THYROID DISEASE CLASSIFICATION USING ML

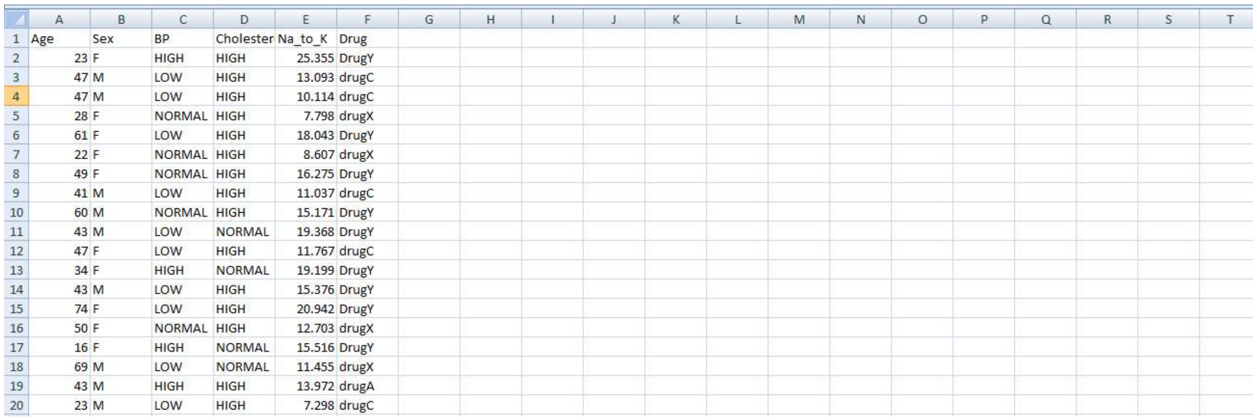
MILESTONE 1: Data Collection & Preparation

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 drug200.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

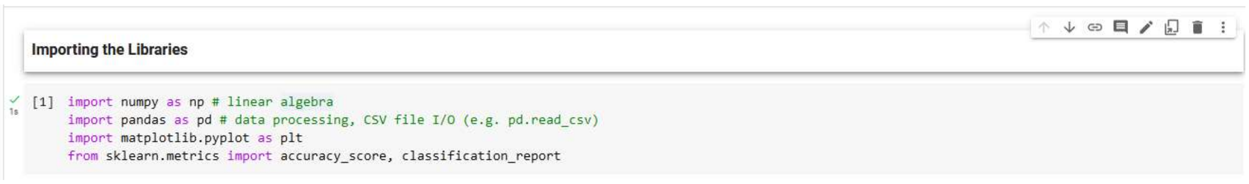
⇨ Link: https://www.kaggle.com/prathamtripathi/drug-classification.



	A	B	C	D	E	F
1	Age	Sex	BP	Cholester	Na_to_K	Drug
2	23	F	HIGH	HIGH	25.355	DrugY
3	47	M	LOW	HIGH	13.093	drugC
4	47	M	LOW	HIGH	10.114	drugC
5	28	F	NORMAL	HIGH	7.798	drugX
6	61	F	LOW	HIGH	18.043	DrugY
7	22	F	NORMAL	HIGH	8.607	drugX
8	49	F	NORMAL	HIGH	16.275	DrugY
9	41	M	LOW	HIGH	11.037	drugC
10	60	M	NORMAL	HIGH	15.171	DrugY
11	43	M	LOW	NORMAL	19.368	DrugY
12	47	F	LOW	HIGH	11.767	drugC
13	34	F	HIGH	NORMAL	19.199	DrugY
14	43	M	LOW	HIGH	15.376	DrugY
15	74	F	LOW	HIGH	20.942	DrugY
16	50	F	NORMAL	HIGH	12.703	drugX
17	16	F	HIGH	NORMAL	15.516	DrugY
18	69	M	LOW	NORMAL	11.455	drugX
19	43	M	HIGH	HIGH	13.972	drugA
20	23	M	LOW	HIGH	7.298	drugC

