R and Python Integration for Natural Language Processing (NLP)

Lifeng Ren

2023-11-27

Table of contents

Before we start learning Large Language Models (LLM), let's use this lecture to get our feet wet for Natural Language Processing (NLP) in both `R` and `Python`.

1 Lecture Agenda

In this lecture, we will cover the following topics:

- `R` in `VS Code`
  - Install `R` in `VS Code`
  - Get familiar with `R` in `VS Code`
- Natural Language Processing (NLP) in `R` and `Python`
- Next Lecture: Large Language Models (LLM) with applications in Economics

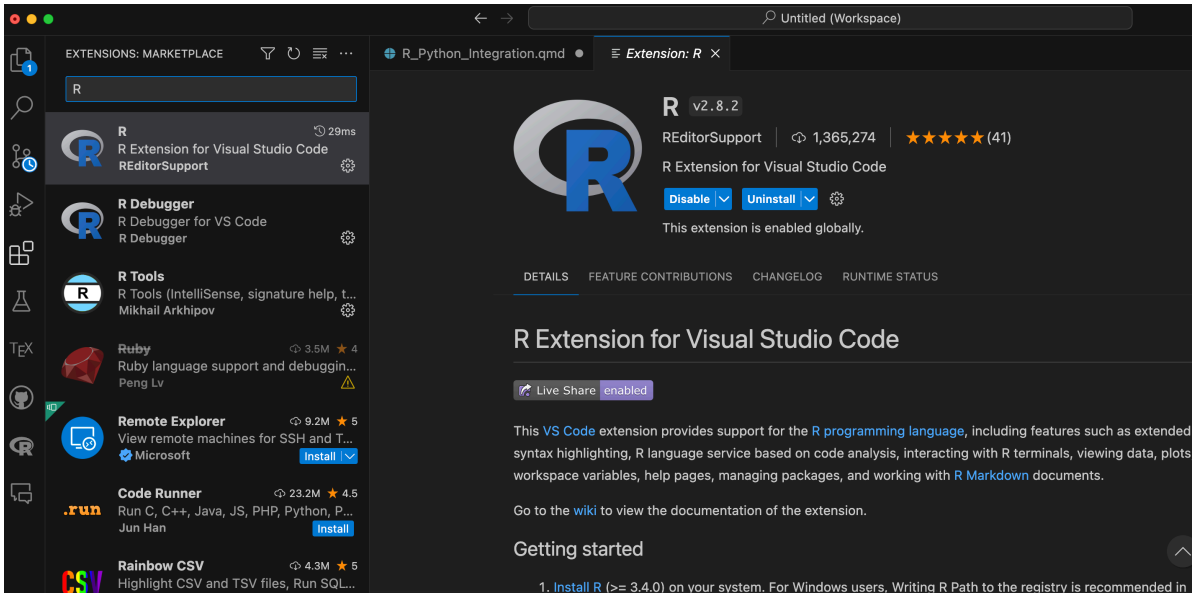
2 R in VS Code

2.1 Install R in `VS Code`

- I assume you all already installed `R` in your computer. If not, please go to this website and download the latest version of `R` and install it in your computer.

2.1.1 Install `R` extension in `VS Code`

Go to the VS Code extension marketplace and search for `R` and install the extension.



2.1.2 Install `radian`

- In your 8222 python environment, install pacakge `radian` using the following command:

```
mamba activate 8222env3
mamba install -c conda-forge radian
```

Or, you can use `pip` to install `radian`:

```
pip install radian
```

- To Run `radian`:

  ```
  radian
  ```

- To exit `radian`:

  ```
  q()
  ```

2.1.3 Launch `radian` and install `languageserver`

In the `radian` console, run the following command to install `languageserver`, say `yes`, `1`, and `yes` to the following questions.

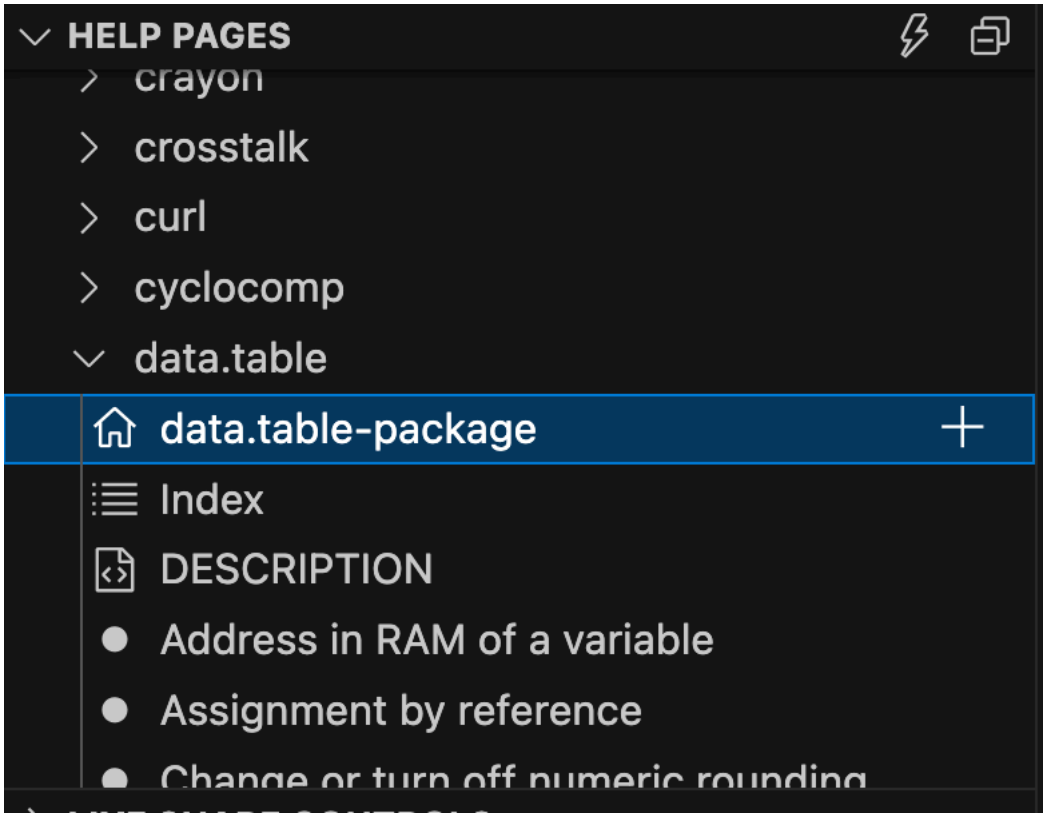
```
r$> install.packages "languageserversetup"
trying URL 'https://cloud.r-project.org/bin/macosx/big-sur-arm64/contrib/4.3/languageserversetup
_0.1.2.tgz'
Content type 'application/x-gzip' length 50526 bytes (49 KB)
==================================================
downloaded 49 KB


The downloaded binary packages are in
        /var/folders/5f/q5pf53k13pz9bgpvkc6q9s5r0000gn/T//Rtmp30PAEM/downloaded_packages
```

