Machine Learning Methods for Survival Analysis with Clinical and Transcriptomics Data of Breast Cancer

Le Minh Thao Doan, Claudio Angione, and Annalisa Occhipinti

Abstract Breast cancer is one of the most common cancers in women worldwide, which causes an enormous number of deaths annually. However, early diagnosis of breast cancer can improve survival outcomes enabling simpler and more cost-effective treatments. The recent increase in data availability provides unprecedented opportunities to apply data-driven and machine learning methods to identify early-detection prognostic factors capable of predicting the expected survival and potential sensitivity to treatment of patients, with the final aim of enhancing clinical outcomes.

Le Minh Thao Doan
School of Computing, Engineering and Digital Technologies, Teesside University, Middlesbrough, UK, e-mail: l.doan@tees.ac.uk

Claudio Angione
School of Computing, Engineering and Digital Technologies, Teesside University, Middlesbrough, UK; Centre for Digital Innovation, Teesside University, Middlesbrough, UK; Healthcare Innovation Centre, Teesside University, Middlesbrough, UK; National Horizons Centre, Teesside University, Darlington, UK; e-mail: c.angione@tees.ac.uk

Annalisa Occhipinti
School of Computing, Engineering and Digital Technologies, Teesside University, Middlesbrough, UK; Centre for Digital Innovation, Teesside University, Middlesbrough, UK; National Horizons Centre, Teesside University, Darlington, UK; e-mail: a.occhipinti@tees.ac.uk

This tutorial presents a protocol for applying machine learning models in survival analysis for both clinical and transcriptomic data. We show that integrating clinical and mRNA expression data is essential to explain the multiple biological processes driving cancer progression. Our results reveal that machine learning-based models such as random survival forests, gradient boosted survival model, and survival support vector machine can outperform the traditional statistical methods, i.e., Cox proportional hazard model. The highest C-index among the machine learning models was recorded when using survival support vector machine, with a value 0.688, whereas the C-index recorded using the Cox model was 0.677. Shapley Additive Explanation (SHAP) values were also applied to identify the feature importance of the models and their impact on the prediction outcomes.

Key words: Breast cancer, machine learning, survival analysis, data integration, interpretability.

1 Introduction

Breast cancer is a leading cause of cancer-related deaths worldwide [1]. According to the latest report published by Cancer Research UK, breast cancer occupies 15% of annual new cancer cases, and 7% of all cancer mortality in the UK [2]. With advancements in medical treatment and research, the overall survival rate has nearly doubled in the last 40 years, e.g., around 78% of patients survive more than ten years [2]. The survival rate after five years for early diagnosed patients varies between 90% and 99%, while this rate sharply drops to only 28% for late diagnosed patients [3]. Therefore, early detection and treatment are crucial for breast cancer patients as malignant cells tend to metastasise in later phases [4].

Clinical data has been often used to develop clinical prediction models and gain disease insights [5]. More recently, with the advancement of high-throughput sequencing technology, extensive omics data and methods for their integration have been produced, including genomics, transcriptomics, proteomics and metabolomics data [6, 7, 8]. The study of multi-omics data allows to investigate the relationships, roles, and actions of the various types of molecules constituting the cells of an organism and gain a comprehensive understanding of the biological system under examination. Information from omics data can be used to identify diagnostic and prognostic markers and support the development of personalised treatments [9]. Many studies have used omics data to develop accurate prognostic models for different cancer types [10, 11, 12], achieving more precise predictions than conventional clinical methods.

Following the breakthrough in exploring omics data, multiple assays from the same set of instances have been recently consolidated to generate multi-omics data. Their availability has reformed the biological and medical fields by making avenues for system-level integration tactics. Multi-omics integration has been used with great success to understand cancer and other disease progression mechanisms, to eventually obtain patient-specific clinical treatments and prevention strategies [13, 14, 15]. In fact, developing algorithms able to process multi-omics data could provide sharpness on biomolecules from different layers, pave the way towards large-scale cell optimisation, and facilitate the understanding of complex biological processes involved in cancer progression [16, 17]. Hence, compared to single-omics data, models using multi-omics data and mechanism-based approaches can provide a deeper understanding of cancer progression and related mechanisms, including discovering novel biomarkers, studying the interaction with viruses, and detecting cancer subtypes [18, 19, 20, 21].

Most survival-based molecular models have mainly used a single type of omic data [22]. However, recent investigations found that a proper combination of clinical and omics survival data could significantly improve clinical outcomes [5]. This integration usually outperforms the models that rely only on clinical or omics data [23]. Hence, it is necessary to investigate the effectiveness of using different types of data, such as clinical data, omics data, and their integration on the performance of the predictive models.

Recently, machine learning (ML) models have been successfully developed to process biomedical data, including characterisation of cell phenotype, detection of cancer, and prediction survival outcomes [24, 25, 26, 27]. Specifically, ML has been widely applied in clinical diagnosis and medical image analysis to develop computer-aided diagnosis systems [28]. The volume and variety of clinical and genomic data collected from patients are significantly increasing, revealing novel opportunities to apply ML and generate more insights into the molecular investigation of tumours and cancer prognosis. ML methods have facilitated the development of a more precise landscape about tumour heterogeneity and contributed to precision oncology. This allows specific patients to have an individual treatment plan based on a personalised diagnostic and prognostic risk profile. However, precise diagnosis and treatment of breast cancer are still representing one of the main challenges in healthcare [29]. Hence, developing accurate prognostic methods is necessary to significantly improve risk stratification after diagnosis and increase survival expectation. In order to achieve this, several patient-specific techniques have been proposed, either relying on clinical records, biological markers, or their combinations [30, 31]. However, there is still a need to identify the key bio-markers affecting cancer progression and survival outcomes in order to develop more accurate personalised treatments.

Survival analysis is a reliable and widely applied statistical technique among prognostic modelling methods, which attempt to evaluate the probability of events to occur within a specific time [32]. The prediction outcomes of this type of analysis, such as cancer death or recurrence, are fundamental to numerous clinical judgments in oncology and play an essential role for patients, doctors, and scientists [33]. Among the currently available survival analysis models, the Cox proportional hazards (CPH) regression model is the most widely applied approach to investigate the effect of the input features on the survival time of the patients [34, 35]. So far, numerous prognostic models have been proposed to apply the CPH regression model on clinical and transcriptomic data [36], and on multi-omics data [37]. However, ML has recently shown its successful applications in the medical and healthcare fields. Many ML models have been employed in cancer survival analysis because of their ability to handle high dimensional data, non-linear relationships, and interaction effects [38, 39]. ML-based approaches for survival analysis, such as random survival forests [40], gradient boosted survival model [41], survival support vector machine [42], Cox-nnet [43], and SALMON [44], have emphasised the feasibility of accurately predicting cancer outcomes using clinical and omics data.

Although survival analysis is widely applied in clinical studies, its prediction in practice still relies heavily on the subjective interpretation of the clinician, limiting reproducibility and accuracy [45]. Therefore, this tutorial aims to investigate breast cancer survival analysis by proposing a framework based on ML algorithms to perform survival analysis using clinical and transcriptomic data. CPH model and three ML-based models, namely random survival forests (RSF), gradient boosted survival model (GBS), and survival support vector machine (SSVM), are implemented and tested on the METABRIC dataset [46]. Our objectives are to classify the patients into risk groups (i.e., high-risk and low-risk), and unveil the prognostic predictors impacting the survival outcomes of patients. Consequently, identifying the patient risk groups could assist doctors in determining the course of treatment, promoting effective therapies, and supporting personalised clinical decision-making and recommendation.

Hence, the aim of our work is two-fold: (i) to present a protocol for applying ML algorithms in survival analysis. Specifically, elements of the study design, experiment process, and performance evaluation criteria are described and outlined to generalise and adapt our protocol to other public available clinical and transcriptomic data; and (ii) to uncover critical prognostic factors affecting the survival likelihood of breast cancer patients by employing the most recent statistical techniques for the interpretability of ML models (i.e., SHAP values) [47].

2 Backgrounds

This section presents the main methodologies and ML algorithms applied in our tutorial. The main differences between the three ML algorithms applied for survival analysis are also discussed.

2.1 Survival analysis

Survival analysis is a statistical procedure applied for analysing the expected duration of time until the occurrence of an event of interest (e.g., death or disease recurrence). One of the main challenges associated with survival analyses consists of dealing with censored data, a form of missing information that occurs because of the limited observation time, observation withdrawal, or lost to follow-up during the study period [48]. Censored data can be classified into two groups: left-censored and right-censored data. The former occurs when the event has already occurred before the beginning of the study, while the latter occurs when the survival time is only known to exceed a certain value, but the exact time is unknown. Right-censored data is the most common type of censored data [48]; therefore, this study will focus on the survival analysis for right-censored data.

For a given instance i, the survival information associated with i is comprised of two elements: a binary event indicator E_i, in which $E_i = 0$ for censored instance and $E_i = 1$ if the event (e.g., death) is observed, and a failure event time T_i, a non-negative random variable representing the duration between the beginning of the study and the occurrence of the event. The formula below reports the probability of observing the event by time t.

$$F(t) = Pr[T < t]. \tag{1}$$

The function $F(t)$ is defined as the cumulative distribution function.

Let the probability density function be denoted as $f(t)$. The survival function, $S(t)$, provides the probability that the event is observed after time t, and it is defined as

$$S(t) = Pr[T \geq t] = 1 - F(t) = \int_t^\infty f(x)dx. \tag{2}$$

The hazard function $h(t)$ represents the probability that the event will happen within the interval $[t, t + dt)$, given that it did not occur before time t. Thus, a lower hazard corresponds to a greater chance of survival. The hazard function $h(t)$ is defined as

$$h(t) = \lim_{dt \to 0} \frac{Pr[t \leq T < t + dt | T \geq t]}{dt}. \tag{3}$$

By using the definition of $S(t)$ in Eq. 2, the hazard function can also be written as

$$h(t) = \frac{f(t)}{S(t)} = -\frac{d}{dt} \log S(t). \tag{4}$$

Survival and hazard functions are two fundamental concepts in survival analysis, and they are connected by the expression below

$$S(t) = e^{-\int_0^t h(x)dx}. \tag{5}$$

Eq. 5 can be derived by integrating the first and last terms in Eq. 4 from 0 to t. The integral inside the parenthesis in Eq. 5 describes the sum of the risks of observing the event between time 0 and time t. This quantity is called cumulative hazard and it is defined as $H(t) = \int_0^t h(x)dx$.

2.2 Cox proportional hazards model

The Cox proportional hazards (CPH) model [49] has been the most commonly applied method in clinical studies to investigate the relationships between time-to-event or survival-time outcomes and explanatory variables. The CPH model is a regression approach used to calculate the hazard ratio (HR) and its confidence interval between patients belonging to different risk groups. Specifically, the HR can be interpreted as a relative risk. The CPH model is a semi-parametric model and it is denoted by the hazard function $h(t)$ representing the hazard at time t defined as

$$h(t) = h_0(t)e^{\beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k}, \tag{6}$$

where $h_0(t)$ is the baseline hazard function, and $\beta_1, \beta_2, .., \beta_k$ are the corresponding regression coefficients of covariates $x_1, x_2, ..., x_k$.

A value of e^{β_i} above 1, or β_i above zero, shows that the increase in value of the i^{th} covariate will lead to the rise in event hazard and, consequently, the reduction in survival time length. In other words, the covariate is positively correlated with the event likelihood or negatively associated with the survival time length. In contrast, a value of e^{β_i} below 1, or β_i below zero, shows that an increase in value of the i^{th} covariate will lead to a decreased probability of observing the event. If e^{β_i} is equal to one, that covariate does not affect the survival probability. Overall, observing e^{β_i} above one is a bad prognostic indicator in cancer studies, whereas observing e^{β_i} below one is a good indicator.

Let us consider two observations μ, ν with covariates $x_{\mu i}$ and $x_{\nu i}$, $i = 1, ..., k$ and hazard functions defined as

$$h_\mu(t) = h_0(t)e^{\beta_1 x_{\mu 1} + \beta_2 x_{\mu 2} + \ldots + \beta_k x_{\mu k}} \tag{7}$$

$$h_\nu(t) = h_0(t)e^{\beta_1 x_{\nu 1} + \beta_2 x_{\nu 2} + \ldots + \beta_k x_{\nu k}} \tag{8}$$

Using the definitions of $h_\mu(t)$ and $h_\nu(t)$ in Eq. 7 and Eq. 8, the HR for the two observations μ, ν is calculated as

$$HR = \frac{h_\mu(t)}{h_\nu(t)} = \frac{h_0(t)e^{\sum_{i=1}^{k} \beta_i x_{\mu i}}}{h_0(t)e^{\sum_{i=1}^{k} \beta_i x_{\nu i}}} = \frac{e^{\sum_{i=1}^{k} \beta_i x_{\mu i}}}{e^{\sum_{i=1}^{k} \beta_i x_{\nu i}}} = e^{\sum_{i=1}^{k} \beta_i (x_{\mu i} - x_{\nu i})} \tag{9}$$

Since the HR is not a function of time t, the hazard risk of the two groups must remain constant through the whole study, and their hazard curves should not cross. In fact, the CPH model is based on two assumptions: (1) the survival curves for two or more strata must have proportional hazard functions over time t, and (2) each covariate makes a linear contribution to the model.

2.3 Machine learning models

ML models employed for survival analysis have recently received increasing interest due to their promising applications in cancer research [39]. They are mainly applied to predict survival outcomes and the corresponding survival likelihood following statistical survival analysis approaches. However, rather than focusing on survival curves estimation, ML approaches mainly focus on predicting the time of event occurrence by merging the traditional statistical survival analysis techniques with the most recent statistical models. The advantages of using ML algorithms to perform survival analysis include the opportunity of providing more accurate solutions allowing the analysis of survival data while dealing with the statistical challenges associated with high dimensional data.

In this tutorial, patient-specific survival risk probabilities are predicted using the most recent ML algorithms, including random survival forests, gradient boosting model, and survival support vector machine, which have recently become popular due to their effectiveness in handling survival data [39].

2.3.1 Random survival forests

Random survival forests (RSF) is a random forest-based learning method used to analyse right-censored survival data [50]. The model uses an ensemble approach to generate predictions by integrating the estimations of multiple trees. This allows the model to gain more precise predictions than using a single tree. The algorithm employs tree-structured and bagging algorithms, typical of the random forest model [51], based on the three steps below.

1. A random bootstrap sample from the training set is selected to grow a tree;
2. The tree nodes are divided by a random attribute selection rather than using all the features available in the dataset;
3. The prediction of the random forest algorithm is determined by averaging the predictions of the individual tree.

Consequently, each tree in the forest is grown on an independent bootstrap sample extracted from the training data. This model is more independent and lowers the correlation between features, thus reducing the variance of the unbiased base learners occurring when using a single decision tree, and gaining better predictive performance. This technique aggregates different trees' decisions, and it often offers a better generalisation. The random forest algorithm has been demonstrated to be a widely adopted and effective ML technique for high-dimensional data and it is regarded as one of the most successful ensemble methods [52].

RSF extends the above approach by integrating censored information from survival data into the splitting rules applied for the growth of the forest. RSF is one of the most powerful and widely used learning algorithms for survival analysis. Each survival tree splitting employs the log-rank splitting rule to develop a set of survival trees, maximising the log-rank test statistic. Other splitting rules, such as log-rank score or conservation-of-events, can be used during the growing phase of the forest. However, log-rank splitting is the most popular technique and it is the focus of this tutorial algorithm.

According to Ishwaran et al. [50], the description of the RSF algorithm can be summarised as below.

1. The number of trees n in the forest and the number of predictors k for the splitting of each node are defined;

2. n bootstrap samples from data are drawn. Each sampling excludes out-of-bag data, which can be proven to be approximately equal to 37% of the full dataset [53].
3. A survival tree in each bootstrap sample is grown using the following approach:
   - k candidate predictor variables are randomly chosen;
   - For the possible splitting point of each k, the log-rank statistic is computed;
   - The node is split based on the log-rank splitting rule that maximises the survival difference between children nodes;
   - The tree continues to grow to full size under the constraint that the number of event observations (e.g., deaths) in each node is greater than a predefined minimum terminal node size.
4. A cumulative hazard function is computed for each tree. Then, the results are averaged to estimate the ensemble cumulative hazard function for all trees;
5. Harrell's concordance index [54] is calculated on the out-of-bag data and used to determine the predictive accuracy of the model.

2.3.2 Gradient boosted survival

Gradient boosted survival analysis (GBS) is a gradient boosting machine learning model applied to analyse censored data [55]. The predictive algorithm is based on an additive regression model of sequentially fitted weak learners (base-learners), while minimising the loss function. It is thus regarded as an ensemble learning method. It is a nonparametric approach and does not require any functional form assumption, providing researchers with more flexibility than other survival models. GBS also generates more robust returns than one single learner as it consolidates predictions from various estimations of weak learners.

In GBS, each successive tree is an enhancement over the previous one. In other words, the second tree improves over the first tree by learning from the residual of its prediction, while the third tree enhances over the first and second ones, and so on. The outcome is estimated by the weighted sum of all the predicted values given by the individual trees.

The gradient boosting algorithm can be summarised as below [56, 57].

1. The number of iterations M, the base learner model $h(\boldsymbol{x}, \boldsymbol{\theta})$, and the loss function $l(y, f)$ are defined, where $(\boldsymbol{x}, y)_{i=1}^{N}$ is the input data, $\boldsymbol{\theta}$ are the parameter estimates, and f is the unknown function that maps the input variables $\boldsymbol{x}$ to the target variables y;
2. An initial random guess $\widehat{f_0}$ of the unknown function f is defined;
3. For each iteration k, the following steps are performed:
    - The negative gradient of the loss function at iteration k is calculated;
    - The new base learner function $h(\boldsymbol{x}, \theta_k)$ is fitted;
    - The best gradient descent step-size ρ_k is estimated as follows

$$\rho_k = arg\, min_{\rho} \sum_{i=1}^{N} l[y_i, \widehat{f}_{k-1}(x_i) + \rho h(x_i, \theta_k)]; \qquad (10)$$

    - The estimated function $\widehat{f_k}$ is updated as $\widehat{f_k} = \widehat{f}_{k-1} + \rho_k h(\boldsymbol{x}, \theta_k)$;
4. The output of the final model is defined as $\widehat{f}(\boldsymbol{x}) = \sum_{k=1}^{M} \widehat{f_k}(\boldsymbol{x})$.

In this tutorial, we use regression trees as the base learner model, and CPH as the loss function in the GBS [58]. By doing this, the hazard, survival function, and log-hazard ratio are estimated by summing up the prediction of each regression tree.

2.3.3 Survival support vector machine

Support vector machine (SVM) is a very popular supervised learning method for regression and classification problems. SVM has also been applied to censored data for survival analysis [59]. The central idea of SVM is to classify data points by maximising the margin between groups in a high-dimensional space, and finding a separating hyperplane that minimises misclassification [60]. The hyperplane separates the classes and is as far from the closest observations as possible. Then, support vectors are defined as the data nearest to the maximum margin hyperplane.

Survival support vector machine (SSVM) follows the same approach as SVM but it employs an asymmetric penalty function to handle survival data. Specifically, linear SVM can be adapted to solve survival analysis by ranking, regression, and hybrid approaches. In a ranking approach, the learning model assigns a lower rank to instances with a shorter time of an event by examining all possible combinations of instances in the training data, while predicting the exact survival times in the regression problem. Because of its efficiency and optimal performance, this work focus on linear SSVM to handle survival analysis problems. We apply a more efficient SVM algorithm called FastSVM [61]. This model has lower computational training costs since it is based on truncated Newton optimisation and order statistic trees.

2.4 Feature selection

When working with transcriptomic data, the number of features often exceeds significantly the number of observations leading ML algorithms to overfit the data and report poor performance. For this reason, several features selections techniques have been proposed, with the aim of identifying and selecting an optimal subset of features. The most widely applied feature selection models include Pearson correlation, Spearman correlation [62], principal component analysis (PCA) [63], and genetic algorithm (GA) [64]. However, Schemper et al. [65] have shown that the Pearson and Spearman correlation models are unsuitable to work with censored data. Besides, dimensionality reduction methods, such as PCA, are difficult to interpret and are more suitable to use for the linear or approximately linear high dimensional data [66], while GA-based wrapper techniques have low computational efficiency [67]. Maximum relevance and minimum redundancy (mRMR) [68], a technique applied to select features based on their correlation with the response variables, has the advantage of fast computation and stronger robustness than the above feature selection techniques. Hence, mRMR is applied in this tutorial. According to Peng et al. [68], the model ranks the features according to both their relevance to the outcome and the low correlation between themselves. The steps performed by the mRMR algorithm are described below. Firstly, mRMR identifies the first feature based on the maximum relevance value.

Let I be the mutual information (MI) to measure both relevance and redundancy between features. The MI of two random features m and n is given by

$$I(m,n) = \iint p(m,n) \log \frac{p(m,n)}{p(m)p(n)}\, dm\, dn \tag{11}$$

where $p(m)$, $p(n)$, and $p(m,n)$ denote the probabilistic density function of m, n and their joint probabilistic density function, respectively.

Next, let X denotes the whole feature set, while S denotes the selected feature set containing s features, and c is the outcome class. For an individual feature x_i, $I(x_i, c)$ denotes its MI with the class c. The maximum relevance criterion, reflecting the largest dependence of x_i on the target class c, is computed by

$$\max D(S,c), \quad D(S,c) = \frac{1}{|S|} \sum_{x_i \in S} I(x_i; c). \tag{12}$$

The features selected by the maximum relevance criterion are likely to have large dependency among them. Hence, the minimum redundancy condition is added and calculated by

$$\min R(S), \quad R(S) = \frac{1}{|S|^2} \sum_{x_i, x_j \in S} |I(x_i, x_j)|, \tag{13}$$

where $I(x_i, x_j)$ is the MI of feature x_i and x_j.

The final mRMR feature set is chosen by simultaneously optimising Eq. 12 and Eq. 13. An incremental search approach is used to find the near-optimal features. Let us consider the S_{s-1} feature set with $s-1$ features already identified. The s^{th} feature is selected from the remainder feature set $\{X - S_{s-1}\}$ by optimising the following condition

$$\max_{x_j \in X - S_{s-1}} \left[I(x_j; c) - \frac{1}{s-1} \sum_{x_i \in S_{s-1}} I(x_j; x_i) \right]. \tag{14}$$

3 Methods

This section describes the dataset used in this tutorial and the experiments run to perform survival analysis. We conducted three experiments to investigate the performance of CPH and ML-based models on clinical data, transcriptomic data, and on the integration of the two data types. First, we report the dataset description, then the study design, initial setting, and details of three experiments are discussed.

