Regular Expressions in DataFrames

Module 2 | Chapter 1 | Notebook 5

DataFrames don't always contain numbers exclusively. They often contain texts that need to be processed more before we can analyze them or pass them to a machine learning model. That's what this lesson is about. In this Lesson you will learn:

- How to extract data from *strings* in a `DataFrame` with *regular expressions*
- How to use *regular expressions* to replace sections of text

---

Extracting and replacing text data from DataFrames

Scenario: The Taiwanese investor from *Module 1, Chapter 1* gets in touch with you again. This time he's not interested in house prices. Instead, he wants to invest in DAX-listed companies. However, he doesn't yet have enough data on the companies to make an informed decision. So he asks you to collect publicly available data on the companies and to deliver it to him in a structured format.

In the first lessons in this chapter we identified and downloaded some data with a web scraper. We then saved this as a `DataFrame`. However, a lot of the columns contain text with some data in it. We should extract the important information from the text entries and save it as separate columns to make it easier to work with.

First import the `DataFrame` from the file *company_data.p* which you have created in *Web Scraping* (module 2, chapter 1, lecture 3). Assign it to the variable `df_company`. Then print the first five rows.

In [8]:
```
import pandas as pd
df_company = pd.read_pickle('company_data.p')
df_company.head()
```

Out[8]:

key	Factory outlet in Herzogenaurach, Germany	Formerly	Company type	Traded as	Industry	Founded
Adidas	Factory outlet in Herzogenaurach, Germany	Gebrüder Dassler Schuhfabrik (1924–1949)	Public (AG)	FWB: ADSDAX component	Textile, footwear	July 1924; 99 years agoHerzogenaurach, Germany...
Airbus	NaN	.mw-parser-output .plainlist ol,.mw-parser-out...	Public (Societas Europaea)	BMAD: AIR Euronext Paris: AIR FWB: AIR CAC 40 ...	Aerospace, Defence	18 December 1970; 53 years ago
Allianz	NaN	NaN	Public (SE)	.mw-parser-output .plainlist ol,.mw-parser-out...	Financial services	05 February 1890; 134 years ago
BASF	NaN	NaN	Public (Societas Europaea)	.mw-parser-output .plainlist ol,.mw-parser-out...	Chemicals	6 April 1865; 159 years agoMannheim, Baden
Bayer	NaN	NaN	Public	.mw-parser-output .plainlist ol,.mw-parser-out...	Pharmaceuticals Chemicals Biotechnology Health...	1 August 1863; 160 years ago[1]

5 rows × 76 columns

As the investor is mainly interested in the financial figures, we'll concentrate on the following columns: `['Operating income', 'Net income', 'Revenue', 'Total assets', 'Total equity']`. Remove the remaining columns. Then print the first five rows to check it.

In [9]:
```
df_company = df_company.loc[:, ['Operating income', 'Net income', 'Revenue', 'Total as
df_company.head()
```

Out[9]:

key	Operating income	Net income	Revenue	Total assets	Total equity
Adidas	€2.368 billion (2018)[3]	€1.702 billion (2018)[3]	€21.915 billion (2018)[3]	€15.612 billion (2018)[3]	€6.364 billion (2018)[3]
Airbus	€4.60 billion (2023)	€3.79 billion (2023)	€65.45 billion (2023)	€118.87 billion (2023)	€17.73 billion (2023)
Allianz	€14.16 billion (2022)	€7.18 billion (2022)	€152.7 billion (2022)	€1.02 trillion (2022)	€51.0 billion (2022)
BASF	€6.55 billion (2022)[1]	€−627 million (2022)[1]	€87.3 billion (2022)[1]	€84.5 billion (2022)[1]	€40.9 billion (2022)[1]
Bayer	€7.01 billion (2022)[2]	€4.15 billion (2022)[2]	€50.74 billion (2023)[2]	€124.9 billion (2022)[2]	€38.93 billion (2022)[2]

Now `df_company` should look something like this:

Key financial figures

The key figures are displayed as texts. They consist of the actual amount, with a currency symbol and a word (e.g. `'billion'`), as well as a year and a note (e.g. `'[1]'`). So the data doesn't yet follow the *tidy data principles*. So now we'll create two columns from each column - one for the amount and one for the year. We'll use *regular expressions* to do this. The following tables contain the most important ones:

Generalizations

Expression	Meaning
.	any character (except line break)
\d	digit
\w	letter, digit or underscore
\s	*Whitespace*: space, tab, line break

Repetitions

Expression	Meaning
{min, max}	Repetition of the preceding expression at least min-times and no more than max-times
{3}	Repeat the preceding expression exactly 3 times
+	Repeat the preceding expression at least once
*	Repeat the preceding expression at least 0 times

Combinations

Expression	Meaning
...|...	or operator: Matches the expression to the left or right of |
[...]	or operator: Matches an expression within the brackets
[a-z]	from-to operator: Only matches a lower case letter between a and z
[A-Z0-9]	from-to-or operator: Matches a capital letter between A and Z or a digit between 0 and 9
[^...]	Except: Only matches an expression not listed in the brackets
(...)	*capture group*: All expressions within the parentheses are searched for together and form a separate result
?	Optional: The preceding expression may occur exactly once or zero times
\	*Escape*: The following character is searched for as it is (e.g. \. searches for a period)

Boundaries

Expression	Meaning
^	Matches the beginning of the string
\$	Matches the end of the string
\b	Matches the beginning or the end of a word

Working with text in a `DataFrame` can often be a long-winded process. But thankfully, `pandas` also gives us the ability to use regular expressions. You can use the `my_Series.str.extract()` function to do this. It expects a *regex* with at least one capture group. You create these by putting parentheses around the search pattern. So if you're looking for a number, the capture group will look like this: `r'(\d)'`. If the search pattern finds more matches, it only outputs the first result.

Try it out and extract the year from the `'Total assets'` column. Store it as a new column `'Total assets year'` and look at it to check the result.

Tip: In our case, the numbers of the amounts have a maximum of 3 digits before and after the decimal point.

Important: `my_series.str.extract()` returns a `DataFrame` by default. It contains one column for each *capture group* in the search pattern. As a result, the column you get back can't be added to without using additional functions. So use the parameter `expand=False`. This ensures that the output is a `Series`.

In [11]:
```
df_company.loc[:, 'Total assets year'] = df_company.loc[:, 'Total assets'].str.extract
df_company.loc[:, 'Total assets year']
```

Out[11]:
```
Adidas                    2018
Airbus                    2023
Allianz                   2022
BASF                      2022
Bayer                     2022
Beiersdorf                2019
BMW                       2022
Brenntag                   NaN
Commerzbank               2018
Continental_AG            2022
Covestro                  2022
Daimler_Truck              NaN
Deutsche_Bank             2023
Deutsche_B%C3%B6rse       2022
Deutsche_Post              NaN
Deutsche_Telekom          2022
E.ON                      2022
Hannover_Re                NaN
Henkel                    2023
Infineon_Technologies     2023
Mercedes-Benz_Group       2021
Merck_Group               2023
MTU_Aero_Engines          2019
Munich_Re                 2022
Porsche                   2022
Porsche_SE                2021
Qiagen                    2023
Rheinmetall               2022
RWE                       2019
SAP                       2023
Sartorius_AG               NaN
Siemens                   2023
Siemens_Healthineers      2022
Symrise                   2022
Volkswagen_Group          2023
Vonovia                   2020
Zalando                   2022
Name: Total assets year, dtype: object
```

Now we just need the amount in a separate column. Extracting this is a little trickier. It's best to also take the words (such as `'billion'`) so that we can replace them later with the corresponding numbers. We can do without the currency symbol for now, but, we should keep in mind that not all the amounts are necessarily stated in euros. Store the result as a new column named `'Total assets value'`.

Tips: You can use square brackets to indicate that only one of the characters enclosed in them has to match. For example, with `r'[\d.,]+'` you can get all the numbers separated by dots or commas. If you only want letters, you also use square brackets. However, you don't have to insert each letter individually, but can use a dash to mean "from-to". For example, `r'[a-zA-Z]'` applies to all lower and upper case letters between a and z. With `r'?'` you can define the preceding part of the search pattern as optional. If you are not sure if there is a space between the number and the word, you can cover both eventualities by using `'\s?'`.

