

Assignment: 5

AIM : To Perform Data Preprocessing and model building.

PROBLEM STATEMENT /DEFINITION

Perform the following operations using Python on the Air quality and Heart Diseases data sets

1. Data cleaning
2. Data integration
3. Data transformation
4. Error correcting
5. Data model building

OBJECTIVE

- To understand the concept of Data Preprocessing
- To understand the methods of Data preprocessing.

THEORY:

Data Preprocessing:

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Preprocessing of data is mainly to check the data quality. The quality can be checked by the following

- Accuracy: To check whether the data entered is correct or not.
- Completeness: To check whether the data is available or not recorded.
- Consistency: To check whether the same data is kept in all the places that do or do not match.
- Timeliness: The data should be updated correctly.
- Believability: The data should be trustable.
- Interpretability: The understandability of the data

Major Tasks in Data Preprocessing:

1. Data cleaning
2. Data integration
3. Data transformation
4. Error correcting
5. Data model building



Let's discuss each of these points in detail:

Before Data Preprocessing. The necessary libraries and dataset have to be imported.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
df = pd.read_csv('heart.csv')
```

1.Data cleaning:

Data Cleaning means the process of identifying the incorrect, incomplete, inaccurate, irrelevant or missing part of the data and then modifying, replacing or deleting them according to the necessity. Data cleaning is considered a foundational element of basic data science.

Different Ways of Cleaning Data

Changing the Datatype of the columns:

The variables types are in heart dataset are:

- Binary: sex, fbs, exang, target
- Categorical: cp, restecg, slope, ca, thal
- Continuous: age, trestbps, chol, thalac, oldpeak

But, df.info() gives the following output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   age         303 non-null    int64
 1   sex         303 non-null    int64
 2   cp          303 non-null    int64
 3   trestbps    303 non-null    int64
 4   chol        303 non-null    int64
 5   fbs         303 non-null    int64
 6   restecg     303 non-null    int64
 7   thalach     303 non-null    int64
 8   exang       303 non-null    int64
 9   oldpeak     303 non-null    float64
10   slope       303 non-null    int64
11   ca          303 non-null    int64
12   thal        303 non-null    int64
13   target      303 non-null    int64
```

```
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Hence, We will change the Datatypes of columns to appropriate types.

```
df['sex'] = df['sex'].astype('object')
df['cp'] = df['cp'].astype('object')
df['fbs'] = df['fbs'].astype('object')
df['restecg'] = df['restecg'].astype('object')
df['exang'] = df['exang'].astype('object')
df['slope'] = df['slope'].astype('object')
df['ca'] = df['ca'].astype('object')
df['thal'] = df['thal'].astype('object')
df.dtypes
```

Inconsistent column :

If your DataFrame contains columns that are irrelevant or you are never going to use them then you can drop them to give more focus on the columns you will work on. Let's see an example of how to deal with such a data set.

From the Air Quality dataset,

Dropping of less valued columns: stn_code, agency, sampling_date, location_monitoring_ do not add much value to the dataset in terms of information. Therefore, we can drop those columns.

Changing the types to uniform format: When you see the dataset, you may notice that the 'type' column has values such as 'Industrial Area' and 'Industrial Areas' — both actually mean the same, so let's remove such type of stuff and make it uniform.

Creating a year column To view the trend over a period of time, we need year values for each row and also when you see in most of the values in date column only has 'year' value. So, let's create a new column holding year values.

1.stn_code, agency, sampling_date, location_monitoring_ agency do not add much value to the dataset in terms of information. Therefore, we can drop those columns.

2.Dropping rows where no date is available.

```
df=df.drop(['stn_code',
'agency','sampling_date','location_monitoring_station'], axis = 1)
#dropping columns that aren't required
df=df.dropna(subset=['date']) # dropping rows where no date is available
```

Missing data:

It is rare to have a real world dataset without having any missing values. When you start to work with real world data, you will find that most of the dataset contains missing values. Handling missing values is very important because if you leave the missing values as it is, it may affect your analysis and machine learning models. So, you need to be sure that your dataset contains missing values or not. If you find missing values in your dataset you must handle it. If you find any missing values in the dataset you can perform any of these three task on it:

1. Leave as it is
2. Filling the missing values
3. Drop them

For filling the missing values we can perform different methods. For example, airquality dataset has missing values.

The column such as SO₂, NO₂, rspm, spm, pm_{2.5} are the ones which contribute much to our analysis. So, we need to remove null from those columns to avoid inaccuracy in the prediction. We use the Imputer from sklearn.preprocessing to fill the missing values in every column with the mean.

