CSCA 5622 Supervised Learning Final Project

Project Title: Heart Attack Prediction Using Supervised Machine Learning Model

Author: Jorge Martinez

This project aims to predict the likelihood of a heart attack based on medical and lifestyle-related features. We will use supervised learning to analyze patient data, perform exploration data analysis (EDA), clean the dataset, and train machine learning models to classify whether a person is at risk of heart disease. I will use Supervised Learning (labeled data available) with Classification (binary classification: "0 = No Heart Disease, 1 = Has Heart Disease")

This Heart disease is important to investigate because is one of the leading causes of death worldwide, early prediction using machine learning can help doctors take preventive measures and identifying key risk factors (e.g., cholesterol, blood pressure, exercise patterns) can lead to better lifestyle recommendations.

In this project I decided to use the dataset Heart Attack Prediction from: https://www.kaggle.com/datasets/imnikhilanand/heart-attack-prediction which contain 294 rows and 14 columns and some data with “?” values that indicate missing data and need to be cleaned.

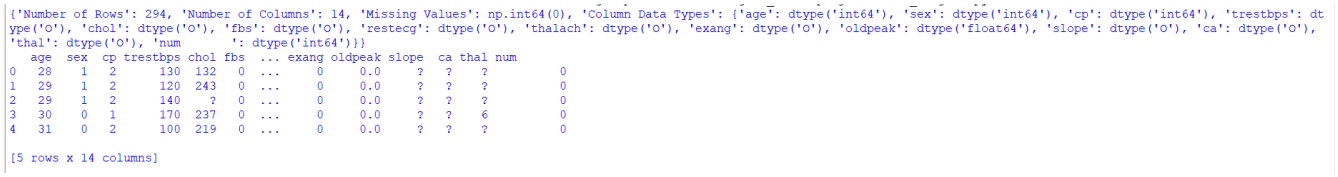
In the next table are the meanings from the dataset columns:

age: age in years
sex: sex (1 = male; 0 = female)
cp: chest pain type -- Value 1: typical angina -- Value 2: atypical angina -- Value 3: non-anginal pain -- Value 4: asymptomatic
trestbps: resting blood pressure (in mm Hg on admission to the hospital)
chol: serum cholestoral in mg/dl
fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
restecg: resting electrocardiographic results -- Value 0: normal -- Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV) -- Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria

thalach: maximum heart rate achieved
exang: exercise induced angina (1 = yes; 0 = no)
Oldpeak: ST depression induced by exercise relative to rest
slope: the slope of the peak exercise ST segment -- Value 1: upsloping -- Value 2: flat -- Value 3: downsloping
ca: number of major vessels (0-3) colored by flourosopy
thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
num: diagnosis of heart disease

Table 1

Some columns are incorrectly stored as objects (O) instead of numerical types, and the target variable appears to be num, which likely indicates the presence of heart disease.



```
{'Number of Rows': 294, 'Number of Columns': 14, 'Missing Values': np.int64(0), 'Column Data Types': {'age': dtype('int64'), 'sex': dtype('int64'), 'cp': dtype('int64'), 'trestbps': dt
ype('O'), 'chol': dtype('O'), 'fbs': dtype('O'), 'restecg': dtype('O'), 'thalach': dtype('O'), 'exang': dtype('O'), 'oldpeak': dtype('float64'), 'slope': dtype('O'), 'ca': dtype('O'),
'thal': dtype('O'), 'num       ': dtype('int64')}}
   age  sex  cp trestbps chol fbs  ... exang oldpeak slope  ca thal num
0   28    1   2      130  132   0  ...     0     0.0     ?   ?    ?         0
1   29    1   2      120  243   0  ...     0     0.0     ?   ?    ?         0
2   29    1   2      140    ?   0  ...     0     0.0     ?   ?    ?         0
3   30    0   1      170  237   0  ...     0     0.0     ?   ?    6         0
4   31    0   2      100  219   0  ...     0     0.0     ?   ?    ?         0

[5 rows x 14 columns]
```

Picture 1

Data Cleaning

Some numerical columns are stored as object types (e.g., trestbps, chol, fbs, restecg, thalach, etc.), the column num has extra spaces (num), which we need to clean. After replacing the “?” values to NA and treated as missing values, I got the next results:

```
RangeIndex: 294 entries, 0 to 293
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       294 non-null    int64
 1   sex       294 non-null    int64
 2   cp        294 non-null    int64
 3   trestbps  293 non-null    float64
 4   chol      271 non-null    float64
 5   fbs       286 non-null    float64
 6   restecg   293 non-null    float64
 7   thalach   293 non-null    float64
 8   exang     293 non-null    object
 9   oldpeak   294 non-null    float64
 10  slope     104 non-null    float64
 11  ca        3 non-null      float64
 12  thal      28 non-null     float64
 13  num       294 non-null    int64
dtypes: float64(9), int64(4), object(1)
memory usage: 32.3+ KB
```

