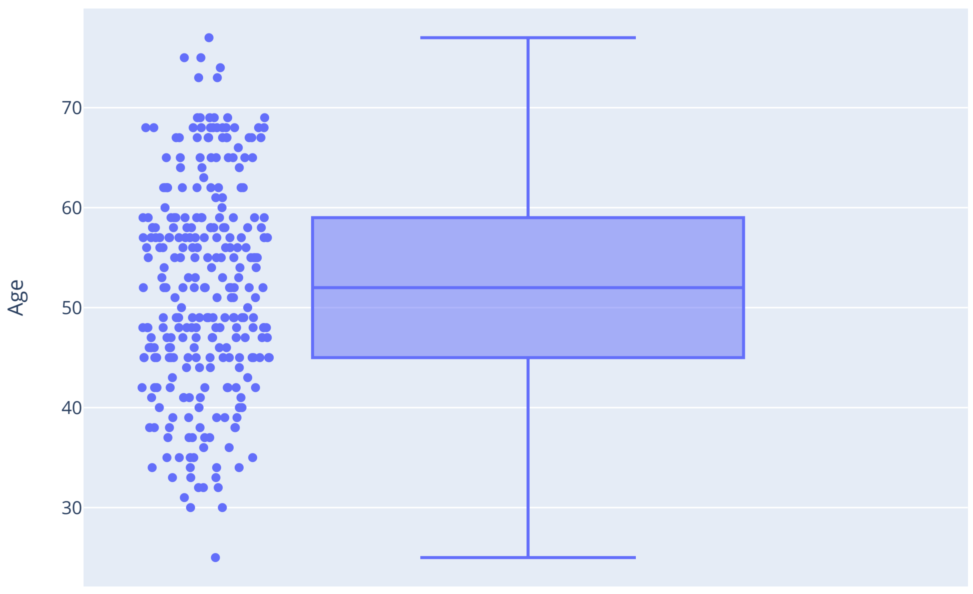
1 row, showing 10 per page Page 1 of 1

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

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

Data cleaning: Checking outlier

```
# for Age

import plotly.express as px

fig = px.box(New_data, y="Age", points="all")
fig.show()
```



Validation step

```python
from ydata_profiling import ProfileReport

# Generate the data profiling report
report2 = ProfileReport(New_data, title='Disease-Reccurence - report2 ')
report2.to_file("Disease-Reccurence - report2.html")
```

```python
report2
```

Overview

Dataset statistics

Number of variables	10
Number of observations	284
Missing cells	9
Missing cells (%)	0.3%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	24.4 KiB
Average record size in memory	88.0 B

Variable types

Numeric	2
Categorical	8

Please find below the full report: https://msc-science-in-computing-2023.github.io/ML-Reports2

Based on the report outcomes following the implementation of the necessary checks, it appears that addition values have surfaced.

Data cleaning: Removing None values

```python
# Current dataset with None = New_data
# New dataset without None = New_data_without_none

New_data_without_none = New_data.dropna()
```

```python
New_data_without_none.shape
```

```
(275, 10)
```

```python
from ydata_profiling import ProfileReport

# Generate the data profiling report
report3 = ProfileReport(New_data_without_none, title='Disease-Reccurence - report3 ')
report3.to_file("Disease-Reccurence - report3.html")
```

```python
report3
```

Overview

Dataset statistics

Number of variables	10
Number of observations	275
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	23.6 KiB
Average record size in memory	88.0 B

Variable types

Numeric	2
Categorical	8

Please find below the full report: https://msc-science-in-computing-2023.github.io/ML-Reports3

After creating a new dataset without None values. The dataset has been decreased from **284** to **275.**

