Cheatsheets / **Python for Data Science: Working with Data** codecademy

Merging and Aggregating Data

Aggregations

Aggregation refers to using one value to describe multiple datapoints. Calculating an average is the classic example of aggregation, because we use one value (the average) to describe the "center" of multiple datapoints.
Aggregations like the average are also called summary statistics because they summarize an entire group of data using a statistic.

NumPy Library

NumPy is a Python library for mathematical computations, including summary statistics.
The standard convention to import NumPy is to use the alias `np`:

```
import numpy as np
```

The pandas library is built on top of NumPy, which means we can apply NumPy functions to pandas objects like Series and DataFrames.

codecademy

NumPy Functions

Some common NumPy aggregation functions include

- `np.mean()`, `np.median()` for the mean and median
- `np.max()`, `np.min()` for maximum and minimum values
- `np.sum()` to sum all the values in an array

For example, let's use NumPy to calculate the mean test score in `scores`:

student_ID	test_score
001	87
002	93
003	90
004	96
005	85

```
np.mean(scores['test_score'])
# Output: 90.2
```

Pandas Aggregations

Pandas provides built-in methods to aggregate DataFrame columns.
Common built-in summary methods include:

- `.mean()` returns the mean
- `.median()` returns the median
- `.std()` returns the standard deviation
- `.max()` and `.min()` return the maximum and minimum values respectively
- `.nunique()` returns the count of unique values
- `.count()` returns the count of non-null values
- `.sum()` returns the sum

```
# Summarize a single column
df['col'].summary_method()

# Summarize multiple columns
df[['col1', 'col2']].summary_method()
```

codecademy

Pandas `.groupby()` Method

The pandas `.groupby()` method splits a DataFrame into groups corresponding to the unique values in a column.

Here is the `df` referred to in the code snippet:

date	home_team	away_team
2021-09-24	El Salvador	Guatemala
2019-07-07	United States	Mexico
2021-06-27	El Salvador	Guatemala
2016-03-25	El Salvador	Honduras

```
# Split df into groups based on each
home_team
df.groupby('home_team')

# Split df into groups based on
home_team vs away_team matchups
df.groupby(['home_team', 'away_team'])
```

Applying Pandas Aggregations

Calling an aggregation method in Pandas after performing a groupby will apply that aggregation to the individual groups in the groupby.

For example, the code snippet uses `.sum()` to sum the total number of `home_score` goals by each `home_team` in `df`:

home_team	home_score
England	3
Brazil	3
South Korea	2
England	2

```
# Applying the aggregation function
.sum()
df.groupby('home_team')
['home_score'].sum()

# Output:
# home_team
# Brazil         3
# England        5
# South Korea    2
```

codecademy

Pandas `.agg()` Method

Pandas' `.agg()` method is a flexible way to apply aggregation functions to groups of a `groupby`. The input to `.agg()` is a dictionary where

- the keys specify which columns to aggregate
- the values are the aggregation functions to apply

For example, the code snippet applies `sum` to compute the total number of `home_score` goals by each `home_team` in `df`:

home_team	home_score
England	3
Brazil	3
South Korea	2
England	2

```
# Using .agg() to apply aggregation functions
df.groupby('home_team').agg({'home_score':'sum'})

# Output:
# home_team
# Brazil         3
# England        5
# South Korea    2
```

codecademy

Pandas `.pivot_table()` Method

The pandas method `pd.pivot_table()` transforms a DataFrame into **wide** format that is more human-readable using the input parameters:

- `index` is used to label the rows of the table
- `columns` is used to label the columns of the table
- `values` is used to fill the table
- `aggfunc` is the aggregation function applied to `values`

For example, say `df` contains sales data for two products in two regions:

Product	Region	Sales
A	Asia	1000
B	Asia	1500
A	Asia	800
B	North America	1200
A	North America	1100
B	North America	1700

The code snippet pivots `df` to calculate the mean sales of each product in each region:

Product	A	B
Region		
Asia	900	1500
North America	1100	1450

```
# Pivot df to calculate the mean sales
of each product in each region
pivot_table = pd.pivot_table(df,

values='Sales',

index='Region',

columns='Product',

aggfunc='mean')
```

codecademy

Split-Apply-Combine

Split-apply-combine or **SAC** is a common workflow in data science to answer data questions. SAC involves a three-step process where we

1. **Split** the dataset into one or more pieces
2. **Apply** a function or transformation to each piece
3. **Combine** the results from each piece

Pandas Inner Merge

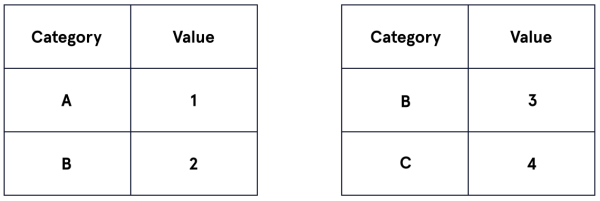
An **inner merge** combines two DataFrames by identifying matching values. For example, here are two DataFrames

X

Category	Value
A	1
B	2

Y

Category	Valu
B	3
C	4



Category	Value
A	1
B	2

Category	Value
B	3
C	4

In an inner merge on the `Category` columns, only the matching category `B` would be retained, together with the corresponding data from both DataFrames:

Category	Value_x	Value_y
B	2	3

The pandas code is

```
pd.merge(left = X,
         right = Y,
         left_on = 'Category',
         right_on = 'Category'
         how = 'inner')
```

codecademy

Pandas Left/Right Merge

In Pandas, a **left** or **right merge** combines two DataFrames, preserving all rows on either the left or the right DataFrame respectively.
The type is specified using the `how` keyword:

```
pd.merge(left = X,
         right = Y,
         left_on = 'Category',
         right_on = 'Category'
         how = 'right')
```

Category	Value
A	1
B	2

Category	Value
B	3
C	4

For example, here are two DataFrames:

X

Category	Value
A	1
B	2

Y

Category	Valu
B	3
C	4

Performing a right merge, we maintain all rows of the right DataFrame:

Category	Value_x	Value_y
B		3
C		4

and then add on any existing data from the left DataFrame:

Category	Value_x	Value_y
B	2.0	3
C	NaN	4

codecademy

Pandas Outer Merge

In Pandas, an **outer merge** combines two (or more) DataFrames, maintaining all rows from both DataFrames. To perform an outer merge, specify `outer` as the `how` argument:

```
pd.merge(left = X,
         right = Y,
         left_on = 'Category',
         right_on = 'Category'
         how = 'outer')
```

Category	Value
A	1
B	2

Category	Value
B	3
C	4

For example, here are two DataFrames

X

Category	Value
A	1
B	2

Y

Category	Valu
B	3
C	4

To outer merge them, we match values to the different categories, filling in `NaN` s where no value exists:

Category	Value_x	Value_y
A	1	NaN
B	2	3
C	NaN	4

Print Share