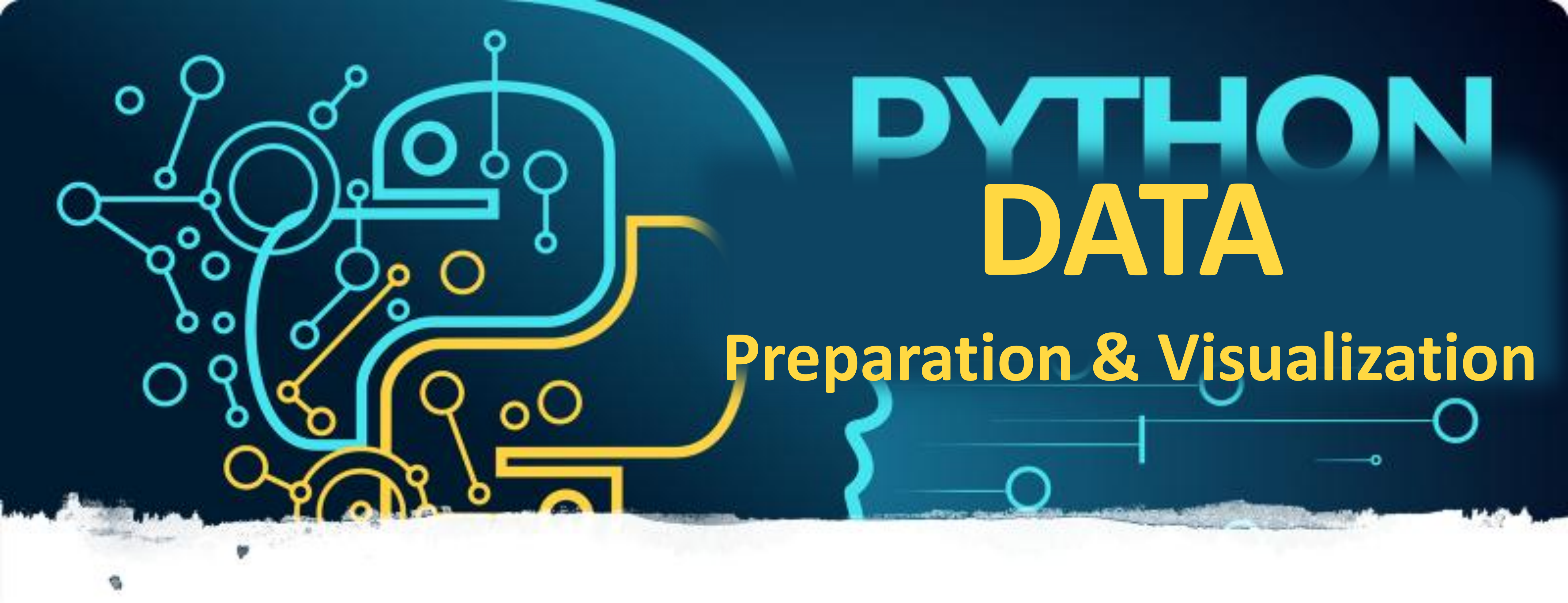


Lesson 3: Data Aggregation and Group Operations

Lecturer: **Nguyen Tuan Long, Phd**
Email: ntlong@neu.edu.vn
Mobile: 0982 746 235

Contents

Reviews: Aggregation

Simple Aggregation in Pandas

- Methods like **sum()**, **mean()**, and **count()** can be directly applied to DataFrames.

Table summarizes some other built-in Pandas aggregations

Aggregation	Description
`count()`	Total number of items
`first()`, `last()`	First and last item
`mean()`, `median()`	Mean and median
`min()`, `max()`	Minimum and maximum
`std()`, `var()`	Standard deviation and variance
`mad()`	Mean absolute deviation
`prod()`	Product of all items
`sum()`	Sum of all items

Practice


```python
import seaborn as sns
planets = sns.load_dataset('planets')
planets.head()
```

Question:

1. What is the total number of planets discovered?
2. What is the average orbital period of the discovered planets?
3. How many unique methods were used to discover the planets?
4. What is the maximum mass of the discovered planets?
5. What is the minimum distance of the discovered planets from Earth?

