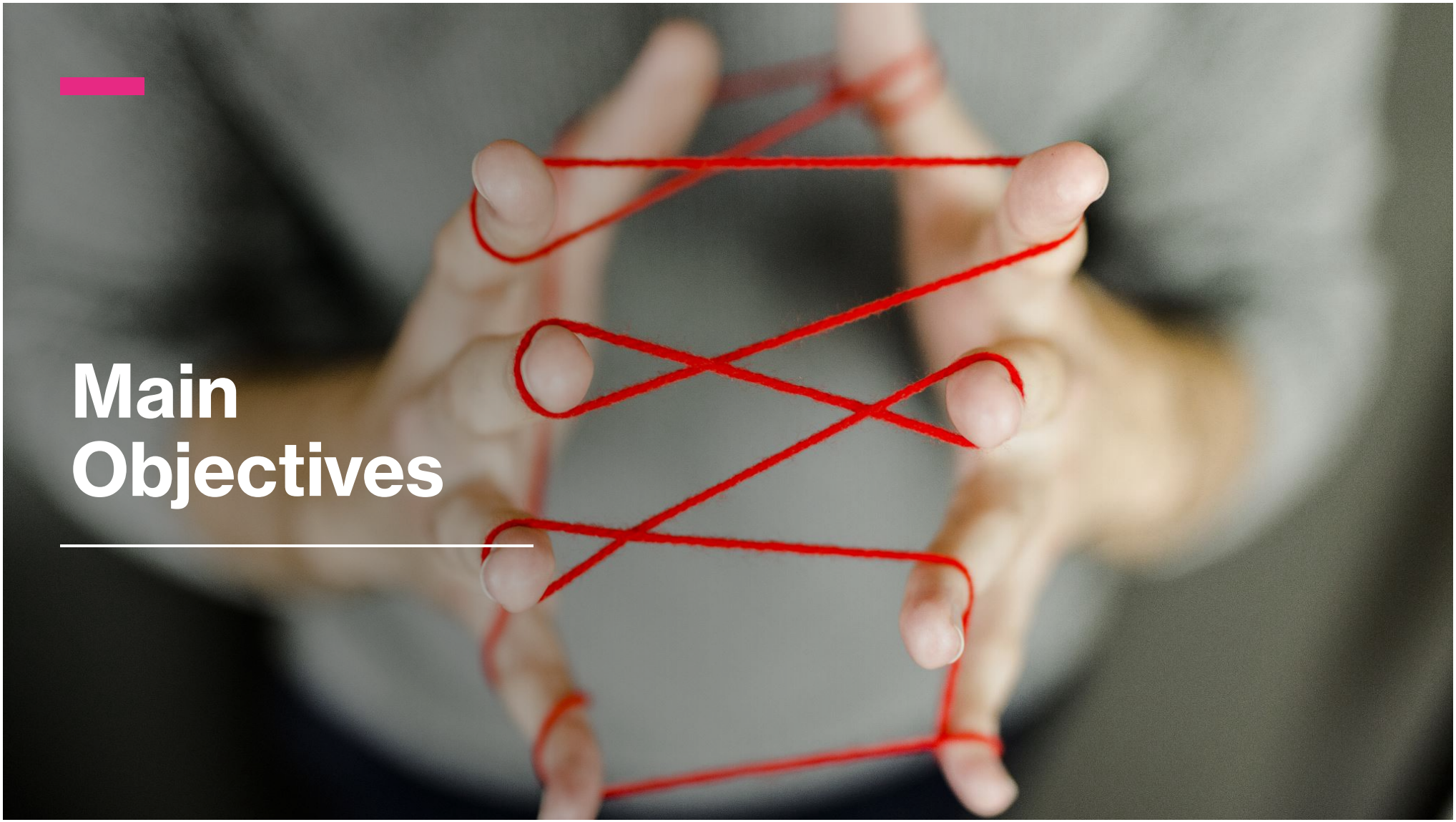
Heart Failure Prediction

Ben Cottrell


Main
Objectives

Cardiovascular diseases rank at no 1 for cause of death across the world, estimated at a total of 17.9 million lives each year which accounts for 31% of all deaths worldwide

Heart failure is a common event caused by CVDs and this dataset contains 12 features that can be used to predict mortality by heart failure. Most cardiovascular diseases can be prevented by addressing behavioural risk factors such as tobacco use, unhealthy diet and obesity, physical inactivity and harmful use of alcohol using population-wide strategies.

People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or already established disease) need **early detection** and management wherein a machine learning model can be of great help.


Dataset

Dataset Overview

Dataset from Davide Chicco, Giuseppe Jurman: Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone. BMC Medical Informatics and Decision Making 20, 16 (2020).

Dataset contains 12 clinical features for predicting death events, with 'DEATH_EVENT' being the "target".

