Double-click (or enter) to edit

New Section

loading the **dataset**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Loading the dataset
data = pd.read_csv('/content/HEPAR_simulated_patients (1).csv')

print(data.head())
# Checking data types and missing values
print(data.info())
print(data.isnull().sum())
```

```
 63  edge                 10000 non-null  object
 64  irregular_liver      10000 non-null  object
 65  hbc_anti             10000 non-null  object
 66  hcv_anti             10000 non-null  object
 67  palms                10000 non-null  object
 68  hbeag                10000 non-null  object
 69  carcinoma            10000 non-null  object
dtypes: object(70)
memory usage: 5.3+ MB
None
alcoholism     0
vh_amn         0
hepatotoxic    0
THepatitis     0
hospital       0
              ..
hbc_anti       0
hcv_anti       0
palms          0
hbeag          0
carcinoma      0
Length: 70, dtype: int64
```

PREPROCESSING THE DATA {HANDLING ALL THE MISSING VALUES }

```
import pandas as pd

# Loading the dataset
data = pd.read_csv('/content/HEPAR_simulated_patients (1).csv')

#  Handling Missing Values
if 'ChHepatitis' in data.columns and data['ChHepatitis'].isnull().sum() > 0:
    data['ChHepatitis'].fillna('absent', inplace=True)

# Check for any other columns with missing values
missing_columns = data.columns[data.isnull().any()].tolist()
for col in missing_columns:

    if data[col].dtype == 'object':
        data[col].fillna(data[col].mode()[0], inplace=True)
    else:
        data[col].fillna(data[col].median(), inplace=True)

# Encode Categorical Variables with "absent" and "present"
binary_columns = [
    'Cirrhosis', 'ChHepatitis', 'THepatitis', 'RHepatitis', 'PBC',
    'Hyperbilirubinemia', 'ascites', 'hepatomegaly', 'hepatalgia',
    'spiders', 'itching', 'fatigue', 'encephalopathy', 'alcoholism',
    'obesity', 'diabetes'
]

# Mapping "absent" to 0, "present" to 1, and specific states for "Cirrhosis"
for col in binary_columns:
    if col == 'Cirrhosis':
        data[col] = data[col].map({'absent': 0, 'compensate': 1, 'decompensate': 2})
    else:
        data[col] = data[col].map({'absent': 0, 'present': 1})
```

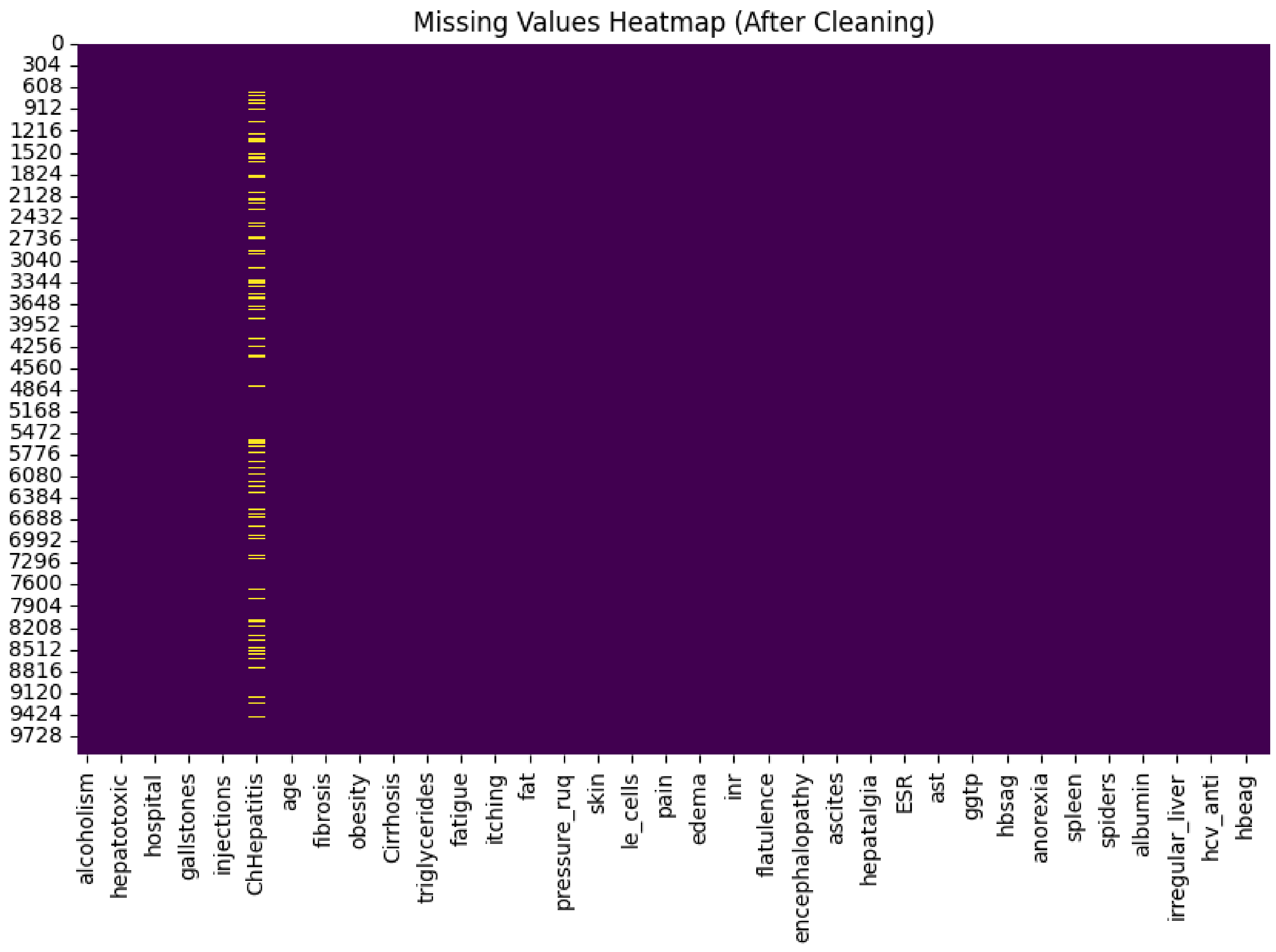
```
# Convert Coded Numerical Values (e.g., "a1_0", "a6_2") by extracting the numeric part
coded_columns = ['bilirubin', 'albumin', 'phosphatase', 'ggtp', 'cholesterol']

for col in coded_columns:
    # Extract the part after "_" and convert it to a float
    data[col] = data[col].str.split('_').str[1].astype(float)

#  Confirm Data Structure and Check for Remaining Missing Values
print("Preview of the cleaned data:")
print(data.head())
# Display the first few rows of the cleaned data

print("\nMissing values in each column after imputation:")
print(data.isnull().sum())
```

```
Preview of the cleaned data:
   alcoholism   vh_amn hepatotoxic  THepatitis hospital surgery gallstones  \
0           0  present      absent           0   absent  absent     absent
1           0   absent      absent           0   absent  absent    present
2           0   absent      absent           0  present  absent     absent
3           0   absent      absent           0  present  absent     absent
4           0  present      absent           0   absent  absent    present

  choledocholithotomy injections transfusion  ...  spiders jaundice albumin  \
0              absent     absent      absent  ...        0  present    50.0
1              absent     absent      absent  ...        0  present     0.0
2              absent    present      absent  ...        0   absent    50.0
3              absent     absent      absent  ...        0  present    50.0
4             present     absent      absent  ...        1   absent    50.0

     edge irregular_liver  hbc_anti  hcv_anti    palms   hbeag  carcinoma
0  absent         present    absent   present   absent  absent     absent
1  absent          absent    absent    absent   absent  absent     absent
2  absent          absent    absent    absent  present  absent     absent
3  absent          absent    absent    absent   absent  absent     absent
4  absent          absent    absent    absent  present  absent     absent

[5 rows x 70 columns]

Missing values in each column after imputation:
alcoholism     0
vh_amn         0
hepatotoxic    0
THepatitis     0
hospital       0
              ..
hbc_anti       0
hcv_anti       0
palms          0
hbeag          0
carcinoma      0
Length: 70, dtype: int64
```

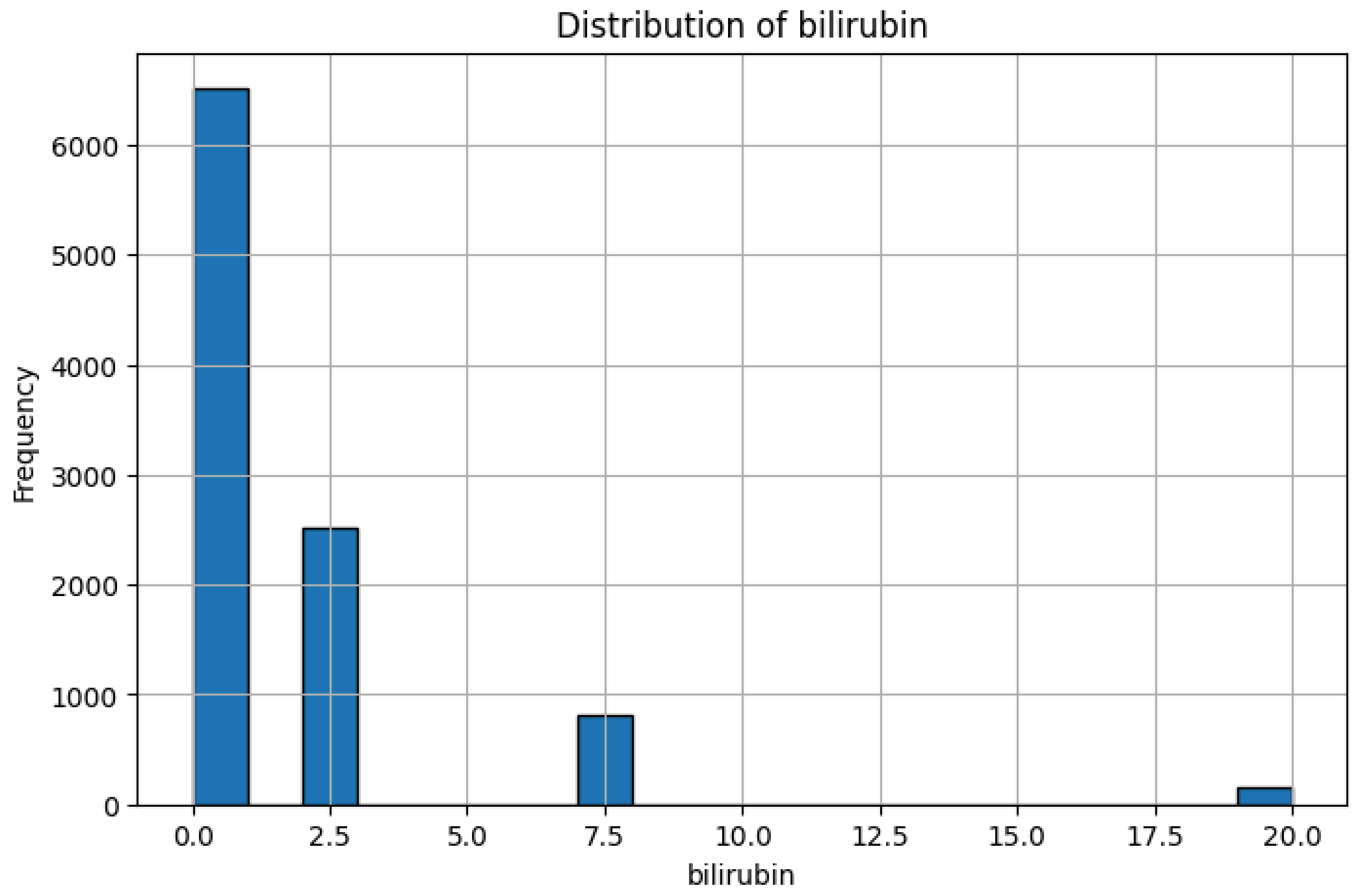
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv('/content/HEPAR_simulated_patients (1).csv')
```

```python
#  Handle Missing Values
if 'ChHepatitis' in data.columns and data['ChHepatitis'].isnull().sum() > 0:
    data['ChHepatitis'].fillna('absent', inplace=True)

missing_columns = data.columns[data.isnull().any()].tolist()
for col in missing_columns:
    if data[col].dtype == 'object':
        data[col].fillna(data[col].mode()[0], inplace=True)
    else:
        data[col].fillna(data[col].median(), inplace=True)

#  Encode Categorical Variables with "absent" and "present"
binary_columns = [
    'Cirrhosis', 'ChHepatitis', 'THepatitis', 'RHepatitis', 'PBC',
    'Hyperbilirubinemia', 'ascites', 'hepatomegaly', 'hepatalgia',
    'spiders', 'itching', 'fatigue', 'encephalopathy', 'alcoholism',
    'obesity', 'diabetes'
]

for col in binary_columns:
    if col == 'Cirrhosis':
        data[col] = data[col].map({'absent': 0, 'compensate': 1, 'decompensate': 2})
    else:
        data[col] = data[col].map({'absent': 0, 'present': 1})

# Convert Coded Numerical Values (e.g., "a1_0", "a6_2")
coded_columns = ['bilirubin', 'albumin', 'phosphatase', 'ggtp', 'cholesterol']

for col in coded_columns:
    data[col] = data[col].str.split('_').str[1].astype(float)

# Visualizations

# 1. Missing Values Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(data.isnull(), cbar=False, cmap="viridis")
plt.title("Missing Values Heatmap (After Cleaning)")
plt.show()

# 2. Distribution of Numerical Columns
numerical_columns = ['bilirubin', 'albumin', 'phosphatase', 'ggtp', 'cholesterol']

for col in numerical_columns:
    plt.figure(figsize=(8, 5))
    data[col].hist(bins=20, edgecolor='black')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()

# 3. Proportions of Binary Columns
binary_proportions = data[binary_columns].mean()

plt.figure(figsize=(12, 6))
binary_proportions.plot(kind='bar')
plt.title('Proportion of Present (1) for Binary Columns')
plt.ylabel('Proportion')
plt.xticks(rotation=45, ha='right')
plt.show()
```

