LEARNING PYTHON, PART 3: PANDAS

0. Import Pandas as a library

At the beginning, execute:

```
import pandas as pd
```

Then, to execute a function in Pandas, we use the format `pd.func()`

1. Pandas Series

Create a Pandas series

Very similar to Numpy array.

```
x = pd.Series(data = [1, 2, 3], index = ['a', 'b', 'c'])
```

Notes:

1. The `S` in `Series` should be capitalized!
2. The `index` argument is optional. If omitted, then the index is 0, 1, 2, ...
3. We can use both labels (the input of `index`) and natural integer index (0, 1, 2, ...) to visit elements. For example, both `x[0]` and `x['a']` return `1`.
4. The function `pd.Series` also take a dictionary as input.

Series operations

Matching elements based on the index.

```
y = pd.Series(data = [10, 20, 30], index = ['a', 'b', 'd'])
x + y
## returns:
a    11.0
b    22.0
```

```
c     NaN
d     NaN
dtype: float64
```

Exploratory data analysis on a series

- Show summary statistics
  - This method can also be applied to a column of a data frame
  - It shows data type and sample size (`count`, *excluding missing values*) of the column, and also
    - for numeric column: mean, median, std, Q1, Q3, min, max.
    - for categorical column: number of unique values (`unqiue`), largest category (`top`) and its count (`freq`).

```
x.describle()
```

- Show one way contingency table for a categorical series, or numerical series with a small number of values.

```
x.value_counts()
```

Note 1: the optional argument `dropna=False` also includes the counts of missing values.
Note 2: the optional argument `normalize=True` shows proportions instead of counts.

- Show unique values

```
x.unique()
```

Accessing overlap between two series

For every entry in series `x`, check if its in series `y`.

```
x.isin(y)
```

2. Data Frames

Create a data frame

- A data frame is matrix with row names `index` and column names `columns`.

```
df = pd.DataFrame(data = [[1,2,3], [4,5,6]], index = ['A','B'],
columns = 'X Y Z'.split())
```

Results:

	X	Y	Z
A	1	2	3
B	4	5	6

Note: `pd.DataFrame` can also take a dictionary, whose each component is a list and becomes a column in the data frame.

Copy a data frame

Use the `.copy()` method.

```
## Good; passed by values.
df_new = df.copy()

## Bad; passed by reference.
## Changing df_new will also change df.
df_new = df
```

Exploratory data analysis on a data frame

- Check dimension: the same as in Numpy.

```
df.shape
```

- Show column names as a list:

```
df.columns.tolist()
```

- Show information about the data frame, including dimensions, column names and their data type, and also memory usage of the data frame.

```
df.info()
```

Note: optional argument `verbose = True` forces the output to contain information of all columns (otherwise will be suppressed if number of columns is greater than 60).

- Show first `n` rows, default is 5.

```
df.head(n)
```

- Display multiple summary statistics at once, for each column in the data frame:

```
df.discribe()
```

Column operations

- Select a column:

```
df['X']
```

Note 1: `df[0]` returns an error!
Note 2: `df.X` also works, but is not recommended.

- Select multiple columns:

```
df[['X', 'Y']]
```

Note: the input is a list itself, so we have double brackets.

- Add a new column:

```
df['new'] = df['X'] * 2
```

- Remove a column:

```
df.drop('new', axis = 1, inplace = True)
```

Note 1: the `axis` argument decides if to drop a row or a column. `0` is for row (default), and `1` is for column.

Note 2: the `inplace` argument decides if the change applies to the `df` object. Default is `False`. Many other Pandas methods also have this `inplace` argument.

- Check column data type

```
## Check which columns are objects (non numerical)
df.select_dtypes(include = 'O')
```

- Change data type

```
## For example, change an object type to category type
df['X'].astype('category')
```

Row operations

- Select a row using label index:

```
df.loc['A']
```

Note 1: the `.loc` method can also be used to select an element in the data frame, with input row index and column index.

```
df.loc['A', 'X']
```

Note 2: to select multiple rows, we use double brackets, or slicing in labels. For the latter case, *both the start and the end are inclusive* (unlike usual slicing in Python).

```
df.loc[['A', 'B']]
df.loc['A':'B']
```

Note 3: to select a sub-matrix

```
df.loc[['A', 'B'], ['X', 'Y']]
```

Note 4: select certain columns, while keeping all rows

```
df.loc[:, 'X':'Y']
df['X':'Y']
```

- Select a row using integer index (position inputs):

```
df.iloc[0]
```

- Drop a row: `df.drop('A', axis = 0, inplace = True)`. See details of `.drop` in the “Column Operations” section.

