Informatics Institute of Technology

Module: Machine Learning and Data Mining

Module Leader: Mr. Nipuna Senanayake

Final Analysis Report

Student Name: Sanuli Gehara Jayasekara

IIT ID: 20220855

UOW ID: w2053019

Tutorial Group: SE-09

Contents

Table of figures

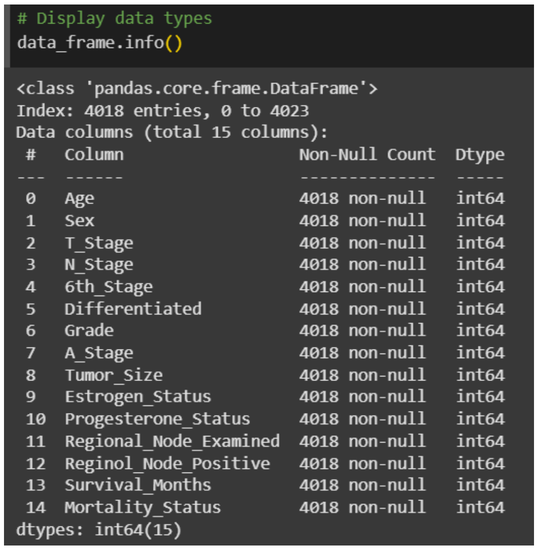
Case Study (A)

Task 1

Table 1. Retain or Drop table for classification Task

Variable	**RETAIN or DROP**	**justification for retention or dropping**
Patient ID	DROP	A distinct identifier with no prediction value
Month of Birth	DROP	No supportive evidence showing it influences cancer mortality. Keeping it might result in noise data
Age	REATIN	Age is a significant factor in survival month prediction
Sex	REATIN	Biological sex affects for the cancer behavior
Occupation	DROP	It has huge number of missing values and unnecessary for predicting breast cancer
T Stage	REATIN	The size of the tumor is a critical factor in the cancer prediction and survival rates
N Stage	REATIN	The lymph node is crucial for predicting breast cancer and mortality risk
6th Stage	REATIN	The stage grouping is needed to understand the severity of the cancer
Differentiated	REATIN	Required to model the difference between normal cells and cancer cells—more abnormally often means more aggressive.
Grade	REATIN	Another measure of tumor aggressiveness showing how abnormal tumor cells look
A Stage	REATIN	Decides how far it has spread
Tumor Size	REATIN	Exact tumor size in mm required for prediction
Estrogen Status	REATIN	Hormone receptor info affects treatment and chances of survival. Help predict how cancer behaves.
Progesterone Status	REATIN	Progesterone receptor status affects the survival of the cancer patient, so it is needed for predictions
Regional Node Examined	REATIN	For the count of examined regional lymph nodes to see the cancer spread
Regional Node Positive	REATIN	Show the cancerous lymph nodes
Survival Months	DROP	This is for the regression task later, not for classifying alive vs. dead.
Mortality Status	REATIN	Classification target variable which is significant to predict dead or alive mortality status

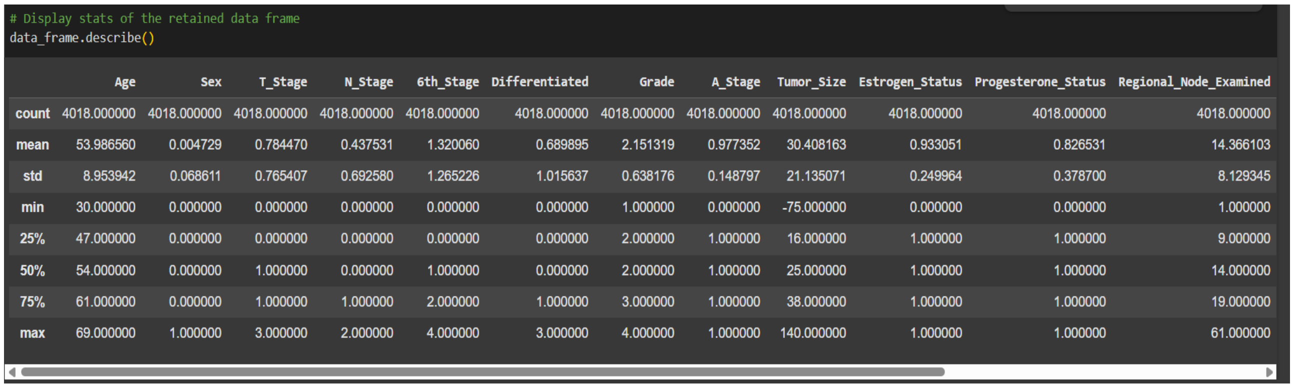
Task 2



```
# Display data types
data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4018 entries, 0 to 4023
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Age                     4018 non-null   int64
 1   Sex                     4018 non-null   int64
 2   T_Stage                 4018 non-null   int64
 3   N_Stage                 4018 non-null   int64
 4   6th_Stage               4018 non-null   int64
 5   Differentiated          4018 non-null   int64
 6   Grade                   4018 non-null   int64
 7   A_Stage                 4018 non-null   int64
 8   Tumor_Size              4018 non-null   int64
 9   Estrogen_Status         4018 non-null   int64
 10  Progesterone_Status     4018 non-null   int64
 11  Regional_Node_Examined  4018 non-null   int64
 12  Reginol_Node_Positive   4018 non-null   int64
 13  Survival_Months         4018 non-null   int64
 14  Mortality_Status        4018 non-null   int64
dtypes: int64(15)
```

Figure 2:Variable types



```
# Display stats of the retained data frame
data_frame.describe()
```

	Age	Sex	T_Stage	N_Stage	6th_Stage	Differentiated	Grade	A_Stage	Tumor_Size	Estrogen_Status	Progesterone_Status	Regional_Node_Examined
count	4018.000000	4018.000000	4018.000000	4018.000000	4018.000000	4018.000000	4018.000000	4018.000000	4018.000000	4018.000000	4018.000000	4018.000000
mean	53.986560	0.004729	0.784470	0.437531	1.320060	0.689895	2.151319	0.977352	30.408163	0.933051	0.826531	14.366103
std	8.953942	0.068611	0.765407	0.692580	1.265226	1.015637	0.638176	0.148797	21.135071	0.249964	0.378700	8.129345
min	30.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	-75.000000	0.000000	0.000000	1.000000
25%	47.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.000000	1.000000	16.000000	1.000000	1.000000	9.000000
50%	54.000000	0.000000	1.000000	0.000000	1.000000	0.000000	2.000000	1.000000	25.000000	1.000000	1.000000	14.000000
75%	61.000000	0.000000	1.000000	1.000000	2.000000	1.000000	3.000000	1.000000	38.000000	1.000000	1.000000	19.000000
max	69.000000	1.000000	3.000000	2.000000	4.000000	3.000000	4.000000	1.000000	140.000000	1.000000	1.000000	61.000000

Figure 1: Descriptive Statistics

Task 3

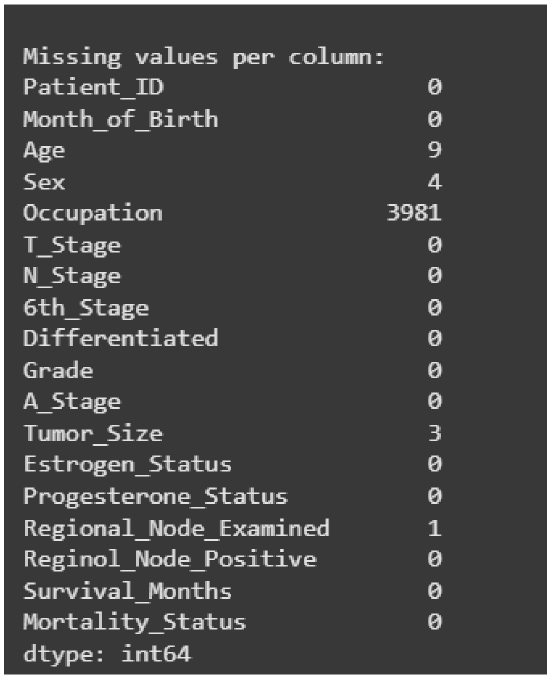
Here are the issues that I found in dataset and the solution used to fix it.

Table 2. Issues found and solutions

Variable Name	Issue Found	Proposed Fix	Justification for used fix method
Age	Identified 9 Missing values (NULL Values)	Simple Imputation (mean)	Age is a continuous variable, mean imputation maintains central tendency.
	Found Some Outliers	Removed the outliers.	Outliers can distort model performance. removal improves data quality.

