Data Science

Unit 1-05: Data Wrangling

AKADEMI GA

GENERAL ASSEMBLY

COURSE CONTENT

Week 1 : Data Science Foundations

Congratulations!

Installation and Github, Python fundamentals, Introduction to Pandas

Week 2 : Working with Data

More pandas, basics of probability and statistics, Exploratory Data Analysis (EDA), working with data, use statistical analysis and visualisation

Week 3 : Data Science Modeling

Linear regression Train/Test/Split, Classification, Logistic Regression

Week 4 : Data Science Applications

Using APIs, Natural Language Processing, Time Series Analysis

Week 5: Final Presentation

Present your capstone project

Week 1: Data Science Foundations

So far:

- a review of Python fundamentals

- Introduction to Pandas

	Week 1 Units
	1-01 Installation and Github
	1-02 Python Review and Practice
	1-03 List Comprehension
	1-04 Introduction to Pandas
Today:	1-05 Data Wrangling

Unit 1-05 Data Wrangling

Lesson 1: Data Wrangling

AKADEMI GA

GENERAL ASSEMBLY

Data Preparation

- Data preparation is an important step before creating a data science model.

Data Cleaning

- Remove inconsistencies and errors
- Put into correct format for modelling
- Renaming data items

Data Wrangling

- Extracting parts of information from data
- Combining data and performing calculations

Data restructuring

- Removing columns
- Combining data sources

Data Wrangling

- We usually need to transform and/or combine data so that it can be used more effectively for analysis
- This is also known as
    - Data cleaning
    - Data wrangling
    - Data transformation
    - Data munging
    - Data remediation
    - Feature extraction

Data Wrangling with Pandas

Handling Missing Data

How do we handle missing data?

To handle missing data, we must:

- Identify we have missing data from our DataFrame
- Determine, to the best of our ability, the cause of this missingness
- Justify how we will handle the missing data (drop or fill in with a specific value?)

Pro tip: The faster you understand *why* some observations are missing, the faster and more accurately you can handle them.

Identifying Missing Data

- Missing Data in Pandas is marked as NaN.
- For example, lets say we have a DataFrame `orders`:

	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales	quantity	discount	profit	profit_margin
0	ID-2022-83625	28/07/2022	31/07/2022	Second Class	RS-19420	FUR-BO-10000008	465.156	2	0.4	-255.864	-0.55
1	IN-2020-85480	31/07/2020	02/08/2020	First Class	CS-12490	FUR-BO-10000021	243.060	2	NaN	102.060	NaN
2	IN-2020-21206	07/02/2020	12/02/2020	NaN	SC-20800	FUR-BO-10000035	1236.330	3	0.0	519.210	NaN
3	IN-2019-50060	07/09/2019	14/09/2019	NaN	MC-17575	FUR-BO-10000035	2472.660	6	NaN	1038.420	NaN
4	IN-2019-25889	08/12/2019	12/12/2019	Standard Class	BP-11185	FUR-BO-10000035	2596.293	7	0.1	923.013	NaN

Identifying Missing Data

- Methods to check whether there is missing data in a DataFrame `df`:
    - **`df.notnull()`** evaluates to True when the data is **not** missing
    - **`df.isnull()`** evaluates to True when data **is** missing

```python
# here is a quick and dirty way to do it
orders.isnull().sum()

# counts the number of missing values
#in each column of the dataframe
```

```
order_id          0
order_date        0
ship_date         0
ship_mode         4
customer_id       0
product_id        0
sales             0
quantity          0
discount         24
profit            0
profit_margin    11
dtype: int64
```

Understanding Missing Data

- Once we know there is missing data, we need to know
  - **Why** the data is missing
  - **What** to do next

Filling in Missing Data

- Once we understand why the data is missing, we may:
  - Delete missing data altogether

  - Fill in missing data with the most likely value:
    - The average of the column
    - The median of the column
    - A predicted amount based on other factors

  - Collect more data:
    - Resample the population
    - Follow up with the authority providing data that is missing

Discussion:
Deleting Missing Data

1 minute

- Option 1: Delete missing data altogether.
    - Deleting missing data means **deleting the entire row or column** that contains the missing data.
    - When might you do this?