Data frame conditional selection

- `df > 0` will return a matrix of boolean values.
- `df[df > 0]` will return a matrix: values where it is `True`, and `NaN` where it is `False`.
- When input a Series of boolean values in the bracket, it will return a sub-matrix whose corresponding *rows* are `True`. For example,

```
df[df['X'] > 3] ## returns the 2nd row in df
```

- We can stack multiple `[]`: e.g., `df[df['X'] > 3][['X', 'Y']]`
- Multiple conditions: use `&` and `|` to connect them, not `and` and `or`. Here, parentheses are necessary.

```
df[(df['X'] > 3) & (df['Y'] > 4)]
```

Resetting index

- The original index will be reset to a column, and row index will become integer ones (0, 1, 2, ...).

```
df.reset_index(inplace = true)
```

- Use a column as the new index
  (Benefit: retrieving data by index is quicker than by a column)

```
df.set_index('Z', inplace = true)
```

Reshaping data

Use `.pivot` method to create a new data frame, where

1. Row indices are the values of `Column1` in the data frame `df`. The number of rows equals the number of unique values in `Column1`.
2. Column names are the values of `Column2` in the data frame `df`. The number of columns equals the number of unique values in `Column2`.
3. Cells of the new data frame are from `Column3` in the data frame `df`.

```
df.pivot(index = 'Column1', columns = 'Column2', values =
'Column3')
```

Merging tables

Use the `pd.merge` function to perform a `JOIN` as in SQL

```
pd.merge(left_table, right_table, left_column = 'Column1',
right_column = 'Column2', how = 'inner')
```

Note 1: if use the row index for the left table to match, add the argument `left_index = True`, and remove the `left_column` argument (similarly, `right_index = True`).

Note 2: the `how` argument can be one of the four choices `'inner', 'outer', 'left', 'right'`.

Other data frame operations

- Matrix transpose

```
df.transpose()
```

Note: again, we can use argument `inplace = True` if we want to actually change the `df` data frame.

- Sum, mean, median, std, count

```
df.sum(axis = 0) ## column-wise sum, resulting a row (default)
```

```
df.sum(axis = 1) ## row-wise sum, resulting a column
```

Similarly, we have `.mean`, `.median`, `.std`, `.count` methods.

Note: missing values `NaN` are skipped.

- Quantile

```
df.quantile(q = 0.5, axis = 0)
```

- `.apply()`

```
df.apply(function, axis = 0) ## column-wise, default
df.apply(function, axis = 1) ## row-wise
```

Note: the function can be defined by the `lambda` expression.

- Group by

```
df.groupby(column_to_group)[column_to_summarize].operation_name()
```

Note 1: numerical operations such as `.sum`, `.mean`, `.median`, `.std` do not apply for character/string columns.
Note 2: `.groupby` method takes an optional argument `as_index = False` so that the `column_to_group` won't be the index in the resulting table.

- Clipping

Cap and floor a Pandas series or data frame.

```
df.clip(lower, upper) ## applies to every entry of the data frame
df.clip(lower_list, upper_list, axis = 0) ## column-wise
```

- Create dummy variables based on categorial variable

```
pd.get_dummies(df, dummy_na = False, drop_first = False)
```

Note 1: If the argument `columns = None`, then all the columns with object or category type will be converted.

Note 2: If the argument `dummy_na = True`, then all missing values (`np.nan` or `None`) will yield a new dummy column.

3. Missing Values

- A missing value is `np.nan`, and is shown as `NaN`. For example, the following data frame has one missing value.

```
df = pd.DataFrame(data = [[1,2], [3,np.nan]], columns = ['X','Y'])
```

- Another missing value (null value) value is `None`.
  - According to this discussion, using `np.nan` for missing values is better than using `None`.
  - `None` is the same as itself, but `np.nan` is not.

```
None == None ## returns True
np.nan == np.nan ## returns False
```

- In Pandas, the `pd.isnull()` function, or the `.isnull()` method for series and data frame can check if values are null. These will return True for both `None` and `np.nan`.

- Drop the rows or (columns) which contain missing values

```
df.dropna() ## rows
df.dropna(axis = 1) ## columns
```

Note: again, we can use argument `inplace = True` if we want to actually change the `df` data frame.

- Fill missing values as some other value.

```
df.fillna(value)
```

4. Read and Write: CSV Files

Use the `read_csv` function to read.

```
pd.read_csv(filename, nrow = 5, index_col = ['X', 'Y'])
```

- `nrow`: optional, number of rows to read.
- `index_col`: optional, use which column(s) as index.

Use the `.to_csv` method to write.

```
df.to_csv(filename)
```