Learning Diary: Data Preprocessing and Visualisation

Fabian Strauch

January - March 2025

Lectures

1 Introduction

I chose this course in my exchange semester at Umeå, because I'm interested in the whole domain of data science and machine learning, but didn't have any experience with it yet, as I had mainly focused on software development so far.
Especially with the visualisation focus of this course I was hoping to get a nice and intuitive entry point to the whole domain and, together with the machine learning course I'm also doing at Umeå, return back home with a solid foundation.
After hearing that the course would not have an exam, but a project and a learning diary instead, I was also quite happy, because I could look forward to a lot of meaningful and fun hands-on experience.
I'm studying applied computer science in my bachelors at TU Dortmund with the application subject "Enterprise Computing", which focuses mainly on business information systems and contains an elective area largely made up of data science and machine learning modules, as those fields are the backbone for many business applications. In this area I'm able to get this course accredited and I think it fits quite nicely within my subject.
I was also looking forward to use the knowledge I was about to gain for some personal projects, such as visualizing my progress at the gym and analyzing it together with e.g. sleep or nutrition tracking.

2 Introduction to Data Science and R

Lecture from: 21.01.2025
Held by: Kary Främling, Gustav Pihlgren, Per Arnqvist

2.1 Data Visualisation

The lecture began with an introduction to Data Visualisation and marked it's importance: why it is used, how it can be used and also how it can be misused. First, the why: Looking at raw data makes it very hard for humans to quickly comprehend the information being displayed. I think that this makes Data Visualisation an increasingly more important tool, as more and more data is available to us all the time. Because of this immense information overflow the demand for ways to cope with this and derive opinions and decisions from data quickly is getting bigger every year.
Some basic visualisations that most of us see day to day were quickly introduced with some examples. There it was quite intuitive to see why certain visualisations were chosen to display certain types of data. I always thought of tables as a very bad visualisation. But looking at the example in the slides, where certain cells were highlighted in different colors and with different symbols, I changed my mind on that: If done correctly they can be very useful to get an overview over a lot of data. For getting a closer look at data of a specific sample, other visualisations, such as graphs are more useful.
Looking at the maps visualisations, where you could see how a bubble map could give you wrong impressions when displayed over deeply crowded central europe vs. huge countries like Russia or Canada, reminded me of
www.thetruesize.com. Earth often gets displayed in a distorted way, where countries further from the equator

seem much bigger than they actual are. On website page you can easily compare country sizes from different locations. There I tried out a few things and found out that e.g. Sweden isn't even that much bigger than Germany. I could have easily looked up the area size of both countries. But seeing it visualized like this made me understand their sizes much quicker.
Further more some real-life industrial tools, that use visualisations were introduced. As I am currently working on a similar project from my home university, I found that quite interesting and might take some inspirations from that.
They were followed by some simple statistic visualisations, that I remembered from my statistics introductory course.
At the end of this chapter some common misleading graphs were presented. I think simple illusions like truncating a bar chart can easily be debunked my most people, but especially the improper scaling example, very common in articles, where cheeky authors want to convince people of very drastic opinions, gets just accepted my most people without any critical thinking, I think. It reminded me of a propaganda graph from the nationalsocialists we discussed in a history lessons many years ago. This shows the dark side of visualisation. And today there are much more sophisticated ways in which misleading visualisations can be used to convince masses on social media of distorted truths. That's why I think it is crucial, to educate people, not just computer scientists and engineers needing it for their work, about proper and improper visualisation of data.

2.2 Data Preprocessing

The domain of data preprocessing was also briefly introduced in the lecture. A given example was the preprocessing of aircraft sensor data. I think here it becomes very intuitive why preprocessing is needed and why it is important to properly understand it. When people's life are at stake, an engineer must know how to properly have a computer read and understand the data it is receiving.
Data preprocessing involes cleaning data (especially sensor data can be very noisy), standardizing and normalizing it (e.g. removing the mean of two time series and adjuting them to the same scale in order to be able to compare them properly), transforming it (e.g. to train machine learning models properly) , extracting features (e.g. trend in a time series.), among other tasks.
Those were some immediate examples that I had in mind. As I am currently writing a report for a Seminar on Time Series Analysis from my home university, I recognize some of the mentioned tasks.
With higher demand for machine learning applications, I think data preprocessing is also getting more and more important. If it is not done correctly, a trained model could return false outputs. Because enterprises and organizations are exploring data-driven decision making now more than ever, improper data preprocessing could lead to dangerous decisions, no matter how could the ML model itself is.
As I am also curious about machine learning, I am looking forward to learn more about data preprocessing in this course to learn more of this crucial step.

2.3 R Basics

I have installed R and R Studio and tried out some of the presented code and data types:

```
library(ggplot2)

new_multiply_function <- function(a, b) {
  result <- a * b
  print(result)
}

new_multiply_function(5,3)

list_x <- list(c(5,3),5.8,list(2, 7.6, "Hi"))
myList <- list(tupel(4,5), "Hello there",list("this","is","a","sublist",TRUE))

myFunction <- function(s,n,l){
```

```
  print(s)
  print(n)
  print(l)
}

myFunction("Hello World", 5150, myList)
```

2.4 TED Talk: "Let my dataset change your mindset"

Out of the available TED Talks I chose "Let my dataset change your mindset", because I am always interested in seeing, how objective my common knowledge (or common knowledge itself) is.
And I found out that it seems to be very subjective in certain domains. A sentence that really stuck with me was: "... The worldview that my students had, corresponds to reality in the world the year, their teacher was born". This goes to show that the real world (and with it its data we collect) can change much more rapidly than we as humans may be able to comprehend.
We also often tend to simply the world / domain we are working with. The professor showed that the classical categorization in western and developing countries made sense 80 years ago. But since then the actual data shows that a clear binary classification simply can't be done anymore, as the countries have mingled much more.
When working with data it is always derived from some domain, that might change very rapidly. I think that when working with data, one should always keep that in mind and always question an intuition that might have formed around the data, as it might be outdated by then.
What the TED Talk also made me think about was how hard it can be to compare data. It was said, that China first developed its health and then its economy much more quickly. Instead of the USA, which did it rather the other way around. Is it then fair to compare the GDP of both countries? In such a domain with many dimensions it might be to simple to just look at one. Here it is also easy to make one country look better than another by cherry-picking certain features. This goes to show that domain knowledge and a critical view of own biases is needed before creating data visualisations. Something that then might look clear and objective might not be as appropriate as it may seem.
What I also liked about the speech is the disection of countries in different areas. For the sake of simplicity this gets often overlooked. In the domain of healthcare this can lead to false conclusion on how to e.g. fight an epidemic. Infections might be highly localized in a couple of regions. If a global cause is then wrongly assumed, seemingly objective and data-driven decisions to fight the epidemic might than actually be highly inefficient.
So for the domain in question one should always ask themselves, if they derive the right information from given data.
The only time I really see a such a disection, is when former east and west germany get compared. In a lot of current maps you can actually directly see the former borders. Here it might be a bit more obvious to look at both individually. In other instances it might not be.

