# Data Preprocessing For Machine learning

# About

## Ayham Ramadan

### Machine Learning Engineer

+201229492980

**Mail: ay_ramadan@hotmai.com**

## Work Experience

**DevisionX , Cairo** **Oct 2021 - Present**
Computer Vision Engineer

**Vodafone UK VIS , Cairo** **Aug 2017 - Sep 2018**
Crisis Management Team Leader

**NCCI - for Insurance , Riyadh** **Jun 2015 - Sep 2015**
Oracle Developer

**Information Technology Institute - MCIT , Alexandria Branch Apr 2021 – Jan 2021**
AI – PRO EPITA Certified

**International School of Scandinavia, Copenhagen Dec 2019 – Present**
Project Management and Supply Chain optimization

**Alexandria University of Engineering Jan 2013 - Jan 2017**
BSc in Computer and Communication

# Agenda For Today

## 01 Pandas

What is Pandas and Why is it used ?

## 02 Dataframe

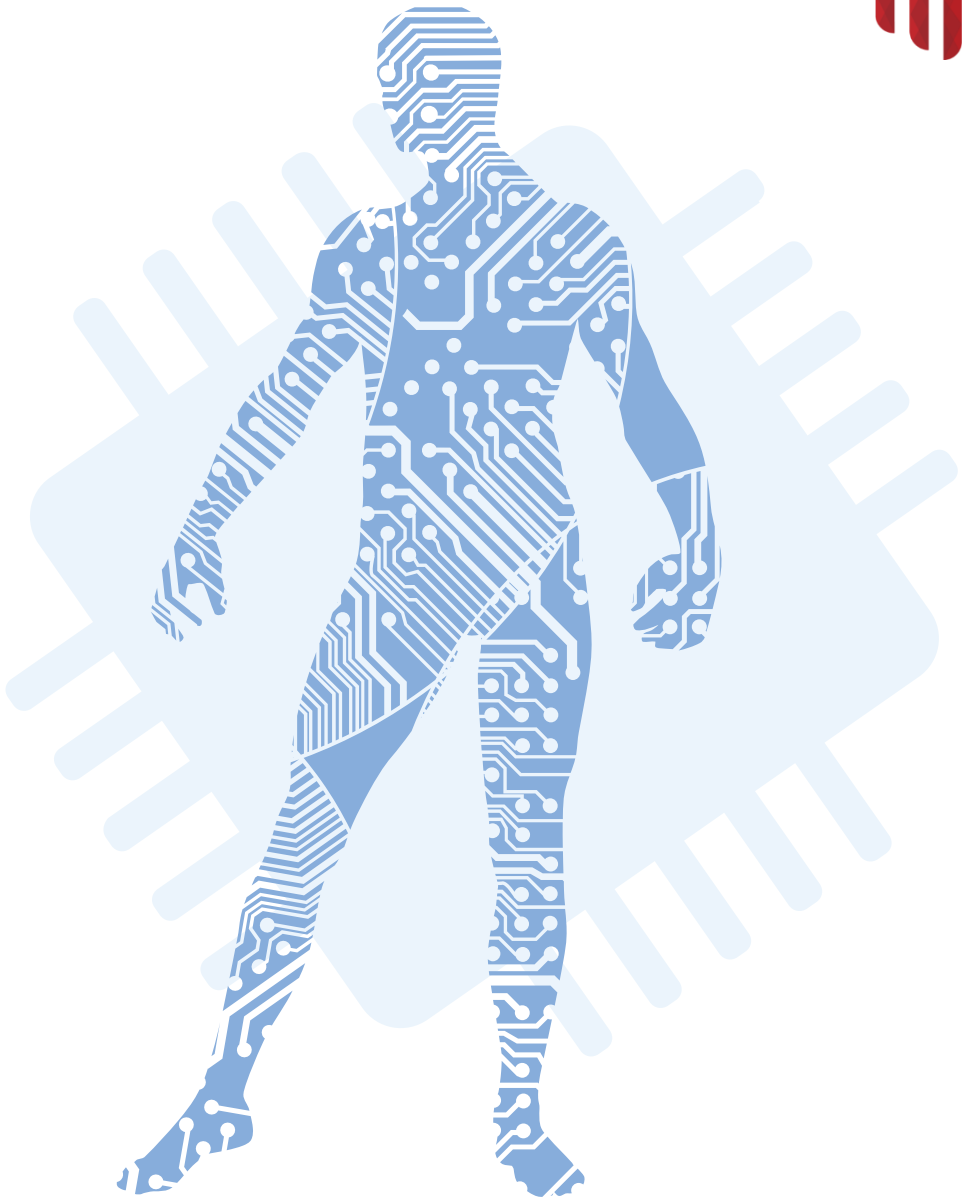
How do we apply changes and Preprocessing on Dataframes?

## 03 Loading Datasets (Practical )

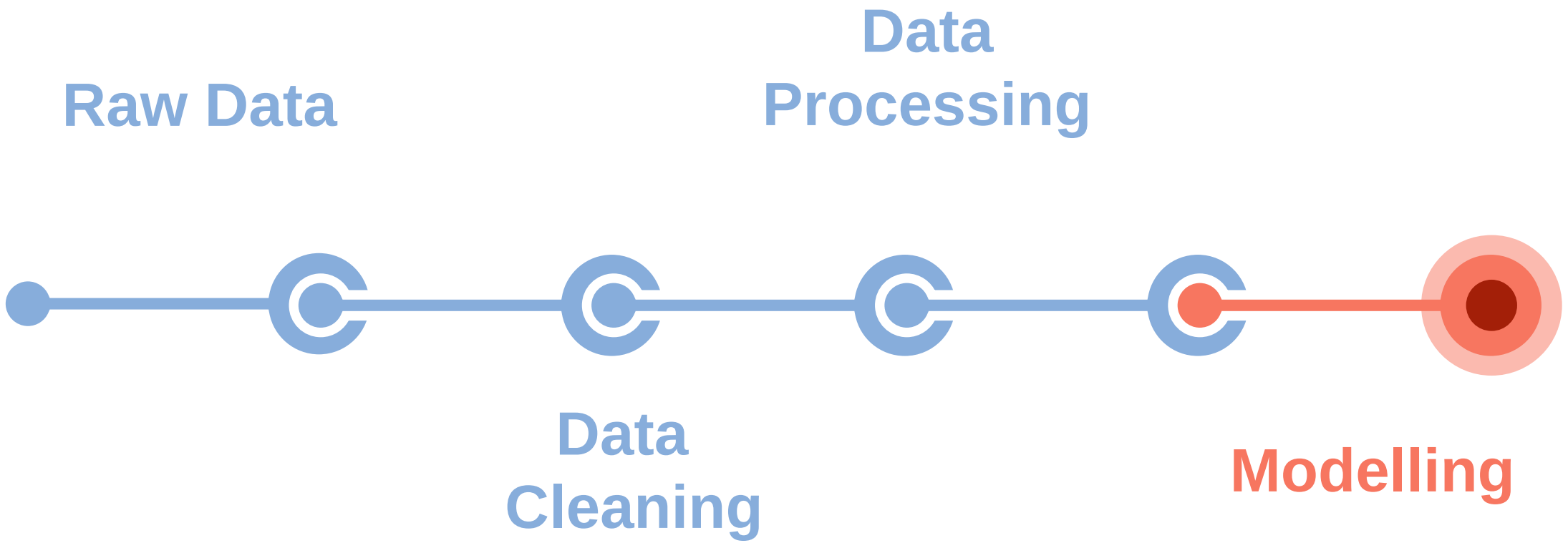
How to load different Datasets and How to use the documentation ?

