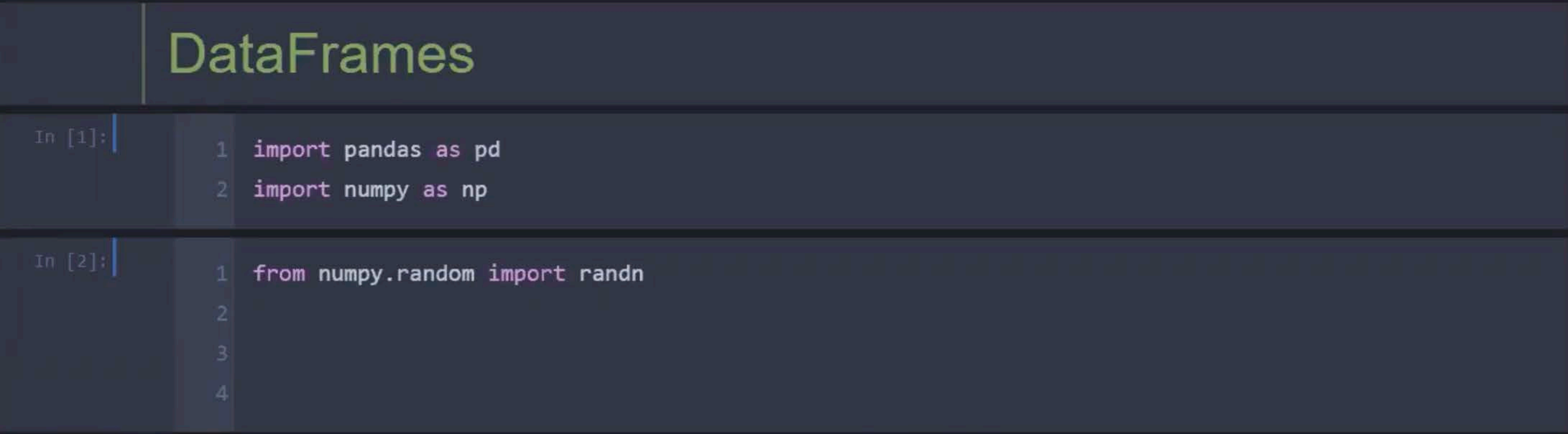


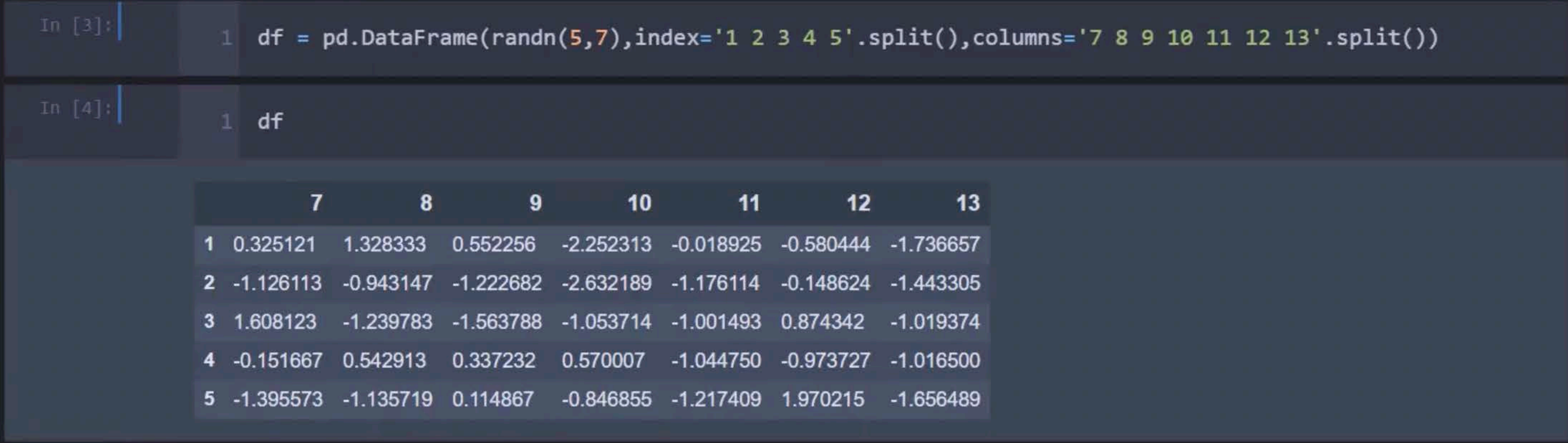
DataFrames

In [1]:

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

In [2]:

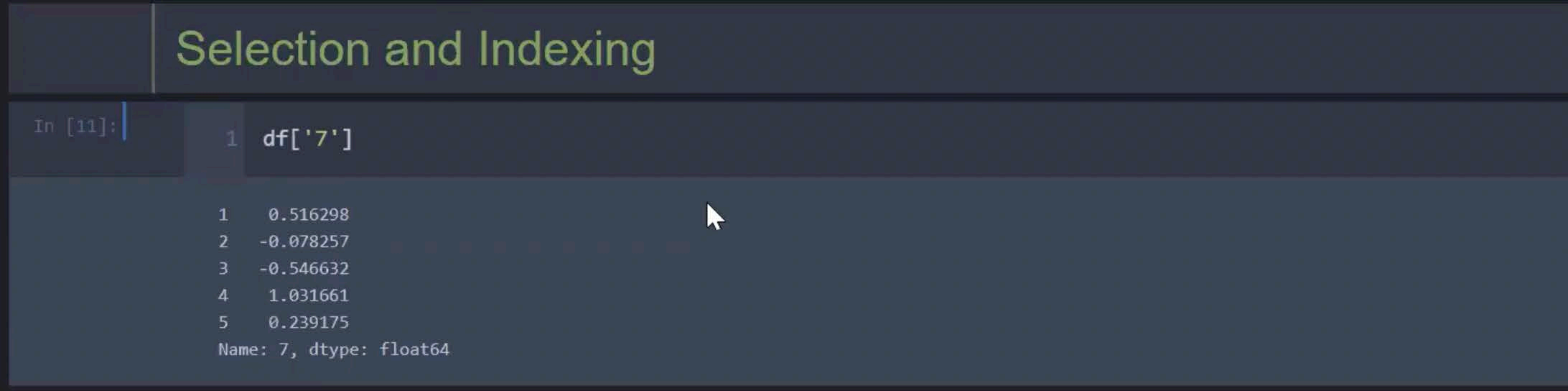
```python
from numpy.random import randn



```



```
In [3]:  df = pd.DataFrame(randn(5,7),index='1 2 3 4 5'.split(),columns='7 8 9 10 11 12 13'.split())
```

```
In [4]:  df
```

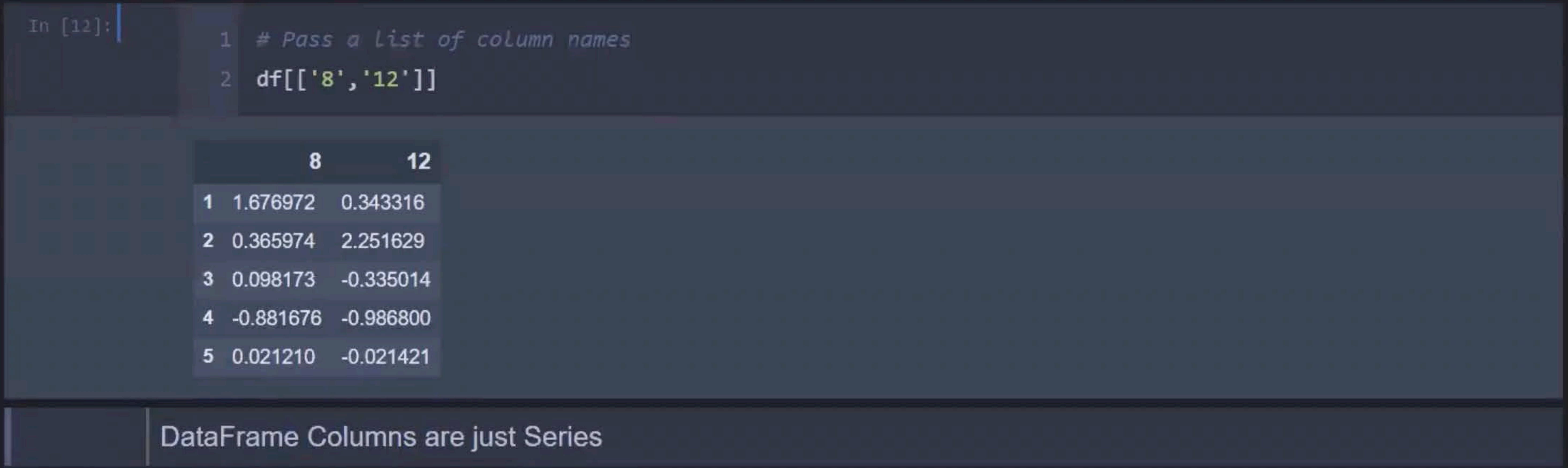
	7	8	9	10	11	12	13
1	0.325121	1.328333	0.552256	-2.252313	-0.018925	-0.580444	-1.736657
2	-1.126113	-0.943147	-1.222682	-2.632189	-1.176114	-0.148624	-1.443305
3	1.608123	-1.239783	-1.563788	-1.053714	-1.001493	0.874342	-1.019374
4	-0.151667	0.542913	0.337232	0.570007	-1.044750	-0.973727	-1.016500
5	-1.395573	-1.135719	0.114867	-0.846855	-1.217409	1.970215	-1.656489



Selection and Indexing

```
In [11]:
df['7']
```

```
1    0.516298
2   -0.078257
3   -0.546632
4    1.031661
5    0.239175
Name: 7, dtype: float64
```

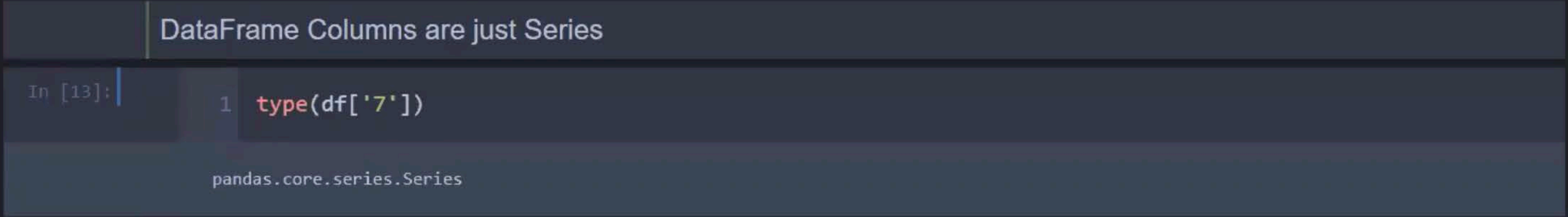


In [12]:

```python
# Pass a list of column names
df[['8','12']]
```

	8	12
1	1.676972	0.343316
2	0.365974	2.251629
3	0.098173	-0.335014
4	-0.881676	-0.986800
5	0.021210	-0.021421

