PER SCHOLAS

Advance Python Pandas and Data Analytics

Learning Objectives

By the end of the lesson, learners will be able to:

- Apply data cleaning and preprocessing methods to ensure data quality and integrity.
- Define advanced data manipulation concepts, such as merging, concatenating, and reshaping DataFrames.
- Identify the statistical functions and aggregations in pandas for descriptive and exploratory data analysis.
- Apply “GroupBy” operations to analyze data at different levels of granularity.
- Apply pandas skills to real-world datasets and case studies, solving practical problems in various domains.
- Utilize concat() function for concatenating two or more DataFrames.
- Execute the principles of efficient data processing for large datasets.
- Use the get_dummies() function to convert categorical variables into dummy or indicator variables.
- Perform cumulative calculations using functions like cumsum().

Table of Contents

Sample Data Set Resources

In this lesson, we will use the Datasets below for demonstrations., we will use the Datasets below for demonstrations.

- Cars.json
- RealEstate1.csv
- RealEstate2.csv
- RealEstate3.csv
- student_scores.csv
- sales_transaction.csv

Section 1

Merging with Pandas

PER SCHOLAS

Merge, Join, and Concatenate DataFrames

- Pandas provides various facilities for easily combining together Series or DataFrame with various kinds of set logic for the indexes and relational algebra functionality in the case of join / merge-type operations.
- In addition, pandas also provides utilities to compare two Series or DataFrame and summarize their differences.
- Some of the popular functions are:
    - pandas.concat() function
    - pandas.append() function
    - pandas.merge() function
    - pandas.join() function

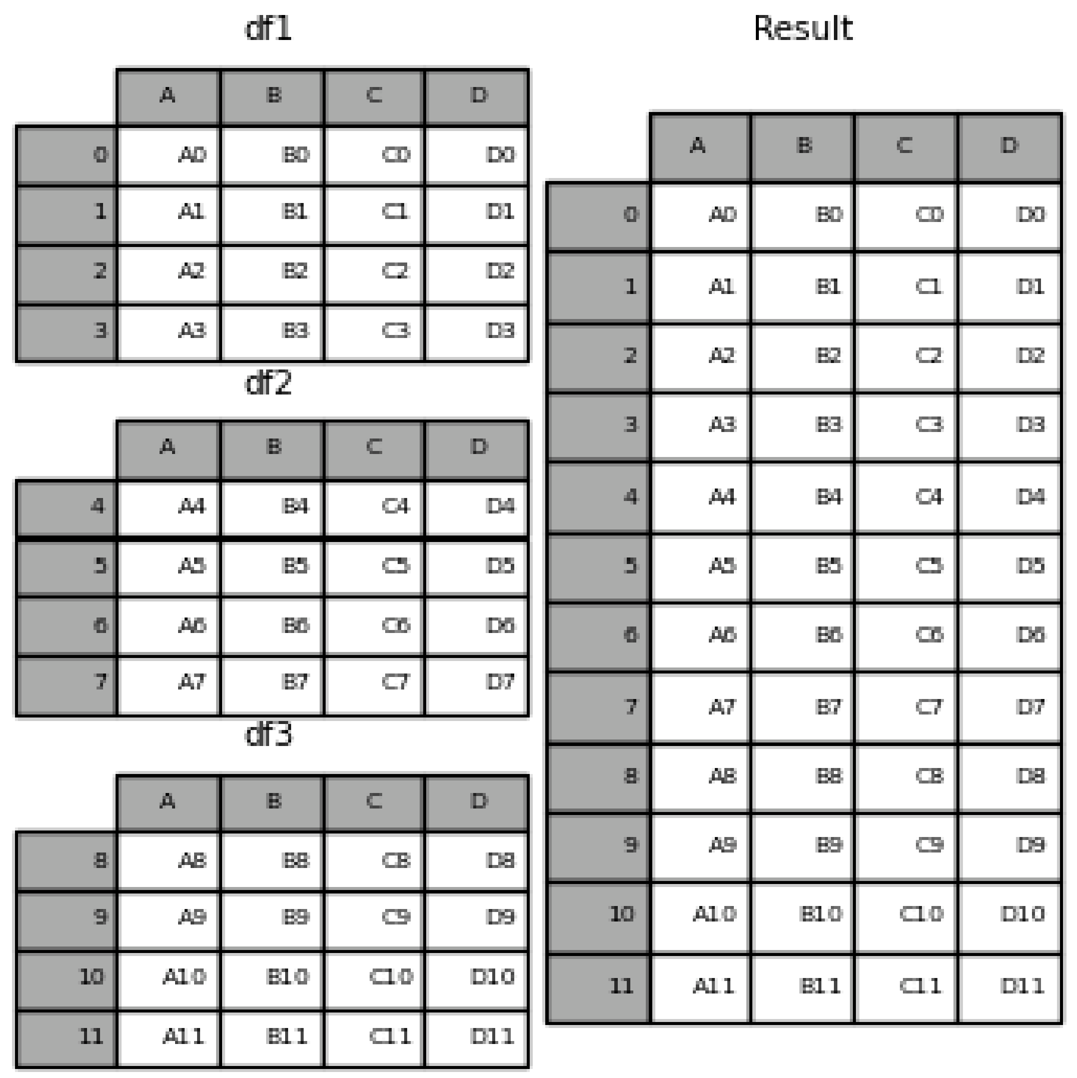
All of these functions are very similar but join() is considered a more efficient way to join indices. pandas.concat() and pandas.append() functions concatenate two DataFrames by setting axis=1. and merge() is considered the most efficient to combine DataFrames on multiple columns.

Overview of Pandas concat() Function

Pandas **pandas.concat()** function is used to concatenate pandas objects into a DataFrame output. By default, it performs append operations similar to a union where it bright all rows from both DataFrames to a single DataFrame. Below is the **syntax of concat():**

```
pandas.concat(objs, *, axis=0, join='outer', ignore_index=False, names=None, sort=False
```

- **objs : a sequence or mapping of Series or DataFrame objects. If a dict is passed, the sorted keys will be used as the keys argument, unless it is passed, in which case the values will be selected (see below). Any None objects will be dropped silently unless they are all None in which case a ValueError will be raised.**
- **axis : {0, 1, ...}, default 0. The axis to concatenate along.**
- **join : {'inner', 'outer'}, default 'outer'. How to handle indexes on other axis(es). Outer for union and inner for intersection.**
- **ignore_index : boolean, default False. If True, do not use the index values on the concatenation axis. The resulting axis will be labeled 0, ..., n - 1. This is useful if you are concatenating objects where the concatenation axis does not have meaningful indexing information. Note the index values on the other axes are still respected in the join.**
- **names : list, default None. Names for the levels in the resulting hierarchical index.**



df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Guided Lab - Concatenation

For the concat() function demonstration, visit **Guided Lab - 343.4.1 - Concatenate Multiple Data Frames.** You can find this lab on Canvas under the Assignment section.