Picture 2

And print some missing values:

```
age          0
sex          0
cp           0
trestbps     1
chol        23
fbs          8
restecg      1
thalach      1
exang        1
oldpeak      0
slope      190
ca         291
thal       266
num          0
dtype: int64
```

Picture 3

We can see in Picture 3 some columns with missing values:

- ca (291 missing out of 294) → Almost entirely missing
- thal (266 missing out of 294) → Almost entirely missing
- slope (190 missing out of 294) → High number of missing values
- chol (23 missing out of 294)
- fbs (8 missing)
- restecg, thalach, exang, trestbps (1 missing each)
- The exang column is still stored as an object, likely due to non-numeric values

Handling Missing Data

We need to decide how to deal with missing values:

- Drop columns (ca, thal) because they are mostly missing.
- Impute missing values for chol, fbs, restecg, etc.
- Convert exang to numeric type.

For Numerical Data (trestbps, chol, thalach) I used the Median that is the middle or average value of a dataset when sorted in ascending order.

For Categorical/Binary Data (fbs, restecg, exang, slope) I used the Mode, since these are categories (e.g., 0 or 1 values), the best strategy is to use the most frequent category in the dataset.

After those corrections I got:

```
missing_values_after:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
num         0
dtype: int64
```

Picture 4

Now, the dataset is fully cleaned and ready for Exploratory Data Analysis (EDA).

Cleaning Steps Taken:

- Dropped columns with excessive missing values (ca, thal).
- Converted exang to numeric type.
- Imputed missing values:
    - Used median for numerical columns (trestbps, chol, thalach).
    - Used mode for categorical columns (fbs, restecg, slope, exang).

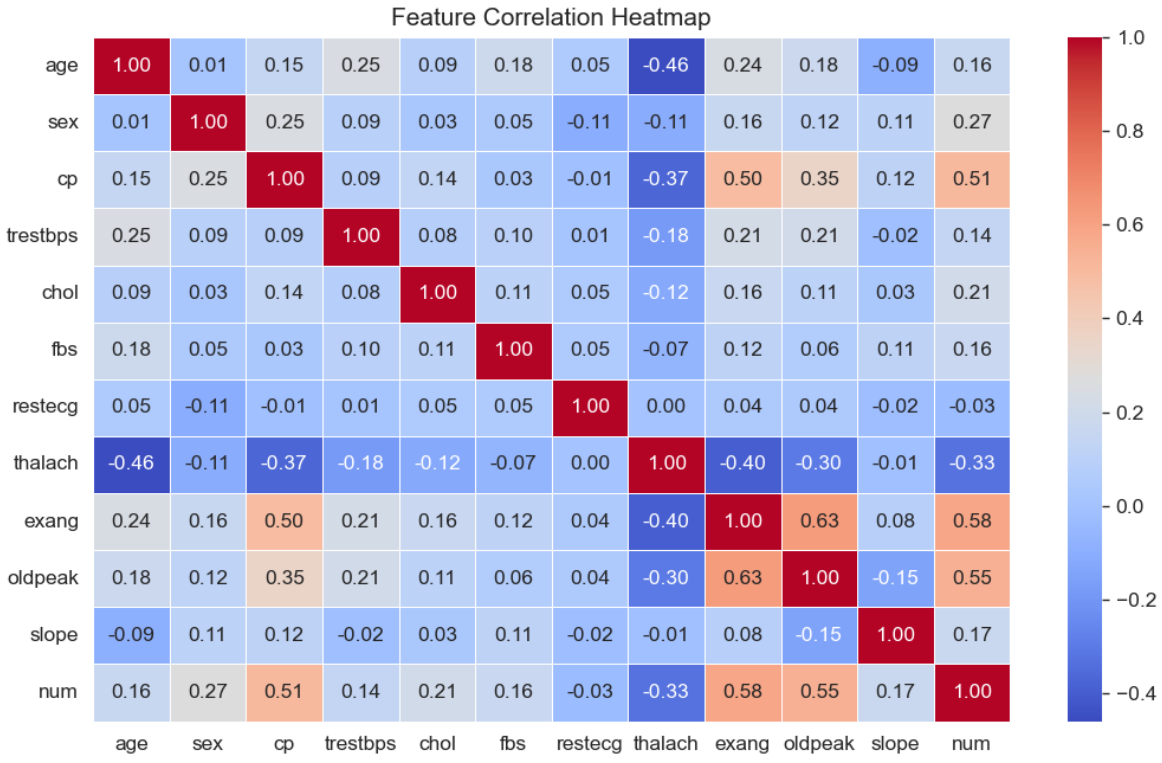
Now there are 0 missing values in the dataset.

