Mid Journey V4 prompt: grass on golf course was managed in a shape of an excel spreadsheet, one header row in darker color in the spreadsheet, pond, sand traps, top view, satellite view, photo by Nikon, Sunny, 8k --ar 16:9

Lecture 5-1: Pandas Dataframe

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

2023 APSC-5984 SS: Agriculture Data Science

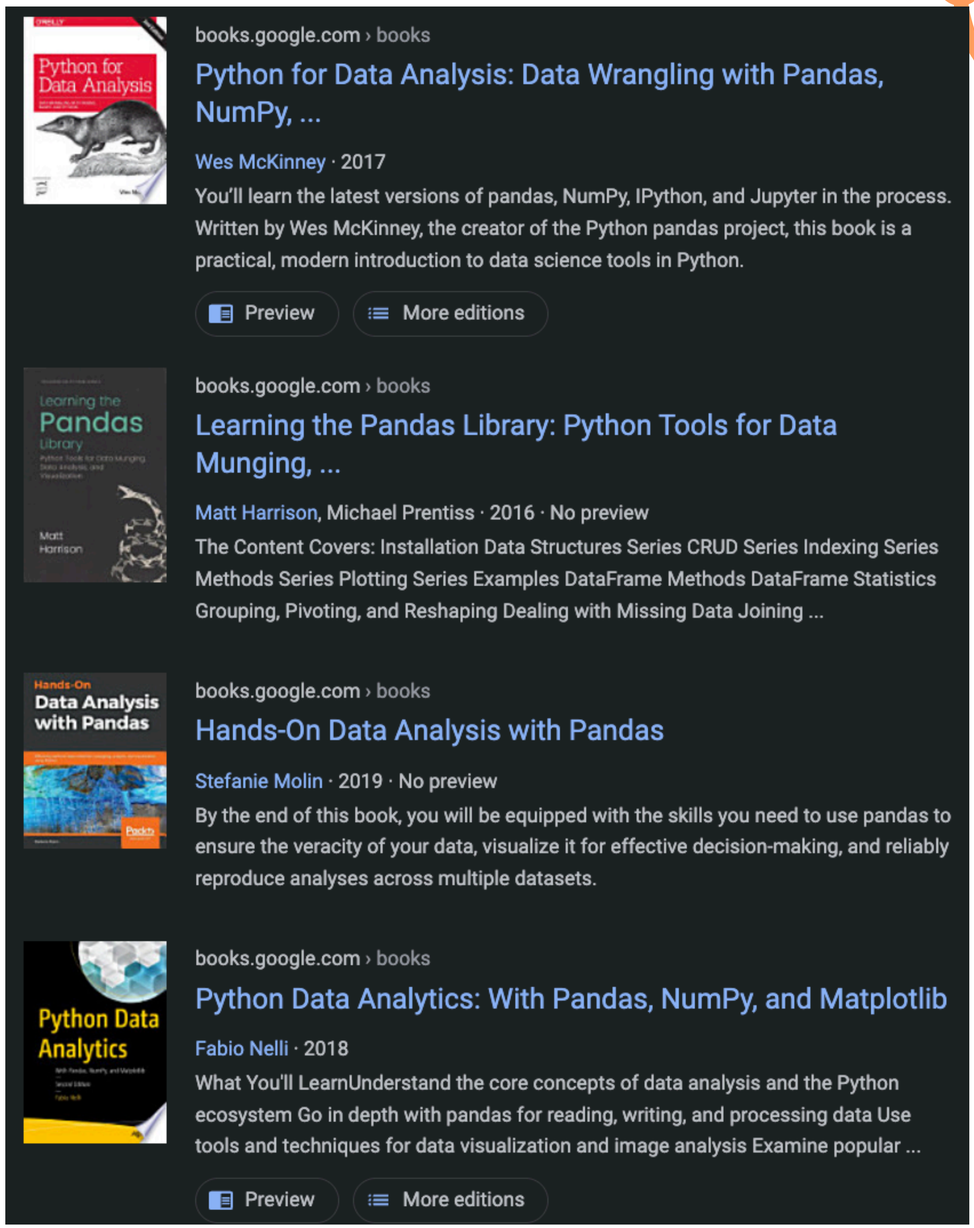
Pandas

Pandas library is the primary approach to manipulate data frames in Python

pandas

Original author(s)	Wes McKinney
Developer(s)	Community
Initial release	11 January 2008; 15 years ago[*citation needed*]
Stable release	1.5.3[1] / 18 January 2023; 24 days ago
Repository	github.com/pandas-dev/pandas
Written in	Python, Cython, C
Operating system	Cross-platform
Type	Technical computing
License	New BSD License
Website	pandas.pydata.org

```
import pandas as pd
```



https://en.wikipedia.org/wiki/Pandas_(software)

Pandas - CSV and Tab-Delimited Files

1.1.1 Separators

The basic function to load data in `pandas` is `pd.read_csv()`. It can read data from a CSV file or a tab-delimited file. The default delimiter is comma ",", but it also allows you to specify other delimiters, such as tab "`\t`".

The file `file_A.csv` is a CSV file with comma as the delimiter:

Shell

```
!cat file_A.csv
```

```
id,A,B,C
a1,1,1,1
a2,0,1,0
a3,1,0,1
```

Python

```
pd.read_csv('file_A.csv')
```

By default, “,” is the delimiter

```
   id  A  B  C
0  a1  1  1  1
1  a2  0  1  0
2  a3  1  0  1
```

Pandas - CSV and Tab-Delimited Files

1.1.1 Separators

The basic function to load data in `pandas` is `pd.read_csv()`. It can read data from a CSV file or a tab-delimited file. The default delimiter is comma ",", but it also allows you to specify other delimiters, such as tab "`\t`".

