NANYANG
THE INNOVATIVE POLYTECHNIC

2024S2 - IT2313

Programming for Data Science

Data Manipulation with Pandas (Part 1)

Introduction to Pandas

Introduction to Pandas

Pandas is an open-source library providing high-performance, easy-to-use data structures and data analysis tools for the Python language.

The two data structures are DataFrame and Series.

The library's name is derived from panel data, a common term for multidimensional datasets encountered in statistics and econometrics.

Series	Data Frames
1-D array of labelled data	Labelled 2-D array if data
Series can be viewed as a hybrid of a 1-D NumPy array and a dictionary	Each column us a series sharing common row labels

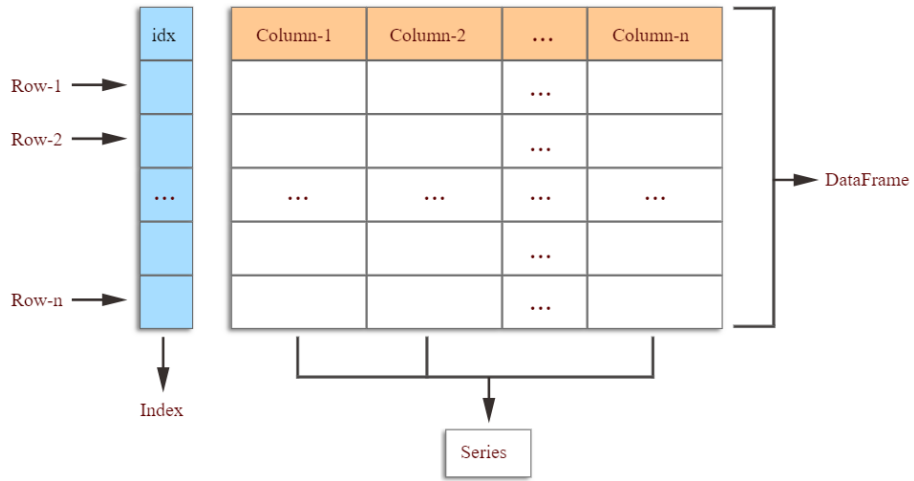
Introduction to Pandas

Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data with row and column labels
- Any other form of observational / statistical data sets.

Pandas Exercises

Pandas Data structure



What is Pandas used for?

Pandas is used throughout the data analysis workflow.

Python Pandas Tutorial

With pandas, you can:

- Import datasets from databases, spreadsheets, comma-separated values (CSV) files, and more.
- Clean datasets, for example, by dealing with missing values.
- Tidy datasets by reshaping their structure into a suitable format for analysis.
- Aggregate data by calculating summary statistics such as the mean of columns, correlation between them, and more.
- Visualize datasets and uncover insights.

Pandas also contains functionality for time series analysis and analyzing text data.

Importing Data into Pandas
Creation of Series

Importing Data into Pandas – Series Creation

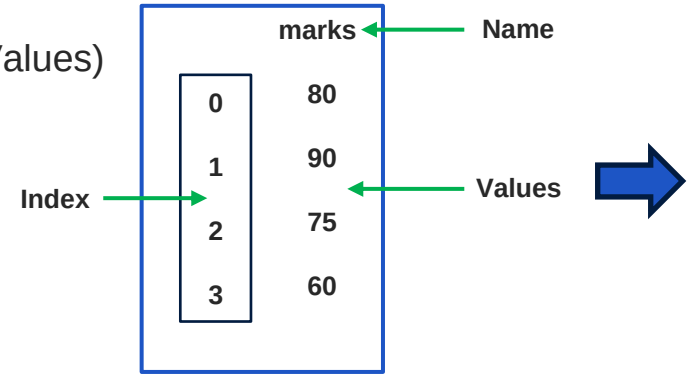
Throughout this course, I use the following import convention for Pandas.

```
# First, import the Pandas library as pd
import pandas as pd
```

The Pandas Series

The series object represents one-dimensional data structures in Pandas. A series consist of two components:

- One-dimensional data (Values)
- Index




```
pd.Series([80, 90, 75, 60],name='marks')
```

```
0    80
1    90
2    75
3    60
Name: marks, dtype: int64
```

Importing Data into Pandas – Series Creation

To create a series, you simply call the Series() function and pass as argument containing the data to be include in it. The data can be one of the following:

- A one-dimensional ndarray
- A Python list
- A Python dictionary
- A scalar value

If an index is not specified, the default index [0,… n-1] will be created, where n is the length of the data. A series can be created in a variety of ways.

Importing Data into Pandas – Series Creation

One-dimensional ndarray

The following example creates a Series of the 1st 5 Even numbers

```
np_array = np.arange(2,12,2)
ser = pd.Series(np_array)
ser
```

```
0     2
1     4
2     6
3     8
4    10
dtype: int32
```

If you do not specify any index in the function, by default, pandas will assign numerical values increasing from 0 as labels. In this case, the labels correspond to the indexes (position in the array) of the elements in the series object

Importing Data into Pandas – Series Creation

If you want to create this series using meaningful labels, you will specify the index parameter during the series creation. Labels are included inside a list of the same length of an_array

```
np_array = np.arange(2,12,2)
ser = pd.Series(np_array, index=['1st','2nd','3rd','4th','5th'])
ser
```

```
1st     2
2nd     4
3rd     6
4th     8
5th    10
dtype: int32
```

If you want to individually see the two arrays that make up this series, youcan call index and values attributes of the series.

```
ser.index
```

```
Index(['1st', '2nd', '3rd', '4th', '5th'], dtype='object')
```

```
ser.values
```

```
array([ 2,  4,  6,  8, 10])
```

Importing Data into Pandas – Series Creation

Python List

To create a series using a Python list, you can just pass a list to the data parameter of the Series() class constructor.

```
e_list = [2,4,6,8,10]
ser = pd.Series(e_list, index=['1st','2nd','3rd','4th','5th'])
ser
```

```
1st     2
2nd     4
3rd     6
4th     8
5th    10
dtype: int64
```

Python Dictionary

To create a series using a Python list, you can just pass a list to the data parameter of the Series() class constructor.

```
e_dict = {'1st':2,'2nd':4,'3rd':6,'4th':8, '5th':10}
ser = pd.Series(e_dict)
ser
```

```
1st     2
2nd     4
3rd     6
4th     8
5th    10
dtype: int64
```

Importing Data into Pandas – Series Creation

Scalar Value

The Series can also be created from a scalar value. If you do not specify the index argument, the default index is 0. If you specify the index, the value will be repeated for specified index values.

```
ser_s = pd.Series(6.8)
ser_s
```

```
0    6.8
dtype: float64
```

```
ser_s = pd.Series(6.8, index=['1st','2nd','3rd'])
ser_s
```

