None

We use "DataFrame.value_counts" to count the frequency of each unique row in the DataFrame.

df:

Manager	Product
Debra	CPU
Debra	CPU
Fred	Monitor
John	Software
Fred	Monitor

→

Output:

```
Manager  Product
Debra    CPU       2
Fred     Monitor   2
John     Software  1
```

Contents

What Makes This Ebook Unique

Tips for Effective Use of This Ebook

Introduction

subset

normalize

sort

ascending

dropna

Contacts and Social Media

Sources & References

Thank You!

What Makes This Ebook Unique

> *"The aim of this ebook is to give you the 'aha' moment right away at the start of learning a new concept."*

- Practical Step By Step Guide With Simple Examples
- Visual Illustrations and Interactive
- Simple Datasets
- Comprehensive Coverage (pandas Documentation used as reference)

Tips for Effective Use of This Ebook

- Use Examples First: If you don't understand the text, go straight to the examples—they're self-explanatory.
- After reviewing them, return to the text to grasp the practical concepts.
- Practice with Datasets: Copy the datasets provided and practice using them to reinforce your understanding.

- Click on the blue links to go directly to the section you want to learn about.

Intoduction

We use "DataFrame.value_counts" to count the frequency of each unique row in the DataFrame.

df:

Manager	Product
Debra	CPU
Debra	CPU
Fred	Monitor
John	Software
Fred	Monitor

→

Output:

```
Manager  Product
Debra    CPU       2
Fred     Monitor   2
John     Software  1
```

Code Snippet

```
data = {
    'Manager': ['Debra', 'Debra', 'Fred', 'John', 'Fred'],
    'Product': ['CPU', 'CPU', 'Monitor', 'Software', 'Monitor']
}
df = pd.DataFrame(data)
```

→

df:

Manager	Product
Debra	CPU
Debra	CPU
Fred	Monitor
John	Software
Fred	Monitor

`subset`

Accepts a **label** or a **list of labels**.

It specifies the columns to consider when counting unique combinations.

How many times does each product appear in the DataFrame, regardless of the manager?

The `subset` parameter with a single column like `'Product'` returns the count of each product, regardless of manager.

df:

	Manager	Product
0	Debra	CPU
1	Debra	CPU
2	John	Monitor
3	John	Software
4	Fred	Monitor

→

Output:

```
       Product
CPU          2
Monitor      2
Software     1
Name: count, dtype: int64
```

Code:

```
df.value_counts(
    subset=['Product']
    )
```

Code Snippet

```
data = {
    'Manager': ['Debra', 'Debra', 'John', 'John', 'Fred'],
    'Product': ['CPU', 'CPU', 'Monitor', 'Software', 'Monitor']
```

df:

	Manager	Product
0	Debra	CPU

```
}
df = pd.DataFrame(data)
```

	Manager	Product
1	Debra	CPU
2	John	Monitor
3	John	Software

How many times does each unique combination of product and price appear in the DataFrame, disregarding the manager?

When `subset` accepts a list of column labels (like `['Product', 'Price']`), it returns counts of unique combinations of those columns, regardless of manager.

df:

	Manager	Product	Price
0	Debra	Monitor	300
1	Debra	CPU	600
2	Fred	Monitor	300
3	John	Monitor	480
4	Fred	Monitor	300

Output:

```
       Product  Price
Monitor  300      3
CPU      600      1
Monitor  480      1
Name: count, dtype: int64
```

Code:

```
df.value_counts(
    ['Product', 'Price']
)
```

Code Snippet

```
data = {
    'Manager': ['Debra', 'Debra', 'Fred', 'John', 'Fred'],
    'Product': ['Monitor', 'CPU', 'Monitor', 'Monitor', 'Monitor'],
    'Price': [300, 600, 300, 480, 300]
}
df = pd.DataFrame(data)
```

df:

	Manager	Product	Price
0	Debra	Monitor	300
1	Debra	CPU	600
2	Fred	Monitor	300

	Manager	Product	Price
3	John	Monitor	480

How many times does each combination of 'Manager' and 'Product' occur in the DataFrame?

Without specifying a subset, `value_counts()` counts unique combinations of all columns in the DataFrame.

df:

	Manager	Product
0	Debra	CPU
1	Debra	CPU
2	Fred	Monitor
3	John	Software
4	Fred	Monitor

→

Output:

```
       Manager  Product
Debra      CPU        2
Fred     Monitor      2
John     Software     1
Name: count, dtype: int64
```

Code:

```
df.value_counts()
```

Code Snippet

```
data = {
    'Manager': ['Debra', 'Debra', 'Fred', 'John', 'Fred'],
    'Product': ['CPU', 'CPU', 'Monitor', 'Software', 'Monitor']
}
df = pd.DataFrame(data)
```

→

df:

	Manager	Product
0	Debra	CPU
1	Debra	CPU
2	Fred	Monitor
3	John	Software
4	Fred	Monitor

normalize

The **"normalize"** parameter returns proportions rather than frequencies. It accepts a **Boolean** value and defaults to **False**.

