

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: 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/mansoordaku/ckdisease>

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 several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

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

Importing Libraries

```
import pandas as pd #used for data manipulation
import numpy as np #used for numerical analysis
from collections import Counter #it return counts of number of classes
import matplotlib.pyplot as plt #used for data visualization
import seaborn as sns #data visualization library
import missingno as msno #finding missing values
from sklearn.metrics import accuracy_score, confusion_matrix #model performance
from sklearn.model_selection import train_test_split #splits data in random train and test array
from sklearn.preprocessing import LabelEncoder #encoding the levels of categorical features
from sklearn.linear_model import LogisticRegression #Classification ML algorithm
import pickle #Python object hierarchy is converted into a byte stream.
```

Activity 1.2: 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.

```
data=pd.read_csv("ChronicDisease.csv") #loading the csv data
data.head() #return you the first 5 rows values
```

| | id | age | bp | sg | al | su | rbc | pc | pot | ba | pcv | wc | rc | htn | dm | cad | appet | pe | ane | classification | |
|---|----|------|------|-------|-----|-----|--------|----------|------------|------------|-----|----|------|-----|-----|-----|-------|------|-----|----------------|-----|
| 0 | 0 | 48.0 | 80.0 | 1.000 | 1.0 | 0.0 | NaN | normal | notpresent | notpresent | - | 44 | 7800 | 52 | yes | yes | no | good | no | no | ckd |
| 1 | 1 | 7.0 | 50.0 | 1.000 | 4.0 | 0.0 | NaN | normal | notpresent | notpresent | - | 38 | 6000 | NaN | no | no | no | good | no | no | ckd |
| 2 | 2 | 62.0 | 80.0 | 1.010 | 2.0 | 1.0 | normal | normal | notpresent | notpresent | - | 31 | 7500 | NaN | no | yes | no | poor | no | yes | ckd |
| 3 | 3 | 48.0 | 70.0 | 1.005 | 4.0 | 0.0 | normal | abnormal | present | notpresent | - | 32 | 6700 | 3.9 | yes | no | no | poor | yes | yes | ckd |
| 4 | 4 | 51.0 | 80.0 | 1.010 | 2.0 | 0.0 | normal | normal | notpresent | notpresent | - | 25 | 7300 | 4.6 | no | no | no | good | no | no | ckd |

5 rows x 26 columns

Activity 2: 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.

- Rename the columns
- Handling missing values
- Handling categorical data
- Handling Numerical 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.

Activity 2.1: Rename the columns

```
data.columns #return all the column names

Index(['age', 'bu', 'sg', 'al', 'wa', 'wbc', 'pcv', 'pcr', 'ba', 'hgt', 'bu',
       'tc', 'sod', 'pot', 'hema', 'pcr', 'wa', 'pcr', 'hgt', 'ba', 'sod',
       'appet', 'pe', 'ane', 'classification'],
      dtype='object')

data.columns=['age','blood_pressure','specific_gravity','albumin',
             'sugar','red_blood_cells','pus_cell','pus_cell_clumps','bacteria',
             'blood_glucose_random','blood_urea','serum_creatinine','sodium','potassium',
             'hemoglobin','packed_cell_volume','white_blood_cell_count','red_blood_cell_count',
             'hypertension','diabetesmellitus','coronary_artery_disease','appetite',
             'pedal_edema','anemia','class'] # manually giving the name of the columns

data.columns

Index(['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar',
       'red_blood_cells', 'pus_cell', 'pus_cell_clumps', 'bacteria',
       'blood glucose random', 'blood_urea', 'serum_creatinine', 'sodium',
       'potassium', 'hemoglobin', 'packed_cell_volume',
       'white_blood_cell_count', 'red_blood_cell_count', 'hypertension',
       'diabetesmellitus', 'coronary_artery_disease', 'appetite',
       'pedal_edema', 'anemia', 'class'],
      dtype='object')
```

Activity 2.2: 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() #This will give you a summary of dataset
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 444 entries, 0 to 443
Data columns (total 28 columns):
 #   Column                Non-Null Count  Dtype  
---  --   ---
 0   age                   444 non-null    float64
 1   blood_pressure        444 non-null    float64
 2   specific_gravity      444 non-null    float64
 3   albumin               444 non-null    float64
 4   sugar                 444 non-null    float64
 5   red_blood_cells       444 non-null    object  
 6   wbc                   444 non-null    object  
 7   wbc_cell_clumps       444 non-null    object  
 8   bacteria              444 non-null    object  
 9   blood_glucose_random  444 non-null    float64
10   blood_urea            444 non-null    float64
11   serum_creatinine      444 non-null    float64
12   sodium                444 non-null    float64
13   potassium             444 non-null    float64
14   hemoglobin            444 non-null    float64
15   packed_cell_volume    444 non-null    object  
16   white_blood_cell_count 444 non-null    object  
17   red_blood_cell_count  444 non-null    object  
18   hypertension          444 non-null    object  
19   diabetesmellitus      444 non-null    object  
20   coronary_artery_disease 444 non-null    object  
21   appetite              444 non-null    object  
22   pedal_edema           444 non-null    object  
23   anemia                444 non-null    object  
24   class                 444 non-null    object  
dtypes: float64(13), object(15)
memory usage: 76.0+ KB
```

