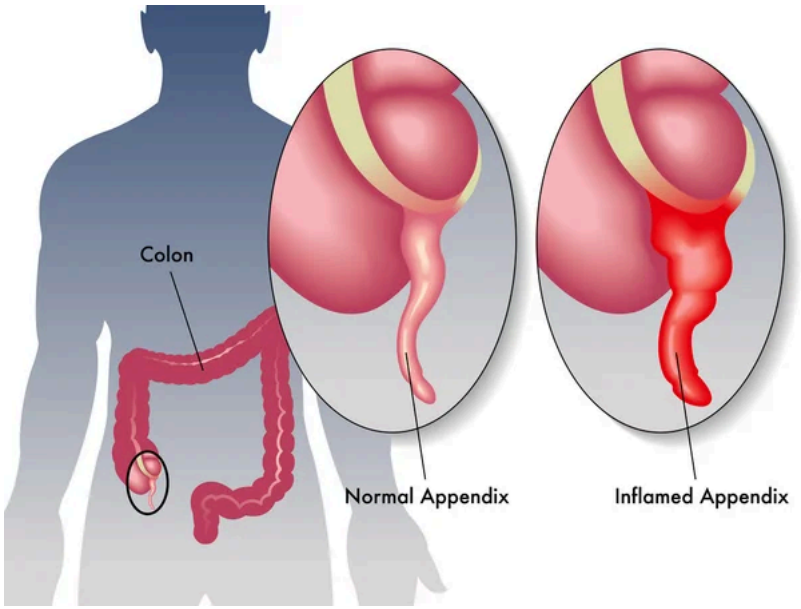
Machine learning - Coursework 2

Predicting Pediatric Appendicitis Diagnosis and Severity Using Clinical Data

Author: *Eoin Houstoun*
Affiliation: Department of Computing, Goldsmiths, University of London
Email: ehous001@gold.ac.uk



Interpretable and Intervenable Ultrasonography-based Machine Learning Models for Pediatric Appendicitis
Ričards Marcinkevičs & et al, Published in Medical Image Analysis 2023
Introductory Paper

Objective

This project aims to predict:

1. **Diagnosis**: Whether a pediatric patient has appendicitis
2. **Severity**: Whether the appendicitis is uncomplicated or complicated

using only **structured clinical, laboratory, and ultrasound-derived tabular features**, excluding raw ultrasound images.

> Uncomplicated Appendicitis includes subacute/catarrhal inflammation and fibrosis and complicated Appendicitis includes phlegmonous, gangrenous, perforated, or abscessed forms.

Dataset

The project uses the Regensburg Pediatric Appendicitis Dataset, which contains data from **782 pediatric patients** aged 0-18.

Predictor variables:

- Demographics (age, sex, BMI)
- Clinical scores (e.g. Alvarado, Pediatric Appendicitis Score)
- Physical signs and symptoms (e.g. rebound tenderness, nausea)
- Lab values (e.g. WBC, CRP, hemoglobin)
- Sonographic findings (e.g. appendix diameter, free fluids)

Outcome variables:

- `Diagnosis` : appendicitis vs. no appendicitis
- `Severity` : uncomplicated vs. complicated

The data used in this project comes from a retrospective study conducted at the Children's Hospital St. Hedwig in Regensburg, Germany (Interpretable and Intervenable Ultrasonography-based Machine Learning Models for Pediatric Appendicitis, Ričards Marcinkevičs & et al, Published in Medical Image Analysis 2023). It includes pediatric patients who were admitted with abdominal pain. For most patients, multiple abdominal ultrasound images were captured, focusing on areas such as the appendix, intestines, lymph nodes, and reproductive organs.

In addition to the ultrasound images, the dataset contains structured clinical information, including laboratory test results, physical examination findings, and clinical scores like the Alvarado and Pediatric Appendicitis Score.

The study received ethical approval from the University of Regensburg and was conducted in accordance with relevant medical research guidelines.

Libraries

```
# Numerical & Data Manipulation
import numpy as np
import pandas as pd
import scipy.stats as st
from scipy.stats import t

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.cm import ScalarMappable
from matplotlib.colors import Normalize
import matplotlib.ticker as mtick
from tabulate import tabulate
from tqdm import trange


# Scikit-learn - Preprocessing & Pipelines
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Scikit-learn - Imputation
from sklearn.impute import KNNImputer

# Scikit-learn - Model Selection
from sklearn.model_selection import (
    train_test_split, GridSearchCV, StratifiedKFold,
    ParameterGrid, cross_val_score
)

# Scikit-learn - Classification Models
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

# Scikit-learn - Regression Models
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import ElasticNet, LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor

# Scikit-learn - Evaluation Metrics
from sklearn.metrics import (
    accuracy_score, roc_auc_score, f1_score, precision_score, recall_score,
    confusion_matrix, precision_recall_curve, roc_curve,
    mean_squared_error, mean_absolute_error,root_mean_squared_error, r2_score,
    make_scorer, classification_report
)
from sklearn.calibration import calibration_curve

# XGBoost
from xgboost import XGBClassifier, XGBRegressor

# Neural Network
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import KFold
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score, roc_auc_score, roc_curve
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import KNNImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from copy import deepcopy
```

```
# General Utilities
import os
import re
import warnings
from collections import defaultdict, Counter

# Google Drive (Colab Only)
from google.colab import drive
drive.mount('/content/drive')

# Paths for saving results
save_path_c_f = "/content/drive/MyDrive/Colab Notebooks/ML/CW2/RESULTS/CLASSIFICATION/DIAG"
save_path_c_m = "/content/drive/MyDrive/Colab Notebooks/ML/CW2/RESULTS/CLASSIFICATION/SEVERITY"


# Suppress Warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

1. DATASET EXPLORATION

```
!pip install ucimlrepo

from ucimlrepo import fetch_ucirepo

# fetch dataset
regensburg_pediatric_appendicitis = fetch_ucirepo(id=938)

# data (as pandas dataframes)
X = regensburg_pediatric_appendicitis.data.features
y = regensburg_pediatric_appendicitis.data.targets

# metadata
print(regensburg_pediatric_appendicitis.metadata)

# variable information
regensburg_pediatric_appendicitis.variables
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.11/dist-packages (from ucimlrepo) (2025.1.
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrep
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->panda
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
{'uci_id': 938, 'name': 'Regensburg Pediatric Appendicitis', 'repository_url': 'https://archive.ics.uci.edu/dataset/93
```

	name	role	type	demographic	description	units	missing_values
0	Age	Feature	Continuous	Age	Obtained from the date of birth	years	yes
1	BMI	Feature	Continuous	None	Measures body fat; patient's weight divided by...	None	yes
2	Sex	Feature	Categorical	Sex	Registered gender	None	yes
3	Height	Feature	Continuous	None	Patient's height	None	yes
4	Weight	Feature	Integer	None	Patient's weight	None	yes
5	Length_of_Stay	Feature	Integer	None	Length of the stay in the hospital	None	yes
6	Management	Target	Categorical	None	(conservative, primary surgical, secondary sur...	None	yes
7	Severity	Target	Categorical	None	(uncomplicated, complicated) Severity of appen...	None	yes
8	Diagnosis_Presumptive	Other	Binary	None	Patient's suspected diagnosis	None	yes
9	Diagnosis	Target	Binary	None	Patient's diagnosis, histologically confirmed ...	None	yes
10	Alvarado_Score	Feature	Integer	None	Patient's score according to the scoring system	None	yes
11	Paedriatic_Appendicitis_Score	Feature	Integer	None	Patient's score according to the scoring system	None	yes
12	Appendix_on_US	Feature	Binary	None	Detectability of the vermiform appendix during...	None	yes
13	Appendix_Diameter	Feature	Integer	None	Maximal outer diameter of the appendix	None	yes
14	Migratory_Pain	Feature	Binary	None	Abdominal pain; usually starts in epigastrium ...	None	yes
15	Lower_Right_Abd_Pain	Feature	Binary	None	Right iliac fossa pain detected on palpation	None	yes
16	Contralateral_Rebound_Tenderness	Feature	Binary	None	A state in which pain of the contralateral sid...	None	yes
17	Coughing_Pain	Feature	Binary	None	Abdominal pain by forced cough	None	yes
18	Nausea	Feature	Binary	None	Feeling of sickness/ejection of contents from ...	None	yes
19	Loss_of_Appetite	Feature	Binary	None	None	None	yes
20	Body_Temperature	Feature	Continuous	None	Measured by a thermometer placed in the rectum...	None	yes
21	WBC_Count	Feature	Continuous	None	The number of leucocytes in a unit volume of b...	None	yes
22	Neutrophil_Percentage	Feature	Integer	None	Mature WBC in the granulocytic series	None	yes
23	Segmented_Neutrophils	Feature	Integer	None	Most mature neutrophilic granulocytes present ...	None	yes
24	Neutrophilia	Feature	Categorical	None	Relative neutrophilic leucocytosis, often a re...	None	yes
25	RBC_Count	Feature	Continuous	None	The number of erythrocytes in a unit volume of...	None	yes
26	Hemoglobin	Feature	Continuous	None	Hemoglobin level; a red protein in the red blo...	None	yes
27	RDW	Feature	Continuous	None	A blood test that measures the differences in ...	None	yes
28	Thrombocyte_Count	Feature	Integer	None	The number of platelets in a unit volume of blood	None	yes
29	Ketones_in_Urine	Feature	Categorical	None	Presence of ketone bodies in urine, e.g. in ca...	None	yes

30	RBC_in_Urine	Feature	Categorical	None	Blood in urine	None	yes
31	WBC_in_Urine	Feature	Categorical	None	Leucocytes in urine, e.g., in case of infection	None	yes
32	CRP	Feature	Integer	None	Protein produced by the liver, elevated in cas...	None	yes
33	Dysuria	Feature	Binary	None	Pain or other difficulty during urination	None	yes
34	Stool	Feature	Categorical	None	Characteristics of bowel movements	None	yes
35	Peritonitis	Feature	Categorical	None	Spasm of abdominal wall muscles detected on pa...	None	yes
36	Psoas_Sign	Feature	Categorical	None	Abdominal pain produced by extension of the hip	None	yes
37	Ipsilateral_Rebound_Tenderness	Feature	Categorical	None	A state in which pain of the ipsilateral side ...	None	yes
38	US_Performed	Feature	Categorical	None	If an abdominal ultrasonography was performed ...	None	yes
39	US_Number	Other	Integer	None	None	None	yes
40	Free_Fluids	Feature	Categorical	None	Free fluids inside the abdomen	None	yes
41	Appendix_Wall_Layers	Feature	Categorical	None	Distribution and characteristics of appendix l...	None	yes

1.1 Feature Exploration Functions

```
## OUTLIER DETECTION
def detect_outliers(X, extreme_threshold=5):
    """
    Detects extreme outliers in numerical features using an aggressive IQR method (3 × IQR rule).

    Args:
        X (pd.DataFrame): The dataset (features only, no target variable).
        extreme_threshold (float): The multiplier for the IQR to define extreme outliers (default is 3).

    Returns:
        pd.DataFrame: A summary of extreme outliers for each numeric column.
    """

    if not isinstance(X, pd.DataFrame):
        print("❌ Error: Input `X` must be a Pandas DataFrame.")
        return None

    numeric_cols = X.select_dtypes(include=['number'])
    summary = numeric_cols.describe()

    # Compute IQR
    Q1 = summary.loc['25%']
    Q3 = summary.loc['75%']
    IQR = Q3 - Q1

    lower_bound = Q1 - extreme_threshold * IQR
    upper_bound = Q3 + extreme_threshold * IQR

    extreme_outlier_mask = (numeric_cols < lower_bound) | (numeric_cols > upper_bound)
    extreme_outlier_counts = extreme_outlier_mask.sum()
    extreme_outlier_percent = (extreme_outlier_counts / len(X)) * 100
    extreme_outlier_summary = pd.DataFrame({
        "Outlier Count": extreme_outlier_counts,
        "Outlier Percentage (%)": extreme_outlier_percent
    }).sort_values(by="Outlier Percentage (%)", ascending=False)

    print("\\n Outlier Detection Summary:")
    print(extreme_outlier_summary)

    extreme_cols = extreme_outlier_summary[extreme_outlier_summary["Outlier Count"] > 0].index
    if len(extreme_cols) > 0:
        for col in extreme_cols:
            plt.figure(figsize=(8, 5))
            sns.histplot(X[col], bins=30, kde=True, color="red")
            plt.axvline(lower_bound[col], color='blue', linestyle="dashed", label="Lower Bound")
            plt.axvline(upper_bound[col], color='blue', linestyle="dashed", label="Upper Bound")
            plt.title(f"Histogram of {col} with Outliers", fontsize=14, fontweight="bold")
            plt.xlabel(col)
            plt.ylabel("Count")
```

```
            plt.show()

    return extreme_outlier_summary


# Class Distribution (BINARY)
def class_distribution(y, target_name="Target Variable"):
    """
    Plots and prints the class distribution for a binary classification problem.

    Args:
        y (pd.Series): The target variable (should contain 'Yes'/'No' or 0/1).
        target_name (str): The name of the target variable for plot title.
    """
    if not isinstance(y, pd.Series):
        print("Error: Input `y` must be a Pandas Series.")
        return

    unique_vals = y.unique()
    if len(unique_vals) != 2:
        print(f"Error: {target_name} is not binary. Found values: {unique_vals}")
        return

    class_counts = y.value_counts()

    plt.figure(figsize=(8, 5))
    ax = sns.countplot(x=y, palette=sns.color_palette(['cyan', 'purple']),
                       order=class_counts.index, legend=False)

    ax.set_title(f'Frequency of {target_name}', fontsize=16, fontweight='bold')
    ax.set_xlabel('Class', fontsize=14)
    ax.set_ylabel('Count', fontsize=14)

    for p in ax.patches:
        ax.annotate(f'{int(p.get_height())}',
                    (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='baseline', fontsize=15, color='black', fontweight='bold')

    yes_count, no_count = class_counts.iloc[0], class_counts.iloc[1]
    yes_label, no_label = class_counts.index[0], class_counts.index[1]

    print(f"Number of {yes_label}: {yes_count} ||| Proportion: {round(100 * (yes_count / (yes_count + no_count)), 1)}%")
    print(f"Number of {no_label}: {no_count} ||| Proportion: {round(100 * (no_count / (yes_count + no_count)), 1)}%")

    plt.show()
```

1.2 Predictors

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 782 entries, 0 to 781
Data columns (total 53 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Age                               781 non-null    float64
 1   BMI                               755 non-null    float64
 2   Sex                               780 non-null    object
 3   Height                            756 non-null    float64
 4   Weight                            779 non-null    float64
 5   Length_of_Stay                    778 non-null    float64
 6   Alvarado_Score                    730 non-null    float64
 7   Paedriatic_Appendicitis_Score     730 non-null    float64
 8   Appendix_on_US                    777 non-null    object
 9   Appendix_Diameter                 498 non-null    float64
 10  Migratory_Pain                    773 non-null    object
 11  Lower_Right_Abd_Pain              774 non-null    object
 12  Contralateral_Rebound_Tenderness  767 non-null    object
 13  Coughing_Pain                     766 non-null    object
 14  Nausea                            774 non-null    object
 15  Loss_of_Appetite                  772 non-null    object
 16  Body_Temperature                  775 non-null    float64
 17  WBC_Count                         776 non-null    float64
 18  Neutrophil_Percentage             679 non-null    float64
 19  Segmented_Neutrophils             54 non-null     float64
 20  Neutrophilia                      732 non-null    object
 21  RBC_Count                         764 non-null    float64
 22  Hemoglobin                        764 non-null    float64
 23  RDW                               756 non-null    float64
 24  Thrombocyte_Count                 764 non-null    float64
 25  Ketones_in_Urine                  582 non-null    object
 26  RBC_in_Urine                      576 non-null    object
 27  WBC_in_Urine                      583 non-null    object
```

```
28  CRP                              771 non-null    float64
29  Dysuria                          753 non-null    object
30  Stool                            765 non-null    object
31  Peritonitis                      773 non-null    object
32  Psoas_Sign                       745 non-null    object
33  Ipsilateral_Rebound_Tenderness   619 non-null    object
34  US_Performed                     778 non-null    object
35  Free_Fluids                      719 non-null    object
36  Appendix_Wall_Layers             218 non-null    object
37  Target_Sign                      138 non-null    object
38  Appendicolith                    69 non-null     object
39  Perfusion                        63 non-null     object
40  Perforation                      81 non-null     object
41  Surrounding_Tissue_Reaction      252 non-null    object
42  Appendicular_Abscess             85 non-null     object
43  Abscess_Location                 13 non-null     object
44  Pathological_Lymph_Nodes         203 non-null    object
45  Lymph_Nodes_Location             121 non-null    object
46  Bowel_Wall_Thickening            99 non-null     object
47  Conglomerate_of_Bowel_Loops      43 non-null     object
48  Ileus                            60 non-null     object
49  Coprostasis                      71 non-null     object
50  Meteorism                        140 non-null    object
51  Enteritis                        66 non-null     object
52  Gynecological_Findings           26 non-null     object
```

- Lots of missing values
- Many categorical variables

```
round(X.describe().T,3)
```

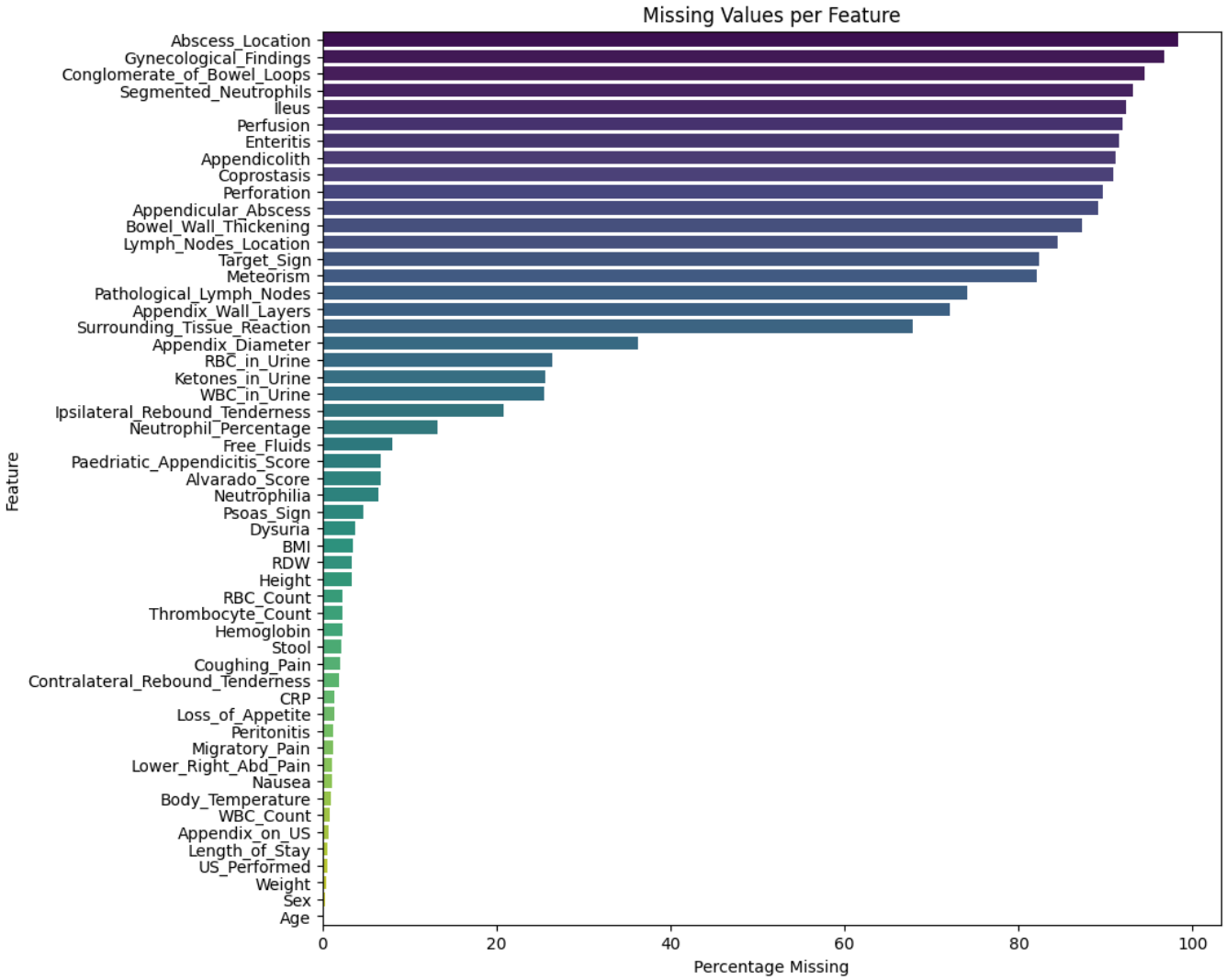
	count	mean	std	min	25%	50%	75%	max
Age	781.0	11.346	3.530	0.00	9.200	11.44	14.10	18.36
BMI	755.0	18.907	4.385	7.83	15.725	18.06	21.18	38.16
Height	756.0	148.017	19.732	53.00	137.000	149.65	163.00	192.00
Weight	779.0	43.173	17.391	3.96	29.500	41.40	54.00	103.00
Length_of_Stay	778.0	4.284	2.574	1.00	3.000	3.00	5.00	28.00
Alvarado_Score	730.0	5.922	2.156	0.00	4.000	6.00	8.00	10.00
Paedriatic_Appendicitis_Score	730.0	5.253	1.958	0.00	4.000	5.00	7.00	10.00
Appendix_Diameter	498.0	7.763	2.537	2.70	6.000	7.50	9.10	17.00
Body_Temperature	775.0	37.405	0.904	26.90	36.800	37.20	37.90	40.20
WBC_Count	776.0	12.671	5.367	2.60	8.200	12.00	16.20	37.70
Neutrophil_Percentage	679.0	71.791	14.464	27.20	61.400	75.50	83.60	97.70
Segmented_Neutrophils	54.0	64.930	15.085	32.00	54.500	64.50	77.50	91.00
RBC_Count	764.0	4.799	0.499	3.62	4.538	4.78	5.02	14.00
Hemoglobin	764.0	13.380	1.393	8.20	12.600	13.30	14.00	36.00
RDW	756.0	13.180	4.539	11.20	12.300	12.70	13.30	86.90
Thrombocyte_Count	764.0	285.253	72.494	91.00	236.000	276.00	330.00	708.00
CRP	771.0	31.387	57.434	0.00	1.000	7.00	33.00	365.00

- Median and mean align well across all numerical features except CRP
- **C-Reactive Protein (CRP)** stands out with a **mean of 31.39** and a **median of 7**, suggesting the presence of **potential outliers or extreme inflammation cases** with very high CRP values.

Summarise Missing Data

```
missing_percent = X.isnull().mean() * 100
missing_percent_sorted = missing_percent.sort_values(ascending=False)


plt.figure(figsize=(10,10))
sns.barplot(x=missing_percent_sorted[missing_percent_sorted > 0],
            y=missing_percent_sorted[missing_percent_sorted > 0].index,
            palette='viridis')
plt.xlabel('Percentage Missing')
plt.ylabel('Feature')
plt.title('Missing Values per Feature')
```

```
plt.show()
```




```
cols_to_drop = missing_percent[missing_percent > 50].index
X = X.drop(columns=cols_to_drop)
```

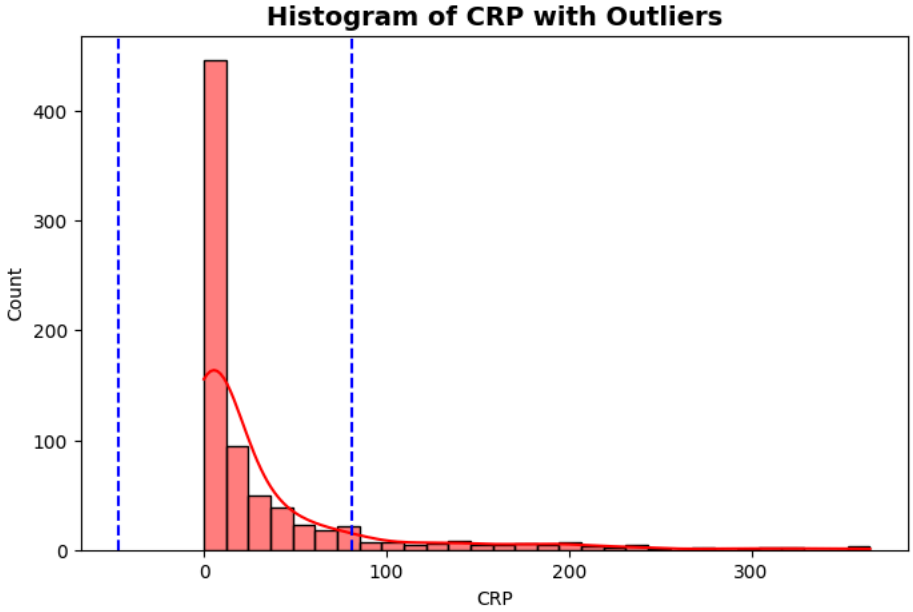
Variables with more than 50% missing values were removed, as such a high proportion of missing data could compromise model performance.

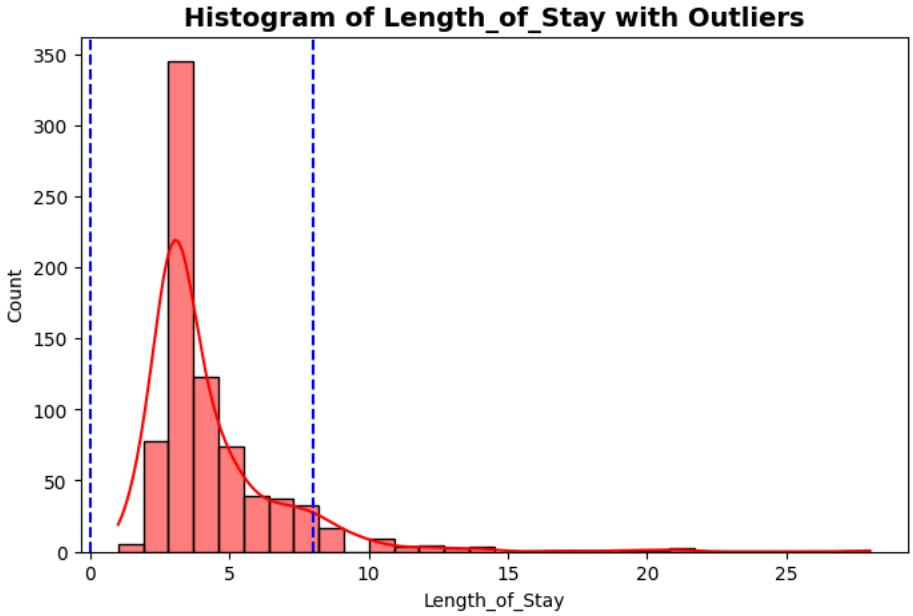
```
numeric_cols = X.select_dtypes(include=['float64']).columns
categorical_cols = X.select_dtypes(include=['object']).columns
```

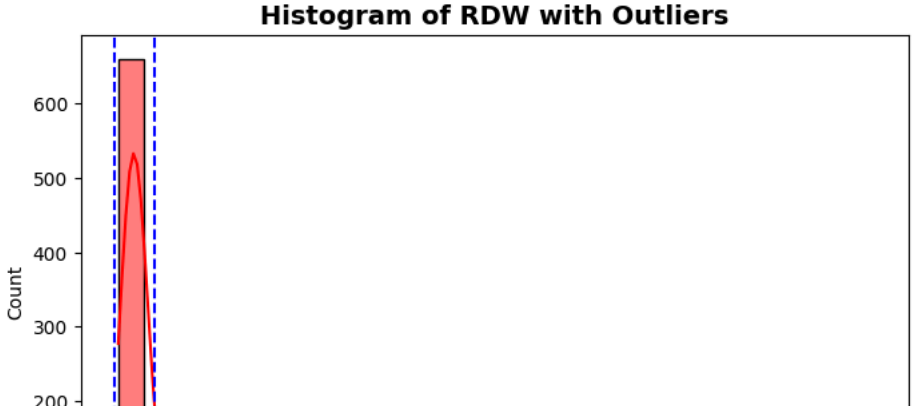
```
outliers = detect_outliers(X, 1.5)  # Detect outliers in features
```

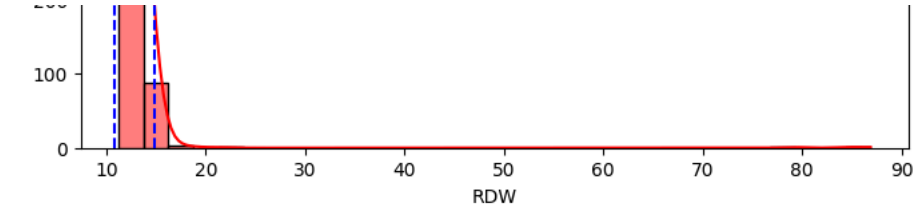
\n Outlier Detection Summary:

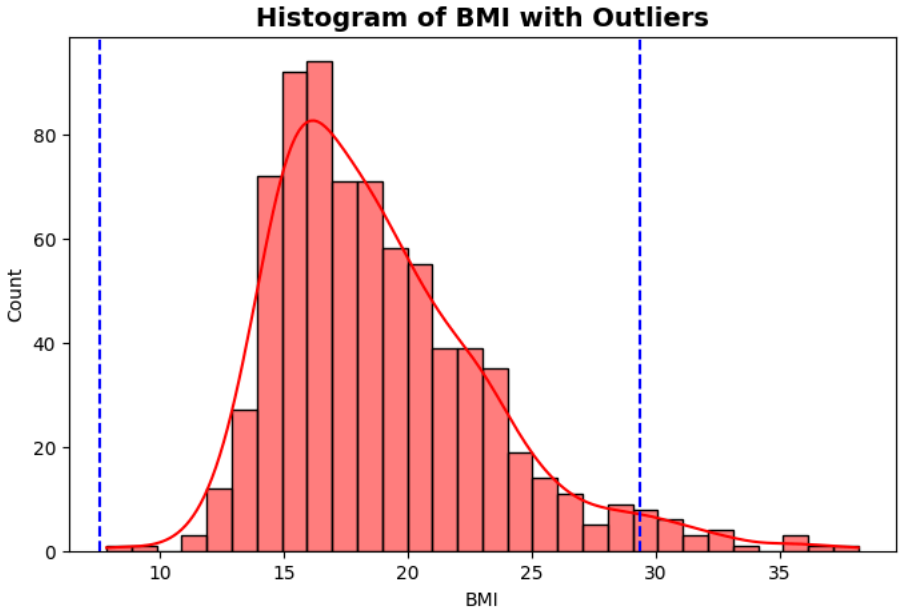
	Outlier Count	Outlier Percentage (%)
CRP	85	10.869565
Length_of_Stay	44	5.626598
RDW	23	2.941176
BMI	23	2.941176
Hemoglobin	16	2.046036
RBC_Count	14	1.790281
Height	12	1.534527
Body_Temperature	12	1.534527
Appendix_Diameter	10	1.278772
Thrombocyte_Count	9	1.150895
Weight	8	1.023018
WBC_Count	6	0.767263
Age	5	0.639386
Neutrophil_Percentage	1	0.127877
Paedriatic_Appendicitis_Score	0	0.000000
Alvarado_Score	0	0.000000