Out of interest I also took a quick look on the mentioned tool Gapminder. Of course I didn't obtain any world-changing insights from this, but it was interesting anyway.
As an example I compared the GDPs of some former Sovien Union countries.

3 Basic Data analysis & visualisation

Lecture from: 24.01.2025
Held by: Kary Främling

3.1 The R programming langauge

This lecture provided a first more practical hands-on with R; How data can be imported, properly prepared and visualised with some commonly used graphs. Especially the provided R examples where very useful as a reference for the exercise; A very welcome alternative to loosing myself in a thousand StackOverflow articles. As the lecture itself was mostly a demonstration of a lot of small things, I want to focus more on the exercise rather than commenting on every type of visualisation and import method introduced.
The power of R, to create substantial and insightful visualisations in just a few lines really impressed me. Even though it takes more time to get used to it than to other programming languages and I haven't quite developed an intuition on how values are mutated and referenced, I enjoyed the learning process and look forward to more.
Being mainly a javascript developer I really embrace the loose typing and R being multiparadigm - you can get a lot of things done very fast. I think the declarative and partly functional nature of R makes a lot of sense for a non-general-purpose language largely addressed to non-computer-scientists, who want to get things done without having to worry about the implementation. As R is my first non-general-purpose language I looked up the Wikipedia page and wanted to gather some more information about it - A lot of the principles instantly made sense to me for a language that is supposed to process large amounts of data; Deep-copying for nested datasets, lazy evalutation to prevent high computational costs of big data, among other things.

3.2 The Exercise - Road accidents and speed limits

At first I wanted to check out all the datasets already available in R. To do that I ran these two lines:

```r
data(package="ggplot2")
data(package = .packages(all.available = TRUE))
```

After scrolling thru the quite big selection I decided to use the *Traffic* dataset from the *MASS* package. According to the short description provided, it shows the effect of swedish speed limit on accidents.
As the possible introduction of a speed limit has been a hot topic in Germany for quite a while and we usually just copy what the Scandinavians do, after having failed at it ourselves for 10+ years, I found that interesting.

3.2.1 A first look at the data

While taking a first look at the dataset, I noticed that it is not very big, only 184 observations. This makes it not very useful to really draw any real conclusions from, but very good to examine whether the processing I did to the data did what I wanted it to.
Executing

```r
head(Traffic, 3)
tail(Traffic, 3)
summary(Traffic)
str(Traffic)
var(Traffic$y) # contains the amount of accidents from that day
```

showed, that the dataset contained observations from the years 1961 and 1962, 92 for each. Here's the first and the last 3 samples to portray the structure of the data:

```
> head(Traffic, 3)
  year day limit  y
```

```
1 1961   1    no  9
2 1961   2    no 11
3 1961   3    no  9
> tail(Traffic, 3)
  year day limit  y
182 1962  90   yes 14
183 1962  91   yes 15
184 1962  92   yes  9
```

Seeing how old and small the dataset is and how big its variance (77.19978), instantly showed me, that this will not be a good base for any conclusions... But I wanted to continue with it anyway. I could already see that there was no missing data, but I tested it anyway with `colSums(is.na(Traffic))`, which confirmed my assumption.
Before looking further into the data, I assumed, that it would reflect the introduction of a general speed limit law, with some amount of observations before and some after that critical timepoint. Because of that I wanted to plot the data on a graph, as I expected there to be a drastic change visible. So I created a plot, which made me see the high variance very clearly:

```
ggplot(Traffic, aes(x = day, y = y)) +
  geom_point(aes(color = limit), size = 1) +
  labs(y = "Accidents", x = "Day")
```

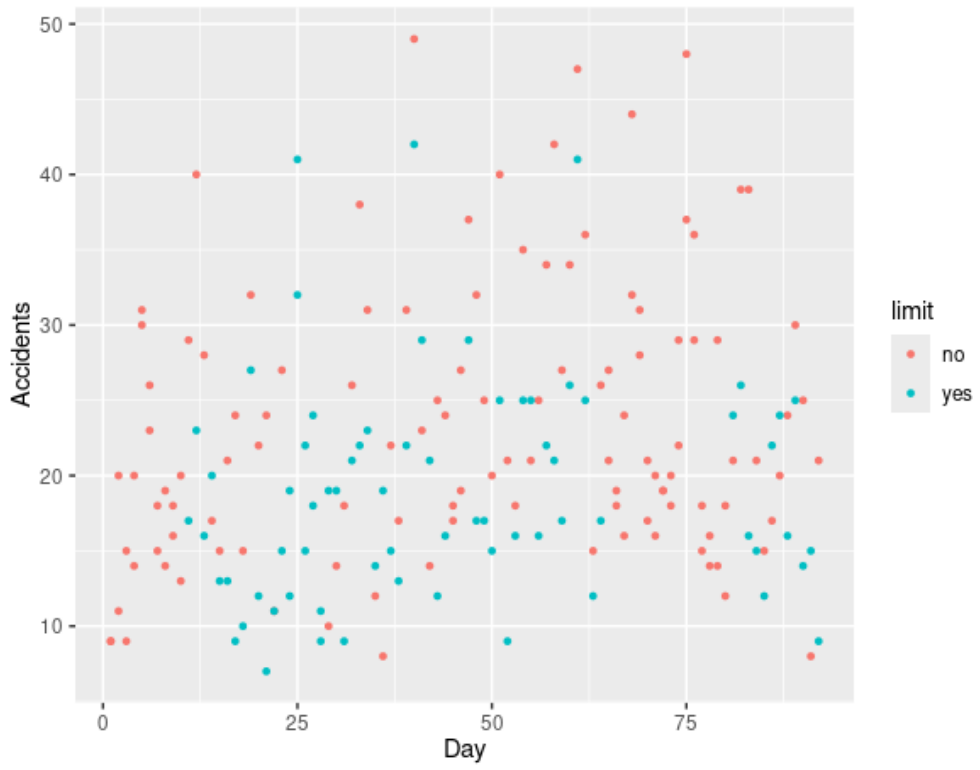

Figure 1: Rookie scatter plot of the traffic dataset

I quickly noticed however, that I had made a mistake; The i-th days of both years get rendered together at the same x coordinate, as the date of the observation is saved in the combination of the 'day' and the 'year' column instead of just one column. As a quick fix I added a new column that would arrange continuous dates for each observation and used it as the x domain for a new plot:

```
Traffic$date_cont <- ifelse(
    Traffic$year == "1961", Traffic$day, Traffic$day + 92
)
```