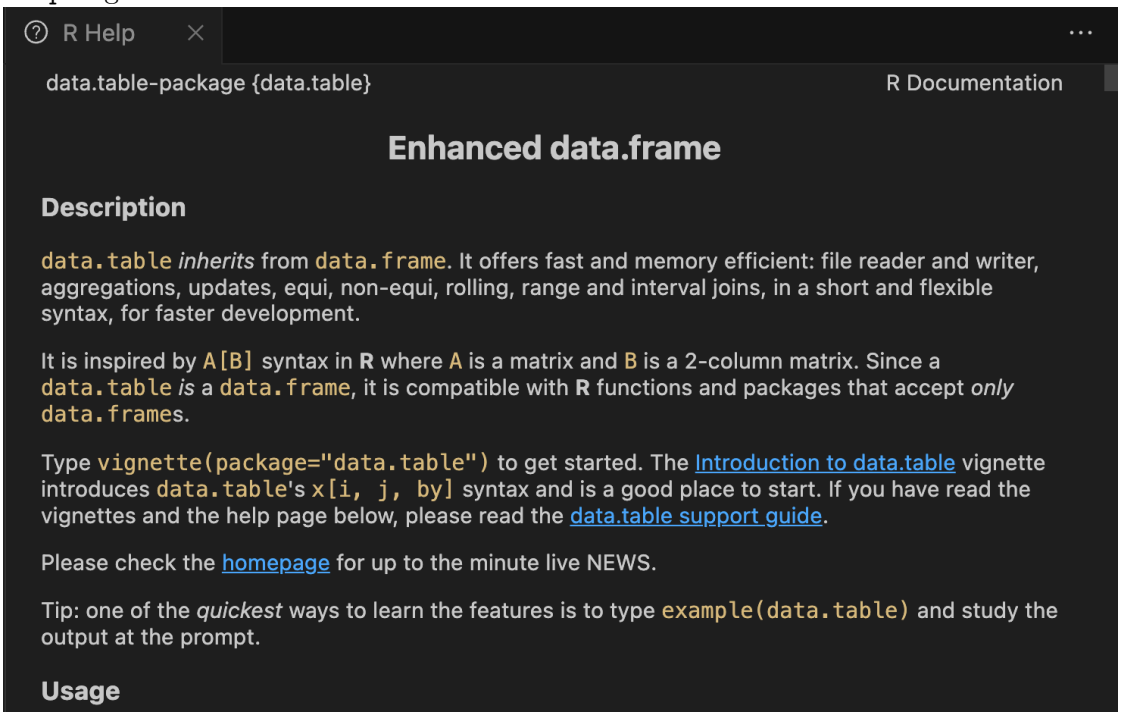
```
r$> languageserversetup::languageserver_install()
This will attempt to use:
source('https://install-github.me/REditorSupport/languageserver')
to install REditorSupport/languageserver into:
/Users/lifengren/languageserver-library
All dependencies will also be installed there

Do you agree?

(yes/No/cancel) yes
The install-github.me service is deprecated, please stop using it.
Downloading GitHub repo REditorSupport/languageserver@master
These packages have more recent versions available.
It is recommended to update all of them.
Which would you like to update?

 1: All

 2: CRAN packages only

 3: None

 4: lintr       (3.1.0   -> 3.1.1 ) [c(name = "lintr", repos = "https://cloud.r-project.org", pk
g_type = "both", sha = "3.1.0")]
 5: stringi     (1.7.12  -> 1.8.2 ) [c(name = "stringi", repos = "https://cloud.r-project.org",
pkg_type = "both", sha = "1.7.12")]
 6: withr       (2.5.0   -> 2.5.2 ) [c(name = "withr", repos = "https://cloud.r-project.org", pk
r$> languageserversetup::languageserver_add_to_rprofile()
This will append the following code:

# LanguageServer Setup Start (do not change this chunk)
# to remove this, run languageserversetup::remove_from_rprofile
if (requireNamespace('languageserversetup', quietly = TRUE)) {
  options(langserver_library = '/Users/lifengren/languageserver-library')
  languageserversetup::languageserver_startup()
  unloadNamespace('languageserversetup')
}
# LanguageServer Setup End

to: /Users/lifengren/.Rprofile

Do you agree?

(yes/No/cancel) yes
r$>
r$>
```

```r
install.packages("languageserversetup")
languageserversetup::languageserver_install()
languageserversetup::languageserver_add_to_rprofile()
```