Data cleaning: encoding categorical variables

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

```
# Encoding categorical variables
#encoder = LabelEncoder()
#categorical_cols = ['Menopause', 'Tumor-size', 'Node-caps', 'Degree-of-malignance', 'Brea
#for col in categorical_cols:
#    New_data[col] = encoder.fit_transform(New_data[col])
# Encoding New_data_without_none

encoder = LabelEncoder()
categorical_cols = ['Menopause', 'Tumor-size', 'Node-caps', 'Degree-of-malignance', 'Breas
for col in categorical_cols:
    New_data_without_none[col] = encoder.fit_transform(New_data_without_none[col])
```

```
New_data_without_none
```

	Age int64 25 - 77	Menopause int64 0 - 2	Tumor-size int64 0 - 10	Nodes int64 0 - 25	Node-caps int64 0 - 1	Deg 0 - 2
0	45	2	2	1	1	
1	57	0	2	0	0	
2	56	0	6	2	0	
3	42	2	6	2	1	
4	44	2	5	5	1	
5	55	2	4	4	0	
6	56	0	7	2	0	
7	49	2	1	1	0	
8	48	2	0	2	0	
9	44	0	7	15	1	

275 rows, showing 10 per page Page 1 of 28

Data cleaning: imbalanced data

Resampling: Address class imbalance using techniques like oversampling (creating more samples of the minor undersampling (reducing samples of the majority class). Use Appropriate Evaluation Metrics: Consider metrics precision, recall, F1-score, or AUC-ROC for imbalanced datasets instead of accuracy.

Several techniques can be employed to assess whether the target variable is imbalanced or not.

```
# Class distribution

class_distribution = New_data['Reccurence'].valu
class_distribution
```

```
'no-recurrence-events'    201
'recurrence-events'        83
Name: Reccurence, dtype: int64
```

```
# Class distribution

class_distribution = New_data_without_
class_distribution
```

```
0    196
1     79
Name: Reccurence, dtype: int64
```

```
# Class ratio

class_ratios = New_data['Reccurence'].value_coun
class_ratios
```

```
'no-recurrence-events'    0.707746
'recurrence-events'       0.292254
Name: Reccurence, dtype: float64
```

```
# Class ratio

class_ratios = New_data_without_none['
class_ratios
```

```
0    0.712727
1    0.287273
Name: Reccurence, dtype: float64
```

Implementation and Evaluation

Experimenting with multiple models and evaluating their performance using metrics like accuracy, precision, re score, and area under the ROC curve (AUC-ROC) would help determine the most suitable model for predicting recurrence in your specific dataset.

Naïve Bayes Classifier GaussianNB

Description: Naïve Bayes is a probabilistic classifier based on Bayes' theorem. It assumes independence betw features given the class. Implementation: Train the Naïve Bayes model using the dataset. Evaluation: Assess m performance using accuracy, precision, recall, and F1-score for both hold-out sampling and K-fold cross-valid

```
from sklearn                 import preprocessing
from sklearn.metrics         import classification_report
from sklearn.metrics         import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes     import GaussianNB
from sklearn.metrics         import accuracy_score
from sklearn.preprocessing   import LabelEncoder, MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.impute          import SimpleImputer
```

For continuous and categorical data

Data Splitting: Train-Test Split: Divide the dataset into training and testing sets to train the model on one subs validate its performance on another.

```
# Define features and target variable
x = New_data_without_none.drop('Reccurence', axis=1)
y = New_data_without_none['Reccurence']
```

Feature Scaling/Normalization: Scale Numerical Features: Standardize or normalize numerical features to a si to avoid bias in models that rely on distance measures.

```
# Separate categorical and numeric features
numeric_features = x.select_dtypes(include=['float64', 'int64'])
```

```
# Normalizing numeric features
scaler = MinMaxScaler()
x_normalized_numeric = scaler.fit_transform(numeric_features)

# Ensure x and y have the same number of rows
assert len(x_normalized_numeric) == len(y), "Number of samples in x and y must be the same
```

```
# Split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x_normalized_numeric, y, test_size=0.2
```

Initialize the model

```
# Modeling and evaluation
model = GaussianNB()

# Train the model
model.fit(x_train, y_train)
```

```
▾ GaussianNB

GaussianNB()
```

