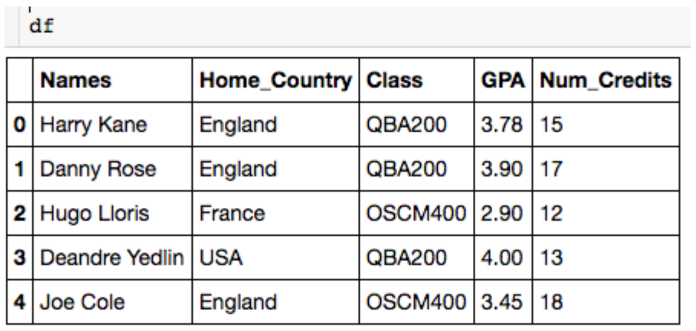
Filtering in Pandas I

Dennis J. Zhang
Washington University in St. Louis

Filtering Dataframes

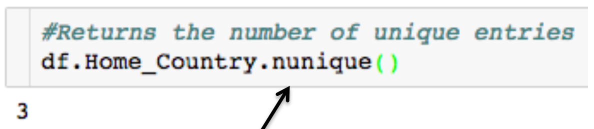
- New dataframe methods
  - Nunique(), Duplicated(), drop_duplicates()
- Faster way to use apply
  - Defining functions using lambda
- How do I only select rows that satisfy a certain condition? Multiple conditions?
  - What is going on under the hood?
- Compute **aggregate** statistics of groups in the data
  - Compute group sums or means

Using nunique()



```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18



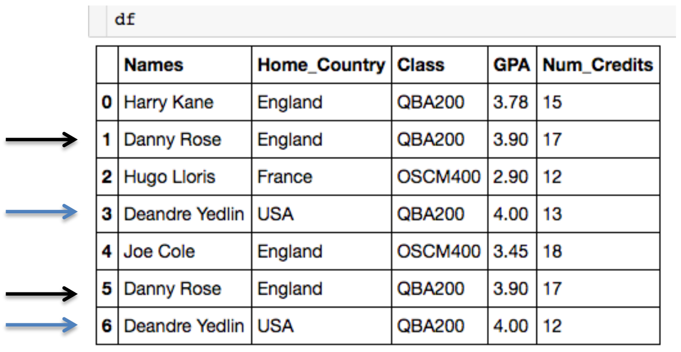
```
#Returns the number of unique entries
df.Home_Country.nunique()
```

```
3
```

Can only be applied to single columns

Using duplicated()

Let’s look at how to deal with duplicate columns:

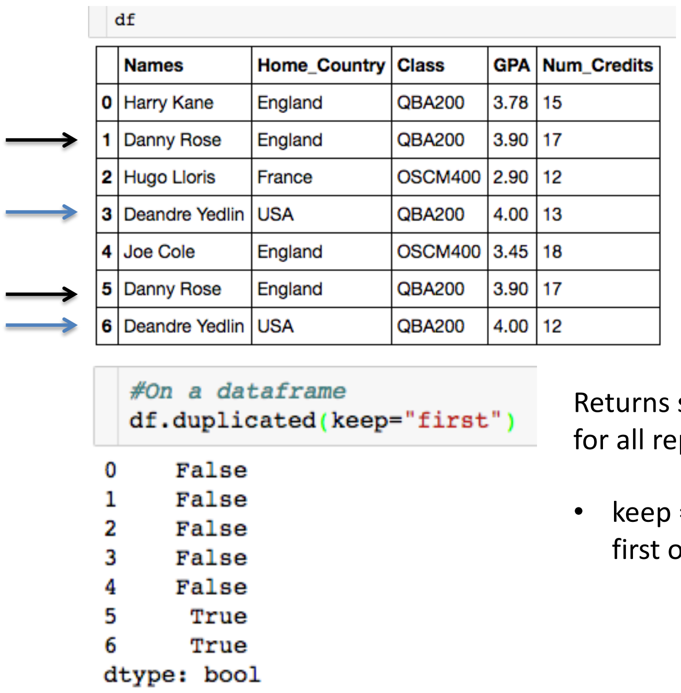


```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	12

Using duplicated()

Let’s look at how to deal with duplicate columns:



```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	12

```
#On a dataframe
df.duplicated(keep="first")
```

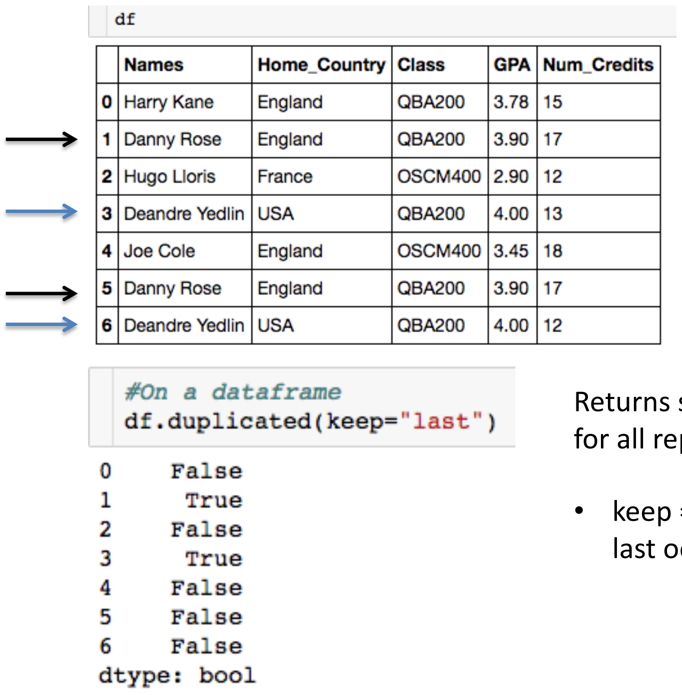
```
0    False
1    False
2    False
3    False
4    False
5     True
6     True
dtype: bool
```

Returns series of boolean with True for all repeated rows.

- keep = “first” – put False for the first occurrence.

Using duplicated()

Let’s look at how to deal with duplicate columns:



```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	12

```
#On a dataframe
df.duplicated(keep="last")
```

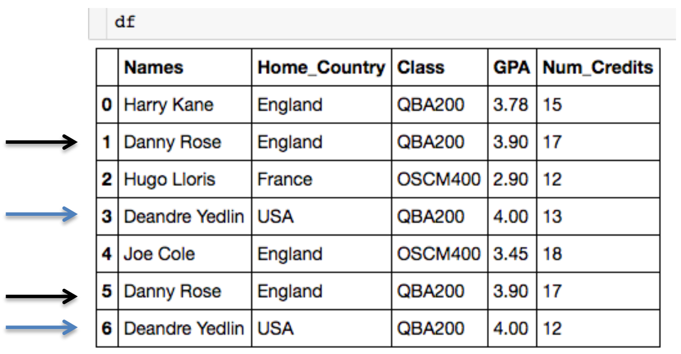
```
0    False
1     True
2    False
3     True
4    False
5    False
6    False
dtype: bool
```

Returns series of boolean with True for all repeated rows.

- keep = “last” – put False for the last occurrence

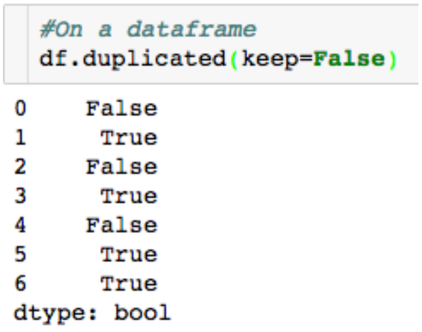
Using duplicated()

Let’s look at how to deal with duplicate columns:



```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	12