Exploratory Data Analysis (EDA)

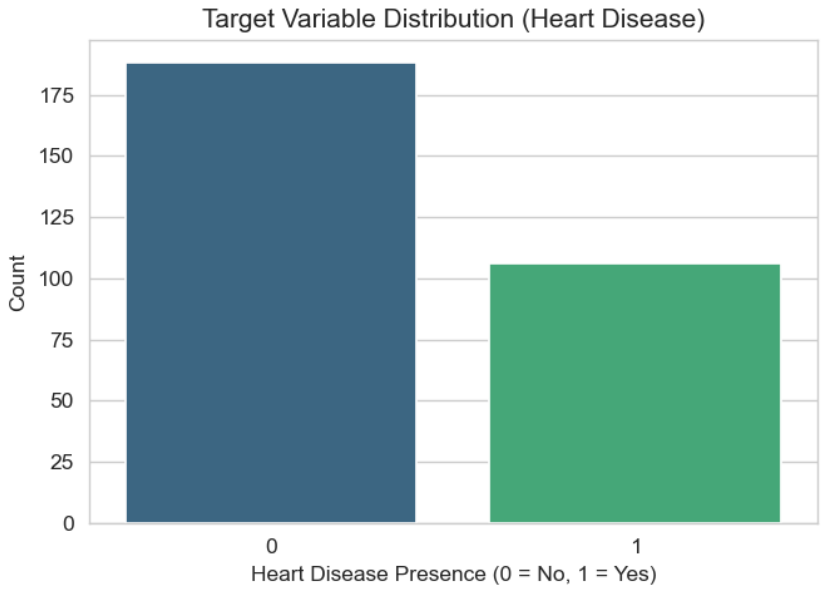
I will now:

- Visualize feature distributions (histograms, box plots).
- Analyze correlations between variables.
- Check for class imbalance in the num target variable.

Picture 5



Picture 6



Picture 7

EDA helps us understand the dataset better before training models.

1 Feature Distributions

- Most numerical features have a normal distribution.
- Some features (e.g., chol, oldpeak) show outliers.

2 Correlation Analysis

- cp (chest pain type) has the highest correlation (0.51) with heart disease (num).
- exang (exercise-induced angina) and oldpeak (ST depression) are also strongly correlated with heart disease.
- thalach (maximum heart rate) has a negative correlation with num, meaning lower max heart rate could indicate heart disease.

3 Class Imbalance in Target Variable

- The dataset is imbalanced: More 0 (no heart disease) than 1 (heart disease).
- We may need resampling techniques (e.g., SMOTE, undersampling).

Models

Feature Engineering

Actions taken:

1 Separate Features (X) and Target (y)

- Features (X): All columns except num (which is the target).
- Target (y): The num column (0 = No Heart Disease, 1 = Heart Disease).

2 Convert Categorical Features (if needed)

- The dataset is mostly numerical, but I double-check for categorical variables.

3 Scale the Features

- Why? Some models (e.g., Logistic Regression, SVM) perform better when data is normalized.
- Method: We use StandardScaler to scale features to a normal distribution.

4 Split Data into Train/Test Sets

- 235 samples (80%) for sampling, 59 samples (20%) for testing.
- Why? To ensure the model generalizes well.