PER SCHOLAS

Overview of Pandas merge() Function

- The **pandas.merge()** function merge two DataFrames based on a common column or index. It resembles SQL's JOIN operation and offers more control over how DataFrames are combined.
- Using merge(), you can merge by column, by index, on multiple columns, and on different join types. By default, merge() merges on all common columns that exist on both DataFrames and performs an inner join.

- The basic syntax for the pandas.merge() function is:

```
pd.merge(left, right, on=None, left_on=None, right_on=None, left_index=False,
right_index=False, how='inner', suffixes=('_x', '_y'), copy=True)
```

Example: merge() Function

This example demonstrates how to use the pd.merge() function in Pandas to perform a database-style join operation on two DataFrames (df1 and df2) based on the common column 'subject_id'. Let's break down the code step by step:

```
import pandas as pd
demoData_One = {
  'subject_id': ['1', '2', '3', '4', '5'],
  'student_name': ['Mark', 'Khalid', 'Deborah', 'Trevon', 'Raven']
}
df1 = pd.DataFrame(demoData_One, columns=['subject_id', 'student_name'])
print(df1)
demoData_Two = {
  'subject_id': ['4', '5', '6', '7', '8'],
  'student_name': ['Eric', 'Imani', 'Cece', 'Darius', 'Andre']
}
df2 = pd.DataFrame(demoData_Two, columns=['subject_id', 'student_name'])
print(df2)
pd.merge(df1, df2, on='subject_id')
```

Resulr: The merged DataFrame will look like this:

```
subject_id  student_name_x  student_name_y
     0    4     Trevon        Eric
     1    5     Raven         Imani
```

PER SCHOLAS

Guided Lab - Merge Pandas DataFrames

Complete the lab: **Guided LAB - 343.4.2 - How to merge Pandas DataFrames by multiple columns**. You can find this lab on Canvas under the Assignment section.

If you have technical questions while performing the labs, ask your instructors for assistance.

Overview of Pandas join() Function

- The **pandas.join()** function combines two DataFrames based on their index values or common column. It allows join DataFrames with different columns while preserving the index(label) structure. The basic syntax for the **pandas.join()** function is:

```
pandas.join(other, on=None, how='left', lsuffix='', rsuffix='', sort=False)
```

By default, join uses the left join on the row index. If you want to join on columns you should use pandas.merge() function as this by default performs on columns.

Guided Lab - Join Pandas DataFrames

Please Complete the **Guided Lab - 343.4.3 - How to join Multiple DataFrames columns using join()**. You can find this lab on Canvas under the Assignment section.

If you have technical questions while performing the labs, ask your instructors for assistance.

PER SCHOLAS

Choosing between concat(), join(), and merge()

Selecting between concat(), join(), and merge() depends on specific needs and the data structure you are working with. Here are some general guidelines:

Function Description	Advantages / Disadvantages
Use **‘concat()’** for combining DataFrames or Series along a particular axis (rows or columns) without considering any common keys or indexes. It is best suited for merging datasets with similar structures for further analysis.	• Advantages: Versatile, handles simple concatenations. • Disadvantages: Requires additional steps for complex merges.
Use **‘join()’** for combining DataFrames based on their index values. It’s useful when DataFrames have different columns but share an index structure.	• Advantages: Powerful for merging on common columns. • Disadvantages: Can be complex for new users.
Use **‘merge()’** for combining DataFrames based on a common column or index. It provides more control over how DataFrames are combined and resembles SQL’s JOIN operation.	• Advantages: Ideal for merging on indexes. • Disadvantages: Limited to index-based merging.

Knowledge Check

- What is the purpose of concatenating two DataFrames in Pandas?
- How can I concatenate two DataFrames in Pandas?
- How do you specify the columns that should be used for merging two dataframes?

Section 2

Grouping and Aggregation

PER SCHOLAS

Overview of Pandas Aggregate Functions

- An aggregate is a function where the values of multiple rows are grouped together to form a single summary value.

- Pandas also supports multiple aggregate functions that perform a calculation on a set of values (grouped data) and return a single value.

- The most common aggregation functions are a simple average or summation of values. In Pandas, you may call an aggregation function on one or more columns of a DataFrame.

PER SCHOLAS

Overview of Pandas Aggregate Functions

- An aggregate is a function where the values of multiple rows are grouped together to form a single summary value. Pandas also offers the **aggregate()** function, which takes another function (or list of functions) as its argument, returning the name of the function as the index and the result of the function's application for each column.
- The official documentation says: **agg()** is an alias for **aggregate()** function.
- Below are some of the aggregate functions supported by pandas using:
  - `DataFrame.aggregate(func=None, axis=0, *args, **kwargs)`
  - `Series.aggregate(func=None, axis=0, *args, **kwargs)`
  - `DataFrameGroupBy.aggregate(func=None, *args, **kwargs)`

Parameter:

func - an aggregate function like sum, mean, etc.

axis - specifies whether to apply the aggregation operation along rows or columns.

***args and **kwargs - additional arguments that can be passed to the aggregation functions.**

PER SCHOLAS

Important Pandas Aggregate Functions

AGGREGATE FUNCTIONS	DESCRIPTION
count()	Returns count for each group.
size()	Returns size for each group.
sum()	Returns total sum for each group.
mean()	Returns mean for each group. Same as average().
average()	Returns average for each group. Same as mean().
min()	Returns minimum value for each group.
max()	Returns maximum value for each group.

AGGREGATE FUNCTIONS	DESCRIPTION
first()	Returns first value for each group.
cumsum()	Cumulative sum of a column in Pandas
last()	Returns last value for each group.
nth()	Returns nth value for each group.
std()	Standard deviation of column
var()	Compute variance of column

Hands-On Lab: Apply Single Aggregate Function

Please Complete the Guided LAB - 343.4.4 - Pandas Aggregate Function. You can find this lab on Canvas under the Assignment section.

