Import Libraries

```
Requirement already satisfied: scipy in c:\users\xyber\anaconda3\envs\sanraj1\lib\site-packages
(1.9.0)
Requirement already satisfied: numpy<1.25.0,>=1.18.5 in c:\users\xyber\anaconda3\envs\sanraj1\lib
\site-packages (from scipy) (1.23.1)
Requirement already satisfied: sklearn in c:\users\xyber\anaconda3\envs\sanraj1\lib\site-packages
(0.0)
Requirement already satisfied: scikit-learn in c:\users\xyber\anaconda3\envs\sanraj1\lib\site-pack
ages (from sklearn) (1.1.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\xyber\anaconda3\envs\sanraj1\lib\site-pac
kages (from scikit-learn->sklearn) (1.23.1)
Requirement already satisfied: joblib>=1.0.0 in c:\users\xyber\anaconda3\envs\sanraj1\lib\site-pac
kages (from scikit-learn->sklearn) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\xyber\anaconda3\envs\sanraj1\lib\s
ite-packages (from scikit-learn->sklearn) (3.1.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\xyber\anaconda3\envs\sanraj1\lib\site-pack
ages (from scikit-learn->sklearn) (1.9.0)
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Requirement already satisfied: imbalanced-learn in c:\users\xyber\anaconda3\envs\sanraj1\lib\site-
packages (from imblearn) (0.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\xyber\anaconda3\envs\sanraj1\lib\s
ite-packages (from imbalanced-learn->imblearn) (3.1.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\xyber\anaconda3\envs\sanraj1\lib\site-pac
kages (from imbalanced-learn->imblearn) (1.23.1)
Requirement already satisfied: joblib>=1.0.0 in c:\users\xyber\anaconda3\envs\sanraj1\lib\site-pac
kages (from imbalanced-learn->imblearn) (1.1.0)
Requirement already satisfied: scikit-learn>=1.1.0 in c:\users\xyber\anaconda3\envs\sanraj1\lib\si
te-packages (from imbalanced-learn->imblearn) (1.1.2)
Requirement already satisfied: scipy>=1.3.2 in c:\users\xyber\anaconda3\envs\sanraj1\lib\site-pack
ages (from imbalanced-learn->imblearn) (1.9.0)
Installing collected packages: imblearn
Successfully installed imblearn-0.0
```

Read in data

	ID	Survive	Age	AgeGroup	Sex	Infection	SysBP	Pulse	Emergency
0	4	0	87	3	1	1	80	96	1
1	8	1	27	1	1	1	142	88	1
2	12	1	59	2	0	0	112	80	1
3	14	1	77	3	0	0	100	70	0
4	27	0	76	3	1	1	128	90	1

Clean data

- Null values
- Duplicated rows
- Column data types

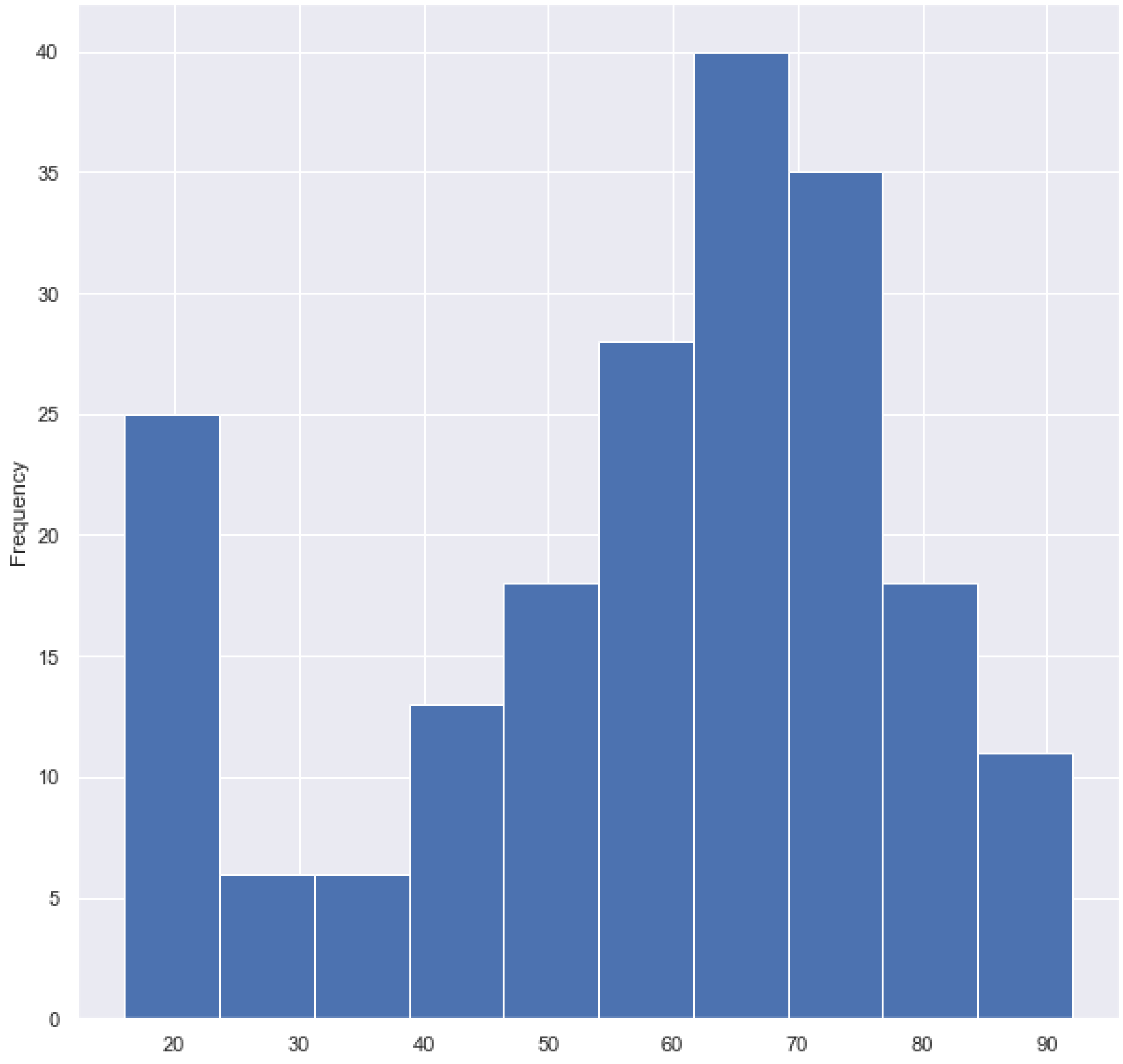
- Outliers

```
ID           False
Survive      False
Age          False
AgeGroup     False
Sex          False
Infection    False
SysBP        False
Pulse        False
Emergency    False
dtype: bool
```

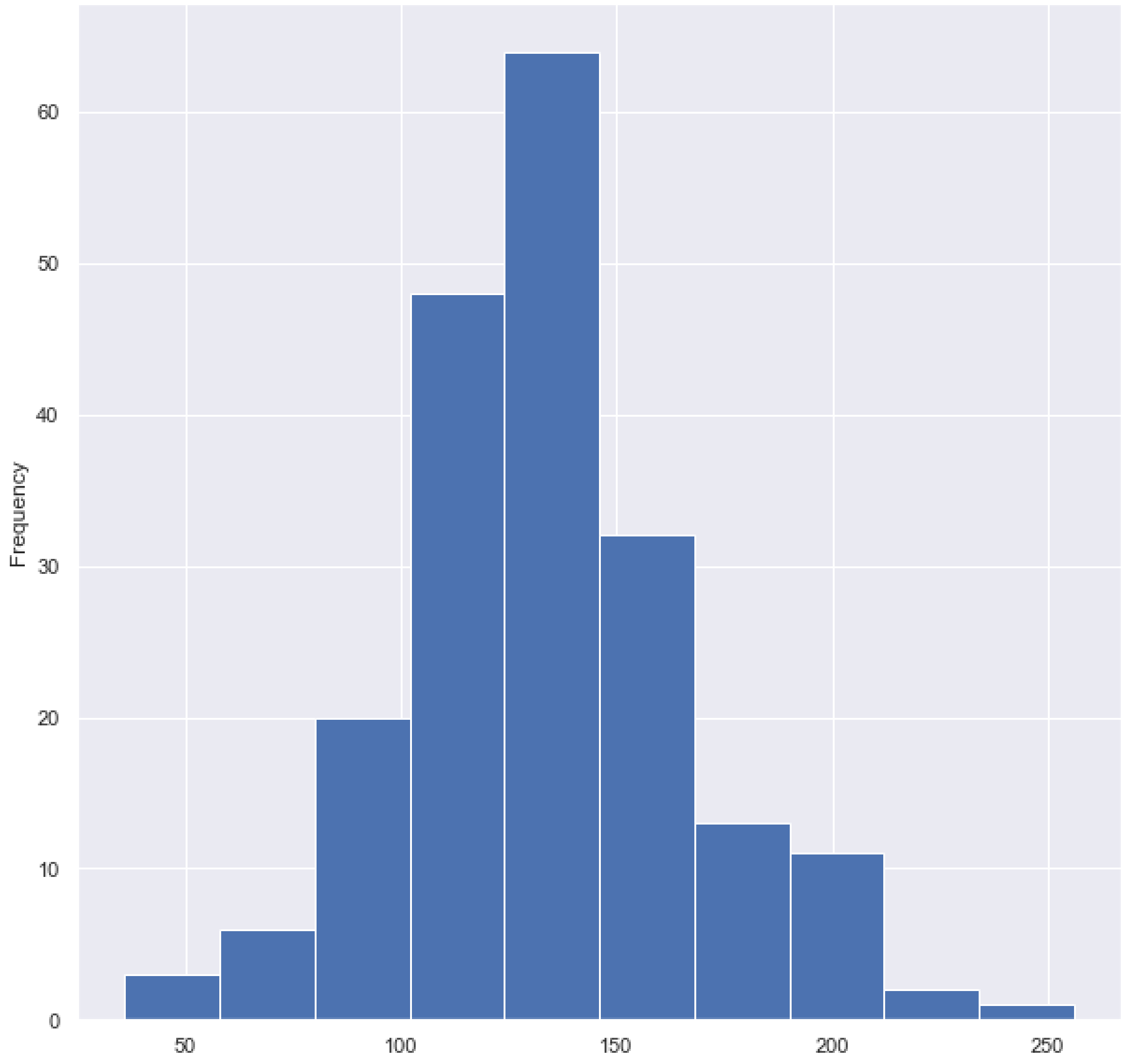
```
Number of duplicated rows: 0
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   ID         200 non-null    int64
 1   Survive    200 non-null    int64
 2   Age        200 non-null    int64
 3   AgeGroup   200 non-null    int64
 4   Sex        200 non-null    int64
 5   Infection  200 non-null    int64
 6   SysBP      200 non-null    int64
 7   Pulse      200 non-null    int64
 8   Emergency  200 non-null    int64
dtypes: int64(9)
memory usage: 14.2 KB
```

