LIVER CIRRHOSIS STAGE PREDICTION with 95% Accuracy

Abstract

Liver cirrhosis is a progressive and irreversible condition that results from chronic liver diseases, leading to fibrosis, functional decline, and eventual organ failure (Schuppan & Afdhal, 2008). The accurate staging of cirrhosis is critical for clinical decision-making, guiding therapeutic interventions, and improving patient prognosis (Tsochatzis et al., 2014). Traditional diagnostic methods such as biopsy, imaging, and laboratory tests, while effective, present limitations in accuracy, invasiveness, and inter-observer variability (Marcellin & Kutala, 2018). Machine learning (ML) has emerged as a powerful tool in medical diagnostics, offering automated, accurate, and scalable solutions for disease classification (Esteva et al., 2019).This study explores the potential of ML models in classifying the histologic stages of liver cirrhosis using a dataset from a Mayo Clinic study on primary biliary cirrhosis (PBC), collected between 1974 and 1984, and publicly available on Kaggle (Dickson et al., 1989). The dataset contains clinical and laboratory parameters, including demographic details (age, sex), biochemical markers (bilirubin, albumin, cholesterol, SGOT, platelets), and disease-related attributes (ascites, edema, hepatomegaly, and spider angiomas). Before model training, data preprocessing was performed, including handling missing values, encoding categorical variables, and normalizing numerical attributes. Additionally, synthetic data augmentation techniques were applied to address class imbalances and improve model robustness (He et al., 2009). Logistic Regression yielded an accuracy of 60%, indicating limited efficacy in this context. The Decision Tree Classifier improved performance with an 88% accuracy but posed a risk of overfitting. The Random Forest Classifier achieved the highest accuracy at 95%, demonstrating superior precision, recall, and F1-scores across all classes, and

effectively mitigating overfitting. The K-Nearest Neighbors (KNN) algorithm also performed well with an 89% accuracy, though it is computationally intensive for larger datasets. Gradient Boosting Classifier attained an 83% accuracy, outperforming Logistic Regression but underperforming compared to other models. These findings underscore the efficacy of ensemble tree-based methods, particularly Random Forest, in accurately staging liver cirrhosis. The effectiveness of ML-based approaches in staging liver cirrhosis, particularly ensemble tree-based methods such as Random Forest, which demonstrated superior predictive performance. The study underscores the potential of ML in augmenting traditional diagnostic frameworks, offering a non-invasive, data-driven alternative for early cirrhosis detection and staging. Future research should explore deep learning architectures and external validation using independent datasets to enhance the robustness and clinical applicability of ML models in hepatology (Liu et al., 2022).

Introduction

Liver cirrhosis is a progressive and irreversible condition characterized by extensive fibrosis and the formation of regenerative nodules, which ultimately impair liver function. It represents the final stage of chronic liver diseases (CLDs) and is a leading cause of morbidity and mortality worldwide (Schuppan & Afdhal, 2008). The disease is a consequence of sustained liver injury due to various etiologies, including chronic hepatitis B and C infections, excessive alcohol consumption, and metabolic dysfunction–associated steatotic liver disease (MASLD) (Tsochatzis et al., 2014). Liver cirrhosis is a major global health concern, with approximately 2 million deaths annually attributed to cirrhosis-related complications such as liver failure, portal hypertension, and hepatocellular carcinoma (Asrani et al., 2019).

The pathophysiology of cirrhosis is driven by a persistent cycle of liver injury, inflammation, and wound-healing responses, leading to excessive deposition of extracellular matrix (ECM) proteins, primarily collagen (Bataller & Brenner, 2005). Hepatic stellate cells (HSCs) play a central role in fibrosis progression, as they transform into myofibroblast-like cells that secrete ECM components in response to chronic injury (Hernandez-Gea & Friedman, 2011). Over time, fibrotic tissue disrupts the normal hepatic architecture, leading to increased intrahepatic resistance and portal hypertension, a hallmark of cirrhosis.

Clinically, cirrhosis progresses through compensated and decompensated stages. The compensated stage is often asymptomatic, with normal liver function despite fibrotic damage. As the disease advances, patients develop complications such as ascites, hepatic encephalopathy, and variceal bleeding, indicating decompensation and an increased risk of liver-related mortality (D'Amico et al., 2018). Diagnosis of cirrhosis typically involves a combination of clinical assessment, biochemical markers, imaging techniques (e.g., elastography), and liver biopsy when necessary (European Association for the Study of the Liver [EASL], 2015).

Problem Statement

Liver cirrhosis is a chronic and progressive liver disease that leads to liver failure if not detected and managed in its early stages. Traditional diagnostic approaches, such as biopsies and blood tests, often have limitations in accuracy, invasiveness, and cost-effectiveness. With the advancement of machine learning (ML), automated models can assist in classifying liver cirrhosis stages more accurately and efficiently. However, selecting the best ML model remains a challenge due to differences in model performance, feature importance, and computational efficiency. This study evaluates multiple ML models to determine the most effective approach for Liver Cirrhosis Stage Prediction using a dataset containing key clinical features such as Bilirubin, Albumin, Platelets, and Prothrombin levels, as well as patient demographics and medical conditions.The dataset used in this study contains various medical and demographic attributes that serve as potential predictors for liver cirrhosis stages. These attributes include patient demographics (age, sex), clinical symptoms (ascites, hepatomegaly, edema, and spider angiomas), liver function biomarkers (bilirubin, albumin, SGOT, and prothrombin), metabolic markers (cholesterol, triglycerides, alkaline phosphatase, and copper), and hematological factors (platelet count) (Pimpin et al., 2018).

N_Days – Represents the number of days the patient has been under observation in the study. This can be useful for understanding disease progression over time.

Status – Indicates the patient’s current condition, which can be:

```
*Alive (patient is still living)
 *Dead (patient has passed away)
 *Transplanted (patient has undergone a liver transplant)
```

Drug – Specifies the type of treatment the patient has received. This can help analyze whether certain treatments affect disease progression.

Age – The patient’s age at the time of observation, which may be a risk factor for disease severity.

Sex – The biological sex of the patient (Male/Female), which can influence disease presentation and progression.

Ascites – A binary indicator (Yes/No) of fluid accumulation in the abdominal cavity, a common symptom of liver cirrhosis.

Hepatomegaly – A binary indicator (Yes/No) of an enlarged liver, often associated with liver disease.

Spiders – A binary indicator (Yes/No) for the presence of spider angiomas, which are small, dilated blood vessels seen in liver disease patients.

Edema – A categorical variable indicating the presence of swelling due to fluid retention. Possible values:

```
* No edema
* Edema not responding to diuretics
* Edema responding to diuretics
```

Bilirubin – A liver function biomarker indicating the level of bilirubin in the blood. High bilirubin levels suggest impaired liver function.

Cholesterol – Measures the cholesterol level in the blood. Liver dysfunction can cause abnormal cholesterol metabolism.

Albumin – A protein produced by the liver, essential for maintaining blood volume and transporting substances. Low levels indicate poor liver function.

Copper – Measures copper levels in the blood. The liver plays a key role in copper metabolism, and abnormal levels may indicate liver disease.

Alk_Phos (Alkaline Phosphatase) – An enzyme that indicates liver function. Elevated levels may signal bile duct obstruction or liver damage.

SGOT (Serum Glutamic Oxaloacetic Transaminase) – Also known as AST (Aspartate Aminotransferase), this enzyme is released when liver cells are damaged.

Tryglicerides – The level of triglycerides (a type of fat) in the blood, which may be affected by liver disease.

Platelets – The number of platelets in the blood, important for clotting. Low platelet count is common in liver cirrhosis.

Prothrombin – The time taken for blood to clot. A prolonged prothrombin time indicates impaired liver function and a higher risk of bleeding.

Stage – The target variable representing different stages of liver cirrhosis. The number of stages may vary depending on the classification system used (e.g., mild, moderate, severe).

This dataset includes a mix of categorical and numerical variables, all of which are crucial for predicting liver cirrhosis stages using machine learning models

Methodology

The following Python libraries were utilized for various stages of data processing, analysis, and model building

Libraries Used

Pandas: A powerful Python library used for data manipulation and analysis, particularly for handling tabular data. It was used for loading, cleaning, and transforming the dataset.NumPy: A library for numerical operations and handling arrays. It was used for mathematical functions and numerical computations on data.Matplotlib: A plotting library used for creating static, interactive, and animated visualizations, including histograms, scatter plots, and box plots, to explore and visualize data.Seaborn: A statistical data

visualization library built on top of Matplotlib, used for creating advanced visualizations such as heatmaps and pair plots to analyze relationships in data.Scikit-learn: A machine learning library used for model building and evaluation. It provides tools for training and evaluating classifiers, including Decision Trees, Logistic Regression, SVM, Random Forest, and K-Nearest Neighbors (KNN).Plotly: A graphing library used for creating interactive visualizations. It was used for generating subplots to represent complex data in an interactive format.itertools: A Python module used for creating iterators that efficiently loop through data. It was utilized to generate combinations and permutations during the analysis.Warnings: A module used to filter out unnecessary warnings during the execution of the code, ensuring cleaner output and focus during the analysis.Scikit-learn: A comprehensive machine learning library that provides tools for data preprocessing, model building, and evaluation. It was used for training various classifiers such as Decision Tree, Logistic Regression, Support Vector Machine (SVM), Random Forest, and K-Nearest Neighbors (KNN). It also provided functions for model evaluation through metrics like confusion matrix, ROC curve, precision-recall curve, and AUC.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
import warnings
warnings.filterwarnings("ignore")
```

Data Collection

The dataset for this project was collected from two reliable sources:

1. **Cirrhosis Patient Survival Prediction Dataset** from the UCI Machine Learning Repository, which provides data on patients diagnosed with cirrhosis. This dataset contains information related to the survival prediction of these patients based on various medical features. The dataset can be accessed here.
2. **Liver Cirrhosis Stage Classification Dataset** from Kaggle, which offers data about liver cirrhosis stages, containing records related to different stages of cirrhosis in patients. This dataset is useful for predicting the stage of cirrhosis based on patient attributes. The dataset can be accessed here.

These datasets provide valuable features for analyzing liver cirrhosis and predicting patient outcomes, which are pivotal for the development of predictive models.

In [2]:
```
df = pd.read_csv(r"C:\Users\manoj\Downloads\Liver_Cirrhosis\liver_cirrhosis.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol	Albumin	Copper	Alk_F
0	2221	C	Placebo	18499	F	N	Y	N	N	0.5	149.0	4.04	227.0	5
1	1230	C	Placebo	19724	M	Y	N	Y	N	0.5	219.0	3.93	22.0	6
2	4184	C	Placebo	11839	F	N	N	N	N	0.5	320.0	3.54	51.0	12
3	2090	D	Placebo	16467	F	N	N	N	N	0.7	255.0	3.74	23.0	10
4	2105	D	Placebo	21699	F	N	Y	N	N	1.9	486.0	3.54	74.0	10

In [4]:
```
df.tail()
```

Out[4]:

	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol	Albumin	Copp
24995	3584	D	D-penicillamine	23612	F	N	N	N	N	0.8	231.000000	3.87	17
24996	3584	D	D-penicillamine	23612	F	N	N	N	N	0.8	231.000000	3.87	17
24997	971	D	D-penicillamine	16736	F	N	Y	Y	Y	5.1	369.510563	3.23	1
24998	3707	C	D-penicillamine	16990	F	N	Y	N	N	0.8	315.000000	4.24	1
24999	3707	C	D-penicillamine	16990	F	N	Y	N	N	0.8	315.000000	4.24	1

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   N_Days         25000 non-null  int64  
 1   Status         25000 non-null  object 
 2   Drug           25000 non-null  object 
 3   Age            25000 non-null  int64  
 4   Sex            25000 non-null  object 
 5   Ascites        25000 non-null  object 
 6   Hepatomegaly   25000 non-null  object 
 7   Spiders        25000 non-null  object 
 8   Edema          25000 non-null  object 
 9   Bilirubin      25000 non-null  float64
 10  Cholesterol    25000 non-null  float64
 11  Albumin        25000 non-null  float64
 12  Copper         25000 non-null  float64
 13  Alk_Phos       25000 non-null  float64
 14  SGOT           25000 non-null  float64
 15  Tryglicerides  25000 non-null  float64
 16  Platelets      25000 non-null  float64
 17  Prothrombin    25000 non-null  float64
 18  Stage          25000 non-null  int64  
dtypes: float64(9), int64(3), object(7)
memory usage: 3.6+ MB
```

In [6]:
```
df.columns
```

Out[6]:
```
Index(['N_Days', 'Status', 'Drug', 'Age', 'Sex', 'Ascites', 'Hepatomegaly',
       'Spiders', 'Edema', 'Bilirubin', 'Cholesterol', 'Albumin', 'Copper',
       'Alk_Phos', 'SGOT', 'Tryglicerides', 'Platelets', 'Prothrombin',
       'Stage'],
      dtype='object')
```

DATA CLEANING

In [7]:
```
df.shape
```

Out[7]:
```
(25000, 19)
```

In [8]:
```
df.shape[0]
```

Out[8]:
```
25000
```

rows are 2500

In [10]:
```
df.shape[1]
```

Out[10]:
```
19
```

coumns are 19

In [12]:
```
df.isnull()
```

Out[12]:

	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol	Albumin	Copper	All
0	False	False	False	False	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
24995	False	False	False	False	False	False	False	False	False	False	False	False	False	
24996	False	False	False	False	False	False	False	False	False	False	False	False	False	
24997	False	False	False	False	False	False	False	False	False	False	False	False	False	
24998	False	False	False	False	False	False	False	False	False	False	False	False	False	
24999	False	False	False	False	False	False	False	False	False	False	False	False	False	

25000 rows × 19 columns

In [13]:

```
df.isnull().sum()
```

Out[13]:
```
N_Days           0
Status           0
Drug             0
Age              0
Sex              0
Ascites          0
Hepatomegaly     0
Spiders          0
Edema            0
Bilirubin        0
Cholesterol      0
Albumin          0
Copper           0
Alk_Phos         0
SGOT             0
Tryglicerides    0
Platelets        0
Prothrombin      0
Stage            0
dtype: int64
```

In [14]:
```
df = df.dropna()
```

In [15]:
```
df.describe()
```

Out[15]:

	N_Days	Age	Bilirubin	Cholesterol	Albumin	Copper	Alk_Phos	SGOT	Tryglice
count	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000	25000.0
mean	1887.117040	18495.877080	3.402644	372.331471	3.486578	100.184663	1995.675597	123.166345	123.82
std	1091.690918	3737.596616	4.707491	193.668452	0.380488	73.184840	1798.885660	47.747616	52.78
min	41.000000	9598.000000	0.300000	120.000000	1.960000	4.000000	289.000000	26.350000	33.0
25%	1080.000000	15694.000000	0.800000	275.000000	3.290000	52.000000	1032.000000	92.000000	92.0
50%	1680.000000	18499.000000	1.300000	369.510563	3.510000	97.648387	1828.000000	122.556346	124.7
75%	2576.000000	20955.000000	3.400000	369.510563	3.750000	107.000000	1982.655769	134.850000	127.0
max	4795.000000	28650.000000	28.000000	1775.000000	4.640000	588.000000	13862.400000	457.250000	598.0

In [16]:
```
df.columns
```

Out[16]:
```
Index(['N_Days', 'Status', 'Drug', 'Age', 'Sex', 'Ascites', 'Hepatomegaly',
       'Spiders', 'Edema', 'Bilirubin', 'Cholesterol', 'Albumin', 'Copper',
       'Alk_Phos', 'SGOT', 'Tryglicerides', 'Platelets', 'Prothrombin',
       'Stage'],
      dtype='object')
```

In [17]:
```
df.dtypes
```

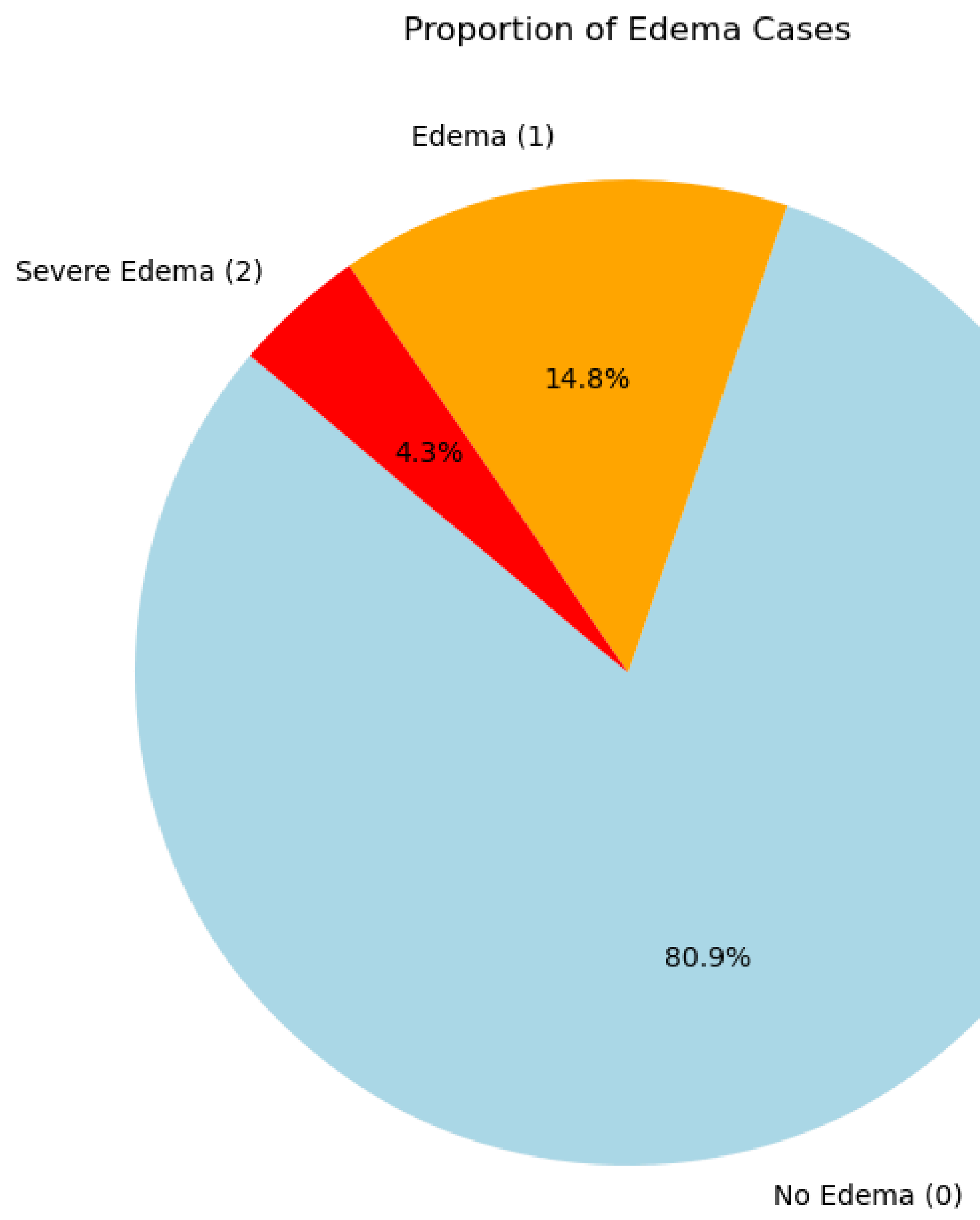
Out[17]:

```
N_Days             int64
Status            object
Drug              object
Age                int64
Sex               object
Ascites           object
Hepatomegaly      object
Spiders           object
Edema             object
Bilirubin        float64
Cholesterol      float64
Albumin          float64
Copper           float64
Alk_Phos         float64
SGOT             float64
Tryglicerides    float64
Platelets        float64
Prothrombin      float64
Stage              int64
dtype: object
```

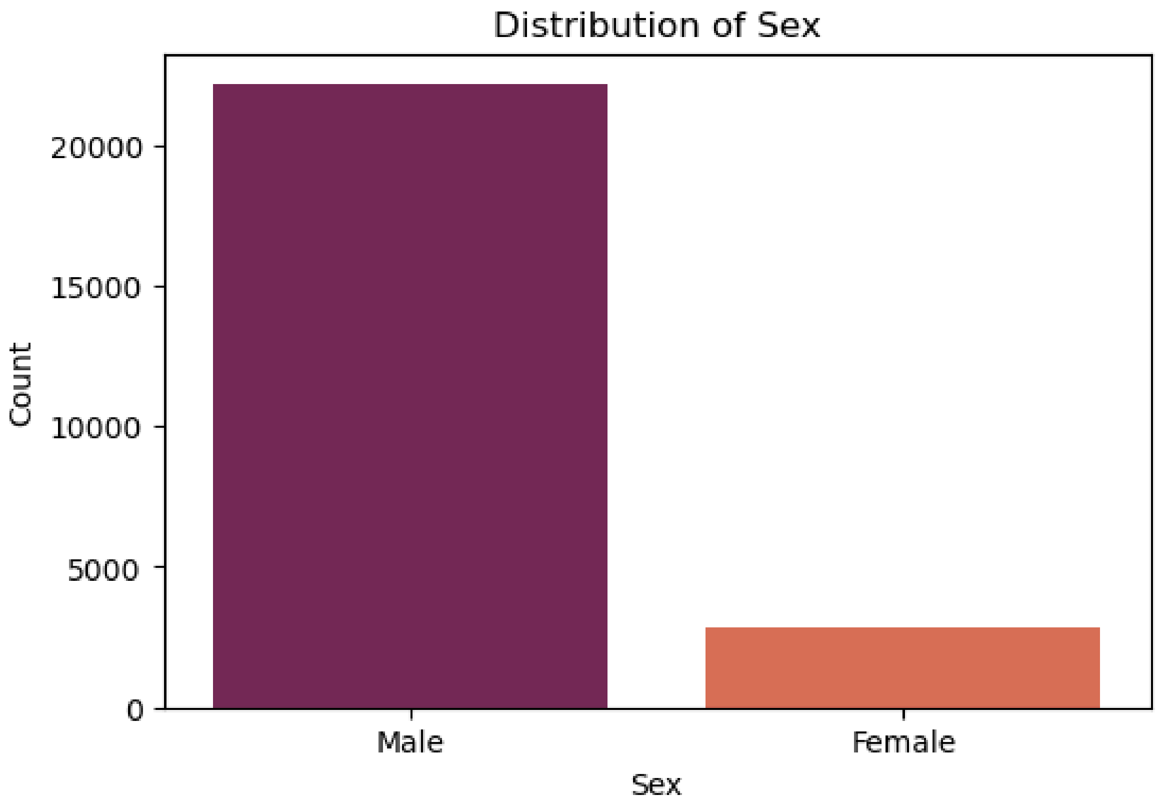
EXPLORATORY DATA ANALIYSIS

In [19]:

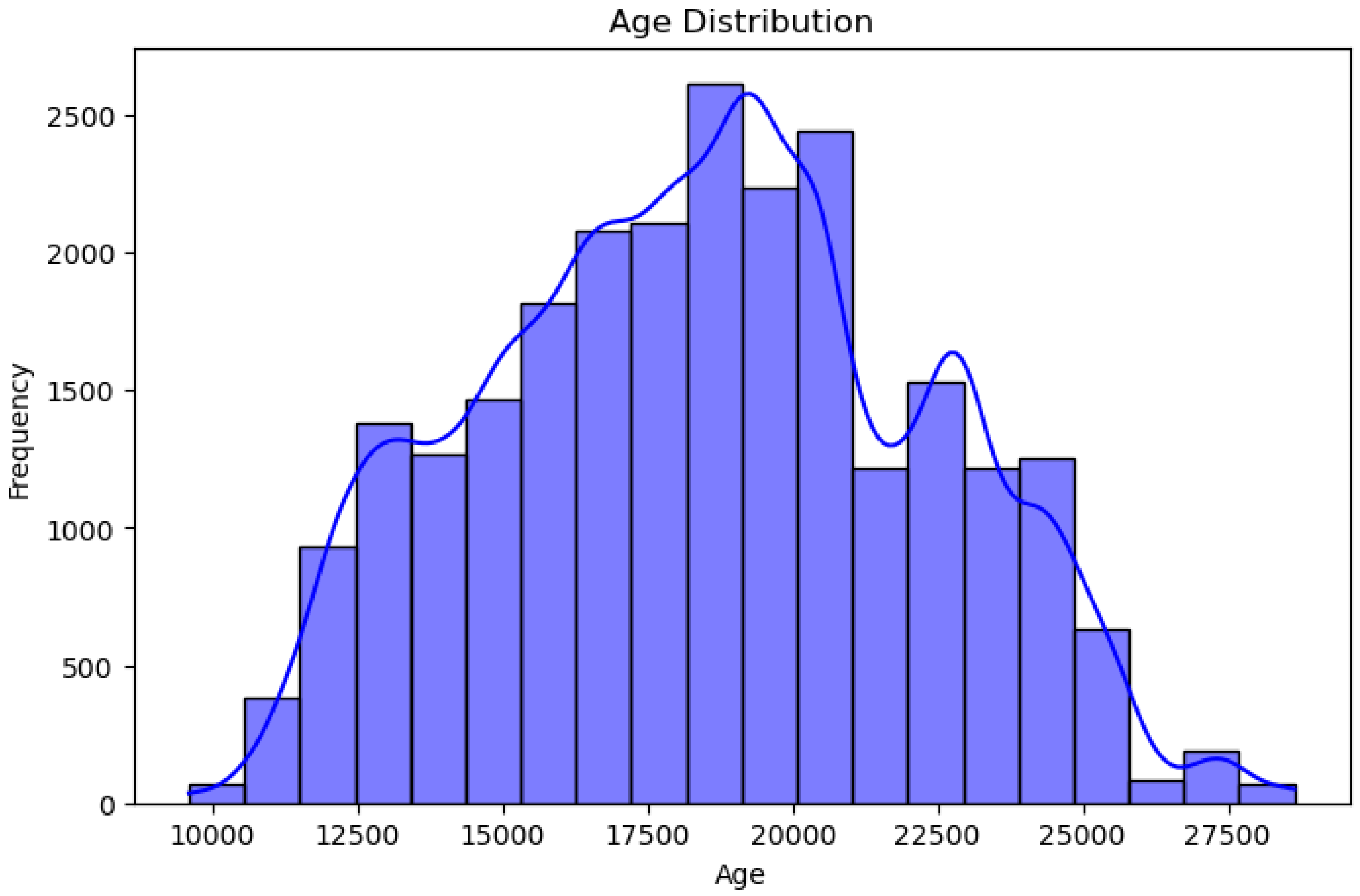
```
edema_counts = df['Edema'].value_counts()
labels = ['No Edema (0)', 'Edema (1)', 'Severe Edema (2)']
plt.figure(figsize=(8,9))
colors = ['lightblue', 'orange', 'red']
plt.pie(edema_counts, labels=labels, autopct='%1.1f%%', colors=colors, startangle=140)
plt.title("Proportion of Edema Cases")
plt.show()
```


Proportion of Edema Cases
Edema (1)
Severe Edema (2)
14.8%
4.3%
80.9%
No Edema (0)

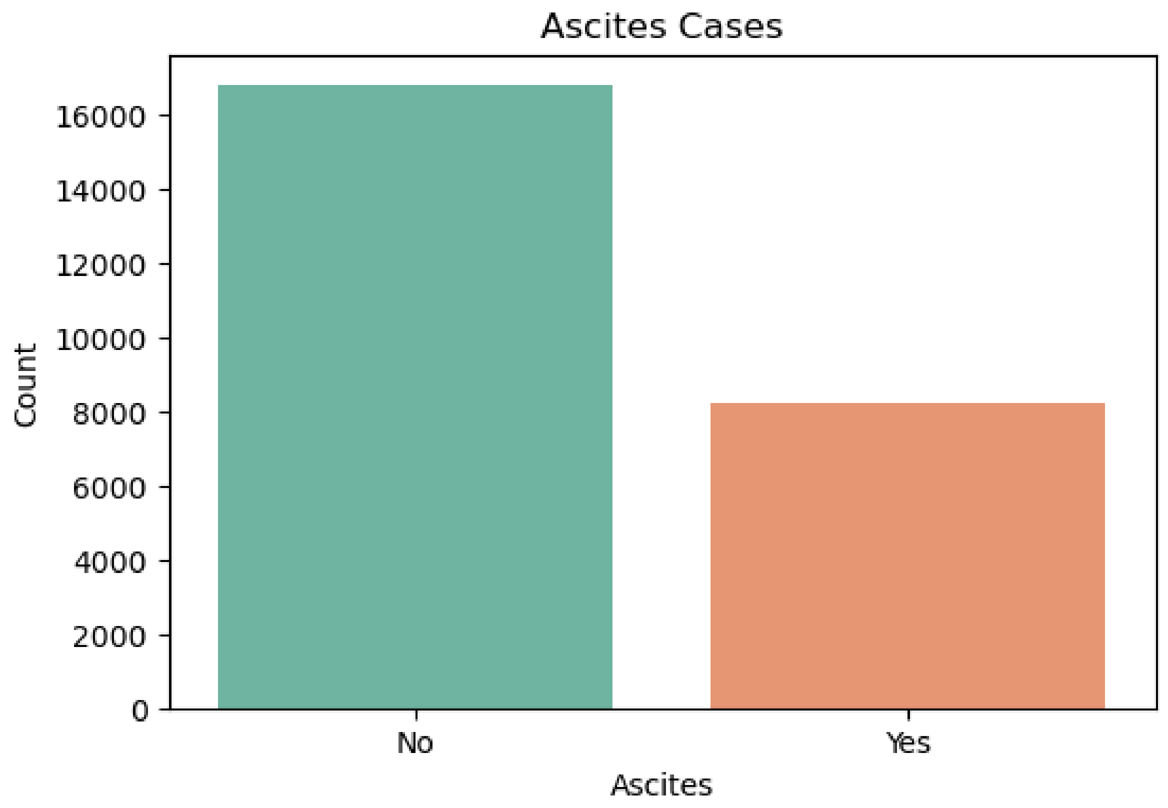
In [20]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='Sex', data=df, palette="rocket")
plt.xticks(ticks=[0, 1], labels=["Male", "Female"])
plt.title("Distribution of Sex")
plt.xlabel("Sex")
plt.ylabel("Count")
plt.show()
```



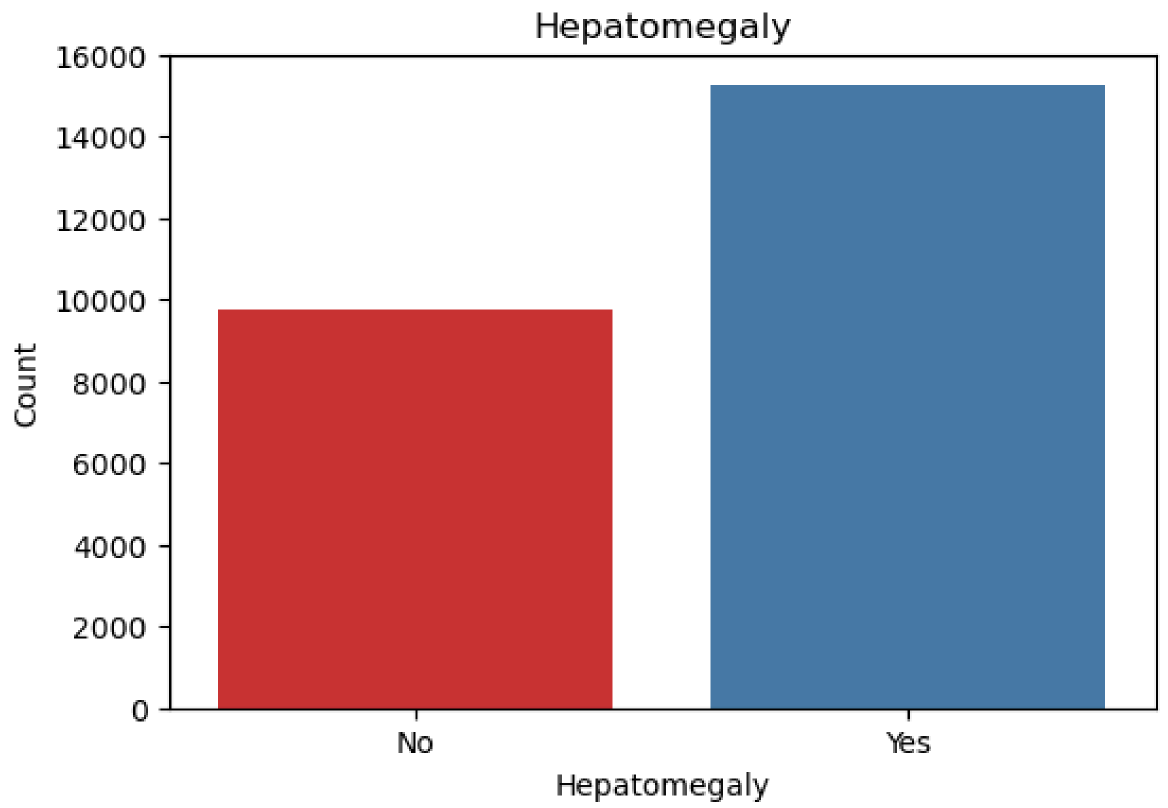
In [21]:
```
plt.figure(figsize=(8, 5))
sns.histplot(df['Age'], bins=20, kde=True, color="blue")
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```



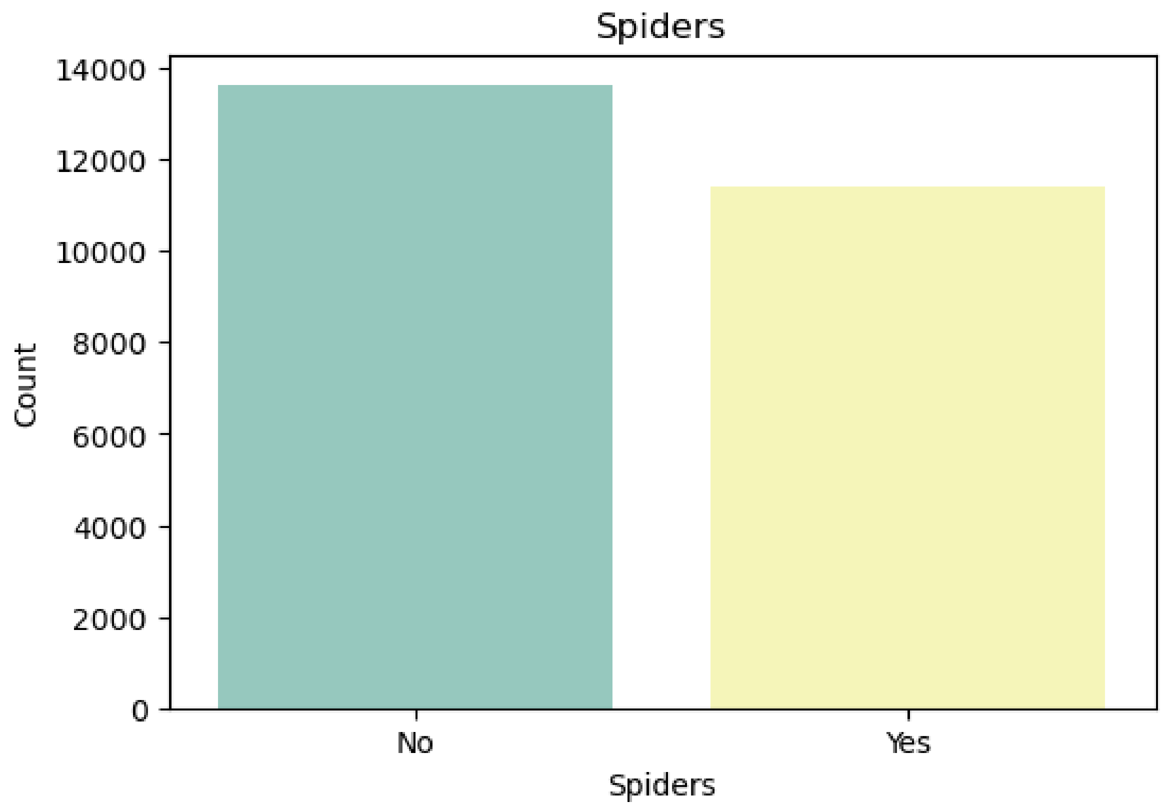
In [22]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='Ascites', data=df, palette="Set2")
plt.xticks(ticks=[0, 1], labels=["No", "Yes"])
plt.title("Ascites Cases")
plt.xlabel("Ascites")
plt.ylabel("Count")
plt.show()
```



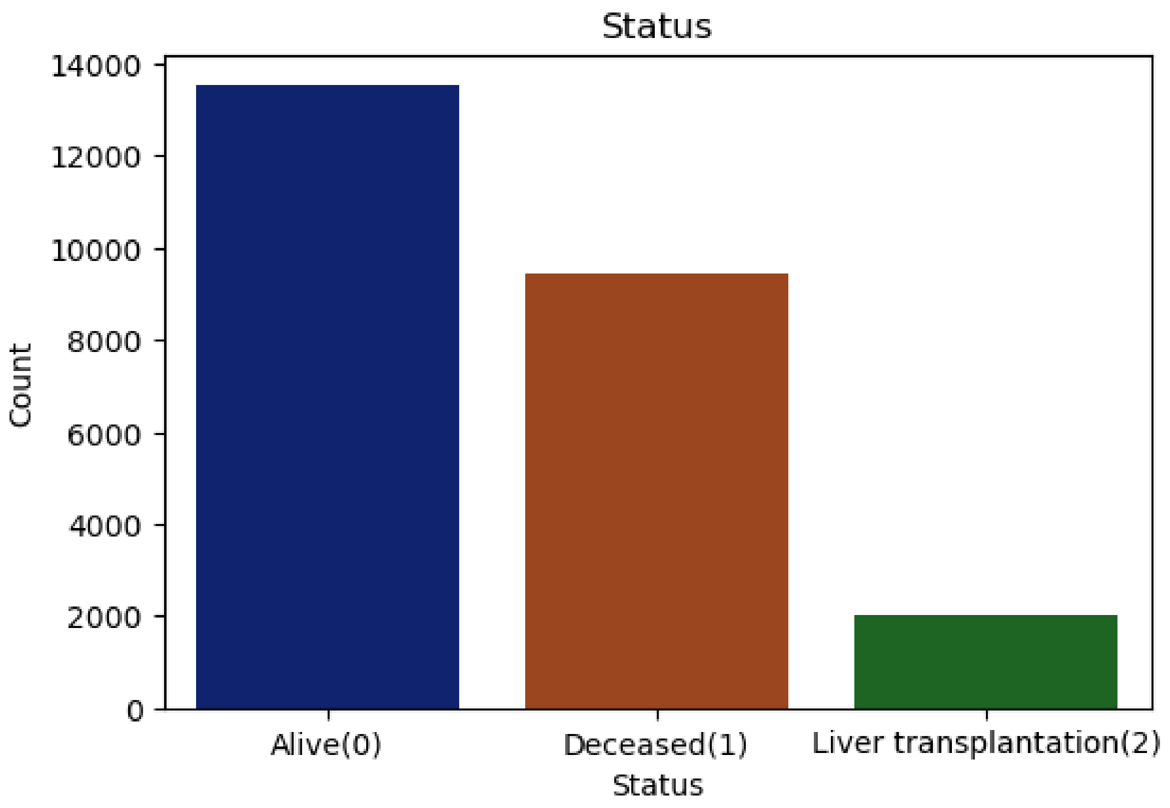
In [23]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='Hepatomegaly', data=df, palette="Set1")
plt.xticks(ticks=[0, 1], labels=["No", "Yes"])
plt.title("Hepatomegaly")
plt.xlabel("Hepatomegaly")
plt.ylabel("Count")
plt.show()
```



In [24]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='Spiders', data=df, palette="Set3")
plt.xticks(ticks=[0, 1], labels=["No", "Yes"])
plt.title("Spiders")
plt.xlabel("Spiders")
plt.ylabel("Count")
plt.show()
```



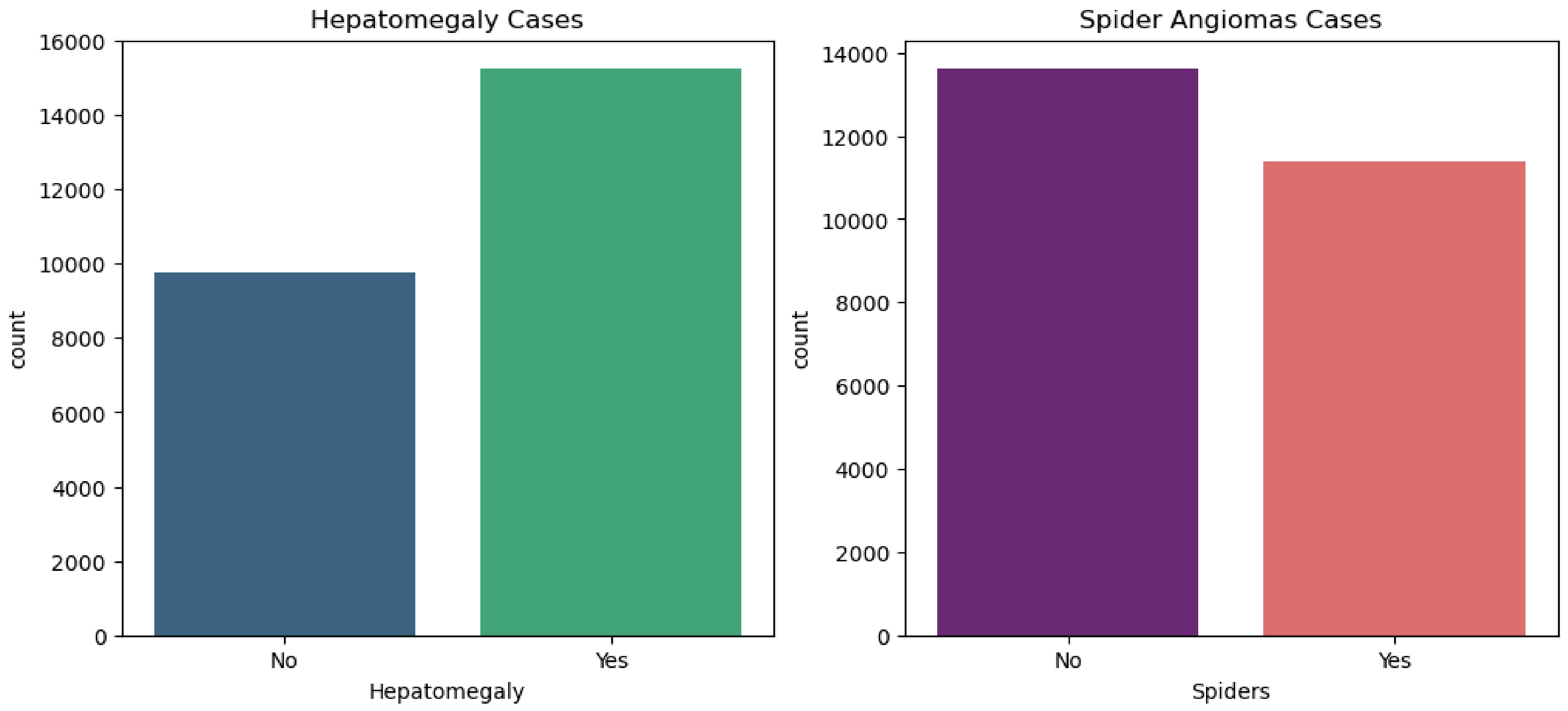
In [25]:
```
status_counts = df['Status'].value_counts()
plt.figure(figsize=(6, 4))
sns.countplot(x='Status', data=df, palette="dark")
plt.xticks(ticks=[0, 1, 2], labels=["Alive(0)", "Deceased(1)", "Liver transplantation(2)"])
plt.title("Status")
plt.xlabel("Status")
plt.ylabel("Count")
plt.show()
```



```
In [26]: fig, axes = plt.subplots(1, 2, figsize=(12, 5))
         sns.countplot(x='Hepatomegaly', data=df, ax=axes[0], palette="viridis")
         axes[0].set_xticks([0, 1])
         axes[0].set_xticklabels(["No", "Yes"])
         axes[0].set_title("Hepatomegaly Cases")

         sns.countplot(x='Spiders', data=df, ax=axes[1], palette="magma")
         axes[1].set_xticks([0, 1])
         axes[1].set_xticklabels(["No", "Yes"])
         axes[1].set_title("Spider Angiomas Cases")
```

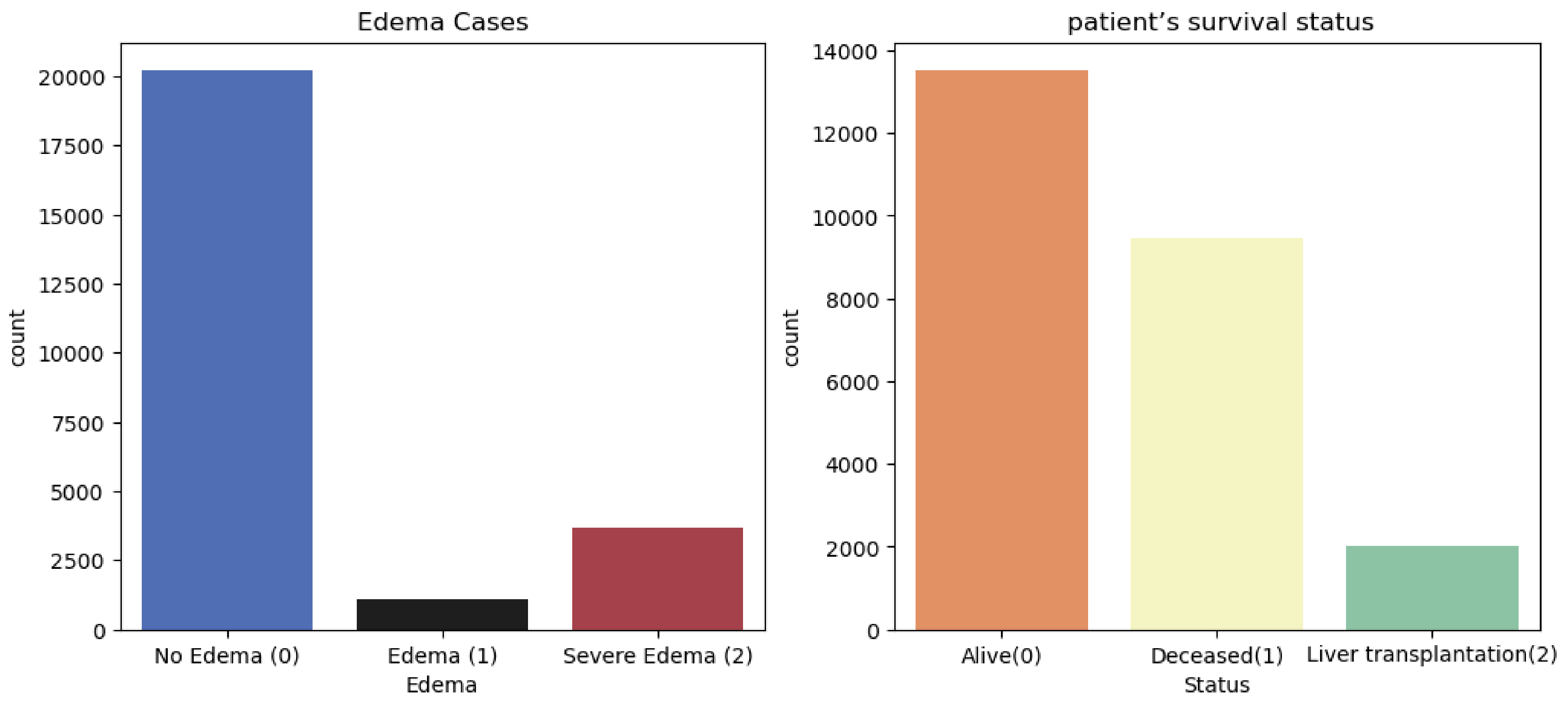
```
Out[26]: Text(0.5, 1.0, 'Spider Angiomas Cases')
```



