Predicting Heart Disease Risk Using Supervised Machine Learning

DTSA 5509

Final Project

Github Link

Project Scope – Predicting Heart Disease Risk Using Supervised Machine Learning

Heart failure is a critical medical condition where the heart cannot pump blood effectively to meet the body's needs. It affects millions globally and is a leading cause of morbidity and mortality. Early detection and management of heart failure can significantly improve patient outcomes, reduce healthcare costs, and enable timely interventions.

The objective of this project is to predict the likelihood of heart failure using a comprehensive dataset of patient health metrics and medical history. We aim to integrate clinical and demographic features, such as age, blood pressure, cholesterol levels, and lifestyle factors, to develop a predictive model capable of identifying individuals at high risk for heart failure.

Through exploratory data analysis, we will uncover trends and relationships within the data, providing insights into critical risk factors for heart failure. Subsequently, a random forest classifier will be employed as a robust predictive tool for early diagnosis.

Code for the visualizations provided below is accessible through the attached Jupyter notebook for this assignment.

About the Data

The dataset for this project was obtained from Kaggle's open data repository, contributed by Federico Soriano. The dataset is publicly available at the following URL: Heart Failure Prediction Dataset.

- 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]

This dataset was created by the author by combining different datasets that are already available independently, but not combined before. This dataset is comprised of over 11 common features listed above. The individual sources can be found on the author's Kaggle site.

Exploratory Data Analysis (EDA)

Here we will load the dataset and performance preliminary exploratory data analysis to get a better understanding of the data that we're working with.

We first start by downloading the dataset from Kaggle

```
# Download latest version
path = kagglehub.dataset_download("fedesoriano/heart-failure-prediction")
print("Path to dataset files:", path)

## Import dataset downloaded locally from Kaggle
dataset = pd.read_csv("heart.csv")
```

To make sure the dataset was loaded correctly, we load the first few lines of the dataset

Code 4

```
## Verify that the dataset has been uploaded correctly.

dataset.head()
```

	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

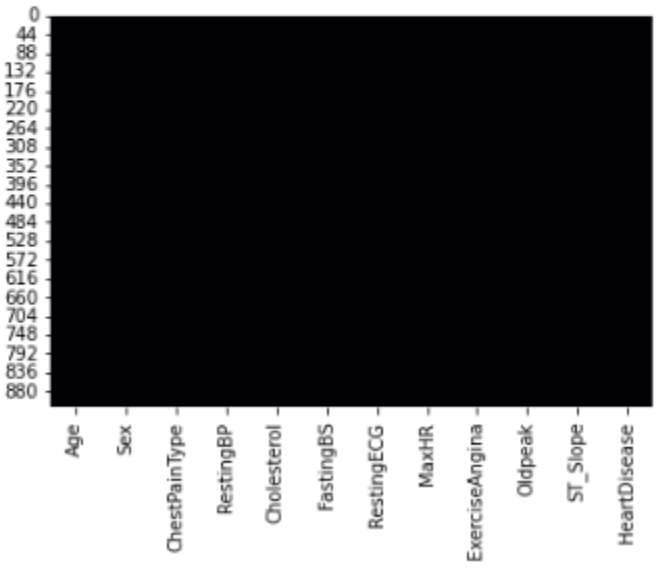
5 rows

We then inspect the dataset to determine if there are any values that are missing/null, using a heatmap to visualize:

