GROUP 1 - CAP 5610

HEART DISEASE PREDICTION

FINAL PROJECT REPORT - SPRING 2021

Authors

Lamia Alshahrani

Anwesh Praharaj

Dharitri Rathod

Syed Muhammad Sabih

Introduction

This Final Project Report is for CAP 5610 (Spring 2021) final submission. This report consists of:

- The Project Statement and the Problem
- Description and Analysis of the Data
- Modeling Approach
- Investigation of Modeling and Results
- Conclusive Solutions and Future Work

To address the aforementioned deliverables for our Final Report, we decided to work on a project in the field of healthcare and collected the data based on patient attributes that classified the presence of heart disease. Our main goal pertained to the accurate classification/prediction of the presence of heart disease in patients. Further in the report, we will see the repository from which the data was collected, description of the data, the approach to use the data to solve the problem, and the results produced with various testing and modeling approaches.

Table of Contents

Project Statement and Motivation

Project Statement

For the Heart Disease Dataset, we wanted to classify the presence of heart disease using the best combinations of the dataset files (Cleveland, Hungarian, Switzerland, VA) based on the most relevant patient attributes.

Motivation

We decided to choose a dataset in the field of healthcare because medical problems are serious significant problems in the society and the prediction of cardiovascular disease is regarded as one of the most important subjects in the section of clinical data analysis. Furthermore, more people die due to heart disease everyday, so we found a raw database on the UCI Repository that we could investigate and in turn change it into information that could help make informed decisions and predictions in the future.

Heart Disease Dataset

The Dataset

The dataset used for this project was collected from the database located in the UCI Repository. The database contained 76 attributes, but all published experiments referred to using a subset of 14 of them. In particular, the Cleveland database was the only one that had been used by ML researchers to this date. The "goal" field referred to the presence of heart disease in the patient. It was integer valued from 0 (no presence) to 4. Experiments with the Cleveland database had concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0). The names and social security numbers of the patients were recently removed from the database and replaced with dummy values. The input variables for all the databases used were: *age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca*, and *thal* while the output variable used was *num*.

Despite the Cleveland database only being used for research to this date, we decided to test different combinations of databases rather than only using the Cleveland database, along with imputing the missing values to see if we can achieve better results.

Exploratory Data Analysis of the Dataset

Loading the datasets and checking for Missing Values

After loading the dataset files for Cleveland, Hungarian, Switzerland, and VA, the exploration of the data showed that the Cleveland dataset had 303 rows and 14 columns, the Hungarian dataset had 294 rows and 14 columns, the Switzerland dataset had 123 rows and 14 columns, and the VA dataset had 200 rows and 14 columns.

The datasets were further investigated for missing values and the following missing values were found:

- *Cleveland: 6 missing values*
- *Switzerland: 273 missing values*
- *Hungarian: 782 missing values*
- *VA: 698 missing values*

Finally, the descriptive statistics for each feature in each dataset were produced to identify type errors or zero rows. The results of the descriptive statistics are shown in the accompanying Jupyter Notebook for this Milestone of the project.

Basic Data Cleaning

Once the missing values were checked, the next step was to look for redundancy within the datasets. First, we checked for redundant columns/features in the datasets and as a result of the investigation, we found that no column had only one value and the columns with more than one values were categorical variables. Second, we checked for duplicate rows in the datasets and as a result of this investigation, we found that the duplicate values might exist due to the missing values.

Imputation to replace missing values

At this point, we decided to impute the missing values before we could proceed with further exploration and investigation of the datasets. As mentioned before, our team wanted to see if we can use different combinations of datasets rather than only using the Cleveland dataset to achieve better results.

The procedure for imputation of missing values is as follows:

The 5 combinations of datasets that we decided to use were:

- *Cleveland, Hungarian, Switzerland, and VA*
- *Cleveland, Hungarian*
- *Cleveland, Switzerland*
- *Cleveland, Hungarian, and Switzerland*
- *Cleveland (For comparison with other combinations)*

The Imputation methods we decided to use on the aforementioned combinations were:

- *KNN Imputer - Odd k values ranging 1 through 50*
- *Iterative Imputer - Order = ascending, descending, roman, arabic, and random*

Other changes made to the data for imputation:

- *The inputs were normalized using MinMax Scaler*
- *The Target variable was changed to binary (0: Absence of heart disease, 1: Presence of heart disease)*

Evaluation methods to see which combination has a better *accuracy* score:

- *Model used: RandomForestClassifier - default*
- *Cross Validation: RepeatedStratifiedKFold - 10 folds, repeated 3 times, with a random state of 1*

The results of the imputation

- KNN

	index	Name	K	MeanAccuracy	stdAccuracy
2	14	cleveland+switzerland	29.0	0.861443	0.053701
3	1	cleveland+hungarian+switzerland	3.0	0.857870	0.027704
1	1	cleveland+hungarian	3.0	0.843682	0.042083
0	18	cleveland+hungarian+switzerland+va	37.0	0.838043	0.031170
4	17	cleveland	35.0	0.829570	0.048719

- Iterative

	index	Name	ImputationOrder	Mean_Accuracy	std_Dev
2	0	Clev+Switz	ascending	0.856792	0.058223
3	2	Cleve+Hung+Switz	roman	0.852315	0.030320
1	3	clev+hung	arabic	0.837015	0.048449
0	1	Clevel + Hung + Switz + VA	descending	0.836232	0.029957
4	4	Cleveland	random	0.833943	0.048287

After the imputation of the missing values using RandomForestClassifier and RepeatedStratifiedKFold to measure which combination would be better for classification, we decided that Imputation with KNN (K=3) produced the highest mean accuracy and the lowest standard deviation for the combination of Cleveland, Switzerland, and Hungarian datasets.
We decided to use the imputer method and combination of datasets mentioned above to produce our final dataset for this Project.

Analysis of the Imputed Dataset (Final Dataset)

The Final dataset produced after the imputation on the combination of Cleveland, Switzerland, and Hungarian datasets had 720 rows and 14 columns. Upon checking for missing values on the Final dataset, no missing values were found, which meant all missing values were replaced as a result of the imputation. Due to the missing values that existed before imputation, the reduction of redundancy on the raw datasets did not make much sense. After the imputation, the Final Dataset was checked for redundancy and as a result, one duplicate row which existed was removed. Furthermore, the type of variables for the Final Dataset were investigated and findings showed that the input variables were scaled while the target variable remained binary. Additionally, we decided to check that the class distribution is balanced as there are 359 observations in class 0 and 360 observations in class 1.

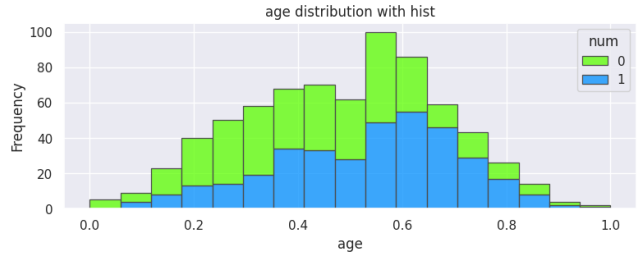
After double-checking for missing values and exploring the Final Dataset, the descriptive statistics were produced for the Final Dataset and the results are shown in the accompanying Jupyter Notebook for this Milestone of the project. Lastly, the correlation between the attributes and skewness of the attributes was produced. At this point, we decided that we needed visualizations for the descriptive statistics, correlation between the attributes, distribution of the dataset, and skewness of the attributes in order to produce more meaningful statistical inferences for the Final Dataset.

