Build your own Machine Learning Profile Project: Heart disease classification

I. Definition

Project overview

The World Health Organization (WHO) has reported that Cardiovascular diseases (CVDs) stand as the foremost global cause of mortality. In 2019, an estimated 17.9 million individuals succumbed to CVDs, accounting for 32% of all worldwide fatalities. Of these fatalities, 85% were attributed to heart attacks and strokes. Among the 17 million untimely deaths (occurring before the age of 70) resulting from noncommunicable diseases in 2019, 38% were directly linked to CVDs [1]. Heart failure is a common occurrence stemming from CVDs. The utilization of Machine Learning models to predict heart diseases has the potential to significantly mitigate Cardiovascular risk [2].

The dataset is sourced from Kaggle [2] and it was created by combining different datasets already available independently but not combined before. In this dataset, 5 heart datasets are combined over 11 common features which makes it the largest heart disease dataset available so far for research purposes. The five datasets used for its curation are:

- Cleveland: 303 observations.
- Hungarian: 294 observations.
- Switzerland: 123 observations.
- Long Beach VA: 200 observations.
- Stalog (Heart) Data Set: 270 observations.

Total: 1190 observations.
Duplicated: 272 observations.
Final dataset: 918 observations.

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

Feature information:

- Age: age of the patient [years]
- Sex: sex of the patient [M: Male, F: Female]
- ChestPainType: chest pain type [TA: Typical Angina, ATA:
- Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
- RestingBP: resting blood pressure [mm Hg]
- Cholesterol: serum cholesterol [mm/dl]

- FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
- RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
- MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
- ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
- Oldpeak: oldpeak = ST [Numeric value measured in depression]
- ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
- HeartDisease: output class [1: Heart disease, 0: Normal]

Problem statement

The data is structured in a tabular format, and the problem is a classification task, specifically a binary classification problem with two classes: 0 ("Normal") and 1 ("Heart disease"). With a binary classification, there are many potential approaches: Logistic Regression, Random Forest Classifier, LightGBM, ...

In addressing this problem, I employed a variety of algorithms to determine the optimal solution. The key stages I encompassed include data preparation, exploratory data analysis (EDA), feature engineering, model training, evaluation, and ultimately, model deployment.

<u>Platform</u>:

I utilized AWS SageMaker for various stages of this project, including data preparation, EDA, feature engineering, and model-related tasks, such as training, hyperparameter tuning and evaluation. Subsequently, I deployed the model using two distinct methods. The first entailed leveraging a AWS SageMaker endpoint and AWS Lambda function, while the second involved using the best model's weights and/or hyperparameters for deployment on the FastAPI framework locally.

<u>Algorithm</u>:

I explored various algorithms using Pycaret. I experimented with hyperparameter tuning using Optuna and also did feature engineering. The primary metric for Optuna optimization was the F1 score.

Metrics

Given the nature of this classification problem, I used both Accuracy and F1 score as metrics to assess the performance of models.

II. Analysis

Data Exploration

The first 5 rows of the data:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

Feature characteristic:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

→ This data does not have missing values.

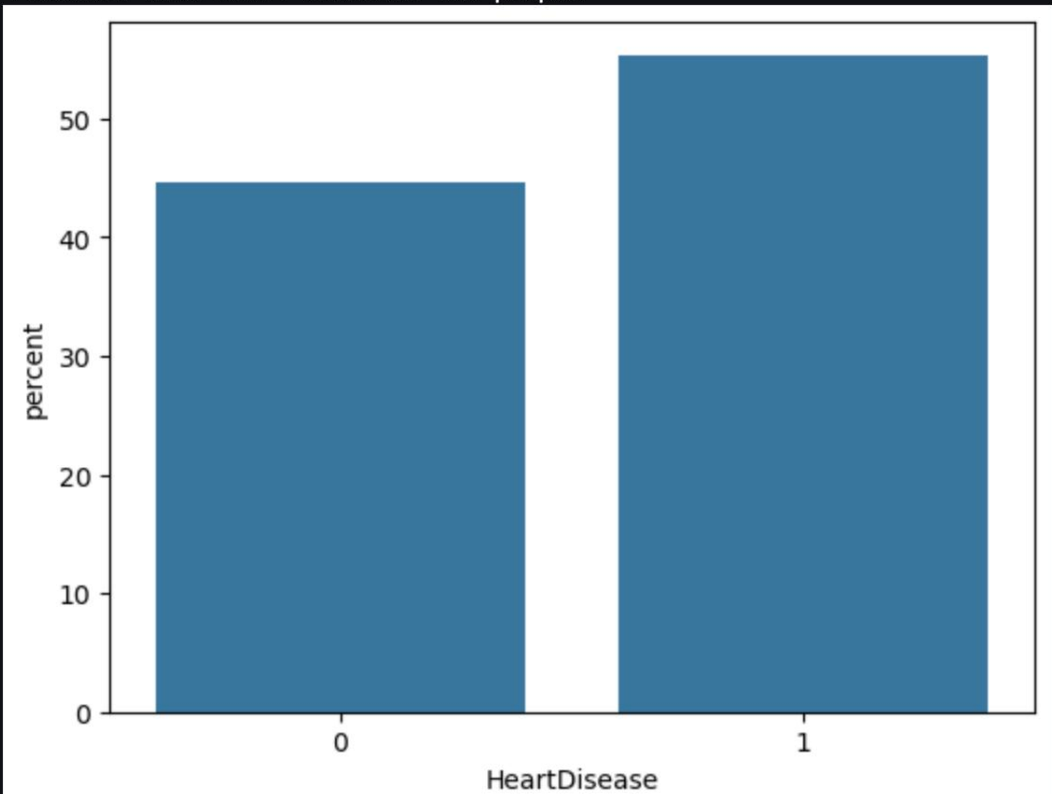
The number of unique values of each feature:

```
Number of unique values of feature Age with type int64 is 50
Number of unique values of feature Sex with type object is 2
Number of unique values of feature ChestPainType with type object is 4
Number of unique values of feature RestingBP with type int64 is 67
Number of unique values of feature Cholesterol with type int64 is 222
Number of unique values of feature FastingBS with type int64 is 2
Number of unique values of feature RestingECG with type object is 3
Number of unique values of feature MaxHR with type int64 is 119
Number of unique values of feature ExerciseAngina with type object is 2
Number of unique values of feature Oldpeak with type float64 is 53
Number of unique values of feature ST_Slope with type object is 3
Number of unique values of feature HeartDisease with type int64 is 2
```

