Pandas

1. **DataFrame.head(num)**
   - Get the first *num* rows of the df

2. **Rename columns**
   df.rename(columns = {colCurrentName: rename}, inplace = True)

3. **Df.drop_duplicates()**
   - Drop duplicate rows; can specify which particular subset to target for specificity
     https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop_duplicates.html

4. **Reset column order**
   1) cols = list(df.columns)
   2) cols = cols[x:y] + cols[:x] + …
   3) df = df[cols]

5. **Pandas rename multiplexing index**
   - Why useful?
     Change column names to regular column names (not a tuple) after pivot() or other functions that cause multi-level columns
   - How?
     Conditional list comprehension + f string for max efficiency

```
- secF_1213_wide.columns = [col if type(Q) == str else f"{col}-Q{Q}" for col, Q in
  secF_1213_wide.columns]
```

6. **Checkout pivot() and pivot_table()**
   https://stackoverflow.com/questions/4406389/if-else-in-a-list-comprehension

7. **Get different columns between two dataFrames**
   Difference = df2.columns.difference(df1.columns)

8. **df.update(other)**
   https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.update.html

   - Modify in place using non-NA values from another df
   - Can pass in either a df or a Series; if pass in a df, must contain at least one matching index/column label, and update all columns in df with the values from "other" df. Can also pass in a Series (returned by "replace").

9. **df.replace(value = "")**
   https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.replace.html

   - replace all values within a dataset with value (a single value or dictionary etc)
   - returns a dataframe with replaced values; can specify "inplace = True" to modify Series/dataframe directly

10. **df.apply(lambda x: someFunc(x))**
    - apply the *someFunc* function to all elements in the specified Series
    - **Use in combination with a dictionary and will be able to transform all values in a column to respective other values quickly**

    To apply to all elements in the dataframe (not a Series but the whole df), use

    df.applymap(lambda x: someFunc(x))

11. **new_df = df.reindex(*list of column names or index names*, axis = 0 (default)/1)**
    https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.reindex.html
    - reindex Series or columns of Series with axis = 1
    - might be more useful at reordering columns of dataframes

12. **DataFrame.sort_values(by = [col_name], ascending = True, inplace = False)**
    https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_values.html
    - Sort rows by the values of a particular column

    To mimic a only single column merge, set the "kind" option to the only stable merging algorithm ***mergesort***

13. **DataFrame.reset_index(inplace = False)**
    - Reset indices of rows previously disrupted by sorting or filtering

14. **Difference between DataFrame.loc and using bracket to locate for specific subsets**
    https://stackoverflow.com/questions/48409128/what-is-the-difference-between-using-loc-and-using-just-square-brackets-to-filte

15. **DataFrame.groupby()**
    https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html

    Use Pandas.groupby() and transform() together:
    https://www.statology.org/pandas-groupby-transform/

16. **Knowing when changes to a subset of dataframe will affect original**

17. **DataFrame.iterrows()**
    https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.iterrows.html#pandas-dataframe-iterrows

18. **Asterisk (*) sign in Python and Pandas**
    https://stackoverflow.com/questions/33802940/python-pandas-meaning-of-asterisk-sign-in-expression

19. **Pandas.series.str.contains()**
    https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.contains.html

Test if a pattern or regex is contained in a column

20. **Pandas.explode()**
    https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.explode.html
    Transform each element of a list-like to a row, replicating index values.

21. **Pandas.json_normalize()**
    https://pandas.pydata.org/docs/reference/api/pandas.json_normalize.html
    Normalize semi-structured JSON data into a flat table.

22. **df.compare()**
    https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.compare.html
    Compare 2 different dataframes (only if they are identical shaped)

23. **Pandas.wide_to_long()**
    https://pandas.pydata.org/docs/reference/api/pandas.wide_to_long.html
    Convert a wide-format dataset to long-format dataset

24. Checking which entries are of “NaN” in a pandas dataframe?
    Get a Boolean Series: **pd.isna(df[‘my_column’])**
    OR
    df[‘my_column’].isnull()

25. Df.to_clipboard()
    https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_clipboard.html
    Copy the object to the system’s clipboard.

26. Check whether the value of a row is in a list
    Df[‘col_1’].isin([list_item1, list_item2, …])

27. Pandas.GroupBy.first()
    https://pandas.pydata.org/docs/reference/api/pandas.core.groupby.DataFrameGroupBy.first.html
    For each group (which will take one row in the result dataset), get the first non-null value of each column

General Python:

1. **Conditional in List Comprehension**

```
>>> xs = [22, 13, 45, 50, 98, 69, 43, 44, 1]
>>> [x+1 if x >= 45 else x+5 for x in xs]
[27, 18, 46, 51, 99, 70, 48, 49, 6]
```

2. **Dictionary Comprehension**

>>>mydict = {f'Col{i}':i for i in range(1,5) if i < 4} **(If you only want to use one conditional)**

```
>>>mydict
{'Col1': 1, 'Col2': 2, 'Col3': 3}
```