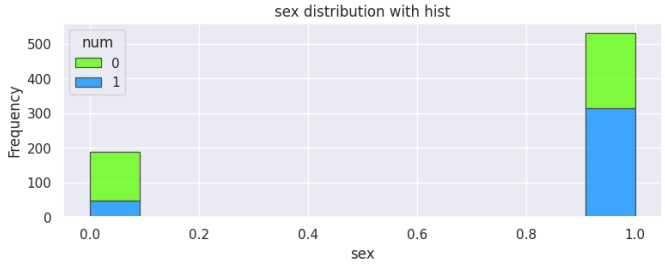
Visualizing the Final Dataset

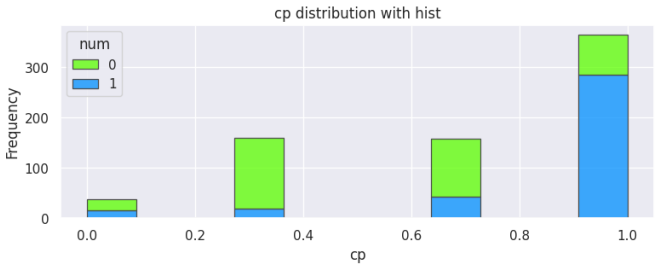
For the visualization of the Final Dataset and its attributes, we decided to produce:

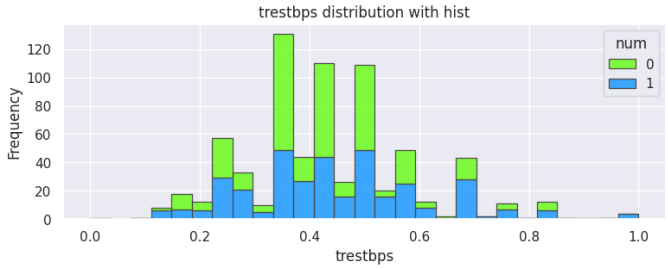
- Unimodal Visualization: *Histogram, Density Curve, and Box Plot*
- Multimodal Visualization: *Scatter Plot and Correlation Matrix*