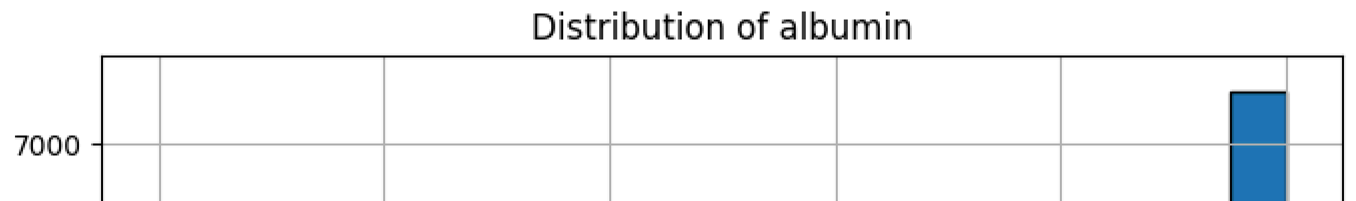
```
# 5. Boxplots for Numerical Columns vs Binary Outcomes
for col in numerical_columns:
    plt.figure(figsize=(8, 5))
    sns.boxplot(x='Cirrhosis', y=col, data=data)
    plt.title(f'{col} by Cirrhosis Status')
    plt.xlabel('Cirrhosis Status')
    plt.ylabel(col)
    plt.show()

# 6. Pairplot for Key Features
key_features = ['bilirubin', 'albumin', 'Cirrhosis', 'ascites', 'hepatomegaly']
sns.pairplot(data[key_features], hue='Cirrhosis', palette='coolwarm')
plt.show()
```



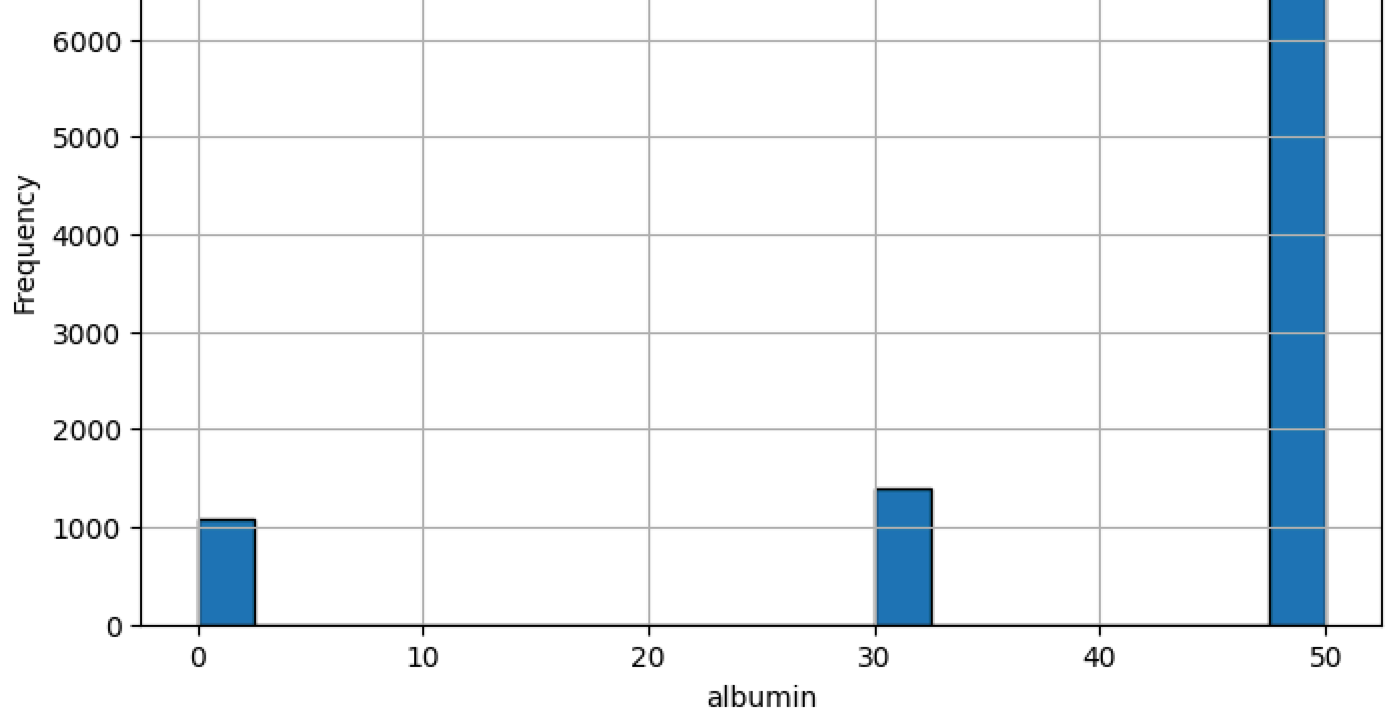
Missing Values Heatmap (After Cleaning)
0
304
608
912
1216
1520
1824
2128
2432
2736
3040
3344
3648
3952
4256
4560
4864
5168
5472
5776
6080
6384
6688
6992
7296
7600
7904
8208
8512
8816
9120
9424
9728
alcoholism
hepatotoxic
hospital
gallstones
injections
ChHepatitis
age
fibrosis
obesity
Cirrhosis
triglycerides
fatigue
itching
fat
pressure_ruq
skin
le_cells
pain
edema
inr
flatulence
encephalopathy
ascites
hepatalgia
ESR
ast
ggtp
hbsag
anorexia
spleen
spiders
albumin
irregular_liver
hcv_anti
hbeag



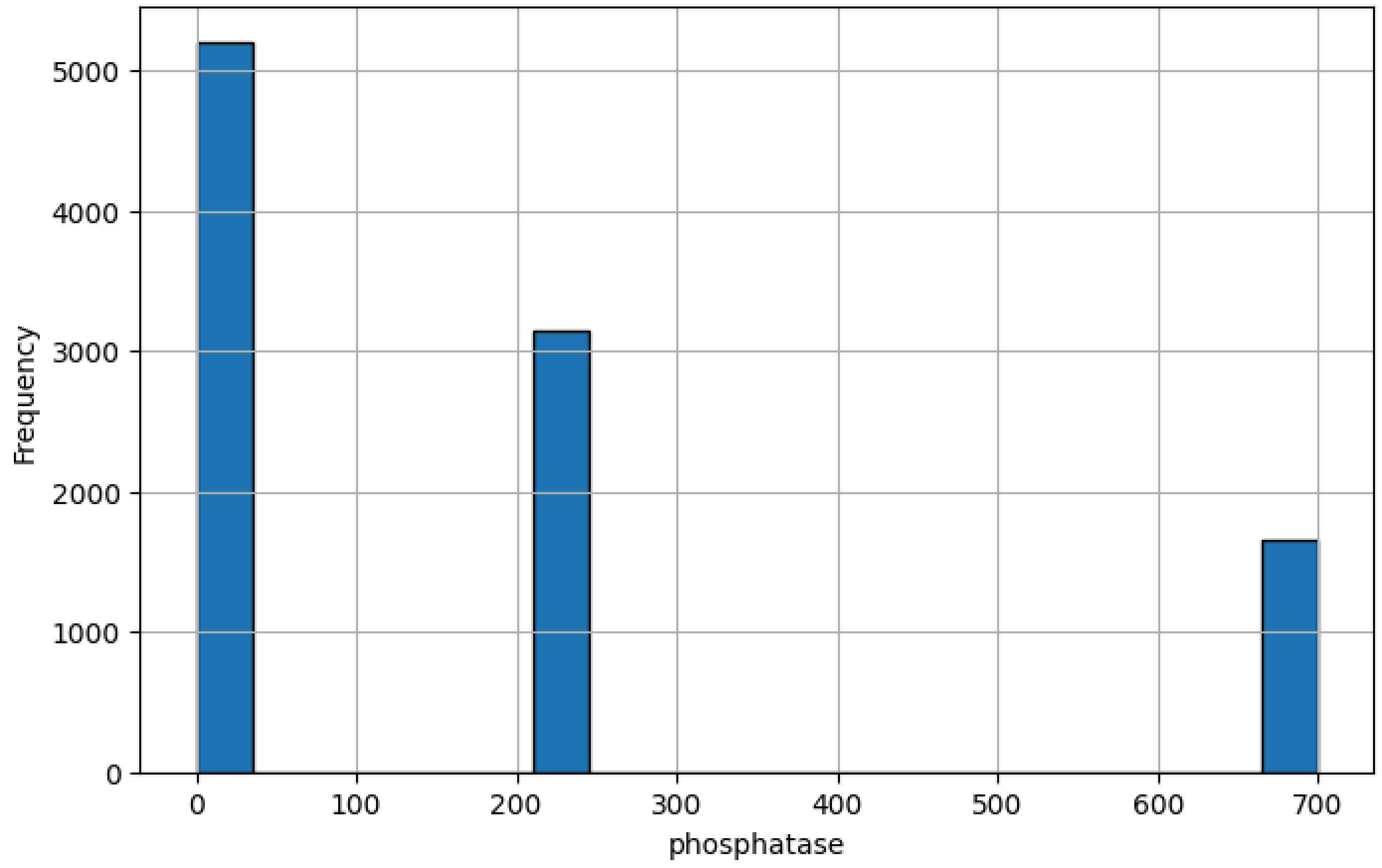
Distribution of bilirubin
Frequency
6000
5000
4000
3000
2000
1000
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
bilirubin



