Managing Data II

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

Instructor: Dr. DING Chao

Agenda

- SciPy
- NumPy
- pandas

SciPy

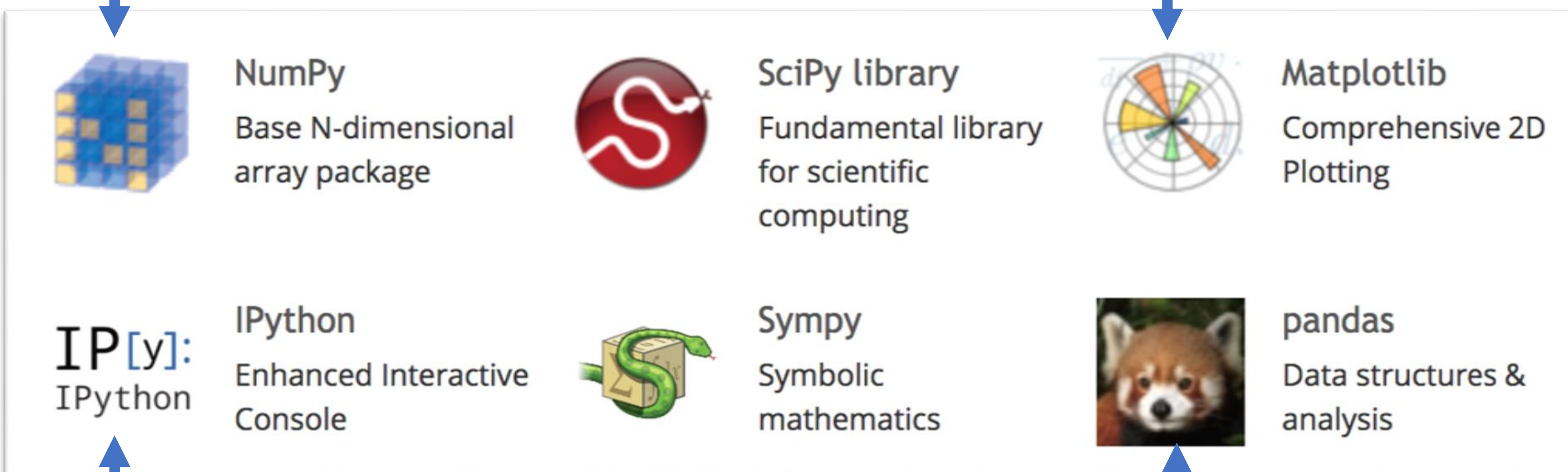
What is SciPy?

- SciPy (pronounced /saɪpaɪ/) is a Python-based ecosystem of open-source software for mathematics, science, and engineering.
- The SciPy ecosystem includes general and specialized tools for data management and computation, productive experimentation and high-performance computing.
- It offers over 1000 modules/packages for Python

The SciPy Ecosystem

It defines numerical array and matrix types

It provides data visualization tools



It makes possible Jupyter Notebook

It provides high-performance, easy to use data structures

NumPy

What is the problem with lists?

- Lists are ok for storing small amounts of one-dimensional data
- But, we can't use them directly with arithmetical operators such as +, -, *, /, …
- Need efficient arrays with arithmetic and better multidimensional tools

What is NumPy?

- **NumPy** (/nʌmpaɪ/), short for Numerical Python, is the fundamental package required for high performance scientific computing and data analysis.
- It provides:
  - Arrays, a fast and space-efficient multidimensional array providing vectorized arithmetic operations and sophisticated broadcasting capabilities
  - Standard mathematical functions for fast operations on entire arrays of data without having to write loops
  - Tools for reading / writing array data to disk and working with memory-mapped files
  - Linear algebra, random number generation, and Fourier transform capabilities

The NumPy Arrays

- A NumPy array (also called ndarray) is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- Typical examples of multidimensional arrays include vectors, matrices, images and spreadsheets.
- Before using NumPy, we need to import the **`numpy`** module

