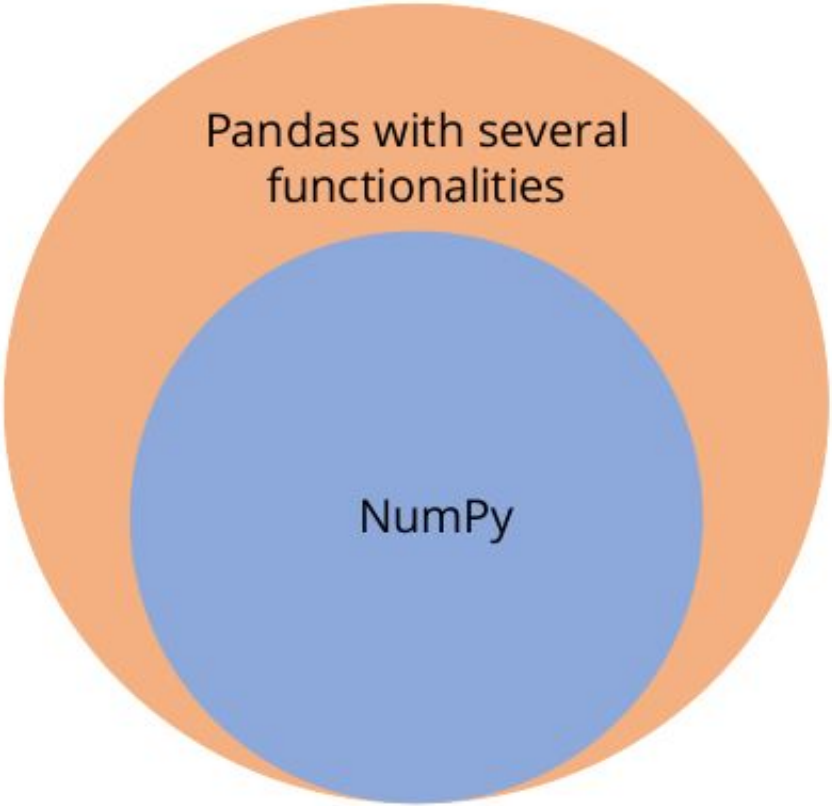
Data Manipulation with Pandas

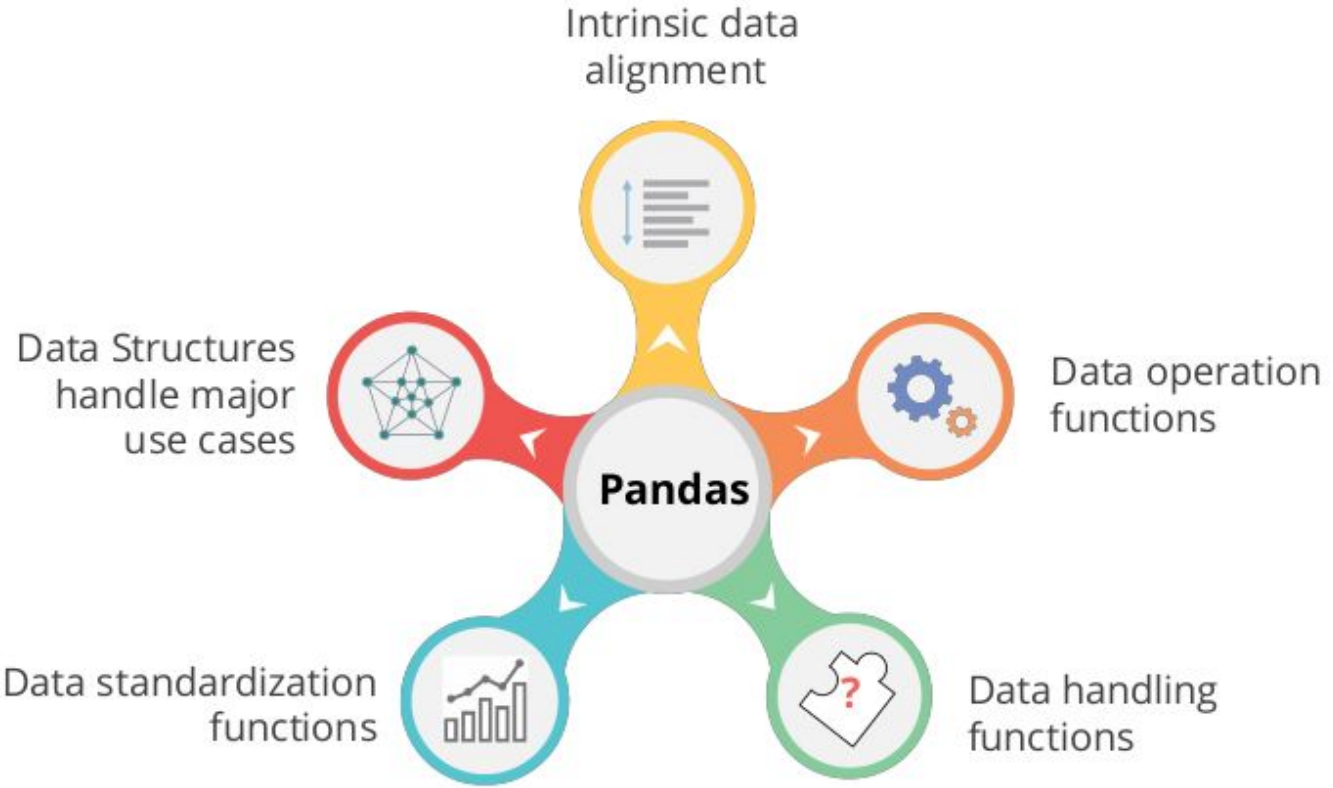
Learning Objectives

- Understand pandas and its features
- List different data structures of Pandas
