Prof. Dr. Alexander Hillert

Textual Analysis

Chapter 4: Programming in Python

July 18, 20, and 22, 2022

WHU – Otto Beisheim School of Management

Agenda

- **Part 1:** Getting familiar with Python
- **Part 2:** Preparing documents for textual analysis
- **Part 3:** Textual analysis (tone)
- **Part 4:** Readability, language complexity, and textual similarity
- **Part 5:** Machine learning

Part I – Programming Problems

Agenda of part I

- “Problem 0”: if you have not installed Python on your computer yet, read the document “Getting_started_Python.pdf”. It shows how to install Python and how to start/use “Spyder”.

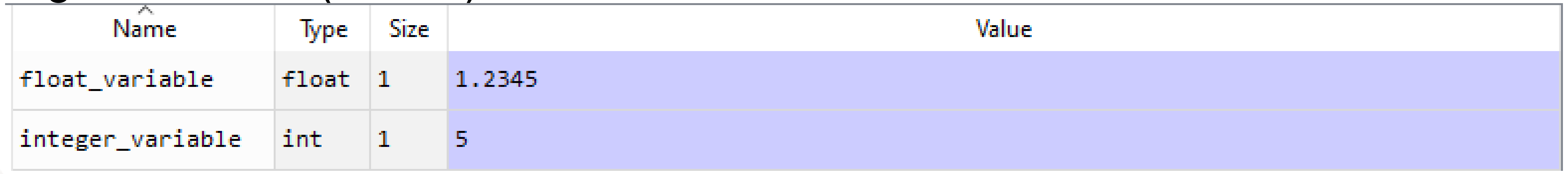
- Problem 1: basic Python commands for working with text data.

- Problem 2: identifying and downloading data from the SEC’s EDGAR system.
  - Task 1: identify the files of interest from the aggregate file list.
  - Task 2: download all files of interest.

Relevant Variable Types in Python (1)

Variable types

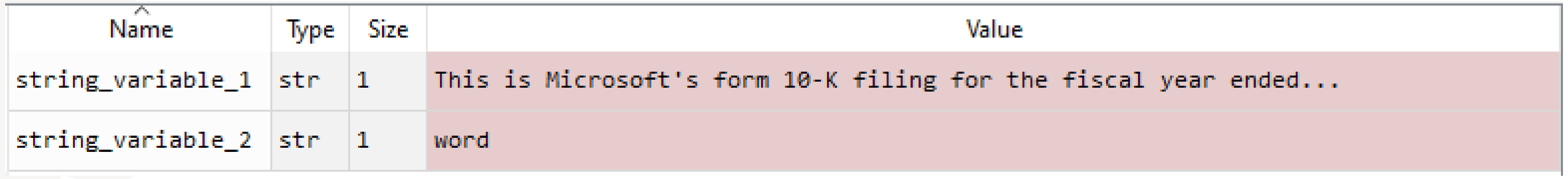
1. Integer / float = (whole) number



Name	Type	Size	Value
float_variable	float	1	1.2345
integer_variable	int	1	5

   - Relevant for us: number/percentage of X words, iterate over Y documents.

2. String = text
   - Can be entire document but also just a single word.



Name	Type	Size	Value
string_variable_1	str	1	This is Microsoft's form 10-K filing for the fiscal year ended...
string_variable_2	str	1	word

   - Relevant for us: text is main variable of interest.

Relevant Variable Types in Python (2)

Generate and replace variables

Syntax: *Variable name = value*

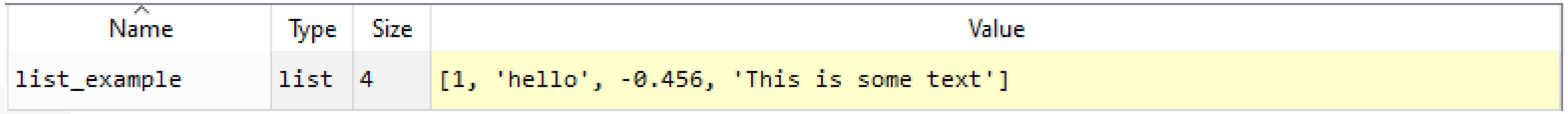
Examples:

1. i=4
2. text="hello"
3. list_var=[1,"hello","some text", -1.234]

There is no "generate" or "replace" command as in Stata.

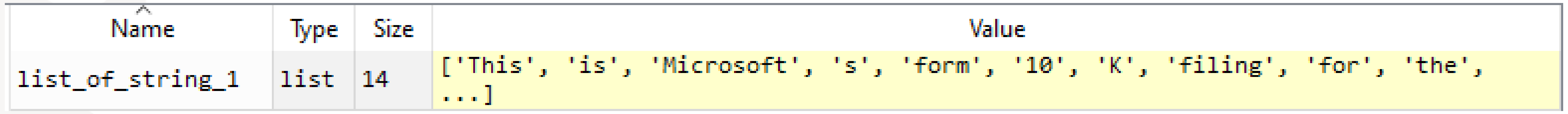
Variable types - continued

3. List = collection of variables
   - Different variable types can be combined.



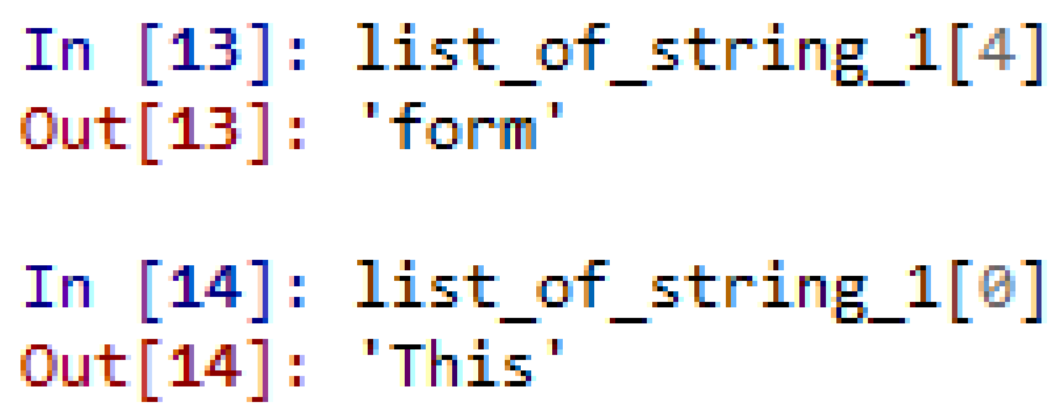
Name	Type	Size	Value
list_example	list	4	[1, 'hello', -0.456, 'This is some text']

   - 1st element: integer (1), 2nd element: string ("hello"), 3rd element: float (-0.456), 4th element: string ("This is some text").
   - <u>Relevant for us</u>: split a text into parts (paragraphs, sentences, words). Example: string_variable_1 split into words.



Name	Type	Size	Value
list_of_string_1	list	14	['This', 'is', 'Microsoft', 's', 'form', '10', 'K', 'filing', 'for', 'the', ...]

   - List elements can be called by putting [X] after the name of the list. Caution: element count starts at 0.



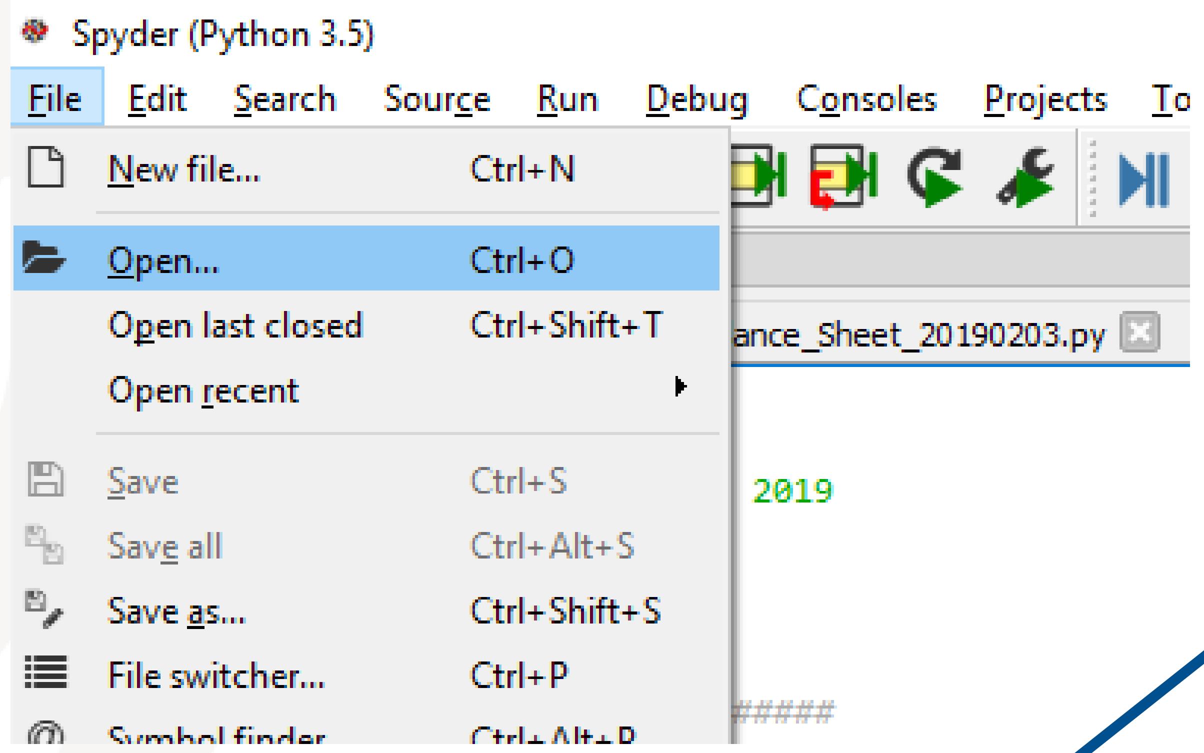
```
In [13]: list_of_string_1[4]
Out[13]: 'form'

In [14]: list_of_string_1[0]
Out[14]: 'This'
```

Problem 1: Introduction to Python (1)

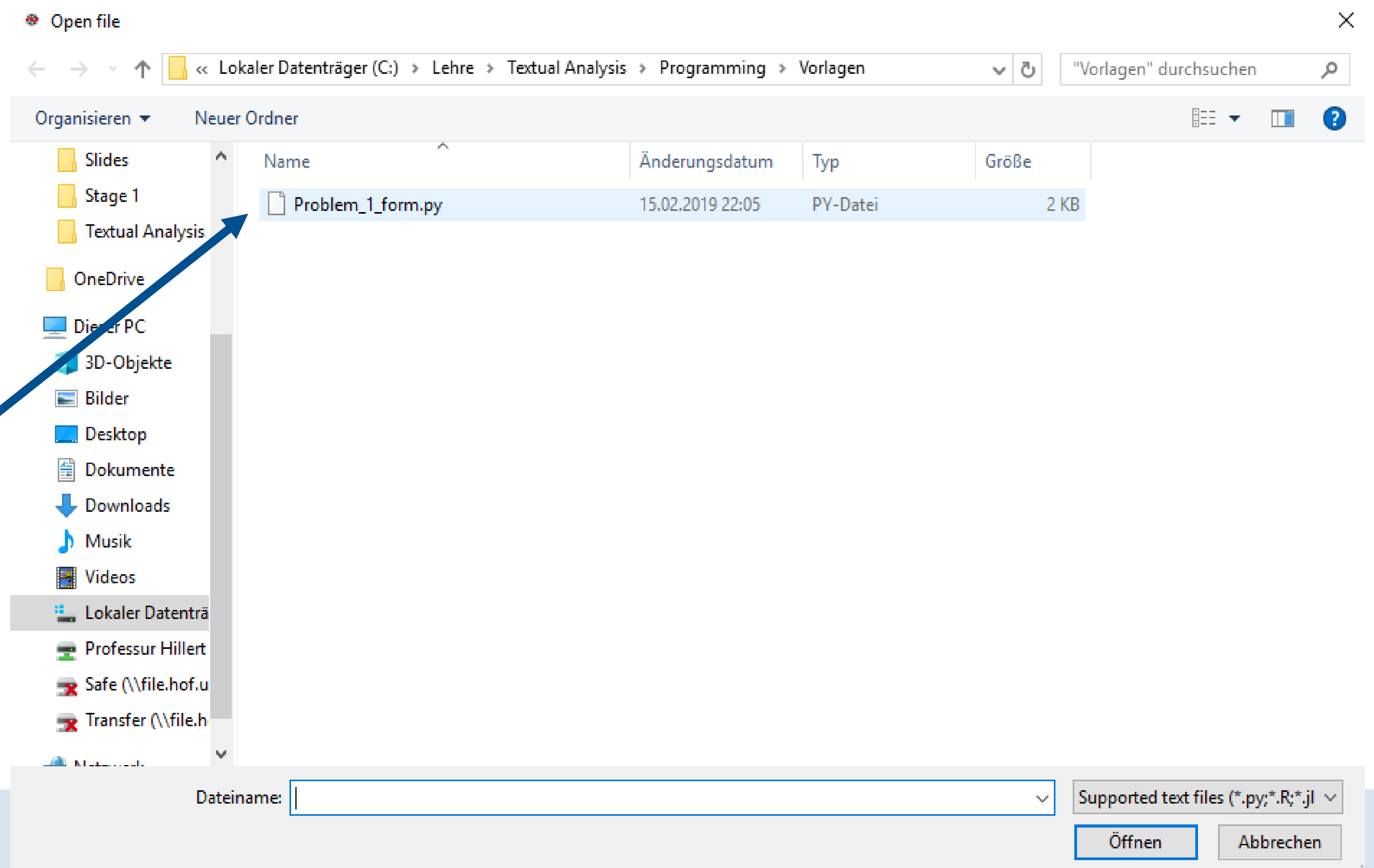
Problem 1

- covers basic commands that we will need in our textual analyses.
- consists of eight tasks that are explained on the subsequent slides.
  - ➢ Please open 'Spyder' and start working on the tasks.
  - ➢ You can use the 'Problem_1_form.py' ("Programming" → "Programming Problems – Template") and add your programming code to the template.

To open the template, click on "File" → "Open"

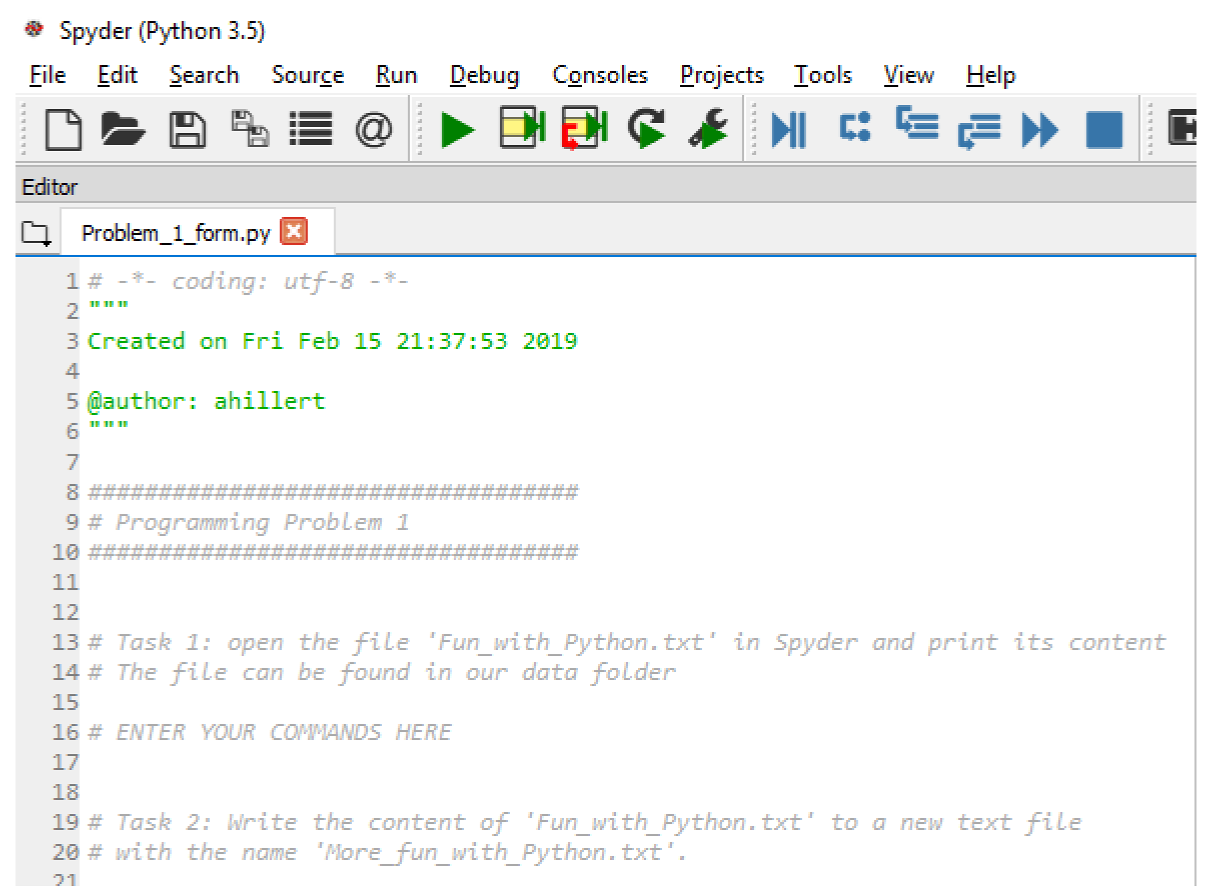


and then select the file



Name	Änderungsdatum	Typ	Größe
Problem_1_form.py	15.02.2019 22:05	PY-Datei	2 KB

→ it will show up in your editor.



Spyder (Python 3.5)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor
Problem_1_form.py

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 15 21:37:53 2019

@author: ahillert
"""

####################################
# Programming Problem 1
####################################


# Task 1: open the file 'Fun_with_Python.txt' in Spyder and print its content
# The file can be found in our data folder

# ENTER YOUR COMMANDS HERE


# Task 2: Write the content of 'Fun_with_Python.txt' to a new text file
# with the name 'More_fun_with_Python.txt'.
```

Problem 1: Introduction to Python (2)

- Python commands are *italicized*.
- When you copy commands from the slides make sure that the quotes ' are copied correctly (not ‘ or ’).
- Python accepts both single quotes ' and double quotes ".

- Open files
  - *input_file* = open('path/filename', 'r')
    Example
    input_file=open('C:/PhD Course on Textual Analysis/document.txt','r')
  - 'r' means read; 'w' means write.
  - In the write mode, Python will create the file automatically if it does not exist.
  - Open() just creates a reference to the file but is not the content of the file.

- Read content of a file
  - *input_text = input_file.read()*
    → input_text will be a string variable

'input_file' and 'input_text' are variable names. You can use whatever names you like.

- Display text: *print('Good morning'), print(input_text)*

- **Task 1:** open the file 'Fun_with_Python.txt' in Spyder and print its content.

Problem 1: Introduction to Python (3)

- Write content to a file
  - *output_file = open('path/filename', 'w')*
  - *output_file.write('Text')*

- **Task 2:** Write the text of 'Fun_with_Python.txt' to a new text file with the name 'More_fun_with_Python.txt'.

- At the end of your program code, you should close all files.
  - This is particularly important when you write something in a file.
    If you do not close the file, the content will not be saved on the hard drive.
  - Example:
    output_file.close()

Problem 1: Introduction to Python (4)

- Loops
  - While loop

    ```
    i = 1
    while i<10:
        some command(s)
        i = i +1
    ```

  - For loop

    ```
    for i in range(2,10):
        some command(s)
    ```

  - In for loops, the upper bound is not included → the example above starts at 2 and ends at 9.

> - When using while loops remember to increase the counter in each iteration (*i* = *i* +*1*)!
> - In for loops, Pythons does that automatically.

- **Task 3:** Write a loop that prints some text ten times.

Problem 1: Introduction to Python (5)

- How to <u>split text</u> in single lines?
  - *input_text_line = input_text.split('\n')*
  - Example:
    - *line_of_text = input_text.split('\n')*
    - *print(line_of_text[0])*

- How many lines of text are there?
  - *length*=len(*input_text_line*)
  - Example: print('There are :'+str(*length*)+' lines of text')

- The split commands transforms a string variable (text) to a list of individual elements.
- One can access the individual elements by the element ID ranging from 0 to N-1.
- The element ID has to be put in brackets, e.g. *list[4]*.
- The ID starts at 0, i.e. the first element of a list is *list[0]*.

- Python cannot combine text (strings) and numerical values (integers) → you need to convert numbers to text first using '*str(numerical_variable)*'.

- **Task 4:** Print the text of the "Fun_with_Python" file line by line!
More specifically, the line should read 'Line i: TEXT', where i is the number of the line and TEXT is the text from the "Fun_with_Python" file.
For example, *'Line 4: See slide 7 for help.'*

Problem 1: Introduction to Python (6)

- Count how often a word appears in a text
  - *text* = '*This is some example of some text*'
  - *text.count(*'*some*'*)*
    → result: 2

- **Task 5:** Count how often the word ‘good' appears in the document 'Fun_with_Python.txt'!

Problem 1: Introduction to Python (7)

- <u>If condition</u>
  - *if variable == value:*
    some command(s)
  - Example:

```
i = 2
if i>=1:
  print('variable i larger or equal one')
```

- The commands in the if conditions must be indented (i.e., put a tab before the command).
- There is no closing statement like 'end if' → indents indicate which commands are grouped together.

- **Task 6a:** Now, print only the lines that contain the word ‘good'!
  - Hint: combine the line-by-line printing from Task 4 with the command “.count()” from Task 5 and the if condition from this slide.

- **Task 6b:** Look at the output of Task 6a. Is there a problem?

Problem 1: Introduction to Python (8)

- **Task 6b solution:**
  - The problem is that the command ".count()" considers all potential matches, including matches within another word.
  - Example: *"The factory produces many goods.".count("good")* → result: 1 match
    "good" is part of "goods".
  - Going back to Task 5 (count how often "good" is found) shows a massive overcounting:

1. *Count how often the word 'good' appears in this text.* ✓
2. *Now, print only the lines that contain the word 'good'.* ✓
3. *Replace the word 'good' by 'excellent' and […]* ✓
4. *The factory produces many goods.* ✗
5. *Due to supply shortages the cost of goods sold […]* ✗
6. *This is not good news!* ✓

→ *Only 4 out of the 6 matches are correct.*

Problem 1: Introduction to Python (9)

- **Task 6b solution - continued:**
  - How to get an accurate count?
  - Adding leading and trailing whitespaces can help.

    → *input_text.count(' good ')* → Result: 1
  - Problem: the other three matches are enclosed by quotes, e.g., "*Count how often the word* ***'****good****'*** *appears in this text.*" → whitespaces do not match.
  - There can be further problems like the word of interest being followed by symbols ("good," or "good!").

  - **Conclusion**: We need a more general solution that takes all potential word boundaries (including " ", ".", "!", and ",") into account.

    → Regular expressions (Problem 3) allow to search for word boundaries.

Problem 1: Introduction to Python (10)

- Count how often a word appears at a specific position.
  - *text = 'This is some example, some text'*
  - *text.startswith('some')*
  - *text.endswith('some')*

- **Task 7:** Next, print only the lines that start with the word ‘This‘.
  Does it make a difference whether you search for “This” or “this”?

- *Note that “.startwith()” and “.endswith()” are subject to the same problem as “.count()”.*
- *In this specific example “.startswith(“This”)” it is less likely to result in mismatches, as there are few words other than “This” that start with “This”.*

Problem 1: Introduction to Python (11)

- How to replace text?
  - *text = 'The company sells some products.'*
  - *new_text = text.replace('some', 'any')*
  - *print(new_text)*
    → *'The company sells any products.'*

- **Task 8a:** Replace the word ‘good' by ‘excellent' and display the new text.

- **Task 8b:** Look at the new text. Is there a problem with the “.replace()” command?

Problem 1: Introduction to Python (12)

- **Task 8b solution:**
  - Yes, there is the familiar problem that matching text parts are changed no matter whether they are actual words or parts of words.
  - Examples:
    1. *The factory produces many goods.* → *The factory produces many excellents.*
    2. *Due to supply shortages the cost of goods sold […]* → *Due to supply shortages the cost of excellents sold […]*
  - We could try to fix this problem by requiring the word that should be replace to be enclosed by whitespaces. However, there are many word boundaries (incl. " ", ",", ".").
  - **Conclusion**: We need a more general solution that takes all potential word boundaries (including " ", ".", "!", and ",") into account.
    → Regular expressions (Problem 3) allow to replace words enclosed by word boundaries.

Problem 1: Introduction to Python (13)

- How to delete variables?
  - A single variable: *del variable_name*
  - All variables: *%reset*
    This command must be executed in the command window.

- Hint: Work with the Editor and not with the Python Console
  - You can save your code and modify it later if necessary.
  - To insert comments use:
    - #, .e.g., “# This line is a comment and not program code”, or
    - ''' (three single quotes) before the lines that should not be executed and ''' after the lines, e.g.,

      '''

      comment line 1
      comment line 2
      comment line 3
      '''

Problem 2: how to find and download SEC filings?

Problem 2

- Scenario:
  - Assume you want to obtain the 10-K filings that have been filed with the SEC between March 10 and March 20, 1998.
  - Task 1: How do you identify them?
  - Task 2: How do you download them automatically?

  - We will split this Problem in two parts. The first part (task 1) deals with the identification of the filings while the second part (task 2) covers the download.
  - For the second part, there are some additional commands you must learn.

As in Problem 1, you find two code templates on Moodle. The templates show the basic program structure, and your task is to fill in the missing commands. “Problem_2_SEC_Filings_Part1_Identification_form.py” and “Problem_2_SEC_Filings_Part2_Download_form.py”.

Problem 2: instructions (1)

Your task in this problem:

- In our data folder, you find the file “formidx_1998Q1.txt”. It is a copy from the “form.idx” available at https://www.sec.gov/Archives/edgar/full-index/1998/QTR1/. This file contains a list of all documents that have been filed with the SEC in the first quarter of 1998.
- In the first step, you should identify all filings that match the conditions mentioned on the previous slide.
  Please write a program that copies the information of these filings to a new csv file.
  - Column 1: Form type
  - Column 2: Company name
  - Column 3: CIK
  - Column 4: Filing date
  - Column 5: Link to the complete submission file

Problem 2: hints for task 1 (1)

Hints for this problem

Program structure

- Open the txt file with all filings.
- Create the new csv file and write the variable names in the first line.
- Go over the txt file line by line (→ use a loop) and check whether the selection criteria are satisfied.
    - If so, copy the line to the csv file and go to the next line.
    - If not, go to the next line directly.
    - How can you identify the different information items? → see next slide
- Close the files.

Problem 2: hints for task 1 (2)

Hints for this problem

- How can you identify the different information items?
  The list of filings, form.idx, from the SEC has a fixed column width for each variable.
  - Filing type: position 1 to 12
  - Company name: position 13 to 74
  - CIK: position 75 to 86
  - Filing date: position 87 to 98
  - Link: position 99 to end of line
- Python command for substrings
  - *text='This is some text '*
  - *print(text[5:7])*
  - *print(text[:5])*
  - *print(text[9:])*

- The information on the position/length of the columns is based on "human counting", i.e., the first character in a line is position 1.
- Python, however, starts counting at 0. → the first character is element 0!
- Remember that Python does not include the upper bound in intervals. E.g., *text[2:4]* gives you the third and fourth character using "human counting" and elements 2 and 3 in Python counting.

Problem 2: new commands for task 2

New commands to download files:

- There are several packages to download files.
- We use the package urllib.request.
  - *import urllib.request*
  - The command to download a file is *urllib.request.urlretrieve(URL, folder_filename)*
  - URL should contain the url, the internet address, of the file you want to download.
  - *folder_filename* is a string with the folder and filename
  - Try the following command:
    - *url='http://www.sec.gov/Archives/edgar/data/933136/0000891020-98-000348.txt'*
    - *folder_filename='C:/Courses/Textual Analysis/Washington_Mutual_10-K.txt'* Choose some folder where you have writing permission.
    - *urllib.request.urlretrieve(url, folder_filename)*

Problem 2: instructions (2)

Your task in this problem:

- In the second step, you should download the first 20 filings you have identified in first part of the Problem. (We download only the first 20 filings to save time.)
  - Create a new folder with the name "SEC Filings" in which you save the 10-K filings.
    - To create a folder:
      import os
      os.makedirs(folder), exist_ok=True)
      E.g.: *os.makedirs('C:/Courses/Textual Analysis/Problems/Task2')*
  - To be able to uniquely identify files, use a combination of CIK and the last part of the link (the part after the last /) as file name, e.g., 933136_0000891020-98-000348.txt.
  - You have to put 'http://www.sec.gov/Archives/' before the link (last column in the csv file) otherwise the url is incomplete.

Problem 2: hints for task 2

Hints for this problem:

Program structure

- Open the csv file from part one of the problem.
- Go over the csv file line by line (→ use a loop).
- Extract the CIK and link (→ last part of link → filename).
- Download the 10-K filings.
- Close the csv file.

Agenda

- **Part 1:** Getting familiar with Python
- **Part 2:** Preparing documents for textual analysis
- **Part 3:** Textual analysis (tone)
- **Part 4:** Readability, language complexity, and textual similarity
- **Part 5:** Machine learning

Introduction to Part 2 (1)

Motivation

- In the first part, we have learnt
  - The basic Python commands for handling text data.
  - How to identify SEC filings in EDGAR.
  - How to obtain filings from EDGAR.
- In this part, we will learn how to edit filings such that we can perform a textual analysis.

- What do we need to do before performing the textual analysis, i.e., the counting of sentiment words?

  → Look at the following example: 10-K of ABT Building Products Corp from March 20, 1998 (the first file in the output list from the previous Problem; filename "0000950130-98-001359.txt")

  → For a better formatted version, access the file at http://www.sec.gov/Archives/edgar/data/902476/0000950130-98-001359-index.html.

  → As a second, more recent example, consider Microsoft Corp.'s 10-Q filing from April 26, 2018.

  → Full submission file: https://www.sec.gov/Archives/edgar/data/789019/000156459018009307/0001564590-18-009307.txt

  → Main document: https://www.sec.gov/Archives/edgar/data/789019/000156459018009307/msft-10q_20180331.htm

Introduction to Part 2 (2)

Editing operations

- Identify and extract the relevant part of the complete submission file.
- Remove tables.
- Delete html code, in particular for the more recent filings.
- Remove the document header, legal disclosures, etc.

How to implement the editing? → Regular expressions

- are a powerful tool that will help us in performing the steps described above.
- are related to the commands “count()”, “replace()”, and “startswith()” from the first part. However, they are more powerful, as they allow to search for general text patterns. For example:
  - All words written in upper case
  - All words with five characters
  - Numbers

Overview of Problems in Part 2: Preparing documents for textual analysis

Programming Problems in Part 2

- Problem 3: Work through the introduction to regular expressions.
- Problem 4: Application of regular expressions on a stylized file.
- Problem 5: Preparing an exemplary SEC filing for textual analysis.
- Problem 6: Preparing a set of SEC filings for textual analysis.

→Before we turn to Problem 3, we need to discuss encoding of text files.

Encoding of SEC filings

Encoding of SEC filings

- Having obtained the SEC filings, the first step is to open the file. Even though opening a file sounds easy, one can make errors at this stage already due to an incorrect encoding format.
  - There are different encoding formats, e.g., ASCII, UTF-8.
  - Using an incorrect encoding format, results in incorrectly displayed text characters.

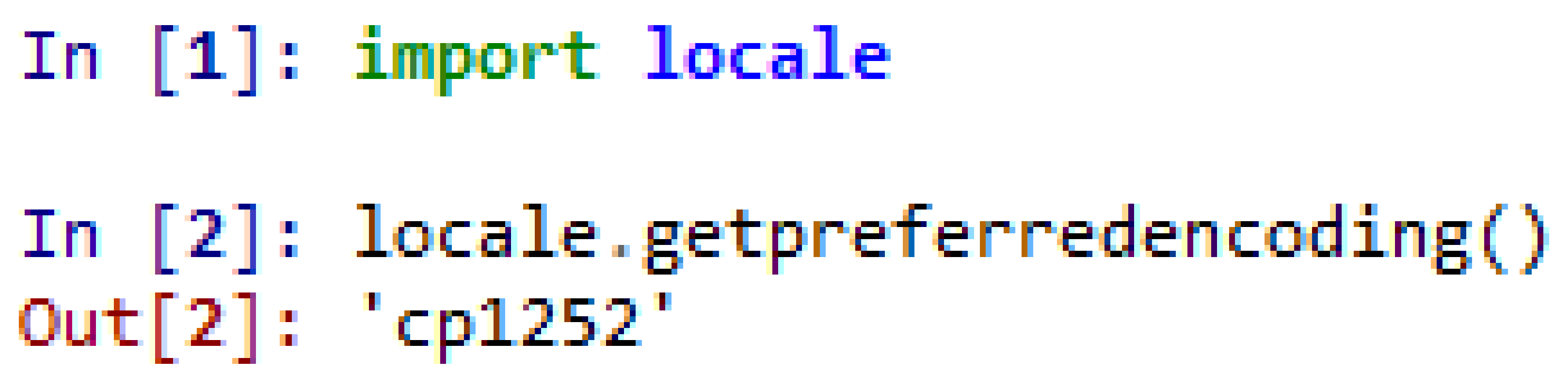
  → When working with text files, you should make sure that you use the correct format.

- The SEC requires filings to be encoded in ASCII (American Standard Code for Information Interchange) format.
- The SEC's guideline for filers is available at: https://www.sec.gov/info/edgar/quick-reference/create-ascii-files.pdf.
- The first 128 characters of Unicode correspond one-to-one with ASCII → ASCII text will be correctly encoded by UTF-8 (Unicode).

Encoding in Python (1)

Encoding in Python

- In Python you can specify how files should be encoded when opening files:
  open(file, mode=, encoding='X', errors='Y')
  - The default encoding depends on your machine's environment:
    import locale
    locale.getpreferredencoding()
    → It is better to specify the encoding explicitly.
  - Encoding: *X* can be
    - *'utf-8'*: all languages; should include all symbols
    - *'ascii'*: English language; does not include special characters é, ï, ô, etc.
    - *'cp1252'*: Western Europe
  - See https://docs.python.org/3/library/codecs.html#module-codecs for details.



```
In [1]: import locale

In [2]: locale.getpreferredencoding()
Out[2]: 'cp1252'
```

→ Recommendation: in general, use utf-8; for SEC filings, ascii also okay.

Encoding in Python (2)

Encoding in Python - continued

- *open(file, mode=, encoding='X', errors='Y')*
  error handling: *Y* can be
  - *'strict'* raises an error.
    The default (not selecting an option) has the same effect.
  - *'ignore'* ignores errors. Can lead to data loss.
  - *'replace'* causes a replacement marker (such as '?') to be inserted where there is malformed data.
  - There are further options (see https://docs.python.org/3/library/functions.html#open)

Problem 3 – Introduction to regular expressions

Problem 3: introduction to regular expressions

- In our course folder, there is a detailed introduction to regular expression: "Introduction_Regular_Expressions.py".
  The accompanying txt file ("Text_Introduction_Regular_Expressions.txt") is in the "Data" folder.
- This code presents popular regular expression commands and illustrates how the commands work using examples.
- Regular expressions (regex) are not only an essential tool for textual analysis but also for collecting unstructured data. For example, assume that you want to identify the year of birth, the university education, and/or previous employers of executives from their bios. Regex will be very helpful in this task and save you from collecting data manually.

- **Your task in Problem 3**: work thoroughly through the code of the regex introduction and learn what the commands are doing.

Problem 4 – Application of regular expressions (1)

The programming template for this problem is "Problem_4_Application_Regular_Expressions_form.py".

Problem 4

- Look at the file "Exercise_4_Application_Regular_Expressions.txt", which contains text structured in a similar way as SEC filings.
- Please complete following tasks:
  In the end, you should have a new document that only contains the relevant (main) text.

- Task 1: Delete the tables from the file.
  Hints:
  - identify the position of the start and of the end of each table and remove the text in between.
  - Commands:
    match=re.search(…) → *match.start()* and *match.end()*
    and *new_text=text[:begin_table]+text[end_table:]*

- Task 2: Delete the exhibit from the file.
  Hint: you can proceed as in task 1. You do not need to delete the <DOCUMENT> in the line before <TYPE> because we will delete all (remaining) html code in task 3 anyway.

Problem 4 – Application of regular expressions (2)

The programming template for this problem is "Problem_4_Application_Regular_Expressions_form.py".

Problem 4 - continued

- Task 3: Delete the remaining html code.
  Hint: html code starts with < and ends with >. Use *re.sub()* to identify and delete html code.

- Task 4: Delete numbers.

- Task 5: Delete symbols.
  Hints:
  - *re.sub()* is helpful here.
  - some symbols have special meanings in regular expressions. You have to put the backslash before the symbol to escape its special meaning.

- Task 6: Write the (remaining) main text to a new file.

Removing html code (1)

Professional package to remove html code

- There are cases in which a simple command like re.sub('<[^>]{1,}>', '', text) does not properly remove html code.
- The Python tool "Beautiful Soup" handles html documents very well → use it.

- If you use Anaconda/Spyder the "Beautiful Soup" package should be pre-installed.
  You can skip the subsequent slide and only need to import the package at the beginning of your code.

Removing html code (2)

Installation of "Beautiful Soup" (if not pre-installed)

Alternative 1:

- Open the command prompt (admin mode) and type "pip install beautifulsoup"

Alternative 2:

1. Download the package from https://www.crummy.com/software/BeautifulSoup/#Download. (M*ost recent version may no longer be version 4.7.1.*)
2. Unzip the installation file
   I. Unzip the downloaded file "beautifulsoup4-4.7.1.tar.gz".
   II. Go to the (in I. created) folder "beautifulsoup4-4.7.1.tar" and open the subfolder "dist" and unzip the file "beautifulsoup4-4.7.1.tar"
3. Go to the sub folder "beautifulsoup4-4.7.1.tar\dist\beautifulsoup4-4.7.1\" (in this subfolder there should be the file "setup.py") and press the shift key and do a right mouse click. Select "open command window here" ("Eingabeaufforderung hier öffnen")
4. Type "setup.py install"
5. If everything went smoothly the command prompt will display that the installation was successful.

Removing html code (3)

How to use Beautiful Soup in Python?

- Import the module at the beginning of your code „*from bs4 import BeautifulSoup*“
- Open the „input_text“ that contains html code: *html_text=BeautifulSoup(input_text, 'html.parser')*
- And have Beautiful Soup delete the html code: *text_without_html=html_text.get_text()*

Problem 4 – alternative Task 3

- Repeat task 3 (“Delete the remaining html code”) using Beautiful Soup.

Problem 5 – Prepare a 10-K filing for textual analysis

The programming template for this problem is "Problem_5_Clean_SEC_Filing_form.py".

Problem 5

Edit the 10-K of ABT Building Products Corp from March 20, 1998 (the first file in the output list from Problem 2; filename "0000950130-98-001359.txt") in a way that we could run a sentiment analysis afterwards. In the end, you should have a new txt-file with the main text.

- Major tasks: delete
  - the document header, i.e., all uninformative text at the beginning of the 10-K
  - the tables → *copy your command(s) from Problem 4*
  - the exhibits → *copy your command(s) from Problem 4*
  - the html code → *copy your command(s) from Problem 4*
- Further tasks: delete
  - numbers → *copy your command(s) from Problem 4*
  - symbols → *copy your command(s) from Problem 4*
  - single character words

Problem 6 – Prepare a set of 10-K filings for textual analysis

As Problem 6 is very similar to P5 and to save time, we will not discuss it in class. You find the solution on Moodle.

Problem 6 – Additional Problem:

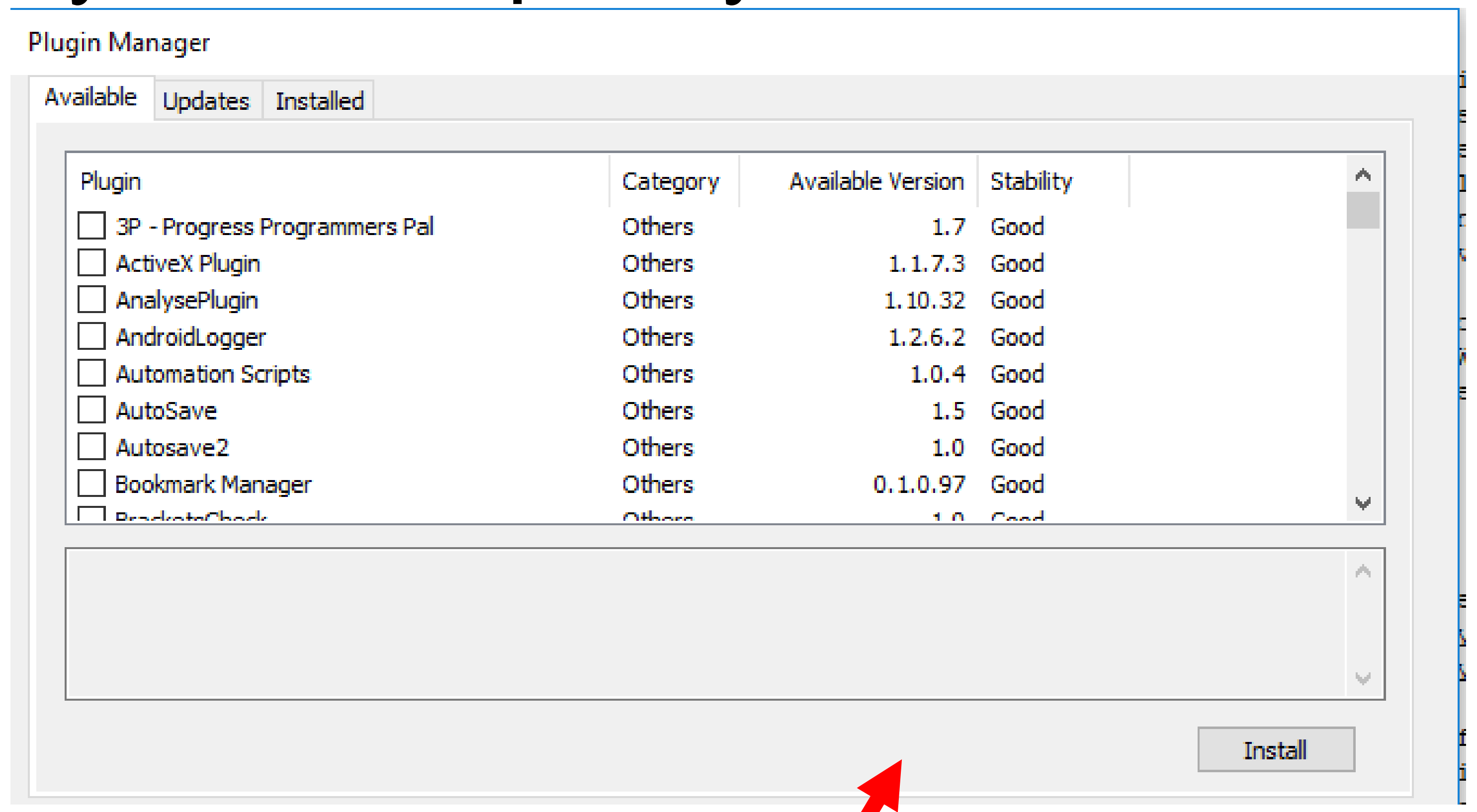
- Prepare the 200 10-Ks from the file “10-K_Sample_2011Q1_Input.csv” for a tone analysis. You have to repeat the tasks from the previous problem in a loop.
- The files are available as zip file “10-K_Sample.zip” in our course folder.
- In the end, you should have 200 new “clean” files. Save these files using the old filename and add “_clean”, e.g., “54681_0000054681-11-000009_clean.txt”

- Hints:
  - The program is very similar to the one from the previous problem.
  - However, you need a loop to go over all 200 10-K filings.
  - In case of encoding errors, use the errors=“ignore” option in the open() command.

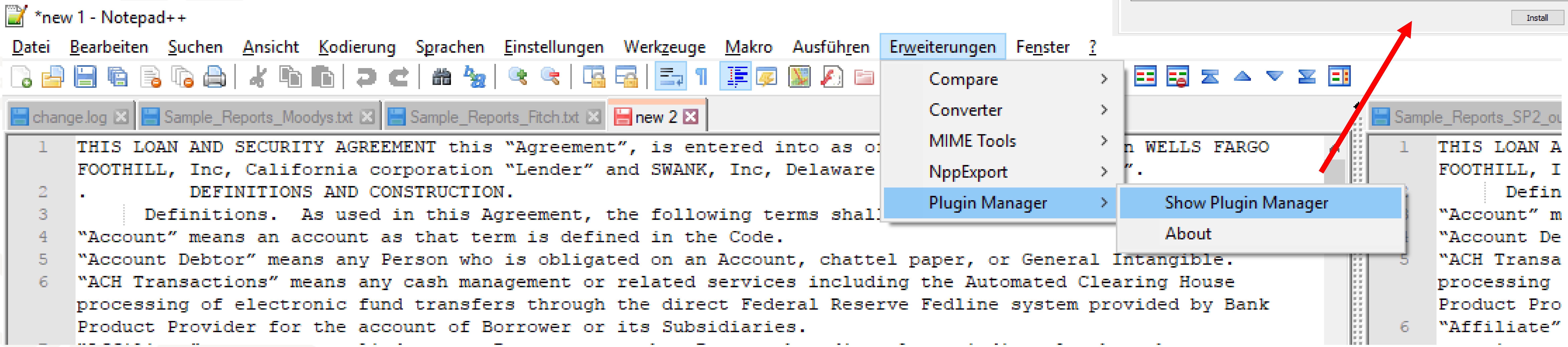
Helpful program for editing texts (1)

Text editors - Software recommendation

- When editing texts (e.g., removing disclaimers, tables, numbers) we may want to quickly compare the original and edited text to identify the changes.
- Notepad++ is a good choice.
  - Available for free at https://notepad-plus-plus.org/.
  - Very handy “Compare” plugin.



Plugin	Category	Available Version	Stability
3P - Progress Programmers Pal	Others	1.7	Good
ActiveX Plugin	Others	1.1.7.3	Good
AnalysePlugin	Others	1.10.32	Good
AndroidLogger	Others	1.2.6.2	Good
Automation Scripts	Others	1.0.4	Good
AutoSave	Others	1.5	Good
Autosave2	Others	1.0	Good
Bookmark Manager	Others	0.1.0.97	Good

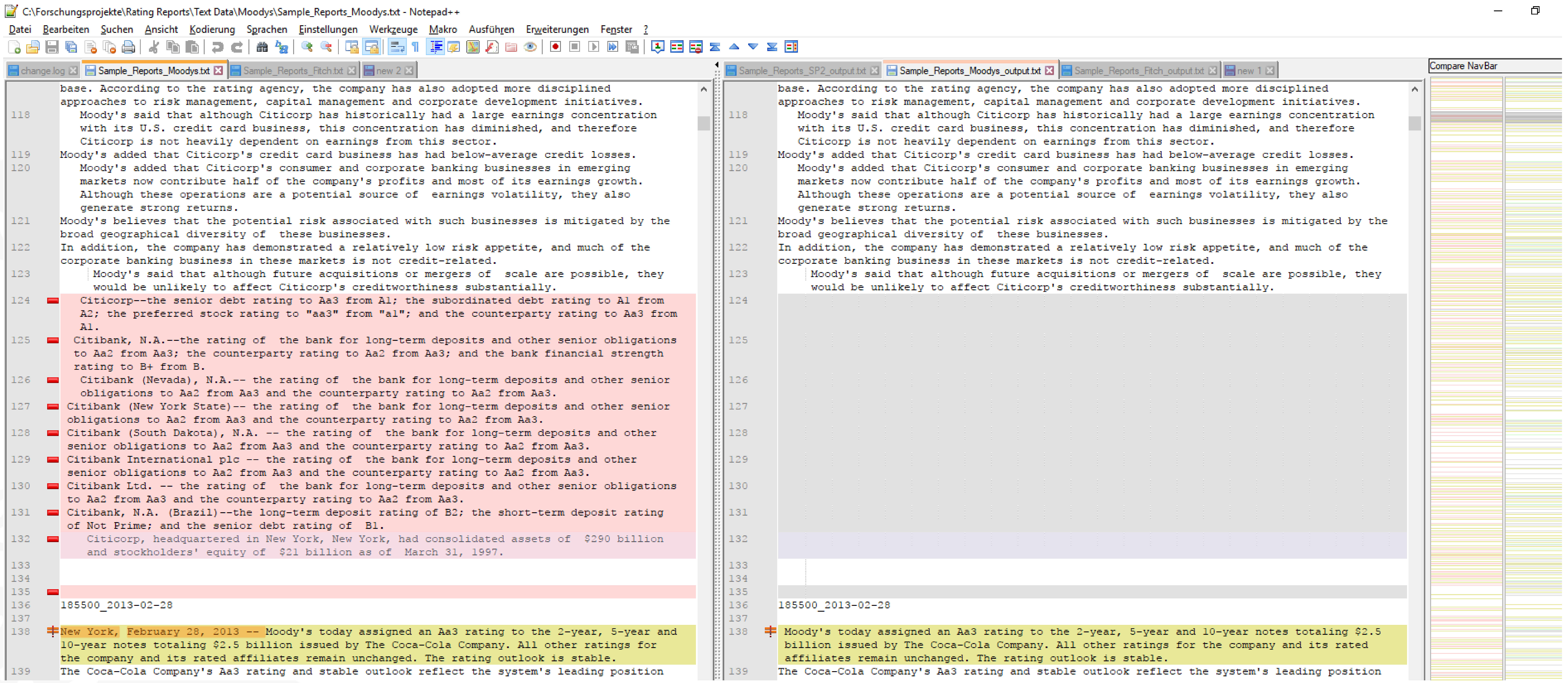


```
THIS LOAN AND SECURITY AGREEMENT this “Agreement”, is entered into as o        n WELLS FARGO
FOOTHILL, Inc, California corporation “Lender” and SWANK, Inc, Delaware        ”.
.          DEFINITIONS AND CONSTRUCTION.
    Definitions.  As used in this Agreement, the following terms shal
“Account” means an account as that term is defined in the Code.
“Account Debtor” means any Person who is obligated on an Account, chattel paper, or General Intangible.
“ACH Transactions” means any cash management or related services including the Automated Clearing House
processing of electronic fund transfers through the direct Federal Reserve Fedline system provided by Bank
Product Provider for the account of Borrower or its Subsidiaries.
```

```
THIS LOAN A
FOOTHILL, I
      Defin
“Account” m
“Account De
“ACH Transa
processing
Product Pro
“Affiliate”
```

Helpful program for editing texts (2)

Compare plugin in Notepad++



C:\Forschungsprojekte\Rating Reports\Text Data\Moodys\Sample_Reports_Moodys.txt - Notepad++

Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Werkzeuge Makro Ausführen Erweiterungen Fenster ?

change.log
Sample_Reports_Moodys.txt
Sample_Reports_Fitch.txt
new 2
Sample_Reports_SP2_output.txt
Sample_Reports_Moodys_output.txt
Sample_Reports_Fitch_output.txt
new 1
Compare NavBar

```
base. According to the rating agency, the company has also adopted more disciplined
approaches to risk management, capital management and corporate development initiatives.
    Moody's said that although Citicorp has historically had a large earnings concentration
    with its U.S. credit card business, this concentration has diminished, and therefore
    Citicorp is not heavily dependent on earnings from this sector.
Moody's added that Citicorp's credit card business has had below-average credit losses.
    Moody's added that Citicorp's consumer and corporate banking businesses in emerging
    markets now contribute half of the company's profits and most of its earnings growth.
    Although these operations are a potential source of  earnings volatility, they also
    generate strong returns.
Moody's believes that the potential risk associated with such businesses is mitigated by the
broad geographical diversity of  these businesses.
In addition, the company has demonstrated a relatively low risk appetite, and much of the
corporate banking business in these markets is not credit-related.
     Moody's said that although future acquisitions or mergers of  scale are possible, they
     would be unlikely to affect Citicorp's creditworthiness substantially.
    Citicorp--the senior debt rating to Aa3 from A1; the subordinated debt rating to A1 from
    A2; the preferred stock rating to "aa3" from "a1"; and the counterparty rating to Aa3 from
    A1.
   Citibank, N.A.--the rating of  the bank for long-term deposits and other senior obligations
   to Aa2 from Aa3; the counterparty rating to Aa2 from Aa3; and the bank financial strength
   rating to B+ from B.
    Citibank (Nevada), N.A.-- the rating of  the bank for long-term deposits and other senior
    obligations to Aa2 from Aa3 and the counterparty rating to Aa2 from Aa3.
 Citibank (New York State)-- the rating of  the bank for long-term deposits and other senior
 obligations to Aa2 from Aa3 and the counterparty rating to Aa2 from Aa3.
 Citibank (South Dakota), N.A. -- the rating of  the bank for long-term deposits and other
 senior obligations to Aa2 from Aa3 and the counterparty rating to Aa2 from Aa3.
 Citibank International plc -- the rating of  the bank for long-term deposits and other
 senior obligations to Aa2 from Aa3 and the counterparty rating to Aa2 from Aa3.
 Citibank Ltd. -- the rating of  the bank for long-term deposits and other senior obligations
 to Aa2 from Aa3 and the counterparty rating to Aa2 from Aa3.
 Citibank, N.A. (Brazil)--the long-term deposit rating of B2; the short-term deposit rating
 of Not Prime; and the senior debt rating of  B1.
     Citicorp, headquartered in New York, New York, had consolidated assets of  $290 billion
     and stockholders' equity of  $21 billion as of  March 31, 1997.



185500_2013-02-28

New York, February 28, 2013 -- Moody's today assigned an Aa3 rating to the 2-year, 5-year and
10-year notes totaling $2.5 billion issued by The Coca-Cola Company. All other ratings for
the company and its rated affiliates remain unchanged. The rating outlook is stable.
The Coca-Cola Company's Aa3 rating and stable outlook reflect the system's leading position
```

```
base. According to the rating agency, the company has also adopted more disciplined
approaches to risk management, capital management and corporate development initiatives.
    Moody's said that although Citicorp has historically had a large earnings concentration
    with its U.S. credit card business, this concentration has diminished, and therefore
    Citicorp is not heavily dependent on earnings from this sector.
Moody's added that Citicorp's credit card business has had below-average credit losses.
    Moody's added that Citicorp's consumer and corporate banking businesses in emerging
    markets now contribute half of the company's profits and most of its earnings growth.
    Although these operations are a potential source of  earnings volatility, they also
    generate strong returns.
Moody's believes that the potential risk associated with such businesses is mitigated by the
broad geographical diversity of  these businesses.
In addition, the company has demonstrated a relatively low risk appetite, and much of the
corporate banking business in these markets is not credit-related.
     Moody's said that although future acquisitions or mergers of  scale are possible, they
     would be unlikely to affect Citicorp's creditworthiness substantially.




185500_2013-02-28

 Moody's today assigned an Aa3 rating to the 2-year, 5-year and 10-year notes totaling $2.5
 billion issued by The Coca-Cola Company. All other ratings for the company and its rated
 affiliates remain unchanged. The rating outlook is stable.
The Coca-Cola Company's Aa3 rating and stable outlook reflect the system's leading position
```

Importance of proper document editing (1)

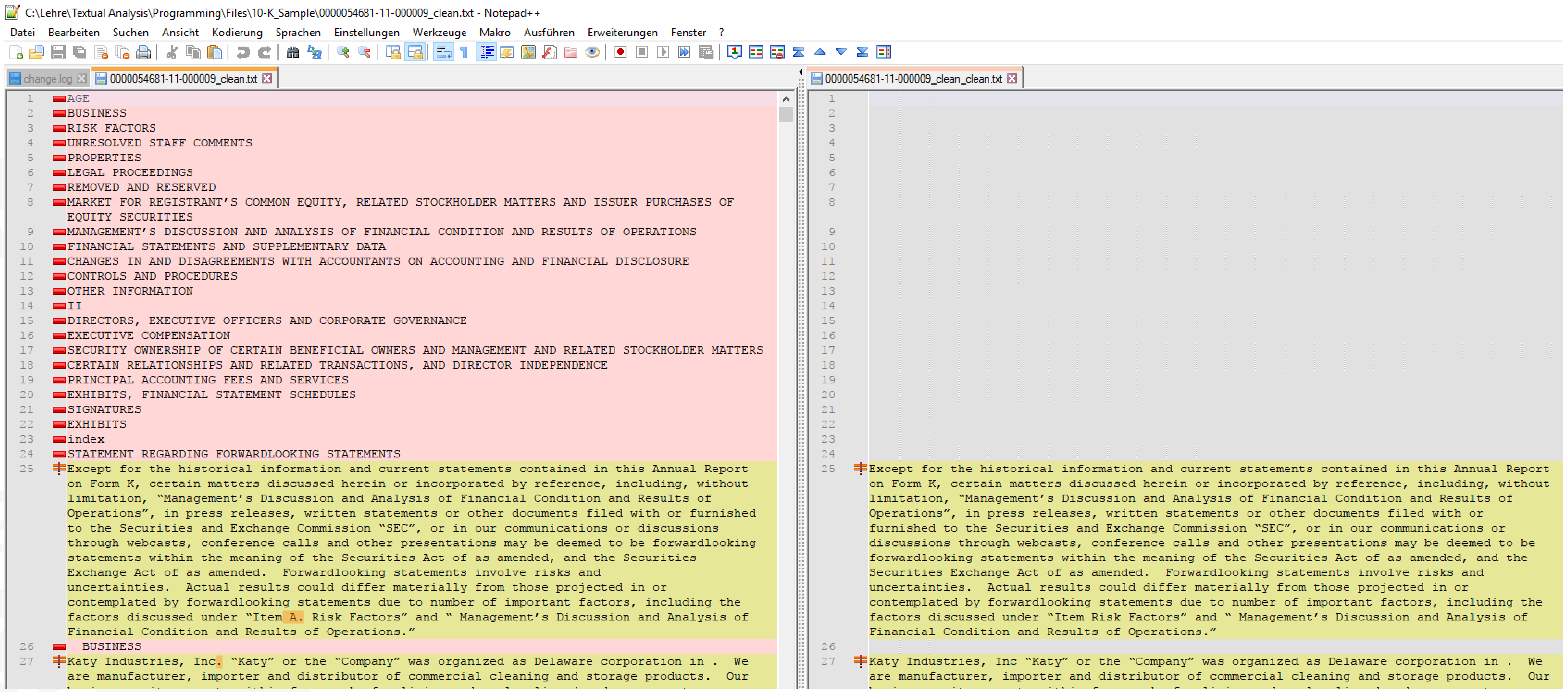
To get texts that can be used in the later problems (e.g., determining the words per sentence (WPS), measuring textual similarity), we have to do further editing steps.

- When computing WPS, we split the text into sentences.
  → split by ".", "?", and "!".
  - Delete dots in abbreviations: Inc., Corp., Mr., Ms., St. [WPS biased downwards]
  - Delete URLs: www.someadress.com [WPS biased downwards]
  - Delete file names: document.pdf or picture.jpg [WPS biased downwards]
  - Delete enumerations [WPS biased upwards]
    They can be identified by lines without a symbol indicating the end of sentence (".", "?", and "!")
  - Delete disclaimers and text parts that are mandated by the SEC [WPS biased upwards].
    They can be identified by sentences in UPPER CASE LETTERS.

→ There are more procedures you can (and should) implement in your research project.

Importance of proper document editing (2)

Illustration of “sloppy” and “thorough” editing


C:\Lehre\Textual Analysis\Programming\Files\10-K_Sample\0000054681-11-000009_clean.txt - Notepad++
Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Werkzeuge Makro Ausführen Erweiterungen Fenster ?
change.log
0000054681-11-000009_clean.txt
0000054681-11-000009_clean_clean.txt

```
AGE
BUSINESS
RISK FACTORS
UNRESOLVED STAFF COMMENTS
PROPERTIES
LEGAL PROCEEDINGS
REMOVED AND RESERVED
MARKET FOR REGISTRANT’S COMMON EQUITY, RELATED STOCKHOLDER MATTERS AND ISSUER PURCHASES OF
EQUITY SECURITIES
MANAGEMENT’S DISCUSSION AND ANALYSIS OF FINANCIAL CONDITION AND RESULTS OF OPERATIONS
FINANCIAL STATEMENTS AND SUPPLEMENTARY DATA
CHANGES IN AND DISAGREEMENTS WITH ACCOUNTANTS ON ACCOUNTING AND FINANCIAL DISCLOSURE
CONTROLS AND PROCEDURES
OTHER INFORMATION
II
DIRECTORS, EXECUTIVE OFFICERS AND CORPORATE GOVERNANCE
EXECUTIVE COMPENSATION
SECURITY OWNERSHIP OF CERTAIN BENEFICIAL OWNERS AND MANAGEMENT AND RELATED STOCKHOLDER MATTERS
CERTAIN RELATIONSHIPS AND RELATED TRANSACTIONS, AND DIRECTOR INDEPENDENCE
PRINCIPAL ACCOUNTING FEES AND SERVICES
EXHIBITS, FINANCIAL STATEMENT SCHEDULES
SIGNATURES
EXHIBITS
index
STATEMENT REGARDING FORWARDLOOKING STATEMENTS
Except for the historical information and current statements contained in this Annual Report
on Form K, certain matters discussed herein or incorporated by reference, including, without
limitation, “Management’s Discussion and Analysis of Financial Condition and Results of
Operations”, in press releases, written statements or other documents filed with or furnished
to the Securities and Exchange Commission “SEC”, or in our communications or discussions
through webcasts, conference calls and other presentations may be deemed to be forwardlooking
statements within the meaning of the Securities Act of as amended, and the Securities
Exchange Act of as amended.  Forwardlooking statements involve risks and
uncertainties.  Actual results could differ materially from those projected in or
contemplated by forwardlooking statements due to number of important factors, including the
factors discussed under “Item A. Risk Factors” and “ Management’s Discussion and Analysis of
Financial Condition and Results of Operations.”
   BUSINESS
Katy Industries, Inc. “Katy” or the “Company” was organized as Delaware corporation in .  We
are manufacturer, importer and distributor of commercial cleaning and storage products.  Our
```

```
Except for the historical information and current statements contained in this Annual Report
on Form K, certain matters discussed herein or incorporated by reference, including, without
limitation, “Management’s Discussion and Analysis of Financial Condition and Results of
Operations”, in press releases, written statements or other documents filed with or
furnished to the Securities and Exchange Commission “SEC”, or in our communications or
discussions through webcasts, conference calls and other presentations may be deemed to be
forwardlooking statements within the meaning of the Securities Act of as amended, and the
Securities Exchange Act of as amended.  Forwardlooking statements involve risks and
uncertainties.  Actual results could differ materially from those projected in or
contemplated by forwardlooking statements due to number of important factors, including the
factors discussed under “Item Risk Factors” and “ Management’s Discussion and Analysis of
Financial Condition and Results of Operations.”

Katy Industries, Inc “Katy” or the “Company” was organized as Delaware corporation in .  We
are manufacturer, importer and distributor of commercial cleaning and storage products.  Our
```

Agenda

- **Part 1:** Getting familiar with Python
- **Part 2:** Preparing documents for textual analysis
- $\boxed{\textbf{Part 3:} \text{ Textual analysis (tone)}}$
- **Part 4:** Readability, language complexity, and textual similarity
- **Part 5:** Machine learning

Overview of Problems in Part 3: Textual Analysis – Computing Tone

Programming problems in Part 3

- Problem 7: Determine the negativity of the edited 10-K filings from Problem 6. Negativity is measured by the percentage of negative words.
- Problem 8: Determine the positivity of the edited 10-K filings from Problem 6. Positivity is measured by percentage of positive words <u>controlling for negations</u>.

- As we deal with business/finance texts, we will use the dictionaries of Loughran and McDonald (2011).

- To get comparable results, please use the edited 10-K filings from the zip file “10-K_Sample_clean.zip”. These files have been created by the solution to Problem 6.

Problem 7 – Determine the negativity of 10-Ks

Problem 7: determine the negativity of annual reports

- Compute the percentage of negative words according to the Loughran and McDonald (2011) dictionary of negative words for the 10-K filings from Problem 6.
- Your program should create a csv file that contains the CIK (column 1), the filename (column 2), the total number of words (column 3), the number of negative words (column 4) and the percentage of negative words (column 5).
- The file "LMD_Neg.txt" contains the list of negative words according to the Loughran and McDonald (2011) dictionary. Each line contains exactly one word.

→ *See the next slide for hints.*

The programming template for this problem is "Problem_7_Tone_Analysis_form.py".

Problem 7: determine the negativity of annual reports - continued

- Hints:
  - You need two loops. The first (outer) loop iterates the list of the 200 10-Ks, while the second (inner) loop iterates the 2,239 negative words in the LMD dictionary.
  - You can split the text into single words by using the "*.split()*" or "*re.split()*" command. In regular expressions, "*\W*" refers to a word boundary.
  - The "*.count()*" command is helpful as well.
    Compare the result of the following two count commands:
    text='poor poorer poorest'
    1. *text.count('poor')*
    2. *list_of_words=re.split('\W',text)*
       list_of_words.count('poor')
  - There are several ways of how to program a word count.

The programming template for this problem is "Problem_7_Tone_Analysis_form.py".

Problem 7 – Validation of results (1)

Validation of results

- Can we validate our programming results?
  - Loughran and McDonald provide a file with information on all 10-K filings from 1994 to 2014.
  - The file is named "LoughranMcDonald_10X_2014.xlsx" and can be downloaded from http://www3.nd.edu/~mcdonald/Word_Lists.html. → Compare our results to LMD

- Correlations
  - number of words: 0.7845
  - number of negative words: 0.8185
  - negativity: 0.9183
  - Not bad. Where does the difference come from?
    → Most likely from editing the documents.

Length according to our computations:
Mean: 29,134 words; Min: 2684; Max: 93,369.
Length according to LMD's file:
Mean: 30,950 words; Min: 791; Max: 188,781.
Negativity according to our computations:
Mean: 1.64%; Min: 0.45%; Max: 3.30%.
Negativity according to LMD's file:
Mean: 1.69%; Min: 0.55%; Max: 3.52%.

Problem 7 – Validation of results (2)

Validation of results - continued

- There are also commercial textual analysis programs. For example, Linguistic Inquiry and Word Count (LIWC).
  - http://www.liwc.net/
  - The program performs a dictionary-based word count analysis.
    However, it is <u>less powerful</u> than self-made Python program.
    - does not account for negations when measuring positive tone.
    - cannot search for terms that consist of multiple words, e.g., “bank run”.
  - However, we can use it as a benchmark for our self-programmed Python program.

Problem 8 – Determine the positivity of 10-Ks

Problem 8: determine the positivity of annual reports

- Determine the percentage of non-negated positive words according to the Loughran and McDonald (2011) dictionary for 10-K filings from Problem 6.
- Use Loughran and McDonald's (2011) procedure to control for negations.
  "We account for simple negation only for Fin-Pos words. Simple negation is taken to be observations of one of six words (no, not, none, neither, never, nobody) occurring within three words preceding a positive word." Loughran and McDonald (2011), p. 44.
- Create a csv file that contains the CIK (column 1), the filename (column 2), the total number of words (column 3), the number of positive words (column 4), and the number of non-negated positive words (column 5).
- The file "LMD_Pos.txt" contains the list of positive words according to the Loughran and McDonald (2011) dictionary. Each line contains exactly one word.

→ *See the next slide for hints.*

Problem 8 – Determine the positivity of 10-Ks

Problem 8: determine the positivity of annual reports - continued

- Hints:
  - The program structure is very similar to the one of Problem 7 (two loops).
  - However, to determine the non-negated number of negative words, you need to take into account the three words before a positive word. Identify the position of the positive words (e.g., the 241st word of the document) and check whether the previous three words are negations (e.g., the words at positions 238 to 240).

The programming template for this problem is "Problem_8_Tone_Analysis_Positive_Words_form.py".

Agenda

- **Part 1:** Getting familiar with Python
- **Part 2:** Preparing documents for textual analysis
- **Part 3:** Textual analysis (tone)
- **Part 4:** Readability, language complexity, and textual similarity
- **Part 5:** Machine learning

Overview of Part 4: Readability, language complexity, and textual similarity

Text characteristics other than positivity and negativity

- Other word lists
- Document complexity
- Textual similarity

Programming problems in Part 4

- Problem 9: Calculate the average words per sentence (WPS).
- Problem 10: Compute the fraction of complex words.
- Problem 11: Determine the gross and net file size.
- Problem 12: Identify the most frequent words in a text.
- Problem 13: Word Stemming.
- Problem 14: Jaccard Similarity.

Readability – Measured by Words Per Sentence

Problem 9: determine average number of words per sentence

- A common and appropriate measure of complexity is the average number of words per sentence (WPS). Determine the average WPS for 10-K filings from Problem 6.
- Create a csv file that contains the CIK (column 1), the filename (column 2), the total number of words (column 3), the number of sentences (column 4), and the average WPS (column 5).

- Hints:
  - You can use the same procedure as in Problem 7 and 8 to determine the number of words.
  - To determine the number of sentences, you can split the document by symbols indicating the end of a sentence, e.g., full stop, exclamation mark, and question mark.

The programming template for this problem is "Problem_9_Words_per_Sentence_form.py".

How to identify sentences correctly? (1)

Sentence identification

- Are there Python packages that do the work for us? Maybe they also do a better job?!
  - Most popular package is NLTK (Natural Language Tool Kit), which we will discuss later in the course.
  - NLTK includes a sentence tokenizer.
  - Syntax
    - *from nltk.tokenize import sent_tokenize*
    - *list_of_sentences=sent_tokenize(text)*
  - Let's assess NLTK's performance by looking at some examples.

How to identify sentences correctly? (2)

Sentence identification - continued

- Example 1:
  - *"The S&P 500 rose 43.44 points to 4,159.12. The Dow Jones industrial average added 188.11 points, or 0.6 percent, to 34,084.15. The tech-heavy Nasdaq fared better than the rest of the market, climbing 236 points, or 1.8 percent, to 13,535.74."*
  - Result:
    1. *The S&P 500 rose 43.44 points to 4,159.12.* ✓
    2. *The Dow Jones industrial average added 188.11 points, or 0.6 percent, to 34,084.15.* ✓
    3. *The tech-heavy Nasdaq fared better than the rest of the market, climbing 236 points, or 1.8 percent, to 13,535.74.* ✓

    → Good job.
  - Repeat the analysis with the text being lower case and there is NO split → NLTK considers the text to be a single sentence. Poor job!

You find this example in "NLTK_Sentence_Tokenizer.py".

How to identify sentences correctly? (3)

You find this example in “NLTK_Sentence_Tokenizer.py”.

Sentence identification - continued

- Example 2:
  - *“On Sept. 16, 2020, the U.S. president appointed John D. Smith as head of the F. B. I. While Jane C. Taylor became the president of the S. E. C. On Jan. 5, 2020, J. C. Penny filed for bankruptcy. Michael T. Brown - reporting from Washington D.C.”*
  - Result:
    1. *On Sept. 16, 2020, the U.S. president appointed John D. Smith as head of the F. B. I.* ✓
    2. *While Jane C. Taylor became the president of the S. E. C. On Jan. 5, 2020, J. C. Penny filed for bankruptcy.* ✗
    3. *Michael T. Brown - reporting from Washington D.C.* ✓

    → Okay performance.

    → Performance depends on specific abbreviations (FBI vs. SEC).

  - Repeat the analysis with the text being lower case and there is just ONE split (sentences 1 and 2 from above are not identified) → NLTK considers the text to be a two sentence. Poor job.

How to identify sentences correctly? (4)

Sentence identification - continued

- When you apply the NLTK tokenizer to the 10-Ks from Problem 9, the average WPS is 41.85 vs. 31.67 (our approach).
- Key problem: missing splits when there are no whitespaces!
  - *"This Annual Report contains forwardlooking statements as that term is defined in the federal securities laws.The Company wishes to insure that […] may not occur.Generally, these statements relate to business plans or strategies"*
  - Where does this come from?
    - *"in the federal securities laws.&#160;&#160;The Company wishes to insure"*
    - *"may not occur.&#160;&#160;Generally, these statements"*

      see https://www.sec.gov/Archives/edgar/data/719494/000114420411018856/0001144204-11-018856.txt

  → It is the initial filings!
  - "&#160;" is html code for "non-breaking space" → html code is deleted by BeautifulSoup → NLTK tokenizer does not identify end of sentence, but our approach does!

How to identify sentences correctly? (5)

Sentence identification - continued

- Conclusion: extensive text editing is required!
  - Keeping upper- and lower-case letters may be helpful.
  - Remove numbers (incl. dots, commas).
  - Delete abbreviations (e.g., middle names, U.S.A., Mr., Dr., Jan., Feb.).
  - The documents we work with have been edited to some extent (see my solution to problem 6). In a research project, you should do more.
  - Always manually check your results!

- Bill McDonald's view: *"Textual analysis is not like inverting a matrix, where a routine developed in another context will still provide exactly the same result. For example, using **NLTK** to produce an average-words-per-sentence measure for 10-Ks is very simple but produces **disastrous results**."*
  https://sraf.nd.edu/textual-analysis/for-beginners/

Problem 10 – Determine the fraction of complex words of 10-Ks

Problem 10: determine the fraction of complex words for 10-K filings from Problem 6

- A complex words is a word with more than two syllables.
- Loughran and McDonald provide the number of syllables for all words that appear in at least one SEC filing in their master dictionary which is available at http://www3.nd.edu/~mcdonald/Word_Lists.html. The file ("LoughranMcDonald_MasterDictionary_2014.xlsx") is available in our course folder.
- Create a csv file that contains the CIK (column 1), the filename (column 2), the total number of words (column 3), and the number of complex words (column 4).
- The file "Complex_Words.txt" (also available in our course folder) contains the list of words with more than two syllables according to the master dictionary. Each line contains exactly one word.

- Hints:
  - The programming is very similar to Problem 7 → use the solution to Problem 7 as starting point.
  - Since there are many complex words (47,155), the execution of the program will take more time.

Problem 11 – Determine the file size of 10-Ks

Problem 11: determine the file size for 10-K filings from Problem 6

- Create a csv file that contains the CIK (column 1), the filename (column 2), the file size of the complete submission file (column 3), and the file size of the main text ("_cleaned.txt") (column 4).

- Hints:
  - The command os.path.getsize() is helpful here.
  - Before you can use the previous command, you must import the "os" module.

The programming template for this problem is "Problem_11_determine_file_size_form.py".

Further data editing operations

- **Motivation**:
  - It is important to know which words drive results (see Loughran and McDonald (2011)).
    → how to get a list of the most frequent words?
    → Problem 12: working with counter objects.
  - For textual similarity, additional editing steps may improve accuracy.
    – Some words (e.g., “the”, “a”, “and”) are omnipresent → might be better to drop them.
    – Using inflections may not be helpful:
    “According to its report, X lost $10 million.” vs. “X reports a loss of $10 million.”

After Problem 12, we will look at Python’s Natural Language Toolkit (NLTK).

NLTK helps to:

- drop stop words.
- reduce words to their stem.

Problem 12

The programming template for this problem is "Problem_12_Most_Frequent_Words_form".

Problem 12: identify the most frequent words in a text

In our course folder, you find the file "10-Ks_Textual_Similarity.zip". The file contains the 10-Ks (full submission files) of Microsoft, Coca Cola, and 3M.

The file names are "cik_link_adj.txt" where *link_adj* is the part of the link after the last slash /.

All files are listed in the file "list_10-K_filings_textual_similarity.csv".

Your task:

- Create a list of the 100 most frequently used words in this text corpus.
  The output should be a table (e.g., csv file) with the words and their absolute frequencies.
  Are the top 100 words plausible?
- Optional extra task: create a list of the top 100 bigrams and their frequencies.
- To save time, you do not need to program the editing of the 10-Ks. The second zip file "*10-Ks_Textual_Similarity_edited.zip*" contains the edited files. The editing is very similar to the one in Problem 6.

Problem 12 – Hints

Problem 12 - continued

The programming template for this problem is “Problem_12_Most_Frequent_Words_form.py”.

Container datatypes

- Python has several container datatypes.
- You already know lists, which are one of the container datatypes.
- Here, you need counters, which are a special version of dictionaries (another container datatype)
- For more information on the different container datatypes see https://docs.python.org/3/library/collections.html#collections.Counter

- Before trying to solve Problem 12, go over my **introduction on datatypes**.
    - The file (**Introduction_Container_Datatypes.py**) is available in our course folder.
    - The first part is just a recap of lists.
    - The second part covers counters.

Introduction to NLTK (1)

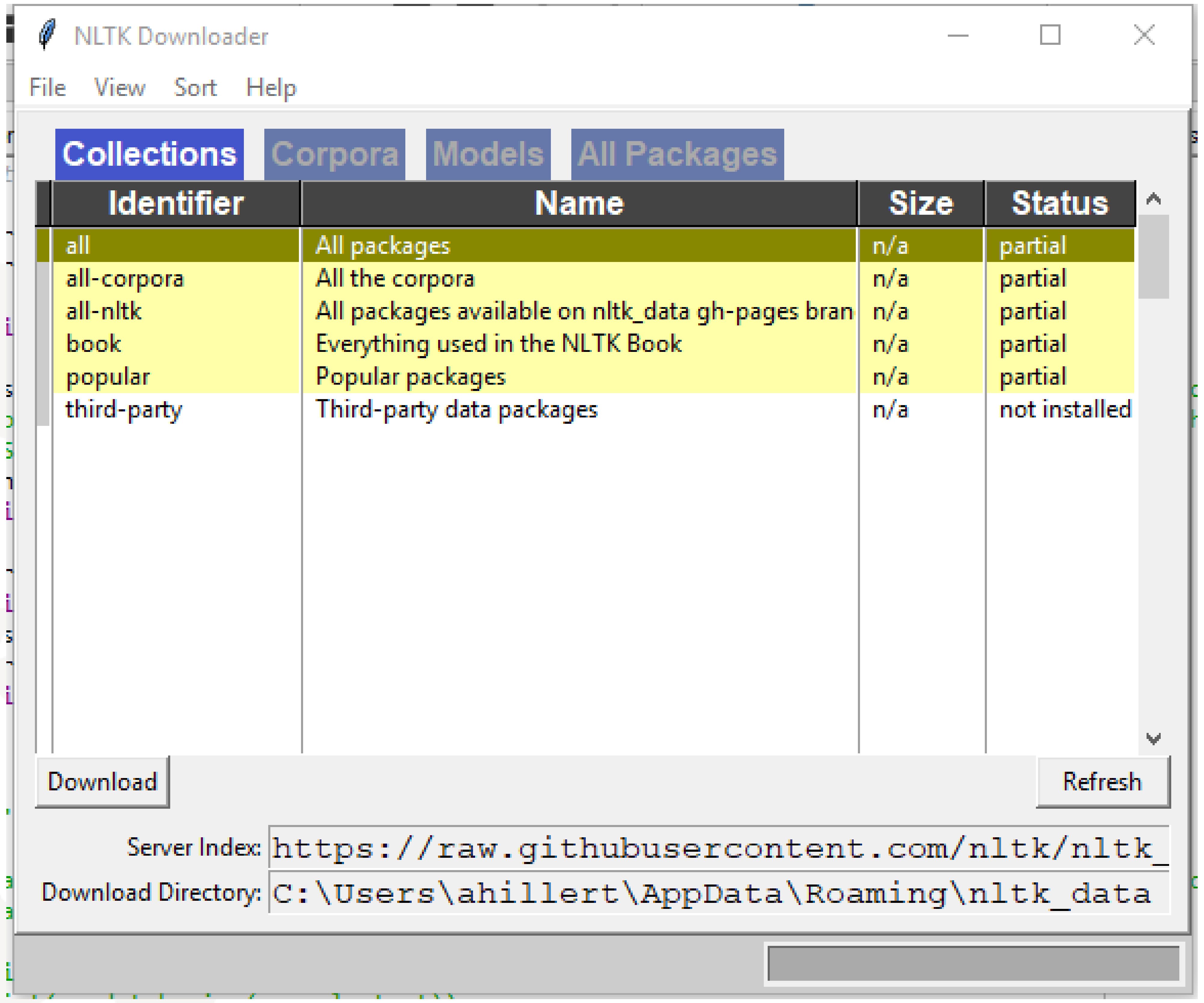
Python’s Natural Language ToolKit

- NLTK can split texts in sentences and/or words (“tokenization”), delete commonly used words from text (“stop words”), and much more.
- We will discuss tokenization, stop words, and stemming on the subsequent slides.
  → There is also a short introduction (“NLKT_introduction.py”) available in our course folder.

- Before you can use these tools, you need to download the NLTK components.
  - Alternative 1:
    import nltk
    nltk.download(component_name) → e.g., *nltk.download('punkt')*
  - Alternative 2:
    import nltk
    nltk.download() → a new window opens

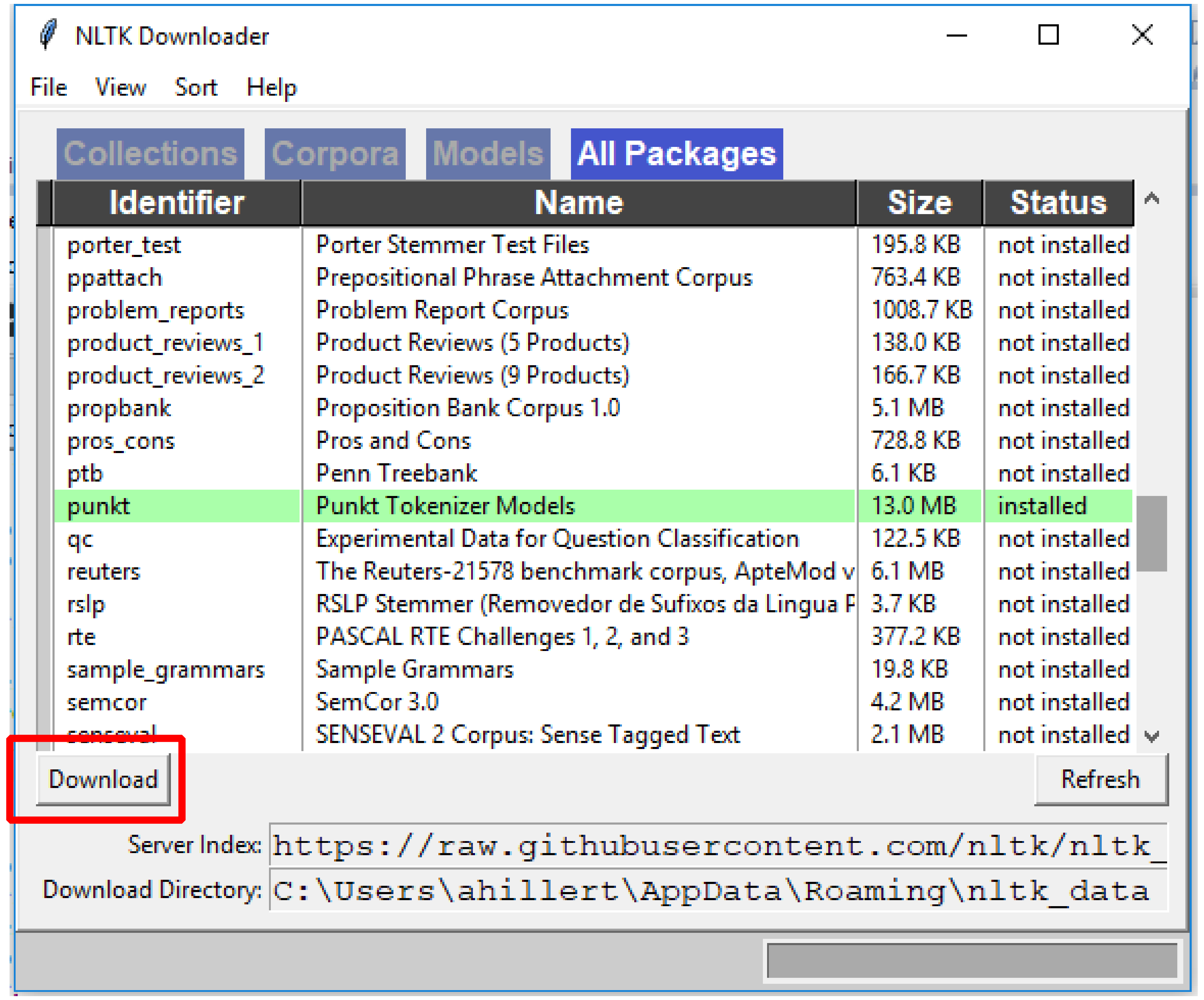
Introduction to NLTK (2)

Downloading NLTK components – continued



Identifier	Name	Size	Status
all	All packages	n/a	partial
all-corpora	All the corpora	n/a	partial
all-nltk	All packages available on nltk_data gh-pages bran	n/a	partial
book	Everything used in the NLTK Book	n/a	partial
popular	Popular packages	n/a	partial
third-party	Third-party data packages	n/a	not installed

Server Index: https://raw.githubusercontent.com/nltk/nltk_
Download Directory: C:\Users\ahillert\AppData\Roaming\nltk_data



Identifier	Name	Size	Status
porter_test	Porter Stemmer Test Files	195.8 KB	not installed
ppattach	Prepositional Phrase Attachment Corpus	763.4 KB	not installed
problem_reports	Problem Report Corpus	1008.7 KB	not installed
product_reviews_1	Product Reviews (5 Products)	138.0 KB	not installed
product_reviews_2	Product Reviews (9 Products)	166.7 KB	not installed
propbank	Proposition Bank Corpus 1.0	5.1 MB	not installed
pros_cons	Pros and Cons	728.8 KB	not installed
ptb	Penn Treebank	6.1 KB	not installed
punkt	Punkt Tokenizer Models	13.0 MB	installed
qc	Experimental Data for Question Classification	122.5 KB	not installed
reuters	The Reuters-21578 benchmark corpus, ApteMod v	6.1 MB	not installed
rslp	RSLP Stemmer (Removedor de Sufixos da Lingua P	3.7 KB	not installed
rte	PASCAL RTE Challenges 1, 2, and 3	377.2 KB	not installed
sample_grammars	Sample Grammars	19.8 KB	not installed
semcor	SemCor 3.0	4.2 MB	not installed
senseval	SENSEVAL 2 Corpus: Sense Tagged Text	2.1 MB	not installed

Server Index: https://raw.githubusercontent.com/nltk/nltk_
Download Directory: C:\Users\ahillert\AppData\Roaming\nltk_data

- Select the components you need and click on “download”.

Tokenization with NLTK

What does tokenize mean?

- Tokenize = split a text into groups.
- Common groups
  - Sentences (see Problem 9)
    from nltk.tokenize import sent_tokenize
    test_text="Sentence one. Sentence two? This is sentence three!"
    sentence_list=sent_tokenize(test_text)
  - Words
    from nltk.tokenize import word_tokenize
    test_text="This is a sentence with a lot of different words and 2 numbers."
    sentence_list=sent_tokenize(test_text)
- The nltk.tokenize is based on a combination of several regular expressions.
  → similar to what we have developed. It is more advanced, but not necessarily better (see Problem 9)

NLTK word tokenizer

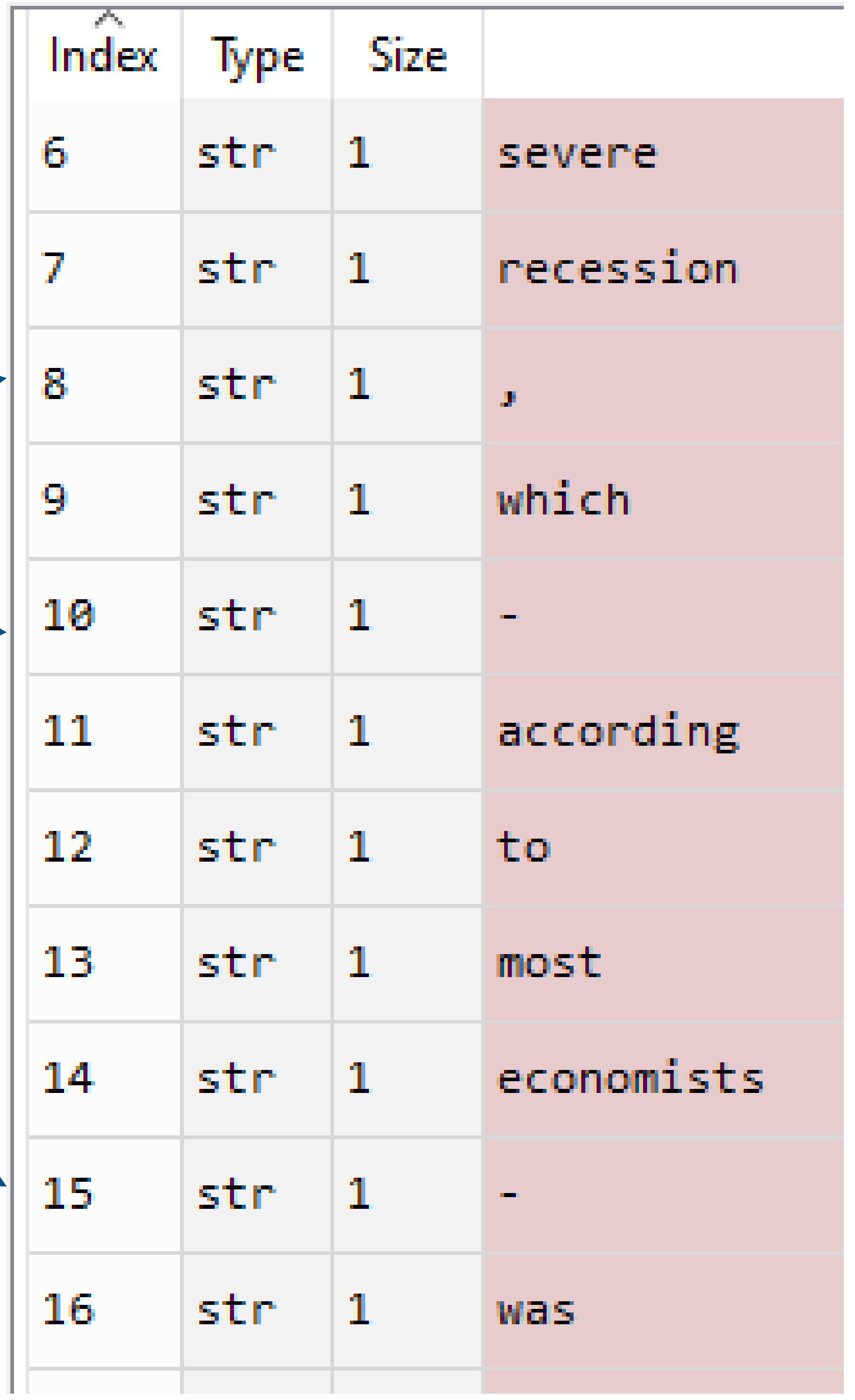
Performance of NLTK word tokenizer

- <u>Example</u>
  - *“The covid pandemic resulted in a severe recession, which - according to most economists - was the worst recession since the Great Depression.“*
  - Word length according to NLTK: 25
  - Actual length (and length according to our approach using re.split(“\W{1,}”)): 21
- What is the problem?

  →NLTK includes symbols: “,”, “-” (2 times), and “.”

  →Word count overstated!

- **<u>Conclusion:</u>**
  - Use our re.split(“\W{1,}”) approach.
  - NLTK word tokenizer only helpful if you want to keep symbols.



Index	Type	Size	
6	str	1	severe
7	str	1	recession
8	str	1	,
9	str	1	which
10	str	1	-
11	str	1	according
12	str	1	to
13	str	1	most
14	str	1	economists
15	str	1	-
16	str	1	was

Stop words with NLTK

What are stop words?

- Words with
  1. a very high frequency and that are usually uninformative like “and”, “the”, “or”, etc.
  2. ambiguous meaning, e.g., words that often express irony and/or sarcasm.
     → Hard to identify. For company filings irony is not an issue, but analysts’ questions in a conference may be ironic or sarcastic.
- Remember: if stops words are equally distributed across documents, they will not bias sentiment scores.
- NLTK has a list of stop words.
  from nltk.corpus import stopwords
  stop_words = set(stopwords.words("english"))
  print(stop_words)

- See the NLTK introduction for how to remove stop words from texts.

Stemming with NLTK

What does stemming mean?

- Reducing inflected words to their word stem/root.
- The most common stemming algorithm for English is the Porter algorithm.
- NLTK syntax

```
from nltk.stem import PorterStemmer
example_words=["play", "player", "players", "played", "playing"]
for word in example_words:
    print(PorterStemmer().stem(word))
```

- It is not clear whether stemming is the best approach. For instance, Loughran and McDonald (2011) argue against stemming.

Further information on NLTK

NLTK

- Python's NLTK can do much more than the three tasks from the previous slides.
- NLTK does not only work for English but also for other languages. However, the English toolkit is the most powerful.
- To get a detailed introduction, I recommend the learning videos by "sentdex". It is a series of 21 videos covering different topics/NLTK modules. The link is: https://www.youtube.com/watch?v=FLZvOKSCkxY&list=PLQVvvaa0QuDf2JswnfiGkliBInZnIC4HL

- Further references:
  - https://pythonprogramming.net/stemming-nltk-tutorial/
  - https://pythonspot.com/en/nltk-stemming/
  - http://www.nltk.org/

Problem 13

Problem 13: Word Stemming

- Take the edited files from the zip file “10-Ks_Textual_Similarity_edited.zip” and transform the text to word stems.
- Save the stemmed filings as new files, e.g., “…_stemmed.txt”
- Slide 71 provides information on how to use NLTK’s word stemming tool

The programming template for this problem is “Problem_13_Stemming_form.py”.

Problem 14 (1)

Problem 14: Jaccard Similarity

Compute the Jaccard similarity of a firm's 10-K to the firm's 10-K from the previous year.

Use different versions of the 10-K filings:

1. the edited files from the zip file "10-Ks_Textual_Similarity_edited.zip".
2. the files from 1. but delete stop words.
   a. define your own list of stop words based on the 100 most frequently used words. Select only the words that you consider uninformative.
   b. use the stop word list from Python's NLTK package (see slide 70).
3. the files from Problem 13, i.e., the stemmed version ("…_stemmed.txt").
4. the files from 3. but remove stop words.

→Are the results similar? Does the data editing matter?

The programming template for this problem is "Problem_14_Jaccard_Similarity_form.py".

See the next slides for additional information.

Problem 14 (2)

Problem 14 – continued

- Recap of the Jaccard similarity measure: $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$
- Example:
  - A: “This is the first text of the Jaccard example.”
  - B: “This sentence represents the second Jaccard example.”

  $\rightarrow |A \cap B| = 4$ and $|A \cup B| = 11$ $\rightarrow$ Jaccard similarity = 4 / 11=0.3636

- Stop words and stemming in Tetlock (2011)
  - *“Before identifying unique words and bigrams, I exclude a standard list of 119 extremely common words such as “into,” “so,” and “that”; 42 common numbers (0 through 9 and 1978 through 2009); and 27 terms that are ubiquitous in financial news stories, such as “Dow Jones,” “New York,” and “newswire.””* Footnote 2
  - *“I also use a standard word-stemming algorithm to equate all similar forms of a word—e.g., “changing” and “changed” are both derivatives of “change.””* Footnote 2

Summary statistics and correlations of Jaccard similarity

Summary stats

Specification	mean	std dev	min	p50	max
default	0.806	0.102	0.515	0.820	1.000
own stop words	0.803	0.103	0.509	0.818	1.000
NLTK stop words	0.801	0.105	0.505	0.816	1.000
stemmed	0.820	0.088	0.571	0.835	1.000
stemmed + own stop words	0.817	0.090	0.563	0.833	1.000
stemmed + NLTK stop words	0.814	0.091	0.560	0.830	1.000

1. Removing stops words → decline in similarity.
2. Stemming words → increase in similarity.

Correlations

Specification	default	own stop words	NLTK stop words	stemmed	stemmed + own stop words	stemmed + NLTK stop words
default	1					
own stop words	1	1				
NLTK stop words	0.9999	1	1			
stemmed	0.9935	0.9932	0.9929	1		
stemmed + own stop words	0.9939	0.9937	0.9933	1	1	
stemmed + NLTK stop words	0.9941	0.9938	0.9935	0.9999	0.9999	1

→ Removing stop words does hardly affect Jaccard similarity. Stemming has a small effect.

Agenda

- **Part 1:** Getting familiar with Python
- **Part 2:** Preparing documents for textual analysis
- **Part 3:** Textual analysis (tone)
- **Part 4:** Readability, language complexity, and textual similarity
- $\boxed{\textbf{Part 5:}\text{ Machine learning}}$

Overview of Part 5: Machine learning

Programming problems in Part 5: Machine learning

- Problem 15: Naïve Bayes using NLTK movie review corpus.
- Problem 16: Introduction to linear models using SciKitLearn package.
- Problem 17: Application of Ridge and LASSO to text data.

Naïve Bayes – Problem 15

Problem 15 - Naïve Bayes with NLTK

- In our course folder, you find the script "NLTK_Sentiment_Analysis.py". This script uses the "Movie Review" corpus from NLTK. The corpus contains 2,000 movie reviews. The first 1,000 reviews are negative and the second 1,000 are positive.
- Tasks:

1. Go over the script and see what the commands do.
2. The script uses 1,900 reviews as the training set and 100 as the testing set. What changes if you use 1,000 reviews as training and the other 1,000 as testing set?
3. The script uses only the 3,000 most frequently occurring words for the classification. What happens if you use 5,000 words?
4. What happens if you use the 3,000 most frequent words but exclude the 200 most frequent words?

Introduction to Linear Models – Problem 16

Problem 16 – Introduction to linear models with SciKitLearn

In our course folder (introductions → SciKitLearn), you find the script "Introduction_SciKitLearn.py". This script comprises 10 parts that guide you through different linear models.

The introduction starts with a standard OLS regression using a toy data set and, in the second step, a larger data set. It also shows you how to standardize variables. Next, it introduces you to the syntax of Ridge regressions and explains how to perform cross-validation. Last, it repeats the parts on Ridge regressions for LASSO regressions.

The accompanying data set is "regression_data_scikit.csv". You find it in the same folder.

<u>Task</u>: go over this introduction and learn how to use Ridge and LASSO!

Application of Ridge and LASSO to text data – Problem 17

Problem 17 – Application of Ridge and LASSO to text data

In this problem, we will apply Ridge and LASSO regressions to text data. As text corpus, we will use all form 10-K filings from 2007 and 2008 that can be matched with CRSP (stock market data). Using these filings, we will analyze the relation between the filing CARs (abnormal return from day t [filing day] to day t+3 as in Loughran and McDonald (2011)) and the frequencies of the words in the 10-Ks. More specifically, we will regress the CARs on the (relative) frequency of all words that are found in at least one form 10-K filing. Thus, we have a high-dimensionality problem (n is about 8,500 filings, k is about 40,000 unique words).

Tasks:

- Train and tune a Ridge/LASSO model on the CARs and texts of the filings from 2007.
- Test the model out-of-sample for 2008. What is the R2? What is the MSE?
- Compare the coefficients between Ridge and LASSO.

The programming template for this problem is "Problem_17_Ridge_LASSO_text_data_form.py".