HEART DISEASE PREDICTION

Prepared by :
Rana Alsattari

Supervised by:
Dr. Huseyin Kusetogullari

2025

REPORT

TABLE OF CONTENTS

Part. I

1. Introduction

Heart disease is one of the most common health problems worldwide, and being able to predict it early can make a big difference in saving lives. In this project, we built a complete data engineering pipeline that takes raw medical data and transforms it into useful insights and predictions.

We followed the entire data lifecycle step by step. We started by collecting the datasets, cleaning them, handling missing values, and preparing them for analysis. After that, we integrated the datasets into one organized file. Then, we applied machine-learning models to predict whether a person has heart disease based on medical and lifestyle information.

By the end of the project, we created a full system that goes from raw data to accurate predictions and visual dashboards. This shows how data engineering and machine learning can support better understanding and early detection of heart disease.

2. Project Description

In this project, we built a full end-to-end data engineering pipeline to predict heart disease. We worked with two datasets, a heart dataset and a medical dataset, and our goal was to prepare them for machine-learning analysis. We started by extracting the raw data, cleaning it, fixing missing values, removing outliers, and creating new features that could help improve predictions.

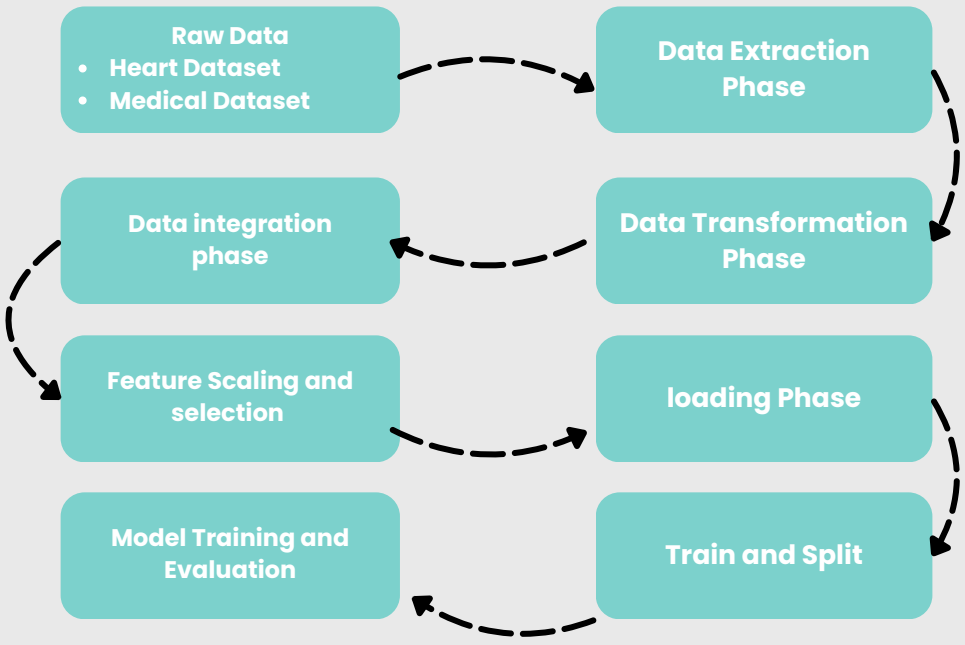
After preparing the data, we integrated both datasets using age and gender, applied scaling, and selected the most important features. We then trained two models, Random Forest and Logistic Regression, to compare their accuracy. Finally, we visualized the results using ROC curves, confusion matrices, and probability plots to understand how well the models predict heart disease. This completed the full pipeline from raw data to final predictions.

Part. I

3. Data Lineage and Data Flow

In this project, we followed a clear data flow that shows how the data moved and changed from the beginning until the final model. The process started with extracting the raw heart and medical datasets from Kaggle. After extraction, the data entered the transformation phase, where we cleaned it, removed duplicates, handled missing values, fixed incorrect formats, and created new engineered features such as risk_score and lifestyle_score.

Next, the cleaned datasets went through integration, where we merged both datasets using age and gender. After that, we applied scaling, feature selection, and split the data into training and testing sets. Finally, the processed data moved into the loading phase, where we stored all outputs and used them to train machine-learning models. The flow ended with model evaluation and visualization, allowing us to understand prediction accuracy and overall performance. This step-by-step lineage ensures that every transformation is traceable and the final results are reliable.



Part. I

4. Metadata and Data Source

In this project, we worked with two datasets taken from Kaggle: the Heart Disease dataset and the Medical Records dataset. Before starting any cleaning or transformation, we checked the metadata of both datasets. This included the number of rows and columns, the names of the features, and the data types (numeric, categorical, etc.). Understanding this helped us know what each column represents and how it should be handled later during analysis.

We looked at the key fields such as age, gender, cholesterol, blood pressure, CK-MB, and troponin levels, which are important for identifying heart disease patterns. Reviewing the metadata also helped us identify which features could be used for merging, which ones needed encoding, and which required feature engineering. Overall, metadata acted like a "map" that guided us through the entire data pipeline and made sure we processed everything correctly.

```
=== HEART DATASET METADATA ===
Shape: (1000, 16)

Column Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   age                      896 non-null    float64
 1   gender                   1000 non-null   object
 2   cholesterol              906 non-null    float64
 3   blood_pressure           907 non-null    float64
 4   heart_rate               915 non-null    float64
 5   smoking                  1000 non-null   object
 6   alcohol_intake           677 non-null    object
 7   exercise_hours           908 non-null    float64
 8   family_history           1000 non-null   object
 9   diabetes                 1000 non-null   object
 10  obesity                  1000 non-null   object
 11  stress_level             877 non-null    float64
 12  blood_sugar              910 non-null    float64
 13  exercise_induced_angina  1000 non-null   object
 14  chest_pain_type          1000 non-null   object
 15  heart_disease            889 non-null    float64
dtypes: float64(8), object(8)
memory usage: 125.1+ KB
None
```

```
=== MEDICAL DATASET METADATA ===
Shape: (1319, 9)

Column Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1319 entries, 0 to 1318
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   age                      1179 non-null   float64
 1   gender                   1203 non-null   float64
 2   heart_rate               1180 non-null   float64
 3   systolic_blood_pressure  1175 non-null   float64
 4   diastolic_blood_pressure 1178 non-null   float64
 5   blood_sugar              1184 non-null   float64
 6   ck_mb                    1181 non-null   float64
 7   troponin                 1180 non-null   float64
 8   result                   1319 non-null   object
dtypes: float64(8), object(1)
memory usage: 92.9+ KB
None

Preview:
    age  gender  heart_rate  systolic_blood_pressure  \
0   NaN     1.0        66.0                      NaN
1  21.0     1.0        94.0                     98.0
2  55.0     1.0        64.0                    160.0
```

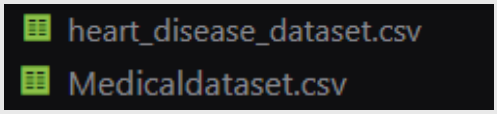
Part. II

EXTRACTION

2.1 Data Sources

In this project, we have used two CSV datasets that contain some medical information about patients. Those datasets have been provided by Kaggle and were stored on our computer. The basic goal of the extraction process is to access those datasets, load them into Python, and inspect their basic structure before performing any transformations on them.

