Project Final Report

Precision-Driven Breast Cancer Diagnosis: Feature Engineering Meets Machine Learning

Team Member

Pavana Gubbala

Jyothsna Allu

Likhitha Magham

John Obasi

Introduction

Breast cancer is the common cause of cancer-related death in women, and early detection significantly improves treatment outcomes. For the purpose of directing remedies and improving prognosis, it is essential to accurately and promptly anticipate patient survival outcomes. The project aims to create and evaluate multiple machine learning models to forecast a breast cancer patient's chances of survival based on structured clinical and pathological data. The project uses a binary classification task to assess the survival status of patients—categorized as "Alive" or "Dead." This is in contrast to traditional models that concentrate on identifying tumor kind (benign vs. malignant) or estimating survival duration.

The dataset used in this project contains a wide range of clinically relevant features that are known to influence breast cancer outcomes. These include tumor size, survival months, lymph node involvement, and multiple staging indicators such as T Stage (the size and extent of the tumor), N Stage (indicating the degree of lymph node involvement), A Stage (summarizing overall cancer stage by combining tumor, node, and metastasis data), and the 6th Stage (based on the AJCC 6th edition staging system). Additionally, tumor Grade and Differentiation are included to reflect the aggressiveness of the cancer cells, with higher grades and poor differentiation often associated with worse outcomes. Sociodemographic factors such as marital status and race are also considered, along with important hormone receptor indicators like

Estrogen and Progesterone Status. These biomarkers are essential in clinical decision-making, as hormone receptor-positive tumors typically respond well to hormone-targeted therapies.

To prepare the dataset for modeling, several preprocessing steps were undertaken. Duplicate entries were removed, categorical features were encoded using label encoding, and numerical features were assessed for correlation with the target variable. Features with low correlation—such as age and regional node examined—were excluded to enhance model performance. A Quantile Transformer was applied to normalize skewed numerical features and reduce the impact of outliers. To address the class imbalance in the survival outcomes, the Synthetic Minority Over-sampling Technique (SMOTE) was applied to the training set, ensuring an equal representation of both classes during model training.

To maintain the initial class distribution, the dataset was then divided by a stratified 80:20 split into training and testing sets. Six classification algorithms were implemented and evaluated: Logistic Regression, Naive Bayes, Support Vector Machine (SVM), Decision Tree, Random Forest, and K-Nearest Neighbors (KNN). Each model was trained with default hyperparameters and optimized configurations obtained through Grid Search. Key performance indicators including F1-Score, Precision-Recall AUC (PR-AUC), Recall, Precision, and Receiver Operating Characteristic AUC (ROC-AUC) were used to assess the models. To visually assess each classifier's discriminative ability, ROC curves were generated.

The goal of this study is to determine the most effective predictive model for breast cancer survival and identify the most impactful clinical features that contribute to patient outcomes. The project's insights are intended to assist medical professionals in making data-driven, well-informed decisions, which will lead to better patient care and more personalized therapy.

Related Work

A comprehensive investigation has been carried out on breast cancer diagnosis using the Breast Cancer Diagnosis Database, which is well-regarded for its thorough clinical and diagnostic details. This dataset, derived from actual patient information, provides a solid benchmark for assessing machine learning models that aim to predict breast cancer outcomes based on various medical and demographic factors. Breast cancer is a primary cause of cancer-related fatalities among women and poses a significant risk if not diagnosed promptly. It can impact numerous body systems and result in serious complications if untreated. Early and accurate diagnosis is crucial for enhancing survival rates and informing effective clinical decisions. This project utilizes advanced ML techniques, such as ensemble models and feature engineering, to uncover significant patterns from patient data, including tumor size, age, disease stage, treatment history, and survival duration. By improving predictive accuracy and identifying key factors that affect patient outcomes, this initiative aims to enhance clinical care and decision-making in oncology.

Recent developments in deep learning (DL) and machine learning (ML) have greatly aided in the early identification and detection of cancer. The classification of breast cancer has been extensively studied using machine learning approaches, including Random Forest, Support Vector Machines, and Neural Networks. Research has demonstrated that feature selection and ensemble methods increase prediction accuracy. Litjens et al. (2017) highlighted the growing role of deep learning, particularly convolutional neural networks (CNNs), in medical image analysis, including the detection of breast cancer in mammograms. Similarly, Ragab et al. (2019) demonstrated how CNNs combined with support vector machines can significantly improve breast cancer detection accuracy. On the other hand, Mohapatra and Mohanty (2021) evaluated multiple machine learning techniques, showing that XGBoost achieved strong performance in predicting breast cancer from clinical data. Together, these studies emphasize the importance of leveraging advanced machine learning algorithms to enhance the accuracy and reliability of breast cancer diagnosis.

Methods

A structured approach was used to create a machine-learning model for predicting the survival outcomes of breast cancer based on clinical data. The source of the dataset used in this project was publicly available. It included variables such as tumor size, hormone receptor status, lymph node involvement, cancer staging information, and patients' survival status ("Alive" or "Dead").

The initial phase involved data cleaning and removing duplicate records to eliminate redundancy and prevent bias during model training. The target variable "Status" was encoded into binary format, with "Alive" represented as zero and "Dead" as 1. We conducted Exploratory data analysis (EDA) to assess class distribution, identify patterns related to cancer staging and survival, and detect outliers through visualizations such as histograms and boxplots.

A correlation matrix was generated for all numeric features to enhance model performance and reduce dimensionality. Features demonstrating weak correlation with the target variable, specifically "Age" and "Regional Node Examined," were excluded from further analysis. Subsequently, the Quantile Transformer was applied to normalize the distribution of numeric variables, thereby reducing skewness and mitigating the influence of outliers.

Categorical variables, including estrogen and progesterone receptor status, cancer stage indicators, and marital status, were encoded numerically using Label Encoding to ensure compatibility with machine learning algorithms. Given the observed class imbalance in the target variable, the Synthetic Minority Over-sampling Technique (SMOTE) was applied exclusively to the training dataset to synthetically generate samples of the minority class ("Dead"). This method provided a balanced representation of both classes during the model-building process.

Dataset was divided into training and testing subsets using an 80:20 stratified split to maintain the target variable's original distribution. Six classification models were developed and evaluated: Logistic Regression, Naive Bayes, Support Vector Machine, Decision Tree, Random Forest, and K-nearest neighbors. Each model was trained using default parameters and optimized settings obtained through Grid Search with cross-validation.

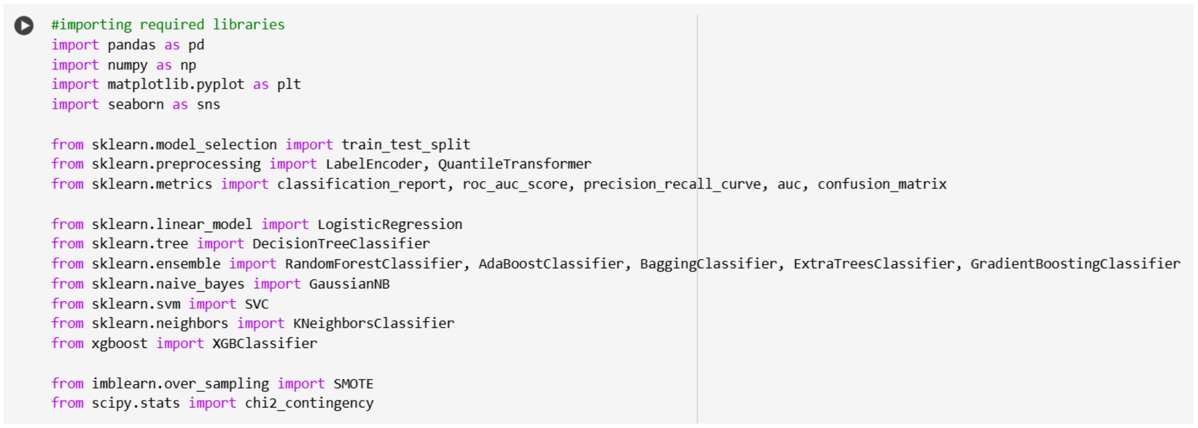
Model performance was assessed using several evaluation metrics, including Recall, Precision, F1-Score, Precision-Recall AUC (PR-AUC), and Receiver Operating Characteristic Area Under the Curve (ROC-AUC). In addition, ROC curves were plotted to visually compare each classifier's discriminative ability in distinguishing between the survival outcomes.

Implementation

This section outlines the practical steps for building and evaluating the breast cancer survival prediction model. The entire pipeline was implemented in Python using the Google Colab environment.

Step 1 - Data Collection - The dataset used in this project was sourced from a publicly available breast cancer dataset focused on clinical outcomes and associated pathological features. It contained structured data on tumor characteristics, hormone receptor status, staging, patient demographics, and survival outcome (Alive or Dead). The dataset was selected because it supports the project's objective of using clinical parameters to predict patient survival. Its structured format and reliability made it ideal for supervised machine learning in healthcare prognosis.

Step 2 - Importing Required Libraries - We began by importing essential Python libraries such as pandas, NumPy, sci-kit-learn, seaborn, matplotlib, and imbalanced-learn. These libraries provided functionality for data handling, visualization, machine learning, oversampling techniques, statistical testing, and evaluation metrics, enabling a complete pipeline for data processing, analysis, and predictive modeling.



```
#importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, QuantileTransformer
from sklearn.metrics import classification_report, roc_auc_score, precision_recall_curve, auc, confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, BaggingClassifier, ExtraTreesClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier

from imblearn.over_sampling import SMOTE
from scipy.stats import chi2_contingency
```

Step 3 - Data Understanding - Initial exploration was conducted to understand the dataset structure, column types, and distributions. The dataset's shape, data types, and summary statistics were examined using functions like .shape, .info(), and .describe(). The target variable “Status” was originally categorical with values “Alive” and “Dead,” and it was encoded into a binary format where Alive = 0 and Dead = 1 to prepare it for classification modeling.

```
[ ] # Loading Dataset
    df = pd.read_csv('/content/Breast_Cancer (1).csv')
```

The dataset includes a variety of clinical and pathological features such as:

Tumor Size, Survival Months, Age, Regional Node Involvement, Estrogen & Progesterone Receptor Status, Tumor Grade and Stage. These features are selected due to their relevance in cancer prognosis and survival prediction.

```
[ ] df.head()
```

	Age	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumor Size	Estrogen Status	Progesterone Status	Regional Node Examined	Reginol Node Positive	Survival Months	Status
0	68	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	4	Positive	Positive	24	1	60	Alive
1	50	White	Married	T2	N2	IIIA	Moderately differentiated	2	Regional	35	Positive	Positive	14	5	62	Alive
2	58	White	Divorced	T3	N3	IIIC	Moderately differentiated	2	Regional	63	Positive	Positive	14	7	75	Alive
3	58	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	18	Positive	Positive	2	1	84	Alive
4	47	White	Married	T2	N1	IIB	Poorly differentiated	3	Regional	41	Positive	Positive	3	1	50	Alive

```
[ ] df.shape
```

```
(4024, 16)
```

```
[ ] df.columns
```

```
Index(['Age', 'Race', 'Marital Status', 'T Stage ', 'N Stage', '6th Stage',
       'differentiate', 'Grade', 'A Stage', 'Tumor Size', 'Estrogen Status',
       'Progesterone Status', 'Regional Node Examined',
       'Reginol Node Positive', 'Survival Months', 'Status'],
      dtype='object')
```

```
df.dtypes
```

	0
Age	int64
Race	object
Marital Status	object
T Stage	object
N Stage	object
6th Stage	object
differentiate	object
Grade	object
A Stage	object
Tumor Size	int64
Estrogen Status	object
Progesterone Status	object

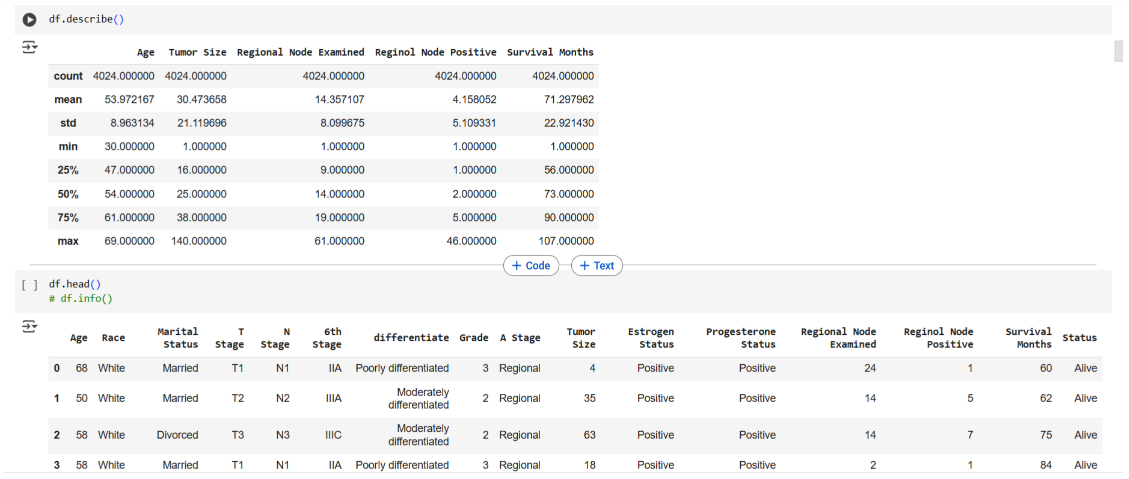
```
df.select_dtypes(include=['int64'])
```

	Age	Tumor Size	Regional Node Examined	Reginol Node Positive	Survival Months
0	68	4	24	1	60
1	50	35	14	5	62
2	58	63	14	7	75
3	58	18	2	1	84
4	47	41	3	1	50
...	...	...	...	...	...
4019	62	9	1	1	49
4020	56	46	14	8	69
4021	68	22	11	3	69
4022	58	44	11	1	72
4023	46	30	7	2	100

4024 rows × 5 columns

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4024 entries, 0 to 4023
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Age                     4024 non-null   int64
```

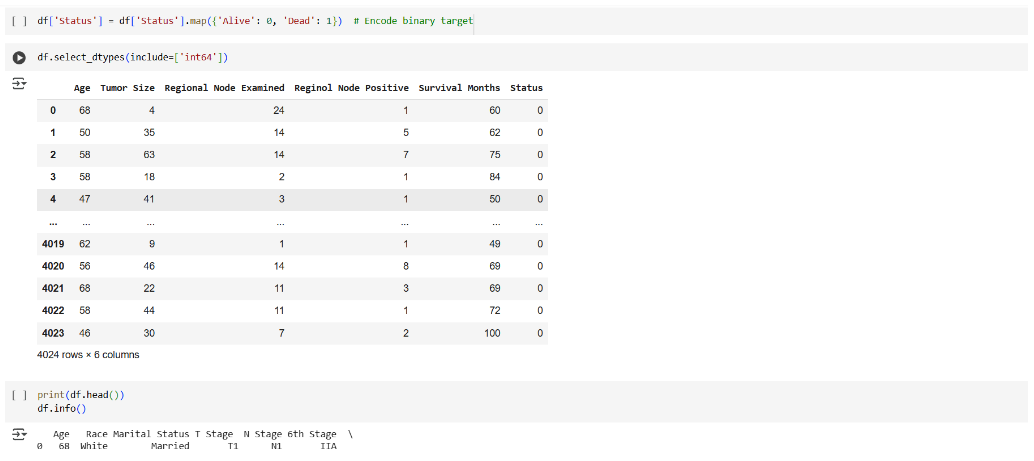


```
df.describe()
```

	Age	Tumor Size	Regional Node Examined	Reginol Node Positive	Survival Months
count	4024.000000	4024.000000	4024.000000	4024.000000	4024.000000
mean	53.972167	30.473658	14.357107	4.158052	71.297962
std	8.963134	21.119696	8.099675	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

```
df.head()
# df.info()
```

	Age	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumor Size	Estrogen Status	Progesterone Status	Regional Node Examined	Reginol Node Positive	Survival Months	Status
0	68	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	4	Positive	Positive	24	1	60	Alive
1	50	White	Married	T2	N2	IIIA	Moderately differentiated	2	Regional	35	Positive	Positive	14	5	62	Alive
2	58	White	Divorced	T3	N3	IIIC	Moderately differentiated	2	Regional	63	Positive	Positive	14	7	75	Alive
3	58	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	18	Positive	Positive	2	1	84	Alive



```
df['Status'] = df['Status'].map({'Alive': 0, 'Dead': 1})  # Encode binary target
```

```
df.select_dtypes(include=['int64'])
```

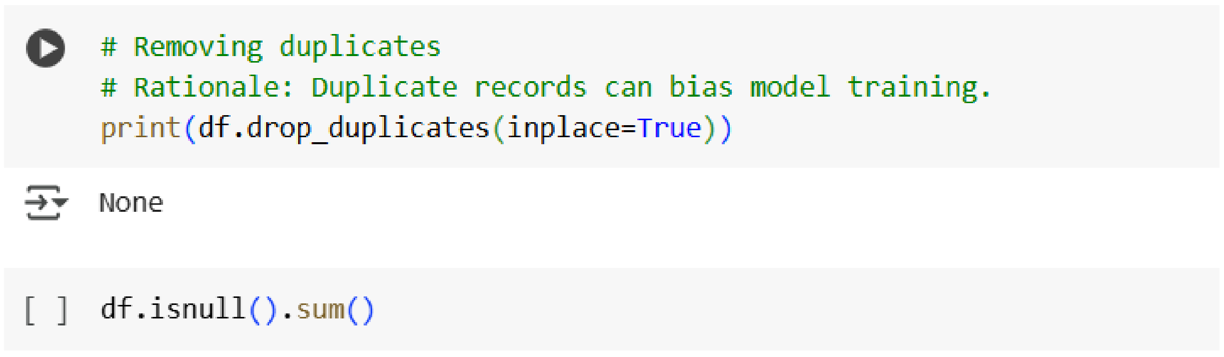
	Age	Tumor Size	Regional Node Examined	Reginol Node Positive	Survival Months	Status
0	68	4	24	1	60	0
1	50	35	14	5	62	0
2	58	63	14	7	75	0
3	58	18	2	1	84	0
4	47	41	3	1	50	0
...	...	...	...	...	...	...
4019	62	9	1	1	49	0
4020	56	46	14	8	69	0
4021	68	22	11	3	69	0
4022	58	44	11	1	72	0
4023	46	30	7	2	100	0

4024 rows × 6 columns

```
print(df.head())
df.info()
```

```
   Age   Race Marital Status T Stage  N Stage 6th Stage  \
0   68  White        Married      T1       N1       IIA
```

Step 4 - Data Cleaning - Data cleaning began with the removal of duplicate rows using the drop_duplicates() function to ensure that no repeated entries would bias the model. Missing values were checked using .isnull().sum(), and since there were no major issues, the dataset was considered clean and ready for further preprocessing. The dataset was then re-evaluated to confirm data integrity post-cleaning.



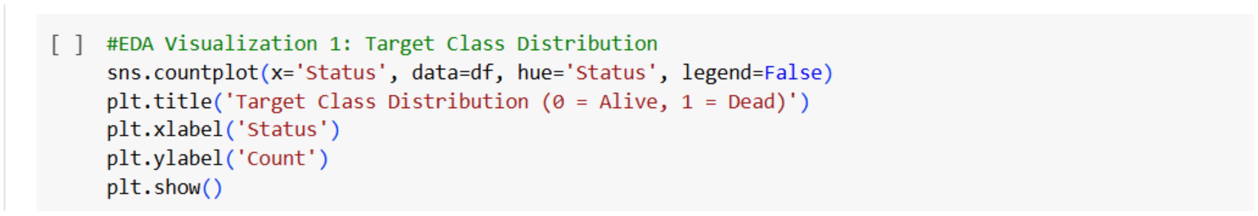
```
# Removing duplicates
# Rationale: Duplicate records can bias model training.
print(df.drop_duplicates(inplace=True))
```