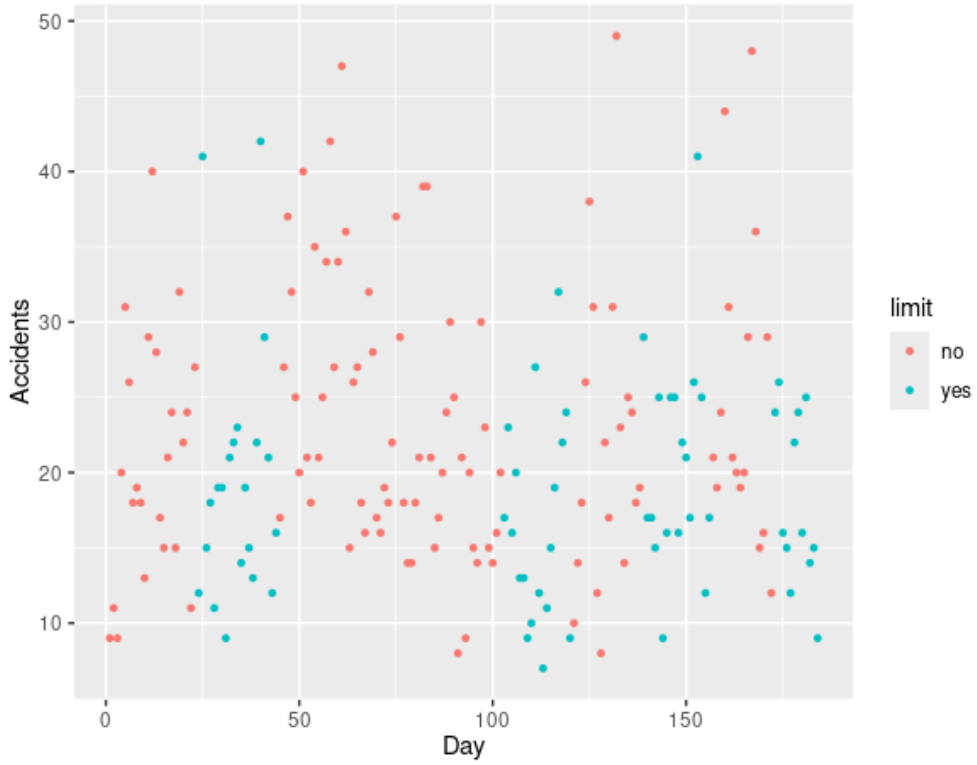

Figure 2: Fixed scatter plot of the traffic dataset

Looking at this showed me that my assumption was wrong. The data did not reflect the introduction of a law with a clear "before" and "after" period. Later I found a more detailed description of the data online:

> An experiment was performed in Sweden in 1961–2 to assess the effect of a speed limit on the motorway accident rate. The experiment was conducted on 92 days in each year, matched so that day j in 1962 was comparable to day j in 1961. On some days the speed limit was in effect and enforced, while on other days there was no speed limit and cars tended to be driven faster. The speed limit days tended to be in contiguous blocks.
>
> *taken from pmagunia.com*

So the data contains alternating time periods with and without speed limit. Sadly the length of time periods were not matched so that e.g. for a specific time period in 1961 with a limit, the same time period in 1962 would exactly not have a limit. That would have allowed for many more interesting comparisons.
Because I could not really gain a lot of insights from this scatterplot, I wanted to instead create a simple Boxplot, where the data gets seperated into whether it came from a day with or without a speed limit:

```
ggplot(Traffic, aes(x=limit, y=y, fill=limit)) +
  geom_boxplot() +
  labs(x="Speed Limit", y="Accidents")
```

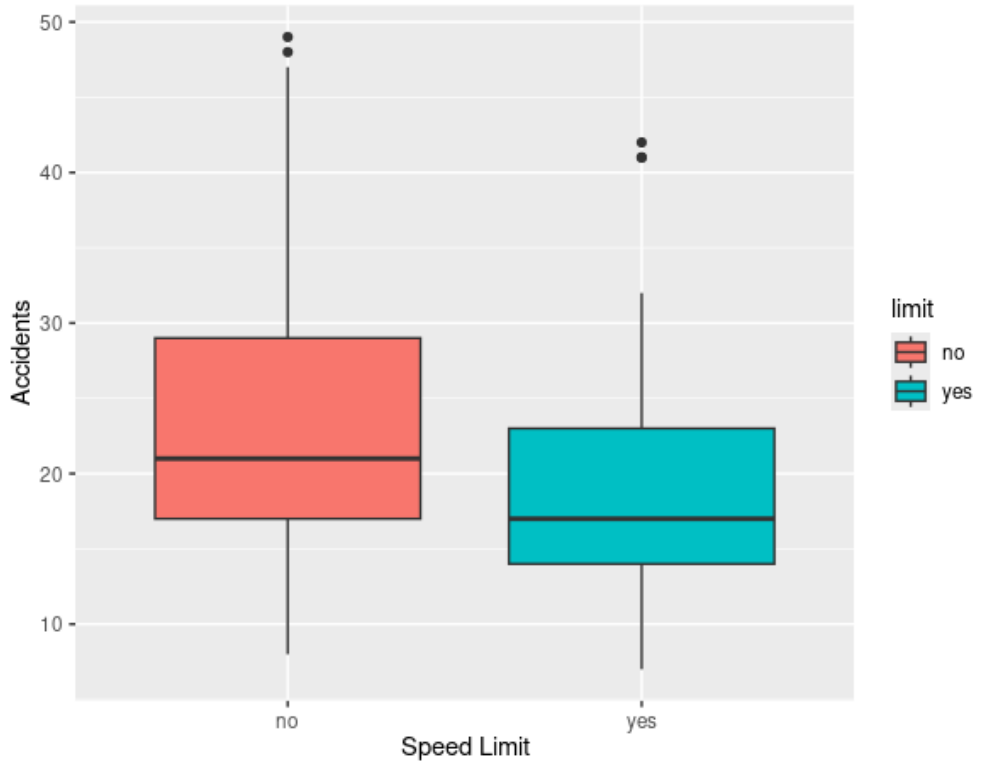

Figure 3: Boxplot of the traffic dataset, separating limit and no limit

This visualizes much more useful information about the data: The medians of both categories are not that far away from each other, however on most days with a speed limit were generally less accidents. Furthermore, the amount of accidents also varies less on days with a speed limit and the maximum values (without outliers) are drastically lower. I also wondered if the outliers from the 'yes' category, which are very far of from the rest of the data, might have been from special days, like e.g. from holidays where there would be more traffic. Sadly, because the data does not capture the concrete dates, I could not investigate this any further. If so, it would have also find it interesting to compare the difference between weekdays and weekends.

