DataScientest • com

Pandas for Data Science

Introduction to DataFrames

Introduction

The `pandas` module has been developed to provide `Python` with the tools necessary to manipulate and analyze large volumes of data.

`Pandas` introduces the **`DataFrame`** class, an array-like data structure that offers more advanced data manipulation and exploration than `NumPy` arrays.

The main features of `pandas` are:

- data recovery from files (CSV, Excel tables, etc.)
- handling this data (deletion / addition, modification, statistical visualization, etc.).

This notebook aims at:

- Understanding the format of a `DataFrame`.
- Creating a first `Dataframe`.
- Carrying out a first exploration of a dataset using the `DataFrame` class.

- **(a)** Import the `pandas` module under the name `pd`.

In [1]:

```python
# Insert your code here
import pandas as pd
```

Show solution

1. Format of a DataFrame

A DataFran
an **index**. Typ
identifiers.

A DataFran
database (inc
features are

•

•

The column c
not managed

The index ca
one (or sever
that we speci

Example: Def
anything :

Example: Ind

Example: Ind
'person_3

We will detai

The `DataFrame` class has several advantages over a `Numpy` array:

- Visually, a `DataFrame` is much more **readable** thanks to more explicit column and row indexing.
- Within the same column the elements are of the same type but from one column to another, the **type of the elements may vary**, which is not the case of `Numpy` arrays which only support data of the same type.
- The `DataFrame` class contains more methods for handling and preprocessing databases, while `NumPy` specializes instead in optimized computation.

2. Creation of a DataFrame: from a NumPy array

It is possible t
`DataFrame`
very practica

Let's take a c

```
pd.Dat
```

For othe
(https://pa
/api/panda

Example:

```
# Crea
umns
array


# Inst
df = p
rmat

# The

']) #
```

This produce

3. Creation of

In [3]:

```python
# Insert your code here

dic = {'name' : ['honey', 'flour', 'wine'],
       'date' : ['08/10/2025','09/25/2024', '10/15/2023'],
       'quantity' : [100, 55, 1800],
       'price' : [2, 3, 10]}

df = pd.DataFrame(dic)

df.head()
```

Out[3]:

	name	date	quantity	price
0	honey	08/10/2025	100	2
1	flour	09/25/2024	55	3
2	wine	10/15/2023	1800	10

Hide solution

In [ ]:

```python
dictionary = {"Product"          : ['honey', 'flour', 'wine'],
              "Expiration date"  : ['10/08/2025', '25/09/2024', '15/
              "Quantity"         : [100, 55, 1800],
              "Price per unit"   : [2, 3, 10]}

df = pd.DataFrame(dictionary)

print(df)
```

4. Creation of a DataFrame: from a data file

Most often a
of interest. T

The most cor
Separated Val
separated by

Here is an ex

```
A, B,
1, 2,
5, 6,
9, 10,
```

In this forma

To import the
function of p

```
pd.rea
= 0, i
```

The **essentia**

In [8]:

```
# Insert your code here

df = pd.read_csv('transactions.csv', sep=',', header=0, index_col=0)
df
```

Out[8]:

	cust_id	tran_date	prod_subcat_code	prod_cat_code	Qty	Rate	Tax	total_a
transaction_id								
80712190438	270351	28-02-2014	1.0	1	-5	-772.0	405.300	-4265.3
29258453508	270384	27-02-2014	5.0	3	-5	-1497.0	785.925	-8270.9
51750724947	273420	24-02-2014	6.0	5	-2	-791.0	166.110	-1748.1
93274880719	271509	24-02-2014	11.0	6	-3	-1363.0	429.345	-4518.3
51750724947	273420	23-02-2014	6.0	5	-2	-791.0	166.110	-1748.1
...	...	...	...	...	...	...	...	...
94340757522	274550	25-01-2011	12.0	5	1	1264.0	132.720	1396.7
89780862956	270022	25-01-2011	4.0	1	1	677.0	71.085	748.0
85115299378	271020	25-01-2011	2.0	6	4	1052.0	441.840	4649.8
72870271171	270911	25-01-2011	11.0	5	3	1142.0	359.730	3785.7
77960931771	271961	25-01-2011	11.0	5	1	447.0	46.935	493.9

23053 rows × 9 columns

Hide solution

In [10]:

```
# You can directly specify the name of the column containing the ind
transactions = pd.read_csv(filepath_or_buffer = 'transactions.csv',
                           sep = ',',
                           header = 0,
                           index_col = 'transaction_id')

# You can also directly enter the number of the column that indexes
transactions = pd.read_csv(filepath_or_buffer = 'transactions.csv',
                           sep = ',',
                           header = 0,
                           index_col = 0) # number of the column tha
```

We loaded the `transactions.csv` file in the `DataFrame` **`transactions`** which gathers a history of transactions carried out between 2011 and 2014. In the next section, we will study this dataset.

