Visualising Texts and their Features

Max Callaghan

2022-09-15

Objectives

Objectives

In this session we will start exploring how to visualise texts and their features with R and Python.

We are going to explore this using ggplot2 and seaborn.

First plots with ggplot2 / Seaborn

First plots

Let's first load some data.

```
df <- readr::read_csv("data/hertie_papers.csv")
head(df,3)
```

```
## # A tibble: 3 x 6
##   id                                doi               title publi~1 abstr~2 authors
##   <chr>                             <chr>             <chr>   <dbl> <chr>   <chr>
## 1 https://openalex.org/W2195453830 https://doi.or~ Biop~    2016 To hav~ Pete S~
## 2 https://openalex.org/W18536190   https://doi.or~ New ~    2019 Politi~ Claus ~
## 3 https://openalex.org/W2092902022 https://doi.or~ The ~    2014 We exa~ Alnoor~
## # ... with abbreviated variable names 1: publication_year, 2: abstract
```

```
import pandas as pd
df = pd.read_csv("data/hertie_papers.csv")
df.head(3)
```

```
##                                  id  ...                                          authors
## 0  https://openalex.org/W2195453830  ...  Pete Smith, Steven J. Davis, Felix Creutzig, S...
## 1    https://openalex.org/W18536190  ...                                       Claus Offe
## 2  https://openalex.org/W2092902022  ...      Alnoor Ebrahim, Julie Battilana, Johanna Mair
##
## [3 rows x 6 columns]
```

A line plot with ggplot

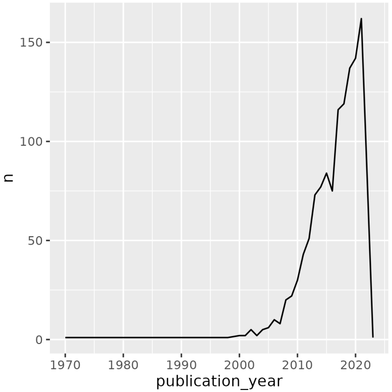
The first thing we will do is plot the number of papers per year.

count() gives us the number of observations of each value of the variable(s) we give it.

Now we can say to ggplot that the "aesthetic mapping" we want is that x should show the publication year and y should show the count of papers in that year

```
annual_pubs <- df %>% count(publication_year)

ggplot(annual_pubs, aes(publication_year, n)) +
  geom_line()
```

```
ggsave("plots/pubs_time_gg.png", width=4, height=4)
```

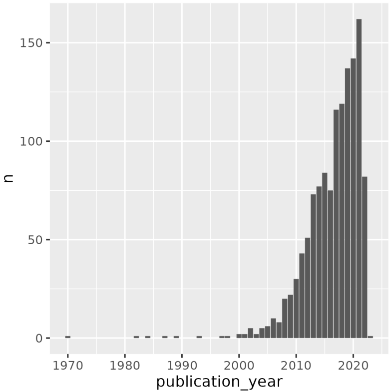


A bar plot with ggplot

ggplot has a variety of different geoms. Each translates our aesthetic mapping to ink on paper in a consistent and clearly defined way.

```
annual_pubs <- df %>% count(publication_year)

ggplot(annual_pubs, aes(publication_year, n)) +
  geom_col()
```

```
ggsave("plots/pubs_time_bar_gg.png", width=4, height=4)
```



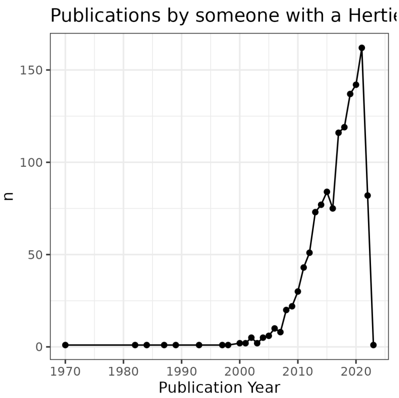
A scatter plot with ggplot

With ggplot, we define the parameters of the plot, and then we can keep adding "geoms" that inherit these parameters.

