Analysis of Heart Disease Stages with One-Versus-Rest Logistic Regression

GallantlyStellar

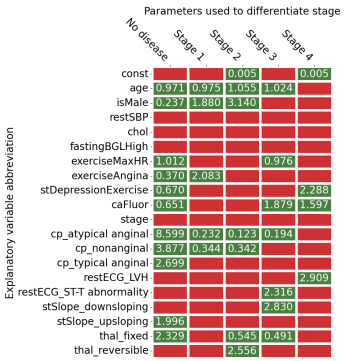
December 11, 2025

Abstract

A public dataset is available from the University of California, Irvine that contains several features and the heart disease stage of 918 patients from four distinct global regions (Detrano et al., 1989; Janosi et al., 1989). This dataset was used to fit one-versus-rest (OVR) logistic regression models to predict heart disease stage probabilities. These models were optimized by dropping statistically-insignificant features to determine which clinical features had significant impacts on identification of a given heart disease stage. Significant features are shown by stage in Figure 1. The OVR models were combined with a SoftMax function to create a model that can predict heart disease stage with an accuracy of 58.7%.

Figure 1

Presence matrix showing the explanatory variables incorporated by each model stage.



Note. Green annotated spaces represent included features while red spaces represent dropped features. Annotations show exponentiated coefficients (e^{β}), or the odds ratios. A one-unit increase in an explanatory variable is associated with a fold-change to the odds equal to the annotation (for a given stage) (Frost, 2025).

1 Introduction

1.1 Justification

Heart disease is responsible for approximately one in five deaths in the United States (Center for Disease Control and Prevention [CDC], 2024). Thus, data-driven predictions regarding heart disease and its contributing factors have the potential to improve quality of life and lessen healthcare expenses for individuals by predicting heart disease for undiagnosed patients based on clinical findings. This would allow healthcare providers to identify targets for definitive screening (thus facilitating early recognition) and provide recommendations based on existing data within a patient's medical record.

1.2 Research Question

A research question was formulated based on this problem. "To what extent do heart disease-related variables correlate with heart disease stages?" A public dataset was found containing data on age, sex, chest pain type, resting systolic blood pressure, total cholesterol, fasting blood sugar, resting ECG findings, maximum heart rate during exercise, presence of exercise-induced angina, ST depression during exercise, ST segment slope, coronary artery fluoroscopy results, and thallium scintography results; these features will be used as the "heart disease-related variables."

1.3 Context

In the United States, 22% of deaths are due to heart disease (CDC, 2024). Many risk factors are known to contribute to the incidence of heart disease and 47% of Americans present with one or more risk factors (CDC, 2024). An algorithmic way to screen patients for heart disease based on their clinical presentation has been the subject of prior reasearch (Detrano et al., 1989; Karabulut & İbrikçi, 2012).

The University of California, Irvine (UCI) maintains several public datasets for machine learning tasks (Janosi et al., 1989). The UCI Heart Disease Dataset

contains data for four regions around the globe that, when concatenated, include 920 rows with 13 variables to use for prediction and one variable to predict (the stage of heart disease) (Janosi et al., 1989). This dataset is licensed under CC-BY-4.0 and is thus free of confidential or proprietary information; additionally, it contains only deindentified information. This data is used in prior research (Detrano et al., 1989; Karabulut & İbrikçi, 2012).

The paper that introduced this dataset used clinical features (age, gender, chest pain type, and systolic blood pressure), routine test results (serum cholesterol, fasting blood glucose level, and electrocardiogram features), and noninvasive cardiology tests (exercise electrocardiogram features, exercise thallium scintigraphy, and coronary artery fluoroscopy) to create a logistic regression model to predict heart disease in patients from various cities (Detrano et al., 1989). The original algorithm was trained only on patients from Cleveland and tested on patients from other regions; additionally, this algorithm was found to generally over-predict the probability of heart disease (in the Long Beach and Hungarian cohorts) in patients but to under-predict in the Swiss cohort (Detrano et al., 1989).

This study will use one-versus-rest multiclass logistic regression (OVR) on a combined cohort to make similar predictions and create a model of a more global nature in the hopes of creating a model than generalizes more effectively. Features that have a statistically significant impact on the target variable will be incorporated into a final model that will be evaluated on unseen data. By selecting these coefficients using a statistical test, only significantly correlated features will be retained.

1.4 Hypothesis

The research hypothesis for this project will be that all of the clinical features (listed in 1.2) have a significant impact on the prediction of heart disease.

The null hypothesis can be expressed mathematically as:

$$H_0 : \beta_i = 0$$

Where β_i is the coefficient fit by the logistic regression model for feature i.

As coefficients can be positive or negative, the alternate hypothesis can be expressed as a bidirectional inequality.

$$H_a : \beta_i \neq 0$$

The significance level for this project will be $\alpha = 0.05$. The null hypothesis will be rejected if a two-sided z-test provides a p-value that is less than or equal to this value for any parameter.

2 Data Collection

2.1 Collection Process

The dataset can be downloaded from UCI. There are four datasets within the archive that are of interest; one dataset for each locality's cohort of patients. Data is available from the V.A. Medical Center in Long Beach, California, the Hungarian Institute of Cardiology in Budapest, Hungary, the Cleveland Clinic Foundation in Cleveland, Ohio, and from the University Hospital in Zurich, Switzerland (Janosi et al., 1989).

To ensure integrity, the data were controlled with DVC to ensure that dataset hashes do not change. Changes to the underlying dataset were only be performed in memory to not alter or bias the source data.

2.2 Advantages and Disadvantages

Use of a public dataset allowed this project to proceed at a much faster rate; establishing community relationships and obtaining IRB approval to utilize patient data would add significant time and administrative burden to this project.

However, the UCI dataset was compiled in 1989 (Janosi et al., 1989). It is possible that there are features that can be measured modernly that are of great potential predictive value, as well, and the economics of certain tests could have changed. It would be reasonable to consult a subject matter expert about the relevance of these findings to modern healthcare before implementation in a health system.

2.3 Collection Challenges

The UCI dataset is has many features. The Cleveland cohort contains 76 distinct features (Janosi et al., 1989). However, as stated above, this project aims to perform an analysis with a global character. Only the features reported by all four localities were used in this analysis to prevent the need for mass imputation of many features only available for the Cleveland cohort.

3 Data Extraction and Preparation

This analysis was performed using the Python language (Python Software Foundation, 2025).

3.1 Data Preparation Process

Processes to import, clean, and prepare the data for the analysis were performed using the Pandas library (NumFOCUS, Inc., 2025).