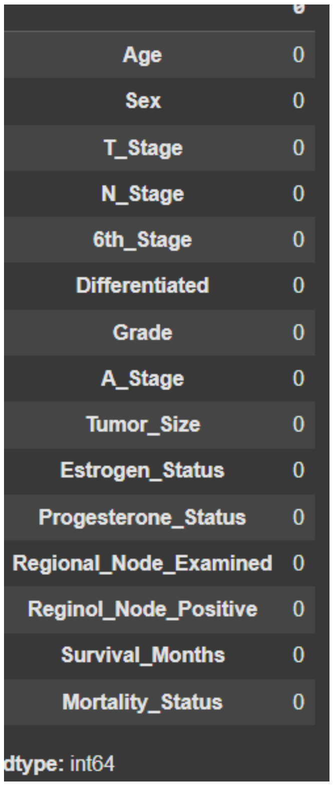
	Data type mismatch	Convert float type into int using astype()	Easier way to cast float into integer
Sex	Identified missing values (NULL Values) .	Simple Imputation (mode)	Sex is a categorical feature; mode imputation is good for categorical variables.
	Contains inconsistent values including strings and integers	replace values as 0 and 1 (0 – Female, 1 – Male)	Inconsistent values are not efficient for model performance in later stages
Tumor_Size	Identified 3 missing values .	Simple Imputation (mean)	Numeric variable. So, mean imputation used.
	Found Some Outliers.	Removed the outliers with IQR.	Found negative tumor size is invalid; outlier removed.
Regional Node Examined	Identified a missing Value	Simple Imputation (mean)	Median is robust to skewed distributions and suitable for count variables.
	Found Outliers.	Removed the outliers.	Outlier removal improves model reliability by reducing noise.
Mortality_Status	Contains inconsistent and different values	Mapping values to 0 and 1 (0 – Alive, 1 – dead)	Inconsistent values are not efficient for model performance in later stages

Problem: Missing values in variables – Before & After



```
Missing values per column:
Patient_ID                 0
Month_of_Birth             0
Age                        9
Sex                        4
Occupation              3981
T_Stage                    0
N_Stage                    0
6th_Stage                  0
Differentiated             0
Grade                      0
A_Stage                    0
Tumor_Size                 3
Estrogen_Status            0
Progesterone_Status        0
Regional_Node_Examined     1
Reginol_Node_Positive      0
Survival_Months            0
Mortality_Status           0
dtype: int64
```

Figure 3: Before-with missing values



```
                        0
Age                     0
Sex                     0
T_Stage                 0
N_Stage                 0
6th_Stage               0
Differentiated          0
Grade                   0
A_Stage                 0
Tumor_Size              0
Estrogen_Status         0
Progesterone_Status     0
Regional_Node_Examined  0
Reginol_Node_Positive   0
Survival_Months         0
Mortality_Status        0

dtype: int64
```

Figure 4: After-Imputing missing values

Outliers – Before & After in Age, Survival Months, Tumor Size variables

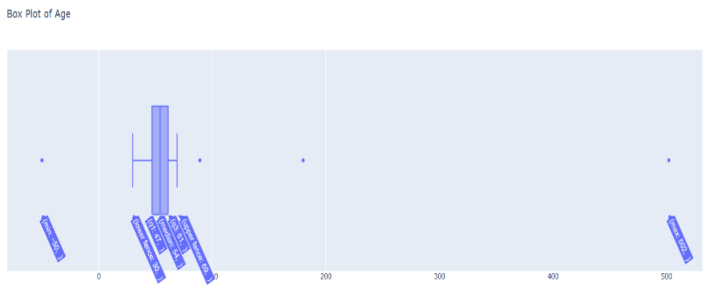


Figure 6: Age- with outliers

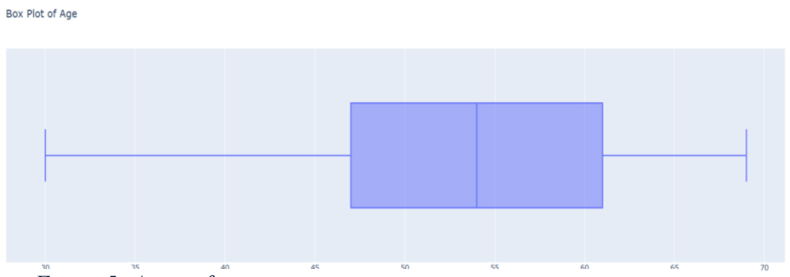


Figure 5: Age – after removing

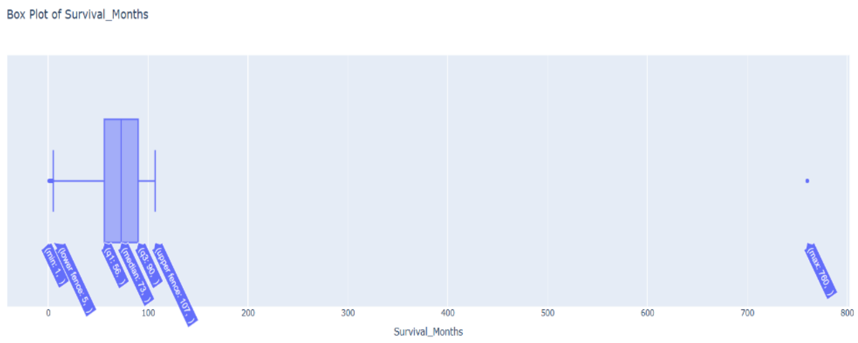


Figure 8: Survival months - with outliers

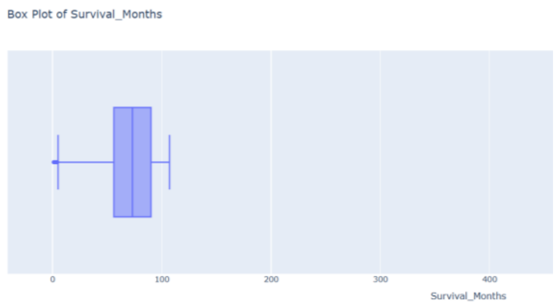
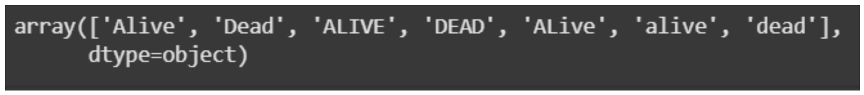


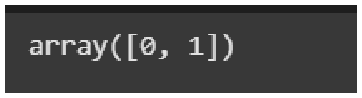
Figure 7: After removing

Inconsistent data values – Before and After



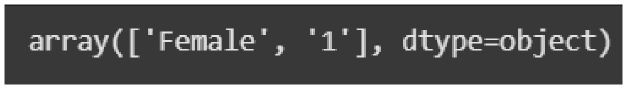
```
array(['Alive', 'Dead', 'ALIVE', 'DEAD', 'ALive', 'alive', 'dead'],
      dtype=object)
```

Figure 10: Survival month with inconsistent values



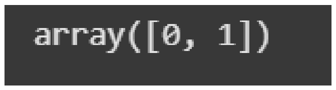
```
array([0, 1])
```

Figure 9: Survival month values after mapping into 1 - Dead, 0- Alive



```
array(['Female', '1'], dtype=object)
```

Figure 11: Sex with different values



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

Figure 12: After replacing Male - 1, Female - 0

Task (4)

a)

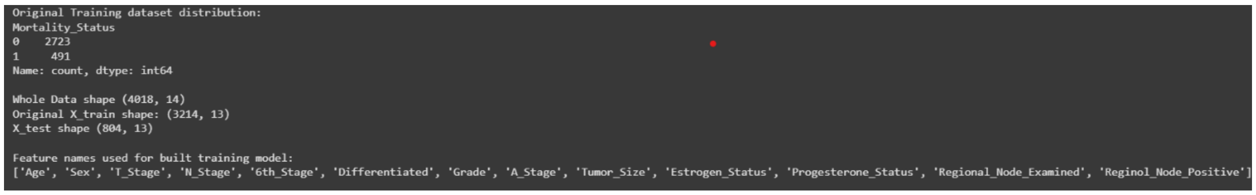
Table 3. classification algorithms summary

Algorithm Name	Algorithm Type	Learnable Parameters	Some Strategic Hyperparameters
Naïve Bayes	Parametric	Mean, and Variance	var_smoothing
Logistic Regression	Parametric	Coefficients, Intercept	C (inverse regularization strength), solver
K-Nearest Neighbors	Non-Parametric	None. (No training phase, lazy Learner)	Number of neighbors (n_neighbours), metric

b)

i.