Numerical features: Age, RestingBP, Cholesterol, FastingBS, MaxHR, Oldpeak.
Categorical features: Sex, FastingBS, ChestPainType, RestingECG, ExerciseAngina, ST_Slope.

Label distribution:

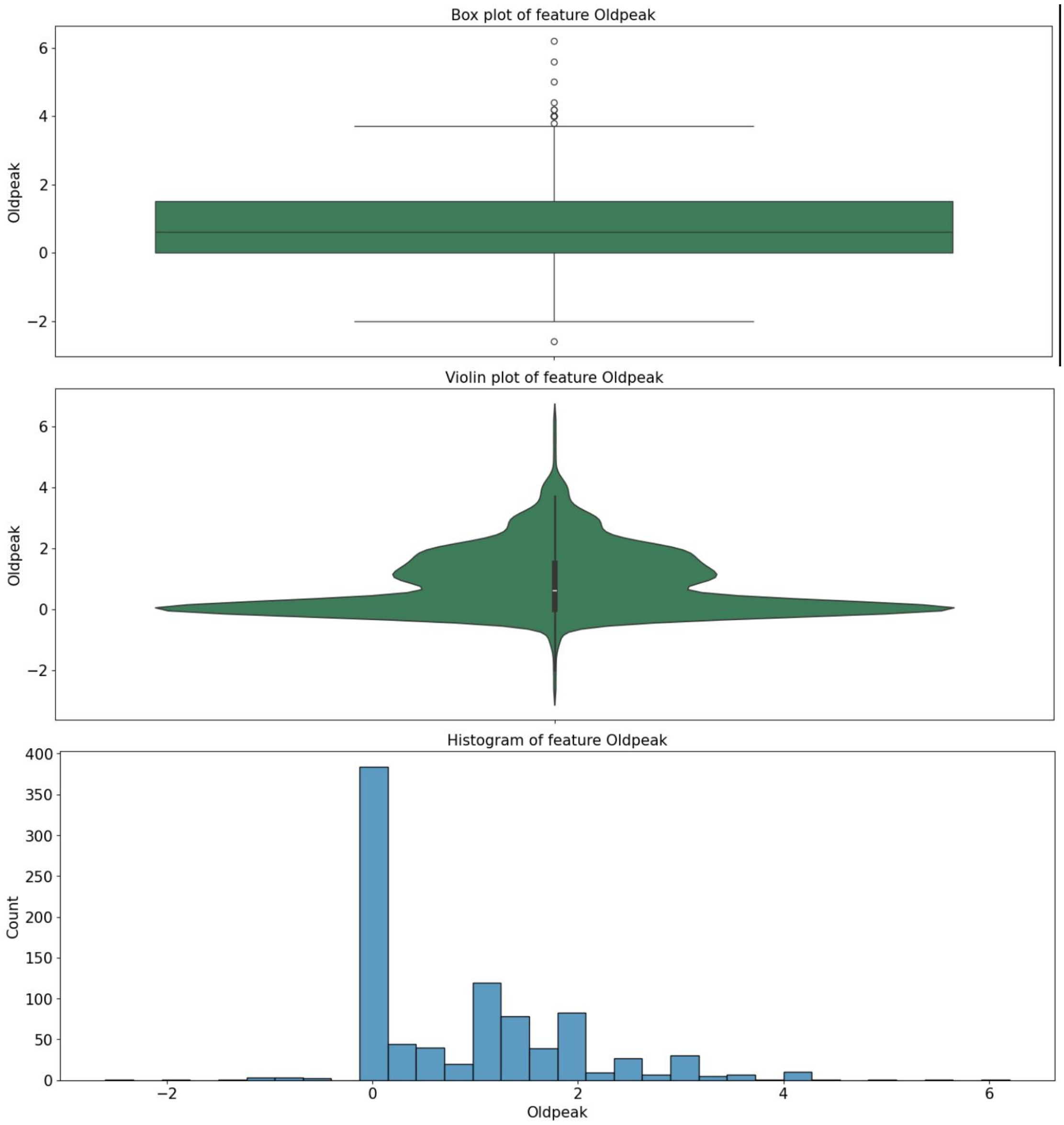
```
Patients with normal health condiontion: 410 people
Patients with heart disease: 508 people
```