3.1.1 Importing Data

The datasets for each locality were first imported and combined.

```
path = "../assets/data/raw"
dfva = pd.read_csv(path +
  ↪  "/processed.va.data", na_values="?",
  ↪  header=None)
dfva.index = dfva.index.astype(str) + "va"
dfhu = pd.read_csv(path +
  ↪  "/processed.hungarian.data",
  ↪  na_values="?", header=None)
dfhu.index = dfhu.index.astype(str) + "hu"
dfcl = pd.read_csv(path +
  ↪  "/processed.cleveland.data",
  ↪  na_values="?", header=None)
dfcl.index = dfcl.index.astype(str) + "cl"
dfsw = pd.read_csv(path +
  ↪  "/processed.switzerland.data",
  ↪  na_values="?", header=None)
dfsw.index = dfsw.index.astype(str) + "sw"

df = pd.concat([dfva, dfhu, dfcl, dfsw])
```

Figure 2
The concatenated data after importing.

```
In [12]: df = importUCI()
          0    1    2      3      4    5    6      7    8    9   10   11   12  13
0va    63.0  1.0  4.0  140.0  260.0  0.0  1.0  112.0  1.0  3.0  2.0  NaN  NaN   2
1va    44.0  1.0  4.0  130.0  209.0  0.0  1.0  127.0  0.0  0.0  NaN  NaN  NaN   0
2va    60.0  1.0  4.0  132.0  218.0  0.0  1.0  140.0  1.0  1.5  3.0  NaN  NaN   2
3va    55.0  1.0  4.0  142.0  228.0  0.0  1.0  149.0  1.0  2.5  1.0  NaN  NaN   1
4va    66.0  1.0  3.0  110.0  213.0  1.0  2.0   99.0  1.0  1.3  2.0  NaN  NaN   0
...     ...  ...  ...    ...    ...  ...  ...    ...  ...  ...  ...  ...  ...  ..
118sw  70.0  1.0  4.0  115.0    0.0  0.0  1.0   92.0  1.0  0.0  2.0  NaN  7.0   1
119sw  70.0  1.0  4.0  140.0    0.0  1.0  0.0  157.0  1.0  2.0  2.0  NaN  7.0   3
120sw  72.0  1.0  3.0  160.0    0.0  NaN  2.0  114.0  0.0  1.6  2.0  2.0  NaN   0
121sw  73.0  0.0  3.0  160.0    0.0  0.0  1.0  121.0  0.0  0.0  1.0  NaN  3.0   1
122sw  74.0  1.0  2.0  145.0    0.0  NaN  1.0  123.0  0.0  1.3  1.0  NaN  NaN   1

[920 rows x 14 columns]
```

3.1.2 Setting Column Names

Column names were then set manually, as the source datafiles did not have any header rows. The UCI-provided "heart-disease.names" file provided descriptions of each column based on index positions.

```
df.columns = [
    "age",
    "isMale",
    "cp",
    "restSBP",
    "chol",
    "fastingBGLHigh",
    "restECG",
    "exerciseMaxHR",
    "exerciseAngina",
    "stDepressionExercise",
    "stSlope",
    "caFluor",
    "thal",
    "stage",
]
```

Figure 3
The data with column names matched from the UCI data dictionary (Janosi et al., 1989).

```
In [12]: df = importUCI()
         age  isMale   cp  restSBP  ...  stSlope  caFluor  thal  stage
0va     63.0     1.0  4.0    140.0  ...      2.0      NaN   NaN      2
1va     44.0     1.0  4.0    130.0  ...      NaN      NaN   NaN      0
2va     60.0     1.0  4.0    132.0  ...      3.0      NaN   NaN      2
3va     55.0     1.0  4.0    142.0  ...      1.0      NaN   NaN      1
4va     66.0     1.0  3.0    110.0  ...      2.0      NaN   NaN      0
...      ...     ...  ...      ...  ...      ...      ...   ...    ...
118sw   70.0     1.0  4.0    115.0  ...      2.0      NaN   7.0      1
119sw   70.0     1.0  4.0    140.0  ...      2.0      NaN   7.0      3
120sw   72.0     1.0  3.0    160.0  ...      2.0      2.0   NaN      0
121sw   73.0     0.0  3.0    160.0  ...      1.0      NaN   3.0      1
122sw   74.0     1.0  2.0    145.0  ...      1.0      NaN   NaN      1

[920 rows x 14 columns]
```

3.1.3 Setting Data Types

Some dtypes were then manually set to use less system memory and to discourage unexpected behavior from data type inconguence. Some column datatypes were not set during this stage as label-encoded values would later require replacement to incompatible types.

```
df = df.astype(
    {
        "age": "uint8",
        "isMale": "uint8",
        # "cp" currently label encoded
        "restSBP": "Int16",
        "chol": "Int16",
        "fastingBGLHigh": "Int8",
        # "restECG" currently label encoded
        "exerciseMaxHR": "Int16",
        "exerciseAngina": "Int8",
        "stDepressionExercise": "float32",
```

```
        # "stSlope" currently label encoded
        "caFluor": "Int8",
        # "thal" currently label encoded
        "stage": "uint8",
    }
)
```

Figure 4
Data types as automatically assigned before explicit typecasting.

```
In [12]: df = importUCI()
<class 'pandas.core.frame.DataFrame'>
Index: 920 entries, 0va to 122sw
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   age                     920 non-null    float64
 1   isMale                  920 non-null    float64
 2   cp                      920 non-null    float64
 3   restSBP                 861 non-null    float64
 4   chol                    890 non-null    float64
 5   fastingBGLHigh          830 non-null    float64
 6   restECG                 918 non-null    float64
 7   exerciseMaxHR           865 non-null    float64
 8   exerciseAngina          865 non-null    float64
 9   stDepressionExercise    858 non-null    float64
 10  stSlope                 611 non-null    float64
 11  caFluor                 309 non-null    float64
 12  thal                    434 non-null    float64
 13  stage                   920 non-null    int64
dtypes: float64(13), int64(1)
memory usage: 107.8+ KB
None
```

Figure 5
Data types as explicitly cast (prior to replacement of label-encoded values).

```
In [12]: df = importUCI()
<class 'pandas.core.frame.DataFrame'>
Index: 920 entries, 0va to 122sw
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   age                     920 non-null    uint8
 1   isMale                  920 non-null    uint8
 2   cp                      920 non-null    float64
 3   restSBP                 861 non-null    Int16
 4   chol                    890 non-null    Int16
 5   fastingBGLHigh          830 non-null    Int8
 6   restECG                 918 non-null    float64
 7   exerciseMaxHR           865 non-null    Int16
 8   exerciseAngina          865 non-null    Int8
 9   stDepressionExercise    858 non-null    float32
 10  stSlope                 611 non-null    float64
 11  caFluor                 309 non-null    Int8
 12  thal                    434 non-null    float64
 13  stage                   920 non-null    uint8
dtypes: Int16(3), Int8(3), float32(1), float64(4), uint8(3)
memory usage: 55.7+ KB
None
```