2.2 Get Familiar with `R` in `VS Code`

- `Help` Pages





- New R terminal

2.3 Switch Gear to `VS Code`

- Run selected R code in `VS Code`
  - Windows: `Ctrl + Enter` to run selected code
  - Mac: `Command + Enter` to run selected code
- Run all R code in `VS Code`:
  - Windows: `Ctrl + Shift + S` to run all code
  - Mac: `Command + Shift + S` to run all code

I assume you all have the background of `R`, and I assume the first half of APEC 8222 taught you some text analysis in `R`, so I will start using `R` to do some NLP application, and gradually switch to `Python` to do the same thing.

For more information on how `R` and `Python` packages are equivalent, please refer to this website:

- https://aeturrell.github.io/coding-for-economists/coming-from-r.html

---

3 Introduction to Natural Language Processing (NLP)

NLP is one of the major branches of Artificial Intelligence (AI). The following table shows each branch with a brief description and examples of applications:

Branch of AI	Description	Examples of Applications
Machine Learning (ML)	Algorithms that enable computers to learn from and make predictions or decisions based on data.	- Recommendation systems- Self-driving cars- Predictive analytics
Natural Language Processing (NLP)	Focuses on the interaction between computers and human language, enabling understanding, interpretation, and generation of human language.	- Speech recognition- Language translation- Sentiment analysis
Computer Vision	Deals with how computers can gain high-level understanding from digital images or videos.	- Facial recognition systems- Medical image analysis- Autonomous vehicle navigation
Robotics	Involves designing and building robots that perform tasks, often using AI for perception and decision-making.	- Industrial robots for manufacturing- Surgical robots in healthcare- Exploration robots in space missions
Expert Systems	AI systems that simulate the decision-making ability of a human expert in specific domains.	- Medical diagnosis systems- Financial investment advising- Weather prediction models
Neural Networks and Deep Learning	A subset of ML based on artificial neural networks, particularly useful for handling large amounts of unstructured data.	- Voice recognition and assistants- Advanced game-playing AI (like chess and Go)- Enhanced medical diagnoses

- This table provides a concise overview of the major branches of AI, their core functionalities, and examples of where they are applied in various industries and domains. Note that there will be overlap between these branches, and some applications may use multiple branches of AI.
- The famous transformer model, is based on a paper called Attention is All You Need, which was published in 2017 by Google. The transformer model is a type of neural network architecture that is based on the concept of attention. The transformer model is the foundation of many of the most advanced NLP models today, including BERT, GPT-3, and T5.
  - It was designed by solving a translation problem, but it can be used for many other NLP tasks, including text classification, question answering, and summarization.
  - We will talk more about it in the next lecture.

3.1 Foundamental Concepts in NLP

Here's a table that defines each NLP term, along with an example for each:

Term	Definition	Example
Type	A distinct element of vocabulary in a text, representing a unique word.	In "cat, dog, dog", "cat" and "dog" are two types.
Token	An instance of a sequence of characters in a text, often corresponding to a word or symbol.	In "I love AI", "I", "love", and "AI" are tokens.
Term	A word or phrase used in a text or a collection of texts (corpus).	"Artificial intelligence" is a term in computer science texts.
Document	A single text or record in a dataset, like an article, an email, or a webpage.	An individual Wikipedia article is a document.
Corpus	A collection of documents, often used as a dataset in NLP.	A collection of all articles from a news website forms a corpus.
Bag of Words (BOW)	A model treating text as a collection of words without considering syntax or word order but maintaining frequency.	In "cat and dog", BOW represents two words: "cat" (1), "dog" (1).
Term Frequency (TF)	The frequency of a term in a document.	In a document with 100 words, where the word "AI" appears 5 times, TF for "AI" is 5/100.

Term	Definition	Example
Inverse Document Frequency (IDF)	A measure of how much information a term provides by considering how common or rare it is across all documents.	"AI" appearing in 1 out of 1000 documents has higher IDF than if it appears in 100 out of 1000.
TF-IDF	A statistical measure used to evaluate the importance of a word to a document in a corpus; combines TF and IDF.	High TF-IDF for "neural network" in a document indicates its importance in that document within the given corpus.
Stop Words	Commonly used words in a language that are filtered out before processing text.	Words like "is", "and", "the" are often considered stop words.
Stemming	The process of reducing words to their base or root form, often crudely by chopping off word endings.	"Running", "runner" stem to "run".
Lemmatization	Similar to stemming but more sophisticated, reducing words to their base or dictionary form.	"Better" is lemmatized to "good".
Part of Speech (POS) Tagging	The process of marking up a word in a text as corresponding to a particular part of speech.	In "The quick brown fox", "quick" is tagged as an adjective.
Named Entity Recognition (NER)	The process of identifying and classifying key information (entities) in text into predefined categories.	In "Apple Inc. was founded by Steve Jobs", "Apple Inc." is recognized as an organization.
Word Embedding	A technique in NLP where words or phrases are encoded as real-valued vectors in a predefined vector space.	Each word in a corpus is represented as a vector in a multi-dimensional space.

3.2 Very Very Basic NLP Application in `R` and `Python`

NLP has many applications, including but not limited to: - Document classification - Sentiment analysis - Author identification - Question answering - Topic modeling

3.2.1 Some Rules and Math

3.2.1.1 Zipf's Law

$$\text{word frequency} \propto \frac{1}{\text{word rank}}.$$

It is usually found that the most common word occurs approximately twice as often as the next common one, three imes as often as the third most common, and so on.

