HW 3

Name: Alex Kapera
NetID: aak10157
N Number: 10693571

GitHub Link: https://github.com/OlekKapera/predicting-breast-cancer

1. Preprocessing

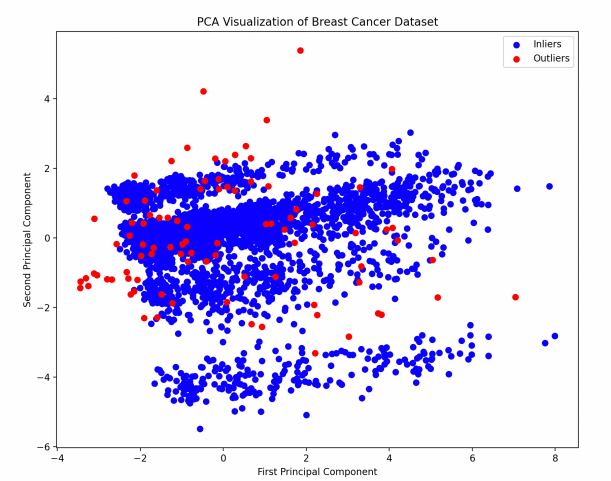
First, I displayed basic characteristics of data like min, max, mean values, data types and missing values count.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4024 entries, 0 to 4023
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Age                     3823 non-null   float64
 1   Race                    3622 non-null   object 
 2   Marital Status          3703 non-null   object 
 3   T Stage                 4024 non-null   object 
 4   N Stage                 4024 non-null   object 
 5   6th Stage               4024 non-null   object 
 6   differentiate           4024 non-null   object 
 7   Grade                   4024 non-null   object 
 8   A Stage                 4024 non-null   object 
 9   Tumor Size              3622 non-null   float64
 10  Estrogen Status         3823 non-null   object 
 11  Progesterone Status     4024 non-null   object 
 12  Regional Node Examined  3421 non-null   float64
 13  Reginol Node Positive   4024 non-null   int64  
 14  Survival Months         4024 non-null   int64  
 15  Status                  4024 non-null   object 
dtypes: float64(3), int64(2), object(11)
memory usage: 503.1+ KB
None

Basic Statistics:
               Age   Tumor Size  Regional Node Examined  Reginol Node Positive  Survival Months
count  3823.000000  3622.000000             3421.000000            4024.000000      4024.000000
mean     53.931467    30.560188               14.351652               4.158052        71.297962
std       8.972253    21.428535                8.170800               5.109331        22.921430
min      30.000000     1.000000                1.000000               1.000000         1.000000
25%      47.000000    16.000000                9.000000               1.000000        56.000000
50%      54.000000    25.000000               14.000000               2.000000        73.000000
75%      61.000000    38.000000               19.000000               5.000000        90.000000
max      69.000000   140.000000               61.000000              46.000000       107.000000

Missing Values Count:
Age                       201
Race                      402
Marital Status            321
T Stage                     0
N Stage                     0
6th Stage                   0
differentiate               0
Grade                       0
A Stage                     0
Tumor Size                402
Estrogen Status           201
Progesterone Status         0
Regional Node Examined    603
Reginol Node Positive       0
Survival Months             0
Status                      0
dtype: int64
```

Next, I proceeded with handling missing values using mean for numerical columns and mode for categorical columns. Before proceeding, I needed to transform categorical columns into numerical and for that I used OrdinalEncoding. Subsequently, I normalized the data using Z-score standardization and removed outliers using Local Outlier Factor method. This method removed 92 outliers. I then visualized the dataset by reducing the dimensionality to two using PCA:



2. Feature selection

To select the right features to train our models, I started with feature ranking using entropy. To be more specific I used sklearn's method called *mutual_info_classif.* The results showed a big drop in entropy for six columns which I decided to drop if their entropy was below 0.01.

```
→  HW3 /Users/aleksanderkapera/.pyenv/versions/3.12.1/bin/python "/Users/aleksanderkapera/School/Uni/NYU/Sem 3/PA/HW3/feature_selection.py"
{'Survival Months': 0.13734327726086693, '6th Stage': 0.033974630379879356, 'N Stage': 0.02631538034934988, 'Reginol Node Positive': 0.021565849
69569253, 'Progesterone Status': 0.017343852876732013, 'T Stage ': 0.017012253521944576, 'differentiate': 0.015369056889422472, 'Grade': 0.01320
0536090228976, 'Estrogen Status': 0.012456459393355157, 'Marital Status': 0.007726107514020919, 'Tumor Size': 0.005848183255553252, 'A Stage': 0
.005778163577640605, 'Age': 0.004611553289120218, 'Regional Node Examined': 0.0014706103187251962, 'Race': 0.0001994301537253662}

