

# Practical 7: Working with Text

The basics of Text Mining and NLP

## Table of contents

1	Preamble	2
2	Setup	2
3	Exploratory Textual Analysis	7
4	Using Regular Expressions	9
5	Text Cleaning	18
6	Pandas.Apply	24
7	Data Normalisation	26
8	Word Clouds	36

Complete	Part 1: Foundations	Part 2: Data	Part 3: Analysis	
60%	██████████	██████	██████	7/10

A lot of the content here is provided to help you *understand* what text-cleaning does and how it generates tokens that can be processed by the various analytical approaches commonly-used in NLP. The best way to think about this is as a practical in three parts, not all of which you should expect to complete in this session:

1. Tasks 1–3: these are largely focussed on the basics: exploring text and using regular expressions to find and select text.
2. Tasks 4–5: this might seem like a *bit* of a detour, but it’s intended to show you in a more tangible way how ‘normalisation’ works when we’re working with text. **You should feel free to stop here and return to the rest later.**
3. Tasks 6–7: are about finding important vocabulary (think ‘keywords’ and ‘significant terms’) in documents so that you can start to think about what is *distinctive* about documents and groups of documents. **This is quite useful and relatively easier to understand than what comes next!**
4. Tasks 8–9: are about fully-fledged NLP using Latent Dirichlet Allocation (topic modelling) and Word2Vec (words embeddings for use in clustering or similarity work).

The later parts are largely complete and ready to run; however, that *doesn't* mean you should just skip over them and think you've grasped what's happening and it will be easy to apply in your own analyses. I would *not* pay as much attention to LDA topic mining since I don't think it's results are that good, but I've included it here as it's still commonly-used in the Digital Humanities and by Marketing folks. Word2Vec is much more powerful and forms the basis of the kinds of advances seen in ChatGPT and other LLMs.

#### Connections

Working with text is unquestionably *hard*. In fact, *conceptually* this is probably the most challenging practical of the term! But data scientists are *always* dealing with text because so much of the data that we collect (even more so thanks to the web) is not only text-based (URLs are text!) but, increasingly, unstructured (social media posts, tags, etc.). So while getting to grips with text is a challenge, it also uniquely positions you with respect to the skills and knowledge that other graduates are offering to employers.

## 1 Preamble

This practical has been written using nltk, but would be *relatively* easy to rework using spacy. Most programmers tend to use one *or* the other, and the switch wouldn't be hard other than having to first load the requisite language models:

```
import spacy

# `...web_md` and `...web_lg` are also options
corp = "en_core_web_sm"

try:
    nlp = spacy.load(corp)
except OSError:
    spacy.cli.download(corp)
    nlp = spacy.load(corp)
```

You can [read about the models](#), and note that they are also [available in other languages](#) besides English.

## 2 Setup

#### Difficulty Level: Low

But this is only because this has been worked out for you. Starting from scratch in NLP is *hard* so people try to avoid it as much as possible.

## 2.1 Required Modules

### Note

Notice that the number of modules and functions that we import is steadily increasing week-on-week, and that for text processing we tend to draw on quite a wide range of utilities! That said, the three most commonly used are: `sklearn`, `nlTK`, and `spacy`.

Standard libraries we've seen before.

```
import os
import numpy as np
import pandas as pd
import geopandas as gpd
import re
import math
import matplotlib.pyplot as plt
```

Vectorisers we will use from the 'big beast' of Python machine learning: Sci-Kit Learn.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
```

```
# We don't use this but I point out where you *could*
from sklearn.preprocessing import OneHotEncoder
```

NLP-specific libraries that we will use for tokenisation, lemmatisation, and frequency analysis.

```
import nltk
import spacy
from nltk.corpus import wordnet as wn
from nltk.stem.wordnet import WordNetLemmatizer
```

```
try:
    from nltk.corpus import stopwords
except:
    nltk.download('stopwords')
    from nltk.corpus import stopwords
stopword_list = set(stopwords.words('english'))
```

```
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.tokenize.toktok import ToktokTokenizer
```

```
from nltk.stem.porter import PorterStemmer
from nltk.stem.snowball import SnowballStemmer
```

```
from nltk import ngrams, FreqDist
```

```
lemmatizer = WordNetLemmatizer()
tokenizer = ToktokTokenizer()
```

Remaining libraries that we'll use for processing and display text data. Most of this relates to dealing with the various ways that text data cleaning is *hard* because of the myriad formats it comes in.

```
import string
import unicodedata
from bs4 import BeautifulSoup
from wordcloud import WordCloud, STOPWORDS
```

This next is just a small utility function that allows us to output Markdown (like this cell) instead of plain text:

```
from IPython.display import display_markdown

def as_markdown(head="", body='Some body text'):
    if head != "":
        display_markdown(f"##### {head}\n\n>{body}\n", raw=True)
    else:
        display_markdown(f">{body}\n", raw=True)

as_markdown('Result!', "Here's my output...")
```

#### 2.1.0.0.1 Result!

Here's my output...

## 2.2 Loading Data

### Connections

Because I generally want each practical to stand on its own (unless I'm trying to make a *point*), I've not moved this to a separate Python file (e.g. `utils.py`, but in line with what we covered back in the lectures on [Functions and Packages](#), this sort of thing is a good candidate for being split out to a separate file to simplify re-use.

Remember this function from last week? We use it to save downloading files that we already have stored locally. But notice I've made some small changes... what do these do to help the user?

```
import os
from requests import get
from urllib.parse import urlparse
```

```
def cache_data(src:str, dest:str) -> str:
    """Downloads and caches a remote file locally.
```

The function sits between the 'read' step of a pandas or geopandas data frame and downloading the file from a remote location. The idea is that it will save it locally so that you don't need to remember to

do so yourself. Subsequent re-reads of the file will return instantly rather than downloading the entire file for a second or n-th time.

#### Parameters

-----

src : str

The remote *\*source\** for the file, any valid URL should work.

dest : str

The *\*destination\** location to save the downloaded file.

#### Returns

-----

str

A string representing the local location of the file.

"""

```
url = urlparse(src) # We assume that this is some kind of valid URL
```

```
fn = os.path.split(url.path)[-1] # Extract the filename
```

```
dfn = os.path.join(dest,fn) # Destination filename
```

```
# Check if dest+filename does *not* exist --
```

```
# that would mean we have to download it! We
```

```
# also check for *very* small files that are
```

```
# likely to represent an incomplete download.
```

```
if not os.path.isfile(dfn) or os.stat(dfn).st_size < 250:
```

```
    print(f"{dfn} not found, downloading!")
```

```
# Convert the path back into a list (without)
```

```
# the filename -- we need to check that directories
```

```
# exist first.
```

```
path = os.path.split(dest)
```

```
# Create any missing directories in dest(ination) path
```

```
# -- os.path.join is the reverse of split (as you saw above)
```

```
# but it doesn't work with lists... so I had to google how
```

```
# to use the 'splat' operator! os.makedirs creates missing
```

```
# directories in a path automatically.
```

```
if len(path) >= 1 and path[0] != ":
```

```
    os.makedirs(os.path.join(*path), exist_ok=True)
```

```
# Download and write the file
```

```
with open(dfn, "wb") as file:
```

```
    response = get(src)
```

```
    file.write(response.content)
```

```
print("\tDone downloading...")
```

```
# What's this doing???
```

```
f_size = os.stat(dfn).st_size
```

```
print(f"\tSize is {f_size/1024**2:,.0f} MB ({f_size:,} bytes)")
```

```

else:
    print(f"Found {dfn} locally!")

    # And why is it here as well???
    f_size = os.stat(dfn).st_size
    print(f"\tSize is {f_size/1024**2:,.0f} MB ({f_size:,} bytes)")

return dfn

```

#### Tip

For very large *non-geographic* data sets, remember that you can use `use_cols` (or `columns` depending on the file type) to specify a subset of columns to load.

Load the main data set:

```

# Set download URL
ymd = '2023-09-06'
host = 'https://orca.casa.ucl.ac.uk'
url = f'{host}/~jreades/data/{ymd}-listings.geoparquet'

gdf = gpd.read_parquet( cache_data(url, os.path.join('data','geo')),
                       columns=['geometry', 'listing_url', 'name',
                               'description', 'amenities', 'price'])

gdf = gdf.to_crs('epsg:27700')

print(f"gdf has {gdf.shape[0]:,} rows and CRS is {gdf.crs.name}.")

```

```

Found data/geo/2023-09-06-listings.geoparquet locally!
  Size is 42 MB (44,152,609 bytes)
gdf has 85,134 rows and CRS is OSGB36 / British National Grid.

```

Load supporting Geopackages:

```

ddir = os.path.join('data','geo') # destination directory
spath = 'https://github.com/jreades/fsds/blob/master/data/src/' # source path

boros = gpd.read_file( cache_data(spath+'Boroughs.gpkg?raw=true', ddir) )
water = gpd.read_file( cache_data(spath+'Water.gpkg?raw=true', ddir) )
green = gpd.read_file( cache_data(spath+'Greenspace.gpkg?raw=true', ddir) )

print('Done.')

```

```

Found data/geo/Boroughs.gpkg locally!
  Size is 1 MB (905,216 bytes)
Found data/geo/Water.gpkg locally!
  Size is 0 MB (208,896 bytes)
Found data/geo/Greenspace.gpkg locally!
  Size is 1 MB (1,146,880 bytes)
Done.

```

### 3 Exploratory Textual Analysis

#### Connections

If you plan to work with data post-graduation then you will *need* to become comfortable with Regular Expressions (aka. regexes). These are the focus of the [Patterns in Text](#) lecture but they barely even scratch the surface of what regexes can do. They are *hard*, but they are powerful.

#### Tip


In a full text-mining application I would spend a lot more time on this stage: sampling, looking at descriptions in full, performing my analysis (the rest of the steps) and then coming back with a deeper understanding of the data to make further changes to the analysis.

It's helpful to have a sense of what data look like before trying to do something with them, but by default pandas truncates quite a lot of output to keep it from overwhelming the display. For text processing, however, you should probably change the amount of preview text provided by pandas using the available options. *Note*: there are lots of other options that you can tweak in pandas.

