Applied Data Science with Python

Working With Pandas

Learning Objectives

By the end of this lesson, you will be able to:

- Work with the two primary data structures in Pandas: Series and DataFrame
- Explore the functions used on Series, DataFrame, and Panel
- Apply statistical functions in Pandas
- Utilize some commonly used functions in Pandas

Business Scenario

XYZ Pharmacy offers over-the-counter (OTC) medicines to its customers. The pharmacy maintains a dataset containing information about various OTC medicines, including their use cases and recommended dosages.

They utilize the Pandas library to manage and analyze their over-the-counter (OTC) medicines dataset, enabling them to provide better recommendations to customers and optimize inventory management.

They iterate through the dataset, sort data based on specific criteria, and perform text processing tasks for data standardization. Next, they categorize data for easier analysis and handle various file formats for seamless data import and export. This comprehensive data analysis helps XYZ Pharmacy to improve its overall customer service and business operations.

Introduction to Pandas

Discussion: Pandas

Duration: 10 minutes

- What is Pandas?
- What are the different types of data analysis functions available in Pandas?

Pandas

Pandas is a simple and easy-to-use open-source tool for data analysis.

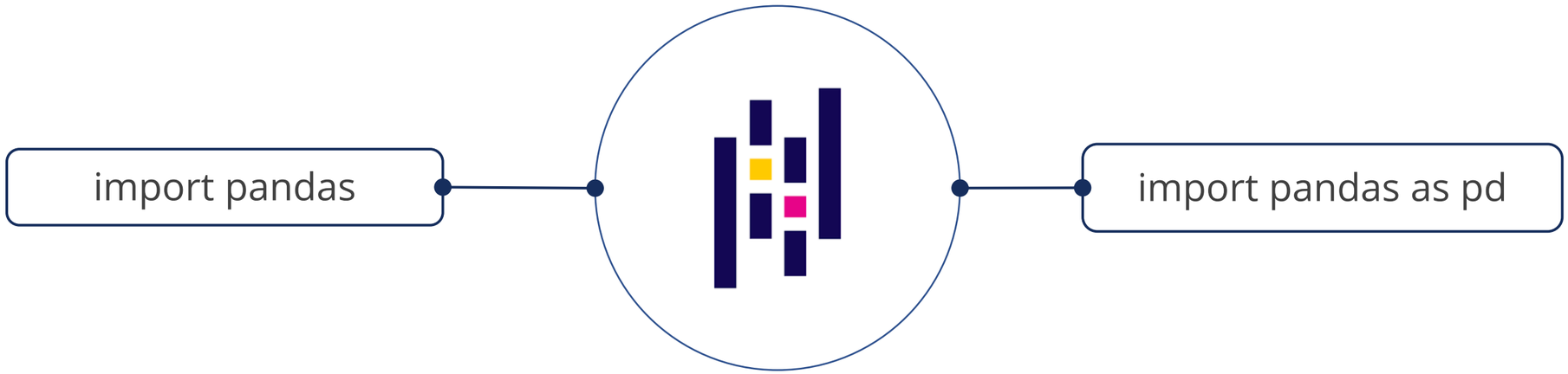
It is released under a BSD license. The BSD (Berkeley Software Distribution) license grants users the freedom to use, modify, and distribute software with minimal restrictions and disclaimers of warranty.

Data scientists use it as their framework of choice.

Pandas

To use Pandas in a Python project, import the Pandas library using the **import** statement.

It can be done in two ways:



pd is an alias and is preferred by developers.

Pandas

Pandas are used to handle two major types of data structures:

Series	It is a one-dimensional ndarray with axis labels.
DataFrame	It is a two-dimensional heterogeneous tabular data with data as rows and columns.

Pandas DataFrame

Some of the functions commonly performed on a DataFrame are:

Functions	Description	Syntax
head()	Returns the first few rows of the DataFrame	df.head(n)
tail()	Returns the last few rows of the DataFrame	df.tail(n)
info()	Provides a summary of the DataFrame's structure, including column names, data types, and non-null values	df.info()
describe()	Generates descriptive statistics of the DataFrame, such as count, mean, standard deviation, minimum, maximum, and quartiles	df.describe()
shape	Returns the dimensions (number of rows and columns) of the DataFrame	df.shape
columns	Returns the column labels of the DataFrame	df.columns
loc[] or iloc[]	Allows for indexing and slicing of rows and columns based on labels (loc[]) or integer-based positions (iloc[])	df.loc[row_label, column_label]

Pandas DataFrame

Some of the functions commonly performed on a DataFrame are:

Functions	Description	Syntax
sort_values()	Sorts the DataFrame based on one or more columns	df.sort_values(by, axis=0, ascending=True, inplace=False)
groupby()	Groups the DataFrame based on one or more columns and allows for aggregating functions such as sum(), mean(), or custom aggregations	df.groupby(by, axis=0, sort=True)
apply()	Applies a function to each element, row, or column of the DataFrame	df.apply(func, axis=0)
merge() or concat()	Combines multiple DataFrames based on common columns (merge()) or concatenates DataFrames along a particular axis (concat())	pd.merge(left, right, on=None, how='inner')
plot()	Creates various types of plots and visualizations based on the DataFrame's data using libraries like Matplotlib or Seaborn	df.plot(x=None, y=None, kind='line', ax=None, ...)
drop()	Removes specified rows or columns from the DataFrame	df.drop(index_label, axis=0, inplace=False)

Assisted Practices

Let’s understand the topics below using Jupyter Notebooks.

