ML@NOVA DATAHOUSE

Survival Time in Multiple Myeloma Patients

DATAHOUSE - TEAM IDENTIFICATION

Name 1: **Guilherme Antunes**
Number 1: 70231

Name 2: **João Cristóvão**
Number 2: 70569

Name 3: **Ricardo Rodrigues**
Number 3: 72054

Final Score: 2.73313
Leaderboard private ranking: 32

CONTENTS

TASK 1

SETTING THE BASELINE

1.1 Data preparation and validation pipeline

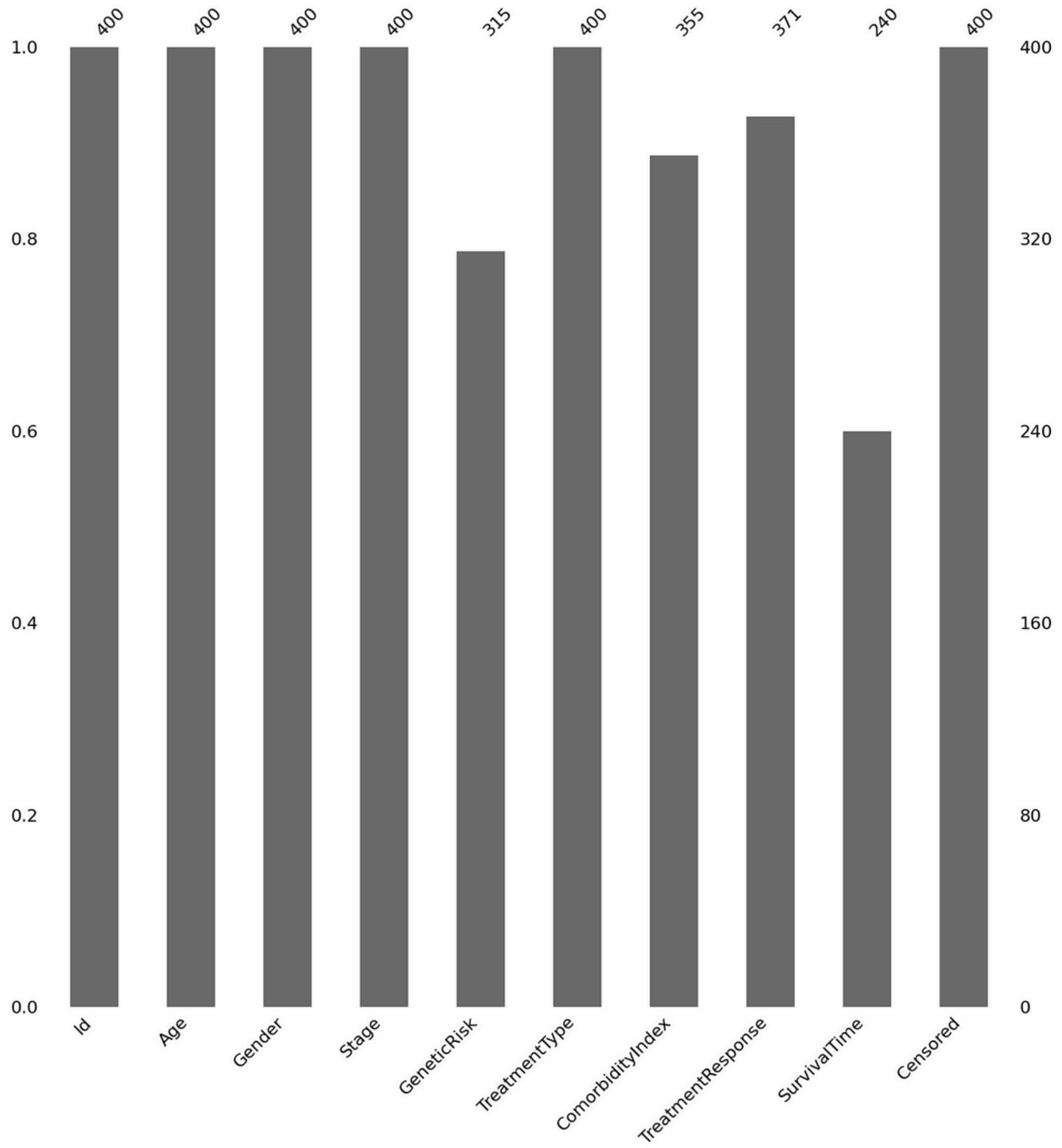
1.2 Learn the baseline model

1.3 Learn with the cMSE

WHAT WAS DONE IN TASK 1.1 - **BAR PLOT OF MISSING VALUES**

We began our analysis by generating the recommended by the professor, starting with the bar graph.

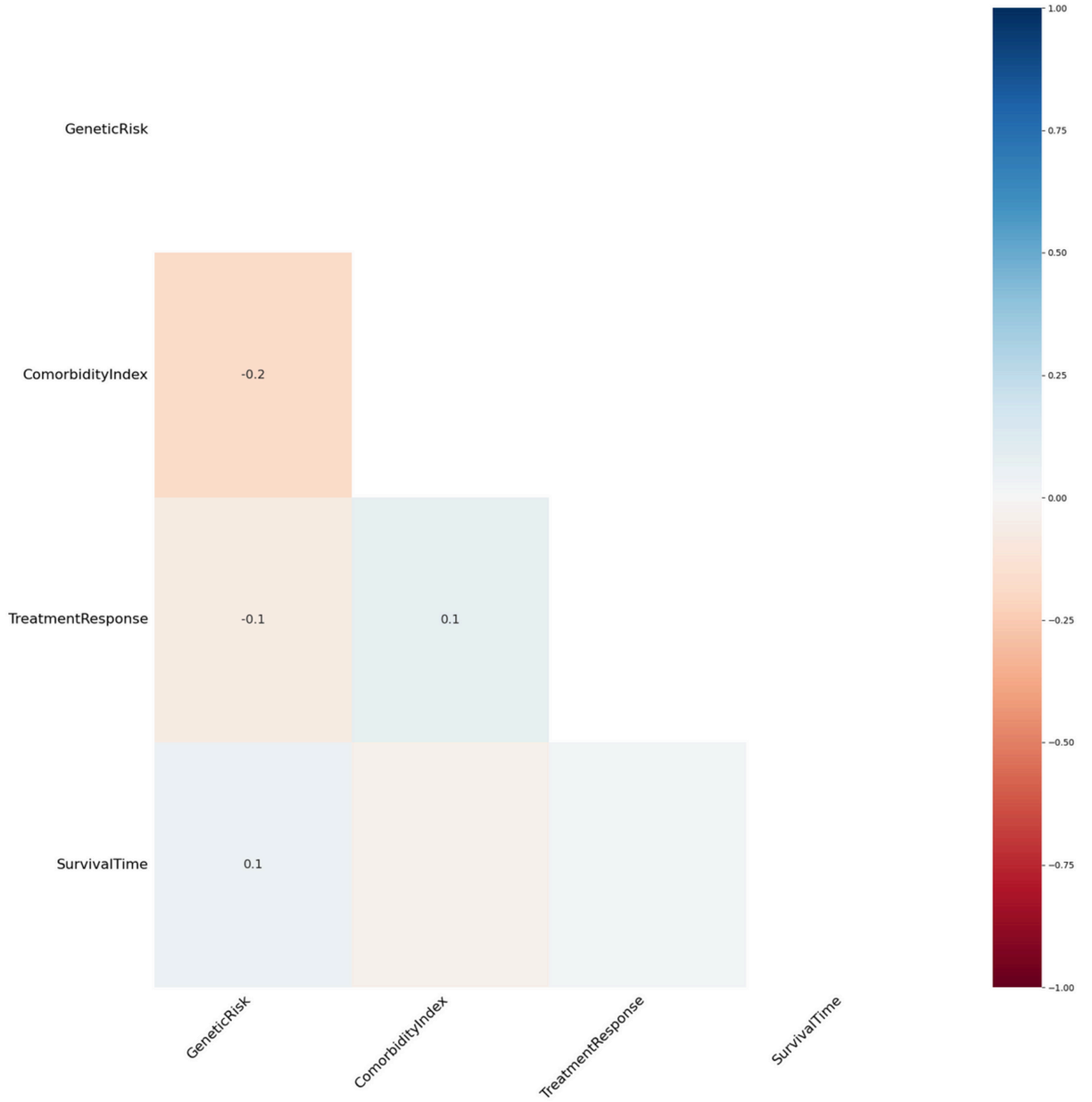
This graph show how many values are missing for each feature. For example, **SurvivalTime** is missing the most, followed by **GeneticRisk**. The fact that the SurvivalTime is missing so many times can be detrimental to training the model.



WHAT WAS DONE IN TASK 1.1 - **HEATMAP**

Next we analyzed the heatmap.

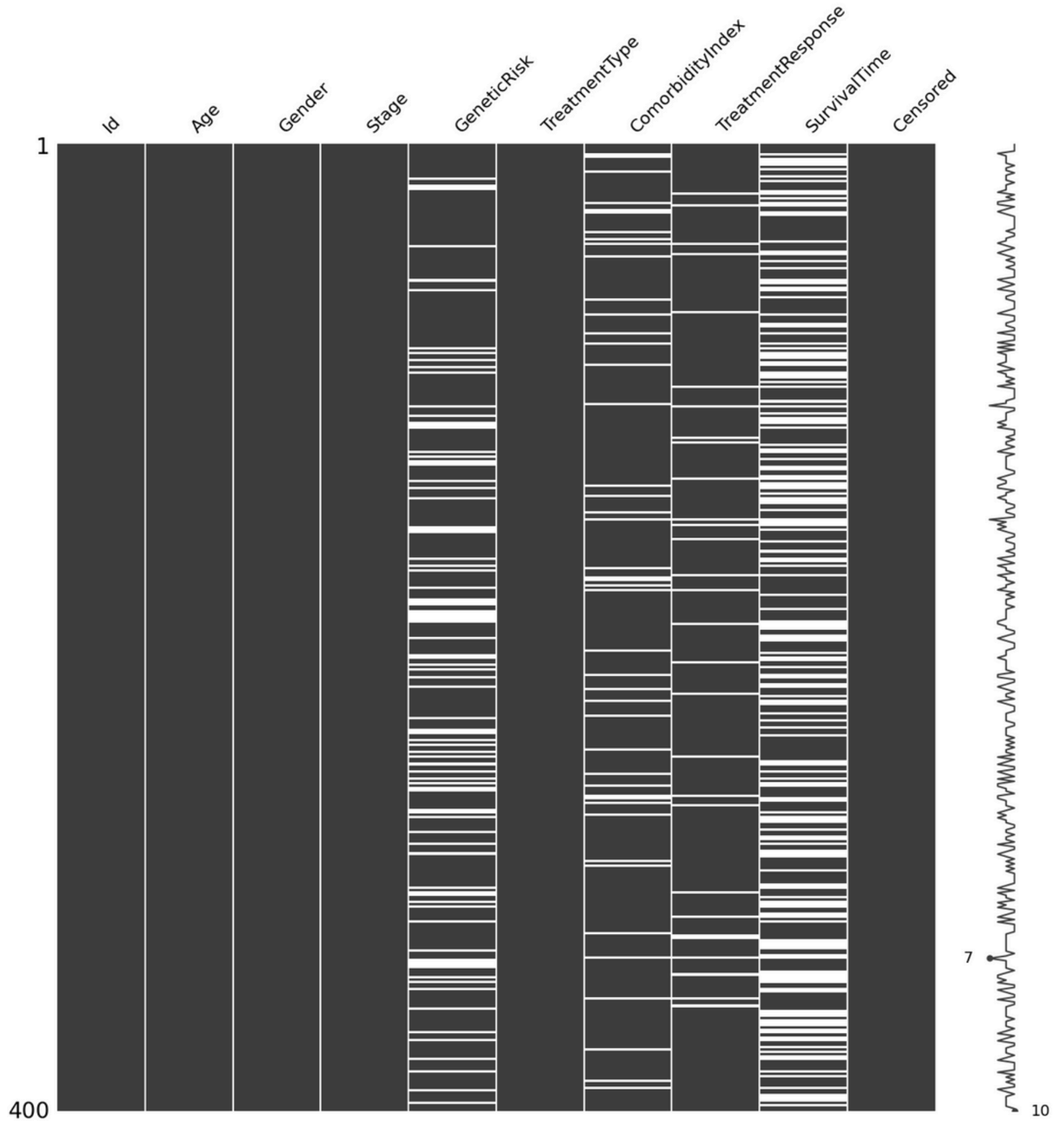
This heatmap shows the **correlation between the missing values** for each column. We can see that, the correlations in missing values are weak, when present.