Figure 6
The data after setting preliminary data types.

```
In [12]: df = importUCI()
      age  isMale   cp  restSBP  ...  stSlope  caFluor  thal  stage
0va    63       1  4.0      140  ...      2.0     <NA>   NaN      2
1va    44       1  4.0      130  ...      NaN     <NA>   NaN      0
2va    60       1  4.0      132  ...      3.0     <NA>   NaN      2
3va    55       1  4.0      142  ...      1.0     <NA>   NaN      1
4va    66       1  3.0      110  ...      2.0     <NA>   NaN      0
...   ...     ...  ...      ...  ...      ...      ...   ...    ...
118sw  70       1  4.0      115  ...      2.0     <NA>   7.0      1
119sw  70       1  4.0      140  ...      2.0     <NA>   7.0      3
120sw  72       1  3.0      160  ...      2.0        2   NaN      0
121sw  73       0  3.0      160  ...      1.0     <NA>   3.0      1
122sw  74       1  2.0      145  ...      1.0     <NA>   NaN      1

[920 rows x 14 columns]
```

3.1.4 Reversal of Label Encoding

Some columns (indicated above as commented lines of code) contained integer values corresponding to qualitative variables. These integer values were replaced with their respective labels to facilitate one-hot encoding later on in this analysis as the values do not convey quantitative or ordinal meaning. Replacements were performed in accordance with the value definitions distributed by UCI alongside the datasets.

```
df["cp"] = (
    df["cp"]
    .replace({1: "typical anginal", 2:
    ↪  "atypical anginal", 3:
    ↪  "nonanginal", 4: "asymptomatic"})
    .astype("category")
)
df["restECG"] = (
    df["restECG"].replace({0: "normal", 1:
    ↪  "ST-T abnormality", 2:
    ↪  "LVH"}).astype("category")
)
df["stSlope"] = (
    df["stSlope"].replace({1: "upsloping",
    ↪  2: "flat", 3:
    ↪  "downsloping"}).astype("category")
)
df["thal"] = df["thal"].replace({3:
  ↪  "fixed", 6: "reversible", 7:
  ↪  "none"}).astype("category")
```

Figure 7
Data following replacement of label-encoded values with the relevant labels.

```
In [12]: df = importUCI()
       age  isMale               cp  restSBP  ...      stSlope  caFluor   thal  stage
0va     63       1   asymptomatic      140  ...         flat     <NA>    NaN      2
1va     44       1   asymptomatic      130  ...          NaN     <NA>    NaN      0
2va     60       1   asymptomatic      132  ...  downsloping     <NA>    NaN      2
3va     55       1   asymptomatic      142  ...    upsloping     <NA>    NaN      1
4va     66       1     nonanginal      110  ...         flat     <NA>    NaN      0
...    ...     ...            ...      ...  ...          ...      ...    ...    ...
118sw   70       1   asymptomatic      115  ...         flat     <NA>   none      1
119sw   70       1   asymptomatic      140  ...         flat     <NA>   none      3
120sw   72       1     nonanginal      160  ...         flat        2    NaN      0
121sw   73       0     nonanginal      160  ...    upsloping     <NA>  fixed      1
122sw   74       1  atypical anginal   145  ...    upsloping     <NA>    NaN      1

[920 rows x 14 columns]
```

3.1.5 Deduplication

Finally, two duplicates were dropped. Since the data are de-identified, it is possible that these are not true duplicates (and are simply improbable collisions), but the identical values for continuous numeric values like resting systolic blood pressure and total cholesterol makes this seem less likely. Additionally, in the case of one of the duplicates, the rows were adjacent, suggesting a possible data entry error where one patient was imported twice. Also notably, both duplicates come from the same locality.

Figure 8
The duplicates within the dataset.

```
In [12]: df = importUCI()
       age  isMale               cp  restSBP  ...  stSlope  caFluor thal  stage
139va   58       1        nonanginal      150  ...      NaN     <NA>  NaN      2
187va   58       1        nonanginal      150  ...      NaN     <NA>  NaN      2
101hu   49       0  atypical anginal      110  ...      NaN     <NA>  NaN      0
102hu   49       0  atypical anginal      110  ...      NaN     <NA>  NaN      0

[4 rows x 14 columns]
```

```
df.drop_duplicates(inplace=True)
```

Figure 9
The data after deduplication. Note that the number of rows has decreased from 920 to 918.

```
In [12]: df = importUCI()
       age  isMale               cp  restSBP  ...      stSlope  caFluor   thal  stage
0va     63       1   asymptomatic      140  ...         flat     <NA>    NaN      2
1va     44       1   asymptomatic      130  ...          NaN     <NA>    NaN      0
2va     60       1   asymptomatic      132  ...  downsloping     <NA>    NaN      2
3va     55       1   asymptomatic      142  ...    upsloping     <NA>    NaN      1
4va     66       1     nonanginal      110  ...         flat     <NA>    NaN      0
...    ...     ...            ...      ...  ...          ...      ...    ...    ...
118sw   70       1   asymptomatic      115  ...         flat     <NA>   none      1
119sw   70       1   asymptomatic      140  ...         flat     <NA>   none      3
120sw   72       1     nonanginal      160  ...         flat        2    NaN      0
121sw   73       0     nonanginal      160  ...    upsloping     <NA>  fixed      1
122sw   74       1  atypical anginal   145  ...    upsloping     <NA>    NaN      1

[918 rows x 14 columns]
```

3.1.6 Visualization

The imported, combined data can then be viewed. Univariate and bivariate distributions can also be examined with matplotlib (Hunter et al., 2025).

Figure 10
Univariate distributions of variables within the dataset. Colors present to distinguish plots and do not convey meaning.

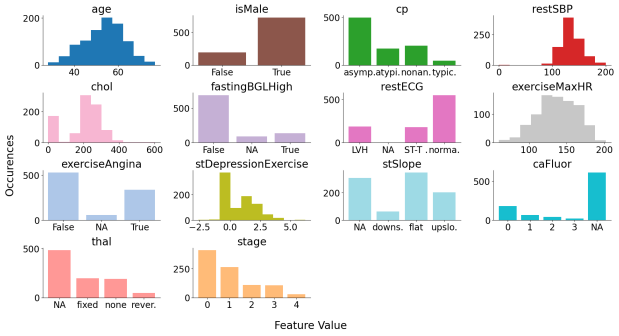
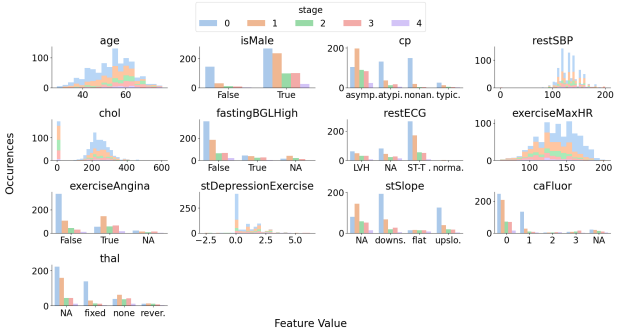


Figure 11
Bivariate distributions of variables with respect to heart disease stage.



