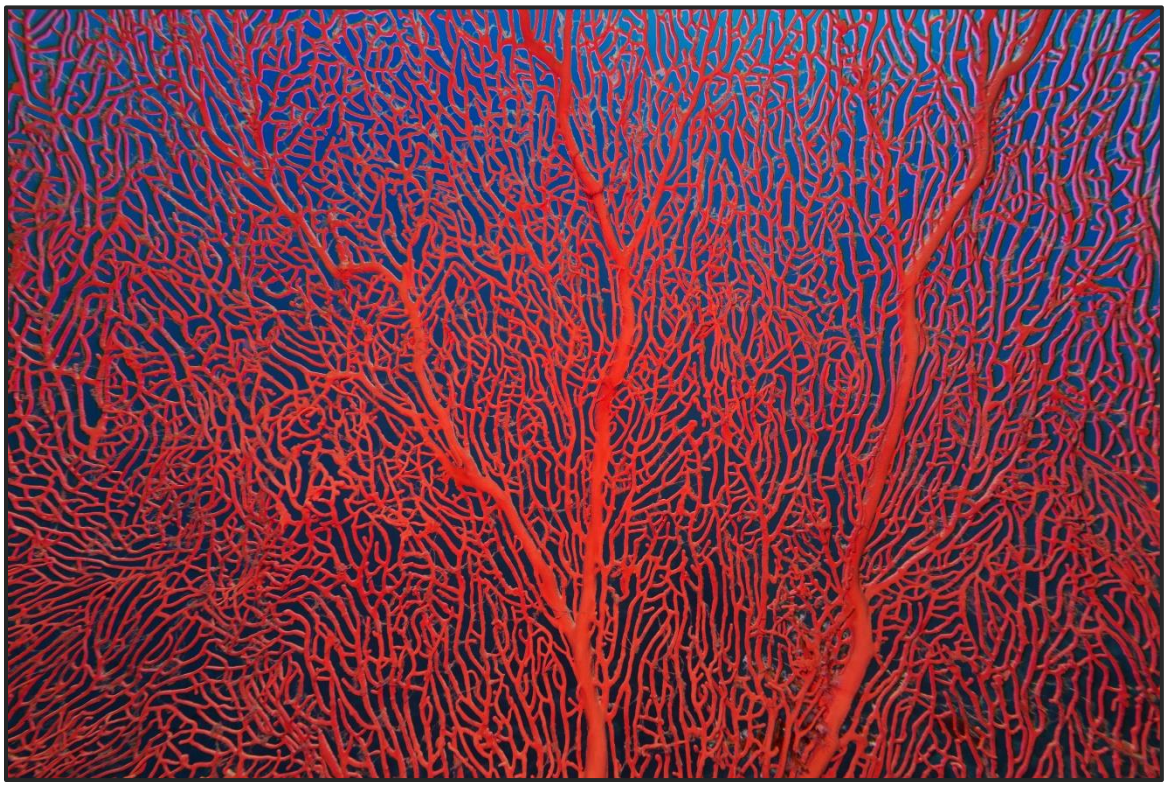

CMI7507 Assignment II

PREDICTION OF HEART FAILURE USING ARTIFICIAL NEURAL NETWORK

Carl Wilson | U0370630 | December 2021

THE UNIVERSITY OF HUDDERSFIELD
School of Computing and Engineering

Abstract

A dataset of 299 patient records containing a mix of binary and continuous health indicators was used to develop an Artificial Neural Network [ANN] to predict patient outcomes, based on five key indicators, to test accuracy and precision scores of 0.733 and 0.581 respectively through fine tuning of hyper-parameters.

Introduction

Identifying a patient's risk of heart failure within a given window of time can potentially improve the outcome, survival rate, for said patient. This report will study a dataset of health indicators to determine which indicators are strongly correlated to patient outcome and then attempt to build a model utilising an ANN architecture to precisely predict patient outcome. This model will further be developed through the tuning of ANN hyper-parameters such as the type of solver used, the learning rate of the solver, the number of neurons in and the number of hidden layers as well as the alpha regularisation parameter, with the aim of further improving the precision of the model so that more patients might be correctly identified as at risk.

Questions

Question 1A

The dataset (Chicco & Jurman, 2020) was provided in a comma separated value format with 13 features, including the target prediction of "death event". The dataset was imported into a Pandas data-frame in Python.

For each feature there are 299 records that are all numeric in type, with 3 of the 13 being floating point values and the remaining 10 being integers. The breakdown of feature name, count and data type can be seen in Table 1.

```
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    int64
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    int64
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    int64
 10  smoking                   299 non-null    int64
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
```

Table 1: Raw dataset imported from csv into Pandas.

From Table 1 it can be observed that there are no non-null values, and that the dataset appears complete; hence no values at this point need to be filled in.

All variables available within the dataset were used to produce a box plot, using Plotly for Python, to examine their distributions and type of variable. The visualisation was encoded to split out the two "death event" outcomes (dead and survived) on colour, so the distributions could be examined, this can be observed in Figure 1.

From the boxplot in Figure 1, it can be observed that 5 of the predictor features are binomial, meaning they are only a 0 or a 1. These variables are Anaemia, Diabetes, High Blood Pressure, Sex and Smoking with a 0 representing "Negative" and a 1 representing "Positive" respectively.

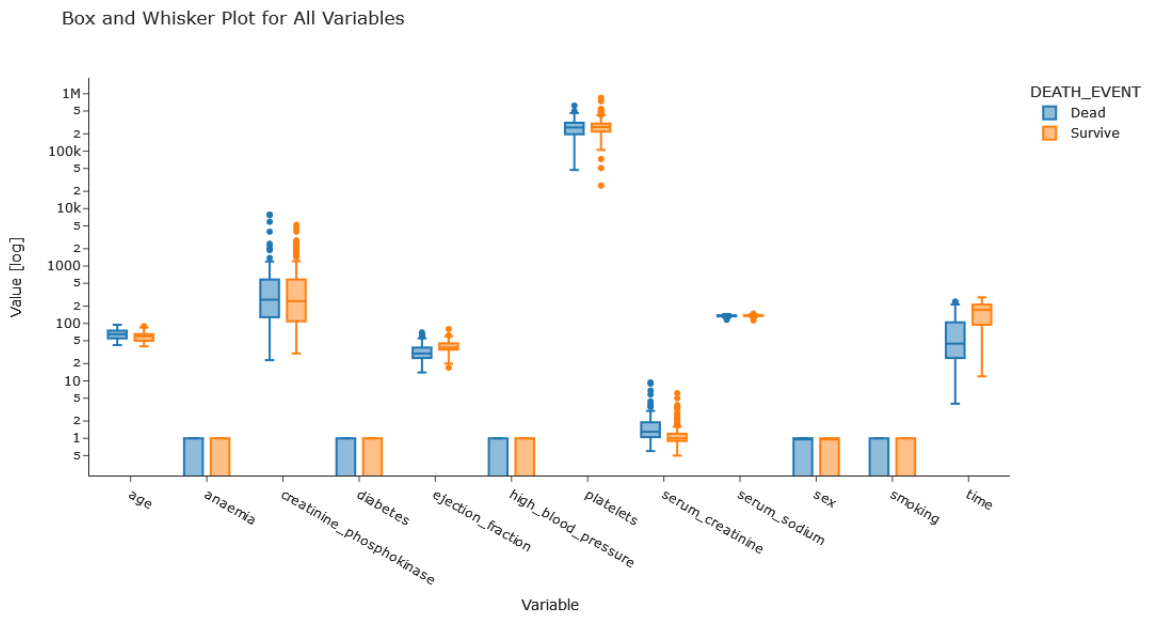

Figure 1: Boxplot of all variables available in the data with the y-axis on a log scale.