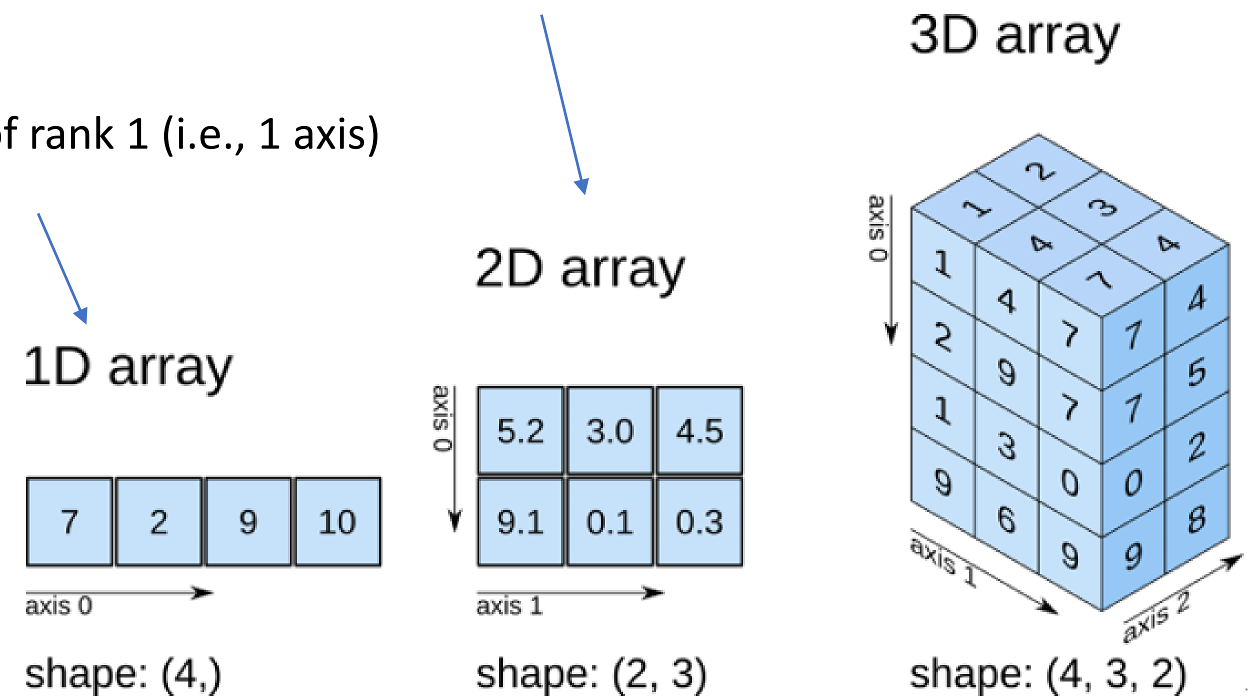
```
import numpy as np
```

The NumPy Arrays

- Dimensions are usually called axes, the number of axes is the rank.

An array of rank 2 (i.e., 2 axes)

An array of rank 1 (i.e., 1 axis)



Creating Arrays

- The easiest way to create an array is to use the **array** method.
- This accepts any sequence-like object (e.g., list, tuple) and produces a new array containing the data passed to it.

```
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1, dtype = np.float32)
arr1
```

array([6. , 7.5, 8. , 0. , 1.])

```
data2= [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2= np.array(data2)
print(arr2)
```

```
[[1 2 3 4]
 [5 6 7 8]]
```

Data Types (dtype)

- In addition to int, float, and str, we can also use the following types.