WHAT WAS DONE IN TASK 1.1 - **MATRIX**

Next we analyzed the matrix.

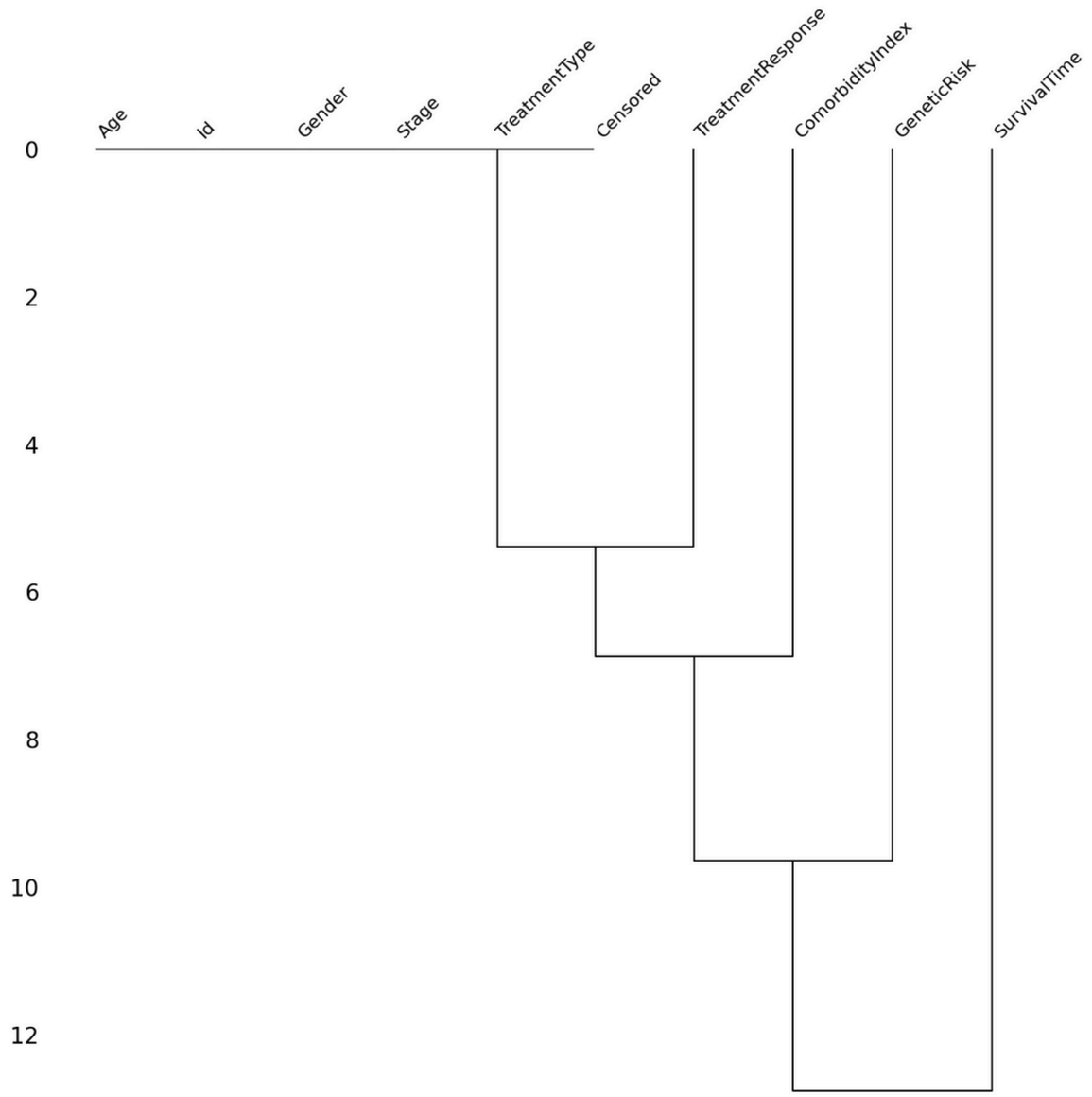
This heatmap shows the **correlation between the missing values** for each column. We can see that, the correlations in missing values are weak, when present.



WHAT WAS DONE IN TASK 1.1 - **DENDOGRAM**

Next we analyzed the dendogram.

The dendrogram shows **how similar the missing data patterns are**, and in this specific case it's visible that the patterns are quite similar.



WHAT WAS DONE IN TASK 1.1 - **DATA PREPARATION**

To prepare the dataset we started by **removing all censored datapoints**. The reason for this is that these data points don't have a value for the survival time, and we wanted to avoid training our model with incomplete information.

We also removed the data points that had null values, leaving us with a total of 109 data points, we then trained the model.

We for cross validation over a regular train, test, validation split, because our dataset is small, and cross validation allows us to use more data for training. We also saw a high degree of variance in our local results, and cross validation helps with evening things out.
The resource intensiveness of cross validation is not a problem due to the small size of the data set.

We used a standard **80/20 split** for the train and test data.

WHAT WAS DONE IN TASK 1.1 - **PAIRPLOT**

Due to the size of the pairplot graph, we deciced to move it to the next slide.

On the graph we can see that the features **Gender** and **GeneticRisk** have no influence on the target variable **SurvivalTime**.

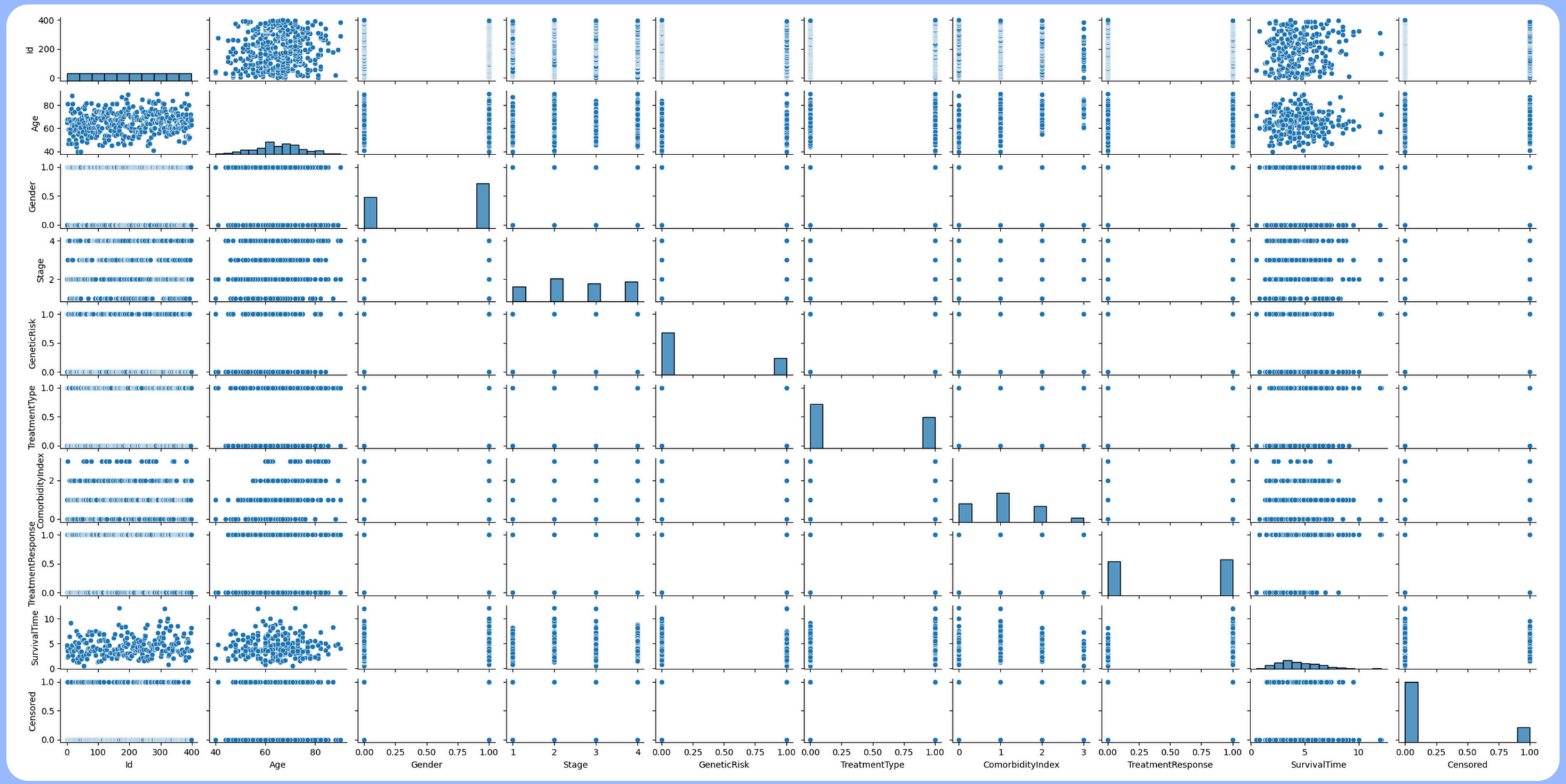
With all the features we obtain the following results:

- Cross-validated RMSE: **1.8534**
- Mean Squared Error (MSE): **2.4583**

After removing the Gender and GeneticRisk features we got the following results:

- Cross-validated RMSE: **1.7885**
- Mean Squared Error (MSE): **2.4239**

As shown by these results we see that by removing these two features the score improves.