3.2.2 Correlation - Or lack thereof

As, besides the amount of accidents itself, the dataset contains rather categorial data, I did not except a high correlation between any of the columns, but at least some significant one between "limit" and "y". Because the "limit" column saved string values "yes" and "no", I had to first make it numeric, so that I can use it in a correlation check:

```
Traffic$limit_num <- ifelse(Traffic$limit == 'no', 0, 1)
```

The correlation check had the following results:

```
> cor(Traffic[,c("year", "day","y","limit_num")])
                year         day          y   limit_num
year       1.0000000  0.00000000 -0.1383128  0.30310304
day        0.0000000  1.00000000  0.1698516 -0.04501991
y         -0.1383128  0.16985162  1.0000000 -0.23301017
limit_num  0.3031030 -0.04501991 -0.2330102  1.00000000
```

I would have expected some more significant values for the limit and y. That the rest would not have a high correlation was to be expected. However, looking at the first scatter plot again, those values make sense;

the data is very dispersed.

3.2.3 Removal of Outliers

Next up I wanted to remove the outliers of the data. Looking at the boxplot, it was visible that the "yes"-category had outliers quite far of from the rest of the data. Before performing any processing on the data I wanted to quickly take a look at the few largest values myself: (As the domain from y starts at 0 and the data was already pretty close to that, I did not expect there to be any negative outliers)

```
> tail(Traffic[order(Traffic$y),], 10)
    year day limit  y date_cont limit_num
12  1961  12    no 40        12         0
51  1961  51    no 40        51         0
25  1961  25   yes 41        25         1
153 1962  61   yes 41       153         1
40  1961  40   yes 42        40         1
58  1961  58    no 42        58         0
160 1962  68    no 44       160         0
61  1961  61    no 47        61         0
167 1962  75    no 48       167         0
132 1962  40    no 49       132         0
```

By looking at the whole dataset globally, there seem to be not any drastic outliers, at least under the assumption that the largest 10 values aren't all outliers.
I also generated a Histogram to further investigate this:

```
hist(Traffic$y, xlab="Accidents")
```

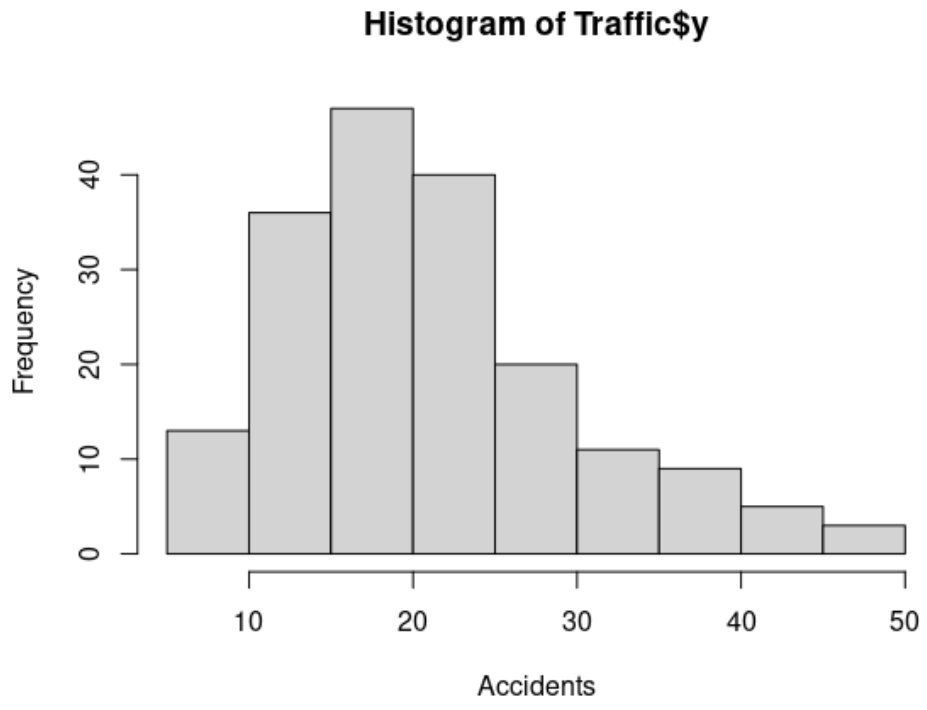

Figure 4: Histogram of the unprocessed Traffic dataset

Looking at this give the impression of more positive outliers then the tail above does.
Now I wanted to perform a the proper outlier removal. A common approach I found for this online was the "Interquartile Range": The difference of the upper and lower quartiles gets calculated. Subtracting from the lower/adding to the upper quartile this difference multiplied by 1.5 then returns a lower and upper bound. If a sample does not lie between those bounds, it is considered to be an outlier. To perform this removal I wrote the following lines and saved the new dataset without the outliers into *traffic_1*:

```
Q1 <- quantile(Traffic$y, 0.25)
Q3 <- quantile(Traffic$y, 0.75)
IQR <- Q3 - Q1

lower_bound <- Q1 - 1.5 * IQR
upper_bound <- Q3 + 1.5 * IQR

cat(lower_bound, upper_bound)

outliers <- subset(Traffic, y > upper_bound)
print(outliers)

traffic_1 <- subset(Traffic, y <= upper_bound)
```

The calculated bounds were $-1, 5$ and 42.5, so my assumption, that there would be no negative outliers was right. There ended up being 4 outliers in total, all samples without a speed limit.
Because of this, I wondered whether it might be better to remove outliers from both categories (with and without speed limit) separately. As mentioned before, the Boxplot showed quite a big difference between a few big values from the "yes"-category and the rest of it's data.
In order to do that I created two subsets of the data, aswell as a function to remove outliers:

```
traffic_limit <- subset(Traffic, limit == 'yes')
traffic_no_limit <- subset(Traffic, limit == 'no')

remove_outliers <- function(dataset, property){
  Q1 <- quantile(property, 0.25)
  Q3 <- quantile(property, 0.75)
  IQR <- Q3 - Q1

  lower_bound <- Q1 - 1.5 * IQR
  upper_bound <- Q3 + 1.5 * IQR

  cat(lower_bound, upper_bound, "\n")
  print(subset(dataset, (property > upper_bound | property < lower_bound)))

  return (subset(dataset, (property < upper_bound & property > lower_bound)))
}
```

After that I applied the function to both subsets, created a new merged dataset *traffic_2* and created a new Boxplot:

```
traffic_2 <- rbind(
  remove_outliers(traffic_limit, traffic_limit$y),
  remove_outliers(traffic_no_limit, traffic_no_limit$y)
)

ggplot(traffic_2, aes(x=limit, y=y, fill=limit)) +
  geom_boxplot() +
  labs(x="Speed Limit", y="Accidents")
```