The file `file_A.csv` is a CSV file with comma as the delimiter:

Shell

```
!cat file_B.txt
```

```
id  A   B   C
a1  1   1   1
a2  0   1   0
a3  1   0   1
```

Python

```
pd.read_csv('file_B.txt')
```

“,” cannot separate the tab-delimited file

```
   id\tA\tB\tC
0  a1\t1\t1\t1
1  a2\t0\t1\t0
2  a3\t1\t0\t1
```

Pandas - CSV and Tab-Delimited Files

Sep = “,” (default)

```
pd.read_csv('file_B.txt')
```

C1

```
   id\tA\tB\tC
0  a1\t1\t1\t1
1  a2\t0\t1\t0
2  a3\t1\t0\t1
```

Sep = “\t”

```
pd.read_csv('file_B.txt', sep='\t')
```

C1 C2 C3 C4

	id	A	B	C
0	a1	1	1	1
1	a2	0	1	0
2	a3	1	0	1

Pandas - Header

File

```
a1,1,1,1
a2,0,1,0
a3,1,0,1
```

Default

```
pd.read_csv('file_A_nh.csv')
```

```
   a1  1  1.1  1.2
0  a2  0    1    0
1  a3  1    0    1
```

No header

```
pd.read_csv('file_A_nh.csv', header=None)
```

```
    0  1  2  3
0  a1  1  1  1
1  a2  0  1  0
2  a3  1  0  1
```

Pandas - Header


```
!cat file_A_2h.csv
```

```
R0  id,A,B,C
R1  a1,1,1,1
R2  a2,0,1,0
R3  a3,1,0,1
R4  id,D,E,F
R5  a4,1,1,1
R6  a5,0,1,0
R7  a6,1,0,1
```

Take the 5th row as the header

```
pd.read_csv('file_A_2h.csv', header=4)
```

```
   id  D  E  F
0  a4  1  1  1
1  a5  0  1  0
2  a6  1  0  1
```

Pandas - Excel Spreadsheet

Excel spreadsheet is a common format for data storage. However, given it is a format that contains multiple sheets, it is not straightforward to load it into a tabular format.

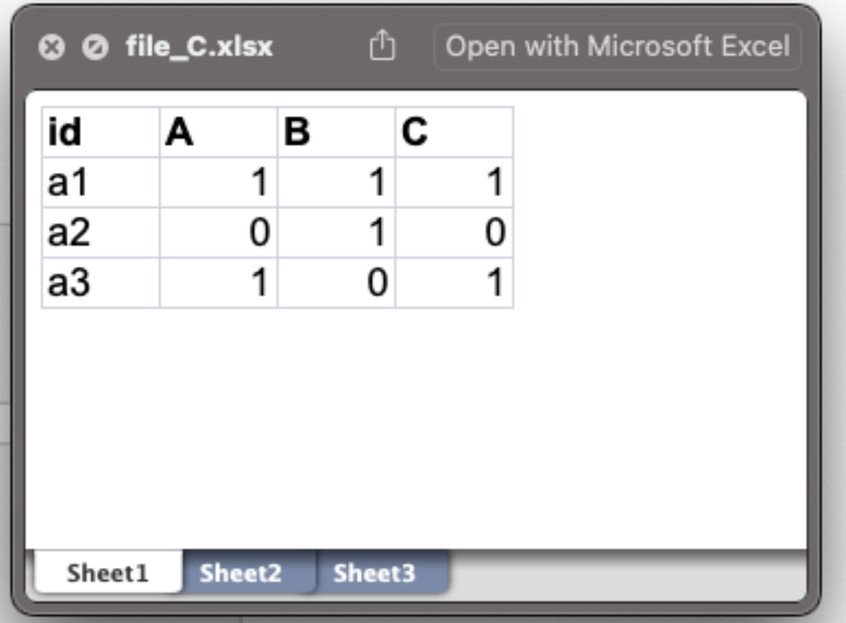
1.2.1 Load a single sheet

Here is an example of using `pd.read_excel()` to load the spreadsheet `file_C.xlsx`:

```
pd.read_excel('file_C.xlsx')
```

```
   id  A  B  C
0  a1  1  1  1
1  a2  0  1  0
2  a3  1  0  1
```

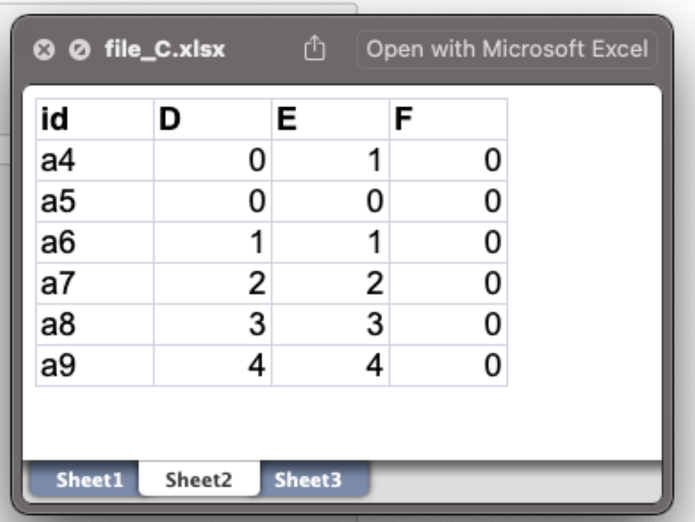
Only the first sheet was loaded



Pandas - Excel Spreadsheet

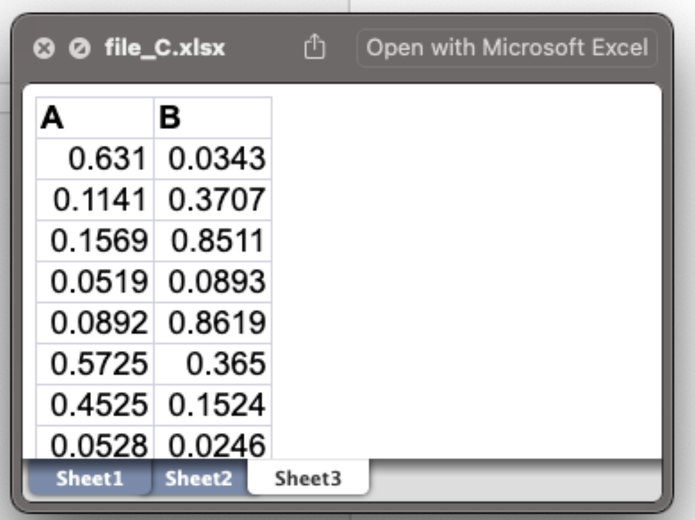

```
pd.read_excel('file_C.xlsx', sheet_name='Sheet2')
```

```
   id  D  E  F
0  a4  0  1  0
1  a5  0  0  0
2  a6  1  1  0
3  a7  2  2  0
4  a8  3  3  0
5  a9  4  4  0
```




```
pd.read_excel('file_C.xlsx', sheet_name='Sheet3')
```

```
          A         B
0  0.631007  0.034287
1  0.114071  0.370723
2  0.156949  0.851093
3  0.051913  0.089328
4  0.089216  0.861941
5  0.572473  0.364972
6  0.452546  0.152391
7  0.052752  0.024641
```