Id
Age
Gender
Stage
GeneticRisk
TreatmentType
ComorbidityIndex
TreatmentResponse
SurvivalTime
Censored

WHAT WAS DONE IN TASK 1.2 - **BASELINE MODEL**

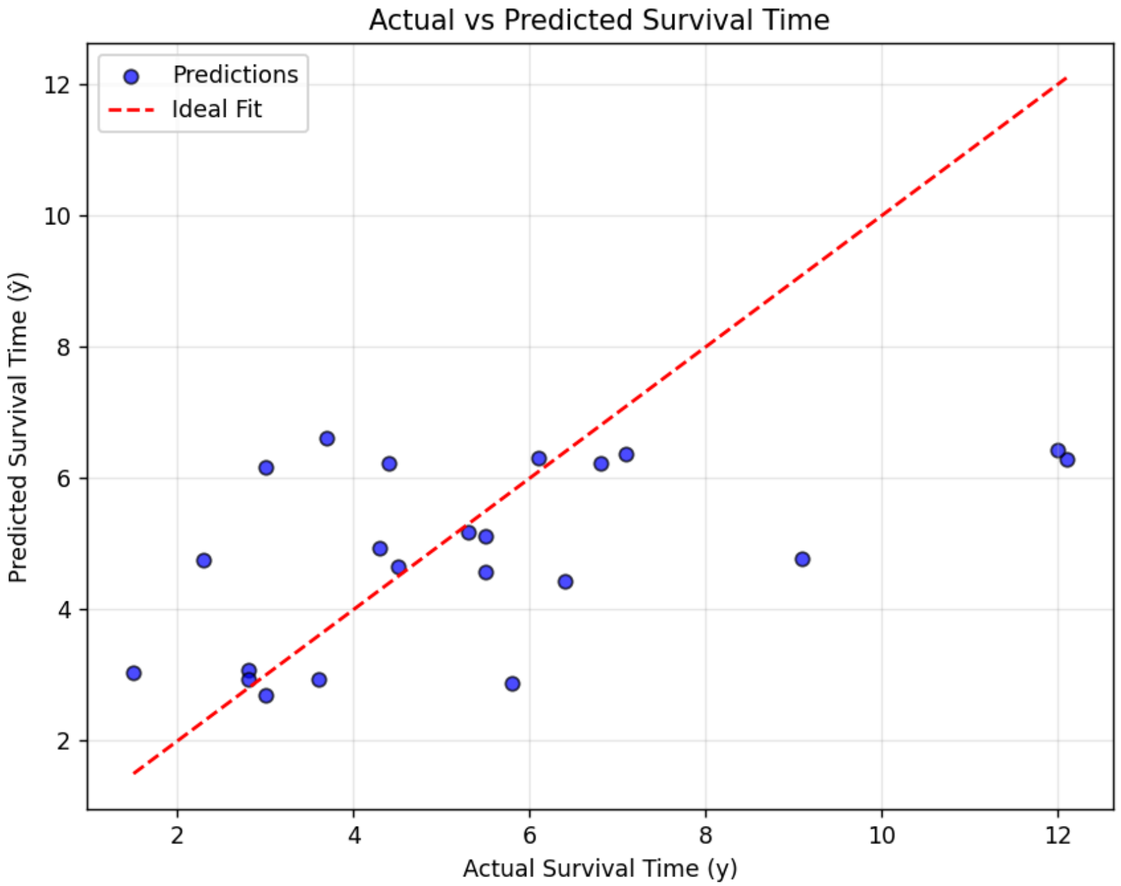
In this task we trained a baseline model for the problem.

For this model we used a **Standard Scaler** and a **Linear Regressor**. With this aproach we obtained a score of **2.77797** on Kaggle.

To assess how good this model is, we built the **y-y hat plot**, displayed on the side.

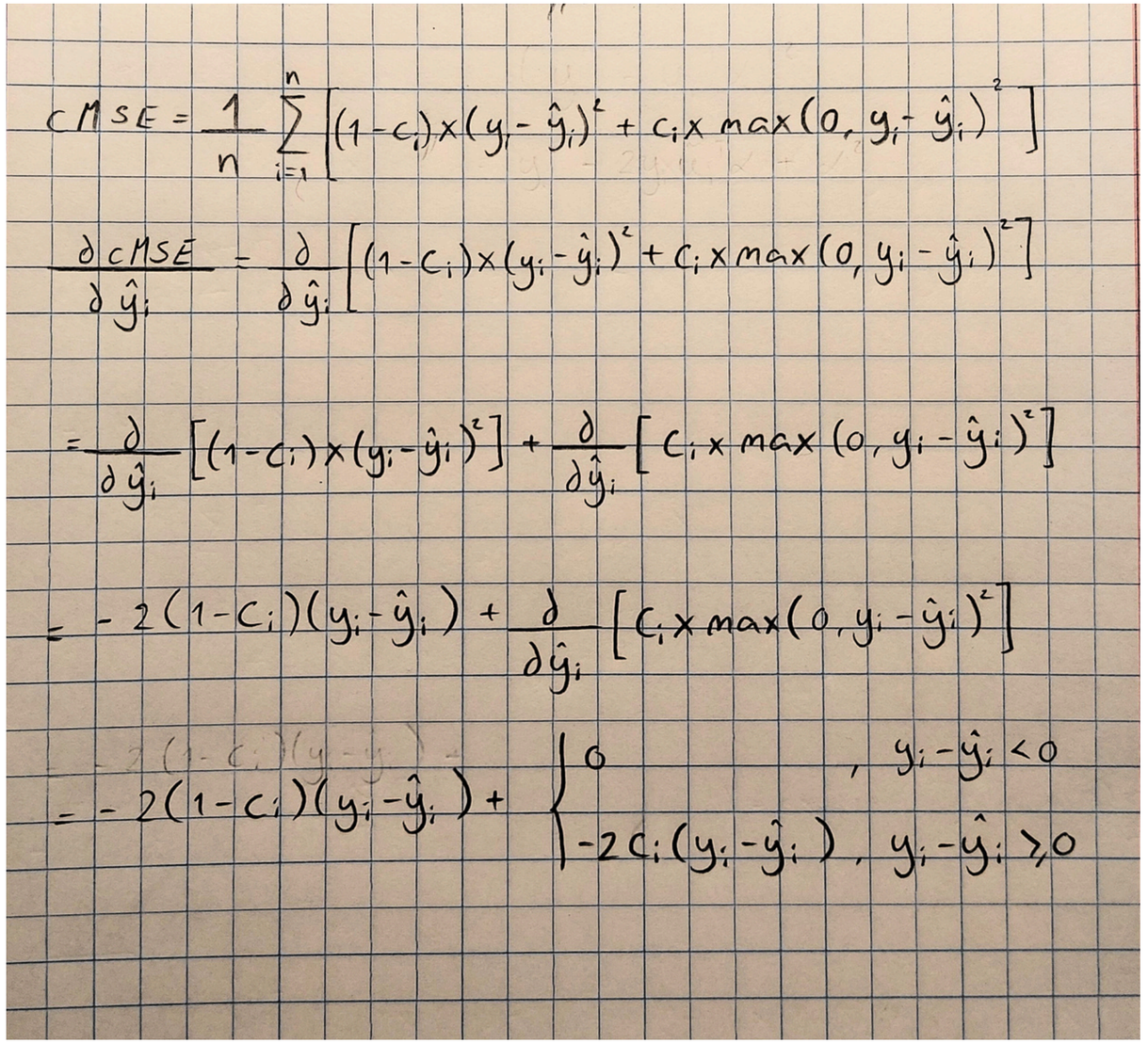
cMSE: **5.8756**
MSE: **2.4239**

Kaggle
Public score: **2.77797**
Private score: **3.21643**



WHAT WAS DONE IN TASK 1.3 - **CMSE DERIVATIVE COMPUTATION**

In this task we started by computing the **expression of the derivative** of the **cMSE Loss**.



WHAT WAS DONE IN TASK 1.3 - **GRADIENT DESCENT**

In this task we implemented **Gradient Descent**, and used it for our pipeline.

We tested both **Lasso** and **Ridge** regularizations with **different lambda values** to find the best model.

WHAT WAS DONE IN TASK 1.3 - **REGULARIZATION**

In this part we applied **Regularization** to our model with the **Gradient Descent.**

We tried both **Ridge** and **Lasso** regularization. In both cases we used a **learning rate of 0.01**, and tested **different lambda values** (1, 0.1 and 0.01). This resulted in 6 different cases:

- **Ridge Regularization with lambda equal to 1, 0.1 and 0.01.**
- **Lasso Regularization with lambda equal to 1, 0.1 and 0.01.**

For each case we ran the model 100 times, registering the best result of those 100 runs.

RIDGE REGULARIZATION RESULTS

Here are the results obtained for the model with **Gradient Descent** and **Ridge Regularization.**

Lambda = 0.01

Locally

Cross-validated RMSE: **2.0166**

Mean Squared Error: **1.1209**

Test cMSE Loss: **1.2564**

Kaggle

Test cMSE Loss: **3.03099**

Lambda = 0.1

Locally

Cross-validated RMSE: **2.0056**

Mean Squared Error: **1.1185**

Test cMSE Loss: **1.2512**

Kaggle

Test cMSE Loss: **3.01151**

Lambda = 1

Locally

Cross-validated RMSE: **1.9677**

Mean Squared Error: **1.4552**

Test cMSE Loss: **2.1175**

Kaggle

Test cMSE Loss: **3.26258**

RIDGE REGULARIZATION RESULTS ANALYSIS

Analysing the results obtained for the Ridge Regularization we see that the **best result** was obtained with **lambda equal to 0.1**. Locally, the model with Gradient Descent and Ridge Regularization with lambda equal to 0.1, scored a cMSE of **1.2512** and on Kaggle scored a cMSE of **3.01151**.

It is clear that there is a significant discrepancy between cMSE we estimated locally and the one computed by Kaggle. This can be caused by (tentar perceber a razão, talvez esteja relacionado com o facto de o kaggle usar apenas 53% da test data)

Lambda = 0.01

Locally
Test cMSE Loss: **1.2564**

Kaggle
Test cMSE Loss: **3.03099**

Lambda = 0.1

Locally
Test cMSE Loss: **1.2512**

Kaggle
Test cMSE Loss: **3.01151**

Lambda = 1

Locally
Test cMSE Loss: **2.1175**

Kaggle
Test cMSE Loss: **3.26258**

LASSO REGULARIZATION RESULTS

Here are the results obtained for the model with **Gradient Descent** and **Lasso Regularization.**

Lambda = 0.01

Locally

Cross-validated RMSE: **2.0281**

Mean Squared Error: **1.2360**

Test cMSE Loss: **1.5275**

Kaggle

Test cMSE Loss: **2.99546**

Lambda = 0.1

Locally

Cross-validated RMSE: **2.0193**

Mean Squared Error: **1.2367**

Test cMSE Loss: **1.5294**

Kaggle

Test cMSE Loss: **2.98203**

Lambda = 1

Locally

Cross-validated RMSE: **2.0224**

Mean Squared Error: **1.2011**

Test cMSE Loss: **1.4425**

Kaggle

Test cMSE Loss: **3.04954**

LASSO REGULARIZATION RESULTS ANALYSIS

Analysing the results obtained for the Lasso Regularization we see that the **best result** was obtained with **lambda equal to 0.1**. Locally, the model with Gradient Descent and Lasso Regularization with lambda equal to 0.1, scored a cMSE of **1.5294** and on Kaggle scored a cMSE of **2.98203**.

It is clear that there is a significant discrepancy between cMSE we estimated locally and the one computed by Kaggle. This can be caused by (tentar perceber a razão, talvez esteja relacionado com o facto de o kaggle usar apenas 53% da test data)

Lambda = 0.01

Locally
Test cMSE Loss: **1.5275**

Kaggle
Test cMSE Loss: **2.99546**

Lambda = 0.1

Locally
Test cMSE Loss: **1.5294**

Kaggle
Test cMSE Loss: **2.98203**

Lambda = 1

Locally
Test cMSE Loss: **1.4425**

Kaggle
Test cMSE Loss: **3.04954**

TASK 2
NONLINEAR MODELS

2.1 Development

2.2 Evaluation

WHAT WAS DONE IN TASK 2.1 - **DEVELOPMENT**

For this task we kept the dataset preparation from our baseline model and trained models with **Polynomial Features** with degrees ranging from 2 to 6, and also models with the **K neighbors regressor** with the k values = [3, 5, 7, 9, 11].