3.1.7 Impute Missing Values

Many values are missing within the provided datatsets. These can be grouped by locality and viewed with the missingno package to visualize patterns.

```
from missingno import matrix

# num pts from each site
lenva = df.index.str.contains("va").sum()
  ↪   # VA in Long Beach
lenhu = df.index.str.contains("hu").sum()
  ↪   # Hungary
```

```
lencl = df.index.str.contains("cl").sum()  # Cleveland
lensw = df.index.str.contains("sw").sum()  # Switzerland

# offsets of 0.5 are due to msno matrix defaults
start = -0.5
stop = lenva - 0.5

# create msno matrix plot
ax = matrix(df)
# shade each region
ax.axhspan(start, stop, color="#009E73", alpha=0.3, label="VA")
start += stop - start
stop += lenhu
ax.axhspan(start, stop, color="#D55E00", alpha=0.3, label="Hungary")
start += stop - start
stop += lencl
ax.axhspan(start, stop, color="#56B4E9", alpha=0.3, label="Cleveland")
start += stop - start
stop += lensw
ax.axhspan(start, stop, color="#CC79A7", alpha=0.3, label="Switzerland")
# position legend below, centered
ax.legend(loc="lower center", bbox_to_anchor=(0.5, -0.12), ncols=4)
ax.set_ylabel("Patient")
```

Figure 12
Missing values grouped by locality of origin.

This figure suggests that many of the clinical features are absent from some of the localities (particularly the VA). Since the data were compiled by a team from the Cleveland region, it makes sense that those data appear to be the most complete. It is possible that some clinical features are more commonly assessed in some localities than in others. The original paper does not provide an explanation for this (Detrano et al., 1989). The lack of coronary artery fluoroscopy in other localities in meaningful quantities suggests that it is unlikely to have much predictive value.

Values can then be imputed by median (to resist outliers compared to a mean) for quantitative values or as mode for qualitative values. Some impossible values (like a blood pressure of zero) are also imputed.

```
# quantitative values as median (to resist outliers)
for quantCol in ["restSBP", "chol", "exerciseMaxHR", "stDepressionExercise"]:
    df.loc[df[quantCol].isna(), quantCol] = round(df[quantCol].median())

# qualitative (and CA fluoroscopy due to only 3 discrete values) as mode
for qualCol in ["fastingBGLHigh", "restECG", "exerciseAngina", "stSlope", "caFluor", "thal"]:
    df.loc[df[qualCol].isna(), qualCol] = df[qualCol].mode()[0]

# inappropriate values
df.loc[df["restSBP"] == 0, "restSBP"] = round(df["restSBP"].mean())
df.loc[df["chol"] == 0, "chol"] = round(df["chol"].mean())
```

Figure 13
Data after imputation of missing or nonsensical values.

```
In [17]: df
Out[17]:
      age  isMale               cp  restSBP  ...      stSlope  caFluor   thal  stage
0va    63       1   asymptomatic      140  ...         flat        0  fixed      2
1va    44       1   asymptomatic      130  ...         flat        0  fixed      0
2va    60       1   asymptomatic      132  ...   downsloping       0  fixed      2
3va    55       1   asymptomatic      142  ...     upsloping       0  fixed      1
4va    66       1     nonanginal      110  ...         flat        0  fixed      0
...   ...     ...            ...      ...  ...          ...      ...    ...    ...
118sw  70       1   asymptomatic      115  ...         flat        0   none      1
119sw  70       1   asymptomatic      140  ...         flat        0   none      3
120sw  72       1     nonanginal      160  ...         flat        2  fixed      0
121sw  73       0     nonanginal      160  ...     upsloping       0  fixed      1
122sw  74       1  atypical anginal   145  ...     upsloping       0  fixed      1

[918 rows x 14 columns]
```

3.1.8 Reset Data Types

Finally, statsmodels will be used to to fit the model and was found to be incompatible with some of the

Pandas nullable integer types (that are no longer necessary post-imputation). Some columns were consequently cast to numpy datatypes.

```
df = df.astype(
    {
        "restSBP": "uint8",
        "chol": "uint16",
        "fastingBGLHigh": "uint8",
        "exerciseMaxHR": "uint8",
        "exerciseAngina": "uint8",
        "caFluor": "uint8",
    }
)
```

Figure 14

Data types after resetting pandas datatypes to numpy types. Neither values nor indices were changed by this operation.

```
In [22]: df.info()
<class 'pandas.core.frame.DataFrame'>
Index: 918 entries, 0va to 122sw
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   age                   918 non-null    uint8
 1   isMale                918 non-null    uint8
 2   cp                    918 non-null    category
 3   restSBP               918 non-null    uint8
 4   chol                  918 non-null    uint16
 5   fastingBGLHigh        918 non-null    uint8
 6   restECG               918 non-null    category
 7   exerciseMaxHR         918 non-null    uint8
 8   exerciseAngina        918 non-null    uint8
 9   stDepressionExercise  918 non-null    float32
 10  stSlope               918 non-null    category
 11  caFluor               918 non-null    uint8
 12  thal                  918 non-null    category
 13  stage                 918 non-null    uint8
dtypes: category(4), float32(1), uint16(1), uint8(8)
memory usage: 56.2+ KB
```

3.1.9 One-Hot Encoding

The final step of data preparation was one-hot encoding of categorical variables.

```
# one-hot encoding
df = pd.get_dummies(df, dtype="uint8",
  ↪  drop_first=False)
# keep first to drop normal class manually
# so coeffs are changes from baseline
df = df.drop(
    [
        "cp_asymptomatic",
        "restECG_normal",
        "stSlope_flat",
        "thal_none",
    ],
    axis=1,
)
```

Following this, the data are fully cleaned and ready for logistic regression.

Figure 15

Sample of the fully cleaned, one-hot encoded data.

```
In [23]: df.sample(15)
Out[23]:
      age  isMale  restSBP  chol  ...  stSlope_downsloping  stSlope_upsloping  thal_fixed  thal_reversible
25hu   37       1      120   223  ...                    0                  0           1                0
256hu  47       0      135   248  ...                    0                  0           1                0
43hu   40       1      130   281  ...                    0                  0           1                0
72hu   45       0      180   223  ...                    0                  0           1                0
120hu  51       1      125   188  ...                    0                  0           1                0
125cl  45       0      130   234  ...                    0                  0           1                0
122sw  74       1      145   200  ...                    0                  1           1                0
26hu   37       1      130   315  ...                    0                  0           1                0
107sw  65       1      115   200  ...                    0                  0           0                0
183cl  59       1      178   270  ...                    1                  0           0                0
155hu  54       1      150   365  ...                    0                  1           1                0
59hu   43       0      100   223  ...                    0                  0           1                0
98cl   66       1      120   302  ...                    0                  0           1                0
112cl  52       1      118   186  ...                    0                  0           0                1
242cl  49       0      130   269  ...                    0                  1           1                0
```

3.2 Tools, Techniques, and Justification

To complete these preparation steps, the Python programming language was used (Python Software Foundation, 2025). Several packages were used to extend the base functionality of Python.