3.2.1.2 TF-IDF

Term frequency, $\text{tf}(t, d)$, is the relative frequency of term t within document d,

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

where $f_{t,d}$ is the raw count of a term in a document, i.e., the number of times that term t occurs in document d. Note the denominator is simply the total number of terms in document d (counting each occurrence of the same term separately).

Inverse Document Frequency

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|} = \log_{10}(\text{count}(t, d) + 1)$$

with

- N : total number of documents in the corpus $N = |D|$
- $|\{d \in D : t \in d\}|$: number of documents where the term t appears (i.e., $\text{tf}(t, d) \neq 0$).

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D) = \log_{10} \frac{N}{\text{df}_t}$$

Example:

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

Corpus of Shakespeare plays, ranging from extremely informative words that occur in only one play like Romeo, to those that occur in a few like salad or Falstaff, to those that are very common like fool or so common as to be completely non-discriminative since they occur in all 37 plays like good or sweet. (Source: https://web.stanford.edu/~jurafsky/slp3/14.pdf)

3.2.1.3 Cosine Similarity

$$\text{score}(q, d) = \cos(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q}}{|\mathbf{q}|} \cdot \frac{\mathbf{d}}{|\mathbf{d}|}$$

This is equivlaent to the following:

$$\text{score}(q, d) = \sum_{t \in \mathbf{q}} \frac{\text{tf} - \text{idf}(t, q)}{\sqrt{\sum_{q_i \in q} \text{tf} - \text{idf}^2 (q_i, q)}} \cdot \frac{\text{tf} - \text{idf}(t, d)}{\sqrt{\sum_{d_i \in d} \text{tf} - \text{idf}^2 (d_i, d)}}$$

$$\text{score}(q, d) = \sum_{t \in q} \frac{\text{tf} - \text{idf}(t, d)}{|d|}$$

So, for the following Query and provided document, we can calculate if the document is relevant to the query:

Query	Documents	Content
Sweet love	Doc 1	Sweet sweet nurse! Love?
	Doc 2	Sweet sorrow
	Doc 3	How sweet is love?
	Doc 4	Nurse!

3.2.2 Application: Sentiment Analysis

Nowadays, many economics papers uses the text analysis to do sentiment analysis, which requires VERY BIG DATA. For example, the following paper uses the text analysis to do sentiment analysis:

- Shapiro, A. H., Sudhof, M., & Wilson, D. J. (2022). Measuring news sentiment. Journal of econometrics, 228(2), 221-243.
- Benhima, K., & Cordonier, R. (2022). News, sentiment and capital flows. Journal of International Economics, 137, 103621.
- Chen, C. Y. H., Härdle, W. K., & Klochkov, Y. (2022). Sonic: Social network analysis with influencers and communities. Journal of Econometrics, 228(2), 177-220
- Ash, E., & Hansen, S. (2023). Text algorithms in economics. Annual Review of Economics, 15, 659-688.

	Document 1					Document 2				
word	**count**	**tf**	**df**	**idf**	**tf-idf**	**count**	**tf**	**df**	**idf**	**tf-idf**
love	1	0.301	2	0.301	0.091	0	0	2	0.301	0
sweet	2	0.477	3	0.125	0.060	1	0.301	3	0.125	0.038
sorrow	0	0	1	0.602	0	1	0.301	1	0.602	0.181
how	0	0	1	0.602	0	0	0	1	0.602	0
nurse	1	0.301	2	0.301	0.091	0	0	2	0.301	0
is	0	0	1	0.602	0	0	0	1	0.602	0
$\lvert d_1\rvert = \sqrt{.091^2 + .060^2 + .091^2} = .141$						$\lvert d_2\rvert = \sqrt{.038^2 + .181^2} = .185$				

Figure 14.2 Computation of tf-idf for nano-documents 1 and 2, using Eq. 14.3, Eq. 14.4, and Eq. 14.5.

Doc	$\lvert d\rvert$	**tf-idf**(*sweet*)	**tf-idf**(*love*)	**score**
1	.141	.060	.091	1.07
3	.274	.038	.091	0.471
2	.185	.038	0	0.205
4	.090	0	0	0

Figure 14.3 Ranking documents by Eq. 14.9.

Figure 1: Source: https://web.stanford.edu/~jurafsky/slp3/14.pdf

So, we will use Sentiment Analysis as the realzied application of NLP in this lecture.

Get the Sentiment Score (Basic Method)

$$\text{sentiment}_{\ i} = \frac{\#\text{ Positive terms}_{\ i} - \#\text{ Negative terms}_{\ i}}{\#\text{ Positive terms}_{\ i} + \#\text{ Negative terms}_{\ i}}$$

Let us start with a simple example of sentiment analysis in `R`.

