⚕️ 🩺 💉 Chronic Liver Disease 💉 🩺 ⚕️

🎯 Workflow of the Project 🎯

1. 🗂️ **Importing Libraries: - To perform Data Manipulation,Visualization & Model Building.**

2. ⏳ **Loading Dataset: - Load the dataset into a suitable data structure using pandas.**

3. 📑 **Exploration of Dataset: - Generate basic informations about the data.**

4. 🧽 **Data Cleaning: - garbage cleaning, removing duplicate, and hadling missing values and treating outliers**

5. 📊 **Exploatory Data Analysis: - To identify trends, patterns, and relationships among the variabels.**

6. ⚙️ **Data Preprocessing: - To transform data for creating more accurate & robust model.**

7. 🎯 **Model building:- To build predictive models, using various algorithms.**

8. ⚡ **Model evaluation: - To analyze the Model performance using metrics.**

9. 📝 **Conclusion: - Conclude the project by summarizing the key findings.**

🗂️ Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

⏳ Loading Dataset

In [2]:
```
dataset = 'indian_liver_patient.csv'
df = pd.read_csv(dataset)
df.head()
```

Out[2]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Pr
0	65	Female	0.7	0.1	187	16	18	
1	62	Male	10.9	5.5	699	64	100	
2	62	Male	7.3	4.1	490	60	68	
3	58	Male	1.0	0.4	182	14	20	
4	72	Male	3.9	2.0	195	27	59	

⏳ Exoloring Dataset

In [3]:
```
df.shape
```

Out[3]:
```
(583, 11)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Age                         583 non-null    int64  
 1   Gender                      583 non-null    object 
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64  
 5   Alamine_Aminotransferase    583 non-null    int64  
 6   Aspartate_Aminotransferase  583 non-null    int64  
 7   Total_Protiens              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  579 non-null    float64
 10  Dataset                     583 non-null    int64  
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

In [5]:
```
from summarytools import dfSummary
dfSummary(df)
```

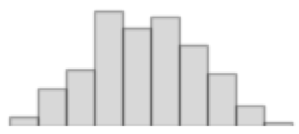
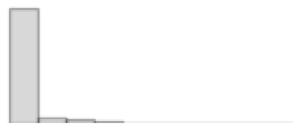
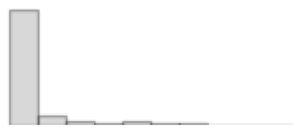
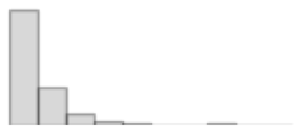
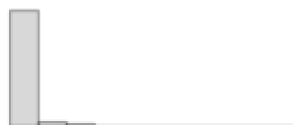
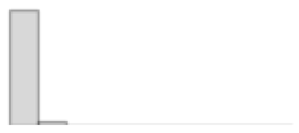
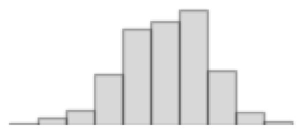
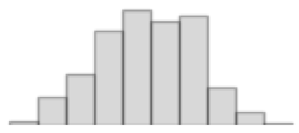
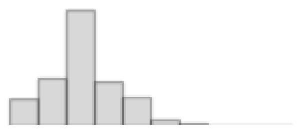
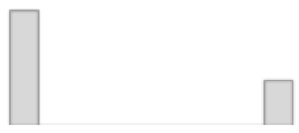
Out[5]:

Data Frame Summary

df

Dimensions: 583 x 11

Duplicates: 13

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
1	**Age** [int64]	Mean (sd) : 44.7 (16.2) min < med < max: 4.0 < 45.0 < 90.0 IQR (CV) : 25.0 (2.8)	72 distinct values		0 (0.0%)
2	**Gender** [object]	1. Male 2. Female	441 (75.6%) 142 (24.4%)		0 (0.0%)
3	**Total_Bilirubin** [float64]	Mean (sd) : 3.3 (6.2) min < med < max: 0.4 < 1.0 < 75.0 IQR (CV) : 1.8 (0.5)	113 distinct values		0 (0.0%)
4	**Direct_Bilirubin** [float64]	Mean (sd) : 1.5 (2.8) min < med < max: 0.1 < 0.3 < 19.7 IQR (CV) : 1.1 (0.5)	80 distinct values		0 (0.0%)
5	**Alkaline_Phosphot ase** [int64]	Mean (sd) : 290.6 (242.9) min < med < max: 63.0 < 208.0 < 2110.0 IQR (CV) : 122.5 (1.2)	263 distinct values		0 (0.0%)
6	**Alamine_Aminotra nsferase** [int64]	Mean (sd) : 80.7 (182.6) min < med < max: 10.0 < 35.0 < 2000.0 IQR (CV) : 37.5 (0.4)	152 distinct values		0 (0.0%)
7	**Aspartate_Aminotr ansferase** [int64]	Mean (sd) : 109.9 (288.9) min < med < max: 10.0 < 42.0 < 4929.0 IQR (CV) : 62.0 (0.4)	177 distinct values		0 (0.0%)
8	**Total_Protiens** [float64]	Mean (sd) : 6.5 (1.1) min < med < max: 2.7 < 6.6 < 9.6 IQR (CV) : 1.4 (6.0)	58 distinct values		0 (0.0%)
9	**Albumin** [float64]	Mean (sd) : 3.1 (0.8) min < med < max: 0.9 < 3.1 < 5.5 IQR (CV) : 1.2 (3.9)	40 distinct values		0 (0.0%)
10	**Albumin_and_Glob ulin_Ratio** [float64]	Mean (sd) : 0.9 (0.3) min < med < max: 0.3 < 0.9 < 2.8 IQR (CV) : 0.4 (3.0)	69 distinct values		4 (0.7%)
11	**Dataset** [int64]	Mean (sd) : 1.3 (0.5) min < med < max: 1.0 < 1.0 < 2.0 IQR (CV) : 1.0 (2.8)	2 distinct values		0 (0.0%)

Cleaning of dataset

Checking and Removing Duplicates

In [6]:
```
# Remove duplicates

def drop_dup(df):
    if df.duplicated().any() == True:
        print('The total duplicate row before removing duplicate:', df.duplicated().sum())
        df.drop_duplicates(inplace=True , keep = 'last')  # Remove duplicates
        df = df.reset_index(drop=True) #Reset the index
        print('The total duplicate row after removing duplicate:', df.duplicated().sum(), ' \nshape of datase
    else:
        return 'No duplicate entries'
drop_dup(df)
```

```
The total duplicate row before removing duplicate: 13
The total duplicate row after removing duplicate: 0
shape of dataset after removing duplicate columns : (570, 11)
```

Checking null values

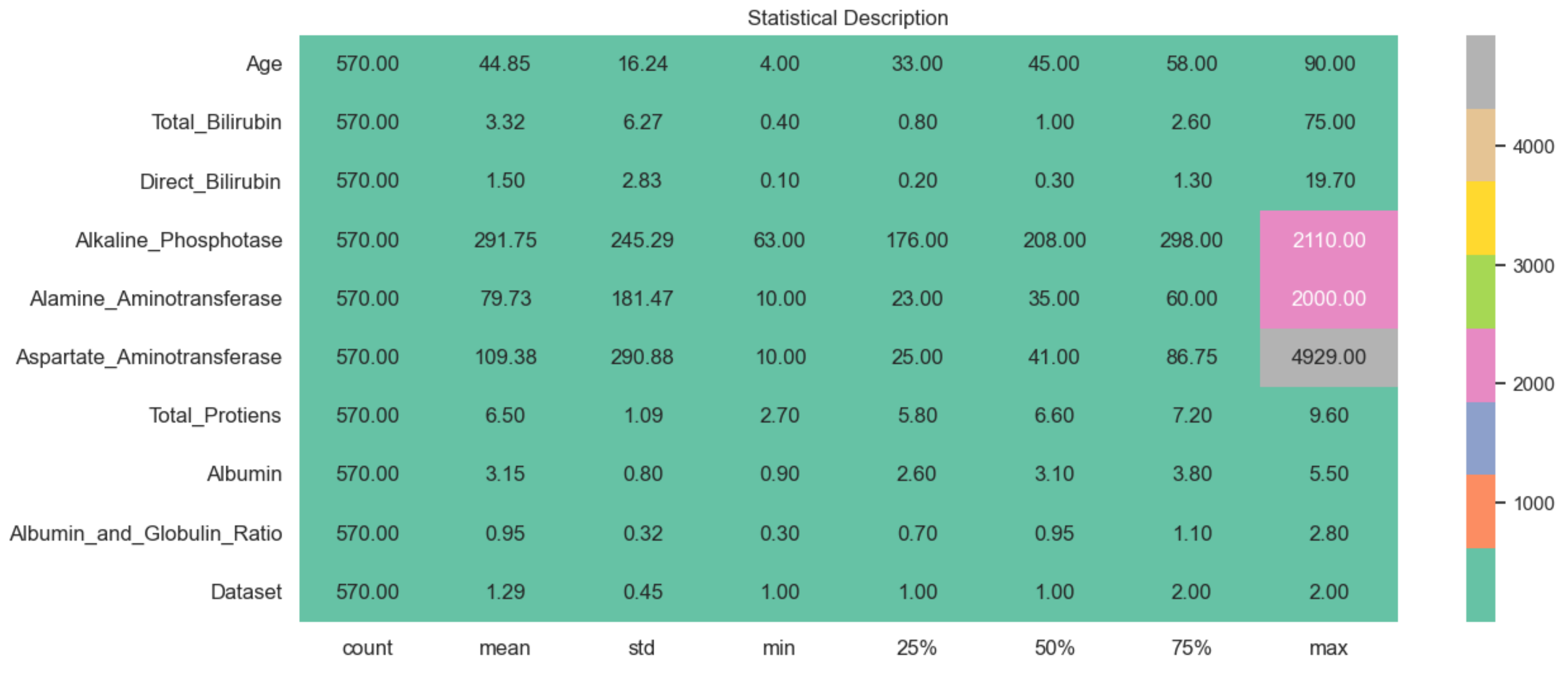
In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Age                           0
Gender                        0
Total_Bilirubin               0
Direct_Bilirubin              0
Alkaline_Phosphotase          0
Alamine_Aminotransferase      0
Aspartate_Aminotransferase    0
Total_Protiens                0
Albumin                       0
Albumin_and_Globulin_Ratio    4
Dataset                       0
dtype: int64
```

In [8]:
```
df['Albumin_and_Globulin_Ratio'].fillna(df['Albumin_and_Globulin_Ratio'].median(), inplace = True)
```

Descriptive Statistics

In [9]:
```
desc=df.describe().T
def descriptive_stats(df):


    plt.figure(figsize=(14,6))
    sns.heatmap(df, annot=True, cmap='Set2', fmt=".2f")
    plt.xticks(size = 12)
    plt.yticks(size = 12, rotation = 0)
    plt.title('Statistical Description')
    plt.show()


descriptive_stats(desc)
```


Statistical Description

	count	mean	std	min	25%	50%	75%	max
Age	570.00	44.85	16.24	4.00	33.00	45.00	58.00	90.00
Total_Bilirubin	570.00	3.32	6.27	0.40	0.80	1.00	2.60	75.00
Direct_Bilirubin	570.00	1.50	2.83	0.10	0.20	0.30	1.30	19.70
Alkaline_Phosphotase	570.00	291.75	245.29	63.00	176.00	208.00	298.00	2110.00
Alamine_Aminotransferase	570.00	79.73	181.47	10.00	23.00	35.00	60.00	2000.00
Aspartate_Aminotransferase	570.00	109.38	290.88	10.00	25.00	41.00	86.75	4929.00
Total_Protiens	570.00	6.50	1.09	2.70	5.80	6.60	7.20	9.60
Albumin	570.00	3.15	0.80	0.90	2.60	3.10	3.80	5.50
Albumin_and_Globulin_Ratio	570.00	0.95	0.32	0.30	0.70	0.95	1.10	2.80
Dataset	570.00	1.29	0.45	1.00	1.00	1.00	2.00	2.00