3.1 Dataset

METABRIC dataset [46] was used to assess the predictive performance of the CPH model and the ML methods implemented in this study. The dataset has been downloaded from cBioPortal (www.cbioportal.org/datasets). The tumour information in the original METABRIC study was collected from five centres in the UK and Canada. The objective of the study was to analyse the effect of genomic and transcriptomic profiles on breast cancer survival to discover the optimal treatment approach of patients. The dataset contains clinical information for 2,509 primary breast cancer samples, and 2,509 molecular profiling, including 1,904 transcriptomic data with a maximum follow-up period of 355 months. Clinical data was obtained from cohort studies and trials, including the survival time in months and status (deceased or censored), while gene expression data was extracted from mRNAseq, which provides a snapshot of the transcript abundance of different gene transcripts of the cell. The detailed description of tissue specimens and staging can be found in the original METABRIC study of Curtis et al. [69]. To explore the power of CPH and ML models, we considered all the clinical and transcriptomic covariates available in the dataset.

3.2 Study design

We set up three experiments to evaluate the CPH and ML models for survival analysis, including (1) clinical data, (2) transcriptomic data, and (3) integrating clinical and transcriptomic data. Python programming language (version 3.8.8) and its libraries on the Anaconda environment (version 4.10.3) were used to conduct the experiments. Python 3 can be run on any popular operating system such as Windows, Mac, and Linux. However, the steps in this work are demonstrated on a Windows 10 Pro - 64-bit operating system.

We separately implemented all the experiments in Jupiter notebooks, an open-source web application that integrates code, visualisations, computational output, and other resources in one single file. However, the code can also be efficiently run online on Google Colab (https://colab.research.google.com) without any software installation.

The CPH and the current state-of-the-art ML algorithms presented in Sect. 2 were applied and evaluated in this study using scikit-survival packages [70] for Python.

The procedure of the experiments followed in this tutorial is described below and presented in Fig. 1.

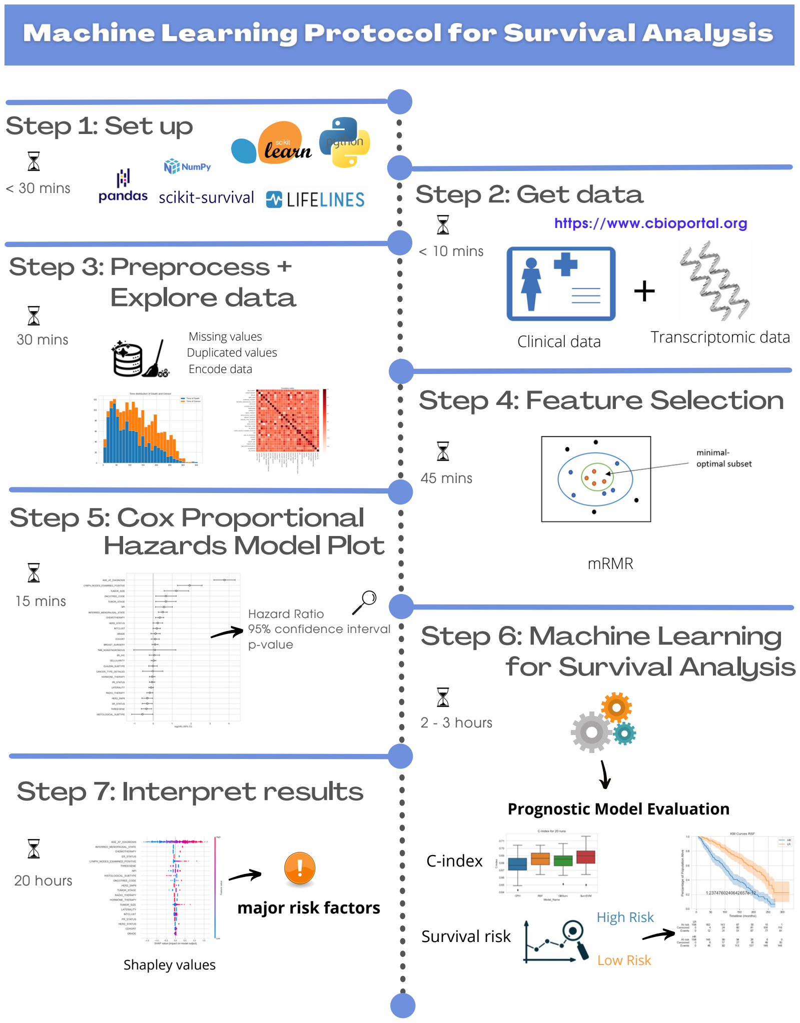


Fig. 1: Tutorial Workflow. After setting the environment (Step 1), clinical and transcriptomic data was retrieved from cBioPortal (Step 2). To start the experiment, we loaded the data, followed by data cleaning and data exploration steps (Step 3). Due to the high dimensional nature of transcriptomic data, a feature extraction step was required before running the machine learning models (Step 4). Next, the CPH model was run and the results were plotted to investigate the HR and p-value associated with each risk factor (Step 5). Then, we built, trained, and evaluated the ML models for survival analysis. Patients were then divided into high and low-risk groups based on the predicted risks scores, and the survival risk differences between groups were investigated (Step 6). Finally, the top critical prognostic markers were identified and interpreted using SHAP values (Step 7).

Step 1: The libraries and packages required for the analysis are first installed and imported;

Step 2: The METABRIC dataset is loaded;

Step 3: Preprocessing steps and data exploration techniques are performed to investigate the dataset;

Step 4: Feature selection is applied to select the optimal number of features from a large set of variables (this step is applied only to the transcriptomic data, where data reduction is necessary to improve the performance of models);

Step 5: The CPH model is run and the results are plotted and interpreted;

Step 6: ML algorithms are set up and run to generate the final predictive models;

Step 7: Results are interpreted using SHAP values, and models are compared.

The outputs of the survival ML models are patient-specific risk scores, which incorporate OS time and the corresponding event censorship indicator. A higher risk score indicates a greater likelihood of observing the event of interest (e.g., decease) early. Therefore, it is necessary to find an appropriate metric to evaluate the performance of models based on such predicted risk scores.

Harrell's concordance index (C-index) [54], a goodness of fit for survival models, is used to measure the concordance probability $P(\eta_j > \eta_i | T_i > T_j)$ for two instances i and j to rank association between their OS time points T_i, T_j and the models' prediction η_i, η_j. It assesses the possibility for a random observations pair that the patient with a higher risk score is the one who has a shorter survival time. Hence, it estimates how well a model predicts the ordering of decease times of patients. C-index values range from 0 to 1, where a value of 0.5 corresponds to a random model or no predictive discrimination. In contrast, C-index equal to 1 implies a precise association or perfect ranking of the observed and predicted survival times.

3.3 Initial setting

Before starting the analysis, the folders named *Data* and *Plot* are required to be set up in your local machine to store all data and figures for the experiments. Then, Python 3 [71] needs to be installed. The software is free and can be downloaded from www.python.org/downloads/. We recommend using the Anaconda environment for Python and its libraries to run the experiments presented in this project.

The data files used for this project and the complete codes notebooks are available at (https://github.com/Angione-Lab/survival_analysis_tutorial). The repository includes the clinical and transcriptomic data (i.e., data_clinical_patient.csv, data_clinical_sample.csv, data_mRNA_median_all_sample_Zscores.csv), which are required to run the following steps.

After creating a new notebook, a new cell/field to run the codes needs to be created. By clicking on the "Run Cell" button, the code will be executed cell-by-cell. Finally, libraries and packages of Python need to be installed as shown below.

```
!pip install dataprep  # data exploration
!pip install scikit-survival  # survival analysis
!pip install lifelines  # plotting survival analysis
!pip install git+https://github.com/smazzanti/mrmr  # feature
    selection
!pip install shap  # model interpretation
```

Other than the above packages, some primary data preprocessing and visualisation libraries such as Pandas, NumPy, Matplotlib, and Seaborn are expected to be installed if the code is run on a local machine. The syntax !pip install + libraries_names can be followed to install the preliminary packages.

```
!pip install pandas  # loading and preprocessing data
!pip install numpy  # loading and preprocessing data
!pip install matplotlib  # visualisation
```

```
!pip install seaborn  # visualisation
!pip install -U scikit-learn  # preparing ML algorithms
```

Once the required libraries are installed, they need to be imported at the beginning of the notebook to use the relevant functions.

```
# Packages to load and preprocess data
import numpy as np
import pandas as pd

# Packages to visualise and explore data
import seaborn as sns
sns.set_style("whitegrid")
import matplotlib.pyplot as plt
from dataprep.eda import plot, create_report, plot_missing,
    plot_correlation

# Feature selection
from mrmr import mrmr_classif

# Packages to prepare data for ML
from sklearn. preprocessing import OrdinalEncoder
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.model_selection import train_test_split
from sklearn. preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline

# Packages for survival analysis
from lifelines import CoxPHFitter
from lifelines.utils import k_fold_cross_validation
from lifelines.statistics import logrank_test
from lifelines import KaplanMeierFitter
from lifelines.plotting import add_at_risk_counts
```

```
# Packages for ML in survival analysis
from sksurv.linear_model import CoxPHSurvivalAnalysis
from sksurv.svm import FastSurvivalSVM
from sksurv.ensemble import RandomSurvivalForest
from sksurv.ensemble import GradientBoostingSurvivalAnalysis
from sksurv.metrics import concordance_index_censored

# Package to interpret data
import shap
```

3.4 Experiment 1: clinical data

Following the workflow presented in Fig. 1, Experiment 1 was conducted to perform survival analysis on the clinical data. The data was first loaded into a data frame for data cleaning and exploratory data analysis (EDA). Then, the CPH model and ML models were trained and evaluated to predict the survival risk of the patients. Finally, the results were interpreted to identify the critical clinical factors associated with low survival of the breast cancer patient.

The details of the experiment are presented in the following sections.

3.4.1 Load data

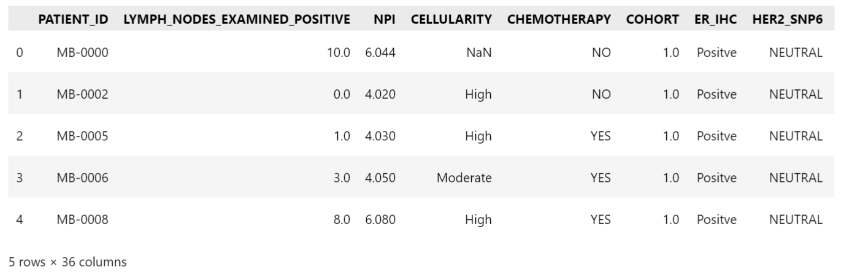
The patients information used in our analysis is stored in two files, one with clinical information and the other with the demographic characteristics of the patients. These two files need to be merged into a single data frame for easy processing.

```
# Load data
file1 = pd.read_csv('Data/data_clinical_patient.csv')
```

```
file2 = pd.read_csv('Data/data_clinical_sample.csv')

# Merge clinical data
data = pd.merge(file1,file2, how="inner", on=["PATIENT_ID"])
```

Once the data was loaded and merged, the first five rows of the new data frame and its information were extracted to get an overview of the data using the lines below. The outcome is reported in Fig. 2.

```
# Have a quick look at data
data.head()
```



	PATIENT_ID	LYMPH_NODES_EXAMINED_POSITIVE	NPI	CELLULARITY	CHEMOTHERAPY	COHORT	ER_IHC	HER2_SNP6
0	MB-0000	10.0	6.044	NaN	NO	1.0	Positve	NEUTRAL
1	MB-0002	0.0	4.020	High	NO	1.0	Positve	NEUTRAL
2	MB-0005	1.0	4.030	High	YES	1.0	Positve	NEUTRAL
3	MB-0006	3.0	4.050	Moderate	YES	1.0	Positve	NEUTRAL
4	MB-0008	8.0	6.080	High	YES	1.0	Positve	NEUTRAL

5 rows × 36 columns

Fig. 2: First five rows of the merged data frame. The data is presented in a table with the clinical features as columns and patients as rows. As the data frame comprised many columns, only the first eight columns are displayed in this figure.

Next, an overview of the data frame information can be generated by running the lines below. The output is displayed in Fig. 3.

```
# Data information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2509 entries, 0 to 2508
Data columns (total 36 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   PATIENT_ID                    2509 non-null   object
 1   LYMPH_NODES_EXAMINED_POSITIVE 2243 non-null   float64
 2   NPI                           2287 non-null   float64
 3   CELLULARITY                   1917 non-null   object
 4   CHEMOTHERAPY                  1980 non-null   object
 5   COHORT                        2498 non-null   float64
 6   ER_IHC                        2426 non-null   object
 7   HER2_SNP6                     1980 non-null   object
 8   HORMONE_THERAPY               1980 non-null   object
 9   INFERRED_MENOPAUSAL_STATE     1980 non-null   object
 10  SEX                           2509 non-null   object
 11  INTCLUST                      1980 non-null   object
 12  AGE_AT_DIAGNOSIS              2498 non-null   float64
 13  OS_MONTHS                     1981 non-null   float64
 14  OS_STATUS                     1981 non-null   object
 15  CLAUDIN_SUBTYPE               1980 non-null   object
 16  THREEGENE                     1764 non-null   object
 17  VITAL_STATUS                  1980 non-null   object
 18  LATERALITY                    1870 non-null   object
 19  RADIO_THERAPY                 1980 non-null   object
 20  HISTOLOGICAL_SUBTYPE          2374 non-null   object
 21  BREAST_SURGERY                1955 non-null   object
```

Fig. 3: Clinical data information. The figure reports an overview of the clinical data frame, including total entries, data types, the names of the columns, and the number of validated data points. There are 2,509 entries and 36 columns in the clinical data frame. The first 21 columns are shown in this figure, which include two types of data: (1) float or numeric and (2) object or non-numeric. Some columns contained missing values such as *LYMPH_NODES_EXAMINED_POSITIVE*, and *NPI*. This analysis provides a useful summary of the data before implementing any preprocessing steps.

The data contained some missing values; hence, it is essential to understand the data and preprocess it carefully before implementing any predictive models. In the next section, different techniques to explore and clean the data are performed.

3.4.2 Preprocess and explore data

In order to save computation time, duplicate observations and unused columns were dropped before conducting exploratory data analysis. This is one of the fundamental data cleaning steps to prepare the data for further analysis.

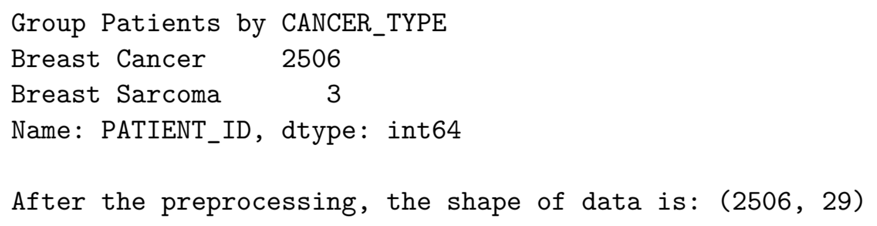
- *VITAL_STATUS* and *SAMPLE_ID* columns were dropped because they reported the same information as *OS_STATUS* and *PATIENT_ID*, respectively;
- *SEX* and *SAMPLE_TYPE* columns had only a single value; hence they were not providing any useful information for the predictive models and they were removed;
- *RSF_STATUS* and *RSF_MONTHS* were derived variables and were not used in our survival analysis;

```
# Drop unused columns: Based on data.info(), we will drop some
    unused cols and null cols

drop_list = ['VITAL_STATUS', 'SAMPLE_ID', 'SEX', 'SAMPLE_TYPE', '
    RSF_STATUS', 'RSF_MONTHS']
data = data.drop(drop_list, axis=1)
```

We also checked the number of patients for each cancer type since the target of our study is breast cancer. The dataset included some breast sarcoma instances, a sporadic form of breast cancer. However, since a normal breast cancer prognosis is our primary objective, the data was filtered by *CANCER_TYPE* to keep *normal breast cancer* only. The lines below show the implementation of the filtering steps. The output of these steps is reported in Fig. 4.

```
# We check the number of patients by cancer type
print('\nGroup Patients by',data.groupby('CANCER_TYPE')['
    PATIENT_ID'].count())
```

```
# There are only three patients with Breast Sarcoma
# So we will filter those patients with Breast Cancer type
data = data[data['CANCER_TYPE'] == 'Breast Cancer']

# Delete Cancer type columns as this column reports the same
    value for all the samples, and it does not bring any useful
    information for the following steps of the analysis.
data = data.drop(['CANCER_TYPE'], axis=1)
print('\nAfter the preprocessing, the shape of data is:', data.
    shape))
```



```
Group Patients by CANCER_TYPE
Breast Cancer     2506
Breast Sarcoma       3
Name: PATIENT_ID, dtype: int64

After the preprocessing, the shape of data is: (2506, 29)
```

Fig. 4: Output of preprocessing step. Only three breast sarcoma samples were present in the data; therefore, we dropped those three samples and left only one single value in the *CANCER_TYPE* column, i.e., normal breast cancer. As a result, since the *CANCER_TYPE* column reported the same value for all the samples, and it did not add any extra information about the samples, the column was removed and not included in the future steps of the analysis. Finally, after preprocessing, the final dataset consisted of 2506 samples and 29 features.

Before continuing the preprocessing phase (Step 2 in Fig. 1), data was explored to investigate data types, data distribution, and missing values. The library *dataprep* was used for exploratory data analysis (EDA). Other options to explore specific parts of the report, such as missing values and data distribution, were also used, as shown in the code below.

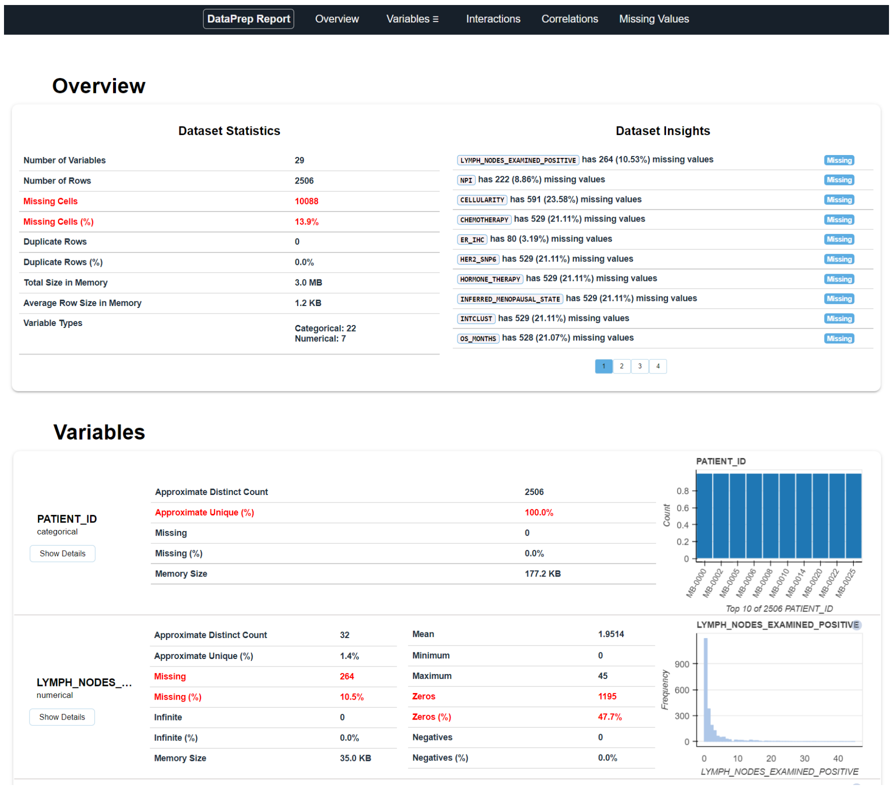
```
# Understand data
# Save to report as html file
create_report(data).save('Plot/EDA_clinical_report')
```

```

# Optional to explore parts of the report
plot_missing(data).save('Plot/missing_values.html')
plot(data).save('Plot/data.html')
```

The library generates an interactive EDA report that can be exported as an HTML file, as shown in Fig. 5, and opened in a web browser. This is a comprehensive report presenting all information about the features in the data frame. Besides the comprehensive report, the library allows to extract specific parts of the report. This can be achieved by selecting "Missing Values" in the menu at the top of the report page. For instance, the percentage of missing values for each variable is illustrated in Fig. 6. Once an overview of the data had been obtained, the next step was dealing with missing values. Our strategy was to remove the rows and columns with more than 50% missing values. Fig. 6 shows that there were no columns with more than 50% missing values.

We removed the rows with more than 50% blank values by running the lines of codes below. The output is reported in the first two rows of Fig. 7.

```
# Deal with missing values
# There is no columns more than 50% missing value
cols_mv_50 = data.columns[data.isnull().mean()>0.5]
print('Number of columns having more 50% missing data', len(
    cols_mv_50))

# Remove row with more than 50% missing
percent = 50
min_count =  int(((100-percent)/100)*data.shape[1] + 1)
data = data.dropna(axis=0, thresh=min_count)
print('After removing rows with more than 50% missing value:',
    data.shape)
```



Overview

Dataset Statistics

Number of Variables	29
Number of Rows	2506
Missing Cells	10088
Missing Cells (%)	13.9%
Duplicate Rows	0
Duplicate Rows (%)	0.0%
Total Size in Memory	3.0 MB
Average Row Size in Memory	1.2 KB
Variable Types	Categorical: 22 Numerical: 7

Dataset Insights

LYMPH_NODES_EXAMINED_POSITIVE has 264 (10.53%) missing values	Missing
NPI has 222 (8.86%) missing values	Missing
CELLULARITY has 591 (23.58%) missing values	Missing
CHEMOTHERAPY has 529 (21.11%) missing values	Missing
ER_IHC has 80 (3.19%) missing values	Missing
HER2_SNP6 has 529 (21.11%) missing values	Missing
HORMONE_THERAPY has 529 (21.11%) missing values	Missing
INFERRED_MENOPAUSAL_STATE has 529 (21.11%) missing values	Missing
INTCLUST has 529 (21.11%) missing values	Missing
OS_MONTHS has 528 (21.07%) missing values	Missing

1 2 3 4

Variables

PATIENT_ID
categorical
Show Details

Approximate Distinct Count	2506
Approximate Unique (%)	100.0%
Missing	0
Missing (%)	0.0%
Memory Size	177.2 KB

LYMPH_NODES_...
numerical
Show Details

Approximate Distinct Count	32	Mean	1.9514
Approximate Unique (%)	1.4%	Minimum	0
Missing	264	Maximum	45
Missing (%)	10.5%	Zeros	1195
Infinite	0	Zeros (%)	47.7%
Infinite (%)	0.0%	Negatives	0
Memory Size	35.0 KB	Negatives (%)	0.0%

Fig. 5: EDA report for clinical data. The report shows that the dataset consists of 29 features (22 categorical and 7 numerical features) and 2,506 rows. There are no duplicate rows, and 10,088 missing values account for 13.9% of the data. Besides, the report also reveals insights for each column (top-right panel), such as the number of missing values, skewness, unique number of values, and statistical summary. The distribution of each variable and information about missing values are also provided in the final report (bottom panel).

After removing the rows with more than 50% of missing values, we replaced any missing values with their mode (for categorical variables), and their average (for numeric variables). To achieve this, firstly, we identified which columns contained blanks and classified them into either categorical or continuous numeric types. Once this step was completed, we checked again the number of missing values to ensure there were no other missing values in the data. The output of the below codes is displayed in Fig. 7.

