YData: Introduction to Data Science

Class 08: Pandas continued

Overview

Quick review of pandas:

- Tuples and dictionaries
- Series and DataFrames

Continuation of pandas:

- Sorting values and adding new columns
- Calculating aggregate statistics for separate groups
- Joining DataFrames

Announcement: Homework 3

Homework 2 is due on Gradescope on Sunday February 11th 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']

Let's do some very quick warm-up exercise in Jupyter!

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

Example: egg _prices

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

Index

values (ndarray)

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

pandas DataFrames

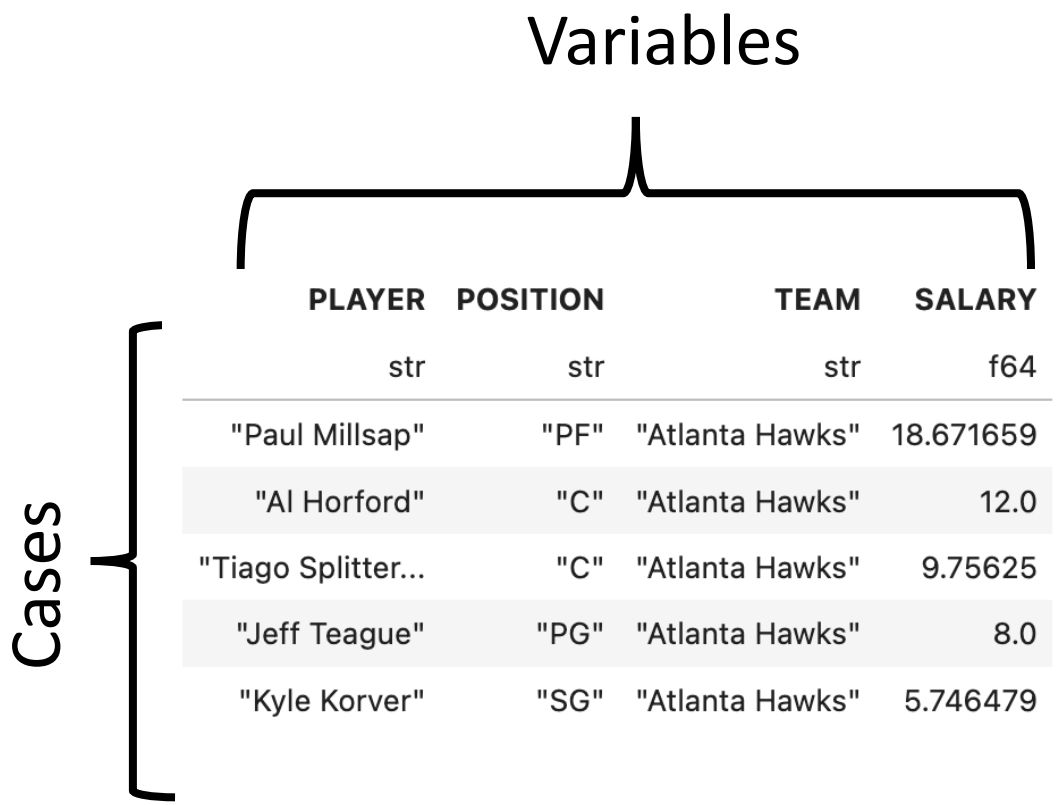
Pandas DataFrame hold Table data

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] # by position
- my_df.loc["index_name"] # by index name



	PLAYER	POSITION	TEAM	SALARY
	str	str	str	f64
	"Paul Millsap"	"PF"	"Atlanta Hawks"	18.671659
	"Al Horford"	"C"	"Atlanta Hawks"	12.0
	"Tiago Splitter...	"C"	"Atlanta Hawks"	9.75625
	"Jeff Teague"	"PG"	"Atlanta Hawks"	8.0
	"Kyle Korver"	"SG"	"Atlanta Hawks"	5.746479

Let’s do a few warm-up exercise in Jupyter!

Sorting rows from a DataFrame

We can sort values in a DataFrame using .sort_values("col_name")

- my_df.sort_values("col_name")

We can sort from highest to lowest by setting the argument ascending = False

- my_df.sort_values("col_name", ascending = False)

Let's explore this in Jupyter!