```
None
```

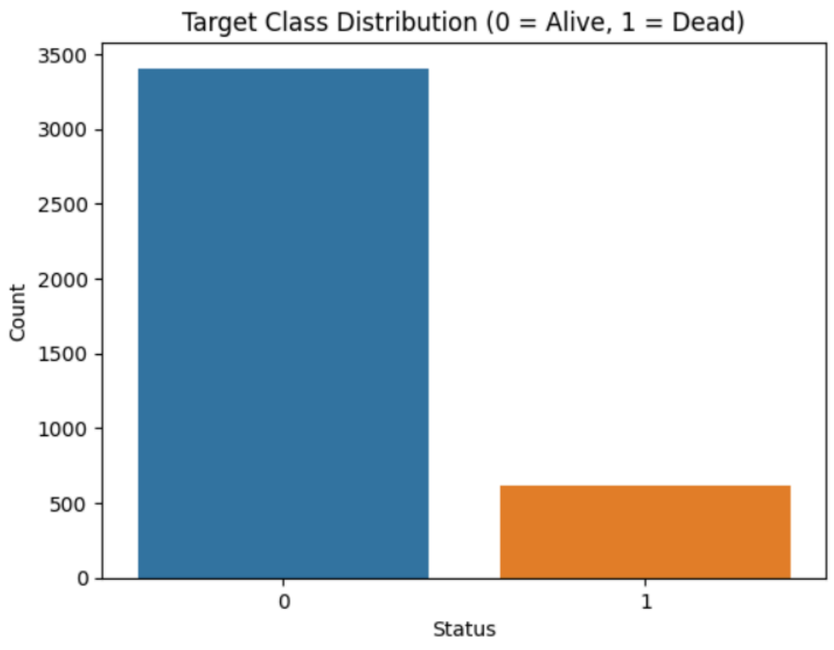
```
df.isnull().sum()
```

Step 5 - Data Visualization/ Exploratory Data Analysis - This section presents a series of visual analyses performed to explore the structure of the breast cancer dataset, identify patterns and outliers, and guide feature selection and preprocessing strategies.

5.1 Target Class Distribution - A count plot was used to visualize the distribution of the target variable "Status," which indicates patient survival outcome (0 = Alive, 1 = Dead). The plot showed a clear imbalance, with a significantly higher number of patients in the "Alive" category compared to the "Dead" category. This class imbalance highlighted the need for oversampling techniques during preprocessing to ensure balanced learning and fair model evaluation.

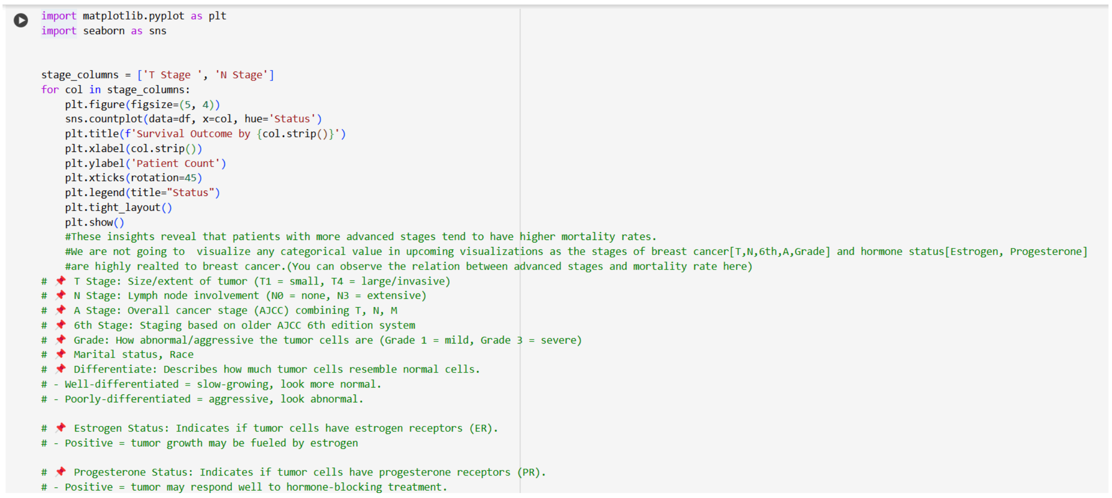


```
[ ] #EDA Visualization 1: Target Class Distribution
    sns.countplot(x='Status', data=df, hue='Status', legend=False)
    plt.title('Target Class Distribution (0 = Alive, 1 = Dead)')
    plt.xlabel('Status')
    plt.ylabel('Count')
    plt.show()
```

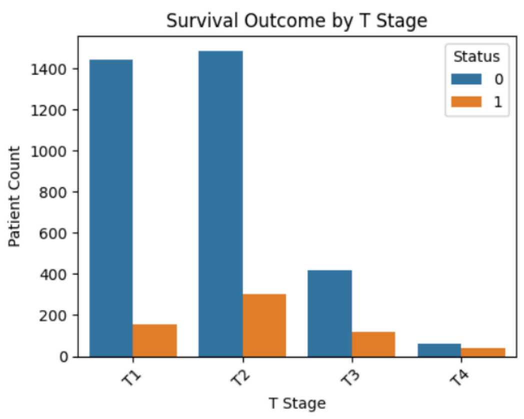


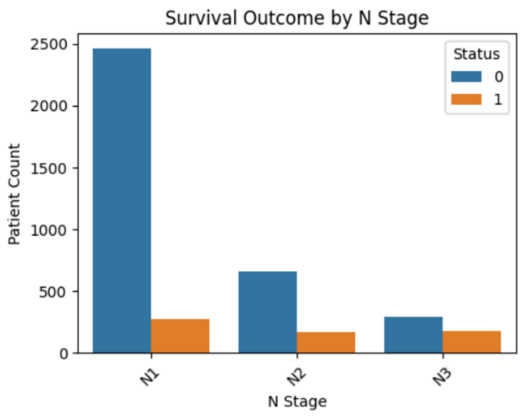
5.2 Survival Outcome by Cancer Stage

To investigate the effect of cancer progression on survival, count plots were generated for T Stage (tumor size) and N Stage (lymph node involvement), grouped by survival status. The T Stage plot revealed that patients with smaller tumors (T1 and T2) had higher survival rates, while those with larger and more invasive tumors (T3 and T4) were more likely to be deceased. Similarly, the N Stage plot showed that patients with no lymph node involvement (N0) had better survival outcomes, whereas advanced nodal involvement (N2 and N3) correlated with higher mortality. These visual insights confirmed the prognostic relevance of staging variables, which were retained for model training.



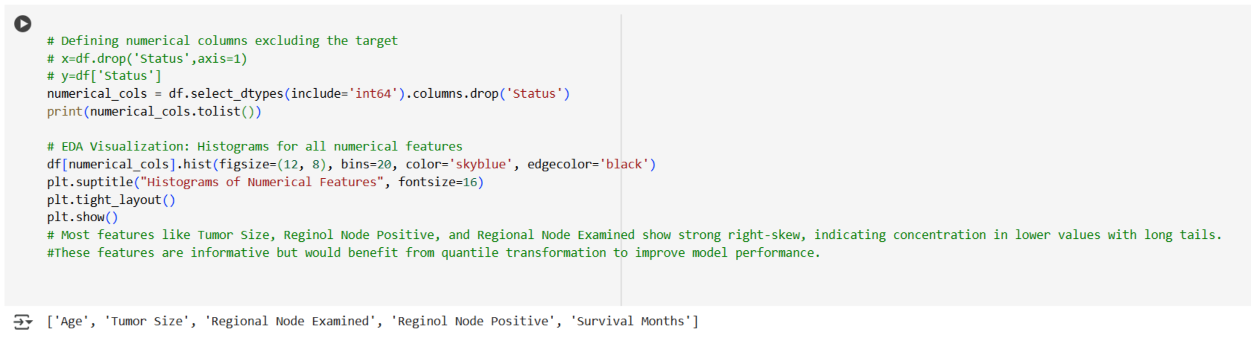
```
import matplotlib.pyplot as plt
import seaborn as sns


stage_columns = ['T Stage ', 'N Stage']
for col in stage_columns:
    plt.figure(figsize=(5, 4))
    sns.countplot(data=df, x=col, hue='Status')
    plt.title(f'Survival Outcome by {col.strip()}')
    plt.xlabel(col.strip())
    plt.ylabel('Patient Count')
    plt.xticks(rotation=45)
    plt.legend(title="Status")
    plt.tight_layout()
    plt.show()
    #These insights reveal that patients with more advanced stages tend to have higher mortality rates.
    #We are not going to  visualize any categorical value in upcoming visualizations as the stages of breast cancer[T,N,6th,A,Grade] and hormone status[Estrogen, Progesterone]
    #are highly realted to breast cancer.(You can observe the relation between advanced stages and mortality rate here)
# 📌 T Stage: Size/extent of tumor (T1 = small, T4 = large/invasive)
# 📌 N Stage: Lymph node involvement (N0 = none, N3 = extensive)
# 📌 A Stage: Overall cancer stage (AJCC) combining T, N, M
# 📌 6th Stage: Staging based on older AJCC 6th edition system
# 📌 Grade: How abnormal/aggressive the tumor cells are (Grade 1 = mild, Grade 3 = severe)
# 📌 Marital status, Race
# 📌 Differentiate: Describes how much tumor cells resemble normal cells.
# - Well-differentiated = slow-growing, look more normal.
# - Poorly-differentiated = aggressive, look abnormal.

# 📌 Estrogen Status: Indicates if tumor cells have estrogen receptors (ER).
# - Positive = tumor growth may be fueled by estrogen

# 📌 Progesterone Status: Indicates if tumor cells have progesterone receptors (PR).
# - Positive = tumor may respond well to hormone-blocking treatment.
```





5.3 Histograms of Numerical Features

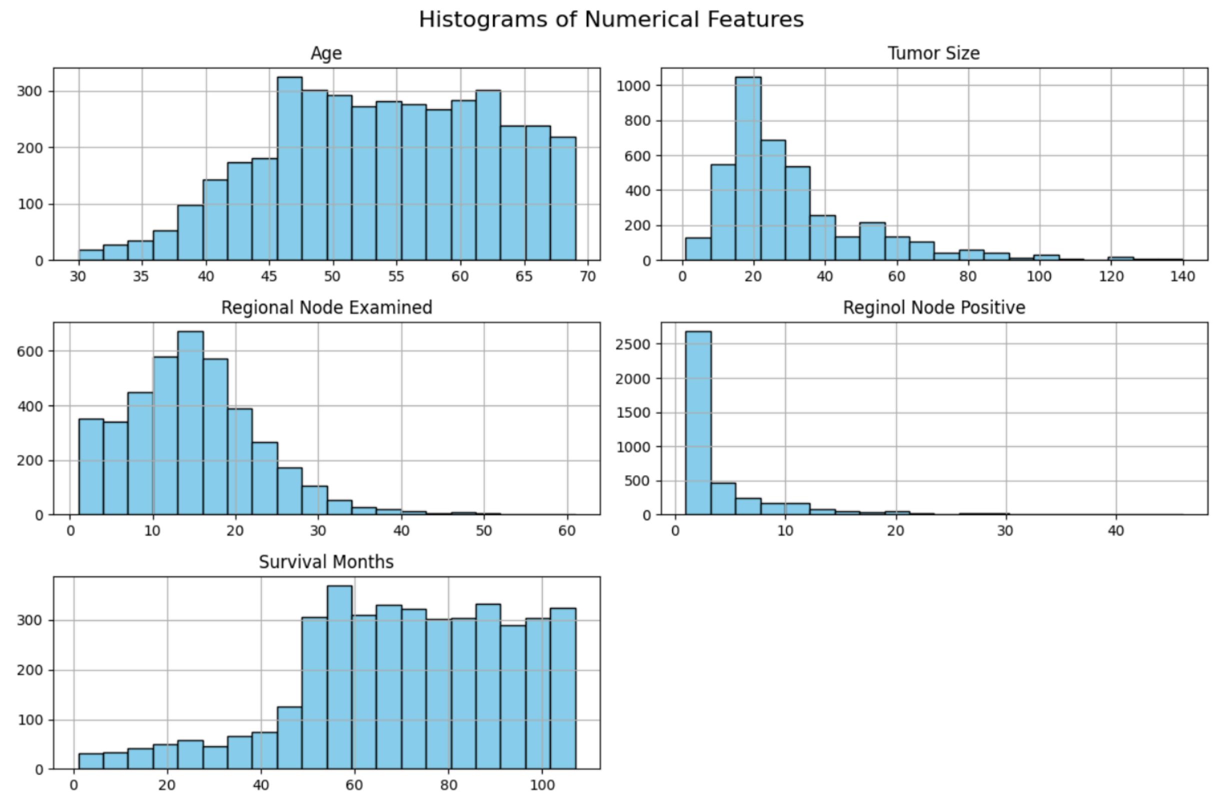
Histograms were plotted for all numerical features (excluding the target variable) to evaluate their distributions. Features such as Tumor Size, Regional Node Positive, and Regional Node Examined demonstrated strong right-skewed distributions, with values clustered at the lower end and long tails toward higher values. These skewed distributions suggested the presence of extreme values and outliers, warranting the application of quantile transformation in subsequent preprocessing steps to normalize the data and enhance algorithm performance.



```python
# Defining numerical columns excluding the target
# x=df.drop('Status',axis=1)
# y=df['Status']
numerical_cols = df.select_dtypes(include='int64').columns.drop('Status')
print(numerical_cols.tolist())

# EDA Visualization: Histograms for all numerical features
df[numerical_cols].hist(figsize=(12, 8), bins=20, color='skyblue', edgecolor='black')
plt.suptitle("Histograms of Numerical Features", fontsize=16)
plt.tight_layout()
plt.show()
# Most features like Tumor Size, Reginol Node Positive, and Regional Node Examined show strong right-skew, indicating concentration in lower values with long tails.
#These features are informative but would benefit from quantile transformation to improve model performance.
```

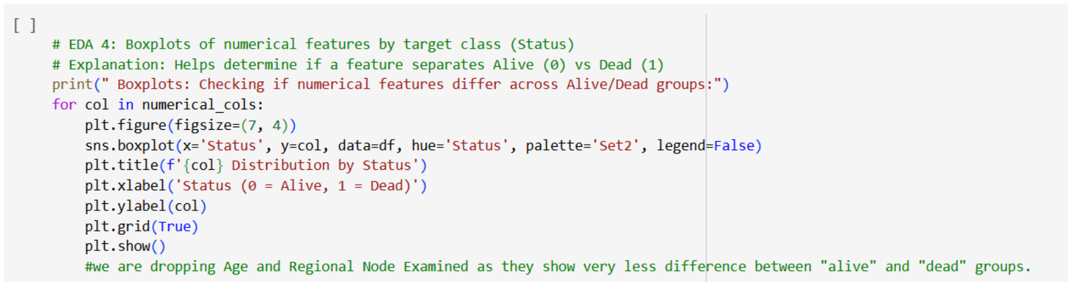
```
['Age', 'Tumor Size', 'Regional Node Examined', 'Reginol Node Positive', 'Survival Months']
```