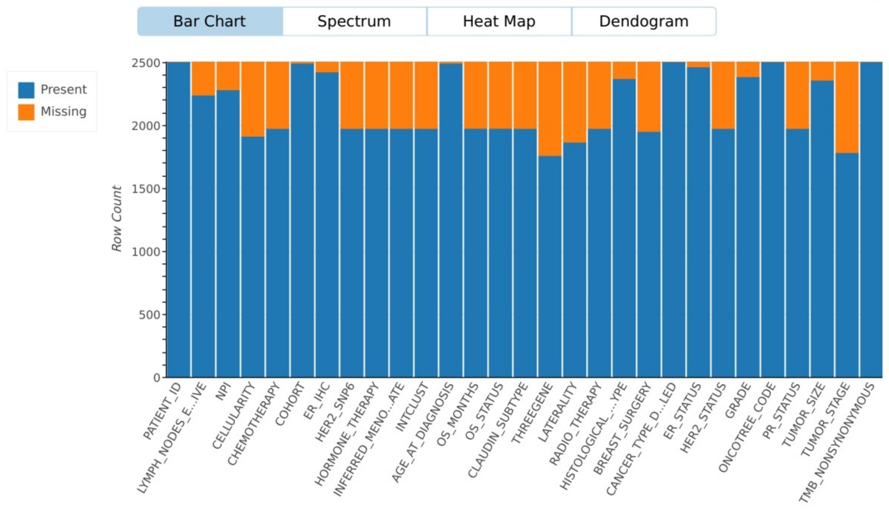


Fig. 6: Missing values chart. The plot shows the missing values information by column. In the stacked column chart, the orange section represents the number of blank rows, whereas the blue represents the non-blank ones. As shown in the chart, no columns have more than 50% missing values.

```
# Print columns name having blanks
cols_missvalue = data.columns[data.isnull().sum()>0]
print('List columns having missing data:', cols_missvalue)

cat_var = ['LYMPH_NODES_EXAMINED_POSITIVE', 'CELLULARITY', '
    ER_IHC', 'THREEGENE', 'LATERALITY', 'HISTOLOGICAL_SUBTYPE', '
    BREAST_SURGERY', 'GRADE', 'TUMOR_STAGE']
num_var = ['TUMOR_SIZE']

# Replace missing values with most frequent values
data[cat_var] = data[cat_var].fillna(data[cat_var].mode().iloc
    [0])

# Replace missing values with average values
data[num_var] = data[num_var].fillna(data[num_var].mean())
```

```
# Check missing values again
print('Missing value number:', data.isna().sum().sum())
```

```
Number of columns having more 50% missing data: 0
After removing rows with more than 50% missing values: (1977, 29)
List columns having missing data: Index(['LYMPH_NODES_EXAMINED_POSITIVE',
'CELLULARITY', 'ER_IHC', 'THREEGENE',
       'LATERALITY', 'HISTOLOGICAL_SUBTYPE', 'BREAST_SURGERY', 'GRADE',
       'TUMOR_SIZE', 'TUMOR_STAGE'],
      dtype='object')
After preprocessing, missing value number: 0
```

Fig. 7: Output of dealing with missing values steps. There are no columns that missed more than 50% of values. After removing the rows with more than 50% of missing values, the data's remaining rows are 1,977. Also, 10 columns, namely *LYMPH_NODES_EXAMINED_POSITIVE*, *CELLULARITY, ER_IHC, THREEGENE, LATERALITY, HISTOLOGICAL_SUBTYPE*, *GRADE, TUMOR_SIZE, TUMOR_STAGE*, *BREAST_SURGERY* contains missing values, which are replaced by their mode (if categorical) or their average (if numeric). Once the preprocessing steps are completed, no missing values are found in the dataset.

Before moving into the next step of the pipeline, distribution visuals for each variable were plotted using the below line of code. The output is reported in Fig. 8.

```
# Exploring clean data
plot(data.iloc[:,1:]).save('Plot\preprocessed_data.html')
```

In the following step (Step 2 in Fig. 1), some categorical variables were encoded to numeric to process and analyse data. Firstly, we prepared a list of features/columns to be encoded. Then the *OrdinalEncoder* function was used to transform the columns in the list to numeric.

```
# Encode categorical data

# Encode OS status to dummy
data['OS_STATUS']=np.where(data['OS_STATUS']=='1:DECEASED', 1, 0)
```

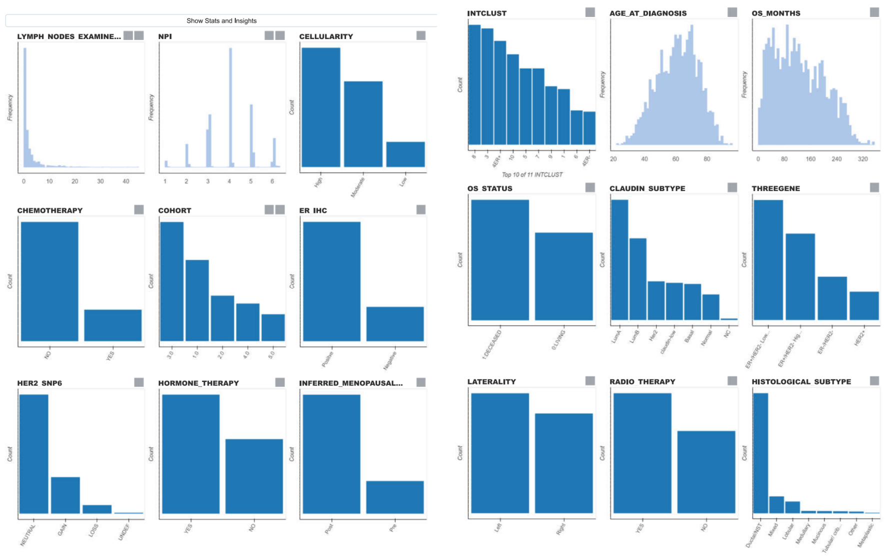


Fig. 8: Features distribution in clinical data. Excluding *PATIENT_ID*, the remaining 28 features, including *OS_MONTHS* and *OS_STATUS*, are plotted. The figure shows that *LYMPH_NODES_EXAMINED_POSITIVE* and *OS_MONTHS* are right-skewed, while the *CELLULARITY* values, i.e., amount of tumour cells, is mostly high, followed by moderate and low status.

```
# Encode other categorical variables
other_var = ['LYMPH_NODES_EXAMINED_POSITIVE', 'NPI','
    AGE_AT_DIAGNOSIS', 'COHORT', 'GRADE', 'TUMOR_SIZE', '
    TUMOR_STAGE', 'TMB_NONSYNONYMOUS', 'OS_MONTHS', 'OS_STATUS','
    PATIENT_ID']
df_encode = data.drop(other_var, axis=1)

# Some variables' values are not in order, so we have to specify
    the variables and their corresponding orders
modified_list =['CELLULARITY', 'HER2_SNP6', '
    INFERRED_MENOPAUSAL_STATE', 'INTCLUST', 'THREEGENE']
keep_list = df_encode.columns[~df_encode.columns.isin(
    modified_list)]
cel_cat = ['Low', 'Moderate', 'High']
```

```
her2_cat = ['UNDEF', 'LOSS', 'NEUTRAL', 'GAIN']
inf_cat = ['Pre', 'Post']
intclust_cat = ['1', '2', '3', '4ER+', '4ER-', '5', '6', '7', '8'
    , '9', '10']
three_gene_cat = ['ER-/HER2-', 'HER2+', 'ER+/HER2- Low Prolif', '
    ER+/HER2- High Prolif']

# Encode the predefined order variables
enc = OrdinalEncoder(categories=[cel_cat, her2_cat, inf_cat,
    intclust_cat, three_gene_cat]).fit(df_encode[modified_list])
encoder = enc.transform(df_encode[modified_list])
df_encode_new = pd.DataFrame(encoder, columns=modified_list)

# Encode the other variables
enc1 = OrdinalEncoder().fit(df_encode[keep_list])
encoder1 = enc1.transform(df_encode[keep_list])
df_encode_new1 = pd.DataFrame(encoder1, columns=keep_list)
```

Finally, the columns were concatenated to the other numeric columns to generate the final data frame.

```
# Merge encode data and original data
df =pd.concat([df_encode_new, df_encode_new1, data[other_var].
    reset_index(drop=True)], axis=1)
print(df.shape)
```

To check the mapping between the encoded categories and the original ones, the code below can be executed.

```
# To check the encoded categories
for i in range(len(col)):
    print(col[i], enc.categories_[i])
for i in range(len(keep_list)):
    print(keep_list[i], enc1.categories_[i])
```

```
CELLULARITY ['Low' 'Moderate' 'High']
HER2_SNP6 ['UNDEF' 'LOSS' 'NEUTRAL' 'GAIN']
INFERRED_MENOPAUSAL_STATE ['Pre' 'Post']
INTCLUST ['1' '2' '3' '4ER+' '4ER-' '5' '6' '7' '8' '9' '10']
THREEGENE ['ER-/HER2-' 'HER2+' 'ER+/HER2- Low Prolif' 'ER+/HER2- High Prolif']
CHEMOTHERAPY ['NO' 'YES']
ER_IHC ['Negative' 'Positve']
HORMONE_THERAPY ['NO' 'YES']
CLAUDIN_SUBTYPE ['Basal' 'Her2' 'LumA' 'LumB' 'NC' 'Normal' 'claudin-low']
LATERALITY ['Left' 'Right']
RADIO_THERAPY ['NO' 'YES']
HISTOLOGICAL_SUBTYPE ['Ductal/NST' 'Lobular' 'Medullary' 'Metaplastic' 'Mixed'
'Mucinous'
 'Other' 'Tubular/ cribriform']
BREAST_SURGERY ['BREAST CONSERVING' 'MASTECTOMY']
CANCER_TYPE_DETAILED ['Breast' 'Breast Invasive Ductal Carcinoma'
 'Breast Invasive Lobular Carcinoma'
 'Breast Invasive Mixed Mucinous Carcinoma'
 'Breast Mixed Ductal and Lobular Carcinoma' 'Invasive Breast Carcinoma'
 'Metaplastic Breast Cancer']
ER_STATUS ['Negative' 'Positive']
HER2_STATUS ['Negative' 'Positive']
ONCOTREE_CODE ['BRCA' 'BREAST' 'IDC' 'ILC' 'IMMC' 'MBC' 'MDLC']
PR_STATUS ['Negative' 'Positive']
```

Fig. 9: Output of the mapping between the encoded and the original categories. The figure shows the original values for each encoded categorical column. The original categories are presented in ascending order based on their corresponding encoded values.

Next, the clean data was saved in a CSV file, *clinical.csv*, in the *Data* folder to be used for the following analysis and to be integrated with transcriptomic data.

```
# Save preprocess data to csv to merge to gene data
df.to_csv('Data/clinical.csv', index=False)
```

Correlation analysis was performed to understand the relationship between features in the data. The heat map in Fig. 10 presents the Pearson correlation matrix where the varying intensity of colour represents the values of correlation. There were some highly correlated features observed in the data, such as *ER_STATUS* and *ER_IHC*, and *AGE_AT_DIAGNOSIS* and *INFERRED_MENOPAUSAL_STATE*.

```
# Drop Patient ID column as this is not relevant for the analysis
df = df.drop(['PATIENT_ID'], axis=1)
```

```
# Correlation analysis
colormap = plt.cm.Reds
plt.figure(figsize=(12,10))
sns.heatmap(df.corr(),linewidths=0.1,vmax=0.8,
            square=True, cmap = colormap, linecolor='white')
plt.title('Correlation matrix', fontsize=14)
plt.show()
```

Since the next steps of the pipeline are based on survival analysis, we calculated the percentage of censored data using the lines of code below. Overall, there was 42.2% of censored information.

```
num_censored = df.shape[0] - df["OS_STATUS"].sum()
print("%.1f%% of records are censored" % (num_censored/df.shape
    [0]*100))
```

Then, the follow-up time distribution of death and censored patients was plotted using the code below. The final chart is shown in Fig. 11. This step allows a further investigation into the time-to-event distribution for censored/non-censored patients.

```
# Time Distribution of Death and Censor
plt.figure(figsize=(9, 6))
val, bins, patches = plt.hist((df.query('OS_STATUS == 1')['
    OS_MONTHS'],df.query('OS_STATUS == 0')['OS_MONTHS']),
bins=30, stacked=True)
_ = plt.legend(patches, ["Time of Deaths", "Time of Censored"])
plt.title("Time Distribution of Censored and Death Patients")
```

3.4.3 Plot Cox proportional hazards model

In the next step of our pipeline (Step 5 in Fig. 1), the CPH model was fitted on the clinical data. The results were then visualised and reported to view the coefficients

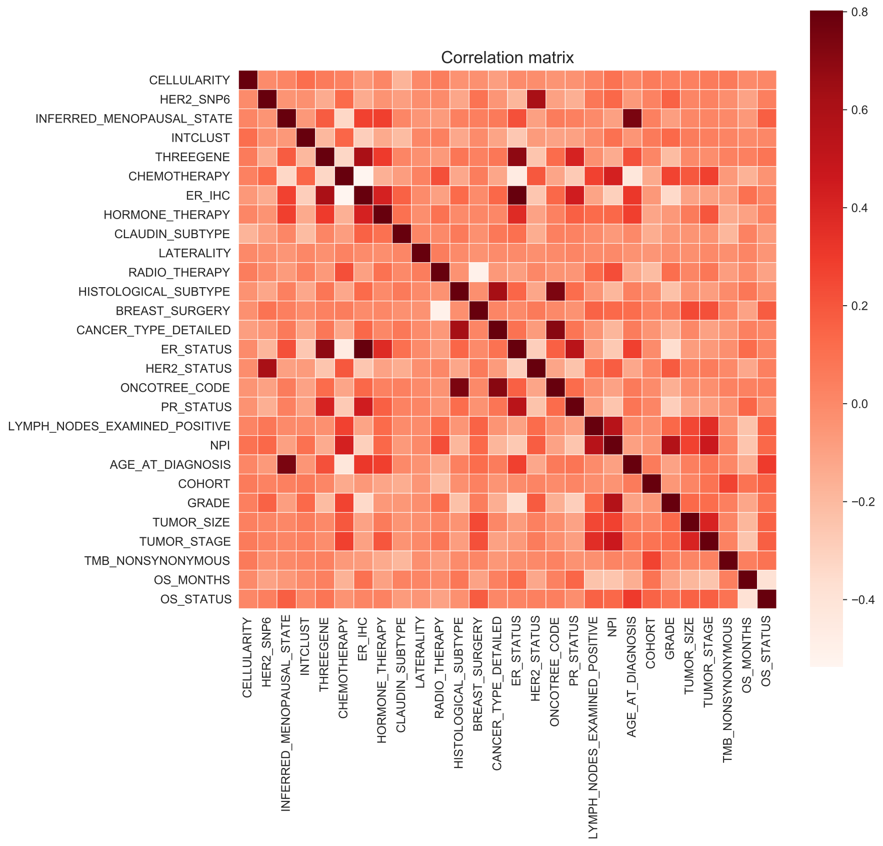


Fig. 10: Correlation matrix of clinical features. The correlation matrix depicts the linear correlation between all the pairs of attributes and ranges from -1 (perfect negative correlation) to +1 (perfect positive correlation), with the value of zero being no correlation between the features. Colour density represents the correlation's values, where the darker colour implies higher values and the lighter colour implies lower ones. The figure shows some high correlated features in the data, such as *ER_STATUS* and *ER_IHC*; *AGE_AT_DIAGNOSIS* and *INFERRED_MENOPAUSAL_STATE*.

and ranges of features. Before running the analysis, data needed to be normalised. Min-max normalisation, one of the most popular methods to normalise data, was applied. The method is based on the formula in Eq. 15 and the transformed data values range between 0 and 1.

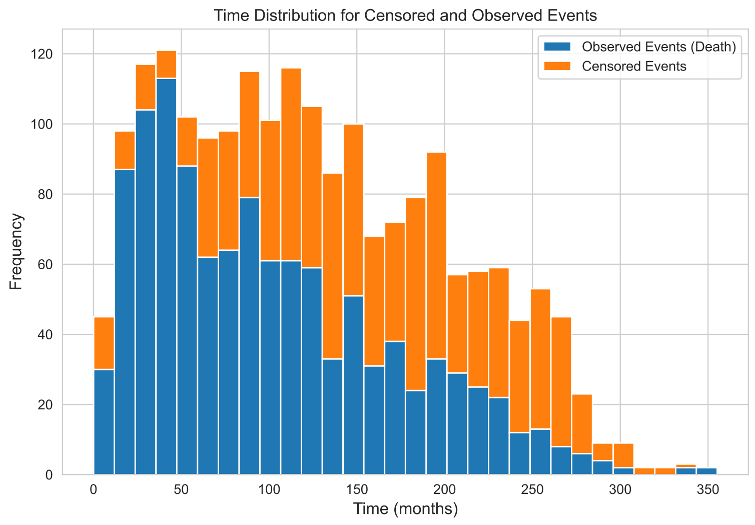


Fig. 11: Distribution of follow-up times of censored and uncensored (death) data. 42.2% of the total observations were censored. The distribution is right-skewed and is different between censored patients and those who experienced the event. The censored group has more patients with longer survival times.

$$x_{scaled} = \frac{x - min(x)}{max(x) - min(x)} \tag{15}$$

```
# Cox survival analysis
# Normalise data
ss = MinMaxScaler()
df_norm = df.drop(['OS_STATUS', 'OS_MONTHS'], axis = 1)
df_norm = pd.DataFrame(ss.fit_transform(df_norm), columns=df_norm
    .columns)
df_norm['OS_STATUS'] = df['OS_STATUS']
df_norm['OS_MONTHS'] = df['OS_MONTHS']
```

The next step was to use the entire dataset to fit the Cox regression model and the final results were plotted using the code below.

```
# Build model
# Cox proportional hazards model
cph = CoxPHFitter()
cph.fit(df_norm, duration_col='OS_MONTHS', event_col='OS_STATUS')

# Plot
plt.figure(figsize=(9, 12))
plt.title('Cox Proportional Hazards Model for Clinical data')
cph.plot()

# Report
cph.print_summary(columns=["coef","exp(coef)","exp(coef) lower
    95%","exp(coef) upper 95%", "z", "p"], decimals=3)
```

The hazard ratio of each feature and its statistical report are presented in Fig. 12 and Fig. 13, respectively. According to Fig. 12, *AGE_AT_DIAGNOSIS* was found as the most significant factor associated with the death events with the coefficient or hazard ratio value of 3.753. To be specific, elderly patients were 3.753 times as likely to die as younger ones. *LYMPH_NODES_EXAMINED_POSITIVE* was the second critical factor among the clinical data. Patients with positive lymph nodes tended to have a risk of death 1.888 times higher compared to those who did not have positive lymph nodes. As shown in Fig. 13, the overall C-index of this model is 0.685, which shows an acceptable predictive model.

The advantage of fitting the entire dataset to a regression model is that more data is fitted to the CPH model, which usually increases the accuracy of the model. Besides, the predictive capabilities of the CPH model fitted can be evaluated to see how well the algorithm performs on the entire data. However, the generalisation of the model cannot be assessed if the entire data is fitted to the model, and it is usually considered less trustworthy. Hence, cross-validation can be performed to reduce selection bias and overfitting. This approach also provides more insight into how

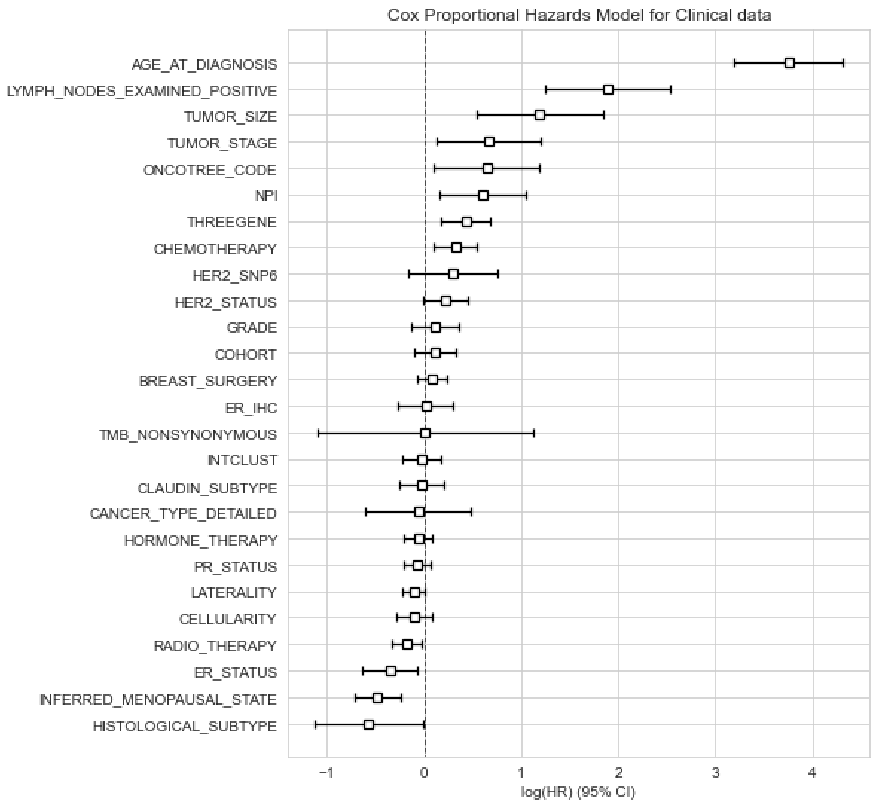
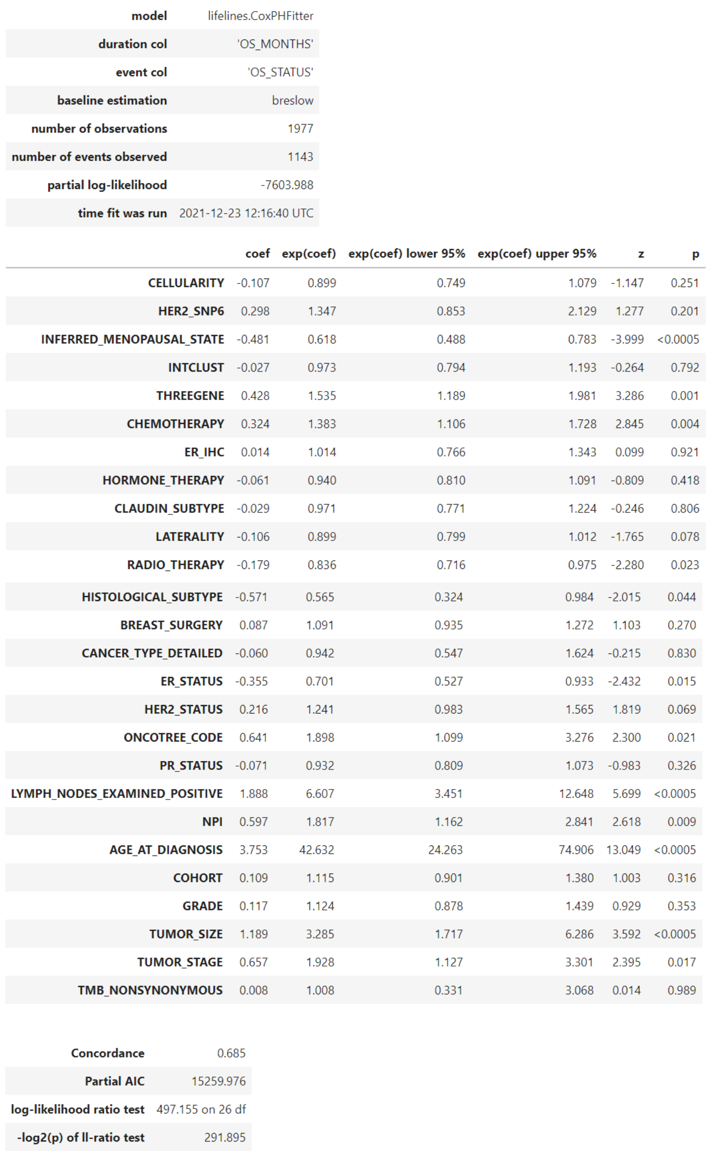


