Filtering in Pandas II

Dennis J. Zhang
Washington University in St. Louis

Filtering Dataframes

- Group by
  - Perform some group specific **transformations** on the data
    - Fill NaNs of groups with mean of group
  - **Filter** the data based on some group-wise operations
    - Discard data that belongs to groups with only a few data points.
- Merge:
  - **Inner Merge**
  - **Left and Outer Merge**

Looking at Modified mtcars

```
df_mtcars
```

	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	NaN	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	NaN	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	NaN	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

- Transforms preserve the shape of the original data frames.
- Lets us transform the data for each group.

Using fillna()

```
df_mtcars
```

	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	NaN	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	NaN	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	NaN	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

```
#Dealing with Missing Data
mean_mpg = df_mtcars["mpg"].mean()
df_mtcars.mpg.fillna(mean_mpg, inplace=True)
df_mtcars
```

All missing values replaced by same mean!

	car_name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.00	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	19.84	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	19.84	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.40	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	19.84	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.10	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.30	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.40	4	146.7	62	3.69	3.190	20.00	1	0	4	2

Using a Transform

```
df_mtcars
```

	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	NaN	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	NaN	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	NaN	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

```
df_mtcars.groupby(by = ["cyl"]).mpg.mean()
```

```
cyl
4    24.400000
6    20.166667
8    14.300000
Name: mpg, dtype: float64
```

I'd like to replace each NaN with its cyl group mean for example.

Using a Transform


```
df_mtcars.groupby(by = ["cyl"]).mpg.mean()
```

```
cyl
4    24.400000
6    20.166667
8    14.300000
Name: mpg, dtype: float64
```


x is group as series

```
#replace each mpg with group mean
df_mtcars.groupby(by = ["cyl"]).mpg.transform(lambda x: x.mean())
```

Function to be applied to group

```
0    20.166667
1    20.166667
2    24.400000
3    20.166667
4    14.300000
5    20.166667
6    14.300000
7    24.400000
Name: mpg, dtype: float64
```

Using a Transform

```
df_mtcars.groupby(by = ["cyl"]).mpg.mean()
```

```
cyl
4    24.400000
6    20.166667
8    14.300000
Name: mpg, dtype: float64
```

```
#Get new mpg column
df_mtcars["mpg"] = df_mtcars.groupby(by = ["cyl"]).mpg.transform(lambda x: x.mean())
df_mtcars
```

	car_name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	20.166667	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	20.166667	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	24.400000	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	20.166667	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	14.300000	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	20.166667	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.300000	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.400000	4	146.7	62	3.69	3.190	20.00	1	0	4	2

Using a Transform

```
df_mtcars.groupby(by = ["cyl"]).mpg.mean()
```

```
cyl
4    24.400000
6    20.166667
8    14.300000
Name: mpg, dtype: float64
```

```
#Use fillna()
df_mtcars["mpg"] = df_mtcars.groupby(by = ["cyl"]).mpg.transform(lambda x: x.fillna(1))
df_mtcars
```

	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	1.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	1.0	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	1.0	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

Filled each NaN with 1

Using a Transform

```
df_mtcars.groupby(by = ["cyl"]).mpg.mean()
```

```
cyl
4    24.400000
6    20.166667
8    14.300000
Name: mpg, dtype: float64
```

```
#Use fillna() -  Fill with mean of group
df_mtcars["mpg"] = df_mtcars.groupby(by = ["cyl"]).mpg.transform(lambda x: x.fillna(x.mean()))
df_mtcars
```

	car_name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.000000	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	20.166667	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	24.400000	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.400000	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	14.300000	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.100000	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.300000	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.400000	4	146.7	62	3.69	3.190	20.00	1	0	4	2

Normalizing for Units

```
df_mtcars
```

	car_name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.000000	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	20.166667	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	24.400000	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.400000	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	14.300000	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.100000	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.300000	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.400000	4	146.7	62	3.69	3.190	20.00	1	0	4	2

