

CS167: Machine Learning

Pandas Tutorial:
Subsetting (Columns, Rows, or both) in a DataFrame
Useful Functions

Thursday, February 8th, 2024



Announcement



HCML LAB @ DRAKE

PARTICIPATE IN RESEARCH

Participate in research that uses **Brain-Computer Interfaces** to help people with **ADHD** focus.

We are looking for both neurotypical and neurodivergent participants.

Interested?

Sign Up Here -->

Questions?

meredith.moore@drake.edu



Announcement

- If you are interested in participating fill out the following form
 - Google form: <https://forms.gle/w4j2gQyjMdQHkEK68>

Recap: Overview of Pandas Tutorial

- Overview of Pandas
 - Datatypes `DataFrame` and `Series`
 - helpful functions
- Other goals are as follows:
 - Select `columns` in DataFrames
 - Select `rows` in DataFrames
 - Select `subsets` of the DataFrame (both rows and columns)

Recap: Pandas Datatypes (DataFrame)

- [Pandas Documentation](#) defines `DataFrames` as:
 - *'Two-dimensional, size-mutable, potentially heterogeneous tabular data'*
 - basically, think of `DataFrames` as our excel sheets--two dimensional, tabular data
 - Each column has a name, and you can use these names to filter and create subsets of data
 - often, you'll see `DataFrames` abbreviated to `df`

Recap: Creating DataFrame

- The syntax for creating a `DataFrame` from scratch looks like this: `pandas.DataFrame(data, index, columns)`

```
▶ df = pd.DataFrame() # creates an empty DataFrame
print(df)
```

```
Empty DataFrame
Columns: []
Index: []
```

```
▶ data = {'col1':[1,2,3], 'col2':[4,5,6], 'col3':[7,8,9]}
df_3 = pd.DataFrame(data)
print('size of the dataframe df_2', df_2.shape)
df_3
```

```
↳ size of the dataframe df_2 (3, 3)
```

	col1	col2	col3
0	1	4	7
1	2	5	8
2	3	6	9

```
[15] data = [10, 20, 30, 40, 50, 60]
df_1 = pd.DataFrame(data, columns=['numbers'])
print('size of the dataframe df_2', df_1.shape)
df_1
```

```
size of the dataframe df_2 (6, 1)
```

	numbers
0	10
1	20
2	30
3	40
4	50
5	60

Recap: Creating DataFrame from a .csv file

- To access this file in Google Colab, you'll need a little bit of code.

```
[ ] # The first step is to mount your Google Drive to your Colab account.  
#You will be asked to authorize Colab to access your Google Drive. Follow the steps they lead you through.  
  
#this will only work in Google Colab.  
  
from google.colab import drive  
drive.mount('/content/drive')
```

- You will be able to show the path of `restaurant.csv` on your Google Drive as follows:

```
0s [▶] #change this path to point to where your data is:  
# if you're using colab it should be something like: '/content/drive/MyDrive/CS167/datasets/restaurant.csv'  
  
import pandas as pd  
path = '/content/drive/MyDrive/cs167_fall23/datasets/restaurant.csv'  
  
restaurant_data = pd.read_csv(path)  
print('data is a ', type(restaurant_data))
```

```
data is a <class 'pandas.core.frame.DataFrame'>
```

Recap: Creating DataFrame from a .csv file

- To access this file in Google Colab, you'll need a little bit of code.

```
[ ] # The first step is to mount your Google Drive to your Colab account.  
#You will be asked to authorize Colab to access your Google Drive. Follow the steps they lead you through.  
  
#this will only work in Google Colab.  
  
from google.colab import drive  
drive.mount('/content/drive')
```

- You will be able to show the path of `restaurant.csv` on your Google Drive as follows:

```
0s [▶] #change this path to point to where your data is:  
# if you're using colab it should be something like: '/content/drive/MyDrive/CS167/datasets/restaurant.csv'  
  
import pandas as pd  
path = '/content/drive/MyDrive/cs167_fall23/datasets/restaurant.csv'  
  
restaurant_data = pd.read_csv(path)  
print('data is a ', type(restaurant_data))
```

```
data is a <class 'pandas.core.frame.DataFrame'>
```