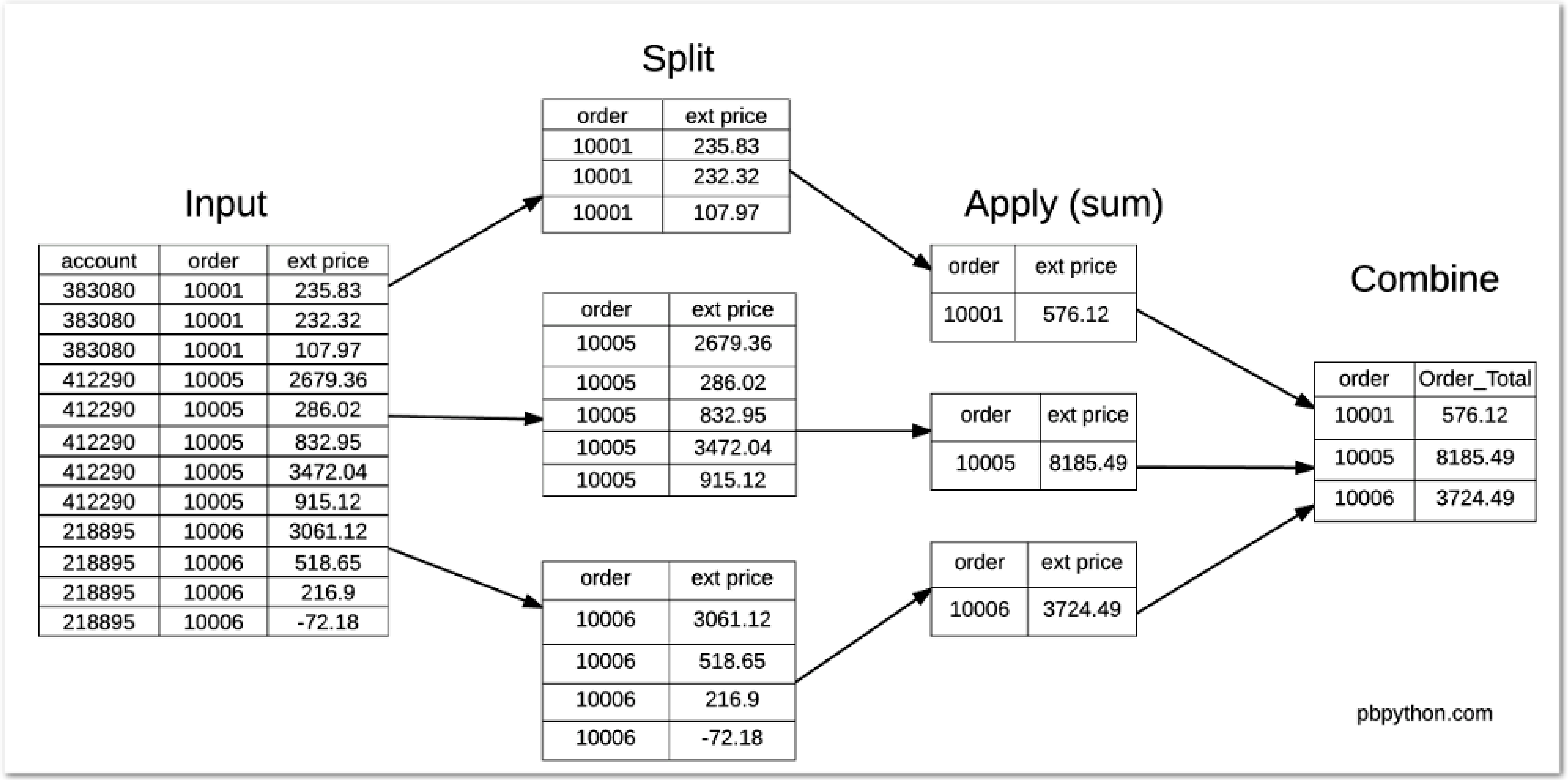
If you have technical questions while performing the labs, ask your instructors for assistance.

Overview of Python Pandas - GroupBy

- Similar to the SQL **GROUP BY** clause pandas **groupby()** function is used to collect identical data into groups and perform aggregate functions on the grouped data. GroupBy operation involves splitting the data, applying some functions, and finally, aggregating the results.
- Any GroupBy operation involves one of the following operations on the original object, including:
  - **Splitting the object.**
  - **Applying a function.**
  - **Combining the results.**
- In many situations, we split the data into sets and we apply some functionality on each subset. In the apply functionality, we can perform the following operations:
  - **Aggregation – computing a summary statistic.**
  - **Transformation – perform some group-specific operation.**
  - **Filtration – discarding the data with some condition.**

PER SCHOLAS

Overview of Python Pandas - GroupBy



Input

account	order	ext price
383080	10001	235.83
383080	10001	232.32
383080	10001	107.97
412290	10005	2679.36
412290	10005	286.02
412290	10005	832.95
412290	10005	3472.04
412290	10005	915.12
218895	10006	3061.12
218895	10006	518.65
218895	10006	216.9
218895	10006	-72.18

Split

order	ext price
10001	235.83
10001	232.32
10001	107.97

order	ext price
10005	2679.36
10005	286.02
10005	832.95
10005	3472.04
10005	915.12

order	ext price
10006	3061.12
10006	518.65
10006	216.9
10006	-72.18

Apply (sum)

order	ext price
10001	576.12

order	ext price
10005	8185.49

order	ext price
10006	3724.49

Combine

order	Order_Total
10001	576.12
10005	8185.49
10006	3724.49

PER SCHOLAS

Syntax of pandas.groupby()

```
pandas.groupby(by=column or index, axis=0, level=None, as_index=True, sort=True,
group_keys=True,  observed=False, dropna=True)
```

- by – List of column names or index label to group by.
- axis – Default to 0. It takes 0 or 'index', 1 or 'columns.'
- level – Used with MultiIndex.
- as_index – sql style grouped output.
- sort – Default to True. Specify whether to sort after group.
- group_keys – add group keys or not.
- observed – This only applies if any of the groupers are Categorical.
- dropna – Default false. True, and if group keys contain NA values, NA values together with row/column will be dropped. If False, NA values will also be treated as the key in groups.