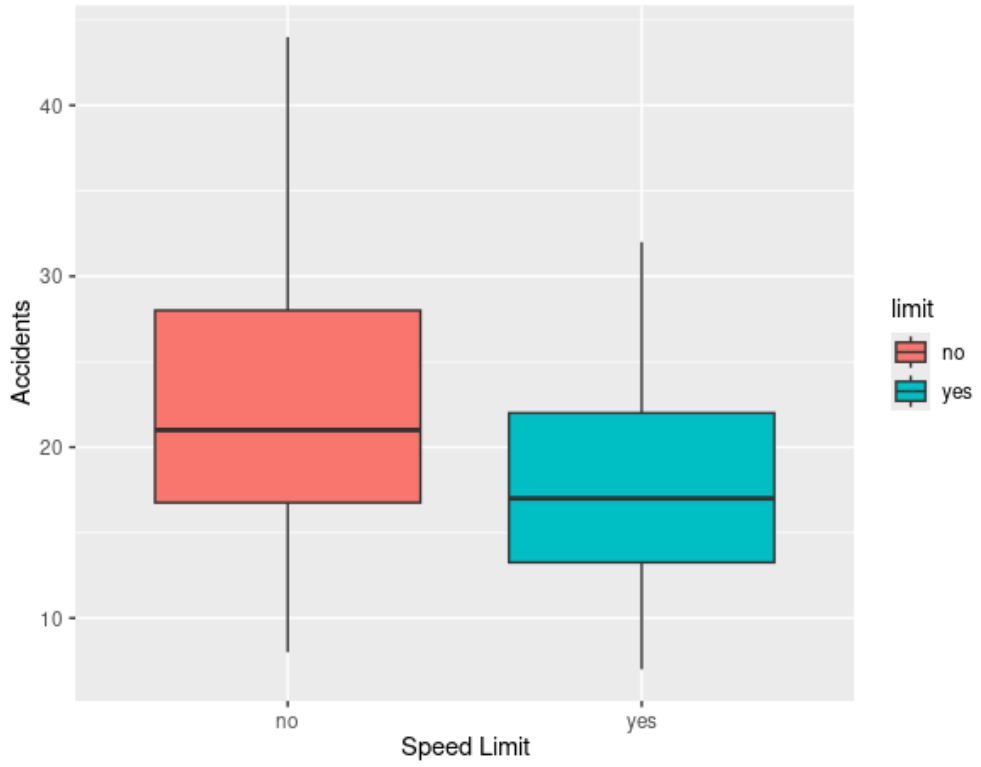

Figure 5: Boxplot of the Traffic dataset, with outliers removed in groups

This goes to show that even something seemingly mundane as removing outliers should be done in different ways, depending on the structure and distribution of the data being worked with. As this result satisfied me much more than the one from the global outlier removal, I decided to continue working with *traffic_2* instead of *traffic_1*.

3.2.4 Mean Removal and Scaling

After having quickly checked that the mean is 20.76966 with `cat(mean(traffic_2\$y)`, I simply removed it with an elegant functional statement and plotted a Boxplot once again:

```
traffic_3 <- mutate(traffic_2, y = y - mean_y)

ggplot(traffic_3, aes(x=limit, y=y, fill=limit)) +
  geom_boxplot() +
  labs(x="Speed Limit", y="Accidents")
```

Of course the Boxplot showed the same proportions as Figure 5, just moved -20.76 points on the y scale. Again I wondered, whether mean removal in groups would make sense here and tried it out with a similar approach as for the outlier removal:

```
remove_mean <- function(dataset, property){
  mean_prop <- mean(property)

  return (mutate(dataset, y = property - mean_prop))
}

traffic_limit_clean <- remove_outliers(traffic_limit, traffic_limit$y)
traffic_no_limit_clean <- remove_outliers(traffic_no_limit, traffic_no_limit$y)
```

```
traffic_4 <- rbind(
  remove_mean(traffic_limit_clean, traffic_limit_clean$y),
  remove_mean(traffic_no_limit_clean, traffic_no_limit_clean$y)
)

ggplot(traffic_4, aes(x=limit, y=y, fill=limit)) +
  geom_boxplot() +
  labs(x="Speed Limit", y="Accidents")
```

Which resulted in:

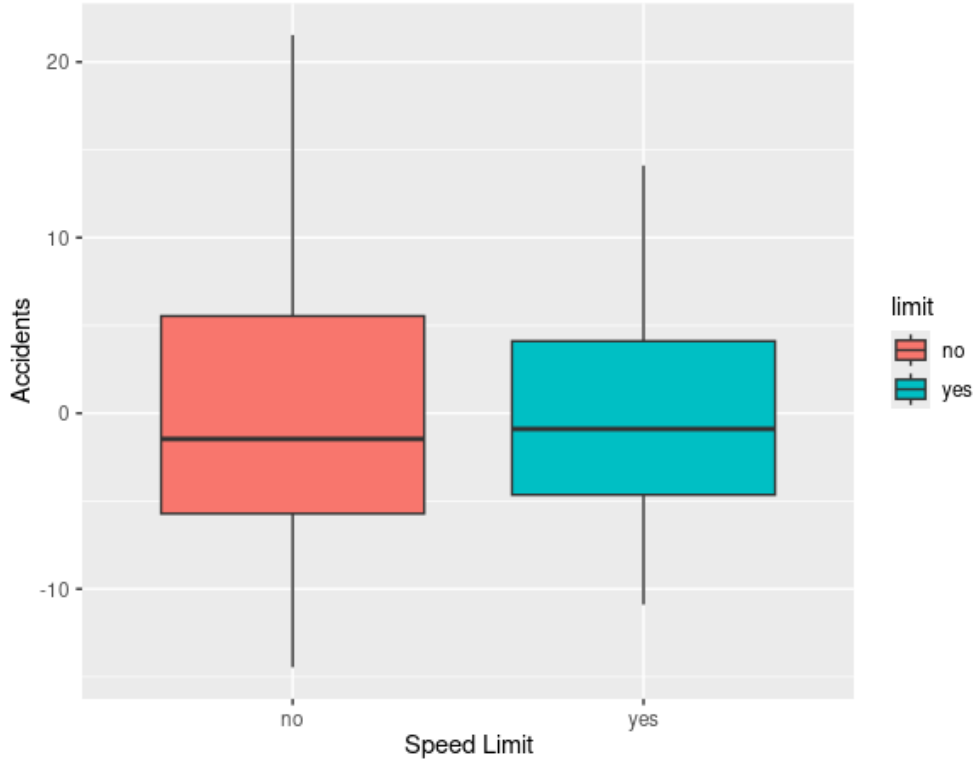

Figure 6: Boxplot of Traffic dataset, with means removed in groups

Here I'm not so sure, whether I would prefer this over a simple general mean removal. Now the data shows a distorted view of the distances between the groups. I can however imagine, that this could be useful for other tasks. Here I decided to not use it any further and chose to continue working with *traffic_3* instead of *traffic_4*.
For the scaling I calculated the (global) standard deviation and used it on the dataset, where the global mean was already removed and created once again a Boxplot:

```
sd_y <- sd(traffic_3$y)
traffic_5 <- mutate(traffic_3, y = y / sd_y)

ggplot(traffic_5, aes(x=limit, y=y, fill=limit)) +
  geom_boxplot() +
  labs(x="Speed Limit", y="Accidents")
```