Recap: Helpful Method: `df.head()`

- The `.head()` method can be called on any `DataFrame`, and by default will display the first 5 lines/rows of the data, as well as the names of the columns.
 - if you want it to display more than 5 rows, you can provide a number as an argument to the method.

```
[20] #change this path to point to where your data is:
      # if you're using colab it should be something like below:
      path = '/content/drive/MyDrive/cs167_fall23/datasets/restaurant.csv'

      # read the data from the csv file
      df_4 = pd.read_csv(path)

      # show the dataframe
      df_4.head()
```

	alt	bar	fri	hun	pat	price	rain	res	type	est	target
0	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0-10	Yes
1	Yes	No	No	Yes	Full	\$	No	No	Thai	30-60	No
2	No	Yes	No	No	Some	\$	No	No	Burger	0-10	Yes
3	Yes	No	Yes	Yes	Full	\$	No	No	Thai	10-30	Yes
4	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	No



Recap: Selecting Rows in DataFrames using `loc` and `iloc`:


- Simply put:
 - `loc` gets DataFrame rows and columns by **labels/names**
 - `iloc` gets DataFrame rows and columns by **index/position**

Recap: Selecting Rows using loc

- `loc` gets DataFrame rows and columns by labels/names
- Let's take a subset of titanic and try to use `loc` and `iloc`:

```
subset = df_titanic.loc[800:805]  
subset.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
800	0	2	male	34.00	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
801	1	2	female	31.00	1	1	26.2500	S	Second	woman	False	NaN	Southampton	yes	False
802	1	1	male	11.00	1	2	120.0000	S	First	child	False	B	Southampton	yes	False
803	1	3	male	0.42	0	1	8.5167	C	Third	child	False	NaN	Cherbourg	yes	False
804	1	3	male	27.00	0	0	6.9750	S	Third	man	True	NaN	Southampton	yes	True



labels/names

ALERT: `df.head()` only shows the first 5 rows

Recap: Selecting Rows using iLoc

- `iLoc` gets DataFrame rows and columns by index/position

```
subset = df_titanic.loc[800:805]
subset.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
800	0	2	male	34.00	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
801	1	2	female	31.00	1	1	26.2500	S	Second	woman	False	NaN	Southampton	yes	False
802	1	1	male	11.00	1	2	120.0000	S	First	child	False	B	Southampton	yes	False
803	1	3	male	0.42	0	1	8.5167	C	Third	child	False	NaN	Cherbourg	yes	False
804	1	3	male	27.00	0	0	6.9750	S	Third	man	True	NaN	Southampton	yes	True

```
subset.iloc[0] #works
```

```
survived      0
pclass        2
sex           male
age          34.0
sibsp         0
parch         0
fare          13.0
embarked      S
class         Second
who           man
adult_male    True
deck          NaN
embark_town   Southampton
alive         no
alone         True
Name: 800, dtype: object
```

```
subset.iloc[1] #works
```

```
survived      1
pclass        2
sex           female
age          31.0
sibsp         1
parch         1
fare          26.25
embarked      S
class         Second
who           woman
adult_male    False
deck          NaN
embark_town   Southampton
alive         yes
alone         False
Name: 801, dtype: object
```

```
subset.iloc[5] #works
```

```
survived      0
pclass        3
sex           male
age          31.0
sibsp         0
parch         0
fare          7.775
embarked      S
class         Third
who           man
adult_male    True
deck          NaN
embark_town   Southampton
alive         no
alone         True
Name: 805, dtype: object
```

Recap: Subsetting Columns

- Why might we want a subset of the columns of a DataFrame?



Recap: Subsetting Columns

- So, if we wanted to look at the **price** column, we could do:

```
▶ import pandas as pd
path = '/content/drive/MyDrive/cs167_fall23/datasets/restaurant.csv'
restaurant_data = pd.read_csv(path)
print('data is a ', type(restaurant_data))
restaurant_data.head()
```

data is a <class 'pandas.core.frame.DataFrame'>

	alt	bar	fri	hun	pat	price	rain	res	type	est	target
0	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0-10	Yes
1	Yes	No	No	Yes	Full	\$	No	No	Thai	30-60	No
2	No	Yes	No	No	Some	\$	No	No	Burger	0-10	Yes
3	Yes	No	Yes	Yes	Full	\$	No	No	Thai	10-30	Yes
4	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	No

```
▶ prices = restaurant_data['price']
prices
```