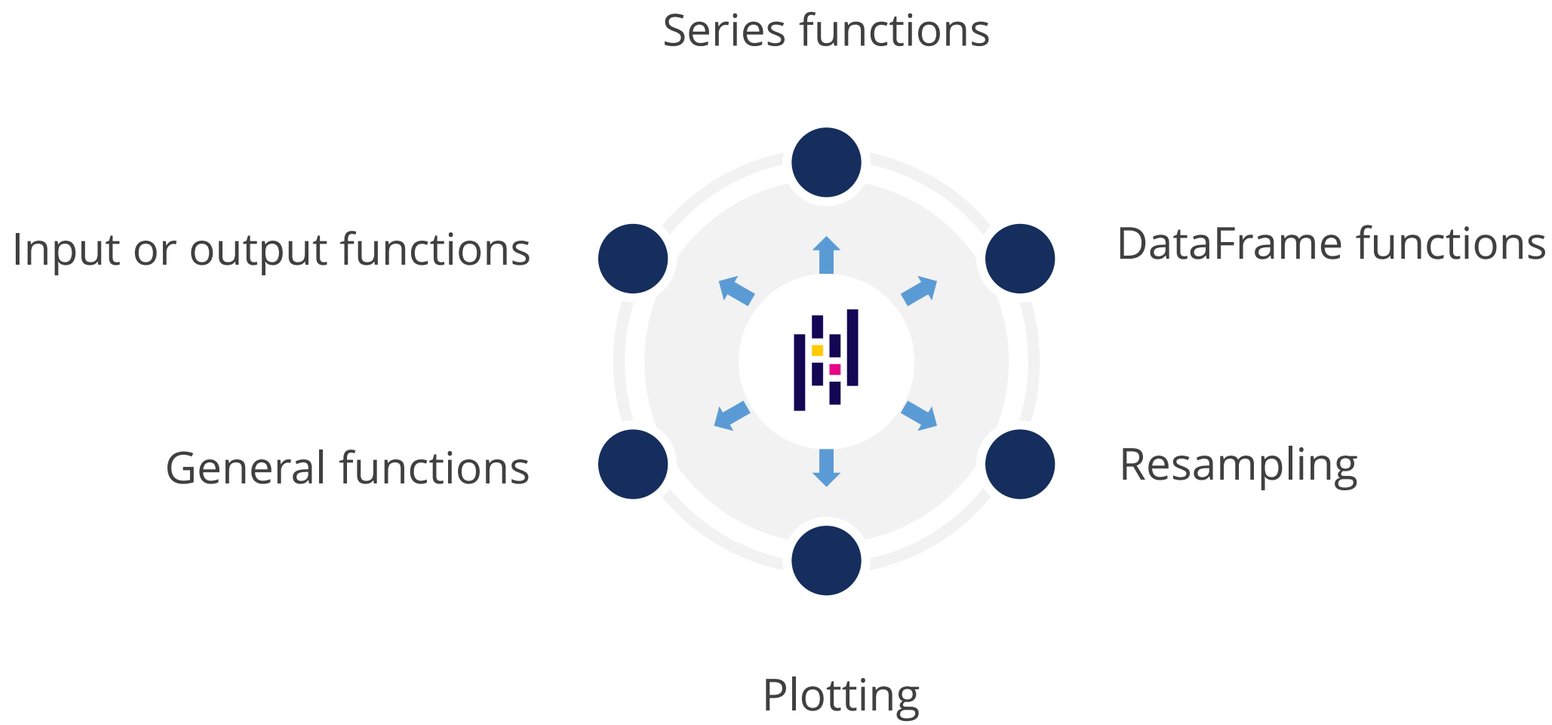
- 9.3_Introduction to Pandas Series
- 9.4_Querying a Series
- 9.5_Pandas DataFrame
- 9.6_Introduction to Pandas Panel

Note: Please download the pdf files for each topics mentioned above from the Reference Material section.

Common Functions in Pandas

Functions in Pandas

Pandas data analysis functions can be categorized as:



Input or Output Functions

These functions give the ability to read from or write into various data sources.

Examples:

`read_excel(io[, sheet_name, header, names, ...])` → This function reads an Excel spreadsheet into a DataFrame object.

`DataFrame.to_excel(excel_writer[, ...])` → This DataFrame method writes into an Excel spreadsheet.

Input or Output Functions

The following methods read JSON data into a DataFrame or export DataFrame into a JSON file.

read_json([path_or_buf, orient, typ, dtype, ...])	Reads the JSON data into a Pandas object
to_json(path_or_buf, obj[, orient, ...])	Writes the Pandas data into a JSON object

IO Tools

Pandas library provides a number of read and write functions, each packed with a set of utility methods to tackle different file formats.

IO Tools

Here is a comprehensive list of file formats that Pandas can handle:

CSV	XLS	XLSX	JSON	HTML
SQL	Parquet	Feather	HDF5	Stata
SAS	SQL Query Results	Clipboard	Fixed-width text files	MATLAB files
Pickle files	Msgpack	GBQ	Google Sheets	Zip archives

IO Tools

Pandas provides specific read and write functions for every file format.

read_html(): Reads an HTML file

html(): Writes an HTML file

The name of the file must be passed as a string argument to the read and write functions.

Comma Separated Values (CSV) Format

CSV is the most popular textual format in which Pandas can perform IO operations.

read_csv()
function is used
to read a CSV file.

to_csv()function is
used to write into
a CSV file.

Comma Separated Values (CSV) Format

Example of a CSV file and how it would look when loaded into a Pandas DataFrame

Text file (example.csv):

Item 1	Item 2	Item 3	Item 4
Value 1	Value 2	Value 3	Value 4
Value 5	Value 6	Value 7	Value 8

Python code to load the CSV into a Pandas DataFrame:

```
import pandas as pd

# Read the CSV file into a
DataFrame
df = pd.read_csv('example.csv')

# Display the DataFrame in table
format
print(df.to_string(index=False))
```

Comma Separated Values (CSV) Format

Output of the above code is:

	Item 1	Item 2	Item 3	Item 4
0	Value 1	Value 2	Value 3	Value 4
1	Value 5	Value 6	Value 7	Value 8

- In this example, the CSV file has four columns (item1, item2, item3, item4).
- The first row contains the column headers, and the subsequent rows contain the data values.
- When loaded into a Pandas DataFrame using the read_csv() function, the CSV data is displayed as a table-like structure, where each column represents a variable, and each row represents a record or observation.

Comma Separated Values (CSV) Format

Functions are packed with rich parameters.

Parameters of the read() function can be categorized as:

- Column, index locations, and names
- General parsing configurations
- NA and missing data handling
- Specifying column data types

Comma Separated Values (CSV) Format

Consider the following examples:

df = pd.read_csv('data.csv')	Reads data from a file named data.csv
df.to_csv('output.csv')	Writes Pandas data into the output.csv file

df can be a series or DataFrame.

JavaScript Object Notation (JSON) Format

It handles the ReST-based web services' requests and responses.

df_json = pd.read_json('data.json')	Reads data from a file named data.json
df_json.to_json('output.json')	Writes a Pandas series, tabulated data, or multi-dimensional data into output.json file

Extensible Markup Language (XML) Format

It is a textual file format where data is marked up with a number of tags.

df_xml = pd.read_xml('data.xml')	Reads data from a file named data.xml
df_xml.to_xml('output.xml')	Writes a Pandas series, tabulated data, or multi-dimensional data into output.xml file

Classification of General Functions

Operations performed using general functions are classified under:

Data manipulation

Top-level missing data

Top-level conversion

General Functions: Examples

pivot(data[, index, columns, values])

Returns a reshaped DataFrame organized by index or column values

isna(obj)

Detects missing values from an array-like object

to_numeric(arg[, errors, downcast])

Converts the argument to numeric

Series Attributes and Functions

copy([deep])

Makes a copy of an object's indices and data

iloc()

Provides integer-location-based indexing for selection by position

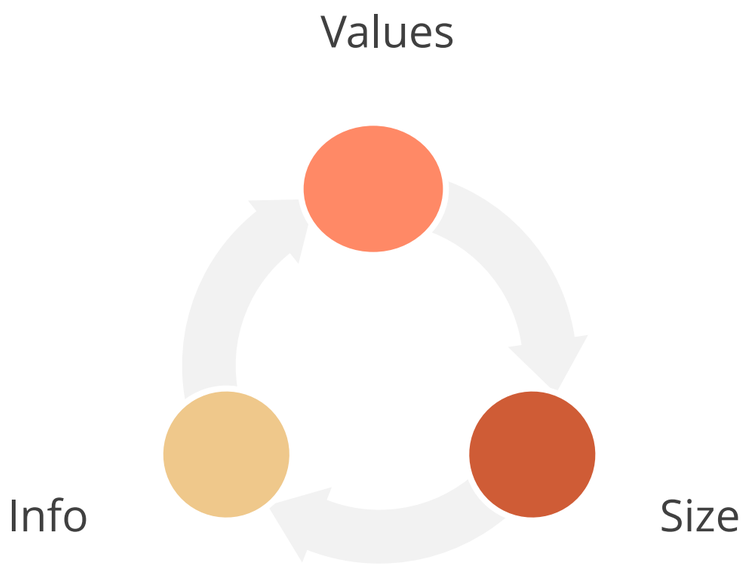
add(other[, level, fill_value, axis])

Returns addition of series and other element-wise addition (Similarly, **sub** and **mul** can be used for subtraction and multiplication)

DataFrame Functions

The DataFrame functions provide attributes and methods for tabulated 2D data operations.