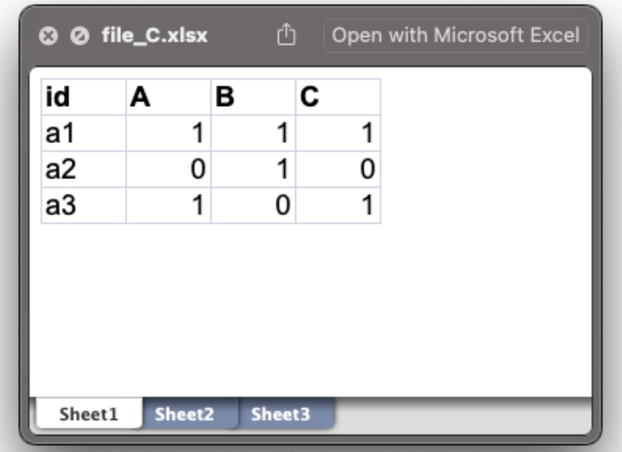


Pandas - Excel Spreadsheet (Dictionary Data Structure)

1.2.2 Dictionary of dataframes

In pandas, Excel spreadsheet is loaded as a dictionary of dataframes. The keys are the sheet names, and the values are the dataframes.

To load the entire spreadsheet taht contains all sheets, we can use `pd.read_excel()` with `sheet_name=None`:




```
data = pd.read_excel('file_C.xlsx', sheet_name=None)
print(data)
```

```
{'Sheet1':    id  A  B  C
0  a1  1  1  1
1  a2  0  1  0
2  a3  1  0  1, 'Sheet2':    id  D  E  F
0  a4  0  1  0
1  a5  0  0  0
2  a6  1  1  0
3  a7  2  2  0
4  a8  3  3  0
5  a9  4  4  0, 'Sheet3':           A         B
0  0.631007  0.034287
1  0.114071  0.370723
2  0.156949  0.851093
3  0.051913  0.089328
4  0.089216  0.861941
5  0.572473  0.364972
6  0.452546  0.152391
7  0.052752  0.024641}
```

It is a dictionary of data:
use “Sheet3” as a key to look up its value

```
data["Sheet3"]
```

	A	B
0	0.631007	0.034287
1	0.114071	0.370723
2	0.156949	0.851093
3	0.051913	0.089328
4	0.089216	0.861941
5	0.572473	0.364972
6	0.452546	0.152391
7	0.052752	0.024641

Pandas - Save Data

1.3.1 Save as CSV

We can use `df.to_csv()` to save a dataframe as a CSV file. Here are parameters that we can use:

- `sep`: the delimiter. Default is comma ",".
- `index`: whether to save the index column. Default is `True`.
- `header`: whether to save the header. Default is `True`.
- `columns`: the columns to save. Default is `None` (all columns).
- `mode`: the mode to open the file. Default is `"w"` (write). Other options are `"a"` (append) and `"r"` (read).

```
data["Sheet1"].to_csv('out_A.csv')
!cat out_A.csv
```

Output table

```
,id,A,B,C
0,a1,1,1,1
1,a2,0,1,0
2,a3,1,0,1
```

header

index

Table in Python

	id	A	B	C
0	a1	1	1	1
1	a2	0	1	0
2	a3	1	0	1

header

index

Pandas - Save Data

Default

```
data["Sheet1"].to_csv('out_A.csv')
!cat out_A.csv
```

```
,id,A,B,C
0,a1,1,1,1
1,a2,0,1,0
2,a3,1,0,1
```

No index, header

```
data["Sheet1"].to_csv('out_A.csv', index=False, header=None)
!cat out_A.csv
```

```
a1,1,1,1
a2,0,1,0
a3,1,0,1
```

No index

```
data["Sheet1"].to_csv('out_A.csv', index=False)
!cat out_A.csv
```

```
id,A,B,C
a1,1,1,1
a2,0,1,0
a3,1,0,1
```

Tab-delimited

```
data["Sheet1"].to_csv('out_A.csv', index=False, header=None, sep='\t')
!cat out_A.csv
```

```
a1  1   1   1
a2  0   1   0
a3  1   0   1
```

Pandas - Save Data

Table in Python

	id	A	B	C
0	a1	1	1	1
1	a2	0	1	0
2	a3	1	0	1

Only the columns of ‘A’ and ‘B’ were saved

```
data["Sheet1"].to_csv('out_A.csv', index=False, columns=['A', 'B'])
!cat out_A.csv
```

```
A,B
1,1
0,1
1,0
```

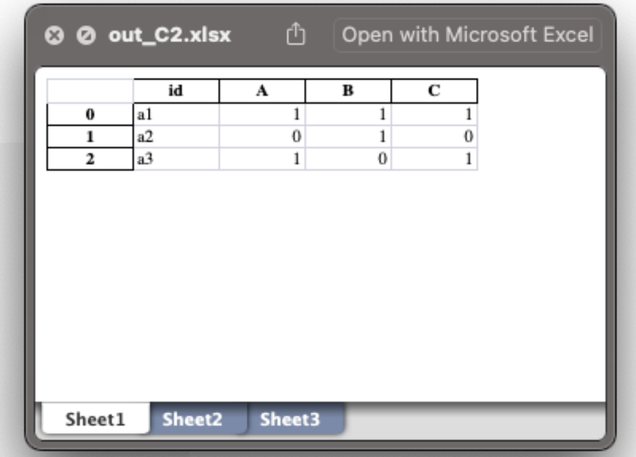
Pandas - Save Spreadsheet

1.3.2 Save as Excel spreadsheet

Pandas also allows us to save a dataframe as an Excel spreadsheet. It is highly recommended to interact with Excel spreadsheet using `with` statement when you want to work with multiple sheets. Here is an example:

Create a new spreadsheet

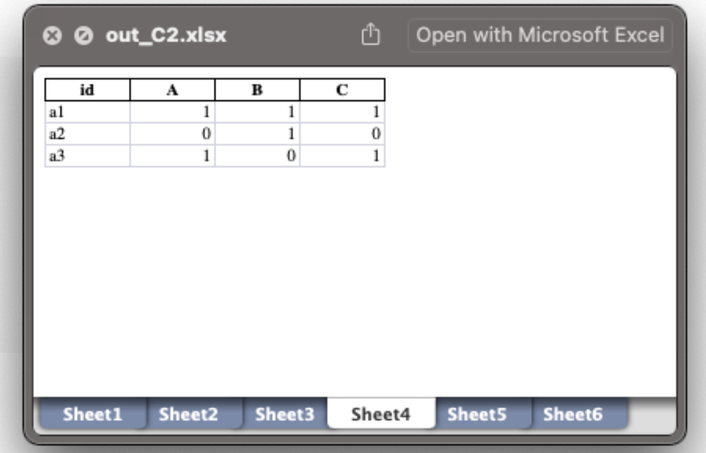
```
with pd.ExcelWriter('out_C2.xlsx') as writer:
    data["Sheet1"].to_excel(writer, sheet_name='Sheet1')
    data["Sheet2"].to_excel(writer, sheet_name='Sheet2')
    data["Sheet3"].to_excel(writer, sheet_name='Sheet3')
```



	id	A	B	C
0	a1	1	1	1
1	a2	0	1	0
2	a3	1	0	1

Append the spreadsheet

```
with pd.ExcelWriter('out_C2.xlsx', mode="a") as writer:
    data["Sheet1"].to_excel(writer, sheet_name='Sheet4', index=False)
    data["Sheet2"].to_excel(writer, sheet_name='Sheet5', index=False)
    data["Sheet3"].to_excel(writer, sheet_name='Sheet6', index=False)
```



id	A	B	C
a1	1	1	1
a2	0	1	0
a3	1	0	1

Pandas - Create a Dataframe

2. Construct a dataframe

We can also construct a dataframe from scratch. We can start with a dictionary of lists to define our dataframe:

```
data = dict()
data["id"] = ["id1", "id2", "id3", "id4"]
data["factor"] = ["A", "B", "A", "B"]
data["value"] = [1, 2, 3, 4]
print(data)
```

```
{'id': ['id1', 'id2', 'id3', 'id4'], 'factor': ['A', 'B', 'A', 'B'],
'value': [1, 2, 3, 4]}
```

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

	id	factor	value
0	id1	A	1
1	id2	B	2
2	id3	A	3
3	id4	B	4

Pandas - Dataframe Manipulation

3.1 Index location (.iloc)

We can use `.iloc()` method to access the data by numeric index location. The indexing rule is the same as what we have learned in the sections of `list` and `numpy`. In `.iloc()`, the first argument is the row index, and the second argument is the column index.

Dataframe

	C0	C1	C2	C3
	id	A	B	C
R0	0 a1	1	1	1
R1	1 a2	0	1	0
R2	2 a3	1	0	1

Row index

```
data.iloc[[1, 2], :]
```

```
   id  A  B  C
1  a2  0  1  0
2  a3  1  0  1
```

Column index

```
data.iloc[:, [0, 1]]
```

```
   id  A
0  a1  1
1  a2  0
2  a3  1
```

Column slicing

```
data.iloc[:, :2]
```

```
   id  A
0  a1  1
1  a2  0
2  a3  1
```

Pandas - Dataframe Manipulation

3.2 Label-based indexing (.loc)

The `.loc()` method is another way to access the data. It works with either column/index names or boolean arrays.

Dataframe

	C0	C1	C2	C3
	id	A	B	C
R0	0 a1	1	1	1
R1	1 a2	0	1	0
R2	2 a3	1	0	1

Row index

```
data.loc[[0, 1], :]
```

```
   id  A  B  C
0  a1  1  1  1
1  a2  0  1  0
```

Column index

```
data.loc[:, ['id', 'B']]
```

```
   id  B
0  a1  1
1  a2  1
2  a3  0
```

Boolean index

Use boolean to select column containing a letter "B". (We can use `df.columns` to list all column names)

```
colnames = data.columns
bol_B = ["B" in col for col in colnames]
print(bol_B)
```

Select column names that contain letter “B”

```
[False, False, True, False]
```

```
data.loc[:, bol_B]
```

```
   B
0  1
1  1
2  0
```

3.3 Create a new column

The `.loc()` method is also a recommended way (compared to `df["new_column"]`) to create a new column. Simply put a desired column name in the second argument, and assign a value to it.

```
data.loc[:, "new_col"] = ["new"] * 3
# or
data.loc[:, "new_col"] = "new"
data
```

```
   id  A  B  C new_col
0  a1  1  1  1     new
1  a2  0  1  0     new
2  a3  1  0  1     new
```

Pandas - Dataframe Manipulation

Dataframe

	id	A	B	C	new_col
0	a1	1	1	1	new
1	a2	0	1	0	new
2	a3	1	0	1	new

3.4.3 inspect the dimension and summary

`df.shape` returns the dimension of the dataframe. This tells us that the dataframe has 3 rows and 5 columns.

```
data.shape
```

```
(3, 5)
```

df.info() is another way to inspect the dataframe of its dimension and data types of each column.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   id       3 non-null      object
 1   A        3 non-null      int64 
 2   B        3 non-null      int64 
 3   C        3 non-null      int64 
 4   new_col  3 non-null      object
dtypes: int64(3), object(2)
memory usage: 248.0+ bytes
```

Data type

Pandas - Dataframe Manipulation

Dataframe

	id	A	B	C	new_col
0	a1	1	1	1	new
1	a2	0	1	0	new
2	a3	1	0	1	new

```
data.describe()
```

Only apply to numeric columns

	A	B	C
count	3.000000	3.000000	3.000000
mean	0.666667	0.666667	0.666667
std	0.577350	0.577350	0.577350
min	0.000000	0.000000	0.000000
25%	0.500000	0.500000	0.500000
50%	1.000000	1.000000	1.000000
75%	1.000000	1.000000	1.000000
max	1.000000	1.000000	1.000000

`df["column"].value_counts()` returns the counts of unique values in that specified column. Below the example tells us that there are two rows with value 1 and one row with value 0.