Fig. 12: Results of the Cox proportional hazards model for clinical data. The log hazard ratio is plotted for all the features, with a 95% confidence interval (CI). *AGE_AT_DIAGNOSIS*, *LYMPH_NODES_EXAMINED_POSITIVE*, and *TUMOR_SIZE* were found as the top three most significant factors associated with the death events with the log(HR) value of 3.753, 1.888, and 1.189, respectively. In other words, patients having higher values of these three predictors are more likely to have lower survival times. In contrast, the less than zero log(HR) value predictors, such as *HISTOLOGICAL_SUBTYPE* and *INFERRED_MENOPAUSAL_STATE*, were negatively associated with the death event. Patients with higher values of these factors tend to live longer compared to those who have low values.

well the model will perform on unseen data. Therefore, the next step of the analysis was to conduct a five-fold cross-validation to get an average C-index and generate more robust prediction scores. Specifically, a five-fold cross-validation approach splits the data into five folds, four of which are used as a training set to fit the model.



model	lifelines.CoxPHFitter
duration col	'OS_MONTHS'
event col	'OS_STATUS'
baseline estimation	breslow
number of observations	1977
number of events observed	1143
partial log-likelihood	-7603.988
time fit was run	2021-12-23 12:16:40 UTC

	coef	**exp(coef)**	**exp(coef) lower 95%**	**exp(coef) upper 95%**	**z**	**p**
CELLULARITY	-0.107	0.899	0.749	1.079	-1.147	0.251
HER2_SNP6	0.298	1.347	0.853	2.129	1.277	0.201
INFERRED_MENOPAUSAL_STATE	-0.481	0.618	0.488	0.783	-3.999	<0.0005
INTCLUST	-0.027	0.973	0.794	1.193	-0.264	0.792
THREEGENE	0.428	1.535	1.189	1.981	3.286	0.001
CHEMOTHERAPY	0.324	1.383	1.106	1.728	2.845	0.004
ER_IHC	0.014	1.014	0.766	1.343	0.099	0.921
HORMONE_THERAPY	-0.061	0.940	0.810	1.091	-0.809	0.418
CLAUDIN_SUBTYPE	-0.029	0.971	0.771	1.224	-0.246	0.806
LATERALITY	-0.106	0.899	0.799	1.012	-1.765	0.078
RADIO_THERAPY	-0.179	0.836	0.716	0.975	-2.280	0.023
HISTOLOGICAL_SUBTYPE	-0.571	0.565	0.324	0.984	-2.015	0.044
BREAST_SURGERY	0.087	1.091	0.935	1.272	1.103	0.270
CANCER_TYPE_DETAILED	-0.060	0.942	0.547	1.624	-0.215	0.830
ER_STATUS	-0.355	0.701	0.527	0.933	-2.432	0.015
HER2_STATUS	0.216	1.241	0.983	1.565	1.819	0.069
ONCOTREE_CODE	0.641	1.898	1.099	3.276	2.300	0.021
PR_STATUS	-0.071	0.932	0.809	1.073	-0.983	0.326
LYMPH_NODES_EXAMINED_POSITIVE	1.888	6.607	3.451	12.648	5.699	<0.0005
NPI	0.597	1.817	1.162	2.841	2.618	0.009
AGE_AT_DIAGNOSIS	3.753	42.632	24.263	74.906	13.049	<0.0005
COHORT	0.109	1.115	0.901	1.380	1.003	0.316
GRADE	0.117	1.124	0.878	1.439	0.929	0.353
TUMOR_SIZE	1.189	3.285	1.717	6.286	3.592	<0.0005
TUMOR_STAGE	0.657	1.928	1.127	3.301	2.395	0.017
TMB_NONSYNONYMOUS	0.008	1.008	0.331	3.068	0.014	0.989

Concordance	0.685
Partial AIC	15259.976
log-likelihood ratio test	497.155 on 26 df
-log2(p) of ll-ratio test	291.895

Fig. 13: Cox proportional hazards report for clinical data. The report indicates that *OS_MONTHS* was the duration variable, while *OS_STATUS* was the event variable used for survival analysis. The figure also reports the HR values (exp(coef)), with the corresponding 95% confident interval, and p-values of the clinical features. The accuracy prediction of the CPH model, i.e. the C-index, was 0.685, which indicates an acceptable model. Similar to the results presented in Fig. 12, *AGE_AT_DIAGNOSIS*, *LYMPH_NODES_EXAMINED_POSITIVE*, and *TUMOR_SIZE* were identified as the top three most significant factors associated with the death event with a p-value less than 0.0005, and coefficient/log(HR) value of 3.753, 1.888, and 1.189, respectively.

The fitted model is then evaluated on the left-out fold and a C-index is computed. The process is repeated for all the possible combinations of training and testing sets using the five folds. The final C-index is calculated as the average of the five C-index values generated during the five-fold cross-validation process. The code below can be run to perform cross-validation and generate the average C-index.

```
# Cross validation (optional)
scores = k_fold_cross_validation(cph, df_norm, 'OS_MONTHS',
    event_col='OS_STATUS', k=5, scoring_method="concordance_index
    ", seed=18)
print("Average score", round(np.mean(scores),3))
```

```
Average score 0.677
```

Fig. 14: Average C-index of five-fold cross-validation for Cox proportional hazards models. The figure shows the average C-index generated during the five-fold cross-validation process. The final C-index was 0.677, which was lower than the C-index of 0.685 reported in Fig. 13.

3.4.4 Set up and evaluate machine learning algorithms

In order to run the ML algorithms (Step 6 in Fig. 1), the following steps were applied:

1. Data was split into training and testing sets using a stratified split with a ratio of 80:20.
2. The machine learning models were trained using a five-fold cross-validation approach on the training set. Grid search was applied to autotune hyperparameters to get optimal solutions.
3. The trained models were applied to the testing set to generate patient-specific predictions.

4. Steps 1-3 were repeated 20 times on different splits of training and testing sets to obtain an average C-index. This process provides a more robust evaluation of the models, since it is not dependent on the training/testing split.
5. The prediction scores generated by the models was used to separate the patients in the testing set into higher risk and lower risk to investigate any significant difference in the survival rates of the two groups.

The five steps presented above are further discussed and illustrated below. Firstly, we set up a seed value to ensure the reproducibility of the results. Then, the data was arranged into a data frame X containing the prognostic attributes, and a y data frame containing the target variables (survival time and status). The new data frames were split into training and testing sets using a random and stratified approach with a ratio of 80:20.

```python
# Set up seed and the options for the cross-validation approach
SEED = 5
CV = KFold(n_splits=5, shuffle=True, random_state=0)

# Split data to prepare for ML
X = df.drop(['OS_MONTHS','OS_STATUS'], axis = 1)
df['OS_STATUS'] = np.where(df['OS_STATUS'] == 1, True, False)
y = df[['OS_STATUS','OS_MONTHS']].to_records(index=False)

# Split the data set into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, stratify=y['OS_STATUS'],random_state=SEED)
```

Once data was prepared, the ML models were applied by defining a function to train and evaluate the procedure. We used grid search with five-fold cross-validation to train and tune the hyperparameters for each estimator. Then, we applied the optimal algorithms to generate the final prediction on the testing set. The function returns the optimal model and C-index.

```python
# Build model
# Define a function for grid search to tune training model
# and predict the results
def grid_search(estimator, param, X_train, y_train, X_test,
    y_test, CV):

    # Define Grid Search
    gcv = GridSearchCV(estimator, param_grid=param, cv=CV,
                       n_jobs=-1).fit(X_train, y_train)

    # Find best model
    model = gcv.best_estimator_
    print(model)

    # Generate predictions
    prediction = model.predict(X_test)
    result = concordance_index_censored(y_test["OS_STATUS"],
    y_test["OS_MONTHS"], prediction)
    print('C-index for test set (Hold out):', result[0])

    return [model,  prediction]
```

Next, to avoid bias in our final evaluation, we ran each ML model 20 times. By defining the below function, the number of re-run times can be easily changed by modifying the value of n. The function below randomly generates n different seeds, one for each iteration. Training and testing set splitting is performed in each loop, the data is fitted to identify the optimal algorithm, and the final model is evaluated on unseen data. By doing so, we randomly created n different testing sets and evaluated each algorithm *n* times. Finally, the average results of the *n* runs were calculated and reported.

```
# Re-run experiment 20 times
def c_index(model, X, y, n=20):
    np.random.seed(1)
    seeds = np.random.permutation(1000)[:n]

    # Train and evaluate model with 20 times
    cindex_score = []
    predict_list = []

    for s in seeds:
        X_trn, X_test, y_trn, y_test = train_test_split(X, y,
    test_size=0.2, stratify=y['OS_STATUS'], random_state=s)
        model.fit(X_trn, y_trn)
        prediction = model.predict(X_test)
        predict_list.append(prediction)
        result = concordance_index_censored(y_test["OS_STATUS"],
    y_test["OS_MONTHS"], prediction)

        cindex_score.append(round(result[0],3))

    print('Average C-index for {} runs'.format(n), np.mean(
    cindex_score))

    return [cindex_score, predict_list]
```

After defining the two functions above for the ML process, we designed the experiment pipeline by specifying the algorithms and establishing their hyperparameters. Before applying the algorithms, all the data had to be normalised using Min-max normalisation. Different values of ridge regression penalty were tested to tune the CPH model (the values varied between 0.001 and 100, as shown in Table 1).

```
# Define the Pipeline and hyperparameter
# CoxPHSurvivalAnalysis
pipe_cox = Pipeline([('scaler', MinMaxScaler()),('model',
    CoxPHSurvivalAnalysis())])
param_cox ={'scaler': [MinMaxScaler()],
        "model__alpha": [0.001, 0.01, 0.1, 1, 10, 100]}
```

Then, we set up the hyperparameters for the three ML-based algorithms, namely RSF, GBS, and SSVM. In the RSF algorithm, the *n_estimators* and the *max_depth* hyperparameters can be set to specify the number of trees and the maximum depth of the tree in the forest, while the *min_samples_leaf* and the *min_samples_split* parameters can be set to specify the minimum number of samples required to be at a leaf node, and the minimum number of samples required to split an internal node, respectively. The deeper the tree grows in the forest, the more complex the model, which can easily lead to overfitting and increased computational complexity. In order to avoid these problems, a predefined *max_depth* parameter can be set; otherwise, the trees are grown until each leave contains less than *min_samples_split* samples. The *max_features* hyperparameter can be also defined to set the number of features to consider when looking for the best split. By default, the algorithm considers all the features and selects the one with the optimal metric to perform the split. If the *max_features* parameter is set equal to *sqrt*, the maximum number of features considered at each split is equal to the square root of the total number of features in the dataset. Reducing the number of features can save computational resources, increase the stability of the forest, reduce variance, and overfitting.

```
# Random Survival Forests
pipe_rsf = Pipeline([('scaler', MinMaxScaler()),('model',
    RandomSurvivalForest())])
param_rsf ={'scaler': [MinMaxScaler()],
        'model__random_state': [SEED],
        'model__max_features': ['sqrt'],
        'model__max_depth': [8],
        'model__min_samples_leaf': [50, 100],
        'model__min_samples_split': [100],
        'model__n_estimators':[500]}
```

In the GBS algorithm, the *n_estimators* parameter can be used to set the number of trees to generate, while the *learning_rate* parameter can be set to regulate the learning rate that shrinks the contribution of each tree. The GBS model is robust to overfitting, so a higher value of the *n_estimators* parameter often results in better performance. However, there is a trade-off between *n_estimators* and *learning_rate*. Thus, different combinations of the list of values of the above hyperparameters were tried in the tuning phase.

```
# Gradient Boost Survival
pipe_gbs = Pipeline([('scaler', MinMaxScaler()),('model',
    GradientBoostingSurvivalAnalysis())])
param_gbs ={'scaler': [MinMaxScaler()],
        'model__random_state': [SEED],
        'model__learning_rate': [0.01, 0.1, 1],
        'model__n_estimators':[200, 500, 800, 1000]}
```

The hyperparameters defined for the SSVM algorithm included the *optimizer*, which refers to the optimization techniques, such as the AVL tree (avltree), the red-black tree (rbtree), and the simple methods. The *max_iter* parameter can be set to define the maximum number of iterations to perform in the Newton optimization. These hyperparameters are necessary to design an effective and efficient SSVM model. A summary of the hyperparameters tuned for each model using grid search and their final values are presented in Table 1.

```
# Survival SVM
```

```
pipe_svm = Pipeline([('scaler', MinMaxScaler()),('model',
    FastSurvivalSVM())])
param_svm ={'scaler': [MinMaxScaler()],
        'model__random_state': [SEED],
        'model__max_iter': [500, 5000],
        'model__optimizer':['avltree', 'rbtree','simple']}
```

Models	Hyperparameters name	Hyperparameters set	Selected value
CPH	ridge regression parameter	[0.001, 0.01, 0.1, 1, 10, 100]	1
RSF	max_features	sqrt	sqrt
	max_depth	8	8
	min_samples_leaf	[50, 100]	50
	min_samples_split	100	100
	n_estimators	500	500
GBS	learning_rate	[0.01, 0.1, 1]	0.1
	n_estimators	[200, 500, 800, 1000]	200
SSVM	optimizer	[avltree, rbtree, simple]	avltree
	max_iter	[500, 5000]	500

Table 1: Hyperparameters of the models. Each method was parametrised and trained using a five-fold cross-validation approach. Grid search was used with different hyperparameters while maximising the C-index.

Once data preparation and models design were completed, an estimators dictionary, containing the pairs of names of the algorithms, and their corresponding pipelines, were generated to support looping the same procedure for each model.

```
# Estimator list:
estimator_list = {'Cox Regression':[pipe_cox, param_cox ],
                'Random Forest Survival':[pipe_rsf, param_rsf],
                'Gradient Boosting Survival':
                [pipe_gbs, param_gbs],
                'SVM Survival': [pipe_svm, param_svm]}
```

Since the training and testing phases for each algorithm follow the same approach, we put them into a list of estimators and iterate the same procedure over this list. The output of the procedure displays the optimal algorithm, the holdout-test results, and the average C-index for each model, as shown in Fig. 15. The results show that the average C-index over 20 runs of the three ML-based models outperformed CPH, a well-known statistical approach for survival analysis. SSVM had the highest average C-index with a value of 0.688, followed by RSF, GBS, and CPH with a C-index of 0.685, 0.683, and 0.678, respectively.

```
model_list = []
pred_list = []
c_index_list = []
pred_list_n = []

for model_name, index in estimator_list.items():
    print('\n',model_name)
    estimator = index[0]
    param = index[1]
    outcome = grid_search(estimator, param, X_train, y_train,
                          X_test, y_test, CV)
    model = outcome[0]
    pred_list.append(outcome[1])

    # Run model n times to check C-index
    score, pre = c_index(model, X, y, n=20)
    c_index_list.append(score)
    pred_list_n.append(pre)
```

```
 Cox Regression
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model', CoxPHSurvivalAnalysis(alpha=0.1))])
C-index for test set (Hold out): 0.660715999616086
Average C-index for 20 runs 0.6778500000000001

 Random Forest Survival
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model',
                 RandomSurvivalForest(max_depth=8, max_features='sqrt',
                                      min_samples_leaf=50,
                                      min_samples_split=100, n_estimators=500,
                                      random_state=5))])
C-index for test set (Hold out): 0.6678184086764565
Average C-index for 20 runs 0.6854499999999998

 Gradient Boosting Survival
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model',
                 GradientBoostingSurvivalAnalysis(n_estimators=200,
                                                  random_state=5))])
C-index for test set (Hold out): 0.667875995776946
Average C-index for 20 runs 0.6825

 SVM Survival
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model',
                 FastSurvivalSVM(max_iter=500, optimizer='avltree',
                                 random_state=5))])
C-index for test set (Hold out): 0.6747288607351953
Average C-index for 20 runs 0.68755
```

Fig. 15: CPH and ML models results for clinical data. The selected hyperparameters, initial test result, and the average C-index of each model are displayed in the outcome. Overall, the average performance over 20 runs of the three ML-based models outperformed the CPH model, a well-known statistical approach for survival analysis. SSVM had the highest average C-index with a value of 0.688, followed by RSF, GBS, and CPH.

Boxplots were then used to visualise and compare the distributions of C-index values for the 20 runs for each model (Fig. 16). On average, SSVM had the highest performance, followed by RSF, GBS, and CPH.

```
# Visualise results
name = ['CPH', 'RSF', 'GBS', 'SSVM']
cv_res = []

for i in range(0,4):
    for c in c_index_list[i]:
        cv_res.append([name[i],c])

c_plot = pd.DataFrame(cv_res, columns=['Model Name','C-index'])
ax = sns.boxplot(x="Model Name", y="C-index", data=c_plot)
plt.title('C-index for 20 runs')
```

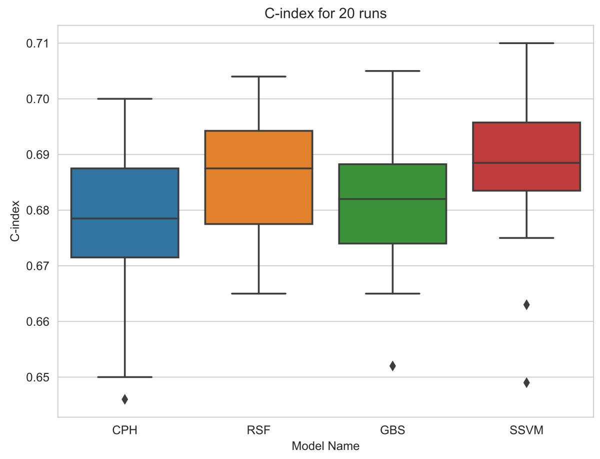
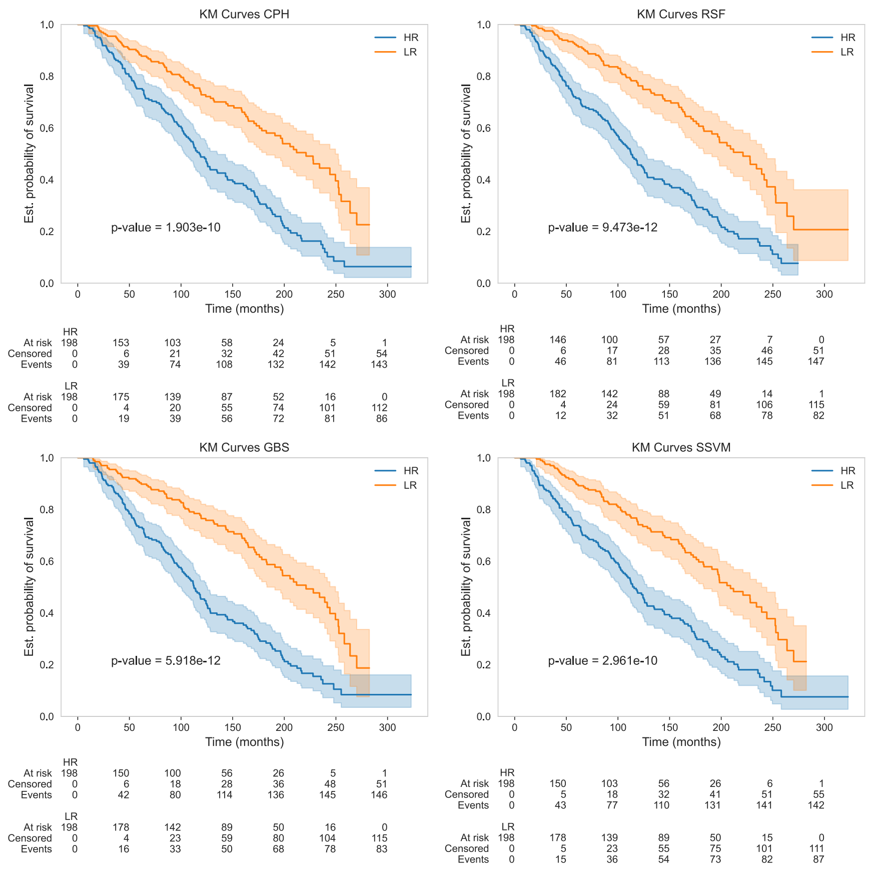


Fig. 16: C-index comparisons for Experiment 1. Boxplots of C-index results of clinical data using CPH, RSF, GBS, and SSVM. The experiments were replicated 20 times. In each experiment, the data was randomly divided into training and testing sets with a ratio of 80:20, while guaranteeing the same censoring percentage on each subset of data. SSVM was found to have the highest median C-index, followed by RSF, GBS, and CPH.

The patients in the testing set were then ranked by their predicted risk score and split into two equal-sized groups using the median risk score. High-risk groups included patients with prognostic risk scores greater than or equal to the median value, while low-risk groups included those with prognostic risk scores below the median value.