Recap: Subsetting Columns

- Imagine you want to only work with 'rain', 'hun', 'target'

	alt	bar	fri	hun	pat	price	rain	res	type	est	target
0	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0-10	Yes
1	Yes	No	No	Yes	Full	\$	No	No	Thai	30-60	No
2	No	Yes	No	No	Some	\$	No	No	Burger	0-10	Yes
3	Yes	No	Yes	Yes	Full	\$	No	No	Thai	10-30	Yes
4	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	No



```
▶ col_subset = restaurant_data[['rain', 'hun', 'target']]  
col_subset.head()
```

	rain	hun	target
0	No	Yes	Yes
1	No	Yes	No
2	No	No	Yes
3	No	Yes	Yes
4	No	No	No



New Material

- Subsetting rows and columns:
 - Select **rows** in DataFrames
 - Select **subsets** of the DataFrame (both rows and columns)

Subsetting Rows in a DataFrame

- Why might you want a subset of the rows?



- Maybe you want only rows that satisfy a certain condition--in the restaurant dataset, maybe:
 - Italian Restaurants
 - only examples when it didn't rain
 - etc.

Subsetting Rows in a DataFrame

- To understand the syntax for subsetting rows in a DataFrame, we need to understand how conditionals work in Python/Pandas:
 - to check whether each row in a DataFrame meets a criteria, use the following syntax
 - it will return a Series with **True/False**, where rows that are **True** meet the criteria, and **False** do not

```
▶ restaurant_data['type'] == 'French'
```

Subsetting Rows in a DataFrame

```
restaurant_data.head()
```

```
data is a <class 'pandas.core.frame.DataFrame'>
```

	alt	bar	fri	hun	pat	price	rain	res	type	est	target
0	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0-10	Yes
1	Yes	No	No	Yes	Full	\$	No	No	Thai	30-60	No
2	No	Yes	No	No	Some	\$	No	No	Burger	0-10	Yes
3	Yes	No	Yes	Yes	Full	\$	No	No	Thai	10-30	Yes
4	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	No

```
▶ restaurant_data['type'] == 'French'
```

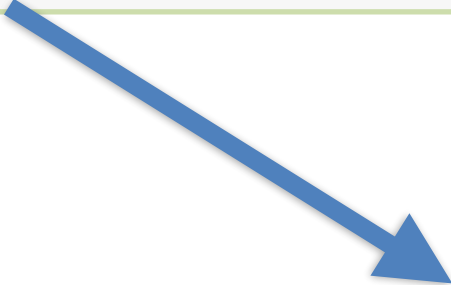
```
0      True
1     False
2     False
3     False
4      True
5     False
6     False
7     False
8     False
9     False
10    False
11    False
```

Subsetting Rows in a DataFrame

- Taking this one step further, we can use this boolean Series to filter our rows:

- `condition = df['column name'] == 'something'`
- `subset_rows = df[condition]`

```
▶ # the conditional from a few slides ago was:  
condition = restaurant_data['target'] == 'No'  
  
italian_rest = restaurant_data[condition]  
italian_rest
```



	alt	bar	fri	hun	pat	price	rain	res	type	est	target
1	Yes	No	No	Yes	Full	\$	No	No	Thai	30-60	No
4	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	No
6	No	Yes	No	No	None	\$	Yes	No	Burger	0-10	No
8	No	Yes	Yes	No	Full	\$	Yes	No	Burger	>60	No
9	Yes	Yes	Yes	Yes	Full	\$\$\$	No	Yes	Italian	10-30	No
10	No	No	No	No	None	\$	No	No	Thai	0-10	No

Subsetting Rows in a DataFrame

- It can be done in one step as follows:
 - `subset_rows = df[df['column name'] == 'something']`

```
▶ # or in one step:  
italian_restaurants = restaurant_data[ restaurant_data['type'] == 'Italian' ]  
italian_restaurants
```