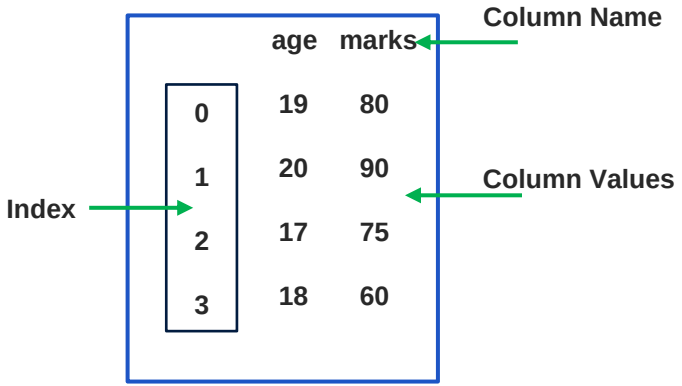
```
1st    6.8
2nd    6.8
3rd    6.8
dtype: float64
```

Importing Data into Pandas
Creation of DataFrame

Importing Data into Pandas – DataFrame Creation

The Pandas DataFrame

A DataFrame is a two-dimensional data structure consisting of rows and columns. Each column within a DataFrame is a pandas Series, and while these columns are expected to have equal lengths, they can have varying data types, such as float, int, boolean, and more.



A DataFrame consists of three components:

- Two-dimensional data (Values)
- Row index
- Column index

The DataFrame comprises two index arrays. The first array's labels are linked to entire rows, while the second array's labels correspond to specific columns. The DataFrame has two axes: axis 0 (row/index) and axis 1 (column).

Importing Data into Pandas – DataFrame Creation

The Pandas DataFrame

A DataFrame is the most used data structure in pandas. The DataFrame() class constructor accepts many different types of arguments:

- Two-dimensional ndarray
- Dictionary of dictionaries
- Dictionary of lists
- Dictionary of series

Importing Data into Pandas – DataFrame Creation

Two Dimensional ndarray

```
marks_array = np.array([[19,80],[20,90],[17,75],[18,60]])
ser = pd.DataFrame(marks_array,index=['David','John','Peter','Jessie'],columns=['age','marks'])
ser
```

	age	marks
David	19	80
John	20	90
Peter	17	75
Jessie	18	60

If you want to see the individual components which make up the DataFrame, you can call values, index and columns attributes of the DataFrame.

```
ser.values
```

```
array([[19, 80],
       [20, 90],
       [17, 75],
       [18, 60]])
```

```
ser.index
```

```
Index(['David', 'John', 'Peter', 'Jessie'], dtype='object')
```

```
ser.columns
```

```
Index(['age', 'marks'], dtype='object')
```

Importing Data into Pandas – DataFrame Creation

Dictionary of Dictionaries

```
d_dict = {'name' : {'Dd':'David','Jn':'John','Pr':'Peter','Je':'Jessie'},
          'age' : {'Dd':19,'Jn':20,'Pr':17,'Je':18},
          'marks' :{'Dd':80,'Jn':90,'Pr':75,'Je':60}}

df = pd.DataFrame(d_dict)
df
```

	name	age	marks
Dd	David	19	80
Jn	John	20	90
Pr	Peter	17	75
Je	Jessie	18	60

Column names are created from the keys of the main dictionary, and the rowindex is created from the keys of the sub dictionaries.

Importing Data into Pandas – DataFrame Creation

Dictionary of Lists

If you want to see the individual components which make up the DataFrame, you can call values, Index and columns attributes of the DataFrame.

```
d_dict = {'name' : ['David','John','Peter','Jessie'],
          'age' : [19,20,17,18],
          'marks' :[80,90,75,60]}

df = pd.DataFrame(d_dict, index=['Dd','Jn','Pr','Je'])
df
```

	name	age	marks
Dd	David	19	80
Jn	John	20	90
Pr	Peter	17	75
Je	Jessie	18	60

```
df.index
```

```
Index(['Dd', 'Jn', 'Pr', 'Je'], dtype='object')
```

```
df.columns
```

```
Index(['name', 'age', 'marks'], dtype='object')
```

```
df.values
```

```
array([['David', 19, 80],
       ['John', 20, 90],
       ['Peter', 17, 75],
       ['Jessie', 18, 60]], dtype=object)
```

```
df.values.ndim
```

```
2
```

```
df.values.shape
```

```
(4, 3)
```

Importing Data into Pandas – DataFrame Creation

Dictionary of Series

In Pandas, a DataFrame can be created by combining a dictionary of Series and organizing data into a structured tabular format.

```
ser1 = pd.Series(['David','John','Peter','Jessie'],index=['Dd','Jn','Pr','Je'])
ser2 = pd.Series([19,20,17,18],index=['Dd','Jn','Pr','Je'])
ser3 = pd.Series([80,90,75,60],index=['Dd','Jn','Pr','Je'])
e_dict = {'name':ser1, 'age':ser2, 'marks':ser3}

df = pd.DataFrame(e_dict)
df
```

	name	age	marks
Dd	David	19	80
Jn	John	20	90
Pr	Peter	17	75
Je	Jessie	18	60

Importing Data into Pandas – Using Pandas Read

Dictionary of Dictionaries

Creating a Pandas Data Frame from a dictionary of dictionaries is a powerful method, allowing for the straightforward transformation of nested key-value pairs into a structured tabular representation for efficient data manipulation and analysis

```
d_dict = {'name' : {'Dd':'David','Jn':'John','Pr':'Peter','Je':'Jessie'},
          'age' : {'Dd':19,'Jn':20,'Pr':17,'Je':18},
          'marks' :{'Dd':80,'Jn':90,'Pr':75,'Je':60}}

df = pd.DataFrame(d_dict)
df
```

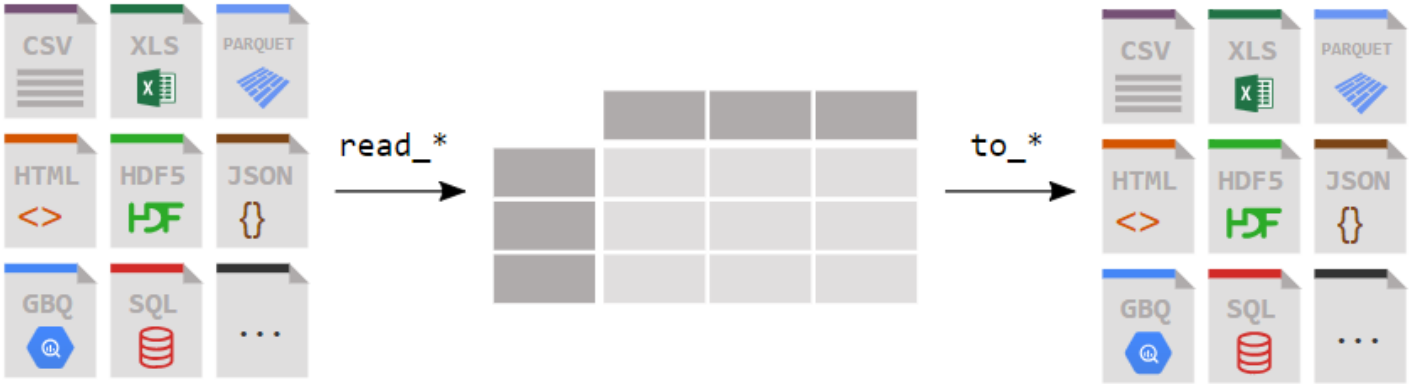
	name	age	marks
Dd	David	19	80
Jn	John	20	90
Pr	Peter	17	75
Je	Jessie	18	60

Column names are created from the keys of the main dictionary, and the row index is created from the keys of the sub dictionaries.