Các cột chính trong bộ dữ liệu:

- **method**: Phương pháp được sử dụng để phát hiện hành tinh.
- **number**: Số lượng hành tinh được phát hiện trong mỗi hệ thống hành tinh.
- **orbital_period**: Thời gian quỹ đạo của hành tinh (tính bằng ngày).
- **mass**: Khối lượng của hành tinh (tính bằng khối lượng của Sao Mộc).
- **distance**: Khoảng cách từ Trái Đất đến hành tinh (tính bằng parsec).
- **year**: Năm phát hiện hành tinh.

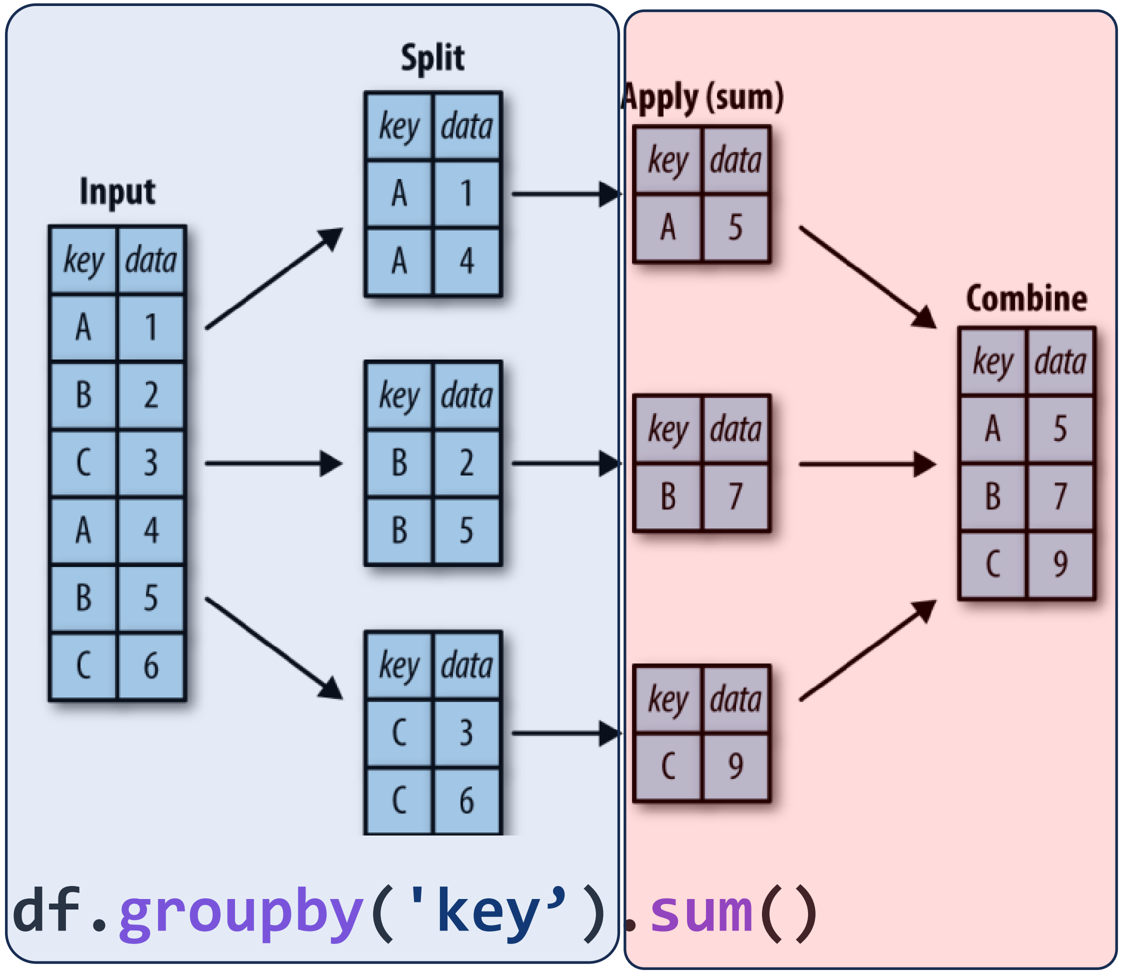
GroupBy: Split, Apply, Combine

The **groupby()** method **splits** the data into groups, **applies** a function to each group, and **combines** the results. This is known as the "split-apply-combine" strategy.