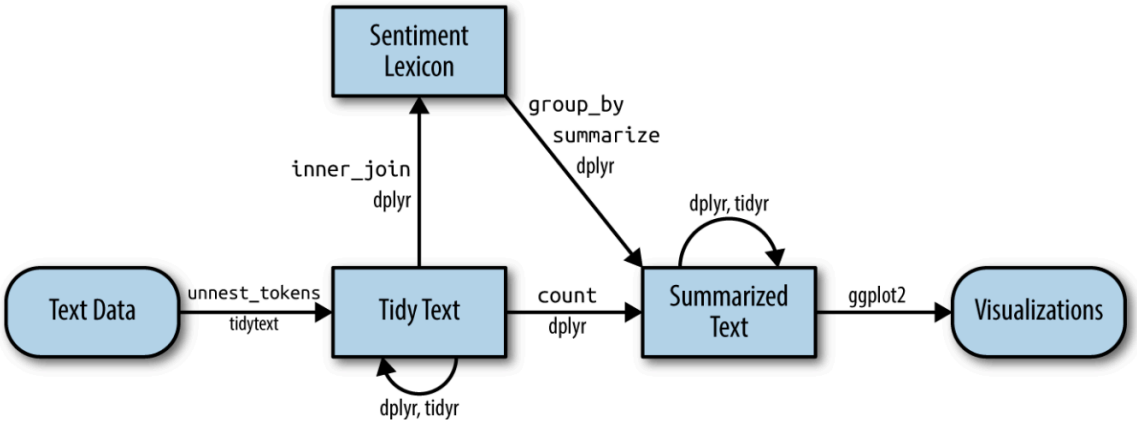


Figure 2: Source: Text Mining with R: A Tidy Approach" was written by Julia Silge and David Robinson

---

Note: We are going to only applied the Dictionary/Lexical Methods in this lecture. There are other advanced methods using other machine learning methods, and we are not going to cover them in this lecture. Refer to the following repository for more works:

- https://github.com/xiamx/awesome-sentiment-analysis

3.2.2.1 R Example: Jane Austen's Books-Emma's Sentiment Analysis

This is an exmaple from Introduction to tidytext by Julia Silge and David Robinson. The book is available online at https://www.tidytextmining.com/.

1. **Loading and Preparing the Data**:

```
# Load required libraries
library(janeaustenr)
library(textdata)
library(tidytext)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(stringr)
library(tidyr)
library(ggplot2)
library(wordcloud)
```

```
Loading required package: RColorBrewer
```

```
library(reshape2)
```

```
Attaching package: 'reshape2'

The following object is masked from 'package:tidyr':

    smiths
```

```
original_books <- austen_books() %>%
  group_by(book) %>%
  mutate(line = row_number(),
         chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]", ignore_case =
```

```
  ungroup()
```

- This code loads Jane Austen's books using the `austen_books` function.
- It groups the data by each book and adds two new columns: `line` for line numbers and `chapter` for chapter numbers.
- The `chapter` number is determined using regex to identify chapter headings in the text.

2. **Tokenization**:

```
tidy_books <- original_books %>%
  unnest_tokens(word, text)
```

- This part tokenizes the text into words. Each word in the text is separated into its own row, creating a tidy data frame of one-token-per-row.

3. **Cleaning the Data**:

```
cleaned_books <- tidy_books %>%
  anti_join(get_stopwords())
```

```
Joining with `by = join_by(word)`
```

- This code removes common stop words from the data to focus on more meaningful words.

4. **Word Count**:

```
cleaned_books %>%
  count(word, sort = TRUE)
```

```
# A tibble: 14,375 x 2
   word      n
   <chr> <int>
 1 mr     3015
 2 mrs    2446
 3 must   2071
 4 said   2041
 5 much   1935
 6 miss   1855
 7 one    1831
 8 well   1523
```

```
 9 every  1456
10 think  1440
# i 14,365 more rows
```

- Counts and sorts the words in the cleaned data. This helps in identifying the most common words in the books.

5. **Sentiment Analysis - Positive Words in 'Emma'**:

```
positive <- get_sentiments("bing") %>%
  filter(sentiment == "positive")

tidy_books %>%
  filter(book == "Emma") %>%
  semi_join(positive) %>%
  count(word, sort = TRUE)
```

```
Joining with `by = join_by(word)`
```

```
# A tibble: 668 x 2
   word         n
   <chr>    <int>
 1 well       401
 2 good       359
 3 great      264
 4 like       200
 5 better     173
 6 enough     129
 7 happy      125
 8 love       117
 9 pleasure   115
10 right       92
# i 658 more rows
```

- Extracts positive words from the Bing lexicon.
- Counts the frequency of positive words in the book "Emma."

6. **Sentiment Analysis Across Books**:

```
bing <- get_sentiments("bing")

janeaustensentiment <- tidy_books %>%
  inner_join(bing) %>%
```

```
  count(book, index = line %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

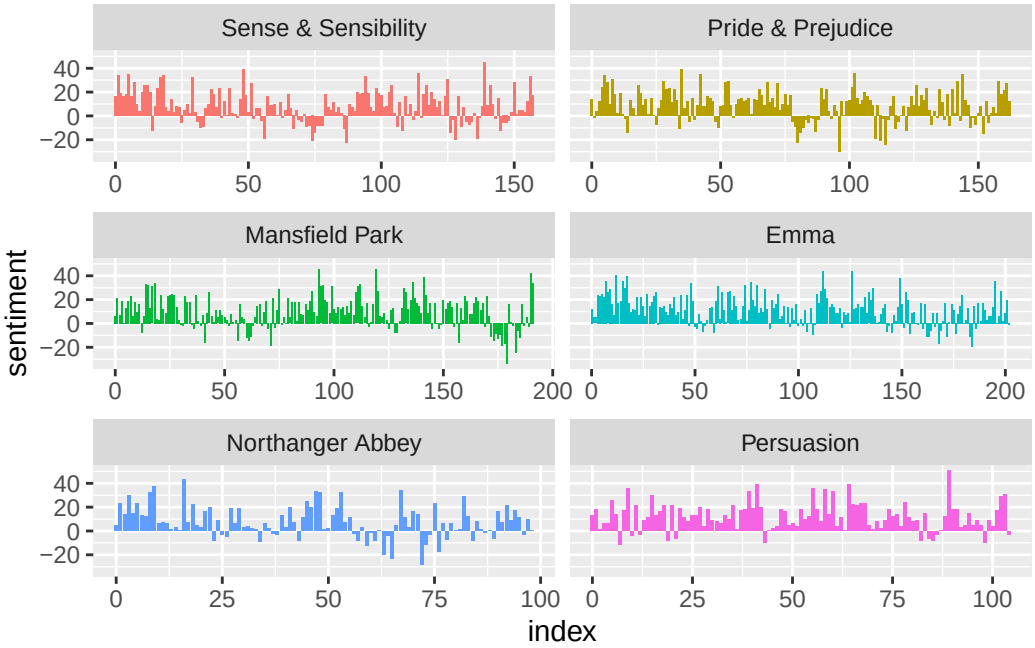
```
Joining with `by = join_by(word)`
```