Hard to tell quickly what is small and large values for these columns

Normalizing for Units

```
#One way to normalize
hp_mean = df_mtcars["hp"].mean()
hp_std = df_mtcars["hp"].std()
df_mtcars["hp_new"] = (df_mtcars["hp"] - hp_mean)/hp_std

df_mtcars
```

	car_name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	hp_new
0	Mazda RX4	21.000000	6	160.0	110	3.90	2.620	16.46	0	1	4	4	-0.283765
1	Mazda RX4 Wag	20.166667	6	160.0	110	3.90	2.875	17.02	0	1	4	4	-0.283765
2	Datsun 710	24.400000	4	108.0	93	3.85	2.320	18.61	1	1	4	1	-0.580627
3	Hornet 4 Drive	21.400000	6	258.0	110	3.08	3.215	19.44	1	0	3	1	-0.283765
4	Hornet Sportabout	14.300000	8	360.0	175	3.15	3.440	17.02	0	0	3	2	0.851295
5	Valiant	18.100000	6	225.0	105	2.76	3.460	20.22	1	0	3	1	-0.371077
6	Duster 360	14.300000	8	360.0	245	3.21	3.570	15.84	0	0	3	4	2.073668
7	Merc 240D	24.400000	4	146.7	62	3.69	3.190	20.00	1	0	4	2	-1.121964

Standard deviations from the mean

Normalizing for Units

```
#One way to normalize
hp_mean = df_mtcars["hp"].mean()
hp_std = df_mtcars["hp"].std()
df_mtcars["hp_new"] = (df_mtcars["hp"] - hp_mean)/hp_std

df_mtcars
```

	car_name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	hp_new
0	Mazda RX4	21.000000	6	160.0	110	3.90	2.620	16.46	0	1	4	4	-0.283765
1	Mazda RX4 Wag	20.166667	6	160.0	110	3.90	2.875	17.02	0	1	4	4	-0.283765
2	Datsun 710	24.400000	4	108.0	93	3.85	2.320	18.61	1	1	4	1	-0.580627
3	Hornet 4 Drive	21.400000	6	258.0	110	3.08	3.215	19.44	1	0	3	1	-0.283765
4	Hornet Sportabout	14.300000	8	360.0	175	3.15	3.440	17.02	0	0	3	2	0.851295
5	Valiant	18.100000	6	225.0	105	2.76	3.460	20.22	1	0	3	1	-0.371077
6	Duster 360	14.300000	8	360.0	245	3.21	3.570	15.84	0	0	3	4	2.073668
7	Merc 240D	24.400000	4	146.7	62	3.69	3.190	20.00	1	0	4	2	-1.121964

Standard deviations from the mean

What if I want to normalize based on groups?

Normalizing for Units

```
#Normalize with transform
df_mtcars["hp_new"] = df_mtcars.groupby(by = ["cyl"]).hp.transform(lambda x: (x-x.mean())/x.std())

df_mtcars
```

	car_name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	hp_new
0	Mazda RX4	21.000000	6	160.0	110	3.90	2.620	16.46	0	1	4	4	0.500000
1	Mazda RX4 Wag	20.166667	6	160.0	110	3.90	2.875	17.02	0	1	4	4	0.500000
2	Datsun 710	24.400000	4	108.0	93	3.85	2.320	18.61	1	1	4	1	0.707107
3	Hornet 4 Drive	21.400000	6	258.0	110	3.08	3.215	19.44	1	0	3	1	0.500000
4	Hornet Sportabout	14.300000	8	360.0	175	3.15	3.440	17.02	0	0	3	2	-0.707107
5	Valiant	18.100000	6	225.0	105	2.76	3.460	20.22	1	0	3	1	-1.500000
6	Duster 360	14.300000	8	360.0	245	3.21	3.570	15.84	0	0	3	4	0.707107
7	Merc 240D	24.400000	4	146.7	62	3.69	3.190	20.00	1	0	4	2	-0.707107

Normalize based on mean and stdev for each group.

Can we use apply()?