For each iteration we used cross validation on each pipeline and selected the best one based on the result. We then took the best pipeline, trained it with a random 80/20 split, and verified the cMSE. If the cMSE happened to be the current best we would save it, as well as the prediction on the test dataset, the degree or k value, and the MSE.

Below are our best local results after 1000 iterations:

Polynomial cMSE: **1.4977**
Polynomial MSE: **1.5331**
Polynomial Degree: **2**
kNN cMSE: **1.2228**
kNN MSE: **1.2851**
kNN K: **9**

WHAT WAS DONE IN TASK 2.1 - **DEVELOPMENT**

While training our polynomial models we saw that there was quite the extreme **overfitting** with **polynomial features with degrees higher than 2.**

Below are the results of two different iterations. We can see that for the same data split the **validation errors go up by a lot with higher degrees**. This makes it clear that with higher degrees, the models capture more relationships between features, which accentuates the overfitting.

Polynomial Model

Degree = **2**, Validation Error = **2.5993**
Degree = **3,** Validation Error = **28.0909**
Degree = **4,** Validation Error = **167321903.05**
Degree = **5,** Validation Error = **29218048.85**
Degree = **6,** Validation Error = **33730394.38**

Polynomial Model

Degree = **2**, Validation Error = **3.5993**
Degree = **3,** Validation Error = **12.8670**
Degree = **4,** Validation Error = **80403.64**
Degree = **5,** Validation Error = **31750.69**
Degree = **6,** Validation Error = **25024.49**

WHAT WAS DONE IN TASK 2.1 - **DEVELOPMENT**

For the knn regressor however we observer similar validation error values regardless of the number of neighbours, with a slight downwards tendency with higher k values.

The knn regressor shows much less overfiting than the linear regressor with polynomal features due to its non-parametric nature. Polynomial regression attempts to create a general model that fits all the data, making it more susceptible to noise and complex data.

KNN Model
K = **3**, Validation Error = **2.6267**
K = **5,** Validation Error = **2.3485**
K = **7,** Validation Error = **2.2587**
K = **9,** Validation Error = **2.2983**
K = **11,** Validation Error = **2.1749**

KNN Model
K = **3**, Validation Error = **2.7952**
K = **5,** Validation Error = **2.5800**
K = **7,** Validation Error = **2.4873**
K = **9,** Validation Error = **2.3280**
K = **11,** Validation Error = **2.3358**

WHAT WAS DONE IN TASK 2.2 - **EVALUATION**

After one thousand iterations we saved the predictions from the models with the lowest local cMSE and submitted them to kaggle.

The local results and kaggle scores are below:

✓	**Nonlinear-submission-poly-01.csv** Complete · Guilherme Antunes · 4d ago · Poly com degree 2, mil iterações, erro local = 1.49	**3.52439**	**3.04860**
✓	**Nonlinear-submission-knn-01.csv** Complete · Guilherme Antunes · 4d ago · KNN com k = 9, mil iterações, erro local = 1.22	**3.38363**	**3.28581**

```
Best Polynomial cMSE: 1.4977495966522845
Best Polynomial MSE: 1.5331486255373312
Best Polynomial Degree: 2
Best kNN cMSE: 1.2227957818930042
Best kNN MSE: 1.285199861160735
Best kNN K: 9
```

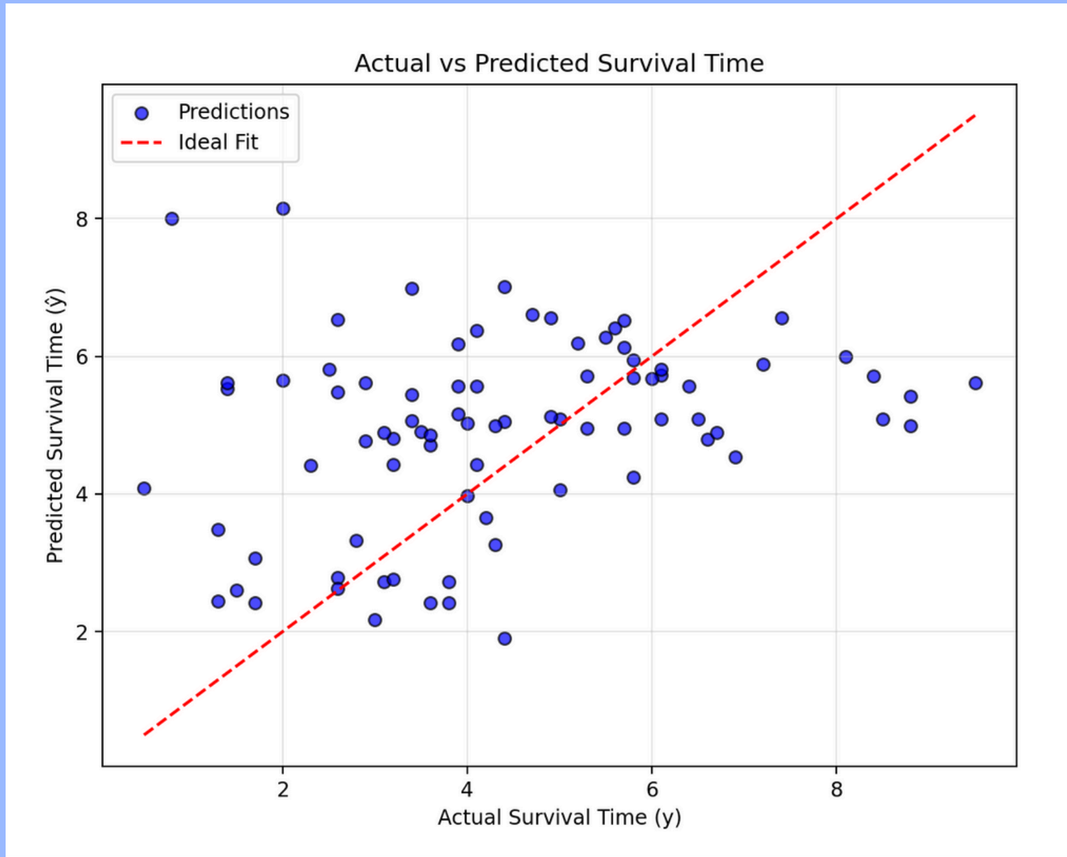
WHAT WAS DONE IN TASK 2.2 - **EVALUATION**

Our kaggle scores were more than double of our local scores, and slightly worse than our baseline mode, which was unexpected. We tried using a 53/47 split to mimic kaggle, and the cMSE was higher but our kaggle results were better.

Nonlinear-submission-poly-02.csv Complete · Guilherme Antunes · 3d ago · Polynomial features, degree = 2, cMSE local = 2.151, ...		**3.13423**	**2.95481**
Nonlinear-submission-knn-02.csv Complete · Guilherme Antunes · 3d ago · KNN regressor, k = 11, cMSE local = 1.768, 53/47 split		**3.28380**	**3.33178**

```
Best Polynomial cMSE: 2.1515901563272752
Best Polynomial MSE: 1.5880437462364485
Best Polynomial Degree: 2
Best kNN cMSE: 1.768345644701236
Best kNN MSE: 1.4498912363973033
Best kNN K: 11
```

