Pandas1-Objects

Course content

Part 1: Basics of programming (W34-40)

- Programming languages
- Programming environment
- Python syntax
- Variables
- Strings and numbers
- Lists
- Conditional statements
- Loop statements
- Functions
- Dictionary, tuple, and set

Part 2: Data extraction and visualization (W41-47)

- Pandas Series and DataFrame
- Read csv files
- Basic statistics
- Data manipulation
- Missing data handling
- Data aggregation
- Data visualization – Pandas
- Data visualization - Matplotlib
- Data visualization - Seaborn
- Time series data

Outline

- Python
  - Modules
  - Packages
  - Class
- Python package: Pandas
  - Pandas data structure
    - Series: Creation, index, subset selection, filter
    - DataFrame: Creation, index, subset selection, filter
  - Read file as a DataFrame
    - View data
    - Subset selection
    - Descriptive statistics

Modules

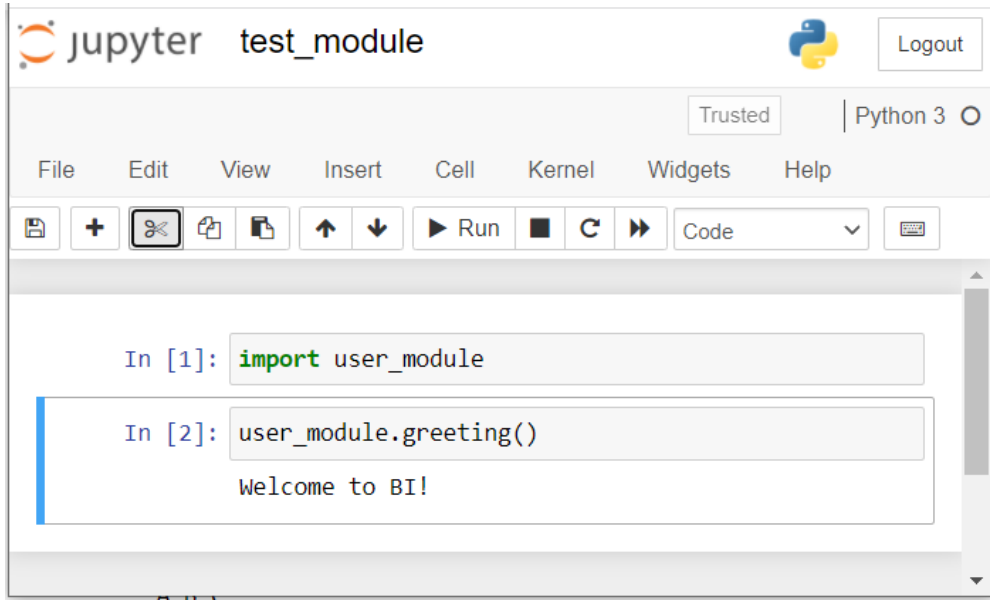
- Modules:
  - A module is a `.py` file with python code. The code can be in the form of variables, functions, or class defined. The filename is the module name.
  - The function or variables present inside the file can be used in another file.
  - To use the module, you can import it using the `import` keyword.

- https://docs.python.org/3/tutorial/modules.html#
- .py is a regular python file. It's plain text and contains just your code.
- .ipynb is an interactive python notebook and it contains the notebook code, the execution results and other internal settings in a specific format.

Modules - create and use a module

Example

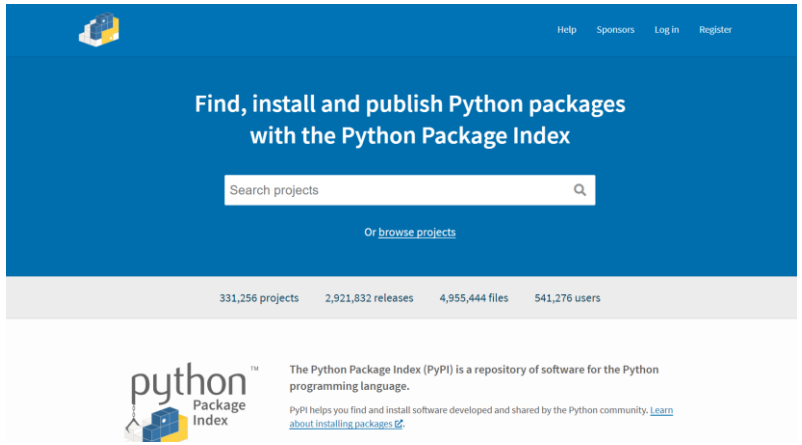
- Step1: Create a **user_module.py** file.
- Step2: Write some functions.
- Step3: Create another file **test_module.ipynb.**
- Step4: `Import` **user_module.**
- Step5: Call the function **greeting()** from the module.