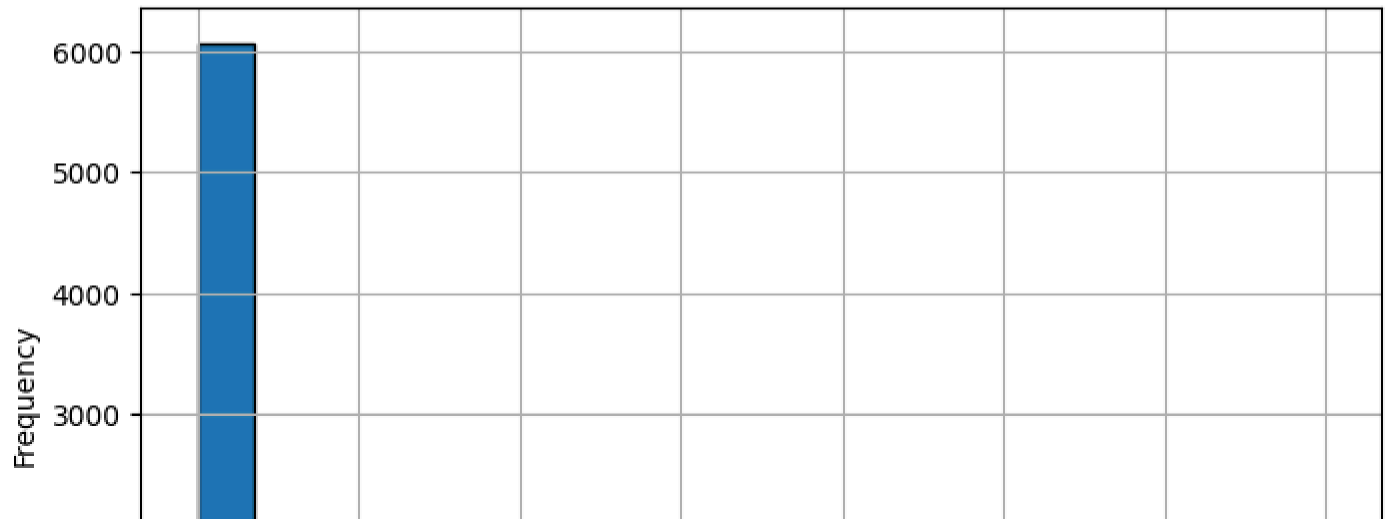
Distribution of albumin
7000

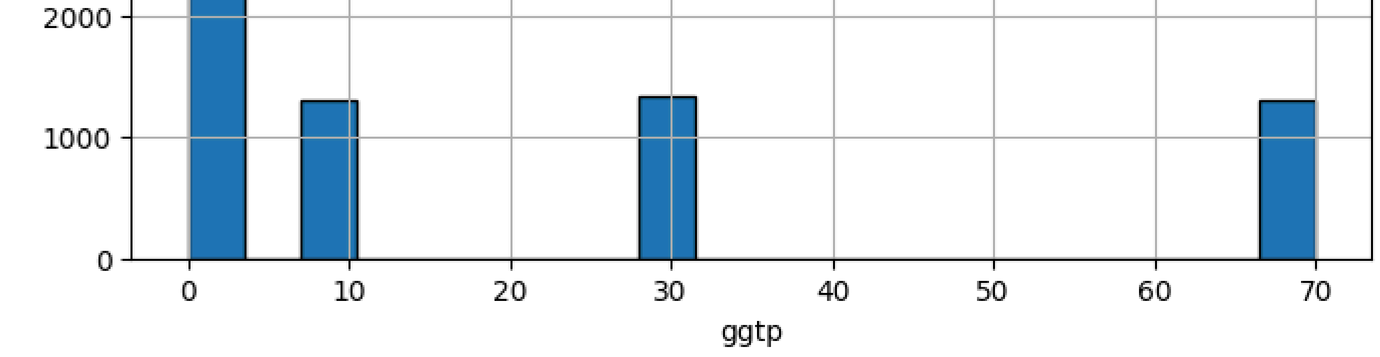


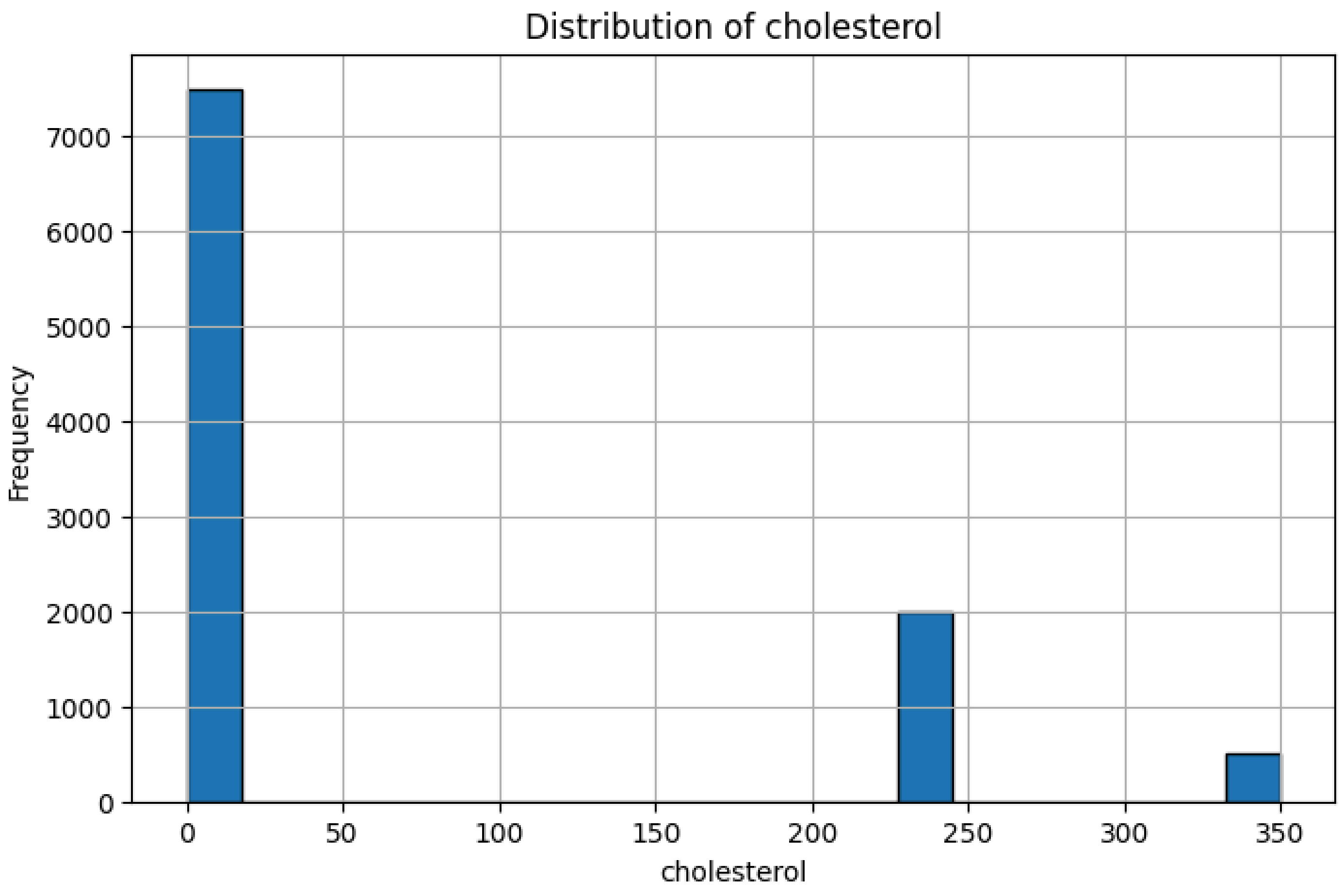
Distribution of phosphatase

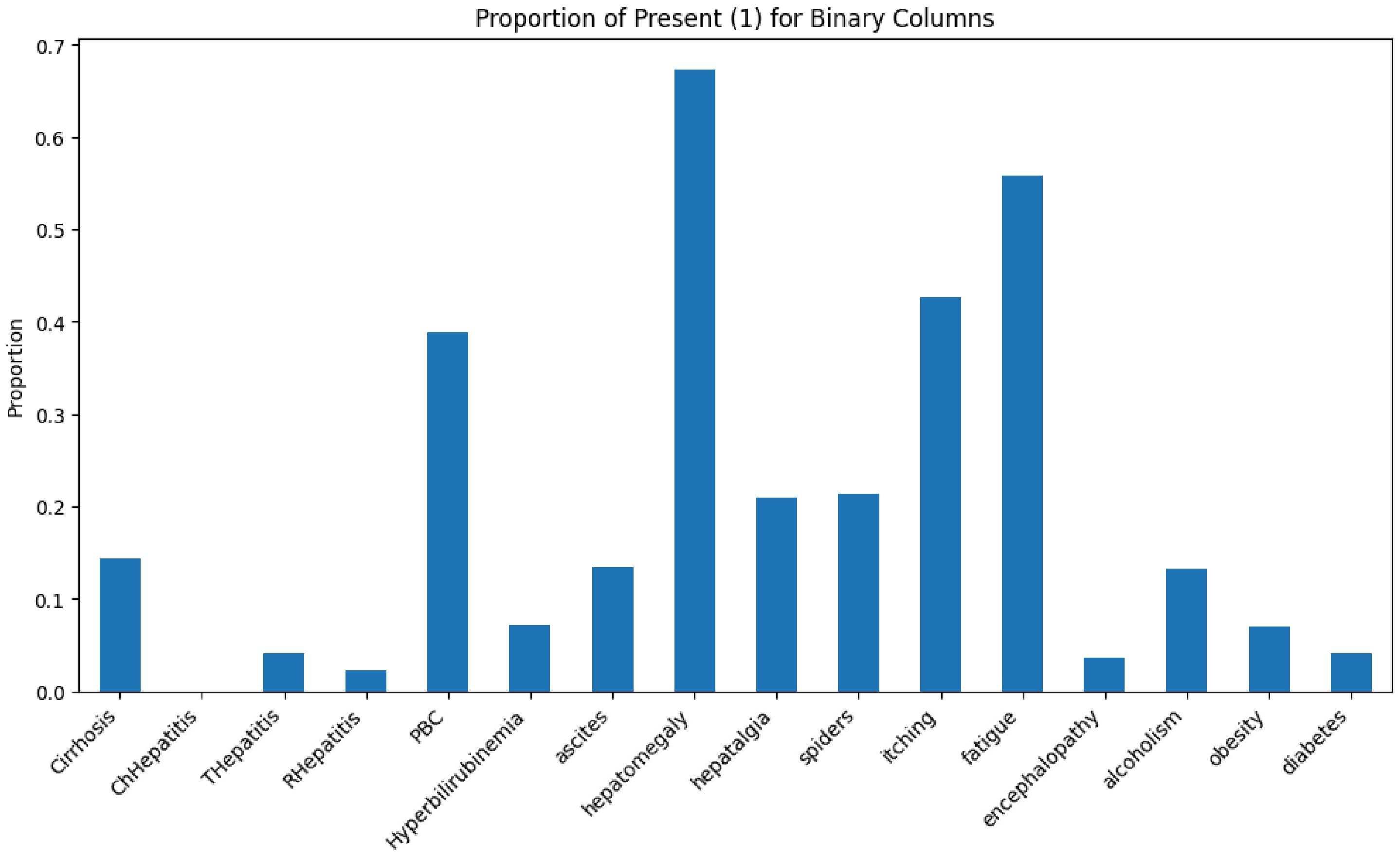


Distribution of ggtp

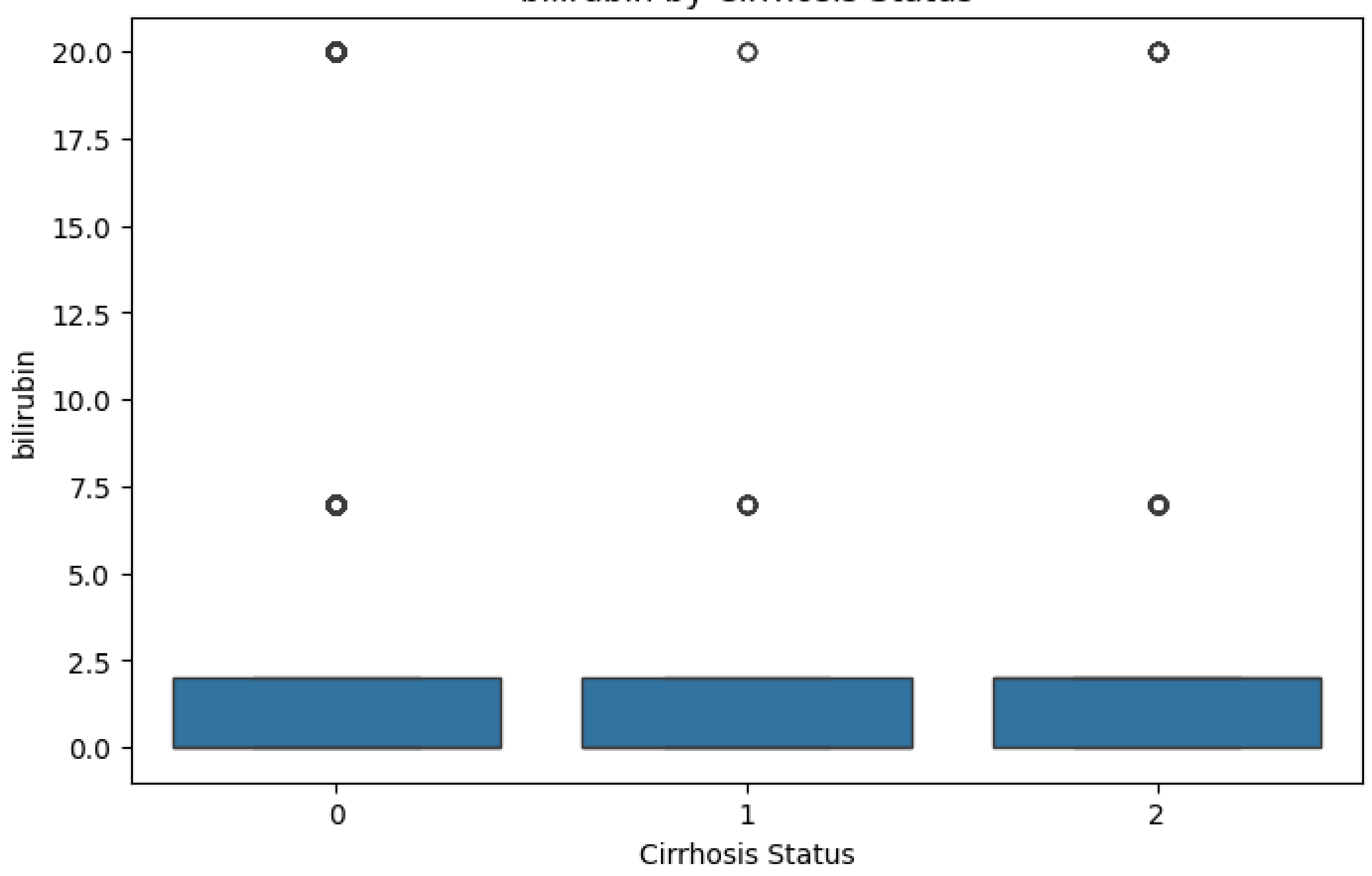