```
data.isnull().any() #It will return true if any column is having null values
```

```
age                True
blood_pressure     True
specific_gravity   True
albumin            True
sugar              True
red_blood_cells    True
wbc                True
wbc_cell_clumps    True
bacteria           True
blood_glucose_random True
blood_urea         True
serum_creatinine   True
sodium             True
potassium          True
hemoglobin         True
packed_cell_volume True
white_blood_cell_count True
red_blood_cell_count True
hypertension       True
diabetesmellitus   True
coronary_artery_disease True
appetite           True
pedal_edema        True
anemia             True
class              False
dtypes: bool
```

```

1 data['blood_glucose_random'].fillna(data['blood_glucose_random'].mean(),inplace=True)
2 data['blood_pressure'].fillna(data['blood_pressure'].mean(),inplace=True)
3 data['blood_urea'].fillna(data['blood_urea'].mean(),inplace=True)
4 data['hemoglobin'].fillna(data['hemoglobin'].mean(),inplace=True)
5 data['packed_cell_volume'].fillna(data['packed_cell_volume'].mean(),inplace=True)
6 data['potassium'].fillna(data['potassium'].mean(),inplace=True)
7 data['red_blood_cell_count'].fillna(data['red_blood_cell_count'].mean(),inplace=True)
8 data['serum_creatinine'].fillna(data['serum_creatinine'].mean(),inplace=True)
9 data['sodium'].fillna(data['sodium'].mean(),inplace=True)
10 data['white_blood_cell_count'].fillna(data['white_blood_cell_count'].mean(),inplace=True)

```

```

1 data['age'].fillna(data['age'].mode()[0],inplace=True)
2 data['hypertension'].fillna(data['hypertension'].mode()[0],inplace=True)
3 data['pus_cell_clumps'].fillna(data['pus_cell_clumps'].mode()[0],inplace=True)
4 data['appetite'].fillna(data['appetite'].mode()[0],inplace=True)
5 data['albumin'].fillna(data['albumin'].mode()[0],inplace=True)
6 data['pus_cell'].fillna(data['pus_cell'].mode()[0],inplace=True)
7 data['red_blood_cells'].fillna(data['red_blood_cells'].mode()[0],inplace=True)
8 data['coronary_artery_disease'].fillna(data['coronary_artery_disease'].mode()[0],inplace=True)
9 data['bacteria'].fillna(data['bacteria'].mode()[0],inplace=True)
10 data['anemia'].fillna(data['anemia'].mode()[0],inplace=True)
11 data['sugar'].fillna(data['sugar'].mode()[0],inplace=True)
12 data['diabetesmellitus'].fillna(data['diabetesmellitus'].mode()[0],inplace=True)
13 data['pedal_edema'].fillna(data['pedal_edema'].mode()[0],inplace=True)
14 data['specific_gravity'].fillna(data['specific_gravity'].mode()[0],inplace=True)

```

Let's now check the count of null values after filling all null values using **isnull.sum()**

Activity 2.3: Handling Categorical columns

The below code is used for fetching all the object or categorical type of columns from our data and we are storing it as **set** in variable **catcols**.

```

catcols=set(data.dtypes[data.dtypes=="O"].index.values) # only fetch the object type columns
print(catcols)

{'hypertension', 'packed_cell_volume', 'class', 'coronary_artery_disease', 'anemia', 'red_blood_cell_count', 'red_blood_cells', 'bacteria',
'pedal_edema', 'appetite', 'pus_cell', 'diabetesmellitus', 'pus_cell_clumps', 'white_blood_cell_count'}

```

As, you can observe that it gives us the same count of columns which we find previously.

Scanned by TapScanner

In the above we are looping with each categorical column and printing the classes of each categorical columns using counter function so that we can detect which columns are categorical and which are not.

If you observe some columns have a few classes and some have many, those columns are having many classes can be considered as numerical column and we have to remove it and add it to the continuous columns.

```
catcols.remove('red_blood_cell_count') # remove is used for removing a particular column
catcols.remove('packed_cell_volume')
catcols.remove('white_blood_cell_count')
print(catcols)

{'hypertension', 'class', 'coronary_artery_disease', 'anemia', 'red_blood_cells', 'bacteria', 'renal_cream', 'appetite', 'pus_cell', 'diabetesmellitus', 'pus_cell_clumps'}
```

As we store our columns as set, we can make use of **remove** function which is used to remove the element in our case we can take it as columns.

Activity 2.3.1: Label Encoding for categorical columns

Typically, any structured dataset includes multiple columns with combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithms too. We need to convert each text category to numbers in order for the machine to process those using mathematical equations.

How should we handle categorical variables? There are Multiple way to handle, but will see one of it is LabelEncoding.

Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

Let's see how to implement label encoding in Python using the scikit-learn library.

we have to convert only the text class category columns; we first select it then we will implement Label Encoding to it.

Labeling Encoding of Categorical Column