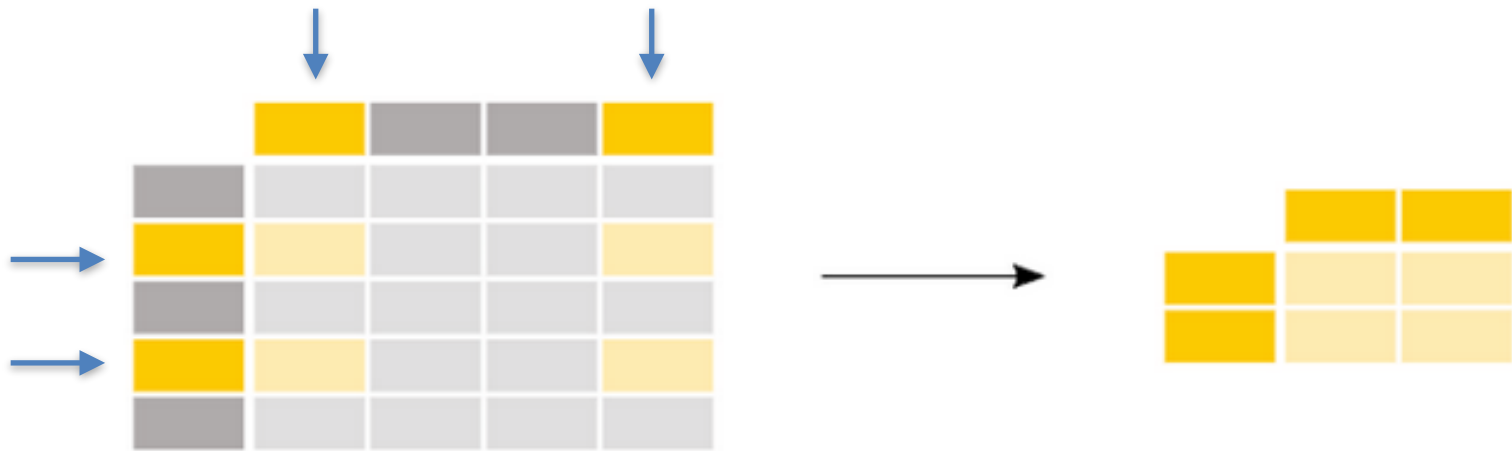
	alt	bar	fri	hun	pat	price	rain	res	type	est	target	
5	No	Yes	No	Yes	Some	\$\$	Yes	Yes	Italian	0-10	Yes	
9	Yes	Yes	Yes	Yes	Full	\$\$\$	No	Yes	Italian	10-30	No	

Group Exercise

- See if you can create a subset called `rainy_day`, of rows where it rained from the dataset `'restaurant.csv'`

Subsetting Columns and Rows

- Let's imagine we want a subset that contains the ages of people who did not survive the Titanic. Technically, you have the knowledge now to be able to do this, if you just break it up into two steps
 - make a subset, `victims`, of rows where `survived == 0`
 - use `victims` to create a second subset that only contains the 'Age' column.



Subsetting Columns and Rows


- Let's imagine we want a subset that contains the ages of people who did not survive the Titanic. Technically, you have the knowledge now to be able to do this, if you just break it up into two steps
 - make a subset, `victims`, of rows where `survived == 0`
 - use `victims` to create a second subset that only contains the 'Age' column.

```
# load a new csv file 'titanic.csv'. you can find it on Blackboard under datasets module
path = '/content/drive/MyDrive/cs167_fall23/datasets/titanic.csv'
# read the file into a dataframe
titanic = pd.read_csv(path)
titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

Subsetting Columns and Rows

- Let's imagine we want a subset that contains the ages of people who did not survive the Titanic. Technically, you have the knowledge now to be able to do this, if you just break it up into two steps
 - make a subset, `victims`, of rows where `survived == 0`
 - use `victims` to create a second subset that only contains the 'Age' column.