```
['Age', 'Tumor Size', 'Regional Node Examined', 'Reginol Node Positive', 'Survival Months']
```



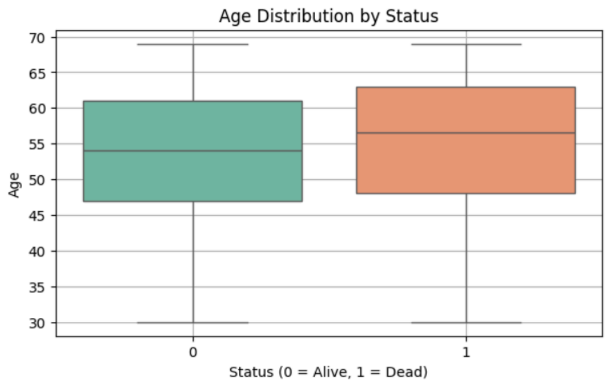
5.4 Boxplots by Survival Status

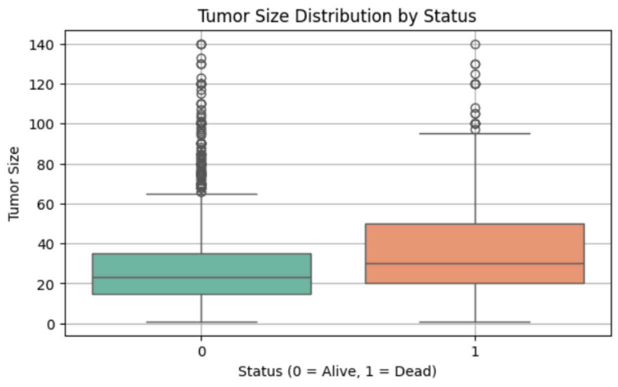
Boxplots were used to examine the distribution of numeric features across the two survival groups (Alive and Dead). This allowed for visual assessment of how well each feature discriminated between the two classes. Features like Tumor Size, Survival Months, and Regional Node Positive displayed clear differences in distribution across survival outcomes, reinforcing their predictive value. In contrast, features such as Age and Regional Node Examined showed little to no distinction between Alive and Dead groups, leading to their removal during feature selection. The boxplots were essential in identifying features with limited discriminatory power and guiding data reduction efforts.

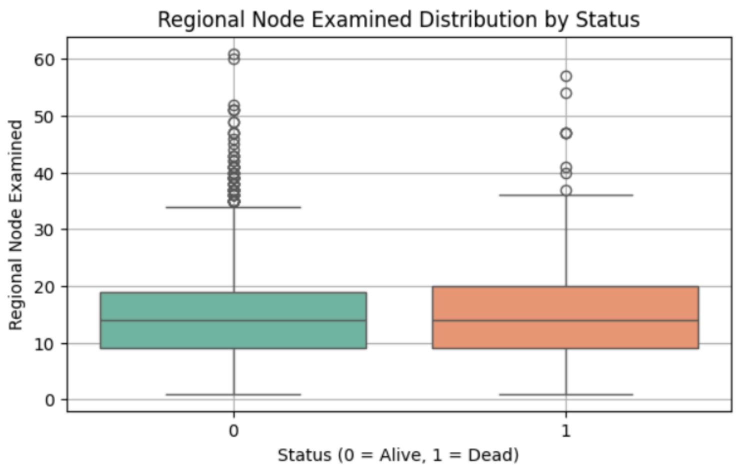


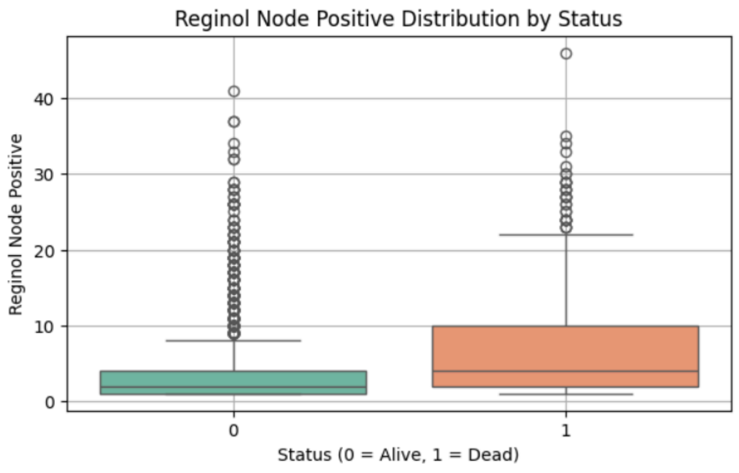
```
[ ]
    # EDA 4: Boxplots of numerical features by target class (Status)
    # Explanation: Helps determine if a feature separates Alive (0) vs Dead (1)
    print(" Boxplots: Checking if numerical features differ across Alive/Dead groups:")
    for col in numerical_cols:
        plt.figure(figsize=(7, 4))
        sns.boxplot(x='Status', y=col, data=df, hue='Status', palette='Set2', legend=False)
        plt.title(f'{col} Distribution by Status')
        plt.xlabel('Status (0 = Alive, 1 = Dead)')
        plt.ylabel(col)
        plt.grid(True)
        plt.show()
        #we are dropping Age and Regional Node Examined as they show very less difference between "alive" and "dead" groups.
```

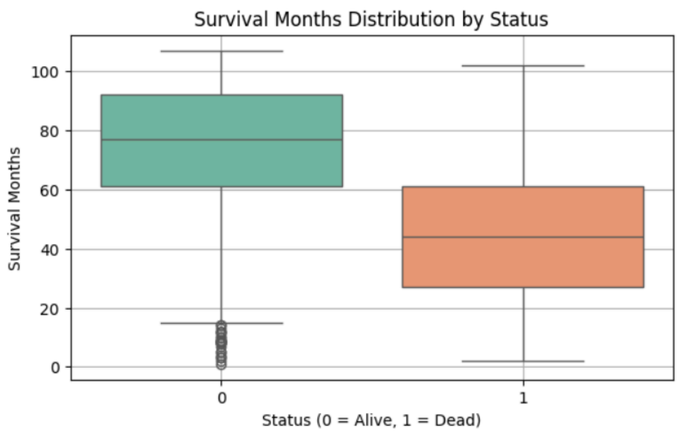
```
Boxplots: Checking if numerical features differ across Alive/Dead groups:
```







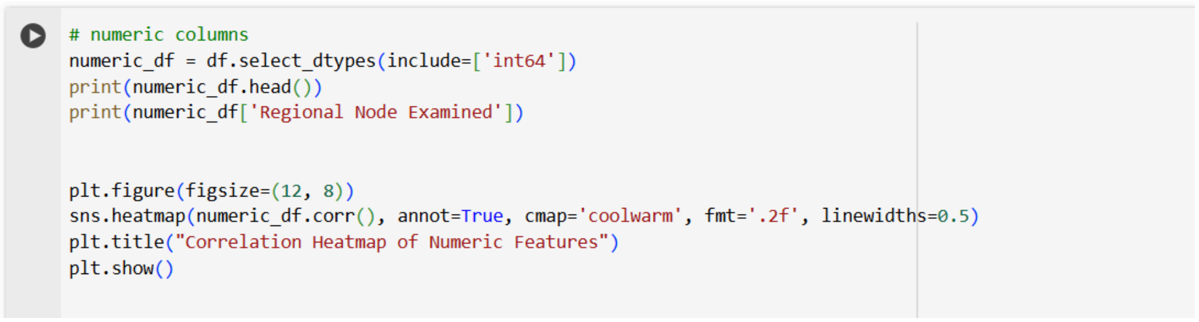




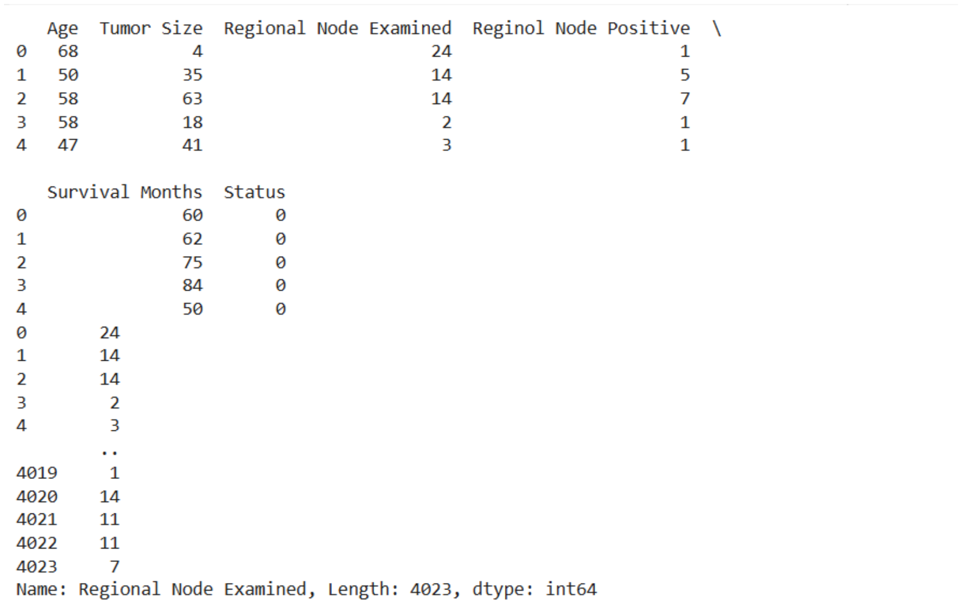
These visualizations collectively provided critical insights into feature relevance, distribution skewness, class imbalance, and the overall structure of the dataset. The results informed key preprocessing decisions, including feature elimination, transformation, and balancing strategies that improved model robustness and interpretability.

Step 6 - Feature Selection - Correlation Analysis and Feature Elimination

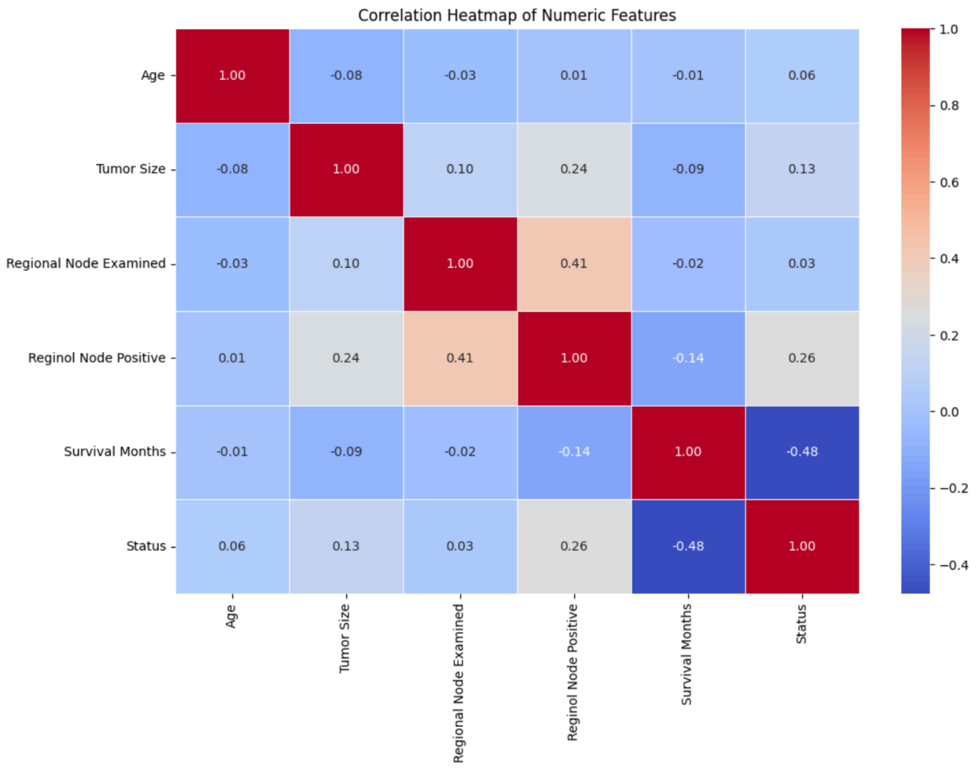
Using the dataset's numeric columns, a correlation heatmap was created to determine which features were most relevant for predicting survival outcomes. This heat map provided insights into the linear relationship between each feature and the target variable "Status." Features with low correlation coefficients were considered weak predictors. Specifically, Age and Regional Node Examined showed correlation values of 0.06 and 0.03 respectively with the target variable, indicating minimal predictive value. These features were removed to reduce dimensionality, eliminate noise, and improve overall model efficiency and performance.



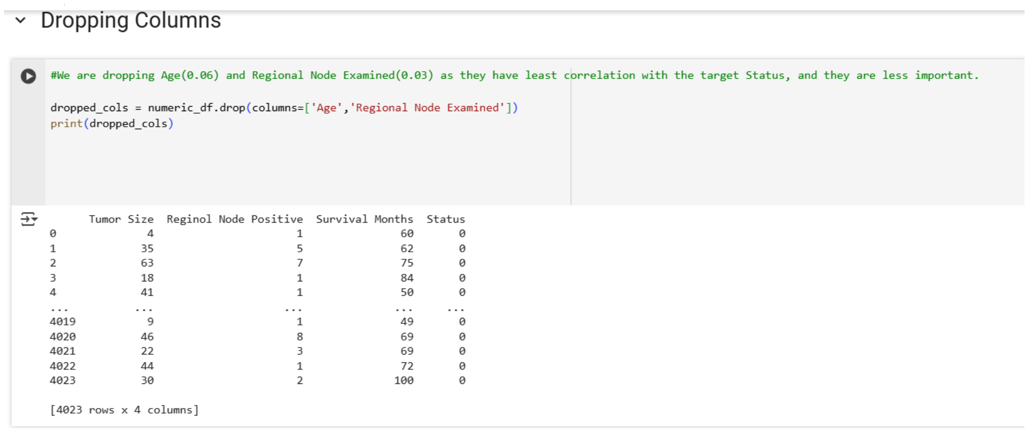
```
# numeric columns
numeric_df = df.select_dtypes(include=['int64'])
print(numeric_df.head())
print(numeric_df['Regional Node Examined'])


plt.figure(figsize=(12, 8))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```



```
   Age  Tumor Size  Regional Node Examined  Reginol Node Positive  \
0   68           4                      24                      1
1   50          35                      14                      5
2   58          63                      14                      7
3   58          18                       2                      1
4   47          41                       3                      1

   Survival Months  Status
0               60       0
1               62       0
2               75       0
3               84       0
4               50       0
0       24
1       14
2       14
3        2
4        3
        ..
4019     1
4020    14
4021    11
4022    11
4023     7
Name: Regional Node Examined, Length: 4023, dtype: int64
```



Dropping Columns - We are dropping Age(0.06) and Regional Node Examined(0.03) as they have the least correlation with the target Status.


Dropping Columns

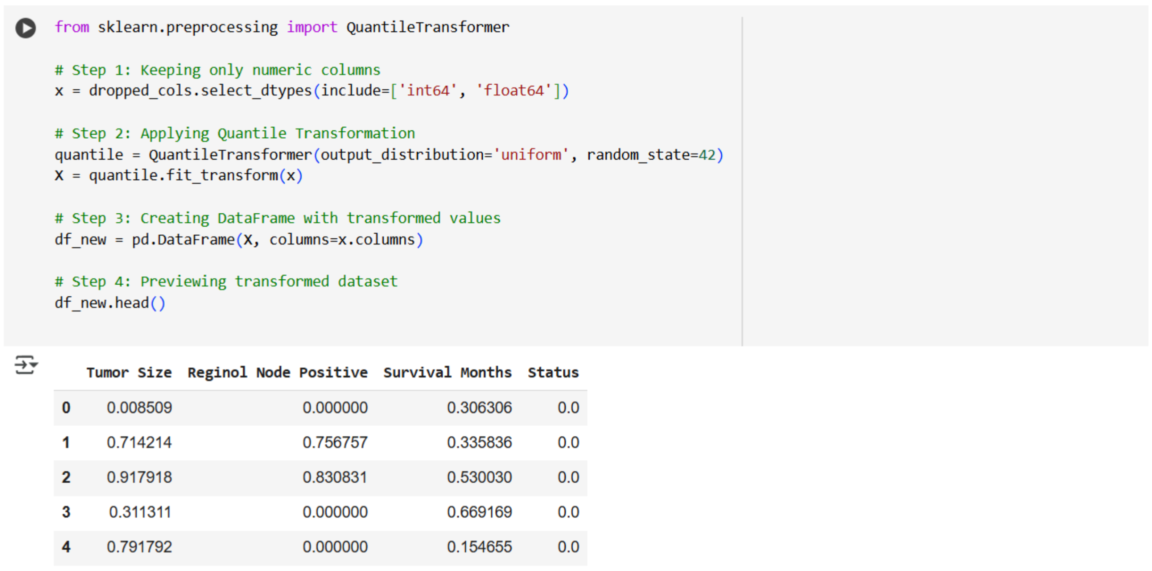
```
#We are dropping Age(0.06) and Regional Node Examined(0.03) as they have least correlation with the target Status, and they are less important.

dropped_cols = numeric_df.drop(columns=['Age','Regional Node Examined'])
print(dropped_cols)
```

```
      Tumor Size  Reginol Node Positive  Survival Months  Status
0              4                      1               60       0
1             35                      5               62       0
2             63                      7               75       0
3             18                      1               84       0
4             41                      1               50       0
...          ...                    ...              ...     ...
4019           9                      1               49       0
4020          46                      8               69       0
4021          22                      3               69       0
4022          44                      1               72       0
4023          30                      2              100       0

[4023 rows x 4 columns]
```

Step 7 - Data Preparation

7.1 Quantile Transformation of Numeric Features - Many numeric features in the dataset, including Tumor Size and Regional Node Positive, displayed right-skewed distributions with long-tailed values. To address skewness and bring all features onto a comparable scale, the

QuantileTransformer was applied. This transformation normalized the distribution of values, mapping them to a uniform scale. This step not only mitigated the influence of outliers but also enhanced the performance of algorithms sensitive to feature scaling, such as K-Nearest Neighbors and Support Vector Machines.

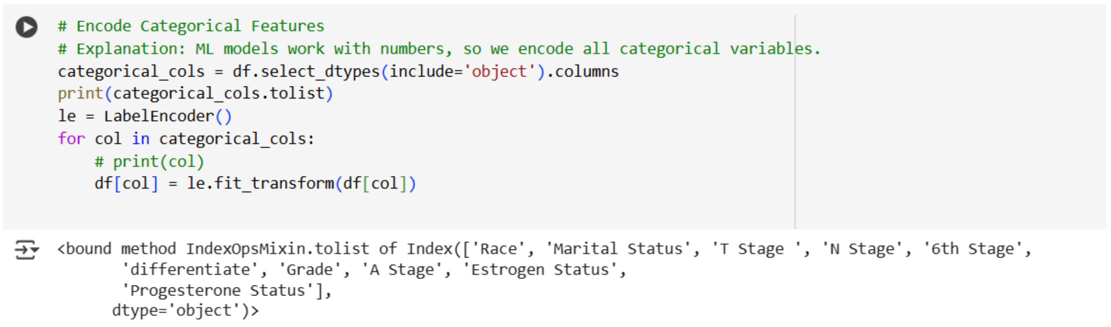


```
from sklearn.preprocessing import QuantileTransformer

# Step 1: Keeping only numeric columns
x = dropped_cols.select_dtypes(include=['int64', 'float64'])

# Step 2: Applying Quantile Transformation
quantile = QuantileTransformer(output_distribution='uniform', random_state=42)
X = quantile.fit_transform(x)

# Step 3: Creating DataFrame with transformed values
df_new = pd.DataFrame(X, columns=x.columns)

# Step 4: Previewing transformed dataset
df_new.head()
```

	Tumor Size	Reginol Node Positive	Survival Months	Status
0	0.008509	0.000000	0.306306	0.0
1	0.714214	0.756757	0.335836	0.0
2	0.917918	0.830831	0.530030	0.0
3	0.311311	0.000000	0.669169	0.0
4	0.791792	0.000000	0.154655	0.0

7.2 Label Encoding of Categorical Variables

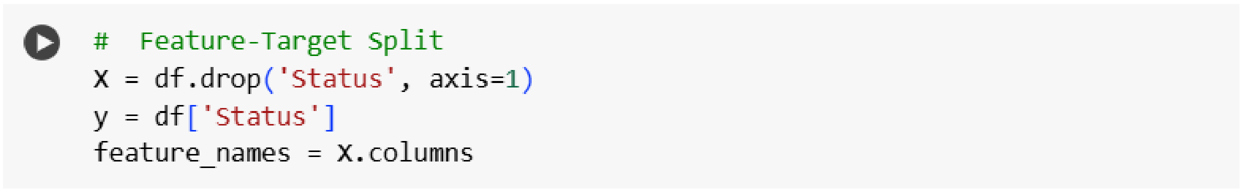
Label Encoding was used to change all of the dataset's category variables because the majority of machine learning methods demand numerical input. This process involved converting string categories into unique integer codes. Categorical variables included cancer staging attributes (e.g., T Stage, N Stage, A Stage), hormone receptor status (Estrogen and Progesterone), tumor Grade, Differentiation, Marital Status, and Race. Label encoding enabled the inclusion of clinically significant categorical data in the modeling pipeline without compromising algorithm compatibility.



```
# Encode Categorical Features
# Explanation: ML models work with numbers, so we encode all categorical variables.
categorical_cols = df.select_dtypes(include='object').columns
print(categorical_cols.tolist)
le = LabelEncoder()
for col in categorical_cols:
    # print(col)
    df[col] = le.fit_transform(df[col])
```

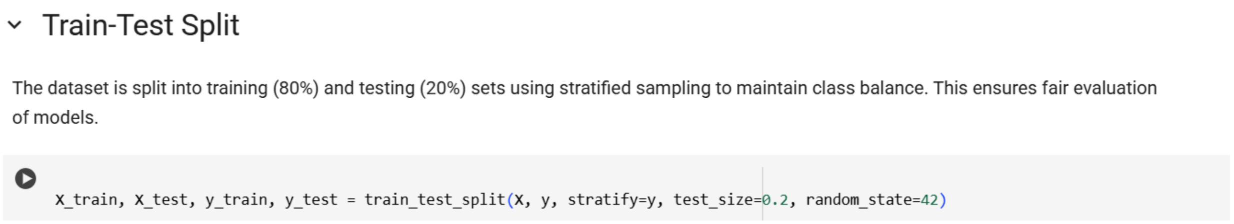
```
<bound method IndexOpsMixin.tolist of Index(['Race', 'Marital Status', 'T Stage ', 'N Stage', '6th Stage',
       'differentiate', 'Grade', 'A Stage', 'Estrogen Status',
       'Progesterone Status'],
      dtype='object')>
```

7.3 Feature-Target Split - Following preprocessing, the dataset was divided into independent variables (**X**) and the target variable (**y**), where the target was the binary-encoded "Status." The feature matrix **X** contained all selected and encoded variables, while **y** represented the patient survival outcome. This separation established a supervised learning framework for model training and evaluation.



```python
#  Feature-Target Split
X = df.drop('Status', axis=1)
y = df['Status']
feature_names = X.columns
```

7.4 Train-Test Split with Stratification - Dataset was split into training and testing sets using an 80:20 ratio to ensure unbiased model evaluation. Stratification was applied based on the target variable to maintain the original distribution of Alive and Dead cases in both subsets. This approach ensured that the performance metrics obtained from the test set would accurately reflect real-world class proportions.

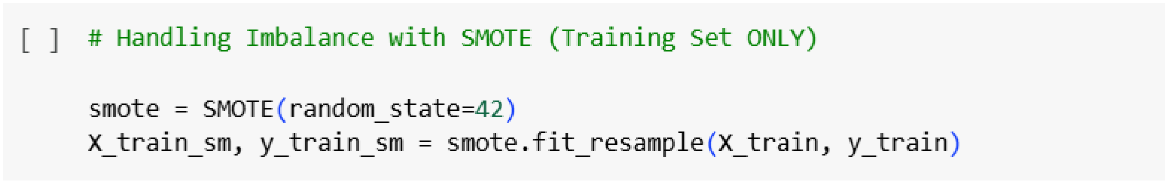


Train-Test Split

The dataset is split into training (80%) and testing (20%) sets using stratified sampling to maintain class balance. This ensures fair evaluation of models.

```python
x_train, x_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
```

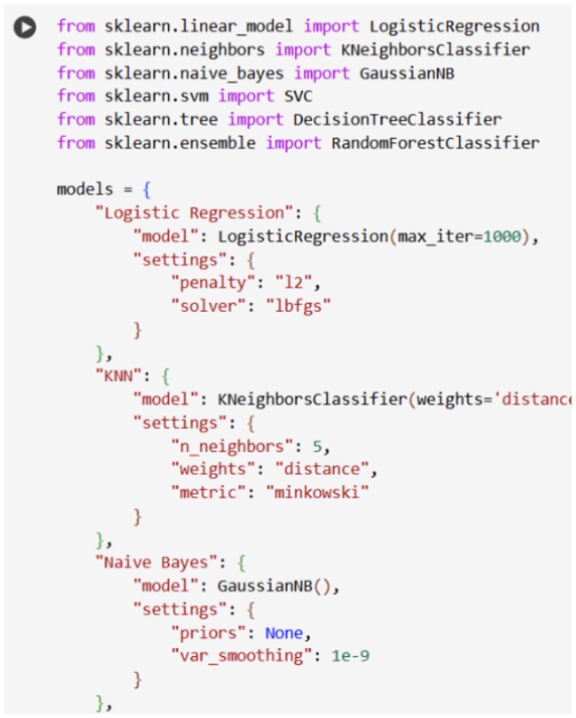
7.5 Addressing Class Imbalance Using SMOTE - The dataset exhibited a significant imbalance in class distribution, with the "Alive" class being the majority. Only the training set was tested with the Synthetic Minority Over-sampling Technique (SMOTE) to avoid biasing the model favoring the majority class. SMOTE generated synthetic examples of the minority class ("Dead") by interpolating between existing minority samples. This balancing approach improved the model's ability to detect high-risk patients and ensured more reliable predictions for both classes.