In [10]:
```
#!pip install skimpy
```

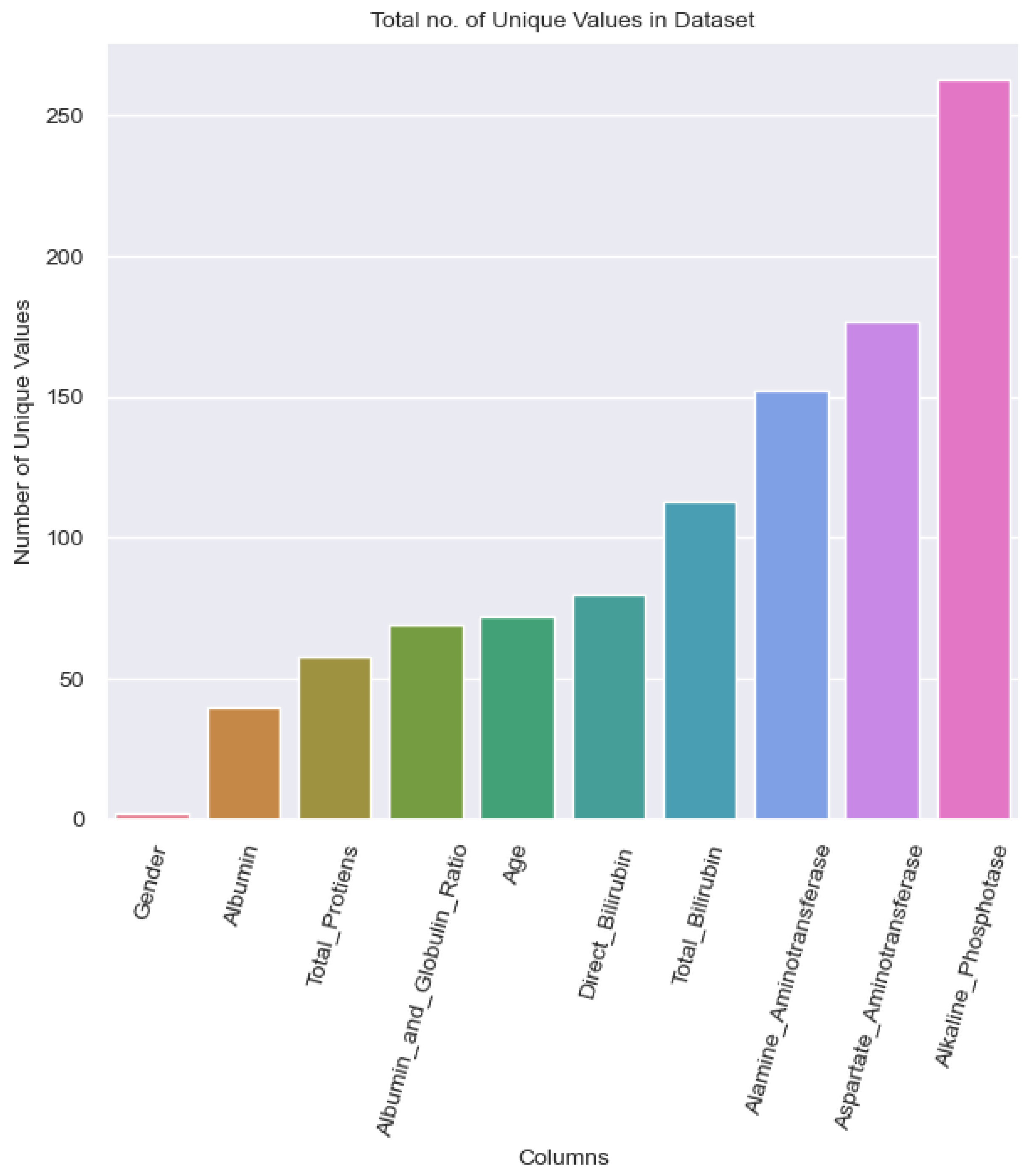
In [11]:
```python
from skimpy import skim
skim(df)
```

```
╭──────────────────────────────────────────── skimpy summary ─────────────────────────────────────────────
│          Data Summary                Data Types
│ ┏━━━━━━━━━━━━━━━━━━━┳━━━━━━━━┓ ┏━━━━━━━━━━━━━┳━━━━━━━┓
│ ┃ dataframe         ┃ Values ┃ ┃ Column Type ┃ Count ┃
│ ┡━━━━━━━━━━━━━━━━━━━╇━━━━━━━━┩ ┡━━━━━━━━━━━━━╇━━━━━━━┩
│ │ Number of rows    │ 570    │ │ int32       │ 5     │
│ │ Number of columns │ 11     │ │ float64     │ 5     │
│ └───────────────────┴────────┘ │ string      │ 1     │
│                                └─────────────┴───────┘
│                                                 number
│ ┏━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━┳━━━━━━━┳━━━━━━━┳━━━━━━┳━━━━━━┳━━━━━━┳━━━━━━┳━━━━━━┳━━━━━━
│ ┃ column_name              ┃ NA  ┃ NA %  ┃ mean  ┃ sd   ┃ p0   ┃ p25  ┃ p75  ┃ p100 ┃ hist
│ ┡━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━╇━━━━━━━╇━━━━━━━╇━━━━━━╇━━━━━━╇━━━━━━╇━━━━━━╇━━━━━━╇━━━━━━
│ │ Age                      │   0 │     0 │    45 │   16 │    4 │   33 │   58 │   90 │  ▁▁▃▃▇
│ │ Total_Bilirubin          │   0 │     0 │   3.3 │  6.3 │  0.4 │  0.8 │  2.6 │   75 │    ▇
│ │ Direct_Bilirubin         │   0 │     0 │   1.5 │  2.8 │  0.1 │  0.2 │  1.3 │   20 │    ▇
│ │ Alkaline_Phosphotase     │   0 │     0 │   290 │  250 │   63 │  180 │  300 │ 2100 │   ▇▁
│ │ Alamine_Aminotransferase │   0 │     0 │    80 │  180 │   10 │   23 │   60 │ 2000 │    ▇
│ │ Aspartate_Aminotransfera │   0 │     0 │   110 │  290 │   10 │   25 │   87 │ 4900 │    ▇
│ │ Total_Protiens           │   0 │     0 │   6.5 │  1.1 │  2.7 │  5.8 │  7.2 │  9.6 │   ▁▅▇
│ │ Albumin                  │   0 │     0 │   3.1 │  0.8 │  0.9 │  2.6 │  3.8 │  5.5 │  ▁▃▇▆
│ │ Albumin_and_Globulin_Rat │   0 │     0 │  0.95 │ 0.32 │  0.3 │  0.7 │  1.1 │  2.8 │  ▃▇▂▁
│ │ Dataset                  │   0 │     0 │   1.3 │ 0.45 │    1 │    1 │    2 │    2 │ ▇    ▃
│ └──────────────────────────┴─────┴───────┴───────┴──────┴──────┴──────┴──────┴──────┴──────
│                                                 string
│ ┏━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━┳━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━
│ ┃ column_name            ┃ NA     ┃ NA %     ┃ words per row           ┃ total words
│ ┡━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━╇━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━
│ │ Gender                 │      0 │        0 │                       1 │                    5
│ └────────────────────────┴────────┴──────────┴─────────────────────────┴──────────────────────
╰────────────────────────────────────────────────── End ──────────────────────────────────────────────────
```

Checking Unique values of different features

In [12]:
```
# Count the number of unique values in each column
def check_unquie_count(df):
    unique_counts = df.nunique().sort_values()
    print('=='*30)
    print('  '*10, 'Total no. of Unique Values')
    print('=='*30)
    print(unique_counts)
    print('=='*30)
# Create a bar plot or count plot of unique values
    #plt.style.use('dark_background')
    plt.figure(figsize=(7, 6))
    sns.barplot(x=unique_counts.index, y=unique_counts.sort_values(),palette='husl' )

    plt.xticks(rotation=75, fontsize= 10)
    plt.yticks( fontsize= 10 )
    plt.xlabel('Columns',fontsize=10)
    plt.ylabel('Number of Unique Values', fontsize=10)
    plt.title('Total no. of Unique Values in Dataset', fontsize=10)

# Display the plot
    plt.show()

check_unquie_count(df.iloc[:,0:-1])
```

```
============================================================
                     Total no. of Unique Values
============================================================
Gender                          2
Albumin                        40
Total_Protiens                 58
Albumin_and_Globulin_Ratio     69
Age                            72
Direct_Bilirubin               80
Total_Bilirubin               113
Alamine_Aminotransferase      152
Aspartate_Aminotransferase    177
Alkaline_Phosphotase          263
dtype: int64
============================================================
```



In [13]:
```
df['Dataset'].replace({2:'Hepatic', 1:'Non-hepatic'},inplace = True)
```

In [14]:
```
df['Dataset'].unique()
```

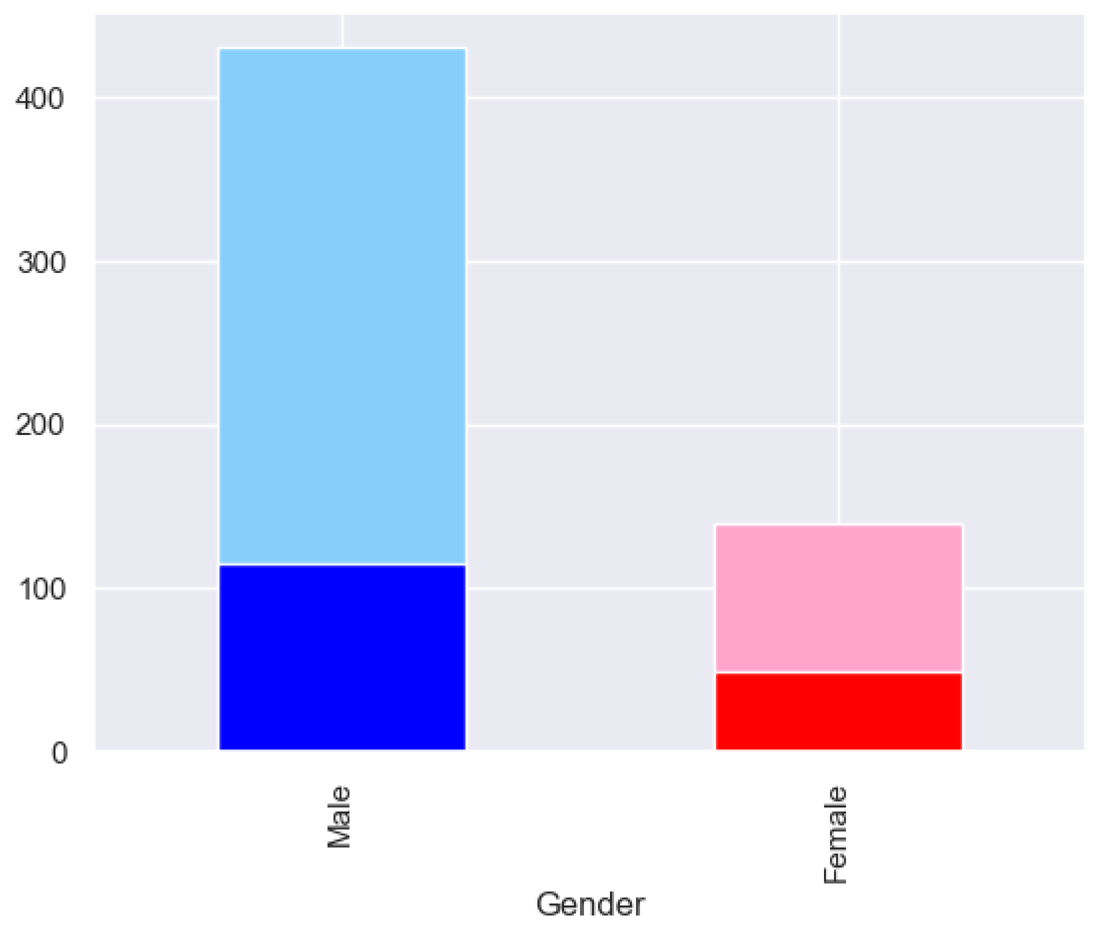
Out[14]:
```
array(['Non-hepatic', 'Hepatic'], dtype=object)
```

The total count of male and female in the dataset and male and female in the dataset having liver disease:

In [15]:
```
gender_count = df['Gender'].value_counts()
hepatic_gender_count =df[(df.Dataset =='Hepatic')]['Gender'].value_counts()
print('==='*30)
print(f'total count of male and female  :{gender_count}')
print('---'*30)
print(f'Count of male and female in the dataset having liver disease : {hepatic_gender_count}')
print('==='*30)

gender_count.plot( kind='bar',  color=['#87CEFA','#FFA4CB', ])

hepatic_gender_count.plot( kind='bar',  color=[ 'blue','red', ])
```

```
==========================================================================================
total count of male and female  :Gender
Male      430
Female    140
Name: count, dtype: int64
------------------------------------------------------------------------------------------
Count of male and female in the dataset having liver disease : Gender
Male      115
Female     49
Name: count, dtype: int64
==========================================================================================
```

Out[15]:
```
<Axes: xlabel='Gender'>
```



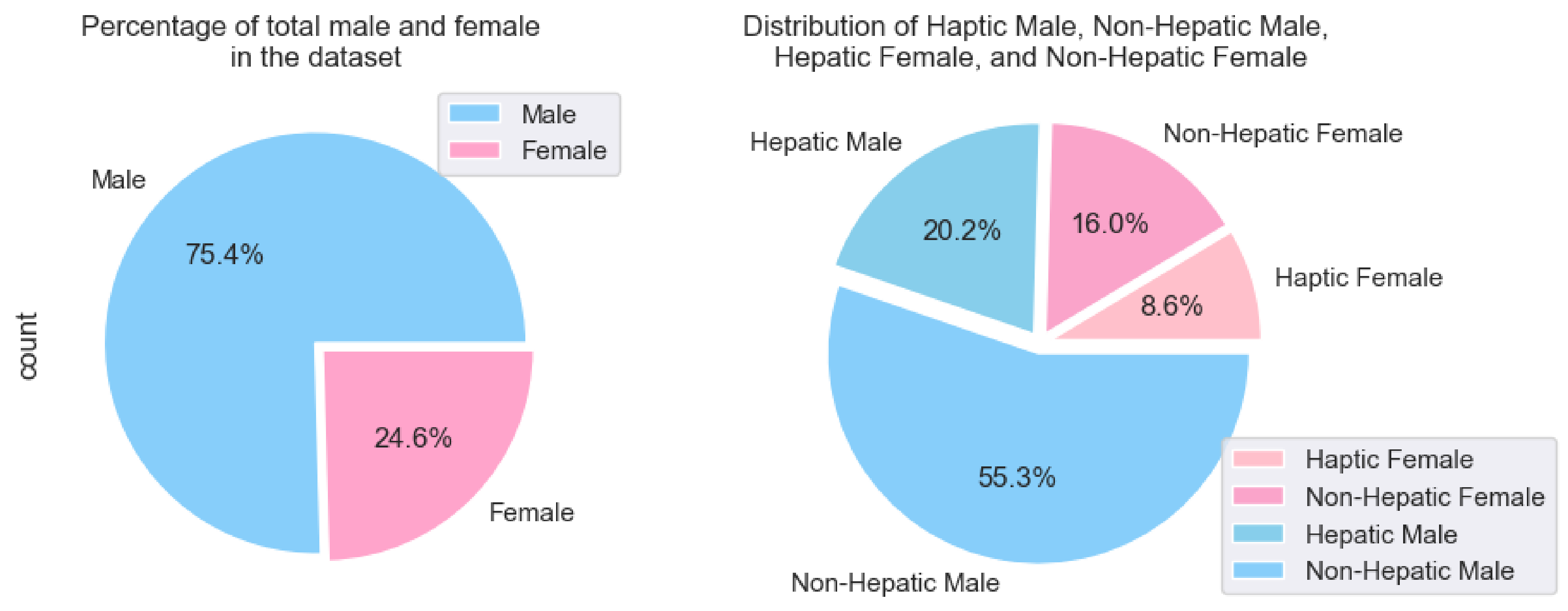
The total count of male and female in the dataset and having liver disease:

In [16]:
```
grouped_gender_data = df.groupby(['Gender', 'Dataset'])[['Gender', 'Dataset']].value_counts()#.size()#.reset_
print(grouped_gender_data)

#df.reset_index??
```

```
Gender  Dataset
Female  Hepatic         49
        Non-hepatic     91
Male    Hepatic        115
        Non-hepatic    315
Name: count, dtype: int64
```

In [17]:
```
total_gender =df['Gender'].value_counts()*100/len(df)
hepatic_gender= df[(df.Dataset =='Hepatic')]['Gender'].value_counts()*100/len(df)
print('==='*30)
print(f'Percentage of  male and female in the dataset :{total_gender.round(2)}%')
print('---'*30)
print(f'Percentage of total male and female in the dataset :{hepatic_gender.round(2)}')

print('==='*30)
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
df['Gender'].value_counts().plot(
    kind='pie',
    explode=[0, 0.05],     # Explode the second category to emphasize it
    autopct='%1.1f%%',         # Display percentage with one decimal place
    #shadow=True,               # Add a shadow effect to the chart
    colors=[ '#87CEFA','#FFA4CB', ]  # Custom colors for the pie chart
)
plt.title('Percentage of total male and female \nin the dataset')
plt.legend(loc='upper right',bbox_to_anchor=(1.1, 1.0))
plt.subplot(1,2,2)
data = grouped_gender_data#grouped_gender_data['Count'].tolist()
data
# Labels for the pie chart
labels = ['Haptic Female', 'Non-Hepatic Female', 'Hepatic Male', 'Non-Hepatic Male']

# Light shade colors for the pie chart
colors = ['pink', '#FAA4CA','skyblue', '#87CEFA', ]

# Plotting the pie chart
plt.pie(data, labels=labels,explode=[0.05,0.05, 0.05,0.05], colors=colors, autopct='%1.1f%%')
plt.title('Distribution of Haptic Male, Non-Hepatic Male, \nHepatic Female, and Non-Hepatic Female')
plt.legend(loc='lower right',bbox_to_anchor=(1.5, 0.0))
plt.show()
plt.tight_layout()
```

```
==========================================================================================
Percentage of  male and female in the dataset :Gender
Male      75.44
Female    24.56
Name: count, dtype: float64%
------------------------------------------------------------------------------------------
Percentage of total male and female in the dataset :Gender
Male      20.18
Female     8.60
Name: count, dtype: float64
==========================================================================================
```



```
<Figure size 640x480 with 0 Axes>
```

- In the dataset there are 75% male and 24% female.

- 8.6% female and 20.2% male of this dataset are affected with liver disease.