Features dropped due to low ranking:
['Marital Status', 'Tumor Size', 'A Stage', 'Age', 'Regional Node Examined', 'Race']

Remaining features: ['T Stage ', 'N Stage', '6th Stage', 'differentiate', 'Grade', 'Estrogen Status', 'Progesterone Status', 'Reginol Node Posit
ive', 'Survival Months', 'Status']
```

For feature selection, I experimented with forward and backward sequential feature selectors. Their results mostly overlapped and they only differed on three columns. I decided to stick with backward selection as it contained a feature that was individually ranked as the third most important, and the forward selector didn't select it.

3. Modeling

3.1. KNN

By implementing KNN, I achieved an accuracy of **87.95%**. I set k=3 based on a visual depiction of the dataset reduced to two dimensions where I visually identified 3 groups. The correctness of my KNN implementation I crosschecked with the one from sklearn library where I achieved similar results. Pros of KNN are that it's simple, understandable, and needs only one parameter (k) which can be easily optimized. Cons are that it's computationally intensive, and the performance degrades with the number of features (curse of dimensionality).

3.2. Naïve Bayes

Implementation with Naïve Bayes resulted in an accuracy of **81.49%.** The pros of Naïve Bayes are that it's simple and fast, can be trained on a low amount of training data, and performs well in multi-class classification. Cons are that it assumes all features are independent (in most cases they are not), Gaussian Naïve Bayes assumes normal distribution and may perform weakly if the data is distributed in an assumed way.

3.3. C4.5 Decision Tree

This model performed very well, scoring **86.96%** in terms of accuracy. Pros: because it's a decision tree, it's easy to visualize and explain the reasoning behind the prediction, and it can also handle both numerical and categorical data. Cons: it has tendencies to overfit, the decision tree can vary greatly when introduced to a new small set of training data, and tends to favor the majority class. Main hyperparameters: maximum depth of a tree.

3.4. Random Forest

This model achieved an accuracy of **87.06%.** Pros: because of multiple decision trees, this model is less susceptible to overfitting, it usually performs better than other forms of decision trees, and it doesn't bear any assumptions about data distribution. Cons: it is more computationally expensive (more trees to train), it's more difficult to interpret the data (many trees), and it is sensitive to hyperparameter tuning. Main hyperparameters: number of trees in a forest, maximum depth of a tree.

