Data Collection & Preparation

Collect The Dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset

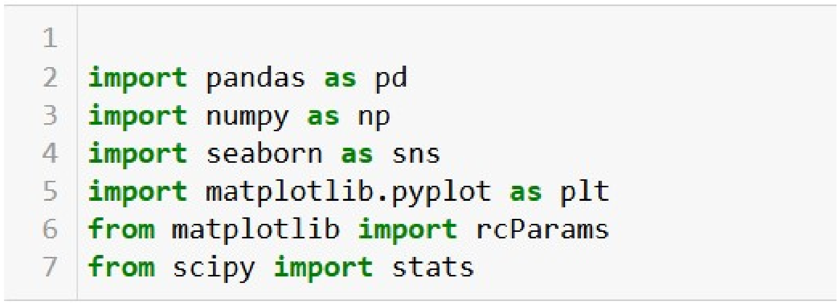
Link: https://www.kaggle.com/datasets/uciml/indian-liver-patient-records

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Importing The Libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

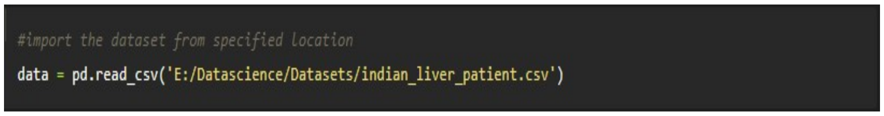


```

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import rcParams
from scipy import stats
```

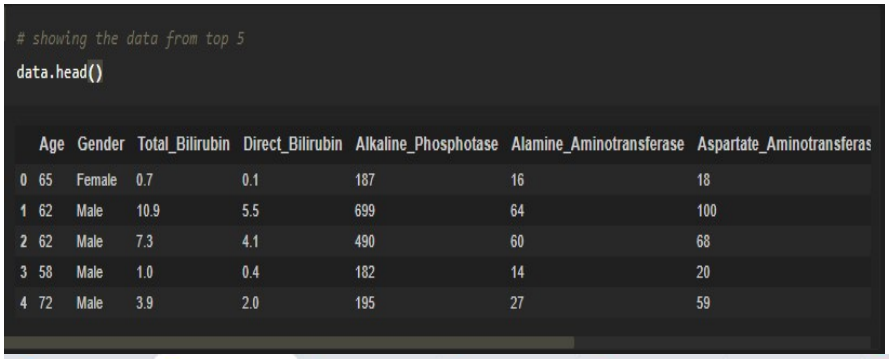
Read The Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

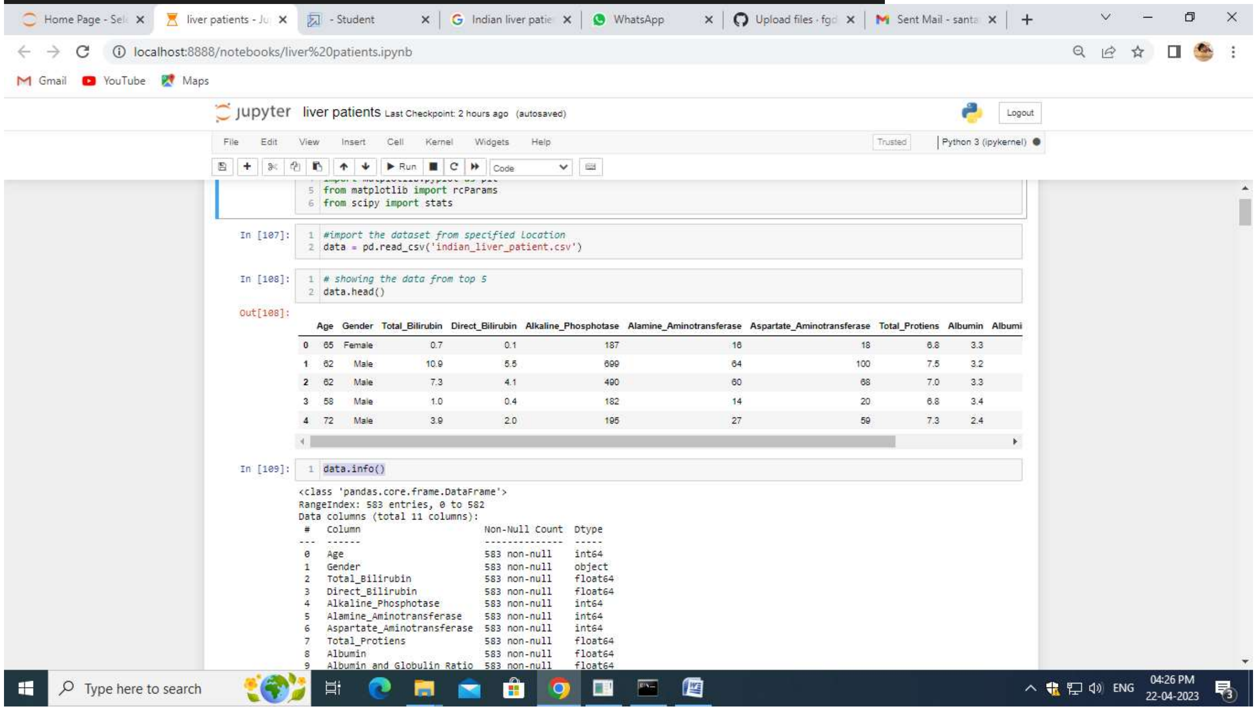


```python
#import the dataset from specified location
data = pd.read_csv('E:/Datascience/Datasets/indian_liver_patient.csv')
```



```python
# showing the data from top 5
data.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferas
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



