Bone Marrow Transplant in Children

Group Members: Himanshu Gandhi (1770138), Venkata Shreya Kala (1764875), Shubham Prasad Sahoo (1824661), Pleezent Brass (1803833)

1 Abstract

Pediatric bone marrow transplantation represents a critical treatment avenue for various hematologic disorders, including malignant and non-malignant conditions. This research delves into understanding the variables that influence the success of unmanipulated allogeneic unrelated donor hematopoietic stem cell transplantation in pediatric patients. Our investigation seeks to assess how various factors influence the overall success of transplantation outcomes, considering a wide spectrum of patient demographics, medical history, transplantation details, and recovery metrics.

Utilizing classification and regression techniques, this research endeavours to discern key elements influencing both short-term and long-term outcomes in pediatric bone marrow transplants. The findings from this investigation seek to provide valuable insights into optimizing transplant procedures, enhancing patient care, and potentially refining predictive strategies for pediatric hematologic conditions.

2 Introduction

Advancements in healthcare, particularly in pediatric medicine, have witnessed the integration of machine learning (ML) into critical domains like prognostics and treatment planning. An expanding area that demands precision and accuracy is the projection and success prediction of bone marrow transplants in children. In pediatric hematologic disorders, bone marrow transplantation is a crucial intervention for various malignant and non-malignant conditions. The success of these procedures heavily relies on intricate variables, as the intricate nature of these variables poses a challenge in accurately predicting the transplantation outcome.

In this study, we delve into the realm of pediatric bone marrow transplantation, aiming to harness the power of ML algorithms to distinguish and evaluate critical factors by uncovering patterns and relationships among them, impacting the success of these procedures. We aim to gauge their effectiveness in predicting and optimizing the success of bone marrow transplants in children.

3 Data

The dataset retrieved from the UCI Machine Learning Repository comprises a vital compilation defining pediatric hematologic disorders and bone marrow transplantation. With 187 instances and 36 features, it provides an extensive understanding of pediatric healthcare. These encompass a broad spectrum, encompassing patient and donor demographics, comprehensive medical records, specific details on transplant procedures, and

resultant outcomes. The dataset comprises Integer, Categorical, and Binary data types, reflecting various essential information crucial for investigating transplant success factors. Figure 1 is a brief description of some of the important features of the dataset.

4 Methodology

4.1 Data Pre-processing and Analysis

Data pre-processing is a crucial step in the data analysis pipeline, encompassing various techniques to enhance the quality of the raw data for analysis. The primary objectives of data pre-processing include addressing issues such as missing values, outliers, and irrelevant information, as well as preparing the data for specific analytical tasks. Python was used for the analysis.

4.1.1 Null Value Handling

Missing values cannot be used in ML algorithms. Given the limited size of the dataset, the chosen strategy involved incorporating ML models to deal with these. The count is given in Figure 2.

- Regressor Imputation for Numerical Features: Random Forest Regressor was deployed to impute missing values in numerical features, namely 'CD3dkgx10d8' and 'Rbodymass'. The model predicted missing values based on the relationships observed in the other features, providing a data-driven imputation.
- For Categorical Features:
  - Mode Imputation: For categorical features with low null values, such as 'RecipientABO', 'Antigen', 'Allele', 'RecipientRh', 'ABOmatch', and 'DonorCMV', the mode (most frequent value) was used for imputation.
  - Classifier Imputation: Random Forest Classifier is used to impute missing values in features with high null values - 'extcGvHD', 'CMVstatus', and 'RecipientCMV'.

4.1.2 Outlier Treatment

1. Transformations help mitigate the impact of extreme values, making the data more amenable to modelling. They are given in Figures 3-6.
   'Rbodymass': Log transformation was implemented.
   'CD34kgx10d6': Capped the two smallest values just above 1 followed by a log transformation.
   'CD3dCD34': Cube root transformation with an upper cap at 0.97 quantile.
   'CD3dkgx10d8': Square root transformation capped at 0.95 quantile
2. 'ANCrecovery', 'PLTrecovery', and 'time to aGvHDIIIIV' exhibit outlier values ranging from 8 to 26, with a drastic spike to values in the millions. We've retained these outliers to observe their impact on the model's behaviour. Subsequently, we may contemplate categorizing these features if they prove beneficial for our analysis.

4.1.3 Feature Engineering

1. Utilized one-hot encoding for nominal features such as 'DonorABO' and 'RecipientABO', converting them into a binary numerical representation.

2. Employed ordinal encoding for ordinal features like 'HLAmatch', 'Antigen', 'Allele', 'HLAgrI', and 'Recipientageint', ensuring a meaningful numerical order.

4.1.4 Feature Selection

Correlation Heatmap: A correlation analysis with 'survival status' was conducted and it is given in Figure 7. No features were removed as all exhibited low correlation, indicating that each feature contributes unique information to survival_status.

Multicollinearity: Utilized Variance Inflation Factor (VIF) with a threshold of 10 to detect multicollinearity. Identified 'CD3dCD34', 'Recipientage', and 'Rbodymass' with VIF values exceeding the threshold. Based on multicollinearity concerns, 'CD3dCD34' was removed, while 'Recipientage' and 'Rbodymass' were retained due to their theoretical importance.

Redundant Feature Removal: Based on properties and relevance, we removed redundant features namely, 'DonorABO', 'RecipientABO', 'HLAmismatch', 'Recipientage10', 'Recipientageint', 'aGvHDIIIIV', 'DonorABO', 'RecipientABO' and 'Donorage35'.

4.2 Modeling Techniques

Classification models (Logistic Regression, Decision Tree, SVM) and XGBoost were used to analyze the impact of explanatory variables on survival status. Categorical/binary columns were standardized before splitting the dataset into training and testing sets. Additionally, we created learning curves to visualize each model's performance. (Figures 15-23).

4.2.1 Logistic Regression

Logistic Regression is a classification algorithm used to predict binary variables from the relationship it models between the features and the binary target variable using the logistic function. The logistic function maps the input features to a probability score of the positive class. A logistic regression model was created using 26 features, considering survival status as the target variable. A significance threshold of 0.05 was set to identify features significantly associated with the target variable using Wald's test. Backward feature elimination was employed for feature selection, and models were designated as logistic regression 1, 2, and 3, representing different selection methods.

4.2.2 Decision Tree

The Decision Tree, a supervised learning algorithm for classification tasks, uses data labels to build a tree structure. Internal nodes represent features, while leaf nodes hold class labels. The tree splits data based on feature values, maximizing information gain or minimizing Gini impurity. [1]. Hyperparameter optimization was done via Grid Search in Sci-kit Learn, configuring parameters like minimum samples for node splitting, samples for leaf nodes, and

maximum depth. Subsequent models refer to the original and feature-selected versions as model 1 and model 2, respectively. The visualizations are given as Figures 13 and 14.

4.2.3 Support Vector Machine

SVM constructs a hyperplane to separate classes, minimizing error. Support vectors, the closest points to the hyperplane, determine its position. The goal is to maximize margins, ensuring a clear class boundary. Kernel methods map data to higher dimensions for an optimal hyperplane. Optimized parameters include trade-off control, kernel choice, and coefficients influencing the decision boundary. [2]

4.2.4 XGBoost

XGBoost, an ensemble method, combines decision trees to enhance accuracy. It uses gradient boosting to build sequential, interpretable trees, correcting errors and updating residuals. Optimal parameters, including maximum tree depth, learning rate, and training instance fraction, were selected to prevent overfitting. The final prediction is derived from the sum of all tree predictions.