In the next step of the pipeline (Step 6 in Fig. 1), Kaplan–Meier plot and log-rank tests were conducted for all the models to statistically investigate the differences between the survival curves of the two groups. Fig. 17 reveals that the lower-risk patients, or those with lower predicted risk scores, were associated with better survival outcomes (i.e., higher survival probability). Besides, there were statistically significant differences in the survival distributions of high-risk and low-risk patients for all four models (p-values < 0.0001). Log-rank test was used to assess the statistical significance and compute the p-value. This analysis shows that the clinical factors can be used to split the patients into risk groups based on their predicted scores. GBS was the best model in prognostic diagnosis with a p-value of 5.918E-12.

```
fig, ax = plt.subplots(2,2,figsize=(12,12))
k  = 0
for pred in pred_list:
    df1 = X_test.reset_index(drop=True)
    risk =[]
    y_pred = pred
    med = np.median(y_pred)
    r = np.where(y_pred >= med, 1, 0)
    df1['Risk'] = r
    print(df1.shape)
    ix = df1['Risk'] == 1

    df_y = pd.DataFrame(y_test)
    df_y['OS_STATUS'] = np.where(df_y['OS_STATUS'] == True, 1, 0)
    df1['OS_STATUS']= df_y['OS_STATUS']
    df1['OS_MONTHS']= df_y['OS_MONTHS']
    T_hr, E_hr =df1.loc[ix]['OS_MONTHS'],df1.loc[ix]['OS_STATUS']
    T_lr, E_lr = df1.loc[~ix]['OS_MONTHS'], df1.loc[~ix]['
    OS_STATUS']

    # Set-up plots
    k+=1
    plt.subplot(2,2,k)

    # Fit survival curves
    kmf_hr = KaplanMeierFitter()
    ax = kmf_hr.fit(T_hr, E_hr, label='HR').
    plot_survival_function()
    kmf_lr = KaplanMeierFitter()
    ax = kmf_lr.fit(T_lr, E_lr, label='LR').
    plot_survival_function()
    add_at_risk_counts(kmf_hr, kmf_lr)

    # Format graph
    plt.ylim(0,1);
    ax.set_xlabel('Timeline (months)',fontsize='large')
    ax.set_ylabel('Percentage of Population Alive',fontsize='
    large')

    # Calculate p-value
    res = logrank_test(T_hr, T_lr, event_observed_A=E_hr,
    event_observed_B=E_lr, alpha=.95)
    print('\nModel', name[k-1])
    res.print_summary()

    # Locate the label at the 1st out of 9 tick marks
    xloc = max(np.max(T_hr),np.max(T_lr)) / 10
    ax.text(xloc,.2, res.p_value,fontsize=15)
    ax.set_title('KM Curves {}' .format(name[k-1]))
    plt.tight_layout()
```



KM Curves CPH

	0	50	100	150	200	250	300
HR							
At risk	198	153	103	58	24	5	1
Censored	0	6	21	32	42	51	54
Events	0	39	74	108	132	142	143
LR							
At risk	198	175	139	87	52	16	0
Censored	0	4	20	55	74	101	112
Events	0	19	39	56	72	81	86

KM Curves RSF

	0	50	100	150	200	250	300
HR							
At risk	198	146	100	57	27	7	0
Censored	0	6	17	28	35	46	51
Events	0	46	81	113	136	145	147
LR							
At risk	198	182	142	88	49	14	1
Censored	0	4	24	59	81	106	115
Events	0	12	32	51	68	78	82

KM Curves GBS

	0	50	100	150	200	250	300
HR							
At risk	198	150	100	56	26	5	1
Censored	0	6	18	28	36	48	51
Events	0	42	80	114	136	145	146
LR							
At risk	198	178	142	89	50	16	0
Censored	0	4	23	59	80	104	115
Events	0	16	33	50	68	78	83

KM Curves SSVM

	0	50	100	150	200	250	300
HR							
At risk	198	150	103	56	26	6	1
Censored	0	5	18	32	41	51	55
Events	0	43	77	110	131	141	142
LR							
At risk	198	178	139	89	50	15	0
Censored	0	5	23	55	75	101	111
Events	0	15	36	54	73	82	87

Fig. 17: Kaplan-Meier curves to compare the high-risk and low-risk breast cancer groups, stratified by the predicted survival risk scores generated by the four models. The low-risk group (n=198) included patients with predicted risk scores above the median value, while the high-risk group (n=198) comprised those less than the median value. Also, the p-value from the log-rank test was calculated to determine the statistical significance of the difference in survival functions between the two groups. The figure shows statistically significant differences in survival distributions between the two groups for all four models with a p-value lower than 0.0001.

3.4.5 Interpret model

Clinicians can rely on a predictive model when its outcome can be interpreted. This is especially crucial for the healthcare domain, where every decision relates to human life. Interpretability of ML can be defined as the extent to which an individual can understand the cause of the predicted outcome [72]. Shapely Additive Explanations values (SHAP) [47] can be applied to interpret the results of the ML models run in the previous section. SHAP values represent a unified approach to interpret predicted outcomes made by complex ML algorithms. This explainable approach has gained much attention from researchers and it has been increasingly applied in many fields, including medical and oncology applications [39, 73].

As shown in Step 7 in Fig. 1, SHAP values can be used to measure the importance of the features by calculating the impact of each estimator on the model prediction. In other words, it mainly focuses on explaining the importance or the weight of a specific feature on the model prediction. Each patient is represented by one data point with positive or negative values indicating the direction of the impact. The higher the SHAP value associated with the patient, the higher the mortality risk. For example, for the age feature, a 20-year-old patient might have a negative SHAP value of -1.5, meaning this young patient has a better prognosis and would live longer. In contrast, a 70-year-old patient might have a positive SHAP value of 1.0, indicating that this patient faces a higher mortality risk. Hence, age, in this case, is an important feature significantly influencing the survival rate of the patient. The code below can be run to perform the SHAP interpretability analysis. The run time for CPH, GBS, and SSVM is about 30 minutes per model, while RSF requires about 16 hours to generate the SHAP plot.

```
# Initialize JS For Plot
shap.initjs()
```

```
for i in range(0,4):
    print('\nModel', name[i])
    m = model_list[i][1]
    m.fit(X_train,y_train)
    explainer = shap.Explainer(m.predict, X_train, feature_names=
    X_train.columns)
    shaps = explainer(X_test)
    shap.summary_plot(shaps, X_test)
```

Fig. 18 shows the SHAP summary plot for clinical data, where a single patient is represented by one data point for each feature. The x-axis represents the effect of the features on the prediction of the algorithm for a specific observation in the testing set, while the y-axis reports the top prognostic predictors in descending order based on their importance ranking. *AGE_AT_DIAGNOSIS* was found consistently among the four models as the top significant factor impacting the survival risk. Specifically, higher age is associated with higher mortality risk. The SHAP values and features ranking are slightly different across the models. For example, according to CPH and SSVM, *INFERRED_MENOPAUSAL_STATE* was the second most important feature associated with survival risk, while *NPI* was the second most important feature in RSF and GBS. In contrast, *LATERALITY* and *ER_STATUS* had the lowest impact on the outcome of the model as they had convergent data points. Hence, we can get a holistic picture of the model prediction from the SHAP plots as they illustrate the importance of features and their corresponding impact on the outcome, while determining the value distribution of those features in the test set.

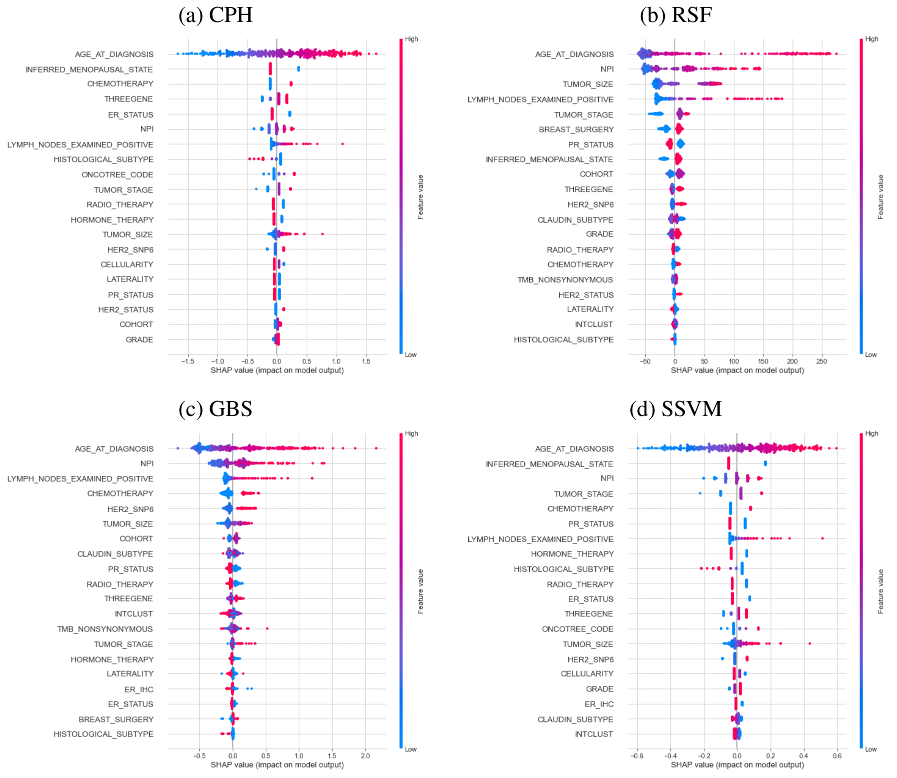


Fig. 18: SHAP summary plot for clinical data for (a) CPH, (b) RSF, (c) GBS, and (d) SSVM models. For each clinical feature, a single patient is represented by one point. The y-axis lists the top prognostic features and presents them in descending order based on their importance ranking provided by the mean of their absolute SHAP values. The x-axis reports the SHAP value indicating the impact of the feature on the prediction of the algorithm for a specific observation in the testing set. The colour represents the value of the feature. The higher the SHAP value the patient had, the higher the risk of death or the shorter survival time. *AGE_AT_DIAGNOSIS* was found consistently among the four models as the top significant factors impacting the survival risk.

3.5 Experiment 2: transcriptomic data

The second experiment presented in our work consists in conducting survival analysis on transcriptomic data following similar steps presented in Sect. 3.4. Since transcriptomic data is high dimensional data, features extraction is highly recommended to avoid overfitting and save computational time and resources. 50 features were extracted from the transcriptomic data and used to train and evaluate the models. To quickly reproduce the experiment and for a more straightforward presentation, we divided this experiment into two notebooks. The first one includes the preprocessing and feature selection steps, where the number of extracted features can be easily changed. The training and evaluation of the models are performed in the second notebook, where the data extracted from the first workbook are explored and used for the ML models.

3.5.1 Load data

Firstly, transcriptomic data was loaded. As it omitted *OS_MONTHS* and *OS_STATUS*, clinical data was also required to be loaded into the data frame to extract the relevant information about survival time and status.

```
# Load data
file1=pd.read_csv('Data/data_clinical_patient.csv')
file2=pd.read_csv('Data/data_mRNA_median_all_sample_Zscores.csv')
```

Then, the first five rows and data information are displayed, as shown in Fig. 19.

```
# Have a quick look on data
file2.info()
file2.head()
```

The transcriptomic data included 1,906 columns and 24,368 rows. The first two columns report the gene identifiers in different formats, namely *Hugo_Symbol* and *Entrez_Gene_Id*, while the rest of the columns report the data for the 1,904 patients.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24368 entries, 0 to 24367
Columns: 1906 entries, Hugo_Symbol to MB-4313
dtypes: float64(1905), object(1)
memory usage: 354.4+ MB
```

```
   Hugo_Symbol  Entrez_Gene_Id  MB-0362  MB-0346  MB-0386  MB-0574  MB-0503
0         RERE           473.0  -0.7082   1.2179   0.0168  -0.4248   0.4916
1       RNF165        494470.0  -0.4419   0.4140  -0.6843  -1.1139  -0.6875
2     CD049690             NaN   0.2236   0.2255   0.5691   0.3545   0.7865
3     BC033982             NaN  -2.1485   0.4763  -0.2446   0.2618  -0.2695
4         PHF7         51533.0  -0.3220  -1.0921   0.2830  -0.2864   0.0772
```

Fig. 19: Output of transcriptomic data information. The figure presents an overview of the transcriptomic data frame, including the total number of entries and the number of columns. There are 24,368 genomic entries and 1,906 patient columns in the data. As the dataset contains too many columns, the output shows only the first 7 columns, while the first five rows are extracted.

3.5.2 Preprocess data

Hugo Symbols (*Hugo_Symbol* column) were used in the transcriptomic data as a stable identifier for genes. Therefore, the *Entrez_Gene_Id* column was removed from the data frame.

```
# Drop unused column
file2 = file2.drop('Entrez_Gene_Id', axis=1)
```

We filtered the non-blank values in the *Hugo_Symbol* column. Next, missing and duplicate values were checked and removed (Step 3 in Fig. 1). A different approach for dealing with duplicates in trancriptomic data consists is replacing all the duplicates for each gene with their average value. For the sake of simplicity, in this tutorial, we decided to simply remove the duplicate values.

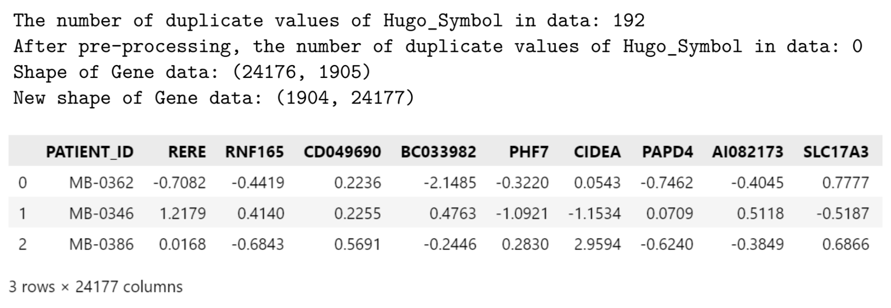
```
# Drop NA in GeneID
file2 = file2[file2['Hugo_Symbol'].notna()]

# Check null in GeneID columns
file2['Hugo_Symbol'].isnull().sum()
```

```
# Check duplicate values
print('The number of duplicate values of Hugo_Symbol in data:',
    file2['Hugo_Symbol'].duplicated().sum())

# Drop duplicate values for Gene ID
file2 = file2.drop_duplicates(subset=['Hugo_Symbol'])
print('After pre-processing, the number of duplicate values of
    Hugo_Symbol:', file2['Hugo_Symbol'].duplicated().sum())
print('Shape of Gene data:', file2.shape)
```

Fig. 20 shows that initially there were 192 repeated *Hugo_Symbol* values in our data. After the preprocessing, there were no duplicates and the final data frame had 24,176 rows (i.e., Hugo symbols) and 1,905 columns (i.e., 1,904 patients and one Hugo symbols ID). After eliminating duplicate values, the data frame was readily transposed to allow the matching of the Patient IDs in the transcriptomic data with those in the clinical data. As shown in Fig. 20, the new shape of the data frame was 1904 rows (i.e., patients) and 24,177 columns (i.e., 24,176 Hugo symbols and one Patient ID column). The first three rows in the new data frame were extracted to check the format of the transposed matrix.

```
# Tranpose Patient ID to rows in order to match two data
file2 = file2.set_index('Hugo_Symbol').T.rename_axis('PATIENT_ID'
    ).rename_axis(None, axis=1).reset_index()
print('New shape of Gene data:', file2.shape)
file2.head(3)
```

The new data frame was then merged with the *OS_MONTHS* and *OS_STATUS* columns from the clinical data based on the Patient ID information. The resulting data frame only comprised those matched patients between the transcriptomic and clinical tables.



```
The number of duplicate values of Hugo_Symbol in data: 192
After pre-processing, the number of duplicate values of Hugo_Symbol in data: 0
Shape of Gene data: (24176, 1905)
New shape of Gene data: (1904, 24177)
```

	PATIENT_ID	RERE	RNF165	CD049690	BC033982	PHF7	CIDEA	PAPD4	AI082173	SLC17A3
0	MB-0362	-0.7082	-0.4419	0.2236	-2.1485	-0.3220	0.0543	-0.7462	-0.4045	0.7777
1	MB-0346	1.2179	0.4140	0.2255	0.4763	-1.0921	-1.1534	0.0709	0.5118	-0.5187
2	MB-0386	0.0168	-0.6843	0.5691	-0.2446	0.2830	2.9594	-0.6240	-0.3849	0.6866

3 rows × 24177 columns

Fig. 20: Output of transcriptomic data preprocessing. The dataset includes 192 duplicate Hugo symbols. After removing duplicate values in the *Hugo_Symbol* column, there are 24,176 rows (i.e., Hugo symbols) and 1,905 columns (i.e., 1904 patients and one Hugo symbol column) in the dataset. We transposed the data frame before merging it with the clinical data to retrieve the *OS_MONTHS* and *OS_STATUS* columns, needed for performing survival analysis. After transposing the data, the new table contained 1,904 patient rows and 24,177 columns (24,176 Hugo symbols and one patient IDs column). The first three rows of the table are shown in the figure.

```
# Merge gene data with OS time and status
data = pd.merge(file1[['PATIENT_ID','OS_MONTHS','OS_STATUS']],
    file2, how="inner", on=["PATIENT_ID"])

# Have a quick look at data
data.head()
```

In the next step (Step 3 in Fig. 1), we checked if the new data frame contained any missing values.

```
# Check missing values
print('Total missing value in the dataset:', data.isnull().sum().
    sum())
cols_missvalue = data.columns[data.isnull().sum()>0]
print('List columns having missing data:', cols_missvalue)
```

According to the output presented in Fig. 21, there were 10 missing values in the entire data. We replaced those with their average values in the corresponding columns. Several techniques have been proposed to handle missing values in transcriptomic data, such as k-nearest neighbours imputation, Gaussian mixture clustering imputation, and weighted least square imputation [74, 75, 76]. For simplicity, we replaced the missing values with their average values in this tutorial.

```
# Deal with missing values
# Replace missing values with average values
data[cols_missvalue] = data[cols_missvalue].fillna(data[
    cols_missvalue].mean())

# Check missing values again
print('After preprocessing, the number of missing values:', data.
    isna().sum().sum())
```

```
Total missing values in the dataset: 10
List columns having missing data: Index(['TMPRSS7', 'SLC25A19', 'IDO1',
'CSNK2A1', 'BAMBI', 'MRPL24', 'AK127905',
       'FAM71A'],
      dtype='object')
After preprocessing, the number of missing values: 0
```

Fig. 21: Output of the processing of missing values. There are 10 missing values in the entire data. The Hugo ID columns with missing values are *TMPRSS7, SLC25A19, IDO1, CSNK2A1, BAMBI, MRPL24, AK127905*, and *FAM71A*. As the missing values in the data are numeric, we replace them with their average values in the corresponding columns.

3.5.3 Feature selection

mRMR was applied to extract the most relevant features from the *Hugo_Symbol* column to be used for the ML models (Step 4 in Fig. 1). Before employing mRMR, it is recommended to normalise the data to boost the performance of the algorithm and save computational time. Hence, after removing the survival and patient ID information, min-max normalisation was implemented to normalise the transcriptomic data.

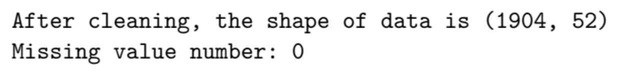
```
# Normalise data
ss = MinMaxScaler()
X_norm = data.drop(['OS_STATUS', 'OS_MONTHS','PATIENT_ID'], axis
    = 1)
X_norm = pd.DataFrame(ss.fit_transform(X_norm), columns=X_norm.
    columns)
```

For the mRMR algorithm, the number of selected features can be easily changed by modifying the value of K in the code below. In this experiment, we extracted 50 features (K=50) to demonstrate how to run the pipeline. The more features are removed, the longer is the time required by the mRMR algorithm. For 50 features, the model took around 30 to 45 minutes to run. After the features were extracted, the new data frame was saved to a new CSV file (*Gene_MRMR_50.csv*); this file will be required for the ML process in the second notebook.

```
# Features extraction
# Select features using mRMR
y_mrmr = data['OS_MONTHS']

features_selected = mrmr_classif(X_norm, y_mrmr, K = 50)
X_mrmr = data[features_selected]

# Save to csv file
df_mrmr = X_mrmr
df_mrmr['PATIENT_ID'] = data['PATIENT_ID']
df_mrmr.to_csv('Data/GeneID_MRMR_50.csv', index=False)
```

For easier processing, a new Jupyter notebook was created and the extracted data was loaded to carry on the next steps of the analysis. Then, the transcriptomic data of the 50 extracted features was merged with the clinical data by *Patient_ID*. After the merging, the *PATIENT_ID* column was not relevant for the ML analysis, and it was removed from the data frame. Before analysing data, the *OS_STATUS* column was encoded to numeric values. The final data frame included 1,904 rows (patients) and 52 columns (the survival time and status of patients, and 50 genes), as shown in Fig. 22.

```
# Load data
file1 = pd.read_csv('Data/data_clinical_patient.csv')
file2 = pd.read_csv('Data/GeneID_MRMR_50.csv')

# Merge gene data with OS time and status
data = pd.merge(file1[['PATIENT_ID','OS_MONTHS','OS_STATUS']],
    file2, how="inner", on=["PATIENT_ID"])

# Preprocess data
# Drop unused columns
drop_list = ['PATIENT_ID']
df = data.drop(drop_list, axis=1)
print('After the first preprocessing, the shape of data is', df.
    shape)
# Encode OS status to dummy
df['OS_STATUS'] = np.where(df['OS_STATUS'] == '1:DECEASED', 1, 0)
```



```
After cleaning, the shape of data is (1904, 52)
Missing value number: 0
```

Fig. 22: Output of preprocessing step. The figure shows that there were 1,904 rows and 52 columns in the final data frame. The columns in the final dataset comprised *OS_MONTHS*, *OS_STATUS*, and 50 transcriptomic extracted features columns. No missing values were found in the data.

Once the preprocessing steps (Step 3 in Fig. 1) on the data were completed, we conducted a correlation analysis and plotted the followed-up survival time distribution to investigate the data, as displayed in Fig. 23 and Fig. 24. No high correlated features were found in the selected transcriptomic data.

