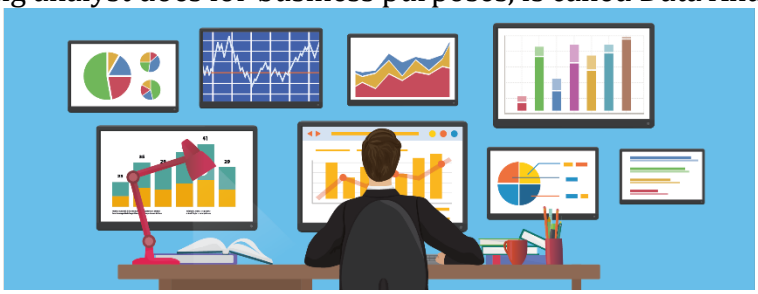
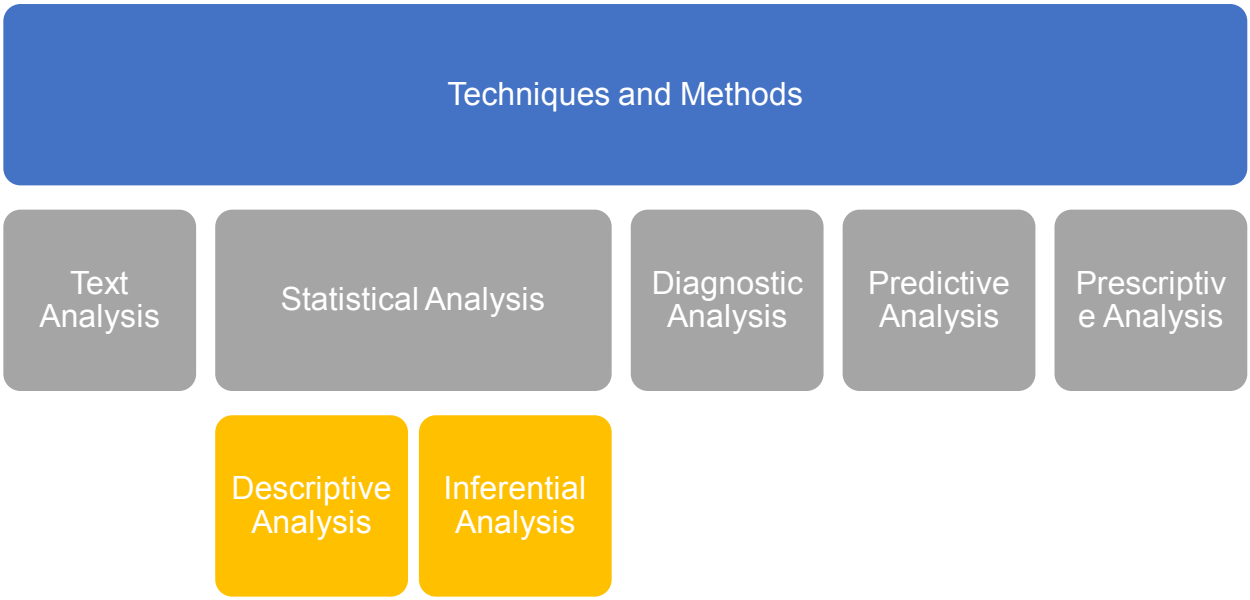
PYTHON FOR DATA ANALYSIS

Data analysis is defined as a process of cleaning, transforming, and modeling data to discover useful information for business decision-making. The purpose of Data Analysis is to extract useful information from data and taking the decision based upon the data analysis. Whenever we take any decision in our day-to-day life is by thinking about what happened last time or what will happen by choosing that particular decision. This is nothing but analyzing our past or future and making decisions based on it. For that, we gather memories of our past or dreams of our future. So that is nothing but data analysis. Now same thing analyst does for business purposes, is called Data Analysis.



Types of Data Analysis: Techniques and Methods



There are several types of Data Analysis techniques that exist based on business and technology. However, the major types of data analysis are:

- Text Analysis;
- Statistical Analysis;
- Diagnostic Analysis;
- Predictive Analysis;
- Prescriptive Analysis.