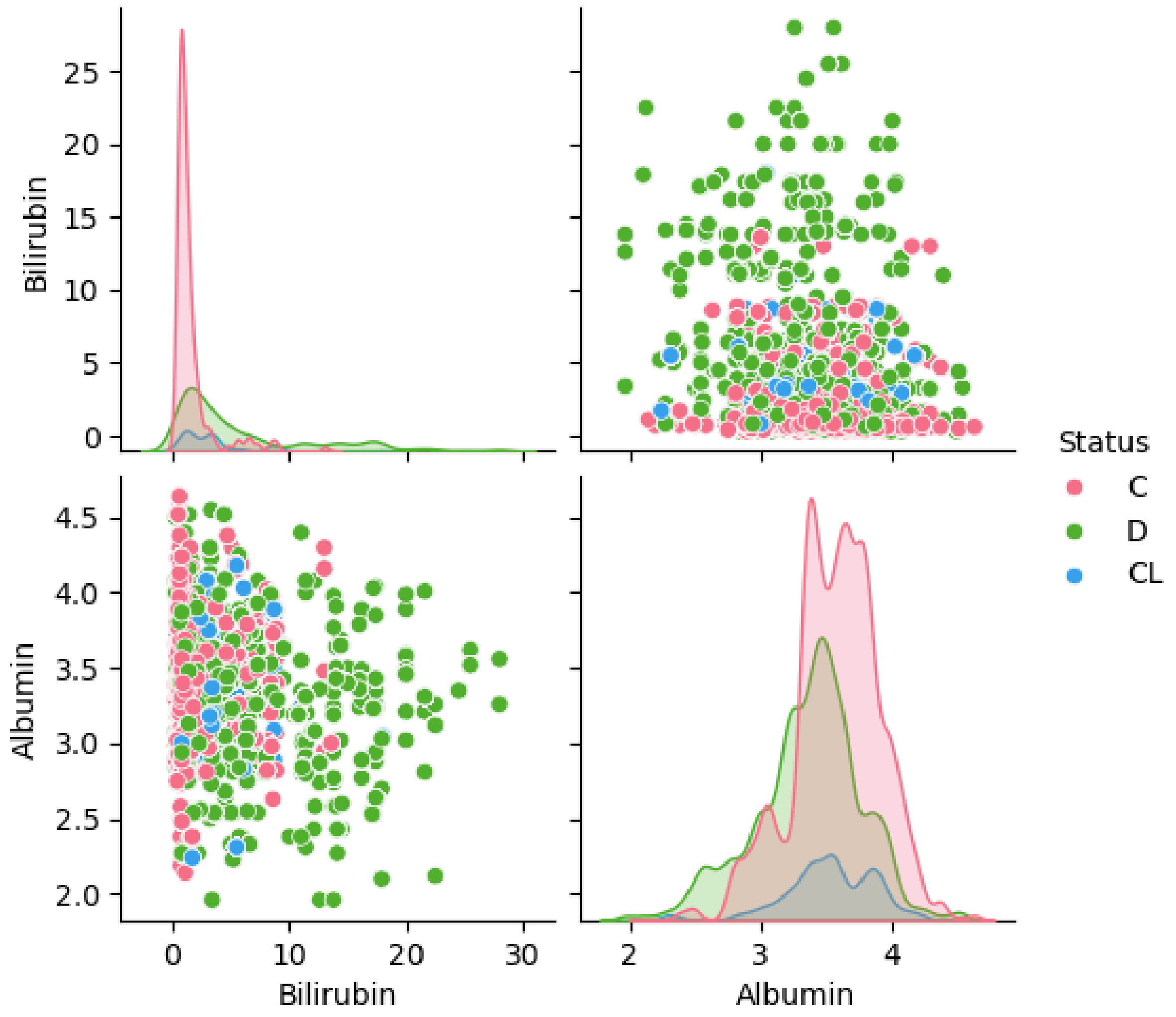
In [27]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
sns.countplot(x='Edema', data=df, ax=axes[0], palette="icefire")
axes[0].set_xticks([0, 1, 2])
axes[0].set_xticklabels(['No Edema (0)', 'Edema (1)', 'Severe Edema (2)'])
axes[0].set_title("Edema Cases")

sns.countplot(x='Status', data=df, ax=axes[1], palette="Spectral")
axes[1].set_xticks([0, 1, 2])
axes[1].set_xticklabels(["Alive(0)", "Deceased(1)", "Liver transplantation(2)"])
axes[1].set_title(" patient’s survival status")
```

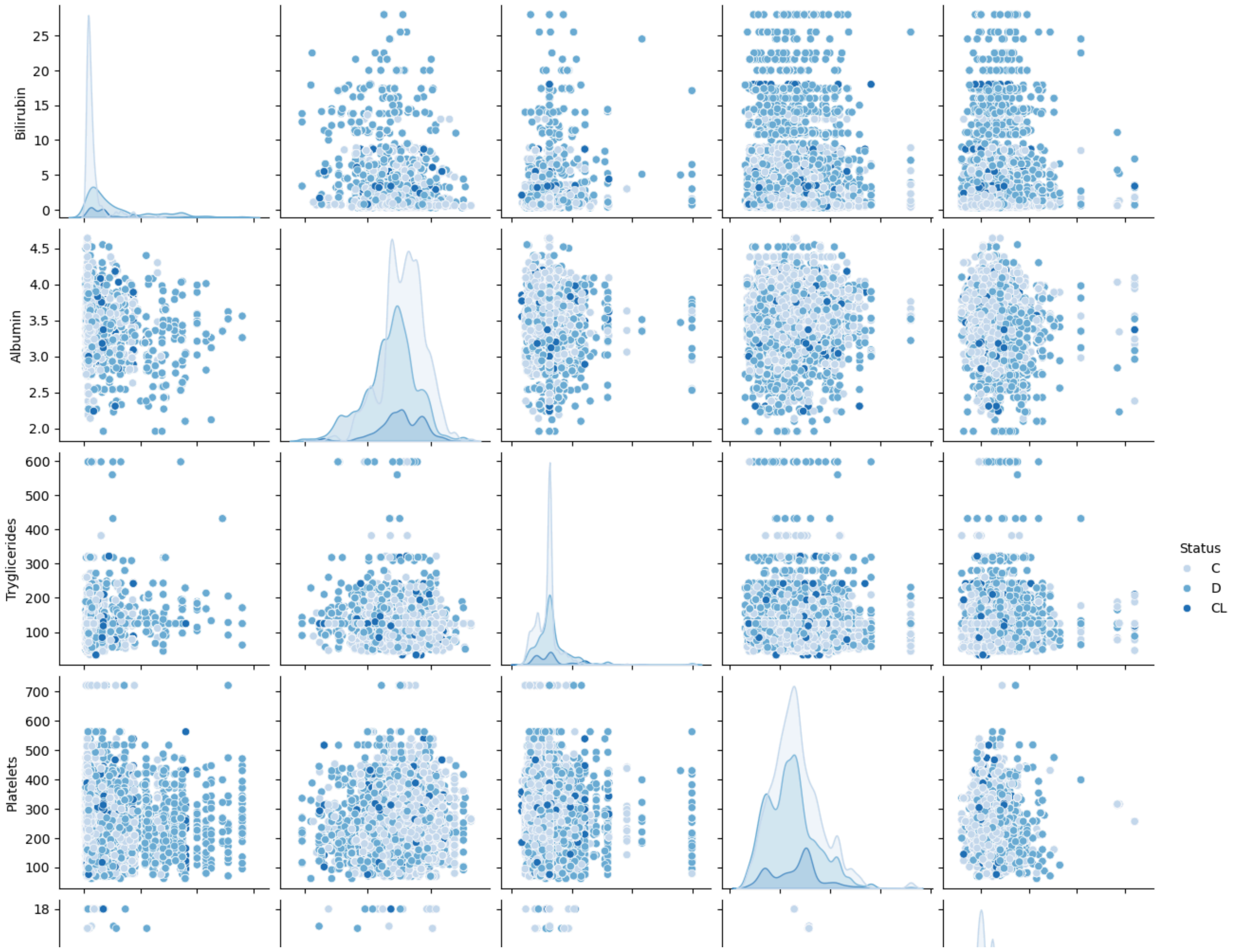
Out[27]:
```
Text(0.5, 1.0, ' patient’s survival status')
```

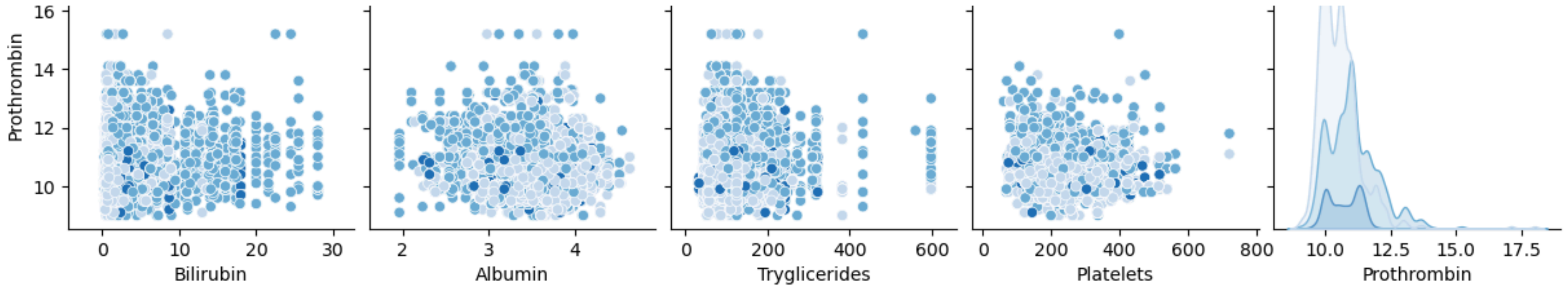


In [28]:
```
sns.pairplot(df, vars=['Bilirubin', 'Albumin'], hue="Status", palette="husl")
plt.show()
```

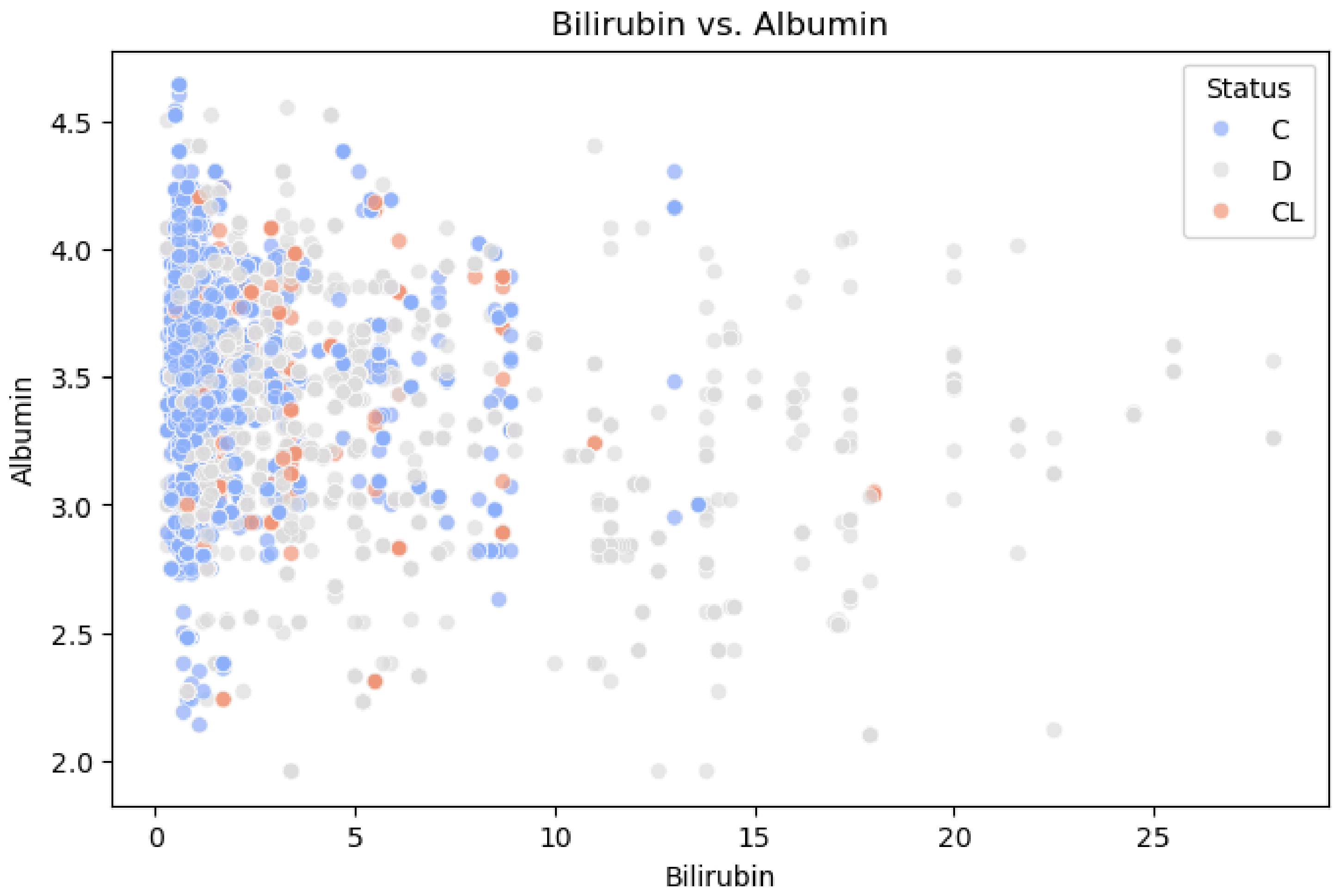


```
In [29]: sns.pairplot(df, vars=['Bilirubin', 'Albumin', 'Tryglicerides', 'Platelets', 'Prothrombin'],
                      hue="Status", palette="Blues")
         plt.show()
```

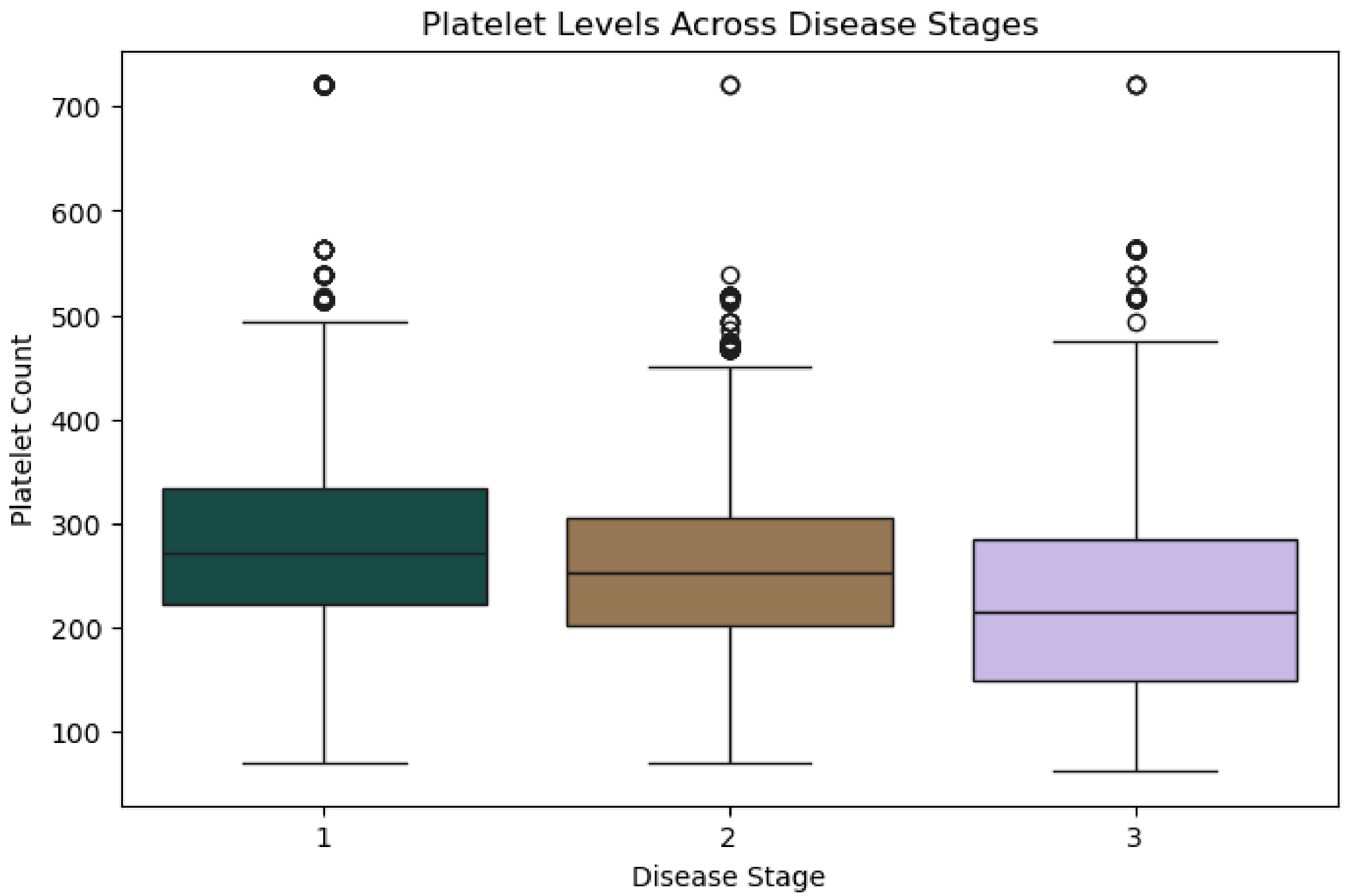

Bilirubin
25
20
15
10
5
0
Albumin
4.5
4.0
3.5
3.0
2.5
2.0
Tryglicerides
600
500
400
300
200
100
Platelets
700
600
500
400
300
200
100
18
Status
C
D
CL



In [30]:
```
plt.figure(figsize=(8, 5))
sns.scatterplot(x='Bilirubin', y='Albumin', data=df, hue='Status', palette="coolwarm", alpha=0.7)
plt.title("Bilirubin vs. Albumin")
plt.xlabel("Bilirubin")
plt.ylabel("Albumin")
plt.show()
```

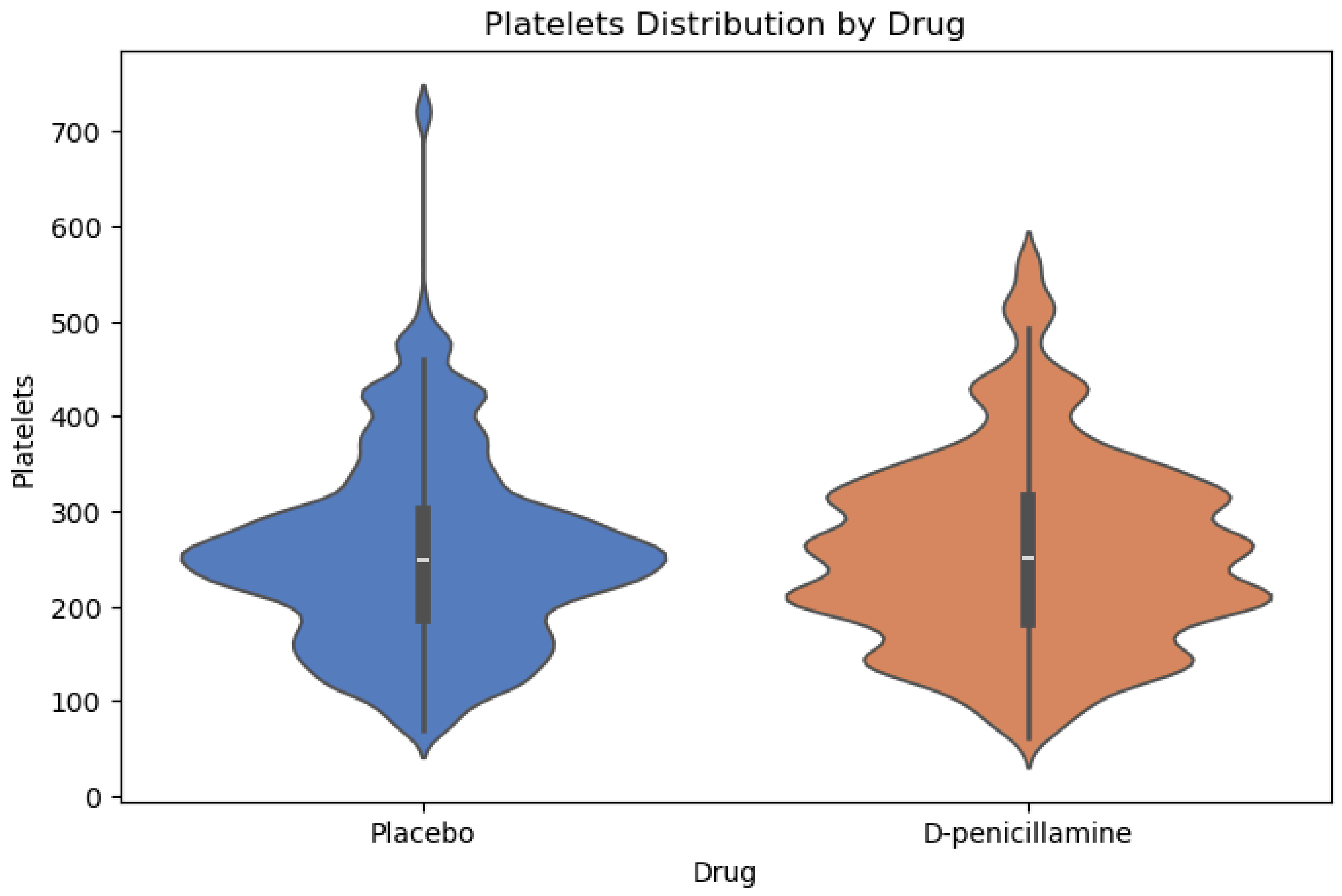