```
print(f"Default maximum column width: {pd.options.display.max_colwidth}") # What's this currently set to?
pd.options.display.max_colwidth=250 # None = no maximum column width (you probably don't want to leave it at
print(f"Now maximum column width set to: {pd.options.display.max_colwidth}")
```

```
Default maximum column width: 50
Now maximum column width set to: 250
```

#### 3.1 The Description Field

 Difficulty level: Moderate, because of the questions.

To explore the description field properly you'll need to filter out any NA/NaN descriptions before sampling the result. *Hint*: you'll need to think about negation (~) of a method output that tells you if a field is NA.

##### 3.1.0.1 Answer

```
gdf[~gdf.description.isna()].sample(5, random_state=42)[['description']]
```

description	
id	
21898360	A wonderful double room in harrow on the hill.  Very quite and secure
40269638	- 3 Bedrooms 2 beds and 1 sofa bed - 2 Bathroom - 0.5 miles Underground
610813289456549376	St George Wharf is a landmark riverside development spanning across 7 acres
917365608608238592	My place is in the Heart of London. It is only 15 mins to KINGS CROSS Underground

id	description
6319621	This artsy and cozy small 1 bedroom split level flat is 4 min. from Seven Sister

id	description
21898360	A wonderful double room in harrow on the hill.  Very quite and sec
40269638	- 3 Bedrooms 2 beds and 1 sofa bed - 2 Bathroom - 0.5 miles Underg
610813289456549376	St George Wharf is a landmark riverside development spanning across 7 acres
917365608608238592	My place is in the Heart of London. It is only 15 mins to KINGS CROSS Unde
6319621	This artsy and cozy small 1 bedroom split level flat is 4 min. from Seven Sister

### 🔥 Stop

What do you notice about the above? Are they simple text? Are there patterns of problems? Are there characters that represent things other than words and simple punctuation?

#### 3.1.1 Questions

- What patterns can you see that might need 'dealing with' for text-mining to work?
- What non-text characters can you see? (Things *other* than A-Z, a-z, and simple punctuation!)

#### 3.2 The Amenities Field

⚠️ Difficulty level: Moderate, because of the questions.

This field presents a subtle issue that might not be obvious here:

```
gdf.amenities.sample(5, random_state=42)
```

```
id
23851484
930201284366445056  ["Bathtub", "Hot water kettle", "Shampoo", "Luggage dropoff allowed", "Portable heater", "
19955089           ["Shampoo", "Dryer", "Microwave", "Coffee maker", "Hot water", "Iron", "First aid kit", "Washer", "
886165743387675008
13921898           ["Bathtub", "Indoor fireplace", "Shampoo", "Microwave", "Bed linens", "Hot water", "Iron", "Host g
Name: amenities, dtype: object
```

But look what happens now, can you see the issue a little more easily?

```
gdf.amenities.iloc[0]
```

```
['Heating', 'TV with standard cable', 'Wifi', 'Smoke alarm', 'Dryer', 'Kitchen', 'Washer', 'Essentials']
```



### 3.2.1 Questions

- What's the implicit format of the Amenities columns?
- How could you represent the data contained in the column?

### 3.3 Remove NaN Values

#### Note

I would be wary of doing the below in a 'proper' application without doing some careful research first, but to make our lives easier, we're going to drop rows where one of these values is NaN *now* so it will simplify the steps below. In reality, I would spend quite a bit more time investigating which values are NaN and why before simply dropping them.

Anyway, drop all rows where *either* the description or amenities (or both) are NA:

#### 3.3.0.1 Answer

```
gdf = gdf.dropna(subset=['description','amenities'])  
print(f"Now gdf has {gdf.shape[0]:,} rows.")
```

Now gdf has 84,273 rows.

You should get that there are 84,273 rows.


## 4 Using Regular Expressions

#### Connections

We're building on the work done in [Practical 6](#), but making use now of the lecture on [Patterns in Text](#) to quickly sort through the listings.

There is a *lot* that can be done with Regular Expressions to identify relevant records in textual data and we're going to use this as a starting point for the rest of the analysis. I would normally consider the regexes here a 'first pass' at the data, but would look very carefully at the output of the TF/IDF vectorizer, Count vectorizer, and LDA to see if I could improve my regexes for further cycles of analysis... the main gain there is that regexes are *much* faster than using the full NLP (Natural Language Processing) pipeline on the *full* data set each time. As an alternative, you could develop the pipeline using a random subsample of the data and then process the remaining records sequentially – in this context there is no justification for doing that, but with a larger corpus it might make sense.

## 4.1 Luxury Listings

 Difficulty level: Hard, because of the regular expression and questions.

I would like you to find listings that *might* (on the basis of word choice) indicate ‘luxury’ accommodation.

### 4.1.1 Create the Regular Expression

You should start with variations on ‘luxury’ (i.e. luxurious, luxuriate, ...) and work out a **single regular expression** that works for variations on this *one* word. **Later**, I would encourage you to come back to this and consider what other words might help to signal ‘luxury’... perhaps words like ‘stunning’ or ‘prestigious’? Could you add those to the regex as well?

*Hints:* this is a toughy, but...

1. All regular expressions work best using the `r'...'` (which means raw string) syntax.
2. You need to be able to *group* terms. Recall, however, that in Python a ‘group’ of the form `r'(some text)'` refers to matching (some text will be ‘memoized’/remembered), whereas what you need here is a “non-capturing group” of the **positive lookahead** type. That’s a Google clue right there, but you’ve also seen this in the lecture.

In fact, in my real-world applications you might even need more than one group/non-capturing group in a *nested* structure.

#### 4.1.1.1 Answer

```
gdf[
    gdf.description.str.contains(r'(?!luxur(?:y|ious|iat)|prestigious|stunning)', regex=True, flags=re.IGNORECASE)
].sample(3, random_state=42)[['description']]
```

description	
id	
21052595	NEW 4 BEDROOM HOUSE WITH PRIVATE GARDEN IN WEST HAMPST
3798279	Beautiful home in Kingston upon Thames. Central location - short walk to th
821457432530779648	Stunning spacious one bedroom apartment in one of London's most prime lo

description	
id	
21052595	NEW 4 BEDROOM HOUSE WITH PRIVATE GARDEN IN WEST HAMPST
3798279	Beautiful home in Kingston upon Thames. Central location - short walk to th
821457432530779648	Stunning spacious one bedroom apartment in one of London's most prime lo

### 4.1.2 Apply it to Select Data

Assign it to a new data frame called lux:

#### 4.1.2.1 Answer

```
lux = gdf[gdf.description.str.contains(r'(?::luxur(?:y|ious|iat)|prestigious|stunning)', regex=True, flags=re.IGNORECASE)]  
print(f"Found {lux.shape[0]:,} records for 'luxury' flats")
```

Found 10,368 records for 'luxury' flats

You should get 10,368 rows.

#### 4.1.3 Plot the Data

Now we are going to create a more complex plot that will give space to both the spatial and price distributions using subplot2grid.

