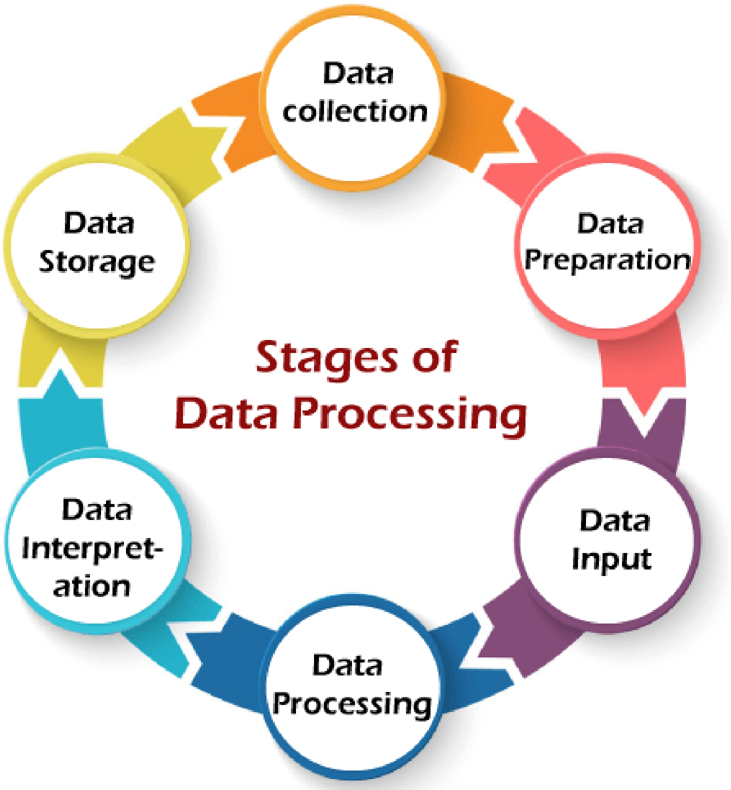
Data Overview

Mari Leonard, Gabrielle Berasi, Tyler Fusco

Steps of Data Processing

1. Data Collection
2. Cleaning
3. Input
4. Processing
5. Output and Interpretation
6. Storage



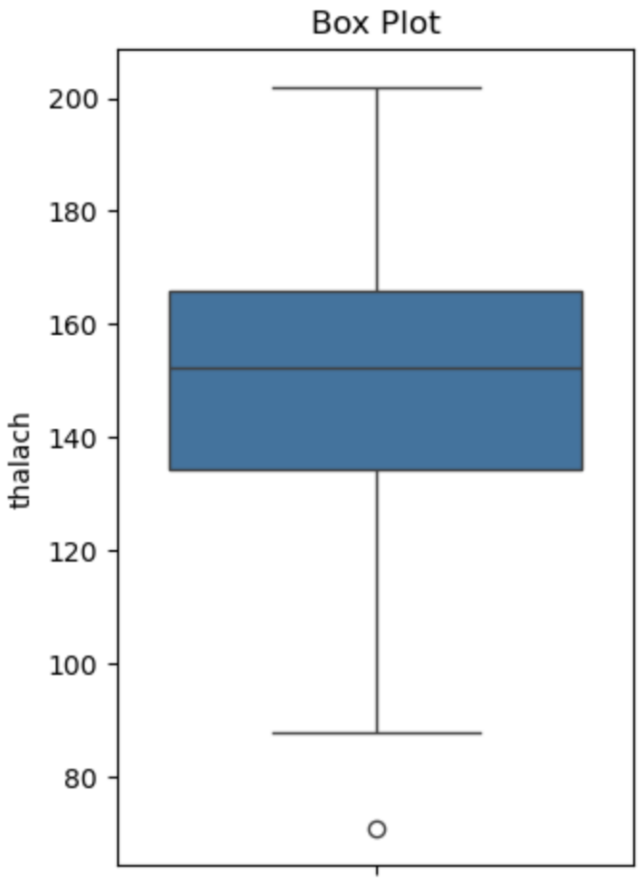
Data Cleaning

- Deal with empty cells
    - Told to fill empty cells with 0.