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Data Analysis Process

Data Analysis consists of the following phases:

- Data Requirement Gathering;
- Data Collection;
- Data Cleaning;
- Data Analysis;
- Data Interpretation;
- Data Visualization.

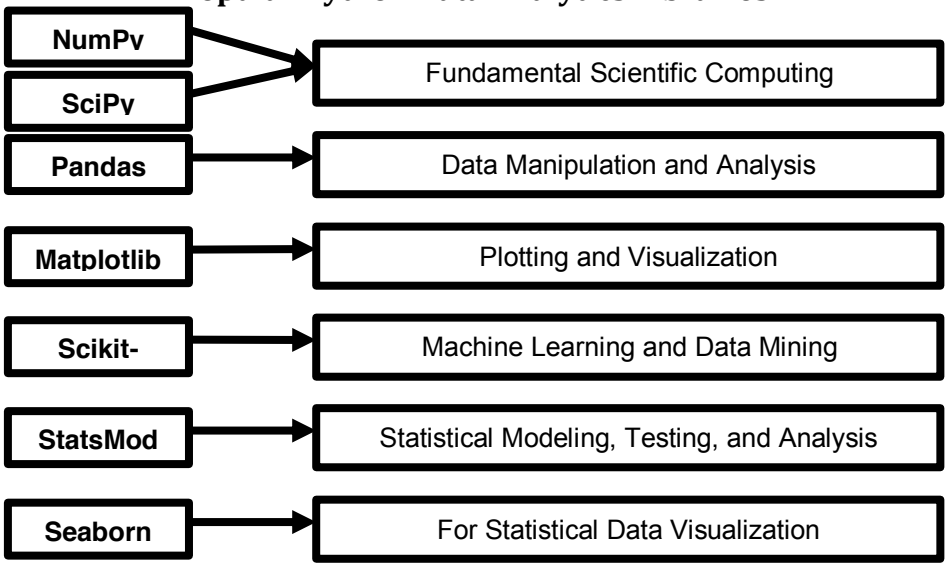
Python as a Data Analytics Tool

Data analysis tools make it easier for users to process and manipulate data, analyze the relationships and correlations between data sets, and it also helps to identify patterns and trends for interpretation.

The nature of Python makes it a perfect-fit for data analytics:

- Easy to learn;
- Readable;
- Scalable;
- Extensive set of libraries;
- Easy integration with other apps;
- Active community & ecosystem.

Popular Python Data Analytics Libraries



Python has extensive library support for data science and analytics. There are many Python libraries that contain a host of functions, tools, and methods to manage and analyze data. Each of these libraries has a particular focus with some libraries managing image and textual data, data mining, neural networks, data visualization, and so on.

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

NumPy stands for Numerical Python. The most powerful feature of NumPy is n-dimensional array. This library also contains basic linear algebra functions, Fourier transforms, advanced random number capabilities and tools for integration with other low level languages like Fortran, C and C++.

SciPy stands for Scientific Python. It is built on NumPy. Scipy is one of the most useful library for variety of high level science and engineering modules like discrete Fourier transform, Linear Algebra, Optimization and Sparse matrices.

Pandas for structured data operations and manipulations. It is extensively used for data munging and preparation. Pandas were added relatively recently to Python and have been instrumental in boosting Python's usage in data scientist community.

Matplotlib for plotting vast variety of graphs, starting from histograms to line plots to heat plots.. You can use Pylab feature in ipython notebook (ipython notebook –pylab = inline) to use these plotting features inline. If you ignore the inline option, then pylab converts ipython environment to an environment, very similar to Matlab.

Scikit-Learn for machine learning. Built on NumPy, SciPy and matplotlib, this library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensional reduction.

Statsmodels for statistical modeling. It is a Python module that allows users to explore data, estimate statistical models, and perform statistical tests. An extensive list of descriptive statistics, statistical tests, plotting functions, and result statistics are available for different types of data and each estimator.

Seaborn for statistical data visualization. It is a library for making attractive and informative statistical graphics in Python. It is based on matplotlib. Seaborn aims to make visualization a central part of exploring and understanding data.