```
help(plt.subplot2grid)
```

Notice that there are two ways to create the plot specified above. I chose route 1, but in some ways route 2 (where you specify a gridspec object and *then* add the axes might be a bit simpler to work out if you're starting from scratch.

The critical thing here is to understand how we're initialising a plot that has **4 rows** and **1 column** even though it is only showing **2 plots**. What we're going to do is set the *first* plot to span **3 rows** so that it takes up 75% of the plot area (3/4), while the *second* plot only takes up 25% (1/4). They will appear one above the other, so there's only 1 column. Here's how to read the key parts of subplot2grid:

- `nrows` – how many rows of *plots* in the figure.
- `ncols` – how many columns of *plots* in the figure.
- `row` – what row of the figure does *this* plot start on (0-indexed like a list in Python).
- `col` – what column of the figure does *this* plot start on (0-indexed like a list in Python).
- `rowspan` – how many rows of the figure does *this* plot span (*not* 0-indexed because it's not list-like).
- `colspan` – how many columns of the figure does *this* plot span (*not* 0-indexed because it's not list-like).

Every time you call subplot2grid you are initialising a new axis-object into which you can then draw with your geopackage or pandas plotting methods.

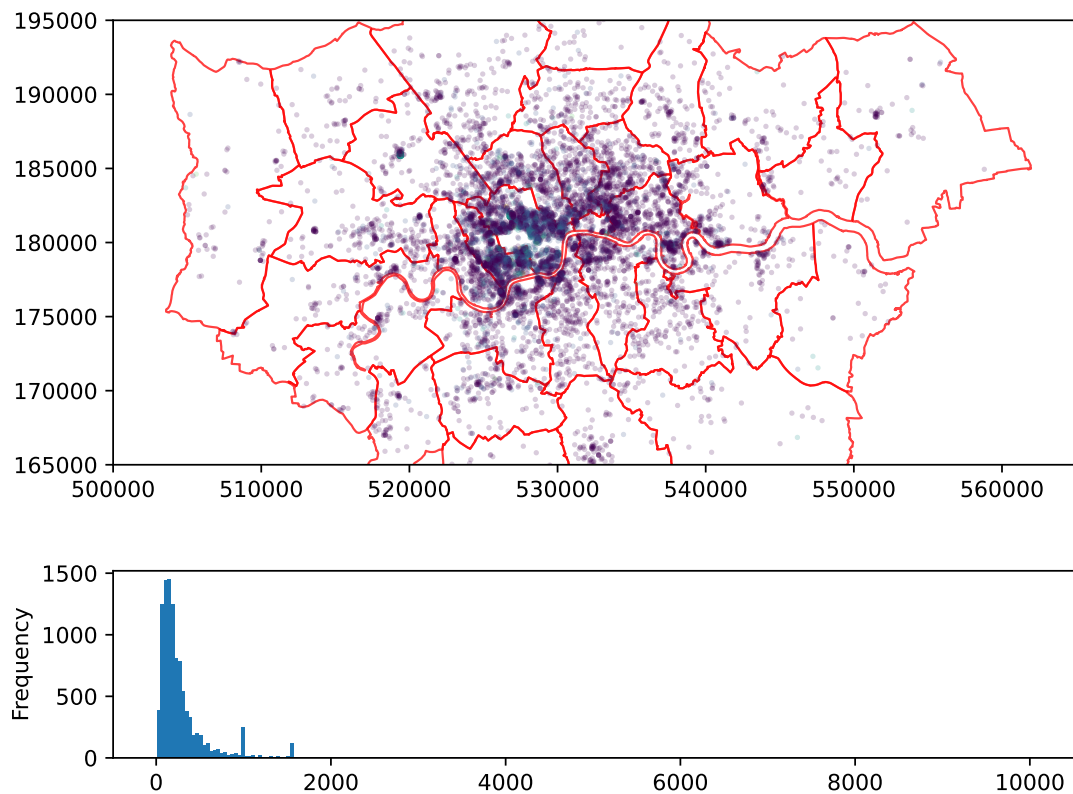
#### 4.1.3.1 Answer

```
f,ax = plt.subplots(1,1,figsize=(7,6))  
ax.remove()  
  
# The first plot  
ax1 = plt.subplot2grid((4, 1), (0, 0), rowspan=3)  
boros.plot(edgecolor='red', facecolor='none', linewidth=1, alpha=0.75, ax=ax1)  
lux.plot(markersize=2, column='price', cmap='viridis', alpha=0.2, scheme='Fisher_Jenks_Sampled', ax=ax1)  
  
ax1.set_xlim([500000, 565000])  
ax1.set_ylim([165000, 195000]);
```

```
# The second plot
ax2 = plt.subplot2grid((4, 1), (3, 0), rowspan=1)
lux.price.plot.hist(bins=250, ax=ax2)

plt.suptitle("Listings Advertising Luxury") # <-- How does this differ from title? Change it and see!
plt.tight_layout() # <-- Try creating the plot *without* this to see what it changes
```

### Listings Advertising Luxury



Your result should look similar to:

## Listings Advertising Luxury

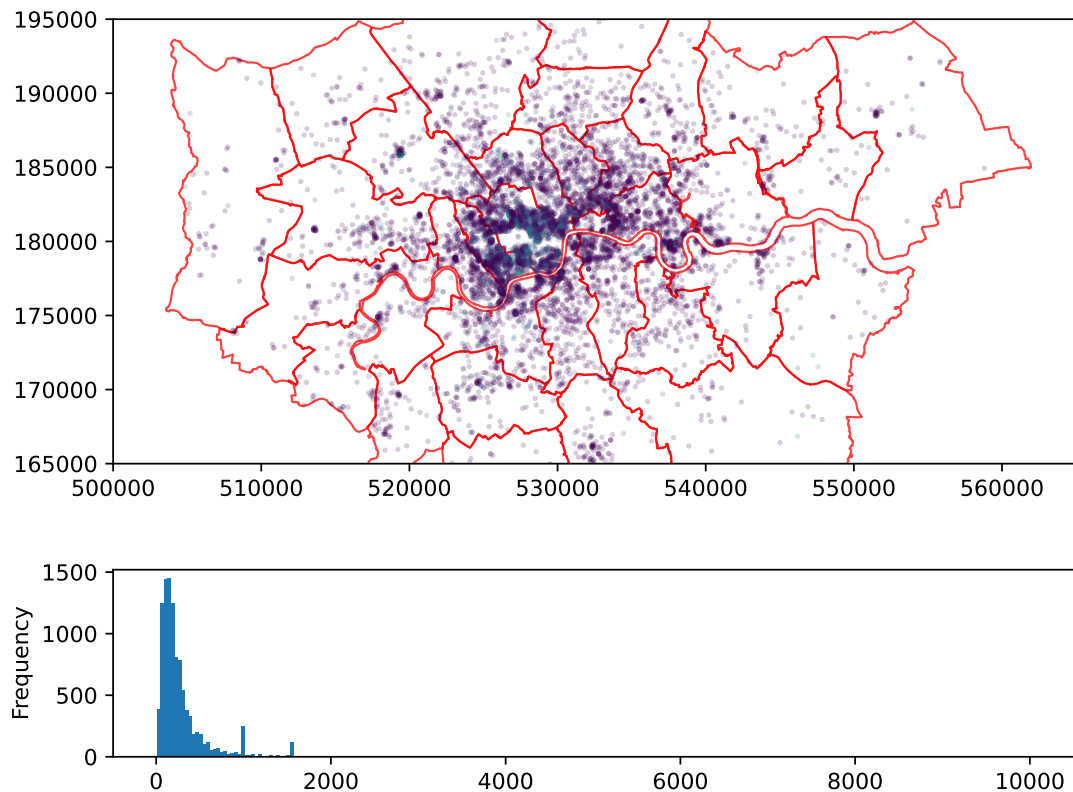


Figure 1: 'Luxury' listings in London

### 4.1.3.2 Answer

- title applies to a single subplot, whereas `suptitle` applies to the entire figure.
- `tight_layout` basically tries to adjust the margins around each figure so as to minimise the amount of whitespace.

## 4.2 Budget Listings



Difficulty level: Easy, because you've worked out the hard bits already.

### 4.2.1 Create the Regular Expression

What words can you think of that might help you to spot affordable and budget accommodation? Start with just a couple of words and then I would encourage you to consider what *other* words might help to signal 'affordability'... perhaps words like 'cosy' or 'charming' and then think about how you could add those to the regex?

*Hints:* this just builds on what you did above with one exception:

1. I'd try adding word boundary markers to the regex (`\b`) where appropriate...

#### 4.2.1.1 Answer

```
gdf[
    gdf.description.str.contains(r'\b(?:affordable|budget|cheap|cosy)\b',
    regex=True, flags=re.IGNORECASE)
].sample(5, random_state=42)[['description']]
```

description	
id	
10466073	This is a charming little flat in crouch end, North London. It is located 10min
679538210387434496	This trendy and spacious 3 bed house in Bow, is ideal for people who want to
7560171	The spacious house offers double bedroom with ensuit bathroom, a large kitc
721518857445429120	Kick back and relax in this calm, stylish one-bedroom, ground-floor flat in Le
731014319309725184	Beautiful 1 bedroom apartment in the heart of Peckham, only 10 minutes wall

description	
id	
10466073	This is a charming little flat in crouch end, North London. It is located 10min
679538210387434496	This trendy and spacious 3 bed house in Bow, is ideal for people who want to
7560171	The spacious house offers double bedroom with ensuit bathroom, a large kitc
721518857445429120	Kick back and relax in this calm, stylish one-bedroom, ground-floor flat in Le
731014319309725184	Beautiful 1 bedroom apartment in the heart of Peckham, only 10 minutes wall

#### 4.2.2 Apply it to Select Data

##### 4.2.2.1 Answer