5 Results and Inferences

A classification report summarizes the performance of a classification model by providing metrics such as precision (accuracy of positive predictions), recall (ability to identify positive instances), F1 score (a balanced measure of precision and recall), and accuracy. The formulae are given in the Appendix as Figures 9 and 10. For the logistic regression, the likelihood ratio test for each model against the null models was performed. However, the p-value was 1 at all times and the maximum number of iterations was set very high for feature selection. The models were termed the best depending on the highest accuracy followed by a better precision, recall, and F1 score and considering the lesser difference in the training and testing accuracy if the former is higher. The best outputs from each model are given in Table 1.

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression-1	0.79	0.74	0.89	0.81
Decision Tree-1	0.66	0.62	0.84	0.71
SVM-2	0.74	0.68	0.89	0.77
XGBoost-1	0.79	0.72	0.95	0.82

Table 1: Best Evaluation Metrics for Classification

For the Logistic Regression, Decision tree, and XGBoost, all features were selected but for SVM - 'ABOmatch', 'DonorCMV', 'RecipientCMV', 'Allele', 'Recipientage', 'extcGvHD', 'CD34kgx10d6', 'CD3dkgx10d8', 'Rbodymass', 'PLTrecovery', 'Stemcellsource_1', 'Relapse_1', and 'aGvHDIIIIV' features were selected. The outputs of the confusion matrix and Wald's test for logistic regression are given in Table 2 and Figure 8. The logistic regression models didn't show signs of overfitting, yet encountered convergence issues.

However, the other models demonstrated slight overfitting, with XGBoost exhibiting a wider margin in comparison. The training accuracy of both XGBoost models was perfect. The XGBoost model achieved good accuracy, the satisfactory precision indicates that the predicted positive cases are true positives, the high recall suggests a lower likelihood of missing actual positives, and the balanced F1 score indicates false positives and negatives are minimal. The feature importance from the best XGBoost model are given in Figure 11 (for those of model-1 refer to Figure 12).

6 Impact of CD34 Cells

Our investigation also aimed to validate the hypothesis that an increased dosage of CD34+ cells per kilogram extends overall survival time while avoiding concurrent adverse events impacting patients' quality of life. Rigorous analysis focused on specific features—Relapse, aGvHDIIIIV, and extcGvHD—each representing their absence. The primary objective was to construct a predictive model for Survival time, emphasizing the augmentation of CD34+ cell dosage. Our findings unveiled a noteworthy trend in the relationship between CD34+ cells and disease types. Notably, a significant linear correlation was evident between CD34+ cell dosage and Non-Malignant disease cases (Figure 24). However, in Malignant disease cases, the relationship exhibited less linearity (Figure 25). Specifically, a higher dose of infused CD34+ cells demonstrated an association with improved outcomes in the low-risk disease subgroup. This analysis is in line with the research conducted in this paper [4].

To evaluate the model's performance excluding undesirable events, such as the recipient developing acute or chronic graft versus host disease, a subset of test values meeting this criterion was isolated. Subsequently, a linear regression model was employed to establish the slope and intercept of the best-fit line for the data points. This analysis provided insights into the relationship between CD34+ cell dose and survival time, distinctly for malignant and non-malignant diseases.

7 Conclusion

In essence, our exploration into pediatric bone marrow transplantation uncovered pivotal factors influencing transplant success. The investigation scrutinized many variables and their profound impact on transplantation outcomes. Noteworthy among our findings was the discernible influence of CD34+ cell dosage on treatment efficacy, particularly in different disease subgroups. This highlighted the nuanced relationship between dosage escalation and improved outcomes, underscoring its potential in specific patient cohorts. Despite minor model convergence challenges, limited samples, and mild overfitting, our analysis yields valuable insights for predicting and optimizing bone marrow transplant outcomes in children. These insights promise to refine prognostic approaches, elevate patient care standards, and contribute to enhancing the well-being of pediatric patients undergoing hematologic treatments.

8 References and Appendices

References

[1] A. Navlani, "Python decision tree classification tutorial: Scikit-Learn Decisiontreeclassifier," DataCamp, https://www.datacamp.com/tutorial/decision-tree-classification-python (accessed Nov. 30, 2023).

[2] A. Navlani, "Scikit-learn SVM tutorial with Python (Support Vector Machines)," DataCamp, https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python (accessed Nov. 30, 2023).

[3] P. Huilgol, "Precision and recall: Essential metrics for machine learning (2023 update)," Analytics Vidhya, https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/#What_is_Precision? (accessed Dec. 1, 2023).

[4] Y. Yokoyama et al., "A high CD34+ cell dose is associated with better disease-free survival in patients with low-risk diseases undergoing peripheral blood stem cell transplantation from HLA-matched related donors," Nature News, https://www.nature.com/articles/s41409-020-0817-5 (accessed Dec. 9, 2023).

8.1 Appendices

8.1.1 Tables and Figures

For detailed and better visualizations please refer to the code at this link. The pre-processing is at this link. They can be viewed at Section 8.1.2 too.

Model Variant	True Positive	False Positive	True Negative	False Negative
1	13	2	17	6
2	9	2	17	10
3	12	2	17	7

Table 2: Logistic Regression Confusion Matrix Outputs for Testing Data

Recipientgender	Male - 1, Female - 0
Recipientage	Age of the recipient of hematopoietic stem cells at the time of transplantation
Rbodymass	Body mass of the recipient of hematopoietic stem cells at the time of transplantation
Disease	Type of disease (ALL, AML, Chronic, Non-malignant, lymphoma)
Relapse	Reoccurrence of the disease (No - 0, Yes - 1)
Txpostrelapse	The second bone marrow transplantation after relapse (No - 0; Yes - 1)
Stemcellsource	Source of hematopoietic stem cells (Peripheral blood - 1, Bone marrow - 0)
ABOMatch	Compatibility of hematopoietic stem cells according to ABO blood group (matched - 1, mismatched - 1)
HLAmatch	Compatibility of antigens of the main histocompatibility complex of hematopoietic stem cells based on criteria (10/10 - 0, 9/10 - 1, 8/10 - 2, 7/10 - 3 (allele/antigens))
CD3dCD34	CD3+ cell to CD34+ cell ratio
ANCrecovery	Time to neutrophils recovery defined as neutrophils count >0.5 x 10^9/L
PLTrecovery	Time to platelet recovery defined as platelet count >50000/mm3
aGvHDIIIIV	Development of acute graft versus host disease stage III or IV (Yes - 0, No - 1)
Survival Time	Time of observation (if alive) or time to event (if dead) in days
Survival Status	Survival status (0 - alive, 1 - dead)

Figure 1: An overview of some of the important features

Features	RecipientABO	RecipientRh	ABOmatch	CMVstatus	DonorCMV	RecipientCMV	Antigen	Allele	extcGvHD	CD3dCD34	CD3dkgx10d8	Rbodymass
Null values	1	2	1	16	2	14	1	1	31	5	5	2