Importing Data into Pandas
Using Pandas Read

Importing Data into Pandas – Using Pandas Read

Pandas supports many different file format read and write operations such as csv, text, Excel, sql, json and each of them with prefix read_*. In this course, we will be focus on csv and Excel file formats only.



[How do I read and write tabular data]

- Pandas read_csv() function: Reads a comma-separated values (csv) file ora text file into a Pandas DataFrame.
- Pandas read_excel() function: Reads an Excel file into a Pandas DataFrame

Importing Data into Pandas – Using Pandas Read

Let start with reading in the following csv file which contain information of a class test. This CSV file appears to contain information about individuals, including their names, ages, and marks. Each row represents a person, with columns specifying their name, age, and marks

```
name,age,marks
David,19,80
John,20,90
Peter,17,75
Jessie,18,60
```

DataFrame Creation from csv File

The following command was used to read in the dataset from the csv file into Pandas DataFrame

```
import pandas as pd

# The CSV file is named 'marks.csv'
df = pd.read_csv('marks.csv')
df
```

	name	age	marks
0	David	19	80
1	John	20	90
2	Peter	17	75
3	Jessie	18	60

Importing Data into Pandas – Using Pandas Read

DataFrame Creation from Excel File

The following command was used to read in the dataset from the Excel file into Pandas DataFrame

name	age	marks
David	19	80
John	20	90
Peter	17	75
Jessie	18	60

```
import pandas as pd

# The CSV file is named 'xlsx.csv'
df = pd.read_excel('marks.xlsx')
df
```

	name	age	marks
0	David	19	80
1	John	20	90
2	Peter	17	75
3	Jessie	18	60

Are you able to spot the differences in importing data into Pandas from csv and Excel file?

Changing the Values in DataFrame

Changing the Values in DataFrame

Assigning Values to the Elements and Adding New Columns

DataFrame are both ***value-mutable*** and ***size-mutable***. You can change values within the DataFrame or add/delete columns to/from the DataFrame.

Value Mutability (changing the values)

Let change David's marks from 80 to 85 using the following command

```
df.iloc[0,2] = 85
df
```

	name	age	marks
0	David	19	80
1	John	20	90
2	Peter	17	75
3	Jessie	18	60

	name	age	marks
0	David	19	85
1	John	20	90
2	Peter	17	75
3	Jessie	18	60

Changing the Values in DataFrame

Size Mutability (Adding a new Column)

Let add a new column to indicate their Grading.

```
df['grade'] = ['A', 'A+', 'B+', 'C']
df
```

	name	age	marks
0	David	19	85
1	John	20	90
2	Peter	17	75
3	Jessie	18	60

	name	age	marks	grade
0	David	19	85	A
1	John	20	90	A+
2	Peter	17	75	B+
3	Jessie	18	60	C

Size Mutability (Adding a new Row)

Let add a new row to include one more student mark.

```
df.loc[len(df)] = ['Aaron', 23, 54, 'C']
df
```

	name	age	marks	grade
0	David	19	85	A
1	John	20	90	A+
2	Peter	17	75	B+
3	Jessie	18	60	C

	name	age	marks	grade
0	David	19	85	A
1	John	20	90	A+
2	Peter	17	75	B+
3	Jessie	18	60	C
4	Aaron	23	54	C

Changing the Values in DataFrame

Size Mutability (Removing an existing Column)

Let remove column ‘age’ from the DataFrame.

```
df = df.drop('age', axis=1)
df
```

	name	age	marks	grade
0	David	19	85	A
1	John	20	90	A+
2	Peter	17	75	B+
3	Jessie	18	60	C

	name	marks	grade
0	David	85	A
1	John	90	A+
2	Peter	75	B+
3	Jessie	60	C
4	Aaron	54	C

Size Mutability (Removing an existing Row)

Let remove ‘Jessie’ from the row in the DataFrame

```
# This will remove the specified row from the DataFrame.
# If you want to remove a row based on a condition,
# you can use boolean indexing.
df = df[df['name'] != 'Jessie']
df
```

	name	marks	grade
0	David	85	A
1	John	90	A+
2	Peter	75	B+
3	Jessie	60	C
4	Aaron	54	C

	name	marks	grade
0	David	85	A
1	John	90	A+
2	Peter	75	B+
4	Aaron	54	C

DataFrame Data Exploration

DataFrame Data Exploration

Data Exploration in a DataFrame is crucial for uncovering patterns and trends, making complex information more understandable and aiding in Informed decision-making through visual insights.

We will be using the following DataFrame in this topic. You can also down download the dataset file “dogs.csv” from the PoliteMall.

```
import pandas as pd

# The CSV file is named 'dogs.csv'
df = pd.read_csv('dogs.csv')
df
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
0	Bella	Labrador	Brown	56	25	07/01/15
1	Charlie	Poodle	Black	43	23	09/16/18
2	Lucy	Chow Chow	Brown	46	22	08/25/16
3	Cooper	Schnauzer	Gray	49	17	12/11/14
4	Max	Labrador	Black	59	29	01/20/19
5	Stella	Chihuahua	Tan	18	2	04/20/15
6	Bernie	St.Bernard	White	77	74	02/27/20
7	Daisy	Bulldog	Brindle	31	23	05/14/17
8	Milo	Golden Retriever	Golden	63	30	03/08/16
9	Lola	Pug	Fawn	25	6	10/03/18

DataFrame Data Exploration

We will be covering the following 10 most commonly used functions for Pandas Data Exploration in this topic.

1. head(): Displays the first 5 rows of the DataFrame, 5 is the default value.
2. head(3): Shows the first 3 rows of the DataFrame.
3. tail(): Displays the last 5 rows of the DataFrame, 5 is the default value.
4. tail(3): Shows the last 3 rows of the DataFrame.
5. info(): Provides a concise summary of the DataFrame, including data types and missing values.
6. shape: Returns the number of rows and columns in the DataFrame.
7. describe(): Generates descriptive statistics, including measures of central tendency and variability.
8. values: Returns a Numpy representation of the DataFrame.
9. columns: Displays the column labels of the DataFrame.
10. index: Shows the index (row labels) of the DataFrame.

DataFrame Data Exploration

head(): Displays the first 5 rows of the DataFrame, 5 is the default value.

