YData: Introduction to Data Science

Class 09: Pandas continued

Overview

Quick review of pandas:

- Tuples and dictionaries
- Series and DataFrames methods

Continuation of pandas:

- Calculating aggregate statistics for separate groups
- Joining DataFrames

If there is time:

- Additional practice!

Announcement: Homework 4

Homework 2 is due on Gradescope on Sunday September 29th at 11pm

- **Be sure to mark each question on Gradescope!**

Quick review: tuples and dictionaries

Tuples are like lists but they are immutable

- my_tuple = (10, 20, 30)   # Creating a tuple
- my_tuple[1]   # accessing items
- my_tuple[1] = 50   # Error!   Tuples are immutable
- val1, val2, val3 = my_tuple # tuple unpacking

Dictionaries allow you to look up ***values*** based on a ***key***

- my_dict = { 'key1': 5, 'key2': 20}
- my_dict['key2']

Review: pandas Series and DataFrames

There are two main data structures in pandas:

- **Series**: represent one-dimensional data
- **DataFrames**: represent data tables
  - i.e., relational data

pandas Series are: One-dimensional ndarray with an Index

- egg_prices.iloc[0]   # use index location
- egg_prices.loc["1980-01-01"] # use Index names

Example: egg _prices

DATE	
1980-01-01	0.879
1980-02-01	0.774
1980-03-01	0.812

Index

values (ndarray)

Review: pandas DataFrames

Pandas DataFrame hold Table data

Variables

Cases

	arr_delay	dep_delay	carrier
tailnum			
N25201	205.0	203.0	UA
N830DN	53.0	78.0	DL
N807JB	34.0	47.0	B6
N265JB	166.0	173.0	B6
N17730	211.0	228.0	UA

Extracting columns:

- my_df["my_col"]   # returns a Series!
- my_df[["my_col"]]   # returns a DataFrame
- my_df[["col1", "col2"]]   # multiple columns

Extracting rows

- my_df.iloc[0:3]   # by position
- my_df.loc["index_name"]   # by index name

Sorting rows

my_df.sort_values("col_name")

my_df.sort_values("col_name",
                 ascending = True)

Warm-up exercises

Warm-up 1: tuples and dictionaries

Warm-up 2: DataFrame operations

- Motivation: The Dow Jones Industrial Average (DJIA or Dow), is a stock market index of 30 prominent companies listed on stock exchanges in the United States.

Date	Year	Month	Day	Open	High	Low	Close	Volume
1992-01-02	1992	1	Thursday	3152.100098	3172.629883	3139.310059	3172.399902	23550000
1992-01-03	1992	1	Friday	3172.399902	3210.639893	3165.919922	3201.500000	23620000
1992-01-06	1992	1	Monday	3201.500000	3213.330078	3191.860107	3200.100098	27280000
1992-01-07	1992	1	Tuesday	3200.100098	3210.199951	3184.479980	3204.800049	25510000
1992-01-08	1992	1	Wednesday	3204.800049	3229.199951	3185.820068	3203.899902	29040000

Let’s try it in Jupyter!

Pandas continued

Adding new columns and renaming columns

We can add a column to a data frame using square backets. For example:

```
my_df["new_col"] = values_array
my_df["new col"] = my_df["col1"] + my_df["col2"]
```

We can rename columns by passing a dictionary to the `.rename()` method.

```
rename_dictionary = {"old_col_name": "new_col_name"}
my_df.rename(columns = rename_dictionary )
```

Let's explore this in Jupyter!