```
from matplotlib import rcParams
from scipy import stats
```

```
In [107]:  #import the dataset from specified location
           data = pd.read_csv('indian_liver_patient.csv')
```

```
In [108]:  # showing the data from top 5
           data.head()
Out[108]:
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumi
0	65	Female	0.7	0.1	187	16	18	6.8	3.3	
1	62	Male	10.9	5.5	699	64	100	7.5	3.2	
2	62	Male	7.3	4.1	490	60	68	7.0	3.3	
3	58	Male	1.0	0.4	182	14	20	6.8	3.4	
4	72	Male	3.9	2.0	195	27	59	7.3	2.4	

```
In [109]:  data.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  583 non-null    float64
```

Data Preparation

As we have understood how the data is, let's pre-process the collected data.

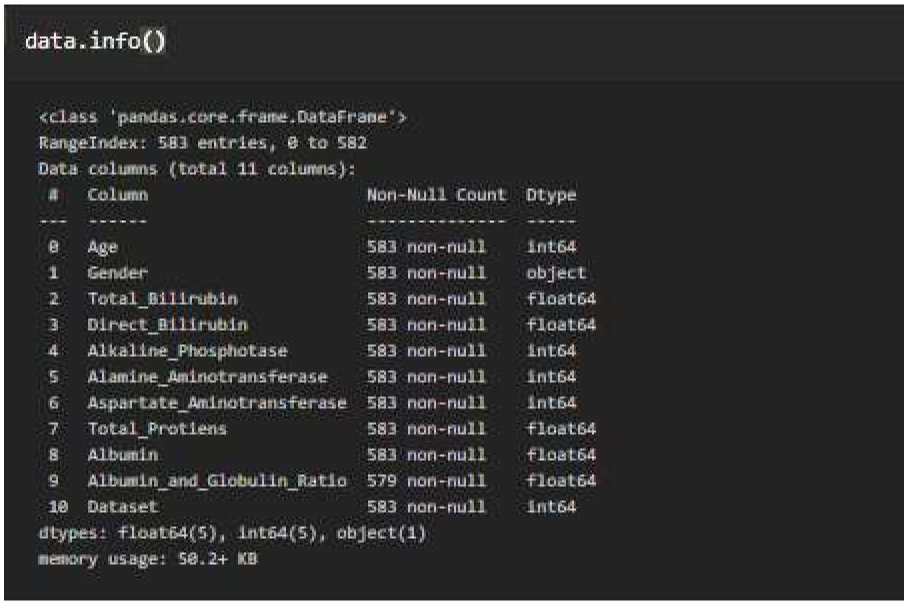
The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps

Handling Missing Values

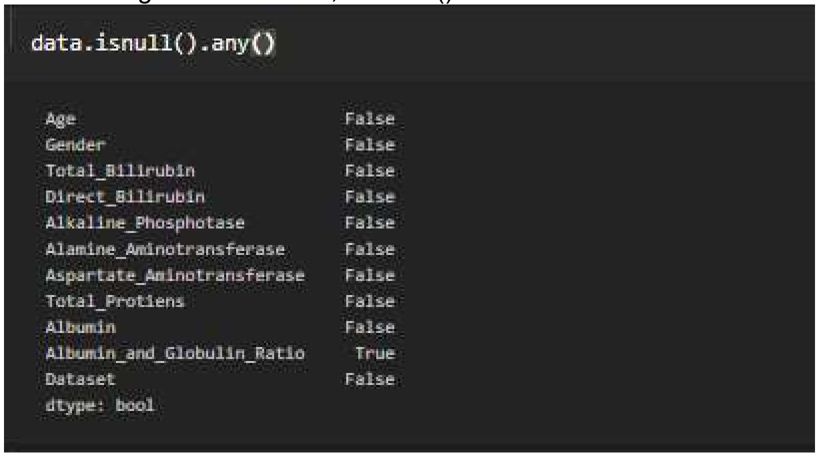
Let’s find the shape of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used.



```
data.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
```

For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function.