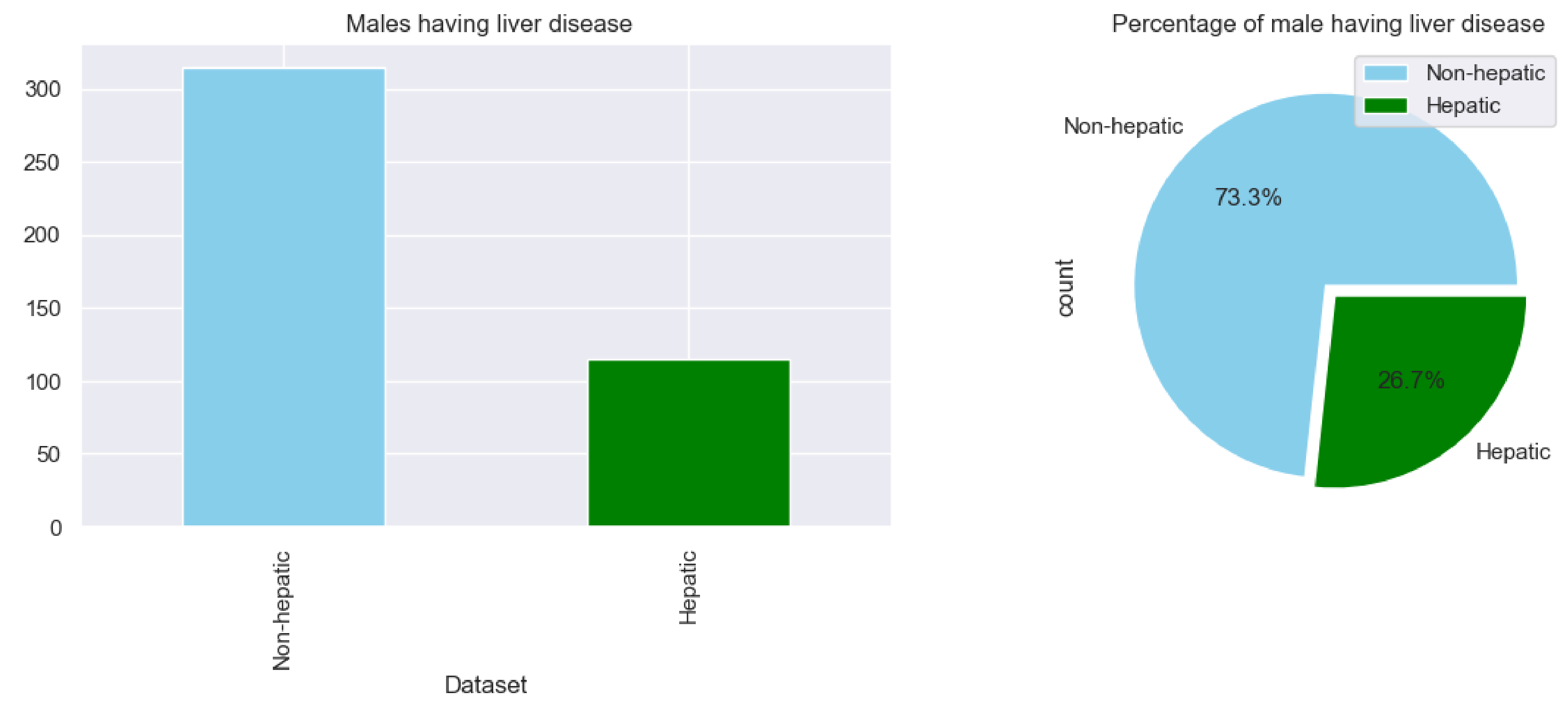
In [ ]:

Details of male affected and not affected with liver disease

In [18]:
```
print(df[(df.Gender =='Male')]['Dataset'].value_counts())
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
df[(df.Gender =='Male') ]['Dataset'].value_counts().plot( kind='bar',  color=['skyblue',  'green'])

plt.title(' Males having liver disease')
plt.subplot(1,2,2)
df[(df.Gender =='Male')]['Dataset'].value_counts().plot(
    kind='pie',
    explode=[0, 0.07],     # Explode the second category to emphasize it
    autopct='%1.1f%%',         # Display percentage with one decimal place
    #shadow=True,               # Add a shadow effect to the chart
    colors=['skyblue',  'green']  # Custom colors for the pie chart
)
plt.legend(loc='upper right')
plt.tight_layout()
plt.title(' Percentage of male having liver disease')
```

```
Dataset
Non-hepatic    315
Hepatic        115
Name: count, dtype: int64
```

Out[18]:
```
Text(0.5, 1.0, ' Percentage of male having liver disease')
```



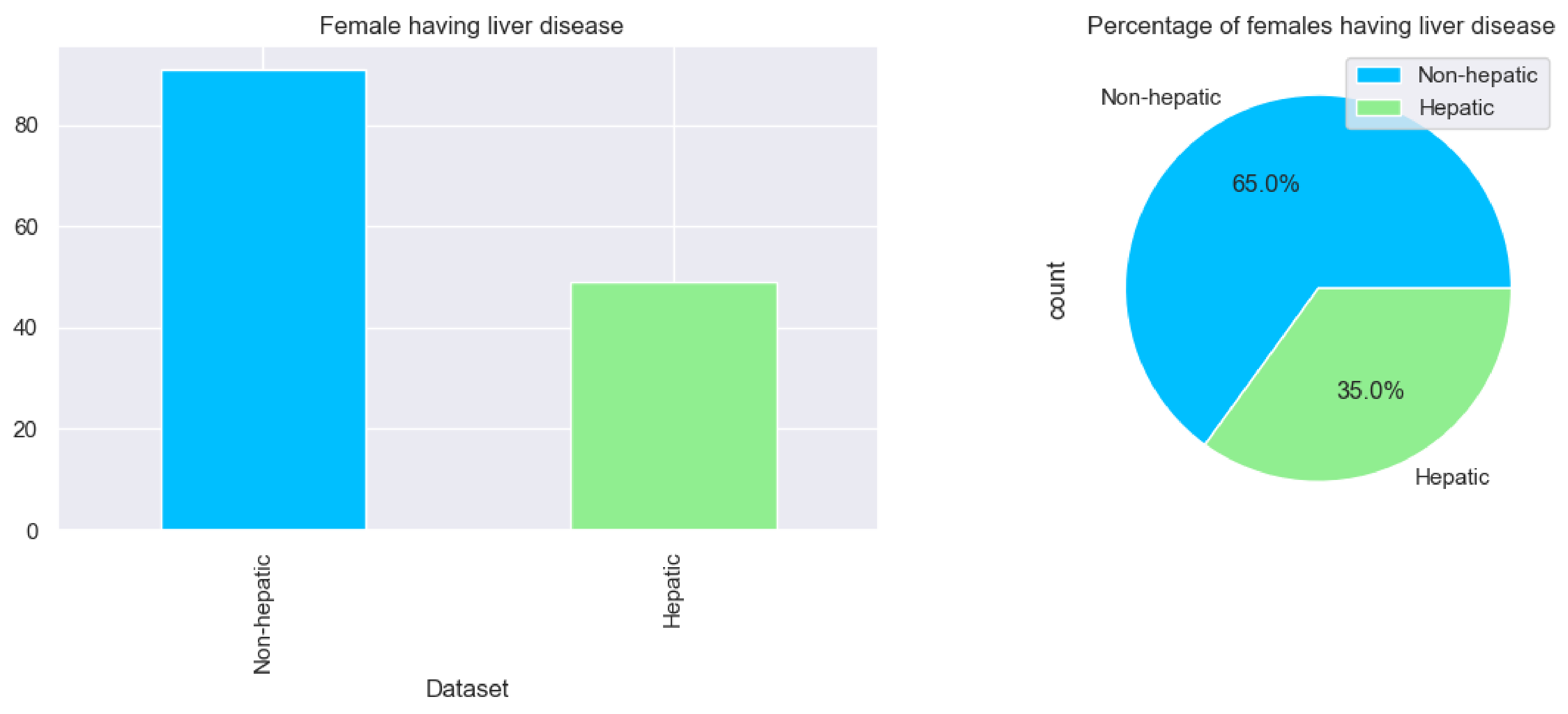
27% of males are diagonised with liver disease.

Details of female affected and not affected with liver disease

In [19]:
```
print(df[(df.Gender =='Female')]['Dataset'].value_counts())
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
df[(df.Gender =='Female') ]['Dataset'].value_counts().plot( kind='bar',  color=['deepskyblue',  'lightgreen']
plt.title(' Female having liver disease')
plt.subplot(1,2,2)
df[(df.Gender =='Female')]['Dataset'].value_counts().plot(
    kind='pie',
    #explode=[0.1, 0.1,0,0, 0, 0],     # Explode the second category to emphasize it
    autopct='%1.1f%%',         # Display percentage with one decimal place
    #shadow=True,               # Add a shadow effect to the chart
    colors=['deepskyblue',  'lightgreen']  # Custom colors for the pie chart
)
plt.legend(loc='upper right')
plt.tight_layout()
plt.title(' Percentage of females having liver disease')
```

```
Dataset
Non-hepatic    91
Hepatic        49
Name: count, dtype: int64
```

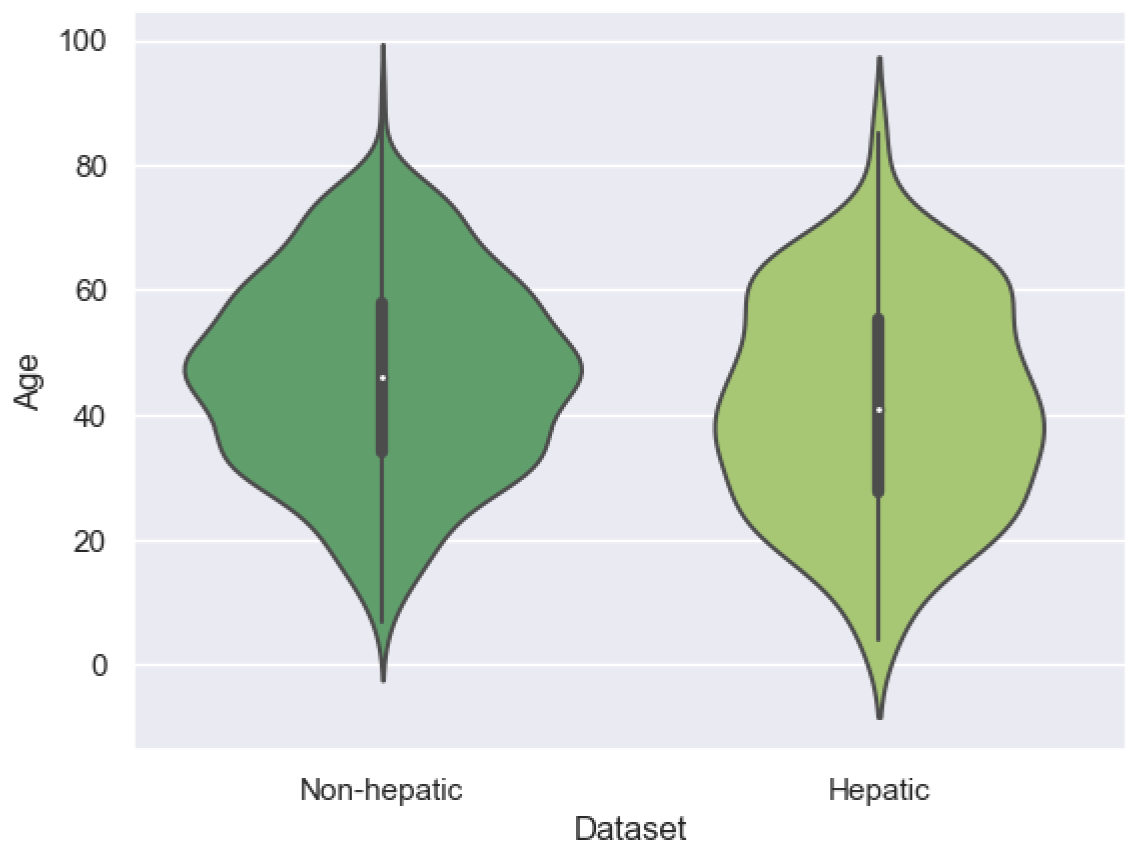
Out[19]:
```
Text(0.5, 1.0, ' Percentage of females having liver disease')
```



35% of females are diagonised with liver disease.

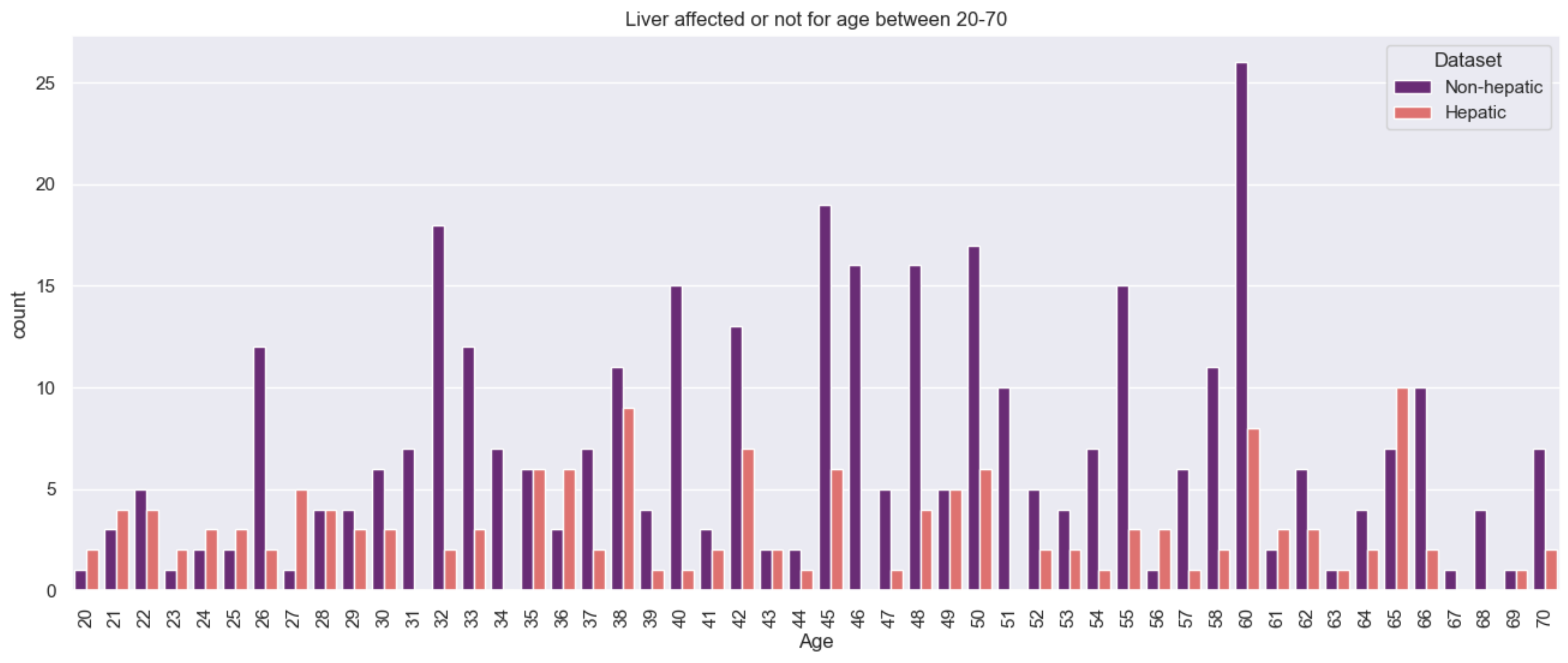
In [20]:
```
sns.violinplot(x='Dataset', y='Age', data=df,palette='summer',)
```