```
df.head()
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
0	Bella	Labrador	Brown	56	25	07/01/15
1	Charlie	Poodle	Black	43	23	09/16/18
2	Lucy	Chow Chow	Brown	46	22	08/25/16
3	Cooper	Schnauzer	Gray	49	17	12/11/14
4	Max	Labrador	Black	59	29	01/20/19

head(3): Shows the first 3 rows of the DataFrame.

```
df.head(3)
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
0	Bella	Labrador	Brown	56	25	07/01/15
1	Charlie	Poodle	Black	43	23	09/16/18
2	Lucy	Chow Chow	Brown	46	22	08/25/16

DataFrame Data Exploration

tail(): Displays the last 5 rows of the DataFrame, 5 is the default value.

```
df.tail()
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
5	Stella	Chihuahua	Tan	18	2	04/20/15
6	Bernie	St.Bernard	White	77	74	02/27/20
7	Daisy	Bulldog	Brindle	31	23	05/14/17
8	Milo	Golden Retriever	Golden	63	30	03/08/16
9	Lola	Pug	Fawn	25	6	10/03/18

tail(3): Shows the last 3 rows of the DataFrame.

```
df.tail(3)
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
7	Daisy	Bulldog	Brindle	31	23	05/14/17
8	Milo	Golden Retriever	Golden	63	30	03/08/16
9	Lola	Pug	Fawn	25	6	10/03/18

DataFrame Data Exploration

info(): Provides a concise summary of the DataFrame, including data types and missing values.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Name           10 non-null     object
 1   Breed          10 non-null     object
 2   Color          10 non-null     object
 3   Height(cm)     10 non-null     int64
 4   Weight(kg)     10 non-null     int64
 5   Date of Birth  10 non-null     object
dtypes: int64(2), object(4)
memory usage: 612.0+ bytes
```

shape: Returns the number of rows and columns in the DataFrame.

```
df.shape
```

```
(10, 6)
```

DataFrame Data Exploration

describe(): Generates descriptive statistics, including measures of central tendency and variability.

```
df.describe()
```

	Height(cm)	Weight(kg)
count	10.000000	10.000000
mean	46.700000	25.100000
std	18.202869	19.473344
min	18.000000	2.000000
25%	34.000000	18.250000
50%	47.500000	23.000000
75%	58.250000	28.000000
max	77.000000	74.000000

values: Returns a Numpy representation of the DataFrame.

```
df.values
```

```
array([['Bella', 'Labrador', 'Brown', 56, 25, '07/01/15'],
       ['Charlie', 'Poodle', 'Black', 43, 23, '09/16/18'],
       ['Lucy', 'Chow Chow', 'Brown', 46, 22, '08/25/16'],
       ['Cooper', 'Schnauzer', 'Gray', 49, 17, '12/11/14'],
       ['Max', 'Labrador', 'Black', 59, 29, '01/20/19'],
       ['Stella', 'Chihuahua', 'Tan', 18, 2, '04/20/15'],
       ['Bernie', 'St.Bernard', 'White', 77, 74, '02/27/20'],
       ['Daisy', 'Bulldog', 'Brindle', 31, 23, '05/14/17'],
       ['Milo', 'Golden Retriever', 'Golden', 63, 30, '03/08/16'],
       ['Lola', 'Pug', 'Fawn', 25, 6, '10/03/18']], dtype=object)
```

DataFrame Data Exploration

columns: Displays the column labels of the DataFrame.

```
df.columns
```

```
Index(['Name', 'Breed', 'Color', 'Height(cm)', 'Weight(kg)', 'Date of Birth'], dtype='object')
```

index: Shows the index (row labels) of the DataFrame.

```
df.index
```

```
RangeIndex(start=0, stop=10, step=1)
```

Data Exploration lays the foundation for a deeper understanding of the dataset, setting up the basics for things like cleaning data, data transformation, creating features, and building models.

It helps us figure out how the data is set up, check its quality, and discover connections between different parts, guiding us in making smart decisions for further analysis.

DataFrame Sorting and Filtering

DataFrame Data Sorting

Pandas DataFrame data selection allows for easy and efficient extraction and manipulation of specific subsets of data using intuitive methods and powerful indexing. Using back the dataset file “dogs.csv” from the PoliteMall.

```
import pandas as pd

# The CSV file is named 'dogs.csv'
df = pd.read_csv('dogs.csv')
df
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
0	Bella	Labrador	Brown	56	25	07/01/15
1	Charlie	Poodle	Black	43	23	09/16/18
2	Lucy	Chow Chow	Brown	46	22	08/25/16
3	Cooper	Schnauzer	Gray	49	17	12/11/14
4	Max	Labrador	Black	59	29	01/20/19
5	Stella	Chihuahua	Tan	18	2	04/20/15
6	Bernie	St.Bernard	White	77	74	02/27/20
7	Daisy	Bulldog	Brindle	31	23	05/14/17
8	Milo	Golden Retriever	Golden	63	30	03/08/16
9	Lola	Pug	Fawn	25	6	10/03/18

DataFrame Data Sorting

Let perform the DataFrame sorting based on the Dogs Height. We can do it in ascending and descending order. The sort_values() function sorts the values of the DataFrame based on the column name. It has an optional parameter - ascending. States if the column is sorted in an ascending order

```
# Sort in ascending order based on Height
height_ascending_order = df.sort_values('Height(cm)')
height_ascending_order
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
5	Stella	Chihuahua	Tan	18	2	04/20/15
9	Lola	Pug	Fawn	25	6	10/03/18
7	Daisy	Bulldog	Brindle	31	23	05/14/17
1	Charlie	Poodle	Black	43	23	09/16/18
2	Lucy	Chow Chow	Brown	46	22	08/25/16
3	Cooper	Schnauzer	Gray	49	17	12/11/14
0	Bella	Labrador	Brown	56	25	07/01/15
4	Max	Labrador	Black	59	29	01/20/19
8	Milo	Golden Retriever	Golden	63	30	03/08/16
6	Bernie	St.Bernard	White	77	74	02/27/20

```
# Sort in descending order based on Height
height_descending_order = df.sort_values('Height(cm)', ascending=False)
height_descending_order
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
6	Bernie	St.Bernard	White	77	74	02/27/20
8	Milo	Golden Retriever	Golden	63	30	03/08/16
4	Max	Labrador	Black	59	29	01/20/19
0	Bella	Labrador	Brown	56	25	07/01/15
3	Cooper	Schnauzer	Gray	49	17	12/11/14
2	Lucy	Chow Chow	Brown	46	22	08/25/16
1	Charlie	Poodle	Black	43	23	09/16/18
7	Daisy	Bulldog	Brindle	31	23	05/14/17
9	Lola	Pug	Fawn	25	6	10/03/18
5	Stella	Chihuahua	Tan	18	2	04/20/15

DataFrame Data Sorting

The sort_values() also provides an option to sort the DataFrame by multiple variables.