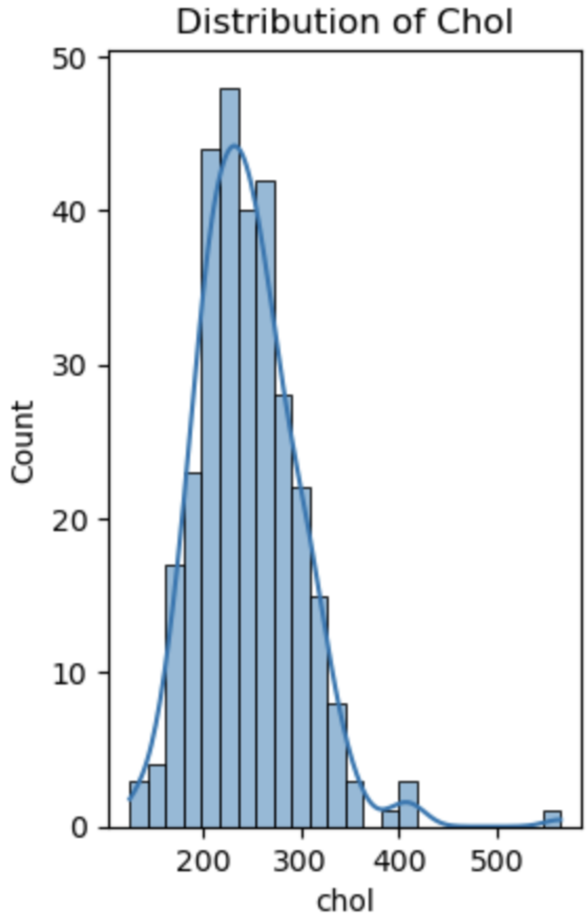
- Convert Variables to proper types using dummy variables
    - For Knn, converted Sex variable to boolean True/False instead of Male/Female

- Creating training and testing data sets
    - Remove the target variable from training data set