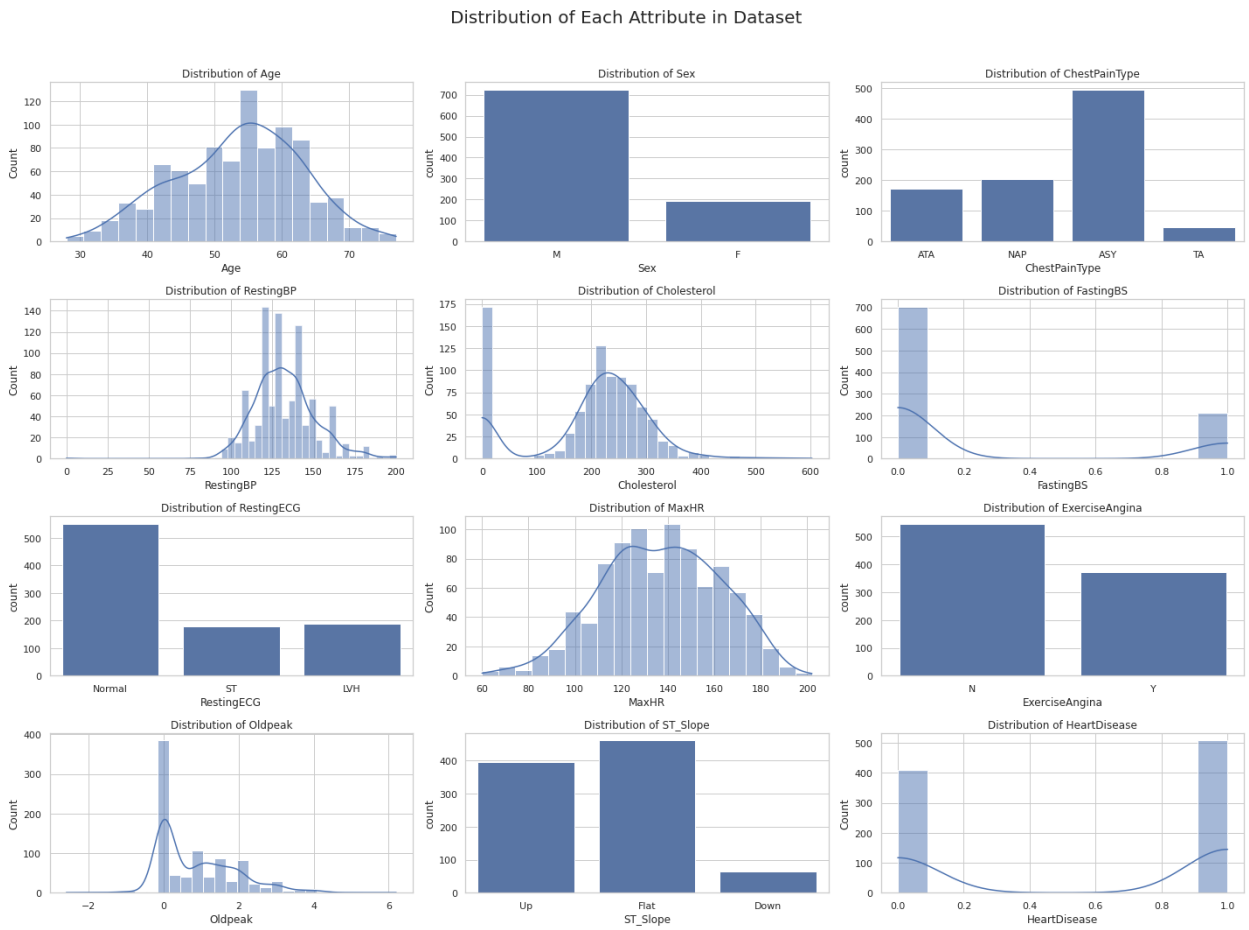
Heatmap Null Values

```
sns.heatmap(dataset.isnull(),cmap = 'magma',cbar = False);
```

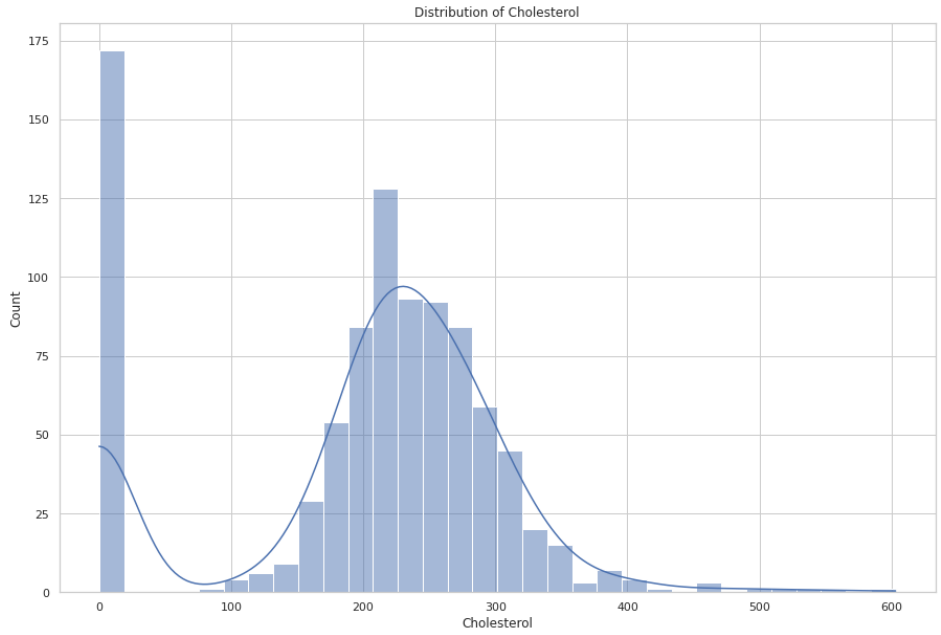
```
<Axes: >
```



We don’t see any missing data values – that’s good news. Now we inspect the distributions of each of the parameters for the dataset.



From the above distribution, we can derive there may be a potential issue with the dataset's completeness. The distribution of cholesterol below shows a high count of rows containing a value of 0, which is an invalid measure for cholesterol.