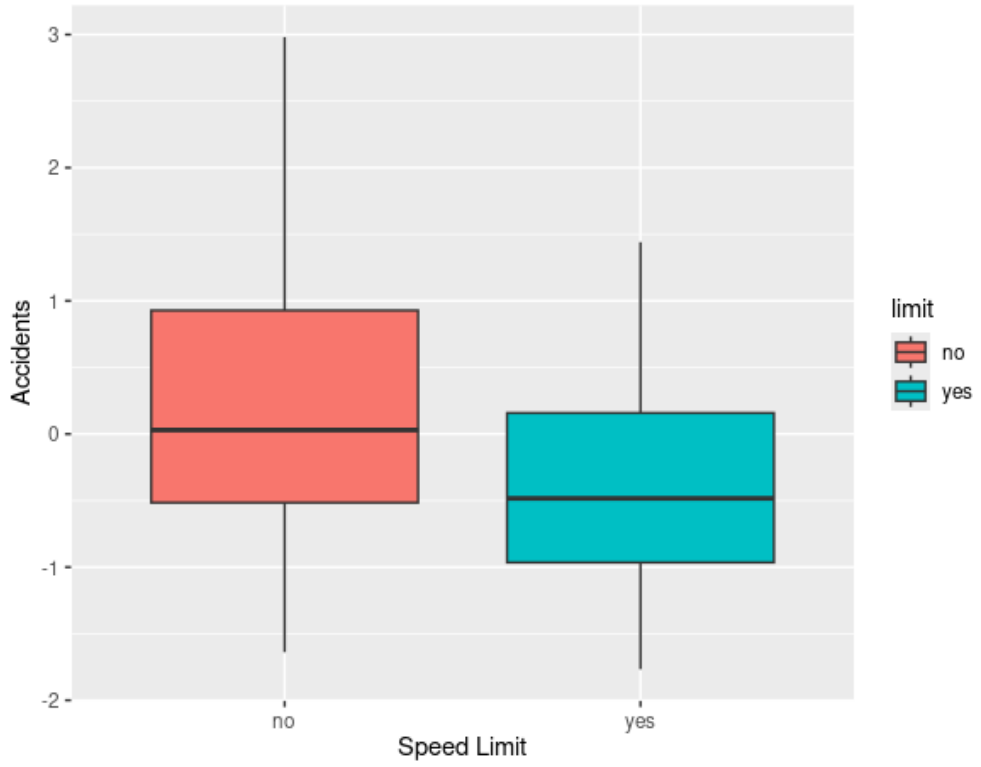

Figure 7: Boxplot of Traffic dataset, outliers removed in groups, mean removed globally and scaled with standard deviation globally

3.2.5 More Visualisations

In all of the following visualisations I used the *traffic_2* dataset, where outliers were removed in groups, but no other processing was done.
First I wanted to see, if I could answer my initial research question better by portraying the difference between a limit and no limit better then with a scatter plot. After playing around with a few different graphs I settled on a smooth graph:

```
ggplot(traffic_2, aes(x = date_cont, y = y)) +
  geom_smooth(aes(color = limit)) +
  labs(y = "Accidents", x = "Day")
```

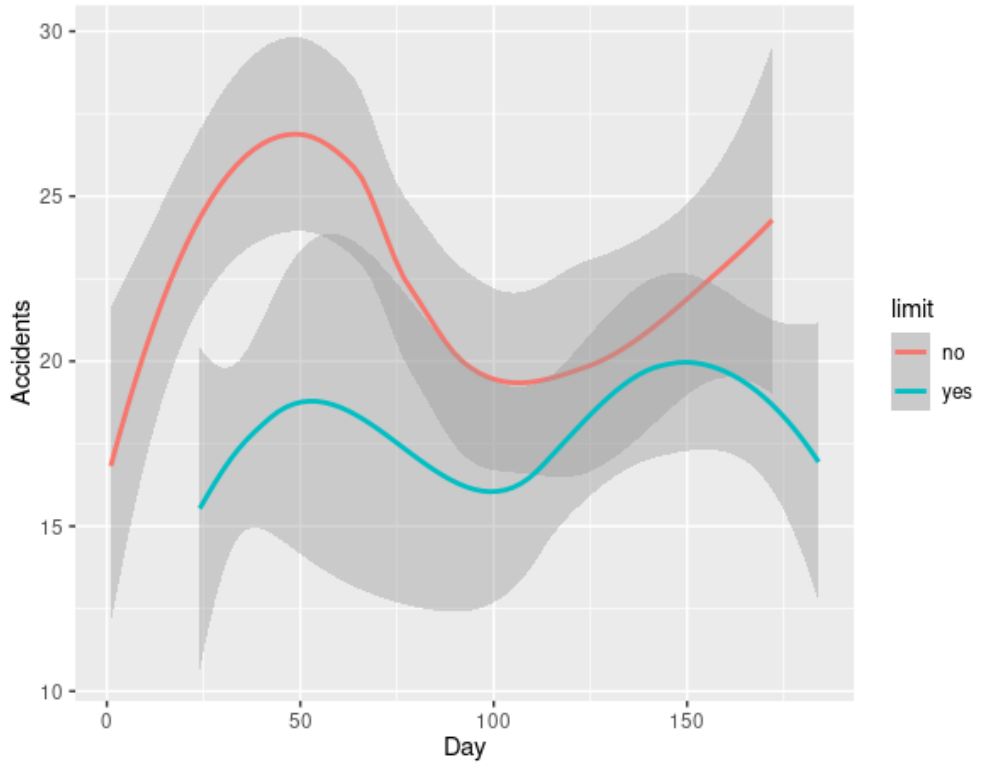

Figure 8: Smoothed Plot of Traffic dataset

While this does clearly show, that significantly less accidents occured with a speed limit, I would not want to use this plot; It looks to much how I would want the data to look like, not how the data actually is. Simplifying data with such a high variance into a graph like this distorts the nature of it and gives wrong impressions about it's distribution and trends that, when looking at the Scatterplot from the beginning, do not really exist.

