Understanding user experience using text mining & analysis

A case study of Amazon.com reviews for smart-home products

Technical documentation

Qian Fu

School of Engineering, University of Birmingham, UK

Yixiu Yu

Miller College of Business, Ball State University, USA

Dong Zhang

School of Maritime Economics and Management, Dalian Maritime University, China

First created: **January 2022**

Last updated: **November 2024**

Table of Contents

List of Figures

Chapter 1

Introduction

This study examines the growing use of smart home products, focusing on user experiences beyond initial adoption. By analysing Amazon.com reviews of robotic vacuum cleaners, it aimed to uncover what drives user satisfaction and dissatisfaction.

Key insights include:

- **Satisfaction dimensions**: Users value functionality, smart capabilities and enhanced performance.
- **Dissatisfaction dimensions**: Common issues include limited “smartness”, poor customer service and functionality issues (e.g. connectivity).

Notably, the concept of **“smartness”** emerges as a double-edged sword - contributing to satisfaction when effective, yet leading to disappointment when poorly implemented.

Through fuzzy-set qualitative comparative analysis (fsQCA), this study offers a comprehensive framework for understanding key user experience dimensions. The methods and insights may be valuable for designers and marketers across various smart home product categories, such as electric vehicles, to inform product design and strategy.

Chapter 2

Methodology

The methodological framework for data processing and text mining applied in study is illustrated in Fig. 1:

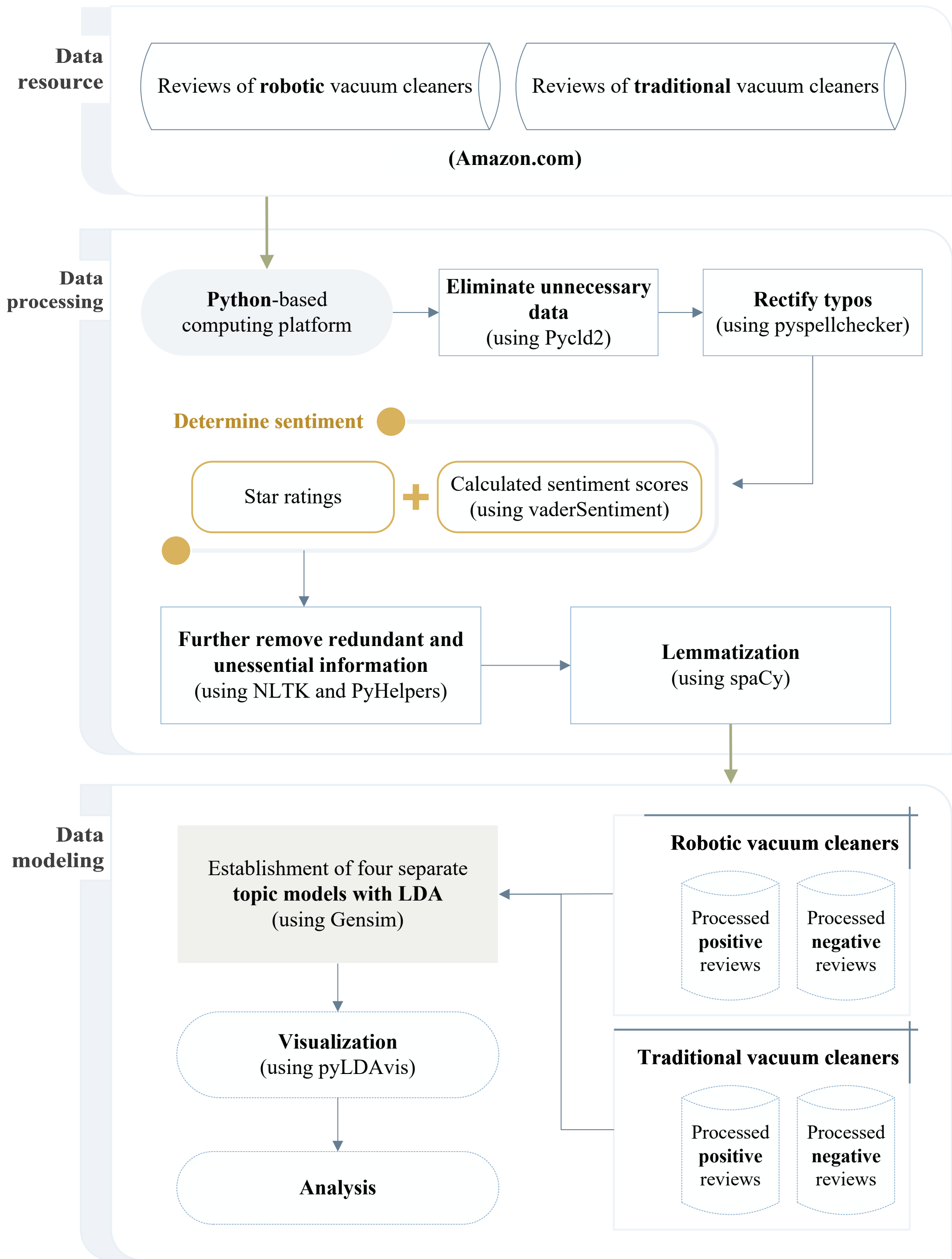

Fig. 1: Text-mining framework applied in this study (Yu, et al., 2024).

Chapter 3

Data

The data used in this study consists of Amazon.com product reviews.

A Jupyter Notebook is provided in the repository, demonstrating a snapshot of the data. For additional examples of the data sets, please refer to the examples provided in the `processor` module.

Furthermore, the notebooks available in the demos directory provide a high-level overview of the data analysis and modeling steps conducted in this study. These notebooks are intended for demonstration purposes only and do not include the full data set.

Chapter 4

Modules

utils	The module provides helper classes/functions for facilitating the implementation of other modules.
processor	The module is used to (pre-)process and I/O management of all the data resources.
modeller	The module is used to apply algorithms on the data generated from *processor*.
analyser	The module is used to analyse the modelling results produced by *modeller*.

4.1 utils

The module provides helper classes/functions for facilitating the implementation of other modules.

4.1.1 Database tools

CustomerReviewsAnalysis([host, port, ...])	Provide a basic PostgreSQL instance for managing data of the project.

CustomerReviewsAnalysis

class src.utils.**CustomerReviewsAnalysis**(*host=None*, *port=None*, *username=None*, *password=None*, *database_name='UoB_CustomerReviewsAnalysis'*, ***kwargs*)

Provide a basic PostgreSQL instance for managing data of the project.

This class inherits from the class pyhelpers.dbms.PostgreSQL.

Parameters

- **host** (*str | None*) – The database host; defaults to None.
- **port** (*int | None*) – The database port; defaults to None.

- **username** (*str | None*) – The database username; defaults to `None`.
- **password** (*str | int | None*) – The database password; defaults to `None`.
- **database_name** (*str*) – The name of the database; defaults to `STFC_DAFNI_ClimaTracks`.
- **kwargs** – [Optional] parameters of the class pyhelpers.dbms.PostgreSQL.

Examples:

```
>>> from src.utils import CustomerReviewsAnalysis
>>> db_instance = CustomerReviewsAnalysis()
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/postgres ... Successfully.
>>> db_instance.database_name
'UoB_CustomerReviewsAnalysis'
>>> # Remote server
>>> db_instance = CustomerReviewsAnalysis()
>>> db_instance.database_name
'UoB_CustomerReviewsAnalysis'
```

Note: No directly defined attributes. See inherited class attributes.

Note: No directly defined methods. See inherited class methods.

4.1.2 General utilities

normalise_text(x)	Normalise textual data.
correct_typo(x)	Correct misspelled words and/or typos.
identify_language(x)	Identify language of textual data.
is_english_word(x)	Check whether a given word is English.
is_english(x)	Check whether a given piece of textual data is written in English.
remove_stopwords(x)	Remove stop words from textual data.
remove_single_letters(x)	Remove all single letters from textual data.
remove_digits(x)	Remove digits from textual data.
lemmatize_text(x[, allowed_postags, detokenized])	Lemmatize textual data.

normalise_text

`src.utils.`**`normalise_text`**`(`*x*`)`

Normalise textual data.

Parameters

x (*str*) – textual data

Returns

normalised text

Return type

str

Examples:

```
>>> from src.utils import normalise_text

>>> # noinspection SpellCheckingInspection
>>> normalise_text('Ítem last a whole 20 minutes')
'Item last a whole 20 minutes'

>>> normalise_text('2000 ft. ²')
'2000 ft 2'
```

correct_typo

`src.utils.`**`correct_typo`**`(`*x*`)`

Correct misspelled words and/or typos.

Parameters

x (*str*) – textual data

Returns

text with misspelled words being corrected

Return type

str

Examples:

```
>>> from src.utils import correct_typo

>>> # noinspection SpellCheckingInspection
>>> correct_typo('We shoud replce letter A with frut apple')
'we should replace letter a with fruit apple'
```

identify_language

`src.utils.`**`identify_language`**`(`*x*`)`

Identify language of textual data.

Parameters

x (*str*) – textual data

Returns

full name of a language

Return type
str or None

Examples:

```
>>> from src.utils import identify_language

>>> identify_language('This is about Amazon product reviews.')
'English'

>>> # noinspection SpellCheckingInspection
>>> identify_language('Se trata de las reseñas de productos de Amazon.')
'Spanish'

>>> identify_language('+-*/')  # None
'Unknown'
```

is_english_word

`src.utils.`**`is_english_word`**(*x*)

Check whether a given word is English.

Parameters
x (*str*) – textual data of a word

Returns
whether `x` is an English word

Return type
bool

Examples:

```
>>> from src.utils import is_english_word

>>> is_english_word(x='apple')
True

>>> is_english_word(x='apples')
True

>>> is_english_word(x='xyz')
False
```

is_english

`src.utils.`**`is_english`**(*x*)

Check whether a given piece of textual data is written in English.

Parameters
x (*str*) – textual data

Returns
whether `x` is written in English

Return type
bool

Examples:

```
>>> from src.utils import is_english

>>> is_english(x='This is about Amazon product reviews.')
True

>>> # noinspection SpellCheckingInspection
>>> is_english(x='ESe trata de las reseñas de productos de Amazon.')
False
```

remove_stopwords

src.utils.**remove_stopwords**(*x*)

Remove stop words from textual data.

Parameters
x (*str*) – textual data

Returns
text without stopwords

Return type
str

Examples:

```
>>> from src.utils import remove_stopwords

>>> remove_stopwords('This is an apple.')
'apple.'

>>> remove_stopwords('There were some apples.')
'apples.'

>>> remove_stopwords("I'm going to school.")
"I'm going school."
```

remove_single_letters

src.utils.**remove_single_letters**(*x*)

Remove all single letters from textual data.

Parameters
x (*str*) – textual data

Returns
text without single letters

Return type
str

Examples:

```
>>> from src.utils import remove_single_letters
>>> from pyhelpers.text import remove_punctuation
```

(continues on next page)

(continued from previous page)

```
>>> remove_single_letters('There is a bug.')
'There is bug.'

>>> remove_single_letters('It is a b c.')
'It is c.'

>>> remove_single_letters(remove_punctuation('It is a b c.'))
'It is'
```

remove_digits

src.utils.**remove_digits**(*x*)

Remove digits from textual data.

Parameters
x (*str*) – textual data

Returns
text without digits

Return type
str

Examples:

```
>>> from src.utils import remove_digits

>>> remove_digits('There are 2 bugs.')
'There are bugs.'

>>> remove_digits("Hello world! 666")
'Hello world!'

>>> remove_digits("Hello world! 666!")
'Hello world! !'
```

lemmatize_text

src.utils.**lemmatize_text**(*x, allowed_postags=None, detokenized=True*)

Lemmatize textual data.

See also https://spacy.io/api/annotation.

Parameters

- **x** (*str*) – textual data
- **allowed_postags** (*list or None*) – allowed postags, defaults to `None`
- **detokenized** (*bool*) – whether to detokenize the texts, defaults to `True`

Returns
lemmatized textual data

Return type
str

Examples:

```
>>> from src.utils import lemmatize_text

>>> lemmatize_text('This is an apple.')
'apple'

>>> lemmatize_text('There were some apples.')
'be apple'

>>> lemmatize_text("I'm going to school.")
'go school'
```

4.1.3 Misc.

save_partitioned_df(data, path_to_file[, ...])	Split (a very large) dataframe into smaller partitions and save the partitions to separate files.
load_partitioned_df(path_to_file[, verbose])	Load partitions of data and concatenate them into one dataframe.

save_partitioned_df

`src.utils.`**`save_partitioned_df`**(*data*, *path_to_file*, *number_of_chunks=5*, *verbose=False*, ***kwargs*)

Split (a very large) dataframe into smaller partitions and save the partitions to separate files.

Parameters

- **data** (*pandas.DataFrame*) – dataframe
- **path_to_file** (*str or os.PathLike[str]*) – pathname of a file, or pathname of a directory where partitioned data files are to be saved
- **number_of_chunks** (*int*) – number of chunks/partitions, defaults to `5`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

See also:

- Examples of the method `LatentDirichletAllocation.make_evaluation_summary()`.

load_partitioned_df

`src.utils.`**`load_partitioned_df`**(*path_to_file*, *verbose=False*, ***kwargs*)

Load partitions of data and concatenate them into one dataframe.

Parameters

- **path_to_file** (*str*) – pathname of a directory where partitioned data files are saved, or pathname of a target file
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

dataframe concatenated from partitions

Return type
pandas.DataFrame

See also:

- Examples of the methods `LatentDirichletAllocation.make_evaluation_summary()` and `LatentDirichletAllocation.fetch_evaluation_summary()`.

4.2 processor

The module is used to (pre-)process and I/O management of all the data resources.

`_Reviews`([db_instance, update, verbose, ...])	A class for preprocessing the data of reviews collected from Amazon.com.
`RoboticVacuumCleaners`([load_preprocd_data, ...])	Process the reviews of *robot vacuum cleaners*.
`TraditionalVacuumCleaners`([...])	Process the reviews of *traditional vacuum cleaners*.
`SmartThermostats`([load_preprocd_data])	Process the reviews of *smart thermostats*.

4.2.1 _Reviews

class src.processor.**_Reviews**(*db_instance=None, update=False, verbose=True, load_preprocd_data=False, load_prep_data=False, load_raw_data=False, verified_reviews_only=False, word_count_threshold=20, dual_scale=False, use_db=False, **kwargs*)

A class for preprocessing the data of reviews collected from Amazon.com.

Parameters

- **db_instance** (*None | CustomerReviewsAnalysis*) – An instance of the project database. Defaults to `None`.
- **update** (*bool | int*) – Whether to reprocess the original data file(s). Defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information in the console. Defaults to `True`.
- **load_preprocd_data** (*bool*) – Whether to load the preprocessed data. Defaults to `False`.
- **load_prep_data** (*bool*) – Whether to load the preparatory data. Defaults to `False`.
- **load_raw_data** (*bool*) – Whether to load the raw data. Defaults to `False`.
- **verified_reviews_only** (*bool*) – Whether to consider only the verified reviews; defaults to `False`.
- **word_count_threshold** (*int*) – Word count in a review, beyond which the review is not considered for further analysis. Defaults to `20`.

- **dual_scale** (*bool*) – Whether the sentiment is determined based on both rating and VADER sentiment score. Defaults to `False`.
- **use_db** (*bool*) – Whether to use the database. Defaults to `False`.
- **kwargs** – [Optional] Parameters for the methods: `load_prep_data()`, `load_preprocd_data()` and `load_raw_data()`.

Variables

- **raw_column_name_changes** (*dict*) – Changes in column names of the raw data.
- **column_name_changes** (*dict*) – Changes in column names of the preparatory data.
- **index_names** (*list*) – Names of the columns used as index when stored in the database.
- **sentiment_column_names** (*list*) – Names of the columns indicating sentiment.
- **verified_reviews_only** (*bool*) – Whether to consider only the verified reviews.
- **word_count_threshold** (*int*) – Review word count threshold; reviews longer than this are excluded.
- **dual_scale** (*bool*) – Whether sentiment analysis uses both rating and VADER scores.
- **use_db** (*bool*) – Whether the class instance uses the database.
- **db_instance** (*None | CustomerReviewsAnalysis*) – Instance of the project database.
- **raw_data** (*pandas.DataFrame | None*) – Raw data loaded from the source.
- **prep_data** (*pandas.DataFrame | None*) – Data prepared for preprocessing.
- **preprocd_data** (*pandas.DataFrame | None*) – Preprocessed data with sentiment labels.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners
>>> rvc = RoboticVacuumCleaners()
>>> rvc.PRODUCT_NAME
'Robotic vacuum cleaners'
>>> rvc.preprocd_data.shape
(101608, 19)
>>> rvc = RoboticVacuumCleaners(verified_reviews_only=True)
>>> rvc.preprocd_data.shape
(89988, 19)
>>> tvc = TraditionalVacuumCleaners()
>>> tvc.PRODUCT_NAME
'Traditional vacuum cleaners'
>>> tvc.preprocd_data.shape
```

(continues on next page)

(continued from previous page)

```
(146656, 19)
>>> tvc = TraditionalVacuumCleaners(verified_reviews_only=True)
>>> tvc.preprocd_data.shape
(131998, 19)
```

Attributes:

ORIGINAL_REVIEW_COLUMN_NAME	Default column name of original review text.
PROCESSED_REVIEW_COLUMN_NAME	Default column name of preprocessed review text.
PRODUCT_CATEGORY	Category of the product.
PRODUCT_NAME	Name of the product.
PRODUCT_TYPE	Type of the product.
SCHEMA_NAME	Schema name.
SENTIMENT_COLUMN_NAME	Default column name of sentiment label.
SQL_QUERY	PostgreSQL query statement to read the whole table.
TABLE_IN_QUERY	Full table in PostgreSQL query statement.
TABLE_NAME	Table name.
VADER_COLUMN_NAME	Default column name of VADER sentiment score.

_Reviews.ORIGINAL_REVIEW_COLUMN_NAME

`_Reviews.ORIGINAL_REVIEW_COLUMN_NAME: str = ''`

Default column name of original review text.

_Reviews.PROCESSED_REVIEW_COLUMN_NAME

`_Reviews.PROCESSED_REVIEW_COLUMN_NAME: str = 'review_text'`

Default column name of preprocessed review text.

_Reviews.PRODUCT_CATEGORY

`_Reviews.PRODUCT_CATEGORY: str | None = None`

Category of the product.

_Reviews.PRODUCT_NAME

`_Reviews.PRODUCT_NAME: str = 'Product name'`

Name of the product.

_Reviews.PRODUCT_TYPE

`_Reviews.PRODUCT_TYPE: str | None = None`

Type of the product.

_Reviews.SCHEMA_NAME

`_Reviews.SCHEMA_NAME: str = '_reviews'`

Schema name.

_Reviews.SENTIMENT_COLUMN_NAME

`_Reviews.SENTIMENT_COLUMN_NAME: str = 'sentiment'`

Default column name of sentiment label.

_Reviews.SQL_QUERY

`_Reviews.SQL_QUERY: str = 'SELECT * FROM "_reviews"."_product_name"'`

PostgreSQL query statement to read the whole table.

_Reviews.TABLE_IN_QUERY

`_Reviews.TABLE_IN_QUERY: str = '"_reviews"."_product_name"'`

Full table in PostgreSQL query statement.

_Reviews.TABLE_NAME

`_Reviews.TABLE_NAME: str = '_product_name'`

Table name.

_Reviews.VADER_COLUMN_NAME

`_Reviews.VADER_COLUMN_NAME: str = 'vs_compound_score'`

Default column name of VADER sentiment score.

Methods:

cdd(*subdir[, mkdir])	Get the full pathname of a directory (or file) under the default data directory.
convert_to_integer(column_names[, int_type, ...])	Convert float values to integers.
correct_identified_typos(review_text[, ...])	Correct typos that have been identified.
determine_sentiment([dual_scale, ...])	Determine the sentiment of each product review.
get_descriptive_stats([data, by, ...])	Get some descriptive statistics.
get_ratings_stats(data, group_label[, ...])	Calculate proportions of different ratings by year or month.
get_vader_sentiment_score([...])	Add calculated VADER sentiment score to the preprocessed data.
if_is_verified_note()	Returns a note message indicating whether the data is considered verified only.
load_prep_data([before_date, ...])	Load the preparatory version of the product reviews data.
load_preprocd_data([verified_reviews_only, ...])	Read the preprocessed product reviews.
load_raw_data([index_columns, ...])	Reads the original version (raw data) of product reviews.
make_prep_data([ret_prep_data, verbose])	Make preparatory data from the raw data.
parse_review_date([column_name, ...])	Parse the information about dates for each record of the product reviews.
preprocess_prep_data([...])	Preprocess the preparatory data.
preprocess_review_text([rm_punctuation, ...])	Process review text.
read_raw_data(path_to_file[, verbose])	Read and preprocess the original product review data.
regulate_people_found_helpful([column_ı ...])	Regulates the data regarding how many people found reviews helpful.
remove_non_english_reviews([...])	Remove cases where the reviews were NOT written in English.
remove_short_reviews([word_count_threshc ...])	Remove cases where the reviews were too short to provide adequate or useful information.
remove_unverified_reviews([refresh, verbose])	Remove cases where the reviews were not verified.
specify_sql_query(table_name[, before_date, ...])	Specify SQL statement for querying data.
view_stats_on_products([data, by, ...])	Make a bar chart of descriptive statistics on the products (and brands).
view_stats_on_ratings([data, by, ...])	Create a bar chart of descriptive statistics on customers' ratings (and proportions of reviews).

_Reviews.cdd

classmethod _Reviews.**cdd**(**subdir, mkdir=False, **kwargs*)

Get the full pathname of a directory (or file) under the default data directory.

Parameters

- **subdir** (*str*) – name of directory or names of directories (and/or a filename)
- **mkdir** (*bool*) – Whether to create a directory. Defaults to `False`.
- **kwargs** – [Optional] parameters of the function pyhelpers.dir.cd

Return path

full pathname of a directory (or a file) under `"data\"`

Return type

str | pathlib.Path

Examples:

```
>>> from src.processor import *
>>> import os
>>> rvc = RoboticVacuumCleaners(load_preprocd_data=False)
>>> os.path.relpath(rvc.cdd())
'data\amazon_reviews\vacuum_cleaners\robotic'
>>> tvc = TraditionalVacuumCleaners(load_preprocd_data=False)
>>> os.path.relpath(tvc.cdd())
'data\amazon_reviews\vacuum_cleaners\traditional'
>>> smt = SmartThermostats(load_preprocd_data=False)
>>> os.path.relpath(smt.cdd())
'data\amazon_reviews\thermostats\smart'
```

_Reviews.convert_to_integer

_Reviews.**convert_to_integer**(*column_names, int_type=<class 'numpy.uint8'>, refresh=False*)

Convert float values to integers.

Parameters

- **column_names** (*str | list*) – Name of a column or list of column names to convert.
- **int_type** (*type*) – Specific integer type to cast the values. Defaults to `np.uint8`.
- **refresh** (*bool*) – Whether to apply the conversion on the raw or preparatory data and update the preprocessed data. Defaults to `False`.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_raw_data=True, load_preprocd_data=False)
>>> rating_col_name = 'Rating'
>>> rvc.raw_data[rating_col_name] = rvc.raw_data[rating_col_name].astype(float)
>>> rvc.raw_data[rating_col_name].head()
0    2.0
```

(continues on next page)

(continued from previous page)

```
1    3.0
2    1.0
3    1.0
4    2.0
Name: Rating, dtype: float64
>>> rvc.convert_to_integer(column_names=rating_col_name)
>>> rvc.prep_data['rating'].head()
0    2
1    3
2    1
3    1
4    2
Name: rating, dtype: uint8
```

_Reviews.correct_identified_typos

classmethod _Reviews.**correct_identified_typos**(*review_text*, *processes=None*)

Correct typos that have been identified.