We build up the plot bit by bit by adding more grammar.

```
annual_pubs <- df %>% count(publication_year)

ggplot(annual_pubs, aes(publication_year, n)) +
  geom_line() +
  geom_point() +
  theme_bw() +
  labs(
    title="Publications by someone with a Hertie affiliation",
    x="Publication Year"
  )
```

```
ggsave("plots/pubs_time_point_gg.png", width=4, height=4)
```

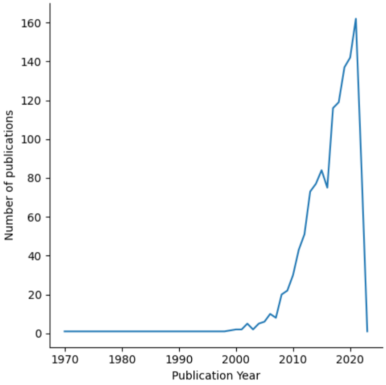


A line plot with seaborn

Seaborn works nicely with things in dataframes, so we need to groupby and count, and coerce the result into a dataframe

```
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("data/hertie_papers.csv")
yps = (df
        .groupby(["publication_year"])["id"]
        .count()
        .to_frame("n_pubs")
        .reset_index()
      )
ax = sns.relplot(
  data=yps, kind="line",
  x="publication_year", y="n_pubs"
)
ax.set(xlabel="Publication Year", ylabel="Number of publications")
```

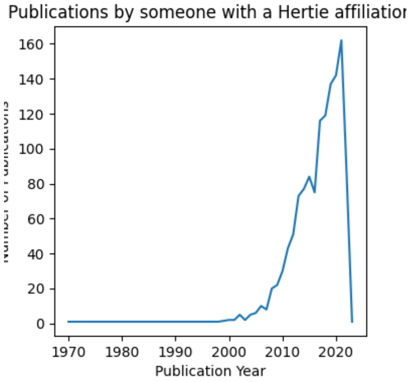
```
plt.savefig("plots/pubs_time_sns.png")
```



A line plot with pandas

Pandas can already produce a lot of the plots we want

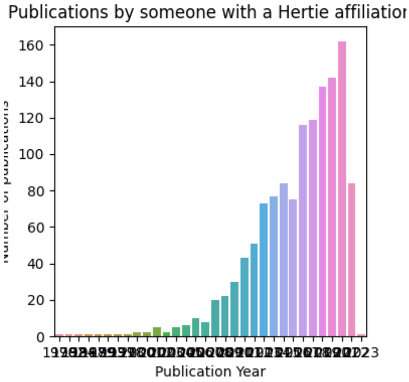
```
import matplotlib.pyplot as plt
import seaborn as sns
fig, ax = plt.subplots(figsize=(4,4))
df.groupby(["publication_year"])["id"].count().plot(ax=ax)
ax.set_xlabel("Publication Year")
ax.set_ylabel("Number of Publications")
ax.set_title("Publications by someone with a Hertie affiliation")
plt.savefig("plots/pubs_time_pd.png")
```



A bar plot with seaborn

Seaborn is also “opinionated” and makes strong assumptions about what you want to do. According to seaborn, if you are making a bar plot, then one of your variables is likely categorical and it will plot it accordingly.

```
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("data/hertie_papers.csv")
yps = (df
        .groupby(["publication_year"])["id"]
        .count()
        .to_frame("n_pubs")
        .reset_index()
      )
ax = sns.barplot(data=yps, x="publication_year", y="n_pubs")
ax.set(xlabel="Publication Year", ylabel="Number of publications")
plt.savefig("plots/pubs_time_bar_sns.png")
```