In [13]:
```
df_company.loc[:, 'Total assets value'] = df_company.loc[:, 'Total assets'].str.extrac
df_company.loc[:, 'Total assets value']
```

Out[13]:

```
Adidas                      15.612 billion
Airbus                      118.87 billion
Allianz                      1.02 trillion
BASF                          84.5 billion
Bayer                        124.9 billion
Beiersdorf                    9.63 billion
BMW                        246.926 billion
Brenntag                             NaN
Commerzbank                   462 billion
Continental_AG           37,926.7 million
Covestro                     14.6 billion
Daimler_Truck                        NaN
Deutsche_Bank                1.31 trillion
Deutsche_B%C3%B6rse          269.1 billion
Deutsche_Post                        NaN
Deutsche_Telekom             298.6 billion
E.ON                       134.009 billion
Hannover_Re                          NaN
Henkel                      17.965 billion
Infineon_Technologies       28.439 billion
Mercedes-Benz_Group          258.8 billion
Merck_Group                  48.49 billion
MTU_Aero_Engines             7.765 billion
Munich_Re                    298.5 billion
Porsche                     47.673 billion
Porsche_SE                  42.533 billion
Qiagen                        6.12 billion
Rheinmetall                  8.089 billion
RWE                         39.846 billion
SAP                         68.291 billion
Sartorius_AG                         NaN
Siemens                    145.067 billion
Siemens_Healthineers        33.614 billion
Symrise                       7.78 billion
Volkswagen_Group           630.826 billion
Vonovia                   58,910.7 million
Zalando                      7.626 billion
Name: Total assets value, dtype: object
```

Now in the `'Total assets value'` column, we have numbers and words that form part of the number. The words in our case are `'million'`, `'billion'`, `'trillion'` and `'bn'`. To be able to convert the column into numbers, we should replace these values. We can replace them with the scientific notation `1e9` and `1e12`, and then the numbers can easily be converted into floating point numbers. We also need to fix the thousand separator `,`, which has been used by several companies, such as Vonovia. We'll replace the thousand separator `,` with an empty string without spaces. This will delete the comma. Thankfully `my_series.str.replace()` can also handle *regexes*. For example, you can use the code `'r'\s'` to replace the space character.

Run the following code cell to replace the comma and the words `'million'`, `'billion'`, `'trillion'` and `'bn'`. If there are more numerical values, you can insert the corresponding replacement here. At the end of the cell, the column is converted into a `float` column.

In [14]:

```
df_company.loc[:,'Total assets value'] = df_company.loc[:,'Total assets value'].str.re
df_company.loc[:,'Total assets value'] = df_company.loc[:,'Total assets value'].str.re
```

```
df_company.loc[:,'Total assets value'] = df_company.loc[:,'Total assets value'].str.re
df_company.loc[:,'Total assets value'] = df_company.loc[:,'Total assets value'].str.re
df_company.loc[:,'Total assets value'] = df_company.loc[:,'Total assets value'].str.re
df_company.loc[:,'Total assets value'] = df_company.loc[:,'Total assets value'].astype
df_company.loc[:,'Total assets value']
```

Out[14]:

```
Adidas                    1.561200e+10
Airbus                    1.188700e+11
Allianz                   1.020000e+12
BASF                      8.450000e+10
Bayer                     1.249000e+11
Beiersdorf                9.630000e+09
BMW                       2.469260e+11
Brenntag                           NaN
Commerzbank               4.620000e+11
Continental_AG            3.792670e+10
Covestro                  1.460000e+10
Daimler_Truck                      NaN
Deutsche_Bank             1.310000e+12
Deutsche_B%C3%B6rse       2.691000e+11
Deutsche_Post                      NaN
Deutsche_Telekom          2.986000e+11
E.ON                      1.340090e+11
Hannover_Re                        NaN
Henkel                    1.796500e+10
Infineon_Technologies     2.843900e+10
Mercedes-Benz_Group       2.588000e+11
Merck_Group               4.849000e+10
MTU_Aero_Engines          7.765000e+09
Munich_Re                 2.985000e+11
Porsche                   4.767300e+10
Porsche_SE                4.253300e+10
Qiagen                    6.120000e+09
Rheinmetall               8.089000e+09
RWE                       3.984600e+10
SAP                       6.829100e+10
Sartorius_AG                       NaN
Siemens                   1.450670e+11
Siemens_Healthineers      3.361400e+10
Symrise                   7.780000e+09
Volkswagen_Group          6.308260e+11
Vonovia                   5.891070e+10
Zalando                   7.626000e+09
Name: Total assets value, dtype: float64
```

Now we have the years and the values in two separate columns. There is one piece of information that we've not yet taken into account: the currency. Almost all of our values are in EUR. However, there are also values which are given in USD. To make the data directly comparable, we should convert all the values into the same currency. Since most of the information is already in â‚¬, it makes sense to convert everything to this. First we need a mask to contains the \$ values.

The `my_series.str.startswith()` method will help us to create the mask. This method does not use *regex* but is still helpful. We give it a *string* and get a `Series` back, indicating whether the corresponding row begins with this *string*. Just like `my_series.str.extract()` and `my_series.str.replace()`, missing values are ignored and remain as they are. However,

you can replace them by using the `na` parameter. Create a mask indicating which cells begin with dollar symbols \$. Store it as `mask_dollar`. Use the parameter `na=False` to replace the missing values. Otherwise, they will generate errors when we use the mask to select rows. You can use `my_series.str.strip()` to remove spaces before or after the string.

In [15]:
```
mask_dollar = df_company.loc[:, 'Total assets'].str.strip().str.startswith('$', na=Fal
mask_dollar
```

Out[15]:
```
Adidas                   False
Airbus                   False
Allianz                  False
BASF                     False
Bayer                    False
Beiersdorf               False
BMW                      False
Brenntag                 False
Commerzbank              False
Continental_AG           False
Covestro                 False
Daimler_Truck            False
Deutsche_Bank            False
Deutsche_B%C3%B6rse      False
Deutsche_Post            False
Deutsche_Telekom         False
E.ON                     False
Hannover_Re              False
Henkel                   False
Infineon_Technologies    False
Mercedes-Benz_Group      False
Merck_Group              False
MTU_Aero_Engines         False
Munich_Re                False
Porsche                  False
Porsche_SE               False
Qiagen                   False
Rheinmetall              False
RWE                      False
SAP                      False
Sartorius_AG             False
Siemens                  False
Siemens_Healthineers     False
Symrise                  False
Volkswagen_Group         False
Vonovia                  False
Zalando                  False
Name: Total assets, dtype: bool
```

Now we can use the mask to convert the dollar values in the `'Total assets value'` column using the exchange rate. Let's assume that one dollar is about 0.86 euros. So replace the dollar values with 0.86 times the value.

In [16]:
```
df_company.loc[mask_dollar, 'Total assets value'] = df_company.loc[mask_dollar, 'Total
df_company.loc[mask_dollar, 'Total assets value']
```

Out[16]:
```
Series([], Name: Total assets value, dtype: float64)
```

Previously, we split the data from the `'Total assets'` column into one column containing the year and one column showing the amount. We did this in the following steps:

- Extract the year with a *regex* and store it in a new column
- Extract the amount and the corresponding number names (i.e. billion) and store it in a new column
- Convert the number names into the scientific notation
- Replace the decimal comma with a decimal point
- Convert the columns into a numerical format
- Select the \$ values and convert them into â‚¬ with an exchange rate of 0.86

Now we have to repeat this for the following columns: `['Operating income', 'Net income', 'Revenue', 'Total equity']`. We recommend using a loop.

Tip: You can create a new column name by combining two *strings*. Assuming that the variable `column` contains the column name `'Total assets'`, `df_company.loc[:, column+' year']` will create the column name `'Total assets year'`. This operation is also called *string concatenation*.

Important: Wikipedia entries are edited by a variety of different people. People don't always follow the same conventions when they input data. Depending on how the websites are processed, it's possible that you'll receive countless error messages in the following cell because the data cleaning steps above no longer take all the exceptions into account. So be prepared to have to add new rules to your loop that aren't yet specified in the solution or hints.

In [17]:
```
# Solution:
for col in ['Operating income', 'Net income', 'Revenue', 'Total equity']:  # repeat fo
    print(col)
    df_company.loc[:, col+' year'] = df_company.loc[:, col].str.extract(r'(\d{4})', ex
    df_company.loc[:, col+' value'] = df_company.loc[:, col].str.extract(r'([\d.,]+\s?

    df_company.loc[:, col+' value'] = df_company.loc[:, col+' value'].str.replace('\st
    df_company.loc[:, col+' value'] = df_company.loc[:, col+' value'].str.replace('\sb
    df_company.loc[:, col+' value'] = df_company.loc[:, col+' value'].str.replace('bn'
    df_company.loc[:, col+' value'] = df_company.loc[:, col+' value'].str.replace('\sm
    df_company.loc[:, col+' value'] = df_company.loc[:, col+' value'].str.replace('\sM
    df_company.loc[:, col+' value'] = df_company.loc[:, col+' value'].str.replace('M',
    df_company.loc[:, col+' value'] = df_company.loc[:, col+' value'].str.replace('B',
    df_company.loc[:, col+' value'] = df_company.loc[:, col+' value'].str.replace('\se
    df_company.loc[:, col+' value'] = df_company.loc[:, col+' value'].str.replace(',',
    df_company.loc[:, col+' value'] = df_company.loc[:, col+' value'].astype(float)  #

    mask_dollar = df_company.loc[:, col].str.strip().str.startswith('$', na=False)  #
    df_company.loc[mask_dollar, col+' value'] = df_company.loc[mask_dollar, col+' valu
```

```
Operating income
Net income
Revenue
Total equity
```

Our `DataFrame` now consists of 26 rows and 15 columns. Now you've finished processing the columns for the investor. Now remove the initial columns `['Operating income', 'Net income', 'Revenue', 'Total assets', 'Total equity']` and then we're almost done. Then print the first five rows.

In [18]:
```
df_company = df_company.drop(['Operating income', 'Net income', 'Revenue', 'Total asse
df_company.head()
```

Out[18]:

key	Total assets year	Total assets value	Operating income year	Operating income value	Net income year	Net income value	Revenue year	Revenue value
Adidas	2018	1.561200e+10	2018	2.368000e+09	2018	1.702000e+09	2018	2.191500e+1(
Airbus	2023	1.188700e+11	2023	4.600000e+09	2023	3.790000e+09	2023	6.545000e+1(
Allianz	2022	1.020000e+12	2022	1.416000e+10	2022	7.180000e+09	2022	1.527000e+11
BASF	2022	8.450000e+10	2022	6.550000e+09	2022	6.270000e+08	2022	8.730000e+1(
Bayer	2022	1.249000e+11	2022	7.010000e+09	2022	4.150000e+09	2023	5.074000e+1(

`df_company` should now look something like this:

Prepared financial data

There are a few missing values, but that's not unusual with *web crawling*. The pages don't all contain the same information. Now you've summarized the key financial figures in a clear structure. Each row contains one observation. Each variable has its own column and each cell contains one value. This should make it easy for the investor to use this data in his analysis. If the this process of collecting and cleaning the data seemed like a lot of work to you, you're not mistaken. When you gather data in this way, it's rarely in the format you need it in and has to be painstakingly restructured before you can do anything else with it. Data preparation takes up a lot of time and is an important part of working with data.

Finally, save the `DataFrame` as a *pickle* named `dax_financial_data.p`.

In [19]:
```
df_company.to_pickle('dax_financial_data.p')
```

Congratulations: You've separated and prepared the information about the amount and the year for each entry with `pandas` and *regexes*. You turned unclear texts into numbers that you can now analyze and use in models. The investor is very grateful for all your hard work! Now he can use the data to identify stocks that might be of interest to him.

Remember:

- Extract text in `DataFrame` columns with `my_series.str.extract()` using regexes
- Extract text from `DataFrame` columns with `my_series.str.extract()` using regexes
- Check whether texts in the `DataFrame` columns start with a certain *string* with `my_series.str.startswith()`

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

Found a mistake? Contact Support at support@stackfuel.com.