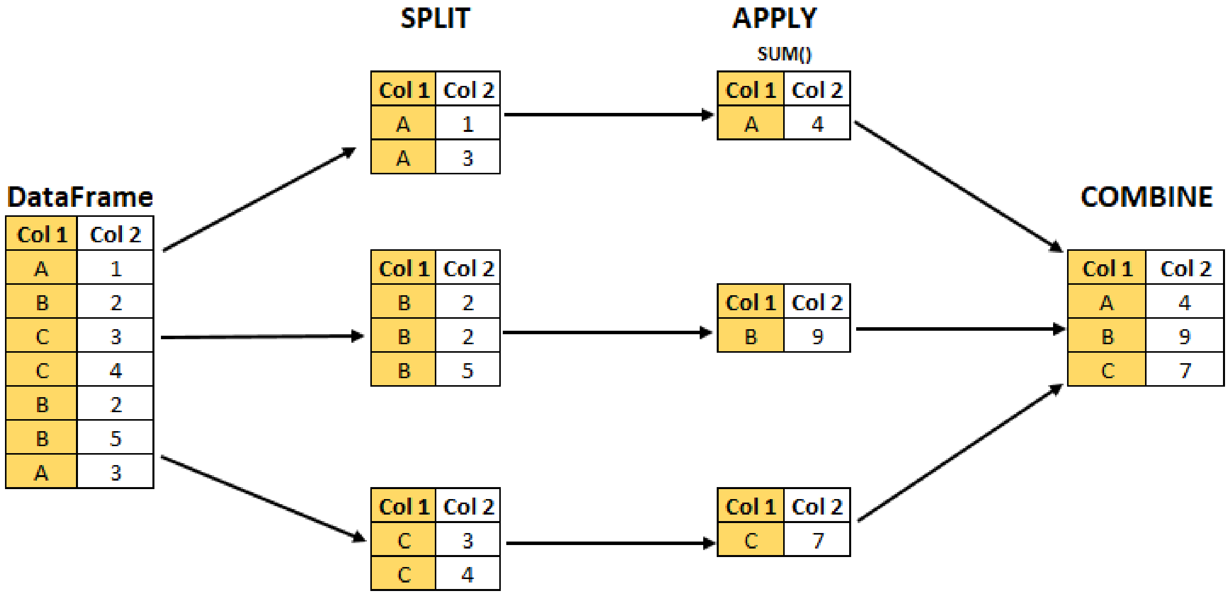
Creating aggregate statistics by group

We can get statistics separately by group using the .groupby() and .agg() methods

- E.g. dow.groupby("Year").agg("max")

This implements:

"Split-apply-combine"



Creating aggregate statistics by group

There are several ways to get multiple statistics by group

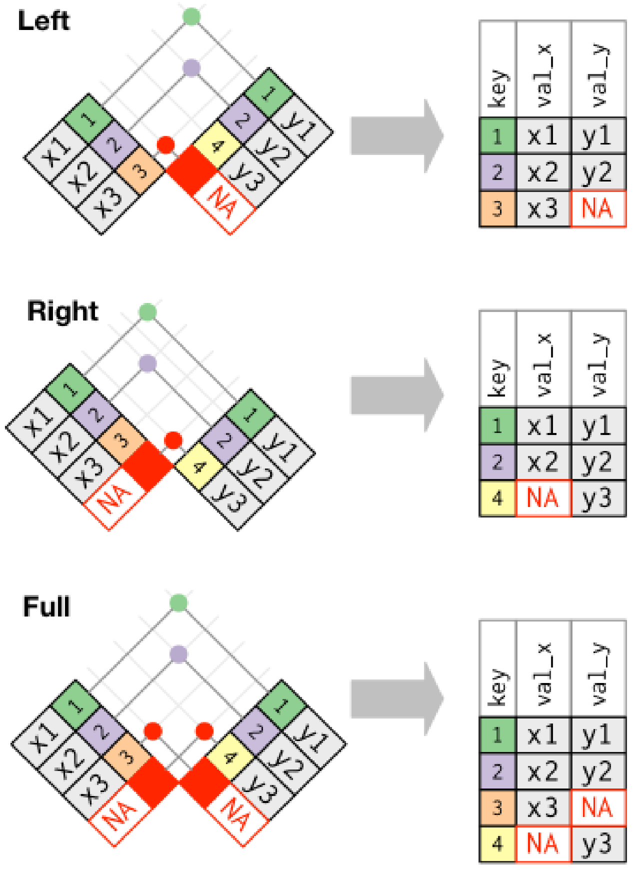
Perhaps the most useful way is to use the syntax:

```
my_df.groupby("group_col_name").agg(
    new_col1 = ('col_name1', 'statistic_name1'),
    new_col2 = ('col_name2', 'statistic_name2'),
    new_col3 = ('col_name3', 'statistic_name3')
)
```

```
nba_salaries.groupby("TEAM").agg(
    max_salary = ("SALARY", "max"),
    min_salary = ("SALARY", "min"),
    first_player = ("PLAYER", "min")
)
```

Let's explore this in Jupyter!