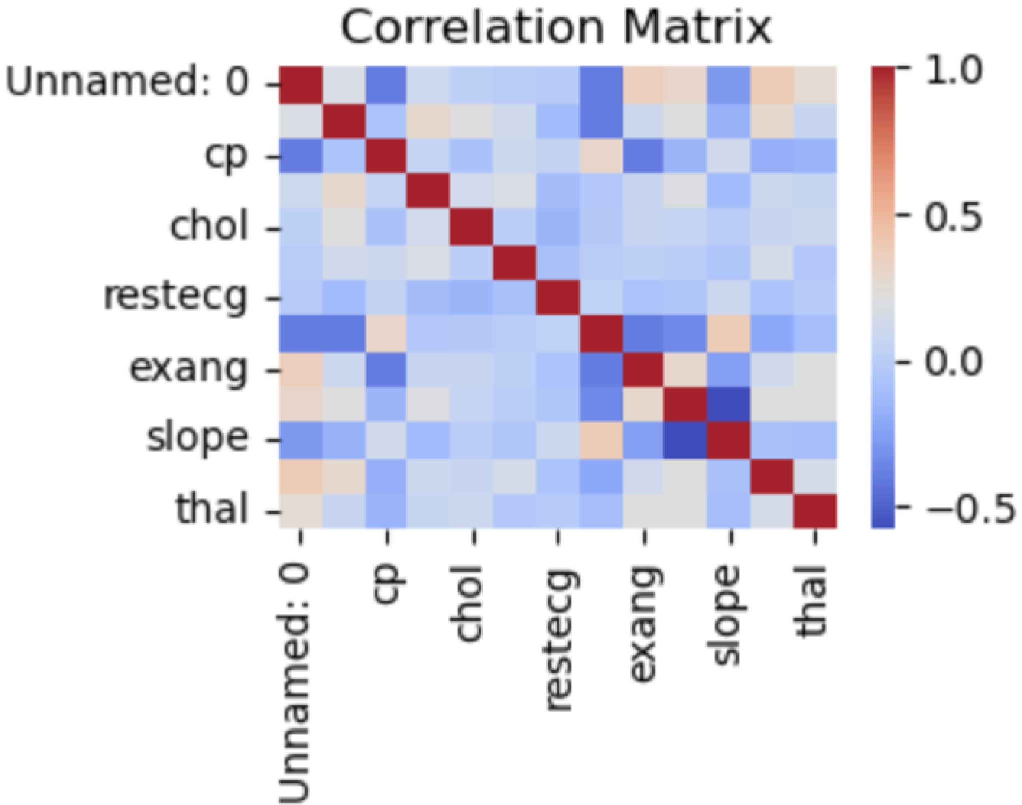
Exploratory Data Analysis



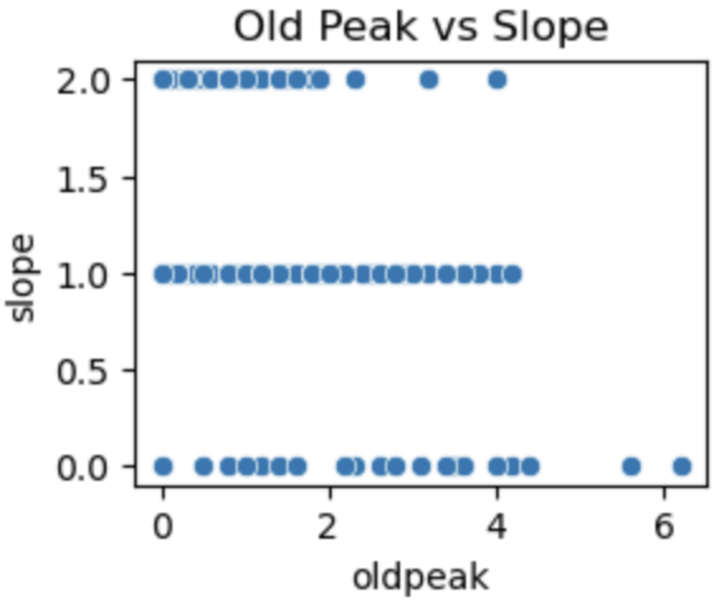

```
Text(0.5, 1.0, 'Distribution of Chol')
```



Exploratory Data Analysis




```
Text(0.5, 1.0, 'Old Peak vs Slope')
```



Input for Logistic Regression Model One

Settings Used:

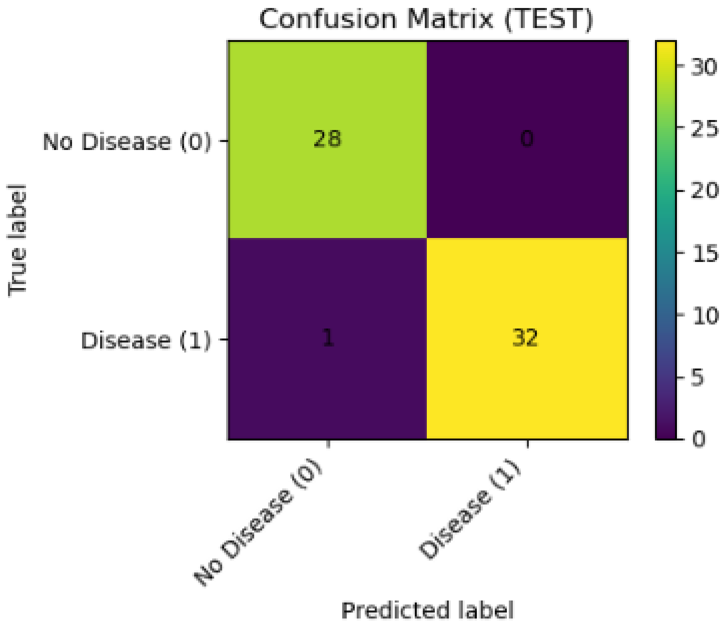
- Solver: `saga` (handles Elastic Net well)
- Max iterations: 2000 (lets the model fully converge)
- Balanced class weights (so both outcomes matter equally)

Tuning the Model:
We tested different levels of regularization:

- **C values**: 0.01 → 1.0
- **L1 ratio**: 0.1 → 0.7
- We used **5-fold cross-validation** to see which combo gave the best **F1 score** (a balance of precision and recall)**.**
- Then we retrained the model using the best settings.

Choosing the Cutoff:
Tested different thresholds on the **validation set** to find where accuracy was highest