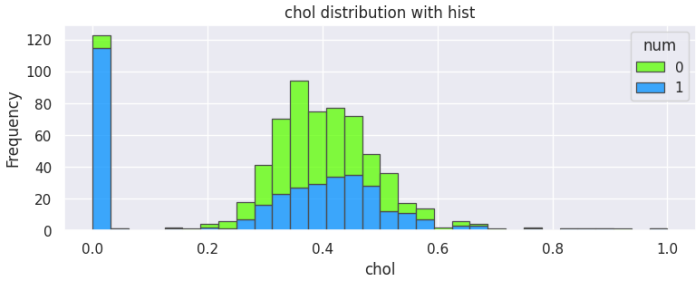
Histograms of the attributes

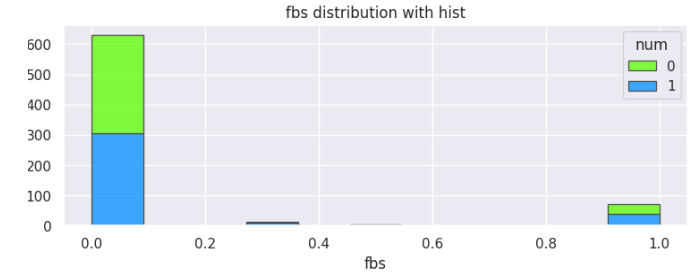


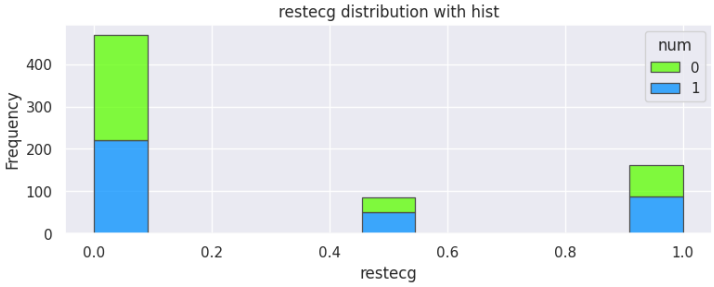


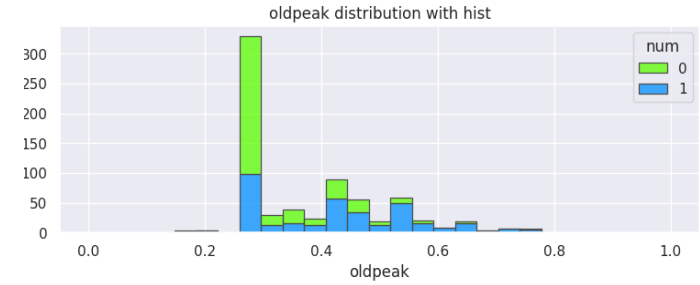


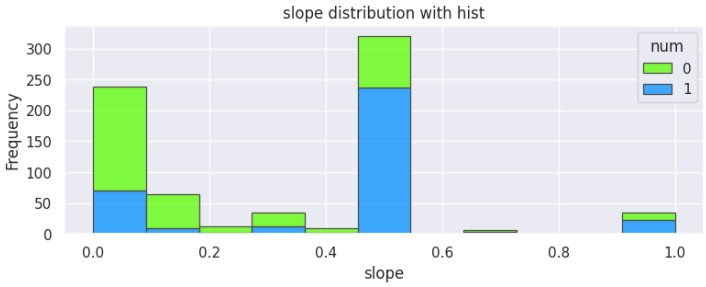


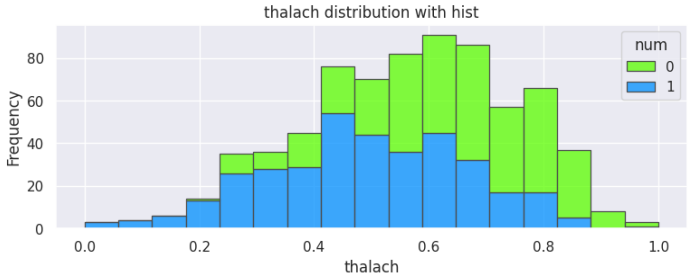


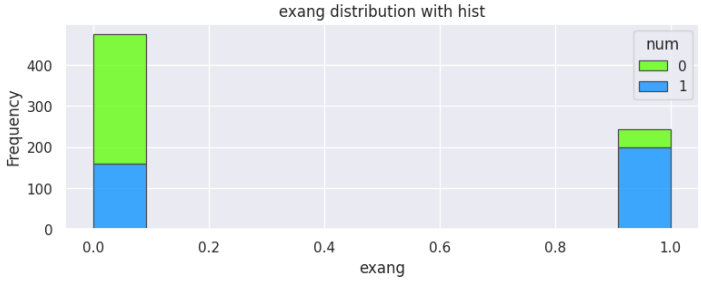


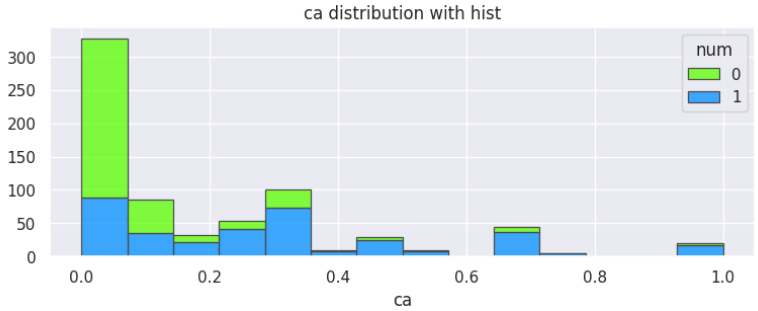


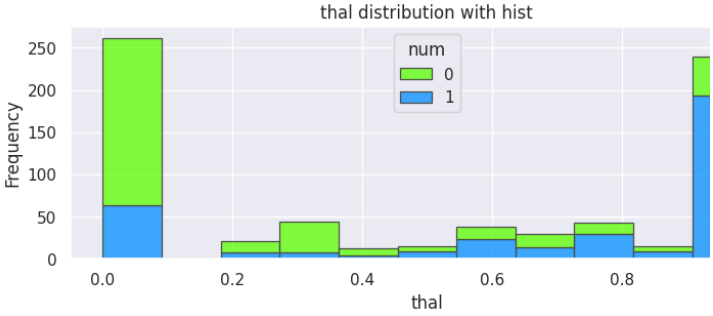



slope distribution with hist
num
0
1
Frequency
slope


thalach distribution with hist
num
0
1
Frequency
thalach

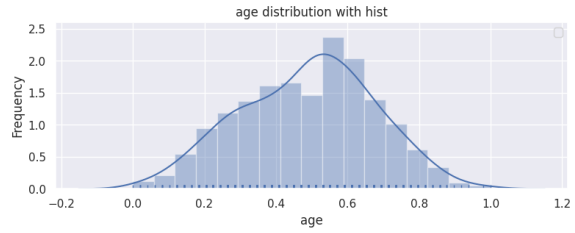

exang distribution with hist
num
0
1
Frequency
exang


ca distribution with hist
num
0
1
ca

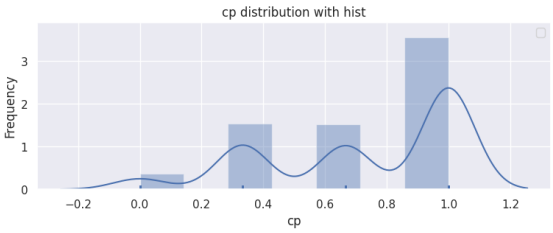

thal distribution with hist
num
0
1
Frequency
thal


num distribution with hist
num
0
1
Frequency
num

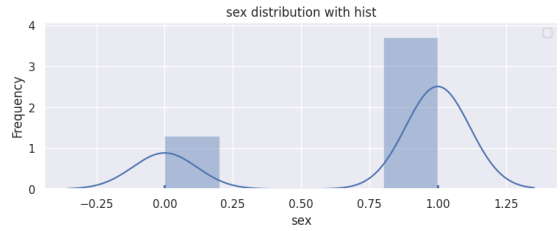
Density Curves for the attributes

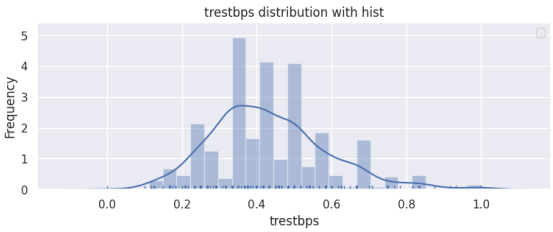


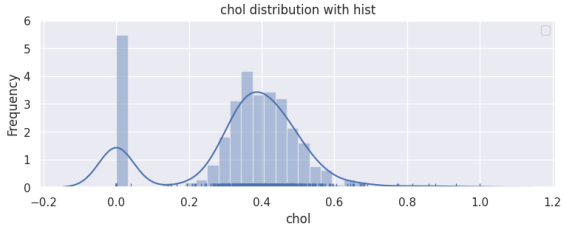
No handles with labels found to put in legend.

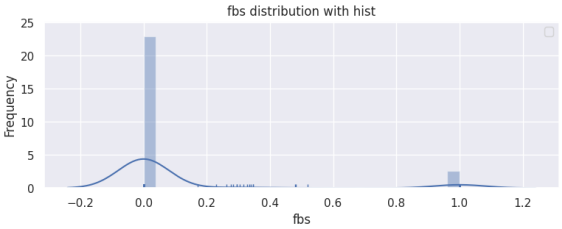


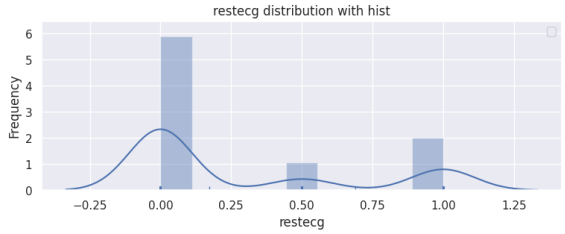
No handles with labels found to put in legend.



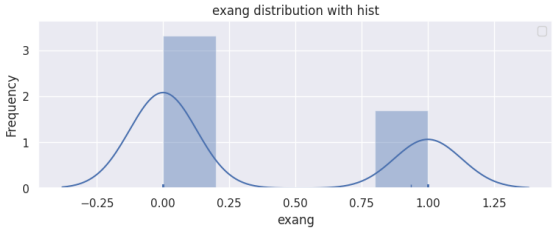




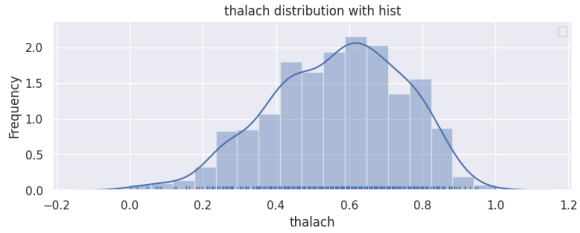


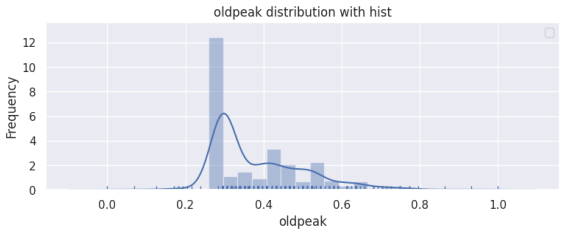


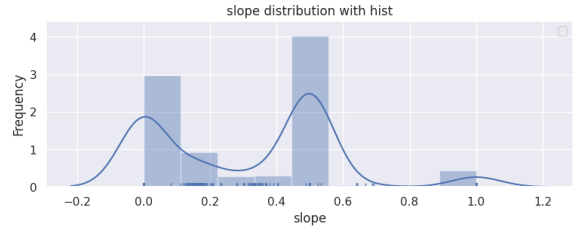
No handles with labels found to put in legend.



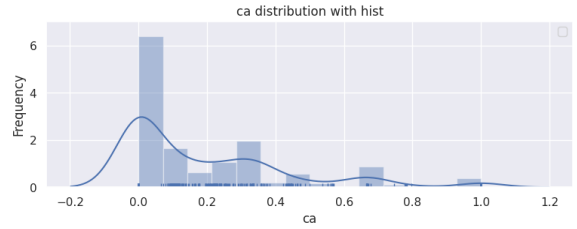
No handles with labels found to put in legend.

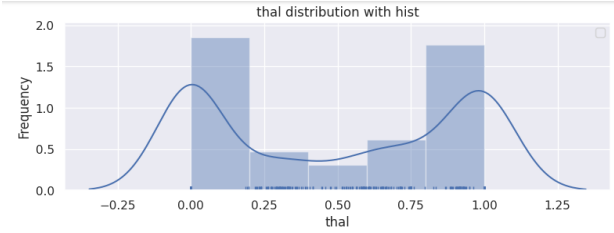




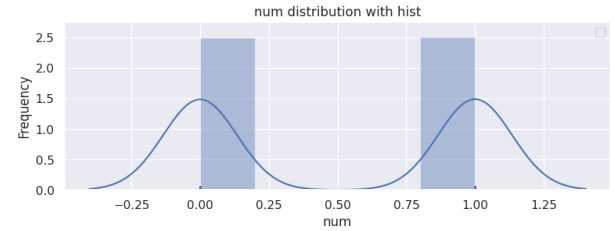


No handles with labels found to put in legend.