Out[20]:
```
<Axes: xlabel='Dataset', ylabel='Age'>
```

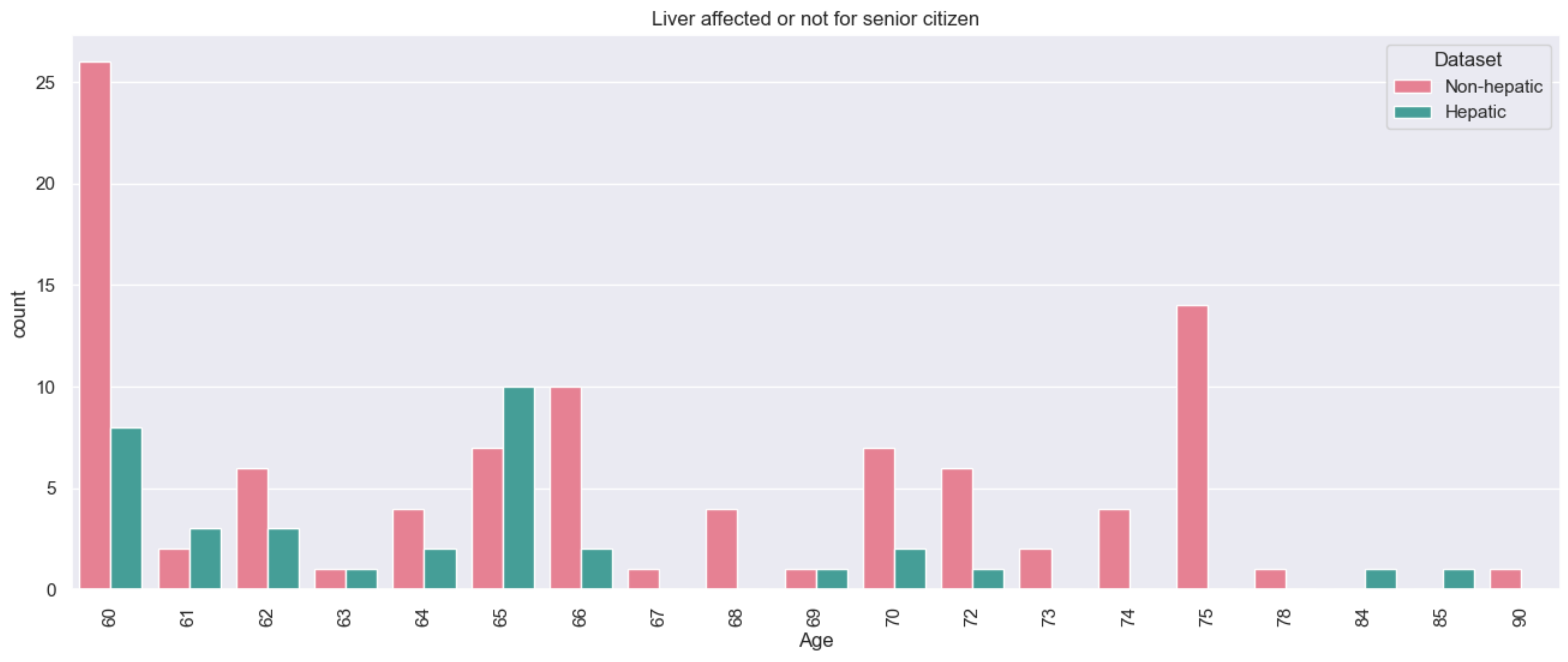


Liverpatitents are mostly in the range 20 to 70

In [21]:
```
Liver_age = df[(df['Age']<=70) & (df['Age']>=20 )][['Age','Dataset']]
plt.figure(figsize=(16,6))
plt.xticks(rotation = 90)
ax = sns.countplot(x= Liver_age['Age'], hue= Liver_age['Dataset'] , palette ='magma')
ax.set_title('Liver affected or not for age between 20-70')
plt.show()
```

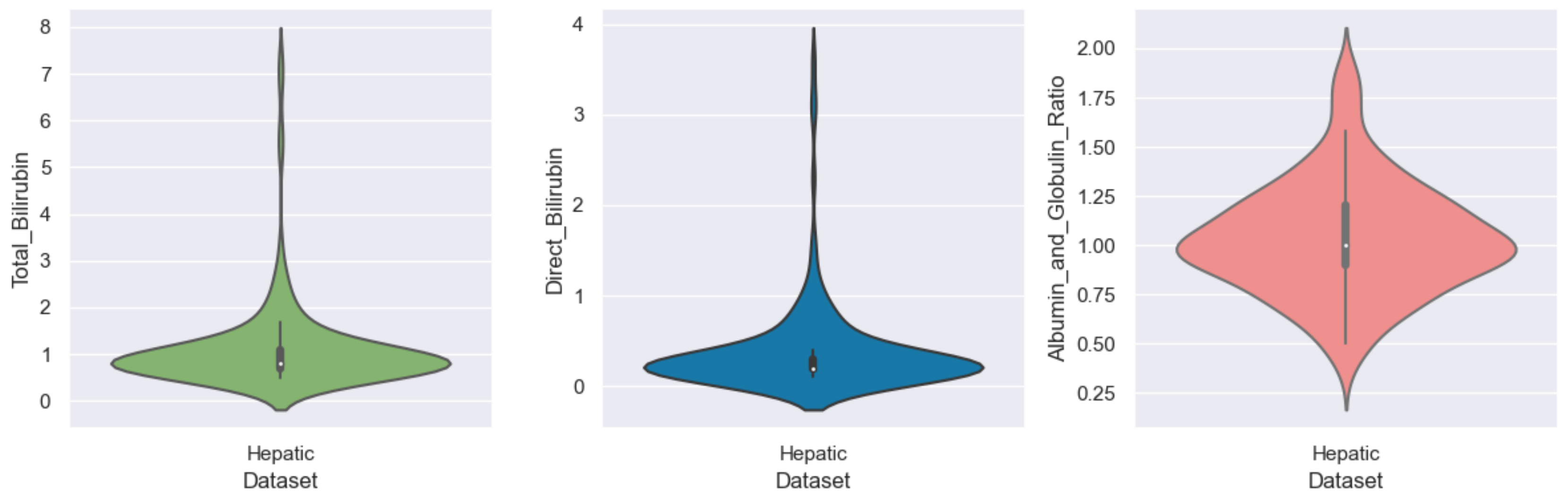


In [22]:
```
Liver_sr = df[(df['Age']>=60)][['Age','Dataset']]
plt.figure(figsize=(16,6))
plt.xticks(rotation = 90)
ax = sns.countplot(x= Liver_sr['Age'] , hue= Liver_sr['Dataset'], palette ='husl')
ax.set_title('Liver affected or not for senior citizen')
plt.show()
```



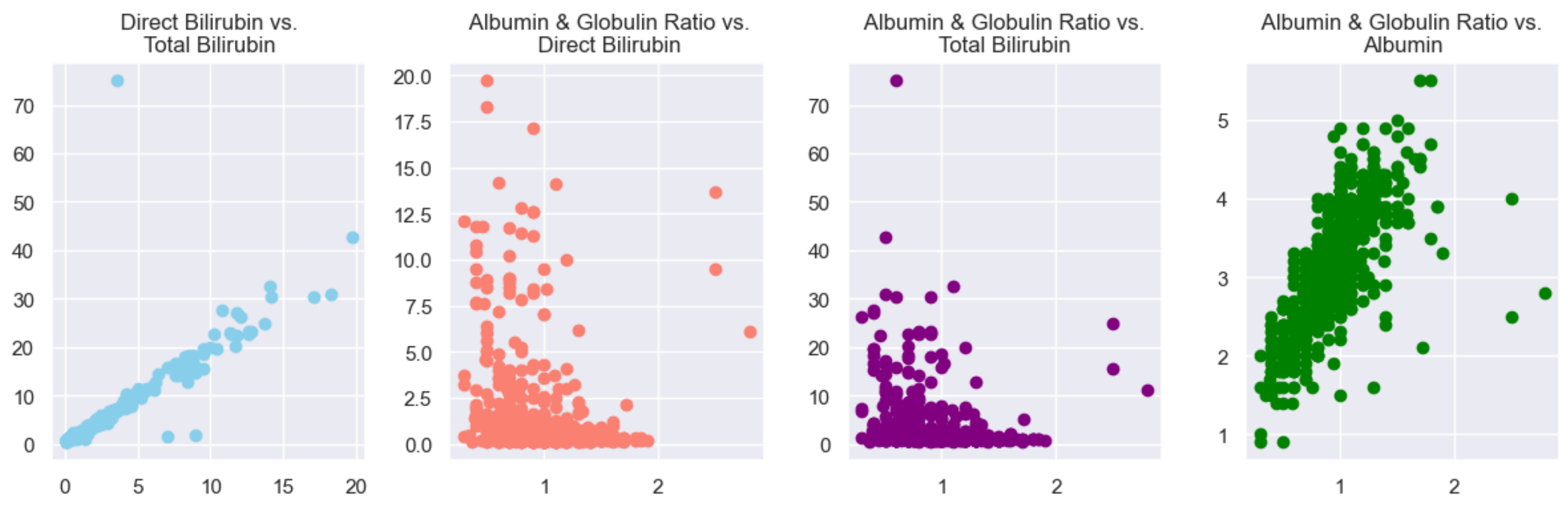
Distribution of `Total_Bilirubin` , `Direct_Bilirubin` and `Albumin_and_Globulin_Ratio` for hepatic patitent

In [23]:
```
plt.figure(figsize=(12,4))
plt.subplot(1,3,1)
sns.violinplot(x='Dataset', y='Total_Bilirubin', data=df[df.Dataset =='Hepatic'],palette='summer',)
plt.subplot(1,3,2)
sns.violinplot(x='Dataset', y='Direct_Bilirubin', data=df[df.Dataset =='Hepatic'],palette='winter',)
plt.subplot(1,3,3)
sns.violinplot(x='Dataset', y='Albumin_and_Globulin_Ratio', data=df[df.Dataset =='Hepatic'],palette='spring',
plt.tight_layout()
```

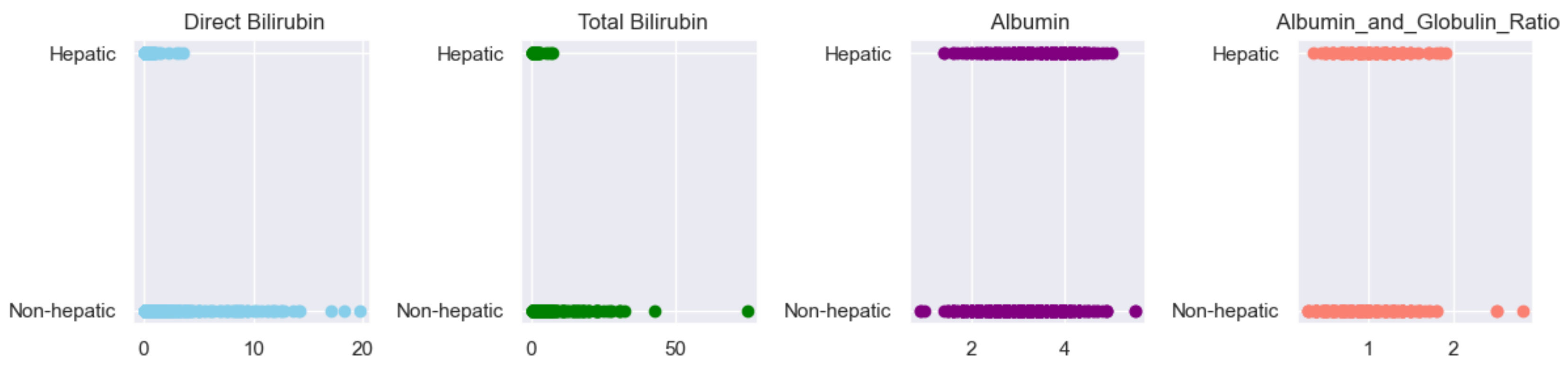


Bivariate Analysis of `Total_Bilirubin` , `Direct_Bilirubin` , `Albumin and Albumin_and_Globulin_Ratio`

In [24]:
```
plt.figure(figsize=(12,4))
plt.subplot(1,4,1)
plt.scatter(df['Direct_Bilirubin'], df['Total_Bilirubin'],color='skyblue')
plt.title(' Direct Bilirubin vs. \nTotal Bilirubin')
plt.subplot(1,4,2)
plt.scatter(df['Albumin_and_Globulin_Ratio'],df['Direct_Bilirubin'], color='#FA8072')
plt.title(' Albumin & Globulin Ratio vs.\n Direct Bilirubin')
plt.subplot(1,4,3)
plt.scatter(df['Albumin_and_Globulin_Ratio'], df['Total_Bilirubin'], color='purple')
# plt.xlabel('Direct Bilirubin')
# plt.ylabel('Total Bilirubin')
plt.title(' Albumin & Globulin Ratio vs. \nTotal Bilirubin')
plt.subplot(1,4,4)
plt.scatter(df['Albumin_and_Globulin_Ratio'], df['Albumin'], color='green')
# plt.xlabel('Direct Bilirubin')
# plt.ylabel('Total Bilirubin')
plt.title(' Albumin & Globulin Ratio vs. \nAlbumin')
plt.tight_layout()
plt.show()
```



