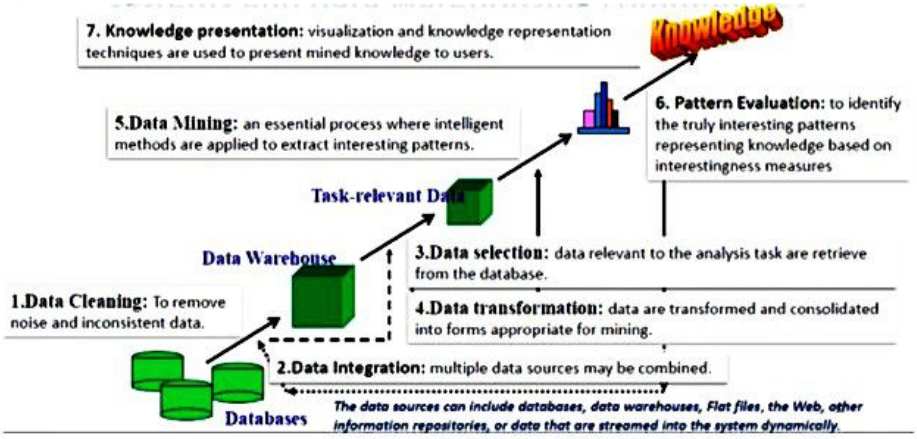
Introduction to Required Libraries

Typical KDD (Knowledge Discovery from Data) Workflow:



For each step above, we'll use some python libraries to perform these tasks.

Python Libraries for Data Mining:

I. *Pandas (Python Data Analysis Library):*

- ✓ Explore, analyze, manipulate and prepare data ready for data mining task.
- ✓ Simple to use, integrated with other required libraries.
- ✓ Helps us to turn our data from different formats into other formats that could embedded in data mining algorithm.

II. *NumPy (Numerical Python):*

- ✓ It just like python arrays and lists, and one of most used libraries when it comes to data mining and machine learning problems because all its functions written in C (ensure fast). snice we do a lot of computation, so we need faster solutions than standard python arrays and lists.
- ✓ The other reasons to use NumPy is that it more understandable by machines to work with, think about representing image in array of numbers with each number consider as pixel color. Or convert some one has a disease or not to 1s and 0s so could machines make a better use of data.

III. *Matplotlib:*

- ✓ It is a visualization library used to make charts and diagrams that describe data, these charts and diagrams help us understanding data better.
- ✓ This library gives us all facilities to create meaningful charts that describe data well.

IV. *SciKit-Learn (Science Python Toolkit):*

- ✓ A standard machine learning library.
- ✓ Massive library that has a lot of functionalities and abilities for making models that help us extract interesting patterns and evaluate them.

V. *Tensor Flow:*

- ✓ A deep learning or numerical computing library.

- ✓ Conceived by Google and they used it within their own like.
- ✓ Now it's open-source software.
- ✓ Used to build deep learning and neural networks models to gain insights out of unstructured data.
- ✓ Its code is very fast since we can run it on GPUs.

We have many others such as Keras, PyTorch, ... We'll use Sci-Kit learn as a library for building models.

Example:

Let's use heart disease dataset, which consists of the following attributes:

#	Attribute	Explanation
1	Age	Age of the patient
2	Sex	1 for male and 0 for female
3	cp	Chest Pain type (1 to 4)
4	trestbps	resting blood pressure in (mm Hg)
5	chol	serum cholesterol in mg/dl
6	fbs	(fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7	thalach	: maximum heart rate achieved
8	Target (class label attribute)	Asymmetric binary attribute. 1 disease, 0 no disease

You can see more about this data using this link (https://archive.ics.uci.edu/ml/datasets/heart+disease).

What we're going to do is make use of libraries to load, explore and understand data.