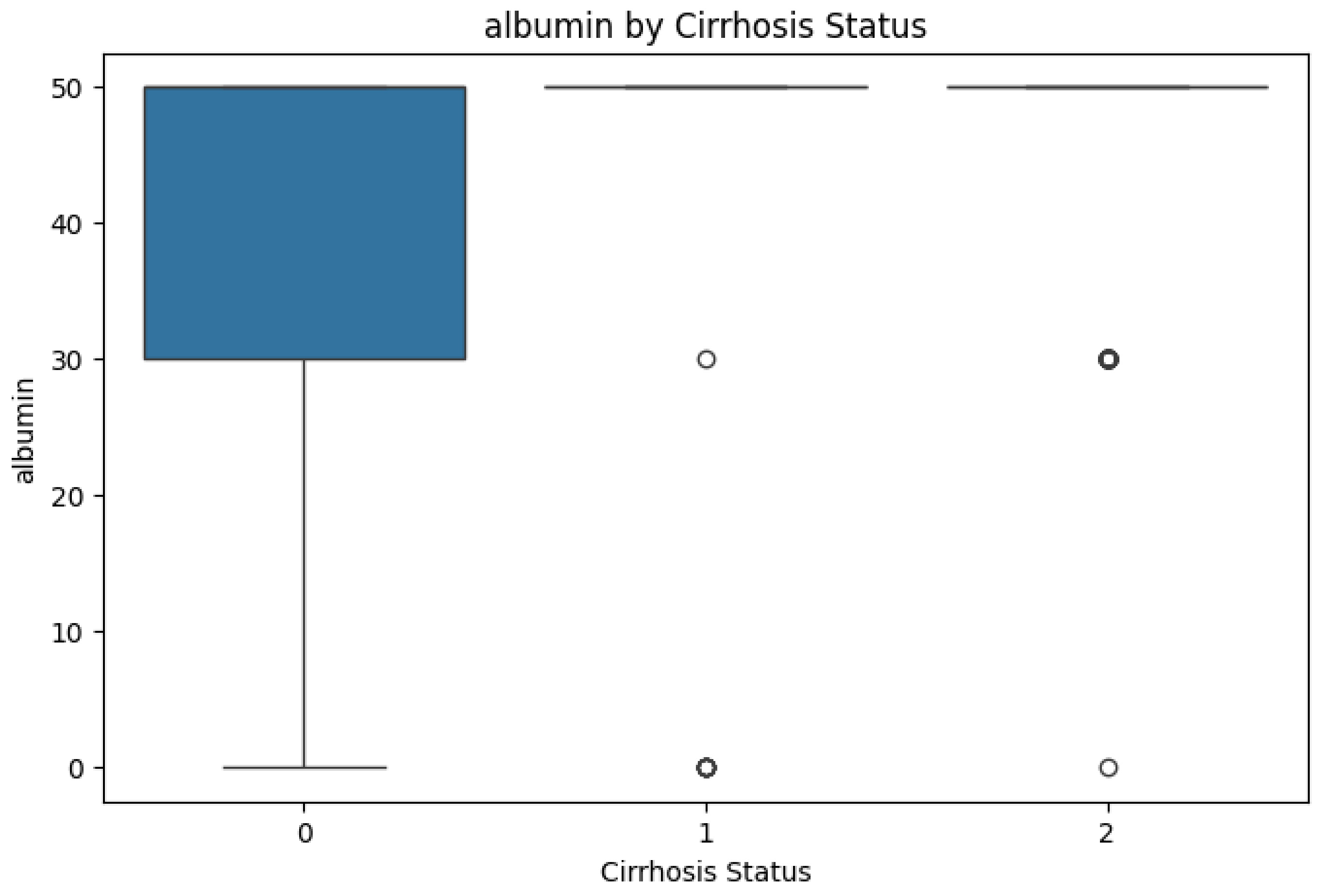


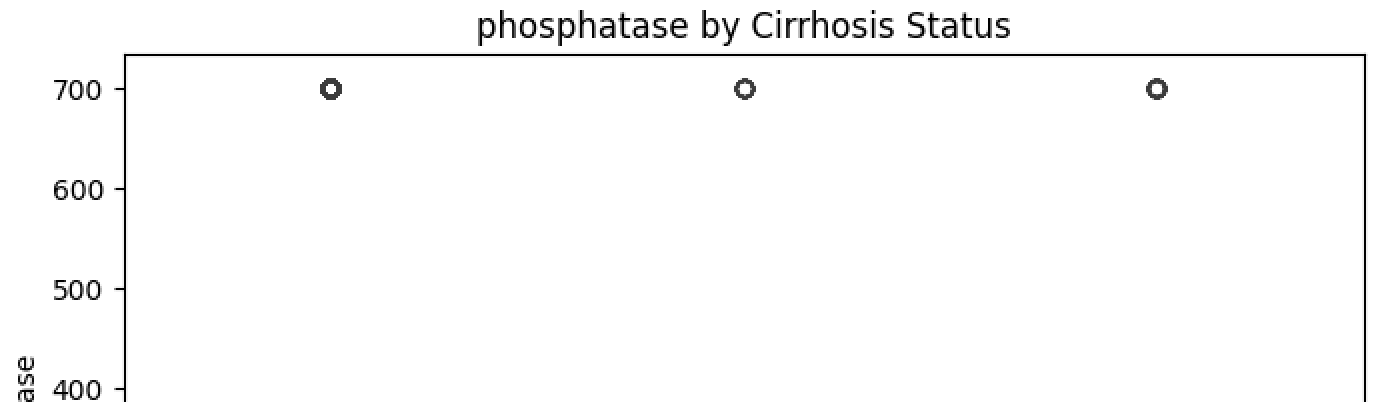


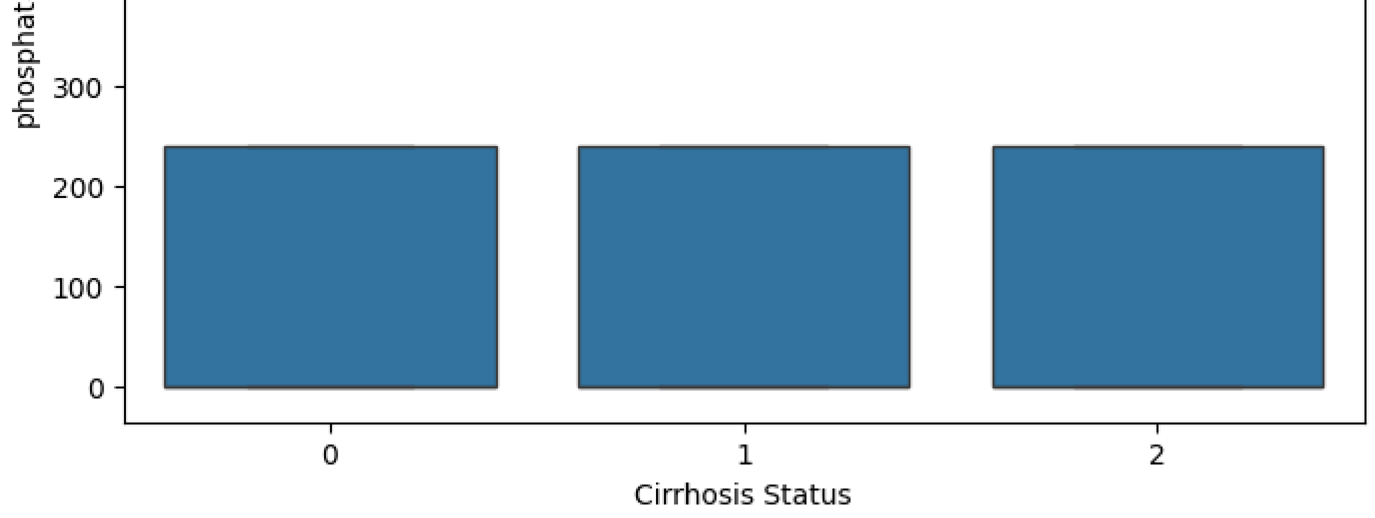


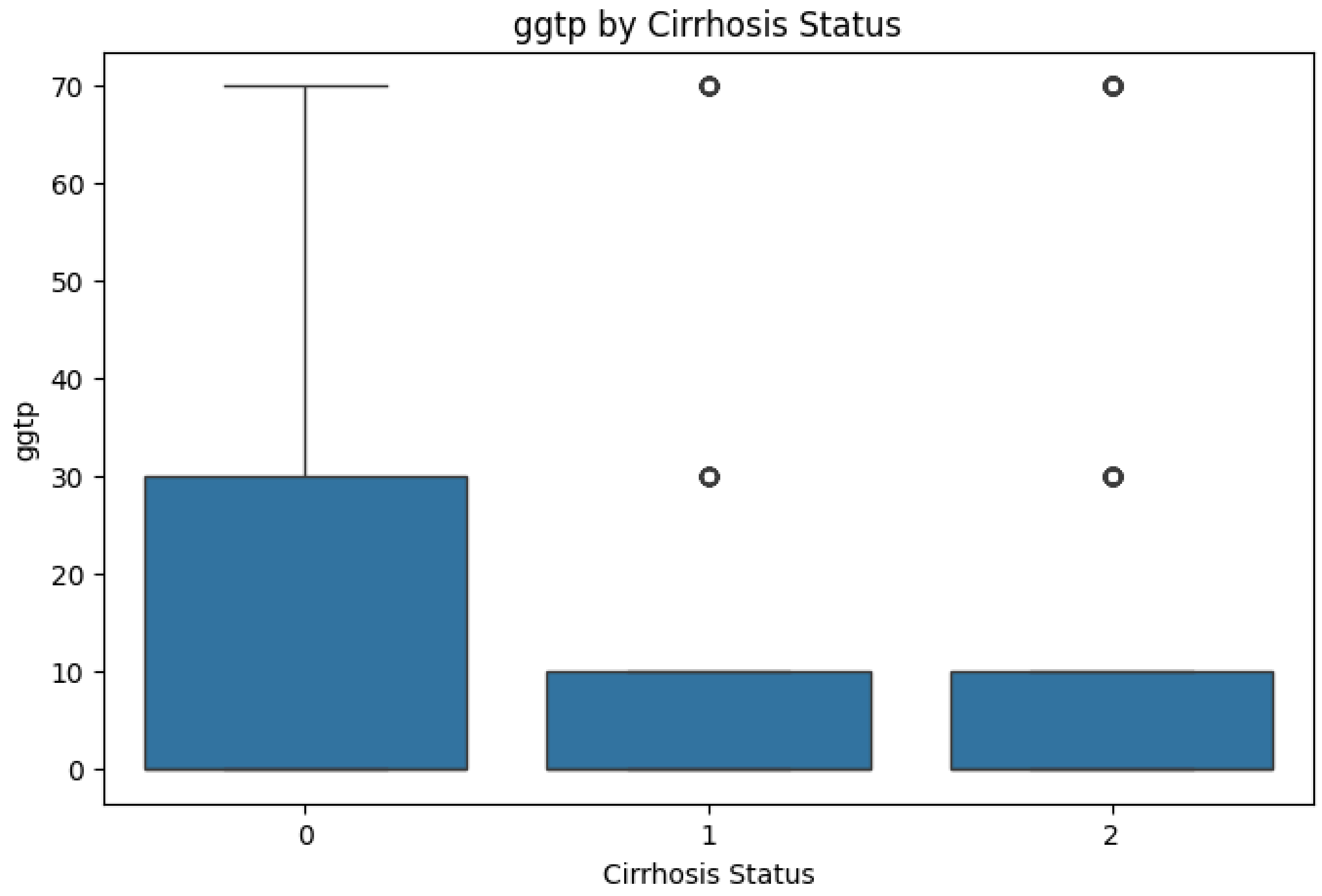
bilirubin by Cirrhosis Status

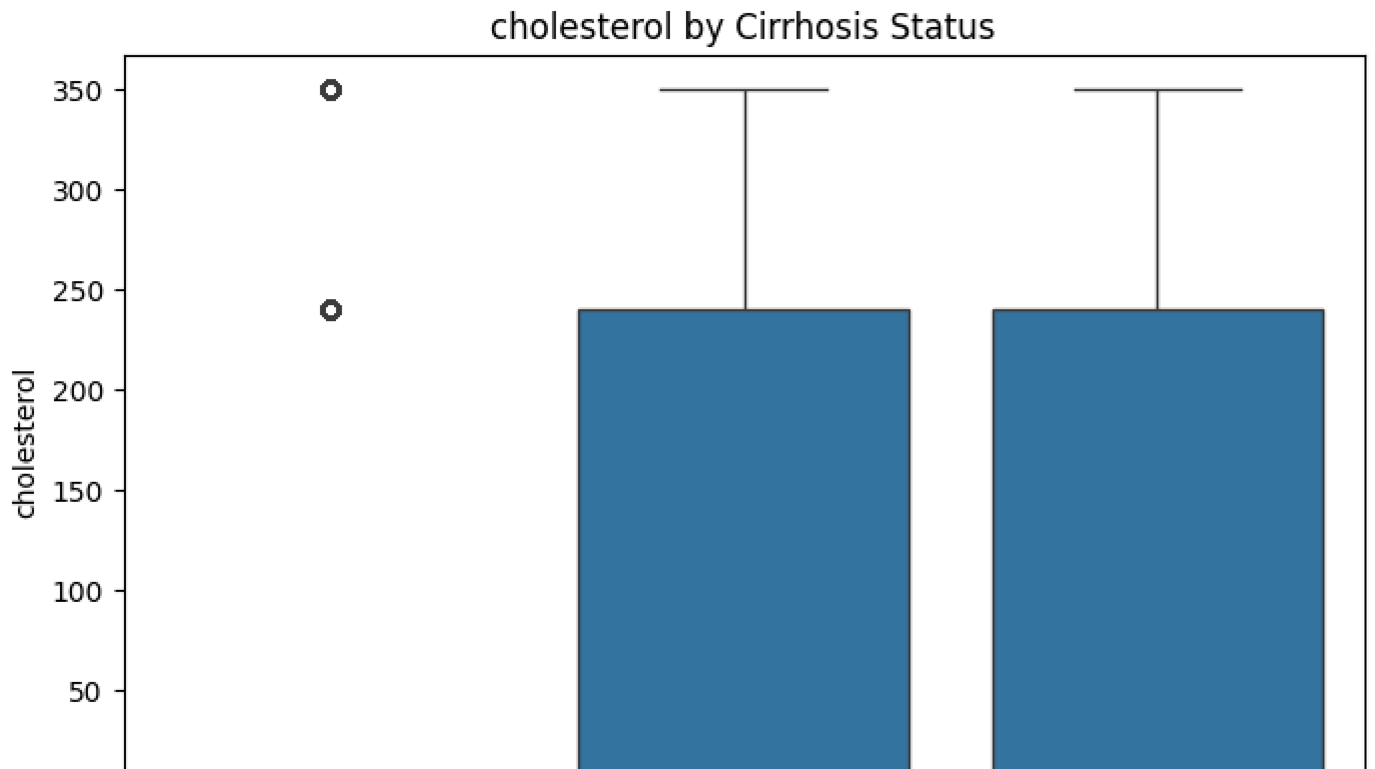

bilirubin
Cirrhosis Status
0
1
2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