Practice

```
df = pd.DataFrame({'key': list("ABCABC"),
                   "data":range(6)},
                   columns=['key','data'])
```

- The key should be the **category** or **discrete**



Aggregation and Grouping

Aggregate, filter, transform, apply

Aggregation

We're now familiar with GroupBy aggregations with sum(), median(), and the like, but the aggregate() method allows for even more flexibility.

Practcie

```
df = pd.DataFrame({'key': list("ABCABC"),
                   "data1":range(6),
                   'data2':range(6,12)},
                   columns=['key','data1','data2'])
```

```
def myfunc(x):
    return x.sum()//2
```

- `df.groupby('key').aggregate(['sum','min',myfunc])`
- `df.groupby('key').aggregate({'data1':['sum','min'],'data2':myfunc})`

Filtering

A filtering operation allows you to drop data based on the group properties.

```
Let's look at the following codes and explain how filters work
    1. df.groupby('key').sum()
    2. df.groupby('key').filter(lambda x: x.data1.sum()>=5)
```

Transformation

Applies a function to each group and returns an object with the same size as the original group. Often used for normalization or data transformation.

```
1  df.groupby('key').transform(lambda x: x-x.mean())
```

✓ 0.0s

	data1	data2
0	-1.5	-1.5
1	-1.5	-1.5
2	-1.5	-1.5
3	1.5	1.5
4	1.5	1.5
5	1.5	1.5

The apply() method

Applies an arbitrary function to each group and can return an object with a different shape than the original group.

```
def norm_by_data2(x):
    # x is a DataFrame of group values
    x['data1'] /= x['data2'].sum()
    return x
```

```
df
```

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

```
df.groupby('key').apply(norm_by_data2)
```

	key	data1	data2
0	A	0.000000	5
1	B	0.142857	0
2	C	0.166667	3
3	A	0.375000	3
4	B	0.571429	7
5	C	0.416667	9

Summary

- **Aggregate (agg)**: Apply multiple aggregation functions on group columns.
- **Filter**: Filter groups based on a condition.
- **Transform**: Apply a function to each group, returning an object of the same size.
- **Apply**: Apply an arbitrary function to each group, returning an object with a potentially different shape.

Practice


```
import seaborn as sns
planets = sns.load_dataset('planets')
```

Practice Requirements

1. Handling missing data
2. Using describe() method
3. Calculate the total number of planets discovered by each method.
4. Calculate the average orbital period of planets for each discovery method.
5. Determine the maximum mass of planets discovered by each method.
6. Calculate the average distance from Earth to the planets discovered each year.
7. Count the number of planet discoveries by each method and year.
8. Aggregate the total number of planets and average mass by each method.
9. Filter methods of discovery that have discovered more than 20 planets in total.
10. Apply a custom function to calculate the average mass of planets if the number of planets is greater than 1 within the group.

MultiIndex – Hierachical Indexing

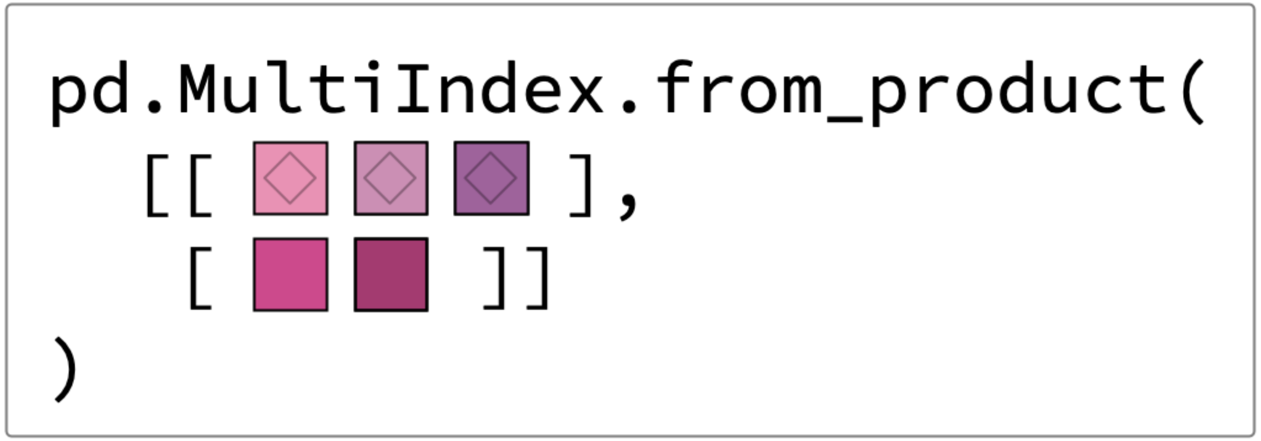
- MultiIndex in pandas allows for the creation of hierarchical indexes, making it easier to manage and analyze complex data.
- It is useful for working with hierarchical data and performing complex group operations.
- MultiIndex can be used for both rows and columns