Data type	Description
np.int64	Signed 64-bit integer types
np.float32	Standard double-precision floating point
np.complex	Complex numbers represented by 128 floats
np.bool	Boolean type storing TRUE and FALSE values
np.object	Python object type
np.string_	Fixed-length string type
np.unicode_	Fixed-length unicode type

Array Properties

Property	Description
arr.size	Returns number of elements in arr
arr.shape	Returns dimensions of arr (rows,columns)
arr.ndim	Returns the dimension of arr
arr.dtype	Returns type of elements in arr
np.info(np.eye)	View documentation for np.eye

Dimension of the array → `arr2.ndim` 2

Structure of the array → `arr2.shape` (2, 4)

Creating Special Arrays

Method	Description
np.zeros(3)	1D array of length 3 all values 0
np.ones((3,4))	3x4 array with all values 1
np.eye(5)	5x5 array of 0 with 1 on diagonal (Identity matrix)
np.empty((2,3,2))	2x3x2 array without initializing its values to any particular value
np.full((2,3),8)	2x3 array with all values 8
np.linspace(0,100,6)	Array of 6 evenly divided values from 0 to 100
np.arange(0,10,3)	Array of values from 0 to less than 10 with step 3

Creating Random Arrays

Method	Description
np.random.rand(4,5)	4x5 array of random floats between 0–1
np.random.rand(6,7)*100	6x7 array of random floats between 0–100
np.random.randint(5,size=(2,3))	2x3 array with random ints between 0–4
np.random.choice([3,5,7,9], size=(3,5))	3x5 array randomly drawn from the list
np.random.randn(5, 3)	5x3 array drawn from a standard normal distribution
np.random.normal(mu, sigma, 10)	1x10 array drawn from a normal distribution

- For a full list of available distributions, see
  https://numpy.org/doc/stable/reference/random/legacy.html

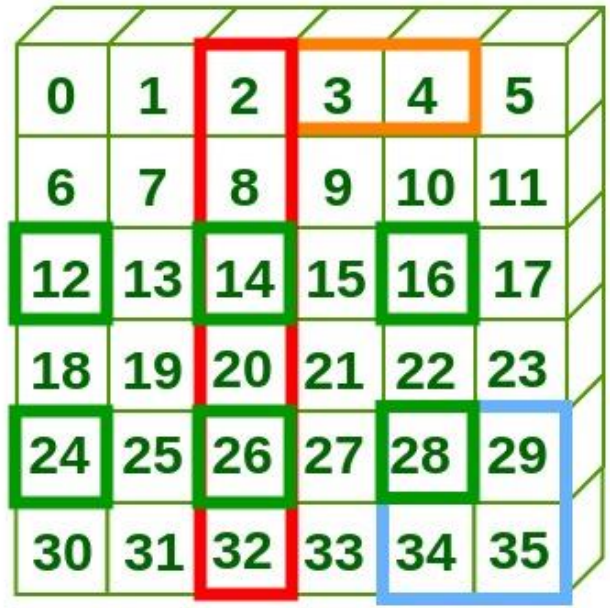
Basic Array Indexing and Slicing

```
>>> a[0,3:5]
array( [3,4] )

>>> a[4:, 4:]
array( [ 28, 29],
       [ 34, 35] ] )

>>> a[ :, 2]
array( [2, 8, 14, 20, 26, 32] )

>>> a[2 : : 2, : : 2]
array( [ 12, 14, 16],
       [ 24, 26, 28] ] )
```



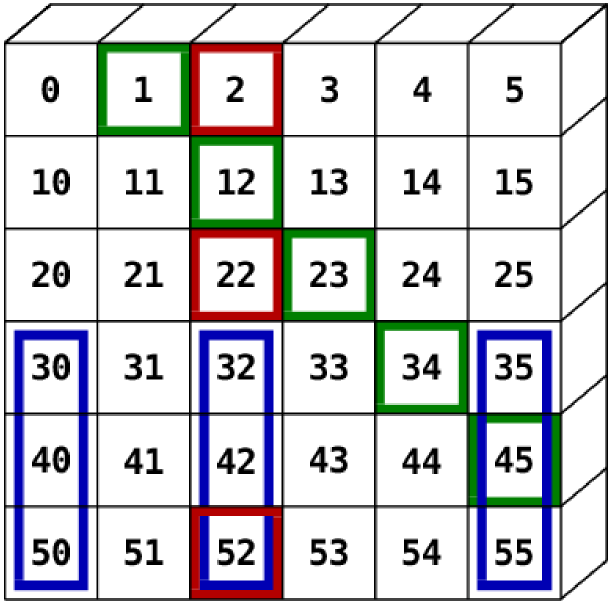
source: www.geeksforgeeks.org

Fancy Indexing

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])