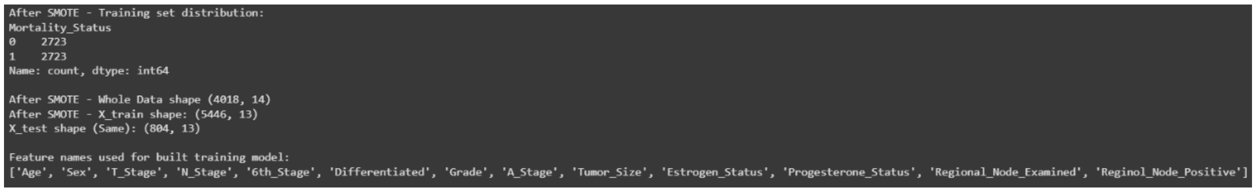
This is splitting on original dataset.



```
Original Training dataset distribution:
Mortality_Status
0    2723
1     491
Name: count, dtype: int64

Whole Data shape (4018, 14)
Original X_train shape: (3214, 13)
X_test shape (804, 13)

Feature names used for built training model:
['Age', 'Sex', 'T_Stage', 'N_Stage', '6th_Stage', 'Differentiated', 'Grade', 'A_Stage', 'Tumor_Size', 'Estrogen_Status', 'Progesterone_Status', 'Regional_Node_Examined', 'Reginol_Node_Positive']
```

Figure 13: training_testing_ratio_before_SMOTE

But since the data instances for mortality status (target variable) has a class imbalance issue (where alive as 2723 and dead as 491 records). So, I selected SMOTE (Synthetic Minority Over-sampling Technique) for balancing the training data. Unlike random oversampling (which just duplicates minority class examples), SMOTE creates synthetic examples by interpolating between real examples which makes it the best option.



```
After SMOTE - Training set distribution:
Mortality_Status
0    2723
1    2723
Name: count, dtype: int64

After SMOTE - Whole Data shape (4018, 14)
After SMOTE - X_train shape: (5446, 13)
X_test shape (Same): (804, 13)

Feature names used for built training model:
['Age', 'Sex', 'T_Stage', 'N_Stage', '6th_Stage', 'Differentiated', 'Grade', 'A_Stage', 'Tumor_Size', 'Estrogen_Status', 'Progesterone_Status', 'Regional_Node_Examined', 'Reginol_Node_Positive']
```

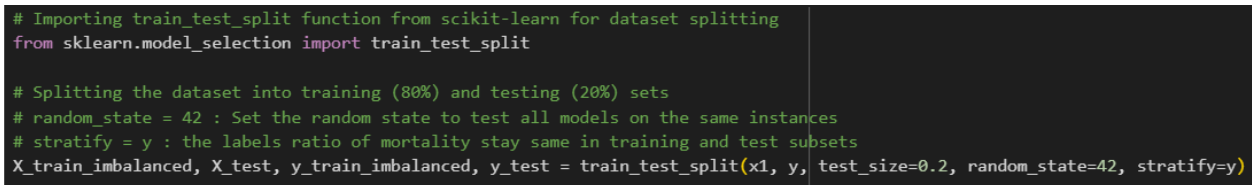
Figure 14: training_testing_ratio_after_SMOTE

ii.

In most supervised-learning problems, an 80/20 split is widely used because it gives a good balance between model training and reliable evaluation. For example, Jason Brownlee (2020) notes that an 80 %

training set “provides sufficient data for the model to learn patterns effectively,” while a 20 % hold-out “is large enough to give meaningful estimates of performance on unseen data”. Likewise, the scikit-learn documentation shows that common practice is to reserve 10–30 % of data for testing to guard against overfitting while ensuring the training set remains representative. In our imbalanced breast-cancer dataset, stratifying this 80/20 split preserves class ratios in both sets, ensuring the model sees enough minority-class examples during training and that our test metrics reflect real-world performance.

iii.



```
# Importing train_test_split function from scikit-learn for dataset splitting
from sklearn.model_selection import train_test_split

# Splitting the dataset into training (80%) and testing (20%) sets
# random_state = 42 : Set the random state to test all models on the same instances
# stratify = y : the labels ratio of mortality stay same in training and test subsets
X_train_imbalanced, X_test, y_train_imbalanced, y_test = train_test_split(x1, y, test_size=0.2, random_state=42, stratify=y)
```

Figure 15: Splitting dataset with ratio

Using stratify=y in train_test_split ensures that both training and test sets preserve the original “Alive” vs. “Dead” class proportions. This is crucial in imbalanced-class problems to avoid a test set with too few minority-class examples, which would yield unreliable evaluation (Fazzolini, 2016). Additionally, a fair, direct performance comparison without data leakage or shifting sample distributions is ensured by maintaining the same test instances across all models (via a fixed random_state) (scikit-learn developers, 2024). After stratified splitting, I applied SMOTE only on the training set to address class imbalance. This ensures the test set remains a realistic hold-out for unbiased evaluation.

Task (5)

a)

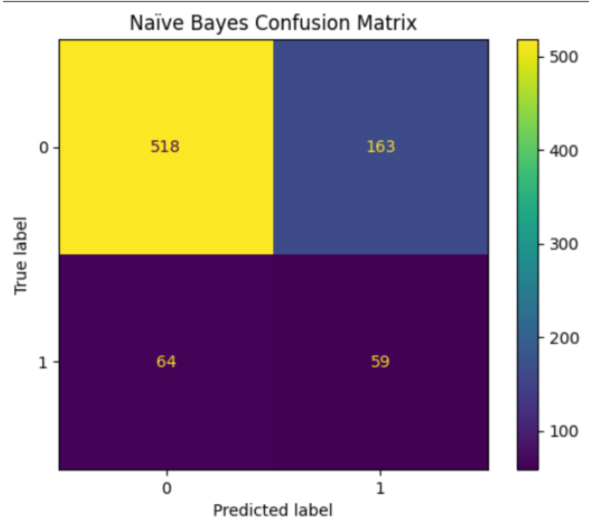


Figure 17: NB Confusion matrix

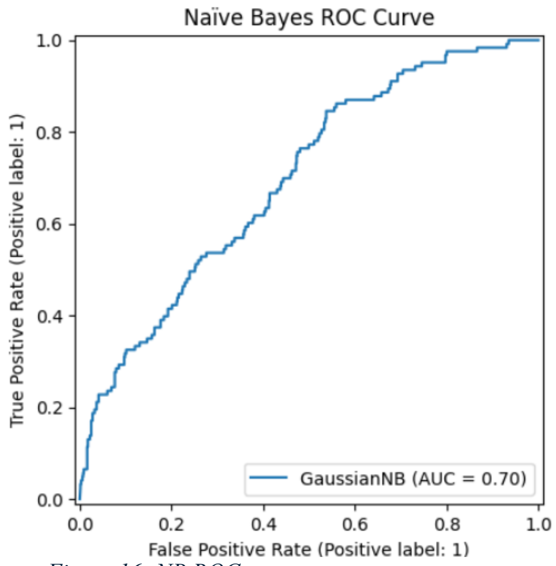
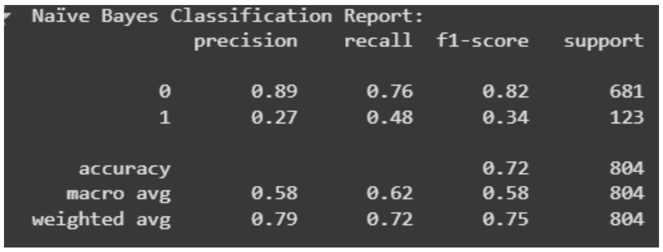


Figure 16: NB ROC curve



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

           0       0.89      0.76      0.82       681
           1       0.27      0.48      0.34       123

    accuracy                           0.72       804
   macro avg       0.58      0.62      0.58       804
weighted avg       0.79      0.72      0.75       804
```

Figure 18: NB Classification report

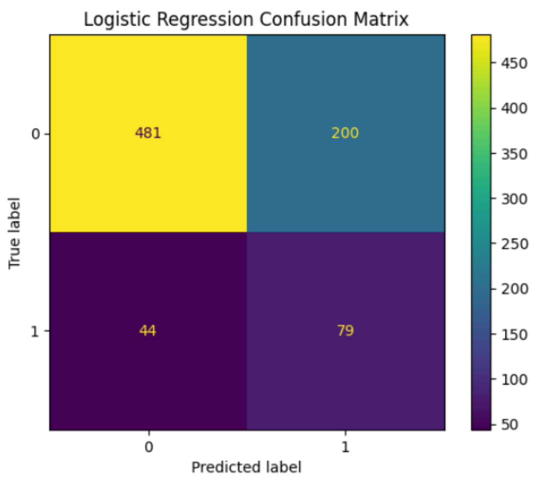


Figure 20: LR confusion matrix

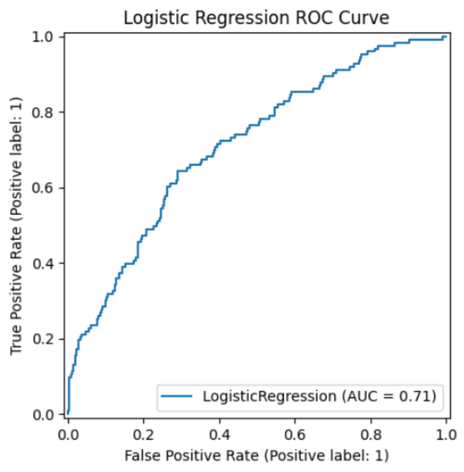
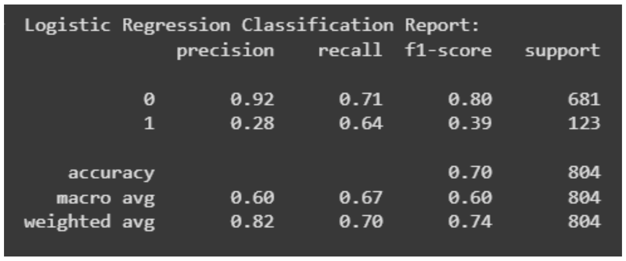


Figure 19: LR ROC curve



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

           0       0.92      0.71      0.80       681
           1       0.28      0.64      0.39       123

    accuracy                           0.70       804
   macro avg       0.60      0.67      0.60       804
weighted avg       0.82      0.70      0.74       804
```