Source - https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data/data

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4	1
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6	1
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7	1
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7	1
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8	1

Dataset Description

```
[42]: age                         float64
      anaemia                       int64
      creatinine_phosphokinase      int64
      diabetes                      int64
      ejection_fraction             int64
      high_blood_pressure           int64
      platelets                   float64
      serum_creatinine            float64
      serum_sodium                  int64
      sex                           int64
      smoking                       int64
      time                          int64
      DEATH_EVENT                   int64
      dtype: object
```

Features

Age – Age of the patient

Anaemia – Decrease of red blood cells or hemoglobin

Creatinine Phosphokinase – Level of the CPK enzyme in the blood

Diabetes – If the patient has diabetes

Ejection Fraction – Percentage of blood leaving the heart at each contraction

High Blood Pressure – If the patient has hypertension

Platelets – Platelets in the blood (kiloplatelets/mL)

Serum Creatinine – Level of serum creatinine in the blood (mg/dL)

Serum Sodium – Level of serum sodium in the blood (mEq/L)

Sex – Women or Man (binary)

Smoking – If the patient smokes or not

Time – Follow up period (days)

Target

DEATH_EVENT – If the patient deceased during the follow-up period

Checking for NULL values

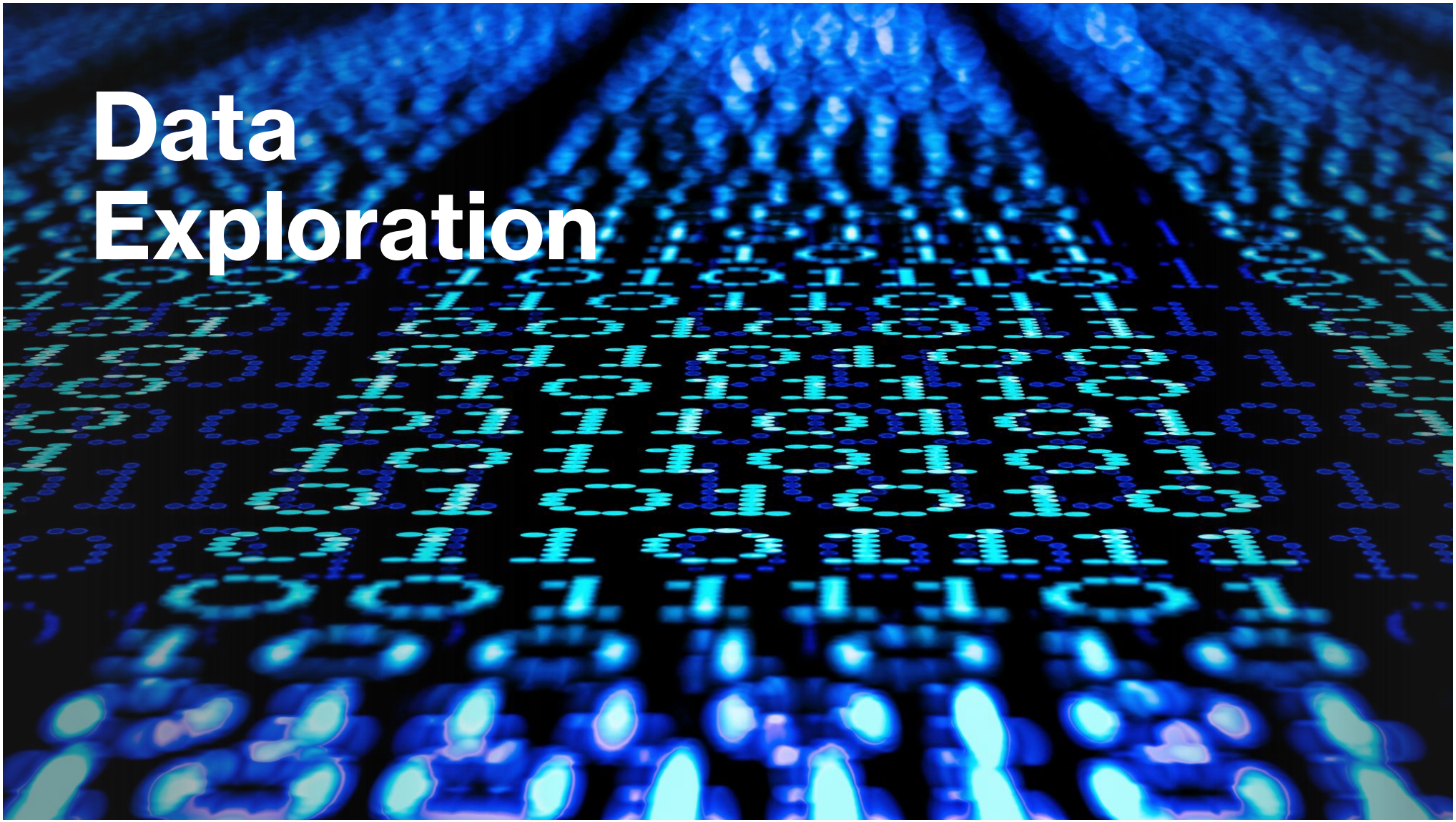
```
[40]: data.isna().sum()
```

```
[40]: age                         0
      anaemia                     0
      creatinine_phosphokinase    0
      diabetes                    0
      ejection_fraction           0
      high_blood_pressure         0
      platelets                   0
      serum_creatinine            0
      serum_sodium                0
      sex                         0
      smoking                     0
      time                        0
      DEATH_EVENT                 0
      dtype: int64
```