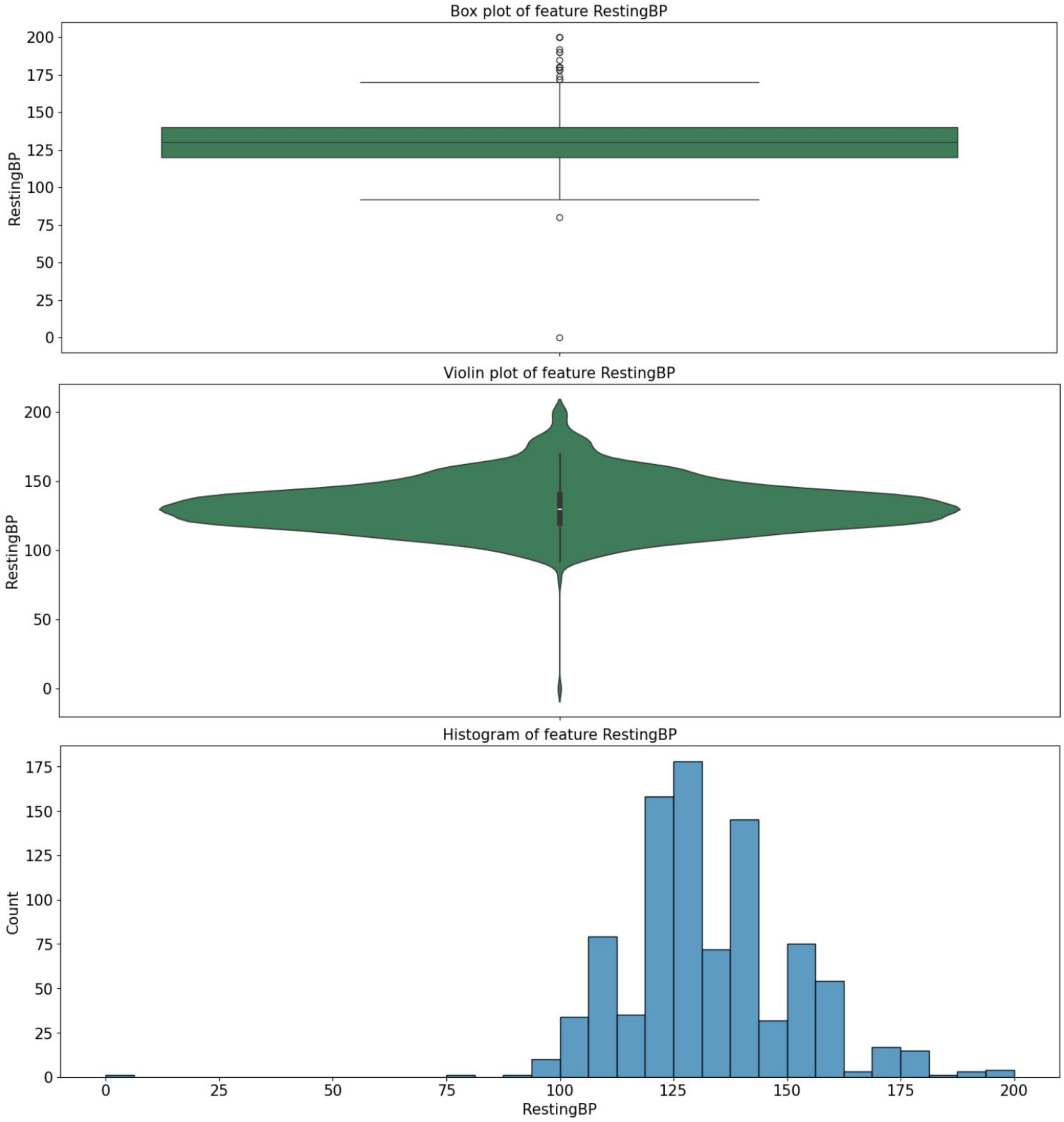


→ The ratio of patients with heart disease to individuals with normal health conditions is relatively balanced. I decided not to do undersampling or oversampling.