Cross-Validation: Use cross-validation techniques to assess model performance robustly and avoid overfittin

Hold-out Sampling: Split the dataset into training and testing sets, train the models on the training set, and evaluate their performance on the unseen testing set.

```
# Testing the model on hold-out set
y_pred_holdout = model.predict(x_test)
print('Hold-out Test Accuracy: {:.3f}'.format(mo
print('Hold-out Training Accuracy: {:.3f}'.forma
```

```
Hold-out Test Accuracy: 0.727
Hold-out Training Accuracy: 0.736
```

This is an indicator of the accuracy of the model on the test set that was not seen during training. In this case, our model correctly predicted the target variable for approximately **72.7**% of the samples. In addition to how well the model performs on the training data. The model

K-fold Cross-Validation: Divide the dataset int (folds), train the models on k-1 folds, and valida remaining fold. This process is repeated k times performance is averaged.

```
# Evaluate the model using k-fold cros
cv_scores = cross_val_score(model, x_n

# Results
print('K-fold Cross-Validation Scores:
print('Average K-fold Cross-Validation
```

```
K-fold Cross-Validation Scores: [0.70909091 0
Average K-fold Cross-Validation Score: 0.731
```

The k-fold cross-validation scores represent th of your model across different folds (splits) of t The scores for each fold are as follows: [0.709, 0.691, 0.800, 0.691]. The average k-fold cross-v score is calculated as **0.731**, representing the c model performance across the different folds.

achieved an accuracy of approximately **73.6**% on the training set.

Testing the model

```
# Create a prediction model
y_pred = model.predict(x_test)
print('Test Accuracy: {:.3f}'.format(model.score(x_test, y_test)))
print('Training Accuracy: {:.3f}'.format(model.score(x_train, y_train)))
```

```
Test Accuracy: 0.727
Training Accuracy: 0.736
```

```
# Compute confusion matrix

cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[30  4]
 [11 10]]
```

30 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0) **4** non-recurre
are incorrectly predicted as recurrence - False Negative (FN) - (0-1) **11** recurrence cases are incorrectly predic
nonrecurrence - False Positive (FP) - (1-0) **10** recurrence cases are correctly predicted as recurrence - True P
- (1-1)

```
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

```
Classification Report :
               precision    recall  f1-score   support


           0       0.73      0.88      0.80        34
           1       0.71      0.48      0.57        21


    accuracy                           0.73        55
   macro avg       0.72      0.68      0.69        55
weighted avg       0.73      0.73      0.71        55
```

Precision: The proportion of true positive predictions among all positive predictions. Recall: The proportion of
positive predictions among all actual positive instances. F1-score: The harmonic mean of precision and recall.
The number of actual occurrences of the class in the specified dataset.

K-Nearest Neighbors (KNN)

Description: KNN is an instance-based learning algorithm that classifies new instances based on the majority k-nearest neighbors. Implementation: Train the KNN model using different values of k and evaluate. Evaluatior accuracy, precision, recall, and F1-score using hold-out sampling and K-fold cross-validation.

KNN Classification

```
from sklearn.neighbors        import KNeighborsClassifier
from sklearn.model_selection  import train_test_split
from sklearn.metrics          import accuracy_score
import numpy as np
```

Data Splitting: Train-Test Split: Divide the dataset into training and testing sets to train the model on one subs validate its performance on another.

```
# Define features and target variable
x = New_data_without_none.drop('Reccurence', axis=1)
y = New_data_without_none['Reccurence']
```

Feature Scaling/Normalization: Scale Numerical Features: Standardize or normalize numerical features to a si to avoid bias in models that rely on distance measures.