- Outline the process to create series and DataFrame with data inputs
- missing values
- Analyze data with different data operation methods

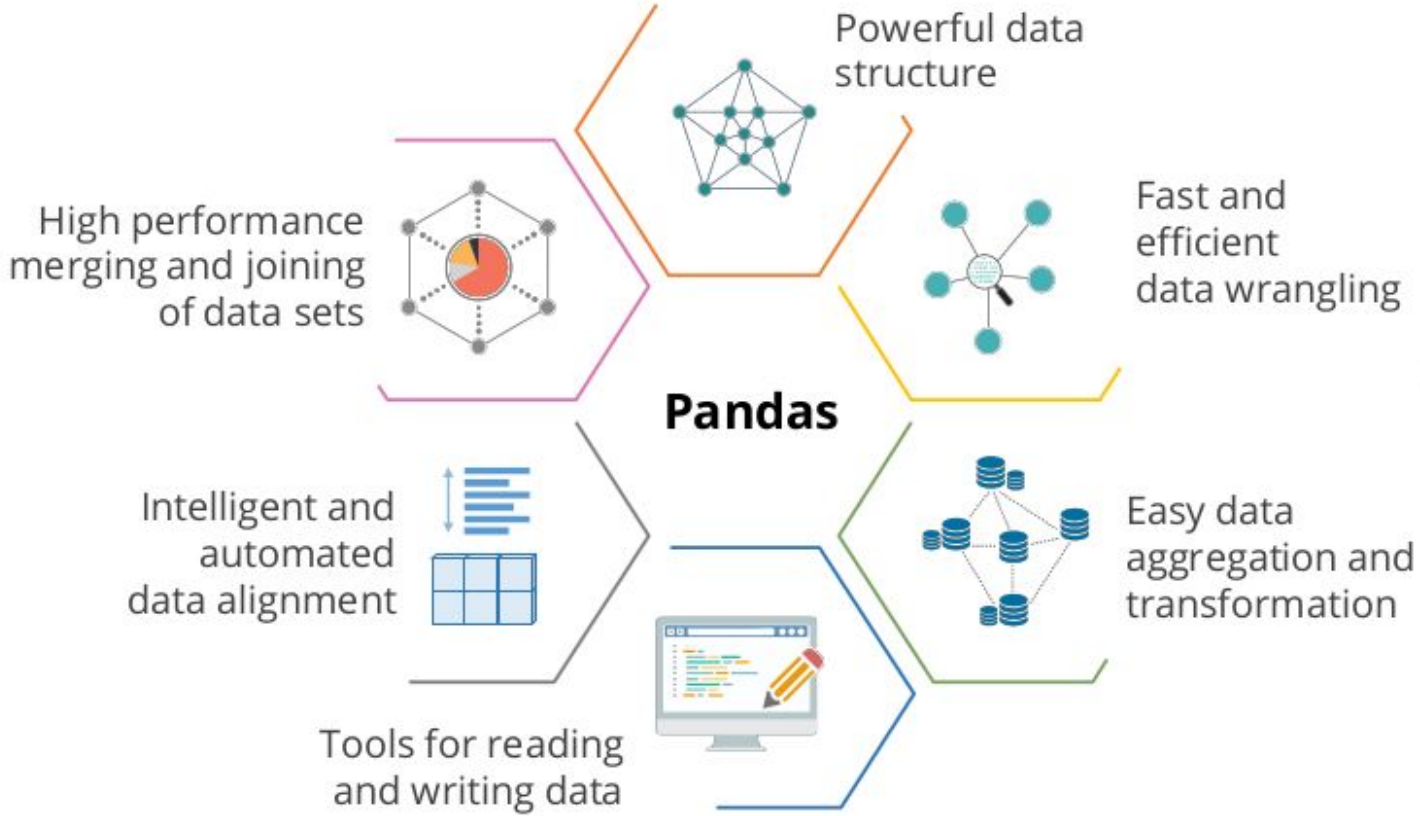
Why pandas??