Notebooks

- Unit-1-05 Lesson 1: data-wrangling-pandas

Option 1: Deleting Missing Data

- Check how many missing values there are for 'ship_mode':

```
# let's get a value count with the nulls included
orders['ship_mode'].value_counts(dropna=False)
```

```
Standard Class    6611
Second Class      2199
First Class       1576
Same Day           533
NaN                  4
Name: ship_mode, dtype: int64
```

The keyword argument (kwarg) **`dropna=False`** means **don't** drop nulls from the value count!

Option 1: Deleting Missing Data

- Check what will happen if we drop the missing values in 'ship_mode':

```
# drops rows where any row has a missing value -
# this does not happen *in place*,
# so we are not actually dropping any rows

orders['ship_mode'].dropna()
```

```
0          Second Class
1           First Class
4        Standard Class
5          Second Class
6           First Class
              ...
10918          Same Day
10919      Second Class
10920    Standard Class
10921    Standard Class
10922    Standard Class
Name: ship_mode, Length: 10919, dtype: object
```

We would have 10919 rows left

Option 1: Deleting Missing Data

- The dropna() method is used to drop rows or columns.
- For example, using it on the *orders* DataFrame:

```
orders.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)
```

- Parameters:
  - *axis*: **0** – drop **rows** which contain missing values, **1** – drop **columns** which contain missing values
  - *how:* 'any' – if **any** NA values are present, 'all' = if **all** values are NA
  - *thresh*: how **many** NA values should be present before dropping (cannot be combined with how)
  - *subset*: which columns to check the NA values in,
    e.g. `subset=['ship_mode', 'discount', 'profit_margin']`
  - *inplace*: **False**: don't change the source DataFrame, **True** drops from the source.

Option 1: Deleting Missing Data

```
# drops all nulls from the ship_mode column,
#but returns the entire dataframe instead of just the ship_mode column

orders.dropna(subset=['ship_mode'])
```

	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales	quantity	discount
0	ID-2022-83625	28/07/2022	31/07/2022	Second Class	RS-19420	FUR-BO-10000008	465.1560	2	0.40
1	IN-2020-85480	31/07/2020	02/08/2020	First Class	CS-12490	FUR-BO-10000021	243.0600	2	NaN
4	IN-2019-25889	08/12/2019	12/12/2019	Standard Class	BP-11185	FUR-BO-10000035	2596.2930	7	0.10
5	IN-2022-23894	02/04/2022	05/04/2022	Second Class	LP-17095	FUR-BO-10000035	766.5246	2	0.07
6	IN-2021-25560	26/07/2021	29/07/2021	First Class	GH-14425	FUR-BO-10000035	1236.3300	3	0.00

Option 2: Fill in Missing Values

- Traditionally, we fill missing data with a median, average, or mode (most frequently occurring value).
- For 'ship_mode', let's replace it with the **mode**, 'Standard Class', using **fillna()** to fill missing values.

```
1  orders['ship_mode'].fillna(value="Standard Class")
```

```
0          Second Class
1           First Class
2        Standard Class
3        Standard Class
4        Standard Class
              ...
10918          Same Day
10919      Second Class
10920    Standard Class
10921    Standard Class
10922    Standard Class
Name: ship_mode, Length: 10923, dtype: object
```

Option 2: Fill in Missing Values

- Fill in values with a formula based on the column or other columns!

```
orders.fillna(value={'ship_mode':orders['ship_mode'].mode()[0],
                     'discount':orders['discount'].mean(),
                     'profit_margin':orders['profit']/orders['sales']}).head(5)
```