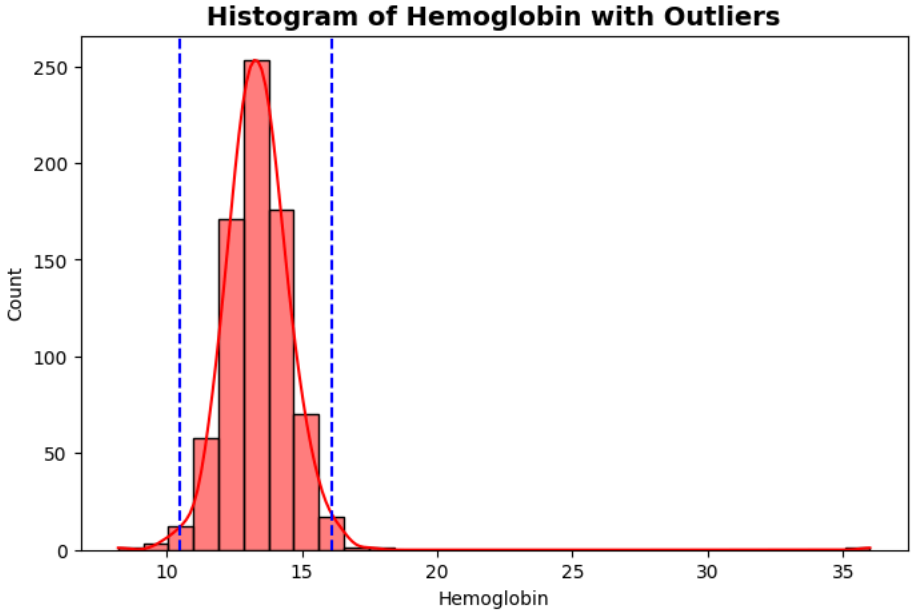


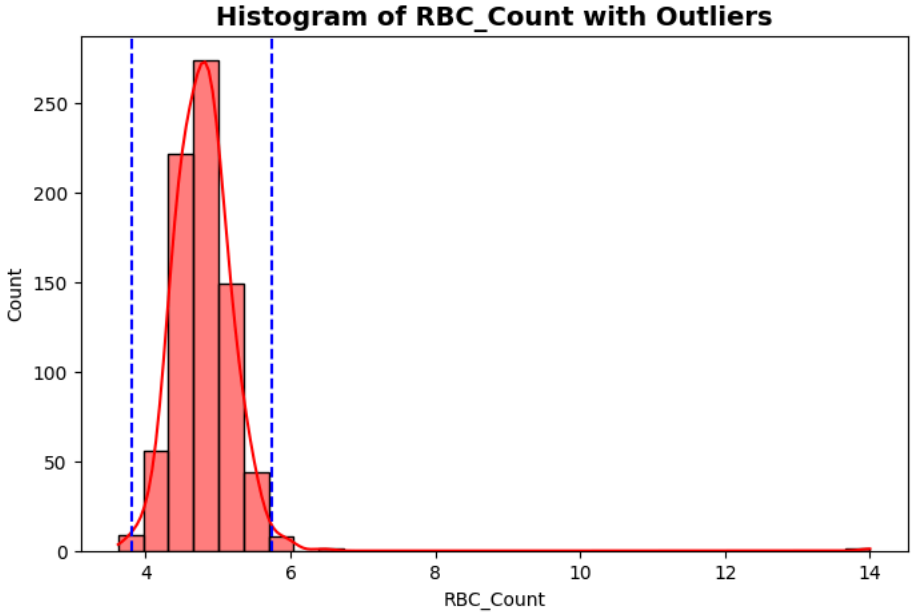


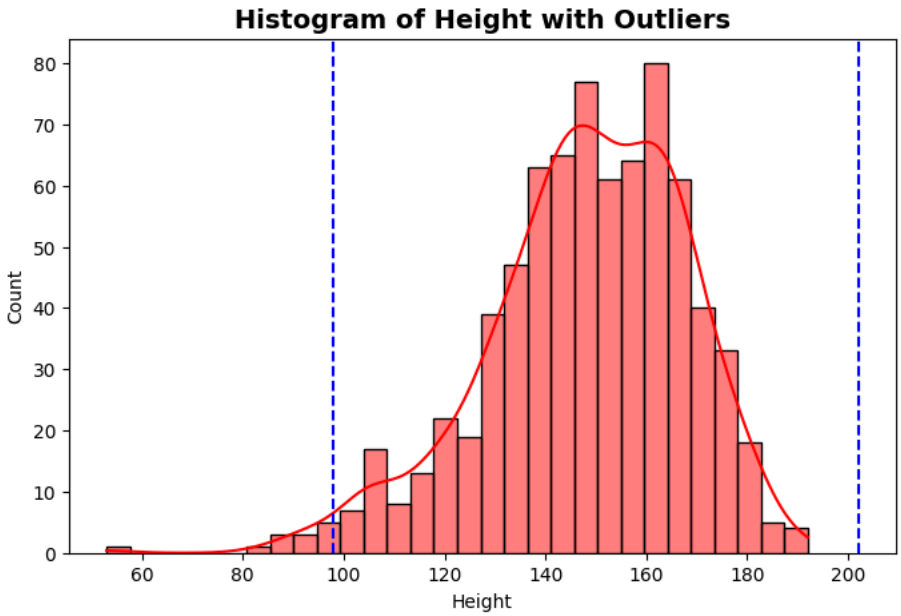


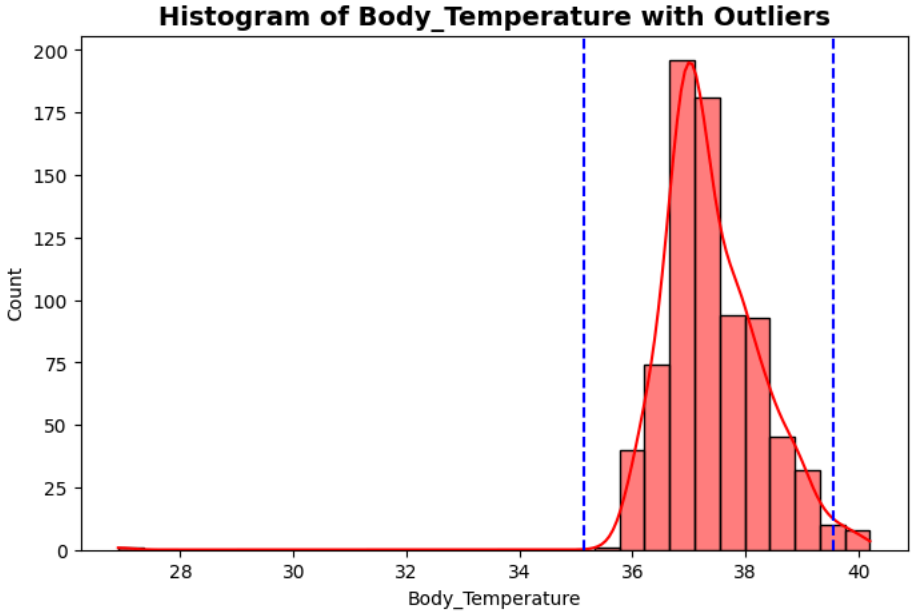

100
0
10
20
30
40
50
60
70
80
90
RDW

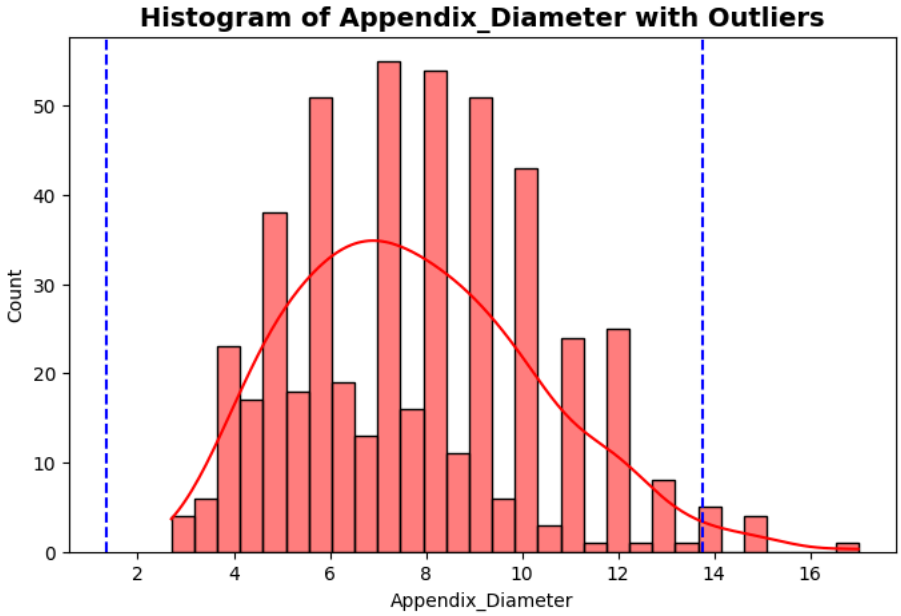

Histogram of BMI with Outliers
Count
80
60
40
20
0
10
15
20
25
30
35
BMI

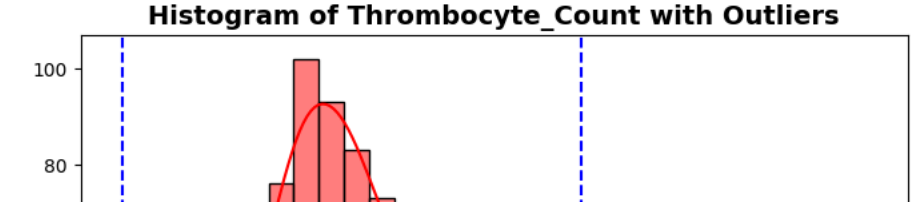

Histogram of Hemoglobin with Outliers
Count
250
200
150
100
50
0
10
15
20
25
30
35
Hemoglobin

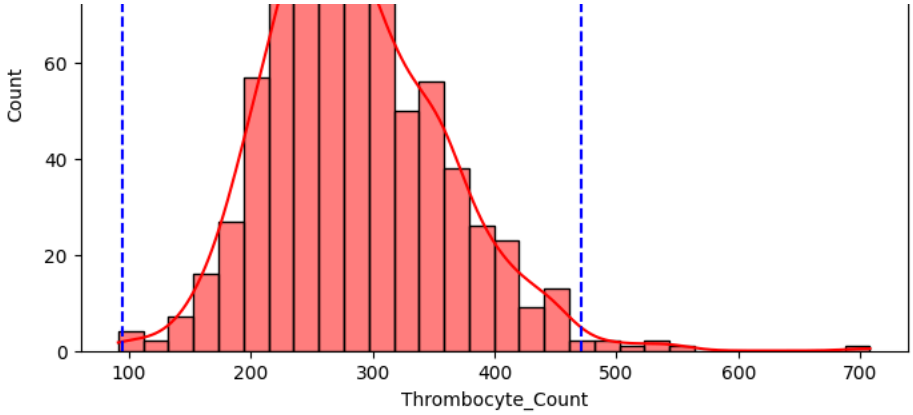

Histogram of RBC_Count with Outliers
Count
250
200
150
100
50
0
4
6
8
10
12
14
RBC_Count

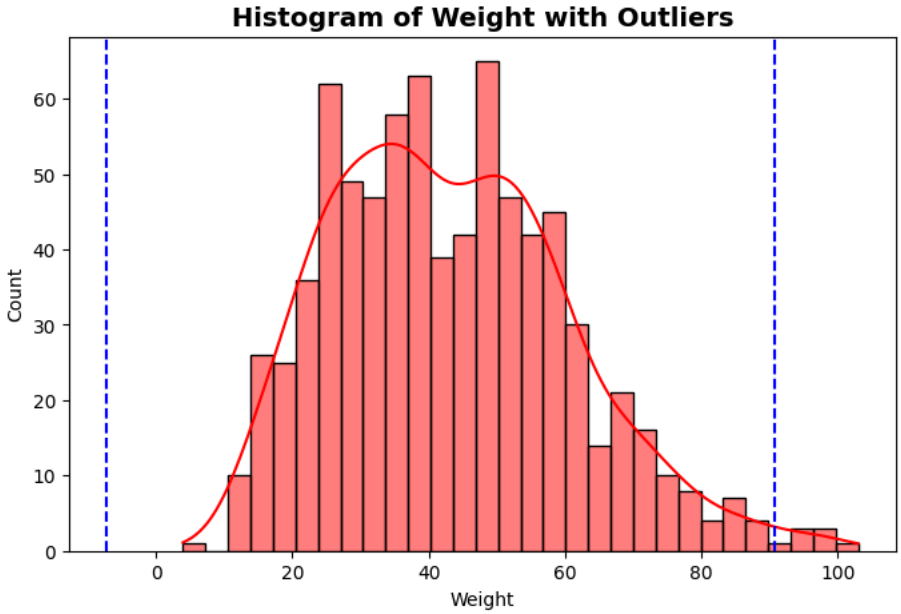

Histogram of Height with Outliers
Count
Height
0
10
20
30
40
50
60
70
80
60
80
100
120
140
160
180
200

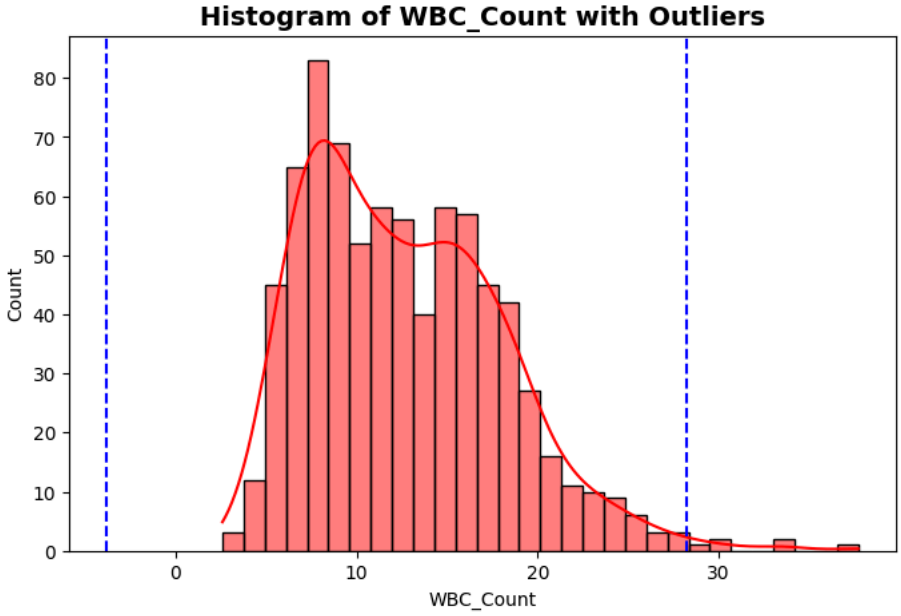

Histogram of Body_Temperature with Outliers
Count
Body_Temperature
0
25
50
75
100
125
150
175
200
28
30
32
34
36
38
40

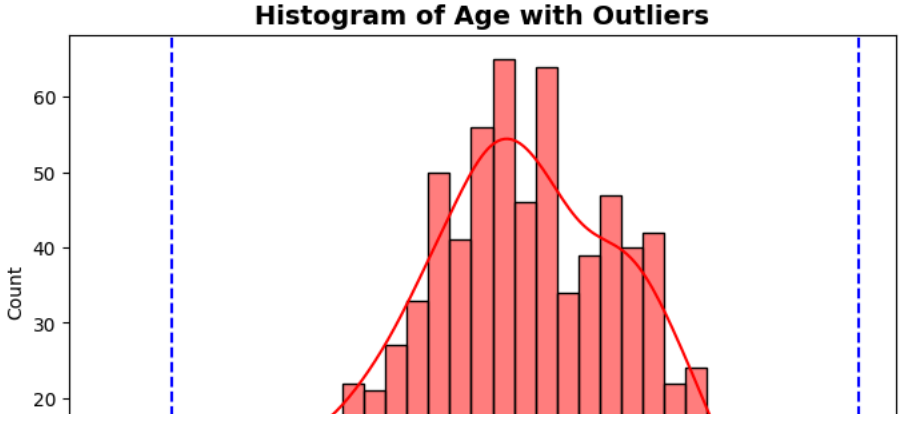

Histogram of Appendix_Diameter with Outliers
Count
Appendix_Diameter
0
10
20
30
40
50
2
4
6
8
10
12
14
16

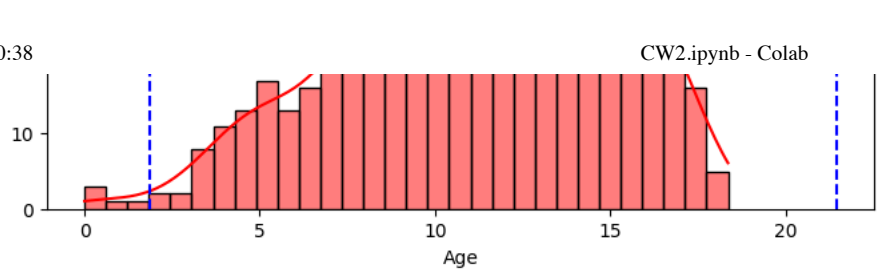

Histogram of Thrombocyte_Count with Outliers
80
100

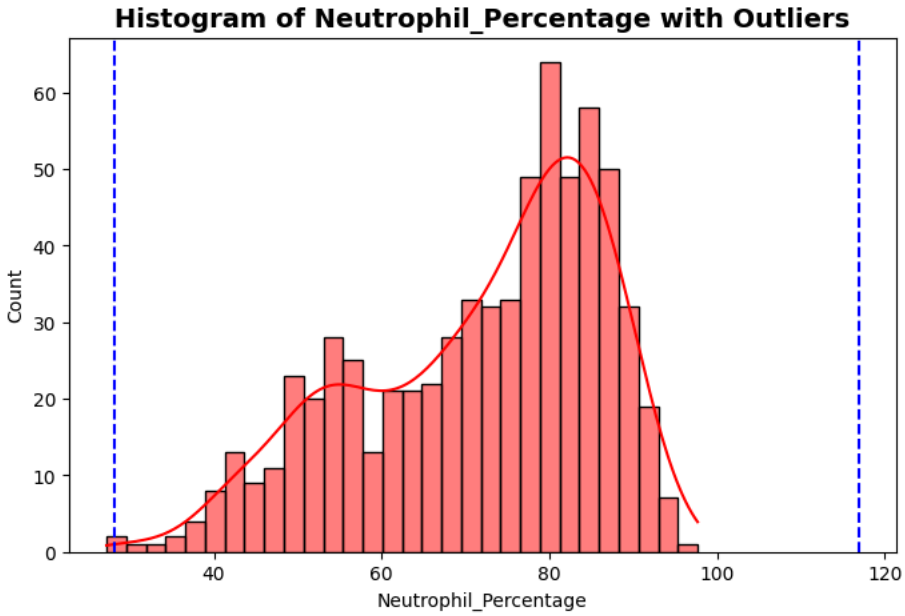

Count
Thrombocyte_Count


Histogram of Weight with Outliers
Count
Weight


Histogram of WBC_Count with Outliers
Count
WBC_Count


Histogram of Age with Outliers
Count


10
0
0
5
10
15
20
Age

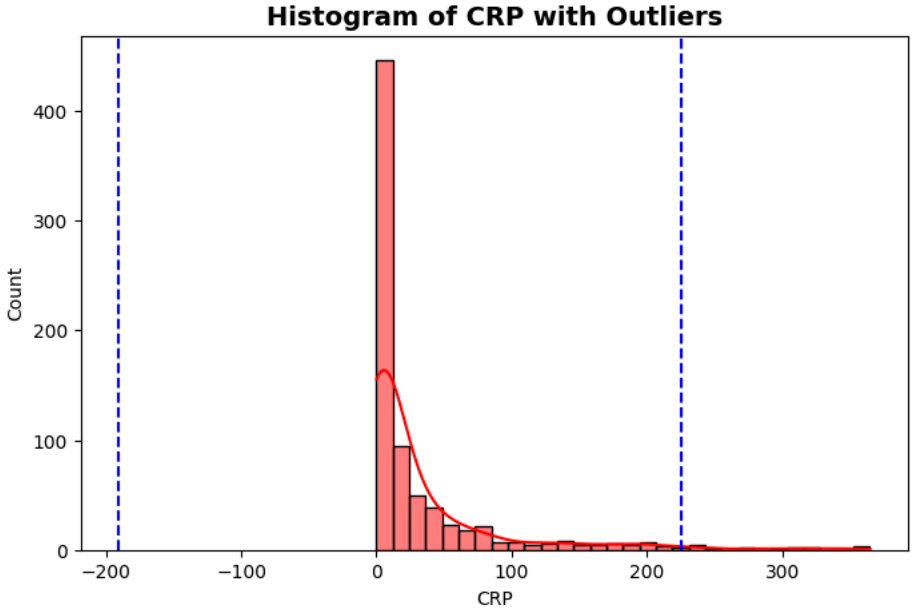

Histogram of Neutrophil_Percentage with Outliers
60
50
40
30
20
10
0
Count
40
60
80
100
120
Neutrophil_Percentage

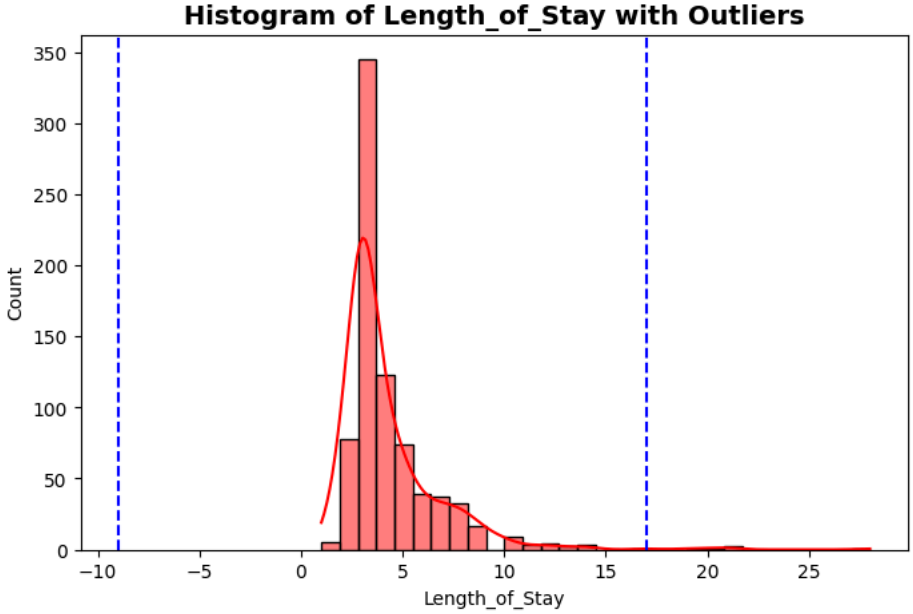
Outlier detection was based on the interquartile range (IQR), where any value falling outside the range $[Q1 - k \cdot \text{IQR},\ Q3 + k \cdot \text{IQR}]$ was flagged as an outlier. Here, k was set to 1.5 (1.5 x IQR) rule.

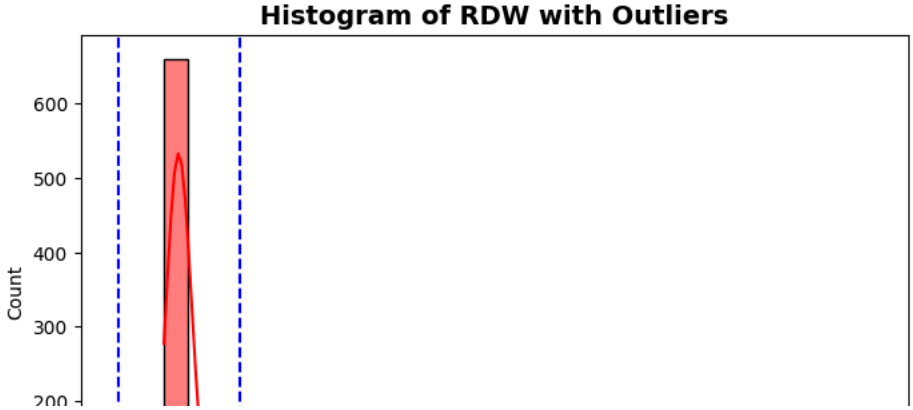
```
outliers = detect_outliers(X, 6)  # Detect Extreme outliers in features
```

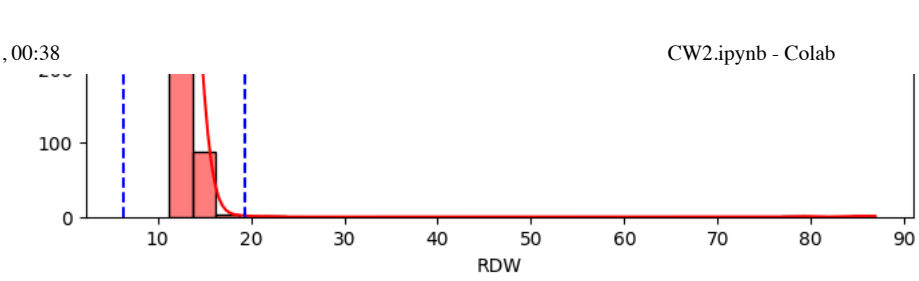
\n Outlier Detection Summary:

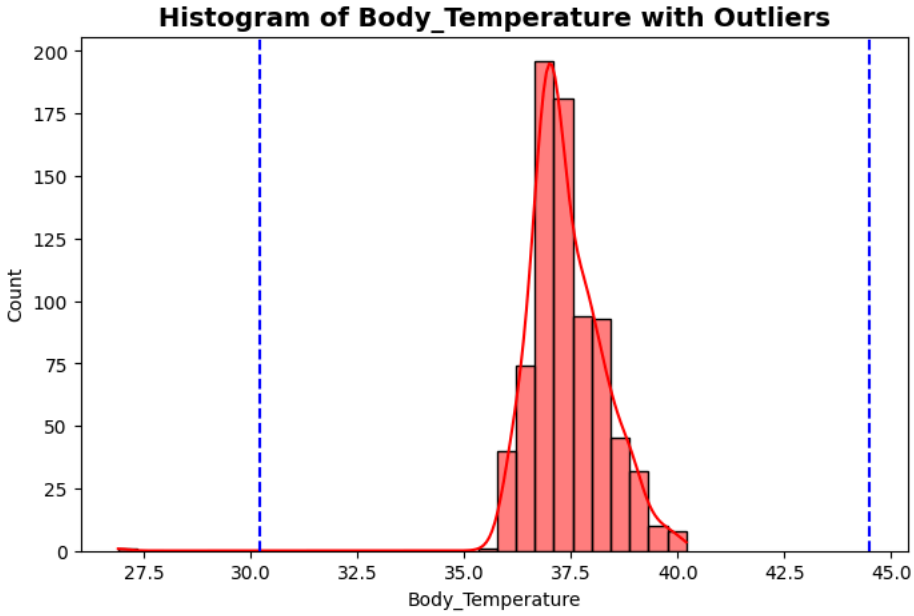
	Outlier Count	Outlier Percentage (%)
CRP	15	1.918159
Length_of_Stay	5	0.639386
RDW	4	0.511509
Body_Temperature	1	0.127877
RBC_Count	1	0.127877
Hemoglobin	1	0.127877
Height	0	0.000000
Age	0	0.000000
Appendix_Diameter	0	0.000000
Paedriatic_Appendicitis_Score	0	0.000000
Alvarado_Score	0	0.000000
Weight	0	0.000000
BMI	0	0.000000
WBC_Count	0	0.000000
Neutrophil_Percentage	0	0.000000
Thrombocyte_Count	0	0.000000

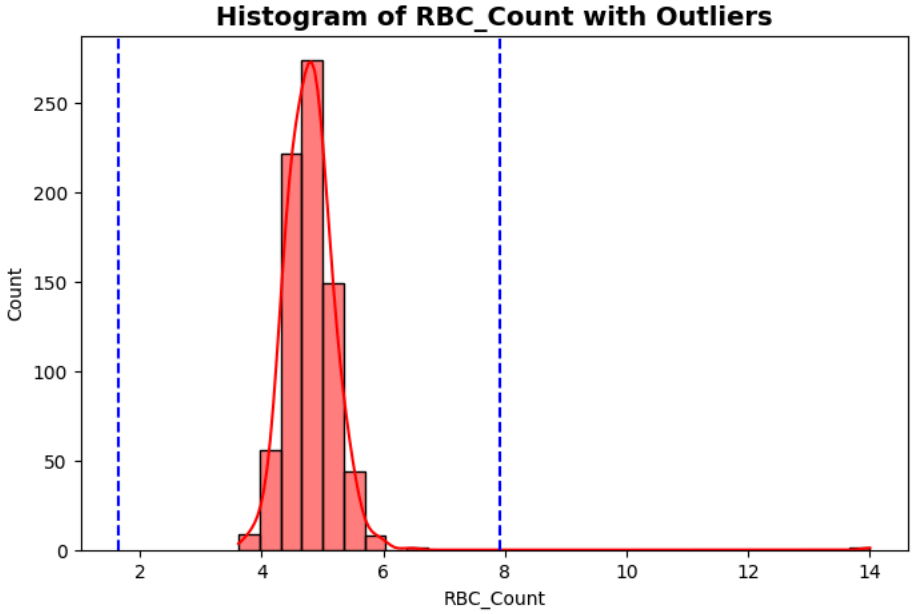


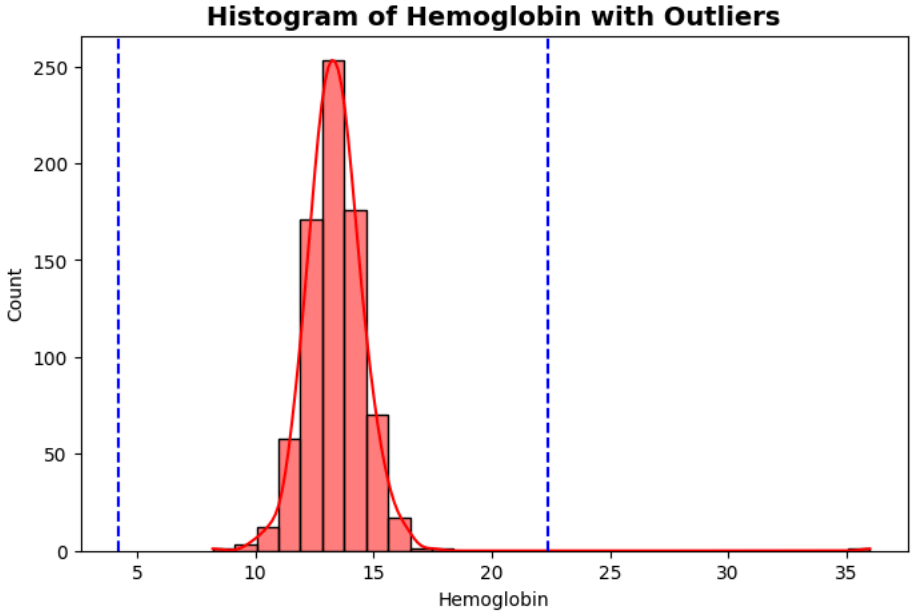





RDW


Histogram of Body_Temperature with Outliers
Count
Body_Temperature

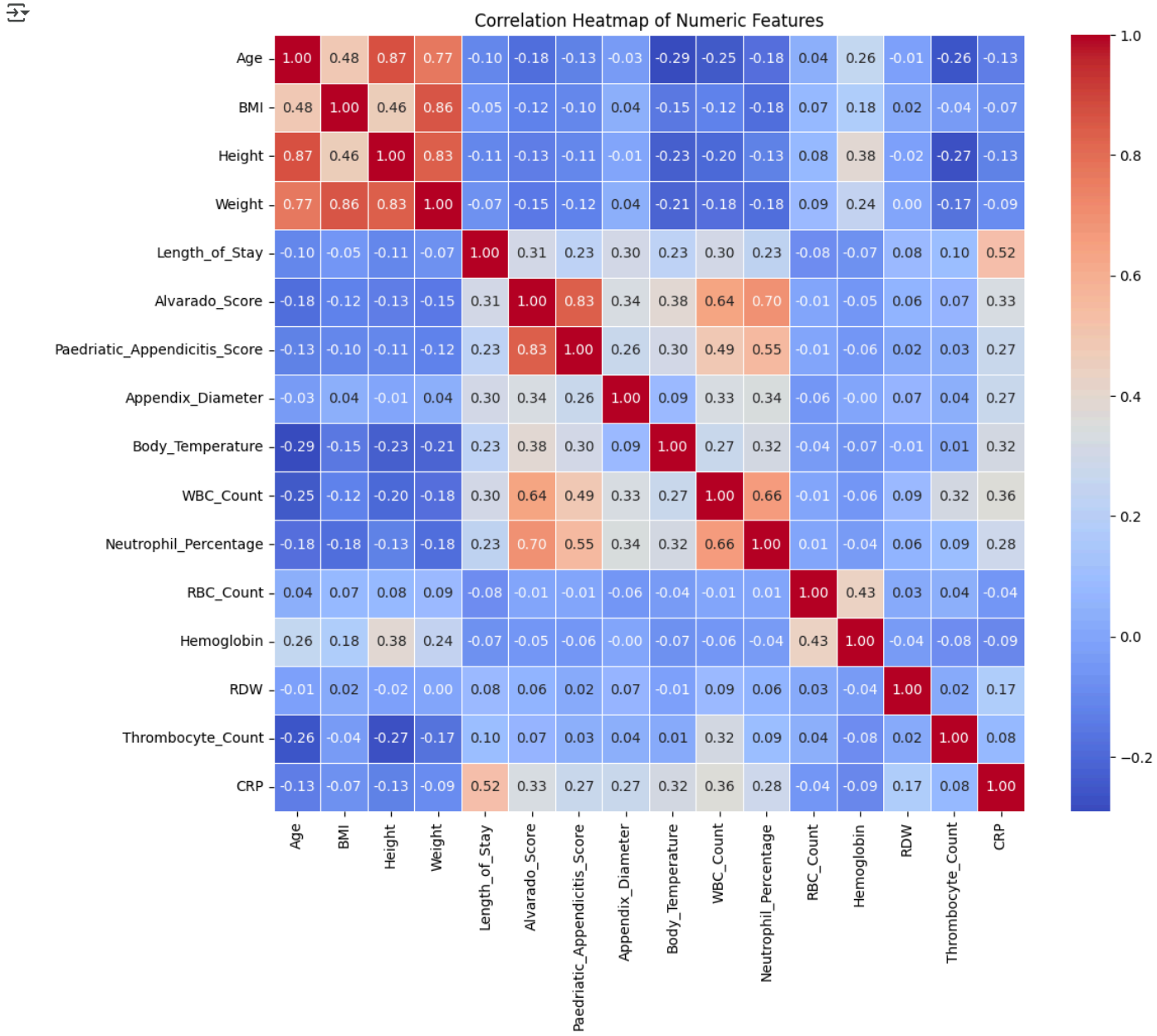

Histogram of RBC_Count with Outliers
Count
RBC_Count


Histogram of Hemoglobin with Outliers
Count
Hemoglobin

When the threshold was increased to 6×IQR to focus only on extreme outliers, CRP remained the most affected variable,

Predictor Correlation

```
num_x = X.select_dtypes(exclude=['object', 'category'])
f,ax = plt.subplots(figsize=(12, 10))
sns.heatmap(num_x.corr(), annot=True, cmap='coolwarm', linewidths=.5, fmt= '.2f',ax=ax)
plt.title('Correlation Heatmap of Numeric Features')
plt.show()
```



In a pediatric population (ages 0-18), the strong correlation between age and height (.87), and age and weight (.77) is expected due to natural growth patterns. However, this level of multicollinearity could effect the stability of a logistic regression model, suggesting that it may be beneficial to exclude one of the variables with a high correlation.

```
y_full = y.copy()
```

```
y = y_full['Diagnosis']
```

