

ISA 419: Data-Driven Security

07: Aggregating Data with Pandas

Fadel M. Megahed, PhD

Endres Associate Professor
Farmer School of Business
Miami University

 @FadelMegahed

 fmegahed

 fmegahed@miamioh.edu

 Automated Scheduler for Office Hours

Spring 2024

Quick Refresher of Last Class

- ✓ Ensure that your imported data is **technically correct** (rename columns and fix **dtypes**)
- ✓ Clean data to ensure that your data is consistent
- ✓ Understand the difference between **concatenate**, **merge**, and **join**.

Learning Objectives for Today's Class

- Understand how to change the unit of analysis by grouping and aggregating data.
- Use the `agg()` function to do aggregations on grouped data.

Grouping and Aggregating Data

Our Data

- We will use the `merged_ips` data set from the previous class to demonstrate how to group and aggregate data.

```
import pandas as pd

toxic_ips = pd.read_csv(
    "https://raw.githubusercontent.com/fmegahed/isa419/main/data/listed_ip_90_all.csv",
    header = None, names = ['ip', 'frequency', 'lastseen']
)

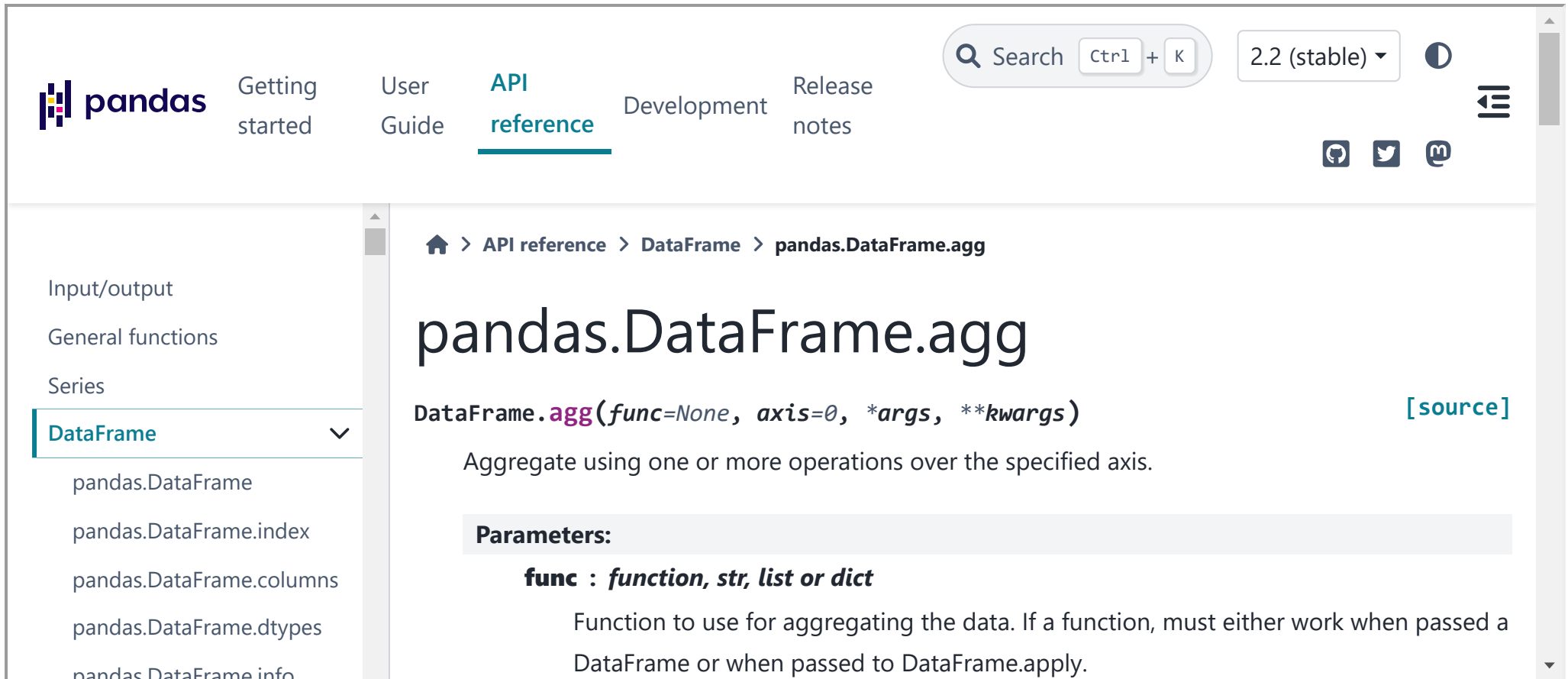
geolocation = pd.read_csv(
    'https://raw.githubusercontent.com/fmegahed/isa419/main/data/ip_geolocation.csv',
    names = ['ip', 'country', 'city', 'latitude', 'longitude']
)

merged_ips = (
    toxic_ips
    .merge(right = geolocation, how = 'left', on = 'ip')
    .dropna()
    .assign( lastseen = lambda df: df['lastseen'].astype('datetime64[ns]') )
)
merged_ips.dtypes[0:3]
```

```
## ip                object
## frequency         int64
## lastseen          datetime64[ns]
## dtype: object
```

Aggregating Data

- The `agg()` function is used to apply one or more functions to a column in a data frame.