```
# Sort in ascending order based on Height follow by Weight
dogs_ascending_order = df.sort_values(['Height(cm)','Weight(kg)'])
dogs_ascending_order
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
5	Stella	Chihuahua	Tan	18	2	04/20/15
9	Lola	Pug	Fawn	25	6	10/03/18
7	Daisy	Bulldog	Brindle	31	23	05/14/17
1	Charlie	Poodle	Black	43	23	09/16/18
2	Lucy	Chow Chow	Brown	46	22	08/25/16
3	Cooper	Schnauzer	Gray	49	17	12/11/14
0	Bella	Labrador	Brown	56	25	07/01/15
4	Max	Labrador	Black	59	29	01/20/19
8	Milo	Golden Retriever	Golden	63	30	03/08/16
6	Bernie	St.Bernard	White	77	74	02/27/20

The DataFrame is being arranged in ascending order first by 'Height(cm)'.

For rows with the same height, it's further sorted by 'Weight(kg)' in ascending order.

DataFrame Data Filtering

```
# Select only the 'Name' and 'Breed' columns
selected_columns_1 = df[['Name', 'Breed']]
selected_columns_1
```

	Name	Breed
0	Bella	Labrador
1	Charlie	Poodle
2	Lucy	Chow Chow
3	Cooper	Schnauzer
4	Max	Labrador
5	Stella	Chihuahua
6	Bernie	St.Bernard
7	Daisy	Bulldog
8	Milo	Golden Retriever
9	Lola	Pug

```
# Filter rows where the 'Breed' is 'Labrador'
labrador_dogs = df[df['Breed'] == 'Labrador']
labrador_dogs
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
0	Bella	Labrador	Brown	56	25	07/01/15
4	Max	Labrador	Black	59	29	01/20/19

Do take note on the ‘=‘ and ‘==‘ operators

DataFrame Data Filtering

Let try to perform two filtering's on the DataFrame the dogs which are > 25kg and < 30cm

```
# Filter for dogs with weight more than 25kg
dogs_gt_25kg = df[df['Weight(kg)'] > 25]

# Display the result
dogs_gt_25kg
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
4	Max	Labrador	Black	59	29	01/20/19
6	Bernie	St.Bernard	White	77	74	02/27/20
8	Milo	Golden Retriever	Golden	63	30	03/08/16

```
# Filter for dogs with weight less than 30cm
dogs_lt_30cm = df[df['Height(cm)'] < 30]

# Display the result
dogs_lt_30cm
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
5	Stella	Chihuahua	Tan	18	2	04/20/15
9	Lola	Pug	Fawn	25	6	10/03/18

DataFrame Data Filtering

Let filter dogs that are either Labrador Retrievers or Poodles using the isin() method:

```
# Create a list of breeds to filter
desired_breeds = ['Labrador', 'Poodle']

# Filter rows where the 'Breed' is in the list of desired breeds
filtered_dogs_by_breed = df[df['Breed'].isin(desired_breeds)]
filtered_dogs_by_breed
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
0	Bella	Labrador	Brown	56	25	07/01/15
1	Charlie	Poodle	Black	43	23	09/16/18
4	Max	Labrador	Black	59	29	01/20/19

DataFrame Data Filtering

The groupby() function in Pandas is useful for grouping rows based on a certain criterion. Let's group the dogs by color and find the average height and weight for each color:

```
# Group the DataFrame by 'Color'
grouped_by_color = df.groupby('Color')

# Calculate the average height and weight for each color
average_stats_by_color = grouped_by_color[['Height(cm)', 'Weight(kg)']].mean()
average_stats_by_color
```

Color	Height(cm)	Weight(kg)
Black	51.0	26.0
Brindle	31.0	23.0
Brown	51.0	23.5
Fawn	25.0	6.0
Golden	63.0	30.0
Gray	49.0	17.0
Tan	18.0	2.0
White	77.0	74.0

Slicing and Subsetting with .loc and .iloc

Slicing and Subsetting with .loc and .iloc

Slicing Lists

Let start off with the following list.

```
breeds = ["Labrador", "Poodle", "Chow Chow",
          "Schnauzer", "Labrador", "Chihuahua",
          "St. Bernard", "Bulldog",
          "Golden Retriever", "Pug"]
breeds
```

```
['Labrador',
 'Poodle',
 'Chow Chow',
 'Schnauzer',
 'Labrador',
 'Chihuahua',
 'St. Bernard',
 'Bulldog',
 'Golden Retriever',
 'Pug']
```

```
breeds[2:5]
```

```
['Chow Chow', 'Schnauzer', 'Labrador']
```

```
breeds[:3]
```

```
['Labrador', 'Poodle', 'Chow Chow']
```

```
breeds[:]
```

```
['Labrador',
 'Poodle',
 'Chow Chow',
 'Schnauzer',
 'Labrador',
 'Chihuahua',
 'St. Bernard',
 'Bulldog',
 'Golden Retriever',
 'Pug']
```

Slicing and Subsetting with .loc and .iloc

Let Sort the index before you slice.

```
dogs_srt = dogs.set_index(["Breed", "Color"]).sort_index()
dogs_srt
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
0	Bella	Labrador	Brown	56	25	07/01/15
1	Charlie	Poodle	Black	43	23	09/16/18
2	Lucy	Chow Chow	Brown	46	22	08/25/16
3	Cooper	Schnauzer	Gray	49	17	12/11/14
4	Max	Labrador	Black	59	29	01/20/19
5	Stella	Chihuahua	Tan	18	2	04/20/15
6	Bernie	St.Bernard	White	77	74	02/27/20
7	Daisy	Bulldog	Brindle	31	23	05/14/17
8	Milo	Golden Retriever	Golden	63	30	03/08/16
9	Lola	Pug	Fawn	25	6	10/03/18

Breed	Color	Name	Height(cm)	Weight(kg)	Date of Birth
Bulldog	Brindle	Daisy	31	23	05/14/17
Chihuahua	Tan	Stella	18	2	04/20/15
Chow Chow	Brown	Lucy	46	22	08/25/16
Golden Retriever	Golden	Milo	63	30	03/08/16
Labrador	Black	Max	59	29	01/20/19
	Brown	Bella	56	25	07/01/15
Poodle	Black	Charlie	43	23	09/16/18
Pug	Fawn	Lola	25	6	10/03/18
Schnauzer	Gray	Cooper	49	17	12/11/14
St.Bernard	White	Bernie	77	74	02/27/20

Slicing and Subsetting with .loc and .iloc

Slicing the outer index level

```
dogs_srt.loc["Chow Chow":"Poodle"]
```

Breed	Color	Name	Height(cm)	Weight(kg)	Date of Birth
Bulldog	Brindle	Daisy	31	23	05/14/17
Chihuahua	Tan	Stella	18	2	04/20/15
Chow Chow	Brown	Lucy	46	22	08/25/16
Golden Retriever	Golden	Milo	63	30	03/08/16
Labrador	Black	Max	59	29	01/20/19
	Brown	Bella	56	25	07/01/15
Poodle	Black	Charlie	43	23	09/16/18
Pug	Fawn	Lola	25	6	10/03/18
Schnauzer	Gray	Cooper	49	17	12/11/14
St.Bernard	White	Bernie	77	74	02/27/20

Breed	Color	Name	Height(cm)	Weight(kg)	Date of Birth
Chow Chow	Brown	Lucy	46	22	08/25/16
Golden Retriever	Golden	Milo	63	30	03/08/16
Labrador	Black	Max	59	29	01/20/19
	Brown	Bella	56	25	07/01/15
Poodle	Black	Charlie	43	23	09/16/18

The final value "Poodle" is included

Slicing and Subsetting with .loc and .iloc

Slicing columns