Interpretation of Logistic regression Model One



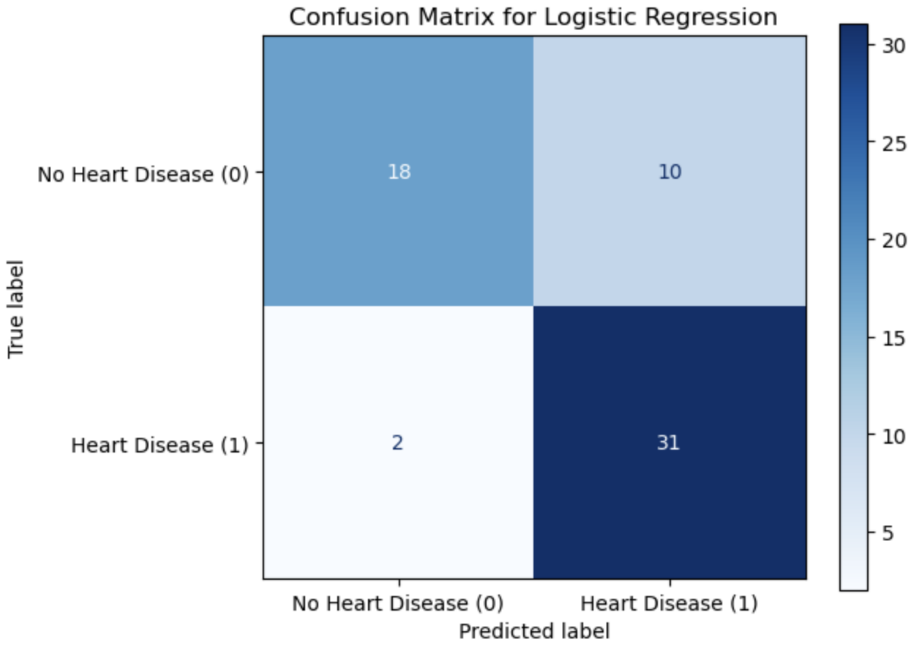

```
TEST metrics -> acc=0.984  prec=1.000  rec=0.970  f1=0.985  auc=0.999
              precision    recall  f1-score   support

           0      0.966     1.000     0.982        28
           1      1.000     0.970     0.985        33

    accuracy                          0.984        61
   macro avg      0.983     0.985     0.984        61
weighted avg      0.984     0.984     0.984        61
```

Input for Logistic Regression Two

- Data preprocessing steps
  - Handling NA values (putting them as 0)
  - Splitting the data
  - Standardization
- Training the logistic regression

```python
# Data Cleaning and Preparation
# Convert the categorical target ('yes'/'no') to binary (1/0)
df['target'] = df['target'].map({'yes': 1, 'no': 0})

# Convert the categorical sex ('male'/'female') to binary (1/0)
df['sex'] = df['sex'].map({'male': 1, 'female': 0})

# Replace any non-standard missing values (like empty strings) with NaN
df = df.replace('', np.nan)

# Convert all columns to numeric, coercing any non-convertible values (if any) to NaN
for col in df.columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Separate features (X) and target (y)
X = df.drop('target', axis=1)
y = df['target']

# Handle Missing Values (Imputation)
imputer = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value=0)
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.2, random_state=42, stratify=y)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the Logistic Regression Classifier
logreg_classifier = LogisticRegression(random_state=42)
logreg_classifier.fit(X_train_scaled, y_train)
```

Logistic Regression Model 2 Confusion Matrix



Classification Report for Logistic Regression:

	precision	recall	f1-score	support
No Heart Disease (0)	0.90	0.64	0.75	28
Heart Disease (1)	0.76	0.94	0.84	33
accuracy			0.80	61
macro avg	0.83	0.79	0.79	61
weighted avg	0.82	0.80	0.80	61

Accuracy: 0.8033
Recall Score: 0.9394
F1 Score: 0.8378

Input for Knn Model 1

- Perform Preprocessing:
  - Fill missing values with 0
  - Create flag variables

```
unfilled_columns = ["trestbps", "chol", "thalach"]

for column_name in unfilled_columns:
    df[column_name] = df[column_name].fillna(0)

df["sex"] = df["sex"] == "female"
df["target"] = df["target"] == "yes"

df.info()
```

- Split the data into training and testing dataframes

```
from sklearn.model_selection import train_test_split
#train = df.iloc[:212]
#data = df.drop("target", axis = 1)
#label = df["target"]
#test = df.iloc[212:]

#train.info()
#test.info()

y = df["target"]
x = df.drop("target", axis = 1)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3)

#print(x_test)
#print(y_test)
```

Interpretation for Knn Model 1

- 70% training data, 30% testing data
  - 212 training entries and 91 testing entries
- Found best results with k = 3 nearest neighbors
- 96.7% accuracy
- Very accurate predictions

```
import sklearn as sk
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(x_train, y_train)
predicted = knn.predict(x_test)
acc = knn.score(x_test, y_test)

print(acc)
print(predicted)
print(len(predicted))
```

```
TEST (KNN) metrics:
Accuracy : 0.967
Precision: 0.968
Recall   : 0.966
F1-score : 0.967
```

	precision	recall	f1-score	support
False	0.98	0.95	0.96	43
True	0.96	0.98	0.97	48
accuracy			0.97	91
macro avg	0.97	0.97	0.97	91
weighted avg	0.97	0.97	0.97	91