>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



source: www.scipy-lectures.org

Array Operations

Method	Description
np.copy(arr)	Copies arr to new memory
arr.view(dtype)	Creates view of arr elements with type dtype
np.append(arr,values)	Appends values to end of arr
np.insert(arr,2,values)	Inserts values into arr before index 2
np.delete(arr,3,axis=0)	Deletes row on index 3 of arr
np.delete(arr,4,axis=1)	Deletes column on index 4 of arr
np.isnan(arr)	Checks for nan values and returns Boolean results.
arr.fill(value)	Fills the array with scalar values.
arr.astype(np.int64)	Converts arr elements to type np.int64
arr.tolist()	Converts arr to a Python list

Maths

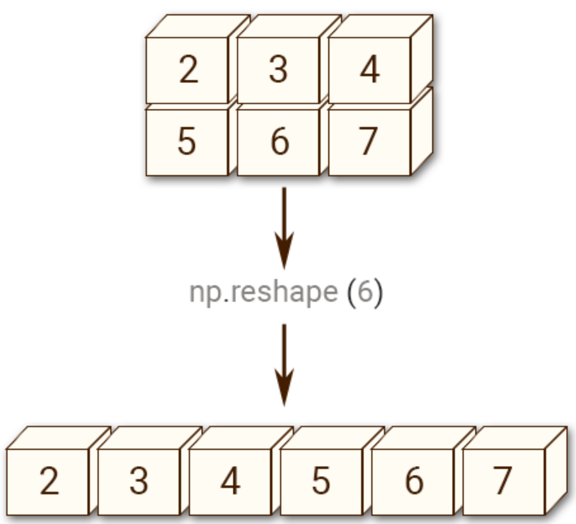
Method	Description
np.add(arr1,arr2)	Elementwise add arr2 to arr1
np.subtract(arr1,arr2)	Elementwise subtract arr2 from arr1
np.multiply(arr1,arr2)	Elementwise multiply arr1 by arr2
np.divide(arr1,arr2)	Elementwise divide arr1 by arr2
np.power(arr1,arr2)	Elementwise raise arr1 raised to the power of arr2
np.array_equal(arr1,arr2)	Returns True if the arrays have the same elements and shape
np.sqrt(arr)	Square root of each element in the array
np.sin(arr)	Sine of each element in the array
np.log(arr)	Natural log of each element in the array
np.abs(arr)	Absolute value of each element in the array
np.round(arr)	Rounds to the nearest int

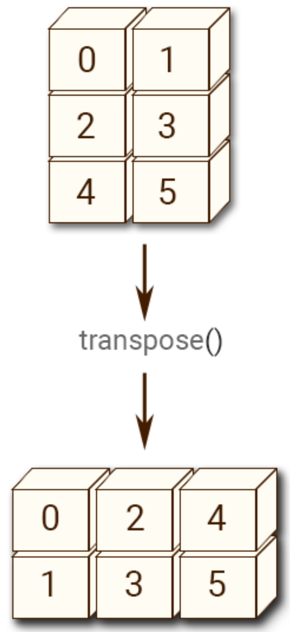
Statistics

Method	Description
arr.mean(arr,axis=0)	Returns mean along specific axis
arr.sum()	Returns sum of arr
arr.min()	Returns minimum value of arr
arr.max(axis=0)	Returns maximum value of specific axis
np.var(arr)	Returns the variance of array
np.std(arr,axis=1)	Returns the standard deviation of specific axis
np.corrcoef(arr1,arr2)	Returns correlation coefficient of arr1 and arr2