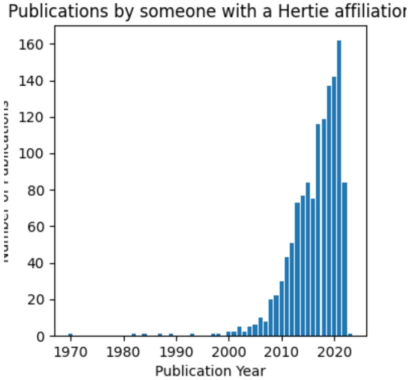


A bar plot with matplotlib

Matplotlib is sometimes the simplest option for simple plots.

```
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("data/hertie_papers.csv")
yps = (df
        .groupby(["publication_year"])["id"]
        .count()
        .to_frame("n_pubs")
        .reset_index()
      )
fig, ax = plt.subplots(figsize=(4,4))
ax.bar(yps["publication_year"], yps["n_pubs"])
```

```
ax.set_xlabel("Publication Year")
ax.set_ylabel("Number of Publications")
ax.set_title("Publications by someone with a Hertie affiliation")
plt.savefig("plots/pubs_time_bar_mpl.png")
```



Exercise

Load the authorship data in `data/author_df.csv` and make a horizontal bar plot showing the 10 authors who have published the most papers *with Hertie affiliations*. In R you may need the functions `filter()`, `count()`, `arrange()`, and `head()/tail()`. In python you will need to filter data `df[df["x"]=="y"]`, and to use the `sort_values()` as well as `head()/tail()`

Plotting text data

What text data can we plot

- Frequencies of features
- frequencies of features in subgroups or over time
- relationships between features
- relationships between features and text/author variables

Back to our document feature matrix

Let's create a document feature matrix from our list of abstracts

```
library(quanteda)
df <- df %>% filter(!is.na(abstract))
dfmat <- df$abstract %>%
  tokens(remove_punc=TRUE) %>%
  tokens_remove(pattern=stopwords("en")) %>%
  tokens_wordstem("english") %>%
  dfm()
dfmat
```

```
## Document-feature matrix of: 1,112 documents, 12,035 features (99.41% sparse) and 0 docvars.
##        features
## docs    > 50 chanc limit warm 2 ° c recent scenario
##   text1 1  1     1     2    1 1 1 1      1        1
##   text2 0  0     0     0    0 0 0 0      0        0
##   text3 0  0     0     1    0 0 0 0      1        0
##   text4 0  0     0     1    0 0 0 0      0        0
##   text5 0  0     0     0    0 0 0 0      0        0
##   text6 0  0     0     0    0 0 0 0      0        0
## [ reached max_ndoc ... 1,106 more documents, reached max_nfeat ... 12,025 more features ]
```

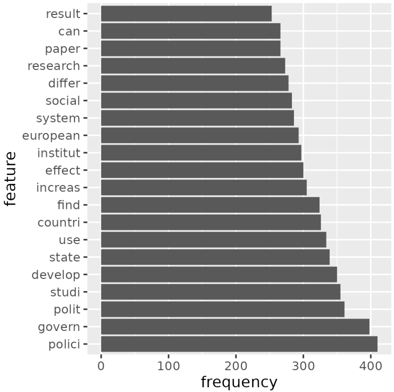
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
vectorizer = CountVectorizer(stop_words="english")
df = df[pd.notna(df["abstract"])].reset_index(drop=True)
dfm = vectorizer.fit_transform(df["abstract"])
dfm
```

Most common features

`quanteda.textstats::textstat_frequency()` gives us the frequency of each term in the corpus.