Figure 21: LR classification report

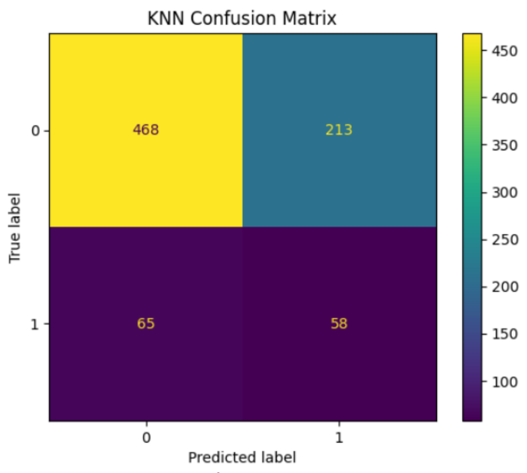


Figure 22: KNN confusion matrix

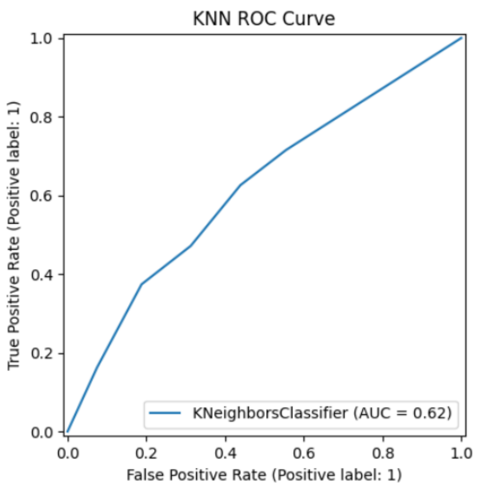
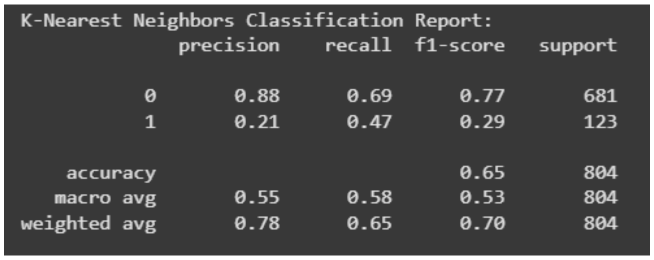


Figure 23: KNN ROC curve



```
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.69      0.77       681
           1       0.21      0.47      0.29       123

    accuracy                           0.65       804
   macro avg       0.55      0.58      0.53       804
weighted avg       0.78      0.65      0.70       804
```

Figure 24: KNN classification report

b)

Table 4: classification evaluation metrics

Metrics	USE or DO NOT USE	Justification	Model	Test Score	
Accuracy	DO NOT USE	With high Alive vs low Dead, high accuracy merely reflects majority-class bias and hides how well "Dead" are detected. So, it can be misleading in imbalanced datasets	NB	0.72	
			LR	0.70	
			KNN	0.65	
Recall	USE	Captures how many actual "Dead" or "Alive" patients truly are. It prevents over-treating or panicking healthy patients, so it is important.	NB	**Alive - 0**	**Dead - 1**
				0.76	0.48
			LR	0.71	0.64
			KNN	0.69	0.47
Precision	USE	Precision shows of the patients we flagged as "Dead," and "Alive", how many truly are. It prevents over-treating or panicking healthy patients, so it remains important.	NB	0.89	0.27
			LR	0.92	0.28
			KNN	0.88	0.21

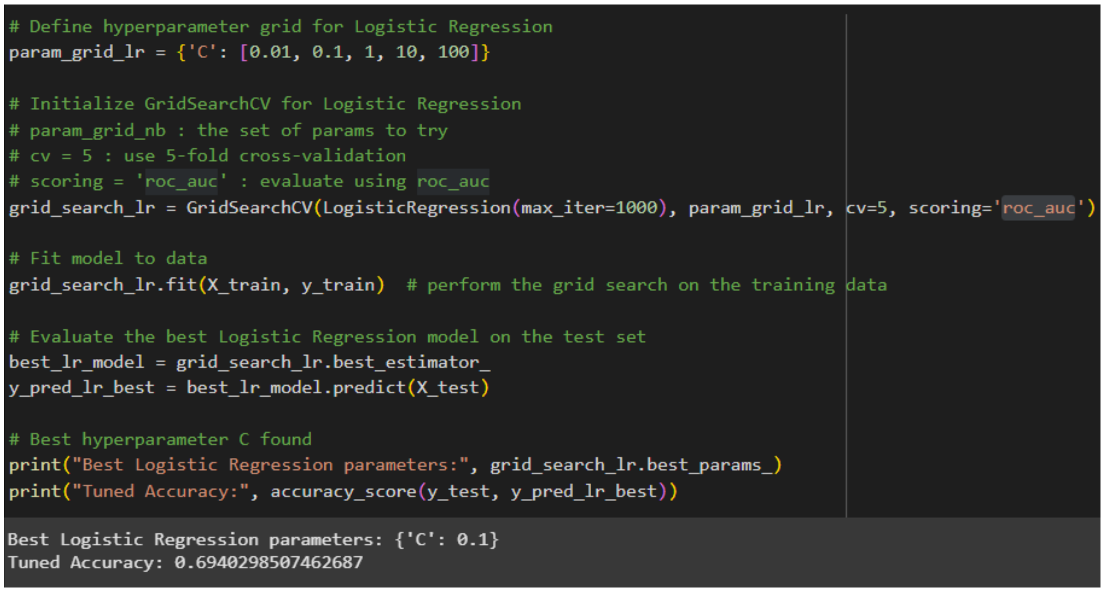
F1-Score	USE	F1 balances recall and precision, which is essential for cases with class imbalance and where both false negatives and false positives are serious.	NB	0.82	0.34
			LR	0.80	0.39
			KNN	0.77	0.29
AUC-ROC	USE	Measures overall discrimination (how well the model ranks Dead vs Alive) across all thresholds, robust under imbalance. ideal for measuring overall classification quality.	NB	0.70	
			LR	0.71	
			KNN	0.62	

c)

Based on the test metrics in Notebook 2, Logistic Regression is the top performer: it achieves the highest recall for “Dead” patients (0.64), the best F_1 score (0.39), and the strongest AUC-ROC (0.74). This means it catches more true “Dead” cases than Naïve Bayes or KNN while maintaining reasonable precision—exactly what clinicians need to flag high-risk patients early. We can further improve this model with hyperparameter tuning.

d)

i.



```
# Define hyperparameter grid for Logistic Regression
param_grid_lr = {'C': [0.01, 0.1, 1, 10, 100]}

# Initialize GridSearchCV for Logistic Regression
# param_grid_nb : the set of params to try
# cv = 5 : use 5-fold cross-validation
# scoring = 'roc_auc' : evaluate using roc_auc
grid_search_lr = GridSearchCV(LogisticRegression(max_iter=1000), param_grid_lr, cv=5, scoring='roc_auc')

# Fit model to data
grid_search_lr.fit(X_train, y_train)  # perform the grid search on the training data

# Evaluate the best Logistic Regression model on the test set
best_lr_model = grid_search_lr.best_estimator_
y_pred_lr_best = best_lr_model.predict(X_test)

# Best hyperparameter C found
print("Best Logistic Regression parameters:", grid_search_lr.best_params_)
print("Tuned Accuracy:", accuracy_score(y_test, y_pred_lr_best))
```

```
Best Logistic Regression parameters: {'C': 0.1}
Tuned Accuracy: 0.6940298507462687
```

Figure 25:hyperparameter tuning for lr

ii.

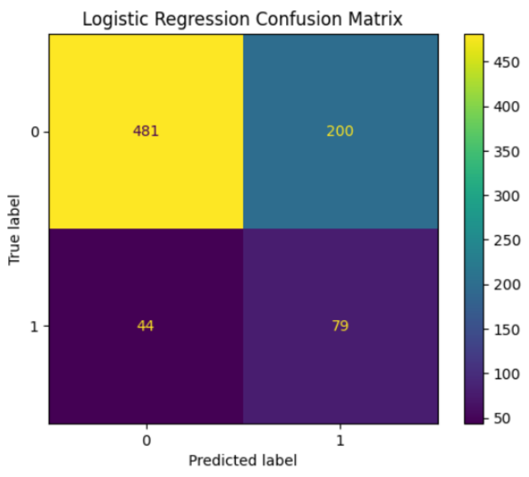


Figure 26: Before

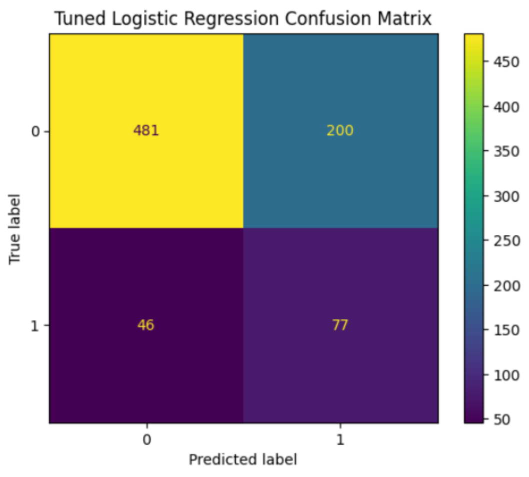


Figure 27: After

After applying hyperparameter tuning to the logistic regression model the performance metrices remain the same without any significant change. The best parameters found were {'C': 0.1}. Additionally, the confusion matrix and classification reports do not have any change for improvement after tuning.