```
dogs_srt.loc[:, "Name":"Height(cm)"]
```

Breed	Color	Name	Height(cm)	Weight(kg)	Date of Birth
Bulldog	Brindle	Daisy	31	23	05/14/17
Chihuahua	Tan	Stella	18	2	04/20/15
Chow Chow	Brown	Lucy	46	22	08/25/16
Golden Retriever	Golden	Milo	63	30	03/08/16
Labrador	Black	Max	59	29	01/20/19
	Brown	Bella	56	25	07/01/15
Poodle	Black	Charlie	43	23	09/16/18
Pug	Fawn	Lola	25	6	10/03/18
Schnauzer	Gray	Cooper	49	17	12/11/14
St.Bernard	White	Bernie	77	74	02/27/20

		Name	Height(cm)
Breed	Color		
Bulldog	Brindle	Daisy	31
Chihuahua	Tan	Stella	18
Chow Chow	Brown	Lucy	46
Golden Retriever	Golden	Milo	63
Labrador	Black	Max	59
	Brown	Bella	56
Poodle	Black	Charlie	43
Pug	Fawn	Lola	25
Schnauzer	Gray	Cooper	49
St.Bernard	White	Bernie	77

Slicing and Subsetting with .loc and .iloc

Slice Twice

```
dogs_srt.loc[("Labrador", "Brown"):("Schnauzer", "Grey"),"Name":"Height(cm)"]
```

Breed	Color	Name	Height(cm)	Weight(kg)	Date of Birth
Bulldog	Brindle	Daisy	31	23	05/14/17
Chihuahua	Tan	Stella	18	2	04/20/15
Chow Chow	Brown	Lucy	46	22	08/25/16
Golden Retriever	Golden	Milo	63	30	03/08/16
Labrador	Black	Max	59	29	01/20/19
	Brown	Bella	56	25	07/01/15
Poodle	Black	Charlie	43	23	09/16/18
Pug	Fawn	Lola	25	6	10/03/18
Schnauzer	Gray	Cooper	49	17	12/11/14
St.Bernard	White	Bernie	77	74	02/27/20

Breed	Color	Name	Height(cm)
Labrador	Brown	Bella	56
Poodle	Black	Charlie	43
Pug	Fawn	Lola	25
Schnauzer	Gray	Cooper	49

Slicing and Subsetting with .loc and .iloc

Dog by Days

```
# Convert 'Date of Birth' to datetime format in Pandas
# errors='coerce' replaces any invalid date strings with NaT (Not a Time),
# for missing or incorrect dates in datetime objects.
dogs['Date of Birth'] = pd.to_datetime(dogs['Date of Birth'], errors='coerce')

# Set 'Date of Birth' as the index and sort
dogs = dogs.set_index('Date of Birth').sort_index()
dogs
```

Breed	Color	Name	Height(cm)	Weight(kg)	Date of Birth
Bulldog	Brindle	Daisy	31	23	05/14/17
Chihuahua	Tan	Stella	18	2	04/20/15
Chow Chow	Brown	Lucy	46	22	08/25/16
Golden Retriever	Golden	Milo	63	30	03/08/16
Labrador	Black	Max	59	29	01/20/19
	Brown	Bella	56	25	07/01/15
Poodle	Black	Charlie	43	23	09/16/18
Pug	Fawn	Lola	25	6	10/03/18
Schnauzer	Gray	Cooper	49	17	12/11/14
St.Bernard	White	Bernie	77	74	02/27/20

Date of Birth	Name	Breed	Color	Height(cm)	Weight(kg)
2014-12-11	Cooper	Schnauzer	Gray	49	17
2015-04-20	Stella	Chihuahua	Tan	18	2
2015-07-01	Bella	Labrador	Brown	56	25
2016-03-08	Milo	Golden Retriever	Golden	63	30
2016-08-25	Lucy	Chow Chow	Brown	46	22
2017-05-14	Daisy	Bulldog	Brindle	31	23
2018-09-16	Charlie	Poodle	Black	43	23
2018-10-03	Lola	Pug	Fawn	25	6
2019-01-20	Max	Labrador	Black	59	29
2020-02-27	Bernie	St.Bernard	White	77	74

Slicing and Subsetting with .loc and .iloc

Slicing by Dates

```
# Get dogs with date_of_birth between 2014-08-25 and 2016-09-16
dogs.loc["2014-08-25":"2016-09-16"]
```

Breed	Color	Name	Height(cm)	Weight(kg)	Date of Birth
Bulldog	Brindle	Daisy	31	23	05/14/17
Chihuahua	Tan	Stella	18	2	04/20/15
Chow Chow	Brown	Lucy	46	22	08/25/16
Golden Retriever	Golden	Milo	63	30	03/08/16
Labrador	Black	Max	59	29	01/20/19
	Brown	Bella	56	25	07/01/15
Poodle	Black	Charlie	43	23	09/16/18
Pug	Fawn	Lola	25	6	10/03/18
Schnauzer	Gray	Cooper	49	17	12/11/14
St.Bernard	White	Bernie	77	74	02/27/20

Date of Birth	Name	Breed	Color	Height(cm)	Weight(kg)
2014-12-11	Cooper	Schnauzer	Gray	49	17
2015-04-20	Stella	Chihuahua	Tan	18	2
2015-07-01	Bella	Labrador	Brown	56	25
2016-03-08	Milo	Golden Retriever	Golden	63	30
2016-08-25	Lucy	Chow Chow	Brown	46	22

Slicing and Subsetting with .loc and .iloc

Slicing by Partial Dates

```
# Get dogs with date_of_birth between 2014-01-01 and 2016-12-31
dogs.loc["2014":"2016"]
```

Breed	Color	Name	Height(cm)	Weight(kg)	Date of Birth
Bulldog	Brindle	Daisy	31	23	05/14/17
Chihuahua	Tan	Stella	18	2	04/20/15
Chow Chow	Brown	Lucy	46	22	08/25/16
Golden Retriever	Golden	Milo	63	30	03/08/16
Labrador	Black	Max	59	29	01/20/19
	Brown	Bella	56	25	07/01/15
Poodle	Black	Charlie	43	23	09/16/18
Pug	Fawn	Lola	25	6	10/03/18
Schnauzer	Gray	Cooper	49	17	12/11/14
St.Bernard	White	Bernie	77	74	02/27/20

Date of Birth	Name	Breed	Color	Height(cm)	Weight(kg)
2014-12-11	Cooper	Schnauzer	Gray	49	17
2015-04-20	Stella	Chihuahua	Tan	18	2
2015-07-01	Bella	Labrador	Brown	56	25
2016-03-08	Milo	Golden Retriever	Golden	63	30
2016-08-25	Lucy	Chow Chow	Brown	46	22

Slicing and Subsetting with .loc and .iloc

Subsetting by Row and Column Number

