Initial Remarks

This paper contains a final project made in collaboration with Daniel Kasper (202009150) and six individual assignments made by me.

The overall portfolio will be included in this .pdf file, but all individual assignments and the final project can be found and replicated in my GitHub repository: https://github.com/Digital-Methods-HASS/au634054_joshua_hansen

Food for Thought: An investigation of food waste habits in major Danish Cities

Aarhus University - Institute for Communication and Culture

Lecturer: Adéla Sobotkova

GitHub: The repositories containing all relevant data and code are available here:

https://github.com/Digital-Methods-HASS/au634054_joshua_hansen

https://github.com/Digital-Methods-HASS/au681244_kasper_daniel

Authors Hansen, J. & Kasper, D.

Keywords: *Food Waste; Salling Group; API; Python; Pandas; Seaborn*

A - Free text

1 Introduction

According to the Danish Veterinary and Food Administration approximately 1.214.000 tons of food are being thrown out each year, where two thirds originate from places in the food business and the rest comes from danish homes (Bureau Veritas, n.d.). To reduce the amount of food waste, Salling Group writes in their API Documentation:

“Salling Group has a lot of food on sale or about to expire while a lot of customers may not be aware of it. To help reduce food waste in Denmark, our food waste API provides information about food on sale or soon to be expired in stores.” (Salling Group, n.d.)

The amount of food wasted each year, reveals an opportunity for investigation. By highlighting food waste habits, this paper aims to provide insight to the trends of food waste in 5 major, Danish cities throughout the course of 10 days. We will request data on products that are currently on sale in supermarkets owned by Salling Group, which includes the following: Føtex, Netto, Basalt and Bilka (Salling Group, n.d.). After collecting data from the postal codes of each city, we aim to visualize specific food waste habits for each individual city. We therefore seek to investigate the following research question:

What type of food is mainly put to waste and is there a notable difference in food waste habits for Danish cities?

We will be providing a short explanation of the cultural relevance and motivation regarding the project and furthermore listing the software framework in which our data acquisition, processing and visualization is applied. After providing an overview of the data wrangling and visualization, we aim to analyze and describe tendencies occurring in the different postal codes. By using Salling Group’s categorisations of each individual product, a comparison of the cities' tendencies will be made. To supply the analysis of the data, a critical evaluation of our findings will be included to discuss the methodological approaches used. Here we seek to highlight strengths and weaknesses of our methodological approaches in this project.

2 Problems and Background (*Joshua*)

Stop Wasting Food movement Denmark (Stop Spild Af Mad) has been Denmark’s largest non-profit movement against food waste since its foundation in 2008. The organization supports and collaborates with over 60.000 people both nationally and internationally. The organization actively works with implementation of UN Sustainable Development Goal 12.3 (FAO, n.d.; Stop Wasting Food, n.d.). In their section about food waste traps, they specifically mention to buy food that is close to its expiration date, in order to reduce the supermarkets food waste (Stop Wasting Food, n.d.). By visualizing through digital methods the food categories that are thrown out the most, we hereby wish to contribute to minimize supermarkets food waste and implement UN Sustainable Development Goal 12.3.

3 Software Framework *(Joshua and Daniel)*

The code written for this project is available for replication on both Windows 11 and MacOS Ventura 13.1. We wrote the code for this project on a 3 year old Acer Aspire 5, 8 GB RAM, which runs Microsoft Windows 11 Home. The code was also tested on a Macbook Pro 2021 M1 Pro, 8-core CPU with 6 performance cores and 2 efficiency cores, 14-core GPU, 16-core Neural Engine and 200GB/s memory bandwidth. The software used for processing code is Python (3.8.14) based on the Jupyter Notebook (6.4.12) browser solution. Jupyter Notebook was used as our integrated development environment (IDE) and was used in the environment of Anaconda (v. 4.10.3) The following python libraries used were:

- ➢ Requests (Chandra & Varanasi 2015)
- ➢ JSON (Pezoa et al., 2016)
- ➢ Pandas (McKinney et. al., 2010)
- ➢ Matplotlib (Hunter, 2007)
- ➢ Seaborn (Waskom et al., 2017)

4 Data Acquisition and Processing (*Joshua*)

To acquire our data, we used the official Salling Group API, specifically their Anti Food Waste API, which consists of two endpoints in which you can retrieve data. This can either be specified through geographical coordinates, postal codes or a specific store ID. It is only possible to retrieve real-time data and not archived data, which meant that we had to request data manually each given day. We chose the API endpoint, which requests data by postal codes and with a timescope of ten days. We set our timescope to ten days, because we considered the amount of data entries retrieved over the given period to be a better subject for analysis. We chose to request the data with the programming language Python and write our own code to both retrieve, process and visualize the data. In order to retrieve data from the Salling Group API, it is a requirement to register an account on their developer-page to receive an authentication token for API requests.

The data responses consist of JSON (Javascript Object Notation), that include a list of price reduced food. The object includes an ‘offer’ section, that describes the discount together with valid time and pieces in stock. Furthermore it includes a ‘product’ section that describes the food item with an optional image. The data we need is thereby nested within multiple lists and dictionaries, which requires an extensive pre-processing of the data responses. We created a function that makes an individual data request with a required parameter that specifies the postal code. The function requests data and pre-processes it into a dataframe, by using methods of the pandas library and we refer to our processing script for further explanation on individual steps.

We considered a pandas dataframe to be most sufficient for further analysis and visualization of the data, as it is a structured and well-documented data type (McKinney et. al., 2010). All dataframes are saved to a .csv file, which gives us a single .csv file for each day of each postal code. In the end we acquired 70 individual .csv files. After requesting data for ten days, it was required to concatenate all the individual .csv files into a complete dataframe for each postal code and one dataframe that contains all data entries of all postal codes.

After acquiring and preprocessing the data, certain data cleaning was required. The column we were interested in, “description”, was a string of multiple descriptions and special characters in both English and Danish. We had to use regular expressions and string methods to splice and replace the string in order to get the labeled categories of the product. We were able to clean the data, so we ended up having three columns that include an overall section of the product, the type of product and a specified category of the product.

Lastly, we used the cleaned dataframes to visualize certain trends and meta-data of our findings, where we focused on the categorisation of the products.

5 Empirical Results (*Joshua and Daniel*)

5.1 - Meta-data foundation for empirical analysis

Our empirical results consist of a dataframe with 4175 data entries from all five cities throughout ten days. Copenhagen has the most data entries with 2073, following Aarhus with 907, Esbjerg with 830, Aalborg with 792 and lastly Odense with 404 entries.

An interesting finding is the entries for Odense. It is the third biggest city in Denmark, but attracts our attention since it contains the least food waste entries. This result could have multiple arbitrary reasons. It could be because of the specific timespan, or could indicate that there are less Salling Group stores around that area. The high number of entries for Copenhagen is not surprising, as it is the biggest city in Denmark. We also used three different postal codes from the city, in order to encapsulate the scale of the capital. For our analysis, we want to visualize our dataset based on Sallings meta-data, specifically their category labels and their percentage discount. We evaluate those to be the most telling and interesting in terms of understanding certain food waste tendencies. By splitting the data, we acquired three different types of categories, which we interpreted as: ‘overall category’, ‘type of product’ and ‘specific product’, where every category has a more thorough description of the product.