Reference: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html

PER SCHOLAS

Example 1 - pandas.groupby()

This example uses the pandas library in Python to perform a groupby operation on a DataFrame based on the 'Year' column and then extracts a specific group for the year 2011.

```
import pandas as pd
raw = {
  'Name': ['Darell', 'Darell', 'Lilith', 'Lilith', 'Tran', 'Tran', 'Tran', 'Tran', 'John',
'Darell', 'Darell', 'Darell'],
  'Position': [2, 1, 1, 4, 2, 4, 3, 1, 3, 2, 4, 3],
  'Year': [2009, 2010, 2009, 2010, 2010, 2010, 2011, 2012, 2011, 2013, 2013, 2012],
  'Marks':[408, 398, 422, 376, 401, 380, 396, 388, 356, 402, 368, 378]
}
df = pd.DataFrame(raw)
group = df.groupby('Year')
print(group.get_group(2011))
```

The output shows the rows from the original DataFrame where the 'Year' is 2011, including the 'Name', 'Position', 'Year', and 'Marks' columns. In this case, the group includes two rows with 'Tran' and 'John'.

```
      Name  Position  Year  Marks
6     Tran         3  2011    396
8     John         3  2011    356
```

Example 2 - pandas.groupby()

This example demonstrates how to aggregate columns in a DataFrame both without grouping and with grouping using the aggregate function. The results show the total sum for the specified columns and the sum for each group based on the 'Category' column.

```
import pandas as pd
data = {
  'Category': ['A', 'A', 'B', 'B', 'A', 'B'],
  'Value': [10, 15, 20, 25, 30, 35],
  'Fee' :[20000,25000,26000,22000,24000,35000],
  'Duration':['30day','40days','35days','40days','60days','60days'],
  'Discount':[1000,2300,1200,2500,2000,2000]
}
df = pd.DataFrame(data)
result_sum = df[['Fee','Discount']].aggregate('sum')
print(result_sum)

# Now, let’s see how to group the rows and calculate the sum for each group.
#To do grouping use DataFrame.groupby() function. This function returns the DataFrameGroupBy object
# and use aggregate() function to calculate the sum.
result_Group = df.groupby('Category')['Fee','Discount'].aggregate('sum')
print(result_Group)
```

PER SCHOLAS

Guided Labs - Grouping and Aggregate Functions

Please complete the following labs:

- Guided LAB - 343.4.5 - Pandas Aggregate Functions
- Guided LAB - 343.4.6 - Pandas Grouping and Aggregate Functions

You can find these labs on Canvas under the Assignment section.

Section 3

Cumulative sum , Transform and Difference of a DataFrame

Overview of pandas.cumsum() - 1 of 2

- A cumulative sum is a sequence of partial sums of a given sequence. For example, the cumulative sums of the sequence , are a,a_b,b+c,c_d, so on ...... and the data (value) that are added to the cumulative sum, those data(value) will be positive, and the sum will increase steadily or continuously.

- The pandas.cumsum() function is important in data analysis and time series analysis because it helps in calculating the cumulative sum of a series. This can be particularly useful for scenarios where you want to understand the running total or cumulative effect of a variable over time or across observations.

PER SCHOLAS

Overview of pandas.cumsum() - 2 of 2

- **Trend Analysis:** Cumulative sums help in identifying trends and patterns over time.
- **Performance Tracking:** It is crucial for tracking the overall performance of a variable. In business scenarios, cumulative sums are often used to assess financial performance, inventory levels, or customer acquisition over time.
- **Decision Making:** Cumulative sums provide a quick overview of the total impact, aiding in decision-making processes. For instance, understanding the cumulative revenue helps in assessing whether the company is meeting its financial goals.

PER SCHOLAS

Scenario-Based Example: Sales Revenue Analysis

Let's consider a scenario where you have a DataFrame representing monthly sales revenue for a company. You want to analyze the cumulative revenue over time to understand the company's overall financial performance.

```
import pandas as pd
# Sample DataFrame representing monthly sales revenue
data = {'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jan', 'Feb', 'Mar', 'Apr', 'May'],
  'Revenue': [1000, 1500, 12000, 20000, 18000,2000, 13400, 14000, 10000, 1800],
  'store': ['A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B']}
df = pd.DataFrame(data)

# Calculate the cumulative sum of revenue using cumsum()
df['Cumulative_Revenue'] = df['Revenue'].cumsum()
# Display the DataFrame
print(df)
#calculate and add column that shows cumulative sum of revenue by store
df['revenue_bystore'] = df.groupby(['store'])['Revenue'].cumsum()
print(df)
```

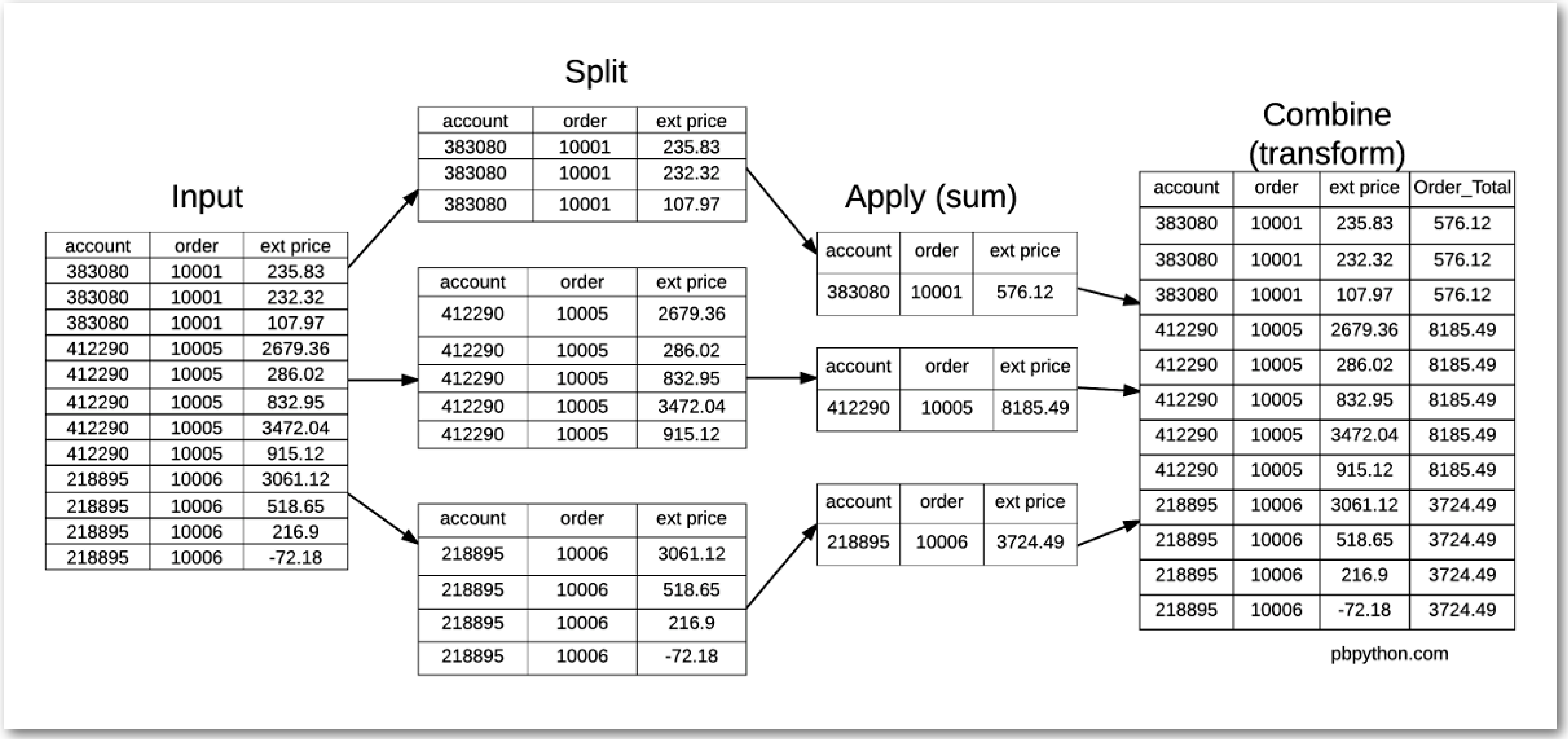
The revenue_bystore column shows the cumulative sales, grouped by each store. In our example, the Cumulative_Revenue column allows us to see how the sales accumulate month by month.