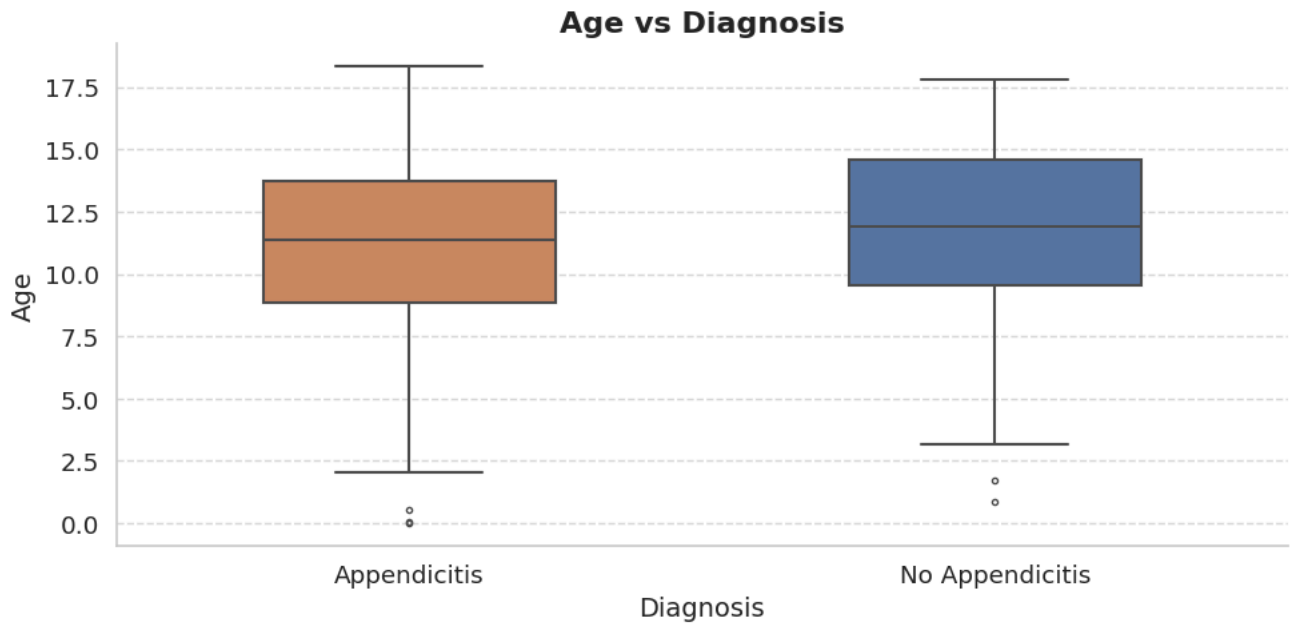
```
y
```

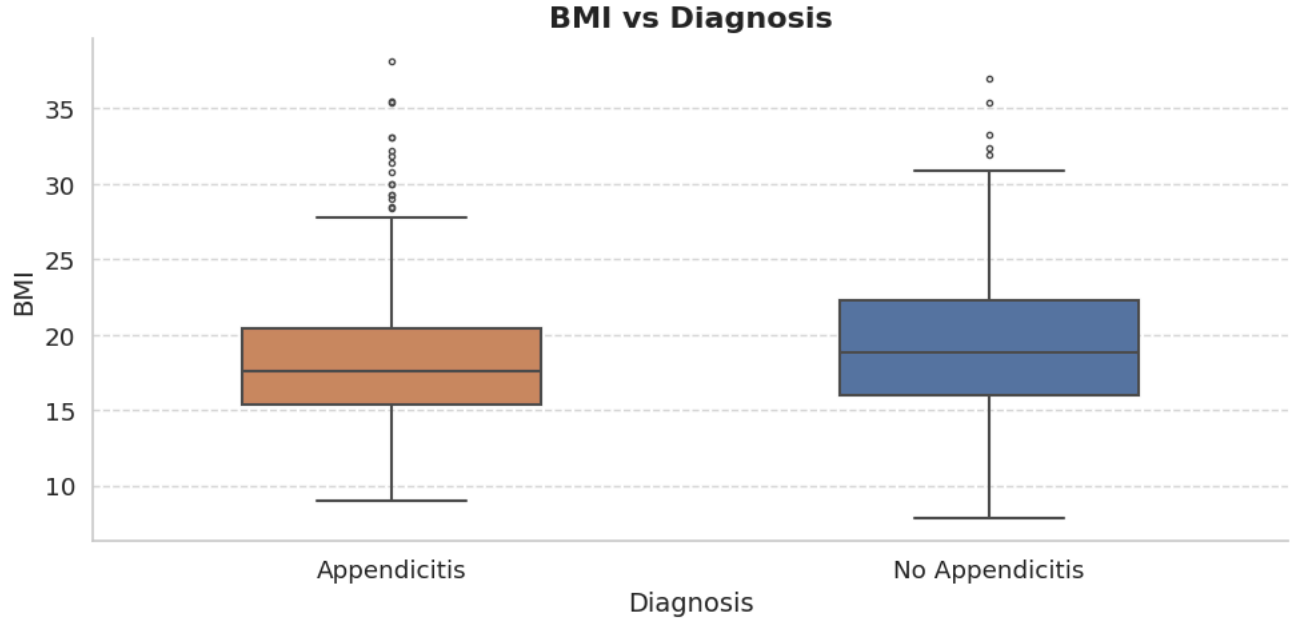
	Diagnosis
0	appendicitis
1	no appendicitis
2	no appendicitis
3	no appendicitis
4	appendicitis
...	...
777	appendicitis
778	appendicitis
779	appendicitis
780	appendicitis
781	appendicitis

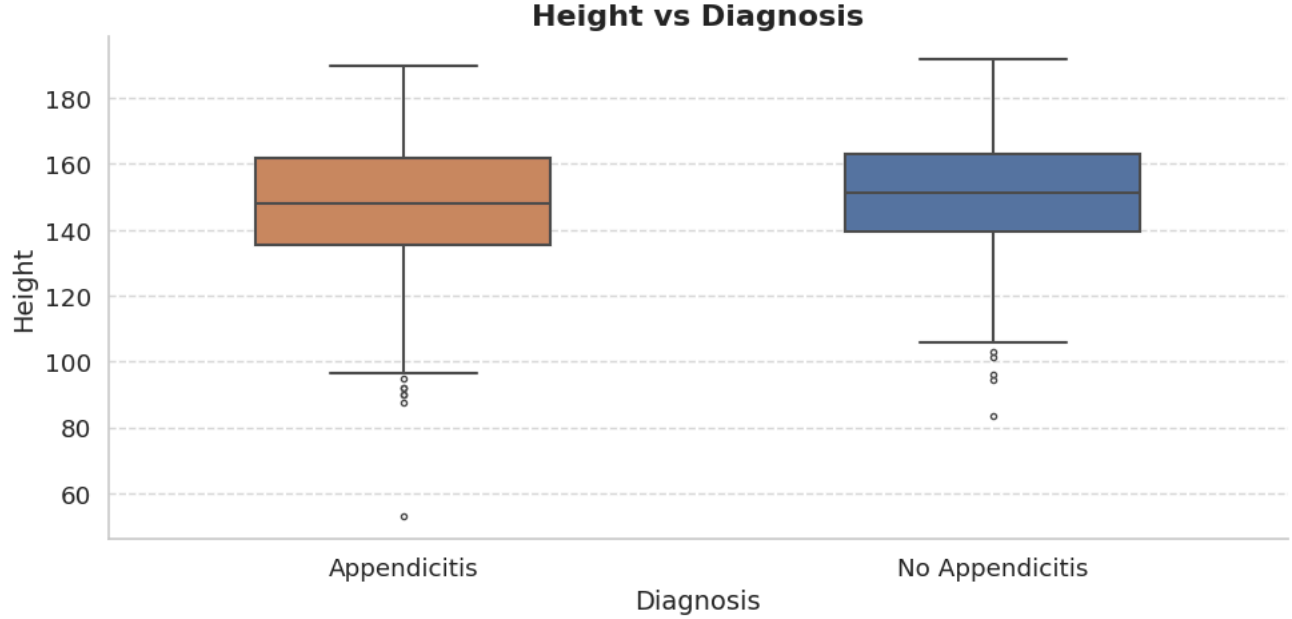
782 rows × 1 columns

dtype: object

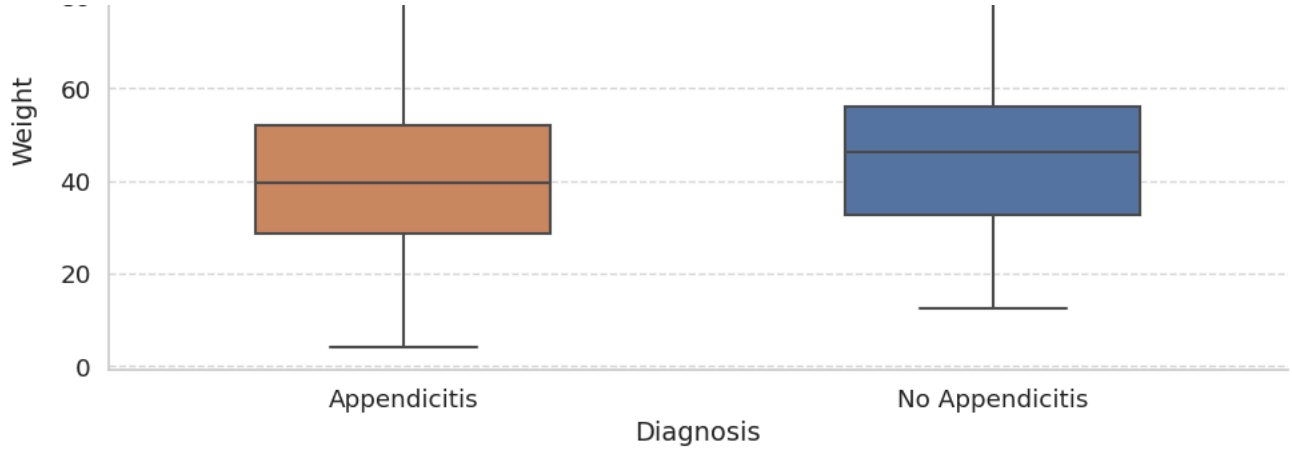
```
sns.set_theme(style="whitegrid", palette="muted", font_scale=1.2)
custom_palette = {
    'no appendicitis': "#4C72B0",  # Blue for "No Appendicitis"
    'appendicitis': "#DD8452"   # Orange for "Appendicitis"
}

for col in numeric_cols:
    plt.figure(figsize=(10, 5))
    sns.boxplot(
        x=y,
        y=X[col],
        palette=custom_palette,
        width=0.5,
        fliersize=3,  # Smaller outliers
        linewidth=1.5  # Thicker lines
    )

    plt.title(f'{col} vs Diagnosis', fontsize=16, weight='bold')
    plt.xlabel('Diagnosis', fontsize=14)
    plt.ylabel(col, fontsize=14)
    plt.xticks(ticks=['no appendicitis', 'appendicitis'], labels=['No Appendicitis', 'Appendicitis'])
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    sns.despine()

    plt.tight_layout()
    plt.show()
```

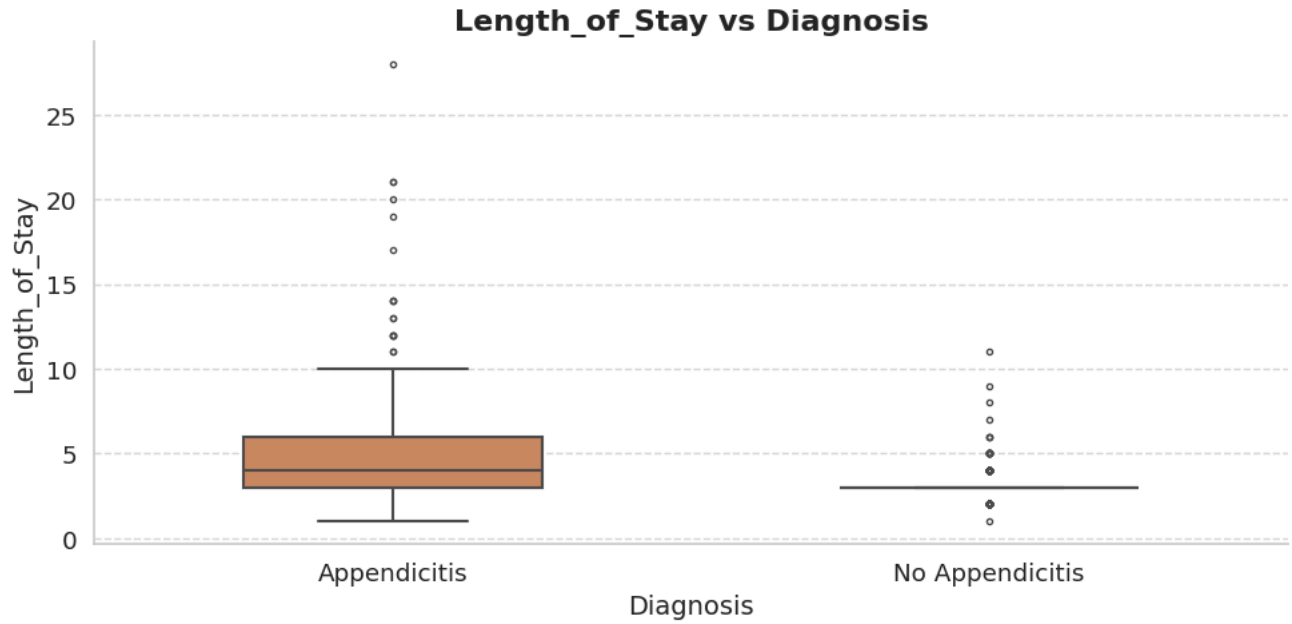

Age vs Diagnosis
Age
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Appendicitis
No Appendicitis
Diagnosis

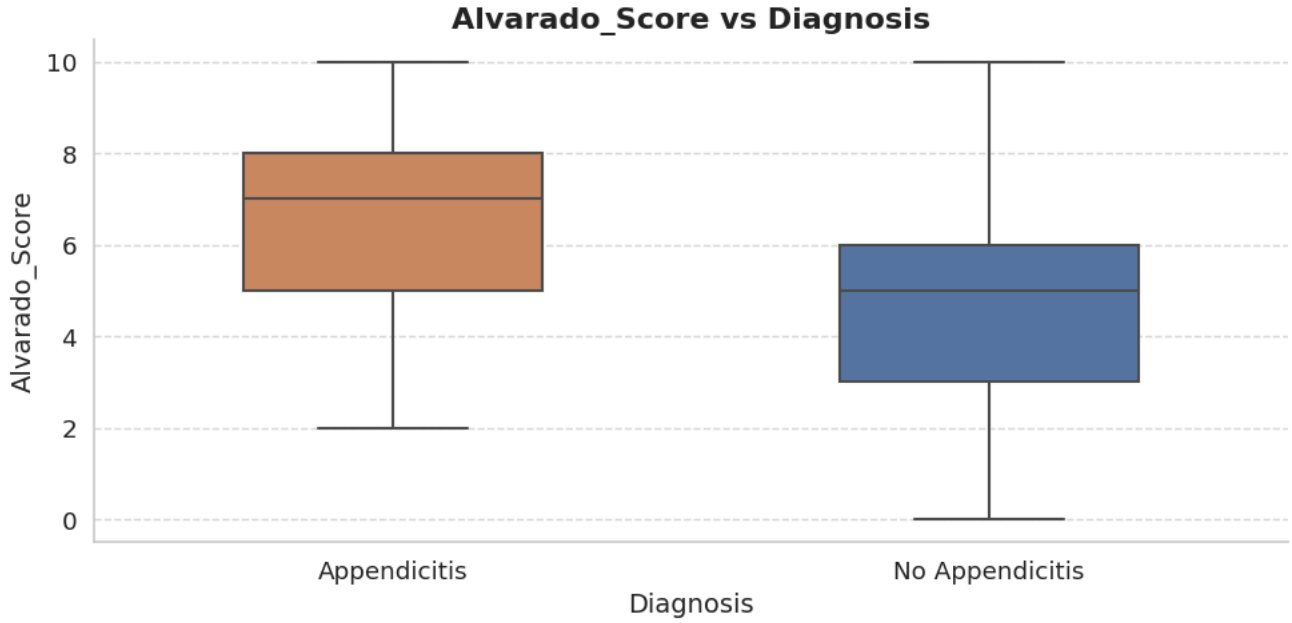

BMI vs Diagnosis
BMI
35
30
25
20
15
10
Appendicitis
No Appendicitis
Diagnosis

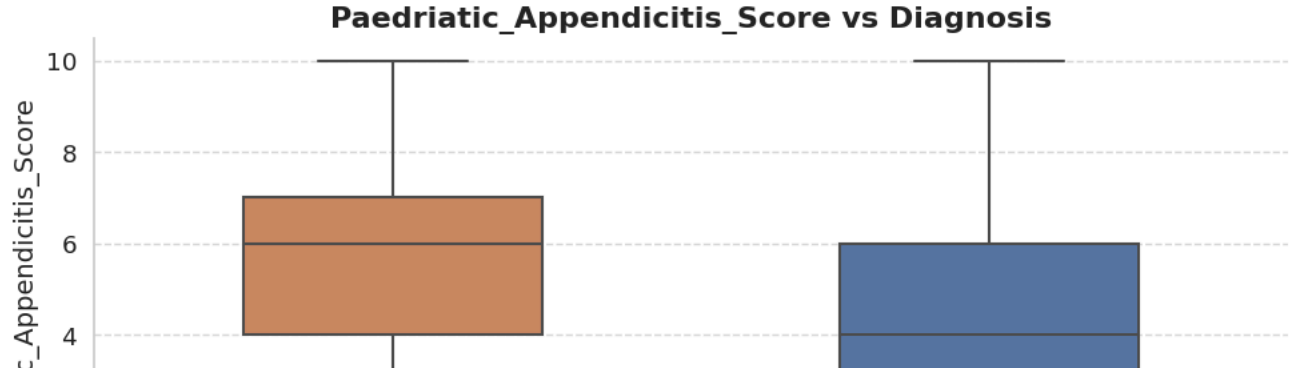

Height vs Diagnosis
Height
180
160
140
120
100
80
60
Appendicitis
No Appendicitis
Diagnosis

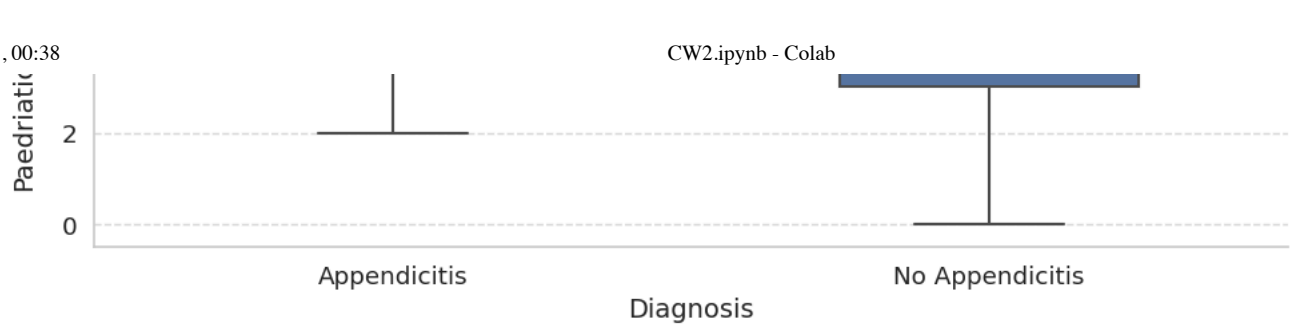

Weight vs Diagnosis
100
80

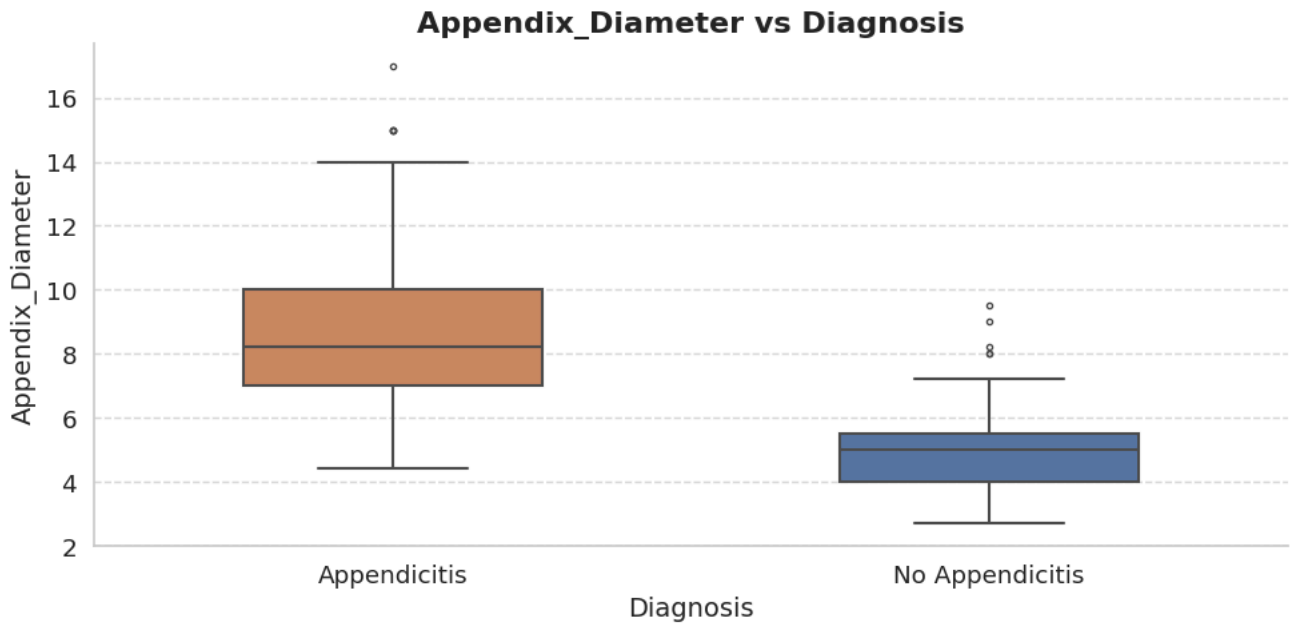

Weight
60
40
20
0
Appendicitis
No Appendicitis
Diagnosis

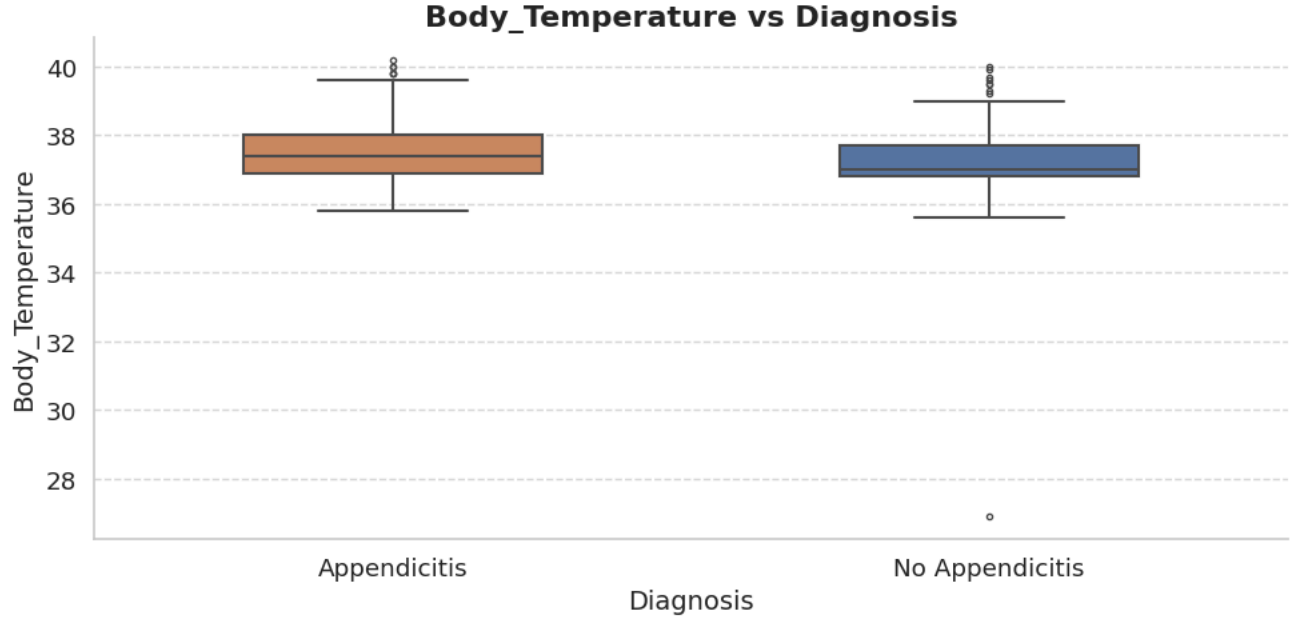

Length_of_Stay vs Diagnosis
Length_of_Stay
25
20
15
10
5
0
Appendicitis
No Appendicitis
Diagnosis

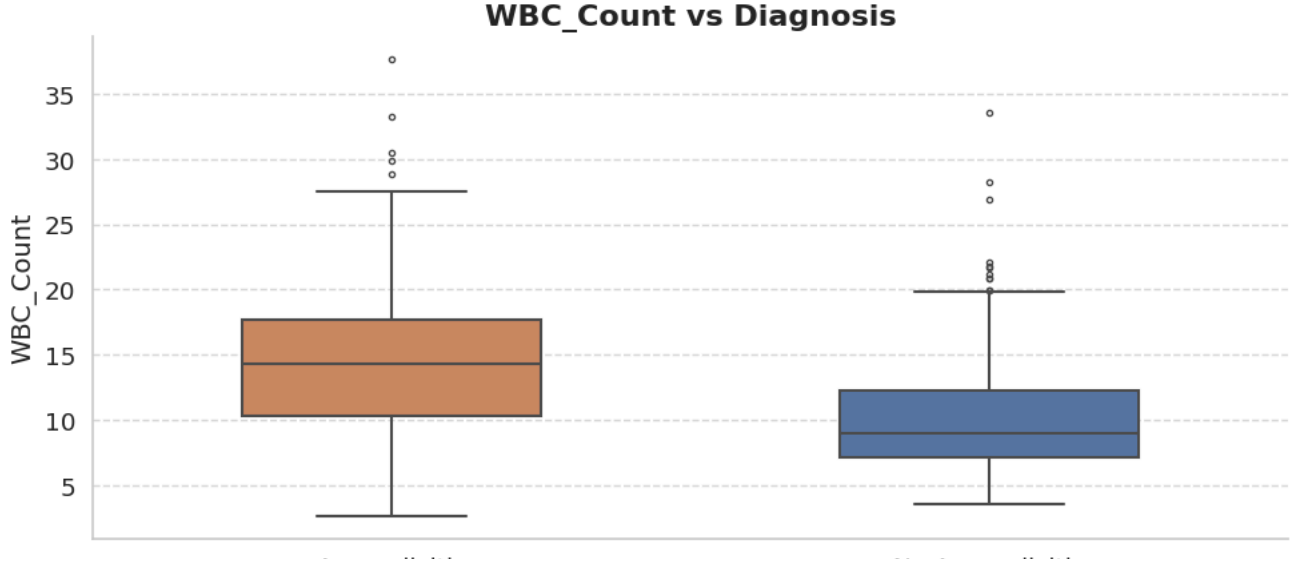

Alvarado_Score vs Diagnosis
Alvarado_Score
10
8
6
4
2
0
Appendicitis
No Appendicitis
Diagnosis

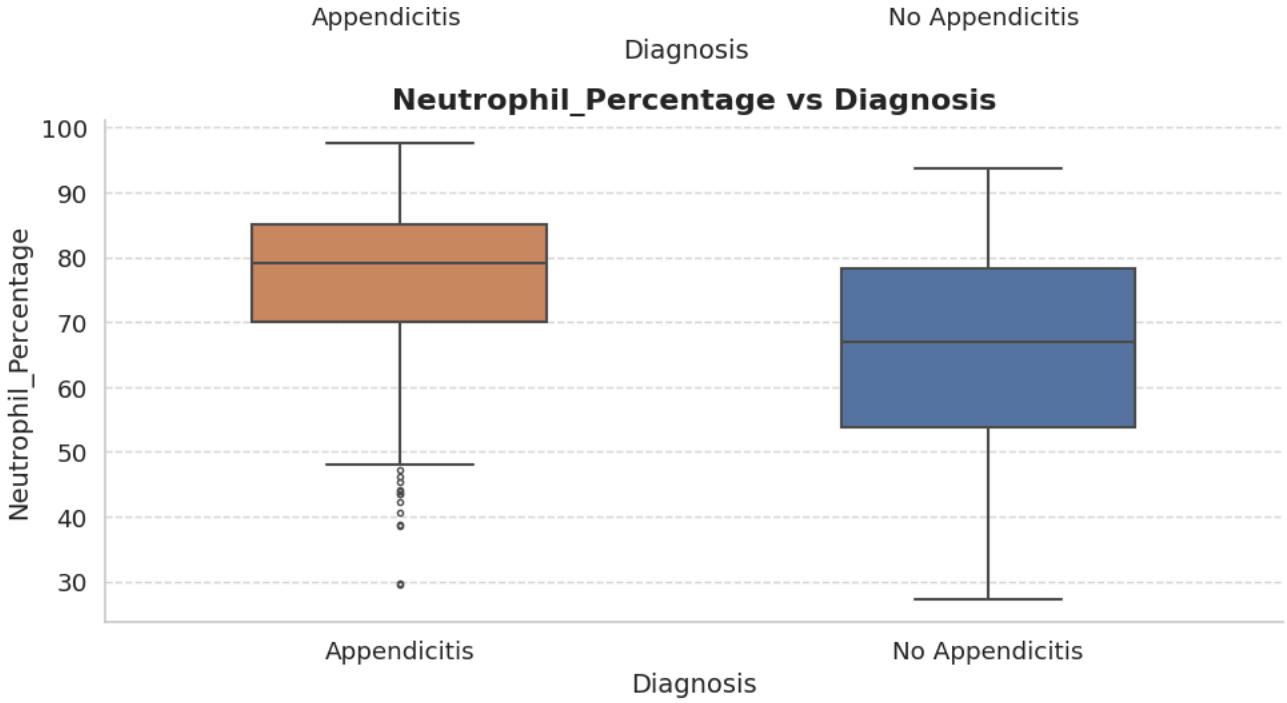

Paedriatic_Appendicitis_Score vs Diagnosis
10
8
6
4

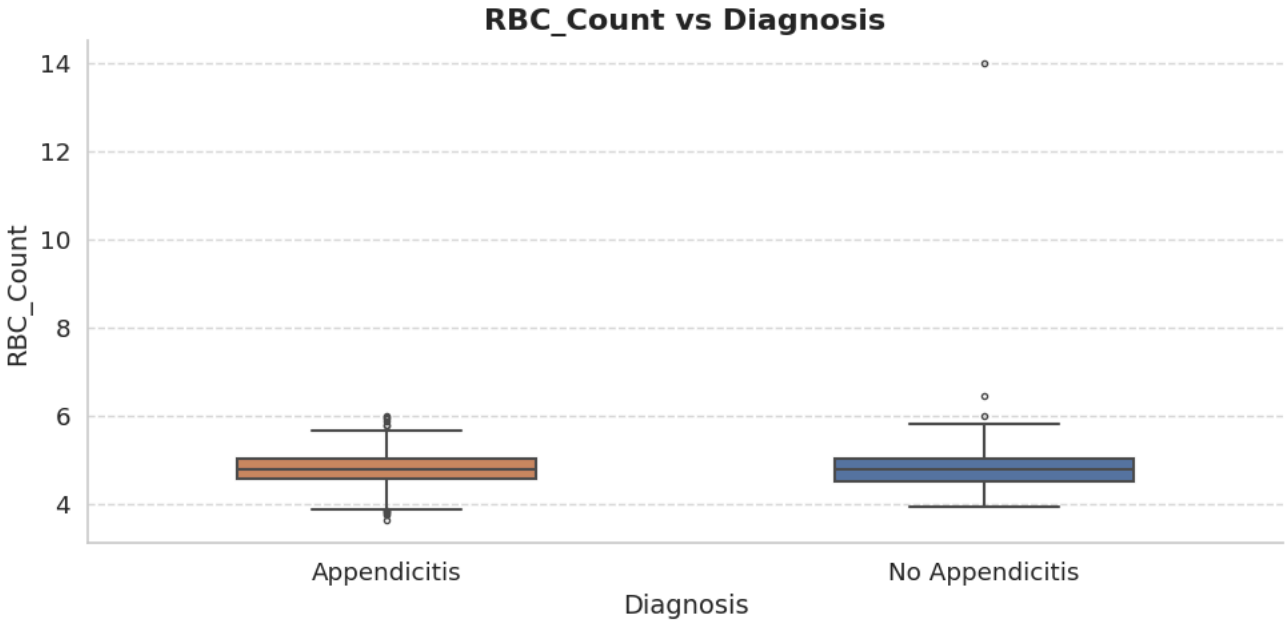

Paedriatic
2
0
Appendicitis
No Appendicitis
Diagnosis

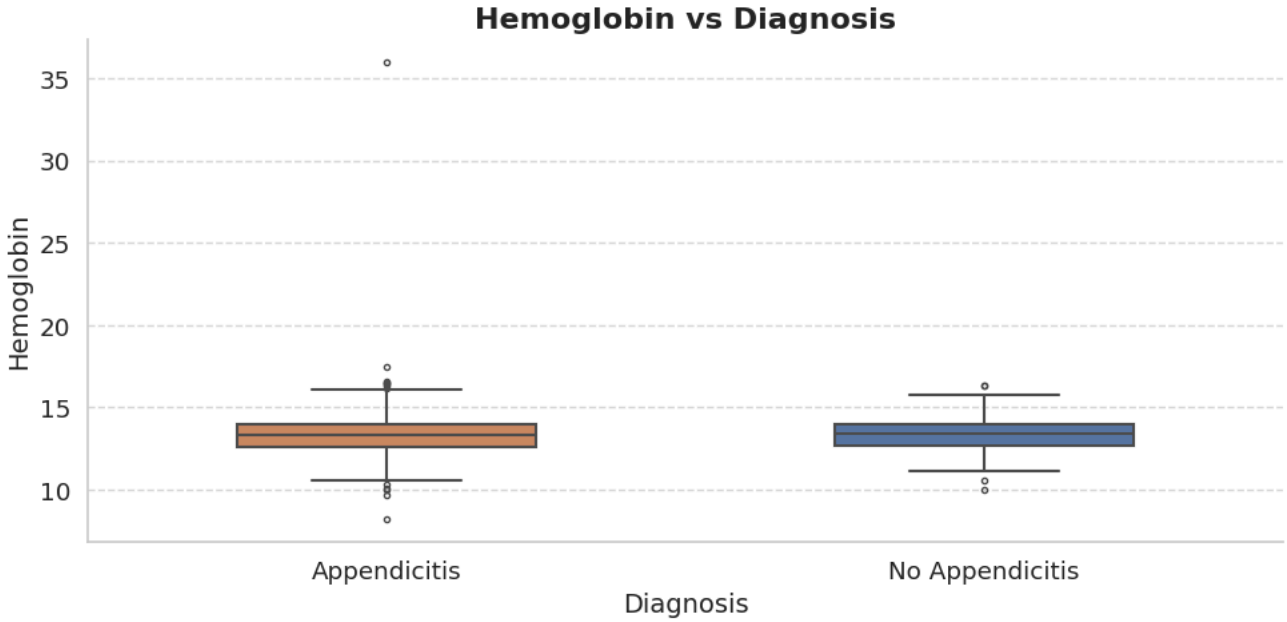

Appendix_Diameter vs Diagnosis
Appendix_Diameter
16
14
12
10
8
6
4
2
Appendicitis
No Appendicitis
Diagnosis