```
aff = gdf[gdf.description.str.contains(r'\b(?:affordable|budget|cheap|cosy)\b',
    regex=True, flags=re.IGNORECASE)].copy()
print(f"There are {aff.shape[0]:,} rows flagged as 'affordable'.")
```

There are 8,937 rows flagged as 'affordable'.

You should get {python} f"{aff.shape[0]:,}" rows.

#### 4.2.3 Plot the Data

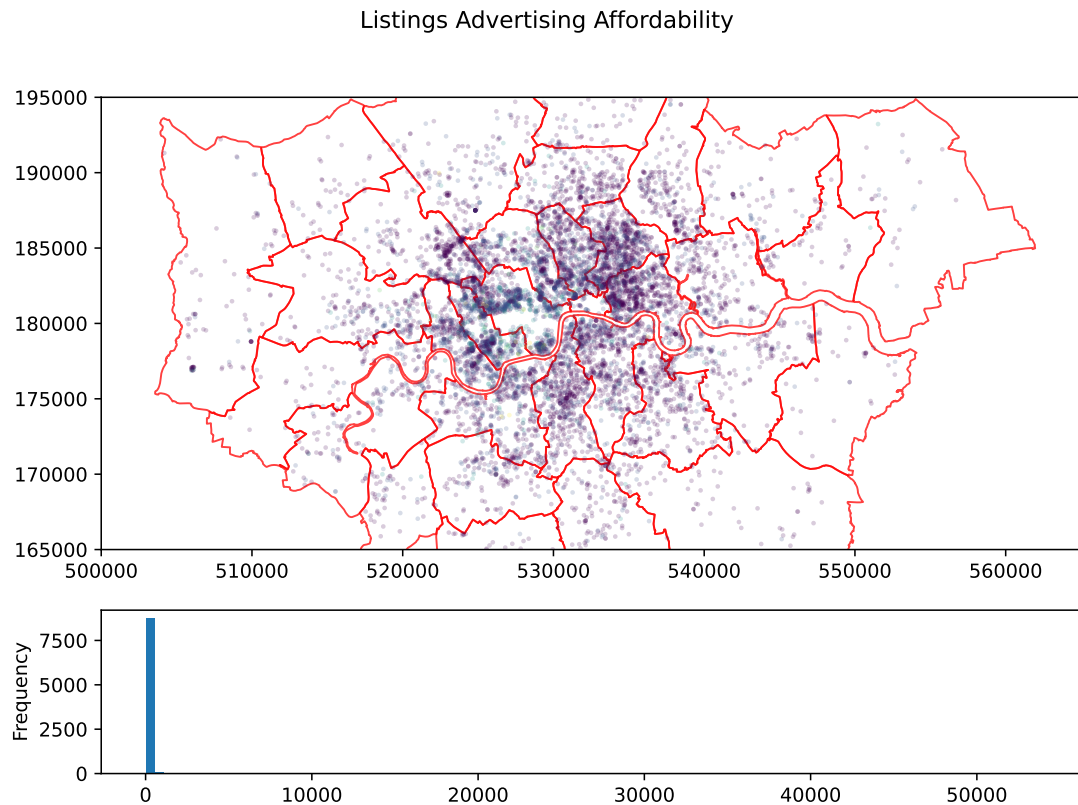
```
f,ax = plt.subplots(1,1,figsize=(8,6))
ax.remove()

# The first plot
ax1 = plt.subplot2grid((4, 1), (0, 0), rowspan=3)
boros.plot(edgecolor='red', facecolor='none', linewidth=1, alpha=0.75, ax=ax1)
aff.plot(markersize=2, column='price', cmap='viridis', alpha=0.2, scheme='Fisher_Jenks_Sampled', ax=ax1)

ax1.set_xlim([500000, 565000])
ax1.set_ylim([165000, 195000]);
```

```
# The second plot
ax2 = plt.subplot2grid((4, 1), (3, 0), rowspan=1)
aff.price.plot.hist(bins=100, ax=ax2)


plt.suptitle("Listings Advertising Affordability")
plt.tight_layout()
#plt.savefig("Affordable_Listings.png", dpi=150)
```



#### 4.2.4 Questions

- Do you think that this is a *good* way to select affordable options?
- Do you understand what dpi means and how savefig works?
- Copy the code from above but modify it to constrain the histogram on a more limited distribution by *filtering* out the outliers *before* drawing the plot. I would copy the cell above to one just below here so that you keep a working copy available and can undo any changes that break things.

#### 4.3 Near Bluespace

 Difficulty level: Medium, because you're still learning about regexes.

Now see if you can work out a regular expression to find accommodation that emphasises accessibility to the Thames and other ‘blue spaces’ as part of the description? One thing you’ll need to tackle is that some listings seem to say something about Thameslink and you wouldn’t want those be returned as part of a regex looking for *ivers*. So by way of a hint:

- You probably need to think about the Thames, rivers, and water.
- These will probably be *followed* by a qualifier like a ‘view’ (e.g. Thames-view) or a front (e.g. water-front).
- But you need to rule out things like “close the Thameslink station...”

#### 4.3.1 Create the regular Expression

##### 4.3.1.1 Answer

```
gdf[
    gdf.description.str.contains(r'(?:(Thames|river|water|canals)?)(?:-|\s+)(?: view|\s?front)\b',
    regex=True, flags=re.IGNORECASE)
].sample(5, random_state=42)[['description']]
```

	description
id	
906924808902274560	Brand new ultra spacious apartment just one minute walk from the Excel cen
600645213392859136	Are you a woman? We offer a warm and friendly, Home from home, welc
868117243960619264	Just across the street from water front, near shadwell basin. View of Canary V
722937027420061056	There’s no better way to experience the beauty of London City than by sleepin
26748596	Welcome to Sunbury Lane in Battersea :)  My house is just off the ri

	description
id	
906924808902274560	Brand new ultra spacious apartment just one minute walk from the Excel cen
600645213392859136	Are you a woman? We offer a warm and friendly, Home from home, welc
868117243960619264	Just across the street from water front, near shadwell basin. View of Canary V
722937027420061056	There’s no better way to experience the beauty of London City than by sleepin
26748596	Welcome to Sunbury Lane in Battersea :)  My house is just off the ri

#### 4.3.2 Apply it to the Select Data

##### 4.3.2.1 Answer

```
bluesp = gdf[
    (gdf.description.str.contains(r'(?:(Thames|river|water|canals)?)(?:-|\s+)(?:view|front)\b', regex=True, flags=re.IGNORECASE)
    & (gdf.description.str.contains(r'(?:(walk|close|near) to (?:(Thames|river|water|canal)', regex=True, flags=re.IGNORECASE)
)].copy()
print(f"Found {bluesp.shape[0]:,} rows.")
```

Found 408 rows.

You should get {python} f"{bluesp.shape[0]:,}" rows.



### 4.3.3 Plot the Data

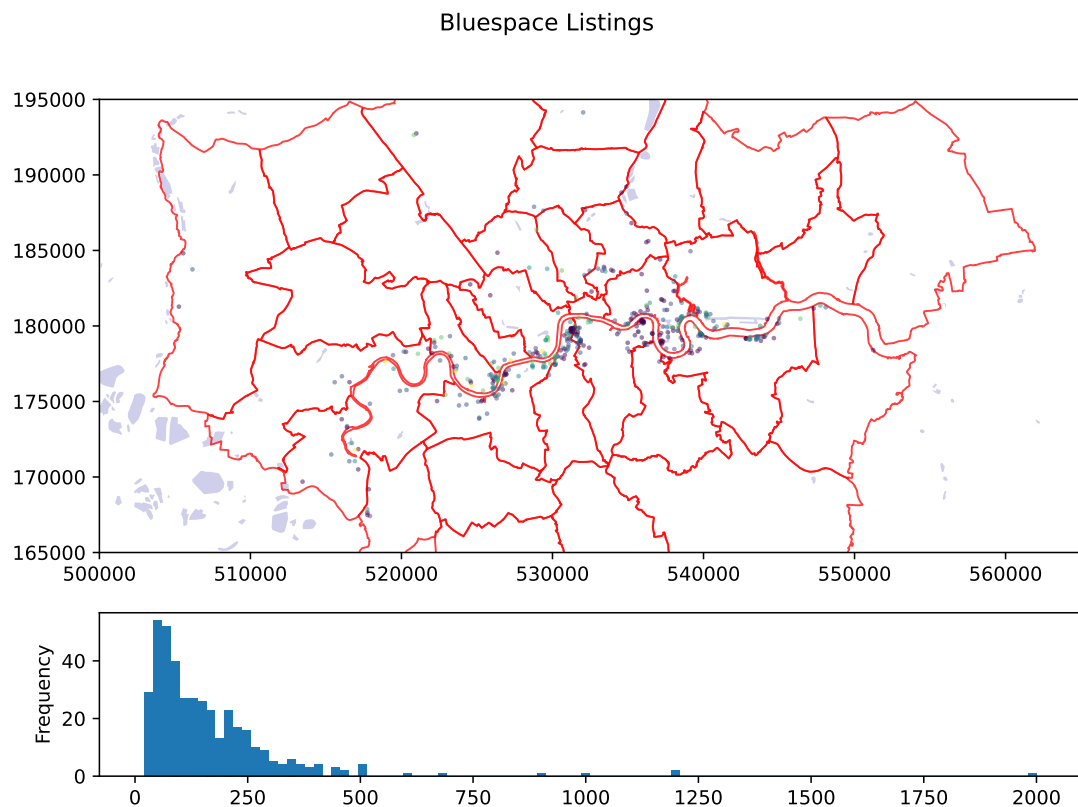
```
f,ax = plt.subplots(1,1,figsize=(8,6))
ax.remove()