```
#On a dataframe
df.duplicated(keep=False)
```

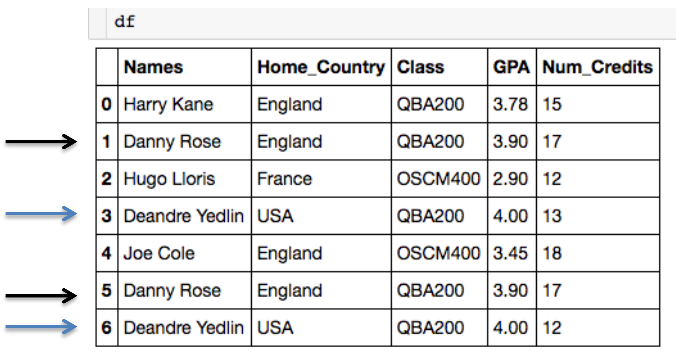
```
0     False
1      True
2     False
3      True
4     False
5      True
6      True
dtype: bool
```

Returns series of boolean with True for all repeated rows.

- keep = False – put True for every row of a repeated occurrence

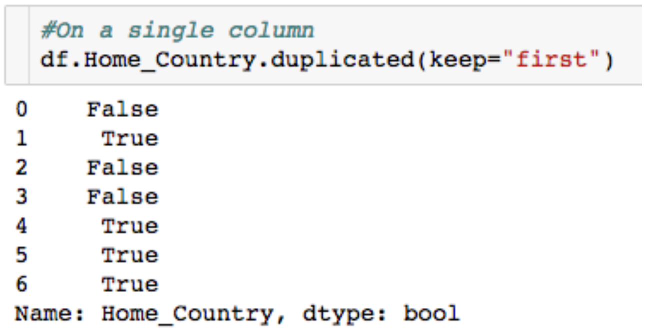
Using duplicated()

Let’s look at how to deal with duplicate columns:



```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	12

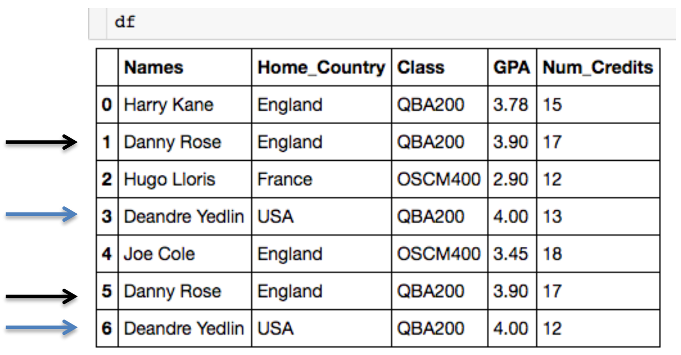


```
#On a single column
df.Home_Country.duplicated(keep="first")
```

```
0     False
1      True
2     False
3     False
4      True
5      True
6      True
Name: Home_Country, dtype: bool
```

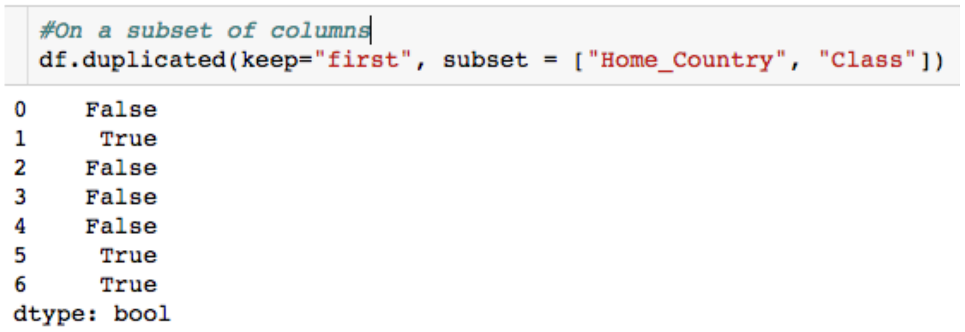
Using duplicated()

Let’s look at how to deal with duplicate columns:



```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	12

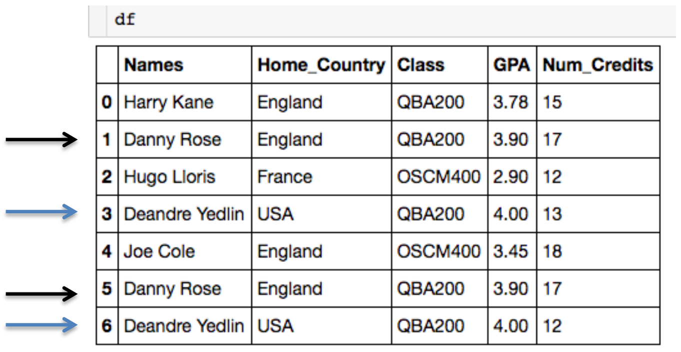


```
#On a subset of columns
df.duplicated(keep="first", subset = ["Home_Country", "Class"])
```

```
0    False
1     True
2    False
3    False
4    False
5     True
6     True
dtype: bool
```

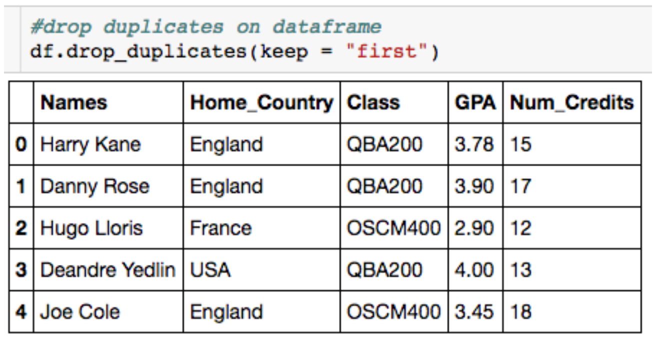
Using drop_duplicates()

Let’s look at how to deal with duplicate columns:



```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	12



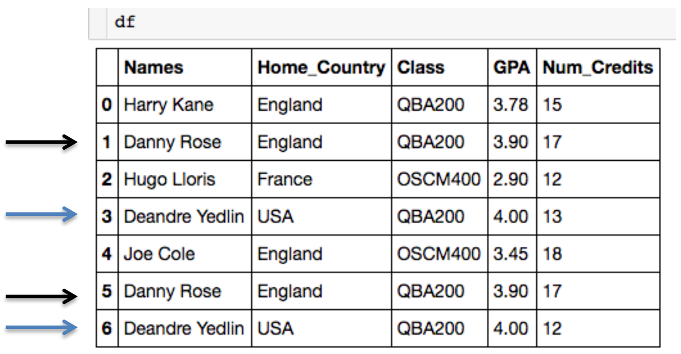
```
#drop duplicates on dataframe
df.drop_duplicates(keep = "first")
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

“Drop all duplicate rows except for the first.”

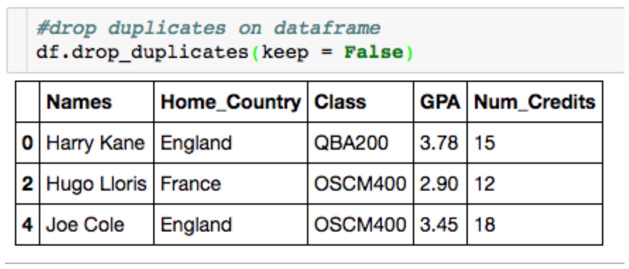
Using drop_duplicates()

Let’s look at how to deal with duplicate columns:



```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	12



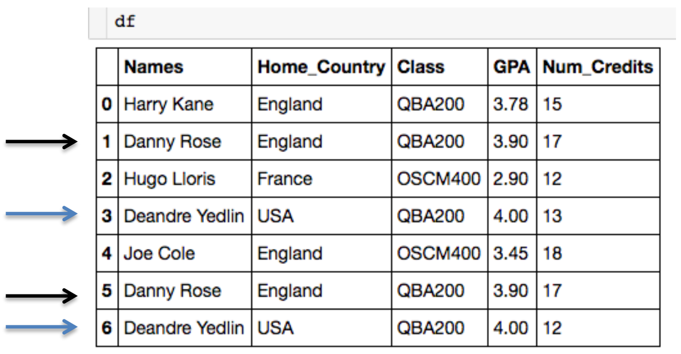
```
#drop duplicates on dataframe
df.drop_duplicates(keep = False)
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
2	Hugo Lloris	France	OSCM400	2.90	12
4	Joe Cole	England	OSCM400	3.45	18

“Drop all duplicate rows.”

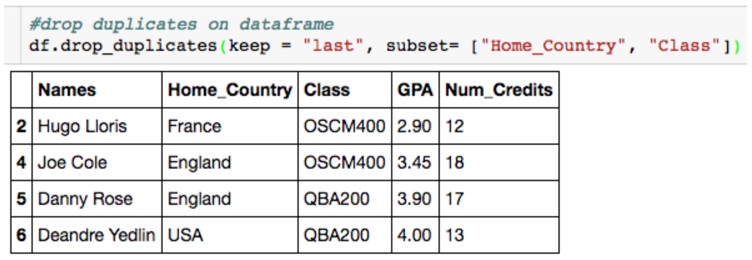
Using drop_duplicates()

Let’s look at how to deal with duplicate columns:



```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	12



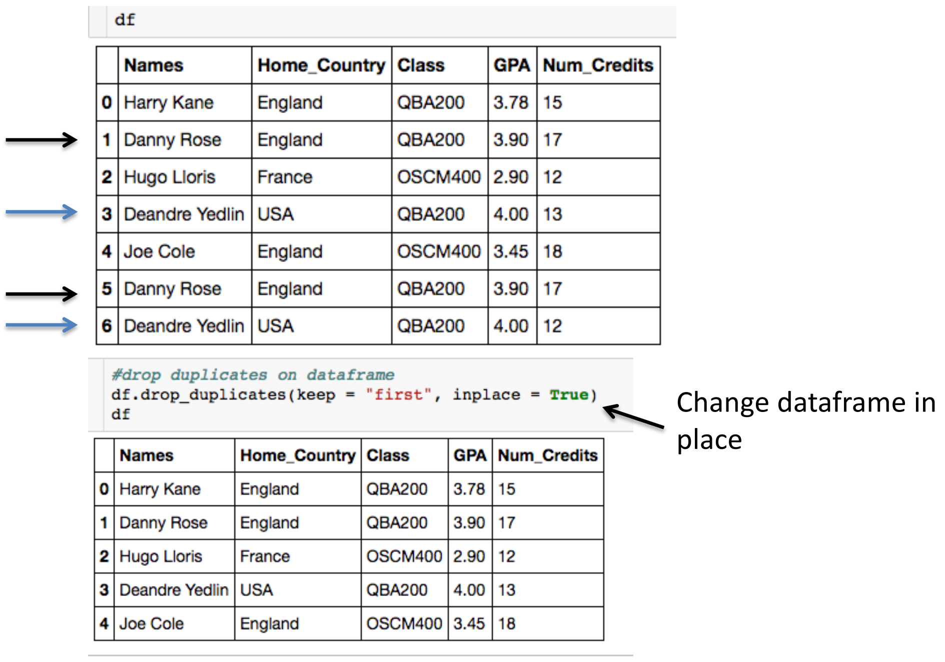
```
#drop duplicates on dataframe
df.drop_duplicates(keep = "last", subset= ["Home_Country", "Class"])
```

	Names	Home_Country	Class	GPA	Num_Credits
2	Hugo Lloris	France	OSCM400	2.90	12
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	13

“Drop all duplicate rows based on Home_Country and Class except for the last.”

Using drop_duplicates()

Let's look at how to deal with duplicate columns:



```
df
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18
5	Danny Rose	England	QBA200	3.90	17
6	Deandre Yedlin	USA	QBA200	4.00	12

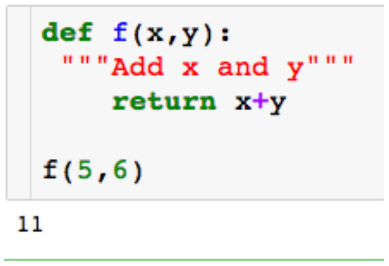
```
#drop duplicates on dataframe
df.drop_duplicates(keep = "first", inplace = True)
df
```

Change dataframe in place

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

Lambda Functions

Lambda function allows us to create simple function in one line without a def.

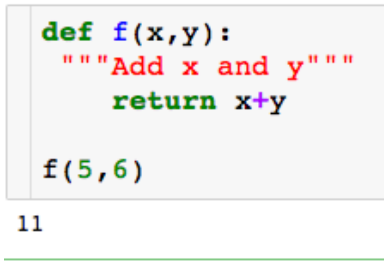


```
def f(x,y):
  """Add x and y"""
    return x+y

f(5,6)
```

```
11
```

Lambda Functions

Lambda function allows us to create simple function in one line without a def.



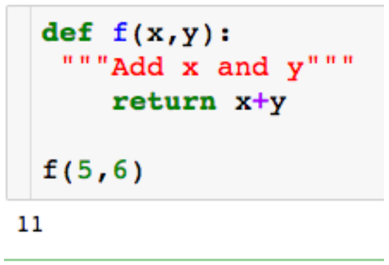
```
def f(x,y):
 """Add x and y"""
    return x+y

f(5,6)
```

```
11
```

Let’s see how we can use a lambda function to do the same thing

Lambda Functions

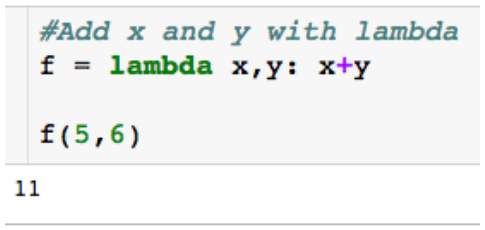
Lambda function allows us to create simple function in one line without a def.



```
def f(x,y):
 """Add x and y"""
    return x+y

f(5,6)
```

```
11
```

Let’s see how we can use a lambda function to do the same thing



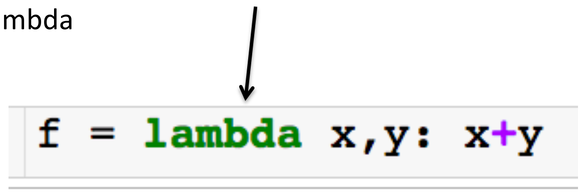
```
#Add x and y with lambda
f = lambda x,y: x+y

f(5,6)
```

```
11
```

Lambda Functions

Dissecting the lambda function:

All lambda function begin by typing lambda

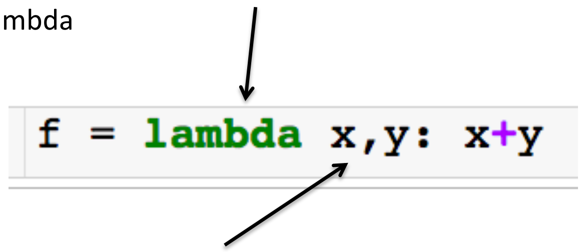


```
f = lambda x,y: x+y
```

Lambda Functions

Dissecting the lambda function:

All lambda function begin by typing lambda

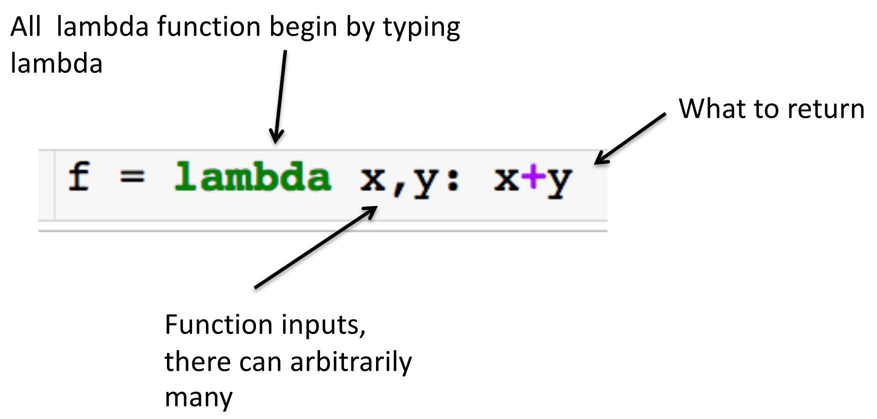


```
f = lambda x,y: x+y
```

Function inputs, there can arbitrarily many

Lambda Functions

Dissecting the lambda function:



All lambda function begin by typing lambda

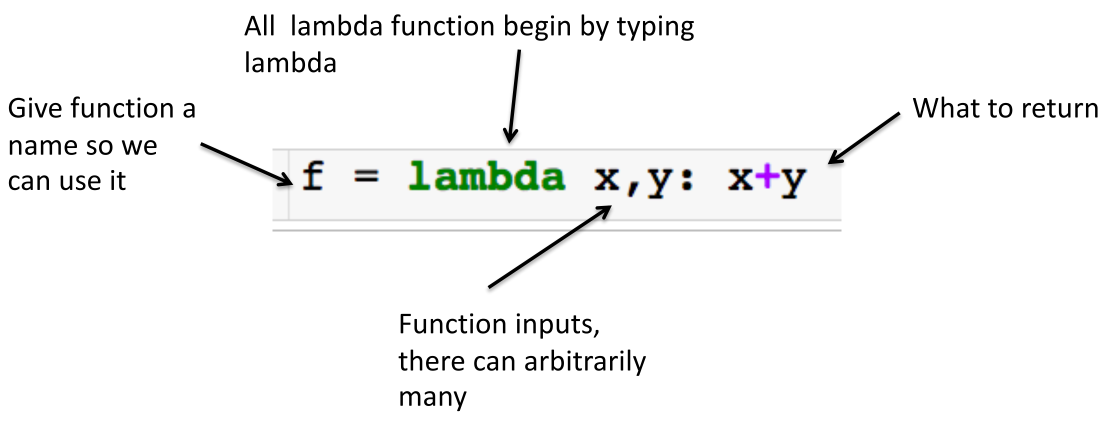
What to return