```
# defining columns of importance, which shall be used reguarly
COLS = ['so2', 'no2', 'rspm', 'spm', 'pm2_5']
from sklearn.impute import SimpleImputer
# invoking SimpleImputer to fill missing values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
df[COLS] = imputer.fit_transform(df[COLS])
```

Outliers:

*“In statistics, an **outlier** is a data point that differs significantly from other observations.”*

That means an outlier indicates a data point that is significantly different from the other data points in the data set. Outliers can be created due to the errors in the experiments or the variability in the measurements. Let's look at an example to clear the concept.

So, now the question is how can we detect the outliers in the dataset.

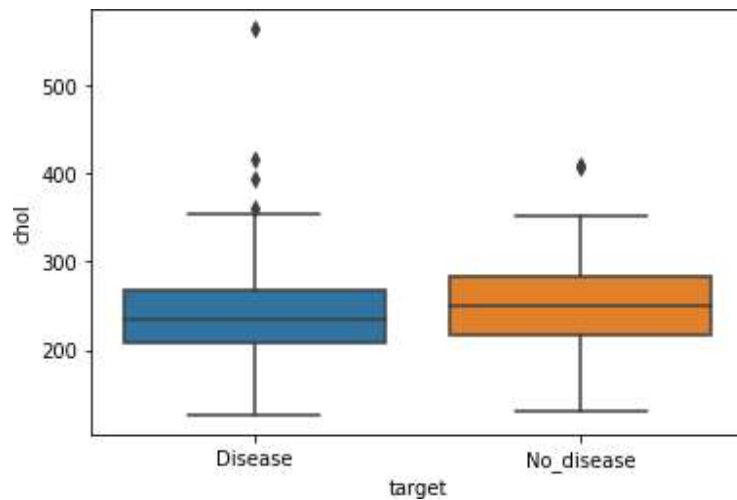
For detecting the outliers we can use :

1. Box Plot
2. Scatter plot
3. Z-score etc.

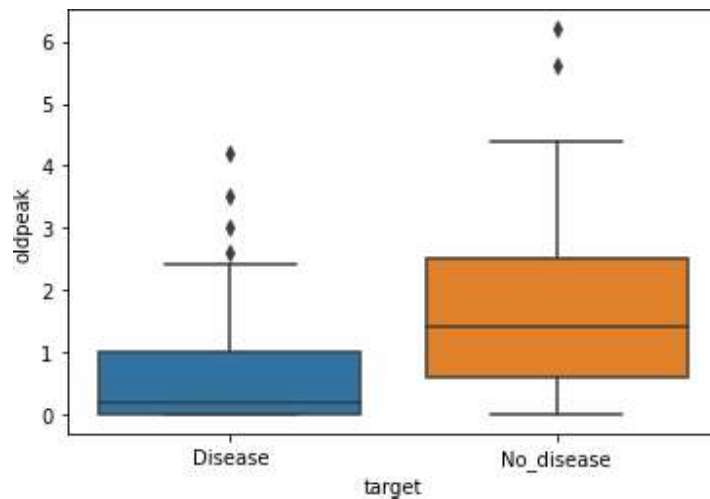
Before we plot the outliers, let's change the labeling for better visualization and interpretation for heart dataset.

```
df['target'] = df.target.replace({1: "Disease", 0: "No_disease"})
df['sex'] = df.sex.replace({1: "Male", 0: "Female"})
df['cp'] = df.cp.replace({0: "typical_angina",
                        1: "atypical_angina",
                        2: "non-anginal pain",
                        3: "asymtomatic"})
df['exang'] = df.exang.replace({1: "Yes", 0: "No"})
df['fbs'] = df.fbs.replace({1: "True", 0: "False"})
df['slope'] = df.slope.replace({0: "upsloping", 1: "flat", 2: "downsloping"})
df['thal'] = df.thal.replace({1: "fixed_defect", 2: "reversible_defect", 3: "normal"})

import matplotlib.pyplot as plt
import seaborn as sns
bxplt = sb.boxplot(df["target"], df["chol"])
plt.show()
```



```
sb.boxplot(x='target', y='oldpeak', data=df)
<matplotlib.axes._subplots.AxesSubplot at 0x7f95fceb7e10>
```