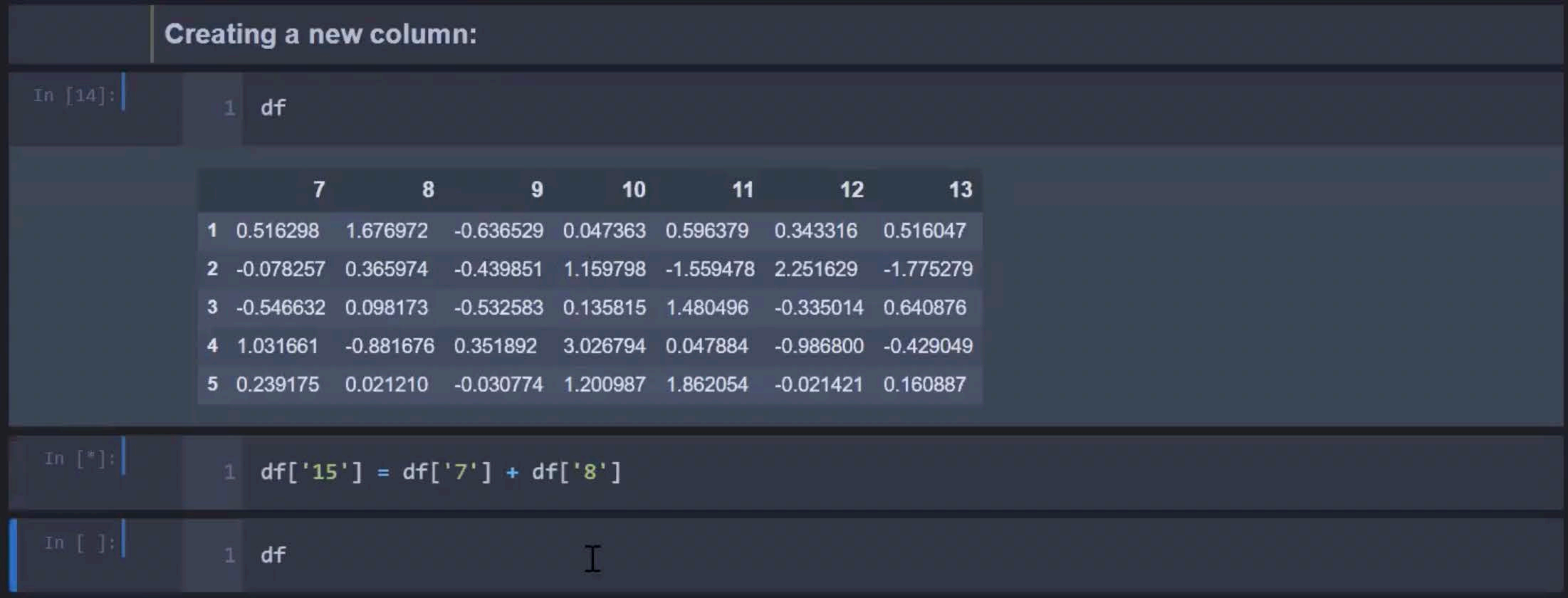
DataFrame Columns are just Series



DataFrame Columns are just Series

```
In [13]: type(df['7'])
```

```
pandas.core.series.Series
```



Creating a new column:

In [14]:

```
df
```

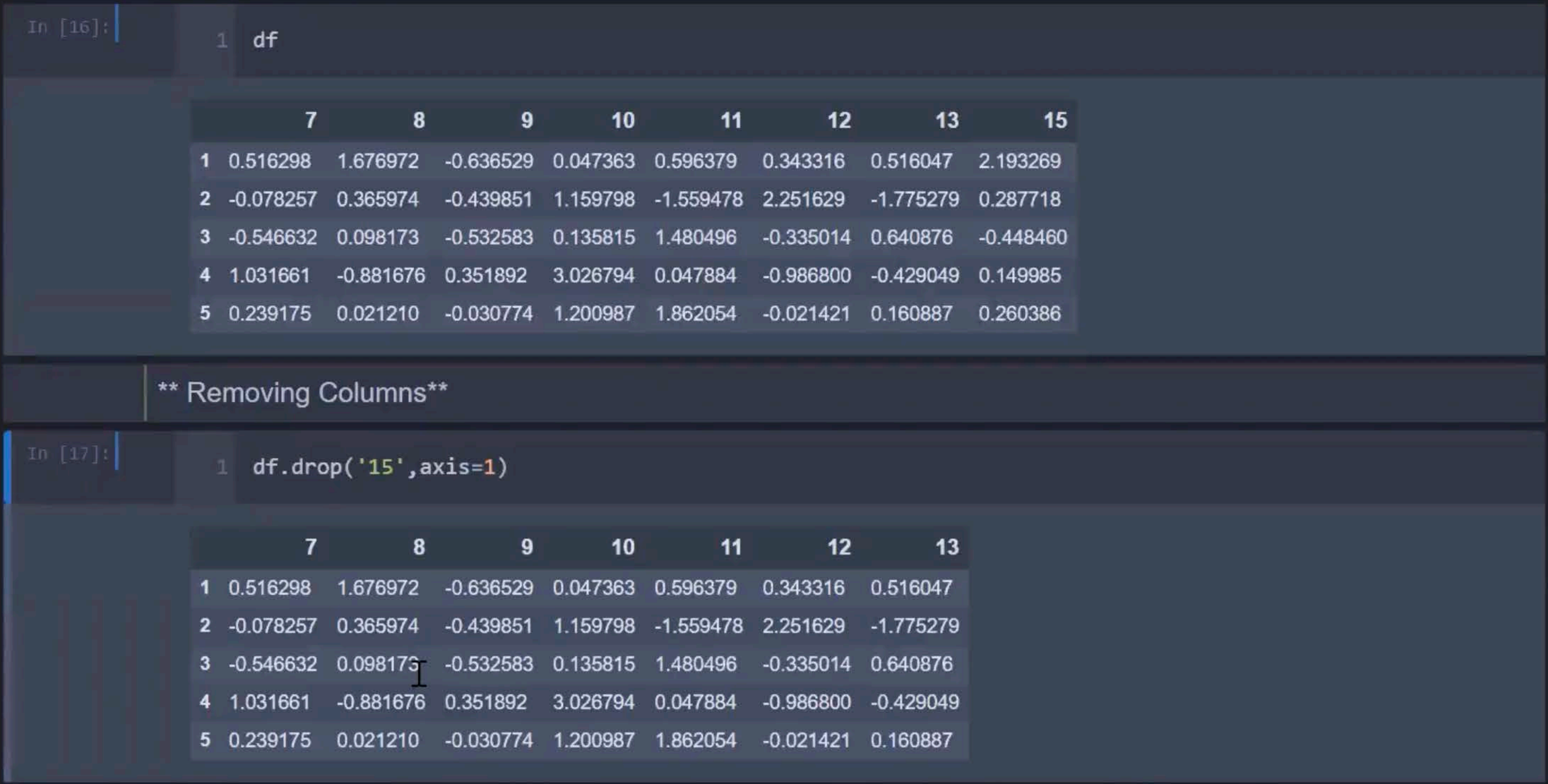
	7	8	9	10	11	12	13
1	0.516298	1.676972	-0.636529	0.047363	0.596379	0.343316	0.516047
2	-0.078257	0.365974	-0.439851	1.159798	-1.559478	2.251629	-1.775279
3	-0.546632	0.098173	-0.532583	0.135815	1.480496	-0.335014	0.640876
4	1.031661	-0.881676	0.351892	3.026794	0.047884	-0.986800	-0.429049
5	0.239175	0.021210	-0.030774	1.200987	1.862054	-0.021421	0.160887

In [*]:

```
df['15'] = df['7'] + df['8']
```

In [ ]:

```
df
```



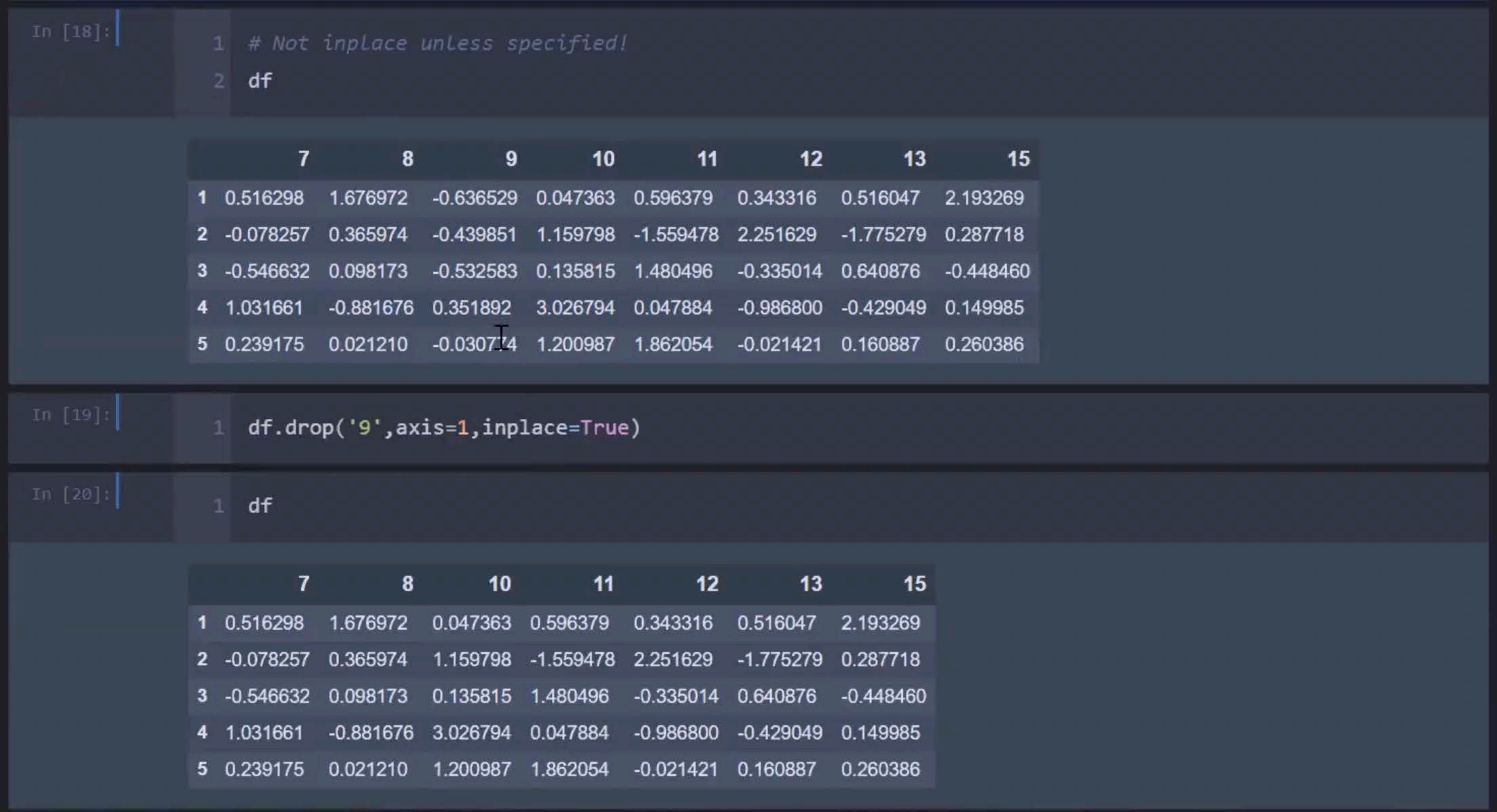
```
In [16]: df
```

	7	8	9	10	11	12	13	15
1	0.516298	1.676972	-0.636529	0.047363	0.596379	0.343316	0.516047	2.193269
2	-0.078257	0.365974	-0.439851	1.159798	-1.559478	2.251629	-1.775279	0.287718
3	-0.546632	0.098173	-0.532583	0.135815	1.480496	-0.335014	0.640876	-0.448460
4	1.031661	-0.881676	0.351892	3.026794	0.047884	-0.986800	-0.429049	0.149985
5	0.239175	0.021210	-0.030774	1.200987	1.862054	-0.021421	0.160887	0.260386

