project

November 14, 2024

1 Data 1030 Project

1.1 Name: Jimmy Lin

1.2 Link to your project github repo: (https://github.com/sirjimmylin/DATA-1030-Project.git)

1.3 How long did you work on this project? 15.25 hours

Make sure you are in the DATA1030 environment.

1.4 Conda Environment Setup

[55]:
```
from __future__ import print_function
from packaging.version import parse as Version
from platform import python_version

OK = '\x1b[42m[ OK ]\x1b[0m'
FAIL = "\x1b[41m[FAIL]\x1b[0m"

try:
    import importlib
except ImportError:
    print(FAIL, "Python version 3.12 is required,"
                " but %s is installed." % sys.version)

def import_version(pkg, min_ver, fail_msg=""):
    mod = None
    try:
        mod = importlib.import_module(pkg)
        if pkg in {'PIL'}:
            ver = mod.VERSION
        else:
            ver = mod.__version__
        if Version(ver) == Version(min_ver):
            print(OK, "%s version %s is installed."
                  % (lib, min_ver))
        else:
            print(FAIL, "%s version %s is required, but %s installed."
```

```
                % (lib, min_ver, ver))
    except ImportError:
        print(FAIL, '%s not installed. %s' % (pkg, fail_msg))
    return mod


# first check the python version
pyversion = Version(python_version())

if pyversion >= Version("3.12.5"):
    print(OK, "Python version is %s" % pyversion)
elif pyversion < Version("3.12.5"):
    print(FAIL, "Python version 3.12.5 is required,"
                " but %s is installed." % pyversion)
else:
    print(FAIL, "Unknown Python version: %s" % pyversion)


print()
requirements = {'numpy': "1.26.4", 'matplotlib': "3.9.2",'sklearn': "1.5.1",
                'pandas': "2.2.2",'xgboost': "2.1.1", 'shap': "0.45.1",
                'plotly': "5.23.0"}

# now the dependencies
for lib, required_version in list(requirements.items()):
    import_version(lib, required_version)
```

```
[ OK ] Python version is 3.12.7
```

```
[ OK ] numpy version 1.26.4 is installed.
[ OK ] matplotlib version 3.9.2 is installed.
[ OK ] sklearn version 1.5.1 is installed.
[ OK ] pandas version 2.2.2 is installed.
[ OK ] xgboost version 2.1.1 is installed.
[ OK ] shap version 0.45.1 is installed.
[ OK ] plotly version 5.23.0 is installed.
```

1.5 Step 1: EDA

1.5.1 Read in Data

[56]:
```
import pandas as pd
import numpy as np

#read in txt file using pandas read_csv function with a tab delimiter
df = pd.read_csv('hcvdat0.csv')

#display the first few rows of the table
```

```
df.head()
```

[56]:

	Unnamed: 0	Category	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CREA	GGT	PROT
0	1	0=Blood Donor	32	m	38.5	52.5	7.7	22.1	7.5	6.93	3.23	106.0	12.1	69.0
1	2	0=Blood Donor	32	m	38.5	70.3	18.0	24.7	3.9	11.17	4.80	74.0	15.6	76.5
2	3	0=Blood Donor	32	m	46.9	74.7	36.2	52.6	6.1	8.84	5.20	86.0	33.2	79.3
3	4	0=Blood Donor	32	m	43.2	52.0	30.6	22.6	18.9	7.33	4.74	80.0	33.8	75.7
4	5	0=Blood Donor	32	m	39.2	74.1	32.6	24.8	9.6	9.15	4.32	76.0	29.9	68.7

[57]:
```
#Number of rows and columns in the dataset (rows,columns)
print(df.shape)

# Print the data types
print(df.dtypes.to_string())

print(df['Category'].value_counts())
print('This column is ordinal.')
```

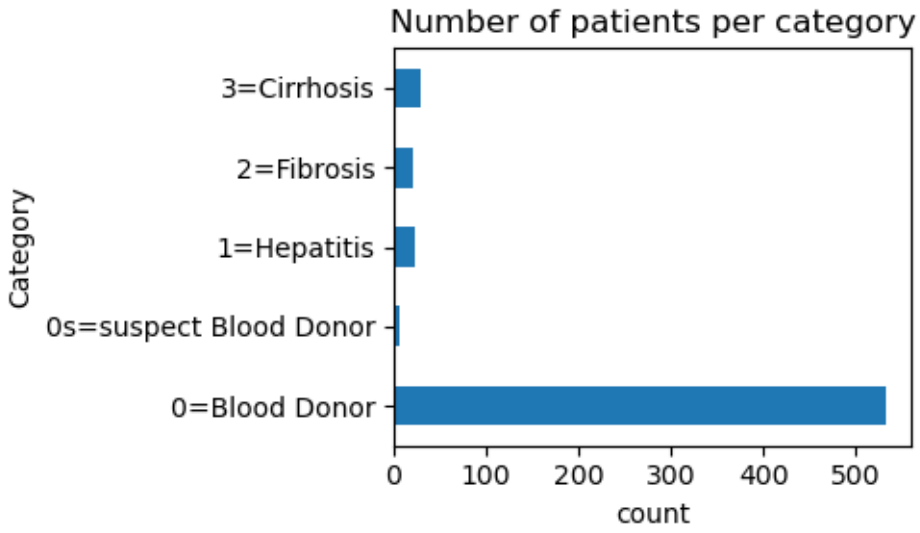
```
(615, 14)
Unnamed: 0      int64
Category       object
Age             int64
Sex            object
ALB           float64
ALP           float64
ALT           float64
AST           float64
BIL           float64
CHE           float64
CHOL          float64
CREA          float64
GGT           float64
PROT          float64
Category
0=Blood Donor             533
3=Cirrhosis                30
1=Hepatitis                24
2=Fibrosis                 21
0s=suspect Blood Donor      7
Name: count, dtype: int64
This column is ordinal.
```

1.5.2 Visualize Target Variable

[58]:
```
correct_order = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
↪'2=Fibrosis', '3=Cirrhosis']

df['Category'].value_counts().reindex(correct_order)
```

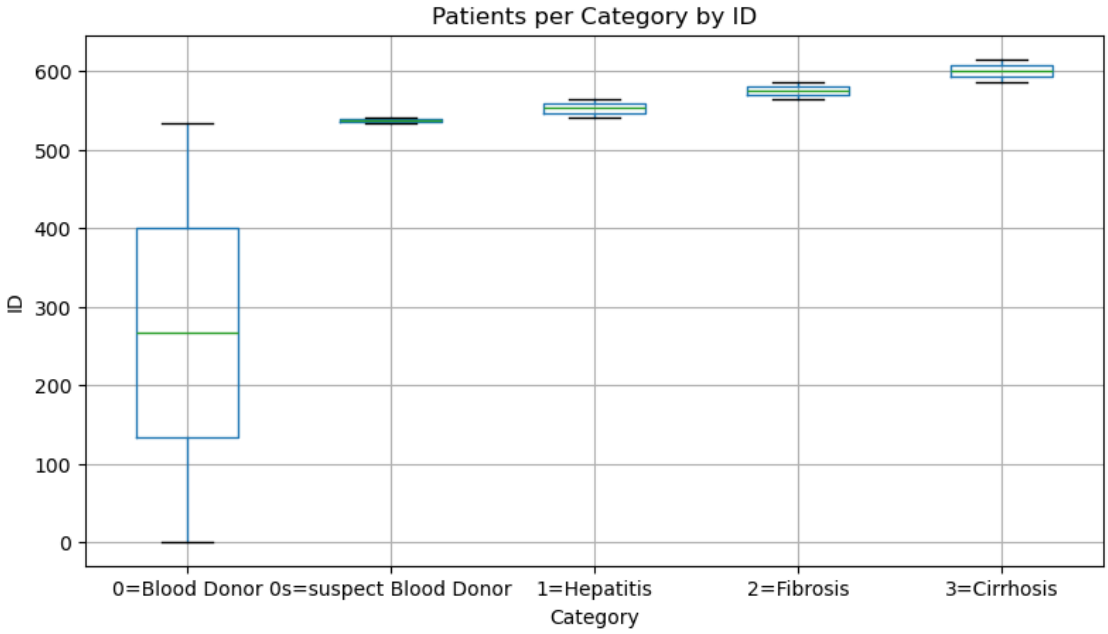
[58]:
```
Category
0=Blood Donor              533
0s=suspect Blood Donor       7
1=Hepatitis                 24
2=Fibrosis                  21
3=Cirrhosis                 30
Name: count, dtype: int64
```

[59]:
```
# import pandas as pd
# import numpy as np
# import matplotlib
from matplotlib import pylab as plt

plt.figure(figsize=(5,3))

df['Category'].value_counts().reindex(correct_order).plot.barh()
plt.xlabel('count')
plt.ylabel('Category')
plt.title('Number of patients per category')
plt.tight_layout()
plt.show()
```



1.5.3 Column Pair Plots with Target Variable (Category)

[60]:
```
df[['Category','Unnamed: 0']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by ID')
plt.xlabel('Category')
plt.ylabel('ID')
plt.show()
print('This box plot groups the ID by category.')
```

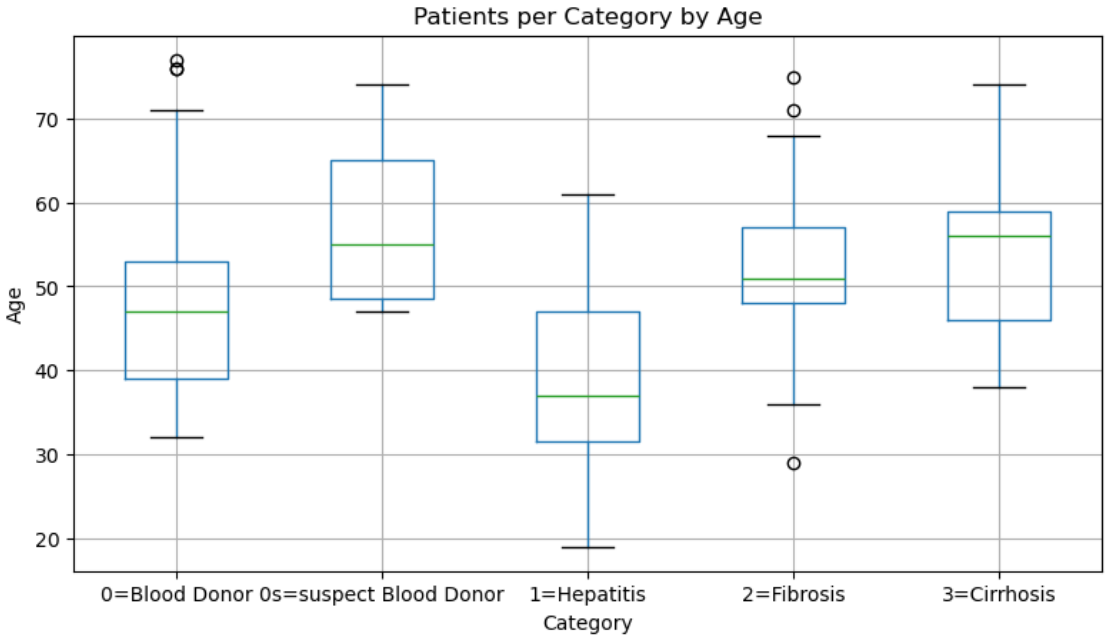



```
This box plot groups the ID by category.
```

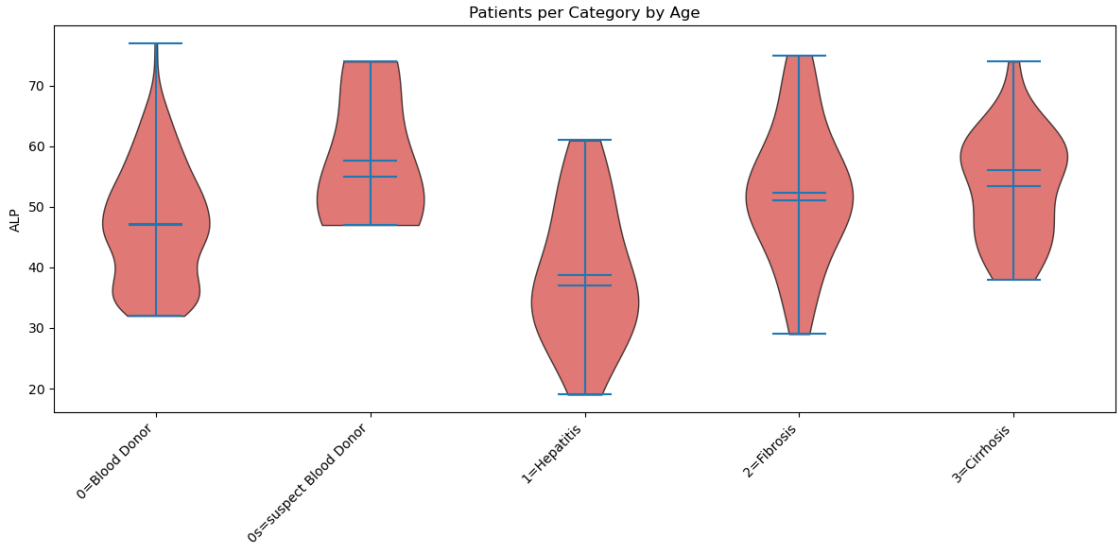
[61]:
```
df[['Category','Age']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by Age')
plt.xlabel('Category')
plt.ylabel('Age')
plt.show()
print('This box plot groups the age by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['Age'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['Age'].dropna().values,
```

```
    df[df['Category'] == '1=Hepatitis']['Age'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['Age'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['Age'].dropna().values
]

# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]

plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('ALP')
plt.title('Patients per Category by Age')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the age by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
bin_range = (df['Age'].min(),df['Age'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['Age'],alpha=0.
 ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
```

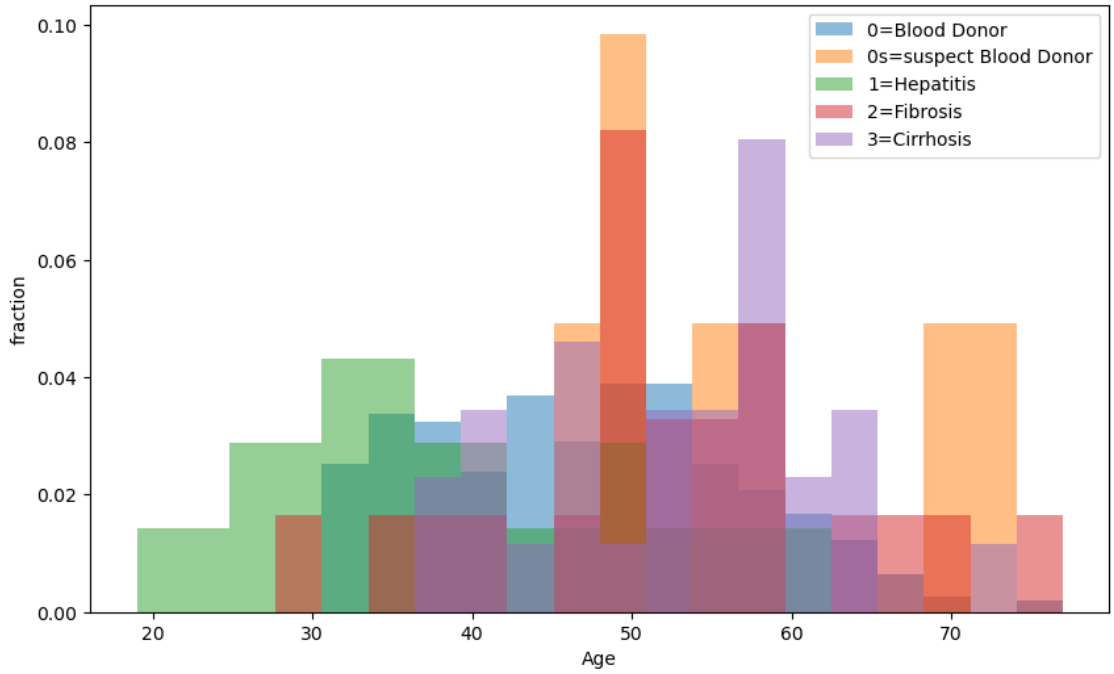
```
plt.xlabel('Age')
plt.show()
print('This category-specific histogram groups the age by category.')
```



This box plot groups the age by category.



This violin plot groups the age by category.



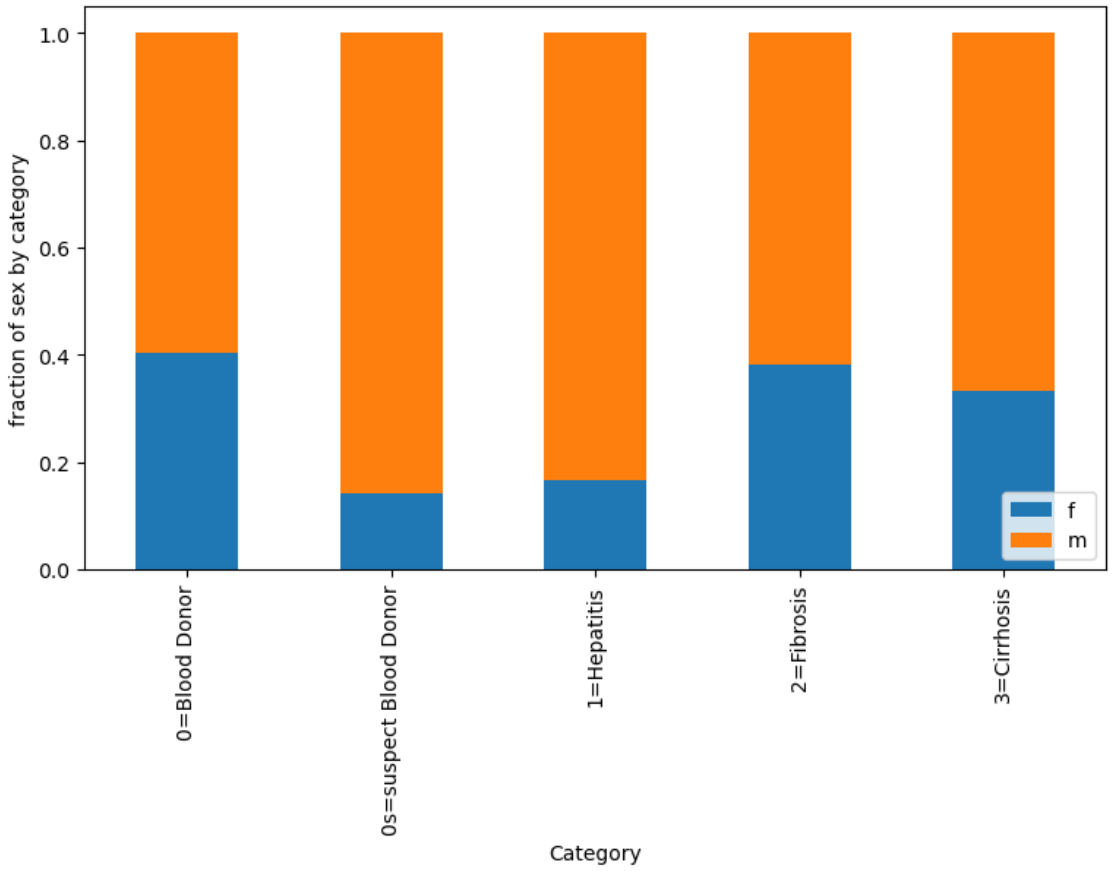
This category-specific histogram groups the age by category.