## 04 Basic Preprocessing

Applying what was learned today?

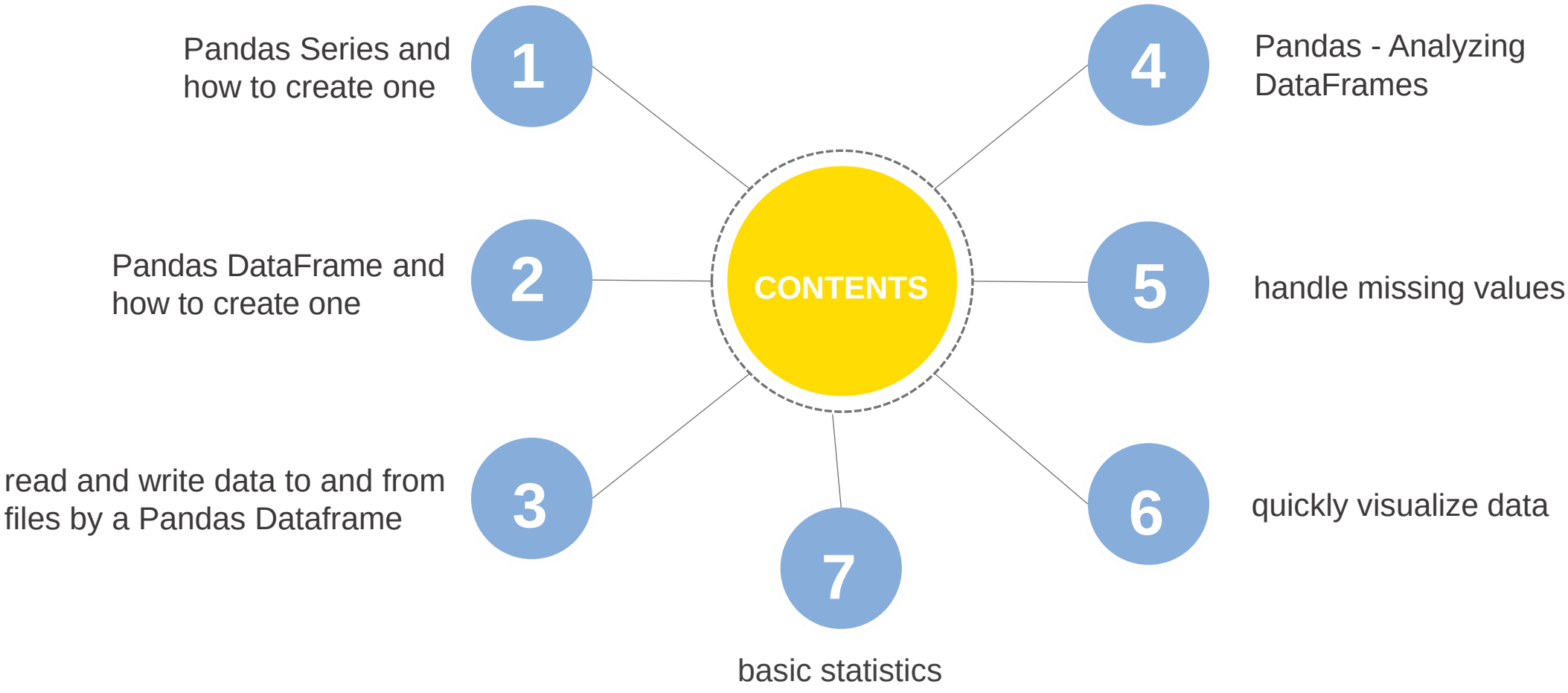


# Data Pipeline



# Pandas

# At this session



## What is Pandas?

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis".
- Why Use Pandas?

  Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy datasets and make them readable and relevant.
- Relevant data is very important in data science.

## Install it using this command:

```
1 !pip install pandas
```

## Checking Pandas Version

The version string is stored under **version** attribute.

```
:
1 import pandas as pd
2
3 print(pd.__version__)
```

```
1.1.5
```

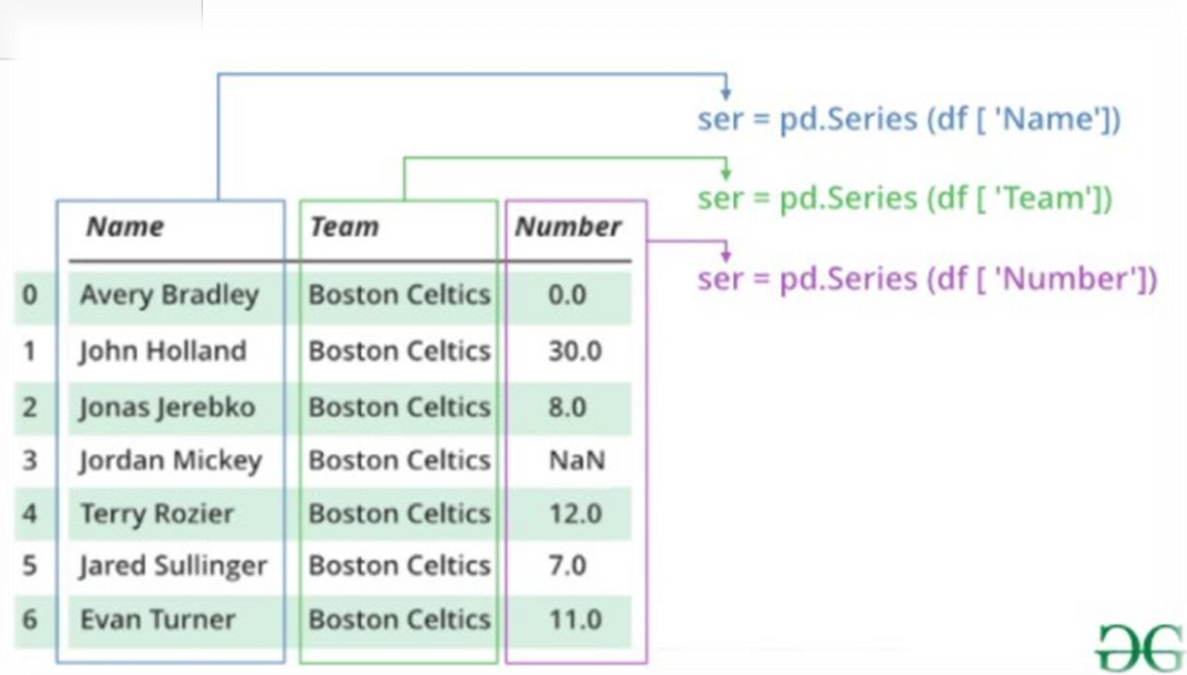
# Pandas Series

A Pandas Series is like a column in a table.
It is a one-dimensional array holding data of any type.