Figure 2: Null Values Count

Figure 3: Transforming Rbodymass

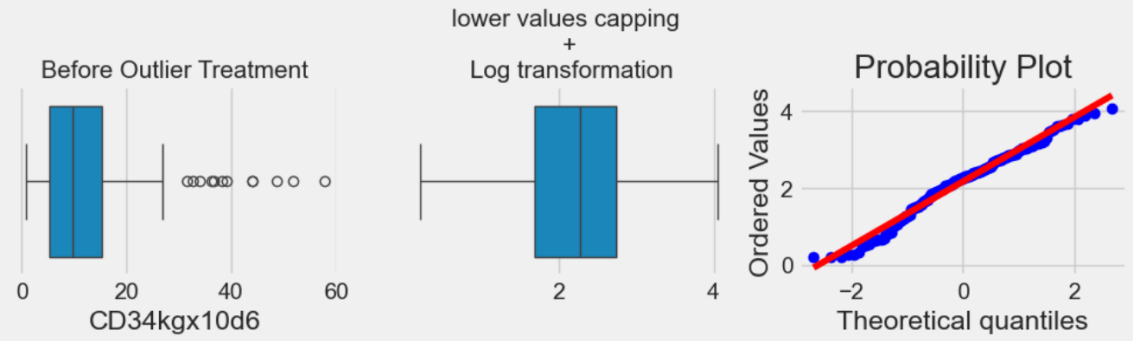

Figure 4: Transforming CD34kgx10d6

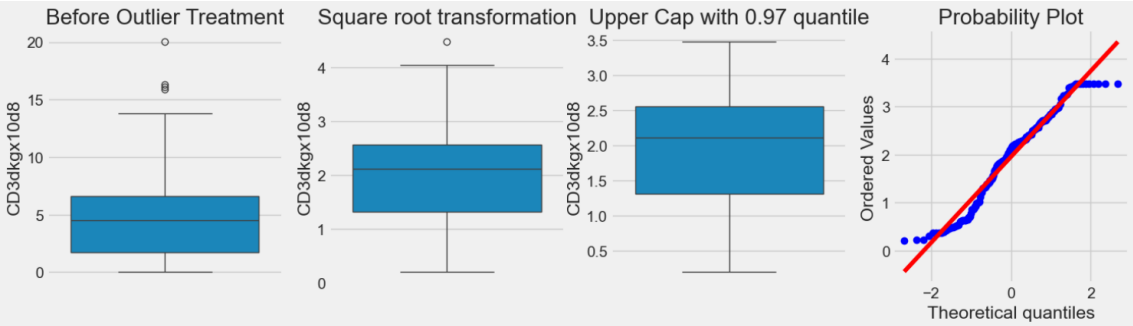

Figure 5: Transforming CD3dkgx10d8

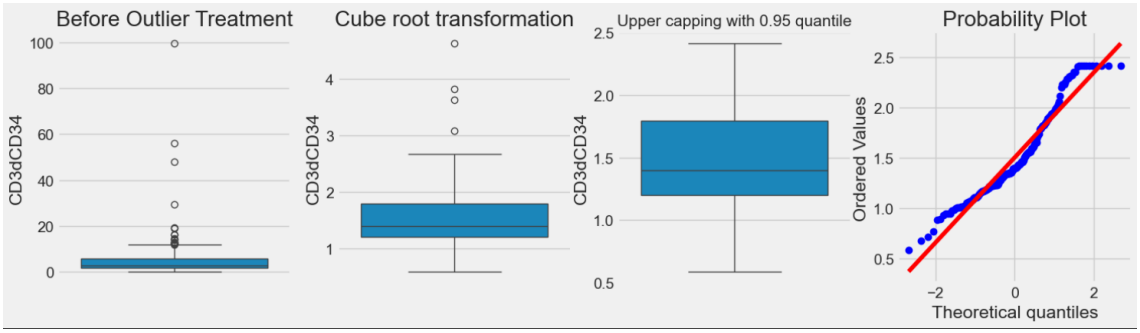

Figure 6: Transforming CD3dCD34

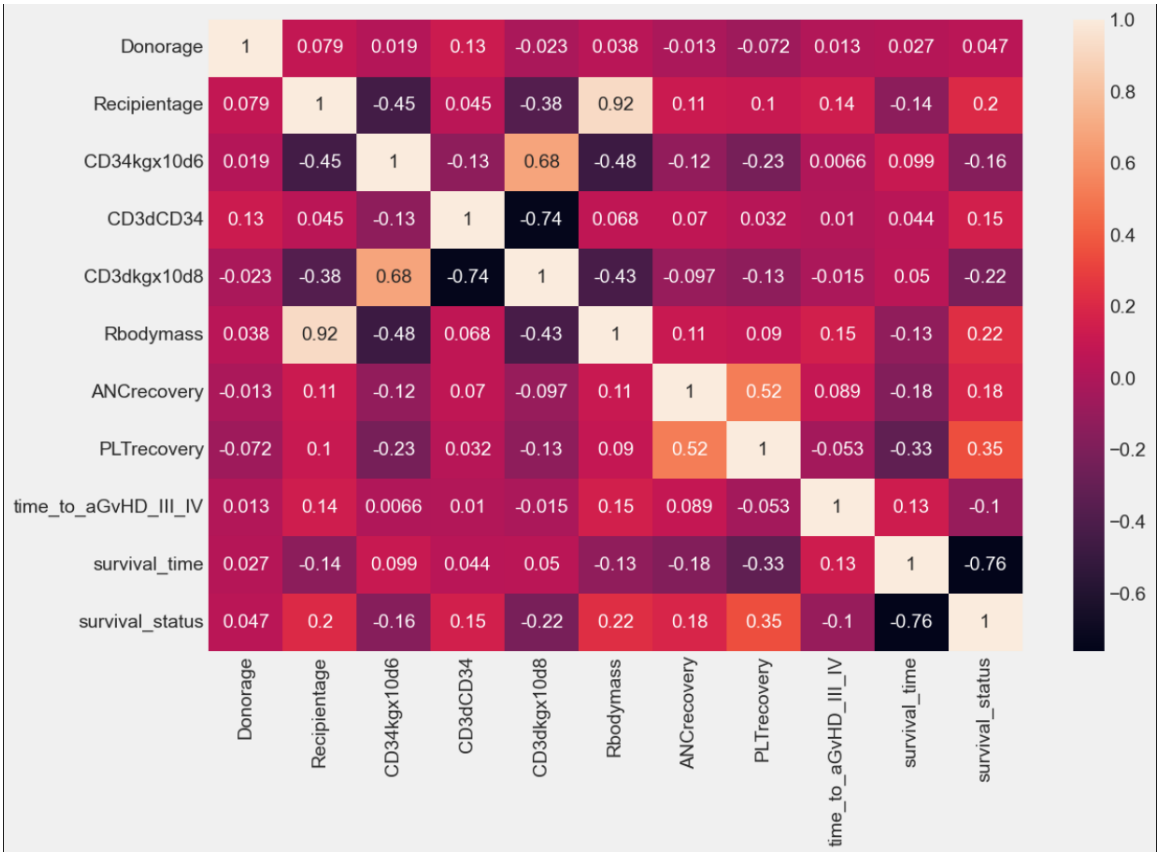

Figure 7: Correlation Heatmap

	chi2	P>chi2	df constraint
const	[[5.507416777372648e-11]]	0.9999940787413373	1
Donorage	[[2.1020110649040937]]	0.14710554321619818	1
RecipientRh	[[2.7509228554429788]]	0.09719831254809713	1
ABOmatch	[[0.09471382768810216]]	0.7582679152760617	1
CMVstatus	[[0.000475737775229027]]	0.9825983893980016	1
DonorCMV	[[0.06289889522079571]]	0.80197143151018	1
RecipientCMV	[[0.00398274140583886]]	0.9496797358567537	1
HLAmatch	[[0.3259453598971426]]	0.5680564178671794	1
Antigen	[[0.0231629220993435]]	0.8790341633231659	1
Allele	[[1.7013477519911173]]	0.19211181483494427	1
HLAgrI	[[0.0001584077249823693]]	0.9895580796102132	1
Recipientage	[[0.40235811196084775]]	0.5258739357174129	1
extcGvHD	[[8.89955135931423]]	0.0028524073005794107	1
CD34kgx10d6	[[1.3066478015807177]]	0.25300248718788915	1
CD3dkgx10d8	[[2.59586246112627]]	0.1071431011608487	1
Rbodymass	[[2.7021142528642708]]	0.10021527052824417	1
ANCrecovery	[[1.7876800537908695e-12]]	0.9999989331952058	1
PLTrecovery	[[1.8601391945312808e-08]]	0.9998911789825791	1
Recipientgender_1	[[0.5947794821256316]]	0.44057685125855994	1
Stemcellsource_1	[[0.8389326303154623]]	0.3597022175910555	1
IIIV_1	[[0.07765197254649699]]	0.780504967806342	1
Gendermatch_1	[[0.6022662877871828]]	0.43771463873916305	1
Riskgroup_1	[[1.9423490198466316]]	0.16341370804099603	1
Txpostrelapse_1	[[1.5457288270516954e-06]]	0.9990080115792009	1
Diseasegroup_1	[[0.11140818176238185]]	0.7385466013796285	1
Relapse_1	[[16.52212088173859]]	4.808579204801541 6e-05	1
aGvHDIIIIV_1	[[4.046881538844908]]	0.04425302173211861	1

Figure 8: Walds Test for Logistic Regression-1

$$Precision = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(FP)}$$

Figure 9: Precision [3]

$$Recall = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negative(FN)}$$

Figure 10: Recall [3]