** Removing Columns**

```
In [17]: df.drop('15',axis=1)
```

	7	8	9	10	11	12	13
1	0.516298	1.676972	-0.636529	0.047363	0.596379	0.343316	0.516047
2	-0.078257	0.365974	-0.439851	1.159798	-1.559478	2.251629	-1.775279
3	-0.546632	0.098173	-0.532583	0.135815	1.480496	-0.335014	0.640876
4	1.031661	-0.881676	0.351892	3.026794	0.047884	-0.986800	-0.429049
5	0.239175	0.021210	-0.030774	1.200987	1.862054	-0.021421	0.160887



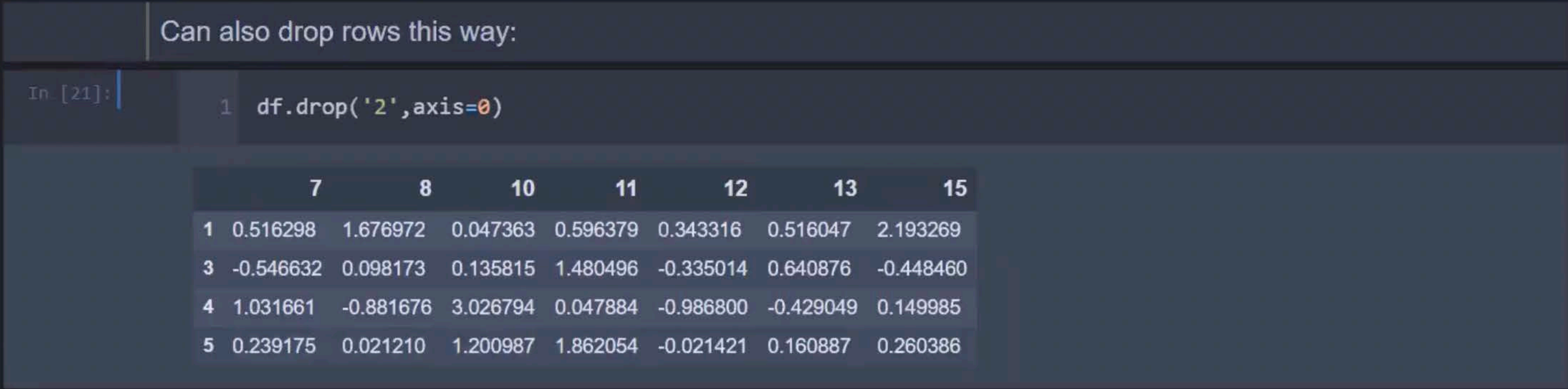
In [18]:
```
# Not inplace unless specified!
df
```

	7	8	9	10	11	12	13	15
1	0.516298	1.676972	-0.636529	0.047363	0.596379	0.343316	0.516047	2.193269
2	-0.078257	0.365974	-0.439851	1.159798	-1.559478	2.251629	-1.775279	0.287718
3	-0.546632	0.098173	-0.532583	0.135815	1.480496	-0.335014	0.640876	-0.448460
4	1.031661	-0.881676	0.351892	3.026794	0.047884	-0.986800	-0.429049	0.149985
5	0.239175	0.021210	-0.030774	1.200987	1.862054	-0.021421	0.160887	0.260386

In [19]:
```
df.drop('9',axis=1,inplace=True)
```

In [20]:
```
df
```

	7	8	10	11	12	13	15
1	0.516298	1.676972	0.047363	0.596379	0.343316	0.516047	2.193269
2	-0.078257	0.365974	1.159798	-1.559478	2.251629	-1.775279	0.287718
3	-0.546632	0.098173	0.135815	1.480496	-0.335014	0.640876	-0.448460
4	1.031661	-0.881676	3.026794	0.047884	-0.986800	-0.429049	0.149985
5	0.239175	0.021210	1.200987	1.862054	-0.021421	0.160887	0.260386

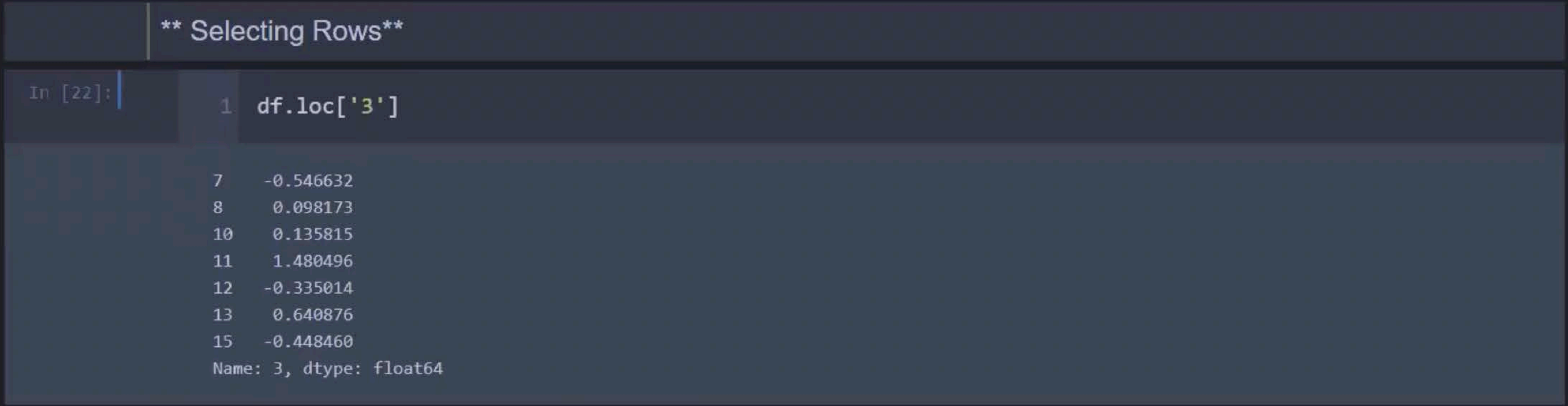


Can also drop rows this way:

In [21]:

```python
df.drop('2',axis=0)
```

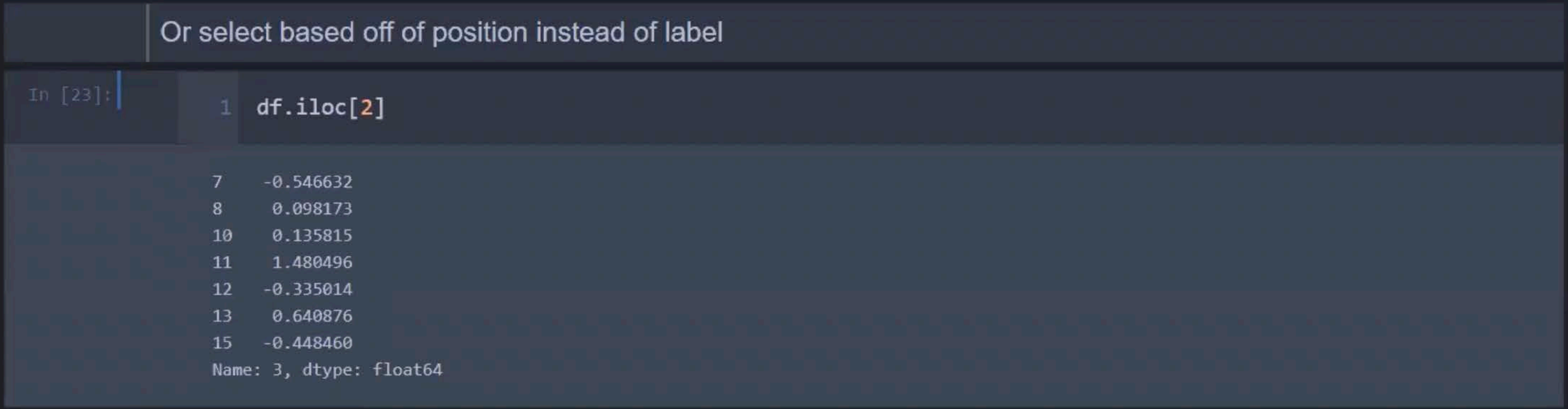
	7	8	10	11	12	13	15
1	0.516298	1.676972	0.047363	0.596379	0.343316	0.516047	2.193269
3	-0.546632	0.098173	0.135815	1.480496	-0.335014	0.640876	-0.448460
4	1.031661	-0.881676	3.026794	0.047884	-0.986800	-0.429049	0.149985
5	0.239175	0.021210	1.200987	1.862054	-0.021421	0.160887	0.260386



** Selecting Rows**

```
In [22]:
df.loc['3']
```

```
7    -0.546632
8     0.098173
10    0.135815
11    1.480496
12   -0.335014
13    0.640876
15   -0.448460
Name: 3, dtype: float64
```

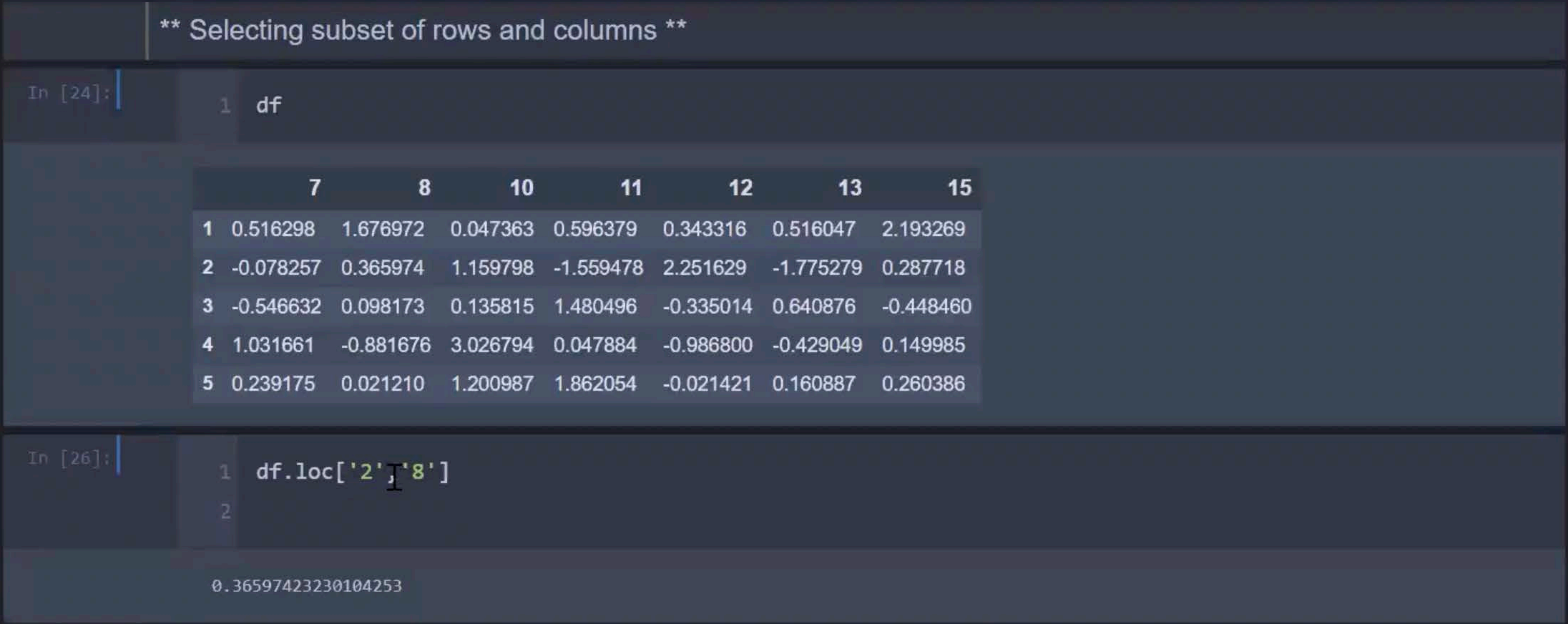


Or select based off of position instead of label

In [23]:

```python
df.iloc[2]
```

```
7    -0.546632
8     0.098173
10    0.135815
11    1.480496
12   -0.335014
13    0.640876
15   -0.448460
Name: 3, dtype: float64
```