Some of the main attributes are:



Other Important Methods in Pandas

Some important methods are:

Replaces values where the condition is false

Iterates over DataFrame rows as (index, series) pairs

Prints the top **n** rows

Other Important Methods in Pandas

Some other methods that are available are:

Conversion

Syntax:

result = object.method_name(parameters)

Indexing

Syntax:

result = object[indexing_expression]

Iterating

Syntax:

for index, row in df.iterrows():

Binary operations

Syntax:

result = object1 binary_operator object2

Discussion: Pandas

Duration: 10 minutes

- What is Pandas?

Answer: Pandas is an open-source tool for data analysis, renowned for its user-friendly nature and simplicity. It is licensed under Berkeley Software Distribution, which permits users to freely use, modify, and distribute the software with minimal restrictions and warranty disclaimers.

- What are the different types of data analysis functions available in Pandas?

Answer: Pandas offers several types of data analysis functions, including series functions, input or output functions, general functions, plotting functions, DataFrame functions, and resampling functions.

Statistical Functions in Pandas

Discussion: Statistical Functions in Pandas

Duration: 10 minutes

- What is the difference between mean() and median() functions in Pandas?
- What is categorical data?

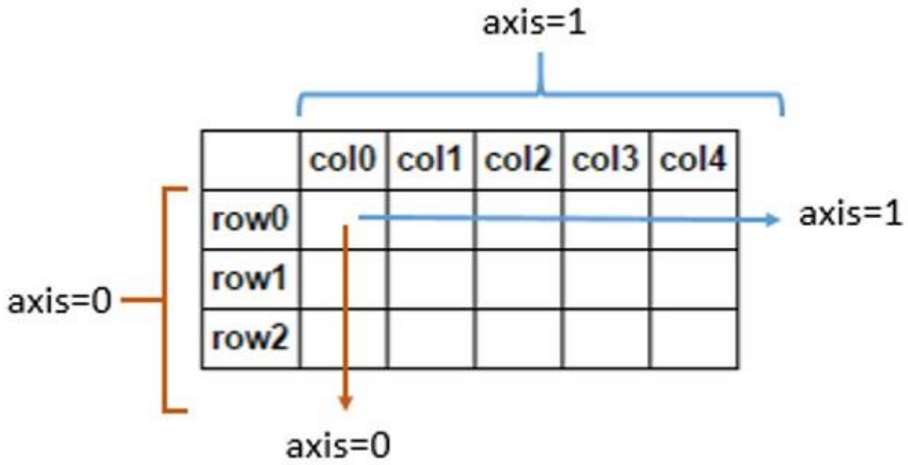
Statistical Functions in Pandas

In the following functions, the skipna parameter is True by default, excluding NA or null values from the result:

- Mean
- Median
- Mode
- Sum
- Std
- Skew functions

Statistical Functions in Pandas

In pandas, the terms axis=0 and axis=1 refer to the two different axes of a DataFrame.



- In pandas, axis=0 refers to the row axis, representing horizontal rows in the DataFrame.
- Operations along axis=0 are performed row-wise, independently on each row.
- axis=1 refers to the column axis, representing vertical columns in the DataFrame.
- Operations along axis=1 are performed column-wise, independently on each column.

Statistical Functions in Pandas

The mean() function returns the mean of the values over the requested axis.	mean(axis=None, skipna=None, level=None, numeric_only=None, **kwargs)
The median() function returns the median of the values over the requested axis.	median(axis=None, skipna=None, level=None, numeric_only=None, **kwargs)
The mode() function retrieves the mode(s) of each element along the selected axis.	mode(axis=0, numeric_only=False, dropna=True)
The sum() function computes the sum of the values over the requested axis.	sum(axis=None, skipna=None, level=None, numeric_only=None, min_count=0, **kwargs)

Statistical Functions in Pandas

std() returns the standard deviation over the requested axis.	std(axis=None, skipna=None, level=None, ddof=1, numeric_only=None, **kwargs)
cov() computes the pairwise covariance among the series, excluding the NA or null values.	cov(min_periods=None, ddof=1)
corr() calculates the pairwise correlation of columns, excluding null or NA values.	corr(method='pearson', min_periods=1)

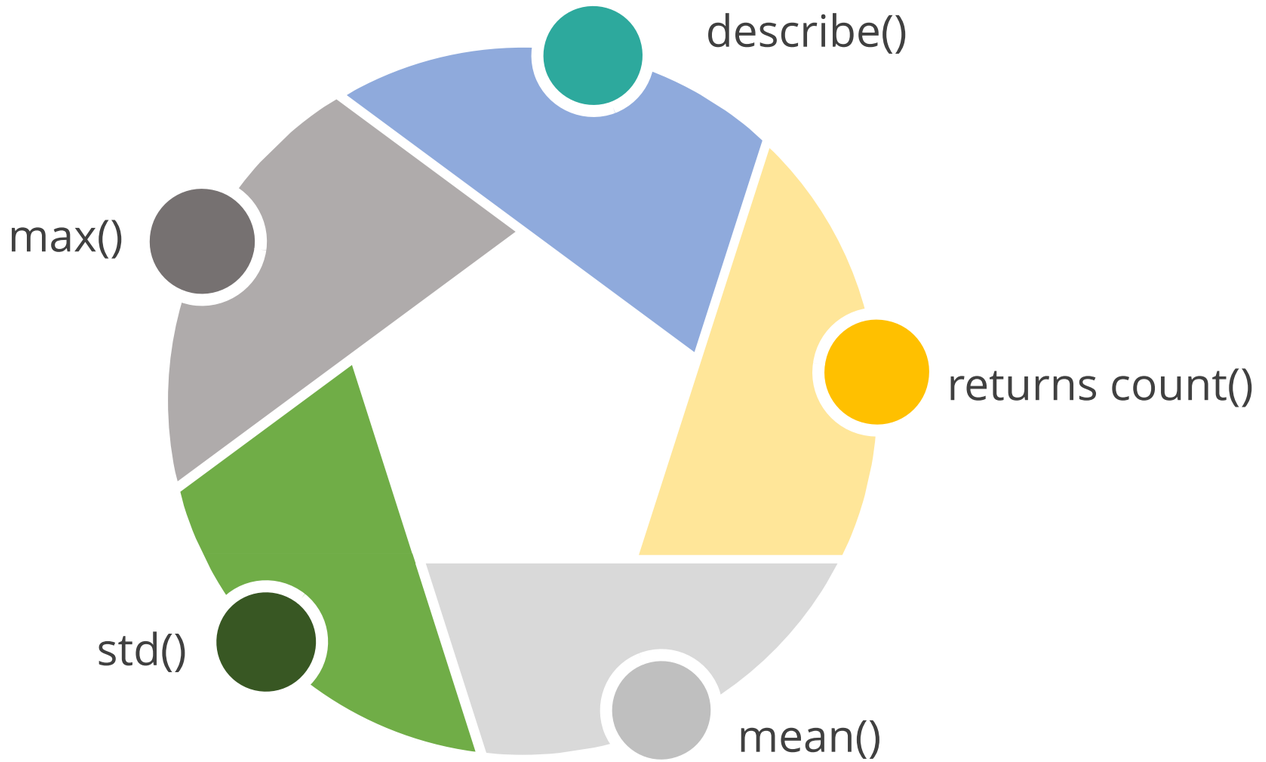
Pearson is the default correlation method, while Kendall or Spearman are also available.

