



PYTHON FOR R USERS

Selecting data in pandas

Daniel Chen
Instructor



Manually Create DataFrame

```
df = pd.DataFrame({  
    'A': [1, 2, 3],  
    'B': [4, 5, 6],  
    'C': [7, 8, 9]},  
    index = ['x', 'y', 'z'])
```

```
print(df)
```

	A	B	C
x	1	4	7
y	2	5	8
z	3	6	9



Subsetting Columns

```
In [11]: df = pd.DataFrame({  
    ....:     'A': [1, 2, 3],  
    ....:     'B': [4, 5, 6],  
    ....:     'C': [7, 8, 9]},  
    ....:     index = ['x', 'y', 'z'])
```

```
In [12]: df
```

```
Out[12]:
```

	A	B	C
x	1	4	7
y	2	5	8
z	3	6	9

```
In [13]: df['A']
```

```
Out[13]:
```

x	1
y	2
z	3

```
Name: A, dtype: int64
```

```
In [14]: df.A
```

```
Out[14]:
```

x	1
y	2
z	3

```
Name: A, dtype: int64
```

```
In [15]: df[['A', 'B']]
```

```
Out[15]:
```

	A	B
x	1	4
y	2	5
z	3	6



Subsetting Rows

- Row-label (loc) vs row-index (iloc)
- Python starts counting from 0



Subsetting Rows .iloc

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

```
Out[16]:
```

	A	B	C
x	1	4	7
y	2	5	8
z	3	6	9

```
In [17]: df.iloc[0]
```

```
Out[17]:
```

A	1
B	4
C	7

```
Name: x, dtype: int64
```

```
In [18]: df.iloc[[0, 1]]
```

```
Out[18]:
```

	A	B	C
x	1	4	7
y	2	5	8

```
In [19]: df.iloc[0, :]
```

```
Out[19]:
```

A	1
B	4
C	7

```
Name: x, dtype: int64
```

```
In [20]: df.iloc[[0, 1], :]
```

```
Out[20]:
```

	A	B	C
x	1	4	7
y	2	5	8



Subsetting Rows .loc

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

```
Out[16]:
```

	A	B	C
x	1	4	7
y	2	5	8
z	3	6	9

```
In [23]: df.loc['x']
```

```
Out[23]:
```

A	1
B	4
C	7

Name: x, dtype: int64

```
In [24]: df.loc[['x', 'y']]
```

```
Out[24]:
```

	A	B	C
x	1	4	7
y	2	5	8



Multiple rows and columns

```
In [27]: df
```

```
Out[27]:
```

	A	B	C
x	1	4	7
y	2	5	8
z	3	6	9

```
In [28]: df.loc['x', 'A']
```

```
Out[28]: 1
```

```
In [29]: df.loc[['x', 'y'], ['A', 'B']]
```

```
Out[29]:
```

	A	B
x	1	4
y	2	5



Conditional Subsetting

```
In [30]: df[df.A == 3]
```

```
Out[30]:
```

	A	B	C
z	3	6	9

```
In [31]: df[(df.A == 3) | (df.B == 4)]
```

```
Out[31]:
```

	A	B	C
x	1	4	7
z	3	6	9



Attributes

```
In [16]: df.shape  
Out[16]: (3, 2)
```

```
In [17]: df.shape()
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-17-0e566b70f572> in <module>()  
----> 1 df.shape()
```

```
TypeError: 'tuple' object is not callable
```



PYTHON FOR R USERS

Let's practice!