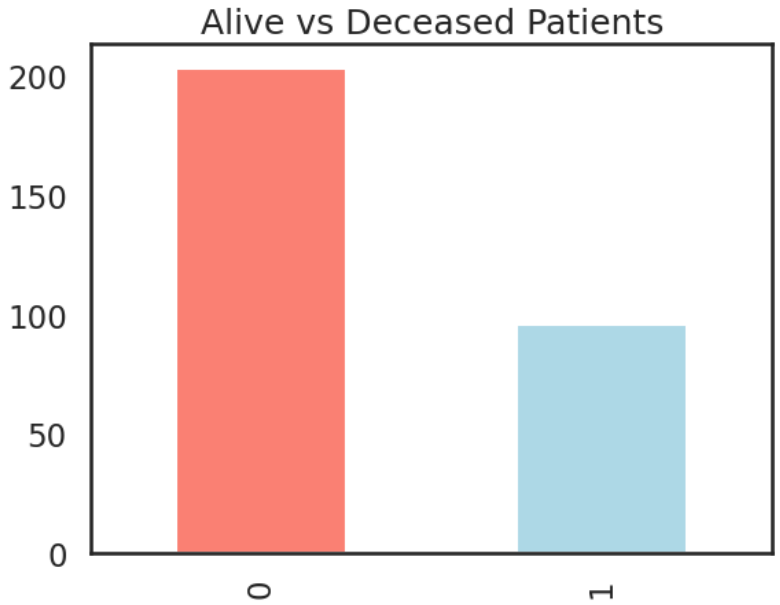
Data Cleanup

[7]:
```
#rename DEATH_EVENT column to keep consistency with other columns
data.rename(columns={'DEATH_EVENT': 'death_event'}, inplace=True)
```

[44]:
```
age                         float64
anaemia                       int64
creatinine_phosphokinase      int64
diabetes                      int64
ejection_fraction             int64
high_blood_pressure           int64
platelets                   float64
serum_creatinine            float64
serum_sodium                  int64
sex                           int64
smoking                       int64
time                          int64
death_event                   int64
dtype: object
```


Data
Exploration

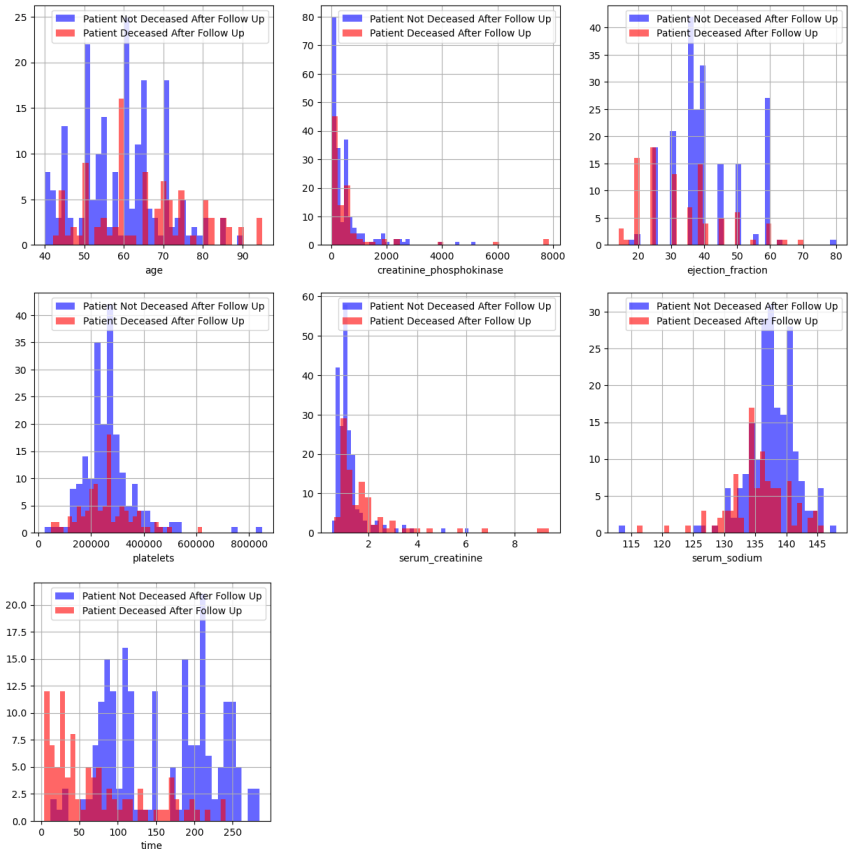
Viewing status of people in the data set



The chart on the left shows the number of patients who had heart failure and survived (value = 0) vs the number of patients who died after heart failure (value = 1).