Exploratory visulization

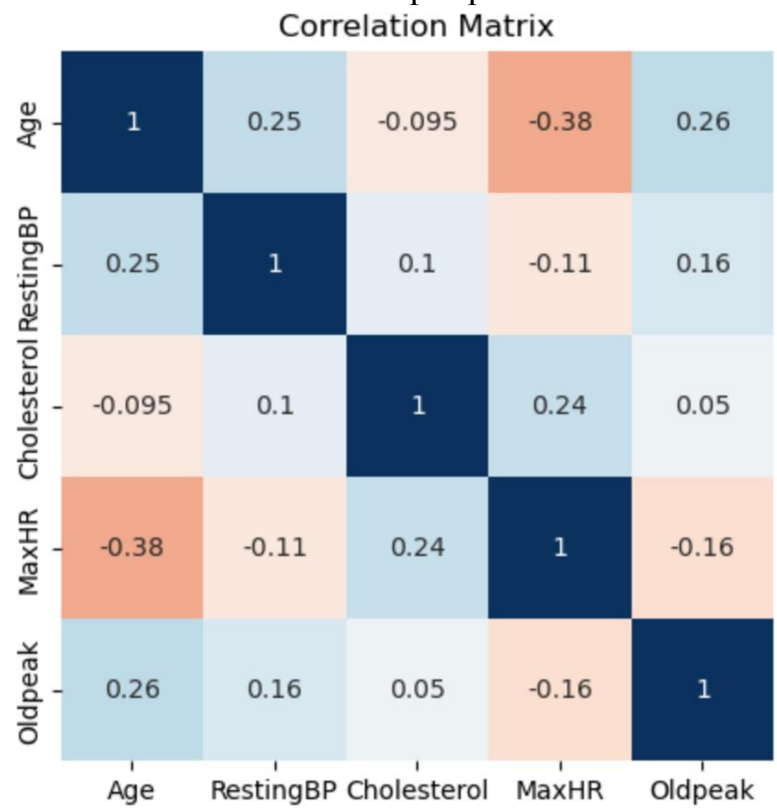
I created different interactive diagrams to understand the characteristic of all features. First, I explored numerical features. Here are the box, violin and histogram plots of feature Oldpeak and RestingBP:

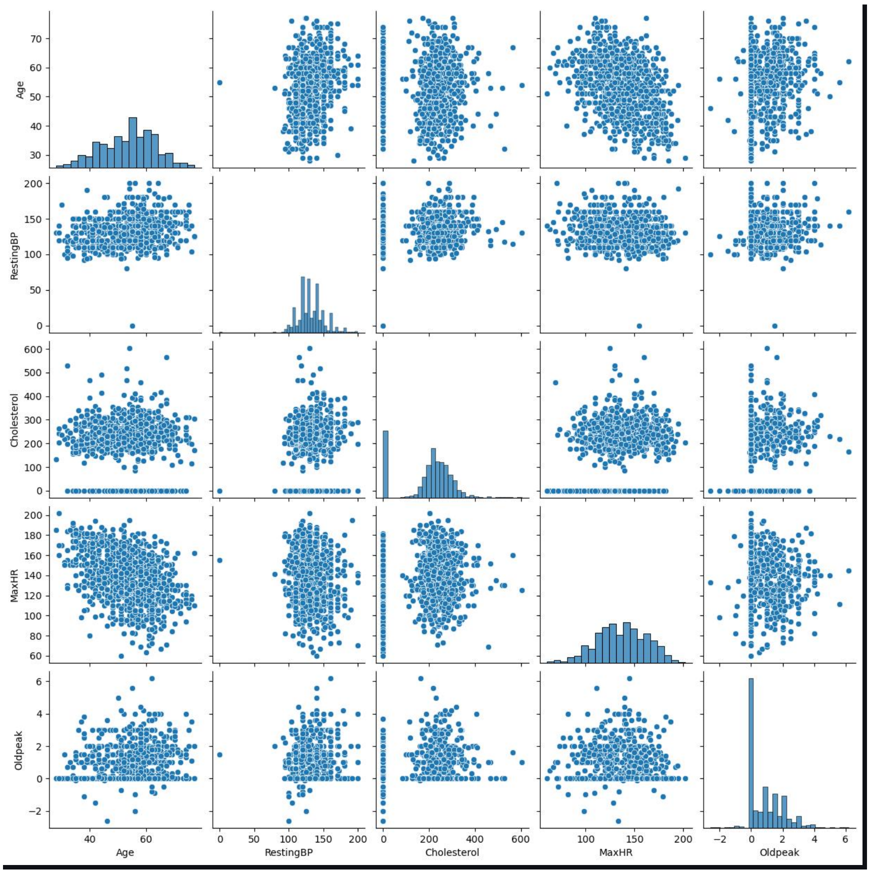




Box plot of feature RestingBP
RestingBP
0
25
50
75
100
125
150
175
200
Violin plot of feature RestingBP
RestingBP
0
50
100
150
200
Histogram of feature RestingBP
Count
0
25
50
75
100
125
150
175
0
25
50
75
100
125
150
175
200
RestingBP