Normalizing for Units – Apply()

```
def standardize(group):
    return pd.DataFrame({"hp_standardized": (group["hp"] - group["hp"].mean()) / group["hp"].std()})

df_mtcars["hp_standardized"] = df_mtcars.groupby(by=["cyl"]).apply(standardize)
df_mtcars.head()
```

	car_name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	text	hp_standardized
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	-0.506408	-0.506408
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	-0.506408	-0.506408
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	0.495050	0.495050
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	-0.506408	-0.506408
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	-0.671173	-0.671173

Groupby in Pandas - Filtering

Looking at Modified mtcars

```
import pandas as pd
import numpy as np

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

	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 if I only want to look at certain groups…

- Groups that have “enough” data
- Groups whose mean is above a certain value.

Filtering

```
#Look at groups of cyl that have at least 3 cars
df_mtcars.groupby(by = ["cyl"]).filter(lambda x: len(x)>=3)
```

	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
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1

Function you pass to filter must return a boolean: True for keeping the group and False for getting rid of it.

Filtering

```
#Look at groups of cyl that have at least 3 cars
df_mtcars.groupby(by = ["cyl"]).filter(lambda x: len(x)>=3)
```

	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
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1

```
#Group by am, kep cars whos mpg is above the mean mpg of all cars
df_mtcars.groupby(by = ["am"]).filter(lambda x: x.mpg.mean() >= df_mtcars.mpg.mean())
```

	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

```
df_mtcars.mpg.mean()
```

```
20.212500000000002
```

Filtering + Aggregate

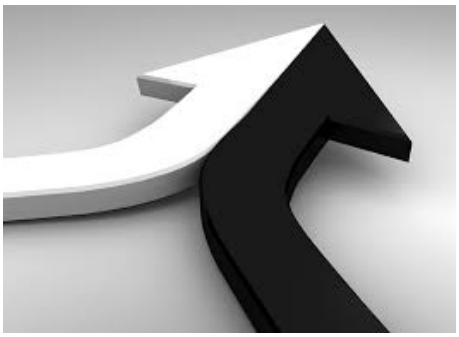
```
#Group by carb, but only group with avg hp >=110
df_mtcars.groupby("carb").filter(lambda x: x.hp.mean()>=110)
```

	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
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
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

```
#Group by carb, but only group with avg hp >=110 +
#avg mpg when then grouped by am
df_mtcars.groupby("carb").filter(lambda x: x.hp.mean()>=110).\
            groupby("am").mpg.mean()
```