Of the 299 patients in the data set, 203 patients were alive and 96 had died after heart failure

Feature Analysis



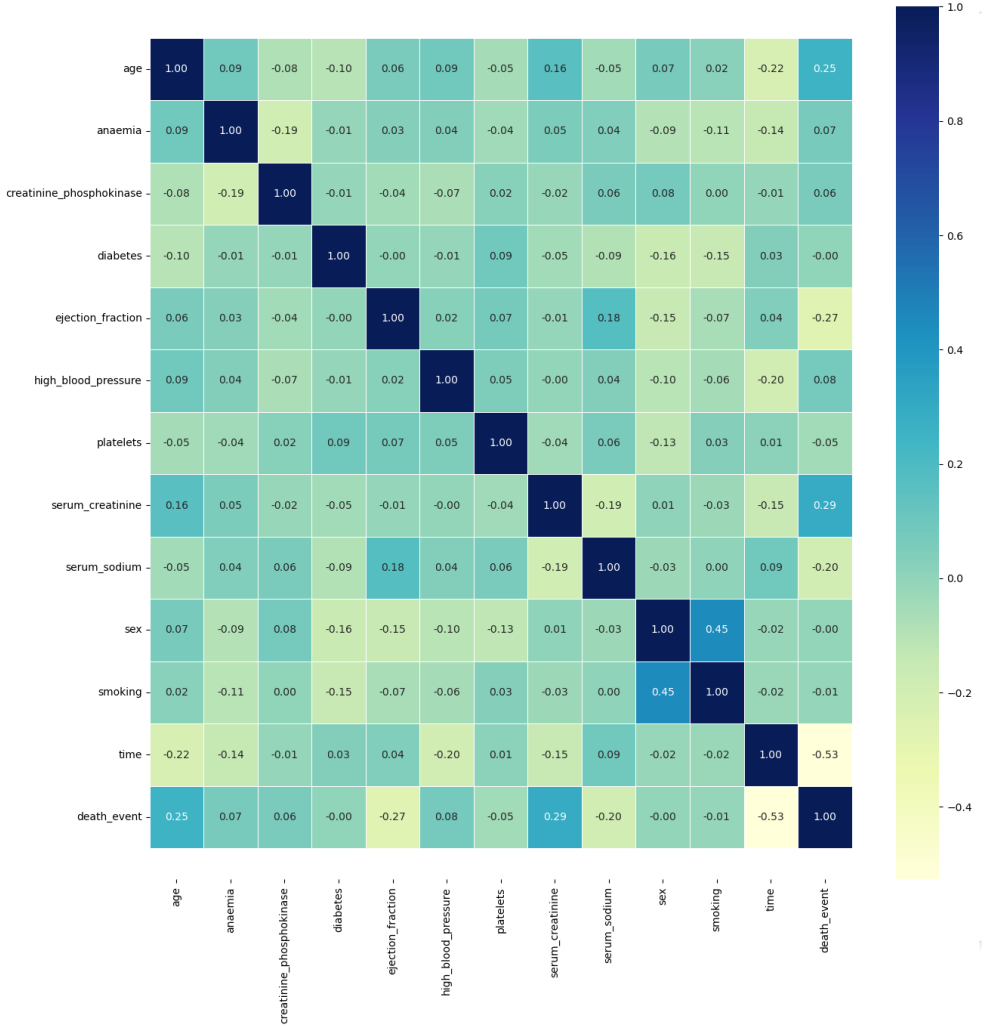
Age – Patients over the age of 70 have a higher chance of not surviving heart failure

Serum Creatinine – Patients with lower levels of serum creatinine - between 0-2 (mg/dL) – tend to have a better chance of survival from heart failure

Serum Creatinine – Patients with higher levels of serum sodium - between 135-150 (mEq/L) – tend to have a better chance of survival from heart failure

Time – here we see that the likelihood of a patient dying after having heart failure is within the first 50-100 days, highlighting the need for patients to have increased care during this period

Correlation Matrix



The correlation matrix is used to show the relationship between features, but most importantly the relationship between the target (‘death event’) and the rest of the features.

Here we see:

- Positive relationship between age and death event (0.25)
- Positive relationship between serum creatinine and death event (0.29)
- Weak positive relationship between anaemia and death event (0.07)


Machine
Learning
Analysis &
Findings

Model Approach

The following analysis compares 4 different Classification models:

- Logistic Regression
- K-Nearest Neighbor (KNN)
- XGBoost
- Random Forest

Using techniques such as standard scaling, cross-validation, grid search along with metrics such as precision, accuracy and F1 score will help to determine the best model in terms of predicting survival of patents with heart failure

Data Splitting

```
]: from sklearn.model_selection import StratifiedShuffleSplit
   from sklearn.metrics import classification_report, confusion_matrix
   from sklearn.metrics import ConfusionMatrixDisplay
   feature_cols = [col_name for col_name in data.columns if col_name != 'death_event']

   # Get the split indexes
   strat_shuf_split = StratifiedShuffleSplit(n_splits=1,
                                             test_size=0.3,
                                             random_state=42)

   train_idx, test_idx = next(strat_shuf_split.split(data[feature_cols], data.death_event))
   # Create the dataframes
   X_train = data.loc[train_idx, feature_cols]
   y_train = data.loc[train_idx, 'death_event']

   X_test  = data.loc[test_idx, feature_cols]
   y_test  = data.loc[test_idx, 'death_event']
```

