Finance with Python
Jan Schneider

For best viewing experience open this document in a "two page view" (and use a big monitor . . .).

4 Working with Tables: SEC Filings

4.1 CSV Files

Suppose we have the following table:

```
import pandas as pd                                  # Import pandas library and abbreviate as "pd".

prices = pd.DataFrame()                              # Create empty table.
prices['day']    = ['Monday','Tuesday','Wednesday','Thursday','Friday']       # Add column 'day'.
prices['stock1'] = [20, 21, 18, 19, 20]
prices['stock2'] = [80, 75, 70, 72, 71]
prices['stock3'] = [60, 65, 68, 66, 67]
```

Let's use column "day" as the row index:

```
prices = prices.set_index('day')
```

We can export this table as a csv file:

```
prices.to_csv('prices.csv')                          # Write file 'prices.csv' into current directory.
```

Now we have a file *prices.csv* in the current directory (same directory as the notebook we are currently working with). Figure 1 shows the notebook and the new file "prices.csv" inside the current directory. We could now open this file with any spreadsheet program such as Excel or Numbers. If you double-click on "prices.csv" you can see the contents directly in JupyterLab (Figure 2).
We can now read the file like this:

```
prices = pd.read_csv('prices.csv')                   # Read file 'prices.csv' from current directory.
```

Reading the cvs file produces Table 3. By default, pandas uses 0, 1, 2, .. as row indexes. We can also tell pandas to use any of the columns of "prices.csv" as the index. For example if we want the first column (column 0) to be the index we write:

```
prices = pd.read_csv('prices.csv', index_col=0)    # Read file 'prices.csv' and use column 0 as index.
```

Or equivalently:

```
pd.read_csv('prices.csv', index_col='day')
pd.read_csv('prices.csv').set_index('day')         # Same.
```

Sometimes we don't need the index, and then we save the table like this:

```
prices.to_csv('prices.csv', index=False)
```

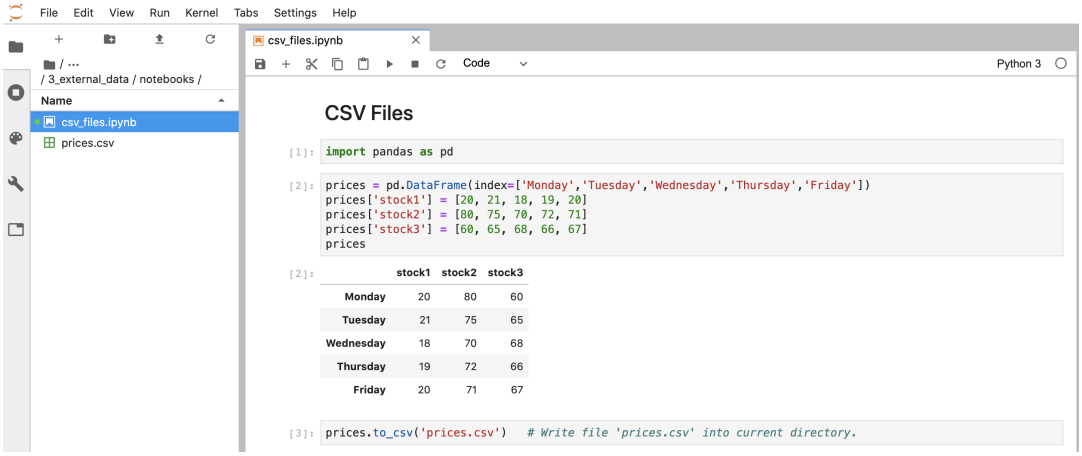
If we work with a lot of different data files, we might want to put these files into a separate directory. For example, click on the "add folder" symbol on top of the left panel to create a directory called "data" inside your current working directory (see left panel in Figure 5). And now we can save the table to the "data" directory like this:

```
prices.to_csv('data/prices1.csv')                    # Write file 'prices1.csv' into subdirectory 'data'.
```

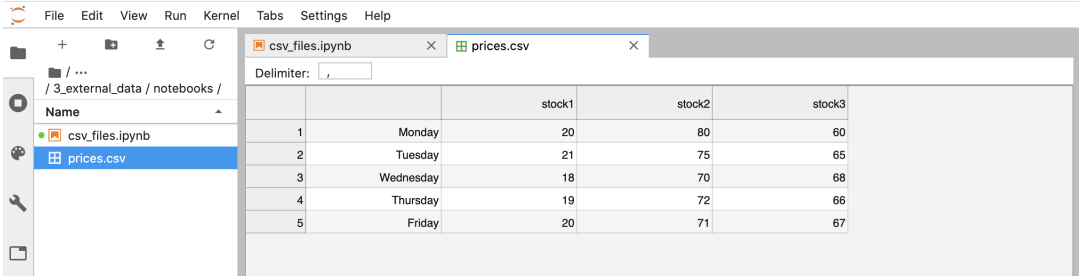
In a similar way we can also navigate into any other directory:

```
prices.to_csv('../prices2.csv')                      #  One directory up.
prices.to_csv('../../prices2.csv')                   #  Two directories up.
prices.to_csv('../folder1/folder2/prices2.csv')      #  One directory up, then go to folder1, then to folder2.
```

1 Saving a table as a csv file



2 Viewing the csv file in JupyterLab



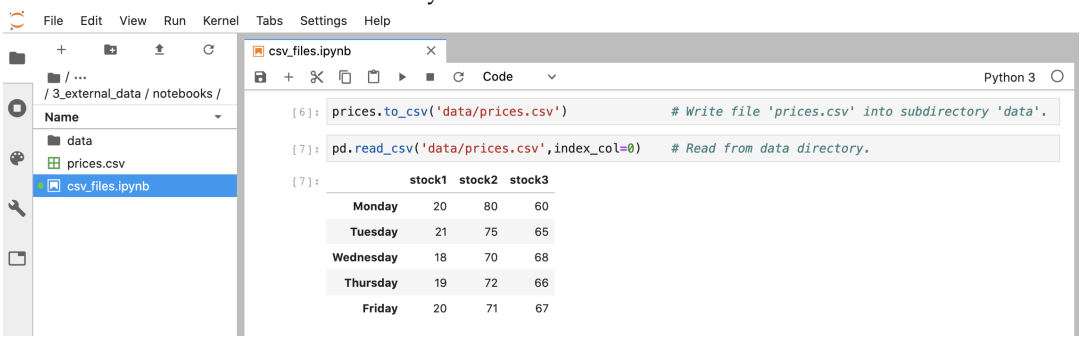
3 pd.read_csv('prices.csv')

	day	stock1	stock2	stock3
0	Monday	20	80	60
1	Tuesday	21	75	65
2	Wednesday	18	70	68
3	Thursday	19	72	66
4	Friday	20	71	67

4 pd.read_csv('prices.csv', index_col=0)

	stock1	stock2	stock3
Monday	20	80	60
Tuesday	21	75	65
Wednesday	18	70	68
Thursday	19	72	66
Friday	20	71	67

5 Read and write from subdirectory



4.2 ZIP Files: Download SEC Data

We are going to download financial statement data from the SEC.
Start by importing these libraries:

```
import pandas as pd                 #  Working with tables
import requests, zipfile, io        #  Downloading and unzipping Files
from pathlib import Path            #  Work with directories (folders)
```

Now go to: https://www.sec.gov/dera/data/financial-statement-and-notes-data-set.html. (Figure 1).
If you scroll down to the bottom of this website you see a table titled "Data Downloads". Right-click on any of the links in this table and select "Copy Link Address" to get the address of the file (Figure 2).
Now paste the address into your Jupyter notebook and download the file like this:

```
url = 'https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2020_12_notes.zip'
r = requests.get(url)
r.ok                          # Returns TRUE if status code not 4xx or 5xx
```

Open the zip file:

```
z = zipfile.ZipFile(io.BytesIO( r.content ))
```

Get the names of all the files in the zip folder:

```
z.namelist()     # ['sub.tsv', 'tag.tsv', 'dim.tsv', 'ren.tsv', 'cal.tsv', 'pre.tsv', 'num.tsv', ...]
```

And now we can choose any of the files inside the zip folder and open it like this:

```
sub = z.open( 'sub.tsv' )
```

Read this file with pandas:

```
pd.read_table(sub)          # Use read_table() to read a tsv (tab-separated values) file.
```

Create a directory for the downloads:

```
unzip_folder_name = 'data/sec/downloads/2020_12'

Path(unzip_folder_name).mkdir(parents=True, exist_ok=True)
```

Save selected files to this directory:

```
z.extractall(members=['sub.tsv','num.tsv'], path=unzip_folder_name)
```