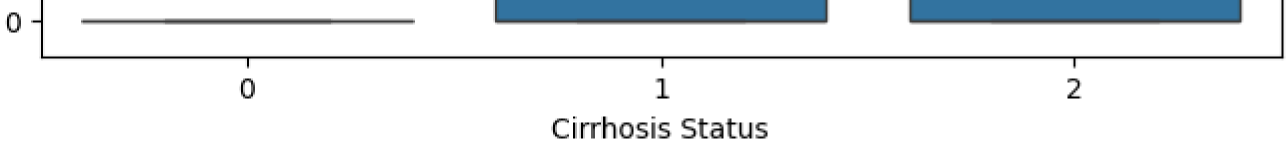

albumin by Cirrhosis Status
albumin
Cirrhosis Status
0
1
2
0
10
20
30
40
50

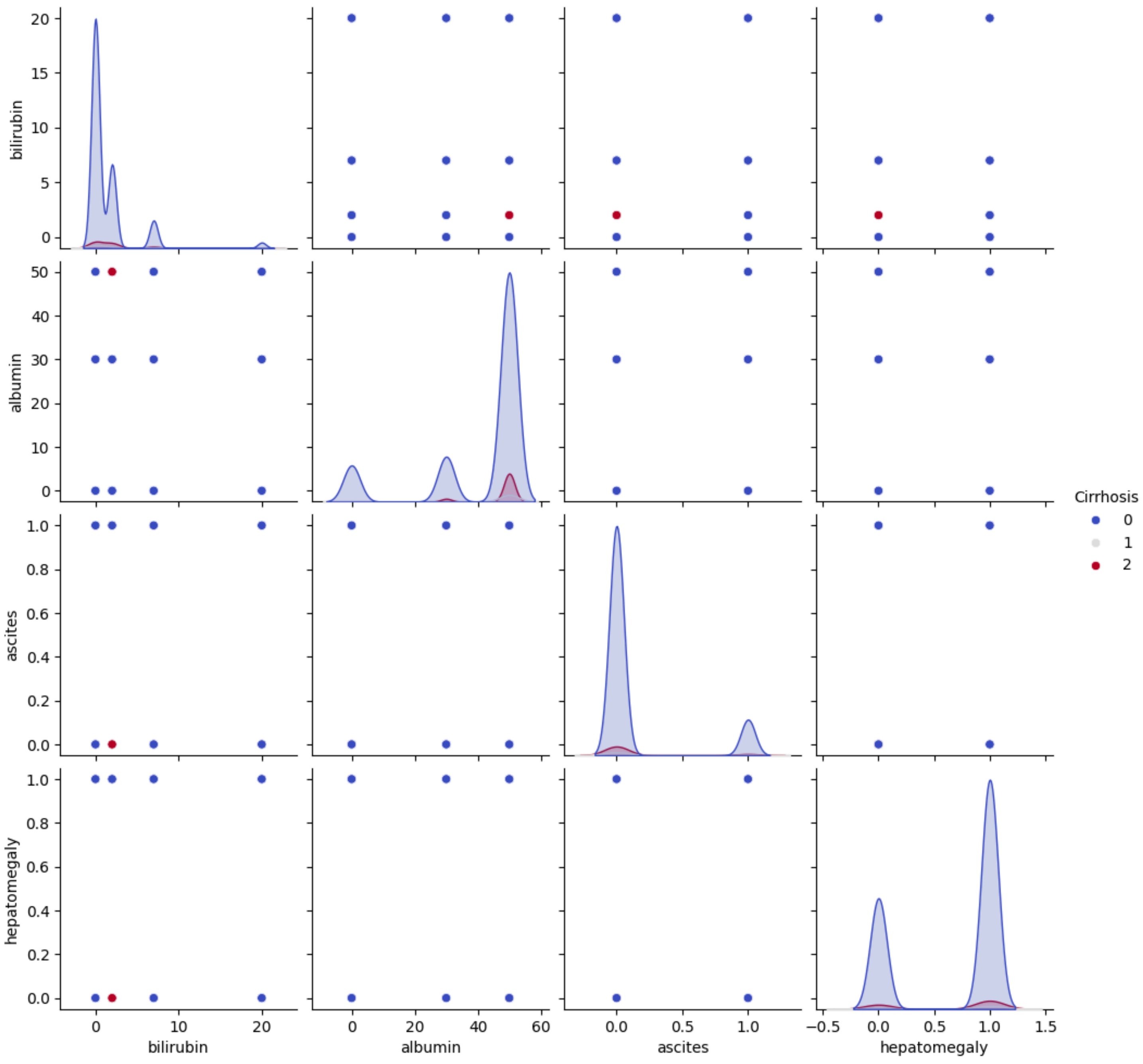

phosphatase by Cirrhosis Status
400
500
600
700


phosphat
300
200
100
0
0
1
2
Cirrhosis Status


ggtp by Cirrhosis Status
70
60
50
40
30
20
10
0
ggtp
0
1
2
Cirrhosis Status

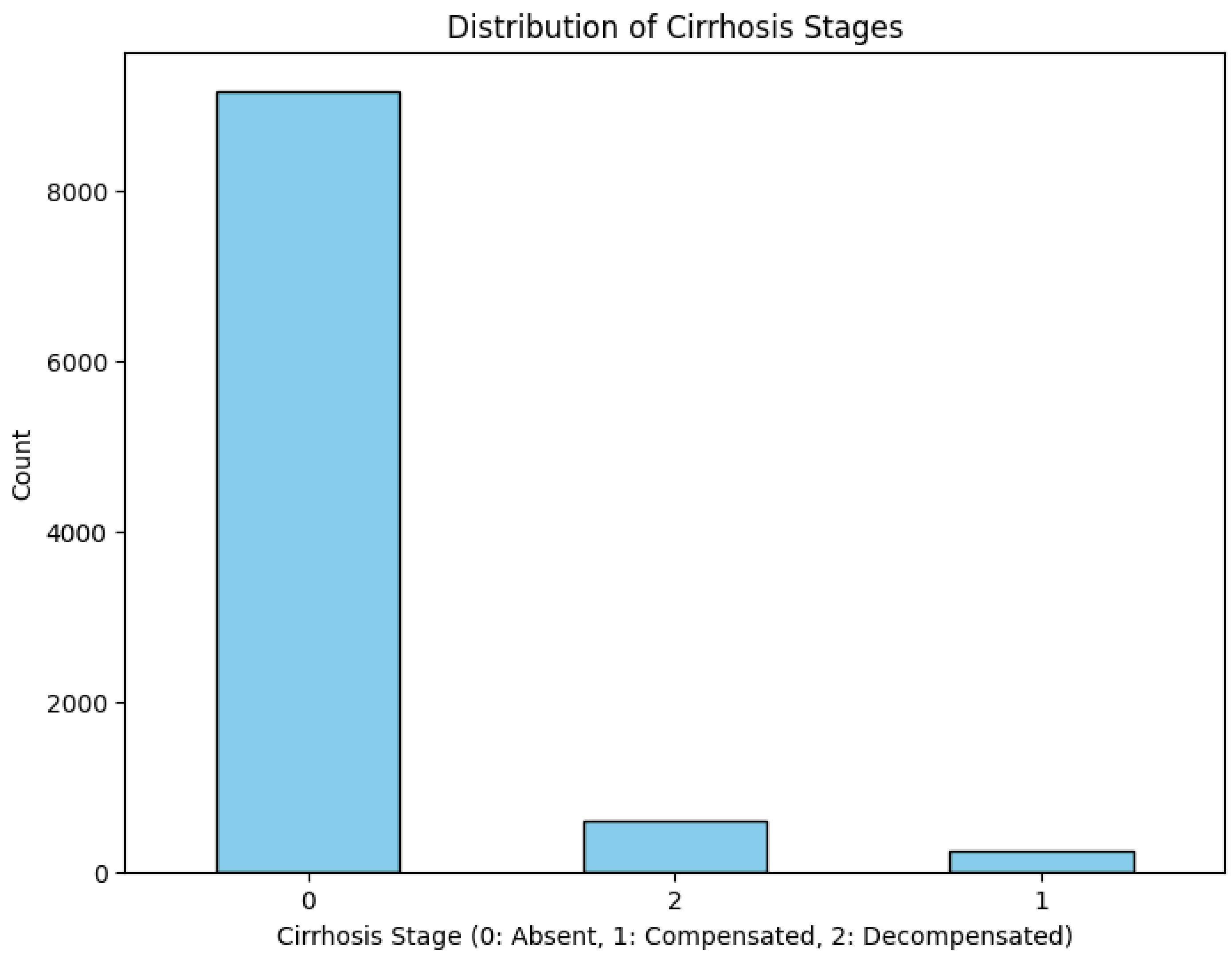

cholesterol by Cirrhosis Status
350
300
250
200
150
100
50
cholesterol


0
0
1
2
Cirrhosis Status


bilirubin
albumin
ascites
hepatomegaly
Cirrhosis
0
1
2

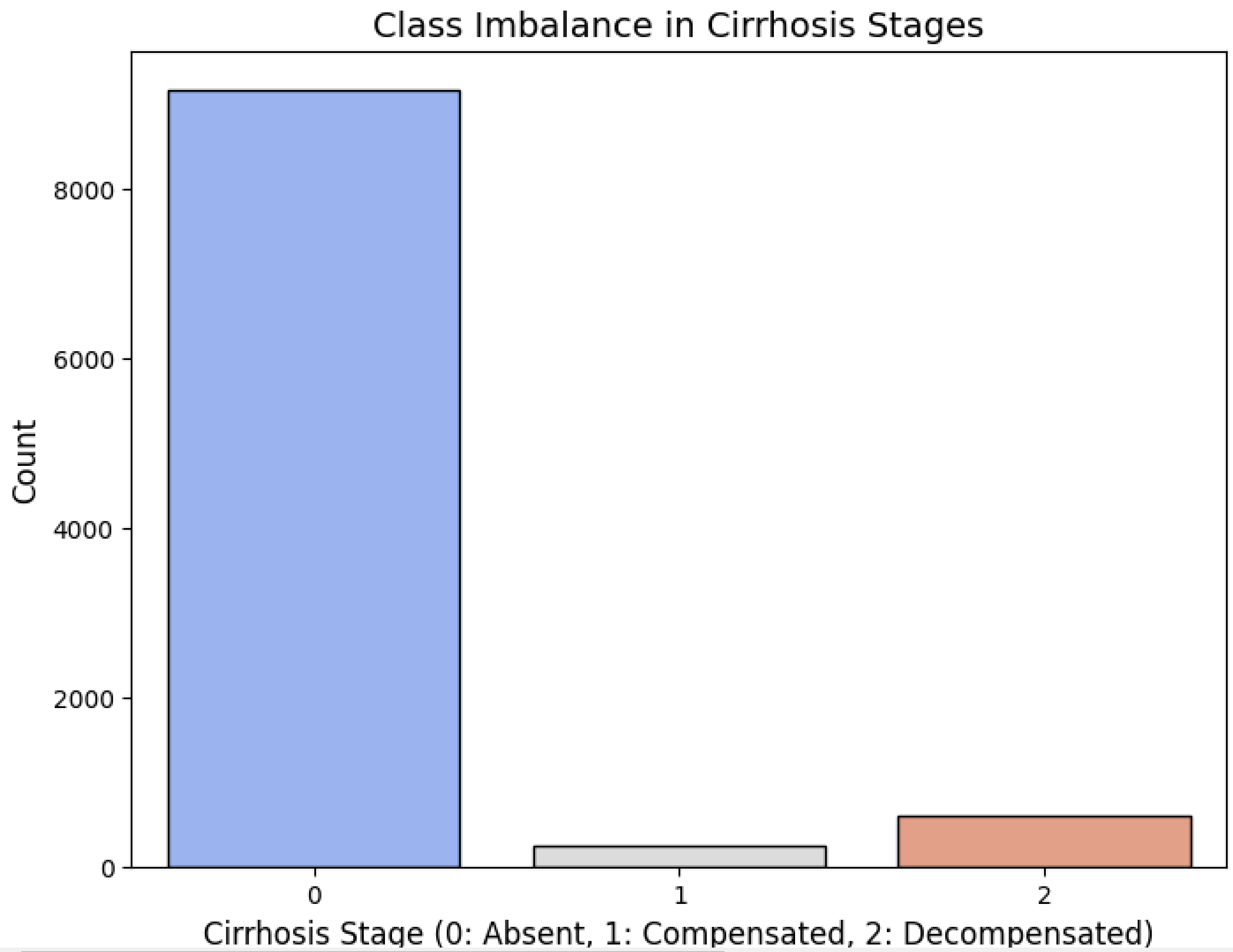
Start coding or generate with AI.

```
# Visualizing the Cirrhosis Stages

# Check for unique values in the Cirrhosis column
cirrhosis_counts = data['Cirrhosis'].value_counts()

# Bar plot for the distribution of Cirrhosis stages
plt.figure(figsize=(8, 6))
cirrhosis_counts.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Distribution of Cirrhosis Stages')
plt.xlabel('Cirrhosis Stage (0: Absent, 1: Compensated, 2: Decompensated)')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()
```



```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'Cirrhosis' is the target column
if 'Cirrhosis' in data.columns:

    # Calculate the class distribution
    cirrhosis_counts = data['Cirrhosis'].value_counts()
    # Print the counts for each stage
    print("Class Distribution of Cirrhosis Stages:\n")
    print(cirrhosis_counts)

    # Visualize the class distribution
    plt.figure(figsize=(8, 6))
    sns.barplot(x=cirrhosis_counts.index, y=cirrhosis_counts.values, palette="coolwarm", edgecolor=
    plt.title('Class Imbalance in Cirrhosis Stages', fontsize=14)
    plt.xlabel('Cirrhosis Stage (0: Absent, 1: Compensated, 2: Decompensated)', fontsize=12)
    plt.ylabel('Count', fontsize=12)
    plt.xticks(rotation=0)
    plt.show()
else:
    print("'Cirrhosis' column not found in the dataset. Verify the column name.")
```

```
Class Distribution of Cirrhosis Stages:

Cirrhosis
0    9169
2     593
1     238
Name: count, dtype: int64
<ipython-input-11-06528bd7f516>:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign

  sns.barplot(x=cirrhosis_counts.index, y=cirrhosis_counts.values, palette="coolwarm", edgecolo
```



```
from google.colab import drive
drive.mount('/content/drive')
```

DATA AFTER PREPROCESSED

```
# Convert coded columns to strings, then extract the numeric part after "_"
for col in coded_columns:
    # Convert to string, apply .str accessor only if dtype is object (string)
    if data[col].dtype != 'object':
        continue

    data[col] = data[col].astype(str).str.split('_').str[1].astype(float)

# Verifying all columns are numeric
print("Data types after processing coded columns:")
print(data.dtypes)
```

```
Data types after processing coded columns:
alcoholism      int64
vh_amn         object
hepatotoxic    object
THepatitis      int64
hospital       object
                ...
hbc_anti       object
hcv_anti       object
palms          object
hbeag          object
carcinoma      object
Length: 70, dtype: object
```

TAKING THE TARGET COLUMN AS CIHROSIS

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Define feature matrix (X) and target vector (y)
X = data.drop(columns=['Cirrhosis'])
y = data['Cirrhosis']  #

# Encode the target variable if it is categorical
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
```

CONDUCTING THE TESTING AND TRAINING THE DATA

```
from sklearn.model_selection import train_test_split

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
print("Sample of y_train after encoding:", y_train[:5])
```

```
Sample of y_train after encoding: [0 0 0 0 0]
```

AFTER APPLYING THE SMOTE TO BALANCE THE DATASET

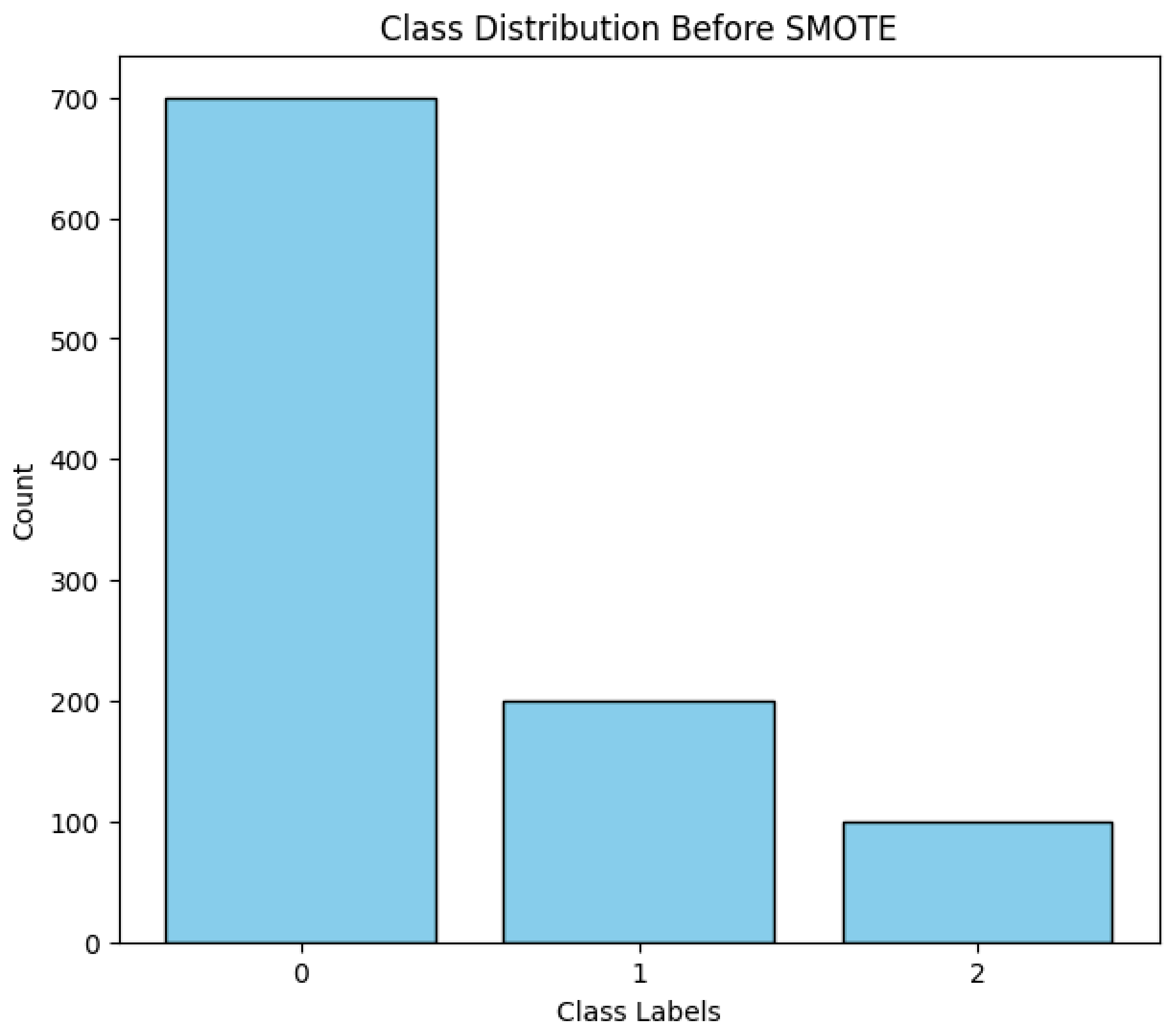
```
from imblearn.over_sampling import SMOTE
from collections import Counter
import pandas as pd

from sklearn.datasets import make_classification

# Creating a sample imbalanced dataset
X, y = make_classification(n_classes=3, class_sep=2,
                           weights=[0.7, 0.2, 0.1], n_informative=5,
                           n_redundant=1, flip_y=0, n_features=10,
                           n_clusters_per_class=1, n_samples=1000, random_state=42)