- ***Import Required Libraries:***

We're going to import Pandas, NumPy and Matplotlib for this example using the following code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

- ***Reading Dataset and store it in dataframe using Pandas:***

```
#reading file using Pandas:
hd_csv = pd.read_csv("lab-1- hdcsv.csv")
```

A Dataframe is:

- ✓ the primary Pandas data structure.
- ✓ Two-dimensional, size-mutable, potentially heterogeneous tabular data.
- ✓ contains labeled axes (rows and columns).
- ✓ Arithmetic operations align on both row and column labels.
- ✓ Can be thought of as a dict-like container for Series objects.

- ***Viewing Dataset:***

```
#view first 5 lines of our dataset:
hd_csv.head() #you can pass the number of lines you want to read.
hd_csv.head(10) #you can view the first 10-lines by passing it to head function().
```

and the result may look like the following:

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

We can also parse some commonly known files such as JSON, XML, SQL, ... As Pandas data frame. Like the following example:

```
#reading our dataset (JSON Format) using Pandas:
hd_json = pd.read_json("hdd.json")
hd_json.head()
```

and we get the same result as above.

You can also use a method called `tail()` that reads the last five lines from our dataframe. You can also pass the number of lines we want to view.

Finally, we can export any dataframe to any common format (csv, excel, JSON, ...), for example to export `hd_csv` to a csv file we use the following method:

```
#exporting a dataframe to a csv file:
#using (index = False) prevents the function from export the index column into the
file.
hd_csv.to_csv('exported-hd-csv.csv', index=False)
```

- ***viewing datatypes of our dataframe:***

we can use an attribute called `dtypes` to view each attribute type:

```
#viewing a dataframe's attributes types:
hd_csv.dtypes
```

```
Out[7]: age          int64
        sex          int64
        cp           int64
        trestbps     int64
        chol         int64
        fbs          int64
        restecg      int64
        thalach      int64
        exang        int64
        oldpeak    float64
        slope        int64
        ca           int64
        thal         int64
        target       int64
        dtype: object
```

You can retrieve the name of any dataframe's attribute names using columns attribute.

```
hd_csv.columns
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

- ***Getting records from our dataframe:***

We have many ways to do this step, we're going to use two indexing techniques (loc and iloc):

```
# using loc with passing the index of record we want to retrieve:

# if we have to records with the same index loc will return the both records.

hd_csv.loc[9]
```

```
Out[9]: age         57.0
        sex          1.0
        cp           2.0
        trestbps   150.0
        chol       168.0
        fbs          0.0
        restecg      1.0
        thalach    174.0
        exang        0.0
        oldpeak      1.6
        slope        2.0
        ca           0.0
        thal         2.0
        target       1.0
        Name: 9, dtype: float64
```

```
#using iloc with passing the position of the record we want to record we want to
retrieve, for our example the result will be the same.

hd_csv.iloc[9]
```

finally, we can use the both ways to get multiple records at once by using slicing like the following example which returns the first 2 lines:

```
hd_csv.loc[:2] #the same for hd_.iloc[:2]
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1

- ***Get some statistics on our dataset:***

now we're going to use Pandas functions to make a bar plot that shows us summary about target attribute. (How many patients have heart disease and how many does not have heart disease). Then customize the plot using matplotlib library.

```
# drawing and customizing Our Graph:
hd_csv.target.value_counts().plot(kind='bar',
                                  figsize=(10,6),
                                  color=["salmon"])

plt.title("Heart Disease Frequency")
plt.xlabel("0= No Disease, 1 = Disease")
plt.ylabel("Amount")
plt.legend(["Target"])
plt.xticks(rotation=0)
```

- Target attribute is used to access data label attribute, usually called target in any dataset.
- value_counts() method is used to count how many times each label appear in our dataset.
- plot() method is used to make a chart, kind attribute specifies chart type which is bar chart in our example. It takes also figure size as a tuple and the color for our bar.
- We can customize our plot by setting its title and axis' names using xlabel and ylabel.