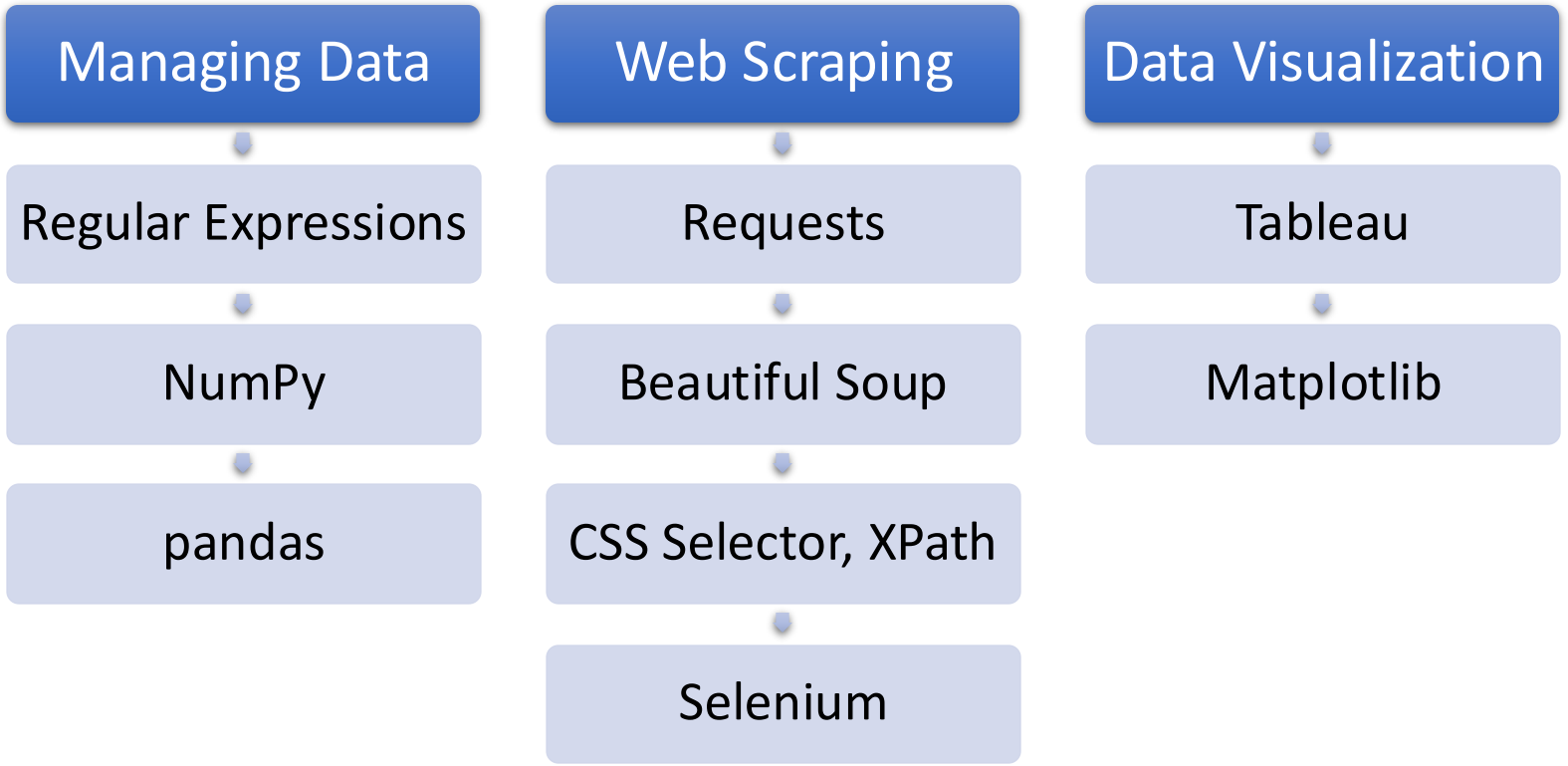
Managing Data III

MSBA7001 Business Intelligence and Analytics
HKU Business School
The University of Hong Kong