```
f = lambda x,y: x+y
```

Function inputs, there can arbitrarily many

Lambda Functions

Dissecting the lambda function:



All lambda function begin by typing lambda

Give function a name so we can use it

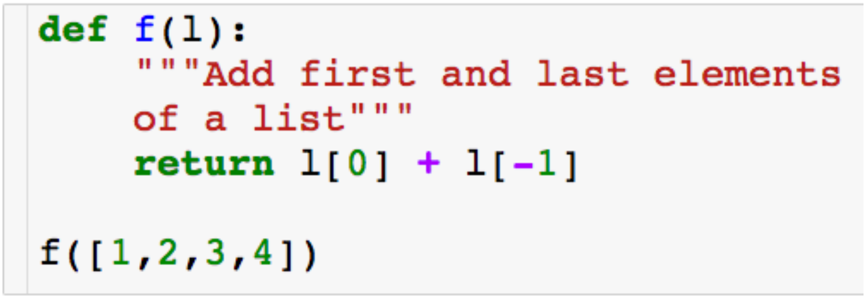
What to return

```
f = lambda x,y: x+y
```

Function inputs, there can arbitrarily many

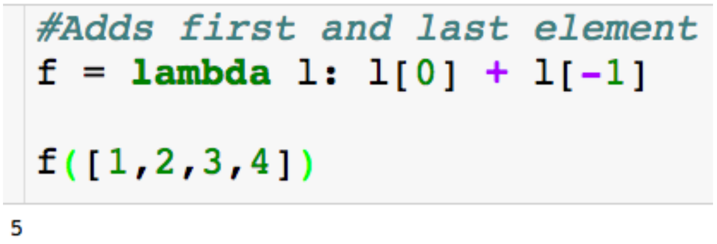
Lambda Functions

Another example:



```
def f(l):
    """Add first and last elements
    of a list"""
    return l[0] + l[-1]

f([1,2,3,4])
```

Lambda function:

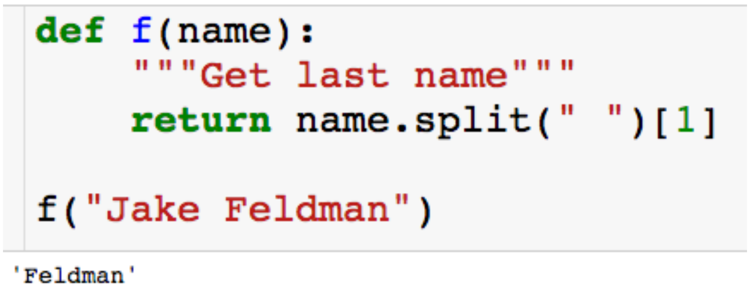


```
#Adds first and last element
f = lambda l: l[0] + l[-1]

f([1,2,3,4])
```

```
5
```

Lambda Functions

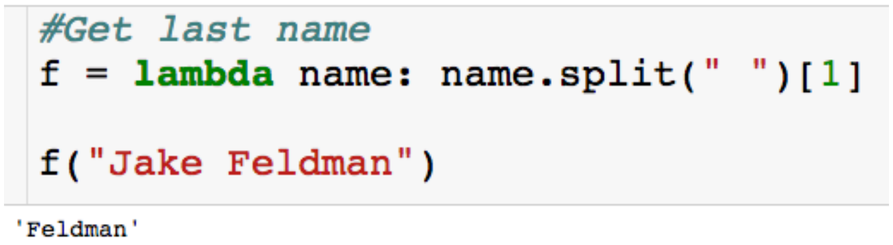
Another example:



```
def f(name):
    """Get last name"""
    return name.split(" ")[1]

f("Jake Feldman")
```

```
'Feldman'
```

Lambda function:



```
#Get last name
f = lambda name: name.split(" ")[1]

f("Jake Feldman")
```

```
'Feldman'
```

Uses of Lambda Functions

Can uses lamba functions as input to key in sorted function:

```
l = [[1,2], [5,3], [1,1], [7,4], [12,1.5]]
#Sorts by first element
sorted(l)
```

```
[[1, 1], [1, 2], [5, 3], [7, 4], [12, 1.5]]
```

```
l = [[1,2], [5,3], [1,1], [7,4], [12,1.5]]
#Sort by second element
sorted(l, key=lambda l: l[1])
```

```
[[1, 1], [12, 1.5], [1, 2], [5, 3], [7, 4]]
```

Lambda function takes list and returns the second element

With the key parameter you can specify a function to be called on each list element prior to making comparisons. The value of the key parameter should be a function that takes a single argument and returns a key to use for sorting purposes.

Use of Lambda Functions

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S

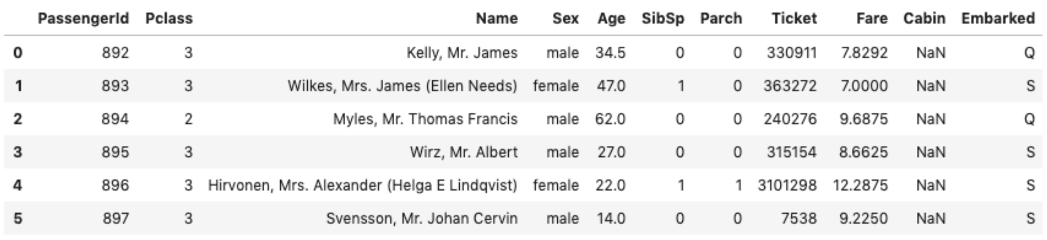
How do I add a column for each person's last name?

```
# Method one using Apply
def Get_First_Name(name):
    return name.split(",")[0]

df["Last_Name"] = df["Name"].apply(Get_First_Name)
df
```

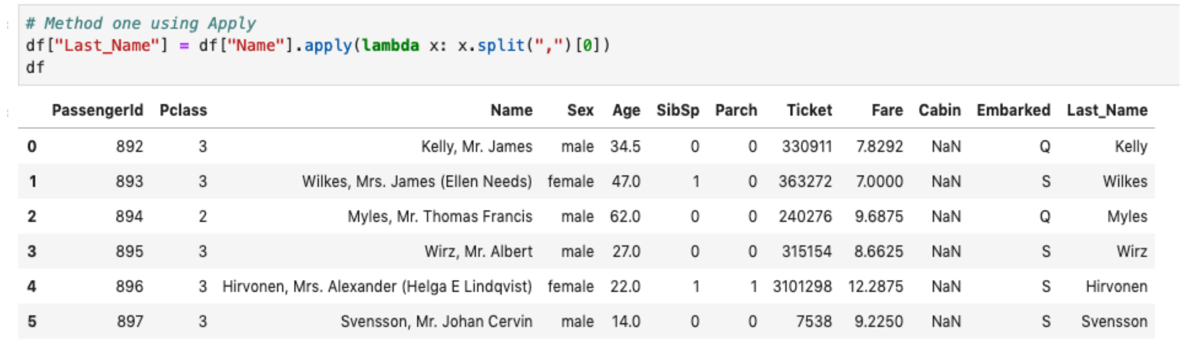
Already wrote this function as a lambda function

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Last_Name
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	Kelly
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	Wilkes
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	Myles
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	Wirz
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	Hirvonen
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S	Svensson

Use of Lambda Functions



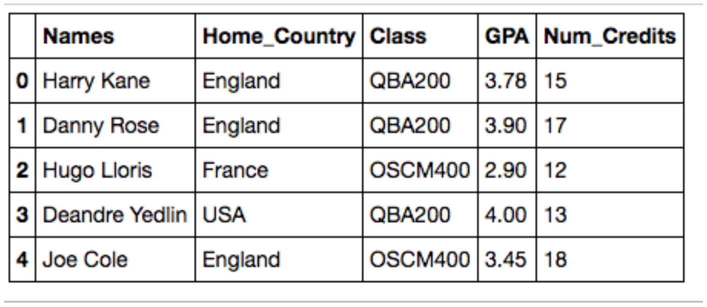
	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S