Packages

- Packages are a collection of modules.
- The Python Package Index (PyPI) is a repository of pacakges for the Python programming language.
- **pip** is the package installer for Python. You can use it to install packages from PyPI.

https://pypi.org/



https://docs.python.org/3/tutorial/modules.html#packages

Packages – packages for data analysis

- **NumPy** (Numerical Python)
  - Large multidimensional array operations
- **SciPy** (Scientific Python)
  - Many efficient numerical routines such as routines for numerical integration and optimization
- **Pandas**
  - Data manipulation and data visualization
- **Matplotlib**
  - Data exploration and data visualization
- **Seaborn**
  - High-level data visualization library based on Matplotlib
- **Scikit-learn**
  - Machine learning and statistical modeling

Class

- Python is an object-oriented programming (OOP) language.
- Object-oriented programming is a programming paradigm based on the concept of objects.
- Class
  - A class is a blueprint to create objects.
  - Classes provide a way to bundle data and functions together. Creating a new class creates a new type of object.
- Object
  - An object is an instance of a class.
- Example

```
mylist = [1,2,3]
print(type(mylist))
```

Create a new object of class “list“.

```
<class 'list'>
```

```
mylist.append(4)
```

Functions bound to objects are called methods.

- https://www.geeksforgeeks.org/difference-between-class-and-object/

Class

- You can define your own classes.
  - Attributes: Variables of a class.
  - Methods: Functions of a class.

```python
# Define a class
class Student:
    def __init__(self, school, name):
        self.school = school
        self.name = name

    def greeting(self):
        print("Hi, I'am a student at {}.".format(self.school))
```

```python
# Create objects
student1 = Student("BI", "Anna")
student2 = Student("BI", "Lucas")
```

```python
print(type(student1))
```

```
<class '__main__.Student'>
```

```python
# Call the method
student1.greeting()
```

```
Hi, I'am a student at BI.
```

```python
# Access the attribute
student1.name
```

```
'Anna'
```


- https://www.w3schools.com/python/python_classes.asp
- https://docs.python.org/3/tutorial/classes.html

Pandas

Pandas

- Pandas
  - An open-source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- Installation
  - Installing pandas from a jupyter notebook

```
pip install pandas
```

  - Installing pandas from Anaconda Navigator
    - https://docs.anaconda.com/anaconda/navigator/tutorials/pandas/
- Import package
  - The "`as pd`" part means that we can write the shorthand "pd" instead of "pandas" when we use it later. In principle, you could use other names than "pd", but this is the common naming convention.

```
import pandas as pd
```


https://pandas.pydata.org/pandas-docs/stable/user_guide/cookbook.html#cookbook

Data structures

- **Python list**
  - A data structure that can store values of different data types.

0	'Spam'
1	2.0
2	5
3	'A'
4	[10,20]

- **NumPy array**:
  - A data structure that stores values of the same data type.
  - NumPy uses arrays to perform a wide variety of mathematical operations.

0	'B'
1	'C'
2	'C'
3	'A'
4	'B'

0	2.0
1	1.5
2	3.2
3	4.9
4	2.5

Pandas data structures

- **Pandas data structures**
  - **Series class:** A one-dimensional array-like structure.
    - A series contains an array of data and an associated array of index.

0	216
1	143
2	98
3	455
4	108

  - **DataFrame class:** A tabular, spreadsheet-like structure.
    - A DataFrame contains an ordered collection of columns, each of which can be a different data type (numeric, string, boolean, etc.).
    - A DataFrame has both a row and column index.

	High	Width	weight	group
0	20	20	0.1	A
1	45	30	0.8	C
2	54	43	1.5	C
3	25	15	2.3	B
4	18	34	0.2	A

DataFrame

	High
0	20
1	45
2	54
3	25
4	18

Series

	Width
0	20
1	30
2	43
3	15
4	34

Series

	weight
0	0.1
1	0.8
2	1.5
3	2.3
4	0.2

Series

	group
0	A
1	C
2	C
3	B
4	A

Series

Series - creation

- Form a series from an array of data

```
Series1 = pd.Series([4, 7, -5, 3])
Series1
```

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

index

- Get values and index

```
Series1.values
```

```
array([ 4,  7, -5,  3], dtype=int64)
```