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image



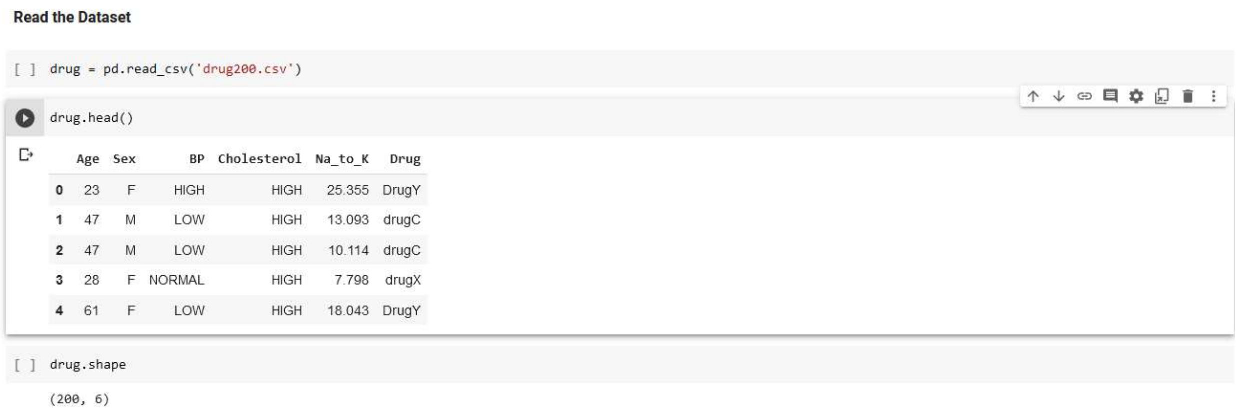
Importing the Libraries

```
[1] import numpy as np # linear algebra
    import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
    import matplotlib.pyplot as plt
    from sklearn.metrics import accuracy_score, classification_report
```

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



Read the Dataset

```
[ ] drug = pd.read_csv('drug200.csv')
```

```
drug.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY

```
[ ] drug.shape

(200, 6)
```

Activity2: Data Pre-processing

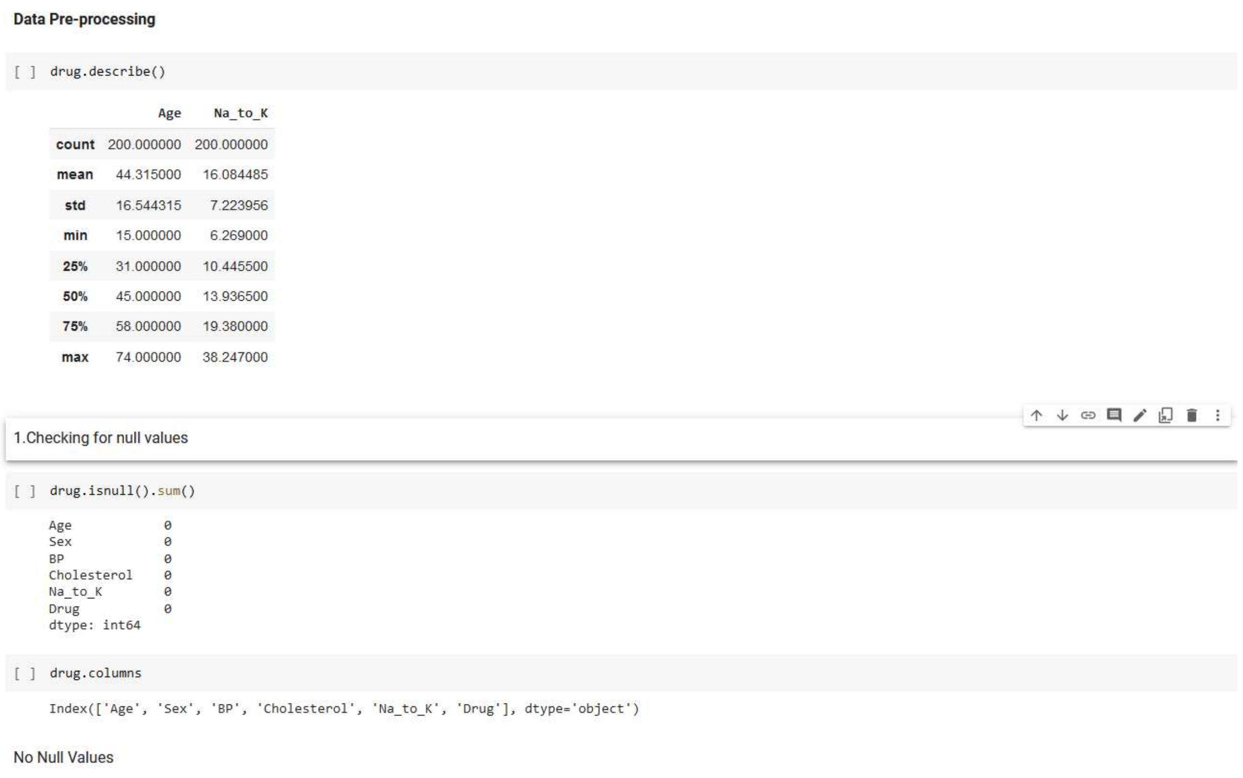
As we have understood how the data is, let's pre-process the collected data. The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Descriptive analysis
- Splitting the dataset as x and y
- Handling Categorical Values
- Splitting dataset into training and test set

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: Checking for null values

⇨ For checking the null values, data.isnull() function is used. To sum those null values we use the .sum() function to it. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.



Data Pre-processing