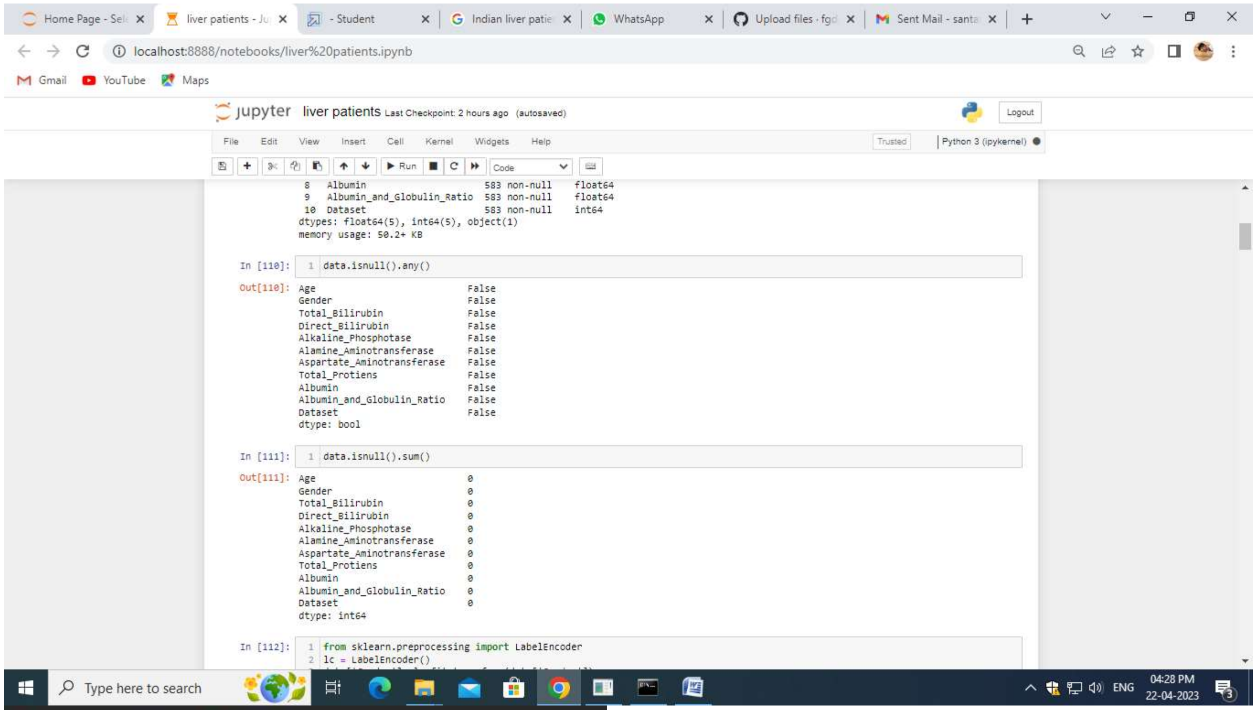


```
data.isnull().any()
```

```
Age                           False
Gender                        False
Total_Bilirubin               False
Direct_Bilirubin              False
Alkaline_Phosphotase          False
Alamine_Aminotransferase      False
Aspartate_Aminotransferase    False
Total_Protiens                False
Albumin                       False
Albumin_and_Globulin_Ratio     True
Dataset                       False
dtype: bool
```

We can see that there are null values in the Albumin_and_Globulin_Ration Column.

Let us check how many numbers of null records present in the Closing Value column using sum() function.



```
     8   Albumin                     583 non-null    float64
     9   Albumin_and_Globulin_Ratio  583 non-null    float64
     10  Dataset                     583 non-null    int64
    dtypes: float64(5), int64(5), object(1)
    memory usage: 50.2+ KB
```

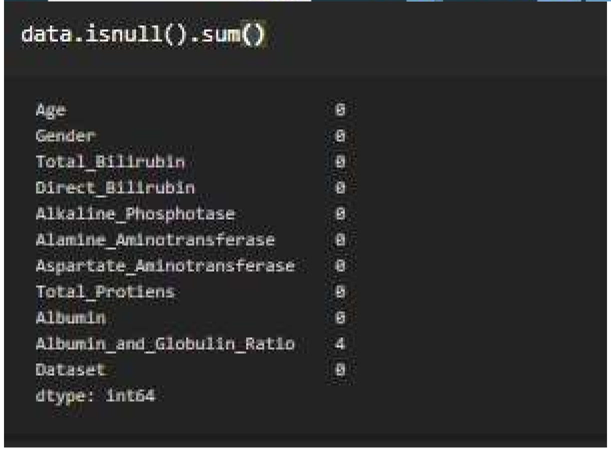
```
In [110]: data.isnull().any()
```

```
Out[110]: Age                           False
          Gender                        False
          Total_Bilirubin               False
          Direct_Bilirubin              False
          Alkaline_Phosphotase          False
          Alamine_Aminotransferase      False
          Aspartate_Aminotransferase    False
          Total_Protiens                False
          Albumin                       False
          Albumin_and_Globulin_Ratio    False
          Dataset                       False
          dtype: bool
```

```
In [111]: data.isnull().sum()
```

