

Group by

Group by

- Goal: compute aggregate measures by categories
- Examples:
 - For each program, count the students with a full-time job
 - For each job situation, compute the proportion of students who know Java

Today's data set

- cleaned_survey.csv

Df.groupby(by=column)

- groupby returns a DataFrameGroupBy object
- This object has various aggregating methods, like mean, min, max, etc
- An aggregating method returns a DataFrame
 - Whose index is the column we grouped by, and
 - Whose columns report the aggregate value within the group

```
gb = df.groupby(by='Program')  
gb.mean()
```

| | Job | ProgSkills | C |
|-------------------|------|------------|------|
| Program | | | |
| Business Man | 1.00 | 1.00 | 0.00 |
| Faculty! | 1.00 | 3.00 | 1.00 |
| MBA | 0.66 | 2.50 | 0.31 |
| MSIS | 0.20 | 3.08 | 0.70 |
| Master of Finance | 0.00 | 4.00 | 1.00 |
| Supply | | | |

The average of 'C'
among MBAs is 0.31

Most common aggregate functions

- `count`: number of non-null values
- `size`: number of values including nulls
- `mean`: mean of non-null values
- `min`: min among non-null values
- `max`: max among non-null values
- `sum`: sum of non-null values

Group by in slow motion – step 1: SPLIT

```
gb = df.groupby(by='Program')  
gb.mean()
```

← STEP 1: Split by program. This operation does not compute anything (very fast)

We can think of these partitions as DataFrames (will be useful later)

| | Job | Program | ProgSkills | C | CPP |
|----|-----|----------|------------|---|-----|
| 16 | 1.0 | Faculty! | 3 | 1 | 0 |

| | | | | | |
|----|-----|-------------------------------|---|---|---|
| 13 | 0.0 | Supply Chain Mgmt & Analytics | 2 | 1 | 1 |
| 5 | 1.0 | Supply Chain Mgmt & Analytics | 1 | 0 | 0 |

| | | | | | |
|----|-----|------|---|---|---|
| 21 | 0.0 | MSIS | 2 | 0 | 0 |
| 23 | 0.5 | MSIS | 3 | 1 | 1 |
| 20 | 0.5 | MSIS | 2 | 0 | 0 |
| 25 | 0.0 | MSIS | 3 | 1 | 1 |
| 19 | 0.0 | MSIS | 5 | 1 | 0 |
| 26 | 0.0 | MSIS | 2 | 0 | 0 |
| 27 | 0.5 | MSIS | 3 | 1 | 1 |
| 0 | 0.0 | MSIS | 4 | 1 | 1 |
| 12 | 0.5 | MSIS | 4 | 1 | 1 |
| 9 | 0.5 | MSIS | 3 | 1 | 0 |
| 7 | 0.0 | MSIS | 2 | 1 | 0 |
| 6 | 0.0 | MSIS | 3 | 1 | 1 |
| 4 | 0.0 | MSIS | 3 | 1 | 0 |
| 3 | 0.0 | MSIS | 3 | 1 | 0 |
| 2 | 0.0 | MSIS | 3 | 0 | 0 |
| 1 | 0.5 | MSIS | 3 | 1 | 1 |
| 28 | 0.5 | MSIS | 3 | 1 | 1 |
| 29 | 0.0 | MSIS | 3 | 1 | 1 |

| | | | | | |
|----|-----|-----|---|---|---|
| 17 | 1.0 | MBA | 4 | 0 | 0 |
| 18 | 0.5 | MBA | 2 | 0 | 0 |
| 11 | 1.0 | MBA | 2 | 0 | 0 |
| 10 | 0.5 | MBA | 4 | 1 | 1 |
| 15 | 1.0 | MBA | 1 | 0 | 0 |
| 14 | 1.0 | MBA | 5 | 1 | 1 |
| 24 | 0.0 | MBA | 2 | 1 | 1 |
| 22 | 0.0 | MBA | 1 | 0 | 0 |
| 8 | 1.0 | MBA | 1 | 0 | 0 |

Group by in slow motion – step 2: APPLY

```
gb = df.groupby(by='Program')  
gb.mean()
```