5.2 - Visualizations and findings

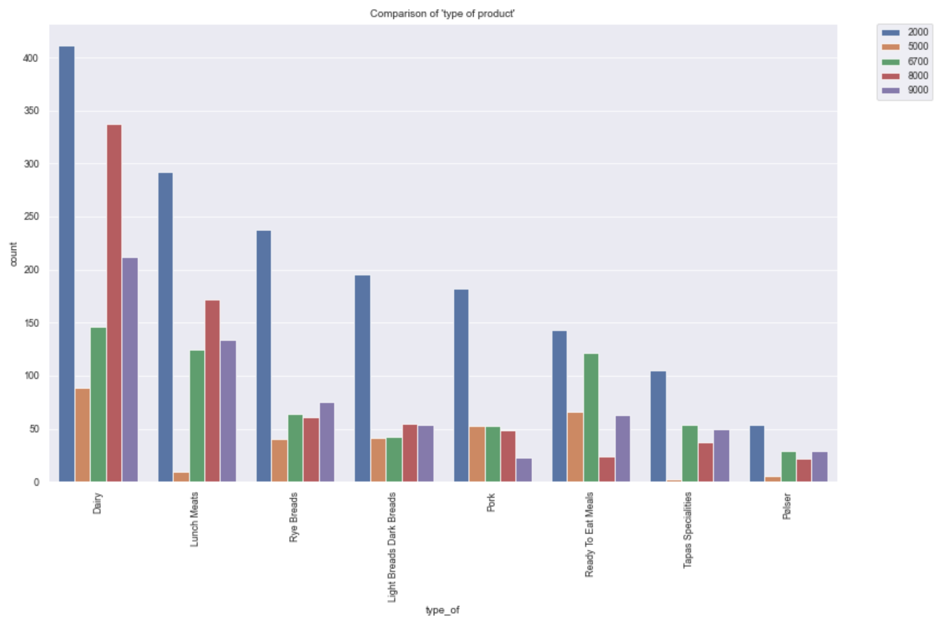


fig. 1: comparison of top 8 type of products and value count for each city

The first barchart presents a comparison for the eight most used ‘overall categories’ in the data set and how often these categories are represented for each city. The first type of product‘ Dairy’ is by far the most wasted food in each city. The second type is ‘Lunch Meats’ and is the second biggest for most of the cities except for Odense, where it is placed seventh. In general there is a trend for dairy, meat and bread to be the products wasted most frequently. This makes sense in terms of the products’ short expiration date. One interesting type of category we want to highlight is the ‘Ready to Eat Meal’, which is the sixth highest category of food thrown out. We would not consider ‘Ready to Eat Meals’ being the same category as dairy, bread and meat products regarding their expiration date, which makes it stand out in our analysis. The ‘Ready to Eat Meal” category is a clear outlier in our data and could indicate a cultural tendency in terms of the product, which is usually considered to be unhealthy. This interpretation could explain its appearance in the top categories. It is also worth considering the product's stock quantity and other variables, which may affect the results. Regardless of the reason, this could be a tendency that is worth investigating further.

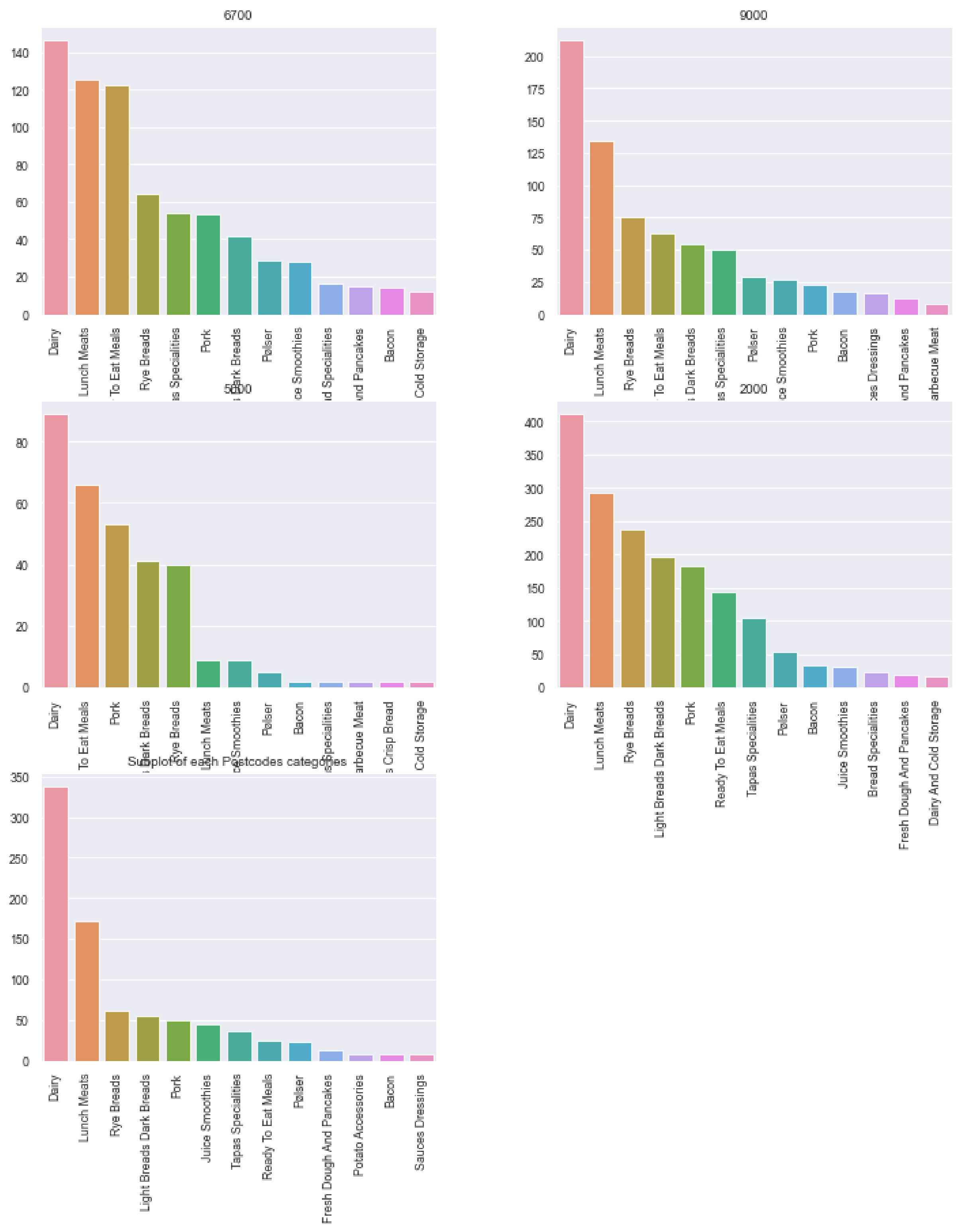


fig. 2: Subplot of first 13 categories for each city

To include a better overview for each of the cities' types of products, we created a subplot with all categories for each city. The cities show a similar distribution of each product category that is thrown out. There are no visible outliers in any city, which shows consistency in what is thrown out by each city. An interesting finding is that Odense and Esbjerg are the only cities that have ‘ready to eat meals’ as their second and third highest product type to be wasted. Furthermore, the similarity of products being thrown out in each city is an interesting pattern, as each city has dairy, meats and bread as top contenders for food wasted. This suggests a larger, structural pattern in what Danish supermarkets usually have to throw out.

To continue our analysis, we want to highlight the high amount of short expiration date food that is thrown out. Here we have visualized a bar plot with the specific products that are thrown out the most. ‘Meat’, ‘Cheese’, ‘Seeded Rye Breads’, ‘Dinner Meals’ and ‘Light Dark Buns’, ‘Yoghurt Soured Milk Products’, ‘Toast Breads’ and ‘Mayo Salads’ are the most prominent. It is interesting to consider that these products are thrown out the most, as they have recently been featured on a list of products with the highest price rises with the current inflation rates (Paulsen, 2022). This could indicate an interesting causality which stores should consider when creating discounts.