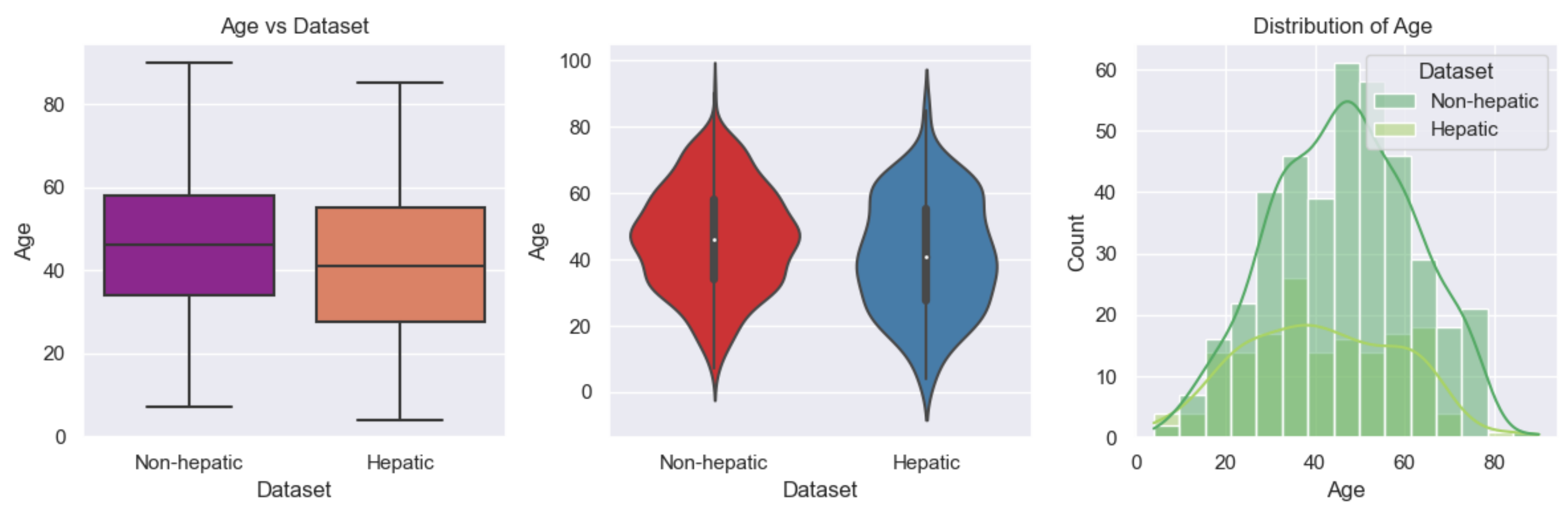
In [25]:
```
plt.figure(figsize=(12,3))
plt.subplot(1,4,1)
plt.scatter(df['Direct_Bilirubin'], df['Dataset'],color='skyblue')
plt.title(' Direct Bilirubin')
plt.subplot(1,4,2)
plt.scatter(df['Total_Bilirubin'], df['Dataset'], color='green')
plt.title(' Total Bilirubin')
plt.subplot(1,4,3)
plt.scatter(df['Albumin'], df['Dataset'], color='purple')
plt.title(' Albumin')
plt.subplot(1,4,4)
plt.scatter(df['Albumin_and_Globulin_Ratio'],df['Dataset'], color='#FA8072')
plt.title(' Albumin_and_Globulin_Ratio')

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

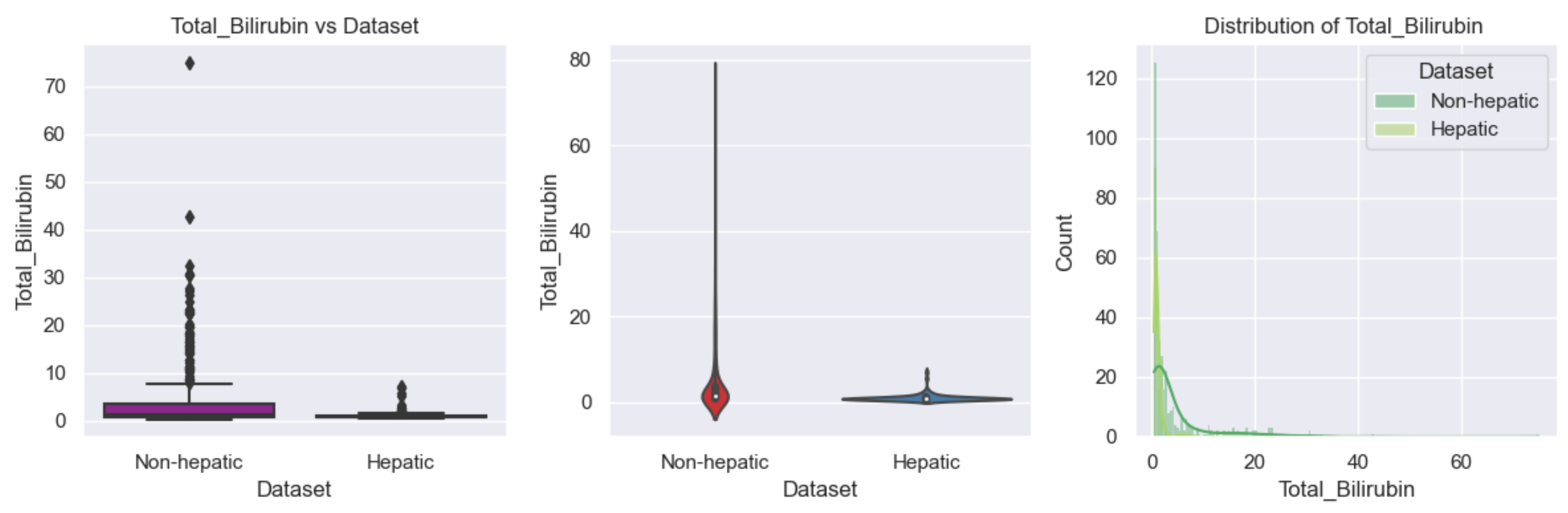


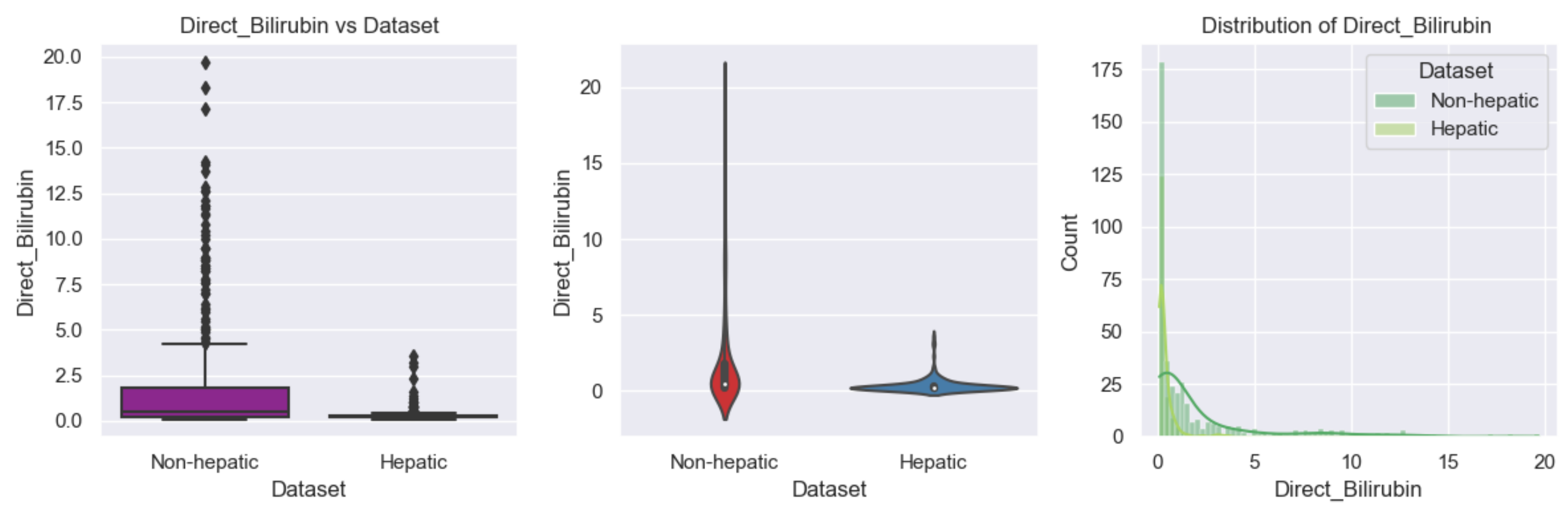
Distribution

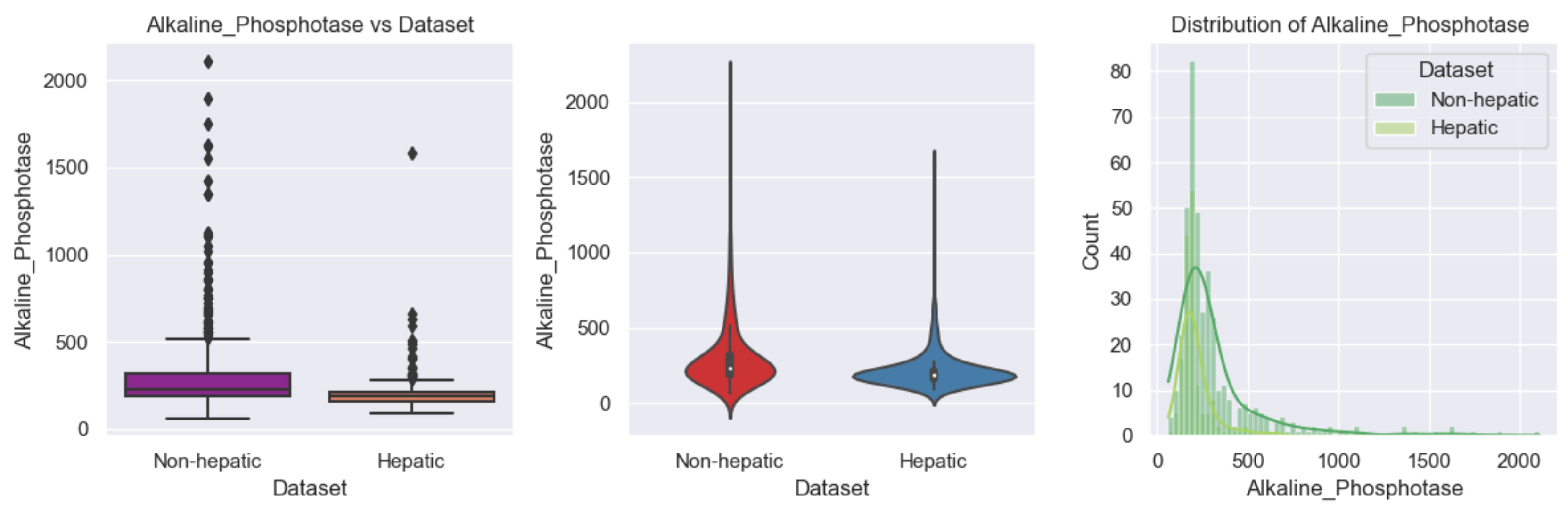
In [26]:
```
def num_dist(column, target= 'Dataset'):
    plt.figure(figsize=(12,4))
    plt.subplot(1,3,1)
    sns.boxplot(x=target,y=column, data=df, palette="plasma")
    plt.title(f"{column.title()} vs Dataset")

    plt.subplot(1,3,2)
    sns.violinplot(x=target, y=column, data=df,palette='Set1',)