Loading Python Libraries

```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```

A library is a collection of files that contains functions for use by other programs.

May also contain data values and other things. Library's contents are supposed to be related, but there's no way to enforce that.

The Python standard library is an extensive suite of modules that comes with Python itself.

Many additional libraries are available from the Python Package Index.

Reading data using pandas

```
#Read csv file
df = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/Salaries.csv")
```

There is a number of pandas commands to read other data formats:

```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1', index_col=None, na_values=['NA'])
pd.read_stata('myfile.dta')
pd.read_sas('myfile.sas7bdat')
pd.read_hdf('myfile.h5','df')
```

Pandas is the most popular data manipulation package in Python.

CSV (comma-separated value) files are a common file format for transferring and storing data. The ability to read, manipulate, and write data to and from CSV files using Python is a key skill to master for any data scientist or business analysis.

You might have your data in .csv files or SQL tables. Maybe Excel files. Or .tsv files. Or something else.

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

But the goal is the same in all cases. If you want to analyze that data using pandas, the first step will be to read it into a data structure that's compatible with pandas.

Exploring data frames

```
#List first 5 records
df.head()
```

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

The **head()** function is used to get the first **n** rows.

This function returns the first **n** rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

In this example, you can see the first 5 records.

DataFrame data types

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

A data type is essentially an internal construct that a programming language uses to understand how to store and manipulate data. For instance, a program needs to understand that you can add two numbers together like 5 + 10 to get 15.

A possible confusing point about pandas data types is that there is some overlap between pandas, python and numpy. This table summarizes the key points.

Data Frame data types

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

In [4]:
#Check a particular column type
df['salary'].dtype

Out[4]:
dtype('int64')

In [5]:
#Check types for all the columns
df.dtypes
Out[4]:
rank object
discipline object
phd int64
service int64
sex object
salary int64
dtype: object

Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. It can be thought of as a dict-like container for Series objects. This is the primary data structure of the Pandas.

Pandas ***DataFrame.dtypes*** attribute return the ***dtypes*** in the DataFrame. It returns a Series with the data type of each column.

Data Frames attributes

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

DATA FRAMES METHODS

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Python objects have attributes and methods.
Unlike attributes, python methods have parenthesis.
All attributes and methods can be listed with a **dir()** function: **dir(df)**

Data Frames *groupby* method

```
In [ ]: #Group data using rank
        df_rank = df.groupby(['rank'])
```

```
In [ ]: #Calculate mean value for each numeric column per each group
        df_rank.mean()
```

rank	phd	service	salary
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

Using **"group by"** method we can:
- ✓ Split the data into groups based on some criteria;
- ✓ Calculate statistics (or apply a function) to each group;
- ✓ Similar to **dplyr()** function in **R**.

Data Frames *groupby* method

```
In [ ]: #Calculate mean salary for each professor rank:
        df.groupby('rank')[['salary']].mean()
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

	salary
rank	
AssocProf	91786.230769
AsstProf	81362.789474
Prof	123624.804348

Once **groupby** object is create we can calculate various statistics for each group.

Data Frames *groupby* method

```
In [ ]:  #Calculate mean salary for each professor rank:
         df.groupby(['rank'], sort=False)[['salary']].mean()
```

groupby performance notes:

- no grouping/splitting occurs until it's needed. Creating the **groupby** object only verifies that you have passed a valid mapping;

- by default the group keys are sorted during the **groupby** operation. You may want to pass sort=False for potential speedup:

Data Frame: filtering

```
In [ ]:  #Calculate mean salary for each professor rank:
         df_sub = df[ df['salary'] > 120000 ]
```

> greater;
< less;
== equal;
>= greater or equal;
<= less or equal;
!= not equal;

```
In [ ]:  #Select only those rows that contain female
         professors:
         df_f = df[ df['sex'] == 'Female' ]
```

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example, if we want to subset the rows in which the salary value is greater than $120K:

Any Boolean operator can be used to subset the data:

> greater;
< less;
== equal;
>= greater or equal;

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

<= less or equal;
!= not equal;

Data Frames: Slicing

In [ ]:
```
#Select column salary:
df['salary']
```

In [ ]:
```
#Select column salary:
df[['rank','salary']]
```

There are a number of ways to subset the Data Frame:

- ✓ one or more columns;
- ✓ one or more rows;
- ✓ a subset of rows and columns.