Or alternatively save all files to this directory

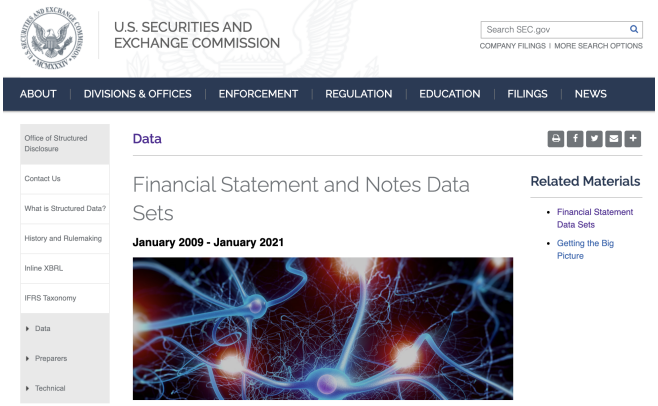
```
z.extractall(unzip_folder_name)
```

Table 3 shows some more URLs from the SEC website. Note how the SEC switched from quarterly to monthly updates at the end of 2021.
Figure 4 shows a function "download_file_from_SEC()" that contains our code from above (in rows 2 and 3 we construct the download link, using the general structure of the URLs in Table 3).
We can use this function like this:

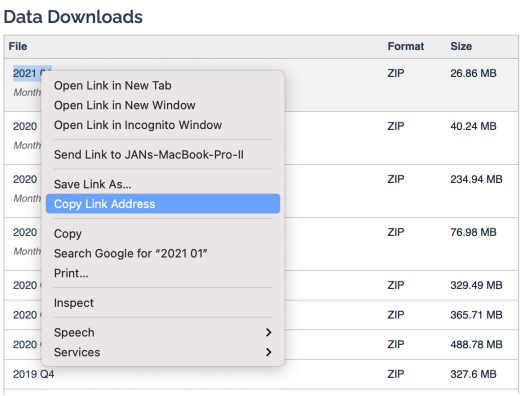
```
download_file_from_SEC('2021_01')       # Download files from the "2021_01" link.
```

Now we can use this function to download all the data on the SEC website (bottom cell in Figure 4).
(This takes about 40GB. If you don't have that much space, wait with the downloads until page 13.)

1



2



3 URLs from SEC website

https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2009q1_notes.zip
https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2009q2_notes.zip
⋮
https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2020q3_notes.zip
https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2020_11_notes.zip
https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2020_12_notes.zip
https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2021_01_notes.zip
⋮

4

[23]:
```python
def download_file_from_SEC(period):                                  # period = '2020q1' or '2021_01`, ...
    file_ref = '/files/dera/data/financial-statement-and-notes-data-sets/'+period+'_notes.zip'
    url      = 'https://www.sec.gov' + file_ref
    unzip_folder_name = 'data/sec/downloads/' + period               # Where to put contents of unzipped file

    r = requests.get(url)
    if r.ok:
        print('Downloaded:', url)
        Path(unzip_folder_name).mkdir(parents=True, exist_ok=True)       # Create folder for unzipped file.
        z = zipfile.ZipFile(io.BytesIO(r.content))                       # Open zip file.
        z.extractall(members=['sub.tsv','num.tsv'], path=unzip_folder_name) # Save unzipped files.
    else:
        print('file does not exist:',url)
```

[24]:
```python
# Download specific folder
download_file_from_SEC('2021_01')
```

```
Downloaded: https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2021_01_notes.zip
```

[ ]:
```python
# Download all historical files (2009q1 is empty)
download_file_from_SEC('2009q2')
download_file_from_SEC('2009q3')
download_file_from_SEC('2009q4')

for year in range(2010,2021):                    # All quarters from 2010 to 2020.
    for quarter in [1,2,3,4]:
        period = str(year)+'q'+str(quarter)
        download_file_from_SEC(period)

for year in range(2020,2022):                    # All months from 2020 to 2021.
    for month in range(1,13):
        period = str(year)+'_'+str(month)
        download_file_from_SEC(period)
```

4.3 Selecting Rows and Columns

Example table:

```
import pandas as pd

t = pd.DataFrame(index=['a','b','c','b','d'])
t['A'] = [1,   2,   3,   4,   5]
t['B'] = [10,  20,  30,  40,  50]
t['C'] = [100, 200, 300, 400, 500]
t['D'] = [1000,2000,3000,4000,5000]
```

(Table 1.) We can select subsets of this table by ".loc[]" or ".iloc[]" like this:

tablename.loc [row names, column names]

tablename.iloc[row indexes, column indexes]

For example, get row 'a' and column 'B':

```
t.loc['a','B']                          # Returns 10
t.iloc[0,1]                             # Same (row 0, column 1).
```

If we don't specify the columns we get the entire row:

```
t.loc['a']                              # Row a and all columns (since no columns specified)
t.iloc[0]                               # Same.
```

(Table 2). For the row and column names and indexes we can use in general:

1. a single label/index 'A' 0
2. a list of labels/indexes ['b','c'] [1,2]
3. a slice of labels/indexes 'b':'d' [1:4]

When we select a slice (from:to) then .loc[] includes the last label but .iloc does not include the last index:

```
t.loc['b':'e']                          # Rows b,c,d,e.
t.iloc[1:5]                             # Same (rows 1,2,3,4;   5 is not included).
```

More examples for access by labels:

```
t.loc['d']                              # Both rows d (Table t has 2 rows d) and all columns.
t.loc['b':'e', 'B']                     # Rows from b to e and column B
t.loc['c':, ['A','D']]                  # Rows from c, columns A and D
t.loc[:, 'A':'C']                       # All rows and columns A to C
```

Note how we can select return types by enclosing single labels in brackets:

```
t.loc['a',   'B']                       # Integer 10.
t.loc['a',  ['B']]                      # Series with value 10     ('B' as a list).
t.loc[['a'], 'B' ]                      # Series with value 10     ('a' as a list).
t.loc[['a'],['B']]                      # DataFrame with value 10  ('a' and 'B' as a list).
```

(see also Table 10).

1 Example table

	A	B	C	D
a	1	10	100	1000
b	2	20	200	2000
c	3	30	300	3000
d	4	40	400	4000
e	5	50	500	5000
d	6	60	600	6000

2 loc['a'] or iloc[0]

	A	B	C	D
a	1	10	100	1000
b	2	20	200	2000
c	3	30	300	3000
d	4	40	400	4000
e	5	50	500	5000
d	6	60	600	6000

3 loc['d'] or iloc[[3,5]]

	A	B	C	D
a	1	10	100	1000
b	2	20	200	2000
c	3	30	300	3000
d	4	40	400	4000
e	5	50	500	5000
d	6	60	600	6000

4 loc['b':'e', 'B'] or iloc[1:5, 1]

	A	B	C	D
a	1	10	100	1000
b	2	20	200	2000
c	3	30	300	3000
d	4	40	400	4000
e	5	50	500	5000
d	6	60	600	6000

10 Return types

	Example (same for iloc)	Return type
single row, single column	t.loc['a', 'A']	cell value
single row, multiple columns	t.loc['a', ['A','B']]	pandas series
	t.loc['a', 'A':'C']	pandas series
	t.loc['a', ['A']]	pandas series
multiple rows, single column	t.loc['a':'c', 'B']	pandas series
	t.loc[['a'], 'A']	pandas series
multiple rows, multiple columns	t.loc[['a','b'], ['A','B']]	pandas dataFrame
	t.loc[['a'], ['A']]	pandas dataFrame

4.4 Selecting with []

Example table:

```
import pandas as pd

t = pd.DataFrame(index=['a','b','c','b','d'])
t['A'] = [1,   2,   3,   4,   5]
t['B'] = [10,  20,  30,  40,  50]
t['C'] = [100, 200, 300, 400, 500]
t['D'] = [1000,2000,3000,4000,5000]
```

Sometimes it is too much work to always type *.loc[]* or *.iloc[]*.
We can abbreviate *.loc[]* or *.iloc[]* with "." or "[]" according to the following rules:

- table.column_label
- table[X], where X is:
  - single column label
  - list of column lables
  - slice of row numbers or row labels (from:to)

Examples for access by labels:

```
t.A                                     # Column A as a series
t['A']                                  # Same
t[['A']]                                # Column A as as a dataFrame
t[['A','B']]                            # Columns A and B
t['a':'c']                              # Rows a to c
```

Examples for access by index:

```
t[0:3]                                  #  Rows 0, 1, 2 (3 not included)
t[:3]                                   #  Same (0, 1, 2)
t[-3:]                                  #  Last 3 rows (-3,-2,-1).
```

But we cannot access a single row like this:

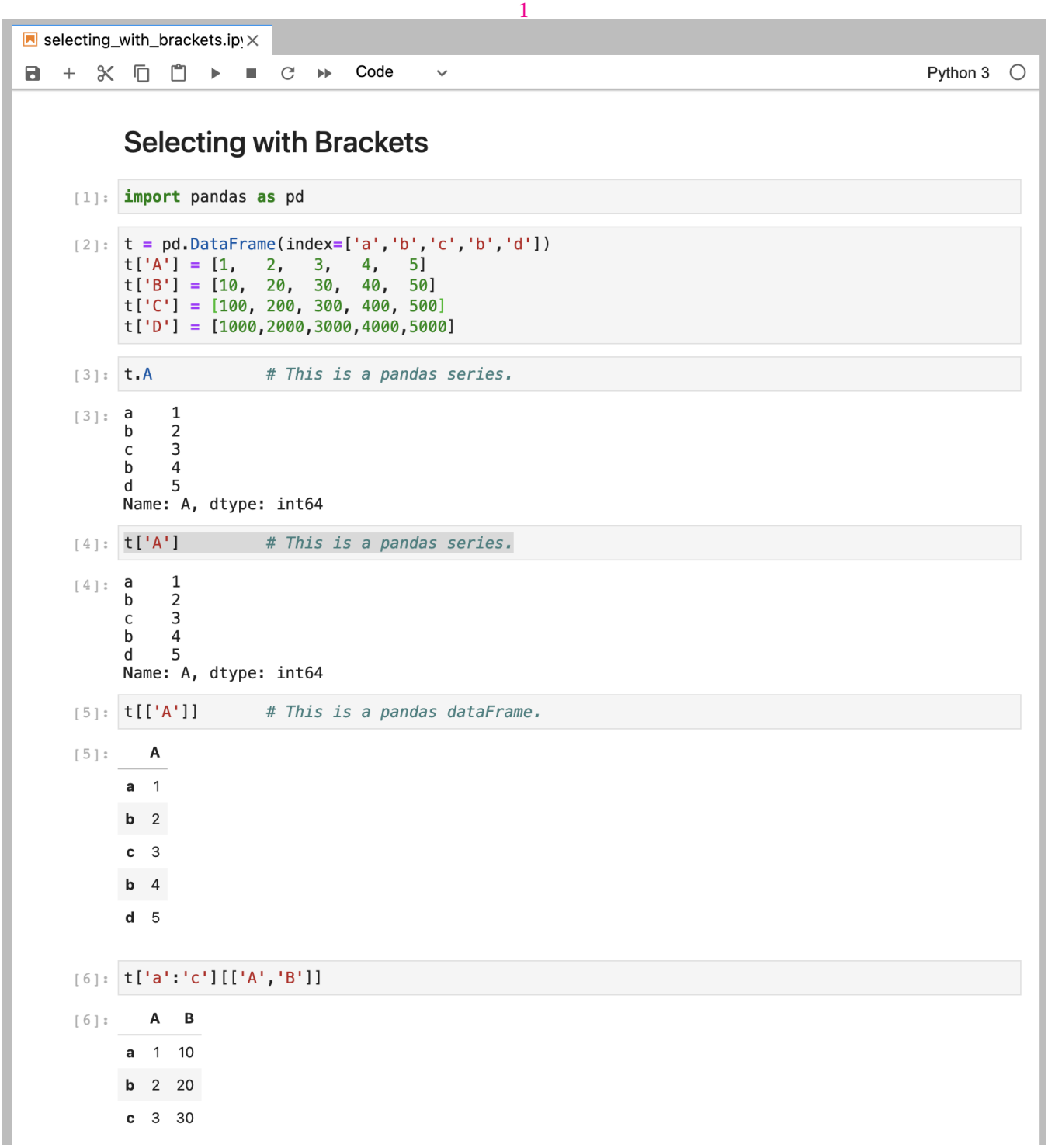
```
t[1]                                    # Error, this does not work (use data.iloc[1])
```

(here Pandas looks for a column with label "1").
We can chain access methods:

```
t[['A','B']] [a:c]                      # First select columns ['A','B'], then rows a to c
t[a:c] [['A','B']]                      # First select rows a to c, then columns ['A','B']
```

These two expressions are equivalent and they both produce the table in Cell 6 in Figure 1.
Or combine labels and indexes:

```
t['A'][-2:]                             # Last 2 rows of column A.
t[-2:]['A']                             # Same.
t.A[-2:]                                # Same.
```


selecting_with_brackets.ipy
Code
Python 3
Selecting with Brackets
[1]: import pandas as pd
[2]: t = pd.DataFrame(index=['a','b','c','b','d'])
t['A'] = [1, 2, 3, 4, 5]
t['B'] = [10, 20, 30, 40, 50]
t['C'] = [100, 200, 300, 400, 500]
t['D'] = [1000,2000,3000,4000,5000]
[3]: t.A # This is a pandas series.
[3]: a 1
b 2
c 3
b 4
d 5
Name: A, dtype: int64
[4]: t['A'] # This is a pandas series.
[4]: a 1
b 2
c 3
b 4
d 5
Name: A, dtype: int64
[5]: t[['A']] # This is a pandas dataFrame.
[5]: A
a 1
b 2
c 3
b 4
d 5
[6]: t['a':'c'][['A','B']]
[6]: A B
a 1 10
b 2 20
c 3 30

4.5 Dates

Define a date string.

```
date = '2021-1-28'
```

Now *date* refers to the string '2021-1-28'. Convert this string to a pandas "Timestamp":

```
import pandas as pd

date = pd.to_datetime(date)       # Now date is a "Timestamp".
```

And now we can apply various pandas date methods. For example:

```
date.year                         # 2021
date.day_name()                   # 'Monday'
```

Suppose we have the following table:

```
t = pd.DataFrame({'A':['20200101','20191130','111']})
```

Convert column "A" to datetime and put the result in column "B":

```
t['B'] = pd.to_datetime(t.A, errors='coerce')  # errors='coerce': invalid dates will be converted to NaT
```

Here pandas converts the string '111' into NaT ("not a time"). All rows where we have a viable date:

```
t[t.B.notnull()]                  # Returns rows 0 and 1.
```

If we want to apply the date functions to an entire series, we need to write *series.dt.method()*:

```
t['C'] = t.B.dt.day_name()        # Get names of the weekdays in column B.
```

Save this table to the current working directory:

```
t.to_csv('test.csv', index=False)      # Save as "test.csv" (do not save the row indexes 0,1,2).
```

And now read this table:

```
t2 = pd.read_csv('test.csv')
t2.B[0]                           # '2020-01-01' (this is a string)
```

Note how the dates in table t2 are strings now. If we want the dates to be dates, we can convert them with "pd.to_datetime()" or we can try to parse the columns as dates when we read the file:

```
t3 = pd.read_csv('test.csv', parse_dates=['A','B'])  # Try to parse columns A nd B as dates
t3.A[0]                                              # '20200101'  (still a string -> parsing did not work).
t3.B[0]                                              # Timestamp('2020-01-01 00:00:00')
```

So now column B contains dates. We where not able to parse column A as dates because of the invalid date format '111' in row 2.
Check Figure 1 to see how we can use pandas to create a range of dates and select rows from a table with a date index.

dates.ipynb

Code Python 3

Dates

[1]:
```
import pandas as pd
```

[2]:
```
dates = pd.date_range('2020-1-1','2020-12-31')    # Create a date range.
dates
```

[2]:
```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
               '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08',
               '2020-01-09', '2020-01-10',
               ...
               '2020-12-22', '2020-12-23', '2020-12-24', '2020-12-25',
               '2020-12-26', '2020-12-27', '2020-12-28', '2020-12-29',
               '2020-12-30', '2020-12-31'],
              dtype='datetime64[ns]', length=366, freq='D')
```

[3]:
```
t      = pd.DataFrame(index=dates)              # Use the dates as index for a table.
t['A'] = range(len(t))                          # Add column A with values 0,1,2,...
t
```

[3]:

	A
2020-01-01	0
2020-01-02	1
2020-01-03	2
2020-01-04	3
2020-01-05	4
...	...
2020-12-27	361
2020-12-28	362
2020-12-29	363
2020-12-30	364
2020-12-31	365

366 rows × 1 columns

[4]:
```
t['2020-1-12':'2020-1-15']                  # Select rows from this table.
```

[4]:

	A
2020-01-12	11
2020-01-13	12
2020-01-14	13
2020-01-15	14

[5]:
```
t.loc['2020-2']                             # All rows in February.
```

[5]:

	A
2020-02-01	31
2020-02-02	32
2020-02-03	33
2020-02-04	34
2020-02-05	35
2020-02-06	36
2020-02-07	37
2020-02-08	38
2020-02-09	39

4.6 Merging Tables

Suppose we have the following two tables:

```
t1 = pd.DataFrame({'A':[1, 2, 3], 'B':[4, 5, 6]},  index=['a','b','c'])
t2 = pd.DataFrame({'C':[10,20,30],'D':[40,50,60]}, index=['b','c','d'])
```

(Tables 1 and 2). We can join the two tables like this:

```
t3 = t1.join(t2)
```

(Table 3). Note how the combined table contains all rows of the table on the left (t1). Since table t2 does not have a row 'a', the corresponding values appear as *NaN* in t3.
If we want to keep all rows of both tables, we can perform an "outer" join:

```
t1.join(t2, how='outer')
```