PYTHON FOR R USERS

datatypes

Daniel Chen
Instructor

Type

R

```
> df <- data.frame(
  'A' = c(1, 2, 3),
  'B' = c(4, 5, 6)
)

> df
  A B
1 1 4
2 2 5
3 3 6

> class(df)
[1] "data.frame"
```

PYTHON

```
In [33]: import pandas as pd
        ...: df = pd.DataFrame(
            {'A': [1, 2, 3],
             'B': [4, 5, 6]})

In [34]: df
Out[34]:
   A  B
0  1  4
1  2  5
2  3  6

In [35]: type(df)
Out[35]: pandas.core.frame.DataFrame
```



Info Method

R

```
> str(df)
'data.frame':   3 obs. of  2 variables:
 $ A: num  1 2 3
 $ B: num  4 5 6
```

PYTHON

```
In [36]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 2 columns):
A      3 non-null int64
B      3 non-null int64
dtypes: int64(2)
memory usage: 128.0 bytes
```



Convert to string

R

```
> df$A <- as.character(df$A)
> str(df)
'data.frame':   3 obs. of  2 variables:
 $ A: chr  "1" "2" "3"
 $ B: num  4 5 6
```

PYTHON

```
In [39]: df['A'] = df['A'].astype(str)
In [40]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 2 columns):
A      3 non-null object
B      3 non-null int64
dtypes: int64(1), object(1)
memory usage: 128.0+ bytes
```



String Objects

```
In [40]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 2 columns):
A      3 non-null object
B      3 non-null int64
dtypes: int64(1), object(1)
memory usage: 128.0+ bytes
```

- When you see "object" it is a string
- Access built-in string methods with str accessor



String Accessor

```
In [5]: df = pd.DataFrame({'name': ['Daniel ', ' Eric', ' Julia ']})
```

```
In [6]: df
```

```
Out[6]:
```

	name
0	Daniel
1	Eric
2	Julia

```
In [7]: df['name_strip'] = df['name'].str.strip()
```

```
In [8]: df
```

```
Out[8]:
```

	name	name_strip
0	Daniel	Daniel
1	Eric	Eric
2	Julia	Julia



Category

```
In [2]: df = pd.DataFrame({'name': ['Daniel', 'Eric', 'Julia'],  
    ....:                  'gender': ['Male', 'Male', 'Female']})
```

```
In [3]: df.dtypes
```

```
Out[3]:
```

```
gender      object
```

```
name        object
```

```
dtype: object
```

```
In [4]: df['gender_cat'] = df['gender'].astype('category')
```

```
In [5]: df.dtypes
```

```
Out[5]:
```

```
gender      object
```

```
name        object
```

```
gender_cat  category
```

```
dtype: object
```



Category Accessor

```
In [7]: df['gender_cat'].cat.categories
Out[7]: Index(['Female', 'Male'], dtype='object')

In [6]: df.gender_cat.cat.codes
Out[6]:
0      1
1      1
2      0
dtype: int8
```



Datetime

```
In [16]: df = pd.DataFrame({'name': ['Rosaline Franklin', 'William Gosset'],  
    ....:                  'born': ['1920-07-25', '1876-06-13']})  
    ....:
```

```
In [18]: df['born_dt'] = pd.to_datetime(df['born'])
```

```
In [19]: df
```

```
Out[19]:
```

	born	name	born_dt
0	1920-07-25	Rosaline Franklin	1920-07-25
1	1876-06-13	William Gosset	1876-06-13

```
In [20]: df.dtypes
```

```
Out[20]:
```

born	object
name	object
born_dt	datetime64[ns]
dtype:	object

Datetime Accessor

```
In [21]: df['born_dt'].dt.day
```

```
Out[21]:
```

```
0    25
```

```
1    13
```

```
Name: born_dt, dtype: int64
```

```
In [22]: df['born_dt'].dt.month
```

```
Out[22]:
```

```
0     7
```

```
1     6
```

```
Name: born_dt, dtype: int64
```

```
In [23]: df['born_dt'].dt.year
```

```
Out[23]:
```

```
0    1920
```

```
1    1876
```

```
Name: born_dt, dtype: int64
```



PYTHON FOR R USERS

Let's practice!