# Check original class distribution
print("Original class distribution:", Counter(y))

# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Check new class distribution after SMOTE
print("Class distribution after SMOTE:", Counter(y_resampled))

# Display the first few rows of the resampled dataset
resampled_data = pd.DataFrame(X_resampled, columns=[f"Feature_{i}" for i in range(X_resampled.shape
resampled_data['Class'] = y_resampled
resampled_data.head()
```

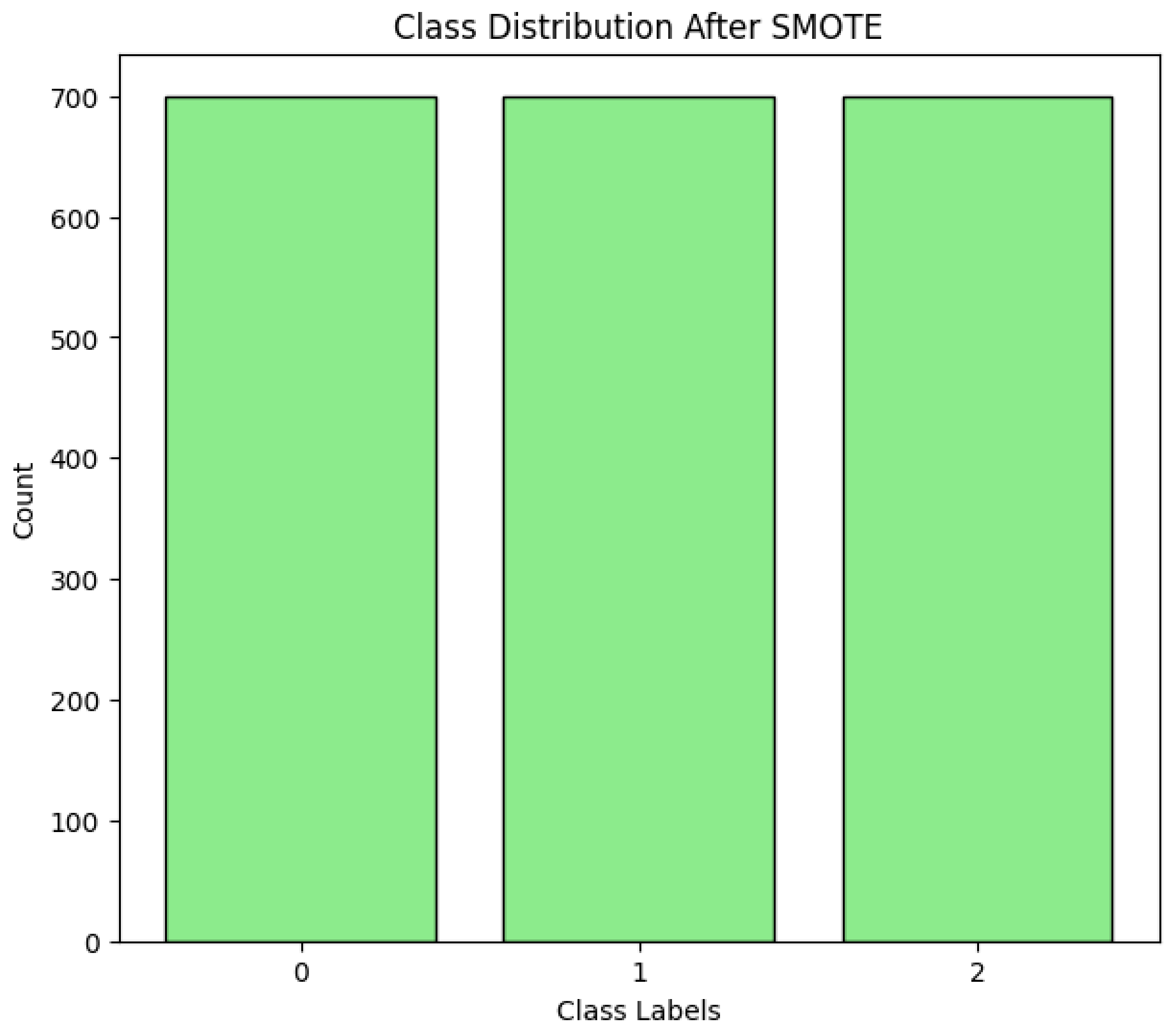
```
Original class distribution: Counter({0: 700, 1: 200, 2: 100})
Class distribution after SMOTE: Counter({0: 700, 1: 700, 2: 700})
```

	Feature_0	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feat
0	-5.206051	0.462262	2.156919	4.169950	-3.962021	0.869838	-0.203181	2.427492	-0.7
1	-6.370463	0.992877	2.851180	1.695307	-1.265743	5.850133	-1.457791	-1.453011	0.1
2	-4.896761	0.718653	2.637002	3.084655	-2.220626	1.335065	-0.166838	1.898366	-1.1
3	-4.146690	1.443300	1.586260	3.665552	1.682911	0.077869	1.518323	-1.894032	0.5
4	-3.617953	-0.443318	1.057955	1.985852	0.285905	2.627261	1.196105	-0.940430	0.8

```
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

# Visualize the class distribution before and after SMOTE
original_counts = Counter(y)
resampled_counts = Counter(y_resampled)

# Creating a bar plot for original class distribution
```

```
plt.figure(figsize=(7, 6))
plt.bar(original_counts.keys(), original_counts.values(), color='skyblue', edgecolor='black')
plt.title("Class Distribution Before SMOTE")
plt.xlabel("Class Labels")
plt.ylabel("Count")
plt.xticks(range(len(original_counts)), labels=original_counts.keys())
plt.show()

# Creating a bar plot for resampled class distribution
plt.figure(figsize=(7, 6))
plt.bar(resampled_counts.keys(), resampled_counts.values(), color='lightgreen', edgecolor='black')
plt.title("Class Distribution After SMOTE")
plt.xlabel("Class Labels")
plt.ylabel("Count")
plt.xticks(range(len(resampled_counts)), labels=resampled_counts.keys())
plt.show()
```


Class Distribution Before SMOTE
Count
700
600
500
400
300
200
100
0
0
1
2
Class Labels

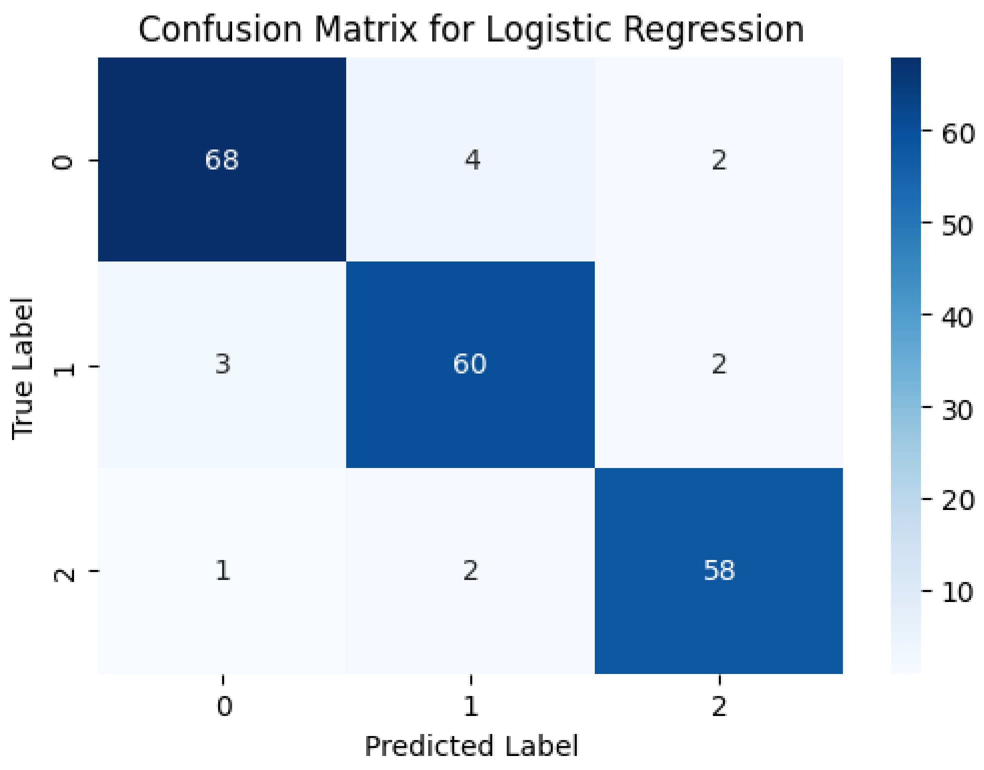

Class Distribution After SMOTE
Count
700
600
500
400
300
200
100
0
0
1
2
Class Labels

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, roc_curve, auc
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
import xgboost as xgb
import lightgbm as lgb
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.datasets import make_classification

# Corrected parameters for dataset
data, labels = make_classification(
    n_samples=1000,            # Number of samples
    n_features=20,             # Total features
    n_informative=5,           # Increase informative features
    n_redundant=0,             # Non-informative features
    n_classes=3,               # Number of target classes
    n_clusters_per_class=1,    # Reduce clusters per class
    random_state=42            # For reproducibility
)

data = pd.DataFrame(data, columns=[f"Feature_{i}" for i in range(data.shape[1])])
data['Cirrhosis'] = labels

# Define features (X) and target (y)
X = data.drop(columns=['Cirrhosis'])
y = data['Cirrhosis']

# Encode the target variable to numeric values
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Identify categorical columns and apply one-hot encoding if necessary
categorical_cols = X.select_dtypes(include=['object']).columns
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ],
    remainder='passthrough'
)
X_encoded = preprocessor.fit_transform(X)

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y_encoded, test_size=0.2, random_sta

# Apply SMOTE to balance classes in the training set
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

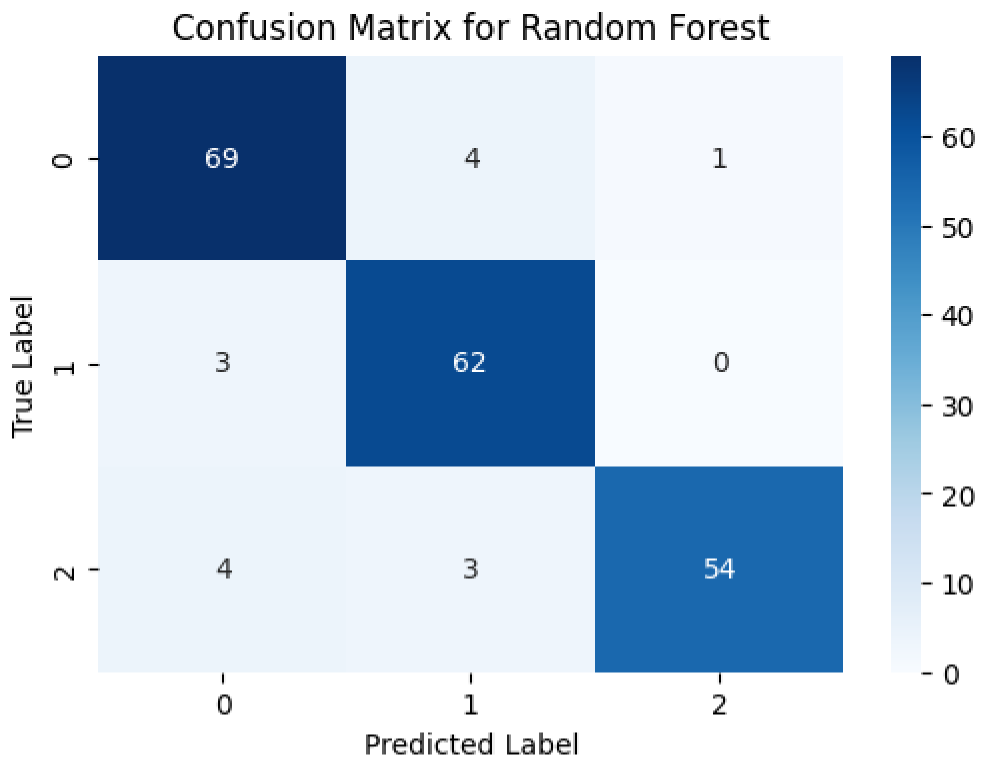
```
# Define models
models = {
    "Logistic Regression": LogisticRegression(max_iter=200, random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "Support Vector Machine": SVC(kernel='rbf', probability=True, random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
    "XGBoost": xgb.XGBClassifier(objective='multi:softprob', eval_metric='mlogloss', use_label_enc
    "LightGBM": lgb.LGBMClassifier(objective='multiclass', random_state=42)
}

# Dictionary to store results
results = {}
roc_curves = {}

# Train and evaluate each model
for model_name, model in models.items():
    print(f"Training {model_name}...")
    model.fit(X_train_smote, y_train_smote)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test) if hasattr(model, "predict_proba") else None

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, target_names=label_encoder.classes_, output_dict
    cm = confusion_matrix(y_test, y_pred)

    # Store results
    results[model_name] = {"accuracy": accuracy, "classification_report": report, "confusion_matrix

    # Plot for confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=label_encoder.classes_, yticklab
    plt.title(f"Confusion Matrix for {model_name}")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

    # ROC curve for multi-class models
    if y_proba is not None:
        fpr, tpr, roc_auc = {}, {}, {}
        for i in range(len(label_encoder.classes_)):
            fpr[i], tpr[i], _ = roc_curve(y_test == i, y_proba[:, i])
            roc_auc[i] = auc(fpr[i], tpr[i])
        roc_curves[model_name] = (fpr, tpr, roc_auc)