	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales	quantity	discount	profit	profit_margin
0	ID-2022-83625	28/07/2022	31/07/2022	Second Class	RS-19420	FUR-BO-10000008	465.156	2	0.400000	-255.864	-0.550000
1	IN-2020-85480	31/07/2020	02/08/2020	First Class	CS-12490	FUR-BO-10000021	243.060	2	0.149847	102.060	0.419896
2	IN-2020-21206	07/02/2020	12/02/2020	Standard Class	SC-20800	FUR-BO-10000035	1236.330	3	0.000000	519.210	0.419961
3	IN-2019-50060	07/09/2019	14/09/2019	Standard Class	MC-17575	FUR-BO-10000035	2472.660	6	0.149847	1038.420	0.419961
4	IN-2019-25889	08/12/2019	12/12/2019	Standard Class	BP-11185	FUR-BO-10000035	2596.293	7	0.100000	923.013	0.355512

Option 2: Fill in Missing Values

- Filling the missing values in the DataFrame by using a dictionary for the related columns:

```
1  orders.fillna(value={"ship_mode":"Standard Class","discount":0}).head(10)
```

	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales	quantity	discount	profit	profit_margin
0	ID-2022-83625	28/07/2022	31/07/2022	Second Class	RS-19420	FUR-BO-10000008	465.1560	2	0.40	-255.8640	-0.55
1	IN-2020-85480	31/07/2020	02/08/2020	First Class	CS-12490	FUR-BO-10000021	243.0600	2	0.00	102.0600	NaN
2	IN-2020-21206	07/02/2020	12/02/2020	Standard Class	SC-20800	FUR-BO-10000035	1236.3300	3	0.00	519.2100	NaN
3	IN-2019-50060	07/09/2019	14/09/2019	Standard Class	MC-17575	FUR-BO-10000035	2472.6600	6	0.00	1038.4200	NaN
4	IN-2019-25889	08/12/2019	12/12/2019	Standard Class	BP-11185	FUR-BO-10000035	2596.2930	7	0.10	923.0130	NaN

Quick Review

- We can check for missing values using isnull() or notnull()
- Then we need understand why the values are missing, so that we can decide whether to:
    - Drop missing data,
    - Impute values into the missing data, or
    - Don't do anything! Decide whether to leave it or correct it, or find more data.
- Next: Merging Data

Data Wrangling with Pandas

Merging Data

Merging Data

- You might have noticed that the orders and products data are related, but are loaded into different DataFrames:

	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales
0	ID-2022-83625	28/07/2022	31/07/2022	Second Class	RS-19420	FUR-BO-10000008	465.156
1	IN-2020-85480	31/07/2020	02/08/2020	First Class	CS-12490	FUR-BO-10000021	243.060
2	IN-2020-21206	07/02/2020	12/02/2020	Standard Class	SC-20800	FUR-BO-10000035	1236.330

	product_id	category	sub_category	product_name	product_cost_to_consumer
0	FUR-BO-10000008	Furniture	Bookcases	Sauder Library with Doors, Traditional	360.51
1	FUR-BO-10000021	Furniture	Bookcases	Dania Corner Shelving, Metal	70.50
2	FUR-BO-10000035	Furniture	Bookcases	Dania Classic Bookcase, Pine	239.04

Merging Data

- To perform more analysis on the data, we will need to JOIN the DataFrames just like how we join tables in SQL.
- We can do this using the **pd.merge()** function:

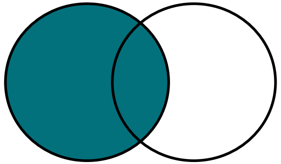
```
pd.merge(left, right, how='inner', on=IndexLabel,
                                   left_on=IndexLabel,
                                   right_on=IndexLabel)
```

- Parameters:
    - `how`: specify whether it is an **'inner', 'left', 'right'** or **'outer**' join
    - `on`: the column name to join on, if both have DataFrames have same column name
    - `left_on, right_on`: if the column names are not the same in both DataFrames, specify the columns that should be matched.