Another Plot that I was interested in and that ended up giving me much better insights without distorting the data was a violin plot:

```
ggplot(traffic_2, aes(x=limit, y=y, fill=limit)) +
  geom_violin(trim=FALSE)+
  geom_boxplot(width=0.1, fill="white")+
  labs(title="Amount of daily accidents with and without speed limit",x="Limit", y = "Accidents")
```

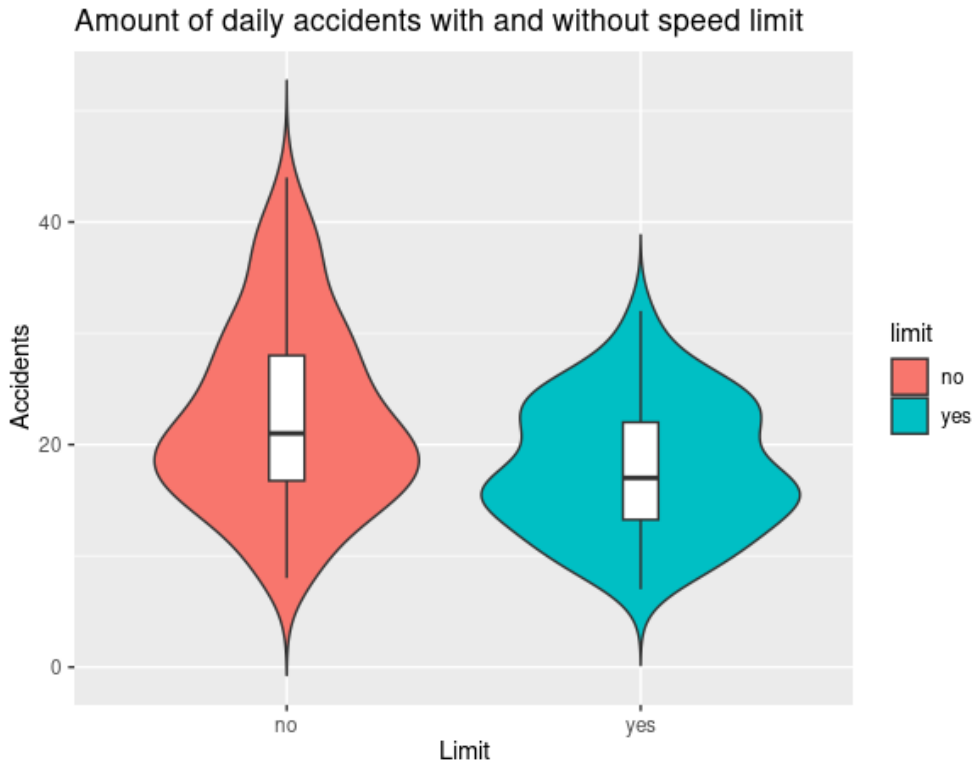

Figure 9: Violin Plot of Traffic dataset with integrated Boxplot

This Plot makes many more characteristics of the data visible then the previous ones: On days with a limit, the amount of accidents varies much less and while there were days with a lot of accidents, the amount was mostly much lower than on days without a limit.
The samples from a day without a limit vary much more and include days with a drastically higher amount of accidents, even with outliers removed. Comparing this to a violin plot of the original dataset, without the removed outliers, also shows that this preparation step was a good idea for this data:

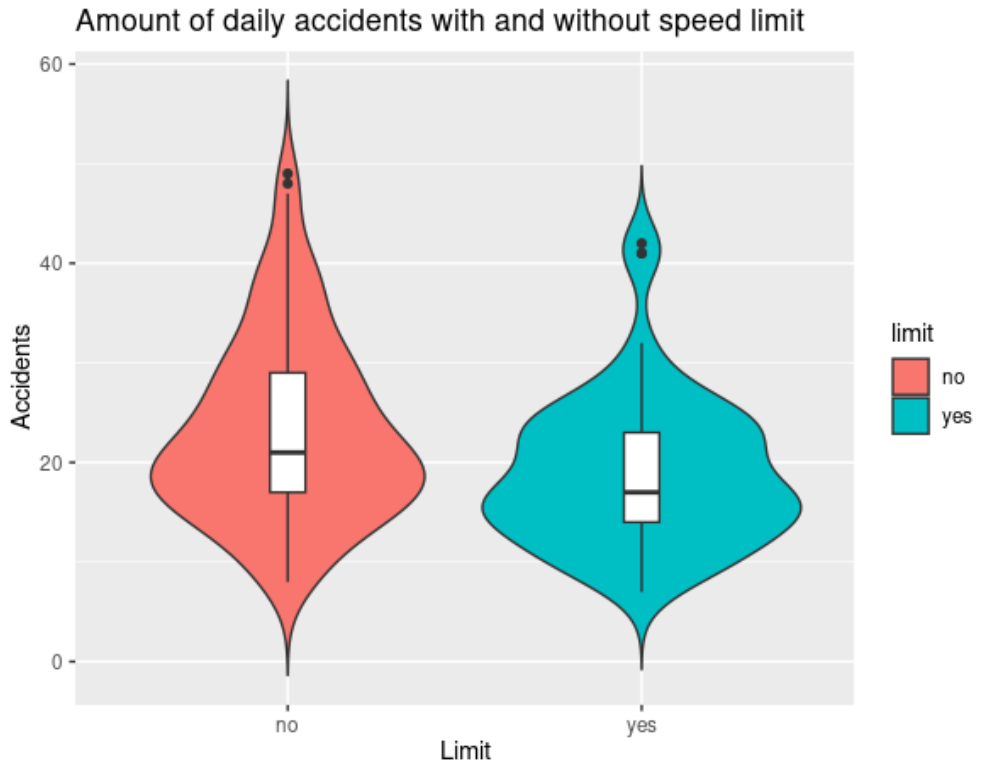

Figure 10: Violin Plot of the unprocessed Traffic dataset

Especially the few outliers of the "yes"-category could give a wrong impression of this group's accident amount extent.

A last question I wanted to answer was, whether there would be a visible difference between the two years. I wanted to try using a clustering diagramm for this. I knew that this would not make much sense for the x dimenion (days), but I wanted to do it anyway. Here it was actually useful, that I could render day n of both years at the same x coordinate, so I used the original *day* column instead of my *date_cont*:

```
ggplot(traffic_2, aes(x = day, y = y)) +
  geom_point(aes(color = paste0(year), shape=limit)) +
  geom_encircle(data = subset(traffic_2, year == 1961), aes(day, y, col=paste0(year))) +
  geom_encircle(data = subset(traffic_2, year == 1962), aes(day, y, col=paste0(year)))
labs(y = "Accidents", x = "Day")
```

(I had to cast the year property to a string in some places to make R understand, that it is categorial rather than continual)