The values are simply a NumPy array.

```
Series1.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

RangeIndex is the default index type used by Series and DataFrame when no explicit index is provided by the user.

Series - index

- Assign index

```
Series2 = pd.Series([4, 7, -5, 3], index = ["a","b","c","d"])
Series2
```

```
a    4
b    7
c   -5
d    3
dtype: int64
```

- Get index

```
Series2.index
```

```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

https://pandas.pydata.org/docs/reference/api/pandas.Index.html

Series – subset selection

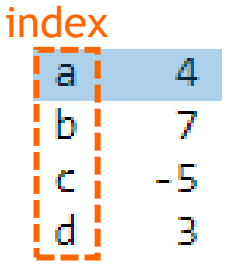
```
Series2 = pd.Series([4, 7, -5, 3], index = ["a","b","c","d"])
Series2
```

```
a    4
b    7
c   -5
d    3
dtype: int64
```

- Select a single value

```
Series2['a']
```

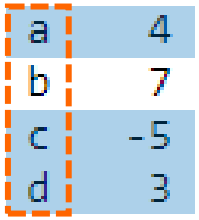
```
4
```



- Select a set of values

```
Series2[['c', 'a', 'd']]
```

```
c   -5
a    4
d    3
dtype: int64
```



Series - filter

- Use Boolean array to filter data.

```
Series2 > 2
```

```
a     True
b     True
c    False
d     True
dtype: bool
```

Boolean array

		>2
a	4	True
b	7	True
c	-5	False
d	3	True

```
Series2[Series2 > 2]
```

```
a    4
b    7
d    3
dtype: int64
```

Recall – logical operators

- Logical operators are used to combine boolean constraints.

Logical operator	Meaning
and	and
or	or
not	not

```
x = 2
y = 5
```

```
x == 2 and y == 5
```

```
True
```

```
x < 0 and y == 5
```

```
False
```

```
x < 0 or y == 5
```

```
True
```

Series - filter

- Pandas uses bitwise operators to combine conditions.

Bitwise operator	Meaning
&	and
|	or
~	not

```
(Series2 > 2) & (Series2 < 5)
```

```
a     True
b    False
c    False
d     True
dtype: bool
```

		>2		<5		
a	4	True		True		True
b	7	True	&	False	➡	False
c	-5	False		False		False
d	3	True		True		True

```
Series2[(Series2 > 2) & (Series2 < 5)]
```

```
a    4
d    3
dtype: int64
```

a	4	True
b	7	False
c	-5	False
d	3	True

https://www.geeksforgeeks.org/python-bitwise-operators/

Exercise

(A.1) Create a series with `score_list` as the value and `ID_list` as the index.

```
ID_list = ['S01', 'S02', 'S03', 'S04', 'S05']
score_list = [7.0, 5.5, 9.0, 5.0, 7.5]
```

(A.2) Select the data of students `'S01'` and `'S03'`.

(A.3) Select students with a score less than 6.

DataFrame - creation

- DataFrame: A tabular, spreadsheet-like structure.
- Form a DataFrame from a dictionary of equal-length lists.

```
data = {"state": ["Ohio","Ohio","Ohio","Nevada","Nevada","Nevada"],
        "year":[2000,2001,2002,2001,2002,2003],
        "pop":[1.5,1.7,3.6,2.4,2.9,3.2]}
```

```
df = pd.DataFrame(data, index = ["a","b","c","d","e","f"])
df
```

	state	year	pop
a	Ohio	2000	1.5
b	Ohio	2001	1.7
c	Ohio	2002	3.6
d	Nevada	2001	2.4
e	Nevada	2002	2.9
f	Nevada	2003	3.2

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

DataFrame – labels and integer positions

- Axis -labels

```
df.index
```

```
Index(['a', 'b', 'c', 'd', 'e', 'f'], dtype='object')
```

```
df.columns
```

```
Index(['state', 'year', 'pop'], dtype='object')
```

	state	year	pop
a	Ohio	2000	1.5
b	Ohio	2001	1.7
c	Ohio	2002	3.6
d	Nevada	2001	2.4
e	Nevada	2002	2.9
f	Nevada	2003	3.2

Columns (label)

Index (label)

- Integer positions

	0	1	2
	state	year	pop
0	a	Ohio	2000	1.5
1	b	Ohio	2001	1.7
2	c	Ohio	2002	3.6
3	d	Nevada	[3,1]	2.4
4	e	Nevada	2002	2.9
5	f	Nevada	2003	3.2