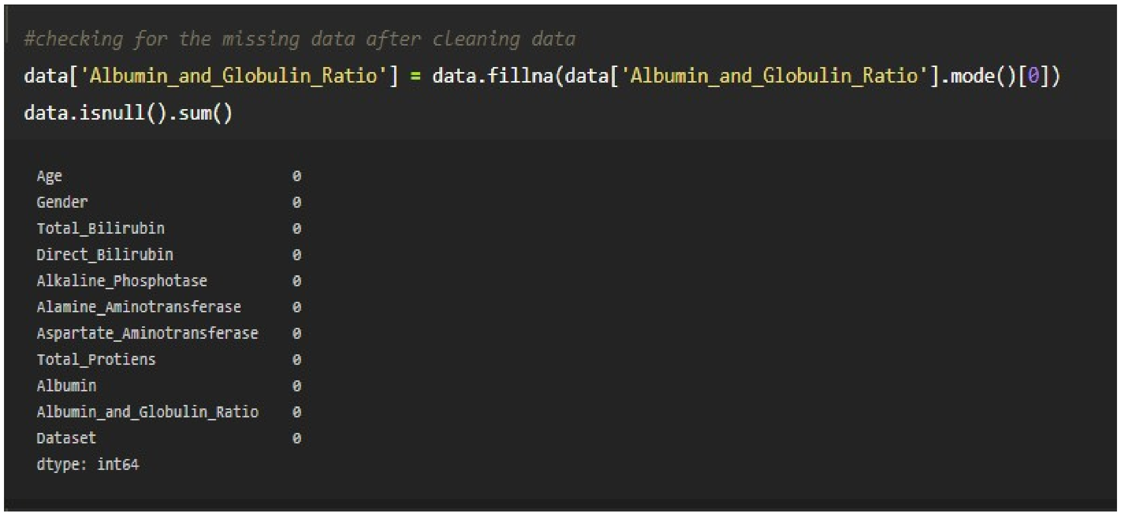
```
Out[111]: Age                           0
          Gender                        0
          Total_Bilirubin               0
          Direct_Bilirubin              0
          Alkaline_Phosphotase          0
          Alamine_Aminotransferase      0
          Aspartate_Aminotransferase    0
          Total_Protiens                0
          Albumin                       0
          Albumin_and_Globulin_Ratio    0
          Dataset                       0
          dtype: int64
```

```
In [112]: from sklearn.preprocessing import LabelEncoder
          lc = LabelEncoder()
```



```
data.isnull().sum()

 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
```

From the above code of analysis, we can infer that columns such as Albumin and Globulin Ratio is having the missing values, we need to treat them in a required way.



```
#checking for the missing data after cleaning data
data['Albumin_and_Globulin_Ratio'] = data.fillna(data['Albumin_and_Globulin_Ratio'].mode()[0])
data.isnull().sum()
```

```
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    0
Dataset                       0
dtype: int64
```

We will fill in the missing values in the numeric data type using the mean value of that particular column and categorical data type using the most repeated values.

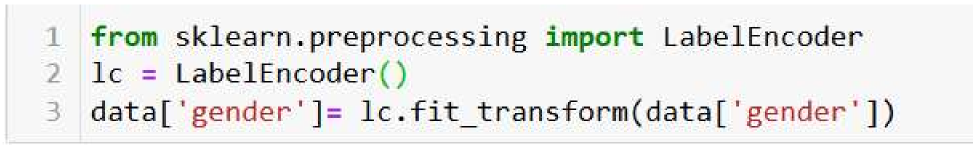
Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project

we are using manual encoding with the help of list comprehension.

In our project, for Gender, encoding is done.

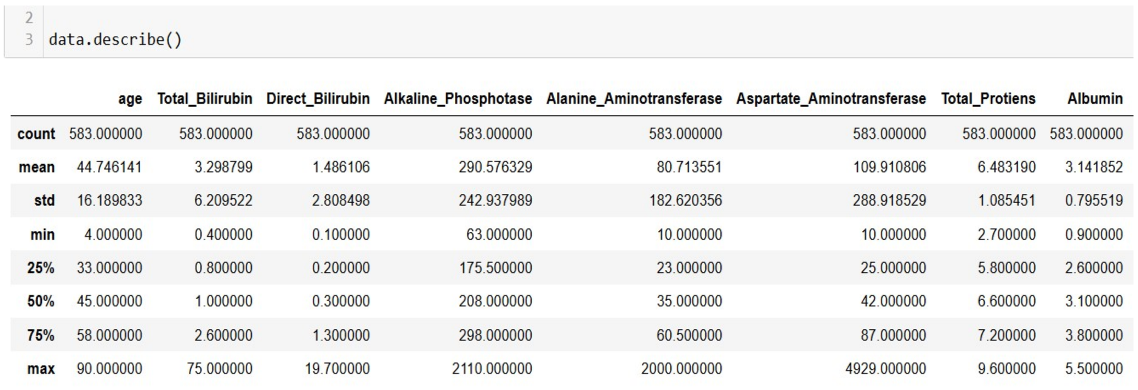


```
from sklearn.preprocessing import LabelEncoder
lc = LabelEncoder()
data['gender']= lc.fit_transform(data['gender'])
```