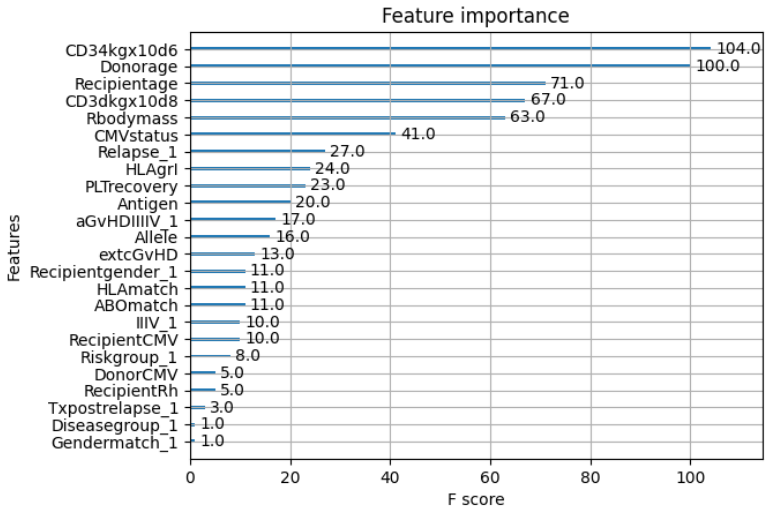

Figure 11: Feature Importances from XGBoost-1

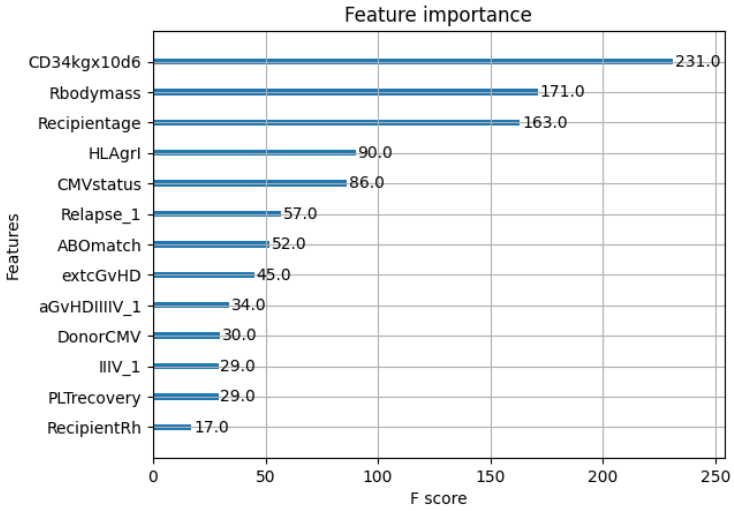

Figure 12: Feature Importances from XGBoost-2

Decision Trees

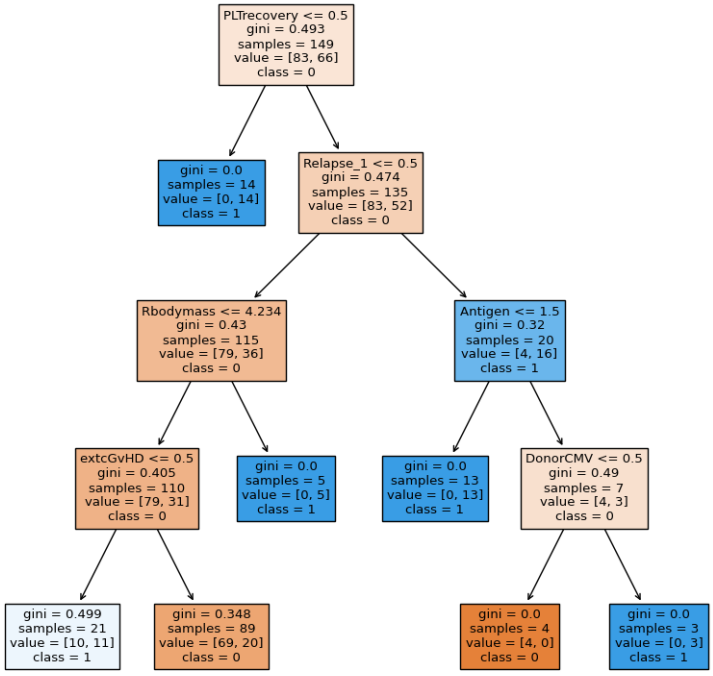

Figure 13: Decision tree-1

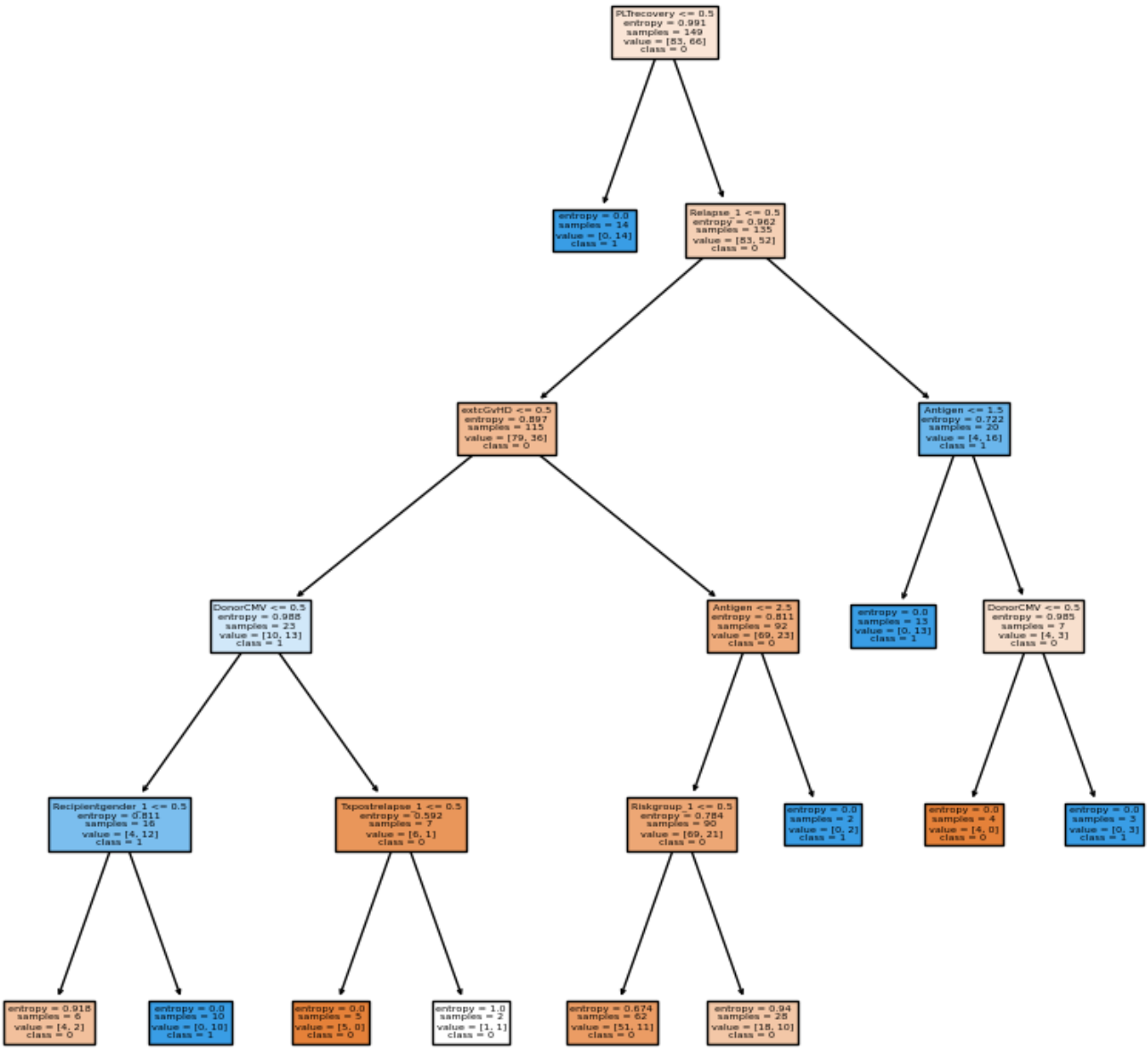

Figure 14: Decision tree-2

Learning Curves

For Logistic Regression 1:

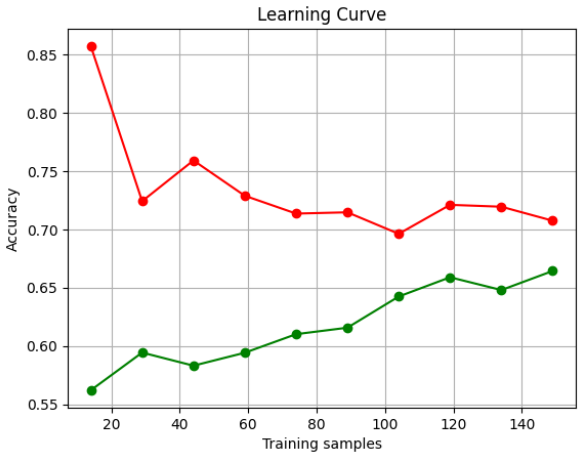