```
data["B"].value_counts()
```

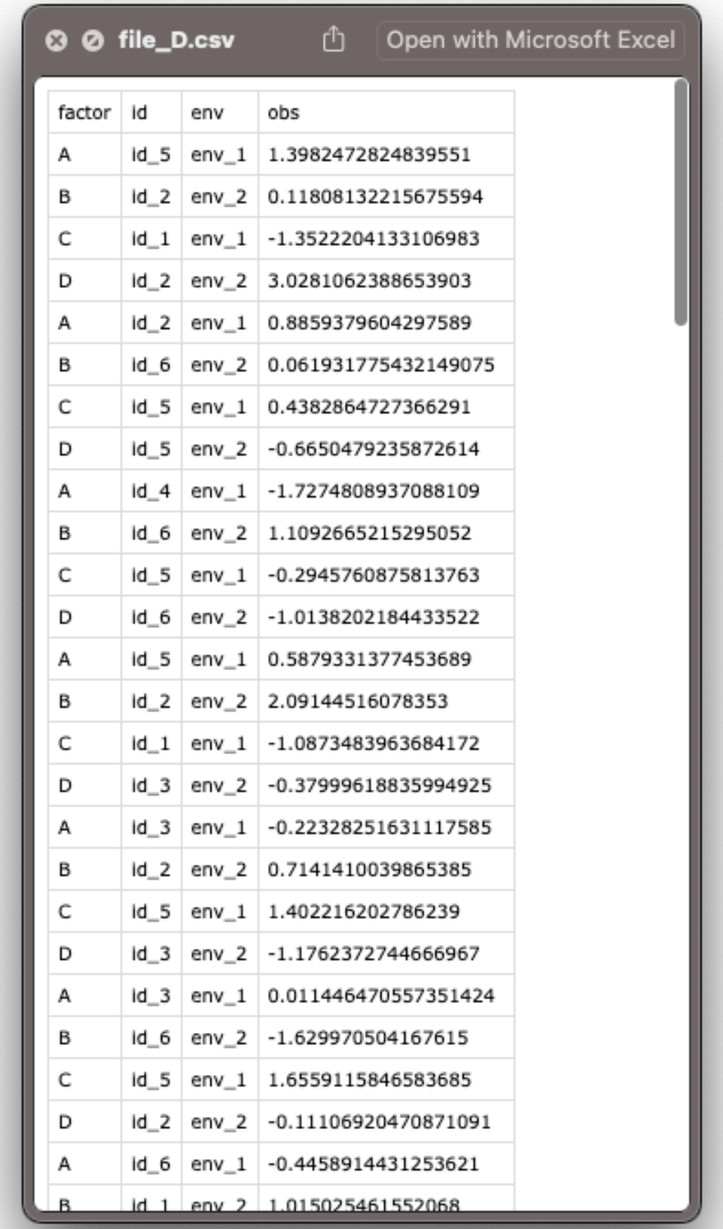
```
1    2
0    1
Name: B, dtype: int64
```

Pandas - Example Data Frame

Create a data frame with 120 rows

```
import numpy as np
import pandas as pd

factors = [i for _ in range(30) for i in ["A", "B", "C", "D"]]
# random sample from id {1, 2, 3, 4, 5, 6}
ids = np.random.choice(["id_%d" % (i + 1) for i in range(6)], 120)
envs = [i for _ in range(60) for i in ["env_1", "env_2"]]
obs = np.random.normal(0, 1, 120)
data = pd.DataFrame({"factor": factors, "id": ids, "env": envs, "obs":
obs})
data.to_csv("file_D.csv", index=False)
```



factor	id	env	obs
A	id_5	env_1	1.3982472824839551
B	id_2	env_2	0.11808132215675594
C	id_1	env_1	-1.3522204133106983
D	id_2	env_2	3.0281062388653903
A	id_2	env_1	0.8859379604297589
B	id_6	env_2	0.061931775432149075
C	id_5	env_1	0.4382864727366291
D	id_5	env_2	-0.6650479235872614
A	id_4	env_1	-1.7274808937088109
B	id_6	env_2	1.1092665215295052
C	id_5	env_1	-0.2945760875813763
D	id_6	env_2	-1.0138202184433522
A	id_5	env_1	0.5879331377453689
B	id_2	env_2	2.09144516078353
C	id_1	env_1	-1.0873483963684172
D	id_3	env_2	-0.37999618835994925
A	id_3	env_1	-0.22328251631117585
B	id_2	env_2	0.7141410039865385
C	id_5	env_1	1.402216202786239
D	id_3	env_2	-1.1762372744666967
A	id_3	env_1	0.011446470557351424
B	id_6	env_2	-1.629970504167615
C	id_5	env_1	1.6559115846583685
D	id_2	env_2	-0.11106920470871091
A	id_6	env_1	-0.4458914431253621
B	id_1	env_2	1.015025461552068

Pandas - Example Data Frame


```
data = pd.read_csv("file_D.csv")
data.info()
data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   factor  120 non-null    object
 1   id      120 non-null    object
 2   env     120 non-null    object
 3   obs     120 non-null    float64
dtypes: float64(1), object(3)
memory usage: 3.9+ KB
```

```
    factor    id    env       obs
0        A  id_5  env_1  1.398247
1        B  id_2  env_2  0.118081
2        C  id_1  env_1 -1.352220
3        D  id_2  env_2  3.028106
4        A  id_2  env_1  0.885938
..     ...   ...    ...       ...
115      D  id_3  env_2  0.352519
116      A  id_5  env_1 -1.363961
117      B  id_4  env_2 -1.148599
118      C  id_5  env_1 -0.769891
119      D  id_5  env_2  1.626178