(Table 4). Here are the four possible join methods:

```
t1.join(t2, how='left')   # Keep rows of left table (this is default; we can omit the 'how').
t1.join(t2, how='right')  # keep rows of right table
t1.join(t2, how='inner')  # Keep only rows that appear in both tables.
t1.join(t2, how='outer')  # Keep all rows from both tables.
```

Suppose the two table also have a common column X:

```
t1['X'] = [1,2,3]
t2['X'] = [2,4,3]
```

(Tables 5 and 6). When we join the tables now, we need to specify a suffix for the overlapping columns:

```
t1.join(t2, lsuffix='_t1', rsuffix='_t2', how='outer')          # Only 1 suffix is required.
```

(Table 7).
Alternatively we can also use the "combine_first()" method to perform an outer join and use the right table to fill missing values in the overlapping columns:

```
t4 = t1.combine_first(t2)
```

(Table 8, note how t1 only has rows a,b,c, so the first 3 values (1,2,3) in t4.X come from table t1 and the fourth value (3) comes from table t2).
We can also use column X to merge the tables like this:

```
t1.merge(t2, on='X', how='outer')     # how='inner' is default.
```

(Table 9). We can also merge on columns with different names. For example, in table t2 rename column 'X' to 'Y' and then merge:

```
t1.merge( t2.rename(columns={'X':'Y'}), left_on='X', right_on='Y', how='outer')
```

(Table 10).
Sometimes we want to combine tables vertically. Here are two equivalent ways to put table t2 below table t1::

```
t1.append(t2)             #  Appending t2 to t1.
pd.concat([t1,t2])        #  Same
```

(Table 11). The "concat()" method is especially useful if we have more than two tables.
Also note: if a column contains a missing value, pandas transforms all integers to floats (check the columns in the various tables).

1 t1

	A	B
a	1	4
b	2	5
c	3	6

2 t2

	C	D
b	10	40
c	20	50
d	30	60

3 t1.join(t2)

	A	B	C	D
a	1	4	NaN	NaN
b	2	5	10.0	40.0
c	3	6	20.0	50.0

4 t1.join(t2, how='outer')

	A	B	C	D
a	1.0	4.0	NaN	NaN
b	2.0	5.0	10.0	40.0
c	3.0	6.0	20.0	50.0
d	NaN	NaN	30.0	60.0

5 t1 with X

	A	B	X
a	1	4	1
b	2	5	2
c	3	6	3

6 t2 with X

	C	D	X
b	10	40	2
c	20	50	4
d	30	60	3

7 t1.join(t2, lsuffix='_t1', rsuffix='_t2', how='outer')

	A	B	X_t1	C	D	X_t2
a	1.0	4.0	1.0	NaN	NaN	NaN
b	2.0	5.0	2.0	10.0	40.0	2.0
c	3.0	6.0	3.0	20.0	50.0	4.0
d	NaN	NaN	NaN	30.0	60.0	3.0

8 t1.combine_first(t2)

	A	B	C	D	X
a	1.0	4.0	NaN	NaN	1.0
b	2.0	5.0	10.0	40.0	2.0
c	3.0	6.0	20.0	50.0	3.0
d	NaN	NaN	30.0	60.0	3.0

9 t1.merge(t2, on='X', how='outer')

	A	B	X	C	D
0	1.0	4.0	1	NaN	NaN
1	2.0	5.0	2	10.0	40.0
2	3.0	6.0	3	30.0	60.0
3	NaN	NaN	4	20.0	50.0

10 merge with left_on='X', right_on='Y', how='outer'

	A	B	X	C	D	Y
0	1.0	4.0	1.0	NaN	NaN	NaN
1	2.0	5.0	2.0	10.0	40.0	2.0
2	3.0	6.0	3.0	30.0	60.0	3.0
3	NaN	NaN	NaN	20.0	50.0	4.0

11 t1.append(t2)

	A	B	X	C	D
a	1.0	4.0	1	NaN	NaN
b	2.0	5.0	2	NaN	NaN
c	3.0	6.0	3	NaN	NaN
b	NaN	NaN	2	10.0	40.0
c	NaN	NaN	4	20.0	50.0
d	NaN	NaN	3	30.0	60.0

4.7 Merge SEC Files

Read the files "sub.tsv" and "num.tsv" from January 2021:

```
folder    = '2021_01'
directory = 'data/sec/downloads/'

filings = pd.read_table(directory+folder+'/sub.tsv')
numbers = pd.read_table(directory+folder+'/num.tsv', encoding='ISO-8859-1', error_bad_lines=False)
```

(see Tables 1 and 2).
We want all 10-Qs and 10-Ks and we exclude any filing without CIK:

```
filings = filings[filings.form.isin(['10-Q','10-K']) & filings.cik.notnull()]
```

We only want data for the entire firm and not for the segments, except for share outstanding:

```
keep_these_tags_with_segments = ['EntityCommonStockSharesOutstanding','CommonStockSharesOutstanding']
numbers = numbers[(numbers.dimh=='0x00000000') | numbers.tag.isin(keep_these_tags_with_segments)]
```

We want merger the two tables and then keep these columns:

```
keep_these_columns = ['cik','sic','countryinc','tag','filed','ddate','qtrs','value']
```

Merge the two files by accession number, only keeping rows that appear in both files (inner merge):

```
merged = numbers .merge(filings, on='adsh', how='inner') .set_index('adsh') [keep_these_columns]
```

Transform columns with date strings to pandas dates:

```
merged['filed'] = pd.to_datetime(merged.filed, format='%Y%m%d', errors='coerce')
merged['ddate'] = pd.to_datetime(merged.ddate, format='%Y%m%d', errors='coerce')

merged = merged[merged.filed.notnull() & merged.ddate.notnull()]  # Drop all rows with invalid dates.
```

(Tables 3). Drop any duplicated rows:

```
merged = merged.drop_duplicates()
```

Now create a directory for the "merged" folder inside your "sec" folder (data/sec/merged) and then save the table:

```
merged.to_csv('data/sec/merged/'+folder+'.csv', index=False)
```

Figure 4 shows how we can put all this code into a "merge_sec_files()" function.
And now we can loop over all folders inside the "data/sec/downloads" directory and use this function to merged the files and then save the merged table into the "data/sec/merged" directory (so now we have one merged file for each folder in "downloads"):

```
for folder in os.listdir('data/sec/downloads/'):    # Loop over all folders in directory "downloads".
    print(folder)
    merged = merge_sec_files(folder)                                    # Generate the merged table.
    merged.to_csv('data/sec/merged/'+folder+'.csv', index=False)        # Save the nerged table.
```

If you would like to free up space you can delete the contents of the "downloads" folder now. If you do not have 40GB to download all SEC files, download only part of the files, merge them, delete them and repeat this for the remaining files (or write a loop that downloads, merges, deletes for every single SEC file).

1 filings

	adsh	cik	name	sic	countryinc	form	period	filed	···
0	0000007332-21-000003	7332	SOUTHWESTERN ENERGY CO	1311.0	US	8-K	20201231	20210104	
1	0000009346-21-000002	9346	PROTECTIVE INSURANCE CORP	6331.0	US	8-K	20201231	20210104	
2	0000020212-21-000003	20212	CHURCHILL DOWNS INC	7948.0	US	8-K	20201231	20210104	
⋮									

2 numbers

	adsh	tag	version	ddate	qtrs	uom	dimh	iprx	value	···
0	0001477932-21-000050	AccountsPayableAndAccruedLiabilitiesCurrent	us-gaap/2019	20201130	0	USD	0x00000000	0	7094.0	
1	0001477932-21-000050	AccountsPayableAndAccruedLiabilitiesCurrent	us-gaap/2019	20200229	0	USD	0x00000000	0	8698.0	
2	0001640334-21-000042	AccountsPayableAndAccruedLiabilitiesCurrent	us-gaap/2019	20180731	0	USD	0x00000000	0	5890.0	
⋮										

3 merged

adsh	cik	sic	countryinc	tag	filed	ddate	qtrs	value
0001477932-21-000050	1517389	7371.0	US	AccountsPayableAndAccruedLiabilitiesCurrent	2021-01-06	2020-11-30	0	7094.0
0001477932-21-000050	1517389	7371.0	US	AccountsPayableAndAccruedLiabilitiesCurrent	2021-01-06	2020-02-29	0	8698.0
0001477932-21-000050	1517389	7371.0	US	AccountsReceivableNetCurrent	2021-01-06	2020-11-30	0	9384.0
⋮								