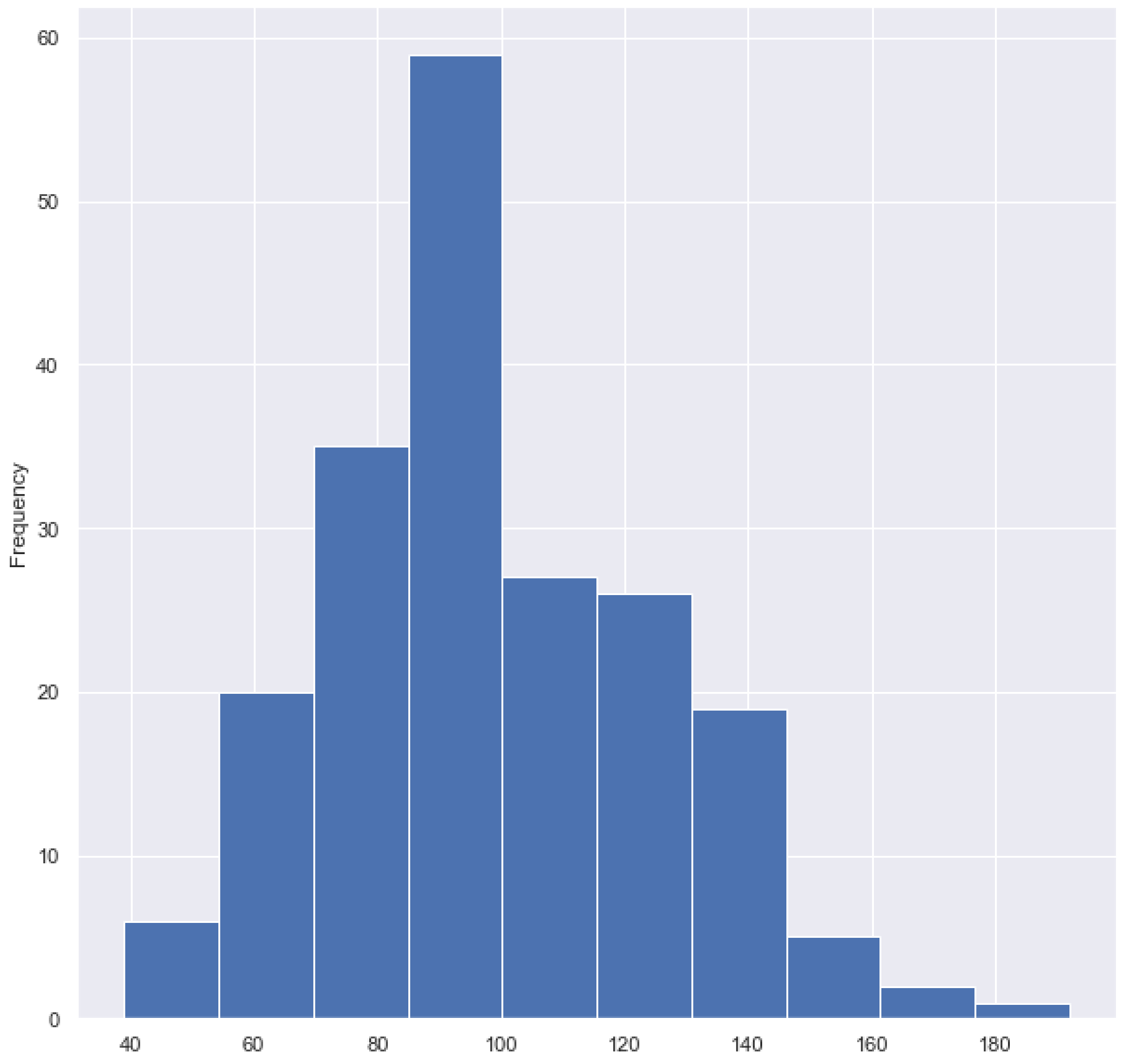
```
<AxesSubplot:ylabel='Frequency'>
```


Frequency
0
5
10
15
20
25
30
35
40
20
30
40
50
60
70
80
90

```
<AxesSubplot:ylabel='Frequency'>
```


Frequency
0
10
20
30
40
50
60
50
100
150
200
250

```
<AxesSubplot:ylabel='Frequency'>
```



Overall, the dataset looks very clean. Let us convert it to a sqlite3 table to run some queries on it.

```
200
```

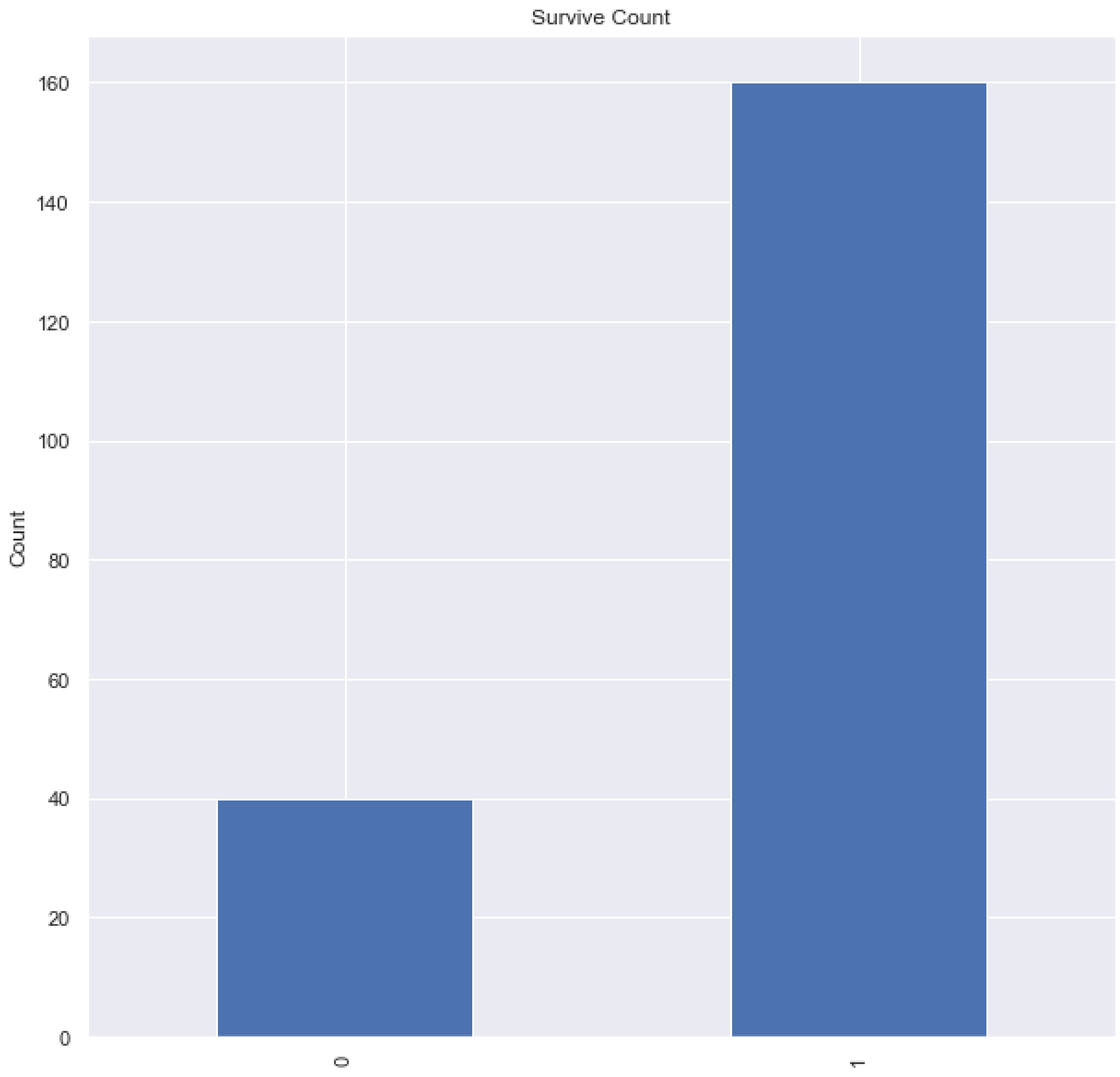
Visualizations

Let us investigate:

- Imbalance in response variable
- How each feature affects the response variable
- Correlation between features

Response Variable

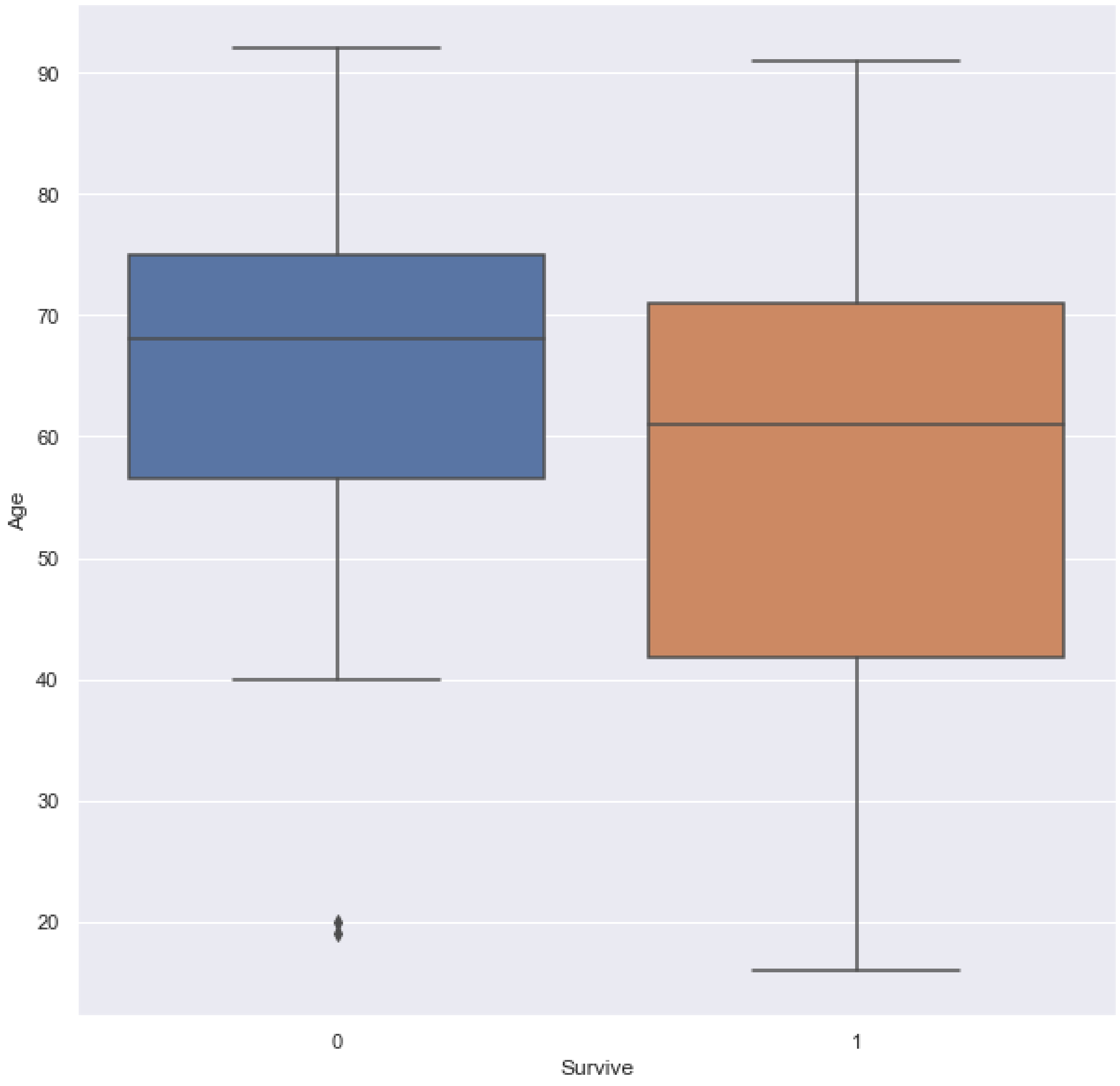
```
<AxesSubplot:title={'center':'Survive Count'}, ylabel='Count'>
```



We can see that there is an imbalance in the response variable where there are 4 times more patients who survives than those who did not surive, and we can deal with this in a few ways. We can either use resamling techniques, changing weights or leave it as it is and see model performance. For now, I plan to change the weights of given models.