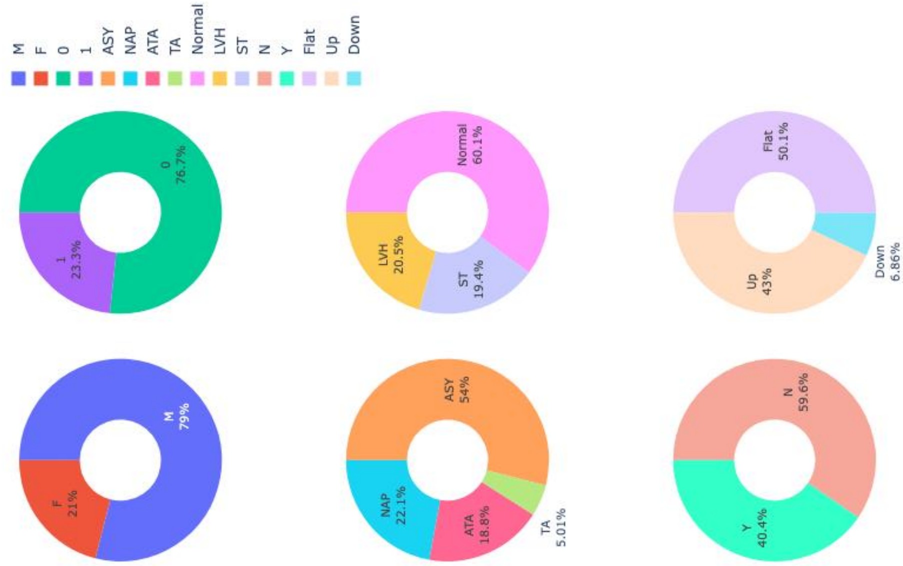
The correlation matrix and pairplot of numerical features:

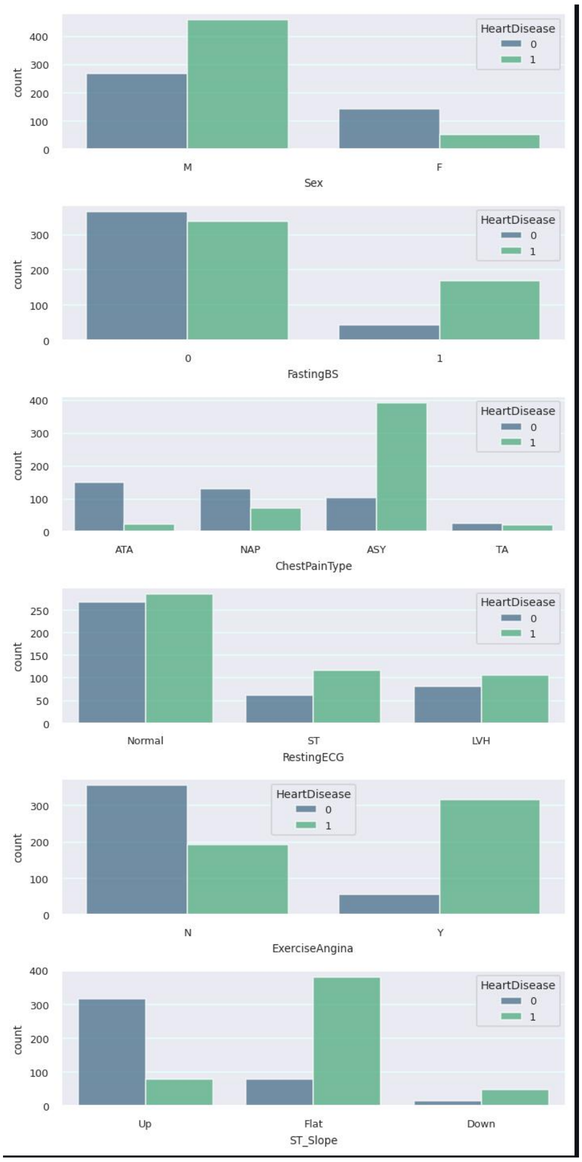




→ <u>The numerical features are not strongly correlated.</u>

Then I explored categorical features:





→

- The majority of patients with the disease are male.
- There are 4 types of ChestPain: TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic. Patients showing no symptoms (ASY) are the most vulnerable to the disease.
- People with exercise-induced angina tend to have a higher proportion of individuals with the disease, while those without exercise-induced angina tend to have a higher proportion of individuals in good health.

Algorithms and Techniques

All the exploration and visulization were done by Pandas, Matplotlib and Seaborn. This data is rather balanced and does not have missing values. The box plots reveal the presence of outliers in certain features, however I decided not to remove them.

Benchmark

In my proposal, I had originally planned to utilize the benchmark results obtained from this source [3]. However, upon attempting to replicate the author's approach (Logistic Regression from scratch), I achieved a notably improved outcome. The accuracy increased from approximately 55% to 83.04%. As a result, we had a new benchmark score:

- Accuracy: 83.04%
- F1-score: 86.12%

III. Methodology

Data preprocessing

I employed the LabelEncoder from Scikit-Learn to transform categorical features into numerical representations.

