EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Philosophische Fakultät
Seminar für Sprachwissenschaft

Data Science for Linguists

Session 7: Data Aggregation and Grouping

Johannes Dellert

8 December, 2023

Table of Contents

Data Aggregation and Grouping: Overview

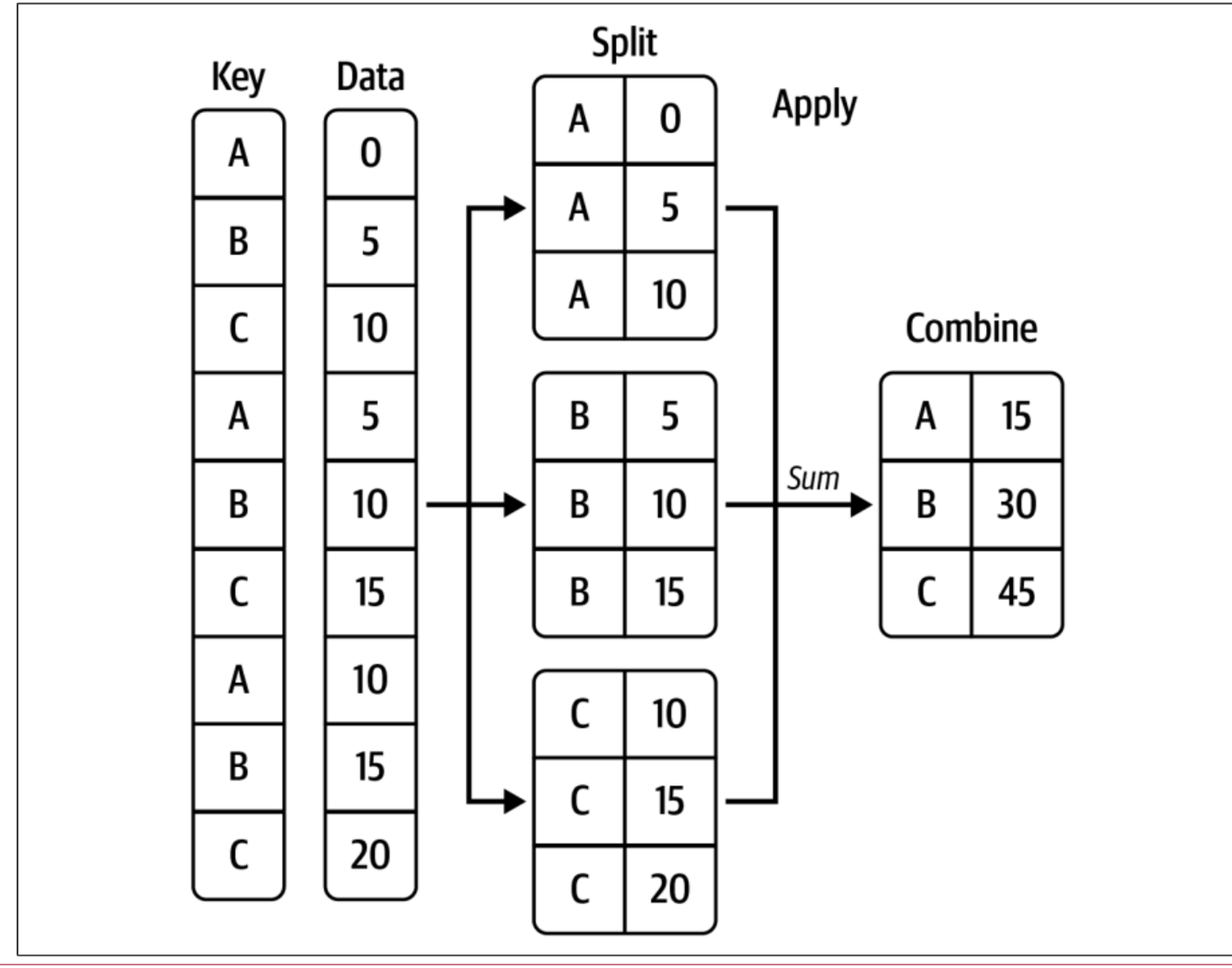
- data analysis workflows often involve spliting a dataset into categories and applying a function to each group (e.g. computation of group statistics, comparison of groups, visualisation)
- Python and Pandas provide an interface for quite complex group operations involving arbitrary manipulations through custom functions
- this session introduces some central capabilities in this area:
  - splitting Pandas objects into pieces using one or more keys of various types
  - calculating group summary statistics like count, mean, or standard deviation
  - applying within-group transformations (normalisation, ranking, subset selection)
  - computing pivot tables and cross-tabulations
  - quantile analysis and other statistical group analyses

Table of Contents

Group Operations: Motivation

- group operations can be described in terms of a **split-apply-combine sequence**
  - data contained in a pandas object is split into groups along one axis (rows or columns) based on one or more keys we provide
  - some function is applied to each group, producing a new value
  - the results of those function applications are combined into a result object, the form of which will depend on what we have been doing to the dta
- grouping keys can take many forms, and do not have to be all of the same type:
  - list or array of values of the same length as the axis being grouped
  - value indicating a column name in a DataFrame
  - dictionary or Series defining a correspondence between the values on the axis being grouped, and the names of the resulting groups
  - function to be invoked on the axis index or individual labels in it