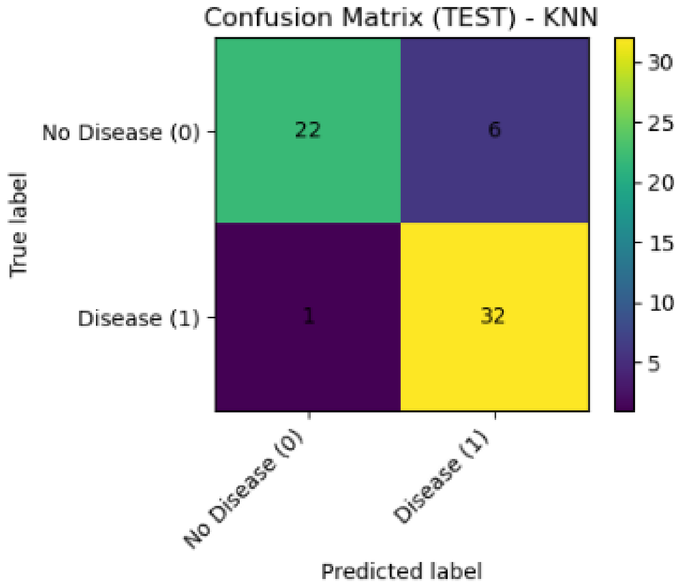
Input for KNN Model 2

Perform Preprocessing:

```python
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
predicted = knn.predict(X_test)
acc = knn.score(X_test, y_test)

print("Accuracy:", acc)
print("Predictions:", predicted)
```

- One-hot encode categoricals
    - min_frequency=0.01 to merge ultra-rare levels
- Drop constant/near-constant numerics
    - VarianceThreshold.
- Stratified split to keep class balance
- Leakage guard: dropped any feature perfectly/near-perfectly tied to the target
- 5-fold Stratified CV to pick hyperparameters
- Threshold tuning on validation to maximize accuracy
- Split data: 70% training, 30% testing
- Evaluated different values of K

Interpretation of KNN Model 2




```
TEST (KNN) metrics:
  Accuracy : 0.885
  Precision: 0.842
  Recall   : 0.970
  F1-score : 0.901

Classification report:
              precision    recall  f1-score   support

           0      0.957     0.786     0.863        28
           1      0.842     0.970     0.901        33

    accuracy                          0.885        61
   macro avg      0.899     0.878     0.882        61
weighted avg      0.895     0.885     0.884        61
```

Input for Random Forest Model One

- Same

- Tried multiple Random Forests

- Different number of estimators

- Different accuracies each time

```
model = RandomForestClassifier(n_estimators=100, random_state=123)
model.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=123)
```

```
model = RandomForestClassifier(n_estimators=3, random_state=123)
model.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=3, random_state=123)
```

Interpretation for Random Forest Model One

- Using n_estimators = 3 gave an accuracy of 98.9%

- Using n_estimators >= 6 gave an accuracy of 1.0 or 100%

- 100% accuracy indicates model overfitting

- Needs to be adjusted for more reliable accuracy output

Input for Random Forest Model Two

```
new_model = RandomForestClassifier(n_estimators=5, random_state=42)
new_model.fit(X_train_scaled, y_train)

y_pred_new = new_model.predict(X_test_scaled)

new_accuracy = accuracy_score(y_test, y_pred_new)

print(f"New Accuracy with 5 Estimators: {new_accuracy:.4f}")
```

- Same data preprocessing steps used
- Tested 3 different estimator values
- Significantly different accuracies

Interpretation For Random Forest Model Two

- n_estimators = 5 gave accuracy of 75.41%
- n_estimators = 10 gave me accuracy of 80.33%
- n_estimators = 100 gave me accuracy of 85.25%

Comparison of All Models

Logistic Regression

Model 1:

Accuracy: 98.4%

F1: 98.5%

Recall: 97.0%

Model 2:

Accuracy: 80.3%

F1: 83.8%

Recall: 93.9%

KNN Model

Model 1:

Accuracy: 96.7%

F1: 96.7%

Recall: 96.6%

Model 2:

Accuracy: 88.5%

F1: 90.1%

Recall: 97.0%

Random Forest

Model 1:

Accuracy: 100%

F1: 100%

Recall: 100%

Model 2:

Accuracy: 85.3%

F1: 78.3%

Recall: 81.8%

In Conclusion

Models Tested

- KNN
- Logistic Regression
- Random Forest

Storage

Data and Models are both stored locally on our devices and personal Github repositories as well as shared electronically with each other through a shared GitHub repository.

Questions?