```
library(quanteda.textstats)
tfreq <- dfmat %>% textstat_frequency() %>% head(20)
tfreq$feature <- factor(tfreq$feature, levels=tfreq$feature)
ggplot(tfreq, aes(x=frequency, y=feature)) +
  geom_col()
```

```
ggsave("plots/top_terms_gg.png", width=4, height=4)
```

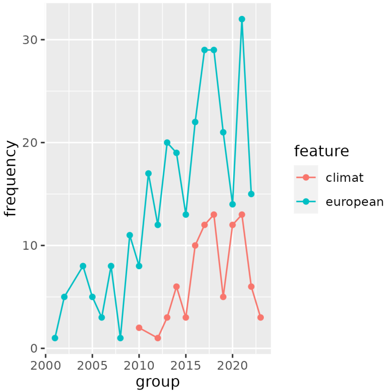


Common features in subgroups

We can also get the frequency of features per subgroup

```
ytfreq <- dfmat %>%
  textstat_frequency(groups=df$publication_year)
ytfreq$group <- as.numeric(ytfreq$group)
interesting_features <- ytfreq %>%
  filter(feature %in% c("european","climat"))

ggplot(
  interesting_features,
  aes(x=group, y=frequency, colour=feature)
) +
  geom_point() +
  geom_line() +
  theme_bw()
```

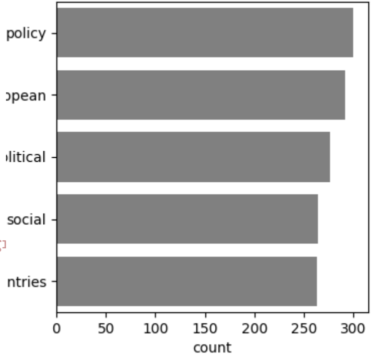
```
ggsave("plots/top_terms_time.png", width=4, height=4)
```



Most common features in Python

In pandas we can make a dataframe of the sum of each column and the feature names

```
counts = dfm.sum(axis=0).A1
tidy_dfm = pd.DataFrame({
    "count": counts,
    "feature": vectorizer.get_feature_names_out()
}).sort_values("count",ascending=False).reset_index(drop=True)

fig, ax = plt.subplots(figsize=(4,4))
sns.barplot(data=tidy_dfm.head(), x="count", y="feature", color="gr

plt.savefig("plots/top_terms_sns.png")
```

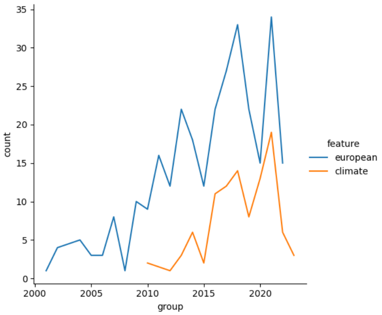


Common features in subgroups in Python

Summing the features per subgroup in Python simply requires some low-level arithmetic and indexing

```
tidy_dfm = pd.DataFrame()
features = vectorizer.get_feature_names_out()
for name, group in df.groupby("publication_year"):
    counts = dfm[group.index,:].sum(axis=0).A1
    group_df = pd.DataFrame({
        "count": counts,
        "feature": features,
        "group": name
    })
    tidy_dfm = pd.concat([
      tidy_dfm,
      group_df[group_df["count"]!=0]
    ]).reset_index(drop=True)

interesting_features = tidy_dfm[
  tidy_dfm["feature"].isin(["climate","european"])
]
sns.relplot(
  data=interesting_features, x="group", y="count",
  hue="feature", kind="line"
)
```

```
plt.savefig("plots/top_terms_time_sns.png")
```



Comparing subgroups

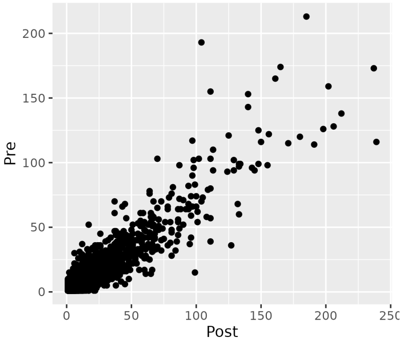
If we want to compare two subgroups directly, we might plot one against the other