Those Datasets are:
Heart Disease Dataset: Contains 1,000 patient records including age, cholesterol, blood pressure, lifestyle habits, stress levels, and a heart disease label.
Medical Dataset: Contains 1,319 patient records with clinical measurements such as systolic/diastolic blood pressure, blood sugar levels, CK-MB, and troponin.



2.2 Data Loading

Now we need to extract raw data, so we loaded both csv files using `pandas.read_csv()` function, also to ensure that we follow and check the dataset in each step, we created an output directory.

After loading the datasets, we wanted to check if the datasets has been read correctly, so we printed the shapes of the datasets.

```
Heart Dataset Shape: (1000, 16)
Medical Dataset Shape: (1319, 9)
```

2.3 Column Normalization

Before analyzing the data, we standardized all column names so they follow a consistent format. This step included:

- converting all names to lowercase
- removing spaces
- replacing symbols with underscores
- removing trailing underscores

2.4 Initial Profiling

Before starting any cleaning, we first explored the raw datasets to understand their overall quality and structure. This step helped us to see what issues the data had, such as missing values, duplicates, or unusual numbers. By doing this early check, we got a clear idea of what needed to be fixed later during the transformation process.

2.4.1 Duplicate Detection

we checked both datasets for duplicate rows to make sure there were no repeated records. The results showed zero duplicates in both datasets, so nothing needed to be removed.

2.4.2 Missing Values

I then checked for missing values. The heart dataset had 340 missing values in the `alcohol_intake` column, while the medical dataset had no missing values. This helped me understand what needed cleaning later.

2.4.3 Summary Statistics

Finally, we viewed the summary statistics to get a quick idea of the data ranges, averages, and distributions. This confirmed that the datasets contained realistic medical values but also showed variations that might require cleaning in the next steps.

Heart Dataset Summary Statistics:

	age	cholesterol	blood_pressure	heart_rate	exercise_hours	\
count	1000.000000	1000.000000	1000.0000	1000.000000	1000.000000	
mean	52.293000	249.939000	135.2810	79.204000	4.529000	
std	15.727126	57.914673	26.3883	11.486092	2.934241	
min	25.000000	150.000000	90.0000	60.000000	0.000000	
25%	39.000000	200.000000	112.7500	70.000000	2.000000	
50%	52.000000	248.000000	136.0000	79.000000	4.500000	
...						
max		154.000000	541.000000	300.000000	10.300000	

```python
def normalize_columns(df):
    df = df.copy()
    df.columns = (
        df.columns
        .str.strip()
        .str.lower()
        .str.replace(r'[^0-9a-zA-Z]+', '_', regex=True)
        .str.strip('_')
    )
    return df

heart = normalize_columns(heart)
medical = normalize_columns(medical)
```

Transformation

3.1 Data Injection

In this step, we purposely made the datasets “messy” so we could test our data engineering skills and see how well our cleaning methods would handle real-world problems. Real medical data is almost never perfect, so this simulation helped us practice dealing with common issues.

I applied two types of data injection:

1. Adding Missing Values:
We randomly turned about 10% of all numeric values into NaN. This represents situations like incomplete patient records, missing measurements, or mistakes during data entry.

2. Adding Invalid Values:
For categorical columns, we replaced around 5% of the values with the word "INVALID". This simulates human typing errors, corrupted records, or system bugs.

To understand the damage more clearly, I generated a profiling summary.

Heart dataset after corruption (sample):

	age	gender	cholesterol	blood_pressure	heart_rate	smoking
0	75.0	Female	228.0	119.0	66.0	Current
1	48.0	Male	204.0	165.0	62.0	Current
2	53.0	Male	234.0	91.0	67.0	Never
3	69.0	Female	192.0	90.0	72.0	Current
4	62.0	Female	172.0	NaN	93.0	Never
5	77.0	Male	NaN	110.0	NaN	Never
6	64.0	Female	211.0	NaN	86.0	Former
7	60.0	Female	208.0	148.0	83.0	Never
8	37.0	Female	317.0	137.0	66.0	Current
9	63.0	Male	204.0	141.0	NaN	Former

	alcohol_intake	exercise_hours	family_history	diabetes	obesity
0	Heavy	1.0	No	No	Yes
1	NaN	5.0	No	No	No
2	Heavy	3.0	Yes	No	Yes
3	NaN	4.0	No	Yes	No
4	NaN	6.0	No	Yes	INVALID
5	NaN	0.0	No	Yes	Yes
6	Heavy	NaN	Yes	Yes	Yes
7	Moderate	4.0	No	Yes	Yes
8	Heavy	3.0	No	Yes	Yes
9	Heavy	NaN	No	Yes	No

Medical dataset after corruption (sample):

	age	gender	heart_rate	systolic_blood_pressure
0	64.0	1.0	NaN	160.0
1	21.0	1.0	94.0	98.0
2	55.0	1.0	NaN	160.0
3	NaN	NaN	70.0	120.0
4	55.0	1.0	64.0	112.0
5	58.0	NaN	NaN	112.0
6	32.0	0.0	40.0	179.0
7	63.0	1.0	NaN	214.0
8	44.0	0.0	60.0	154.0
9	67.0	1.0	NaN	NaN

	diastolic_blood_pressure	blood_sugar	ck_mb	troponin	result
0	83.0	160.0	1.80	0.012	negative
1	46.0	296.0	6.75	1.060	positive
2	77.0	270.0	1.99	0.003	negative
3	55.0	NaN	13.87	0.122	INVALID
4	65.0	NaN	1.08	0.003	negative
5	58.0	87.0	1.83	0.004	negative
6	68.0	102.0	0.71	0.003	negative
7	82.0	87.0	300.00	2.370	positive
8	81.0	135.0	2.35	0.004	negative
9	NaN	100.0	2.84	0.011	negative

3.2 Data Cleaning

After the injection part, we moved on to the cleaning phase. The goal here was to fix all the missing values, invalid values, outliers, and any other issues that could affect the analysis. This step is important in data engineering because clean data produces more accurate models and more reliable insights.

3.2.1. Replacing “INVALID” Values

Some categorical columns contained the word "INVALID".
we replaced all “INVALID” entries with NaN so I could treat them like missing values.

```
df = df.replace(r"^\s*INVALID\s*$", np.nan, regex=True)
```

3.2.2. Identifying Numeric vs Categorical Columns

I separated the dataset into:

- numeric columns → integers, floats
- categorical columns → object (text)

3.2.3. Handling Outliers

For all numeric columns, we used the IQR method to clip extreme values.
This means values below the lower limit or above the upper limit were brought back inside a reasonable range.

```
for col in num_cols:
    Q1, Q3 = df[col].quantile(0.25), df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower, upper = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR
    df[col] = df[col].clip(lower, upper)
```

3.2.4. Imputing Missing Values

For fixing the missing values:

- Numeric columns → replaced using median.
- Categorical columns → replaced using most frequent value.

This made the datasets complete again with no missing entries.

Transformation

3.2.5. Encoding Categorical Columns

Since machine learning models cannot understand text, we converted all categorical values into numbers using Label Encoding.
For example:

- "Male" → 1
- "Female" → 0

This prepares the data for modeling.

```
le = LabelEncoder()
for col in cat_cols:
    df[col] = le.fit_transform(df[col])
```

gender
1
1
1
1
1
0
0
1

During cleaning, we used label encoding to convert gender from text to numbers. Here, 0 = Female and 1 = Male, making the column easier to use in analysis and machine-learning models.