```
am
0    19.133333
1    21.000000
Name: mpg, dtype: float64
```

Merges



Dennis J. Zhang
Washington University in St. Louis

Toy Soccer Data Set

`df_players`

	Player	Team	Goals	Assists
0	Milner	Liverpool	2	12
1	Coutinho	Liverpool	5	4
2	Kane	Tottenham	10	4
3	Son	Tottenham	6	10
4	Rooney	Everton	4	7
5	Baines	Everton	1	1
6	Hazard	Chelsea	7	8

`df_teams`

	Team	Home_City	Num_Fans
0	Liverpool	Liverpool	0.2
1	Tottenham	London	2.3
2	Everton	Liverpool	0.1
3	Chelsea	London	5.1

`df_city`

	City	Population
0	London	9.0
1	Liverpool	0.5

- What fraction of the population does each team's fan base make up?
- How many goals are scored by players in London?

Basic JOIN Operation

What fraction of the population does each team's fan base make up?

`df_teams`

	Team	Home_City	Num_Fans
0	Liverpool	Liverpool	0.2
1	Tottenham	London	2.3
2	Everton	Liverpool	0.1
3	Chelsea	London	5.1

`df_city`

	City	Population
0	London	9.0
1	Liverpool	0.5

First we need to add a population column to df_teams

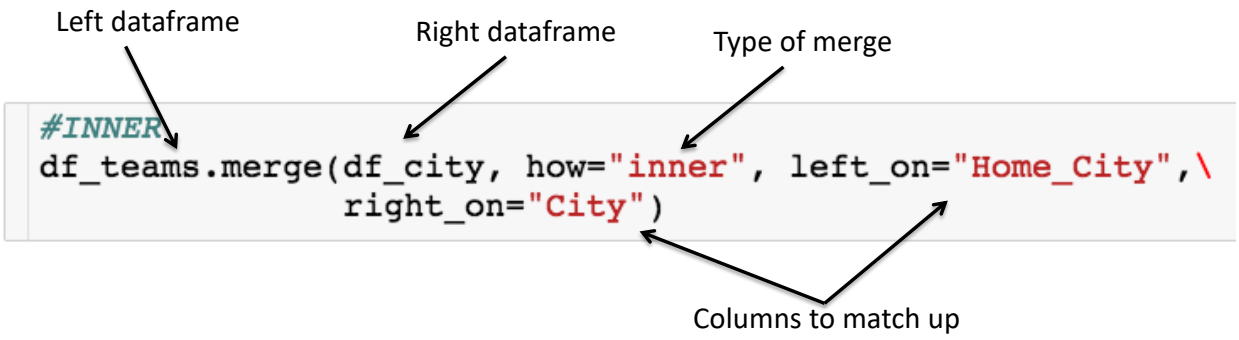
Inner Merge Operation

df_teams

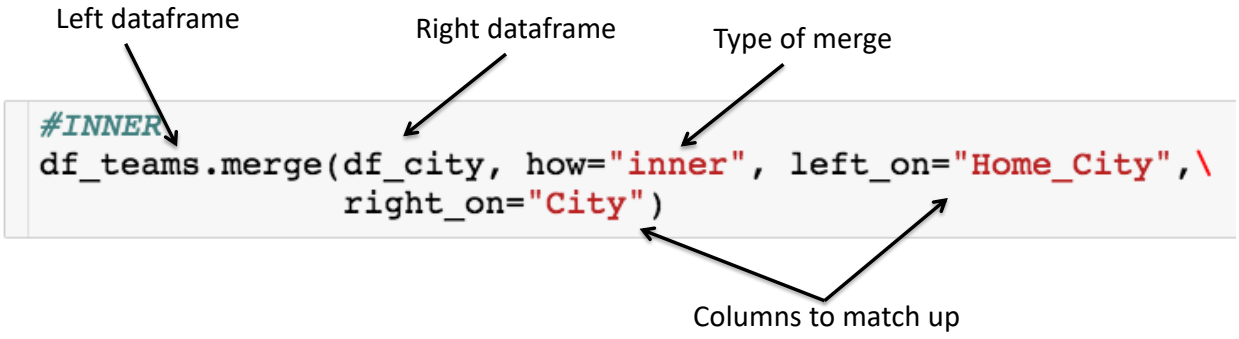
	Team	Home_City	Num_Fans
0	Liverpool	Liverpool	0.2
1	Tottenham	London	2.3
2	Everton	Liverpool	0.1
3	Chelsea	London	5.1

df_city

	City	Population
0	London	9.0
1	Liverpool	0.5



Inner Merge Operation



	Team	Home_City	Num_Fans	City	Population
0	Liverpool	Liverpool	0.2	Liverpool	0.5
1	Everton	Liverpool	0.1	Liverpool	0.5
2	Tottenham	London	2.3	London	9.0
3	Chelsea	London	5.1	London	9.0

Inner Merge Operation

What happened…

```
#INNER
df_teams.merge(df_city, how="inner", left_on="Home_City",\
               right_on="City")
```

Columns to match up

	Team	Home_City	Num_Fans	City	Population
0	Liverpool	Liverpool	0.2	London	9.0
1	Liverpool	Liverpool	0.2	Liverpool	0.5
2	Tottenham	London	2.3	London	9.0
3	Tottenham	London	2.3	Liverpool	0.5
4	Everton	Liverpool	0.1	London	9.0
5	Everton	Liverpool	0.1	Liverpool	0.5
6	Chelsea	London	5.1	London	9.0
7	Chelsea	London	5.1	Liverpool	0.5

Inner Merge Operation

What happened…Only keep columns where Home_City = City

```
#INNER
df_teams.merge(df_city, how="inner", left_on="Home_City",\
               right_on="City")
```

Columns to match up

	Team	Home_City	Num_Fans	City	Population
0	Liverpool	Liverpool	0.2	London	9.0
1	Liverpool	Liverpool	0.2	Liverpool	0.5
2	Tottenham	London	2.3	London	9.0
3	Tottenham	London	2.3	Liverpool	0.5
4	Everton	Liverpool	0.1	London	9.0
5	Everton	Liverpool	0.1	Liverpool	0.5
6	Chelsea	London	5.1	London	9.0
7	Chelsea	London	5.1	Liverpool	0.5

Answer Question 1

What fraction of the population does each team's fan base make up?