```
[ ] drug.describe()
```

	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

1.Checking for null values

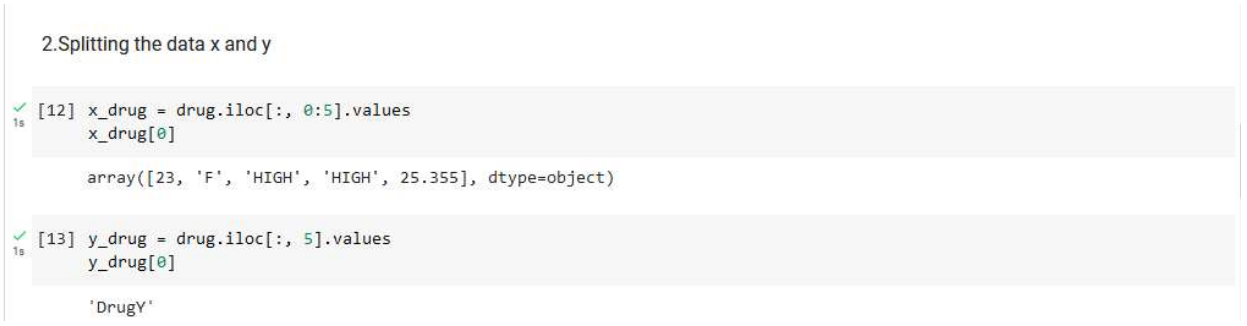
```
[ ] drug.isnull().sum()

    Age            0
    Sex            0
    BP             0
    Cholesterol    0
    Na_to_K        0
    Drug           0
    dtype: int64