```
# Separate categorical and numeric features
# categorical_features = x.select_dtypes(include=['object'])
numeric_features = x.select_dtypes(include=['float64', 'int64'])

# Normalizing numeric features
scaler = MinMaxScaler()
x_normalized_numeric = scaler.fit_transform(numeric_features)
```

```
# Split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x_normalized_numeric, y, test_size=0.2
```

Initialize the model

```
# Modeling and evaluation
model = KNeighborsClassifier()

# Train the model
model.fit(x_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Cross-Validation: Use cross-validation techniques to assess model performance robustly and avoid overfittin

Hold-out Sampling: Split the dataset into training and testing sets, train the models on the training set, and evaluate their performance on the unseen testing set.

```
# Evaluate the model on hold-out set

# Results
print('Hold-out Test Accuracy: {:.3f}'.format(mo
```

```
Hold-out Test Accuracy: 0.673
```

This metric indicates the accuracy of the KNN model on the test set that was not seen during training. In this case, the model correctly predicted the target variable for approximately 67.3% of the samples.

K-fold Cross-Validation: Divide the dataset int (folds), train the models on k-1 folds, and valida remaining fold. This process is repeated k times performance is averaged.

```
# Evaluate the model using k-fold cros
cv_scores = cross_val_score(model, x_n

# Results
print('K-fold Cross-Validation Scores:
print('Average K-fold Cross-Validation
```

```
K-fold Cross-Validation Scores: [0.76363636 0
Average K-fold Cross-Validation Score: 0.749
```

The k-fold cross-validation scores represent th of the model across different folds (splits) of th The scores for each fold are as follows: [0.764, 0.709, 0.727, 0.709]. The average k-fold cross- score is calculated as **0.749**, representing the model performance across the different folds.

Testing the model

```
# Testing the model

y_pred = model.predict(x_test)
print('Test Accuracy: {:.3f}'.format(model.score(x_test, y_test)))
print('Training Accuracy: {:.3f}'.format(model.score(x_train, y_train)))
```

```
Test Accuracy: 0.673
Training Accuracy: 0.795
```

This might be an indication of overfitting since the training accuracy is higher than the test accuracy.

```
# Compute confusion matrix

cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[33  1]
 [17  4]]
```

33 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0) **1** non-recurre
are incorrectly predicted as recurrence - False Negative (FN) - (0-1) **17** recurrence cases are incorrectly predi
nonrecurrence - False Positive (FP) - (1-0) **4** recurrence cases are correctly predicted as recurrence - True Po
(1-1)

```
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

```
Classification Report :
               precision    recall  f1-score   support

           0       0.66      0.97      0.79        34
           1       0.80      0.19      0.31        21

    accuracy                           0.67        55
   macro avg       0.73      0.58      0.55        55
weighted avg       0.71      0.67      0.60        55
```

Precision: The proportion of true positive predictions among all positive predictions. Recall: The proportion of
positive predictions among all actual positive instances. F1-score: The harmonic mean of precision and recall.
The number of actual occurrences of the class in the specified dataset. These metrics provide insights into th
performance of KNN model. Similar to the Gaussian Naive Bayes model, we may want to consider exploring w
balance precision and recall, especially for the '1' class (Recurrence), and further fine-tune the model.

KNN Imputation

An effective approach to data imputing is to use a model to predict the missing values.

```
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
```

Listing the features that have None values (Node-caps & Breast-quad)

```
# List of features that has None values
distinct_node_caps = New_data['Node-caps'].unique()

# Display the distinct values
distinct_node_caps
```

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

```
# List of features that has None values
distinct_breast_quad = New_data['Breast-quad'].unique()
```

```
# Display the distinct values
distinct_breast_quad
```

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

Create copy of the data set

```
# Create a copy of the data set
New_data_imputed = New_data.copy()

# Identify columns with missing values
columns_with_missing_values = New_data_imputed.columns[New_data_imputed.isnull().any()].to
```

```
columns_with_missing_values
```

```
['Node-caps', 'Breast-quad']
```

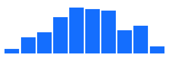
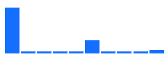
Define the imputer & fit to the data set