Merging with Left Join

Employees

Employee ID	Employee Name
E2009	Joe Markus
E2010	Abby Chen
E2011	Michael Caine



Departments

Dept ID	Dept Name	Manager ID
D01	Sales	E2009
D02	Marketing	E2010
D03	Finance	-

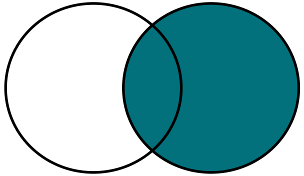
ALL Employees (LEFT) with Departments (RIGHT) ON EmployeeID=ManagerID

Employee ID	Employee Name	Dept ID	Dept Name	Manager ID
E2009	Joe Markus	D01	Sales	**E2009**
E2010	Abby Chen	D02	Marketing	**E2010**
E2011	Michael Caine	-	-	-

Merging with Right Join

Employees

Employee ID	Employee Name
E2009	Joe Markus
E2010	Abby Chen
E2011	Michael Caine



Departments

Dept ID	Dept Name	Manager ID
D01	Sales	E2009
D02	Marketing	E2010
D03	Finance	-

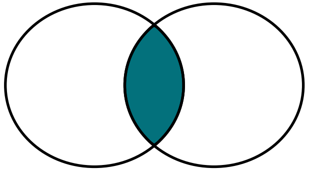
ALL Departments (RIGHT) with Employees (LEFT) ON EmployeeID=ManagerID

Employee ID	Employee Name	Dept ID	Dept Name	Manager ID
E2009	Joe Markus	D01	Sales	**E2009**
E2010	Abby Chen	D02	Marketing	**E2010**
-	-	D03	Finance	-

Merging with Inner Join

Employees

Employee ID	Employee Name
E2009	Joe Markus
E2010	Abby Chen
E2011	Michael Caine



Departments

Dept ID	Dept Name	Manager ID
D01	Sales	E2009
D02	Marketing	E2010
D03	Finance	-

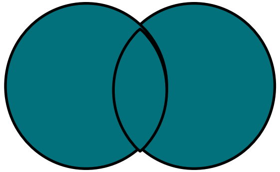
Only Employees (LEFT) and matching Departments (RIGHT) ON EmployeeID=ManagerID

Employee ID	Employee Name	Dept ID	Dept Name	Manager ID
E2009	Joe Markus	D01	Sales	E2009
E2010	Abby Chen	D02	Marketing	E2010

Merging with Outer Join

Employees

Employee ID	Employee Name
E2009	Joe Markus
E2010	Abby Chen
E2011	Michael Caine



Departments

Dept ID	Dept Name	Manager ID
D01	Sales	E2009
D02	Marketing	E2010
D03	Finance	-

ALL Employees (LEFT) and ALL Departments (RIGHT) matched ON EmployeeID=ManagerID where possible

Employee ID	Employee Name	Dept ID	Dept Name	Manager ID
E2009	Joe Markus	D01	Sales	**E2009**
E2010	Abby Chen	D02	Marketing	**E2010**
E2011	Michael Caine	-	-	-
-	-	D03	Finance	-

Guided Walk-Through:

Merging Orders and Products

- Let's try to merge the **`orders`** and **`products`**, so that we can analyse the sales by products.
    - Read the 'products.csv' file into a DataFrame called **`products`**
    - Examine it to check the columns
    - Check the **`orders`** DataFrame
    - Perform the merge using **`pd.merge()`**

Merging Data

```python
# Merge orders (left) with products (right) using product_id

merged = pd.merge(orders, products, how='left', on='product_id')
merged.head()
```

	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales	quantity	discount	profit	profit_margin	category	sub_category	produc
0	ID-2022-83625	28/07/2022	31/07/2022	Second Class	RS-19420	FUR-BO-10000008	465.156	2	0.400000	-255.864	-0.550000	Furniture	Bookcases	Saude with Tr
1	IN-2020-85480	31/07/2020	02/08/2020	First Class	CS-12490	FUR-BO-10000021	243.060	2	0.149847	102.060	0.419896	Furniture	Bookcases	Dania Shelvin
2	IN-2020-21206	07/02/2020	12/02/2020	Standard Class	SC-20800	FUR-BO-10000035	1236.330	3	0.000000	519.210	0.419961	Furniture	Bookcases	Dania Bc
3	IN-2019-50060	07/09/2019	14/09/2019	Standard Class	MC-17575	FUR-BO-10000035	2472.660	6	0.149847	1038.420	0.419961	Furniture	Bookcases	Dania Bc
4	IN-2019-25889	08/12/2019	12/12/2019	Standard Class	BP-11185	FUR-BO-10000035	2596.293	7	0.100000	923.013	0.355512	Furniture	Bookcases	Dania Bc