Overview of Python - Transform

- As the name suggests, we extract new features from existing ones.
- In pandas, the **transform()** function is used to perform group-wise operations and produce an output with the same shape as the original DataFrame. It is a powerful tool for broadcasting aggregated results back to the original DataFrame based on the groups defined by the groupby operation.
- While aggregation must return a reduced version of the data, transformation can return some transformed version of the full data to recombine. For such a transformation, the output is the same shape as the input. A common example is to center the data by subtracting the group-wise mean.

PER SCHOLAS

Overview of Python - Transform (continued)



Input

account	order	ext price
383080	10001	235.83
383080	10001	232.32
383080	10001	107.97
412290	10005	2679.36
412290	10005	286.02
412290	10005	832.95
412290	10005	3472.04
412290	10005	915.12
218895	10006	3061.12
218895	10006	518.65
218895	10006	216.9
218895	10006	-72.18

Split

account	order	ext price
383080	10001	235.83
383080	10001	232.32
383080	10001	107.97

account	order	ext price
412290	10005	2679.36
412290	10005	286.02
412290	10005	832.95
412290	10005	3472.04
412290	10005	915.12

account	order	ext price
218895	10006	3061.12
218895	10006	518.65
218895	10006	216.9
218895	10006	-72.18

Apply (sum)

account	order	ext price
383080	10001	576.12

account	order	ext price
412290	10005	8185.49

account	order	ext price
218895	10006	3724.49

Combine (transform)

account	order	ext price	Order_Total
383080	10001	235.83	576.12
383080	10001	232.32	576.12
383080	10001	107.97	576.12
412290	10005	2679.36	8185.49
412290	10005	286.02	8185.49
412290	10005	832.95	8185.49
412290	10005	3472.04	8185.49
412290	10005	915.12	8185.49
218895	10006	3061.12	3724.49
218895	10006	518.65	3724.49
218895	10006	216.9	3724.49
218895	10006	-72.18	3724.49

PER SCHOLAS

Syntax for transform()

- The general syntax for **transform** is:

```
df['new_column'] = df.groupby('group_column')['target_column'].transform(function)
```

Here, **group_column** is the column used for grouping, **target_column** is the column on which you want to perform the transformation, and function is the transformation function applied to each group.

Here is a breakdown of how transform works:

1. **Grouping**:
   - The DataFrame is first grouped based on the specified group_column. This creates separate groups for each unique value in the grouping column.
2. **Transformation**:
   - The specified function is applied to each group independently. This function could be any aggregation or transformation function, such as mean, sum, standard deviation, or a custom function.
3. **Broadcasting:**
   - The results of the transformation are then broadcasted back to the original DataFrame. This is done by aligning the group labels and assigning the transformed values to the corresponding rows in the original DataFrame.

PER SCHOLAS

Guided Lab - Pandas Transform

Please Complete the Guided LAB - 343.4.7 - Understanding the Transform Function in Pandas. You can find this lab on Canvas under the Assignment section.

If you have technical questions while performing the labs, ask your instructors for assistance.

PER SCHOLAS

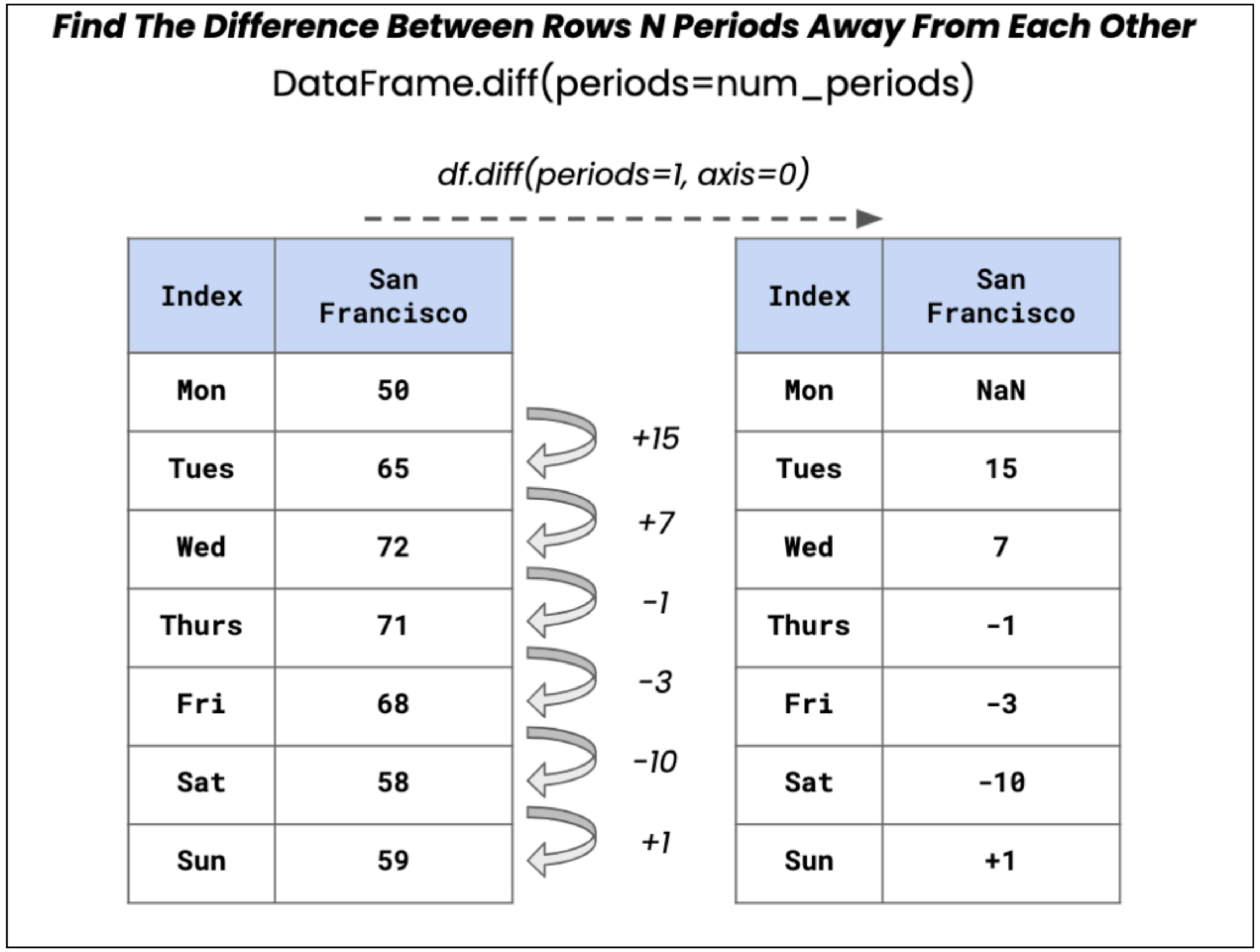
Difference Your Data – pandas.diff()