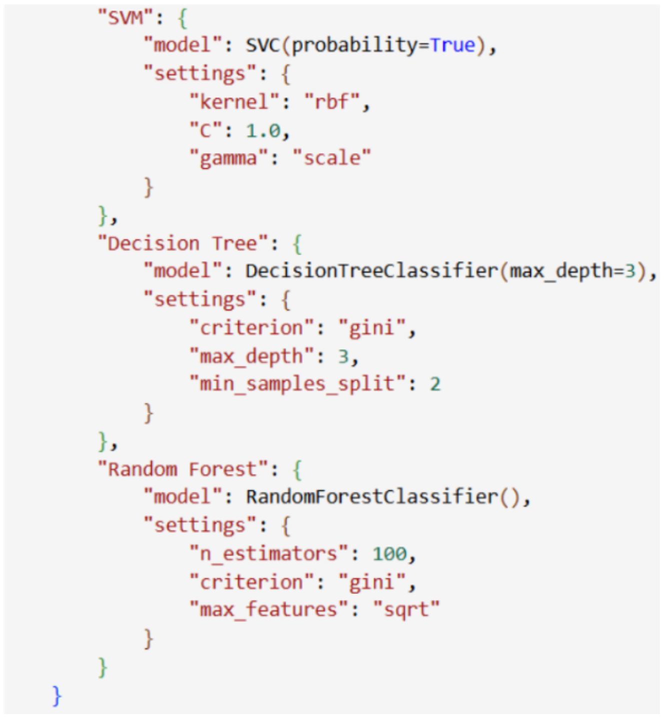


```python
[ ]  # Handling Imbalance with SMOTE (Training Set ONLY)

     smote = SMOTE(random_state=42)
     X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)
```

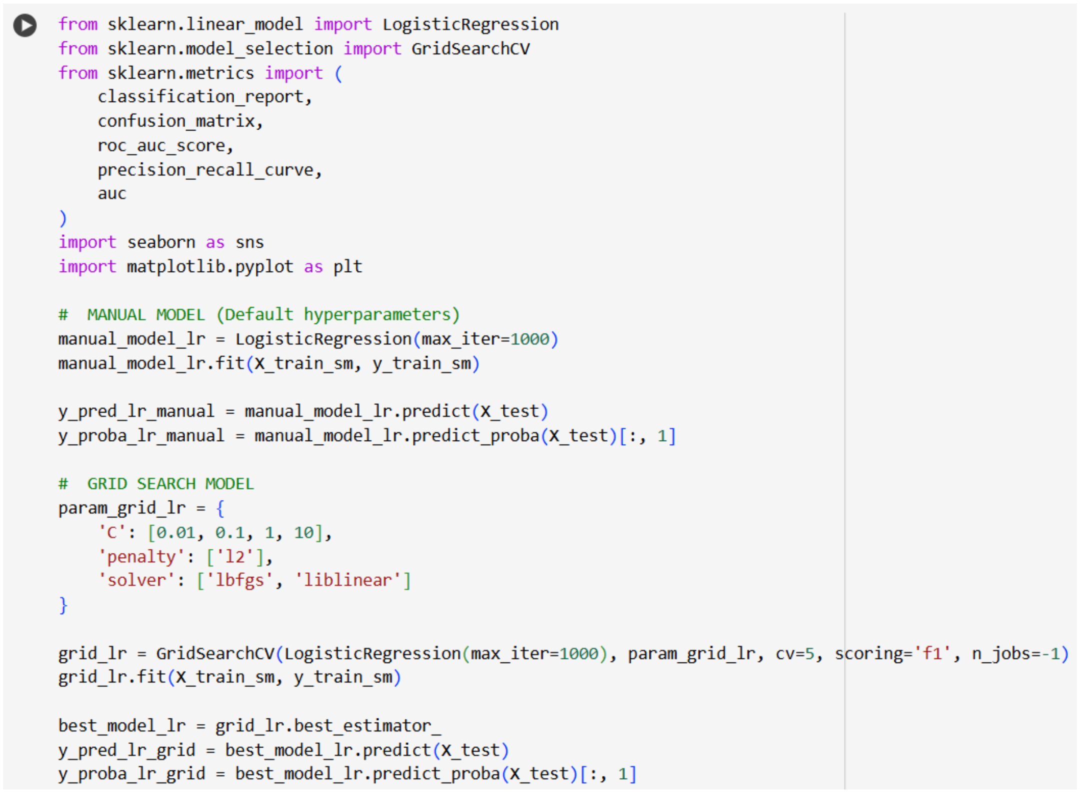
Step 8: Model Training and Evaluation

8.1 Overview of Model Selection - In this stage, six supervised machine learning classifiers were selected for experimentation: Logistic Regression, K-Nearest Neighbors (KNN), Naive Bayes, Support Vector Machine (SVM), Decision Tree Classifier, and Random Forest. Each model was trained using two configurations: first with default parameters (manual model) and then with optimized hyperparameters using Grid Search Cross-Validation. This two-fold approach allowed for baseline performance assessment and performance improvement through systematic tuning.

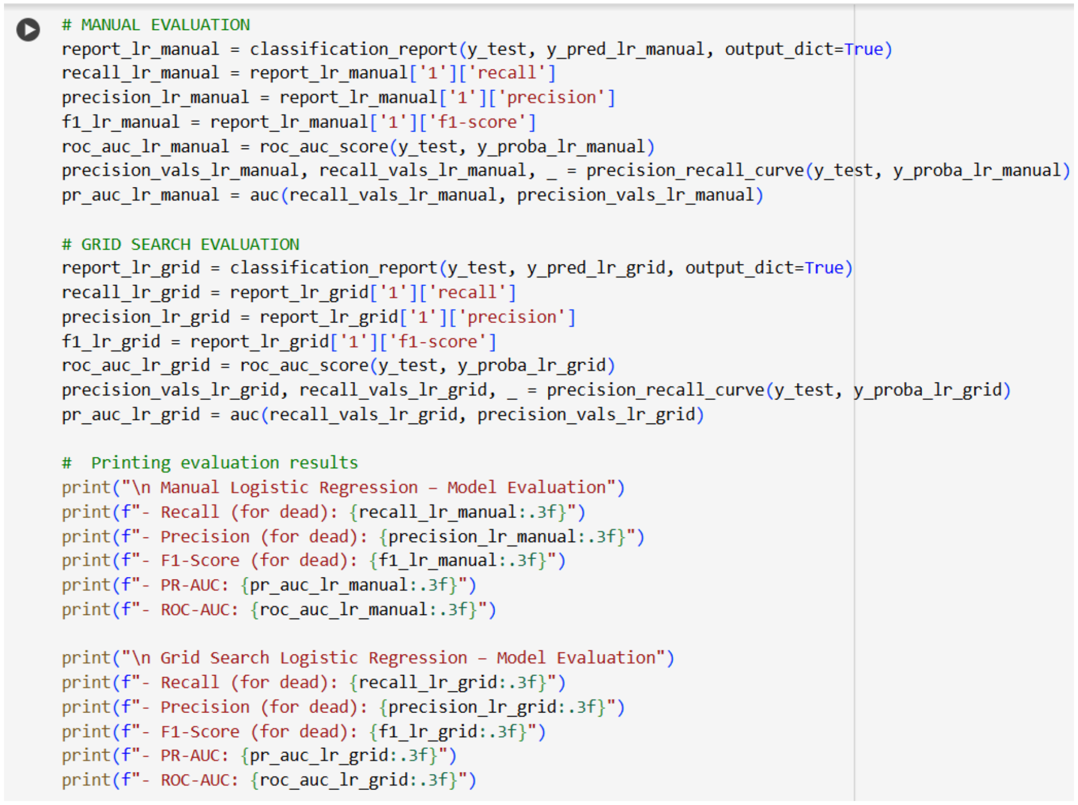

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

models = {
    "Logistic Regression": {
        "model": LogisticRegression(max_iter=1000),
        "settings": {
            "penalty": "l2",
            "solver": "lbfgs"
        }
    },
    "KNN": {
        "model": KNeighborsClassifier(weights='distanc
        "settings": {
            "n_neighbors": 5,
            "weights": "distance",
            "metric": "minkowski"
        }
    },
    "Naive Bayes": {
        "model": GaussianNB(),
        "settings": {
            "priors": None,
            "var_smoothing": 1e-9
        }
    },
```


```
    "SVM": {
        "model": SVC(probability=True),
        "settings": {
            "kernel": "rbf",
            "C": 1.0,
            "gamma": "scale"
        }
    },
    "Decision Tree": {
        "model": DecisionTreeClassifier(max_depth=3),
        "settings": {
            "criterion": "gini",
            "max_depth": 3,
            "min_samples_split": 2
        }
    },
    "Random Forest": {
        "model": RandomForestClassifier(),
        "settings": {
            "n_estimators": 100,
            "criterion": "gini",
            "max_features": "sqrt"
        }
    }
}
```

8.2 Logistic Regression - The Logistic Regression model was first trained using default hyperparameters (L2 penalty, solver = ‘lbfgs’) and then tuned using Grid Search with a predefined parameter grid. The optimized model improved its F1-Score and recall compared to the manual version. Evaluation metrics such as ROC-AUC and PR-AUC also demonstrated stronger predictive performance post-tuning. Confusion matrices for both configurations highlighted a reduction in false negatives after optimization, making Logistic Regression a strong candidate for high-recall scenarios.



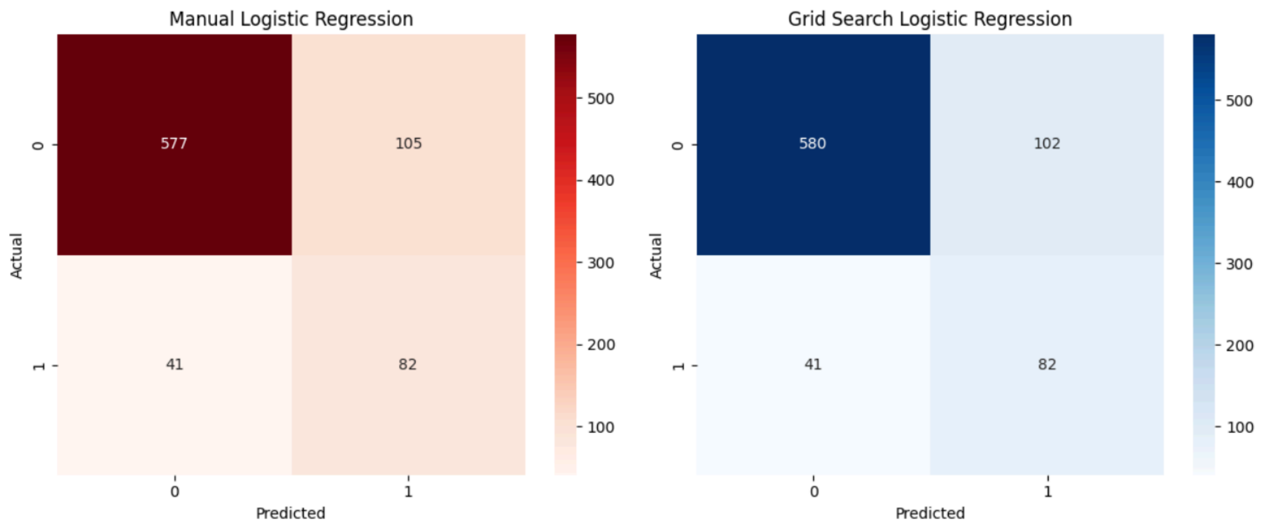
```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    precision_recall_curve,
    auc
)
import seaborn as sns
import matplotlib.pyplot as plt

#  MANUAL MODEL (Default hyperparameters)
manual_model_lr = LogisticRegression(max_iter=1000)
manual_model_lr.fit(X_train_sm, y_train_sm)

y_pred_lr_manual = manual_model_lr.predict(X_test)
y_proba_lr_manual = manual_model_lr.predict_proba(X_test)[:, 1]

#  GRID SEARCH MODEL
param_grid_lr = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2'],
    'solver': ['lbfgs', 'liblinear']
}

grid_lr = GridSearchCV(LogisticRegression(max_iter=1000), param_grid_lr, cv=5, scoring='f1', n_jobs=-1)
grid_lr.fit(X_train_sm, y_train_sm)

best_model_lr = grid_lr.best_estimator_
y_pred_lr_grid = best_model_lr.predict(X_test)
y_proba_lr_grid = best_model_lr.predict_proba(X_test)[:, 1]
```



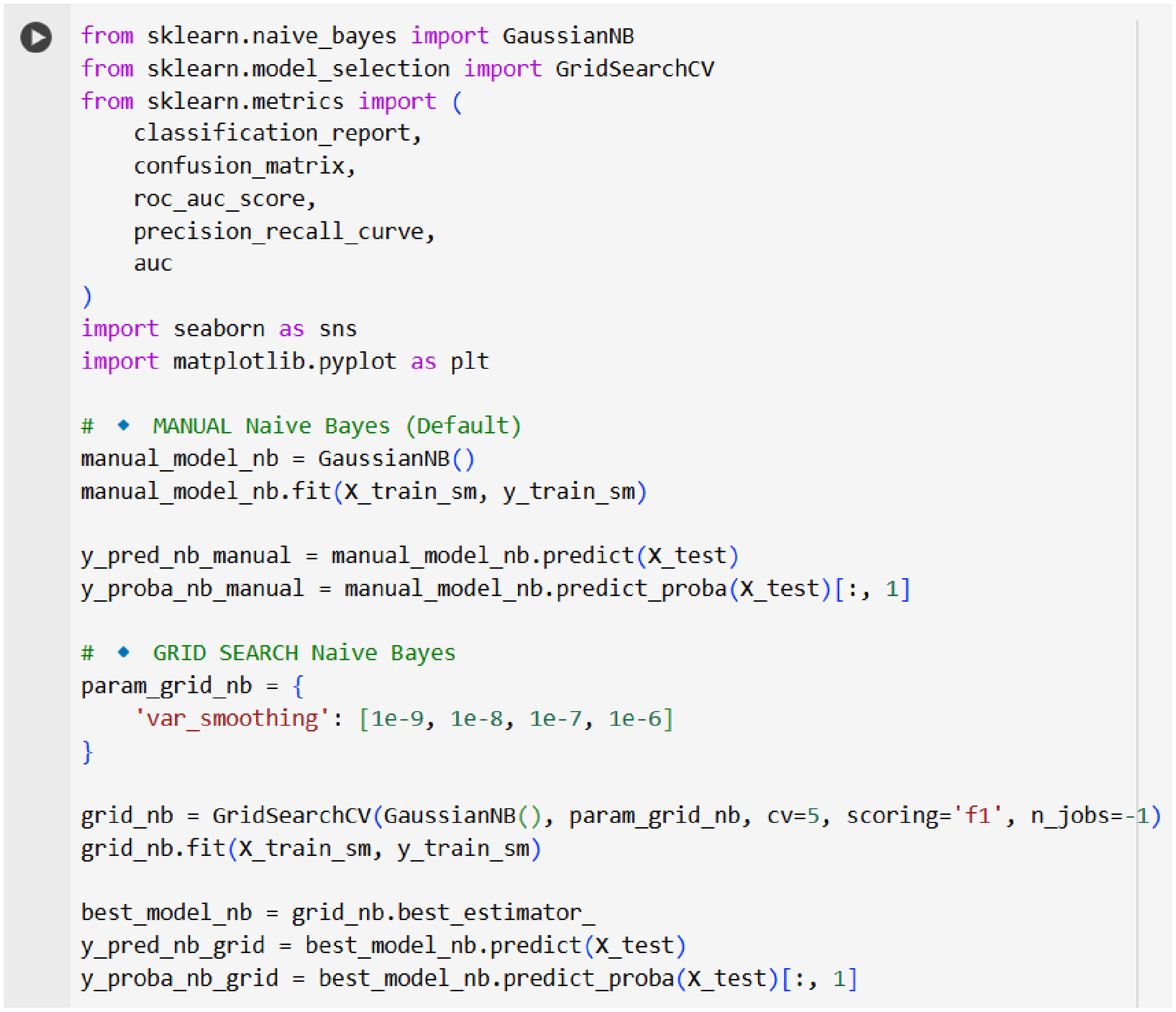
```python
# MANUAL EVALUATION
report_lr_manual = classification_report(y_test, y_pred_lr_manual, output_dict=True)
recall_lr_manual = report_lr_manual['1']['recall']
precision_lr_manual = report_lr_manual['1']['precision']
f1_lr_manual = report_lr_manual['1']['f1-score']
roc_auc_lr_manual = roc_auc_score(y_test, y_proba_lr_manual)
precision_vals_lr_manual, recall_vals_lr_manual, _ = precision_recall_curve(y_test, y_proba_lr_manual)
pr_auc_lr_manual = auc(recall_vals_lr_manual, precision_vals_lr_manual)

# GRID SEARCH EVALUATION
report_lr_grid = classification_report(y_test, y_pred_lr_grid, output_dict=True)
recall_lr_grid = report_lr_grid['1']['recall']
precision_lr_grid = report_lr_grid['1']['precision']
f1_lr_grid = report_lr_grid['1']['f1-score']
roc_auc_lr_grid = roc_auc_score(y_test, y_proba_lr_grid)
precision_vals_lr_grid, recall_vals_lr_grid, _ = precision_recall_curve(y_test, y_proba_lr_grid)
pr_auc_lr_grid = auc(recall_vals_lr_grid, precision_vals_lr_grid)

#  Printing evaluation results
print("\n Manual Logistic Regression – Model Evaluation")
print(f"- Recall (for dead): {recall_lr_manual:.3f}")
print(f"- Precision (for dead): {precision_lr_manual:.3f}")
print(f"- F1-Score (for dead): {f1_lr_manual:.3f}")
print(f"- PR-AUC: {pr_auc_lr_manual:.3f}")
print(f"- ROC-AUC: {roc_auc_lr_manual:.3f}")

print("\n Grid Search Logistic Regression – Model Evaluation")
print(f"- Recall (for dead): {recall_lr_grid:.3f}")
print(f"- Precision (for dead): {precision_lr_grid:.3f}")
print(f"- F1-Score (for dead): {f1_lr_grid:.3f}")
print(f"- PR-AUC: {pr_auc_lr_grid:.3f}")
print(f"- ROC-AUC: {roc_auc_lr_grid:.3f}")
```

```
#  Best Parameters
print("\n Best Parameters from Grid Search:")
print(grid_lr.best_params_)