```
# Correlation analysis
colormap = plt.cm.Reds
plt.figure(figsize=(8,8))
sns.heatmap(df.corr(),linewidths=0.1,vmax=0.8,
            square=True, cmap = colormap, linecolor='white')
plt.title('Correlation matrix', fontsize=14)
plt.show()
```

```
# Time Distribution of Death and Censored
num_censored = df.shape[0] - df["OS_STATUS"].sum()
print("%.1f%% of records are censored" % (num_censored/df.shape
    [0]*100))

plt.figure(figsize=(10, 6))
val, bins, patches =
plt.hist((df.query('OS_STATUS == 1')['OS_MONTHS'],
        df.query('OS_STATUS == 0')['OS_MONTHS']),
        bins=30, stacked=True)
_ = plt.legend(patches, ["Time of Death", "Time of Censored"])
plt.title("Time Distribution of Censored and Death Patients")
```

The rest of Experiment 2, including plotting the CPH model, preparing and evaluating the ML algorithms, and interpreting the results, was set up as previously done in Experiment 1 in Sect. 3.4.3, 3.4.4, and 3.4.5. Since the code to run the analysis in the sections below is the same as the one shown in Sect. 3.4.3, 3.4.4, and 3.4.5, we do not repeat it below. However, the complete notebook can be accessed at https://github.com/Angione-Lab/survival_analysis_tutorial.

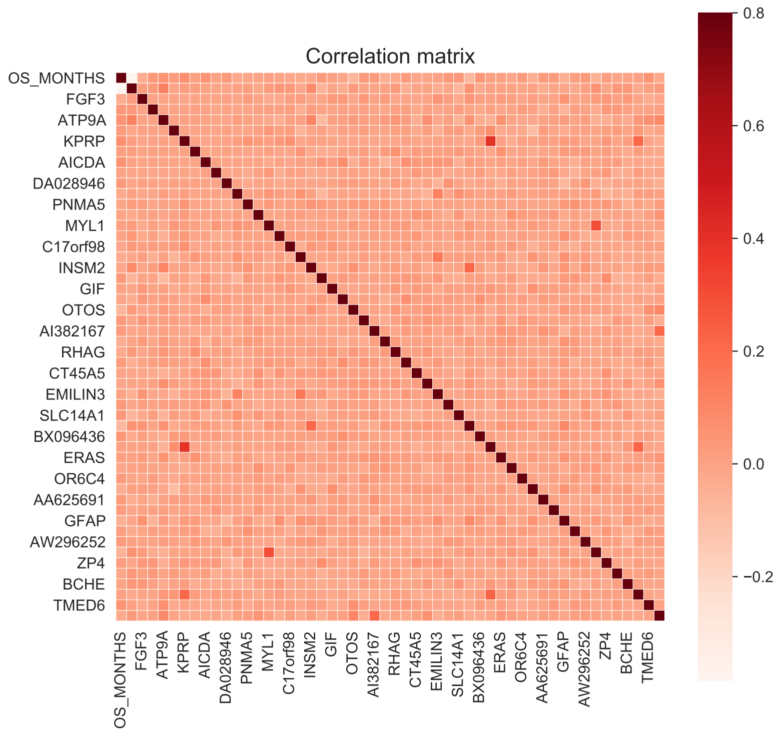


Fig. 23: Correlation matrix of 50 gene expression features. The correlation matrix depicts the linear correlation between all the pairs of attributes and ranges from -1 (perfect negative correlation) to +1 (perfect positive correlation), with the value of zero representing no correlation between the features. Colour density represents the values of the correlation, where a darker colour implies higher values and a lighter colour implies the lower ones. There were no highly correlated features observed in the data.

3.5.4 Plot Cox proportional hazards model

Before fitting the CPH model (Step 5 in Fig. 1), the data was normalised applying the Min-max method (as shown in Sect. 3.4.3). Then, the log(HR) values were plotted as shown in Fig. 25, and the statistical report, including HR with a 95%

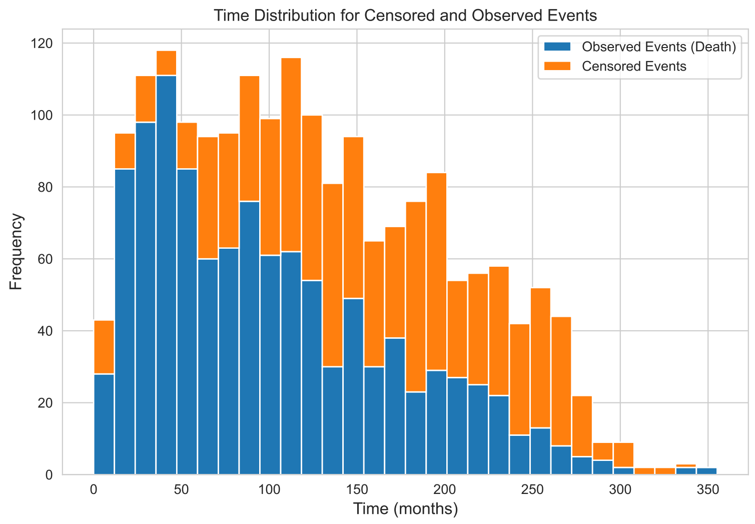


Fig. 24: Distribution of follow-up times of censored and uncensored (death) data associated with the transcriptomic data selected by mRMR. The data contained 42.1% censored observations. The distribution is right-skewed and it is different between censored patients and those who experienced the event. The censored group has more patients with longer survival times.

confidence interval and log-rank p-values, was generated. Genes *LCN15*, *OTOS*, and *INSM2* were identified as the top three most significant factors associated with a high probability of experiencing the event of interest (i.e., death), while genes *MATN1* and *KPRP* were negatively associated with the death event (as shown by their negative log(HR) values). As shown in Fig. 26, the overall C-index of this model is 0.574, which shows an acceptable predictive model.

3.5.5 Set up and evaluate machine learning algorithms

After visualising the results of the CPH model, we performed the same steps as Sect. 3.4.4 to build and evaluate the ML models (Step 6 in Fig. 1). The following steps were applied:

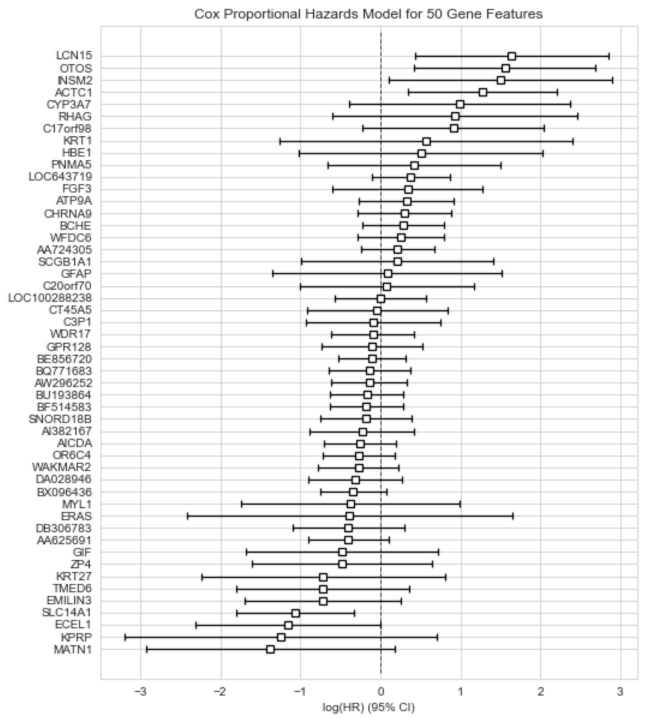
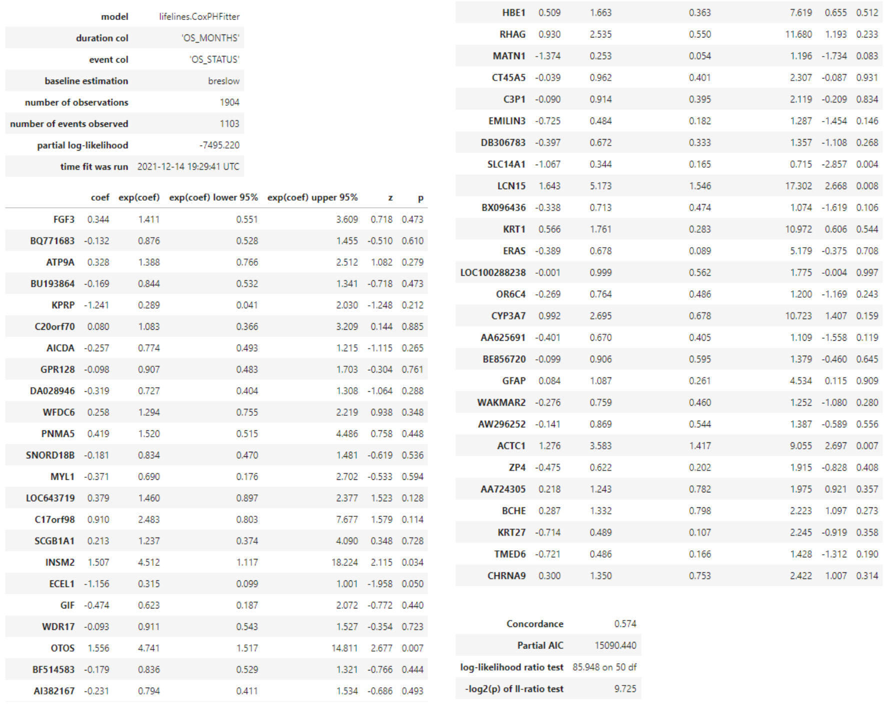


Fig. 25: Results of the CPH model applied to transcriptomic data. The genes *LCN15*, *OTOS*, and *INSM2* were found as the top three most significant factors associated with low survival with HR values of 1.643, 1.556, and 1.507, respectively. Hence, patients having higher values of these three predictors are more likely to have a shorter survival time. In contrast, the less than zero log(HR) value predictors (i.e., HR less than one), such as *MATN1* and *KPRP*, were negatively associated with the death event. Patients with higher values of these genes tend to live longer compared to those who have lower expression values of the same genes.

1. Data was split into training (80%) and testing sets (20%).
2. Data was normalised using the Min-max normalisation. Normalised data was used to fit and train the four predictive algorithms.
3. Five-fold cross-validation with grid search was used for tuning the hyperparameters and selecting the optimal hyperparameters.
4. The models were evaluated on the testing set and the full process (Steps 1- 3) was repeated 20 times to obtain the average C-index.



model	lifelines.CoxPHFitter
duration col	'OS_MONTHS'
event col	'OS_STATUS'
baseline estimation	breslow
number of observations	1904
number of events observed	1103
partial log-likelihood	-7495.220
time fit was run	2021-12-14 19:29:41 UTC

	coef	exp(coef)	exp(coef) lower 95%	exp(coef) upper 95%	z	p
FGF3	0.344	1.411	0.551	3.609	0.718	0.473
BQ771683	-0.132	0.876	0.528	1.455	-0.510	0.610
ATP9A	0.328	1.388	0.766	2.512	1.082	0.279
BU193864	-0.169	0.844	0.532	1.341	-0.718	0.473
KPRP	-1.241	0.289	0.041	2.030	-1.248	0.212
C20orf70	0.080	1.083	0.366	3.209	0.144	0.885
AICDA	-0.257	0.774	0.493	1.215	-1.115	0.265
GPR128	-0.098	0.907	0.483	1.703	-0.304	0.761
DA028946	-0.319	0.727	0.404	1.308	-1.064	0.288
WFDC6	0.258	1.294	0.755	2.219	0.938	0.348
PNMA5	0.419	1.520	0.515	4.486	0.758	0.448
SNORD18B	-0.181	0.834	0.470	1.481	-0.619	0.536
MYL1	-0.371	0.690	0.176	2.702	-0.533	0.594
LOC643719	0.379	1.460	0.897	2.377	1.523	0.128
C17orf98	0.910	2.483	0.803	7.677	1.579	0.114
SCGB1A1	0.213	1.237	0.374	4.090	0.348	0.728
INSM2	1.507	4.512	1.117	18.224	2.115	0.034
ECEL1	-1.156	0.315	0.099	1.001	-1.958	0.050
GIF	-0.474	0.623	0.187	2.072	-0.772	0.440
WDR17	-0.093	0.911	0.543	1.527	-0.354	0.723
OTOS	1.556	4.741	1.517	14.811	2.677	0.007
BF514583	-0.179	0.836	0.529	1.321	-0.766	0.444
AI382167	-0.231	0.794	0.411	1.534	-0.686	0.493
HBE1	0.509	1.663	0.363	7.619	0.655	0.512
RHAG	0.930	2.535	0.550	11.680	1.193	0.233
MATN1	-1.374	0.253	0.054	1.196	-1.734	0.083
CT45A5	-0.039	0.962	0.401	2.307	-0.087	0.931
C3P1	-0.090	0.914	0.395	2.119	-0.209	0.834
EMILIN3	-0.725	0.484	0.182	1.287	-1.454	0.146
DB306783	-0.397	0.672	0.333	1.357	-1.108	0.268
SLC14A1	-1.067	0.344	0.165	0.715	-2.857	0.004
LCN15	1.643	5.173	1.546	17.302	2.668	0.008
BX096436	-0.338	0.713	0.474	1.074	-1.619	0.106
KRT1	0.566	1.761	0.283	10.972	0.606	0.544
ERAS	-0.389	0.678	0.089	5.179	-0.375	0.708
LOC100288238	-0.001	0.999	0.562	1.775	-0.004	0.997
OR6C4	-0.269	0.764	0.486	1.200	-1.169	0.243
CYP3A7	0.992	2.695	0.678	10.723	1.407	0.159
AA625691	-0.401	0.670	0.405	1.109	-1.558	0.119
BE856720	-0.099	0.906	0.595	1.379	-0.460	0.645
GFAP	0.084	1.087	0.261	4.534	0.115	0.909
WAKMAR2	-0.276	0.759	0.460	1.252	-1.080	0.280
AW296252	-0.141	0.869	0.544	1.387	-0.589	0.556
ACTC1	1.276	3.583	1.417	9.055	2.697	0.007
ZP4	-0.475	0.622	0.202	1.915	-0.828	0.408
AA724305	0.218	1.243	0.782	1.975	0.921	0.357
BCHE	0.287	1.332	0.798	2.223	1.097	0.273
KRT27	-0.714	0.489	0.107	2.245	-0.919	0.358
TMED6	-0.721	0.486	0.166	1.428	-1.312	0.190
CHRNA9	0.300	1.350	0.753	2.422	1.007	0.314

Concordance	0.574
Partial AIC	15090.440
log-likelihood ratio test	85.948 on 50 df
-log2(p) of ll-ratio test	9.725

Fig. 26: Cox proportional hazards report for transcriptomic data. The report indicates that *OS_MONTHS* was the duration variable, while *OS_STATUS* was the event variable used for survival analysis. The figure also reports the HR values (exp(coef)), with the corresponding 95% confident interval, and p-values of the 50 extracted features. The accuracy prediction of the CPH model, i.e. the C-index, was 0.574, which indicates an acceptable model. Similar to the results presented in Fig. 25, the genes *LCN15*, *OTOS*, and *INSM2* were identified as the top three most significant factors associated with the death event with a p-value less than 0.05, and coefficient/log(HR) value of 1.643, 1.556, and 1.507, respectively.

5. Finally, patients in the testing set were ranked in descending order based on their predicted risk scores, and split into two groups according to the median values. The comparisons between the two groups (high-risk and low-risk groups) for all the four algorithms was performed using Kaplan–Meier curves and log-rank test.

The details of the set-up and evaluation of ML models codes are the same as in Sect. 3.4.4. The outcomes of the four algorithms are presented in Fig. 27 and Fig. 28. RSF had the highest C-index value of 0.53, followed by GBS, SSVM, and CPH.

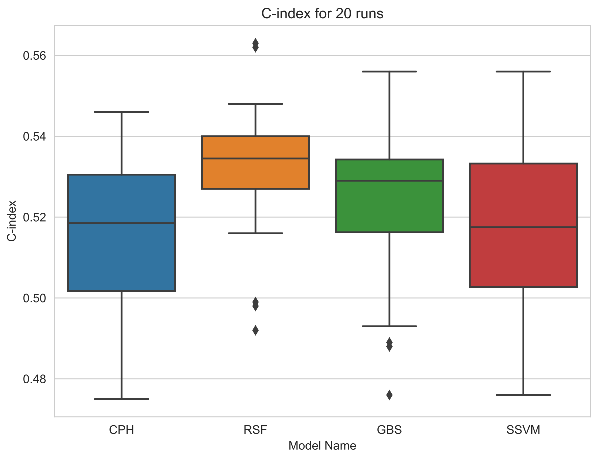


Fig. 27: C-index comparisons for Experiment 2. Boxplots of C-index results of transcriptomic data using CPH, RSF, GBS, and SSVM. The experiments were replicated 20 times. In each experiment, the data was randomly divided into training and testing sets with a ratio of 80:20, while guaranteeing the same censoring percentage on each set of data. RSF was found to have the highest median C-index, followed by GBS, SSVM, and CPH.

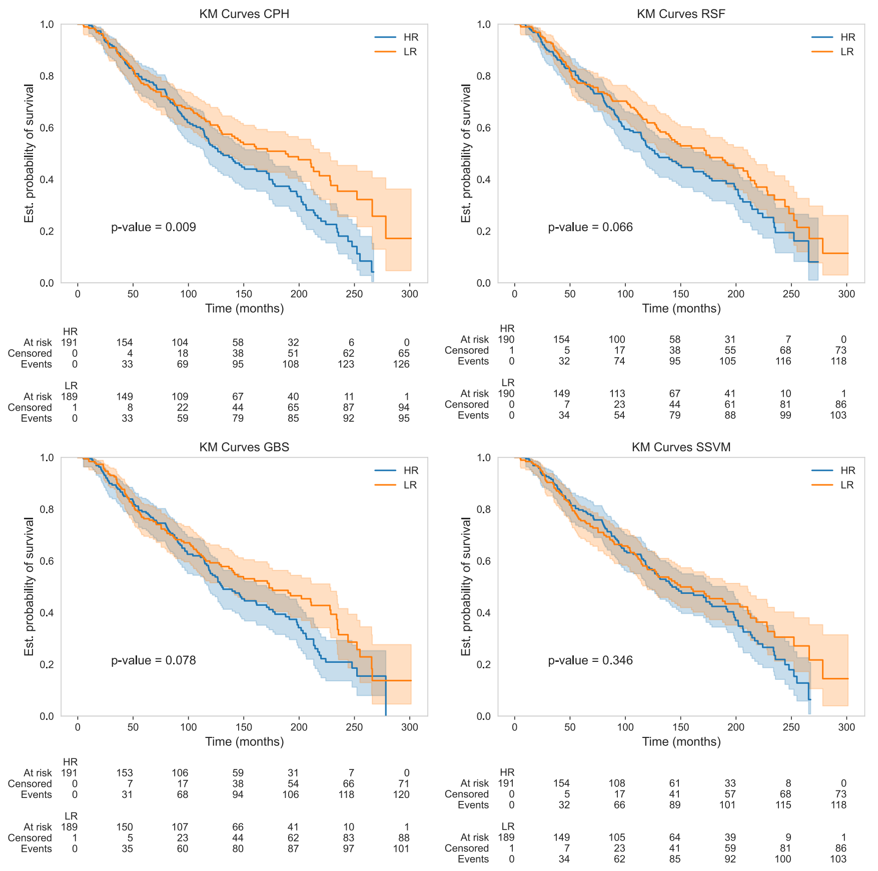
The Kaplan–Meier curves for breast cancer patients in the testing set according to their predicted prognostic score using 50 features revealed that only the CPH model reported a significant difference in the survival distributions of high-risk and low-risk patients with a p-value of 0.009, as shown in Fig. 29. This result might be due to the number of features selected in the preprocessing steps. For this reason, several approaches have been proposed to select the optimal subset of features and achieve more accurate and robust results [77, 78, 79].

```
 Cox Regression
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model', CoxPHSurvivalAnalysis(alpha=0.001))])
C-index for test set (Hold out): 0.5157661773365008
Average C-index for 20 runs 0.5149500000000001

 Random Forest Survival
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model',
                 RandomSurvivalForest(max_depth=8, max_features='sqrt',
                                      min_samples_leaf=50,
                                      min_samples_split=100, n_estimators=500,
                                      random_state=5))])
C-index for test set (Hold out): 0.5301415199691377
Average C-index for 20 runs 0.53135

 Gradient Boosting Survival
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model',
                 GradientBoostingSurvivalAnalysis(n_estimators=1000,
                                                  random_state=5))])
C-index for test set (Hold out): 0.5120302125845161
Average C-index for 20 runs 0.52285

 SVM Survival
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model',
                 FastSurvivalSVM(max_iter=500, optimizer='avltree',
                                 random_state=5))])
C-index for test set (Hold out): 0.4973096992954458
Average C-index for 20 runs 0.5169000000000001
```

Fig. 28: ML models results for 50 selected genes on the transcriptomic data. The selected hyperparameters, initial test results, and the average C-index of each model are displayed in the output. Overall, the average performance over 20 runs of the three ML-based models outperformed the CPH model on the analysis of transcriptomic data for survival prediction. RSF had the highest average C-index with a value of 0.530, followed by GBS, SSVM, and CPH.

3.5.6 Interpret model

In order to provide an interpretation of the results of models (Step 7 Fig. 1), we computed and plotted the SHAP values for all the features in the test set. The SHAP values for the top 20 features are shown in Fig. 30. A single patient is represented by each data point. The y-axis lists the top 20 most influential genes in descending order, represented by its Hugo symbol. The x-axis reports the corresponding SHAP values



KM Curves CPH

	0	50	100	150	200	250	300
HR							
At risk	191	154	104	58	32	6	0
Censored	0	4	18	38	51	62	65
Events	0	33	69	95	108	123	126
LR							
At risk	189	149	109	67	40	11	1
Censored	1	8	22	44	65	87	94
Events	0	33	59	79	85	92	95

KM Curves RSF

	0	50	100	150	200	250	300
HR							
At risk	190	154	100	58	31	7	0
Censored	1	5	17	38	55	68	73
Events	0	32	74	95	105	116	118
LR							
At risk	190	149	113	67	41	10	1
Censored	0	7	23	44	61	81	86
Events	0	34	54	79	88	99	103

KM Curves GBS

	0	50	100	150	200	250	300
HR							
At risk	191	153	106	59	31	7	0
Censored	0	7	17	38	54	66	71
Events	0	31	68	94	106	118	120
LR							
At risk	189	150	107	66	41	10	1
Censored	1	5	23	44	62	83	88
Events	0	35	60	80	87	97	101

KM Curves SSVM

	0	50	100	150	200	250	300
HR							
At risk	191	154	108	61	33	8	0
Censored	0	5	17	41	57	68	73
Events	0	32	66	89	101	115	118
LR							
At risk	189	149	105	64	39	9	1
Censored	1	7	23	41	59	81	86
Events	0	34	62	85	92	100	103

Fig. 29: Kaplan-Meier curves to compare high-risk and low-risk breast cancer groups, stratified by predicted survival risk score based on the transcriptomic data when using 50 features. The low-risk group includes patients with predicted risk scores above the median value, while the high-risk group comprises patients with predicted risk scores lower than the median value. The p-value from the log-rank test was calculated to statistically determine the difference in survival functions between the two groups. The figure shows that only the CPH model showed a statistically significant difference between risk groups with a p-value of 0.009.

for a specific observation in the testing set. The higher the SHAP value, the higher the mortality risk of the patient represented by the data point. The colour represents low or high gene expression values. Particularly, the genes *LCN15* and *AA625691* were identified among the top 10 features by the four models. *OTOS* was selected by

CPH, RSF, and SSVM. High values of this gene had a positive impact on the models' outcome (i.e., high values of this gene correlates with higher risk of experiencing the event of interest). All these genes were associated with patient survival and could represent useful prognostic biomarkers for breast cancer patients. Most of the gene features in the CPH model were convergent and had SHAP values distribution around 0, indicating no significant influence on the outcome of the model.

3.6 Experiment 3: integrating clinical to transcriptomic data

In the last experiment presented in our tutorial, clinic information (from Experiment 1) and transcriptomic data (from Experiment 2) were integrated to improve the predictive power of the ML models. The workflow followed in this experiment is similar to the one previously followed in Sect. 3.4 and Sect. 3.5. First, data was loaded and cleaned before performing EDA. Next, the CPH results were visualised, and the reports were extracted for further analysis. Then, we prepared the data for survival analysis and constructed the ML models for training and evaluating models performance. Finally, the outcomes were interpreted to identify the important markers associated with low survival.

3.6.1 Load data

The data used for this experiment are derived from the data already used for Experiment 1 (Sect. 3.4) and Experiment 2 (Sect. 3.5), i.e., the encoded clinical data and the 50 Hugo Symbol extracted using mRMR. There were 1,977 and 1,903 observations in the preprocessed clinical data and transcriptomic data, respectively. We extracted the matching observations between these two datasets and used them in this experiment.

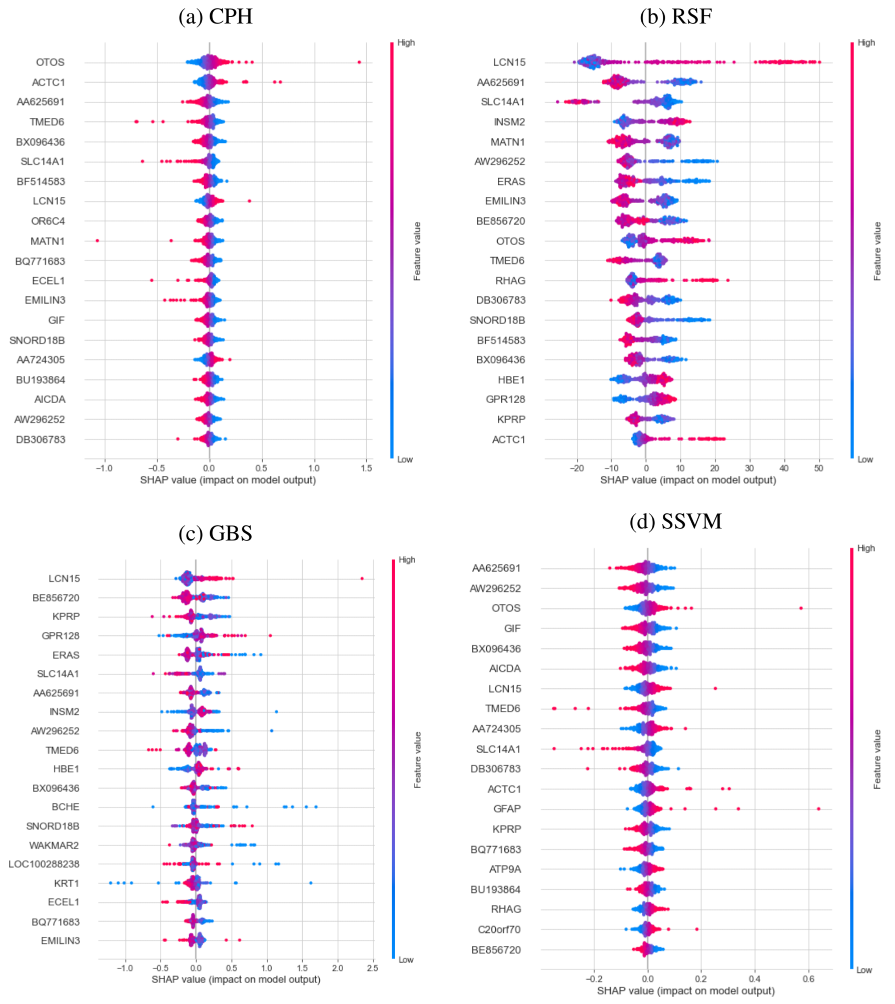


Fig. 30: SHAP summary plot for transcriptomic data for (a) CPH, (b) RSF, (c) GBS, and (d) SSVM models. For each gene feature, a single patient is represented by each data point. The y-axis lists the top 20 prognostic biomarkers and presents them in descending order based on the ranking provided by the mean of their absolute SHAP values. The x-axis reports the SHAP value indicating the impact of the feature on the prediction of the algorithm for a specific observation in the testing set. The colour represents the value of the feature for each patient. The higher the SHAP value the patient had, the higher the risk of death. The genes *OTOS* and *AA625691* were respectively found as the most important predictors for the CPH and SSVM models, while *LCN15* was identified as the top most significant feature for the RSF and GBS models.