Body_Temperature vs Diagnosis
Body_Temperature
40
38
36
34
32
30
28
Appendicitis
No Appendicitis
Diagnosis

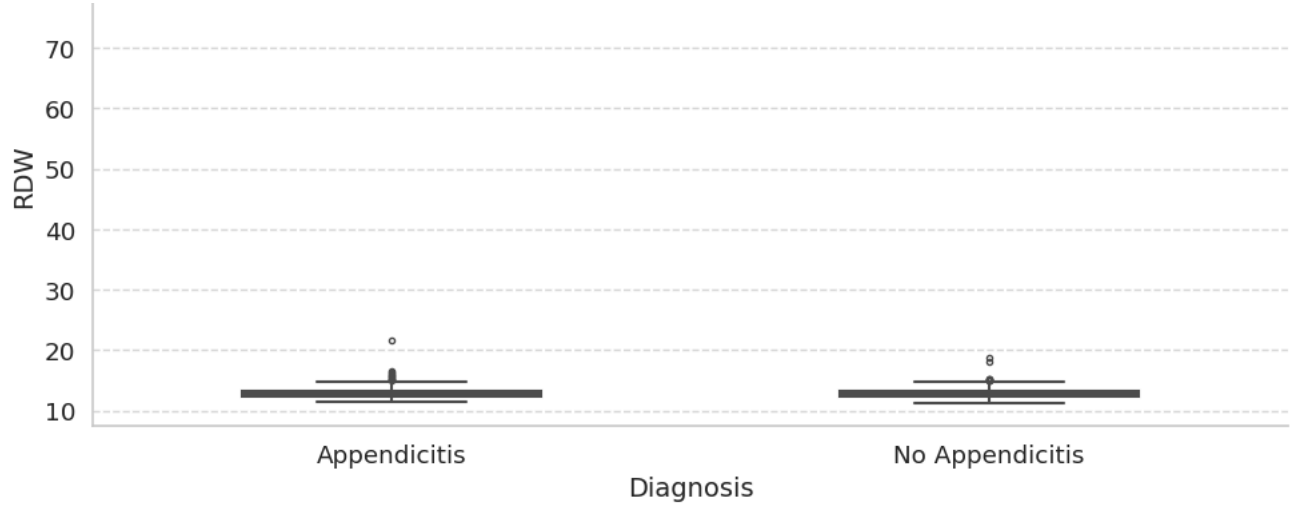

WBC_Count vs Diagnosis
WBC_Count
35
30
25
20
15
10
5

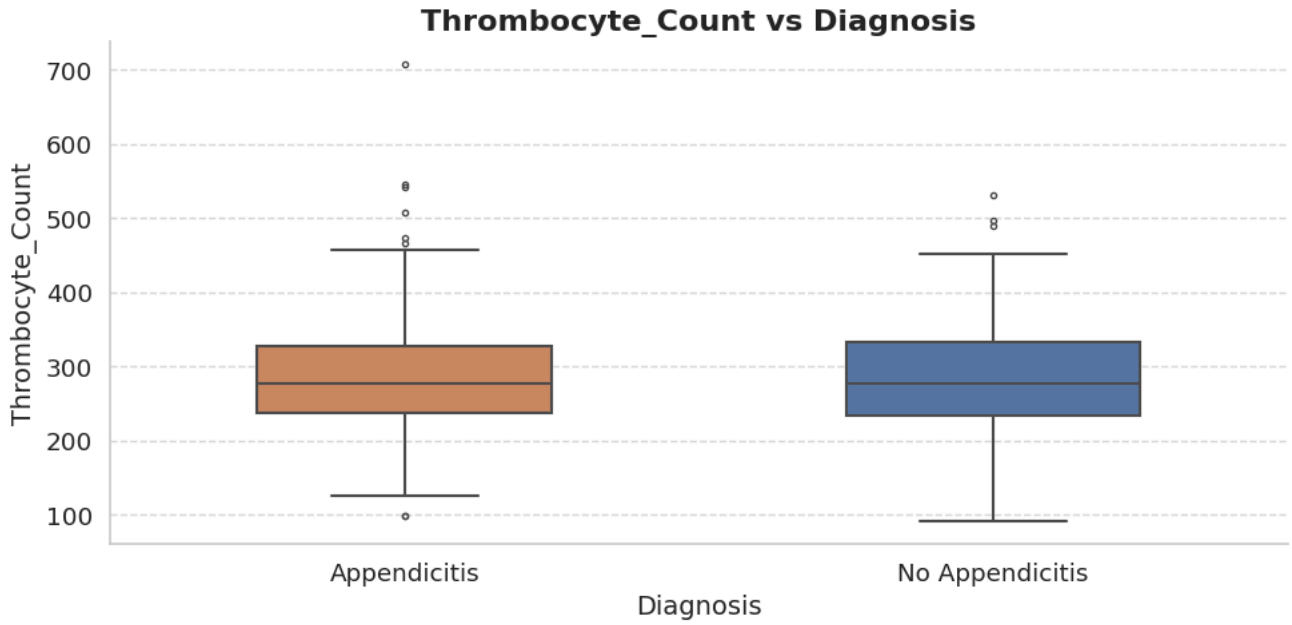

Appendicitis
No Appendicitis
Diagnosis
Neutrophil_Percentage vs Diagnosis
Neutrophil_Percentage
100
90
80
70
60
50
40
30
Appendicitis
No Appendicitis
Diagnosis

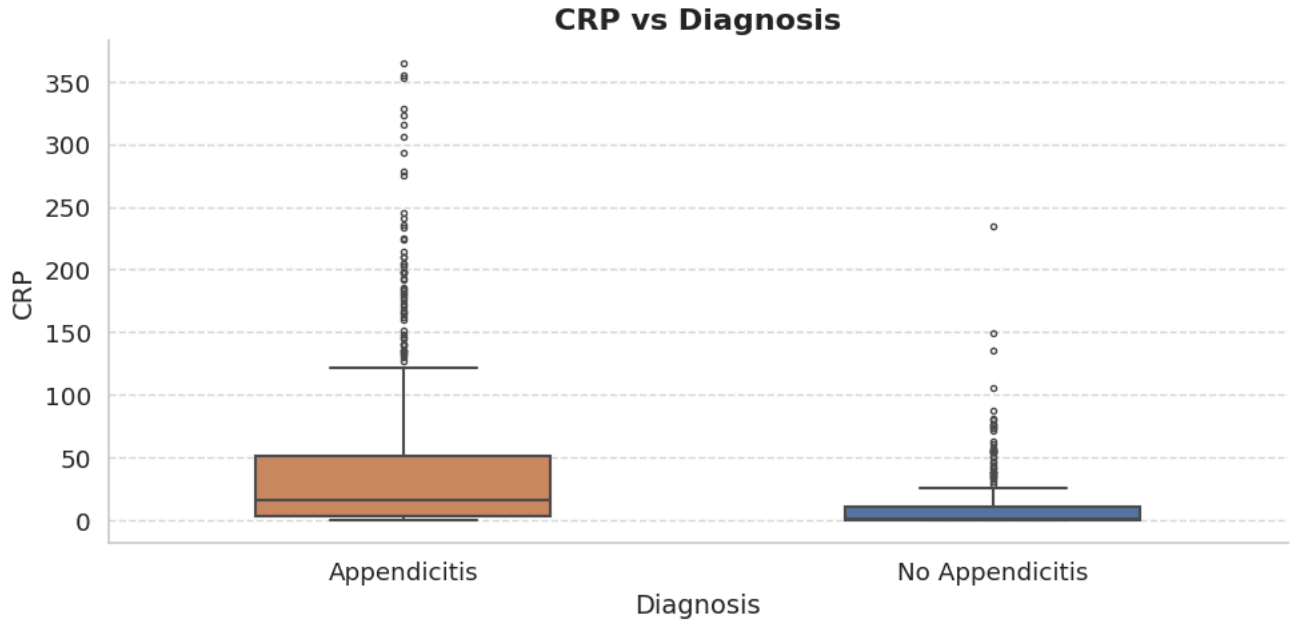

RBC_Count vs Diagnosis
RBC_Count
14
12
10
8
6
4
Appendicitis
No Appendicitis
Diagnosis


Hemoglobin vs Diagnosis
Hemoglobin
35
30
25
20
15
10
Appendicitis
No Appendicitis
Diagnosis

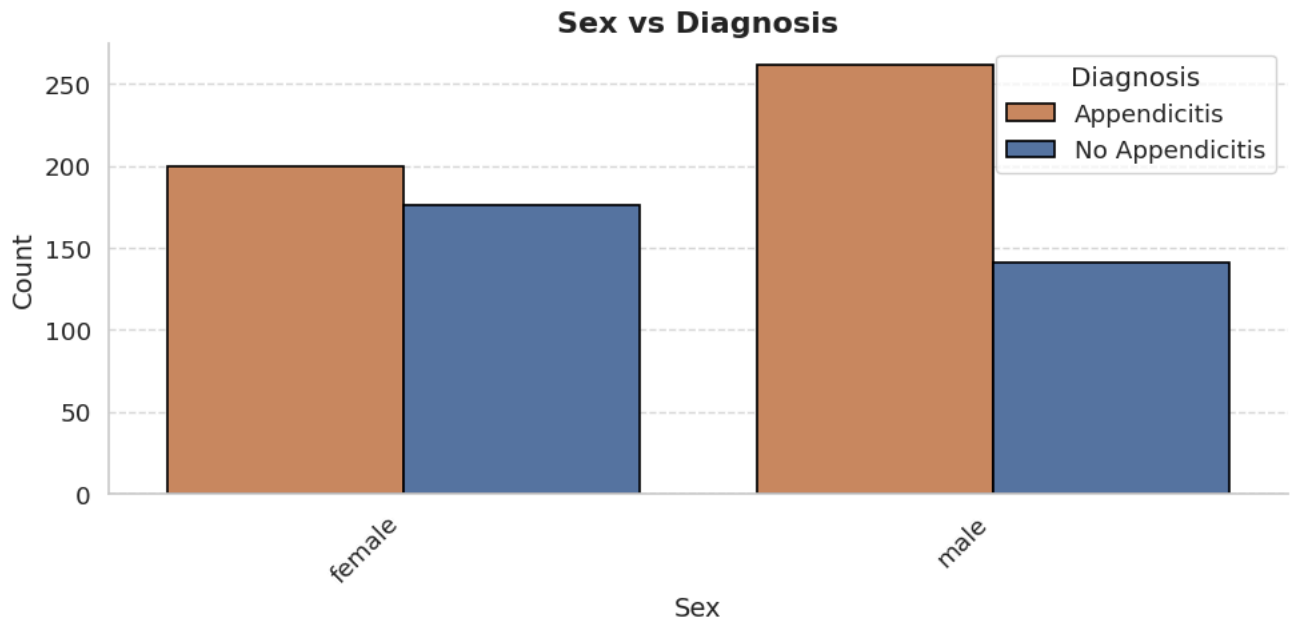

RDW vs Diagnosis
90
80

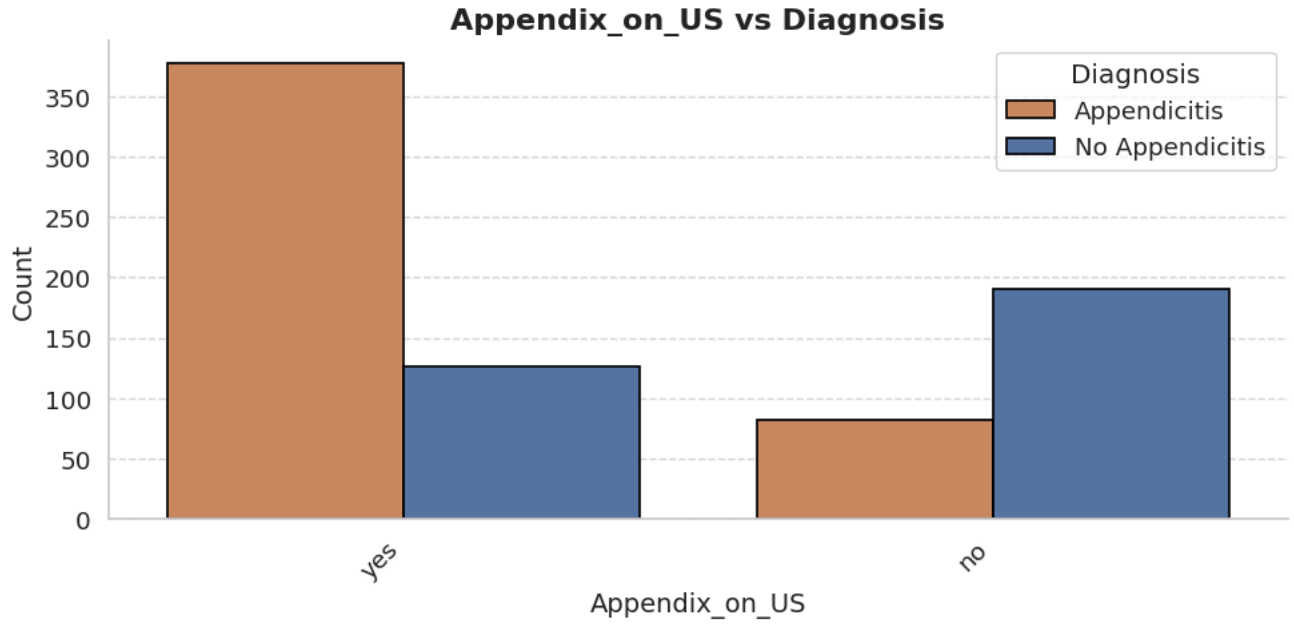

RDW
70
60
50
40
30
20
10
Appendicitis
No Appendicitis
Diagnosis

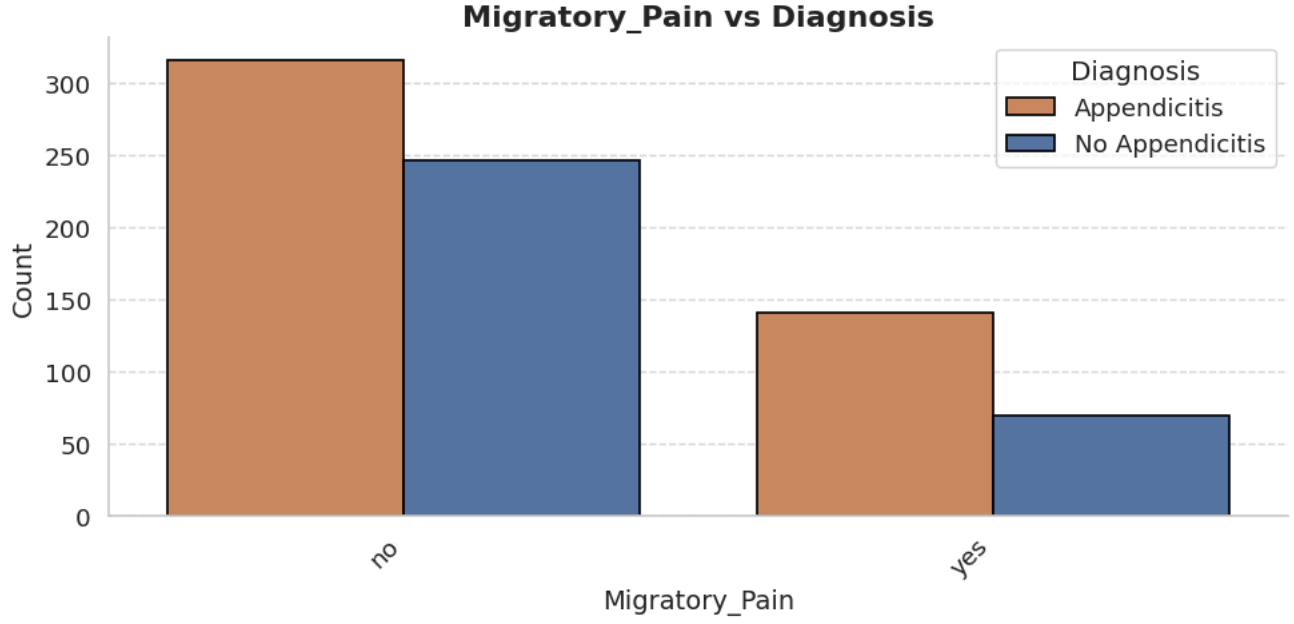

Thrombocyte_Count vs Diagnosis
Thrombocyte_Count
700
600
500
400
300
200
100
Appendicitis
No Appendicitis
Diagnosis


CRP vs Diagnosis
CRP
350
300
250
200
150
100
50
0
Appendicitis
No Appendicitis
Diagnosis

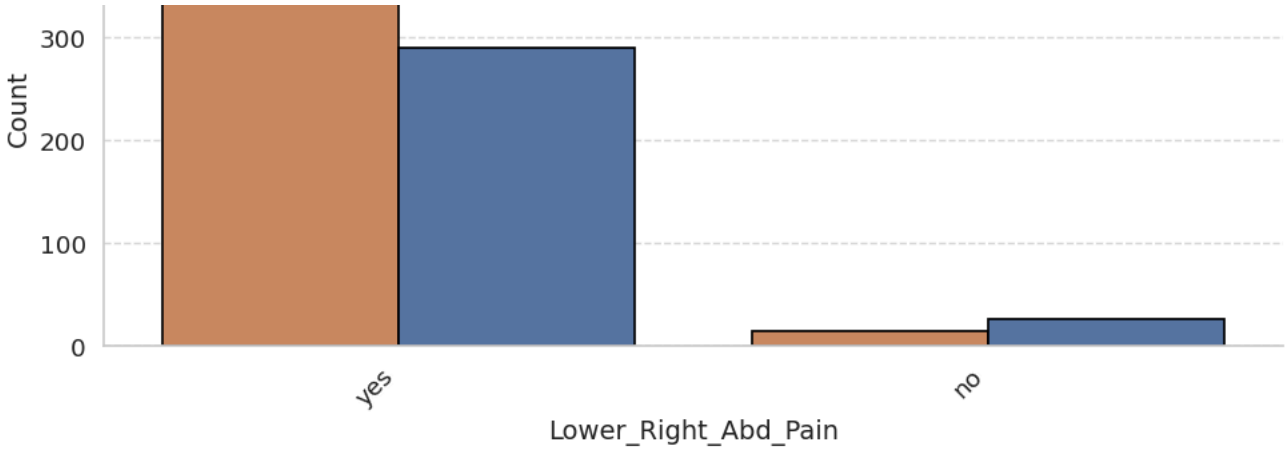
```
sns.set_theme(style="whitegrid", palette="muted", font_scale=1.2)
custom_palette = {
    'no appendicitis': "#4C72B0",  # Blue for "No Appendicitis"
    'appendicitis': "#DD8452"   # Orange for "Appendicitis"
}
for col in categorical_cols:
    plt.figure(figsize=(10, 5))
    sns.countplot(
        x=X[col],
        hue=y,
        palette=custom_palette,
        edgecolor='black',  # Adds nice separation
        linewidth=1.2
    )

    plt.title(f'{col} vs Diagnosis', fontsize=16, weight='bold')
    plt.xlabel(col, fontsize=14)
    plt.ylabel('Count', fontsize=14)
    plt.xticks(rotation=45, ha='right')
    plt.legend(title='Diagnosis', labels=['Appendicitis', 'No Appendicitis'])

    plt.grid(axis='y', linestyle='--', alpha=0.7)
    sns.despine()

    plt.tight_layout()
    plt.show()
```

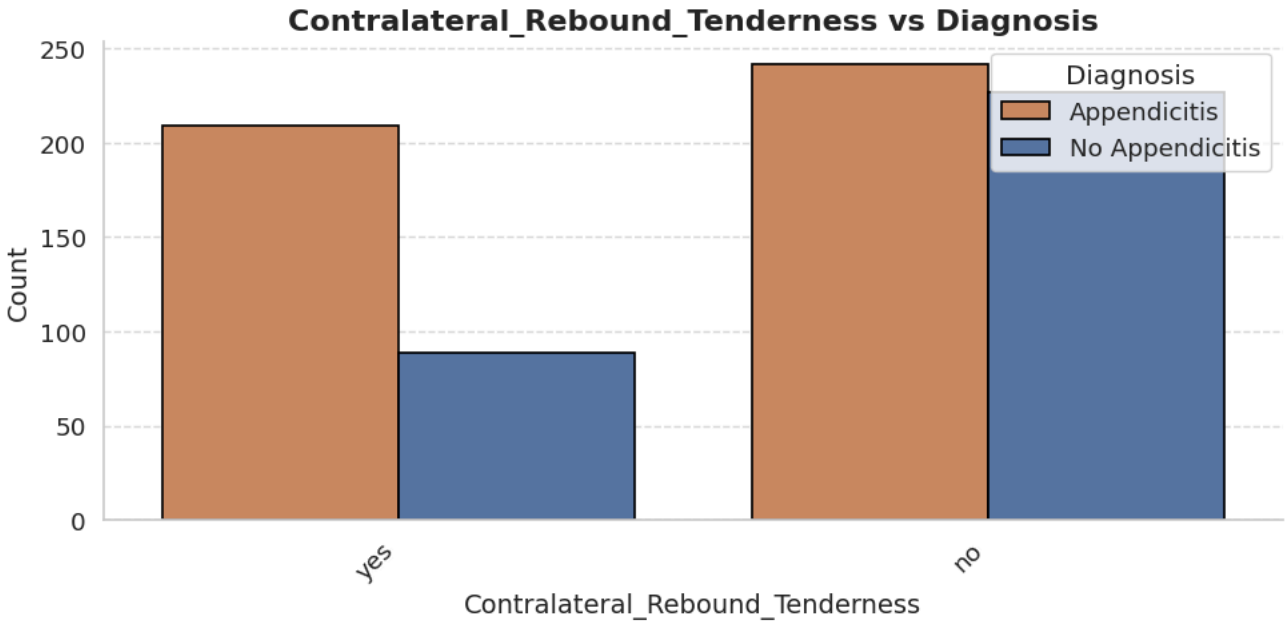

Sex vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
0
50
100
150
200
250
female
male
Sex

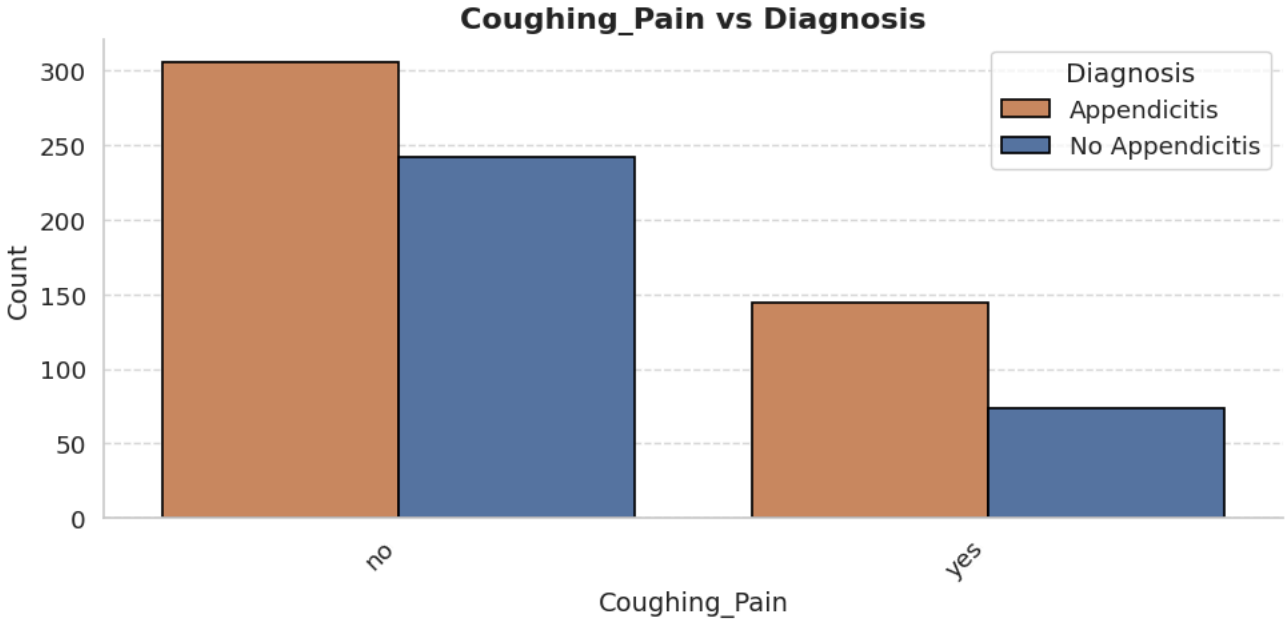

Appendix_on_US vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
0
50
100
150
200
250
300
350
yes
no
Appendix_on_US

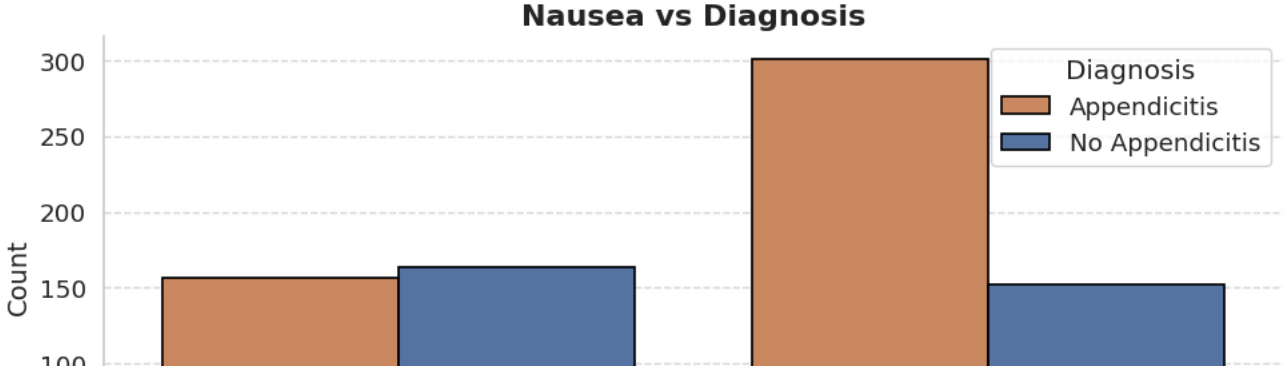

Migratory_Pain vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
0
50
100
150
200
250
300
no
yes
Migratory_Pain

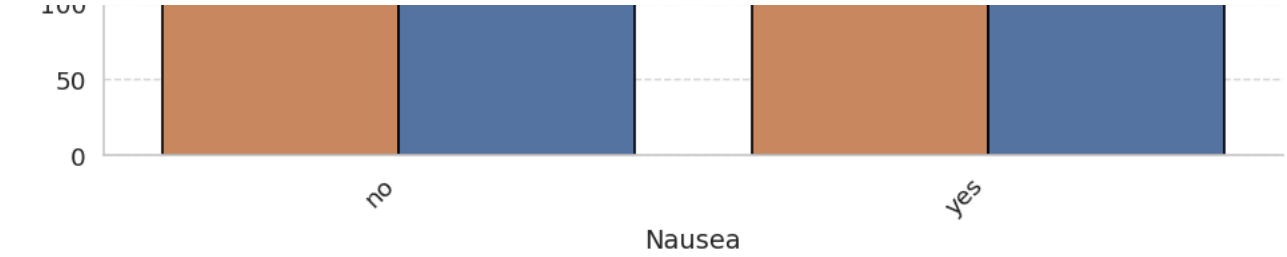

Lower_Right_Abd_Pain vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
400

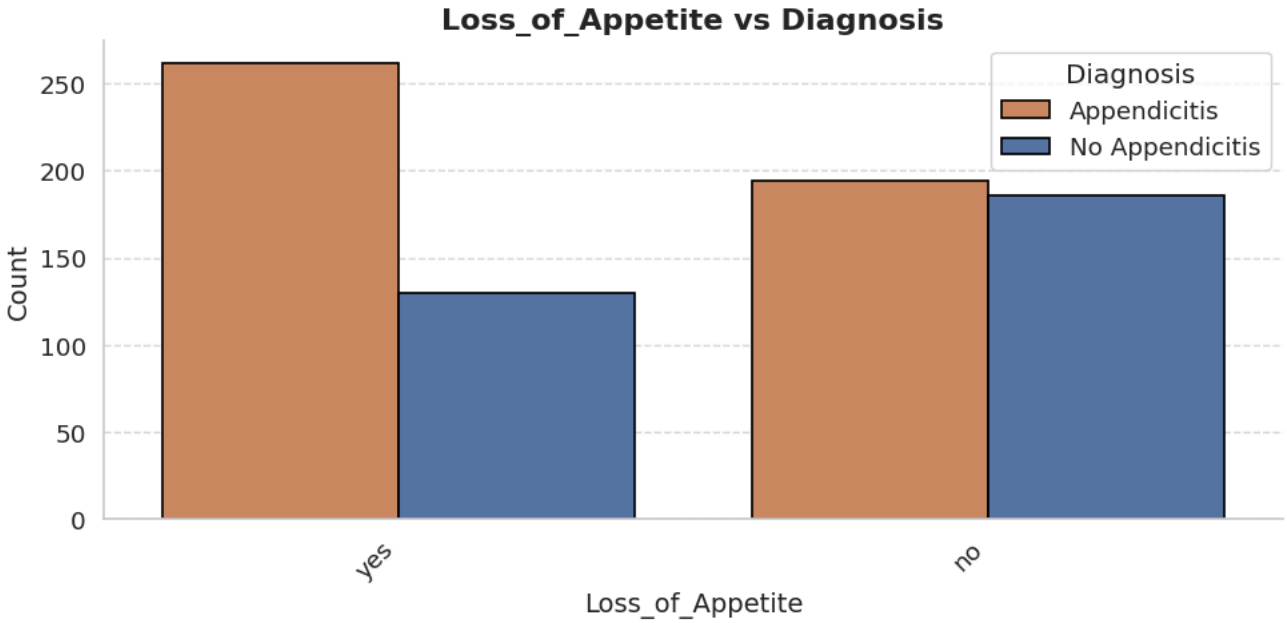

Count
300
200
100
0
yes
no
Lower_Right_Abd_Pain

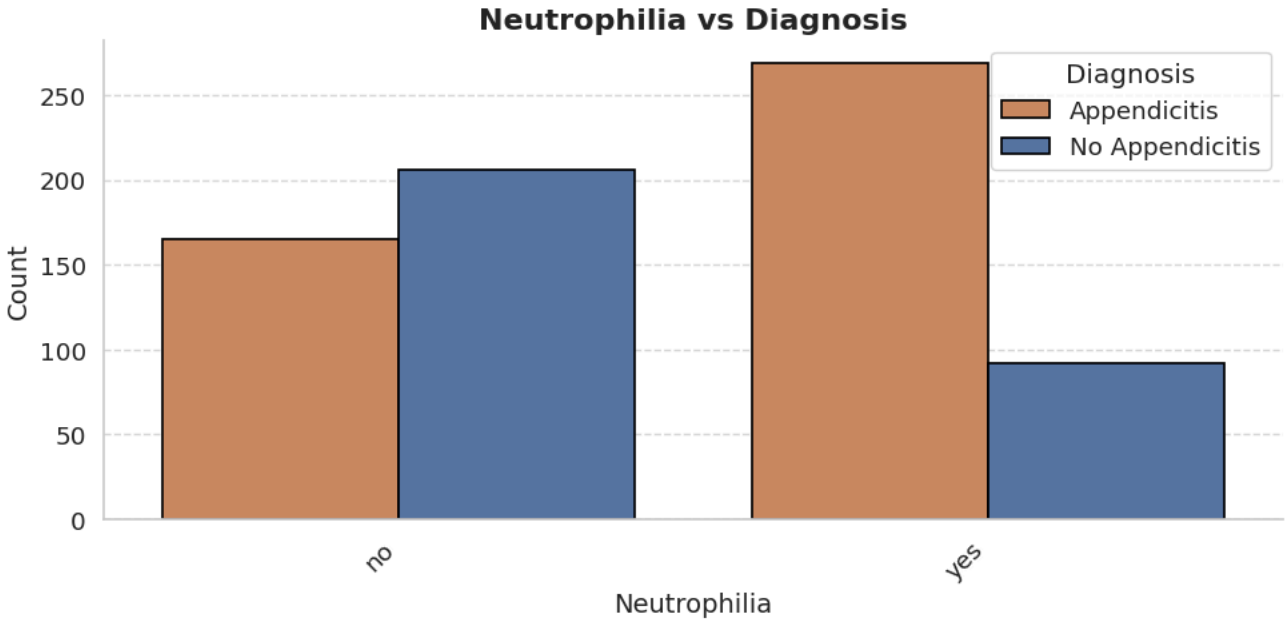

Contralateral_Rebound_Tenderness vs Diagnosis
Count
250
200
150
100
50
0
Diagnosis
Appendicitis
No Appendicitis
yes
no
Contralateral_Rebound_Tenderness