Figure 15: Logistic Regression-1

For Logistic Regression 2:

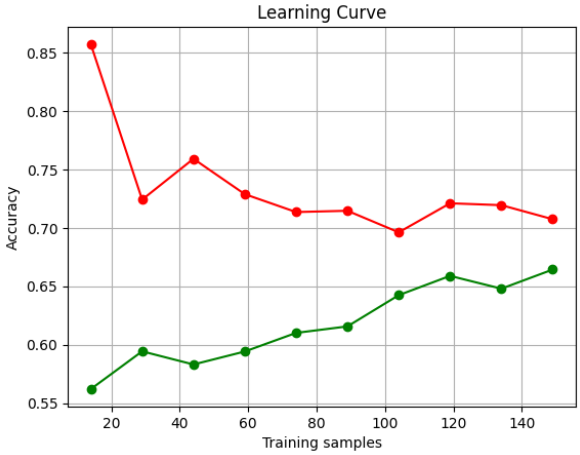

Figure 16: Logistic Regression-2

For Logistic Regression 3:

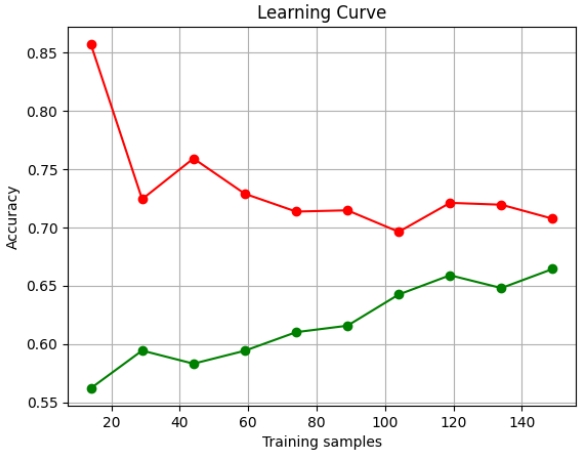

Figure 17: Logistic Regression-3

For Decision Tree 1:

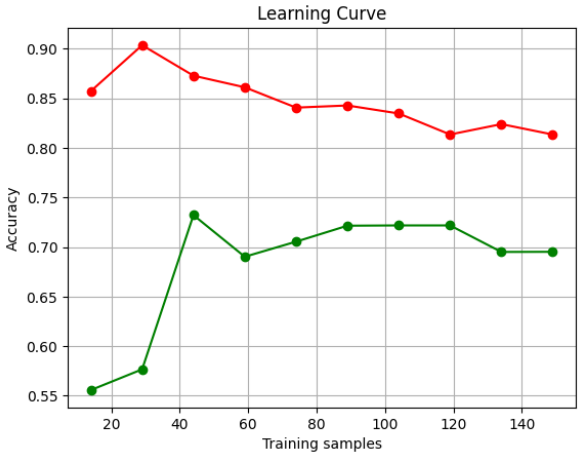

Figure 18: Decision Tree-1

For Decision Tree 2:

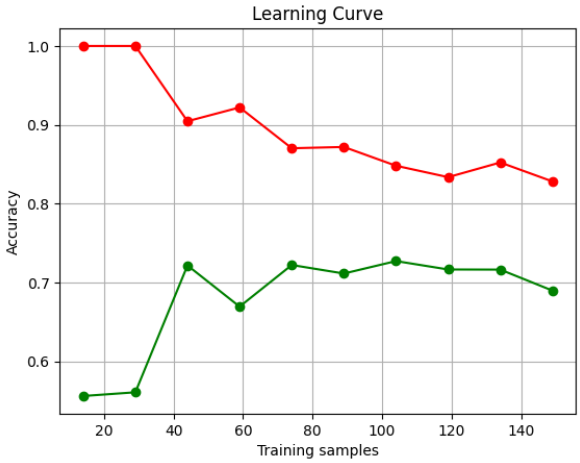

Figure 19: Decision Tree-2

For SVM 1:

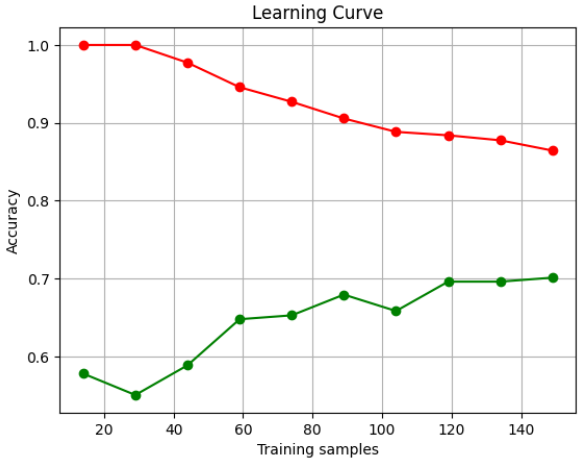

Figure 20: SVM-1

For SVM 2:

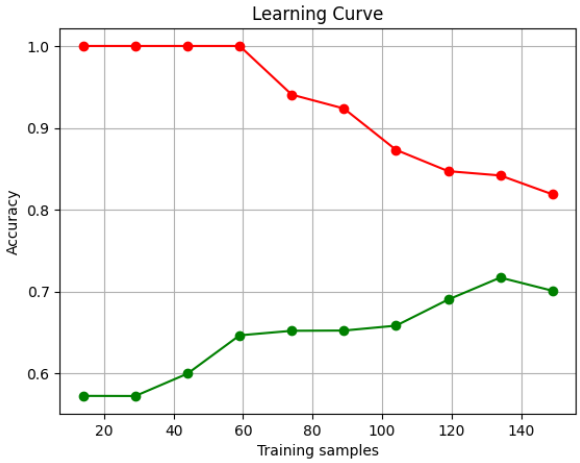

Figure 21: SVM-2

For XGBOOST 1:

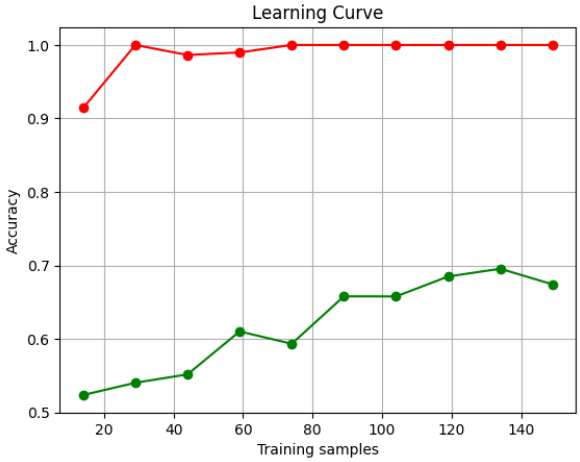

Figure 22: XGBoost-1

For XGBOOST 2:

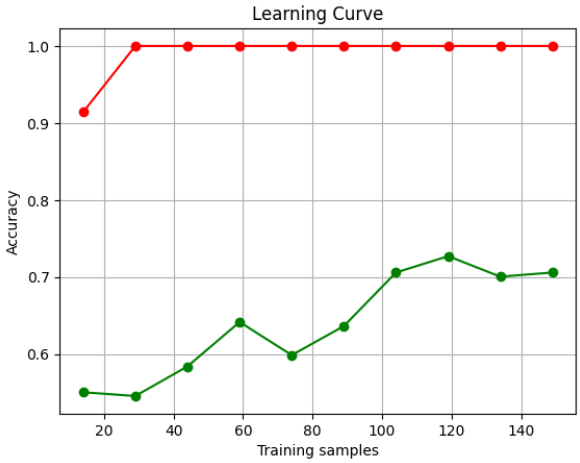

Figure 23: XGBoost-2

Linear Regression Plots

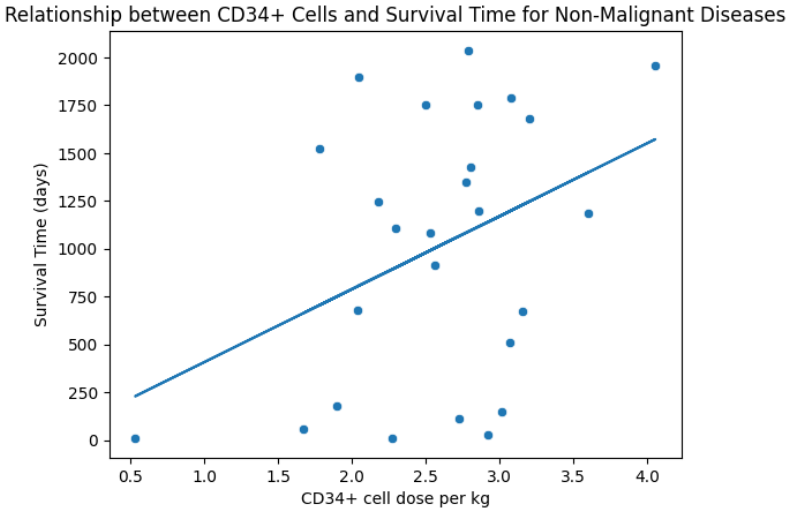

Figure 24: Relationship between CD34 cells and survival time for non-malignant diseases

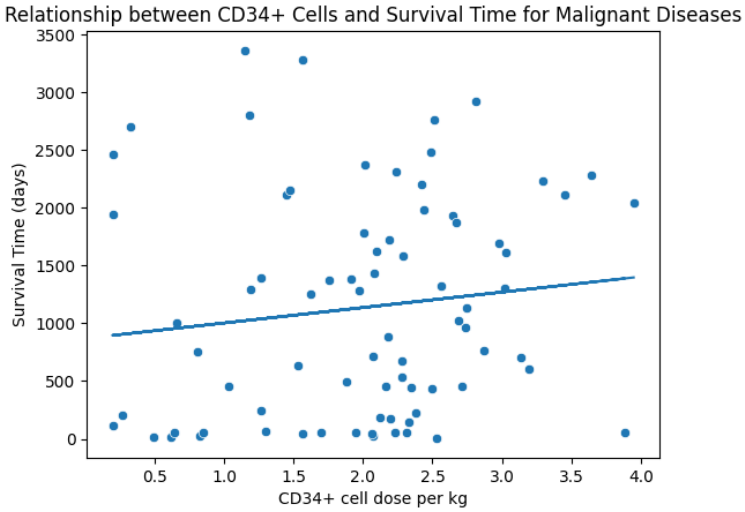

Figure 25: Relationship between CD34 cells and survival time for malignant diseases