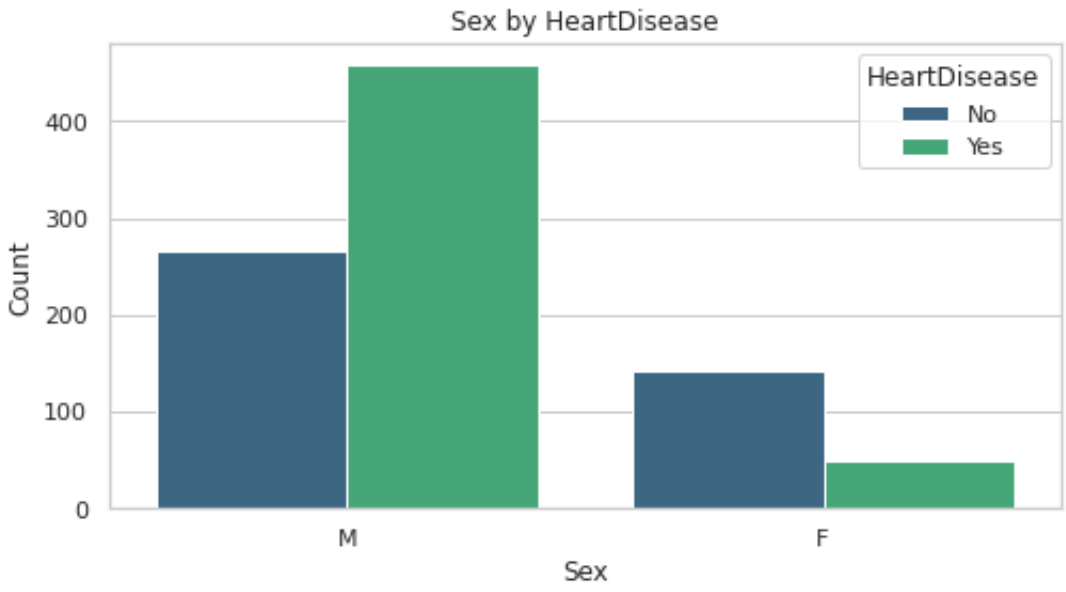
We can also see that there is a single individual with a 0 value for RestingBP (Resting Blood Pressure) that may be problematic. We will handle this in the data cleaning step.

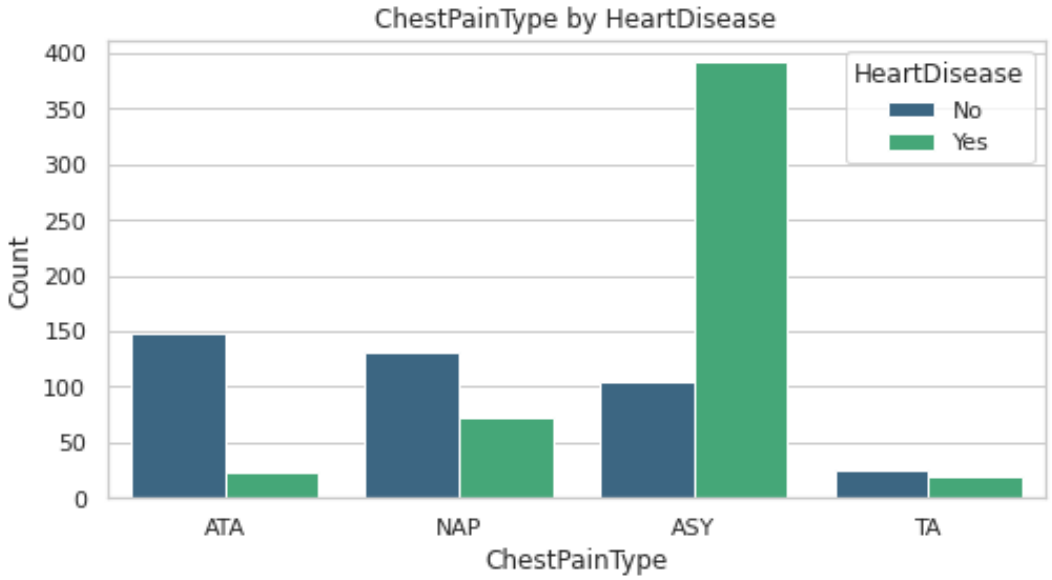
```
1  dataset[['RestingBP', 'Cholesterol']].loc[dataset['RestingBP'] == 0]
```