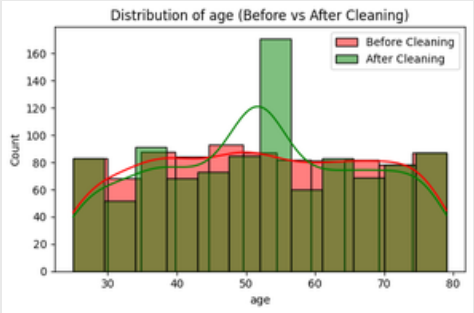
3.2.6. Duplicate Removal

We checked again for duplicates after cleaning. Both datasets still had no duplicate rows, which means we didn't need to remove anything.

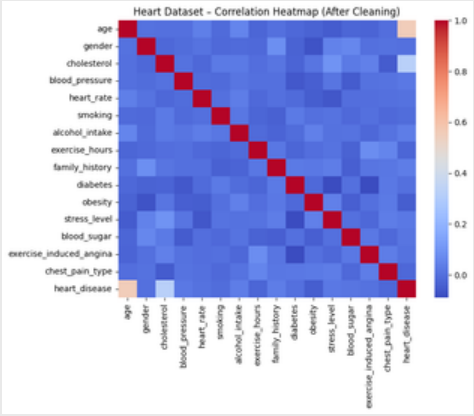
```
dup_count = df.duplicated().sum()
if dup_count > 0:
    df = df.drop_duplicates()
    print(f"Removed {dup_count} duplicate rows.")
else:
    print("No duplicates found.")

print(f"{name} cleaned successfully. Final shape: {df.shape}")
return df
```

```
 Cleaning Heart Dataset...
No duplicates found.
Heart Dataset cleaned successfully. Final shape: (1000, 16)

 Cleaning Medical Dataset...
No duplicates found.
Medical Dataset cleaned successfully. Final shape: (1319, 9)
```

After finishing the cleaning steps, both datasets were finally in good shape and ready to use. All missing and invalid values were fixed, the text categories were encoded into numbers, and the outliers were handled. We also generated a new profiling summary to compare it with the "before cleaning" version, just to make sure everything was cleaned properly before moving on to feature engineering.

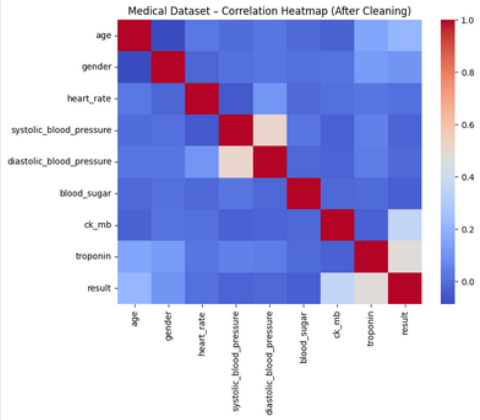


This graph shows how the age values looked before and after cleaning. The red part is the original data, and the green part is the cleaned version. After cleaning, the age distribution looks smoother and more balanced, especially in the middle ages. This means the cleaning process removed unusual or missing values and made the data more accurate to use in the next steps.



This heatmap shows how each feature in the heart dataset is related to the others after cleaning. Most of the squares are blue, which means the correlations between variables are low. This is good because it shows there is no strong multicollinearity, and each feature provides its own information. The diagonal line is dark red because each feature perfectly correlates with itself. Overall, this visualization helped us understand which features have stronger relationships and which ones are independent before moving into feature selection and modeling.

Transformation



This heatmap shows how the medical dataset features relate to each other. Most correlations are weak, which means the variables are fairly independent. A few pairs, like systolic and diastolic blood pressure, show moderate correlation. Overall, the cleaned medical data looks consistent and ready for modeling.

3.3. Feature Engineering

In this phase, we created new features to make the dataset more informative for the prediction model. After loading the cleaned heart and medical datasets, we focused on the columns that could add more meaning when combined. From the heart dataset, we created a lifestyle_score using smoking and alcohol intake, a risk_score based on cholesterol, blood pressure, and age, and an age_group column that groups patients into meaningful age categories. We also saved small preview samples and full engineered versions of both datasets. These new features help capture important patterns that will improve the model's performance in the next phases.

	risk_score	lifestyle_score	age_group
0	4.63	0.0	Elder
1	7.69	0.0	Senior
2	6.13	0.0	Senior
3	4.09	0.0	Elder
4	4.95	1.2	Elder
5	5.44	1.2	Elder
6	4.94	0.6	Elder
7	5.93	1.6	Senior
8	12.27	0.0	Middle-Aged
9	5.48	0.6	Elder

3.4. Dataset Integration

In this phase, we combined the heart dataset and the medical dataset into one integrated dataset. Since both datasets share the demographic features age and gender, we used these two columns as the merge keys. We performed an inner join, which means only records that match in both datasets were kept in the final result. After merging, the integrated dataset reached 11,002 rows and 26 columns, showing that many records from both files were successfully aligned. We also created a summary file and saved a sample of the integrated table to verify that the merging worked correctly. This integration step is important because it brings together all the health indicators in one place, allowing the next phases like feature selection and model training to use a complete and richer dataset.

```
merge_keys = ['age', 'gender']
merged = pd.merge(heart, medical, on=merge_keys, how='inner')

print(f" Merged successfully on keys: {merge_keys}")
print(f"Final merged shape: {merged.shape}")
```

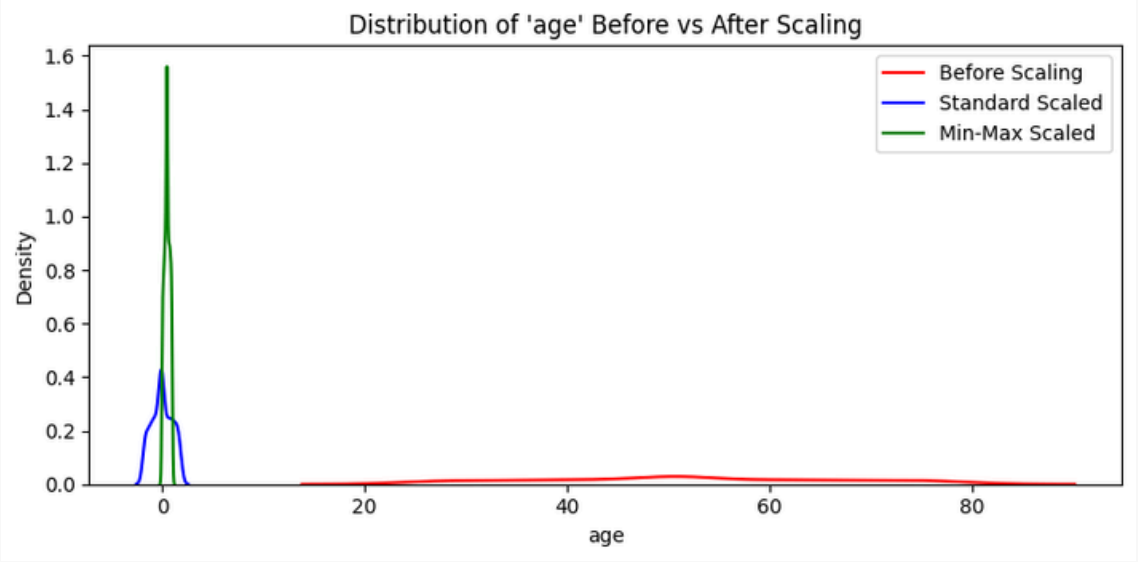
```
Sample of merged dataset:
    age  gender  cholesterol  blood_pressure  heart_rate_x  smoking  \
0  75.0       0        228.0           119.0          66.0        0
1  75.0       0        228.0           119.0          66.0        0
2  75.0       0        228.0           119.0          66.0        0
3  75.0       0        228.0           119.0          66.0        0
4  75.0       0        228.0           119.0          66.0        0
5  75.0       0        228.0           119.0          66.0        0
6  75.0       0        228.0           119.0          66.0        0
7  75.0       0        228.0           119.0          66.0        0
8  75.0       0        228.0           119.0          66.0        0
9  75.0       0        228.0           119.0          66.0        0

   alcohol_intake  exercise_hours  family_history  diabetes  ...  risk_score  \
0               0             1.0               0         0  ...        4.63
1               0             1.0               0         0  ...        4.63
2               0             1.0               0         0  ...        4.63
3               0             1.0               0         0  ...        4.63
4               0             1.0               0         0  ...        4.63
5               0             1.0               0         0  ...        4.63
6               0             1.0               0         0  ...        4.63
...
8                  68.0          99.0   2.85   0.0750       1
9                  74.0          92.0   3.61   0.0090       0
```