Group Operations: Illustration



Group Operations: Splitting into Groups

- `data.groupby(keys)` expects an array providing the group indices of each row, and returns a `GroupBy` object which summarises all the information needed to apply some operation (e.g. `mean`, `sum`, `size`, `count`) to each of the groups it represents
- example on a database of languages (similar to the Glottolog data you have seen):
  ```
  grouped = df["num_speakers"].groupby(df["family"])
  grouped.mean()
  ```
  (computes and prints the average number of speakers for each family)
- passing a list of arrays leaves us with a hierarchical index over the result Series:
  `df["num_speakers"].groupby([df["family"],df["subfamily"]]).mean()`
- if the grouping information is found in the same dataframe, this can be shortened by only passing the column names:
  `df["num_speakers"].groupby(["family", "subfamily"]]).mean()`
- if we call `groupy` on a dataframe, columns to which the function cannot be applied will be treated as **nuisance columns**, i.e. excluded from the result
- the same is true for missing values in a group key (such rows will be discarded)

Group Operations: Iterating over Groups

- `GroupBy` objects support iteration, generating a sequence of 2-tuples consisting of the group name and the chunk of data
- in case of multiple keys, the first element will be a tuple of key values
  (in our example, pairs of shape `(family, subfamily)`)

Group Operations: Selection by Columns

- using the `axis` parameter, we can group columns instead of rows: `df.groupby({"col1": "a", "col2": "a", "col2": "b"}, axis="columns")`
- GroupBy objects can be indexed by a column name or array of column names: `df.groupby("key1")["data1"]` is the same as `df["data"].groupby(df["key1"])`

Grouping with Dictionaries and Series

-

Grouping with Functions

- function passed as group key will be called once per index value, with return values being used as the group names (example: `len`)

Grouping by Index Levels

-

Table of Contents

Data Aggregation: Optimised GroupBy Methods

`all`	`True` if all non-NA values are “truthy”
`any`	`True` if one or more values are “truthy”
`count`	number of non-NA values
`mean`	
`median`	
`min`	
`max`	
`nth`	value that would appear at position n with data in sorted order
`quantile`	computes sample quantile
`rank`	
`size`	
`sum`	
`std`	
`var`	

Data Aggregation: Custom Functions

- Example: `grouped.agg(peak_to_peak)` for `arr.max() - arr.min()`
- these are generally much slower than the optimised methods because of the extra overhead involved in constructing intermediate group data chunks

Data Aggregation: Column-Wise and Multiple Function Application

- aggregation by functions which differ per column
  - Example: `grouped.agg(["similarity": np.max, "size": "sum"])`
- aggregation by multiple functions at once:
  - Example: `grouped.agg(["mean", "std"])`
  - result will be a DataFrame with function names as column names

Table of Contents

Apply: Basic Split-Apply-Combine Pattern

- `apply` is the most general-purpose GroupBy method
  - spits the object being manipulated into pieces
  - invokes the passed function on each piece
  - then attempts to concatenate the pieces
- function must either return a pandas object or a scalar value
- if the function passed takes other arguments or keywords, they can be passed to `apply` after the function

Example: Quantile and Bucket Analysis

- McKinney (2022), p. 338f

Example: Group-Specific Default Values

- McKinney (2022), p. 340ff

Example: Random Sampling and Permutation

- McKinney (2022), p. 343f

Example: Group Weighted Average and Correlation

- McKinney (2022), p. 344ff

Table of Contents

Pivot Tables

- a **pivot table**
  - aggregates a table of data by one more keys
  - arranges the data in a rectangle with some keys along the rows and some along the columns
- they are possible in Pandas through `groupby` in combination with reshape operations which utilize hierarchical indexing
- convenience method `df.pivot_table` can add partial totals (**margins**)
- basic usage: `df.pivot_table(index=["col1", "col3"])` creates hierarchical index, and by default provides the group means for the remaining columns

Pivot Tables: Example

- McKinney (2022), p. 352ff

Cross-Tabulation

- a **cross-tabulation** is a special case of a pivot tale that computes group frequencies
- takes two (lists of) columns as arguments, the first will form the (hierarchical) row index, the second the (hierarchical) column index

Cross-Tabulation: Example

- `pd.crosstab(data["Nationality"], data["Handedness"], margins=True)`

Preliminary Course Plan

1	**27/10**	**IPython and Jupyter**
2	**03/11**	**Introduction to NumPy**
3	**10/11**	**Pandas and Data Frames**
4	**17/11**	**Data Cleaning and Preparation**
5	**24/11**	**Linguistic Preprocessing**
6	**01/12**	**Data Wrangling**
7	**08/12**	**Data Aggregation and Grouping**
8	15/12	Visualisation with Seaborn
9	22/12	Modeling and Prediction
10	12/01	Classification
11	19/01	Clustering
12	26/01	Pattern Extraction and Density Estimation
13	02/02	Statistical Inference
14	09/02	Data Science Projects

Questions

Questions?

Comments?

Suggestions?