    plt.subplot(1,3,3)
    sns.histplot(data=df,x=column,kde=True,hue= target, palette="summer")
    plt.title(f"Distribution of {column.title()} ")
    plt.tight_layout()
    plt.show()
```

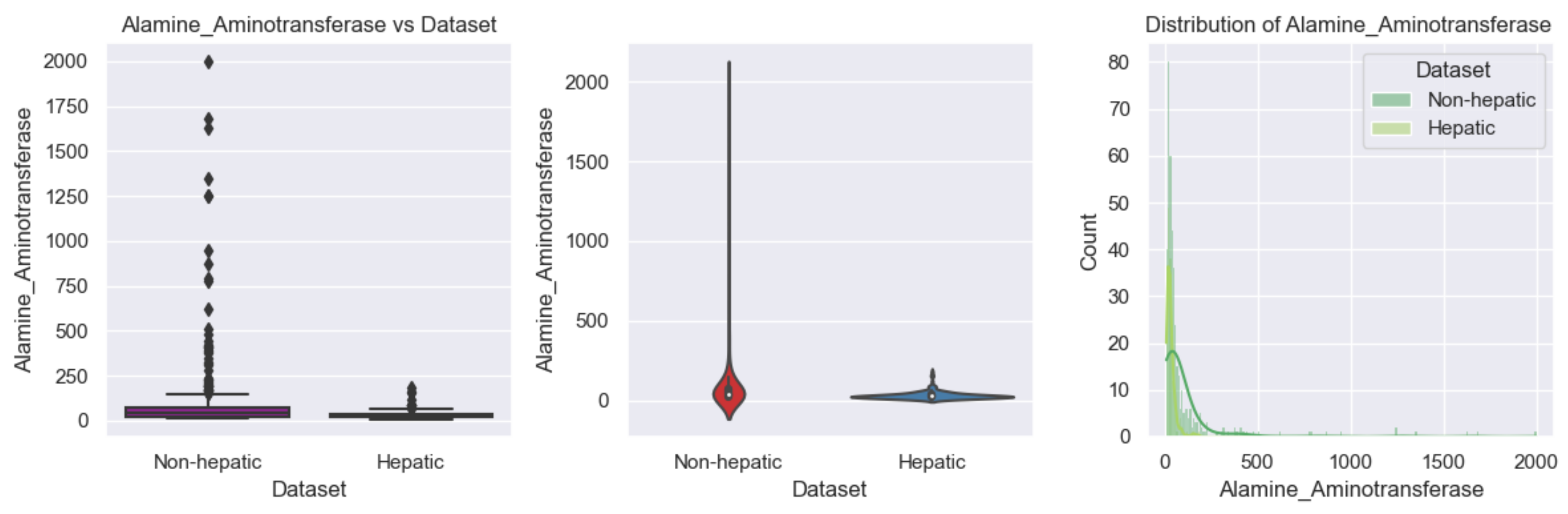
In [27]:
```
num_col = df.select_dtypes(include='number').columns
for col in num_col[:-1]:
    num_dist(col)
```

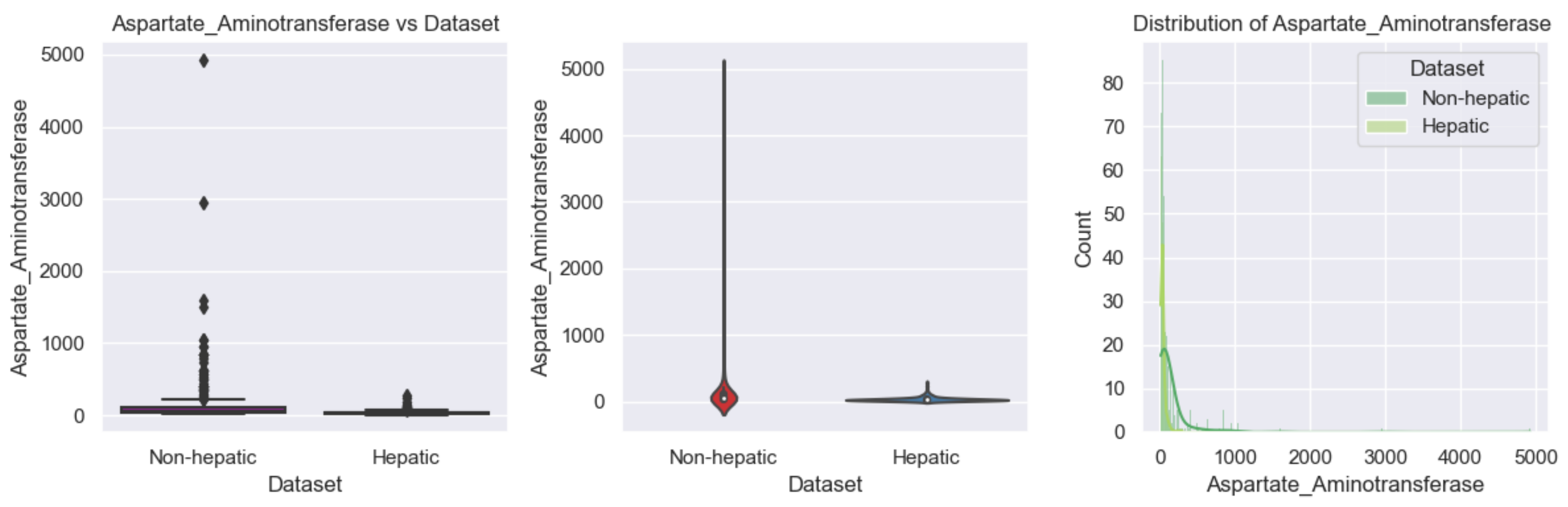


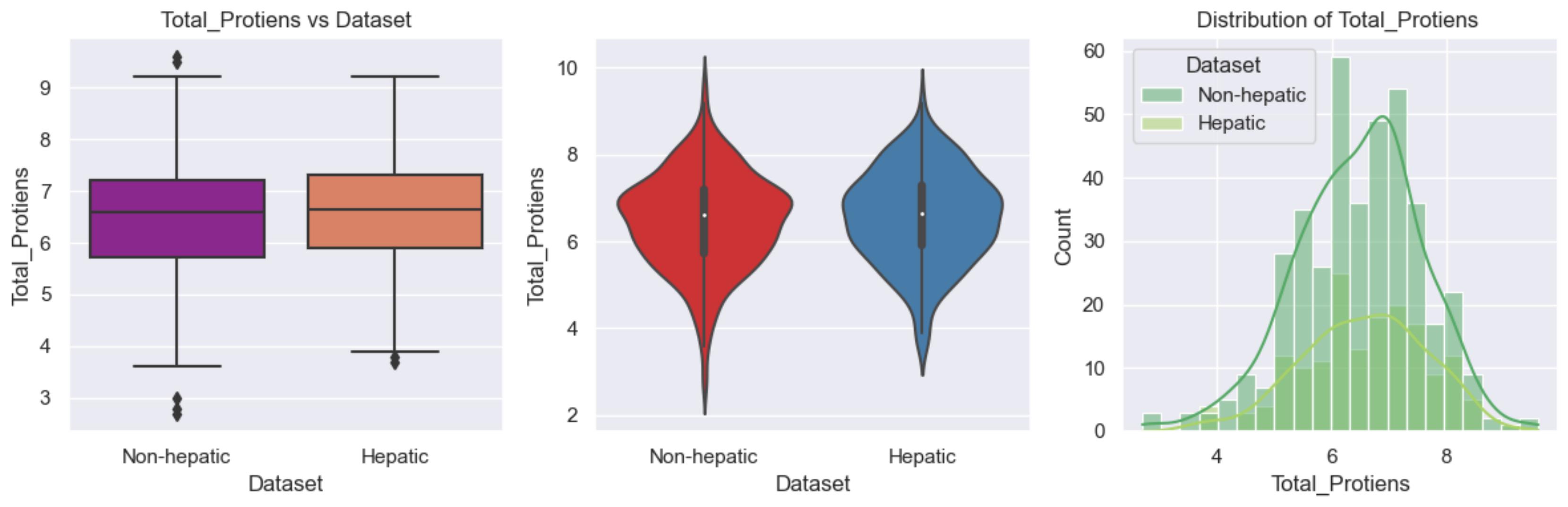


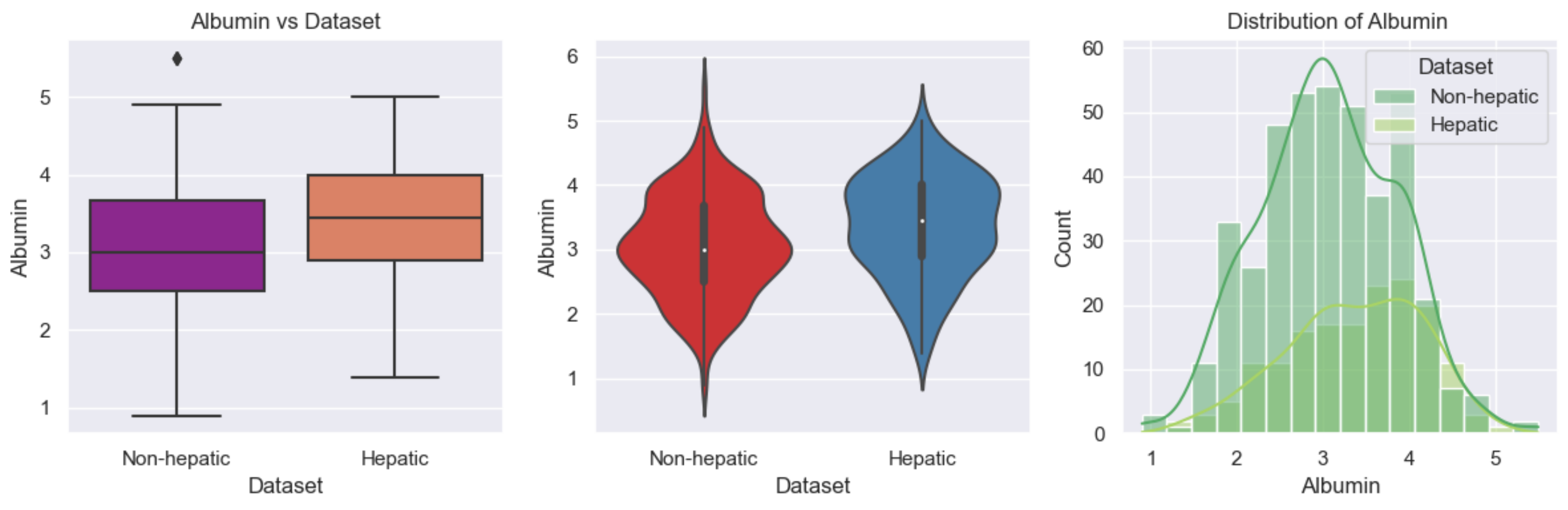






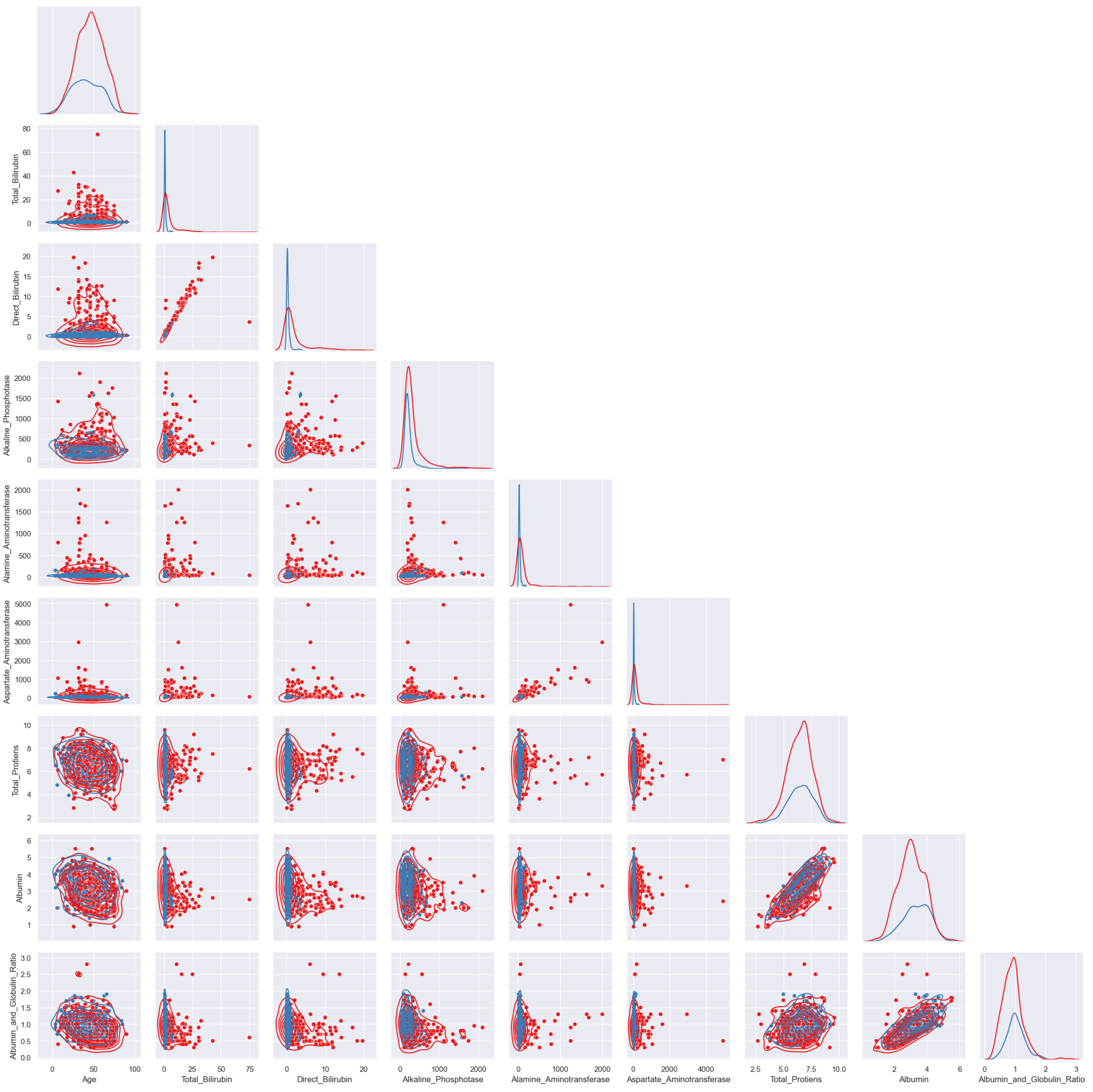






Outlier Treatment : Not performed as it is a clinical dataset

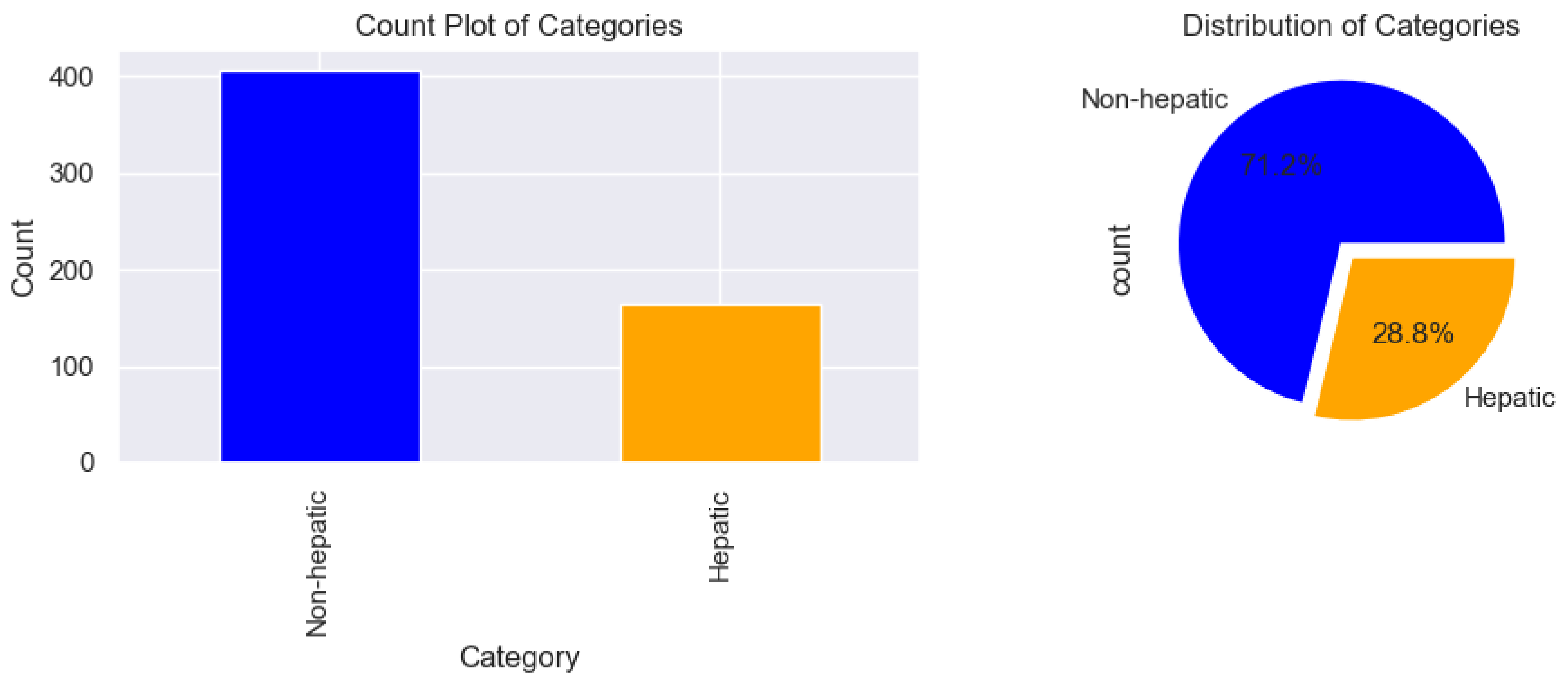
In [28]:
```
g = sns.PairGrid(df, hue='Dataset',palette ='Set1' , diag_sharey=False,corner =True)
g.map_lower(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.kdeplot)
plt.show()
```



📊 Checking Imbalance between the different classes

In [29]:
```
value_counts =df.Dataset.value_counts()
print(value_counts)


fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# Plot the bar plot on the first subplot (axes[0])
value_counts.plot(kind='bar', ax=axes[0],color=['blue', 'orange'])
axes[0].set_xlabel('Category')
axes[0].set_ylabel('Count')
axes[0].set_title('Count Plot of Categories')

plt.subplot(1,2,1)
value_counts.plot(
    kind='pie',
    ax=axes[1],
    explode=[ 0.1,0],     # Explode the second category to emphasize it
    autopct='%1.1f%%',         # Display percentage with one decimal place
    #shadow=True,               # Add a shadow effect to the chart
    colors=['blue', 'orange']  # Custom colors for the pie chart
)
axes[1].set_title('Distribution of Categories')


plt.tight_layout()

plt.show()
```

```
Dataset
Non-hepatic    406
Hepatic        164
Name: count, dtype: int64
```



** Imbalancy in dataset found**

- A **class-imbalance** in the target feature is observed.
- To overcome this imbalancy, oversampling method is used for modeling.

🛠️ 🧱 ⛏️ Model Building Predict Data and Evaluation of Model

Sklearn Libraries

In [30]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay,
from sklearn.datasets import make_classification

from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC
```