Logistic Regression

[15]:
```
### BEGIN SOLUTION
from sklearn.linear_model import LogisticRegression

# Standard logistic regression
lr = LogisticRegression(solver='liblinear').fit(X_train, y_train)
y_pred_0 = lr.predict(X_test)
clf_report = pd.DataFrame(classification_report(y_test, y_pred_0, output_dict=True))
clf_report
```

[15]:

	0	1	accuracy	macro avg	weighted avg
precision	0.81	0.76	0.80	0.79	0.80
recall	0.92	0.55	0.80	0.73	0.80
f1-score	0.86	0.64	0.80	0.75	0.79
support	61.00	29.00	0.80	90.00	90.00

Logistic Regression with L1 Penalty

```
from sklearn.linear_model import LogisticRegressionCV

# L1 regularized logistic regression
lr_l1 = LogisticRegressionCV(Cs=10, cv=4, penalty='l1', solver='liblinear').fit(X_train, y_train)
y_pred_1 = lr_l1.predict(X_test)
clf_report = pd.DataFrame(classification_report(y_test, y_pred_1, output_dict=True))
clf_report
```

	0	1	accuracy	macro avg	weighted avg
precision	0.84	0.78	0.82	0.81	0.82
recall	0.92	0.62	0.82	0.77	0.82
f1-score	0.88	0.69	0.82	0.78	0.82
support	61.00	29.00	0.82	90.00	90.00

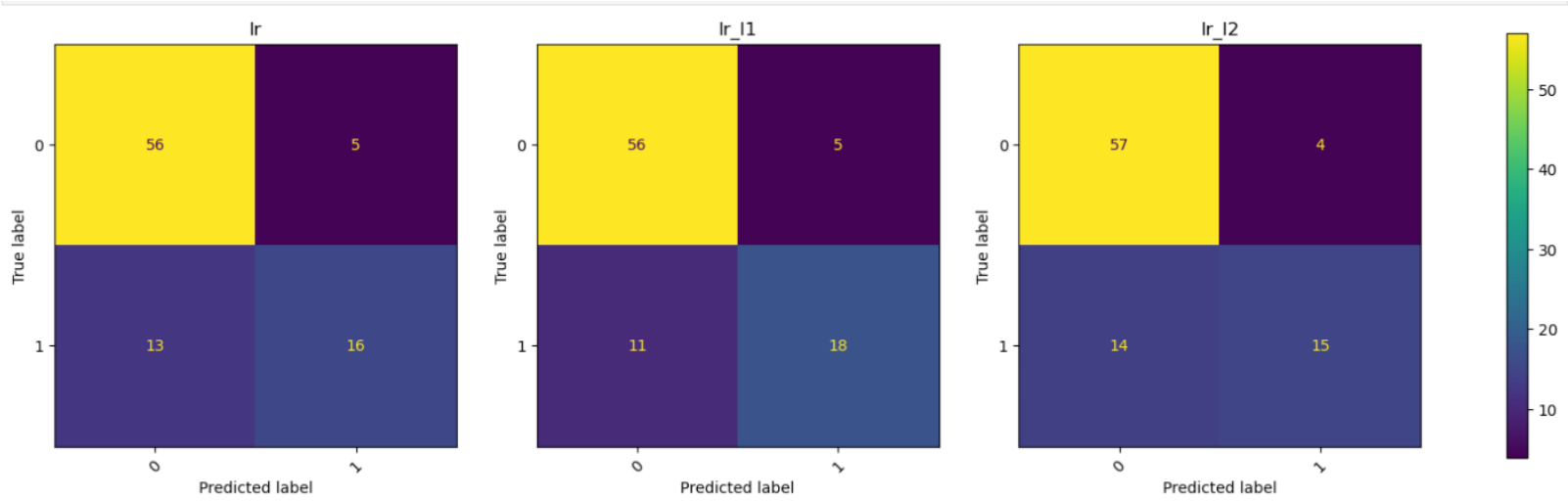
Logistic Regression with L2 Penalty

```
# L2 regularized logistic regression
lr_l2 = LogisticRegressionCV(Cs=10, cv=4, penalty='l2', solver='liblinear').fit(X_train, y_train)
y_pred_2 = lr_l2.predict(X_test)
clf_report = pd.DataFrame(classification_report(y_test, y_pred_2, output_dict=True))
clf_report
```

:

	0	1	accuracy	macro avg	weighted avg
precision	0.80	0.79	0.80	0.80	0.80
recall	0.93	0.52	0.80	0.73	0.80
f1-score	0.86	0.62	0.80	0.74	0.79
support	61.00	29.00	0.80	90.00	90.00