Integer position

DataFrame – subset selection (1) by columns and rows

	state	year	pop
a	Ohio	2000	1.5
b	Ohio	2001	1.7
c	Ohio	2002	3.6
d	Nevada	2001	2.4
e	Nevada	2002	2.9
f	Nevada	2003	3.2

- Using axis-lable (**loc**).

```
df.loc[["b","c","d"], ["state","year"]]
```

	state	year
b	Ohio	2001
c	Ohio	2002
d	Nevada	2001

- Using integer position (**iloc**).

```
df.iloc[1:4, 0:2]
```

	state	year
b	Ohio	2001
c	Ohio	2002
d	Nevada	2001

DataFrame – subset selection (2) by columns or rows

	state	year	pop
a	Ohio	2000	1.5
b	Ohio	2001	1.7
c	Ohio	2002	3.6
d	Nevada	2001	2.4
e	Nevada	2002	2.9
f	Nevada	2003	3.2

- Using column names to select columns

```
df[["state","year"]]   # same as df.loc[:,["state","year"]]
```

	state	year
a	Ohio	2000
b	Ohio	2001
c	Ohio	2002
d	Nevada	2001
e	Nevada	2002
f	Nevada	2003

		state	year	pop
0	a	Ohio	2000	1.5
1	b	Ohio	2001	1.7
2	c	Ohio	2002	3.6
3	d	Nevada	2001	2.4
4	e	Nevada	2002	2.9
5	f	Nevada	2003	3.2

- Using integer positions to select rows

```
df[1:4]                # same as df.iloc[1:4,:]
```

	state	year	pop
b	Ohio	2001	1.7
c	Ohio	2002	3.6
d	Nevada	2001	2.4

DataFrame - filter

- Use Boolean array to filter data.

```
df[df.year > 2001]
```

	state	year	pop
2	Ohio	2002	3.6
4	Nevada	2002	2.9
5	Nevada	2003	3.2

```
df[df.state == "Ohio"]
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6

	state	year	pop	df.year>2001
0	Ohio	2000	1.5	False
1	Ohio	2001	1.7	False
2	Ohio	2002	3.6	True
3	Nevada	2001	2.4	False
4	Nevada	2002	2.9	True
5	Nevada	2003	3.2	True

	state	year	pop	df.state=Ohio
0	Ohio	2000	1.5	True
1	Ohio	2001	1.7	True
2	Ohio	2002	3.6	True
3	Nevada	2001	2.4	False
4	Nevada	2002	2.9	False
5	Nevada	2003	3.2	False

DataFrame - filter

- Use bitwise operators to combine conditions

```
df[(df.state == "Ohio") & (df.year > 2000)]
```

	state	year	pop
b	Ohio	2001	1.7
c	Ohio	2002	3.6

	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
5	Nevada	2003	3.2

Exercise

(B.1) Create a dictionary with `company_name, profit, assets` as the keys and `list1, list2, list3` as the values. Print out the dictionary.