```
# Load data
file1 = pd.read_csv('Data\clinical.csv')
file2 = pd.read_csv('Data\GeneID_MRMR_50.csv')

# Merge clinical data
data = pd.merge(file1,file2, how="inner", on=["PATIENT_ID"])
```

Then, an overview of the information and the first five rows of the merged data frame was displayed. As shown in Fig. 31, there were 79 columns and 1,903 rows in the new data frame. No missing values were found in the final dataset.

```
# Have a quick look at data
data.info()
data.head()
```

3.6.2 Preprocess and explore data

For this experiment, it is optional to check the duplicate and missing values since the data was already processed in the previous two experiments. However, it is always a good practice to conduct the preprocessing step after loading data (Step 3 in Fig. 1). The *PATIENT_ID* column was removed from the dataset before moving into the next step of the pipeline. As shown in Fig. 32, the new shape of the data was 1903 rows and 78 columns. No duplicates and missing values were found in the final dataset.

```
# Preprocess data & Explore data
# Check duplicate values
print('The number of duplicate values in data', data.duplicated()
    .sum())

# Drop unused cols: Based on data.info(), we will drop some
    unused cols and null cols
drop_list = ['PATIENT_ID']
```

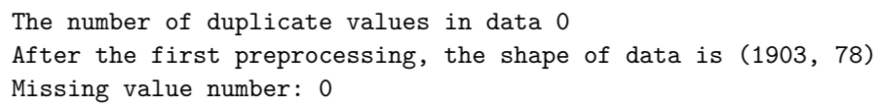
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1903 entries, 0 to 1902
Data columns (total 79 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   CELLULARITY                   1903 non-null   float64
 1   HER2_SNP6                     1903 non-null   float64
 2   INFERRED_MENOPAUSAL_STATE     1903 non-null   float64
 3   INTCLUST                      1903 non-null   float64
 4   THREEGENE                     1903 non-null   float64
 5   CHEMOTHERAPY                  1903 non-null   float64
 6   ER_IHC                        1903 non-null   float64
 7   HORMONE_THERAPY               1903 non-null   float64
 8   CLAUDIN_SUBTYPE               1903 non-null   float64
 9   LATERALITY                    1903 non-null   float64
 10  RADIO_THERAPY                 1903 non-null   float64
 11  HISTOLOGICAL_SUBTYPE          1903 non-null   float64
 12  BREAST_SURGERY                1903 non-null   float64
 13  CANCER_TYPE_DETAILED          1903 non-null   float64
 14  ER_STATUS                     1903 non-null   float64
 15  HER2_STATUS                   1903 non-null   float64
 16  ONCOTREE_CODE                 1903 non-null   float64
 17  PR_STATUS                     1903 non-null   float64
 18  LYMPH_NODES_EXAMINED_POSITIVE 1903 non-null   float64
 19  NPI                           1903 non-null   float64
 20  AGE_AT_DIAGNOSIS              1903 non-null   float64
 21  COHORT                        1903 non-null   float64
 22  GRADE                         1903 non-null   float64
 23  TUMOR_SIZE                    1903 non-null   float64
 24  TUMOR_STAGE                   1903 non-null   float64
```

Fig. 31: Output of the integrated clinical and transcriptomic data information. The output gives an overview of the merged data frame, including total entries, data types, names of columns, and number of validated data points. There are 1,903 entries and 79 columns in the merged data frame. The first 25 features are shown in this figure. There are no missing values in the final dataset.

.

```
df = data.drop(drop_list, axis=1)
print('After the first preprocessing, the shape of data is', data
    .shape)
```

```
# Check missing values again
print('Missing value number:', df.isna().sum().sum())
```



```
The number of duplicate values in data 0
After the first preprocessing, the shape of data is (1903, 78)
Missing value number: 0
```

Fig. 32: Output of the integrated clinical and transcriptomic data preprocessing. After removing *PATIENT_ID* column, the new dataset consists of 1,903 rows and 78 columns. There are no duplicates and missing values in the final dataset

.

Next, the correlation matrix was plotted to provide more insights into the relationships of features in the merged dataset (Fig. 33). Except for some pair of clinical features such as *ER_STATUS* and *ER_IHC*, no high correlated values were observed between clinical and transcriptomic features.

```
# Correlation analysis
colormap = plt.cm.Reds
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(),linewidths=0.1,vmax=0.8,
            square=True, cmap = colormap, linecolor='white')
plt.title('Correlation matrix', fontsize=14)
plt.show()
```

In the next step, the follow-up survival time distribution was plotted. Overall, the time distribution plot for this experiment was similar to the one observed in Experiment 2 and shown in Fig. 34. There were 42% censored observations in the integrated clinical and transcriptomic data.

```
# Time Distribution of Death and Censored
num_censored = df.shape[0] - df["OS_STATUS"].sum()
```

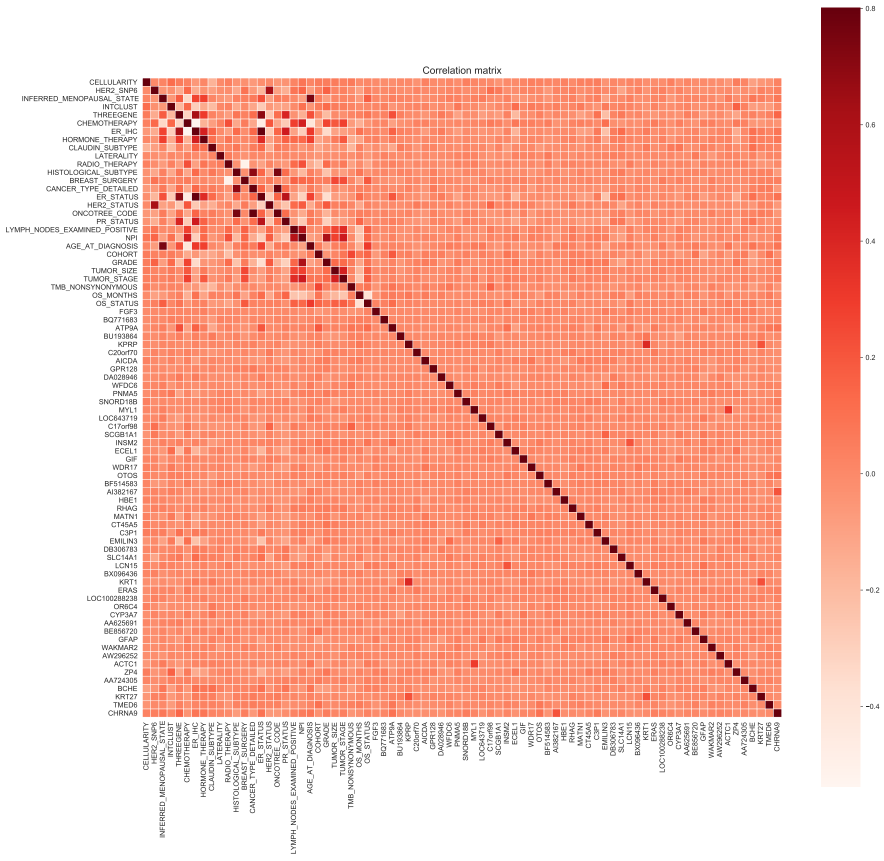


Fig. 33: Correlation matrix of clinical and transcriptomic features. The correlation matrix depicts the linear correlation between all the pairs of attributes and ranges from -1 (perfect negative correlation) to +1 (perfect positive correlation), with the value of zero being no correlation between the features. Colour density represents the correlation values, where a darker colour implies a higher value and a lighter colour implies a lower value. Except for some pair of clinical features having high correlation values (as previously seen in Fig. 10), no high correlated values were observed between clinical and transcriptomic features.

```
print("%.1f%% of records are censored" % (num_censored/df.shape
    [0]*100))

plt.figure(figsize=(10, 6))
val, bins, patches =
plt.hist((df.query('OS_STATUS == 1')['OS_MONTHS'],
        df.query('OS_STATUS == 0')['OS_MONTHS']),
        bins=30, stacked=True)
_ = plt.legend(patches, ["Time of Death", "Time of Censored"])
plt.title("Time Distribution of Censored and Death Patients")
```

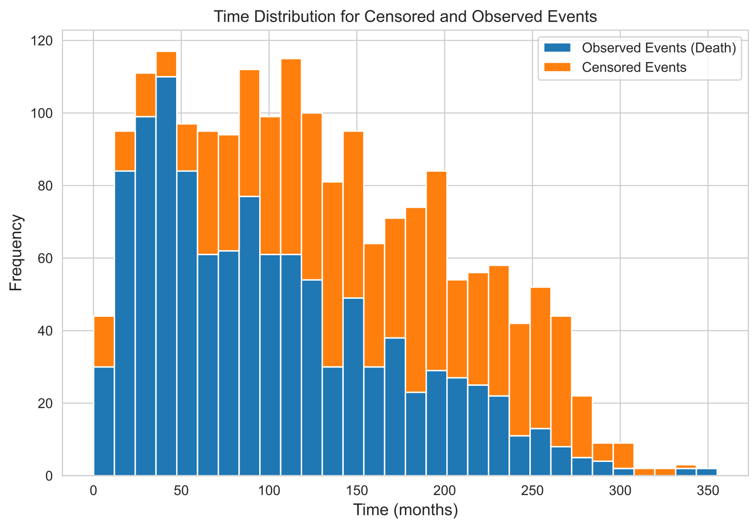


Fig. 34: Distribution of follow-up times of censored and uncensored (death) observations in the integrated clinical and transcriptomic data. The data contains 42% of censored observations. The distribution is right-skewed and it is different between censored patients and those who experienced an event. The censored group has more patients with longer survival times.

The rest of Experiment 3 includes plotting the CPH model, preparing and evaluating the ML algorithms, and interpreting the results. These were set up as previously done in Experiment 1 in Sect. 3.4.3, 3.4.4, and Sect. 3.4.5. Hence, we do not repeat the code in the sections below, but we discuss and interpret the results. However, the complete notebook can be accessed at https://github.com/Angione-Lab/survival_analysis_tutorial.

3.6.3 Plot Cox proportional hazards model

Following the same approach presented in Sect. 3.4.3 in Experiment 1 (Step 5 in Fig. 1), the merged data was normalised and the CPH model was fitted to generate the log(HR)s and the final statistical report. Fig. 35 shows that *AGE_AT_DIAGNOSIS* was identified as the most significant factor associated with a higher probability of experiencing the event, with a log(HR) value of 3.800. In contrast, the genes *ECEL1* and *KPRP* were found negatively associated with the death event as shown by their negative log(HR) values. Patients having higher expression values for these two genes tend to live longer compared to those who show lower expression values of the same genes.

3.6.4 Set up and evaluate machine learning algorithms

Following the same steps (Step 6 in Fig. 1) described in Sect. 3.4.4, data was prepared by splitting it into training and testing sets. The experiment pipeline was the same as in Experiments 1 and 2. Firstly, the ML models were trained using a grid search approach with five-fold cross-validation to identify the optimal hyperparameters. Next, the fitted models were evaluated on the testing set. The data was split into training and testing sets and the evaluation were repeated 20 times to obtain an average C-index. Finally, Kaplan–Meier curves and log-rank tests were applied to statistically compare the differences between the two predicted risk groups. The high-risk group contained the patients in the testing set with the expected risk scores above the median value. In contrast, the low-risk group included the patients with the expected risk score below the median value. This process helped us to identify the optimal algorithm that could successfully estimate the survival risk of the cancer patient.

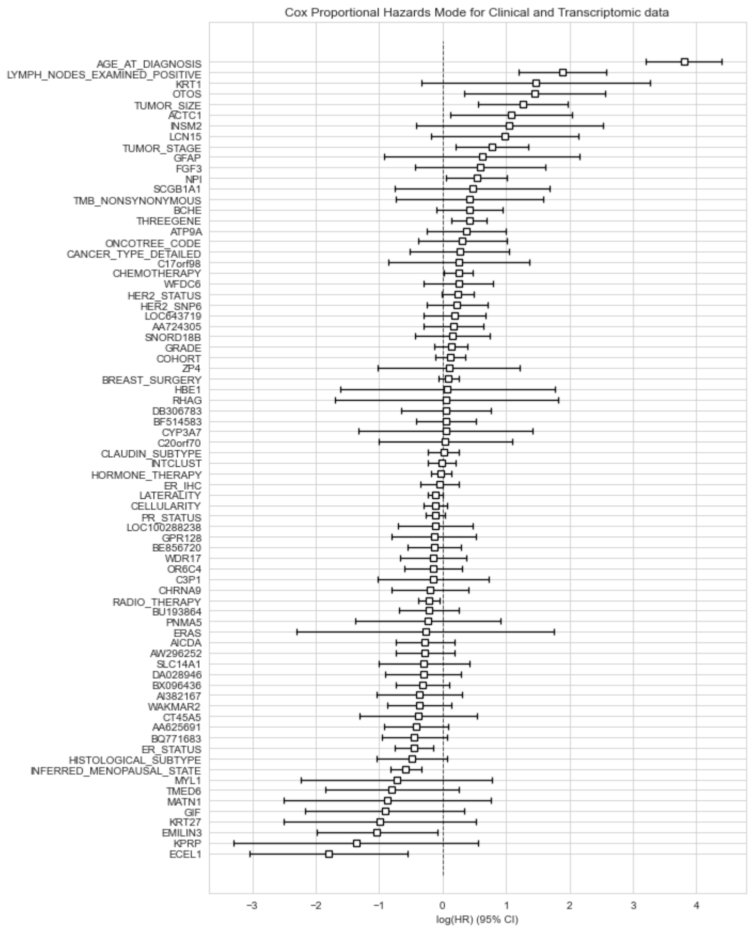


Fig. 35: Results of the CPH model applied to the merged clinical and transcriptomic dataset. *AGE_AT_DIAGNOSIS* was identified as the most significant factors associated with the death events with a log(HR) value of 3.800, followed by *LYMPH_NODES_EXAMINED_POSITIVE*, genes *KRT1*, *OTOS*, and *ACTC1*. In contrast, the negative HR value predictors, such as gene *ECEL1* and *KPRP*, were negatively associated with the death event. Patients with higher values of these factors tend to live longer compared to those who have lower expression values of those genes.

Fig. 36 and Fig. 37 show the results of Experiment 3. RSF was the best performing model with an average C-index value of 0.683, followed by GBS, SSVM, and CPH, with C-index equal to 0.675, 0.673, 0.670, respectively. Kaplan–Meier curves, reported in Fig. 38, show that all four models had statistically significant differences in survival distributions between risk groups. RSF was again the best survival algorithm with the lowest p-value of 1.197E-14.

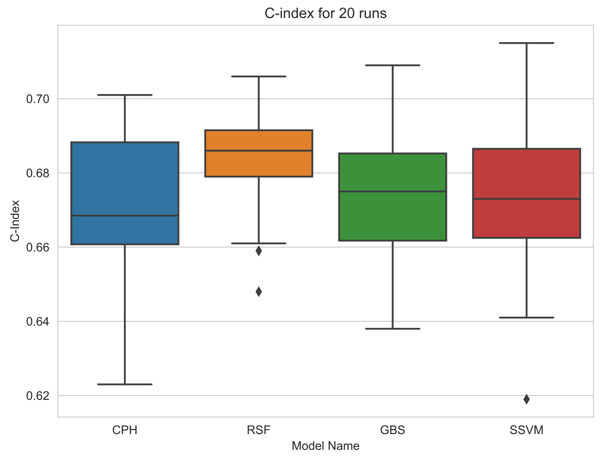


Fig. 36: C-index comparisons for Experiment 3. Boxplots of C-index results of the integrated clinical and transcriptomic data using CPH, RSF, GBS, and SSVM. The experiments were replicated 20 times. In each experiment, the data was randomly divided into training and testing sets with a ratio of 80:20, while guaranteeing the same censoring percentage in each splitting. On average, RSF was found to have the highest median C-index, followed by GBS, SSVM, and CPH.

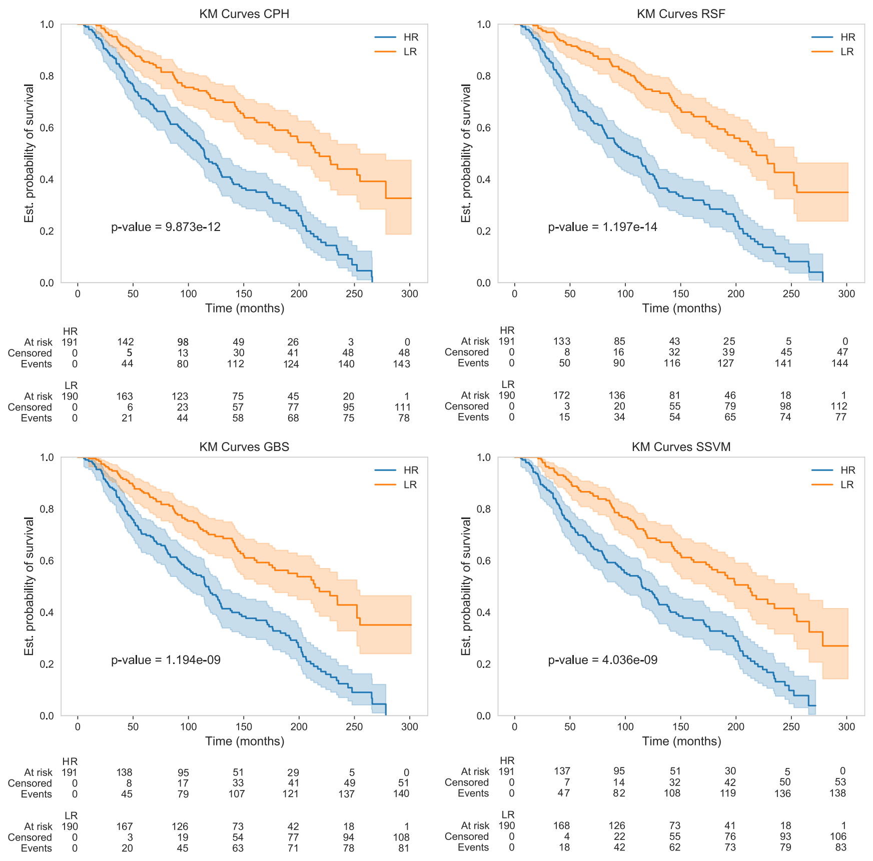
3.6.5 Interpret model

For the final step of the pipeline (Step 7 in Fig. 1), SHAP values were used to interpret the results of the models. The SHAP plots of the top 20 most important features are reported in Fig. 39. A single patient is represented by each data point for each feature. The y-axis lists the top 20 most influential features in descending order, while the x-axis reports their corresponding SHAP values for a specific observation in the testing set. The higher the SHAP value associated with a patient, the higher

```
 Cox Regression
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model', CoxPHSurvivalAnalysis(alpha=1))])
C-index for test set (Hold out): 0.6594013249366157
Average C-index for 20 runs 0.66945

 Random Forest Survival
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model',
                 RandomSurvivalForest(max_depth=8, max_features='sqrt',
                                      min_samples_leaf=50,
                                      min_samples_split=100, n_estimators=500,
                                      random_state=5))])
C-index for test set (Hold out): 0.6730187290422834
Average C-index for 20 runs 0.6833

 Gradient Boosting Survival
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model',
                 GradientBoostingSurvivalAnalysis(learning_rate=0.01,
                                                  n_estimators=800,
                                                  random_state=5))])
C-index for test set (Hold out): 0.6474809847059786
Average C-index for 20 runs 0.67505

 SVM Survival
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('model',
                 FastSurvivalSVM(max_iter=500, optimizer='avltree',
                                 random_state=5))])
C-index for test set (Hold out): 0.6683978081295494
Average C-index for 20 runs 0.6729499999999999
```

Fig. 37: ML models results for the integration clinical and transcriptomic data. The selected hyperparameter, initial test result, and average C-index of each model are displayed in the outcome. Overall, the average performance over 20 runs of the three ML models outperformed CPH when using the integrated data for survival prediction. RSF had the highest average C-index with a value of 0.683, followed by GBS, SSVM, and CPH.

the mortality risk that the patient would have. *AGE_AT_DIAGNOSIS* was selected among the top features by the four models, suggesting the high impact of this feature on the survival outcome. Fig. 39 also identifies some critical biomarkers affecting the prediction outcomes of algorithms, including genes *ERAS, SLC14A1*, and *LCN15*. Specifically, high values of *ERAS* and *SLC14A1* had a negative impact on the outcome of the models (i.e., high expression values of these genes correlated negatively with



KM Curves CPH

	0	50	100	150	200	250	300
HR							
At risk	191	142	98	49	26	3	0
Censored	0	5	13	30	41	48	48
Events	0	44	80	112	124	140	143
LR							
At risk	190	163	123	75	45	20	1
Censored	0	6	23	57	77	95	111
Events	0	21	44	58	68	75	78

KM Curves RSF

	0	50	100	150	200	250	300
HR							
At risk	191	133	85	43	25	5	0
Censored	0	8	16	32	39	45	47
Events	0	50	90	116	127	141	144
LR							
At risk	190	172	136	81	46	18	1
Censored	0	3	20	55	79	98	112
Events	0	15	34	54	65	74	77

KM Curves GBS

	0	50	100	150	200	250	300
HR							
At risk	191	138	95	51	29	5	0
Censored	0	8	17	33	41	49	51
Events	0	45	79	107	121	137	140
LR							
At risk	190	167	126	73	42	18	1
Censored	0	3	19	54	77	94	108
Events	0	20	45	63	71	78	81

KM Curves SSVM

	0	50	100	150	200	250	300
HR							
At risk	191	137	95	51	30	5	0
Censored	0	7	14	32	42	50	53
Events	0	47	82	108	119	136	138
LR							
At risk	190	168	126	73	41	18	1
Censored	0	4	22	55	76	93	106
Events	0	18	42	62	73	79	83

Fig. 38: Kaplan-Meier curves to compare high-risk and low-risk groups, stratified by predicted survival risk scores. The low-risk group (n=190) included patients with predicted risk scores above the median value, while the high-risk group (n=190) comprised patients with risk scores calculated to determine the statistical difference between the survival distributions of the two groups. The figure shows statistically significant differences between survival groups for all four models with a p-value lower than 0.0001.

the probability of experiencing the event), while high values of *LCN15* showed a positive impact on the outcome of models (i.e., high values of this gene correlated positively with the probability of experiencing the event). All these genes were associated with patient survival and could be useful prognostic biomarkers for breast cancer patients.