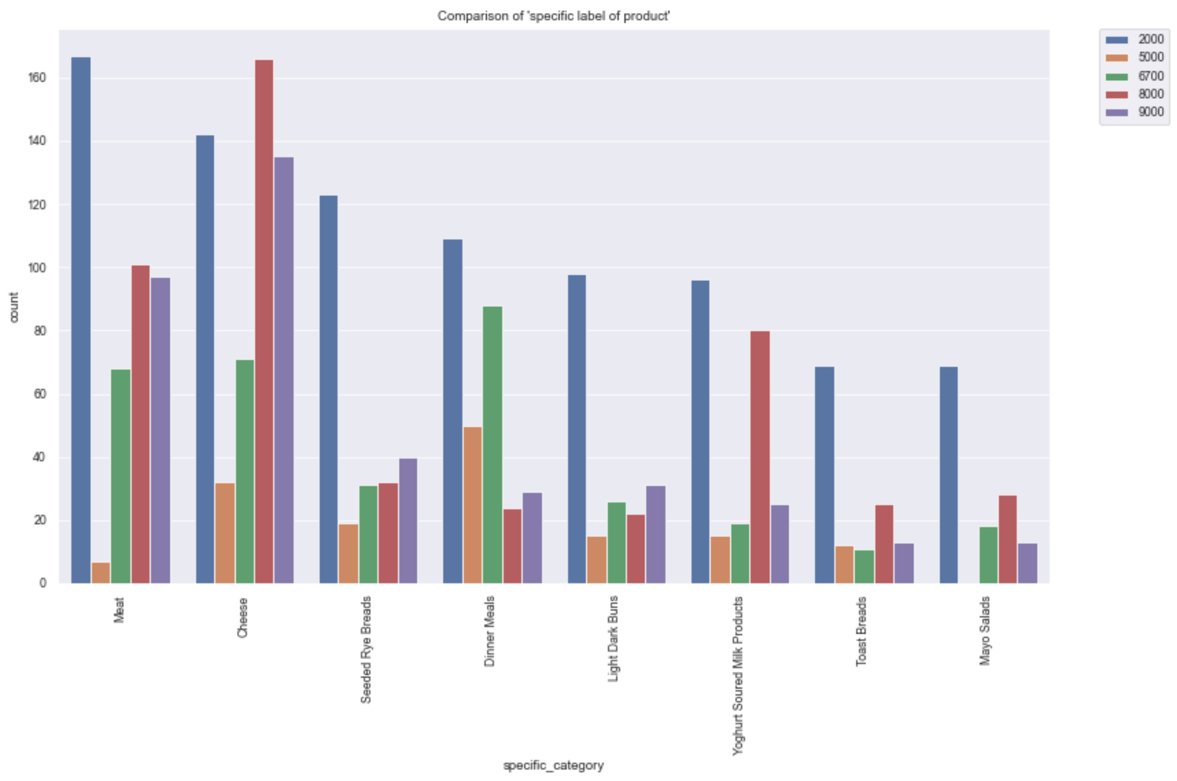


fig. 3: comparison of top 8 specific products and value count for each city

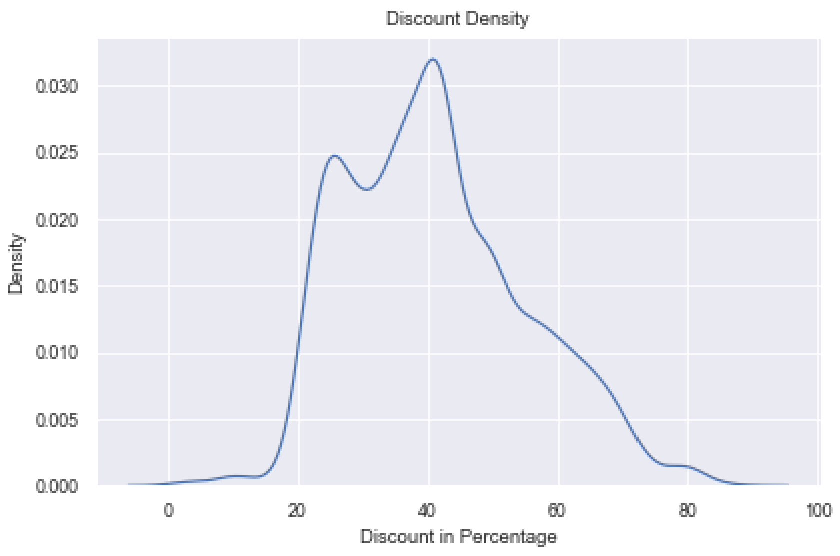


fig. 4: Discount Density, showing a distribution of the discount percentages of the products

The last chart displays the discount density of the products which are nearing, or has surpassed, their expiration date. The chart illustrates the occurrences of discount-percentage in which a product's original price is reduced. From interpretation of our data we can assume that most products are not heavily reduced in price, when nearing or surpassing their expiration date. Rather, the graph shows that most discounts are around 20 to 40 percent, almost emulating a normal distribution. This is a result which we did not initially anticipate in our findings, since we went into our research with a presumption about high discounts on nearly expired food. Furthermore, looking back at our previous visualizations, we observed dairy, meat and bread to be the most prominent products to go to waste. We find it rather paradoxical, that discount densities are not distributed higher, percentage-wise, when the most prominent food going to waste is products which easily spoil, when past their expiration date.

5.3 Processing Salling Group API

When processing our data, we had to decide what our focal point for analysis would be. Pulling and processing requests from the Salling Group API's food waste data produced various columns in our data frame each consisting of different information about the stores, products, pricings, ean-numbers etc. In visualizing the data, we chose to focus on the product categories of the food wasted. This decision created obstacles for our further work, since the category-column contained a lot of concatenated information. It was a messy column. In our source code, under the preprocessing headline, we display step by step how to split the category-data from each respective postal code's data frame. We chose to work further with this section of our data because we consider it applicable to larger visualizations regarding specific food categories wasted, as seen in our visuals above.

Lastly, the scope of the data for this project only represents a specific timeframe and spatial distribution. This means that the empirical results, when brought to a larger scale, might differ from what is originally conceived in this paper. We will further evaluate the strengths and weaknesses of our methodological approach in the next section.

6 Critical evaluation and limitations of our research (*Daniel*)

We will now explore the limitations of our project and how it has affected our results. We identified three major limitations regarding the research project: The first limitation is the data applicability. The API is only applicable to singular postal codes, but major cities in Denmark often consist of multiple postal codes. Aarhus, as an example, is divided into multiple postal codes such as 8000 Aarhus C, 8200 Aarhus N and 8210 Aarhus V. As a result of this, a definition of specific data limitation was needed. When choosing the amount of postal codes to request data from, an assessment of the representation from the different major cities had to be defined. We choose the respective city centers' postal codes as our main point of data acquisition, as to not create unnecessary complexity and confusion in the data acquisition.

The second limitation to our method of acquisition is the variety of stores, in which the API is applicable. As stated earlier, the Salling Group API only pulls data from their respective stores. This is an important consideration regarding our empirical evidence, since it only partly displays food waste tendencies for Danish supermarkets. Other supermarket conglomerates may display different patterns than those of Salling Group, and therefore our results do not account for absolute empirical evidence, but rather a contribution for further investigation.

The third limitation we identified was the timeframe of datasets and how it may affect future reproducibility. The Salling Group API only offers acquisition of real time data as opposed to requesting archived data. This raises considerations about future reproducibility. By applying

Tedersoo et al. (2021) and their thoughts on reproducible research, we aim to create transparency and availability of our meta-data used for the project. Our wish is to avoid “data availability upon request” (Tedersoo et al., 2021, p. 9) by sharing the meta-data in our repositories, accompanied with a step by step reproducibility guide, our specified dates will be available for further research, re-analysis and to display the authenticity of the method of acquisition. It is worth noting that the new API request will differ from our current empirical data, which is the reasoning for providing the datasets in our repositories.