[62]:
```
count_matrix = df.groupby(['Category', 'Sex']).size().unstack()
print(count_matrix)

count_matrix_norm = count_matrix.div(count_matrix.sum(axis=1),axis=0)
print(count_matrix_norm)
```

```
Sex                      f    m
Category
0=Blood Donor          215  318
0s=suspect Blood Donor   1    6
1=Hepatitis              4   20
2=Fibrosis               8   13
3=Cirrhosis             10   20
Sex                           f         m
Category
0=Blood Donor          0.403377  0.596623
0s=suspect Blood Donor 0.142857  0.857143
1=Hepatitis            0.166667  0.833333
2=Fibrosis             0.380952  0.619048
3=Cirrhosis            0.333333  0.666667
```

[63]:
```
count_matrix_norm.plot(kind='bar', stacked=True,figsize=(9,5))
plt.ylabel('fraction of sex by category')
plt.legend(loc=4)
```

```
plt.show()
```



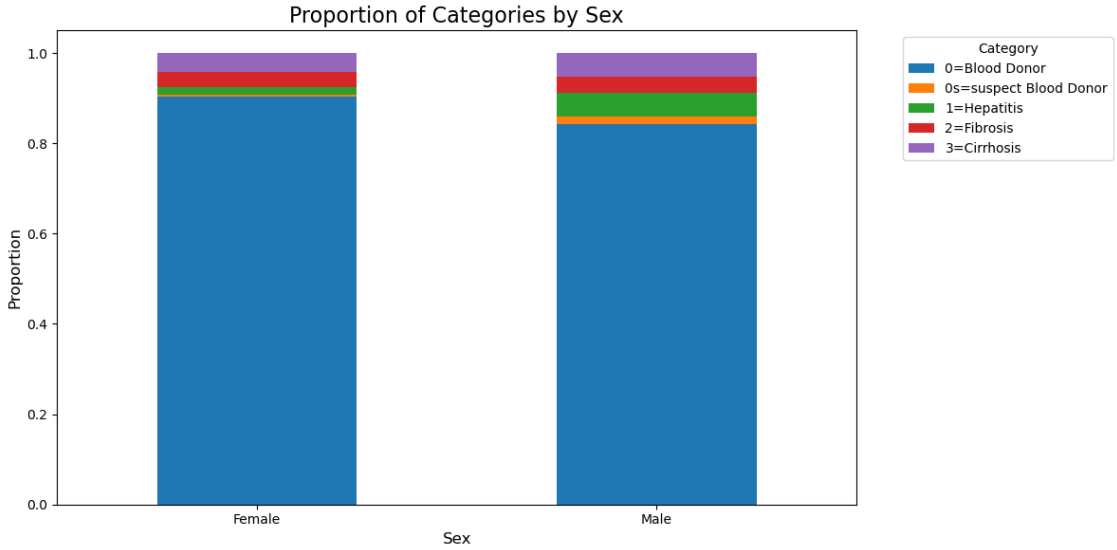
[64]:
```
count_matrix = df.groupby(['Sex', 'Category']).size().unstack()
print(count_matrix)

count_matrix_norm = count_matrix.div(count_matrix.sum(axis=1),axis=0)
print(count_matrix_norm)
```

```
Category  0=Blood Donor  0s=suspect Blood Donor  1=Hepatitis  2=Fibrosis  \
Sex
f                   215                       1            4           8
m                   318                       6           20          13

Category  3=Cirrhosis
Sex
f                  10
m                  20
Category  0=Blood Donor  0s=suspect Blood Donor  1=Hepatitis  2=Fibrosis  \
Sex
f              0.903361                0.004202     0.016807    0.033613
```

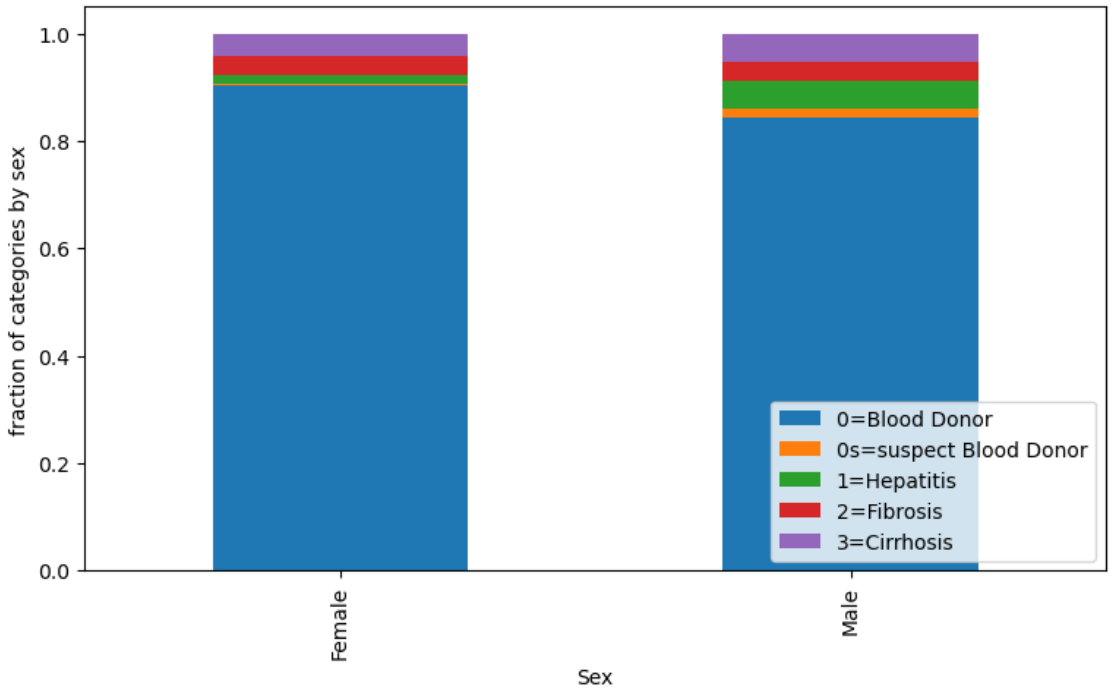
```
m               0.843501                0.015915      0.053050      0.034483
```

```
Category  3=Cirrhosis
Sex
f            0.042017
m            0.053050
```

[65]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Map 'm' and 'f' to 'male' and 'female'
df['Sex'] = df['Sex'].map({'m': 'Male', 'f': 'Female'})

# Create a cross-tabulation of Category by Sex, normalized by columns
count_matrix_norm = pd.crosstab(df['Sex'], df['Category'], normalize='index')

# Plot the stacked bar plot
ax = count_matrix_norm.plot(kind='bar', stacked=True, figsize=(12, 6),␣
 ↪color=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd'])

plt.title('Proportion of Categories by Sex', fontsize=16)
plt.xlabel('Sex', fontsize=12)
plt.ylabel('Proportion', fontsize=12)
plt.legend(title='Category', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xticks(rotation=0)

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

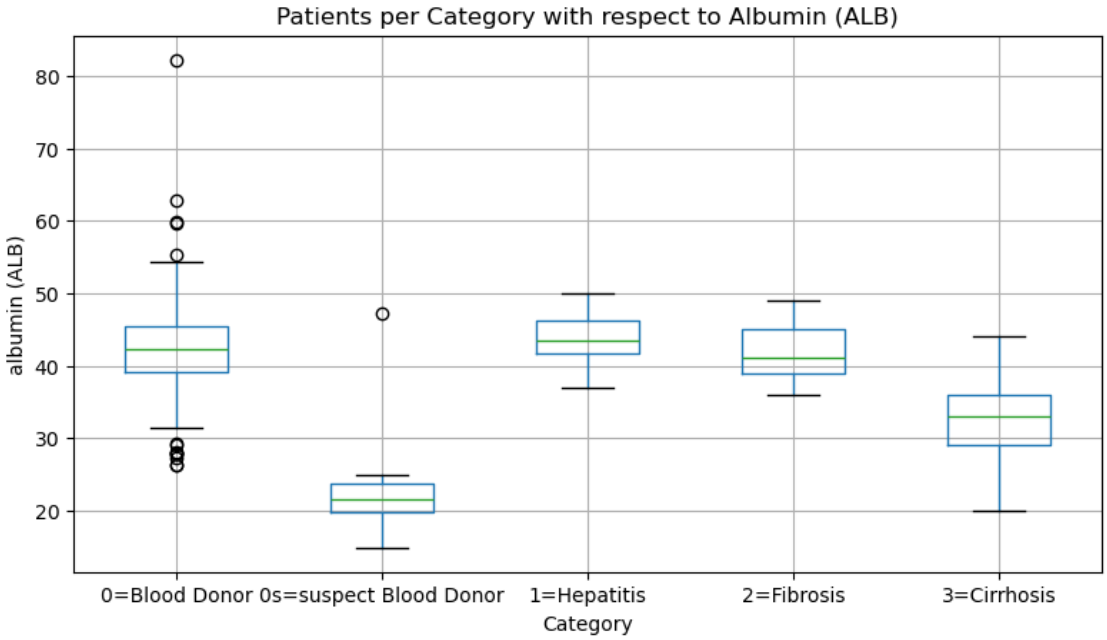


[66]:
```
count_matrix_norm.plot(kind='bar', stacked=True,figsize=(9,5))
plt.ylabel('fraction of categories by sex')
plt.legend(loc=4)
plt.show()
```

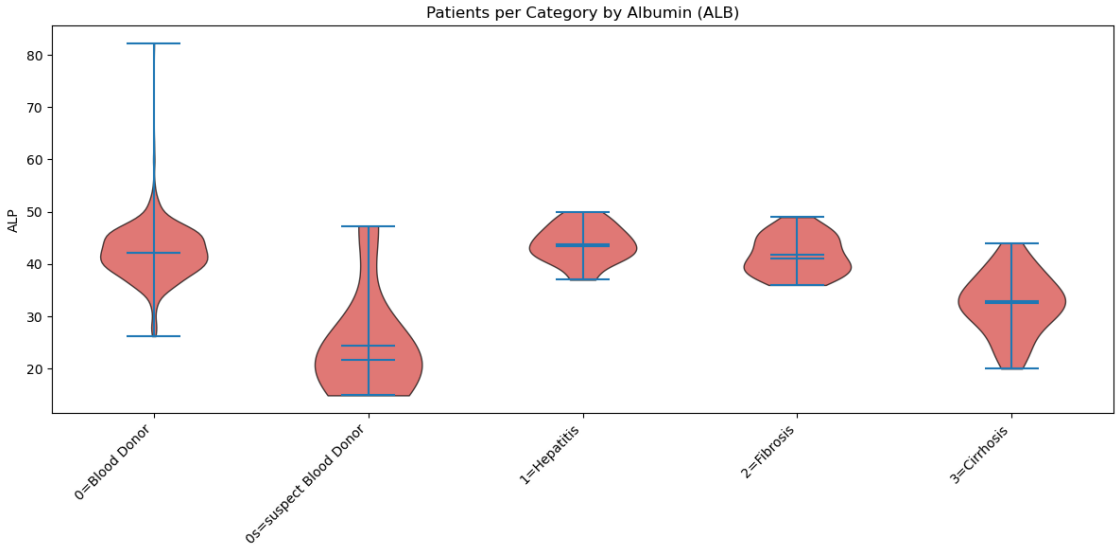


[67]:
```
df[['Category','ALB']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category with respect to Albumin (ALB)')
plt.xlabel('Category')
plt.ylabel('albumin (ALB)')
plt.show()
print('This box plot groups the albumin (ALB) by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['ALB'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['ALB'].dropna().values,
    df[df['Category'] == '1=Hepatitis']['ALB'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['ALB'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['ALB'].dropna().values
]
```

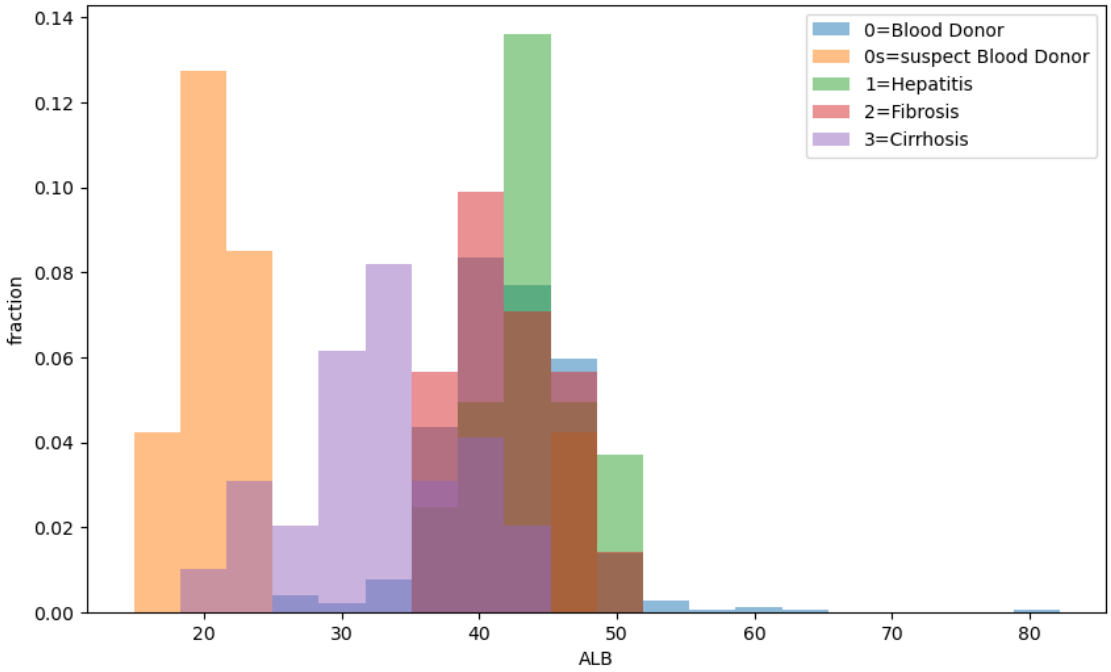
```
# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]

plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('ALP')
plt.title('Patients per Category by Albumin (ALB)')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the ALB by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
bin_range = (df['ALB'].min(),df['ALB'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['ALB'],alpha=0.
 ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
plt.xlabel('ALB')
plt.show()
print('This category-specific histogram groups the ALB by category.')
```



This box plot groups the albumin (ALB) by category.



This violin plot groups the ALB by category.

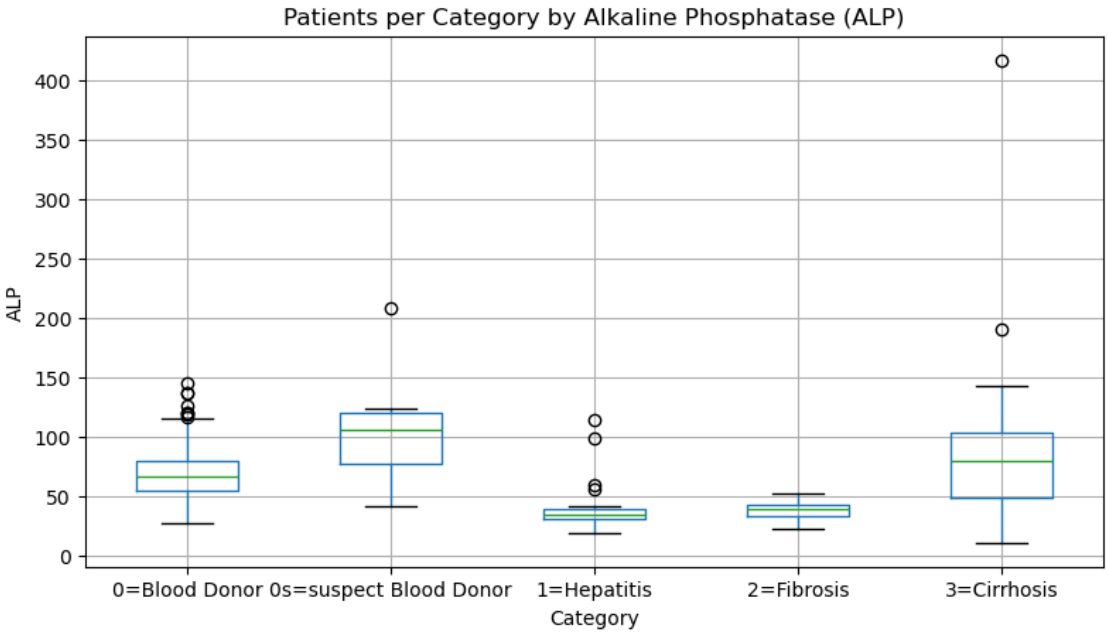


This category-specific histogram groups the ALB by category.

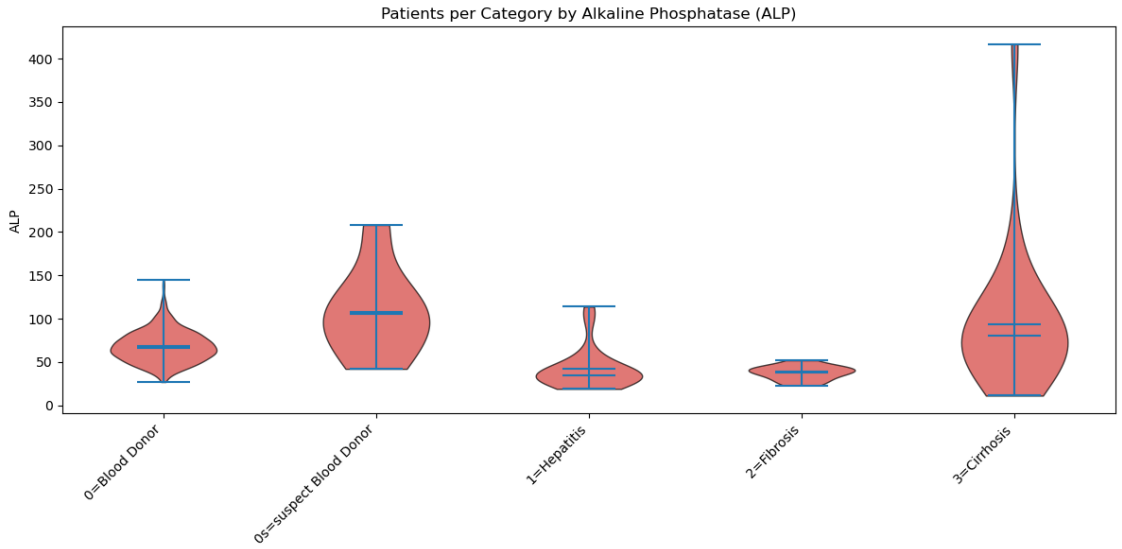
[68]:
```
df[['Category','ALP']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by Alkaline Phosphatase (ALP)')
plt.xlabel('Category')
plt.ylabel('ALP')
plt.show()
print('This box plot groups the ALP by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['ALP'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['ALP'].dropna().values,
    df[df['Category'] == '1=Hepatitis']['ALP'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['ALP'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['ALP'].dropna().values
]

# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]
```

```
plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('ALP')
plt.title('Patients per Category by Alkaline Phosphatase (ALP)')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the ALP by category.')


# dataset = [df[df['Category']=='0=Blood Donor']['ALP'].values,
#            df[df['Category']=='0s=suspect Blood Donor']['ALP'].values,
#            df[df['Category']=='1=Hepatitis']['ALP'].values,
#            df[df['Category']=='2=Fibrosis']['ALP'].values,
#            df[df['Category']=='3=Cirrhosis']['ALP'].values]

# plt.figure(figsize=(10,6))

# plt.violinplot(dataset = dataset, showmeans=True, showmedians=True)
# plt.xticks([1,2,3,4,5],['0=Blood Donor','0s=suspect Blood␣
 ↪Donor','1=Hepatitis',' 2=Fibrosis',' 3=Cirrhosis'])
# plt.ylabel('ALP')
# plt.title('Patients per Category by Alkaline Phosphatase (ALP)')
# plt.show()
# print('This violin plot groups the ALP by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
```

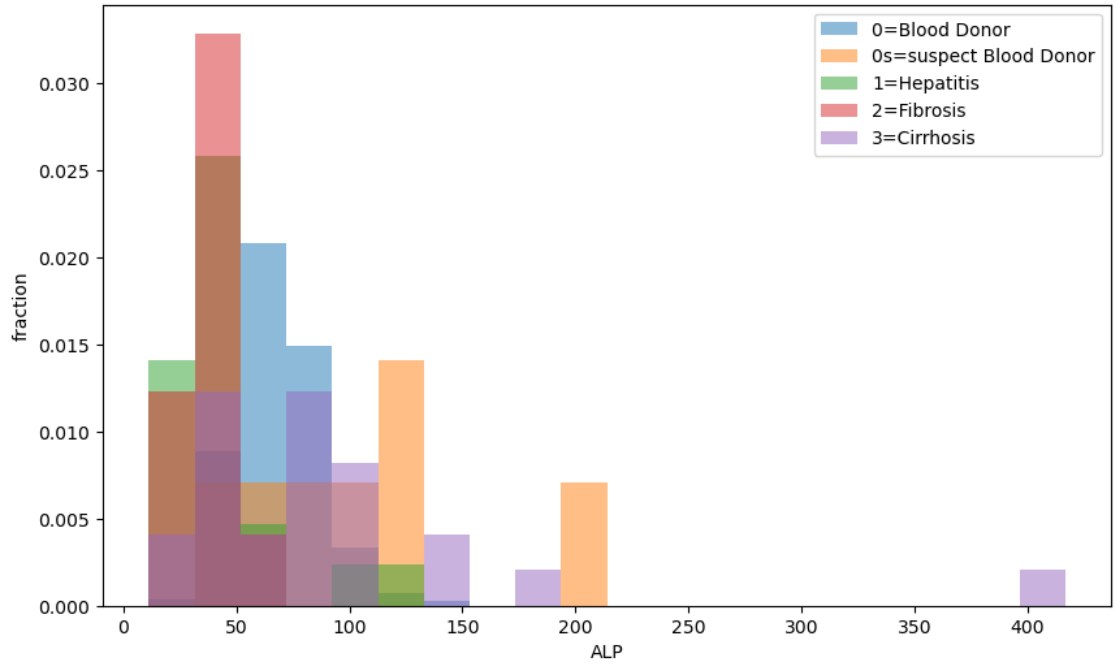
```
bin_range = (df['ALP'].min(),df['ALP'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['ALP'],alpha=0.
  ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
plt.xlabel('ALP')
plt.show()
print('This category-specific histogram groups the ALP by category.')
```



This box plot groups the ALP by category.



This violin plot groups the ALP by category.

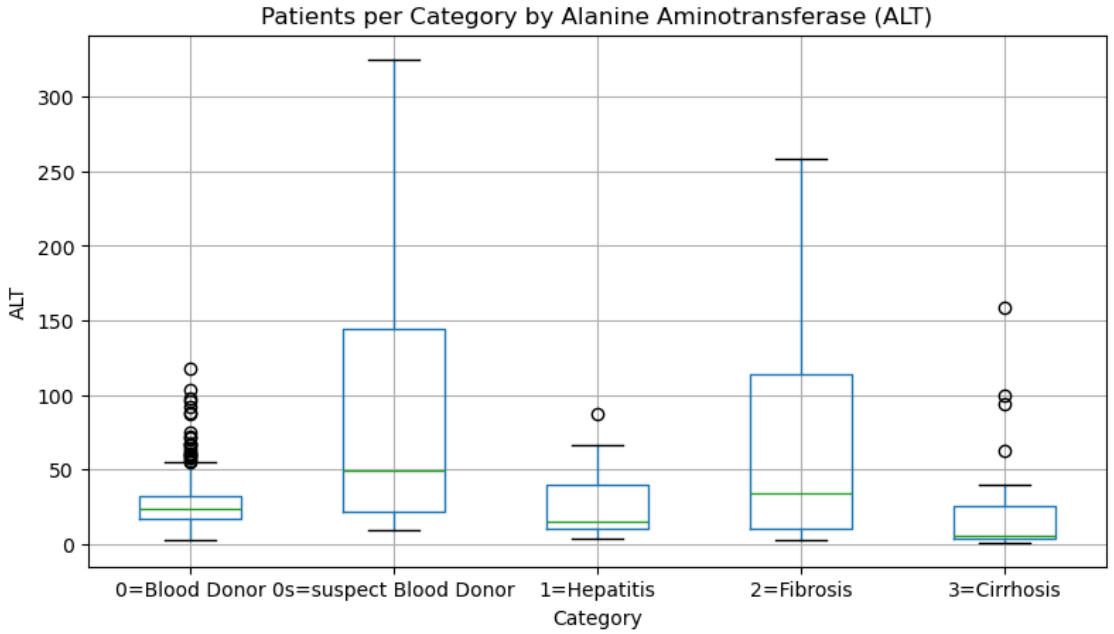


This category-specific histogram groups the ALP by category.