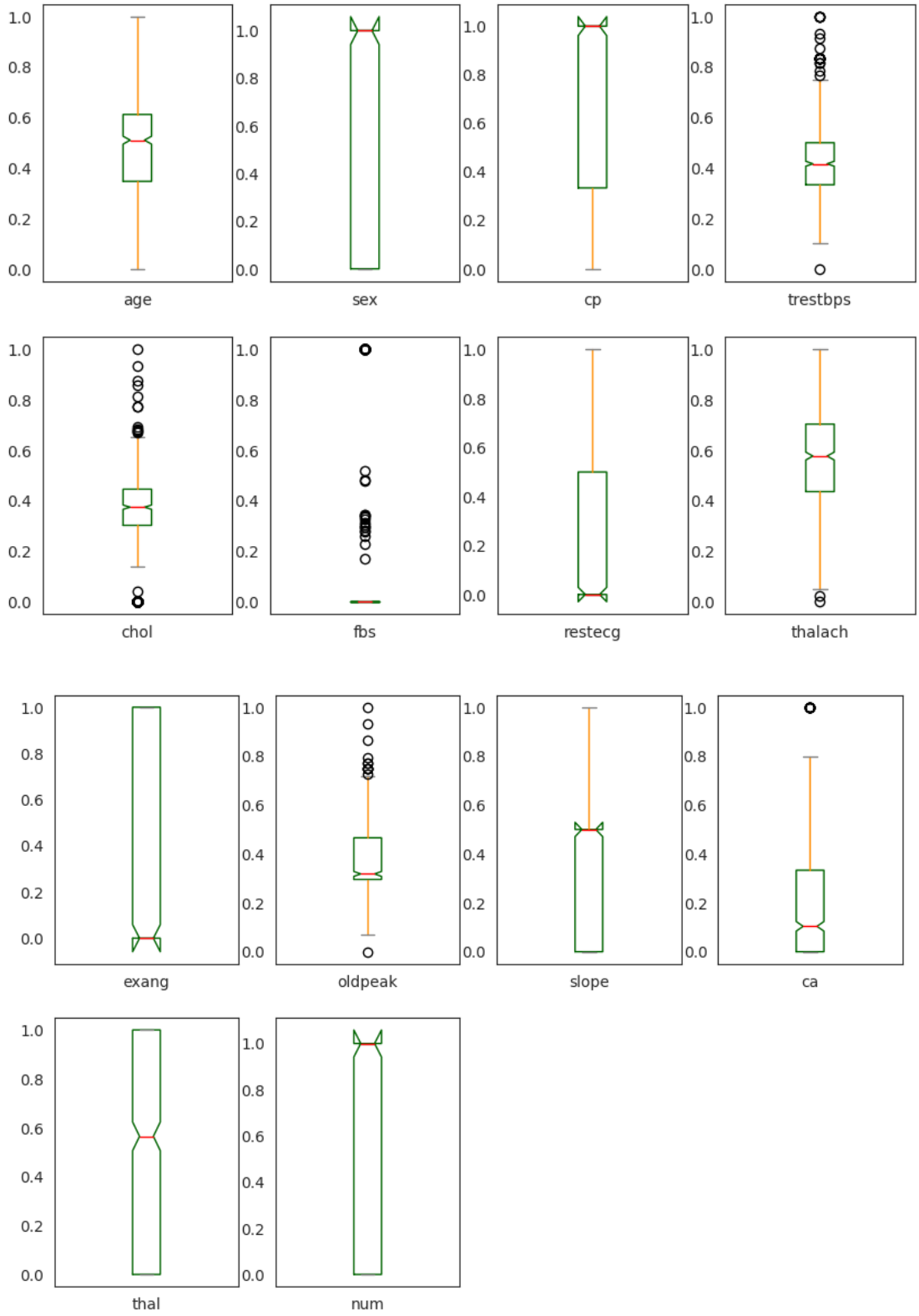




No handles with labels found to put in legend.

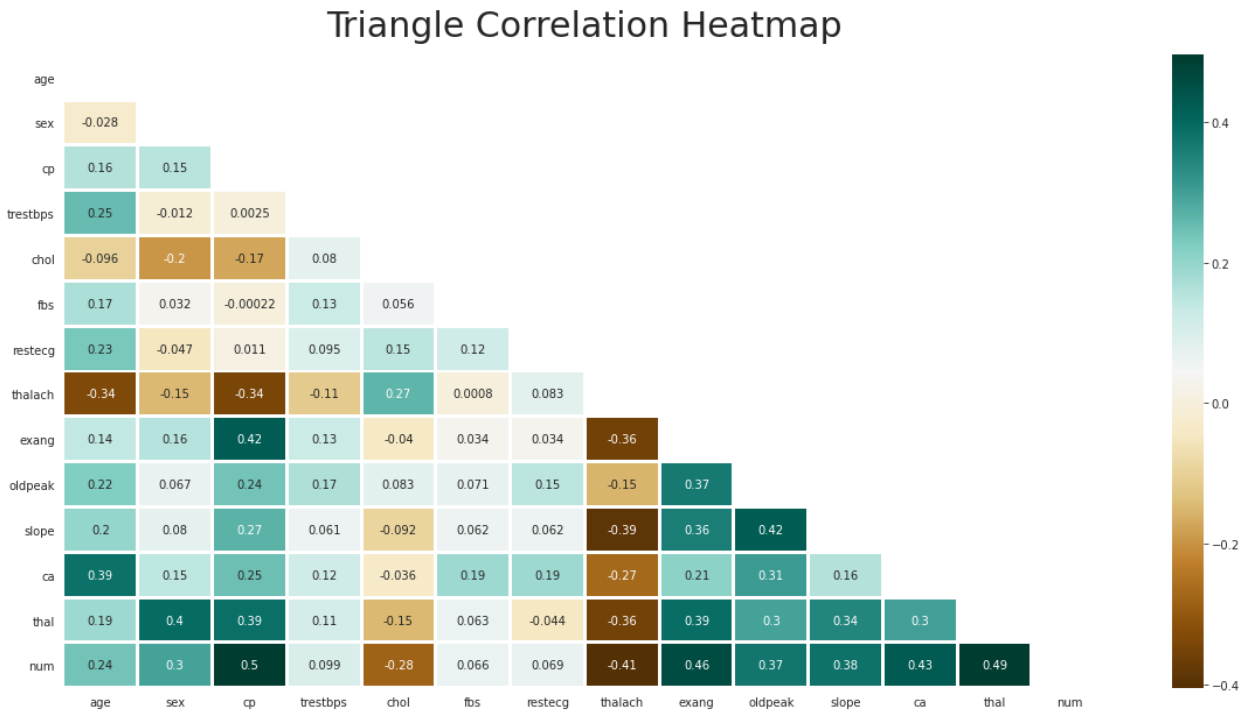


Box Plots for the attributes



Scatterplot for Multimodal Analysis

Correlation Matrix for Multimodal Analysis



Interpreting the Visualizations

After investigating the visualizations, the findings from the histograms and density curves stated that some of the quantitative attributes are skewed. These findings suggested that standardizing the data might produce better results. Moreover, the investigation of the boxplots indicated that some of the variables have outliers which might affect the results of some of the algorithms. Although, we decided to not remove the outliers since the data is concerned with the medical field and the outliers could work in our favor in producing accurate results. However, results produced with and without the outliers would give us the final verdict. Furthermore, upon investigating the Scatter-plot and the correlation matrix, the findings suggested strong correlation between the output variable, *num*, and some input variables such as *thal, ca, slope, oldpeak, exang, thalach, cp, and sex,* which indicated that these predictors could produce more accurate results.
After analyzing the data through producing statistics and visualizations, we produced a few questions that need to be investigated for the dataset:

- Would fewer predictors rather than all the predictors in classifying the presence of heart disease produce a more accurate result?
- If yes, which predictors are the best?
- Are the predictors controllable?
- Were there any new findings after visualizing the data?

Building the Baseline Models

Procedure

At this point, we decided to use all the predictors to build our Baseline Model. Before building a Baseline Mode, we decided to split the data into two parts: *70% training* and *30% testing*. After that, we considered 10 classification algorithms for model selection*: Logistic Regression, KNN Classifier, Support Vector Machine, Random Forest Classifier, Ada Boosting, Decision Tree Classifier, Bagging Classifier, XG Boosting, Linear Discriminant Analysis, and Naive Bayes*.

A spot-check along with the ROC curve and confusion matrix for the algorithms was performed to measure the most effective model performances that produced the best results. Furthermore, we have plotted the variable importance for the tree classifiers to understand the splitting in the data.

During the spot-check for algorithms, performances of the algorithms were measured using metrics such as: prediction accuracy, log loss, precision, F1- score, R-Square, and AUC.

After running the 10 models, the results were as follows:

	ModelName	logloss	Test accuracy	precision	F1	r2	AUC
0	XG Boosting	4.957006	0.856481	0.865385	0.853081	0.425877	0.856341
0	Random Forest Classifier : RFC	5.116904	0.851852	0.871287	0.846154	0.407357	0.851582
0	Ada Boosting	5.436722	0.842593	0.841121	0.841121	0.370316	0.842579
0	KNN	5.596609	0.837963	0.867347	0.829268	0.351796	0.837563
0	GaussianNB : NB	5.756529	0.833333	0.831776	0.831776	0.333276	0.833319
0	SVM	5.756529	0.833333	0.831776	0.831776	0.333276	0.833319
0	bagging Classifier	5.756522	0.833333	0.844660	0.828571	0.333276	0.833148
0	Logistic Regression : LR	6.396140	0.814815	0.819048	0.811321	0.259196	0.814713
0	Linear Discriminant Analysis : LDA	6.556038	0.810185	0.823529	0.803828	0.240676	0.809955
0	Decision Tree Classifier	7.355565	0.787037	0.785047	0.785047	0.148075	0.787019

Baseline Models Performance Conclusion

Based on the findings above, the results indicated that *XG Boosting, RandomForest, Ada Boosting, KNN Classifier, and SVM* were performing in the most effective manner and producing the best results.