4 Merge function for SEC files

```
def merge_sec_files(folder):

    keep_these_columns = ['cik','sic','countryinc','tag','filed','ddate','qtrs','value']
    keep_these_tags_with_segments = ['EntityCommonStockSharesOutstanding','CommonStockSharesOutstanding']

    filings = pd.read_table('data/sec/downloads/'+folder+'/sub.tsv')
    numbers = pd.read_table('data/sec/downloads/'+folder+'/num.tsv', encoding='ISO-8859-1', error_bad_lines=False)

    filings = filings[filings.form.isin(['10-Q','10-K']) & filings.cik.notnull()]
    numbers = numbers[(numbers.dimh=='0x00000000') | numbers.tag.isin(keep_these_tags_with_segments)]

    merged = numbers.merge(filings, on='adsh', how='inner').set_index('adsh')[keep_these_columns]

    merged['filed'] = pd.to_datetime(merged.filed, format='%Y%m%d', errors='coerce')
    merged['ddate'] = pd.to_datetime(merged.ddate, format='%Y%m%d', errors='coerce')

    return merged[merged.filed.notnull() & merged.ddate.notnull()].drop_duplicates()
```

4.8 Sorting and Grouping

Example table:

```
t = pd.DataFrame()
t['Firm']      = ['A','B','C','D','E','F']
t['Sector']    = ['Healthcare','Industrials','Healthcare','Healthcare','Industrials','Industrials']
t['Style']     = ['Growth','Growth','Value','Growth','Value','Value']
t['Revenue']   = [800,1600,500,4000,6000,400]                    # Units: USD 100M
t['Earnings']  = [200,400,100,800,600,5000]                      # Units: USD 100M
```

(Table 1). Sum the "Earnings" column to calculate the total earnings of all firms:

```
t.Earnings.sum()
```

To calculate the total earnings for each sector, group the rows by sector and then calculate the sum:

```
t.groupby('Sector').sum()
```

(Table 2). Note how the grouping turned the column "Sector" inot the row index now. If we want to keep "Sector" as a column:

```
t.groupby('Sector', as_index=False).sum()                        # Don't use "Sector" as index.
```

(Table 3). We can also group by multiple columns. For example, calculate the total earnings by style within each sector:

```
t.groupby(['Sector','Style']).sum()
```

(Table 4). Now we have a MultiIndex with two index levels: "Sector" on level zero and "Style" on level one.
We can also change the shape of this table. For example, put index level 1 ("Style") as the columns:

```
t.groupby(['Sector','Style']).sum().unstack(level=1)             # Use index level 1 as columns.
t.groupby(['Sector','Style']).sum().unstack(level='Style')       # Same.
t.groupby(['Sector','Style']).sum().unstack()                    # Same (level=1 is default).
```

(Table 5). Or use the first level ("Earnings") as columns:

```
t.groupby(['Sector','Style']).sum().unstack(level='Earnings')
```

Or select only "Earnings":

```
t.groupby(['Sector','Style']).Earnings.sum().unstack()
t.groupby(['Sector','Style']).sum().unstack().Earnings           # Same.
```

(Table 6). Similar to grouping, we can also sort the table by columns:

```
t.sort_values('Earnings')                                        # Sort by Earnings (ascending).
t.sort_values(['Sector','Earnings'])                             # Sort by Sector and then Earnings.
t.sort_values(['Sector','Earnings'], ascending=[True,False])     # Specify sorting order.
```

(Table 7 shows the result of the 2nd line).
Or combine sorting and grouping:

```
t.sort_values(['Sector','Earnings']).groupby(['Sector']).first()
```

(Table 8). Here we sort by Sector and Earnings and then select the first row within each Sector (firm with lowest earnings since we sort ascending).

1 Example table t

	Firm	Sector	Style	Revenue	Earnings
0	A	Healthcare	Growth	800	200
1	B	Industrials	Growth	1600	400
2	C	Healthcare	Value	500	100
3	D	Healthcare	Growth	4000	800
4	E	Industrials	Value	6000	600
5	F	Industrials	Value	400	5000

2 groupby('Sector').sum()

Sector	Revenue	Earnings
Healthcare	5300	1100
Industrials	8000	6000

3 groupby('Sector', as_index=False).sum()

	Sector	Revenue	Earnings
0	Healthcare	5300	1100
1	Industrials	8000	6000

4 groupby(['Sector','Style']).sum()

Sector	**Style**	Revenue	Earnings
Healthcare	**Growth**	4800	1000
	Value	500	100
Industrials	**Growth**	1600	400
	Value	6400	5600

5 groupby(['Sector','Style']).sum().unstack()

	Revenue		Earnings	
Style	Growth	Value	Growth	Value
Sector				
Healthcare	4800	500	1000	100
Industrials	1600	6400	400	5600

6 groupby(['Sector','Style']).Earnings.sum().unstack()

Style	Growth	Value
Sector		
Healthcare	1000	100
Industrials	400	5600

7 sort_values(['Sector','Earnings'])

	Firm	Sector	Style	Revenue	Earnings
2	C	Healthcare	Value	500	100
0	A	Healthcare	Growth	800	200
3	D	Healthcare	Growth	4000	800
1	B	Industrials	Growth	1600	400
4	E	Industrials	Value	6000	600
5	F	Industrials	Value	400	5000

8 sort_values(['Sector','Earnings']).groupby(['Sector']).first()

	Firm	Style	Revenue	Earnings
Sector				
Healthcare	C	Value	500	100
Industrials	B	Growth	1600	400

4.9 Get All Filing Dates

We want to generate a table that contains every filing date for every firm in our SEC database.
For exanple, let's get data for quarter two, 2010::

```
directory = 'data/sec/merged/'
filename  = '2010q2.csv'
data      = pd.read_csv(directory+filename, parse_dates=['filed','ddate'])
```

(Table 1). This table contains every item that any firm filed in Q2 2010. Select one row for each firm (cik) and filing:

```
data.groupby(['cik','filed']).first()
```

(Table 2). Now we have a table with one row for each firm and filing and "cik" and "filed" as the row indexes. We would like to have "cik" and "filed" as columns and we don't need any of the other columns, so we write:

```
data.groupby(['cik','filed'], as_index=False).first()[['cik','filed']]  # cik and filed as columns.
```

(Table 3). And now repeat this procedure for multiple data files like this:

```
results = pd.DataFrame()     # Create an empty table where we put the results from each data file.

filename  = '2010q1.csv'
data      = pd.read_csv(directory+filename, parse_dates=['filed','ddate'])
results   = results.append( data.groupby(['cik','filed'], as_index=False).first()[['cik','filed']] )

filename  = '2010q2.csv'
data      = pd.read_csv(directory+filename, parse_dates=['filed','ddate'])
results   = results.append( data.groupby(['cik','filed'], as_index=False).first()[['cik','filed']] )
```

Now sort the results by firm and filing date and set the "cik" as the index:

```
results = results.sort_values(['cik','filed']).set_index('cik')
```

The table "results" now contains all filing dates for all firms that filed in the first two quarters of 2010.
For example, get the filing dates for Apple (cik = 320193):

```
results.loc[320193]          # 2010-01-25, 2010-04-21.
```

(Table 4). Figure 5 shows how we can put all of this into a function "get_all_filing_dates()":

- The function takes an optional argument "filename" (for example '2010q1.csv').
- In line 4 we define a variable "filenames" as a list containig the provided filename (if provided) or a list of all filenames in the "merged" directory.
- In line 5 we filter out any file names that start with a "." (some operating systems automatically create hidden files starting with ".").
- In line 9 we loop over the list of file names and append the filing dates for each firm (cik) to the table "results".

Create a folder "dates" inside your "data/sec" directory and use the function like this:

```
filing_dates = get_all_filing_dates()
filing_dates.to_csv('data/sec/dates/filing_dates.csv')   # Save the filing_dates table.
```

And now we can read the table like this:

```
filing_dates = pd.read_csv('data/sec/dates/filing_dates.csv', index_col='cik', parse_dates=['filed'])
```

1 data

	cik	sic	countryinc	tag	filed	ddate	qtrs	value
0	7084	2070	US	EarningsPerShareBasic	2010-02-08	2008-12-31	1	0.90
1	7084	2070	US	EarningsPerShareBasic	2010-02-08	2008-12-31	2	2.52
2	7084	2070	US	EarningsPerShareBasic	2010-02-08	2009-12-31	1	0.88
3	7084	2070	US	EarningsPerShareBasic	2010-02-08	2009-12-31	2	1.65
4	7084	2070	US	EarningsPerShareDiluted	2010-02-08	2008-12-31	1	0.90
⋮								

2 data.groupby(['cik','filed']).first()

		sic	countryinc	tag	ddate	qtrs	value
cik	**filed**						
1800	**2010-02-19**	2834	US	AccountsPayableTradeCurrent	2007-12-31	0	1.219529e+09
2969	**2010-01-26**	2810	US	AccountsPayableAndAccruedLiabilitiesCurrent	2009-09-30	0	1.660400e+09
3673	**2010-03-01**	4911	US	AccountsPayableCurrent	2008-12-31	0	3.680000e+08
4281	**2010-02-18**	3350	US	AccountsPayableCurrent	2008-12-31	0	2.518000e+09
4447	**2010-02-26**	2911	US	AccountsPayableCurrent	2008-12-31	0	5.045000e+09
⋮							