Encoding

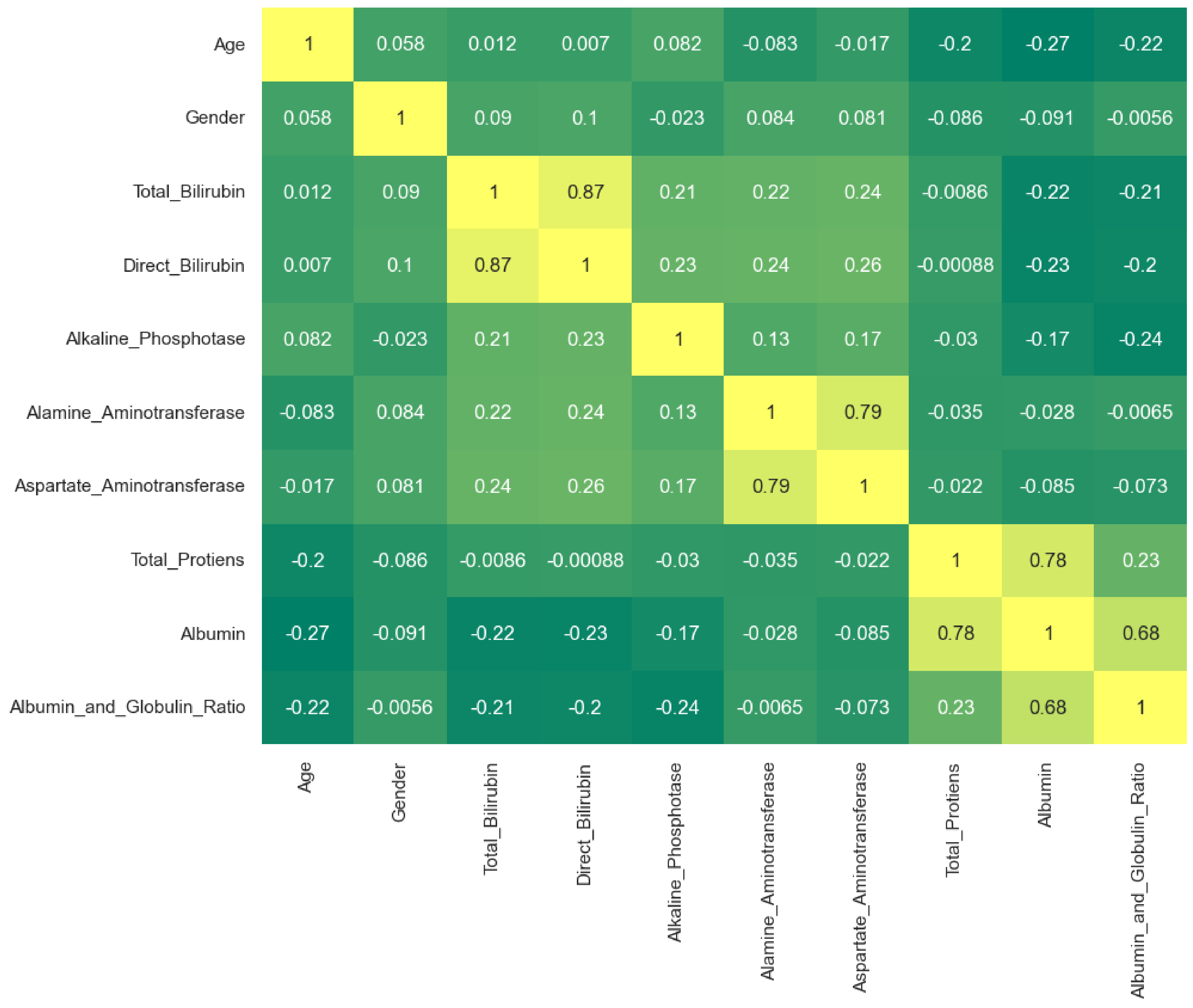
In [31]:
```
df['Gender']=df['Gender'].astype('category')
df['Gender'] =df['Gender'].cat.codes

df['Dataset']=df['Dataset'].astype('category')
df['Dataset'] =df['Dataset'].cat.codes
```

Splitting into target and independent features

In [32]:
```
target = "Dataset"
x = df.drop(columns=target)
y = df[target]
```

Correlation between independent features

In [33]:
```
plt.figure(figsize=(10,8))
corr = x.corr()
sns.heatmap(corr, annot=True, cmap='summer',cbar=False)
plt.show()
```



	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio
Age	1	0.058	0.012	0.007	0.082	-0.083	-0.017	-0.2	-0.27	-0.22
Gender	0.058	1	0.09	0.1	-0.023	0.084	0.081	-0.086	-0.091	-0.0056
Total_Bilirubin	0.012	0.09	1	0.87	0.21	0.22	0.24	-0.0086	-0.22	-0.21
Direct_Bilirubin	0.007	0.1	0.87	1	0.23	0.24	0.26	-0.00088	-0.23	-0.2
Alkaline_Phosphotase	0.082	-0.023	0.21	0.23	1	0.13	0.17	-0.03	-0.17	-0.24
Alamine_Aminotransferase	-0.083	0.084	0.22	0.24	0.13	1	0.79	-0.035	-0.028	-0.0065
Aspartate_Aminotransferase	-0.017	0.081	0.24	0.26	0.17	0.79	1	-0.022	-0.085	-0.073
Total_Protiens	-0.2	-0.086	-0.0086	-0.00088	-0.03	-0.035	-0.022	1	0.78	0.23
Albumin	-0.27	-0.091	-0.22	-0.23	-0.17	-0.028	-0.085	0.78	1	0.68
Albumin_and_Globulin_Ratio	-0.22	-0.0056	-0.21	-0.2	-0.24	-0.0065	-0.073	0.23	0.68	1

Feature Scaling

In [34]:
```
#StandardScaler in dataframe
sc = StandardScaler()
sc_x = pd.DataFrame(sc.fit_transform(x) , columns=x.columns)
sc_x.head()
```

Out[34]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	T
0	1.241741	-1.752549	-0.418647	-0.493702	-0.427421	-0.351482	-0.314428	
1	1.056874	0.570597	1.210111	1.413923	1.661722	-0.086746	-0.032278	
2	1.056874	0.570597	0.635255	0.919354	0.808927	-0.108807	-0.142385	
3	0.810385	0.570597	-0.370743	-0.387723	-0.447823	-0.362513	-0.307546	
4	1.673096	0.570597	0.092336	0.177500	-0.394778	-0.290813	-0.173353	

Imbalance treatment

In [35]:
```
from imblearn.over_sampling import RandomOverSampler, SMOTE
# over = RandomOverSampler()
# x_over, y_over = over.fit_resample(sc_x,y)

smote = SMOTE()
x_over, y_over = smote.fit_resample(sc_x,y)
```

In [36]:
```
x_train,x_test,y_train,y_test = train_test_split(x_over,y_over,test_size=0.25,random_state=123,  stratify = y
```

case 1: Logistic Regression

In [37]:
```

logit = LogisticRegression(#penalty =  'l2',
                           fit_intercept = True,
                          # multi_class = 'multinomial' ,
                           #dual = False,
                           class_weight = None,
                           solver = 'saga',
                           )
```

case 2: Decision Tree

In [38]:
```
dtc = DecisionTreeClassifier(#min_samples_split= 4,
                             #min_samples_leaf = 2,
                              max_depth =8,
                             criterion='log_loss',
                             #class_weight= None,
                             random_state =123
                            )
```

case 3: Bagging

In [39]:
```
bag = BaggingClassifier( oob_score = False,
                         n_estimators = 200,
                         max_samples = 1.0,
                         max_features = 3,
                         estimator = None,
                         bootstrap_features = False,
                         bootstrap = True,
                         random_state =123)
```

case 4: RandomForest

In [40]:
```
rfc  = RandomForestClassifier(n_estimators =200,
                                  min_samples_split = 8,
                                  min_samples_leaf = 2,
                                  max_features = 'log2',
                                  max_depth = None,
                                  criterion = 'gini',
                                  class_weight = None,
                                  bootstrap = True,
                                  oob_score = True,
                                  random_state =123
                                 )
```

case-5 : KNearestNeighbour

In [41]:
```
knn  = KNeighborsClassifier( )
```

case-6 : SupportVectorMachine

In [42]:
```
#kernel - linear
svm_linear = SVC(kernel='linear')

#kernel - poly
svm_poly = SVC(kernel='poly', C=4, gamma=1)

#kernel - sigmoid
svm_sigmoid = SVC(kernel='sigmoid',C=0.5, gamma=0.1)

#kernel - rbf
svm_rbf = SVC(kernel='rbf', C=3, gamma=1)

# model = svm_sigmoid
# model.fit(x_train,y_train)
# y_pred_train = model.predict(x_train)
# y_pred_test = model.predict(x_test)
# CV_train_acc =cross_val_score(model, x_train, y_train, cv =10).mean()
# CV_test_acc =cross_val_score(model, x_test, y_test, cv =10).mean()

# print(f'━ Training Accuracy(CrossValidation), {CV_train_acc.round(2)*100} %' )
# print(f'━ TestAccuracy(CrossValidation), {CV_test_acc.round(2)*100} %' )
```

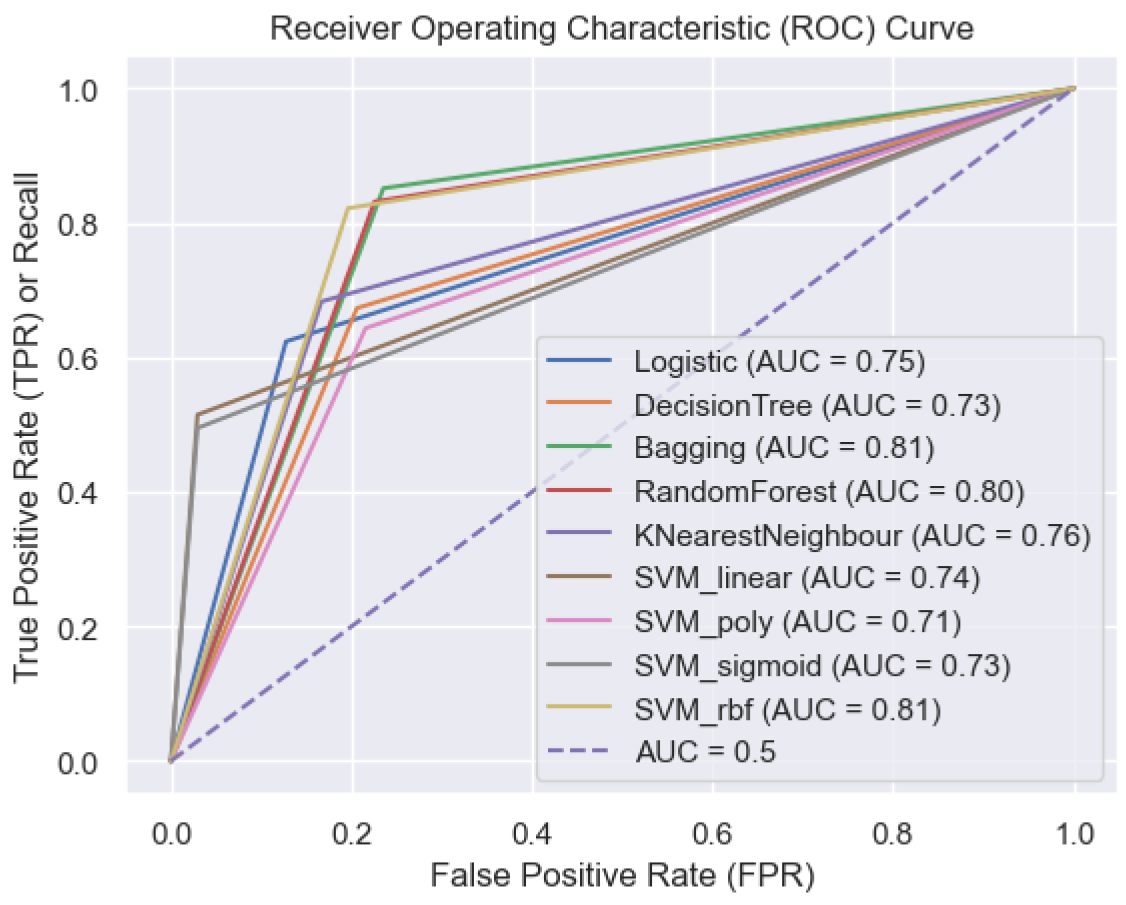
Evaluation of all models

In [43]:
```
estimators =[
          ('Logistic',logit, ),
          ('DecisionTree',dtc, ),
          ('Bagging',bag,),
          ('RandomForest',rfc, ),
          ('KNearestNeighbour',knn,),
          ('SVM_linear',svm_linear),
          ('SVM_poly',svm_poly),
          ('SVM_sigmoid',svm_sigmoid),
          ('SVM_rbf',svm_rbf),
          ]
estimator_good =[]
estimator_bad =[]

for model_name, model in estimators:
    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)

    print('==='*10)
    print(model_name)
    print('==='*10)
#========================================================
#            Cross Validation
#--------------------------------------------------------
    CV_train_acc =(cross_val_score(model, x_train, y_train, cv =10).mean()).round(2)*100
    CV_test_acc =(cross_val_score(model, x_test, y_test, cv =10).mean()).round(2)*100

    print(f'= Training Accuracy(CrossValidation), {CV_train_acc} %' )
    print(f'= TestAccuracy(CrossValidation), {CV_test_acc} %' )
    variance_CV = abs(CV_train_acc - CV_test_acc).round(2)

    print(f'Variance (CrossValidation) : {variance_CV}')
    if (variance_CV > 9) or (CV_train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')

    if(CV_train_acc >70) & (CV_test_acc> 70):
        estimator_good.append((model_name, model))
    elif (CV_train_acc <70) | (CV_test_acc< 70):
        estimator_bad.append((model_name, model))

        print()
```

```
==============================
Logistic
==============================
- Training Accuracy(CrossValidation), 70.0 %
- TestAccuracy(CrossValidation), 74.0 %
Variance (CrossValidation) : 4.0
==============================
DecisionTree
==============================
- Training Accuracy(CrossValidation), 73.0 %
- TestAccuracy(CrossValidation), 64.0 %
Variance (CrossValidation) : 9.0

==============================
Bagging
==============================
- Training Accuracy(CrossValidation), 78.0 %
- TestAccuracy(CrossValidation), 74.0 %
Variance (CrossValidation) : 4.0
==============================
RandomForest
==============================
- Training Accuracy(CrossValidation), 78.0 %
- TestAccuracy(CrossValidation), 72.0 %
Variance (CrossValidation) : 6.0
==============================
KNearestNeighbour
==============================
- Training Accuracy(CrossValidation), 71.0 %
- TestAccuracy(CrossValidation), 71.0 %
Variance (CrossValidation) : 0.0
==============================
SVM_linear
==============================
- Training Accuracy(CrossValidation), 70.0 %
- TestAccuracy(CrossValidation), 72.0 %
Variance (CrossValidation) : 2.0
==============================
SVM_poly
==============================
- Training Accuracy(CrossValidation), 73.0 %
- TestAccuracy(CrossValidation), 65.0 %
Variance (CrossValidation) : 8.0

==============================
SVM_sigmoid
==============================
- Training Accuracy(CrossValidation), 69.0 %
- TestAccuracy(CrossValidation), 74.0 %
Variance (CrossValidation) : 5.0

==============================
SVM_rbf
==============================
- Training Accuracy(CrossValidation), 80.0 %
- TestAccuracy(CrossValidation), 71.0 %
Variance (CrossValidation) : 9.0
```

ROC-AUC charecteristics

In [44]:
```
for model_name, model in estimators:
    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    y_test_binary = (y_test == 1).astype(int)
    y_pred_test_binary = (y_pred_test == 1).astype(int)
    fpr, tpr, _ = roc_curve(y_test_binary, y_pred_test_binary, pos_label=1)
    roc_auc = auc(fpr, tpr)


    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

# Plot the diagonal line representing a random classifier (AUC = 0.5)

plt.plot([0, 1], [0, 1], 'm--', label='AUC = 0.5')

# Set labels and title

plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR) or Recall')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')

# Show the plot
plt.show()
```



VotingClassifier

In [45]:
```
estimator_good
```

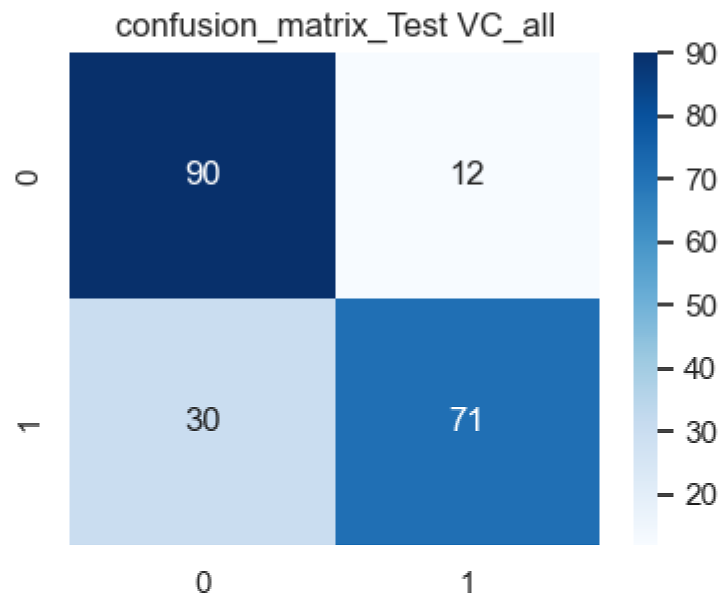
Out[45]:
```
[('Bagging',
  BaggingClassifier(max_features=3, n_estimators=200, random_state=123)),
 ('RandomForest',
  RandomForestClassifier(max_features='log2', min_samples_leaf=2,
                         min_samples_split=8, n_estimators=200, oob_score=True,
                         random_state=123)),
 ('KNearestNeighbour', KNeighborsClassifier()),
 ('SVM_rbf', SVC(C=3, gamma=1))]
```