Addressing the Questions from Visualization

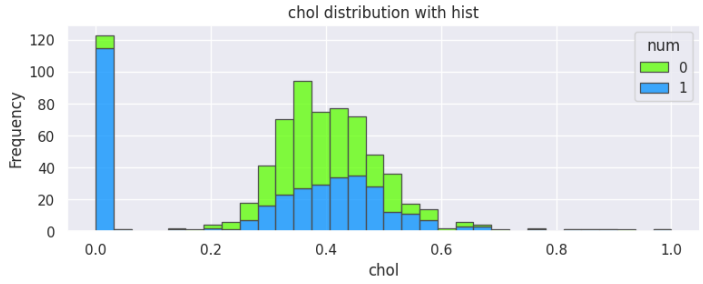
- Would fewer predictors rather than all the predictors in classifying the presence of heart disease produce a more accurate result?
- If yes, which predictors are the best?
- Are the predictors controllable?
- Were there any new findings after visualizing the Final Dataset?

Feature (Predictor) Selection

For the most effective results, depending on the algorithm used, the number of features would vary. For example, feature A might be useful for model A, but might not be useful for model B. To tackle this problem, we planned to use Principal Component Analysis(PCA) and SelectKBest depending on the algorithm's requirement to select the best features which produced the most effective model performance. Depending on the model, different techniques of feature selection should be used.

New Findings after visualizing the Final Dataset

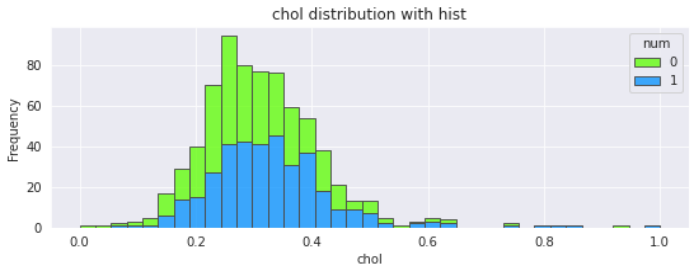
After visualizing the histogram for the feature *chol,* we noticed an abundance of 0 values and the rest of the data was normal.



Hence, we investigated the statistics for unique values where we had previously checked for redundancy.

```
Unique column values in Switzerland dataset
 age         37
sex           2
cp            4
trestbps     20
chol          1
fbs           2
restecg       3
thalach      67
exang         2
oldpeak      35
slope         3
ca            2
thal          3
num           5
dtype: int64
```

The findings from the investigation proved that the missing values for *chol* were recorded as '0' for the Switzerland dataset. Therefore, to solve this problem, we replaced the '0' values with NaN and imputed the missing values. Following the imputation procedure mentioned earlier in this report, we imputed the missing values using KNN Imputer (K=47) on the same combination of datasets (Cleveland, Switzerland, and Hungarian) as it produced the best accuracy and standard deviation. A new Final Imputed Dataset was produced for building the models. The histogram for the feature *chol* after imputing the '0' values in the Switzerland is as follows:



The feature *chol* above after the imputation of '0' values seems to follow a normal distribution.

Baseline Models - New Imputed Dataset

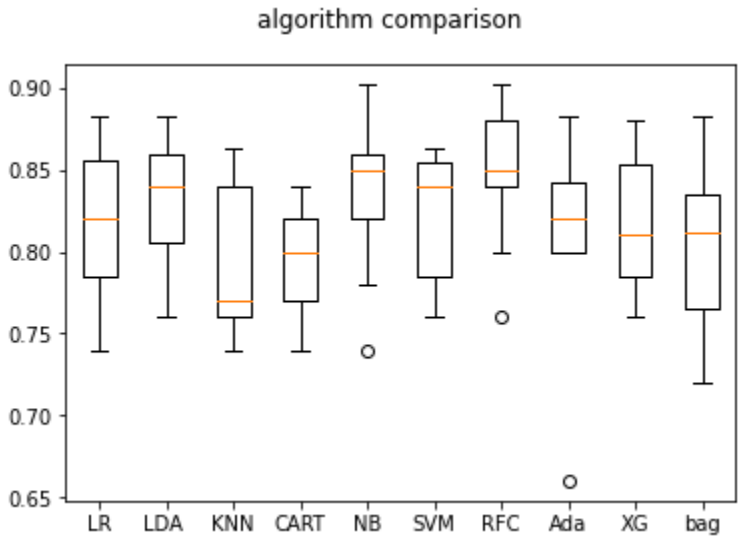
Procedure

At this point, we decided to build the Baseline models on the New Imputed Dataset using the procedure used previously. This was necessary so could compare the model performances to the previous Baseline models using the same metrics used before (prediction accuracy, log loss, precision, F1- score, R-Square, and AUC). The results are as follows:

	ModelName	logloss	Test accuracy	precision	F1	r2	AUC
0	Linear Discriminant Analysis : LDA	5.597	0.838	0.864	0.836	0.352	0.839
0	Random Forest Classifier : RFC	5.757	0.833	0.849	0.833	0.333	0.834
0	Ada Boosting	5.916	0.829	0.854	0.826	0.315	0.829
0	Logistic Regression : LR	6.236	0.819	0.838	0.819	0.278	0.820
0	KNN	6.236	0.819	0.838	0.819	0.278	0.820
0	GaussianNB : NB	6.236	0.819	0.845	0.817	0.278	0.820
0	XG Boosting	6.396	0.815	0.843	0.811	0.259	0.815
0	bagging Classifier	6.396	0.815	0.830	0.815	0.259	0.815
0	SVM	6.716	0.806	0.821	0.806	0.222	0.806
0	Decision Tree Classifier	7.036	0.796	0.806	0.798	0.185	0.796

Box Plots for model performance

The box plots for the prediction accuracy for the New Baseline Models is as follows:



New Baseline Models Performance Conclusion

Based on the findings above, the results indicated that *Random Forest Classifier, AdaBoosting, and Linear Discriminant Analysis* were performing in the most effective manner and producing the best results.

Hyperparameter Tuning for the Selected Models

Logistic Regression:

Logistic regression is one of the statistical models , mainly used in modeling a binary dependent variable.

Rescaling

For LR we had applied `StandardScaler()` to standardize the data inside the pipeline .

Feature selection

In the feature selection we had used the PCA .Also, we had tried the RFE but it did not improve the accuracy .

Hyper parameter tuning

In the LR we had used the default hyper parameters `pca__n_components': [5, 15, 30, 45, 64], 'logistic__C':(-4, 4, 4)` and run the gridsearch to look for the best accuracy we got : `0.8240740740740741` using `Best parameter (CV score=0.837), 'pca__n_components': 5 .`

Final Model

Finally, we had fit and used the best parameters we had fit and predicted the model in order to print the performance metrics to evaluate the model.

Model performance on the test data set:

Metric	Test
accuracy	0.819444
precision	0.858586
F1	0.813397
r2	0.27753
AUC	0.820326

SVM

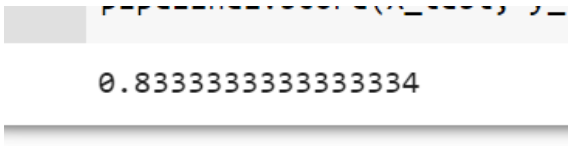
A support vector machine (SVM) is a supervised machine learning model that is mainly used in classification problems.

Rescaling

In SVM we had applied the StandardScalar() inside the pipeline , in order to improve our accuracy. However, it did not change or raise that.

Feature selection

In this model, we had tried RFE,PCA,and RepeatedSelectKBest() for the feature selection . The best accuracy we got assigning the (n_features_to_select=8)



Hyper parameter tuning

The most important hyperparameters in SVM are C and the Kernel and gamma . In the REF we had assigned the default value of SVM (kernel="linear", C=1). Using the grid search we had to add gamma to search for the best accuracy.Also, we had assigned random_state=7 to generate the same result.