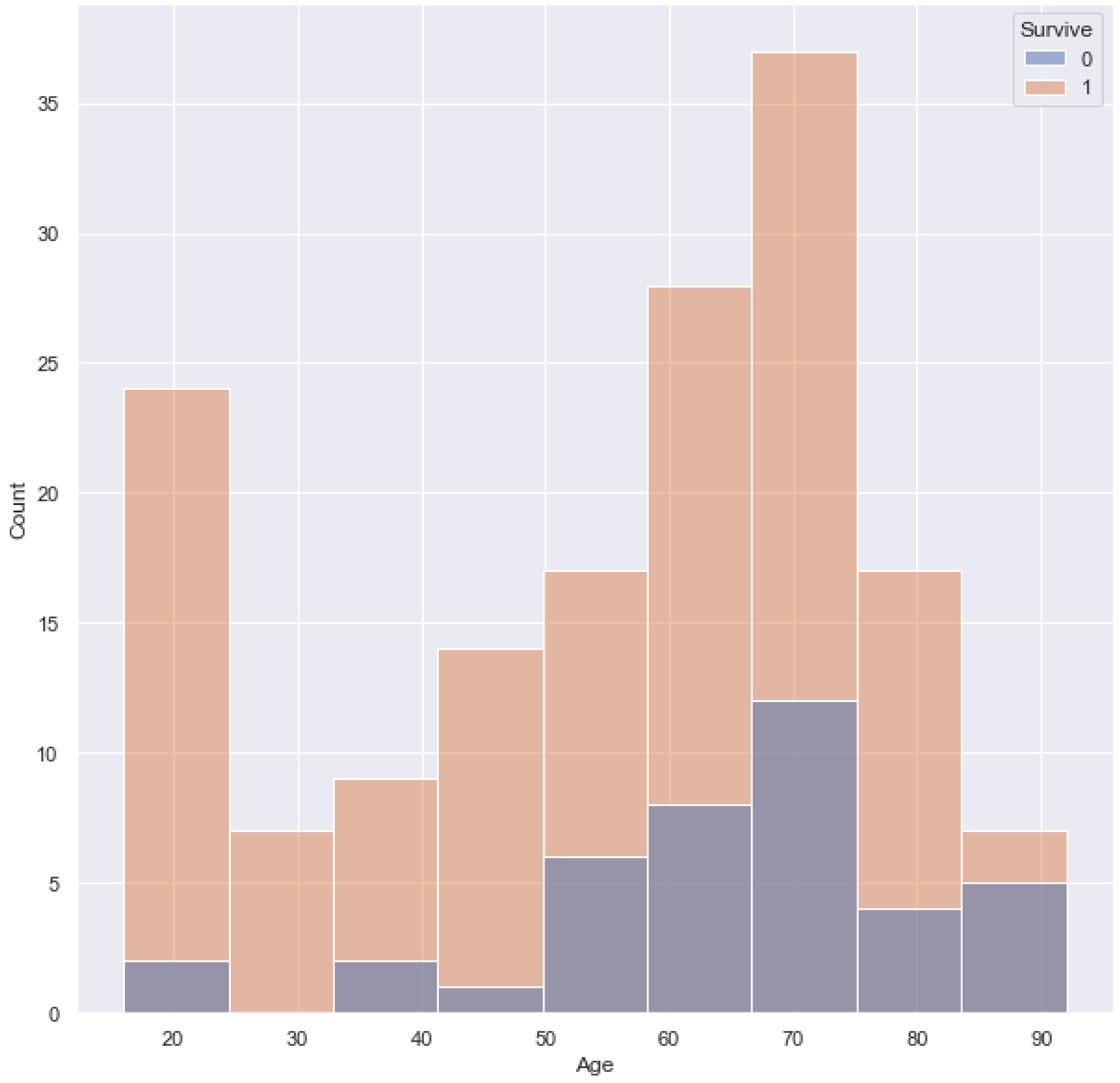
Age

```
<AxesSubplot:xlabel='Survive', ylabel='Age'>
```

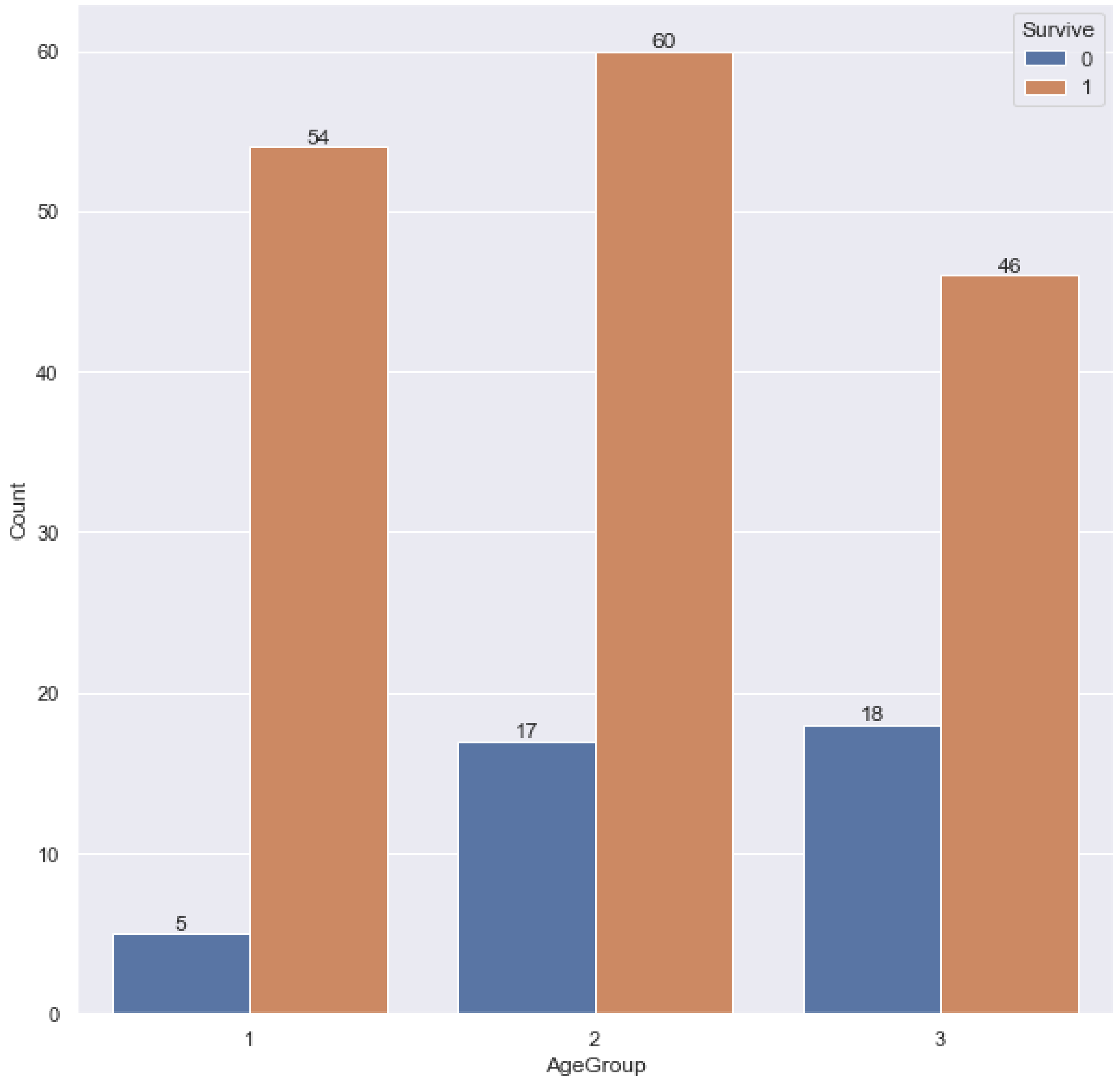


We can see that those who don't survive are likely to be older.

```
<AxesSubplot:xlabel='Age', ylabel='Count'>
```

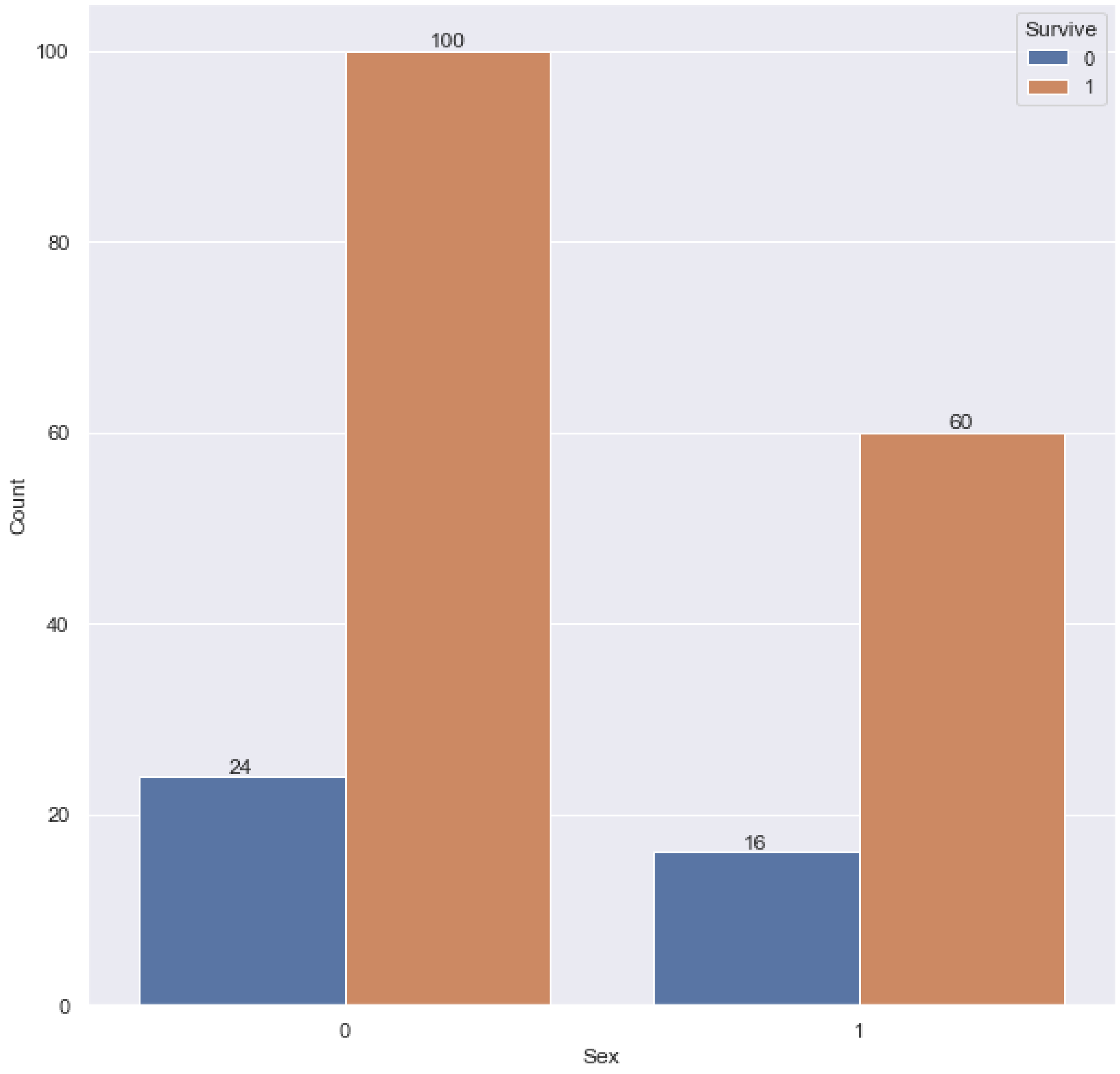


We can also see that for the distribution of the Ages, there are mainly 70 year olds, and the majority of the 20 year olds survive.