7 Conclusions

This project sought to investigate what type of food is mainly put to waste in different Danish cities. By using the Salling Group API, data from seven different postal codes were acquired, processed, analyzed and visualized for further investigation. The processing and visualization of our data was done with the programming language Python and the pandas library, which made it possible to structure and analyze our acquired data. By examining the product-categories of the Salling Group API, we were able to compare postal codes and general food waste tendencies. Through our analysis, we found general patterns and tendencies regarding dairy, meat and bread being the most wasted type of food product. Furthermore, we observed a peculiar outlier in the “Ready to Eat Meals” which became subject for interpretation. There was an overall similarity for each city and the products wasted, which could suggest a structural pattern for Danish supermarkets and their food waste habits. We also observed a causality between the specific products most thrown out and rising inflation prices. Lastly, we identified three key limitations for our project: *data applicability, method of acquisition* and *timeframe,* with considerations to what effect it may have had on the final empirical results.

Acknowledgements

We would like to thank Salling Group for making their API available for public access. This has helped our project tremendously, since it allows for real-time application of food waste data.

References

- Bureau Veritas. (n.d.). *Food waste* | *Denmark*. Bureau Veritas. Retrieved December 29, 2022, from https://www.bureauveritas.dk/en/needs/food-waste
- Chandra, R. V., & Varanasi, B. S. (2015). *Python requests essentials*. Packt Publishing Ltd.
- FAO. (n.d.). *12.3.1 Global food losses* | *Sustainable Development Goals*. Food and Agriculture Organization of the United Nations. Retrieved December 29, 2022, from https://www.fao.org/sustainable-development-goals/indicators/1231/en/
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science &amp; Engineering*, *9*(3), 90–95.
- McKinney, W., & others. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, pp. 51–56).
- Paulsen, S. B. (2022, July 11). *Overblik: Disse fødevarer er steget mest i pris i 2022*. TV 2 Nyheder. Retrieved January 5, 2023, from https://nyheder.tv2.dk/penge/2022-07-11-se-listen-disse-foedevarer-er-steget-mest-i-pris
- Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, Martin, & Virgo
  c, Domagoj. (2016). Foundations of JSON schema. In *Proceedings of the 25th International Conference on World Wide Web* (pp. 263–273).
- Salling Group. (n.d.). *Salling Group developer website*. Salling Group developer website. Retrieved December 29, 2022, from https://developer.sallinggroup.com/api-reference
- Stop Wasting Food. (n.d.). *Avoid Food Waste Traps*. Stop Wasting Food. Retrieved December 29, 2022, from https://stopwastingfoodmovement.org/food-waste/avoid-food-waste-traps/
- Tedersoo, L., Küngas, R., Oras, E. et al. Data sharing practices and data availability upon request differ across scientific disciplines. Sci Data 8, 192 (2021). https://doi.org/10.1038/s41597-021-00981-0
- Waskom, M., Botvinnik, Olga, Kane, Drew, Hobson, Paul, Lukauskas, Saulius, Gemperline, David C, Qalieh, Adel. (2017). *mwaskom/seaborn: v0.8.1 (September 2017)*. Zenodo. https://doi.org/10.5281/zenodo.883859

B- Required Metadata

Table 1 – Software metadata

Nr	**Software metadata description**	
S1	Current software version	Python v3.8.14, Jupyter notebook v6.4.12
S2	Permanent link to Github repository where you put your script or R project	https://github.com/Digital-Methods-HASS/au681244_kasper_daniel & https://github.com/Digital-Methods-HASS/au634054_joshua_hansen
S3	Legal Software License	3-Clause BSD License (https://jupyter.org/governance/projectlicense.html)
S4	Computing platform / Operating System	Microsoft Windows 11 or Mac OS 12.5
S5	Installation requirements & dependencies for software not used in class	You need to have Python 3.6 installed
S6	If available Link to software documentation for special software	Salling Group API documentation: https://developer.sallinggroup.com/api-reference Matplotlib documentation: https://matplotlib.org/stable/api/index
S6	Support email for questions	For inquiries please contact: 202009150@post.au.dk or 201905780@post.au.dkk

Table 2 – Data metadata

Nr	**Metadata description**	**Content**
D1	Our data consists of multiple .csv files for each day for 7 postal codes in total. The total amount of .csv files is: 70 .csv files The python script named the files based on this formula: *data_foodwaste_postalcode{postcode}_{currentDate}.csv*	Food waste products that are currently on discount and close to being thrown out. Every file consists of the same columns*: currency, discount, ean_x, endTime, lastUpdate, newPrice, originalPrice, percentDiscount, startTime, stock, stockUnit, categories, description, ean_y, image.* Data entries vary for each day, but the total amount of data entries over 10 days and 7 postcodes is: ***4175***. **No. of days included:** *06.12.2022 + 08.12.2022 + 11.12.2022 + 12.12.2022 + 13.12.2022 + 14.12.2022 + 15.12.2022 + 17.12.2022 + 18.12.2022 + 19.12.2022* **Postal codes included:** *5000, 6700, 8000, 9000, 2100, 2200, 2300*

Regular expressions and spreadsheets homework 1

Joshua Hansen

2022-08-30

1. What regular expressions do you use to extract all the dates in this blurb: http://bit.ly/regexexercise2 and to put them into the following format YYYY-MM-DD ?

Answer to 1): https://regex101.com/r/um8qMj/1

2. Write a regular expression to convert the stopwordlist (list of most frequent Danish words) from Voyant in http://bit.ly/regexexercise3 into a neat stopword list for R (which comprises "words" separated by commas, such as http://bit.ly/regexexercise4). Then take the stopwordlist from R http://bit.ly/regexexercise4 and convert it into a Voyant list (words on separate line without interpunction)

From Voyant to List: https://regex101.com/r/tUR6bA/1

From List to Voyant: https://regex101.com/r/tUR6bA/2

3. In 250 words, answer the following question: "What are the basic principles for using spreadsheets for good data organisation?"

Answer 3): A good practice would first of all to be consistent with the data. You should always try to follow through with any conventions you have so you can manipulate all data entries at once. That means that for each column there should only be one data type and one way of spelling. You should also try to make the data entries computer-readable for further analysis. Colors are e.g., not readable in R and should be conveyed differently.

You should always have a documentary on each column or entry on what the data type is, what it means and describe the context of it. You could resolve some issues by implementing a data validation, especially when collaborating with others. One Column should also only have one value or information and each entry should have an individual keyID. It is also good to not have multiple pages in one sheet as the computer will only export a single page.

In general you should try to make every data entry understandable for a human and provide intuitive description and documentations of your data. Spreadsheets are good for having an overview of your data or making simple statistics, but they lack versioning and are prone to invalid data entries. It is always recommended to have a changelog that will show any changes made in the data, as raw, captured data will be very valuable and necessary for a complete research.

4. Challenge (OPTIONAL)!Can you find all the instances of 'Dis Manibus' invocation in the EDH inscriptions in https://bit.ly/regexexercise5? Beware of the six possible canonical versions of the Dis Manibus formula (see day 1 slides)!

4) next time 😊

Open Refine homework 2

Joshua Hansen

2022-08-30

1. *Create a spreadsheet listing the names of Danish monarchs with their birth- and death-date and start and end year of reign. Make it *tidy*! They should be sortable by year of birth. Suitable source websites are here and here, but you can also use another source, provided you reference it. (Group collaboration is expected and welcome. Remember to attach this spreadsheet to Brightspace submission)*

For the first task, we took the information of this website and copy/pasted into a spreadsheet. We know had to split the information of the column into fitting and tidy colums. We used a split() method within Google Sheets and split it into the colums: "Regent_name" , "Reign_start" and "Reign_end" , where we put NA as the value for missing information. The page didn´t give us more data then their time of reign, so we only included these two colums.