Parameters

- **review_text** (*pandas.Series | list*) – textual data of product reviews
- **processes** (*int*) – number of worker processes to use by multiprocessing.Pool(), defaults to `None`

Returns

textual data of which identified typos are corrected

Return type

pandas.Series

Examples:

```
>>> from src.processor import RoboticVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_raw_data=True, load_preprocd_data=False)
>>> sample = rvc.raw_data['ReviewText'].sample(random_state=3)
>>> sample
63709    Great investment, must have got every carpet h...
Name: ReviewText, dtype: object
>>> sample_ = rvc.correct_identified_typos(sample)
>>> sample_
63709    Great investment, must have got every carpet h...
dtype: object
>>> sample.equals(sample_)
False
```

_Reviews.determine_sentiment

_Reviews.**determine_sentiment**(*dual_scale=False, review_column_name=None, refresh=False, verbose=False*)

Determine the sentiment of each product review.

Parameters

- **dual_scale** (*bool*) – Whether to consider both rating and VADER sentiment score to determine sentiment. Defaults to `False`.
- **review_column_name** (*str*) – name of the column that contains review text, when `review_column_name=None`, it defaults to `REVIEW_COLUMN_NAME`
- **refresh** (*bool*) – Whether to perform this function on the raw or preparatory data, and update the preprocessed data. Defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information in console. Defaults to `False`.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners
```

Reviews on the robot vacuum cleaners:

```
>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> rvc.determine_sentiment(review_column_name='ReviewText', verbose=True)
Determining sentiment on rating ... Done.
Calculating VADER sentiment scores ... Done.
Determining sentiment on VADER sentiment score ... Done.
>>> rvc.preprocd_data[['sentiment_on_rating', 'sentiment_on_vs_score']].head()
  sentiment_on_rating sentiment_on_vs_score
0            negative              negative
1            positive              positive
2            positive              positive
3            positive              positive
4            positive              positive
>>> rvc.preprocd_data.shape
(143217, 18)
>>> rvc.preprocd_data_ is None
True
>>> rvc.determine_sentiment(
...     dual_scale=True, review_column_name='ReviewText', verbose=True)
Sentiment on rating is available.
Sentiment on VADER sentiment score is available.
Determining sentiment on both rating and VADER sentiment score ... Done.
>>> rvc.preprocd_data_.shape
(107707, 17)
>>> (rvc.preprocd_data_.sentiment_on_dual_scale == 'positive').sum()
91215
>>> (rvc.preprocd_data_.sentiment_on_dual_scale == 'negative').sum()
15652
>>> (rvc.preprocd_data_.sentiment_on_dual_scale == 'neutral').sum()
840
```

Reviews on the traditional vacuum cleaners:

```
>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> tvc.determine_sentiment(review_column_name='ReviewText', verbose=True)
Determining sentiment on rating ... Done.
Calculating VADER sentiment scores ... Done.
Determining sentiment on VADER sentiment score ... Done.
>>> tvc.preprocd_data[['sentiment_on_rating', 'sentiment_on_vs_score']].head()
  sentiment_on_rating sentiment_on_vs_score
0            positive              positive
1            positive              positive
2            negative              negative
3            positive              positive
4            positive              positive
>>> tvc.preprocd_data.shape
(230479, 18)
>>> tvc.preprocd_data_ is None
True
>>> tvc.determine_sentiment(dual_scale=True, review_column_name='ReviewText',
...                         verbose=True)
Sentiment on rating is available.
Sentiment on VADER sentiment score is available.
Determining sentiment on both rating and VADER sentiment score ... Done.
>>> tvc.preprocd_data_.shape
(175153, 17)
>>> (tvc.preprocd_data_.sentiment_on_dual_scale == 'positive').sum()
144158
>>> (tvc.preprocd_data_.sentiment_on_dual_scale == 'negative').sum()
29422
>>> (tvc.preprocd_data_.sentiment_on_dual_scale == 'neutral').sum()
1573
```

Reviews on the smart thermostats:

```
>>> from src.processor import SmartThermostats
>>> smt = SmartThermostats(load_prep_data=True, load_preprocd_data=False)
>>> smt.determine_sentiment(review_column_name='ReviewText', verbose=True)
Determining sentiment on rating ... Done.
Calculating VADER sentiment scores ... Done.
Determining sentiment on VADER sentiment score ... Done.
>>> smt.preprocd_data[['sentiment_on_rating', 'sentiment_on_vs_score']].head()
  sentiment_on_rating sentiment_on_vs_score
0            positive              positive
1            negative              negative
2            negative              positive
3            negative              positive
4            positive              positive
>>> smt.preprocd_data.shape
(50835, 18)
>>> smt.preprocd_data_ is None
True
>>> smt.determine_sentiment(dual_scale=True, review_column_name='ReviewText',
...                         verbose=True)
Sentiment on rating is available.
Sentiment on VADER sentiment score is available.
Determining sentiment on both rating and VADER sentiment score ... Done.
>>> smt.preprocd_data_.shape
(39709, 17)
>>> (smt.preprocd_data_.sentiment_on_dual_scale == 'positive').sum()
```

(continues on next page)

(continued from previous page)

```
34569
>>> (smt.preprocd_data_.sentiment_on_dual_scale == 'negative').sum()
4880
>>> (smt.preprocd_data_.sentiment_on_dual_scale == 'neutral').sum()
260
```

_Reviews.get_descriptive_stats

_Reviews.**get_descriptive_stats**(*data=None, by='year', before_date='2022-01-01', rating_scores=None, as_percentage=True*)

Get some descriptive statistics.

Parameters

- **data** (*pandas.DataFrame*) – data of the product reviews
- **by** (*str*) – name of a label by which the descriptive statistics is calculated
- **before_date** (*datetime.datetime | pandas.Timestamp | str | None*) – date before which the data will be considered. Defaults to None.
- **rating_scores** (*int | float | list | None*) – rating scores that are under investigation. Defaults to None.
- **as_percentage** (*bool*) – Whether to return percentages (instead of counts); defaults to True.

Returns

data of descriptive statistics (proportions)

Return type

pandas.DataFrame

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> # len(rvc.prep_data.Brand.str.lower().unique())  # 81
>>> # len(rvc.prep_data.ProductTitle.str.lower().unique())  # 284
>>> # len(rvc.prep_data)  # 143217
>>> rvc.get_descriptive_stats(by='year')
                Brand   Product    Review  ...   Neutral  Rating=4*  HighestRating
review_date                                ...
2013         0.012346  0.004149  0.000503  ...  0.111111   0.236111       0.527778
2014         0.012346  0.008299  0.002395  ...  0.075802   0.198251       0.553936
2015         0.037037  0.020747  0.003282  ...  0.059574   0.136170       0.638298
2016         0.049383  0.037344  0.008868  ...  0.068504   0.175591       0.617323
2017         0.086420  0.062241  0.019111  ...  0.070150   0.150895       0.608696
2018         0.111111  0.099585  0.053017  ...  0.072962   0.148294       0.598578
2019         0.234568  0.319502  0.121438  ...  0.068537   0.142537       0.615628
2020         0.493827  0.560166  0.323223  ...  0.066104   0.139811       0.628718
2021         0.876543  0.879668  0.384542  ...  0.074447   0.122147       0.579268
2022         0.876543  0.863071  0.083621  ...  0.089178   0.121576       0.493821
[10 rows x 8 columns]
>>> rvc.get_descriptive_stats(by='month')
                Brand   Product    Review  ...   Neutral  Rating=4*  HighestRating
```

(continues on next page)

(continued from previous page)

```
review_date                                    ...
1            0.753086  0.771784  0.126368  ...  0.078683   0.144988       0.586418
2            0.802469  0.838174  0.088746  ...  0.079701   0.134461       0.551613
3            0.827160  0.842324  0.088614  ...  0.074541   0.124813       0.575762
4            0.876543  0.817427  0.070690  ...  0.068056   0.127222       0.598084
5            0.530864  0.609959  0.068079  ...  0.065231   0.133333       0.623692
6            0.666667  0.684647  0.077135  ...  0.060016   0.127274       0.641803
7            0.691358  0.721992  0.087301  ...  0.067104   0.132528       0.636247
8            0.691358  0.717842  0.073630  ...  0.067520   0.130109       0.614320
9            0.703704  0.734440  0.067017  ...  0.066472   0.130756       0.610440
10           0.728395  0.759336  0.070020  ...  0.074791   0.134324       0.587455
11           0.765432  0.763485  0.072945  ...  0.077534   0.125586       0.559969
12           0.777778  0.804979  0.109456  ...  0.076231   0.139768       0.571574
[12 rows x 8 columns]
>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> # len(tvc.prep_data.Brand.str.lower().unique())  # 65
>>> # len(tvc.prep_data.ProductTitle.str.lower().unique())  # 205
>>> # len(tvc.prep_data)  # 230479
>>> tvc.get_descriptive_stats(by='year')
                Brand   Product    Review  ...   Neutral  Rating=4*  HighestRating
review_date                                ...
2010         0.015152  0.004926  0.000004  ...  0.000000   0.000000       0.000000
2011         0.030303  0.009852  0.000165  ...  0.000000   0.236842       0.605263
2012         0.030303  0.009852  0.000390  ...  0.055556   0.211111       0.511111
2013         0.075758  0.029557  0.000798  ...  0.076087   0.277174       0.510870
2014         0.090909  0.044335  0.003297  ...  0.057895   0.197368       0.635526
2015         0.121212  0.064039  0.011342  ...  0.059679   0.151109       0.694338
2016         0.136364  0.093596  0.021251  ...  0.065741   0.152103       0.654757
2017         0.196970  0.152709  0.044507  ...  0.069799   0.135796       0.592221
2018         0.212121  0.216749  0.070275  ...  0.071248   0.120948       0.594616
2019         0.287879  0.374384  0.116136  ...  0.061979   0.122053       0.617327
2020         0.439394  0.556650  0.180875  ...  0.067310   0.116940       0.609984
2021         0.696970  0.793103  0.308939  ...  0.072145   0.115878       0.580066
2022         1.000000  1.000000  0.242018  ...  0.081821   0.106508       0.523127
[13 rows x 8 columns]
>>> tvc.get_descriptive_stats(by='month')
                Brand   Product    Review  ...   Neutral  Rating=4*  HighestRating
review_date                                ...
1            0.757576  0.807882  0.093106  ...  0.071858   0.121161       0.600261
2            0.787879  0.822660  0.079135  ...  0.074730   0.116344       0.574812
3            0.863636  0.881773  0.088984  ...  0.072456   0.117558       0.574772
4            0.984848  0.940887  0.085045  ...  0.069639   0.114739       0.576297
5            0.969697  0.950739  0.087257  ...  0.072199   0.111829       0.583163
6            1.000000  0.975369  0.083214  ...  0.071224   0.112415       0.586318
7            0.969697  0.931034  0.094946  ...  0.072431   0.117169       0.583512
8            0.984848  0.950739  0.095991  ...  0.071913   0.117565       0.572546
9            0.954545  0.916256  0.082585  ...  0.077178   0.113324       0.567563
10           0.727273  0.798030  0.060786  ...  0.069094   0.123697       0.581656
11           0.621212  0.724138  0.063620  ...  0.069836   0.122349       0.572052
12           0.696970  0.783251  0.085331  ...  0.069456   0.122998       0.590431
[12 rows x 8 columns]
```

_Reviews.get_ratings_stats

classmethod `_Reviews.get_ratings_stats`(*data, group_label, rating_scores=None, as_percentage=True*)

Calculate proportions of different ratings by year or month.

Parameters

- **data** (*pandas.DataFrame*) – data of the product reviews
- **group_label** (*pandas.Series*) – labels used for grouping the data of ratings
- **rating_scores** (*int | float | list | None*) – rating scores that are under investigation. Defaults to `None`.
- **as_percentage** (*bool*) – Whether to return percentages (instead of counts); defaults to `True`.

Returns

proportions of different ratings by year or month (depending on `group_label`)

Return type

pandas.DataFrame

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> group_label_yearly = rvc.prep_data.review_date.dt.year
>>> rvc.get_ratings_stats(rvc.prep_data, group_label_yearly, rating_scores=[5, 1])
             HighestRating  LowestRating
review_date
2013              0.527778      0.069444
2014              0.553936      0.128280
2015              0.638298      0.110638
2016              0.617323      0.077165
2017              0.608696      0.094629
2018              0.598578      0.109311
2019              0.615628      0.112293
2020              0.628718      0.108077
2021              0.579268      0.154740
2022              0.493821      0.210421
>>> group_label_monthly = rvc.prep_data.review_date.dt.month
>>> rvc.get_ratings_stats(rvc.prep_data, group_label_monthly, rating_scores=[1, 2])
             LowestRating  Rating=2*
review_date
1                0.124655   0.065256
2                0.161133   0.073092
3                0.153101   0.071783
4                0.139471   0.067167
5                0.115385   0.062359
6                0.113606   0.057301
7                0.109494   0.054627
8                0.128023   0.060028
9                0.131382   0.060950
10               0.135221   0.068209
```

(continues on next page)

(continued from previous page)

```
11               0.159184   0.077726
12               0.143404   0.069023
>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> group_label_yearly = tvc.prep_data.review_date.dt.year
>>> tvc.get_ratings_stats(tvc.prep_data, group_label_yearly, rating_scores=[5, 1])
             HighestRating  LowestRating
review_date
2010              0.000000      1.000000
2011              0.605263      0.131579
2012              0.511111      0.133333
2013              0.510870      0.103261
2014              0.635526      0.068421
2015              0.694338      0.061591
2016              0.654757      0.083708
2017              0.592221      0.136966
2018              0.594616      0.148855
2019              0.617327      0.143236
2020              0.609984      0.146109
2021              0.580066      0.166213
2022              0.523127      0.207099
>>> group_label_monthly = tvc.prep_data.review_date.dt.month
>>> tvc.get_ratings_stats(tvc.prep_data, group_label_monthly, rating_scores=[1, 2])
             LowestRating  Rating=2*
review_date
1                0.144322   0.062398
2                0.167498   0.066615
3                0.168365   0.066849
4                0.173359   0.065966
5                0.166426   0.066382
6                0.165598   0.064445
7                0.160810   0.066079
8                0.170584   0.067393
9                0.172218   0.069717
10               0.162598   0.062955
11               0.164632   0.071131
12               0.152489   0.064626
```

_Reviews.get_vader_sentiment_score

`_Reviews.`**`get_vader_sentiment_score`**(*review_column_name=None*, *processes=None*, *refresh=False*, *verbose=False*)

Add calculated VADER sentiment score to the preprocessed data.

Parameters

- **`review_column_name`** (*`str`*) – name of the column that contains preprocessed review text, when `review_column_name=None`, it defaults to `ORIGINAL_REVIEW_COLUMN_NAME`
- **`processes`** (*`int`*) – number of worker processes to use by multiprocessing.Pool(), defaults to `None`
- **`refresh`** (*`bool`*) – Whether to perform this function on the raw or preparatory data, and update the preprocessed data. Defaults to `False`.
- **`verbose`** (*`bool | int`*) – Whether to print relevant information in console. Defaults to `False`.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners

>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> rvc.preprocd_data is None
True
>>> rvc.get_vader_sentiment_score(review_column_name='ReviewText', verbose=True)
Calculating VADER sentiment scores ... Done.
>>> len(rvc.preprocd_data)
143217
>>> score_cols = ['vs_neg_score', 'vs_neu_score', 'vs_pos_score', 'vs_compound_score']
>>> rvc.preprocd_data[score_cols].head()
   vs_neg_score  vs_neu_score  vs_pos_score  vs_compound_score
0         0.174         0.723         0.103            -0.4359
1         0.000         0.630         0.370             0.9781
2         0.000         0.624         0.376             0.8878
3         0.019         0.699         0.283             0.9591
4         0.093         0.711         0.196             0.6862

>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> tvc.preprocd_data is None
True
>>> tvc.get_vader_sentiment_score(review_column_name='ReviewText', verbose=True)
Calculating VADER sentiment scores ... Done.
>>> len(tvc.preprocd_data)
230479
>>> score_cols = ['vs_neg_score', 'vs_neu_score', 'vs_pos_score', 'vs_compound_score']
>>> tvc.preprocd_data[score_cols].head()
   vs_neg_score  vs_neu_score  vs_pos_score  vs_compound_score
0         0.063         0.790         0.147             0.7389
1         0.093         0.713         0.194             0.9002
2         0.156         0.686         0.158            -0.1923
3         0.000         0.664         0.336             0.9647
4         0.045         0.823         0.132             0.8805
```

_Reviews.if_is_verified_note

_Reviews.**if_is_verified_note**()

Returns a note message indicating whether the data is considered verified only.

Returns
A note message indicating whether the data is verified only.

Return type
str

Examples:

```
>>> from src.processor import RoboticVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_preprocd_data=False)
>>> rvc.verified_reviews_only
False
>>> rvc.if_is_verified_note()
''
```

_Reviews.load_prep_data

_Reviews.**load_prep_data**(*before_date=None, verified_reviews_only=False, update=False, verbose=False, ret_data=False, **kwargs*)

Load the preparatory version of the product reviews data.

Parameters

- **before_date** (*str | None*) – date before which the preparatory data is considered, e.g. '2021-04-01' and '2022-04-01'. Defaults to `None`.
- **verified_reviews_only** (*bool*) – consider only the verified reviews. Defaults to `False`.
- **update** (*bool | int*) – Whether to reprocess the original data file(s). Defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information in console. Defaults to `False`.
- **ret_data** (*bool*) – Whether to return the raw data that is read/loaded. Defaults to `False`.
- **kwargs** – [Optional] parameters of the method pyhelpers.dbms.PostgreSQL.read_sql_query.