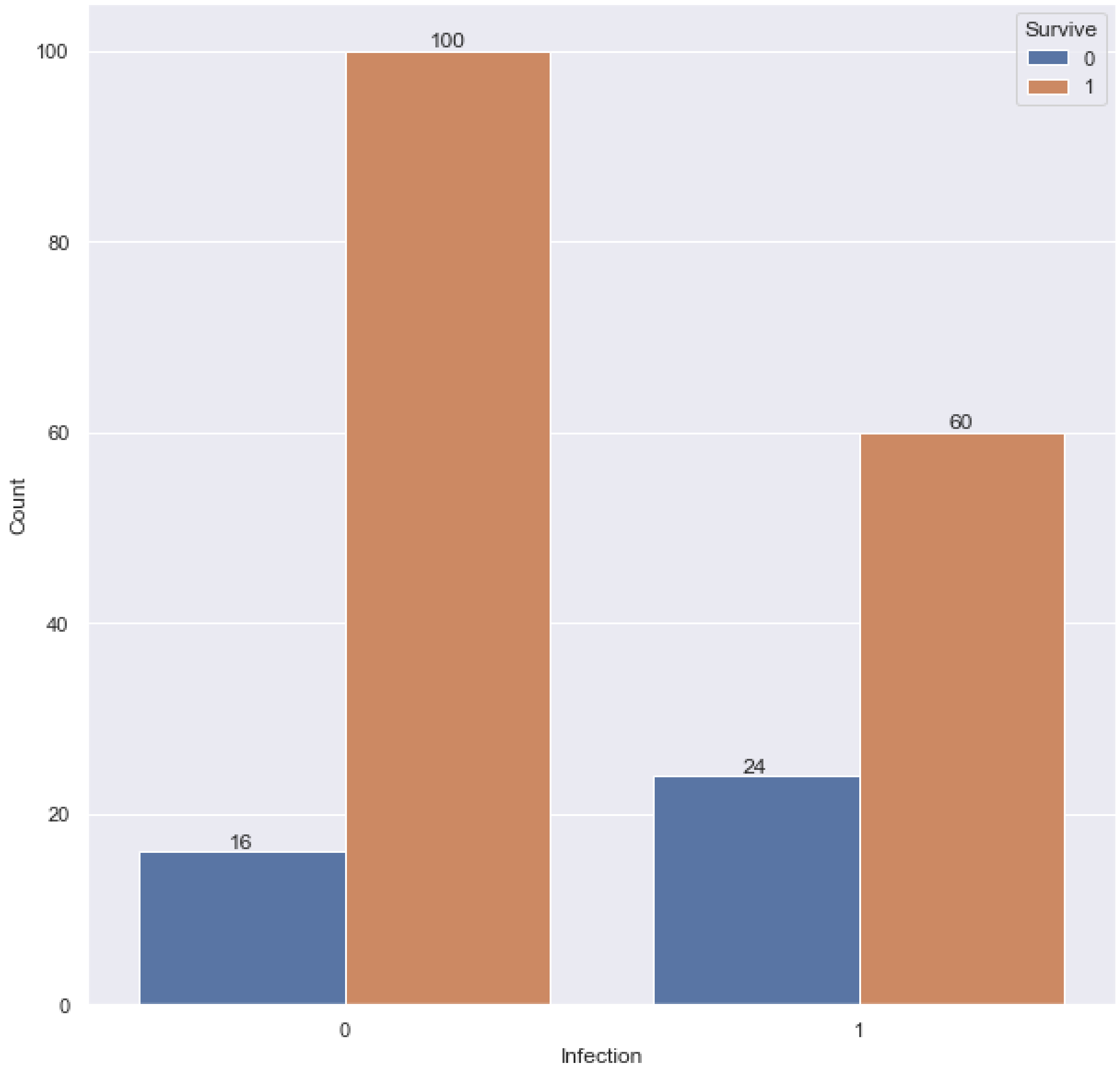
From the above plot, we can tell that those who fall into the third group for Age, presumably the oldest age range group, has the lowest chance of surviving whereas if you are from group 1, presumably the youngest, you have the highest chance of surviving.

Sex



There is no information given about which value of sex corresponds to male/female so we cannot conclude much using those terms. However, based on the plot, we can see that for those who are of Sex == 1, they have a higher proportion of people who don't survive compared to those who are of Sex==0.

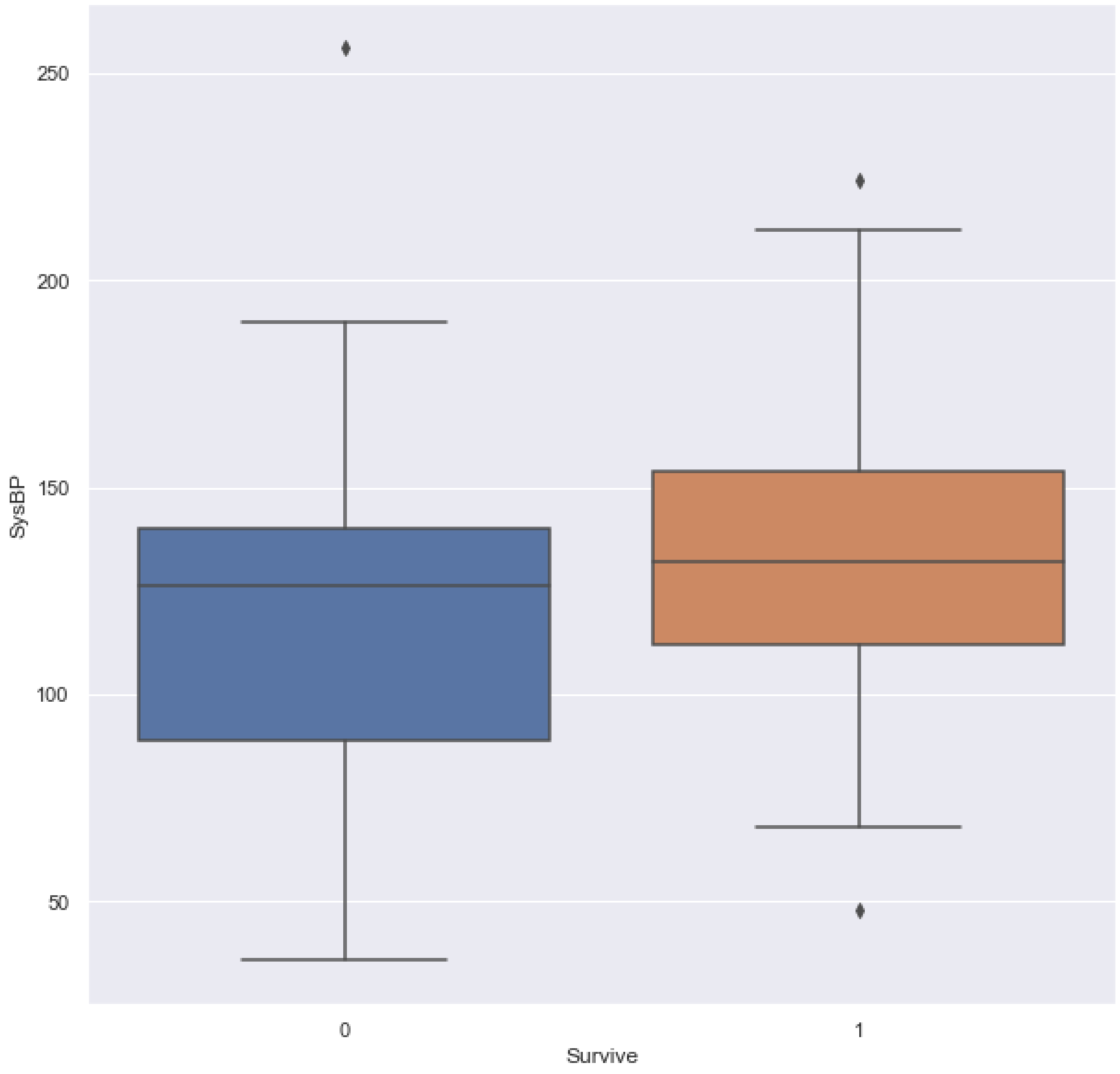
Infection



There is no information given about which value Infection corresponds to, but let us assume Infection == 1 means they have an infection. Based on the plot, we can see that for those who have Infection == 1, they have a higher proportion of people who don't survive compared to those who have Infection == 0.

Blood Pressure

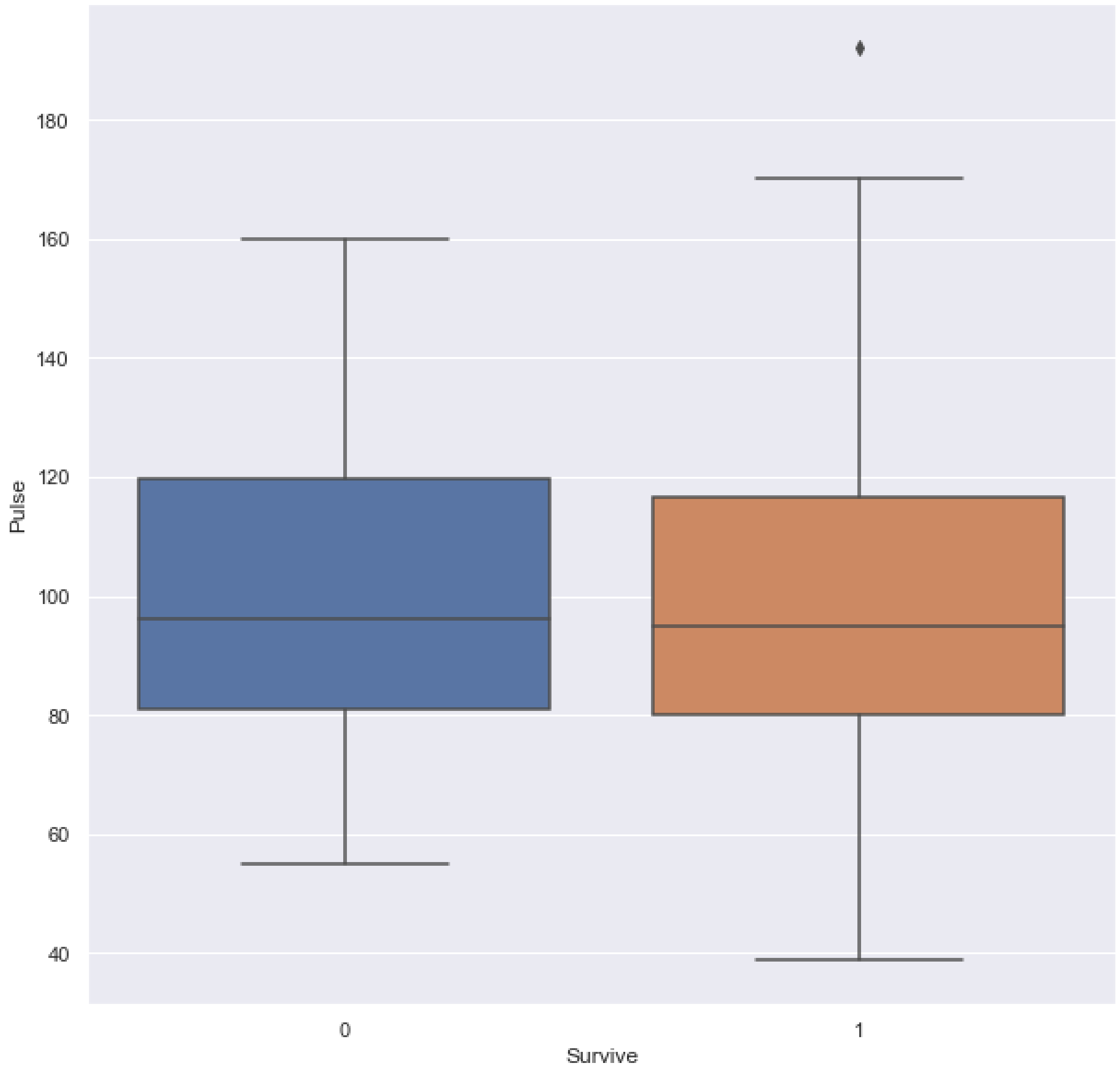
```
<AxesSubplot:xlabel='Survive', ylabel='SysBP'>
```



For those who tend to not survive, their blood pressure is lower.