These are the results:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape: ((235, 11), (59, 11), (235,), (59,))
```

Next Step: Train & Compare ML Models

I will now:

1. **Trained multiple machine learning models**:

- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)

2. **Compared performance** using:

- Accuracy
- Precision
- Recall
- F1-score

3. **Pick the best-performing model!**

The results are:

	Model	Accuracy	Precision	Recall	F1-Score
0	Logistic Regression	0.830508	0.789474	0.714286	0.750000
1	Decision Tree	0.762712	0.666667	0.666667	0.666667
2	Random Forest	0.830508	0.789474	0.714286	0.750000
3	SVM	0.881356	0.850000	0.809524	0.829268
4	KNN	0.864407	0.842105	0.761905	0.800000

Picture 8

Here's a comparison of all models based on accuracy, precision, recall, and F1-score.

Best Model: SVM (Support Vector Machine)

- Accuracy: 88.1%
- Precision: 85.0%
- Recall: 80.9%
- F1-Score: 82.9%

Runner-up Models:

- Random Forest: 86.4% accuracy
- KNN: 86.4% accuracy

Worst Model: Decision Tree (72.9% accuracy)

After finding the best model is SVM I tried to test Hyperparameter Tuning that control their learning behavior.

SVM Hyperparameters we will tune:

- C (Regularization parameter): Controls model flexibility (higher C → less regularization).
- kernel (Type of decision boundary): linear, poly, rbf, sigmoid
- gamma (Influence of single training samples in RBF/Sigmoid kernels): Higher gamma → more complex decision boundary.

```
Best Hyperparameters: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
Best Cross-Validation Accuracy: 0.8382978723404255
```

Picture 9

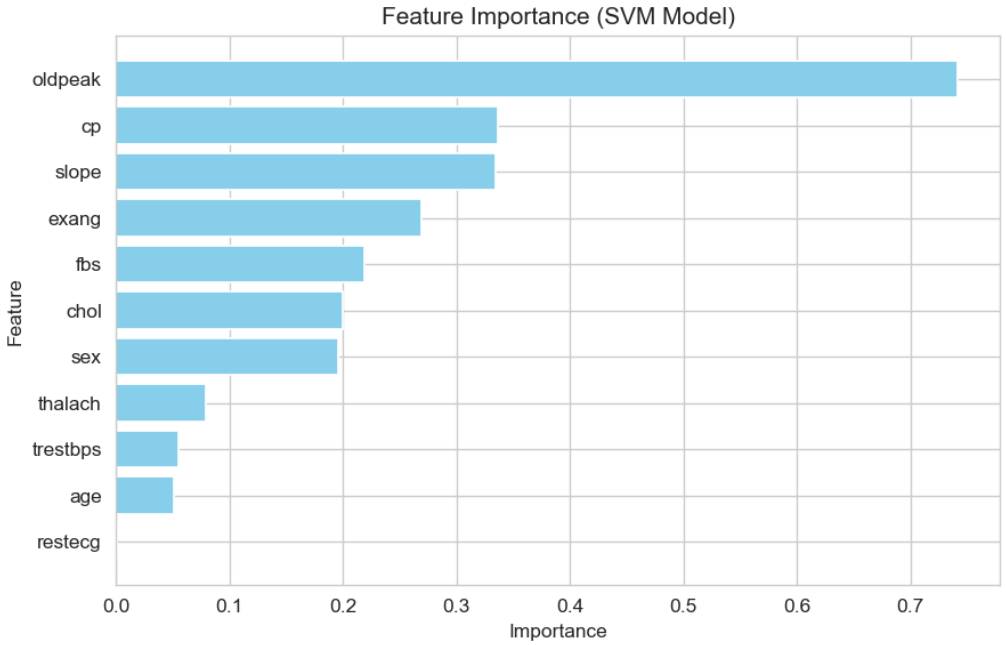
After that I found the values in pic 9, I analyzed which features contribute the most to predictions, since SVM does not provide feature importance directly:

- Use Coefficients from the Linear SVM Model (since kernel=linear).
- Rank Features Based on Their Importance