5. First exploration of a dataset using the `DataFrame` class

The rest of th
`DataFrame`
that is:

-
-
-

As a ren
DataFram
object. Exa

6. Visualizati `columns` an

- It is poss
lines of t

For that, we r
number of lin

It is also poss
is applied in t

```
# Disp
my_dat
```

In [50]:

```
# Insert your co
transactions.hea
```

Out[50]:

	cust_id
transaction_id	
80712190438	270351
29258453508	270384

Show solutio

- **(b)** Display the la

In [51]:

```
# Insert your code here
transactions.tail(1)
```

Out[51]:

	cust_id	tran_date	prod_subcat_code	prod_cat_code	Qty	Rate	Tax	total_amt
transaction_id								
77960931771	271961	25-01-2011	11.0	5	1	447.0	46.935	493.935

Hide solution

In [ ]:

```
transactions.tail(10)
```

It is possible to retrieve the **name of the columns** of a `DataFrame` thanks to its **`columns`** attribute.

```
# Creation of a DataFrame df from a dictionary
dictionary = {'A': [1, 5, 9],
              'B': [2, 6, 10],
              'C': [3, 7, 11],
              'D': [4, 8, 12]}

df = pd.DataFrame (data = dictionary, index = ['i
_1', 'i_2', 'i_3'])
```

These instructions produce the same `DataFrame` as before:

	A	B	C	D
i_1	1	2	3	4
i_2	5	6	7	8
i_3	9	10	11	12

```
# Display of df DataFrame columns
print(df.columns)
>>> ['A', 'B', 'C', 'D']
```

The list of the column names can be used to iterate over the columns of a `DataFrame` within a loop.

It can be interesting to know how many transactions (rows) and how many features (columns) the dataset contains.

For this we will use the **`shape`** attribute of the `DataFrame` class which displays the **dimensions** of our `DataFrame` in the form of a tuple (number of rows, number of columns):

```
# Display the dimensions of df
print (df.shape)
>>> (3,4)
```

- **(c)** Display the **di**
  the 5th column

In [15]:

```
# Insert your co
print(transactio

transactions.co
```

(23053, 9)

Out[15]: 'Qty'

Hide solution

In [16]:

```
print(transactio
print(transactio
```

```
(23053, 9)
Qty
```

7. Selecting c

Extracting columns from a `DataFrame` is almost identical to extracting data from a dictionary.

To extract a **column** from a `DataFrame` , all we have to do is enter **between brackets** the **name** of the column to extract. To extract **several** columns, we must enter between brackets **the list of the names** of the columns to extract:

```
# Display of the 'cust_id' column
print(transactions['cust_id'])

# Extraction of the 'cust_id' and 'Qty' columns from
transactions
cust_id_qty = transactions[["cust_id", "Qty"]]
```

`cust_id_qty` is a new **`DataFrame`** containing only the `'cust_id'` and `'Qty'` columns.

The display of the first 3 lines of **`cust_id_qty`** yields:

	cust_id	Qty
transactions_id		
80712190438	270351	-5
29258453508	270384	-5
51750724947	273420	-2

In [23]:

```
### Insert your

cat_vars = trans
num_vars = trans

print(cat_vars.h
print(30*'-')
print(num_vars.h
```

```
                  c
\
transaction_id
80712190438
29258453508
51750724947
93274880719
51750724947

                 St
transaction_id
80712190438
29258453508
51750724947
93274880719
51750724947
------------------
                  Q
transaction_id
80712190438
29258453508
51750724947
93274880719
51750724947
```