```
Warning in inner_join(., bing): Detected an unexpected many-to-many relationship betweer
i Row 435434 of `x` matches multiple rows in `y`.
i Row 5051 of `y` matches multiple rows in `x`.
i If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.
```

- This section joins the tidy book data with the Bing sentiment lexicon.
- It calculates the sentiment scores for sections of text across different books.

7. **Sentiment Visualization**:

```
ggplot(janeaustensentiment, aes(index, sentiment, fill = book)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_x")
```



- Visualizes the sentiment data using a bar chart.

8. **Word Sentiment Counts**:

```
bing_word_counts <- tidy_books %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE)
```

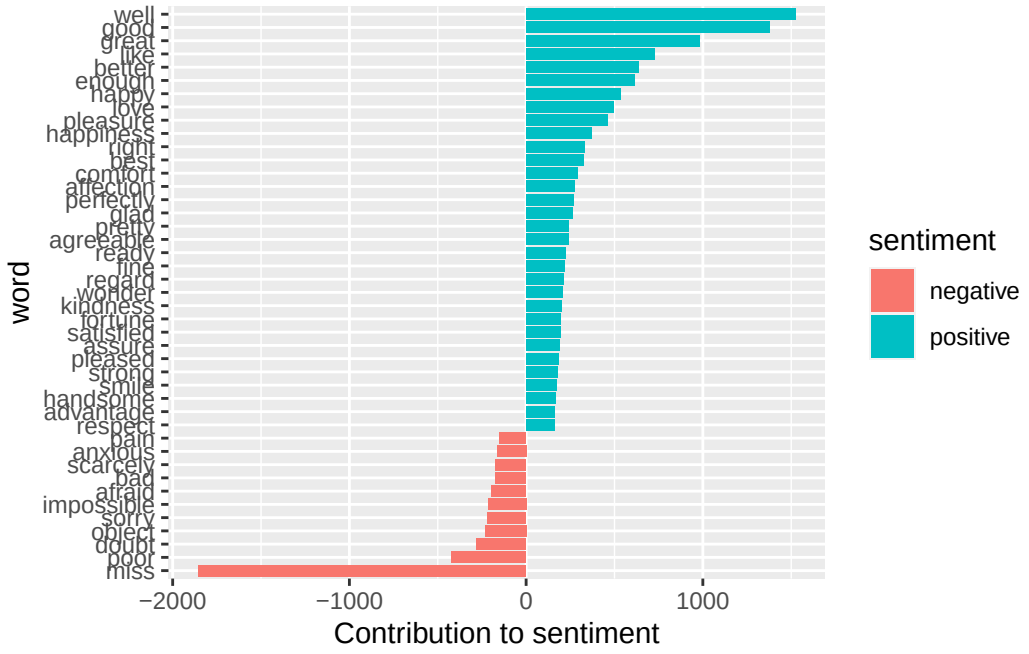
```
Joining with `by = join_by(word)`
```

```
Warning in inner_join(., bing): Detected an unexpected many-to-many relationship betweer
i Row 435434 of `x` matches multiple rows in `y`.
i Row 5051 of `y` matches multiple rows in `x`.
i If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.
```

- Counts the occurrence of words associated with each sentiment.

9. **Sentiment Contribution Visualization**:

```
bing_word_counts %>%
  filter(n > 150) %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col() +
  coord_flip() +
  labs(y = "Contribution to sentiment")
```



- Visualizes the contribution of each word to the overall sentiment.

10. **Word Cloud Generation**:

```
cleaned_books %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))
```

```
Warning in wordcloud(word, n, max.words = 100): much could not be fit on page.
It will not be plotted.
```

```
Warning in wordcloud(word, n, max.words = 100): good could not be fit on page.
It will not be plotted.
```

```
Warning in wordcloud(word, n, max.words = 100): always could not be fit on
page. It will not be plotted.
```

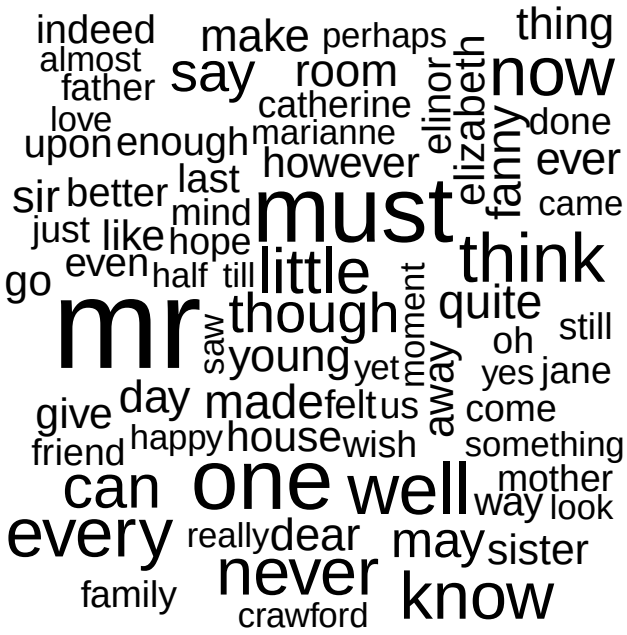
```
Warning in wordcloud(word, n, max.words = 100): great could not be fit on page.
It will not be plotted.
```