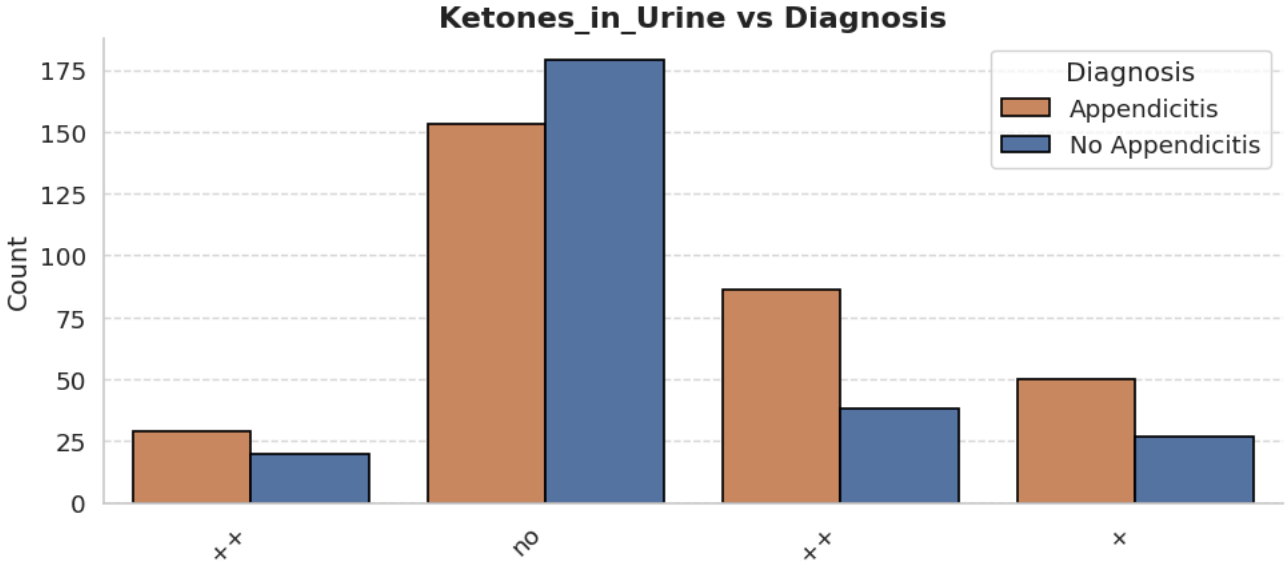

Coughing_Pain vs Diagnosis
Count
300
250
200
150
100
50
0
Diagnosis
Appendicitis
No Appendicitis
no
yes
Coughing_Pain

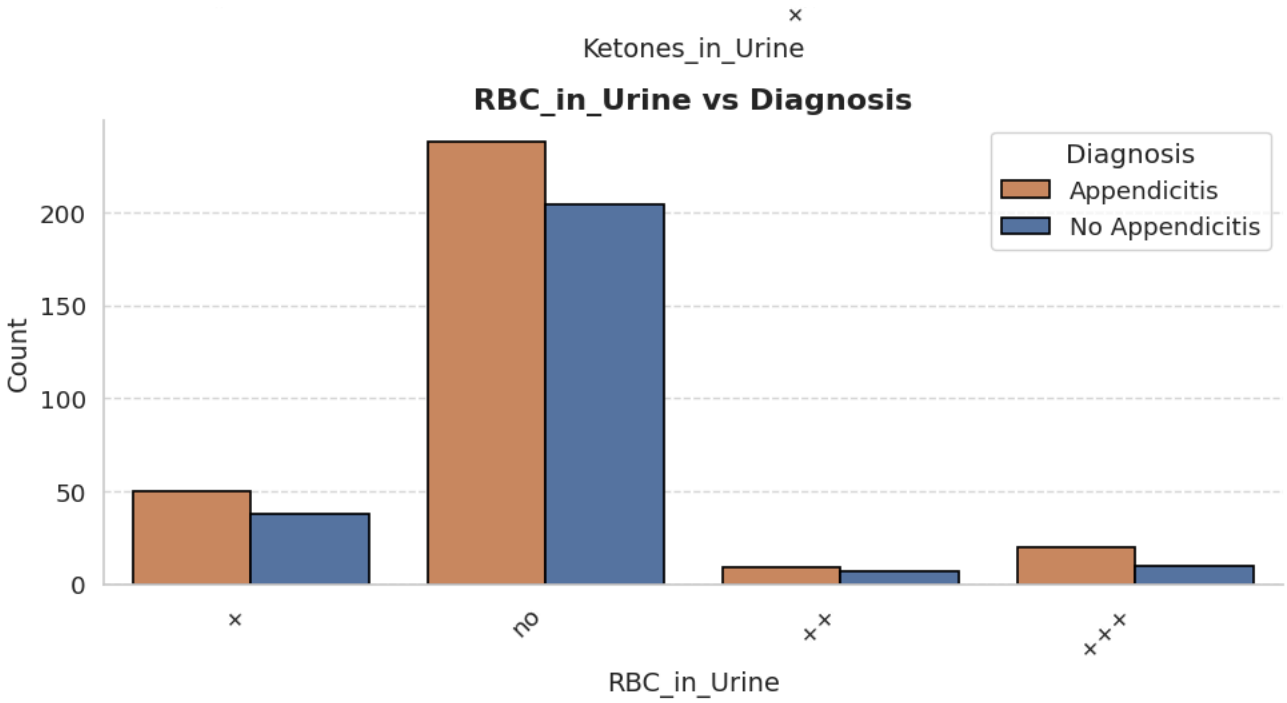

Nausea vs Diagnosis
Count
300
250
200
150
100
Diagnosis
Appendicitis
No Appendicitis

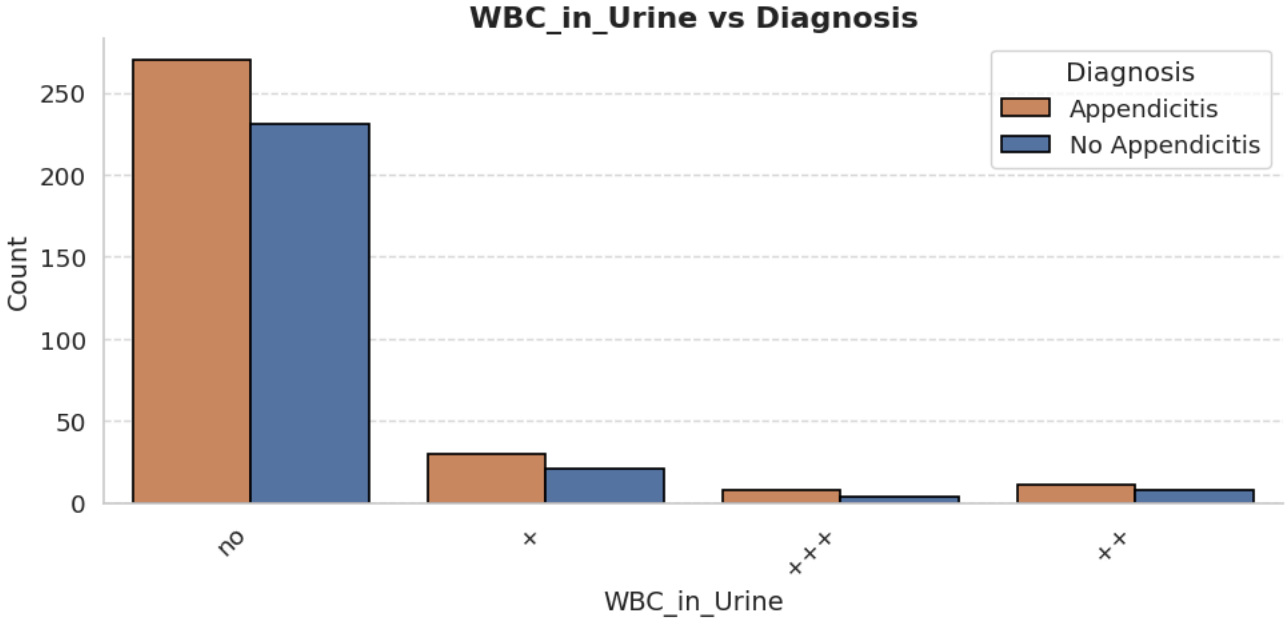

100
50
0
no
yes
Nausea

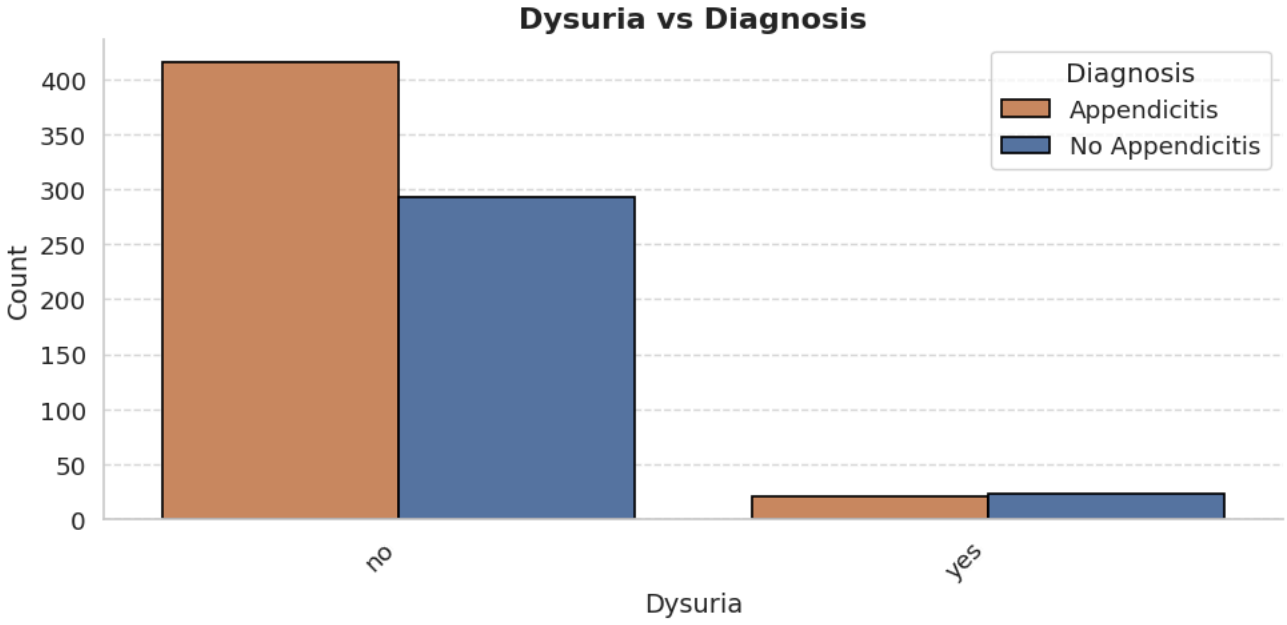

Loss_of_Appetite vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
250
200
150
100
50
0
yes
no
Loss_of_Appetite


Neutrophilia vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
250
200
150
100
50
0
no
yes
Neutrophilia

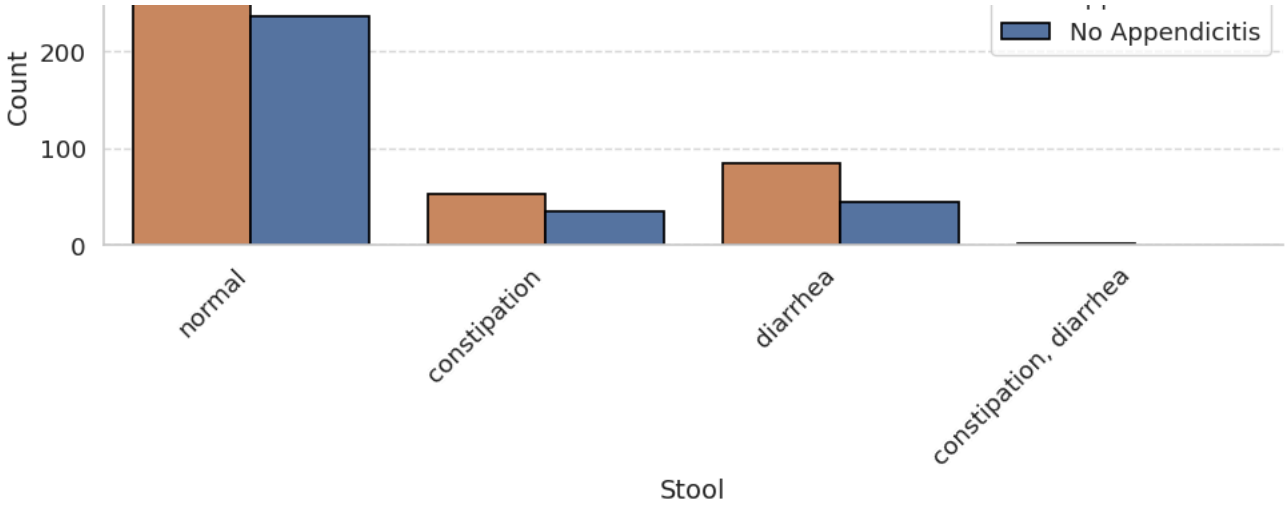

Ketones_in_Urine vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
175
150
125
100
75
50
25
0
no
×

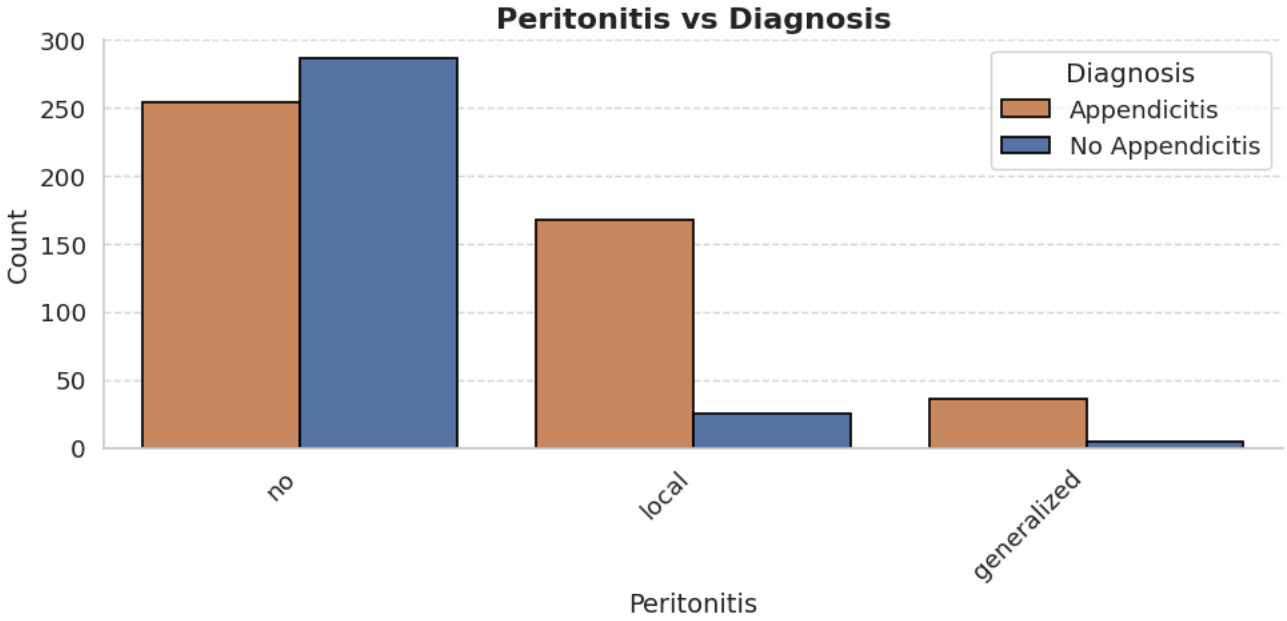

Ketones_in_Urine
RBC_in_Urine vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
0
50
100
150
200
no
RBC_in_Urine

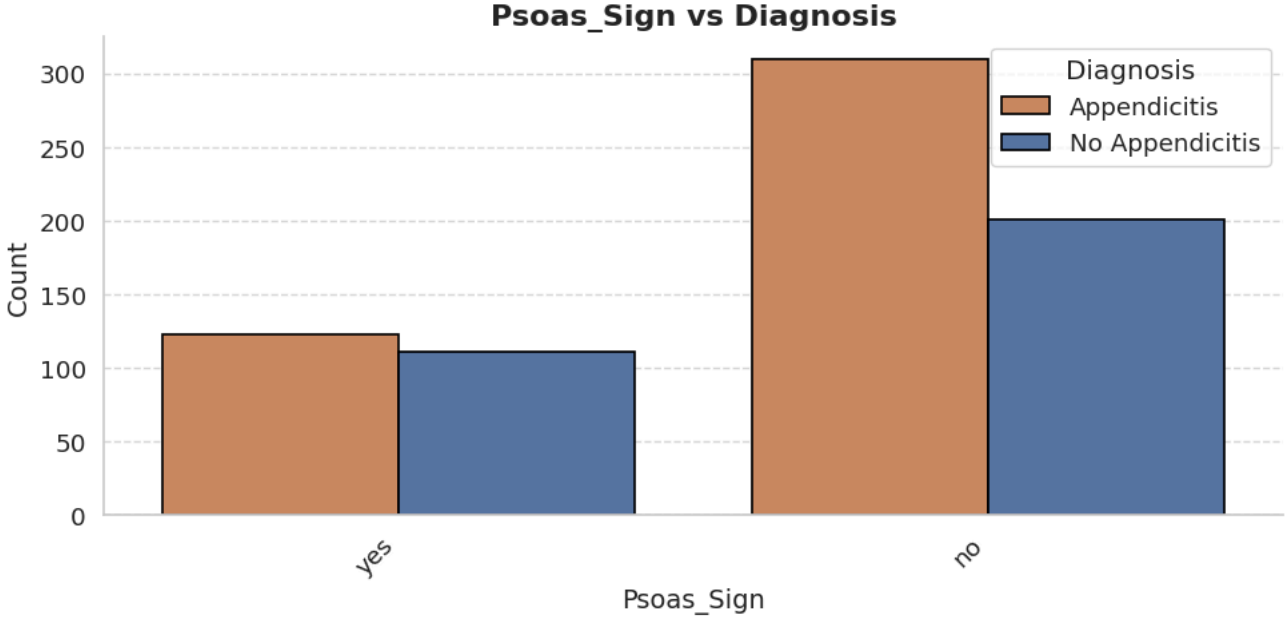

WBC_in_Urine vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
0
50
100
150
200
250
no
WBC_in_Urine

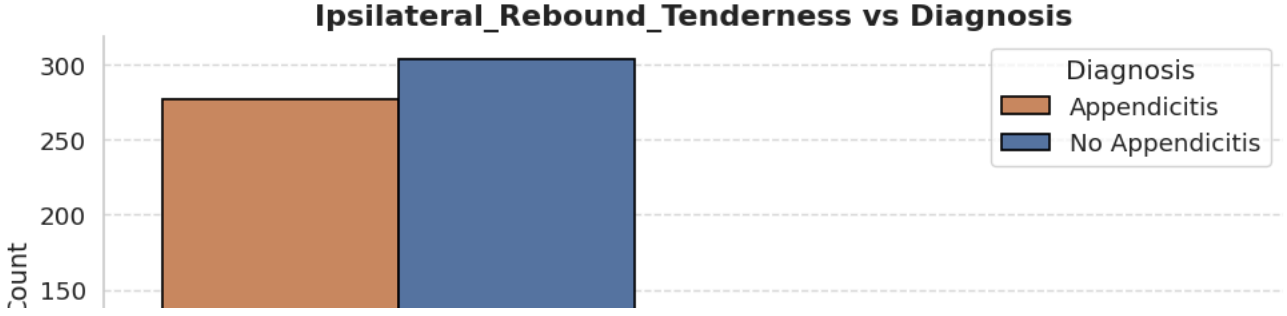

Dysuria vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
0
50
100
150
200
250
300
350
400
no
yes
Dysuria

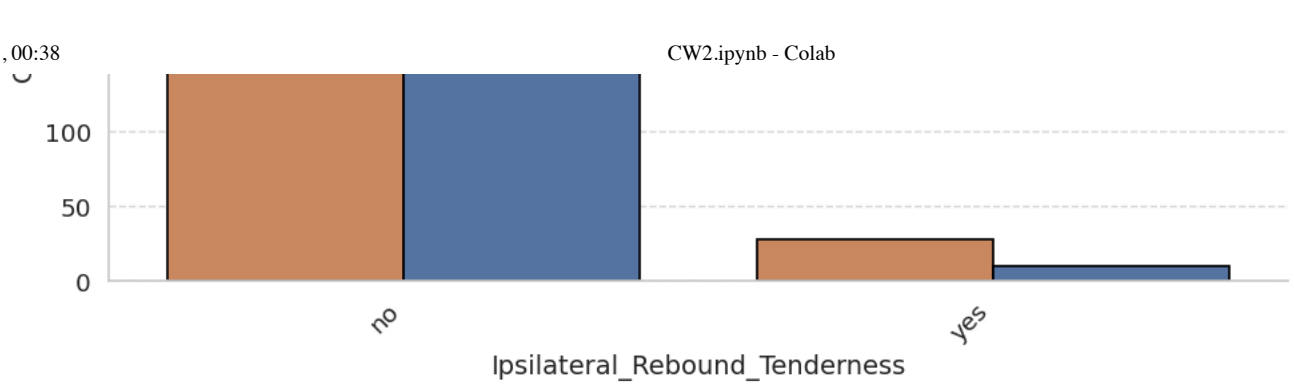

Stool vs Diagnosis
Diagnosis
Appendicitis
300

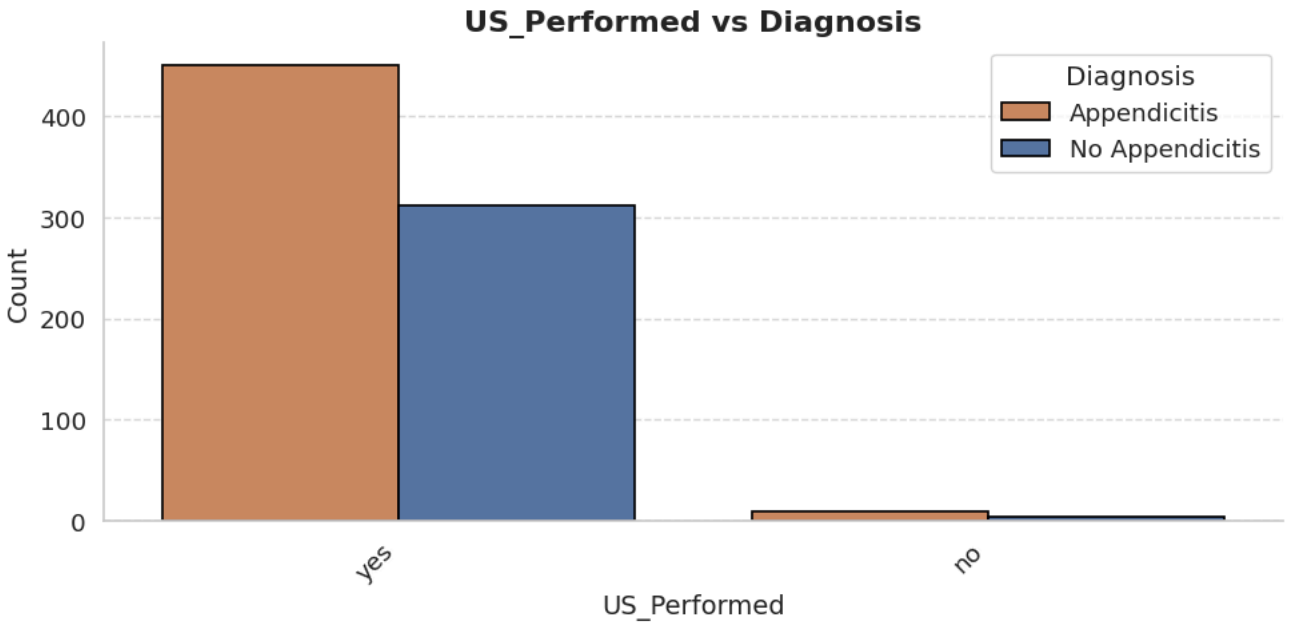

No Appendicitis
Count
200
100
0
normal
constipation
diarrhea
constipation, diarrhea
Stool

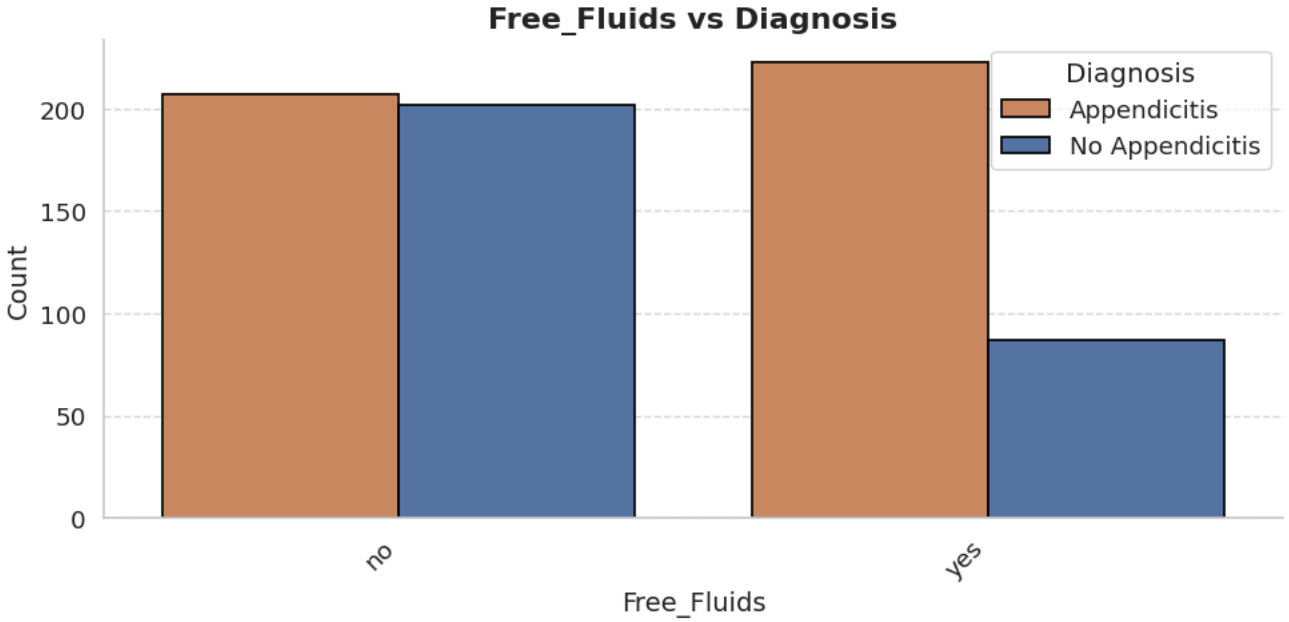

Peritonitis vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
300
250
200
150
100
50
0
no
local
generalized
Peritonitis


Psoas_Sign vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
300
250
200
150
100
50
0
yes
no
Psoas_Sign


Ipsilateral_Rebound_Tenderness vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
300
250
200
150


100
50
0
no
yes
Ipsilateral_Rebound_Tenderness

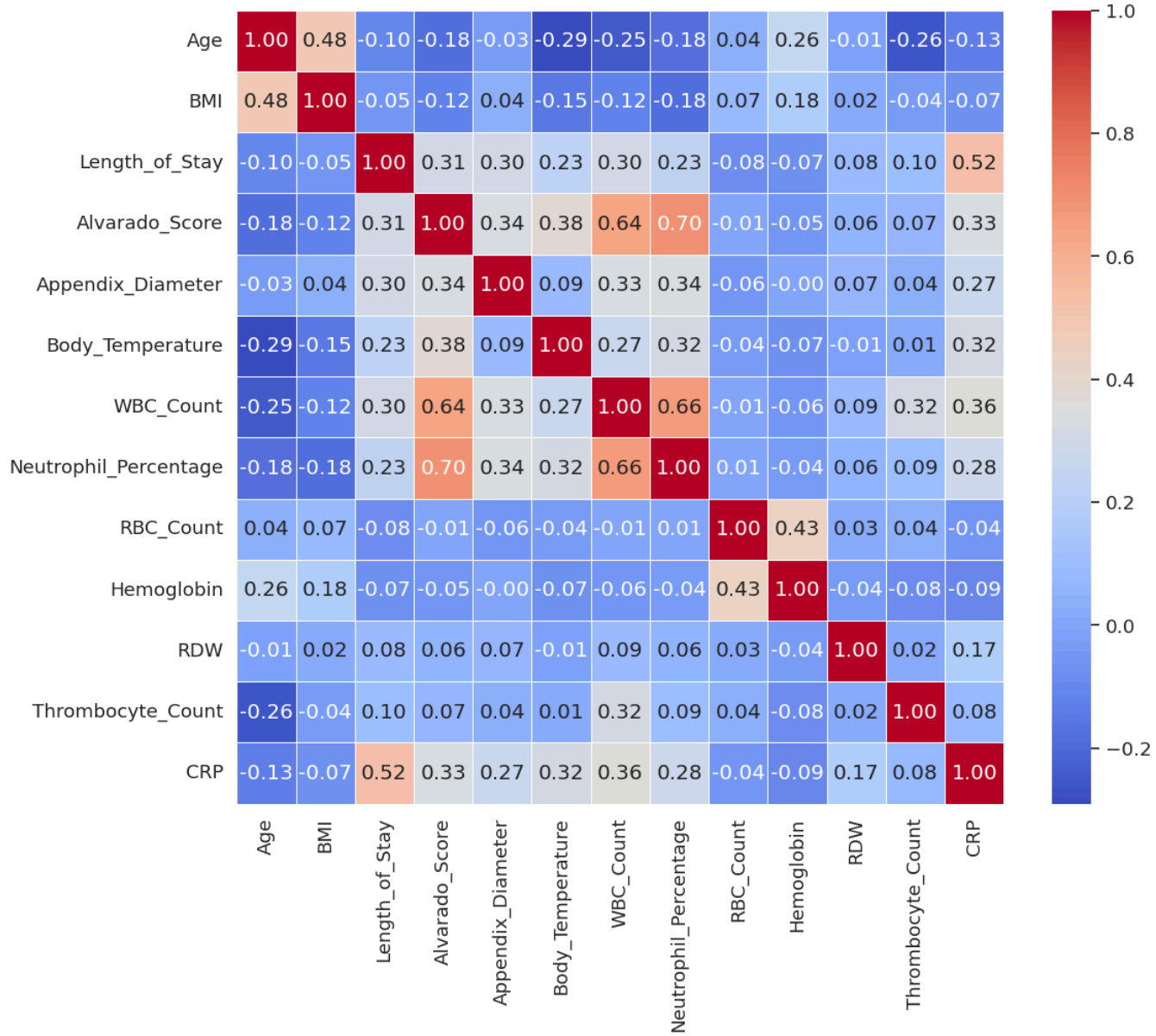

US_Performed vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
400
300
200
100
0
yes
no
US_Performed


Free_Fluids vs Diagnosis
Diagnosis
Appendicitis
No Appendicitis
Count
200
150
100
50
0
no
yes
Free_Fluids

The "Appendix_on_US" (detectibility of appendix on ultrasound) variable stands out in the graphs, as cases where the appendix was visible were more frequently associated with appendicitis diagnoses (orange bar), while cases of the appendix not being visible on the ultrasound was more often linked to the absence of appendicitis (blue bar).

```
corr_matrix = num_x .corr().abs()  # Get absolute value of correlations
upper_tri = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))  # Upper triangle matrix of correlatior

# Find features with correlation above the threshold
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.8)]
# Drop the selected features
x_reduced = num_x.drop(columns=to_drop)

print("Removed Variables:", to_drop)

f,ax = plt.subplots(figsize=(12, 10))
sns.heatmap(x_reduced.corr(), annot=True, linewidths=.5, cmap='coolwarm',fmt= '.2f',ax=ax)
```

```
Removed Variables: ['Height', 'Weight', 'Paedriatic_Appendicitis_Score']
<Axes: >
```



Removed the following variables as they had a correlation >.8 with anohter variable ('Height', 'Weight', 'Paedriatic_Appendicitis_Score'). After removing variables with pairwise correlations above 0.8, the reduced set of features was used specifically for logistic regression, as it is sensitive to multicollinearity. For the other machine learning models, the full set of features was retained, since algorithms like random forests, and boosting methods are not impacted in the same way by highly correlated predictors.

```
X_REDUCED = pd.concat([x_reduced, X.select_dtypes(include=['object', 'category'])], axis=1)
X_REDUCED
```

	Age	BMI	Length_of_Stay	Alvarado_Score	Appendix_Diameter	Body_Temperature	WBC_Count	Neutrophil_Percentage	RB
0	12.68	16.90	3.0	4.0	7.1	37.0	7.7	68.2	
1	14.10	31.90	2.0	5.0	NaN	36.9	8.1	64.8	
2	14.14	23.30	4.0	5.0	NaN	36.6	13.2	74.8	
3	16.37	20.60	3.0	7.0	NaN	36.0	11.4	63.0	
4	11.08	16.90	3.0	5.0	7.0	36.9	8.1	44.0	
...	...	...	...	...	...	...	...	...	
777	12.41	25.25	4.0	8.0	7.5	39.4	11.4	76.6	
778	17.09	20.43	6.0	5.0	NaN	37.8	17.4	89.2	
779	14.99	19.91	4.0	5.0	NaN	37.3	14.6	68.5	
780	7.20	14.30	5.0	9.0	14.0	37.5	17.8	77.0	
781	11.51	18.17	4.0	2.0	8.0	36.8	9.3	70.0	

782 rows × 32 columns

1.3 Outcome Variables

```
y_full
```

	Management	Severity	Diagnosis
0	conservative	uncomplicated	appendicitis
1	conservative	uncomplicated	no appendicitis
2	conservative	uncomplicated	no appendicitis
3	conservative	uncomplicated	no appendicitis
4	conservative	uncomplicated	appendicitis
...	...	...	...
777	primary surgical	uncomplicated	appendicitis
778	secondary surgical	complicated	appendicitis
779	primary surgical	uncomplicated	appendicitis
780	primary surgical	uncomplicated	appendicitis
781	primary surgical	uncomplicated	appendicitis

782 rows × 3 columns

Next steps: Generate code with y_full | View recommended plots | New interactive sheet

```
y_full.describe().T
```

	count	unique	top	freq
Management	781	4	conservative	483
Severity	781	2	uncomplicated	662
Diagnosis	780	2	appendicitis	463

Diagnosis missing one value, therefore removed that row from the dataset.

