Pandas 2 – Data Preprocessing

Data analysis process

Data Understanding	•Descriptive statistics •Types of data (numerical/categorical)
Data Preprocessing	•Subset selection •Data consolidation •Missing data handling
Calculation (Modeling)	•Derived variables •Data aggregation
Data Visualization	•Univariate chart •Bivariate chart •Multivariate chart

Outline

- Subset selection
  - Series: index, filter
  - DataFrame: label, integer position, filter

- Data consolidation
  - Concatenate
  - Merge

Subset selection

Series – subset selection

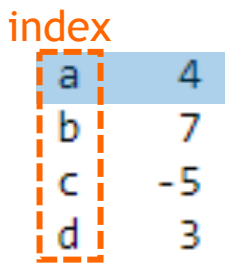
```
s1 = pd.Series([4, 7, -5, 3], index = ["a", "b", "c", "d"])
s1
```

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

- Select a single value

```
s1["a"]
```

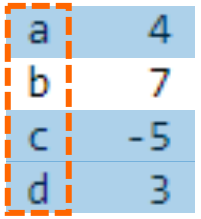
```
4
```



- Select a set of values

```
s1[["c", "a", "d"]]
```

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



Series - filter

- Use Boolean array to filter data.

```
s1 > 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

```
s1[s1 > 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

```
(s1 > 2) & (s1 < 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

```
s1[(s1 > 2) & (s1 < 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

Exercise.A

(A.1) Given the following pandas series representing students' scores, where the index corresponds to their student IDs. Show the scores of both student `S01` and `S03`.

```
scores = pd.Series([7.0, 5.5, 9.0, 5.0, 7.5], index = ['S01', 'S02', 'S03', 'S04', 'S05'])
scores
```

```
S01    7.0
S02    5.5
S03    9.0
S04    5.0
S05    7.5
dtype: float64
```

```

```

(A.2) Using the same series in (A.1), display the data for students who scored less than 6.

```

```

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) loc and iloc

	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-label (**loc**).

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

	state	year
b	Ohio	2001
c	Ohio	2002
d	Nevada	2001

```
df.loc[ row_labels, columns_labels ]
```

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

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

	state	year
b	Ohio	2001
c	Ohio	2002
d	Nevada	2001

```
df.iloc[ row_positions, column_positions ]
```

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 – subset selection (3) filter by condition

- 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

	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

```
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.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 - subset selection (3) filter by multiple conditions

- 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

Exercise.B

(B.1) Read the csv file `diabetes.csv` using pandas. Display the first 5 rows.

(B.2) Select (display) column `BloodPressure` and column `BMI` .

(B.3) Select rows with `BMI` greater than 50.

(B.4) Select the rows where either the `BMI` is greater than 50 or the `BloodPressure` is greater than 110.

More on subset selection

- RangeIndex

```
df
```

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

integer position | label

- Using axis-lable (`loc`).

```
df.loc[[0, 1, 2], : ]
```

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

- Using integer position (`iloc`).

```
df.iloc[0:3, : ]
```

	state	year	pop
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

Data consolidation - Concatenate & Merge

Concatenate - along rows

- Suppose you have two DatafFrames with the same columns and indexes.

```
df1 = pd.DataFrame({"col1":[1,2,3],"col2":[4,5,6],"col3":[7,8,9]}, index = ['a','b','c'])
df2 = pd.DataFrame({"col1":[11,22,33],"col2":[44,55,66],"col3":[77,88,99]},index = ['a','b','c'])
```

df1

	col1	col2	col3
a	1	4	7
b	2	5	8
c	3	6	9

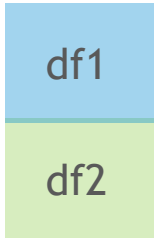
df2

	col1	col2	col3
a	11	44	77
b	22	55	88
c	33	66	99

- Concatenate DataFrames

```
pd.concat([df1,df2])
```
by default, axis = 0

	col1	col2	col3
a	1	4	7
b	2	5	8
c	3	6	9
a	11	44	77
b	22	55	88
c	33	66	99



- Ignore index

```
pd.concat([df1,df2], ignore_index = True)
```

	col1	col2	col3
0	1	4	7
1	2	5	8
2	3	6	9
3	11	44	77
4	22	55	88
5	33	66	99

- axis=0 (or axis = ”rows”) refers to operations along rows (vertical axis).
- axis=1 (or axis = "columns”) refers to operations along columns (horizontal axis).

Concatenate - along columns

- If you pass `axis = 1`, df1 and df2 will be concatenated along the columns.

```
pd.concat([df1,df2], axis = 1)
```

	col1	col2	col3	col1	col2	col3
a	1	4	7	11	44	77
b	2	5	8	22	55	88
c	3	6	9	33	66	99

Concatenate

- Two DatafFrames with different columns.

df3

	col1	col2	col3
a	1	4	7
b	2	5	8
c	3	6	9

df4

	col2	col3	col4
a	11	44	77
b	22	55	88
c	33	66	99

```
pd.concat([df3, df4], axis = 0)
```

	col1	col2	col3	col4
a	1.0	4	7	NaN
b	2.0	5	8	NaN
c	3.0	6	9	NaN
a	NaN	11	44	77.0
b	NaN	22	55	88.0
c	NaN	33	66	99.0

- Two DatafFrames with different indexes.

df5

	col1	col2	col3
a	1	4	7
b	2	5	8
c	3	6	9

df6

	col1	col2	col3
b	11	44	77
c	22	55	88
d	33	66	99