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!

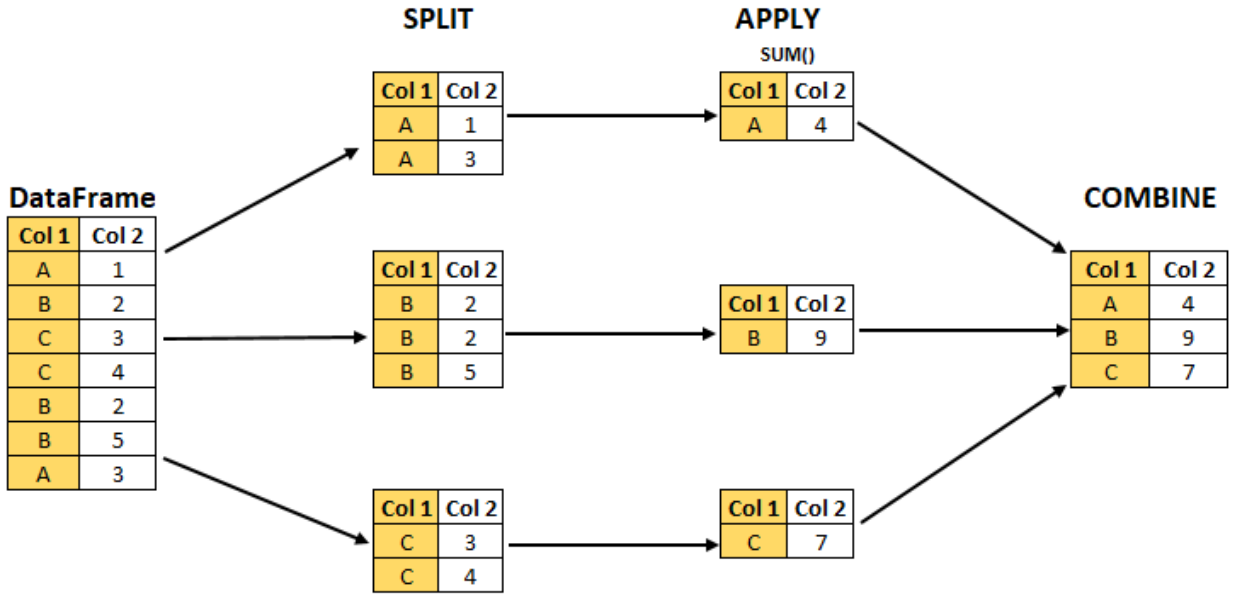
Creating aggregate statistics by group

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

- E.g. nba_salaries.groupby("TEAM").agg("sum")

This implements:

"Split-apply-combine"