# The first plot
ax1 = plt.subplot2grid((4, 1), (0, 0), rowspan=3)
water.plot(edgecolor='none', facecolor=(.25, .25, .7, .25), ax=ax1)
boros.plot(edgecolor='red', facecolor='none', linewidth=1, alpha=0.75, ax=ax1)
bluesp.plot(markersize=2, column='price', cmap='viridis', alpha=0.5, scheme='Fisher_Jenks_Sampled', ax=ax1)

ax1.set_xlim([500000, 565000])
ax1.set_ylim([165000, 195000]);

# The second plot
ax2 = plt.subplot2grid((4, 1), (3, 0), rowspan=1)
bluesp.price.plot.hist(bins=100, ax=ax2)

plt.suptitle("Bluespace Listings")
plt.tight_layout()
plt.show()
```




### 4.3.4 Questions

- How else might you select listings with a view of the Thames or other bluespaces?

## 5 Text Cleaning

Now we're going to step through the *parts* of the process in which we apply to clean and transform text. We'll do this individually before using a function to apply them *all at once*.

### 5.1 Removing HTML

 Difficulty level: Moderate

Because what we're doing will seem really strange and uses some previously unseen libraries that you'll have to google.

*Hint:* you need to need to **get the text** out of the each returned `<p>` and `<div>` element! I'd suggest also commenting this up since there is a *lot* going on on some of these lines of code!

We're going to take a single record to process:

```
sample_row = bluesp.sample(1, random_state=42).description.to_list()[0]
```

Our sample data looks like this:

An amazing room with equipment you need to make the most of your trip to London. The room has a double size bed, double wardrobe, chest of drawers. We include bedsheets, towels, hand soap, toilet. The house has a bathroom and kitchen that you will share with other roommates. We are 5 min. walk from Canada Water station on zone 2. Central london is 15-20 in tube. You can walk to Tower Bridge, close to river. The space A house near the tube station, a 24 hr supermarket, in a quiet area of London, but very close to the center too. Guest access Guests can use their room and bathroom. Bathroom and kitchen are shared with roommates. During your stay I'm available for questions or tips around the area, through Airbnb messages. Always happy to help! Other things to note Please do not use shoes inside the house We kindly ask to keep noise low from 2

Note, however, that it includes some formatting (bold, line breaks, etc.) that isn't strictly relevant and might actually give us issues downstream because it's 'markup' and not content:

An amazing room with equipment you need to make the most of your trip to London. `<br />`  
`<br />`

The room has a double size bed, double wardrobe, chest of drawers. `<br />`  
`<br />`

We include bedsheets, towels, hand soap, toilet. `<br />`  
`<br />`

The house has a bathroom and kitchen that you will share with other roommates. `<br />`  
`<br />`

We are 5 min. walk from Canada Water station on zone 2. Central london is 15-20 in tube. You can walk to Tower B  
`<br />`

`<b>`The space`</b><br />`

A house near the tube station, a 24 hr supermarket, in a quiet area of London, but very close to the center too.<br />  
 <br />  
 <b>Guest access</b><br />  
 Guests can use their room and bathroom. Bathroom and kitchen are shared with roommates.<br />  
 <br />  
 <b>During your stay</b><br />  
 I'm available for questions or tips around the area, through Airbnb messages. Always happy to help!<br />  
 <br />  
 <b>Other things to note</b><br />  
 Please do not use shoes inside the house<br />  
 <br />  
 We kindly ask to keep noise low from 2<br />

#### 5.1.0.1 Answer

```
cleaned = []
```

```
html = sample_row
soup = BeautifulSoup(html)
body = soup.find('body')
```

```
for c in body.findChildren(recursive=False):
    if c.name in ['div','p'] and c.get_text().strip() != "":
        # \xa0 is a non-breaking space in Unicode (&nbsp; in HTML)
        txt = [re.sub(r'(?:\u202f\u200b)', '', x.strip()) for x in c.get_text(separator=" ").split("\n") if x.strip() != ""]
        cleaned += txt

print(cleaned)
```

["An amazing room with equipment you need to make the most of your trip to London. The room has a double size b

## 5.2 Lower Case

 Difficulty Level: Low.

#### 5.2.0.1 Answer

```
lower = [s.lower() for s in cleaned]
lower
```

["an amazing room with equipment you need to make the most of your trip to london. the room has a double size b

## 5.3 Stripping 'Punctuation'

### Difficulty level: Hard

This is because you need to understand: 1) why we're *compiling* the regular expression and how to use character classes; and 2) how the NLTK tokenizer differs in approach to the regex.

### 5.3.1 Regular Expression Approach

We want to clear out punctuation using a regex that takes advantage of the [...] (character class) syntax. The really tricky part is remembering how to specify the 'punctuation' when some of that punctuation has 'special' meanings in a regular expression context. For instance, . means 'any character', while [ and ] mean 'character class'. So this is another *escaping* problem and it works the *same* way it did when we were dealing with the Terminal...

*Hints:* some other factors...

1. You will want to match more than one piece of punctuation at a time, so I'd suggest add a + to your pattern.
2. You will need to look into *metacharacters* for creating a kind of 'any of the characters *in this class*' bag of possible matches.

#### 5.3.1.1 Answer

```
pattern = re.compile(r'[.,!:-><=\\(\\)\\[\\]\\\\&\\\\\'";\\\\+\\\\-\\\\-]+')
```

```
re.compile('[.,!:-><=\\(\\)\\[\\]\\\\&\\\\\'";\\\\+\\\\-\\\\-]+')
```

### 5.3.2 Tokenizer

The other way to do this, which is probably *easier* but produces more complex output, is to draw on the tokenizers [already provided by NLTK](#). For our purposes word\_tokenize is probably fine, but depending on your needs there are other options and you can also write your own.

```
nltk.download('punkt')
nltk.download('wordnet')
from nltk.tokenize import word_tokenize
print(word_tokenize)
```

```
<function word_tokenize at 0x303ff6200>
```

```
[nltk_data] Downloading package punkt to /Users/jreades/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/jreades/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
```

### 5.3.3 Compare

Look at how these outputs differ in subtle ways:

```
subbed = []
tokens = []
for l in lower:
    subbed.append(re.sub(pattern, '', l))
    tokens.append(word_tokenize(l))

for s in subbed:
    as_markdown("Substituted", s)

for t in tokens:
    as_markdown("Tokenised", t)
```


#### 5.3.3.0.1 Substituted

an amazing room with equipment you need to make the most of your trip to london the room has a double size bed double wardrobe chest of drawers we include bedsheets towels hand soap toilet the house has a bathroom and kitchen that you will share with other roommates we are 5 min walk from canada water station on zone 2 central london is 15 20 in tube you can walk to tower bridge close to river the space a house near the tube station a 24 hr supermarket in a quiet area of london but very close to the center too guest access guests can use their room and bathroom bathroom and kitchen are shared with roommates during your stay i m available for questions or tips around the area through airbnb messages always happy to help other things to note please do not use shoes inside the house we kindly ask to keep noise low from 2

#### 5.3.3.0.2 Tokenised

['an', 'amazing', 'room', 'with', 'equipment', 'you', 'need', 'to', 'make', 'the', 'most', 'of', 'your', 'trip', 'to', 'london', ':', 'the', 'room', 'has', 'a', 'double', 'size', 'bed', ':', 'double', 'wardrobe', ':', 'chest', 'of', 'drawers', ':', 'we', 'include', 'bed-sheets', ':', 'towels', ':', 'hand', 'soap', ':', 'toilet', ':', 'the', 'house', 'has', 'a', 'bathroom', 'and', 'kitchen', 'that', 'you', 'will', 'share', 'with', 'other', 'roommates', ':', 'we', 'are', '5', 'min', ':', 'walk', 'from', 'canada', 'water', 'station', 'on', 'zone', '2', 'central', 'london', 'is', '15-20', 'in', 'tube', ':', 'you', 'can', 'walk', 'to', 'tower', 'bridge', ':', 'close', 'to', 'river', ':', 'the', 'space', 'a', 'house', 'near', 'the', 'tube', 'station', ':', 'a', '24', 'hr', 'supermarket', ':', 'in', 'a', 'quiet', 'area', 'of', 'london', ':', 'but', 'very', 'close', 'to', 'the', 'center', 'too', ':', 'guest', 'access', 'guests', 'can', 'use', 'their', 'room', 'and', 'bathroom', ':', 'bathroom', 'and', 'kitchen', 'are', 'shared', 'with', 'roommates', ':', 'during', 'your', 'stay', 'i', '"', 'm', 'available', 'for', 'questions', 'or', 'tips', 'around', 'the', 'area', ':', 'through', 'airbnb', 'messages', ':', 'always', 'happy', 'to', 'help', '!', 'other', 'things', 'to', 'note', 'please', 'do', 'not', 'use', 'shoes', 'inside', 'the', 'house', 'we', 'kindly', 'ask', 'to', 'keep', 'noise', 'low', 'from', '2']

## 5.4 Stopword Removal

 Difficulty Level: Moderate

You need to remember how list comprehensions work to use the `stopword_list`.