```
In [31]: plt.figure(figsize=(8, 5))
         sns.boxplot(x='Stage', y='Platelets', data=df, palette="cubehelix")
         plt.title("Platelet Levels Across Disease Stages")
         plt.xlabel("Disease Stage")
         plt.ylabel("Platelet Count")
         plt.show()
```

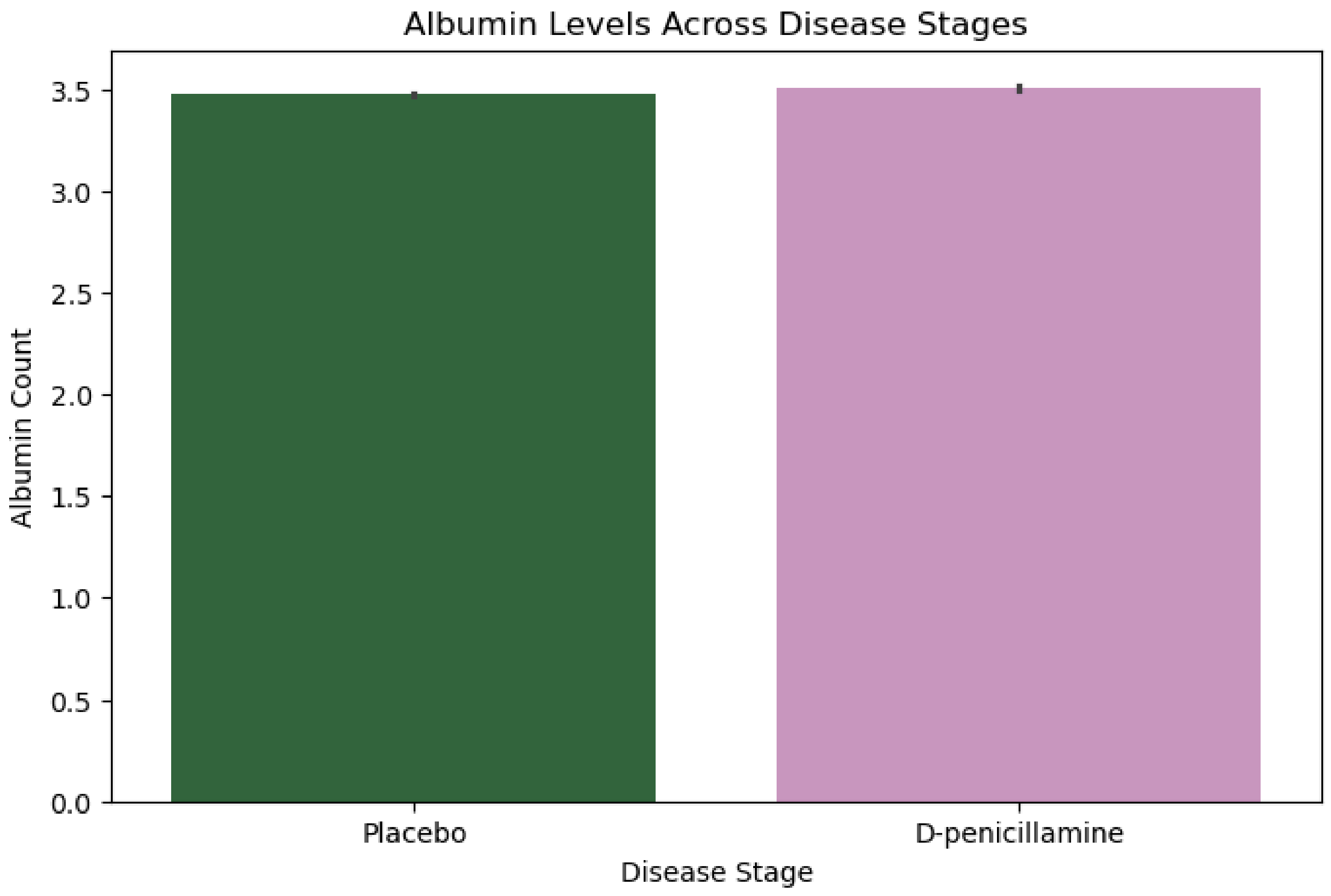


In [32]:

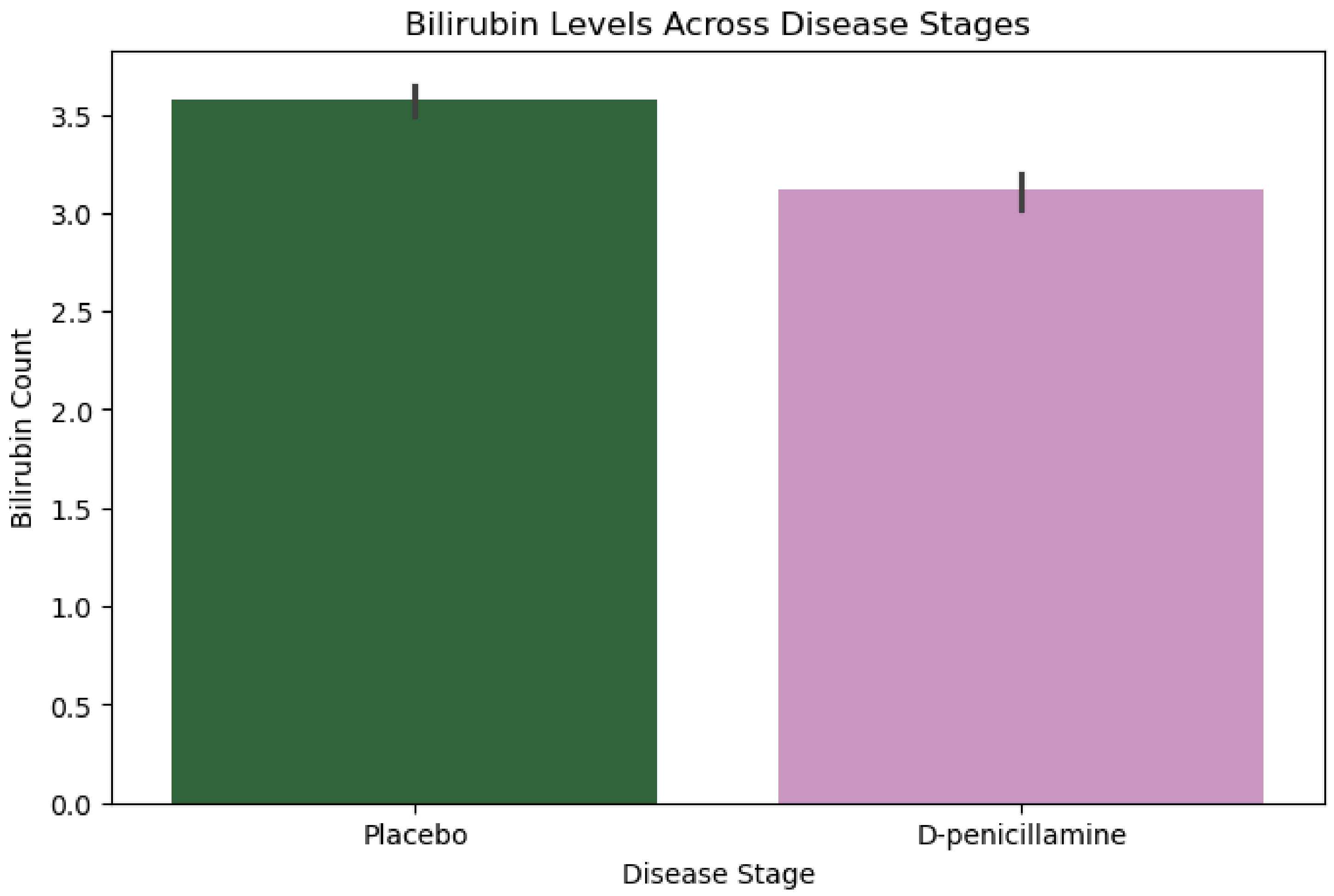
```
plt.figure(figsize=(8, 5))
sns.violinplot(x='Drug', y='Platelets', data=df, palette="muted")
plt.title("Platelets Distribution by Drug")
plt.xlabel("Drug")
plt.ylabel("Platelets")
plt.show()
```



```
In [33]: plt.figure(figsize=(8, 5))
         sns.barplot(x='Drug', y='Albumin', data=df, palette="cubehelix")
         plt.title("Albumin Levels Across Disease Stages")
         plt.xlabel("Disease Stage")
         plt.ylabel("Albumin Count")
         plt.show()
```

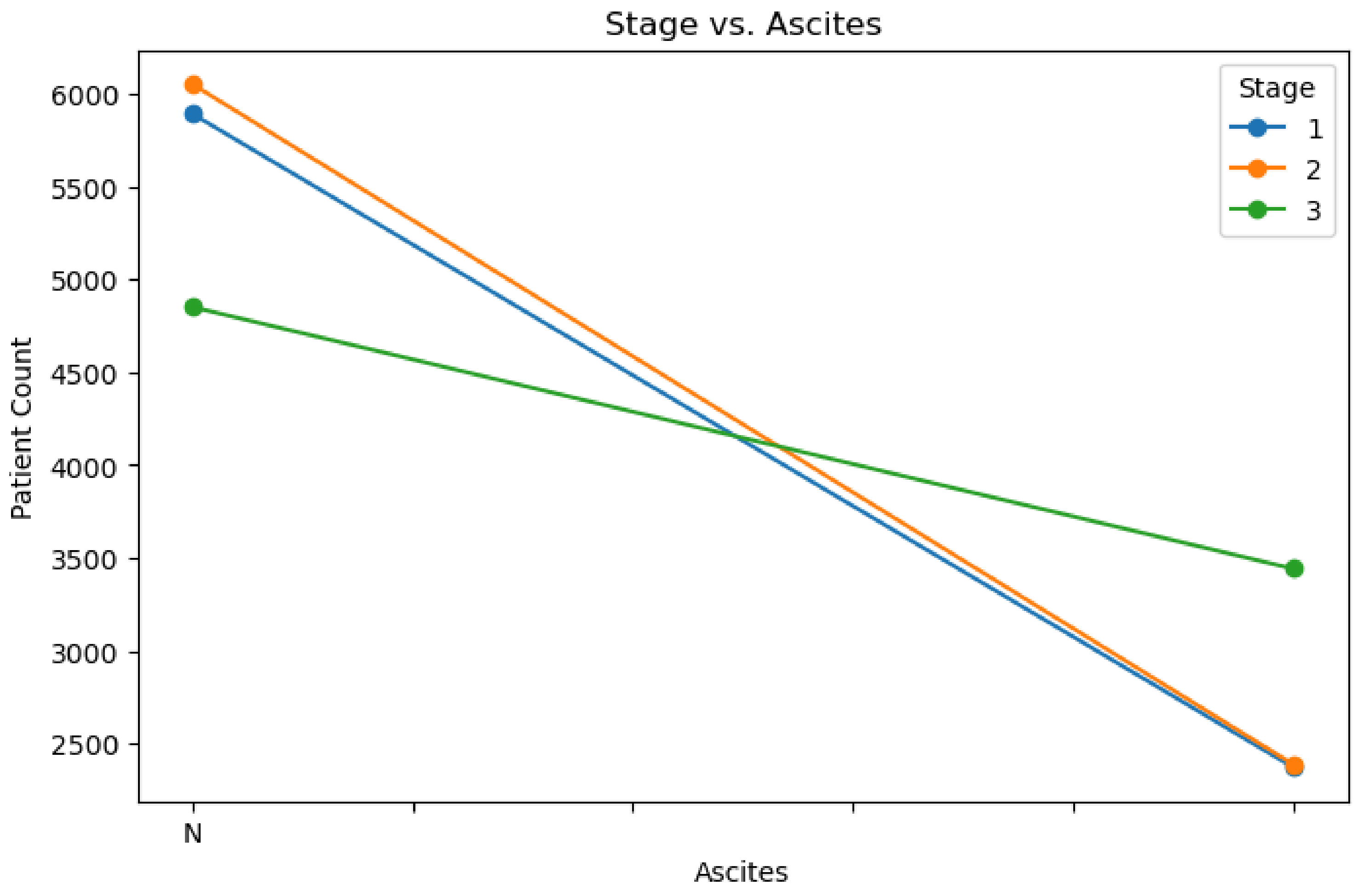


```
In [34]: plt.figure(figsize=(8, 5))
         sns.barplot(x='Drug', y='Bilirubin', data=df, palette="cubehelix")
         plt.title("Bilirubin Levels Across Disease Stages")
         plt.xlabel("Disease Stage")
         plt.ylabel("Bilirubin Count")
         plt.show()
```



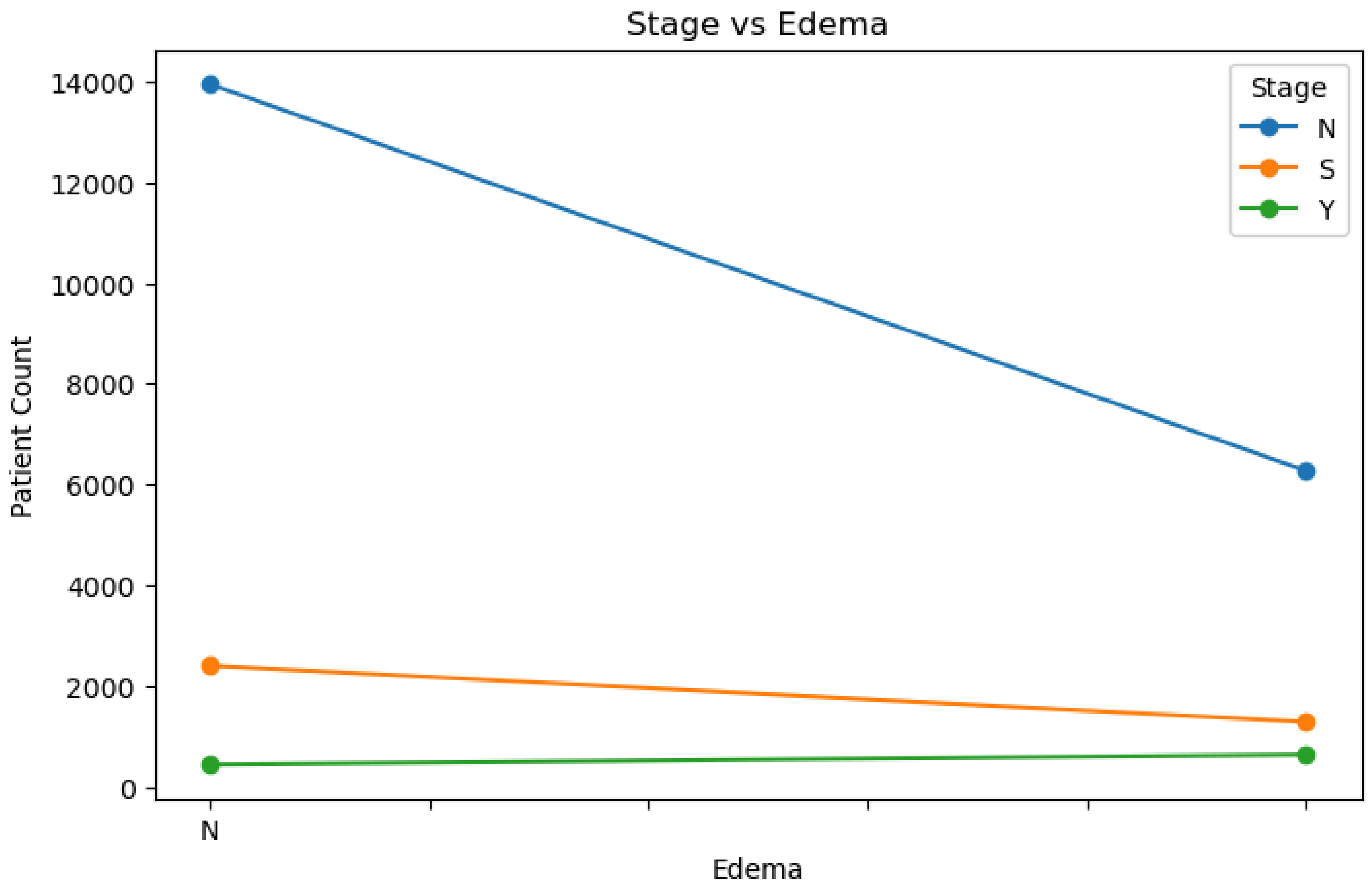
In [35]:
```
plt.figure(figsize=(8, 5))
status_drug_counts = df.groupby(['Ascites', 'Stage']).size().unstack()
status_drug_counts.plot(kind="line", marker="o", figsize=(8, 5))
plt.title("Stage vs. Ascites")
plt.xlabel("Ascites")
plt.ylabel("Patient Count")
plt.legend(title="Stage")
plt.show()
```

```
<Figure size 800x500 with 0 Axes>
```

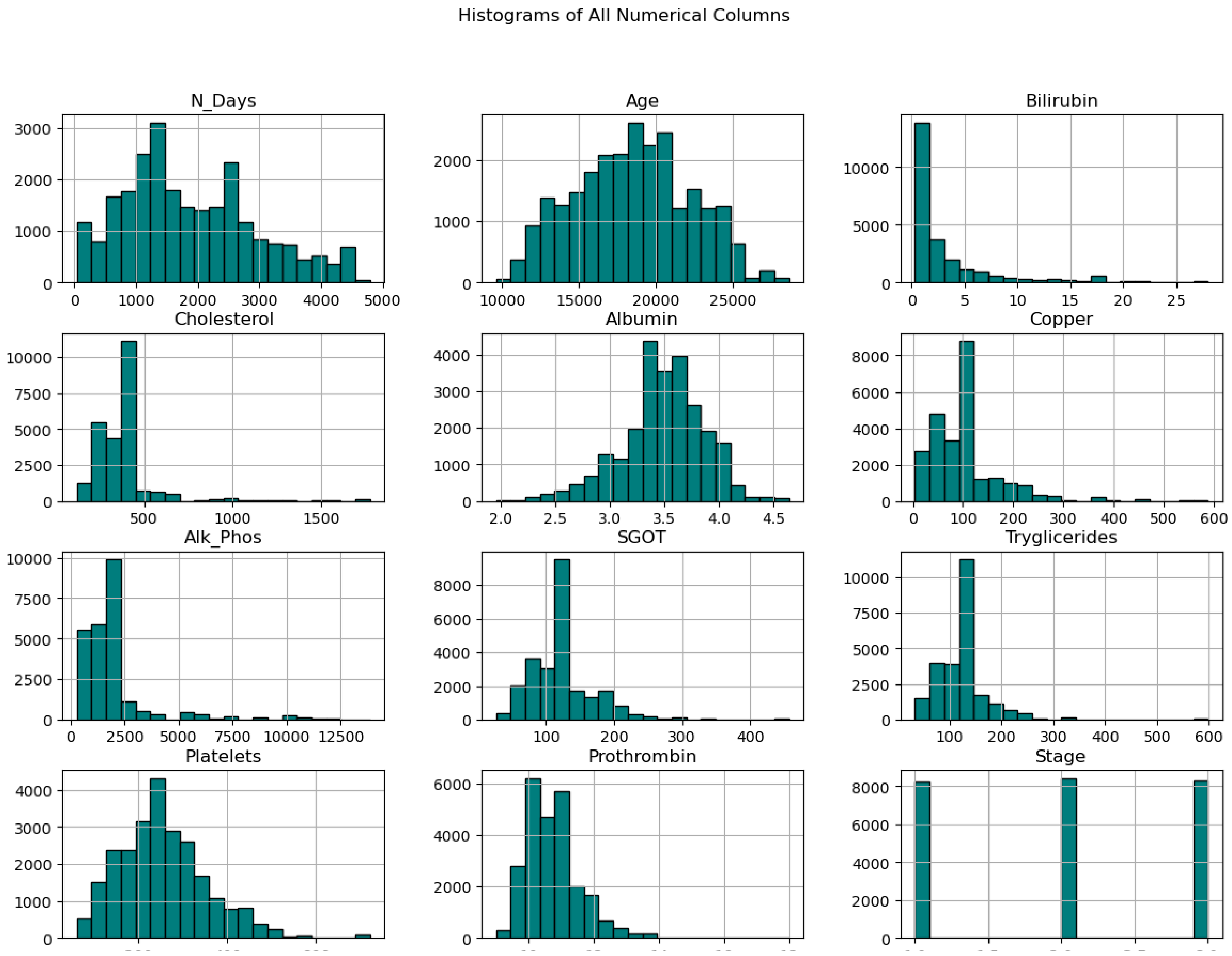


```
In [36]: plt.figure(figsize=(8, 5))
         status_drug_counts = df.groupby(['Ascites', 'Edema']).size().unstack()
         status_drug_counts.plot(kind="line", marker="o", figsize=(8, 5))
         plt.title("Stage vs Edema")
         plt.xlabel("Edema")
         plt.ylabel("Patient Count")
         plt.legend(title="Stage")
         plt.show()
```

```
<Figure size 800x500 with 0 Axes>
```



```
In [37]: df.hist(figsize=(14, 10), bins=20, color="teal", edgecolor="black")
         plt.suptitle("Histograms of All Numerical Columns")
         plt.show()
```


Histograms of All Numerical Columns
N_Days
Age
Bilirubin
Cholesterol
Albumin
Copper
Alk_Phos
SGOT
Tryglicerides
Platelets
Prothrombin
Stage

DATA PREPROCESSING

Feature Encoding in Machine Learning

Feature encoding is the process of converting categorical data into numerical form so that machine learning models can understand and process it. Since many models (like linear regression, SVM, and neural networks) work with numerical values, encoding is crucial when dealing with categorical variables.

In [93]:
```
from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [95]:
```
Cat_Col = []
Num_Col = []
```

In [97]:
```
for col in df.columns:
    if df[col].dtype == 'object':
        Cat_Col.append(col)
    else:
        Num_Col.append(col)
```

In [103…
```
Cat_Col
```

Out[103…
```
['Status', 'Drug', 'Sex', 'Ascites', 'Hepatomegaly', 'Spiders', 'Edema']
```

In [105…
```
Num_Col
```

Out[105…
```
['N_Days',
 'Age',
 'Bilirubin',
 'Cholesterol',
 'Albumin',
 'Copper',
 'Alk_Phos',
 'SGOT',
 'Tryglicerides',
 'Platelets',
 'Prothrombin',
 'Stage']
```

Label Encoding (LE)

Label Encoding is a technique used to convert categorical values into numerical values. It assigns a unique integer (0, 1, 2, ...) to each category in a column.

Machine Learning Algorithms Need Numbers: Most models can’t work with text data directly. They require numerical inputs.

Simple & Efficient: It replaces categories with numbers efficiently, taking up less memory than one-hot encoding.

Useful for Ordered Data: If your categorical values have a meaningful order (e.g., "Low", "Medium", "High"), then label encoding makes sense.

In [109…
```
LE = LabelEncoder()
```

In [113…
```
LE = {}
```

In [138…
```
for col in Cat_Col:
    encoder = LabelEncoder()
    df[col] = encoder.fit_transform(df[col])
    LE[f"{col}_Encoder"] = encoder
    df[col].value_counts()
```

In [140…

```
LE
```

Out[140…

```
{'Status_Encoder': LabelEncoder(),
 'Drug_Encoder': LabelEncoder(),
 'Sex_Encoder': LabelEncoder(),
 'Ascites_Encoder': LabelEncoder(),
 'Hepatomegaly_Encoder': LabelEncoder(),
 'Spiders_Encoder': LabelEncoder(),
 'Edema_Encoder': LabelEncoder()}
```

In [142…

```
df.columns
```

Out[142…

```
Index(['N_Days', 'Status', 'Drug', 'Age', 'Sex', 'Ascites', 'Hepatomegaly',
       'Spiders', 'Edema', 'Bilirubin', 'Cholesterol', 'Albumin', 'Copper',
       'Alk_Phos', 'SGOT', 'Tryglicerides', 'Platelets', 'Prothrombin',
       'Stage'],
      dtype='object')
```

In [144…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   N_Days         25000 non-null  int64  
 1   Status         25000 non-null  int64  
 2   Drug           25000 non-null  int64  
 3   Age            25000 non-null  int64  
 4   Sex            25000 non-null  int64  
 5   Ascites        25000 non-null  int64  
 6   Hepatomegaly   25000 non-null  int64  
 7   Spiders        25000 non-null  int64  
 8   Edema          25000 non-null  int64  
 9   Bilirubin      25000 non-null  float64
 10  Cholesterol    25000 non-null  float64
 11  Albumin        25000 non-null  float64
 12  Copper         25000 non-null  float64
 13  Alk_Phos       25000 non-null  float64
 14  SGOT           25000 non-null  float64
 15  Tryglicerides  25000 non-null  float64
 16  Platelets      25000 non-null  float64
 17  Prothrombin    25000 non-null  float64
 18  Stage          25000 non-null  int64  
dtypes: float64(9), int64(10)
memory usage: 3.6 MB
```

In [146…

```
df.head()
```

Out[146…

	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol	Albumin	Copper	Alk_Pho
0	2221	0	1	18499	0	0	1	0	0	0.5	149.0	4.04	227.0	598
1	1230	0	1	19724	1	1	0	1	0	0.5	219.0	3.93	22.0	663
2	4184	0	1	11839	0	0	0	0	0	0.5	320.0	3.54	51.0	1243
3	2090	2	1	16467	0	0	0	0	0	0.7	255.0	3.74	23.0	1024
4	2105	2	1	21699	0	0	1	0	0	1.9	486.0	3.54	74.0	1052

In [150…

