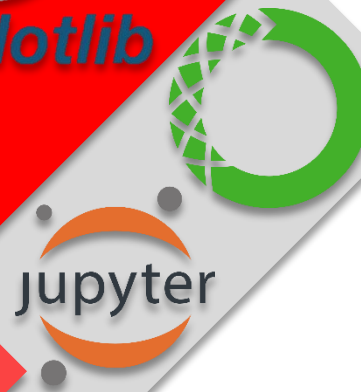
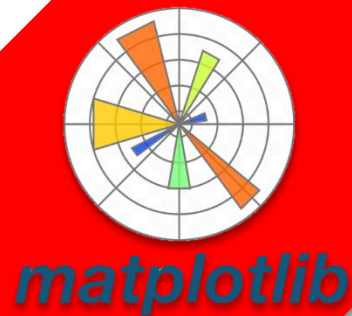


# Knowledge Databases Labs

5<sup>th</sup> Year



## Lab -1- Introduction to Required Libraries

*(Pandas, NumPy and Matplotlib)*

**Introduced By:**

*Eng. Khaled Kondakji*

*Eng. Heba Tadmouri*

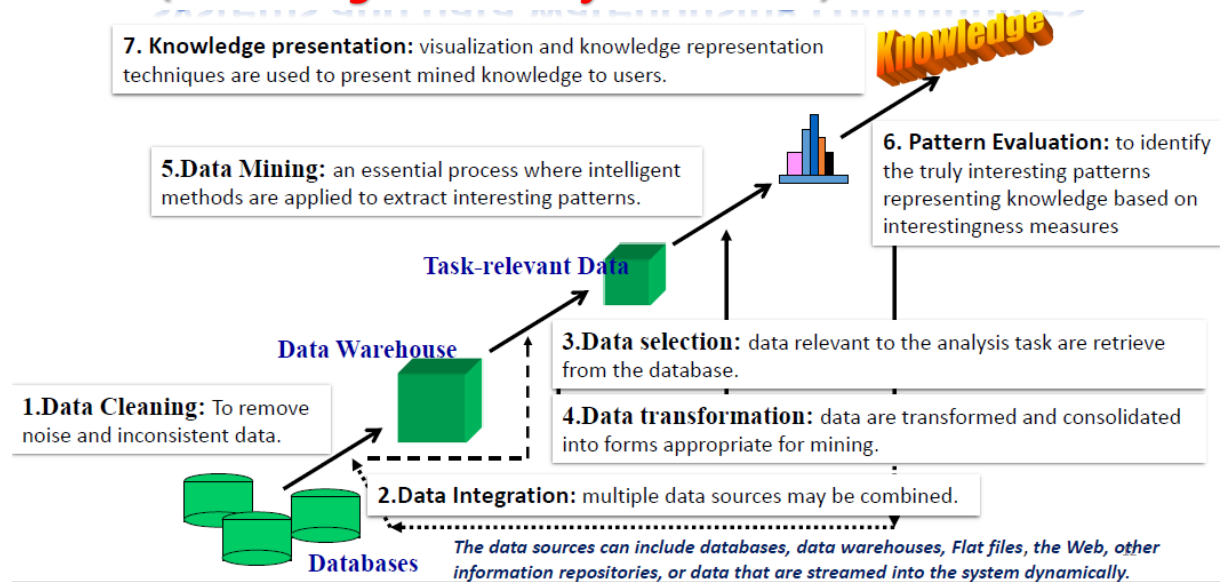
*Eng. Manar AL-Marai*

**Supervised By:**

*Dr. Eng. Asmaa Shaar*

# 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 like sequences and images.
-  Its code is very fast since we can run it on GPUs.

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

## Working with Data using Pandas:

In the following example we'll use music albums dataset, which consists of the following attributes:

#	Attribute	Explanation
1	Number	The ID of the album.
2	Year	The released year of the album.
3	Album	Album's title.
4	Artist	The singer of the album.
5	Genre	The categories that album belongs to.
6	Subgenre	The sub categories that album belongs to.

