Grouping

Learn how DataFrames can be grouped based on particular columns.

Chapter Goals:

- Learn how to group DataFrames by columns
- Write code to retrieve home run statistics through DataFrame grouping

A. Grouping by column

When dealing with large amounts of data, it is usually a good idea to group the data by common categories. For example, we could group a large dataset of MLB player statistics by year, so we can deal with each year's data separately.

With pandas DataFrames, we can perform dataset grouping with the `groupby (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html)` function. A common usage of the function is to group a DataFrame by values from a particular column, e.g. a column representing years.

The code below shows how to use the `groupby` function, with the example of grouping MLB data by year.

```
1  # Predefined df of MLB stats
2  print('{}\n'.format(df))
3
4  groups = df.groupby('yearID')
5  for name, group in groups:
6    print('Year: {}'.format(name))
7    print('{}\n'.format(group))
8
9  print('{}\n'.format(groups.get_group(2016)))
10 print('{}\n'.format(groups.sum()))
11 print('{}\n'.format(groups.mean()))
```

```
    yearID teamID    H    R
0    2017    CLE  1449  818
1    2015    CLE  1395  669
2    2016    BOS  1598  878
3    2015    DET  1515  689
4    2016    DET  1476  750
5    2016    CLE  1435  777
6    2015    BOS  1495  748
7    2017    BOS  1461  785
```

0.737s

The grouping code example produced three DataFrames for the years 2015, 2016, and 2017. The three DataFrame groups are contained in the `groups` variable, and we used its `sum` and `mean` functions to retrieve the total and average per-year statistics.

In addition to aggregation functions like `sum` and `mean`, we can also filter the groups using `filter`. The `filter` function takes in another function as its required argument, which specifies how we want to filter the groups. The output of `filter` is the concatenation of all the groups that pass the filter.

The code below shows how to use the `filter` function.

```
1  no2015 = groups.filter(lambda x: x.name > 2015)
2  print(no2015)
```

Output

0.648s

```
      yearID teamID     H    R
0     2017    CLE   1449  818
2     2016    BOS   1598  878
4     2016    DET   1476  750
5     2016    CLE   1435  777
7     2017    BOS   1461  785
8     2017    DET   1435  735
```

In the above code example, the lambda function passed into `filter` returns `True` if the group (represented as `x`) represents a year greater than 2015. The output is the concatenation of the 2016 and 2017 groups.

B. Multiple columns

DataFrame grouping is not just limited to a single column. Rather than passing a single column label into `groupby`, we can use a list of column labels to specify grouping by multiple columns.

Grouping by multiple columns can be useful when multiple data features have many different values. For example, if our dataset consisted of MLB players, grouping by both team and year would give us an organized way to view a team's roster throughout the years.

```
1  # player_df is predefined
2  groups = player_df.groupby(['yearID', 'teamID'])
3
4  for name, group in groups:
5    print('Year, Team: {}'.format(name))
6    print('{}\n'.format(group))
7
8  print(groups.sum())
```

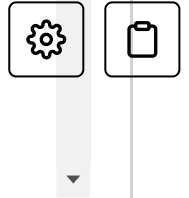Output   0.646s

```
Year, Team: (2016, 'BOS')
    yearID    playerID teamID    H
1    2016  bettsmo01     BOS  214
```

```
2    2016  bogaexa01    BOS  192
4    2016  pedrodu01    BOS  201

Year, Team: (2016, 'HOU')
   yearID   playerID teamID    H
```

In the code above, we grouped the MLB data by both year and team, resulting in each group's name being a tuple of year and team. Using the `sum` function, we obtained the annual total hits for each team.

Time to Code!

The coding exercises for this chapter involve performing grouping operations on `df`, which contains all MLB batting data from 1871-2017. Using `df`, our goal is to retrieve home run (HR) statistics for 2017.

To do this, we need to calculate the total number of home runs hit each year. This involves first grouping `df` by year.

Set `year_group` equal to `df.groupby` with `'yearID'` as the lone argument.

```
1   # CODE HERE
```

Solution

```
1   year_group = df.groupby('yearID')
2
3
4
```

The yearly stats can be obtained from summing the values across the year-separated groups.

Set `year_stats` equal to `year_group.sum` applied with no arguments.

```
1   # CODE HERE
```

```
1   # CODE HERE
```

Solution

```
1   year_stats = year_group.sum()
2
3
4
```

The `year_stats` DataFrame represents the total value for each stat per year. The row labels are the years and the column labels are the stat categories, e.g. home runs. Using the `loc` property, we'll retrieve the home run total for 2017.

Set `hr_2017` equal to `year_stats.loc` with `2017` as the row index and `'HR'` as the column index.

```
1   # CODE HERE
```

Solution

```
1   hr_2017 = year_stats.loc[2017, 'HR']
2
3
4
```

Next we want to get the yearly totals for each batting statistic *per team*. To do this, we group the data by both the year and team.

Set `year_team_group` equal to `df.groupby` applied with the list `['yearID', 'teamID']`.

```
1   # CODE HERE
2
```

```
1   year_team_group = df.groupby(['yearID', 'teamID'])
2
3
4
```

Once again, to obtain the yearly stats we just sum over all the groups.

Set `year_team_stats` equal to `year_team_group.sum` applied with no arguments.

```
1   # CODE HERE
2
```

Solution

```
1   year_team_stats = year_team_group.sum()
2
3
4
```

← **Back**

Next →

✔ **Mark as Completed**

20% completed, meet the criteria and claim your course certificate!

Buy Certificate

Report an Issue

⚠ Ask a Question (https://discuss.educative.io/tag/grouping__data-analysis-with-pandas__machine-learning-for-software-engineers)