It can also be observed that there are several orders of magnitude between many of the continuous predictors, for instance "Platelets" is in the order of 1 x 10^5 whereas Serum Creatinine is in the order of single digits. Because of this it will be important to scale the data appropriately when applying any machine learning algorithm, for instance when applying Principal Component Analysis [PCA] or a Multi-Layer Perceptron [MLP] network.

Categorical predictors were examined using a Chi^2 statistical test to understand their influence on the target, with a threshold of $P < 0.05$. The results can be found in Table 2. All categorical predictors were found to be statistically insignificant in predicting the outcome of "Death Event", however "Anaemia" and "High Blood Pressure" were the most likely categorical predictors to not be able to be repeated by chance with P-Values of 0.31 and 0.21 respectively.

	Variable	Chi^2	P-Value
0	anaemia	1.042	0.307
1	diabetes	0.008	0.927
2	high_blood_pressure	1.543	0.214
3	sex	0.003	0.956
4	smoking	0.007	0.932

Table 2: Chi^2 Results Table for Categorical Predictors

All continuous variables were examined using a “pair plot” to examine the individual distributions of each variable and the correlation between each continuous variable. The dataset was again split based on outcome of “Dead” or “Survive” by way of colour. The resulting plot can be found in Figure 2.

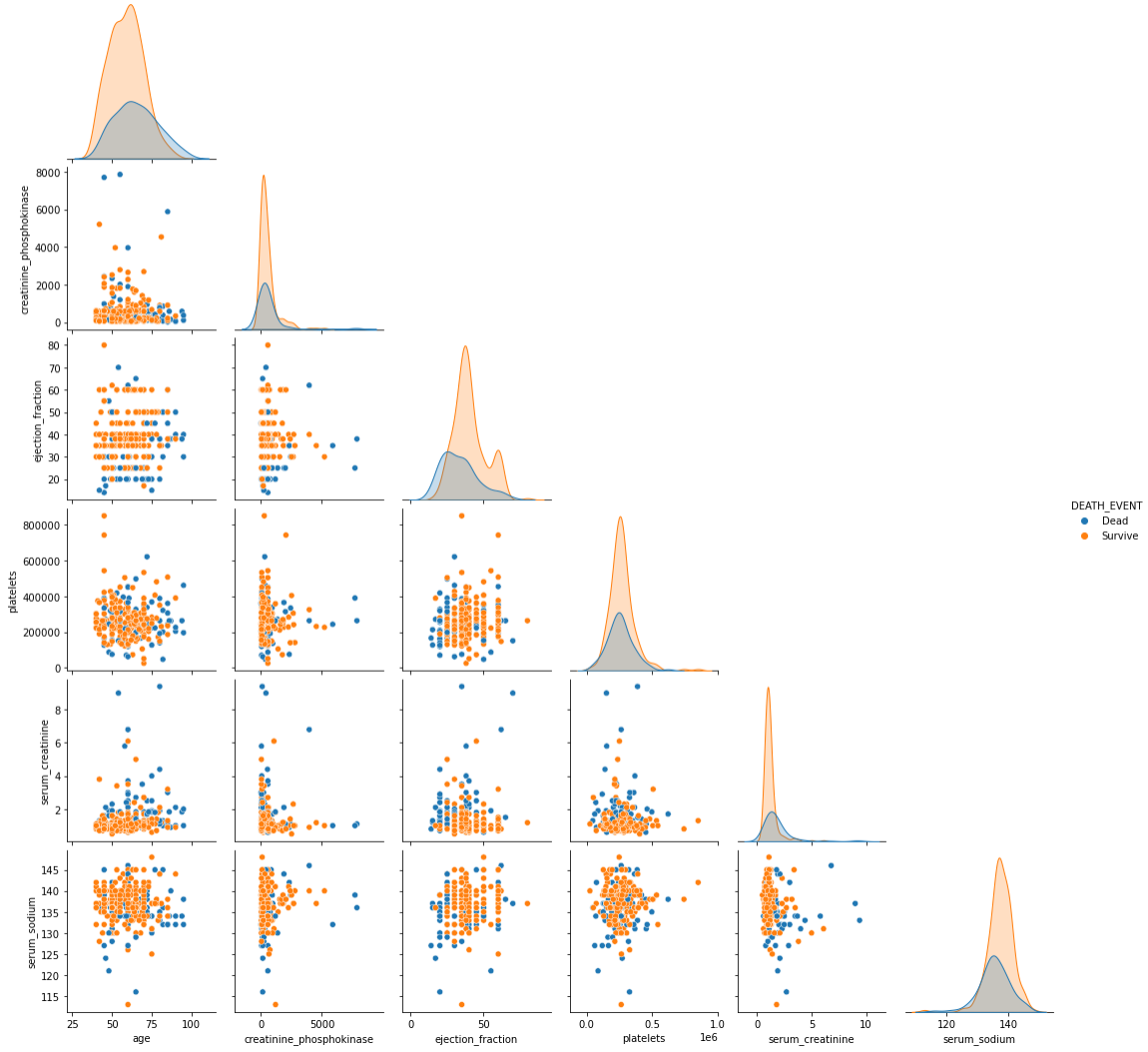

Figure 2: pair-plot of all continuous predictors.

The four key continuous predictors that stand out in pair-plot and boxplot, in terms of their univariate distribution, are Age, Ejection Fraction, Serum Creatinine and Time. The remaining predictors have relatively similar distributions but different sample sizes, with the “Survived” sample being significantly higher than the “Dead” sample. This imbalance will need to be accounted for when developing the classification model.

The Kruskal-Wallis test was used, to take into account any variable sample distributions that might not be Gaussian, to determine if any of the continuous variables were statistically predictors of death event. The results can be observed in Table 3.

The Null Hypothesis H_0 being that the distributions between the two death event outcomes are the same, with the alternative H_1 being that they are not the same.

	Variable	Test Stat	P-Value	Hypothesis
0	age	14.178	0.000	Reject
1	creatinine_phosphokinase	0.166	0.684	Keep
2	ejection_fraction	24.523	0.000	Reject
3	platelets	0.636	0.425	Keep
4	serum_creatinine	40.935	0.000	Reject
5	serum_sodium	13.121	0.000	Reject
6	time	87.923	0.000	Reject

Table 3: Kruskal-Wallis Results Table for Continuous Predictors

Examining the statistical summary values for the continuous variables in Table 3 it can be observed that the Age, Ejection Fraction, Serum Creatinine, Serum Sodium and Time are able to reject the Null Hypothesis and are correlated to the death event outcome.

Ranking all the continuous and categorical variables, based on their P-Value for their respective statistical tests against Death Event, provides Table 4. There is an obvious and clean break point between variables that should absolutely be considered in the predictive model and those that should initially be dropped; all variables with a P-Value of 0.000 will be included in the initial model.

Priority	Variable	Type	P-Value
1	Time	Continuous	0.000
2	Serum Creatinine	Continuous	0.000
3	Ejection Fraction	Continuous	0.000
4	Age	Continuous	0.000
5	Serum Sodium	Continuous	0.000
6	High Blood Pressure	Categorical	0.214
7	Anaemia	Categorical	0.307
8	Platelets	Continuous	0.425
9	Creatinine Phosphokinase	Continuous	0.684
10	Diabetes	Categorical	0.927
11	Smoking	Categorical	0.932
12	Sex	Categorical	0.956

Table 4: Priority Ranking of All Variables based on P-Value for predicting death event.