Statistical Functions in Pandas

skew() returns an unbiased skew over the requested axis.	skew(axis=None, skipna=None, level=None, numeric_only=None, **kwargs)
kurt() returns unbiased kurtosis over the requested axis.	kurt(axis=None, skipna=None, level=None, numeric_only=None, **kwargs),
kurtosis() also returns unbiased kurtosis over the requested axis.	kurtosis(array, axis=0, fisher=True, bias=True)

Statistical Functions in Pandas

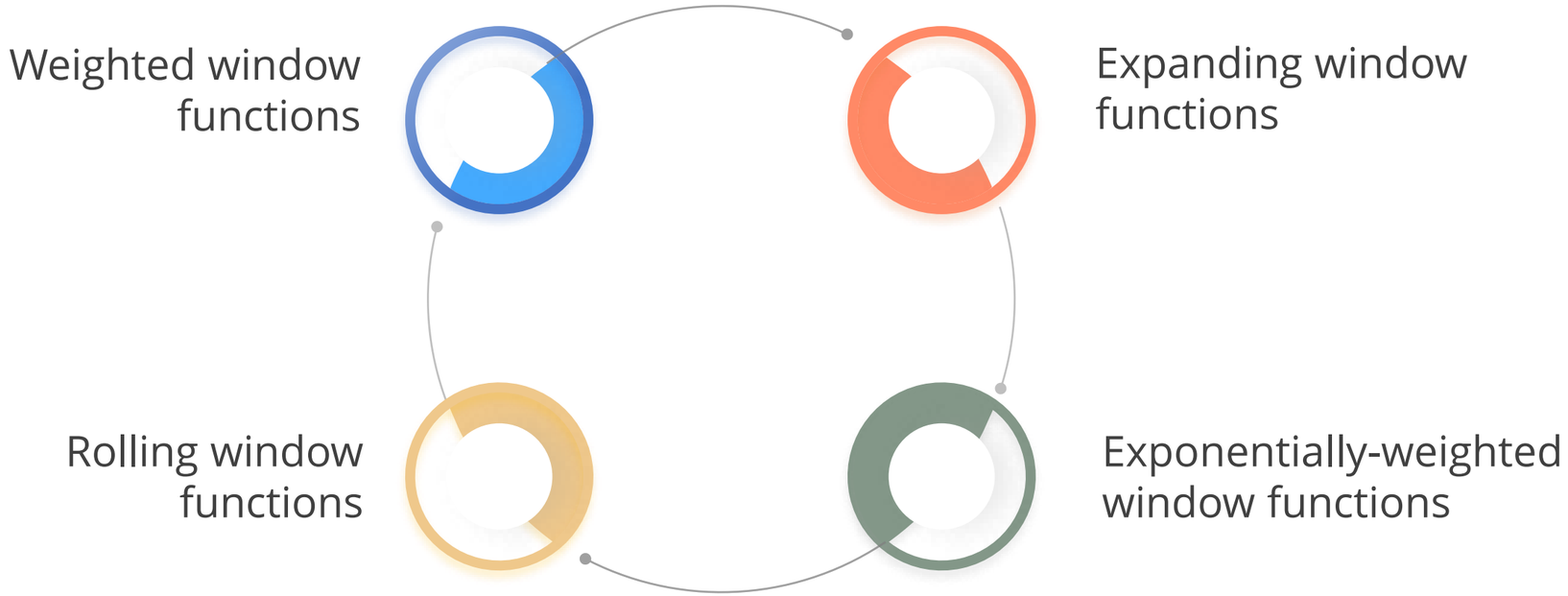
The following methods give a statistical snapshot of the DataFrame, excluding the NA or null values:



The functions std(), mean(), and max() can be applied to the whole DataFrame instead of a single column in Pandas.

Statistical Functions in Pandas

Pandas window functions include count(), mean(), median(), etc. under the following categories:



Date and Timedelta

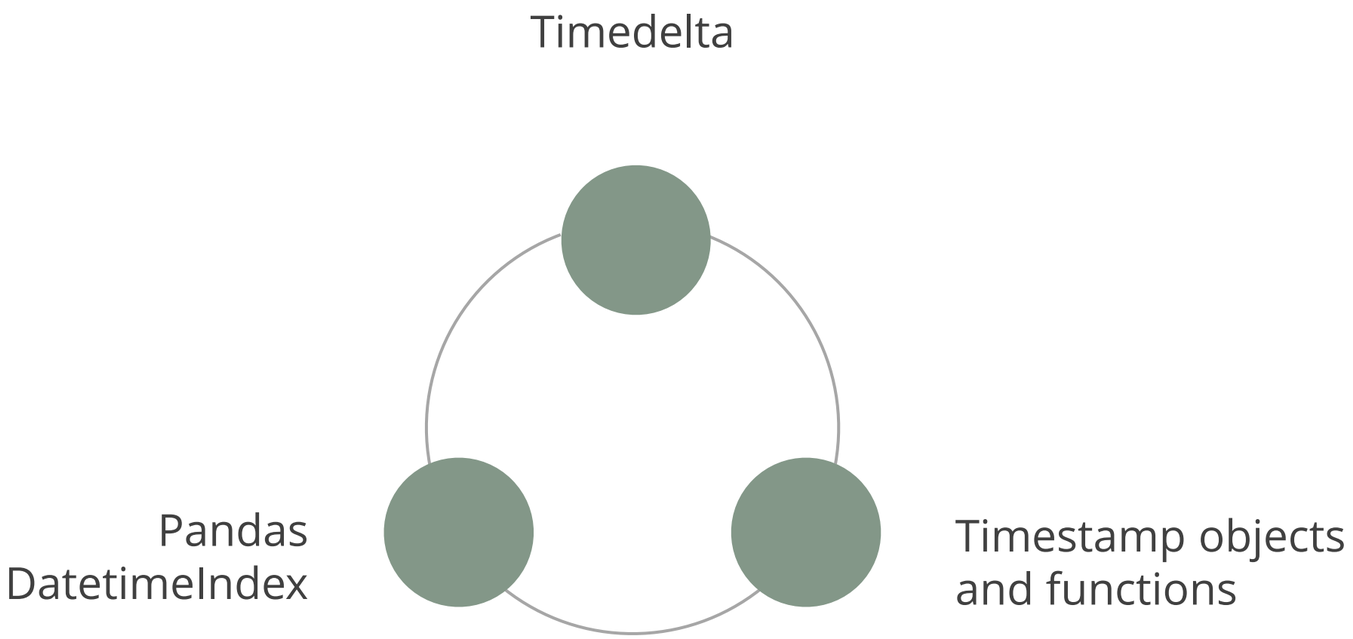
Date and Timedelta

Timeseries is a series of data points indexed in date or time order.

They occur in successive, equally-spaced date or time intervals.

Date and Timedelta

The following list of Pandas functions categories is equivalent to Python’s datetime and timedelta.



Date and Timedelta

The DatetimeIndex class is crucial in Pandas for time series data handling, offering a wide range of methods and functionalities, making it one of the most important classes for working with datetime data.

class pandas.DatetimeIndex(*data=None*, *freq=_NoDefault.no_default*, *tz=_NoDefault.no_default*, *normalize=False*, *closed=None*, *ambiguous='raise'*, *dayfirst=False*, *yearfirst=False*, *dtype=None*, *copy=False*, *name=None*)

The pandas.DatetimeIndex class is used to represent an index of dates or times in Pandas. It provides a powerful data structure for efficiently indexing and manipulating time series data.

Attributes

The DatetimeIndex object has many attributes and functions.

Day or date related:

day, date, month

Time related:

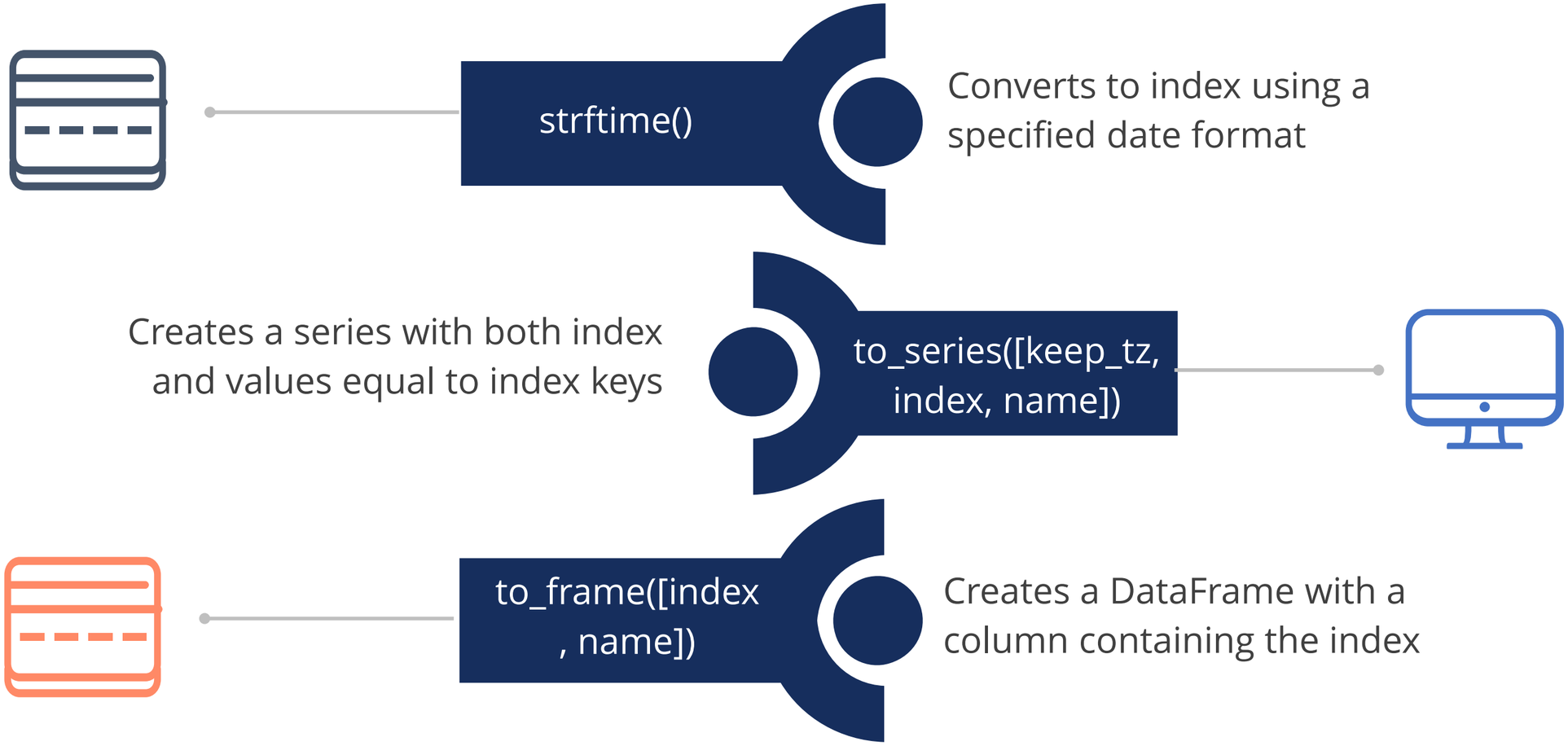
hour, minute, microsecond

Utility related:

timetz, dayofweek, is_month_end

Functions

Some important methods include:



Timedelta

This object represents a duration, the difference between two date or time values.

```
Timedelta(value=<object object>,
unit=None, **kwargs)
```

The output could be in a day(s) or date.

Timedelta

For computing time-related timedelta, the value can be:

```
datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0,
minutes=0, hours=0, weeks=0)
```

Attributes of Timedelta

Some attributes of timedelta are:

Days	Number of days
Seconds	Number of seconds
Microseconds	Number of microseconds

Functions of Timedelta

Some timedelta functions are:

isoformat( )	Formats a Timedelta object into ISO 8601 format
to_pytimedelta( )	Converts a Pandas Timedelta object into a Python timedelta object

Categorical Data

Categorical Data

Categorical data refers to qualitative data that represent distinct categories or groups rather than numerical values.

Data classification and categorization tasks often include categorical data; data that can be divided into different groups. For example:

- Blood group
- Gender
- Age group
- Customer rating

Pandas' library provides *category* data types that can be used for categorical data.

Categorical Data

To set the data type as categorical, the dtype must be specified as a category.

The following can be set as a categorical data type:

Pandas' series

Tabulated data

DataFrame

Note: Categorical data is not ordered; it consists of distinct categories that are inferred directly from the dataset. These categories have no inherent ranking or sequence associated with them.