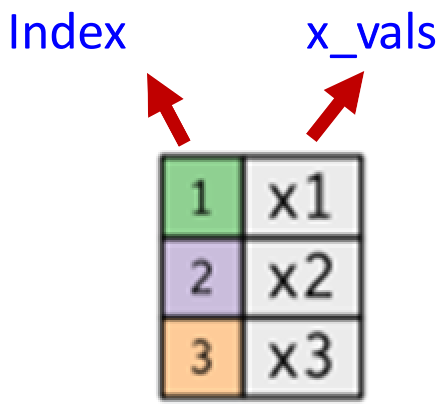
Joining data frames



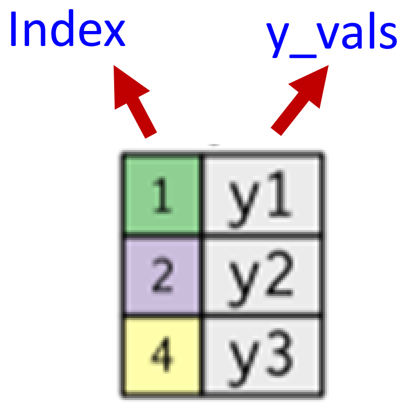
Left and right tables

Suppose we have two DataFrames (or Series) called **x_df** and **y_df**

- x_df have one column called x_vals
- y_df has one column called y_vals



DataFame: x_df



DataFrame: y_df

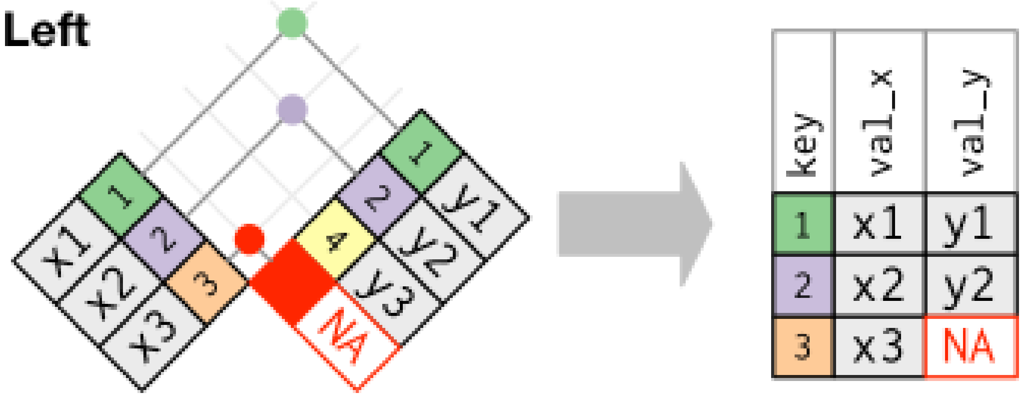
We can join these two DataFrames into a single DataFrame by aligning rows with the same Index value using the general syntax: x_df.join(y_df)

- i.e., the new joined data frame will have two columns: x_vals, and y_vals

Left joins

Left joins keep all rows in the left table.

Data from right table is added when there is a matching Index value, otherwise NA as added

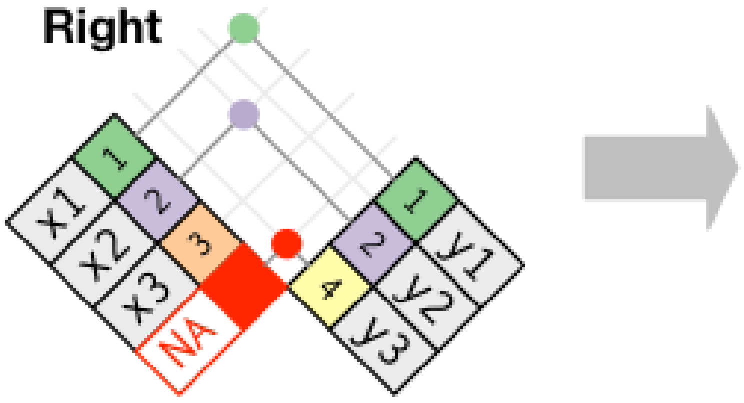


x_df.join(y_df, how = "left”)

Right joins

Right joins keep all rows in the right table.

Data from left table added when there is a matching Index value otherwise NA as added

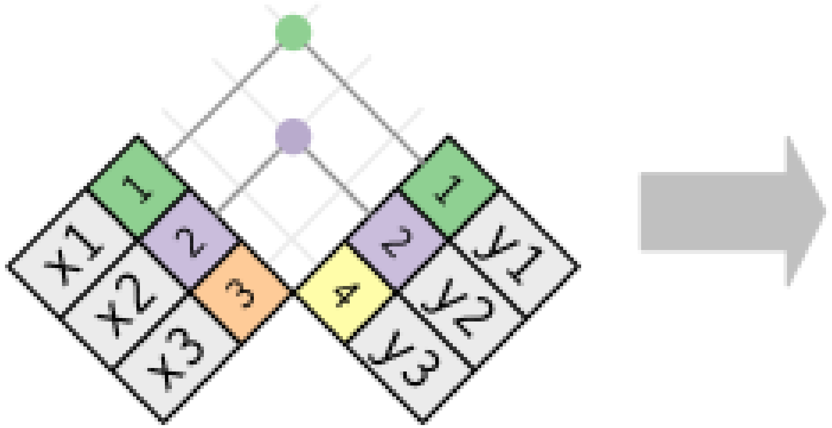


key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

x_df.join(y_df, how = "right")

Inner joins

Inner joins only keep rows in which there are matches between the Index values in both tables.