[120 rows x 4 columns]
```

Nice interface!

file_D.csv — Open with Microsoft Excel

factor	id	env	obs
A	id_5	env_1	1.3982472824839551
B	id_2	env_2	0.11808132215675594
C	id_1	env_1	-1.3522204133106983
D	id_2	env_2	3.0281062388653903
A	id_2	env_1	0.8859379604297589
B	id_6	env_2	0.061931775432149075
C	id_5	env_1	0.4382864727366291
D	id_5	env_2	-0.6650479235872614
A	id_4	env_1	-1.7274808937088109
B	id_6	env_2	1.1092665215295052
C	id_5	env_1	-0.2945760875813763
D	id_6	env_2	-1.0138202184433522
A	id_5	env_1	0.5879331377453689
B	id_2	env_2	2.09144516078353
C	id_1	env_1	-1.0873483963684172
D	id_3	env_2	-0.37999618835994925
A	id_3	env_1	-0.22328251631117585
B	id_2	env_2	0.7141410039865385
C	id_5	env_1	1.402216202786239
D	id_3	env_2	-1.1762372744666967
A	id_3	env_1	0.011446470557351424
B	id_6	env_2	-1.629970504167615
C	id_5	env_1	1.6559115846583685
D	id_2	env_2	-0.11106920470871091
A	id_6	env_1	-0.4458914431253621
B	id_1	env_2	1.015025461552068

Pandas - Summary


```
data["factor"].value_counts()
```

```
A    30
B    30
C    30
D    30
Name: factor, dtype: int64
```

```
data["id"].value_counts()
```

```
id_5    32
id_6    24
id_3    20
id_2    19
id_1    15
id_4    10
Name: id, dtype: int64
```

```
data["env"].value_counts()
```

```
env_1    60
env_2    60
Name: env, dtype: int64
```

value_counts()

Not for continuous values

```
data["obs"].value_counts()
```

```
 1.398247    1
 0.118081    1
-0.250095    1
 1.214479    1
 1.006255    1
            ..
 0.645562    1
 0.397835    1
-0.784988    1
 0.732589    1
 1.626178    1
Name: obs, Length: 120, dtype: int64
```

describe()

Data distribution

```
data["obs"].describe()
```

```
count    120.000000
mean      -0.083406
std        1.053231
min       -2.408920
25%       -0.850797
50%       -0.147953
75%        0.602563
max        3.028106
Name: obs, dtype: float64
```

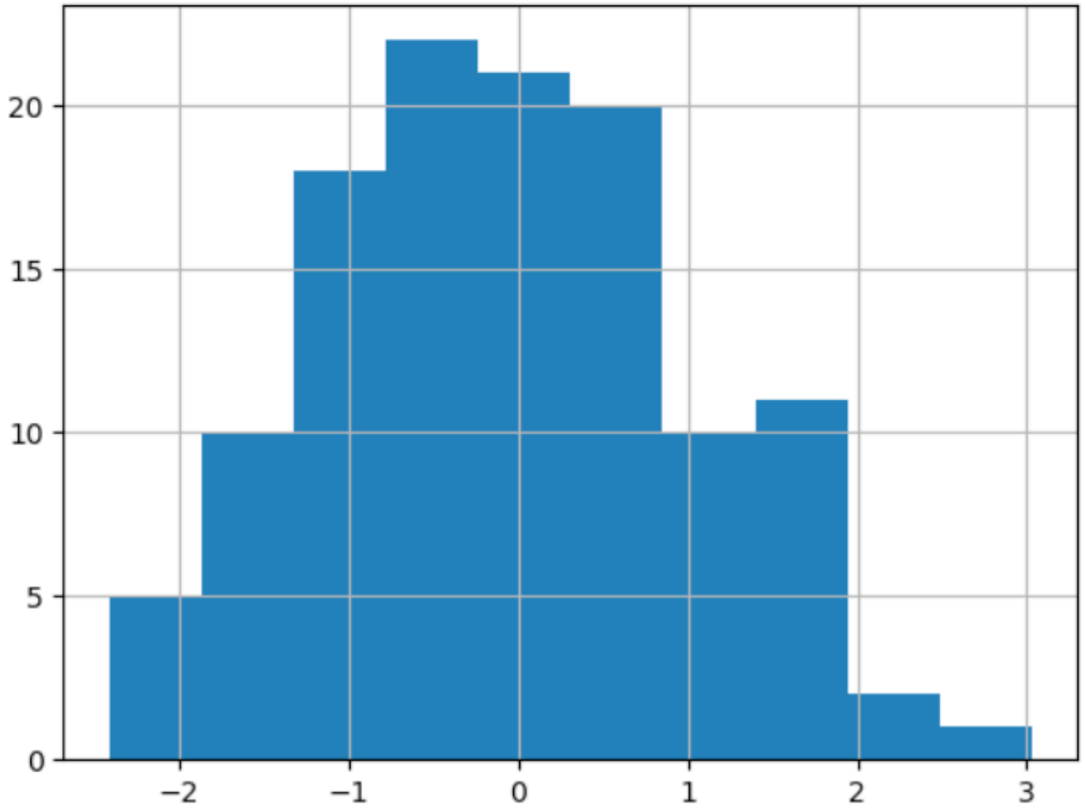
file_D.csv — Open with Microsoft Excel

factor	id	env	obs
A	Id_5	env_1	1.3982472824839551
B	Id_2	env_2	0.11808132215675594
C	Id_1	env_1	-1.3522204133106983
D	Id_2	env_2	3.0281062388653903
A	Id_2	env_1	0.8859379604297589
B	Id_6	env_2	0.061931775432149075
C	Id_5	env_1	0.4382864727366291
D	Id_5	env_2	-0.6650479235872614
A	Id_4	env_1	-1.7274808937088109
B	Id_6	env_2	1.1092665215295052
C	Id_5	env_1	-0.2945760875813763
D	Id_6	env_2	-1.0138202184433522
A	Id_5	env_1	0.5879331377453689
B	Id_2	env_2	2.09144516078353
C	Id_1	env_1	-1.0873483963684172
D	Id_3	env_2	-0.37999618835994925
A	Id_3	env_1	-0.22328251631117585
B	Id_2	env_2	0.7141410039865385
C	Id_5	env_1	1.402216202786239
D	Id_3	env_2	-1.1762372744666967
A	Id_3	env_1	0.011446470557351424
B	Id_6	env_2	-1.629970504167615
C	Id_5	env_1	1.6559115846583685
D	Id_2	env_2	-0.11106920470871091
A	Id_6	env_1	-0.4458914431253621
B	Id_1	env_2	1.015025461552068

Pandas - Visualization

For better visualization, we can use `df.hist()` to plot the histogram of each column.