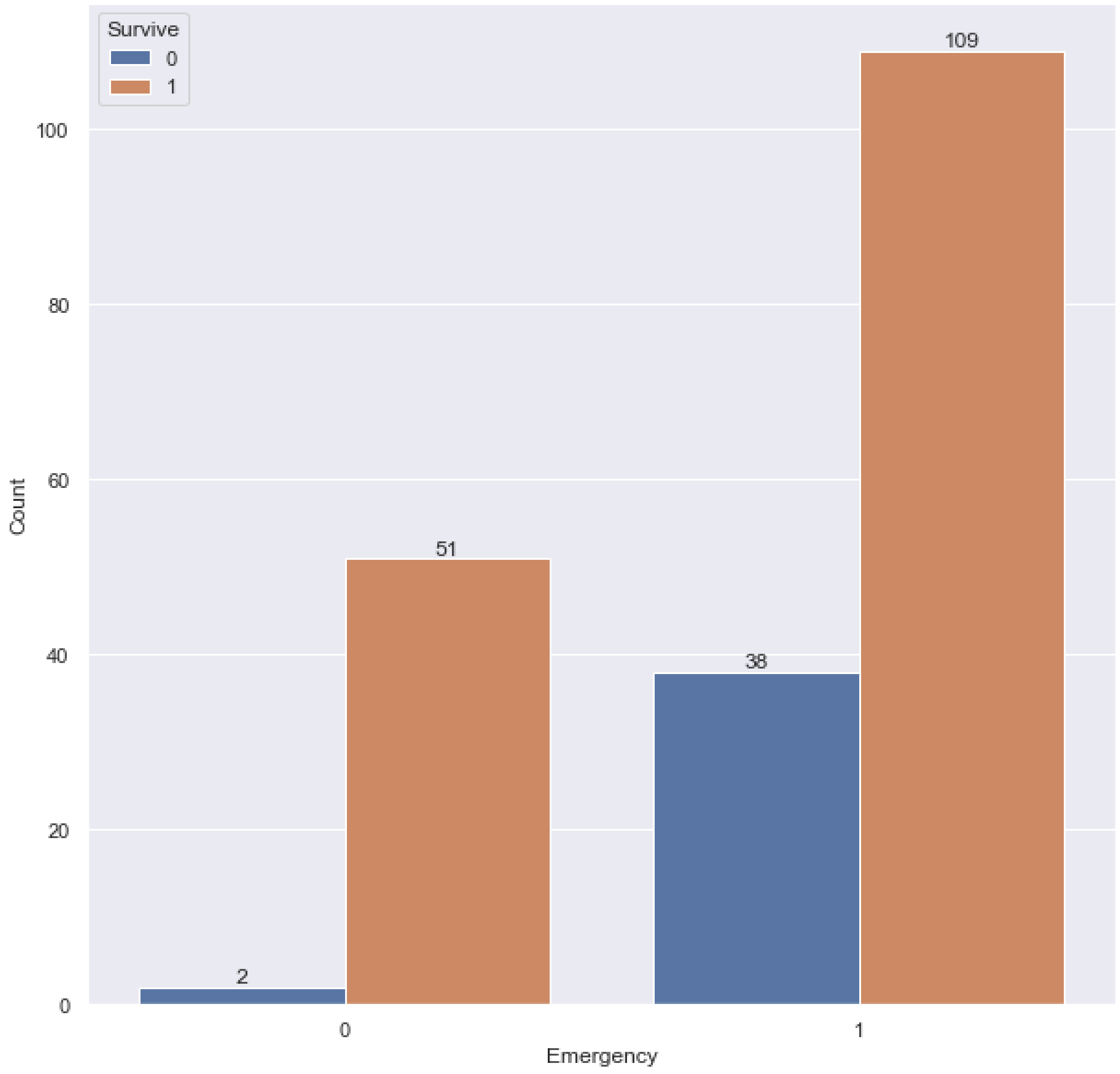
Pulse

```
<AxesSubplot:xlabel='Survive', ylabel='Pulse'>
```



It seems as though Pulse of a patient does not significantly determine whether a patient survives or not, but maybe one could say based on the Box Plot above, people who don't survive have a slightly higher pulse.

Emergency



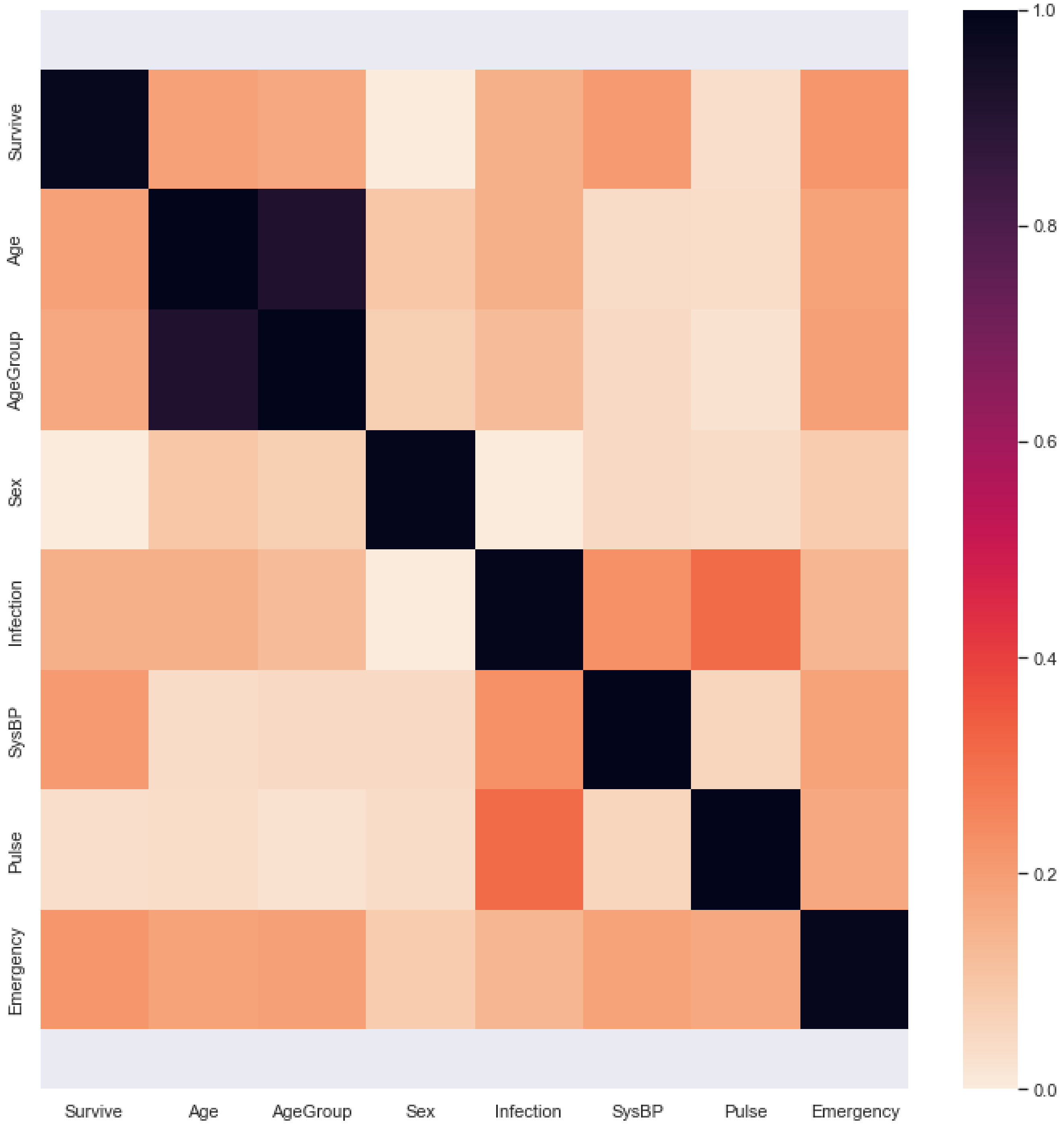
We can clearly see from the above plot that, for a patient who is admitted to the emergency room, they have a higher chance of surviving.

Correlation

Let us plot the correlation for between each feature and check for multicollinearity as well as which features are correlated to the response variable, Surviveability. We use the following metrics to measure correlation:

- Pearson (Cont - Cont)
- Cramer v (Cat - Cat)
- Correlation Ratio (Cat - Cont)

```
Age,AgeGroup 0.915577261411191
```



From the above plot, we can see that Age and AgeGroup are highly correlated which is not surprising. We may drop one of the columns for the Logistic Regression Model since it is affected by multi-collinearity but leave it in for the Decision Tree and Random Forest since mutlicollinearity does not affect decision trees. Let us drop AgeGroup as it is less correlated to the response variable than Age.

Modelling v1 - Train using original data without resampling techniques

- Train Test Split
- Standardize
- Modelling

Train Test Split

We used Stratified Sampling so as to ensure that the features that have the greatest influence on our response are equally distributed in the training and testing data set. Doing so would also ensure that both the training and testing dataset contain the same ratio of classes. Using the StratifiedShuffleSplit class from the sklearn library in python, we split the original data set into a 80:20 ratio, where 80% of the data set is used for training, and the rest for testing.