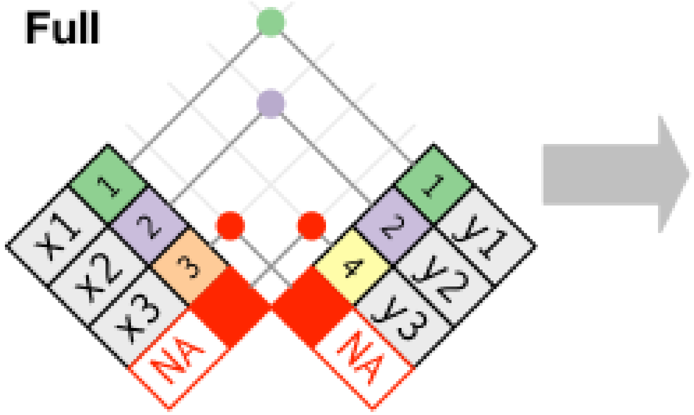
key	val_x	val_y
1	x1	y1
2	x2	y2

x_df.join(y_df, how = "inner")

Full (outer) joins

Full joins keep all rows in both table

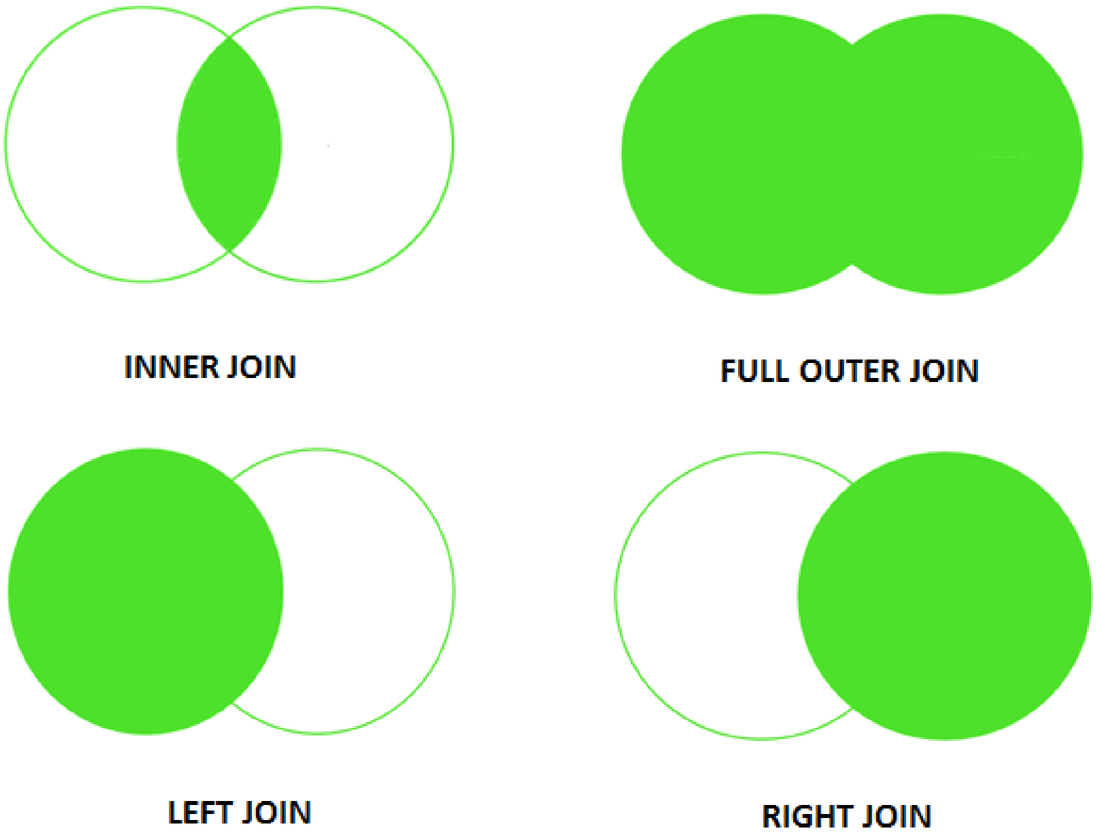
NAs are added where there are no matches



key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

x_df.join(y_df, how = "outer")

Summary



“Merging” data frames

We can also join DataFrames based on values in ***columns*** rather than based on the DataFrames Index values

To do this we can use the merge method which has the form:

- x_df.merge(y_df, how = "left", left_on = "x_col", right_on = "y_col")

All the same types of joins still work

- i.e., we can do: left, right, inner and outer joins

Let's explore this in Jupyter!

Let’s do a few more practice exercises!

Work in pairs to see if you can calculate and visualize how the mean delay differs for:

1. Different hours of the day
2. Months of the year
3. Airport flight left from

If you solve these, see if you can calculate how the mean delay differs by wind speed

- You will need the *nyc23_weather.csv* to solve this