#  Confusion Matrix Comparison
fig, axs = plt.subplots(1, 2, figsize=(12, 5))
sns.heatmap(confusion_matrix(y_test, y_pred_lr_manual), annot=True, fmt='d', cmap='Reds', ax=axs[0])
axs[0].set_title('Manual Logistic Regression')
axs[0].set_xlabel('Predicted')
axs[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test, y_pred_lr_grid), annot=True, fmt='d', cmap='Blues', ax=axs[1])
axs[1].set_title('Grid Search Logistic Regression')
axs[1].set_xlabel('Predicted')
axs[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()
```

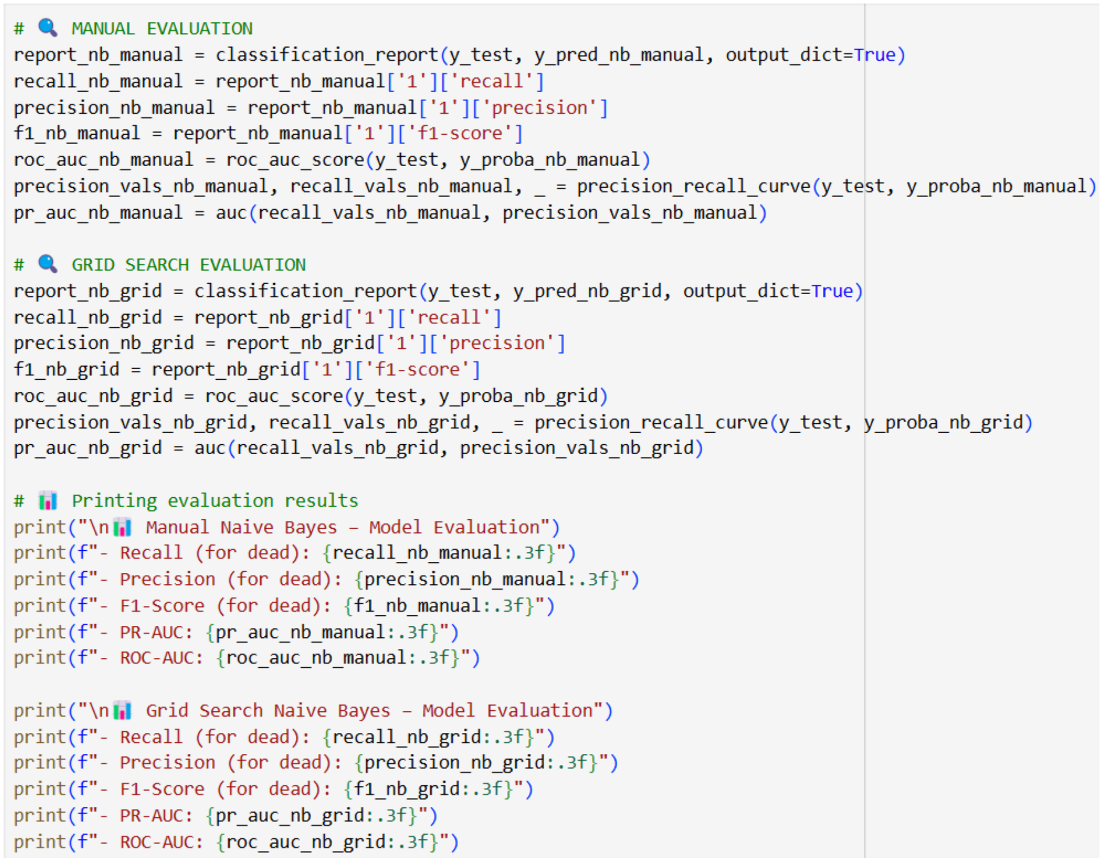
```
📊 Manual Logistic Regression – Model Evaluation
- Recall (for dead): 0.667
- Precision (for dead): 0.439
- F1-Score (for dead): 0.529
- PR-AUC: 0.592
- ROC-AUC: 0.840

📊 Grid Search Logistic Regression – Model Evaluation
- Recall (for dead): 0.667
- Precision (for dead): 0.446
- F1-Score (for dead): 0.534
- PR-AUC: 0.598
- ROC-AUC: 0.842

🛠 Best Parameters from Grid Search:
{'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
```



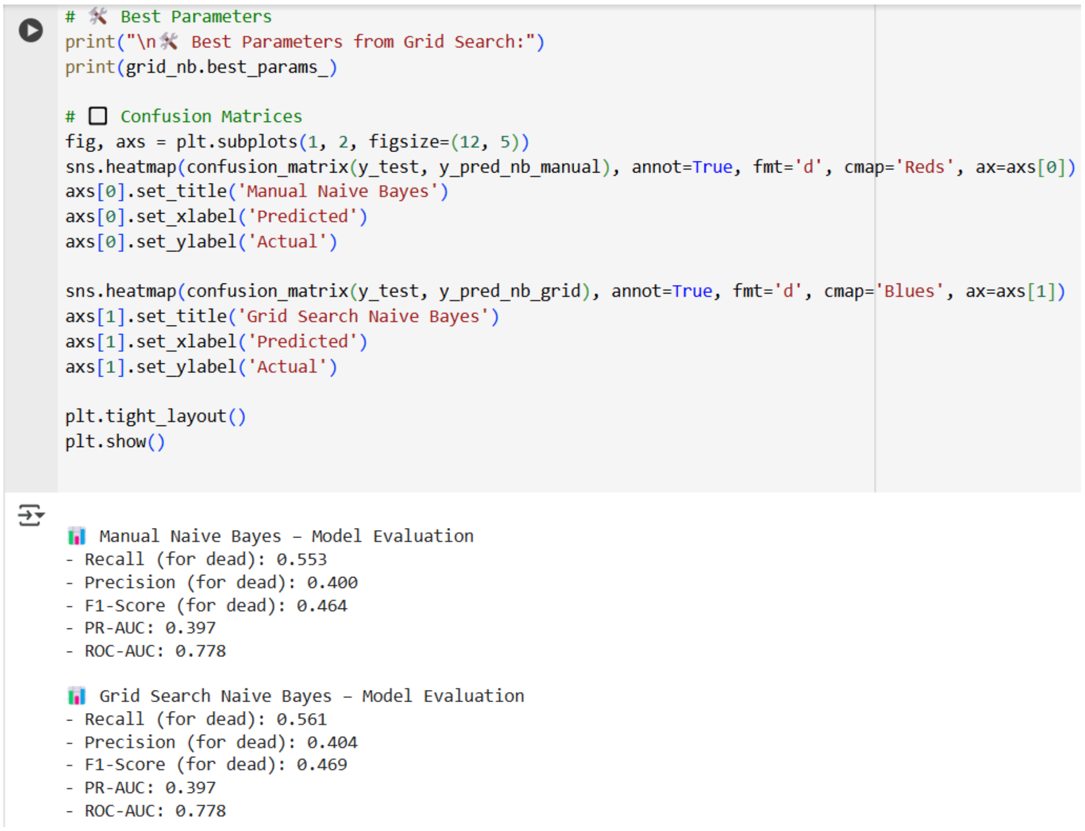
8.3 Naive Bayes - Gaussian Naive Bayes was implemented as a probabilistic classifier. The manual model was initially trained using default settings, while the tuned version utilized a parameter grid focusing on the var_smoothing value. Although Naive Bayes performed adequately in terms of precision and ROC-AUC, its recall remained relatively modest. The Grid Search model showed a slight improvement over the manual setup, but overall performance was moderate compared to ensemble-based methods.



```python
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    precision_recall_curve,
    auc
)
import seaborn as sns
import matplotlib.pyplot as plt

# 🔹 MANUAL Naive Bayes (Default)
manual_model_nb = GaussianNB()
manual_model_nb.fit(X_train_sm, y_train_sm)

y_pred_nb_manual = manual_model_nb.predict(X_test)
y_proba_nb_manual = manual_model_nb.predict_proba(X_test)[:, 1]

# 🔹 GRID SEARCH Naive Bayes
param_grid_nb = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6]
}

grid_nb = GridSearchCV(GaussianNB(), param_grid_nb, cv=5, scoring='f1', n_jobs=-1)
grid_nb.fit(X_train_sm, y_train_sm)

best_model_nb = grid_nb.best_estimator_
y_pred_nb_grid = best_model_nb.predict(X_test)
y_proba_nb_grid = best_model_nb.predict_proba(X_test)[:, 1]
```



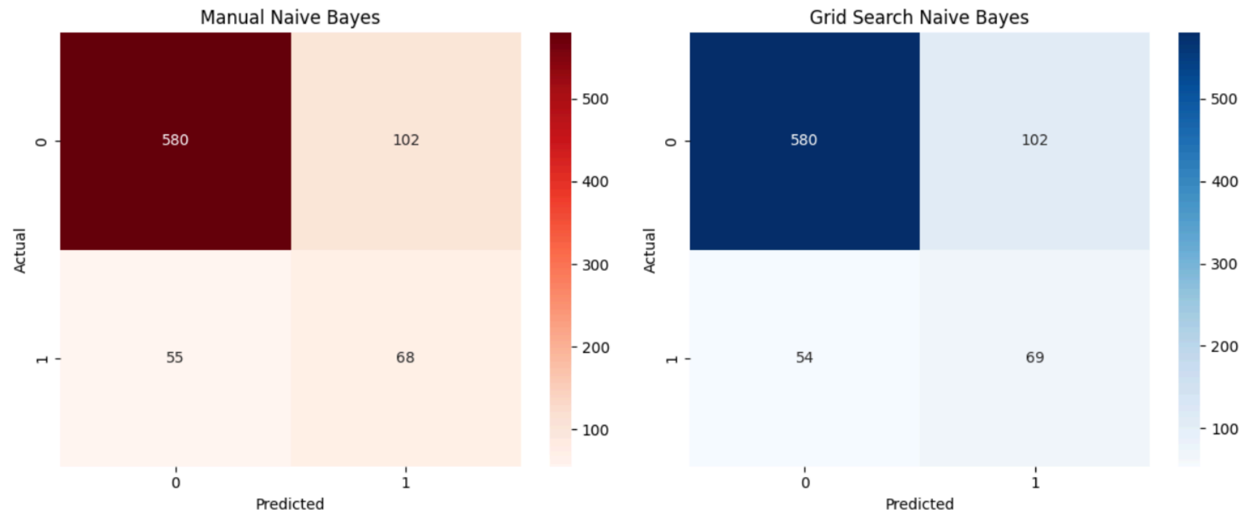
```
# 🔍 MANUAL EVALUATION
report_nb_manual = classification_report(y_test, y_pred_nb_manual, output_dict=True)
recall_nb_manual = report_nb_manual['1']['recall']
precision_nb_manual = report_nb_manual['1']['precision']
f1_nb_manual = report_nb_manual['1']['f1-score']
roc_auc_nb_manual = roc_auc_score(y_test, y_proba_nb_manual)
precision_vals_nb_manual, recall_vals_nb_manual, _ = precision_recall_curve(y_test, y_proba_nb_manual)
pr_auc_nb_manual = auc(recall_vals_nb_manual, precision_vals_nb_manual)

# 🔍 GRID SEARCH EVALUATION
report_nb_grid = classification_report(y_test, y_pred_nb_grid, output_dict=True)
recall_nb_grid = report_nb_grid['1']['recall']
precision_nb_grid = report_nb_grid['1']['precision']
f1_nb_grid = report_nb_grid['1']['f1-score']
roc_auc_nb_grid = roc_auc_score(y_test, y_proba_nb_grid)
precision_vals_nb_grid, recall_vals_nb_grid, _ = precision_recall_curve(y_test, y_proba_nb_grid)
pr_auc_nb_grid = auc(recall_vals_nb_grid, precision_vals_nb_grid)

# 📊 Printing evaluation results
print("\n📊 Manual Naive Bayes – Model Evaluation")
print(f"- Recall (for dead): {recall_nb_manual:.3f}")
print(f"- Precision (for dead): {precision_nb_manual:.3f}")
print(f"- F1-Score (for dead): {f1_nb_manual:.3f}")
print(f"- PR-AUC: {pr_auc_nb_manual:.3f}")
print(f"- ROC-AUC: {roc_auc_nb_manual:.3f}")

print("\n📊 Grid Search Naive Bayes – Model Evaluation")
print(f"- Recall (for dead): {recall_nb_grid:.3f}")
print(f"- Precision (for dead): {precision_nb_grid:.3f}")
print(f"- F1-Score (for dead): {f1_nb_grid:.3f}")
print(f"- PR-AUC: {pr_auc_nb_grid:.3f}")
print(f"- ROC-AUC: {roc_auc_nb_grid:.3f}")
```



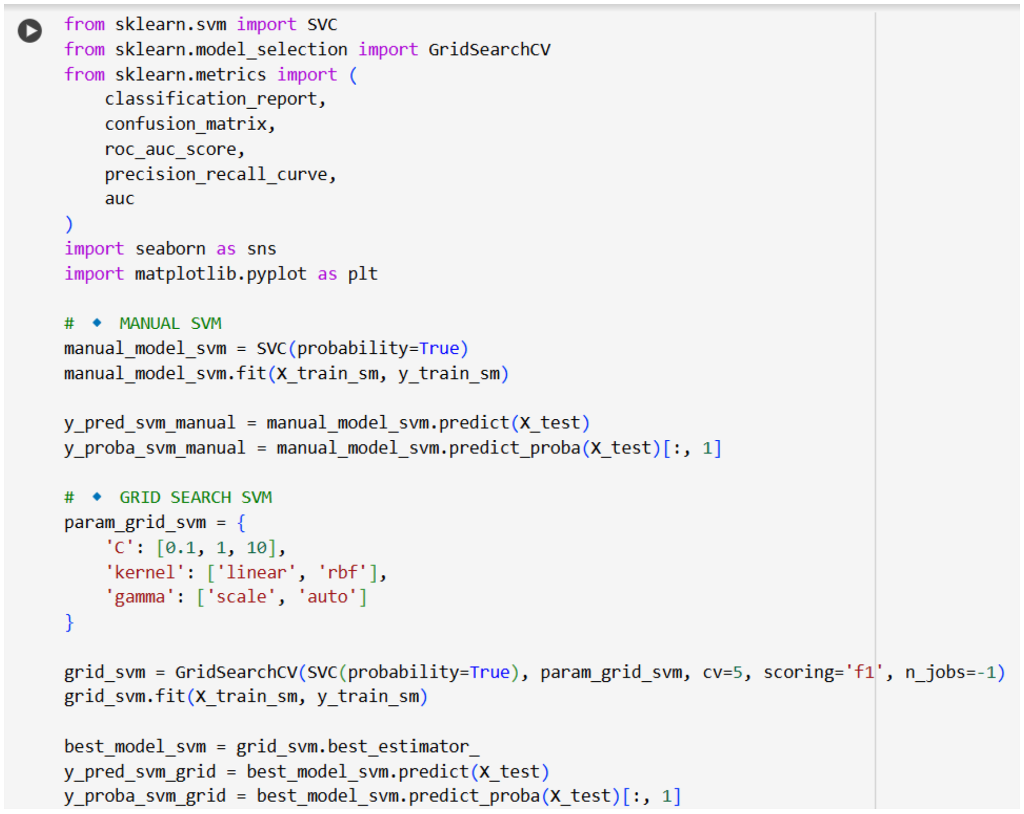
```
# 🛠 Best Parameters
print("\n🛠 Best Parameters from Grid Search:")
print(grid_nb.best_params_)

# □ Confusion Matrices
fig, axs = plt.subplots(1, 2, figsize=(12, 5))
sns.heatmap(confusion_matrix(y_test, y_pred_nb_manual), annot=True, fmt='d', cmap='Reds', ax=axs[0])
axs[0].set_title('Manual Naive Bayes')
axs[0].set_xlabel('Predicted')
axs[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test, y_pred_nb_grid), annot=True, fmt='d', cmap='Blues', ax=axs[1])
axs[1].set_title('Grid Search Naive Bayes')
axs[1].set_xlabel('Predicted')
axs[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()
```

```
📊 Manual Naive Bayes – Model Evaluation
- Recall (for dead): 0.553
- Precision (for dead): 0.400
- F1-Score (for dead): 0.464
- PR-AUC: 0.397
- ROC-AUC: 0.778

📊 Grid Search Naive Bayes – Model Evaluation
- Recall (for dead): 0.561
- Precision (for dead): 0.404
- F1-Score (for dead): 0.469
- PR-AUC: 0.397
- ROC-AUC: 0.778
```

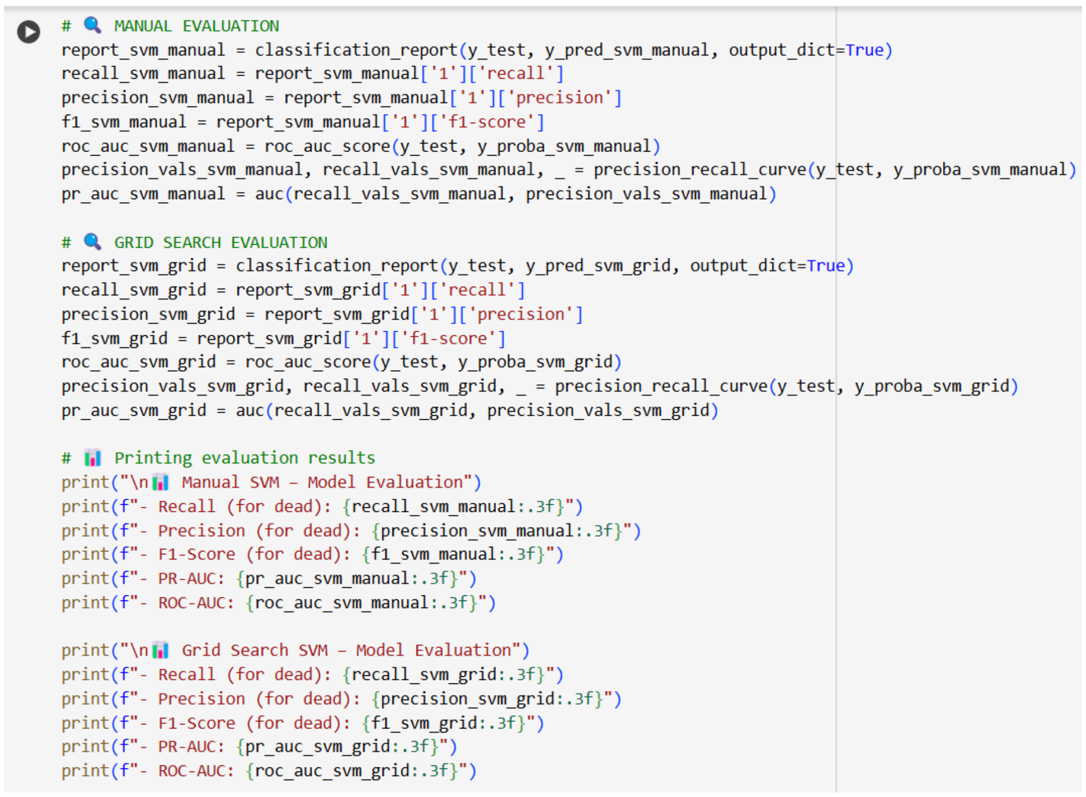
```
🛠 Best Parameters from Grid Search:
{'var_smoothing': 1e-06}
```



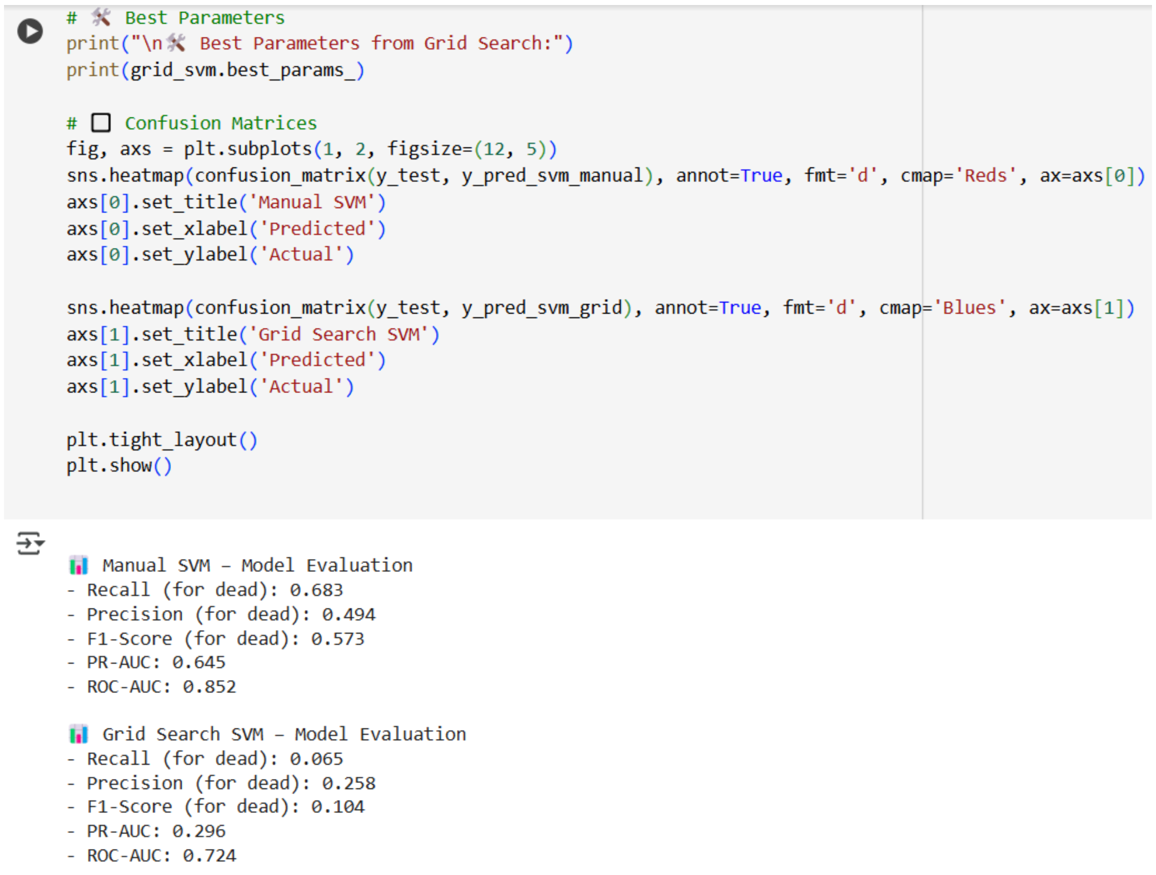
8.4 Support Vector Machine (SVM) - SVM was configured to allow probability predictions (probability=True) and trained with both manual and tuned setups. The manual model used an RBF kernel with default regularization, while the grid search evaluated combinations of kernel types, C values, and gamma settings. Despite good ROC-AUC values, the SVM model showed poor recall and F1-Score, especially in detecting the minority class (Dead). The confusion matrix confirmed a high number of false negatives, rendering SVM less effective for this task.