[ ] drug.columns

    Index(['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug'], dtype='object')
```

No Null Values

⇨ NO null values, stop the checking for null values.

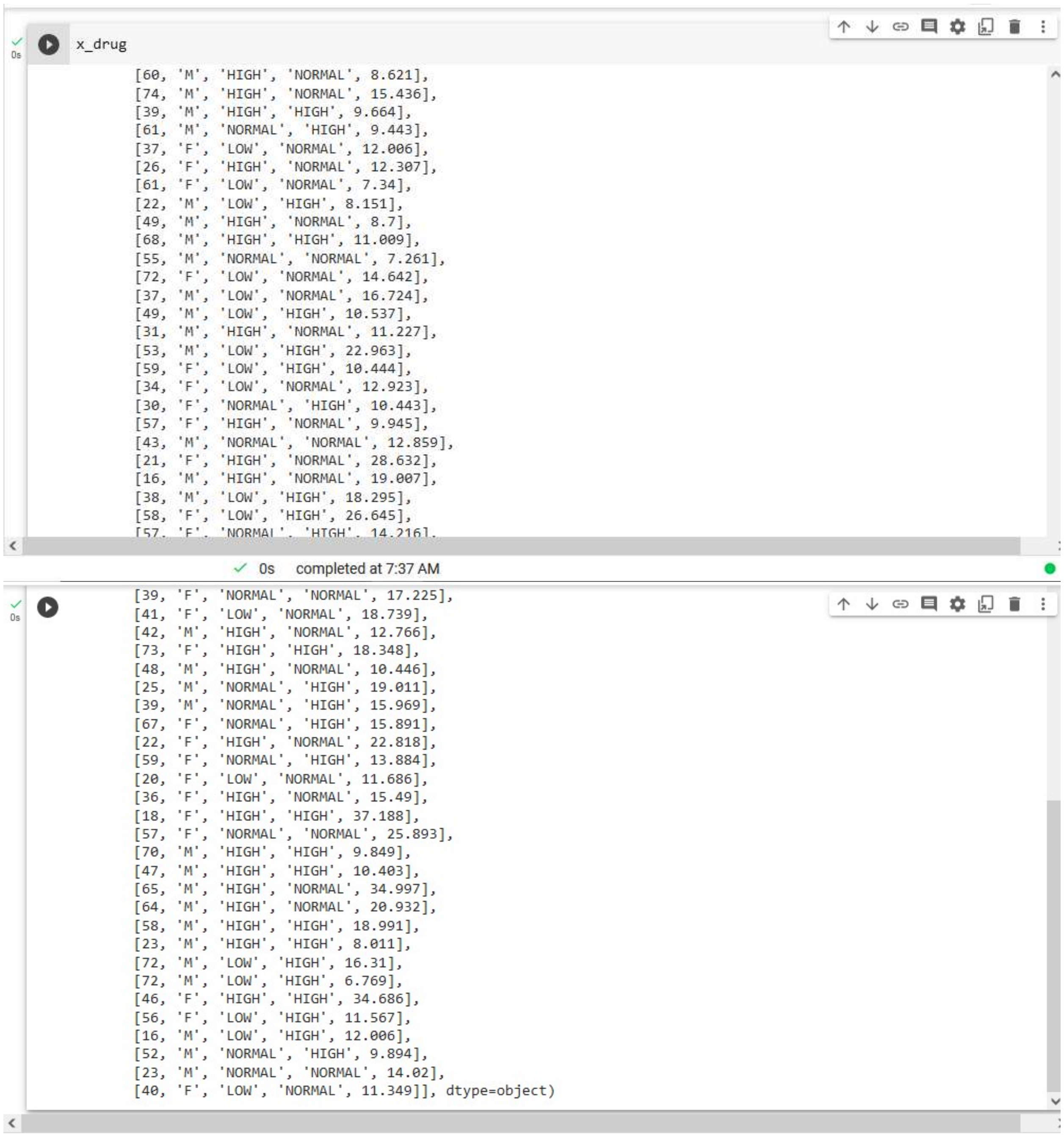
Activity 2.2: Splitting the data x and y

⇨ Splitting the data x and y



2.Splitting the data x and y

```
[12] x_drug = drug.iloc[:, 0:5].values
     x_drug[0]

     array([23, 'F', 'HIGH', 'HIGH', 25.355], dtype=object)

[13] y_drug = drug.iloc[:, 5].values
     y_drug[0]