```
list1 = ['JPMorgan Chase','Apple','Bank of America','Amazon','Microsoft']
list2 = [40.4, 63.9, 17.9, 21.3, 51.3]
list3 = [3689.3, 354.1, 2832.2, 321.2, 304.1]
```

(B.2) Create a dataframe named `mydf` based on the dictionary defined in (B.1) and use `a, b, c, d, e` as index. Print out the dataframe.

(B.3) Use `loc` to select a subset as follows.

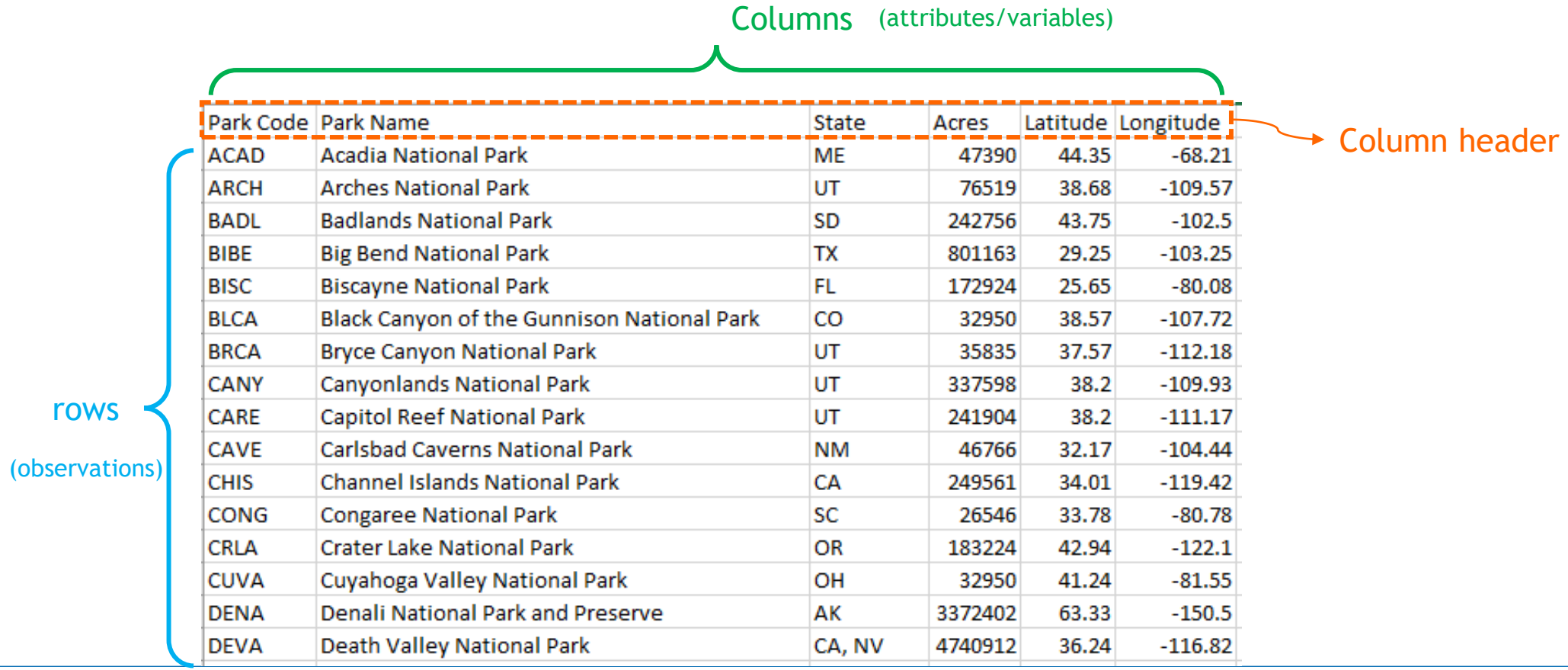
company name	assets
Apple	354.1
Amazon	321.2
Microsoft	304.1

(B.4) Use `iloc` to select the same subset.

Read data from file to DataFrame

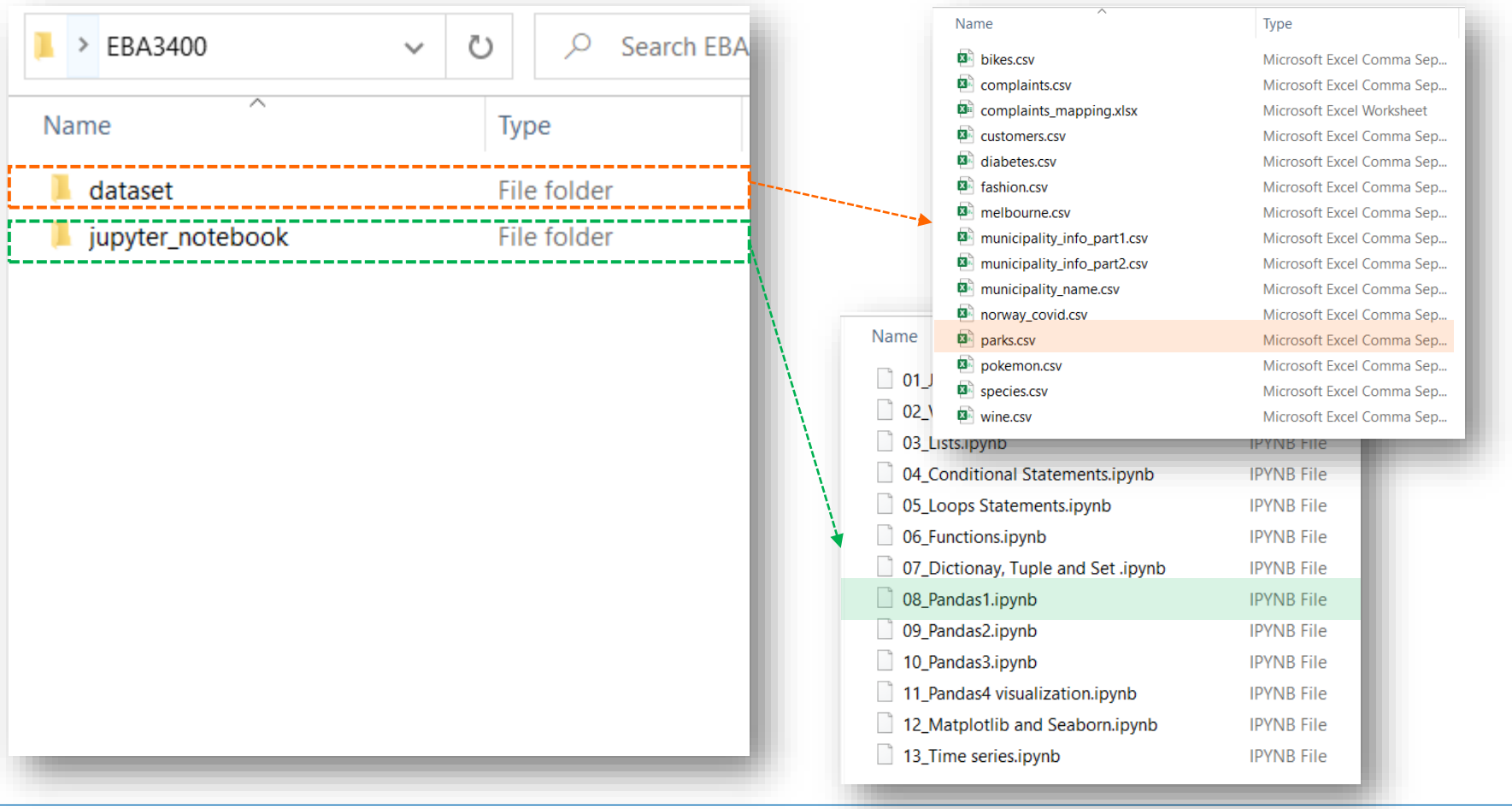
Dataset

- A dataset is a collection of data with a defined structure.



Park Code	Park Name	State	Acres	Latitude	Longitude
ACAD	Acadia National Park	ME	47390	44.35	-68.21
ARCH	Arches National Park	UT	76519	38.68	-109.57
BADL	Badlands National Park	SD	242756	43.75	-102.5
BIBE	Big Bend National Park	TX	801163	29.25	-103.25
BISC	Biscayne National Park	FL	172924	25.65	-80.08
BLCA	Black Canyon of the Gunnison National Park	CO	32950	38.57	-107.72
BRCA	Bryce Canyon National Park	UT	35835	37.57	-112.18
CANY	Canyonlands National Park	UT	337598	38.2	-109.93
CARE	Capitol Reef National Park	UT	241904	38.2	-111.17
CAVE	Carlsbad Caverns National Park	NM	46766	32.17	-104.44
CHIS	Channel Islands National Park	CA	249561	34.01	-119.42
CONG	Congaree National Park	SC	26546	33.78	-80.78
CRLA	Crater Lake National Park	OR	183224	42.94	-122.1
CUVA	Cuyahoga Valley National Park	OH	32950	41.24	-81.55
DENA	Denali National Park and Preserve	AK	3372402	63.33	-150.5
DEVA	Death Valley National Park	CA, NV	4740912	36.24	-116.82

Download the dataset from itslearning



Read csv file

- Dataset: parks.csv (US national park data)
  - Variables: Park code, park name, state, acres, longitude and latitude.
- Use the `read_csv` to read a csv file into dataframe.