Returns

Preparatory data of product reviews

Return type

pandas.DataFrame | None

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_preprocd_data=False)
>>> # rvc.load_prep_data(update=True, verbose=True)  # Update prep_data
>>> rvc.load_prep_data(verbose=True)
>>> rvc.prep_data.shape
(143217, 12)
>>> tvc = TraditionalVacuumCleaners(load_preprocd_data=False)
>>> # tvc.load_prep_data(update=True, verbose=True)  # Update prep_data
>>> tvc.load_prep_data(verbose=True)
>>> tvc.prep_data.shape
(230479, 12)
```

_Reviews.load_preprocd_data

_Reviews.**load_preprocd_data**(*verified_reviews_only=False, word_count_threshold=20, dual_scale=False, before_date=None, update=False, verbose=False, ret_data=False*)

Read the preprocessed product reviews.

Parameters

- **verified_reviews_only** (*bool*) – consider only the verified reviews. Defaults to `False`.

- **word_count_threshold** (*int*) – word count in a review, beyond which the review is not considered for further analysis. Defaults to `20`.
- **dual_scale** (*bool*) – indicate whether the sentiment is determined on both rating and VADER sentiment score. Defaults to `False`.
- **before_date** (*str | None*) – date before which the preparatory data is considered, e.g. `'2021-04-01'` and `'2022-04-01'`. Defaults to `None`.
- **update** (*bool | int*) – Whether to reprocess the data. Defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information in console. Defaults to `False`.
- **ret_data** (*bool*) – Whether to return the preprocessed data. Defaults to `False`.

Returns

preprocessed data of the product reviews

Return type

pandas.DataFrame

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners

>>> rvc = RoboticVacuumCleaners(load_preprocd_data=False)
>>> # rvc.load_preprocd_data(update=True, verbose=True)  # Update preprocd_data
>>> rvc.load_preprocd_data(verified_reviews_only=False, verbose=True)
>>> rvc.preprocd_data.shape
(101608, 19)
>>> rvc.preprocd_data_ is None
True
>>> rvc.load_preprocd_data(verified_reviews_only=True, verbose=True)
>>> rvc.preprocd_data.shape
(89989, 19)
>>> rvc.preprocd_data_ is None
True
>>> rvc.load_preprocd_data(dual_scale=True, verbose=True)
>>> rvc.preprocd_data.shape  # dual_scale=True
(77775, 18)
>>> (rvc.preprocd_data.sentiment_on_dual_scale == 'positive').sum()
64836
>>> (rvc.preprocd_data.sentiment_on_dual_scale == 'negative').sum()
12587
>>> rvc.preprocd_data_.shape  # dual_scale=False
(101608, 19)

>>> tvc = TraditionalVacuumCleaners(load_preprocd_data=False)
>>> # tvc.load_preprocd_data(update=True, verbose=True)  # Update preprocd_data
>>> tvc.load_preprocd_data(verified_reviews_only=False, verbose=True)
>>> tvc.preprocd_data.shape
(146656, 19)
>>> tvc.preprocd_data_ is None
True
>>> tvc.load_preprocd_data(verified_reviews_only=True, verbose=True)
>>> tvc.preprocd_data.shape
```

(continues on next page)

(continued from previous page)

```
(131998, 19)
>>> tvc.preprocd_data_ is None
True
>>> tvc.load_preprocd_data(dual_scale=True, verbose=True)
>>> tvc.preprocd_data.shape  # dual_scale=True
(110978, 18)
>>> (tvc.preprocd_data.sentiment_on_dual_scale == 'positive').sum()
89384
>>> (tvc.preprocd_data.sentiment_on_dual_scale == 'negative').sum()
21109
>>> tvc.preprocd_data_.shape  # dual_scale=False
(146656, 19)

>>> from src.processor import SmartThermostats
>>> smt = SmartThermostats(load_preprocd_data=False)
>>> # smt.load_preprocd_data(update=True, verbose=True)  # Update preprocd_data
>>> smt.load_preprocd_data(verified_reviews_only=False, verbose=True)
>>> smt.preprocd_data.shape
(46317, 19)
>>> smt.preprocd_data_ is None
True
>>> smt.load_preprocd_data(verified_reviews_only=True, verbose=True)
>>> smt.preprocd_data.shape
(38810, 19)
>>> smt.preprocd_data_ is None
True
>>> smt.load_preprocd_data(dual_scale=True, verbose=True)
>>> smt.preprocd_data.shape  # dual_scale=True
(36254, 18)
>>> (smt.preprocd_data.sentiment_on_dual_scale == 'positive').sum()
34569
>>> (smt.preprocd_data.sentiment_on_dual_scale == 'negative').sum()
4880
>>> smt.preprocd_data_.shape  # dual_scale=False
(46317, 19)
```

_Reviews.load_raw_data

`_Reviews.`**`load_raw_data`**(*index_columns=None, verified_reviews_only=False, update=False, verbose=False, ret_data=False, **kwargs*)

Reads the original version (raw data) of product reviews.

Parameters

- **`index_columns`** (*`str | list | None`*) – Name(s) of column(s) to set as the index; defaults to `['Brand', 'ASIN', 'ParentID']` if not specified.
- **`verified_reviews_only`** (*`bool`*) – Whether to consider only verified reviews; defaults to `False`.
- **`update`** (*`bool | int`*) – Whether to reprocess the original data file(s). Defaults to `False`.
- **`verbose`** (*`bool | int`*) – Whether to print relevant information in the console. Defaults to `False`.
- **`ret_data`** (*`bool`*) – Whether to return the raw data that is read/loaded.

Defaults to False.

- **kwargs** (*dict*) – [Optional] parameters for the method *pyhelpers.dbms.PostgreSQL.read_sql_query*.

Returns
Original version (raw data) of the product reviews.

Return type
pandas.DataFrame | None

Examples:

```
>>> from src.processor import RoboticVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_preprocd_data=False)
>>> rvc.load_raw_data(verbose=True)
>>> rvc.raw_data.shape
(143217, 13)
```

_Reviews.make_prep_data

_Reviews.**make_prep_data**(*ret_prep_data=False*, *verbose=False*)
Make preparatory data from the raw data.

Parameters

- **ret_prep_data** (*bool*) – Whether to return the preparatory data. Defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information in console. Defaults to False.

Returns
the preparatory data (when ret_prep_data=True)

Return type
pandas.DataFrame

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_raw_data=True, load_preprocd_data=False)
>>> rvc.prep_data is None
True
>>> rvc.make_prep_data(verbose=True)
>>> rvc.prep_data.head()
         ASIN   Brand  ... review_date review_location
0  B085D45SZF  iRobot  ...  2021-07-12   United States
1  B08TN2GC94  iRobot  ...  2021-07-07   United States
2  B085D45SZF  iRobot  ...  2021-07-11   United States
3  B085D45SZF  iRobot  ...  2021-07-09   United States
4  B085D45SZF  iRobot  ...  2021-07-12   United States
[5 rows x 12 columns]
>>> rvc.prep_data.shape
(143217, 12)
>>> tvc = TraditionalVacuumCleaners(load_raw_data=True, load_preprocd_data=False)
>>> tvc.prep_data is None
True
```

(continues on next page)

(continued from previous page)

```
>>> tvc.make_prep_data(verbose=True)
>>> tvc.prep_data.head()
         ASIN   Brand  ... review_date review_location
0  B07SMJJTL7  EUREKA  ...  2022-09-19   United States
1  B07SMJJTL7  EUREKA  ...  2022-09-19   United States
2  B07SMJJTL7  EUREKA  ...  2022-09-19   United States
3  B07SMJJTL7  EUREKA  ...  2022-09-19   United States
4  B07SMJJTL7  EUREKA  ...  2022-09-18   United States
[5 rows x 12 columns]
>>> tvc.prep_data.shape
(230479, 12)
```

_Reviews.parse_review_date

_Reviews.**parse_review_date**(*column_name='ReviewDate'*, *parsed_column_name=None*, *refresh=False*)

Parse the information about dates for each record of the product reviews.

Parameters

- **column_name** (*str*) – Name of the column that contains information about review dates; defaults to `'ReviewDate'`.
- **parsed_column_name** (*list | None*) – New column names for parsed date data, in cases where original records contain both date and location information; defaults to `None`.
- **refresh** (*bool*) – Whether to perform this function on the raw data. Defaults to `False`.

Note: Newly created column names are set to lowercase by default.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_raw_data=True, load_preprocd_data=False)
>>> rvc.raw_data['ReviewDate'].head()
0     July 12, 2021
1      July 7, 2021
2     July 11, 2021
3      July 9, 2021
4     July 12, 2021
Name: ReviewDate, dtype: object
>>> rvc.parse_review_date()  # Transform the review date data
>>> new_column_names = rvc.column_name_changes['ReviewDate']
>>> new_column_names
['review_date', 'review_location']
>>> rvc.prep_data[new_column_names].head()
  review_date review_location
0  2021-07-12   United States
1  2021-07-07   United States
2  2021-07-11   United States
3  2021-07-09   United States
4  2021-07-12   United States
```

_Reviews.preprocess_prep_data

_Reviews.**preprocess_prep_data**(*verified_reviews_only=False*, *word_count_threshold=20*, *dual_scale=False*, *refresh=False*, *verbose=False*, ***kwargs*)

Preprocess the preparatory data.

Parameters

- **verified_reviews_only** (*bool*) – consider only the verified reviews. Defaults to `False`.
- **word_count_threshold** (*int*) – word count in a review, beyond which the review is not considered for further analysis. Defaults to `20`
- **dual_scale** (*bool*) – indicate whether the sentiment is determined on both rating and VADER sentiment score. Defaults to `False`.
- **refresh** (*bool*) – Whether to perform this function on the raw data or preparatory data, and update the preprocessed data. Defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information in console. Defaults to `False`.
- **kwargs** – [Optional] parameters of the method `preprocess_review_text()`

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners

>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> len(rvc.prep_data)
143217
>>> rvc.preprocd_data is None
True
>>> rvc.preprocess_prep_data(verbose=True)
Preprocessing the preparatory data ...
    Removing short reviews (No. of words < 20) ... Done.
    Removing non-English reviews ... Done.
    Determining sentiment on rating ... Done.
    Calculating VADER sentiment scores ... Done.
    Determining sentiment on VADER sentiment score ... Done.
    Processing review text ...
        Removing stopwords ... Done.
        Removing punctuation ... Done.
        Removing single letters ... Done.
        Removing digits ... Done.
        Lemmatizing texts ... Done.
Done.
>>> len(rvc.preprocd_data)
101608

>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> len(tvc.prep_data)
230479
>>> tvc.preprocd_data is None
True
>>> tvc.preprocess_prep_data(verbose=True)
```

(continues on next page)

(continued from previous page)

```
Preprocessing the preparatory data ...
    Removing short reviews (No. of words < 20) ... Done.
    Removing non-English reviews ... Done.
    Determining sentiment on rating ... Done.
    Calculating VADER sentiment scores ... Done.
    Determining sentiment on VADER sentiment score ... Done.
    Processing review text ...
        Removing stopwords ... Done.
        Removing punctuation ... Done.
        Removing single letters ... Done.
        Removing digits ... Done.
        Lemmatizing texts ... Done.
Done.
>>> len(tvc.preprocd_data)
146656
```

_Reviews.preprocess_review_text

_Reviews.**preprocess_review_text**(*rm_punctuation=True, rm_stopwords=True, rm_single_letters=True, rm_digits=True, lemmatize_words=True, refresh=False, verbose=False*)

Process review text.

Parameters

- **rm_punctuation** (*bool*) – Whether to remove punctuation. Defaults to True.
- **rm_stopwords** (*bool*) – Whether to remove stopwords. Defaults to True.
- **rm_single_letters** (*bool*) – Whether to remove single letters. Defaults to True.
- **rm_digits** (*bool*) – Whether to remove digits. Defaults to True.
- **lemmatize_words** (*bool*) – Whether to lemmatize the words in review texts; defaults to True.
- **refresh** (*bool*) – Whether to perform this function on the raw or preparatory data, and update the preprocessed data. Defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information in console. Defaults to False.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners

>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> rvc.preprocess_review_text(verbose=True)
Processing review text ...
    Removing stopwords ... Done.
    Removing punctuation ... Done.
    Removing single letters ... Done.
    Removing digits ... Done.
    Lemmatizing texts ... Done.
```

(continues on next page)

(continued from previous page)

```
Done.
>>> len(rvc.preprocd_data)
143217

>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> tvc.preprocess_review_text(verbose=True)
Processing review text ...
    Removing stopwords ... Done.
    Removing punctuation ... Done.
    Removing single letters ... Done.
    Removing digits ... Done.
    Lemmatizing texts ... Done.
Done.
>>> len(tvc.preprocd_data)
230479
```

_Reviews.read_raw_data

`_Reviews.`**`read_raw_data`**(*path_to_file*, *verbose=False*)

Read and preprocess the original product review data.

Parameters

- **`path_to_file`** (*`str | pathlib.Path`*) – Pathname of the raw data file.
- **`verbose`** (*`bool | int`*) – Whether to print relevant information in the console. Defaults to `False`.

Returns

Roughly-preprocessed data of the product reviews.

Return type

pandas.DataFrame | None

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners
>>> from pyhelpers.dirs import cdd
>>> import os
>>> rvc = RoboticVacuumCleaners(load_preprocd_data=False)
>>> temp_path_to_file = rvc._get_backup_temp(idx=0)
>>> raw_dat = rvc.read_raw_data(temp_path_to_file, verbose=3)
Total of records: 92742.
>>> raw_dat.shape
(92742, 13)
>>> os.remove(temp_path_to_file)
```

_Reviews.regulate_people_found_helpful

_Reviews.**regulate_people_found_helpful**(*column_name='PeopleFoundHelpful'*, *refresh=False*)

Regulates the data regarding how many people found reviews helpful.

Parameters

- **column_name** (*str*) – Name of the column that contains information about the number of people who found a review helpful. Defaults to `'PeopleFoundHelpful'`.
- **refresh** (*bool*) – Whether to perform this function on the raw data and update the preparatory data. Defaults to `False`.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_raw_data=True, load_preprocd_data=False)
>>> rvc.raw_data['PeopleFoundHelpful'].head()
0
1    2
2
3
4
Name: PeopleFoundHelpful, dtype: object
>>> rvc.regulate_people_found_helpful()  # Cleanse the data
>>> new_column_name = rvc.column_name_changes['PeopleFoundHelpful']
>>> new_column_name
'people_found_helpful'
>>> rvc.prep_data[new_column_name].head()
0    0
1    2
2    0
3    0
4    0
Name: people_found_helpful, dtype: int64
```

_Reviews.remove_non_english_reviews

_Reviews.**remove_non_english_reviews**(*word_count_threshold=20*, *refresh=False*, *verbose=False*)

Remove cases where the reviews were NOT written in English.

Parameters

- **word_count_threshold** (*int*) – word count in a review, beyond which the review is not considered for further analysis. Defaults to `20`.
- **refresh** (*bool*) – Whether to perform this function on the raw or preparatory data, and update the preprocessed data. Defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information in console. Defaults to `False`.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners

>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> len(rvc.prep_data)
143217
>>> rvc.preprocd_data is None
True
>>> rvc.remove_non_english_reviews(verbose=True)
Removing short reviews (No. of words < 20) ... Done.
Removing non-English reviews ... Done.
>>> len(rvc.preprocd_data)
101608
>>> rvc.remove_non_english_reviews(word_count_threshold=25, verbose=True)
Removing short reviews (No. of words < 25) ... Done.
Removing non-English reviews ... Done.
>>> len(rvc.preprocd_data)
93902
>>> rvc.remove_non_english_reviews(word_count_threshold=50, verbose=True)
Removing short reviews (No. of words < 50) ... Done.
Removing non-English reviews ... Done.
>>> len(rvc.preprocd_data)
63584

>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> len(tvc.prep_data)
230479
>>> tvc.preprocd_data is None
True
>>> tvc.remove_non_english_reviews(verbose=True)
Removing short reviews (No. of words < 20) ... Done.
Removing non-English reviews ... Done.
>>> len(tvc.preprocd_data)
146656
>>> tvc.remove_non_english_reviews(word_count_threshold=25, verbose=True)
Removing short reviews (No. of words < 25) ... Done.
Removing non-English reviews ... Done.
>>> len(tvc.preprocd_data)
130923
>>> tvc.remove_non_english_reviews(word_count_threshold=50, verbose=True)
Removing short reviews (No. of words < 50) ... Done.
Removing non-English reviews ... Done.
>>> len(tvc.preprocd_data)
77196
```

_Reviews.remove_short_reviews

`_Reviews.`**`remove_short_reviews`**(*word_count_threshold=20*, *refresh=False*, *verbose=False*)

Remove cases where the reviews were too short to provide adequate or useful information.

Parameters

- **`word_count_threshold`** (*`int`*) – word count in a review, beyond which the review is not considered for further analysis. Defaults to `20`.
- **`refresh`** (*`bool`*) – Whether to perform this function on the raw or preparatory data, and update the preprocessed data. Defaults to `False`.
- **`verbose`** (*`bool | int`*) – Whether to print relevant information in console.

Defaults to False.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> len(rvc.prep_data)
143217
>>> # Keep the cases where the word count of the review were greater than 20
>>> rvc.remove_short_reviews(verbose=True)
Removing short reviews (No. of words < 20) ... Done.
>>> len(rvc.preprocd_data)
104340
>>> rvc.remove_short_reviews(word_count_threshold=25, verbose=True)
Removing short reviews (No. of words < 25) ... Done.
>>> len(rvc.preprocd_data)
96254
>>> rvc.remove_short_reviews(word_count_threshold=50, verbose=True)
Removing short reviews (No. of words < 50) ... Done.
>>> len(rvc.preprocd_data)
64834
>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> len(tvc.prep_data)
230479
>>> # Keep the cases where the word count of the review were greater than 20
>>> tvc.remove_short_reviews(verbose=True)
Removing short reviews (No. of words < 20) ... Done.
>>> len(tvc.preprocd_data)
148003
>>> tvc.remove_short_reviews(word_count_threshold=25, verbose=True)
Removing short reviews (No. of words < 25) ... Done.
>>> len(tvc.preprocd_data)
131977
>>> tvc.remove_short_reviews(word_count_threshold=50, verbose=True)
Removing short reviews (No. of words < 50) ... Done.
>>> len(tvc.preprocd_data)
77577
```

_Reviews.remove_unverified_reviews

_Reviews.**remove_unverified_reviews**(*refresh=False, verbose=False*)

Remove cases where the reviews were not verified.

Parameters

- **refresh** (*bool*) – Whether to perform this function on the raw or preparatory data, and update the preprocessed data. Defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information in console. Defaults to False.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners, TraditionalVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> len(rvc.prep_data)
143217
>>> rvc.remove_unverified_reviews()
```

(continues on next page)

(continued from previous page)

```
>>> len(rvc.prep_data)  # Remove unverified reviews does not change `rvc.raw_data`
143217
>>> len(rvc.preprocd_data)
129109
>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> len(tvc.prep_data)
230479
>>> tvc.remove_unverified_reviews()
>>> len(tvc.prep_data)
230479
>>> len(tvc.preprocd_data)
213052
```

Note: This method does not make any changes to `.prep_data`.

_Reviews.specify_sql_query

classmethod `_Reviews.`**`specify_sql_query`**(*table_name*, *before_date=None*, *verified_reviews_only=False*)

Specify SQL statement for querying data.

Parameters

- **`table_name`** (*`str`*) – Name of the table to query.
- **`before_date`** (*`str | None`*) – Filter data to include only records before this date (exclusive). Defaults to `None`.
- **`verified_reviews_only`** (*`bool`*) – Whether to include only verified reviews in the query; Defaults to `True`.

Returns

SQL query string.

Return type

str

Examples:

```
>>> from src.processor import RoboticVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_preprocd_data=False)
>>> rvc.specify_sql_query(table_name='<table_name>')
'SELECT * FROM amazon_reviews."<table_name>"'
>>> rvc.specify_sql_query(table_name='<table_name>', before_date='2022-01-01')
'SELECT * FROM amazon_reviews."<table_name>" WHERE "review_date" < '2022-01-01''
>>> rvc.specify_sql_query(table_name='<table_name>', verified_reviews_only=True)
'SELECT * FROM amazon_reviews."<table_name>" WHERE "Verified" IS TRUE'
```

_Reviews.view_stats_on_products

`_Reviews.view_stats_on_products`(*data=None*, *by='year'*, *horizontal=False*, *save_as=None*, *verbose=False*, ***kwargs*)

Make a bar chart of descriptive statistics on the products (and brands).

Parameters

- **data** (*pandas.DataFrame*) – data of the product reviews
- **by** (*str*) – label by which the descriptive statistics is calculated. Defaults to `'year'`
- **horizontal** (*bool*) – Whether to create a horizontal bar chart. Defaults to `False`.
- **save_as** (*str | bool | None*) – extension of figure filename, or whether to save the figure; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information in console; defaults to `False`.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> # rvc.view_stats_on_products(by='year', save_as=".svg", verbose=True)
>>> rvc.view_stats_on_products(by='year')
```

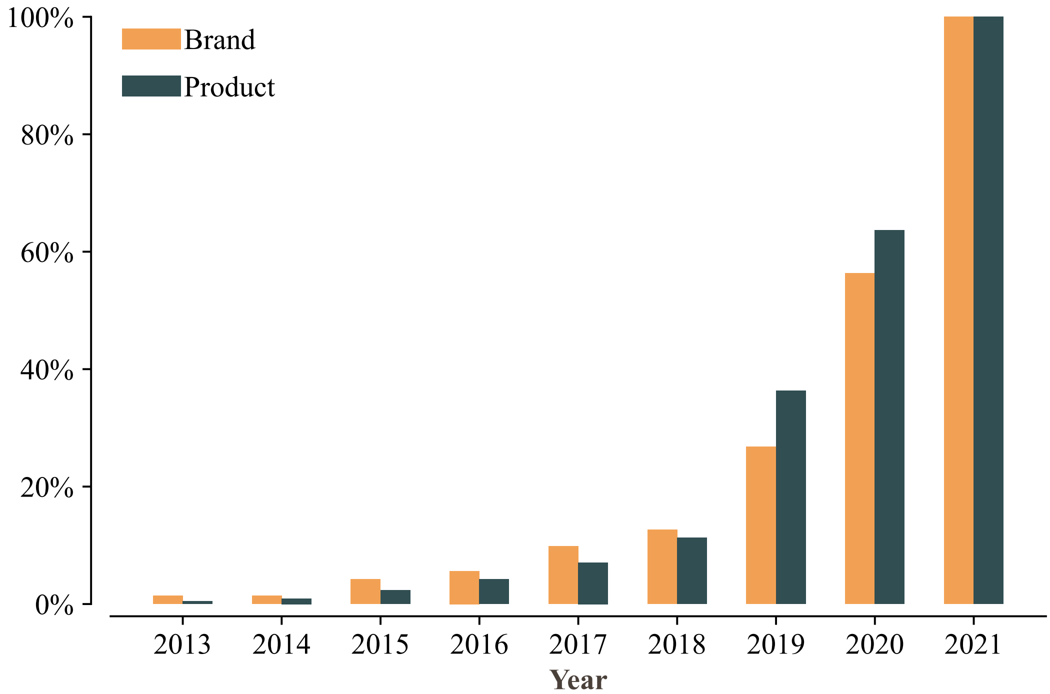

Fig. 2: Descriptive statistics of robot vacuums purchased (on a yearly basis).

```
>>> # rvc.view_stats_on_products(by='month', save_as=".svg", verbose=True)
>>> rvc.view_stats_on_products(by='month')
```