```python
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    precision_recall_curve,
    auc
)
import seaborn as sns
import matplotlib.pyplot as plt

# 🔹 MANUAL SVM
manual_model_svm = SVC(probability=True)
manual_model_svm.fit(X_train_sm, y_train_sm)

y_pred_svm_manual = manual_model_svm.predict(X_test)
y_proba_svm_manual = manual_model_svm.predict_proba(X_test)[:, 1]

# 🔹 GRID SEARCH SVM
param_grid_svm = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

grid_svm = GridSearchCV(SVC(probability=True), param_grid_svm, cv=5, scoring='f1', n_jobs=-1)
grid_svm.fit(X_train_sm, y_train_sm)

best_model_svm = grid_svm.best_estimator_
y_pred_svm_grid = best_model_svm.predict(X_test)
y_proba_svm_grid = best_model_svm.predict_proba(X_test)[:, 1]
```



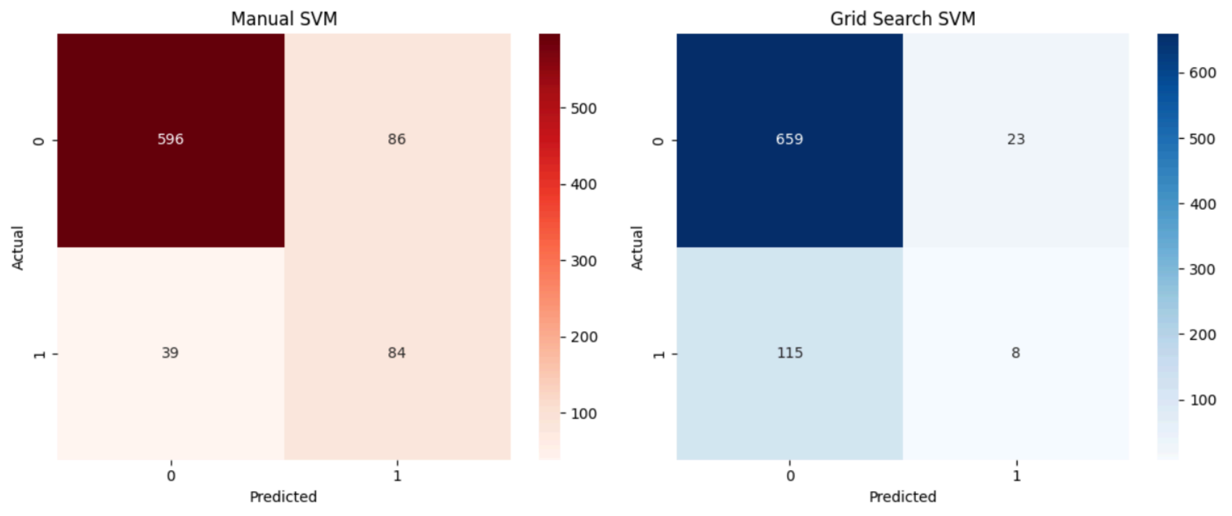
```python
# 🔍 MANUAL EVALUATION
report_svm_manual = classification_report(y_test, y_pred_svm_manual, output_dict=True)
recall_svm_manual = report_svm_manual['1']['recall']
precision_svm_manual = report_svm_manual['1']['precision']
f1_svm_manual = report_svm_manual['1']['f1-score']
roc_auc_svm_manual = roc_auc_score(y_test, y_proba_svm_manual)
precision_vals_svm_manual, recall_vals_svm_manual, _ = precision_recall_curve(y_test, y_proba_svm_manual)
pr_auc_svm_manual = auc(recall_vals_svm_manual, precision_vals_svm_manual)

# 🔍 GRID SEARCH EVALUATION
report_svm_grid = classification_report(y_test, y_pred_svm_grid, output_dict=True)
recall_svm_grid = report_svm_grid['1']['recall']
precision_svm_grid = report_svm_grid['1']['precision']
f1_svm_grid = report_svm_grid['1']['f1-score']
roc_auc_svm_grid = roc_auc_score(y_test, y_proba_svm_grid)
precision_vals_svm_grid, recall_vals_svm_grid, _ = precision_recall_curve(y_test, y_proba_svm_grid)
pr_auc_svm_grid = auc(recall_vals_svm_grid, precision_vals_svm_grid)

# 📊 Printing evaluation results
print("\n📊 Manual SVM – Model Evaluation")
print(f"- Recall (for dead): {recall_svm_manual:.3f}")
print(f"- Precision (for dead): {precision_svm_manual:.3f}")
print(f"- F1-Score (for dead): {f1_svm_manual:.3f}")
print(f"- PR-AUC: {pr_auc_svm_manual:.3f}")
print(f"- ROC-AUC: {roc_auc_svm_manual:.3f}")

print("\n📊 Grid Search SVM – Model Evaluation")
print(f"- Recall (for dead): {recall_svm_grid:.3f}")
print(f"- Precision (for dead): {precision_svm_grid:.3f}")
print(f"- F1-Score (for dead): {f1_svm_grid:.3f}")
print(f"- PR-AUC: {pr_auc_svm_grid:.3f}")
print(f"- ROC-AUC: {roc_auc_svm_grid:.3f}")
```



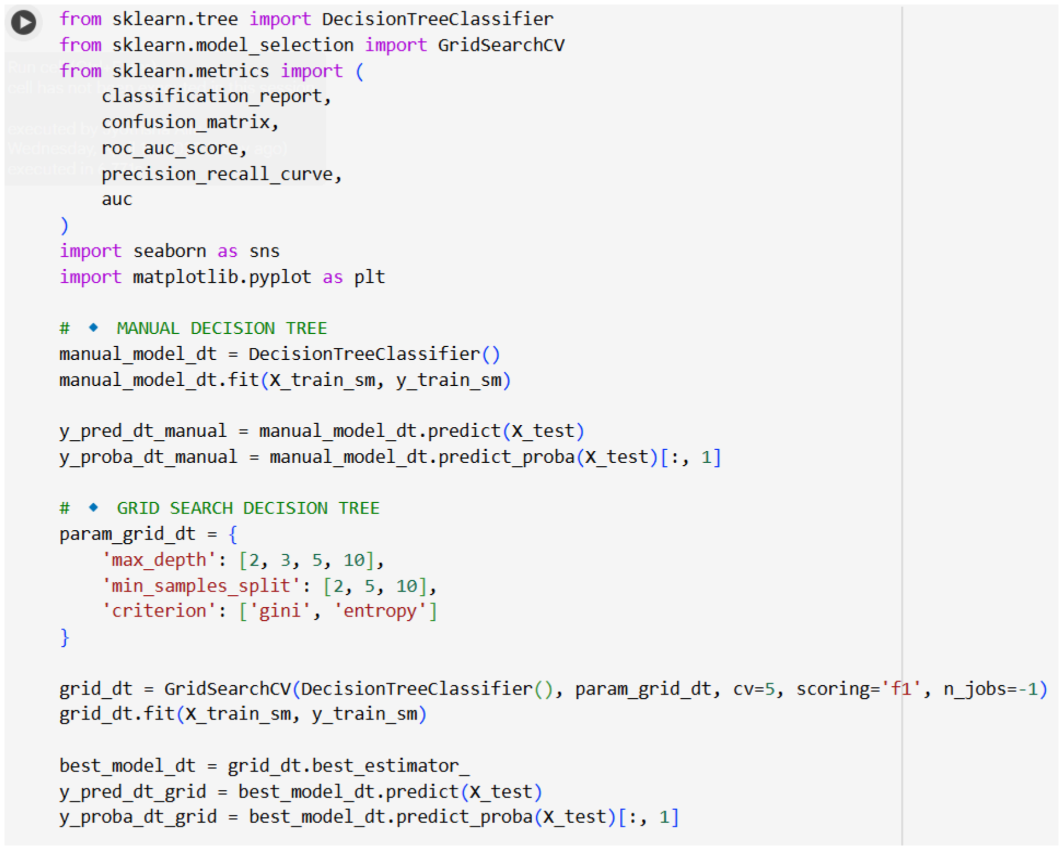
```python
# 🛠 Best Parameters
print("\n🛠 Best Parameters from Grid Search:")
print(grid_svm.best_params_)

# □ Confusion Matrices
fig, axs = plt.subplots(1, 2, figsize=(12, 5))
sns.heatmap(confusion_matrix(y_test, y_pred_svm_manual), annot=True, fmt='d', cmap='Reds', ax=axs[0])
axs[0].set_title('Manual SVM')
axs[0].set_xlabel('Predicted')
axs[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test, y_pred_svm_grid), annot=True, fmt='d', cmap='Blues', ax=axs[1])
axs[1].set_title('Grid Search SVM')
axs[1].set_xlabel('Predicted')
axs[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()
```

```
📊 Manual SVM – Model Evaluation
- Recall (for dead): 0.683
- Precision (for dead): 0.494
- F1-Score (for dead): 0.573
- PR-AUC: 0.645
- ROC-AUC: 0.852

📊 Grid Search SVM – Model Evaluation
- Recall (for dead): 0.065
- Precision (for dead): 0.258
- F1-Score (for dead): 0.104
- PR-AUC: 0.296
- ROC-AUC: 0.724
```

```
🛠 Best Parameters from Grid Search:
{'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
```



8.5 Decision Tree - Decision Tree models were built using Gini and Entropy criteria with a range of depths and split thresholds. The manual version used default settings, while the Grid Search model selected optimal depth and splitting parameters. Performance metrics showed that the tuned model improved slightly in F1-Score and precision, but Decision Trees generally exhibited lower generalization capability than more complex ensemble methods. The tuned tree remained interpretable but outperformed other models in most metrics.



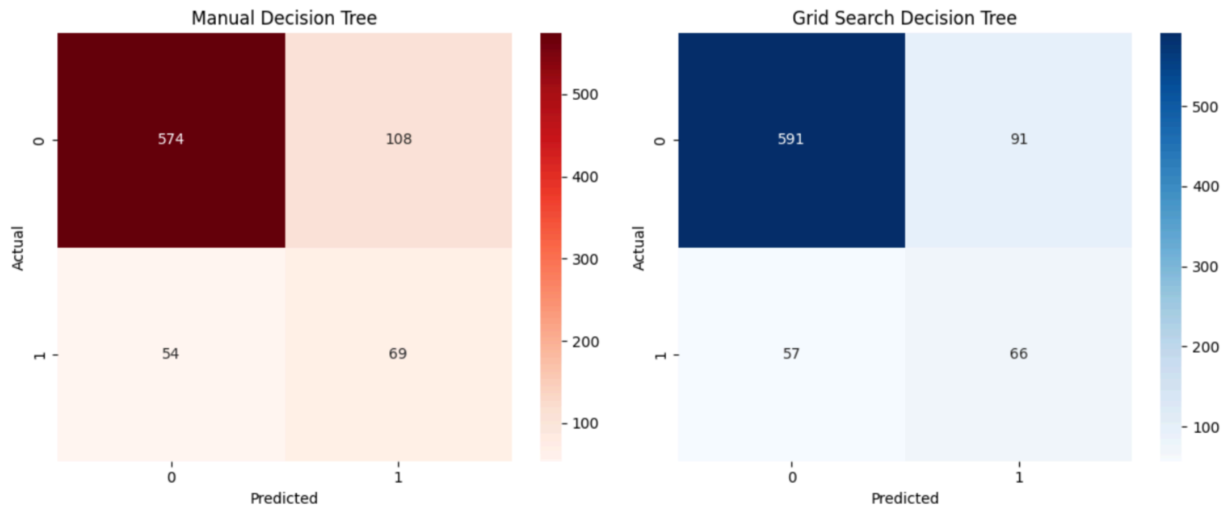
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    precision_recall_curve,
    auc
)
import seaborn as sns
import matplotlib.pyplot as plt

# 🔹 MANUAL DECISION TREE
manual_model_dt = DecisionTreeClassifier()
manual_model_dt.fit(X_train_sm, y_train_sm)

y_pred_dt_manual = manual_model_dt.predict(X_test)
y_proba_dt_manual = manual_model_dt.predict_proba(X_test)[:, 1]

# 🔹 GRID SEARCH DECISION TREE
param_grid_dt = {
    'max_depth': [2, 3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']
}

grid_dt = GridSearchCV(DecisionTreeClassifier(), param_grid_dt, cv=5, scoring='f1', n_jobs=-1)
grid_dt.fit(X_train_sm, y_train_sm)

best_model_dt = grid_dt.best_estimator_
y_pred_dt_grid = best_model_dt.predict(X_test)
y_proba_dt_grid = best_model_dt.predict_proba(X_test)[:, 1]
```

```python
# 🔍 MANUAL EVALUATION
report_dt_manual = classification_report(y_test, y_pred_dt_manual, output_dict=True)
recall_dt_manual = report_dt_manual['1']['recall']
precision_dt_manual = report_dt_manual['1']['precision']
f1_dt_manual = report_dt_manual['1']['f1-score']
roc_auc_dt_manual = roc_auc_score(y_test, y_proba_dt_manual)
precision_vals_dt_manual, recall_vals_dt_manual, _ = precision_recall_curve(y_test, y_proba_dt_manual)
pr_auc_dt_manual = auc(recall_vals_dt_manual, precision_vals_dt_manual)

# 🔍 GRID SEARCH EVALUATION
report_dt_grid = classification_report(y_test, y_pred_dt_grid, output_dict=True)
recall_dt_grid = report_dt_grid['1']['recall']
precision_dt_grid = report_dt_grid['1']['precision']
f1_dt_grid = report_dt_grid['1']['f1-score']
roc_auc_dt_grid = roc_auc_score(y_test, y_proba_dt_grid)
precision_vals_dt_grid, recall_vals_dt_grid, _ = precision_recall_curve(y_test, y_proba_dt_grid)
pr_auc_dt_grid = auc(recall_vals_dt_grid, precision_vals_dt_grid)

# 📊 Printing evaluation results
print("\n📊 Manual Decision Tree – Model Evaluation")
print(f"- Recall (for dead): {recall_dt_manual:.3f}")
print(f"- Precision (for dead): {precision_dt_manual:.3f}")
print(f"- F1-Score (for dead): {f1_dt_manual:.3f}")
print(f"- PR-AUC: {pr_auc_dt_manual:.3f}")
print(f"- ROC-AUC: {roc_auc_dt_manual:.3f}")

print("\n📊 Grid Search Decision Tree – Model Evaluation")
print(f"- Recall (for dead): {recall_dt_grid:.3f}")
print(f"- Precision (for dead): {precision_dt_grid:.3f}")
print(f"- F1-Score (for dead): {f1_dt_grid:.3f}")
print(f"- PR-AUC: {pr_auc_dt_grid:.3f}")
print(f"- ROC-AUC: {roc_auc_dt_grid:.3f}")
```

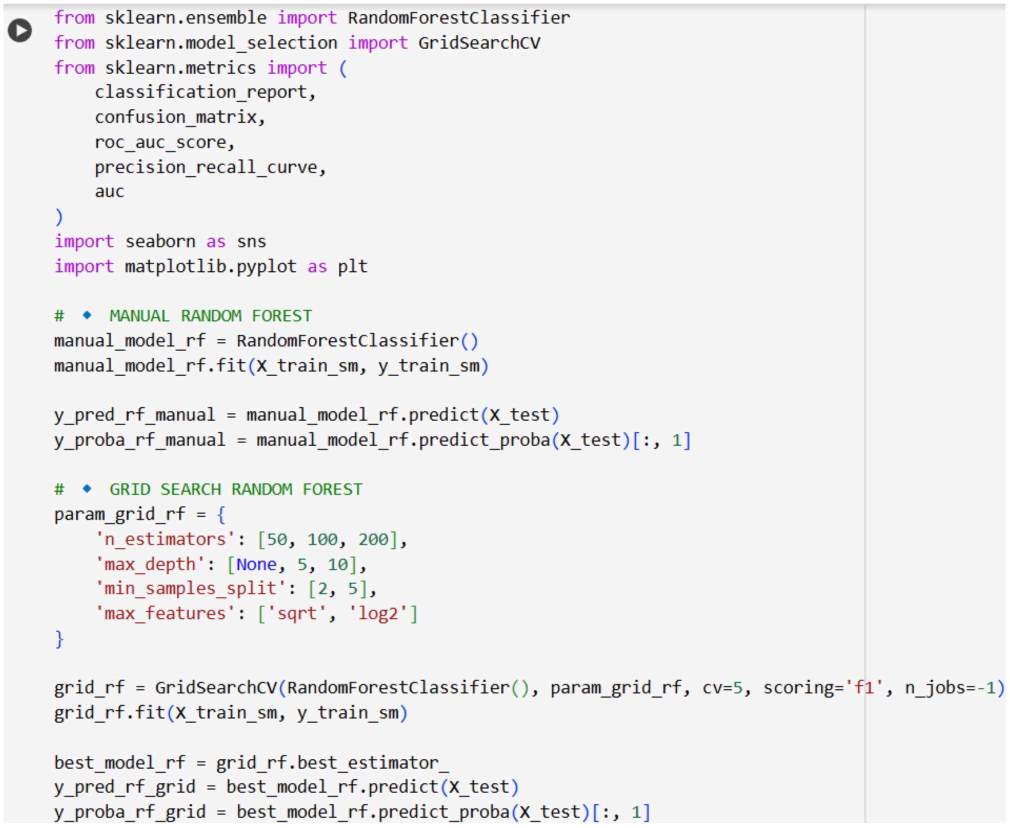
```python
# 🛠 Best Parameters
print("\n🛠 Best Parameters from Grid Search:")
print(grid_dt.best_params_)

# 📊 Confusion Matrix Comparison
fig, axs = plt.subplots(1, 2, figsize=(12, 5))
sns.heatmap(confusion_matrix(y_test, y_pred_dt_manual), annot=True, fmt='d', cmap='Reds', ax=axs[0])
axs[0].set_title('Manual Decision Tree')
axs[0].set_xlabel('Predicted')
axs[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test, y_pred_dt_grid), annot=True, fmt='d', cmap='Blues', ax=axs[1])
axs[1].set_title('Grid Search Decision Tree')
axs[1].set_xlabel('Predicted')
axs[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()
```

```
📊 Manual Decision Tree – Model Evaluation
- Recall (for dead): 0.561
- Precision (for dead): 0.390
- F1-Score (for dead): 0.460
- PR-AUC: 0.509
- ROC-AUC: 0.701

📊 Grid Search Decision Tree – Model Evaluation
- Recall (for dead): 0.537
- Precision (for dead): 0.420
- F1-Score (for dead): 0.471
- PR-AUC: 0.479
- ROC-AUC: 0.710
```

```
🛠 Best Parameters from Grid Search:
{'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 2}
```