```
define continuous variable & plot
continuous_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
def outliers(df_out, drop = False):
    for each_feature in df_out.columns:
        feature_data = df_out[each_feature]
        Q1 = np.percentile(feature_data, 25.) # 25th percentile of the
data of the given feature
        Q3 = np.percentile(feature_data, 75.) # 75th percentile of the
data of the given feature
        IQR = Q3-Q1 #Interquartile Range
        outlier_step = IQR * 1.5 #That's we were talking about above
```

```

        outliers = feature_data[~((feature_data >= Q1 - outlier_step) &
(feature_data <= Q3 + outlier_step))].index.tolist()
        if not drop:
            print('For the feature {}, No of Outliers is
{}'.format(each_feature, len(outliers)))
        if drop:
            df.drop(outliers, inplace = True, errors = 'ignore')
            print('Outliers from {} feature removed'.format(each_feature))

outliers(df[continous_features])

```

```

For the feature age, No of Outliers is 0
For the feature trestbps, No of Outliers is 9
For the feature chol, No of Outliers is 5
For the feature thalach, No of Outliers is 1
For the feature oldpeak, No of Outliers is 5

```

Drop Outliers

```

outliers(df[continous_features],drop=True)
Outliers from age feature removed
Outliers from trestbps feature removed
Outliers from chol feature removed
Outliers from thalach feature removed
Outliers from oldpeak feature removed

```

Duplicate rows:

Datasets may contain duplicate entries. It is one of the easiest tasks to delete duplicate rows. To delete the duplicate rows you can use —*dataset_name.drop_duplicates()*.

We have used another approach on the heart dataset.

Checking for the duplicate rows

```

duplicated=df.duplicated().sum()

if duplicated:

    print("Duplicated rows :{}".format(duplicated))

else:

    print("No duplicates")

```



```
Duplicated rows :1
```

Displaying duplicate rows

```
duplicates=df[df.duplicated(keep=False)]  
  
duplicates.head()
```

Remove the duplicate rows using the above mentioned method.

2.Data integration

So far, we've made sure to remove the impurities in data and make it clean. Now, the next step is to combine data from different sources to get a unified structure with more meaningful and valuable information. This is mostly used if the data is segregated into different sources. To make it simple, let's assume we have data in CSV format in different places, all talking about the same scenario. Say we have some data about an employee in a database. We can't expect all the data about the employee to reside in the same table. It's possible that the employee's personal data will be located in one table, the employee's project history will be in a second table, the employee's time-in and time-out details will be in another table, and so on. So, if we want to do some analysis about the employee, we need to get all the employee data in one common place. This process of bringing data together in one place is called data integration. To do data integration, we can merge multiple pandas DataFrames using the merge function.

In this exercise, we'll merge the details of students from two datasets, namely student.csv and marks.csv. The student dataset contains columns such as Age, Gender, Grade, and Employed. The marks.csv dataset contains columns such as Mark and City. The Student_id column is common between the two datasets. Follow these steps to complete this exercise:

```
df1=pd.read_excel('student.xlsx',header=0)  
df2=pd.read_excel('mark.xlsx',header=0)
```

To print the first five rows of the first DataFrame, add the following code:

```
df1.head()
```

The preceding code generates the following output:

	Student_id	Mark	City
0	1	95	Chennai
1	2	70	Delhi
2	3	98	Mumbai
3	4	75	Pune
4	5	89	Kochi

Figure : The first five rows of the first DataFrame

To print the first five rows of the second DataFrame, add the following code:

```
df2.head()
```

The preceding code generates the following output:

	Student_id	Age	Gender	Grade	Employed
0	1	19	Male	1st Class	yes
1	2	20	Female	2nd Class	no
2	3	18	Male	1st Class	no
3	4	21	Female	2nd Class	no
4	5	19	Male	1st Class	no

Figure : The first five rows of the second DataFrame

Student_id is common to both datasets. Perform data integration on both the DataFrames with respect to the Student_id column using the pd.merge() function, and then print the first 10 values of the new DataFrame:

```
df=pd.merge(df1,df2,on='Student_id')
df.head(10)
```

	Student_id	Mark	City	Age	Gender	Grade	Employed
0	1	95	Chennai	19	Male	1st Class	yes
1	2	70	Delhi	20	Female	2nd Class	no
2	3	98	Mumbai	18	Male	1st Class	no
3	4	75	Pune	21	Female	2nd Class	no
4	5	89	Kochi	19	Male	1st Class	no
5	6	76	Kolkata	17	Female	2nd Class	yes
6	7	81	Bengalore	18	Male	1st Class	yes
7	8	99	Bhopal	19	Female	2nd Class	yes
8	9	73	Indore	16	Male	1st Class	yes
9	10	87	Surat	18	Female	2nd Class	yes

Figure : First 10 rows of the merged DataFrame

Here, the data of the df1 DataFrame is merged with the data of the df2 DataFrame. The merged data is stored inside a new DataFrame called df.

We have now learned how to perform data integration. In the next section, we'll explore another pre-processing task, data transformation.

3.Data transformation

Previously, we saw how we can combine data from different sources into a unified dataframe. Now, we have a lot of columns that have different types of data. Our goal is to transform the data into a machine-learning-digestible format. All machine learning algorithms are based on mathematics. So, we need to convert all the columns into numerical format. Before that, let's see all the different types of data we have.

Taking a broader perspective, data is classified into numerical and categorical data:

- Numerical: As the name suggests, this is numeric data that is quantifiable.
- Categorical: The data is a string or non-numeric data that is qualitative in nature.

Numerical data is further divided into the following:

- Discrete: To explain in simple terms, any numerical data that is countable is called discrete, for example, the number of people in a family or the number of students in a class. Discrete data can only take certain values (such as 1, 2, 3, 4, etc).
- Continuous: Any numerical data that is measurable is called continuous, for example, the height of a person or the time taken to reach a destination. Continuous data can take virtually any value (for example, 1.25, 3.8888, and 77.1276).

Categorical data is further divided into the following:

- Ordered: Any categorical data that has some order associated with it is called ordered categorical data, for example, movie ratings (excellent, good, bad, worst) and feedback (happy, not bad, bad). You can think of ordered data as being something you could mark on a scale.
- Nominal: Any categorical data that has no order is called nominal categorical data. Examples include gender and country.

From these different types of data, we will focus on categorical data.

Handling Categorical Data

There are some algorithms that can work well with categorical data, such as decision trees. But most machine learning algorithms cannot operate directly with categorical data. These algorithms require the input and output both to be in numerical form. If the output to be predicted is categorical, then after prediction we convert them back to categorical data from numerical data. Let's discuss some key challenges that we face while dealing with categorical data:

- High cardinality: Cardinality means uniqueness in data. The data column, in this case, will have a lot of different values. A good example is User ID – in a table of 500 different users, the User ID column would have 500 unique values.
- Rare occurrences: These data columns might have variables that occur very rarely and therefore would not be significant enough to have an impact on the model.
- Frequent occurrences: There might be a category in the data columns that occurs many times with very low variance, which would fail to make an impact on the model.
- Won't fit: This categorical data, left unprocessed, won't fit our model.

Encoding

To address the problems associated with categorical data, we can use encoding. This is the process by which we convert a categorical variable into a numerical form. Here, we will look at three simple methods of encoding categorical data.

Replacing

This is a technique in which we replace the categorical data with a number. This is a simple replacement and does not involve much logical processing. Let's look at an exercise to get a better idea of this.

Exercise 6: Simple Replacement of Categorical Data with a Number

In this exercise, we will use the student dataset that we saw earlier. We will load the data into a pandas dataframe and simply replace all the categorical data with numbers. Follow these steps to complete this exercise:

In AirQuality dataset, applying Simple Replacement of Categorical Data with a Number.

```
df['type'].value_counts()
RRO      179013
I         148069
RO        86791
S         15010
RIRUO     1304
R           158
```

The column type in the dataframe has 6 unique values, which we will be replacing with numbers.

```
df['type'].replace({"RRO":1, "I":2, "RO":3, "S":4, "RIRUO":5, "R":6},
inplace= True)
df['type']
0      1.0
1      2.0
2      1.0
```

```

3          1.0
4          2.0
...
435734     5.0
435735     5.0
435736     5.0
435737     5.0
435738     5.0
Name: type, Length: 435735, dtype: float64

```

Converting Categorical Data to Numerical Data Using Label Encoding

This is a technique in which we replace each value in a categorical column with numbers from 0 to N-1. For example, say we've got a list of employee names in a column. After performing label encoding, each employee name will be assigned a numeric label. But this might not be suitable for all cases because the model might consider numeric values to be weights assigned to the data. Label encoding is the best method to use for ordinal data. The scikit-learn library provides `LabelEncoder()`, which helps with label encoding.