Result: see appendix

2. *Does OpenRefine alter the raw data during sorting and filtering?*

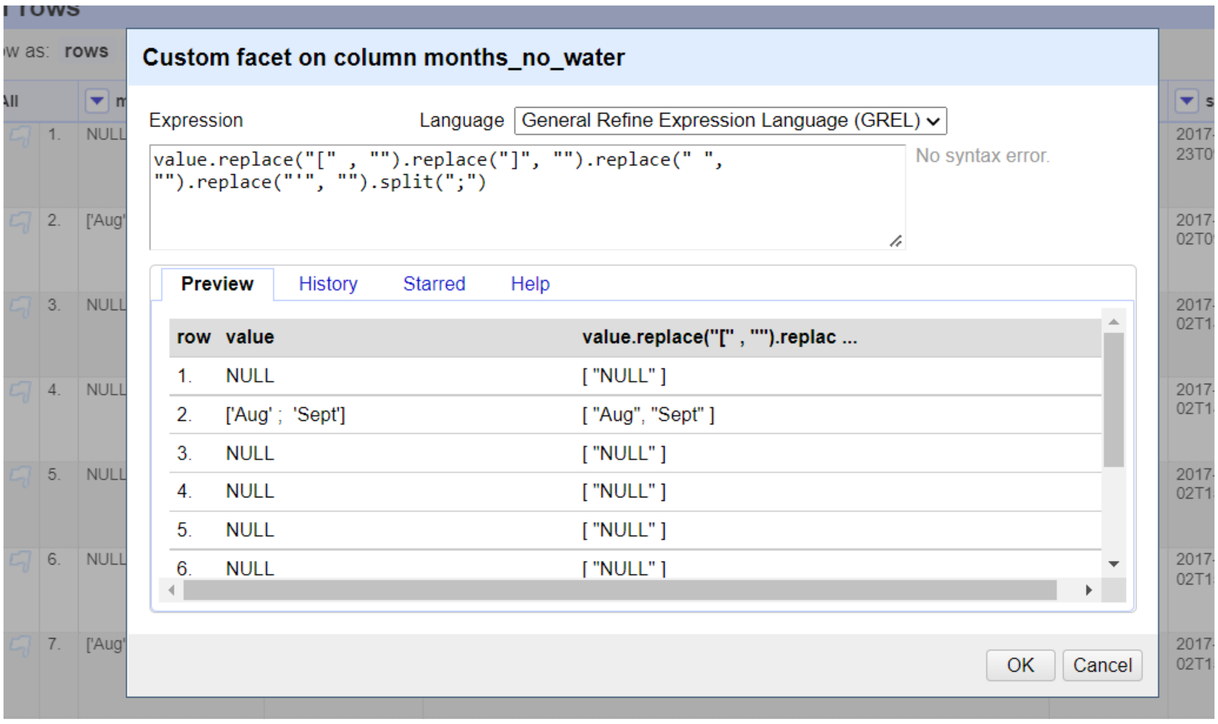
OpenRefine will not alter the raw data when using the sorting and filtering functions. It will only show a preview of your changes on the right site menu, so you will also have an overview of the original data. OpenRefine also includes an automated versioning, so it is always possible to go back to the previous version.

3. *Fix the interviews dataset in OpenRefine enough to answer this question: "Which two months are reported as the most water-deprived/driest by the interviewed farmer households?"*

To figure out, which two months are reported the most water-deprived months according to the farmer households, I first have to find a column with the relevant data. There is a column called "Months_no_water", which probably includes the information I need.

First I moved the column to the start of the data-set for better overview and inspected the data entries with a simple “text-facet”, which shows me a summary of all the data entries and how often they are counted in the data-set.

To fix that, I create a custom text facet where I type this function-call in: “value.replace("[" , "").replace("]", "").replace(" ", "").replace("'", "").split(";")”



Custom facet on column months_no_water

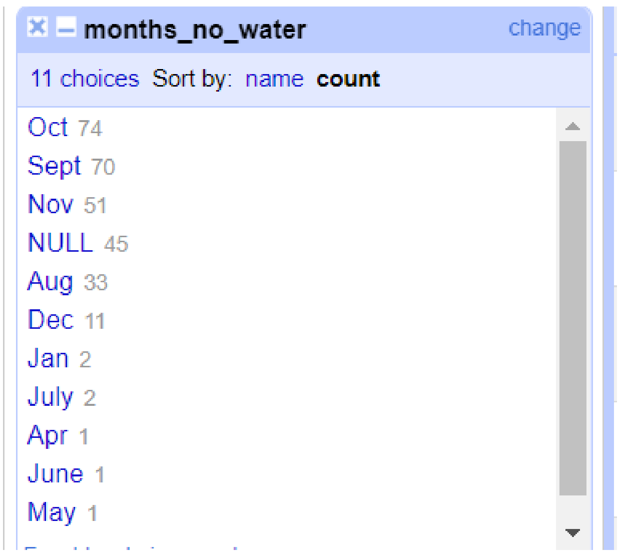
Expression — Language: General Refine Expression Language (GREL)

```
value.replace("[" , "").replace("]", "").replace(" ",
"").replace("'", "").split(";")
```

No syntax error.

row	value	value.replace("[" , "").replac ...
1.	NULL	["NULL"]
2.	['Aug' ; 'Sept']	["Aug", "Sept"]
3.	NULL	["NULL"]
4.	NULL	["NULL"]
5.	NULL	["NULL"]
6.	NULL	["NULL"]

This will now give me a preview of all the values by their own and how often they occur.



months_no_water

11 choices Sort by: name count

Oct	74
Sept	70
Nov	51
NULL	45
Aug	33
Dec	11
Jan	2
July	2
Apr	1
June	1
May	1

Where the answer of the two most water-deprived months is: October and September.

4. *Real-Data-Challenge: What are the 10 most frequent occupations (erhverv) among unmarried men and women in 1801 Aarhus? (hint: some expert judgement interpretation is necessary, look at the HISCO classification "Historical International Standard of Classification of Occupations" on Dataverse if ambitious)*

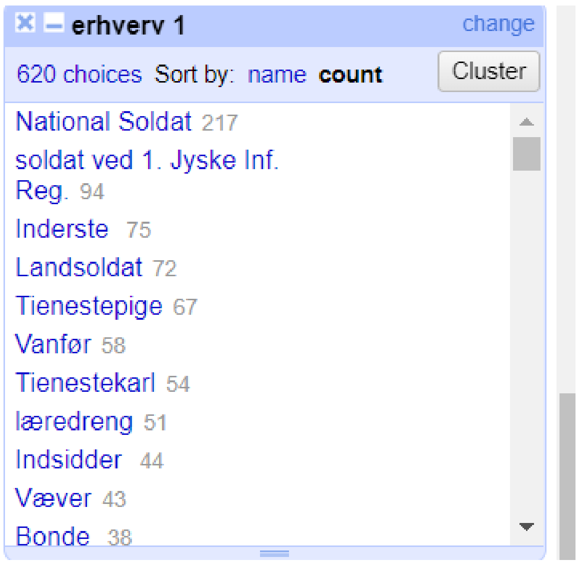
First I put the data-set into OpenRefine and inspected the columns and information.

To answer the question, I need the column “civilstand” and the column “erhverv”. I can already see, that in the column “erhverv” are multiple entries per row sometimes, separated by a comma or the Danish word “og”. So first of all, I would split the column into multiple columns, where I separate them by this regular expression “,|\bog”, which splits it at a comma or a the specific word “og”.