```
library(quanteda.textstats)
df$era <- ifelse(df$publication_year<2017, "Pre", "Post")

ytfreq <- dfmat %>% textstat_frequency(groups=df$era) %>%
  pivot_wider(id_cols=feature, names_from=group, values_from=freque

ggplot(ytfreq, aes(x=Post, y=Pre)) +
  geom_point() +
  coord_fixed()
```

```
## Warning: Removed 8427 rows containing missing values (geom_point
```

```
ggsave("plots/scattertext_gg.png", width=4, height=4)
```

```
## Warning: Removed 8427 rows containing missing values (geom_point
```



Long vs wide data

We often need to rely on the `tidyr::pivot_wider()` and `tidyr::pivot_longer()` functions (formerly `spread()` and `gather()`) to get data into the format we need.

```
dfmat %>% textstat_frequency(groups=df$era) %>%
  head()
```

```
##   feature frequency rank docfreq group
## 1   studi       239    1     176  Post
## 2  polici       237    2     165  Post
## 3 develop       212    3     149  Post
## 4     use       206    4     172  Post
## 5   polit       202    5     162  Post
## 6    find       198    6     177  Post
```

```
dfmat %>% textstat_frequency(groups=df$era) %>%
  pivot_wider(id_cols=feature, names_from=group, va
  head()
```

```
## # A tibble: 6 x 3
##   feature  Post   Pre
##   <chr>   <dbl> <dbl>
## 1 studi     239   116
## 2 polici    237   173
## 3 develop   212   138
## 4 use       206   128
## 5 polit     202   159
## 6 find      198   126
```

```
dfmat %>% textstat_frequency(groups=df$era) %>%
  pivot_wider(id_cols=feature, names_from=group, values_
  pivot_longer(cols=Post:Pre, names_to="group") %>%
  head()
```

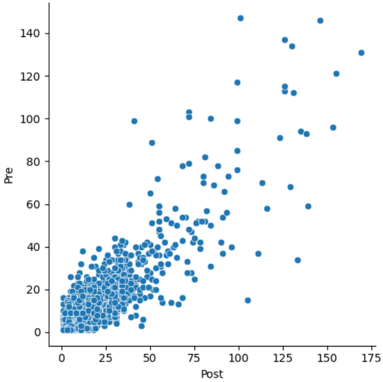
```
## # A tibble: 6 x 3
##   feature group value
##   <chr>   <chr> <dbl>
## 1 studi   Post    239
## 2 studi   Pre     116
## 3 polici  Post    237
## 4 polici  Pre     173
## 5 develop Post    212
## 6 develop Pre     138
```

Comparing subgroups

In Pandas the functions we need to switch between wide and long data are `pivot_table()` and `melt()`

```
import numpy as np
df["era"] = np.where(df["publication_year"]<2017, "Pre", "Post")
tidy_dfm = pd.DataFrame()
features = vectorizer.get_feature_names_out()
for name, group in df.groupby("era"):
    counts = dfm[group.index,:].sum(axis=0).A1
    group_df = pd.DataFrame({
        "count": counts,
        "feature": features,
        "group": name
    })
    tidy_dfm = pd.concat([
      tidy_dfm,
      group_df[group_df["count"]!=0]
    ]).reset_index(drop=True)
wide_dfm = tidy_dfm.pivot_table(
  index="feature", columns="group", values="count"
).reset_index().reset_index(drop=True)
sns.relplot(data=wide_dfm, x="Post", y="Pre")
```