Logistic Regression Model Findings



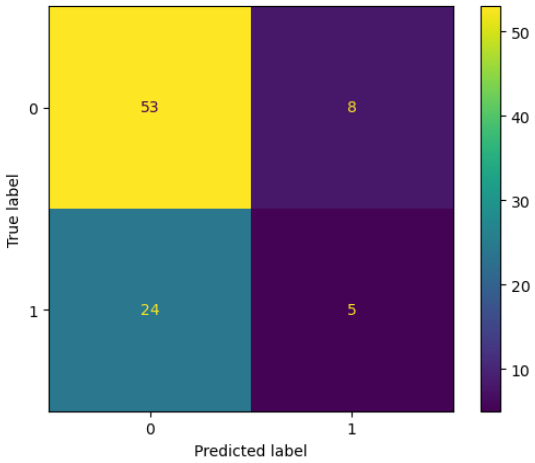
The best model in terms of prediction performance is Logistic Regression with penalty = 1

- Accuracy : 82%
- Precision : 82%
- Recall : 82%
- F1-score : 82%
- Support : 90%

K-Nearest Neighbor (KNN) Algorithm

```
knn = KNeighborsClassifier(n_neighbors=25, weights='distance')
knn = knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

KNN_report = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True))
KNN_report
```

	0	1	accuracy	macro avg	weighted avg
precision	0.69	0.38	0.64	0.54	0.59
recall	0.87	0.17	0.64	0.52	0.64
f1-score	0.77	0.24	0.64	0.50	0.60
support	61.00	29.00	0.64	90.00	90.00



- Accuracy : 64%
- Precision : 59%
- Recall : 64%
- F1-score : 60%
- Support : 90%

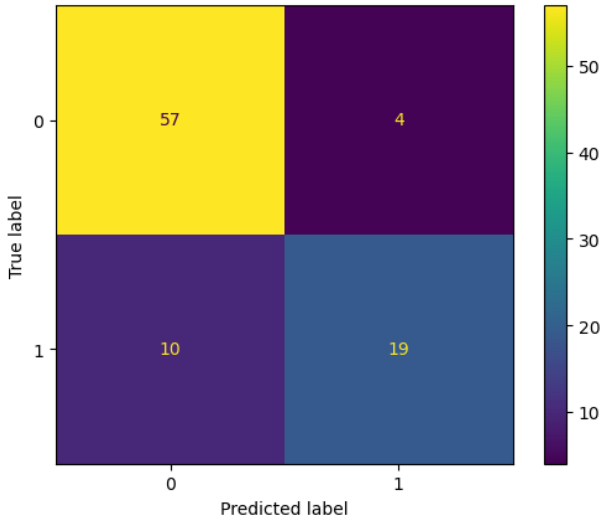
XGBoost Algorithm

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

param_grid = {
    "max_depth": [5],
    "learning_rate": [0.05],
    "gamma": [0, 0.25, 1, 10],
    "reg_lambda": [0],
    "scale_pos_weight": [1, 3, 5, 7, 10],
    "subsample": [0.1,0.2, 0.3, 0.4, 0.5, 0.8],
    "colsample_bytree": [0.5,0.7],
}


# Init classifier
xgb_cl = xgb.XGBClassifier(objective="binary:logistic")

# Init Grid Search
grid_cv = GridSearchCV(xgb_cl, param_grid, n_jobs=-1, cv=3, scoring="roc_auc")

# Fit
_ = grid_cv.fit(X_train, y_train)
```

XGBoost Algorithm Cont.

```
final_xgb_cl = xgb.XGBClassifier(
    **grid_cv.best_params_,
    objective="binary:logistic",
)

_ = final_xgb_cl.fit(X_train, y_train)

y_pred = final_xgb_cl.predict(X_test)

xgb_report = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True))
xgb_report
```

	0	1	accuracy	macro avg	weighted avg
precision	0.85	0.83	0.84	0.84	0.84
recall	0.93	0.66	0.84	0.79	0.84
f1-score	0.89	0.73	0.84	0.81	0.84
support	61.00	29.00	0.84	90.00	90.00



- Accuracy : 84%
- Precision : 84%
- Recall : 84%
- F1-score : 84%
- Support : 90%

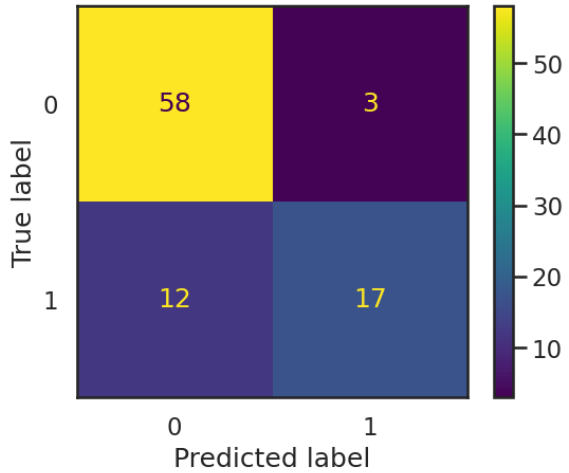
Random Forest

```
rfm = RandomForestClassifier(oob_score=True, n_estimators=100, max_depth=4, min_samples_split=2, random_state=42)

oob_list = list()