3.2.1 Data preparation

1. Python was used over other statistical computing environments due to author familiarity and the combined extensibility and utility of the Matplotlib and Statsmodels libraries.

2. Pandas was used to store the tabular data and perform most of the preprocessing steps (NumFOCUS, Inc., 2025).

3. Matplotlib was used to visualize the data during the preprocessing steps. It was also used to visualize model results later in the analysis (Hunter et al., 2025).

4. Seaborn provided a convenient wrapper around Matplotlib to make heatmaps and bivariate visualizations easier to produce (Waskom, 2024).

5. Missingno was used to create a plot of the locality-grouped variables to look for patterns in missing values (Bilogur, 2023).

3.2.2 Logistic Regression

The subsequent analysis of these data will also use additional packages.

1. Sci-Kit Learn was used to perform a stratified, shuffled split into training and testing data. It also provided an accuracy score function that was used for model performance evaluation (Buitinck et al., 2025).

2. Statsmodels was used to perform the logistic regression (Perktold et al., 2025). It calculates and exposes statistical measures (like z-test p-values) that are more useful that those provided out-of-the-box by Sci-Kit Learn models.

3. SciPy was used to provide a softmax function to convert the individual logistic regression stage outcomes into probabilities that all sum to one as described in 4.2.2 (Virtanen et al., 2025).

3.2.3 Advantages and Disadvantages

Imputing values in this way, as described in 3.1.7, confers an advantage in that the predictive value of columns where at least one value is absent need not be dropped (as all but three explanatory variables have missing values); this allows those features to provide predictive value instead of discarding potentially-meaningful data. Median (or mode, for categorical variables) imputation is simple and fast to compute and is somewhat robust against outliers (when compared to mean imputation).

However, there are some disadvantages to this approach. Some features have more imputed values than values initially present in the data; Figure 12 presents these graphically. Columns with a large proportion of imputed values (like coronary artery fluoroscopy results) are unlikely to be of much predictive value outside of the Cleveland cohort due to data sparsity. If this feature is not dropped as insignificant by a model, its results should thus be interpreted with caution. Also, mode imputation may bias models towards the majority class.

4 Analysis

4.1 Multinomial Logistic Regression Attempt

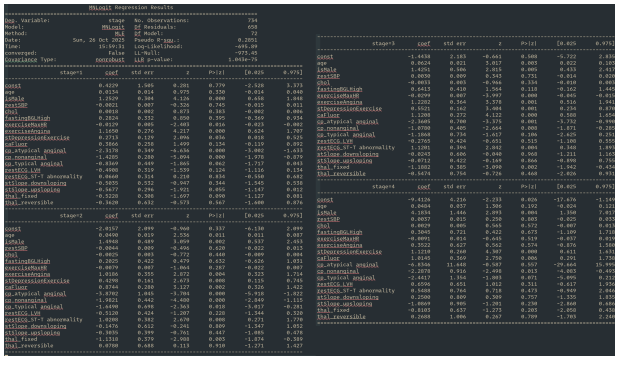
The analysis next used logistic regression to create a model. Initially, multinomial logistic regression was attempted using the Statsmodels package.

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import
  ↪  train_test_split
from statsmodels.api import MNLogit,
  ↪  add_constant

dfMN = df.copy(deep=True)  # don't mutate
  ↪  df
predictors = dfMN.columns.drop("stage")  #
  ↪  columns for X
X_train, X_test, y_train, y_test =
  ↪  train_test_split(
    dfMN[predictors],
    dfMN["stage"],
    test_size=0.2,
    random_state=1571,
    stratify=dfMN["stage"],
)

X_train = add_constant(X_train)
model = MNLogit(y_train, X_train).fit()

pred = model.predict(add_constant(X_test))
pred = pred.idxmax(axis=1)
print("Test set accuracy:",
  ↪  accuracy_score(y_test, pred))
```

Figure 16
Accuracy of the initial multinomial logistic regression model.

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.948077
         Iterations: 35
                                                analysis/.ve
statsmodels/base/model.py:607: ConvergenceWarning: Maximum
to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to
Test set accuracy: 0.5380434782608695
```

Figure 17
Summary (via the model.summary() method) of the multinomial logistic regression model. Calculations performed with Statsmodels (Perktold et al., 2025).



4.2 OVR Model

4.2.1 Justification, Advantages, and Disadvantages

Since multinomial logistic regression computes one model fewer than the number of levels of the response variable, each stage is a difference-from-baseline model (Tilevik, 2024, 8:10; Perktold et al., 2025). Consequently, simple dropping of features with statistically insignificant coefficients is not efficient. Thus, this model is not the best way to answer the research question, as individual features can only be considered in how they affect all stages, when it is possible that some features are only impactful on some stage predictions (this is preliminarily suggested by the model summary; for example, the age coefficient is significant at all stages except stage four with the multinomial model).

A one-verses-rest (OVR) model was then instead developed to allow consideration of independent feature sets for prediction of each stage. In this OVR model, each stage predicts whether a patient is part of a stage (1) or not part of a stage (0) (Tilevik, 2024, 4:03). Fitting the stages in this way allows for each stage to incorporate a separate set of features. Also, the ability to drop one feature at a time within a given stage can prevent model deterioration due to dropping two insignificant features that may have exhibited some multicolinearity.

This OVR model provides an advantage in that each stage is easy to conceptualize in terms of monomial logistic regression. Each model is also quick and efficient to train. This method could have disadvantages, particularly if there were many more levels of the response variable, as each model has to be trained individually. Another disadvantage is that the error of each stage propagates to the combined model; each stage can predict with a generally favorable accuracy of around 80%, but the final model propagates and compounds these errors to a final accuracy of 58.7%.

Logistic regression was selected over other classification methods as it permits assessment of the statistical significance of each coefficient, so is most in line with the research question.

4.2.2 Equations

Logistic regression uses the logit function on the response variable to transform a problem into one that can be modelled linearly. (Shumeli et al., 2019, Ch. 10.2). A linear model is constructed to fit a simple equation (Shumeli et al., 2019, Eq. 10.1).

$$p = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

Where p is the probability a patient belongs to a given stage, β is one of the calculated coefficients, and x is any of the available explanatory variables.

To bound this between zero and one, Euler's constant e is used to create the logistic response function.

$$p = \frac{1}{1 + e^{\beta_0 + \beta_1 x_1 + ... + \beta_n x_n}}$$

The odds of an event are defined as $odds_{stage=True} = \frac{p}{1-p}$; a simple substitution can be made to define a relationship between the predictors and the resultant odds (Shumeli et al., 2019, Eqs. 10.5-10.6).

$$odds_{stage=True} = e^{\beta_0 + \beta_1 x_1 + ... + \beta_n x_n}$$

$$log_e(odds_{stage=True}) = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

This analysis selected significant values of β, calculated them, and used these principals to interpret the meanings of these values.