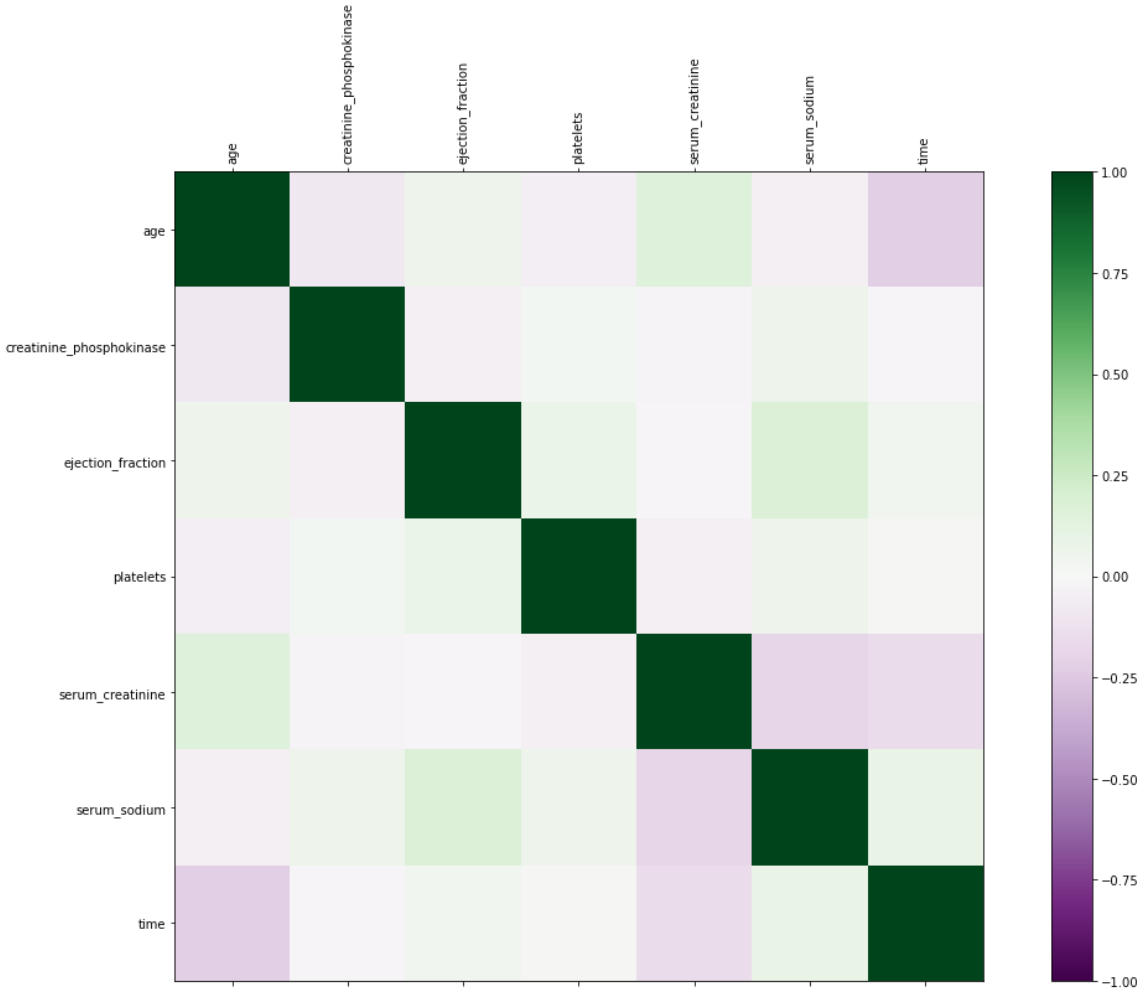

Figure 3: Heatmap of correlations between continuous variables.

A heatmap of correlations between the different continuous predictor variables, can be found under Figure 3. This plot indicates a generally low level of correlation between the variables, with most correlations between variables being within +/- 0.3. This agreed with the scatter plots between each variable combination on the pair plot in Figure 2.

This suggests the information available to the machine learning algorithm, in this case an ANN, will be evenly distributed across the continuous variables offering little entitlement for dimensionality reduction through Principal Component Analysis [PCA].

Question 1B

Principal Component Analysis [PCA] was conducted to better understand the covariance in the continuous variable and to understand if the dimensionality of the data could be reduced. The PCA algorithm from Scikit-Learn was applied in Python.

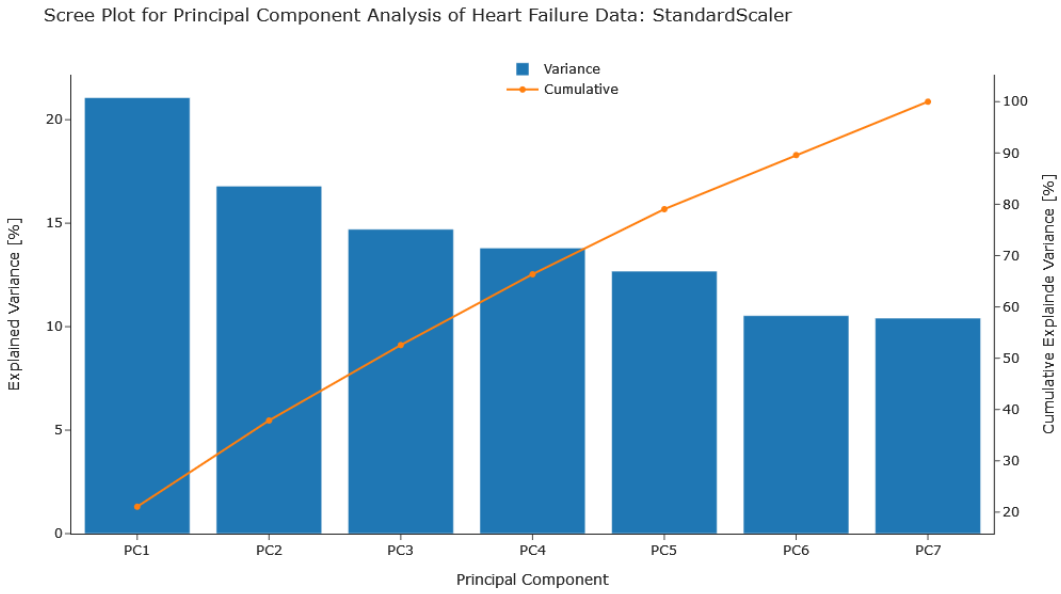

Figure 5: Scree plot of Principal Component Analysis of Heart Failure: Standard Scaler

Machine learning models can be very sensitive to scaling, with a pre-requisite for PCA of data being scaled around a mean of zero with unit variance. Hence, the Standard Scaler was implemented from Scikit-Learn. It was fitted to and then used to transform the continuous variables within the dataset.

The scree plot in figure 5 illustrates that the variance within the continuous predictors in the heart failure dataset is evenly distributed and cannot be reduced in dimensionality without losing at least 10% of the variance [PC7] when using the Standard Scaler. This makes sense as the heatmap of correlations between the continuous variables suggested only weak correlations.

When examining the interaction plots between the principal components, it can be observed that there is significant overlap between the two target classes of "dead" and "survive"; hence all the continuous variable information will be required to inform the model.

The conclusion is that the dataset does not allow for dimensionality reduction of continuous variables due to the variance being evenly distributed throughout the dataset and across all variables. To reduce the dimensionality of the dataset could likely harm the model more than help it. However, the reduced variable set based on statistical analysis in part A will initially be used and then variables added in if required.