Array Transformations

Method	Description
arr.sort(axis=0)	Sorts specific axis of arr
arr.T or arr.transpose()	Transposes arr (rows become columns and vice versa)
arr.reshape(3,4)	Reshapes arr to 3 rows, 4 columns without changing data
arr.ravel()	Flattens the array

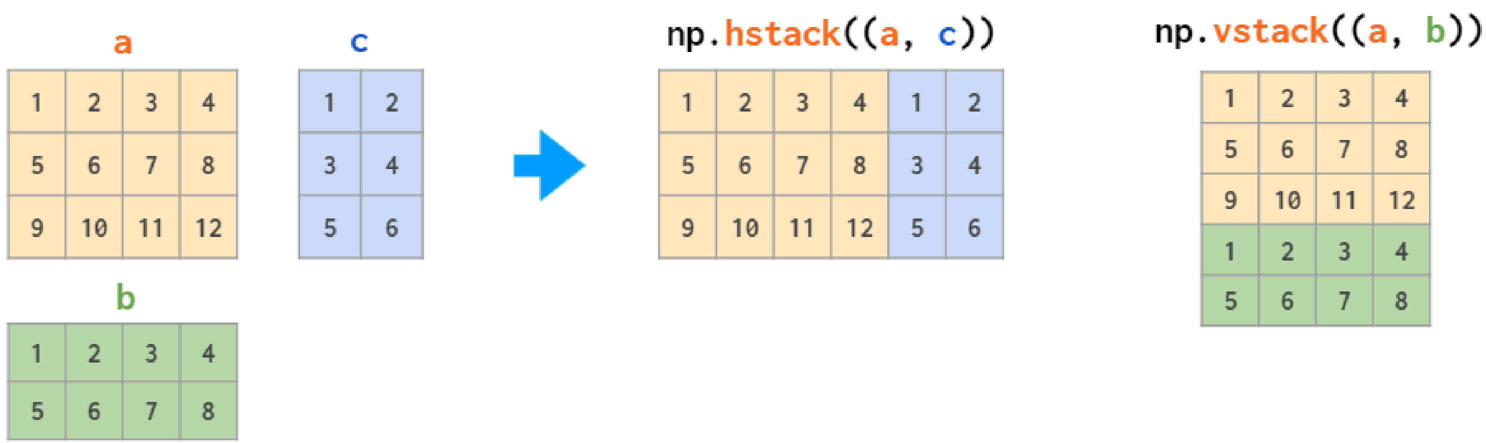




source: www.w3schools.com

Merging and Splitting Arrays

Method	Description
np.concatenate((arr1,arr2),axis=0)	Adds arr2 as rows to the end of arr1
np.concatenate((arr1,arr2),axis=1)	Adds arr2 as columns to end of arr1
np.vstack((arr1,arr2))	Stack arrays in sequence vertically
np.hstack((arr1,arr2))	Stack arrays in sequence horizontally
np.split(arr,3)	Splits arr into 3 sub-arrays
np.hsplit(arr,5)	Splits arr horizontally on the 5th index