Since conventional (and OVR) logistic regression models require binary (not multi-class, as in the case of the UCI heart data) response variables, a different equation was calculated for each stage (Frost, 2025). These probabilities can be combined using the SoftMax function so that the probability of each class can be obtained (Sanchhaya Education Private Limited [Sanchhaya Ed.], 2025; Virtanen et al., 2025).

$$\sigma(\{p_{model0}, p_{model1}, ..., p_{model4}\})_j = \frac{e^{p_j}}{\sum_{k=model0}^{model4} e^{p_k}}$$

With these, one set of explanatory variables can be fed through each of the five stage-specific models to obtain the probability of that stage being present compared to that stage being absent. These probabilities can then be SoftMax-transformed to calculate the probability for each stage with respect to each other, as is done in multinomial regression (Sanchhaya Ed., 2025; Tilevik, 2024, 10:36).

4.2.3 Calculations

Each stage was trained with a custom function.

```
def trainStage(
    df: pd.DataFrame, stage: int, colDrop: List[str] = [], alpha: float = 0.05
) -> ResultsWrapper:
    """Train a single logistic regression model for a given stage."""
    from sklearn.metrics import accuracy_score
    from sklearn.model_selection import train_test_split
```

```
    from statsmodels.api import Logit, add_constant

    # Predict if it belongs to a stage (0) or not (1)
    dfStage = df.copy(deep=True)  # don't mutate df
    dfStage["stageSource"] = dfStage["stage"]
    dfStage.loc[dfStage["stageSource"] == stage, "stage"] = 1
    dfStage.loc[dfStage["stageSource"] != stage, "stage"] = 0
    dfStage.drop(["stageSource"], axis=1, inplace=True)

    # Assumption 1: binary response levels
    assert dfStage["stage"].nunique() == 2, "Response levels must be binary"

    predictors = dfStage.columns.drop("stage")  # columns for X
    X_train, X_test, y_train, y_test = train_test_split(
        dfStage[predictors],
        dfStage["stage"],
        test_size=0.2,
        random_state=1571,
        stratify=dfStage["stage"],
    )

    # Droping cols here lets arbitrary cols, incl const
    # be dropped if needed
    X_train = add_constant(X_train).drop(colDrop, axis=1)
    X_test = add_constant(X_test).drop(colDrop, axis=1)
    model = Logit(y_train, X_train).fit()

    # Drop for p-values below alpha
    p-val = model.pvalues
    p-val = p-val.sort_values(ascending=False)
    while p-val.iloc[0] >= alpha:
        X_train = X_train.drop(p-val.index[0], axis=1)
        X_test = X_test.drop(p-val.index[0], axis=1)
        del model
        model = Logit(y_train, X_train).fit()

    pred = model.predict(X_test).round()
    print(f"Test set accuracy for stage {stage}: {accuracy_score(y_test, pred)}")
    return model

model0 = trainStage(df, stage=0)
model1 = trainStage(df, stage=1)
model2 = trainStage(df, stage=2)
model3 = trainStage(df, stage=3)
model4 = trainStage(df, stage=4)
models = [model0, model1, model2, model3, model4]
```

This function will also feature engineer each OVR model stage by dropping the coefficient with the largest p-value and refitting that stage. Only one coefficient is dropped at a time due to the possibility of multicollinearity leading to predictive features being erroneously dropped. This process was repeated until only features with a p-value below $\alpha = 0.05$ remained. This operation provides an answer to the research question by tuning each OVR stage model until only features with a statistically significant impact on the outcome remain.

The accuracy scores for each stage were calculated during training.

Figure 18

Accuracy scores on the test set for each stage of the OVR model. Calculated with (Buitinck et al., 2025).

```
In [32]: accuracies
Out[32]:
Stage 0    0.782609
Stage 1    0.706522
Stage 2    0.853261
Stage 3    0.885870
Stage 4    0.956522
Name: Test set accuracy scores per stage, dtype: float64
```

The accuracy of the overall model was calculated by taking the softmax of each OVR stage and using the largest probability to predict the most likely stage.

```
def testStages(df: pd.DataFrame, models: List[ResultsWrapper]) -> pd.DataFrame:
    """Apply the models to a test set and use softmax to return probabilities."""
    from scipy.special import softmax
    from sklearn.metrics import accuracy_score
    from sklearn.model_selection import train_test_split
```

```
    from statsmodels.api import
    ↪  add_constant

    predictorsAll =
    ↪  df.columns.drop("stage")
    _X_train, X_test, _y_train, y_test =
    ↪  train_test_split(
        df[predictorsAll],
        df["stage"],
        test_size=0.2,
        random_state=1571,
        stratify=df["stage"])
    stage = 0
    results = pd.DataFrame()
    for model in models:
        # This allows testing if model had
        ↪  constant dropped during
        ↪  training
        X_test_local =
        ↪  X_test.copy(deep=True)
        if "const" in model.params.index:
            X_test_local =
            ↪  add_constant(X_test_local)
        predColDrop = [
            item for item in
            ↪  X_test_local.columns if
            ↪  item not in
            ↪  model.params.index
        ]
        X_test_local =
        ↪  X_test_local.drop(predColDrop,
        ↪  axis=1)
        series =
        ↪  model.predict(X_test_local)
        series = pd.Series(series,
        ↪  name=stage)
        results = pd.concat([results,
        ↪  series], axis=1)
        stage += 1
    results = results.apply(softmax,
    ↪  axis=1).apply(pd.Series)
    print(f"Combined test accuracy:
    ↪  {accuracy_score(y_test,
    ↪  results.idxmax(axis=1))}")
    return results
```

Figure 19
The accuracy of the combined model is 58.7%.

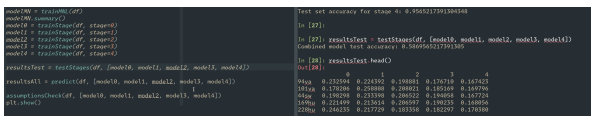
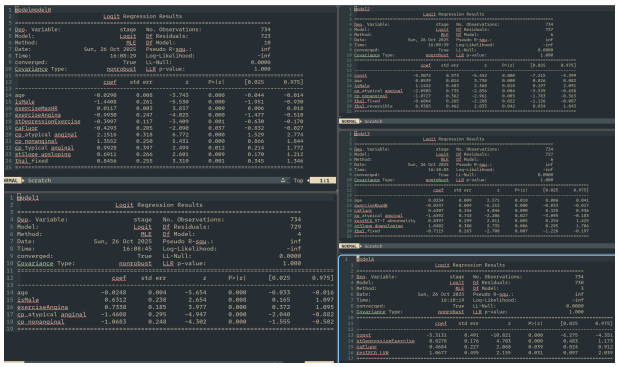


Figure 20
Summaries for each of the OVR models. Calculated with (Perktold et al., 2025).