```
plt.savefig("plots/scattertext_sns.png")
```

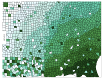


Using colour

Colour is another great way to convey information, and colorbrewer.org tells you all about colour scales, of which there are three kinds:

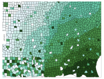
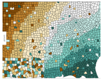
Using colour

Colour is another great way to convey information, and colorbrewer.org tells you all about colour scales, of which there are three kinds:

- **Sequential**  colourscales show differences in magnitude of a continuous variable

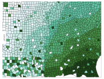
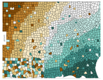
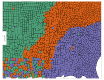
Using colour

Colour is another great way to convey information, and colorbrewer.org tells you all about colour scales, of which there are three kinds:

- **Sequential**  colourscales show differences in magnitude of a continuous variable
- **Diverging**  colourscales show *symmetrical* differences in magnitude either side of a meaningful central point

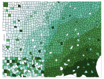
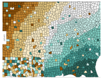
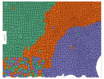
Using colour

Colour is another great way to convey information, and colorbrewer.org tells you all about colour scales, of which there are three kinds:

- **Sequential**  colourscales show differences in magnitude of a continuous variable
- **Diverging**  colourscales show *symmetrical* differences in magnitude either side of a meaningful central point
- **Qualitative**  colourscales shows different categories where there one category is neither greater than nor less than another

Using colour

Colour is another great way to convey information, and colorbrewer.org tells you all about colour scales, of which there are three kinds:

- **Sequential**  colourscales show differences in magnitude of a continuous variable
- **Diverging**  colourscales show *symmetrical* differences in magnitude either side of a meaningful central point
- **Qualitative**  colourscales shows different categories where there one category is neither greater than nor less than another

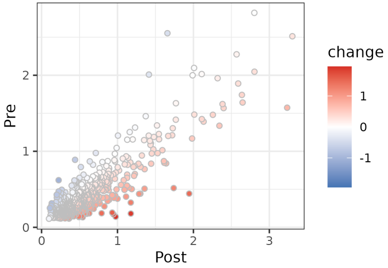
PAY ATTENTION! to the colorblind-safe filter. A large proportion of people have reduced or no color discrimination along the red-green axis.

Using colour II

```
ytfreq <- dfmat %>% dfm_weight(scheme="prop") %>%
  textstat_frequency(groups=df$era) %>%
  filter(docfreq>10) %>%
  pivot_wider(
    id_cols=feature,
    names_from=group,
    values_from=frequency
  )

ytfreq$change <- log(ytfreq$Post / ytfreq$Pre)
max_change <- max(abs(ytfreq$change), na.rm=TRUE)

p <- ggplot(ytfreq, aes(x=Post, y=Pre, fill=change)) +
  geom_point(color="grey", shape=21) +
  coord_fixed() +
  scale_fill_gradientn(
    colors = c("#4575b4","white","#d73027"),
    values = scales::rescale(c(max_change*-1,0,max_change)),
    limits = c(max_change*-1,max_change)
  ) +
  theme_bw()
p
```

```
ggsave("plots/scattertext_gg_2.png", width=4, height=3.5)
```

In this plot we get the *proportion* of documents from each group each term occurs in. We represent the **change** from one era to another as a symmetrical variable either side of 0, and colour the points on an appropriate **diverging** scale.

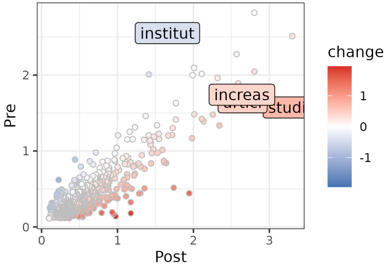


Adding labels

We can add labels so we know what the points represent, but these often get in the way of readability

```
#ytfreq <- ytfreq >max_value <- max(ytfreq$Post_2017, ytfreq$Pre_2(
labels <- ytfreq %>% rowwise() %>%
  mutate(max_value = max(Post,Pre)) %>%
  filter(
    (abs(change)>0.4 & max_value>2.5)
  )

p + geom_label(data=labels, aes(label=feature))
```

```
ggsave("plots/scattertext_gg_3.png", width=4, height=3.5)
```



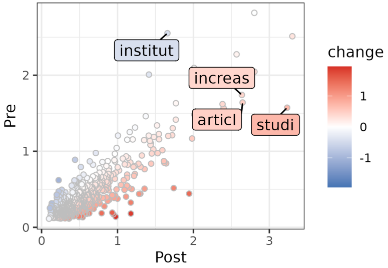
Adding labels with ggrepel