Instructor: Dr. DING Chao

Course Roadmap



pandas

The SciPy Ecosystem

It defines numerical array and matrix types

It provides data visualization tools

NumPy

Base N-dimensional array package

SciPy library

Fundamental library for scientific computing

Matplotlib

Comprehensive 2D Plotting

IP[y]: IPython

IPython

Enhanced Interactive Console

Sympy

Symbolic mathematics

pandas

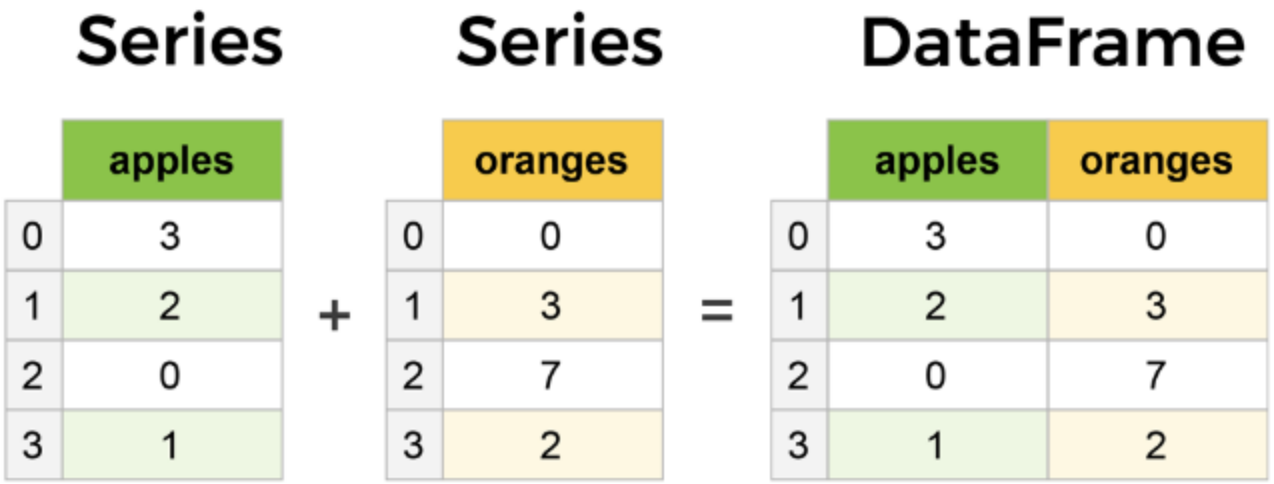
Data structures & analysis

It makes possible Jupyter Notebook

It provides high-performance, easy to use data structures

What is pandas?

- pandas contains high-level data structures and manipulation tools designed to make data analysis fast and easy in Python.
- pandas has two workhorse data structures: Series and DataFrame.



Series

- A Series is a one-dimensional array-like object containing an array of data and an associated array of data labels, called its index.
- We can use the **`Series`** method to create a Series object.
- It works on an array-like objects, dictionaries, and scalar values.
- By default, the index is consisted of integers 0 through n – 1

```
import pandas as pd

obj1 = pd.Series([4, 7, -5, 3])
obj1
```

```
0  4
1  7
2 -5
3  3
dtype: int64
```

Common Operations on Series

Method	Description
s.agg() s.aggregate()	Returns a summary of a Series based on the aggregate functions such as min, max, median, mean, etc
s.apply()	Invokes a python function on values of Series
s.map()	Maps values of a Series according to input correspondence
s.transform()	Transforms the values of a Series based on a function
s.rolling()	Provides rolling window calculations
s.astype(float)	Converts the datatype of a Series to float
s.tolist()	Converts a Series to a list
s.replace(1,'one')	Replaces all values equal to 1 with ‘one’
s.where(s > 10)	Returns a Series and replaces False values with NaN
s.mask(s > 10)	Returns a Series and replaces True values with NaN

Working with Series of String Values

- For columns with string values, we can apply string methods using .str to process the data.
- It is compatible with Regex.
- For example: search for 4 numbers in every cell in the “name” column.

```
import re
name.str.contains(re.compile('\d{4}'))
```