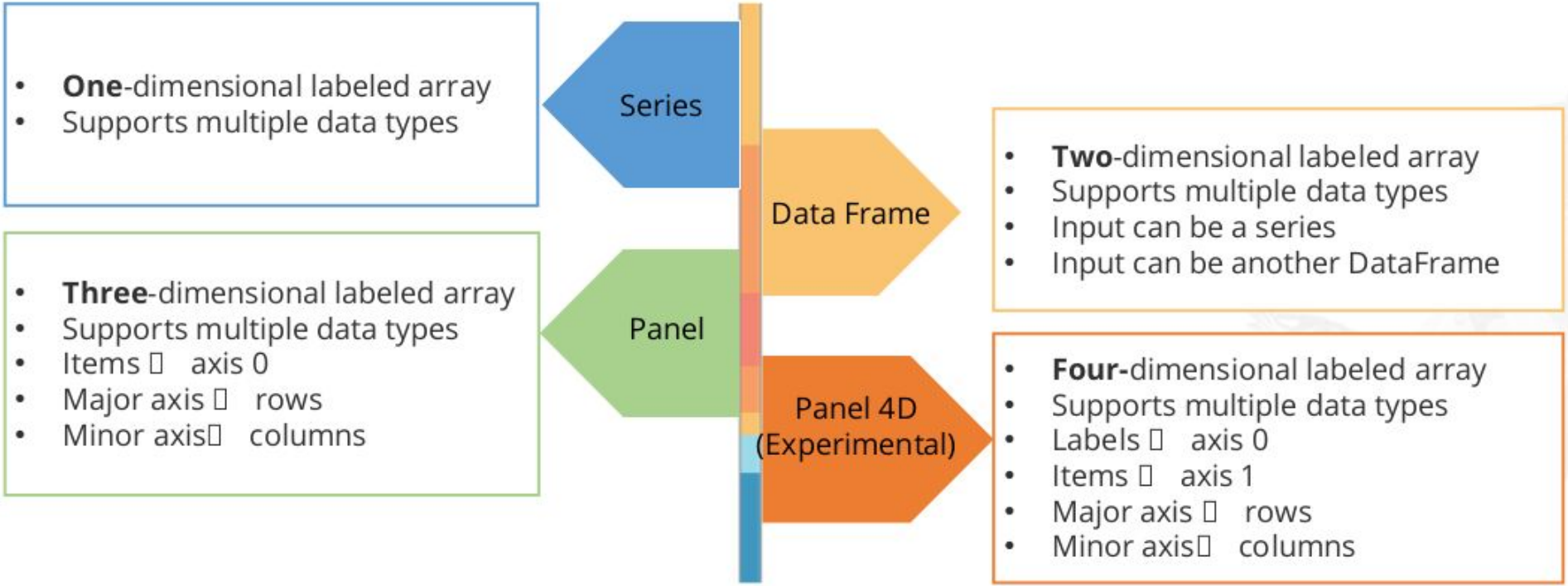
Why pandas??