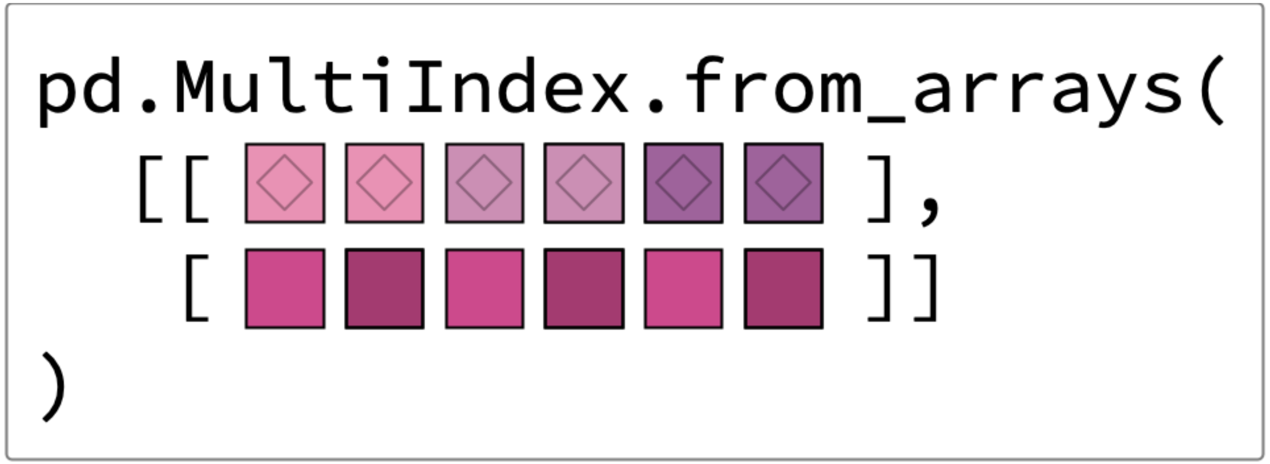
		pop	under_18
states	**years**		
California	2000	33871648	9267089
	2010	37253956	9284094
New York	2000	18976457	4687374
	2010	19378102	4318033
Texas	2000	20851820	5906301
	2010	25145561	6879014

