**1. Dataset**

The dataset that the research is working with is the “Cuban breast cancer” dataset. The dataset contains 23 columns and over 1600 rows. A portion of the dataset is still inefficient and requires further processing.

**1.1. Messy and Noisy Data**

Columns such as “menopause” and “agefirst” are unnecessarily using the object datatype and contains string values which can be represented using an integer. Some instance of the data uses the value “No” instead of an integer value to represent patients who have not experienced what the question asked. There are other columns with messy and noisy data that has duplicate values that represent the same meaning or unnecessary suffixes that are appended to the end of the data that can confuse the machine learning model.

A number and numbers on a white background

Description automatically generated

Figure Values of the "breastfeeding" column

**1.2. Instances with Multiple Values Assigned**

Columns such as “nrelbc” and “allergies” contain a few rows that are assigned multiple values by appending additional values to a string separated by a slash. Since the machine learning model may not recognize rows with multiple values assigned in that way, the machine learning model may see it as a completely new value instead of two values being assigned to an instance which could confuse the machine learning model.

A white background with black text

Description automatically generated

Figure Values of the "nrelbc" column

**1.3. Missing Data**

A few columns in the dataset contains instances of data that are missing. Some columns contain only a few missing data populating only around 10 instances of data while other populate more then 500 instances of data.

A screenshot of a computer

Description automatically generated

Figure Amount of missing values in each column

**2. Dataset Preprocessing**

To handle all the problems stated in the previous segment, the dataset was pre-processed before being utilized in any way. The relevant code used to preprocess the dataset can be found in the research’s [GitHub page](https://github.com/ArkCayne/Breast.Cancer.Research) on a file named “Breast Cancer Research Data Prep FINAL.ipynb”. The results of the data preprocessing can be found in a csv file in the “Datasets” folder of the same GitHub page with the name “PreprocessedData.csv”. For the raw unprocessed dataset, it can also be found in the same folder with the name “CubanDataset.csv”.

**2.1. Cleaning Messy and Noisy Data**

This segment will aim to clean messy and noisy data contained in columns such as: “menopause”, “agefirst”, “breastfeeding”, and “exercise”

**2.1.1. Menopause Column**

The “menopause” column contains the value “No” using the string datatype to represent the patient’s inexperience having a menopause. This “No” value can easily be replaced by an existing integer value -1 that can be used to represent the same meaning. Although an integer value of 0 was already used to represent the same meaning, for the purpose of this research, all instances of the value 0 will be replaced with -1 except for a few cases that will be discussed later.

A number with black numbers

Description automatically generated with medium confidence

Figure Values of the "menopause" column

Once the “No” values are replaced, the column will no longer be needed as an object datatype since all the values are now in integers and all string values have been replaced with its integer representation.

A close-up of a computer code

Description automatically generated

Figure Code responsible for processing "menopause"

**2.1.2. Agefirst Column**

The “agefirst” column has the exact same problem as the “menopause” column where the string value “No” is used inefficiently and the column is casted as an object datatype.

A number and numbers on a white background

Description automatically generated

Figure Values of the "agefirst" column

The solution to the problems of the “agefirst” column is also like the solution used in the “menopause” column.

A close-up of a white background

Description automatically generated

Figure Code responsible for processing "agefirst"

**2.1.3. Exercise Column**

The “exercise” column contains duplicate values that represent the same meaning. The values “No”, “NO”, and integer value 0 all represent the same meaning. An instance of data in the column can also have the string value “Diary”.



Figure Values of the "exercise" column

Instances of data that contains the value “No”, “NO”, and integer value 0 will be replaced with integer value -1 to ensure uniformity with previous processed data. Instances of data that contains the string value “Diary” will instead be replaced with 0. Once all string values are replaced with an integer value, the column will be casted as an integer datatype.

A white background with text

Description automatically generated

Figure Code responsible for processing "exercise"

**2.1.4. Breastfeeding Column**

The “breastfeeding” column also has the same problems as the “exercise” column where there are duplicate values representing the same meaning in the form of string values “No”, “No ”, and integer value 0. In addition to that, some instances of data in the column contains unnecessary suffixes such as “-month” or “-months” appended to the values.

A number of months and months

Description automatically generated

Figure Values of the "breastfeeding" column

The string values “No”, “No “, and integer value 0 will be replaced with integer value -1. Unnecessary suffixes will be removed using string replace functions alongside regex to help format the values. Once all the string values are replaced and all suffixes are removed, the column can be casted as an integer datatype.