## 1. Creating a series from array

```python
#Create a simple Pandas Series from a list
import pandas as pd

Name = ["Avery Bradley","John Holland", "Jonas Jerebko","Jordan Mickey","Terry Rozier",
        "Jared Sullinger","Evan Turner"]

myvar = pd.Series(Name)

print(myvar)
```

```
0      Avery Bradley
1       John Holland
2      Jonas Jerebko
3      Jordan Mickey
4       Terry Rozier
5    Jared Sullinger
6        Evan Turner
dtype: object
```




| | Name | Team | Number |
|---|---|---|---|
| 0 | Avery Bradley | Boston Celtics | 0.0 |
| 1 | John Holland | Boston Celtics | 30.0 |
| 2 | Jonas Jerebko | Boston Celtics | 8.0 |
| 3 | Jordan Mickey | Boston Celtics | NaN |
| 4 | Terry Rozier | Boston Celtics | 12.0 |
| 5 | Jared Sullinger | Boston Celtics | 7.0 |
| 6 | Evan Turner | Boston Celtics | 11.0 |

# 1.Accessing element of Series

There are two ways through which we can access element of series, they are :

- Accessing Element from Series with Position
- Accessing Element Using Label (index)

**2.1 Accessing Element from Series with Position :** In order to access the series element refers to the index number. Use the index operator [ ] to access an element in a series. The index must be an integer. In order to access multiple elements from a series, we use Slice operation.

Accessing first 5 elements of Series

```
# import pandas and numpy
import pandas as pd
import numpy as np

# creating simple array
data = np.array(['g','e','e','k','s','f', 'o','r','g','e','e','k','s'])
ser = pd.Series(data)
#retrieve the first 5 elements
print("\n",ser[:5])
print("\n",ser[[0,1,2,4]])
#retrieve element
print("\n- 5th ele. is ",ser[5])

```

```
 0    g
1    e
2    e
3    k
4    s
dtype: object

 0    g
1    e
2    e
4    s
dtype: object

- 5th ele. is  f
```

## 2.2 Accessing Element Using Label (index) :

In order to access an element from series, we have to set values by index label. A Series is like a fixed-size dictionary in that you can get and set values by index label.

```
# import pandas and numpy
import pandas as pd
import numpy as np

# creating simple array
data = np.array(['g','e','e','k','s','f', 'o','r','g','e','e','k','s'])
ser = pd.Series(data,index=["10","11","12","13","14","15","16","17","18","19","20","21","22"])

# accessing a element using label "index"
print(ser["16"])
print(ser[["16","15"]])
```

```
o
16    o
15    f
dtype: object
```

# What will be the result if you pass dict into pd.Series() ?

```python
# Create a simple Pandas Series from a dictionary:

import pandas as pd

calories = {"day1": [420,50], "day2": [380,34], "day3": [390,43]}

myvar = pd.Series(calories)

print(myvar)
print("\nDay1 data =",myvar["day1"])
```

```
day1    [420, 50]
day2    [380, 34]
day3    [390, 43]
dtype: object

Day1 data = [420, 50]
```

```python
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)
```

```
day1    420
day2    380
dtype: int64
```

# Binary operation methods on series:

| FUNCTION | DESCRIPTION |
|---|---|
| **add()** | Method is used to add series or list like objects with same length to the caller series |
| **sub()** | Method is used to subtract series or list like objects with same length from the caller series |
| **mul()** | Method is used to multiply series or list like objects with same length with the caller series |
| **div()** | Method is used to divide series or list like objects with same length by the caller series |
| **sum()** | Returns the sum of the values for the requested axis |
| **prod()** | Returns the product of the values for the requested axis |
| **mean()** | Returns the mean of the values for the requested axis |
| **pow()** | Method is used to put each element of passed series as exponential power of caller series and returned the results |
| **abs()** | Method is used to get the absolute numeric value of each element in Series/DataFrame |
| **cov()** | Method is used to find covariance of two series |

# Pandas series method

| FUNCTION | DESCRIPTION |
|---|---|
| **Series()** | A pandas Series can be created with the Series() constructor method. This constructor method accepts a variety of inputs |
| **combine_first()** | Method is used to combine two series into one |
| **count()** | Returns number of non-NA/null observations in the Series |
| **size()** | Returns the number of elements in the underlying data |
| **name()** | Method allows to give a name to a Series object, i.e. to the column |
| **is_unique()** | Method returns boolean if values in the object are unique |
| **idxmax()** | Method to extract the index positions of the highest values in a Series |
| **idxmin()** | Method to extract the index positions of the lowest values in a Series |
| **sort_values()** | Method is called on a Series to sort the values in ascending or descending order |
| **sort_index()** | Method is called on a pandas Series to sort it by the index instead of its values |
| **head()** | Method is used to return a specified number of rows from the beginning of a Series. The method returns a brand new Series |
| **tail()** | Method is used to return a specified number of rows from the end of a Series. The method returns a brand new Series |
| **le()** | Used to compare every element of Caller series with passed series.It returns True for every element which is Less than or Equal to the element in passed series |
| **ne()** | Used to compare every element of Caller series with passed series. It returns True for every element which is Not Equal to the element in passed series |
| **ge()** | Used to compare every element of Caller series with passed series. It returns True for every element which is Greater than or<br>Equal to the element in passed series |
| **eq()** | Used to compare every element of Caller series with passed series. It returns True for every element which is Equal to the element in passed series |

# Pandas series method (Continue )

| | |
|---|---|
| **lt()** | Used to compare two series and return Boolean value for every respective element |
| **clip()** | Used to clip value below and above to passed Least and Max value |
| **clip_lower()** | Used to clip values below a passed least value |
| **clip_upper()** | Used to clip values above a passed maximum value |
| **astype()** | Method is used to change data type of a series |
| **tolist()** | Method is used to convert a series to list |
| **get()** | Method is called on a Series to extract values from a Series. This is alternative syntax to the traditional bracket syntax |
| **unique()** | Pandas unique() is used to see the unique values in a particular column |
| **nunique()** | Pandas nunique() is used to get a count of unique values |
| **value_counts()** | Method to count the number of the times each unique value occurs in a Series |
| **factorize()** | Method helps to get the numeric representation of an array by identifying distinct values |
| **map()** | Method to tie together the values from one object to another |
| **between()** | Pandas between() method is used on series to check which values lie between first and second argument |
| **apply()** | Method is called and feeded a Python function as an argument to use the function on every Series value. This method is helpful for executing custom operations that are not included in pandas or numpy |

# ❑ Pandas DataFrame

**Pandas DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types. It is generally the most commonly used pandas object.
Pandas DataFrame can be created in multiple ways. Let's discuss different ways to create a DataFrame one by one.