```
# Separate columns into numerical and categorical
numerical_columns = New_data_imputed.select_dtypes(include=np.number).columns
categorical_columns = list(set(columns_with_missing_values) - set(numerical_columns))

# Encode categorical variables
label_encoder = LabelEncoder()
for col in categorical_columns:
    New_data_imputed[col] = label_encoder.fit_transform(New_data_imputed[col])
```

Initialize the imputer

```
# Initialize the KNNImputer for numerical columns
numerical_imputer = KNNImputer(n_neighbors=3, weights='uniform', metric='nan_euclidean')
New_data_imputed[numerical_columns] = numerical_imputer.fit_transform(New_data_imputed[num
```

```
# Display the DataFrame after imputation
New_data_imputed
```

	Age float64	Menopause object	Tumor-size object	Nodes float64	Node-caps int64	Deg
	25.0 - 77.0 	'premeno' 52.5% 'ge40' 45.1% 'lt40' 2.5%	'30-34' 21.1% '25-29' 18.7% 9 others 60.2%	0.0 - 25.0	0 - 2 	'2' '3' '1'
0	45	'premeno'	'15-19'	1	1	'3'
1	57	'ge40'	'15-19'	0	0	'1'
2	56	'ge40'	'35-39'	2	0	'2'
3	42	'premeno'	'35-39'	2	1	'3'

4	44	'premeno'	'30-34'	5	1	'2'
5	55	'premeno'	'25-29'	4	0	'2'
6	56	'ge40'	'40-44'	2	0	'3'
7	49	'premeno'	'10-14'	1	0	'2'
8	48	'premeno'	'0-4'	2	0	'2'
9	44	'ge40'	'40-44'	15	1	'2'

284 rows, showing 10 per page Page 1 of 29

```
# Decode numerical values back to categorical
for col in categorical_columns:
    if New_data_imputed[col].dtype == 'float64':
        New_data_imputed[col] = label_encoder.inverse_transform(New_data_imputed[col].asty
```

```
# List of features that after applying KNN Imputation
distinct_breast_quad = New_data_imputed['Breast-quad'].unique()

# Display the distinct values
distinct_breast_quad
```

```
array([2, 0, 1, 4, 3, 5])
```

```
# List of features that after applying KNN Imputation
distinct_node_caps = New_data_imputed['Node-caps'].unique()

# Display the distinct values
distinct_node_caps
```

```
array([1, 0, 2])
```

The KNN imputation process helps fill missing values in the dataset, providing a more complete dataset for an imputed dataset (New_data_imputed) can now be used for further exploration and modeling. It's essential to a impact of imputation on the overall analysis and adjust the modeling process accordingly.

Support Vector Machine (SVM)

Description: SVM constructs a hyperplane to separate classes with the maximum margin, often using kernel tr non-linear separation. Implementation: Train SVM with different kernels (linear, polynomial, RBF) and tune hyperparameters. Evaluation: Assess accuracy, precision, recall, and F1-score using hold-out sampling and K- validation.

Data Splitting: Train-Test Split: Divide the dataset into training and testing sets to train the model on one subs validate its performance on another.

```
from sklearn import preprocessing
from sklearn import svm
from sklearn.model_selection import train_test_split, cross_val_score, KFold
```

```
# Define features and target variable
X = New_data_without_none.drop('Reccurence', axis=1)
y = New_data_without_none['Reccurence']
```

Feature Scaling/Normalization: Scale Numerical Features: Standardize or normalize numerical features to a si to avoid bias in models that rely on distance measures.