Categorical Data

Consider a bloodType Pandas series:

```
bloodType = pd.Series(["a+","b-","ab+","ab-", "o+", "o-", "a+", "a-", "ab-", "ab+",
"o-", "o+", "b-", "b+", "a+", "a+"], dtype="category")
```

The series lists 15 data points with eight categories.

0	a+
1	b-
2	ab+
3	ab-
4	o+
5	o-
6	a+
7	a-

8	ab-
9	ab+
10	o-
11	o+
12	b-
13	b+
14	a+
15	a+

dtype: category

Categories: a+, a-, ab+, ab-, b+, b-, o+, o-

The type in the output has been marked as category.

Categorical Data

Categorical data can also be created using the categorical class.

This allows indicating the order of the data.

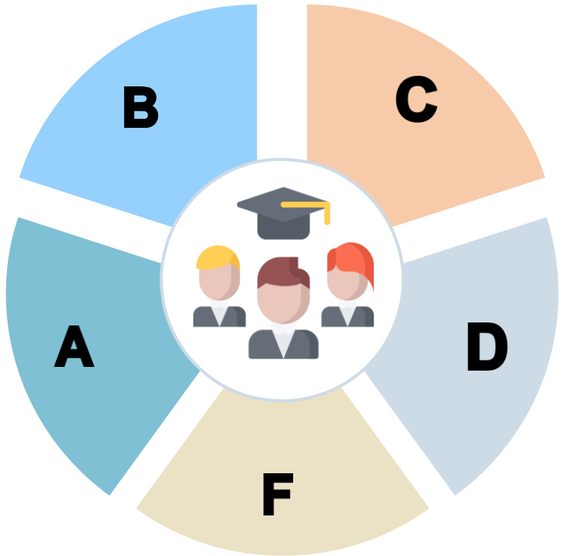
Categorical Data: Example

Consider a list of performance grades, A, B, C, D, and F awarded to ten students.

The performance of every student falls into one of the five categories shown below:

Very good performance

Good performance



Excellent performance

Average performance

Failing grade

The order of grades should be A > B > C > D > F.

Ordinal Categorical Data: Example

Create a categorical variable for the data set and name its grades using the categorical class:

```
grades = pd.Categorical(['A', 'A', 'C', 'F', 'B', 'D', 'B', 'C', 'F', 'D' ],
ordered=True)
```

The reorder_categories function can be used to reorder the categories.

```
grades = grades.reorder_categories(['F', 'D', 'C', 'B', 'A'])
```

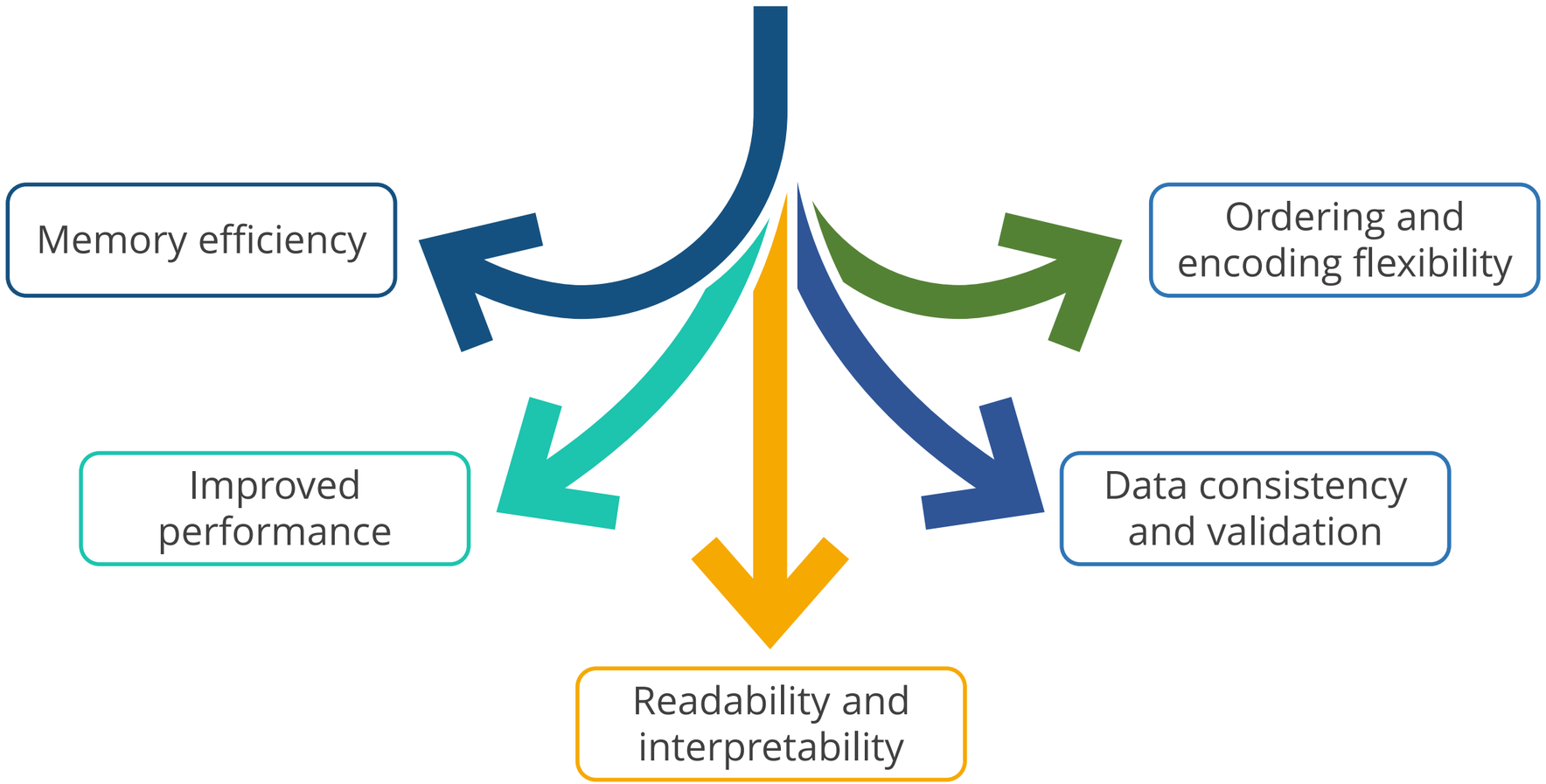
The parameter ordered, when set to true, creates an ordered categorical variable.

Categorical Data

Pandas support many operations to be performed on categorical data, including:

Categorical Data: Uses

The following are benefits of using categorical data:



Discussion: Statistical Functions in Pandas

Duration: 15 minutes

- What is the difference between the mean() and median() functions in Pandas?

Answer: The mean() function calculates the average of values along a specified axis, giving a measure of central tendency. On the other hand, the median() function finds the middle value in a sorted dataset, representing the central value that separates the data into two equal halves.

- What is categorical data?

Answer: Categorical data refers to qualitative information that groups observations into distinct categories, rather than representing them as numerical values.