Linear Regressor Polynomial Features



cMSE: **2.1515**

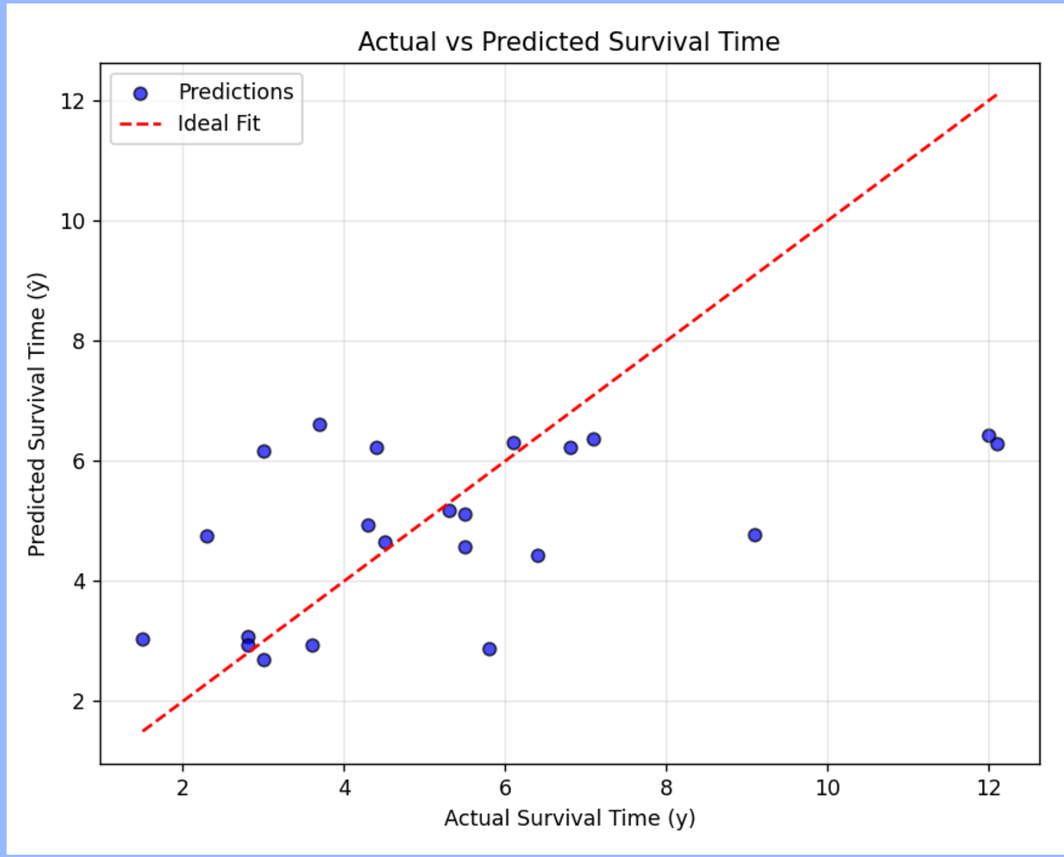
MSE: **1.5880**

Kaggle

Public score: **2.95481**

Private score: **3.13423**

Best Cross Validation Model



cMSE: **5.8756**

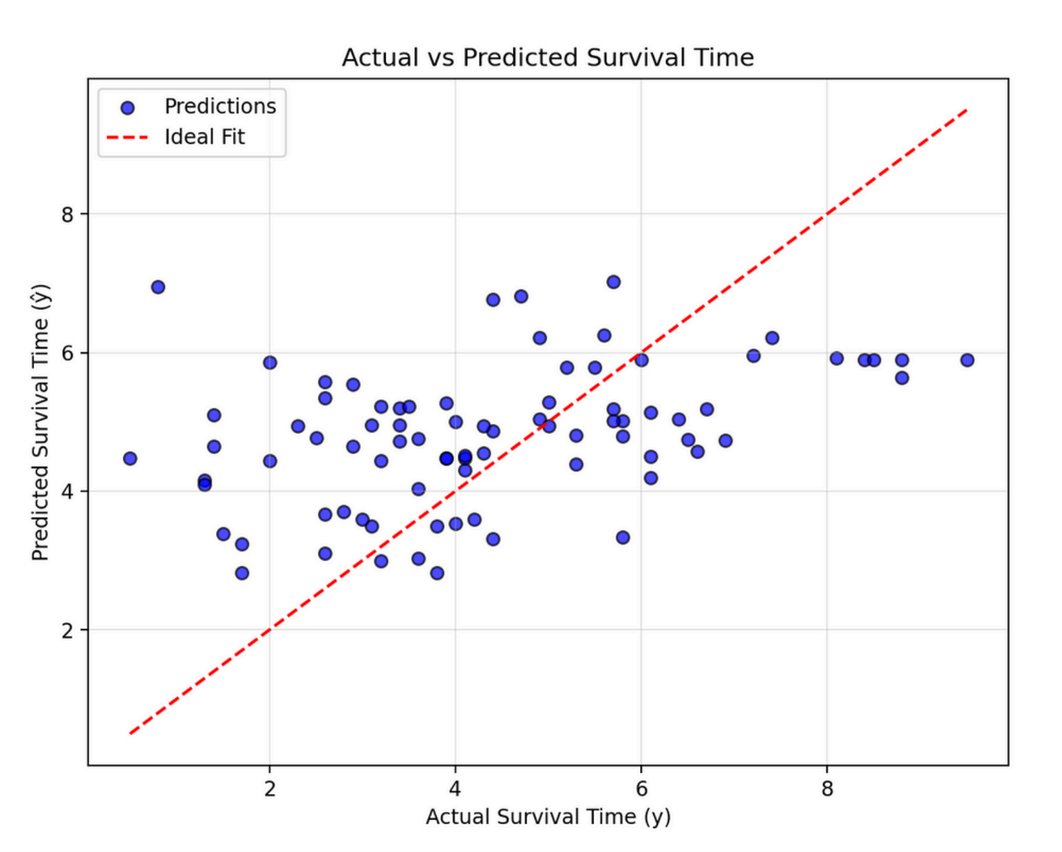
MSE: **2.4239**

Kaggle

Public score: **2.77797**

Private score: **3.21643**

Linear Regressor Polynomial Features



cMSE: **1.7683**

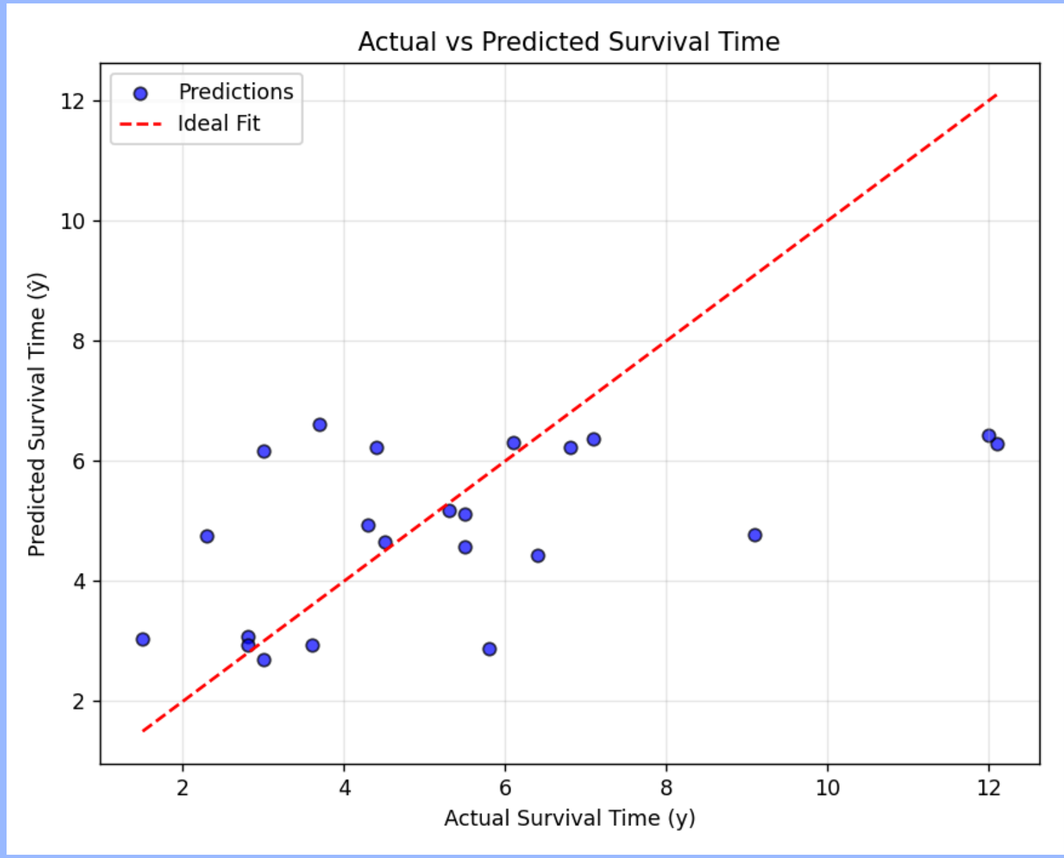
MSE: **1.4498**

Kaggle

Public score: **3.33178**

Private score: **3.28380**

Best Cross Validation Model



cMSE: **5.8756**

MSE: **2.4239**

Kaggle

Public score: **2.77797**

Private score: **3.21643**

WHAT WAS DONE IN TASK 2.2 - **EVALUATION**

The main difference between our baseline model and our models with polynomial features and knn regressor is that the baseline model, with a linear regressor, is worse at capturing relations between the features, and since it got the best score we believe that our non linear models are capturing irrelevant relations and are overfitting to the training data.

For reference our previous scores were: baseline – 2.77, polynomial features – 3.05, knn regressor – 3.26.

When comparing our private and public scores, we can see that for our first attempt, with an 80/20 split, the results got for both models, however it was more noticeable on our polynomial model.
For our second attempt with a 53/47 split, we can see that our polynomial model got a slightly worse result while our knn model actually improved a little bit.
This shows once again that the polynomial model is more prone to overfitting than the knn model, and that using less datapoints in the training data, also leads to less overfitting by weakening giving the model weaker relationships between features.

TASK 3
HANDLING MISSING DATA

3.1 Missing data imputation

3.2 Learn the baseline model

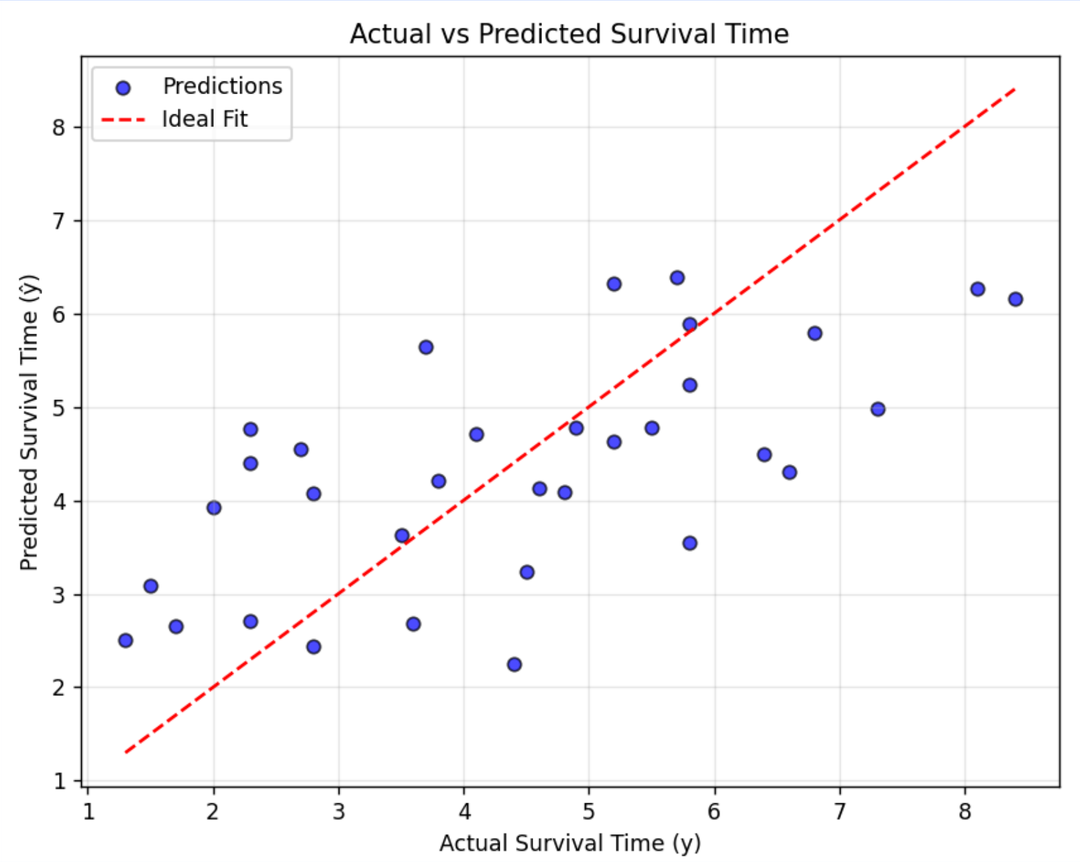
3.3 Evaluation

WHAT WAS DONE IN TASK 3.1/3.3 - **MISSING DATA IMPUTATION**

To handle missing data, various imputation strategies were applied to the dataset. The strategies tested included Mean, Median, Most Frequent, K-Nearest Neighbors (KNN), and Iterative Imputer. These imputation methods were applied to key features with missing values, such as GeneticRisk, ComorbidityIndex, and TreatmentResponse.
After imputation, a baseline regression model comprising a pipeline with StandardScaler and LinearRegression was trained and evaluated using cross-validation. Each strategy's performance was assessed using RMSE and cMSE metrics. The Median imputation strategy emerged as the best performer.

We identified an issue in our cross-validation selection process. Although we are confident in our chosen model for Task 3, it is important to highlight that this model did not achieve the lowest validation and training errors. In the following slide, we will compare the chosen model with the best-performing cross-validation model, for Task 3. Notably, the median imputation method yielded the best results and was used in our approach. Subsequently, we will also compare the performance of our model with the model from Task 1.2(Baseline Model) to provide a comprehensive analysis.