```
stopword_list = set(stopwords.words('english'))
print(stopword_list)
```

```
{'had', 'until', 'have', 'when', 'are', 'such', 'didn', "you've", 'by', "you'll", 'our', 'any', 'this', 'each', "mustn't", "wouldn't", 'o
```

### 5.4.0.1 Answer


```
stopped = []
for p in tokens: # <-- why do I just take these items from the list?
    stopped.append([x for x in p if x not in stopword_list and len(x) > 1])

for s in stopped:
    as_markdown("Line", s)
```

#### 5.4.0.1.1 Line

```
['amazing', 'room', 'equipment', 'need', 'make', 'trip', 'london', 'room', 'double', 'size', 'bed', 'double', 'wardrobe', 'chest', 'drawers', 'include', 'bedsheets', 'towels', 'hand', 'soap', 'toilet', 'house', 'bathroom', 'kitchen', 'share', 'roommates', 'min', 'walk', 'canada', 'water', 'station', 'zone', '2.', 'central', 'london', '15-20', 'tube', 'walk', 'tower', 'bridge', 'close', 'river', 'space', 'house', 'near', 'tube', 'station', '24', 'hr', 'supermarket', 'quiet', 'area', 'london', 'close', 'center', 'guest', 'access', 'guests', 'use', 'room', 'bathroom', 'bathroom', 'kitchen', 'shared', 'roommates', 'stay', "'m", 'available', 'questions', 'tips', 'around', 'area', 'airbnb', 'messages', 'always', 'happy', 'help', 'things', 'note', 'please', 'use', 'shoes', 'inside', 'house', 'kindly', 'ask', 'keep', 'noise', 'low']
```

## 5.5 Lemmatisation vs Stemming

 Difficulty level: Low.

```
from nltk.stem.porter import PorterStemmer
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize('monkeys'))
print(lemmatizer.lemmatize('cities'))
print(lemmatizer.lemmatize('complexity'))
print(lemmatizer.lemmatize('Reades'))
```

monkey  
city  
complexity  
Reades

```
stemmer = PorterStemmer()
print(stemmer.stem('monkeys'))
print(stemmer.stem('cities'))
print(stemmer.stem('complexity'))
print(stemmer.stem('Reades'))
```

monkey  
citi  
complex  
read

```
stemmer = SnowballStemmer(language='english')
print(stemmer.stem('monkeys'))
print(stemmer.stem('cities'))
print(stemmer.stem('complexity'))
print(stemmer.stem('Reades'))
```

monkey  
citi  
complex  
read

```
lemmatizer = WordNetLemmatizer()
lemmas = []
stemmed = []
```

```
# This would be better if we passed in a PoS (Part of Speech) tag as well,
# but processing text for parts of speech is *expensive* and for the purposes
# of this tutorial, not necessary.
```

```
for s in stopped:
    lemmas.append([lemmatizer.lemmatize(x) for x in s])
```

```
for s in stopped:
    stemmed.append([stemmer.stem(x) for x in s])
```

```
for l in lemmas:
    as_markdown('Lemmatised',l)
```

```
for s in stemmed:
    as_markdown('Stemmed',s)
```

#### 5.5.0.0.1 Lemmatised

['amazing', 'room', 'equipment', 'need', 'make', 'trip', 'london', 'room', 'double', 'size', 'bed', 'double', 'wardrobe', 'chest', 'drawer', 'include', 'bedsheets',

‘towel’, ‘hand’, ‘soap’, ‘toilet’, ‘house’, ‘bathroom’, ‘kitchen’, ‘share’, ‘roommate’, ‘min’, ‘walk’, ‘canada’, ‘water’, ‘station’, ‘zone’, ‘2.’, ‘central’, ‘london’, ‘15-20’, ‘tube’, ‘walk’, ‘tower’, ‘bridge’, ‘close’, ‘river’, ‘space’, ‘house’, ‘near’, ‘tube’, ‘station’, ‘24’, ‘hr’, ‘supermarket’, ‘quiet’, ‘area’, ‘london’, ‘close’, ‘center’, ‘guest’, ‘access’, ‘guest’, ‘use’, ‘room’, ‘bathroom’, ‘bathroom’, ‘kitchen’, ‘shared’, ‘roommate’, ‘stay’, “‘m”, ‘available’, ‘question’, ‘tip’, ‘around’, ‘area’, ‘airbnb’, ‘message’, ‘always’, ‘happy’, ‘help’, ‘thing’, ‘note’, ‘please’, ‘use’, ‘shoe’, ‘inside’, ‘house’, ‘kindly’, ‘ask’, ‘keep’, ‘noise’, ‘low’]

#### 5.5.0.0.2 Stemmed

[‘amaz’, ‘room’, ‘equip’, ‘need’, ‘make’, ‘trip’, ‘london’, ‘room’, ‘doubl’, ‘size’, ‘bed’, ‘doubl’, ‘wardrob’, ‘chest’, ‘drawer’, ‘includ’, ‘bedsheet’, ‘towel’, ‘hand’, ‘soap’, ‘toilet’, ‘hous’, ‘bathroom’, ‘kitchen’, ‘share’, ‘roommat’, ‘min’, ‘walk’, ‘canada’, ‘water’, ‘station’, ‘zone’, ‘2.’, ‘central’, ‘london’, ‘15-20’, ‘tube’, ‘walk’, ‘tower’, ‘bridg’, ‘close’, ‘river’, ‘space’, ‘hous’, ‘near’, ‘tube’, ‘station’, ‘24’, ‘hr’, ‘supermarket’, ‘quiet’, ‘area’, ‘london’, ‘close’, ‘center’, ‘guest’, ‘access’, ‘guest’, ‘use’, ‘room’, ‘bathroom’, ‘bathroom’, ‘kitchen’, ‘share’, ‘roommat’, ‘stay’, “‘m”, ‘avail’, ‘question’, ‘tip’, ‘around’, ‘area’, ‘airbnb’, ‘messag’, ‘alway’, ‘happi’, ‘help’, ‘thing’, ‘note’, ‘pleas’, ‘use’, ‘shoe’, ‘insid’, ‘hous’, ‘kind’, ‘ask’, ‘keep’, ‘nois’, ‘low’]

#### 5.5.0.1 Compare and Contrast

What are we doing here?

```
for ix, p in enumerate(stopped):
    stopped_set = set(stopped[ix])
    lemma_set = set(lemmas[ix])
    print(sorted(stopped_set.symmetric_difference(lemma_set)))
```

['drawer', 'drawers', 'guests', 'message', 'messages', 'question', 'questions', 'roommate', 'roommates', 'shoe', 'shoes

## 6 Pandas.Apply

The above approach is fairly hard going since you need to loop through every list element applying these changes one at a time. Instead, we could convert the column to a corpus (or use pandas apply) together with a function imported from a library to do the work.

### 6.1 Downloading the Custom Module

💡 Difficulty level: Low.

This custom module is not perfect, but it gets the job done... mostly and has some additional features that you could play around with for a final project (e.g. detect\_entities and detect\_acronyms).



```
import urllib.request
host = 'https://orca.casa.ucl.ac.uk'
turl = f'{host}/~jreades/__textual__.py'
tdirs = os.path.join('textual')
tpath = os.path.join(tdirs, '__init__.py')

if not os.path.exists(tpath):
    os.makedirs(tdirs, exist_ok=True)
    urllib.request.urlretrieve(turl, tpath)
```

## 6.2 Importing the Custom Module

💡 Difficulty Level: Low.

But only because you didn't have to write the module! However, the questions could be hard...

In a Jupyter notebook, this code allows us to edit and reload the library dynamically:

```
%load_ext autoreload
%autoreload 2
```

Now let's import it.

```
from textual import *
```

All NLTK libraries installed...