3.5. Gradient Boosting

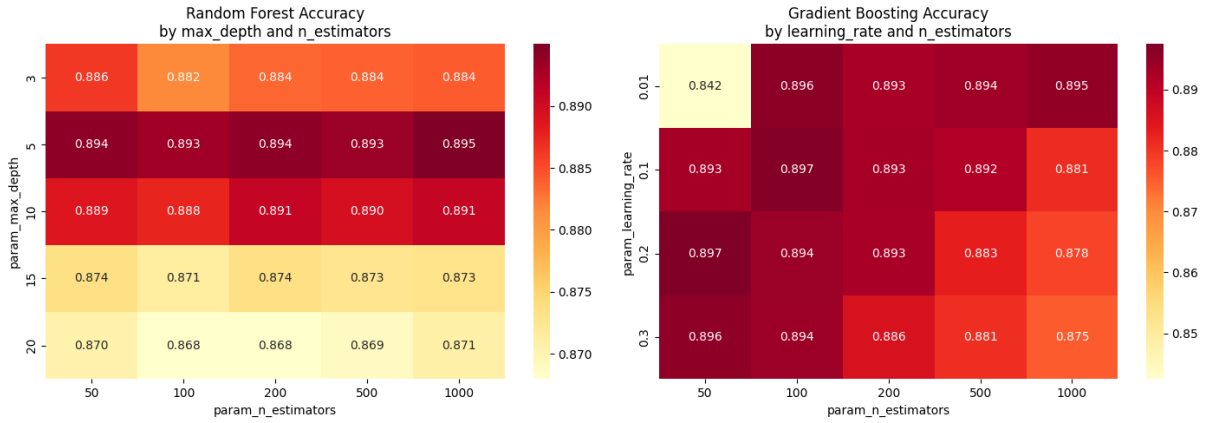
This model achieved an accuracy of **89.31%.** Pros: high performance, is flexible in terms of loss function or classification/regression. Cons: sensitive to hyperparameters, less interpretable, memory intensive. Main hyperparameters: number of stages (trees), learning rate, maximum depth of trees.

3.6. Neural Network

A simple neural network with one hidden layer containing 8 neurons achieved an accuracy of **89.81%.** Pros: flexible application, automatic feature selection, scalable. Cons: need large amounts of data to train, computationally intensive, requires many hyperparameters for tuning. Hyperparameters: network architecture (hidden layers and their sizes), learning rate, activation function, batch size.

4. Hyperparameter tuning

I chose to tune the hyperparameters of the Gradient boosting model as well as the Random forest. Below is the chart visualizing different values of hyperparameters and their respective performance. I used grid search to find the most optimal set of parameters. For Random forest, the hyperparameters that performed the best were maximum depth=5 and number of trees in the forest=1000 with accuracy=89.50%. In terms of Gradient Boosting, the best-performing combination of hyperparameters was learning rate=0.2 and number of gradient stages=50, achieving the accuracy of 89.75%.

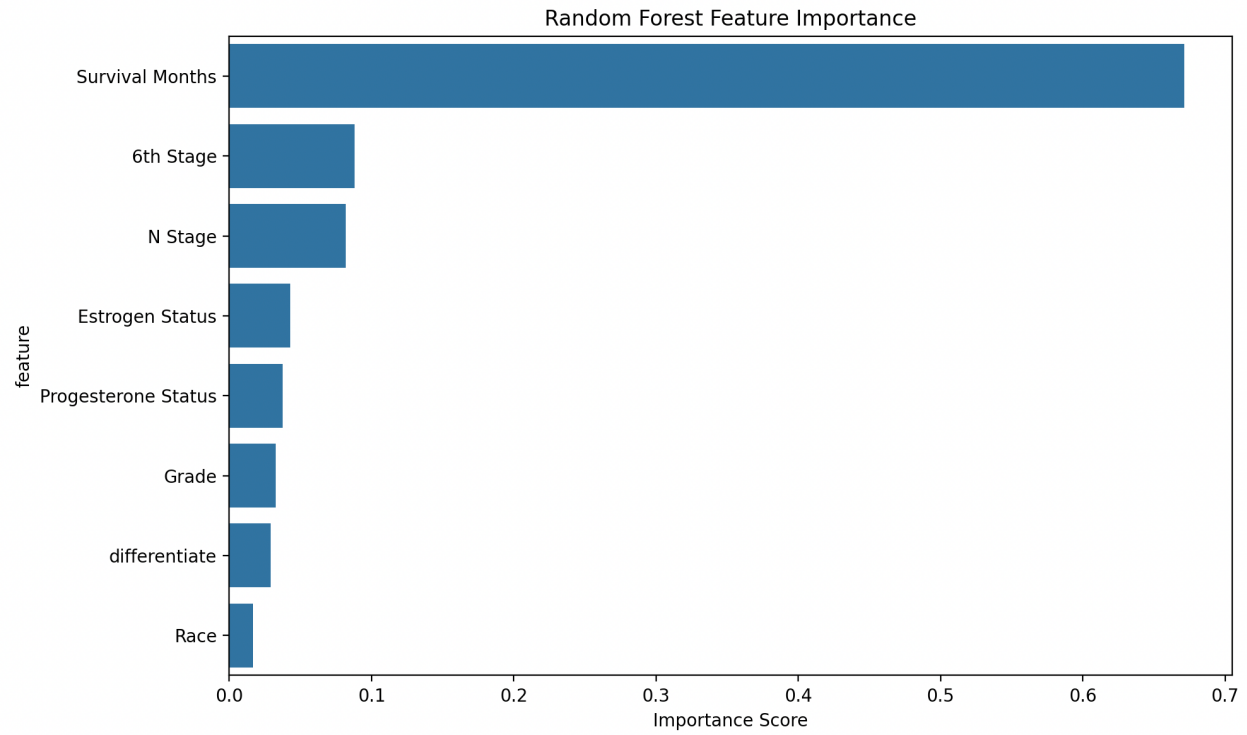


5. Results

The model that achieved the best accuracy score was neural network closely followed by, optimized gradient boosting and optimized random forest. The neural network model wasn't fine-tuned and, with more experimentation, is expected to yield even better results. Here is a table of all models:

Model	**Accuracy**
KNN	87.95%
Naïve Bayes	81.49%
C4.5 Decision Tree	86.96%
Random Forest (not fine-tuned)	87.06%
Random Forest (fine-tuned)	89.50%
Gradient Boosting (not fine-tuned)	89.31%
Gradient Boosting (fine-tuned)	89.75%
Neural network	89.81%

To determine the most important feature for the random forest, I used built-in sklearn's parameter called *feature_importances_*. The most influential parameter, by a big margin, was survival months.



I then followed with the same approach for gradient boosting and received similar results. However, this time, survival months' importance was even more prevalent.