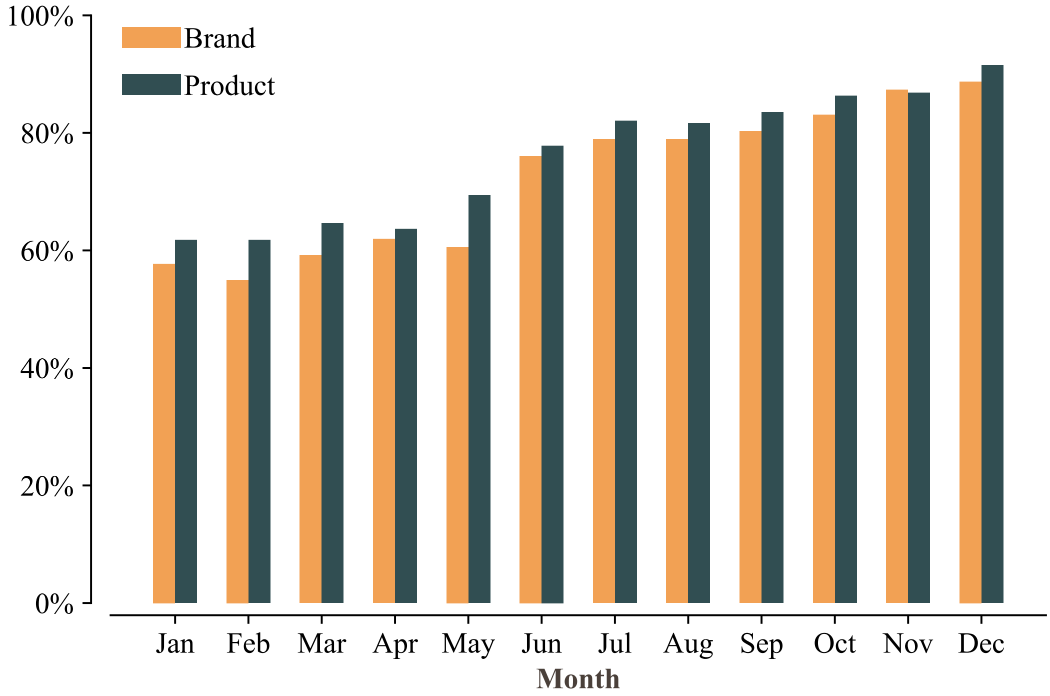

Fig. 3: Descriptive statistics of robot vacuums purchased (on a monthly basis).

```
>>> from src.processor import TraditionalVacuumCleaners
>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
>>> # tvc.view_stats_on_products(by='year', save_as=".svg", verbose=True)
>>> tvc.view_stats_on_products(by='year')
```

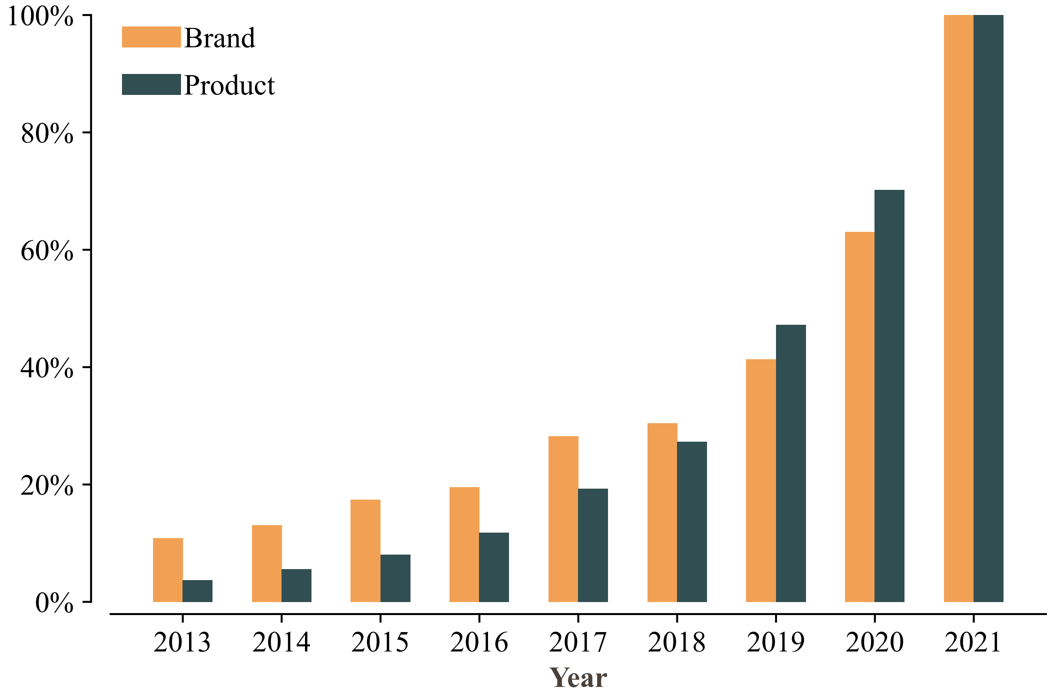

Fig. 4: Descriptive statistics of traditional vacuums purchased (on a yearly basis).

```
>>> # tvc.view_stats_on_products(by='month', save_as=".svg", verbose=True)
>>> tvc.view_stats_on_products(by='month')
```

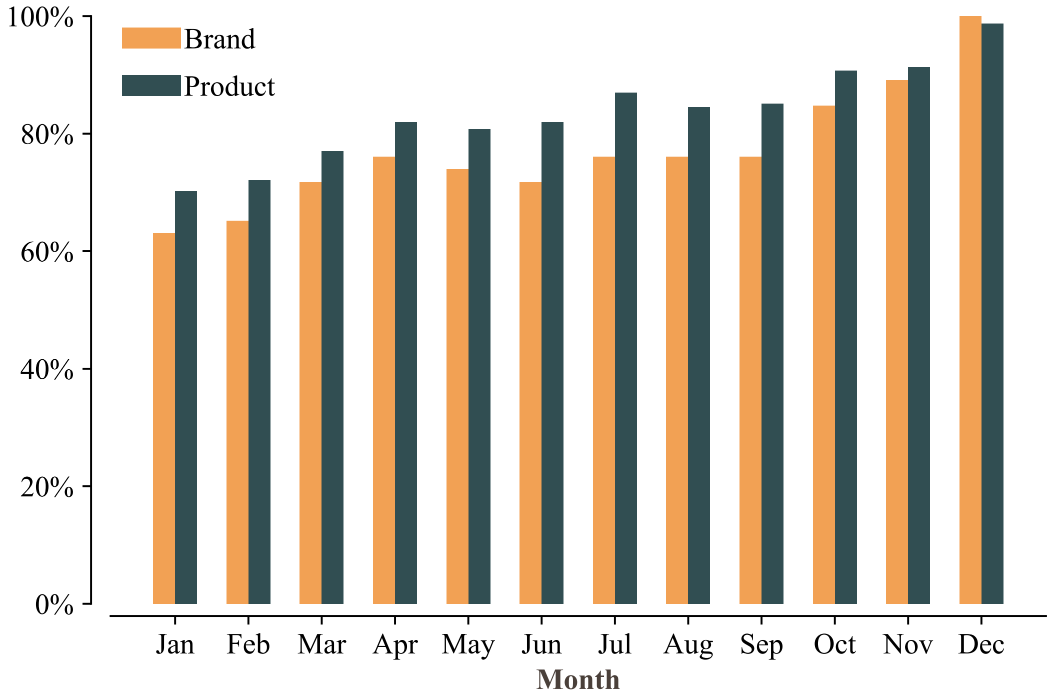

Fig. 5: Descriptive statistics of traditional vacuums purchased (on a monthly basis).

```
>>> from src.processor import SmartThermostats
>>> smt = SmartThermostats(load_prep_data=True, load_preprocd_data=False)
>>> # smt.view_stats_on_products(by='year', save_as=".svg", verbose=True)
>>> smt.view_stats_on_products(by='year')
```

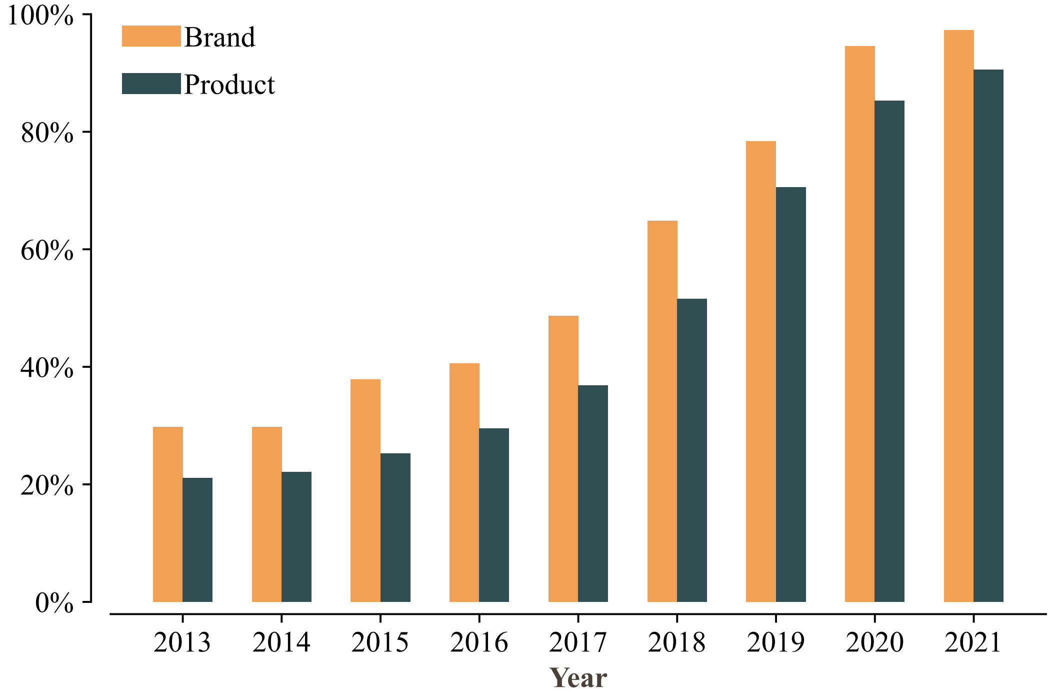

Fig. 6: Descriptive statistics of smart thermostats purchased (on a yearly basis).

```
>>> # smt.view_stats_on_products(by='month', save_as=".svg", verbose=True)
>>> smt.view_stats_on_products(by='month')
```

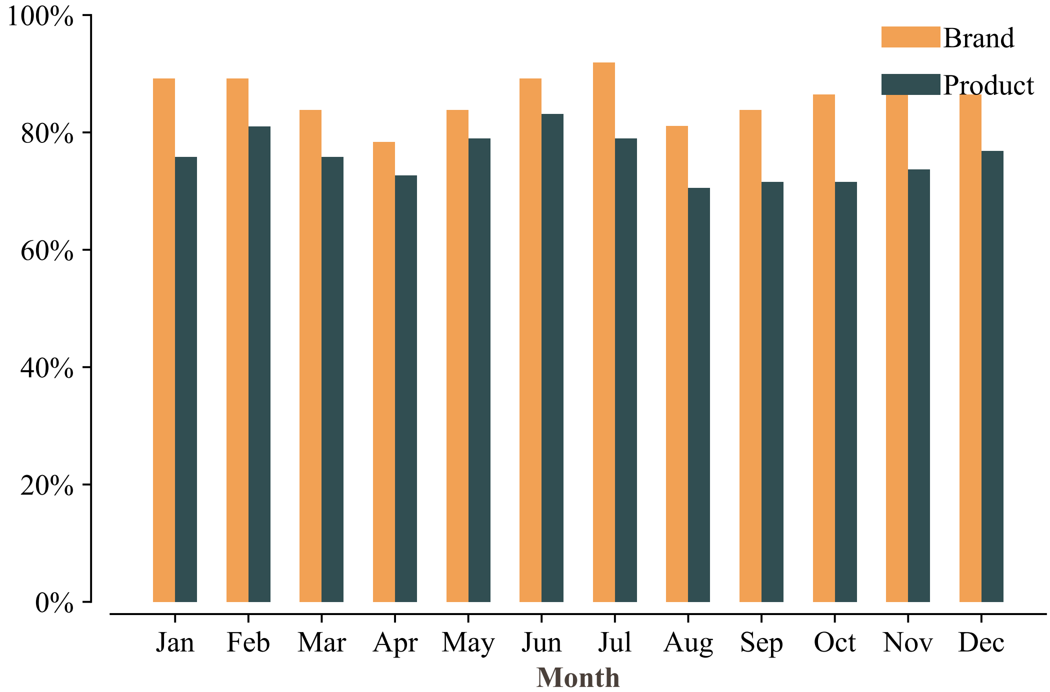

Fig. 7: Descriptive statistics of smart thermostats purchased (on a monthly basis).

_Reviews.view_stats_on_ratings

`_Reviews.`**`view_stats_on_ratings`**(*data=None*, *by='year'*, *review_stats=True*, *horizontal=False*, *save_as=None*, *verbose=False*, ***kwargs*)

Create a bar chart of descriptive statistics on customers' ratings (and proportions of reviews).

Parameters

- **data** (*pandas.DataFrame*) – data of the product reviews
- **by** (*str*) – label by which the descriptive statistics is calculated. Defaults to `'year'`
- **review_stats** (*bool*) – Whether to include the proportions of reviews. Defaults to `True`.
- **horizontal** (*bool*) – Whether to create a horizontal bar chart. Defaults to `False`.
- **save_as** (*str | bool | None*) – extension of figure filename, or whether to save the figure; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information in console; defaults to `False`.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners
>>> rvc = RoboticVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
```

(continues on next page)

(continued from previous page)

```
>>> # rvc.view_stats_on_ratings(by='year', save_as=".svg", verbose=True)
>>> rvc.view_stats_on_ratings(by='year')
```

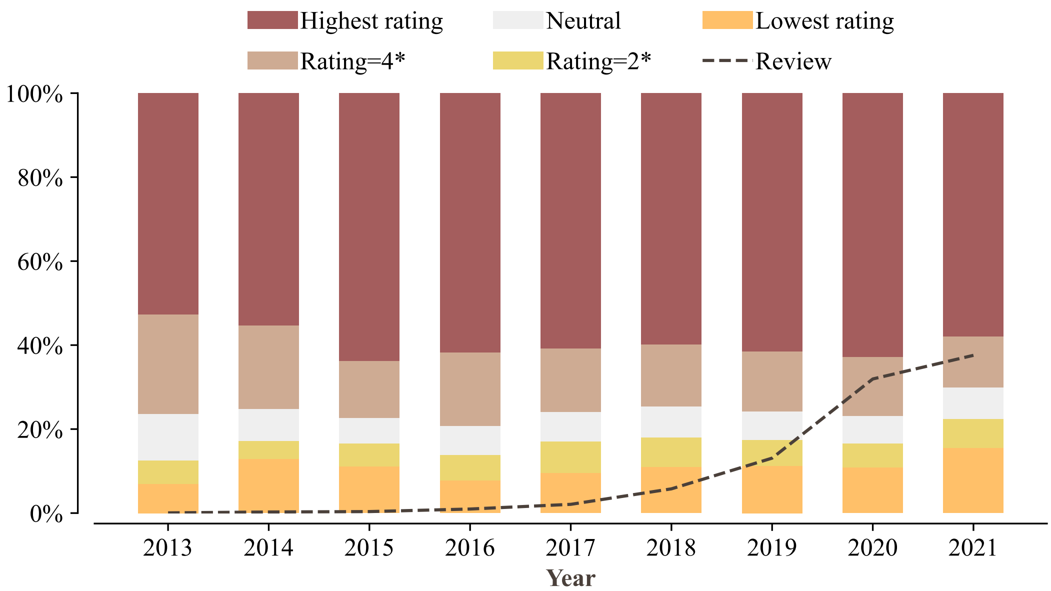

Fig. 8: Customers' ratings on robot vacuums (on a yearly basis).

```
>>> # rvc.view_stats_on_ratings(by='month', save_as=".svg", verbose=True)
>>> rvc.view_stats_on_ratings(by='month')
```

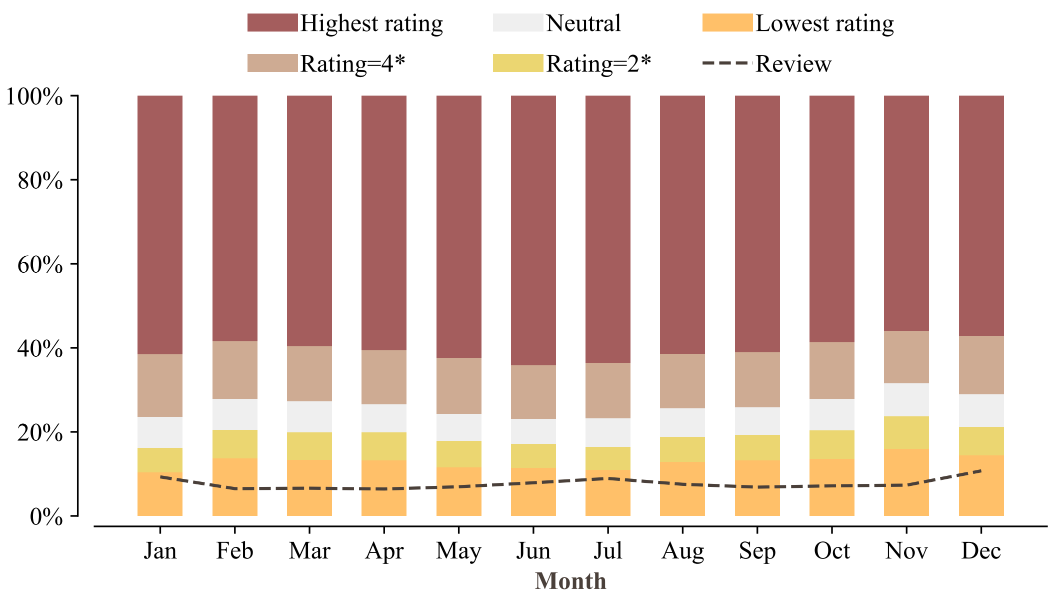

Fig. 9: Customers' ratings on robot vacuums (on a monthly basis).

```
>>> from src.processor import TraditionalVacuumCleaners
>>> tvc = TraditionalVacuumCleaners(load_prep_data=True, load_preprocd_data=False)
```

(continues on next page)

(continued from previous page)

```
>>> # tvc.view_stats_on_ratings(by='year', save_as=".svg", verbose=True)
>>> tvc.view_stats_on_ratings(by='year')
```

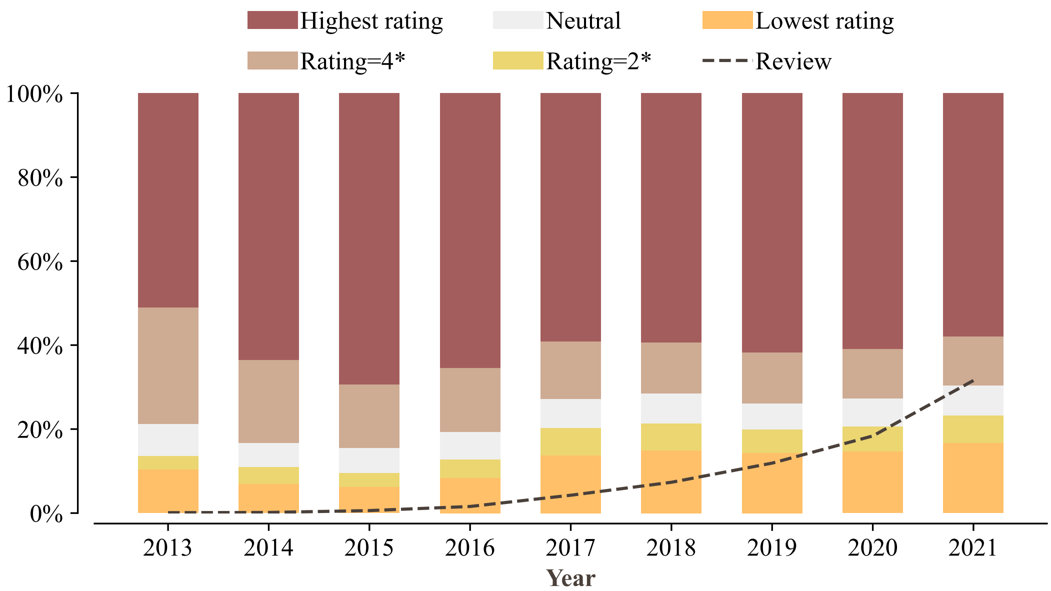

Fig. 10: Customers' ratings on traditional vacuums (on a yearly basis).