Intrinsic data alignment
Data Structures handle major use cases
Data operation functions
Pandas
Data standardization functions
Data handling functions

Features

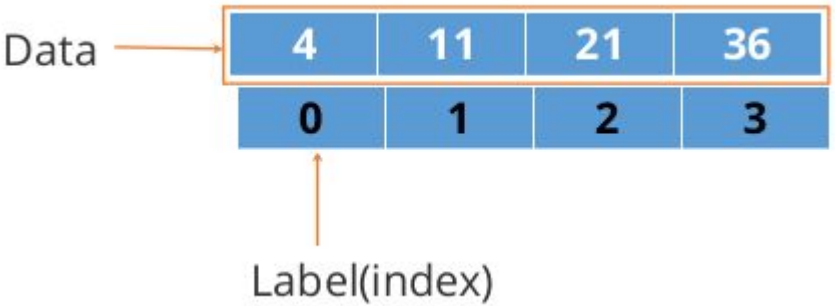

Powerful data structure
Fast and efficient data wrangling
Easy data aggregation and transformation
Tools for reading and writing data
Intelligent and automated data alignment
High performance merging and joining of data sets
Pandas

Data Structures



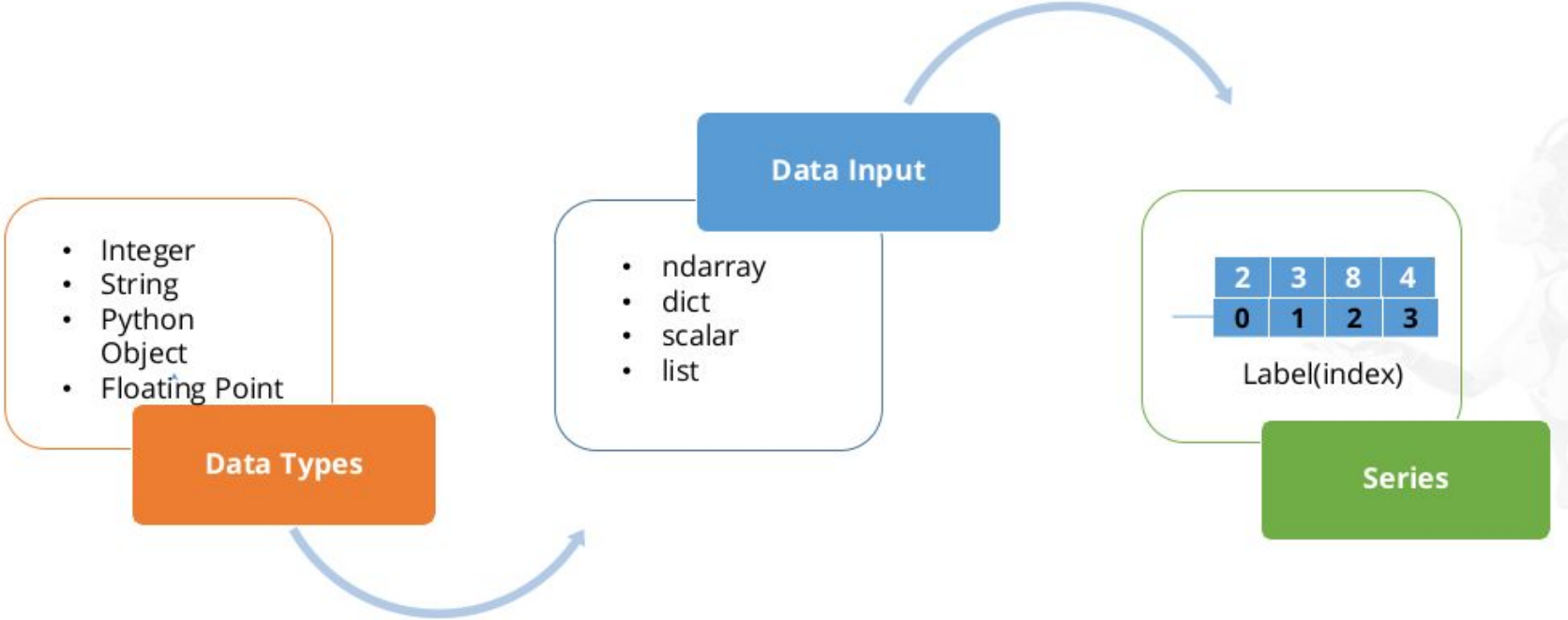
Series

- Series is a one-dimensional array-like object containing data and labels (or index).
- Capable of holding data of any type (integer, string, float, python objects, etc.)