- The Pandas **diff()** function calculates the difference (or rate of change) of a DataFrame or Series element between rows/columns.
- Pandas **diff()** function will *difference* your data. This means calculating the change in your row(s)/column(s) over a set number of periods. Or simply, pandas diff will subtract 1 cell value from another cell value within the same index.
- The **diff()** is very helpful when calculating rates of change. For example, you have temperature readings per day, calculating the difference will tell you how the temperatures have changed Day-Over-Day.
- You can also think of this as taking the derivative (rate of change) of the data. This is also helpful when working with time series data and calculating Week-Over-Week.

Syntax – pandas.diff()

```
pandas.DataFrame.diff(self, periods=1, axis=0)
```

- **periods:** To shift for calculating difference, default Value is 1 and accepts negative values.
- **axis:** Take difference over rows (0) or columns (1). If the axis parameter is set to axes='columns.' the function finds the difference column by column instead of row by row.
- There is one core concept you will need to grasp:
    - **Period** = How many observations do you want to difference your data by? Most of the time this will be 1 period diff, but you can select as many as you want.



Find The Difference Between Rows N Periods Away From Each Other

DataFrame.diff(periods=num_periods)

df.diff(periods=1, axis=0)

Index	San Francisco
Mon	50
Tues	65
Wed	72
Thurs	71
Fri	68
Sat	58
Sun	59

Index	San Francisco
Mon	NaN
Tues	15
Wed	7
Thurs	-1
Fri	-3
Sat	-10
Sun	+1

Guided Lab – pandas.diff()

Please complete the **Guided LAB - 343.4.8 - How to calculate the difference of a DataFrame or Series elements between rows/columns**. You can find this lab on Canvas under the Assignment section.

Section 4

Read data from the database into a Pandas Dataframe

Importing data from a MySQL into Pandas dataFrame

- Pandas has a **read_sql()** function that allows you to read data from a SQL DataFrame in Pandas DataFrame. However, we encounter the following issue when attempting to connect to Pandas using a **mysql-connector-python** driver.
  - `UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.`
- Because Pandas does not support directly **mysql-connector-python** models(driver), instead Pandas only support following models.
- **Database String URI**
- **SQLite3 DBAPI2**
- **SQLAlchemy (engine/connection) - Preferred way for connecting to various databases (MySQL, PostgreSQL, Oracle, etc.)**

For more information about SQLAlchemy, please visit the Wiki document.

PER SCHOLAS

How to install Install SQLAlchemy

To Install SQLAlchemy use following command:

- In the terminal it is:
  - pip install sqlalchemy
- In a notebook it will be:
  - !pip install sqlalchemy
- SQLAlchemy is for any relational database, so you will also need a MySQL-specific driver. We will use **pymysql model**:
  - pip install pymysql
- **# install mysql driver**
  - pip install mysql-connector-python
- **Install Additional model for SQL client**
  - pip install mysqlclient

PER SCHOLAS

Guided Lab - Read data from database into a DataFrame

Please complete the **Guided lab - 343.4.9 - Read data from the database into a Dataframe**. You can find this lab on Canvas under the Assignment section.

Summary

- SQLAlchemy creates a bridge between Pandas and mysql-connector-python. It handles database interactions, letting Pandas focus on data manipulation. This approach works for other database types supported by SQLAlchemy.