```
>>> # tvc.view_stats_on_ratings(by='month', save_as=".svg", verbose=True)
>>> tvc.view_stats_on_ratings(by='month')
```

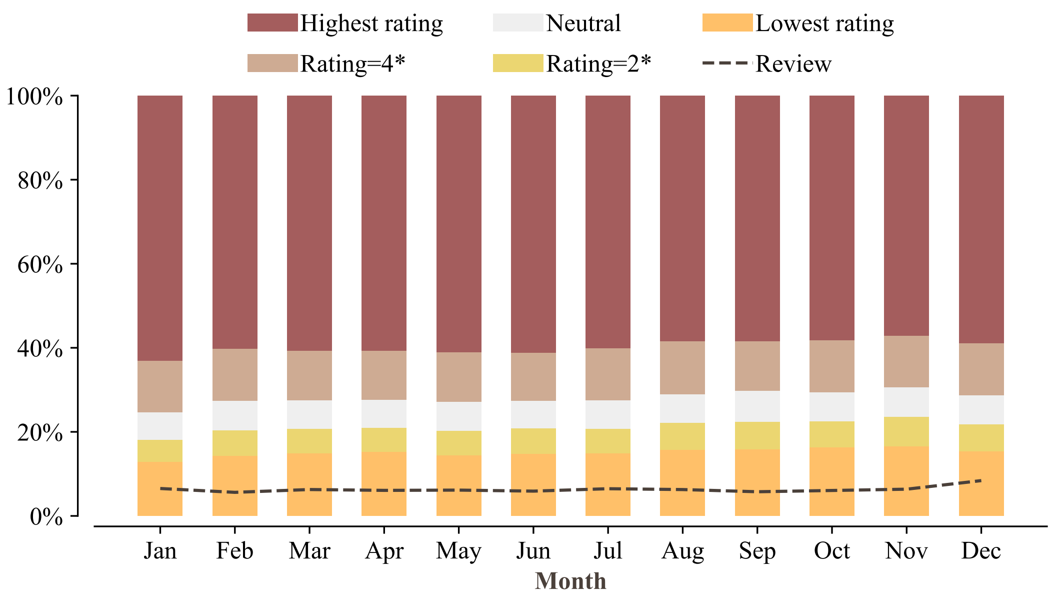

Fig. 11: Customers' ratings on traditional vacuums (on a monthly basis).

```
>>> from src.processor import SmartThermostats
>>> smt = SmartThermostats(load_prep_data=True, load_preprocd_data=False)
```

(continues on next page)

(continued from previous page)

```
>>> # smt.view_stats_on_ratings(by='year', save_as=".svg", verbose=True)
>>> smt.view_stats_on_ratings(by='year')
```

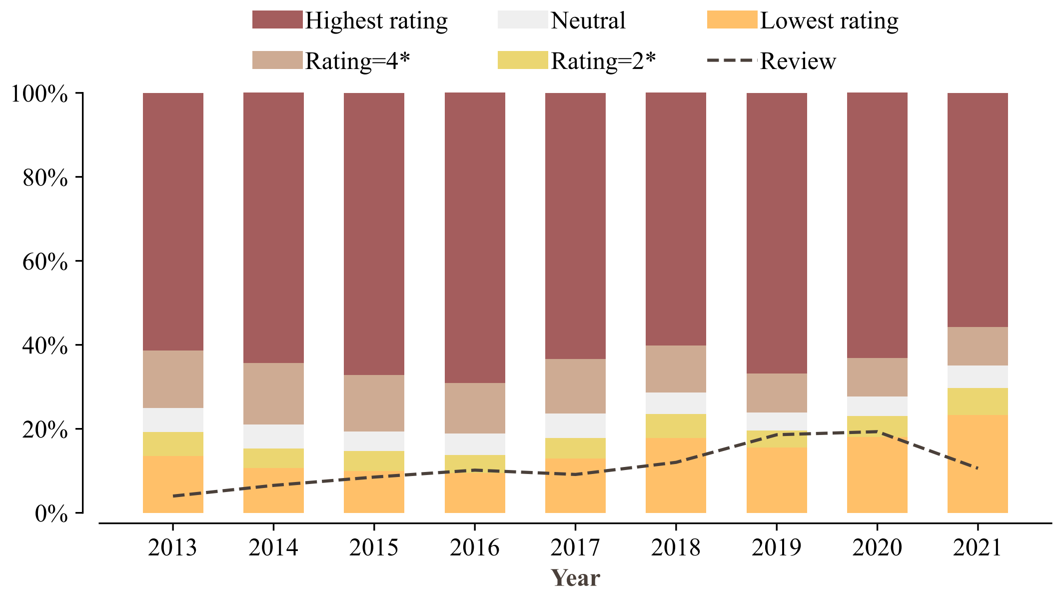

Fig. 12: Customers' ratings on smart thermostats (on a yearly basis).

```
>>> # smt.view_stats_on_ratings(by='month', save_as=".svg", verbose=True)
>>> smt.view_stats_on_ratings(by='month')
```

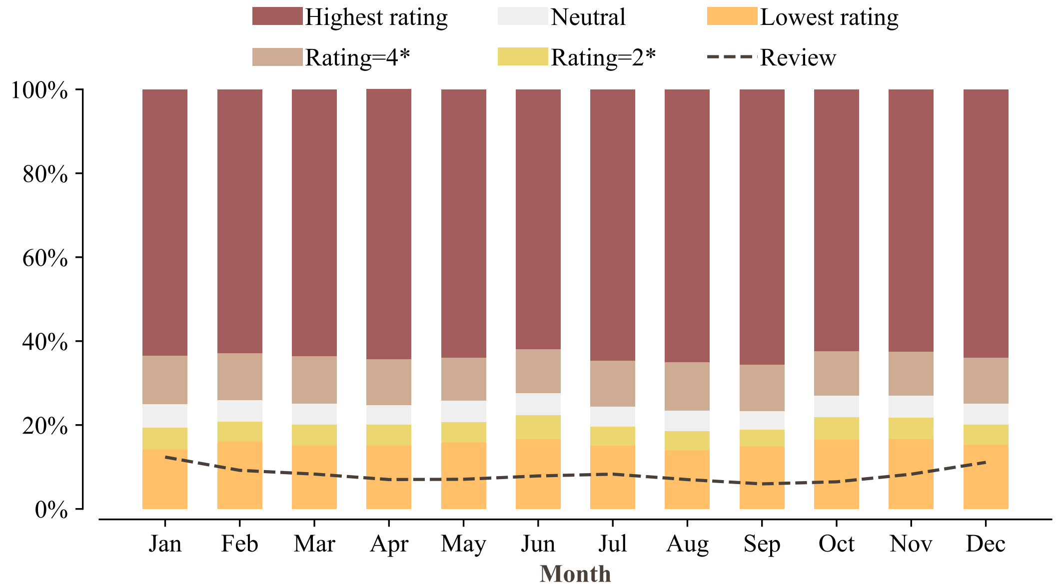

Fig. 13: Customers' ratings on smart thermostats (on a monthly basis).

4.2.2 RoboticVacuumCleaners

class src.processor.**RoboticVacuumCleaners**(*load_preprocd_data=True, load_prep_data=False, load_raw_data=False, **kwargs*)

Process the reviews of *robot vacuum cleaners*.

This class inherits from the `_Reviews` class.

Parameters

- **load_preprocd_data** (*bool*) – Whether to load the preprocessed data; defaults to `False`.
- **load_prep_data** (*bool*) – Whether to load the preparatory data; defaults to `False`.
- **load_raw_data** (*bool*) – Whether to load the raw data; defaults to `False`.
- **kwargs** – [Optional] parameters for initiating the class `_Base`.

Examples:

```
>>> from src.processor import RoboticVacuumCleaners
>>> rvc = RoboticVacuumCleaners()
>>> rvc.PRODUCT_NAME
'Robotic vacuum cleaners'
>>> rvc.preprocd_data.shape
(101608, 19)
>>> rvc = RoboticVacuumCleaners(verified_reviews_only=True)
>>> rvc.preprocd_data.shape
(89989, 19)
```

Attributes:

ORIGINAL_REVIEW_COLUMN_NAME	Default column name of original review text.
PRODUCT_CATEGORY	Category of the product.
PRODUCT_NAME	Name of the product.
PRODUCT_TYPE	Type of the product.
SCHEMA_NAME	Schema name.
SQL_QUERY	PostgreSQL query statement to read the whole table.
TABLE_IN_QUERY	Full table in PostgreSQL query statement.
TABLE_NAME	Table name.

RoboticVacuumCleaners.ORIGINAL_REVIEW_COLUMN_NAME

RoboticVacuumCleaners.**ORIGINAL_REVIEW_COLUMN_NAME: str = 'ReviewText'**

Default column name of original review text.

RoboticVacuumCleaners.PRODUCT_CATEGORY

`RoboticVacuumCleaners.PRODUCT_CATEGORY: str = 'Vacuum cleaners'`

Category of the product.

RoboticVacuumCleaners.PRODUCT_NAME

`RoboticVacuumCleaners.PRODUCT_NAME: str = 'Robotic vacuum cleaners'`

Name of the product.

RoboticVacuumCleaners.PRODUCT_TYPE

`RoboticVacuumCleaners.PRODUCT_TYPE: str = 'Robotic'`

Type of the product.

RoboticVacuumCleaners.SCHEMA_NAME

`RoboticVacuumCleaners.SCHEMA_NAME: str = 'amazon_reviews'`

Schema name.

RoboticVacuumCleaners.SQL_QUERY

`RoboticVacuumCleaners.SQL_QUERY: str = 'SELECT * FROM "amazon_reviews"."vacuum_cleaners_robotic"'`

PostgreSQL query statement to read the whole table.

RoboticVacuumCleaners.TABLE_IN_QUERY

`RoboticVacuumCleaners.TABLE_IN_QUERY: str = '"amazon_reviews"."vacuum_cleaners_robotic"'`

Full table in PostgreSQL query statement.

RoboticVacuumCleaners.TABLE_NAME

`RoboticVacuumCleaners.TABLE_NAME: str = 'vacuum_cleaners_robotic'`

Table name.

Note: No directly defined methods. See inherited class methods.

4.2.3 TraditionalVacuumCleaners

`class src.processor.TraditionalVacuumCleaners`(*load_preprocd_data=True, load_prep_data=False, load_raw_data=False, **kwargs*)

Process the reviews of *traditional vacuum cleaners*.

This class inherits from the `_Reviews` class.

Parameters

- **`load_preprocd_data`** (*`bool`*) – Whether to load the preprocessed data; defaults to `False`.
- **`load_prep_data`** (*`bool`*) – Whether to load the preparatory data; defaults to `False`.
- **`load_raw_data`** (*`bool`*) – Whether to load the raw data; defaults to `False`.

- **kwargs** – [Optional] parameters for initiating the class `_Base`.

Examples:

```
>>> from src.processor import TraditionalVacuumCleaners
>>> tvc = TraditionalVacuumCleaners()
Loading "data\amazon_reviews\vacuum_cleaners\traditional\preprocd_data\preprocd_...
>>> tvc.PRODUCT_NAME
'Traditional vacuum cleaners'
>>> tvc.preprocd_data.shape
(146656, 19)
>>> tvc = TraditionalVacuumCleaners(verified_reviews_only=True)
Loading "data\amazon_reviews\vacuum_cleaners\traditional\preprocd_data\preprocd_...
>>> tvc.preprocd_data.shape
(131971, 19)
```

Attributes:

ORIGINAL_REVIEW_COLUMN_NAME	Default column name of original review text.
PRODUCT_CATEGORY	Category of the product.
PRODUCT_NAME	Name of the product.
PRODUCT_TYPE	Type of the product.
SCHEMA_NAME	Schema name.
SQL_QUERY	PostgreSQL query statement to read the whole table.
TABLE_IN_QUERY	Full table in PostgreSQL query statement.
TABLE_NAME	Table name.

TraditionalVacuumCleaners.ORIGINAL_REVIEW_COLUMN_NAME

`TraditionalVacuumCleaners.`**`ORIGINAL_REVIEW_COLUMN_NAME: str = 'ReviewText'`**

Default column name of original review text.

TraditionalVacuumCleaners.PRODUCT_CATEGORY

`TraditionalVacuumCleaners.`**`PRODUCT_CATEGORY: str = 'Vacuum cleaners'`**

Category of the product.

TraditionalVacuumCleaners.PRODUCT_NAME

`TraditionalVacuumCleaners.`**`PRODUCT_NAME: str = 'Traditional vacuum cleaners'`**

Name of the product.

TraditionalVacuumCleaners.PRODUCT_TYPE

`TraditionalVacuumCleaners.`**`PRODUCT_TYPE: str = 'Traditional'`**

Type of the product.

TraditionalVacuumCleaners.SCHEMA_NAME

`TraditionalVacuumCleaners.SCHEMA_NAME: str = 'amazon_reviews'`

Schema name.

TraditionalVacuumCleaners.SQL_QUERY

`TraditionalVacuumCleaners.SQL_QUERY: str = 'SELECT * FROM "amazon_reviews"."vacuum_cleaners_traditional"'`

PostgreSQL query statement to read the whole table.

TraditionalVacuumCleaners.TABLE_IN_QUERY

`TraditionalVacuumCleaners.TABLE_IN_QUERY: str = '"amazon_reviews"."vacuum_cleaners_traditional"'`

Full table in PostgreSQL query statement.

TraditionalVacuumCleaners.TABLE_NAME

`TraditionalVacuumCleaners.TABLE_NAME: str = 'vacuum_cleaners_traditional'`

Table name.

Note: No directly defined methods. See inherited class methods.

4.2.4 SmartThermostats

class `src.processor.SmartThermostats`(*load_preprocd_data=True, **kwargs*)

Process the reviews of *smart thermostats*.

This class inherits from the `_Reviews` class.

Parameters

- **load_preprocd_data** (`bool`) – Whether to load the preprocessed data; defaults to `False`.
- **kwargs** – [Optional] parameters for initiating the class `_Base`

Examples:

```
>>> from src.processor import SmartThermostats
>>> smt = SmartThermostats()
>>> smt.PRODUCT_NAME
'Smart thermostats'
>>> smt.preprocd_data.shape
(46317, 19)
>>> smt = SmartThermostats(verified_reviews_only=True)
>>> smt.preprocd_data.shape
(38810, 19)
```

Attributes:

ORIGINAL_REVIEW_COLUMN_NAME	Default column name of original review text.
PRODUCT_CATEGORY	Category of the product.
PRODUCT_NAME	Name of the product.
PRODUCT_TYPE	Type of the product.
SCHEMA_NAME	Schema name.
SQL_QUERY	PostgreSQL query statement to read the whole table.
TABLE_IN_QUERY	Full table in PostgreSQL query statement.
TABLE_NAME	Table name.

SmartThermostats.ORIGINAL_REVIEW_COLUMN_NAME

`SmartThermostats.ORIGINAL_REVIEW_COLUMN_NAME: str = 'ReviewText'`

Default column name of original review text.

SmartThermostats.PRODUCT_CATEGORY

`SmartThermostats.PRODUCT_CATEGORY: str = 'Thermostats'`

Category of the product.

SmartThermostats.PRODUCT_NAME

`SmartThermostats.PRODUCT_NAME: str = 'Smart thermostats'`

Name of the product.

SmartThermostats.PRODUCT_TYPE

`SmartThermostats.PRODUCT_TYPE: str = 'Smart'`

Type of the product.

SmartThermostats.SCHEMA_NAME

`SmartThermostats.SCHEMA_NAME: str = 'amazon_reviews'`

Schema name.

SmartThermostats.SQL_QUERY

`SmartThermostats.SQL_QUERY: str = 'SELECT * FROM "amazon_reviews"."thermostats_smart"'`

PostgreSQL query statement to read the whole table.

SmartThermostats.TABLE_IN_QUERY

`SmartThermostats.TABLE_IN_QUERY: str = '"amazon_reviews"."thermostats_smart"'`

Full table in PostgreSQL query statement.

SmartThermostats.TABLE_NAME

`SmartThermostats.TABLE_NAME: str = 'thermostats_smart'`

Table name.

Note: No directly defined methods. See inherited class methods.

4.3 modeller

The module is used to apply algorithms on the data generated from *processor*.

_Base(product_category, product_type[, ...])	A base class for modelling trials.
LogisticRegressionModel(product_category, ...)	A class for instantiating a logistic regression model for the review texts.
LatentDirichletAllocation(product_category, ...)	A class for instantiating LDA (Latent Dirichlet Allocation) model for the review texts.

4.3.1 _Base

class src.modeller.**_Base**(*product_category, product_type, sentiment_on='dual_scale', review_column_name=None, random_state=0, **kwargs*)

A base class for modelling trials.

Parameters

- **product_category** (*str*) – Product category.
- **product_type** (*str*) – product type, valid values include {'Robotic', 'Traditional'}
- **sentiment_on** (*str*) – column name for the metric on which sentiment is determined, defaults to 'dual_scale'
- **review_column_name** (*str | None*) – column name of the review texts; when review_column_name=None (default), it defaults to 'review_text'
- **random_state** (*int | None*) – random seed number, defaults to 0
- **kwargs** – [optional] parameters for initiating the class _Base

Variables

- **random_state** (*int | None*) – A random seed number.
- **product_type** (*str*) – The type of product.
- **reviews** (RoboticVacuumCleaners | TraditionalVacuumCleaners) – An instance of the class *RoboticVacuumCleaners* or *TraditionalVacuumCleaners*.
- **sentiment_column_name** (*str*) – Column name of the sentiment.
- **data** (*pandas.DataFrame*) – Preprocessed data of the reviews.

- **review_column_name** (*str*) – Column name of the review texts.

Attributes:

PRODUCT_CATEGORIES	Valid names of a product category.
PRODUCT_TYPES	Valid types of a product.
REVIEW_COLUMN_NAME	Column name of the review texts.
VALID_SENTIMENT_LABELS	Valid sentiment labels.

_Base.PRODUCT_CATEGORIES

`_Base.PRODUCT_CATEGORIES: str = {'Thermostats', 'Vacuum cleaners'}`

Valid names of a product category.

_Base.PRODUCT_TYPES

`_Base.PRODUCT_TYPES: set = {'Robotic', 'Smart', 'Traditional'}`

Valid types of a product.

_Base.REVIEW_COLUMN_NAME

`_Base.REVIEW_COLUMN_NAME: str = 'review_text'`

Column name of the review texts.

_Base.VALID_SENTIMENT_LABELS

`_Base.VALID_SENTIMENT_LABELS: set = {'negative', 'neutral', 'positive'}`

Valid sentiment labels.

Methods:

cd_models(*subdir, **kwargs)	Change to the directory where the models and their relevant files are saved.

_Base.cd_models

`_Base.cd_models(*subdir, **kwargs)`

Change to the directory where the models and their relevant files are saved.

Parameters

- **subdir** (*str*) – name of directory or names of directories (and/or a filename)
- **kwargs** – [optional] parameters of *src.processor._Base.cdd*

Returns

pathname of the directory for storing models

Return type

str

4.3.2 LogisticRegressionModel

class src.modeller.**LogisticRegressionModel**(*product_category*, *product_type*, *random_state=0*, ***kwargs*)

A class for instantiating a logistic regression model for the review texts.

Parameters

- **product_category** (*str*) – Product category.
- **product_type** (*str*) – product type, valid values include {'Robotic', 'Traditional'}
- **random_state** (*int or None*) – random seed number, defaults to 0
- **kwargs** – [optional] parameters of the class *_Base*

Variables

- **word_vectorizer** (*sklearn.feature_extraction.text.CountVectorizer*) – A collection of text documents represented as a matrix of token counts.
- **word_counter** (*scipy.sparse.csr_matrix*) – Document-term matrix.
- **logit** (*sklearn.linear_model.LogisticRegression or None*) – Object of logistic regression model.
- **score** (*float or None*) – Mean accuracy on test data.
- **coefficients** (*list or None*) – Estimated coefficients.
- **odds_ratios** (*list or None*) – Odds ratios.
- **summary** (*pandas.DataFrame or None*) – Summary of model coefficients.

Examples:

```
>>> from src.modeller import LogisticRegressionModel
>>> logit_rvc = LogisticRegressionModel('vacuum', product_type='robotic')
>>> logit_rvc.NAME
'Logistic Regression'
>>> logit_rvc.review_column_name
'review_text'
>>> logit_rvc.sentiment_column_name
'sentiment_on_dual_scale'
```

Attributes:

NAME	str: Name of the model.

LogisticRegressionModel.NAME

`LogisticRegressionModel.NAME = 'Logistic Regression'`

str: Name of the model.

Methods:

logistic_regression([test_size, ...])	An example model: a multinomial logistic regression model.

LogisticRegressionModel.logistic_regression

`LogisticRegressionModel.logistic_regression`(*test_size=0.15*, *feature_scaled=True*, *cv=None*, *solver='saga'*, *max_iter=10000*, *n_jobs=None*, *verbose=False*, *ret_summary=False*, ***kwargs*)

An example model: a multinomial logistic regression model.

Parameters

- **test_size** (*float*) – proportion of a test set, defaults to `.15`
- **feature_scaled** (*bool*) – whether to scale the feature data, defaults to `True`
- **cv** (*int or None*) – `cv` of the class sklearn.linear_model.LogisticRegressionCV, defaults to `None`
- **solver** (*str*) – name of solver, defaults to `'saga'`
- **max_iter** (*int*) – maximum number of iteration, defaults to `5000`
- **n_jobs** (*int or None*) – defaults to `6`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **ret_summary** (*bool*) – whether to return a summary of estimated coefficients, defaults to `False`
- **kwargs** – [optional] parameters of sklearn.linear_model.LogisticRegression

Examples:

```
>>> from src.modeller import LogisticRegressionModel
>>> logit_rvc = LogisticRegressionModel(product_type='Robotic')
>>> logit_rvc.logistic_regression(verbose=True)
>>> print('Mean accuracy: %.2f%%' % (logit_rvc.score * 100))
Mean accuracy: 95.91%
>>> logit_rvc.summary
      feature_name  coef_positive  coef_neutral  coef_negative
0            great      12.758334     -1.925204     -10.833130
1             love       9.943116     -1.847491      -8.095624
2             easy       6.910460     -1.138867      -5.771593
3          amazing       5.308514     -0.841767      -4.466747
```

(continues on next page)

(continued from previous page)