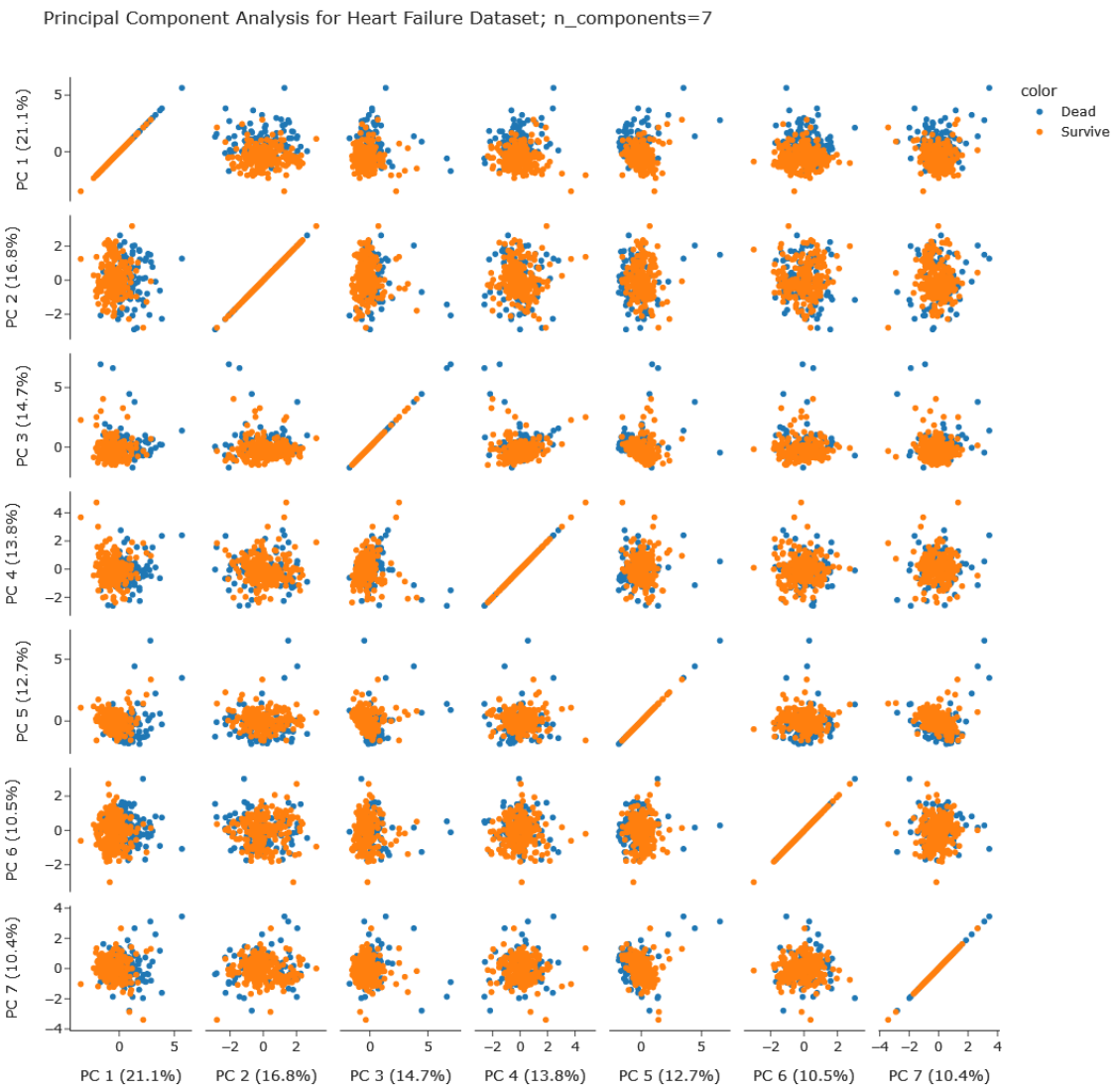

Figure 6: Interaction plots between Principal Components

Question 2A

A Multi-Layer Perception Classification model was built with two hidden layers of 5 and 3, as specified, and all other hyper-parameters set to their default using Scikit Learn. For reproducibility purposes the "random state" was set to 42.

The data set was split into training and test dataset samples with a default split of 75% training data and 25% for testing, again the "random state" was set to 42 for reproducibility.

The continuous training and test predictors were then also scaled using the Standard Scaler, as discussed in question 1B, with the scaler fitted to the training data sample. Once scaled, the categorical variables were joined to the same data-frame.

The "Initial" model, with only five variables, produced a training score of 0.875 and a test score of 0.693, with the test score being the accuracy of the model [True Negative + True Positive / Total] as the MLP classifier correctly predicted 52 cases from 75 test cases. The disparity between the training score and the test score suggests the model is over-fitting to the training data. This is a common challenge with MLP type models and needs to be managed to produce a model that generalizes better, that is a model that has a similar accuracy on training data as it does with the test data. This will always be hard to achieve, statistically speaking, when the test sample size is smaller than the training sample size as an incorrect prediction on a smaller sample size is a larger proportion of the sample. Figure 7 illustrates the confusion matrix for the test data set.

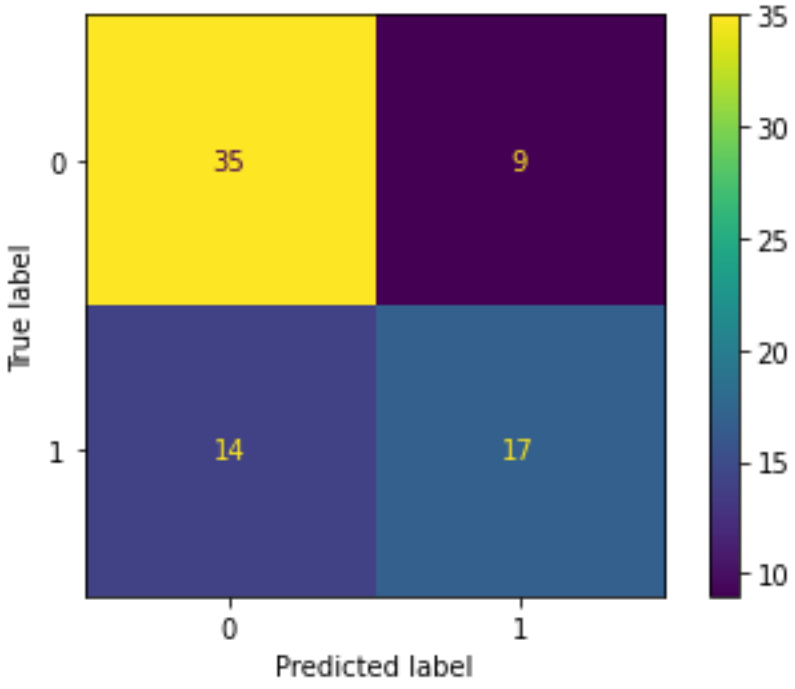

Figure 7: Confusion Matrix of initial MLP Classifier.

The model had a Precision score of 0.548, as defined by Equation 1, a Recall score of 0.658, as defined by Equation 2 and a Specificity score of 0.714 as defined by Equation 3.

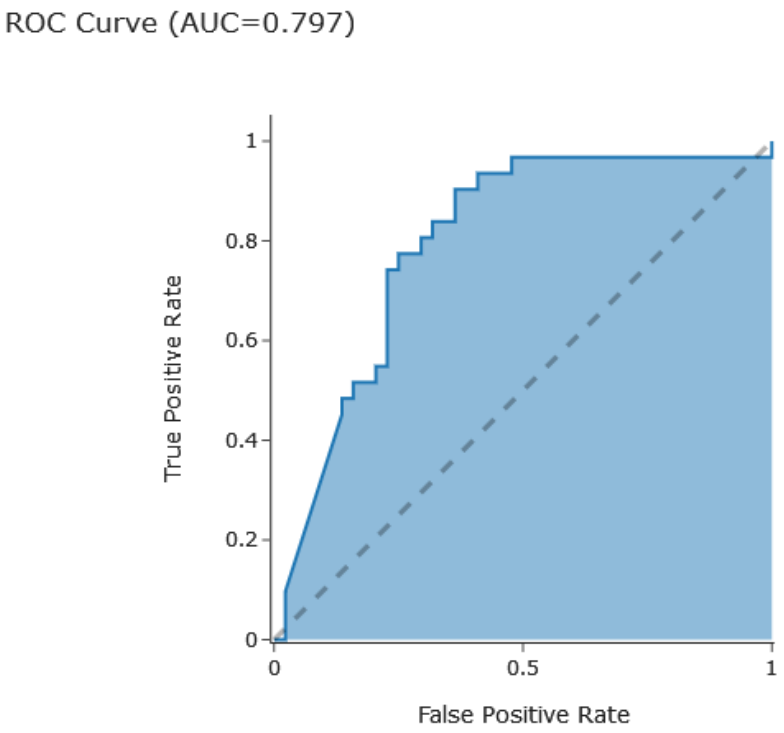

Figure 8: Receiver Operating Characteristic [ROC] curve for MLP classifier.