```
dogs.iloc[2:5, 1:4]
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth
0	Bella	Labrador	Brown	56	25	07/01/15
1	Charlie	Poodle	Black	43	23	09/16/18
2	Lucy	Chow Chow	Brown	46	22	08/25/16
3	Cooper	Schnauzer	Gray	49	17	12/11/14
4	Max	Labrador	Black	59	29	01/20/19
5	Stella	Chihuahua	Tan	18	2	04/20/15
6	Bernie	St.Bernard	White	77	74	02/27/20
7	Daisy	Bulldog	Brindle	31	23	05/14/17
8	Milo	Golden Retriever	Golden	63	30	03/08/16
9	Lola	Pug	Fawn	25	6	10/03/18

	Breed	Color	Height(cm)
2	Chow Chow	Brown	46
3	Schnauzer	Gray	49
4	Labrador	Black	59

Aggregating Methods

•median(), mode(), min(), max(), median(), sum(), var(), std(), quantile()

DataFrame Aggregating Methods

Calculate summary statistics on DataFrame columns, and master grouped summary statistics and pivot tables.

median(): Calculates and returns the median (middle value) of a sequence of numbers.

mode(): Identifies and returns the mode (most frequently occurring value) in a sequence of numbers.

min(): Finds and returns the smallest value in a collection of numbers.

max(): Identifies and returns the largest value in a collection of numbers.

sum(): Adds up all the values in a sequence and returns the total.

var(): Computes and returns the variance of a set of numbers, a measure of their spread.

std(): Calculates and returns the standard deviation, a measure of the amount of variation or dispersion in a set of values.

quantile(): Determines and returns the quantile (a specific percentile) of a dataset.

•median(), mode(), min(), max(), median(), sum(), var(), std(), quantile()

DataFrame Aggregating Methods

The outputs for the listed aggregating methods using the “dogs.csv” dataset from the earlier example.

```
# Applying the functions
median_height = df['Height(cm)'].median()
mode_breed = df['Breed'].mode()[0]
min_weight = df['Weight(kg)'].min()
max_weight = df['Weight(kg)'].max()
total_weight = df['Weight(kg)'].sum()
variance_height = df['Height(cm)'].var()
std_weight = df['Weight(kg)'].std()
quantile_height = df['Height(cm)'].quantile(0.75)

# Displaying the results
print(f"Median Height: {median_height} cm")
print(f"Mode Breed: {mode_breed}")
print(f"Min Weight: {min_weight} kg")
print(f"Max Weight: {max_weight} kg")
print(f"Total Weight: {total_weight} kg")
print(f"Variance in Height: {variance_height} cm^2")
print(f"Standard Deviation of Weight: {std_weight} kg")
print(f"75th Percentile of Height: {quantile_height} cm")
```

```
Median Height: 47.5 cm
Mode Breed: Labrador
Min Weight: 2 kg
Max Weight: 74 kg
Total Weight: 251 kg
Variance in Height: 331.3444444444445 cm^2
Standard Deviation of Weight: 19.473343603785946 kg
75th Percentile of Height: 58.25 cm
```

DataFrame Aggregating Methods

Let try to add two new columns to the DataFrame, the BMI and the Age of the Dogs.

```
# 1. Calculate BMI (Body Mass Index)
df['BMI'] = df['Weight(kg)'] / ((df['Height(cm)'] / 100) ** 2)
df
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth	BMI
0	Bella	Labrador	Brown	56	25	07/01/15	79.719388
1	Charlie	Poodle	Black	43	23	09/16/18	124.391563
2	Lucy	Chow Chow	Brown	46	22	08/25/16	103.969754
3	Cooper	Schnauzer	Gray	49	17	12/11/14	70.803832
4	Max	Labrador	Black	59	29	01/20/19	83.309394
5	Stella	Chihuahua	Tan	18	2	04/20/15	61.728395
6	Bernie	St.Bernard	White	77	74	02/27/20	124.810255
7	Daisy	Bulldog	Brindle	31	23	05/14/17	239.334027
8	Milo	Golden Retriever	Golden	63	30	03/08/16	75.585790
9	Lola	Pug	Fawn	25	6	10/03/18	96.000000

```
# 2. Calculate Age of the Pets
from datetime import datetime
df['Date of Birth'] = pd.to_datetime(df['Date of Birth'], errors='coerce')
current_year = datetime.now().year
df['Age'] = current_year - df['Date of Birth'].dt.year
df
```

	Name	Breed	Color	Height(cm)	Weight(kg)	Date of Birth	BMI	Age
0	Bella	Labrador	Brown	56	25	2015-07-01	79.719388	9
1	Charlie	Poodle	Black	43	23	2018-09-18	124.391563	6
2	Lucy	Chow Chow	Brown	46	22	2016-08-25	103.969754	8
3	Cooper	Schnauzer	Gray	49	17	2014-12-11	70.803832	10
4	Max	Labrador	Black	59	29	2019-01-20	83.309394	5
5	Stella	Chihuahua	Tan	18	2	2015-04-20	61.728395	9
6	Bernie	St.Bernard	White	77	74	2020-02-27	124.810255	4
7	Daisy	Bulldog	Brindle	31	23	2017-05-14	239.334027	7
8	Milo	Golden Retriever	Golden	63	30	2016-03-08	75.585790	8
9	Lola	Pug	Fawn	25	6	2018-10-03	96.000000	6

Adding new columns to a DataFrame is essential for creating derived features, providing valuable insights and context that enhance analysis and decision-making.

Examples on DataFrame Slicing and Indexing

DataFrame Slicing and Indexing

Using the following DataFrame as the reference.