4.2.4 Assumption Verification

Logistic regression has several underlying assumptions (Bobbit, 2020; Bryant, 2024; Frost, 2025).

1. The response variable (within each stage) is binary.

   This was asserted during training of each stage.

```
assert dfStage["stage"].nunique()
↪  == 2, "Response levels must be
↪  binary"
```

2. Observations are idependent

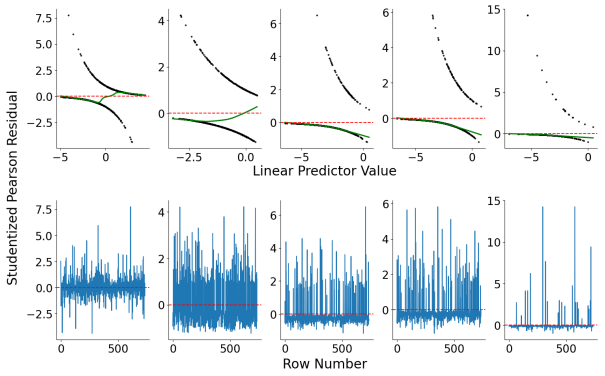
   This was hadled during data cleaning and can be asserted.

```
assert ~df.duplicated().any(), "Check
↪  for duplicates/independence of
↪  rows"
```

   Additionally, Studentized Pearson Residuals can be plotted to verify that the residuals do not exhibit any clear patterns (Bryant, 2024, "Assumption Check").

Figure 21

Plots of residuals.

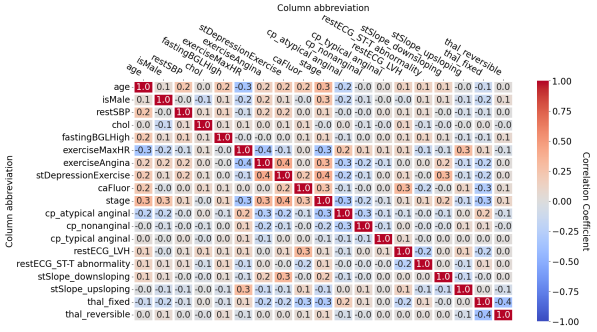


Note. The top row shows the estimated probability versus the Studentized Pearson residuals. Ideally, the regression line with LoWeSS smothing (in green) should have zero slope and intercept (Bryant, 2024, "Logistic Regression Residuals"). The bottom row shows the residual for each row; ideally, residuals will be randomly and normally distributed around a mean of zero. These plots suggest small deviations of these assumptions; this will be included as a caution in the discussion of model implications.

3. No multicolinearity exists between different explanatory variables.

Figure 22

Heatmap showing minimal multicolinearity exists between clinical features.



4. The sample size for each response variable level is large.

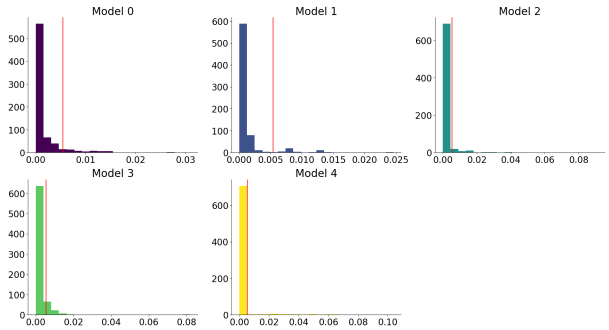
   This can be verified with a simple assert statement.

```
assert (df["stage"].value_counts()
    >= 10).all(), "Insufficient
    number of rows"
```

5. No extreme outliers exist.

Figure 23

Plot of the distribution of Cook's number showing influence for each stage model.



Note. Ideally, no values exceed the red vertical line. Each stage, particularly stage 3, has some excessively-influential values. This understanding will be included as a caution in the discussion of model implications.

Figure 24

The proportions of influential values (Cook's Number $\geq 4/n$) for each stage.

```
In [30]: assumptionsCheck(df, [model0, model1, model2, model3, model4])
Proportion of influential values for model 1: 0.07220708446866485
Proportion of influential values for model 2: 0.07084468664850137
Proportion of influential values for model 3: 0.05313351498637602
Proportion of influential values for model 4: 0.10626702997275204
Proportion of influential values for model 5: 0.039509536784741145
```

6. There is linearity between the log odds of the response variable and the explanatory variable.

   Checking this graphically would require 18 plots (one per encoded explanatory variable) for each stage. Instead, the Box-Tidwell test can be performed to verify this assumption in a rigorous way.

Figure 25

Box-Tidwell test results suggesting that no explanatory variables violate the assumption of linearity to a statistically significant extent (the test fails to reject the null hypothesis that linearity exists).

```
p-values for Box-Tidwell test:
                                0         1         2         3
age_log                  0.823219  0.526514  0.430934  0.695028
restSBP_log              0.361602  0.633107  0.984329  0.874911
chol_log                 0.972993  0.725601  0.174413  0.937247
exerciseMaxHR_log        0.134103  0.367608  0.718017  0.596846
stDepressionExercise_log 0.739092  0.388764  0.820214  0.797880
```

5 Summary and Implications

5.1 Interpretation

Each of the coefficients in Figure 20 can be interpreted as the change to the log odds of the presence of the given stage of heart disease for a unit increase to the explanatory variable. Raising e^{β}, where β is the given coefficient, permits calculation of the magnitude of change to the odds of a particular stage of heart disease for a one-unit increase to the explanatory variable.

For example, in the stage four model, the coefficient associated with ST segment depression during exercise is 0.8278. Since $e^{0.8278} = 2.288$, the odds that a patient with this feature has stage four heart disease are 2.288 times greater than a patient without this feature. Similarly, the coefficient for age in stage three is 0.0234. Since $e^{0.0234} = 1.024$, an increase of one year to age is associated with a 2.4% increase to the odds of a patient having stage three heart disease. This process can be generalized to every coefficient within every stage.

5.1.1 Relation to Reasearch Question

These findings are summarized in Figure 1. This figure shows in green which features contribute significantly to each stage and provide annotations suggesting the extent (fold change to the odds for a unit increase).

In differentiating the absence of heart disease (stage zero) from the presence of heart disease, age, gender, maximum heart rate during exercise, angina during exercise, ST segment depression during exercise, coronary artery fluoroscopy results, presence of atypical angina, presence of nonanginal chest pain, presence of typical angina, presence of an upward-sloping ST segment, and the presence of fixed defects upon thallium scintigraphy had statistically significant impacts. This model performed with 78.3% accuracy on unseen data.

For differentiating stage one heart disease from other stages, age, gender, angina during exercise, presence of atypical angina, and the presence of nonanginal chest pain had significant coefficients. This model stage had an accuracy of 70.7%.