## Method #1: Creating Pandas DataFrame from lists of lists.

```
1  # Import pandas library
2  import pandas as pd
3
4  # initialize list of lists
5  data = [['tom', 10], ['nick', 15], ['juli', 14]]
6  # Create the pandas DataFrame
7  df = pd.DataFrame(data, columns = ['Name', 'Age'])
8  # print dataframe.
9  df
10
```

| | Name | Age |
|---|---|---|
| **0** | tom | 10 |
| **1** | nick | 15 |
| **2** | juli | 14 |

## Method #2: Creating DataFrame from dict of narray/lists

To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

```
# Python code demonstrate creating
# DataFrame from dict narray / lists
# By default addresses.

import pandas as pd

# initialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'],
     'Age':[20, 21, 19, 18]}
# Create DataFrame
df = pd.DataFrame(data)
# Print the output.
df

```

| | Name | Age |
|---|---|---|
| 0 | Tom | 20 |
| 1 | nick | 21 |
| 2 | krish | 19 |
| 3 | jack | 18 |

## **Method #3:** Creates a indexes DataFrame using arrays.

```
# Python code demonstrate creating
# pandas DataFrame with indexed by

# DataFrame using arrays.
import pandas as pd
# initialise data of lists.
data = {'Name':['Tom', 'Jack', 'nick', 'juli'],'marks':[99, 98, 95, 90]}
# Creates pandas DataFrame.
df = pd.DataFrame(data, index =['rank1',
'rank2','rank3','rank4'])
# print the data
df

```

| | Name | marks |
|---|---|---|
| **rank1** | Tom | 99 |
| **rank2** | Jack | 98 |
| **rank3** | nick | 95 |
| **rank4** | juli | 90 |

## **Method #4:** Creating Dataframe from list of dicts

Pandas DataFrame can be created by passing lists of dictionaries as a input data. By default dictionary keys taken as columns.

```python
# Python code demonstrate how to create
# Pandas DataFrame by lists of dicts.
import pandas as pd
# Initialise data to lists.
data = [{'a': 1, 'b': 2, 'c':3},
{'a':10, 'b': 20, 'c': 30}]
# Creates DataFrame.
df = pd.DataFrame(data)
# Print the data
df

```

| | a | b | c |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 1 | 10 | 20 | 30 |

## **Method #5:** Creating DataFrame using zip() function.

Two lists can be merged by using list(zip()) function. Now, create the pandas DataFrame by calling pd.DataFrame() function.

```python
# Python program to demonstrate creating
# pandas Datadaframe from lists using zip.

import pandas as pd
# List1
Name = ['tom', 'krish', 'nick', 'juli']
# List2
Age = [25, 30, 26, 22]
# get the list of tuples from two lists.
# and merge them by using zip().
list_of_tuples = list(zip(Name, Age))
# Assign data to tuples.
list_of_tuples
# Converting lists of tuples into
# pandas Dataframe.
df = pd.DataFrame(list_of_tuples,
columns = ['Name', 'Age'])
# Print data.
df

```

| | Name | Age |
|---|---|---|
| 0 | tom | 25 |
| 1 | krish | 30 |
| 2 | nick | 26 |
| 3 | juli | 22 |

```
[('tom', 25), ('krish', 30), ('nick', 26), ('juli', 22)]
```

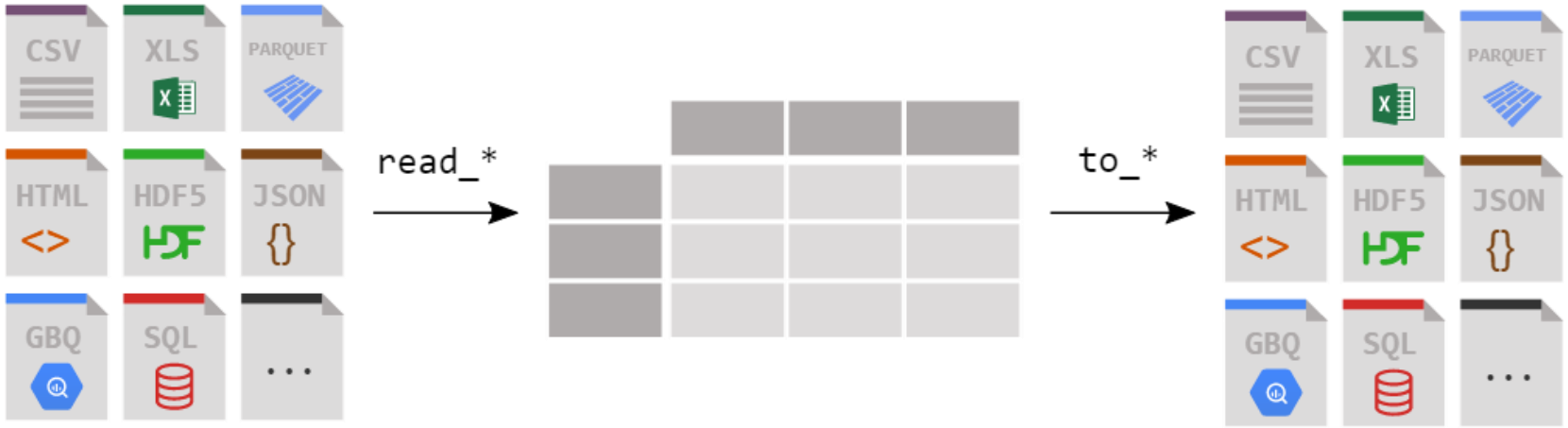
**Method #6: Creating DataFrame from Dicts of series.**

To create DataFrame from Dicts of series, dictionary can be passed to form a DataFrame. The resultant index is the union of all the series of passed indexed.

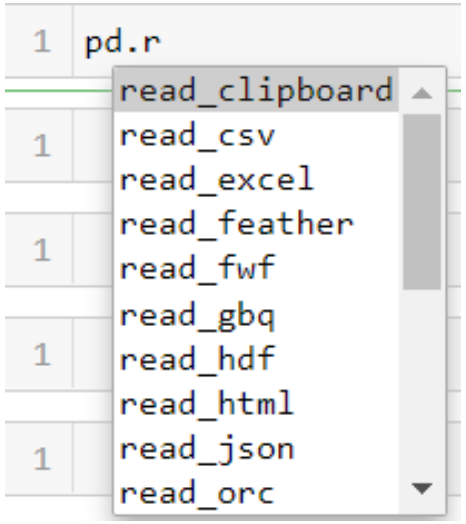
```
# Python code demonstrate creating
# Pandas Dataframe from Dicts of series.

import pandas as pd

# Initialise data to Dicts of series.
d = {'one' : pd.Series([10, 20, 30, 40],
index =['a', 'b', 'c', 'd']),'two' : pd.Series([10, 20, 30, 40],index =['a', 'b', 'c', 'd'])}

# creates Dataframe.
df = pd.DataFrame(d)

# print the data.
df

```

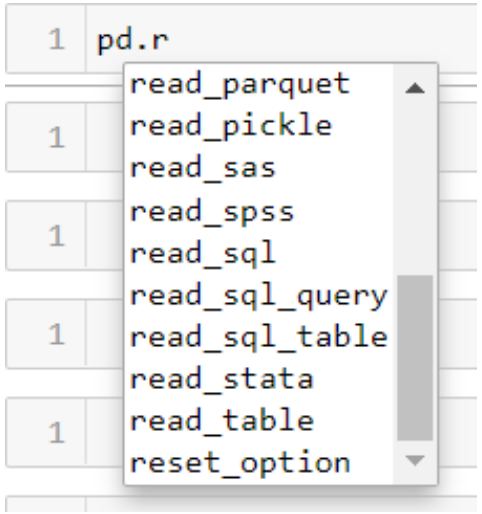
| | one | two |
|---|---|---|
| **a** | 10 | 10 |
| **b** | 20 | 20 |
| **c** | 30 | 30 |
| **d** | 40 | 40 |

# ❑ Read and write data to and from files by a Pandas Dataframe




•pandas provides the **read_csv()** function to read data stored as a csv file into a pandas DataFrame. pandas supports many different file formats or data sources out of the box (csv, excel, sql, json, parquet, …),
each of them with the prefix read_*.




https://pandas.pydata.org/docs/reference/io.html

# 1. Pandas Read CSV

A simple way to store big data sets is to use CSV files (comma separated files).
CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

```
1 #Load the CSV into a DataFrame:
2 import pandas as pd
3 df = pd.read_csv('dataset/data.csv')
4 print(df)
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

Make sure to always have a check on the data after reading in the data. When displaying a DataFrame, the first and last 5 rows will be shown by default

- df.head(): by default, display first 5 rows from data.
- df.tail() : by default, display last 5 rows from data.