Data alignment is intrinsic and will not be broken until changed explicitly by program.

- Series is a one-dimensional labeled array Which are capable of holding data of any type (integer, string, float, python objects, etc.).
- Series can be created with different data inputs:



How to create a Series??

- Import Pandas as it is the main library (Import pandas as pd)
- Import NumPy while working with ndarrays (Import numpy as np)
- Apply the syntax and pass the data elements as arguments

Basic Method

```
S = pd.Series(data, index = [index])
```

Creating Series from a List

```
#Import Pandas as it is the main library
import pandas as pd
#Import NumPy while working with ndarrays
import numpy as np
```

Import libraries

```
#install pandas and numpy if not installed
#!pip3 install pandas
```

```
first_series = pd.Series(list('ICFOSS'))
```

Pass list as an argument

```
print(first_series)
```

```
0    I
1    C
2    F
3    O
4    S
5    S
dtype: object
```

Index

Data Value

Data Type

We have not created index for data but notice that data alignment is done automatically.

Creating Series from an ndarray

```
arr = np.array([22,25,28,30,35])
```

Pass array as an argument

```
pd.Series(arr)
```

```
0    22
1    25
2    28
3    30
4    35
dtype: int64
```

Index

Data Value

Data Type

Creating Series from dict

```
dic = {'a':22,'b':25,'c':28,'d':30,'e':35}
```

```
pd.Series(dic)
```

Pass dictionary as an argument

```
a    22
b    25
c    28
d    30
e    35
dtype: int64
```

labels

Data Value

Data Type

Creating Series from Scalar

Scalar has only one dimension, thus represented by a single value.

```
#print series with scalar inputs
scalar_series = pd.Series(25, index=['a','b','c','d','e'])
```

```
print(scalar_series)
```

```
a    25
b    25
c    25
d    25
e    25
dtype: int64
```

labels ← (a, b, c, d, e) → Data Value

Data Type ← dtype: int64

Accessing Elements in Series

Data can be accessed through different functions like loc, iloc by passing data element position or index range.

```
series_dic[0]
```

```
22
```

```
series_dic[0:3]
```

```
a    22
b    25
c    28
dtype: int64
```

```
series_dic.loc['a']
```

```
22
```

```
series_dic.iloc[3]
```

```
30
```

Vectorizing Operations in Series

```
series1 = pd.Series([22,25,28,30,35], index = ['a','b','c','d','e'])
series2 = pd.Series([28,30,26,32,30],index= ['a','b','c','d','e'])
```

```
series1 + series2
```

```
a    50
b    55
c    54
d    62
e    65
dtype: int64
```

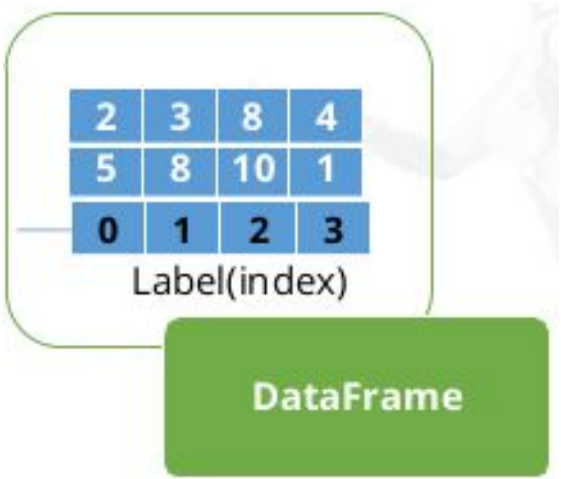
```
series3 = pd.Series([30,31,32,33,34,35,36], index = ['a','b','c','d','f','g','h'])
```

```
series1 +series1+series3
```

```
a    74.0
b    81.0
c    88.0
d    93.0
e     NaN
f     NaN
g     NaN
h     NaN
dtype: float64
```

DataFrame

DataFrame is a two-dimensional labeled data structure with columns of potentially different types.