8.1.2 Python Code for Modelling

For the modelling code with outputs please refer to this link (https://colab.research.google.com/drive/16B33sIlgQHFATpQFsoGI_fY_f-AwxoOX?usp=sharing). The pre-processing is at this link (https://github.com/Hyshubham2504/Bone-Marrow-Transplant/blob/main/Bone%20marrow%20transplant.ipynb).

```
# -*- coding: utf-8 -*-
"""STAT537.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/16B33sIlgQHFATpQFsoGI_fY_f-AwxoOX
"""

!pip install xgboost

import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, mean_squared_error
import statsmodels.api as sm
from scipy import stats
import statsmodels.stats.proportion as proportion
import mlxtend.feature_selection
import statsmodels.api as sm
```

```
from statsmodels.formula.api import ols
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report
from sklearn.svm import SVC
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import learning_curve
from mlxtend.plotting import plot_learning_curves

data = pd.read_csv("/content/drive/MyDrive/Colab␣
   Notebooks/Final_Bone_Marrow_Bucketting.csv") # reading the
   processed csv file into a dataframe

data.head() # glimpse of the data

"""## Preprocessing for Modelling"""

data.info() # to check for missing values and data types

# we noticed that categorical variables have at most 7 unique values
   all in range 0-6
# iterating over the columns to transform column datatypes
for col in data.columns:
    max_val = data[col].value_counts().keys().max()
    unique_values = data[col].nunique()  # Count the number of unique
       values in the column
    if (max_val <= 6) and (unique_values <= 7):
        data[col] = data[col].astype('category')  # Convert the
           column to categorical data type

print(data.dtypes)

# Binary Outcome
# distribution of the target variable
sns.countplot(data=data, x='survival_status')
plt.title('Distribution␣of␣Target␣Variable-1')
plt.show()

# For distb.
sns.histplot(data['survival_time'])
plt.xlabel('Survival␣Time')
plt.ylabel('Frequency')
plt.title('Histogram␣of␣Survival␣Time')
plt.show()

# creating X and y datasets with the features and target attriutes
X = data.drop(['survival_status', 'survival_time'], axis=1)
y_status = data['survival_status']
y_time = data['survival_time']
```

```
X.shape # printing the dimensions

y_status.shape

y_time.shape

count_status_1 = np.sum(data['survival_status'] == 1) # number of
    samples where survival status is 1 or alive
total_subjects = len(data['survival_status']) # total number of
    observations
proportion_status_1 = count_status_1 / total_subjects
print("Proportion of subjects with status = 1:", proportion_status_1)

se = np.sqrt(proportion_status_1 * (1 - proportion_status_1) /
    total_subjects)
print("Standard Error (SE):", se)

ci = proportion.proportion_confint(count_status_1, total_subjects,
    alpha=0.05, method='normal')
print("Confidence Interval (CI):", ci)

data['survival_status'].value_counts() # counts for each class of the
    target

X.shape

y_status.shape

X_train, X_test, y_time_train, y_time_test = train_test_split(X,
    y_time, test_size=0.2, random_state=0) # train-test splitting the
    dataset for the two functions

print(X_train.shape)
print(X_test.shape)

X_status_train, X_status_test, y_status_train, y_status_test =
    train_test_split(X, y_status, test_size=0.2, random_state=0)

# Checking Multicollinearity
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True)
plt.title('Correlation Matrix')
plt.show()

"""## Survival status"""

def learn_curve(model):
    '''
      To draw learning curves
    '''
    plt.figure() # creating the figure to output
    plt.grid()
```

```
    plt.title("Learning Curve")
    plt.xlabel("Training samples")
    plt.ylabel("Accuracy")

    ts = np.linspace(0.1, 1.0, 10) # range of training sizes

    # computing the learning curve and mean rain and test scores
    train_sizes, train_scores, test_scores = learning_curve(model, X,
       y_status, train_sizes=ts)
    train_scores_mean = np.mean(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)

    # plotting the scores and the sample sizes for tain and test
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Validation score")
    plt.show()

def featureselection(model, X_status_train1, y_status_train1):
  '''
    To select features from SequentialFeatureSelector for each model
  '''
  # backward eleimination for all features and the given model is fit
     and suitable features are outputted
  selector = SequentialFeatureSelector(model, direction='backward',
     scoring='accuracy', cv=10, n_features_to_select='auto', tol=None)
  selector.fit(X_status_train1, y_status_train1)
  selected_features = X_status_train1.columns[selector.get_support()]

  print("\nSelected features after backward elimination:")
  print(list(selected_features))
  return selected_features

"""### Logistic Regression"""

def log_metrics(result, X_testing, y_testing):

  # Classification table
  y_status_pred = result.predict(sm.add_constant(X_testing)) #
     constant for the intercept
  y_status_pred_class = (y_status_pred > 0.5).astype(int) # threshold
     for correct classification is 0.5

  conf_mat = pd.crosstab(y_testing, y_status_pred_class,
     rownames=['Actual'], colnames=['Predicted'])
  print('Confusion Matrix:')
  print(conf_mat)

  print("\n",classification_report(y_testing, y_status_pred_class))

def logreg(X_training, X_testing, y_training, y_testing):
```

```
    logit_model = sm.Logit(y_training, sm.add_constant(X_training)) #
        logistic regression model
    result = logit_model.fit(maxiter=10000)
    print(result.summary())

    # testing training
    print("For␣training␣data:")
    log_metrics(result, X_training, y_training)
    print('*'*50)

    # testing the model
    print("For␣testing␣data:")
    log_metrics(result, X_testing, y_testing)
    print('*'*50)

    # calculating the deviance (Likelihood Ratio Test)
    null_model = sm.Logit(y_testing,
        sm.add_constant(np.ones_like(y_testing)))
    null_result = null_model.fit()

    deviance = -2 * (null_result.llf - result.llf)
    p_value = 1 - stats.chi2.cdf(deviance, result.df_model)

    print('Deviance␣(Likelihood␣Ratio␣Test):')
    print('Deviance:', deviance)
    print('p-value:', p_value)
    print('*'*100)

    # Wald's test
    walds_test = result.wald_test_terms(scalar=False)
    print('\nWald\'s␣Test:')
    print(walds_test)

    learn_curve(LogisticRegression(max_iter=10000, C=0.1))

    return result

result = logreg(X_status_train, X_status_test, y_status_train,
    y_status_test)
print(result)

# selection of signigicant features based on p-value
p_values = result.pvalues[1:]  # to exclude the intercept term
selected_features1 = X_status_train.columns[p_values < 0.05]

print("Selected␣features␣based␣on␣p-values:")
print(list(selected_features1))

# the X sets are subsetted to work with the selected features only
subset_X_train = X_status_train.loc[:, list(selected_features1)]
subset_X_test = X_status_test.loc[:, list(selected_features1)]
```

```
result = logreg(subset_X_train, subset_X_test, y_status_train,
    y_status_test)
print(result)

# selecting purely from the sequnetial feature selector
log_model = LogisticRegression(max_iter=10000, C=0.1)
selected_features = featureselection(log_model, X_status_train,
    y_status_train)

subset_X_train = X_status_train.loc[:, list(selected_features)]
subset_X_test = X_status_test.loc[:, list(selected_features)]
result = logreg(subset_X_train, subset_X_test, y_status_train,
    y_status_test)
print(result)

"""### Decision Tree"""

# parameter grid for the grid search
params_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 3, 4, 5],
    'min_samples_split': [2, 5, 10, 20, 25],