Based on the ROC curve I would conclude that this model is suitable for use but does have some significant room for improvement. The model is effectively correct more often that it is wrong. As the purpose of the model is to try identifying and predict patients that might be at risk of heart-failure, the model is able to do this better than if it was not employed.

However, there are risks inherent with False Positives and False Negatives. The precision score of 0.613 is the crucial score for this model; a precision of 1 would suggest the model can identify all patients at risk of heart failure within the test data set. The hyper-parameters could be tuned in this model to provide a better generalising and more precise model and will be explored in part B.

A MLP classifier was then run using all available data for training, with the same default hyper-parameter settings and the same number of hidden layers and neurons. There was a significant improvement in nearly all key metrics, except for AUC which reduced to 0.786.

This would suggest that the PCA was correct in indicating that the covariance of the continuous variables was evenly distributed and for the highest accuracy model the whole dataset should be used for model training.

Question 2B

A grid-search with cross-validation was run using the MLP classifier algorithm to determine if a more accurate model that generalized better could be developed. Six key hyper-parameters were tuned which would help improve both accuracy but also control the algorithm to potentially generalize better on previously unseen data. The hyper-parameter ranges examined are detailed in Table 5.

Hyper-parameter	Values
Solver	Adam, SGD, LBFGS
Learning Rate	Constant, Invscaling, Adaptive
Learning Rate Init	1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001
Activation Function	Identify, Logistic, Tanh, ReLu
Alpha	0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001
Hidden Layer Sizes	Single: 3, 4, 5, 6, 7, 8 Double: (3,3), (4,3), (5,3), (6,3), (7,3), (8,3) Double: (3,2), (4,2), (5,2), (6,2), (7,2), (8,2) Triple: (8,5,3), (8,6,3), (8,7,3), (8,8,3)

Table 5: Hyper-parameter range for grid-search of MLP classifier.

With all combinations available, this provided 38,808 different models to train across with the training data also being run four times in four different splits for each set of hyper-parameters, with only three portions used for training and the fourth held out for validation [known as cross-validation]. While this may seem a significant amount of modelling work, for a relatively simple network of single digit perceptron counts and only up to three layers – this only took less than 30 minutes on an 8-core machine. The scoring criteria used was "precision".

Figure 9: Scatter Plot of Mean Test Score vs. Standard Deviation of Test Score.

The cross-validation scores were plotted on a scatter plot [figure 9] of mean test score against the standard deviation of the test score [each for the 4 folds in the cross-validation]. This allows easy examination of the highest mean scores [which is automatically selected as the "best score" by the cross-validation routine against the hyper-parameter ranges with the lowest variation as this would likely help identify the hyper-parameter settings [Table 6] that would provide a model that generalised well to previously unseen data.

Parameter	Initial Model	Full Model	Final Model
Number of Variables	5	12	5
Solver	Adam	Adam	LBFGS
Learning Rate	Constant	Constant	Constant
Learning Rate Init	0.001	0.001	N/A
Activation Function	Relu	Relu	Logistic
Alpha	0.0001	0.0001	0.0001
Hidden Layer Sizes	5, 3	5, 3	4

Table 6: Hyper-parameters for initial and tuned model.

Applying these settings made some minor improvements to the model on all metrics. The ROC Area Under the Curve [AUC] is now 0.822 versus the original 0.797. Both the training and test accuracy increased slightly, however the model is still significantly over fitted to the training data. The precision has improved marginally but still not to the level of the default hyper-parameter settings when paired with the full data set of all 12 variables.

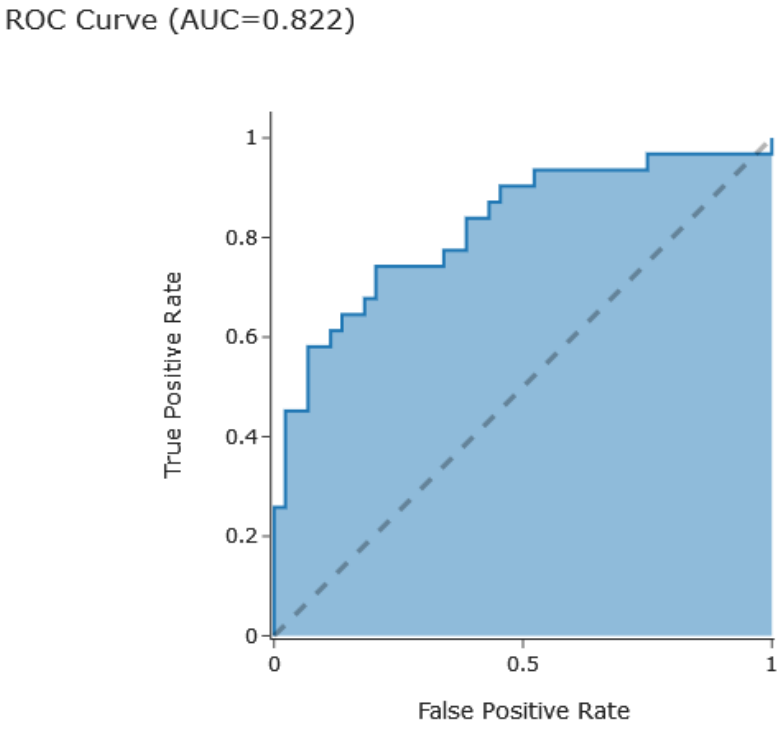

Figure 10: ROC curve for tuned MLP classifier.

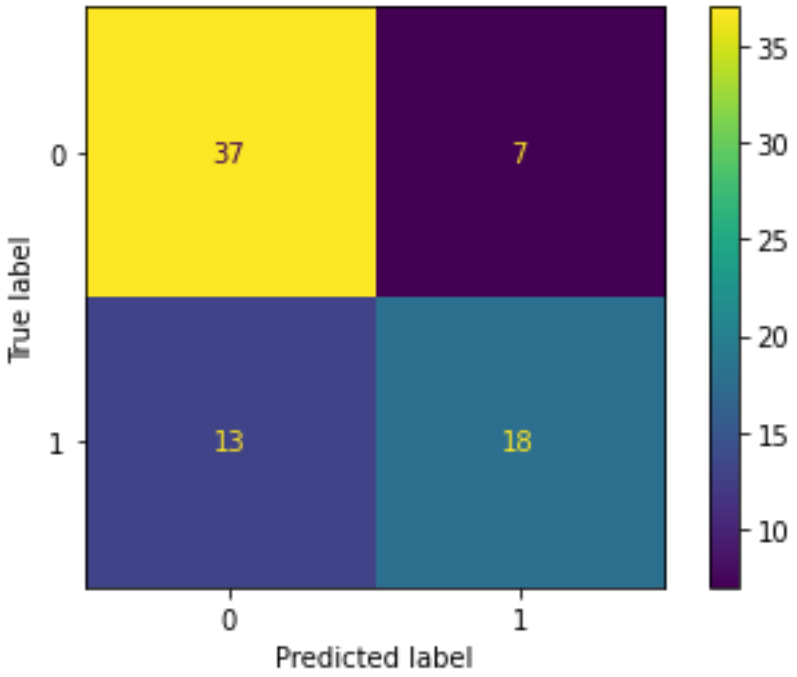

Figure 11: Confusion Matrix for tuned model.