Creating DataFrame from Lists

```
import pandas as pd
import numpy as np
```

```
student_details = {'Name':['Deepu', 'Jaison', 'Sudheesh'],
                   'District':['Kozhikode','Ernakulam', 'Idukki'],
                   'marks': ['35','36','37']}
```

```
df_student_details = pd.DataFrame(student_details)
```

```
df_student_details
```

	Name	District	marks
0	Deepu	Kozhikode	35
1	Jaison	Ernakulam	36
2	Sudheesh	Idukki	37

Creating DataFrame from dict

This example shows you how to create a DataFrame from a series of dicts.

```
student_details = {'Total_trainings':{2016:30, 2017:35, 2018:40},
                   ' Students_impacted':{2016:800,2017:1300,2018:2000}}
```

```
df_student_dict = pd.DataFrame(student_details)
```

```
df_student_dict
```

	Total_trainings	Students_impacted
2016	30	800
2017	35	1300
2018	40	2000

Viewing DataFrame

You can view a DataFrame by referring to the column name or with the describe function.

```
#Select by total trainings
df_student_dict.Total_trainings
```

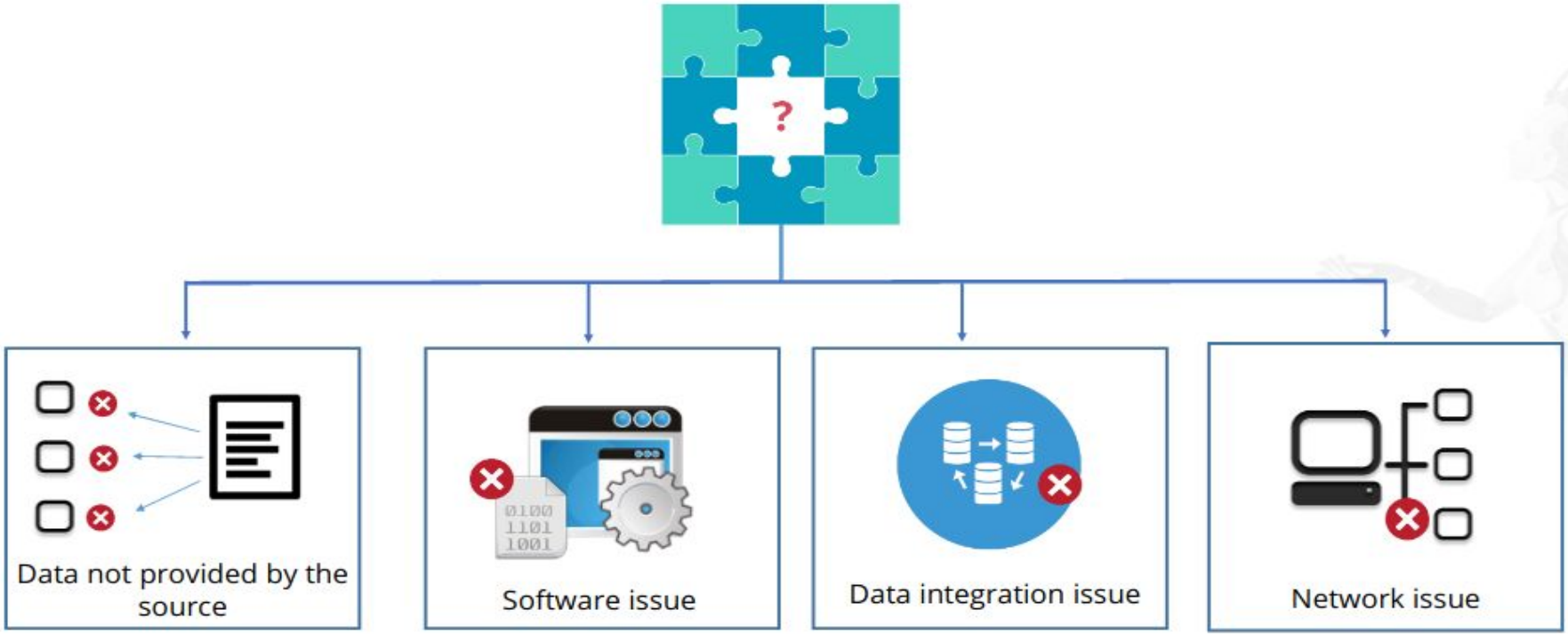
```
2016    30
2017    35
2018    40
Name: Total_trainings, dtype: int64
```

```
# Use describe function to view the content
df_student_dict.describe
```

```
<bound method NDFrame.describe of       Total_trainings   Students_impacted
2016               30                800
2017               35               1300
2018               40               2000>
```