Quick Review

- We can join DataFrames using Pandas' pd.merge()
- This is similar to an SQL join where we can specify the type of join using the how= parameter.
    - Inner
    - Outer
    - Left
    - Right

NEXT: Grouping by categories

Data Wrangling with Pandas

Apply Functions

Apply Functions

- Apply functions allow us to perform a complex operation across an entire column highly efficiently**.**
    - **Clean the data**
    - **Perform calculations**
    - **Create new columns**
- There are three steps to this approach:
    - First we write a function that receives a value from each cell in the column. The function will perform some processing and return a result.
    - Then we use **apply()** to *apply* the function to the column to obtain the results for the entire column
    - We can save the result back to mutate the source dataframe if we want.

Example: Apply Functions

- Let's say we want to classify the margin category for the profit margin as **low, medium** or **high**.

sales	quantity	discount	profit	profit_margin
465.156	2	0.400000	-255.864	-0.550000
243.060	2	0.149847	102.060	0.419896
1236.330	3	0.000000	519.210	0.419961
2472.660	6	0.149847	1038.420	0.419961
2596.293	7	0.100000	923.013	0.355512

We want to classify each value in the profit_margin column

First: Write the Function

- We need a function that
  - Receives a profit_margin value as an argument
  - Determines if the margin category is low, medium or high
  - Returns the string with the margin category value

```
def margin_category(profit_margin):
    if profit_margin >=0.3:
        return 'High'
    elif profit_margin >= 0.1:
        return 'Medium'
    else:
        return 'Low'
```

Next: Apply the Function

- Now we can **apply** the function to the appropriate column:

Column to apply to

Name of function

```
merged['profit_margin'].apply(margin_category)
```

```
0         Low
1        High
2        High
3        High
4        High
         ...
10918    High
10919    High
10920    High
10921     Low
10922    High
Name: profit_margin, Length: 10923, dtype: object
```

Finally: Save the Result

- If we want, we can create a new column with the results!

```
merged['margin_category'] = merged['profit_margin'].apply(margin_category)
```

New column

Lambda Expressions

- We can use a lambda expression to apply a simple calculation without having to write a function:

```
# add 100 dollars to each product cost
merged['product_cost_to_consumer'].apply(lambda x : x+100)
```

```
10913    145.48
10914    142.51
10915    142.51
10916    142.51
10917    221.50
10918    191.89
10919    191.89
10920    191.89
10921    191.89
10922    191.89
Name: product_cost_to_consumer, dtype: float64
```

Data Wrangling with Pandas

Wrapping Up

Data Wrangling!

We've done quite a lot of data wrangling in this unit!

- We identified and imputed missing data in the **orders** DataFrame.
- We **merged** the **orders** and **products** DataFrames
- We performed some **groupby** operations to perform aggregations on the data by category
- We added a new column by using an **apply** function to another column

Should we save all these changes?

Saving the File

We can save the file using the Pandas `to_csv()` function, which will save the file with comma-separated values.

```
# Saving the merged data to a new file called orders_by_product.csv, without adding an index column
merged.to_csv('orders_by_product.csv', index=False)
```

This will save the file in the current directory.

Data Wrangling with Pandas

Groupby Statements

Groupby Statements

- In Pandas, groupby statements are similar to pivot tables in that they allow us to segment our population to a specific subset.
- To think how a groupby statement works, think about it like this:
  - First we splits the DataFrame by a specific attribute, for example, group by 'ship_mode'
  - Then we put our DataFrame back together and return some aggregated metric, such as the **mean, sum, count or max** for each group.