** Selecting subset of rows and columns **

In [24]:

```
df
```

	7	8	10	11	12	13	15
1	0.516298	1.676972	0.047363	0.596379	0.343316	0.516047	2.193269
2	-0.078257	0.365974	1.159798	-1.559478	2.251629	-1.775279	0.287718
3	-0.546632	0.098173	0.135815	1.480496	-0.335014	0.640876	-0.448460
4	1.031661	-0.881676	3.026794	0.047884	-0.986800	-0.429049	0.149985
5	0.239175	0.021210	1.200987	1.862054	-0.021421	0.160887	0.260386

In [26]:

```
df.loc['2','8']
```

```
0.36597423230104253
```

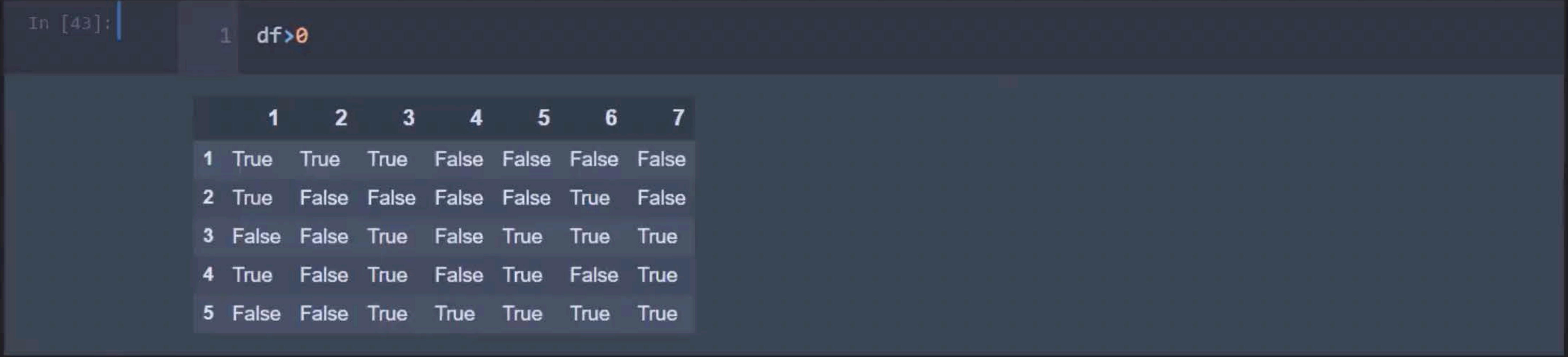
Conditional Selection

An important feature of pandas is conditional selection using bracket notation, very similar to numpy:

In [40]:

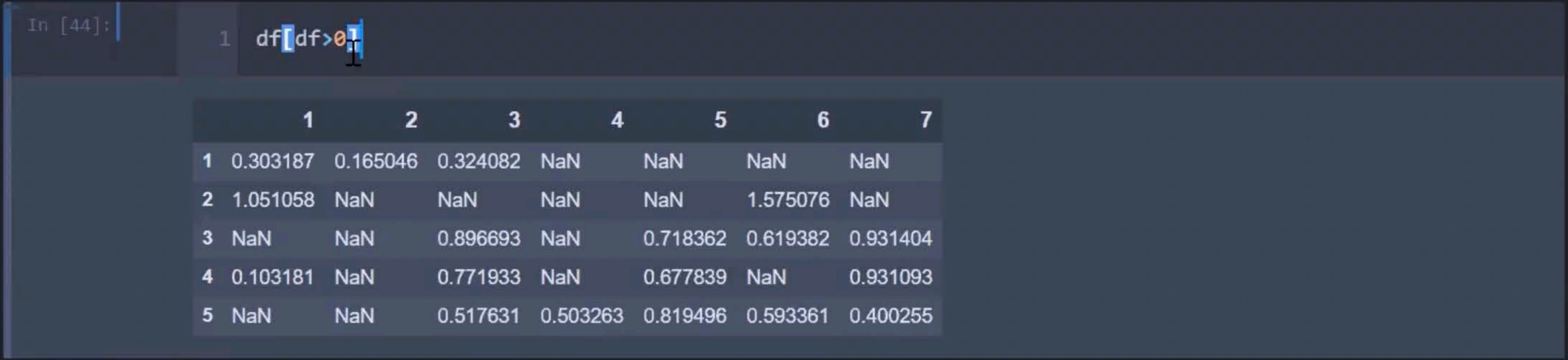
```
df
```

	1	2	3	4	5	6	7
1	0.303187	0.165046	0.324082	-0.136473	-1.046041	-1.796761	-0.622090
2	1.051058	-1.683402	-0.468334	-1.747161	-0.026593	1.575076	-0.307717
3	-0.456119	-0.658700	0.896693	-0.291655	0.718362	0.619382	0.931404
4	0.103181	-2.302459	0.771933	-0.168336	0.677839	-0.543745	0.931093
5	-0.644250	-0.343843	0.517631	0.503263	0.819496	0.593361	0.400255



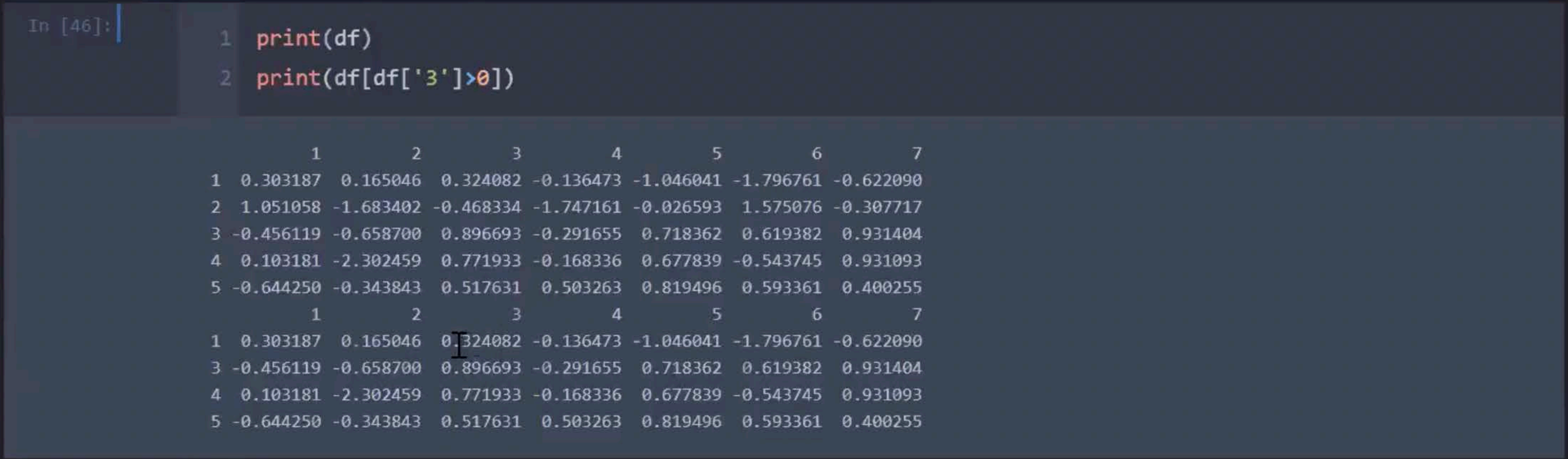
```
In [43]:
df>0
```

	1	2	3	4	5	6	7
1	True	True	True	False	False	False	False
2	True	False	False	False	False	True	False
3	False	False	True	False	True	True	True
4	True	False	True	False	True	False	True
5	False	False	True	True	True	True	True