```

df['state'].value_counts()

```

Maharashtra	60382
Uttar Pradesh	42816
Andhra Pradesh	26368
Punjab	25634
Rajasthan	25589
Kerala	24728
Himachal Pradesh	22896
West Bengal	22463
Gujarat	21279
Tamil Nadu	20597
Madhya Pradesh	19920
Assam	19361
Odisha	19278
Karnataka	17118
Delhi	8551
Chandigarh	8520
Chhattisgarh	7831
Goa	6206
Jharkhand	5968
Mizoram	5338
Telangana	3978
Meghalaya	3853
Puducherry	3785

```
Haryana          3420
Nagaland         2463
Bihar            2275
Uttarakhand     1961
Jammu & Kashmir  1289
Daman & Diu      782
Dadra & Nagar Haveli 634
Uttaranchal     285
Arunachal Pradesh 90
Manipur          76
Sikkim           1
```

```
Name: state, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
df["state"]=labelencoder.fit_transform(df["state"])
df.head(5)
```

	state	location	type	so2	no2	rspm	spm	pm2_5	date	year
0	0	Hyderabad	1.0	4.8	17.4	108.833091	220.78348	40.791467	1990-02-01	1990
1	0	Hyderabad	2.0	3.1	7.0	108.833091	220.78348	40.791467	1990-02-01	1990
2	0	Hyderabad	1.0	6.2	28.5	108.833091	220.78348	40.791467	1990-02-01	1990
3	0	Hyderabad	1.0	6.3	14.7	108.833091	220.78348	40.791467	1990-03-01	1990
4	0	Hyderabad	2.0	4.7	7.5	108.833091	220.78348	40.791467	1990-03-01	1990

One Hot Encoding

In label encoding, categorical data is converted to numerical data, and the values are assigned labels (such as 1, 2, and 3). Predictive models that use this numerical data for analysis might sometimes mistake these labels for some kind of order (for example, a model might think that a label of 3 is "better" than a label of 1, which is incorrect). In order to avoid this confusion, we can use one-hot encoding. Here, the label-encoded data is further divided into n number of columns. Here, n denotes the total number of unique labels generated while performing label encoding. For example, say that three new labels are generated through label encoding. Then,

while performing one-hot encoding, the columns will be divided into three parts. So, the value of n is 3.

```
dfAndhra=df[(df['state']==0)]
```

```
dfAndhra
```

	state	location	type	so2	no2	rspm	spm	pm2_5	date	year
0	0	Hyderabad	1.0	4.8	17.4	108.833091	220.78348	40.791467	1990-02-01	1990
1	0	Hyderabad	2.0	3.1	7.0	108.833091	220.78348	40.791467	1990-02-01	1990
2	0	Hyderabad	1.0	6.2	28.5	108.833091	220.78348	40.791467	1990-02-01	1990
3	0	Hyderabad	1.0	6.3	14.7	108.833091	220.78348	40.791467	1990-03-01	1990
4	0	Hyderabad	2.0	4.7	7.5	108.833091	220.78348	40.791467	1990-03-01	1990
...
26363	0	Rajahmundry	2.0	7.0	13.0	71.000000	220.78348	40.791467	2015-12-13	2015
26364	0	Rajahmundry	2.0	7.0	18.0	77.000000	220.78348	40.791467	2015-12-16	2015
26365	0	Rajahmundry	2.0	8.0	23.0	64.000000	220.78348	40.791467	2015-12-19	2015
26366	0	Rajahmundry	2.0	7.0	19.0	61.000000	220.78348	40.791467	2015-12-22	2015

```
dfAndhra['location'].value_counts()
```

```
Hyderabad      7764
Visakhapatnam  7108
Vijayawada     2093
Chittoor       1003
Tirupati       986
Kurnool        857
Patancheru     698
Guntur         629
Nalgonda       618
Ramagundam     554
Nellore        408
Khammam        385
Warangal       336
Ananthapur     324
Ongole         317
Kadapa         316
Srikakulam     315
Rajahmundry    311
Eluru          300
Vishakhapatnam 297
Kakinada       288
Vizianagaram   282
Sangareddy     85
Karimnagar     67
Nizamabad      27
Name: location, dtype: int64
```

```
from sklearn.preprocessing import OneHotEncoder
onehotencoder=OneHotEncoder(sparse=False,handle_unknown='error',drop='first')
pd.DataFrame(onehotencoder.fit_transform(dfAndhra[["location"]]))
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...
26363	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26364	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26365	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26366	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26367	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

26368 rows × 24 columns

You have successfully converted categorical data to numerical data using the OneHotEncoder method.