```
# Method one using Apply
df["Last_Name"] = df["Name"].apply(lambda x: x.split(",")[0])
df
```

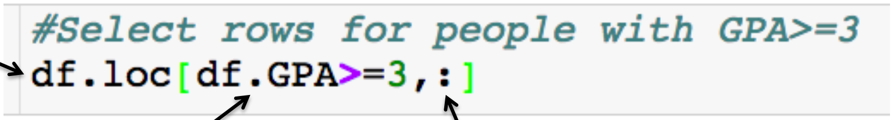
	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Last_Name
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	Kelly
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	Wilkes
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	Myles
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	Wirz
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	Hirvonen
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S	Svensson

Filtering



	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

Always
name_df.loc

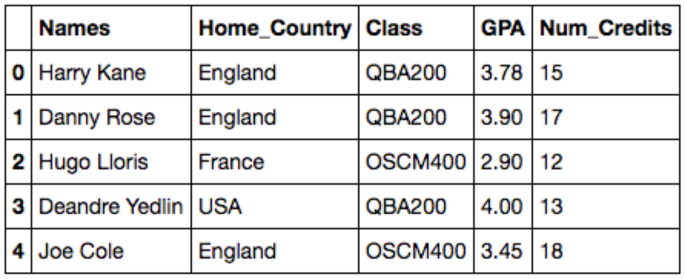


```
#Select rows for people with GPA>=3
df.loc[df.GPA>=3,:]
```

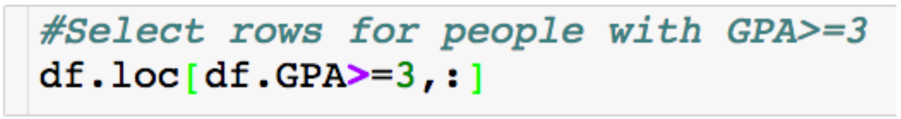
only rows where
GPA>=3

All columns

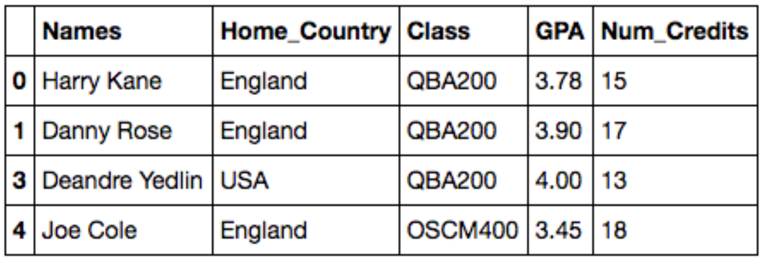
Filtering



	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18



```
#Select rows for people with GPA>=3
df.loc[df.GPA>=3,:]
```

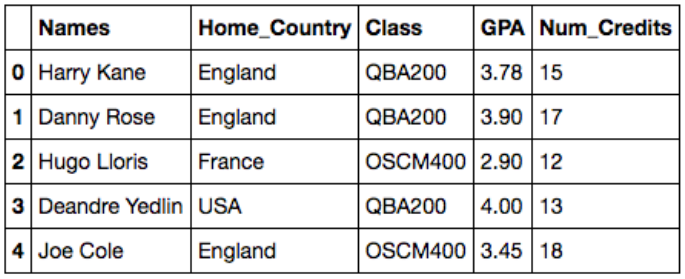


	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

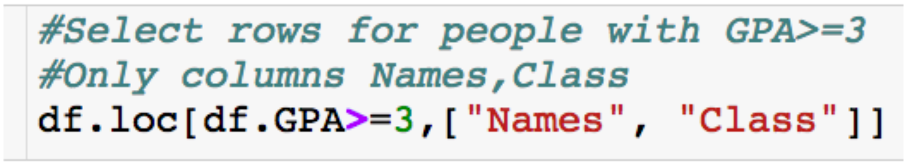
Index is messed up

Returns a dataframe

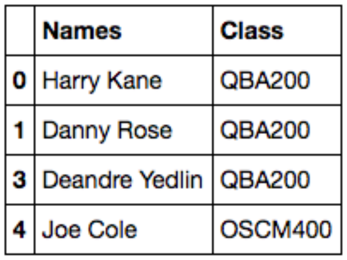
Filtering



	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18



```
#Select rows for people with GPA>=3
#Only columns Names,Class
df.loc[df.GPA>=3,["Names", "Class"]]
```



	Names	Class
0	Harry Kane	QBA200
1	Danny Rose	QBA200
3	Deandre Yedlin	QBA200
4	Joe Cole	OSCM400

We don't have to select the column we filter on.

Filtering

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

```
#Storing the result
df_smart = df.loc[df.GPA>=3,["Names", "Class", "GPA"]]
df_smart
```

	Names	Class	GPA
0	Harry Kane	QBA200	3.78
1	Danny Rose	QBA200	3.90
3	Deandre Yedlin	QBA200	4.00
4	Joe Cole	OSCM400	3.45

Under Hood

```
#Return series of booleans
df.GPA>=3
```

```
0     True
1     True
2    False
3     True
4     True
Name: GPA, dtype: bool
```

Only keep rows where there is a True

```
#GPA>=3
df.loc[df.GPA>=3,:]
```

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

Filtering – Multiple Conditions

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

```
#English People with >15 Credits
df.loc[(df.Home_Country == "England") \
       & (df.Num_Credits>15), : ]
```

"and"

Make sure you put each condition in parentheses

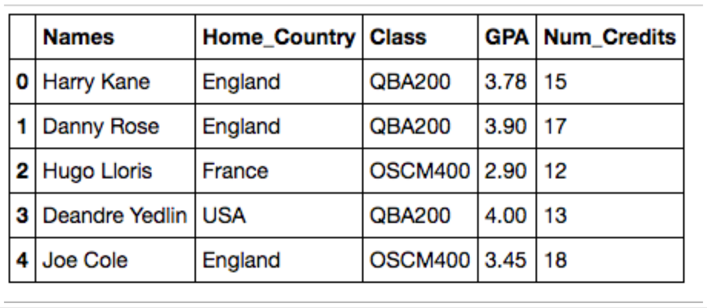
Filtering – Multiple Conditions

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

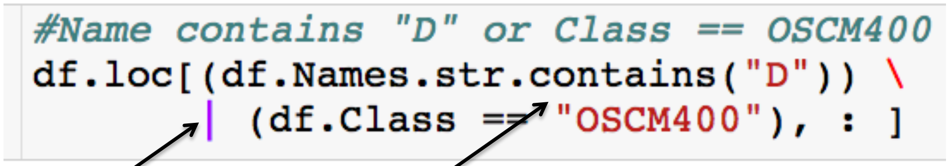
```
#English People with >15 Credits
df.loc[(df.Home_Country == "England") \
       & (df.Num_Credits>15), : ]
```

	Names	Home_Country	Class	GPA	Num_Credits
1	Danny Rose	England	QBA200	3.90	17
4	Joe Cole	England	OSCM400	3.45	18

Filtering – Multiple Conditions



	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

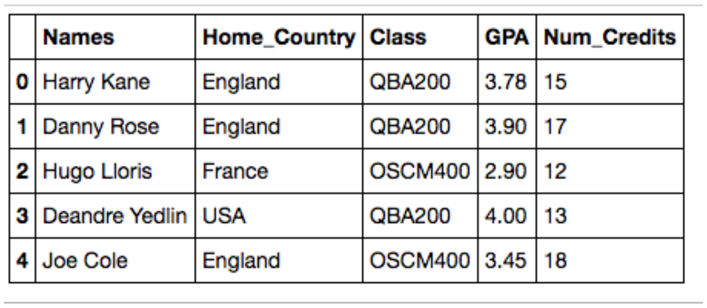


```
#Name contains "D" or Class == OSCM400
df.loc[(df.Names.str.contains("D")) \
        | (df.Class == "OSCM400"), : ]
```

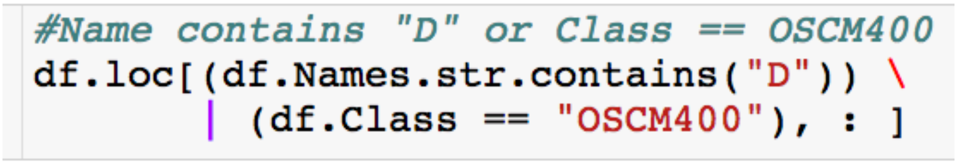
Returns True if string contains a "D"

"or"

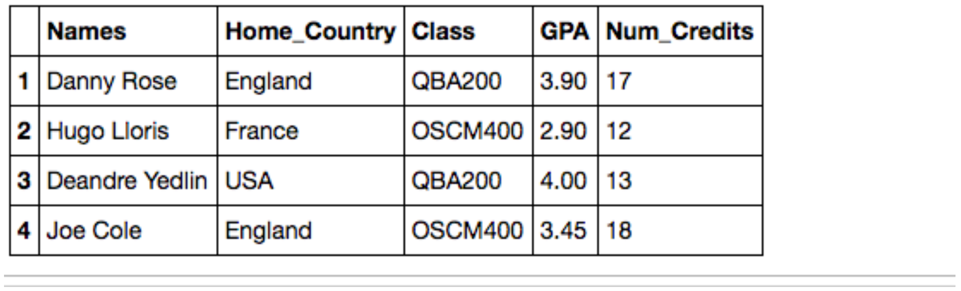
Filtering – Multiple Conditions



	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18