```
victims = titanic[titanic['survived'] == 0]
victims.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
5	0	3	male	NaN	0	0	8.4583	Q	Third	man	True	NaN	Queenstown	no	True
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True
7	0	3	male	2.0	3	1	21.0750	S	Third	child	False	NaN	Southampton	no	False

Subsetting Columns and Rows

- Let's imagine we want a subset that contains the ages of people who did not survive the Titanic. Technically, you have the knowledge now to be able to do this, if you just break it up into two steps
 - make a subset, `victims`, of rows where `survived == 0`
 - use `victims` to create a second subset that only contains the 'Age' column.



```
▶ subset = victims['age']
subset.head()

0    22.0
4    35.0
5     NaN
6    54.0
7     2.0
Name: age, dtype: float64
```

Subsetting Columns and Rows

- Let's imagine we want a subset that contains the ages of people who did not survive the Titanic. We can actually do this on one step if we use `loc/iloc`:

```
subset = titanic.loc[ titanic.survived == 0, 'age'] #the first argument is the condition for the rows, the second is the columns
subset.head()
```



```
▶ subset = victims['age']
subset.head()

0    22.0
4    35.0
5     NaN
6    54.0
7     2.0
Name: age, dtype: float64
```

Subsetting Columns and Rows

- We can actually do this on one step in several ways:

```
[27] # option#1
titanic_victims = titanic[victims][['fare', 'age']]
titanic_victims.head()
```

	fare	age
0	7.2500	22.0
4	8.0500	35.0
5	8.4583	NaN
6	51.8625	54.0
7	21.0750	2.0

```
[28] # option#2
titanic_victims = titanic[titanic.survived == 0][['fare', 'age']]
titanic_victims.head()
```

	fare	age
0	7.2500	22.0
4	8.0500	35.0
5	8.4583	NaN
6	51.8625	54.0
7	21.0750	2.0

```
# option#3
titanic_victims = titanic.loc[victims,['fare', 'age']]
titanic_victims.head()
```

	fare	age
0	7.2500	22.0
4	8.0500	35.0
5	8.4583	NaN
6	51.8625	54.0
7	21.0750	2.0

Group Exercise

- Try these

Multiple Conditions

- What if we want to filter rows by multiple conditions? Make sure each condition is in parentheses and use the old school | and & for operators

```
▶ #women and children on titanic  
women_and_children = titanic[(titanic.age < 18) | (titanic.sex == 'female')]  
women_and_children.shape[0]  
women_and_children.survived.sum()
```

↳ 256

```
▶ # men who survived  
men_who_survived = titanic[(titanic.sex == 'male') & (titanic.survived == 1)]  
men_who_survived.shape[0]  
men_who_survived.age.mean()
```

↳ 27.276021505376345

Some Handy Functions

- `mean()`, `median()`, `sum()`

```
▶ #average age of titanic passengers:  
titanic.age.mean()  
#titanic['age'].mean()
```

```
↳ 29.69911764705882
```

```
▶ #median ticket fare for titanic passengers:  
titanic.fare.median()
```

```
↳ 14.4542
```

```
▶ #number of survivors  
titanic.survived.sum()  
total_num_people = titanic.shape[0]  
did_not_survive = total_num_people - titanic.survived.sum()  
did_not_survive
```

```
↳ 549
```

Some Handy Functions

- `unique()`, `groupby()`

```
▶ #get the unique values of the Deck column
titanic.deck.unique()

array([nan, 'C', 'E', 'G', 'D', 'A', 'B', 'F'], dtype=object)
```

```
▶ titanic.groupby(['survived'])['age'].mean()

survived
0    30.626179
1    28.343690
Name: age, dtype: float64
```

```
[19] condition = titanic['survived'] == 0
survivor_0 = titanic[condition]['age']
survivor_0.mean()

30.62617924528302
```

```
[20] condition = titanic['survived'] == 1
survivor_1 = titanic[condition]['age']
survivor_1.mean()

28.343689655172415
```

```
▶ titanic.groupby('survived')['age'].mean()

☞ survived
0    30.626179
1    28.343690
Name: age, dtype: float64
```