This table shows the first 10 records of the integrated dataset after merging the heart and medical datasets using age and gender as the common keys. Each row now contains features from both datasets, such as cholesterol, blood pressure, stress level, CK-MB, troponin, and the medical result. Seeing both sets of attributes in one row confirms that the datasets were successfully combined, and that the integration step produced a richer dataset for further analysis.

Transformation

3.5 Feature Scaling

In this step, we scaled the numeric columns in both the heart and medical datasets so everything is on a similar range before training the model. We used two methods: Standard Scaling, which makes the values centered around zero, and Min-Max Scaling, which puts all numbers between 0 and 1. This is important because some features have very big values (like blood pressure) while others are small, and scaling helps the model treat them fairly. We also plotted the age column to see how the scaling changed the distribution, and the graph clearly shows the difference between the original values and the two scaled versions. After finishing, we saved all the scaled datasets along with a small summary table.

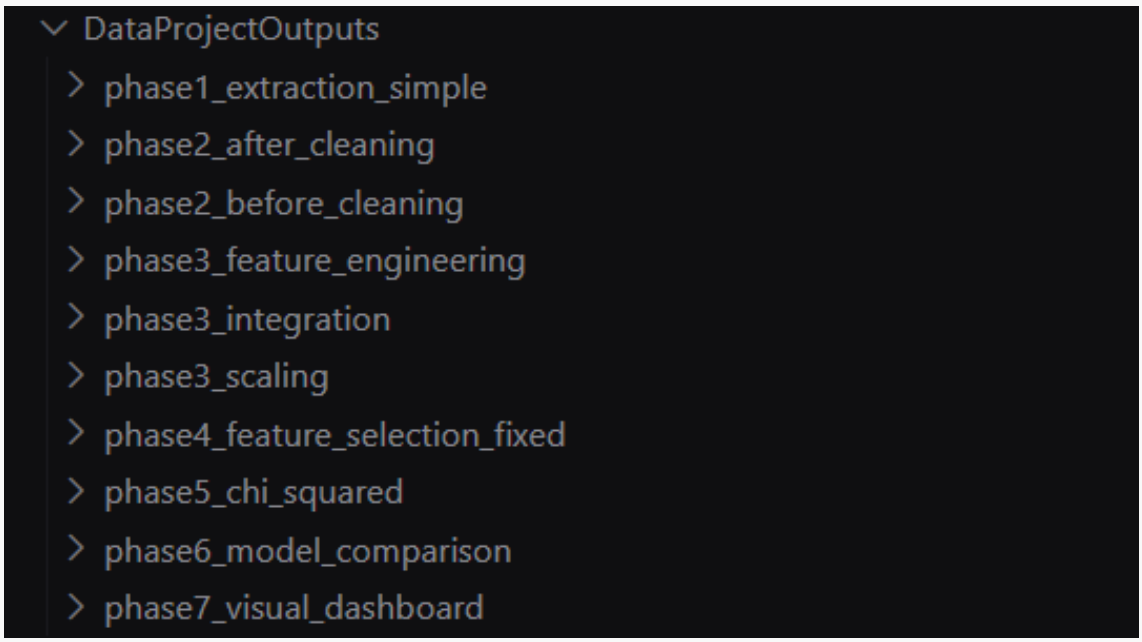


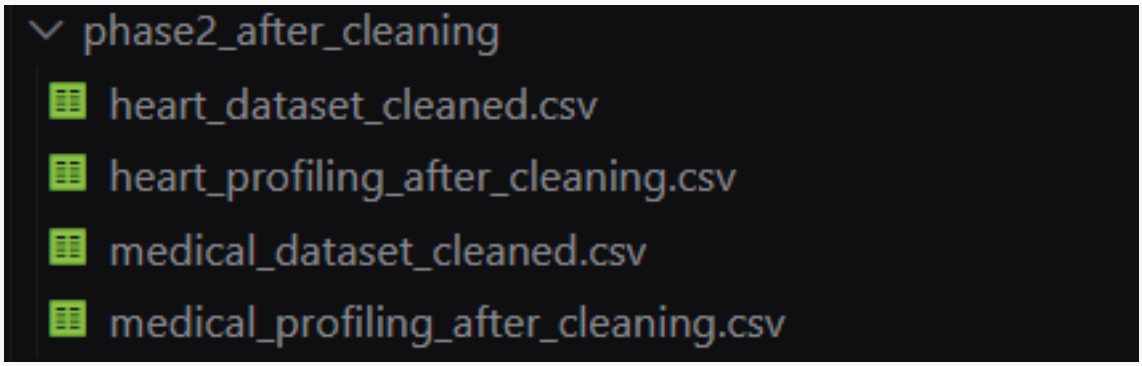
This graph compares the age values before and after applying scaling. The red curve shows the original distribution, while the blue and green curves show how the values look after Standard Scaling and Min-Max Scaling. After scaling, the numbers are squeezed into a smaller range, which makes the data easier for machine learning models to work with. The shape stays similar, but the values are now standardized and more balanced.

Loading Phase

4. Loading Process

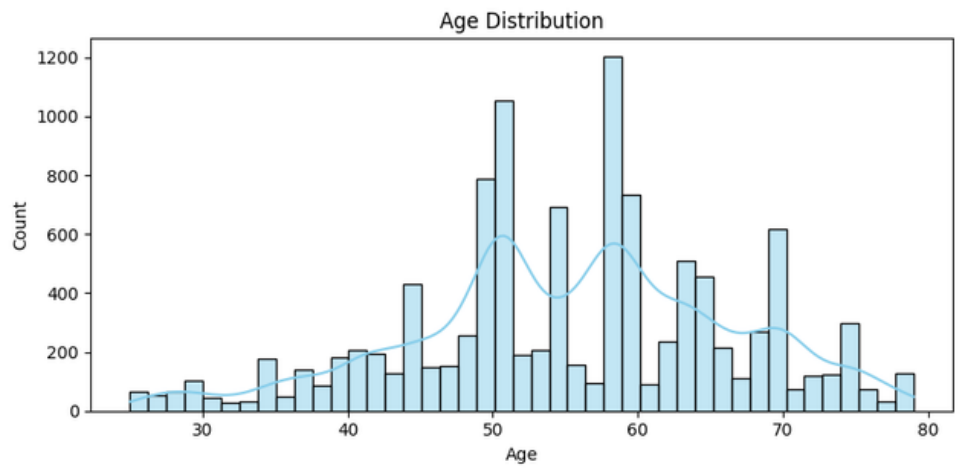
In this phase, we saved all the processed datasets that were created during the transformation steps. The goal of the Loading Phase is to make the cleaned, engineered, scaled, and integrated datasets available for the next stage, which is modeling. By storing the files properly, we ensured that each version of the dataset can be reused without repeating the entire transformation pipeline. We exported multiple versions such as the cleaned datasets, the feature-engineered datasets, the scaled datasets, and the final integrated dataset. These files now represent the “ready-to-use” data that can be directly used for machine learning models.