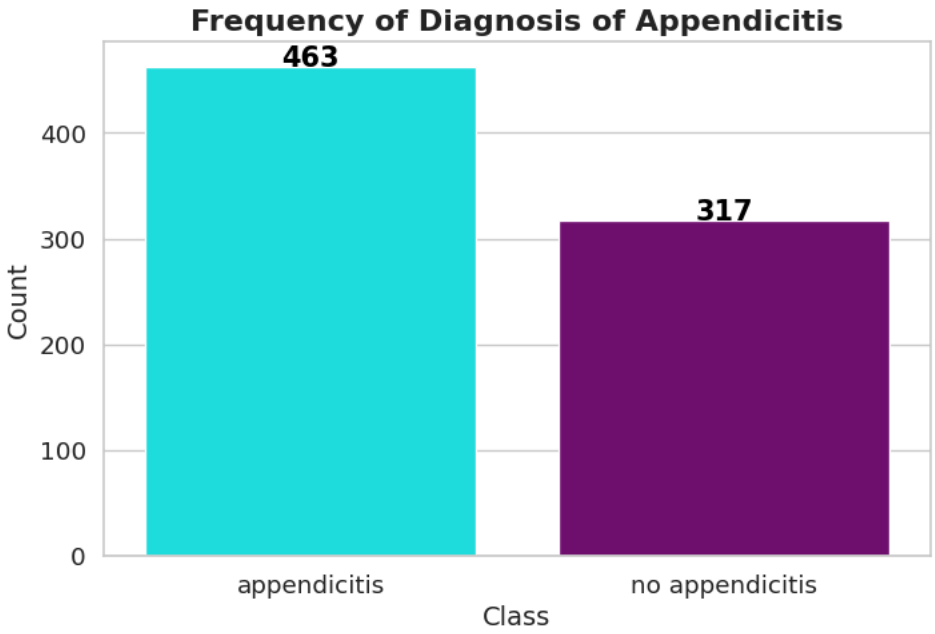
```
valid_idx = y.notnull()

# Filter both X and y
X = X[valid_idx]
y = y[valid_idx]
y_full = y_full[valid_idx]
X_REDUCED = X_REDUCED[valid_idx]
```

Imbalanced classification

Diagnosis

```
class_distribution(y, target_name="Diagnosis of Appendicitis")
```

```
Number of appendicitis: 463 ||| Proportion: 59.4%
Number of no appendicitis: 317 ||| Proportion: 40.6%
```

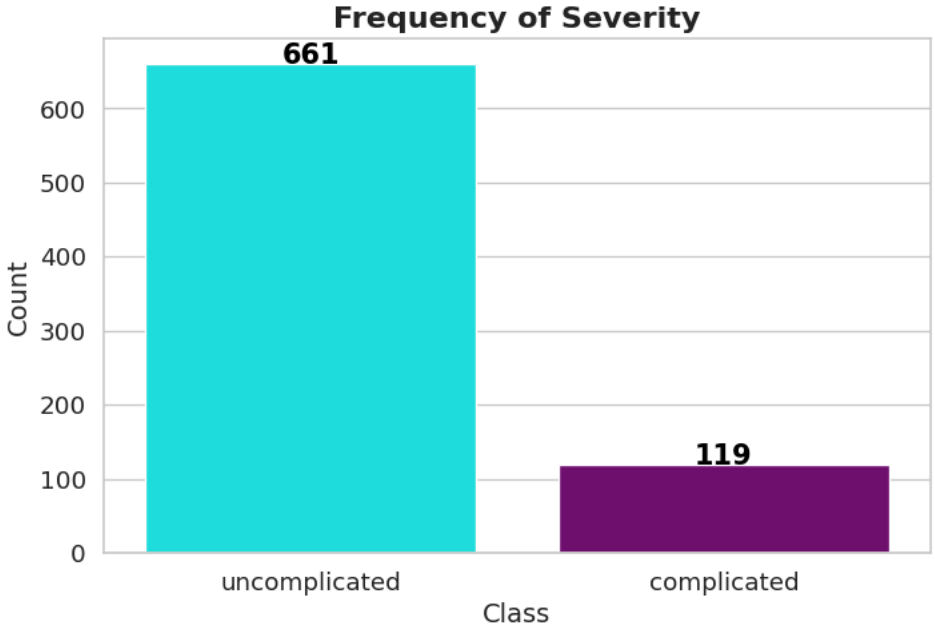


- The diagnosis has a moderate class imbalance, with 463 cases of appendicitis and 317 non-appendicitis cases (approximately 59% vs. 41%).
- This imbalance can bias classification models toward the majority class, resulting in misleadingly high accuracy that fails to reflect performance on the minority class.
- In medical diagnosis, such bias is critical, as false negatives (missed diagnoses) can have serious clinical consequences.
- Accuracy is therefore insufficient as a standalone metric in this setting.
- Class-sensitive metrics such as **precision**, **recall**, and the **F1-score** are more appropriate for evaluating performance.
- The **F1-score**, which balances precision and recall, is particularly suitable when both false positives and false negatives are costly.

Severity

```
class_distribution(y_full["Severity"], target_name="Severity")
```

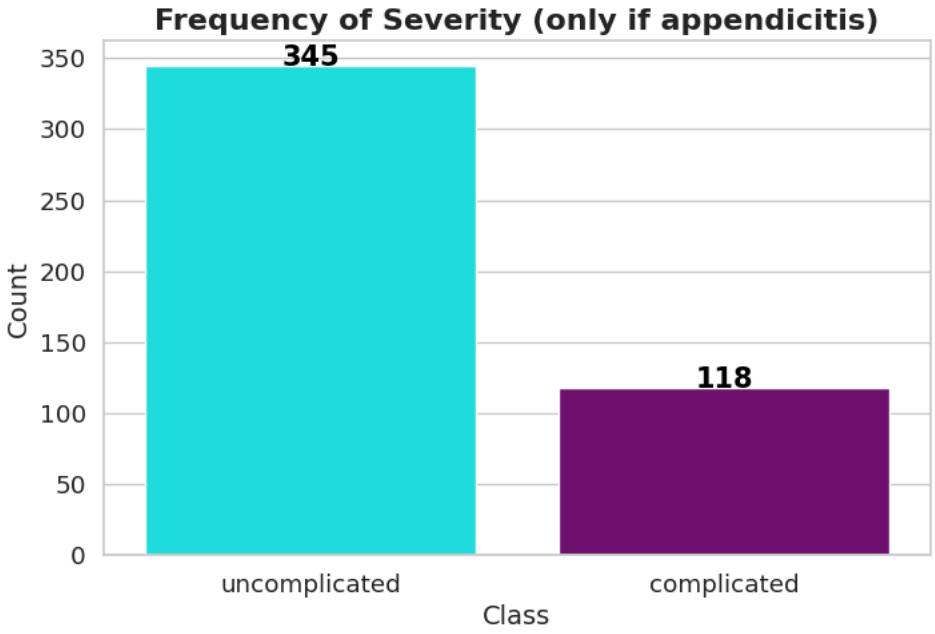
```
Number of uncomplicated: 661 ||| Proportion: 84.7%
Number of complicated: 119 ||| Proportion: 15.3%
```



To accurately analyse severity, only patients diagnosed with appendicitis were included, as those without appendicitis were automatically labelled as uncomplicated; the data were therefore sampled accordingly to reflect true severity classification between uncomplicated and complicated appendicitis.

```
has_appendicitis = y_full["Diagnosis"] == 'appendicitis'
class_distribution(y_full.loc[has_appendicitis, "Severity"], target_name="Severity (only if person has appendicitis)")
```

```
Number of uncomplicated: 345 ||| Proportion: 74.5%
Number of complicated: 118 ||| Proportion: 25.5%
```



The severity classification is also imbalanced (74.5% uncomplicated vs. 25.5% complicated). This imbalance is more extreme and so the need to prioritise metrics like the F1-score over accuracy is even greater in this case to ensure reliable performance across both classes.

2. METHODOLOGY

2.1 Machine learning Models

In this project, a range of machine learning models were evaluated for the tasks of diagnosing appendicitis and predicting severity. Models were selected based on their historical success in healthcare applications, their interpretability, and their ability to manage tabular clinical data with mixed feature types.

To ensure appropriate model deployment, implementation was guided by both academic literature and practical resources. Key references include:

- Group sessions and lab-based feedback
- Aurelien Géron - *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*
- Max Kuhn, Kjell Johnson - *Applied Predictive Modeling*
- Frank E. Harrell, Jr - *Regression Modeling Strategies*
- Kevin P. Murphy - *Machine Learning: A Probabilistic Perspective*
- Gareth James, Daniela Witten, Trevor Hastie - *An Introduction to Statistical Learning*
- Ewout W. Steyerberg - *Clinical Prediction Models*

Each model was developed using a structured workflow, including preprocessing, feature selection, model tuning, and evaluation through metrics suitable for imbalanced medical data (F1-score, precision, recall). The methodology prioritised reproducibility and practical relevance in a clinical context.

Statistical Models

Logistic Regression

Logistic regression is a simple baseline for binary classification, especially valued in clinical settings for its clear interpretability. It models the probability of appendicitis using a weighted combination of predictors, allowing straightforward interpretation of each feature's contribution. However, it assumes linear relationships and can struggle with complex patterns or interactions.

$$P(y = 1 \mid X) = \frac{1}{1 + \exp\left(-(\beta_0 + \sum_{i=1}^{n} \beta_i X_i)\right)}$$

Elastic Net Regularisation

Elastic Net improves logistic regression by combining L1 and L2 penalties to reduce overfitting and handle correlated predictors. It encourages simpler models while retaining important features.

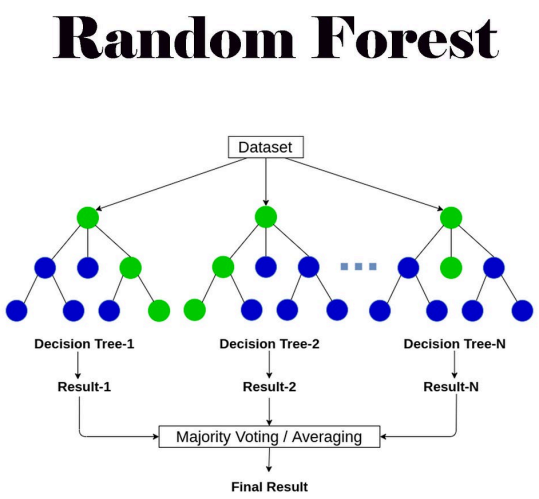
$$L_{\text{class}} = -\sum_{i=1}^{N} \left[y_i \log(p_i) + (1 - y_i) \log(1 - p_i)\right] + \lambda_1 \sum_{j=1}^{n} |\beta_j| + \lambda_2 \sum_{j=1}^{n} \beta_j^2$$

The mixing parameter $\alpha \in [0, 1]$ controls the balance between L1 and L2. Elastic Net is used in this project for both tasks to penalise the use of unnecessary parameters and reduce overfitting.

Ensemble Models

Random Forest

Random Forest combines multiple decision trees to improve predictive stability and reduce overfitting. It handles non-linear relationships and noisy features well, making it suitable for complex clinical data. However, it can be less interpretable than simpler models.



XGBoost

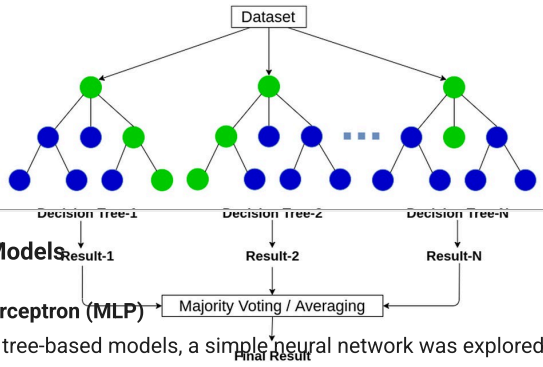
XGBoost is a powerful gradient boosting algorithm that builds decision trees sequentially, with each new tree learning to correct the errors of the previous ones. Boosting is particularly valuable in tabular clinical data where capturing subtle patterns can significantly improve predictive performance. XGBoost internally handles missing values and applies regularisation to prevent overfitting, making it highly effective for structured data. Achieving optimal performance required careful tuning of several hyperparameters, including learning rate, maximum tree depth, and regularisation terms. The main drawbacks are reduced interpretability compared to simpler models and the increased computational cost associated with the amount of hyperparameter to be tuned.

Ensemble Models

Random Forest

Random Forest combines multiple decision trees to improve predictive stability and reduce overfitting. It handles non-linear relationships and noisy features well, making it suitable for complex clinical data. However, it can be less interpretable than simpler models.

Random Forest



Advanced Models

Multilayer Perceptron (MLP)

In addition to tree-based models, a simple neural network was explored to capture potential non-linear patterns in the data. A Multilayer Perceptron (MLP) can model complex feature interactions but typically requires careful regularisation and tuning to prevent overfitting.

Model Structure

- Implemented in PyTorch using `nn.Sequential`.
- Architecture:
  - Two hidden layers with ReLU activations.
  - Dropout applied between layers for regularisation.
  - Output layer with a single neuron for binary classification.

Training and Validation

- Binary cross-entropy loss (`BCEWithLogitsLoss`) was used.
- The optimiser was Adam, with a batch size of 32.
- Early stopping was applied based on validation loss to mitigate overfitting.
- Preprocessing (imputation, scaling, encoding) was performed separately for each train-test split to avoid data leakage.

Evaluation

- Final performance metrics (F1-Score, Precision, Recall, Accuracy, AUC) were reported as averages across Monte Carlo cross-validation splits.
- The MLP was compared against strong baselines such as XGBoost and Random Forest.

XGBoost

XGBoost is a powerful gradient boosting algorithm that builds decision trees sequentially, with each new tree learning to correct the errors of the previous ones. Boosting is particularly valuable in tabular clinical data where capturing subtle patterns can significantly improve predictive performance. XGBoost internally handles missing values and applies regularisation to prevent overfitting, making it highly effective for structured data. Achieving optimal performance required careful tuning of several hyperparameters, including learning rate, maximum tree depth, and regularisation terms. The main drawbacks are reduced interpretability compared to simpler models and the increased computational cost associated with the amount of hyperparameter to be tuned.

Each model was evaluated using Monte Carlo cross-validation, optimising for F1-Score for diagnosis and severity.

2.2 Classification Metrics

- **Accuracy**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

  Measures overall correctness, but can be misleading with imbalanced classes.

- **Precision**

$$\text{Precision} = \frac{TP}{TP + FP}$$

  Reflects the proportion of positive predictions that are actually correct.

- **Recall (Sensitivity)**

$$\text{Recall} = \frac{TP}{TP + FN}$$

  Measures the model's ability to identify all actual positives. In medical screening, high recall helps avoid missing true cases.

- **F1 Score**

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

  Balances precision and recall, making it especially useful when false positives and false negatives are both costly.

- **ROC Curve (Receiver Operating Characteristic)**
  Plots the **True Positive Rate (TPR)** against the **False Positive Rate (FPR)** across thresholds:

$$\text{TPR} = \frac{TP}{TP + FN}, \quad \text{FPR} = \frac{FP}{FP + TN}$$

  A perfect model reaches the top-left corner. This project used the **"top-left method"** to select the threshold closest to ideal performance at (0, 1).

- **AUC (Area Under the ROC Curve)**
  Represents the likelihood that a randomly chosen positive case is ranked above a negative one.
- **Mean Precision-Recall Curve**
  Plots mean precision against mean recall across thresholds to highlight trade-offs in imbalanced datasets.

In this setting, a true positive (TP) refers to correctly identifying a patient with appendicitis. Since both missed diagnoses and unnecessary surgeries carry clinical risks, the F1-score is prioritised to balance sensitivity (recall) and precision, ensuring the model performs well across both types of error.

2.3 Monte Carlo Cross-Validation (MC CV)

> **Important Note:**
> A generalisable Python function developed in the first coursework has been adapted and reused in this project. The function supports binary classification tasks by performing Monte Carlo cross-validation (MCCV), hyperparameter tuning, and automated visualisation for any selected model. Building on this, additional improvements were made for the current project. Reusing and this function provided efficiency, aligning with the goal of creating reusable tools for future machine learning work.

Monte Carlo Cross-Validation (MC CV) is a repeated random subsampling technique used to evaluate model performance. In contrast to k-fold cross-validation, which systematically partitions the dataset into equal folds, MC CV generates multiple random train-test splits, offering increased flexibility and better insights into performance variability.

In this project, an initial hyperparameter search was conducted with a lower number of Monte Carlo iterations ($M = 20$) to efficiently identify promising parameter grids. Once suitable parameters were selected, the number of repetitions was expanded to $M = 200$ to obtain more reliable performance estimates without significantly increasing computational demands.

Each MC CV iteration followed the structured pipeline outlined below, with Steps 1-7 repeated for each randomly shuffled split:

1. **Random Stratified Split**: The dataset was randomly divided into an 80% training set and a 20% testing set, maintaining class balance for classification tasks. (Outer for-loop)
2. **Numerical Imputation**: Missing values in numerical features were imputed using a K-Nearest Neighbors (KNN) approach to preserve data integrity. (k=5)
3. **Encoding of Categorical Features**: Categorical variables were transformed into numerical format using one-hot encoding, enabling compatibility with the learning algorithms.
4. **Feature Normalisation**: Numerical features were standardised using z-score normalisation to ensure a consistent scale across inputs.
5. **Hyperparameter Search**: For models requiring tuning, a grid search within each iteration was performed to optimise parameters based on validation F1-score. (Inner for-loop)
6. **Model Training and Testing**: Models were trained on the training set and evaluated on the test set for each iteration.
7. **Performance Evaluation**: Key metrics, including Accuracy, Precision, Recall, F1-score, and AUC, were computed. The results were aggregated by reporting the mean and standard deviation across all M iterations.
8. **Visualisation of Results**: Aggregated performance graphs were produced, including mean ROC curves with 95% confidence intervals, mean Precision-Recall curves, and averaged confusion matrices.

MC CV Metric Aggregation Formulas

- **Accuracy**:

$$MC\text{-}CV_{\text{test}} = \frac{1}{M} \sum_{m=1}^{M} \text{Accuracy}_{m,\text{test}}$$

- **Precision, Recall, F1-score**:

$$MC\text{-}CV_{\text{test}} = \frac{1}{M} \sum_{m=1}^{M} \text{Metric}_{m,\text{test}}$$

  Where Metric refers to either Precision, Recall, or F1-score.

This methodology provides stable and reliable performance estimates by incorporating variability across multiple random splits. Due to class imbalance considerations, F1-score was prioritised during hyperparameter optimisation.
The figure below illustrates the Monte Carlo Cross-Validation process implemented in the project's evaluation framework. Models with hyperparameters were automatically tuned via internal grid search at each iteration.

2.4 Gradio Deployment

The final models for diagnosis and severity were deployed using the Gradio library to create an interactive web interface. Gradio allows users to input clinical features and receive real-time predictions along with class probabilities, enabling accessible model testing and demonstration without requiring backend development. This setup supports both clinical exploration and future integration into healthcare workflows.

3. MODEL TRAINING & EVALUATION

Encoding

> 'appendicitis' : 1
> 'no appendicitis' : 0

```
def encode_binary_labels(y):
    if isinstance(y, (pd.Series, np.ndarray)):
        unique_labels = np.unique(y)
        if "appendicitis" in unique_labels and "no appendicitis" in unique_labels:
            print("🔄 Converting 'appendicitis' to 1 and 'no appendicitis' to 0...")
            return pd.Series(y).replace({"appendicitis": 1, "no appendicitis": 0})
        elif sorted(unique_labels) == [0, 1]:
            print("✔ Labels are already correctly encoded (0=No Appendicitis, 1=Appendicitis).")
            return pd.Series(y)
        else:
            print(f"⚠ Warning: Unexpected labels {unique_labels}. Please check!")
            return pd.Series(y)
    else:
        raise TypeError("Input must be a pandas Series or numpy array.")
```

```
y = encode_binary_labels(y)
```

```
🔁 Converting 'appendicitis' to 1 and 'no appendicitis' to 0...
```

```
df = pd.concat([X, y], axis=1)
```

3.1 Baseline Accuracy

```
# Most frequent class
most_common = y.value_counts().idxmax()

# Baseline accuracy: how often you'd be right if you always guessed that class
baseline_accuracy = (y == most_common).mean()

print(f"Always predicting Appendicitis: {baseline_accuracy:.3f}")
```

```
Always predicting Appendicitis: 0.594
```

Slight class imbalance. This means, other metrics like precision, recall and F1 score will be a better metric to improve our model's on.

3.2 Monte Carlo Cross-Validation Function

Function for monte carlo cross-validation. See more in methodology for explanation.

```
# Storing results
model_results = {}       # trained models & results
monte_carlo_results = {} # all iteration results for each model
confusion_matrices = {}  # confusion matrices for each iteration

feature_importance_dict = defaultdict(list)
pr_curves = defaultdict(list)
roc_curves = defaultdict(list)
best_hyperparams_counter = defaultdict(int)  # Tracks best hyperparameters

# Categorical columns
cat_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()
# Numerical columns
num_cols = X.select_dtypes(exclude=['object', 'category']).columns.tolist()


def monte_carlo_cv_with_plot(model, param_grid, X, y, model_name, num_iterations=100, test_size=0.2, n_neighbors=5, cv_folds
    scaler = StandardScaler()
    results = []
    best_models = []
    validation_results = {}

    best_overall_model = None
    best_f1_score = 0
    cm_sum = np.zeros((2, 2))

    all_pr_curves = []
    best_thresholds = []
    roc_distances = []

    for i in trange(num_iterations, desc=f"Monte Carlo CV - {model_name}"):
        if num_iterations <= 20 or (i + 1) % (num_iterations // 4) == 0 or i == 0:
            print(f"\n Monte Carlo Iteration {i+1}/{num_iterations}...")

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

        knn_imputer = KNNImputer(n_neighbors=n_neighbors)
        X_train[num_cols] = knn_imputer.fit_transform(X_train[num_cols])
        X_test[num_cols] = knn_imputer.transform(X_test[num_cols])

        preprocessor = ColumnTransformer([
            ("num", StandardScaler(), num_cols),
            ("cat", OneHotEncoder(drop="first", handle_unknown="ignore"), cat_cols)
        ])

        X_train_processed = preprocessor.fit_transform(X_train)
        X_test_processed = preprocessor.transform(X_test)
```

```
        feature_names = preprocessor.get_feature_names_out()
        X_train = pd.DataFrame(X_train_processed, columns=feature_names)
        X_test = pd.DataFrame(X_test_processed, columns=feature_names)

        if param_grid and len(param_grid) >= 1:
            if num_iterations <= 20 or (i + 1) % (num_iterations // 4) == 0 or i == 0:
                print(f"\n Hyperparameter tuning for {model_name} (Optimizing F1-Score)...")

            grid_search = GridSearchCV(model, param_grid, cv=cv_folds, scoring="f1", n_jobs=-1, return_train_score=True)
            grid_search.fit(X_train, y_train)
            best_model = grid_search.best_estimator_
            best_params = frozenset(grid_search.best_params_.items())
            best_hyperparams_counter[best_params] += 1

            if num_iterations <= 20 or (i + 1) % (num_iterations // 4) == 0 or i == 0:
                print(f" Best Parameters: {grid_search.best_params_}")
                print(f" Best Cross-Validation F1 Score: {grid_search.best_score_:.4f}")

            validation_results = {
                "Model": model,
                "Hyperparameters": grid_search.cv_results_["params"],
                "Validation F1-Score": grid_search.cv_results_["mean_test_score"],
                "Training F1-Score": grid_search.cv_results_["mean_train_score"]
            }
        else:
            if num_iterations <= 20 or (i + 1) % (num_iterations // 4) == 0 or i == 0:
                print(f"\n Training {model.__class__.__name__} with Monte Carlo CV (No Hyperparameter)...")
            model.fit(X_train, y_train)
            best_model = model

        best_models.append(best_model)

        best_model.fit(X_train, y_train)
        y_pred = best_model.predict(X_test)
        y_prob = best_model.predict_proba(X_test)[:, 1] if hasattr(best_model, "predict_proba") else None

        accuracy = accuracy_score(y_test, y_pred)
        auc = roc_auc_score(y_test, y_prob) if y_prob is not None else None
        f1 = f1_score(y_test, y_pred, pos_label=1)
        precision = precision_score(y_test, y_pred, pos_label=1)
        recall = recall_score(y_test, y_pred, pos_label=1)

        cm = confusion_matrix(y_test, y_pred, labels=[1, 0])
        cm_sum += cm

        if f1 > best_f1_score:
            best_f1_score = f1
            best_overall_model = best_model
            best_y_test = y_test
            best_y_probs = y_prob
            best_y_pred = y_pred

        results.append({
            "Iteration": i+1,
            "Accuracy": accuracy, "AUC": auc, "Precision": precision,
            "Recall": recall, "F1-Score": f1
        })

        if y_prob is not None:
            precisions, recalls, thresholds = precision_recall_curve(y_test, y_prob)
            f1_scores = (2 * precisions * recalls) / (precisions + recalls + 1e-6)
            best_threshold = thresholds[np.argmax(f1_scores)]
            best_thresholds.append(best_threshold)
            all_pr_curves.append((precisions, recalls, thresholds))

            if track_feature_importance:
                if hasattr(best_model, "coef_"):
                    feature_importance_dict[model_name].append(best_model.coef_[0])
                elif hasattr(best_model, "feature_importances_"):
                    feature_importance_dict[model_name].append(best_model.feature_importances_)

            fpr, tpr, thresholds = roc_curve(y_test, y_prob)
            roc_curves[model_name].append((fpr, tpr, thresholds))
            distances = np.sqrt(fpr**2 + (1 - tpr)**2)
            roc_distances.append((thresholds[np.argmin(distances)], fpr[np.argmin(distances)], tpr[np.argmin(distances)]))

    monte_carlo_results[model_name] = pd.DataFrame(results)

    results_df = pd.DataFrame(results).drop(columns=["Iteration"])
    mean_metrics = results_df.mean()
    std_metrics = results_df.std()
```

```
        final_results = pd.DataFrame({
            "Mean Accuracy": mean_metrics["Accuracy"], "Std Accuracy": std_metrics["Accuracy"],
            "Mean AUC": mean_metrics["AUC"], "Std AUC": std_metrics["AUC"],
            "Mean Precision": mean_metrics["Precision"], "Std Precision": std_metrics["Precision"],
            "Mean Recall": mean_metrics["Recall"], "Std Recall": std_metrics["Recall"],
            "Mean F1": mean_metrics["F1-Score"], "Std F1": std_metrics["F1-Score"]
        }, index=["Results"])

        print("\n Monte Carlo CV Results")
        display(round(final_results, 3))

        if best_hyperparams_counter:
            print("\n🏆 Most Common Best Hyperparameters:")
            sorted_params = Counter(best_hyperparams_counter).most_common()
            for i, (params, count) in enumerate(sorted_params, 1):
                print(f"{i}. Params: {dict(params)} | Chosen: {count} out of {num_iterations} times")
        best_hyperparams_counter.clear()

        mean_cm = cm_sum / num_iterations
        confusion_matrices[model_name] = mean_cm

        model_results[model_name] = {
            "model": best_overall_model,
            "y_test": best_y_test,
            "y_probs": best_y_probs,
            "y_pred": best_y_pred
        }

        plot_precision_recall_tradeoff_with_ci(all_pr_curves, best_thresholds, model_name)
        plot_avg_roc_with_ci(model_name, roc_distances)
        #plot_calibration_curve(model_name)
        plot_mean_confusion_matrix(model_name)

        if param_grid and len(param_grid) == 1:
            plot_validation_trends(validation_results)

        return final_results, best_overall_model


    ## Function for ROC curve
    def plot_avg_roc_with_ci(model_name, roc_distances, force_origin=True):
        curves = roc_curves[model_name]
        all_interp = []
        fpr_grid = np.linspace(0, 1, 100)
        for fpr, tpr, _ in curves:
            interp = np.interp(fpr_grid, fpr, tpr, left=0, right=1)
            all_interp.append(interp)

        all_interp = np.array(all_interp)
        mean_tpr = np.mean(all_interp, axis=0)
        with warnings.catch_warnings():
          warnings.simplefilter("ignore", category=RuntimeWarning)
          ci_lower, ci_upper = st.t.interval(0.95, len(all_interp)-1, loc=mean_tpr, scale=st.sem(all_interp, axis=0))

        if force_origin:
            fpr_grid = np.insert(fpr_grid, 0, 0.0)
            mean_tpr = np.insert(mean_tpr, 0, 0.0)
            ci_lower = np.insert(ci_lower, 0, 0.0)
            ci_upper = np.insert(ci_upper, 0, 0.0)

        plt.figure(figsize=(8, 6))
        plt.plot(fpr_grid, mean_tpr, label="Mean ROC Curve", color="darkorange")
        plt.fill_between(fpr_grid, ci_lower, ci_upper, color="orange", alpha=0.2, label="95% CI")
        plt.plot([0, 1], [0, 1], linestyle="--", color="gray")

        if roc_distances:
            top_lefts = np.array([[fpr, tpr] for _, fpr, tpr in roc_distances])
            avg_top_left = top_lefts.mean(axis=0)
            ci_top_left = st.t.interval(0.95, len(top_lefts)-1, loc=avg_top_left, scale=st.sem(top_lefts, axis=0))

            plt.errorbar(
                x=avg_top_left[0],
                y=avg_top_left[1],
                xerr=[[avg_top_left[0] - ci_top_left[0][0]], [ci_top_left[1][0] - avg_top_left[0]]],
                yerr=[[avg_top_left[1] - ci_top_left[0][1]], [ci_top_left[1][1] - avg_top_left[1]]],
                fmt='o', color='green', ecolor='darkgreen', capsize=5, label="Top-Left Threshold ± 95% CI"
            )

        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.title(f"Mean ROC Curve - {model_name}")
        plt.legend()
```

```
        plt.grid(True)
        plt.show()

        if roc_distances:
            best_thresholds = [t for t, _, _ in roc_distances]
            mean_threshold = np.mean(best_thresholds)
            ci_threshold = st.t.interval(0.95, len(best_thresholds)-1, loc=mean_threshold, scale=st.sem(best_thresholds))
            print(f"\n Mean Top-Left Threshold (ROC): {mean_threshold:.2f} ± {(ci_threshold[1] - ci_threshold[0])/2:.2f} (95% CI

    ## Function for precision recall tradeoff
    def plot_precision_recall_tradeoff_with_ci(all_pr_curves, best_thresholds, model_name):
        common_thresholds = np.linspace(0, 1, 100)
        precisions_interp = []
        recalls_interp = []

        for precisions, recalls, thresholds in all_pr_curves:
            recall_interp = np.interp(common_thresholds, thresholds, recalls[:-1], left=recalls[0], right=recalls[-1])
            precision_interp = np.interp(common_thresholds, thresholds, precisions[:-1], left=precisions[0], right=precisions[-1
            recalls_interp.append(recall_interp)
            precisions_interp.append(precision_interp)

        precisions_interp = np.array(precisions_interp)
        recalls_interp = np.array(recalls_interp)

        mean_prec = np.mean(precisions_interp, axis=0)
        mean_rec = np.mean(recalls_interp, axis=0)
        with warnings.catch_warnings():
          warnings.simplefilter("ignore", category=RuntimeWarning)
          ci_prec = st.t.interval(0.95, len(precisions_interp)-1, loc=mean_prec, scale=st.sem(precisions_interp, axis=0))
          ci_rec = st.t.interval(0.95, len(recalls_interp)-1, loc=mean_rec, scale=st.sem(recalls_interp, axis=0))

        plt.figure(figsize=(10, 6))
        plt.plot(common_thresholds, mean_prec, label="Precision", color="cyan")
        plt.fill_between(common_thresholds, ci_prec[0], ci_prec[1], color="cyan", alpha=0.2)
        plt.plot(common_thresholds, mean_rec, label="Recall", color="purple")
        plt.fill_between(common_thresholds, ci_rec[0], ci_rec[1], color="purple", alpha=0.2)

        best_mean = np.mean(best_thresholds)
        best_ci = st.t.interval(0.95, len(best_thresholds)-1, loc=best_mean, scale=st.sem(best_thresholds))
        plt.axvline(best_mean, color="green", linestyle="--", label=f"Best F1 Cutoff: {best_mean:.2f} ± {(best_ci[1] - best_ci[0
        plt.fill_betweenx([0, 1], best_ci[0], best_ci[1], color="green", alpha=0.15)

        plt.xlabel("Decision Threshold")
        plt.ylabel("Score")
        plt.title(f"Precision-Recall Tradeoff (w/ CI) - {model_name}")
        plt.legend()
        plt.grid(True)
        plt.ylim(0, 1.05)
        plt.tight_layout()
        plt.show()

      # Plot Hyperparameter Tuning Trends (Validation & Training F1 Score)
    def plot_validation_trends(validation_results):
        """
        Plots validation and training F1 Score trends if only one hyperparameter is tuned.
        """
        if not validation_results:
            print("No validation results available.")
            return

        params = validation_results["Hyperparameters"]
        val_f1 = validation_results["Validation F1-Score"]
        train_f1 = validation_results["Training F1-Score"]  # Extract training F1 scores

        param_name = list(params[0].keys())[0]  # Extract hyperparameter name
        param_values = [p[param_name] for p in params]  # Get values

        # Plot Validation & Training F1 Score vs Hyperparameter
        plt.figure(figsize=(10, 6))
        plt.plot(param_values, train_f1, marker='s', linestyle='--', color="purple", label="Training F1 Score")
        plt.plot(param_values, val_f1, marker='o', linestyle='-', color="cyan", label="Validation F1 Score")

        plt.xlabel(f"{param_name} Values", fontsize=12)
        plt.ylabel("F1 Score", fontsize=12)
        plt.title(f"Training vs Validation F1 Score for {param_name}", fontsize=14, fontweight="bold")
        plt.legend(fontsize=12)
        plt.grid(True, linestyle="--", alpha=0.6)
        plt.xticks(fontsize=10)
        plt.yticks(fontsize=10)
        plt.show()

    ## Function for mean confusion matrix
```

```
def plot_mean_confusion_matrix(model_name):
    cm = confusion_matrices[model_name]
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm.T, annot=True, fmt=".2f", cmap="Greens",
                xticklabels=["Appendicitis", "No Appendicitis"], yticklabels=["Appendicitis", "No Appendicitis"])
    plt.xlabel("True Label")
    plt.ylabel("Predicted Label")
    plt.title(f"Mean Confusion Matrix - {model_name}")
    plt.show()
```