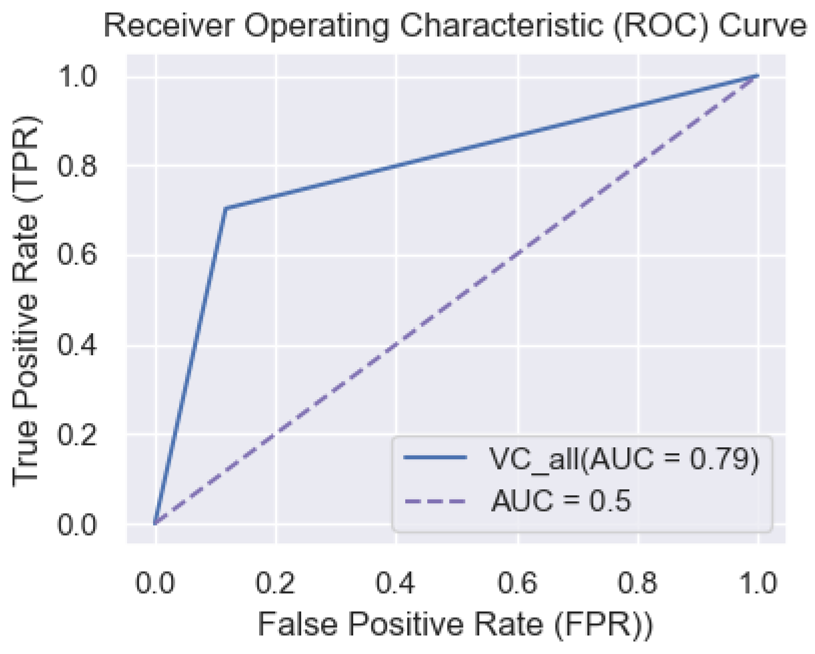
In [46]:
```
estimator_bad
```

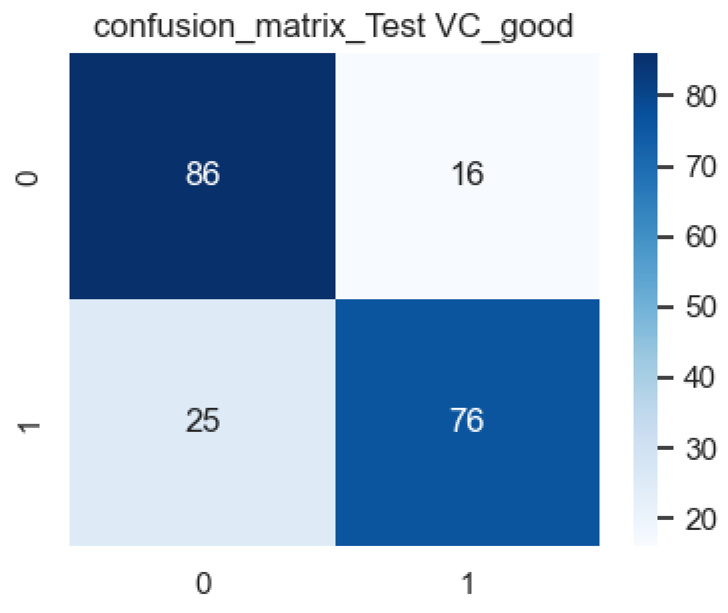
Out[46]:
```
[('DecisionTree',
  DecisionTreeClassifier(criterion='log_loss', max_depth=8, random_state=123)),
 ('SVM_poly', SVC(C=4, gamma=1, kernel='poly')),
 ('SVM_sigmoid', SVC(C=0.5, gamma=0.1, kernel='sigmoid'))]
```

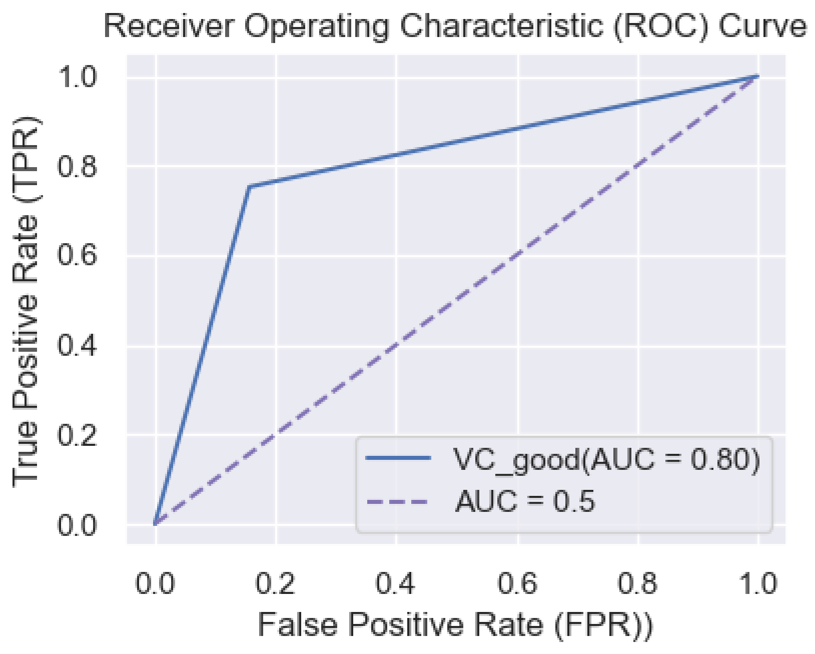
In [47]:
```python
vc_all = VotingClassifier(estimators = estimators, voting='hard',)
vc_good = VotingClassifier(estimators = estimator_good, voting='hard',)
vc_bad = VotingClassifier(estimators = estimator_bad, voting='hard',)
model_vc= [('VC_all', vc_all), ('VC_good', vc_good), ('VC_bad', vc_bad)]

for model_name, model in model_vc:
    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    v_train_accuracy = (cross_val_score(model, x_train, y_train, cv=10, scoring='accuracy').mean()).round(2)*
    v_test_accuracy = (cross_val_score(model, x_test, y_test, cv=10, scoring='accuracy').mean()).round(2)*100

    cm_test = confusion_matrix(y_test, y_pred_test)

    y_test_binary = (y_test == 1).astype(int)
    y_pred_test_binary = (y_pred_test == 1).astype(int)
    fpr, tpr, _ = roc_curve(y_test_binary, y_pred_test_binary, pos_label=1)
    roc_auc = auc(fpr, tpr)
    print('--'*15)
    print(model_name)
    print('--'*15)
    print(f'Train Accuracy(CV) : {v_train_accuracy}%')
    print()
    print(f'Test Accuracy(CV) : {v_test_accuracy}%')
    print()
    print(f'Test confusion matrix :\n {cm_test}')
    print()
    print(f'Test ROC-AUC : {roc_auc:.2f}')

    plt.figure(figsize = (8,3.5))
    plt.subplot(1,2,1)
    sns.heatmap(cm_test, annot = True, cmap= 'Blues')
    plt.title(f'confusion_matrix_Test {model_name}')

    plt.subplot(1,2,2)
    plt.plot(fpr, tpr, label=f'{model_name}(AUC = {roc_auc:.2f})')

# Plot the diagonal line representing a random classifier (AUC = 0.5)

    plt.plot([0, 1], [0, 1], 'm--', label='AUC = 0.5')

# Set labels and title

    plt.xlabel('False Positive Rate (FPR))')
    plt.ylabel('True Positive Rate (TPR)')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.tight_layout()
# Show the plot
plt.show()
```

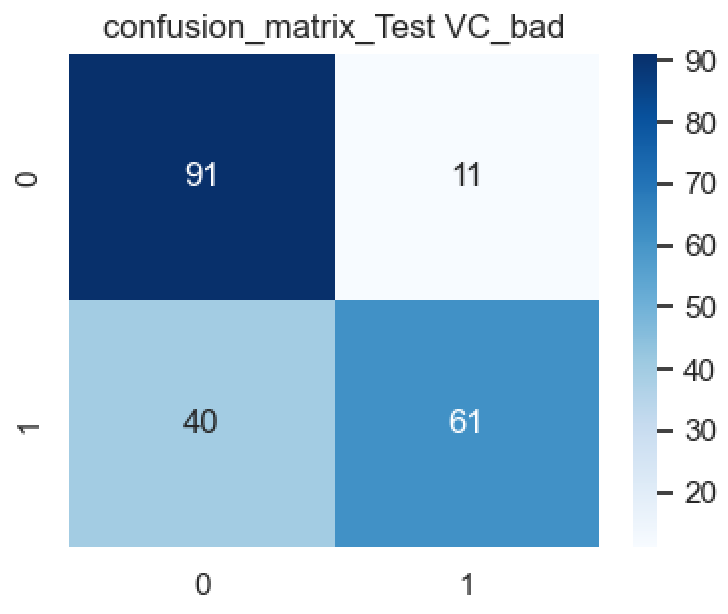
```
------------------------------
VC_all
------------------------------
Train Accuracy(CV) : 77.0%

Test Accuracy(CV) : 74.0%

Test confusion matrix :
 [[90 12]
 [30 71]]

Test ROC-AUC : 0.79
------------------------------
VC_good
------------------------------
Train Accuracy(CV) : 77.0%

Test Accuracy(CV) : 76.0%

Test confusion matrix :
 [[86 16]
 [25 76]]

Test ROC-AUC : 0.80
------------------------------
VC_bad
------------------------------
Train Accuracy(CV) : 74.0%

Test Accuracy(CV) : 74.0%

Test confusion matrix :
 [[91 11]
 [40 61]]

Test ROC-AUC : 0.75
```

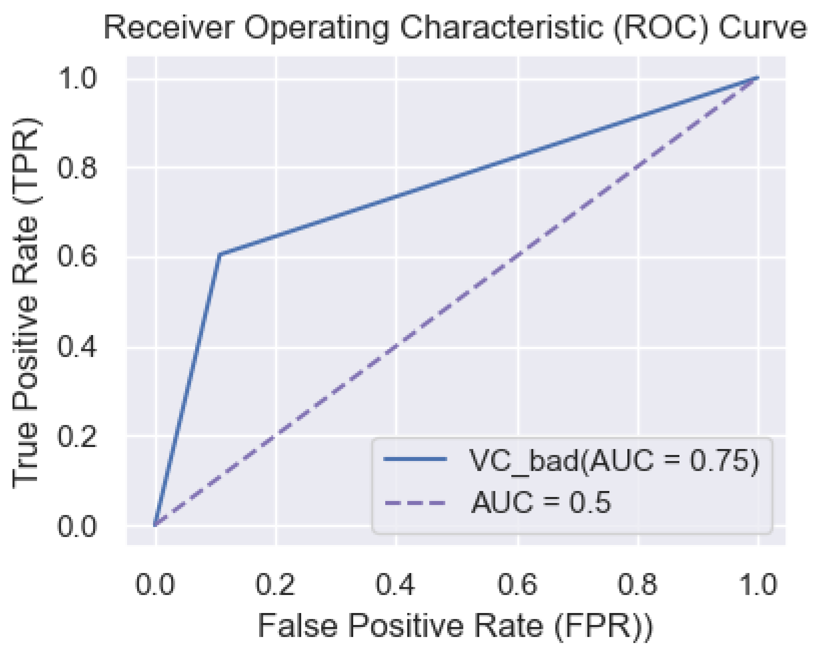

confusion_matrix_Test VC_all
90
12
30
71
0
1
0
1
90
80
70
60
50
40
30
20


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR))
VC_all(AUC = 0.79)
AUC = 0.5
0.0
0.2
0.4
0.6
0.8
1.0


confusion_matrix_Test VC_good
86
16
25
76
0
1
0
1
80
70
60
50
40
30
20


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR))
VC_good(AUC = 0.80)
AUC = 0.5
0.0
0.2
0.4
0.6
0.8
1.0


confusion_matrix_Test VC_bad
91
11
40
61
0
1
0
1
90
80
70
60
50
40
30
20


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR))
VC_bad(AUC = 0.75)
AUC = 0.5
0.0
0.2
0.4
0.6
0.8
1.0

In [48]:
```
for model_name, model in model_vc:
    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
#====================================================
#           Evalutaion Metrics
#----------------------------------------------------
    print( '--'*30)
    print( f'Classification_report:{model_name}\n')
#     print( '--'*30)
#     print( 'Train:')
#     print( classification_report(y_train, y_pred_train))
#    print( 'Test:')
    print( '--'*30)
    print(classification_report(y_test, y_pred_test))
    #print( '--'*30)
```

```
------------------------------------------------------------
Classification_report:VC_all

------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.75      0.88      0.81       102
           1       0.86      0.70      0.77       101

    accuracy                           0.79       203
   macro avg       0.80      0.79      0.79       203
weighted avg       0.80      0.79      0.79       203

------------------------------------------------------------
Classification_report:VC_good

------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.77      0.84      0.81       102
           1       0.83      0.75      0.79       101

    accuracy                           0.80       203
   macro avg       0.80      0.80      0.80       203
weighted avg       0.80      0.80      0.80       203

------------------------------------------------------------
Classification_report:VC_bad

------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.69      0.89      0.78       102
           1       0.85      0.60      0.71       101

    accuracy                           0.75       203
   macro avg       0.77      0.75      0.74       203
weighted avg       0.77      0.75      0.74       203
```

Conclusion

- In this project all the models are tested individually.

Model	Train Accuracy	Test Accuracy
Logistic	70	74
DecisionTree	73	64
Bagging	78	74
RandomForest	78	72
KNN	71	71
SVM_linear	70	72
SVM_poly	73	65
SVM_sigmoid	69	74
SVM_rbf	80	71

-Three different Votingclassifier with `estimator_all` , `estimator_good` , `estimator_bad` is used to check the performance of the model.

> **1. `estimator_all` is constructed with all the individual models.**
>
> **2. `estimator_good` is constructed with the individual models whose train and test accuracy is above 70%.**
>
> **3. `estimator_bad` is constructed with all the individual models whose train or test accuracy is below 70%.**

```
|-----------------|----------|---------|-----------|-------------|
| Estimator       |  Train   | Test    |    ROC    | Confusion   |
|                 |  Accuracy| Accuracy|           |   Matrix    |
|-----------------|----------|---------|-----------|-------------|
|  estimator_all  |     77   |   74    |    0.79   |   [[90 12]  |
|                 |          |         |           |   [30 71]]  |
|-----------------|----------|---------|-----------|-------------|
|  estimator_good |     77   |   76    |    0.80   |  [[86  16]   |
|                 |          |         |           |  [25  76]]  |
```

Project completed on 7th August 2023 by Payal Mohanty