Working with Text Data

Discussion: Statistical Functions in Pandas

Duration: 10 minutes

- What are some popular methods for text processing?
- How to iterate over the rows of a Pandas DataFrame?

Working with Text Data

Text data in Pandas is nothing but string data.

Consider the following processed textual data:

```
text = pd.Series (['A', 'B', 'C', 'D', 'E', 'F'])
```

The data is read as the object type.

Text or string data in Pandas is called an object. It's assigned by default when the text data is imported.

Working with Text Data

To be processed as a regular string, the data needs to be explicitly input as dtype='string' and dtype=pd.StringDtype().

```
text = pd.Series (['A', 'B', 'C', 'D', 'E', 'F'], dtype = 'string')
```

The following methods can then be applied to the text:

- str.lower()
- str.upper()
- str.len()

It is possible to apply string functions to data, even if the data type is an object.

Popular String Operations Performed

Some of the popular text processing methods are:

str.upper()

Converts the textual data to uppercase

str.lower()

Converts the textual data to lowercase

str.len()

Reports the length of the string of the data

Working with Text Data

Example: Create a text series and convert it to upper case.

```
text = pd.Series(['Lorem',
'ipsum', 'dolor', 'sit', 'amet',
'consectetur', 'adipiscing',
'elit']
, dtype='string')

Invoke str.upper() method and see
the input:
text.str.upper()
```

```
# Output:
0           LOREM
1           IPSUM
2           DOLOR
3             SIT
4            AMET
5     CONSECTETUR
6      ADIPISCING
7            ELIT
dtype: string
```

Working with Text Data

The split() method splits the string based on the character or string supplied as a parameter.

Once the method is applied, the resulting object is a list.

This list can be processed or accessed using the get() or get[] syntax, which allows retrieval of specific elements from the list based on their position or index.

Splitting is one of the common string processing methods in Pandas.

Working with Text Data

Consider an example where a string is split by spaces

```
mySer = pd.Series(["this is a sentence",
'sentence num two'])
result = mySer.str.split()
```

The code creates a pandas Series with two string values and then uses the str.split() method to split each string element into a list of words.

The resulting lists are stored in the result variable as a new Series.

Working with Text Data

The replace() method is used to replace specific values in a string or a pandas Series DataFrame with another value.

The find and replace parameters determine what will be processed in a particular string.

Replace() parameters should be provided using pattern-matching strategies.

Working with Text Data

The cat() method in pandas is used to concatenate or combine categorical data.

Use the previous example and invoke the cat() method with space as a separator

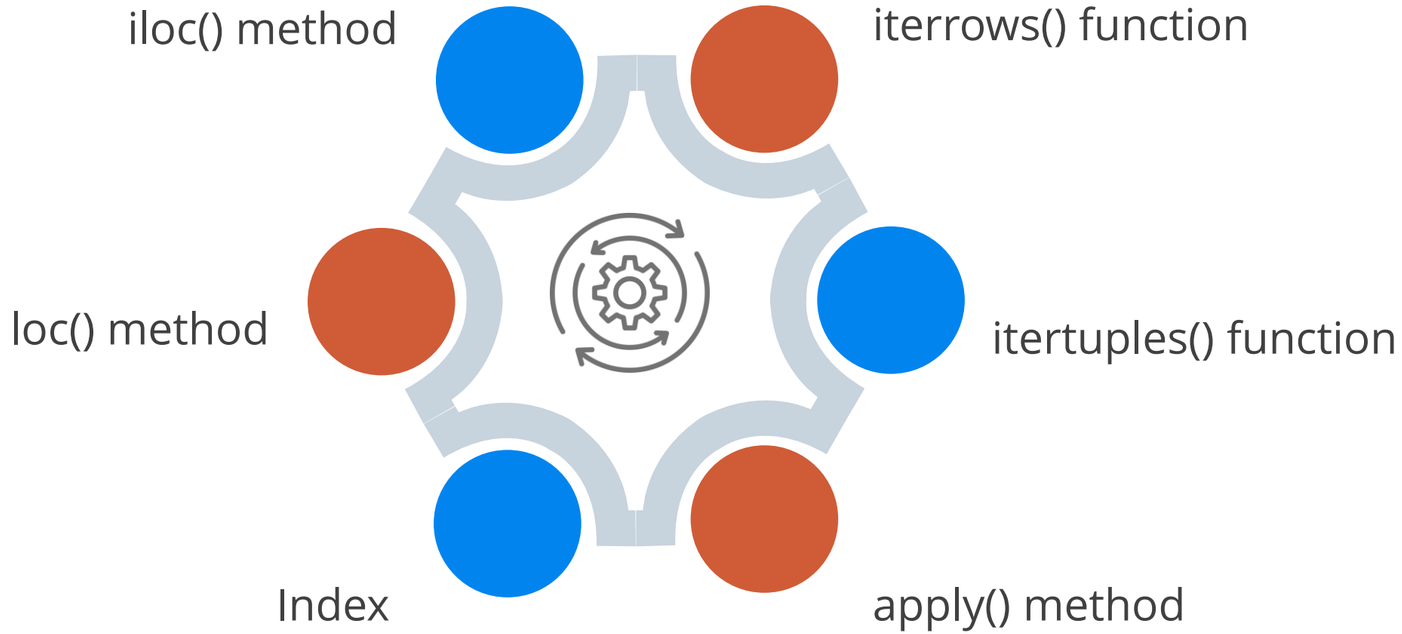
```
text.str.cat(sep=' ')
# Output:
'Lorem ipsum dolor sit amet
consectetur adipiscing elit'
```

Iteration

Iteration

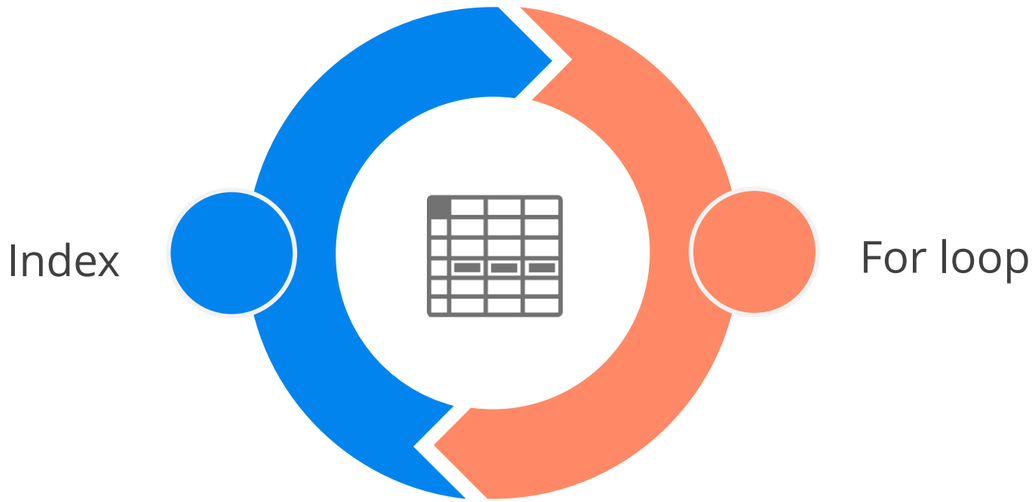
Pandas provide many ways of iterating two-dimensional and multi-dimensional data.