```
df.describe()
```

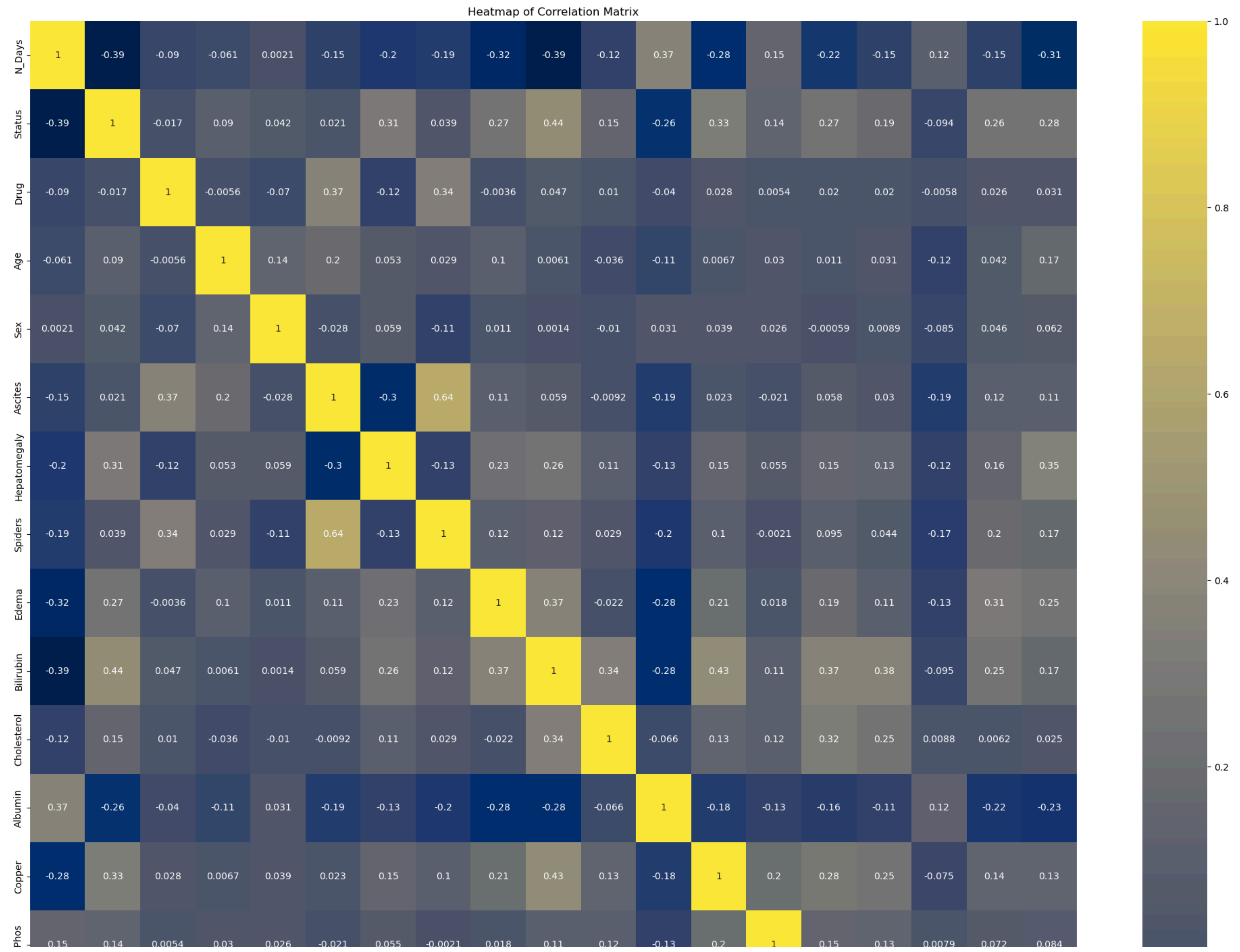
Out[150…

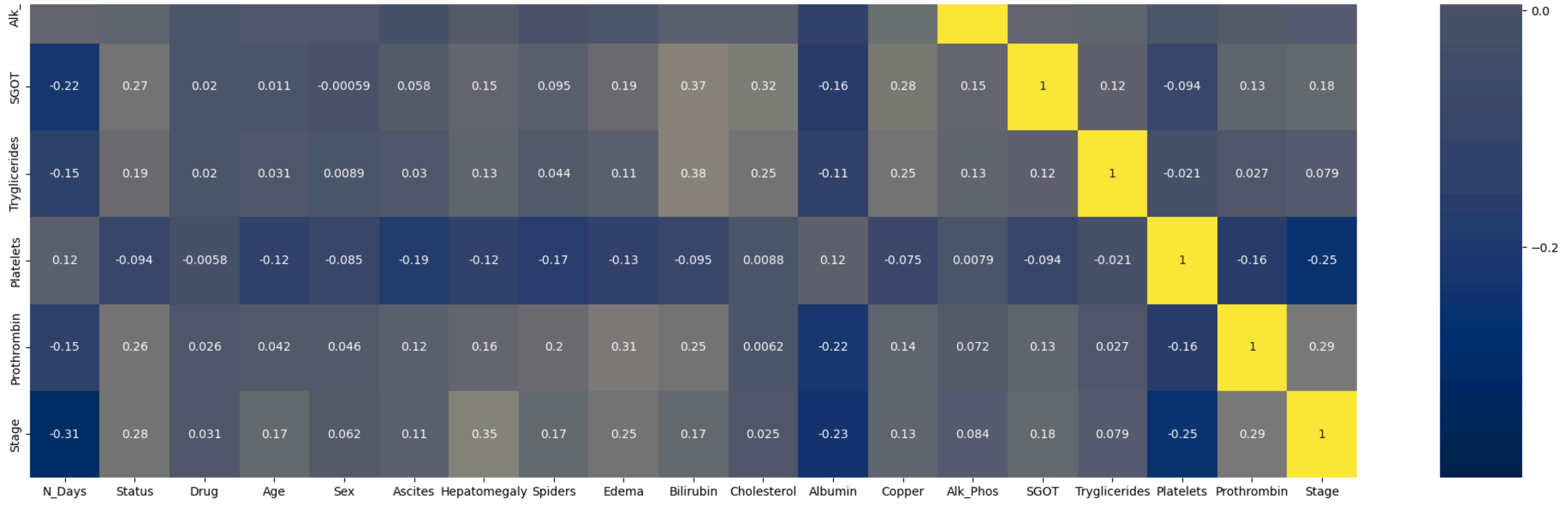
	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Ed
count	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000	25000.00000	25000.0
mean	1887.117040	0.837600	0.633080	18495.877080	0.114520	0.328080	0.390280	0.45544	0.2
std	1091.690918	0.944744	0.481974	3737.596616	0.318448	0.469524	0.487823	0.49802	0.5
min	41.000000	0.000000	0.000000	9598.000000	0.000000	0.000000	0.000000	0.00000	0.0
25%	1080.000000	0.000000	0.000000	15694.000000	0.000000	0.000000	0.000000	0.00000	0.0
50%	1680.000000	0.000000	1.000000	18499.000000	0.000000	0.000000	0.000000	0.00000	0.0
75%	2576.000000	2.000000	1.000000	20955.000000	0.000000	1.000000	1.000000	1.00000	0.0
max	4795.000000	2.000000	1.000000	28650.000000	1.000000	1.000000	1.000000	1.00000	2.0

In [196…

```
plt.figure(figsize=(25,25))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='cividis')
```