    'min_samples_leaf': [1, 2, 4, 6, 8, 10]
}

dt = DecisionTreeClassifier()
grid_search = GridSearchCV(dt, params_grid, cv=10)
grid_search.fit(X_status_train, y_status_train)

best_params = grid_search.best_params_
print(best_params)

def decision_tree(params, X_training, X_testing, y_training,
    y_testing, flag):

  # Fitting the decision tree
  dt = DecisionTreeClassifier(**params)
  dt.fit(X_training, y_training)

  # predictions on the testing data
  y_pred1 = dt.predict(X_training)
  # classification report
  print("For␣training␣data:")
  print(classification_report(y_training, y_pred1))
  print('*'*100)

  # predictions on the testing data
  y_pred2 = dt.predict(X_testing)
  # classification report
  print("For␣testing␣data:")
  print(classification_report(y_testing, y_pred2))
  print('*'*100)
```

```
  # decision tree visualization
  plt.figure(figsize=(10, 10))
  plot_tree(dt, filled=True, feature_names=X_training.columns,
      class_names=['0', '1'])
  plt.show()

  learn_curve(dt)

  if flag==1:
    return featureselection(dt, X_training, y_training)

  return None

# Fitting the decision tree with the best parameters
selected_features = decision_tree(best_params, X_status_train,
   X_status_test, y_status_train, y_status_test, 1)

subset_X_train = X_status_train.loc[:, list(selected_features)]
subset_X_test = X_status_test.loc[:, list(selected_features)]

# parameter grid for the grid search
params_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 3, 4, 5],
    'min_samples_split': [2, 5, 10, 20, 25],
    'min_samples_leaf': [1, 2, 4, 6, 8, 10]
}

dt = DecisionTreeClassifier()
grid_search = GridSearchCV(dt, params_grid, cv=10)
grid_search.fit(subset_X_train, y_status_train)

best_params = grid_search.best_params_
print("Best parameters:")
print(best_params)
print('*'*150)

decision_tree(best_params, subset_X_train, subset_X_test,
   y_status_train, y_status_test, 0)

"""### SVM"""

# For grid search
params_grid = {'C': [0.1, 1, 10, 100, 1000],
               'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
               'kernel': ['linear', 'rbf', 'sigmoid']
}

grid_search = GridSearchCV(SVC(), params_grid, cv=10)
grid_search.fit(X_status_train, y_status_train)
best_params = grid_search.best_params_
```

```
print(best_params)

def SVM(params, X_training, X_testing, y_training, y_testing, flag):

  svm_classifier = SVC(**params) # instance of SVM classifier
  svm_classifier.fit(X_training, y_training)  # Fitting the SVM
     classifier to the training data

  # predictions on the testing data
  y_pred1 = svm_classifier.predict(X_training)
  # classification report
  print("For training data:")
  print(classification_report(y_training, y_pred1))
  print('*'*100)

  # predictions on the testing data
  y_pred2 = svm_classifier.predict(X_testing)
  # classification report
  print("For testing data:")
  print(classification_report(y_testing, y_pred2))
  print('*'*100)

  learn_curve(svm_classifier)

  if flag==1:
    return featureselection(svm_classifier, X_training, y_training)

  return None

# Fitting the SVM with the best parameters
selected_features = SVM(best_params, X_status_train, X_status_test,
   y_status_train, y_status_test,1)

subset_X_train = X_status_train.loc[:, list(selected_features)]
subset_X_test = X_status_test.loc[:, list(selected_features)]

# parameter grid for the grid search
params_grid = {'C': [0.1, 1, 10, 100, 1000],
               'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
               'kernel': ['linear', 'rbf', 'sigmoid']
}

grid_search = GridSearchCV(SVC(), params_grid, cv=10)
grid_search.fit(subset_X_train, y_status_train)
best_params = grid_search.best_params_
print("Best parameters:")
print(best_params)
print('*'*150)

SVM(best_params, subset_X_train, subset_X_test, y_status_train,
   y_status_test,0)
```

```
"""### XGBoost"""

# Define the hyperparameter grid
params_grid = {
    'max_depth': range(2,20),
    'learning_rate': [0.1, 0.01, 0.001],
    'subsample': [0.3, 0.5, 0.7, 0.8, 0.9, 1]
}

grid_search = GridSearchCV(XGBClassifier(objective=
   "binary:logistic", enable_categorical='True'), params_grid, cv=10,
   scoring='accuracy')
grid_search.fit(X_status_train, y_status_train)
best_params = grid_search.best_params_
print("Best parameters:")
print(best_params)

def XGB(params, X_training, X_testing, y_training, y_testing, flag):

  xgbc = XGBClassifier(**params, enable_categorical='True',
     objective= "binary:logistic") # instance of SVM classifier
  xgbc.fit(X_training, y_training)  # Fitting the SVM classifier to
     the training data

  # predictions on the testing data
  y_pred1 = xgbc.predict(X_training)
  # classification report
  print("For training data:")
  print(classification_report(y_training, y_pred1))
  print("RMSE= ", np.sqrt(mean_squared_error(y_training, y_pred1)))
  print('*'*100)

  # predictions on the testing data
  y_pred2 = xgbc.predict(X_testing)
  # classification report
  print("For testing data:")
  print(classification_report(y_testing, y_pred2))
  print("RMSE= ", np.sqrt(mean_squared_error(y_testing, y_pred2)))
  print('*'*100)

  # Plot feature importance
  xgb.plot_importance(xgbc)
  plt.show()

  learn_curve(xgbc)

  if flag==1:
    return featureselection(xgbc, X_training, y_training)

  return None

# Fitting the SVM with the best parameters
```

```
selected_features = XGB(best_params, X_status_train, X_status_test,
    y_status_train, y_status_test,1)

subset_X_train = X_status_train.loc[:, list(selected_features)]
subset_X_test = X_status_test.loc[:, list(selected_features)]

# parameter grid for the grid search
params_grid = {
    'max_depth': [2, 3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'subsample': [0.5, 0.7, 1]
}

grid_search = GridSearchCV(XGBClassifier(enable_categorical='True',
    objective= "binary:logistic"), params_grid, cv=10)
grid_search.fit(subset_X_train, y_status_train)
best_params = grid_search.best_params_
print("Best parameters:")
print(best_params)
print('*'*150)

XGB(best_params, subset_X_train, subset_X_test, y_status_train,
    y_status_test,0)

"""## Survival Time"""

# Fitting the entire linear regression model for reference
model1 = sm.OLS(y_time_train, sm.add_constant(X_train)).fit()
print(model1.summary())

data_subset = data[(data['aGvHDIIIIV_1'] == 1) & (data['extcGvHD'] ==
    1) & (data['Relapse_1'] == 0)] # subset of data where the
    undesirable events affecting patients' quality of life are none

# plotting the relation between the CD34+ cells with survival time
    for non-malignant diseases
x = data_subset[data_subset['Diseasegroup_1'] == 0]['CD34kgx10d6']
y = data_subset[data_subset['Diseasegroup_1'] == 0]['survival_time']
a, b = np.polyfit(x,y,1) # linear regression on the data points, to
    get the slope and the y-intercept

# data points are added to the plot
sns.scatterplot(x=x, y=y)

# find line of best fit
plt.plot(x, a*x+b)
plt.title('Relationship between CD34+ Cells and Survival Time for
    Non-Malignant Diseases')
plt.xlabel('CD34+ cell dose per kg')
plt.ylabel('Survival Time (days)')
plt.show()
```

```
# plotting the relation between the CD34+ cells with survival time
    for malignant diseases
x = data_subset[data_subset['Diseasegroup_1'] == 1]['CD34kgx10d6']
y = data_subset[data_subset['Diseasegroup_1'] == 1]['survival_time']
a, b = np.polyfit(x,y,1) # linear regression on the data points, to
    get the slope and the y-intercept

# data points are added to the plot
sns.scatterplot(x=x, y=y)

# find line of best fit
plt.plot(x, a*x+b)
plt.title('Relationship␣between␣CD34+␣Cells␣and␣Survival␣Time␣for␣
    Malignant␣Diseases')
plt.xlabel('CD34+␣cell␣dose␣per␣kg')
plt.ylabel('Survival␣Time␣(days)')
plt.show()
```