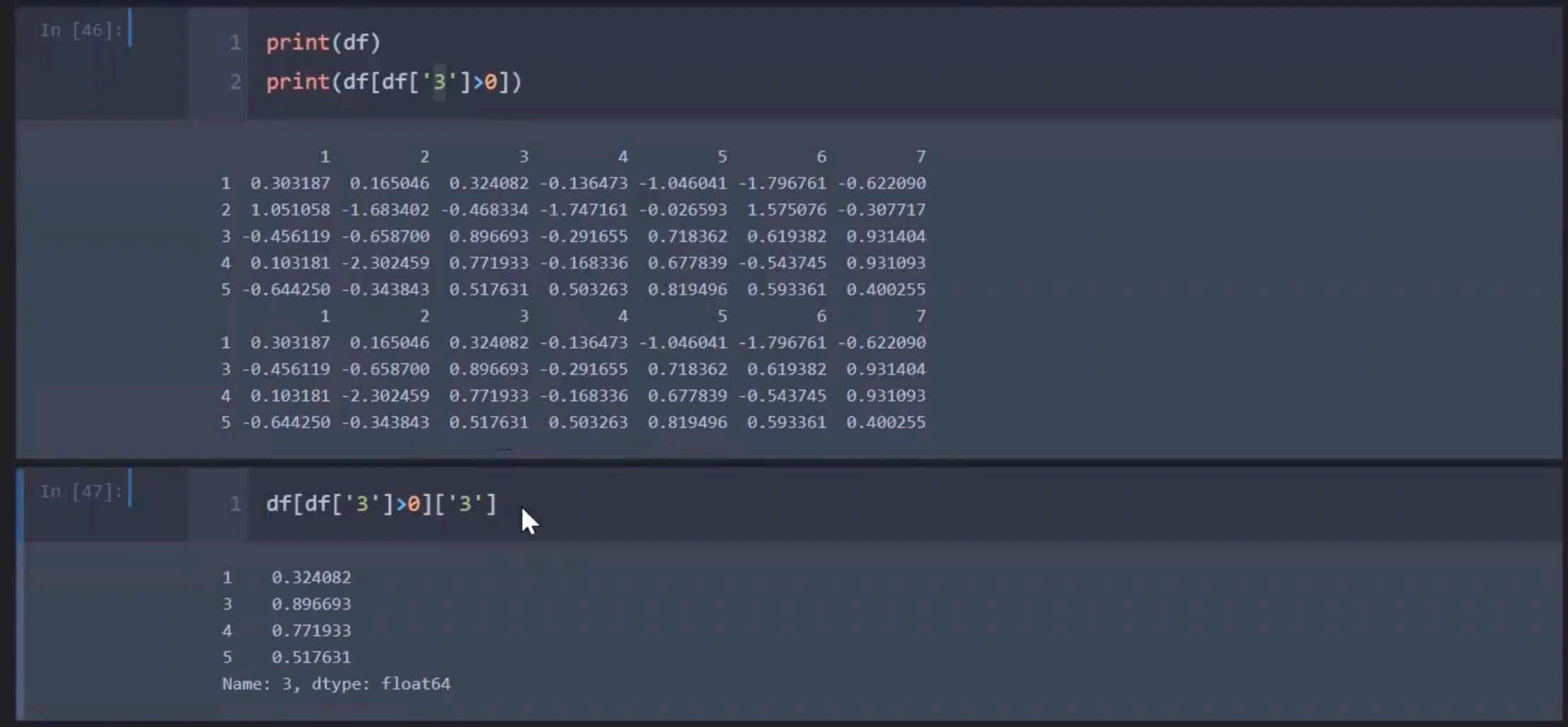
```
In [44]:  df[df>0]
```

	1	2	3	4	5	6	7
1	0.303187	0.165046	0.324082	NaN	NaN	NaN	NaN
2	1.051058	NaN	NaN	NaN	NaN	1.575076	NaN
3	NaN	NaN	0.896693	NaN	0.718362	0.619382	0.931404
4	0.103181	NaN	0.771933	NaN	0.677839	NaN	0.931093
5	NaN	NaN	0.517631	0.503263	0.819496	0.593361	0.400255



```
In [46]:
print(df)
print(df[df['3']>0])
```

```
          1         2         3         4         5         6         7
1  0.303187  0.165046  0.324082 -0.136473 -1.046041 -1.796761 -0.622090
2  1.051058 -1.683402 -0.468334 -1.747161 -0.026593  1.575076 -0.307717
3 -0.456119 -0.658700  0.896693 -0.291655  0.718362  0.619382  0.931404
4  0.103181 -2.302459  0.771933 -0.168336  0.677839 -0.543745  0.931093
5 -0.644250 -0.343843  0.517631  0.503263  0.819496  0.593361  0.400255
          1         2         3         4         5         6         7
1  0.303187  0.165046  0.324082 -0.136473 -1.046041 -1.796761 -0.622090
3 -0.456119 -0.658700  0.896693 -0.291655  0.718362  0.619382  0.931404
4  0.103181 -2.302459  0.771933 -0.168336  0.677839 -0.543745  0.931093
5 -0.644250 -0.343843  0.517631  0.503263  0.819496  0.593361  0.400255
```

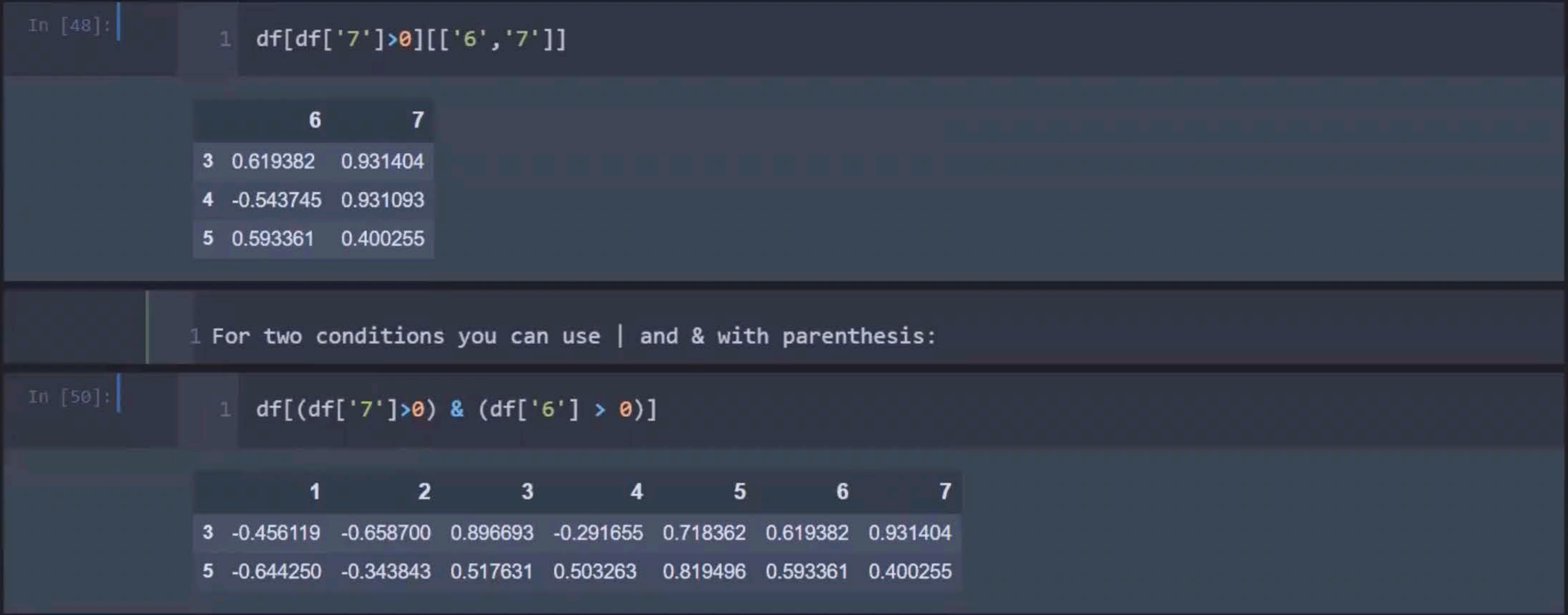


In [46]:
```
print(df)
print(df[df['3']>0])
```

```
          1         2         3         4         5         6         7
1  0.303187  0.165046  0.324082 -0.136473 -1.046041 -1.796761 -0.622090
2  1.051058 -1.683402 -0.468334 -1.747161 -0.026593  1.575076 -0.307717
3 -0.456119 -0.658700  0.896693 -0.291655  0.718362  0.619382  0.931404
4  0.103181 -2.302459  0.771933 -0.168336  0.677839 -0.543745  0.931093
5 -0.644250 -0.343843  0.517631  0.503263  0.819496  0.593361  0.400255
          1         2         3         4         5         6         7
1  0.303187  0.165046  0.324082 -0.136473 -1.046041 -1.796761 -0.622090
3 -0.456119 -0.658700  0.896693 -0.291655  0.718362  0.619382  0.931404
4  0.103181 -2.302459  0.771933 -0.168336  0.677839 -0.543745  0.931093
5 -0.644250 -0.343843  0.517631  0.503263  0.819496  0.593361  0.400255
```

In [47]:
```
df[df['3']>0]['3']
```

```
1    0.324082
3    0.896693
4    0.771933
5    0.517631
Name: 3, dtype: float64
```



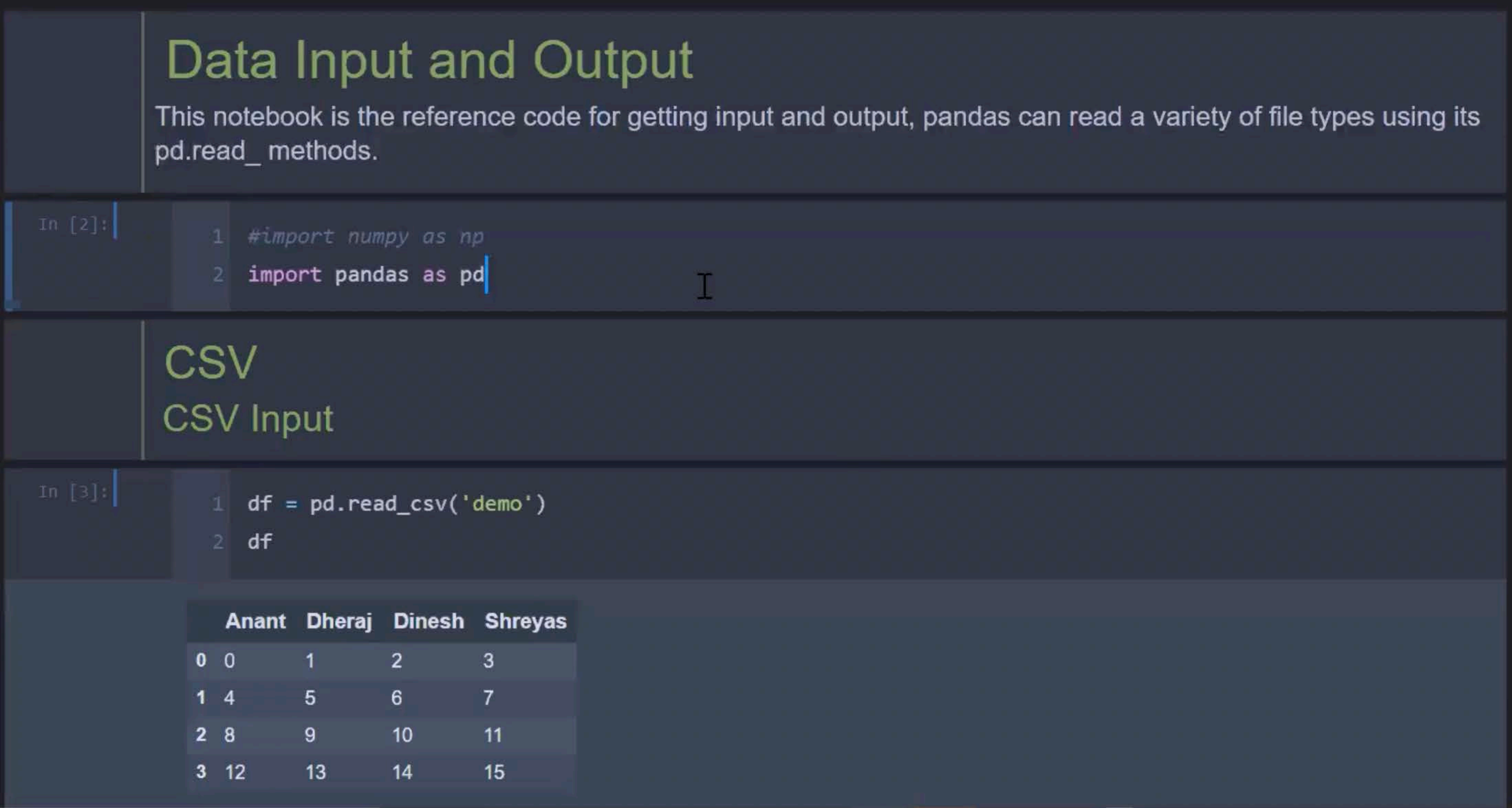
In [48]:
```
df[df['7']>0][['6','7']]
```

	6	7
3	0.619382	0.931404
4	-0.543745	0.931093
5	0.593361	0.400255

For two conditions you can use | and & with parenthesis:

In [50]:
```
df[(df['7']>0) & (df['6'] > 0)]
```

	1	2	3	4	5	6	7
3	-0.456119	-0.658700	0.896693	-0.291655	0.718362	0.619382	0.931404
5	-0.644250	-0.343843	0.517631	0.503263	0.819496	0.593361	0.400255



Data Input and Output

This notebook is the reference code for getting input and output, pandas can read a variety of file types using its pd.read_ methods.