```
plt.title('Heatmap of Correlation Matrix')
plt.show()
```


Heatmap of Correlation Matrix
N_Days 1 -0.39 -0.09 -0.061 0.0021 -0.15 -0.2 -0.19 -0.32 -0.39 -0.12 0.37 -0.28 0.15 -0.22 -0.15 0.12 -0.15 -0.31
Status -0.39 1 -0.017 0.09 0.042 0.021 0.31 0.039 0.27 0.44 0.15 -0.26 0.33 0.14 0.27 0.19 -0.094 0.26 0.28
Drug -0.09 -0.017 1 -0.0056 -0.07 0.37 -0.12 0.34 -0.0036 0.047 0.01 -0.04 0.028 0.0054 0.02 0.02 -0.0058 0.026 0.031
Age -0.061 0.09 -0.0056 1 0.14 0.2 0.053 0.029 0.1 0.0061 -0.036 -0.11 0.0067 0.03 0.011 0.031 -0.12 0.042 0.17
Sex 0.0021 0.042 -0.07 0.14 1 -0.028 0.059 -0.11 0.011 0.0014 -0.01 0.031 0.039 0.026 -0.00059 0.0089 -0.085 0.046 0.062
Ascites -0.15 0.021 0.37 0.2 -0.028 1 -0.3 0.64 0.11 0.059 -0.0092 -0.19 0.023 -0.021 0.058 0.03 -0.19 0.12 0.11
Hepatomegaly -0.2 0.31 -0.12 0.053 0.059 -0.3 1 -0.13 0.23 0.26 0.11 -0.13 0.15 0.055 0.15 0.13 -0.12 0.16 0.35
Spiders -0.19 0.039 0.34 0.029 -0.11 0.64 -0.13 1 0.12 0.12 0.029 -0.2 0.1 -0.0021 0.095 0.044 -0.17 0.2 0.17
Edema -0.32 0.27 -0.0036 0.1 0.011 0.11 0.23 0.12 1 0.37 -0.022 -0.28 0.21 0.018 0.19 0.11 -0.13 0.31 0.25
Bilirubin -0.39 0.44 0.047 0.0061 0.0014 0.059 0.26 0.12 0.37 1 0.34 -0.28 0.43 0.11 0.37 0.38 -0.095 0.25 0.17
Cholesterol -0.12 0.15 0.01 -0.036 -0.01 -0.0092 0.11 0.029 -0.022 0.34 1 -0.066 0.13 0.12 0.32 0.25 0.0088 0.0062 0.025
Albumin 0.37 -0.26 -0.04 -0.11 0.031 -0.19 -0.13 -0.2 -0.28 -0.28 -0.066 1 -0.18 -0.13 -0.16 -0.11 0.12 -0.22 -0.23
Copper -0.28 0.33 0.028 0.0067 0.039 0.023 0.15 0.1 0.21 0.43 0.13 -0.18 1 0.2 0.28 0.25 -0.075 0.14 0.13
Phos 0.15 0.14 0.0054 0.03 0.026 -0.021 0.055 -0.0021 0.018 0.11 0.12 -0.13 0.2 1 0.15 0.13 0.0079 0.072 0.084
1.0
0.8
0.6
0.4
0.2



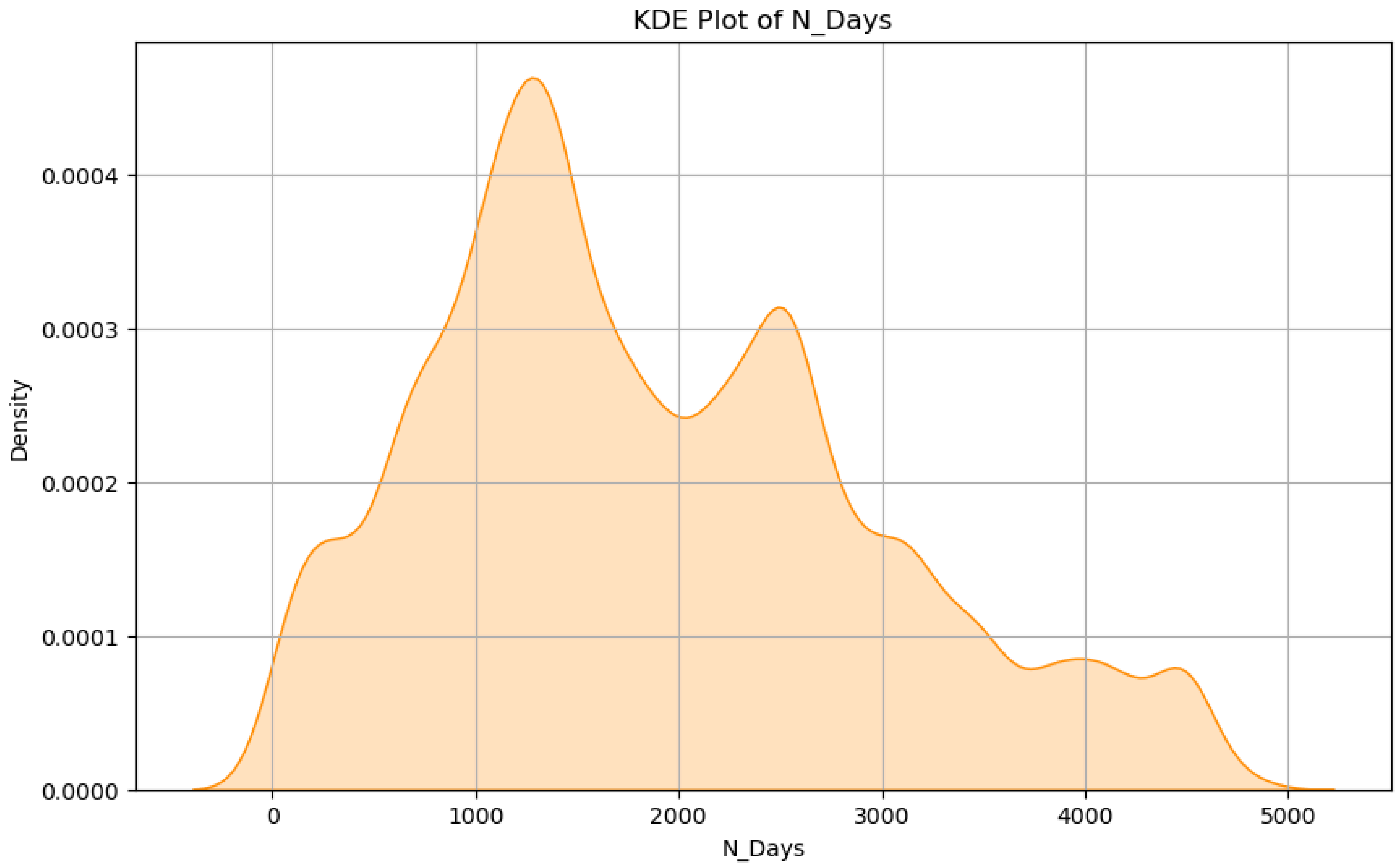
In [156…

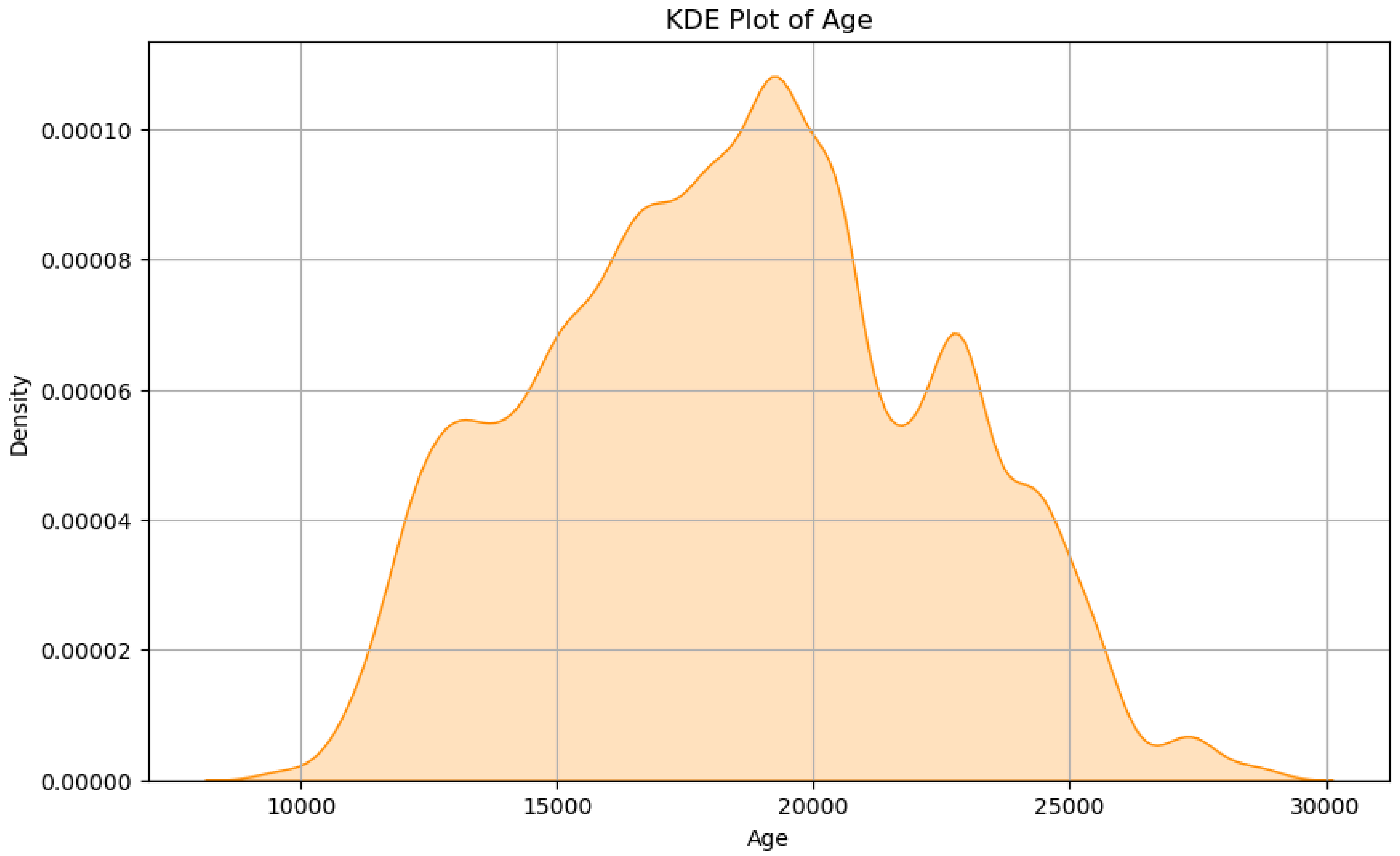
```
print(df.dtypes)
```

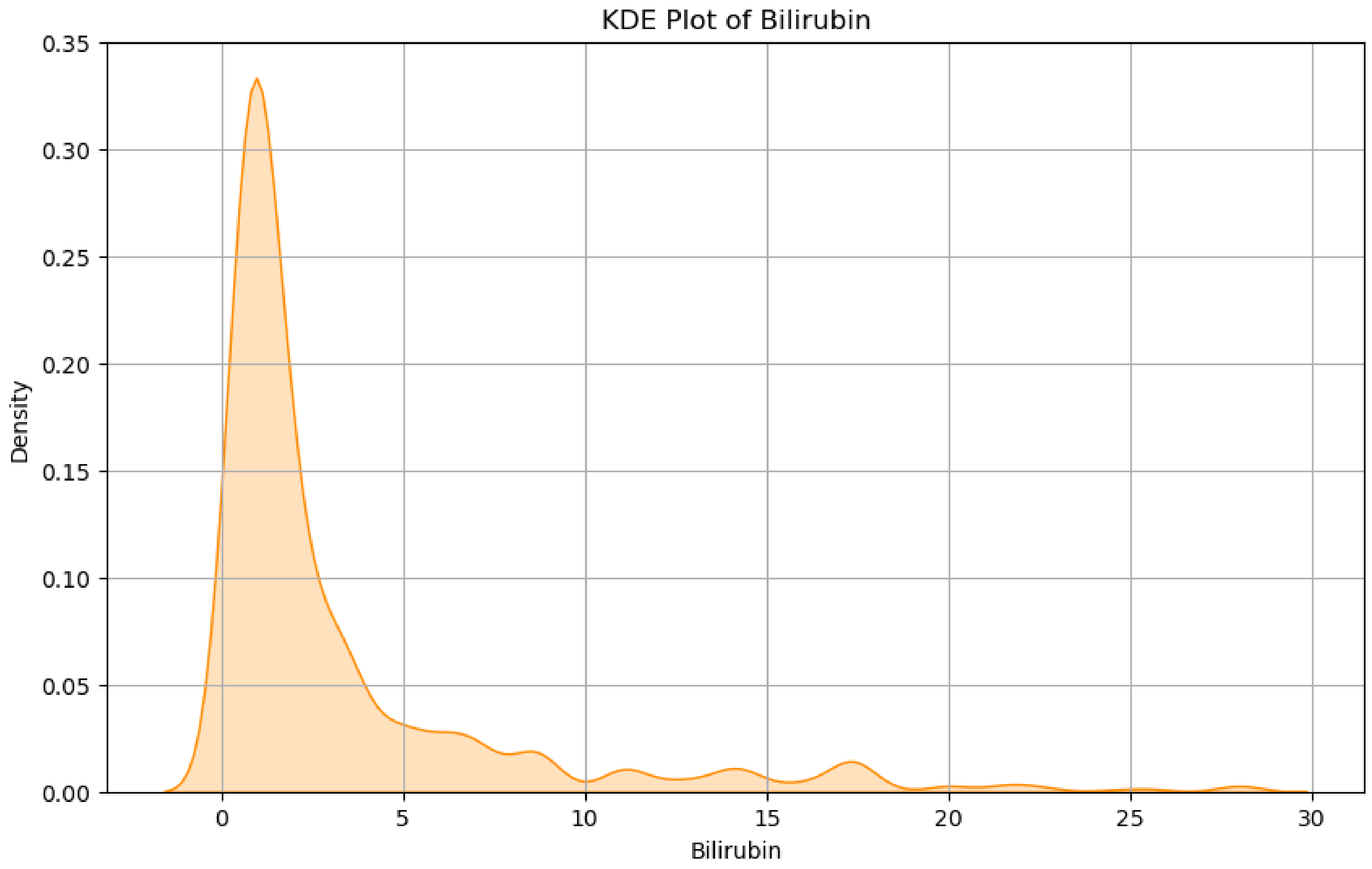
```
N_Days             int64
Status             int64
Drug               int64
Age                int64
Sex                int64
Ascites            int64
Hepatomegaly       int64
Spiders            int64
Edema              int64
Bilirubin        float64
Cholesterol      float64
Albumin          float64
Copper           float64
Alk_Phos         float64
SGOT             float64
Tryglicerides    float64
Platelets        float64
Prothrombin      float64
Stage              int64
dtype: object
```

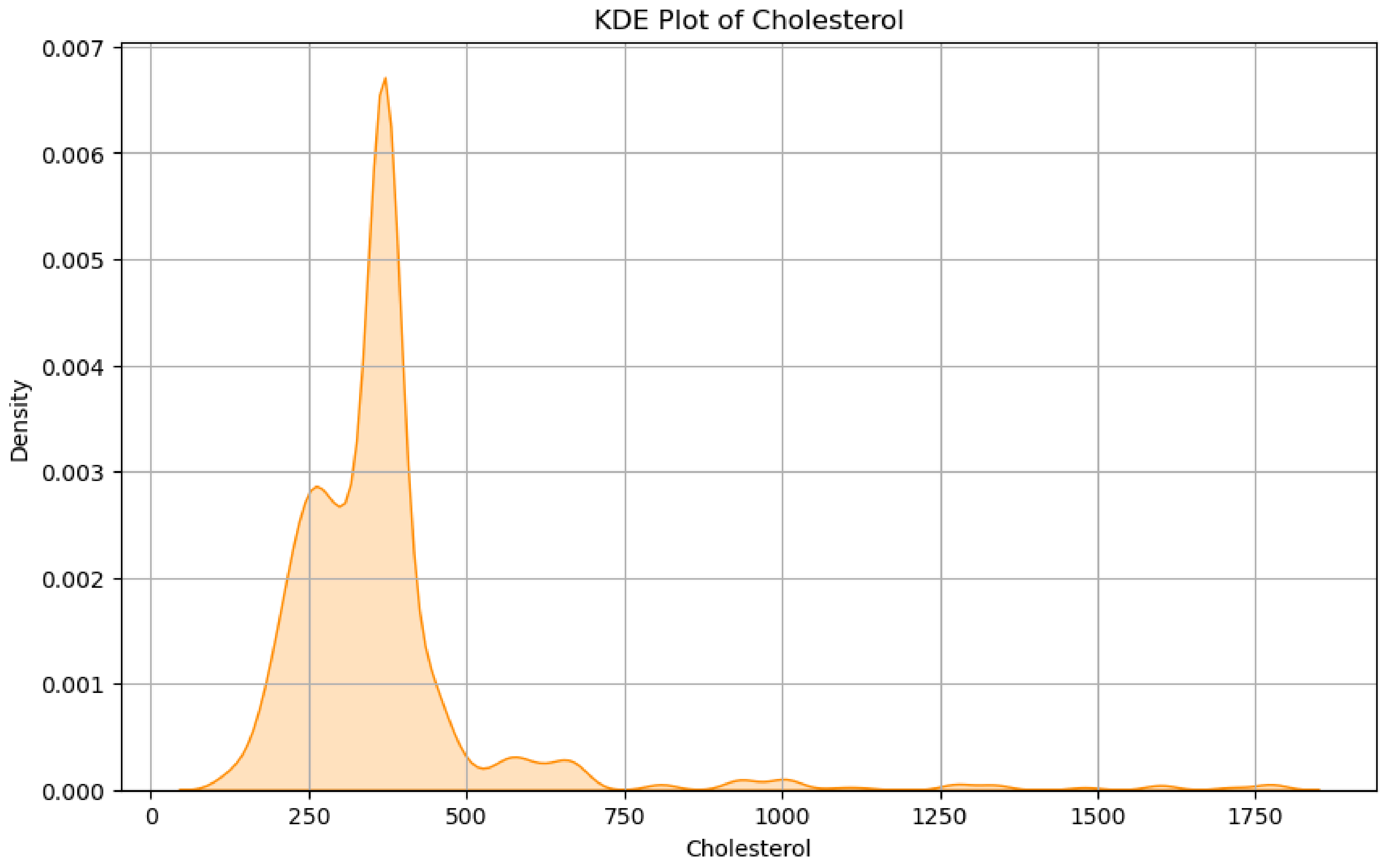
In [220…

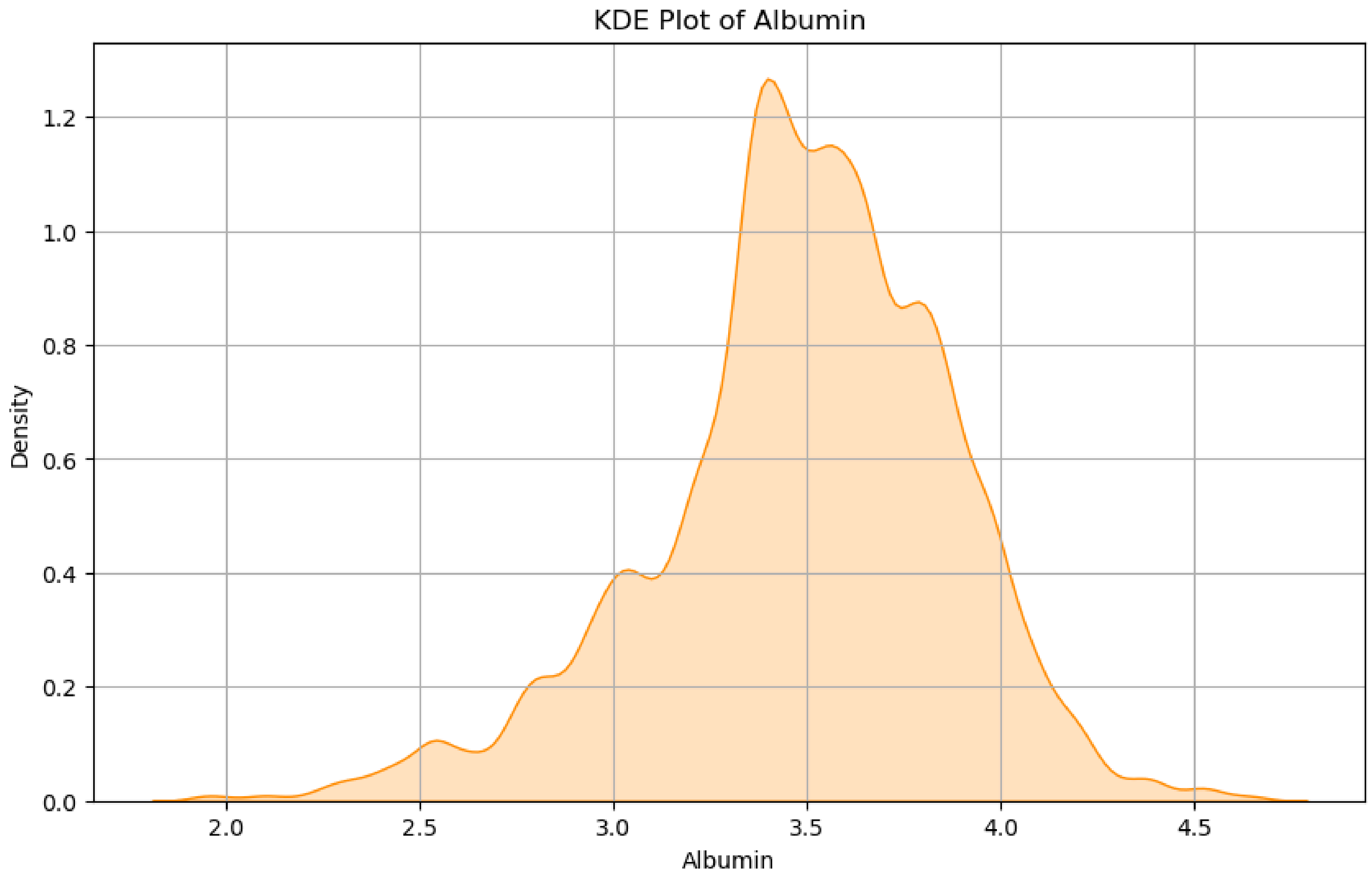
```
for col in Num_Col:
    plt.figure(figsize=(10, 6))
    sns.kdeplot(data=df, x=col, fill=True, color='darkorange', bw_adjust=1)
    plt.title(f'KDE Plot of {col}')
    plt.xlabel(col)
    plt.ylabel('Density')
    plt.grid(True)
    plt.show()
```

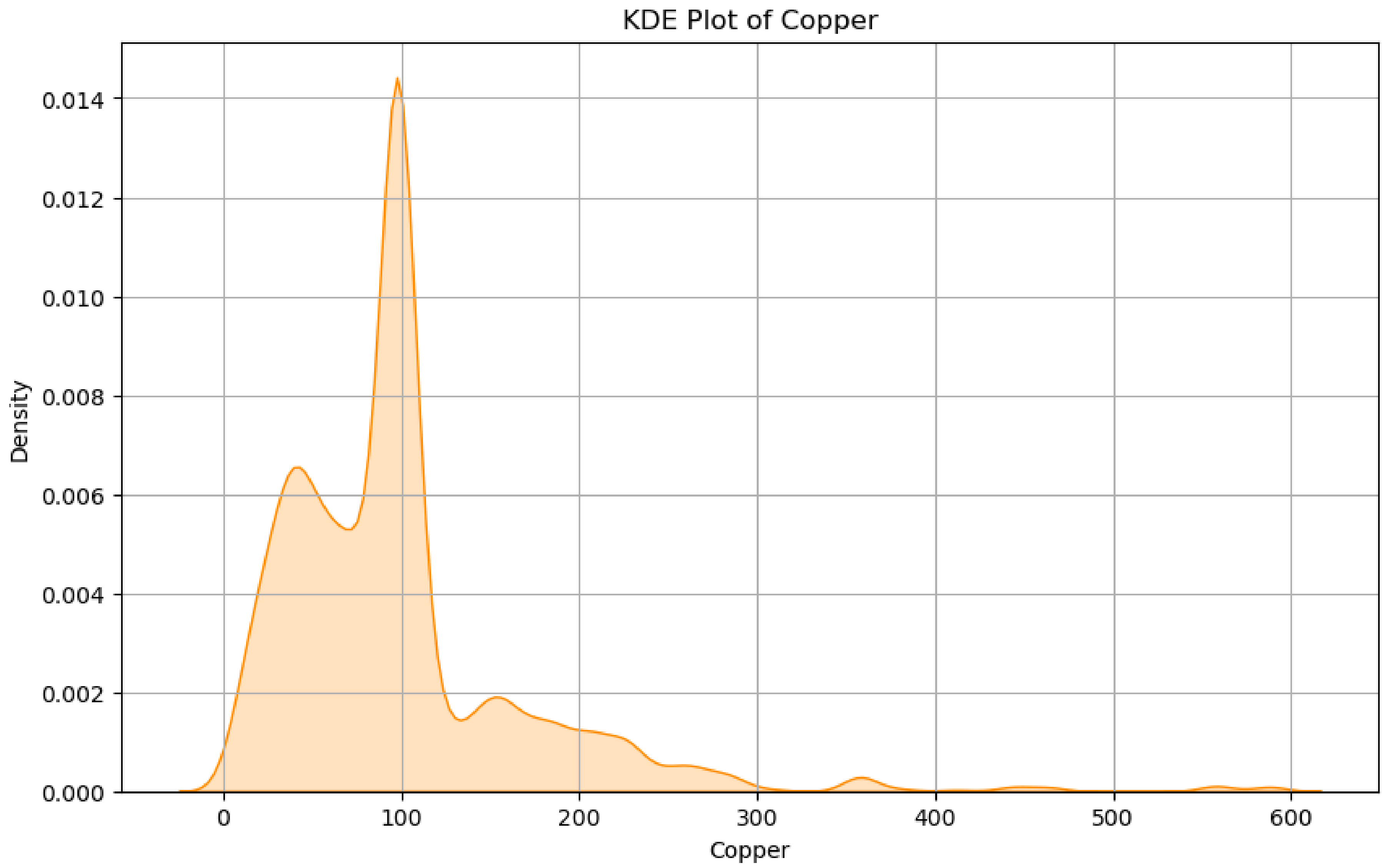

KDE Plot of N_Days
Density
0.0000
0.0001
0.0002
0.0003
0.0004
0
1000
2000
3000
4000
5000
N_Days

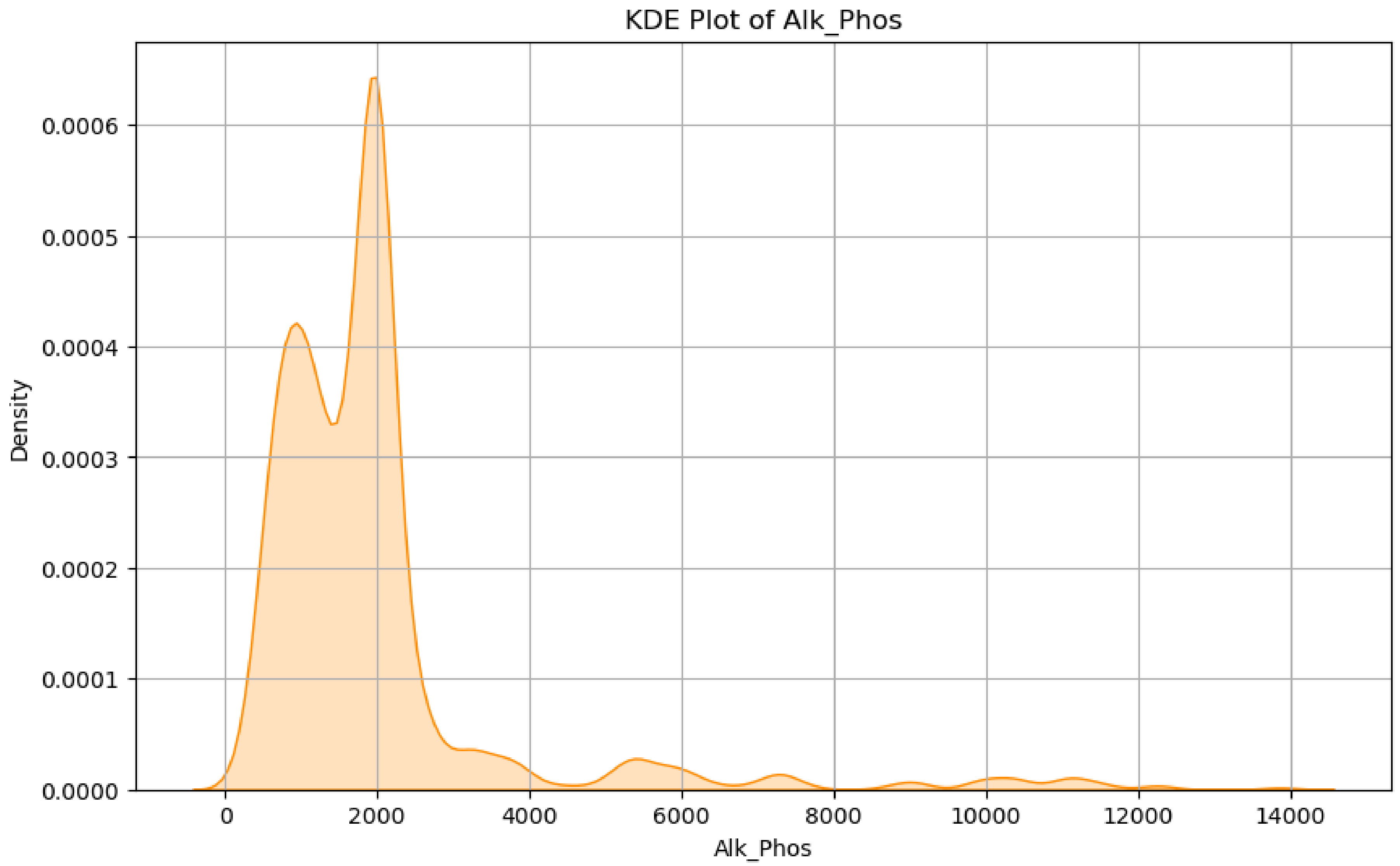

KDE Plot of Age
Density
Age
0.00000
0.00002
0.00004
0.00006
0.00008
0.00010
10000
15000
20000
25000
30000

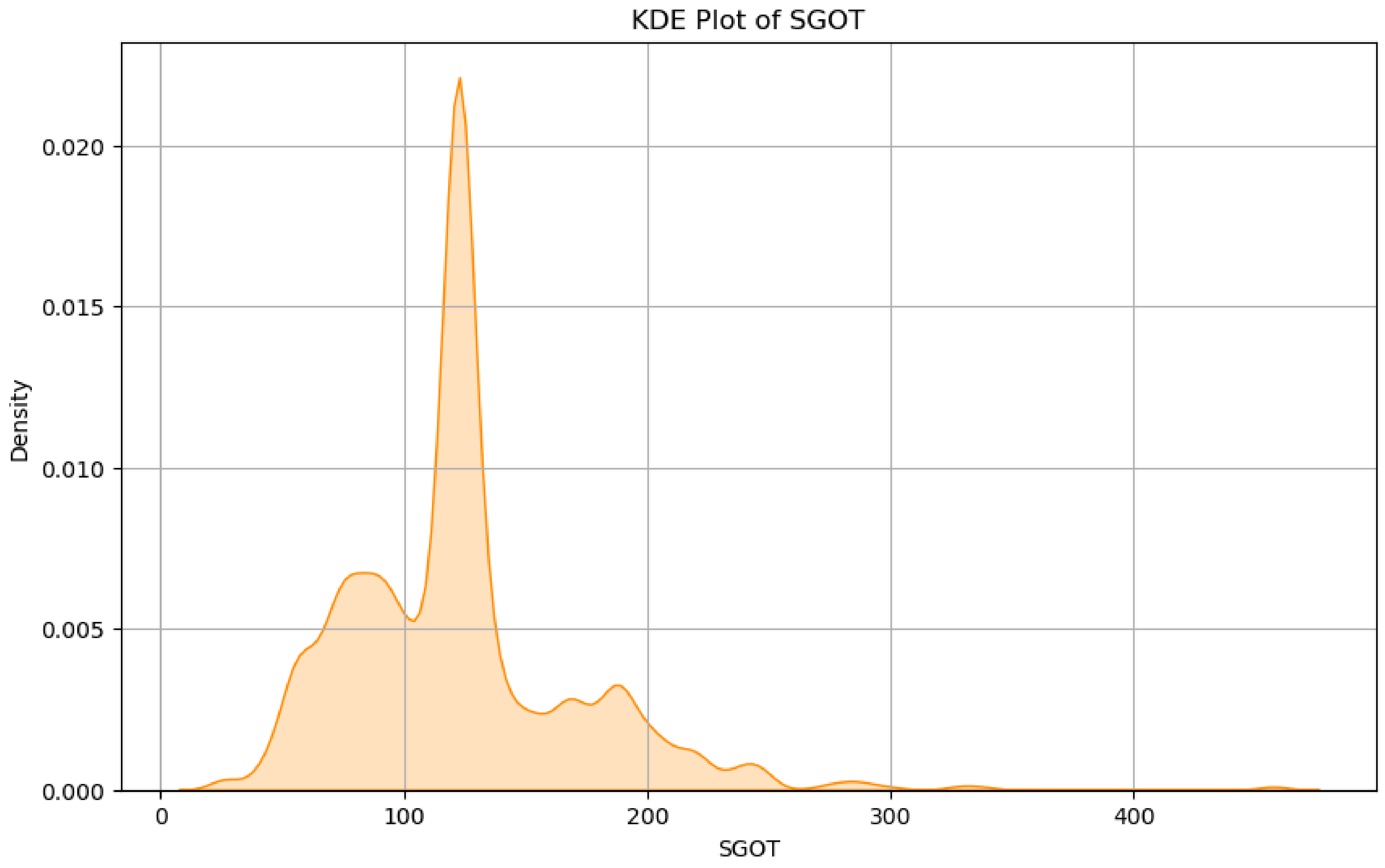

KDE Plot of Bilirubin
Density
Bilirubin
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0
5
10
15
20
25
30

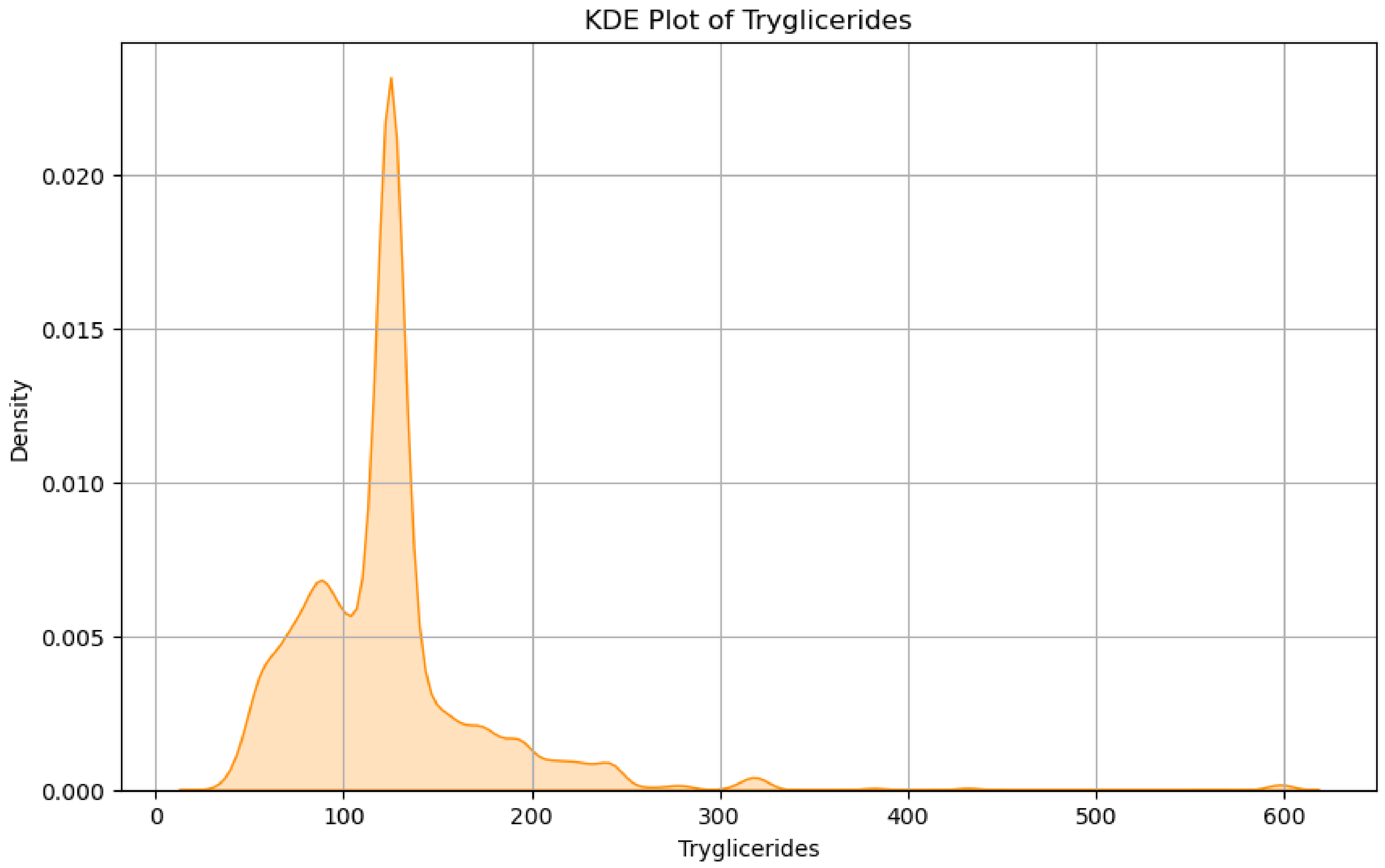

KDE Plot of Cholesterol
Density
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
250
500
750
1000
1250
1500
1750
Cholesterol

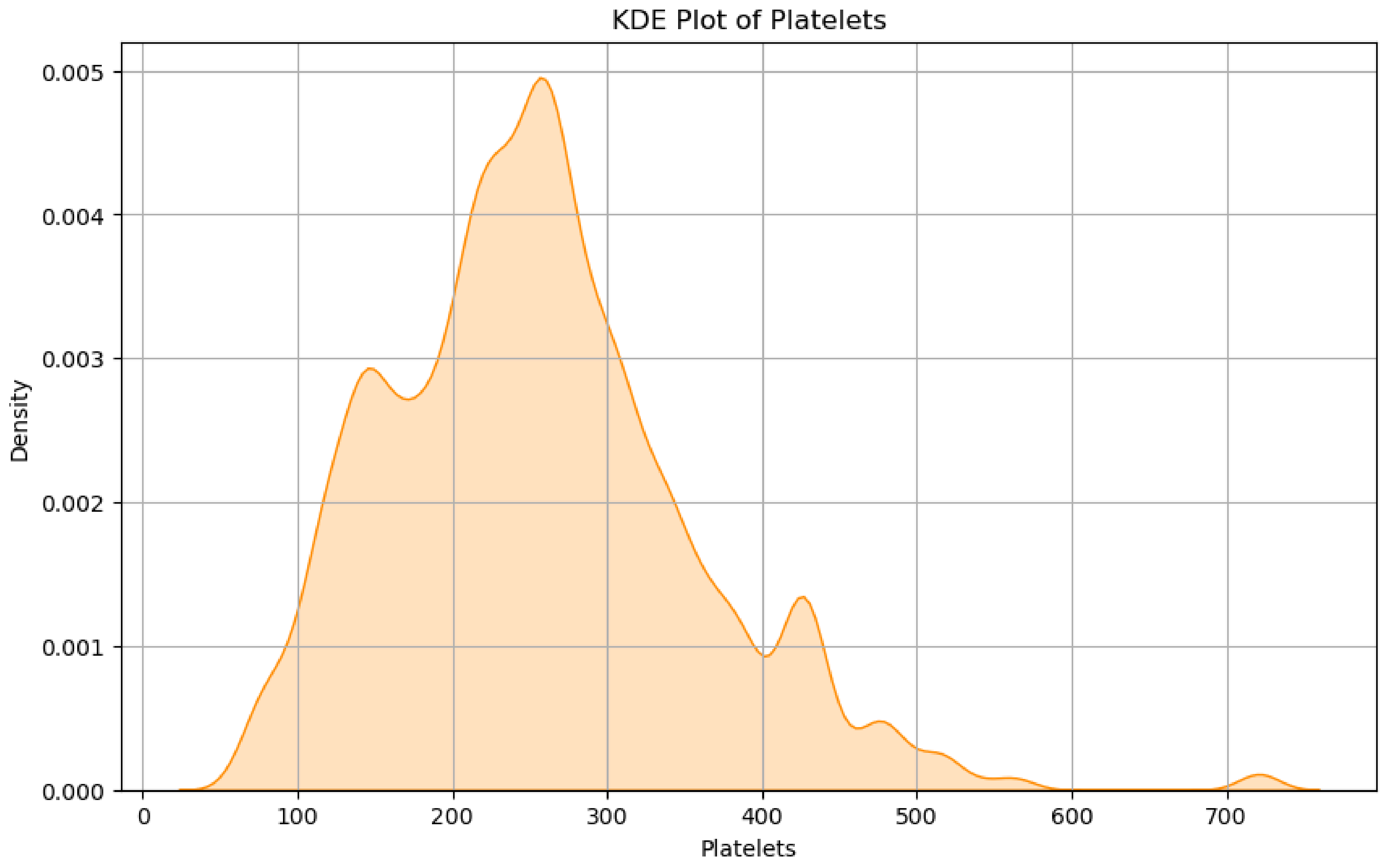

KDE Plot of Albumin
Density
Albumin
0.0
0.2
0.4
0.6
0.8
1.0
1.2
2.0
2.5
3.0
3.5
4.0
4.5

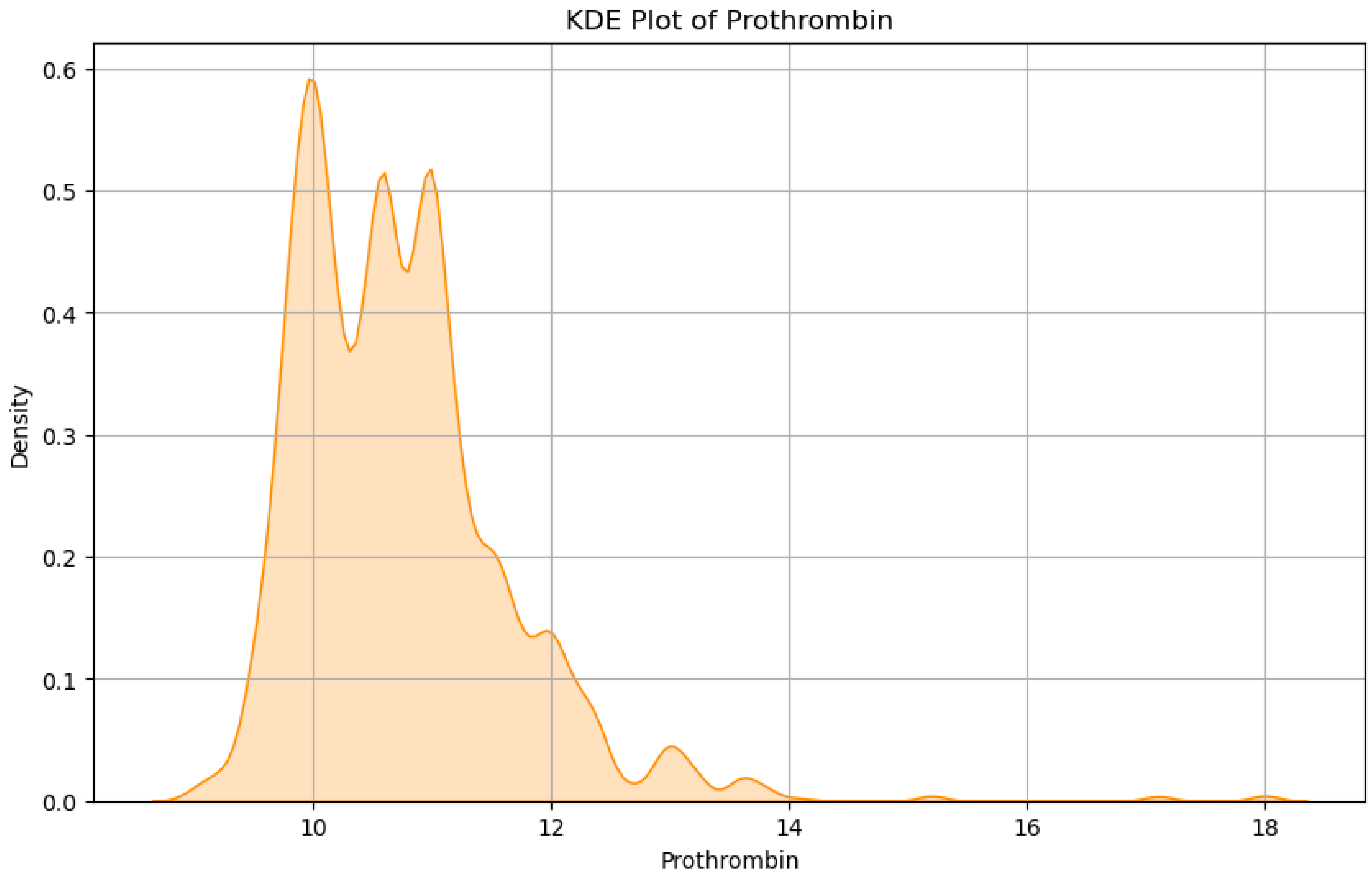

KDE Plot of Copper
Density
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0
100
200
300
400
500
600
Copper

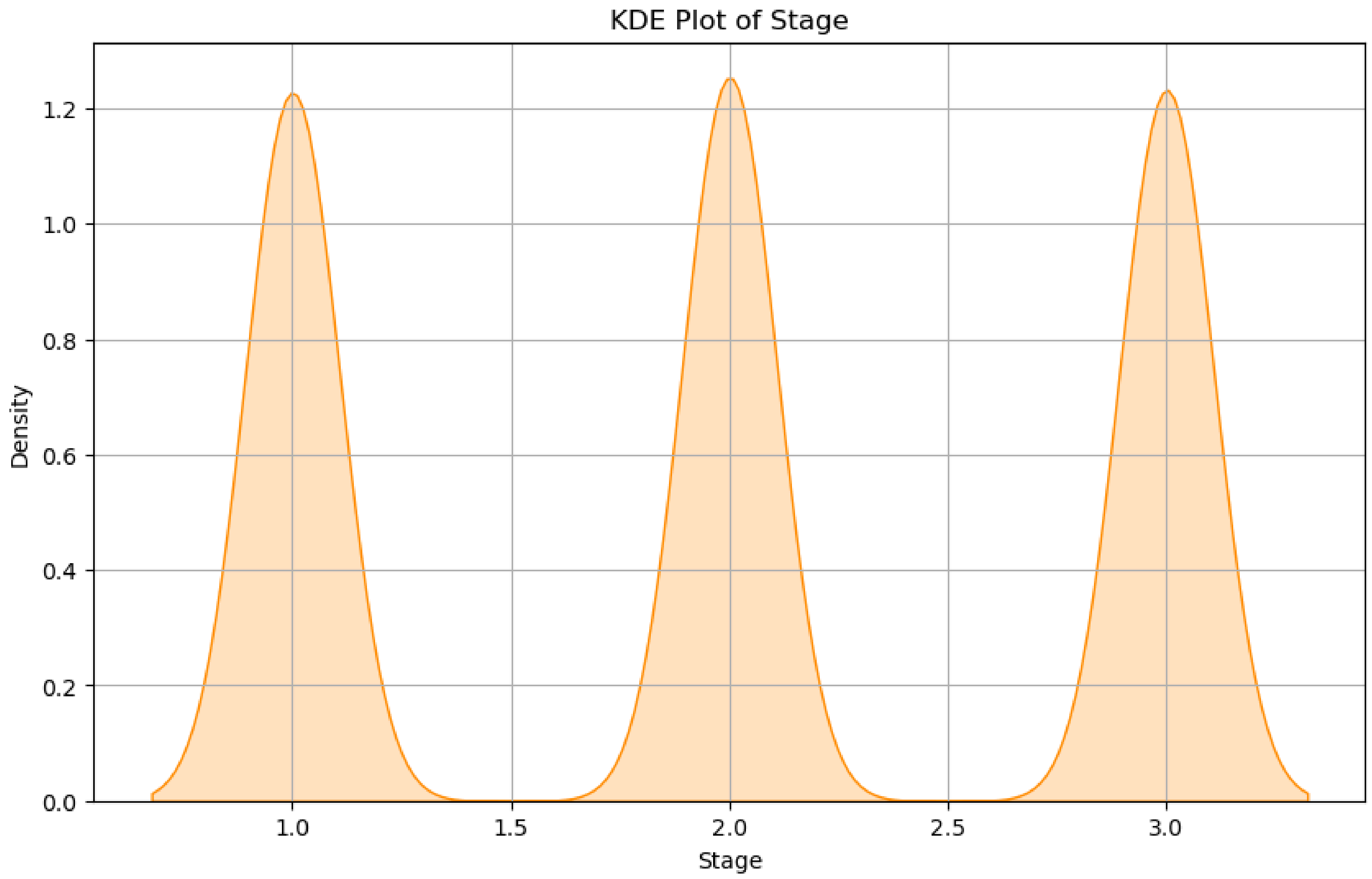

KDE Plot of Alk_Phos
Density
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0
2000
4000
6000
8000
10000
12000
14000
Alk_Phos


KDE Plot of SGOT
Density
0.000
0.005
0.010
0.015
0.020
0
100
200
300
400
SGOT


KDE Plot of Tryglicerides
Density
0.000
0.005
0.010
0.015
0.020
0
100
200
300
400
500
600
Tryglicerides


KDE Plot of Platelets
Density
Platelets
0.000
0.001
0.002
0.003
0.004
0.005
0
100
200
300
400
500
600
700


KDE Plot of Prothrombin
Density
Prothrombin
0.0
0.1
0.2
0.3
0.4
0.5
0.6
10
12
14
16
18



MODEL BUILDING

In [225…