# Iterate through all of the possibilities for
# number of trees
for n_trees in [15, 20, 30, 40, 50, 100, 150, 200, 300, 400]:

    # Use this to set the number of trees
    rfm.set_params(n_estimators=n_trees)

    # Fit the model
    rfm.fit(X_train, y_train)

    # Get the oob error (Out-of-bag Error)
    oob_error = 1 - rfm.oob_score_

    # Store it
    oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))

rf_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')
rf_oob_df
```

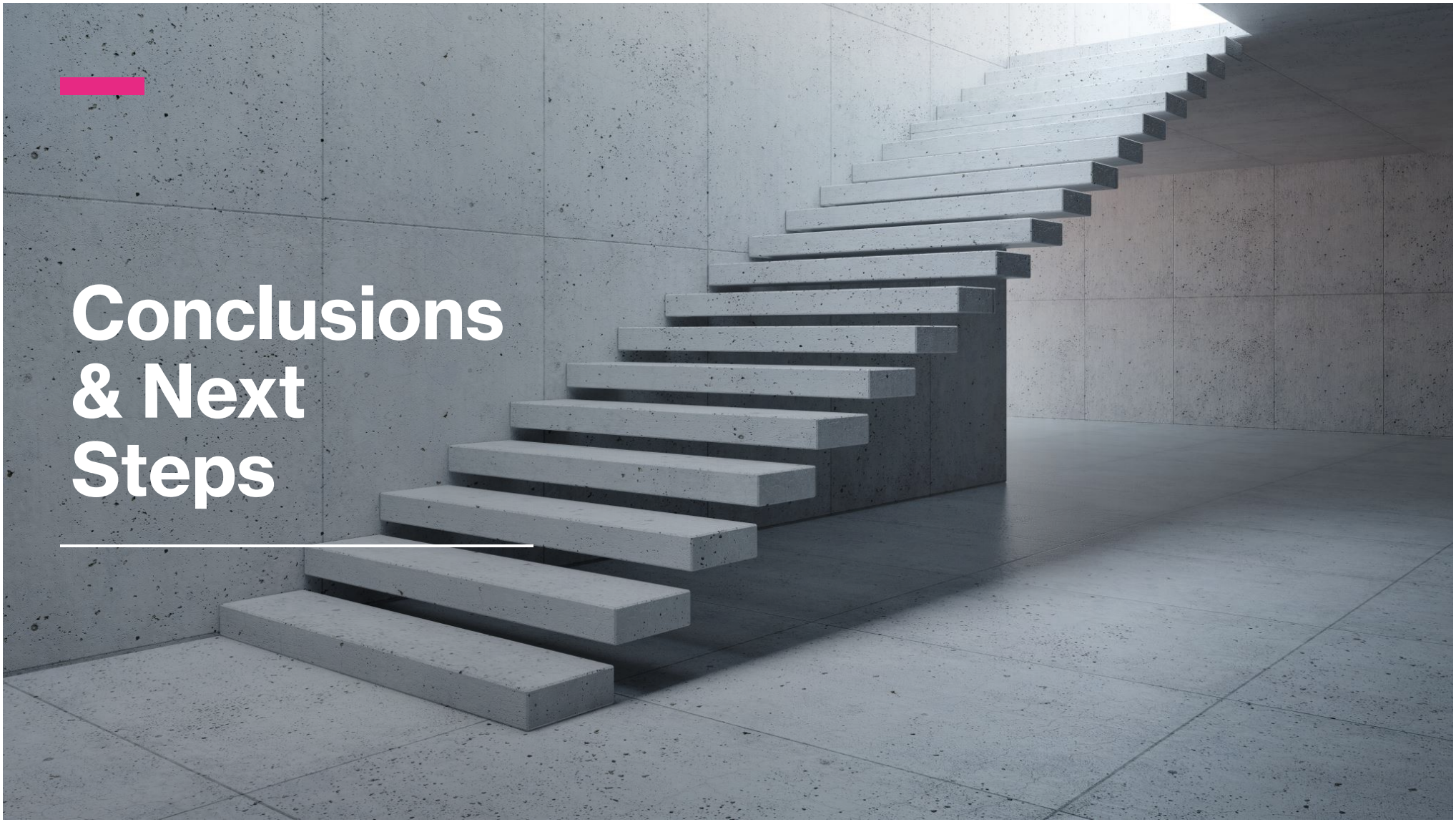
```
# Random forest with 100 estimators
rfm = rfm.set_params(n_estimators=100)