In [2]:

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

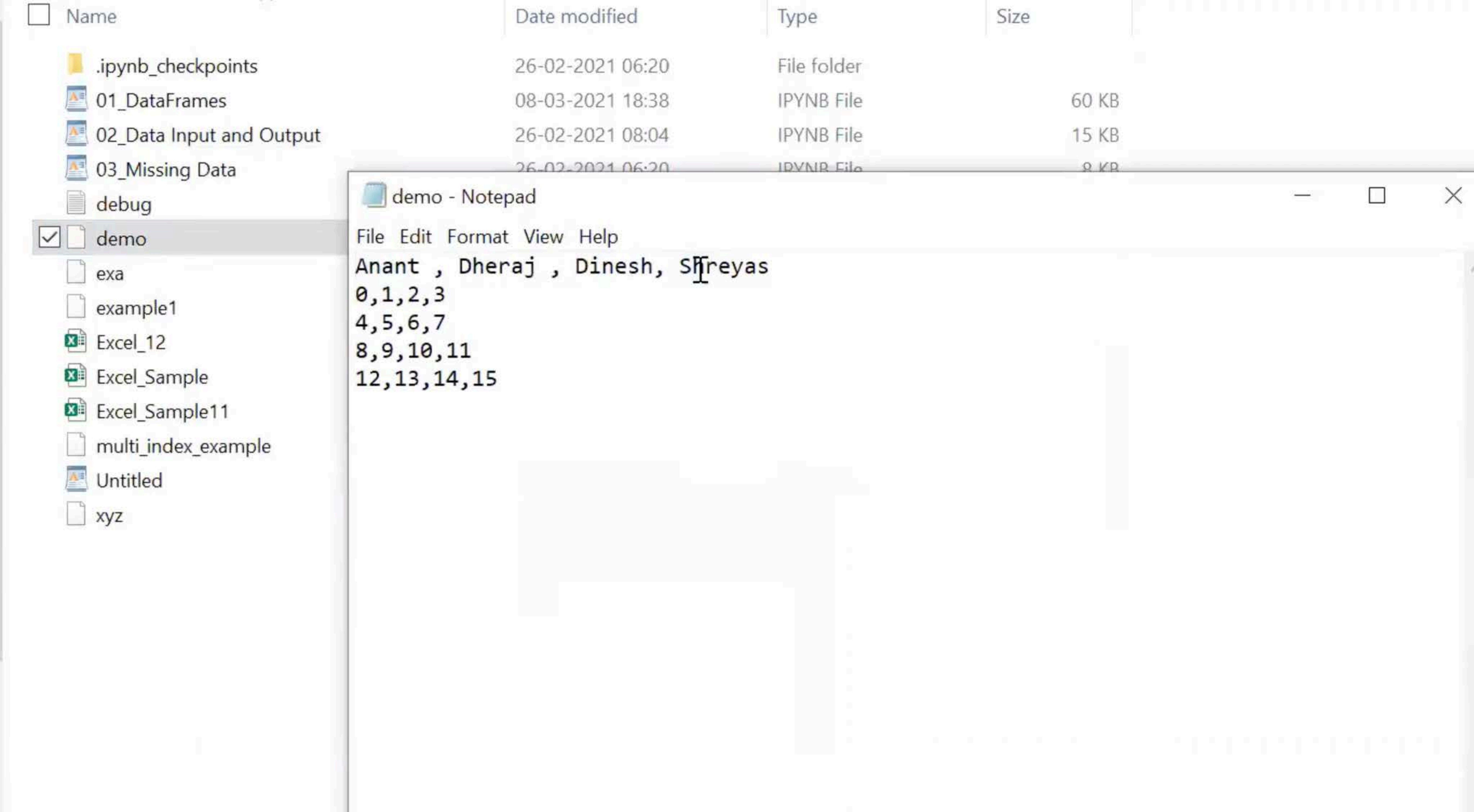
CSV

CSV Input

In [3]:

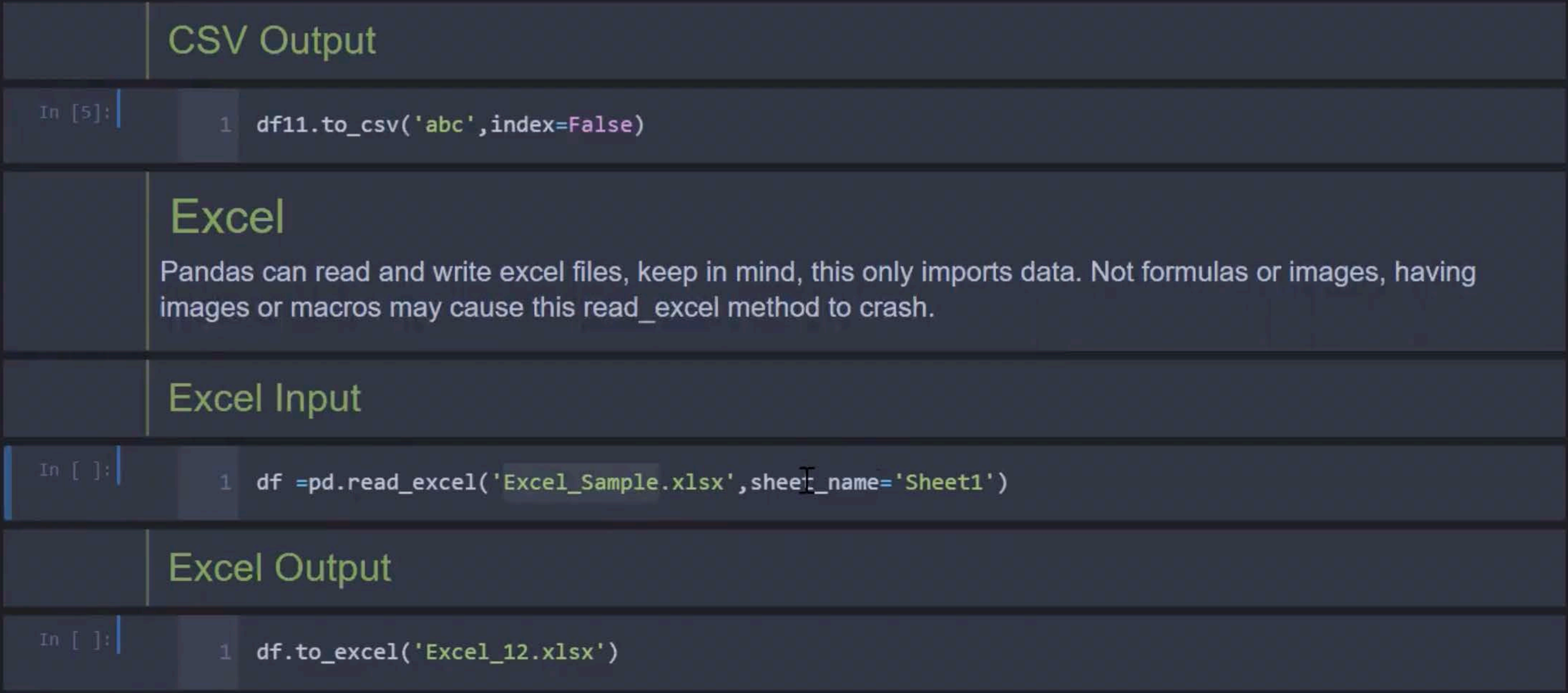
```
df = pd.read_csv('demo')
df
```

	Anant	Dheraj	Dinesh	Shreyas
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15



Name	Date modified	Type	Size
.ipynb_checkpoints	26-02-2021 06:20	File folder	
01_DataFrames	08-03-2021 18:38	IPYNB File	60 KB
02_Data Input and Output	26-02-2021 08:04	IPYNB File	15 KB
03_Missing Data	26-02-2021 06:20	IPYNB File	8 KB
debug			
demo			
exa			
example1			
Excel_12			
Excel_Sample			
Excel_Sample11			
multi_index_example			
Untitled			
xyz			

```
Anant , Dheraj , Dinesh, Shreyas
0,1,2,3
4,5,6,7
8,9,10,11
12,13,14,15
```



CSV Output

In [5]:
```
df11.to_csv('abc',index=False)
```

Excel

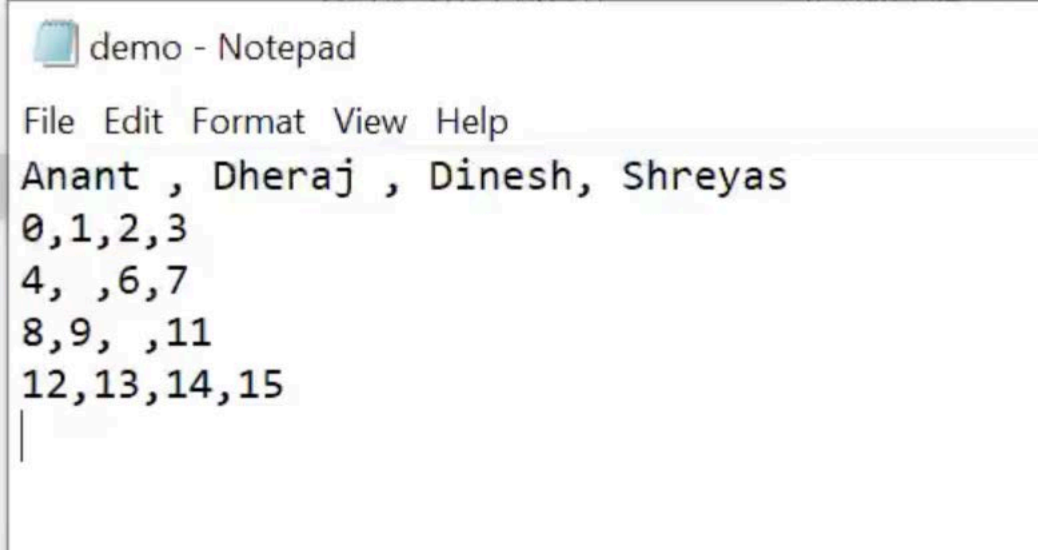
Pandas can read and write excel files, keep in mind, this only imports data. Not formulas or images, having images or macros may cause this read_excel method to crash.

Excel Input

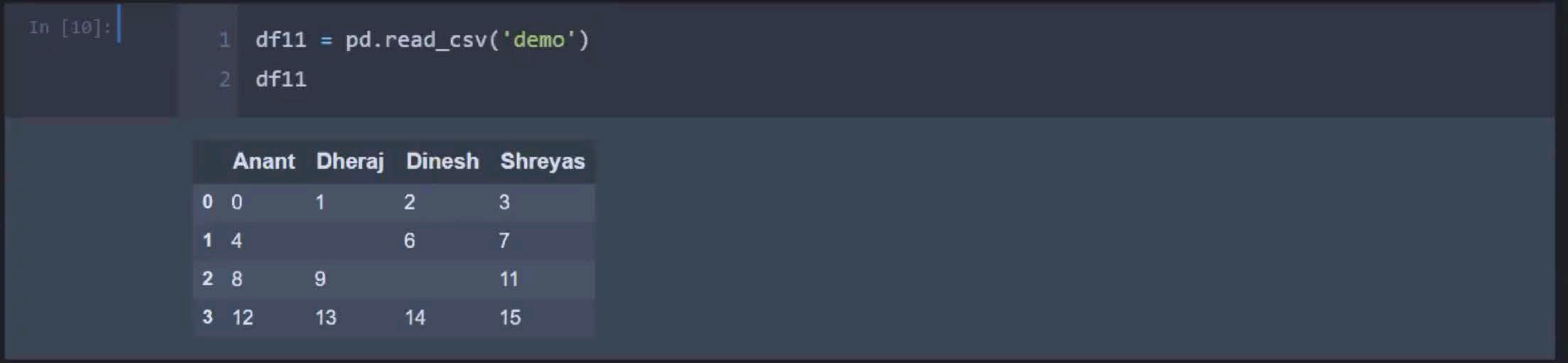
In [ ]:
```
df =pd.read_excel('Excel_Sample.xlsx',sheet_name='Sheet1')
```