Section 5

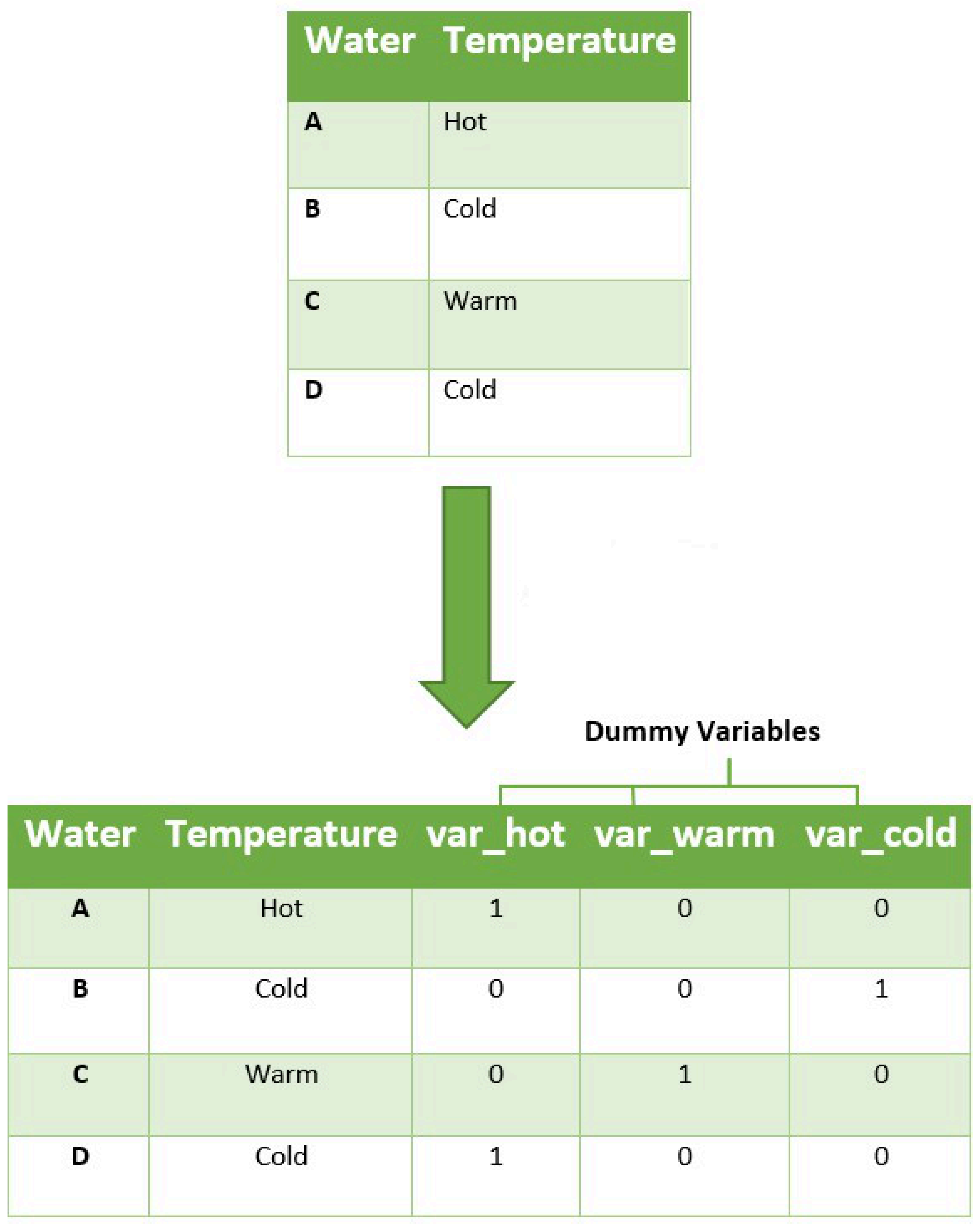
Dummy Variables, pivot() method

PER SCHOLAS

Dummy Variables in Python Pandas

- The **get_dummies()** function is used to convert categorical variables into dummy or indicator variables (binary variable)
- A dummy or indicator variable can have a value of 0 or 1.
- Dealing with Missing Data: If new categories appear in the Data set that were not present in the training dataset, get_dummies() can handle this by adding columns for those categories and filling them with zeros.
- To create a dummy variable in a given DataFrame in pandas, we make use of the **get_dummies()** function.

As you can see three dummy variables are created for the three categorical values of the temperature attribute.


Dummy Variables

Water	Temperature
A	Hot
B	Cold
C	Warm
D	Cold

Water	Temperature	var_hot	var_warm	var_cold
A	Hot	1	0	0
B	Cold	0	0	1
C	Warm	0	1	0
D	Cold	1	0	0

source: geeksforgeeks

Syntax– pandas.get_dummies()

```
pandas.get_dummies(data, prefix=None, dummy_na=False, columns=None, dtype=None)
```

The get_dummies() function takes the following parameter values:

- **data (required):** This is the input data that is used to get the dummy indicators.
- **prefix (optional):** This is a string that is used to append the column names of the DataFrame.
- **dummy_na (optional):** This takes a Boolean value indicating if a column containing NaN is added or not.
- **columns (optional):** This represents the names of the columns for the dummies.
- **dtype (optional):** This is the data type of the resulting columns.

Example 1 – pandas.get_dummies()

In this example, we will demonstrate the use of a handful of components within the get_dummies() function with the following dataframe

```
import numpy as np
import pandas as pd
# Create a DataFrame named Input with two columns: "ID" and "Region"
Input = pd.DataFrame({"ID":[1002, 3201, 4031, 2078, 5897],
           "Region":["Africa","Europe","Asia","Africa", np.nan]})
print(Input)
Region = Input.Region
print(Region)
pd.get_dummies(Region)
Res = pd.get_dummies(Region, prefix='option', prefix_sep="-", dummy_na=True)
print(Res)
```

PER SCHOLAS

Example 2 – pandas.get_dummies()

The objective of this program is to demonstrate the usage of the pd.get_dummies() function in the pandas library to convert categorical variables into dummy/indicator variables

```
import pandas as pd
data = pd.DataFrame({"Water":["A","B","C","D"],
"Temperature":["Hot","Cold","Warm","Normal"],
"region":["East","North","East","South"]
              })
print(data)

resultOne= pd.get_dummies(data)
(resultOne)
# working with single column
resulttwo = pd.get_dummies(data, columns = ['Temperature'])
print(resulttwo)
# Using the prefix parameter
resultthree = pd.get_dummies(data, columns = ['Temperature'], prefix="region")
print(resultthree)
```

PER SCHOLAS

What is the pivot() function in pandas?

- The **pivot()** function in pandas is used to reshape or pivot a DataFrame by rearranging the rows and columns. It allows you to transform long-format data into a wide-format form, making it easier to analyze and visualize certain types of data. The pivot method is particularly useful for creating pivot tables and performing cross-tabulations.
- If there are multiple rows for a unique combination of index and columns, you may need to use the **pivot_table()** function , which allows you to specify an aggregation function to handle duplicate values.

Key Points:

- **No Aggregation:** It does not aggregate duplicate values. It simply rearranges them into a new structure.
- **Single-Level Indices:** It only handles single-level indices and columns. For multi-level structures, consider pivot_table.
- **Basic Reshaping:** It is often used for initial reshaping, while the pivot_table function offers more flexibility for complex pivoting and aggregation
- **Alternative Methods:** Remember to consider **pivot_table()** function for more complex aggregations and multi-level structures when needed.

Syntax: pivot() function

The pivot function has the following syntax:

```
pandas.pivot(data, index=None, columns=None, values=None)
```

- index (str or object): This is the column whose unique values will become the new DataFrame's index.
- columns (str or object): This is the column whose unique values will become the new DataFrame's columns.
- values (str or object): This is the column whose values will populate the new DataFrame.
- Multiple Values: When pivoting without specifying values, it includes all non-index/column values.

Note: Return value - The pivot function returns a reshaped data frame according to the provided columns.

PER SCHOLAS

Real-world scenarios for pivot() function

Here are real-world scenarios where the pivot method shines:

- **Sales Analysis:**
  - Data: Store sales with columns for date, product, quantity sold, and revenue.
  - Pivot: Reshape to analyze sales trends by product across dates or compare revenue for different products on a specific date.
- **Customer Behavior:**
  - Data: Customer interactions with columns for customer ID, interaction type (e.g., purchase, visit), timestamp, and value.
  - Pivot: See how many times each customer engaged in different interaction types or track their spending patterns over time.
- **Financial Portfolio Analysis:**
  - Data: Portfolio holdings with columns for asset class, ticker symbol, quantity, and price.
  - Pivot: Calculate the total value of each asset class or visualize the allocation of different asset classes over time.