'capitalize', 'casefold', 'cat', 'center', 'contains', 'count', 'decode', 'encode', 'endswith', 'extract', 'extractall', 'find', 'findall', 'fullmatch', 'get', 'get_dummies', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'islower', 'isnumeric', 'isspace', 'istitle', 'isupper', 'join', 'len', 'ljust', 'lower', 'lstrip', 'match', 'normalize', 'pad', 'partition', 'removeprefix', 'removesuffix', 'repeat', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'slice', 'slice_replace', 'split', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'wrap', 'zfill'

DataFrame

- A DataFrame represents a tabular, spreadsheet-like data structure containing an ordered collection of columns, each of which can be a different value type (numeric, string, Boolean, etc.).
- A DataFrame has both a row and column index.
- It can be thought of as a dictionary of Series (one for all sharing the same index).

Creating a DataFrame

- One of the most common way to create a DataFrame is from a dictionary with equal-length lists as its values.
- The resulting DataFrame will have its index assigned automatically as with Series.

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame1 = pd.DataFrame(data)
frame1
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9

Inspecting and Summarizing Data

- After creating/importing a DataFrame, the first thing to do is inspect and understand the data.

Method	Description
df.dtypes, s.dtype	Show the data types of columns
df/s.shape	Number of rows and columns
df/s.index	Show the row index
df/s.columns	Show the column index
df/s.values	Show the values
df/s.info()	Show a concise summary
df/s.count()	Show numbers of non-NaN values
df/s.describe()	Summary statistics for numerical columns
df/s.head(n)	Show first n rows. Default is 5
df/s.tail(n)	Show last n rows. Default it 5

Statistics

Method	Description
df.mean()	Returns the mean of all columns
df.corr()	Returns the correlation between columns in a DataFrame
df.max()	Returns the highest value in each column
df.min()	Returns the lowest value in each column
df.median()	Returns the median of each column
df.std()	Returns the standard deviation of each column

Finding Unique Values in Columns

Method	Description
df/s.value_counts(dropna=False)	View unique values and counts
df/s.unique()	View the unique values
df/s.nunique()	View the count of unique values

```
frame1['state'].unique()
```

```
array(['Ohio', 'Nevada'], dtype=object)
```

```
frame1['state'].value_counts()
```

```
Ohio     3
Nevada   2
Name: state, dtype: int64
```

Cleaning Up DataFrames

Method	Description
df.columns = ['a','b','c']	Renames columns
df.rename()	Renames row or columns index by a function
df.drop()	Deletes row(s) or column(s)
df.drop_duplicates()	Drops all duplicates
df.set_index()	Changes the index
df.reset_index()	Resets the index

- For example, reset the index of the df from 0, delete the original index, and update the df.

```
df.reset_index(drop = True, inplace = True)
```

Selecting Subsets by Rows and Columns

Method	Description
df['col'] or df.col	Selects columns by column labels
df.loc[['row1', 'row2']]	Selects row(s) by row label(s)
df.iloc[0,:]	Selects row(s) by row position(s)
df.iloc[0,0]	Selects value(s) by row position(s) and column position(s)

```
frame1['state']
```

```
0     Ohio
1     Ohio
2     Ohio
3   Nevada
4   Nevada
Name: state, dtype: object
```

```
frame1.iloc[2]
```

```
state    Ohio
year     2002
pop       3.6
Name: 2, dtype: object
```

Selecting Subsets by Applying Filters

Method	Description
df.loc[df[col] > 0.5]	Returns rows where col value is greater than 0.5
df.loc[~(df[col] > 0.5)]	Returns rows where col value is NOT greater than 0.5
df.loc[df[col1] > 0.5 & df[col2]%2 == 0]	Returns rows where col1 value is greater than 0.5 and col2 value is even
df.filter(regex = 'e$')	Returns rows whose labels end with letter e
df.query('col1 > col2')	Returns rows where the condition is True
df.loc[df.col1 > df.col2]	Equivalent to the previous one

```
frame1.loc[(frame1['state'] == 'Ohio') & (frame1['pop'] > 1.5)]
```

	state	year	pop
1	Ohio	2001	1.7
2	Ohio	2002	3.6

Working with Date & Time Values

- Pandas provide a different set of tools using which we can perform all the necessary tasks on date-time data.