All metrics for the initial and tuned model can be observed in Table 6, along with the confusion matrix for the tuned model in figure 11.

Metric	Initial Model	Full Model	Final Model
Training Accuracy	0.875	0.951	0.929
Testing Accuracy	0.693	0.773	0.733
AUC	0.797	0.786	0.822
Precision	0.548	0.613	0.581
Recall	0.654	0.792	0.720
Specificity	0.714	0.765	0.740

Table 6: Metrics of model improvements with tuning.

As can be seen in the confusion matrix, the accuracy of the model has marginally improved with the number of True Negatives increasing by 2 and True Positives also increasing by 1. This combined with a reduction by 1 in False Positives explains why the precision has increased. Finally, the False Negatives has decreased by 2 which explains the increase in recall.

I conclude that this tuned model is distinctly better than the initial model when both using the same five predictors, with the mode simple model of only 4 neurons in the hidden layer being the highest scoring. However, the best model on nearly all metrics was the default base MLP classifier using all 12 predictor variables and would recommend that any further work look to find the right balance between a minimal set of variables and the whole dataset.

Literature review

The dataset studied here has previously been used with an array of machine learning models including tree-based, Support Vector Machines [SVM], ANN's and Naïve Bayes (Chicco & Jurman, 2020) where it was found that tree-based methods, specifically Random Forests had the highest accuracy. It was also shown that while ANN's provided reasonable precision, Random Forests were significantly more powerful. This agrees with my conclusions, that the ANN employed here can do a reasonably good job of identifying patients at risk – but still leaves room for improvement. Interestingly, the same report illustrated that Random Forests were capable of accurately predicting the patient outcome from one factor alone: Ejection Fraction.

Survival analysis was conducted in the case of (Ahmad et al., 2017) which agreed with my conclusions that Sex was not a strong predictor and that Ejection Fraction would be. It was shown that both Male and Female patients had the same survival rate, however patients with an Ejection Fraction below 30 had a significantly lower survival rate. This also agrees well with the study completed by (Chicco & Jurman, 2020) which built a Random Forest model based solely on Ejection Fraction.

In the instance of (Heckerling et al., 2003) ANN's were used to predict community-acquired pneumonia and experimented with the number of hidden layers within the network whilst monitoring the AUC. It was found that moving from zero hidden layers to one hidden layer, the test sample AUC increased from 0.808 to 0.868, but then reduced when moving to two hidden layers. However, the training sample marginally increased going from one hidden layer to two hidden layers by 0.005. This suggests the model is starting to overfit and the model with one hidden layer was likely optimal. It was noted that all three models classified with a similar level of accuracy. As a calibration for my model, a test sample AUC for the pneumonia model of 0.868 and 0.822 for my heart failure model, suggests that my model is likely performing quite well.

A slightly different implementation of an ANN was in the instance of (Ma et al., 2020), where images of kidney ultrasounds were used to classify patients with chronic kidney disease. A large portion of the implementation was on reducing the input image from noise and identifying the key features ahead of the actual classification algorithm. It should also be noted that the dataset was significantly smaller than the dataset employed in the heart failure model, however they were able to achieve an AUC of 0.922 with their implementation. The key take-away from this piece of work was the criticality of the feature extraction and dimensionality reduction to achieve a more precise classification model.

References

Ahmad, T., Munir, A., Bhatti, S. H., Aftab, M., & Raza, M. A. (2017). Survival analysis of heart failure patients: A case study. *PLOS ONE, 12*(7), e0181001. https://doi.org/10.1371/journal.pone.0181001

Chicco, D., & Jurman, G. (2020, 2020/02/03). Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone. *BMC Medical Informatics and Decision Making, 20*(1), 16. https://doi.org/10.1186/s12911-020-1023-5

Heckerling, P. S., Gerber, B. S., Tape, T. G., & Wigton, R. S. (2003). Prediction of community-acquired pneumonia using artifical neural networks. *Medical decision making, 23*(2), 112. https://go.exlibris.link/T117qFoN

Ma, F., Sun, T., Liu, L., & Jing, H. (2020). Detection and diagnosis of chronic kidney disease using deep learning-based heterogeneous modified artificial neural network. *Future generation computer systems, 111*, 17-26. https://doi.org/10.1016/j.future.2020.04.036

Appendix

Appendix 1: Equations

Equation 1:

$$Precision = \frac{\sum True\ Positives}{\sum True\ Positives + \sum False\ Positives}$$

Equation 2:

$$Recall\ = \frac{\sum True\ Positives}{\sum True\ Positives + \sum False\ Negatives}$$

Equation 3:

$$Specificity = \frac{\sum True\ Negatives}{\sum True\ Negatives + \sum False\ Positives}$$

Appendix 2.1: Question 1A Code

```
#import required libaries for data exploration
import pandas as pd
import numpy as np
from scipy.stats import chi2_contingency
import scipy.stats as stats

import matplotlib.pyplot as plt
import seaborn as sns

import plotly.express as px
import plotly.io as pio
pio.templates.default="simple_white"


#read the dataset in and make a copy, substituting Death Event 0/1 for
Surveive/Dead
master_read = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learn
ing-databases/00519/heart_failure_clinical_records_dataset.csv")

master_copy=master_read.copy().replace({
    "DEATH_EVENT":{0:"Survive",
                   1:"Dead"}
})
```

```
#Examine the variables for data type and missing counts
master_copy.info()



#create a boxplot of all variables
fig1 = px.box(master_copy,
              log_y=True,
              title="Box and Whisker Plot for All Variables",
              width=1080,
              height=610,
              color="DEATH_EVENT")

fig1.update_xaxes(title="Variable")
fig1.update_yaxes(title="Value [log]")



#make a list of categorical variables for analysis
categorical_list = ["anaemia","diabetes","high_blood_pressure","sex","s
moking"]

#make a list of continuous variables for analysis
continuous_list = ["age","creatinine_phosphokinase","ejection_fraction"
,"platelets",
                            "serum_creatinine","serum_sodium","DEATH_E
VENT","time"]



#create a table of Chi Squared statistical analysis for categorical var
iables
chi_sq_table=pd.DataFrame(columns=["Variable","Chi^2","P-Value"])

for c in categorical_list:
    cross_tab = pd.crosstab([master_copy.DEATH_EVENT],
                            [master_copy[c]],
                            margins=False,
                            normalize=False)
    cross_tab=np.array(cross_tab)

    chi2, p_val, dof, expected = chi2_contingency(cross_tab)

    chi_sq_table=chi_sq_table.append({
        "Variable":c,
        "Chi^2":round(chi2,3),
        "P-Value":round(p_val,3)
    },ignore_index=True)
```

```
chi_sq_table



#create a pair plot for all continuous variables, split by death event
sns.pairplot(master_copy[["age","creatinine_phosphokinase","ejection_fr
action","platelets",
                              "serum_creatinine","serum_sodium","DEATH_E
VENT"]],
             hue="DEATH_EVENT",
             corner=True)



#complete the Kruskal-Wallis test on continuous variables for DEATH_EVE
NT

continuous_variables=["age","creatinine_phosphokinase","ejection_fracti
on","platelets",
                              "serum_creatinine","serum_sodium","time"]

kruskal_results_df=pd.DataFrame(columns=["Variable","Test Stat","P-Valu
e","Hypothesis"])

for i in continuous_variables:
    dead_df=master_read[master_read["DEATH_EVENT"]==1]
    survive_df=master_read[master_read["DEATH_EVENT"]==0]

    dead_df=dead_df[[i]]
    survive_df=survive_df[[i]]

    test_stat, p_value = stats.kruskal(dead_df,survive_df)

    if p_value > 0.05:
        hyp="Keep"
    else:
        hyp="Reject"

    kruskal_results_df=kruskal_results_df.append({
        "Variable":i,
        "Test Stat":round(test_stat,3),
        "P-Value":round(p_value,3),
        "Hypothesis":hyp
    },ignore_index=True)

kruskal_results_df
```

```
#define a function for creating a heatmap of correlation values
def matrix_plot(data,plot_range=0.3):

    fig = plt.figure(figsize=(20,12))
    ax=fig.add_subplot(111)
    cax=ax.matshow(data,cmap="PRGn",vmin=-plot_range,vmax=plot_range)
    fig.colorbar(cax)

    ticks=np.arange(0,len(data.columns),1)

    ax.set_xticks(ticks)
    plt.xticks(rotation=90)
    ax.set_xticklabels(data.columns)

    ax.set_yticks(ticks)
    ax.set_yticklabels(data.columns)

    plt.show()


#create a new dataframe of correlation values between continuous predic
tors
correlation_df = master_copy[continuous_list].corr()


#create a matrix plot of correlation values for continuous predictors
matrix_plot(correlation_df,plot_range=1)
```