Implementation

After processing raw dataset, this is transformed dataset:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	1	1	140	289	0	1	172	0	0.0	2	0
1	49	0	2	160	180	0	1	156	0	1.0	1	1
2	37	1	1	130	283	0	2	98	0	0.0	2	0
3	48	0	0	138	214	0	1	108	1	1.5	1	1
4	54	1	2	150	195	0	1	122	0	0.0	2	0

Pipeline:

- Splitting train/test set: I used Scikit-Learn and utilized a 75% - 25% ratio for the division.

- Comparing models: I utilized Pycaret to find the best algorithm. The objective metric is F1-score (higher means better).

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
et	Extra Trees Classifier	0.8739	0.9292	0.9203	0.8759	0.8975	0.7339	0.7355	0.1700
gbc	Gradient Boosting Classifier	0.8609	0.9148	0.9130	0.8630	0.8873	0.7059	0.7078	0.1500
ada	Ada Boost Classifier	0.8565	0.9012	0.8986	0.8671	0.8826	0.6984	0.6991	0.1000
rf	Random Forest Classifier	0.8522	0.9144	0.8913	0.8662	0.8786	0.6898	0.6902	0.2100
nb	Naive Bayes	0.8478	0.9075	0.8841	0.8652	0.8746	0.6812	0.6815	0.0000
lightgbm	Light Gradient Boosting Machine	0.8478	0.9212	0.8696	0.8759	0.8727	0.6835	0.6836	0.0600
qda	Quadratic Discriminant Analysis	0.8391	0.9134	0.8913	0.8483	0.8693	0.6606	0.6619	0.0100
lr	Logistic Regression	0.8304	0.8974	0.8696	0.8511	0.8602	0.6448	0.6450	0.0900
ridge	Ridge Classifier	0.8304	0.8243	0.8551	0.8613	0.8582	0.6474	0.6474	0.0100
lda	Linear Discriminant Analysis	0.8304	0.8972	0.8551	0.8613	0.8582	0.6474	0.6474	0.0100
dt	Decision Tree Classifier	0.7957	0.7917	0.8116	0.8421	0.8266	0.5781	0.5787	0.0100
svm	SVM - Linear Kernel	0.7783	0.7772	0.7826	0.8372	0.8090	0.5455	0.5472	0.0100
knn	K Neighbors Classifier	0.7043	0.7283	0.7536	0.7536	0.7536	0.3841	0.3841	0.0000
dummy	Dummy Classifier	0.6000	0.5000	1.0000	0.6000	0.7500	0.0000	0.0000	0.0000

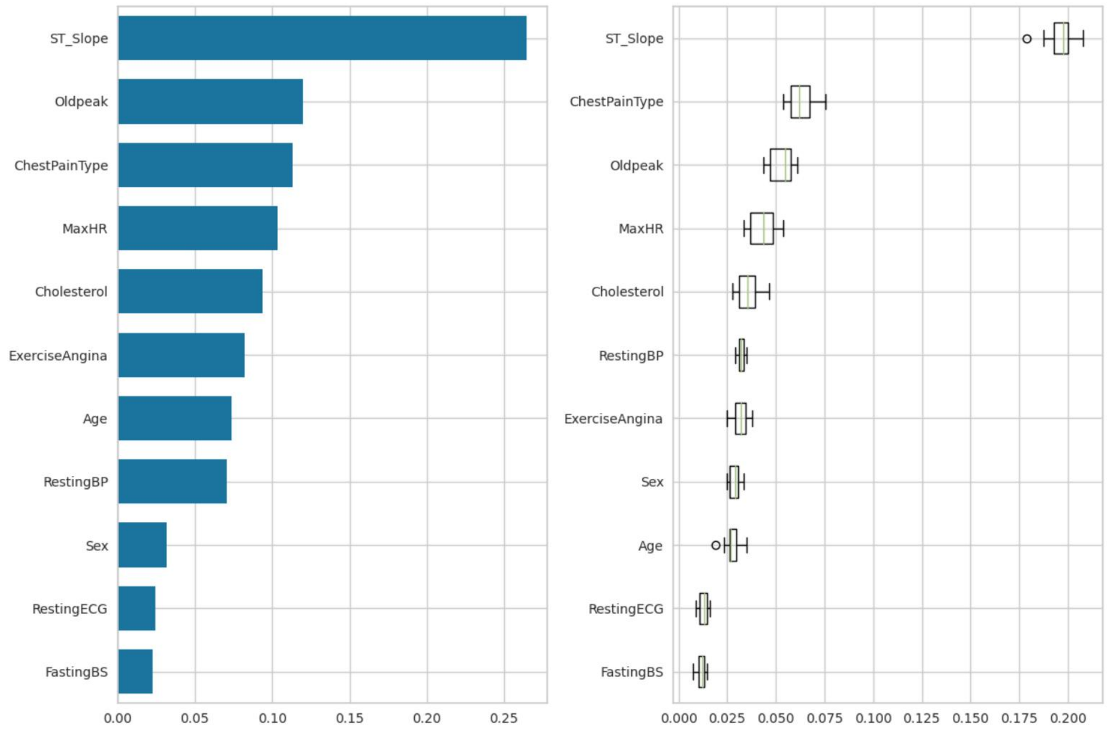
→ As depicted in the figure, Extra Trees Classifier achieved the highest accuracy (87.39%) and the best F1-score (89.75%).