The screenshot shows the pandas API reference website. The top navigation bar includes the pandas logo, links for 'Getting started', 'User Guide', 'API reference' (which is highlighted), 'Development', and 'Release notes'. On the right, there is a search bar with 'Search', 'Ctrl + K' shortcuts, a version dropdown set to '2.2 (stable)', and social media icons for GitHub, Twitter, and Medium. A left sidebar contains a tree view with categories: 'Input/output', 'General functions', 'Series', and 'DataFrame' (which is selected and expanded). Under 'DataFrame', sub-items include 'pandas.DataFrame', 'pandas.DataFrame.index', 'pandas.DataFrame.columns', 'pandas.DataFrame.dtypes', and 'pandas.DataFrame.info'. The main content area displays the breadcrumb 'API reference > DataFrame > pandas.DataFrame.agg' followed by the title 'pandas.DataFrame.agg'. Below the title is the function signature 'DataFrame.agg(func=None, axis=0, *args, **kwargs)' with a '[source]' link. A description states: 'Aggregate using one or more operations over the specified axis.' A 'Parameters:' section follows, detailing the 'func' parameter: '**func** : function, str, list or dict' and 'Function to use for aggregating the data. If a function, must either work when passed a DataFrame or when passed to DataFrame.apply.'

pandas

Getting started User Guide **API reference** Development Release notes

Search Ctrl + K 2.2 (stable)

Input/output
General functions
Series
DataFrame
pandas.DataFrame
pandas.DataFrame.index
pandas.DataFrame.columns
pandas.DataFrame.dtypes
pandas.DataFrame.info

API reference > DataFrame > pandas.DataFrame.agg

pandas.DataFrame.agg

`DataFrame.agg(func=None, axis=0, *args, **kwargs)` [\[source\]](#)

Aggregate using one or more operations over the specified axis.

Parameters:

func : function, str, list or dict

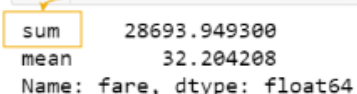
Function to use for aggregating the data. If a function, must either work when passed a DataFrame or when passed to DataFrame.apply.

Aggregating Data

pandas aggregation options

List

```
1 # Use a list
2 df['fare'].agg(['sum', 'mean'])
```



| | |
|------|--------------|
| sum | 28693.949300 |
| mean | 32.204208 |

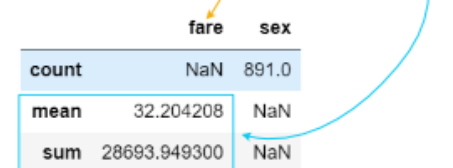
Name: fare, dtype: float64

All aggregations in list will be applied to column

Dictionary

Define columns as dictionary keys
All aggregations in list will be applied

```
1 # Use a dictionary
2 df.agg({'fare': ['sum', 'mean'],
3        'sex': ['count']})
```

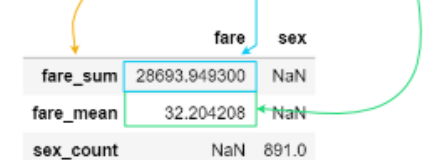


| | fare | sex |
|-------|--------------|-------|
| count | NaN | 891.0 |
| mean | 32.204208 | NaN |
| sum | 28693.949300 | NaN |

Tuple

Pass a tuple of column names and aggregations
Only one aggregation can be passed per tuple
Assign a name for the result

```
1 # Use a named aggregation tuple
2 df.agg(fare_sum=('fare', 'sum'),
3        fare_mean=('fare', 'mean'),
4        sex_count=('sex', 'count'))
```



| | | |
|-----------|--------------|-------|
| fare_sum | 28693.949300 | NaN |
| fare_mean | 32.204208 | NaN |
| sex_count | NaN | 891.0 |

Grouping and Aggregating Data

- Grouping and aggregating data are common operations in data analysis.
- Grouping data is the process of splitting data into groups based on some criteria.
- Aggregating data is the process of applying a function to each group, producing a single value for each group, i.e.:
 - number of rows will equal to the number of groups.
 - number of columns will equal to the number of functions applied.
- The `agg()` function's input can be a dictionary, list, or a tuple (as shown in the image in the previous slide).