```
#Select columns I want
new_df = df_teams.merge(df_city, how="inner", left_on="Home_City",\
right_on="City").loc[:, ["Team", "City", "Num_Fans", "Population"]]

#Make new Column
new_df["Frac"] = new_df["Num_Fans"]/new_df["Population"]
new_df
```

	Team	City	Num_Fans	Population	Frac
0	Liverpool	Liverpool	0.2	0.5	0.400000
1	Everton	Liverpool	0.1	0.5	0.200000
2	Tottenham	London	2.3	9.0	0.255556
3	Chelsea	London	5.1	9.0	0.566667

Inner Merge Operation

How many goals are scored by players in London?

```
df_players
```

	Player	Team	Goals	Assists
0	Milner	Liverpool	2	12
1	Coutinho	Liverpool	5	4
2	Kane	Tottenham	10	4
3	Son	Tottenham	6	10
4	Rooney	Everton	4	7
5	Baines	Everton	1	1
6	Hazard	Chelsea	7	8

```
df_teams
```

	Team	Home_City	Num_Fans
0	Liverpool	Liverpool	0.2
1	Tottenham	London	2.3
2	Everton	Liverpool	0.1
3	Chelsea	London	5.1

Inner Merge Operation

	Player	Team	Goals	Assists	Team	Home_City	Num_Fans
0	Milner	Liverpool	2	12	Liverpool	Liverpool	0.2
1	Milner	Liverpool	2	12	Tottenham	London	2.3
2	Milner	Liverpool	2	12	Everton	Liverpool	0.1
3	Coutinho	Liverpool	5	4	Liverpool	Liverpool	0.2
4	Coutinho	Liverpool	5	4	Tottenham	London	2.3
5	Coutinho	Liverpool	5	4	Everton	Liverpool	0.1
6	Kane	Tottenham	10	4	Liverpool	Liverpool	0.2
7	Kane	Tottenham	10	4	Tottenham	London	2.3
8	Kane	Tottenham	10	4	Everton	Liverpool	0.1
9	Son	Tottenham	6	10	Liverpool	Liverpool	0.2
10	Son	Tottenham	6	10	Tottenham	London	2.3
11	Son	Tottenham	6	10	Everton	Liverpool	0.1

1. What do I merge on?

2. Which columns do I need?

Inner Merge Operation

```
#INNER
new_df = df_players.merge(df_teams, how="inner", on = "Team").\
loc[:, ["Home_City", "Goals"]]

new_df
```

	Home_City	Goals
0	Liverpool	2
1	Liverpool	5
2	London	10
3	London	6
4	Liverpool	4
5	Liverpool	1
6	London	7

Now I just need to pick out the rows that correspond to a London based player

Answering Question 2

How many goals are scored by players in London?

```
#INNER
new_df = df_players.merge(df_teams, how="inner", on = "Team").\
loc[:, ["Home_City", "Goals"]]

london_goals = new_df.loc[new_df["Home_City"] == "London",\
                          "Goals"].sum()
london_goals
```

```
23
```

Answering Question 2

How many goals are scored by players in London?

```
#INNER
new_df = df_players.merge(df_teams, how="inner", on = "Team").\
loc[:, ["Home_City", "Goals"]]

london_goals = new_df.loc[new_df["Home_City"] == "London",\
                          "Goals"].sum()
london_goals
```

```
23
```

```
#One line
df_players.merge(df_teams, how="inner", on = "Team").\
                loc[:, ["Home_City", "Goals"]].\
                loc[new_df["Home_City"] == "London","Goals"].sum()
```