```
Warning in wordcloud(word, n, max.words = 100): thought could not be fit on
page. It will not be plotted.
```

```
Warning in wordcloud(word, n, max.words = 100): nothing could not be fit on
page. It will not be plotted.
```

```
Warning in wordcloud(word, n, max.words = 100): emma could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): two could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): lady could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): without could not be fit on
page. It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): said could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): miss could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): soon could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): mrs could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): many could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): first could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): time could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): sure could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): long could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): place could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): home could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): seemed could not be fit on
page. It will not be plotted.
```

```
Warning in wordcloud(word, n, max.words = 100): see could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): feelings could not be fit on
page. It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): man could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): shall could not be fit on page.
It will not be plotted.

Warning in wordcloud(word, n, max.words = 100): might could not be fit on page.
It will not be plotted.
```



- Creates a word cloud from the most frequent words in the cleaned books data.

11. **Comparison Cloud**:

```
tidy_books %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("#F8766D", "#00BFC4"),
                   max.words = 100)
```

```
Joining with `by = join_by(word)`

Warning in inner_join(., bing): Detected an unexpected many-to-many relationship betweer
i Row 435434 of `x` matches multiple rows in `y`.
i Row 5051 of `y` matches multiple rows in `x`.
i If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.

Warning in comparison.cloud(., colors = c("#F8766D", "#00BFC4"), max.words =
100): favour could not be fit on page. It will not be plotted.

Warning in comparison.cloud(., colors = c("#F8766D", "#00BFC4"), max.words =
100): admiration could not be fit on page. It will not be plotted.

Warning in comparison.cloud(., colors = c("#F8766D", "#00BFC4"), max.words =
100): delight could not be fit on page. It will not be plotted.

Warning in comparison.cloud(., colors = c("#F8766D", "#00BFC4"), max.words =
100): superior could not be fit on page. It will not be plotted.

Warning in comparison.cloud(., colors = c("#F8766D", "#00BFC4"), max.words =
100): gratitude could not be fit on page. It will not be plotted.
```

- Generates a comparison cloud to visually compare the frequency of words associated with different sentiments.

- Negates the count for negative sentiment words for visualization purposes.
- Reorders words based on their count.
- Creates a horizontal bar plot with `ggplot2` showing the contribution of each word to the overall sentiment, distinguishing between positive and negative sentiments.

3.2.2.2 `Python` Example: `vaderSentiment` for social media sentiment analysis

- `vaderSentiment` is a Python library that is optimized for social media sentiment analysis. It uses a lexicon of words that are labeled according to their semantic orientation as either positive or negative. It also incorporates rules for handling sentiment intensity expressed through punctuation, capitalization, degree modifiers, and conjunctions.
- https://github.com/cjhutto/vaderSentiment/tree/master
- Install it: `pip install vaderSentiment`

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# --- examples -------
sentences = ["VADER is smart, handsome, and funny.",  # positive sentence example
             "VADER is smart, handsome, and funny!",  # punctuation emphasis handled corre
             "VADER is very smart, handsome, and funny.", # booster words handled correctl
             "VADER is VERY SMART, handsome, and FUNNY.",  # emphasis for ALLCAPS handled
             "VADER is VERY SMART, handsome, and FUNNY!!!", # combination of signals - VAD
             "VADER is VERY SMART, uber handsome, and FRIGGIN FUNNY!!!", # booster words &
             "VADER is not smart, handsome, nor funny.",  # negation sentence example
             "The book was good.",  # positive sentence
             "At least it isn't a horrible book.",  # negated negative sentence with contr
             "The book was only kind of good.", # qualified positive sentence is handled c
             "The plot was good, but the characters are uncompelling and the dialog is not
             "Today SUX!",  # negative slang with capitalization emphasis
             "Today only kinda sux! But I'll get by, lol", # mixed sentiment example with
             "Make sure you :) or :D today!",  # emoticons handled
             "Catch utf-8 emoji such as such as   and   and ",  # emojis handled
             "Not bad at all"  # Capitalized negation
             ]

analyzer = SentimentIntensityAnalyzer()
for sentence in sentences:
    vs = analyzer.polarity_scores(sentence)
    print("{:-<65} {}".format(sentence, str(vs)))
```

```
VADER is smart, handsome, and funny.---------------------------- {'neg': 0.0, 'neu': 0.254,
VADER is smart, handsome, and funny!---------------------------- {'neg': 0.0, 'neu': 0.248,
VADER is very smart, handsome, and funny.----------------------- {'neg': 0.0, 'neu': 0.299,
VADER is VERY SMART, handsome, and FUNNY.----------------------- {'neg': 0.0, 'neu': 0.246,
VADER is VERY SMART, handsome, and FUNNY!!!---------------------- {'neg': 0.0, 'neu': 0.233,
VADER is VERY SMART, uber handsome, and FRIGGIN FUNNY!!!-------- {'neg': 0.0, 'neu': 0.294,
VADER is not smart, handsome, nor funny.------------------------ {'neg': 0.646, 'neu': 0.354
The book was good.---------------------------------------------- {'neg': 0.0, 'neu': 0.508,
At least it isn't a horrible book.------------------------------ {'neg': 0.0, 'neu': 0.678,
The book was only kind of good.--------------------------------- {'neg': 0.0, 'neu': 0.697,
The plot was good, but the characters are uncompelling and the dialog is not great. {'neg': 0
Today SUX!------------------------------------------------------ {'neg': 0.779, 'neu': 0.221
Today only kinda sux! But I'll get by, lol----------------------- {'neg': 0.127, 'neu': 0.556
Make sure you :) or :D today!----------------------------------- {'neg': 0.0, 'neu': 0.294,
Catch utf-8 emoji such as such as  and  and ----------------- {'neg': 0.0, 'neu': 0.615, '
Not bad at all-------------------------------------------------- {'neg': 0.0, 'neu': 0.513,
```