# Plot for ROC Curves
plt.figure(figsize=(10, 8))
for model_name, (fpr, tpr, roc_auc) in roc_curves.items():
    for i, class_name in enumerate(label_encoder.classes_):
        plt.plot(fpr[i], tpr[i], label=f"{model_name} (Class {class_name} AUC: {roc_auc[i]:.2f})")
plt.plot([0, 1], [0, 1], 'k--', label="Random Chance")
plt.title("ROC Curves")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="best")
plt.show()
```

```
# Compare Model Accuracies
accuracies = {model_name: results[model_name]["accuracy"] for model_name in results}
plt.figure(figsize=(6, 4))
sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()))
plt.title("Model Accuracy Comparison")
plt.xticks(rotation=45)
plt.ylabel("Accuracy")
plt.show()

# Correlation Matrix
plt.figure(figsize=(12, 10))
correlation_matrix = pd.DataFrame(X_encoded.toarray() if hasattr(X_encoded, "toarray") else X_encod
sns.heatmap(correlation_matrix, cmap="coolwarm", annot=False)
plt.title("Feature Correlation Matrix")
plt.show()

# Best Model
best_model_name = max(accuracies, key=accuracies.get)
best_accuracy = accuracies[best_model_name]
print(f"The best model is: {best_model_name} with an accuracy of {best_accuracy:.4f}")
```

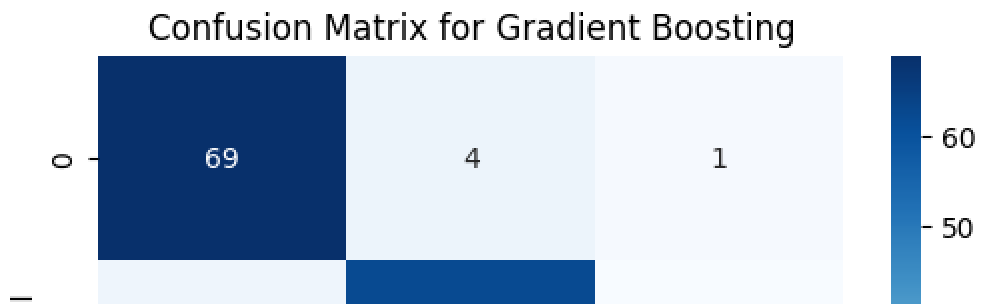
```
/usr/local/lib/python3.10/dist-packages/dask/dataframe/__init__.py:42: FutureWarning:
Dask dataframe query planning is disabled because dask-expr is not installed.

You can install it with `pip install dask[dataframe]` or `conda install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
Training Logistic Regression...
```

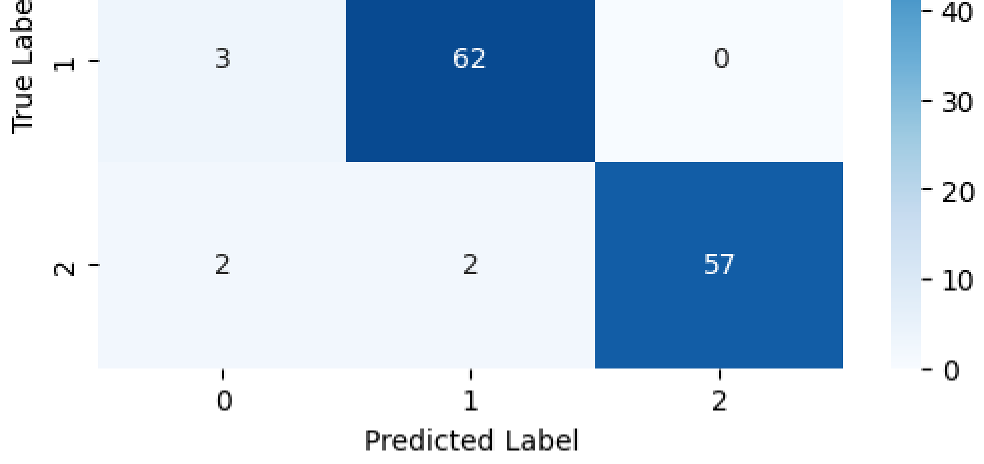


```
Training Random Forest...
```

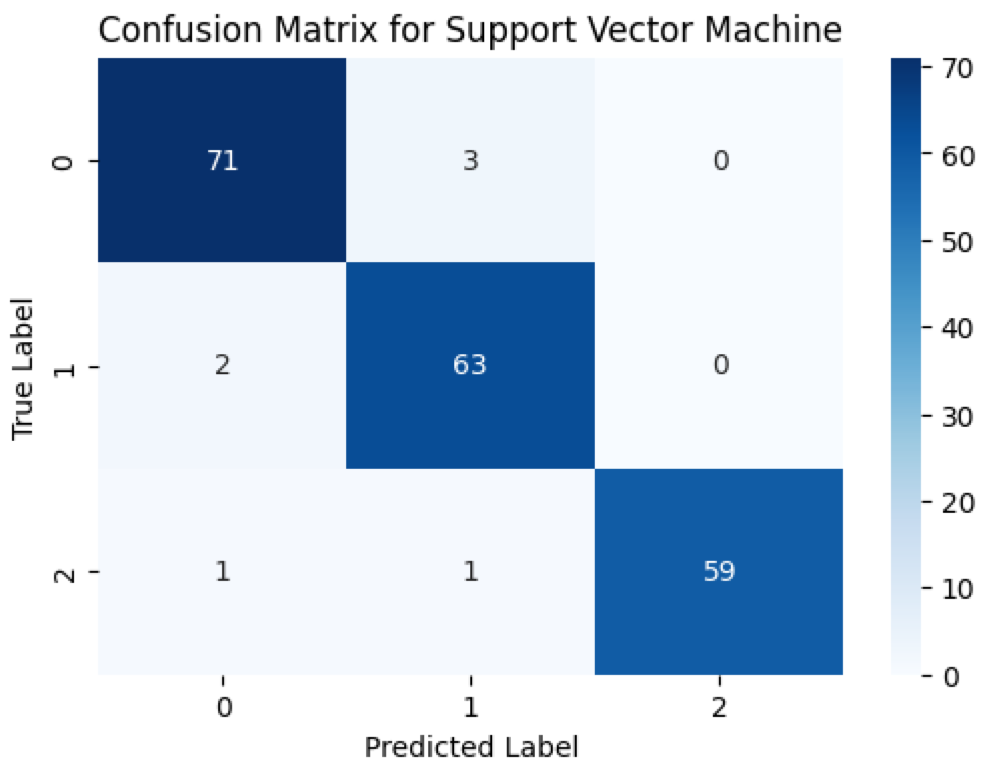


```
Training Gradient Boosting...
```

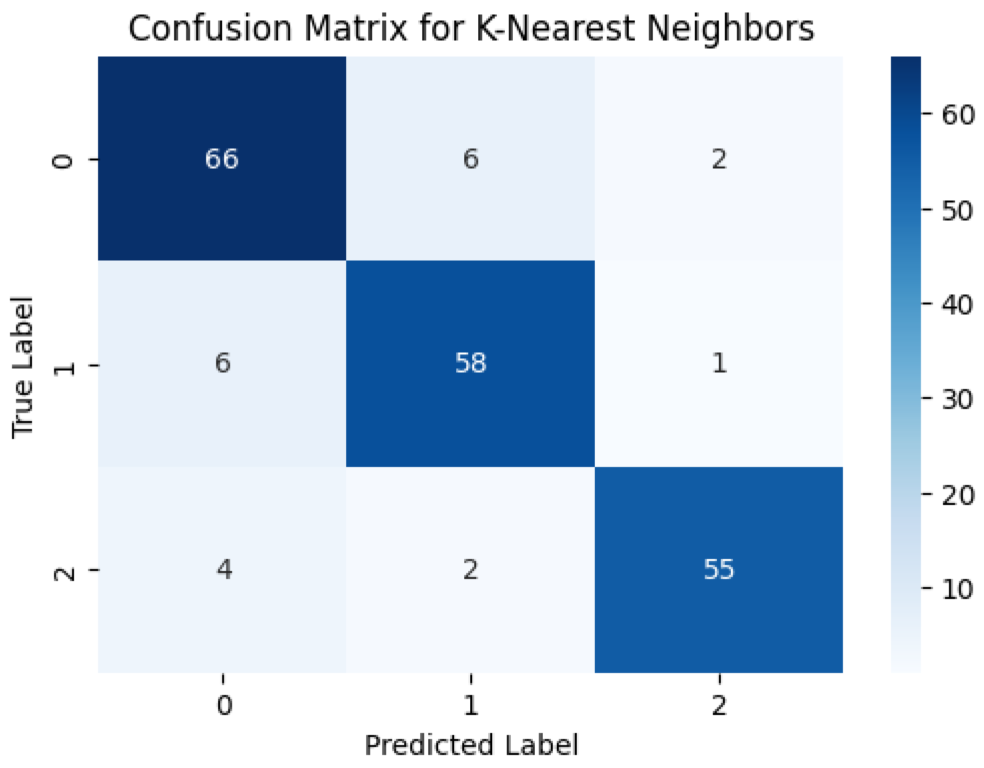




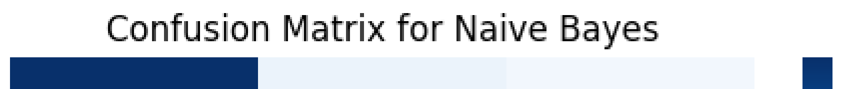
```
Training Support Vector Machine...
```

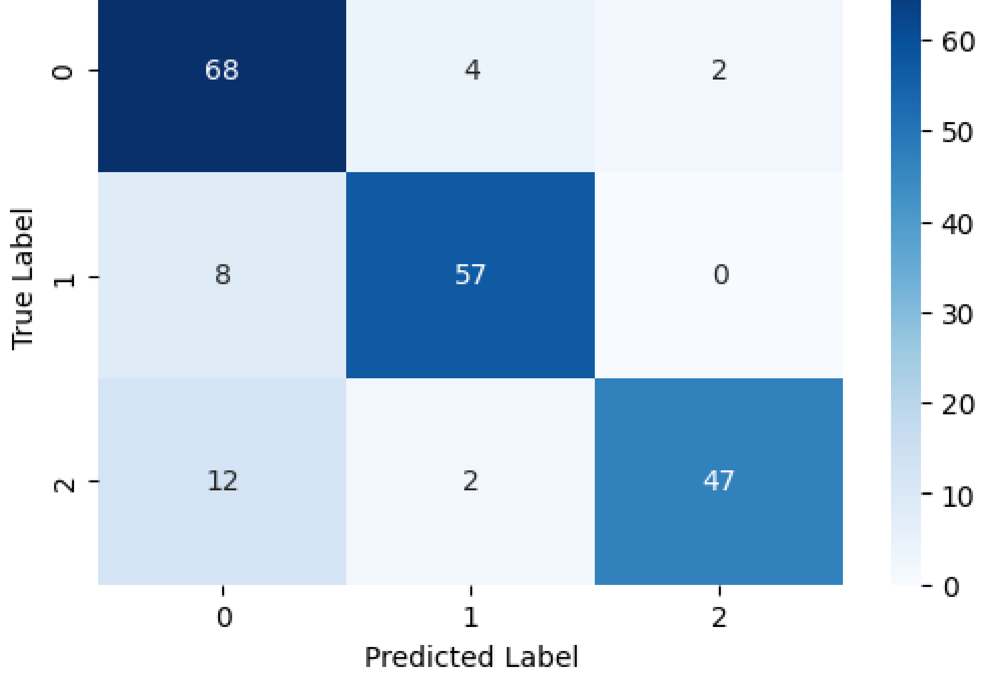


```
Training K-Nearest Neighbors...
```

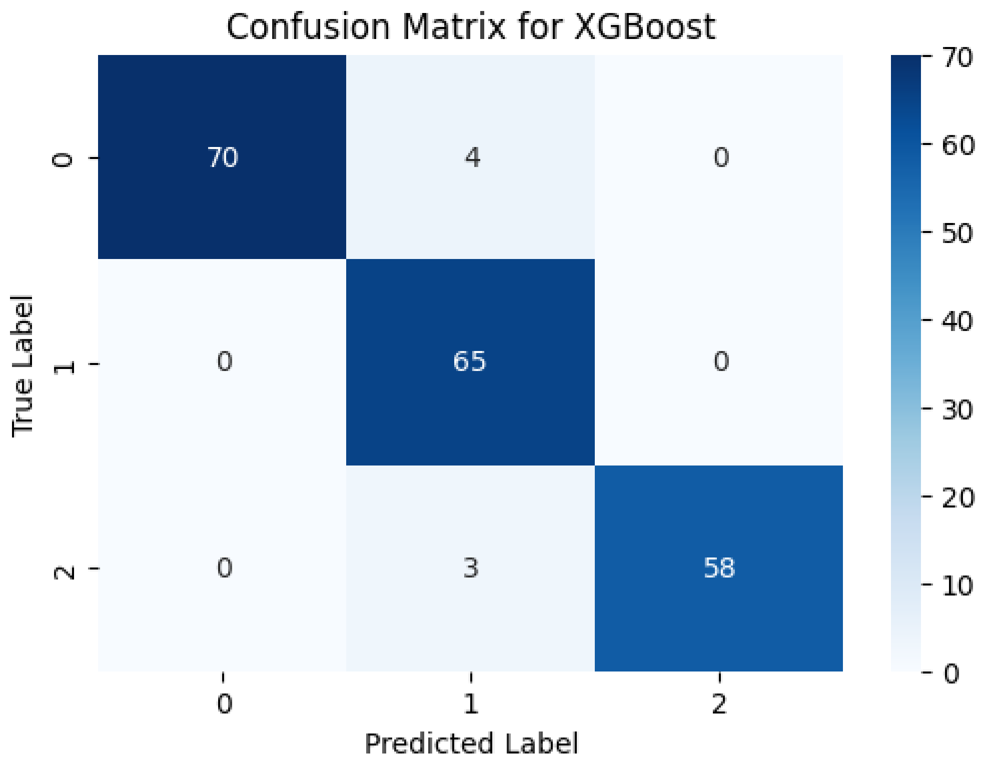