```
1 df.head()
```

| | Duration | Pulse | Maxpulse | Calories |
|---|---|---|---|---|
| **0** | 60 | 110 | 130 | 409.1 |
| **1** | 60 | 117 | 145 | 479.0 |
| **2** | 60 | 103 | 135 | 340.0 |
| **3** | 45 | 109 | 175 | 282.4 |
| **4** | 45 | 117 | 148 | 406.0 |

```
1 df.tail()
```

| | Duration | Pulse | Maxpulse | Calories |
|---|---|---|---|---|
| **164** | 60 | 105 | 140 | 290.8 |
| **165** | 60 | 110 | 145 | 300.0 |
| **166** | 60 | 115 | 145 | 310.2 |
| **167** | 75 | 120 | 150 | 320.4 |
| **168** | 75 | 125 | 150 | 330.4 |

```
1 df.head(10)
```

| | Duration | Pulse | Maxpulse | Calories |
|---|---|---|---|---|
| **0** | 60 | 110 | 130 | 409.1 |
| **1** | 60 | 117 | 145 | 479.0 |
| **2** | 60 | 103 | 135 | 340.0 |
| **3** | 45 | 109 | 175 | 282.4 |
| **4** | 45 | 117 | 148 | 406.0 |
| **5** | 60 | 102 | 127 | 300.0 |
| **6** | 60 | 110 | 136 | 374.0 |
| **7** | 45 | 104 | 134 | 253.3 |
| **8** | 30 | 109 | 133 | 195.1 |
| **9** | 60 | 98 | 124 | 269.0 |

```
1 df.tail(10)
```

| | Duration | Pulse | Maxpulse | Calories |
|---|---|---|---|---|
| **159** | 30 | 80 | 120 | 240.9 |
| **160** | 30 | 85 | 120 | 250.4 |
| **161** | 45 | 90 | 130 | 260.4 |
| **162** | 45 | 95 | 130 | 270.0 |
| **163** | 45 | 100 | 140 | 280.9 |
| **164** | 60 | 105 | 140 | 290.8 |
| **165** | 60 | 110 | 145 | 300.0 |
| **166** | 60 | 115 | 145 | 310.2 |
| **167** | 75 | 120 | 150 | 320.4 |
| **168** | 75 | 125 | 150 | 330.4 |

# Reading in files with encoding problems

Most files you'll encounter will probably be encoded with UTF-8. This is what Python expects by default, so most of the time you won't run into problems. However, sometimes you'll get an error like this:

```
: 1 # modules we'll use
  2 import numpy as np
  3 # helpful character encoding module
  4 import chardet
```

```
: 1 police_killings =pd.read_csv("PoliceKillingsUS.csv")
```

```
pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader.read()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._read_low_memory()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._read_rows()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._convert_column_data()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._convert_tokens()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._convert_with_dtype()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._string_convert()

pandas\_libs\parsers.pyx in pandas._libs.parsers._string_box_utf8()

UnicodeDecodeError: 'utf-8' codec can't decode byte 0x96 in position 2: invalid start byte
```

## *chardet.detect()*

The detect function takes one argument, a non-Unicode string. It returns a dictionary containing the auto-detected character encoding and a confidence level from 0 to 1

```
1 with open("PoliceKillingsUS.csv", 'rb') as rawdata:
2     result = chardet.detect(rawdata.read(100000))
3
4 # check what the character encoding might be
5 print(result)
```

```
{'encoding': 'Windows-1252', 'confidence': 0.73, 'language': ''}
```

**encoding attribute : https://docs.python.org/3/library/codecs.html#standard-encodings**

```
1 police_killings =pd.read_csv("PoliceKillingsUS.csv",encoding='Windows-1252')
2 police_killings
```

| | id | name | date | manner_of_death | armed | age | gender | race | city | state | signs_of_mental_illness | threat_level | flee | body_camer |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | Tim Elliot | 02/01/15 | shot | gun | 53.0 | M | A | Shelton | WA | True | attack | Not fleeing | Fals |
| **1** | 4 | Lewis Lee Lembke | 02/01/15 | shot | gun | 47.0 | M | W | Aloha | OR | False | attack | Not fleeing | Fals |
| **2** | 5 | John Paul Quintero | 03/01/15 | shot and Tasered | unarmed | 23.0 | M | H | Wichita | KS | False | other | Not fleeing | Fals |

# Save the DataFrame to csv file

DataFrame.to_csv(path_or_buf=None, sep=',', na_rep='', float_format=None, columns=None, header=True, index=True, index_label=None, mode='w', encoding=None, compression='infer', quoting=None, quotechar='"', line_terminator=None, chunksize=None, date_format=None, doublequote=True, escapechar=None, decimal='.', errors='strict', storage_options=None)

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html#pandas.DataFrame.to_csv

```
police_killings.to_csv("encoded_PK.csv")
```

```
data=pd.read_csv("encoded_PK.csv")
data
```

| | id | name | date | manner_of_death | armed | age | gender | race | city | state | signs_of_mental_illness | threat_level | flee | body_camera |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | Tim Elliot | 02/01/15 | shot | gun | 53.0 | M | A | Shelton | WA | True | attack | Not fleeing | False |
| **1** | 4 | Lewis Lee Lembke | 02/01/15 | shot | gun | 47.0 | M | W | Aloha | OR | False | attack | Not fleeing | False |
| **2** | 5 | John Paul Quintero | 03/01/15 | shot and Tasered | unarmed | 23.0 | M | H | Wichita | KS | False | other | Not fleeing | False |
| **3** | 8 | Matthew Hoffman | 04/01/15 | shot | toy weapon | 32.0 | M | W | San Francisco | CA | True | attack | Not fleeing | False |
| **4** | 9 | Michael Rodriguez | 04/01/15 | shot | nail gun | 39.0 | M | H | Evans | CO | False | attack | Not fleeing | False |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

# Pandas - Analyzing DataFrames

access, modify, add, sort, filter, and delete data

## 1. Viewing and getting information about data

1. DataFrame.head(N=5): view the first 5 rows by default
2. DataFrame.tail(N=5): view the last 5 rows by default
3. DataFrame.info() :The DataFrames object has a method called info(), that gives you more information about the data set
4. DataFrame.index : to view the DataFrame indexes
5. DataFrame.columns : to view the dataframe columns name

## 2. Access, modify

### data Access

1. DataFrame[“column name”]: access certain column from the main dataframe
2. DataFrame[[“column names list “]]: access multiples columns from the main dataframe
3. DataFrame. Loc[row’s name, column’s name] : access data point by row and column names
4. DataFrame. iLoc[row’s index, column’s index] :access data point by row and column indexes

### modify

1. DataFrame[“column name”]= New value
2. DataFrame[[“column names list “]] =New value
3. DataFrame. Loc[row’s name, column’s name] = New value
4. DataFrame. iLoc[row’s index, column’s index] =New value
5. DataFrame.index = new index array

6.DataFrame.columns = new column names array

### Add new column/s

DataFrame[“new column name”]= value/array of values

## 3. Delete row/column

**DataFrame.drop(**labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise'**)**