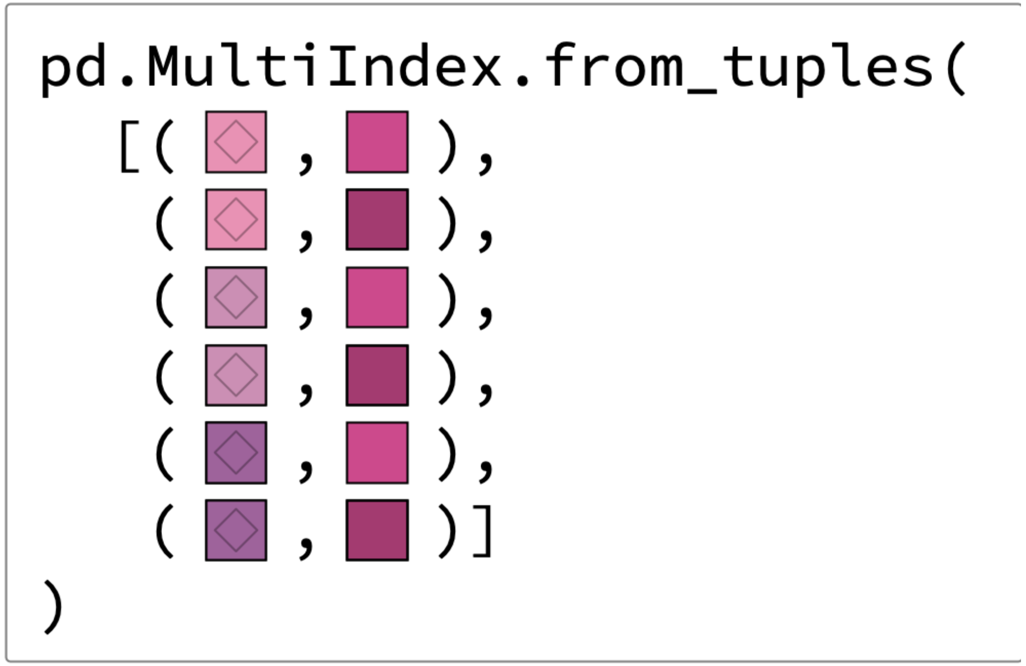
MultiIndex

MultiIndex – Hierachical Indexing

Creating a MultiIndex







MultiIndex - Hierarchical Indexing

Create a DataFrame with MultiIndex

```
index = [('California', 2000), ('California', 2010),
         ('New York', 2000), ('New York', 2010),
         ('Texas', 2000), ('Texas', 2010)]
populations = [33871648, 37253956, 18976457, 19378102,  20851820, 25145561]
under18= [9267089, 9284094, 4687374, 4318033,5906301, 6879014]
ind = pd.MultiIndex.from_tuples(index, names=('states','years'))
pop = pd.DataFrame({'pop':populations, 'under_18':under18}, index=ind)
```

Accessing and Manipulating MultiIndex

- `.loc[]`: `pop.loc['New York’]`
- `.xs()`: `pop.xs(2010,level='years')`

MultiIndex - Hierarchical Indexing

Create a MultiIndex from columns: use 'set_index' method

```
# Step 1: Create a sample DataFrame
data = {
    'Country': ['USA', 'USA', 'Canada', 'Canada'],
    'State': ['California', 'New York', 'Ontario', 'Quebec'],
    'City': ['Los Angeles', 'New York City', 'Toronto', 
'Montreal'],
    'Population': [3970000, 8419000, 2731000, 1705000]
}

df = pd.DataFrame(data)

# Step 2: Use set_index to create a MultiIndex
df_multi_index = df.set_index(['Country', 'State'])
```

Reshaping: stack() and unstack()

stack(): moves columns into row index

unstack(): moves row index into columns

A	B	C	D	E
0	x	1	10	2
	y	2	20	1
1	x	3	30	5
	y	4	20	4

A	B		
0	x	C	1
		D	10
		E	2
	y	C	2
		D	20
		E	1
1	x	C	3
		D	30
		E	5
	y	C	4
		D	20
		E	4

Wide

Long

A	B	C	D	E
0	x	1	10	2
	y	2	20	1
1	x	3	30	5
	z	4	20	4

	C			D			E		
B	x	y	z	x	y	z	x	y	z
A									
0	1	2	NaN	10	20	NaN	2	1	NaN
1	3	NaN	4	30	NaN	20	5	NaN	4

Motivating Pivot Tables

```
import numpy as np
import pandas as pd
import seaborn as sns
titanic = sns.load_dataset('titanic')
titanic.head()
```

sex	female	male
class		
First	0.968085	0.368852
Second	0.921053	0.157407
Third	0.500000	0.135447

- To get a deeper look at survival rates by both gender and ticket class, we can use a long and complex code string:

```
titanic.groupby(['class','sex'])['survived'].mean().unstack()
```

- Pandas provides the pivot_table method to **simplify** ultidimensional aggregation

```
pd.pivot_table(data=titanic,index='class’,
               columns='sex', values='survived')
```

Pivot Tables

- **Additional pivot table options**

```
DataFrame.pivot_table(data,
                      values=None,
                      index=None,
                      columns=None,
                      aggfunc='mean',
                      fill_value=None,
                      margins=False,
                      dropna=True,
                      margins_name='All',
                      observed=False,
                      sort=True)
```

- **data**: The input data for the pivot table.
- **values**: Column(s) to aggregate. If omitted, all remaining numeric columns will be used.
- **index**: Column(s) to group by on the pivot table index. These become the rows.
- **columns**: Column(s) to group by on the pivot table columns.
- **aggfunc**: Function to use for aggregation (e.g., 'mean', 'sum'). Default is 'mean'.
- **fill_value**: Value to replace missing values with in the pivot table.
- **margins**: Add row/column totals (subtotals). Default is False.
- **dropna**: Do not include columns whose entries are all NaN. Default is True.
- **margins_name**: Name of the row/column that will contain the totals when margins=True. Default is 'All'.
- **observed**: If True, only show observed values for categorical groupers. Default is False.
- **sort**: Sort the result. Default is True.