```
#Name contains "D" or Class == OSCM400
df.loc[(df.Names.str.contains("D")) \
        | (df.Class == "OSCM400"), : ]
```



	Names	Home_Country	Class	GPA	Num_Credits
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

Filtering – Multiple Conditions

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

```
#People from France or USA
df.loc[(df.Home_Country == "France") \
       | (df.Home_Country == "USA"), : ]
```

	Names	Home_Country	Class	GPA	Num_Credits
2	Hugo Lloris	France	OSCM400	2.9	12
3	Deandre Yedlin	USA	QBA200	4.0	13

Filtering – Multiple Conditions

	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

```
df.Home_Country.isin(["France", "USA"])
```

```
0    False
1    False
2     True
3     True
4    False
Name: Home_Country, dtype: bool
```

Input is a list

Filtering – Multiple Conditions

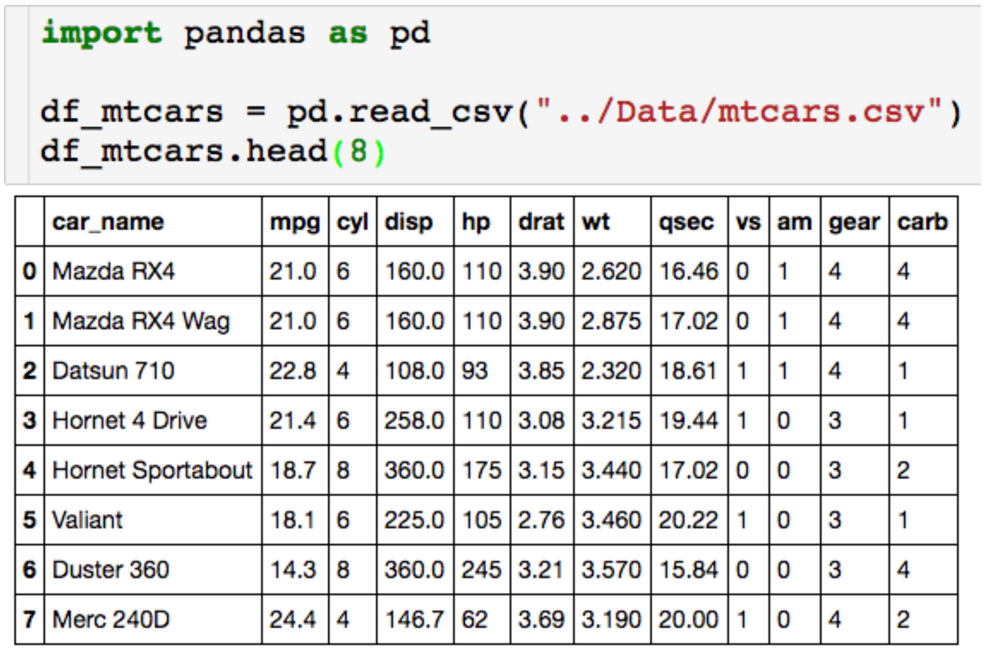
	Names	Home_Country	Class	GPA	Num_Credits
0	Harry Kane	England	QBA200	3.78	15
1	Danny Rose	England	QBA200	3.90	17
2	Hugo Lloris	France	OSCM400	2.90	12
3	Deandre Yedlin	USA	QBA200	4.00	13
4	Joe Cole	England	OSCM400	3.45	18

```
#People from France or USA
#Using isin()
df.loc[df.Home_Country.isin(["France", "USA"]), : ]
```

	Names	Home_Country	Class	GPA	Num_Credits
2	Hugo Lloris	France	OSCM400	2.9	12
3	Deandre Yedlin	USA	QBA200	4.0	13

Groupby in Pandas - Aggregating

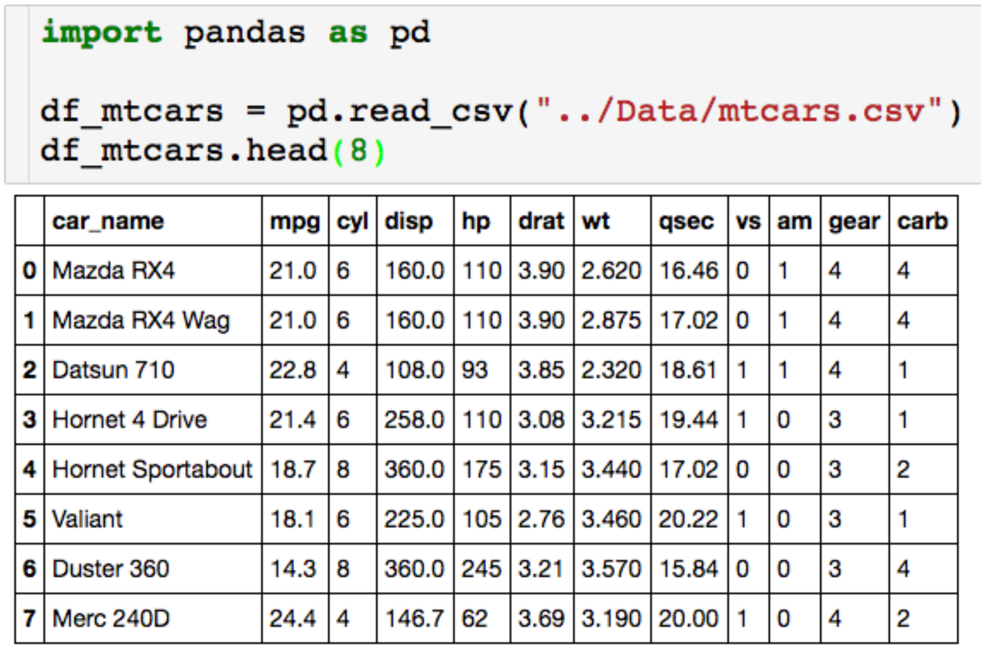
Looking at mtcars Again



```python
import pandas as pd

df_mtcars = pd.read_csv("../Data/mtcars.csv")
df_mtcars.head(8)
```

	car_name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2

Looking at mtcars Again

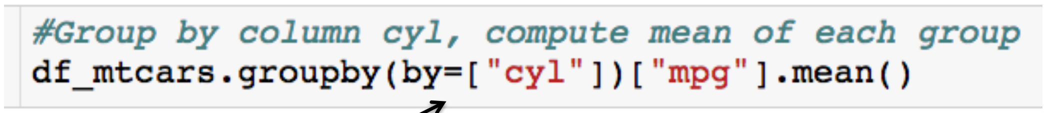


```python
import pandas as pd

df_mtcars = pd.read_csv("../Data/mtcars.csv")
df_mtcars.head(8)
```

	car_name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2

What is the avg mpg for each cylinder type?

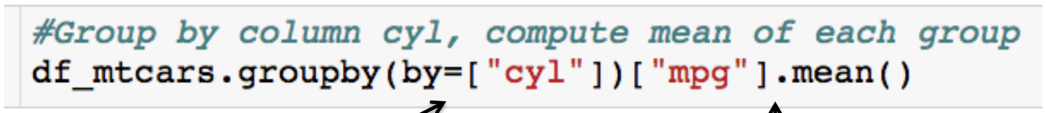
Basic Group By



```
#Group by column cyl, compute mean of each group
df_mtcars.groupby(by=["cyl"])["mpg"].mean()
```

Create groups based on
unique entries in cyl column

Basic Group By



```
#Group by column cyl, compute mean of each group
df_mtcars.groupby(by=["cyl"])["mpg"].mean()
```

Create groups based on
unique entries in cyl column

For each group compute
the mean mpg

Basic Group By

```python
#Group by column cyl, compute mean of each group
df_mtcars.groupby(by=["cyl"])["mpg"].mean()
```

```
cyl
4    26.663636
6    19.742857
8    15.100000
Name: mpg, dtype: float64
```

Returns a series where the indices are the groups

Basic Group By