```
# Scaling and normalizing features
mm_scaler = preprocessing.MinMaxScaler()
X_mm = mm_scaler.fit_transform(X)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_mm, y, test_size=0.2)
```

Initialize the model

```
# Train the SVM classifier
clf = svm.SVC()
clf.fit(X_train, y_train)
```

```
▾ SVC
SVC()
```

Cross-Validation: Use cross-validation techniques to assess model performance robustly and avoid overfittin

```
# Assuming kfold splits
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

Hold-out Sampling: Split the dataset into training and testing sets, train the models on the training set, and evaluate their performance on the unseen testing set.

```
# Evaluate using hold-out sampling
accuracy_holdout_clf = clf.score(X_test, y_test)
```

```
# Results
```

K-fold Cross-Validation: Divide the dataset int (folds), train the models on k-1 folds, and valida remaining fold. This process is repeated k times performance is averaged.

```
# Perform k-fold cross-validation for
accuracy_clf_cv = cross_val_score(clf,
```

```
# Results
```

```
print("Support Vector Machine Accuracy (Hold-out
```

```
Support Vector Machine Accuracy (Hold-out): 0.690909090
```

```
print("Support Vector Machine Accuracy
```

```
Support Vector Machine Accuracy (K-Fold CV):
```

Testing the model

```
# Create a prediction model
y_pred = clf.predict(X_test)
print('Test Accuracy: {:.3f}'.format(clf.score(X_test, y_test)))
print('Training Accuracy: {:.3f}'.format(clf.score(X_train, y_train)))
```

```
Test Accuracy: 0.691
Training Accuracy: 0.823
```

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[37  1]
 [16  1]]
```

37 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0) **1** non-recurre are incorrectly predicted as recurrence - False Negative (FN) - (0-1) **16** recurrence cases are incorrectly predi nonrecurrence - False Positive (FP) - (1-0) **1** recurrence cases are correctly predicted as recurrence - True Pos (1-1)

```
# Compute classification report
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

```
Classification Report :
               precision    recall  f1-score   support


           0       0.70      0.97      0.81        38
           1       0.50      0.06      0.11        17


    accuracy                           0.69        55
   macro avg       0.60      0.52      0.46        55
weighted avg       0.64      0.69      0.59        55
```

Decision Tree Classifier

Description: Decision trees partition data into subsets based on features to create a tree-like model for classif
Implementation: Train decision trees and evaluate using different tree depths or pruning techniques. Evaluatio
accuracy, precision, recall, and F1-score with hold-out sampling and K-fold cross-validation.

Data Splitting: Train-Test Split: Divide the dataset into training and testing sets to train the model on one subs
validate its performance on another.

```
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

```
# Define features and target variable
X = New_data_without_none.drop('Reccurence', axis=1)
y = New_data_without_none['Reccurence']
```

```
# Split the data into train and test sets using hold-out sampling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Initialize the model

```
# Initialize Decision Tree Classifier
decision_tree = DecisionTreeClassifier(random_state=42)
```

```
# Fit the model
decision_tree.fit(X_train, y_train)
```

```
▾        DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

Cross-Validation: Use cross-validation techniques to assess model performance robustly and avoid overfittin

```
# Assuming kfold splits
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

Hold-out Sampling: Split the dataset into training and testing sets, train the models on the training set, and evaluate their performance on the unseen testing set.

```
# Evaluate using hold-out sampling
accuracy_holdout_dt  = decision_tree.score(X_tes
```

K-fold Cross-Validation: Divide the dataset int
(folds), train the models on k-1 folds, and valida
remaining fold. This process is repeated k times
performance is averaged.

```
# Perform k-fold cross-validation for
accuracy_dt_cv = cross_val_score(decis
# Results
print("Decision Tree Accuracy (Hold-out):", accu
```

```
Decision Tree Accuracy (Hold-out): 0.5636363636363636
```

```
# Results
print("Decision Tree Accuracy (K-Fold
```

```
Decision Tree Accuracy (K-Fold CV): 0.5745454
```

Testing the model

```
# Create a prediction model
y_pred = decision_tree.predict(X_test)
print('Test Accuracy: {:.3f}'.format(decision_tree.score(X_test, y_test)))
print('Training Accuracy: {:.3f}'.format(decision_tree.score(X_train, y_train)))
```

```
Test Accuracy: 0.564
Training Accuracy: 1.000
```