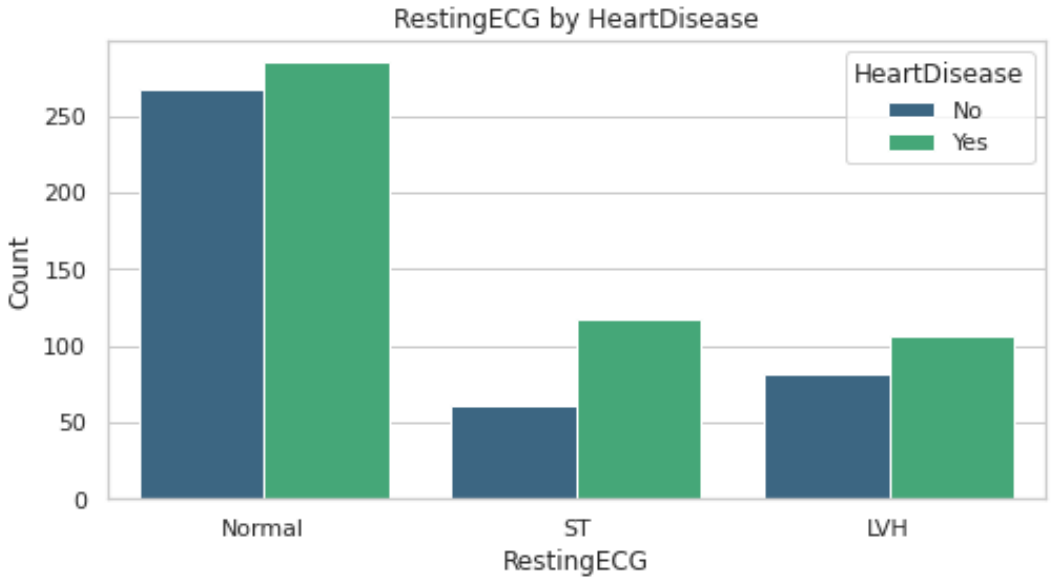
	RestingBP	Cholesterol
449	0	0

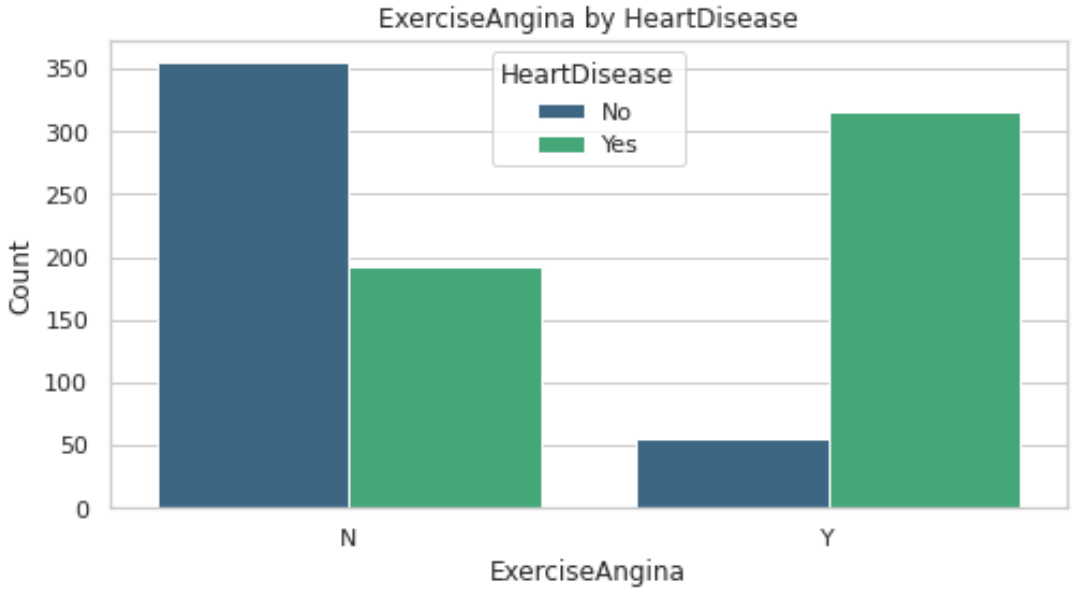
1 rows

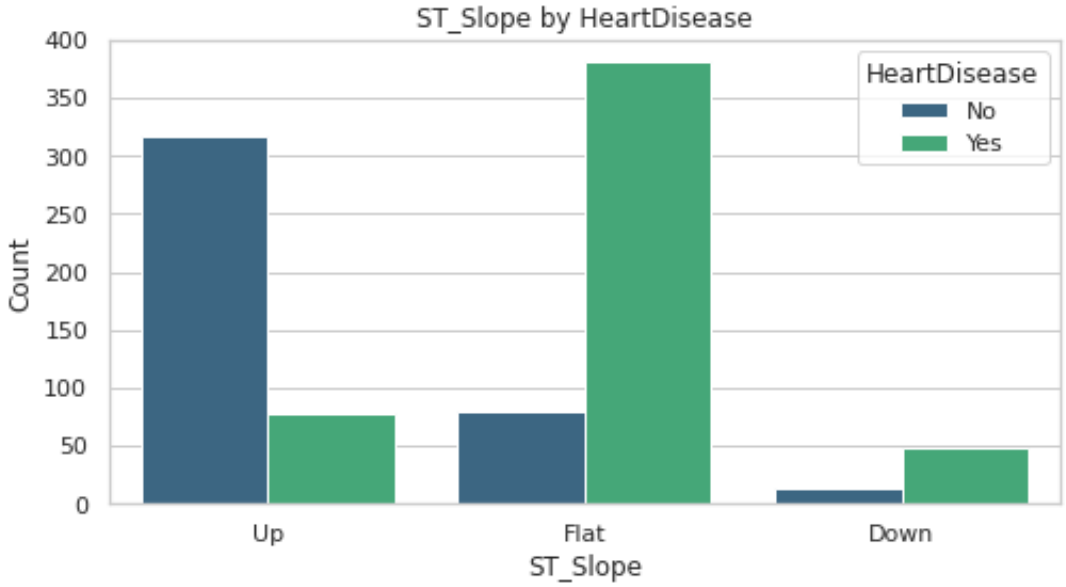
For categorical parameters, we can create a countplot with consideration of HeartDisease value for each individual.