4. Error Correction

In heart dataset it can be observed that feature 'ca' should range from 0–3, however, df.nunique() listed 0–4. So let's find the '4' and change them to NaN.

```
df['ca'].unique()
```

```
array([0, 2, 1, 3, 4], dtype=object)
```

```
df[df['ca']==4]
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
92	52	1	2	138	223	0	1	169	0	0.0	2	4	2	1
158	58	1	1	125	220	0	1	144	0	0.4	1	4	3	1
163	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1
164	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1
251	43	1	0	132	247	1	0	143	1	0.1	1	4	3	0

```
df.loc[df['ca']==4, 'ca']=np.NaN
```

Similarly, Feature ‘thal’ ranges from 1–3, however, df.nunique() listed 0–3. There are two values of ‘0’. They are also changed to NaN using above mentioned technique.

Now, we can replace changed NaN values(missing values).

```
df.isna().sum()
```

```
age      0
sex      0
cp       0
trestbps 0
chol     0
fbs      0
restecg  0
thalach  0
exang    0
oldpeak  0
slope    0
ca       5
thal     2
target   0
dtype: int64
```

```
df = df.fillna(df.median())
```

```
df.isnull().sum()
```

```
age      0
sex      0
cp       0
trestbps 0
chol     0
fbs      0
restecg  0
thalach  0
exang    0
oldpeak  0
slope    0
ca       0
thal     0
target   0
dtype: int64
```

5.Data model building

Once you've pre-processed your data into a format that's ready to be used by your model, you need to split up your data into train and test sets. This is because your machine learning algorithm will use the data in the training set to learn what it needs to know. It will then make a prediction about the data in the test set, using what it has learned. You can then compare this prediction against the actual target variables in the test set in order to see how accurate your model is. The exercise in the next section will give more clarity on this.

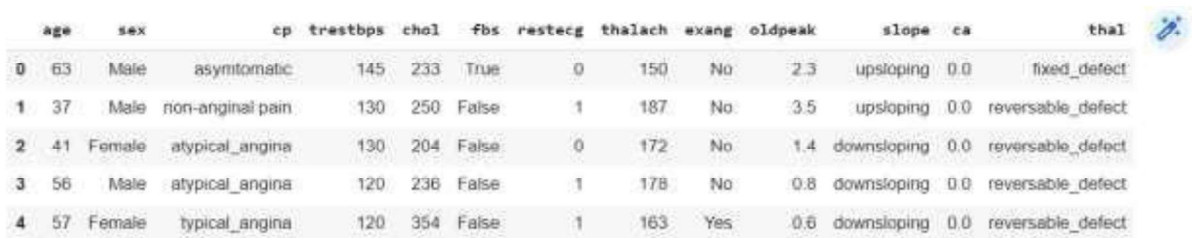
We will do the train/test split in proportions. The larger portion of the data split will be the train set and the smaller portion will be the test set. This will help to ensure that you are using enough data to accurately train your model.

In general, we carry out the train-test split with an 80:20 ratio, as per the Pareto principle. The Pareto principle states that "for many events, roughly 80% of the effects come from 20% of the causes." But if you have a large dataset, it really doesn't matter whether it's an 80:20 split or 90:10 or 60:40. (It can be better to use a smaller split set for the training set if our process is computationally intensive, but it might cause the problem of overfitting – this will be covered later in the book.)

Create a variable called X to store the independent features. Use the drop() function to include all the features, leaving out the dependent or the target variable, which in this case is named 'target' for heart dataset. Then, print out the top five instances of the variable. Add the following code to do this:

```
X = df.drop('target', axis=1)
```

```
X.head()
```



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	Male	asymptomatic	145	233	True	0	150	No	2.3	upsloping	0.0	fixed_defect
1	37	Male	non-anginal pain	130	250	False	1	187	No	3.5	upsloping	0.0	reversible_defect
2	41	Female	atypical_angina	130	204	False	0	172	No	1.4	downsloping	0.0	reversible_defect
3	56	Male	atypical_angina	120	236	False	1	178	No	0.8	downsloping	0.0	reversible_defect
4	57	Female	typical_angina	120	354	False	1	163	Yes	0.6	downsloping	0.0	reversible_defect

Figure : Dataframe consisting of independent variables

1. Print the shape of your new created feature matrix using the X.shape command:

```
X.shape
```

The preceding code generates the following output:

```
(284, 13)
```

Figure : Shape of the X variable

In the preceding figure, the first value indicates the number of observations in the dataset (284), and the second value represents the number of features (13).

2. Similarly, we will create a variable called y that will store the target values. We will use indexing to grab the target column. Indexing allows us to access a section of a larger element. In this case, we want to grab the column named Price from the df dataframe and print out the top 10 values. Add the following code to implement this:

```
y = df['target']
```

y.head(10)

The preceding code generates the following output:

```
0    Disease
1    Disease
2    Disease
3    Disease
4    Disease
5    Disease
6    Disease
7    Disease
9    Disease
10   Disease
Name: target, dtype: object
```

Figure : Top 10 values of the y variable

3. Print the shape of your new variable using the y.shape command:

y.shape

The preceding code generates the following output:

```
(284,)
```

Figure : Shape of the y variable

The shape should be one-dimensional, with a length equal to the number of observations (284).

4. Make train/test sets with an 80:20 split. To do so, use the train_test_split() function from the sklearn.model_selection package. Add the following code to do this:

```
from sklearn.model_selection import train_test_split
```

```
from sklearn import preprocessing
```

```
df=df.apply(preprocessing.LabelEncoder().fit_transform)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

In the preceding code, `test_size` is a floating-point value that defines the size of the test data. If the value is 0.2, then it is an 80:20 split. `train_test_split` splits the arrays or matrices into train and test subsets in a random way. Each time we run the code without `random_state`, we will get a different result.

5. Print the shape of `X_train`, `X_test`, `y_train`, and `y_test`. Add the following code to do this:

```
print("X_train : ",X_train.shape)
print("X_test : ",X_test.shape)
print("y_train : ",y_train.shape)
print("y_test : ",y_test.shape)
```

The preceding code generates the following output:

```
X_train : (227, 13)
X_test : (57, 13)
y_train : (227,)
y_test : (57,)
```

Figure : Shape of train and test datasets

You have successfully split the data into train and test sets.

In the next section, you will complete an activity wherein you'll perform pre-processing on a dataset.

Supervised Learning

Supervised learning is a learning system that trains using labeled data (data in which the target variables are already known). The model learns how patterns in the feature matrix map to the target variables. When the trained machine is fed with a new dataset, it can use what it has learned to predict the target variables. This can also be called predictive modeling.

Supervised learning is broadly split into two categories. These categories are as follows:

1. Classification mainly deals with categorical target variables. A classification algorithm helps to predict which group or class a data point belongs to.

When the prediction is between two classes, it is known as binary classification. An example is predicting whether or not a person has a heart disease (in this case, the classes are yes and no).

If the prediction involves more than two target classes, it is known as multi-classification; for example, predicting all the items that a customer will buy.

2. Regression deals with numerical target variables. A regression algorithm predicts the numerical value of the target variable based on the training dataset.

Linear regression measures the link between one or more predictor variables and one outcome variable. For example, linear regression could help to enumerate the relative impacts of age, gender, and diet (the predictor variables) on height (the outcome variable).

Time series analysis, as the name suggests, deals with data that is distributed with respect to time, that is, data that is in a chronological order. Stock market prediction and customer churn prediction are two examples of time series data. Depending on the requirement or the necessities, time series analysis can be either a regression or classification task.

Unsupervised Learning

Unlike supervised learning, the unsupervised learning process involves data that is neither classified nor labeled. The algorithm will perform analysis on the data without guidance. The job of the machine is to group unclustered information according to similarities in the data. The aim is for the model to spot patterns in the data in order to give some insight into what the data is telling us and to make predictions.

An example is taking a whole load of unlabeled customer data and using it to find patterns to cluster customers into different groups. Different products could then be marketed to the different groups for maximum profitability.

Unsupervised learning is broadly categorized into two types:\

Clustering: A clustering procedure helps to discover the inherent patterns in the data.

Association: An association rule is a unique way to find patterns associated with a large amount of data, such as the supposition that when someone buys product 1, they also tend to buy product2.

CONCLUSION:

This assignment covers the most crucial step in Machine learning is data preprocessing.