> **NOTE: FOR ALL MODELS IF YOU SCROLL TO THE BOTTOM OF THE OUTPUT YOU CAN SEE THE RESULTS AND GRAPHS**

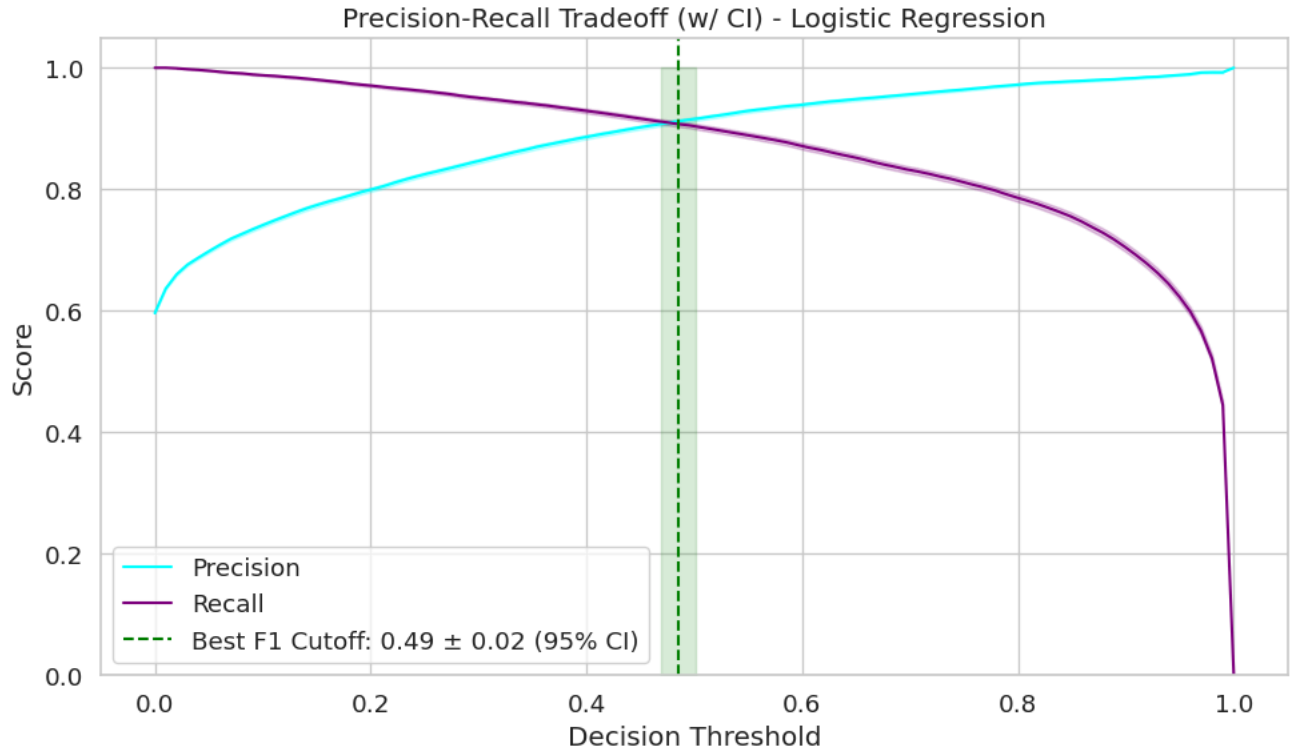
3.3 Logistic Regression

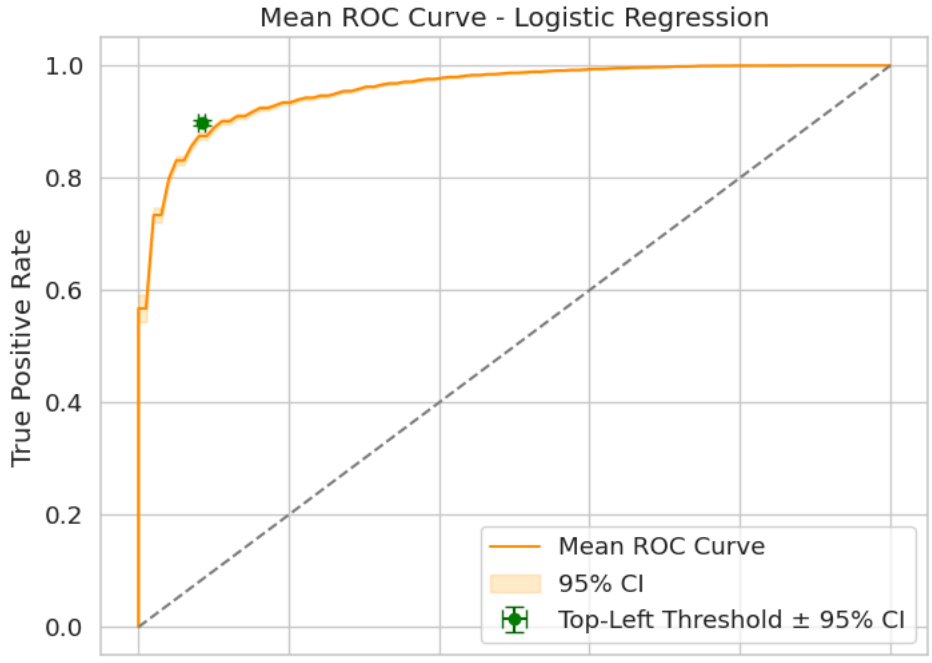
Diagnosis

```
num_cols = X_REDUCED.select_dtypes(exclude=['object', 'category']).columns.tolist()

logreg_model = LogisticRegression(max_iter=10000, random_state=2025)

logreg_results, best_logreg = monte_carlo_cv_with_plot(
    model=logreg_model,
    param_grid=None,  # No tuning
    X=X_REDUCED,
    y=y,
    model_name="Logistic Regression",
    num_iterations=200
)

# Save results
monte_carlo_results["Logistic Regression"].to_csv(f"{save_path_c_f}/logistic_regression_mccv_results.csv", index=False)
logreg_results.to_csv(f"{save_path_c_f}/logistic_regression_summary.csv")
```

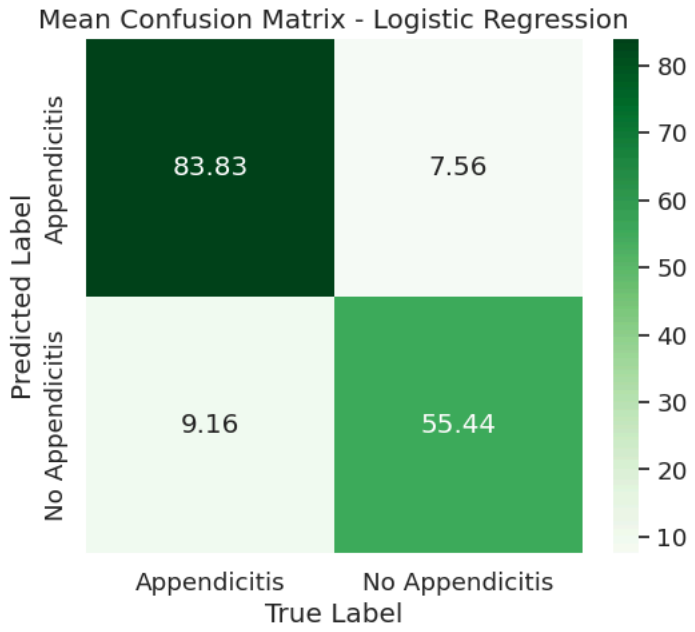
```
Monte Carlo Iteration 1/200...

 Training LogisticRegression with Monte Carlo CV (No Hyperparameter)...
Monte Carlo CV - Logistic Regression:  26%|██▌       | 51/200 [00:06<00:14,  9.95it/s]
 Monte Carlo Iteration 50/200...

 Training LogisticRegression with Monte Carlo CV (No Hyperparameter)...
Monte Carlo CV - Logistic Regression:  50%|████▉     | 100/200 [00:11<00:10,  9.74it/s]
 Monte Carlo Iteration 100/200...

 Training LogisticRegression with Monte Carlo CV (No Hyperparameter)...
Monte Carlo CV - Logistic Regression:  75%|███████▍  | 150/200 [00:18<00:05,  9.50it/s]
 Monte Carlo Iteration 150/200...

 Training LogisticRegression with Monte Carlo CV (No Hyperparameter)...
Monte Carlo CV - Logistic Regression: 100%|█████████| 200/200 [00:23<00:00,  8.41it/s]
 Monte Carlo Iteration 200/200...

 Training LogisticRegression with Monte Carlo CV (No Hyperparameter)...

 Monte Carlo CV Results
```

	Mean Accuracy	Std Accuracy	Mean AUC	Std AUC	Mean Precision	Std Precision	Mean Recall	Std Recall	Mean F1	Std F1
Results	0.893	0.024	0.958	0.013	0.918	0.024	0.901	0.033	0.909	0.021






```
Mean Top-Left Threshold (ROC): 0.56 ± 0.01 (95% CI)
```



- **Features Used:**
  - Reduced predictors from the correlation analysis.
  - `Age`, `BMI`, `Length_of_Stay`, `Alvarado_Score`, `Appendix_Diameter`, `Body_Temperature`, `WBC_Count`, `Neutrophil_Percentage`, `RBC_Count`, `Hemoglobin`, `RDW`, `Thrombocyte_Count`, `CRP`, `Sex`, `Appendix_on_US`, `Migratory_Pain`, `Lower_Right_Abd_Pain`, `Contralateral_Rebound_Tenderness`, `Coughing_Pain`, `Nausea`, `Loss_of_Appetite`, `Neutrophilia`, `Ketones_in_Urine`, `RBC_in_Urine`, `WBC_in_Urine`, `Dysuria`, `Stool`, `Peritonitis`, `Psoas_Sign`, `Ipsilateral_Rebound_Tenderness`, `US_Performed`, `Free_Fluids`
  - These features encompass a broad clinical picture, combining patient demographics, clinical symptoms, laboratory findings, and ultrasound imaging results.
- **Best Hyperparameters:**
  - No hyperparameter tuning was necessary for Logistic Regression.
- **Mean F1 Score (Optimal Threshold = 0.49 ± 0.02):**
  - **F1 Score:** 0.909 ± 0.021
- **Precision-Recall Trade-off Discussion:**
  - The Precision-Recall curve identified an optimal decision threshold around 0.49.
  - Precision and recall were well-balanced near this threshold, with **recall slightly favoured** — a clinically appropriate choice in appendicitis diagnosis where **missing a true case (false negative)** could have serious consequences.
  - The model's behaviour suggests a low rate of missed diagnoses while maintaining a strong ability to rule out patients without appendicitis.
- **Mean ROC Curve and AUC Discussion:**
  - The ROC curve was strong and smooth, achieving a **Mean AUC of 0.958 ± 0.013**, indicating excellent discriminative performance.
  - The Top-Left threshold selected by ROC analysis was close to 0.56, aligning reasonably with the F1-optimal threshold found from Precision-Recall tradeoff.
  - This indicates that the model is not only accurate, but also well-calibrated for clinical use.
- **Other Metrics (at F1-optimal threshold ~0.49):**
  - **Accuracy:** 0.893 ± 0.024
  - **Precision:** 0.918 ± 0.024
  - **Recall:** 0.901 ± 0.033
- **Confusion Matrix Insights:**
  - On average across Monte Carlo iterations:
    - 83.8% of patients with appendicitis were correctly identified.
    - Only 7.6% of patients without appendicitis were incorrectly diagnosed.
    - The false negative rate was kept low (~9%), supporting the model's use in screening setting where sensitivity is critical.

- **Advantages of Logistic Regression:**
  - High interpretability: coefficients provide direct insight into feature importance.
  - Probabilistic output allows for flexible decision threshold adjustment based on clinical risk tolerance.
  - Computationally efficient and easy to deploy in real-time hospital settings.
  - Particularly useful when needing transparency for clinical adoption and auditing.
- **Disadvantages of Logistic Regression:**
  - Assumes a linear relationship between features and log-odds, which may oversimplify complex clinical presentations.
  - May underperform compared to more flexible models when important feature interactions exist.
  - Performance can still be influenced by multicollinearity between laboratory and imaging variables if not properly regularised.

> **Note:**
> The green marker on the ROC curve denotes the mean Top-Left threshold — the average point closest to ideal sensitivity and specificity across all Monte Carlo iterations.
> Threshold optimization was important for balancing sensitivity (recall) and specificity in the appendicitis diagnosis setting, where false negatives carry greater clinical risk.

Severtiy

```
# Select only patients who were diagnosed with appendicitis
appendicitis_patients = y_full["Diagnosis"] == "appendicitis"

# Filter X and y for severity task
X_severity = X[appendicitis_patients]
X_severity_reduced = X_REDUCED[appendicitis_patients]
y_severity = y_full["Severity"][appendicitis_patients]
```

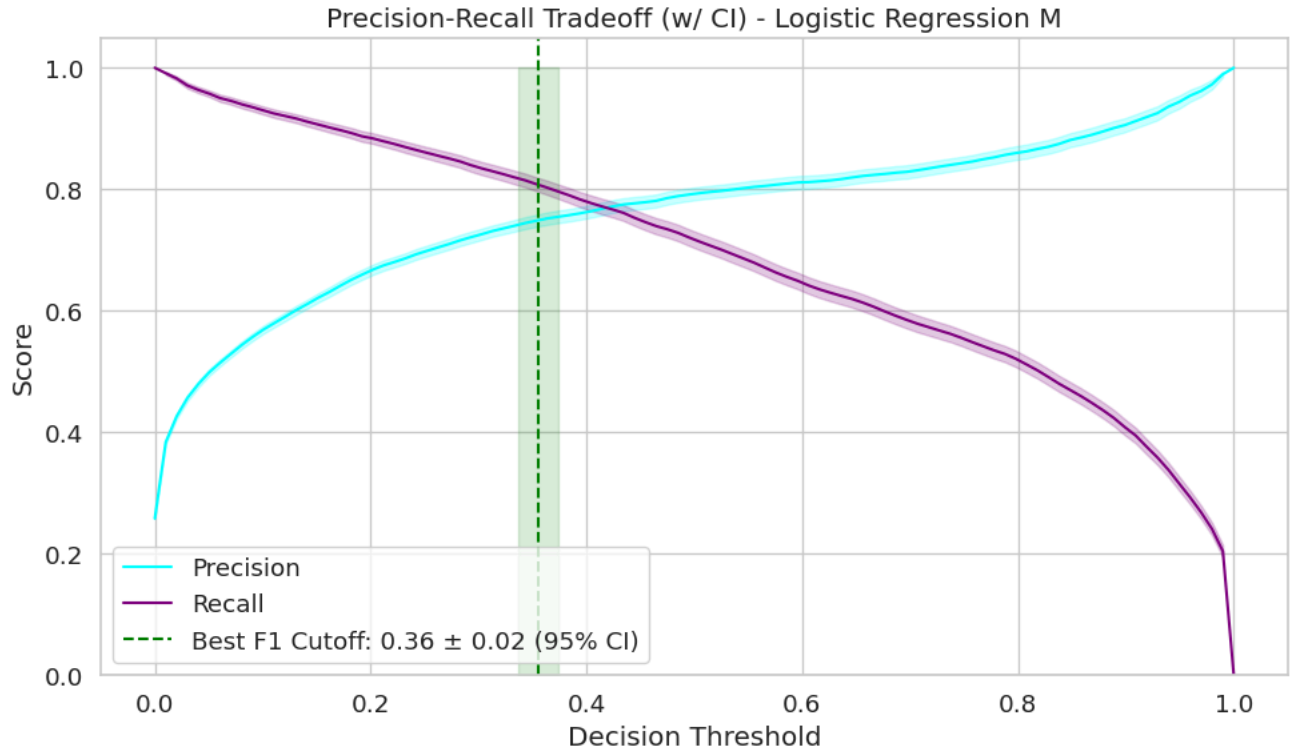
```
y2=y_severity
def encode_binary_labels(y):
    if isinstance(y, (pd.Series, np.ndarray)):
        unique_labels = np.unique(y)
        if "complicated" in unique_labels and "uncomplicated" in unique_labels:
            print("🔄 Converting 'complicated' to 1 and 'uncomplicated' to 0...")
            return pd.Series(y).replace({"complicated": 1, "uncomplicated": 0})
        elif sorted(unique_labels) == [0, 1]:
            print("✔ Labels are already correctly encoded (0=uncomplicated, 1=complicated).")
            return pd.Series(y)
        else:
            print(f"⚠ Warning: Unexpected labels {unique_labels}. Please check!")
            return pd.Series(y)
    else:
        raise TypeError("Input must be a pandas Series or numpy array.")

y2 = encode_binary_labels(y2)
```

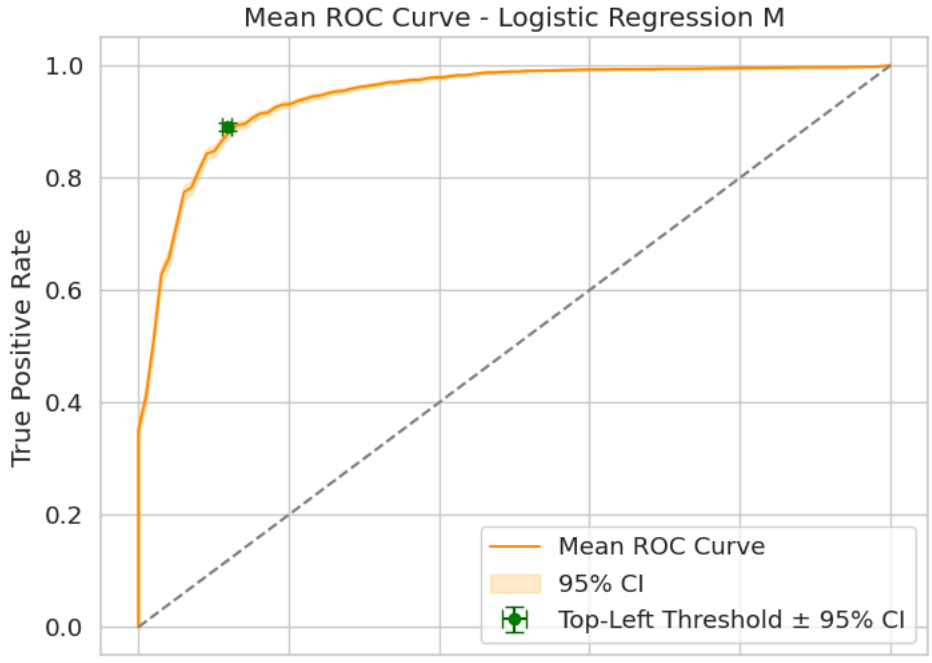
```
🔄 Converting 'complicated' to 1 and 'uncomplicated' to 0...
```

```
num_cols = X_severity_reduced.select_dtypes(exclude=['object', 'category']).columns.tolist()

logreg_model = LogisticRegression(max_iter=1000, random_state=2025)

logreg_resultsM, best_logregM = monte_carlo_cv_with_plot(
    model=logreg_model,
    param_grid=None,  # No tuning
    X=X_severity_reduced,
    y=y2,
    model_name="Logistic Regression M",
    num_iterations=200
)

# Save results
monte_carlo_results["Logistic Regression M"].to_csv(f"{save_path_c_m}/logistic_regression_mccv_results.csv", index=False)
logreg_resultsM.to_csv(f"{save_path_c_m}/logistic_regression_summary.csv")
```

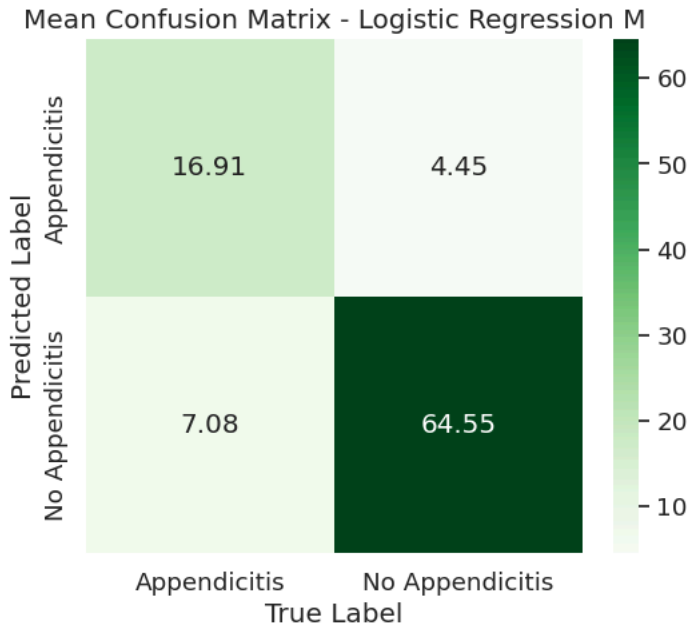
```
Monte Carlo Iteration 1/200...

Training LogisticRegression with Monte Carlo CV (No Hyperparameter)...
Monte Carlo CV - Logistic Regression M:  25%|██▌       | 50/200 [00:03<00:10, 14.98it/s]
Monte Carlo Iteration 50/200...

Training LogisticRegression with Monte Carlo CV (No Hyperparameter)...
Monte Carlo CV - Logistic Regression M:  51%|█████     | 102/200 [00:06<00:06, 15.72it/s]
Monte Carlo Iteration 100/200...

Training LogisticRegression with Monte Carlo CV (No Hyperparameter)...
Monte Carlo CV - Logistic Regression M:  75%|███████▌  | 150/200 [00:10<00:04, 11.24it/s]
Monte Carlo Iteration 150/200...

Training LogisticRegression with Monte Carlo CV (No Hyperparameter)...
Monte Carlo CV - Logistic Regression M: 100%|██████████| 200/200 [00:14<00:00, 13.75it/s]
Monte Carlo Iteration 200/200...

Training LogisticRegression with Monte Carlo CV (No Hyperparameter)...

Monte Carlo CV Results
```

	Mean Accuracy	Std Accuracy	Mean AUC	Std AUC	Mean Precision	Std Precision	Mean Recall	Std Recall	Mean F1	Std F1
Results	0.876	0.03	0.933	0.025	0.798	0.076	0.705	0.097	0.744	0.068






```
Mean Top-Left Threshold (ROC): 0.27 ± 0.01 (95% CI)
```



Appendicitis Severity Prediction

- **Clinical Motivation:**
  - Predicting the severity of appendicitis (complicated vs non-complicated) is important to optimise clinical management.
  - Complicated appendicitis often requires urgent surgical intervention, aggressive antibiotics, and careful perioperative monitoring to prevent serious complications such as sepsis or peritonitis.
  - Early identification allows clinicians to prioritise high-risk patients, allocate surgical resources appropriately, and improve patient outcomes.
- **Features Used:**
  - The same structured clinical, laboratory, and imaging-derived features used in the diagnosis model were applied for severity prediction.
  - These variables were expected to carry additional signal related to disease severity (raised inflammatory markers, ultrasound findings).
- **Best Hyperparameters:**
  - No hyperparameter tuning for this model.
- **Mean F1 Score (Optimal Threshold = 0.36 ± 0.02):**
  - **F1 Score:** 0.744 ± 0.068
- **Precision-Recall Trade-off Discussion:**
  - The Precision-Recall curve identified an optimal threshold of approximately 0.36.
  - This threshold prioritised **higher recall**, which is clinically justified: missing a case of complicated appendicitis is riskier than incorrectly classifying a non-complicated case.
  - However, the tradeoff led to a reduction in precision, meaning some non-complicated cases were overclassified as complicated. This is not ideal, as it may lead to unnecessary invasive surgery, which carries its own risks and patient burden.
- **Mean ROC Curve and AUC Discussion:**
  - The model demonstrated strong class separation with a **Mean AUC of 0.933 ± 0.025**.
  - The ROC curve showed a smooth and consistent separation across Monte Carlo splits, suggesting reliable performance.
  - The Top-Left threshold was similar to the F1-optimal threshold, reinforcing the decision to prioritise recall.
- **Other Metrics (at F1-optimal threshold ~0.33):**
  - **Mean Accuracy:** 0.876 ± 0.03
  - **Precision:** 0.798 ± 0.076
  - **Recall:** 0.705 ± 0.097
- **Advantages of Logistic Regression for Severity Prediction:**
  - Transparent and interpretable output, allowing clinical insight into feature contributions.
  - Fast and computationally lightweight, suitable for clinical decision support tools in real-time hospital systems.

- **Challenges and Limitations:**
  - Overlap in clinical presentation between complicated and non-complicated cases inherently limits model discriminability.
  - Some complicated appendicitis cases present with mild symptoms, leading to missed cases despite model tuning.
  - Future improvements could involve using dynamic trends or advanced imaging modalities (CT findings) to better capture subtle severity signals.

3.4 Elastic Net & Lasso reguralisation

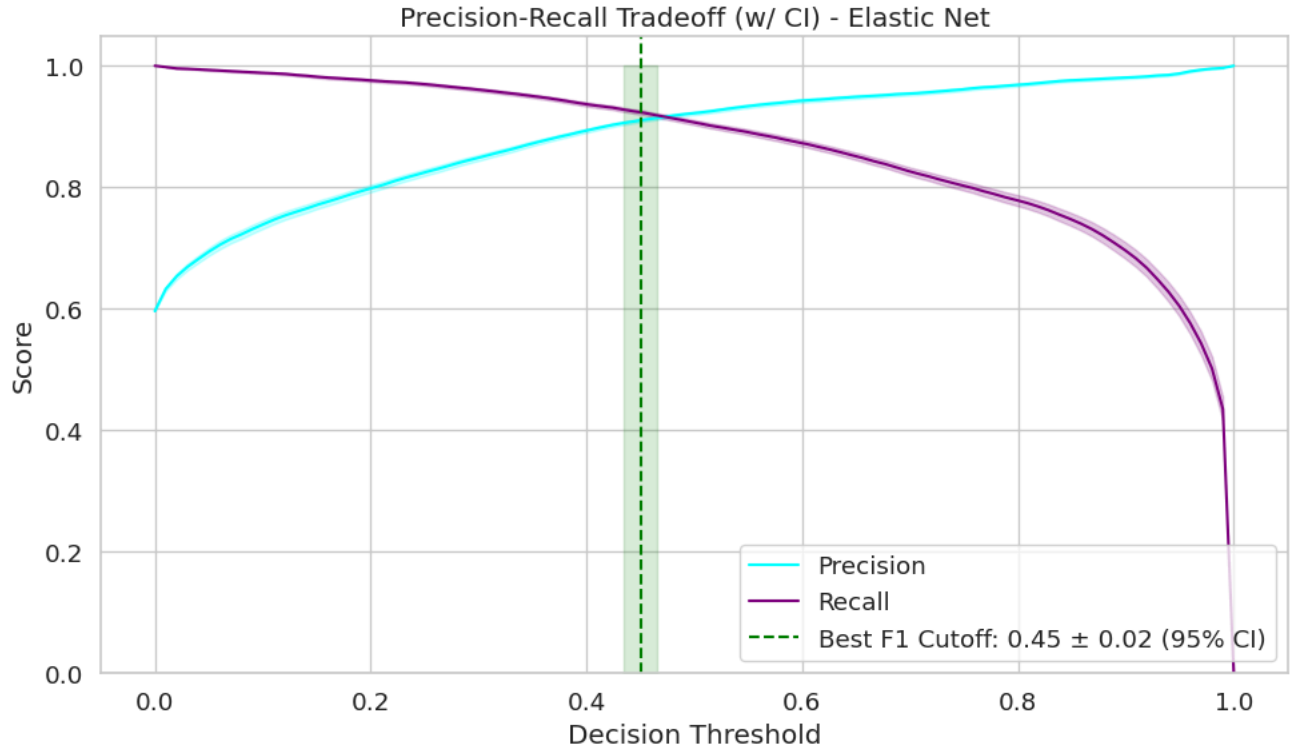
Diagnosis

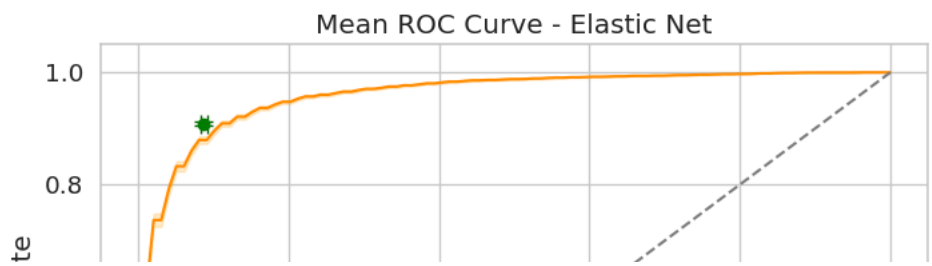
```
num_cols = X.select_dtypes(exclude=['object', 'category']).columns.tolist()
best_hyperparams_counter = defaultdict(int)  # Tracks best hyperparameters

elastic_model = LogisticRegression(
    penalty="elasticnet",solver="saga",
    random_state=2025,max_iter=10000)

elastic_param_grid = {
    "l1_ratio": [.8,.95,1.0],  # 0=L2 only, 1=L1 only
    "C": [0.1, 1.0],     # Lower = stronger regularization
}

elastic_results, best_elastic = monte_carlo_cv_with_plot(
    elastic_model,
    elastic_param_grid,
    X=X,
    y=y,
    model_name="Elastic Net",
    num_iterations=200
)
elastic_results.to_csv(f"{save_path_c_f}/elastic_net_summary.csv")
monte_carlo_results["Elastic Net"].to_csv(f"{save_path_c_f}/elastic_net_mccv_results.csv", index=False)
```

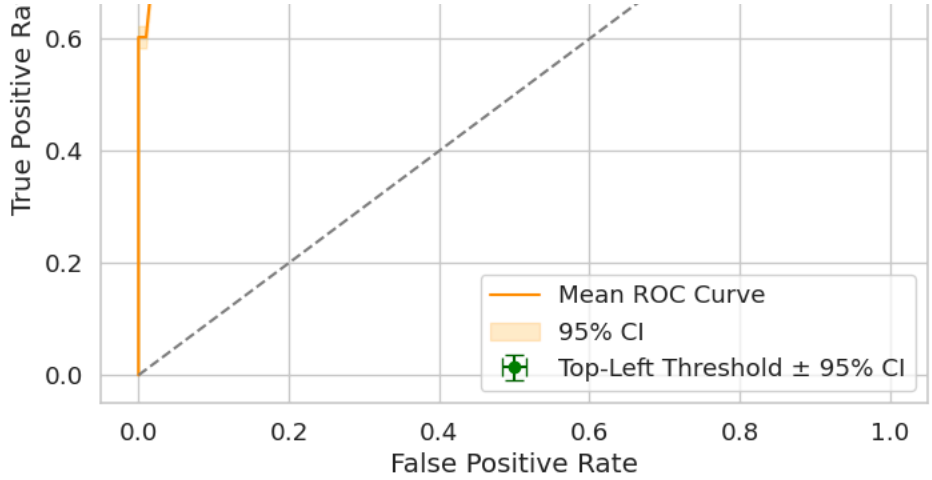
```
 Hyperparameter tuning for Elastic Net (Optimizing F1-Score)...
 Best Parameters: {'C': 0.1, 'l1_ratio': 0.95}
 Best Cross-Validation F1 Score: 0.9229
Monte Carlo CV - Elastic Net:  24%|██        | 49/200 [09:30<30:43, 12.21s/it]
 Monte Carlo Iteration 50/200...

 Hyperparameter tuning for Elastic Net (Optimizing F1-Score)...
 Best Parameters: {'C': 0.1, 'l1_ratio': 1.0}
 Best Cross-Validation F1 Score: 0.9175
Monte Carlo CV - Elastic Net:  50%|█████     | 99/200 [20:39<23:10, 13.77s/it]
 Monte Carlo Iteration 100/200...

 Hyperparameter tuning for Elastic Net (Optimizing F1-Score)...
 Best Parameters: {'C': 1.0, 'l1_ratio': 0.95}
 Best Cross-Validation F1 Score: 0.9114
Monte Carlo CV - Elastic Net:  74%|███████   | 149/200 [29:56<10:51, 12.77s/it]
 Monte Carlo Iteration 150/200...

 Hyperparameter tuning for Elastic Net (Optimizing F1-Score)...
Monte Carlo CV - Elastic Net:  75%|███████   | 150/200 [30:08<10:18, 12.36s/it] Best Parameters: {'C': 0.1, 'l1_ratio'
 Best Cross-Validation F1 Score: 0.9053
Monte Carlo CV - Elastic Net: 100%|██████████| 199/200 [39:25<00:10, 10.90s/it]
 Monte Carlo Iteration 200/200...

 Hyperparameter tuning for Elastic Net (Optimizing F1-Score)...
 Best Parameters: {'C': 1.0, 'l1_ratio': 1.0}
 Best Cross-Validation F1 Score: 0.9086
Monte Carlo CV - Elastic Net: 100%|██████████| 200/200 [39:35<00:00, 11.88s/it]
 Monte Carlo CV Results
```

	Mean Accuracy	Std Accuracy	Mean AUC	Std AUC	Mean Precision	Std Precision	Mean Recall	Std Recall	Mean F1	Std F1
Results	0.898	0.022	0.961	0.012	0.924	0.023	0.904	0.032	0.914	0.019

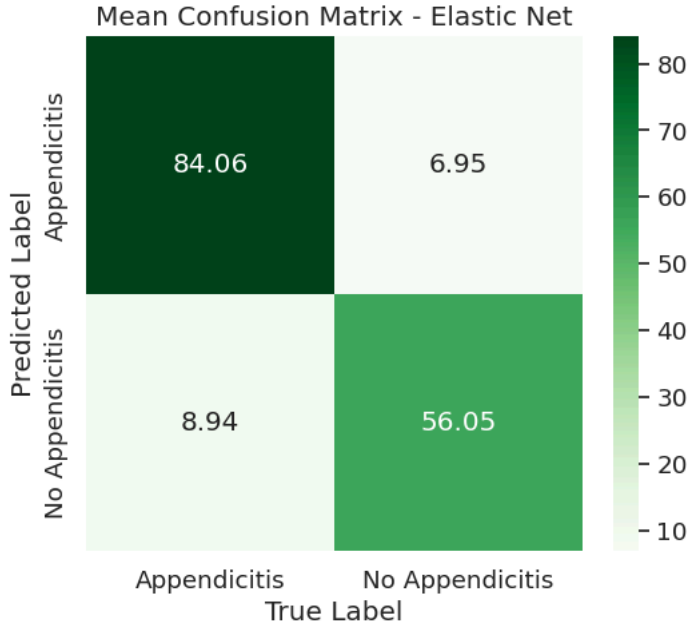
```
🏆 Most Common Best Hyperparameters:
1. Params: {'C': 1.0, 'l1_ratio': 1.0} | Chosen: 61 out of 200 times
2. Params: {'C': 1.0, 'l1_ratio': 0.95} | Chosen: 51 out of 200 times
3. Params: {'C': 1.0, 'l1_ratio': 0.8} | Chosen: 39 out of 200 times
4. Params: {'C': 0.1, 'l1_ratio': 1.0} | Chosen: 27 out of 200 times
5. Params: {'C': 0.1, 'l1_ratio': 0.95} | Chosen: 16 out of 200 times
6. Params: {'C': 0.1, 'l1_ratio': 0.8} | Chosen: 6 out of 200 times
```








```
Mean Top-Left Threshold (ROC): 0.54 ± 0.01 (95% CI)
```



Lasso (L1-Regularised Logistic Regression) - Appendicitis Diagnosis

> **Note on Hyperparameter Tuning:**
> An initial Monte Carlo Cross-Validation run with **20 iterations** was performed across a wide hyperparameter grid.
> After review, the most commonly selected combinations were retained for the final **200-iteration** evaluation.
>
> **Most Common Best Hyperparameters:**
>
> - `C = 1.0`, `l1_ratio = 1.0` (chosen 61/200 times) — pure Lasso
> - `C = 1.0`, `l1_ratio = 0.95` (chosen 51/200 times)

- **Clinical Motivation:**
  - Lasso regularisation encourages sparsity by shrinking less important feature coefficients to zero, offering a simpler, more interpretable model.
  - In clinical settings, this helps prioritise the most relevant features for appendicitis diagnosis while controlling overfitting.
- **Features Used:**
  - All available features were retained, with L1 regularisation naturally performing feature selection during model fitting.
- **Performance Overview (Optimal Threshold ≈ 0.45 ± 0.02):**
  - **F1 Score:** 0.914 ± 0.019
  - **AUC:** 0.961 ± 0.012
  - **Accuracy:** 0.898 ± 0.022
  - **Precision:** 0.924 ± 0.023
  - **Recall:** 0.904 ± 0.032
- **Precision-Recall and ROC Analysis:**
  - The optimal threshold (≈0.45) balanced precision and recall effectively.
  - High AUC indicated excellent overall discrimination, with minimal compromise between sensitivity and specificity.
- **Confusion Matrix Insights:**
  - The model correctly identified 84.1% of appendicitis cases on average.
  - False positive and false negative rates remained low.