Count and Proportion of Unique Combinations of Manager and Product

(normalize=False):

Shows how often each unique row appears.

df:

	Manager	Product
0	Debra	CPU
1	Debra	CPU
2	Fred	Monitor
3	John	Software
4	Fred	Monitor

Output (normalize=False):

```
          Manager  Product
Debra     CPU        2
Fred      Monitor    2
John      Software   1
Name: count, dtype: int64
```

Code (normalize=False):

```
df.value_counts(
    normalize=False
    )
```

(normalize=True):

Shows the proportion (percentage) of each unique row.

df::

	Manager	Product
0	Debra	CPU
1	Debra	CPU
2	Fred	Monitor
3	John	Software

Output (normalize=True):

```
          Manager  Product
Debra     CPU        0.4
Fred      Monitor    0.4
John      Software   0.2
Name: proportion, dtype:
```

Code (normalize=True):

```
df.value_counts(
    normalize=True
    )
```

Code Snippet

```
data = {
    'Manager': ['Debra', 'Debra', 'Fred', 'John', 'Fred'],
    'Product': ['CPU', 'CPU', 'Monitor', 'Software', 'Monitor']
}
df = pd.DataFrame(data)
```

→

df:

	Manager	Product
0	Debra	CPU
1	Debra	CPU
2	Fred	Monitor
3	John	Software
4	Fred	Monitor

`sort`

The **"sort"** parameter accepts a **Boolean** value and defaults to **True**.

It sorts by frequencies when **True**.

It sorts by DataFrame column values when **False**.

value_counts(sort=True vs False) Behavior

When sort=True (default):

It sorts by frequencies (most frequent first).

When sort=False:

Sorts by DataFrame column values.

df:

	fruits
0	apple
1	banana
2	apple
3	orange
4	banana
5	banana

df::

	fruits
0	apple
1	banana
2	apple
3	orange
4	banana
5	banana

Output (sort=True):

```
          fruits
banana     3
apple      2
orange     1
Name: count, dtype:
int64
```

Code (sort=True):

```
df.value_counts(
    sort=True
    )
# or simply:
df.value_counts()
```

Output (sort=False):

```
          fruits
apple      2
banana     3
orange     1
Name: count, dtype:
int64
```

Code (sort=False):

```
df.value_counts(
    sort=False
    )
```


Code Snippet

```
df = pd.DataFrame({
 'fruits': ['apple', 'banana', 'apple', 'orange', 'banana', 'banana']
})
```

df:

	fruits
0	apple
1	banana
2	apple

	fruits
3	orange
4	banana
5	banana

ascending

The **"ascending"** parameter accepts a **Boolean** value and defaults to **False**. It controls the order in which these counts are sorted.

value_counts(ascending=False vs True) Behavior

When ascending=False (default):
It sorts from largest to smallest.

When ascending=True:
It sorts from smallest to largest.

df:

	fruits
0	apple
1	banana
2	apple
3	orange
4	banana
5	banana

df::

Output (ascending=False):

```
        fruits
banana       3
apple        2
orange       1
Name: count, dtype:
int64
```

Code (ascending=False):

```
df.value_counts(
    ascending=False
    )
# or simply:
df.value_counts()
```

Code (ascending=True):

	fruits
0	apple
1	banana
2	apple
3	orange
4	banana
5	banana

Output (ascending=True):

```
          fruits
orange     1
apple      2
banana     3
Name: count, dtype:
int64
```

```
df.value_counts(
    ascending=True
    )
```

Code Snippet

```
df = pd.DataFrame({
 'fruits': ['apple', 'banana', 'apple', 'orange', 'banana', 'banana']
})
```

→

df:

	fruits
0	apple
1	banana
2	apple
3	orange
4	banana
5	banana

dropna

The **"dropna"** parameter accepts a **Boolean** value and defaults to **True**. It doesn't count rows that have missing values.

value_counts(dropna=True vs False) Behavior

When dropna=True (Default Behavior):
Rows that contain NA values are not counted.

When dropna=False:
Rows that contain NA values are included in the counts.

df:

	A
0	1.0
1	2.0
2	NaN
3	2.0
4	3.0
5	NaN
6	3.0

df::

	A
0	1.0
1	2.0
2	NaN
3	2.0
4	3.0
5	NaN
6	3.0

Output (dropna=True):

```
          A
2.0       2
3.0       2
1.0       1
Name: count, dtype:
int64
```

Code (dropna=True):

```
df.value_counts(
    dropna=True
    )
# or simply:
df.value_counts()
```

Output (dropna=False):

```
          A
2.0       2
3.0       2
NaN       2
1.0       1
Name: count, dtype:
int64
```

Code (dropna=False):

```
df.value_counts(
    dropna=False
    )
```

Code Snippet

```
data = {'A': [1, 2, np.nan, 2, 3, np.nan, 3]}
df = pd.DataFrame(data)
df
```

→

df:

	A
0	1.0
1	2.0
2	NaN
3	2.0
4	3.0
5	NaN
6	3.0

Sources & References

pandas.DataFrame.value_counts Documentation (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.value_counts.html)

Contacts and Social Media

Kichere Magubu

Dar es Salaam, Tanzania

Kichere Magubu

Kichere The Data Scientist

kichere-thedatascientist

kicherethedatascientist@gmail.com

+255 654 729 851

Thank You!

Thank you for reading this e-book!

If you found it valuable,

please consider leaving an honest review.

Your feedback and support mean a lot to me!