Chosen Model



cMSE: **2.0665**

MSE: **1.4375**

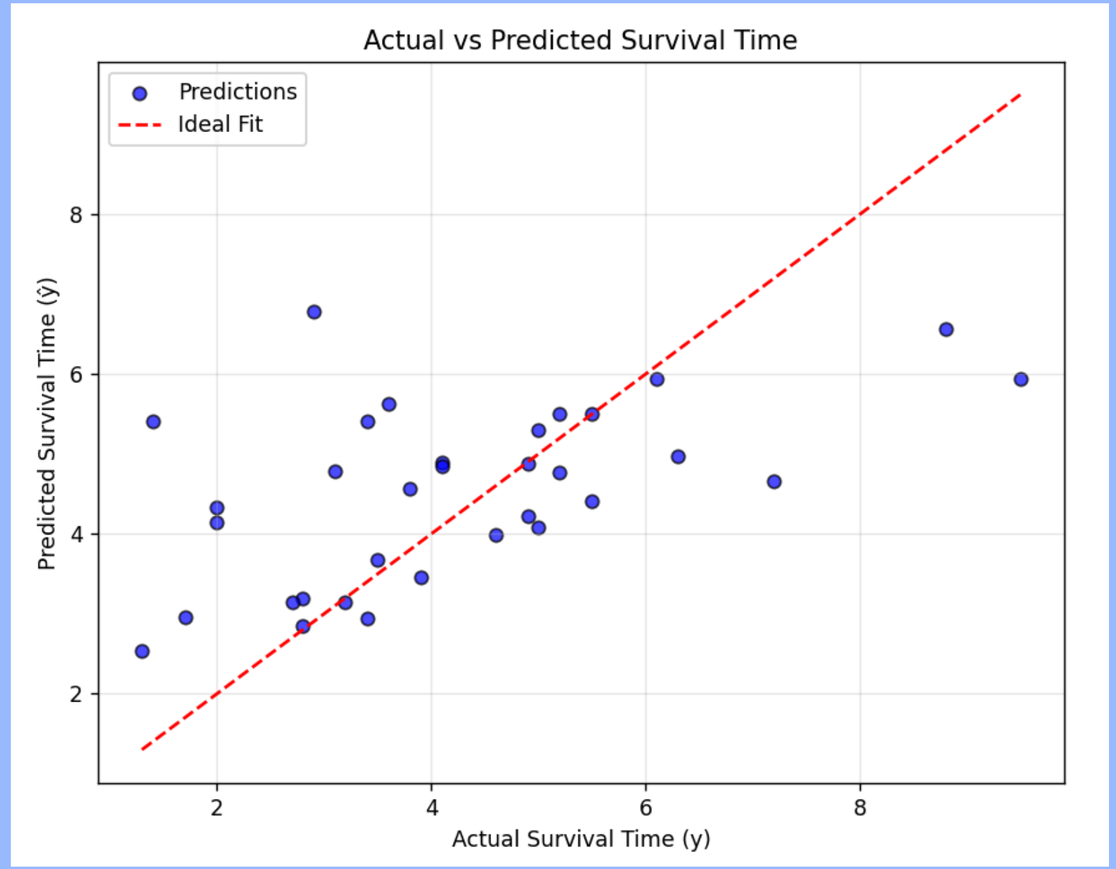
Train/Validation Error: **3.3091 ; 3.7520**

Kaggle

Public score: **2.57026**

Private score: **2.73313**

Best Cross Validation Model



cMSE: **2.6372**

MSE: **1.6240**

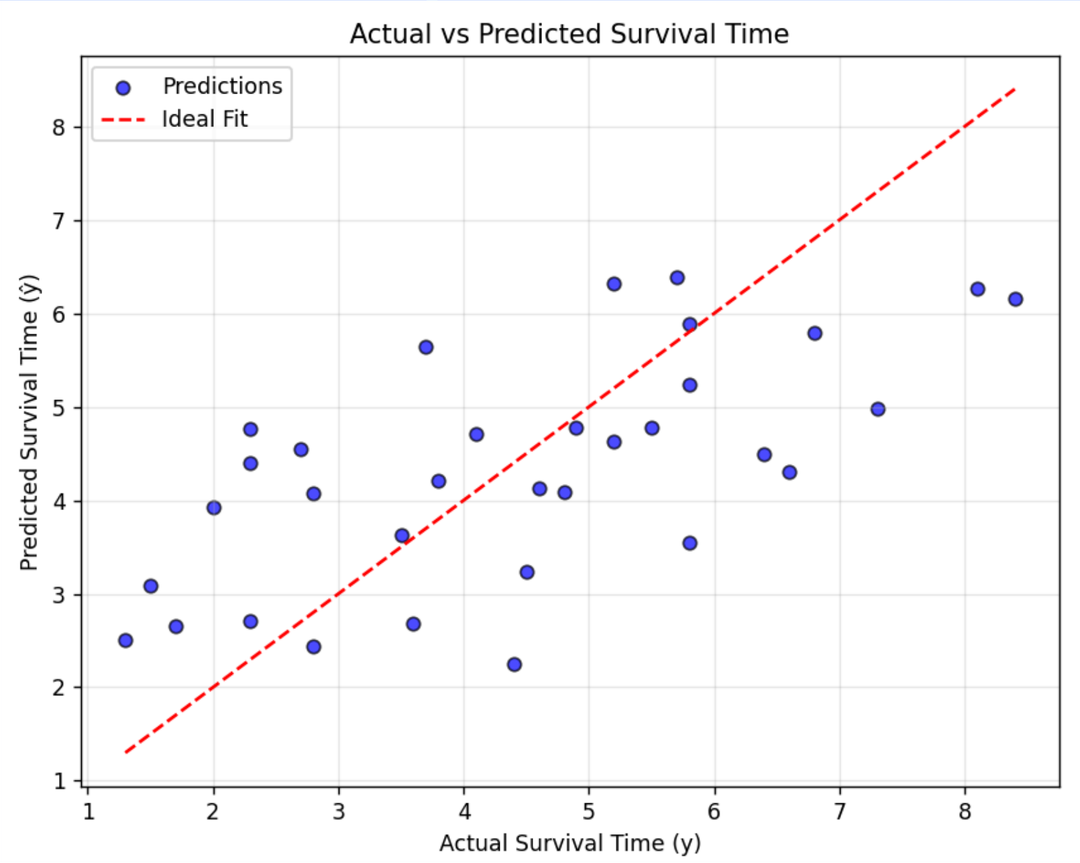
Train/Validation Error: **2.5629 ; 2.8883**

Kaggle

Public score: **3.11989**

Private score: **2.41523**

Median Imputation Model



cMSE: **2.0665**

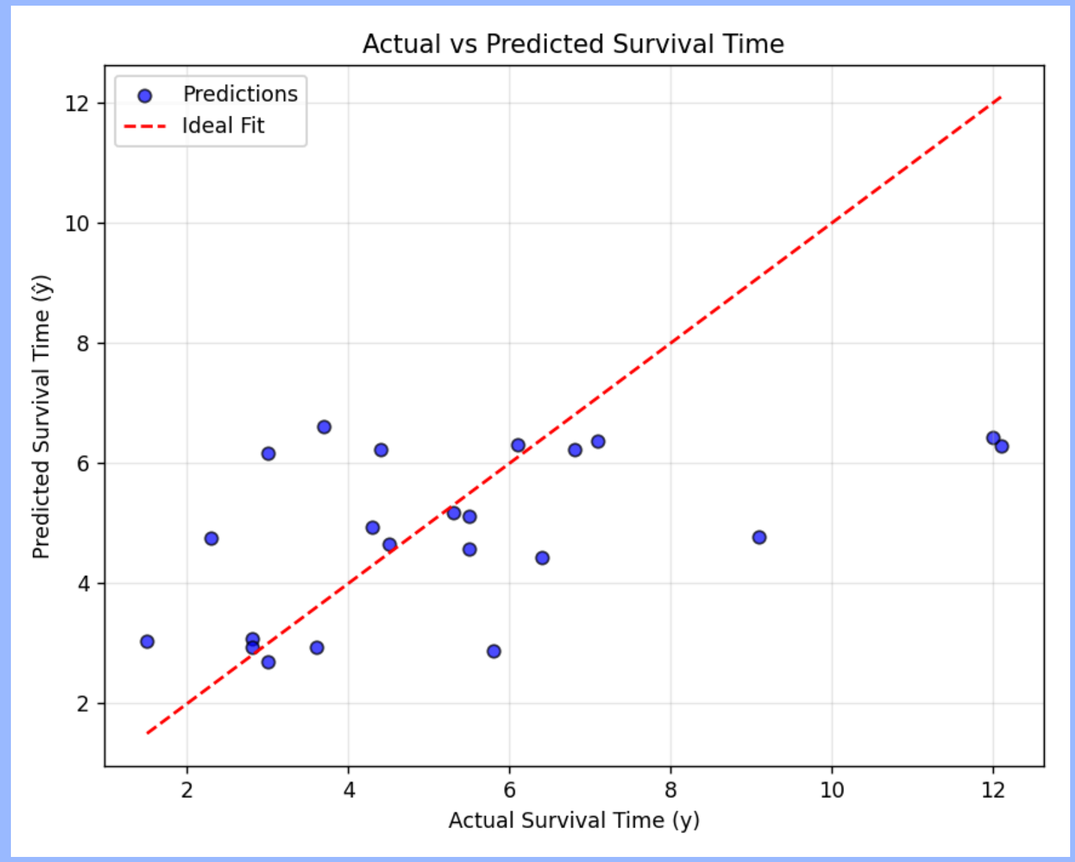
MSE: **1.4375**

Kaggle

Public score: **2.57026**

Private score: **2.73313**

Baseline Model



cMSE: **5.8756**

MSE: **2.4239**

Kaggle

Public score: **2.77797**

Private score: **3.21643**

WHAT WAS DONE IN TASK 3.2/3.3 - **MISSING DATA IMPUTATION**

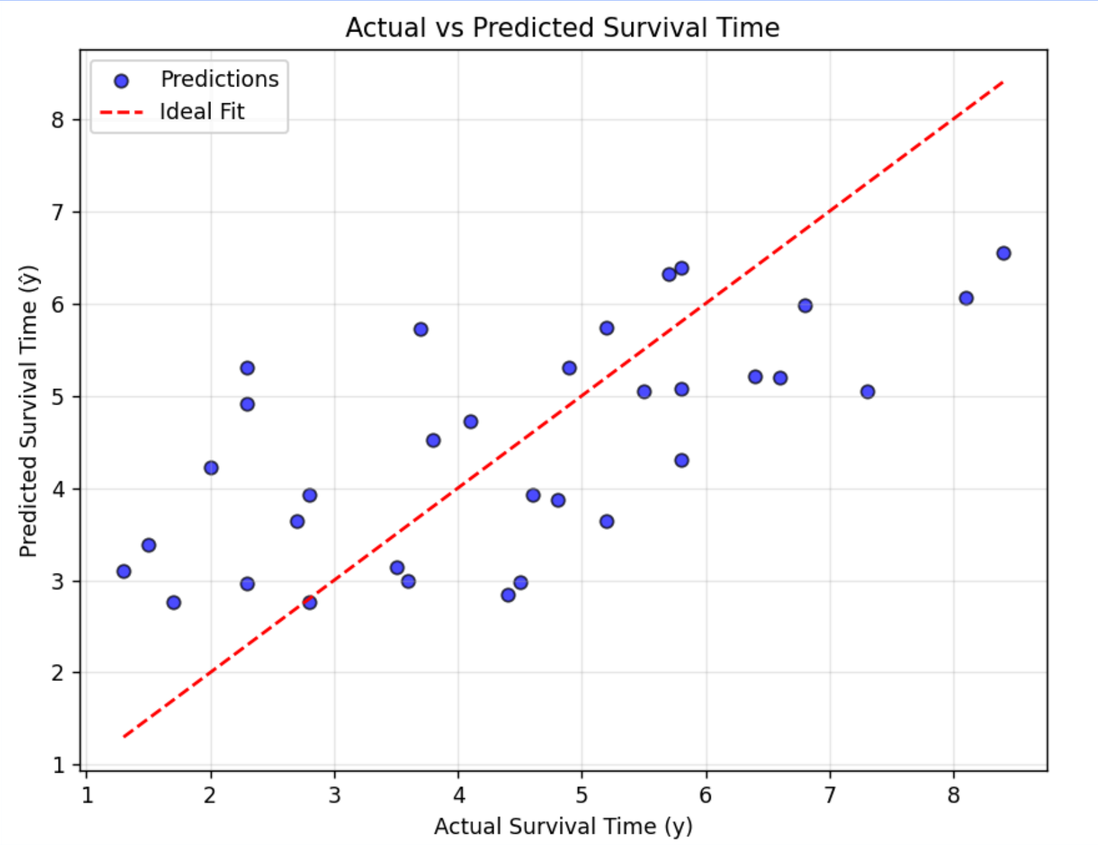
In Task 3.2, two models that can handle missing data natively were implemented: HistGradientBoostingRegressor and CatBoostRegressor. These models were chosen for their robust handling of incomplete datasets without requiring explicit imputation.