Exploratory Data Analysis

Descriptive Statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.



```python

data.describe()
```

	age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alanine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806	6.483190	3.141852
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529	1.085451	0.795519
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000	0.900000
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000	5.800000	2.600000
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000	3.100000
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000	7.200000	3.800000
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000	5.500000

Visual Analysis

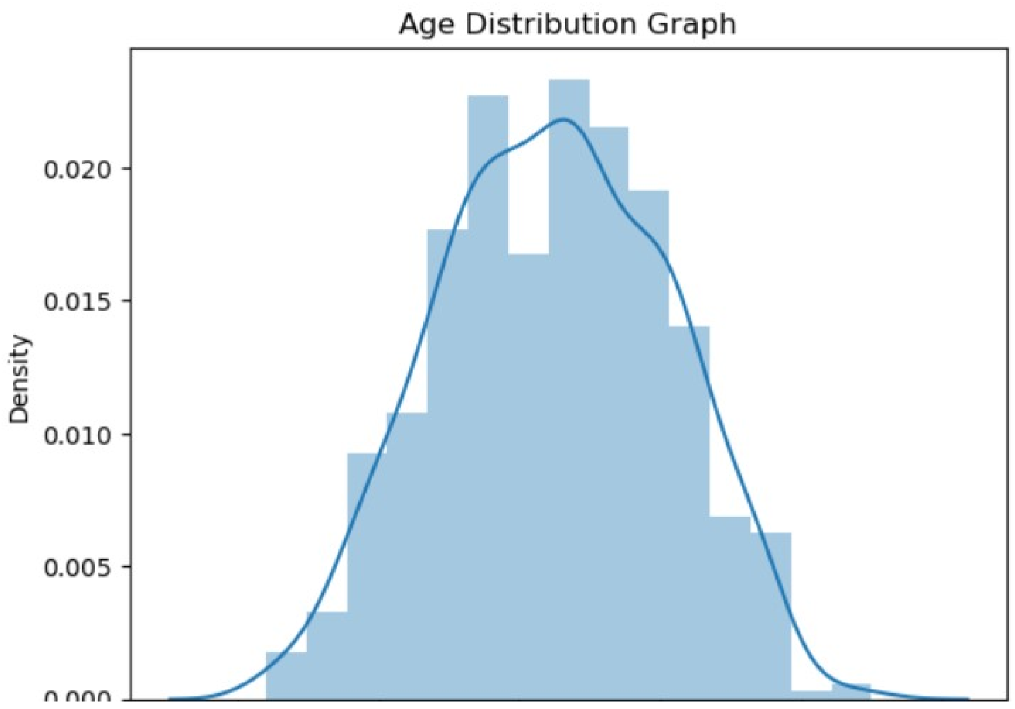
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions

Univariate Analysis

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.
The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

```
sns.distplot(data['age'])
plt.title('Age Distribution Graph')
plt.show()