ChestPainType by HeartDisease
HeartDisease
No
Yes
Count
400
350
300
250
200
150
100
50
0
ATA
NAP
ASY
TA
ChestPainType


RestingECG by HeartDisease
HeartDisease
No
Yes
Count
250
200
150
100
50
0
Normal
ST
LVH
RestingECG





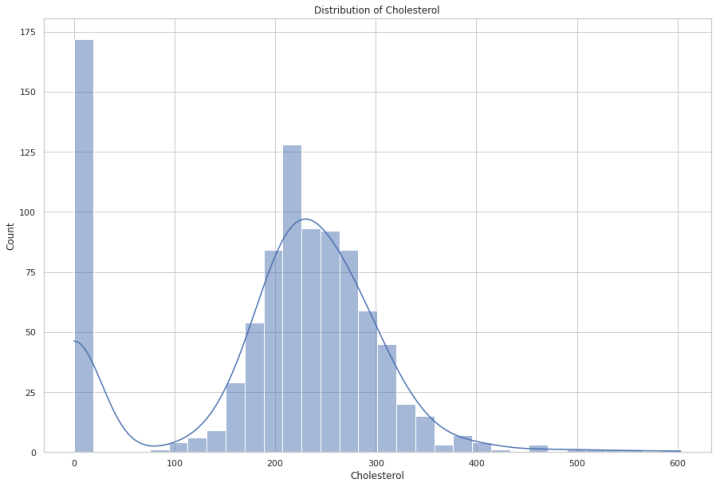
From the above plots, we can observe the following:

- Heart Disease appears more prevalent in males compared to females in the sample set
- A ChestPainType value of ASY correlates with HeartDisease.
- ExerciseAngina and ST_Slope both strongly correlated to HeartDisease outcome (Y & Flat, respectively.

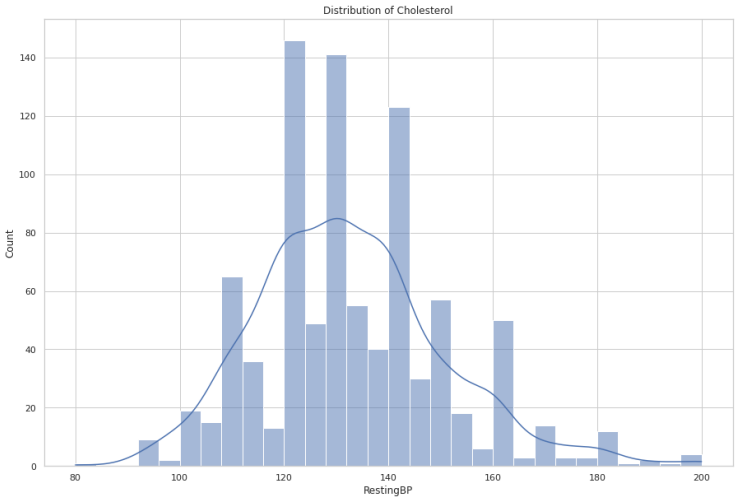
Data Cleaning/Preprocessing

We have identified areas where there are 0 values in both Cholesterol and RestingBP columns. To deal with this, we will first set the associated 0 values to np.nan, and then assigning the mean to replace them. While not perfect, this improves the dataset and resolves the lower value skew that impacts the data integrity:

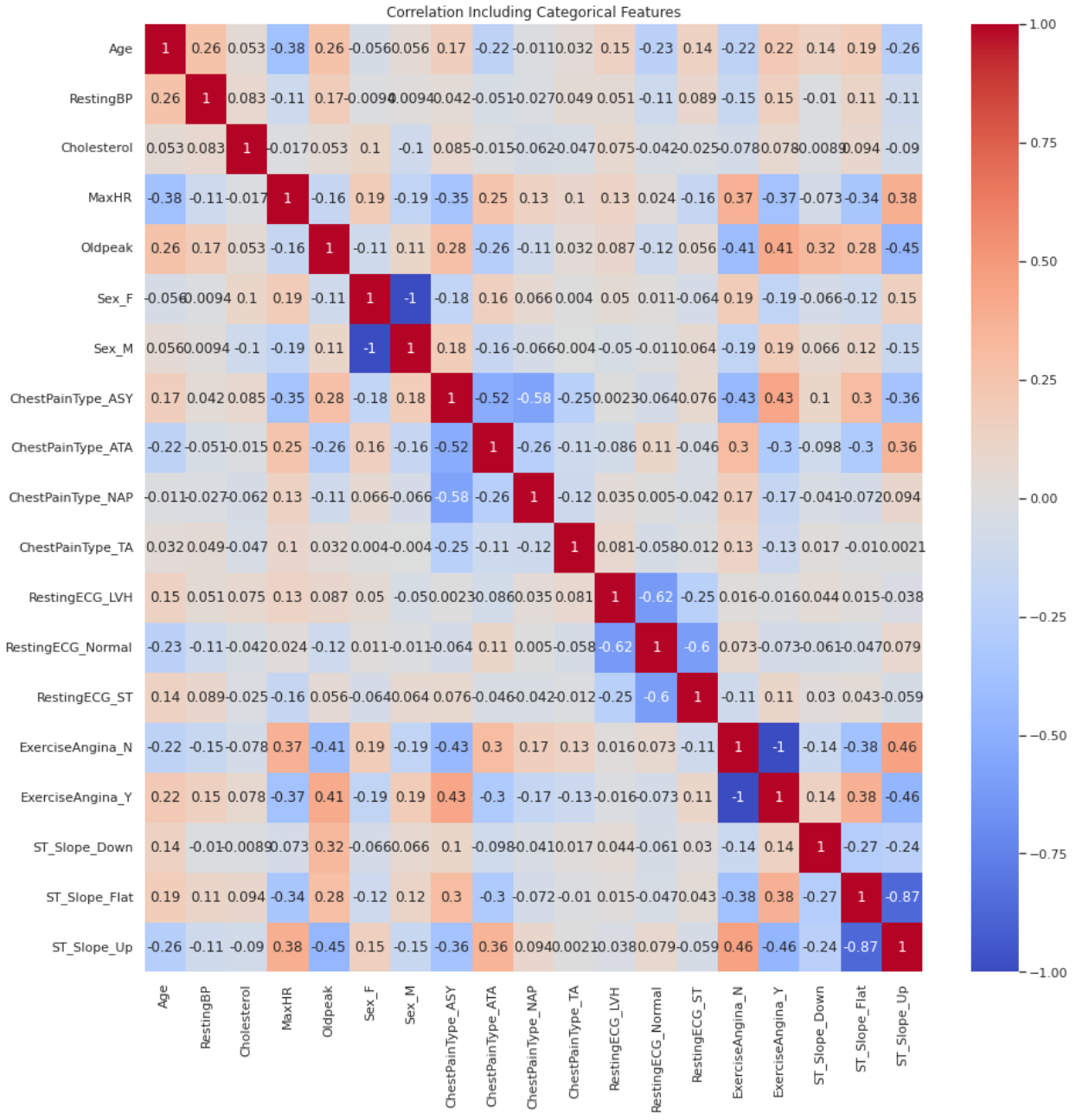
Before:



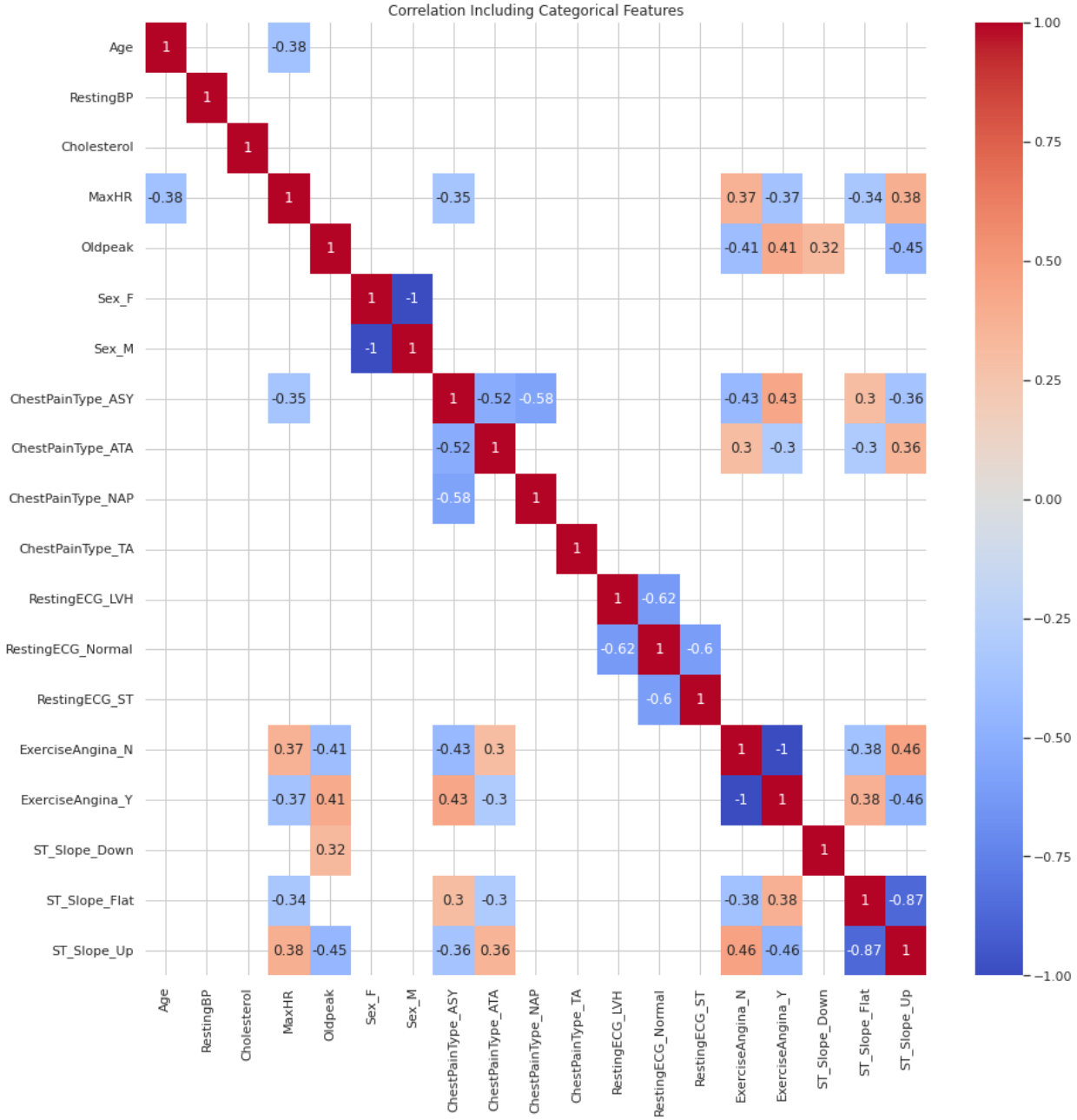
After:



Next we create a correlation matrix of all features, using one-hot econding to create numerical columns for each numerical column.



However, given the number of features, we can apply a mask feature to more easily distinguish correlations that are significant.


Correlation Including Categorical Features
Age
RestingBP
Cholesterol
MaxHR
Oldpeak
Sex_F
Sex_M
ChestPainType_ASY
ChestPainType_ATA
ChestPainType_NAP
ChestPainType_TA
RestingECG_LVH
RestingECG_Normal
RestingECG_ST
ExerciseAngina_N
ExerciseAngina_Y
ST_Slope_Down
ST_Slope_Flat
ST_Slope_Up
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Random Forect Model Development

We start by splitting out our model into training and test sets:

```
# Split dataset into train and test sets
X = dataset[numerical_columns + categorical_columns]
y = dataset[target_column]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=111)

```

We then use a slightly different approach for prepping data for our models – sklearn's Pipelines methods.

We define pipelines for both numerical data and categorical datatypes. Given that we are working with numerical data, we want to scale the datasets to account for the significant ranges (i.e. age or BP).

```
# Define preprocessing pipeline
numerical_pipeline = Pipeline([('scaler', StandardScaler())])
categorical_pipeline = Pipeline([('encoder', OneHotEncoder(handle_unknown='ignore'))])

```

We then combine pipelines into a "preprocessor" object, using a method called ColumnTransformer from sklearn that applies scaling to the numerical pipelines and one-hot encoding to categorical pipelines.

We define the model pipeline with a random_state that we change with each run

```
# Define model pipeline
model_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=111))
])

# Train the model
```

Next, we train the model and make predictions:

```
# Train the model
model_pipeline.fit(X_train, y_train)

# Make predictions
y_pred = model_pipeline.predict(X_test)

# Evaluate the model
```

After training, we get the following results:

```
26
27  # Evaluate the model
28  print("Confusion Matrix:")
29  print(confusion_matrix(y_test, y_pred))
30
31  print("\nClassification Report:")
32  print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[71 11]
 [ 9 93]]

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88        82
           1       0.89      0.91      0.90       102

    accuracy                           0.89       184
   macro avg       0.89      0.89      0.89       184
weighted avg       0.89      0.89      0.89       184
```

The confusion matrix here tells us the following:

True Negatives (71): The model correctly predicted 71 instances of 0 (no heart disease).