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

### I. **Import Required Libraries:**






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

### II. **Load, view and store our dataset in Pandas DataFrame:**



```
albums = pd.read_csv('albumlist.csv', encoding='windows_1258')
albums.head() #view the first 5 rows
```

	Number	Year	Album	Artist	Genre	Subgenre
0	1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	Rock	Rock & Roll, Psychedelic Rock
1	2	1966	Pet Sounds	The Beach Boys	Rock	Pop Rock, Psychedelic Rock
2	3	1966	Revolver	The Beatles	Rock	Psychedelic Rock, Pop Rock
3	4	1965	Highway 61 Revisited	Bob Dylan	Rock	Folk Rock, Blues Rock
4	5	1965	Rubber Soul	The Beatles	Rock, Pop	Pop Rock

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.

## Notes:

1. read\_csv() function takes some parameter like:
  -  file encoding type which is by default utf-8.
  -  sep parameter which specifies the separation between values which is by default is comma (,).

2. `head()` function can take parameter that specifies the number of rows to view, for example if we call `(pd.head(10))` it will show us the first 10 rows of our DataFrame.
3. You can also use a method called `tail()` that reads the last five rows from our DataFrame. You can also pass the number of rows we want to view.
4. We can also parse some commonly known files such as JSON, XML, SQL, .... As Pandas data frame.

Like the following example:

```
df_json = pd.read_json("myfile.json")
df_json.head()
```

5. Finally, we can export any DataFrame to any common format (csv, excel, JSON, ...), for example to export albums to a csv or JSON file we use the following methods:

```
#exporting a DataFrame to a json file:
albums.to_json('albums.json')
#exporting a DataFrame to csv file:
#using (index = False) prevents the function from export the index
column into the file.
albums.to_csv('albums.csv', index=False)
```

### III. Get General information about dataset attributes:

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

```
albums.dtypes
```

```
Number      int64
Year        int64
Album       object
Artist      object
Genre       object
Subgenre    object
dtype: object
```



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

```
albums.columns
```

```
Index(['Number', 'Year', 'Album', 'Artist', 'Genre', 'Subgenre'], dtype='object')
```

### IV. Extracting rows (records) from DataFrame:

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

-  `iloc[]` is an **indexed-based** selecting method which means that we have to pass integer index in the method to select a specific row/column. This method does not include the last element of the range passed in it. It's syntax like: `(iloc[row_index, column_index])`.
-  `loc[]` is a **label-based** data selecting method which means that we have to pass the name of the row or column that we want to select. This method includes the last element of the range passed in it. It's syntax like: `(loc[row_label, column_label])`.

Let's use the previous functions to retrieve data from `albums` DataFrame:

```
#Accessing first 3 rows:
```

```
albums.iloc[0:3]
```

	Number	Year	Album	Artist	Genre	Subgenre
0	1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	Rock	Rock & Roll, Psychedelic Rock
1	2	1966	Pet Sounds	The Beach Boys	Rock	Pop Rock, Psychedelic Rock
2	3	1966	Revolver	The Beatles	Rock	Psychedelic Rock, Pop Rock

#Accessing first 3 rows and first 2 attributes:

```
albums.iloc[0:3,0:2]
```

	Number	Year
0	1	1967
1	2	1966
2	3	1966

By default, when we load our file in a DataFrame it comes with index added to it. In order to understand the usage of `loc[]` we'll change the index to an attribute from our dataset like Album attribute.

```
albums = albums.set_index('Album')
```

```
albums.head()
```