3 data.groupby(['cik','filed'], as_index=False).first()[['cik','filed']]

	cik	filed
0	1800	2010-02-19
1	2969	2010-01-26
2	3673	2010-03-01
3	4281	2010-02-18
4	4447	2010-02-26
⋮		

4 results.loc[320193]

	filed
cik	
320193	2010-01-25
320193	2010-04-21

5 Get all filing dates

```
def get_all_filing_dates(filename=None):                          # Function input: optional filename.

    directory = 'data/sec/merged/'                                # Read data from here.
    filenames = [filename] if filename else os.listdir(directory) # Supplied filename or all files in "merged" directory.
    filenames = [f for f in filenames if not f.startswith(".")]   # Exclude hidden files from file list.

    results   = pd.DataFrame()                                    # Results will be appended to this table.

    for filename in filenames:                                    # Loop over all files.
        data    = pd.read_csv(directory+filename, parse_dates=['filed','ddate'])  # Read the file.
        results = results.append( data.groupby(['cik','filed'],as_index=False).first()[['cik','filed']] )

    return results.sort_values(['cik','filed']).set_index('cik')
```

4.10 Sort SEC Data

Read example file:

```
directory = 'data/sec/merged/'
filename  = '2010q2.csv'
data      = pd.read_csv(directory+filename, parse_dates=['filed','ddate'])
```

Get all earnings data:

```
tag  = 'NetIncomeLoss'
item = data[data.tag==tag]      # Select all rows where tag equals 'NetIncomeLoss'.
```

(Table 1). Let's select data for Apple and Amazon:

```
symbols = pd.read_json('https://www.sec.gov/files/company_tickers.json').transpose().set_index('cik_str')
cik1 = symbols[symbols.ticker=='AAPL'].index[0]    # Get CIK for Apple:  320193
cik2 = symbols[symbols.ticker=='AMZN'].index[0]    # Get CIK for Amazon: 1018724

t = item[item.cik.isin([cik1,cik2])]               # Select all rows where CIK is 320193 or 1018724
```

(Table 2). Note how both firms report four different values for 'NetIncomeLoss' (this is not always four). For example, Apple's net income was 3.875B in the six months (qtrs=2) ending 2009-3-31 (ddate) and 3.074B in the three months (qtrs=1) ending 2010-3-31.
When we work with these data, we usually want to know the values for the most recent quarter (qtrs=1) and the most recent 12 months (qtrs=4). In our example both firms report the 1-quarter results but not the 4-quarter results. (For "NetIncomeLoss" firms typically report at least the 1-quarter results, but for other items firms might report anything from one to four quarters.)
To calculate the one- and four-quarter values, we start by collecting the shortest and the longest duration for every reported item. Sort the table:

```
shortest = t.sort_values(['cik','filed','ddate','qtrs'], ascending=[True,True,True,False])
```

(Table 3). Select shortes quarter and most recent ddate for every firm and every filing:

```
shortest = shortest.groupby(['cik','filed']).last()
```

(Table 4). Same for longest:

```
longest = t.sort_values(['cik','filed','ddate','qtrs'], ascending=[True,True,True,True])
longest = longest.groupby(['cik','filed']).last()
```

Now put "shorters" and "longest" next to each other:

```
shortest = shortest[['value','qtrs']]
longest  = longest[['value','qtrs']]

short_long = shortest.join(longest, lsuffix='_shortest', rsuffix='_longest')
```

(Table 5).
We can now generate this table for all firms and all filing dates and then use the shortes and the longest values to calculate any qtrs=1 and qrts=4 values that have not been reported (in our example we would need to calculate the annual value for Apple by adding the values from two previous quarters).

1 item

	cik	sic	countryinc	tag	filed	ddate	qtrs	value
43	1661920	1311.0	US	NetIncomeLoss	2020-07-02	2019-03-31	1	-58564000.0
44	1661920	1311.0	US	NetIncomeLoss	2020-07-02	2020-03-31	1	-110791000.0
285	1551906	7990.0	US	NetIncomeLoss	2020-07-02	2019-12-31	4	-5293313.0
⋮								

2 item[item.cik.isin([cik1,cik2])]

	cik	sic	countryinc	tag	filed	ddate	qtrs	value
21114	320193	3571	US	NetIncomeLoss	2010-04-21	2009-03-31	2	3.875000e+09
21134	320193	3571	US	NetIncomeLoss	2010-04-21	2010-03-31	2	6.452000e+09
21154	320193	3571	US	NetIncomeLoss	2010-04-21	2009-03-31	1	1.620000e+09
21163	320193	3571	US	NetIncomeLoss	2010-04-21	2010-03-31	1	3.074000e+09
43593	1018724	5961	US	NetIncomeLoss	2010-04-23	2009-03-31	4	6.790000e+08
43608	1018724	5961	US	NetIncomeLoss	2010-04-23	2010-03-31	4	1.024000e+09
43624	1018724	5961	US	NetIncomeLoss	2010-04-23	2009-03-31	1	1.770000e+08
43651	1018724	5961	US	NetIncomeLoss	2010-04-23	2010-03-31	1	2.990000e+08

3 sort_values(['cik','filed','ddate','qtrs'], ascending=[True,True,True,False])

	cik	sic	countryinc	tag	filed	ddate	qtrs	value
21125	320193	3571	US	NetIncomeLoss	2010-04-21	2009-03-31	2	3.875000e+09
21165	320193	3571	US	NetIncomeLoss	2010-04-21	2009-03-31	1	1.620000e+09
21145	320193	3571	US	NetIncomeLoss	2010-04-21	2010-03-31	2	6.452000e+09
21174	320193	3571	US	NetIncomeLoss	2010-04-21	2010-03-31	1	3.074000e+09
43644	1018724	5961	US	NetIncomeLoss	2010-04-23	2009-03-31	4	6.790000e+08
43675	1018724	5961	US	NetIncomeLoss	2010-04-23	2009-03-31	1	1.770000e+08
43659	1018724	5961	US	NetIncomeLoss	2010-04-23	2010-03-31	4	1.024000e+09
43702	1018724	5961	US	NetIncomeLoss	2010-04-23	2010-03-31	1	2.990000e+08

4 shortest.groupby(['cik','filed']).last()

		sic	countryinc	tag	ddate	qtrs	value
cik	**filed**						
320193	**2010-04-21**	3571	US	NetIncomeLoss	2010-03-31	1	3.074000e+09
1018724	**2010-04-23**	5961	US	NetIncomeLoss	2010-03-31	1	2.990000e+08

5 shortest.join(longest, lsuffix='_shortest', rsuffix='_longest')

cik	**filed**	value_shortest	qtrs_shortest	value_longest	qtrs_longest
320193	**2010-04-21**	3.074000e+09	1	6.452000e+09	2
1018724	**2010-04-23**	2.990000e+08	1	1.024000e+09	4

4.11 Get Items From Merged SEC Files

Suppose we want to get these tags:

```
tags = ['NetIncomeLoss', 'ResearchAndDevelopmentExpense']
```

We want to collect all income and R&D values for all firms for all files in our "data/sec/merged" directory.
We will put our results in two tables, one table for each tag. Let's start by creating the empty tables and collecting them in a dictionary like this:

```
results = {t:pd.DataFrame() for t in tags}    # Dictionary of tables.
```

Now we have a dictionary with two empty tables. For example we can get the table for NetIncomeLoss like this:

```
results['NetIncomeLoss']                      # Returns NetIncomeLoss table (still empty at this point)
```

Now we will

1. loop over all files in "data/sec/merged",
2. for each file loop over all tags and write the data in the corresponding table.

Figure 1 shows how we can put all this code into a function "get_items_from_SEC_files()":

- The function takes a list of tags and an optional file name as input (Line 1).
- Lines 3-5 determine which files to loop over (identical to "get_all_filing_dates()" on page 17).
- In Line 7 we create the empty tables for our tags.
- Then we loop over all file names (Line 9) and read every file (Line 11).
- For each file we loop over all tags (Line 13).
- Line 14-19 contains the sorting code from page 19.
- In Line 20 we append the data for the current tag and current file to the "results" table.
- After getting the data from all files we sort each table by filing date (Line 23).

And now we can use the function like this:

```
get_items_from_SEC_files(tags, '2010q1.csv')              # Get items from specific file.
```

Or get all items from all files:

```
items = get_items_from_SEC_files(tags)                    # Get items from all files.
```

(this will take a while). Download the SEC ticker symbols table so we can find the CIK of specific companies:

```
symbols = pd.read_json('https://www.sec.gov/files/company_tickers.json').transpose().set_index('cik_str')
```

And now we can get all values for Apple like this:

```
cik = symbols[symbols.ticker=='AAPL'].index[0]            # 320193.

items['ResearchAndDevelopmentExpense'].loc[cik]
items['NetIncomeLoss']                 .loc[cik]
```

(Tables 2 and 3).