PYTHON FOR R USERS

More Pandas

Daniel Chen
Instructor



Missing data

- NaN missing values from from numpy
- `np.NaN`, `np.NAN`, `np.nan` are all the same as the NA R value
- check missing with `pd.isnull`
 - Check non-missing with `pd.notnull`
 - `pd.isnull` is an alias for `pd.isna`



Working with missing data

```
In [30]: df
```

```
Out[30]:
```

	name	treatment_a	treatment_b
0	John Smith	NaN	2
1	Jane Doe	16.0	11
2	Mary Johnson	3.0	1

```
In [13]: a_mean = df['treatment_a'].mean()
```

```
In [14]: a_mean
```

```
Out[14]: 9.5
```




fillna

```
In [19]: df['a_fill'] = df['treatment_a'].fillna(a_mean)
```

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

```
Out[20]:
```

	name	treatment_a	treatment_b	a_fill
0	John Smith	NaN	2	9.5
1	Jane Doe	16.0	11	16.0
2	Mary Johnson	3.0	1	3.0



More Pandas

- Applying custom functions
- Groupby operations
- Tidying data



Apply your own functions

- Built-in functions
- Custom functions
- apply method
- Pass in an axis

Apply functions on DataFrames

R

```
> df = data.frame('a' = c(1, 2, 3),  
                  'b' = c(4, 5, 6))
```

```
> apply(df, 2, mean)  
a b  
2 5
```

```
> apply(df, 1, mean)  
[1] 2.5 3.5 4.5
```

PYTHON

```
In [19]: import pandas as pd  
        ...: df = pd.DataFrame(  
            {'A': [1, 2, 3],  
             'B': [4, 5, 6]}  
        )
```

```
        ...:  
In [20]: df.apply(np.mean, axis=0)  
Out[20]:  
A    2.0  
B    5.0  
dtype: float64
```

```
In [21]: df.apply(np.mean, axis=1)  
Out[21]:  
0    2.5  
1    3.5  
2    4.5  
dtype: float64
```



Tidy

- Reshaping and tidying our data
- Hadley Wickham, Tidy Data Paper
 - Each row is an observation
 - Each column is a variable
 - Each type of observational unit forms a table

Tidy Data Paper: <http://vita.had.co.nz/papers/tidy-data.pdf>



Tidy Melt

```
In [48]: df
```

```
Out[48]:
```

	name	treatment_a	treatment_b
0	John Smith	NaN	2
1	Jane Doe	16.0	11
2	Mary Johnson	3.0	1

```
In [50]: df_melt = pd.melt(df, id_vars='name')
```

```
In [51]: df_melt
```

```
Out[51]:
```

	name	variable	value
0	John Smith	treatment_a	NaN
1	Jane Doe	treatment_a	16.0
2	Mary Johnson	treatment_a	3.0
3	John Smith	treatment_b	2.0
4	Jane Doe	treatment_b	11.0
5	Mary Johnson	treatment_b	1.0



Tidy Pivot_table

```
In [53]: df_melt_pivot = pd.pivot_table(df_melt,  
....:                                index='name',  
....:                                columns='variable',  
....:                                values='value')  
....: df_melt_pivot
```

```
Out[53]:  
variable      treatment_a  treatment_b  
name  
Jane Doe           16.0           11.0  
John Smith          NaN            2.0  
Mary Johnson        3.0            1.0
```



Reset Index

```
In [54]: df_melt_pivot.reset_index()
```

```
Out[54]:
```

	variable	name	treatment_a	treatment_b
0		Jane Doe	16.0	11.0
1		John Smith	NaN	2.0
2		Mary Johnson	3.0	1.0



Groupby

- groupby: split-apply-combine
- split data into separate partitions
- apply a function on each partition
- combine the results



Performing a groupby

```
Out[43]:
```

	name	variable	value
0	John Smith	treatment_a	NaN
1	Jane Doe	treatment_a	16.0
2	Mary Johnson	treatment_a	3.0
3	John Smith	treatment_b	2.0
4	Jane Doe	treatment_b	11.0
5	Mary Johnson	treatment_b	1.0

```
In [46]: df_melt.groupby('name')['value'].mean()
```

```
Out[46]:
```

```
name
Jane Doe      13.5
John Smith     2.0
Mary Johnson   2.0
Name: value, dtype: float64
```



PYTHON FOR R USERS

Let's practice!