	Number	Year	Artist	Genre	Subgenre
Album					
Sgt. Pepper's Lonely Hearts Club Band	1	1967	The Beatles	Rock	Rock & Roll, Psychedelic Rock
Pet Sounds	2	1966	The Beach Boys	Rock	Pop Rock, Psychedelic Rock
Revolver	3	1966	The Beatles	Rock	Psychedelic Rock, Pop Rock
Highway 61 Revisited	4	1965	Bob Dylan	Rock	Folk Rock, Blues Rock
Rubber Soul	5	1965	The Beatles	Rock, Pop	Pop Rock

Now, let's use `loc()` function to extract data:

#Accessing Revolver album information:

```
albums.loc['Revolver']
```

```

Number          3
Year            1966
Artist          The Beatles
Genre           Rock
Subgenre        Psychedelic Rock, Pop Rock
Name: Revolver, dtype: object
```

#Accessing Revolver album information (from Number to Artist):

```
albums.loc['Revolver', 'Number':'Artist']
```

```

Number          3
Year            1966
Artist          The Beatles
Name: Revolver, dtype: object
```

#Accessing Number, Year and Artist for first 5 albums:

```
albums.loc["Sgt. Pepper's Lonely Hearts Club Band":"Rubber Soul", 'Number':'Artist']
```

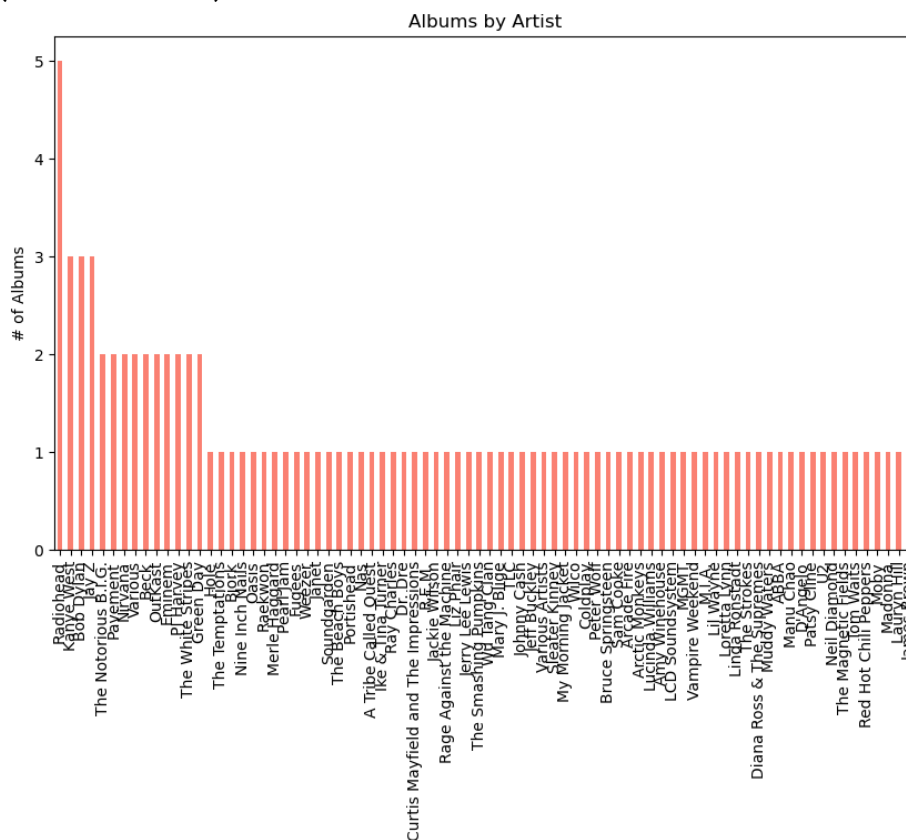
	Number	Year	Artist
Album			
Sgt. Pepper's Lonely Hearts Club Band	1	1967	The Beatles
Pet Sounds	2	1966	The Beach Boys
Revolver	3	1966	The Beatles
Highway 61 Revisited	4	1965	Bob Dylan
Rubber Soul	5	1965	The Beatles