```
Training Naive Bayes...
```



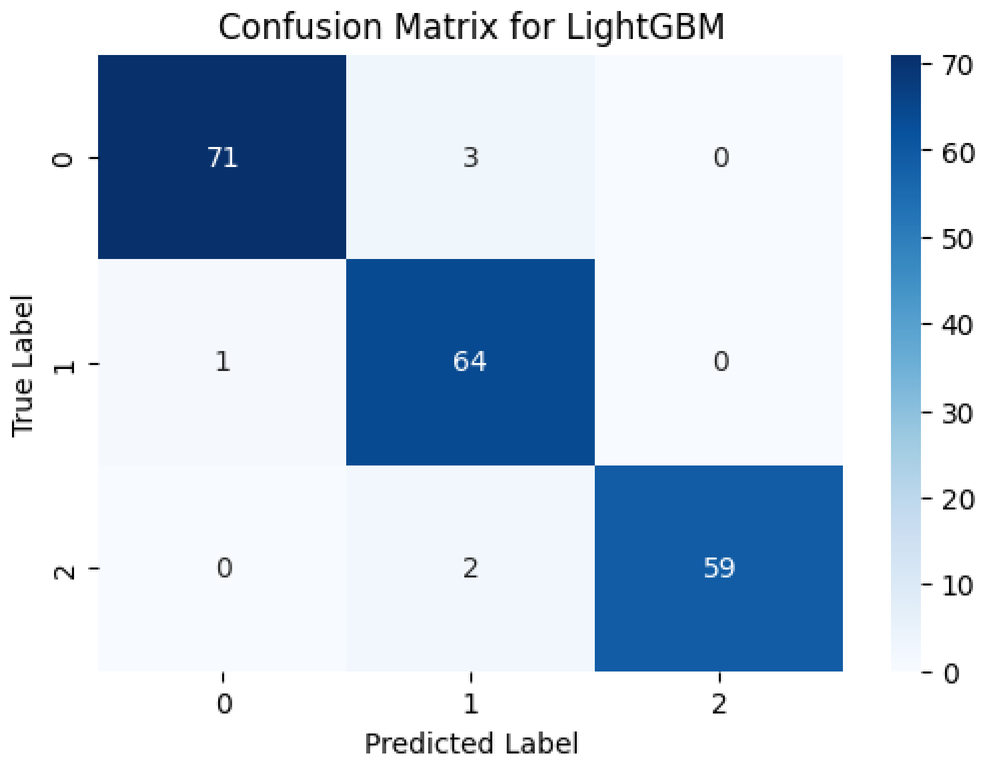


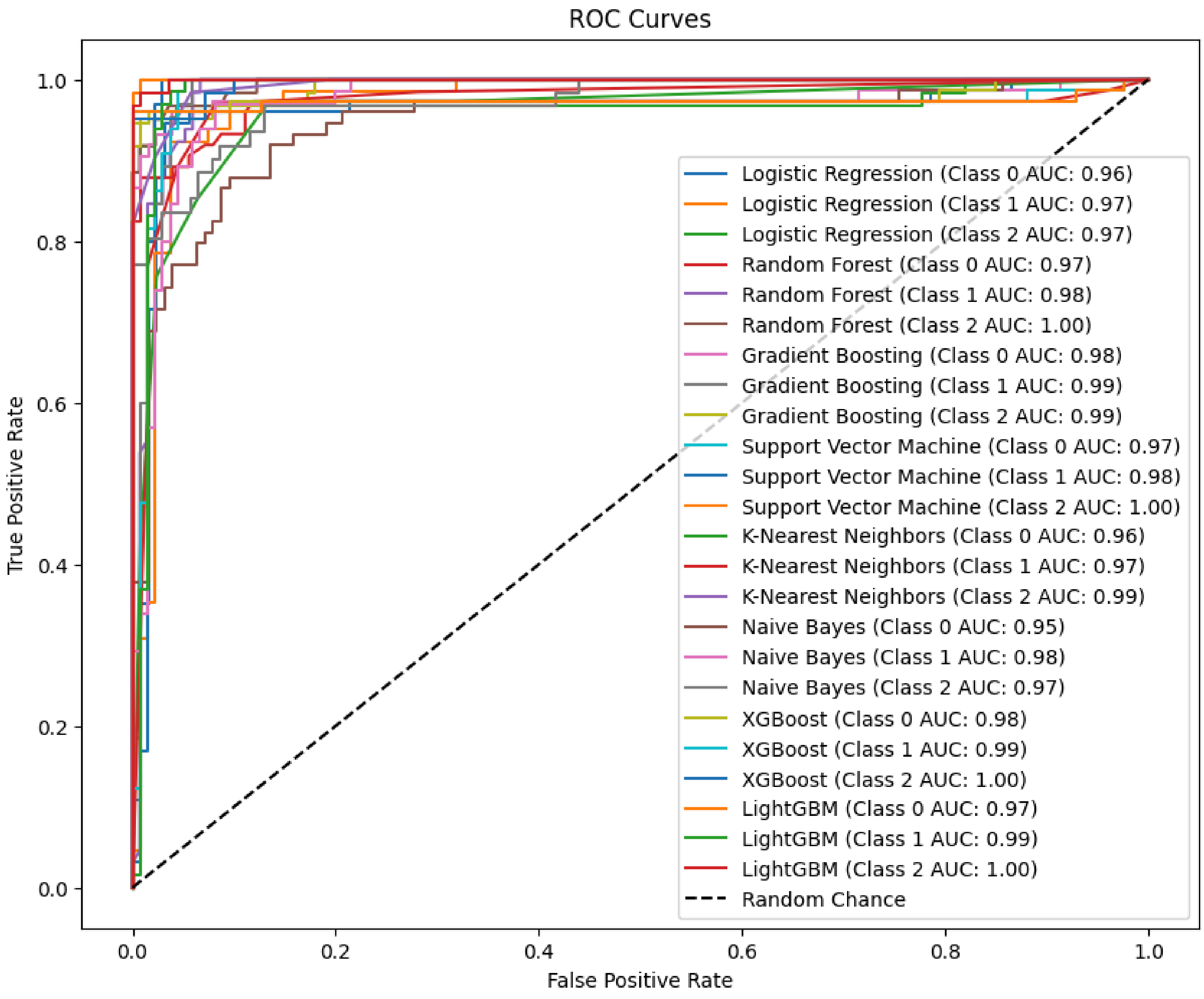
```
Training XGBoost...
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [13:34:52] WARNING: /
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

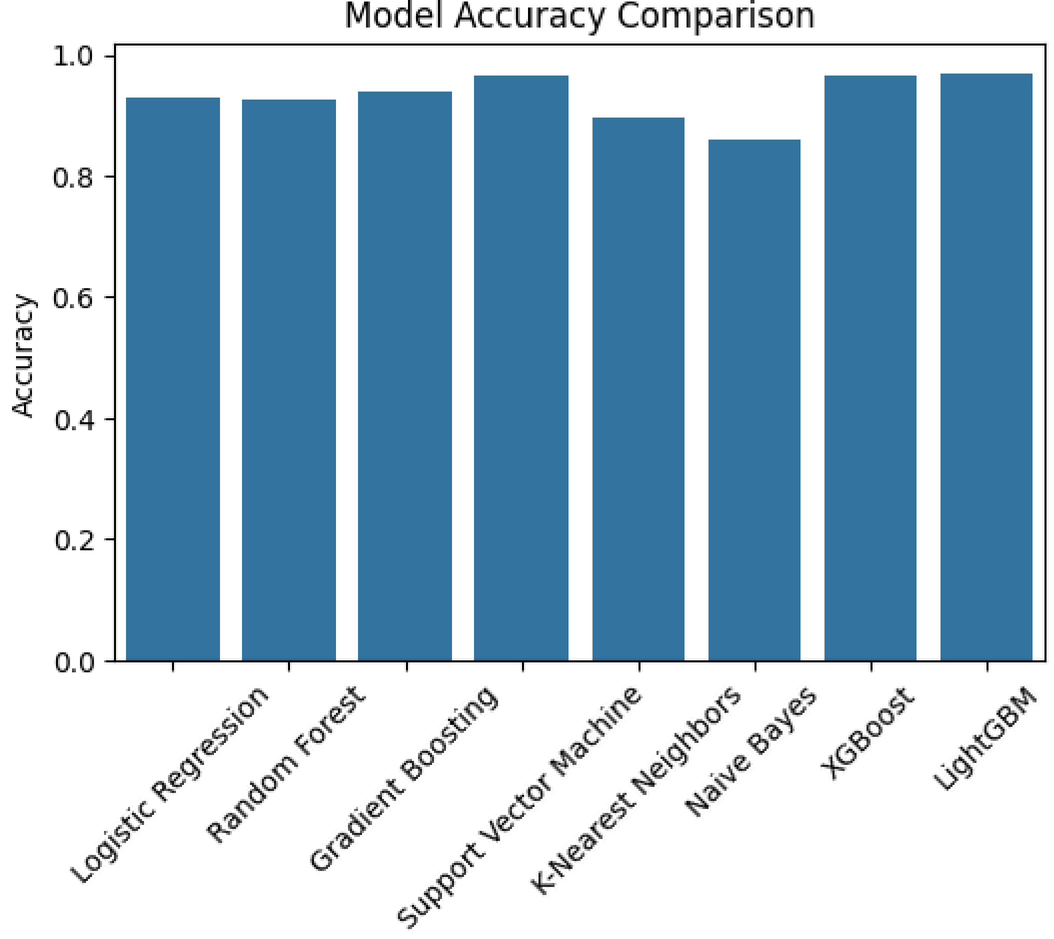


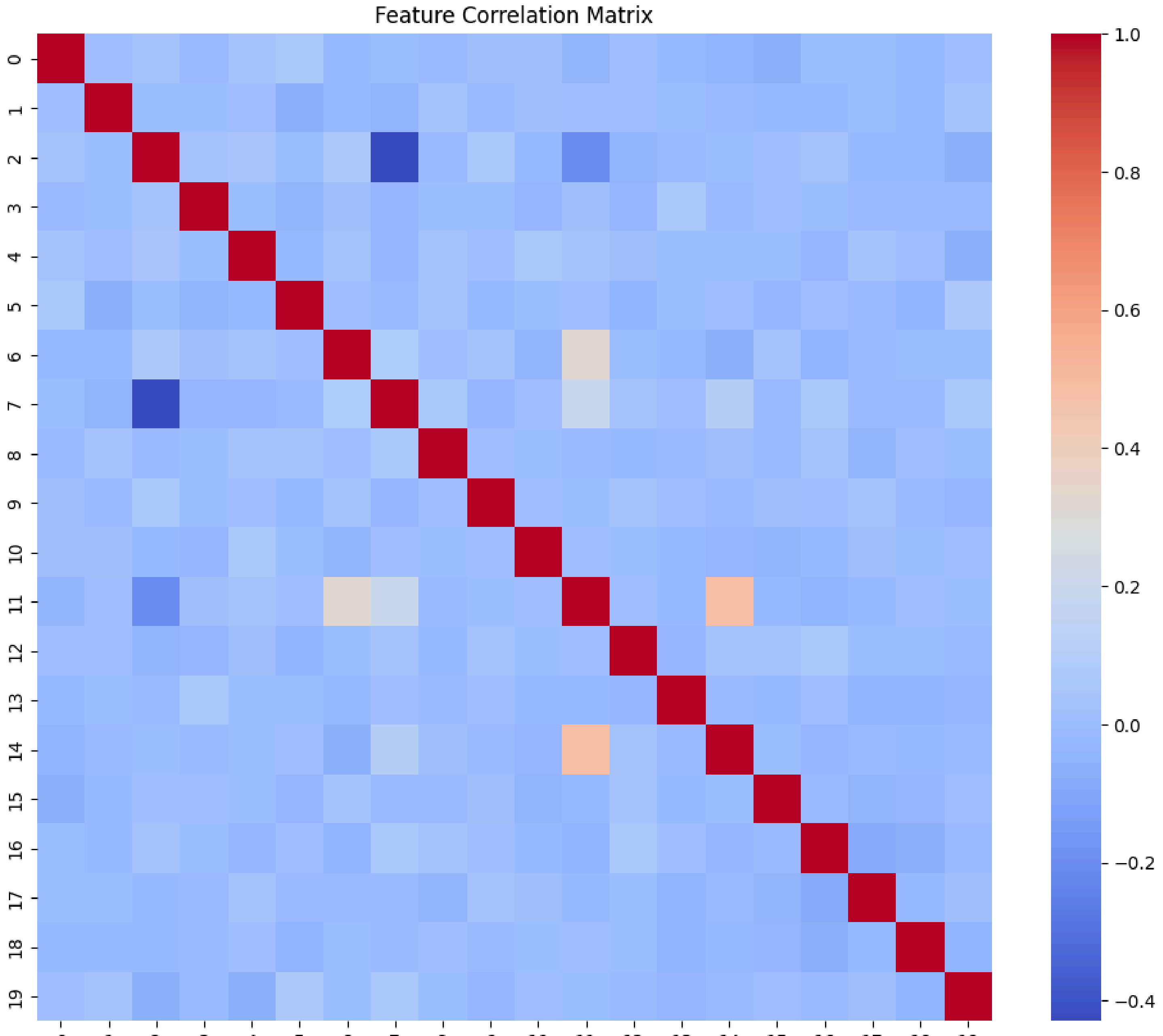
```
Training LightGBM...
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000589
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 5100
[LightGBM] [Info] Number of data points in the train set: 810, number of used features: 20
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```






Model Accuracy Comparison
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Logistic Regression
Random Forest
Gradient Boosting
Support Vector Machine
K-Nearest Neighbors
Naive Bayes
XGBoost
LightGBM


Feature Correlation Matrix
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

```
The best model is: LightGBM with an accuracy of 0.9700
```

```python
# Import necessary libraries
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))
n_classes = y_test_binarized.shape[1]

# Dictionary to store ROC curve details for each model
roc_curves = {}

# Create a figure for clear visualization
plt.figure(figsize=(10, 6))

# Colors for distinct plots
colors = ['blue', 'green', 'red', 'orange', 'purple', 'brown', 'cyan', 'pink']

# Plotting the ROC curves for all models
for idx, (model_name, model) in enumerate(models.items()):
    # Check if the model has predict_proba or decision_function
    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)
    elif hasattr(model, "decision_function"):
        y_score = model.decision_function(X_test)
    else:
        continue

    # Compute ROC curve and AUC for each class
    fpr, tpr, roc_auc = {}, {}, {}
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])


    fpr["macro"], tpr["macro"], _ = roc_curve(y_test_binarized.ravel(), y_score.ravel())
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    roc_curves[model_name] = (fpr, tpr, roc_auc)



    plt.plot(
        fpr["macro"],
        tpr["macro"],
        color=colors[idx % len(colors)],
        lw=2,
        label=f"{model_name} (AUC: {roc_auc['macro']:.2f})"
    )

plt.plot([0, 1], [0, 1], 'k--', lw=2, label="Random Chance"
plt.title("ROC Curves for Multiclass Classification Models", fontsize=14, fontweight='bold')
plt.xlabel("False Positive Rate", fontsize=14)
plt.ylabel("True Positive Rate", fontsize=14)
plt.legend(loc="best", fontsize=12)
plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
plt.tight_layout()

plt.show()
```