Missing Values

Various factors may lead to missing data values:



Handling Missing Values

It’s difficult to operate a dataset when it has missing values or uncommon indices.

```
import pandas as pd
```

```
#declare first and second series
first_series =  pd.Series([1,2,3,4,5], index=['a','b','c','d','e'])
second_series = pd.Series([10,20,30,40,50], index = ['c','e','f','g','h'])
```

```
sum_of_series = first_series + second_series
```

```
sum_of_series
```

```
a     NaN
b     NaN
c    13.0
d     NaN
e    25.0
f     NaN
g     NaN
h     NaN
dtype: float64
```

Handling Missing Values with Functions

The dropna function drops all the values with uncommon indices.

```
sum_of_series
```

```
a     NaN
b     NaN
c    13.0
d     NaN
e    25.0
f     NaN
g     NaN
h     NaN
dtype: float64
```

```
# The dropna function drops all the values with uncommon indices.
# drop NaN(not a number) values from dataset
dropna_s = sum_of_series.dropna()
```

```
dropna_s
```

```
c    13.0
e    25.0
dtype: float64
```

The fillna function fills all the uncommon indices with a number instead of dropping them.

```
#fill the missing values with zero
fillna_s = sum_of_series.fillna(0)
```

```
fillna_s
```

```
a     0.0
b     0.0
c    13.0
d     0.0
e    25.0
f     0.0
g     0.0
h     0.0
dtype: float64
```

Data Operation

Data operation can be performed through various built-in methods for faster data processing.

```
import pandas as pd
```

```
#Declare the scores for mathematics and Chemistry
df_scores = pd.DataFrame({'Mathematics': [30,31,32,33,34],
                          'Chemistry': [33,32,31,30,29]},
                         index = ['depu', 'jaison', 'sudeesh', 'shafeek', 'ajmi'] )
```

```
df_scores
```

	Mathematics	Chemistry
depu	30	33
jaison	31	32
sudeesh	32	31
shafeek	33	30
ajmi	34	29

Data Operation with Functions

```
#Declare a custom function
def mark_list(scores):
    if scores == 34:
        return 'A'
    if scores == 33:
        return 'B'
    if scores == 32:
        return 'C'
    if scores == 31:
        return 'D'
    if scores == 30:
        return 'E'
    else:
        return 'F'
```

```
#Test the function
print (mark_list(30))
```

```
# Apply the function to the Dataframe
df_scores.applymap(mark_list)
```

	Mathematics	Chemistry
depu	E	B
jaison	D	C
sudeesh	C	D
shafeek	B	E
ajmi	A	F

Data Operation with Statistical Functions

```
#create a dataframe with two test
df_scores = pd.DataFrame({'Test1': [88,89,90,91,92],
                          'Test2': [84,85,86,87,88]},
                         index = ['depu', 'jaison', 'sudeesh', 'shafeek', 'ajmi'] )
```

```
#Apply the maximum function to find max score
df_scores.max()
```

```
Test1    92
Test2    88
dtype: int64
```

```
# Apply mean function to find avg score
df_scores.mean()
```

```
Test1    90.0
Test2    86.0
dtype: float64
```

```
#Apply standard deviation
df_scores.std()
```

```
Test1    1.581139
Test2    1.581139
dtype: float64
```

Data Operation Using Groupby

```
# Create a dataframe with first and last name
df_names = pd.DataFrame({'first': ['Deepu', 'Jaison', 'Sudheesh', 'Shafeek','Jaison'],
                         'last': ['C', 'Jacob', ' V S', 'P M','J']})
```

```
df_names
```

	first	last
0	Deepu	C
1	Jaison	Jacob
2	Sudheesh	V S
3	Shafeek	P M
4	Jaison	J

```
# Group the dataframe with first name
grouped = df_names.groupby('first')
```

```
#Display the dataframe with first name
grp_data = grouped.get_group('Jaison')
grp_data
```