1 Get items function

```
import pandas as pd
import os
```

```
def get_items_from_SEC_files(tags, filename=None):                # Function input: list of tags, optional filename.

    directory = 'data/sec/merged/'                                # Read data from here.
    filenames = [filename] if filename else os.listdir(directory) # Supplied filename or all files in "merged" directory.
    filenames = [f for f in filenames if not f.startswith(".")]   # Exclude hidden files from file list.

    results   = {t:pd.DataFrame() for t in tags}                  # Dictionary of tables (1 table for each tag)

    for filename in filenames:                                    # Loop over all files.
        print(filename)
        data = pd.read_csv(directory+filename, parse_dates=['filed','ddate'])  # Read the file.

        for t in tags:                                            # Loop over all tags.
            item  = data[data.tag==t]                             # Select all data for this tag.
            short = item.sort_values(['cik','filed','ddate','qtrs'], ascending=[True,True,True,False]) # Samllest qrts.
            long  = item.sort_values(['cik','filed','ddate','qtrs'], ascending=[True,True,True,True])  # Largest  qtrs.
            short = short.groupby(['cik','filed']).last()[['value','qtrs']]     # One value for each firm and filing.
            long  = long .groupby(['cik','filed']).last()[['value','qtrs']]
            short_long = short.join(long, lsuffix='_shortest', rsuffix='_longest') # Put shortest and longest next to each other.
            results[t] = results[t].append( short_long )

    for t in tags:                                                # Now sort all tables by filing date.
        if not results[t].empty: results[t] = results[t].sort_index(level='filed')

    return results
```

```
tags = ['NetIncomeLoss', 'ResearchAndDevelopmentExpense']

items = get_items_from_SEC_files(tags, '2010q1.csv')              # Get items from specific file.
```

```
2010q1.csv
```

```
items = get_items_from_SEC_files(tags)                            # Get items from all files.
```

```
2018q4.csv
2018q3.csv
2018q2.csv
2021_01.csv
2018q1.csv
2020q2.csv
2020q3.csv
```

2 items['ResearchAndDevelopmentExpense']

cik	filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
883984	**2009-04-23**	738000.0	1	7.380000e+05	1
884905	**2009-04-29**	18000000.0	1	1.800000e+07	1
1164727	**2009-04-30**	31000000.0	1	3.100000e+07	1
1080224	**2009-05-14**	558000.0	1	5.580000e+05	1
38074	**2009-05-29**	661294000.0	4	6.612940e+08	4
769397	**2009-06-03**	124100000.0	1	1.241000e+08	1
875045	**2009-07-17**	416453000.0	1	6.959310e+08	2
12927	**2009-07-22**	960000000.0	1	1.930000e+09	2
⋮					

2 Apple ResearchAndDevelopmentExpense

filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
2009-07-22	3.410000e+08	1	9.750000e+08	3
2009-10-27	1.333000e+09	4	1.333000e+09	4
2010-01-25	3.980000e+08	1	3.980000e+08	1
2010-04-21	4.260000e+08	1	8.240000e+08	2
2010-07-21	4.640000e+08	1	1.288000e+09	3
2010-10-27	1.782000e+09	4	1.782000e+09	4
2011-01-19	5.750000e+08	1	5.750000e+08	1
2011-04-21	5.810000e+08	1	1.156000e+09	2
⋮				

3 Apple NetIncomeLoss

filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
2009-07-22	1.229000e+09	1	4.039000e+09	3
2009-10-27	5.704000e+09	4	5.704000e+09	4
2010-01-25	3.378000e+09	1	3.378000e+09	1
2010-04-21	3.074000e+09	1	6.452000e+09	2
2010-07-21	3.253000e+09	1	9.705000e+09	3
2010-10-27	4.308000e+09	1	1.401300e+10	4
2011-01-19	6.004000e+09	1	6.004000e+09	1
2011-04-21	5.987000e+09	1	1.199100e+10	2
⋮				

4.12 Reporting Frequencies

Get items:

```
tags  = ['ResearchAndDevelopmentExpense']
items = get_items_from_SEC_files(tags)                          # items is a dictionary of tables.
item  = items['ResearchAndDevelopmentExpense']                  # Table with all R&D values.
```

Apple R&D:

```
symbols = pd.read_json('https://www.sec.gov/files/company_tickers.json').transpose().set_index('cik_str')
cik     = symbols[symbols.ticker=='AAPL'].index[0]              # 320193

item.loc[cik] [['value_shortest','qtrs_shortest']]
```

(Table 1). Note how Apple reports a 4-quarter R&D value every fourth quarter (in their 10-K). So if we want to know the 1-quarter value, we have to subtract the previous three quarters.
For example for 2010-10-27 (Row 4 in Table 1):

```
date   = '2010-10-27'
values = item.loc[cik].value_shortest
previous_values = values[:date][-4:-1]                          # The 3 values before 2010-10-27.
value_1_qtr = item.loc[cik,date] - previous_values.sum()        # = 1.78B - (464M + 426M + 398M)
```

But this will not always work. For example, check Ford's R&D:

```
cik = symbols[symbols.ticker=='F'].index[0]                     # 37996
item.loc[cik] [['value_shortest','qtrs_shortest']]
```

Table 2 shows how Ford reports R&D only once per year (in their 10-K).
Let's compare the dates in Table 2 to all filing dates for Ford. Get filing dates for all firms (see page 17):

```
all_filing_dates = pd.read_csv('data/sec/dates/filing_dates.csv', index_col='cik', parse_dates=['filed'])
```

(Table 3). Filing dates for Ford:

```
filing_dates = all_filing_dates.loc[cik].filed
```

(Table 4). And now we can use the "reindex()" method to add these dates to the index of Table 2:

```
item.loc[cik].reindex(filing_dates) [['value_shortest','qtrs_shortest']]
```

Table 5 shows how each year has now three missing values, corresponding to the three quarters where Ford does not report R&D.
Now subtract the quarters in the same way as we did for Apple:

```
date   = '2012-02-21'
values = item.loc[cik].value_shortest.reindex(filing_dates)      # Use Table 3.
previous_values = values[:date][-4:-1]                           # The 3 values before 2012-02-21.
value_1_qtr = item.loc[cik,date] - previous_values.sum(skipna=False) # 1.78B - (NaN + NaN + NaN) = NaN
```

(without "skipna=False" inside the sum, NaN counts as 0). So now we get NaN when we try to calculate Ford's 1-quarter values. And this is the correct result, because it is not possible to calculate 1-quarter values if we only have 4-quarter values availabe, and so the result should be a missing value.
Figure 6 shows how we can apply this code to all filing dates with quarters > 1 ("qtrs[qtrs>1]"). Here we use "for i,v in series.iteritems()" to loop over the index and values of a pandas series. And we collect the previous values with "values[:date][-q-1]" (for example if q=3, we need to subtract the previous 2 quarters, so we select [-3:-1] which gives us the 2 values before the current "date").

1 Apple R&D

filed	value_shortest	qtrs_shortest
2009-07-22	3.410000e+08	1
2009-10-27	1.333000e+09	4
2010-01-25	3.980000e+08	1
2010-04-21	4.260000e+08	1
2010-07-21	4.640000e+08	1
2010-10-27	1.782000e+09	4
2011-01-19	5.750000e+08	1
2011-04-21	5.810000e+08	1
2011-07-20	6.280000e+08	1
2011-10-26	2.429000e+09	4
2012-01-25	7.580000e+08	1
2012-04-25	8.410000e+08	1
2012-07-25	8.760000e+08	1
2012-10-31	3.381000e+09	4
2013-01-24	1.010000e+09	1
2013-04-24	1.119000e+09	1
2013-07-24	1.178000e+09	1
2013-10-30	4.475000e+09	4
2014-01-28	1.330000e+09	1
2014-04-24	1.422000e+09	1
2014-07-23	1.603000e+09	1
2014-10-27	6.041000e+09	4
2015-01-28	1.895000e+09	1
2015-04-28	1.918000e+09	1
2015-07-22	2.034000e+09	1
2015-10-28	8.067000e+09	4
2016-01-27	2.404000e+09	1
2016-04-27	2.511000e+09	1
2016-07-27	2.560000e+09	1
2016-10-26	1.004500e+10	4
2017-02-01	2.871000e+09	1
2017-05-03	2.776000e+09	1
2017-08-02	2.937000e+09	1
2017-11-03	1.158100e+10	4
⋮		
2020-05-01	4.565000e+09	1
2020-07-31	4.758000e+09	1
2020-10-30	1.875200e+10	4
2021-01-28	5.163000e+09	1