Rows and columns can be selected by their position or label.
When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):
When we need to select more than one column and/or make the output to be a DataFrame, we should use double brackets:

Data Frames: Selecting rows

In [ ]:
```
#Select rows by their position:
df[10:20]
```

If we need to select a range of rows, we can specify the range using **colon** ":"
Notice that the first row has a position 0, and the last value in the range is omitted:
So for 0:10 range the first 10 rows are returned with the positions starting with 0 and ending with 9.

Data Frames: method loc

In [ ]:
```
#Select rows by their labels:
df_sub.loc[10:20,['rank','sex','salary']]
```

Out[ ]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

If we need to select a range of rows, using their labels we can use method **loc**.

Data Frames: method iloc

In [ ]:
```
#Select rows by their labels:
df_sub.iloc[10:20,[0, 3, 4, 5]]
```

Out[ ]:

	rank	service	sex	salary
26	Prof	19	Male	148750
27	Prof	43	Male	155865
29	Prof	20	Male	123683
31	Prof	21	Male	155750
35	Prof	23	Male	126933
36	Prof	45	Male	146856
39	Prof	18	Female	129000
40	Prof	36	Female	137000
44	Prof	19	Female	151768
45	Prof	25	Female	140096

If we need to select a range of rows and/or columns, using their positions we can use method **iloc**.

Data Frames: method iloc (summary)

```
df.iloc[0]  # First row of a data frame
df.iloc[i]  #(i+1)th row
df.iloc[-1] # Last row
```

```
df.iloc[:, 0]  # First column
df.iloc[:, -1] # Last column
```

```
df.iloc[0:7]        #First 7 rows
df.iloc[:, 0:2]     #First 2 columns
df.iloc[1:3, 0:2]   #Second through third rows and first 2 columns
df.iloc[[0,5], [1,3]]  #1st and 6th rows and 2nd and 4th columns
```

The **iloc** indexer for Pandas Dataframe is used for integer-location based indexing / selection by position.

The **iloc** indexer syntax is **data.iloc[<row selection>, <column selection>]**, which is sure to be a source of confusion for R users.

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

"iloc" in pandas is used to select rows and columns by number, in the order that they appear in the data frame.

Data Frames: Sorting

In [ ]:
```
# Create a new data frame from the original sorted by the column Salary

df_sorted = df.sort_values( by ='service')

df_sorted.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

We can sort the data by a value in the column. By default the sorting will occur in ascending order and a new data frame is return.

Data Frames: Sorting

In [ ]:
```
df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])

df_sorted.head(10)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

We can sort the data using 2 or more columns.

Missing Values

```
In [ ]: # Read a dataset with missing values
        flights =
        pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/flights.csv")
```

```
In [ ]: # Select the rows that have at least one missing value
        flights[flights.isnull().any(axis=1)].head()
```

Out[ ]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest	air_time	distance	hour	minute
330	2013	1	1	1807.0	29.0	2251.0	NaN	UA	N31412	1228	EWR	SAN	NaN	2425	18.0	7.0
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EHAA	791	LGA	DFW	NaN	1389	NaN	NaN
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	1925	LGA	MIA	NaN	1096	NaN	NaN
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	1299	EWR	RSW	NaN	1068	21.0	45.0
858	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	133	JFK	LAX	NaN	2475	NaN	NaN

Missing values are marked as NaN.

Missing Values

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

There are a number of methods to deal with missing values in the data frame.

- When summing the data, missing values will be treated as zero;
- If all values are missing, the sum will be equal to NaN;
- cumsum() and cumprod() methods ignore missing values but preserve them in the resulting arrays;

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- Missing values in GroupBy method are excluded (just like in R);
- Many descriptive statistics methods have *skipna* option to control if missing data should be excluded . This value is set to *True* by default (unlike R).