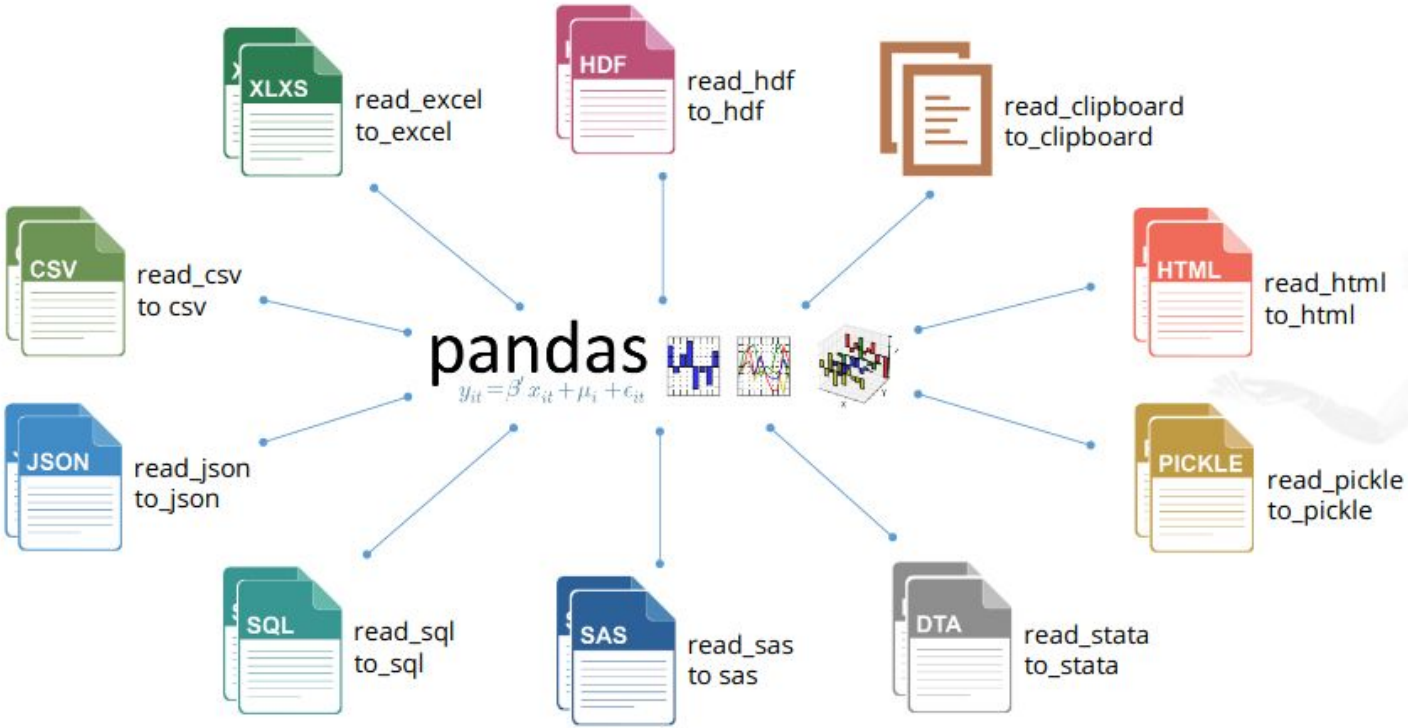
	first	last
1	Jaison	Jacob
4	Jaison	J

Data Operation Using Sorting

```
#sort thse values in ascending order
df_names.sort_values('first')
```

	first	last
0	Deepu	C
1	Jaison	Jacob
4	Jaison	J
3	Shafeek	P M
2	Sudheesh	V S

Data Input and Output



CSV Files

```
import numpy as np
import pandas as pd
```

```
pwd
```

```
'/home/icfoss/Downloads/pandas and seaborn/Pandas'
```

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

```
df.head()
```

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

Json data

```
# json data
data = pd.read_json('datajs.json')
print(data)
```

```
   ID      Name  Salary   StartDate        Dept
0   1      Rick  623.30    1/1/2012          IT
1   2       Dan  515.20   9/23/2013  Operations
2   3  Michelle  611.00  11/15/2014          IT
3   4      Ryan  729.00   5/11/2014          HR
4   5      Gary  843.25   3/27/2015     Finance
5   6      Nina  578.00   5/21/2013          IT
6   7     Simon  632.80   7/30/2013  Operations
7   8      Guru  722.50   6/17/2014     Finance
```

Thanks

Kailas E K