STEP 2: Apply the aggregating function to each group (slow)

| | Job | Program | ProgSkills | C | CPP |
|----|-----|----------|------------|---|-----|
| 16 | 1.0 | Faculty! | 3 | 1 | 0 |



| | | | | |
|-----------------|------|------|------|------|
| Faculty! | 1.00 | 3.00 | 1.00 | 0.00 |
|-----------------|------|------|------|------|

| | | | | | |
|----|-----|-------------------------------|---|---|---|
| 13 | 0.0 | Supply Chain Mgmt & Analytics | 2 | 1 | 1 |
| 5 | 1.0 | Supply Chain Mgmt & Analytics | 1 | 0 | 0 |



| | | | | |
|--|------|------|------|------|
| Supply Chain Mgmt & Analytics | 0.50 | 1.50 | 0.50 | 0.50 |
|--|------|------|------|------|

| | | | | |
|-------------|------|------|------|------|
| MSIS | 0.19 | 3.00 | 0.78 | 0.50 |
|-------------|------|------|------|------|



| | | | | | |
|----|-----|-----|---|---|---|
| 17 | 1.0 | MBA | 4 | 0 | 0 |
| 18 | 0.5 | MBA | 2 | 0 | 0 |
| 11 | 1.0 | MBA | 2 | 0 | 0 |
| 10 | 0.5 | MBA | 4 | 1 | 1 |
| 15 | 1.0 | MBA | 1 | 0 | 0 |
| 14 | 1.0 | MBA | 5 | 1 | 1 |
| 24 | 0.0 | MBA | 2 | 1 | 1 |
| 22 | 0.0 | MBA | 1 | 0 | 0 |
| 8 | 1.0 | MBA | 1 | 0 | 0 |



| | | | | |
|------------|------|------|------|------|
| MBA | 0.67 | 2.44 | 0.33 | 0.33 |
|------------|------|------|------|------|

| | | | | | |
|----|-----|------|---|---|---|
| 21 | 0.0 | MSIS | 2 | 0 | 0 |
| 23 | 0.5 | MSIS | 3 | 1 | 1 |
| 20 | 0.5 | MSIS | 2 | 0 | 0 |
| 25 | 0.0 | MSIS | 3 | 1 | 1 |
| 19 | 0.0 | MSIS | 5 | 1 | 0 |
| 26 | 0.0 | MSIS | 2 | 0 | 0 |
| 27 | 0.5 | MSIS | 3 | 1 | 1 |
| 0 | 0.0 | MSIS | 4 | 1 | 1 |
| 12 | 0.5 | MSIS | 4 | 1 | 1 |
| 9 | 0.5 | MSIS | 3 | 1 | 0 |
| 7 | 0.0 | MSIS | 2 | 1 | 0 |
| 6 | 0.0 | MSIS | 3 | 1 | 1 |
| 4 | 0.0 | MSIS | 3 | 1 | 0 |
| 3 | 0.0 | MSIS | 3 | 1 | 0 |
| 2 | 0.0 | MSIS | 3 | 0 | 0 |
| 1 | 0.5 | MSIS | 3 | 1 | 1 |
| 28 | 0.5 | MSIS | 3 | 1 | 1 |
| 29 | 0.0 | MSIS | 3 | 1 | 1 |

Group by in slow motion – step 3: COMBINE

```
gb = df.groupby(by='Program')
gb.mean()
```

← STEP 3: Combine into a new DataFrame

| | Job | Program | ProgSkills | C | CPP |
|----|-----|----------|------------|---|-----|
| 16 | 1.0 | Faculty! | 3 | 1 | 0 |



| | | | | |
|-----------------|------|------|------|------|
| Faculty! | 1.00 | 3.00 | 1.00 | 0.00 |
|-----------------|------|------|------|------|

| | | | | | |
|----|-----|-------------------------------|---|---|---|
| 13 | 0.0 | Supply Chain Mgmt & Analytics | 2 | 1 | 1 |
| 5 | 1.0 | Supply Chain Mgmt & Analytics | 1 | 0 | 0 |



| | | | | |
|--|------|------|------|------|
| Supply Chain Mgmt & Analytics | 0.50 | 1.50 | 0.50 | 0.50 |
|--|------|------|------|------|

| | | | | |
|-------------|------|------|------|------|
| MSIS | 0.19 | 3.00 | 0.78 | 0.50 |
|-------------|------|------|------|------|



| | | | | | |
|----|-----|-----|---|---|---|
| 17 | 1.0 | MBA | 4 | 0 | 0 |
| 18 | 0.5 | MBA | 2 | 0 | 0 |
| 11 | 1.0 | MBA | 2 | 0 | 0 |
| 10 | 0.5 | MBA | 4 | 1 | 1 |
| 15 | 1.0 | MBA | 1 | 0 | 0 |
| 14 | 1.0 | MBA | 5 | 1 | 1 |
| 24 | 0.0 | MBA | 2 | 1 | 1 |
| 22 | 0.0 | MBA | 1 | 0 | 0 |
| 8 | 1.0 | MBA | 1 | 0 | 0 |