```
23
```

Left Merge Operation

`df_players`

	Player	Team	Goals	Assists
0	Milner	Liverpool	2	12
1	Coutinho	Liverpool	5	4
2	Kane	Tottenham	10	4
3	Son	Tottenham	6	10
4	Rooney	Everton	4	7
5	Baines	Everton	1	1
6	Hazard	Chelsea	7	8

Notice now that Chelsea is missing from the df_tems dataframe

`df_teams`

	Team	Home_City	Num_Fans
0	Liverpool	Liverpool	0.2
1	Tottenham	London	2.3
2	Everton	Liverpool	0.1

Left Merge Operation

`df_players`

	Player	Team	Goals	Assists
0	Milner	Liverpool	2	12
1	Coutinho	Liverpool	5	4
2	Kane	Tottenham	10	4
3	Son	Tottenham	6	10
4	Rooney	Everton	4	7
5	Baines	Everton	1	1
6	Hazard	Chelsea	7	8

Let's see what happens when we do a normal JOIN on Team…

`df_teams`

	Team	Home_City	Num_Fans
0	Liverpool	Liverpool	0.2
1	Tottenham	London	2.3
2	Everton	Liverpool	0.1

Left Merge Operation

Let's see what happens when we do a normal JOIN on Team…

```
df_players.merge(df_teams, how = "inner", on ="Team")
```

	Player	Team	Goals	Assists	Home_City	Num_Fans
0	Milner	Liverpool	2	12	Liverpool	0.2
1	Coutinho	Liverpool	5	4	Liverpool	0.2
2	Kane	Tottenham	10	4	London	2.3
3	Son	Tottenham	6	10	London	2.3
4	Rooney	Everton	4	7	Liverpool	0.1
5	Baines	Everton	1	1	Liverpool	0.1

We lose the information on Hazard (Chelsea) because Chelsea wasn't matched with the on!

Left Merge Operation

7	Kane	Tottenham	10	4	Tottenham	London	✓
8	Kane	Tottenham	10	4	Everton	Liverpool	
9	Son	Tottenham	6	10	Liverpool	Liverpool	✓
10	Son	Tottenham	6	10	Tottenham	London	
11	Son	Tottenham	6	10	Everton	Liverpool	
12	Rooney	Everton	4	7	Liverpool	Liverpool	✓
13	Rooney	Everton	4	7	Tottenham	London	
14	Rooney	Everton	4	7	Everton	Liverpool	
15	Baines	Everton	1	1	Liverpool	Liverpool	✓
16	Baines	Everton	1	1	Tottenham	London	
17	Baines	Everton	1	1	Everton	Liverpool	
18	Hazard	Chelsea	7	8	Liverpool	Liverpool	✗
19	Hazard	Chelsea	7	8	Tottenham	London	✗
20	Hazard	Chelsea	7	8	Everton	Liverpool	✗

Left Merge Operation

7	Kane	Tottenham	10	4	Tottenham	London
8	Kane	Tottenham	10	4	Everton	Liverpool
9	Son	Tottenham	6	10	Liverpool	Liverpool
10	Son	Tottenham	6	10	Tottenham	London
11	Son	Tottenham	6	10	Everton	Liverpool
12	Rooney	Everton	4	7	Liverpool	Liverpool
13	Rooney	Everton	4	7	Tottenham	London
14	Rooney	Everton	4	7	Everton	Liverpool
15	Baines	Everton	1	1	Liverpool	Liverpool
16	Baines	Everton	1	1	Tottenham	London
17	Baines	Everton	1	1	Everton	Liverpool
~~**18**~~	~~Hazard~~	~~Chelsea~~	~~7~~	~~8~~	~~Liverpool~~	~~Liverpool~~
~~**19**~~	~~Hazard~~	~~Chelsea~~	~~7~~	~~8~~	~~Tottenham~~	~~London~~
~~**20**~~	~~Hazard~~	~~Chelsea~~	~~7~~	~~8~~	~~Everton~~	~~Liverpool~~

Left Merge Operation

7	Kane	Tottenham	10	4	Tottenham	London
8	Kane	Tottenham	10	4	Everton	Liverpool
9	Son	Tottenham	6	10	Liverpool	Liverpool
10	Son	Tottenham	6	10	Tottenham	London
11	Son	Tottenham	6	10	Everton	Liverpool
12	Rooney	Everton	4	7	Liverpool	Liverpool
13	Rooney	Everton	4	7	Tottenham	London
14	Rooney	Everton	4	7	Everton	Liverpool
15	Baines	Everton	1	1	Liverpool	Liverpool
16	Baines	Everton	1	1	Tottenham	London
17	Baines	Everton	1	1	Everton	Liverpool
~~**18**~~	~~Hazard~~	~~Chelsea~~	~~7~~	~~8~~	~~Liverpool~~	~~Liverpool~~
~~**19**~~	~~Hazard~~	~~Chelsea~~	~~7~~	~~8~~	~~Tottenham~~	~~London~~
~~**20**~~	~~Hazard~~	~~Chelsea~~	~~7~~	~~8~~	~~Everton~~	~~Liverpool~~

Left merge – If there isn't a match still keep a copy of the row for the left dataframe

Left Merge Operation

Specify “left”