```
4              well       4.889721     -1.484675      -3.405046
            ...            ...          ...            ...
21938        return      -4.118999     -0.010412       4.129411
21939          dead      -4.137166     -0.349761       4.486927
21940      horrible      -4.155905     -0.349945       4.505850
21941          stop      -4.282742      0.665926       3.616816
21942       useless      -4.409014     -0.188151       4.597165
[21943 rows x 4 columns]
>>> logit_tvc = LogisticRegressionModel(product_type='Traditional')
>>> logit_tvc.logistic_regression(verbose=True)
>>> print('Mean accuracy: %.2f%%' % (logit_tvc.score * 100))
Mean accuracy: 96.02%
>>> logit_tvc.summary
        feature_name  coef_positive  coef_neutral  coef_negative
0               easy      11.456976     -1.481144      -9.975831
1               love      10.424510     -2.368445      -8.056065
2              great       8.296676     -2.157647      -6.139029
3            amazing       6.222923     -1.157312      -5.065612
4               well       5.916765     -1.332715      -4.584049
                 ...            ...           ...            ...
21390  disappointing      -3.376083      0.898763       2.477320
21391       terrible      -4.017264     -0.722647       4.739912
21392       horrible      -4.205845     -0.143932       4.349778
21393           poor      -4.870857     -0.408490       5.279347
21394         return      -5.675754      0.858362       4.817391
[21395 rows x 4 columns]
```

4.3.3 LatentDirichletAllocation

class src.modeller.**LatentDirichletAllocation**(*product_category*, *product_type*, *sentiment_on='dual_scale'*, *review_column_name='review_text'*, *random_state=0*, ***kwargs*)

A class for instantiating LDA (Latent Dirichlet Allocation) model for the review texts.

Parameters

- **product_category** (*str*) – Product category.
- **product_type** (*str*) – product type, valid values include {'Robotic', 'Traditional'}
- **sentiment_on** (*str*) – column name for the metric on which sentiment is determined, defaults to 'dual_scale'
- **review_column_name** (*str or None*) – column name of the review texts; when review_column_name=None (default), it defaults to 'review_text'
- **random_state** (*int or None*) – random seed number
- **kwargs** – [optional] parameters of the class *_Base*

Variables

- **min_counts** (*list*) – A list of min_count for model evaluation.
- **thresholds** (*list*) – A list of threshold for model evaluation.

- **corpus_proportions** (*numpy.ndarray*) – An array of corpus proportions for model evaluation.
- **pos_topic_numbers** (*range*) – A range of topic numbers for model evaluation on positive reviews.
- **pos_alphas** (*list*) – A list of `alpha` for model evaluation on positive reviews.
- **pos_etas** (*list*) – A list of `eta` for model evaluation on positive reviews.
- **neg_topic_numbers** (*range*) – A range of topic numbers for model evaluation on negative reviews.
- **neg_alphas** (*list*) – A list of `alpha` for model evaluation on negative reviews.
- **neg_etas** (*list*) – A list of `eta` for model evaluation on negative reviews.
- **sentiment** (*str or None*) – Label of sentiment.
- **pos_tokenized_docs** (*list*) – Tokenized documents of positive reviews.
- **neg_tokenized_docs** (*list*) – Tokenized documents of negative reviews.
- **neu_tokenized_docs** (*list*) – Tokenized documents of neutral reviews.
- **tokenized_docs** (*dict*) – Data of tokenized documents.
- **pos_eval_summary** (*pandas.DataFrame*) – A summary of model evaluation results for positive reviews.
- **neg_eval_summary** (*pandas.DataFrame*) – A summary of model evaluation results for negative reviews.
- **neu_eval_summary** (*pandas.DataFrame*) – A summary of model evaluation results for neutral reviews.
- **eval_summary** (*pandas.DataFrame or None*) – All summaries of model evaluation results.
- **eval_summary** – The summary of model evaluation for the given `sentiment`.

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> lda_robovac = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> lda_robovac.VALID_SENTIMENT_LABELS
{'negative', 'neutral', 'positive'}
>>> lda_robovac.reviews.preprocd_data.shape
(77775, 18)
>>> lda_tradvac = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> lda_tradvac.VALID_SENTIMENT_LABELS
{'negative', 'neutral', 'positive'}
>>> lda_tradvac.reviews.preprocd_data.shape
(110978, 18)
>>> lda_smtherms = LatentDirichletAllocation('thermostats', product_type='smart')
>>> lda_smtherms.VALID_SENTIMENT_LABELS
{'negative', 'neutral', 'positive'}
```

(continues on next page)

(continued from previous page)

```
>>> lda_smtherms.reviews.preprocd_data.shape
(26285, 18)
```

Attributes:

NAME	Name of the model.

LatentDirichletAllocation.NAME

`LatentDirichletAllocation.NAME: str = 'Latent Dirichlet Allocation (LDA)'`

Name of the model.

Methods:

cd_models(*args, **kwargs)	Change to the directory where the models and their relevant files are saved.
evaluate_models([verbose])	Evaluate LDA models for each group of reviews (e.g. positive reviews and negative reviews).
fetch_evaluation_summary(sentiment[, verbose])	Fetch the summary of the LDA model evaluation results.
find_original_reviews(sentiment[, i, ...])	Find original review texts containing terms that are most relevant to each topic, for the top 10 models given their coherence scores.
get_coherence_score(corpus, id2word, texts, ...)	Get the coherence score for an LDA model.
get_common_words(topics_data)	Get common words from a number of topics estimated by an LDA model.
get_tokenized_docs(docs, sentiment[, ...])	Get tokenized documents.
get_tokens(doc[, bespoke_stopwords])	Get tokens of a given document.
get_top_terms_of_topics(sentiment[, i, ...])	Get the top `num_terms` terms for each topic.
get_topics(sentiment, i[, n_top_tokens, ...])	Get the top `n_top_tokens` words/phrases for LDA models.
get_vis_data(sentiment[, i, ...])	Get visualisation data for LDA models.
make_corpus(tokenized_docs[, ngram, ...])	Make a corpus.
prep_eval_corpuses(corpus[, proportions])	Get a number of corpuses by specified proportions for model evaluation.
retrieve_original_text(corpus, id2word, texts)	Retrieve original review texts.
specify_adhoc_stopwords()	Create a set of ad-hoc stopwords.
train_model(corpus, id2word, texts, num_topics)	Train an LDA model.
train_models(corpus, id2word, texts[, ...])	Train a number of LDA models.
view_evaluation_summary(sentiment[, ...])	Visualise the results of the evaluation summary.

LatentDirichletAllocation.cd_models

`LatentDirichletAllocation.`**`cd_models`**`(`**args, **kwargs*`)`

Change to the directory where the models and their relevant files are saved.

Returns

pathname of the directory for storing models

Return type

str

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> import os
>>> lda = LatentDirichletAllocation('vacuum', 'robotic', load_preprocd_data=False)
>>> os.path.relpath(lda.cd_models())
'data\amazon_reviews\vacuum_cleaners\robotic\models\lda'
>>> lda = LatentDirichletAllocation('vacuum', 'traditional', load_preprocd_data=False)
>>> os.path.relpath(lda.cd_models())
'data\amazon_reviews\vacuum_cleaners\traditional\models\lda'
>>> lda = LatentDirichletAllocation('therms', 'smart', load_preprocd_data=False)
>>> os.path.relpath(lda.cd_models())
'data\amazon_reviews\thermostats\smart\models\lda'
```

LatentDirichletAllocation.evaluate_models

`LatentDirichletAllocation.`**`evaluate_models`**`(`*verbose=True*`)`

Evaluate LDA models for each group of reviews (e.g. positive reviews and negative reviews).

Parameters

verbose (*bool | int*) – whether to print relevant information in console, defaults to `True`

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> lda.evaluate_models()  # (This may take a huge amount of time.)
>>> # Traditional vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> lda.evaluate_models()  # (This may take a huge amount of time.)
>>> # Smart thermostats
>>> lda = LatentDirichletAllocation('therms', product_type='smart')
>>> lda.min_counts = range(1, 6)
>>> lda.thresholds = [0.0001, 0.001, 0.01, 0.1, 0.5, 1.0]
>>> lda.corpus_proportions = [1.0]
>>> lda.pos_topic_numbers = range(2, 6)
>>> lda.neg_topic_numbers = range(2, 6)
>>> lda.evaluate_models()  # (This may take a huge amount of time.)
```

LatentDirichletAllocation.fetch_evaluation_summary

`LatentDirichletAllocation.`**`fetch_evaluation_summary`**(*sentiment*, *verbose=False*)

Fetch the summary of the LDA model evaluation results.

Parameters

- **`sentiment`** (*`str`*) – label of sentiment; options include `VALID_SENTIMENT_LABELS`
- **`verbose`** (*`bool or int`*) – whether to print relevant information in console, defaults to `False`

Returns

summary of the LDA model evaluation results for the given `sentiment`

Return type

pandas.DataFrame

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> import pandas as pd
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> pos_lda_eval_summary = lda.fetch_evaluation_summary(sentiment='positive')
>>> isinstance(pos_lda_eval_summary, pd.DataFrame)
True
>>> neg_lda_eval_summary = lda.fetch_evaluation_summary(sentiment='negative')
>>> isinstance(neg_lda_eval_summary, pd.DataFrame)
True
>>> neu_lda_eval_summary = lda.fetch_evaluation_summary(sentiment='neutral')
>>> neu_lda_eval_summary is None
True
>>> # Traditional vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> pos_lda_eval_summary = lda.fetch_evaluation_summary(sentiment='positive')
>>> isinstance(pos_lda_eval_summary, pd.DataFrame)
True
>>> neg_lda_eval_summary = lda.fetch_evaluation_summary(sentiment='negative')
>>> isinstance(neg_lda_eval_summary, pd.DataFrame)
True
>>> neu_lda_eval_summary = lda.fetch_evaluation_summary(sentiment='neutral')
>>> neu_lda_eval_summary is None
True
```

LatentDirichletAllocation.find_original_reviews

`LatentDirichletAllocation.`**`find_original_reviews`**(*sentiment*, *i=None*, *num_terms=15*, *lambda_=0.0*, *export_to_file=False*, *verbose=False*, ***kwargs*)

Find original review texts containing terms that are most relevant to each topic, for the top 10 models given their coherence scores.

Parameters

- **`sentiment`** (*`str`*) – label of sentiment; options include `VALID_SENTIMENT_LABELS`

- **i** (*int or Iterable or None*) – row index or indices of the model evaluation summary, defaults to `None`
- **num_terms** (*int*) – number of terms to be considered
- **lambda** (*float or int*) – lambda value for the LDA model
- **export_to_file** (*bool*) – whether to save the results to a spreadsheet file, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of the method *get_top_terms_of_topics()*

Returns

topic-specific original review texts for the top 10 models given their coherence scores

Return type

collections.OrderedDict

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Positive reviews:
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> # lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> pos_reviews = lda.find_original_reviews(
...     sentiment='positive', i=range(10), ignore_auto_alpha=True, verbose=True)
>>> # Negative reviews:
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> # lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> neg_reviews = lda.find_original_reviews(
...     sentiment='negative', i=range(10), ignore_auto_alpha=True, verbose=True)
```

LatentDirichletAllocation.get_coherence_score

`LatentDirichletAllocation.`**`get_coherence_score`**(*corpus, id2word, texts, num_topics, alpha, eta, **kwargs*)

Get the coherence score for an LDA model.

Parameters

- **corpus** (*list*) – corpus (i.e. term-document frequency, see gensim.corpora.Dictionary.doc2bow())
- **id2word** (*gensim.corpora.Dictionary*) – id-word mapping dictionary (see gensim.corpora.Dictionary())
- **texts** (*list*) – lemmatized review texts
- **num_topics** (*int*) – number of topics, see `num_topics` of gensim.models.LdaMulticore()
- **alpha** (*float or numpy.ndarray or list*) – alpha of gensim.models.LdaMulticore()

- **eta** (*float or numpy.ndarray or list*) – eta of gensim.models.LdaMulticore()
- **kwargs** – [optional] parameters of gensim.models.LdaMulticore()

Returns
coherence score of the LDA model given the specified parameters

Return type
float

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> # Traditional vacuum cleaners
>>> # lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> example_docs = lda.data['review_text']
>>> pos_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='positive')
>>> # Consider bi-grams
>>> pos_corpus, pos_id2word, pos_texts = lda.make_corpus(pos_tokenized_docs, ngram=2)
>>> pos_coherence_score = lda.get_coherence_score(
...     pos_corpus, pos_id2word, pos_texts, num_topics=3, alpha=1, eta=1)
>>> isinstance(pos_coherence_score, float)
True
```

LatentDirichletAllocation.get_common_words

static LatentDirichletAllocation.**get_common_words**(*topics_data*)
Get common words from a number of topics estimated by an LDA model.

Parameters
topics_data (*pandas.DataFrame*) – data of a number of topics

Returns
a set of common words

Return type
set

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> pos_top_topics_50tokens = lda.get_topics('positive', i=76, n_top_tokens=50)
>>> lda.get_common_words(pos_top_topics_50tokens)
{'area',
 'clean',
 'cleaning',
 'long',
 'look',
 'room',
 'set',
 'time',
 'try',
 'want'}
```

LatentDirichletAllocation.get_tokenized_docs

`LatentDirichletAllocation.`**`get_tokenized_docs`**(*docs*, *sentiment*, *bespoke_stopwords=None*)

Get tokenized documents.

Parameters

- **docs** (*Iterable*) – any documents
- **sentiment** (*str*) – label of sentiment; options are `VALID_SENTIMENT_LABELS`
- **bespoke_stopwords** (*set*) – a set of bespoke stopwords

Returns

tokenized documents

Return type

list

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> example_docs = lda.data['review_text']
>>> pos_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='positive')
>>> isinstance(pos_tokenized_docs, list)
True
>>> neg_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='negative')
>>> isinstance(neg_tokenized_docs, list)
True
>>> # Traditional vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> example_docs = lda.data['review_text']
>>> pos_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='positive')
>>> isinstance(pos_tokenized_docs, list)
True
>>> neg_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='negative')
>>> isinstance(neg_tokenized_docs, list)
True
>>> # Smart thermostats
>>> lda = LatentDirichletAllocation('thermos', product_type='smart')
>>> example_docs = lda.data['review_text']
>>> pos_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='positive')
>>> isinstance(pos_tokenized_docs, list)
True
>>> neg_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='negative')
>>> isinstance(neg_tokenized_docs, list)
True
```

LatentDirichletAllocation.get_tokens

classmethod LatentDirichletAllocation.**get_tokens**(*doc*, *bespoke_stopwords=None*)

Get tokens of a given document.

Parameters

- **doc** (*str*) – any document
- **bespoke_stopwords** (*set or list or tuple or None*) – a set of bespoke stopwords, defaults to `None`

Returns

tokens of the given `doc`

Return type

list

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> example_doc = lda.data['review_text'][0]
>>> example_doc_tokens = lda.get_tokens(example_doc, bespoke_stopwords=None)
>>> isinstance(example_doc_tokens, list)
True
>>> # Traditional vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> example_doc = lda.data['review_text'][0]
>>> example_doc_tokens = lda.get_tokens(example_doc, bespoke_stopwords=None)
>>> isinstance(example_doc_tokens, list)
True
```

LatentDirichletAllocation.get_top_terms_of_topics

LatentDirichletAllocation.**get_top_terms_of_topics**(*sentiment*, *i=None*, *ignore_auto_alpha=False*, *num_terms=15*, *lambda_=0.0*, *vis_data_to_html=False*, *update=False*, *verbose=False*, ***kwargs*)

Get the top `num_terms` terms for each topic.

Parameters

- **sentiment** (*str*) – label of sentiment; options include `VALID_SENTIMENT_LABELS`
- **i** (*int or Iterable or None*) – row index or indices of the model evaluation summary, defaults to `None`
- **ignore_auto_alpha** (*bool*) – whether to ignore the situation when `alpha='auto'`
- **num_terms** (*int*) – number of terms to be considered
- **lambda** (*float or int*) – lambda value for the LDA model

- **vis_data_to_html** (*bool*) – whether to save the model visualisation data to an HTML file, defaults to `False`
- **update** (*bool*) – whether to replace the existing HTML file with an updated one, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of pyLDAvis.gensim_models.prepare()

Returns

the top `num_terms` terms for each topic

Return type

collections.OrderedDict

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> top_terms = lda.get_top_terms_of_topics(sentiment='positive', i=76)
>>> top_terms
OrderedDict([('LDA_076',
                          Topic1           Topic2          Topic3
              0          large_dog        keep_zone    washable_pad
              1          twice_day  firmware_update  authentic_part
              2    hair_everywhere           define        chemical
              3                mom       clean_zone    reusable_pad
              4           dog_pick         firmware  disposable_pad
              5     clean_everyday        cleanbase        damp_wet
              6         life_saver  software_update    sweeping_pad
              7          wish_soon            cloud          bottle
              8          life_easy            width       streaking
              9          lifesaver         homebase           bravva
              10            obsess       map_create    dry_sweeping
              11       amazed_pick             beam      change_pad
              12         sweep_day           reboot    cleaning_pad
              13             hairy             ugly          capful
              14        clean_hair            remap          dilute
              15          everyday        avoidance        reusable)])
```

LatentDirichletAllocation.get_topics

`LatentDirichletAllocation.`**`get_topics`**(*sentiment*, *i*, *n_top_tokens=50*, *export_to_file=True*, *verbose=True*, ***kwargs*)

Get the top `n_top_tokens` words/phrases for LDA models.

Parameters

- **sentiment** (*str*) – label of sentiment; options include `VALID_SENTIMENT_LABELS`
- **i** (*int | list*) – an index or a list of indices of the dataframe of model evaluation summary
- **n_top_tokens** (*int*) – number of words/phrases in each of the resulting topics, defaults to `50`; see `topn` of gensim.models.LdaMulticore.top_topics()

- **export_to_file** (*bool*) – whether to save the results to a spreadsheet file; defaults to `True`.
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [Optional] additional parameters for the function pyhelpers.store.save_spreadsheets()

Returns
the top `n_top_tokens` words/phrases for each of the specified LDA models

Return type
collections.OrderedDict

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> import collections
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> pos_top_topics_data = lda.get_topics(
...     sentiment='positive', i=76, n_top_tokens=50, export_to_file=False)
>>> isinstance(pos_top_topics_data, collections.OrderedDict)
True
>>> pos_top_topics_data[76].shape
(50, 6)
>>> neg_top_topics_data = lda.get_topics(
...     sentiment='negative', i=[98, 123], n_top_tokens=50, export_to_file=False)
>>> isinstance(neg_top_topics_data, collections.OrderedDict)
True
>>> list(neg_top_topics_data.keys())
[98, 123]
```

LatentDirichletAllocation.get_vis_data

`LatentDirichletAllocation.`**`get_vis_data`**(*sentiment, i=None, ignore_auto_alpha=False, export_to_html=False, update=False, verbose=False, **kwargs*)

Get visualisation data for LDA models.

Parameters

- **sentiment** (*str*) – label of sentiment; options include `VALID_SENTIMENT_LABELS`
- **i** (*int or Iterable or None*) – row index or indices of the model evaluation summary, defaults to `None`
- **ignore_auto_alpha** (*bool*) – whether to ignore the situation when `alpha='auto'`
- **export_to_html** (*bool*) – whether to save the model visualisation data to an HTML file, defaults to `False`
- **update** (*bool*) – whether to replace the existing HTML file with an updated one, defaults to `False`

- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of pyLDAvis.gensim_models.prepare()

Returns
prepared data for visualising the LDA model

Return type
pyLDAvis.PreparedData

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> product_category = 'vacuum'
>>> product_type = 'robotic'
```

Positive reviews:

```
>>> lda = LatentDirichletAllocation(product_category, product_type)
>>> # lda = LatentDirichletAllocation(product_category, product_type='traditional')
>>> # pos_lda_vis_data_1 = lda.get_vis_data(
... #     sentiment='positive', i=0, export_to_html=True, verbose=True)
>>> # pos_lda_vis_data_2 = lda.get_vis_data(
... #     sentiment='positive', i=range(50), verbose=True)
>>> pos_lda_vis_data_3 = lda.get_vis_data(
...     sentiment='positive', i=range(10), export_to_html=True, verbose=True,
...     ignore_auto_alpha=True)
```

Negative reviews:

```
>>> lda = LatentDirichletAllocation(product_category, product_type)
>>> # lda = LatentDirichletAllocation(product_category, product_type='traditional')
>>> # neg_lda_vis_data_1 = lda.get_vis_data(
... #     sentiment='negative', i=0, export_to_html=True, verbose=True)
>>> # neg_lda_vis_data_2 = lda.get_vis_data(
... #     sentiment='negative', i=range(50), verbose=True)
>>> neg_lda_vis_data_3 = lda.get_vis_data(
...     sentiment='negative', i=range(10), export_to_html=True, verbose=True,
...     ignore_auto_alpha=True)
```

LatentDirichletAllocation.make_corpus

`LatentDirichletAllocation.`**`make_corpus`**(*tokenized_docs, ngram=2, min_count=1, threshold=0.0001, scoring='npmi'*)

Make a corpus.

Parameters

- **tokenized_docs** (*list*) – tokenized documents
- **ngram** (*int*) – number of grams
- **min_count** (*int*) – `min_count` of the class gensim.models.phrases.Phrases(), defaults to `1`
- **threshold** (*float*) – `threshold` of the class gensim.models.phrases.Phrases(), defaults to `10e-5`

- **scoring** (*str*) – `scoring` of the class gensim.models.phrases.Phrases(), defaults to `'npmi'`

Returns
corpus (i.e. term-document frequency, see gensim.corpora.Dictionary.doc2bow()), id-word mapping dictionary (see gensim.corpora.Dictionary()), and lemmatized review texts

Return type
tuple

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> example_docs = lda.data['review_text']
>>> pos_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='positive')
>>> # Considering bi-grams
>>> pos_reviews_corpus = lda.make_corpus(pos_tokenized_docs, ngram=2)
>>> isinstance(pos_reviews_corpus, tuple)
True
>>> # Considering tri-grams
>>> pos_reviews_corpus = lda.make_corpus(pos_tokenized_docs, ngram=3)
>>> isinstance(pos_reviews_corpus, tuple)
True
>>> # Traditional vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> example_docs = lda.data['review_text']
>>> pos_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='positive')
>>> # Considering bi-grams
>>> pos_reviews_corpus = lda.make_corpus(pos_tokenized_docs, ngram=2)
>>> isinstance(pos_reviews_corpus, tuple)
True
>>> # Considering tri-grams
>>> pos_reviews_corpus = lda.make_corpus(pos_tokenized_docs, ngram=3)
>>> isinstance(pos_reviews_corpus, tuple)
True
```

LatentDirichletAllocation.prep_eval_corpuses

classmethod `LatentDirichletAllocation.`**`prep_eval_corpuses`**(*corpus*, *proportions=None*)
Get a number of corpuses by specified proportions for model evaluation.