Creating aggregate statistics by group

There are several ways to get multiple statistics by group

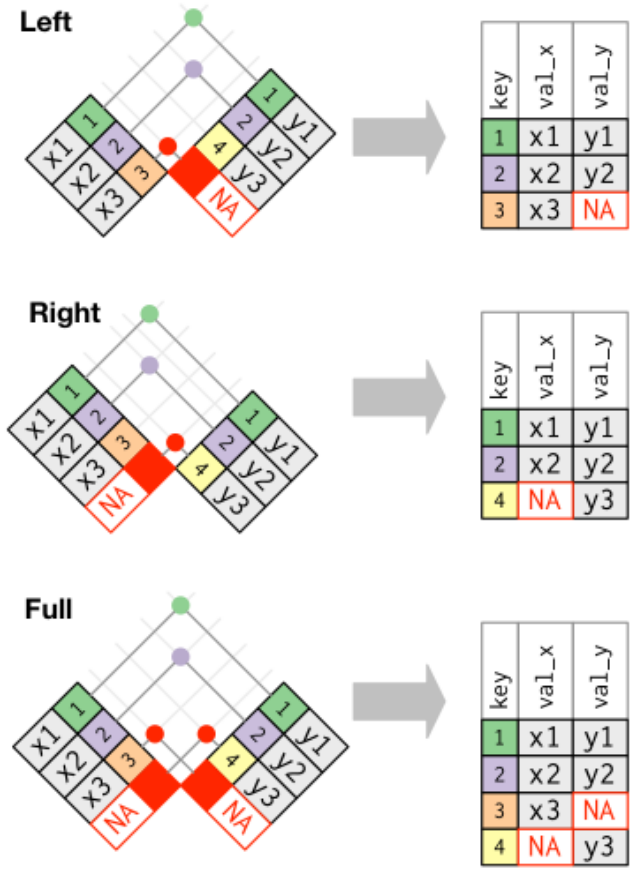
Perhaps the most useful way is to use the syntax:

```
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!

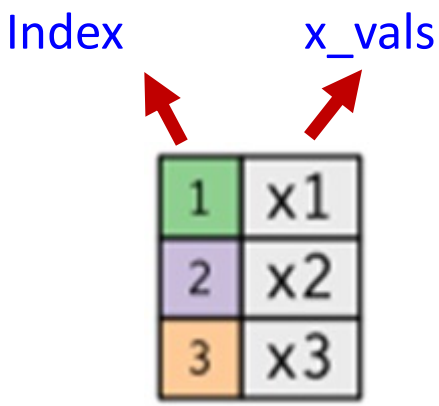
Joining data frames



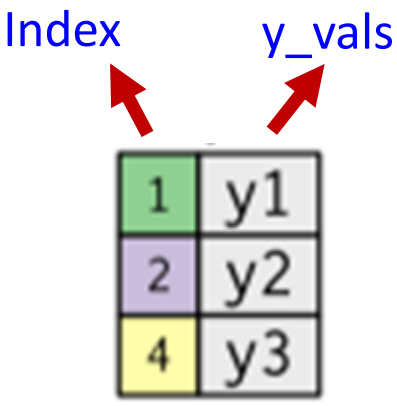
Left and right tables

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

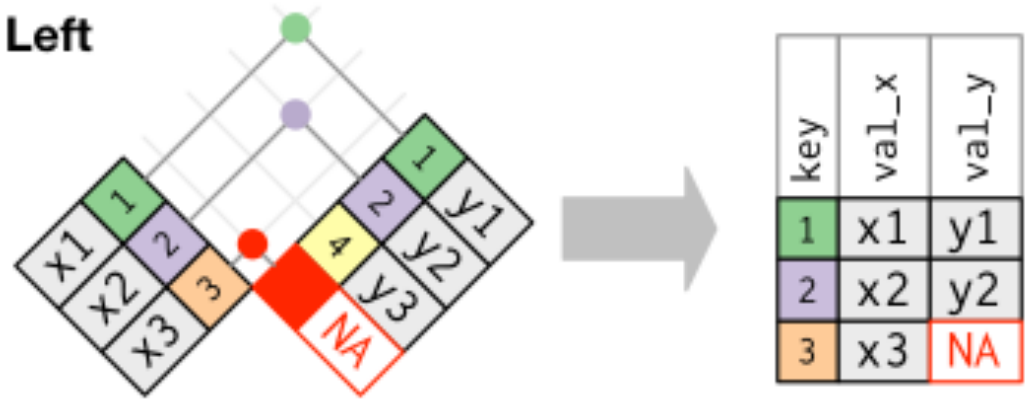
We can join these two DataFrames into a single DataFrame by aligning rows with the same Index value using the general syntax: x_df.join(y_df)

- 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

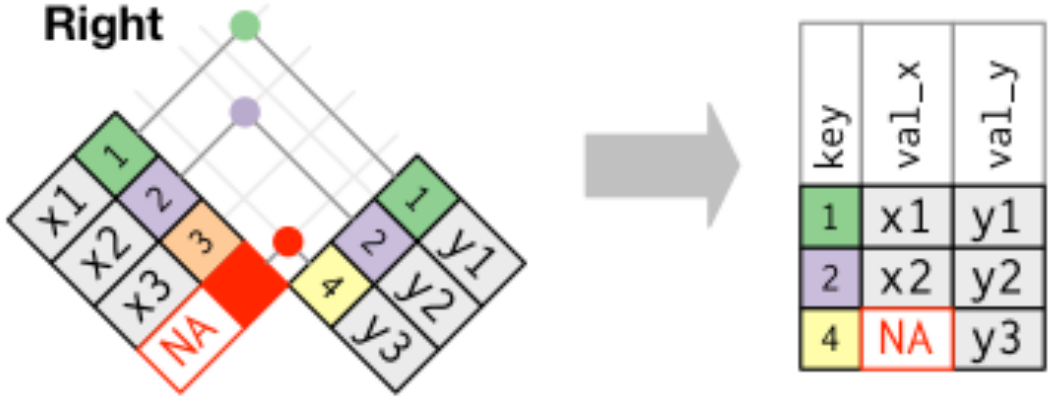