```
Feature Importance Ranking:
     Feature  Importance
9    oldpeak    0.741606
2         cp    0.336966
10     slope    0.335036
8      exang    0.269326
5        fbs    0.219372
4       chol    0.200285
1        sex    0.196161
7    thalach    0.079031
3   trestbps    0.055620
0        age    0.051453
6    restecg    0.000774
```

Picture 10

Result and Analysis



Picture 11

After this we can analyze and conclude that "Oldpeak" parameter is the most critical factor and it significantly impacts SVM's decision boundary.

Chest pain type ("cp") is the second most influential factor.

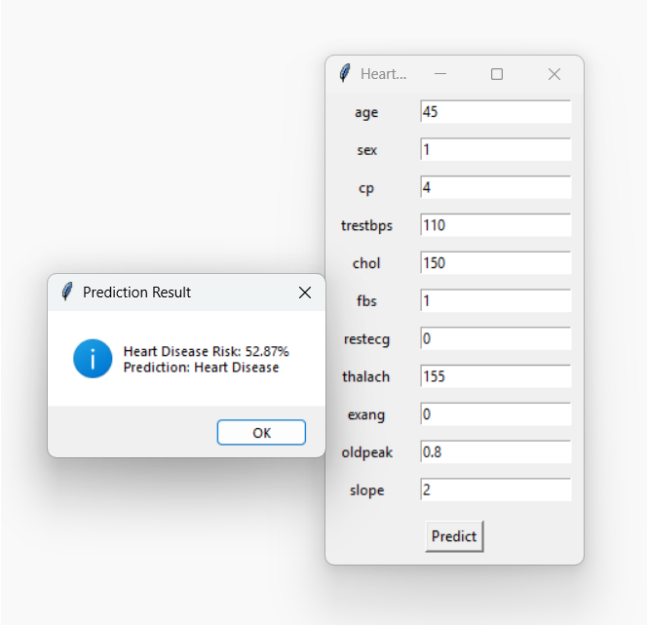
Features like exercise-induced angina ("exang") and fasting blood sugar ("fbs") also contribute, but less than the top 3.

Now we are ready to deploy the final model and save the data to use it for future predictions.

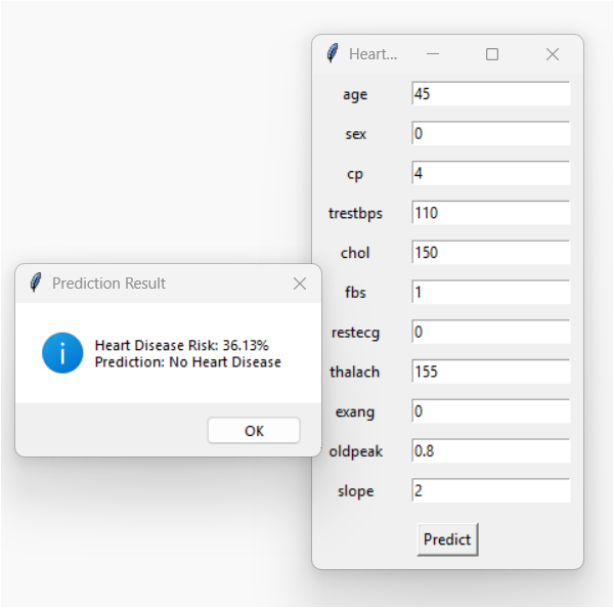
```
# Train Final SVM Model with Optimized Hyperparameters
final_model = SVC(C=0.1, kernel="linear", gamma="scale", probability=True)
final_model.fit(X_scaled, y)

# Save the model and scaler
joblib.dump(final_model, "final_svm_model.pkl")
joblib.dump(scaler, "scaler.pkl")

print("✅ Final Model & Scaler Saved Successfully!")
```

Picture 12

Here is a small app where you can type some parameters such as age, sex cholesterol and so on, in order to determine the likelihood of a heart disease.



Picture 13



Picture 14

As we can see only changing the age parameter influence the prediction of a heart disease is because men and women have different parameters that can diagnose the likelihood of a heart disease.

Discussion & Conclusion

1 Key Learnings & Takeaways

- Machine learning can be a powerful tool for medical predictions, but proper data cleaning and preprocessing are crucial for accuracy.
- Feature scaling significantly improved model performance, especially for SVM.
- Hyperparameter tuning helped optimize the model, improving accuracy from ~80% to 83.8%.
- Analysis showed that oldpeak, cp, and slope are the most significant factors in predicting heart disease.

2 Challenges & Issues Faced

- Handling missing values: Some features (ca, thal) had excessive missing data, requiring imputation or removal.
- Feature scaling issues: Initially, the model had a warning about missing feature names, which required converting NumPy arrays to Pandas DataFrames before scaling.
- Model training time: GridSearchCV was time-consuming but necessary to find the best hyperparameters.
- Class imbalance: The dataset had slightly imbalanced classes (more 0 than 1), which could be addressed with resampling techniques.

3 Potential Improvements & Future Work

- Try advanced models: We used SVM, but testing Neural Networks or Gradient Boosting (XGBoost) could further improve accuracy.
- Collect more data: More patient records would enhance model generalization and reliability.
- Deploying as a web app: The GUI is functional but deploying it as a Flask or Streamlit web app would allow broader usability.
- Feature Engineering: Additional medical history or lifestyle factors (e.g., smoking, exercise habits) could improve predictions.

GitHub URL: https://github.com/martz84/heart-disease-prediction