Importing the Data from CSV file

```
In [20]: sfo_cust_data_2015 = pd.read_csv('sfo cust sat 2015_data file_final_WEIGHTED_flysfo.csv')

In [21]: sfo_cust_data_2015
```

Out[21]:

	RESPNUM	CCGID	RUNID	INTDATE	AIRLINE	FLIGHT	DESTINATION	DESTGEO	DESTMARK	GATE	...	Q17ZIP	Q17COUNTRY	Q18AG
0	3	460.0	15076	13	8	242	18	3	4	58	...	94619.0	US	6
1	4	554.0	15083	12	38	1777	99	4	2	1	...	NaN	CANADA	6
2	5	555.0	15083	12	38	1777	99	4	2	1	...	NaN	CANADA	2
3	6	461.0	15076	13	8	242	18	3	4	58	...	61801.0	US	2
4	7	556.0	15083	12	38	1777	99	4	2	1	...	94119.0	US	3
5	8	557.0	15083	12	38	1777	99	4	2	1	...	0.0	0	0
6	10	558.0	15083	12	38	1777	99	4	2	1	...	94404.0	US	4
7	11	462.0	15076	13	8	242	18	3	4	58	...	95404.0	US	5
8	12	559.0	15083	12	38	1777	99	4	2	1	...	94301.0	US	1

Pandas is an opensource library that allows to you perform data manipulation in Python. Pandas provide an easy way to create, manipulate and delete the data.

Reading the CSV into a pandas DataFrame is very quick and easy.

Using the **read_csv()** function from the pandas package, you can import tabular data from CSV files into pandas dataframe by specifying a parameter value for the file name.

Importing the Data from Excel file

```
In [1]: # Import pandas library
        import pandas as pd

In [2]: # Import data files
        sfo_cust_data_2014 = pd.read_excel('sfo cust sat 2014 data file_WEIGHTED_flysfo.xlsx', 'Sheet 1')

In [3]: sfo_cust_data_2014
```

Out[3]:

	RESPNUM	CCGID	RUN	INTDATE	GATE	AREA	STRATA	PEAK	METHOD	SAQ	...	Q17COUNTRY	HOME	Q18AGE	Q19GENDER	Q20
0	1	348.0	18045	4	54	D	3	2	1	1	...	US	1.0	3	2	2
1	2	349.0	18045	4	54	D	3	2	1	1	...	US	3.0	2	1	1
2	3	350.0	18045	4	54	D	3	2	1	1	...	US	1.0	5	1	4
3	4	351.0	18045	4	54	D	3	2	1	2	...	US	2.0	3	2	4
4	5	352.0	18045	4	54	D	3	2	1	2	...	US	1.0	3	1	1
5	6	353.0	18045	4	54	D	3	2	1	2	...	US	3.0	3	1	1
6	7	354.0	18045	4	54	D	3	2	1	2	...	US	4.0	3	1	2
7	8	355.0	18045	4	54	D	3	2	1	2	...	US	1.0	3	2	2
8	9	356.0	18045	4	54	D	3	2	1	2	...	US	12.0	4	1	3

You can easily import an Excel file into Python using Pandas. In order to accomplish this goal, you'll need to use ***read_excel***.

The method **read_excel()** reads the data into a Pandas Data Frame, where the first parameter is the **filename** and the second parameter is the **sheet**.

Importing the Data from JSON file

```
import json
with open('path_to_file/person.json') as f:
  data = json.load(f)
# Output: {'name': 'Bob', 'languages': ['English', 'Fench']}
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
print(data)
```

The full-form of JSON is JavaScript Object Notation. It means that a script (executable) file which is made of text in a programming language, is used to store and transfer the data. Python supports JSON through a built-in package called json. To use this feature, we import the json package in Python script. The text in JSON is done through quoted string which contains the value in key-value mapping within brackets.

You can use **json.load()** method to read a file containing **JSON** object.

Here, we have used the **open()** function to read the **json** file. Then, the file is parsed using **json.load()** method which gives us a dictionary named data.

ReFerences

1. Link: [https://www.guru99.com/what-is-data-analysis.html#:~:text=Data%20analysis%20is%20defined%20as,based%20upon%20the%20data%20analysis.]
2. Katia Oleinik “Python for Data Analysis” Taken from Slides at Boston University
3. Samuel G. Mori, CISA, Managing Partner, Analytics & Advisory Services, Spyrion LLC, April 12, 2018
4. Link: [http://makemeanalyst.com/data-science-with-python/python-libraries-for-data-analysis/]
5. https://www.programiz.com/python-programming/json

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.