```
player_list = [['Luna Rodriguez', 36, 75, 185],
               ['Oliver Chen', 38, 74, 165],
               ['Ava Singh', 31, 70, 173],
               ['Ethan Patel', 34, 80, 175],
               ['Mia Kim', 40, 100, 165],
               ['Lucas Gupta', 33, 72, 169],
               ['Sophia Zhang', 42, 85, 175]]

df = pd.DataFrame(player_list, columns=['Name', 'Age', 'Weight', 'Height'])
df
```

	Name	Age	Weight	Height
0	Luna Rodriguez	36	75	185
1	Oliver Chen	38	74	165
2	Ava Singh	31	70	173
3	Ethan Patel	34	80	175
4	Mia Kim	40	100	165
5	Lucas Gupta	33	72	169
6	Sophia Zhang	42	85	175

Indexing Pandas DataFrame

```
player_list = [['Luna Rodriguez', 36, 75, 185],
               ['Oliver Chen', 38, 74, 165],
               ['Ava Singh', 31, 70, 173],
               ['Ethan Patel', 34, 80, 175],
               ['Mia Kim', 40, 100, 165],
               ['Lucas Gupta', 33, 72, 169],
               ['Sophia Zhang', 42, 85, 175]]

df = pd.DataFrame(player_list, columns=['Name','Age','Weight','Height'],
            index=['A','B','C','D','E','F','G'])

df
```

	Name	Age	Weight	Height
A	Luna Rodriguez	36	75	185
B	Oliver Chen	38	74	165
C	Ava Singh	31	70	173
D	Ethan Patel	34	80	175
E	Mia Kim	40	100	165
F	Lucas Gupta	33	72	169
G	Sophia Zhang	42	85	175

DataFrame Slicing and Indexing

Slicing Rows

```
# Slicing rows in data frame
df1 = df.iloc[0:4]
df1
```

	Name	Age	Weight	Height
A	Luna Rodriguez	36	75	185
B	Oliver Chen	38	74	165
C	Ava Singh	31	70	173
D	Ethan Patel	34	80	175
E	Mia Kim	40	100	165
F	Lucas Gupta	33	72	169
G	Sophia Zhang	42	85	175

	Name	Age	Weight	Height
A	Luna Rodriguez	36	75	185
B	Oliver Chen	38	74	165
C	Ava Singh	31	70	173
D	Ethan Patel	34	80	175

Slicing Columns

```
# Slicing columnss in data frame
df1 = df.iloc[:, 0:2]
df1
```

	Name	Age	Weight	Height
A	Luna Rodriguez	36	75	185
B	Oliver Chen	38	74	165
C	Ava Singh	31	70	173
D	Ethan Patel	34	80	175
E	Mia Kim	40	100	165
F	Lucas Gupta	33	72	169
G	Sophia Zhang	42	85	175

	Name	Age
A	Luna Rodriguez	36
B	Oliver Chen	38
C	Ava Singh	31
D	Ethan Patel	34
E	Mia Kim	40
F	Lucas Gupta	33
G	Sophia Zhang	42

DataFrame Slicing and Indexing

Add a New Column to Calculate The Player BMI

$$BMI = \frac{\text{weight (kg)}}{\text{height (m)}^2}$$

```
# Your existing DataFrame
player_list = [['Luna Rodriguez', 36, 75, 185],
               ['Oliver Chen', 38, 74, 165],
               ['Ava Singh', 31, 70, 173],
               ['Ethan Patel', 34, 80, 175],
               ['Mia Kim', 40, 100, 165],
               ['Lucas Gupta', 33, 72, 169],
               ['Sophia Zhang', 42, 85, 175]]

df = pd.DataFrame(player_list, columns=['Name', 'Age', 'Weight', 'Height'])
df
```

	Name	Age	Weight	Height
0	Luna Rodriguez	36	75	185
1	Oliver Chen	38	74	165
2	Ava Singh	31	70	173
3	Ethan Patel	34	80	175
4	Mia Kim	40	100	165
5	Lucas Gupta	33	72	169
6	Sophia Zhang	42	85	175

```
# Adding a new column for Body Mass Index (BMI)
df['BMI'] = df['Weight'] / ((df['Height'] / 100) ** 2)
df
```

	Name	Age	Weight	Height	BMI
0	Luna Rodriguez	36	75	185	21.913806
1	Oliver Chen	38	74	165	27.180900
2	Ava Singh	31	70	173	23.388687
3	Ethan Patel	34	80	175	26.122449
4	Mia Kim	40	100	165	36.730946
5	Lucas Gupta	33	72	169	25.209201
6	Sophia Zhang	42	85	175	27.755102

Creating DataFrames With Time-Series Labels

Working with Time Series

Create a pandas DataFrame using the hourly temperature data from a single day.

```
temp_c = [ 8.0,  7.1,  6.8,  6.4,  6.0,  5.4,
           4.8,  5.0,  21.0, 17.9, 15.5, 14.4]
dt = pd.date_range(start='2024-10-01 00:00:00.0', periods=12, freq='h')
temp = pd.DataFrame(data={'temp_c': temp_c}, index=dt)
temp
```

	temp_c
2024-10-01 00:00:00	8.0
2024-10-01 01:00:00	7.1
2024-10-01 02:00:00	6.8
2024-10-01 03:00:00	6.4
2024-10-01 04:00:00	6.0
2024-10-01 05:00:00	5.4
2024-10-01 06:00:00	4.8
2024-10-01 07:00:00	5.0
2024-10-01 08:00:00	21.0
2024-10-01 09:00:00	17.9
2024-10-01 10:00:00	15.5
2024-10-01 11:00:00	14.4

Working with Time Series

Create a pandas DataFrame using the bi-hourly temperature data from a single day.

```
temp_c = [ 8.0,  7.1,  6.8,  6.4,  6.0,  5.4,
           4.8,  5.0,  21.0, 17.9, 15.5, 14.4]
dt = pd.date_range(start='2024-10-01 00:00:00.0', periods=12, freq='2h')
temp = pd.DataFrame(data={'temp_c': temp_c}, index=dt)
temp
```

	temp_c
2024-10-01 00:00:00	8.0
2024-10-01 02:00:00	7.1
2024-10-01 04:00:00	6.8
2024-10-01 06:00:00	6.4
2024-10-01 08:00:00	6.0
2024-10-01 10:00:00	5.4
2024-10-01 12:00:00	4.8
2024-10-01 14:00:00	5.0
2024-10-01 16:00:00	21.0
2024-10-01 18:00:00	17.9
2024-10-01 20:00:00	15.5
2024-10-01 22:00:00	14.4

Working with Time Series

Create a pandas DataFrame using the average daily temperature data from a single month.

```
temp_c = [ 8.0,  7.1,  6.8,  6.4,  6.0,  5.4,
           4.8,  5.0,  21.0, 17.9, 15.5, 14.4]

# Generate daily timestamps instead of hourly
dt = pd.date_range(start='2024-10-01', periods=12, freq='d')

# Create the DataFrame with daily index
temp = pd.DataFrame(data={'temp_c': temp_c}, index=dt)
temp
```

	temp_c
2024-10-01	8.0
2024-10-02	7.1
2024-10-03	6.8
2024-10-04	6.4
2024-10-05	6.0
2024-10-06	5.4
2024-10-07	4.8
2024-10-08	5.0
2024-10-09	21.0
2024-10-10	17.9
2024-10-11	15.5
2024-10-12	14.4

Available Options for **freq** in pd.date_range()

- Seconds: `S`
- Minutes: `T` or `min`
- Hours: `H`
- Days: `D`
- Weeks: `W`
- Months: `M`
- Quarters: `Q`
- Years: `Y` or `A`

- `2D` : Every 2 days
- `3W` : Every 3 weeks
- `4H` : Every 4 hours
- `1H30T` : Every 1 hour and 30 minutes

Thank You!

NANYANG
THE INNOVATIVE POLYTECHNIC

www.nyp.edu.sg