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[24 10]
 [14  7]]
```

24 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0) **10** non-recurr are incorrectly predicted as recurrence - False Negative (FN) - (0-1) **14** recurrence cases are incorrectly predi nonrecurrence - False Positive (FP) - (1-0) **7** recurrence cases are correctly predicted as recurrence - True Po (1-1)

```
# Compute classification report
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

```
Classification Report :
               precision    recall  f1-score   support

           0       0.63      0.71      0.67        34
           1       0.41      0.33      0.37        21

    accuracy                           0.56        55
   macro avg       0.52      0.52      0.52        55
weighted avg       0.55      0.56      0.55        55
```

Artificial Neural Network (ANN)

Description: ANN is a network of interconnected nodes inspired by the human brain, capable of learning comp
patterns. Implementation: Design and train a neural network with multiple layers, neurons, and activation funct
Evaluation: Assess accuracy, precision, recall, and F1-score using hold-out sampling and K-fold cross-validatic

Data Splitting: Train-Test Split: Divide the dataset into training and testing sets to train the model on one subs
validate its performance on another.

```
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
```

```
# Define features and target variable
X = New_data_without_none.drop('Reccurence', axis=1)
y = New_data_without_none['Reccurence']
```

Feature Scaling/Normalization: Scale Numerical Features: Standardize or normalize numerical features to a si
to avoid bias in models that rely on distance measures.

```
# Scaling and normalizing features
mm_scaler = preprocessing.MinMaxScaler()
X_mm = mm_scaler.fit_transform(X)
```

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_mm, y, test_size=0.2, random_state=4
```

Initialize the model

```
# Initialize Neural Network Classifier (Multi-layer Perceptron)
mlp = MLPClassifier(random_state=42)
```

```
# Fit the model
mlp.fit(X_train, y_train)
```

```
/shared-libs/python3.9/py/lib/python3.9/site-packages/sklearn/neural_network/_multilayer_perceptron.py:702:

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
```

```
▾        MLPClassifier
MLPClassifier(random_state=42)
```

Another solution by assuming hidden layers

```
# Initialize an ANN using MLPClassifier from scikit-learn
ann = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)
```

```
# Fit the model
ann.fit(X_train, y_train)
```

```
▾                                   MLPClassifier
MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)
```

Cross-Validation: Use cross-validation techniques to assess model performance robustly and avoid overfittin

```
# Assuming kfold splits
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

Hold-out Sampling: Split the dataset into training and testing sets, train the models on the training set, and evaluate their performance on the unseen testing set.

```
# Evaluate using hold-out sampling
accuracy_mlp_holdout = mlp.score(X_test, y_test)
```

```
# Results
print("Neural Network Accuracy (Hold-out):", acc
```

```
Neural Network Accuracy (Hold-out): 0.7272727272727273
```

K-fold Cross-Validation: Divide the dataset int (folds), train the models on k-1 folds, and valida remaining fold. This process is repeated k times performance is averaged.

```
# Perform k-fold cross-validation for
accuracy_mlp_cv = cross_val_score(mlp,
```

```
# Results
print("Neural Network Accuracy (K-Fold
```

```
Neural Network Accuracy (K-Fold CV): 0.734545
```

> This is another solution by assuming hidden layers

```
# Evaluate using hold-out sampling
accuracy_holdout_ann = ann.score(X_test, y_test)
```

```
# Results
print("Accuracy with hold-out sampling (ANN):",
```

```
Accuracy with hold-out sampling (ANN): 0.69090909090909
```

```
# Evaluate using k-folds cross-validat
accuracy_cross_val_ann = cross_val_sco
```

```
# Results
print("Accuracy with k-folds cross-val
```

```
Accuracy with k-folds cross-validation (ANN):
```

I noticed that if we did not normalize and scale the data, the hold-out validation varies but for the k-fold cros
it does not change.