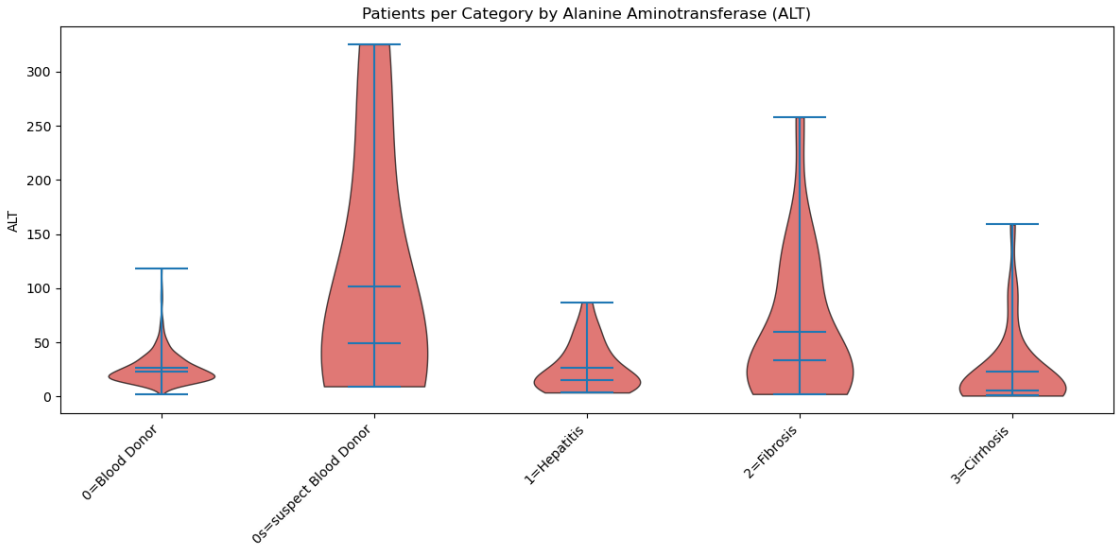
[69]:
```
df[['Category','ALT']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by Alanine Aminotransferase (ALT)')
```

```
plt.xlabel('Category')
plt.ylabel('ALT')
plt.show()
print('This box plot groups the ALT by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['ALT'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['ALT'].dropna().values,
    df[df['Category'] == '1=Hepatitis']['ALT'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['ALT'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['ALT'].dropna().values
]

# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]

plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('ALT')
plt.title('Patients per Category by Alanine Aminotransferase (ALT)')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the ALT by category.')
```

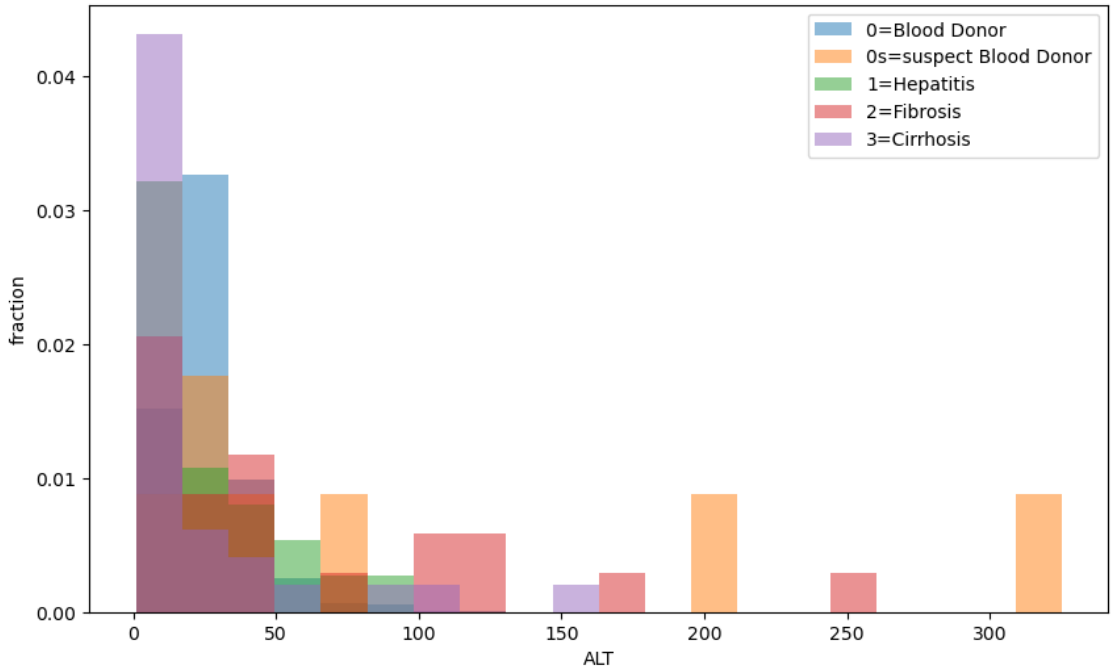
```
# dataset = [df[df['Category']=='0=Blood Donor']['ALT'].values,
#            df[df['Category']=='0s=suspect Blood Donor']['ALT'].values,
#            df[df['Category']=='1=Hepatitis']['ALT'].values,
#            df[df['Category']=='2=Fibrosis']['ALT'].values,
#            df[df['Category']=='3=Cirrhosis']['ALT'].values]

# plt.figure(figsize=(10,6))

# plt.violinplot(dataset = dataset, showmeans=True, showmedians=True)
# plt.xticks([1,2,3,4,5],['0=Blood Donor','0s=suspect Blood␣
 ↪Donor','1=Hepatitis',' 2=Fibrosis',' 3=Cirrhosis'])
# plt.ylabel('ALT')
# plt.title('Patients per Category by Alanine Aminotransferase (ALT)')
# plt.show()
# print('This violin plot groups the ALT by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
bin_range = (df['ALT'].min(),df['ALT'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['ALT'],alpha=0.
 ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
plt.xlabel('ALT')
plt.show()
print('This category-specific histogram groups the ALT by category.')
```



This box plot groups the ALT by category.



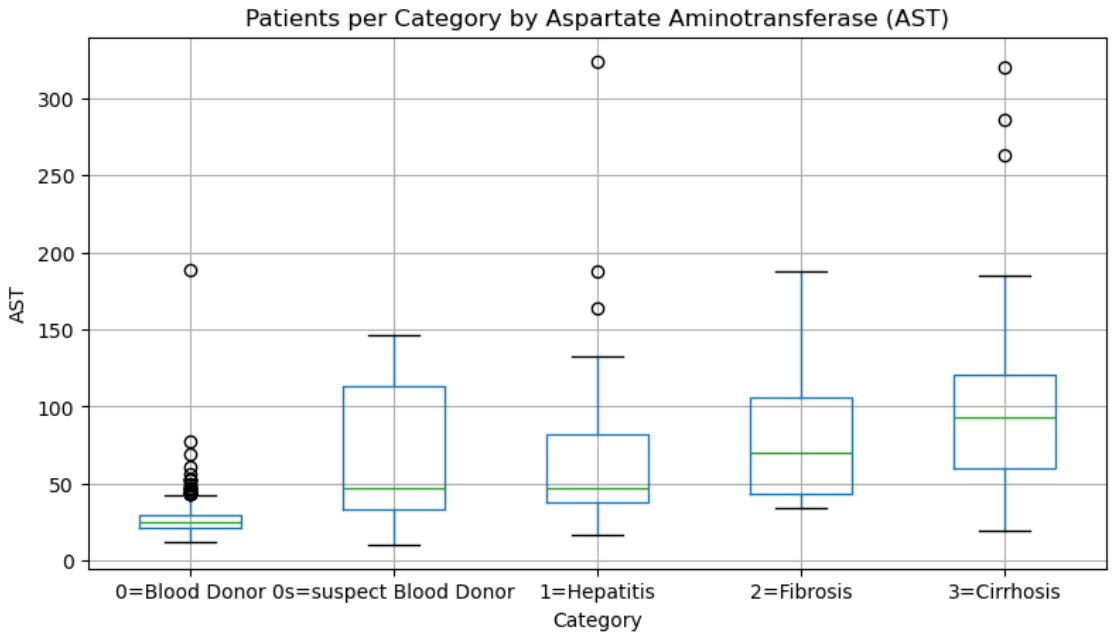
This violin plot groups the ALT by category.



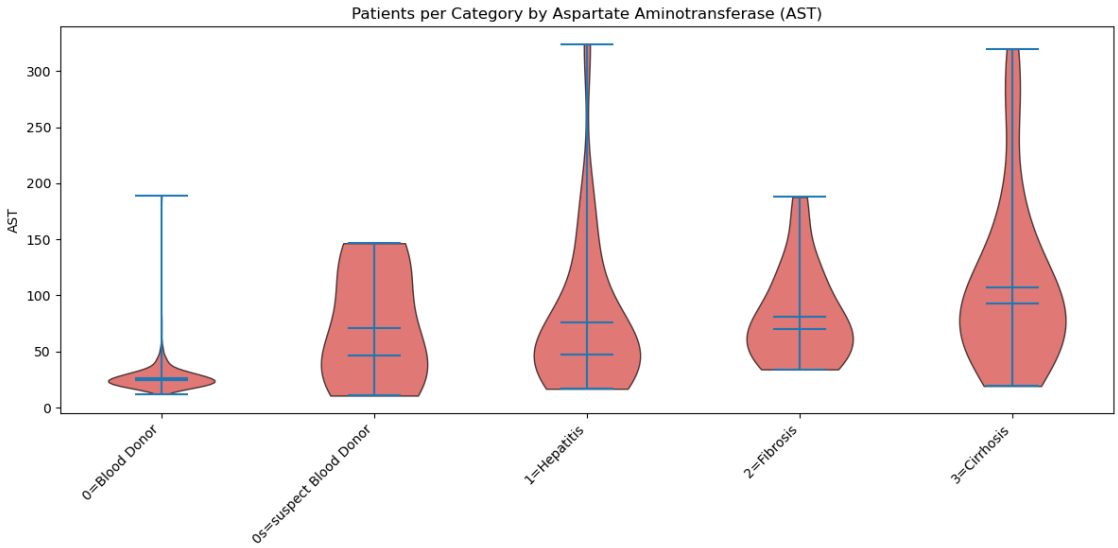
This category-specific histogram groups the ALT by category.

[70]:
```
df[['Category','AST']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by Aspartate Aminotransferase (AST)')
plt.xlabel('Category')
plt.ylabel('AST')
plt.show()
print('This box plot groups the AST by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['AST'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['AST'].dropna().values,
    df[df['Category'] == '1=Hepatitis']['AST'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['AST'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['AST'].dropna().values
]

# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]
```

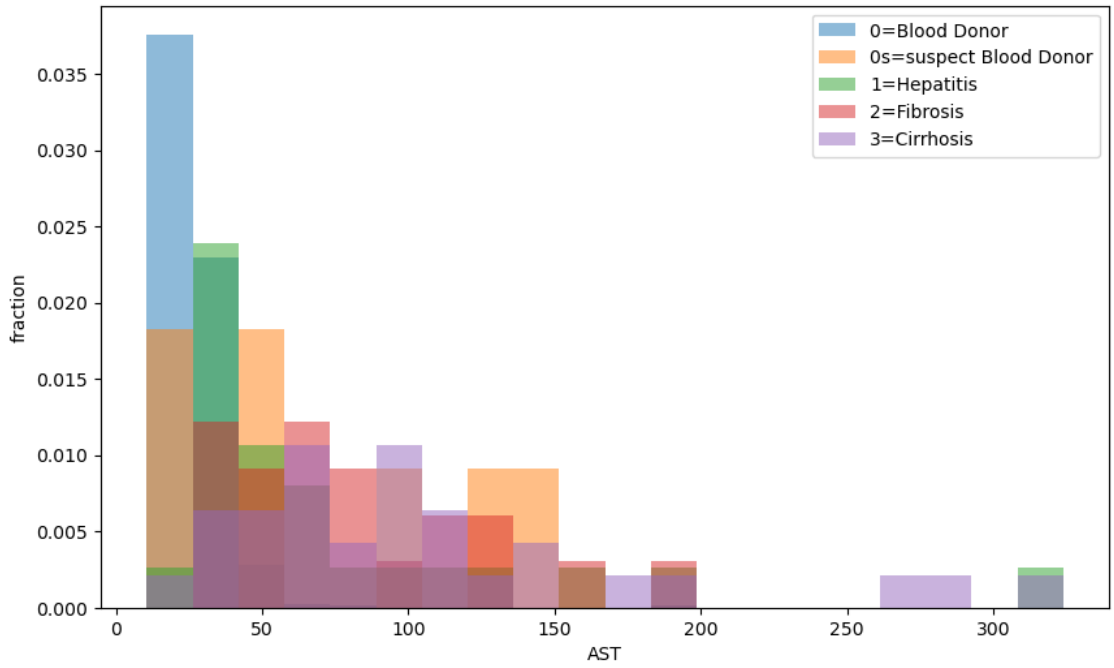
```
plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('AST')
plt.title('Patients per Category by Aspartate Aminotransferase (AST)')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the AST by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
bin_range = (df['AST'].min(),df['AST'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['AST'],alpha=0.
 ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
plt.xlabel('AST')
plt.show()
print('This category-specific histogram groups the AST by category.')
```



This box plot groups the AST by category.



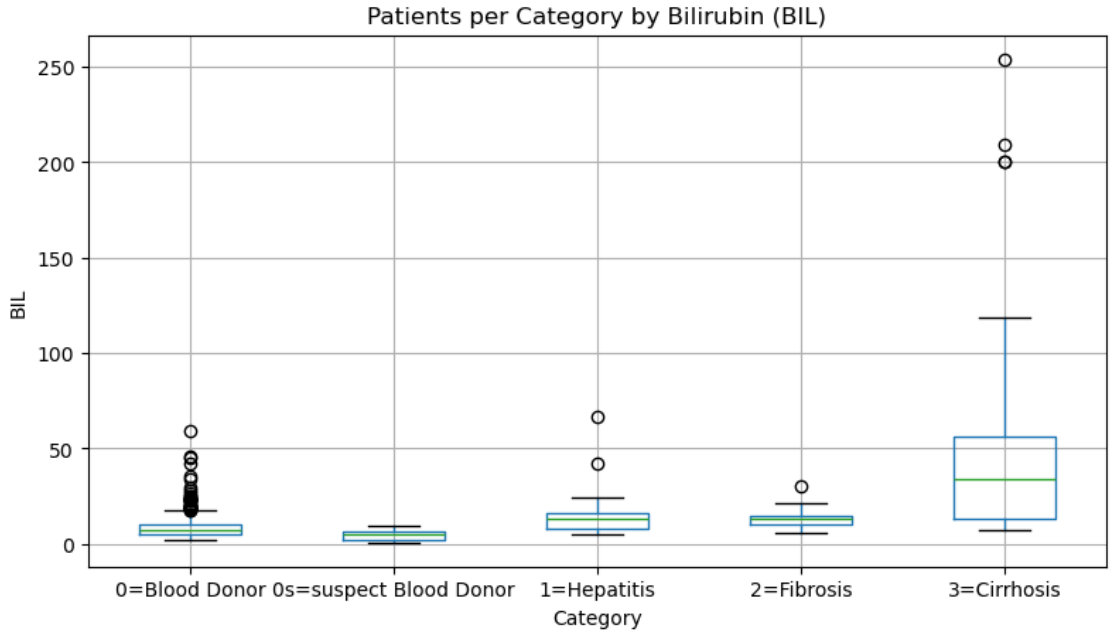
This violin plot groups the AST by category.



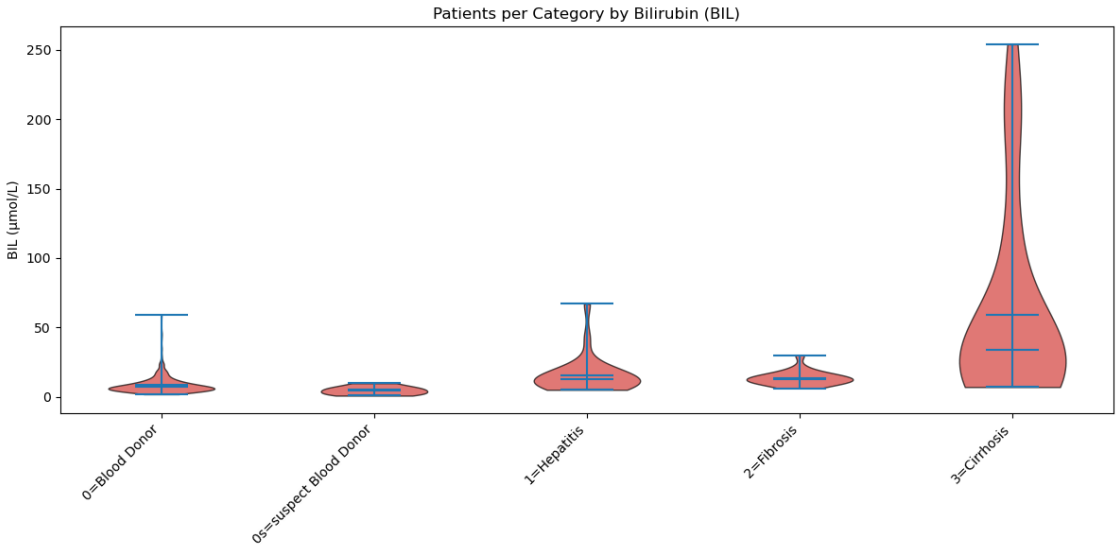
This category-specific histogram groups the AST by category.

[71]:
```
df[['Category','BIL']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by Bilirubin (BIL)')
plt.xlabel('Category')
plt.ylabel('BIL')
plt.show()
print('This box plot groups the BIL by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['BIL'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['BIL'].dropna().values,
    df[df['Category'] == '1=Hepatitis']['BIL'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['BIL'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['BIL'].dropna().values
]

# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]
```

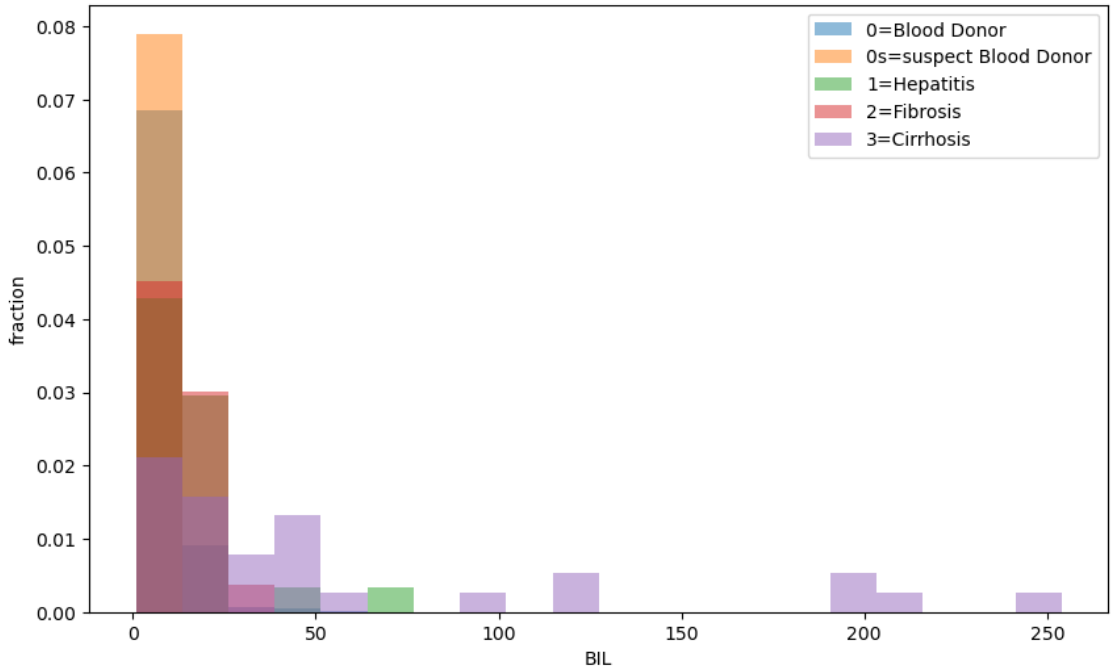
```
plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('BIL ( mol/L)')
plt.title('Patients per Category by Bilirubin (BIL)')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the BIL by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
bin_range = (df['BIL'].min(),df['BIL'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['BIL'],alpha=0.
 ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
plt.xlabel('BIL')
plt.show()
print('This category-specific histogram groups the BIL by category.')
```



This box plot groups the BIL by category.