- Tuning hyperparameters with Optuna: I endeavored to optimize the hyperparameters of the Extra Tree Classifier using Optuna; however, the performance score did not exhibit any enhancement.

```
The metrics before:
Accuracy: 0.8739130434782608
F1 score: 0.8975265017667845

The metrics after:
Accuracy: 0.8478260869565217
F1 score: 0.8780487804878048
```

- Feature engineering: I used Random Forest Classifier to asses the importance of individual features.



→ The least important features, such as Age, RestingBP, Sex, RestingECG, and FastingBS, were removed from the dataset. Subsequently, I fine-tuned the hyperparameters to evaluate whether this adjustment led to any potential improvement in model performance.

- Tuning again with Optuna: F1-score decreased to 85%.

→ In this particular scenario, retaining only the important features while removing the least significant ones did not result in any improvement.

- Deploying to an endpoint:

I created an estimator and deployed it to an endpoint:

```
predictor = sklearn_model.deploy(initial_instance_count=1, instance_type="ml.m5.large")
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /root/.config/sagemaker/config.yaml
----!
```

```
data = {
    "Age": [49, 40],
    "Sex": ["F", "M"],
    "ChestPainType": ["NAP", "ATA"],
    "RestingBP": [140, 140],
    "Cholesterol": [160, 170],
    "FastingBS": [0, 1],
    "RestingECG": ["Normal", "ST"],
    "MaxHR": [156, 147],
    "ExerciseAngina": ["N", "Y"],
    "Oldpeak": [0.0, 1.5],
    "ST_Slope": ["Flat", "Up"]
}

response = predictor.predict(data, initial_args={"ContentType": "application/json"})
response
```

```
[0, 1]
```

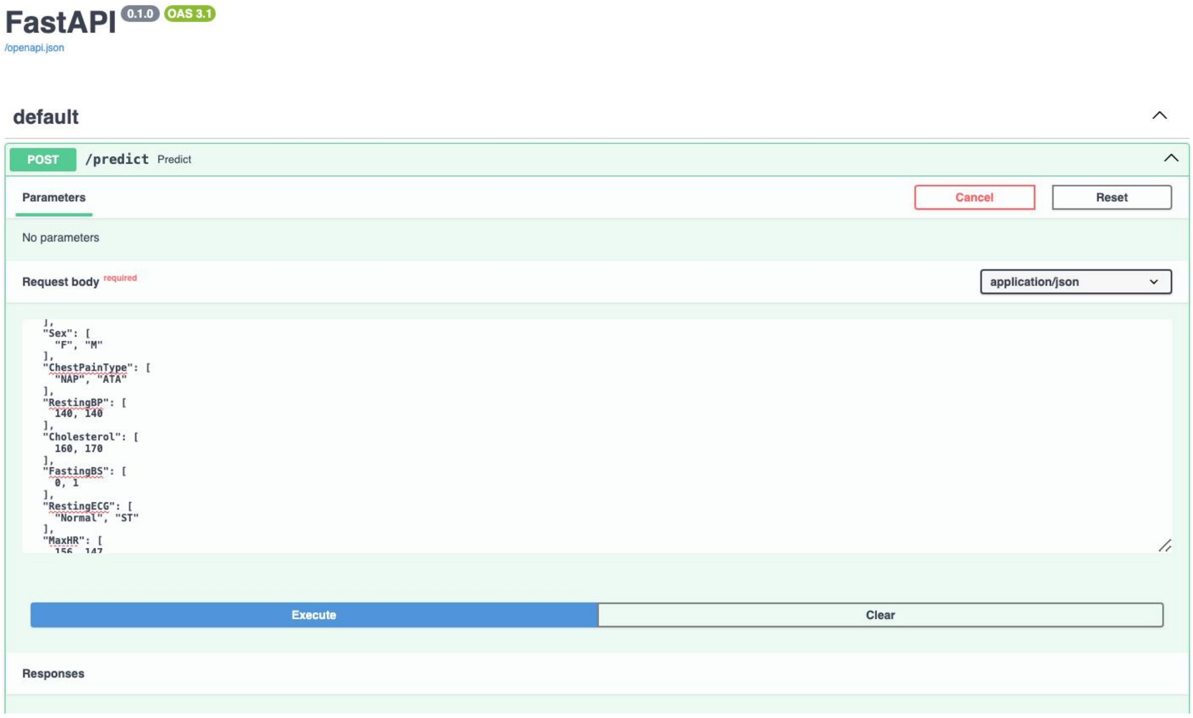
Additionally, I deployed it to an AWS Lambda function:

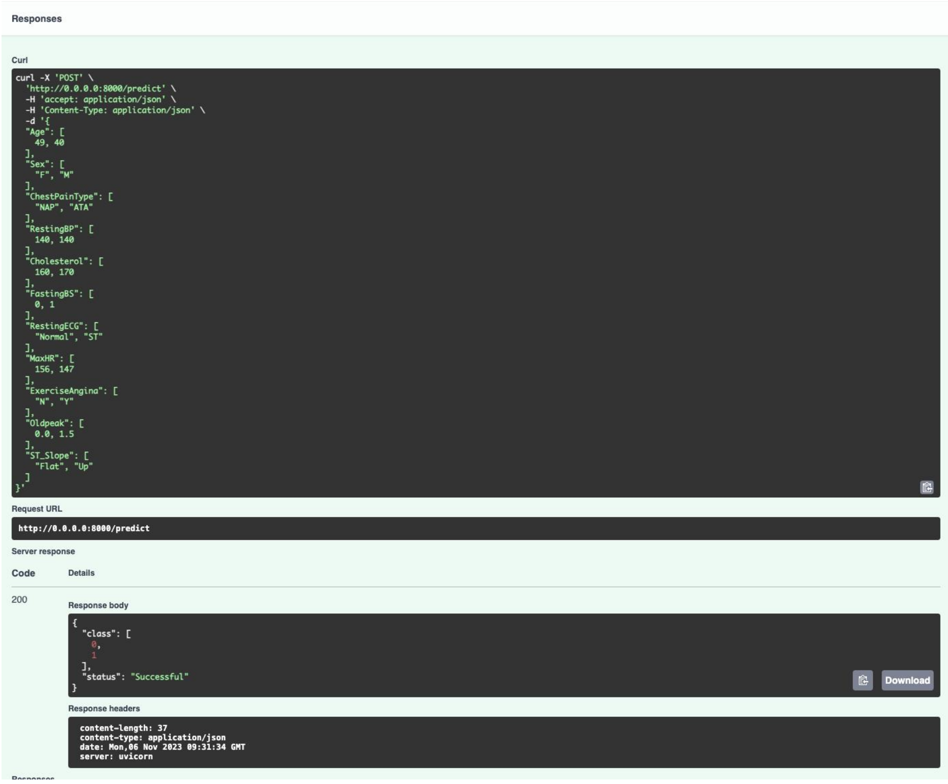
```
Test Event Name
sample-data

Response
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "COntent-Type-In": "LambdaContext([aws_request_id=d614b1f1-e64f-43d2-8026-cfa295f191be,log_group_name=/aws/lambda/heartDiseaseClassification,log_st
  "body": "[0, 1]"
}

Function Logs
START RequestId: d614b1f1-e64f-43d2-8026-cfa295f191be Version: $LATEST
Context:  LambdaContext([aws_request_id=d614b1f1-e64f-43d2-8026-cfa295f191be,log_group_name=/aws/lambda/heartDiseaseClassification,log_stream_name=20
EventType:  <class 'dict'>
END RequestId: d614b1f1-e64f-43d2-8026-cfa295f191be
REPORT RequestId: d614b1f1-e64f-43d2-8026-cfa295f191be  Duration: 304.56 ms Billed Duration: 305 ms Memory Size: 128 MB Max Memory Used: 67 MB  Init
```

- Deploying on FastAPI framework locally





Refinement

The process of hyperparameter tuning and feature engineering did not yield significant improvements in the metric scores. The most effective classifier is the Extra Tree Classifier from Pycaret, achieving an accuracy of 87.39% and an F1-score of 89.75%.

IV. Result

For this heart disease classification, Extra Tree Classifier outperformed other classifiers and the best hyperparameters identified are as follow:

```
▾                        ExtraTreesClassifier
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='sqrt',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=100, n_jobs=-1, oob_score=False,
                     random_state=125, verbose=0, warm_start=False)
```

Some other metrics on test set:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Extra Trees Classifier	0.8739	0.9292	0.9203	0.8759	0.8975	0.7339	0.7355

In comparision to benchmark model (Logistic Regression):

- Accuracy: 83.04%
- F1-score: 86.12%

Extra Tree Classifier delivered significantly improved performance:

- Accuracy: 87.39%
- F1-score: 89.75%

V. Conclusion

I have developed a classifier that exceeded the performance of the benchmark model and effectively deployed it using two methods: AWS Lambda functions and the FastAPI framework. While my implementation adequately addressed the problem, it is imperative to conduct additional assessments before deploying it in a production environment.

VI. Reference

[1]: World Health Organization. (n.d.). Cardiovascular diseases (cvds). World Health Organization. https://www.who.int/news-room/factsheets/detail/cardiovascular-diseases-(cvds)

[2]: Heart Failure Prediction Dataset. (n.d.). Kaggle: Your Machine Learning and Data Science Community. https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction/data

[3]: jiteshmd. (2023, January 24). Logistic_Regression_From-Scratch. Kaggle: Your Machine Learning and Data Science Community. https://www.kaggle.com/code/jiteshmd/logistic-regression-from-scratch