Appendix 2.2: Question 1B Code

```
#import additional libraries for completing PCA
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

import plotly.graph_objects as go
from plotly.subplots import make_subplots



#fit a StandardScaler to the continuous predictors and then transform t
he variables to the fitted scaler
cont_var_df=master_read[continuous_list]

cont_var_scaled=StandardScaler().fit_transform(cont_var_df)



#Complete PCA for with seven components [for seven continuous predictor
s] to understand the variance in each Principal Component
pca=PCA(n_components=7)
cont_var_pca=pca.fit_transform(cont_var_scaled)

pca_results_df=pd.DataFrame(pca.explained_variance_ratio_,columns=["Var
iance"])
pca_results_df["Cumulative"]=pca_results_df.cumsum()
pca_results_df=pca_results_df*100
pca_results_df["PC"]=["PC1","PC2","PC3","PC4","PC5","PC6","PC7"]

labels = {
    str(i):f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_*100)
}



#create a Scree Plot of the resuling PCA
trace1=go.Bar(
    x=pca_results_df["PC"],
    y=pca_results_df["Variance"],
    name="Variance"
)

trace2=go.Scatter(
    x=pca_results_df["PC"],
    y=pca_results_df["Cumulative"],
    name="Cumulative"
)
```

```
fig3 = make_subplots(specs=[[{"secondary_y":True}]])
fig3.add_trace(trace1)
fig3.add_trace(trace2,secondary_y=True)
fig3["layout"].update(
    height=610,
    width=1080,
    title="Scree Plot for Principal Component Analysis of Heart Failure
 Data: RobustScaler",
    legend=dict(yanchor="middle",
                y=0.99,
                xanchor="center",
                x=0.5)
)
fig3.update_xaxes(title="Principal Component")
fig3.update_yaxes(title="Explained Variance [%]",secondary_y=False)
fig3.update_yaxes(title="Cumulative Explainde Variance [%]",secondary_y
=True)

fig3.show()



#define a function to create a matrix of scatterplots for each Principa
l Component.
def pca_examiner(n_components):
    pca=PCA(n_components=n_components)
    cont_var_pca=pca.fit_transform(cont_var_scaled)

    labels = {
        str(i):f"PC {i+1} ({var:.1f}%)"
        for i, var in enumerate(pca.explained_variance_ratio_*100)
    }

    fig2 = px.scatter_matrix(
        cont_var_pca,
        width=1080,
        height=1080,
        labels=labels,
        dimensions=range(n_components),
        color=master_copy.DEATH_EVENT,
        title=f"Principal Component Analysis for Heart Failure Dataset;
 n_components={n_components}"
    )

    fig2.show()
```

Appendix 2.3: Question 2A Code

```
#import additional libraries for the ANN
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, r
oc_curve, auc
import time


#create a master dataframe of all variables except the target vector
X_master = master_read.drop(["DEATH_EVENT"],axis=1)


#create a master target vector
y_master = master_read.DEATH_EVENT


#split the master frames into training and test splits
X_train, X_test, y_train, y_test = train_test_split(X_master,y_master,r
andom_state=42)


#split out the categorical and continuous variables in the training dat
a
X_train_cat=X_train[["anaemia","diabetes","high_blood_pressure","sex","
smoking"]]
X_train_cont=X_train[["age","creatinine_phosphokinase","ejection_fracti
on","platelets",
                      "serum_creatinine","serum_sodium","time"]]

#fit a standard scaler on the continuous variables in the training data
frame
scaler=StandardScaler().fit(X_train_cont)

#transform the continuous variables in the training data
X_train_scaled=scaler.transform(X_train_cont)

#convert the scaled data in a numpy nd-array back into a dataframe
X_train_scaled=pd.DataFrame(X_train_scaled,columns=X_train_cont.columns
,index=X_train_cont.index)

#join the categorical variables back to the scaled dataframe
X_train_scaled=X_train_scaled.join(X_train_cat)

#seperate out the initial five variables of interest
```

```
X_train_scaled_five=X_train_scaled[["time","serum_creatinine","ejection
_fraction","age","serum_sodium"]]


#split out the categorical and continuous variables in the test data
X_test_cat=X_test[["anaemia","diabetes","high_blood_pressure","sex","sm
oking"]]
X_test_cont=X_test[["age","creatinine_phosphokinase","ejection_fraction
","platelets",
                           "serum_creatinine","serum_sodium","time"]]

#transform the continuous variables in the test data based on the train
ing data scaler
X_test_scaled=scaler.transform(X_test_cont)

#convert the scaled data in a numpy nd-array back into a dataframe
X_test_scaled=pd.DataFrame(X_test_scaled,columns=X_test_cont.columns,in
dex=X_test_cont.index)

#join the categorical variables back to the scaled dataframe
X_test_scaled=X_test_scaled.join(X_test_cat)

#seperate out the initial five variables of interest
X_test_scaled_five=X_test_scaled[["time","serum_creatinine","ejection_f
raction","age","serum_sodium"]]
```

Initial Model with Only Five Variables¶

```
#create the classifier instance with hidden layers 5 and 3 and all othe
r parameters as per default
mlp_class=MLPClassifier(hidden_layer_sizes=[5,3],
                            random_state=42,
                            max_iter=10000,
                            verbose=False,
                            )


#fit the model to the training data
mlp_class.fit(X_train_scaled_five,y_train)

#test the model agains the training and the test data
print(f"Training Score: {mlp_class.score(X_train_scaled_five,y_train):.
3f}")
print(f"Test Score: {mlp_class.score(X_test_scaled_five,y_test):.3f}")


#create a confusion matrix of the results
y_pred=mlp_class.predict(X_test_scaled_five)
```

```
mlp_class_conf_matr=confusion_matrix(y_test,y_pred)
mlp_class_conf_matr_display=ConfusionMatrixDisplay(mlp_class_conf_matr)
.plot()



#calculate the precision, recall and specificity of the model
precision=mlp_class_conf_matr[1,1]/(mlp_class_conf_matr[1,1]+mlp_class_
conf_matr[1,0])
print(f"Precision: {precision:.3f}")
recall=mlp_class_conf_matr[1,1]/(mlp_class_conf_matr[1,1]+mlp_class_con
f_matr[0,1])
print(f"Recall: {recall:.3f}")
specificity=mlp_class_conf_matr[0,0]/(mlp_class_conf_matr[0,0]+mlp_clas
s_conf_matr[1,0])
print(f"Specificity: {specificity:.3f}")



#create an ROC curve and calcuate the AUC value
y_score=mlp_class.predict_proba(X_test_scaled_five)[:,1]
fpr, tpr, thresholds = roc_curve(y_test,y_score)

fig4=px.area(
    x=fpr,
    y=tpr,
    title=f"ROC Curve (AUC={auc(fpr,tpr):.3f})",
    labels=dict(x="False Positive Rate",y="True Positive Rate"),
    width=700, height=500

)

fig4.add_shape(
    type="line",
    line=dict(dash="dash",color="black",width=3),
    x0=0,x1=1,y0=0,y1=1
)

fig4.update_yaxes(scaleanchor="x",scaleratio=1)
fig4.update_xaxes(constrain="domain")
fig4.show()
```