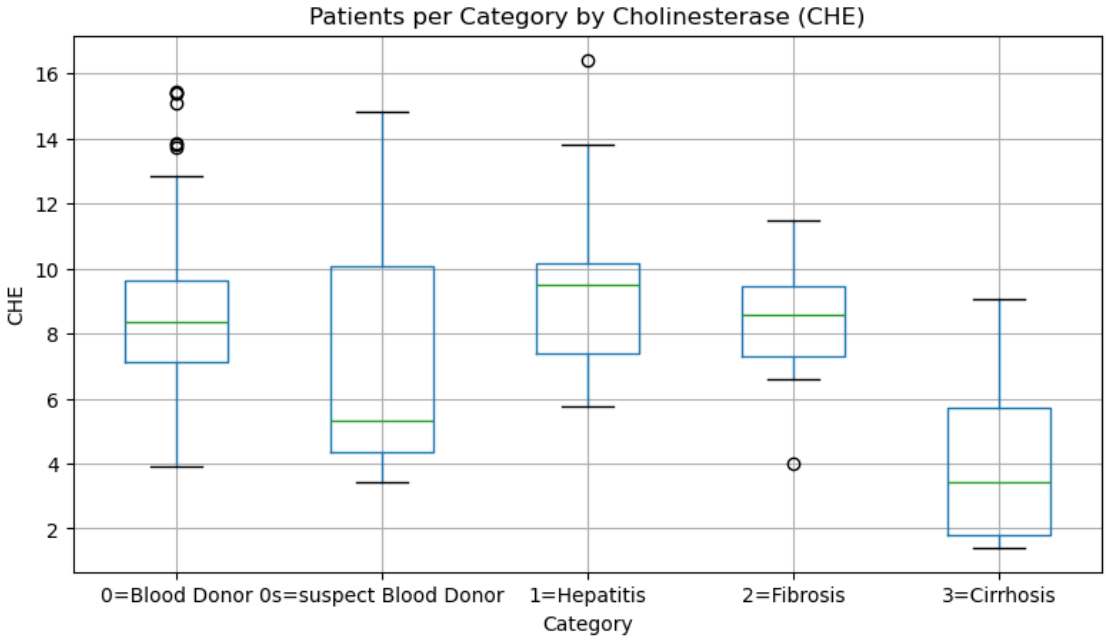
This violin plot groups the BIL by category.



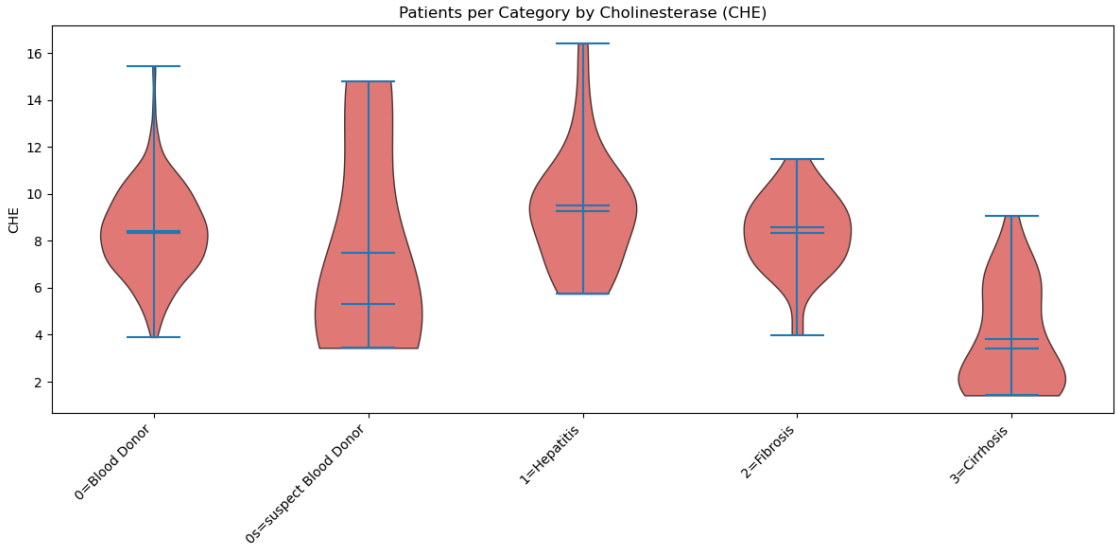
This category-specific histogram groups the BIL by category.

[72]:
```
df[['Category','CHE']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by Cholinesterase (CHE)')
plt.xlabel('Category')
plt.ylabel('CHE')
plt.show()
print('This box plot groups the CHE by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['CHE'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['CHE'].dropna().values,
    df[df['Category'] == '1=Hepatitis']['CHE'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['CHE'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['CHE'].dropna().values
]

# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]
```

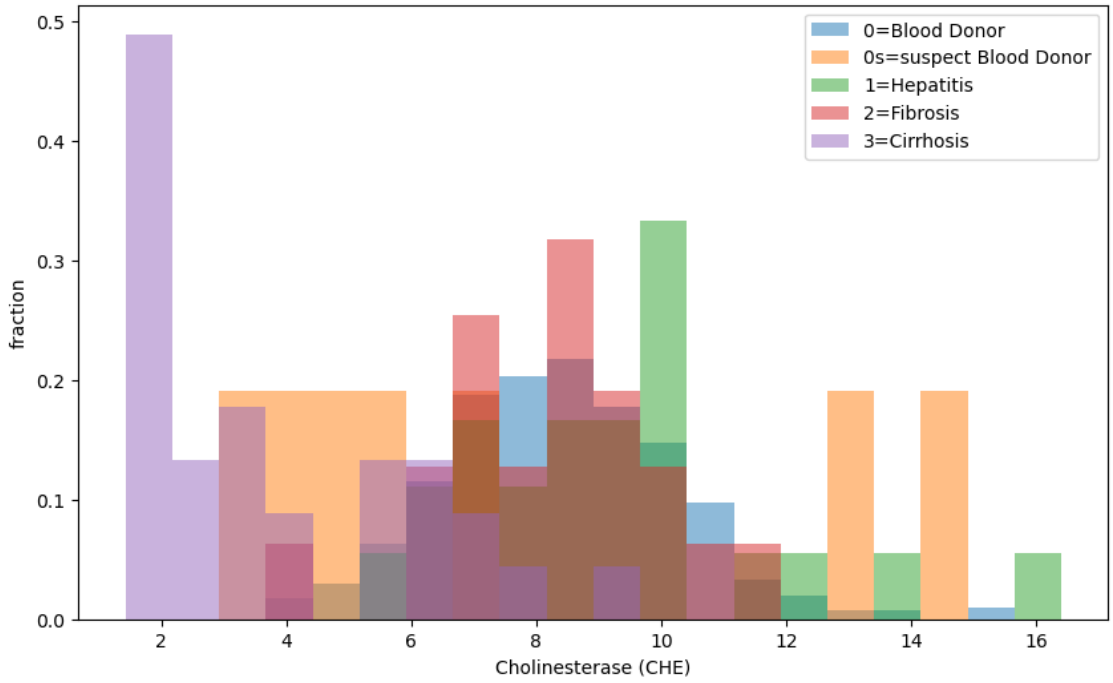
```
plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('CHE')
plt.title('Patients per Category by Cholinesterase (CHE)')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the CHE by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
bin_range = (df['CHE'].min(),df['CHE'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['CHE'],alpha=0.
 ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
plt.xlabel('Cholinesterase (CHE)')
plt.show()
print('This stacked bar chart groups the CHE by category.')
```



This box plot groups the CHE by category.



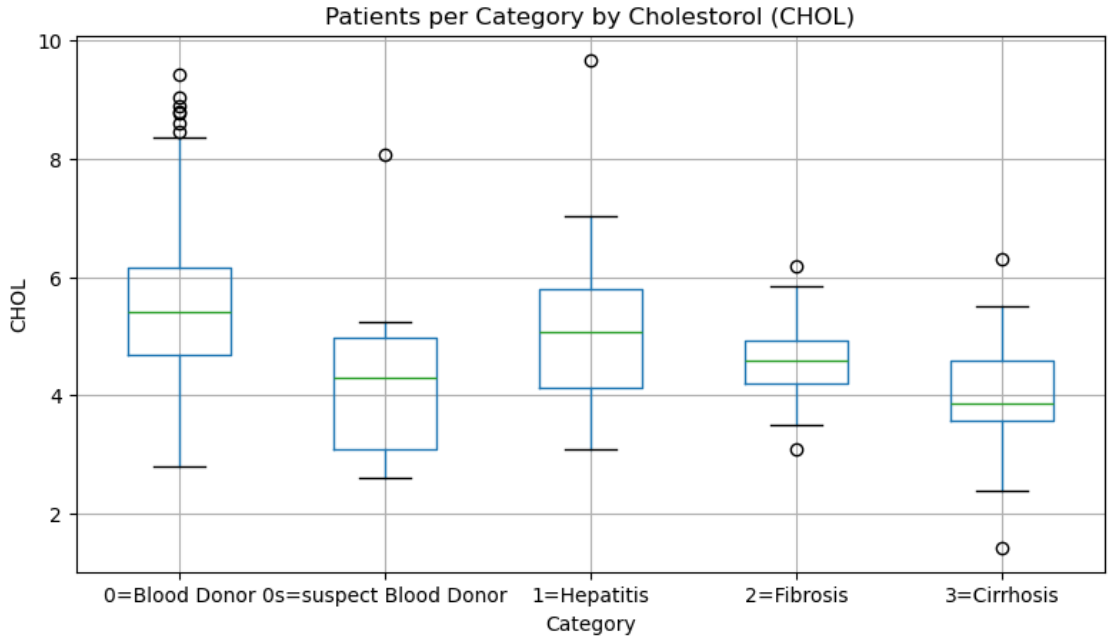
This violin plot groups the CHE by category.



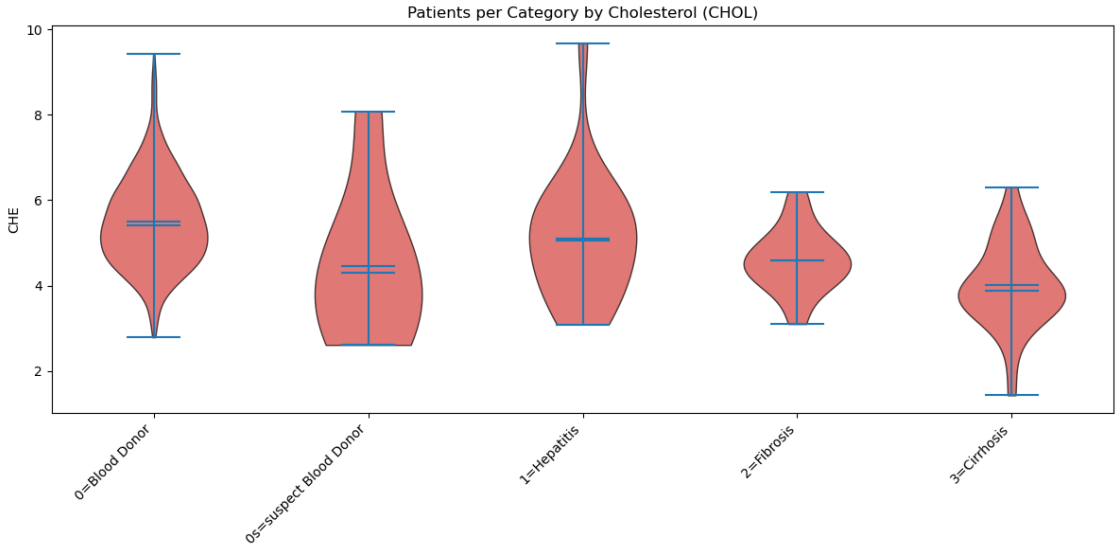
This stacked bar chart groups the CHE by category.

[73]:
```
df[['Category','CHOL']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by Cholestorol (CHOL)')
plt.xlabel('Category')
plt.ylabel('CHOL')
plt.show()
print('This box plot groups the CHOL by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['CHOL'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['CHOL'].dropna().values,
    df[df['Category'] == '1=Hepatitis']['CHOL'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['CHOL'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['CHOL'].dropna().values
]

# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]
```

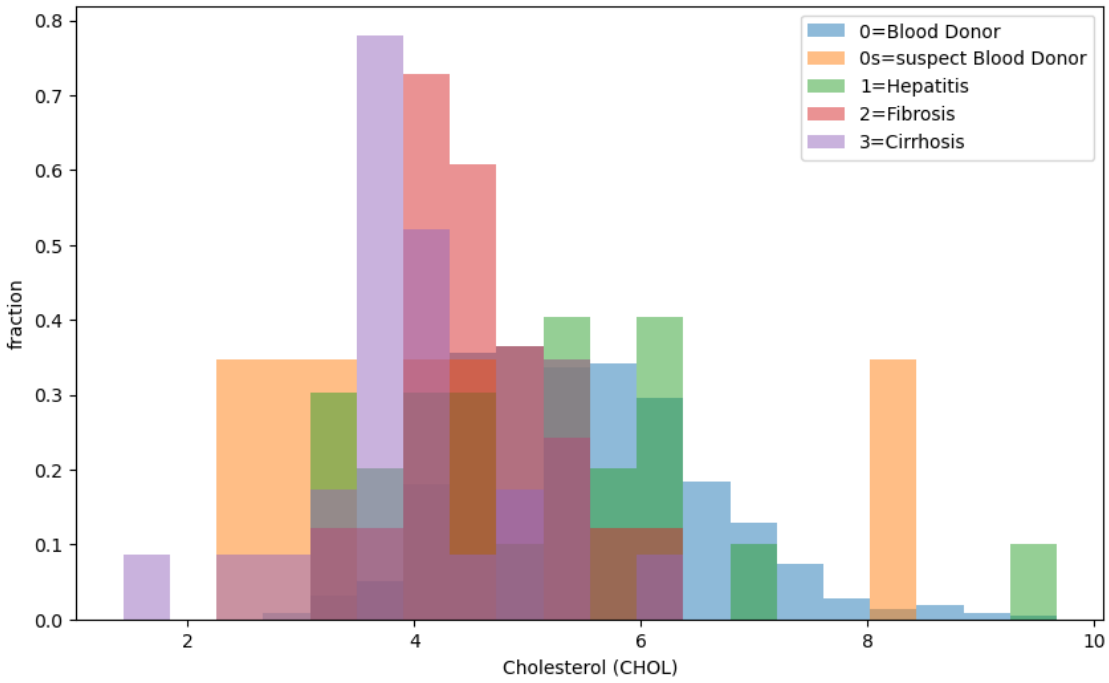
```
plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('CHE')
plt.title('Patients per Category by Cholesterol (CHOL)')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the CHOL by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
bin_range = (df['CHOL'].min(),df['CHOL'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['CHOL'],alpha=0.
 ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
plt.xlabel('Cholesterol (CHOL)')
plt.show()
print('This stacked bar chart groups the CHOL by category.')
```



This box plot groups the CHOL by category.



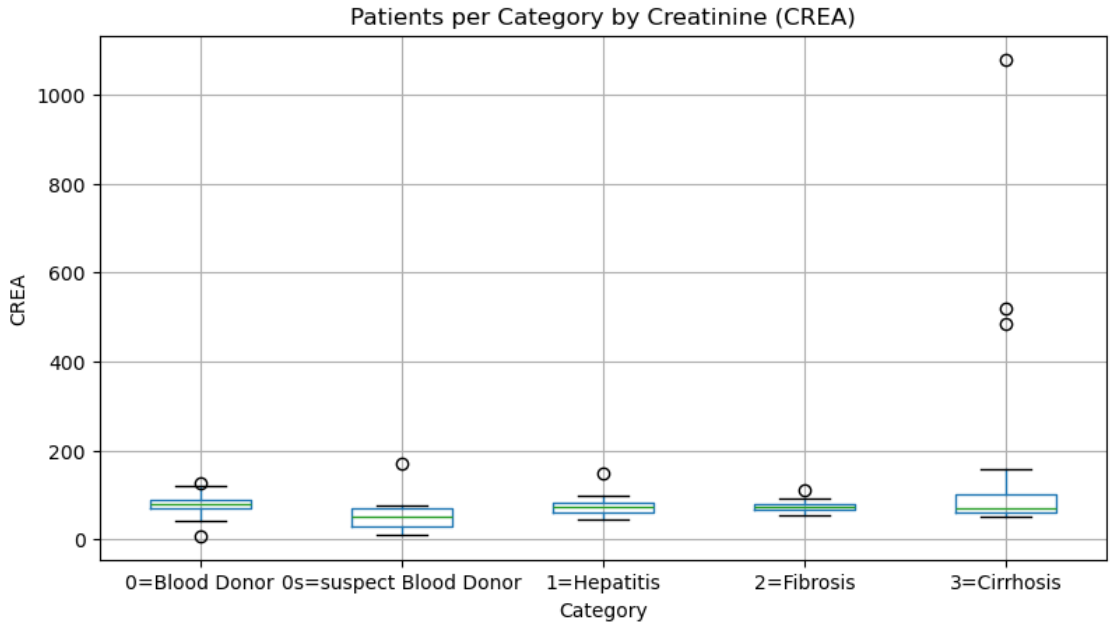
This violin plot groups the CHOL by category.



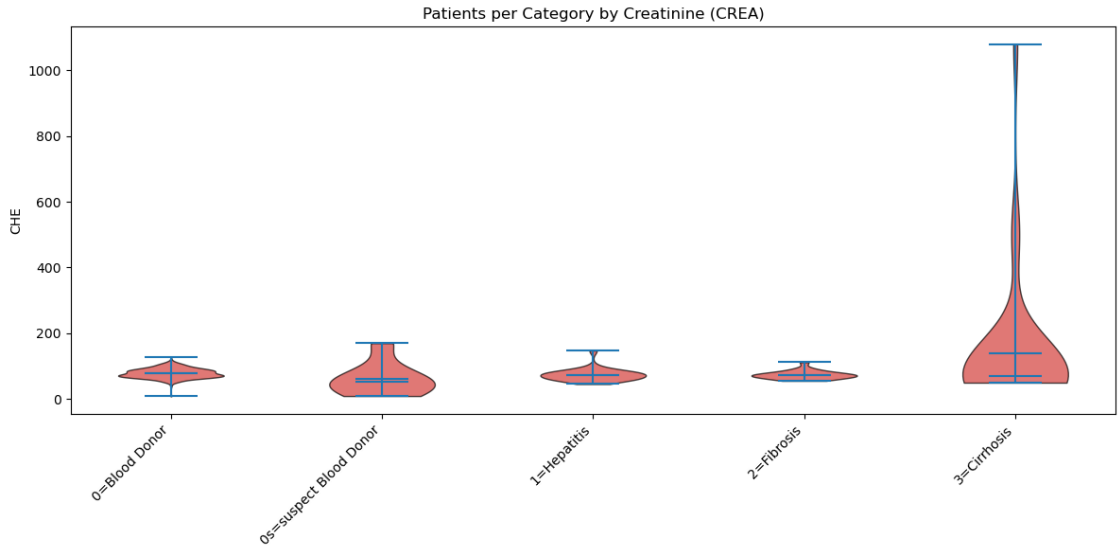
This stacked bar chart groups the CHOL by category.

[74]:
```
df[['Category','CREA']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by Creatinine (CREA)')
plt.xlabel('Category')
plt.ylabel('CREA')
plt.show()
print('This box plot groups the CREA by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['CREA'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['CREA'].dropna().values,
    df[df['Category'] == '1=Hepatitis']['CREA'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['CREA'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['CREA'].dropna().values
]

# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]
```

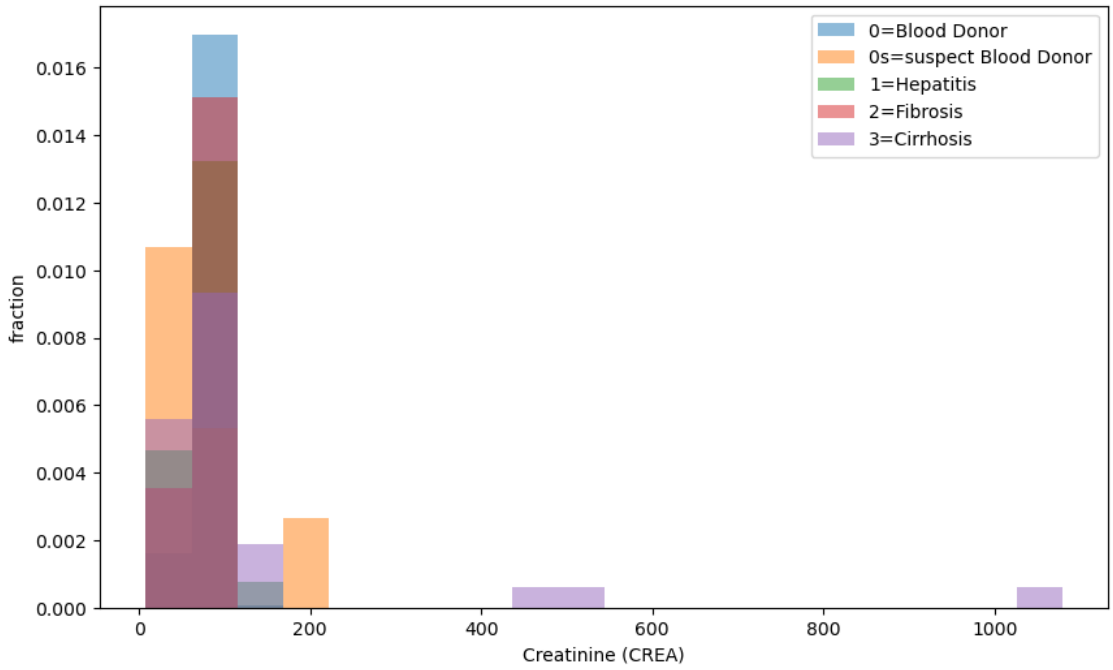
```
plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('CHE')
plt.title('Patients per Category by Creatinine (CREA)')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the CREA by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
bin_range = (df['CREA'].min(),df['CREA'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['CREA'],alpha=0.
 ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
plt.xlabel('Creatinine (CREA)')
plt.show()
print('This stacked bar chart groups the CREA by category.')
```



This box plot groups the CREA by category.