e)

- Limitations

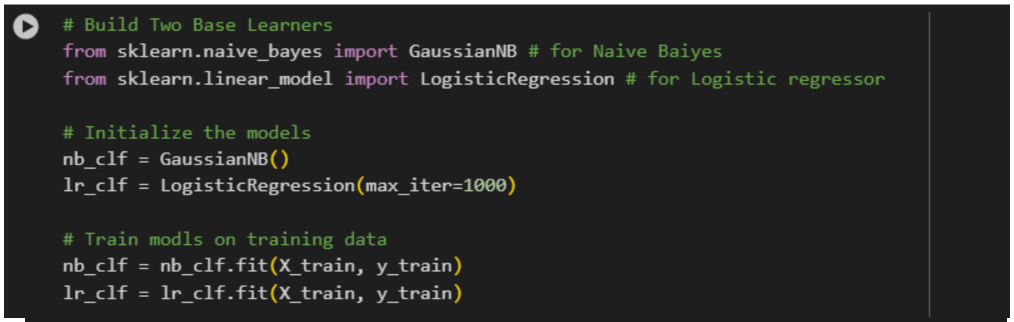
While logistic regression had a good performance in AUC-ROC and recall, it still misses about 36% of actual "Dead" cases and flags many "Alive" patients as high risk. Its straight-line decision boundary can't capture complex interactions between variables. The model also assumes that each feature acts independently, which isn't always true in medical data.

- Ethical Considerations

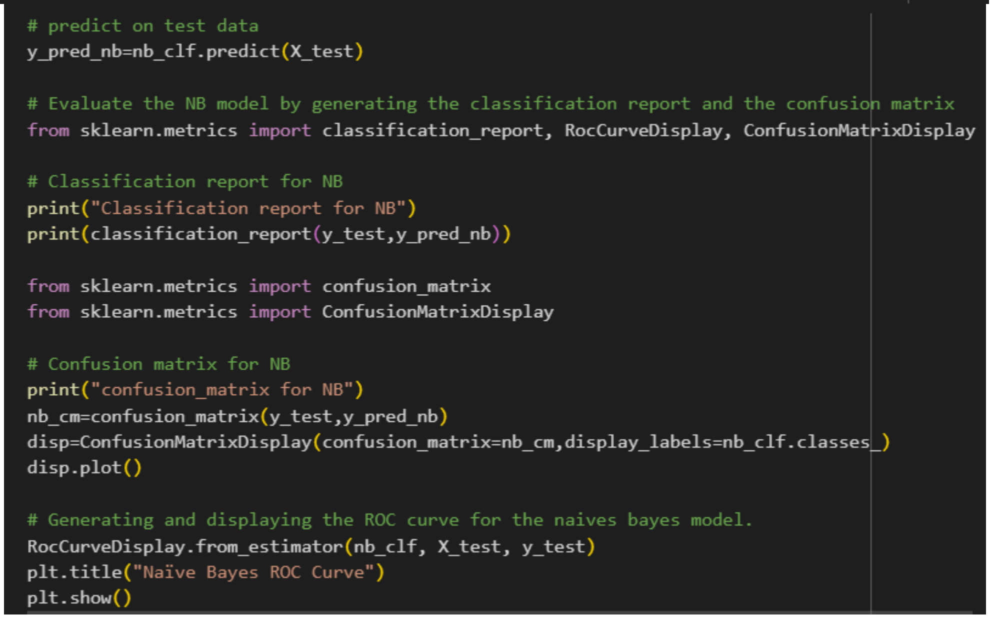
Using this model without human review could put patients' lives at risk if true high-risk cases are missed. False alarms may cause unnecessary worry or treatment. If certain populations were under-represented in the training data, the model could unfairly under- or over-predict risk for those groups.

f)

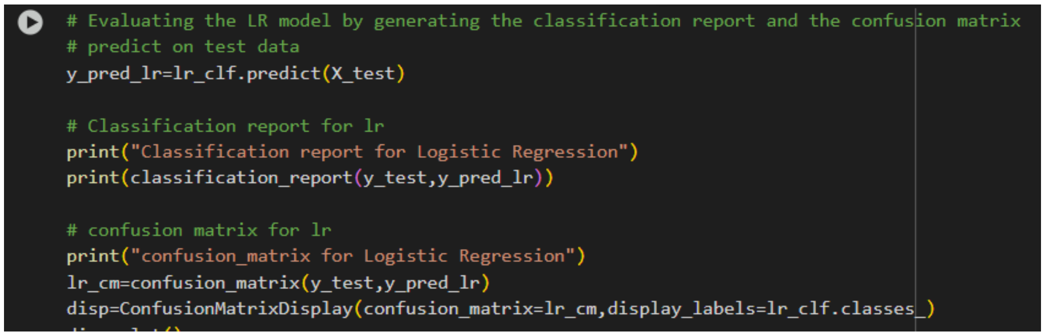
i.



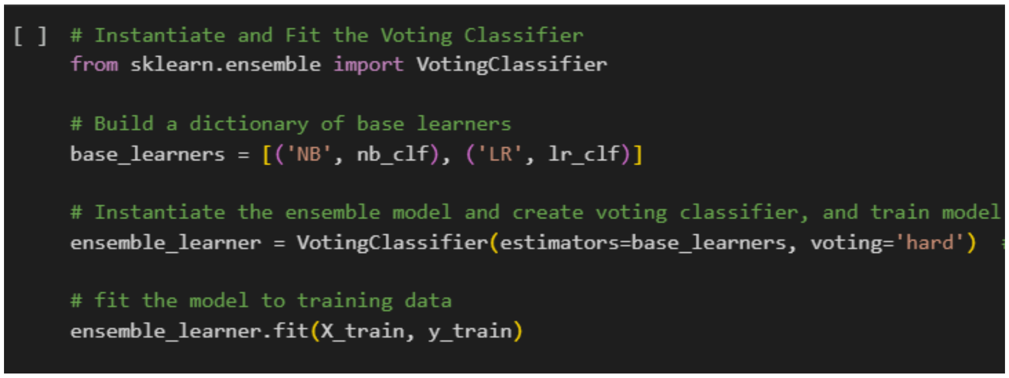
```
# Build Two Base Learners
from sklearn.naive_bayes import GaussianNB # for Naive Baiyes
from sklearn.linear_model import LogisticRegression # for Logistic regressor

# Initialize the models
nb_clf = GaussianNB()
lr_clf = LogisticRegression(max_iter=1000)

# Train modls on training data
nb_clf = nb_clf.fit(X_train, y_train)
lr_clf = lr_clf.fit(X_train, y_train)
```



```
# predict on test data
y_pred_nb=nb_clf.predict(X_test)

# Evaluate the NB model by generating the classification report and the confusion matrix
from sklearn.metrics import classification_report, RocCurveDisplay, ConfusionMatrixDisplay

# Classification report for NB
print("Classification report for NB")
print(classification_report(y_test,y_pred_nb))

from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

# Confusion matrix for NB
print("confusion_matrix for NB")
nb_cm=confusion_matrix(y_test,y_pred_nb)
disp=ConfusionMatrixDisplay(confusion_matrix=nb_cm,display_labels=nb_clf.classes_)
disp.plot()

# Generating and displaying the ROC curve for the naives bayes model.
RocCurveDisplay.from_estimator(nb_clf, X_test, y_test)
plt.title("Naïve Bayes ROC Curve")
plt.show()
```



```
# Evaluating the LR model by generating the classification report and the confusion matrix
# predict on test data
y_pred_lr=lr_clf.predict(X_test)

# Classification report for lr
print("Classification report for Logistic Regression")
print(classification_report(y_test,y_pred_lr))

# confusion matrix for lr
print("confusion_matrix for Logistic Regression")
lr_cm=confusion_matrix(y_test,y_pred_lr)
disp=ConfusionMatrixDisplay(confusion_matrix=lr_cm,display_labels=lr_clf.classes_)
```



```
[ ]  # Instantiate and Fit the Voting Classifier
     from sklearn.ensemble import VotingClassifier

     # Build a dictionary of base learners
     base_learners = [('NB', nb_clf), ('LR', lr_clf)]

     # Instantiate the ensemble model and create voting classifier, and train model
     ensemble_learner = VotingClassifier(estimators=base_learners, voting='hard')

     # fit the model to training data
     ensemble_learner.fit(X_train, y_train)
```

Figure 28: Best base learners, and fit ensemble learner

ii.