Hide solution

In [24]:

```
# Extraction of categorical variables
cat_var_names = ['cust_id', 'tran_date', 'prod_subcat_code', 'prod_ca
cat_vars = transactions[cat_var_names]

# Extraction of quantitative variables
num_var_names = ['Qty', 'Rate', 'Tax', 'total_amt']
num_vars = transactions[num_var_names]

# Display of the first 5 lines of each DataFrame
print ("Categorical variables: \n")
print (cat_vars.head(), "\n \n")

print ("Quantitative variables: \n")
print (num_vars.head())
```

```
Categorical variables: 

                cust_id   tran_date  prod_subcat_code  prod_cat_code  
\
transaction_id                                                         
80712190438      270351  28-02-2014               1.0              1   
29258453508      270384  27-02-2014               5.0              3   
51750724947      273420  24-02-2014               6.0              5   
93274880719      271509  24-02-2014              11.0              6   
51750724947      273420  23-02-2014               6.0              5   

               Store_type  
transaction_id             
80712190438         e-Shop  
29258453508         e-Shop  
51750724947       TeleShop  
93274880719         e-Shop  
51750724947       TeleShop   


Quantitative variables: 

                Qty    Rate      Tax  total_amt
transaction_id                                 
80712190438      -5  -772.0  405.300  -4265.300
29258453508      -5 -1497.0  785.925  -8270.925
51750724947      -2  -791.0  166.110  -1748.110
93274880719      -3 -1363.0  429.345  -4518.345
51750724947      -2  -791.0  166.110  -1748.110
```

8. Selecting rows of a **`DataFrame`**: **`loc`** and **`iloc`** methods

To extract on
`loc` is a ver
between squ
very similar t

In order to re
enter `i` as a

```
# We r
e num_
print(

>>
>> tra
>> 807
>> 807
```

In order to re

-
-

```
# We r
258453
ataFra
transa
724947
```

`loc` can als
refine the da

```
# We e
om the
transa
x', 't
```

This instructi

The **`iloc`** n
array, that is
columns. This

```python
# Extraction of the first 4 rows and the first 3
columns of transactions
transactions.iloc[0:4, 0:3]
```

This instruction produces the following `DataFrame` :

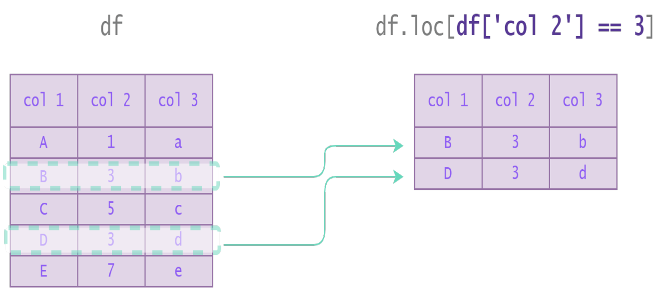
transaction_id	cust_id	tran_date	prod_subcat_code
80712190438	270351	28-02-2014	1.0
29258453508	270384	27-02-2014	5.0
51750724947	273420	24-02-2014	6.0
93274880719	271509	24-02-2014	11.0

If the row indexing is the one by default (row numbering), the `loc` and `iloc` methods are **equivalent**.

9. Conditional indexing of a `DataFrame`

As with Numpy arrays, we can use **conditional indexing** to extract rows from a `Dataframe` that meet a given condition.

In the following illustration, we select the rows of the `DataFrame` `df` **for which the column `col 2` is equal to 3.**



There are two syntaxes for conditionally indexing a `DataFrame` :

```python
# We select the rows of the DataFrame df for whic
h the column 'col 2' is equal to 3.
df[df['col 2'] == 3]

df.loc[df['col 2'] == 3]
```

If we want to **assign a new value** to these entries, we must absolutely use the **`loc`** method.

Indeed, indexing with the syntax `df[df['col 2'] == 3]` only returns a **copy** of these entries and does not provide access the memory location where the data is located.

The manager of the tr
access to the **identifie**
Shop") as well as **the**

In [26]:

```python
# Insert your co

transactions_esh

transactions_id_

transactions_id_
```

Out[26]:

transaction_id	cust_id
80712190438	270351
29258453508	270384
93274880719	271509
45649838090	273667
50076728598	269014

Hide solution

In [ ]:

```python
# Creation of tr
transactions_esh

# Extraction of
transactions_id_

# Display of the
transactions_id_
```

Now, the manager wo
whose identifier is 2

- **(d)** In a `DataFra`
with client identi

- **(e)** A column in a
`value in df[`
compute and dis

In [30]:

```
# Insert your code here

transactions_client_268819 = transactions.loc[transactions.cust_id ==

# First Way
total = 0
for i in transactions_client_268819['total_amt']:
    total += i
print(total)

# Second Way
transactions.loc[transactions.cust_id == 268819]['total_amt'].sum()
```

```
14911.974999999999
```