Now I 5 columns, called “erhverv 1-5”, where I just assume that the first occupation mentioned is their main occupation. After making a text facet on each of the new columns, it seems to be valid to assume this, as all the entries of the other columns would not have an effect on the “erhverv 1” column.

To now see, which occupation is the most frequent for unmarried men and women, I have to apply a text filter with the input “ugift”. Now it will only count in the entries with the people that have the status of unmarried.

Now I can make a normal text-facet on the “erhverv 1” column, and sort it by count. I then check the cluster, to see if there are any similar texts or entries, and there were a lot, which I then clustered accordingly. At last I will have a list of the 10 most frequent occupations among unmarried people in 1801 Aarhus:



```
National Soldat 217
soldat ved 1. Jyske Inf. Reg. 94
Inderste 75
Landsoldat 72
Tienestepige 67
Vanfør 58
Tienestekarl 54
læredreng 51
Indsidder 44
Væver 43
Bonde 38
```

Starting with R homework 3

Joshua Hansen

2022-09-24

Instructions: For this assignment, you need to answer a couple questions with code and then take a screenshot of your working environment.

1) Use R to figure out how many elements in the vector below are greater than 2 and then tell me what their sum (of the larger than 2 elements) is.

rooms <- c(1, 2, 4, 5, 1, 3, 1, NA, 3, 1, 3, 2, 1, NA, 1, 8, 3, 1, 4, NA, 1, 3, 1, 2, 1, 7, 1, 9, 3, NA)

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.8     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.1
## v readr   2.1.2     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
# here I assign the data in a vector vector
rooms <- c(1, 2, 4, 5, 1, 3, 1, NA, 3, 1, 3, 2, 1, NA, 1, 8, 3, 1, 4, NA, 1, 3, 1, 2, 1, 7, 1, 9, 3, NA

# omitting missing values

#rooms <- na.omit(rooms) #- could also use this
rooms <- rooms[!is.na(rooms)]


# define vector with values > 2
two_rooms <- rooms[rooms > 2]

# finding the length of the vector - length = amount of rooms that are left with more than two rooms
length(two_rooms)
```

```
## [1] 12
```

```
# here im taking the sum of this vector
sum(two_rooms)
```

```
## [1] 55
```

answer: There are 12 rooms with a value over 2 and their total sum 55

2) What type of data is in the 'rooms' vector?

```
class(rooms)
```

the datatype of the vector rooms is: numeric. That means the data are integers (numbers)

3) Submit the following image to Github: Inside your R Project (.Rproj), install the 'tidyverse' package and use the download.file() and read_csv() function to read the SAFI_clean.csv dataset into your R project as 'interviews' digital object (see instructions in https://datacarpentry.org/r-socialsci/setup.html and 'Starting with Data' section). Take a screenshot of your RStudio interface showing

- see screenshot(https://github.com/Digital-Methods-HASS/au634054_joshua_hansen/blob/main/homework_3/au634056_joshuahansen1.png)

4) Challenge: If you managed to create your own Danish king dataset, use it. If not, you the one attached to this assignment (it might need to be cleaned up a bit). Load the dataset into R as a tibble. Calculate the mean() and median() duration of rule over time and find the three mondarchs ruling the longest. How many days did they rule (accounting for transition year?)

```
# load the data
df_kings <- read_csv2("kings.csv")
```

```
## i Using "','" as decimal and "'.'" as grouping mark. Use 'read_delim()' for more control.
```

```
## Rows: 47 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ";"
## chr (2): Kings, Yearasruler
## dbl (2): Start_date, End_date
##
## i Use 'spec()' to retrieve the full column specification for this data.
## i Specify the column types or set 'show_col_types = FALSE' to quiet this message.
```

```
# data cleaning + adding days of rule
df_kings <- na.omit(df_kings)




df_kings$"Yearasruler" <- as.integer(df_kings$"Yearasruler") # here I made the column data type to inte


df_kings$"ruleInDays" <- df_kings$"Yearasruler"*365 # here i make the years to days

# finding the mean rule in days
df_kings$"ruleInDays" <- as.integer(df_kings$"ruleInDays")
mean(df_kings$"ruleInDays")
```

```
## [1] 6818.864
```

```
# finding the median rule in days
median(df_kings$"ruleInDays")
```

```
## [1] 5110
```

```
# finding the 3 longest rulers
df_kings %>%
arrange(desc(ruleInDays)) %>%
slice(1:3)
```

```
## # A tibble: 3 x 5
##   Kings              Start_date End_date Yearasruler ruleInDays
##   <chr>                   <dbl>    <dbl>       <int>      <int>
## 1 Christian 4.             1588     1648          60      21900
## 2 Erik 7. af Pommern       1396     1439          43      15695
## 3 Christian 7.             1766     1808          42      15330
```

The mean ruling time is 6819 days and the median time is 5110 days. The monarchs ruling the most time are Christian 4., Erik 7. af Pommern and Christian 7., who ruled 21.900 days, 15.695 days and 15.330 days

- Joshua

Visualize data with ggplot - homework 4

Joshua Hansen

created 4 October 2021, 05 January, 2023

Contents

This exercise is based on the dataset provided by OurWorldInData project based at the Oxford University.

The long-term trend in Homicides in Western Europe

Understanding how homicide rates have changed prior to the modern era requires the help of historians and archivists. Manuel Eisner, a criminology professor at the University of Cambridge, and his colleagues published the Historical Violence Database : a compilation of data on long-term trends in homicide rates, in addition to qualitative information such as the cause of death, perpetrator and victim. This database is limited to countries with relatively complete historical records on violence and crime – mainly Western Europe and the US.

Starting in the second half of the nineteenth century, these European regions have consistent police records of those accused of murder or manslaughter and annual counts of homicide victims. To go back further in time, reaching as far back as the thirteenth century, Eisner collected estimates (from historical records of coroner reports, court trials, and the police) of homicide rates made in over ninety publications by scholars.

Homicide rates – measured as the number of homicides per 100,000 individuals – up to 1990 are sourced from Eisner's (2003) publication and the Historical Violence Database.

Are homicide rates in Europe today lower or higher than in the past? Using the provided dataset, display and describe the long-run homicide rates for the five European regions: Italy, England, Germany, Netherlands and Scandinavia.

```
library(tidyverse)
```

Load the available data from ourworldindata.org

You should always interrogate the source of your data. Who compiled it, from where, what is missing, how representative the data are? Check the data/Metadata.txt to learn about the data provenance.

```
Western_Europe <- read_csv("data/homicide-rates-across-western-europe.csv")
```

Inspect the data

How clean and analysis-ready is the dataset? Do you understand what the column names represent? What is the difference between rate and homicide number?