Standardize Training Data

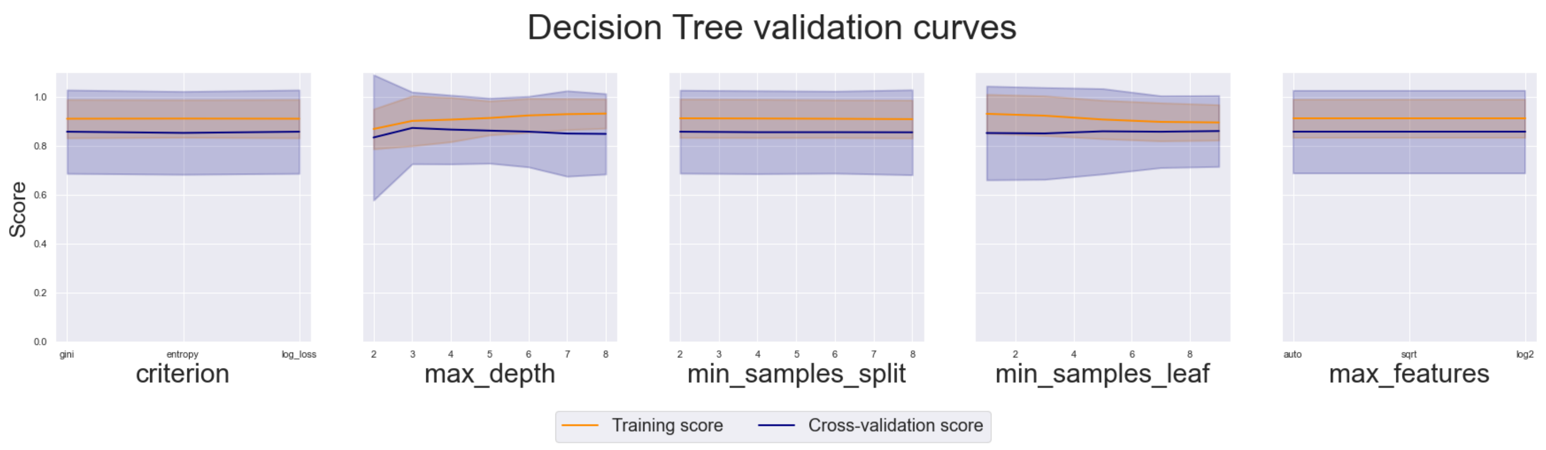
```
C:\Users\xyber\AppData\Local\Temp\ipykernel_9100\1896789836.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  X_train['Age'] = train_scaled_cols[:, 0]
C:\Users\xyber\AppData\Local\Temp\ipykernel_9100\1896789836.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  X_train['SysBP'] = train_scaled_cols[:, 1]
C:\Users\xyber\AppData\Local\Temp\ipykernel_9100\1896789836.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  X_train['Pulse'] = train_scaled_cols[:, 2]
C:\Users\xyber\AppData\Local\Temp\ipykernel_9100\1896789836.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  X_test['Age'] = test_scaled_cols[:, 0]
C:\Users\xyber\AppData\Local\Temp\ipykernel_9100\1896789836.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  X_test['SysBP'] = test_scaled_cols[:, 1]
C:\Users\xyber\AppData\Local\Temp\ipykernel_9100\1896789836.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  X_test['Pulse'] = test_scaled_cols[:, 2]
```

Hyperparameter Tuning

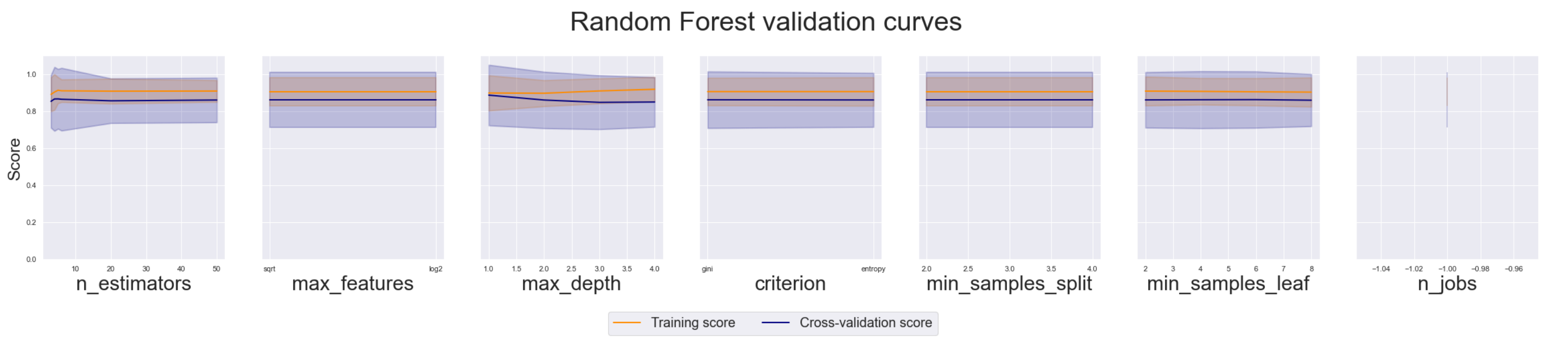
- We need to reduce the number of False Positives in our dataset, and hence, we shall target our tuning on maximizing our precision score. Precision is given by TP/ (TP+FP).

We also want to make sure that our model has not overfitted, so we plot the validation and training curves.

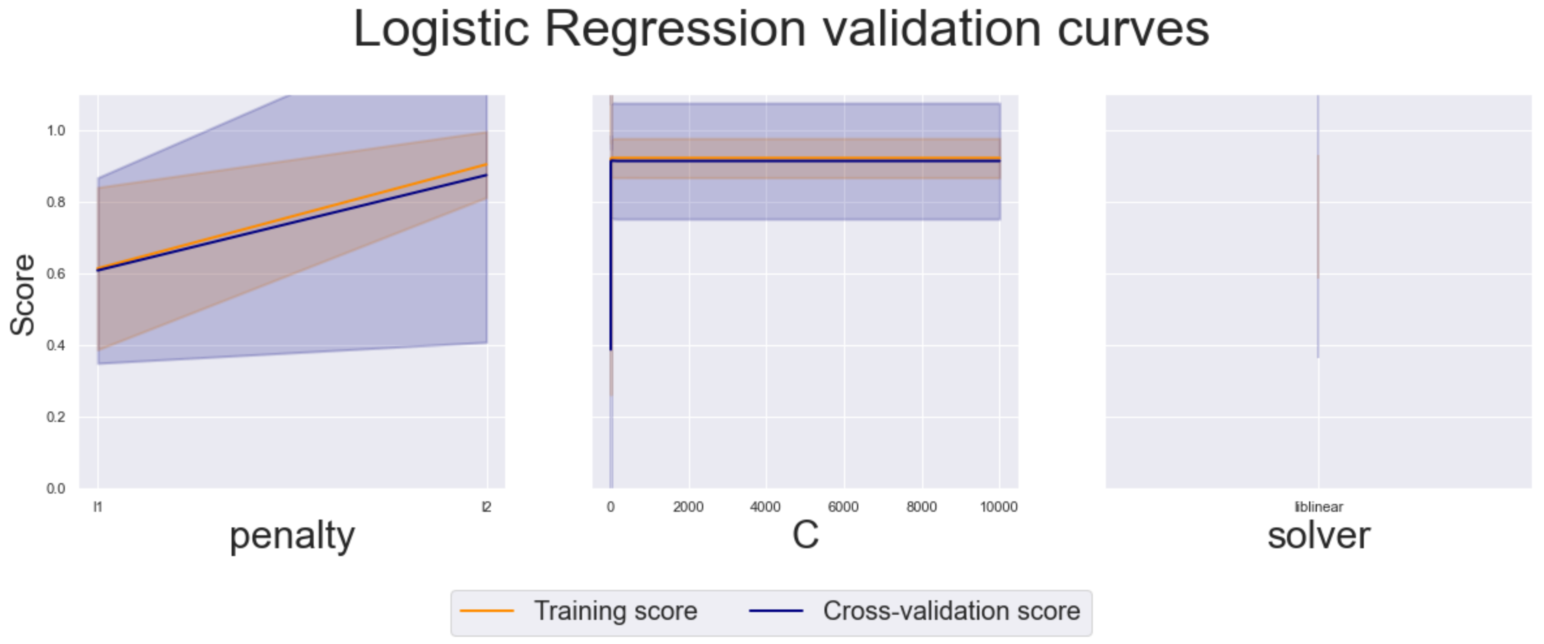
```
{'precision': {'class_weight': 'balanced',
  'criterion': 'entropy',
  'max_depth': 3,
  'max_features': 'auto',
  'min_samples_leaf': 5,
  'min_samples_split': 2}}

{'precision': {'class_weight': 'balanced',
  'criterion': 'gini',
  'max_depth': 1,
  'max_features': 'sqrt',
  'min_samples_leaf': 2,
  'min_samples_split': 2,
  'n_estimators': 5,
  'n_jobs': -1}}

{'precision': {'C': 0.046415888336127774,
  'class_weight': {0: 8, 1: 1},
  'penalty': 'l1',
  'solver': 'liblinear'}}
```



- We can see that for max_depth, the higher the value, the more the overfitting occurs.



Similar to the decision tree, as the max depth value increases, the model starts to overfit. Another thing to note is that the number of estimators , as increased, does not affect the model after a certain threshold of 5.