     'DrugY'
```



```
x_drug
       [60, 'M', 'HIGH', 'NORMAL', 8.621],
       [74, 'M', 'HIGH', 'NORMAL', 15.436],
       [39, 'M', 'HIGH', 'HIGH', 9.664],
       [61, 'M', 'NORMAL', 'HIGH', 9.443],
       [37, 'F', 'LOW', 'NORMAL', 12.006],
       [26, 'F', 'HIGH', 'NORMAL', 12.307],
       [61, 'F', 'LOW', 'NORMAL', 7.34],
       [22, 'M', 'LOW', 'HIGH', 8.151],
       [49, 'M', 'HIGH', 'NORMAL', 8.7],
       [68, 'M', 'HIGH', 'HIGH', 11.009],
       [55, 'M', 'NORMAL', 'NORMAL', 7.261],
       [72, 'F', 'LOW', 'NORMAL', 14.642],
       [37, 'M', 'LOW', 'NORMAL', 16.724],
       [49, 'M', 'LOW', 'HIGH', 10.537],
       [31, 'M', 'HIGH', 'NORMAL', 11.227],
       [53, 'M', 'LOW', 'HIGH', 22.963],
       [59, 'F', 'LOW', 'HIGH', 10.444],
       [34, 'F', 'LOW', 'NORMAL', 12.923],
       [30, 'F', 'NORMAL', 'HIGH', 10.443],
       [57, 'F', 'HIGH', 'NORMAL', 9.945],
       [43, 'M', 'NORMAL', 'NORMAL', 12.859],
       [21, 'F', 'HIGH', 'NORMAL', 28.632],
       [16, 'M', 'HIGH', 'NORMAL', 19.007],
       [38, 'M', 'LOW', 'HIGH', 18.295],
       [58, 'F', 'LOW', 'HIGH', 26.645],
       [57, 'F', 'NORMAL', 'HIGH', 14.216],
```

✓ 0s completed at 7:37 AM

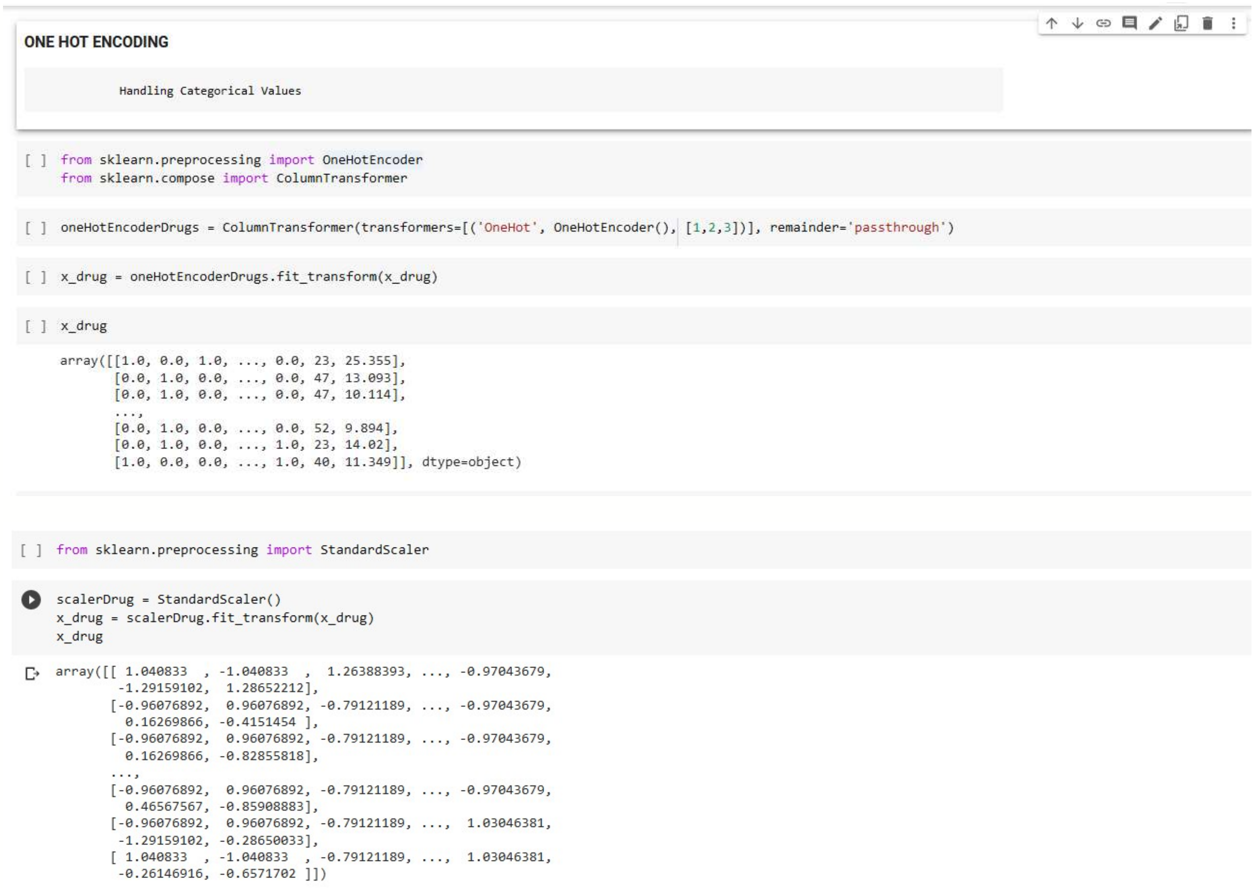
```
       [39, 'F', 'NORMAL', 'NORMAL', 17.225],
       [41, 'F', 'LOW', 'NORMAL', 18.739],
       [42, 'M', 'HIGH', 'NORMAL', 12.766],