2 Ford R&D

filed	value_shortest	qtrs_shortest
2011-02-28	5.000000e+09	4
2012-02-21	5.300000e+09	4
2013-02-19	5.500000e+09	4
2014-02-18	6.400000e+09	4
2015-02-13	6.900000e+09	4
2016-02-11	6.700000e+09	4
2017-02-09	7.300000e+09	4
2018-02-08	8.000000e+09	4
2019-02-21	8.200000e+09	4
2020-02-05	7.400000e+09	4

3 all_filing_dates

cik	filed
1750	2010-09-23
1750	2010-12-21
1750	2011-03-22
⋮	
1825024	2020-12-04
1825042	2020-12-29
1825079	2021-01-15

4 Ford filing_dates

cik	filed
37996	2009-08-05
37996	2009-11-06
37996	2010-02-25
37996	2010-05-07
37996	2010-08-06
37996	2010-11-08
37996	2011-02-28
37996	2011-05-10
⋮	

5 Ford R&D, reindexed

filed	value_shortest	qtrs_shortest
2009-08-05	NaN	NaN
2009-11-06	NaN	NaN
2010-02-25	NaN	NaN
2010-05-07	NaN	NaN
2010-08-06	NaN	NaN
2010-11-08	NaN	NaN
2011-02-28	5.000000e+09	4
2011-05-10	NaN	NaN
2011-08-05	NaN	NaN
2011-11-04	NaN	NaN
2012-02-21	5.300000e+09	4
2012-05-04	NaN	NaN
2012-08-03	NaN	NaN
2012-11-02	NaN	NaN
2013-02-19	5.500000e+09	4
2013-05-01	NaN	NaN
2013-07-31	NaN	NaN
2013-10-31	NaN	NaN
2014-02-18	6.400000e+09	4
2014-05-01	NaN	NaN
2014-07-31	NaN	NaN
2014-10-31	NaN	NaN
2015-02-13	6.900000e+09	4
2015-04-28	NaN	NaN
2015-07-28	NaN	NaN
2015-10-27	NaN	NaN
2016-02-11	6.700000e+09	4
2016-04-28	NaN	NaN
2016-07-28	NaN	NaN
2016-10-27	NaN	NaN
2017-02-09	7.300000e+09	4
2017-04-27	NaN	NaN
2017-07-26	NaN	NaN
2017-10-26	NaN	NaN
⋮		
2019-10-24	NaN	NaN
2020-02-05	7.400000e+09	4
2020-04-29	NaN	NaN
2020-07-31	NaN	NaN
2020-10-29	NaN	NaN

6 Loop over all quarters > 1

[29]:
```
cik = symbols[symbols.ticker=='AAPL'].index[0]
filing_dates = all_filing_dates.loc[cik].filed              # All filing dates for this firm.

values = item.loc[cik].value_shortest.reindex(filing_dates)  # Add (possibly) missing filing dates.
qtrs   = item.loc[cik].qtrs_shortest

for date,q in qtrs[qtrs>1].iteritems():                     # Loop over all dates with > 1 quarters.
    previous_values = values[:date][-q:-1]# Example: for q=3 we need to subtract 2 previous quarters.
    if len(previous_values) == q-1:                         # If all previous values available
        values[date] -= previous_values.sum(skipna=False)    # Subtract previous values.
    else:
        values[date]  = np.nan

pd.DataFrame({'value_shortest':item.loc[cik].value_shortest, 'quarterly':values, 'qtrs':qtrs})
```

[29]:

filed	value_shortest	quarterly	qtrs
2009-07-22	3.410000e+08	3.410000e+08	1
2009-10-27	1.333000e+09	NaN	4
2010-01-25	3.980000e+08	3.980000e+08	1
2010-04-21	4.260000e+08	4.260000e+08	1
2010-07-21	4.640000e+08	4.640000e+08	1
2010-10-27	1.782000e+09	4.940000e+08	4
2011-01-19	5.750000e+08	5.750000e+08	1
2011-04-21	5.810000e+08	5.810000e+08	1

4.13 Quarterly and Annual Values

Get items:

```
tags  = ['ResearchAndDevelopmentExpense']
items = get_items_from_SEC_files(tags)                          # items is a dictionary of tables.
item  = items['ResearchAndDevelopmentExpense']                  # Table with all R&D values.
```

(Table 1).
Figure 2 shows a function "calculate_quarterly_annual_values()", based on our code from page 23:

- Line 6: loop over all firms in "item",
- Line 10: collect shortest reported values together with any possibly missing filing dates,
- Line 12: loop over filings where the firm did not report a 1-quarter value,
- Line 15: subtract the previous q-1 values so get the 1-quarter value,
- Line 20: collect longest reported values together with any possibly missing filing dates,
- Line 22: loop over filings where the firm did not report a 4-quarter value,
- Line 25: add the previous 4-q quarterly values ("valuesQ") so get the 4-quarter value ("valuesA").

Use the function like this:

```
itemQA = calculate_quarterly_annual_values(item)
```

(Table 3). Create a table to compare the results to the reported values:

```
symbols = pd.read_json('https://www.sec.gov/files/company_tickers.json').transpose().set_index('cik_str')
cik     = symbols[symbols.ticker=='TSLA'].index[0]

t = pd.DataFrame()
t['values_short']     = item.loc[cik].value_shortest    # Value for shortest period the firm reported.
t['values_long']      = item.loc[cik].value_longest     # Value for longest  period the firm reported.
t['values_quarterly'] = itemQA.loc[cik].valueQ          # We calculated this value.
t['values_annually']  = itemQA.loc[cik].valueA          # We calculated this value.
t.div(10**9).plot()
```

The "spikes" in Figure 4 (red an blue lines) are the R&D values that Tesla reported. The lower and upper bounds in this graph (purple and black lines) are the quarterly and annual values that we calculated.
Then create a directory "data/sec/items" and save the data like this:

```
itemQA.to_csv('data/sec/items/RnD.csv')
```

And now we can read the data like this:

```
rnd = pd.read_csv('data/sec/items/RnD.csv', parse_dates=['filed'], index_col=['cik','filed'])
```

1 item = items['ResearchAndDevelopmentExpense']

cik	filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
883984	**2009-04-23**	738000.0	1	738000.0	1
884905	**2009-04-29**	18000000.0	1	18000000.0	1
1164727	**2009-04-30**	31000000.0	1	31000000.0	1
1080224	**2009-05-14**	558000.0	1	558000.0	1
38074	**2009-05-29**	661294000.0	4	661294000.0	4
⋮					

3 itemQA

cik	filed	valueQ	valueA
883984	**2009-04-23**	738000.0	NaN
	2009-07-24	617000.0	NaN
	2009-10-22	661000.0	NaN
	2010-02-19	629000.0	2645000.0
	2010-04-23	918000.0	2825000.0
	2010-07-23	952000.0	3160000.0
	2010-10-22	1067000.0	3566000.0
	⋮		

2 Calculate quarterly and annual values

```python
def calculate_quarterly_annual_values(item):                          # item: table with shortest and longest values and quarters.
    result           = pd.DataFrame()                                 # Results go here.
    all_firms        = item.index.get_level_values('cik').unique()    # All CIKs.
    all_filing_dates = pd.read_csv('data/sec/dates/filing_dates.csv', index_col='cik', parse_dates=['filed'])

    for cik in all_firms:                                             # Loop over all firms.
        filing_dates = pd.Series(all_filing_dates.filed[cik])         # All filing dates for this firm.

        # Quarterly values:
        valuesQ = item.loc[cik].value_shortest.reindex(filing_dates) # Values with shortest reported quarters.
        qtrsQ   = item.loc[cik].qtrs_shortest.astype(int)             # Number of quarters for each value.
        for date,q in qtrsQ[qtrsQ>1].iteritems():                     # Loop over all dates with > 1 quarters.
            previous_values = valuesQ[:date][-q:-1]                   # Example: for q=3 we need to subtract 2 previous quarters.
            if len(previous_values) == q-1:                           # If all previous values available.
                valuesQ[date] -= previous_values.sum(skipna=False)    # Subtract previous values to get quarterly value.
            else:
                valuesQ[date]  = np.nan

        # Annual values:
        valuesA = item.loc[cik].value_longest.reindex(filing_dates) # Values with longest reported quarters.
        qtrsA   = item.loc[cik].qtrs_longest.astype(int)            # Number of quarters for each value.
        for date,q in qtrsA[qtrsA<4].iteritems():                   # Loop over all dates with < 4 quarters.
            previous_values = valuesQ[:date][-4:-q]                 # Example: for q=2 we need to add quarters -3 and -4.
            if len(previous_values) == 4-q:                         # If all previous data available.
                valuesA[date] += previous_values.sum(skipna=False)  # Add previous values to get annual values.
            else:
                valuesA[date]  = np.nan

        result = result.append( pd.DataFrame({'cik':cik, 'filed':filing_dates, 'valueQ':valuesQ.values, 'valueA':valuesA.values}) )

    return result.set_index(['cik','filed'])                        # Return a table with columns 'valueQ' and 'valueA'.
```

4 Tesla R&D