- **Importing VADER**: The `SentimentIntensityAnalyzer` class from the `vaderSentiment` package is imported.
- **Sentences for Analysis**: A list of sentences is defined, showcasing different scenarios like positive/negative sentiments, emphasis through punctuation and capitalization, booster words, negations, slang, and emoticons.
- **Analyzing Sentiment**: The `SentimentIntensityAnalyzer` is used to compute a `polarity_scores` dictionary for each sentence. This dictionary contains four scores: 'neg' (negative), 'neu' (neutral), 'pos' (positive), and 'compound' (a normalized, composite score).
- **Printing Results**: For each sentence, the sentiment scores are printed alongside the text.

VADER uses a combination of a lexicon (a list of lexical features, e.g., words, emoticons) which are labeled according to their semantic orientation as either positive or negative. VADER not only tells about the Positivity and Negativity score but also tells us about how positive or negative a sentiment is.

VADER analyzes sentiments primarily based on certain key aspects:

- **Lexicon**: Words are scored for their positive or negative sentiment.
- **Rules**: Incorporates grammatical and syntactical rules like capitalization, degree modifiers (e.g., "very"), and contrastive conjunctions (e.g., "but").
- **Emoticons and Emoji**: Interprets commonly used emoticons and emoji, which are significant in social media text.
- **Punctuation**: Recognizes punctuation marks' role in emphasizing sentiment intensity.

Improvement from Jane Austen Example: In the Jane Austen books example (using 'bing' lexicon), the sentiment analysis is mainly based on the presence of words categorized

as positive or negative. This approach might not capture the nuances of sentiment expressed through modern slang, emoticons, emoji, punctuation, and capitalization, which are common in online communication.

VADER, on the other hand, is designed to understand the nuances of social media language. It can interpret a wider range of expressions (like "SUX" or ":-)") and modifiers (like "uber" or ALLCAPS) that significantly impact sentiment intensity. This makes VADER particularly suitable for analyzing text from platforms like Twitter, where such expressions are prevalent.

3.2.2.3 Inclass Exercise:

- Use ChatGPT to generate texts for you to test how VADER works:

```
test_sentences = [
    "This new song is lit   ",
    "Sigh... I guess today was just not my day  ",
    "Wow, that's awesome!!!  ",
    "I can't stand this! So frustrating!  ",
    "IDK what's going on, kinda confused rn  ",
    "LOL, that was hilarious  ",
    "Ugh, Mondays are the worst  ",
    "OMG, I just got the job offer!!!  ",
    "No way, that's cray cray  ",
    "Why is everyone so glum? Cheer up!  "
]
```

- Demo, and Discussion

4 Summary Today and For Next Lecture

This lecture is designed to refresh your memory on `R` about text analysis, and using `R` in `VS Code` to apply NLP method like sentiment analysis.

For next lecture, we will talk about Large Language Models and how can we use it in Economic Research.

5 References

- Text Mining with R: A Tidy Approach" was written by Julia Silge and David Robinson

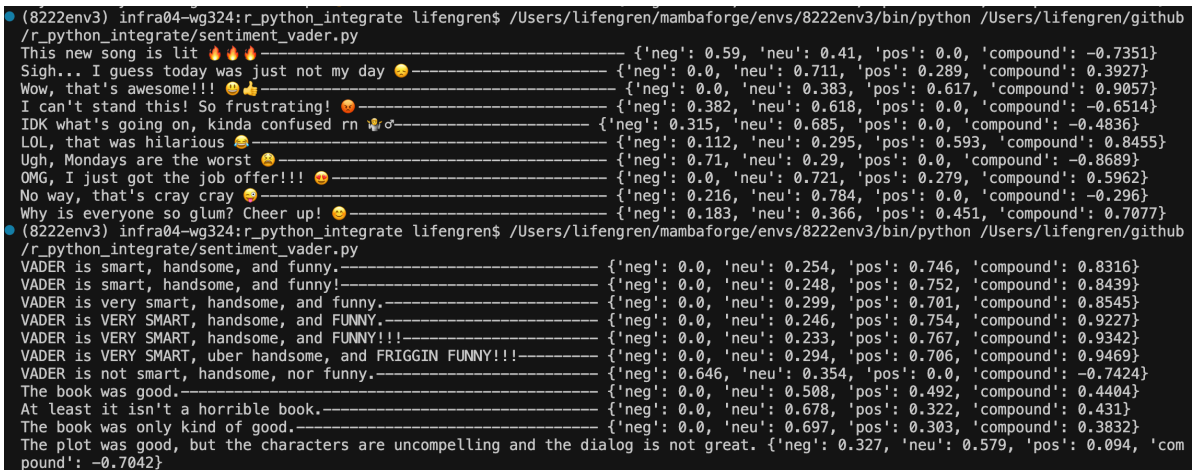


Figure 3: Demo for VADER with ChatGPT generated test texts