       [73, 'F', 'HIGH', 'HIGH', 18.348],
       [48, 'M', 'HIGH', 'NORMAL', 10.446],
       [25, 'M', 'NORMAL', 'HIGH', 19.011],
       [39, 'M', 'NORMAL', 'HIGH', 15.969],
       [67, 'F', 'NORMAL', 'HIGH', 15.891],
       [22, 'F', 'HIGH', 'NORMAL', 22.818],
       [59, 'F', 'NORMAL', 'HIGH', 13.884],
       [20, 'F', 'LOW', 'NORMAL', 11.686],
       [36, 'F', 'HIGH', 'NORMAL', 15.49],
       [18, 'F', 'HIGH', 'HIGH', 37.188],
       [57, 'F', 'NORMAL', 'NORMAL', 25.893],
       [70, 'M', 'HIGH', 'HIGH', 9.849],
       [47, 'M', 'HIGH', 'HIGH', 10.403],
       [65, 'M', 'HIGH', 'NORMAL', 34.997],
       [64, 'M', 'HIGH', 'NORMAL', 20.932],
       [58, 'M', 'HIGH', 'HIGH', 18.991],
       [23, 'M', 'HIGH', 'HIGH', 8.011],
       [72, 'M', 'LOW', 'HIGH', 16.31],
       [72, 'M', 'LOW', 'HIGH', 6.769],
       [46, 'F', 'HIGH', 'HIGH', 34.686],
       [56, 'F', 'LOW', 'HIGH', 11.567],
       [16, 'M', 'LOW', 'HIGH', 12.006],
       [52, 'M', 'NORMAL', 'HIGH', 9.894],
       [23, 'M', 'NORMAL', 'NORMAL', 14.02],
       [40, 'F', 'LOW', 'NORMAL', 11.349]], dtype=object)
```

Activity 2.3: 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 OneHotEncoder and ColumnTransformer.

- In our project, categorical features are x and y values.
- Here, applying Ordinal Encoding on x values.



ONE HOT ENCODING