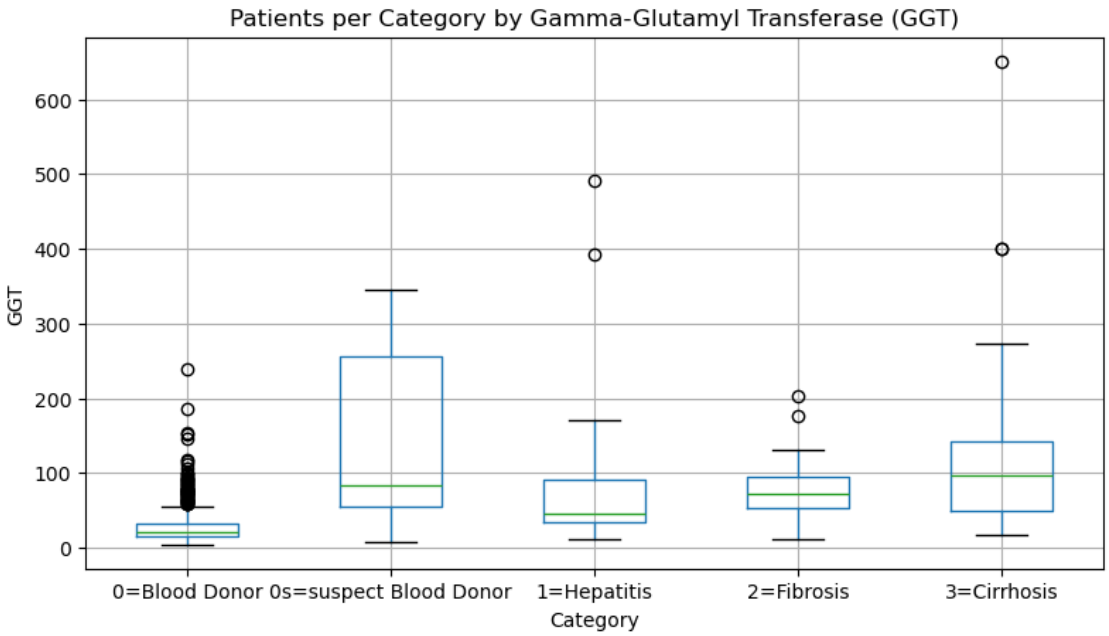
This violin plot groups the CREA by category.



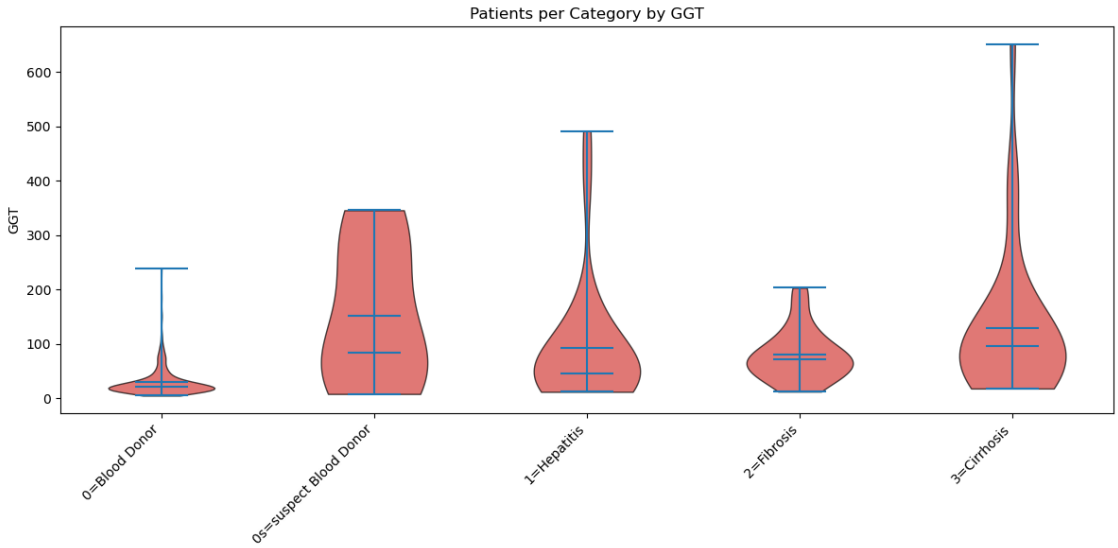
This stacked bar chart groups the CREA by category.

[75]:
```
df[['Category','GGT']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by Gamma-Glutamyl Transferase (GGT)')
plt.xlabel('Category')
plt.ylabel('GGT')
plt.show()
print('This box plot groups the GGT by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['GGT'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['GGT'].dropna().values,
    df[df['Category'] == '1=Hepatitis']['GGT'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['GGT'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['GGT'].dropna().values
]

# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]
```

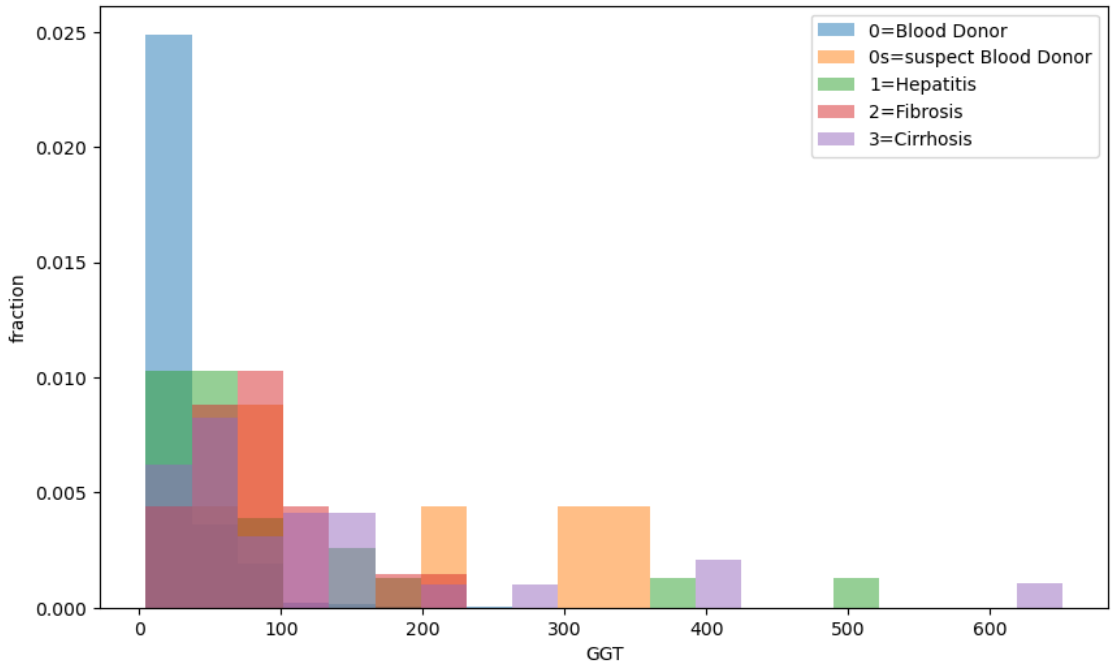
```
plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('GGT')
plt.title('Patients per Category by GGT')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the GGT by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
bin_range = (df['GGT'].min(),df['GGT'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['GGT'],alpha=0.
 ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
plt.xlabel('GGT')
plt.show()
print('This stacked bar chart groups the GGT by category.')
```



This box plot groups the GGT by category.



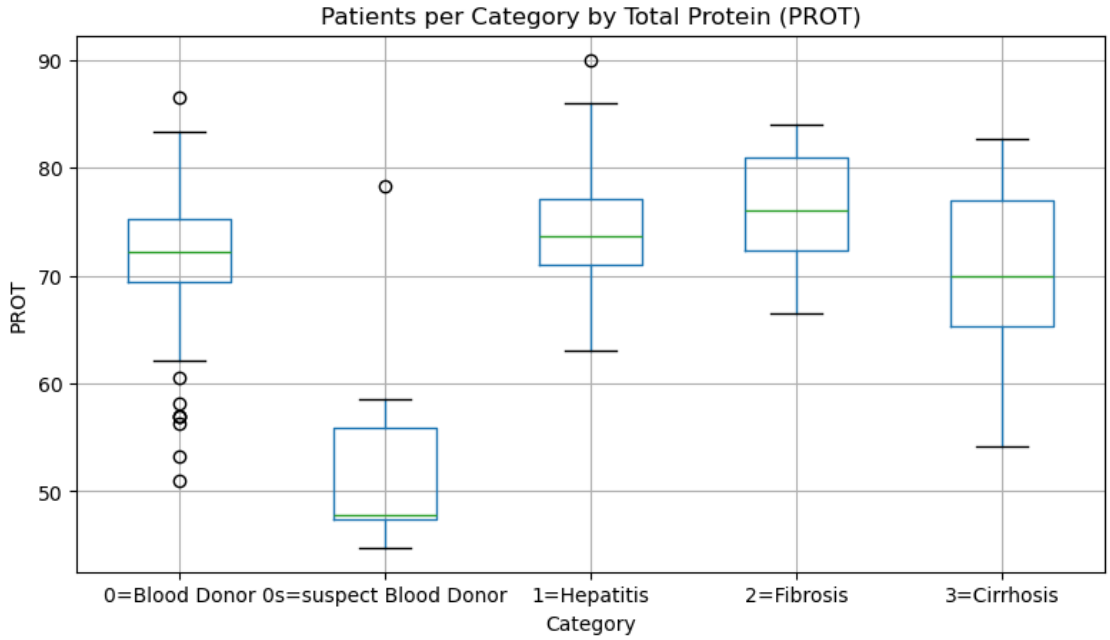
This violin plot groups the GGT by category.



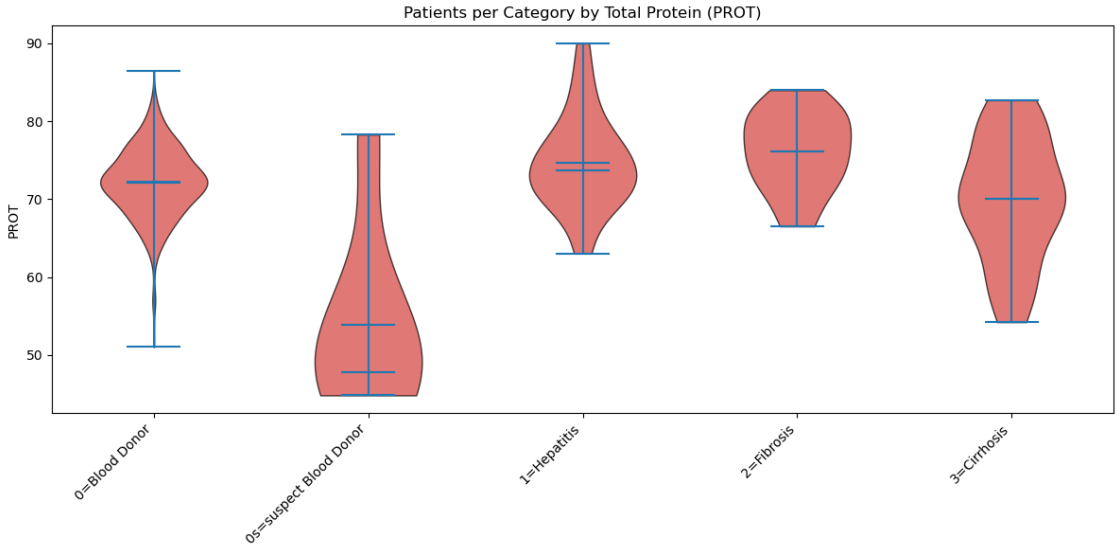
This stacked bar chart groups the GGT by category.

[76]:
```
df[['Category','PROT']].boxplot(by='Category',figsize=(9,5))
plt.suptitle("")
plt.title('Patients per Category by Total Protein (PROT)')
plt.xlabel('Category')
plt.ylabel('PROT')
plt.show()
print('This box plot groups the PROT by category.')

import matplotlib.pyplot as plt
import numpy as np

# Filter out NaN values and create datasets
dataset = [
    df[df['Category'] == '0=Blood Donor']['PROT'].dropna().values,
    df[df['Category'] == '0s=suspect Blood Donor']['PROT'].dropna().values,
    df[df['Category'] == '1=Hepatitis']['PROT'].dropna().values,
    df[df['Category'] == '2=Fibrosis']['PROT'].dropna().values,
    df[df['Category'] == '3=Cirrhosis']['PROT'].dropna().values
]

# Remove empty datasets
dataset = [d for d in dataset if len(d) > 0]
```

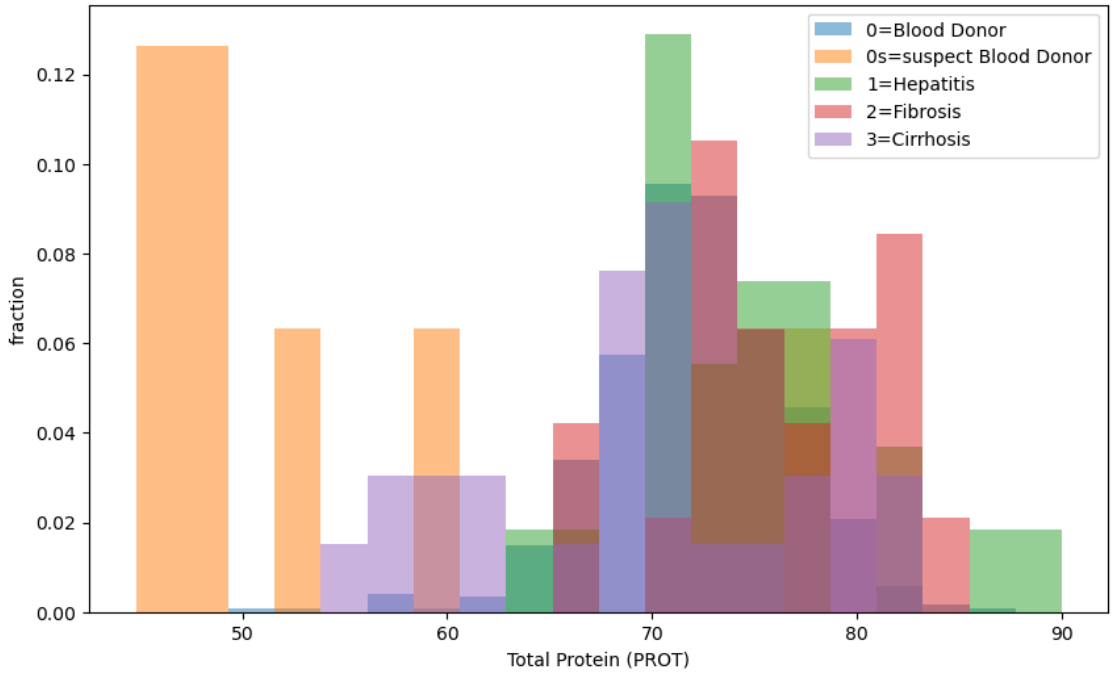
```
plt.figure(figsize=(12, 6))

# Create violin plot
vp = plt.violinplot(dataset=dataset, showmeans=True, showmedians=True)

# Customize violin plot colors
for pc in vp['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

# Set x-axis labels
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
plt.xticks(range(1, len(dataset) + 1), categories, rotation=45, ha='right')

plt.ylabel('PROT')
plt.title('Patients per Category by Total Protein (PROT)')

# Adjust layout to prevent cut-off labels
plt.tight_layout()

plt.show()

print('This violin plot groups the PROT by category.')

import matplotlib
from matplotlib import pylab as plt

categories = df['Category'].unique()
bin_range = (df['PROT'].min(),df['PROT'].max())

plt.figure(figsize=(10,6))

for c in categories:
    plt.hist(df[df['Category']==c]['PROT'],alpha=0.
 ↪5,label=c,range=bin_range,bins=20,density=True)
plt.legend()
plt.ylabel('fraction')
plt.xlabel('Total Protein (PROT)')
plt.show()
print('This stacked bar chart groups the PROT by category.')
```



This box plot groups the PROT by category.



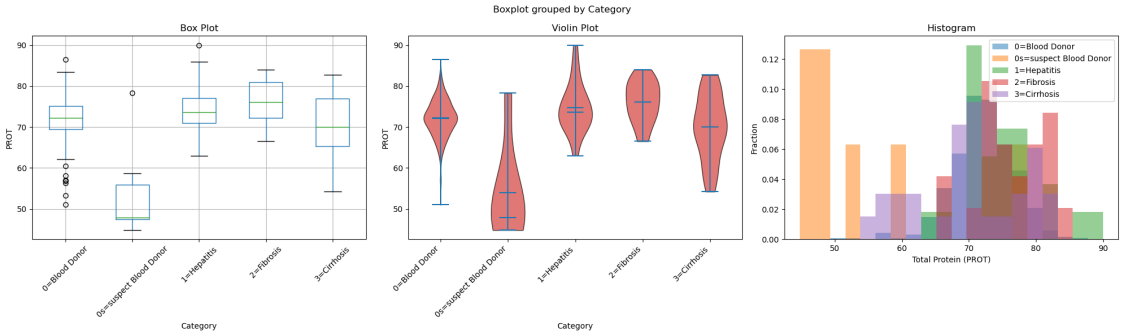
This violin plot groups the PROT by category.



This stacked bar chart groups the PROT by category.

[77]:
```
import matplotlib.pyplot as plt
import numpy as np

# Set up the figure and axes
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20, 6))
fig.suptitle('Patients per Category by Total Protein (PROT)', fontsize=16)

# Box Plot
df.boxplot(column='PROT', by='Category', ax=ax1)
ax1.set_title('Box Plot')
ax1.set_xlabel('Category')
ax1.set_ylabel('PROT')
ax1.tick_params(axis='x', rotation=45)

# Violin Plot
categories = ['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',␣
 ↪'2=Fibrosis', '3=Cirrhosis']
dataset = [df[df['Category'] == cat]['PROT'].dropna().values for cat in␣
 ↪categories]
dataset = [d for d in dataset if len(d) > 0]

vp = ax2.violinplot(dataset=dataset, showmeans=True, showmedians=True)
for pc in vp['bodies']:
```

```
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(0.7)

ax2.set_xticks(range(1, len(dataset) + 1))
ax2.set_xticklabels(categories, rotation=45, ha='right')
ax2.set_title('Violin Plot')
ax2.set_xlabel('Category')
ax2.set_ylabel('PROT')

# Histogram
bin_range = (df['PROT'].min(), df['PROT'].max())
for c in categories:
    ax3.hist(df[df['Category']==c]['PROT'], alpha=0.5, label=c,␣
 ↪range=bin_range, bins=20, density=True)
ax3.legend()
ax3.set_title('Histogram')
ax3.set_xlabel('Total Protein (PROT)')
ax3.set_ylabel('Fraction')

# Adjust layout and display
plt.tight_layout()
plt.show()

print('These plots group the PROT by category.')
```

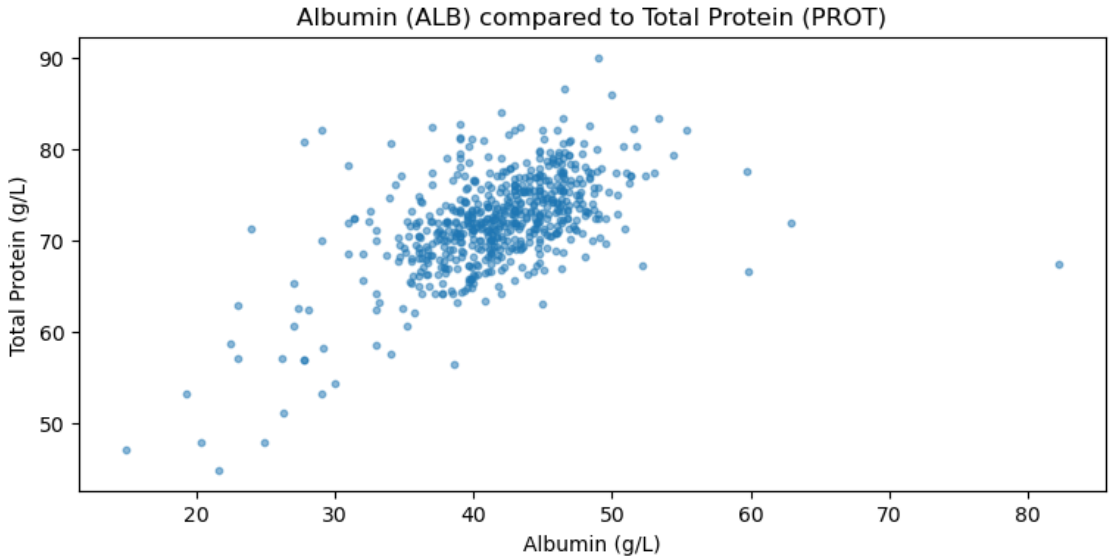



```
These plots group the PROT by category.
```

1.5.4 Column Pair Plots Between Features

[78]:
```
df.plot.scatter('ALB','PROT',figsize=(9,4),alpha=0.5,s=10)

plt.title('Albumin (ALB) compared to Total Protein (PROT)')
plt.xlabel('Albumin (g/L)')
plt.ylabel('Total Protein (g/L)')
```