x_df.join(y_df, how = "left”)

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 <u>both</u> tables.

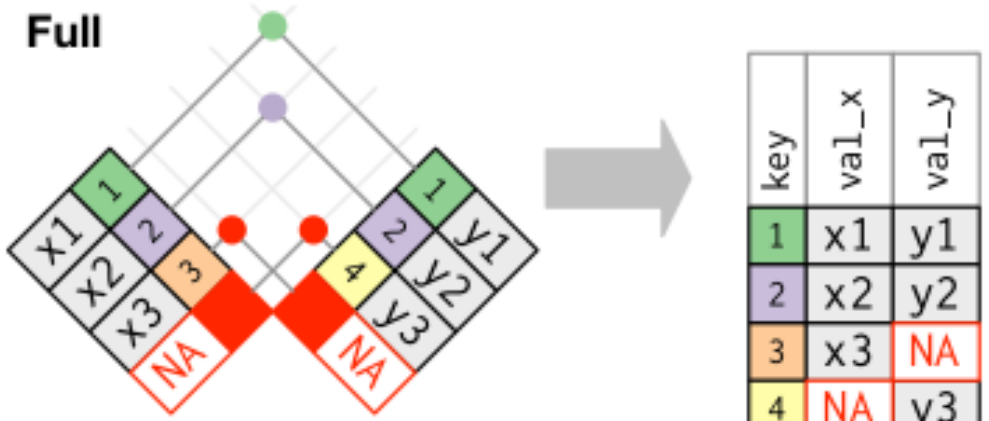
key	val_x	val_y
1	x1	y1
2	x2	y2

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

Full (outer) joins

Full joins keep all rows in both table

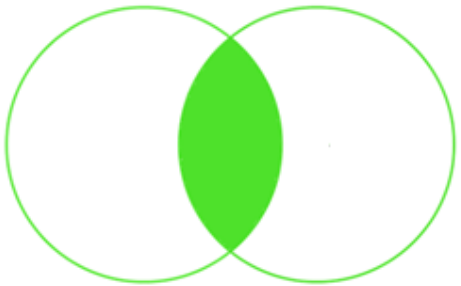
NAs are added where there are no matches



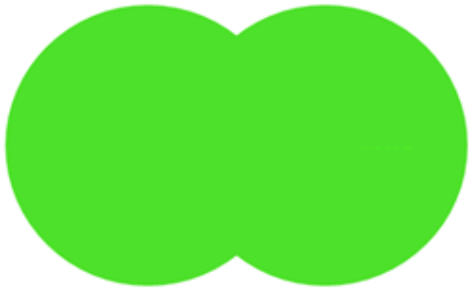
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

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

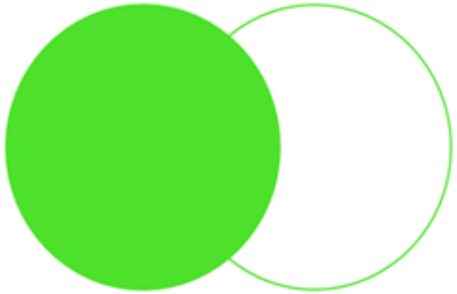
Summary



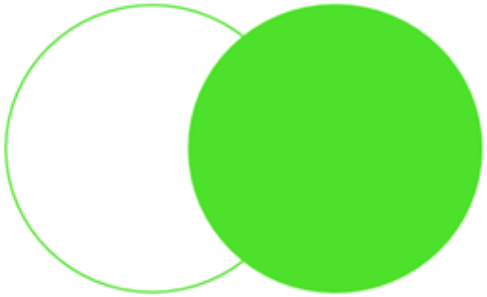
INNER JOIN



FULL OUTER JOIN



LEFT JOIN



RIGHT JOIN

“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!

Questions?