False Positives (11): The model incorrectly predicted 1 (heart disease) for 11 patients who do not have heart disease.

False Negatives (9): The model incorrectly predicted 0 (no heart disease) for 9 patients who actually have heart disease.

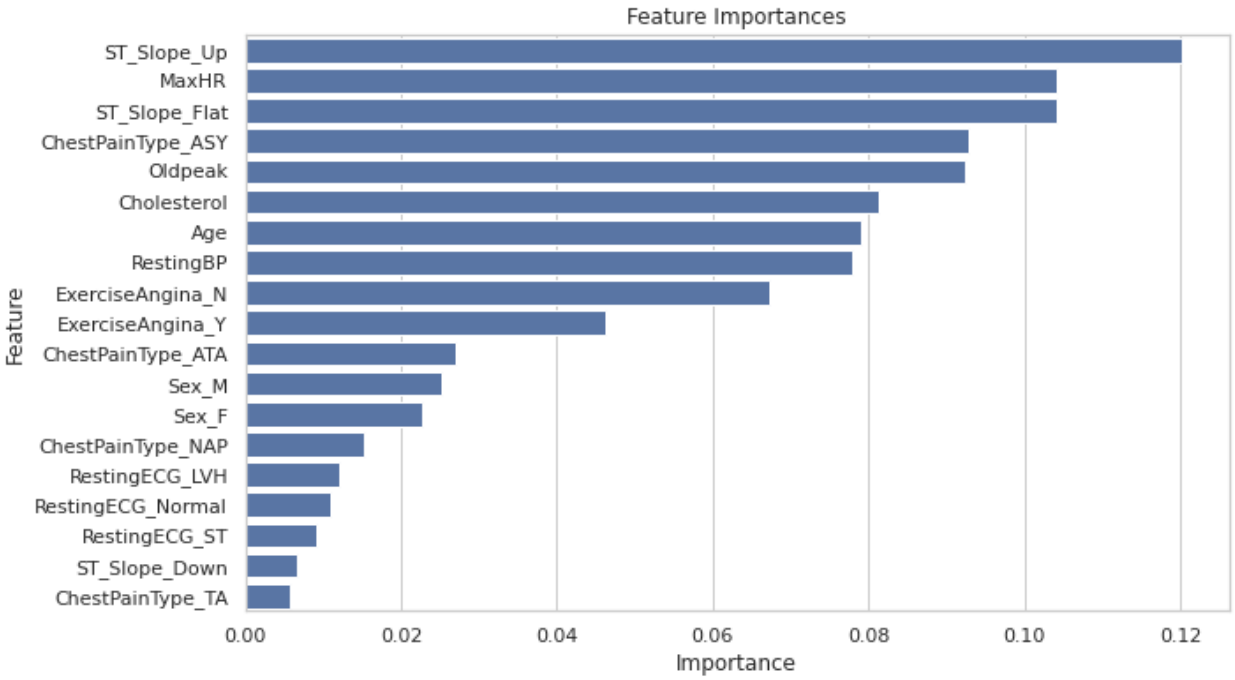
True Positives (93): The model correctly predicted 93 instances of 1 (heart disease).

Insights from the Output

1. **Balanced Precision and Recall:**
    - Both classes (0 and 1) have high precision (89%) and recall (87–91%).
    - This means the model is equally effective in predicting the presence and absence of heart disease.
2. **Low False Negatives:**
    - There are only 9 false negatives, which is critical in healthcare as missing a true positive (patient with heart disease) can have severe consequences.
3. **High F1-Scores:**
    - F1-scores of 88% (class 0) and 90% (class 1) indicate a strong balance between precision and recall, making the model reliable.
4. **High Overall Accuracy:**
    - An accuracy of 89% is strong, indicating the model performs well across the entire dataset.

Feature Importance

Using sklearn's feature importances library, we're able to rank the parameters as visible below



These features line up with the observations made on the correlation matrix – namely that ST_Slope and MaxHR were heavily correlated with HeartDisease, reinforcing the importance of handling categorical instead of only numerical.

For comparison, two other methods were used for prediction – logistic regression and gradient boosting. Their results are presented below:

```
Random Forest Accuracy: 0.8967391304347826
Logistic Regression Accuracy: 0.8695652173913043
Gradient Boosting Accuracy: 0.907608695652174

Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.82      0.85        82
           1       0.86      0.91      0.89       102

    accuracy                           0.87       184
   macro avg       0.87      0.86      0.87       184
weighted avg       0.87      0.87      0.87       184


Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.89      0.90        82
           1       0.91      0.92      0.92       102

    accuracy                           0.91       184
   macro avg       0.91      0.91      0.91       184
weighted avg       0.91      0.91      0.91       184
```

These results show a very similar performance to that of our Random Forest Classifier, suggesting either approach would yield an acceptable result. It would be interesting to see how these results change with a significantly larger dataset.