```
park_df = pd.read_csv('../dataset/parks.csv')
```

NOTE 1: You will have to specify the location of the file on your computer. The location is relative to where you will find this notebook itself. In the case above, the file "parks.csv" is stored one folder back (../), then inside the folder “dataset".

NOTE 2: A .csv file ("comma separated value") is a file where the data is stored in a text format, separated by commas (or other separation marks such as space or semi-colon).

View data

- View the first N rows

```
park_df.head(5)
```

	Park Code	Park Name	State	Acres	Latitude	Longitude
0	ACAD	Acadia National Park	ME	47390	44.35	-68.21
1	ARCH	Arches National Park	UT	76519	38.68	-109.57
2	BADL	Badlands National Park	SD	242756	43.75	-102.50
3	BIBE	Big Bend National Park	TX	801163	29.25	-103.25
4	BISC	Biscayne National Park	FL	172924	25.65	-80.08

- Get the number of rows and columns

```
park_df.shape
```

```
(56, 6)
```

number of rows

number of columns

Subset selection

- Using column names

```
park_df[["Park Code", "State"]]
```

	Park Code	State
0	ACAD	ME
1	ARCH	UT
2	BADL	SD
3	BIBE	TX
4	BISC	FL
5	BLCA	CO
6	BRCA	UT
7	CANY	UT
8	CARE	UT
9	CAVE	NM
10	CHIS	CA

- Using integer positions

```
park_df[10:15]
```

	Park Code	Park Name	State	Acres	Latitude	Longitude
10	CHIS	Channel Islands National Park	CA	249561	34.01	-119.42
11	CONG	Congaree National Park	SC	26546	33.78	-80.78
12	CRLA	Crater Lake National Park	OR	183224	42.94	-122.10
13	CUVA	Cuyahoga Valley National Park	OH	32950	41.24	-81.55
14	DENA	Denali National Park and Preserve	AK	3372402	63.33	-150.50

Subset selection – loc and iloc

- Using axis-lable (`loc`).

```
park_df.loc[[10,11,12,13,14],["Park Code","Park Name"]]
```

	Park Code	Park Name
10	CHIS	Channel Islands National Park
11	CONG	Congaree National Park
12	CRLA	Crater Lake National Park
13	CUVA	Cuyahoga Valley National Park
14	DENA	Denali National Park and Preserve

- Using integer position (`iloc`).

```
park_df.iloc[10:15,0:2]
```

	Park Code	Park Name
10	CHIS	Channel Islands National Park
11	CONG	Congaree National Park
12	CRLA	Crater Lake National Park
13	CUVA	Cuyahoga Valley National Park
14	DENA	Denali National Park and Preserve

Filter

- One condition

```
park_df[park_df.State == "UT"]
```

	Park Code	Park Name	State	Acres	Latitude	Longitude
1	ARCH	Arches National Park	UT	76519	38.68	-109.57
6	BRCA	Bryce Canyon National Park	UT	35835	37.57	-112.18
7	CANY	Canyonlands National Park	UT	337598	38.20	-109.93
8	CARE	Capitol Reef National Park	UT	241904	38.20	-111.17
55	ZION	Zion National Park	UT	146598	37.30	-113.05

- Multiple conditions

```
park_df[(park_df.State == "UT") & (park_df.Acres > 50000)]
```

	Park Code	Park Name	State	Acres	Latitude	Longitude
1	ARCH	Arches National Park	UT	76519	38.68	-109.57
7	CANY	Canyonlands National Park	UT	337598	38.20	-109.93
8	CARE	Capitol Reef National Park	UT	241904	38.20	-111.17
55	ZION	Zion National Park	UT	146598	37.30	-113.05

Filter

- If we want to evaluate many "or" expressions, we can use `isin`.

```
park_df[park_df.State.isin(['WA', 'OR', 'CA'])]
```

	Park Code	Park Name	State	Acres	Latitude	Longitude
10	CHIS	Channel Islands National Park	CA	249561	34.01	-119.42
12	CRLA	Crater Lake National Park	OR	183224	42.94	-122.10
31	JOTR	Joshua Tree National Park	CA	789745	33.79	-115.90
36	LAVO	Lassen Volcanic National Park	CA	106372	40.49	-121.51
39	MORA	Mount Rainier National Park	WA	235625	46.85	-121.75
40	NOCA	North Cascades National Park	WA	504781	48.70	-121.20
41	OLYM	Olympic National Park	WA	922651	47.97	-123.50
43	PINN	Pinnacles National Park	CA	26606	36.48	-121.16
44	REDW	Redwood National Park	CA	112512	41.30	-124.00
47	SEKI	Sequoia and Kings Canyon National Parks	CA	865952	36.43	-118.68
54	YOSE	Yosemite National Park	CA	761266	37.83	-119.50

Same as

```
park_df[(park_df.State =='WA')|(park_df.State =='OR')|(park_df.State == 'CA')]
```

Create a subset

- Assign the returned dataframe to a new variable.