Method	Description
pd.to_datetime()	Converts values to a DateTime object.
pd.Timestamp()	Returns a Timestamp object.
pd.date_range()	Returns a fixed frequency DatetimeIndex object.
pd.Period()	Returns a Period object.

```
examtime = pd.Timestamp('2024-09-04 15:30:00')
module1 = pd.date_range(start = '2024/08/30', periods = 10, freq = '4D')
arrival = pd.to_datetime('12-11-2025 15:27', format ='%d-%m-%Y %H:%M')
```

2025-11-12 15:27:00

DatetimeIndex(['2024-08-30', '2024-09-03', '2024-09-07', '2024-09-11',
'2024-09-15', '2024-09-19', '2024-09-23', '2024-09-27',
'2024-10-01', '2024-10-05'],
dtype='datetime64[ns]', freq='4D')

Working with Date & Time Values

- For columns with DateTime objects, we are able to apply DateTime methods using .dt to process the values.
- For example: extract only the years from every cell in the “datetime” column.

```
datetime.dt.year
```

'asfreq', 'ceil', 'components', 'date', 'day', 'day_name', 'day_of_week', 'day_of_year', 'dayofweek', 'dayofyear', 'days', 'days_in_month', 'daysinmonth', 'end_time', 'floor', 'freq', 'hour', 'is_leap_year', 'is_month_end', 'is_month_start', 'is_quarter_end', 'is_quarter_start', 'is_year_end', 'is_year_start', 'isocalendar', 'microsecond', 'microseconds', 'minute', 'month', 'month_name', 'nanosecond', 'nanoseconds', 'normalize', 'quarter', 'qyear', 'round', 'second', 'seconds', 'start_time', 'strftime', 'time', 'timetz', 'to_period', 'to_pydatetime', 'to_pytimedelta', 'to_timestamp', 'total_seconds', 'tz', 'tz_convert', 'tz_localize', 'week', 'weekday', 'weekofyear', 'year'

Handling NaN Values

- Common solutions to deal with missing (NaN) values:
  1. Delete the entire row/column with missing values
  2. Fill missing values with the mean/median/mode
  3. Fill missing values with neighboring values: forward fill vs backward fill
  4. Impute the missing values

Method	Description
df.isnull()	Checks for missing values and returns Boolean results
df.notnull()	The opposite of isnull
df.dropna()	Drops all rows that contain missing values
df.fillna(x)	Replaces missing values with x
df.interpolate(method = 'linear')	Replaces the missing values with linear method

Transforming DataFrames

Method	Description
df.sort_values(col)	Sorts values by column col in ascending order
df.groupby(col)	Returns a groupby object for values from column col
df.resample()	Converts frequency of time series data
df.pivot_table(index=col1,values =[col2,col3],aggfunc=mean)	Creates a pivot table that groups by col1 and calculates the mean of col2 and col3
df.stack()	Pivots a level of column labels
df.unstack()	Pivots a level of index labels
df.apply()	Applies a function along one of the axis of the df
pd.melt()	Gathers columns into rows
pd.crosstab()	Builds a cross-tabulation table of two (or more) factors

Merging DataFrames

Method	Description
df1.append(df2)	Adds the rows in df1 to the end of df2 (columns should be identical)
pd.concat([df1,df2],axis=1)	Adds the columns in df1 to the end of df2 (rows should be identical)
df1.join(df2,on=col1,how='inner')	SQL-style joins the columns in df1 with the columns on df2 where the rows for col have identical values. 'how' can be one of 'left', 'right', 'outer', 'inner'
pd.merge(df1,df2,how='inner', on=col1)	Similar to inner join of SQL

See a comparison here:
https://pandas.pydata.org/docs/user_guide/merging.html

Files I/O

Method	Description
pd.read_csv()	Reads from a CSV file
pd.read_table()	Reads from a delimited text file (like TSV)
pd.read_excel()	Reads from an Excel file
pd.read_json()	Reads from a JSON formatted string, URL or file
pd.read_html()	Parses a URL, string or file and extracts tables to a list of DataFrames
df.to_csv()	Writes a DataFrame to a CSV file
df.to_excel()	Writes a DataFrame to an Excel file
df.to_json()	Writes a DataFrame to a file in JSON format