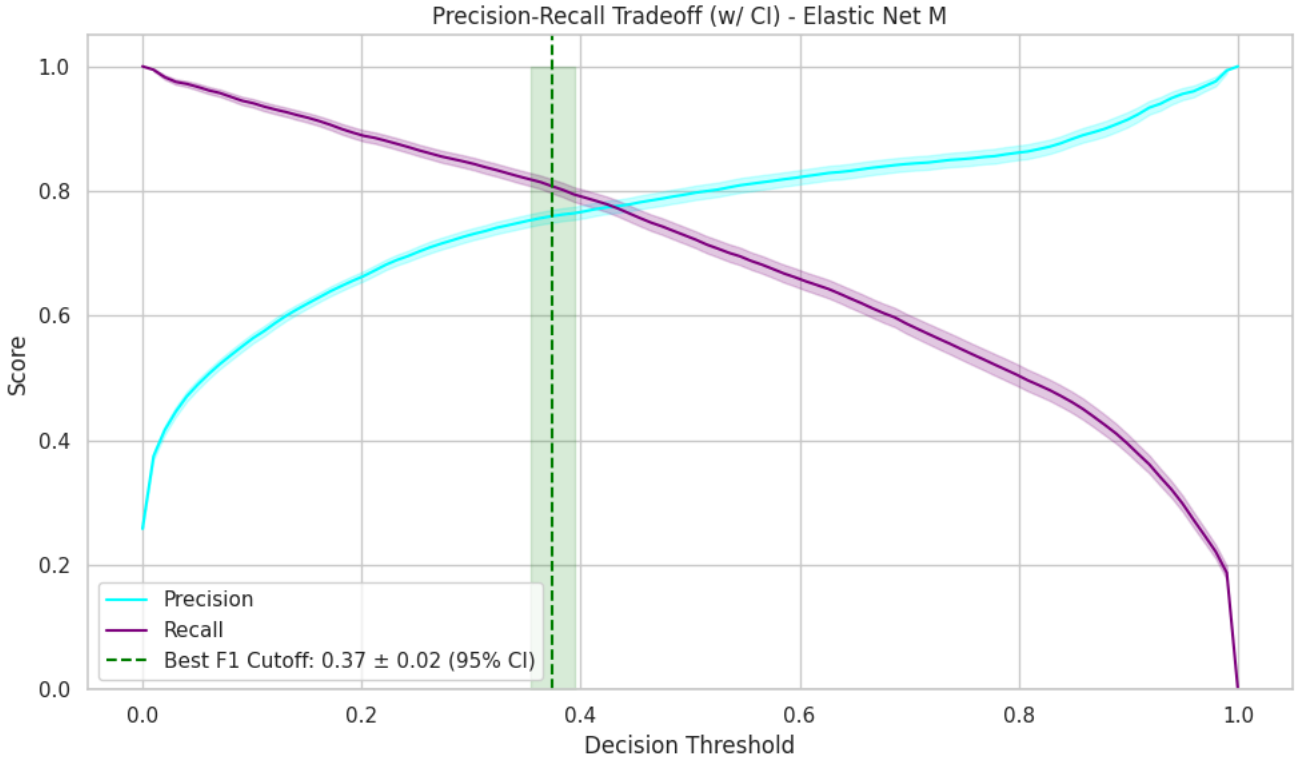
Severity

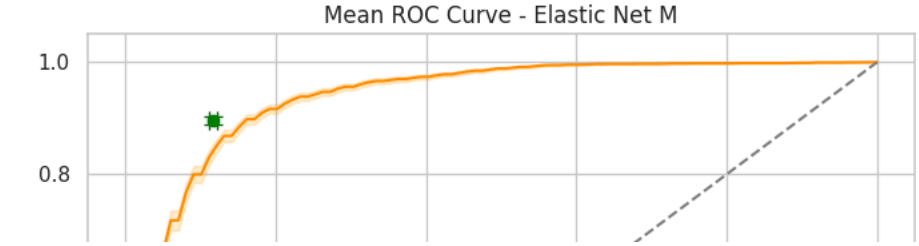
```
elastic_model = LogisticRegression(
    penalty="elasticnet",solver="saga",
    random_state=2025,max_iter=3000)
best_hyperparams_counter = defaultdict(int)  # Tracks best hyperparameters

elastic_resultsM, best_elasticM = monte_carlo_cv_with_plot(
    elastic_model,
    elastic_param_grid,
    X=X_severity,
    y=y2,
    model_name="Elastic Net M",
    num_iterations=200
)
elastic_resultsM.to_csv(f"{save_path_c_m}/elastic_net_summary.csv")
monte_carlo_results["Elastic Net M"].to_csv(f"{save_path_c_m}/elastic_net_mccv_results.csv", index=False)
```

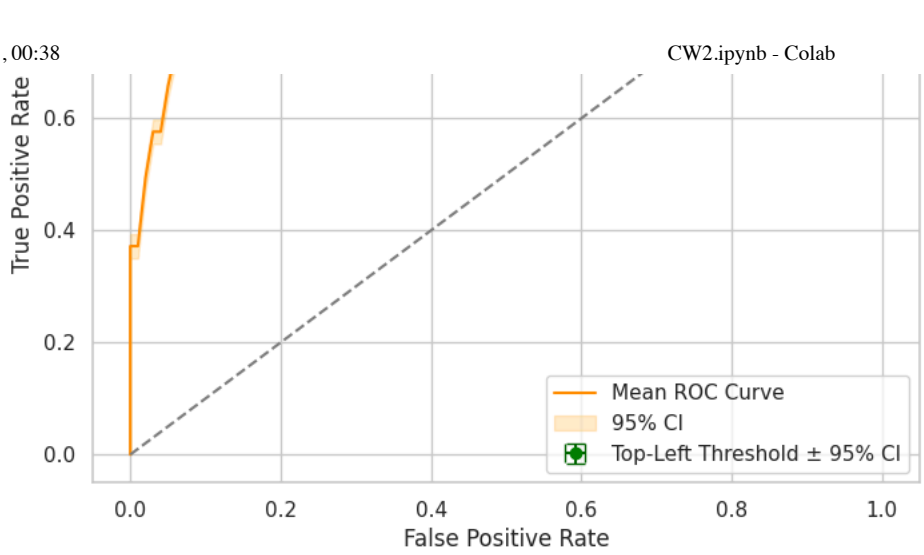
```
 Monte Carlo Iteration 1/200...

 Hyperparameter tuning for Elastic Net M (Optimizing F1-Score)...
Monte Carlo CV - Elastic Net M:   0%|          | 1/200 [00:04<14:58,  4.51s/it] Best Parameters: {'C': 1.0, 'l1_ratio'
 Best Cross-Validation F1 Score: 0.7659
Monte Carlo CV - Elastic Net M:  24%|██        | 49/200 [04:13<10:49,  4.30s/it]
 Monte Carlo Iteration 50/200...

 Hyperparameter tuning for Elastic Net M (Optimizing F1-Score)...
Monte Carlo CV - Elastic Net M:  25%|██        | 50/200 [04:16<09:39,  3.86s/it] Best Parameters: {'C': 1.0, 'l1_ratio
 Best Cross-Validation F1 Score: 0.7555
Monte Carlo CV - Elastic Net M:  50%|█████     | 99/200 [08:09<11:27,  6.81s/it]
 Monte Carlo Iteration 100/200...

 Hyperparameter tuning for Elastic Net M (Optimizing F1-Score)...
 Best Parameters: {'C': 1.0, 'l1_ratio': 1.0}
 Best Cross-Validation F1 Score: 0.8298
Monte Carlo CV - Elastic Net M:  74%|███████   | 149/200 [11:51<04:40,  5.49s/it]
 Monte Carlo Iteration 150/200...

 Hyperparameter tuning for Elastic Net M (Optimizing F1-Score)...
 Best Parameters: {'C': 1.0, 'l1_ratio': 1.0}
 Best Cross-Validation F1 Score: 0.7413
Monte Carlo CV - Elastic Net M: 100%|█████████| 199/200 [15:48<00:04,  4.43s/it]
 Monte Carlo Iteration 200/200...

 Hyperparameter tuning for Elastic Net M (Optimizing F1-Score)...
 Best Parameters: {'C': 1.0, 'l1_ratio': 0.95}
 Best Cross-Validation F1 Score: 0.7985
Monte Carlo CV - Elastic Net M: 100%|█████████| 200/200 [15:51<00:00,  4.76s/it]
 Monte Carlo CV Results
```

	Mean Accuracy	Std Accuracy	Mean AUC	Std AUC	Mean Precision	Std Precision	Mean Recall	Std Recall	Mean F1	Std F1
Results	0.878	0.031	0.937	0.024	0.801	0.082	0.714	0.098	0.75	0.069

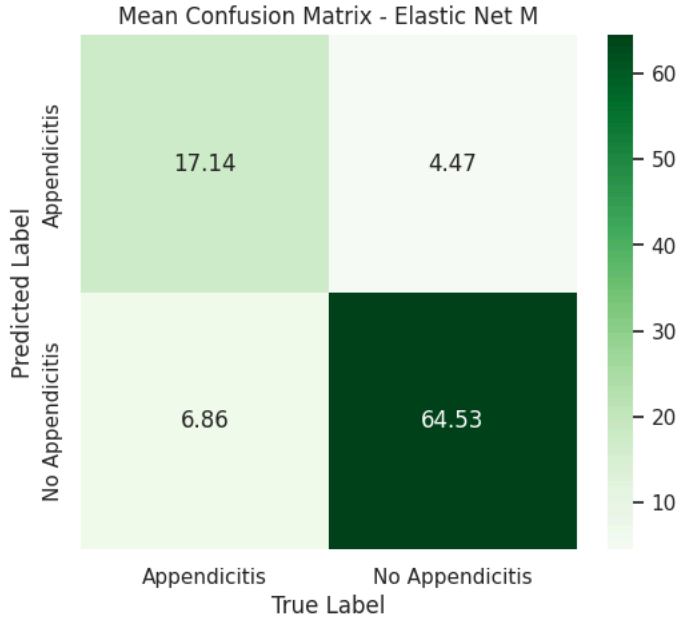
```
 Most Common Best Hyperparameters:
1. Params: {'C': 1.0, 'l1_ratio': 0.8} | Chosen: 76 out of 200 times
2. Params: {'C': 1.0, 'l1_ratio': 0.95} | Chosen: 70 out of 200 times
3. Params: {'C': 1.0, 'l1_ratio': 1.0} | Chosen: 38 out of 200 times
4. Params: {'C': 0.1, 'l1_ratio': 0.8} | Chosen: 9 out of 200 times
5. Params: {'C': 0.1, 'l1_ratio': 0.95} | Chosen: 5 out of 200 times
6. Params: {'C': 0.1, 'l1_ratio': 1.0} | Chosen: 2 out of 200 times
```








```
Mean Top-Left Threshold (ROC): 0.28 ± 0.02 (95% CI)
```



Elastic Net - Appendicitis Severity Classification

> **Note on Hyperparameter Tuning:**
> A Monte Carlo Cross-Validation (200 iterations) was used for model evaluation.
> An initial wider hyperparameter search found that the following combinations were most commonly selected:
>
> - `C = 1.0`, `l1_ratio = 0.8` (chosen 76 times)
> - `C = 1.0`, `l1_ratio = 0.95` (chosen 70 times)

- **Clinical Motivation:**
  Elastic Net combines L1 (Lasso) and L2 (Ridge) penalties, providing regularisation that can handle correlated predictors while still allowing for feature selection.
  This is valuable in clinical data where some predictors may overlap in informational content.
- **Features Used:**
  All available clinical features were included.
  Elastic Net's regularisation inherently reduces overfitting by shrinking less important coefficients towards zero.
- **Best Hyperparameters:**
  The model predominantly favoured mild to moderate L1 regularisation (`l1_ratio ≈ 0.8`), suggesting that some feature sparsity helped improve generalisation without discarding useful predictors.
- **Performance Metrics (Optimal Threshold = 0.37 ± 0.02):**
  - **Mean F1 Score:** 0.750 ± 0.069
  - **Mean Accuracy:** 0.878 ± 0.031
  - **Mean AUC:** 0.937 ± 0.024
  - **Mean Precision:** 0.801 ± 0.082
  - **Mean Recall:** 0.714 ± 0.098
- **Precision-Recall Trade-off Discussion:**
  - The optimal threshold (0.37) shifted lower than 0.5, favouring improved sensitivity to complicated cases.
  - Precision remained slightly higher than recall across thresholds, indicating a cautious model with a lower false positive rate.
- **Mean ROC Curve and AUC Discussion:**
  - The ROC curve achieved a **Mean AUC of 0.937**, demonstrating strong discriminative ability between uncomplicated and complicated appendicitis cases.
  - The ROC optimal threshold aligned closely with the F1-optimal point.
- **Confusion Matrix Insights:**
  - True complicated cases were correctly identified ~71% of the time.
  - Misclassifications skewed towards false negatives (complicated cases missed) rather than false positives.
- **Advantages of Elastic Net:**
  - Good generalisation to unseen data.
  - Provides interpretable coefficients important for clinical explainability.
  - Handles multicollinearity between symptoms/lab findings effectively.
- **Challenges and Limitations:**
  - Linear decision boundary may limit detection of subtle non-linear patterns.
  - Recall for complicated cases, while reasonable, could still be improved.

3.5 Random Forest

Diagnosis

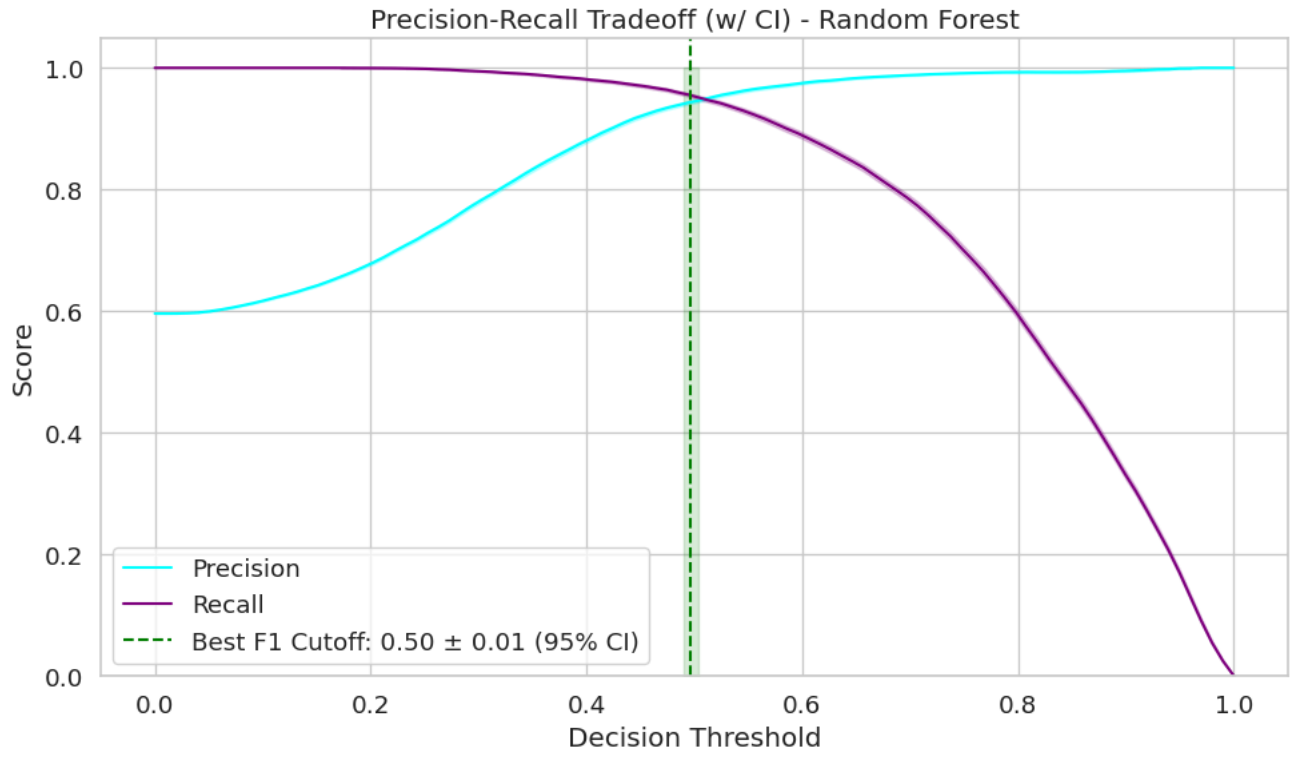
```
rf_param_grid = {
    "n_estimators": [80],
    "max_depth": [8],
    "min_samples_split": [2, 5]
}
best_hyperparams_counter = defaultdict(int)  # Tracks best hyperparameters

rf_model = RandomForestClassifier(random_state=2025)
rf_results, best_rf_model = monte_carlo_cv_with_plot(rf_model,
                                                     rf_param_grid,
                                                     X, y, model_name="Random Forest",
                                                     num_iterations=200)
```

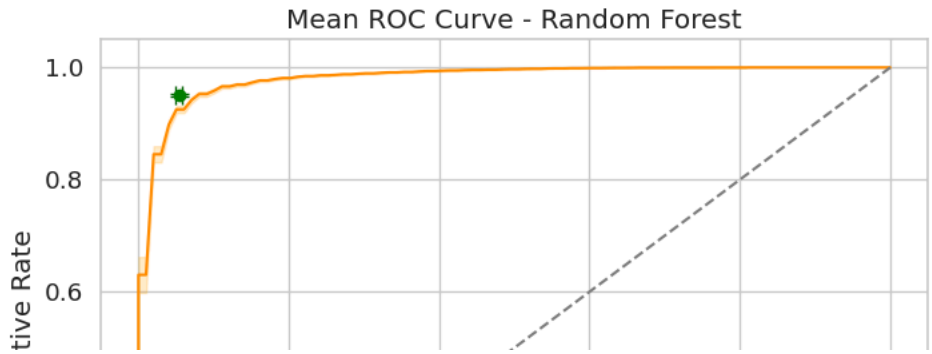
```
monte_carlo_results["Random Forest"].to_csv(f"{save_path_c_f}/random_forest_mccv_results.csv", index=False)
rf_results.to_csv(f"{save_path_c_f}/random_forest_summary.csv")
```

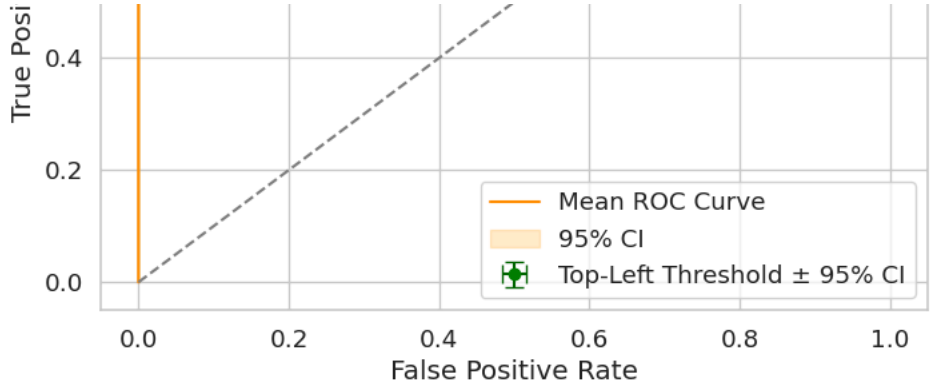
```
Monte Carlo Iteration 1/200...

 Hyperparameter tuning for Random Forest (Optimizing F1-Score)...
Monte Carlo CV - Random Forest:   0%|          | 1/200 [00:01<05:41,  1.72s/it] Best Parameters: {'max_depth': 8, 'min
 Best Cross-Validation F1 Score: 0.9516
Monte Carlo CV - Random Forest:  24%|██        | 49/200 [01:34<04:57,  1.97s/it]
 Monte Carlo Iteration 50/200...

 Hyperparameter tuning for Random Forest (Optimizing F1-Score)...
Monte Carlo CV - Random Forest:  25%|██        | 50/200 [01:35<04:42,  1.88s/it] Best Parameters: {'max_depth': 8, 'mi
 Best Cross-Validation F1 Score: 0.9417
Monte Carlo CV - Random Forest:  50%|█████     | 99/200 [03:09<03:34,  2.12s/it]
 Monte Carlo Iteration 100/200...

 Hyperparameter tuning for Random Forest (Optimizing F1-Score)...
Monte Carlo CV - Random Forest:  50%|█████     | 100/200 [03:11<03:23,  2.04s/it] Best Parameters: {'max_depth': 8, 'm
 Best Cross-Validation F1 Score: 0.9210
Monte Carlo CV - Random Forest:  74%|███████   | 149/200 [04:46<01:48,  2.12s/it]
 Monte Carlo Iteration 150/200...

 Hyperparameter tuning for Random Forest (Optimizing F1-Score)...
Monte Carlo CV - Random Forest:  75%|███████   | 150/200 [04:48<01:44,  2.09s/it] Best Parameters: {'max_depth': 8, 'm
 Best Cross-Validation F1 Score: 0.9382
Monte Carlo CV - Random Forest: 100%|██████████| 199/200 [06:27<00:02,  2.04s/it]
 Monte Carlo Iteration 200/200...

 Hyperparameter tuning for Random Forest (Optimizing F1-Score)...
Monte Carlo CV - Random Forest: 100%|██████████| 200/200 [06:28<00:00,  1.94s/it] Best Parameters: {'max_depth': 8, 'm
 Best Cross-Validation F1 Score: 0.9325

 Monte Carlo CV Results
```

	Mean Accuracy	Std Accuracy	Mean AUC	Std AUC	Mean Precision	Std Precision	Mean Recall	Std Recall	Mean F1	Std F1
Results	0.939	0.019	0.981	0.009	0.947	0.021	0.951	0.024	0.949	0.016

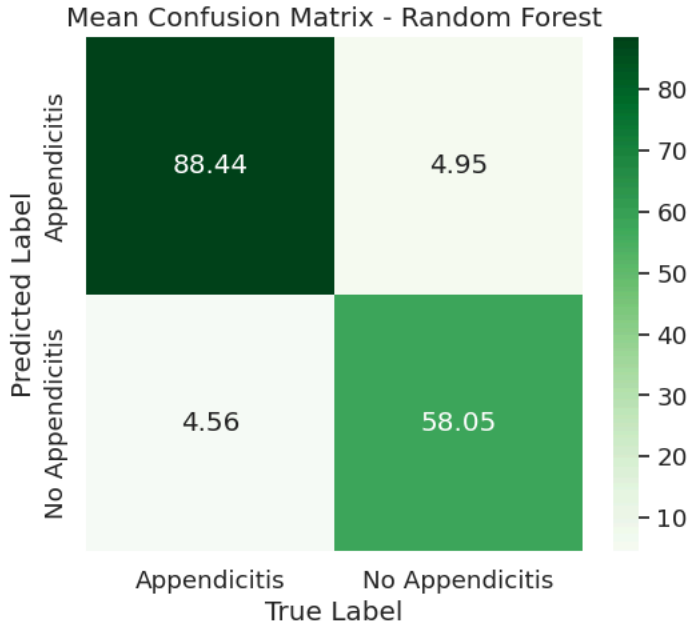
```
🏆 Most Common Best Hyperparameters:
1. Params: {'max_depth': 8, 'min_samples_split': 5, 'n_estimators': 80} | Chosen: 113 out of 200 times
2. Params: {'max_depth': 8, 'n_estimators': 80, 'min_samples_split': 2} | Chosen: 87 out of 200 times
```








```
Mean Top-Left Threshold (ROC): 0.53 ± 0.01 (95% CI)
```



Random Forest - Appendicitis Diagnosis

> **Note on Hyperparameter Tuning:**
> An initial Monte Carlo Cross-Validation run with **20 iterations** was performed to explore a broad hyperparameter space.
> Based on early results, the two most common parameter combinations were selected for the final **200-iteration** evaluation.
>
> **Most Common Best Hyperparameters:**
>
> - `max_depth=8, min_samples_split=5, n_estimators=80` (chosen 113/200 times)
> - `max_depth=8, min_samples_split=2, n_estimators=80` (chosen 87/200 times)

- **Clinical Motivation:**
  - Random Forests are highly effective for structured clinical data, handling non-linearities, and automatic feature selection.
  - They are particularly well-suited for appendicitis diagnosis, where complex interactions between symptoms, lab values, and imaging findings exist.
- **Features Used:**
  - All available clinical, laboratory, and imaging-derived features were included.
- **Performance Overview (Optimal Threshold ≈ 0.50 ± 0.01):**
  - **F1 Score:** 0.949 ± 0.016
  - **AUC:** 0.981 ± 0.009
  - **Accuracy:** 0.939 ± 0.019
  - **Precision:** 0.947 ± 0.021
  - **Recall:** 0.951 ± 0.024
- **Precision-Recall and ROC Analysis:**
  - The F1-optimal threshold was approximately 0.50, suggesting excellent balance between precision and recall.
  - The ROC curve showed near-perfect separation, with a very high AUC indicating strong discriminative power.
- **Confusion Matrix Insights:**
  - The model correctly identified 88.4% of appendicitis cases on average.
  - False positive and false negative rates were both extremely low.
- **Advantages of Random Forest:**
  - Captures complex non-linear relationships without heavy preprocessing.
  - Naturally handles feature interactions.
  - Resistant to overfitting when tuned correctly.
- **Challenges and Limitations:**
  - Less interpretable compared to logistic regression or lasso models.
  - Important features could be harder to extract without further feature importance analysis.

Severity

```
rf_model = RandomForestClassifier(random_state=2025)
best_hyperparams_counter = defaultdict(int)  # Tracks best hyperparameters

rf_resultsM, best_rf_modelM = monte_carlo_cv_with_plot(rf_model,
                                                       rf_param_grid,
                                                       X_severity, y2, model_name="Random Forest M",
                                                       num_iterations=200)

monte_carlo_results["Random Forest M"].to_csv(f"{save_path_c_m}/random_forest_mccv_results.csv", index=False)
rf_resultsM.to_csv(f"{save_path_c_m}/random_forest_summary.csv")
```

```
Monte Carlo CV - Random Forest M:   0%|          | 0/200 [00:00<?, ?it/s]
 Monte Carlo Iteration 1/200...

 Hyperparameter tuning for Random Forest M (Optimizing F1-Score)...
Monte Carlo CV - Random Forest M:   0%|          | 1/200 [00:01<05:17,  1.60s/it] Best Parameters: {'max_depth': 8, 'm
 Best Cross-Validation F1 Score: 0.7672
Monte Carlo CV - Random Forest M:  24%|██        | 49/200 [01:24<04:17,  1.70s/it]
 Monte Carlo Iteration 50/200...

 Hyperparameter tuning for Random Forest M (Optimizing F1-Score)...
Monte Carlo CV - Random Forest M:  25%|██        | 50/200 [01:26<04:06,  1.64s/it] Best Parameters: {'max_depth': 8, '
 Best Cross-Validation F1 Score: 0.7445
Monte Carlo CV - Random Forest M:  50%|████      | 99/200 [02:49<02:47,  1.66s/it]
 Monte Carlo Iteration 100/200...

 Hyperparameter tuning for Random Forest M (Optimizing F1-Score)...
Monte Carlo CV - Random Forest M:  50%|████      | 100/200 [02:51<02:40,  1.60s/it] Best Parameters: {'max_depth': 8,
 Best Cross-Validation F1 Score: 0.7976
Monte Carlo CV - Random Forest M:  74%|██████    | 149/200 [04:14<01:24,  1.66s/it]
 Monte Carlo Iteration 150/200...

 Hyperparameter tuning for Random Forest M (Optimizing F1-Score)...
Monte Carlo CV - Random Forest M:  75%|██████    | 150/200 [04:15<01:20,  1.61s/it] Best Parameters: {'max_depth': 8,
 Best Cross-Validation F1 Score: 0.7565
Monte Carlo CV - Random Forest M: 100%|████████| 199/200 [05:39<00:01,  1.68s/it]
 Monte Carlo Iteration 200/200...

 Hyperparameter tuning for Random Forest M (Optimizing F1-Score)...
Monte Carlo CV - Random Forest M: 100%|████████| 200/200 [05:40<00:00,  1.70s/it] Best Parameters: {'max_depth': 8,
 Best Cross-Validation F1 Score: 0.7737

 Monte Carlo CV Results
```

	Mean Accuracy	Std Accuracy	Mean AUC	Std AUC	Mean Precision	Std Precision	Mean Recall	Std Recall	Mean F1	Std F1
Results	0.887	0.028	0.934	0.029	0.866	0.071	0.669	0.096	0.75	0.071

```
🏆 Most Common Best Hyperparameters:
1. Params: {'max_depth': 8, 'min_samples_split': 5, 'n_estimators': 80} | Chosen: 107 out of 200 times
2. Params: {'max_depth': 8, 'n_estimators': 80, 'min_samples_split': 2} | Chosen: 93 out of 200 times
```