```
library(ggrepel)

labels <- ytfreq %>% rowwise() %>%
  mutate(max_value = max(Post,Pre)) %>%
  filter(
    (abs(change)>0.4 & max_value>2.5)
  )

p + geom_label_repel(
  data=labels,
  aes(label=feature),
  min.segment.length = 0
)
```

```
ggsave("plots/scattertext_gg_4.png", width=4, height=3.5)
```

We can add labels so we know what the points represent, but these often get in the way of readability

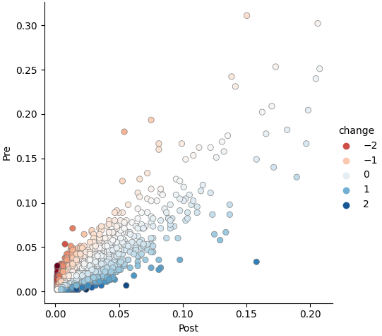
ggrepel allows us to put labels in positions that maintain readability



Color with Python

```
tidy_dfm = pd.DataFrame()
for name, group in df.groupby("era"):
    counts = np.count_nonzero(
      dfm[group.index,].A, axis=0
    ) / group.shape[0]
    group_df = pd.DataFrame({
        "count": counts,
        "feature": features,
        "group": name
    })
    tidy_dfm = pd.concat([
      tidy_dfm, group_df[group_df["count"]!=0]
    ]).reset_index(drop=True)
wide_dfm = tidy_dfm.pivot_table(
  index="feature", columns="group", values="count"
).reset_index().reset_index(drop=True)

from matplotlib.colors import CenteredNorm
import matplotlib.cm as cm
colormap = cm.RdBu
norm = CenteredNorm()
wide_dfm["change"] = np.log(wide_dfm["Post"] / wide_dfm["Pre"])
sns.relplot(
    data=wide_dfm, x="Post", y="Pre", hue="change",
    palette=colormap, norm=norm, edgecolor="grey"
)
```

We can do the color rescaling much more easily with matplotlib (which we use to tweak seaborn)



Color with Python

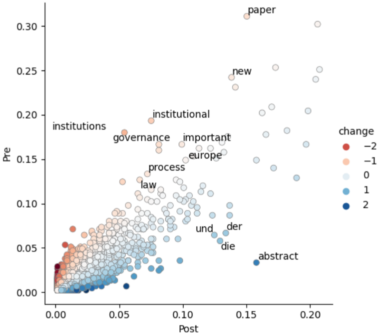
```
labels = wide_dfm[
    (abs(wide_dfm["change"])>0.5) &
    (wide_dfm["Post"]+wide_dfm["Pre"]>.18)
]

from adjustText import adjust_text

scatter = sns.relplot(
    data=wide_dfm, x="Post", y="Pre", hue="change",
    palette=colormap, norm=norm, edgecolor="grey"
)
ax = scatter.ax
texts = []
for i, row in labels.iterrows():
    texts.append(ax.text(row["Post"], row["Pre"], row["feature"]))

adjust_text(texts)
```

500

```
plt.savefig("plots/scattertext_sns_3.png")
```

We can do the color rescaling much more easily with matplotlib (which we use to tweak seaborn)

To arrange text labels nicely we can use `adjustText`, which works like ggrepel.



Wrapup and outlook

Wrapup

Today we strengthened our data our data management skills, and had a refresher on ggplot2 / seaborn / matplotlib.

Getting data into the right format and plotting it is one of the *most import skills* as a data scientist!

The plotting libraries are much bigger than what we can cover, but you have enough to get started and extend by **reading the documentation**.

Outlook

Next week we'll be getting more technical. We'll look at ways of measuring similarity and at how we can do dimensionality reduction.

Homework

I will send you the homework assignment after class. This is due by 11:59 on 13 October.