```python
#Group by column cyl, compute mean of each group
df_mtcars.groupby(by=["cyl"])["mpg"].mean()
```

```
cyl
4    26.663636
6    19.742857
8    15.100000
Name: mpg, dtype: float64
```

Returns a series where the indices are the groups

Remaining Questions:

- Can I group by more than one column?
- Can I compute more than one aggregate statistic for each group?
- For each group can I customize how I summarize each column that I select?

Selecting Multiple Columns

```python
#Selecting multiple columns after grouping
df_mtcars.groupby(by=["cyl"])["mpg", "hp"].mean()
```

Specify the columns you want

	mpg	hp
cyl		
4	26.663636	82.636364
6	19.742857	122.285714
8	15.100000	209.214286

- We get avg mpg and hp for each of the three cylinder groups.
- Since we are selecting two columns we get back a dataframe

Grouping By Multiple Columns

```python
#Selecting multiple columns after grouping
df_mtcars.groupby(by=["cyl", "am"])["mpg", "hp"].mean()
```

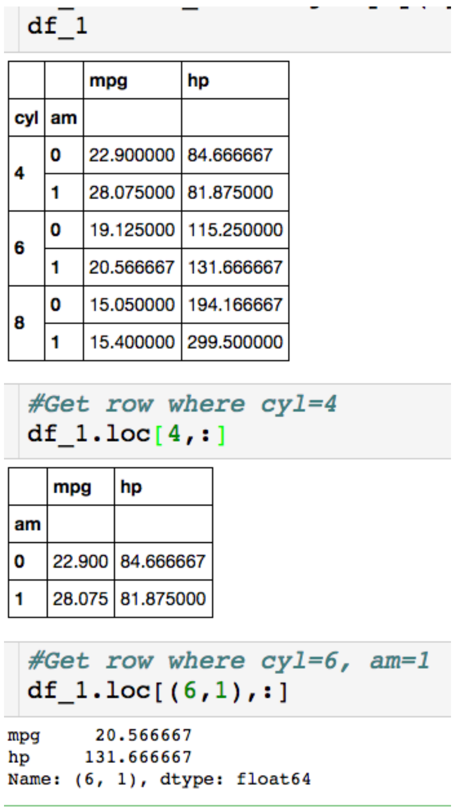
Specify columns you want to group by

		mpg	hp
cyl	am		
4	0	22.900000	84.666667
	1	28.075000	81.875000
6	0	19.125000	115.250000
	1	20.566667	131.666667
8	0	15.050000	194.166667
	1	15.400000	299.500000

- We have a group for every combination of cyl and am.
- We get avg mpg and hp for each of the six groups.
- We get a multi-indexed dataframe (two row names).

Slicing Multi-indexed DataFrame

Options 1:



```
df_1
```

		mpg	hp
cyl	am		
4	0	22.900000	84.666667
	1	28.075000	81.875000
6	0	19.125000	115.250000
	1	20.566667	131.666667
8	0	15.050000	194.166667
	1	15.400000	299.500000

```
#Get row where cyl=4
df_1.loc[4,:]
```

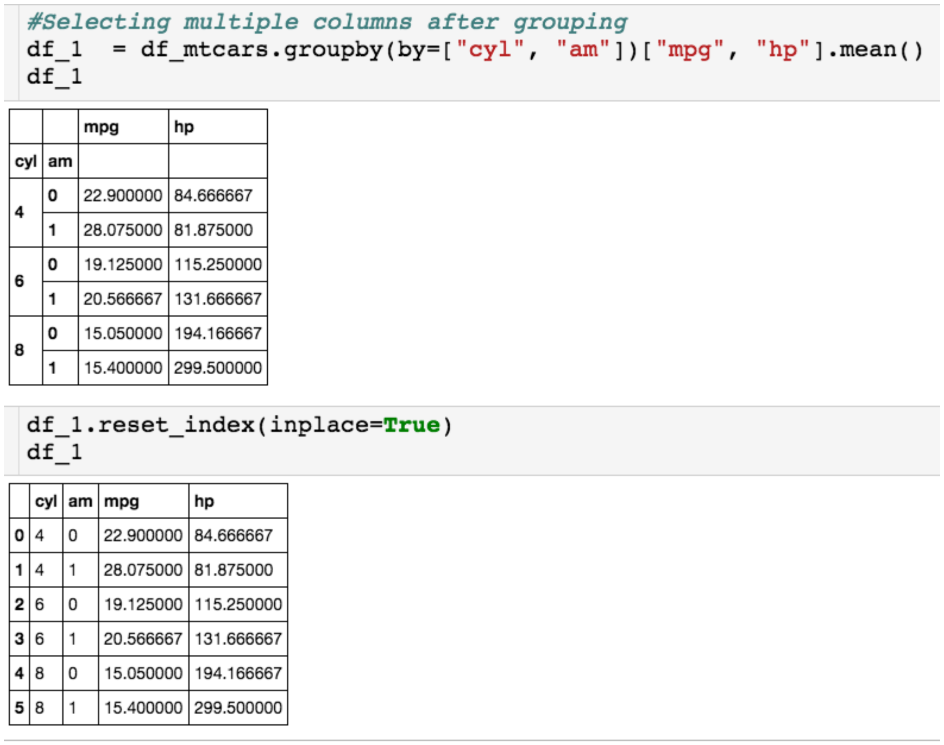
	mpg	hp
am		
0	22.900	84.666667
1	28.075	81.875000

```
#Get row where cyl=6, am=1
df_1.loc[(6,1),:]
```

```
mpg      20.566667
hp      131.666667
Name: (6, 1), dtype: float64
```

Just Reset Index…

Options 2:



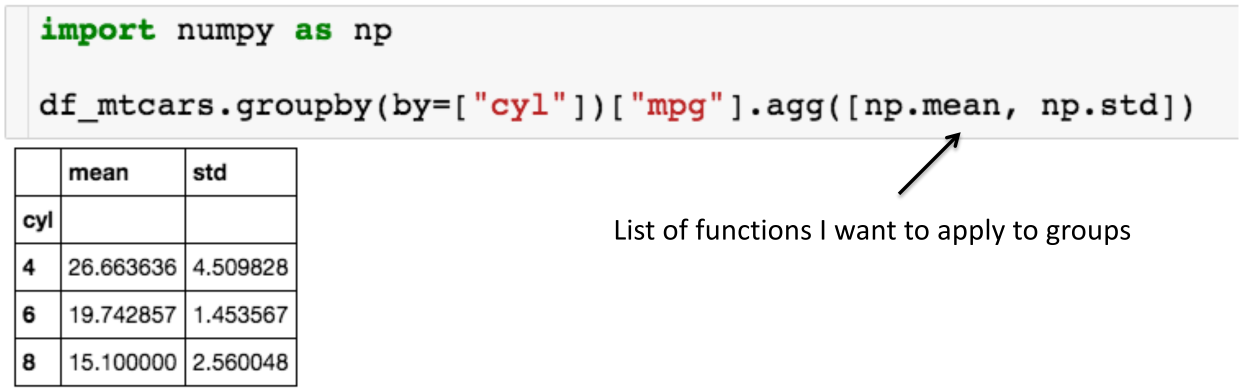
```
#Selecting multiple columns after grouping
df_1  = df_mtcars.groupby(by=["cyl", "am"])["mpg", "hp"].mean()
df_1
```

		mpg	hp
cyl	am		
4	0	22.900000	84.666667
	1	28.075000	81.875000
6	0	19.125000	115.250000
	1	20.566667	131.666667
8	0	15.050000	194.166667
	1	15.400000	299.500000