Final Model

Finally , using the best parameters we had fit and predicted the model in order to print the performance metrics to evaluate the model.

Model performance on the test data set:

Metric	Test
accuracy	0.828704
precision	0.847619
F1	0.827907
r2	0.31458
AUC	0.829074

Decision Tree Classifier

Decision trees are a powerful prediction method and extremely popular. Decision Trees are a type of Supervised Machine Learning where the data is continuously split according to a certain parameter.

Rescaling

For Decision Tree Classifier, the data was scaled using `StandardScaler()` and added into the pipeline with the model.

Feature selection

Feature selection was performed using `PCA()` and added into the pipeline with the model as well to reduce the dimensionality of the data and produce the best possible results using the most useful features.

Hyper parameter tuning

The hyperparameters that were optimized were set to `'dec_tree__criterion': ['gini', 'entropy'], 'dec_tree__max_depth': range(1,15), 'dec_tree__min_samples_split' : range(1,15), 'dec_tree__min_samples_leaf' : range(1,15),` and `'pca__n_components' :[6,7,8,9,10]`.

The optimized hyperparameters using `GridSearchCV()` were:

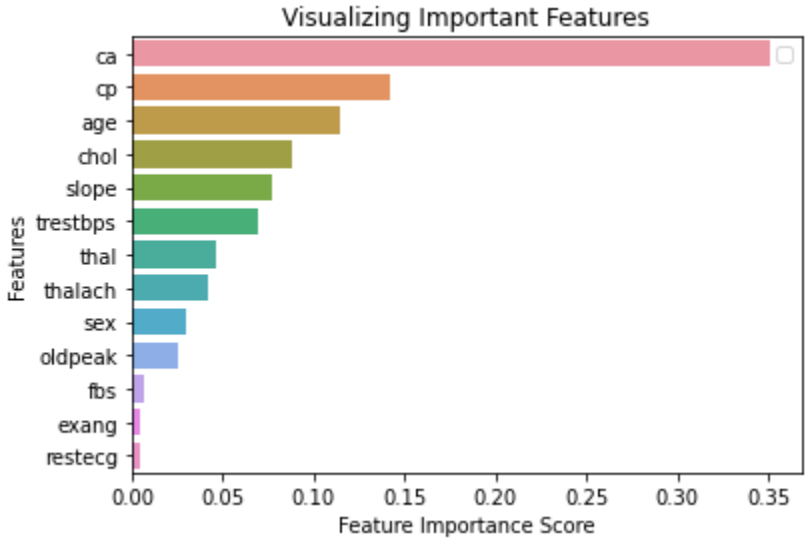
```
ccp_alpha=0.0, class_weight=None, criterion='gini', max_depth=4,
max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=4, min_samples_split=14,
min_weight_fraction_leaf=0.0, presort='deprecated',
random_state=None, splitter='best').
```

Final Model

The results using the optimized model are as follows:

Model performance on the test data set:

Metric	Test
accuracy	0.791667
precision	0.777778
F1	0.801762
r2	0.166381
AUC	0.790995



KNN Classifier

KNN Classifier is used in both classification and regression problems. In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm.

Rescaling

For KNN Classifier, the data was scaled using `StandardScaler()` and added into the pipeline with the model.

Feature selection

Feature selection was performed using `PCA()` and added into the pipeline with the model as well to reduce the dimensionality of the data and produce the best possible results using the most useful features.

Hyper parameter tuning

The hyperparameters that were optimized were set to

```
'KNN__n_neighbors' :[2,4,6,8,10,12,14,16], 'KNN__algorithm' : ['auto',
'ball_tree', 'kd_tree', 'brute'], 'KNN__leaf_size'
:[1,2,3,4,5,6,7,8,9,10], 'KNN__p' :[1,2,3,4,5], and 'pca__n_components'
:[6,7,8,9,10]
```

The optimized hyperparameters using `GridSearchCV()` were:

```
algorithm='auto', leaf_size=1, metric='minkowski',
metric_params=None, n_jobs=None, n_neighbors=14, p=3, and
weights='uniform
```

Final Model

Model performance on the test data set:

Metric	Test
accuracy	0.819444
precision	0.865979
F1	0.811594
r2	0.27753
AUC	0.820497

XGBoost Classifier

XGBoost is also an ensemble algorthim of decision trees.

Rescaling

For XGBoost Classifier, the data was scaled and normalized using `Normalizer()` and added into the pipeline with the model. However, we found that the accuracy had decreased .

Feature selection

In the feature selection the `SelectKBest()` was applied but it did not help to get a high accuracy . Also, it generates its own feature of importance while building a tree.

Hyper parameter tuning

The hyperparameters that have been used were:

```
'xgb_clf__n_estimators': [50, 100, 150, 200], 'xgb_clf__learning_rate':
[0.01, 0.1, 0.2, 0.3], 'xgb_clf__max_depth': range(3, 10),
'xgb_clf__colsample_bytree': [i/10.0 for i in range(1, 3)],
'xgb_clf__gamma': [i/10.0 for i in range(3)], and
'xgb_clf__min_child_weight': [1, 5, 10]
```

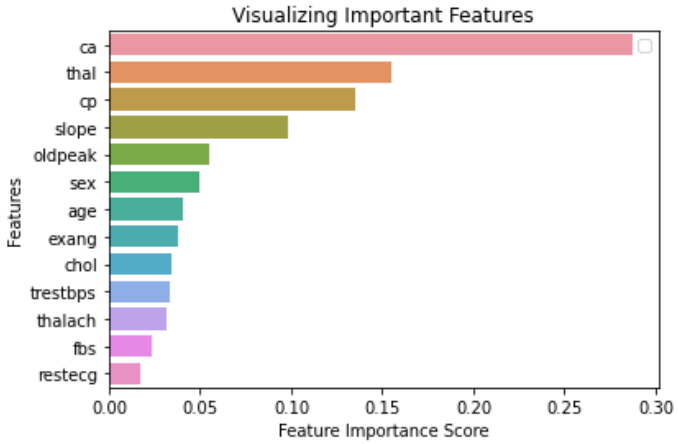
The optimized hyperparameters using `GridSearchCV()` were:

```
'xgb_clf__colsample_bytree': 0.2, 'xgb_clf__gamma': 0.1,
'xgb_clf__learning_rate': 0.01, 'xgb_clf__max_depth': 3,
'xgb_clf__min_child_weight': 1, and 'xgb_clf__n_estimators': 200
```

Final Model

Model performance on the test data set:

Metric	Test
accuracy	0.833333
precision	0.862745
F1	0.830189
r2	0.333105
AUC	0.833962



Gaussian Naive Bayes

The extension of naive Bayes is called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because we only need to estimate the mean and the standard deviation from our training data.

Rescaling

Gaussian Naive Bayes performs standardization internally.

Feature selection

Naive Bayes methods work by determining the conditional and unconditional probabilities associated with the features and predict the class with the highest probability. No Feature selection algorithms were applied.

Hyper parameter tuning

The hyperparameters that were optimized were set to

```
'gnb_clf__priors': [None] and 'gnb_clf__var_smoothing': [0.00000001,
0.000000001, 0.0000000001,0.00000000001,0.000000000001]
```

The optimized hyperparameters using `GridSearchCV()` were:

```
'gnb_clf__priors': None, 'gnb_clf__var_smoothing': 1e-08
```

Final Model

Model performance on the test data set:

Metric	Test
accuracy	0.819444
precision	0.84466
F1	0.816901
r2	0.27753
AUC	0.819983

Linear Discriminant Analysis

LDA is a linear model for multi-class classification. problem, the base accuracy of LDA was 0.8379 on the training dataset.

```
============================Linear Discriminant Analysis : LDA =====
confusion matrix         : [[92 14]
 [21 89]]
r2-score              : 0.35162950257289904
accuracy_score        : 0.8379629629629629
auc_test:               0.838507718696398
```

Rescaling

We have applied the different rescaling methods to improve the results such as StandardScalar(), Normalizer() in the pipeline and outside the pipeline, but didn't see any improvement in the accuracy. For further analysis, we will continue to Standardize data inside the pipeline to avoid leakage of the data.