Groupby Statements

['quantity'].mean()

groupby(['ship_mode'])

ship_mode	quantity
Second Class	2
First Class	3
Standard Class	3
Standard Class	3
First Class	3
Second Class	1
First Class	4

ship_mode	quantity
Second Class	2
Second Class	1

1.5

ship_mode	quantity
First Class	3
First Class	3
First Class	4

3.33

ship_mode	quantity
Standard Class	3
Standard Class	3

3

ship_mode	quantity
Second Class	1.5
First Class	3.33
Standard Class	3

Example: Groupby and count()

- Counting the number of 'order_id' for each 'ship_mode':

```
# Counting AFTER we group by ship mode
orders.groupby(['ship_mode'])['order_id'].count()
```

```
ship_mode
First Class       1576
Same Day           533
Second Class      2199
Standard Class    6615
Name: order_id, dtype: int64
```

Example: Groupby() and max()

- We can get aggregated values for numeric columns, for example finding the highest sales value for each ship mode.

```python
# find the max sales by ship mode, return as a Series
orders.groupby('ship_mode')['sales'].max()
```

```
ship_mode
First Class       5175.1710
Same Day          3741.5238
Second Class      5667.8700
Standard Class    6998.6400
Name: sales, dtype: float64
```

Selecting Columns Before GroupBy

- We can index the DataFrame first:
  - Choose the groupby column and the columns to aggregate
  - Aggregation will be automatically performed on the non-groupby column

```
# Index the dataframe first to choose the groupby column and the aggregation column(s)
# Here we want to sort the results by sales column, it returns a DataFrame
orders[['ship_mode','sales']].groupby('ship_mode').mean().sort_values('sales',ascending=False)

```

	sales
ship_mode	
Standard Class	335.623751
First Class	316.308534
Second Class	309.648138
Same Day	303.988495

Aggregation on other Columns

- If the DataFrame is not indexed, the aggregation will be performed on all the non-groupby columns where possible:

```
# find the mean values for all other columns in the DataFrame by ship mode
orders.groupby('ship_mode').mean()
```

ship_mode	sales	quantity	discount	profit	profit_margin
First Class	316.308534	3.704949	0.155368	36.882062	0.059780
Same Day	303.988495	3.696060	0.151219	34.858326	0.080713
Second Class	309.648138	3.731241	0.144770	34.867296	0.078786
Standard Class	335.623751	3.776871	0.150108	41.894980	0.066416

Solo Exercise:

Find Total Profit by Ship Mode

1 minute

- Find the sum of profit for each ship mode in orders.

Solo Exercise:

Find Total Profit by Ship Mode

1 minute

- Find the sum of profit for each ship mode in orders.

```
1  # return a series
2  orders.groupby('ship_mode')['profit'].sum()
```

```
ship_mode
First Class        58126.1292
Same Day           18579.4878
Second Class       76673.1850
Standard Class    277135.2960
Name: profit, dtype: float64
```

```
1  # return a dataframe
2  orders[['ship_mode','profit']].groupby('ship_mode').sum()
```

ship_mode	profit
First Class	58126.1292
Same Day	18579.4878
Second Class	76673.1850
Standard Class	277135.2960

Multiple Aggregations on the Same Column

- We can also use the `agg()` method with multiple arguments to perform multiple aggregations on the same column.
- Here the column must be specified.

```
orders.groupby('ship_mode')['sales'].agg(['count','mean','min','max'])
```

ship_mode	count	mean	min	max
First Class	1576	316.308534	4.4100	5175.1710
Same Day	533	303.988495	6.5400	3741.5238
Second Class	2199	309.648138	2.8800	5667.8700
Standard Class	6615	335.623751	3.3231	6998.6400

Multi-Level Groupby

- We can specify more than one column to group by, for example grouping by ship mode **and** category for the merged DataFrame:

	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales	quantity	discount	profit	profit_margin	category	sub_category
0	ID-2022-83625	28/07/2022	31/07/2022	Second Class	RS-19420	FUR-BO-10000008	465.156	2	0.400000	-255.864	-0.550000	Furniture	Bookcases
1	IN-2020-85480	31/07/2020	02/08/2020	First Class	CS-12490	FUR-BO-10000021	243.060	2	0.149847	102.060	0.419896	Furniture	Bookcases
2	IN-2020-21206	07/02/2020	12/02/2020	Standard Class	SC-20800	FUR-BO-10000035	1236.330	3	0.000000	519.210	0.419961	Furniture	Bookcases
3	IN-2019-50060	07/09/2019	14/09/2019	Standard Class	MC-17575	FUR-BO-10000035	2472.660	6	0.149847	1038.420	0.419961	Furniture	Bookcases
4	IN-2019-25889	08/12/2019	12/12/2019	Standard Class	BP-11185	FUR-BO-10000035	2596.293	7	0.100000	923.013	0.355512	Furniture	Bookcases

Group by Two Columns

- As before, selecting the column to aggregate returns a Series

```
# Return the result as a Series
merged.groupby(['ship_mode','category'])['order_id'].count()

```

```
ship_mode       category
First Class     Furniture          358
                Office Supplies    881
                Technology         337
Same Day        Furniture          108
                Office Supplies    302
                Technology         123
Second Class    Furniture          469
                Office Supplies   1271
                Technology         459
Standard Class  Furniture         1478
                Office Supplies   3677
                Technology        1460
Name: order_id, dtype: int64
```

Multi-Level GroupBy

```python
# Index the required columns first, then group by to return a DataFrame
merged[['ship_mode','category','order_id']].groupby(['ship_mode','category']).count()
```

		order_id
ship_mode	**category**	
First Class	**Furniture**	358
	Office Supplies	881
	Technology	337
Same Day	**Furniture**	108
	Office Supplies	302
	Technology	123
Second Class	**Furniture**	469
	Office Supplies	1271
	Technology	459
Standard Class	**Furniture**	1478
	Office Supplies	3677
	Technology	1460

- The multi-level groupby returns a multi-index DataFrame where the indexes are the groupby levels

Unstack

- We can apply **`unstack()`** to the multi-indexed DataFrame so that the inner index will be pivoted to be the row header.

```python
# Return the result as a Series
merged.groupby(['ship_mode','category'])['order_id'].count()
```

```
ship_mode       category
First Class     Furniture          358
                Office Supplies    881
                Technology         337
Same Day        Furniture          108
                Office Supplies    302
                Technology         123
Second Class    Furniture          469
                Office Supplies   1271
                Technology         459
Standard Class  Furniture         1478
                Office Supplies   3677
                Technology        1460
Name: order_id, dtype: int64
```

```python
# Unstack the columns so that one becomes a row header
merged.groupby(by=['ship_mode','category'])['order_id'].count().unstack()
```

category ship_mode	Furniture	Office Supplies	Technology
First Class	358	881	337
Same Day	108	302	123
Second Class	469	1271	459
Standard Class	1478	3677	1460

Quick Review

- The **groupby()** method allows us to split the DataFrame by category and obtain aggregated values for each category.

Q&A

Notebooks

- Unit 1-05Lesson 2: Grouping and Summarizing Data
  - Lesson Notebook: grouping-and-summarizing-data

Homework

- Complete the Exercises

Recap

In this unit, we:

- Identified and imputing missing data
- Merged two DataFrames into one based on a key column
- Used Groupby statements to group categories of data and apply aggregation functions
- Processed a column of data using an apply() function

Looking Ahead

Homework : Grouping and Summarizing Data Exercises

- Merge two DataFrames
- Count missing values
- Fill missing values
- Groupby
- Apply Function
- Save the File

Up Next: Data Visualization with Pandas

COURSE CONTENT

Week 1 : Data Science Foundations

Congratulations!

Installation and Github, Python fundamentals, Introduction to Pandas

Week 2 : Working with Data

More pandas, basics of probability and statistics, Exploratory Data Analysis (EDA), working with data, use statistical analysis and visualisation

Week 3 : Data Science Modeling

Linear regression Train/Test/Split, Classification, Logistic Regression

Week 4 : Data Science Applications

Using APIs, Natural Language Processing, Time Series Analysis

Week 5: Final Presentation

Present your capstone project