| | | | | |
|------------|------|------|------|------|
| MBA | 0.67 | 2.44 | 0.33 | 0.33 |
|------------|------|------|------|------|

| | | | | | |
|----|-----|------|---|---|---|
| 21 | 0.0 | MSIS | 2 | 0 | 0 |
| 23 | 0.5 | MSIS | 3 | 1 | 1 |
| 20 | 0.5 | MSIS | 2 | 0 | 0 |
| 25 | 0.0 | MSIS | 3 | 1 | 1 |
| 19 | 0.0 | MSIS | 5 | 1 | 0 |
| 26 | 0.0 | MSIS | 2 | 0 | 0 |
| 27 | 0.5 | MSIS | 3 | 1 | 1 |
| 0 | 0.0 | MSIS | 4 | 1 | 1 |
| 12 | 0.5 | MSIS | 4 | 1 | 1 |
| 9 | 0.5 | MSIS | 3 | 1 | 0 |
| 7 | 0.0 | MSIS | 2 | 1 | 0 |
| 6 | 0.0 | MSIS | 3 | 1 | 1 |
| 4 | 0.0 | MSIS | 3 | 1 | 0 |
| 3 | 0.0 | MSIS | 3 | 1 | 0 |
| 2 | 0.0 | MSIS | 3 | 0 | 0 |
| 1 | 0.5 | MSIS | 3 | 1 | 1 |
| 28 | 0.5 | MSIS | 3 | 1 | 1 |
| 29 | 0.0 | MSIS | 3 | 1 | 1 |

Group by in slow motion – step 3: COMBINE

```
gb = df.groupby(by='Program')  
gb.mean()
```



The column(s) indicated in the by argument will become the index (unless you set `as_index=False`).

In this example, the resulting DataFrame will have one row per Program

Aggregate only few columns

- We want to compute the aggregations among few columns only

On one column:

```
df.groupby('Program')['Job'].mean()
```

```
Program
Business Man    1.00
Faculty!        1.00
MBA             0.66
MSIS            0.20
Master of Finance 0.00
Supply Chain Mgmt & Analytics 0.50
Name: Job, dtype: float64
```

On three columns:

```
df.groupby('Program')[['Job', 'C', 'R']].mean()
```

| | Job | C | R |
|-------------------------------|------|------|------|
| Program | | | |
| Business Man | 1.00 | 0.00 | 1.00 |
| Faculty! | 1.00 | 1.00 | 0.00 |
| MBA | 0.66 | 0.31 | 0.44 |
| MSIS | 0.20 | 0.70 | 0.10 |
| Master of Finance | 0.00 | 1.00 | 0.00 |
| Supply Chain Mgmt & Analytics | 0.50 | 0.50 | 0.00 |

Problems

1. For each Job situation (0=no job, 0.5=part time, 1=full time), find the proportion of students that know SQL
2. For each program, count how many student know SQL.
3. Considering only the students who know SQL, find for each Program the proportion of students who know Java
4. Which one is faster? Why?
 - a) `df.groupby(by='Program')['SQL'].mean()`
 - b) `df.groupby(by='Program').mean()['SQL']`
5. (HARD) For each Classification skill level, how many MBA students are there? Your result should have 5 rows (one for each classification skill level: 1, 2, 3, 4, and 5)

Apply multiple functions to one column (*agg*)

- Oftentimes, we want to apply more than one aggregating function during the same group by operation
- For example:
 - For each Job situation (0=no job, 0.5=part time, 1=full time), find (1) their number and (2) the proportion of students that know SQL.
- We can use agg and pass the list of functions

```
gb = df.groupby('Job')['SQL']
```

```
gb.agg(['mean', 'size'])
```

| | mean | size |
|-----|------|------|
| Job | | |
| 0.0 | 0.87 | 32 |
| 0.5 | 0.93 | 15 |
| 1.0 | 0.71 | 14 |

Apply multiple functions to one column (*agg*) – rename columns

- When using ***agg***, you can rename the column names in the resulting data frame
- For example:
 - For each Job situation (0=no job, 0.5=part time, 1=full time), find (1) their number (call it **n_students**) and (2) the proportion of students that know SQL (call it **SQL_prop**).
- You need to rename the columns “manually”

```
gb.agg(['mean', 'size']).rename(columns={'mean': 'SQL_prop', 'size': 'n_students'})
```

| | SQL_prop | n_students |
|-----|----------|------------|
| Job | | |
| 0.0 | 0.87 | 32 |
| 0.5 | 0.93 | 15 |
| 1.0 | 0.71 | 14 |