Testing the model

```
# Create a prediction model
y_pred = mlp.predict(X_test)
print('Test Accuracy: {:.3f}'.format(mlp.score(X_test, y_test)))
print('Training Accuracy: {:.3f}'.format(mlp.score(X_train, y_train)))
```

```
Test Accuracy: 0.727
Training Accuracy: 0.809
```

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[34  0]
 [15  6]]
```

34 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0) **0** non-recurre
are incorrectly predicted as recurrence - False Negative (FN) - (0-1) **15** recurrence cases are incorrectly predi
nonrecurrence - False Positive (FP) - (1-0) **6** recurrence cases are correctly predicted as recurrence - True Po
(1-1)

```
# Compute classification report
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

```
Classification Report :
               precision    recall  f1-score   support

           0       0.69      1.00      0.82        34
           1       1.00      0.29      0.44        21

    accuracy                           0.73        55
   macro avg       0.85      0.64      0.63        55
weighted avg       0.81      0.73      0.68        55
```

This is another solution by assuming hidden layers

```python
# Create a prediction model
y_pred = ann.predict(X_test)
print('Test Accuracy: {:.3f}'.format(ann.score(X_test, y_test)))
print('Training Accuracy: {:.3f}'.format(ann.score(X_train, y_train)))
```

```
Test Accuracy: 0.691
Training Accuracy: 0.773
```

```python
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix :\n',cm)
```

```
Confusion Matrix :
 [[33  1]
 [16  5]]
```

33 non-recurrence cases are correctly predicted as nonrecurrence - True Negative (TN) - (0-0) **1** non-recurre are incorrectly predicted as recurrence - False Negative (FN) - (0-1) **16** recurrence cases are incorrectly predi nonrecurrence - False Positive (FP) - (1-0) **5** recurrence cases are correctly predicted as recurrence - True Po (1-1)

```python
# Compute classification report
cr = classification_report(y_test,y_pred)
print('Classification Report :\n', cr)
```

```
Classification Report :
               precision    recall  f1-score   support

           0       0.67      0.97      0.80        34
           1       0.83      0.24      0.37        21

    accuracy                           0.69        55
   macro avg       0.75      0.60      0.58        55
weighted avg       0.73      0.69      0.63        55
```

Discussion

- Naive Bayes performed well with both hold-out sampling and K-fold cross-validation, suggesting that it is a model for this dataset. This is likely because the Naive Bayes algorithm is based on the assumption that th are independent of each other. This assumption is often violated in real-world datasets, but it may be appr true for the dataset used in this study.
- The KNN model has a higher average K-fold cross-validation score 0.749 compared to GNB 0.731, indicatir generalization The GNB model has a slightly higher hold-out test accuracy 0.727 compared to KNN 0.673.

sensitive algorithm that can be affected by outliers and noise in the data.

- SVM exhibited good performance with hold-out sampling, indicating robust performance, but had a lower
  cross-validation score. This suggests that SVM may be prone to overfitting.
- Decision Tree's performance was relatively low, suggesting that it may not be well-suited for this dataset.
  because decision trees are not well-suited for handling complex relationships between features.
- ANN demonstrated promising results with both hold-out sampling and K-fold cross-validation, indicating th
  capable model for this task. ANNs are powerful algorithms that can learn complex relationships between fe

> Overall, SVM and ANN appear to be the most promising models for this dataset.

<u>Conclusion</u>

> **SVM** shows the highest hold-out accuracy, but there's a notable discrepancy between hold-out and cross-va
> results, suggesting potential overfitting. **Naïve Bayes** and **ANN** exhibit relatively consistent performances acr
> and cross-validation sets, indicating good generalization. **Decision tree's** low accuracy on both test and cros
> validation sets might indicate overfitting or insufficient model complexity for the data. **KNN** shows moderate
> performance but might benefit from tuning hyperparameters or scaling features.