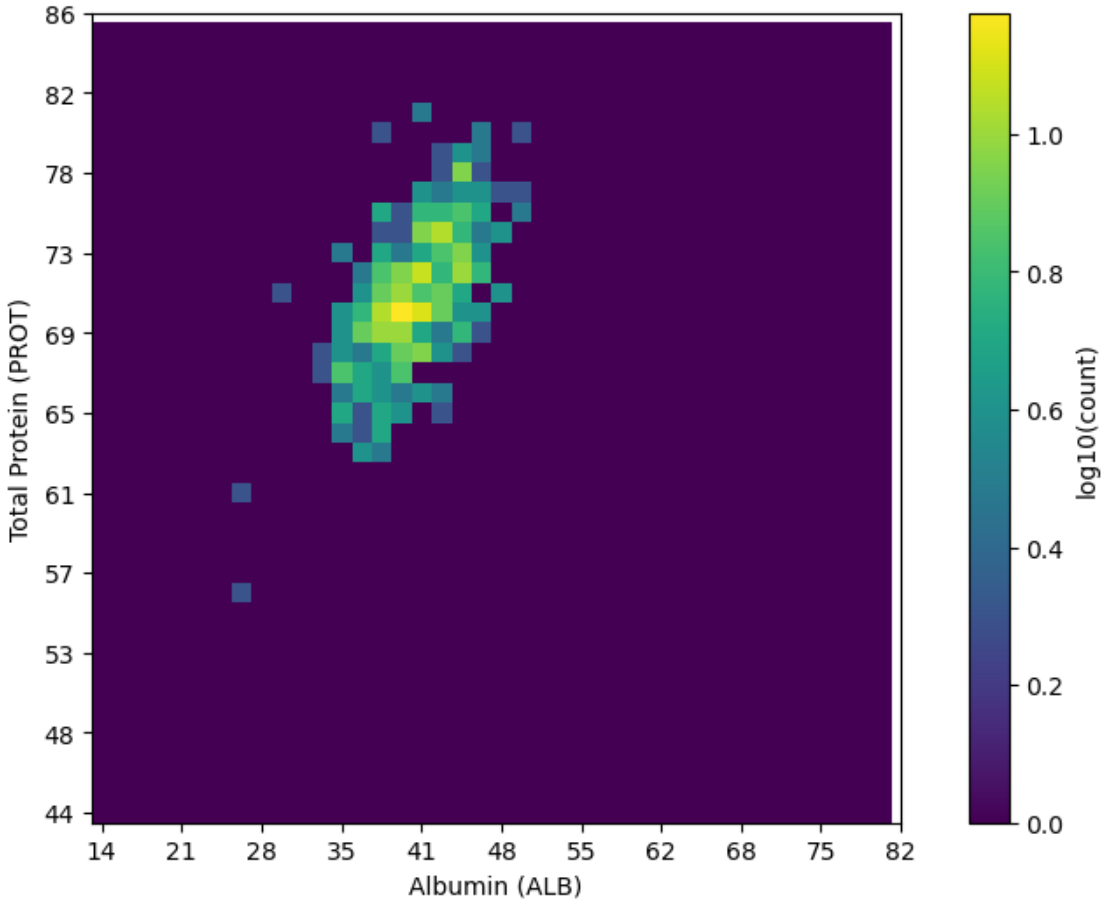
```
plt.show()
print('This scatter plot shows the relationship between albumin and total␣
 ↪protein. Albumin and total protein in blood are positively correlated.')
```



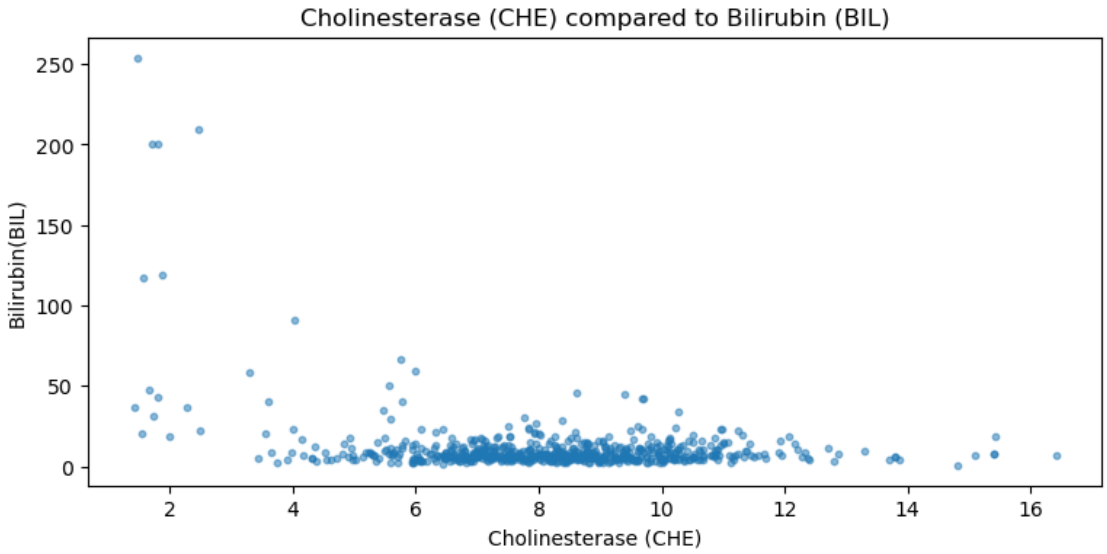
This scatter plot shows the relationship between albumin and total protein. Albumin and total protein in blood are positively correlated.

[79]:
```
df_cleaned = df.dropna()

nbins = 40

heatmap, xedges, yedges = np.histogram2d(df_cleaned['ALB'], df_cleaned['PROT'],␣
 ↪bins=nbins)
extent = [xedges[0], xedges[-1], yedges[0], yedges[-1]]
```

[80]:
```
heatmap[heatmap == 0] = 0.1 # we will use log and log(0) is undefined
plt.figure(figsize=(10,6))

plt.imshow(np.log10(heatmap).T, origin='lower',vmin=0) # use log count
#plt.imshow(heatmap.T, origin='lower',vmin=0) # use log count
plt.xlabel('Albumin (ALB)')
plt.ylabel('Total Protein (PROT)')
plt.xticks(np.arange(nbins+1)[::4],xedges[::4].astype(int))
plt.yticks(np.arange(nbins+1)[::4],yedges[::4].astype(int))
plt.colorbar(label='log10(count)',cmap='coolwarm')
plt.show()
```



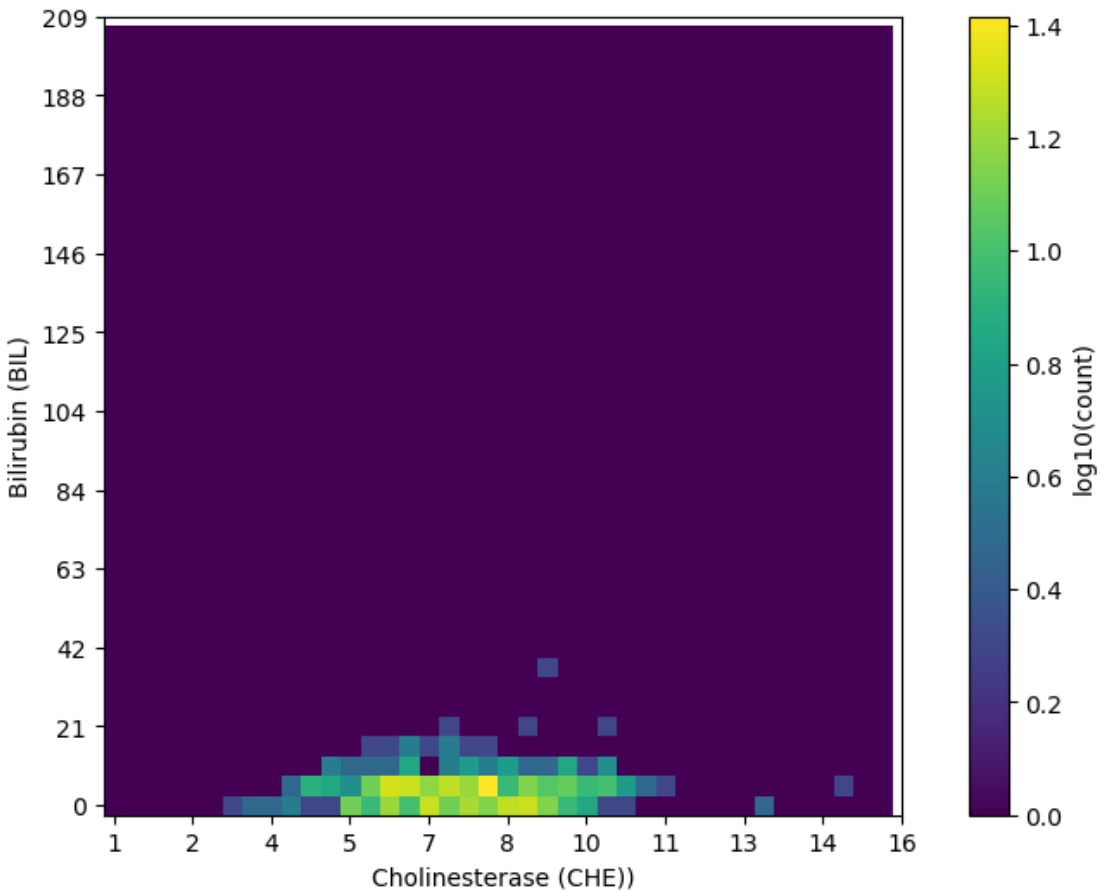
[81]:
```
df.plot.scatter('CHE','BIL',figsize=(9,4),alpha=0.5,s=10)

plt.title('Cholinesterase (CHE) compared to Bilirubin (BIL)')
plt.xlabel('Cholinesterase (CHE)')
plt.ylabel('Bilirubin(BIL)')
plt.show()
print('This scatter plot shows the relationship between cholinesterase and␣
 ↪bilirubin. Cholinesterase and bilirubin in blood are weakly negatively␣
 ↪correlated.')
```



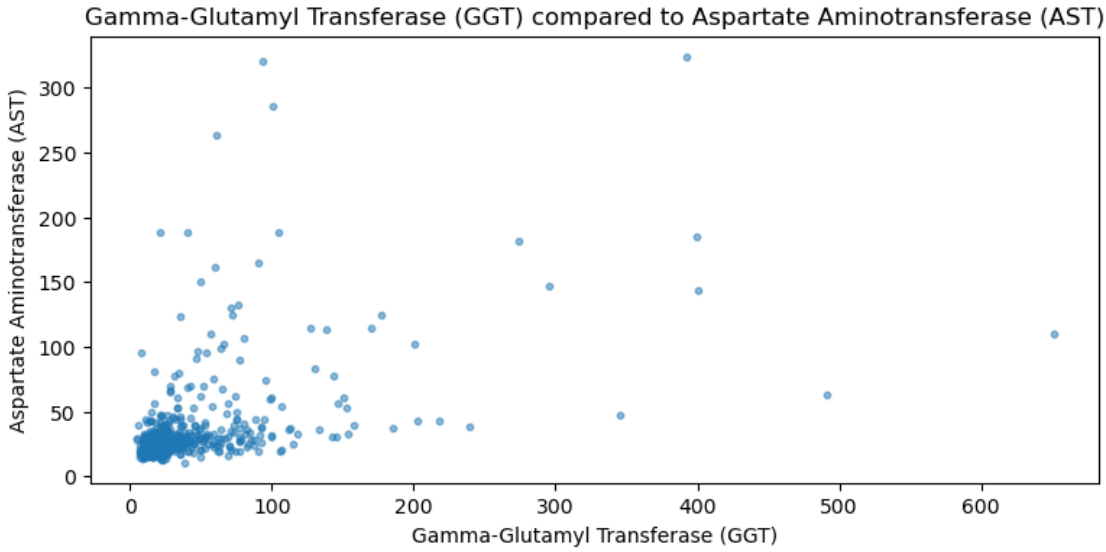
This scatter plot shows the relationship between cholinesterase and bilirubin. Cholinesterase and bilirubin in blood are weakly negatively correlated.

[82]:
```
df_cleaned = df.dropna()

nbins = 40

heatmap, xedges, yedges = np.histogram2d(df_cleaned['CHE'], df_cleaned['BIL'],␣
↪bins=nbins)
extent = [xedges[0], xedges[-1], yedges[0], yedges[-1]]
```

[83]:
```
heatmap[heatmap == 0] = 0.1 # we will use log and log(0) is undefined
plt.figure(figsize=(10,6))

plt.imshow(np.log10(heatmap).T, origin='lower',vmin=0) # use log count
#plt.imshow(heatmap.T, origin='lower',vmin=0) # use log count
plt.xlabel('Cholinesterase (CHE))')
plt.ylabel('Bilirubin (BIL)')
plt.xticks(np.arange(nbins+1)[::4],xedges[::4].astype(int))
plt.yticks(np.arange(nbins+1)[::4],yedges[::4].astype(int))
plt.colorbar(label='log10(count)',cmap='coolwarm')
plt.show()
```



[84]:
```
df.plot.scatter('GGT','AST',figsize=(9,4),alpha=0.5,s=10)

plt.title('Gamma-Glutamyl Transferase (GGT) compared to Aspartate␣
 ↪Aminotransferase (AST)')
plt.xlabel('Gamma-Glutamyl Transferase (GGT)')
plt.ylabel('Aspartate Aminotransferase (AST)')
plt.show()
print('This scatter plot shows the relationship between Gamma-Glutamyl␣
 ↪Transferase and Aspartate Aminotransferase. Gamma-Glutamyl Transferase and␣
 ↪Aspartate Aminotransferase in blood are positively correlated.')
```



This scatter plot shows the relationship between Gamma-Glutamyl Transferase and Aspartate Aminotransferase. Gamma-Glutamyl Transferase and Aspartate Aminotransferase in blood are positively correlated.

1.5.5 Correlation Matrix

[85]:
```
df_numeric = df.drop(columns=['Category', 'Unnamed: 0'])
df_numeric.head()
```

[85]:

	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CREA	GGT	PROT
0	32	Male	38.5	52.5	7.7	22.1	7.5	6.93	3.23	106.0	12.1	69.0
1	32	Male	38.5	70.3	18.0	24.7	3.9	11.17	4.80	74.0	15.6	76.5
2	32	Male	46.9	74.7	36.2	52.6	6.1	8.84	5.20	86.0	33.2	79.3
3	32	Male	43.2	52.0	30.6	22.6	18.9	7.33	4.74	80.0	33.8	75.7
4	32	Male	39.2	74.1	32.6	24.8	9.6	9.15	4.32	76.0	29.9	68.7

[86]:
```
import pandas as pd

# Converting 'Sex' to numeric values
df_numeric['Sex'] = df_numeric['Sex'].map({'m': 1, 'f': 2})
```

[87]:
```
corr_matrix = df_numeric.corr(method='pearson', min_periods=1,␣
 ↪numeric_only=False)
corr_matrix
```

[87]:

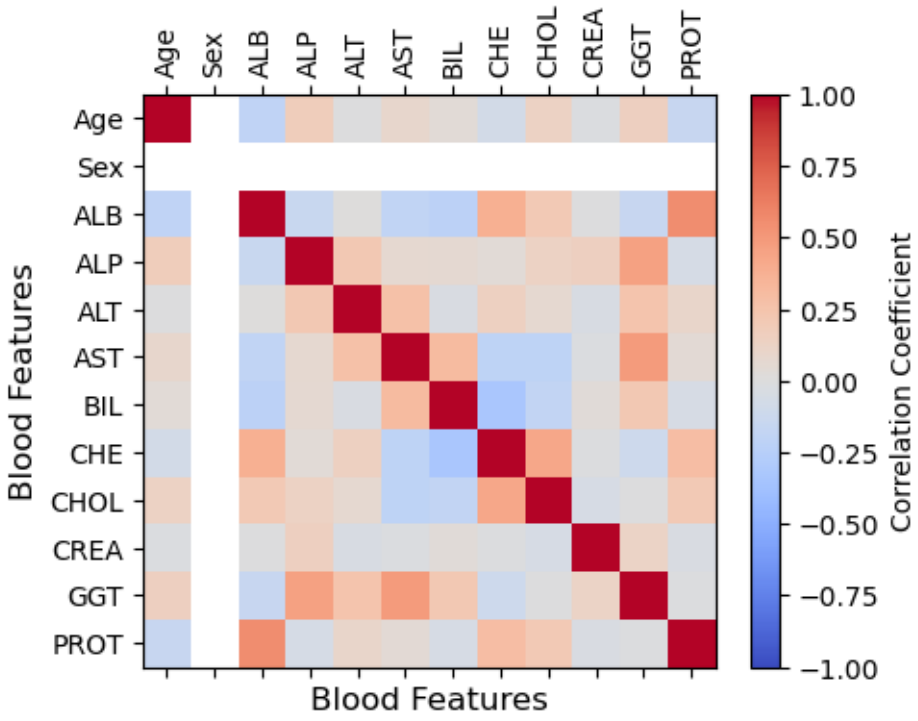
	Age	Sex	ALB	ALP	ALT	AST	BIL	\
Age	1.000000	NaN	-0.197498	0.173340	-0.006021	0.088666	0.032492	
Sex	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
ALB	-0.197498	NaN	1.000000	-0.141584	0.001606	-0.193450	-0.221651	

```
ALP   0.173340  NaN -0.141584  1.000000  0.214480  0.063948  0.056078
ALT  -0.006021  NaN  0.001606  0.214480  1.000000  0.273326 -0.038469
AST   0.088666  NaN -0.193450  0.063948  0.273326  1.000000  0.312231
BIL   0.032492  NaN -0.221651  0.056078 -0.038469  0.312231  1.000000
CHE  -0.075093  NaN  0.375878  0.033753  0.147000 -0.208536 -0.333172
CHOL  0.125641  NaN  0.208248  0.125429  0.068947 -0.209970 -0.180370
CREA -0.022296  NaN -0.001573  0.149832 -0.043025 -0.021387  0.031224
GGT   0.153087  NaN -0.155749  0.454630  0.248114  0.491263  0.217024
PROT -0.153668  NaN  0.557197 -0.055109  0.094730  0.040071 -0.047638

            CHE      CHOL      CREA       GGT      PROT
Age   -0.075093  0.125641 -0.022296  0.153087 -0.153668
Sex         NaN       NaN       NaN       NaN       NaN
ALB    0.375878  0.208248 -0.001573 -0.155749  0.557197
ALP    0.033753  0.125429  0.149832  0.454630 -0.055109
ALT    0.147000  0.068947 -0.043025  0.248114  0.094730
AST   -0.208536 -0.209970 -0.021387  0.491263  0.040071
BIL   -0.333172 -0.180370  0.031224  0.217024 -0.047638
CHE    1.000000  0.425456 -0.011157 -0.110345  0.295427
CHOL   0.425456  1.000000 -0.047744 -0.006895  0.207071
CREA  -0.011157 -0.047744  1.000000  0.121003 -0.031704
GGT   -0.110345 -0.006895  0.121003  1.000000 -0.011767
PROT   0.295427  0.207071 -0.031704 -0.011767  1.000000
```

[88]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Create a larger figure
plt.figure(figsize=(16, 14))

# Create the heatmap
plt.matshow(corr_matrix, fignum=None, vmin=-1, vmax=1, cmap='coolwarm')

# Adjust font size for readability
plt.xticks(range(df_numeric.shape[1]), df_numeric.columns, rotation=90,␣
 ↪fontsize=10)
plt.yticks(range(df_numeric.shape[1]), df_numeric.columns, fontsize=10)

plt.xlabel('Blood Features', fontsize=12)
plt.ylabel('Blood Features', fontsize=12)
plt.colorbar(label='Correlation Coefficient', shrink=0.8)

# Adjust layout to prevent cutting off labels
plt.tight_layout()

plt.show()
```

```
/var/folders/5f/vr6wj5x522d5sydssspp10r80000gn/T/ipykernel_22367/2612980567.py:2
0: UserWarning: This figure includes Axes that are not compatible with
tight_layout, so results might be incorrect.
  plt.tight_layout()
```