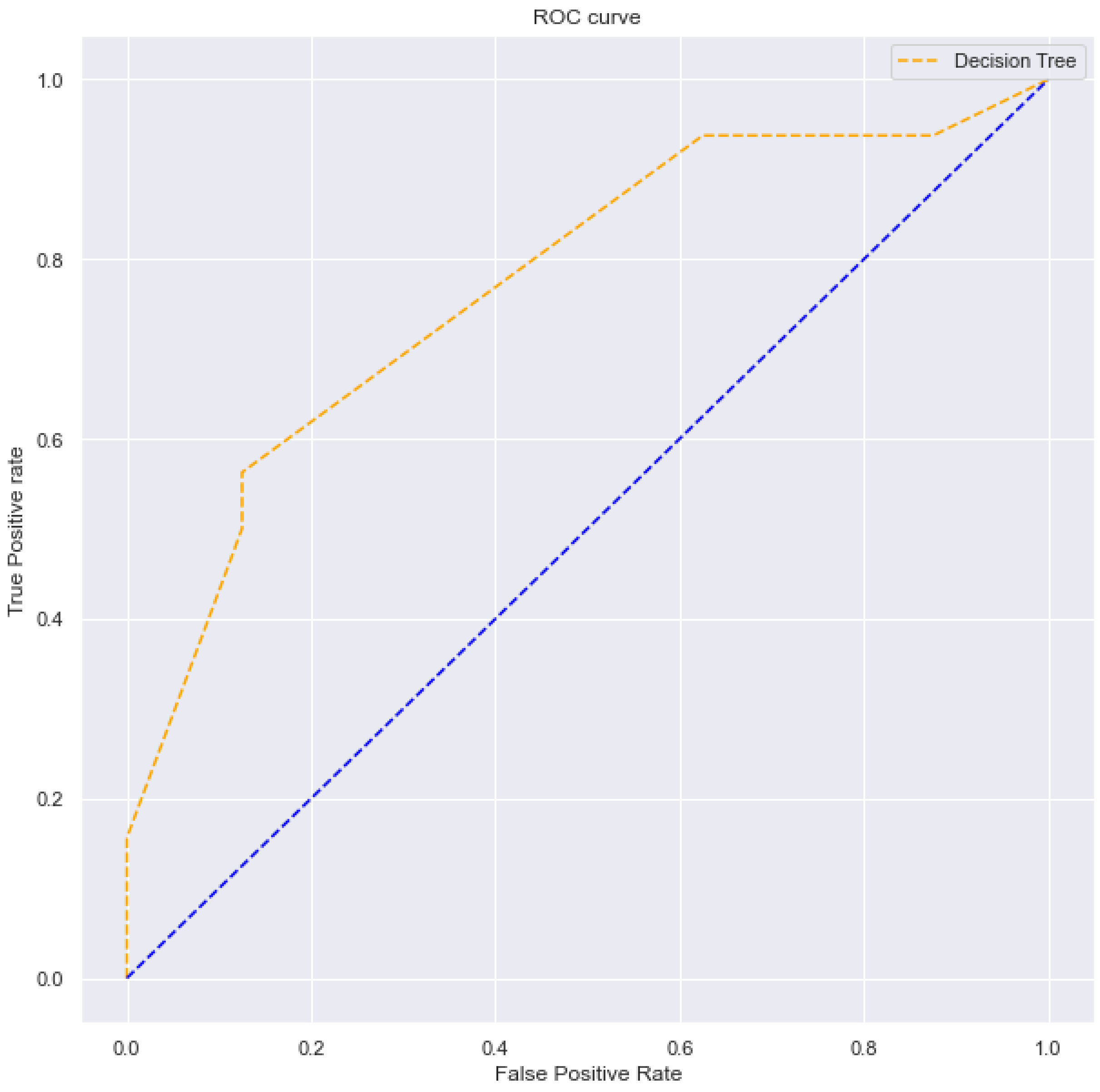
Testing

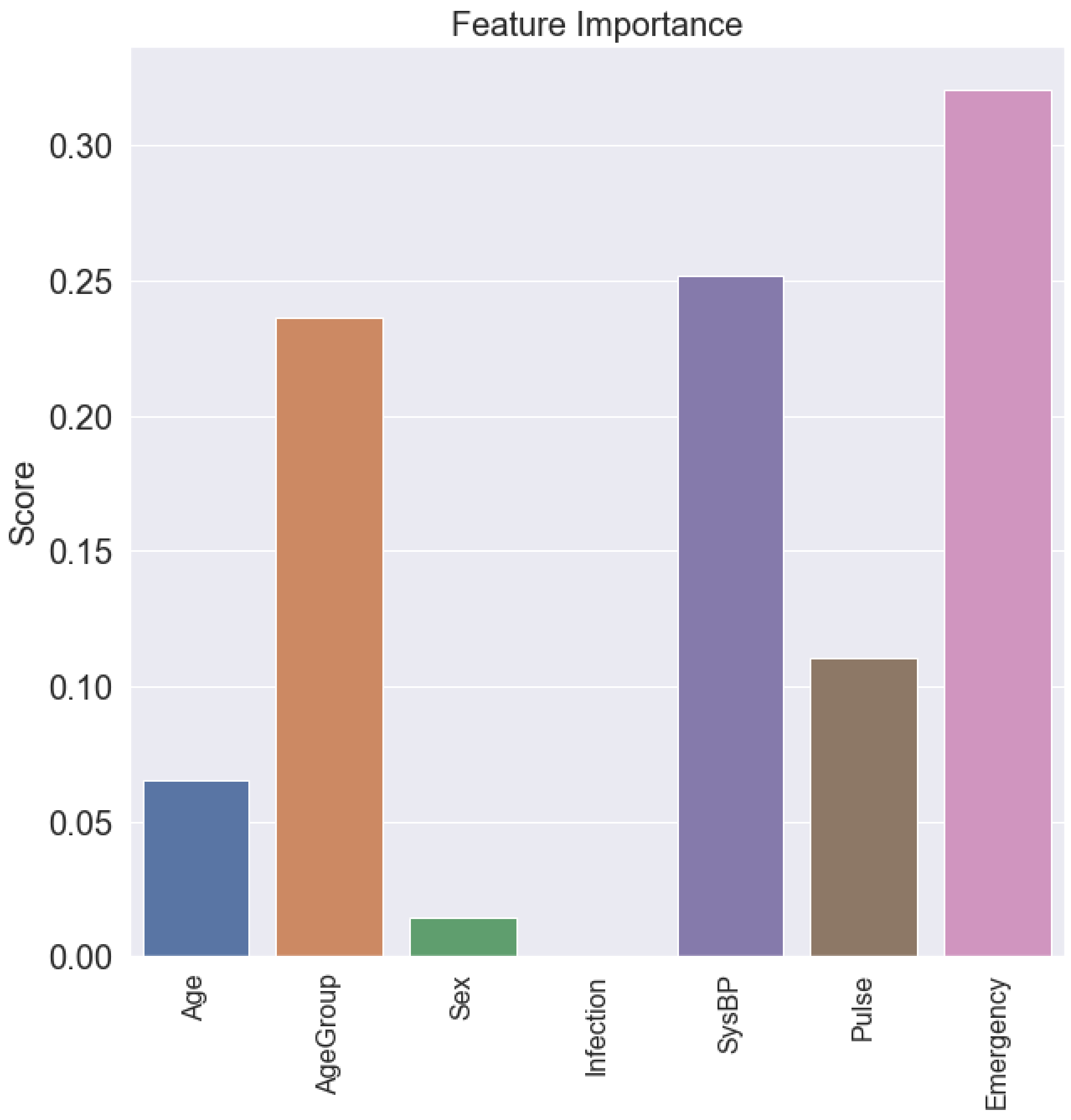
Scores

- Precision, Recall, Accuracy, AUCROC, F1, Specificity
- Confusion Matrix
- ROC Curve

Decision Tree

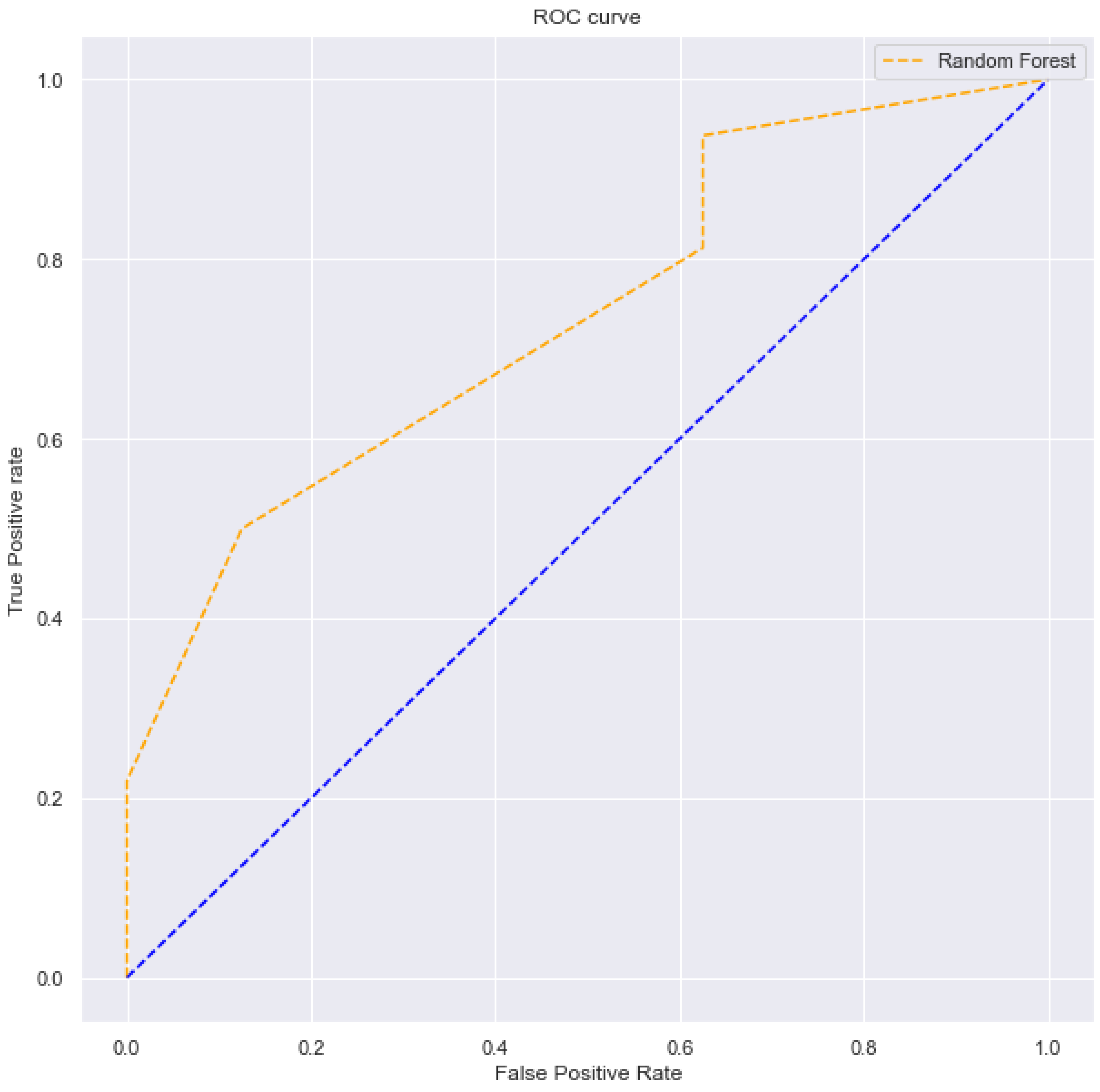
```
{'precision': 0.9444444444444444, 'recall': 0.53125, 'accuracy': 0.6, 'auc': 0.771484375, 'f1_scor
e': 0.6799999999999999, 'specificity': 0.875}
                 True Y = 0  True Y = 1
Predicted Y = 0           7          15
Predicted Y = 1           1          17
```

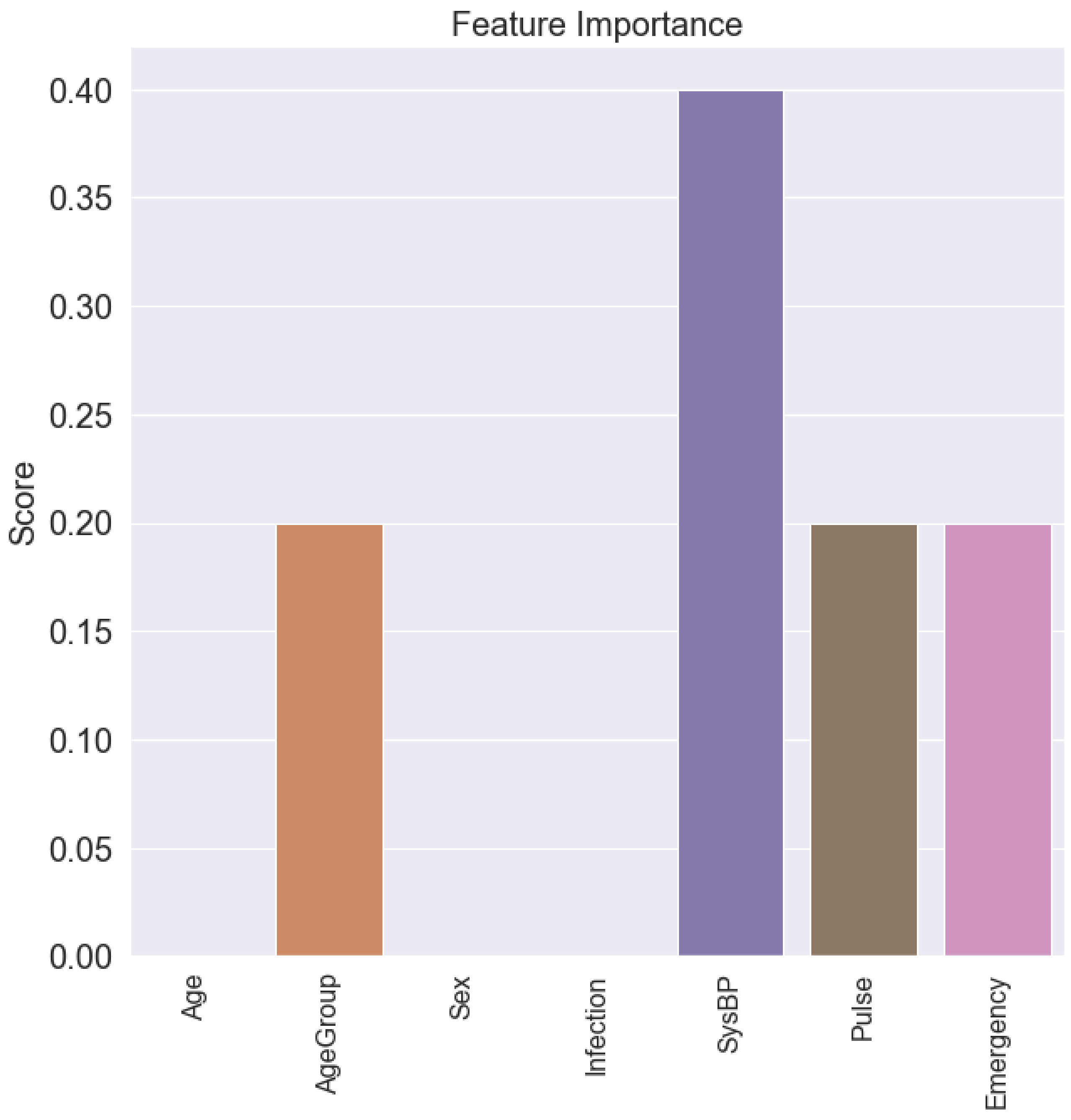

ROC curve
Decision Tree
True Positive rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0