```
as_markdown('Input', cleaned)
```

### 6.2.0.0.1 Input

["An amazing room with equipment you need to make the most of your trip to London. The room has a double size bed, double wardrobe, chest of drawers. We include bedsheets, towels, hand soap, toilet. The house has a bathroom and kitchen that you will share with other roommates. We are 5 min. walk from Canada Water station on zone 2. Central london is 15-20 in tube. You can walk to Tower Bridge, close to river. The space A house near the tube station, a 24 hr supermarket, in a quiet area of London, but very close to the center too. Guest access Guests can use their room and bathroom. Bathroom and kitchen are shared with roommates. During your stay I'm available for questions or tips around the area, through Airbnb messages. Always happy to help! Other things to note Please do not use shoes inside the house We kindly ask to keep noise low from 2"]

```
as_markdown('Normalised', [normalise_document(x, remove_digits=True) for x in cleaned])
```

#### 6.2.0.0.2 Normalised

```
[‘amazing room equipment need make trip london . room double size dou-  
ble wardrobe chest drawer . include bedsheets towel hand soap toilet .  
house bathroom kitchen share roommate . walk canada water station zone .  
central london tube . walk tower bridge close river . space house near tube  
station supermarket quiet area london close center . guest access guests  
room bathroom . bathroom kitchen share roommate . stay available ques-  
tion around area airbnb message . always happy help thing note please shoe  
inside house kindly keep noise’]
```

`help(normalise_document)`

Help on function normalise\_document in module textual:

`normalise_document(doc: str, html_stripping=True, contraction_expansion=True, accented_char_removal=True, te`  
Apply all of the functions above to a document using their  
default values so as to demonstrate the NLP process.

`doc`: a document to clean.

#### 6.2.1 Questions

Let’s assume that you want to analyse web page content...

- Based on the above output, what stopwords do you think are missing?
- Based on the above output, what should be removed but isn’t?
- Based on the above output, how do you think a computer can work with this text?

#### Stop!

Beyond this point, we are moving into Natural Language Processing. If you are already struggling with regular expressions, I would recommend *stopping here*. You can come back to revisit the NLP components and creation of word clouds later.

## 7 Data Normalisation

Now that you’ve seen how the steps are applied to a ‘random’ document, let’s get back to the problem at hand or, as the French would put it, revenons à nos moutons (let’s get back to our sheep).

### 7.1 Process the Selected Listings

 Difficulty level: Low, but you’ll need to be patient!

Notice the use of `%%time` here – this will tell you how long each block of code takes to complete. It’s a really useful technique for reminding *yourself* and others of how long

something might take to run. I find that with NLP this is particularly important since you have to do a *lot* of processing on each document in order to normalise it.

#### Tip

Notice how we can change the default parameters for `normalise_document` even when using `apply`, but that the syntax is different. So whereas we'd use `normalise_document(doc, remove_digits=True)` if calling the function directly, here it's `.apply(normalise_document, remove_digits=True)`!

#### 7.1.0.1 Answer


We're only going to show one of these since that speeds up the rendering, for the luxury and affordable samples you are looking at upwards of 1 minute calculation time.

```
%%time
# I get about 1 seconds on a M2 Mac
bluesp['description_norm'] = bluesp.description.apply(normalise_document, remove_digits=True)
```

CPU times: user 1.62 s, sys: 39.2 ms, total: 1.65 s

Wall time: 1.65 s

## 7.2 Select and Tokenise

 Difficulty level: Low, except for the double list-comprehension.

#### 7.2.1 Select and Extract Corpus

See useful tutorial [here](#). Although we shouldn't have any empty descriptions, by the time we've finished normalising the textual data we may have *created* some empty values and we need to ensure that we don't accidentally pass a NaN to the vectorisers and frequency distribution functions.

```
# The data set to which we want to apply
# the data normalisation code... feel free
# to change (though so will your results)
srcdf = bluesp
```

#### Coding Tip

Notice how you only need to change the value of the variable here to try any of the different selections we did above? This is a simple kind of parameterisation somewhere between a function and hard-coding everything.

```
corpus = srcdf.description_norm.fillna(' ').values
print(corpus[0:3])
```

```
['house garden close thames river . walk private road river nearby . district line underground . walk . direct access  
'space apartment upper floor modernised secure building near canary wharf fantastic view river thames london .  
'newly renovate totally equipped furnished modern apartment heart london . easily accessible kind transport . wall
```

### 7.2.2 Tokenise

There are different forms of tokenisation and different algorithms will expect differing inputs. Here are two:

```
sentences = [nltk.sent_tokenize(text) for text in corpus]  
words     = [[nltk.tokenize.word_tokenize(sentence)  
              for sentence in nltk.sent_tokenize(text)]  
              for text in corpus]
```


Notice how this has turned every sentence into an array and each document into an array of arrays:

```
print(f"Sentences 0: {sentences[0]}")  
print()  
print(f"Words 0: {words[0]}")
```

Sentences 0: ['house garden close thames river .', 'walk private road river nearby .', 'district line underground .', 'wa

Words 0: [['house', 'garden', 'close', 'thames', 'river', '.'], ['walk', 'private', 'road', 'river', 'nearby', '.'], ['district', 'line', 'u

## 7.3 Frequencies and Ngrams

 Difficulty level: Moderate.

One new thing you'll see here is the ngram: ngrams are 'simply' pairs, or triplets, or quadruplets of words. You may come across the terms unigram (ngram(1,1)), bigram (ngram(2,2)), trigram (ngram(3,3))... typically, you will rarely find anything beyond tri-grams, and these present real issues for text2vec algorithms because the embedding for geographical, information, and systems is *not* the same as for geographical information systemts.

### 7.3.1 Build Frequency Distribution

Build counts for ngram range 1..3:

```
fcounts = dict()  
  
# Here we replace all full-stops... can you think why we might do this?  
data = nltk.tokenize.word_tokenize(' '.join([text.replace('.', '') for text in corpus]))  
  
for size in 1, 2, 3:  
    fdist = FreqDist(ngrams(data, size))  
    print(fdist)
```

```
# If you only need one note this: https://stackoverflow.com/a/52193485/4041902
fcounts[size] = pd.DataFrame.from_dict({f'Ngram Size {size}': fdist})
```

<FreqDist with 2692 samples and 26245 outcomes>

<FreqDist with 14173 samples and 26244 outcomes>

<FreqDist with 19540 samples and 26243 outcomes>

### 7.3.2 Output Top-n Ngrams

And output the most common ones for each ngram range:

```
for dfs in fcounts.values():
    print(dfs.sort_values(by=dfs.columns.values[0], ascending=False).head(10))
    print()
```

#### Ngram Size 1

walk	594
room	472
london	469
river	418
bedroom	412
space	391
minute	382
apartment	373
station	307
flat	303

#### Ngram Size 2

minute walk	252
river thames	138
view	138
living room	134
canary wharf	112
guest access	110
central london	107
fully equip	76
equip kitchen	71
thames river	65

#### Ngram Size 3

fully equip kitchen	68
walk river thames	37
close river thames	35
walk thames river	27
minute walk river	23
open plan kitchen	20
thames river view	20
sofa living room	19
within walk distance	19
open plan live	18

### 7.3.3 Questions

- Can you think why we don't care about punctuation for frequency distributions and n-grams?
- Do you understand what n-grams *are*?

## 7.4 Count Vectoriser

💡 Difficulty level: Low, but the output needs some thought!

This is a big foray into sklearn (sci-kit learn) which is the main machine learning and clustering module for Python. For processing text we use *vectorisers* to convert terms to a vector representation. We're doing this on the smallest of the derived data sets because these processes can take a while to run and generate *huge* matrices (remember: one row and one column for each term!).

### 7.4.1 Fit the Vectoriser

```
cvectorizer = CountVectorizer(ngram_range=(1,3))
cvectorizer.fit(corpus)
```

```
CountVectorizer(ngram_range=(1, 3))
```

### 7.4.2 Brief Demonstration

Find the number associated with a word in the vocabulary and how many times it occurs in the original corpus:

```
term = 'stratford'
pd.options.display.max_colwidth=750
# Find the vocabulary mapping for the term
print(f"Vocabulary mapping for {term} is {cvectorizer.vocabulary_[term]}")
# How many times is it in the data
print(f"Found {srcdf.description_norm.str.contains(term).sum():,} rows containing {term}")
# Print the descriptions containing the term
for x in srcdf[srcdf.description_norm.str.contains(term)].description_norm:
    as_markdown('Stratford',x)
```

Vocabulary mapping for stratford is 29373

Found 10 rows containing stratford

#### 7.4.2.0.1 Stratford

house garden close thames river . walk private road river nearby . district line underground . walk . direct access central london near gardens . kids playground walk distance along thames path . space residential neighborhood english corporate expat family . house culdesac private road river thames . river foot away . walking distance subway . central london underground district line . gardens stop walk zone . addition overground stratford also stop gardens underground station . gardens stop walk . overland railway station bridge . walk . take waterloo railway station minute . bicycle follow towpath hammersmith bridge continue putney bridge . lastly several stree

#### 7.4.2.0.2 Stratford

please read things note comfortable clean bright brand flat east london minute central london tube quite central great transport link major london attraction minute walk river park undergroundtubedlr station supermarket docklands stratford olympic stadium westfield shopping centre . enjoy brick lane indian restaurant spitalfields market colombian flower market historical whitechapel . space please read things note nice clean fresh bright airy . space perfect professional single person couple . make feel like home choice anything like wake relaxing cooking . guest access please read things note entire flat . please treat home away . please treat .

#### 7.4.2.0.3 Stratford

comfortable fairly flat east london travel zone minute central london quite central great transport link major london attraction minute walk river park undergroundtube station minute supermarket minute docklands stratford olympic stadium westfield shopping centre . enjoy brick lane indian restaurant spitalfields market colombian flower market historical whitechapel . space spacious comfortable tidy clean airy relaxing . live flat sleep open plan lounge balcony . guest access bathroom share . welcome microwave ready meal toaster make drink till fridge store food . please make sure clean clear immediately . dining table . stay present morning evening weekend . also

#### 7.4.2.0.4 Stratford

entire appartment double bedroom large living area.the apartment feature kitchen come free wifi flat screen tv.to make exicting luxurious even free free sauna well . space stunning apartment london docklands bank thames river close thames barrier park canary wharf . apartment floor spacious living room balcony . nearest station pontoon dock walkable distance direct train stratford . mins . bank . excel centre . walk . arena canary wharf mins train mins central london . world heritage sitethames barrier thames barrier park walk appartment . london city airport train station away . fully kitchen bathroom broadband internet underground secure parking onsite . attract

#### 7.4.2.0.5 Stratford

luxurious bedroom apartment zone love hidden secret part town minute away everywhere river view slow pace main artery town right doorstep well hidden beauty park waterway . easy . walk tube route center town well stratford olympic park canary wharf much much right doorstep space welcome home place love bedroom hold personal belonging bedroom give guest idea size . bedroom large double accommodate comfortably . sofa chair accommodate guest extra extra charge . welcome guest personally wish know . therefore important check time convenient . midnight arrival . time which discuss good

#### 7.4.2.0.6 Stratford

place close mile tube station brick lane shoreditch queen mary university london stratford westfield minute tube central london . love place newly renovate flat amazing canal view guest bedroom clean friendly environment . place good couple solo adventurer business traveller .

#### 7.4.2.0.7 Stratford

locate high street give amazing water view stadium sight amazing architectural structure walk pudding mill lane walk abba walk stratford westfield walk stratfordstratford international station mins walk mins train ride central london

#### 7.4.2.0.8 Stratford

modern spacious bedroom suite apartment close river thames wimbledon . situate wandsworth district london england lawn tennis club centre court .km clapham junction . stratford bridge chelsea . city view free wifi throughout property . apartment feature bedroom kitchen fridge oven wash machine flat screen seating area bathroom shower . eventim .km away .

#### 7.4.2.0.9 Stratford

perfect group trip . modern spacious suite apartment close river thames wimbledon . situated wandsworth district london england lawn tennis club centre court .km clapham junction . stratford bridge chelsea . city view free wifi throughout property . apartment feature bedroom kitchen wfridge oven wash machine flat screen seating area bathroom wshower . eventim .km away .

#### 7.4.2.0.10 Stratford

flat locate zone east london near canary wharf . nice quiet residential area canal . flat amazing canal view balcony . enjoy morning coffee swan goose everyday . huge park opposite flat picnic . canary wharf shop mall . mins bank stratford westfield . mins central oxford circus tube . locate convenient transportation link .



### 7.4.3 Transform the Corpus

You can only *transform* the entire corpus *after* the vectoriser has been fitted. There is an option to fit\_transform in one go, but I wanted to demonstrate a few things here and some vectorisers are don't support the one-shot fit-and-transform approach. **Note the type of the transformed corpus:**

```
cvtcorpus = cvectorizer.transform(corpus)
cvtcorpus # cvtcorpus for count-vectorised transformed corpus

<408x35278 sparse matrix of type '<class 'numpy.int64'>'
  with 71420 stored elements in Compressed Sparse Row format>
```

### 7.4.4 Single Document

Here is the **first** document from the corpus:

```
doc_df = pd.DataFrame(cvtcorpus[0].T.todense(),
                      index=cvectorizer.get_feature_names_out(), columns=["Counts"])
                      ).sort_values("Counts", ascending=False)
doc_df.head(10)
```

	Counts
walk	6
gardens	4
river	4
bridge	3
stop	3
thames	3
station	3
underground	3
railway	2
central london	2

### 7.4.5 Transformed Corpus

```
cvdf = pd.DataFrame(data=cvtcorpus.toarray(),
                    columns=cvectorizer.get_feature_names_out())
print(f"Raw count vectorised data frame has {cvdf.shape[0]:,} rows and {cvdf.shape[1]:,} columns.")
cvdf.iloc[0:5,0:10]
```

Raw count vectorised data frame has 408 rows and 35,278 columns.

	aaathe	aaathe apartment	aaathe apartment quiet	aand	aand comfy	aand comfy sofa	ab
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0

	aaathe	aaathe apartment	aaathe apartment quiet	aand	aand comfy	aand comfy sofa	ab
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0

#### 7.4.6 Filter Low-Frequency Words

These are likely to be artefacts of text-cleaning or human input error. As well, if we're trying to look across an entire corpus then we might not want to retain words that only appear in a couple of documents.

Let's start by getting the *column* sums:

```
sums = cvdf.sum(axis=0)
print(f"There are {len(sums):,} terms in the data set.")
sums.head()
```

There are 35,278 terms in the data set.

```
aaathe      1
aaathe apartment      1
aaathe apartment quiet  1
aand      1
aand comfy      1
dtype: int64
```

Remove columns (i.e. terms) appearing in **less than 1% of documents**. You can do this by thinking about what the shape of the data frame means (rows and/or columns) and how you'd get 1% of that!

##### 7.4.6.1 Answer

```
filter_terms = sums >= cvdf.shape[0] * 0.01
```

Now see how we can use this to strip out the columns corresponding to low-frequency terms:

```
fcvdf = cvdf.drop(columns=cvdf.columns[~filter_terms].values)
print(f"Filtered count vectorised data frame has {fcvdf.shape[0]:,} rows and {fcvdf.shape[1]:,} columns.")
fcvdf.iloc[0:5,0:10]
```

Filtered count vectorised data frame has 408 rows and 2,043 columns.

	able	access	access access	access bathroom	access central	access central london	access
0	0	1	0	0	1	1	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0


```
fcvdf.sum(axis=0)
```

```
able          8
access        242
access access    7
access bathroom  5
access central   8
...
zone comfortable cosy  7
zone near           5
zone near underground  5
zone recently       10
zone recently refurbish 10
Length: 2043, dtype: int64
```

#### 7.4.7 Questions

- Can you explain what doc\_df contains?
- What does cvdf contain? Explain the rows and columns.
- What is the function of filter\_terms?

### 7.5 TF/IDF Vectoriser

 Difficulty level: Moderate

But only if you want to understand how max\_df and min\_df work!

#### 7.5.1 Fit and Transform

```
tfvectorizer = TfidfVectorizer(use_idf=True, ngram_range=(1,3),
                               max_df=0.75, min_df=0.01) # <-- these matter!
tftcorpus = tfvectorizer.fit_transform(corpus) # TF-transformed corpus
```

#### 7.5.2 Single Document

```
doc_df = pd.DataFrame(tftcorpus[0].T.todense(), index=tfvectorizer.get_feature_names_out(), columns=["Weights"])
doc_df.sort_values("Weights", ascending=False).head(10)
```

	Weights
gardens	0.414885
stop	0.241659
district line	0.239192
railway	0.232131
underground	0.201738
district	0.197221
bridge	0.191983
walk	0.189485

	Weights
road	0.151163
distance	0.142999

### 7.5.3 Transformed Corpus

```
tfidf = pd.DataFrame(data=tftcorpus.toarray(),
                     columns=tfvectorizer.get_feature_names_out())
print(f"TF/IDF data frame has {tfidf.shape[0]:,} rows and {tfidf.shape[1]:,} columns.")
tfidf.head()
```

TF/IDF data frame has 408 rows and 1,911 columns.

	able	access	access access	access bathroom	access central	access central london	access
0	0.0	0.043972	0.0	0.0	0.11031	0.11031	0.0
1	0.0	0.000000	0.0	0.0	0.00000	0.00000	0.0
2	0.0	0.000000	0.0	0.0	0.00000	0.00000	0.0
3	0.0	0.000000	0.0	0.0	0.00000	0.00000	0.0
4	0.0	0.044127	0.0	0.0	0.00000	0.00000	0.0

### 7.5.4 Questions

- What does the TF/IDF score *represent*?
- What is the role of `max_df` and `min_df`?

## 8 Word Clouds

### 8.1 For Counts

💡 Difficulty level: Easy!

```
fcvdf.sum().sort_values(ascending=False)
```

```
walk          595
room          472
london        471
river         418
bedroom       412
...
chic           5
term           5
choice         5
teddington     5
london aquarium minute  5
Length: 2043, dtype: int64
```



```
f,ax = plt.subplots(1,1,figsize=(7, 7))
plt.gcf().set_dpi(150)
Cloud = WordCloud(
    background_color="white",
    max_words=100,
    font_path=fp
).generate_from_frequencies(tfidf.sum())
ax.imshow(Cloud)
ax.axis("off");
#plt.savefig("Wordcloud 2.png")
```



- What does the sum represent for the count vectoriser?
- What does the sum represent for the TF/IDF vectoriser?

There is a lot more information out there, including a [whole book](#) and your standard [O'Reilly text](#).

- Pandas String Contains Method
- Using Regular Expressions with Pandas
- Summarising Chapters from Frankenstein using TF/IDF