File I/O

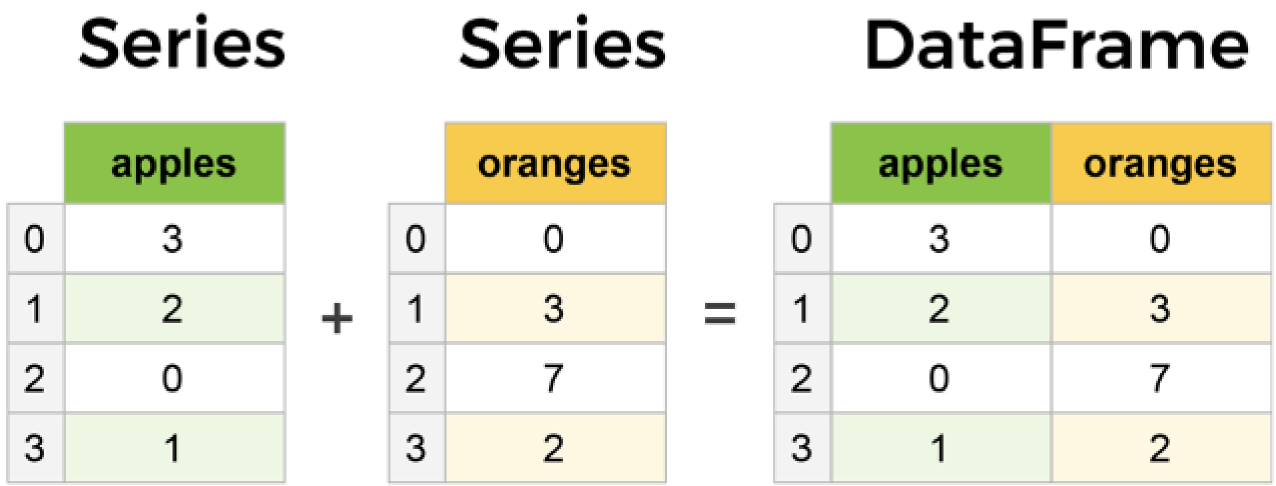
- `loadtxt` method reads text file data into a 2D array.
- `savetxt` method performs the inverse operation: writing an array to a delimited text file.

Method	Description
np.loadtxt()	Read from a text file
np.genfromtxt()	Read from a CSV file
np.savetxt()	Writes to a text file
np.savetxt()	Writes to a CSV file

pandas

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
```

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.).
- The 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
```

Selecting Subsets by Rows and Columns

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

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

	state	year	pop
1	Ohio	2001	1.7
2	Ohio	2002	3.6

Cleaning Data

Method	Description
df.columns = ['a','b','c']	Renames columns
df.rename(columns = lambda x: x+1)	Rename row or columns index by a function
df.drop()	Delete row(s) or column(s)
df.drop_duplicates()	Drops all duplicates
df.set_index('column_one')	Changes the index
df.reset_index()	Resets the index
s.astype(float)	Converts the datatype of the Series to float
s.tolist()	Converts a Series to a list
s.replace(1,'one')	Replaces all values equal to 1 with ‘one’

Working with DateTime

Method	Description
pd.to_datetime()	Converts an argument to DateTime
pd.date_range()	Returns a date range

- For columns with date/time type of data, there is a number of methods to extract information from the data after converting them to DateTime object.

```
dir(pd.Series.dt)
```

'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'

```
df['date'].dt.year
```

Working with Strings

- For columns with text data, we can apply string methods to process the data after converting them to string type.

```
dir(pd.Series.str)
```

'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'

- The best part is that it is compatible with Regex.

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

Handling NaN Values

- The following are 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.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)	Add the rows in df1 to the end of df2 (columns should be identical)
pd.concat([df1,df2],axis=1)	Add the columns in df1 to the end of df2 (rows should be identical)
df1.join(df2,on=col1,how='inner')	SQL-style join 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()	From a CSV file
pd.read_table()	From a delimited text file (like TSV)
pd.read_excel()	From an Excel file
pd.read_json()	Read 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()	Write a DataFrames to a CSV file
df.to_excel()	Write a DataFrames to an Excel file
df.to_json()	Write a DataFrames to a file in JSON format

Before We Move On

Managing Data
- txt, csv, json
- Regular Expressions
- NumPy
- Pandas

Web Scraping
- Requests
- Beautiful Soup
- Selenium
- Using APIs

Data Visualization
- Tableau
- Matplotlib

Install BeautifulSoup 4

- In our next sessions, we need to use a powerful package called BeautifulSoup 4.
- It should be pre-installed with Jupyter Notebook.
- Test the following code to see whether you already have the package.

```
from bs4 import BeautifulSoup
```

- If the code produces error, then follow the instructions on the following page to download and install the package.

https://www.crummy.com/software/BeautifulSoup/