```
df_players.merge(df_teams, how = "left", on ="Team")
```

	Player	Team	Goals	Assists	Home_City	Num_Fans
0	Milner	Liverpool	2	12	Liverpool	0.2
1	Coutinho	Liverpool	5	4	Liverpool	0.2
2	Kane	Tottenham	10	4	London	2.3
3	Son	Tottenham	6	10	London	2.3
4	Rooney	Everton	4	7	Liverpool	0.1
5	Baines	Everton	1	1	Liverpool	0.1
6	Hazard	Chelsea	7	8	NaN	NaN

Get NaN for columns of right df

Left Merge Operation

Specify “left”

```
df_players.merge(df_teams, how = "left", on ="Team")
```

	Player	Team	Goals	Assists	Home_City	Num_Fans
0	Milner	Liverpool	2	12	Liverpool	0.2
1	Coutinho	Liverpool	5	4	Liverpool	0.2
2	Kane	Tottenham	10	4	London	2.3
3	Son	Tottenham	6	10	London	2.3
4	Rooney	Everton	4	7	Liverpool	0.1
5	Baines	Everton	1	1	Liverpool	0.1
6	Hazard	Chelsea	7	8	NaN	NaN

Get NaN for columns of right df

- Useful if you don’t want to lose data after the ON
- There is a RIGHT JOIN … but any RIGHT JOIN can be written as LEFT JOIN

New Data Set

```
df_rush
```

	Week	RushPoints
0	1	12
1	2	32
2	3	34
3	4	12

```
df_rec
```

	Week	RecPoints
0	2	101
1	3	310
2	5	234

- Lets say I wanted to combine the records and put a zero if one of the points categories did not exists for the given week.
- An inner merge on Week will get rid of rows 1,4 (from df_rush) and row 5 (from df_rec)
- A left join on Week will get rid of row 5 (from df_rec)

Outer Merge Operation

Specify “outer”

```
df_rush.merge(df_rec, how = "outer", on = "Week")
```

	Week	RushPoints	RecPoints
0	1	12.0	NaN
1	2	32.0	101.0
2	3	34.0	310.0
3	4	12.0	NaN
4	5	NaN	234.0

Get NaN when either side can’t find a match

Replace NaNs

```
#Outer merge
df_outer = df_rush.merge(df_rec, how = "outer", on = "Week")
#Replace NaNs
df_outer.fillna(0, inplace = True)

df_outer
```

	Week	RushPoints	RecPoints
0	1	12.0	0.0
1	2	32.0	101.0
2	3	34.0	310.0
3	4	12.0	0.0
4	5	0.0	234.0

Putting Everything Together!

- What we have learnt so far?
- Apply():
  - Allows you to write a customized function that takes in
    - an entry (if applied to a column)
    - a dataframe (if applied to a groupby)
    - additional parameters
  - and returns:
    - an entry (the final return is a series)
    - a series (the final return is a row)
    - a dataframe (the final return is multiple rows from groupby)
- Groupby():
  - That allows you to group the data by certain levels and do computation
    - Groupby() + agg()
    - Groupby() + transform()
- Merge():
  - Allows you to merge multiple dataframes