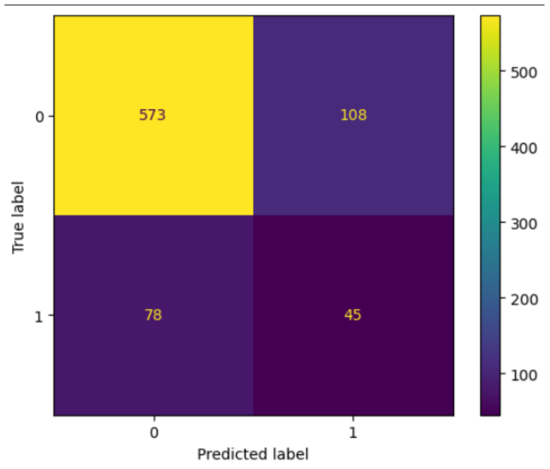


Figure 30: base learner NB - confusion matrix

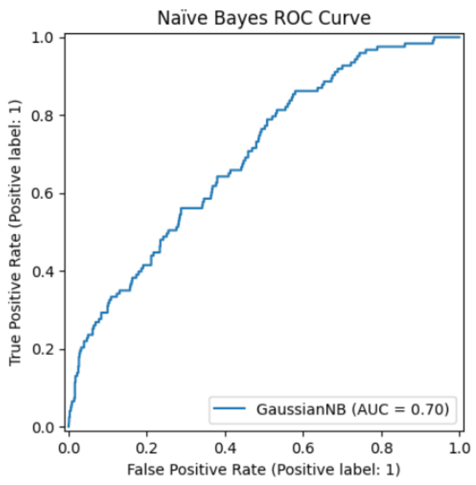
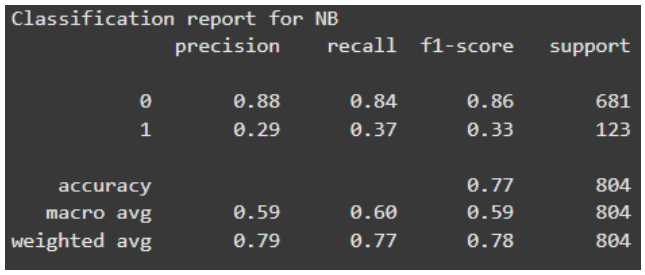


Figure 29:base learner NB - roc curve



```
Classification report for NB
              precision    recall  f1-score   support

           0       0.88      0.84      0.86       681
           1       0.29      0.37      0.33       123

    accuracy                           0.77       804
   macro avg       0.59      0.60      0.59       804
weighted avg       0.79      0.77      0.78       804
```

Figure 31: base learner NB - classification report

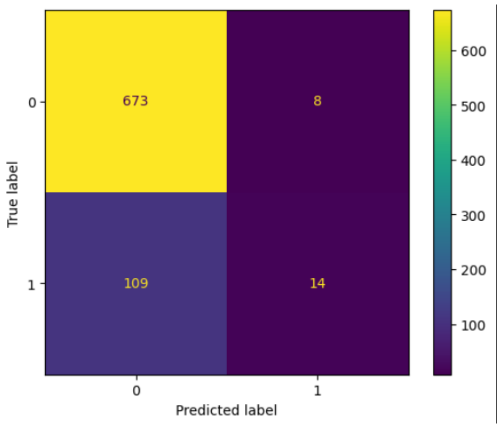


Figure 33: base learner LR - confusion matrix

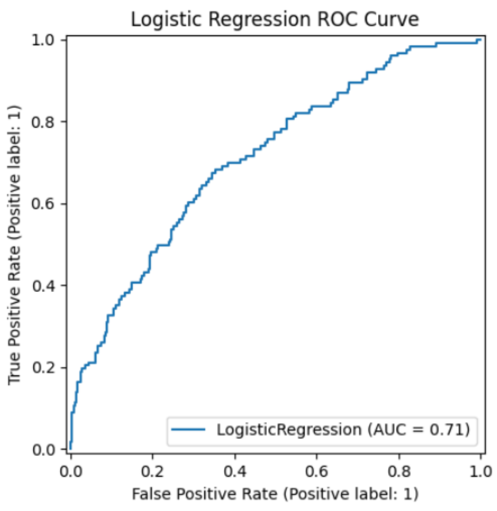
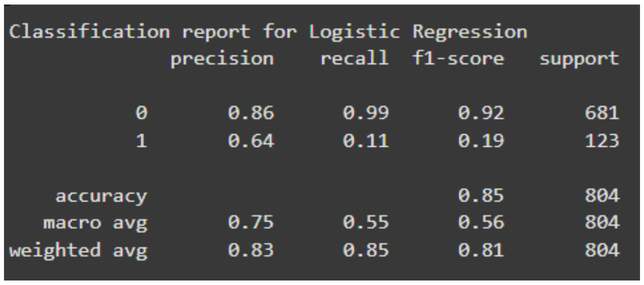


Figure 32: base learner LR - roc curve



```
Classification report for Logistic Regression
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       681
           1       0.64      0.11      0.19       123

    accuracy                           0.85       804
   macro avg       0.75      0.55      0.56       804
weighted avg       0.83      0.85      0.81       804
```

Figure 34: base learner LR - classification report

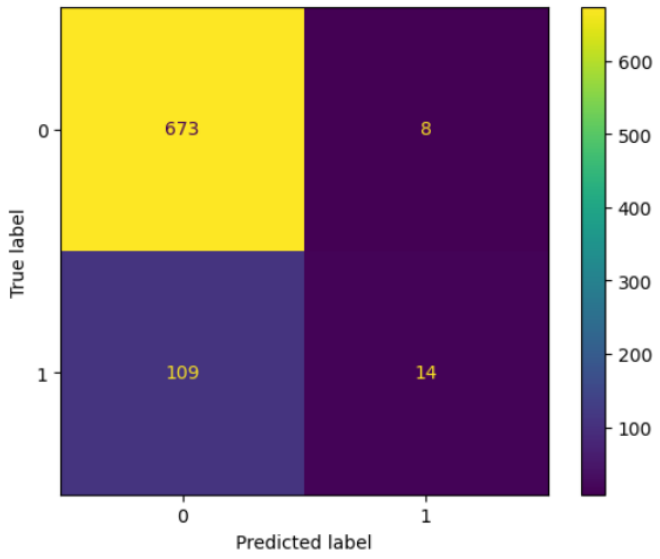
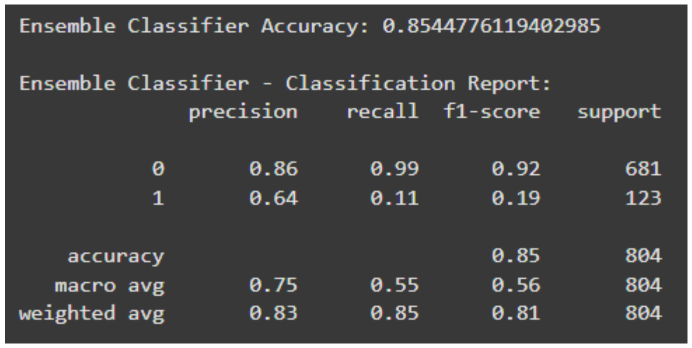


Figure 36: ensemble learner- confusion matrix


```
Ensemble Classifier Accuracy: 0.8544776119402985

Ensemble Classifier - Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       681
           1       0.64      0.11      0.19       123

    accuracy                           0.85       804
   macro avg       0.75      0.55      0.56       804
weighted avg       0.83      0.85      0.81       804
```
Figure 35: ensemble learner - classification report

- Justification

Logistic Regression and Naïve Bayes were paired because they bring different strengths—LR has a strong overall ranking ability and highest recall for "Dead" cases, while NB is a fast, probabilistic model that catches more true positives when LR misses them. Together, they offered the hope of balancing sensitivity and specificity.

ii.

- Logistic Regression (LR) - Recall$_1$ = 0.11, F$_{11}$ = 0.19
- Naïve Bayes (NB) - Recall$_1$ = 0.37, F$_{11}$ = 0.33
- Ensemble (LR + NB, soft voting) - Recall$_1$ = 0.11, F$_{11}$ = 0.19

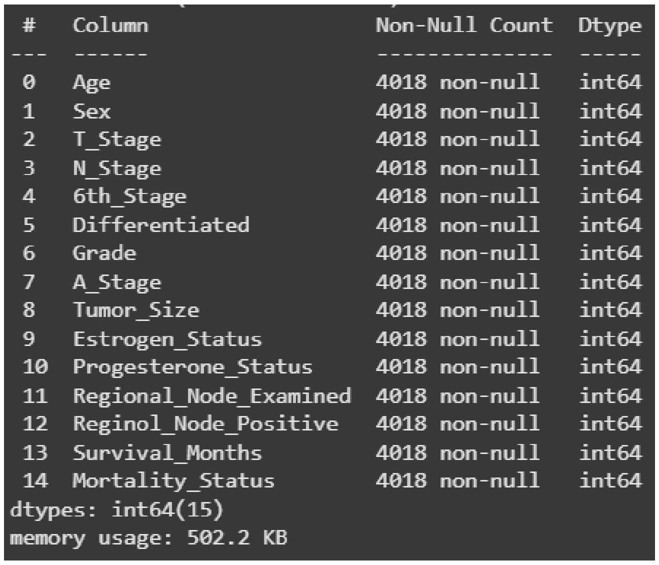
Even after combining two models, the ensemble did not significantly boost detection of "Dead" cases—it simply mirrored LR's poor recall on the minority class (down from NB's 0.37 to 0.11). There is no significant improvement in catching high-risk patients.