```
head(Western_Europe)
```

```
## # A tibble: 6 x 4
##   Entity  Code   Year Homicide rate in Europe over long-term (per 100,000) (ho~1
##   <chr>   <chr> <dbl>                                                     <dbl>
## 1 England <NA>   1300                                                        23
## 2 England <NA>   1550                                                         7
## 3 England <NA>   1625                                                         6
## 4 England <NA>   1675                                                         4
## 5 England <NA>   1725                                                         2
## 6 England <NA>   1775                                                         1
## # ... with abbreviated variable name
## #   1: `Homicide rate in Europe over long-term (per 100,000) (homicides per 100,000 people)`
```

Ok, the data look good except for the column `Homicide rate in Europe over long-term (per 100,000)` which is not very easy to work with.

- Use the `names()` function and assignment key to relabel this column to `homicides_per_100k`

```
names(Western_Europe)
```

```
## [1] "Entity"
## [2] "Code"
## [3] "Year"
## [4] "Homicide rate in Europe over long-term (per 100,000) (homicides per 100,000 people)"
```

```
names(Western_Europe)[names(Western_Europe) == "Homicide rate in Europe over long-term (per 100,000) (ho
```

Now, that you have looked at what the data looks like and what it represents, and streamlined it, let's see what big picture it contains.

Let's see what the long-term trend is in homicides

- use `ggplot()` function and remember the`+` at the end of the line
- chose a meaningful `geom_......()` for geometry (hint: points are not great)
- load `Year` on the `x` axis and `homicides_per_100k` column in y axis
- to color individual country entries consistently, assign the country column to the argument `color`.

- provide meaningful title and axis labels
- remember to change the `eval` flag so that the code chunk renders when knitted

```
Western_Europe %>%
    ggplot(aes(x = Year, y =homicides_per_100k, color = Entity )) +
    geom_line() +
    labs(title = "Homicides in Western Europe through Time",
       colour = "Country",
       x = "Year",
       y = "Homicides per 100,000 individuals")
```

Alright, the homicide rates should all be descending over time. What a comfort. But the viz is not super clear. Let's check the rates for individual countries.

Uncouple the homicides of individual countries for easier view

You can visualize each country's trend separately by adding an extra argument to the ggplot, the `facet_wrap()` and feeding it the country column. If in doubt, check your ggplot tutorial and your country column name for exact usage.

- reuse the ggplot from the chunk above
- insert `facet_wrap()` after the specification of geometry to split countries in separate charts
- change the facet "layout" to two columns and three rows so that the trends are easier to see in horizontal layout.

```
Western_Europe %>%
  ggplot(aes(x = Year, y = homicides_per_100k, colour = Entity)) +
  geom_line() +
  facet_wrap(~ Entity,nrow = 3, ncol = 2) +
  theme_bw() +
  theme(legend.position="bottom") +
  labs(title = "Homicides in Western Europe through Time",
       colour = "Country",
       x = "Year",
       y = "Homicides per 100,000 individual")
```

Compare the trends in homicide with the pattern of reign duration among Danish rulers through time.

- Load your Danish king dataset. Hopefully it is tidy and your years and duration of reign are all numeric.
- You need to have a consistent way of plotting the rulers' reign on the x axis, so I recommend you create a midyear column by calculating the middle of each monarch's rule (Hint: `midyear` = `endyear` - (`endyear-startyear`)/2)
- Start a ggplot plotting midyear on x axis and duration on y axis
- Try `geom_smooth()` for geometry
- Provide meaningful labels and a title
- How would you characterize the trend compared to the homicides above?

```
# YOUR CODE HERE:

king_data <- read.csv2("data/kings.csv")
king_data$Duration <- king_data$End_date-king_data$Start_date # needed the Duration first
king_data$Midyear <- (king_data$End_date - (king_data$End_date-king_data$Start_date)/2) # then I create
  king_data %>%
  ggplot(aes(x = Midyear, y = Duration)) +
  geom_smooth() +
  labs(title = "Duration of Danish Monarchs Reign through Time",
       x = "Year",
       y = "Duration of reign")
```

Well, the Duration of reign for danish rulers seems to be mostly increasing, where the end of the timeline suggests a decline. It is a very different trajectory than the homicide graphs, as they were all declining. A longer danish reign == less homicide???? :o

Final tasks:

1) Plot: In the facetted plot above, move the legend from the current position on the side to below the facets, and label it "Country" instead of "Entity".

2) Rmarkdown:

- edit the author of the document, and convert 'Final Tasks' into heading #2 (like the other headings) #DONE
- add a `floating table of contents` to your Rmarkdown document, #DONE
- provide informative `chunk-names` and edit flags in your R chunks, and # DONE
- automatically generate a `timestamp` to show when the document was last updated. (Hint: check the Rmarkdown episode in our Data Carpentry tutorial) # DONE

3) Question: In <250 words articulate your answer on the basis of the data visualisations to the following question: are we more civilized today?

Joshua: By looking at the different countries and their declining trend of homicides could in fact visualize are more "civilized" society. What you really can tell from the graphs are, that homicides are declining, but whether or not that makes us more "civilized" as a society is up for interpretation. Homicides could be substituted with many other crimes and vices. When plotting data and analyzing it, it is always important to remember what you are visualizing. It would be easy to say that we seem to be more civilized as a society with less homicides, but that is not what the plot is showing. It might be an indication of a more civilized society, but should be further analyzed. Summarized, one could say we seem more civilized throughout time, yes :)

Managing Files on Steroids with Shell – homework 5

Joshua Hansen

2023-01-05

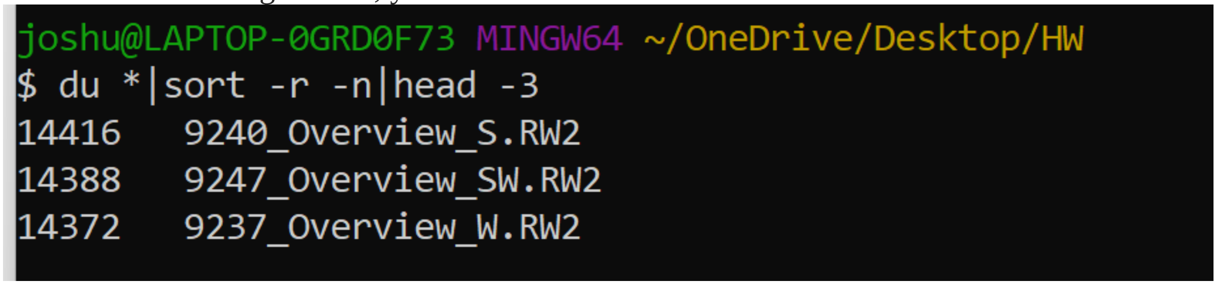
DESCRIPTION

Your supervisor has shared a folder of photos on Sciencedata.dk with you (password is 2020CDS, folder is 500Mb and contains 189 images) and needs your help with a couple diagnostics:

1) Identify the names and format of the 3 biggest files. Can you come up with a command to generate a numerically ordered list of 3 biggest files? (hint: consider using **wc** to gauge image size)

My answer:

To find the three largest files, you can write :



```
joshu@LAPTOP-0GRD0F73 MINGW64 ~/OneDrive/Desktop/HW
$ du *|sort -r -n|head -3
14416   9240_Overview_S.RW2
14388   9247_Overview_SW.RW2
14372   9237_Overview_W.RW2
```

Where :

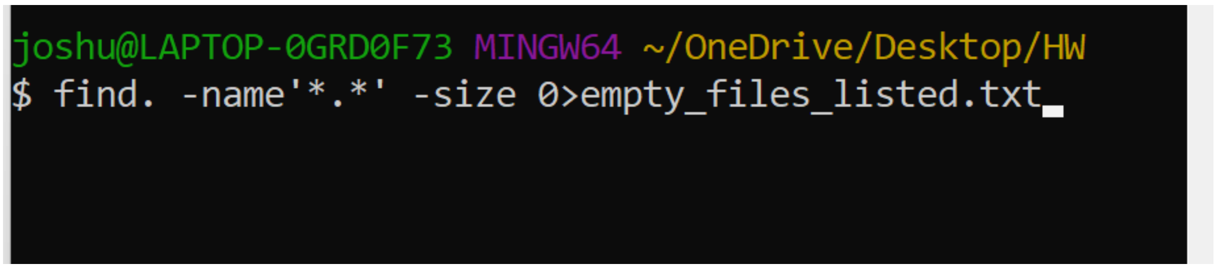
Du *: measures the disk space that files use and * says I want to count all files in the folder

Sort -r -n : sorts the first character of each line of a file and outputs it in alphabetic order, and -r and -n displays them in reverse numerical order.

Head -3: shows the first the lines of the output, which will be the first biggest files.

2) Some of the image files are empty, a sign of corruption. Can you **find** the empty photo files (0 kb size) , count them, and generate a list of their filenames to make their later replacement easier?

To find the empty files and store them in a document you can write:


```
joshu@LAPTOP-0GRD0F73 MINGW64 ~/OneDrive/Desktop/HW
$ find. -name'*.*' -size 0>empty_files_listed.txt
```

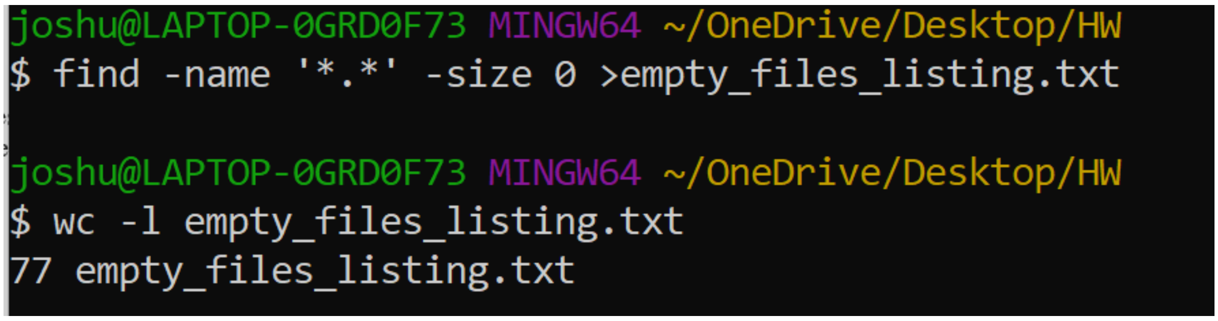
Where:

Find. -name : searches for files in the directory
'*.*': specifies that it can be all file formats and names (the . does that)

-size 0: specifies that the files have a size of 0

>. **Empty_files_listed.txt**: puts the output in a textfile.

With wc -l empty_files_listed.txt we can get a count on the empty files which are 74 files.


```
joshu@LAPTOP-0GRD0F73 MINGW64 ~/OneDrive/Desktop/HW
$ find -name '*.*' -size 0 >empty_files_listing.txt

joshu@LAPTOP-0GRD0F73 MINGW64 ~/OneDrive/Desktop/HW
$ wc -l empty_files_listing.txt
77 empty_files_listing.txt
```

3) **Optional/Advanced:** Imagine you have a directory goodphotos/ (same password as above) with original non-zero-length files sitting at the same level as the current directory. How would you write a loop to replace the zero length files?

Sorry 😊

Practicing functions with GapMinder - homework 6

Joshua Hansen

05/01/2023

Task 1 : Define a defensive function that calculates the Gross Domestic Product of a nation from the data available in the gapminder dataset. You can use the population and GDPpercapita columns for it. Using that function, calculate the GDP of Denmark in the following years: 1967, 1977, 1987, 1997, 2007, and 2017.

In order to work with the gapminder data, we can either download and read it as a csv file or just install it through R, as it is an included dataset.

```
#install.packages("gapminder")
library(gapminder)
```

```
## Warning: package 'gapminder' was built under R version 4.2.2
```

After that, we can inspect if the data is available:

```
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

To write the function, we have to consider to include the year and the country column of the dataset. The GDP we can calculate by taking the GDP per capita times the population of the country.

We want a function, that takes our data is input and can select a specific year and a specific country: The parameters are the "input" we give the function in order for it to work. In the function you can use the variable names from the parameter as a "place-holder", so when we use the function, the parameter variables are gonna be switched to the input we give the function.

```
calc_GDP_by_year_country <- function(data, year=NULL, country=NULL) { # create function called calcGDP,
  if(!is.null(year)) { # i.e. : if the parameter year is not null,check for that yearand return a subse
    data <- data[data$year %in% year, ] # %in% checks if the year given is included in the dataframe
  }
  if (!is.null(country)) { # ifcountry parameter is not null, check for country in DF and return a subs
    data <- data[data$country %in% country,]
  }
  gdp <- data$pop * data$gdpPercap # gdp is population times the gdp per capita

  new <- cbind(data, gdp=gdp) # cbind == column-bind, puts the calculated gdp as a new column in the da
  return(new) # we return the updated dataset with the gdp column
}
calc_GDP_by_year_country(gapminder, country="Denmark") # calling function with gapminder data, and coun
```

```
##    country continent year lifeExp     pop gdpPercap          gdp
## 1  Denmark    Europe 1952  70.780 4334000  9692.385  42006797652
## 2  Denmark    Europe 1957  71.810 4487831 11099.659  49813395320
## 3  Denmark    Europe 1962  72.350 4646899 13583.314  63120285966
## 4  Denmark    Europe 1967  72.960 4838800 15937.211  77116977700
## 5  Denmark    Europe 1972  73.470 4991596 18866.207  94172484445
## 6  Denmark    Europe 1977  74.690 5088419 20422.901 103920280028
## 7  Denmark    Europe 1982  74.630 5117810 21688.040 110995270449
## 8  Denmark    Europe 1987  74.800 5127024 25116.176 128771236166
## 9  Denmark    Europe 1992  75.330 5171393 26406.740 136559629613
## 10 Denmark    Europe 1997  76.110 5283663 29804.346 157476118456
## 11 Denmark    Europe 2002  77.180 5374693 32166.500 172885062707
## 12 Denmark    Europe 2007  78.332 5468120 35278.419 192906627081
```

Here you can see a list of the different GDP's in denmark throughout the years.

Task 2 : Write a script that loops over each country in the gapminder dataset, tests whether the country starts with a 'B' , and prints out whether the life expectancy is smaller than 50, between 50 and 70, or greater than 70. (Hint: remember the grepl function, and review the Control Flow tutorial)

For this task, we: 1. subset the country column, and check whether or not a country starts with a "B" 2. calculate the average life expectancy for that specific country 3. control flow:we want to see if the life expectancy is <50 or between 50 and 70 or >70 and output the fitting category

```
low_exp <- 50 # creating a threshold of the low life expectancy,
high_exp <- 70
candidateCountries <- grep("^B", unique(gapminder$country), value = TRUE) # picking all the countries t

for (country in candidateCountries) { #loop over the countries starting with "B"

    current_country <- gapminder[gapminder$country == country, ] #specifying the country we want the me
    average_exp <- mean(current_country$lifeExp, na.rm = TRUE) # calc. the mean of lifeExp for the coun


  if(average_exp < low_exp) { # if the average life expectancy is lower than 50, print out the expectan
    cat("the average life expectancy in", country, "is less than", low_exp, "\n")
  } else if(average_exp > low_exp && average_exp < high_exp) {
    cat("the average life expectancy in", country, "is between", low_exp, "and", high_exp, "\n")
  } else {
    cat("the average life expectancy in", country, "is greater than", high_exp, "\n") # cat = function
  }

  rm(average_exp) # we are removing the tmp after each iteration, to create the average lifeExp for the
}
```

```
## the average life expectancy in Bahrain is between 50 and 70
## the average life expectancy in Bangladesh is less than 50
## the average life expectancy in Belgium is greater than 70
## the average life expectancy in Benin is less than 50
## the average life expectancy in Bolivia is between 50 and 70
## the average life expectancy in Bosnia and Herzegovina is between 50 and 70
## the average life expectancy in Botswana is between 50 and 70
## the average life expectancy in Brazil is between 50 and 70
## the average life expectancy in Bulgaria is between 50 and 70
## the average life expectancy in Burkina Faso is less than 50
## the average life expectancy in Burundi is less than 50
```

Here we can see an output with the life expectancies of the countries starting with “B”

Task 3 ups sorry :)