Parameters

- **corpus** (*gensim.utils.ClippedCorpus or list*) – corpus (i.e. term-document frequency, see gensim.corpora.Dictionary.doc2bow())
- **proportions** (*Iterable or None*) – proportions, defaults to `None`

Returns
a list of corpuses and their respective proportions

Return type
Tuple[list, list]

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> # Traditional vacuum cleaners
>>> # lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> example_docs = lda.data['review_text']
>>> neg_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='negative')
>>> # Consider bi-grams
>>> neg_corpus, neg_id2word, neg_texts = lda.make_corpus(neg_tokenized_docs, ngram=2)
>>> neg_corpus_eval_lists, neg_corpus_eval_props = lda.prep_eval_corpuses(neg_corpus)
>>> isinstance(neg_corpus_eval_lists, list)
True
>>> len(neg_corpus_eval_lists)
1
>>> len(neg_corpus_eval_lists[0])
77775
>>> neg_corpus_eval_props
['100%']
>>> neg_corpus_eval_lists, neg_corpus_eval_props = lda.prep_eval_corpuses(
...     corpus=neg_corpus, proportions=[0.8])
>>> isinstance(neg_corpus_eval_lists, list)
True
>>> len(neg_corpus_eval_lists)
2
>>> list(map(len, neg_corpus_eval_lists))
[62220, 77775]
>>> neg_corpus_eval_props
['80%', '100%']
```

LatentDirichletAllocation.retrieve_original_text

LatentDirichletAllocation.**retrieve_original_text**(*corpus*, *id2word*, *texts*)

Retrieve original review texts.

Parameters

- **corpus** (*list*) – corpus (i.e. term-document frequency, see gensim.corpora.Dictionary.doc2bow())
- **id2word** (*gensim.corpora.Dictionary or list*) – id-word mapping dictionary (see gensim.corpora.Dictionary())
- **texts** (*list*) – lemmatized review texts

Returns

Data of original review texts.

Return type

pandas.DataFrame

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> lda = LatentDirichletAllocation(product_category='vacuum', product_type='robotic')
>>> sentiment = 'positive'
>>> tokenized_docs = lda.get_tokenized_docs(lda.data[lda.review_column_name], sentiment)
>>> eval_summary = lda.fetch_evaluation_summary(sentiment=sentiment)
```

(continues on next page)

(continued from previous page)

```
>>> params_columns = [
...     'min_count', 'threshold', 'num_topics', 'alpha', 'eta', 'corpus_proportion']
>>> min_count, threshold, num_topics, alpha, eta, corpus_prop = (
...     eval_summary.loc[76, params_columns].values)
>>> corpus, prop, id2word, texts = lda._make_eval_corpus(
...     sentiment=sentiment, tokenized_docs=tokenized_docs, corpus_prop=corpus_prop,
...     min_count=min_count, threshold=threshold, ngram=3)
>>> original_texts = lda.retrieve_original_text(corpus, id2word, texts)
>>> original_texts.head()
```

LatentDirichletAllocation.specify_adhoc_stopwords

classmethod LatentDirichletAllocation.**specify_adhoc_stopwords**()

Create a set of ad-hoc stopwords.

Returns

Ad-hoc stopwords.

Return type

set

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', 'robotic', load_preprocd_data=False)
>>> rslt = lda.specify_adhoc_stopwords()
>>> isinstance(rslt, set)
True
>>> # Traditional vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', 'traditional', load_preprocd_data=False)
>>> rslt = lda.specify_adhoc_stopwords()
>>> isinstance(rslt, set)
True
```

LatentDirichletAllocation.train_model

LatentDirichletAllocation.**train_model**(*corpus, id2word, texts, num_topics, alpha='asymmetric', eta='symmetric', **kwargs*)

Train an LDA model.

Parameters

- **corpus** (*list*) – corpus (i.e. term-document frequency, see gensim.corpora.Dictionary.doc2bow())
- **id2word** (*gensim.corpora.Dictionary or list*) – id-word mapping dictionary (see gensim.corpora.Dictionary())
- **texts** (*list*) – lemmatized review texts
- **num_topics** (*int*) – number of topics, see `num_topics` of gensim.models.LdaMulticore()
- **alpha** (*str or float or numpy.ndarray or list*) – alpha of gensim.models.LdaMulticore(), defaults to `'asymmetric'`

- **eta** (*str or float or numpy.ndarray or list or None*) – eta of gensim.models.LdaMulticore(), defaults to 'symmetric'
- **kwargs** – [optional] parameters of gensim.models.LdaMulticore() or gensim.models.LdaModel()

Returns
a collection of results, including an LDA model, a coherence model and coherence score

Return type
dict

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> # Traditional vacuum cleaners
>>> # lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> example_docs = lda.data['review_text']
>>> neg_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='negative')
>>> # Consider bi-grams
>>> neg_corpus, neg_id2word, neg_texts = lda.make_corpus(neg_tokenized_docs, ngram=2)
>>> # Consider three topics
>>> neg_results = lda.train_model(neg_corpus, neg_id2word, neg_texts, num_topics=3)
>>> isinstance(neg_results, dict)
True
>>> len(neg_results) == 3
True
```

LatentDirichletAllocation.train_models

`LatentDirichletAllocation.`**`train_models`**(*corpus, id2word, texts, num_topics_min=2, num_topics_max=6, alpha='asymmetric', eta='symmetric', verbose=False, **kwargs*)

Train a number of LDA models.

Parameters

- **corpus** (*list*) – corpus (i.e. term-document frequency, see gensim.corpora.Dictionary.doc2bow())
- **id2word** (*gensim.corpora.Dictionary*) – id-word mapping dictionary (see gensim.corpora.Dictionary())
- **texts** (*list*) – lemmatized review texts
- **num_topics_min** (*int*) – number of topics ranging from, defaults to 2
- **num_topics_max** (*int*) – number of topics up to, defaults to 6
- **alpha** (*float or numpy.ndarray or list*) – alpha of gensim.models.LdaMulticore(), defaults to 'auto'
- **eta** (*float or numpy.ndarray or list*) – eta of gensim.models.LdaMulticore(), defaults to 'asymmetric'

- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **kwargs** – [optional] parameters of gensim.models.LdaMulticore() or gensim.models.LdaModel()

Returns
a collection of results, including an LDA model, a coherence model and coherence score, for each given number of topics

Return type
dict

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> # Traditional vacuum cleaners
>>> # lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> # Smart thermostats
>>> # lda = LatentDirichletAllocation('thermostats', product_type='smart')
>>> example_docs = lda.data['review_text']
>>> pos_tokenized_docs = lda.get_tokenized_docs(example_docs, sentiment='positive')
>>> # Consider tri-grams
>>> pos_corpus, pos_id2word, pos_texts = lda.make_corpus(pos_tokenized_docs, ngram=3)
>>> pos_results = lda.train_models(
...     pos_corpus, pos_id2word, pos_texts, num_topics_max=3, verbose=True)
Coherence scores:
    2 topics: 0.6053
    3 topics: 0.5918
>>> isinstance(pos_results, dict)
True
```

LatentDirichletAllocation.view_evaluation_summary

`LatentDirichletAllocation.`**`view_evaluation_summary`**(*sentiment*, *partially=None*, *save_as=None*, *verbose=False*, ***kwargs*)

Visualise the results of the evaluation summary.

Parameters

- **sentiment** (*str*) – Label of sentiment; options include VALID_SENTIMENT_LABELS.
- **partially** (*None | dict*) – View the evaluation summary based a selected set of hyperparameters (particularly when the numbers of some hyperparameters are large); defaults to None.
- **save_as** (*str | bool | None*) – Extension of figure filename, or whether to save the figure; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information in console; defaults to False.
- **kwargs** – [Optional] additional parameters of pyhelpers.store.save_figure.

Examples:

```
>>> from src.modeller import LatentDirichletAllocation
>>> from pyhelpers.settings import mpl_preferences
>>> mpl_preferences(backend='TkAgg')
>>> # Robotic vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='robotic')
>>> # lda.view_evaluation_summary(sentiment='positive', save_as=".svg", verbose=True)
>>> lda.view_evaluation_summary(sentiment='positive')
```

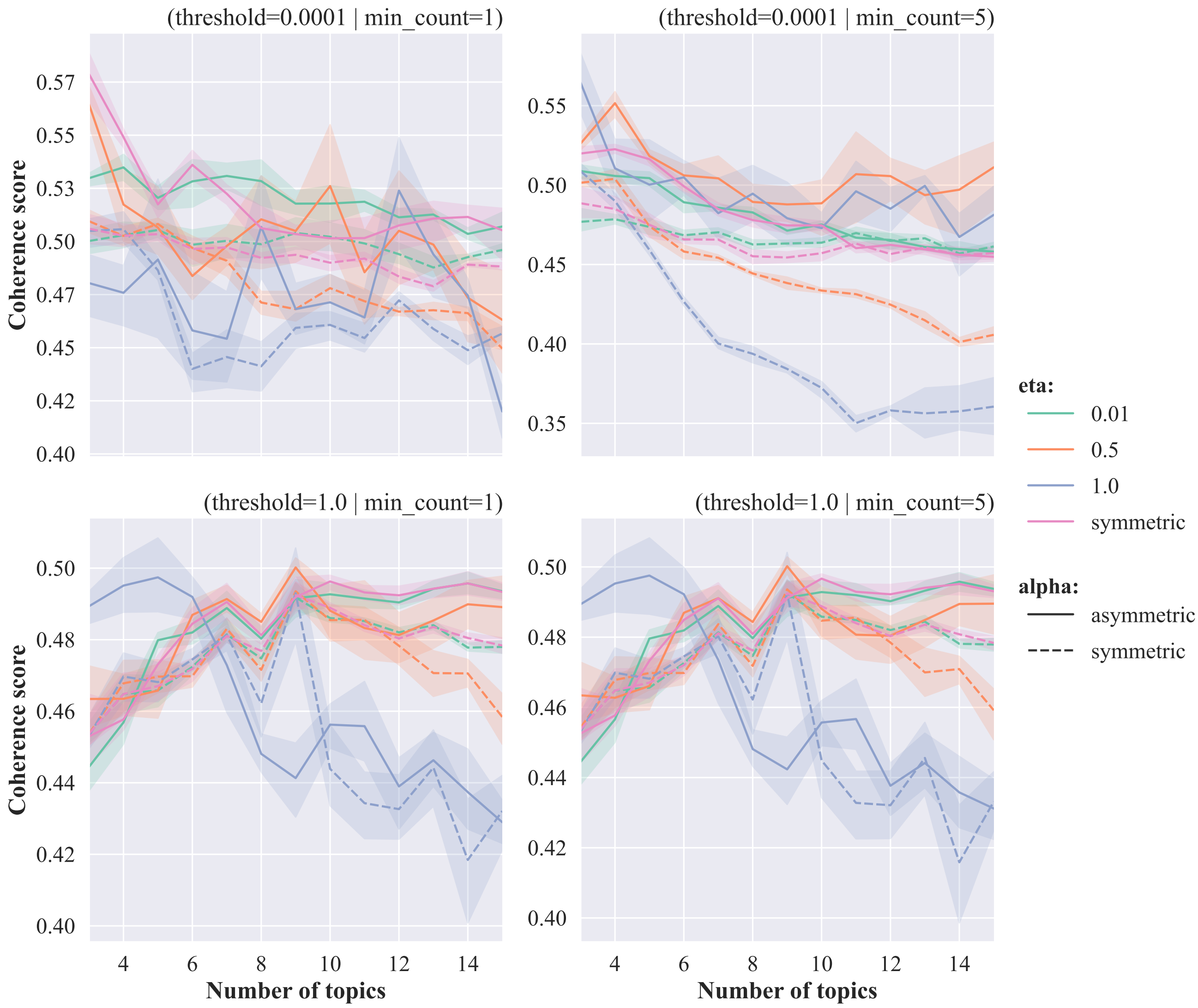

Fig. 14: LDA modeling trials for positive reviews on robotic vacuum cleaners.

```
>>> # lda.view_evaluation_summary(sentiment='negative', save_as=".svg", verbose=True)
>>> lda.view_evaluation_summary(sentiment='negative')
```

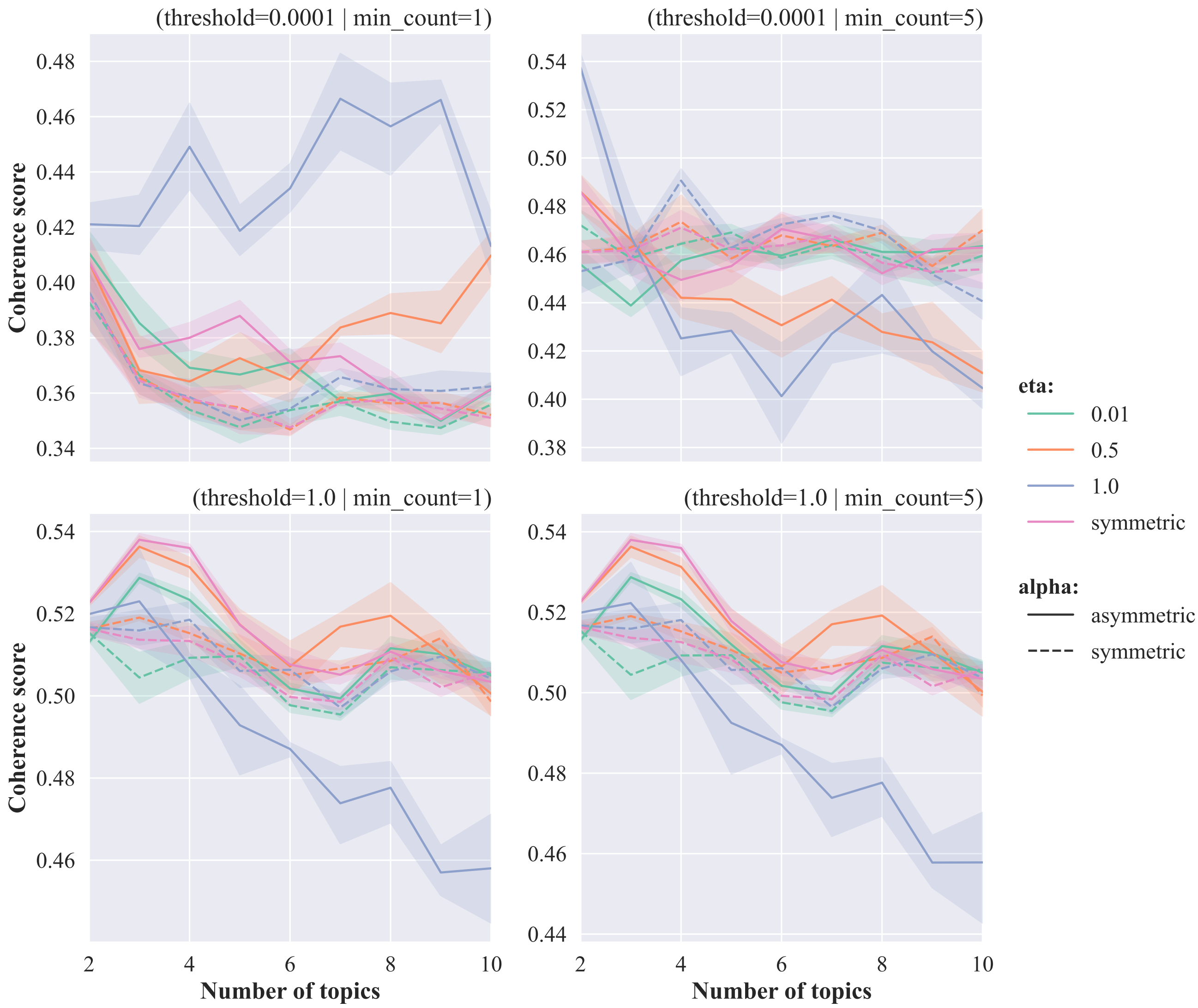

Fig. 15: LDA modeling trials for negative reviews on robotic vacuum cleaners.

```
>>> # Traditional vacuum cleaners
>>> lda = LatentDirichletAllocation('vacuum', product_type='traditional')
>>> # lda.view_evaluation_summary(sentiment='positive', save_as=".svg", verbose=True)
>>> lda.view_evaluation_summary(sentiment='positive')
```

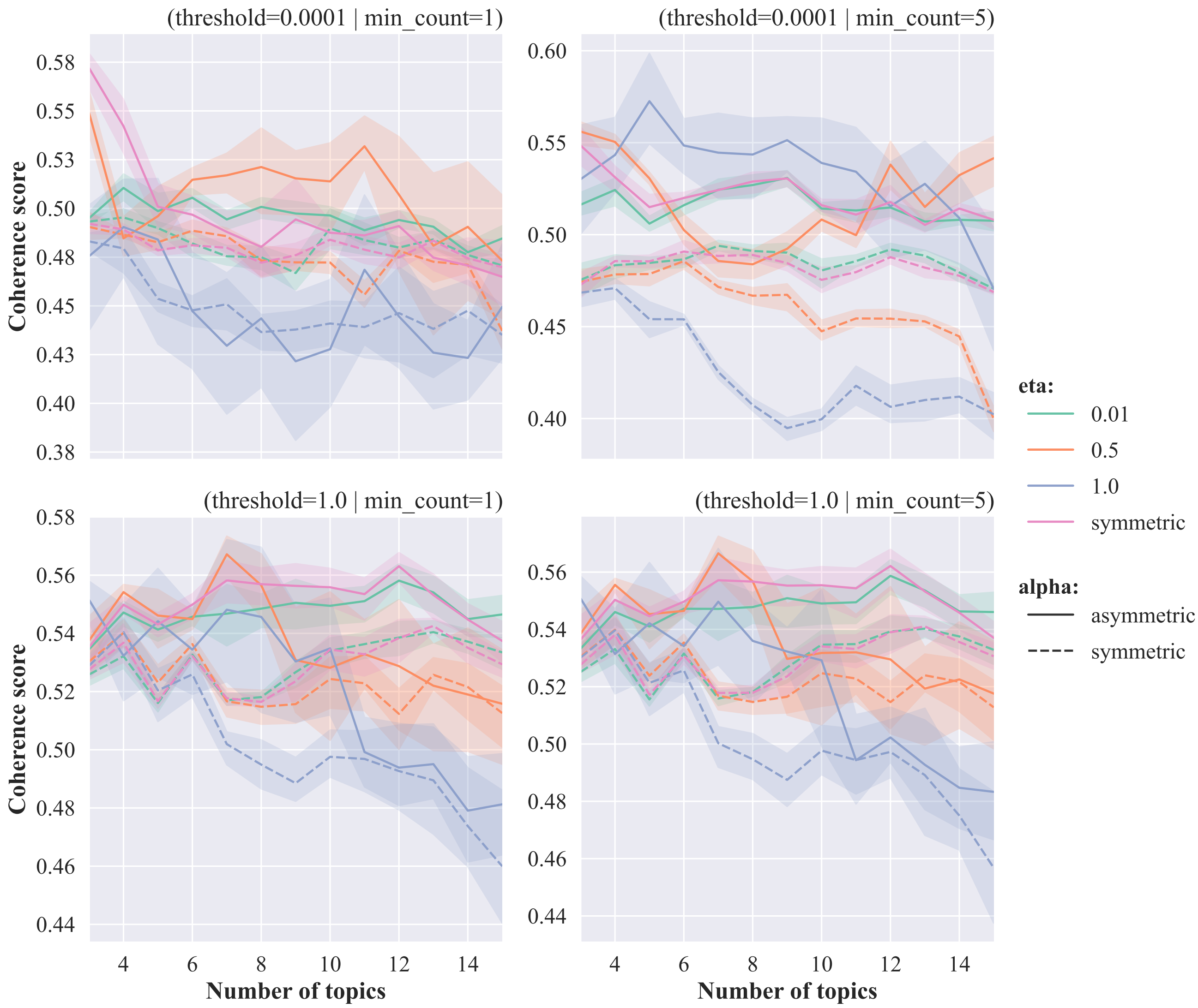

Fig. 16: LDA modeling trials for positive reviews on traditional vacuum cleaners.