```
df_1.reset_index(inplace=True)
df_1
```

	cyl	am	mpg	hp
0	4	0	22.900000	84.666667
1	4	1	28.075000	81.875000
2	6	0	19.125000	115.250000
3	6	1	20.566667	131.666667
4	8	0	15.050000	194.166667
5	8	1	15.400000	299.500000

Apply Multiple Functions



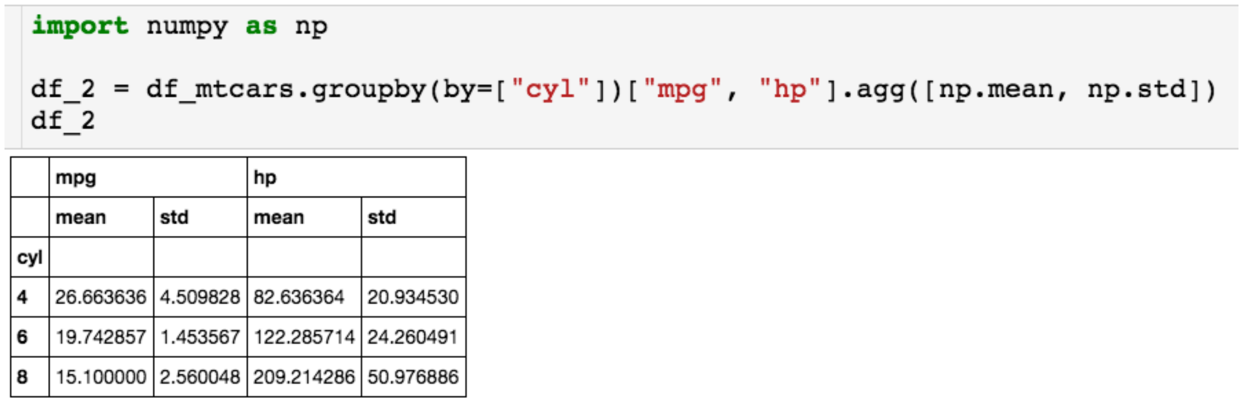
```
import numpy as np

df_mtcars.groupby(by=["cyl"])["mpg"].agg([np.mean, np.std])
```

	mean	std
cyl		
4	26.663636	4.509828
6	19.742857	1.453567
8	15.100000	2.560048

List of functions I want to apply to groups

- For each group of cylinders we get the mean and stdev mpg.

Apply Multiple Functions to Multiple Columns



```
import numpy as np

df_2 = df_mtcars.groupby(by=["cyl"])["mpg", "hp"].agg([np.mean, np.std])
df_2
```

	mpg		hp	
	mean	std	mean	std
cyl				
4	26.663636	4.509828	82.636364	20.934530
6	19.742857	1.453567	122.285714	24.260491
8	15.100000	2.560048	209.214286	50.976886

- For each group of cylinders we get the mean and stdev of mpg and hp.
- Get dataframe with multi-indexed column names

Apply Multiple Functions to Multiple Columns

```
import numpy as np

df_2 = df_mtcars.groupby(by=["cyl"])["mpg", "hp"].agg([np.mean, np.std])
df_2
```

	mpg		hp	
	mean	std	mean	std
cyl				
4	26.663636	4.509828	82.636364	20.934530
6	19.742857	1.453567	122.285714	24.260491
8	15.100000	2.560048	209.214286	50.976886

```
#Select mpg info
df_2.loc[:, "mpg"]
```

	mean	std
cyl		
4	26.663636	4.509828
6	19.742857	1.453567
8	15.100000	2.560048

Get dataframe back

Apply Multiple Functions to Multiple Columns

Dictionary

```
df_mtcars.groupby(by=["cyl"]).agg({"hp":np.mean, "mpg":np.std})
```

	hp	mpg
cyl		
4	82.636364	4.509828
6	122.285714	1.453567
8	209.214286	2.560048

For hp column compute mean of groups and for mpg column compute standard deviation

Using Apply

```
df_mtcars.groupby(by=["cyl"]).agg({"hp":np.mean, "mpg":np.std})
```

```
             hp       mpg
cyl
  4   82.636364  4.509828
  6  122.285714  1.453567
  8  209.214286  2.560048
```

We can use apply().
The input now is a dataframe

```
def app_function(group):
    result = {"hp":np.mean(group["hp"]), "mpg":np.mean(group["mpg"])}
    return pd.Series(result)

df_mtcars.groupby(by=["cyl"]).apply(app_function)
```

Output is a series.

```
             hp        mpg
cyl
  4   82.636364  26.663636
  6  122.285714  19.742857
  8  209.214286  15.100000
```

Using Apply()

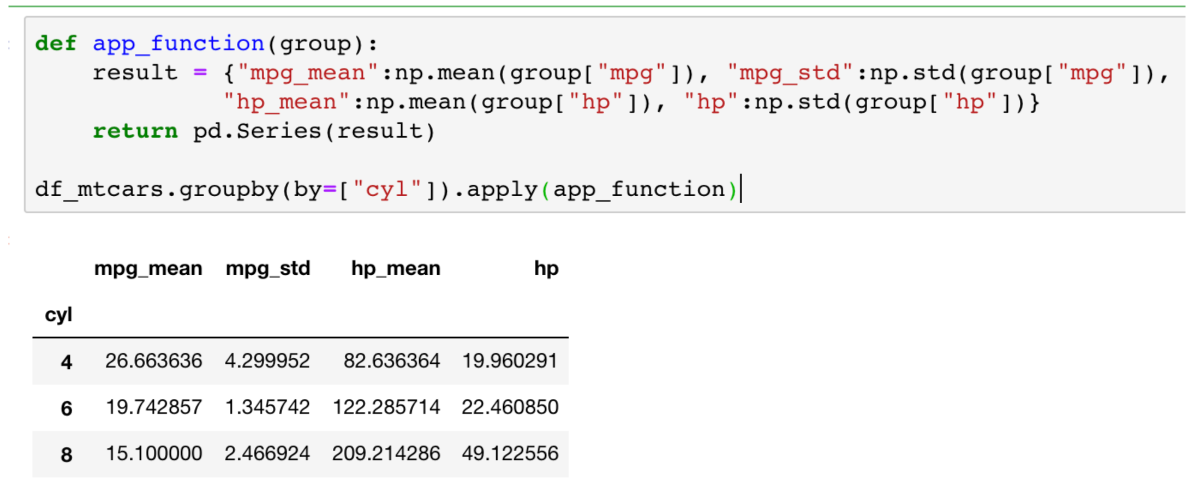
```
import numpy as np

df_2 = df_mtcars.groupby(by=["cyl"])["mpg", "hp"].agg([np.mean, np.std])
df_2
```

	mpg		hp	
	mean	std	mean	std
cyl				
4	26.663636	4.509828	82.636364	20.934530
6	19.742857	1.453567	122.285714	24.260491
8	15.100000	2.560048	209.214286	50.976886

Can we use apply to generate a dataframe without two-level columns?

The resulting column names should be [mpg_mean, mpg_std, hp_mean, hp_std]

Using Apply()

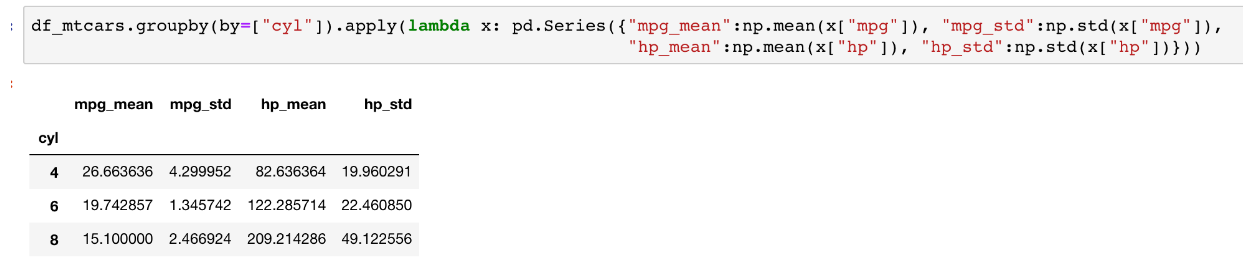


```python
def app_function(group):
    result = {"mpg_mean":np.mean(group["mpg"]), "mpg_std":np.std(group["mpg"]),
              "hp_mean":np.mean(group["hp"]), "hp":np.std(group["hp"])}
    return pd.Series(result)

df_mtcars.groupby(by=["cyl"]).apply(app_function)
```

```
     mpg_mean   mpg_std     hp_mean         hp
cyl
  4  26.663636  4.299952   82.636364  19.960291
  6  19.742857  1.345742  122.285714  22.460850
  8  15.100000  2.466924  209.214286  49.122556
```

Can we do it in one line?

Using Apply() + Lambda functions



```python
df_mtcars.groupby(by=["cyl"]).apply(lambda x: pd.Series({"mpg_mean":np.mean(x["mpg"]), "mpg_std":np.std(x["mpg"]),
                                                          "hp_mean":np.mean(x["hp"]), "hp_std":np.std(x["hp"])}))
```

```
     mpg_mean   mpg_std     hp_mean     hp_std
cyl
  4  26.663636  4.299952   82.636364  19.960291
  6  19.742857  1.345742  122.285714  22.460850
  8  15.100000  2.466924  209.214286  49.122556
```