A close-up of a computer screen

Description automatically generated

Figure Code responsible for processing "breastfeeding"

**2.2. Handling Data with Multiple Values Assigned**

This segment will aim to improve the data representation for columns such as: “nrelbc” and “allergies”. Some values within these columns are assigned multiple values by appending more string values to the previous value separated by a slash. Since machine learning models will identify each instance of data in this form as its own unique value, this will negatively impact the ability of the machine learning model to predict labels. To solve this, each unique value available will get their own binary column to help identify what values are assigned to each instance of data.

**2.2.1. Nrelbc Column**

The “nrelbc” column is used to represent which relative or family member of the patient has previously suffered from breast cancer. The original state of the dataset still contains many instances of data with multiple values. Values such as “Sister/Grandmother” or “Aunt/Cousin” are some examples of instances of data that has multiple values assigned.

A white background with black text

Description automatically generated

Figure Values of the "nrelbc" column

Before creating a binary column, all available unique values must be identified first. The unique values in the “nrelbc” column are: "Mother", "Sister", "Daughter", "Cousin", "Aunt", "Grandmother", and "No". Then, create a binary column for each unique value available. To help identify whether each instance of data should have the value “True” or “False” for each binary column, use a loop that iterates through all available values and string functions that checks if a given string has another string within it. After all instances of data have been assigned a “True” or “False” value, drop the original column.

A screenshot of a computer code

Description automatically generated

Figure Code responsible for processing "nrelbc"

**2.2.2. Allergies Column**

The “allergies” column has the same problem as the “nrelbc” column. However, before creating the binary columns for the “allergies” column, some missing data needs to be handled first. According to figure 3, the “allergies” column has about 276 missing data. Since this dataset is dealing with the patient’s allergies, it can be assumed that any missing data are patients with no allergies. To handle this, all missing data will be replaced with the string value “No”.

A white background with black text

Description automatically generated

Figure Code responsible for handling missing data in "allergies"

Once all missing data is handled with, the column should only have values that are significant to the dataset. The unique values in the “allergies” column are: "Rhinitis", "Medicines", "Laryngitis", "Dermatitis", "Other", and "No".

A close up of text

Description automatically generated

Figure Values in the "allergies" column

Using the unique values, create a binary column for each value and assign a “True” or “False” value using loops and string functions. After the “True” or “False” values are assigned, the original column can be dropped.

A screenshot of a computer screen

Description automatically generated

Figure Code responsible for processing "allergies"

**2.2.3. Renaming Resulting Binary Columns**

To improve readability for the readers, it is recommended that each of the resulting binary columns be renamed to a more appropriate column name. In this case, the columns are renamed using the following code.

A screenshot of a computer code

Description automatically generated A screenshot of a computer code

Description automatically generated

Figure Code for renaming columns

**2.3. Handling Missing Data with Low Significance**

Columns such as “year”, “biopsies”, “imc”, and “weight” all have missing values. According to figure 3, the “year” column has about 537 missing data, but for the sake of this research, the “year” column will be deemed as low significance. Since the data inside the “year” column isn’t as significant, all missing data in the “year” column will be replaced with the median of all the values in the “year” column.

A close-up of a computer screen

Description automatically generated

Figure Code responsible for processing "year"

Missing data in the “biopsies”, “imc”, and “weight” columns might possess crucial details that can be helpful for the machine learning model, but due to the few amount of missing data in these columns, rows that has missing data in these columns can be removed safely.



Figure Code responsible for removing missing data

In the code above, it should be noted that the columns “histologicalclass” and “birads” are exempted from the columns that are checked for missing data because these columns not only have a large amount of missing data, but the values can also be crucial to the machine learning model. These processing of these two columns will be handled in the next segment.

**2.4. Handling Missing Values by Imputation**

This segment aims to process the “histologicalclass” and “birads” column. These two columns possess a significant amount of missing data that can be crucial in training the machine learning model. This is why handling the missing data in these two columns are different to handling the missing data in the other columns. For imputation, the random forest regressor will be used to assign proper for the missing data.

**2.4.1. Preparing Data for Imputation**

Before imputation can be done, the dataset needs to be prepared first. All categorical data needs to be encoded beforehand. For this case, one hot encoding will be used to encode all the data except for the “birads” column which will be the target of the imputation later.

A screen shot of a computer code

Description automatically generated

Figure Code responsible for encoding data

This way of doing one hot encoding creates a new dataframe that has a certain number of binary columns in them depending on the unique values in each categorical column. All the new dataframe and the “birads” column will need to be rejoined back with the main dataframe.