Recommendation

Because the doctors' priority is to catch as many true "Dead" patients as possible, the ensemble is not suitable. Naïve Bayes, with a recall$_1$ of 0.37 and F$_{11}$ of 0.33, outperforms both LR and the ensemble on the minority class. Therefore, NB should be preferred for mortality prediction, as it better aligns with the clinical goal of maximizing true-positive detection

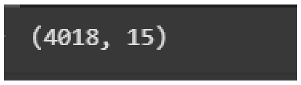
Case Study (B)

Task 1



```
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    4018 non-null   int64
 1   Sex                    4018 non-null   int64
 2   T_Stage                4018 non-null   int64
 3   N_Stage                4018 non-null   int64
 4   6th_Stage              4018 non-null   int64
 5   Differentiated         4018 non-null   int64
 6   Grade                  4018 non-null   int64
 7   A_Stage                4018 non-null   int64
 8   Tumor_Size             4018 non-null   int64
 9   Estrogen_Status        4018 non-null   int64
 10  Progesterone_Status    4018 non-null   int64
 11  Regional_Node_Examined 4018 non-null   int64
 12  Reginol_Node_Positive  4018 non-null   int64
 13  Survival_Months        4018 non-null   int64
 14  Mortality_Status       4018 non-null   int64
dtypes: int64(15)
memory usage: 502.2 KB
```

Figure 37: List of retained features, Data types



```
(4018, 15)
```

Figure 38: dataset dimension (rows & columns)

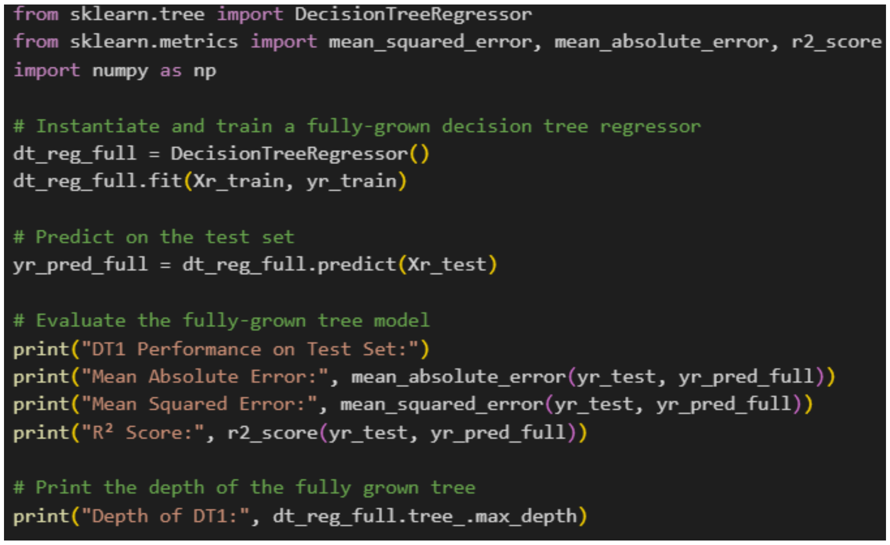
Task 2

a)

A Decision Tree Regressor is a good fit for predicting survival months because it's easy to understand it makes clear "if–then" rules, like "if tumor size is over 30 mm and age is above 60, predict X months." It also handles both numerical and categorical values. Also, trees can spot non-linear patterns and interactions- such as when certain stage and hormone-status combinations change outcomes—that straight-line models miss. They manage outliers and missing values well, so the model stays reliable even if some patient records aren't perfect. All of this makes decision trees simple to use, easy to explain, and dependable for doctors.

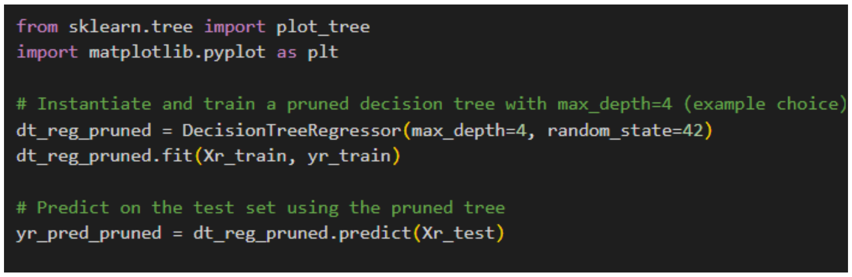
b)

i.



```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Instantiate and train a fully-grown decision tree regressor
dt_reg_full = DecisionTreeRegressor()
dt_reg_full.fit(Xr_train, yr_train)

# Predict on the test set
yr_pred_full = dt_reg_full.predict(Xr_test)

# Evaluate the fully-grown tree model
print("DT1 Performance on Test Set:")
print("Mean Absolute Error:", mean_absolute_error(yr_test, yr_pred_full))
print("Mean Squared Error:", mean_squared_error(yr_test, yr_pred_full))
print("R² Score:", r2_score(yr_test, yr_pred_full))

# Print the depth of the fully grown tree
print("Depth of DT1:", dt_reg_full.tree_.max_depth)
```

Figure 39: DT1 - Fully Grown Decision Tree



```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Instantiate and train a pruned decision tree with max_depth=4 (example choice)
dt_reg_pruned = DecisionTreeRegressor(max_depth=4, random_state=42)
dt_reg_pruned.fit(Xr_train, yr_train)

# Predict on the test set using the pruned tree
yr_pred_pruned = dt_reg_pruned.predict(Xr_test)
```

Figure 40: DT2 - Pruned Decision Tree

ii.

In Notebook 3, the second decision tree (DT-2) was created by pre-pruning—that is, by setting max_depth=4 when instantiating the DecisionTreeRegressor. This limit stops the tree from growing beyond four levels. In practice, it reduces overfitting because the model no longer captures every tiny fluctuation in survival data, much of which can be noise. The result is a simpler, more general model whose splits doctors can follow easily, highlighting only the strongest predictors of survival time. On the other hand, this pruning can miss genuine but subtle patterns deeper in the data. Patients with slightly different profiles might end up with the same predicted survival because the tree does not explore all possible branches. Overall, pre-pruning at depth 4 offers a middle ground between clarity and detail.

C)