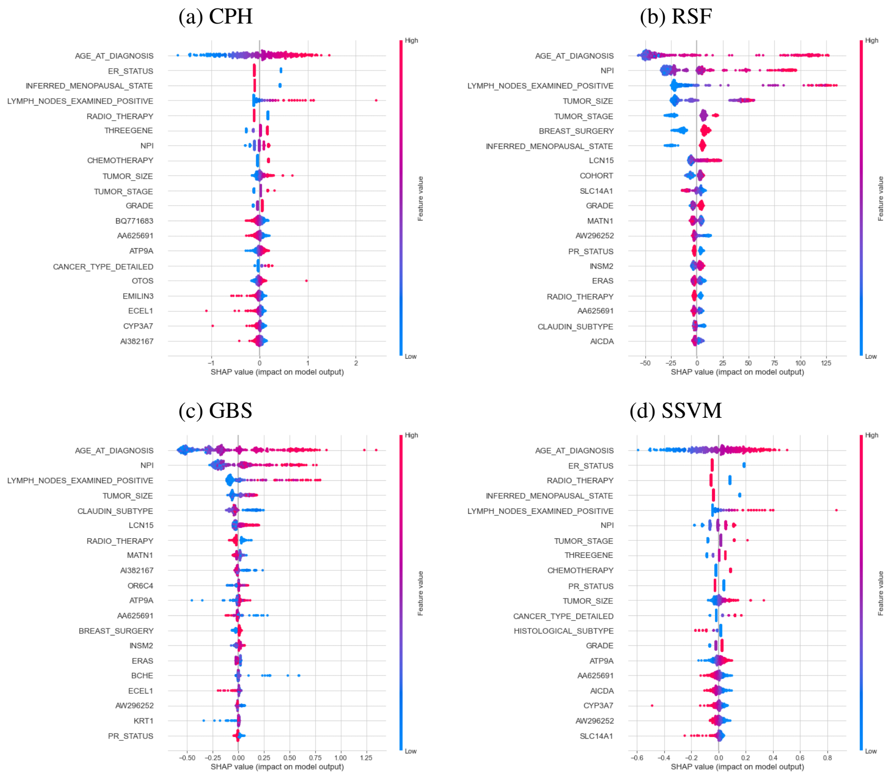


Fig. 39: SHAP summary plot for the integrated clinical and transcriptomic data for (a) CPH, (b) RSF, (c) GBS, and (d) SSVM models. For each gene feature, a single patient is represented by each data point. The y-axis lists the top prognostic biomarkers and presents them in descending order based on their ranking provided by the mean of their absolute SHAP values. The x-axis reports the SHAP value indicating the impact of the feature on the algorithm's prediction outcome for a specific observation in the testing set. The colour represents the value of the feature for each instance. The higher the SHAP value associated with the patient, the higher the risk of death. For the survival risk predictors, *AGE_AT_DIAGNOSIS* was consistently selected by the four models as the top significant factors impacting the outcome of models. Specifically, high values of this feature correlated with a higher probability of experiencing the event. Other biomarkers such as *ERAS, SLC14A1*, and *LCN15* were identified as features having a high impact on the prediction outcomes and associated with the predicted survival likelihood.

4 Conclusions

The application of ML models on the integration of clinical and omics data can provide data insights to improve personalised treatment and precision oncology. However, there are still some challenges to overcome, mainly related to the high dimensionality of the data and the heterogeneity of samples. Hence, better approaches to develop accurate predictive models and identify critical prognostic markers need to be implemented. In this tutorial, we showed that ML models appear as salient and successful methods to analyse medical data and predict patient-specific survival outcomes. Our study proposed a step-by-step protocol to design and evaluate the traditional statistical model CPH and three ML models for breast cancer survival, i.e., RSF, GBS, and SSVM. The performance of the ML models was assessed using the METABRIC dataset. The presented pipeline, based on optimising C-index by using a grid search approach and a five-fold cross-validation method, has a great potential to improve the performance of models and generalise the models for survival prediction on unseen data. Furthermore, we used SHAP values to interpret the model results and identify the features that had the highest impact on the prediction outcomes of models. The improvement in ML interpretability will help researchers and clinicians understand more about ML models and thus gain more credibility and trust. This tutorial represents one step further to bring these novel solutions to clinicians and to the public. Our work offers an exploratory strategy to enhance the biological understanding of the prognosis predictive ML models.

We conducted three different experiments for clinical data, transcriptomic data, as well as the integration of these two data types. Incorporating clinical and mRNA expression data is crucial to uncover a sequence of complicated interactions in multiple biological processes and complex human conditions. Due to the high-dimensional nature of transcriptomic data, mRMR was applied as a feature selection technique. This preprocessing step also helps to boost the performance of models, save computational resources, and reduce overfitting.

Even if we presented the most used ML techniques to perform survival analysis on different types of data, there are some limitations to this tutorial. We only considered three ML algorithms, namely RSF, GBS, and SSVM, because of their popularity and effectiveness in analysing survival data. However, other approaches based on deep learning, a branch of ML, have also proved their capability to work with survival data. Some packages are available to run deep learning-based models for cancer prognosis, such as Deepsurv [80], Cox-nnet [43], and DeepProg [81]. A competitive performance comparison between our approaches to other deep learning-based models could enable researchers to explore and obtain optimal ways to supplement conventional survival analysis techniques.

The number of features selected in our study could also have limited the findings when using transcriptomic data. To save computation time and resources, we only extracted 50 features to demonstrate our approach. Future studies could adopt our framework and repeat our steps exploring different numbers of features.

In summary, by performing survival analysis across different models and data, our results revealed that ML approaches were capable of generating accurate prognostic predictions. The ML-based models showed a better performance compared to traditional statistic methods, i.e., CPH model. Particularly, RSF reported the best performance results in analysing the transcriptomic data (Experiment 2) and the integrated clinical and transcriptomic data (Experiment 3), while SSVM was the best performing model when using clinical data only (Experiment 1).

Acknowledgements

AO and CA acknowledge the support of Earlier.org through their Research Grant "Application of computational models of breast cancer for early-detection personalised tests". CA acknowledges the support of EPSRC and The Alan Turing Institute through their Turing Network Development Award, and the Children’s Liver Disease Foundation through their Research Grant.

References

[1] Ferlay J, Héry C, Autier P, Sankaranarayanan R (2010) Global burden of breast cancer. In: Breast cancer epidemiology, Springer, pp 1–19

[2] Cancer Research UK (2021) Breast cancer statistics. URL https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/breast-cancer

[3] Office for National Statistics (2019) Cancer survival in England Cancer survival in England: national estimates for patients followed up to 2017. URL https://www.ons.gov.uk/peoplepopulationandcommunity/healthandsocialcare/conditionsanddiseases/bulletins/cancersurvivalinengland/nationalestimatesforpatientsfollowedupto2017

[4] Robson M, Im SA, Senkus E et al. (2017) Olaparib for metastatic breast cancer in patients with a germline brca mutation. New England Journal of Medicine 377(6):523–533

[5] De Bin R, Sauerbrei W, Boulesteix AL (2014) Investigating the prediction ability of survival models based on both clinical and omics data: two case studies. Statistics in medicine 33(30):5310–5329

[6] Hira MT, Razzaque M, Angione C et al. (2021) Integrated multi-omics analysis of ovarian cancer using variational autoencoders. Scientific reports 11(1):1–16

[7] Conesa A, Beck S (2019) Making multi-omics data accessible to researchers. Scientific data 6(1):1–4

[8] Vijayakumar S, Conway M, Lió P, Angione C (2018) Optimization of multi-omic genome-scale models: Methodologies, hands-on tutorial, and perspectives. Metabolic Network Reconstruction and Modeling pp 389–408

[9] Angione C (2019) Human systems biology and metabolic modelling: a review—from disease metabolism to precision medicine. BioMed research in-

ternational 2019

[10] Zhao Z, Zhang KN, Wang Q et al. (2021) Chinese glioma genome atlas (cgga): a comprehensive resource with functional genomic data from chinese glioma patients. Genomics, proteomics & bioinformatics 19(1):1

[11] Iuliano A, Occhipinti A, Angelini C et al. (2018) Combining pathway identification and breast cancer survival prediction via screening-network methods. Frontiers in genetics 9:206

[12] Győrffy B (2021) Survival analysis across the entire transcriptome identifies biomarkers with the highest prognostic power in breast cancer. Computational and structural biotechnology journal 19:4101–4109

[13] Higdon R, Earl RK, Stanberry L et al. (2015) The promise of multi-omics and clinical data integration to identify and target personalized healthcare approaches in autism spectrum disorders. Omics: a journal of integrative biology 19(4):197–208

[14] Hasin Y, Seldin M, Lusis A (2017) Multi-omics approaches to disease. Genome biology 18(1):1–15

[15] Yaneske E, Angione C (2018) The poly-omics of ageing through individual-based metabolic modelling. BMC bioinformatics 19(14):83–96

[16] Yan J, Risacher SL, Shen L, Saykin AJ (2018) Network approaches to systems biology analysis of complex disease: integrative methods for multi-omics data. Briefings in bioinformatics 19(6):1370–1381

[17] Occhipinti A, Hamadi Y, Kugler H et al. (2020) Discovering essential multiple gene effects through large scale optimization: an application to human cancer metabolism. IEEE/ACM transactions on computational biology and bioinformatics

[18] Eyassu F, Angione C (2017) Modelling pyruvate dehydrogenase under hypoxia and its role in cancer metabolism. Royal Society open science 4(10):170360

[19] Zhao L, Dong Q, Luo C et al. (2021) Deepomix: A scalable and interpretable multi-omics deep learning framework and application in cancer survival analysis. Computational and structural biotechnology journal 19:2719–2725

[20] Yaneske E, Zampieri G, Bertoldi L et al. (2021) Genome-scale metabolic modelling of sars-cov-2 in cancer cells reveals an increased shift to glycolytic energy production. FEBS letters 595(18):2350–2365

[21] Angione C (2018) Integrating splice-isoform expression into genome-scale models characterizes breast cancer metabolism. Bioinformatics 34(3):494–501

[22] Anaya J, Reon B, Chen WM et al. (2016) A pan-cancer analysis of prognostic genes. PeerJ 3:e1499

[23] Zhu B, Song N, Shen R et al. (2017) Integrating clinical and multiple omics data for prognostic assessment across human cancers. Scientific reports 7(1):1–13

[24] Islam MM, Haque MR, Iqbal H et al. (2020) Breast cancer prediction: a comparative study using machine learning techniques. SN Computer Science 1(5):1–14

[25] Zampieri G, Vijayakumar S, Yaneske E, Angione C (2019) Machine and deep learning meet genome-scale metabolic modeling. PLoS computational biology 15(7):e1007084

[26] Alabi RO, Elmusrati M, Sawazaki-Calone I et al. (2020) Comparison of supervised machine learning classification techniques in prediction of locoregional recurrences in early oral tongue cancer. International journal of medical informatics 136:104068

[27] Culley C, Vijayakumar S, Zampieri G, Angione C (2020) A mechanism-aware and multiomic machine-learning pipeline characterizes yeast cell growth. Proceedings of the National Academy of Sciences 117(31):18869–18879

[28] Chugh G, Kumar S, Singh N (2021) Survey on machine learning and deep learning applications in breast cancer diagnosis. Cognitive Computation pp

1–20

[29] Akram M, Iqbal M, Daniyal M, Khan AU (2017) Awareness and current knowledge of breast cancer. Biological research 50(1):1–23

[30] Simmons CP, McMillan DC, McWilliams K et al. (2017) Prognostic tools in patients with advanced cancer: a systematic review. Journal of pain and symptom management 53(5):962–970

[31] Ascolani G, Occhipinti A, Liò P (2015) Modelling circulating tumour cells for personalised survival prediction in metastatic breast cancer. PLoS computational biology 11(5):e1004199

[32] Wang P, Li Y, Reddy CK (2019) Machine learning for survival analysis: A survey. ACM Computing Surveys (CSUR) 51(6):1–36

[33] Mariotto AB, Noone AM, Howlader N et al. (2014) Cancer survival: an overview of measures, uses, and interpretation. Journal of the National Cancer Institute Monographs 2014(49):145–186

[34] Austin PC (2017) A tutorial on multilevel survival analysis: methods, models and applications. International Statistical Review 85(2):185–203

[35] Iuliano A, Occhipinti A, Angelini C et al. (2016) Cancer markers selection using network-based cox regression: a methodological and computational practice. Frontiers in physiology 7:208

[36] Yang Y, Lu Q, Shao X et al. (2018) Development of a three-gene prognostic signature for hepatitis b virus associated hepatocellular carcinoma based on integrated transcriptomic analysis. Journal of Cancer 9(11):1989

[37] Kiebish MA, Cullen J, Mishra P et al. (2020) Multi-omic serum biomarkers for prognosis of disease progression in prostate cancer. Journal of translational medicine 18(1):1–10

[38] Hao J, Kim Y, Mallavarapu T et al. (2019) Interpretable deep neural network for cancer survival analysis by integrating genomic and clinical data. BMC

medical genomics 12(10):1–13

[39] Moncada-Torres A, van Maaren MC, Hendriks MP et al. (2021) Explainable machine learning can outperform cox regression predictions and provide insights in breast cancer survival. Scientific Reports 11(1):1–13

[40] Akai H, Yasaka K, Kunimatsu A et al. (2018) Predicting prognosis of resected hepatocellular carcinoma by radiomics analysis with random survival forest. Diagnostic and interventional imaging 99(10):643–651

[41] Bibault JE, Chang DT, Xing L (2021) Development and validation of a model to predict survival in colorectal cancer using a gradient-boosted machine. Gut 70(5):884–889

[42] Wang H, Zheng B, Yoon SW, Ko HS (2018) A support vector machine-based ensemble algorithm for breast cancer diagnosis. European Journal of Operational Research 267(2):687–699

[43] Ching T, Zhu X, Garmire LX (2018) Cox-nnet: an artificial neural network method for prognosis prediction of high-throughput omics data. PLoS computational biology 14(4):e1006076

[44] Huang Z, Zhan X, Xiang S et al. (2019) Salmon: survival analysis learning with multi-omics neural networks on breast cancer. Frontiers in genetics 10:166

[45] Cheon S, Agarwal A, Popovic M et al. (2016) The accuracy of clinicians' predictions of survival in advanced cancer: a review. Ann Palliat Med 5(1):22–29

[46] Pereira B, Chin SF, Rueda OM et al. (2016) The somatic mutation profiles of 2,433 breast cancers refine their genomic and transcriptomic landscapes. Nature communications 7(1):1–16, DOI https://doi.org/10.1038/ncomms11479

[47] Lundberg SM, Lee SI (2017) A unified approach to interpreting model predictions. In: Proceedings of the 31st international conference on neural information processing systems, pp 4768–4777

[48] Singh R, Mukhopadhyay K (2011) Survival analysis in clinical trials: Basics and must know areas. Perspectives in clinical research 2(4):145

[49] Cox DR (1972) Regression models and life-tables. Journal of the Royal Statistical Society: Series B (Methodological) 34(2):187–202

[50] Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS (2008) Random survival forests. The annals of applied statistics 2(3):841–860

[51] Breiman L (2001) Random forests. Machine learning 45(1):5–32

[52] Azar AT, Elshazly HI, Hassanien AE, Elkorany AM (2014) A random forest classifier for lymph diseases. Computer methods and programs in biomedicine 113(2):465–473

[53] Qu Z, Li H, Wang Y et al. (2020) Detection of electricity theft behavior based on improved synthetic minority oversampling technique and random forest classifier. Energies 13(8):2039

[54] Harrell FE, Califf RM, Pryor DB et al. (1982) Evaluating the yield of medical tests. Jama 247(18):2543–2546

[55] Hothorn T, Bühlmann P, Dudoit S et al. (2006) Survival ensembles. Biostatistics 7(3):355–373

[56] Natekin A, Knoll A (2013) Gradient boosting machines, a tutorial. Frontiers in neurorobotics 7:21

[57] Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Annals of statistics pp 1189–1232

[58] Ridgeway G (1999) The state of boosting. Computing science and statistics pp 172–181

[59] Khan FM, Zubek VB (2008) Support vector regression for censored data (svrc): a novel tool for survival analysis. In: 2008 Eighth IEEE International Conference on Data Mining, IEEE, pp 863–868

[60] Vapnik V (1999) The nature of statistical learning theory. Springer science & business media

[61] Pölsterl S, Navab N, Katouzian A (2015) Fast training of support vector machines for survival analysis. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, pp 243–259

[62] Leger S, Zwanenburg A, Pilz K et al. (2017) A comparative study of machine learning methods for time-to-event survival data for radiomics risk modelling. Scientific reports 7(1):1–11

[63] Gárate-Escamila AK, El Hassani AH, Andrès E (2020) Classification models for heart disease prediction using feature selection and pca. Informatics in Medicine Unlocked 19:100330

[64] Ewees AA, Al-qaness MA, Abualigah L et al. (2021) Boosting arithmetic optimization algorithm with genetic algorithm operators for feature selection: Case study on cox proportional hazards model. Mathematics 9(18):2321

[65] Schemper M, Kaider A, Wakounig S, Heinze G (2013) Estimating the correlation of bivariate failure times under censoring. Statistics in medicine 32(27):4781–4790

[66] Su Z, Tang B, Liu Z, Qin Y (2015) Multi-fault diagnosis for rotating machinery based on orthogonal supervised linear local tangent space alignment and least square support vector machine. Neurocomputing 157:208–222

[67] Rodrigues D, Pereira LA, Nakamura RY et al. (2014) A wrapper approach for feature selection based on bat algorithm and optimum-path forest. Expert Systems with Applications 41(5):2250–2258

[68] Peng H, Long F, Ding C (2005) Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on pattern analysis and machine intelligence 27(8):1226–1238

[69] Curtis C, Shah SP, Chin SF et al. (2012) The genomic and transcriptomic architecture of 2,000 breast tumours reveals novel subgroups. Nature 486(7403):346–352

[70] Pölsterl S (2020) scikit-survival: A library for time-to-event analysis built on top of scikit-learn. J Mach Learn Res 21(212):1–6

[71] Van Rossum G, Drake FL (2009) Python 3 Reference Manual. CreateSpace, Scotts Valley, CA

[72] Kim B, Khanna R, Koyejo OO (2016) Examples are not enough, learn to criticize! Criticism for Interpretability. Advances in Neural Information Processing Systems 29

[73] Lundberg SM, Nair B, Vavilala MS et al. (2018) Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. Nature biomedical engineering 2(10):749–760

[74] Aittokallio T (2010) Dealing with missing values in large-scale studies: microarray data imputation and beyond. Briefings in bioinformatics 11(2):253–264

[75] Fryett JJ, Inshaw J, Morris AP, Cordell HJ (2018) Comparison of methods for transcriptome imputation through application to two common complex diseases. European Journal of Human Genetics 26(11):1658–1667

[76] Shahjaman M, Rahman MR, Islam T et al. (2021) rmisbeta: A robust missing value imputation approach in transcriptomics and metabolomics data. Computers in Biology and Medicine 138:104911

[77] Park S, Shin B, Shim WS et al. (2019) Wx: a neural network-based feature selection algorithm for transcriptomic data. Scientific reports 9(1):1–9

[78] Han Y, Huang L, Zhou F (2021) Zoo: Selecting transcriptomic and methylomic biomarkers by ensembling animal-inspired swarm intelligence feature selection algorithms. Genes 12(11):1814

[79] Iuliano A, Occhipinti A, Angelini C et al. (2021) Cosmonet: An r package for survival analysis using screening-network methods. Mathematics 9(24):3262

[80] Katzman JL, Shaham U, Cloninger A et al. (2018) Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. BMC medical research methodology 18(1):1–12

[81] Poirion OB, Jing Z, Chaudhary K et al. (2021) Deepprog: an ensemble of deep-learning and machine-learning models for prognosis prediction using multi-omics data. Genome medicine 13(1):1–15