Random Forest

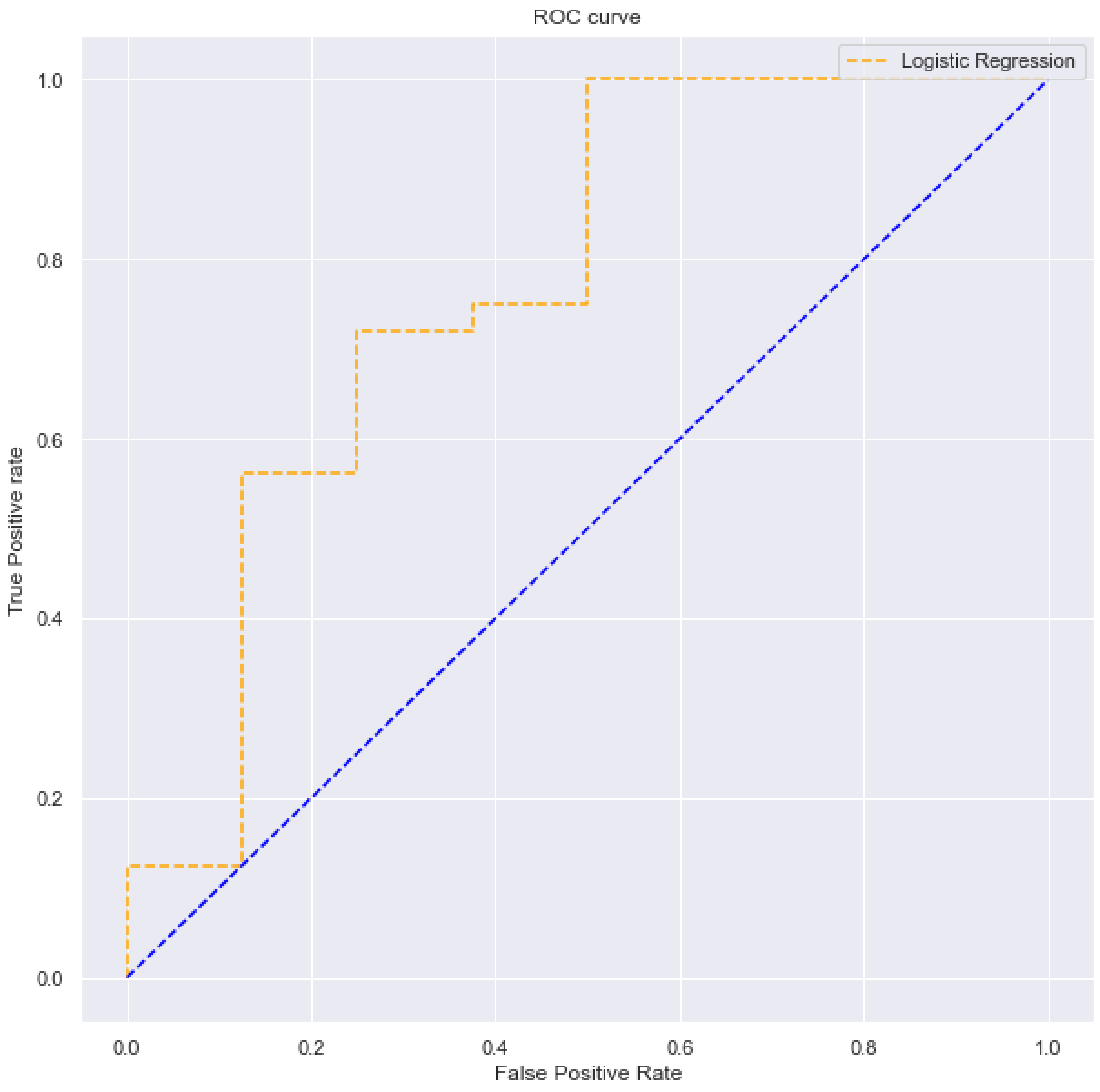
```
{'precision': 0.84375, 'recall': 0.84375, 'accuracy': 0.75, 'auc': 0.736328125, 'f1_score': 0.8437
5, 'specificity': 0.375}
                 True Y = 0  True Y = 1
Predicted Y = 0           3           5
Predicted Y = 1           5          27
```


ROC curve
Random Forest
True Positive rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0



Logistic Regression

```
{'precision': 1.0, 'recall': 0.09375, 'accuracy': 0.275, 'auc': 0.76953125, 'f1_score': 0.17142857
142857143, 'specificity': 1.0}
                 True Y = 0  True Y = 1
Predicted Y = 0           8          29
Predicted Y = 1           0           3
```



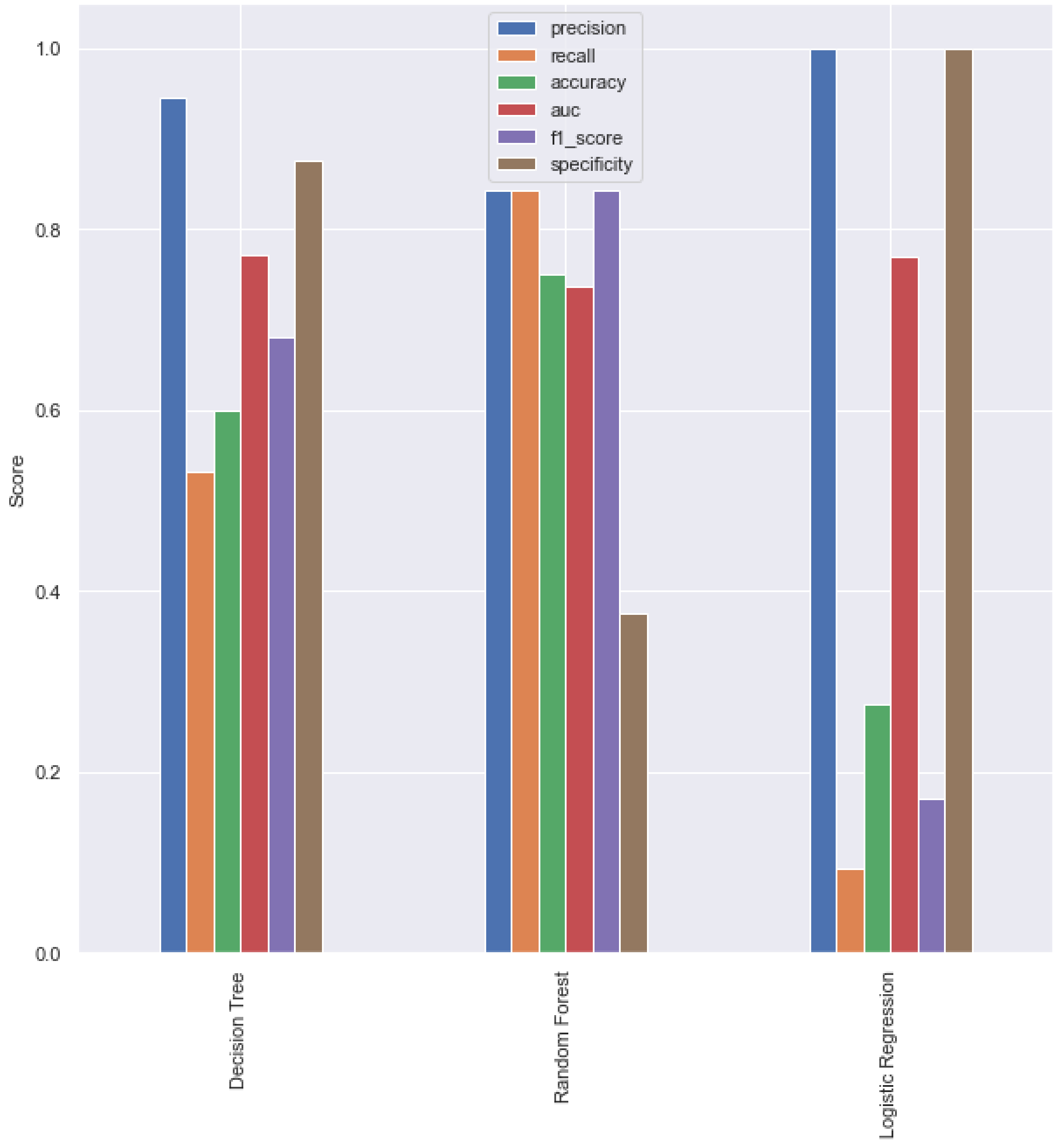
```
array([[-0.30047267,  0.        ,  0.        ,  0.09257882,  0.        ,
        -0.65646986]])
```

```
Index(['Age', 'Sex', 'Infection', 'SysBP', 'Pulse', 'Emergency'], dtype='object')
```

Here, we can see how the coefficients of the Logistic Regression model is behaving. For example, the coefficient of Age is -0.3, which tells us that for 1 unit increase in Age, holding all other predictors constant, the log odds of Survive=1 decreases by 0.3. Then, for features Sex, Infection and Pulse, the log odds of Y remains the same regardless of any change in the Predictor, holding all other predictors constant.

Full results

```
<AxesSubplot:ylabel='Score'>
```



Save Models

Discussion

We see that the Random Forest gives us the best all round results and this is to be expected as it an ensemble of the decision tree, whereas for the Decision Tree and Logistic Regression, though the precision values were high, it was traded off with recall. This means that the decision tree and logistic regression are not able to capture the 1's in the surviveability column well. This could mean there is a high bias present.

The decision tree still outperformed the logistic regression model in most of the metrics except Precision. Perhaps the data is not linearly separable, and hence this causes the Logistic Regression model to perform poorly, whereas for the tree based models, they are non-parametric models and hence are not hindered by non-linear data.

The decision tree gives the Emergency Column the most importance, followed by SysBP and then AgeGroup. For the random forest, SysBP is given the most importance. Interestingly enough, infection was not given any importance by both models.

For the logisitc regression model, we have deemed that any change in Age, Sex and Pulse columns do not change the log odds of Surviveability, while holding all other predictors constant.

Limitations

One major limitation of the project is the lack of data, as we only have access to 200 total samples. Out of that, 160 rows were used for training. To improve the performance of the models, we need access to more data.

Future Works

- Resampling Techniques such as SMOTE and Oversampling<- Need to manually implement Cross Validation if we want to do it.
- Try Recursive Feature Elimination with Cross Validation to get best features before carrying out modelling.
- Try more models such as the Naive Bayes Classifier or KNN.