DT1 – Fully Grown Decision Tree

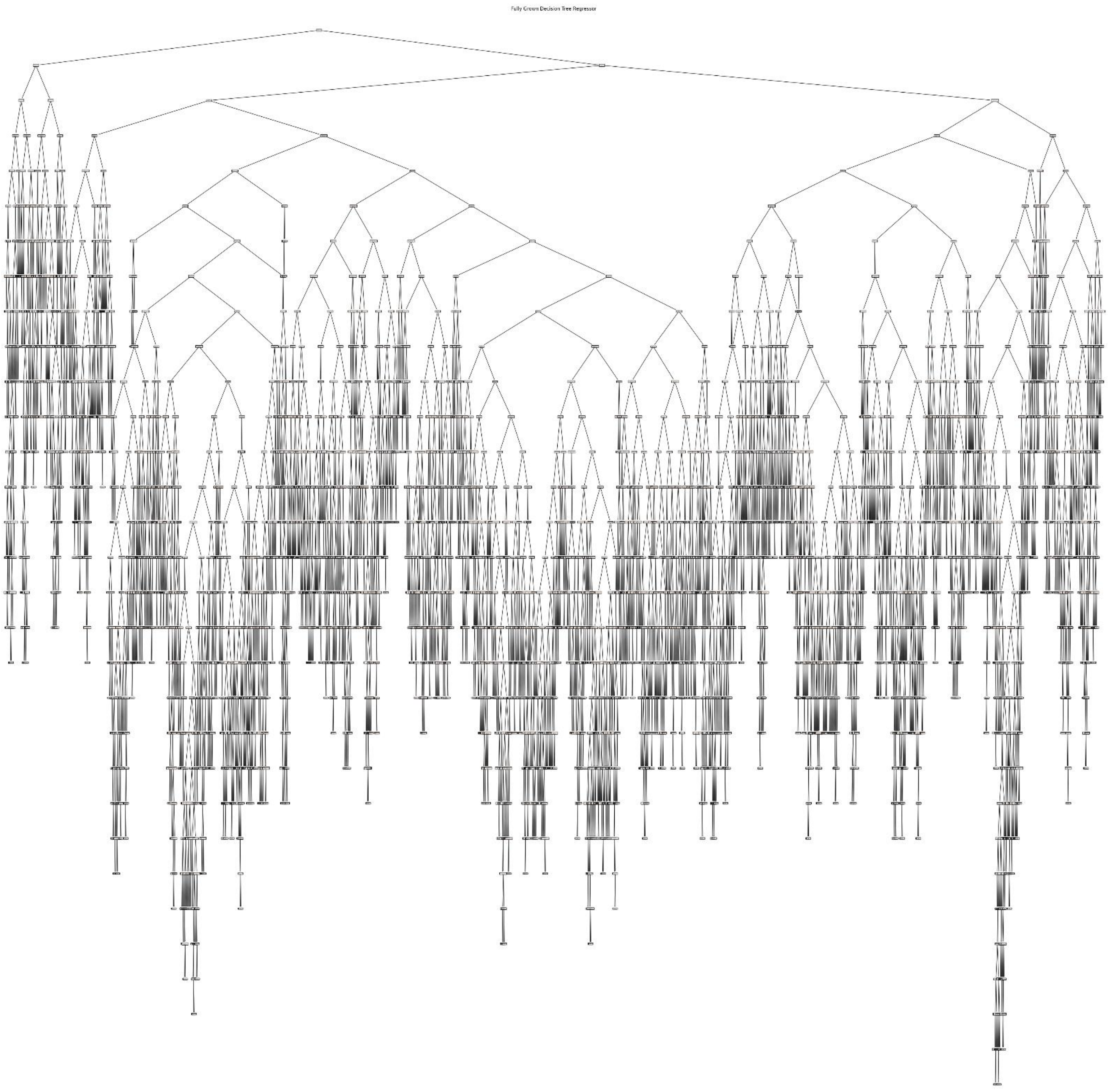


Figure 41: DT1 output

DT2 – Pruned Decision Tree

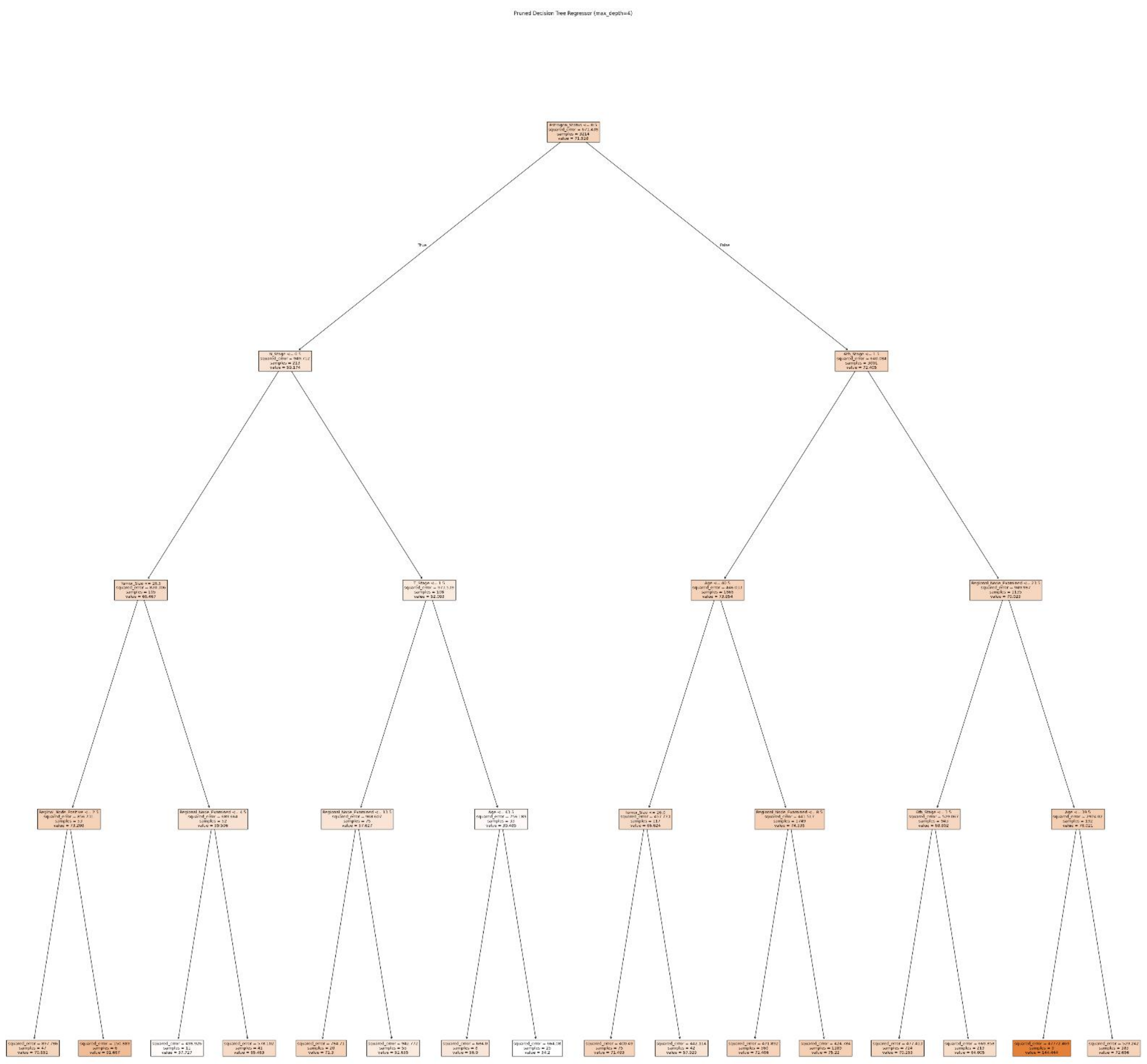


Figure 42: DT2 Output

Task (3)

a)

Metrics	Use or do not use	justification	Model name	Test score
MSE	USE	MSE penalizes large errors	DT-1	1646.4449626865671
			DT-2	522.1249332049442
MAE	USE	MAE gives straightforward average size error	DT-1	26.796641791044777
			DT-2	18.538761801197406
R-Square	DO NOT USE	This can be misleading when predicting individual survival months	DT-1	-2.1273627508812627
			DT-2	0.008242544137590402

Table 5. DT metrics

b)

Recommended Model: DT-2

The pruned model DT-2 is the better choice. It cuts the average error (MAE) from 26.8 months down to 18.5 months and reduces large mistakes (MSE) from 1,646 to 522. Doctors need even small differences in survival time to guide treatment plans, and DT-2's lower errors make its predictions more reliable. Its simpler structure also makes it easier to trace which factors lead to each prediction, helping clinicians trust and use the results.

c)

While MAE and MSE guided the selection of DT-2, each metric has drawbacks. MAE treats all errors equally, so a single large miss carries the same weight as several small ones—even though large errors can be more harmful in treatment planning. MSE penalizes big errors more but can be distorted by a few unusual cases. Neither metric shows how errors differ for early-stage versus late-stage patients. To make sure the model works well for everyone, it's important to also look at median errors or error ranges across different patient groups.

Task (4)

Selected Model: DT-2 (pruned to max_depth=4)

Decision Path for Patient B002565:

1. Estrogen Status: Patient is Negative. In the tree's root node (Estrogen_Status <= 0.5), "Negative" (encoded as 0) satisfies the test, so we follow the left branch.

2. N Stage: Patient's N Stage is N1 (encoded as 1). At the next node (N_Stage <= 0.5), 1 does not satisfy <= 0.5, so we go right.

3. Tumour Size: Patient's tumour is 41 mm. At the third level (Tumour_Size <= 24.5), 41 is greater, so we again go right.
4. Leaf Node: This final leaf has an average survival-months value of 59.4.

Predicted Survival: Based on these splits, DT-2 predicts that patient B002565 will survive approximately 59 months from diagnosis. 0

References

1. Ahmad, Azal (2022). Balanced Split: A new train-test data splitting strategy for imbalanced dataset. Available at: https://arxiv.org/pdf/2212.11116. [accessed April 2, 2025]

2. Brownlee, J. "Train-Test Split for Evaluating Machine Learning Algorithms." Machine Learning Mastery, 26 Aug 2020. https://www.machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/ [accessed April 1, 2025]

3. Fazzolini, "Parameter 'stratify' from method train_test_split (scikit-learn)." StackOverflow, Aug 11, 2016. https://stackoverflow.com/q/34842405 [accessed April 1, 2025]

4. Chawla, N.V., Bowyer, K.W., Hall, L.O. & Kegelmeyer, W.P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Available at: https://www.researchgate.net/publication/220543125_SMOTE_Synthetic_Minority_Over-sampling_Technique [accessed April 25, 2025]

5. Syamsuddin, S. & Nugroho, L.K.A. (2018). A Comparative Study of Synthetic Over-sampling Method to Improve the Classification of Poor Households in Yogyakarta Province. Available at: https://www.researchgate.net/publication/329051480_A_Comparative_Study_of_Synthetic_Over-sampling_Method_to_Improve_the_Classification_of_Poor_Households_in_Yogyakarta_Province [accessed April 17, 2025]

6. Scikit-learn train_test_split docs, 1.6.1. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html [accessed March 31, 2025]

7. Scikit-learn API: sklearn.model_selection.train_test_split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. [accessed March 31, 2025]

8. Jones, M., Patel, R. & Lee, S. (2023). Extracting Knowledge from Machine Learning Models to Diagnose Breast Cancer. Available at: https://www.researchgate.net/publication/388574773_Extracting_Knowledge_from_Machine_Learning_Models_to_Diagnose_Breast_Cancer [accessed March 25, 2025]