```
data["obs"].hist()
```



describe() Data distribution

```
data["obs"].describe()
```

```
count    120.000000
mean      -0.083406
std        1.053231
min       -2.408920
25%       -0.850797
50%       -0.147953
75%        0.602563
max        3.028106
Name: obs, dtype: float64
```

Pandas - Query

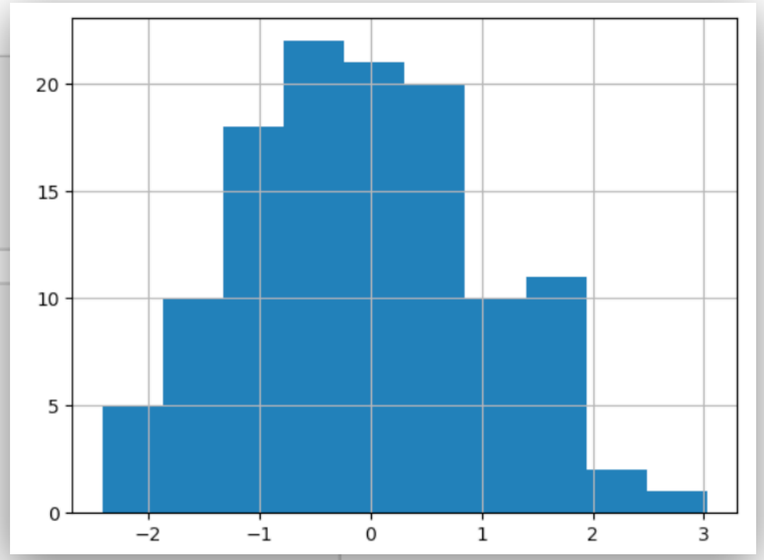
4.2 Subset the dataframe (query)

```
data_sub = data.query("obs > 0")
data_sub[:5]
```

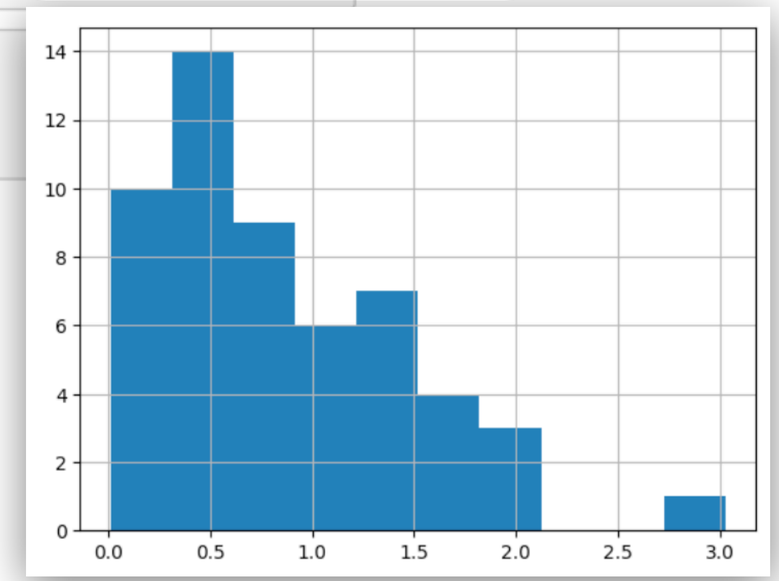
	factor	id	env	obs
0	A	id_5	env_1	1.398247
1	B	id_2	env_2	0.118081
3	D	id_2	env_2	3.028106
4	A	id_2	env_1	0.885938
5	B	id_6	env_2	0.061932

```
data_sub["obs"].hist()
```

Before the query



After the query



Pandas - Query

4.2 Subset the dataframe (query)

Numeric query

```
data_sub = data.query("obs > 0")
data_sub[:5]
```

	factor	id	env	obs
0	A	id_5	env_1	1.398247
1	B	id_2	env_2	0.118081
3	D	id_2	env_2	3.028106
4	A	id_2	env_1	0.885938
5	B	id_6	env_2	0.061932

String query

```
data_id1 = data.query("id == 'id_1'")
data_id1[:5]
```

	factor	id	env	obs
2	C	id_1	env_1	-1.352220
14	C	id_1	env_1	-1.087348
25	B	id_1	env_2	1.015025
35	D	id_1	env_2	0.645562
37	B	id_1	env_2	-0.216822

Pandas - Query

Logic operators

Query 1 Query 2

```
data.query("id == 'id_1' and (obs > 1 or obs < -1)")
```

```
   factor    id    env       obs
2       C  id_1  env_1 -1.352220
14      C  id_1  env_1 -1.087348
25      B  id_1  env_2  1.015025
77      B  id_1  env_2  1.927947
79      D  id_1  env_2 -1.319636
97      B  id_1  env_2  1.051139
```

Pandas - Grouping

Values

Group Function

```
data.groupby("id")["obs"].mean()
```

```
id
id_1    0.011577
id_2    0.072730
id_3   -0.409600
id_4   -0.231937
id_5   -0.036769
id_6    0.005157
Name: obs, dtype: float64
```

Multi-group