```
>>> # lda.view_evaluation_summary(sentiment='negative', save_as=".svg", verbose=True)
>>> lda.view_evaluation_summary(sentiment='negative')
```

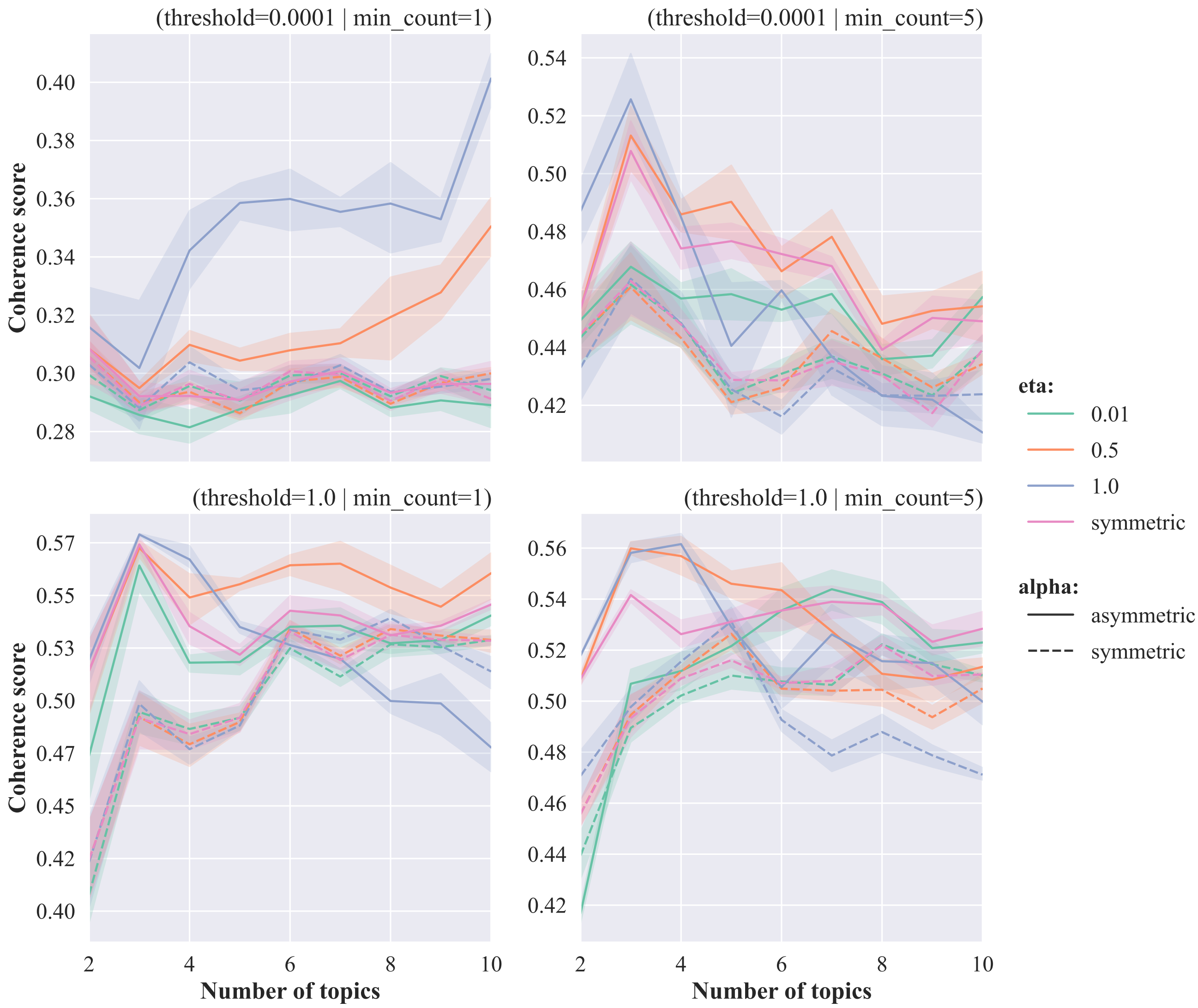

Fig. 17: LDA modeling trials for negative reviews on traditional vacuum cleaners.

```
>>> # Smart thermostats
>>> lda = LatentDirichletAllocation('therms', product_type='smart')
>>> partially = {'min_count': (1, 5), 'threshold': (0.0001, 1)}
>>> # lda.view_evaluation_summary(
... #     sentiment='positive', partially=partially, save_as=".svg", verbose=True)
>>> lda.view_evaluation_summary(sentiment='positive', partially=partially)
```

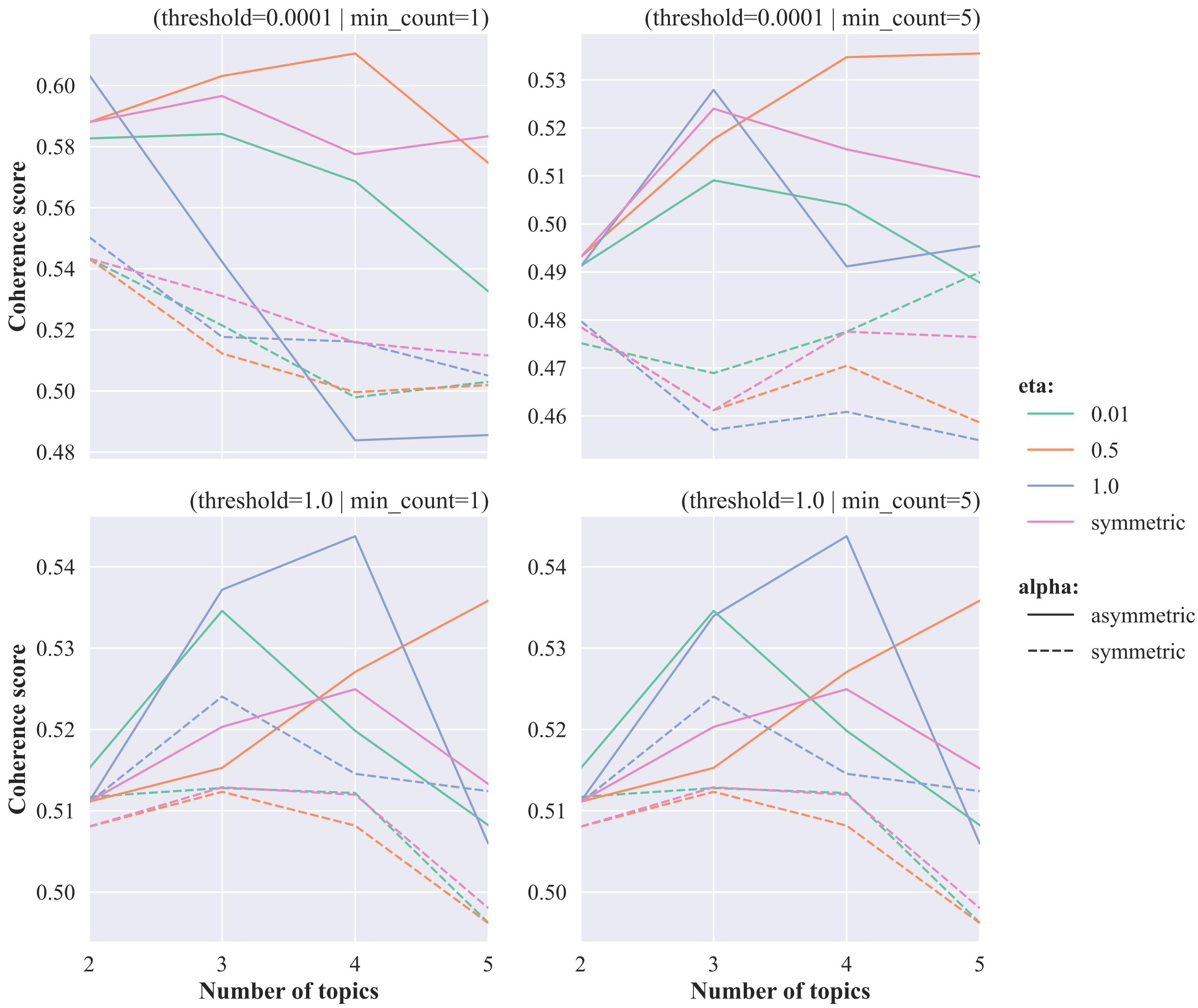

Fig. 18: LDA modeling trials for positive reviews on smart thermostats.

```
>>> # lda.view_evaluation_summary(
... #     sentiment='negative', partially=partially, save_as=".svg", verbose=True)
>>> lda.view_evaluation_summary(sentiment='negative', partially=partially)
```

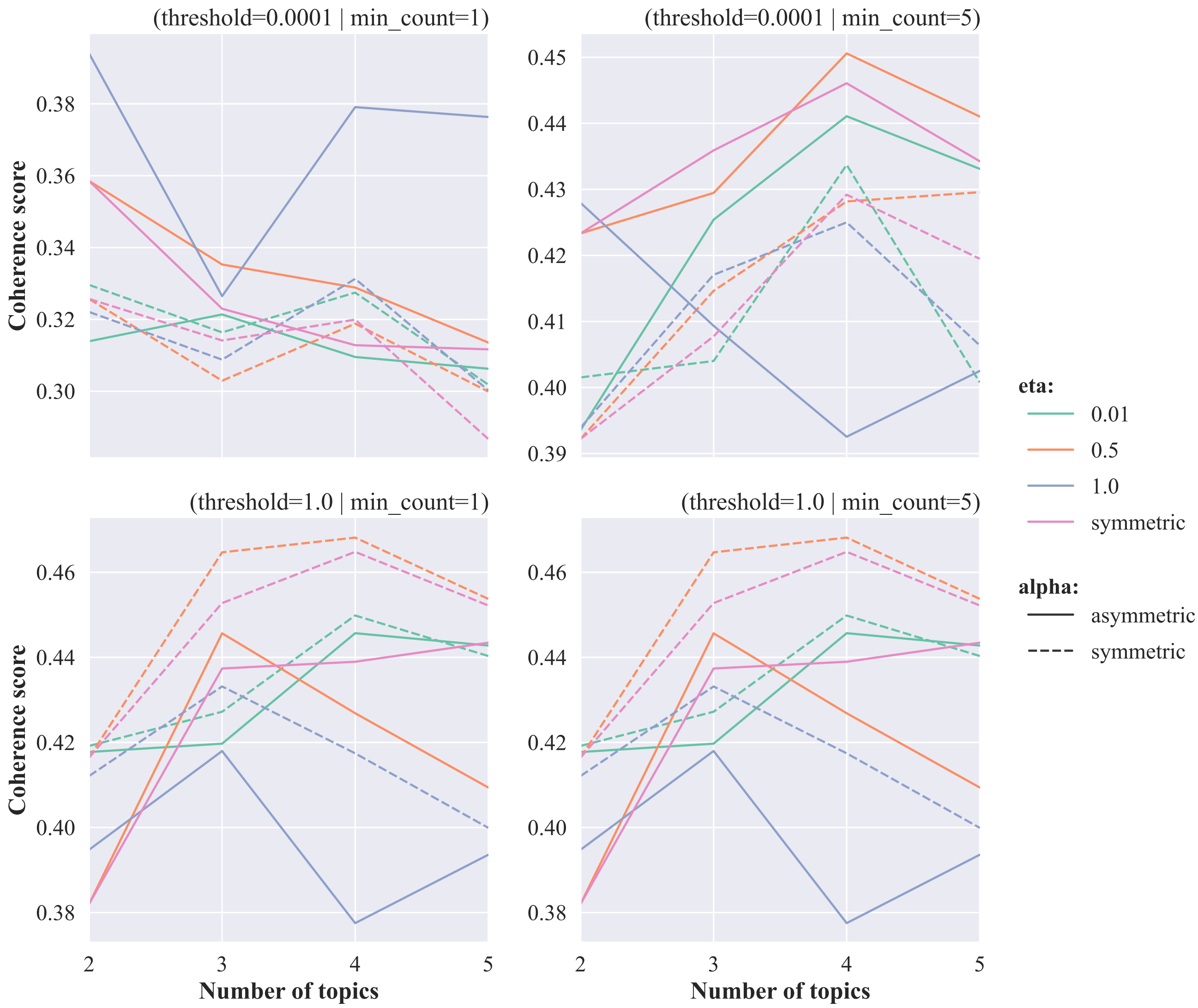

Fig. 19: LDA modeling trials for negative reviews on smart thermostats.

4.4 analyser

The module is used to analyse the modelling results produced by `modeller`.

get_doc_topic_probs(lda, sentiment, ...[, ...])	Get topic probabilities for each review.
calibrate_fuzzy_membership(prob, ...)	Calibrates the probability into fuzzy-set membership score.
prep_fsqca_data(doc_topic_probs, lda, sentiment)	Prepare data sets for fsQCA.

4.4.1 get_doc_topic_probs

`src.analyser.get_doc_topic_probs`(*lda*, *sentiment*, *best_lda_index*, *incl_original_reviews=True*, *further_screen=True*)

Get topic probabilities for each review.

Parameters

- **lda** (LatentDirichletAllocation) – An instance of the *~src.modeller.lda.LatentDirichletAllocation* class.
- **sentiment** (*str*) – Sentiment label.
- **best_lda_index**
- **incl_original_reviews** (*bool*) – Whether to include original review texts; defaults to `True`.
- **further_screen** (*bool*) – Whether to further screen the data of documents for fsQCA; defaults to `True`. This process aims to ensure that vader scores (calculated based on the original review texts) also reflect the true sentiment of the corresponding cleansed tokens in the returned data set.

Returns

Data of topic probabilities for each review.

Examples:

```
>>> from src.analyser import get_doc_topic_probs
>>> from src.modeller import LatentDirichletAllocation
>>> lda = LatentDirichletAllocation(product_category='vacuum', product_type='robotic')
>>> doc_topic_probs = get_doc_topic_probs(lda, sentiment='positive', best_lda_index=76)
>>> doc_topic_probs.head()
    topic_1   topic_2  ...  vs_compound_score new_vs_compound_score
0  0.968450  0.019314  ...             0.9781                0.7430
1  0.918801  0.048385  ...             0.9591                0.4019
2  0.922607  0.045102  ...             0.3971                0.5719
3  0.961893  0.022935  ...             0.9754                0.4939
4  0.971785  0.017273  ...             0.7787                0.6249
[5 rows x 7 columns]
>>> doc_topic_probs = get_doc_topic_probs(lda, sentiment='negative', best_lda_index=0)
>>> doc_topic_probs.head()
   topic_1   topic_2  ...  vs_compound_score new_vs_compound_score
0      0.0  0.356828  ...            -0.4359               -0.5719
1      0.0  0.116166  ...            -0.4012               -0.3182
2      0.0  0.978022  ...            -0.4779               -0.5719
3      0.0  0.981127  ...            -0.7944               -0.2960
4      0.0  0.983219  ...            -0.9284               -0.5574
[5 rows x 7 columns]
>>> lda = LatentDirichletAllocation(product_category='vacuum', product_type='traditional')
>>> doc_topic_probs = get_doc_topic_probs(lda, sentiment='positive', best_lda_index=9)
>>> doc_topic_probs.head()
    topic_1   topic_2  ...  vs_compound_score new_vs_compound_score
0  0.713384  0.283967  ...             0.7389                0.5859
1  0.396577  0.601326  ...             0.9002                0.7264
2  0.529634  0.468892  ...             0.8805                0.5106
3  0.544901  0.451072  ...             0.6908                0.6908
```

(continues on next page)

(continued from previous page)

```
4  0.450820  0.546059  ...             0.6597                0.0772
[5 rows x 7 columns]
>>> doc_topic_probs = get_doc_topic_probs(lda, sentiment='negative', best_lda_index=49)
>>> doc_topic_probs.head()
    topic_1   topic_2  ...  vs_compound_score new_vs_compound_score
0  0.271696  0.081394  ...            -0.1923               -0.7269
1  0.025579  0.635595  ...            -0.6590               -0.4927
2  0.017298  0.529355  ...            -0.8078               -0.5574
3  0.260156  0.219738  ...            -0.8118               -0.5994
4  0.214129  0.248306  ...            -0.8436               -0.4536
[5 rows x 7 columns]
>>> lda = LatentDirichletAllocation(product_category='thermostats', product_type='smart')
>>> doc_topic_probs = get_doc_topic_probs(lda, sentiment='positive', best_lda_index=19)
>>> doc_topic_probs.head()
    topic_1   topic_2  ...  vs_compound_score new_vs_compound_score
0  0.206133  0.000000  ...             0.9902                0.7269
1  0.949537  0.029732  ...             0.8849                0.3612
2  0.960210  0.023300  ...             0.8915                0.0516
3  0.052526  0.928535  ...             0.9692                0.4019
4  0.974451  0.015015  ...             0.7351                0.5106
[5 rows x 7 columns]
>>> doc_topic_probs = get_doc_topic_probs(lda, sentiment='negative', best_lda_index=9)
>>> doc_topic_probs.head()
    topic_1   topic_2  ...  vs_compound_score new_vs_compound_score
0  0.919857  0.039846  ...            -0.6863               -0.4767
1  0.000000  0.000000  ...            -0.7975               -0.5423
2  0.967758  0.015943  ...            -0.9205               -0.3612
3  0.922046  0.039310  ...            -0.1280               -0.1280
4  0.096724  0.531999  ...            -0.4310               -0.3612
[5 rows x 7 columns]
```

4.4.2 calibrate_fuzzy_membership

src.analyser.**calibrate_fuzzy_membership**(*prob*, *full_non_membership*, *crossover*, *full_membership*)

Calibrates the probability into fuzzy-set membership score.

Parameters

- **prob** (*int | float*) – The topic probability to calibrate (between 0 and 1)
- **full_non_membership** (*int | float*) – The threshold for full non-membership (calibrated to 0)
- **crossover** (*int | float*) – The crossover point where membership is 0.5
- **full_membership** (*int | float*) – The threshold for full membership (calibrated to 1)

Returns

Fuzzy-set membership score (between 0 and 1)

Return type

float

Examples:

```
>>> from src.analyser import calibrate_fuzzy_membership, get_doc_topic_probs
>>> from src.modeller import LatentDirichletAllocation
>>> import numpy as np
>>> lda = LatentDirichletAllocation(product_category='vacuum', product_type='robotic')
>>> doc_topic_probs = get_doc_topic_probs(lda, sentiment='positive', best_lda_index=76)
>>> data = doc_topic_probs.copy()
>>> # topic_col_names = [x for x in data.columns if x.startswith('topic_')]
>>> topic_col_names = ['topic_1', 'topic_2', 'topic_3']
>>> for col in topic_col_names + ['rating', 'vs_compound_score', 'new_vs_compound_score']:
...     f_n_membership, crossover_point, f_membership = map(
...         lambda x: np.percentile(data[col].values, x), [5, 50, 95])
...     data[col + '_fuzzy'] = data[col].map(
...         lambda prob: calibrate_fuzzy_membership(
...             prob, f_n_membership, crossover_point, f_membership)).values
>>> data.columns.tolist()
['topic_1',
 'topic_2',
 'topic_3',
 'ReviewText',
 'rating',
 'vs_compound_score',
 'new_vs_compound_score',
 'topic_1_fuzzy',
 'topic_2_fuzzy',
 'topic_3_fuzzy',
 'rating_fuzzy',
 'vs_compound_score_fuzzy',
 'new_vs_compound_score_fuzzy']
```

4.4.3 prep_fsqca_data

src.analyser.**prep_fsqca_data**(*doc_topic_probs*, *lda*, *sentiment*, *save_res=True*)

Prepare data sets for fsQCA.

Parameters

- **doc_topic_probs** (*pandas.DataFrame*) – Data of topic probabilities for each review.
- **lda** (LatentDirichletAllocation) – An instance of the *~src.modeller.lda.LatentDirichletAllocation* class.
- **sentiment** (*str*) – Sentiment label.
- **save_res** (*bool*) – Whether to save the prepared data sets as CSV files; defaults to True.

Returns

Data sets ready for fsQCA.

Return type

collections.OrderedDict

Examples:

```
>>> from src.analyser import prep_fsqca_data, get_doc_topic_probs
>>> from src.modeller import LatentDirichletAllocation
```

(continues on next page)

(continued from previous page)

```
>>> lda = LatentDirichletAllocation(product_category='vacuum', product_type='robotic')
>>> sentiment = 'positive'
>>> best_lda_index = 76
>>> doc_topic_probs = get_doc_topic_probs(lda, sentiment, best_lda_index)
>>> doc_topic_probs.head()
    topic_1   topic_2  ...  vs_compound_score new_vs_compound_score
0  0.968435  0.019337  ...             0.9781                0.7430
1  0.918798  0.048387  ...             0.9591                0.4019
2  0.922610  0.045099  ...             0.3971                0.5719
3  0.961892  0.022936  ...             0.9754                0.4939
4  0.971793  0.017264  ...             0.7787                0.6249
[5 rows x 7 columns]
>>> fsqca_prep_data = prep_fsqca_data(doc_topic_probs, lda, sentiment, save_res=False)
>>> list(fsqca_prep_data.keys())
['robotic-positive-doc_topic_probs_5_10_25',
 'robotic-positive-doc_topic_probs_5_25_50',
 'robotic-positive-doc_topic_probs_5_50_95']
```

Chapter 5

License

The contents of this repository are licensed under the MIT License.

Chapter 6

Publications

- Yu, Y., Fu, Q., Zhang, D., & Gu, Q. (2024). Understanding user experience with smart home products. Journal of Computer Information Systems, 1–23. doi:10.1080/08874417.2024.2408006.
- Yu, Y., Fu, Q., Zhang, D., & Gu, Q. (2024). What are smart home product users commenting on? A case study of robotic vacuums. In: Han, H., Baker, E. (eds) Next Generation Data Science. SDSC 2023. Communications in Computer and Information Science, vol 2113. Springer, Cham. doi:10.1007/978-3-031-61816-1_3.

Python Module Index

S

Index

P

R

S

T

V