Feature selection

We have tried SelectKBest(), RFE, and PCA for the feature selection, but we got the best accuracy of 0.840392 with PCA(n_component=6) in the pipeline.

```
ScaledLDA: 0.840392 (0.049474)
```

Hyperparameter tuning

Most important hyperparameters are *solver*= ['svd', 'lsqr', 'eigen'], *Shrinkage* =[0,1], and n_components[1,13]. As PCA gave us n_components=6,we have run the grid search to find the best solver='lsqr' and shrinkage=0.0 with accuracy 0.835.

```
Mean Accuracy: 0.835
Config: {'shrinkage': 0.0, 'solver': 'lsqr'}
```

Final Model

We will fit and predict the dataset with n_component=6, which is

Model performance on the test data set:

Metric	Test
accuracy	0.824074
precision	0.852941
F1	0.820755
r2	0.296055
AUC	0.8247

Random Forest Classifier

Rescaling

It is one of the ensemble models. Hence, there is no need to rescale the data. Base accuracy is 0.8379

Feature selection

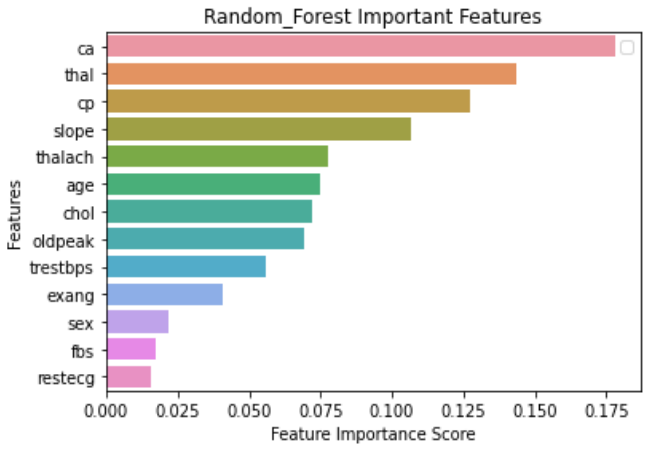
Also, it generates its own feature of importance while building a tree.

Hyper parameter tuning

Random Forest has mainly 3 features to tune namely: n_estimator, max_features, and max_depth. Also, to generate the same result, we have used random_state=7. To select the best parameters using grid search we have n_estimator= 1000 (number of trees), max_features='sqrt' (take only sqrt(p) features in consideration for the split of the node), and max_depth =12 with training accuracy of 0.85533 and predicted accuracy of 0.8333.

Final Model

```
accuracy_score        : 0.8379629629629629
auc_test:               0.8383361921097771
Best: 0.855333 using {'max_depth': 12, 'max_features': 'sqrt', 'n_estimators': 1000}
```




```
r2-score              : 0.33310463121783906
accuracy_score        : 0.8333333333333334
auc_test:               0.8336192109777016
```

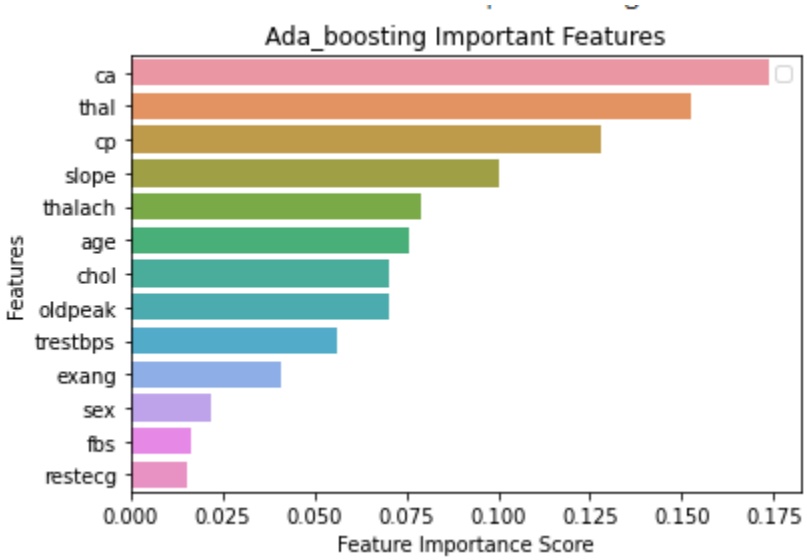
AdaBoost Classifier

Another type of ensemble, but it is a boosting algorithm. We will use Adaboosting on top of the previous Random Forest algorithm to improve its accuracy by passing it as a parameter in Adaboosting. Base accuracy is of Ada boosting algorithm is : 0.828

```
accuracy_score        : 0.8287037037037037
auc_test:               0.8292452830188679
```

As it is a tree based classifier, we don't need to rescale or do feature selection. It has mainly 3 hyper parameters: base_estimator (other algorithm), n_estimators (number of trees), and learning rate. Using hyperparameter tuning on the base_estimator =[RandomForestClassifier(),LogisticRegression(),DecisionTreeClassifier(max_depth=8)], n_estimators= [10, 50, 100, 500],and learning_rate=[0.0001, 0.001, 0.01, 0.1, 1.0], the best_estimator is RandomForestclassifier(),n_estimator=50, and learning_rate=0.0001. For computation, we have used n_estimaors=100 with learning_rate=0.1 as accuracy was almost the same.

```
Best: 0.856667 using {'learning_rate': 0.0001, 'n_estimators': 50}
```

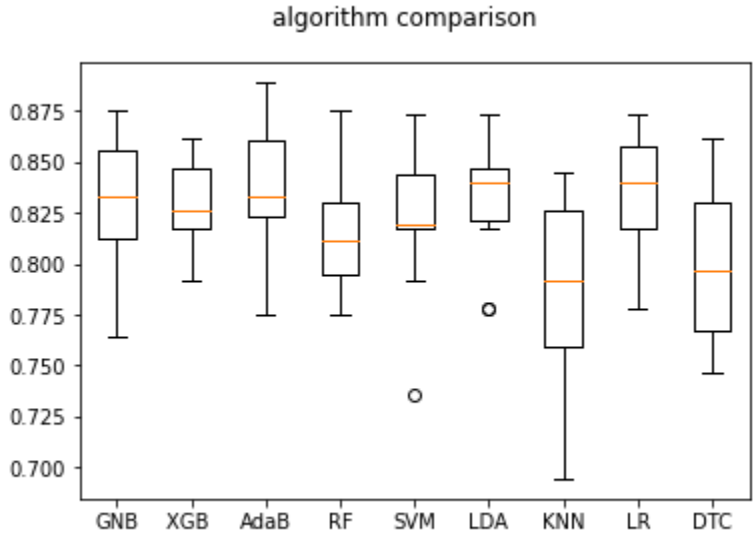


Using best parameter, we fit the final model and predicted accuracy is:

```
accuracy_score       : 0.8425925925925926
auc_test:              0.8430531732418526
```

Best Performing Models after Hyperparameter Tuning

Due to the stochastic nature of the algorithms, the measures used to assess model performance would vary in each instance the models were fitted and tested on the data. However, few of the models stood out with regards to prediction accuracy. The box plot for the prediction accuracy for optimized models is as follows:



Looking at this box plot, we decided to build two voting ensembles in the next part of this report using combinations of three and four algorithms to assess any gains in model performance. We will compare the results of Voting ensembles and optimized algorithms in the next part as well.

Building a Voting Ensemble from the Best Performing Models

Using Voting Classifier to produce the Mean Accuracy

At this point, we decided to use the two combinations of best performing Models:

- AdaBoosting, XGBoosting, LDA, and SVM
- AdaBoosting, LDA, and SVM

Using the Voting Classifier, we combined the Algorithms mentioned above with regards to the combinations mentioned above to build an ensemble. The ensemble would help us

assess if combining these algorithms produced an increase in mean prediction accuracy.

	index	ModelName	Test_accuracy	precision	F1	r2	AUC
8	8	AdaBoostClassifier	0.842593	0.865385	0.841121	0.370154	0.843053
10	10	Voting(LDA,Ada,SVM)	0.842593	0.865385	0.841121	0.370154	0.843053
4	4	XGBoosting	0.833333	0.862745	0.830189	0.333105	0.833962
7	7	RandomForestClassifier	0.833333	0.849057	0.833333	0.333105	0.833619
1	1	SVM	0.828704	0.847619	0.827907	0.314580	0.829074
9	9	Voting(LDA+Ada+XGB+SVM)	0.824074	0.867347	0.817308	0.296055	0.825043
6	6	LDA	0.824074	0.852941	0.820755	0.296055	0.824700
3	3	KNeighborsClassifier	0.819444	0.865979	0.811594	0.277530	0.820497
0	0	LogisticRegression	0.819444	0.858586	0.813397	0.277530	0.820326
5	5	Gussian NB	0.819444	0.844660	0.816901	0.277530	0.819983
2	2	DecisionTreeClassifier	0.791667	0.777778	0.801762	0.166381	0.790995

The results are as follows:

The results shown above indicated that we had the exact same result from Voting Classifier

as AdaBoosting.