8.6 Random Forest - Random Forest, an ensemble-based classifier, performed strongly across all evaluation metrics. The manual model was trained using default settings, while the tuned model optimized the number of estimators, maximum depth, minimum split size, and feature selection strategies (sqrt, log2). The grid-tuned model demonstrated the best F1-Score and highest precision among all models. Its confusion matrix reflected a balanced prediction performance with fewer false positives and negatives, making it the most robust and reliable model in this study.



```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    precision_recall_curve,
    auc
)
import seaborn as sns
import matplotlib.pyplot as plt

# 🔹 MANUAL RANDOM FOREST
manual_model_rf = RandomForestClassifier()
manual_model_rf.fit(X_train_sm, y_train_sm)

y_pred_rf_manual = manual_model_rf.predict(X_test)
y_proba_rf_manual = manual_model_rf.predict_proba(X_test)[:, 1]

# 🔹 GRID SEARCH RANDOM FOREST
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5],
    'max_features': ['sqrt', 'log2']
}

grid_rf = GridSearchCV(RandomForestClassifier(), param_grid_rf, cv=5, scoring='f1', n_jobs=-1)
grid_rf.fit(X_train_sm, y_train_sm)

best_model_rf = grid_rf.best_estimator_
y_pred_rf_grid = best_model_rf.predict(X_test)
y_proba_rf_grid = best_model_rf.predict_proba(X_test)[:, 1]
```

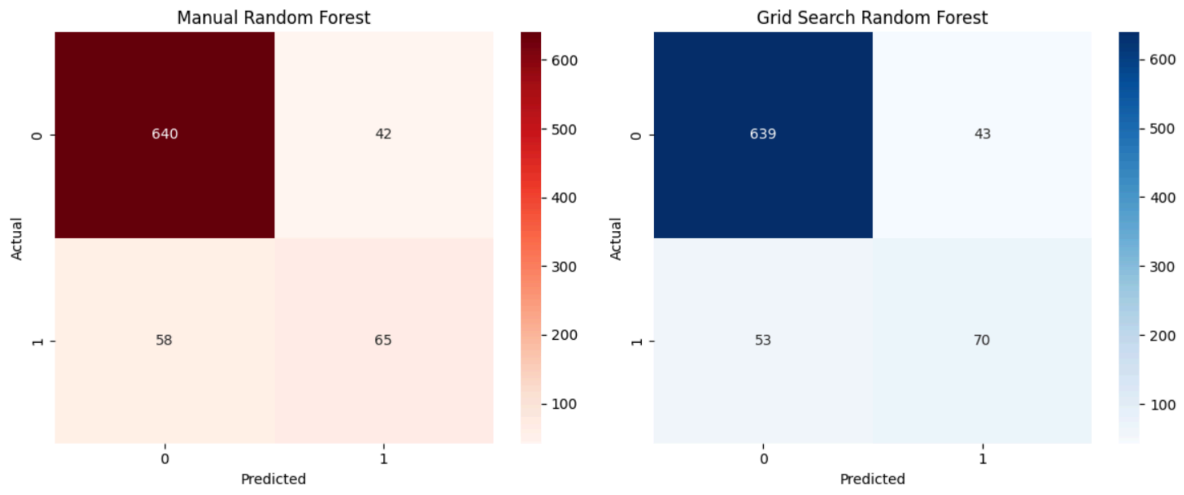
```python
# 🔍 MANUAL EVALUATION
report_rf_manual = classification_report(y_test, y_pred_rf_manual, output_dict=True)
recall_rf_manual = report_rf_manual['1']['recall']
precision_rf_manual = report_rf_manual['1']['precision']
f1_rf_manual = report_rf_manual['1']['f1-score']
roc_auc_rf_manual = roc_auc_score(y_test, y_proba_rf_manual)
precision_vals_rf_manual, recall_vals_rf_manual, _ = precision_recall_curve(y_test, y_proba_rf_manual)
pr_auc_rf_manual = auc(recall_vals_rf_manual, precision_vals_rf_manual)

# 🔍 GRID SEARCH EVALUATION
report_rf_grid = classification_report(y_test, y_pred_rf_grid, output_dict=True)
recall_rf_grid = report_rf_grid['1']['recall']
precision_rf_grid = report_rf_grid['1']['precision']
f1_rf_grid = report_rf_grid['1']['f1-score']
roc_auc_rf_grid = roc_auc_score(y_test, y_proba_rf_grid)
precision_vals_rf_grid, recall_vals_rf_grid, _ = precision_recall_curve(y_test, y_proba_rf_grid)
pr_auc_rf_grid = auc(recall_vals_rf_grid, precision_vals_rf_grid)

# 📊 Printing evaluation results
print("\n📊 Manual Random Forest – Model Evaluation")
print(f"- Recall (for dead): {recall_rf_manual:.3f}")
print(f"- Precision (for dead): {precision_rf_manual:.3f}")
print(f"- F1-Score (for dead): {f1_rf_manual:.3f}")
print(f"- PR-AUC: {pr_auc_rf_manual:.3f}")
print(f"- ROC-AUC: {roc_auc_rf_manual:.3f}")

print("\n📊 Grid Search Random Forest – Model Evaluation")
print(f"- Recall (for dead): {recall_rf_grid:.3f}")
print(f"- Precision (for dead): {precision_rf_grid:.3f}")
print(f"- F1-Score (for dead): {f1_rf_grid:.3f}")
print(f"- PR-AUC: {pr_auc_rf_grid:.3f}")
print(f"- ROC-AUC: {roc_auc_rf_grid:.3f}")
```

[ ]

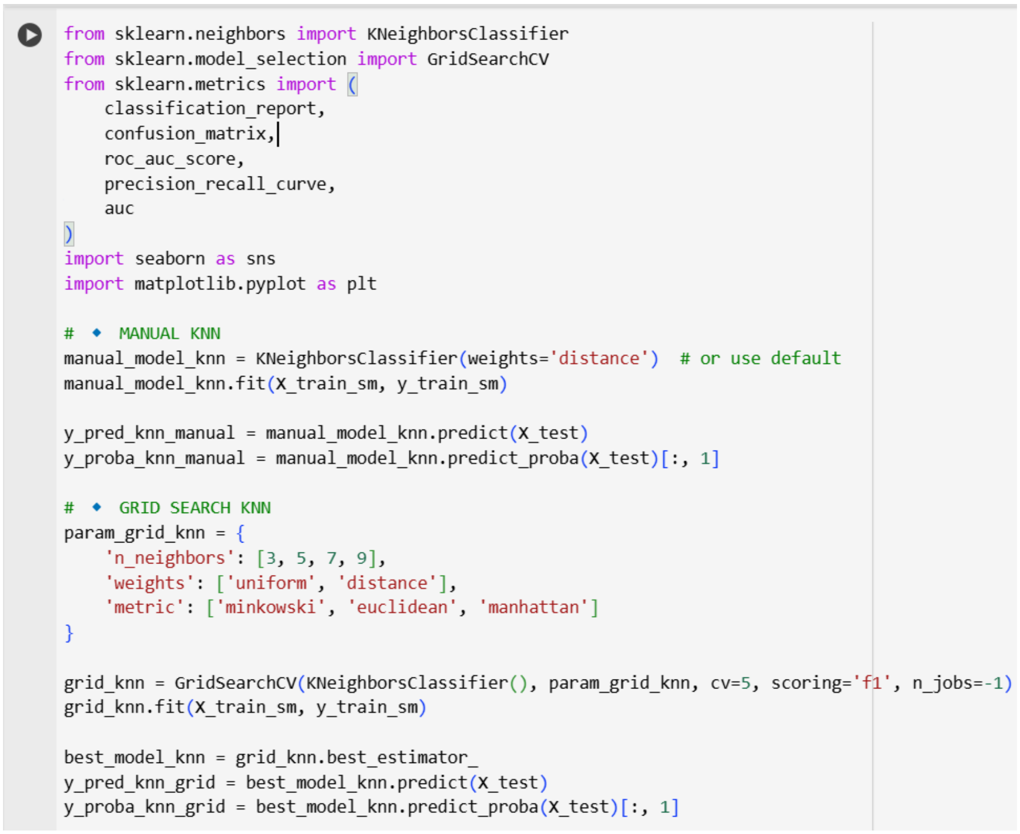
```python
# 🛠 Best Parameters from Grid Search
print("\n🛠 Best Parameters from Grid Search:")
print(grid_rf.best_params_)

# 📊 Confusion Matrix Comparison
fig, axs = plt.subplots(1, 2, figsize=(12, 5))
sns.heatmap(confusion_matrix(y_test, y_pred_rf_manual), annot=True, fmt='d', cmap='Reds', ax=axs[0])
axs[0].set_title('Manual Random Forest')
axs[0].set_xlabel('Predicted')
axs[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test, y_pred_rf_grid), annot=True, fmt='d', cmap='Blues', ax=axs[1])
axs[1].set_title('Grid Search Random Forest')
axs[1].set_xlabel('Predicted')
axs[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()
```

```
📊 Manual Random Forest – Model Evaluation
- Recall (for dead): 0.528
- Precision (for dead): 0.607
- F1-Score (for dead): 0.565
- PR-AUC: 0.596
- ROC-AUC: 0.841

📊 Grid Search Random Forest – Model Evaluation
- Recall (for dead): 0.569
- Precision (for dead): 0.619
- F1-Score (for dead): 0.593
- PR-AUC: 0.629
- ROC-AUC: 0.840
```

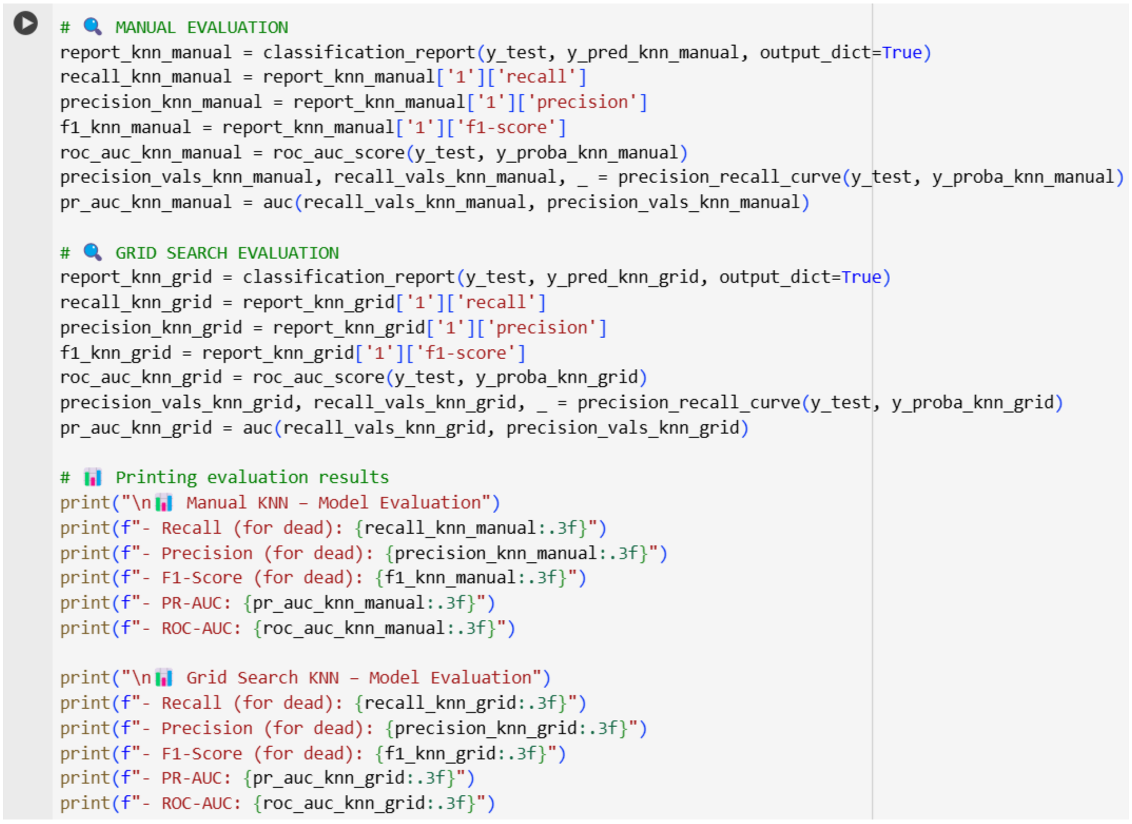
```
🛠 Best Parameters from Grid Search:
{'max_depth': None, 'max_features': 'log2', 'min_samples_split': 2, 'n_estimators': 50}
```



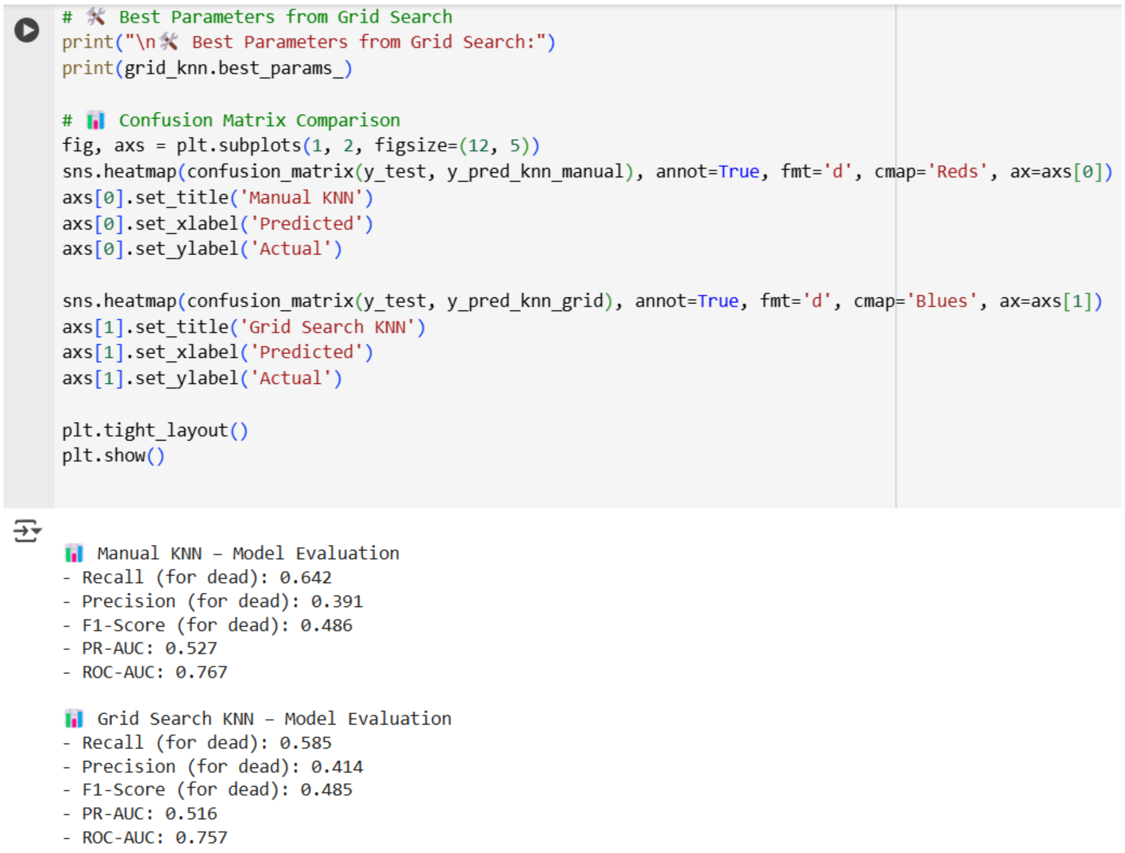
8.7 K-Nearest Neighbors (KNN) - KNN was tested with both uniform and distance-based weighting, and the number of neighbors was varied using Grid Search. The model's performance improved slightly post-tuning, with modest gains in F1-Score and ROC-AUC. However, KNN struggled to achieve the high recall levels necessary for sensitive medical predictions, and its performance was highly sensitive to feature scaling and distance metrics. While it performed better than SVM, it lagged behind Random Forest and Logistic Regression in clinical usefulness.



```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    precision_recall_curve,
    auc
)
import seaborn as sns
import matplotlib.pyplot as plt

# 🔹 MANUAL KNN
manual_model_knn = KNeighborsClassifier(weights='distance')  # or use default
manual_model_knn.fit(X_train_sm, y_train_sm)

y_pred_knn_manual = manual_model_knn.predict(X_test)
y_proba_knn_manual = manual_model_knn.predict_proba(X_test)[:, 1]

# 🔹 GRID SEARCH KNN
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['minkowski', 'euclidean', 'manhattan']
}

grid_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5, scoring='f1', n_jobs=-1)
grid_knn.fit(X_train_sm, y_train_sm)

best_model_knn = grid_knn.best_estimator_
y_pred_knn_grid = best_model_knn.predict(X_test)
y_proba_knn_grid = best_model_knn.predict_proba(X_test)[:, 1]
```