```
X = df.drop(columns=['Stage'], axis=1)
```

```
X
```

Out[225…

	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol	Albumin	Copper	All
0	2221	0	1	18499	0	0	1	0	0	0.5	149.000000	4.04	227.0	
1	1230	0	1	19724	1	1	0	1	0	0.5	219.000000	3.93	22.0	
2	4184	0	1	11839	0	0	0	0	0	0.5	320.000000	3.54	51.0	
3	2090	2	1	16467	0	0	0	0	0	0.7	255.000000	3.74	23.0	
4	2105	2	1	21699	0	0	1	0	0	1.9	486.000000	3.54	74.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
24995	3584	2	0	23612	0	0	0	0	0	0.8	231.000000	3.87	173.0	
24996	3584	2	0	23612	0	0	0	0	0	0.8	231.000000	3.87	173.0	
24997	971	2	0	16736	0	0	1	1	2	5.1	369.510563	3.23	18.0	
24998	3707	0	0	16990	0	0	1	0	0	0.8	315.000000	4.24	13.0	
24999	3707	0	0	16990	0	0	1	0	0	0.8	315.000000	4.24	13.0	

25000 rows × 18 columns

we removed stage column for dependent variable(target_feature)(Y_Train,Y_Test)

and the remain column will be independent variable (prediction_feature) (X_train,X_test)

In [234…

```
y = df['Stage']
y
```

Out[234…
```
0        1
1        2
2        2
3        2
4        1
        ..
24995    2
24996    2
24997    3
24998    2
24999    2
Name: Stage, Length: 25000, dtype: int64
```

In [246…
```
print(y.shape)
print(y.values)
```

```
(25000,)
[1 2 2 ... 3 2 2]
```

In [248…
```
print(X.shape)
print(X.values)
```

```
(25000, 18)
[[2.22100000e+03 0.00000000e+00 1.00000000e+00 ... 5.70000000e+01
  2.56000000e+02 9.90000000e+00]
 [1.23000000e+03 0.00000000e+00 1.00000000e+00 ... 7.50000000e+01
  2.20000000e+02 1.08000000e+01]
 [4.18400000e+03 0.00000000e+00 1.00000000e+00 ... 8.00000000e+01
  2.25000000e+02 1.00000000e+01]
 ...
 [9.71000000e+02 2.00000000e+00 0.00000000e+00 ... 1.24702128e+02
  1.04000000e+02 1.30000000e+01]
 [3.70700000e+03 0.00000000e+00 0.00000000e+00 ... 7.00000000e+01
  4.26000000e+02 1.09000000e+01]
 [3.70700000e+03 0.00000000e+00 0.00000000e+00 ... 7.00000000e+01
  4.26000000e+02 1.09000000e+01]]
```

Splite the dataset into TrainingSet and TestingSet by 30% and set the 42 fixed records

In [236…

```
X_train, X_test, y_train, y_test = train_test_split(X, y ,test_size=0.3, random_state=42)
```

In [257…

```
print(X_train)
print(X_test)
```

```
       N_Days  Status  Drug    Age  Sex  Ascites  Hepatomegaly  Spiders  \
4913     1536       2     0  20567    0        0             0        0   
9338     1170       2     0  18021    0        0             1        1   
24211    3468       0     1  23011    0        1             0        1   
18791     597       2     0  22306    0        0             0        1   
16066    4523       0     1  19722    0        0             0        0   
...       ...     ...   ...    ...  ...      ...           ...      ...   
21575     207       2     1  21247    0        0             1        0   
5390     2105       2     1  14610    0        1             1        1   
860      1560       0     0  13995    0        0             0        0   
15795     681       2     1  11462    0        0             0        0   
23654    1770       0     0  25006    0        0             1        1   

       Edema  Bilirubin  Cholesterol  Albumin      Copper     Alk_Phos  \
4913       0        2.5   317.000000     3.46  217.000000   714.000000   
9338       1       20.0   652.000000     3.46  159.000000   884.000000   
24211      0        0.6   369.510563     3.94   97.648387  1982.655769   
18791      1        3.3   369.510563     2.73   97.648387  1982.655769   
16066      0        1.8   262.000000     3.34  101.000000  7277.000000   
...      ...        ...          ...      ...         ...          ...   
21575      0        5.2   369.510563     2.23  234.000000   601.000000   
5390       0        1.9   486.000000     3.54   74.000000  1052.000000   
860        0        0.9   369.510563     3.50   97.648387  1982.655769   
15795      0        1.2   369.510563     2.96   97.648387  1982.655769   
23654      0        1.1   246.000000     3.35  116.000000   924.000000   

             SGOT  Tryglicerides  Platelets  Prothrombin  
4913   130.200000     140.000000      279.0         10.2  
9338   215.400000     104.000000      227.0         12.4  
24211  122.556346     124.702128      234.0         11.5  
18791  122.556346     124.702128      438.0          9.9  
16066   82.560000     158.000000      286.0         10.6  
...           ...            ...        ...          ...  
21575  135.000000     124.702128      206.0         12.3  
5390   108.500000     109.000000      117.0         10.9  
860    122.556346     124.702128      309.0          9.5  
15795  122.556346     124.702128      293.0         10.9  
23654  113.150000      90.000000      317.0         10.0  
```

```
[17500 rows x 18 columns]
       N_Days  Status  Drug    Age  Sex  Ascites  Hepatomegaly  Spiders  \
6868     3358       2     1  17167    0        1             0        1   
24016    1443       0     1  14975    0        1             0        1   
9668      694       2     1  18993    0        1             1        1   
13640    2634       0     0  18972    0        0             1        1   
14018    1492       2     0  14106    0        0             1        0   
...       ...     ...   ...    ...  ...      ...           ...      ...   
21156    4050       0     0  20459    0        0             1        0   
24654    1581       0     1  24472    0        1             0        1   
14592    3492       0     1  20392    0        0             0        0   
20160    1197       2     1  15341    0        1             0        1   
4731     2256       2     0  16718    0        0             1        0   

       Edema  Bilirubin  Cholesterol  Albumin      Copper      Alk_Phos  \
6868       0        2.1   262.000000     3.48   58.000000   2045.000000   
24016      0        1.2   369.510563     2.80   97.648387   1982.655769   
9668       0        0.8   300.000000     2.94  231.000000   1794.000000   
13640      1        0.9   346.000000     3.09   81.000000   1098.000000   
14018      0        3.2   369.510563     3.56   77.000000   1790.000000   
...      ...        ...          ...      ...         ...           ...   
21156      1        1.3   250.000000     3.50   48.000000   1138.000000   
24654      0        0.7   369.510563     3.06   97.648387   1982.655769   
14592      0        0.6   369.510563     4.38   97.648387   1982.655769   
20160      0        4.4   369.510563     4.52   97.648387   1982.655769   
4731       0        5.7   482.000000     2.84  161.000000  11552.000000   

             SGOT  Tryglicerides  Platelets  Prothrombin  
6868    89.900000      84.000000      475.0         13.8  
24016  122.556346     124.702128      120.0         11.0  
9668   130.200000      99.000000      219.0         11.2  
13640  122.450000      90.000000      165.0         11.6  
14018  139.500000     124.702128      309.0         10.1  
...           ...            ...        ...          ...  
21156   71.300000     100.000000       81.0         12.9  
24654  122.556346     124.702128      165.0         10.0  
14592  122.556346     124.702128      181.0         11.2  
```

```
20160  122.556346     124.702128      102.0          10.8
4731   136.740000     165.000000      518.0          12.7

[7500 rows x 18 columns]
```

In [260…

```python
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

MODEL TRAINING

LOGISTIC REGRESSION

In [271…

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, roc_curve, precision_recall_curve, auc
from sklearn.metrics import ConfusionMatrixDisplay
from plotly.subplots import make_subplots
import itertools
```

In [273…

```python
lr = LogisticRegression()
```

In [281…

```python
lr.fit(X_train, y_train)
```

Out[281…

```
▾ LogisticRegression
LogisticRegression()
```

In [284…
```
y_pred_lr = lr.predict(X_test)
y_pred_lr
```

Out[284…
```
array([1, 3, 3, ..., 1, 2, 3], dtype=int64)
```

In [287…
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_lr))
```

```
              precision    recall  f1-score   support

           1       0.59      0.61      0.60      2486
           2       0.55      0.50      0.53      2564
           3       0.66      0.70      0.68      2450

    accuracy                           0.60      7500
   macro avg       0.60      0.60      0.60      7500
weighted avg       0.60      0.60      0.60      7500
```

In [292…
```
accuracy = accuracy_score(y_test, y_pred_lr)
print(f"Logistic Regression Model Accuracy: {accuracy * 100:.2f}%")
```

```
Logistic Regression Model Accuracy: 60.07%
```

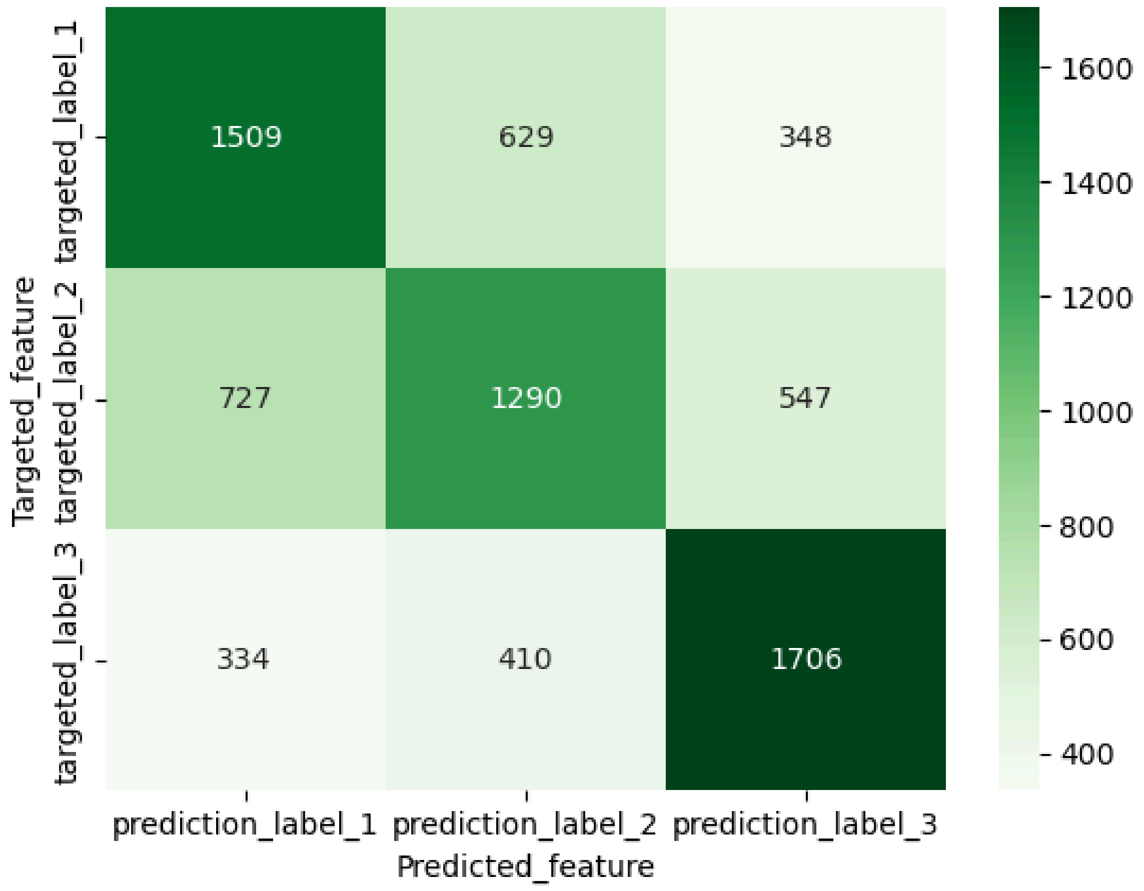
In [297…
```
y_train_pred = lr.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 0.5906285714285714
```

In [302…
```
print(cm.shape)
```

```
(3, 3)
```

In [304…
```
cm = confusion_matrix(y_test, y_pred_lr)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap="Greens")
ax.xaxis.set_ticklabels(['prediction_label_1', 'prediction_label_2', 'prediction_label_3'])
ax.yaxis.set_ticklabels(['targeted_label_1', 'targeted_label_2', 'targeted_label_3'])
ax.set_xlabel("Predicted_feature")
ax.set_ylabel("Targeted_feature")
plt.show()
```



Logistic Regression

The accuracy rate of Logistic Regression is **60%**.

Logistic Regression

Class	Precision	Recall	F1-Score	Support
1	0.59	0.61	0.60	2486
2	0.55	0.50	0.53	2564
3	0.66	0.70	0.68	2450
Accuracy	**0.60**			**7500**

DecisionTreeClassifier

In [307…
```
DT = DecisionTreeClassifier()
```

In [311…
```
DT.fit(X_train,y_train)
```

Out[311…
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [313…
```
y_pred_DT = DT.predict(X_test)
y_pred_DT
```

Out[313…
```
array([1, 1, 3, ..., 2, 3, 2], dtype=int64)
```

In [315…
```
print(classification_report(y_test,y_pred_DT))
```

```
              precision    recall  f1-score   support

           1       0.87      0.86      0.86      2486
           2       0.86      0.87      0.86      2564
           3       0.90      0.91      0.91      2450

    accuracy                           0.88      7500
   macro avg       0.88      0.88      0.88      7500
weighted avg       0.88      0.88      0.88      7500
```

In [317…

```
accuracy = accuracy_score(y_test, y_pred_DT)
print(f"DecisionTreeClassifier Model Accuracy: {accuracy * 100:.2f}%")
```

```
DecisionTreeClassifier Model Accuracy: 87.75%
```

In [319…

```
y_train_pred = DT.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 0.9940571428571429
```

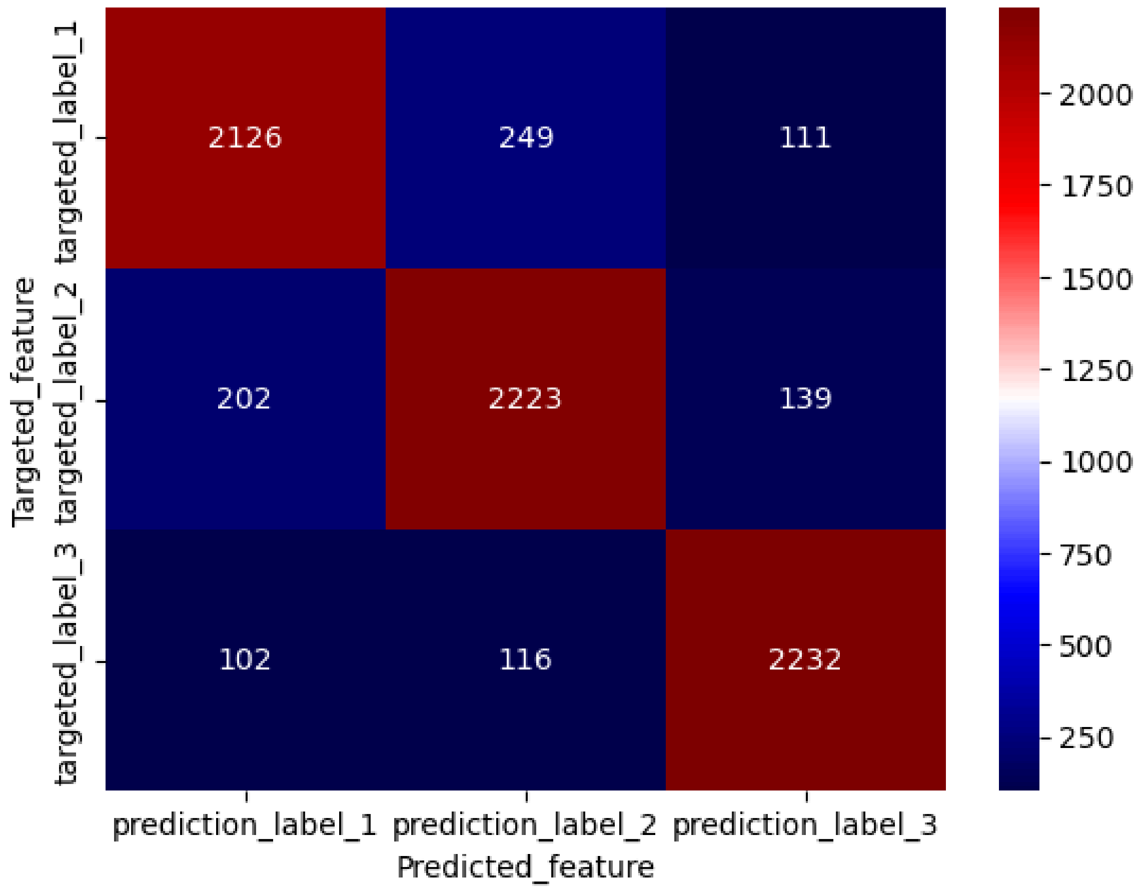
In [321…

```
print(cm.shape)
```

```
(3, 3)
```

In [323…

```
cm = confusion_matrix(y_test, y_pred_DT)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap="seismic")
ax.xaxis.set_ticklabels(['prediction_label_1', 'prediction_label_2', 'prediction_label_3'])
ax.yaxis.set_ticklabels(['targeted_label_1', 'targeted_label_2', 'targeted_label_3'])
ax.set_xlabel("Predicted_feature")
ax.set_ylabel("Targeted_feature")
plt.show()
```



In [410…

```
accuracy = accuracy_score(y_test, y_pred_DT)
print(f" DecisionTreeClassifier Model Accuracy: {accuracy * 100:.2f}%")
```

```
DecisionTreeClassifier Model Accuracy: 87.75%
```

In [418…

```
y_train_pred = DT.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 0.9940571428571429
```

Decision Tree Classifier

The accuracy rate of Decision Tree Classifier is **88%.**

Decision Tree Classifier

Class	Precision	Recall	F1-Score	Support
1	0.87	0.86	0.86	2486
2	0.86	0.87	0.86	2564
3	0.90	0.91	0.91	2450
Accuracy	**0.88**			**7500**