Grouping & Aggregating Data with a Dictionary

```
grouped_merged_ips1 =(
    merged_ips.groupby('country').agg(
        {
            'frequency': ['count', 'sum', 'mean', 'median', 'max'],
        }
    )
)

# printing the top three rows and the column names
grouped_merged_ips1.head(n=3)
```

```
##           frequency
##           count    sum      mean median   max
## country
## Afghanistan      5     5  1.000000    1.0     1
## Albania         120   576  4.800000    2.0    46
## Algeria         229  2081  9.087336    1.0  1554
```

```
grouped_merged_ips1.columns
```

```
## MultiIndex([('frequency', 'count'),
##            ('frequency', 'sum'),
##            ('frequency', 'mean'),
##            ('frequency', 'median'),
##            ('frequency', 'max')],
##           )
```

Grouping & Aggregating Data with a Dictionary (Cont.)

```
# flattening the multi-index for column names
grouped_merged_ips1.columns = ['_'.join(col).strip() for col in grouped_merged_ips1.columns.values]

# print the new column names
grouped_merged_ips1.columns
```

```
## Index(['frequency_count', 'frequency_sum', 'frequency_mean',
##       'frequency_median', 'frequency_max'],
##       dtype='object')
```

Grouping & Aggregating Data with a Tuple

```
grouped_merged_ips2 =(
    merged_ips.groupby('country').agg(
        freq_count = ('frequency', 'count'),
        freq_sum = ('frequency', 'sum'),
        freq_mean = ('frequency', 'mean'),
        freq_median = ('frequency', 'median'),
        freq_max = ('frequency', 'max')
    )
)

# printing the bottom eight rows
grouped_merged_ips2.tail(n=8)
```

| ## | freq_count | freq_sum | freq_mean | freq_median | freq_max |
|---------------|------------|----------|-----------|-------------|----------|
| ## country | | | | | |
| ## Uruguay | 22 | 42 | 1.909091 | 1.0 | 6 |
| ## Uzbekistan | 69 | 2295 | 33.260870 | 1.0 | 778 |
| ## Vanuatu | 2 | 15 | 7.500000 | 7.5 | 14 |
| ## Venezuela | 129 | 417 | 3.232558 | 1.0 | 114 |
| ## Vietnam | 4552 | 19478 | 4.278998 | 1.0 | 793 |
| ## Yemen | 51 | 67 | 1.313725 | 1.0 | 7 |
| ## Zambia | 41 | 65 | 1.585366 | 1.0 | 14 |
| ## Zimbabwe | 57 | 86 | 1.508772 | 1.0 | 11 |

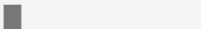

Grouping & Aggregating Data: Jazz It Up

```
import numpy as np
from sparklines import sparklines

def sparkline_str(x):
    bins=np.histogram(x)[0] # from numpy
    sl = ''.join(sparklines(bins))
    return sl

grouped_merged_ips3 =(
    merged_ips
    .groupby('country')
    .agg(
        freq_count = ('frequency', 'count'),
        freq_sum = ('frequency', 'sum'),
        freq_median = ('frequency', 'median'),
        freq_max = ('frequency', 'max'),
        freq_sparkline = ('frequency', sparkline_str)
    )
    .query('country in ["United States", "China"]')
    .reset_index()
)

# printing the two rows
grouped_merged_ips3.tail(n=2)
```

| ## | country | freq_count | freq_sum | freq_median | freq_max | freq_sparkline |
|------|---------------|------------|----------|-------------|----------|---|
| ## 0 | China | 2434 | 44203 | 5.0 | 887 |  |
| ## 1 | United States | 25116 | 230434 | 2.0 | 12943 |  |

Performing your Own Aggregations

| Task | Hints |
|------|-------|
|------|-------|

- In [Google Colab](#), build on our approach in class to compute the following:
 - Compute the median frequency by `country` and `city`.
 - Compute the median latitude by `country`.

Performing your Own Aggregations

| Task | Hints |
|------|-------|
|------|-------|

- You can pass multiple columns in the `groupby` function using a list.
- Check the `dtype` for latitude and ensure that your data is technically correct.

Recap

Summary of Main Points

By now, you should be able to do the following:

- Understand how to change the unit of analysis by grouping and aggregating data.
- Use the `agg()` function to do aggregations on grouped data.



Review and Clarification



- **Class Notes:** Take some time to revisit your class notes for key insights and concepts.
- **Zoom Recording:** The recording of today's class will be made available on Canvas approximately 3-4 hours after the end of class.
- **Questions:** Please don't hesitate to ask for clarification on any topics discussed in class. It's crucial not to let questions accumulate.