```
#specific_gravity', 'albumin', 'sugar'(as these columns are numerical it is removed)
catcols=['anemia', 'pedal_edema', 'appetite', 'bacteria', 'class', 'coronary_artery_disease', 'diabetesmellitus',
'hypertension', 'pus_cell', 'pus_cell_clumps', 'red_blood_cells'] #only considered the text class columns

from sklearn.preprocessing import LabelEncoder #importing the labelEncoding from sklearn
for i in catcols: #looping through all the categorical columns
    print("LABEL ENCODING OF:", i)
    LEi = LabelEncoder() # creating an object of LabelEncoder
    print(c(data[i])) #getting the classes values before transformation
    data[i] = LEi.fit_transform(data[i]) # transforming our text classes to numerical values
    print(c(data[i])) #getting the classes values after transformation
    print("\n\n")
```

In the above code we are looping through all the selected text class categorical columns and performing label encoding.

```
LABEL ENCODING OF: anemia
Counter({'no': 348, 'yes': 48})
Counter([0: 348, 1: 48])
=====
LABEL ENCODING OF: pedal_edema
Counter({'no': 324, 'yes': 76})
Counter([0: 324, 1: 76])
=====
LABEL ENCODING OF: appetite
Counter({'good': 338, 'poor': 42})
Counter([0: 338, 1: 42])
=====
LABEL ENCODING OF: bacteria
Counter({'notpresent': 378, 'present': 22})
Counter([0: 378, 1: 22])
=====
LABEL ENCODING OF: class
Counter({'cad': 258, 'stroke': 348})
Counter([0: 258, 1: 348])
=====
LABEL ENCODING OF: coronary_artery_disease
Counter({'no': 308, 'yes': 34})
Counter([0: 308, 1: 34])
=====
LABEL ENCODING OF: diabetesmellitus
Counter({'no': 383, 'yes': 57})
Counter([0: 383, 1: 57])
=====
LABEL ENCODING OF: hypertension
Counter({'no': 353, 'yes': 57})
Counter([0: 353, 1: 57])
=====
LABEL ENCODING OF: pus_cell
Counter({'normal': 334, 'abnormal': 76})
Counter([1: 334, 0: 76])
=====
LABEL ENCODING OF: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
Counter([0: 358, 1: 42])
=====
LABEL ENCODING OF: red_blood_cells
Counter({'normal': 454, 'abnormal': 47})
```

As you can see here, after performing label encoding alphabetical classes is converted to numeric.

Activity 2.4: Handling Numerical columns

```

controls=set(data.dtypes[data.dtypes!='O'].index.values)# only fetch the float and int type columns
#controls=pd.DataFrame(data,columns=controls)
print(controls)

('blood_sugar', 'serum_creatinine', 'albumin', 'blood_pressure', 'blood glucose random', 'sugar', 'sodium', 'hemoglobin', 'specific_gravity', 'age', 'potassium')

```

Same as we did with categorical columns, we are making use of **dtypes** for finding the continuous columns

```

for i in controls:
    print("Continuous Columns :",i)
    print(c(data[i]))
    print('*'*120+'\n')

```

If we observe the output of the above code we can observe that some columns have few values or you can say classes which can be considered as categorical columns. So, let's remove it and add the columns which we observed into their respective variables.

```

controls.remove('specific_gravity')
controls.remove('albumin')
controls.remove('sugar')
print(controls)

```

With the help of **add()** function we can add an element.

```

controls.add('red_blood_cell_count') # using add we can add the column
controls.add('packed_cell_volume')
controls.add('white_blood_cell_count')
print(controls)

```

```

('blood_sugar', 'serum_creatinine', 'packed_cell_volume', 'blood_pressure', 'blood glucose random', 'sodium', 'hemoglobin', 'red_blood_cell_count', 'age', 'potassium', 'white_blood_cell_count')

```

```

catcols.add('specific_gravity')
catcols.add('albumin')
catcols.add('sugar')
print(catcols)

```

```

('hypertension', 'class', 'albumin', 'coronary_artery_disease', 'anemia', 'sugar', 'red_blood_cells', 'specific_gravity', 'bacteria', 'pedal_sore', 'appetite', 'pus_cell', 'diabetesmellitus', 'pus_cell_clump')

```

In our data some columns some unwanted classes so we have to rectify that also for that we simply use **replace()**


```
1 data['coronary_artery_disease'] = data.coronary_artery_disease.replace({'\tno':'no'}) # replacing \tno w/
2 c(data['coronary_artery_disease'])
```

```
counter({'no': 344, 'yes': 34, nan: 1})
```

```
1 data['diabetesmellitus'] = data.diabetesmellitus.replace(to_replace={'\tno':'no','\tyes':'yes',' yes':''})
2 c(data['diabetesmellitus'])
```

```
counter({'yes': 137, 'no': 341, nan: 1})
```