Guided Lab - pivot() function

Please complete the **Guided LAB - 343.4.10 - How to use pivot() function.** You can find this lab on Canvas under the Assignment section.

PER SCHOLAS

Hands-on Lab

The aim of this lab is to provide you a refresher on the **Pandas** library, so that you can pre-process and analyze the dataset before applying data visualization techniques on it.

Please complete the Guided LAB - 343.4.11: Exploring and Pre-processing a Dataset using Pandas. You can find this lab on Canvas under the Assignment section.

PER SCHOLAS

The content referenced in the following slides will NOT be covered in the scope of this course; however, Per Scholas recommends that you explore those tools and content for upskilling.

Introduction to pivot_table() function

- An important part of data analysis is the process of grouping, summarizing, aggregating and calculating statistics about this data. Pandas pivot tables provide a powerful tool to perform these analysis techniques with Python.

- If you are a spreadsheet user then you may already be familiar with the concept of pivot tables. Pandas pivot tables work in a very similar way to those found in spreadsheet tools such as Microsoft Excel.
  - If you are not familiar with the Pivot Table, wikipedia explains it in high level terms.
- The **pandas.pivot_table()** function takes in a dataframe and the parameters detailing the shape you want the data to take. Then it outputs summarized data in the form of a pivot table.

pivot() function vs. pivot_table() function

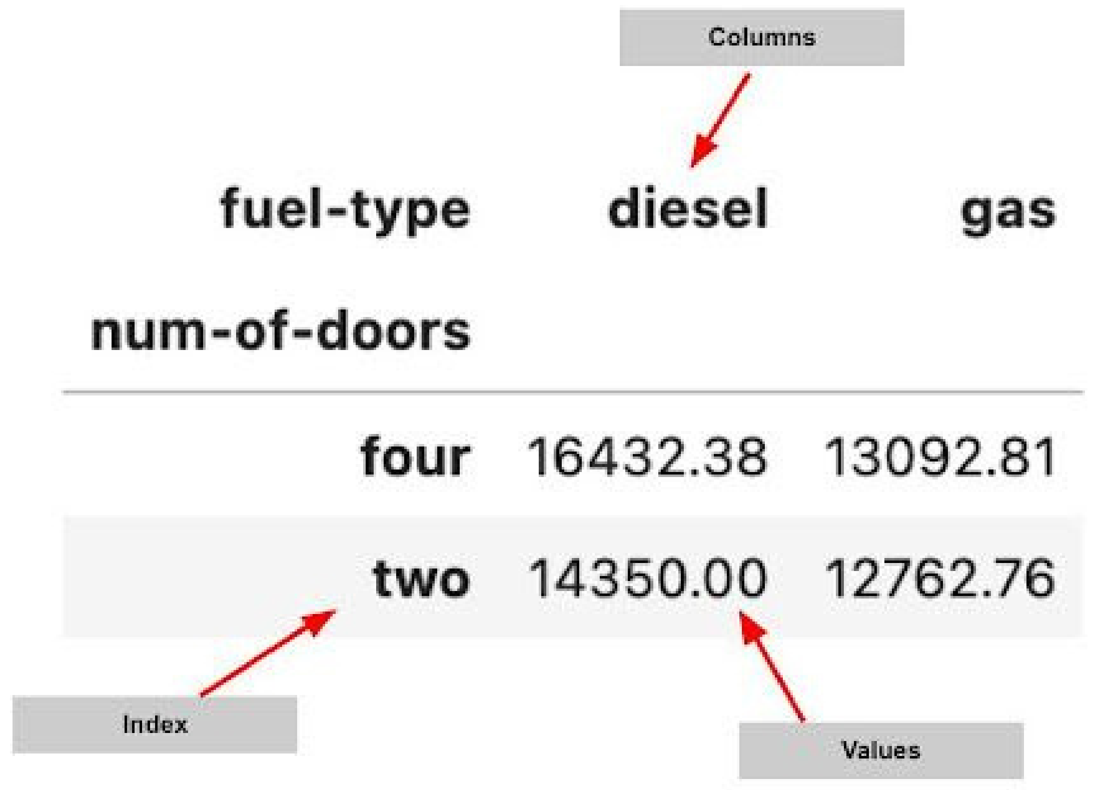
Understanding the Key Differences:

- If there are multiple rows for a unique combination of index and columns, We can not use pivot() function, instead you may need to use the **pivot_table()** function, which allows you to specify an aggregation function to handle duplicate values.
- The **pivot()** function is a specialized form of the more general **pivot_table()** function. **pivot_table()** function provides additional flexibility and functionality for handling duplicate values and aggregations.
- **Aggregation:** **pivot_table()** function aggregates duplicate values using functions like sum, mean, etc., while **pivot()** function simply rearranges them into the new structure.
- **Flexibility:** **pivot_table() function** handles multi-level indices and columns, while **pivot()** function is limited to single-level structures.

How to Create a Pandas Pivot Table

A pandas pivot table has three main elements:

- Index: This specifies the row-level grouping.
- Column: This specifies the column level grouping.
- Values: These are the numerical values you are looking to summarize.


Columns
Index
Values

fuel-type	diesel	gas
num-of-doors		
four	16432.38	13092.81
two	14350.00	12762.76

Pivot tables in Pandas allow users to examine subsets of data depending on indexes and values. Values are organized by index and provided to the user. The syntax for the Pandas.pivot_table() function is as follows:

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

Lab - Reshape and analyze datasets using pivot_table()

Click here for lab: Guided lab - Pivot Tables in Pandas - Reshape and analyze datasets using pivot_table() with various configurations.

Knowledge Check

- How do you aggregate multiple columns of a dataframe together?
- What is the purpose of the groupby method of a dataframe? Illustrate with an example.
- What are the different ways in which you can aggregate the groups created by groupby?
- What do you mean by a running or cumulative sum?
- How do you create a new column containing the running or cumulative sum of another column?
- What are other cumulative measures supported by Pandas dataframes?

References

- https://colab.research.google.com/github/GoogleCloudPlatform/cloud-sql-python-connector/blob/main/samples/notebooks/mysql_python_connector.ipynb#scrollTo=7Pb7xJmIWOwQ
- https://medium.com/analytics-vidhya/importing-data-from-a-mysql-database-into-pandas-data-frame-a06e392d27d7
- https://www.plus2net.com/python/mysql-sqlalchemy.php

PER SCHOLAS

Questions?

PER SCHOLAS

UNLOCKING POTENTIAL

CHANGING THE FACE OF TECH