```
<Figure size 1600x1400 with 0 Axes>
```



1.6 Step 2: Splitting the Data

[89]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold

X = df.drop(columns=['Category','Unnamed: 0'])
y = df['Category']

def StratifiedSplit(X, y, random_state, test_size, n_splits):
    # stratified train-test split
    X_other, X_test, y_other, y_test = train_test_split(X, y,␣
 ↪test_size=test_size, stratify=y, random_state=random_state)

    # do StratifiedKFold split on other
    kf = StratifiedKFold(n_splits=n_splits, shuffle=True,␣
 ↪random_state=random_state)
```

```
    for train_index, val_index in kf.split(X_other, y_other):
        X_train = X_other.iloc[train_index]
        y_train = y_other.iloc[train_index]
        X_val = X_other.iloc[val_index]
        y_val = y_other.iloc[val_index]

        print("Train set:")
        print(X_train)
        print(y_train)
        print("\nValidation set:")
        print(X_val)
        print(y_val)
        print("\nTest set:")
        print(X_test)
        print(y_test)
        print("---------------------")

    return X_train, y_train, X_val, y_val, X_test, y_test

# Call the function
X_train, y_train, X_val, y_val, X_test, y_test = StratifiedSplit(X, y,␣
 ↪random_state=42, test_size=0.2, n_splits=4)

# # Now you can print X_train and y_train separately
# print("Final X_train:")
# print(X_train)
# print("\nFinal y_train:")
# print(y_train)
```

```
Train set:
     Age     Sex   ALB   ALP   ALT    AST    BIL   CHE  CHOL  CREA    GGT  \
441   49  Female  39.3  59.4  18.3   15.0    4.8  8.03  4.58  83.0   12.5
216   52    Male  82.2  82.2  37.0   23.7    7.8  8.90  6.09  77.0   87.8
605   42  Female  33.0  79.0   3.7   55.7  200.0  1.72  5.16  89.1  146.3
341   34  Female  39.7  39.3  11.2   16.4    8.4  5.27  4.68  61.0   24.3
595   56    Male  27.0  81.1  17.0  319.8   37.0  1.42  3.54  66.9   93.7
..   ...     ...   ...   ...   ...    ...    ...   ...   ...   ...    ...
432   48  Female  43.7  50.1  17.3   26.3    8.1  8.15  5.38  64.0   13.4
27    34    Male  29.0  41.6  29.1   16.1    4.8  6.82  4.03  62.0   14.5
561   41  Female  37.0  31.2   8.2   38.3    7.0  7.08  5.30  60.8   24.7
63    37    Male  50.4  48.5  19.4   27.5   11.6  5.78  4.93  90.0   27.8
347   35  Female  42.0  69.0  19.9   16.6   10.8  7.85  4.43  67.0   15.1

     PROT
441  74.3
216  67.4
605  69.9
341  71.5
```

```
595  65.3
..    …
432  73.1
27   53.2
561  82.4
63   75.0
347  64.1

[369 rows x 12 columns]
441    0=Blood Donor
216    0=Blood Donor
605      3=Cirrhosis
341    0=Blood Donor
595      3=Cirrhosis
           …
432    0=Blood Donor
27     0=Blood Donor
561      1=Hepatitis
63     0=Blood Donor
347    0=Blood Donor
Name: Category, Length: 369, dtype: object

Validation set:
     Age     Sex   ALB    ALP   ALT   AST    BIL    CHE  CHOL   CREA    GGT  \
545   29    Male  49.0    NaN  53.0  39.0   15.0   8.79  3.60   79.0   37.0
334   33  Female  40.6   73.7  12.6  16.3    3.1   7.75  6.36   67.0   19.5
351   35  Female  44.7   83.2  25.3  22.6    3.9   8.02  5.73   68.0   10.8
587   41    Male  31.0   85.3   4.8  60.2  200.0   1.80  5.34  106.4  151.0
300   65    Male  44.7   99.4  31.9  30.5   12.2   7.15  6.31   82.0   38.5
..   …     …   …    …   …   …    …   …   …    …    …                    …
364   38  Female  40.0   79.3  11.9  22.0    6.5   8.33  4.58   60.0   13.7
66    37    Male  40.8  118.9  17.2  19.2    3.2   9.17  4.26   88.0   13.5
215   52    Male  42.2   72.2  47.9  23.7    8.6  11.91  6.29   96.0   62.5
94    40    Male  41.4   67.5  59.8  36.8    7.3   4.18  6.02   76.0   92.7
237   54    Male  46.0   70.2  18.6  24.7   24.1   7.83  6.24   76.0   24.3

     PROT
545  90.0
334  71.4
351  76.4
587  71.8
300  75.7
..    …
364  68.1
66   72.0
215  72.9
94   72.5
237  76.8
```

```
[123 rows x 12 columns]
545      1=Hepatitis
334    0=Blood Donor
351    0=Blood Donor
587      3=Cirrhosis
300    0=Blood Donor
          …
364    0=Blood Donor
66     0=Blood Donor
215    0=Blood Donor
94     0=Blood Donor
237    0=Blood Donor
Name: Category, Length: 123, dtype: object

Test set:
     Age     Sex   ALB    ALP   ALT   AST   BIL    CHE  CHOL   CREA   GGT  \
270   59    Male  39.8   49.4  25.4  21.4  24.7   7.50  3.69   86.0  18.7
454   51  Female  38.3   52.9  12.4  16.5   3.8   7.22  5.43   55.0  12.7
235   53    Male  49.2   71.8  42.8  29.4   6.8  15.10  6.24  107.0  48.3
470   52  Female  36.7   87.6  34.3  30.8  17.7  10.12  6.98   72.0  24.2
58    37    Male  44.8   94.3  32.2  36.7   6.3   9.76  4.12  113.0  23.8
..   …     …   …    …   …   …   …    …   …    …   …
337   34  Female  36.3   63.2  21.4  20.4   4.6   7.41  5.17   75.0  18.7
494   56  Female  36.6  102.3  13.5  14.9   8.4   6.94  5.50   65.0  16.2
493   56  Female  34.7   90.3  22.7  21.6   3.5   8.07  5.45   67.0   9.0
443   49  Female  34.9   37.9  15.3  19.4   7.1   5.30  5.88   83.0   7.9
445   49  Female  43.3   71.5  28.4  26.0   6.2   7.68  5.91   77.0  19.1

     PROT
270  71.9
454  70.2
235  77.8
470  66.3
58   72.5
..    …
337  64.2
494  71.0
493  69.4
443  62.5
445  76.9

[123 rows x 12 columns]
270    0=Blood Donor
454    0=Blood Donor
235    0=Blood Donor
470    0=Blood Donor
58     0=Blood Donor
```

```
          …
337    0=Blood Donor
494    0=Blood Donor
493    0=Blood Donor
443    0=Blood Donor
445    0=Blood Donor
Name: Category, Length: 123, dtype: object
--------------------
Train set:
     Age     Sex   ALB   ALP   ALT   AST   BIL   CHE  CHOL  CREA   GGT  PROT
545   29    Male  49.0   NaN  53.0  39.0  15.0  8.79  3.60  79.0  37.0  90.0
441   49  Female  39.3  59.4  18.3  15.0   4.8  8.03  4.58  83.0  12.5  74.3
334   33  Female  40.6  73.7  12.6  16.3   3.1  7.75  6.36  67.0  19.5  71.4
351   35  Female  44.7  83.2  25.3  22.6   3.9  8.02  5.73  68.0  10.8  76.4
341   34  Female  39.7  39.3  11.2  16.4   8.4  5.27  4.68  61.0  24.3  71.5
..   …     …   …   …   …   …   …   …   …   …   …   …
449   50  Female  39.9  80.5  24.2  22.8   5.2  9.25  7.41  84.0  19.4  71.2
237   54    Male  46.0  70.2  18.6  24.7  24.1  7.83  6.24  76.0  24.3  76.8
478   53  Female  41.1  91.7  13.8  19.6   3.4  7.87  5.48  72.0  77.3  77.3
27    34    Male  29.0  41.6  29.1  16.1   4.8  6.82  4.03  62.0  14.5  53.2
347   35  Female  42.0  69.0  19.9  16.6  10.8  7.85  4.43  67.0  15.1  64.1

[369 rows x 12 columns]
545      1=Hepatitis
441    0=Blood Donor
334    0=Blood Donor
351    0=Blood Donor
341    0=Blood Donor
          …
449    0=Blood Donor
237    0=Blood Donor
478    0=Blood Donor
27     0=Blood Donor
347    0=Blood Donor
Name: Category, Length: 369, dtype: object

Validation set:
     Age     Sex   ALB   ALP   ALT    AST    BIL   CHE  CHOL  CREA    GGT  \
216   52    Male  82.2  82.2  37.0   23.7    7.8  8.90  6.09  77.0   87.8
605   42  Female  33.0  79.0   3.7   55.7  200.0  1.72  5.16  89.1  146.3
595   56    Male  27.0  81.1  17.0  319.8   37.0  1.42  3.54  66.9   93.7
110   42    Male  37.8  78.6  51.4   31.8   10.1  9.66  6.15  85.0   15.1
574   59    Male  44.0  34.5   8.9   74.5    6.0  9.45  4.45  65.0   95.3
..   …     …   …   …   …     …     …   …   …   …     …   …
418   46  Female  51.3  84.1  40.6   43.6    9.2  7.10  5.62  62.0   74.9
398   45  Female  39.5  92.2  18.7   19.4    3.5  8.32  5.38  85.0   15.8
432   48  Female  43.7  50.1  17.3   26.3    8.1  8.15  5.38  64.0   13.4
561   41  Female  37.0  31.2   8.2   38.3    7.0  7.08  5.30  60.8   24.7
```

```
63    37    Male  50.4  48.5  19.4   27.5   11.6  5.78  4.93  90.0   27.8

     PROT
216  67.4
605  69.9
595  65.3
110  70.8
574  69.7
..    …
418  77.1
398  72.2
432  73.1
561  82.4
63   75.0

[123 rows x 12 columns]
216    0=Blood Donor
605      3=Cirrhosis
595      3=Cirrhosis
110    0=Blood Donor
574       2=Fibrosis
           …
418    0=Blood Donor
398    0=Blood Donor
432    0=Blood Donor
561      1=Hepatitis
63     0=Blood Donor
Name: Category, Length: 123, dtype: object

Test set:
     Age     Sex   ALB    ALP   ALT   AST   BIL    CHE  CHOL   CREA   GGT  \
270   59    Male  39.8   49.4  25.4  21.4  24.7   7.50  3.69   86.0  18.7
454   51  Female  38.3   52.9  12.4  16.5   3.8   7.22  5.43   55.0  12.7
235   53    Male  49.2   71.8  42.8  29.4   6.8  15.10  6.24  107.0  48.3
470   52  Female  36.7   87.6  34.3  30.8  17.7  10.12  6.98   72.0  24.2
58    37    Male  44.8   94.3  32.2  36.7   6.3   9.76  4.12  113.0  23.8
..   …     …   …    …   …   …   …    …   …    …   …
337   34  Female  36.3   63.2  21.4  20.4   4.6   7.41  5.17   75.0  18.7
494   56  Female  36.6  102.3  13.5  14.9   8.4   6.94  5.50   65.0  16.2
493   56  Female  34.7   90.3  22.7  21.6   3.5   8.07  5.45   67.0   9.0
443   49  Female  34.9   37.9  15.3  19.4   7.1   5.30  5.88   83.0   7.9
445   49  Female  43.3   71.5  28.4  26.0   6.2   7.68  5.91   77.0  19.1

     PROT
270  71.9
454  70.2
235  77.8
470  66.3
```

```
58   72.5
..    …
337  64.2
494  71.0
493  69.4
443  62.5
445  76.9

[123 rows x 12 columns]
270    0=Blood Donor
454    0=Blood Donor
235    0=Blood Donor
470    0=Blood Donor
58     0=Blood Donor
           …
337    0=Blood Donor
494    0=Blood Donor
493    0=Blood Donor
443    0=Blood Donor
445    0=Blood Donor
Name: Category, Length: 123, dtype: object
--------------------
Train set:
     Age     Sex   ALB   ALP   ALT   AST    BIL   CHE  CHOL  CREA    GGT  PROT
545   29    Male  49.0   NaN  53.0  39.0   15.0  8.79  3.60  79.0   37.0  90.0
441   49  Female  39.3  59.4  18.3  15.0    4.8  8.03  4.58  83.0   12.5  74.3
334   33  Female  40.6  73.7  12.6  16.3    3.1  7.75  6.36  67.0   19.5  71.4
216   52    Male  82.2  82.2  37.0  23.7    7.8  8.90  6.09  77.0   87.8  67.4
605   42  Female  33.0  79.0   3.7  55.7  200.0  1.72  5.16  89.1  146.3  69.9
..   …     …   …   …   …   …     …   …   …   …    …   …                    
237   54    Male  46.0  70.2  18.6  24.7   24.1  7.83  6.24  76.0   24.3  76.8
432   48  Female  43.7  50.1  17.3  26.3    8.1  8.15  5.38  64.0   13.4  73.1
561   41  Female  37.0  31.2   8.2  38.3    7.0  7.08  5.30  60.8   24.7  82.4
63    37    Male  50.4  48.5  19.4  27.5   11.6  5.78  4.93  90.0   27.8  75.0
347   35  Female  42.0  69.0  19.9  16.6   10.8  7.85  4.43  67.0   15.1  64.1

[369 rows x 12 columns]
545      1=Hepatitis
441    0=Blood Donor
334    0=Blood Donor
216    0=Blood Donor
605      3=Cirrhosis
           …
237    0=Blood Donor
432    0=Blood Donor
561      1=Hepatitis
63     0=Blood Donor
347    0=Blood Donor
```

```
Name: Category, Length: 369, dtype: object

Validation set:
     Age     Sex   ALB    ALP   ALT   AST   BIL    CHE  CHOL   CREA    GGT  \
290   63    Male  39.5   59.6  26.2  25.4  12.4   6.78  6.18   78.0   22.6
416   46  Female  37.9   59.5  33.0  25.0   3.7   6.06  5.30   92.0   43.9
142   45    Male  43.2   68.2  27.8  42.3   6.6  10.93  6.61  105.0   27.2
238   54    Male  43.0   67.0  36.1  26.1   5.0  10.20  5.98  105.0   45.4
414   46  Female  41.1   47.5  21.0  17.7   7.1   7.55  4.42   62.0   11.9
..   …      …   …    …    …   …   …    …   …    …    …
559   58    Male  43.0   99.1  12.2  63.2  13.0   5.95  6.15  147.3  491.0
462   51  Female  47.4  117.3  62.1  30.4   3.8  10.43  6.59   86.0   69.3
449   50  Female  39.9   80.5  24.2  22.8   5.2   9.25  7.41   84.0   19.4
478   53  Female  41.1   91.7  13.8  19.6   3.4   7.87  5.48   72.0   77.3
27    34    Male  29.0   41.6  29.1  16.1   4.8   6.82  4.03   62.0   14.5

     PROT
290  72.7
416  70.0
142  74.5
238  75.9
414  69.8
..    …
559  65.6
462  71.0
449  71.2
478  77.3
27   53.2

[123 rows x 12 columns]
290    0=Blood Donor
416    0=Blood Donor
142    0=Blood Donor
238    0=Blood Donor
414    0=Blood Donor
          …
559      1=Hepatitis
462    0=Blood Donor
449    0=Blood Donor
478    0=Blood Donor
27     0=Blood Donor
Name: Category, Length: 123, dtype: object

Test set:
     Age     Sex   ALB    ALP   ALT   AST   BIL    CHE  CHOL   CREA   GGT  \
270   59    Male  39.8   49.4  25.4  21.4  24.7   7.50  3.69   86.0  18.7
454   51  Female  38.3   52.9  12.4  16.5   3.8   7.22  5.43   55.0  12.7
235   53    Male  49.2   71.8  42.8  29.4   6.8  15.10  6.24  107.0  48.3
```

```
470   52  Female  36.7   87.6  34.3  30.8  17.7  10.12  6.98   72.0  24.2
58    37    Male  44.8   94.3  32.2  36.7   6.3   9.76  4.12  113.0  23.8
..   …     …   …    …   …   …   …    …   …    …   …
337   34  Female  36.3   63.2  21.4  20.4   4.6   7.41  5.17   75.0  18.7
494   56  Female  36.6  102.3  13.5  14.9   8.4   6.94  5.50   65.0  16.2
493   56  Female  34.7   90.3  22.7  21.6   3.5   8.07  5.45   67.0   9.0
443   49  Female  34.9   37.9  15.3  19.4   7.1   5.30  5.88   83.0   7.9
445   49  Female  43.3   71.5  28.4  26.0   6.2   7.68  5.91   77.0  19.1

     PROT
270  71.9
454  70.2
235  77.8
470  66.3
58   72.5
..    …
337  64.2
494  71.0
493  69.4
443  62.5
445  76.9

[123 rows x 12 columns]
270    0=Blood Donor
454    0=Blood Donor
235    0=Blood Donor
470    0=Blood Donor
58     0=Blood Donor
           …
337    0=Blood Donor
494    0=Blood Donor
493    0=Blood Donor
443    0=Blood Donor
445    0=Blood Donor
Name: Category, Length: 123, dtype: object
--------------------
Train set:
     Age     Sex   ALB   ALP   ALT   AST    BIL   CHE  CHOL  CREA    GGT  PROT
545   29    Male  49.0   NaN  53.0  39.0   15.0  8.79  3.60  79.0   37.0  90.0
334   33  Female  40.6  73.7  12.6  16.3    3.1  7.75  6.36  67.0   19.5  71.4
216   52    Male  82.2  82.2  37.0  23.7    7.8  8.90  6.09  77.0   87.8  67.4
605   42  Female  33.0  79.0   3.7  55.7  200.0  1.72  5.16  89.1  146.3  69.9
351   35  Female  44.7  83.2  25.3  22.6    3.9  8.02  5.73  68.0   10.8  76.4
..   …     …   …   …   …   …   …    …   …   …   …   …
478   53  Female  41.1  91.7  13.8  19.6    3.4  7.87  5.48  72.0   77.3  77.3
432   48  Female  43.7  50.1  17.3  26.3    8.1  8.15  5.38  64.0   13.4  73.1
27    34    Male  29.0  41.6  29.1  16.1    4.8  6.82  4.03  62.0   14.5  53.2
561   41  Female  37.0  31.2   8.2  38.3    7.0  7.08  5.30  60.8   24.7  82.4
```

```
63    37    Male  50.4  48.5  19.4  27.5   11.6  5.78  4.93  90.0   27.8  75.0

[369 rows x 12 columns]
545      1=Hepatitis
334    0=Blood Donor
216    0=Blood Donor
605      3=Cirrhosis
351    0=Blood Donor
          …
478    0=Blood Donor
432    0=Blood Donor
27     0=Blood Donor
561      1=Hepatitis
63     0=Blood Donor
Name: Category, Length: 369, dtype: object

Validation set:
     Age     Sex   ALB   ALP    ALT   AST   BIL    CHE  CHOL   CREA   GGT  \
441   49  Female  39.3  59.4   18.3  15.0   4.8   8.03  4.58   83.0  12.5
341   34  Female  39.7  39.3   11.2  16.4   8.4   5.27  4.68   61.0  24.3
246   55    Male  46.2  87.1   36.9  21.0   4.5   7.55  6.33   80.0  30.2
56    37    Male  41.5  64.6   23.7  29.9   9.3   5.49  3.97  100.0  10.4
568   49    Male  39.0   NaN  118.0  62.0  10.0   7.28  3.50   72.0  74.0
..   …     …   …   …    …   …   …    …   …    …   …
481   53  Female  51.3  84.1   40.6  43.6   9.2   7.10  5.62   62.0  74.9
369   40  Female  43.2  42.4   15.7  23.6   9.7   7.56  6.74   88.0  11.5
40    35    Male  47.4  54.5   18.6  21.6  10.3   8.10  6.23   66.0  28.1
244   55    Male  42.9  92.6   21.6  26.1   7.4  12.86  5.73   94.0  42.9
347   35  Female  42.0  69.0   19.9  16.6  10.8   7.85  4.43   67.0  15.1

     PROT
441  74.3
341  71.5
246  72.2
56   69.3
568  81.0
..    …
481  77.1
369  73.2
40   74.0
244  70.1
347  64.1

[123 rows x 12 columns]
441    0=Blood Donor
341    0=Blood Donor
246    0=Blood Donor
56     0=Blood Donor
```

```
568       2=Fibrosis
           …
481    0=Blood Donor
369    0=Blood Donor
40     0=Blood Donor
244    0=Blood Donor
347    0=Blood Donor
Name: Category, Length: 123, dtype: object

Test set:
     Age     Sex   ALB    ALP   ALT   AST   BIL    CHE  CHOL   CREA   GGT  \
270   59    Male  39.8   49.4  25.4  21.4  24.7   7.50  3.69   86.0  18.7
454   51  Female  38.3   52.9  12.4  16.5   3.8   7.22  5.43   55.0  12.7
235   53    Male  49.2   71.8  42.8  29.4   6.8  15.10  6.24  107.0  48.3
470   52  Female  36.7   87.6  34.3  30.8  17.7  10.12  6.98   72.0  24.2
58    37    Male  44.8   94.3  32.2  36.7   6.3   9.76  4.12  113.0  23.8
..   …     …   …    …   …   …   …    …   …    …   …
337   34  Female  36.3   63.2  21.4  20.4   4.6   7.41  5.17   75.0  18.7
494   56  Female  36.6  102.3  13.5  14.9   8.4   6.94  5.50   65.0  16.2
493   56  Female  34.7   90.3  22.7  21.6   3.5   8.07  5.45   67.0   9.0
443   49  Female  34.9   37.9  15.3  19.4   7.1   5.30  5.88   83.0   7.9
445   49  Female  43.3   71.5  28.4  26.0   6.2   7.68  5.91   77.0  19.1

     PROT
270  71.9
454  70.2
235  77.8
470  66.3
58   72.5
..    …
337  64.2
494  71.0
493  69.4
443  62.5
445  76.9

[123 rows x 12 columns]
270    0=Blood Donor
454    0=Blood Donor
235    0=Blood Donor
470    0=Blood Donor
58     0=Blood Donor
           …
337    0=Blood Donor
494    0=Blood Donor
493    0=Blood Donor
443    0=Blood Donor
445    0=Blood Donor
```

```
Name: Category, Length: 123, dtype: object
--------------------
```

[90]:
```
# from sklearn.model_selection import train_test_split
# from sklearn.model_selection import StratifiedKFold

# X = df.drop(columns=['Category','Unnamed: 0'])
# y = df['Category']

# def StratifiedSplit(X,y,random_state,test_size,n_splits):
#     # stratified train-test split
#     X_other, X_test, y_other, y_test = train_test_split(X,y,test_size =␣
 ↪test_size,stratify=y,random_state=random_state)

#     # do StratifiedKFold split on other
#     kf =␣
 ↪StratifiedKFold(n_splits=n_splits,shuffle=True,random_state=random_state)
#     for train_index, val_index in kf.split(X_other,y_other):
#         X_train = X_other.iloc[train_index]
#         y_train = y_other.iloc[train_index]
#         X_val = X_other.iloc[val_index]
#         y_val = y_other.iloc[val_index]
#         print(f"Train set:\n{X_train,y_train}\n")
#         print(f"Validation set:\n{X_val,y_val}\n")
#         print(f"Test set:\n{X_test,y_test}\n")
#         print("--------------------")


# StratifiedSplit(X,y,42,0.2,4)
```

1.7 Step 3: Preprocessing the Data

1.7.1 Fit and Transform the Data

[91]:
```
import pandas as pd
import numpy as np

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder,␣
 ↪OrdinalEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split

# #np.random.seed(0)

# df = pd.read_csv('hcvdat0.csv')

# # let's separate the feature matrix X, and target variable y
```

```
# y = df['Category'] # remember, we want to predict who earns more than 50k or␣
 ↪less than 50k
# X = df.loc[:, df.columns != 'Category'] # all other columns are features

# random_state = 42

# # first split to separate out the training set
# X_train, X_other, y_train, y_other = train_test_split(X,y,train_size = 0.
 ↪6,random_state=random_state)

# # second split to separate out the validation and test sets
# X_val, X_test, y_val, y_test = train_test_split(X_other,y_other,train_size =␣
 ↪0.5,random_state=random_state)

# print(X_train)
# print(y_train)
```