Excel Output

In [ ]:
```
df.to_excel('Excel_12.xlsx')
```

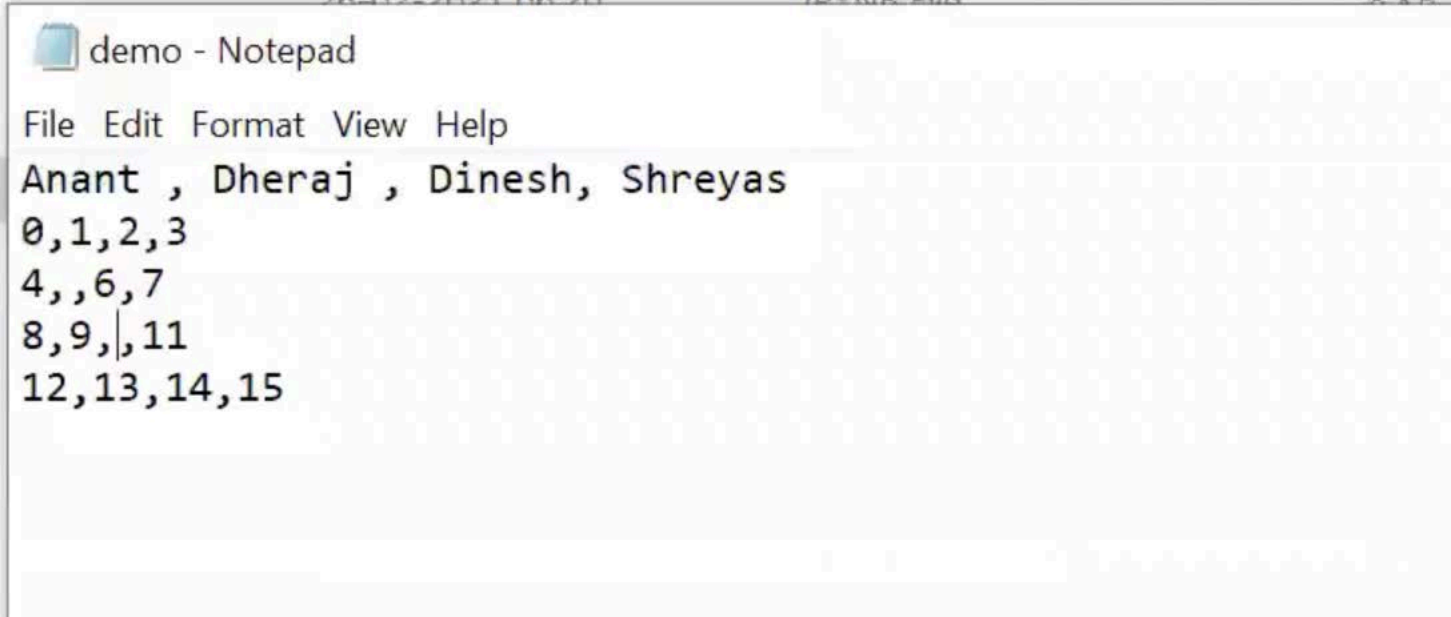


```
Anant , Dheraj , Dinesh, Shreyas
0,1,2,3
4, ,6,7
8,9, ,11
12,13,14,15
```

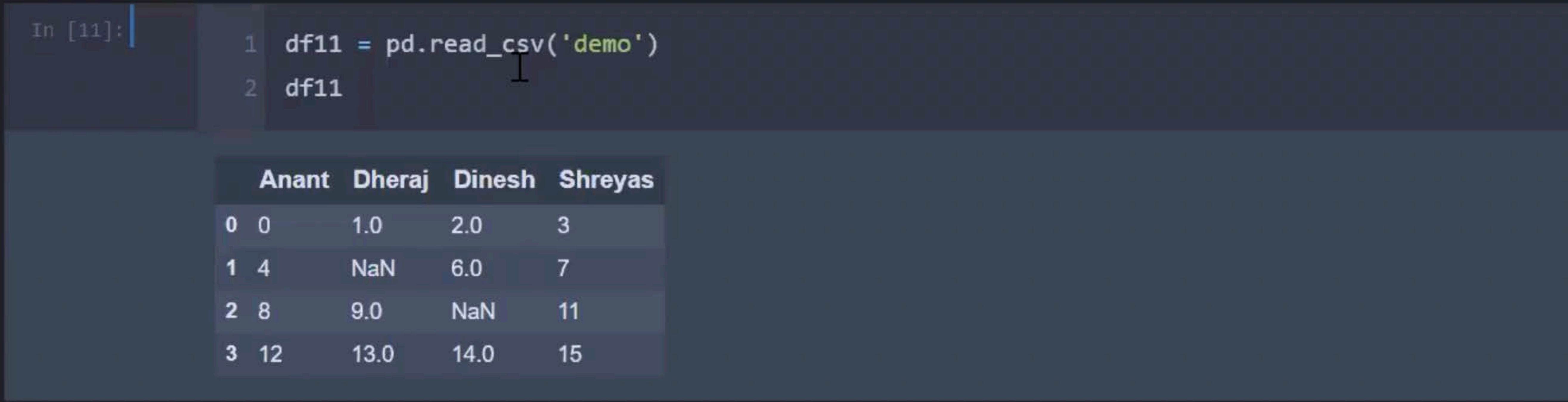


```
In [10]:
df11 = pd.read_csv('demo')
df11
```

	Anant	Dheraj	Dinesh	Shreyas
0	0	1	2	3
1	4		6	7
2	8	9		11
3	12	13	14	15

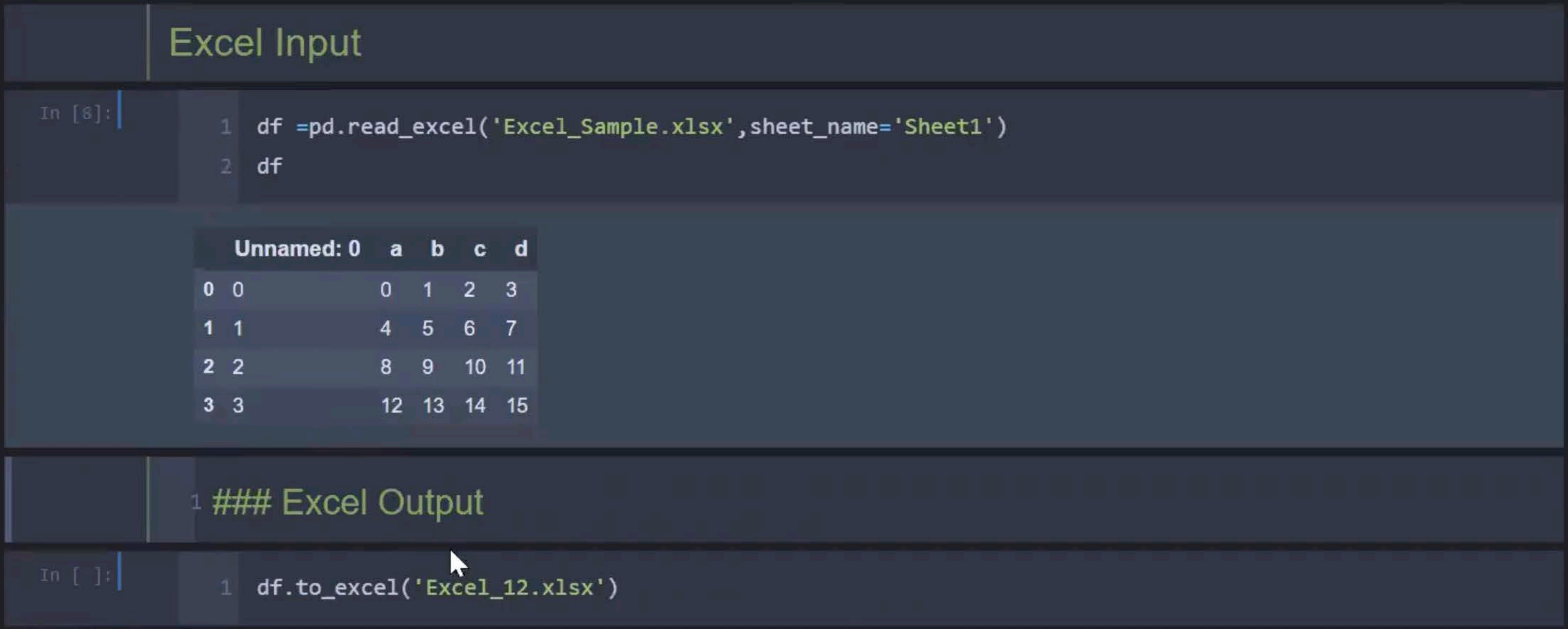


```
Anant , Dheraj , Dinesh, Shreyas
0,1,2,3
4,,6,7
8,9,,11
12,13,14,15
```



```
In [11]:
df11 = pd.read_csv('demo')
df11
```

	Anant	Dheraj	Dinesh	Shreyas
0	0	1.0	2.0	3
1	4	NaN	6.0	7
2	8	9.0	NaN	11
3	12	13.0	14.0	15



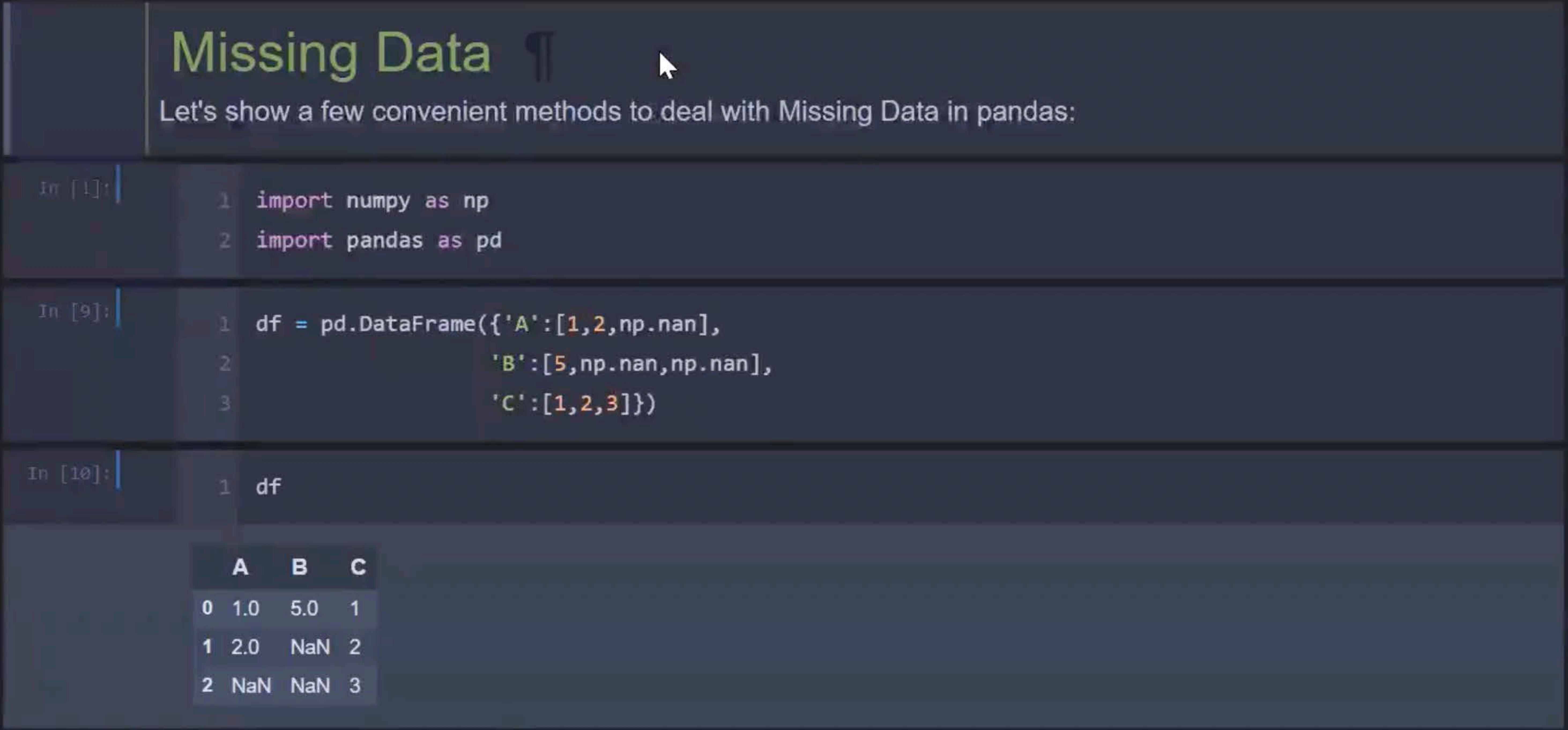
Excel Input

```
In [6]:
df =pd.read_excel('Excel_Sample.xlsx',sheet_name='Sheet1')
df
```

```
   Unnamed: 0   a   b   c   d
0  0            0   1   2   3
1  1            4   5   6   7
2  2            8   9  10  11
3  3           12  13  14  15
```

```
### Excel Output
```

```
In [ ]:
df.to_excel('Excel_12.xlsx')
```