y_pred = rfm.predict(X_test)
```

```
rfm_report = pd.DataFrame(classification_report(y_test, y_pred_0, output_dict=True))
rfm_report
```

	0	1	accuracy	macro avg	weighted avg
precision	0.81	0.76	0.80	0.79	0.80
recall	0.92	0.55	0.80	0.73	0.80
f1-score	0.86	0.64	0.80	0.75	0.79
support	61.00	29.00	0.80	90.00	90.00



- Accuracy : 80%
- Precision : 80%
- Recall : 80%
- F1-score : 79%
- Support : 90%


Conclusions
& Next
Steps

Models Comparison

As shown in the analysis section, except for K-Nearest Neighbor the models used provided good prediction results, but in terms of using the most accurate model the choice is XGBoost with an accuracy of 84%. While L1 Linear Regression and Random Forest were good performing models, XGBoost outperformed both in terms of accuracy, precision, recall and F1 score.

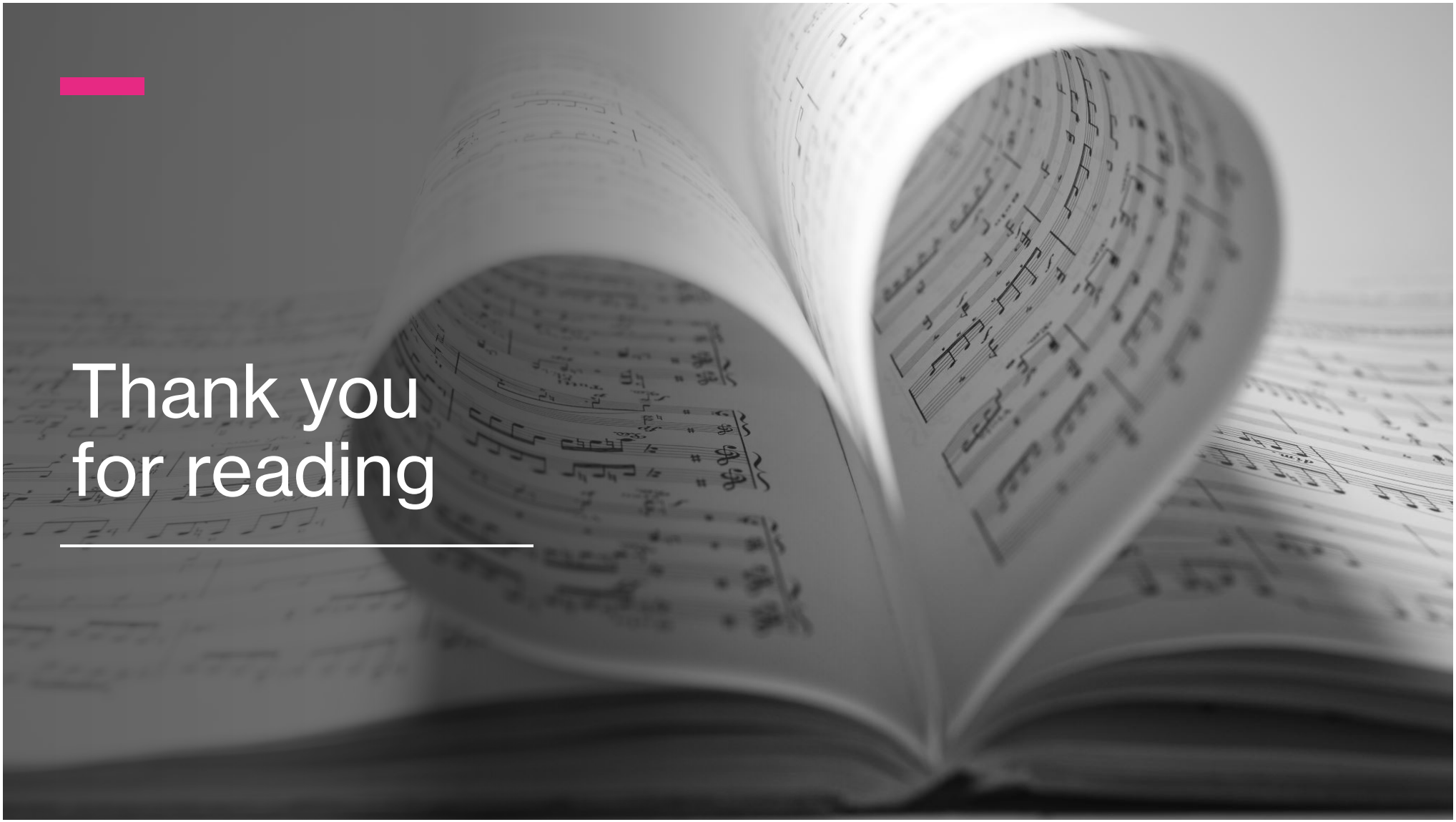
	0	1	accuracy	macro avg	weighted avg
precision	0.85	0.83	0.84	0.84	0.84
recall	0.93	0.66	0.84	0.79	0.84
f1-score	0.89	0.73	0.84	0.81	0.84
support	61.00	29.00	0.84	90.00	90.00

One thing to note however is that while XGBoost performed the best of the four selected models, it also took the longest time to run due to using the grid search technique to search for the best fitting parameters, so this needs to be taken into consideration if using a larger dataset.

Future Development

In terms of future development, I would consider the following:

- Using neural network architectures and other ensemble models such as stacking
- Implementing normalization techniques such as Min-Max scaling and Z-score to improve model generalization and help reduce overfitting
- Using a larger dataset to establish run time trade-off for XGBoost modeling


Thank you
for reading