```
park_west_df = park_df[park_df.State.isin(['WA', 'OR', 'CA'])]
park_west_df
```

	Park Code	Park Name	State	Acres	Latitude	Longitude
10	CHIS	Channel Islands National Park	CA	249561	34.01	-119.42
12	CRLA	Crater Lake National Park	OR	183224	42.94	-122.10
31	JOTR	Joshua Tree National Park	CA	789745	33.79	-115.90
36	LAVO	Lassen Volcanic National Park	CA	106372	40.49	-121.51
39	MORA	Mount Rainier National Park	WA	235625	46.85	-121.75
40	NOCA	North Cascades National Park	WA	504781	48.70	-121.20
41	OLYM	Olympic National Park	WA	922651	47.97	-123.50
43	PINN	Pinnacles National Park	CA	26606	36.48	-121.16
44	REDW	Redwood National Park	CA	112512	41.30	-124.00
47	SEKI	Sequoia and Kings Canyon National Parks	CA	865952	36.43	-118.68
54	YOSE	Yosemite National Park	CA	761266	37.83	-119.50

Exercise

(C.1) Read the csv file `diabetes.csv` using pandas. Display the first 10 rows.

(C.2) What is the number of rows and columns in this data set?

(C.3) Select (display) column `BloodPressure` and column `BMI`.

(C.4) Select rows with `BMI` greater than 50.

(C.5) Select rows with either `BMI` greater than 50 or `BloodPressure` greater than 110.

Descriptive statistics - info

- A summary of a DataFrame
  - Index, data type of each columns, number of non-null values, and memory usage.

```
park_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56 entries, 0 to 55
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Park Code   56 non-null     object 
 1   Park Name   56 non-null     object 
 2   State       56 non-null     object 
 3   Acres       56 non-null     int64  
 4   Latitude    56 non-null     float64
 5   Longitude   56 non-null     float64
dtypes: float64(2), int64(1), object(3)
memory usage: 2.8+ KB
```

In Pandas, string data is always stored with an object dtype.

Descriptive statistics - pandas data types

- Pandas `dtype` mapping

Pandas dtype	Python type	Usage
int64	int	Integer numbers
float64	float	Floating point numbers
object	str or mixed	Text or mixed numeric and non-numeric values
bool	bool	True/False values
datetime64	datetime	Date and time values
timedelta[ns]	--	Differences between two datetimes
category	--	Finite list of text values

numerical data (quantitative)

categorical data (qualitative data)

Descriptive statistics – categorical data and numerical data

- **Categorical data** describes characteristics or groups.
  - Gender (Male, Female)
  - Product types (Wood, Plastic, Metal)
  - Risk level (Low, Medium, High)
  - Days of the week (Monday, Tuesday, Wednesday)
  - Months of the year (January, February, March)

- **Numerical data** expresses information in the form of numbers.
  - Number of customers (25, 19, 35,...)
  - Prices of products (200, 150, 80,...)
  - Interest rate (0.5, 1.4, 2.3,...)

Descriptive statistics – change data type

- Use `astype()` to change the data type (dtype).

```
park_df.Acres = park_df.Acres.astype(float)
park_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56 entries, 0 to 55
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Park Code   56 non-null     object 
 1   Park Name   56 non-null     object 
 2   State       56 non-null     object 
 3   Acres       56 non-null     float64
 4   Latitude    56 non-null     float64
 5   Longitude   56 non-null     float64
dtypes: float64(3), object(3)
memory usage: 2.8+ KB
```

Descriptive statistics - describe

- Descriptive statistics of numerical columns.

```
park_df.describe()
```

	Acres	Latitude	Longitude
count	5.600000e+01	56.000000	56.000000
mean	9.279291e+05	41.233929	-113.234821
std	1.709258e+06	10.908831	22.440287
min	5.550000e+03	19.380000	-159.280000
25%	6.901050e+04	35.527500	-121.570000
50%	2.387645e+05	38.550000	-110.985000
75%	8.173602e+05	46.880000	-103.400000
max	8.323148e+06	67.780000	-68.210000

- https://en.wikipedia.org/wiki/Standard_deviation
- https://en.wikipedia.org/wiki/Quartile
- https://en.wikipedia.org/wiki/Scientific_notation

Descriptive statistics – value_counts

- Count of unique values in a column.

```
df.State.value_counts()
```

```
AK            8
CA            7
UT            5
CO            4
WA            3
FL            3
AZ            3
SD            2
TX            2
HI            2
NM            1
WY, MT, ID    1
OR            1
KY            1
SC            1
MT            1
WY            1
AR            1
ME            1
CA, NV        1
VA            1
NV            1
MN            1
ND            1
OH            1
TN, NC        1
MI            1
Name: State, dtype: int64
```

Exercise

(D.1) Use the same dataset `diabetes.csv` in (C.1). What is the data type of each variable?

(D.2) Change the data type of `Outcome` to `object`.

(D.3) Get the average of variable `Age`.

(D.4) Get the value count of variable `Outcome`.