Pivot Tables

- **Task 1: Survival Rate by Class and Sex:** Create a pivot table to find the survival rate by passenger class (pclass) and sex (sex).
- **Task 2: Average Fare by Class, Sex, and Embarkation Port:** Create a pivot table to find the average fare by passenger class (pclass), sex (sex), and embarkation port (embark_town).
- **Task 3: Total Count of Survivors by Class and Deck:** Create a pivot table to find the total number of survivors by passenger class (pclass) and deck (deck):
- **Task 4: Age Distribution by Class and Gender:** Create a pivot table to find the average and median age of passengers by class (pclass) and gender (sex).
- **Task 5: Embarkation Port Distribution by Class and Gender:** Create a pivot table to find the count of passengers by class (pclass), gender (sex), and embarkation port (embark_town).
- **Task 6: Survival Rate by Age Group and Class**: Create a pivot table to find the survival rate by age group and class (pclass). First, create age groups using pd.cut.

```python
# Create age groups
age_groups = pd.cut(titanic['age'],
                    bins=[0, 12, 18, 35, 60, 80],
                    labels=['Child', 'Teen', 'Adult', 'Middle-aged', 'Senior'])
# Add age group column to the DataFrame
titanic['age_group'] = age_groups
```

Practice: Birthrate Data

Let's take a look at the freely available data on births in the United States, provided by the Centers for Disease Control (CDC)

url = ‘https://raw.githubusercontent.com/jakevdp/data-CDCbirths/master/births.csv’
births = pd.read_csv(url)

```
births.head()
```

	year	month	day	gender	births
0	1969	1	1	F	4046
1	1969	1	1	M	4440
2	1969	1	2	F	4454
3	1969	1	2	M	4548
4	1969	1	3	F	4548

- Let's add a decade column, and take a look at male and female births as a function of decade.
- Total birth in the quarters at male and female births
- …..

Dataframes and Data Cleaning Tasks

The specific tasks that you need to perform depend on the structure and contents of a dataset. In general, you will perform a workflow with the following steps (not necessarily always in this order), all of which can be performed with a Pandas DataFrame:

1. read data into a DataFrame
2. display top of DataFrame
3. display column data types
4. display non-missing values
5. replace NA with a value
6. iterate through the columns
7. statistics for each column
8. find missing values
9. total missing values
10. percentage of missing values
11. sort table values
12. print summary information
13. columns with > 50% missing values
14. rename columns

....

Visualize data with pandas

Method: ‘plot()’

```
df = pd.pivot_table(data=birth,
                    index = 'year',
                    columns= 'gender',
                    values='births',
                    aggfunc='sum')
```

- `df.plot(kind = 'line',title='Title-1', xlabel='X-axis', ylabel='Y-axis')`
- `df.plot(kind='bar',title='Title-2', xlabel='X-axis', ylabel='Y-axis')`
- `df.plot(kind='hist',title='Title-3', xlabel='X-axis', ylabel='Y-axis')`
- `df.plot(kind='box',title='Title-4', xlabel='X-axis', ylabel='Y-axis')`
- `df.plot(kind='scatter', x='F', y='M',title='Title-5', xlabel='X-axis', ylabel='Y-axis')`

Kind*:* The kind of plot to produce:

- ‘line’ : line plot (default)
- ‘bar’ : vertical bar plot
- ‘barh’ : horizontal bar plot
- ‘hist’ : histogram
- ‘box’ : boxplot
- ‘kde’ : Kernel Density Estimation plot
- ‘density’ : same as ‘kde’
- ‘area’ : area plot
- ‘pie’ : pie plot
- ‘scatter’ : scatter plot (DataFrame only)
- ‘hexbin’ : hexbin plot (DataFrame only)

Ref: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html#pandas.DataFrame.plot