>>> mydict2 = {f'Col{i}':i if i < 4 else i + 1 for i in range(1,5)} (**both if and else)**

```
>>> mydict2
{'Col1': 1, 'Col2': 2, 'Col3': 3, 'Col4': 5}
```

3. **Type() vs isinstance()**
   - Type(object) returns type of the object
   - Instance(value, type) returns Boolean value for whether an object is of instance of

4. **Sorted**(*iterable*, *key = someFunc, reverse = Boolean*)
   - returns a sorted list of the specified iterable object
   - use incombination with **lambda()** function for the key method to sort a list of string containing numbers by their numbers

5. **lambda()**
   https://www.w3schools.com/python/python_lambda.asp
   - **Anonymous function**
   - lambda x: *someOp*(x) if x… else

6. **Python's enumerate() method**
   https://realpython.com/python-enumerate/#practicing-with-python-enumerate

   - Put the iterable object inside the **enumerate()** method to return both the iterable's content and the counter of the iterable

   Why use this?

     - Save the trouble of looping through index of the iterable with a separate variable (a "counter" variable)

7. **f-string**
   https://realpython.com/python-f-strings/

more convenient string formatting

Example:

```
name = “Yuanzhan”
print(f ‘Hello World, {name}’)

Console:
Hello World, Yuanzhan
```

8. **Python class’s variables: inside or outside “__init__”?**
    - If you don’t put a variable in “__int__”, it belongs to the class and not to the instance of that class. So when a new instance of the class is created that variable persists without being re-initialized.
    - If you put it into “__int__”, on the other hand, it belongs to that instance of the class and WILL get re-initialized when a new instance is called.

9. **os.getcwd()**
    - Get the current directory where the Python is executed in
    - If the project is created within an upper level folder, you need to manually cd into the directory where the other files that contain the desired functions are located

    - In Pycharm, every folder is potentially a project. When you open a folder, you open up a project. The name of the project is the name of the folder itself, and if you use **os.getcwd()** you will see the directory you are currently at. By default the interpreter will be your local python, but you can change it up by using an existing other interpreter that meets your needs.

10. **Python’s __future__ module**
    https://stackoverflow.com/questions/64290816/do-i-still-need-future-today
    - Allow Python codes that are run in older Python version/interpreter to use functions introduced in “future” (relative to the older version they are using) versions of Python.
    - Why use it?
      There are backward-incompatible changes to the language from time to time and the developers do not want to break many codes because of this

11. **Global** keyword in Python:
- useful for reassigning global variable values within local function;
- Do not need if you are not changing the global variable
- Mutations of mutable global objects (dictionary, list) DO NOT need to use **global**

https://stackoverflow.com/questions/14323817/global-dictionaries-dont-need-keyword-global-to-modify-them

12. **Multiprocessing** vs **Multithreading**

- **Multithreading** divides one memory space of one program into several workers, all workers share same memory so cross-thread data mutation is possible; multiprocessing starts several programs that have separate memories
- Python cannot do multithreading due to GIL, so multithreading is useless and can be even worse since thread needs to wait
- Multiprocessing is possible because each process is its own program and thus does not get constrained by GIL; data sharing across processes need to be achieved by Manager Class object
- Don't start more processes than your physical cores number (4 physical core + 8 logical processor is a common architecture);

https://stackoverflow.com/questions/40217873/multiprocessing-use-only-the-physical-cores

13. **Iterator, Iterable, and for loop**

Python Package Manager:

pip:

- Use PyPI, or The Python Package Index, as the repository (channel) where packages installed are downloaded from;
- Python's Native package manager

Useful Commands:

1) pip freeze/ pip list

- Check install modules' versions

Subcommands:

pip freeze | findstr *module_name* (same works for pip list)

- find all modules whose names contain the text *module_name*

pip freeze > *requirements.txt*

- Dump the information of all current installed modules into *requirements.txt,* which is the standard file name in a repo where a list of all required modules for the project is stored

2) pip install *module_name*

- Install the given module with name *module_name*

Subcommands:

pip install -r[equirement] requirements.txt

- Install all modules in requirements.txt to the current environment you are in

Conda:

- An open source package management system and environment management system

Useful Commands:

1) **conda list**

-- list all installed packages in the current conda environment

2) conda install -c some-channel *module_name*

<u>**Python Virtual Environment:**</u>

<u>Conda Environment:</u>

- A conda base environment is created when you installed **Anaconda** distribution (in which case you have a conda environment pre-installed with 1500+ scientific packages) or **Minconda** distribution **(**in which case you have an almost empty conda environment).

More on Anaconda vs Minconda here: https://stackoverflow.com/questions/45421163/anaconda-vs-miniconda

<u>Useful Commands:</u>

1) **conda create –name myenv python = xx.xx (version that you want)**

-- create conda environment

2) **conda activate + env**

-- activate conda environment

3) **conda env list**

-- check all existing conda environment

<u>Virtualenv</u>

<u>Pipenv</u>