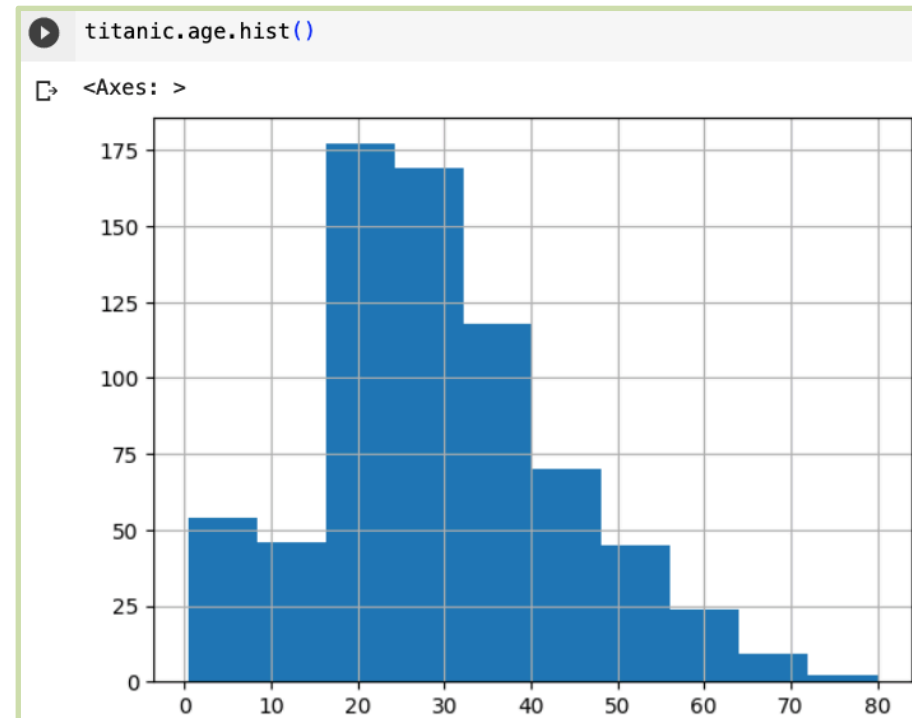

Some Handy Functions

- `describe()`, `hist()`

```
titanic.age.describe()
```

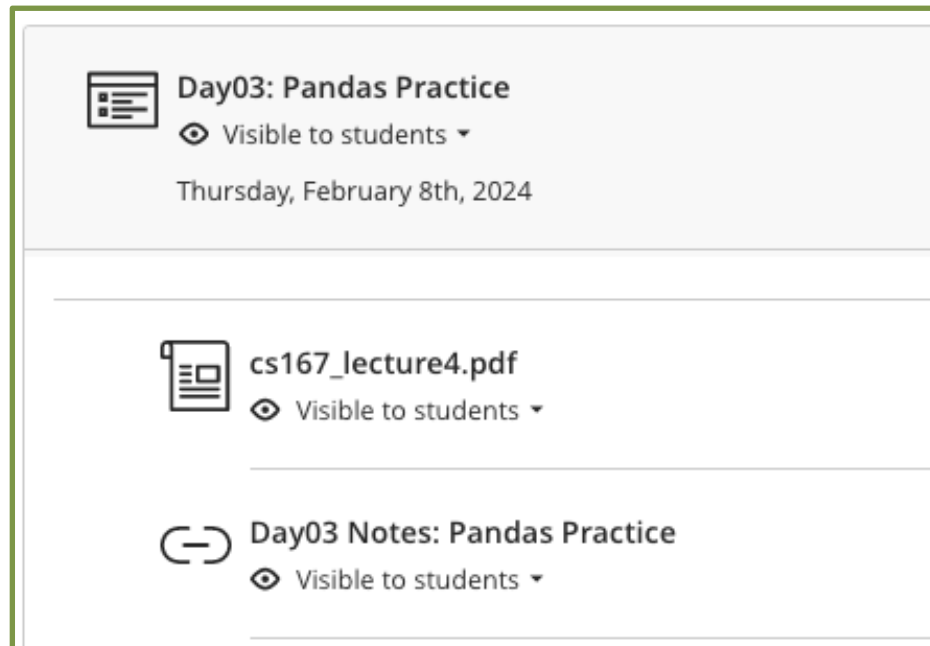
count	714.000000
mean	29.699118
std	14.526497
min	0.420000
25%	20.125000
50%	28.000000
75%	38.000000
max	80.000000

Name: age, dtype: float64



Remaining time: Pandas Exercise

- The following notebook on Blackboard contains exercises for you to try out on your own and play around with Pandas



- This will help you feel prepared for Notebook #1
- Feel free to work with others around you