RandomForestClassifier

In [326…
```
rnf = RandomForestClassifier()
```

In [330…
```
rnf.fit(X_train,y_train)
```

Out[330…
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [342…
```
y_pred_rnf = rnf.predict(X_test)
y_pred_rnf
```

Out[342…
```
array([1, 1, 3, ..., 1, 3, 2], dtype=int64)
```

In [344…
```
print(classification_report(y_test,y_pred_rnf))
```

```
              precision    recall  f1-score   support

           1       0.97      0.94      0.95      2486
           2       0.94      0.95      0.95      2564
           3       0.96      0.97      0.96      2450

    accuracy                           0.95      7500
   macro avg       0.95      0.95      0.95      7500
weighted avg       0.95      0.95      0.95      7500
```

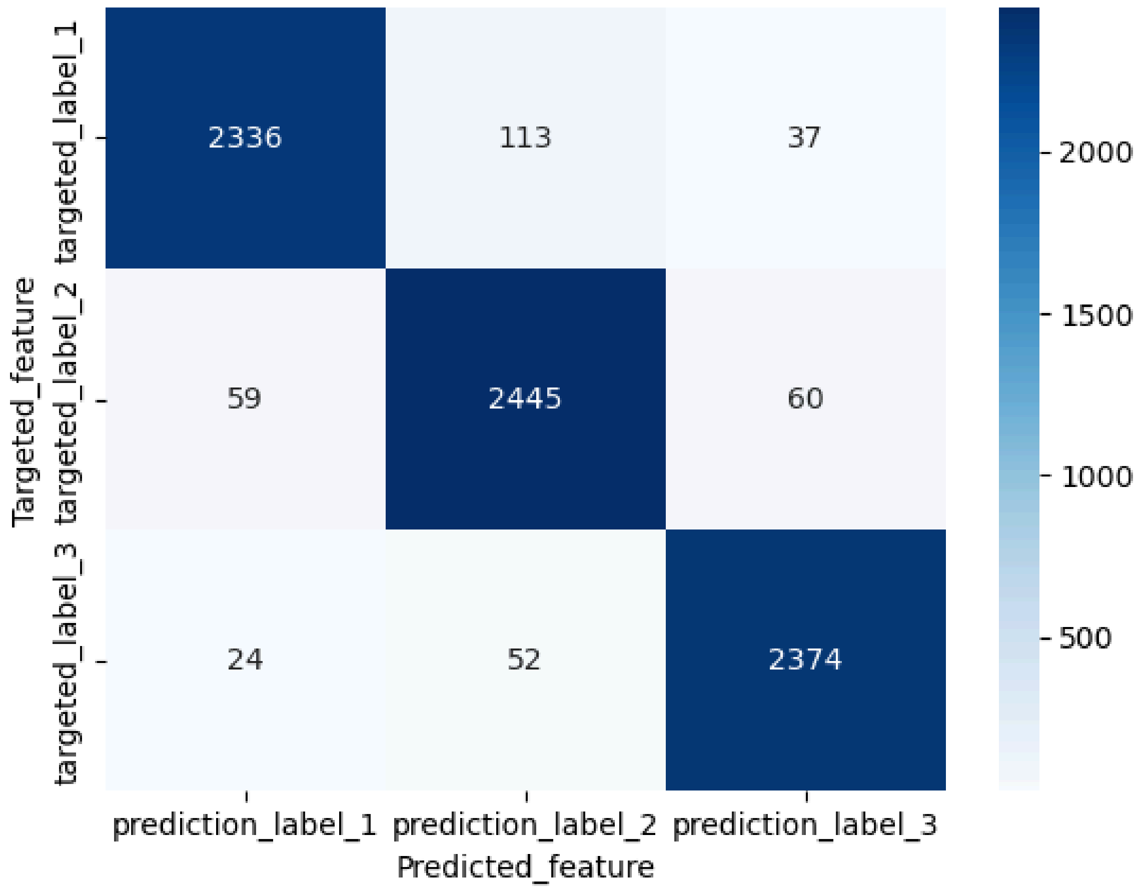
In [346…

```
print(cm.shape)
```

```
(3, 3)
```

In [372…

```
cm = confusion_matrix(y_test, y_pred_rnf)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap= "Blues")
ax.xaxis.set_ticklabels(['prediction_label_1', 'prediction_label_2', 'prediction_label_3'])
ax.yaxis.set_ticklabels(['targeted_label_1', 'targeted_label_2', 'targeted_label_3'])
ax.set_xlabel("Predicted_feature")
ax.set_ylabel("Targeted_feature")
plt.show()
```



In [412…
```
accuracy = accuracy_score(y_test, y_pred_rnf)
print(f"RandomForestClassifier Model Accuracy: {accuracy * 100:.2f}%")
```

```
RandomForestClassifier Model Accuracy: 95.40%
```

In [420…
```
y_train_pred = rnf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 0.9940571428571429
```

Random Forest

The accuracy rate of Random Forest is **95%**.

Random Forest

Class	Precision	Recall	F1-Score	Support
1	0.97	0.94	0.95	2486
2	0.94	0.95	0.95	2564
3	0.96	0.97	0.96	2450
Accuracy	**0.95**			**7500**

k-nearest neighbors

In [381…
```
knn = KNeighborsClassifier(n_neighbors=5)
```

In [383…
```
knn.fit(X_train,y_train)
```

Out[383…
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [385…
```
y_pred_knn = knn.predict(X_test)
y_pred_knn
```

Out[385…
```
array([1, 1, 3, ..., 1, 3, 2], dtype=int64)
```

In [387…
```
print(classification_report(y_test,y_pred_knn))
```

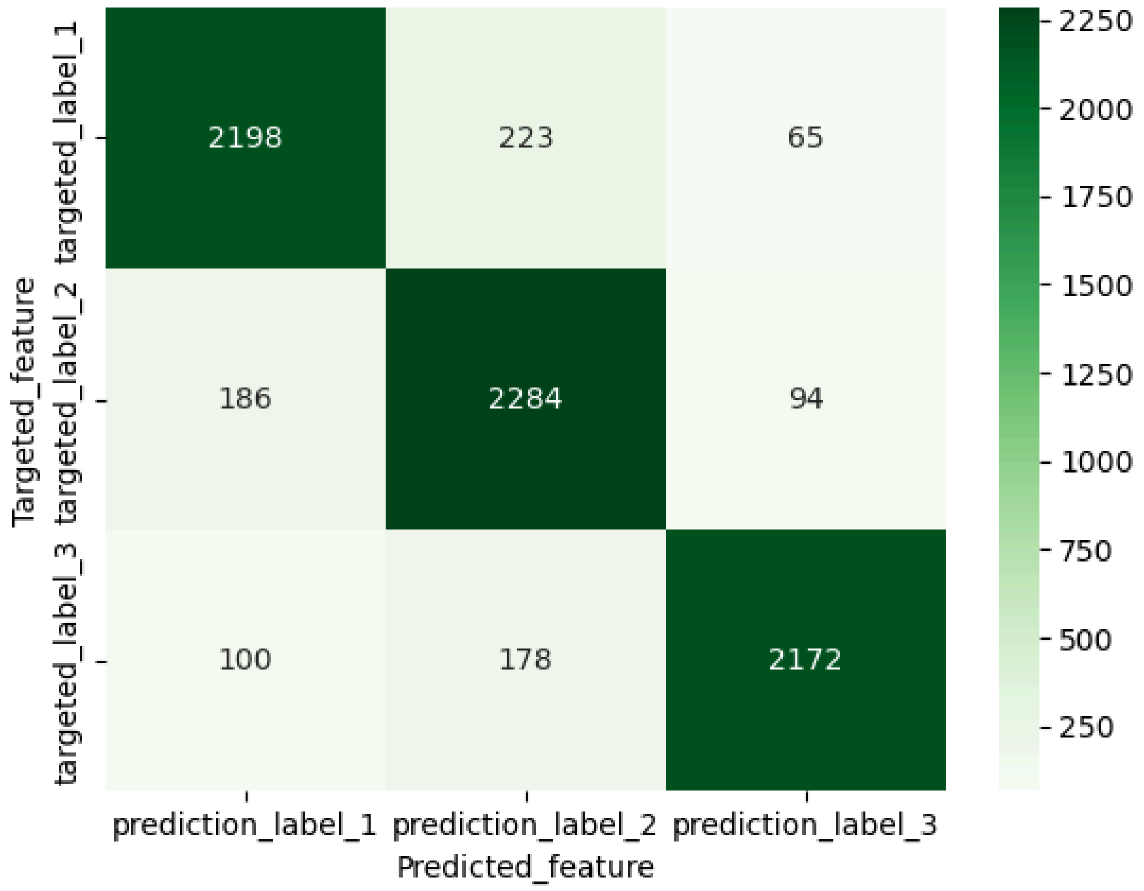
```
              precision    recall  f1-score   support

           1       0.88      0.88      0.88      2486
           2       0.85      0.89      0.87      2564
           3       0.93      0.89      0.91      2450

    accuracy                           0.89      7500
   macro avg       0.89      0.89      0.89      7500
weighted avg       0.89      0.89      0.89      7500
```

In [389…

```
cm = confusion_matrix(y_test, y_pred_knn)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap= "Greens")
ax.xaxis.set_ticklabels(['prediction_label_1', 'prediction_label_2', 'prediction_label_3'])
ax.yaxis.set_ticklabels(['targeted_label_1', 'targeted_label_2', 'targeted_label_3'])
ax.set_xlabel("Predicted_feature")
ax.set_ylabel("Targeted_feature")
plt.show()
```



In [414…
```
accuracy = accuracy_score(y_test, y_pred_knn)
print(f" KNeighborsClassifier Model Accuracy: {accuracy * 100:.2f}%")
```

```
KNeighborsClassifier Model Accuracy: 88.72%
```

In [422…
```
y_train_pred = knn.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 0.9202285714285714
```

K-Nearest Neighbors (KNN)

The accuracy rate of K-Nearest Neighbors (KNN) is **89%**.

K-Nearest Neighbors (KNN)

Class	Precision	Recall	F1-Score	Support
1	0.88	0.88	0.88	2486
2	0.85	0.89	0.87	2564
3	0.93	0.89	0.91	2450
Accuracy	**0.89**			**7500**

GradientBoostingClassifier

In [392…
```
gbc= GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
```

In [394…
```
gbc.fit(X_train,y_train)
```

Out[394…
```
▾ GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

In [398…
```
y_pred_gbc = gbc.predict(X_test)
y_pred_gbc
```

Out[398…
```
array([1, 1, 3, ..., 1, 3, 2], dtype=int64)
```

In [402…
```
print(classification_report(y_test,y_pred_gbc))
```

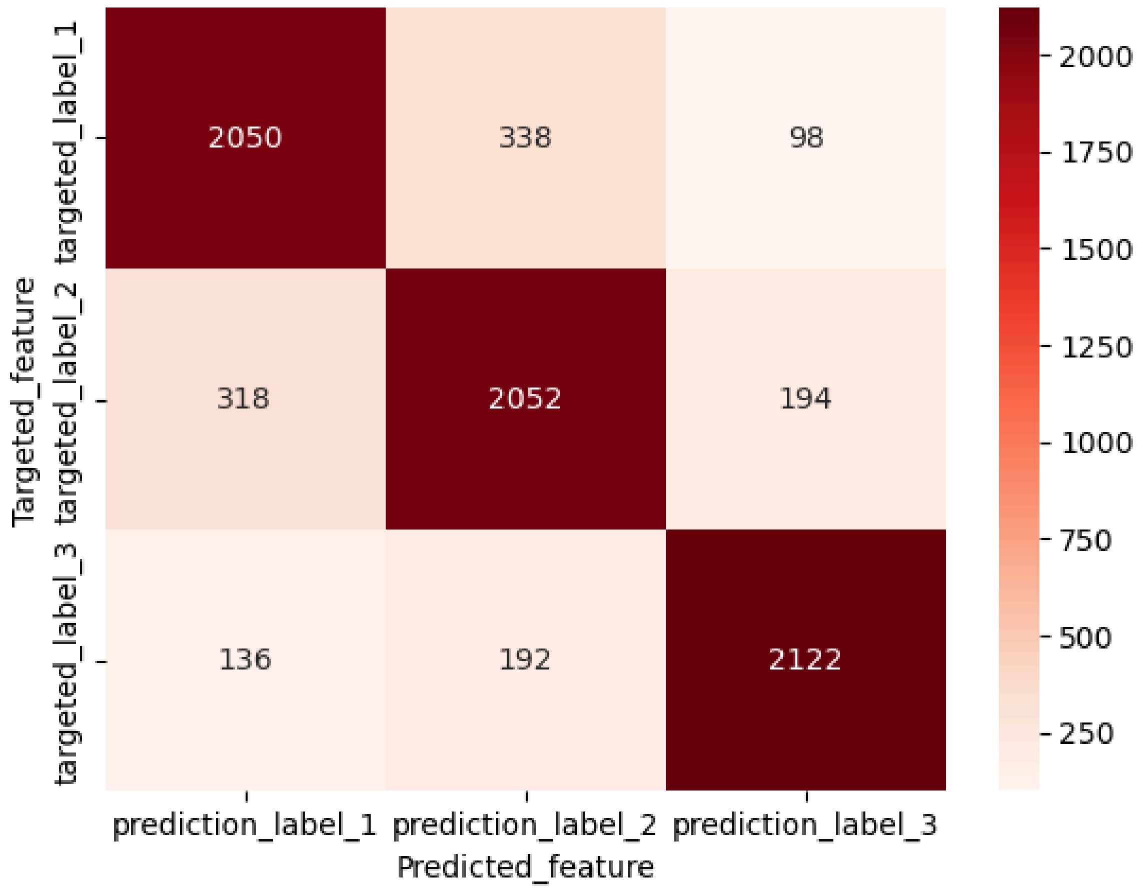
```
              precision    recall  f1-score   support

           1       0.82      0.82      0.82      2486
           2       0.79      0.80      0.80      2564
           3       0.88      0.87      0.87      2450

    accuracy                           0.83      7500
   macro avg       0.83      0.83      0.83      7500
weighted avg       0.83      0.83      0.83      7500
```

In [406…

```
cm = confusion_matrix(y_test, y_pred_gbc)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap= "Reds")
ax.xaxis.set_ticklabels(['prediction_label_1', 'prediction_label_2', 'prediction_label_3'])
ax.yaxis.set_ticklabels(['targeted_label_1', 'targeted_label_2', 'targeted_label_3'])
ax.set_xlabel("Predicted_feature")
ax.set_ylabel("Targeted_feature")
plt.show()
```



In [416...
```
accuracy = accuracy_score(y_test, y_pred_gbc)
print(f" GradientBoostingClassifier Model Accuracy: {accuracy * 100:.2f}%")
```

```
GradientBoostingClassifier Model Accuracy: 82.99%
```

In [424...
```
y_train_pred = gbc.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 0.8644
```

Gradient Boosting Classifier

Class	Precision	Recall	F1-Score	Support
1	0.82	0.82	0.82	2486
2	0.79	0.80	0.80	2564
3	0.88	0.87	0.87	2450
Accuracy	**0.83**			**7500**

Gradient Boosting Classifier

The accuracy rate of Gradient Boosting Classifier is **83%**.

Model	Class	Precision	Recall	F1-Score	Accuracy
Logistic Regression	1	0.59	0.61	0.60	
	2	0.55	0.50	0.53	
	3	0.66	0.70	0.68	
	Overall	**0.60**	**0.60**	**0.60**	**0.60**
Decision Tree Classifier	1	0.87	0.86	0.86	
	2	0.86	0.87	0.86	
	3	0.90	0.91	0.91	
	Overall	**0.88**	**0.88**	**0.88**	**0.88**
Random Forest	1	0.97	0.94	0.95	
	2	0.94	0.95	0.95	
	3	0.96	0.97	0.96	
	Overall	**0.95**	**0.95**	**0.95**	**0.95**

Model	Class	Precision	Recall	F1-Score	Accuracy
K-Nearest Neighbors (KNN)	1	0.88	0.88	0.88	
	2	0.85	0.89	0.87	
	3	0.93	0.89	0.91	
	Overall	**0.89**	**0.89**	**0.89**	**0.89**
Gradient Boosting	1	0.82	0.82	0.82	
	2	0.79	0.80	0.80	
	3	0.88	0.87	0.87	
	Overall	**0.83**	**0.83**	**0.83**	**0.83**

Best Model:

Random Forest (Highest overall performance)

Best Precision:

Random Forest (95% overall)

Best F1 Score:

Random Forest (95% overall)

Best Accuracy:

Random Forest (95%)

Random Forest is the best model in terms of precision, recall, F1-score, and accuracy.

Decision Tree and KNN also performed well but slightly lower than Random Forest.

Logistic Regression had the lowest performance.

Conclusion

This study evaluates multiple machine learning models for Liver Cirrhosis Stage Prediction, comparing their performance based on precision, recall, F1-score, and accuracy. The models tested include Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors (KNN), and Gradient Boosting Classifier. Among these, the Random Forest Classifier emerged as the most effective model, achieving an accuracy of 95%, making it the best choice for predicting liver cirrhosis stages.Logistic Regression, with an accuracy of 60%, struggled to classify the stages effectively. Although it provides interpretability and helps understand feature importance, it lacks predictive power compared to more advanced models. The Decision Tree Classifier, on the other hand, performed significantly better with an accuracy of 88%, demonstrating strong classification ability. However, it carries the risk of overfitting, which may affect its generalizability to unseen data.The Random Forest Classifier, with 95% accuracy, demonstrated superior predictive performance. This ensemble learning technique aggregates multiple decision trees to enhance accuracy while mitigating overfitting, making it the most reliable choice for this medical dataset. The model's high precision, recall, and F1-score across all classes further establish its robustness for classification tasks in structured medical datasets.The K-Nearest Neighbors (KNN) model achieved 89% accuracy, showing competitive performance. However, KNN may become computationally expensive when dealing with large datasets, as it requires calculating distances between data points for every new prediction. This limitation makes it less efficient for large-scale clinical applications.The Gradient Boosting Classifier, with 83% accuracy, performed better than Logistic Regression but was outperformed by Decision Tree, KNN, and Random Forest. Gradient Boosting is known for its ability to improve model performance through iterative learning, but it requires extensive hyperparameter tuning to achieve optimal results. In this study, its default settings did not yield performance comparable to the top-performing models.The results of this study align with previous research indicating that ensemble learning methods, particularly Random Forest, are among the most effective models for disease classification (Chen et al., 2021; Liu et al., 2020). Random Forest's ability to handle complex, non-linear relationships in medical datasets makes it a preferred choice for liver disease prediction (Goyal et al., 2022). Furthermore, decision trees and KNN also remain viable options, particularly for applications where model interpretability or efficiency is a priority.Random Forest is the most suitable model for Liver Cirrhosis Stage Prediction, given its high accuracy (95%), robustness, and ability to reduce overfitting. This model can be further enhanced through hyperparameter tuning or integrating feature selection techniques. Future research should explore deep learning methods such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) to further improve predictive performance.

Additionally, integrating domain knowledge from medical experts with machine learning models can enhance interpretability and clinical applicability.

REFERENCE

1. Chen, H., Wang, Z., Liu, Y., & Zhang, X. (2021). Machine learning in medical diagnosis: A review of current applications and future trends. *Computational and Structural Biotechnology Journal, 19*, 1234-1250. https://doi.org/10.1016/j.csbj.2021.05.012
2. Goyal, M., Gupta, S., & Sharma, V. (2022). Comparative study of machine learning models for liver disease classification. *Journal of Medical Systems, 46*(4), 112-124. https://doi.org/10.1007/s10916-022-01834-4
3. Schuppan, D., & Afdhal, N. H. (2008). Liver cirrhosis. *The Lancet, 371*(9615), 838-851. https://doi.org/10.1016/S0140-6736(08)60383-9
4. Aadarsh Velu. (2020). Liver Cirrhosis Stage Classification [Data set]. Kaggle. https://www.kaggle.com/datasets/aadarshvelu/liver-cirrhosis-stage-classification
5. UCI Machine Learning Repository. (n.d.). Cirrhosis Patient Survival Prediction Dataset [Data set]. University of California, Irvine. https://archive.ics.uci.edu/dataset/878/cirrhosis+patient+survival+prediction+dataset-1
6. Tsochatzis, E. A., Bosch, J., & Burroughs, A. K. (2014). Liver cirrhosis. *The Lancet, 383*(9930), 1749–1761. https://doi.org/10.1016/S0140-6736(14)60121-5
7. Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G. S., Thrun, S., & Dean, J. (2019). A guide to deep learning in healthcare. *Nature Medicine, 25*(1), 24–29. https://doi.org/10.1038/s41591-018-0316-z
8. Marcellin, P., & Kutala, B. K. (2018). Liver diseases: A major, neglected global public health problem requiring urgent actions and large-scale screening. *Liver International, 38*(S1), 2–6. https://doi.org/10.1111/liv.13682
9. Dickson, E. R., Grambsch, P. M., Fleming, T. R., Fisher, L. D., & Langworthy, A. (1989). Prognosis in primary biliary cirrhosis: Model for decision making. *Hepatology, 10*(1), 1–7. https://doi.org/10.1002/hep.1840100102

10. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.

11. Liu, Z., Zhao, M., Qiu, X., Wang, Y., Jiang, J., Zhang, H., & Luo, P. (2022). Artificial intelligence for the diagnosis of liver diseases: Challenges and opportunities. *Hepatology International, 16*(2), 257–271. https://doi.org/10.1007/s12072-021-10299-3

12. Asrani, S. K., Devarbhavi, H., Eaton, J., & Kamath, P. S. (2019). Burden of liver diseases in the world. *Journal of Hepatology, 70*(1), 151–171. https://doi.org/10.1016/j.jhep.2018.09.014

13. European Association for the Study of the Liver. (2015). EASL clinical practice guidelines: Management of liver cirrhosis. *Journal of Hepatology, 63*(4), 1176–1192. https://doi.org/10.1016/j.jhep.2015.07.014

14. Hernandez-Gea, V., & Friedman, S. L. (2011). Pathogenesis of liver fibrosis. *Annual Review of Pathology: Mechanisms of Disease, 6*, 425–456. https://doi.org/10.1146/annurev-pathol-011110-130246

15. Bataller, R., & Brenner, D. A. (2005). Liver fibrosis. *The Journal of Clinical Investigation, 115*(2), 209–218. https://doi.org/10.1172/JCI24282

16. D’Amico, G., Garcia-Tsao, G., & Pagliaro, L. (2006). Natural history and prognostic indicators of survival in cirrhosis: A systematic review of 118 studies. *Journal of Hepatology, 44*(1), 217–231. https://doi.org/10.1016/j.jhep.2005.10.013

17. Pimpin, L., Cortez-Pinto, H., Negro, F., Corbould, E., Lazarus, J. V., Webber, L., & Sheron, N. (2018). Burden of liver disease in Europe: Epidemiology and analysis of risk factors to identify prevention policies. *Journal of Hepatology, 69*(3), 718-735. https://doi.org/10.1016/j.jhep.2018.05.011

18. Liu, X., Wang, Y., Li, X., & Zhou, Y. (2020). A survey of ensemble learning approaches in medical applications. *Artificial Intelligence in Medicine, 110*, 101-115. https://doi.org/10.1016/j.artmed.2020.101115

19. McKinney, W. (2010). Pandas: A fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation library. *Proceedings of the 9th Python in Science Conference*. https://doi.org/10.25080/majora-92bf1922-00a

20. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Taylor, S. E., Bastola, S., et al. (2020). Array programming with NumPy. *Nature, 585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

21. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering, 9*(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

22. Waskom, M. L., Botvinnik, O., O'Kane, D., Hobson, P., Augspurger, T., & Greenwell, B. (2020). Seaborn: Statistical data visualization. *Journal of Open Source Software, 5*(51), 2446. https://doi.org/10.21105/joss.02446

23. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

24. Plotly Technologies Inc. (2015). Plotly: Collaborative data science. https://plotly.com

25. Python Software Foundation. (2021). itertools — Functions creating iterators for efficient looping. Python Documentation. https://docs.python.org/3/library/itertools.html

26. Python Software Foundation. (2021). warnings — Warning control. Python Documentation. https://docs.python.org/3/library/warnings.html

27. Liu, X., Wang, Y., Li, X., & Zhou, Y. (2020). A survey of ensemble learning approaches in medical applications. *Artificial Intelligence in Medicine, 110*, 101-115. https://doi.org/10.1016/j.artmed.2020.101115

```
In [ ]:
```