```

```
D:\Anaconda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a
oved in a future version. Please adapt your code to use either `displot` (a figure-level fun
`histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

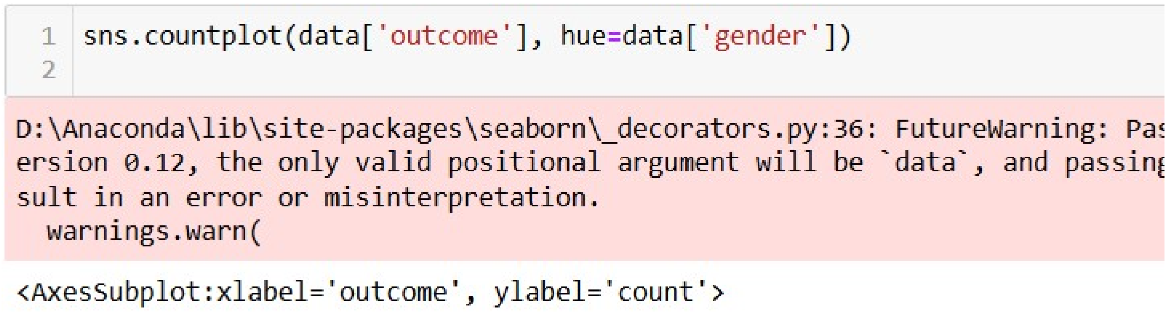


In our dataset we have some categorical features. With the count plot function, we are going to count the unique category in those features.

Countplot:-

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for barplot() , so you can compare counts across nested variables.

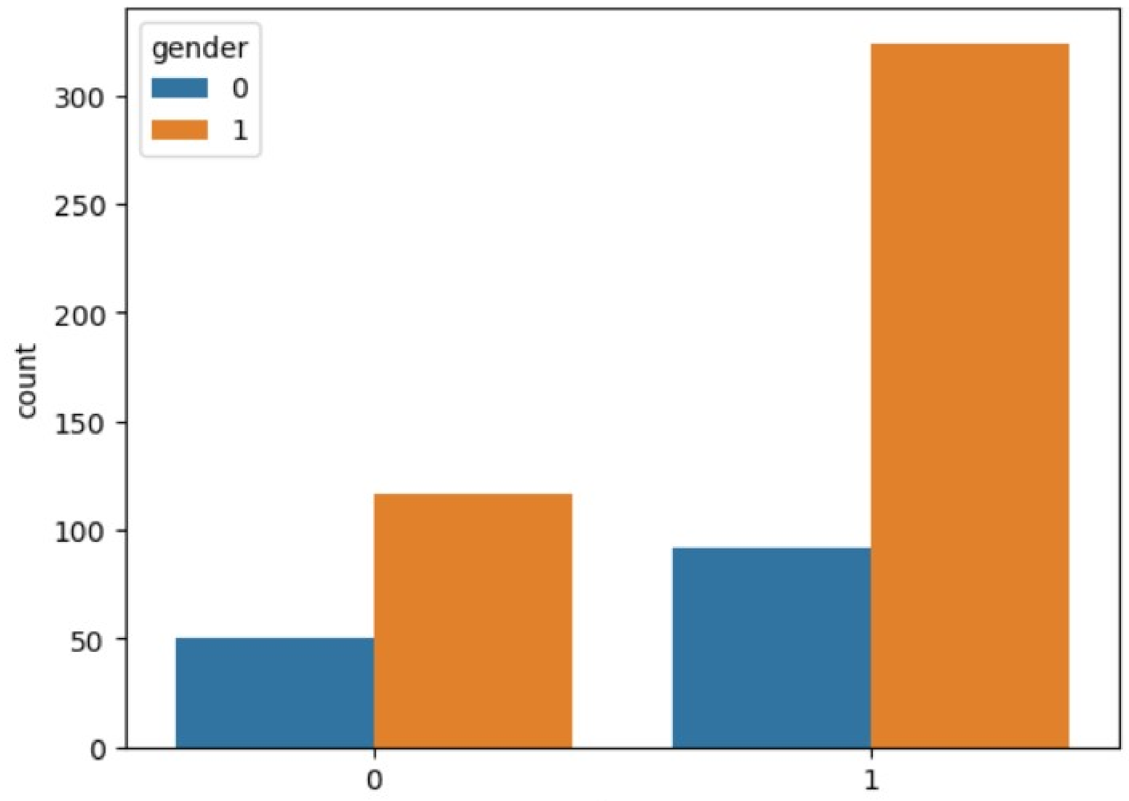
Bivariate Analysis