```
file_dir="default of credit card clients.csv"
Raw_data = pd.read_csv(file_dir,index_col=1,header=1)
Raw_data.head()
```

| ID | 0 | LIMIT_BAL | SEX | EDUCATION | MARRIAGE | AGE | PAY_0 | PAY_2 | PAY_3 | PAY_4 | ... | BILL_AMT4 | BILL_AMT5 | BILL_AMT6 | PAY_AMT1 | PAY_AMT2 | PAY |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 20000 | female | university | married | 24 | 2 | 2 | -1 | -1 | ... | 0 | 0 | 0 | 0 | 689 | |
| 2 | 2 | 120000 | female | university | single | 26 | -1 | 2 | 0 | 0 | ... | 3272 | 3455 | 3261 | 0 | 1000 | |
| 3 | 3 | 90000 | female | university | single | 34 | 0 | 0 | 0 | 0 | ... | 14331 | 14948 | 15549 | 1518 | 1500 | |
| 4 | 4 | 50000 | female | university | married | 37 | 0 | 0 | 0 | 0 | ... | 28314 | 28959 | 29547 | 2000 | 2019 | |
| 5 | 5 | 50000 | male | university | married | 57 | -1 | 0 | -1 | 0 | ... | 20940 | 19146 | 19131 | 2000 | 36681 | |

5 rows × 25 columns

```
Raw_data.drop(labels="0",axis=1)
```

| ID | LIMIT_BAL | SEX | EDUCATION | MARRIAGE | AGE | PAY_0 | PAY_2 | PAY_3 | PAY_4 | PAY_5 | ... | BILL_AMT4 | BILL_AMT5 | BILL_AMT6 | PAY_AMT1 | PAY_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 20000 | female | university | married | 24 | 2 | 2 | -1 | -1 | -2 | ... | | | | | |
| 2 | 120000 | female | university | single | 26 | -1 | 2 | 0 | 0 | 0 | ... | | | | | |
| 3 | 90000 | female | university | single | 34 | 0 | 0 | 0 | 0 | 0 | ... | | | | | |

```
Raw_data.drop(columns=["0"])
```

| ID | LIMIT_BAL | SEX | EDUCATION | MARRIAGE | AGE | PAY_0 | PAY_2 | PAY_3 | PAY_4 | PAY_5 | ... | BILL_AMT4 | BILL_AMT5 | BILL_AMT6 | PAY_AMT1 | PAY_AM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 20000 | female | university | married | 24 | 2 | 2 | -1 | -1 | -2 | ... | 0 | 0 | 0 | 0 | 6 |
| 2 | 120000 | female | university | single | 26 | -1 | 2 | 0 | 0 | 0 | ... | 3272 | 3455 | 3261 | 0 | 10 |
| 3 | 90000 | female | university | single | 34 | 0 | 0 | 0 | 0 | 0 | ... | 14331 | 14948 | 15549 | 1518 | 15 |
| 4 | 50000 | female | university | married | 37 | 0 | 0 | 0 | 0 | 0 | ... | 28314 | 28959 | 29547 | 2000 | 20 |

# 1. Data cleaning :

1. Duplicated Data
   I. check the existence of duplicated data using DataFrame.duplicated()
   **DataFrame.duplicated(**subset=None, keep='first')
   II. then, remove it using DataFrame.drop_duplicates(inplace=True)
   **DataFrame.drop_duplicates(**subset=None, keep='first', inplace=False, ignore_index=False**)**

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop_duplicates.html
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.duplicated.html

```
1  Raw_data.duplicated()
```

```
1        False
2        False
3        False
4        False
5        False
         ...
29996    False
29997    False
29998    False
29999    False
30000    False
Length: 30000, dtype: bool
```

```
1  Raw_data.duplicated().sum()
```

```
35
```

```
1  Raw_data.drop_duplicates(inplace=True)
```

```
1  Raw_data.duplicated().sum()
```

```
0
```

1. Filtration : DataFrame.filter() it is used to select subset of data

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.filter.html

2. missing values

## 3. Check the features datatype : DataFrame.dtypes()

```
1 data_A_filteration.dtypes
2 ### this line represent that all dataset features saved as object
```

```
ID
LIMIT_BAL                    object
SEX                          object
EDUCATION                    object
MARRIAGE                     object
AGE                          object
PAY_0                        object
PAY_2                        object
PAY_3                        object
PAY_4                        object
PAY_5                        object
PAY_6                        object
BILL_AMT1                    object
BILL_AMT2                    object
BILL_AMT3                    object
BILL_AMT4                    object
BILL_AMT5                    object
BILL_AMT6                    object
PAY_AMT1                     object
PAY_AMT2                     object
PAY_AMT3                     object
PAY_AMT4                     object
PAY_AMT5                     object
PAY_AMT6                     object
default payment next month   object
dtype: object
```

## 4. Convert data from one type to another:

**DataFrame.astype(dtype)**
**DataFrame[column names array].astype(dtype)**
**DataFrame.astype({"col1_name": "int32","column2_name":"int64"})**

```
1 data_A_filteration[numerical_columns]=data_A_filteration[numerical_columns].astype("int64")
```

```
1 data_A_filteration.dtypes
```

```
ID
LIMIT_BAL                     int64
SEX                          object
EDUCATION                    object
MARRIAGE                     object
AGE                           int64
PAY_0                         int64
PAY_2                         int64
PAY_3                         int64
PAY_4                         int64
PAY_5                         int64
PAY_6                         int64
BILL_AMT1                     int64
BILL_AMT2                     int64
BILL_AMT3                     int64
BILL_AMT4                     int64
BILL_AMT5                     int64
BILL_AMT6                     int64
PAY_AMT1                      int64
```

# How to analyze , visualize and deal with categorical data ?

| ID | LIMIT_BAL | SEX | EDUCATION | MARRIAGE | AGE | PAY_0 | PAY_2 | PAY_3 | PAY_4 | PAY_5 | ... | BILL_AMT4 | BILL_AMT5 | BILL_AMT6 | PAY_AMT1 | PAY_AMT2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 20000 | female | university | married | 24 | 2 | 2 | -1 | -1 | -2 | ... | 0 | 0 | 0 | 0 | 689 |
| 2 | 120000 | female | university | single | 26 | -1 | 2 | 0 | 0 | 0 | ... | 3272 | 3455 | 3261 | 0 | 1000 |
| 3 | 90000 | female | university | single | 34 | 0 | 0 | 0 | 0 | 0 | ... | 14331 | 14948 | 15549 | 1518 | 1500 |
| 4 | 50000 | female | university | married | 37 | 0 | 0 | 0 | 0 | 0 | ... | 28314 | 28959 | 29547 | 2000 | 2019 |
| 5 | 50000 | male | university | married | 57 | -1 | 0 | -1 | 0 | 0 | ... | 20940 | 19146 | 19131 | 2000 | 36681 |

5 rows × 24 columns

```
1 data_A_filteration["EDUCATION"].unique()
```

```
array(['university', 'graduate school', 'others', 'high school', 0],
      dtype=object)
```

```
1 data_A_filteration["EDUCATION"].value_counts()
```

```
university         13857
graduate school    10513
high school         4811
others               121
0                     14
Name: EDUCATION, dtype: int64
```

```
1 print("University =",(data_A_filteration["EDUCATION"]== 'university' ).sum())
2 print("graduate school =",(data_A_filteration["EDUCATION"]== 'graduate school' ).sum())
3 print("others =",(data_A_filteration["EDUCATION"]== 'others' ).sum())
4 print("high school =",(data_A_filteration["EDUCATION"]== 'high school' ).sum())
5 print("0 =",(data_A_filteration["EDUCATION"]== 0 ).sum())
```

```
University = 13857
graduate school = 10513
others = 121
high school = 4811
0 = 14
```

DataFrame.unique()
DataFrame.value_counts()
DataFrame.values

```
# Let's see the data types and non-null values for each column
data_After_pro.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 29316 entries, 1 to 30000
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   LIMIT_BAL                   29316 non-null  int64
 1   SEX_male                    29316 non-null  int64
 2   EDUCATION_encoded           29316 non-null  int32
 3   Marital_state               29316 non-null  int64
 4   AGE                         29316 non-null  int64
 5   PAY_0                       29316 non-null  int64
 6   PAY_2                       29316 non-null  int64
 7   PAY_3                       29316 non-null  int64
 8   PAY_4                       29316 non-null  int64
 9   PAY_5                       29316 non-null  int64
 10  PAY_6                       29316 non-null  int64
 11  BILL_AMT1                   29316 non-null  int64
 12  BILL_AMT2                   29316 non-null  int64
 13  BILL_AMT3                   29316 non-null  int64
 14  BILL_AMT4                   29316 non-null  int64
 15  BILL_AMT5                   29316 non-null  int64
 16  BILL_AMT6                   29316 non-null  int64
 17  PAY_AMT1                    29316 non-null  int64
 18  PAY_AMT2                    29316 non-null  int64
 19  PAY_AMT3                    29316 non-null  int64
 20  PAY_AMT4                    29316 non-null  int64
 21  PAY_AMT5                    29316 non-null  int64
 22  PAY_AMT6                    29316 non-null  int64
 23  default payment next month  29316 non-null  int64
dtypes: int32(1), int64(23)
memory usage: 6.7+ MB
```

# ❑Handling missing values

The first thing to do when you get a new dataset is take a look at some of it. This lets you see that it all read in correctly and gives an idea of what's going on with the data. In this case, let's see if there are any missing values, which will be represented with NaN or None.

| | Date | GameID | Drive | qtr | down | time | TimeUnder | TimeSecs | PlayTimeDiff | SideofField | ... | yacEPA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2009-09-10 | 2009091000 | 1 | 1 | NaN | 15:00 | 15 | 3600.0 | 0.0 | TEN | ... | NaN |
| 1 | 2009-09-10 | 2009091000 | 1 | 1 | 1.0 | 14:53 | 15 | 3593.0 | 7.0 | PIT | ... | 1.146076 |
| 2 | 2009-09-10 | 2009091000 | 1 | 1 | 2.0 | 14:16 | 15 | 3556.0 | 37.0 | PIT | ... | NaN |
| 3 | 2009-09-10 | 2009091000 | 1 | 1 | 3.0 | 13:35 | 14 | 3515.0 | 41.0 | PIT | ... | -5.03142 |
| 4 | 2009-09-10 | 2009091000 | 1 | 1 | 4.0 | 13:27 | 14 | 3507.0 | 8.0 | PIT | ... | NaN |

# How many missing data points do we have?

Let's see how many we have in each column using dataframe.isnull().sum() , dataframe.isna()

```
1 data.isna().sum()
```

```
ID                    0
Name                  0
Age                   0
Photo                 0
Nationality           0
                     ...
GKHandling           48
GKKicking            48
GKPositioning        48
GKReflexes           48
Release Clause     1564
Length: 88, dtype: int64
```

```
1 data.isnull().sum()
```

```
ID                    0
Name                  0
Age                   0
Photo                 0
Nationality           0
                     ...
GKHandling           48
GKKicking            48
GKPositioning        48
GKReflexes           48
Release Clause     1564
Length: 88, dtype: int64
```

```
1 data.isna()
```

| | ID | Name | Age | Photo | Nationality | Flag | Overall | Potential | Club | Club Logo | ... | Composure | Marking | StandingTackle | SlidingTackle | GKDiving | GKHar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 1 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 2 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 3 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 4 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 18202 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 18203 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 18204 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 18205 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 18206 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |

18207 rows × 88 columns

```
1 data.isnull()
```

| | ID | Name | Age | Photo | Nationality | Flag | Overall | Potential | Club | Club Logo | ... | Composure | Marking | StandingTackle | SlidingTackle | GKDiving | GKHar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 1 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 2 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 3 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 4 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 18202 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 18203 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 18204 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 18205 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |
| 18206 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | |

18207 rows × 88 columns

## It’s very important to figure out why the data is missing

**Is this value missing because it wasn't recorded or because it doesn't exist?**

If a value is missing because it doesn't exist (like the height of the oldest child of someone who doesn't have any children) then it doesn't make sense to try and guess what it might be. These values you probably do want to keep as NaN or change it to zero . On the other hand, if a value is missing because it wasn't recorded, then you can try  to guess what it might have been based on the other values in that column and row. This is called imputation, and  we'll learn how to do it next! :)

## How to deal with missing values ?

1. drop columns with missing values
2. drop rows with missing values
3. replace them with another value
4. imputation :
    1. replace them with mean , median values  if you deal will numerical data or most frequent value in case of categorical data
    2. KNN  supervised ML model
    3. k-mean  Un-supervised ML model

# 1. Drop missing values :

using DataFrame.dropna()

DataFrame.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)

- By default, drop rows with missing value. But, to drop columns set axis=1
- This function return new data after removing missing values. But, to override the old data use inplace = True
- Subset: Define in which columns to look for missing values in case of dropping rows and visa verse in case of dropping columns
- Thresh: Keep only the rows/columns with at least N non-NA values
- how: “any ” >> drop if any missing value exist , “all” >> drop if all the row / column values is nan

```
In [133]:
1 df = pd.DataFrame({"name": [np.nan, 'Batman', 'Catwoman'],
2                    "toy": [np.nan, 'Batmobile', 'Bullwhip'],
3                    "born": [pd.NaT, pd.Timestamp("1940-04-25"),pd.NaT],
4                      "null":[np.nan,np.nan,np.nan]})
5 df
```

Out[133]:

| | name | toy | born | null |
|---|---|---|---|---|
| 0 | NaN | NaN | NaT | NaN |
| 1 | Batman | Batmobile | 1940-04-25 | NaN |
| 2 | Catwoman | Bullwhip | NaT | NaN |

Dropping rows

```
In [140]:
1 df.dropna(how="all")
```

Out[140]:

| | name | toy | born | null |
|---|---|---|---|---|
| 1 | Batman | Batmobile | 1940-04-25 | NaN |
| 2 | Catwoman | Bullwhip | NaT | NaN |

```
In [141]:
1 df.dropna(how="any")
```

Out[141]:

| name | toy | born | null |
|---|---|---|---|

Dropping columns

```
1 df.dropna(axis=1,how="all")
```

| | name | toy | born |
|---|---|---|---|
| 0 | NaN | NaN | NaT |
| 1 | Batman | Batmobile | 1940-04-25 |
| 2 | Catwoman | Bullwhip | NaT |

```
1 df.dropna(axis=1,how="any")
```

| |
|---|
| 0 |
| 1 |
| 2 |

## thresh:

|  | name | toy | born | null |
|---|---|---|---|---|
| 0 | NaN | NaN | NaT | NaN |
| 1 | Batman | Batmobile | 1940-04-25 | NaN |
| 2 | Catwoman | Bullwhip | NaT | NaN |

```
#Keep only the rows with at least 2 non-NA values.
df.dropna(thresh=2)
```

|  | name | toy | born | null |
|---|---|---|---|---|
| 1 | Batman | Batmobile | 1940-04-25 | NaN |
| 2 | Catwoman | Bullwhip | NaT | NaN |

```
#Keep only the rows with at least 3 non-NA values.
df.dropna(thresh=3)
```

|  | name | toy | born | null |
|---|---|---|---|---|
| 1 | Batman | Batmobile | 1940-04-25 | NaN |

## subset:

```
#Define in which columns to look for missing values.
df.dropna(subset=['name', 'toy'])
```

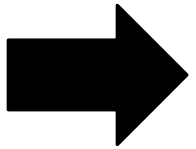
|  | name | toy | born | null |
|---|---|---|---|---|
| 1 | Batman | Batmobile | 1940-04-25 | NaN |
| 2 | Catwoman | Bullwhip | NaT | NaN |

## 1 Filling in missing values:

We can use the Panda's fillna() function to fill in missing values in a dataframe for us

```
1 df
```

| | name | toy | born | null |
|---|---|---|---|---|
| **0** | NaN | NaN | NaT | NaN |
| **1** | Batman | Batmobile | 1940-04-25 | NaN |
| **2** | Catwoman | Bullwhip | NaT | NaN |



```
1 df.fillna(0)
```

| | name | toy | born | null |
|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0.0 |
| **1** | Batman | Batmobile | 1940-04-25 00:00:00 | 0.0 |
| **2** | Catwoman | Bullwhip | 0 | 0.0 |

**DataFrame.fillna(**value=None, method=None, axis=None, inplace=False, limit=None, downcast=None**)**

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html

# ❑ Pandas - Data Correlations

Finding Relationships
A great aspect of the Pandas module is the corr() method.
The corr() method calculates the relationship between each column in your data set.

•**Dataframe.corr()**

**Types of correlations:**

1. Perfect correlation
2. Positive correlation
3. Negative correlation
4. Bad correlation

**How to visualize the correlation between features?**

1. sns.heatmap(correlation_result_data, annot=True)
2. sns.pairplot(dataframe)
3. sns.scatterplot(x=feature1, y=feature2)

## ❑ Basic statistics https://pandas-docs.github.io/pandas-docs-travis/reference/api/pandas.DataFrame.describe.html

calculate summary statistics using **DataFrame.describe()**

```
1 data_A_filteration[numerical_columns].describe()
```

| | ID | AGE | PAY_0 | PAY_2 | PAY_3 | PAY_4 | PAY_5 | PAY_6 | BILL_AMT1 | BILL_AMT2 | BILL_AMT3 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 29316.000000 | 29316.000000 | 29316.000000 | 29316.000000 | 29316.000000 | 29316.000000 | 29316.000000 | 29316.000000 | 29316.000000 | 2.931600e+04 | ... | |
| mean | 35.426695 | -0.017465 | -0.131259 | -0.164074 | -0.219232 | -0.264224 | -0.288580 | 51042.246316 | 49045.626995 | 4.691128e+04 | ... | |
| std | 9.497365 | 1.125777 | 1.199962 | 1.199591 | 1.171496 | 1.136187 | 1.151949 | 73480.513879 | 71051.572267 | 6.925505e+04 | ... | |
| min | 21.000000 | -2.000000 | -2.000000 | -2.000000 | -2.000000 | -2.000000 | -2.000000 | -165580.000000 | -69777.000000 | -1.572640e+05 | ... | - |
| 25% | 28.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | 3519.500000 | 2975.750000 | 2.646750e+03 | ... | |
| 50% | 34.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 22282.000000 | 21095.500000 | 2.006850e+04 | ... | |
| 75% | 41.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 66807.250000 | 63736.250000 | 5.995375e+04 | ... | |
| max | 267.000000 | 8.000000 | 8.000000 | 8.000000 | 8.000000 | 8.000000 | 8.000000 | 964511.000000 | 983931.000000 | 1.664089e+06 | ... | 9 |

8 rows × 21 columns

**Numerical data**

```
1 data_A_filteration[categorical_columns].describe()
```

| | ID | SEX | EDUCATION | MARRIAGE |
|---|---|---|---|---|
| count | 29316 | 29316 | 29316 | 29316 |
| unique | 2 | 5 | 3 | |
| top | female | university | single | |
| freq | 17692 | 13857 | 15797 | |

**Categorical data**

# Aggregating statistics

## I. Numerical data

```
|: 1 data_A_filteration[numerical_columns].describe()
```

|: 

| | ID | AGE | PAY_0 | PAY_2 | PAY_3 | PAY_4 | PAY_5 | PAY_6 | |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 29316.000000 | 29316.000000 | 29316.000000 | 29316.000000 | 29316.000000 | 29316.000000 | 29316.000000 | 2 |
| **mean** | 35.426695 | -0.017465 | -0.131259 | -0.164074 | -0.219232 | -0.264224 | -0.288580 | 5 |
| **std** | 9.497365 | 1.125777 | 1.199962 | 1.199591 | 1.171496 | 1.136187 | 1.151949 | 7 |
| **min** | 21.000000 | -2.000000 | -2.000000 | -2.000000 | -2.000000 | -2.000000 | -2.000000 | -16 |
| **25%** | 28.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | |
| **50%** | 34.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2 |
| **75%** | 41.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 6 |
| **max** | 267.000000 | 8.000000 | 8.000000 | 8.000000 | 8.000000 | 8.000000 | 8.000000 | 96 |

8 rows × 21 columns

```
|: 1 data_A_filteration["AGE"].mean()
```

|: 35.426695319961794

```
|: 1 data_A_filteration["AGE"].count()
```

|: 29316

```
|: 1 data_A_filteration["AGE"].std()
```

|: 9.497365305617228

```
|: 1 data_A_filteration["AGE"].max()
```

|: 267

# II. Categorical data

```
1 data_A_filteration[categorical_columns].describe()
```

| | ID | SEX | EDUCATION | MARRIAGE |
|---|---|---|---|---|
| **count** | 29316 | 29316 | 29316 | 29316 |
| **unique** | 2 | 5 | 3 | |
| **top** | female | university | single | |
| **freq** | 17692 | 13857 | 15797 | |

```
1 data_A_filteration["SEX"].count()
```

```
29316
```

```
1 data_A_filteration["SEX"].value_counts().count()
```

```
2
```

```
1 data_A_filteration["SEX"].value_counts().idxmax()
```

```
'female'
```

```
1 data_A_filteration["SEX"].value_counts().max()
```

```
17692
```

```
1 data_A_filteration["SEX"].nunique()
```

```
2
```