```
data.groupby(["id", "factor"])["obs"].mean()
```

```
id    factor
id_1  A        -0.435927
      B         0.835641
      C        -0.620353
      D        -0.116713
id_2  A         0.457923
      B         0.196333
      C        -0.981377
      D        -0.064095
id_3  A        -0.979025
      B        -0.527586
      C         0.002107
      D        -0.170709
id_4  A        -1.287146
      B        -1.274956
      C         0.183495
      D         1.237600
id_5  A         0.021462
      B        -0.337031
      C        -0.018164
      D         0.073601
id_6  A        -0.132296
      B         0.030516
      C         0.297709
      D        -0.326848
Name: obs, dtype: float64
```

file_D.csv — Open with Microsoft Excel

factor	id	env	obs
A	id_5	env_1	1.3982472824839551
B	id_2	env_2	0.11808132215675594
C	id_1	env_1	-1.3522204133106983
D	id_2	env_2	3.0281062388653903
A	id_2	env_1	0.8859379604297589
B	id_6	env_2	0.061931775432149075
C	id_5	env_1	0.4382864727366291
D	id_5	env_2	-0.6650479235872614
A	id_4	env_1	-1.7274808937088109
B	id_6	env_2	1.1092665215295052
C	id_5	env_1	-0.2945760875813763
D	id_6	env_2	-1.0138202184433522
A	id_5	env_1	0.5879331377453689
B	id_2	env_2	2.09144516078353
C	id_1	env_1	-1.0873483963684172
D	id_3	env_2	-0.37999618835994925
A	id_3	env_1	-0.22328251631117585
B	id_2	env_2	0.7141410039865385
C	id_5	env_1	1.402216202786239
D	id_3	env_2	-1.1762372744666967
A	id_3	env_1	0.011446470557351424
B	id_6	env_2	-1.629970504167615
C	id_5	env_1	1.6559115846583685
D	id_2	env_2	-0.11106920470871091
A	id_6	env_1	-0.4458914431253621
B	id_1	env_2	1.015025461552068

Pandas - Grouping

Custom calculation

```
# multiple calculation
cus_fun = lambda x: x.max() - x.min()
pivot = data.groupby(["id", "factor"])["obs"].agg(["mean", "std", "count",
cus_fun])
```

		mean	std	count	<lambda_0>
id	factor				
id_1	A	-0.435927	NaN	1	0.000000
	B	0.835641	0.801314	5	2.144769
	C	-0.620353	0.578104	5	1.369322
	D	-0.116713	0.847288	4	1.965199
id_2	A	0.457923	0.490873	6	1.238011
	B	0.196333	1.385810	5	3.800971
	C	-0.981377	0.786843	2	1.112764
	D	-0.064095	1.672929	6	4.749550
id_3	A	-0.979025	0.958212	5	2.420366
	B	-0.527586	1.745133	4	3.964272
	C	0.002107	1.062830	4	2.192775
	D	-0.170709	0.616826	7	1.659665
id_4	A	-1.287146	0.622728	2	0.880671
	B	-1.274956	0.826101	3	1.637642
	C	0.183495	0.207320	2	0.293195
	D	1.237600	0.902724	3	1.747751
id_5	A	0.021462	0.901412	9	2.762209
	B	-0.337031	1.111385	5	2.792467
	C	-0.018164	0.955649	11	2.789181
	D	0.073601	1.016560	7	2.758430
id_6	A	-0.132296	0.960805	7	2.785278
	B	0.030516	1.099071	8	3.073730
	C	0.297709	1.135733	6	3.230495
	D	-0.326848	1.506402	3	2.767885

```
pivot.loc["id_5"]
```

	mean	std	count	<lambda_0>
factor				
A	0.021462	0.901412	9	2.762209
B	-0.337031	1.111385	5	2.792467
C	-0.018164	0.955649	11	2.789181
D	0.073601	1.016560	7	2.758430

```
pivot.loc["id_3"].loc["A"]
```

```
mean          -0.979025
std            0.958212
count          5.000000
<lambda_0>     2.420366
Name: A, dtype: float64
```

Pandas - Grouping

id	factor	mean	std	count	<lambda_0>
id_1	A	-0.435927	NaN	1	0.000000
	B	0.835641	0.801314	5	2.144769
	C	-0.620353	0.578104	5	1.369322
	D	-0.116713	0.847288	4	1.965199
id_2	A	0.457923	0.490873	6	1.238011
	B	0.196333	1.385810	5	3.800971
	C	-0.981377	0.786843	2	1.112764
	D	-0.064095	1.672929	6	4.749550
id_3	A	-0.979025	0.958212	5	2.420366
	B	-0.527586	1.745133	4	3.964272
	C	0.002107	1.062830	4	2.192775
	D	-0.170709	0.616826	7	1.659665
id_4	A	-1.287146	0.622728	2	0.880671
	B	-1.274956	0.826101	3	1.637642
	C	0.183495	0.207320	2	0.293195
	D	1.237600	0.902724	3	1.747751
id_5	A	0.021462	0.901412	9	2.762209
	B	-0.337031	1.111385	5	2.792467
	C	-0.018164	0.955649	11	2.789181
	D	0.073601	1.016560	7	2.758430
id_6	A	-0.132296	0.960805	7	2.785278
	B	0.030516	1.099071	8	3.073730
	C	0.297709	1.135733	6	3.230495
	D	-0.326848	1.506402	3	2.767885

Index to column

reset_index()

```
data_pivot = pivot.reset_index()
data_pivot
```

	id	factor	mean	std	count	<lambda_0>
0	id_1	A	-0.435927	NaN	1	0.000000
1	id_1	B	0.835641	0.801314	5	2.144769
2	id_1	C	-0.620353	0.578104	5	1.369322
3	id_1	D	-0.116713	0.847288	4	1.965199
4	id_2	A	0.457923	0.490873	6	1.238011
5	id_2	B	0.196333	1.385810	5	3.800971
6	id_2	C	-0.981377	0.786843	2	1.112764
7	id_2	D	-0.064095	1.672929	6	4.749550
8	id_3	A	-0.979025	0.958212	5	2.420366
9	id_3	B	-0.527586	1.745133	4	3.964272
10	id_3	C	0.002107	1.062830	4	2.192775
11	id_3	D	-0.170709	0.616826	7	1.659665
12	id_4	A	-1.287146	0.622728	2	0.880671
13	id_4	B	-1.274956	0.826101	3	1.637642
14	id_4	C	0.183495	0.207320	2	0.293195
15	id_4	D	1.237600	0.902724	3	1.747751
16	id_5	A	0.021462	0.901412	9	2.762209
17	id_5	B	-0.337031	1.111385	5	2.792467
18	id_5	C	-0.018164	0.955649	11	2.789181
19	id_5	D	0.073601	1.016560	7	2.758430
20	id_6	A	-0.132296	0.960805	7	2.785278
21	id_6	B	0.030516	1.099071	8	3.073730
22	id_6	C	0.297709	1.135733	6	3.230495
23	id_6	D	-0.326848	1.506402	3	2.767885