Some of the iteration methods used in Pandas are:



DataFrame Using Index

Rows of Pandas are iterated using:



If the DataFrame has a default index, that is, starting with 0, then iteration is done using numbers starting with 0.

Iteration

It is the process of iterating over the rows or columns of a DataFrame or series object, using a loop or a function.

Consider the DataFrame given below:

Problem	Medicine	Dosage
Headache	Anacin	One Time
Body Ache	Ibuprofen	3 times a day
Fever	Dolo 650	3 times a day, for 3 days
Cough	Strepsils	3 times a day, for 2 days
Sore Throat	Tylenol	3 times a day
Allergies	Benadryl	1 time in the night, for 2 days
Indigestion	Lopermide	2 times a day

DataFrame Using Index

Iterate over the df DataFrame using index

```
for i in df.index:
    print(df['Problem'][i],df['Medicine'][i])
#i is the intermediate variable, indicating the
row number.
```

Result

Headache Anacin
Body ache Ibuprofen
Fever Dolo 650
Cough Strepsils
Sore Throat Tylenol
Allergies Benadryl
Indigestion Loperamide

DataFrame Using iloc()

Example: Iterate over a DataFrame named df using the iloc() function

```
data = {'Name': ['Alice', 'Bob',
                 'Charlie'],
        'Age': [25, 30, 35],
    'City': ['New York', 'London',
             'Paris']}

df = pd.DataFrame(data)

for index in range(len(df)):
    row = df.iloc[index]
    print(row)
```

Result

```
Name: Alice
Age: 25
City: New York
Name: 0, dtype: object

Name: Bob
Age: 30
City: London
Name: 1, dtype: object

Name: Charlie
Age: 35
City: Paris
Name: 2, dtype: object
```

DataFrame Using iterrows()

The iterrows() method is used to iterate over the rows of a DataFrame and perform row-wise operations.

Example: Iterate the otcm1 DataFrame using the iterrows() method.

```
for index, row in otcm1.iterrows():
    print (row["Problem"], row["Dosage"])
```

Result

Headache one time
Body ache thrice a day
Fever thrice a day, for 3 days
Cough thrice a day, for 2 days
Sore Throat thrice a day
Allergies once in the night, for 2 days
Indigestion twice a day

DataFrame Using apply()

The apply() method uses a lambda function to simplify the iterating syntax.

Example: Using lambda function in a print statement

```
print(otcm1.apply(lambda row: row["Medicine"] + " -- is for -->
                  " + str(row["Problem"]), axis = 1))
```

Result

```
0     Anacin -- is for --> Headache
1     Ibuprofen -- is for --> Body ache
2     Dolo 650 -- is for --> Fever
3     Strepsils -- is for --> Cough
4     Tylenol -- is for --> Sore Throat
5     Benadryl -- is for --> Allergies
6     Loperamide -- is for --> Indigestion
dtype: object
```

Sorting

Sorting

Sometimes sorting a DataFrame by specific columns is needed.

Pandas provides a sort function to sort and rehash a DataFrame in ascending or descending order.

The syntax of the function is shown below:

```
sort_values(by="")
```

Perform a bare-bone sorting based on the **Problem**.

```
otcm1.sort_values(by='Problem')
```

Sorting

The DataFrame is now sorted based on the alphabetical order of the parameter **Problem** as ascending=True, by default.

Problem	Medicine	Dosage
Allergies	Benadryl	Once in the night, for 2 days
Body Ache	Ibuprofen	3 times a day
Cough	Strepsils	3 times a day, for 2 days
Fever	Dolo 650	3 times a day, for 3 days
Headache	Anacin	One time
Indigestion	Lopermide	2 times a day
Sore Throat	Tylenol	3 times a day

Sorting

Set ascending=False and change the sorting order as follows:

```
otcm1.sort_values(by='Problem', ascending=False)
```

Problem	Medicine	Dosage
Sore Throat	Tylenol	3 times a day
Indigestion	Lopermide	2 times a day
Headache	Anacin	One time
Fever	Dolo 650	3 times a day, for 3 days
Cough	Strepsils	3 times a day, for 3 days
Body Ache	Ibuprofen	3 times a day
Allergies	Benadryl	Once in the night, for 2 days

As can be seen from the output, the entire DataFrame is reversed.

Discussion: Statistical Functions in Pandas

Duration: 10 minutes

- What are some popular methods for text processing?

Answer: Some of the popular text processing methods are str.upper(), str.lower(), and str.len().

  - str.upper() converts textual data to uppercase
  - str.lower() converts textual data to lowercase
  - str.len() reports the length of each element in a series or column containing strings

- How to iterate over the rows of a Pandas DataFrame?

Answer: To iterate over the rows of a Pandas DataFrame, an index, and a loop are used.

Assisted Practices

Let’s understand the topics below using Jupyter Notebooks.

- 9.15_Plotting with Pandas

Note: Please download the pdf files for each topic mentioned above from the Reference Material section.

Key Takeaways

- Pandas is a powerful data analysis library that is widely used to prepare data for machine learning models.
- It is a data wrangling tool used to perform data cleansing, data normalization, data visualization, and statistical analysis.
- Pandas provides IO tools to read and write data.
- str.upper(), str.lower(), and str.len() and some of the text processing methods.
- Iteration and sorting are common operations performed on all data types.

Knowledge Check

Knowledge Check

1

What are the two major types of data structures used in Pandas?

A. Rows and columns

B. Series and DataFrames

C. Tables and Charts

D. Arrays and Lists

Knowledge Check

1

What are the two major types of data structures used in Pandas?

A. Rows and columns

B. Series and DataFrames

C. Tables and Charts

D. Arrays and Lists

The correct answer is **B**

The two major types of data structures used in Pandas are series and DataFrames.

Knowledge Check 2

What does the isna() function do?

A. Detects missing values from an array-like object

B. Provides integral location-based indexing for selection by position

C. Returns edition of series and other element-wise edition

D. Reshapes a data frame organized by index or column values

Knowledge Check 2

What does the isna() function do?

A. Detects missing values from an array-like object

B. Provides integral location-based indexing for selection by position

C. Returns edition of series and other element-wise edition

D. Reshapes a data frame organized by index or column values

The correct answer is **A**

The isna() function is used to detect missing values from an array-like object.

Knowledge Check 3

What is the method used in Pandas to sort a DataFrame based on certain column values?

A. sort_values()

B. sort_by()

C. sort_column()

D. sort_dataframe()

Knowledge Check 3

What is the method used in Pandas to sort a DataFrame based on certain column values?

A. sort_values()

B. sort_by()

C. sort_column()

D. sort_dataframe()

The correct answer is **A**

The method used in Pandas to sort a DataFrame based on certain column values is sort_values().

Thank You