A screen shot of a computer code

Description automatically generated

Figure Code responsible for rejoining all relevant dataframes

**2.4.2. Histologicalclass Column**

At this point in time while trying to handle the missing values in “histologicalclass”, the “birads” column still has missing values, which means that it isn’t viable to be used as an argument for the random forest regressor. For this step of imputing values into the “histologicalclass” column, the “birads” column will be dropped.

****

Figure Code responsible for dropping "birads"

For random forest regressor to work, rows with missing “histologicalclass” values and rows without it needs to be separated. They will then be used as training and test sets for the random forest regressor.

**A computer code with numbers

Description automatically generated with medium confidence**

Figure Code responsible for splitting the data

Once the dataset has been split properly, the random forest regressor model can be trained with it and used to predict the missing values in the test set.

A computer screen shot of text

Description automatically generated

Figure Code responsible for training

****

Figure Code responsible for predicting values

Since the results of the predictions come in the form of float type values. The results will be rounded to the nearest whole number.



Figure Code responsible for rounding results

The values of the results will then be assigned to the test set, and the test set will be joined with the train set to create the whole dataset. The previously left out “birads” column will also be readded in this step. Since the data will be unsorted when joined, the “id” column will be used to help sort the data in ascending order based on their ID.



Figure Code responsible for assigning value to the test set

A screenshot of a computer code

Description automatically generated

Figure Code responsible for rejoining all dataframes

**2.4.3. Birads Column**

Since the “histologicalclass” column has been processed and no more missing values can be found in that column, unlike when processing the “histologicalclass” column where we exclude the “birads” column, all the columns in the dataframe can be utilized to help impute the values of the “birads” column.

One additional step needs to be done before taking all the other steps needed to impute the missing values. The “birads” column is currently assigned as an object datatype. Each unique values of the “birads” column will need to be mapped out to an integer value for the random forest regressor to work.

A screen shot of a computer

Description automatically generated

Figure Code responsible for mapping "birads" value

Like the previous segment, once all the categorical values have been mapped to an integer value, the data will need to be split between rows with missing “birads” value and rows with complete data.

A close up of words

Description automatically generated

Figure Code responsible for splitting the data

Once the data has been split into their own dataframes, the random forest regressor model can be trained and used to predict the missing values.

A screenshot of a computer code

Description automatically generated

Figure Code responsible for training



Figure Code responsible for predicting values

The random forest regressor will return its prediction using the float datatype. The values will need to be rounded to its nearest whole number.



Figure Code responsible for rounding

Once the values are rounded, the values will be assigned to the test set. The test set will then be joined with the train set and sorted using the values in the “id” column in ascending to order to form the final processed dataset.

A close-up of a math problem

Description automatically generated

Figure Code responsible for assigning value to the test set

A close up of text

Description automatically generated

Figure Code responsible for rejoining all dataframes

**3. Feature Selection**

A portion of this research will be using a version of the dataset that only has features that are significant to the task. To identify which features will be used for this portion of the research, feature selection will be done on the pre-processed dataset. The code that was used for feature selection can be found on the research’s [GitHub page](https://github.com/ArkCayne/Breast.Cancer.Research) with the name “Breast Cancer Research Feature Selection”.

For feature selection, [N-AMOUNT] methods will be utilized.

Reference: [Feature Selection in Python with Scikit-Learn - GeeksforGeeks](https://www.geeksforgeeks.org/feature-selection-in-python-with-scikit-learn/)

[Feature Selection in Python with Scikit-Learn - GeeksforGeeks](https://www.geeksforgeeks.org/feature-selection-in-python-with-scikit-learn/)

**4. Methods**

This segment will explain the choice of machine learning models that will be used in this research and what method of evaluation will be used.

**5. Training, Testing, and Evaluating the Models**

This segment will focus on training, testing, and evaluating the models whilst also making a comparison between all the models used that were being trained using which type of dataset.

**What models can be used for binary classification?**

Logistic regression, Naïve Bayes, Support Vector Machine (SVM), Decision Tree,  
K-Nearest Neighbors (KNN).

Logistic regression is the most common choice for binary classification.

[Model Summaries](https://medium.com/@karan.kamat1406/which-classification-model-should-you-use-a-cheat-sheet-for-machine-learning-practitioners-3fea0bcab04e)

**How can we evaluate these models?**

Accuracy, Precision, Recall, F1-Score, ROC Curve and AUC (Area under the ROC curve).