For HistGradientBoostingRegressor, optimal hyperparameters were selected to maximize its predictive performance. The model efficiently managed missing values by leveraging its inherent capabilities, avoiding the need for any preprocessing to address missing data.

The CatBoostRegressor was trained using its default handling of missing values, which involves special categorical and numerical processing to manage gaps in the dataset. The model was configured to utilize its built-in missing value treatment without requiring additional steps.

The performance of both models was evaluated using cross-validation and tested on the holdout dataset. Our local results, showed similar perfomances between the HistGradientBoostingRegressor and CatBoostRegressor, having achieved our best result in the former.

HistGradientBoostingRegressor

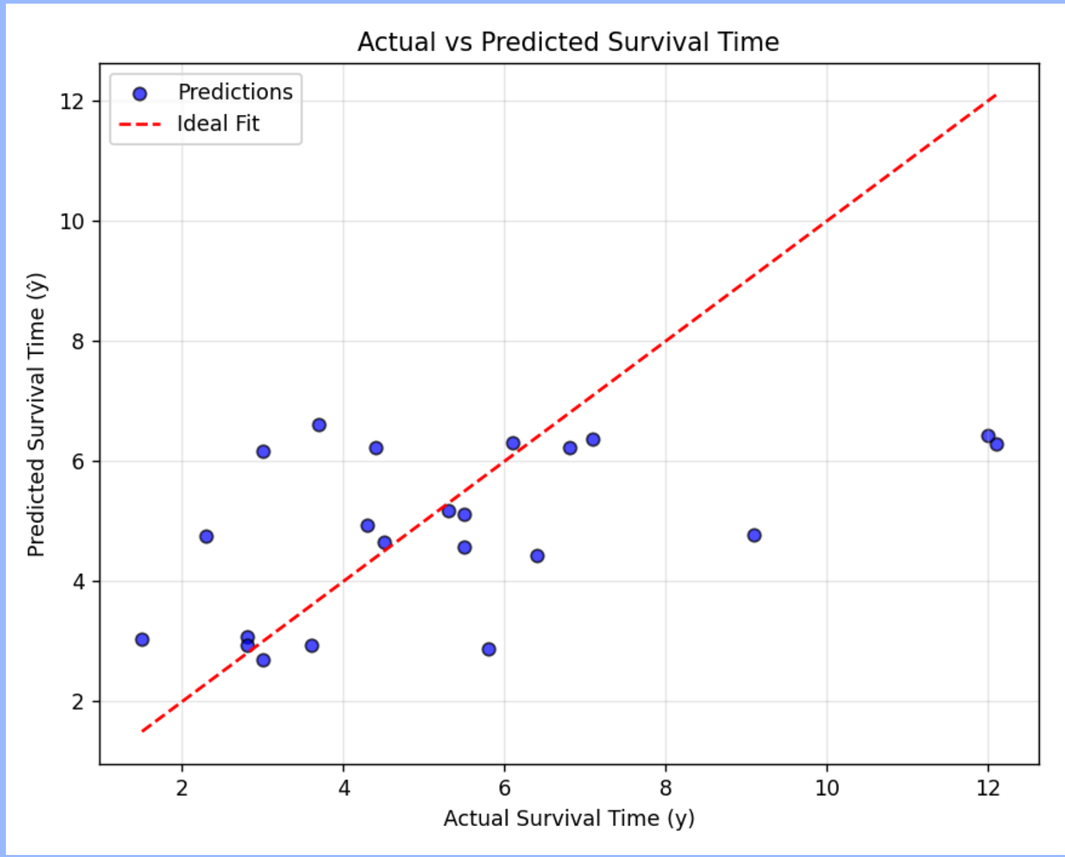


cMSE: 2.0042

MSE: 1.4157

Train/Validation Error: **3.2316 ; 3.7362**

Baseline Model



cMSE: **5.8756**

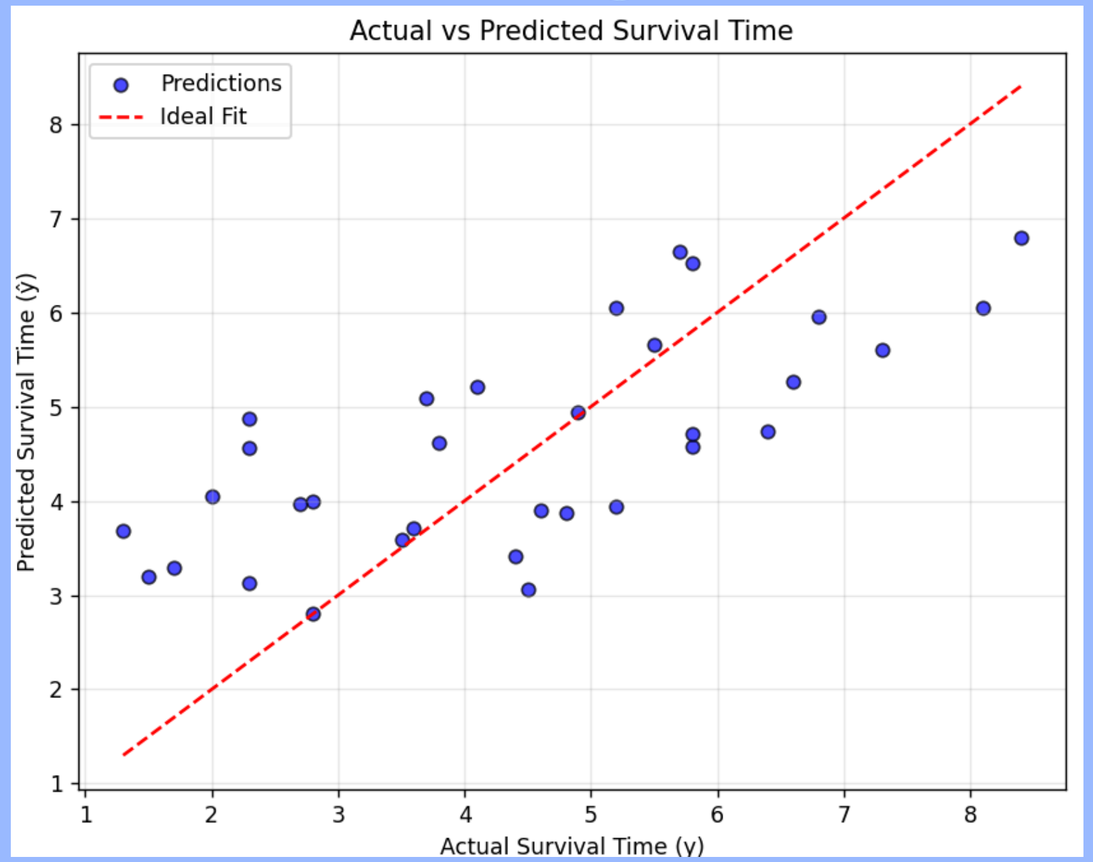
MSE: **2.4239**

Kaggle

Public score: **2.77797**

Private score: **3.21643**

CatBoostRegressor

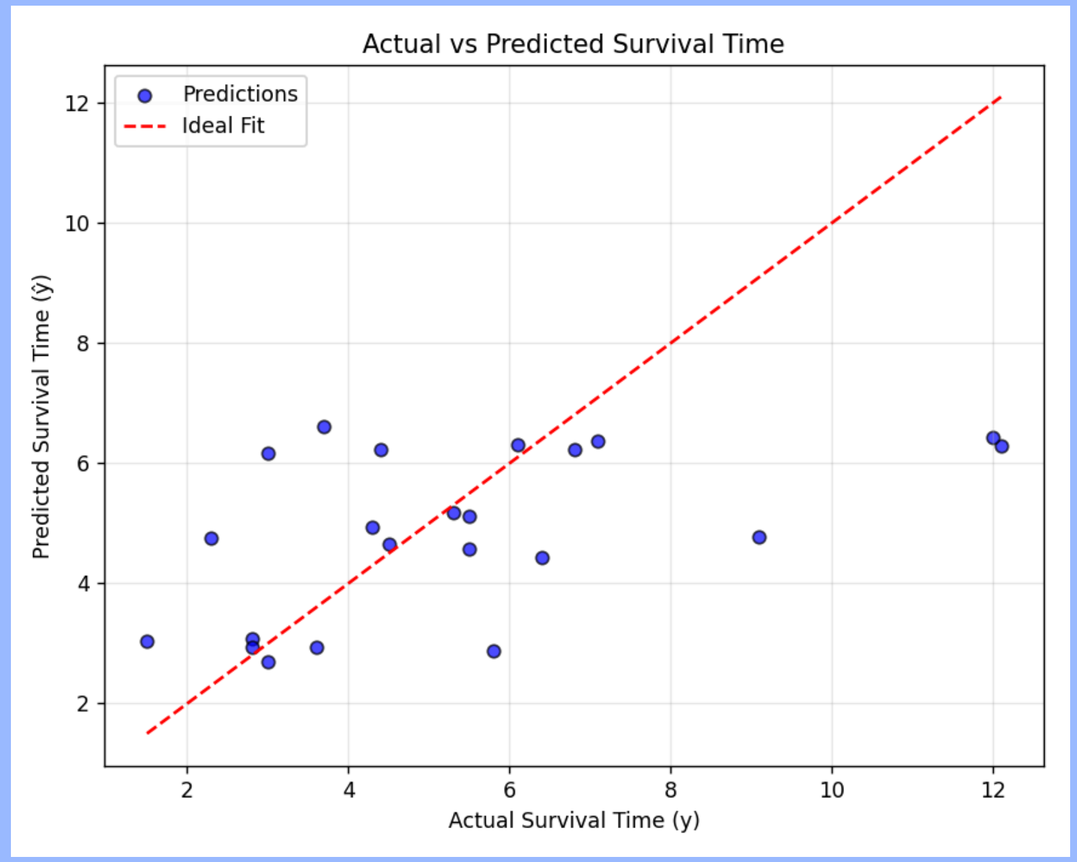


cMSE: 1.8270

MSE: 1.3517

Train/Validation Error: **2.2573 ; 3.8147**

Baseline Model



cMSE: **5.8756**

MSE: **2.4239**

Kaggle

Public score: **2.77797**

Private score: **3.21643**

TASK 4

SEMI-SUPERVISED LEARNING

4.1 Imputation

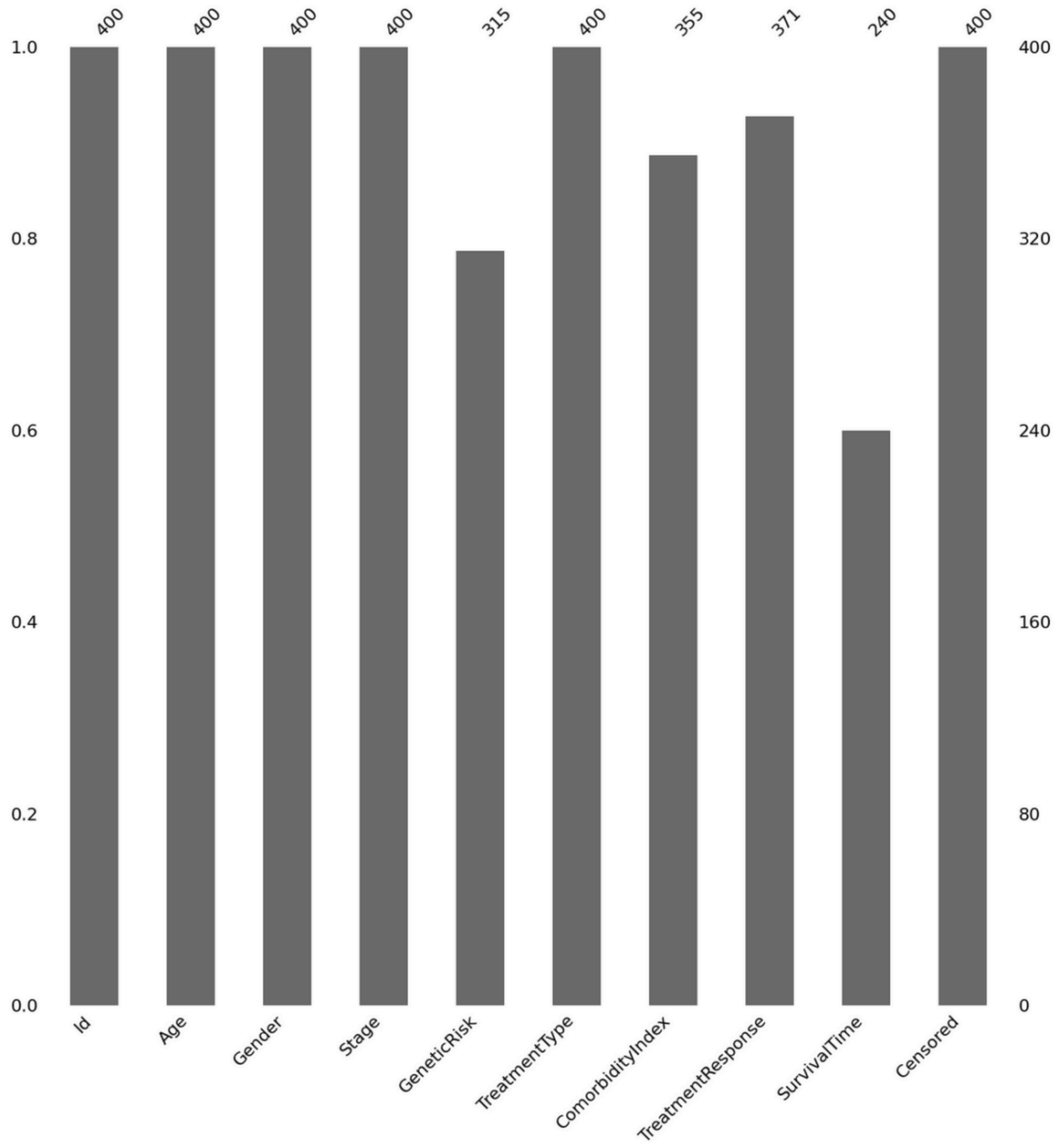
4.2 Evaluation

WHAT WAS DONE IN TASK 4.1 - **IMPUTATION**

As shown in the bar plot, the target variable SurvivalTime has missing values. This is called **unlabelled data.**

In this task we used the features from the labeled data to train an imputer. We then used that imputer to apply **Iterative Imputation** , which was the best imputation strategy found in Task 3.1, to the unlabelled data. Finally, we used the imputed data with labels to train a Linear Regression model. The results obtained by this model are on the next slide.

After training the Linear Regression model we trained an Isomap. We did not reduce the dimensions with a PCA before applying the Isomap to the data.



WHAT WAS DONE IN TASK 4.2 - **EVALUATION**

Baseline Model

Locally
cMSE: **5.8756**
MSE: **2.4239**

Kaggle
Public Score: **2.77797**
Private Score: **3.21643**

Task 3.1 Model

Locally
cMSE: **2.0665**