Create:

1. a column SQL_prop computed by applying the function “mean”
2. a column n_student computed by applying the function “size”

Apply multiple arbitrary functions to multiple columns (*agg*)

- Oftentimes, we want to apply different aggregating functions to different columns
- For example:
 - For each Job situation (0=no job, 0.5=part time, 1=full time), compute the average knowledge of SQL, the maximum knowledge of Classification, and the gap between the min and the max Classification

Apply multiple arbitrary functions to multiple columns (*agg*)

- We can use agg and pass a nested dictionary

```
gb = df.groupby('Job')
```

```
gb.agg({'SQL' : 'mean',  
       'Classification' : ['max', lambda x: x.max() - x.min()]})
```

This function will be called once for each value of Job. Here, x is the array of values of Classification for the current Job value

For each Job: Compute the mean SQL

For each job, summarize the column Classification by:

- 1) Using the function max, and
- 2) Using the anonymous function that we passed (compute the difference between max and min Classification for the current Job)



The classification spread among those with Job=0.5 is equal to 2

| | Classification | | SQL |
|-----|----------------|----------|------|
| | max | <lambda> | mean |
| Job | | | |
| 0.0 | 4 | 3 | 0.87 |
| 0.5 | 3 | 2 | 0.93 |
| 1.0 | 5 | 4 | 0.71 |

Apply multiple arbitrary functions to multiple columns and give them names (*agg*)

```
gb.agg({'SQL' : 'mean',  
       'Classification' : ['max', lambda x: x.max() - x.min()]})
```



```
gb.agg({'SQL' : 'mean',  
       'Classification' : ['max', lambda x: x.max() - x.min()]})  
.rename(columns={'max' : 'maxClassif',  
                  '<lambda>' : 'spreadClassif',  
                  'mean' : 'SQLmean'})
```



| | Classification | | SQL |
|-----|----------------|---------------|---------|
| | maxClassif | spreadClassif | SQLmean |
| Job | | | |
| 0.0 | 4 | 3 | 0.87 |
| 0.5 | 3 | 2 | 0.93 |
| 1.0 | 5 | 4 | 0.71 |

Group by multiple fields

- Sometimes, we want to group by unique combinations of values in multiple fields
- For example: find the mean of all columns grouped by Program and Job situation. That is, we want one row for each combination of (Program, Job)

```
gb = df.groupby(by=['Program', 'Job'])
```

```
gdf = gb.mean()  
gdf
```

```
df.groupby(['Program', 'Job']).mean()
```

| | | ProgSkills | C | CPP | CS |
|--------------|-----|------------|------|------|------|
| Program | Job | | | | |
| Business Man | 1.0 | 1.00 | 0.00 | 0.00 | NaN |
| Faculty! | 1.0 | 3.00 | 1.00 | 0.00 | 0.00 |
| MBA | 0.0 | 1.75 | 0.25 | 0.25 | 0.00 |
| | 0.5 | 3.00 | 0.33 | 0.33 | 0.33 |
| | 1.0 | 2.67 | 0.33 | 0.44 | 0.11 |



Hierarchical index! (Or MultiIndex)

Hierarchical Indices (and how to avoid them)

- A row identifier that is composed of two or more fields
- The same as “composite key” in relational databases
- It is quite complex to deal with them in pandas
- Avoid them using ***as_index=False***

```
df.groupby(['Program', 'Job'], as_index=False).mean()
```



| | Program | Job | ProgSkills | C | C |
|---|--------------|-----|------------|------|----|
| 0 | Business Man | 1.0 | 1.00 | 0.00 | 0. |
| 1 | Faculty! | 1.0 | 3.00 | 1.00 | 0. |
| 2 | MBA | 0.0 | 1.75 | 0.25 | 0. |
| 3 | MBA | 0.5 | 3.00 | 0.33 | 0. |
| 4 | MBA | 1.0 | 2.67 | 0.33 | 0. |
| 5 | MSIS | 0.0 | 3.08 | 0.65 | 0. |
| 6 | MSIS | 0.5 | 3.00 | 0.75 | 0. |

Problems

1. Find the maximum, minimum, and average number of Languages known by students in each Program
2. For each existing combination of programming skills level and Program, report the number of students (call it *nStudents*) and the proportion that know Python (call it *PythonProportion*)
3. HARD. For each Program, report:
 1. the number of students who know both Python and C (call it *C_Python_Students*, and note that it can be equal to 0)
 2. the gap between max and mean Clustering knowledge (call it *CluGap*)