For identifying stage two heart disease from other stages, an adjustment constant, age, gender, presence of atypical angina, presence of nonanginal chest pain, presence of fixed defects upon thallium scintigraphy, and the presence of reversible defects upon thallium scintigraphy had significant effects. This model stage had an accuracy of 85.3%.

Stage three heart disease could be differentiated from other stages with age, maximum heart rate during exercise, coronary artery fluoroscopy results, presence of atypical angina, presence of ST-T wave abnormalities upon ECG, presence of a downward-sloping ST segment, and the presence of fixed defects upon thallium scintigraphy. This model stage had an 88.6% accuracy.

Stage four heart disease could be identified using an adjustment constant, ST segment depression during exercise, coronary artery fluoroscopy results, and the presence of left ventricular hypertrophy signs upon ECG. This model stage had a 95.7% accuracy.

5.2 Limitation

The combined model had an overall accuracy on unseen data of 58.7%. Each individual model performed with a greater accuracy than this, but the error compounds as it propagates across all models (see Figure 18 for the accuracy of each model stage). Typically, the second highest-valued probability was correct for cases where the combined model was incorrect (in other words, the second-best guess was correct). Some of the assumptions of logistic regression are weakly violated by some model stages (there are many influential values and the Studentized Pearson residuals are not normally distributed). This can lead to model bias or poor performance, which is why it is important to perform testing on unseen data.

Figure 26
Accuracy values for the training and testing set of the combined model.

```
In [30]: resultsTest = testStages(df, models)
Combined model train accuracy: 0.5831062670299727
Combined model test accuracy: 0.5869565217391305
```

The proximity of the accuracy scores on the training and testing sets suggest that the model does not appear to be overfit to these data and thus, model bias towards the training data is negligible. However, given the assumption violations, it is still possible that this model could generalize poorly, so it should only be deployed alongside performance monitoring. Since the training data is also from 1989, it is possible that people or test procedures could have changed in the intervening years in a way not in keeping with the training environment.

However, limitations aside, this model still effectively helps answer the research question about the extent to which different clinical features predict heart disease stages (as shown in Figure 1).

5.3 Future Directions

Future studies of this dataset could explore other logistic regression techniques, such as a one-versus-all system as opposed to a one-versus-rest system. Instead

of training a model on differentiating between belonging to or not belonging to a class, it could be useful to train on "belongs to this class," "is less serious than this class," and "is more serious than this class." Alternatively, a web of models where every pairwise comparison is made could also be developed. A OVR model could also be augmented by a second discriminator model in cases where the two (or more) best guesses have similar magnitude. For example, a stage two versus stage three model could be trained to provide a prediction if this OVR model suggests stage two and stage three disease are both relatively probable, with only a slight preference for one over the other.

Insight, and potentially superior accuracy, may be obtained by also using other classification methods (such as a classification tree, random forest, various boosting algorithms, or an artificial neural network). In an analysis aimed to provide the maximum accuracy score, these classification methods could be considered alongside logistic regression models to determine an ideal classification algorithm and hyperparameters. However, in the context of this research question, these schemes are not necessary as this OVR logistic regression model is able to identify the features that correlate significantly with the presence or absence of any given class.

References

Bilogur, A. (2023, February 26). *missingno* (Version 0.5.2) [Python 3.13 package]. Retrieved October 25, 2025, from https://github.com/ResidentMario/missingno

Bobbit, Z. (2020, October 13). *The 6 Assumptions of Logistic Regression* [Statology]. Retrieved October 29, 2025, from https://www.statology.org/assumptions-of-logistic-regression/

Bryant, C. (2024). *Logistic Regression* [Python for data science]. Retrieved October 29, 2025, from https://www.pythonfordatascience.org/logistic-regression-python/

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., & Varoquaux, G. (2025, September 9). *scikit-learn* (Version 1.7.2) [Python 3.13 package]. Retrieved October 25, 2025, from https://scikit-learn.org/stable/index.html

Center for Disease Control and Prevention. (2024, May 15). *About Heart Disease*. Retrieved October 28, 2025, from https://www.cdc.gov/heart-disease/about/index.html

Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J.-J., Sandhu, S., Guppy, K. H., Lee, S., & Froelicher, V. (1989). International Application of a New Probability Algorithm for the Diagnosis of Coronary Artery Disease. *American Journal of Cardiology*, *65*, 304–310. https://doi.org/10.1016/0002-9149(89)90524-9

Frost, J. (2025). *Logistic Regression Overview with Example* [Statistics by jim]. Retrieved October 31, 2025, from https://statisticsbyjim.com/regression/logistic-regression/

Hunter, J., Dale, D., Firing, E., Droettboom, M., & the Matplotlib Development Team. (2025, September 29). *Matplotlib* (Version 3.10.6) [Python 3.13 package]. Retrieved October 25, 2025, from https://matplotlib.org/stable/

Janosi, A., Steinbrunn, W., Pfisterer, M., & Detrano, R. (1989). UCI Heart Disease Dataset. https://doi.org/10.24432/C52P4X

Karabulut, E. M., & İbrikçi, T. (2012). Effective diagnosis of coronary artery disease using the rotation forest ensemble method. *Journal of Medical Systems*, *36*(5), 3011–3018. https://doi.org/10.1007/s10916-011-9778-y

NumFOCUS, Inc. (2025, September 29). *pandas* (Version 2.3.3) [Python 3.13 package]. Retrieved October 25, 2025, from https://pandas.pydata.org/

Perktold, J., Seabold, S., Taylor, J., & Statsmodels Developers. (2025, July 7). *statsmodels* (Version 0.14.5) [Python 3.13 package]. Retrieved October 25, 2025, from https://www.statsmodels.org/stable/install.html

Python Software Foundation. (2025, October 7). *Python* (3.13.8). Retrieved October 25, 2025, from https://www.python.org/

Sanchhaya Education Private Limited. (2025, July 23). *Softmax Activation Function in Neural Networks* [Geeksforgeeks]. Retrieved October 31, 2025, from https://www.geeksforgeeks.org/deep-learning/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/

Shumeli, G., Bruce, P. C., Gedeck, P., & Patel, N. R. (2019). *Data Mining for Business Analytics: Concepts, Techniques, and Applications in Python*. Wiley Global Research (STMS). Retrieved March 29, 2025, from https://www.perlego.com/book/1149054/data-mining-for-business-analytics-concepts-techniques-and-applications-in-python-pdf

Tilevik, A. (2024, May 5). *Multinomial logistic regression: softmax regression explained* [Tilestats via youtube]. Retrieved October 28, 2025, from

https://www.youtube-nocookie.com/embed/KbK_Nb9OS70

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2025, September 11). *Scipy* (Version 1.16.2) [Python 3.13 package]. Retrieved October 25, 2025, from https://scipy.org/

Waskom, M. L. (2024, January 25). *Seaborn: Statistical data visualization* (Version 0.13.2) [Python 3.13 package]. Retrieved October 25, 2025, from https://seaborn.pydata.org/