Second Model with All Variables¶

```
#create a new model and repeat the key indicators, using the full datas
et of 12 variables
mlp_class_all_var=MLPClassifier(hidden_layer_sizes=[5,3],
                                random_state=42,
                                max_iter=10000,
                                verbose=False,
```

```
                                )

mlp_class_all_var.fit(X_train_scaled,y_train)
print(f"Training Score: {mlp_class_all_var.score(X_train_scaled,y_train
):.3f}")
print(f"Test Score: {mlp_class_all_var.score(X_test_scaled,y_test):.3f}
")

y_pred_all_var=mlp_class_all_var.predict(X_test_scaled)
mlp_class_all_var_conf_matr=confusion_matrix(y_test,y_pred_all_var)
mlp_class__all_var_conf_matr_display=ConfusionMatrixDisplay(mlp_class_a
ll_var_conf_matr).plot()

precision=mlp_class_all_var_conf_matr[1,1]/(mlp_class_all_var_conf_matr
[1,1]+mlp_class_all_var_conf_matr[1,0])
print(f"Precision: {precision:.3f}")
recall=mlp_class_all_var_conf_matr[1,1]/(mlp_class_all_var_conf_matr[1,
1]+mlp_class_all_var_conf_matr[0,1])
print(f"Recall: {recall:.3f}")
specificity=mlp_class_all_var_conf_matr[0,0]/(mlp_class_all_var_conf_ma
tr[0,0]+mlp_class_all_var_conf_matr[1,0])
print(f"Specificity: {specificity:.3f}")



#create a ROC curve for the new model using all 12 variables
y_score_all_var=mlp_class_all_var.predict_proba(X_test_scaled)[:,1]
fpr, tpr, thresholds = roc_curve(y_test,y_score_all_var)

fig5=px.area(
    x=fpr,
    y=tpr,
    title=f"ROC Curve (AUC={auc(fpr,tpr):.3f})",
    labels=dict(x="False Positive Rate",y="True Positive Rate"),
    width=700, height=500
)

fig5.add_shape(
    type="line",
    line=dict(dash="dash",color="black",width=3),
    x0=0,x1=1,y0=0,y1=1
)

fig5.update_yaxes(scaleanchor="x",scaleratio=1)
fig5.update_xaxes(constrain="domain")
fig5.show()
```

Appendix 2.4: Question 2B Code

```
#import additional libraries for grid search and cross-validation
from sklearn.model_selection import GridSearchCV
```

Coarse Grid¶

```
#layout a relatively coarse grid of hyper-parameters to train across
hyper_param_grid={"solver":["adam","sgd","lbfgs"],
                  "learning_rate":["constant","invscaling","adaptive"],
                  "activation":["identify","logistic","tanh","relu"],
                  "learning_rate_init":[1,0.1,0.01,0.001,0.0001,0.00001,
0.000001],
                  "alpha":[0.1,0.01,0.001,0.0001,0.00001,0.000001,0.0000
001],
                  "hidden_layer_sizes":[3,4,5,6,7,8,
                                        (3,3),(4,3),(5,3),(6,3),(7,3),(8
,3),
                                        (3,2),(4,2),(5,2),(6,2),(7,2),(8
,2),
                                        (8,5,3), (8,6,3), (8,7,3), (8,8,
3)]
                  }


#create the base classified to process with the grid search
mlp_class = MLPClassifier(random_state=42,
                          max_iter=10000,
                          verbose=False
                         )


#configure the grid-search with cross-validation of 4 folds and using 8
 processors
#and precision as the scoring criteria
mlp_grid_search_cv=GridSearchCV(mlp_class,
                                hyper_param_grid,
                                cv=4,
                                n_jobs=8,
                                scoring="precision")


#complete the grd-search and cross-validation; recording the time taken
.
t0=time.time()
mlp_grid_search_cv.fit(X_train_scaled_five,y_train)
t1=time.time()
print(f"Total time taken: {t1-t0:.2f}s")
```

```
#convert the Grid Search Results into  a DataFrame
results=pd.DataFrame(mlp_grid_search_cv.cv_results_)


#count the number of hyper-parameter combinations examined
len(results.index)


#plot the Mean vs. the Standard Deviation of the Test Score [Precision]
fig5 = px.scatter(results,
                  x="std_test_score",
                  y="mean_test_score")

fig5.update_xaxes(title="Test Score: Standard Deviation")
fig5.update_yaxes(title="Test Score: Mean")


#Filter the results down based on the scatter plot
results_filter=results[results["std_test_score"]<0.05]
results_filter[results_filter["mean_test_score"]>0.80]


#print out the best cross-validation score
print(f"Best Cross Validation Score: {mlp_grid_search_cv.best_score_:.3
f}")
```

Final Model¶

```
#create the final model and it's assocaited metrics based on the GridSe
archCV
mlp_class_final=MLPClassifier(hidden_layer_sizes=[4],
                              activation="logistic",
                              alpha=0.0001,
                              learning_rate="constant",
                              learning_rate_init=0.1,
                              solver="lbfgs",
                              random_state=42,
                              max_iter=10000,
                              verbose=False,
                             )

mlp_class_final.fit(X_train_scaled_five,y_train)
print(f"Training Score: {mlp_class_final.score(X_train_scaled_five,y_tr
ain):.3f}")
print(f"Test Score: {mlp_class_final.score(X_test_scaled_five,y_test):.
```

```
3f}")

y_pred_final=mlp_class_final.predict(X_test_scaled_five)
mlp_class_final_conf_matr=confusion_matrix(y_test,y_pred_final)
mlp_class__final_conf_matr_display=ConfusionMatrixDisplay(mlp_class_fin
al_conf_matr).plot()

precision=mlp_class_final_conf_matr[1,1]/(mlp_class_final_conf_matr[1,1
]+mlp_class_final_conf_matr[1,0])
print(f"Precision: {precision:.3f}")
recall=mlp_class_final_conf_matr[1,1]/(mlp_class_final_conf_matr[1,1]+m
lp_class_final_conf_matr[0,1])
print(f"Recall: {recall:.3f}")
specificity=mlp_class_final_conf_matr[0,0]/(mlp_class_final_conf_matr[0
,0]+mlp_class_final_conf_matr[1,0])
print(f"Specificity: {specificity:.3f}")



#create the ROC curve for the final model
y_score_final=mlp_class_final.predict_proba(X_test_scaled_five)[:,1]
fpr, tpr, thresholds = roc_curve(y_test,y_score_final)

fig5=px.area(
    x=fpr,
    y=tpr,
    title=f"ROC Curve (AUC={auc(fpr,tpr):.3f})",
    labels=dict(x="False Positive Rate",y="True Positive Rate"),
    width=700, height=500
)

fig5.add_shape(
    type="line",
    line=dict(dash="dash",color="black",width=3),
    x0=0,x1=1,y0=0,y1=1
)

fig5.update_yaxes(scaleanchor="x",scaleratio=1)
fig5.update_xaxes(constrain="domain")
fig5.show()
```