```python
sns.countplot(data['outcome'], hue=data['gender'])

```

```
D:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
ersion 0.12, the only valid positional argument will be `data`, and passing
sult in an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='outcome', ylabel='count'>
```



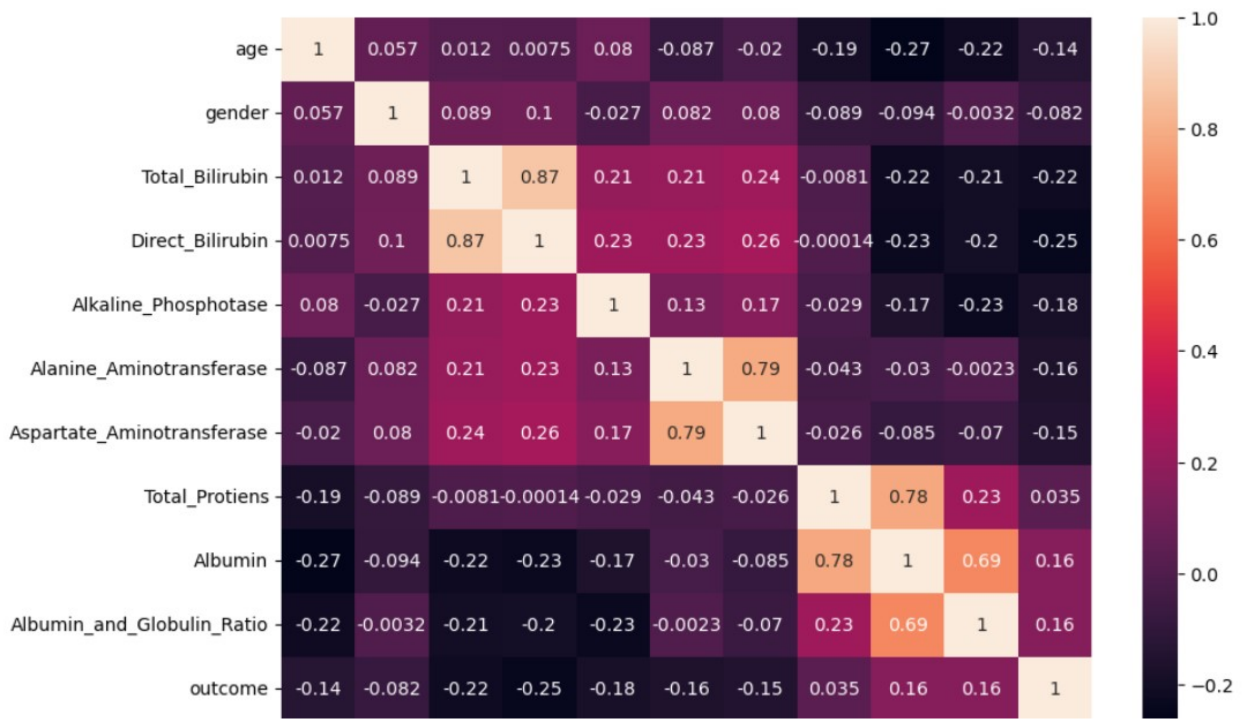
From the graph we can infer that , gender and outcome is a categorical variables with 2 categories , from gender column we can infer that 1-category is having more weightage than category-0, and outcome with 0,it means healthy is a underclass when compared with category -1, which means liver patient.

Multivariate Analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used a heat plot from the seaborn package.

```
plt.figure(figsize=(10,7))

sns.heatmap(df.corr(),annot=True)
```

```
<AxesSubplot:>
```

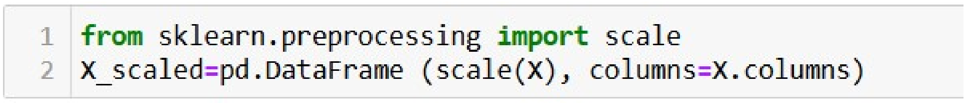


Now, the code would be normalising the data by scaling it to have a similar range of values, and then splitting that data into a training set and a test set for training the model and testing its performance, respectively.

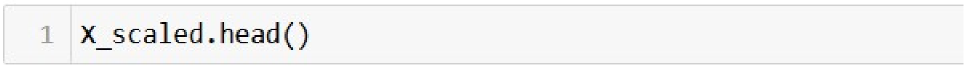
Scaling the Data

Scaling is one the important process, we have to perform on the dataset, because of data measures in different ranges can leads to mislead in prediction

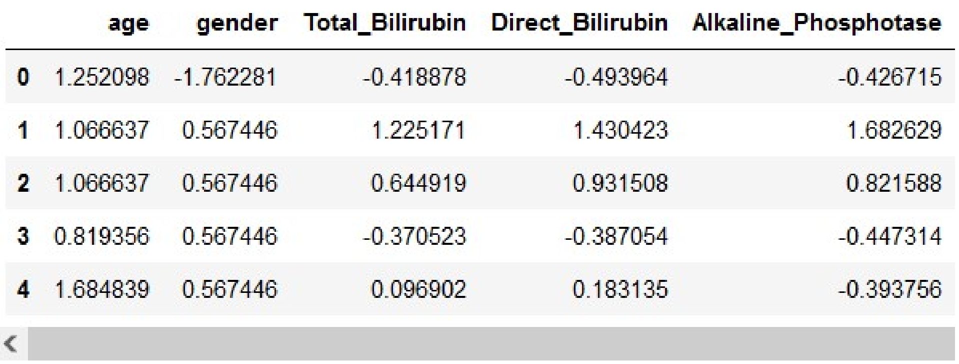
Models such as KNN, Logistic regression need scaled data, as they follow distance based method and Gradient Descent concept



```python
from sklearn.preprocessing import scale
X_scaled=pd.DataFrame (scale(X), columns=X.columns)
```