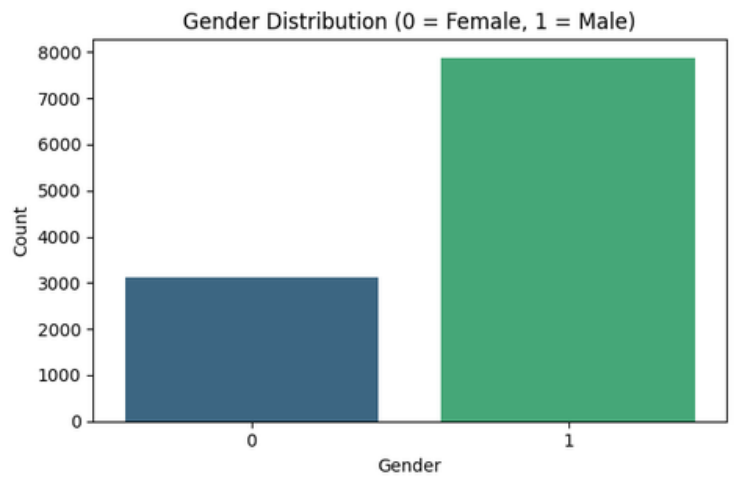
Data Visualization

Age Distribution



This chart shows how the ages are distributed in the dataset. Most people are between 50 and 60 years old, with another group around 70. The chart has bars that show how many people fall into each age range, and the smooth line helps us see the overall trend more clearly. It looks like the majority of people in the dataset are middle-aged to senior.

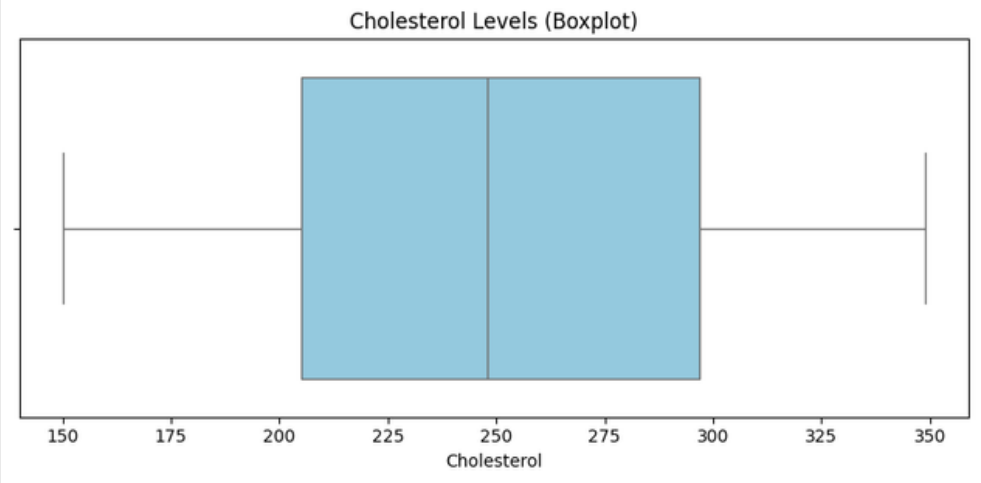
Gender Distribution



This chart shows the Gender Distribution of the dataset. The x-axis represents gender, where 0 stands for Female and 1 stands for Male. The y-axis shows the count of occurrences for each gender. From the chart, we can see that there are significantly more male entries (represented by 1) than female entries (represented by 0).

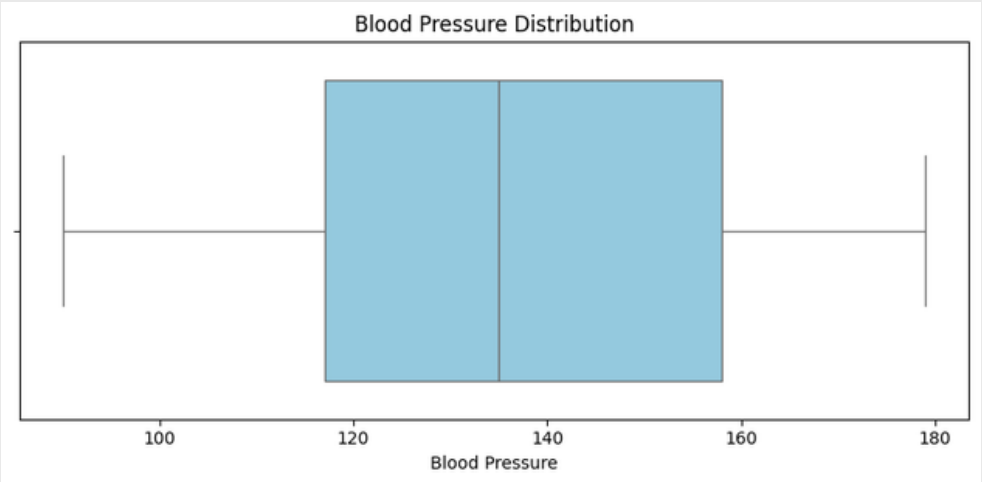
Data Visualization

Cholesterol Levels (Boxplot)



This boxplot shows the distribution of cholesterol levels. The box in the middle represents the middle 50% of the data, with the line inside the box showing the median value. The lines extending from the box (whiskers) show the range of the data, and any dots outside the whiskers are outliers. From this plot, we can see that most cholesterol levels are between 200 and 300, and there are a few values that are considered outliers.

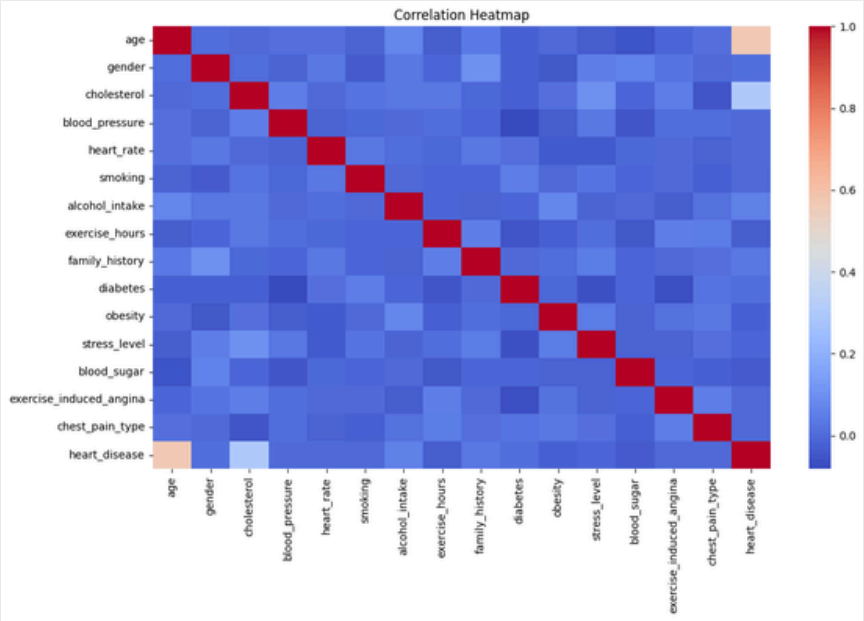
Blood Pressure Distribution



This boxplot illustrates the distribution of blood pressure levels. The box in the center shows the interquartile range (IQR), with the line inside representing the median blood pressure. The whiskers extend to the minimum and maximum values within 1.5 times the IQR, while any values outside of this range are considered outliers. The plot indicates that the majority of the data falls between 110 and 150, with some values extending up to 180, suggesting a small spread of higher blood pressure readings.

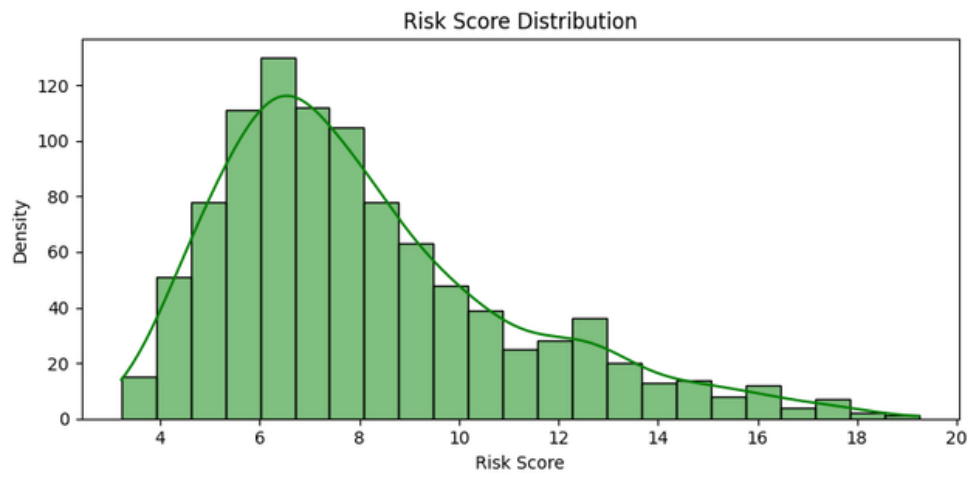
Data Visualization

Correlation Heatmap



This heatmap displays the correlation between various features in the dataset. Dark red squares indicate high correlations between the corresponding features, meaning these variables are closely related. For example, features like age, cholesterol, and blood pressure show strong relationships. The heart disease column, however, has weaker correlations with most of the other features. The heatmap provides a quick overview of how different variables interact with each other, which is helpful as we understand the data and select relevant features for model training.

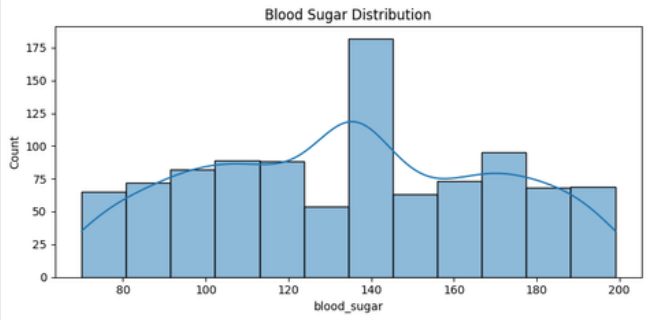
Risk Score Distribution



This plot shows the distribution of the "Risk Score" feature. The histogram displays the frequency of risk scores, with most values between 4 and 10 and a peak around 6, indicating moderate risk. The tail on the higher side suggests fewer individuals have higher risk scores. This helps understand the risk score distribution and aids in model development.

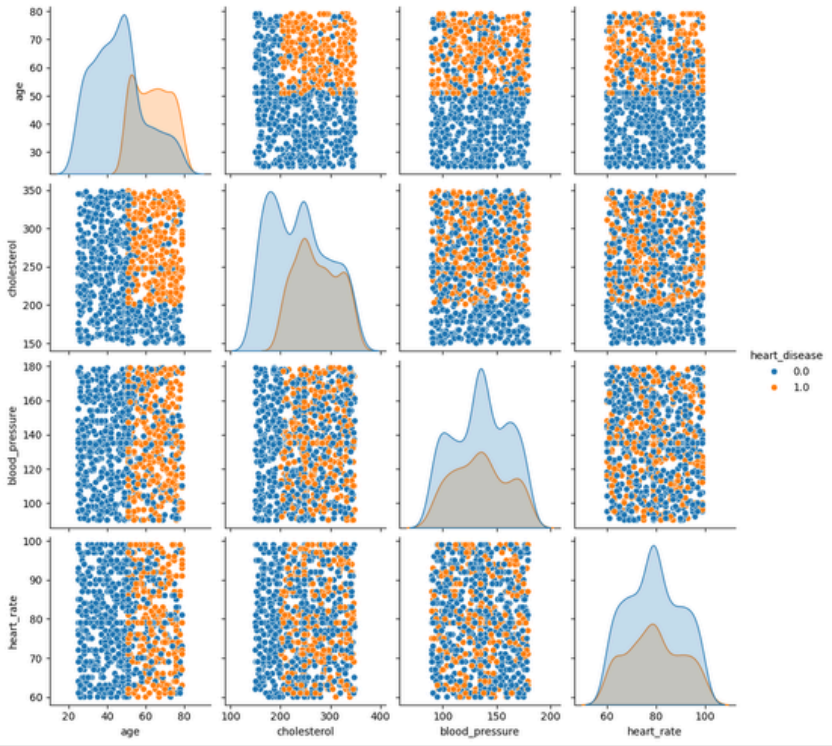
Data Visualization

Blood Sugar Distribution



This plot shows the distribution of the "Blood Sugar" feature in the dataset. The histogram displays the frequency of blood sugar values, with a peak around 140, suggesting a concentration of values near this level. The density curve shows the overall shape of the distribution, which is relatively uniform but with some variations. The distribution helps us understand the spread of blood sugar levels in the dataset, which can be useful for model training and feature analysis.

Pairplot of Key Features Colored by Heart Disease Status



This pairplot shows the relationships between features like age, cholesterol, blood pressure, and heart rate, with points colored by heart disease status (orange for those with the disease and blue for those without). The diagonal histograms show the distribution of each feature. It highlights how these variables interact, revealing trends such as higher age and cholesterol levels in individuals with heart disease.

Train and Test

6.1 Dataset Loading

We started by loading the cleaned datasets that we had worked on earlier. We removed any unnecessary columns, like "result" and "age_group," that could interfere with the model.

```
data_path = Path(r"C:\Users\ranaz\Desktop\Data Engineering proj\DataProj
df = pd.read_csv(data_path)
print(f" Dataset loaded: {df.shape}")
```

6.2 Feature Selection

In this step, the goal was to identify the most relevant features that would help predict heart disease. This is important because using too many unnecessary features can lead to overfitting, while too few features might not capture the full complexity of the data. We used multiple techniques to perform feature selection, starting with correlation analysis, then using a Random Forest classifier to measure feature importance.

1. Dropping Irrelevant Columns

The first step in feature selection was to remove irrelevant or redundant columns. For example, we dropped the "result" and "age_group" columns as they were either not needed for the prediction or were linked to other features.

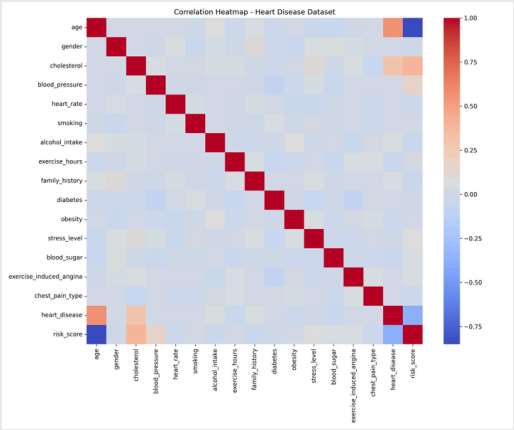
```
target_col = "heart_disease"
drop_cols = ["result", "age_group"]
df = df.drop(columns=[c for c in drop_cols if c in df.columns])
```

This will ensure that the data has only necessary features

6.3 Correlation Analysis

Next, we conducted a correlation analysis to explore the relationships between features. Features that are highly correlated may provide similar information. By identifying such relationships, we could avoid using features that essentially duplicate information. For example, age and cholesterol levels are likely to be correlated with heart disease risk.

```
numeric_df = df.select_dtypes(include=["number"])
corr = numeric_df.corr()

out_dir = Path(r"C:\Users\ranaz\Desktop\Data Engineering proj\Data
out_dir.mkdir(parents=True, exist_ok=True)

corr[target_col].sort_values(ascending=False).to_csv(out_dir / "co
```



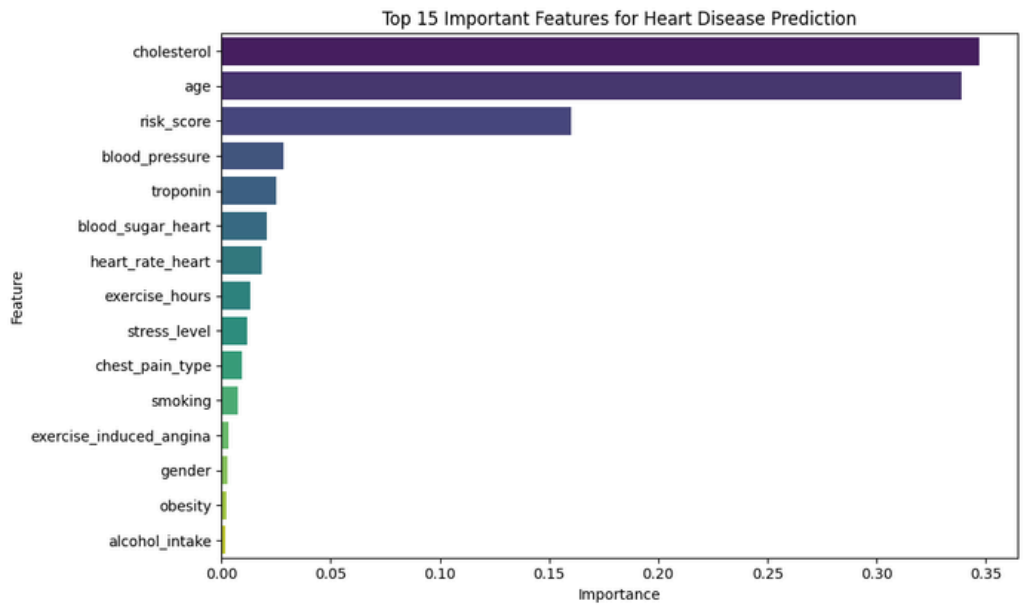
This Correlation Heatmap shows how different features in the Heart Disease Dataset are related to each other. The red areas show features that are strongly related, while the blue areas show weaker relationships. For example, heart disease is strongly related to cholesterol, blood pressure, and risk score, which means these factors are important when predicting heart disease. This helps us understand how features interact and decide which ones to keep for analysis.

6.4 Feature Importance with Random Forest

We used a Random Forest Classifier to evaluate the importance of each feature in predicting heart disease. This analysis helps us identify which features have the greatest impact on the model's predictions.

```
rf = RandomForestClassifier(
    n_estimators=150,
    max_depth=6,
    min_samples_split=5,
    random_state=42
)
rf.fit(X_train, y_train)

importances = pd.DataFrame({
    "Feature": X.columns,
    "Importance": rf.feature_importances_
}).sort_values(by="Importance", ascending=False)

importances.to_csv(out_dir / "feature_importance.csv", index=False)
```

Train and Test



This plot shows the Top 15 Features that are most important for predicting heart disease. The Risk Score, Cholesterol, and Age have the highest importance. Other important features include Blood Pressure and Troponin. This helps us understand which factors are most useful for making predictions about heart disease.

6.5 Train-Test Split

To train the model, we split the data into two parts: one for training the model and another for testing it. This helps in checking if the model is working well with new, unseen data.

```
X = df.drop(columns=[target_col])
y = df[target_col]

for col in X.select_dtypes(include=["object"]).columns:
    X[col] = LabelEncoder().fit_transform(X[col])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=42
)

print("\n Train-Test split complete.")
print(f"Train shape: {X_train.shape}, Test shape: {X_test.shape}")
```

```
 Train-Test split complete.
Train shape: (7787, 22), Test shape: (2596, 22)
```

6.6 Model Evaluation (Random Forest)

Once we trained the model, we checked its performance using accuracy and other metrics like precision, recall, and F1-score. These metrics help us know how well the model is predicting heart disease.

```
Model Evaluation Report:
              precision    recall  f1-score   support

         0.0       1.00      0.87      0.93      1384
         1.0       0.87      1.00      0.93      1212

    accuracy                           0.93      2596
   macro avg       0.93      0.93      0.93      2596
weighted avg       0.94      0.93      0.93      2596
```

The Model Evaluation Report provides key metrics for assessing the heart disease prediction model's performance:

- Precision measures how accurate the positive predictions are. For predicting no heart disease (0.0), the precision is 1.00, but for predicting heart disease (1.0), it drops to 0.87.
- Recall shows the model's ability to correctly identify positive cases. It is perfect for predicting heart disease (1.0 recall), but lower for predicting no heart disease (0.87 recall).
- F1-Score, a balance of precision and recall, is 0.93 for both classes, indicating a good balance between detecting both positive and negative cases.
- Accuracy of 93% shows that the model is highly accurate in overall predictions, correctly identifying 93% of the cases.

The model's evaluation suggests strong performance in predicting heart disease, with a good balance between precision and recall.

Train and Test

6.7 Model Comparison, Visualization & Evaluation

In this step, we trained two different models, Random Forest and Logistic Regression, to predict whether someone has heart disease based on their health data. These models use features like cholesterol, blood pressure, and age.

1. Training the Models

- **Random Forest:** This model uses many decision trees to make a prediction and then combines them for a better result.
- **Logistic Regression:** This is a simpler model that tries to draw a line (or decision boundary) to separate the data into two groups (heart disease vs no heart disease).

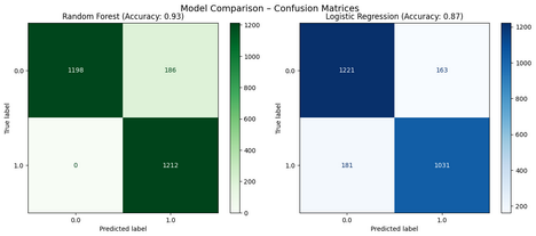
2. Evaluating the Models
To see how well the models work, we looked at:

- **Accuracy:** How many predictions were correct.
- **Precision:** How many predicted positive cases actually had heart disease.
- **Recall:** How many actual heart disease cases the model correctly identified.
- **F1-Score:** A balance between precision and recall.
- **Random Forest** was better, with an accuracy of **93%**. It had fewer mistakes and was good at predicting heart disease without many false negatives.
- **Logistic Regression** had an accuracy of **87%**. It worked well but had more false positives, meaning it sometimes thought people didn't have heart disease when they actually did.

3. Visualizing the Results

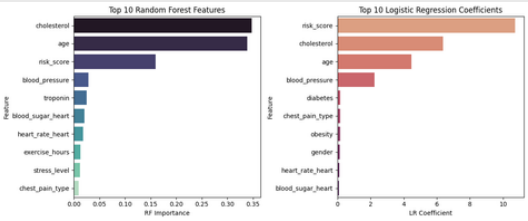
we created a few charts to compare the models:

- **Confusion Matrices:** These show how well the models predicted heart disease. Random Forest performed better with fewer wrong predictions.
- **Accuracy Comparison Chart:** A bar chart showing how Random Forest did better than Logistic Regression.
- **Top Features:** I also looked at which features were most important for making predictions. Both models agreed that cholesterol and age were the most important factors.



The confusion matrix compares the predictions of Random Forest (93% accuracy) and Logistic Regression (87% accuracy). Random Forest performed better, with no false negatives, while Logistic Regression had more false positives.

This bar chart compares the accuracy of Random Forest (93%) and Logistic Regression (87%) models. It visually highlights the higher performance of Random Forest in predicting heart disease.



This side-by-side bar chart compares the Top 10 Important Features for Random Forest and Logistic Regression.

- Random Forest: The most important features for prediction are cholesterol, age, and risk_score.
- Logistic Regression: The most influential features are risk_score, cholesterol, and age, with diabetes and chest_pain_type also having significant contributions.

This helps identify which features are most relevant to each model for heart disease prediction.

Predictive Visualization

7.1 Predictive Visualization

In this phase, we evaluate the model's performance through various visualizations and metrics. These include the ROC curve, prediction probability distribution, confusion matrix, and a comparison of predicted outcomes. Each visualization helps to assess the model's accuracy and its ability to predict heart disease.

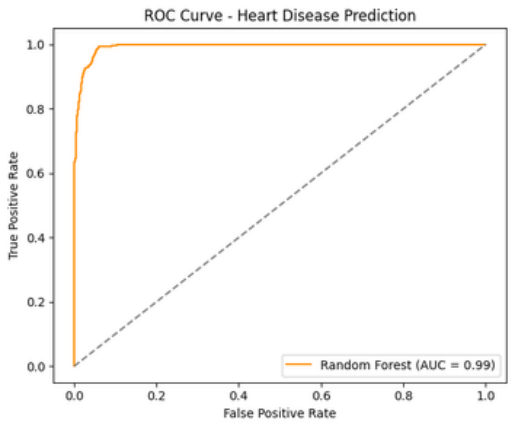
```
Final Model Accuracy: 0.928
ROC-AUC Score: 0.993

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.86      0.93      1384
         1.0       0.87      1.00      0.93      1212

    accuracy                           0.93      2596
   macro avg       0.93      0.93      0.93      2596
weighted avg       0.94      0.93      0.93      2596
```

The Random Forest model performed very well with an accuracy of 92.8%. The ROC-AUC score (0.993) shows the model can almost perfectly separate healthy cases from heart-disease cases.
For the classification report:

- It is very accurate for healthy patients (0) with perfect precision.
- It detects almost all heart disease cases (1) with a recall of 1.00.
- Overall, all scores (precision, recall, F1) are high, meaning the model is reliable and balanced.

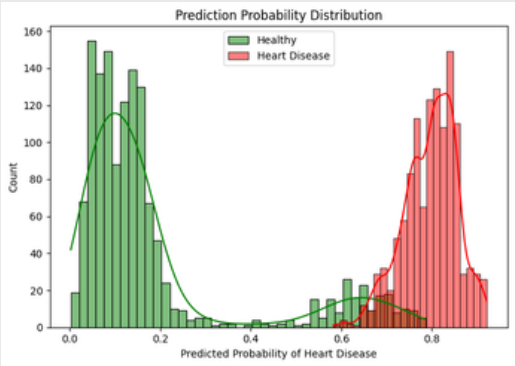
In short, the model makes strong predictions and is especially good at identifying people with heart disease.

1. ROC Curve - Heart Disease Prediction



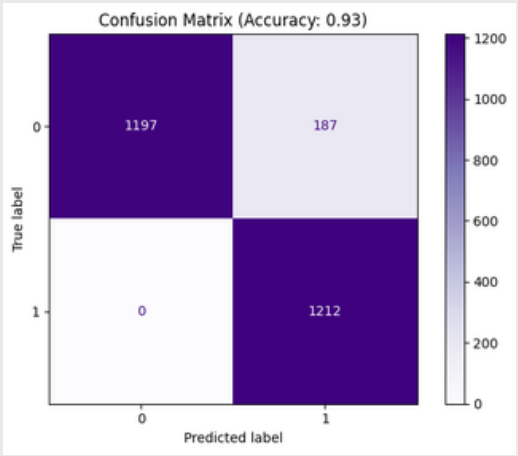
The ROC curve shows how well the model separates healthy and heart-disease cases. The curve is very close to the top-left corner, which means the model is performing extremely well. The AUC score is 0.99, indicating almost perfect prediction ability. The higher the curve, the better the model is at correctly identifying heart disease.

2. Prediction Probability Distribution



This chart shows how the model predicts the probability of heart disease for both groups. The green bars represent healthy people, and most of their predicted probabilities are very low (close to 0). The red bars represent people with heart disease, and their probabilities are mostly high (around 0.7–0.9).

This means the model clearly separates the two classes and is confident in most of its predictions.



The model correctly classified most cases, with high accuracy (93%). It predicted heart-disease patients very well (no missed cases) but made some false alarms for healthy people. Overall, the results show strong and reliable performance.

Finally..

8. Conclusion

In this project, we followed the full data engineering pipeline to build a clean and usable dataset for predicting heart disease. We began by exploring the raw data and identifying issues such as missing values, invalid entries, and inconsistent formats. Through careful cleaning and transformation, we prepared the data so it became accurate and ready for analysis. We also engineered new features, applied scaling, and integrated multiple datasets to create one complete and well-structured file.

Throughout the process, we used different visualizations to understand trends, correlations, and feature importance. We then trained and tested two machine-learning models, Random Forest and Logistic Regression, and compared their performance using accuracy scores and confusion matrices. The Random Forest model gave the best results and showed strong predictive power.

Overall, this project helped us enhance our data engineering skills, especially in cleaning, transforming, integrating, and preparing data for machine learning. It also showed the importance of following a structured pipeline to turn raw data into meaningful insights and reliable predictions.