Out[30]: 14911.974999999999

Hide solution

In [29]:

```
# Extraction of the transactions ofthe customer which identifier is
transactions_client_268819 = transactions[transactions['cust_id'] ==


# Computation of the total amount of transactions
total = 0

# For each amount in the column 'total_amt'
for amount in transactions_client_268819['total_amt']:
    # We sum the amounts
    total += amount

print(total)
```

```
14911.974999999999
```

10. Quick statistical study of the data in a `DataFrame`.

> The **`describe`** method of a `DataFrame` returns a summary of the **descriptive statistics** (min, max, mean, quantiles,...) of its **quantitative** variables. It is therefore a very useful tool for a first visualisation of the type and distribution of these variables.
>
> To analyse the **categorical** variables, it is recommended to start by using the **`value_counts`** method which returns the **number of occurrences** for each modality of these variables. The `value_counts` method cannot be used directly on a `DataFrame` but only on the columns of the `DataFrame` which are objects of the **`pd.Series`** class.

- **(a)** Use the `describe` method of the `DataFrame` `transactions`.
- **(b)** The quantitative variables of `transactions` are `'Qty'`, `'Rate'`, `'Tax'` and `total_amt'`. By default, are the statistics produced by the `describe` method **only** computed on the quantitative variables?
- **(c)** Display the number of occurrences of each modality of the `Store_type` column

using the value

In [32]:

```
# Insert your co

# a
transactions.des

# b
# No some catoge
# categorical va
```

Out[32]:

	cust_id
count	23053.000000
mean	271022.241661
std	2430.830508
min	266783.000000
25%	268936.000000
50%	270981.000000
75%	273114.000000
max	275265.000000

In [33]:

```
# c)
transactions.Sto
```

Out[33]:

```
e-Shop
MBR
Flagship store
TeleShop
Name: Store_type,
```

Hide solution

In [ ]:

```
transactions.des

transactions['St
```

The `describe` method computed statistics on the variables `cust_id`, `prod_subcat_code` and `prod_cat_code` while these are **categorical** variables.

Of course, these statistics make **no sense**. The `describe` method has treated these variables as quantitative because the modalities they take are

In [37]:

```
# Insert your code here

print('average total amount spent :', transactions['total_amt'].mean
print('maximum quantity purchased :', transactions['Qty'].max())

transactions.describe()
```

```
average total amount spent : 2108.007266944553
maximum quantity purchased : 5
```

Out[37]:

	cust_id	prod_subcat_code	prod_cat_code	Qty	Rate	Ta:
count	23053.000000	23021.000000	23053.000000	23053.000000	23031.000000	23031.00000(
mean	271022.241661	6.150298	3.762721	2.434173	636.405019	248.66552(
std	2430.830508	3.726557	1.677314	2.265703	622.053592	187.08770!
min	266783.000000	1.000000	1.000000	-5.000000	-1499.000000	7.35000(
25%	268936.000000	3.000000	2.000000	1.000000	312.000000	98.17500(
50%	270981.000000	5.000000	4.000000	3.000000	710.000000	199.08000(
75%	273114.000000	10.000000	5.000000	4.000000	1109.000000	365.82000(
max	275265.000000	12.000000	6.000000	5.000000	1500.000000	787.50000(

Hide solution

In [ ]:

```
# Applying the describe method to the transactions DataFrame
transactions.describe()

# The average total amount spent is €2109.
# The maximum quantity purchased is 5.
```

Some transactions have **negative** amounts.

These are transactions that have been cancelled and refunded to the client. These amounts will disrupt the distribution of the amounts which gives us **bad estimates** of the mean and quantiles of the variable `total_amt`.

- **(f)** What is the average amount of transactions with **positive** amounts?

In [48]:

```
# Insert your co

print('Average a
      transactio
```

```
Average amount of
6676
```

Hide solution

In [45]:

```
transactions[tra

# the average am
# €500 more than
```

Out[45]:

	cust_id
count	20861.000000
mean	271027.632760
std	2432.810074
min	266783.000000
25%	268938.000000
50%	271009.000000
75%	273122.000000
max	275265.000000

Conclusion a

The `DataFrame` class of the `pandas` module will be your favorite data structure when exploring, analysing and processing datasets and databases.

In this brief introduction, you have learned to:

> - Create a `DataFrame` from a `numpy` array and a dictionary using the **`pd.DataFrame`** constructor.
>
> - Create a `DataFrame` from a `.csv` file using the **`pd.read_csv`** function.

Validate