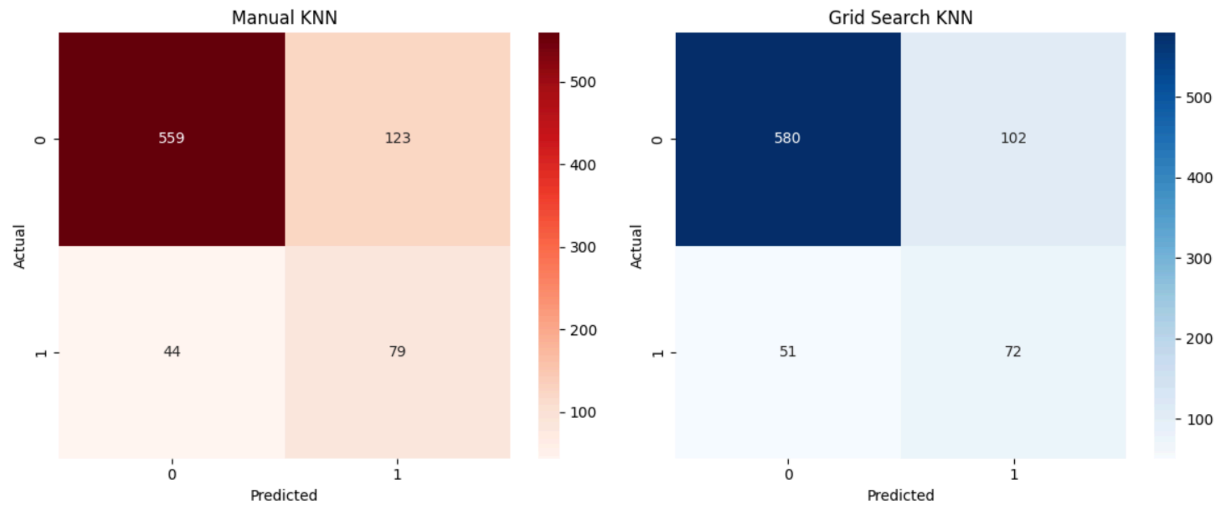
```python
# 🔍 MANUAL EVALUATION
report_knn_manual = classification_report(y_test, y_pred_knn_manual, output_dict=True)
recall_knn_manual = report_knn_manual['1']['recall']
precision_knn_manual = report_knn_manual['1']['precision']
f1_knn_manual = report_knn_manual['1']['f1-score']
roc_auc_knn_manual = roc_auc_score(y_test, y_proba_knn_manual)
precision_vals_knn_manual, recall_vals_knn_manual, _ = precision_recall_curve(y_test, y_proba_knn_manual)
pr_auc_knn_manual = auc(recall_vals_knn_manual, precision_vals_knn_manual)

# 🔍 GRID SEARCH EVALUATION
report_knn_grid = classification_report(y_test, y_pred_knn_grid, output_dict=True)
recall_knn_grid = report_knn_grid['1']['recall']
precision_knn_grid = report_knn_grid['1']['precision']
f1_knn_grid = report_knn_grid['1']['f1-score']
roc_auc_knn_grid = roc_auc_score(y_test, y_proba_knn_grid)
precision_vals_knn_grid, recall_vals_knn_grid, _ = precision_recall_curve(y_test, y_proba_knn_grid)
pr_auc_knn_grid = auc(recall_vals_knn_grid, precision_vals_knn_grid)

# 📊 Printing evaluation results
print("\n📊 Manual KNN – Model Evaluation")
print(f"- Recall (for dead): {recall_knn_manual:.3f}")
print(f"- Precision (for dead): {precision_knn_manual:.3f}")
print(f"- F1-Score (for dead): {f1_knn_manual:.3f}")
print(f"- PR-AUC: {pr_auc_knn_manual:.3f}")
print(f"- ROC-AUC: {roc_auc_knn_manual:.3f}")

print("\n📊 Grid Search KNN – Model Evaluation")
print(f"- Recall (for dead): {recall_knn_grid:.3f}")
print(f"- Precision (for dead): {precision_knn_grid:.3f}")
print(f"- F1-Score (for dead): {f1_knn_grid:.3f}")
print(f"- PR-AUC: {pr_auc_knn_grid:.3f}")
print(f"- ROC-AUC: {roc_auc_knn_grid:.3f}")
```



```python
# 🛠 Best Parameters from Grid Search
print("\n🛠 Best Parameters from Grid Search:")
print(grid_knn.best_params_)

# 📊 Confusion Matrix Comparison
fig, axs = plt.subplots(1, 2, figsize=(12, 5))
sns.heatmap(confusion_matrix(y_test, y_pred_knn_manual), annot=True, fmt='d', cmap='Reds', ax=axs[0])
axs[0].set_title('Manual KNN')
axs[0].set_xlabel('Predicted')
axs[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test, y_pred_knn_grid), annot=True, fmt='d', cmap='Blues', ax=axs[1])
axs[1].set_title('Grid Search KNN')
axs[1].set_xlabel('Predicted')
axs[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()
```

```
📊 Manual KNN – Model Evaluation
- Recall (for dead): 0.642
- Precision (for dead): 0.391
- F1-Score (for dead): 0.486
- PR-AUC: 0.527
- ROC-AUC: 0.767

📊 Grid Search KNN – Model Evaluation
- Recall (for dead): 0.585
- Precision (for dead): 0.414
- F1-Score (for dead): 0.485
- PR-AUC: 0.516
- ROC-AUC: 0.757
```

```
🛠 Best Parameters from Grid Search:
{'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}
```



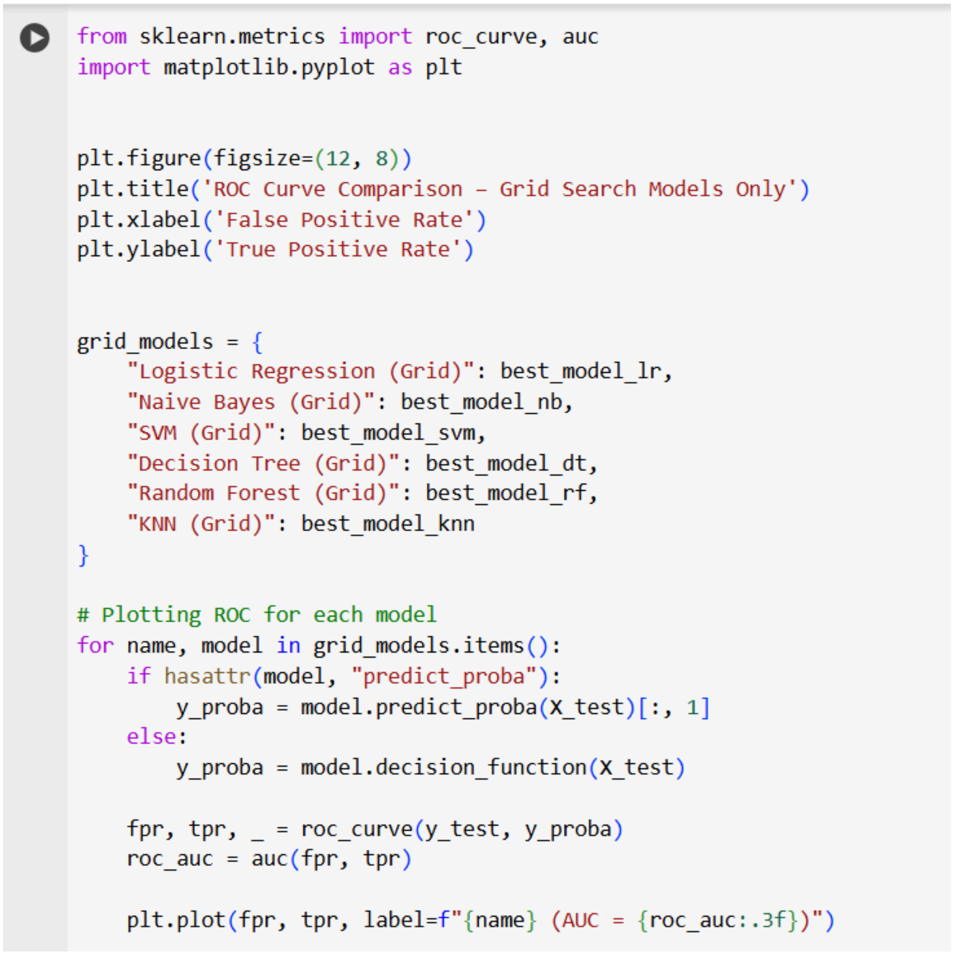
Summary of Grid Search Model Evaluation

A final comparison table was generated to summarize the performance of all six models trained with Grid Search optimization. Evaluation metrics included Recall, Precision, F1-Score (for the "Dead" class), PR-AUC, and ROC-AUC. Random Forest achieved the best balance between recall and precision, followed by Logistic Regression, which offered the highest recall. While models like Naive Bayes and KNN showed competitive performance, SVM and Decision Tree were comparatively weaker. The analysis confirmed Random Forest as the most effective model for predicting breast cancer survival outcomes, combining predictive strength with robustness against overfitting.

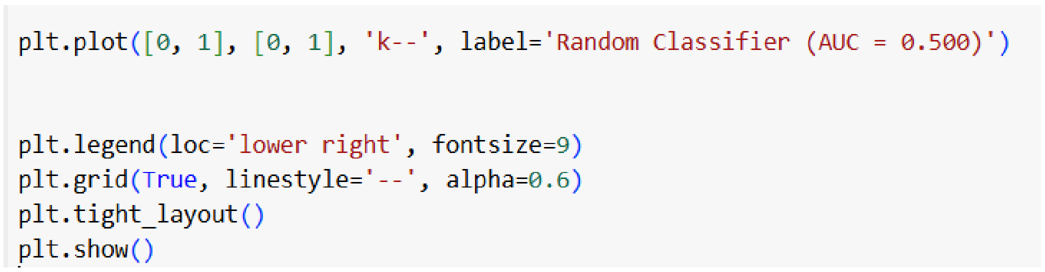
```
df_grid_report = pd.DataFrame(report_data)


print("📊 Grid Search Model Evaluation Table\n")
print(df_grid_report.round(3).to_string(index=False))
```

```
📊 Grid Search Model Evaluation Table

                     Model  Recall (for dead)  Precision (for dead)  F1-Score (for dead)  PR-AUC  ROC-AUC
Logistic Regression (Grid)              0.667                 0.446                0.534   0.598    0.842
        Naive Bayes (Grid)              0.561                 0.404                0.469   0.397    0.778
                SVM (Grid)              0.065                 0.258                0.104   0.296    0.724
      Decision Tree (Grid)              0.537                 0.420                0.471   0.479    0.710
      Random Forest (Grid)              0.569                 0.619                0.593   0.629    0.840
                KNN (Grid)              0.585                 0.414                0.485   0.516    0.757
```

Step 9: ROC Curve Comparison and Final Model Evaluation

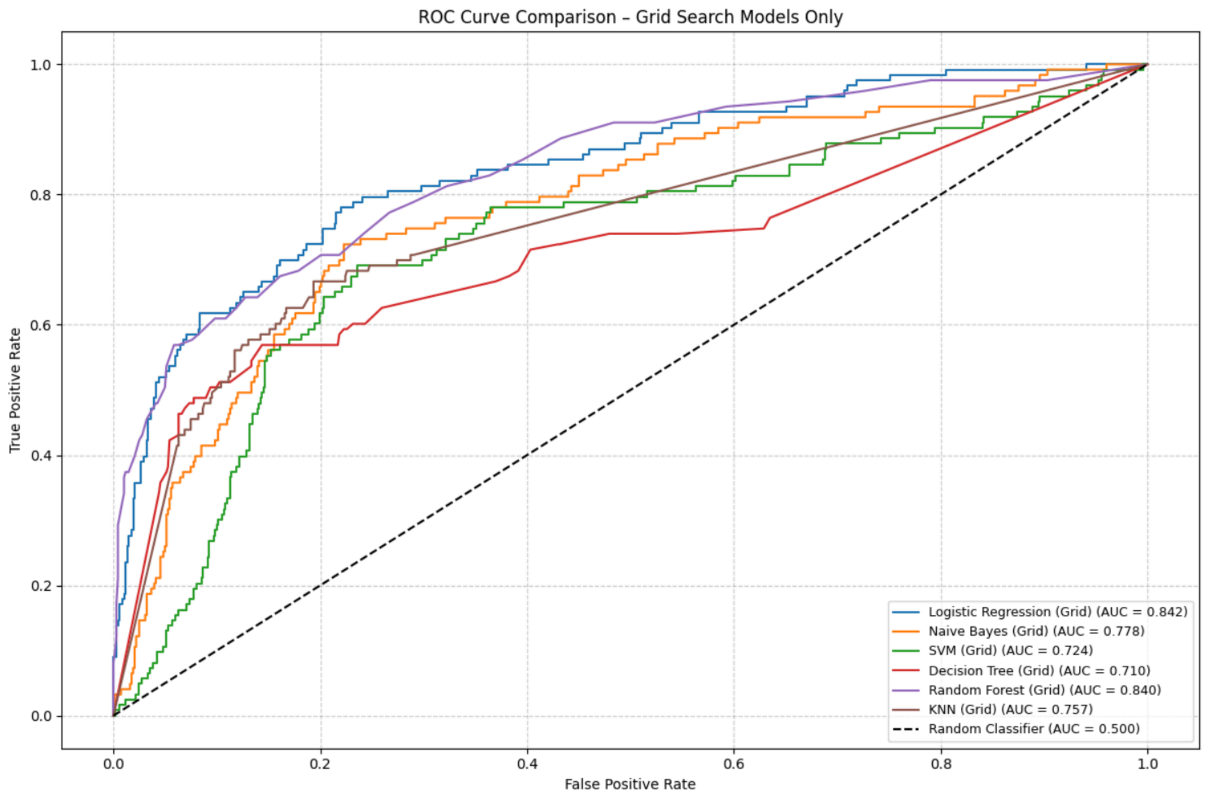


```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt


plt.figure(figsize=(12, 8))
plt.title('ROC Curve Comparison - Grid Search Models Only')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')


grid_models = {
    "Logistic Regression (Grid)": best_model_lr,
    "Naive Bayes (Grid)": best_model_nb,
    "SVM (Grid)": best_model_svm,
    "Decision Tree (Grid)": best_model_dt,
    "Random Forest (Grid)": best_model_rf,
    "KNN (Grid)": best_model_knn
}

# Plotting ROC for each model
for name, model in grid_models.items():
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)[:, 1]
    else:
        y_proba = model.decision_function(X_test)

    fpr, tpr, _ = roc_curve(y_test, y_proba)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, label=f"{name} (AUC = {roc_auc:.3f})")
```



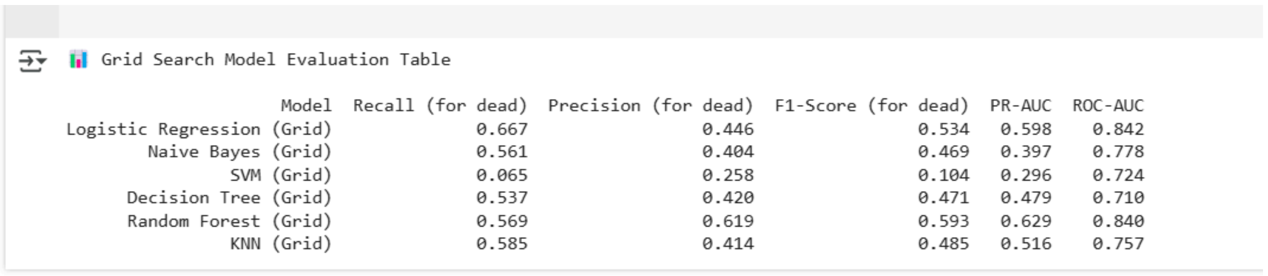
```python
plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier (AUC = 0.500)')


plt.legend(loc='lower right', fontsize=9)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

The ROC curve was used to visualize and compare the ability of each Grid Search–optimized model to distinguish between patients who survived and those who did not. All models showed varying degrees of separation, with **Logistic Regression** achieving the highest ROC-AUC score of **0.842**, followed closely by **Random Forest** with **0.840**. These results indicate that both models are highly effective in separating the two classes. While ROC-AUC provides a general view of classification quality, it does not capture model behavior on imbalanced data. Therefore, it was used in combination with other metrics like precision, recall, and F1 Score to make informed model selection decisions.



Results and Discussion

The performance of six machine learning models was evaluated using key classification metrics, including precision, recall, F1 Score, and PR-AUC, all measured on the test dataset after hyperparameter tuning via Grid Search.


```
📊 Grid Search Model Evaluation Table

                     Model  Recall (for dead)  Precision (for dead)  F1-Score (for dead)  PR-AUC  ROC-AUC
Logistic Regression (Grid)              0.667                 0.446                0.534   0.598    0.842
        Naive Bayes (Grid)              0.561                 0.404                0.469   0.397    0.778
                SVM (Grid)              0.065                 0.258                0.104   0.296    0.724
      Decision Tree (Grid)              0.537                 0.420                0.471   0.479    0.710
      Random Forest (Grid)              0.569                 0.619                0.593   0.629    0.840
                KNN (Grid)              0.585                 0.414                0.485   0.516    0.757
```

Logistic Regression (Grid Search) delivered the highest recall (0.667) among all models, indicating its strong ability to identify high-risk (dead) patients. It achieved a moderate precision (0.446) and a solid F1 Score (0.534), making it an excellent choice when sensitivity and interpretability are prioritized especially in clinical settings where failing to detect a death case can have severe consequences. **Random Forest (Grid Search) emerged as the best overall performer**, with the highest F1 Score (0.593), precision (0.619), and PR-AUC (0.629). This model demonstrated the most balanced trade-off between correctly identifying deaths and minimizing false positives, making it well-suited for real-world deployment where accuracy and reliability are crucial.

K-Nearest Neighbors (KNN – Grid) showed moderate results, with a recall of 0.585, precision of 0.414, and F1 Score of 0.485. Its performance was reasonable and remains a viable alternative in cases where simplicity or non-parametric modeling is desired. Naive Bayes (Grid Search) achieved a recall of 0.561, precision of 0.404, and F1 Score of 0.469. It effectively captured true positives but produced more false positives, making it less favorable for applications requiring precision. Decision Tree (Grid Search) had a recall of 0.537, precision of 0.420, and F1 Score of 0.471. Despite its ease of interpretation, the model underperformed due to limited generalization capabilities and potential overfitting. Support Vector Machine (SVM – Grid) was the weakest model across all metrics, with a very low recall of 0.065 and an F1 Score of 0.104, despite a reasonable ROC-AUC. Its poor performance on the minority class renders it unsuitable for predicting high-risk patients in this healthcare dataset.

Overall, these results support using random forest for its superior balance of precision and recall and logistic regression for its sensitivity and interpretability. Both models are well-aligned with clinical goals where early identification of at-risk patients is essential for improving outcomes.

Conclusion

In conclusion, this project demonstrates the potential of machine learning to support clinical decision-making in breast cancer prognosis. We identified models capable of predicting survival outcomes with meaningful accuracy by systematically preprocessing the data, addressing class imbalance, and fine-tuning model parameters. The comparative evaluation highlighted performance variability across algorithms, underscoring the importance of selecting models that align with clinical priorities. Beyond predictive power, factors such as interpretability, sensitivity to high-risk cases, and robustness to imbalance played a key role in model suitability. These findings pave the way for integrating such models into personalized care strategies and early intervention workflows.

Limitations and Challenges

Despite achieving promising results, the study faced several limitations and challenges. One major challenge was the class imbalance in the dataset, where the number of surviving patients significantly outweighed the deceased, affecting model sensitivity. Although SMOTE helped address this, synthetic oversampling may not fully reflect real-world complexity. Another limitation was the lack of temporal data, such as follow-up duration or treatment changes, which could provide deeper insights into survival prediction. Additionally, some features exhibited high correlation or limited variance, requiring careful feature selection. Models like SVM also struggled with low recall, reflecting the difficulty in capturing minority class patterns. Finally, the absence of external validation limits the generalizability of the models to other clinical populations.

Future work

Future improvements to this study include more clinical and genomic variables, like treatment history, genetic mutations, and lifestyle factors, to increase model robustness and predictive accuracy. By integrating longitudinal follow-up and temporal data, time-to-event analysis and more dynamic survival estimates may be possible. Investigating ensemble stacking strategies or deep learning models may improve performance, particularly for intricate feature interactions. The findings' generalizability would also be strengthened by external validation utilizing multi-center or real-world datasets. Finally, an intuitive clinical decision support tool based on the best-performing models may make it easier to implement for early risk assessment and individualized treatment schedules in hospital settings.

Data and Software Availability:

Dataset link: Breast Cancer Diagnosis Database :

https://www.kaggle.com/datasets/reihanenamdari/breast-cancer

Collab notebook link:

https://colab.research.google.com/drive/1-MUwRj9XIXj8ibeb1xa5isehowjk-Skx?usp=sharing

References:

1. Litjens, G., et al. (2017). A survey on deep learning in medical image analysis. Medical Image Analysis, 42, 60-88. https://pubmed.ncbi.nlm.nih.gov/28778026/
2. Ragab, D. A., et al. (2019). Breast cancer detection using deep convolutional neural networks and support vector machines. PeerJ, 7, e6201. https://pubmed.ncbi.nlm.nih.gov/30713814/
3. Mohapatra, P., & Mohanty, S. (2021). Performance evaluation of machine learning techniques for breast cancer prediction using data analytics. Journal of King Saud University - Computer and Information Sciences, 33(6), 619-628. https://www.sciencedirect.com/science/article/pii/S2352914822001526
4. Esteva, A., Kuprel, B., et al. (2017). Dermatologist-level classification of skin cancer with deep neural networks. Nature, 542(7639), 115-118. https://pubmed.ncbi.nlm.nih.gov/28117445/
5. Chaurasia, V., & Pal, S. (2017). A novel approach for breast cancer detection using data mining techniques. International Journal of Innovative Research in Computer and Communication Engineering, 5(2), 241-246. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2994932