```python
X_scaled.head()
```

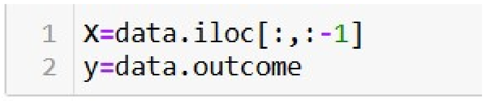


	age	gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase
0	1.252098	-1.762281	-0.418878	-0.493964	-0.426715
1	1.066637	0.567446	1.225171	1.430423	1.682629
2	1.066637	0.567446	0.644919	0.931508	0.821588
3	0.819356	0.567446	-0.370523	-0.387054	-0.447314
4	1.684839	0.567446	0.096902	0.183135	-0.393756

We will perform scaling only on the input values.Once the dataset is scaled, it will be converted into an array and we need to convert it back to a dataframe
Splitting data into train and test

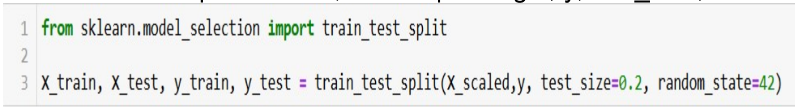
Now let’s split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set



```python
X=data.iloc[:,:-1]
y=data.outcome
```

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.



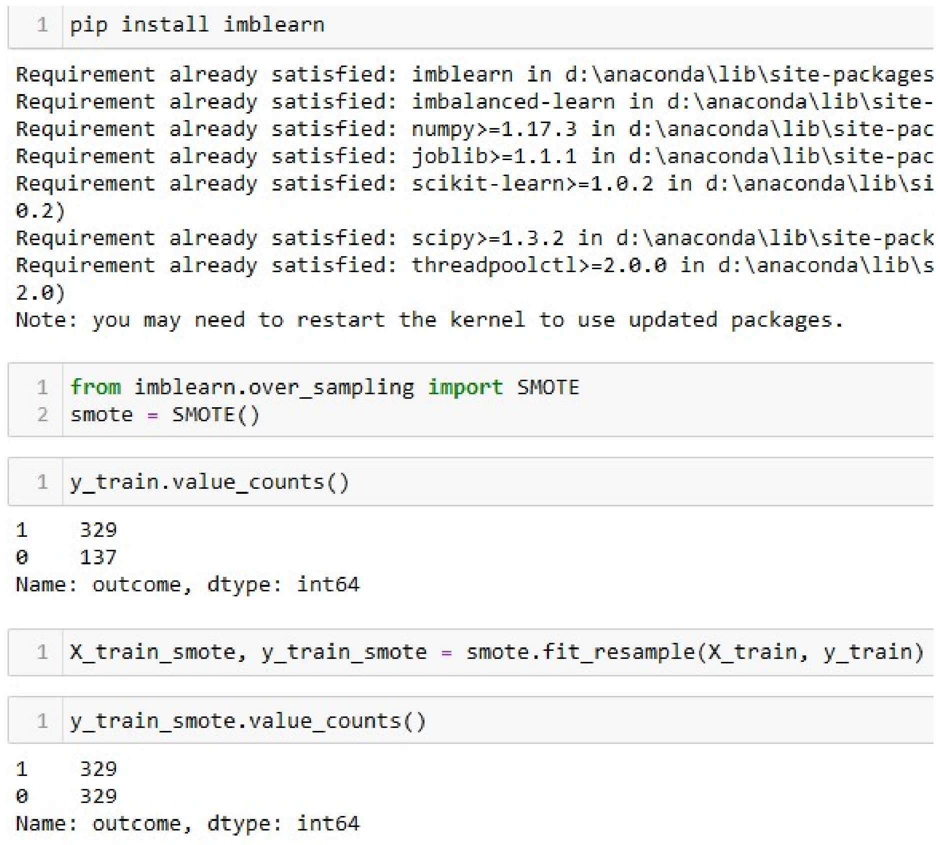
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled,y, test_size=0.2, random_state=42)
```

Handling Imbalance Data

Data Balancing is one of the most important step, which need to be performed for classification models, because when we train our model on imbalanced dataset, we will get biased results, which means our model is able to predict only one class element

For balancing the data we are using the SMOTE Method.

SMOTE: Synthetic minority over sampling technique, which will create new synthetic data points for under class as per the requirements given by us using KNN method.



```
pip install imblearn
```

```
Requirement already satisfied: imblearn in d:\anaconda\lib\site-packages
Requirement already satisfied: imbalanced-learn in d:\anaconda\lib\site-
Requirement already satisfied: numpy>=1.17.3 in d:\anaconda\lib\site-pac
Requirement already satisfied: joblib>=1.1.1 in d:\anaconda\lib\site-pac
Requirement already satisfied: scikit-learn>=1.0.2 in d:\anaconda\lib\si
0.2)
Requirement already satisfied: scipy>=1.3.2 in d:\anaconda\lib\site-pack
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\anaconda\lib\s
2.0)
Note: you may need to restart the kernel to use updated packages.
```

```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
```

```
y_train.value_counts()
```

```
1    329
0    137
Name: outcome, dtype: int64
```

```
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

```
y_train_smote.value_counts()
```

```
1    329
0    329
Name: outcome, dtype: int64
```

From the above picture, we can infer that,previously our dataset had 329 class 1, and 132 class items, after applying smote technique on the dataset the size has become equal.

Model Building

Training The Model In Multiple Algorithms

Now our data is cleaned and it’s time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance