Pandas

Session Objectives

At the this session:

- Pandas Series and how to create one
- Pandas DataFrame and how to create one
- read and write data to and from files by a Pandas Dataframe
- Pandas - Analyzing DataFrames
- handle missing values
- quickly visualize data
- basic statistics

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:

```
!pip install pandas
```

Checking Pandas Version

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

```
import pandas as pd

print(pd.__version__)
```

```
1.1.5
```

Example:

```python
import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pd.DataFrame(mydataset)

print(myvar)
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

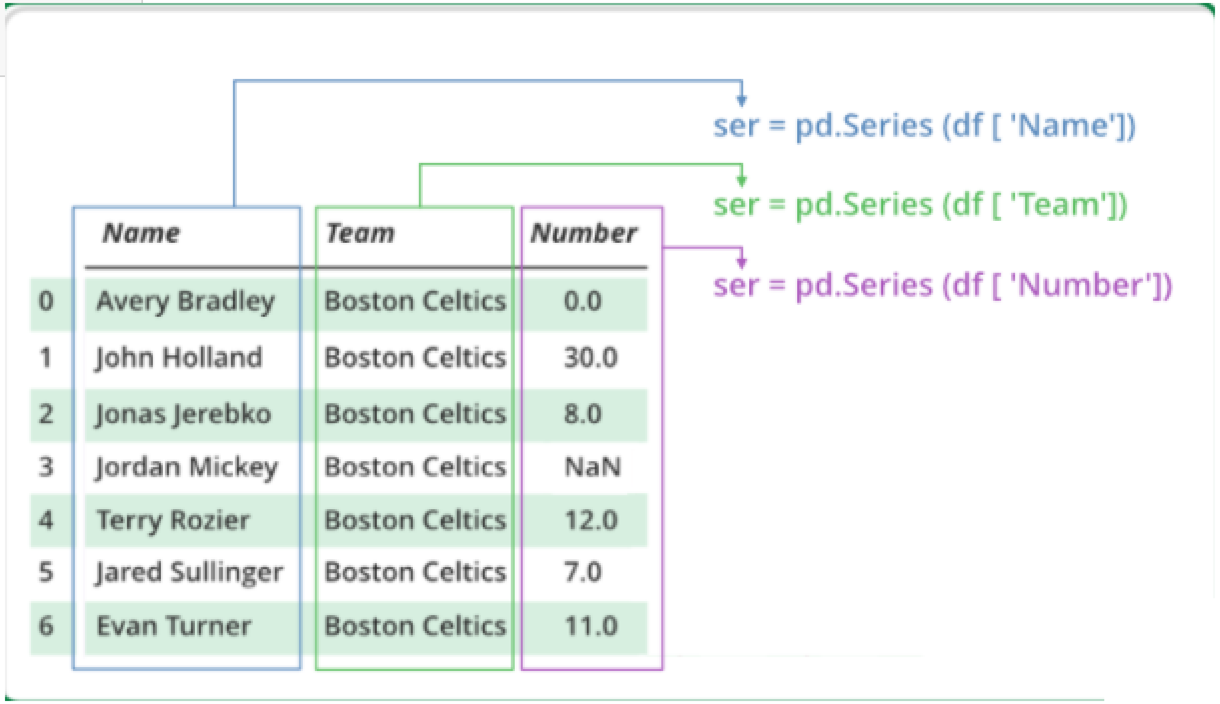
❑ Pandas Series

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
```



2.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

```python
# 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 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

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.

```python
# Import pandas library
import pandas as pd

# initialize list of lists
data = [['tom', 10], ['nick', 15], ['juli', 14]]
# Create the pandas DataFrame
df = pd.DataFrame(data, columns = ['Name', 'Age'])
# print dataframe.
df

```

	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
# 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 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

```

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)]
```

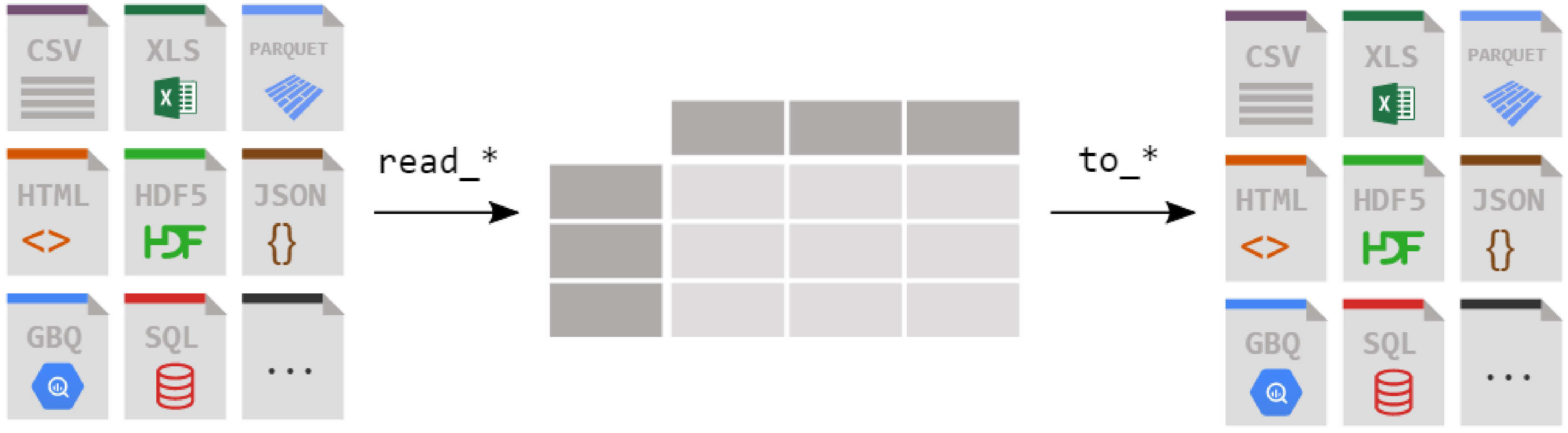
Method #6: Creating DataFrame from Dicts of series.

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.

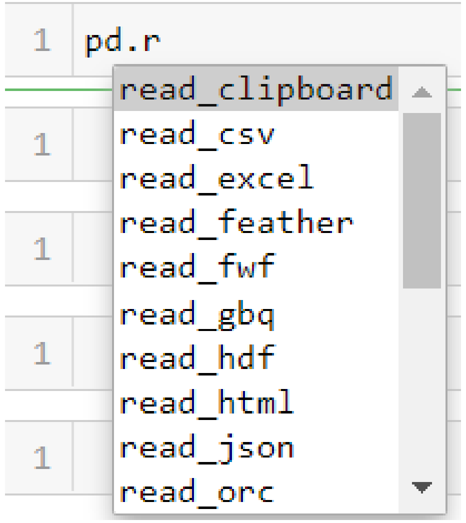
```
# Python code demonstrate creating
# Pandas Dataframe from Dicts of series.

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

```

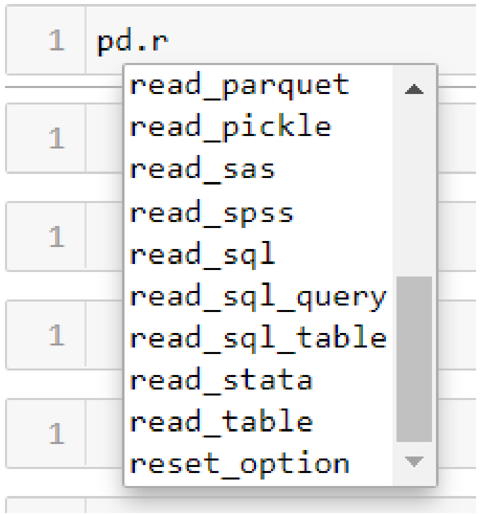
	one	two
a	10	10
b	20	20
c	30	30
d	40	40

❑ 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

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

pandas.read_csv(filepath, sep=<no_default>, delimiter=None, header='infer', names=<no_default>, index_col=None, usecols=None, squeeze=False, prefix=<no_default>, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None, decimal='.’, lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None, error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None, storage_options=None)

Index_col= column number

header= row number

```
file_dir="default of credit card clients.xls"
Raw_data = pd.read_excel(file_dir,index_col=0)
Raw_data.to_csv("clients.csv")
Raw_data.head()

```

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X15	X16	X17	X18	X19	
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	PA
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 rows × 24 columns

```
file_dir="clients.csv"
Raw_data = pd.read_csv(file_dir)
Raw_data.head()
```

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X15	X16	X17	X18	X19
0	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2
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 rows × 25 columns

This data need some modification :

1. Set column unnamed:0 as data indexes column
2. Set row 1 as data columns name

Raw_data = pd.read_csv(file_dir,index_col=0,header=1)

```
Raw_data = pd.read_csv(file_dir,index_col=0,header=1)
Raw_data.head()
```

	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
ID																
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

Why header = 1 not equal 0 ?

By default, header = 0 if column names are not passed while reading data

names: *array-like, optional*

List of column names to use. If the file contains a header row, then you should explicitly pass header=0 to override the column names. Duplicates in this list are not allowed

```python
file_dir="clients.csv"
Raw_data = pd.read_csv(file_dir,header=0,names=['LIMIT', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2',
         'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
         'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
         'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
         'default payment next month'])
Raw_data.head()
```

	LIMIT	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2
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 rows × 24 columns

```python
file_dir="clients.csv"
Raw_data = pd.read_csv(file_dir,header=1,names=['LIMIT', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2',
         'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
         'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
         'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
         'default payment next month'])
Raw_data.head()
```

	LIMIT	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY
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

```python
file_dir="clients.csv"
Raw_data = pd.read_csv(file_dir,header=2,names=['LIMIT', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2',
         'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
         'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
         'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
         'default payment next month'])
Raw_data.head()
```

	LIMIT	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY
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	
6	50000	male	graduate school	single	37	0	0	0	0	0	...	19394	19619	20024	2500	1815	

5 rows × 24 columns

What happens if you try to use names attribute without using header attribute ?

```python
file_dir="clients.csv"
Raw_data = pd.read_csv(file_dir,names=['LIMIT', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2',
        'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
        'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
        'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
        'default payment next month'])
Raw_data.head()
```

	LIMIT	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2
NaN	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X15	X16	X17	X18	X19
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

5 rows × 24 columns

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:

```
# modules we'll use
import numpy as np
# helpful character encoding module
import chardet
```

```
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

```python
with open("PoliceKillingsUS.csv", 'rb') as rawdata:
    result = chardet.detect(rawdata.read(100000))

# check what the character encoding might be
print(result)
```

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

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

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

	id	name	date	manner_of_death	armed	age	gender	race	city	state	signs_of_mental_illness	threat_level	flee	body_camei
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

4. 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
```

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

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

3. missing values

Check the features datatype : **DataFrame.dtypes()**

```
data_A_filteration.dtypes
### 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
```

Convert data from one type to another:

DataFrame.astype(dtype)

DataFrame[column names array].astype(dtype)

DataFrame.astype({“col1_name”: “int32”,”column2_name”:”int64”})

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

```
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

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

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

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

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

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

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

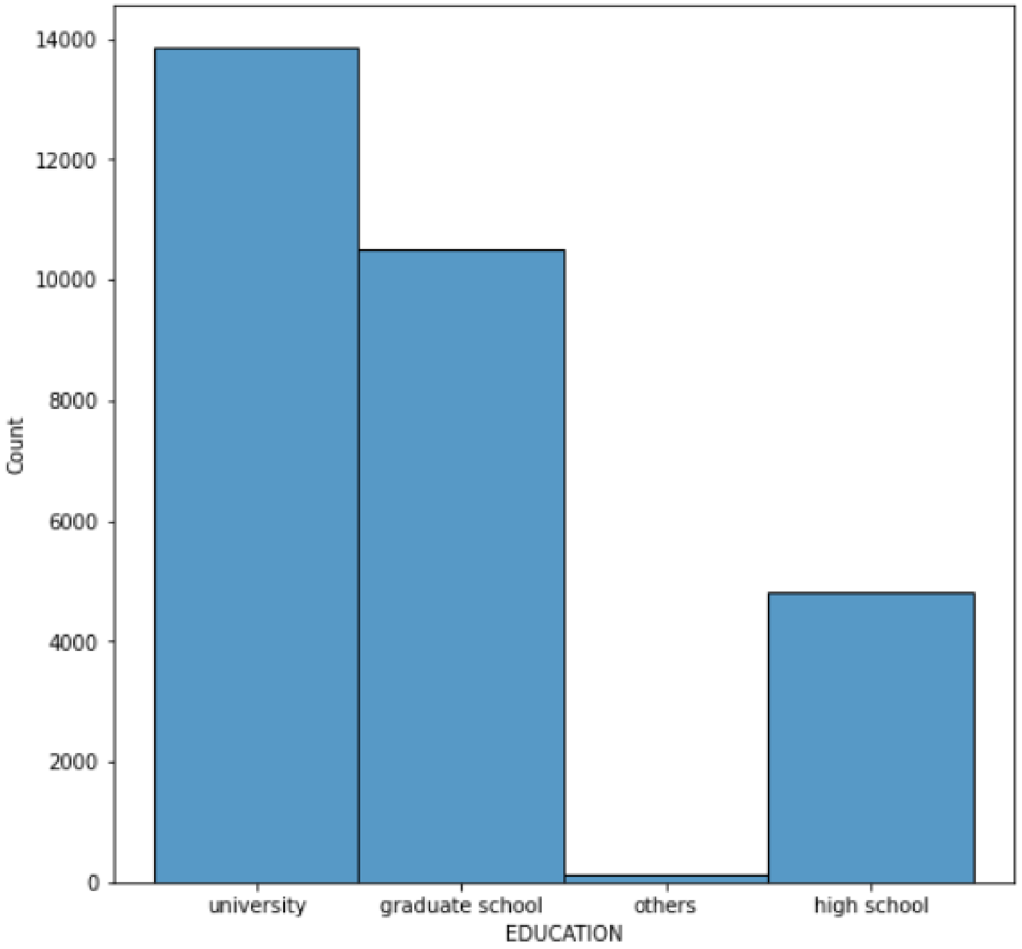
DataFrame.unique()
DataFrame.value_counts()

DataFrame.values

What type of plot should we use to visualize the education feature ?

```
plt.figure(figsize=(8,8))
sns.histplot(data_A_filteration["EDUCATION"])
```

```
<AxesSubplot:xlabel='EDUCATION', ylabel='Count'>
```



Ac

Optional

What kind of processing should be done on categorical data before passing it to the ML model ?

https://www.kaggle.com/alexisbcook/categorical-variables

1. Drop categorical data
2. Ordinal Encoding
3. One-hot encoding

```python
# 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

```
data.head()
```

	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential	Club	
0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	https://cdn.sofifa.org/flags/52.png	94	94	FC Barcelona	https://cdn.sofifa.org/
1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	https://cdn.sofifa.org/flags/38.png	94	94	Juventus	https://cdn.sofifa.org
2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	https://cdn.sofifa.org/flags/54.png	92	93	Paris Saint-Germain	https://cdn.sofifa.org
3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	https://cdn.sofifa.org/flags/45.png	91	93	Manchester United	https://cdn.sofifa.org
4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	91	92	Manchester City	https://cdn.sofifa.org

5 rows × 88 columns

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

```
missing_Data_count_per_column=data.isnull().sum()
```

```
missing_Data_count_per_column
```

```
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
```

```
missing_Data_count_per_column.sum()
```

```
76984
```

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]:

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

```
df.dropna(how="all")
```

Out[140]:

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

In [141]:

```
df.dropna(how="any")
```

Out[141]:

name	toy	born	null

Dropping columns

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

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

```
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

```
df
```

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

```
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)

❑ Pandas - Plotting

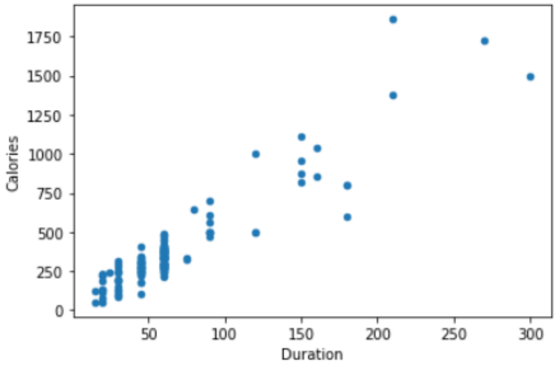
Pandas uses the plot() method to create diagrams.

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

The kind of plot to produce:

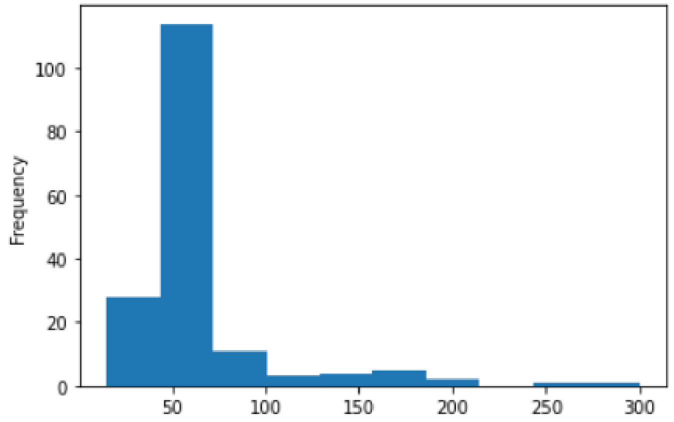
- ‘line’ : line plot (default)
- ‘bar’ : vertical bar plot
- ‘barh’ : horizontal bar plot
- ‘hist’ : histogram
- ‘box’ : boxplot
- ‘kde’ : Kernel Density Estimation plot
- ‘density’ : same as ‘kde’
- ‘area’ : area plot
- ‘pie’ : pie plot
- ‘scatter’ : scatter plot (DataFrame only)
- ‘hexbin’ : hexbin plot (DataFrame only)

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

df = pd.read_csv('data _1.csv')

df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')

plt.show()
```



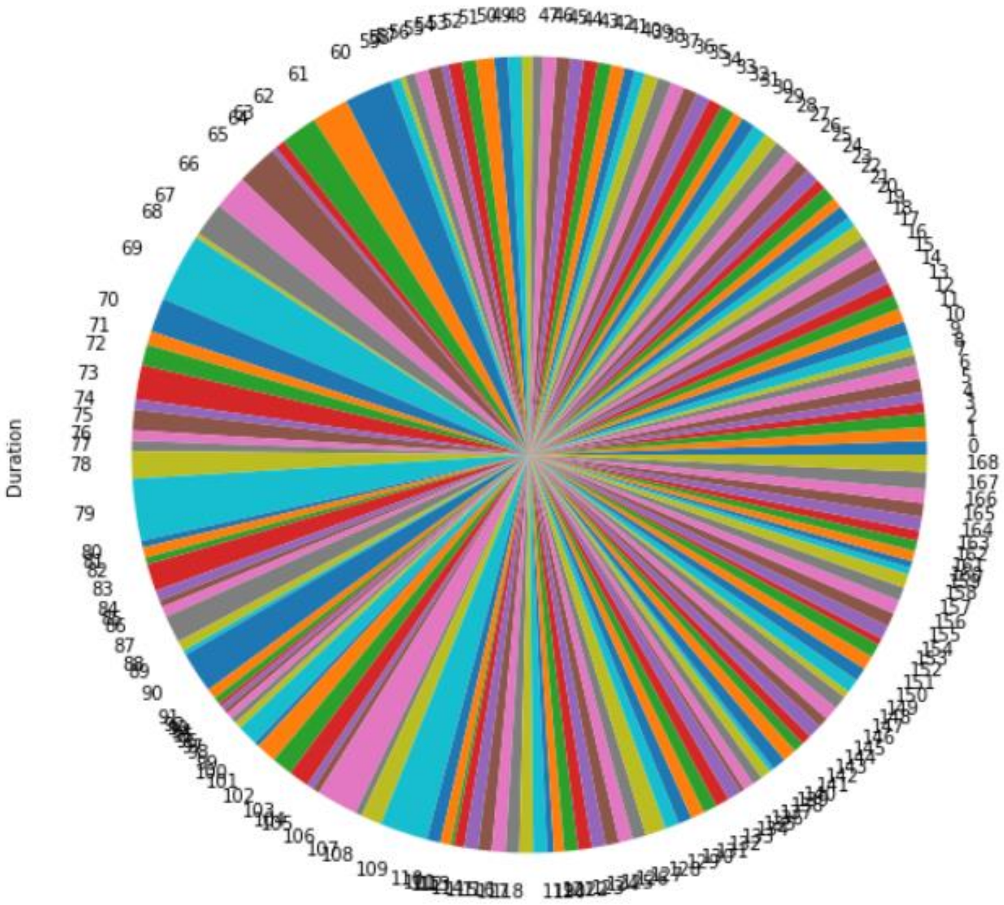

```python
df["Duration"].plot(kind = 'hist')
```

```
<AxesSubplot:ylabel='Frequency'>
```




```python
plt.figure(figsize=(10,10))
df["Duration"].plot(kind="pie")
```

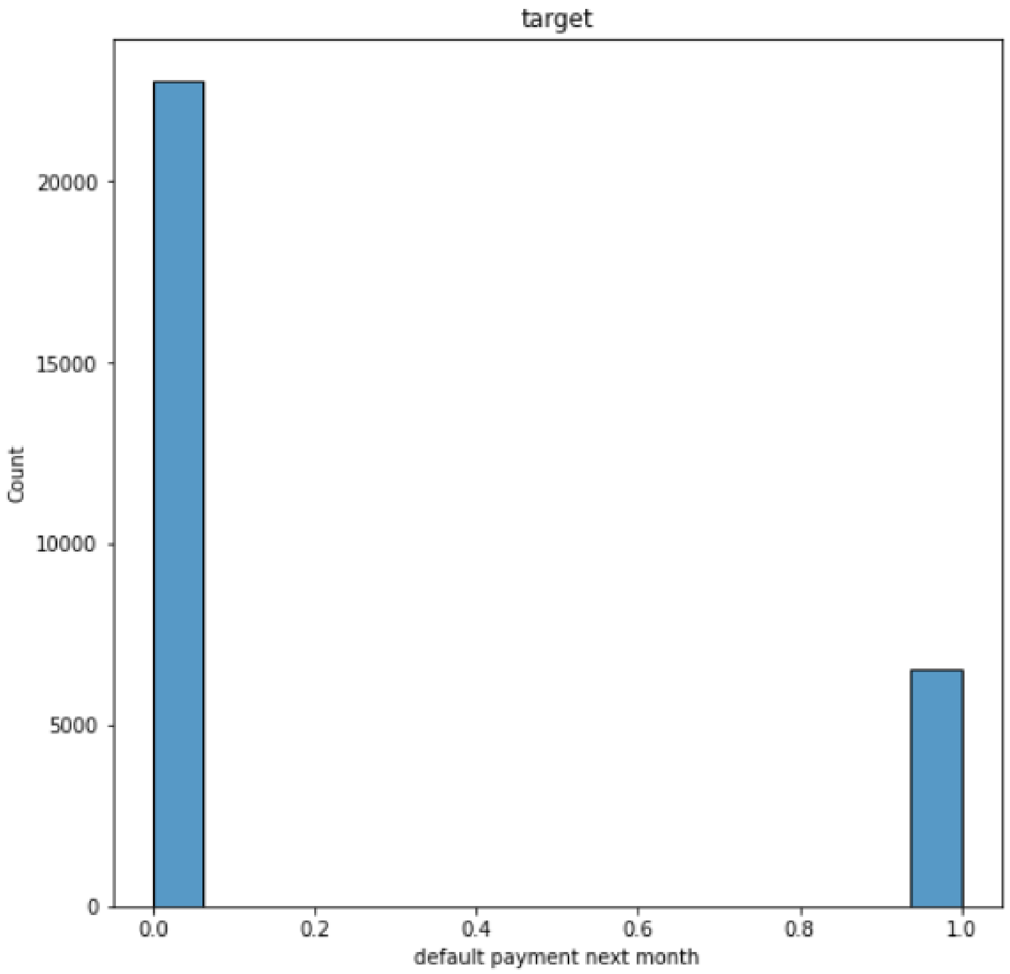
```
<AxesSubplot:ylabel='Duration'>
```



Plotting target data

```
plt.figure(figsize=(8,8))
target=sns.histplot(data_After_pro["default payment next month"])
target.set_title("target")
##
```

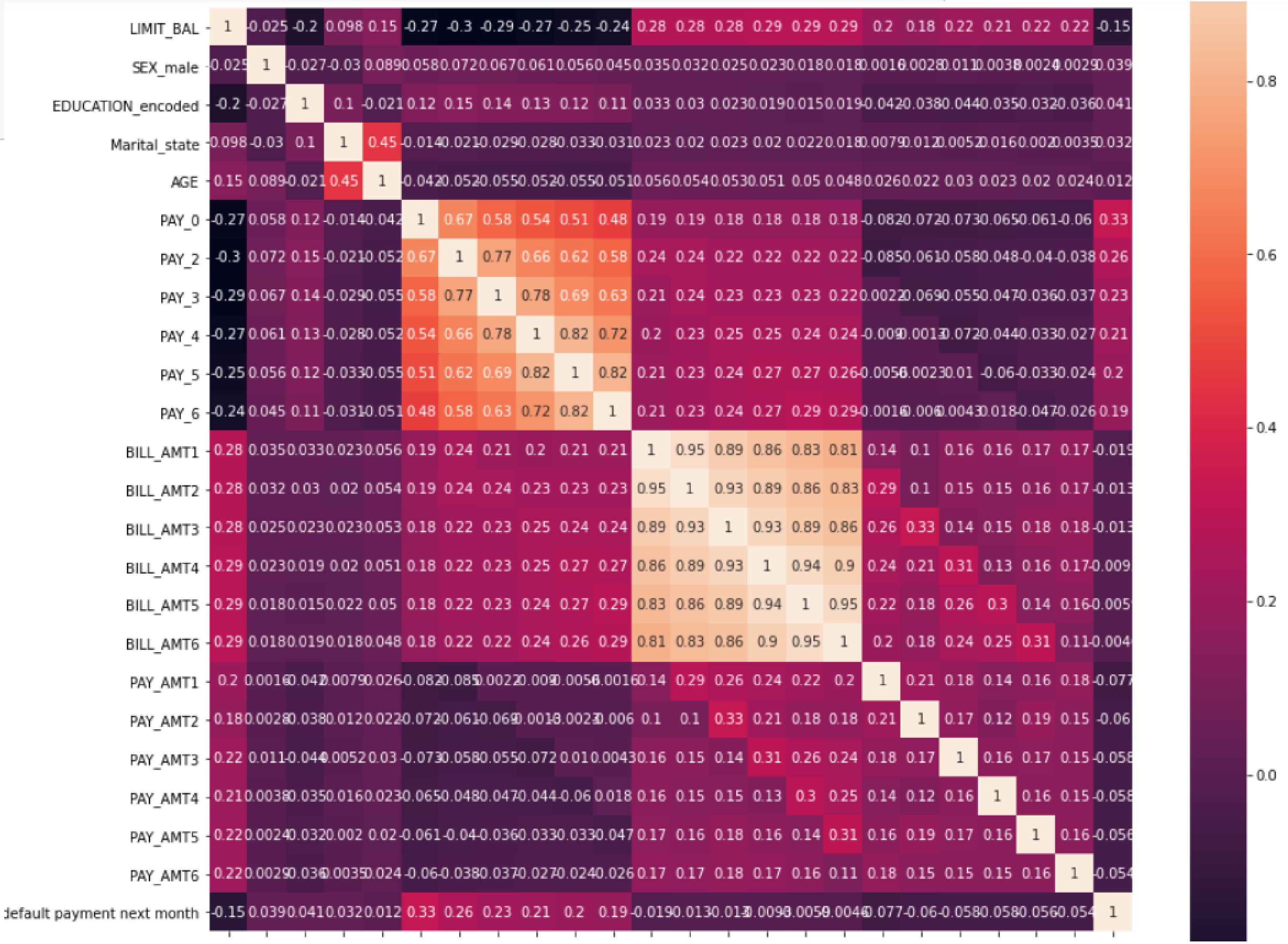
```
Text(0.5, 1.0, 'target')
```



Correlation between features

```python
plt.figure(figsize=(32, 26))
corr = data_After_pro.corr()
mp = sns.heatmap(corr,square=True, annot = True)
mp.set_title(label='dataset correlation', fontsize=20)
```

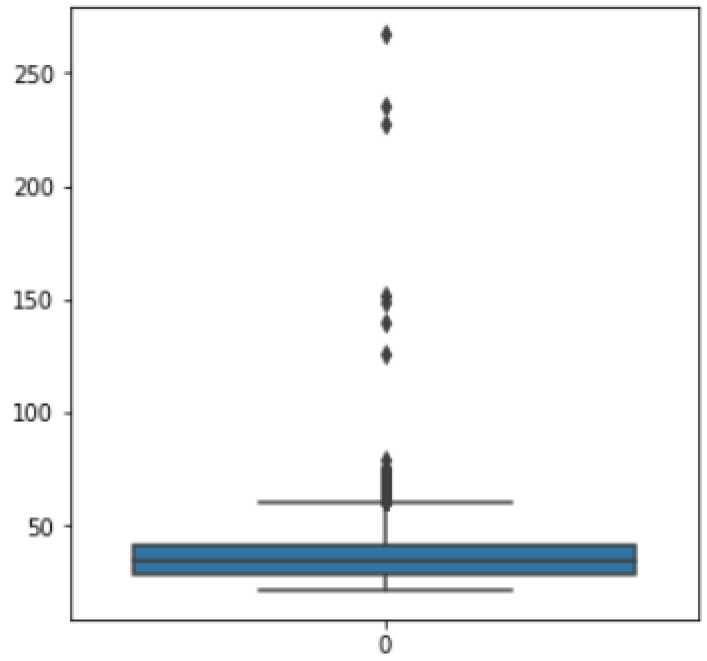
```
Text(0.5, 1.0, 'dataset correlation')
```



Check outliers

```python
plt.figure(figsize=(5,5))
sns.boxplot(data=data_After_pro["AGE"] )
```

```
<AxesSubplot:>
```




```python
data_After_pro["AGE"].unique()
```

```
array([ 24,  26,  34,  37,  57,  29,  23,  28,  35,  51,  41,  30,  49,
        39,  40,  27,  47,  33,  32,  54,  58,  22,  25,  31,  42,  45,
        46,  56,  44,  53,  43,  38,  63,  36,  52,  48,  55,  60,  50,
        75,  61,  73,  59,  21,  67,  62,  66,  70,  72,  64,  65,  71,
       149, 126, 152,  69,  68, 140, 228, 267,  79,  74, 235], dtype=int64)
```

```python
###print(sorted(data_After_pro["AGE"].unique()))
count=0
index_list=[]
for ind,data in  enumerate(data_After_pro["AGE"]) :
    if data > 75:
        index_list.append(ind)
        count+=1

print(count,index_list)
print(data_After_pro["AGE"].loc[28811])
```

```
8 [3933, 4035, 5276, 6800, 7145, 8736, 17845, 28811]
235
```

```python
new_data=data_After_pro.copy()
new_data.drop(index_list,axis=0,inplace=True)
```

```python
new_data["AGE"].unique()
```

```
array([24, 26, 34, 37, 57, 29, 23, 28, 35, 51, 41, 30, 49, 39, 40, 27, 47,
       33, 32, 54, 58, 22, 25, 31, 42, 45, 46, 56, 44, 53, 43, 38, 63, 36,
       52, 48, 55, 60, 50, 75, 61, 73, 59, 21, 67, 62, 66, 70, 72, 64, 65,
       71, 69, 68, 74], dtype=int64)
```

❑ 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	...

8 rows × 21 columns

Numerical data

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

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

Categorical data

Aggregating statistics

I. Numerical data

```
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

```
data_A_filteration["AGE"].mean()
```

35.426695319961794

```
data_A_filteration["AGE"].count()
```

29316

```
data_A_filteration["AGE"].std()
```

9.497365305617228

```
data_A_filteration["AGE"].max()
```

267

II. Categorical data

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

ID	SEX	EDUCATION	MARRIAGE
count	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

Advanced Pandas

Session Objectives

At the this session:

- Pandas Group By operations on real-world data
- split-apply-combine chain of operations works
- concatenate for combining DataFrames across rows or columns
- work with time-series data

❑ Pandas Group By operations on real-world data

Any groupby operation involves one of the following operations on the original object. They are :

1.Splitting the Object
2.Applying a function
3.Combining the results

In many situations, we split the data into sets and we apply some functionality on each subset. In the apply functionality, we can perform the following operations −

1. Aggregation − computing a summary statistic
2. Apply , Transformation − perform some group-specific operation
3. Filtration − discarding the data with some condition

```
#import the pandas library
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

df
```

	Team	Rank	Year	Points
0	Riders	1	2014	876
1	Riders	2	2015	789
2	Devils	2	2014	863
3	Devils	3	2015	673
4	Kings	3	2014	741
5	kings	4	2015	812
6	Kings	1	2016	756
7	Kings	1	2017	788
8	Riders	2	2016	694
9	Royals	4	2014	701
10	Royals	1	2015	804
11	Riders	2	2017	690

1.Splitting

Pandas object can be split into any of their objects. There are multiple ways to split an object like –

obj.groupby('key')
obj.groupby(['key1','key2'])

Key: column name

Let us now see how the grouping objects can be applied to the DataFrame object

```
df.groupby('Team')
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001200444D148>
```

```
list(df.groupby('Team'))
```

```
[('Devils',
     Team  Rank  Year  Points
  2  Devils    2  2014     863
  3  Devils    3  2015     673),
 ('Kings',
     Team  Rank  Year  Points
  4  Kings    3  2014     741
  6  Kings    1  2016     756
  7  Kings    1  2017     788),
 ('Riders',
      Team  Rank  Year  Points
  0   Riders    1  2014     876
  1   Riders    2  2015     789
  8   Riders    2  2016     694
  11  Riders    2  2017     690),
 ('Royals',
      Team  Rank  Year  Points
  9   Royals    4  2014     701
  10  Royals    1  2015     804),
 ('kings',
     Team  Rank  Year  Points
  5  kings    4  2015     812)]
```

groupby.groups : return dict(group name , group labels)

```
df.groupby('Team',).groups
```

```
{'Devils': Int64Index([4, 5], dtype='int64'),
 'Kings': Int64Index([6, 8, 9], dtype='int64'),
 'Riders': Int64Index([2, 3, 10, 13], dtype='int64'),
 'Royals': Int64Index([11, 12], dtype='int64'),
 'kings': Int64Index([7], dtype='int64')}
```

groupby.indices :return dict(group name , group index)

```
df.groupby('Team').indices
```

```
{'Devils': array([2, 3], dtype=int64),
 'Kings': array([4, 6, 7], dtype=int64),
 'Riders': array([ 0,  1,  8, 11], dtype=int64),
 'Royals': array([ 9, 10], dtype=int64),
 'kings': array([5], dtype=int64)}
```

Group by with multiple columns

```
grouped=df.groupby(['Team','Year'])
for column_names,data in grouped:
    print(f"columns={column_names} \n{data}")
```

```
columns=('Devils', 2014)
     Team  Rank  Year  Points
4  Devils     2  2014     863
columns=('Devils', 2015)
     Team  Rank  Year  Points
5  Devils     3  2015     673
columns=('Kings', 2014)
    Team  Rank  Year  Points
6  Kings     3  2014     741
columns=('Kings', 2016)
    Team  Rank  Year  Points
8  Kings     1  2016     756
columns=('Kings', 2017)
    Team  Rank  Year  Points
9  Kings     1  2017     788
columns=('Riders', 2014)
     Team  Rank  Year  Points
2  Riders     1  2014     876
columns=('Riders', 2015)
     Team  Rank  Year  Points
3  Riders     2  2015     789
columns=('Riders', 2016)
      Team  Rank  Year  Points
10  Riders     2  2016     694
columns=('Riders', 2017)
      Team  Rank  Year  Points
13  Riders     2  2017     690
columns=('Royals', 2014)
      Team  Rank  Year  Points
11  Royals     4  2014     701
columns=('Royals', 2015)
      Team  Rank  Year  Points
12  Royals     1  2015     804
columns=('kings', 2015)
    Team  Rank  Year  Points
7  kings     4  2015     812
```

Select a Group

Using the get_group() method, we can select a single group.

```
teams_inf.get_group("Devils")
```

	Team	Rank	Year	Points
2	Devils	2	2014	863
3	Devils	3	2015	673

select certain column after grouping the data

```
list(df.groupby('Team')["Points"])
```

or

```
list(df.groupby('Team').Points)
```

```
list(df.groupby('Team')["Points"])
```

```
[('Devils',
  2    863
  3    673
  Name: Points, dtype: int64),
 ('Kings',
  4    741
  6    756
  7    788
  Name: Points, dtype: int64),
 ('Riders',
  0     876
  1     789
  8     694
  11    690
  Name: Points, dtype: int64),
 ('Royals',
  9     701
  10    804
  Name: Points, dtype: int64),
 ('kings',
  5    812
  Name: Points, dtype: int64)]
```

2- Applying a function
3- Combining the results

Computations / Descriptive Stats	
`GroupBy.count`()	Compute count of group, excluding missing values
`GroupBy.cumcount`([ascending])	Number each item in each group from 0 to the length of that group - 1.
`GroupBy.first`(**kwargs)	Compute first of group values
`GroupBy.head`([n])	Returns first n rows of each group.
`GroupBy.last`(**kwargs)	Compute last of group values
`GroupBy.max`(**kwargs)	Compute max of group values
`GroupBy.mean`(*args, **kwargs)	Compute mean of groups, excluding missing values
`GroupBy.median`(**kwargs)	Compute median of groups, excluding missing values
`GroupBy.min`(**kwargs)	Compute min of group values
`GroupBy.ngroup`([ascending])	Number each group from 0 to the number of groups - 1.
`GroupBy.nth`(n[, dropna])	Take the nth row from each group if n is an int, or a subset of rows if n is a list of ints.
`GroupBy.ohlc`()	Compute sum of values, excluding missing values
`GroupBy.prod`(**kwargs)	Compute prod of group values
`GroupBy.size`()	Compute group sizes
`GroupBy.sem`([ddof])	Compute standard error of the mean of groups, excluding missing values
`GroupBy.std`([ddof])	Compute standard deviation of groups, excluding missing values
`GroupBy.sum`(**kwargs)	Compute sum of group values
`GroupBy.var`([ddof])	Compute variance of groups, excluding missing values
`GroupBy.tail`([n])	Returns last n rows of each group

```python
df.groupby('Team').mean()
```

Team	Rank	Year	Points
Devils	2.500000	2014.500000	768.000000
Kings	1.666667	2015.666667	761.666667
Riders	1.750000	2015.500000	762.250000
Royals	2.500000	2014.500000	752.500000
kings	4.000000	2015.000000	812.000000

```python
df.groupby('Team').sum()
```

Team	Rank	Year	Points
Devils	5	4029	1536
Kings	5	6047	2285
Riders	7	8062	3049
Royals	5	4029	1505
kings	4	2015	812

```python
df.groupby('Team').size()
```

```
Team
Devils     2
Kings      3
Riders     4
Royals     2
kings      1
dtype: int64
```

```python
df.groupby('Team').std()
```

Team	Rank	Year	Points
Devils	0.707107	0.707107	134.350288
Kings	1.154701	1.527525	24.006943
Riders	0.500000	1.290994	88.567771
Royals	2.121320	0.707107	72.831998
kings	NaN	NaN	NaN

```
df.groupby('Team').max()
```

Team	Rank	Year	Points
Devils	3	2015	863
Kings	3	2017	788
Riders	2	2017	876
Royals	4	2015	804
kings	4	2015	812

```
df.groupby('Team')["Points"].max()
```

```
Team
Devils     863
Kings      788
Riders     876
Royals     804
kings      812
Name: Points, dtype: int64
```

```
df.groupby('Team').Points.max()
```

```
Team
Devils     863
Kings      788
Riders     876
Royals     804
kings      812
Name: Points, dtype: int64
```

```
df.groupby('Team').Points.max()
```

```
Team
Devils     863
Kings      788
Riders     876
Royals     804
kings      812
Name: Points, dtype: int64
```

```
df.groupby('Team').Points.get_group("Devils").max()
```

```
863
```

```
1 df.groupby('Team').describe()
```

	Rank								Year					Points	
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean
Team															
Devils	2.0	2.500000	0.707107	2.0	2.25	2.5	2.75	3.0	2.0	2014.500000	...	2014.75	2015.0	2.0	768.000000
Kings	3.0	1.666667	1.154701	1.0	1.00	1.0	2.00	3.0	3.0	2015.666667	...	2016.50	2017.0	3.0	761.666667
Riders	4.0	1.750000	0.500000	1.0	1.75	2.0	2.00	2.0	4.0	2015.500000	...	2016.25	2017.0	4.0	762.250000
Royals	2.0	2.500000	2.121320	1.0	1.75	2.5	3.25	4.0	2.0	2014.500000	...	2014.75	2015.0	2.0	752.500000
kings	1.0	4.000000	NaN	4.0	4.00	4.0	4.00	4.0	1.0	2015.000000	...	2015.00	2015.0	1.0	812.000000

5 rows × 24 columns

Get the main statistics information per group

Aggregations

An aggregated function returns a single aggregated value for each group. Once the group by object is created, several aggregation operations can be performed on the grouped data.

An obvious one is aggregation via the aggregate or equivalent **agg** method:

```
df = pd.DataFrame({'A': 'a a b'.split(), 'B': [1,2,3], 'C': [4,6, 5]})
g = df.groupby('A')
```

```
df
```

	A	B	C
0	a	1	4
1	a	2	6
2	b	3	5

```
list(g)
```

```
[('a',
     A  B  C
  0  a  1  4
  1  a  2  6),
 ('b',
     A  B  C
  2  b  3  5)]
```

```
g.agg(max)
```

A	B	C
a	2	6
b	3	5

```
g.agg(min)
```

A	B	C
a	1	4
b	3	5

```
g.agg([sum,min,max])
```

	B			C		
	sum	min	max	sum	min	max
A						
a	3	1	2	10	4	6
b	3	3	3	5	5	5

2. apply , transform

GroupBy.apply(func, *args*, *kwargs)

GroupBy.transform(func, *args*, *kwargs)

Parameters:	**func** : *function* A callable that takes a dataframe as its first argument, and returns a dataframe, a series or a scalar. In addition the callable may take positional and keyword arguments **args, kwargs** : *tuple and dict* Optional positional and keyword arguments to pass to `func`
Returns:	**applied** : *Series or DataFrame*

3- Filtration

Filtration filters the data on a defined criteria and returns the subset of data. The filter() function is used to filter the data.

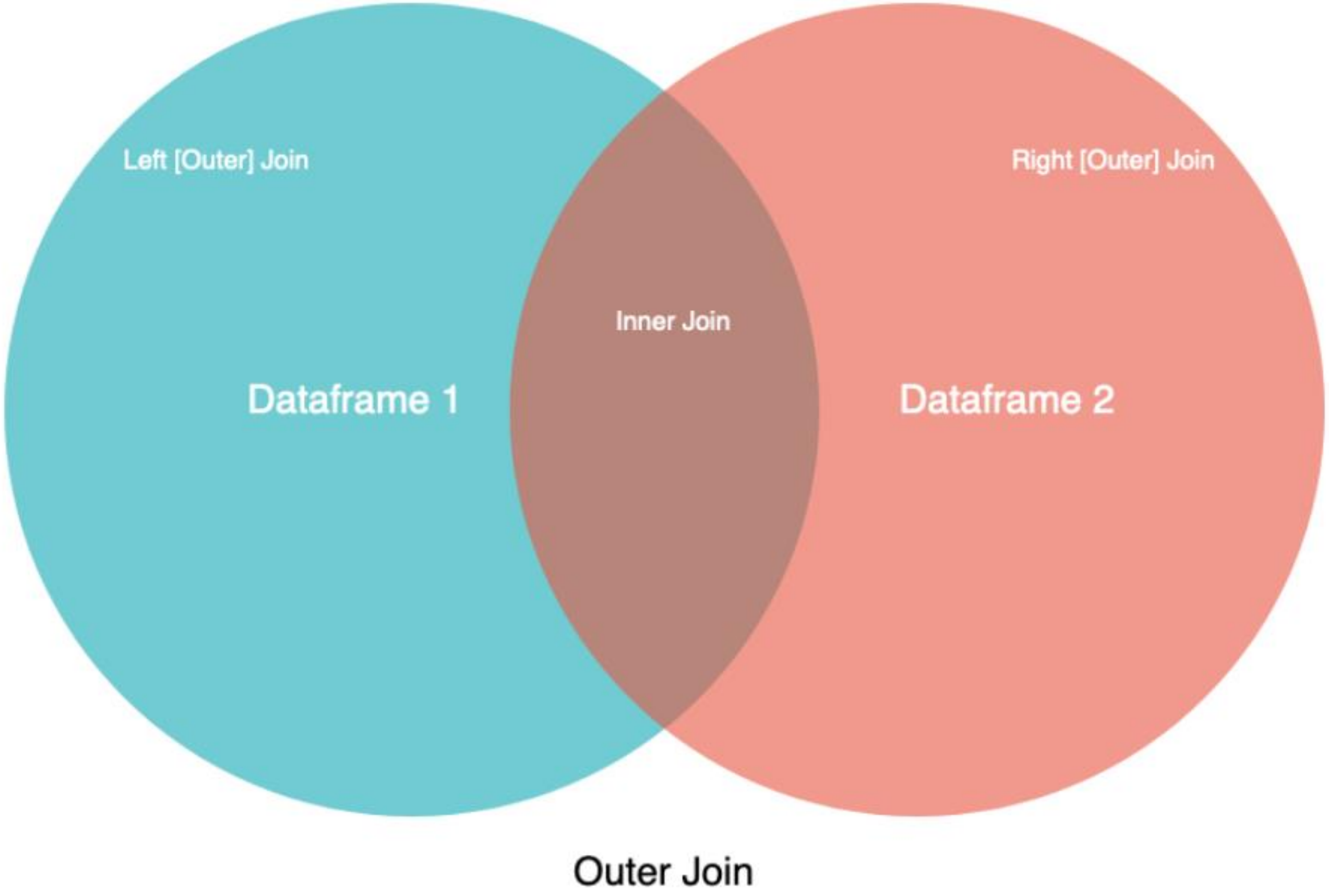
```
import pandas as pd
import numpy as np

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

print(df.groupby('Team').filter(lambda x: len(x) >= 3))
```

```
      Team  Rank  Year  Points
0   Riders     1  2014     876
1   Riders     2  2015     789
4    Kings     3  2014     741
6    Kings     1  2016     756
7    Kings     1  2017     788
8   Riders     2  2016     694
11  Riders     2  2017     690
```

❑ concatenate for combining DataFrames across rows or columns

Concat :

One way to combine or concatenate DataFrames is concat() function. It can be used to concatenate DataFrames along rows or columns by changing the axis parameter. The default value of the axis parameter is 0, which indicates combining along rows.



The default value of the axis parameter is 0, which indicates combining along rows.

```
df1=pd.read_csv("df1.csv",index_col=0)
df1
```

	column_a	column_b	column_c
0	1	a	True
1	2	b	True
2	3	c	False
3	4	d	True

```
df2=pd.read_csv("df2.csv",index_col=0)
df2
```

	column_a	column_b	column_c
0	1	a	False
1	2	k	False
2	9	r	False
3	10	Q	True

```
## concatenate df1 & df2 on axis=0 or across rows
conct_axis0= pd.concat([df1,df2])
conct_axis0
```

	column_a	column_b	column_c
0	1	a	True
1	2	b	True
2	3	c	False
3	4	d	True
0	1	a	False
1	2	k	False
2	9	r	False
3	10	Q	True

The indices of individual DataFrames are kept.
In order to change it and re-index the combined DataFrame,

- ignore_index parameter is set as True

```
conct_axis0_reset_index= pd.concat([df1,df2],ignore_index=True)
conct_axis0_reset_index
```

	column_a	column_b	column_c
0	1	a	True
1	2	b	True
2	3	c	False
3	4	d	True
4	1	a	False
5	2	k	False
6	9	r	False
7	10	Q	True

```
## concatenate df1 & df2 on axis=1 or across columns
conct_axis1= pd.concat([df1,df2],axis=1)
conct_axis1
```

	column_a	column_b	column_c	column_a	column_b	column_c
0	1	a	True	1	a	False
1	2	b	True	2	k	False
2	3	c	False	9	r	False
3	4	d	True	10	Q	True

concatenate dataframes across column by setting (axis=1):

join parameter of concat()

function determines how to combine DataFrames. The default value is ‘outer’ returns all indices in both DataFrames. If ‘inner’ option is selected, only the rows with shared indices are returned. I will change the index of df2 so that you can see the difference between ‘inner’ and ‘outer’.

```
df1
```

	column_a	column_b	column_c
0	1	a	True
1	2	b	True
2	3	c	False
3	4	d	True

```
df2.index=[2,3,4,5]
df2
```

	column_a	column_b	column_c
2	1	a	False
3	2	k	False
4	9	r	False
5	10	Q	True

1. inner join

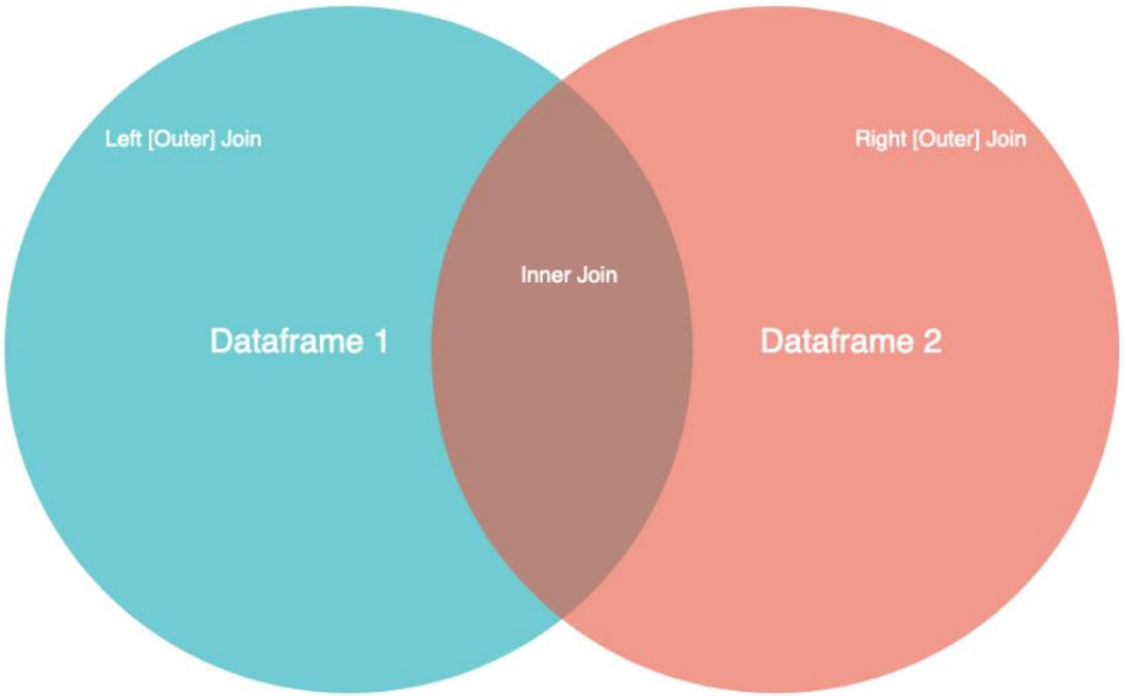
```
df1_df2_innerjoin=pd.concat([df1,df2],axis=1,join="inner")
df1_df2_innerjoin
```

	column_a	column_b	column_c	column_a	column_b	column_c
2	3	c	False	1	a	False
3	4	d	True	2	k	False

2. Outer join

```
df1_df2_OUTERjoin=pd.concat([df1,df2],axis=1,join="outer")
df1_df2_OUTERjoin
```

	column_a	column_b	column_c	column_a	column_b	column_c
0	1.0	a	True	NaN	NaN	NaN
1	2.0	b	True	NaN	NaN	NaN
2	3.0	c	False	1.0	a	False
3	4.0	d	True	2.0	k	False
4	NaN	NaN	NaN	9.0	r	False
5	NaN	NaN	NaN	10.0	Q	True



Outer Join

Pandas also provides ways to label DataFrames so that we know which part comes from which DataFrame. We just pass the list of combined DataFrames in order using **keys** parameter

```
df1_df2_keys=pd.concat([df1,df2],keys=["df1","df2"])
df1_df2_keys
```

		column_a	column_b	column_c
df1	0	1	a	True
	1	2	b	True
	2	3	c	False
	3	4	d	True
df2	2	1	a	False
	3	2	k	False
	4	9	r	False
	5	10	Q	True

```
df1_df2_keys.loc["df1"]
```

	column_a	column_b	column_c
0	1	a	True
1	2	b	True
2	3	c	False
3	4	d	True

Another widely used function to combine DataFrames is merge(). Concat() function simply adds DataFrames on top of each other or adds them side-by-side. It is more like appending DataFrames. Merge() combines DataFrames based on values in shared columns. Merge() function offers more flexibility compared to concat() function.

Check jupyter notebook for merge examples

❑ work with time-series data

Pandas was developed in the context of financial modeling, so as you might expect, it contains a fairly extensive set of tools for working with dates, times, and time-indexed data. Date and time data comes in a few flavors, which we will discuss here:

- *Time stamps* reference particular moments in time (e.g., July 4th, 2015 at 7:00am).
- *Time intervals* and *periods* reference a length of time between a particular beginning and end point; for example, the year 2015. Periods usually reference a special case of time intervals in which each interval is of uniform length and does not overlap (e.g., 24 hour-long periods comprising days).
- *Time deltas* or *durations* reference an exact length of time (e.g., a duration of 22.56 seconds).

Convert our date columns to datetime

Now that we know that our date column isn't being recognized as a date, it's time to convert it so that it is recognized as a date. This is called "parsing dates" because we're taking in a string and identifying its component parts.

We can pandas what the format of our dates are with a guide called as "strftime directive", which you can find more information on at this link. The basic idea is that you need to point out which parts of the date are where and what punctuation is between them. There are lots of possible parts of a date, but the most common are %d for day, %m for month, %y for a two-digit year and %Y for a four digit year.

Some examples:

> 1/17/07 has the format "%m/%d/%y"
> 17-1-2007 has the format "%d-%m-%Y“

Looking back up at the head of the "date" column in the landslides dataset, we can see that it's in the format "month/day/two-digit year"