MSE: **1.4375**

Kaggle
Public Score: **2.57026**
Private Score: **2.73313**

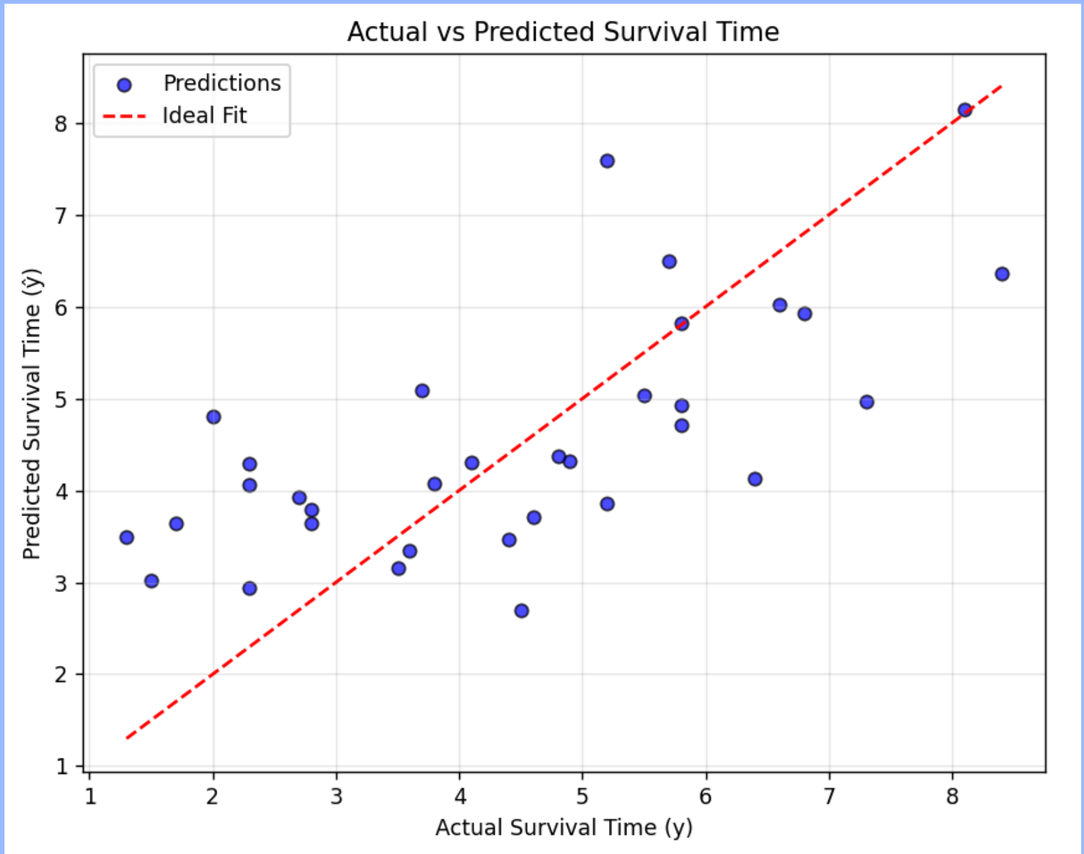
Task 4.1 Model

Locally
cMSE: **1.3265**
MSE: **1.1473**

Kaggle
Public Score: **2.89899**
Private Score: **2.39519**

This Linear Regression model locally obtained a score of **1.3265** and computed a score of **2.39519** in Kaggle. Comparing with the baseline model and the model chosen on Task 3.1 we see an improvement in the **cMSE** and in the **MSE**. This improvement was obtained due to the data imputation performed on the unlabelled data. These results however, were not the ones that were chosen for our final submission for this task, those are in the next slide.

Semi-supervised Model



cMSE: **1.9171**

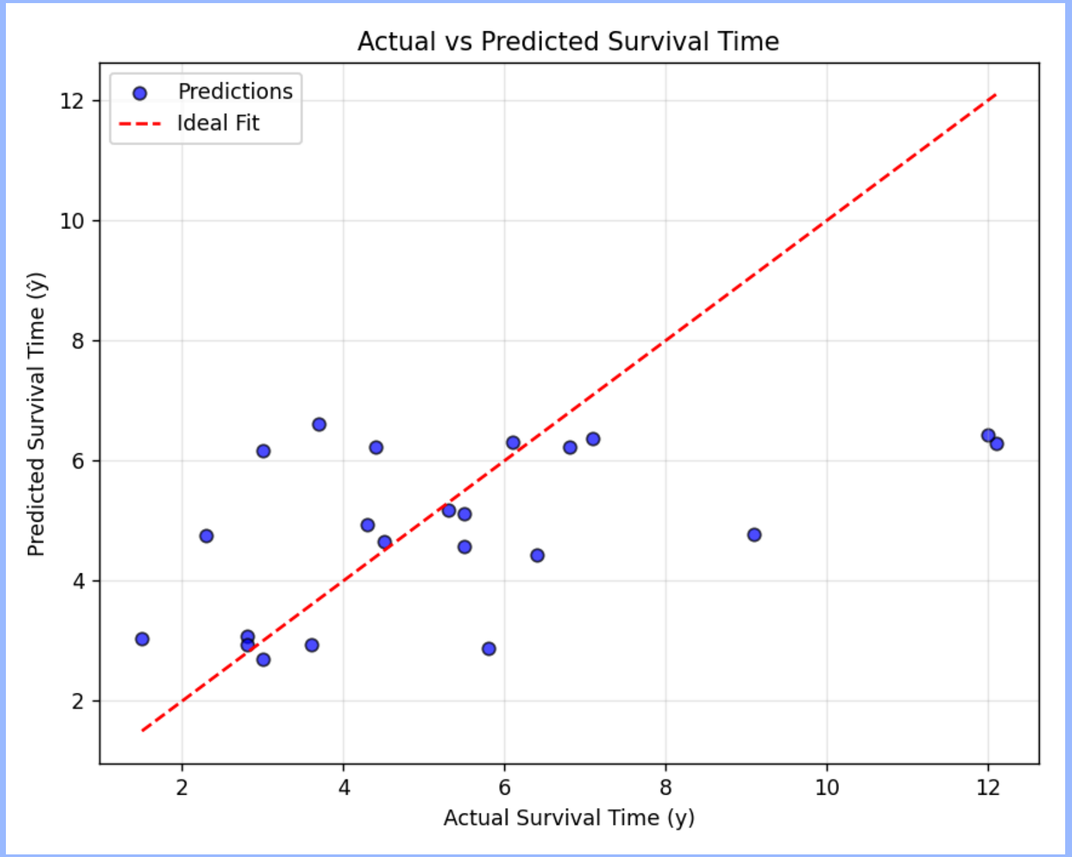
MSE: **1.3846**

Kaggle

Public score: **2.69128**

Private score: **2.74011**

Baseline Model



cMSE: **5.8756**

MSE: **2.4239**

Kaggle

Public score: **2.77797**

Private score: **3.21643**

OVERALL ASSESSMENT

- What went wrong
- What went great

WHAT WENT WRONG

In Task 4, we encountered difficulties understanding the task. Specifically, we misunderstood that we were supposed to use the labeled data features to train an imputer and apply it to the unlabeled data. Although we couldn't confirm whether the approach was wrong, we initially used a cross-validation method that we later found wasn't working properly. We replaced it with another method, but this led to worse results than our public scores, which were performing well. As a result, we decided not to submit models trained with the new cross-validation method, as the results were worse and we couldn't verify if the cross-validation was accurate. Interestingly, these submissions performed a lot better on the private leaderboard, which explains the discrepancy between our public and private results. In my opinion, I don't see the value in this split, of hidding a % of data, because if we were confident that our cross-validation was functioning properly, we would have trusted it and only submitted those models.

WHAT WENT GREAT

While making our baseline model, before reading the whole assignment, we decided to impute the missing data because we wanted to have a larger dataset, so we essentially skipped all the way to 3.1 and we ended up getting our best public submission right away.