Retrieve unaggregated rows (*apply*)

- Sometimes, for each group-by value we want to retrieve one or more rows.
- For example, for each program report **all of the students** who know most languages (i.e., report more than one student in case of ties)
- To do that, we need to define a new logic for the apply phase.
 - Input: A DataFrame relative to one partition
 - Output: A DataFrame with the rows that we want to output
- In the example above:
 - Input: A data frame of the students belonging to one program (because the group by is done by Program)
 - Output: The subset of students with the largest value in Languages

GOAL: For each program report the student who knows most languages (report more than one students in case of ties)

DF of Faculty

| | Job | Program | Languages | C |
|----|-----|----------|-----------|---|
| 16 | 1.0 | Faculty! | 3.0 | 1 |

DF of MBA

| | | | | |
|----|-----|-----|-----|---|
| 17 | 1.0 | MBA | 4.0 | 0 |
| 18 | 0.5 | MBA | 2.0 | 0 |
| 11 | 1.0 | MBA | 2.0 | 0 |
| 10 | 0.5 | MBA | 5.0 | 1 |
| 15 | 1.0 | MBA | 1.0 | 0 |
| 14 | 1.0 | MBA | 5.0 | 1 |
| 24 | 0.0 | MBA | 2.0 | 1 |
| 22 | 0.0 | MBA | 2.0 | 0 |
| 8 | 1.0 | MBA | 1.0 | 0 |

DF of MSIS

| | | | | |
|----|-----|------|-----|---|
| 21 | 0.0 | MSIS | 6.0 | 0 |
| 23 | 0.5 | MSIS | 6.0 | 1 |
| 20 | 0.5 | MSIS | 3.0 | 0 |
| 25 | 0.0 | MSIS | 6.0 | 1 |
| 19 | 0.0 | MSIS | 4.0 | 1 |
| 26 | 0.0 | MSIS | 2.0 | 0 |
| 27 | 0.5 | MSIS | 4.0 | 1 |
| 0 | 0.0 | MSIS | 6.0 | 1 |
| 12 | 0.5 | MSIS | 5.0 | 1 |
| 9 | 0.5 | MSIS | 4.0 | 1 |
| 7 | 0.0 | MSIS | 3.0 | 1 |
| 6 | 0.0 | MSIS | 4.0 | 1 |
| 4 | 0.0 | MSIS | 4.0 | 1 |
| 3 | 0.0 | MSIS | 5.0 | 1 |
| 2 | 0.0 | MSIS | 4.0 | 0 |
| 1 | 0.5 | MSIS | 4.0 | 1 |
| 28 | 0.5 | MSIS | 5.0 | 1 |
| 29 | 0.0 | MSIS | 3.0 | 1 |

DF of Supply Chain

| | | | | |
|----|-----|-------------------------------|-----|---|
| 13 | 0.0 | Supply Chain Mgmt & Analytics | 3.0 | 1 |
| 5 | 1.0 | Supply Chain Mgmt & Analytics | 2.0 | 0 |

These 4 dataframes will be passed as input to this function

```
df.groupby('Program').apply(lambda d : d.loc[d.Languages == d.Languages.max(),:])
```

Which selects the students who know most languages within their group

| | | Job | Program | Languages | C |
|-------------------------------|----|-----|-------------------------------|-----------|---|
| Program | | | | | |
| Faculty! | 16 | 1.0 | Faculty! | 3.0 | 1 |
| MBA | 10 | 0.5 | MBA | 5.0 | 1 |
| | 14 | 1.0 | MBA | 5.0 | 1 |
| MSIS | 0 | 0.0 | MSIS | 6.0 | 1 |
| | 21 | 0.0 | MSIS | 6.0 | 0 |
| | 23 | 0.5 | MSIS | 6.0 | 1 |
| | 25 | 0.0 | MSIS | 6.0 | 1 |
| Supply Chain Mgmt & Analytics | 13 | 0.0 | Supply Chain Mgmt & Analytics | 3.0 | 1 |

DESIRED RESULT

In YELLOW: Students with the largest number of languages within their group

first, group by Program

```
gb = df[['Job', 'Program', 'Languages', 'C']].groupby('Program')
```

then, apply a function that returns the students who know most Languages

the function takes a dataframe of students belonging to the same Program,

```
a = gb.apply(lambda d: d.ix[d.Languages == d.Languages.max(),:])
```

Problems

1. For each ProgSkills level, find whether the student (or students in case of ties) with the highest Classification skills know C and Java
2. For each ProgSkills level, find the Program with most students that have that ProgSkill level