```
pd.concat([df5, df6], axis = 1)
```

	col1	col2	col3	col1	col2	col3
a	1.0	4.0	7.0	NaN	NaN	NaN
b	2.0	5.0	8.0	11.0	44.0	77.0
c	3.0	6.0	9.0	22.0	55.0	88.0
d	NaN	NaN	NaN	33.0	66.0	99.0

- Pandas uses **NaN** (Not a Number) to indicate missing data.

Exercise

Exercise.C

(C.1) Import the datasets `municipality_info_part1.csv` and `municipality_info_part2.csv` as dataframes. The columns in the two datasets are described as follows. Display the first five rows of each dataframe.

- Municipality_number (object)
- Population (int)
- Area (float)

Note: Use the parameter "dtype" to specify the data types.
`dtype = {"Municipality_number": object, "Population": int,  "Area": float}` .

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Municipality_number  50 non-null     int64
 1   Population           50 non-null     int64
 2   Area                 50 non-null     float64
dtypes: float64(1), int64(2)
memory usage: 1.3 KB
```

(C.2) How many rows are there in each dataframe?

(C.3) Concatenate two dataframes in (C.1) along the rows and assign the returned dataframe to a new variable named `mcp_info` .

(C.4) How many rows are in the dataframe `mcp_info` ?

- Pandas read_csv https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

Merge – left join

- `merge()`: Merge dataframes based on the common column (key).
- Left join: Use keys from left frame.

df1

	employID	name
0	E011	John
1	E012	Diana
2	E013	Matthew
3	E014	Jerry
4	E015	Kathy
5	E016	Sara
6	E017	Alex

df2

	employID	birthday
0	E010	20-07
1	E012	12-06
2	E013	18-01
3	E015	16-05
4	E016	02-10
5	E017	19-08

Left DataFrame

Right DataFrame

key

```
pd.merge(df1,df2, how = 'left', on = 'employID' )
```

To be merged on the left side

	employID	name	birthday
0	E011	John	NaN
1	E012	Diana	12-06
2	E013	Matthew	18-01
3	E014	Jerry	NaN
4	E015	Kathy	16-05
5	E016	Sara	02-10
6	E017	Alex	19-08

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

Merge – inner join

- Inner join: Use the intersection of keys from both frames.

df1

	employID	name
0	E011	John
1	E012	Diana
2	E013	Matthew
3	E014	Jerry
4	E015	Kathy
5	E016	Sara
6	E017	Alex

df2

	employID	birthday
0	E010	20-07
1	E012	12-06
2	E013	18-01
3	E015	16-05
4	E016	02-10
5	E017	19-08

```
pd.merge(df1,df2, how = 'inner', on = 'employID' )
```

	employID	name	birthday
0	E012	Diana	12-06
1	E013	Matthew	18-01
2	E015	Kathy	16-05
3	E016	Sara	02-10
4	E017	Alex	19-08

Merge – outer join

- Outer join: Use the union of keys from both frames.

```
pd.merge(df1,df2, how = 'outer', on = 'employID' )
```

df1

	employID	name
0	E011	John
1	E012	Diana
2	E013	Matthew
3	E014	Jerry
4	E015	Kathy
5	E016	Sara
6	E017	Alex

df2

	employID	birthday
0	E010	20-07
1	E012	12-06
2	E013	18-01
3	E015	16-05
4	E016	02-10
5	E017	19-08

	employID	name	birthday
0	E011	John	NaN
1	E012	Diana	12-06
2	E013	Matthew	18-01
3	E014	Jerry	NaN
4	E015	Kathy	16-05
5	E016	Sara	02-10
6	E017	Alex	19-08
7	E010	NaN	20-07

Exercise

Exercise.D

(D.1) Import the dataset `municipality_name.csv` as a dataframe named `mcp_name`. The columns in the dataset are described as follows.

- Municipality_number (object)
- Municipality_name (object)

Hint: Use the argument `encoding = "iso8859_10"` to specify the character encoding.

(D.2) The dataframe `mcp_info` obtained in (C.3) currently lacks the information for "municipality_name". Retrieve the "municipality_name" data from the dataframe `mcp_name` and include it as a new column in "mcp_info". Store the resulting dataframe in a new variable named "mcp_full_info".
Expected result:

	Municipality_number	Population	Area	Municipality_name
0	0301	673469	454.03	OSLO
1	1101	14898	431.66	EIGERSUND
2	...	...	...	...

(D.3) Using the dataframe `mcp_full_info` obtained in (D.2), list the five most populous municipalities.

mcp_info

	Municipality_number	Population	Area
0	0301	673469	454.03
1	1101	14898	431.66
2	1103	141186	262.52
3	1106	37167	72.72
4	1108	76328	304.46
...	...	...	...
45	5059	11891	1906.26
46	5401	74541	2521.28
47	5402	24845	445.17
48	5403	20446	3849.47
49	5421	14930	1953.81

100 rows × 3 columns

mcp_name

	Municipality_number	Municipality_name
0	0301	OSLO
1	3024	BÆRUM
2	3025	ASKER
3	3020	NORDRE FOLLO
4	3021	ÅS
...	...	...
353	5441	DEATNU TANA
354	5444	SØR-VARANGER
355	5404	VARDØ
356	5440	BERLEVÅG
357	5443	BÅTSFJORD

358 rows × 2 columns

- Standard encodings https://docs.python.org/3/library/codecs.html#standard-encodings

Differences between merge() and concat()

- concat() simply stacks multiple DataFrames together either vertically or horizontally.

- merge() first align the selected common columns of the two DataFrames, and then pick up the remaining columns from the aligned rows of each DataFrame.