Missing Data ¶

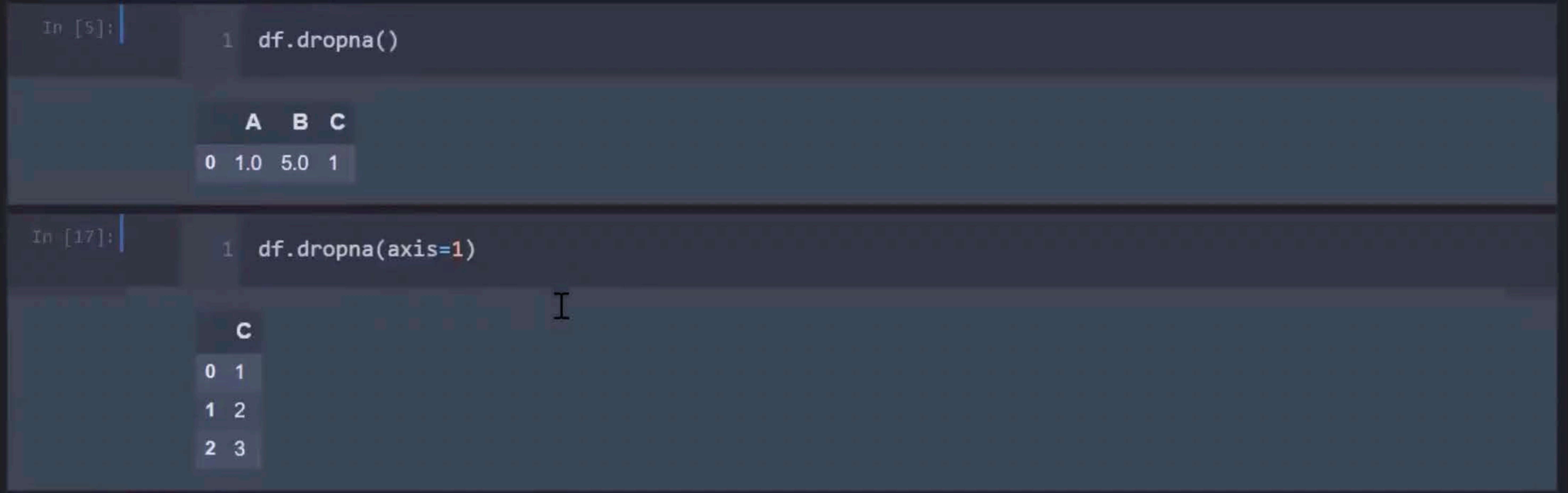
Let's show a few convenient methods to deal with Missing Data in pandas:

In [1]:
```python
import numpy as np
import pandas as pd
```

In [9]:
```python
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
```

In [10]:
```python
df
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

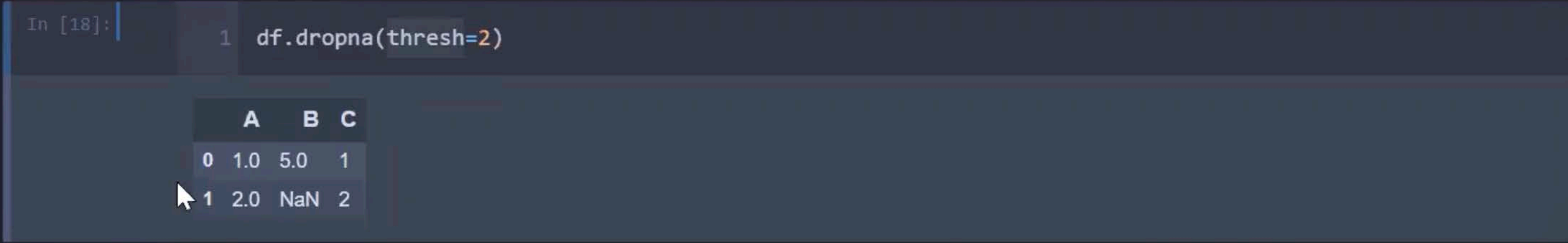


In [5]:
```
df.dropna()
```

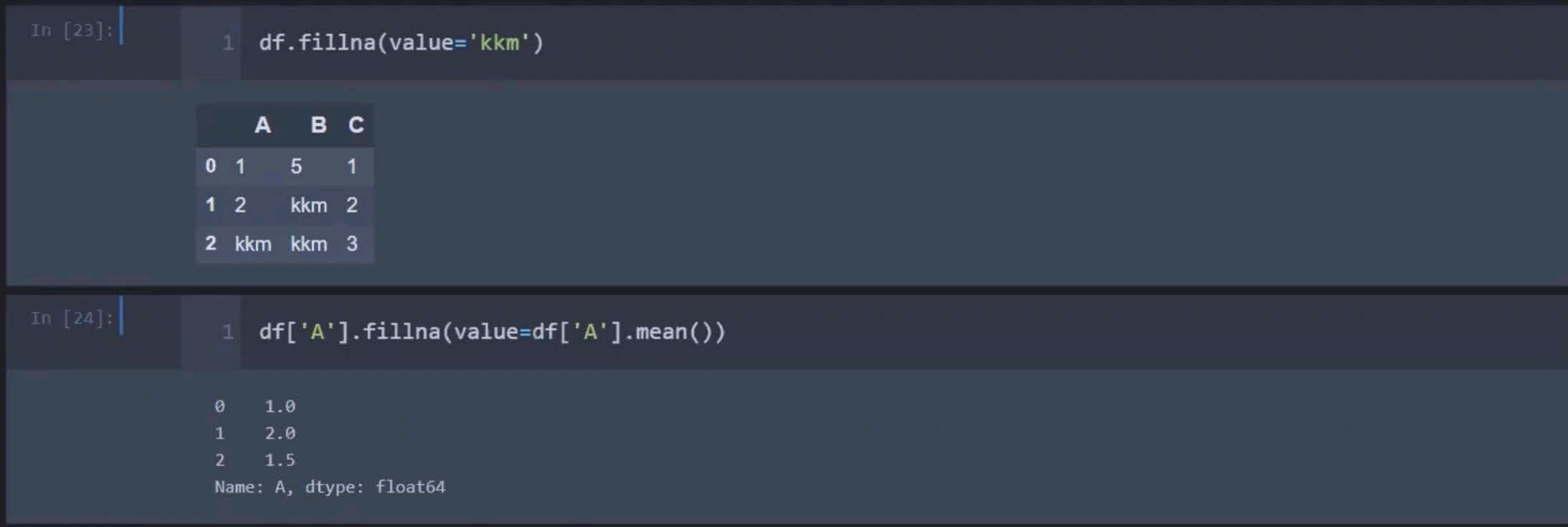
	A	B	C
0	1.0	5.0	1

In [17]:
```
df.dropna(axis=1)
```

	C
0	1
1	2
2	3


In [18]:
```python
df.dropna(thresh=2)
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

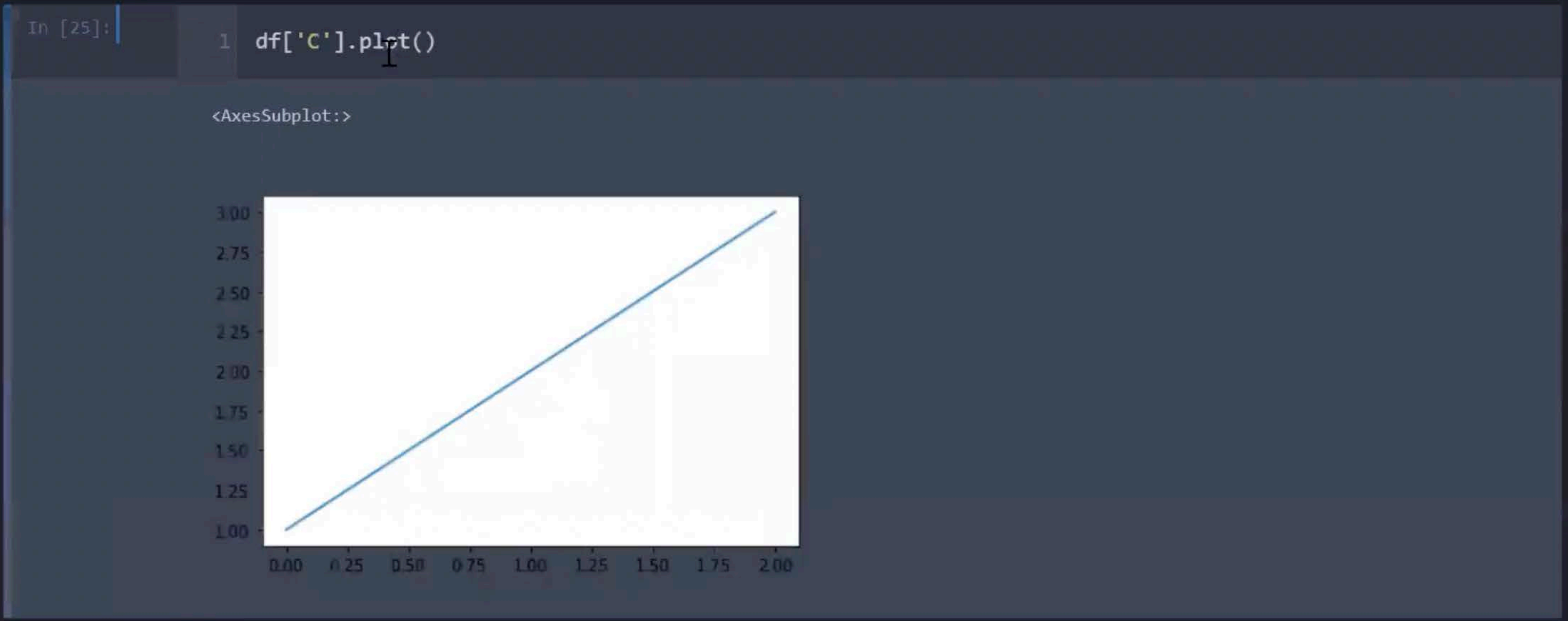


```
In [23]: df.fillna(value='kkm')
```

	A	B	C
0	1	5	1
1	2	kkm	2
2	kkm	kkm	3

```
In [24]: df['A'].fillna(value=df['A'].mean())
```

```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```



```
In [25]:
df['C'].plot()
```

```
<AxesSubplot:>
```