Project Trajectory, Results and Interpretation

Changes in the Project

The only tangible changes in the project were related to the missing values for the feature *chol* in the Switzerland dataset. The missing values in the raw Switzerland dataset were

recorded as ‘0’ values rather than ‘?’ or blanks. The ‘0’ values were replaced with NaN values and then imputed using the imputation procedure as mentioned previously in this report to produce a new dataset. The addition of the *chol* values using imputation improved the results in terms of prediction accuracy using *Random Forest Classifier* as the model for imputation in the KNN and Iterative Imputation methods. This proved that abundance of data can produce better and more effective results.

Analysis and Results

Comparison of Baseline and Optimized Models

Baseline Models

	ModelName	logloss	Test accuracy	precision	F1	r2	AUC
0	Linear Discriminant Analysis : LDA	5.597	0.838	0.864	0.836	0.352	0.839
0	Random Forest Classifier : RFC	5.757	0.833	0.849	0.833	0.333	0.834
0	Ada Boosting	5.916	0.829	0.854	0.826	0.315	0.829
0	Logistic Regression : LR	6.236	0.819	0.838	0.819	0.278	0.820
0	KNN	6.236	0.819	0.838	0.819	0.278	0.820
0	GaussianNB : NB	6.236	0.819	0.845	0.817	0.278	0.820
0	XG Boosting	6.396	0.815	0.843	0.811	0.259	0.815
0	bagging Classifier	6.396	0.815	0.830	0.815	0.259	0.815
0	SVM	6.716	0.806	0.821	0.806	0.222	0.806
0	Decision Tree Classifier	7.036	0.796	0.806	0.798	0.185	0.796

Optimized Models

	index	ModelName	Test_accuracy	precision	F1	r2	AUC
8	8	AdaBoostClassifier	0.842593	0.865385	0.841121	0.370154	0.843053
10	10	Voting(LDA,Ada,SVM)	0.842593	0.865385	0.841121	0.370154	0.843053
4	4	XGBoosting	0.833333	0.862745	0.830189	0.333105	0.833962
7	7	RandomForestClassifier	0.833333	0.849057	0.833333	0.333105	0.833619
1	1	SVM	0.828704	0.847619	0.827907	0.314580	0.829074
9	9	Voting(LDA+Ada+XGB+SVM)	0.824074	0.867347	0.817308	0.296055	0.825043
6	6	LDA	0.824074	0.852941	0.820755	0.296055	0.824700
3	3	KNeighborsClassifier	0.819444	0.865979	0.811594	0.277530	0.820497
0	0	LogisticRegression	0.819444	0.858586	0.813397	0.277530	0.820326
5	5	Gussian NB	0.819444	0.844660	0.816901	0.277530	0.819983
2	2	DecisionTreeClassifier	0.791667	0.777778	0.801762	0.166381	0.790995

When comparing the Baseline Models to the Optimized Models, the analysis results are as follows:

- *Linear Discriminant Analysis* produced **better** results when the hyperparameters were **not** optimized
- *Random Forest Classifier* produced the **same** results with and without hyperparameter tuning
- *Ada Boosting* produced **better** results when the the hyperparameters **were** optimized
- *Logistic Regression* produced the **same** results with and without hyperparameter tuning
- *KNN Classifier* produced the **same** results with and without hyperparameter tuning
- *Gaussian Naive Bayes* produced the **same** results with and without hyperparameter tuning
- *XG Boosting* produced **better** results when the hyperparameters **were** optimized
- *Support Vector Machines* produced **better** results when the hyperparameters **were** optimized
- *Decision Tree Classifier* produced **better** results when the hyperparameters were **not** optimized

When comparing the *Voting Classifier Ensembles* to the *Optimized Models:*

- The *Voting Classifier* consisting of *Linear Discriminant Analysis, XGBoosting, AdaBoosting, and SVM* had the same prediction accuracy as the *optimized Linear Discriminant Analysis* (0.824074)
- The *Voting Classifier* consisting of *Linear Discriminant Analysis, AdaBoosting, and SVM* had the same prediction accuracy as the *optimized AdaBoosting* (0.842593)
- The *Voting Classifier* consisting of *LDA, AdaBoosting, and SVM* performed better.

- Once again, the *Voting Classifier* was only extracting values from *AdaBoosting*. Therefore, the best performing model for this classification problem was

AdaBoosting with hyperparameters: `base_estimator: RandomForest(bestparameters), n_estimators=100`

Interpretation of the Results

To classify the presence of heart disease in a patient for this project, our analysis was based on building a model that would successfully predict heart disease using patient attributes. For this project, we built 10 baseline models, nine optimized models, and two voting ensembles to investigate which model would have the capability to most effectively predict the presence of heart disease in a patient for the given attributes.

The prediction accuracy measure was prioritized in making the decision for this project. After hyperparameter tuning, the most effective models (in terms of prediction accuracy) in predicting the classification of presence of heart disease were *AdaBoosting (0.842593), Linear Discriminant Analysis (0.824074), Random Forest Classifier (0.83333), and XG Boosting (0.83333).* The *Voting Classifier* consisting of *Linear Discriminant Analysis, XGBoosting, AdaBoosting, and SVM (0.824074)* was not as effective as the most effective optimized model *AdaBoosting* and the most effective *Voting Classifier* that consisted of *LDA, AdaBoosting, and SVM (0.842593).*

Since there were not any gains in performance in terms of prediction accuracy using a voting ensemble, the best performing model for this classification problem was ***AdaBoosting*** with hyperparameters:

`base_estimator: RandomForest(best parameters), n_estimators=100`

Measuring the success of the Project

The goal of this project was to successfully predict the classification of presence of heart disease in a patient using the given attributes in the dataset. For a given optimized model, we wanted to achieve at least 84% prediction accuracy. In other words, in the allotted time frame, our goal was to successfully predict the classification of heart disease 84% of the time. After tuning the hyperparameters *AdaBoosting Classifier,* the model was able to achieve a mean prediction accuracy of 84%. Although, there are endless possibilities in terms of using different models and techniques to increase the accuracy when it comes to classification problems.

Final Conclusion and Future Work

Overall, for this project, we were able to reach the target that we were aiming for in the given time frame. Starting with data collection from the repository, incomplete datasets, imputing the missing feature values, and all the way to then building optimized models on imputed datasets to achieve 84% prediction accuracy using *AdaBoosting* gave us confidence that we can build models on healthcare datasets in the future to contribute to the healthcare field to make more informed decisions. Even though we achieved our goal with regards to prediction accuracy, there were still some impediments along the project that we could have avoided or improved on, e.g.:

- Numerous missing values in the datasets
- Locations that the data was collected from was from various locations. Data from one location (e.g. State of Florida), would provide more accurate results and a more confident decision making ability based on results for that location.
- Due to the time constraint, fewer rather than various experimentations could be performed (e.g. Deep Learning, Artificial Neural Networks, and HyperParameter tuning using other libraries).

In conclusion, for this dataset, *AdaBoosting* would successfully classify the presence of heart disease accurately 84% of the time if we were to have their attributes (*age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca*, and *thal*). This could prove useful, as if a patient classifies as having heart disease and they actually don't have heart disease, that would mean they are at a high risk of having heart disease. The controllable features are cholesterol and blood pressure that can controlled with medicine and healthy lifestyle but age and sex are not controllable features. However, the statistics do show men to be at a higher risk for heart disease.