## V. Visualize Some Charts that Describes Data:

Now, we're going to use matplotlib library to visualize some charts that shows some information about our data. In our first scenario we're going to sort our DataFrame based on year attribute in descending order and visualize a bar plot the shows how many albums released by artists for last few years.

```
albums = albums.sort_values(by='Year',ascending=False)
#albums.sort_values(by='Year',ascending=False, inplace=True)
simp_albums = albums[:100]
simp_albums.Artist.value_counts().plot(kind='bar',
                                       figsize=(10,6),
                                       color=["salmon"])
```

```
plt.title("Albums by Artist")
plt.ylabel("# of Albums")
```



- ✎ `sort_values()` is used to sort a DataFrame based on attribute specified by (by) attribute, by default `sort_values()` sorts data in ascending order, when set ascending attribute to false, then the order is reversed.
- ✎ `sort_values()` function takes also `inplace` attribute which replaces the original DataFrame with sorted version without reassign it.
- ✎ `value_counts()` method is used to count how many times each artist appear in our simple 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`.

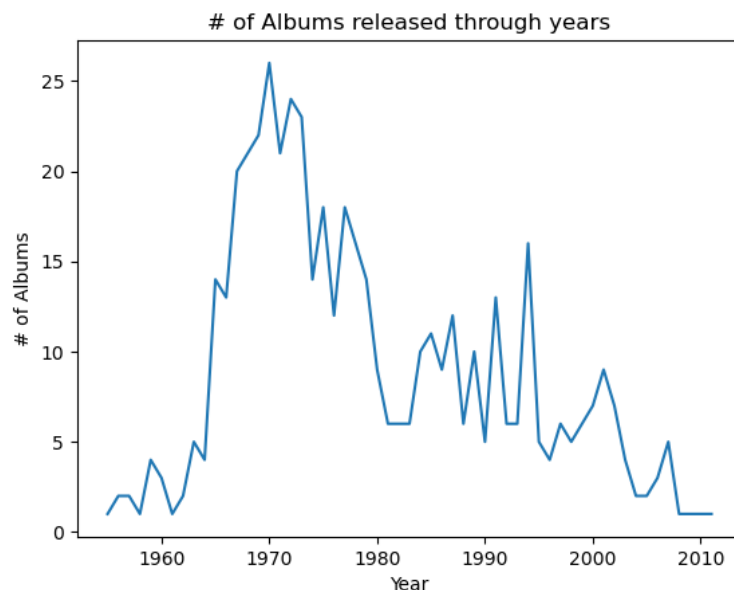
In our next scenario we're going to visualize a line plot that shows how many albums released in every year:

```
#Restore default indexing using reset_index() function
albums = albums.reset_index()
group_by_year = albums.groupby('Year')['Album'].count()
group_by_year.head()
```

```
Year
1955    1
1956    2
1957    2
1958    1
1959    4
Name: Album, dtype: int64
```

Now, let's use the previous code in order to complete our example:

```
plt.plot(sorted(albums['Year'].unique()),albums.groupby('Year')['Album'].count())
plt.xlabel('Year')
plt.ylabel('# of Albums')
plt.title('# of Albums released through years')
```



- ✎ In this scenario we use `unique()` function which returns unique values in year attribute, then used built-in `sorted()` function to sort the values in ascending order.
- ✎ We use `groupby()` function to perform aggregation on album attribute by year, we also use `count()` function to store how many albums released in each year.
- ✎ By default, matplotlib visualize a line then we specified the `xlabel`, `ylabel` and title of our figure.

## References:

- 1- Pandas docs <https://pandas.pydata.org/docs/> .
- 2- NumPy docs <https://numpy.org/doc/> .
- 3- Matplotlib docs <https://matplotlib.org/3.3.3/contents.html> .
- 4- SciKit-learn docs <https://scikit-learn.org/0.21/documentation.html> .
- 5- <https://realpython.com/pandas-groupby/>
- 6- <https://realpython.com/pandas-sort-python/>