```
Handling Categorical Values
```

```
[ ] from sklearn.preprocessing import OneHotEncoder
    from sklearn.compose import ColumnTransformer
```

```
[ ] oneHotEncoderDrugs = ColumnTransformer(transformers=[('OneHot', OneHotEncoder(), [1,2,3])], remainder='passthrough')
```

```
[ ] x_drug = oneHotEncoderDrugs.fit_transform(x_drug)
```

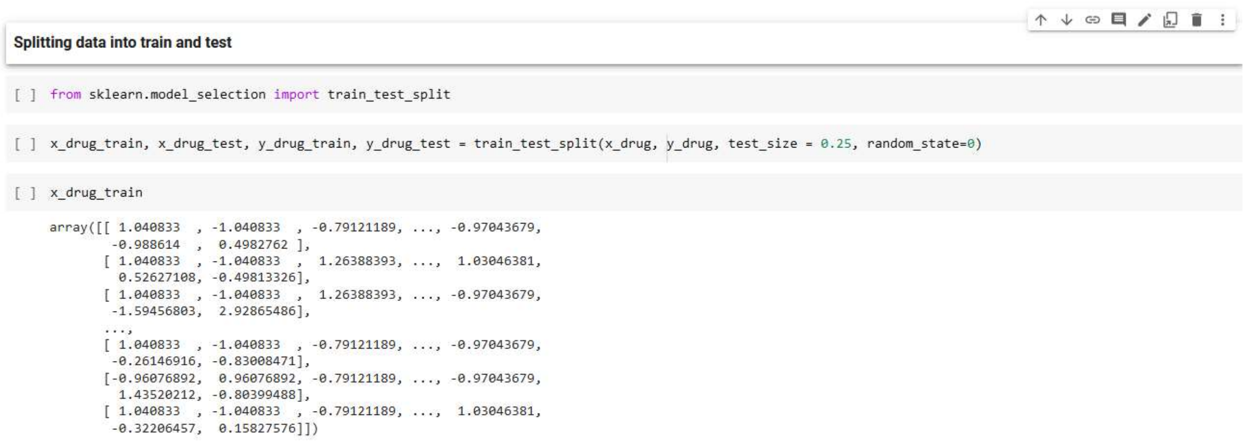
```
[ ] x_drug

    array([[1.0, 0.0, 1.0, ..., 0.0, 23, 25.355],
           [0.0, 1.0, 0.0, ..., 0.0, 47, 13.093],
           [0.0, 1.0, 0.0, ..., 0.0, 47, 10.114],
           ...,
           [0.0, 1.0, 0.0, ..., 0.0, 52, 9.894],
           [0.0, 1.0, 0.0, ..., 1.0, 23, 14.02],
           [1.0, 0.0, 0.0, ..., 1.0, 40, 11.349]], dtype=object)
```

```
[ ] from sklearn.preprocessing import StandardScaler
```

```
scalerDrug = StandardScaler()
x_drug = scalerDrug.fit_transform(x_drug)
x_drug

array([[ 1.040833  , -1.040833  ,  1.26388393, ..., -0.97043679,
        -1.29159102,  1.28652212],
       [-0.96076892,  0.96076892, -0.79121189, ..., -0.97043679,
         0.16269866, -0.4151454 ],
       [-0.96076892,  0.96076892, -0.79121189, ..., -0.97043679,
         0.16269866, -0.82855818],
       ...,
       [-0.96076892,  0.96076892, -0.79121189, ..., -0.97043679,
         0.46567567, -0.85908883],
       [-0.96076892,  0.96076892, -0.79121189, ...,  1.03046381,
        -1.29159102, -0.28650033],
       [ 1.040833  , -1.040833  , -0.79121189, ...,  1.03046381,
        -0.26146916, -0.6571702 ]])
```

Activity 2.4: 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.
- ⇨ Here x and y variables are created. On x variable, data is passed with dropping the target variable. And my 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.



Splitting data into train and test

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] x_drug_train, x_drug_test, y_drug_train, y_drug_test = train_test_split(x_drug, y_drug, test_size = 0.25, random_state=0)
```

```
[ ] x_drug_train

    array([[ 1.040833  , -1.040833  , -0.79121189, ..., -0.97043679,
            -0.988614  ,  0.4982762 ],
           [ 1.040833  , -1.040833  ,  1.26388393, ...,  1.03046381,
             0.52627108, -0.49813326],
           [ 1.040833  , -1.040833  ,  1.26388393, ..., -0.97043679,
            -1.59456803,  2.92865486],
           ...,
           [ 1.040833  , -1.040833  , -0.79121189, ..., -0.97043679,
            -0.26146916, -0.83008471],
           [-0.96076892,  0.96076892, -0.79121189, ..., -0.97043679,
             1.43520212, -0.80399488],
           [ 1.040833  , -1.040833  , -0.79121189, ...,  1.03046381,
            -0.32206457,  0.15827576]])
```