[92]:
```
# from sklearn.compose import ColumnTransformer
# from sklearn.preprocessing import OneHotEncoder, MinMaxScaler,␣
 ↪StandardScaler, OrdinalEncoder
# from sklearn.pipeline import Pipeline

# # Define which features need encoding/scaling
# ordinal_ftrs = []
# ordinal_cats = []
# onehot_ftrs = ['Sex']  # One-hot encode 'Sex'
# minmax_ftrs = ['Age']
# std_ftrs = ['ALB','ALP','ALT','AST','BIL','CHE','CHOL','CREA','GGT','PROT']

# # Create the preprocessor pipeline
# preprocessor = ColumnTransformer(
#     transformers=[
#         ('ord', OrdinalEncoder(categories=ordinal_cats), ordinal_ftrs),
#         ('onehot', OneHotEncoder(sparse_output=False,␣
 ↪handle_unknown='ignore', drop='first'), onehot_ftrs),
#         ('minmax', MinMaxScaler(), minmax_ftrs),
#         ('std', StandardScaler(), std_ftrs)
#     ])

# # Add preprocessor to a full pipeline (you can add a classifier later)
# clf = Pipeline(steps=[('preprocessor', preprocessor)])

# # Transform the training, validation, and test sets
# X_train_prep = clf.fit_transform(X_train)
# X_val_prep = clf.transform(X_val)
# X_test_prep = clf.transform(X_test)
```

```
# print(X_train_prep.shape)
# print(X_train_prep)  # Check if 'Sex' is encoded correctly (as 0/1)
```

[93]:
```
# collect which encoder to use on each feature
# needs to be done manually
ordinal_ftrs = []
ordinal_cats = []
onehot_ftrs = ['Sex']
minmax_ftrs = ['Age']
std_ftrs = ['ALB','ALP','ALT','AST','BIL','CHE','CHOL','CREA','GGT','PROT']

# collect all the encoders
preprocessor = ColumnTransformer(
    transformers=[
        ('ord', OrdinalEncoder(categories = ordinal_cats), ordinal_ftrs),
        ('onehot', OneHotEncoder(sparse_output=False,handle_unknown='ignore'),␣
 ↪onehot_ftrs),
        ('minmax', MinMaxScaler(), minmax_ftrs),
        ('std', StandardScaler(), std_ftrs)])

clf = Pipeline(steps=[('preprocessor', preprocessor)]) # for now we only␣
 ↪preprocess
                                                       # later on we will add␣
 ↪other steps here

X_train_prep = clf.fit_transform(X_train)
X_val_prep = clf.transform(X_val)
X_test_prep = clf.transform(X_test)

print(X_train.shape)
print(X_train_prep.shape)
print(X_train_prep)
print(X_val_prep)
print(X_test_prep)
```

```
(369, 12)
(369, 13)
[[ 0.          1.          0.17241379 … -0.06599065 -0.05450203
   3.2602577 ]
 [ 1.          0.          0.24137931 … -0.25717529 -0.37425025
  -0.07508084]
 [ 0.          1.          0.56896552 … -0.09785476  0.87368137
  -0.79235794]
 …
 [ 0.          1.          0.25862069 … -0.33683556 -0.46560688
  -3.33869166]
 [ 1.          0.          0.37931034 … -0.35595402 -0.27923935
```

```
   1.8974312 ]
 [ 0.          1.          0.31034483 …  0.10926194 -0.22259824
   0.57046856]]
[[ 1.          0.          0.51724138 … -0.00226244 -0.50214954
   0.44494507]
 [ 1.          0.          0.25862069 … -0.35276761 -0.28654788
  -0.05714891]
 [ 0.          1.          0.62068966 … -0.0500586  -0.17874705
   0.06837459]
 …
 [ 0.          1.          0.27586207 … -0.27310734 -0.21711684
   0.39114928]
 [ 0.          1.          0.62068966 …  0.17299015  0.0532988
  -0.30819589]
 [ 1.          0.          0.27586207 … -0.25717529 -0.45464409
  -1.38411155]]
[[ 0.          1.          0.68965517 …  0.04553372 -0.38886731
   0.0145788 ]
 [ 1.          0.          0.55172414 … -0.44835993 -0.49849527
  -0.29026397]
 [ 0.          1.          0.5862069  …  0.38010684  0.15196396
   1.07256253]
 …
 [ 1.          0.          0.63793103 … -0.25717529 -0.56609918
  -0.43371939]
 [ 1.          0.          0.51724138 … -0.00226244 -0.58619764
  -1.67102239]
 [ 1.          0.          0.51724138 … -0.09785476 -0.38155878
   0.91117518]]
```

1.7.2 Missing Values

[94]:
```
# read the data
import pandas as pd
import numpy  as np
from sklearn.model_selection import train_test_split

# drop the ID
df.drop(columns=['Unnamed: 0'],inplace=True)

# the target variable
y = df['Category']
df.drop(columns=['Category'],inplace=True)
# the unprocessed feature matrix
X = df
print(X.shape)
# the feature names
```

```
ftrs = df.columns
print(df.head())
```

```
(615, 12)
   Age   Sex   ALB   ALP   ALT   AST   BIL    CHE  CHOL   CREA   GGT  PROT
0   32  Male  38.5  52.5   7.7  22.1   7.5   6.93  3.23  106.0  12.1  69.0
1   32  Male  38.5  70.3  18.0  24.7   3.9  11.17  4.80   74.0  15.6  76.5
2   32  Male  46.9  74.7  36.2  52.6   6.1   8.84  5.20   86.0  33.2  79.3
3   32  Male  43.2  52.0  30.6  22.6  18.9   7.33  4.74   80.0  33.8  75.7
4   32  Male  39.2  74.1  32.6  24.8   9.6   9.15  4.32   76.0  29.9  68.7
```

[95]:
```
print('data dimensions:',df.shape)
perc_missing_per_ftr = df.isnull().sum(axis=0)/df.shape[0]
print('fraction of missing values in features:')
print(perc_missing_per_ftr[perc_missing_per_ftr > 0])
print('data types of the features with missing values:')
print(df[perc_missing_per_ftr[perc_missing_per_ftr > 0].index].dtypes)
frac_missing = sum(df.isnull().sum(axis=1)!=0)/df.shape[0]
print('fraction of points with missing values:',frac_missing)
```

```
data dimensions: (615, 12)
fraction of missing values in features:
ALB     0.001626
ALP     0.029268
ALT     0.001626
CHOL    0.016260
PROT    0.001626
dtype: float64
data types of the features with missing values:
ALB     float64
ALP     float64
ALT     float64
CHOL    float64
PROT    float64
dtype: object
fraction of points with missing values: 0.04227642276422764
```

[96]:
```
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
print(X_train_prep.shape)
print(X_val_prep.shape)
print(X_test_prep.shape)
```

```
(369, 12)
(123, 12)
(123, 12)
(369, 13)
(123, 13)
```

```
(123, 13)
```

[97]:
```
# collect the various features
cat_ftrs = ['Sex']
ordinal_ftrs = []
ordinal_cats = []
num_ftrs =␣
 ↪['Age','ALB','ALP','ALT','AST','BIL','CHE','CHOL','CREA','GGT','PROT']
```

[98]:
```
# preprocess with pipeline and columntransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor

random_state = 42

# one-hot encoder
# We need to replace the NaN with a string first!
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant',fill_value='missing')),
    ('onehot', OneHotEncoder(sparse_output=False,handle_unknown='ignore'))])

# ordinal encoder
# We need to replace the NaN with a string first!
ordinal_transformer = Pipeline(steps=[
    ('imputer2', SimpleImputer(strategy='constant',fill_value='NA')),
    ('ordinal', OrdinalEncoder(categories = ordinal_cats))])

# standard scaler
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())])

# collect the transformers
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, num_ftrs),
        ('cat', categorical_transformer, cat_ftrs),
        ('ord', ordinal_transformer, ordinal_ftrs)])
```

[99]:
```
# fit_transform the training set
X_prep = preprocessor.fit_transform(X_train)
```

```
# the feature names after fit
feature_names = preprocessor.get_feature_names_out()

# you can convert the numpy array back to a data frame with the feature names␣
 ↪if you want
df_train = pd.DataFrame(data=X_prep,columns=feature_names)
print(df_train.shape)

# transform the CV
df_val = preprocessor.transform(X_val)
df_val = pd.DataFrame(data=df_val,columns = feature_names)
print(df_val.shape)

# transform the test
df_test = preprocessor.transform(X_test)
df_test = pd.DataFrame(data=df_test,columns = feature_names)
print(df_test.shape)
print(feature_names)
```

```
(369, 13)
(123, 13)
(123, 13)
['num__Age' 'num__ALB' 'num__ALP' 'num__ALT' 'num__AST' 'num__BIL'
 'num__CHE' 'num__CHOL' 'num__CREA' 'num__GGT' 'num__PROT'
 'cat__Sex_Female' 'cat__Sex_Male']
```

[100]:
```
print('data dimensions:',df_train.shape)
perc_missing_per_ftr = df_train.isnull().sum(axis=0)/df_train.shape[0]
print('fraction of missing values in features:')
print(perc_missing_per_ftr[perc_missing_per_ftr > 0])
print('data types of the features with missing values:')
print(df_train[perc_missing_per_ftr[perc_missing_per_ftr > 0].index].dtypes)
frac_missing = sum(df_train.isnull().sum(axis=1)!=0)/df_train.shape[0]
print('fraction of points with missing values:',frac_missing)
```

```
data dimensions: (369, 13)
fraction of missing values in features:
num__ALP     0.02981
num__ALT     0.00271
num__CHOL    0.01355
dtype: float64
data types of the features with missing values:
num__ALP     float64
num__ALT     float64
num__CHOL    float64
dtype: object
fraction of points with missing values: 0.04336043360433604
```

1.7.3 Attempt at Linear and Nonlinear Regression

[101]:
```
# # your code here
# from sklearn.datasets import make_regression
# from sklearn.feature_selection import SelectKBest,f_regression

# feature_names = preprocessor.get_feature_names_out()
# X_train_prep_df = pd.DataFrame(
#     data = X_train_prep,
#     columns = feature_names
# )

# # X_train_prep
# f_statistic, p_values = f_regression(X_train_prep, y_train)
# # np.argsort(X_train_prep)
# # np.argsort(y_train)
# feature_importance = pd.DataFrame({
#     'Feature': feature_names,
#     'Importance': f_statistic
# })
# print(feature_names)
# top5features = feature_importance.nlargest(5, 'Importance'
# )

# sorted_indices = np.argsort(f_statistic)[::-1]
# sorted_top5 = sorted_indices[0:5]
# print(sorted_indices)
# print(sorted_top5)

# f_corr = f_statistic[sorted_indices[0:5]], feature_names[sorted_indices[0:5]]
# print(f_corr)

# # print(selected_feature_names)
# print(f_statistic)
# print(p_values)
# print(feature_names)
```

1.7.4 Drop Missing Values Rows and Columns

1. Drop the columns with missing values:

[102]:
```
# import pandas as pd
# import numpy as np

# #read in txt file using pandas read_csv function with a tab delimiter

# #print the shape of the original dataset
# print(df.shape)
```

```
# #replace the ? with NaN so that dropna() method can locate nulls
# # df = df.replace('?',np.NaN)
# # drop features with missing values
# df_c = df.dropna(axis=1)
# print(df_c.shape)
# df_c.head()
```

2. Drop the rows with missing values:

[103]:
```
# #print the shape of the original dataset
# print(df.shape)
# # by default, rows/points are dropped
# df_r = df.dropna()
# print(df_r.shape)
# df_r.head()
```

1.8 Step 4: Choose an Evaluation Metric

For this model, I have chosen to optimize for false negatives, since I have decided that missing a diagnosis for a patient that has hepatitis is greater than the cost associated with running extra tests and procedures. However, I do not want to completely ignore the costs associated with false positives, so I have decided that opting for an f_2 score serves as a way to weight recall more heavily, while not entirely discarding precision in my analysis. The dataset is also imbalanced, so a metric like accuracy does not make much sense.

1.9 Step 5: Choose one or more ML Techniques

[104]:
```
print(df.columns)
```

```
Index(['Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST', 'BIL', 'CHE', 'CHOL', 'CREA',
       'GGT', 'PROT'],
      dtype='object')
```

Random Forest Classifier

[105]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, fbeta_score

# Assuming X_train_prep, X_test_prep, y_train, y_test are already prepared

# Train a RandomForestClassifier using preprocessed data
model = RandomForestClassifier(random_state=42)

# Fit the model on preprocessed training data
model.fit(X_train_prep, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_prep)
```

```
# Evaluate the model using classification report (which includes F1-score)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Calculate F2 score (for each class and overall)
f2_score_macro = fbeta_score(y_test, y_pred, beta=2, average='macro')  #␣
 ↪Macro-average across all classes
f2_score_weighted = fbeta_score(y_test, y_pred, beta=2, average='weighted')  #␣
 ↪Weighted-average based on class support
f2_score_micro = fbeta_score(y_test,y_pred,beta=2,average = 'micro' ) #␣
 ↪Micro-average

print(f"F2 Score (Macro Average): {f2_score_macro}")
print(f"F2 Score (Weighted Average): {f2_score_weighted}")
print(f"F2 Score (Micro Average): {f2_score_micro}")
```

```
Classification Report:
                        precision    recall  f1-score   support

         0=Blood Donor       0.96      1.00      0.98       107
0s=suspect Blood Donor       0.00      0.00      0.00         1
           1=Hepatitis       1.00      0.40      0.57         5
            2=Fibrosis       0.50      0.50      0.50         4
           3=Cirrhosis       1.00      0.83      0.91         6

              accuracy                           0.94       123
             macro avg       0.69      0.55      0.59       123
          weighted avg       0.94      0.94      0.93       123

F2 Score (Macro Average): 0.5614710321606873
F2 Score (Weighted Average): 0.9386536611795931
F2 Score (Micro Average): 0.943089430894309
```

```
/opt/anaconda3/envs/data1030/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/envs/data1030/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/envs/data1030/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

XGBoost

[108]:
```
print(df.columns)
```

```
Index(['Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST', 'BIL', 'CHE', 'CHOL', 'CREA',
       'GGT', 'PROT'],
      dtype='object')
```

[107]:
```
from sklearn.preprocessing import LabelEncoder
import xgboost as xgb
from sklearn.metrics import classification_report, fbeta_score,␣
 ↪confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Assuming df is your DataFrame and 'Category' is your target variable

# Step 1: Label encode the target variable (y)
label_encoder = LabelEncoder()
df['Category_encoded'] = label_encoder.fit_transform(df['Category'])

# Now df['Category_encoded'] will contain numeric values like [0, 1, 2, 3, 4]␣
 ↪instead of strings

# Step 2: Split features (X) and target (y)
X = df.drop(columns=['Category', 'Category_encoded'])  # Features (all columns␣
 ↪except target)
y = df['Category_encoded']  # Target variable (encoded as numbers)

# Step 3: Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Step 4: Train an XGBoost model
xgb_model = xgb.XGBClassifier(random_state=42)
xgb_model.fit(X_train, y_train)

# Step 5: Make predictions and evaluate the model
y_pred = xgb_model.predict(X_test)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Calculate F2 score (for each class and overall)
f2_score_macro = fbeta_score(y_test, y_pred, beta=2, average='macro')
```

```
f2_score_weighted = fbeta_score(y_test, y_pred, beta=2, average='weighted')
f2_score_micro = fbeta_score(y_test,y_pred,beta=2,average = 'micro')

print(f"F2 Score (Macro Average): {f2_score_macro}")
print(f"F2 Score (Weighted Average): {f2_score_weighted}")
print(f"F2 Score (Micro Average): {f2_score_micro}")

# Step 6: Generate and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=label_encoder.
 ↪classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for XGBoost Classifier')
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File /opt/anaconda3/envs/data1030/lib/python3.12/site-packages/pandas/core/
 ↪indexes/base.py:3805, in Index.get_loc(self, key)
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7081, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7089, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'Category'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[107], line 10
      6 # Assuming df is your DataFrame and 'Category' is your target variable
      7
      8 # Step 1: Label encode the target variable (y)
      9 label_encoder = LabelEncoder()
---> 10 df['Category_encoded'] = label_encoder.fit_transform(df['Category'])
     12 # Now df['Category_encoded'] will contain numeric values like [0, 1, 2,
 ↪3, 4] instead of strings
     13
```

```
    14 # Step 2: Split features (X) and target (y)
    15 X = df.drop(columns=['Category', 'Category_encoded'])  # Features (all␣
 ↪columns except target)

File /opt/anaconda3/envs/data1030/lib/python3.12/site-packages/pandas/core/fram .
 ↪py:4102, in DataFrame.__getitem__(self, key)
  4100 if self.columns.nlevels > 1:
  4101     return self._getitem_multilevel(key)
-> 4102 indexer = self.columns.get_loc(key)
  4103 if is_integer(indexer):
  4104     indexer = [indexer]

File /opt/anaconda3/envs/data1030/lib/python3.12/site-packages/pandas/core/
 ↪indexes/base.py:3812, in Index.get_loc(self, key)
  3807     if isinstance(casted_key, slice) or (
  3808         isinstance(casted_key, abc.Iterable)
  3809         and any(isinstance(x, slice) for x in casted_key)
  3810     ):
  3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
  3813 except TypeError:
  3814     # If we have a listlike key, _check_indexing_error will raise
  3815     #  InvalidIndexError. Otherwise we fall through and re-raise
  3816     #  the TypeError.
  3817     self._check_indexing_error(key)

KeyError: 'Category'
```

[106]:
```
import xgboost as xgb
from sklearn.metrics import classification_report, fbeta_score,␣
 ↪confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Assuming X_train_prep and X_test_prep are already prepared (preprocessed␣
 ↪features)
# And y_train and y_test are the target labels (encoded if necessary)

# Step 1: Initialize the XGBoost Classifier
xgb_model = xgb.XGBClassifier(random_state=42)

# Step 2: Train the model on the training data
xgb_model.fit(X_train_prep, y_train)

# Step 3: Make predictions on the test set
y_pred = xgb_model.predict(X_test_prep)
```

```
# Step 4: Evaluate the model using classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Step 5: Calculate F2 score (for each class and overall)
f2_score_macro = fbeta_score(y_test, y_pred, beta=2, average='macro')
f2_score_weighted = fbeta_score(y_test, y_pred, beta=2, average='weighted')
f2_score_micro = fbeta_score(y_test,y_pred,beta=2,average = 'micro')

print(f"F2 Score (Macro Average): {f2_score_macro}")
print(f"F2 Score (Weighted Average): {f2_score_weighted}")
print(f"F2 Score (Micro Average): {f2_score_micro}")

# Step 6: Generate and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=xgb_model.
 ↪classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for XGBoost Classifier')
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[106], line 12
      9 xgb_model = xgb.XGBClassifier(random_state=42)
     11 # Step 2: Train the model on the training data
---> 12 xgb_model.fit(X_train_prep, y_train)
     14 # Step 3: Make predictions on the test set
     15 y_pred = xgb_model.predict(X_test_prep)

File /opt/anaconda3/envs/data1030/lib/python3.12/site-packages/xgboost/core.py:
 ↪726, in require_keyword_args.<locals>.throw_if.<locals>.inner_f(*args,␣
 ↪**kwargs)
    724 for k, arg in zip(sig.parameters, args):
    725     kwargs[k] = arg
--> 726 return func(**kwargs)

File /opt/anaconda3/envs/data1030/lib/python3.12/site-packages/xgboost/sklearn.
 ↪py:1491, in XGBClassifier.fit(self, X, y, sample_weight, base_margin,␣
 ↪eval_set, verbose, xgb_model, sample_weight_eval_set, base_margin_eval_set,␣
 ↪feature_weights)
   1486     expected_classes = self.classes_
   1487 if (
   1488     classes.shape != expected_classes.shape
   1489     or not (classes == expected_classes).all()
   1490 ):
-> 1491     raise ValueError(
   1492         f"Invalid classes inferred from unique values of `y`.  "
```

```
   1493         f"Expected: {expected_classes}, got {classes}"
   1494     )
   1496 params = self.get_xgb_params()
   1498 if callable(self.objective):

ValueError: Invalid classes inferred from unique values of `y`.  Expected: [0 1
 ↪2 3 4], got ['0=Blood Donor' '0s=suspect Blood Donor' '1=Hepatitis'␣
 ↪'2=Fibrosis'
 '3=Cirrhosis']
```

Logistic Regression

[ ]:
```
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.metrics import classification_report, fbeta_score

# # Assuming X_train_prep, X_test_prep, y_train, y_test are already prepared

# # Train a SVC model using preprocessed data
# model = KNeighborsClassifier()

# # Fit the model on preprocessed training data
# model.fit(X_train_prep, y_train)

# # Make predictions on the test set
# y_pred = model.predict(X_test_prep)

# # Evaluate the model using classification report (which includes F1-score)
# print("Classification Report:")
# print(classification_report(y_test, y_pred))

# # Calculate F2 score (for each class and overall)
# f2_score_macro = fbeta_score(y_test, y_pred, beta=2, average='macro')  #␣
 ↪Macro-average across all classes
# f2_score_weighted = fbeta_score(y_test, y_pred, beta=2, average='weighted') ␣
 ↪# Weighted-average based on class support
# f2_score_micro = fbeta_score(y_test,y_pred,beta=2,average = 'micro' ) #␣
 ↪Micro-average

# print(f"F2 Score (Macro Average): {f2_score_macro}")
# print(f"F2 Score (Weighted Average): {f2_score_weighted}")
# print(f"F2 Score (Micro Average): {f2_score_micro}")
```

1.9.1 Confusion Matrix

[ ]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
```

```
# cm = confusion_matrix(y_true,y_pred)
# disp = ConfusionMatrixDisplay(cm,display_labels=['class 0', 'class 1'])
# fig, ax = plt.subplots(figsize=(5,3))
# disp.plot(ax=ax)
# ax.set_title("Confusion Matrix for Critical Probability of 0.25")
# plt.tight_layout()
# plt.show()
```