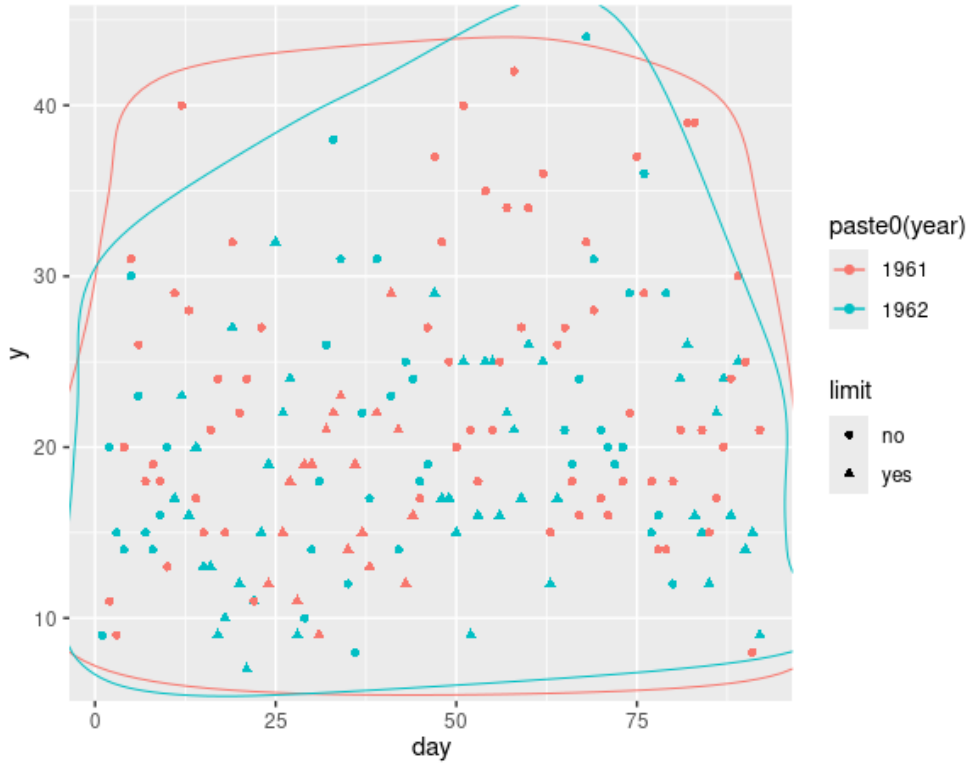

Figure 11: Scatterplot of the Dataset comparing both years

I think this answers the question pretty clearly; There was no significant difference between the two years. I created the same plot also on the two subsets of only samples with/without a speed limit and got similar results.

3.2.6 Conclusion

Looking back, I could have also chosen a different dataset, as I noticed the high variance of this one very quickly. This way I could have easily created beautiful graphs that show nice trends and correlation between their parameters. But I wanted to continue with this one anyway. It becomes quite clear that in the early sixties they did not intend this to be nice for modern data analysis and visualisation, but that is just the reality; real data may not be as nice to us, as we may hope.
So I embraced the challenge, to find some preparations and visualisations that would allow me to get more insight into this data. And with the group-based outlier removal and especially the violin plot I was quite happy. Even though a lot of information gets lost this way, it makes this highly scattered data quickly understandable, without distorting it in any way, like the smooth plot did.
What I learned from this is that to find a good visualisation of data, one has to play around with it and try out a few different things. Also, a bias of how one might expect the data to look like should not influence the work with it.
I enjoyed this exercise and am now less scared of R, then I was initially. I'm looking forward to do more sophisticated data analysis and visualisation with it.

4 Data Preprocessing & Visualization in Python

Lecture from: 28.01.2025
Held by: Gustav Pihlgren

As sadly I had a presentation from my home university during this lecture I could not attend it. Again, and also because of this, I would like to focus more on the exercise instead of the lecture.
The two given Jupyter Notebooks were very useful as a companion along the way and saved me again from a lot of frustration over too many StackOverflow articles.
This exercise was actually my first time using Python, however this did not feel as intimidating as R, as it shares a lot of similarities with programming languages I was already familiar with. I was however surprised by the also pretty declarative nature of pandas - you could definitley see the SQL influence. I think this design decision makes a lot of sense, takes a bit of time to get used to though. Some of the solutions I came up with could have probably been done much more elegantley had I followed this approach, instead of the very textbook-ish procedural ways I chose.

4.1 The Dataset

After browsing thru Kaggle a bit I setteled on a Spotify Music Dataset, as it consists of many interesting attributes for a lot of songs. Compositional Properties such as tempo, key and mode, but also extracted values from Spotify's musical analysis; "energy", "danceability" etc. . Only later I noticed that some other interesting properties were missing (namely "track_genre", data only has "playlist_genre"), but as this dataset is just a scrape of the Spotify API, it would not be so hard to get ahold of them.
The data consists of two csv files - one for low and one for high popularity. Looking at the size of both sets as well as their least and most popular song (thru the property "track_popularity"), I assume they just sorted the data by popularity and split it in half:

```
high = pd.read_csv('../data/high_popularity_spotify_data.csv')
low = pd.read_csv('../data/low_popularity_spotify_data.csv')

print('high', high.shape)
print('low', high.shape)
print('high max popularity level:', high['track_popularity'].max())
print('high min popularity level:', high['track_popularity'].min())
print('low max popularity level:', low['track_popularity'].max())
print('low min popularity level:' , low['track_popularity'].min())
```

```
>
high (1686, 29)
low (1686, 29)
high max popularity level 100
high min popularity level 68
low max popularity level 68
low min popularity level 11
```

Otherwise I could not explain to myself why songs with a popularity value of 68 would be in both datasets. This begs the question how the songs were chosen and how representative they are for current music in general. Sadly I could not find any information about that on the kaggle page.
Had I planned on doing actual research, I would have probably just scraped the Spotify API myself. Nevertheless I continued working and wanted to quickly take a look at the 5 most popular songs:

```
high = high.sort_values(by='track_popularity', ascending=False)
print(
    "Most popular:\n",
    high[["track_name", "track_id", "track_artist",
    "track_album_name", "track_popularity", "playlist_name"]].head(5)
)
```

```
>
Most popular:
            track_name                track_id           track_artist  \
676  Die With A Smile  2plbrEY59IikOBgBGLjaoe  Lady Gaga, Bruno Mars
455  Die With A Smile  2plbrEY59IikOBgBGLjaoe  Lady Gaga, Bruno Mars
0    Die With A Smile  2plbrEY59IikOBgBGLjaoe  Lady Gaga, Bruno Mars
677              APT.  5vNRhkKd0yEAg8suGBpjeY       ROSÉ, Bruno Mars
4                APT.  5vNRhkKd0yEAg8suGBpjeY       ROSÉ, Bruno Mars

     track_album_name  track_popularity      playlist_name
676  Die With A Smile               100      Global Top 50
455  Die With A Smile               100  Top Gaming Tracks
0    Die With A Smile               100   Today's Top Hits
677              APT.                98      Global Top 50
4                APT.                98   Today's Top Hits
```

No Taylor Swift - a surprise to be sure, but a welcome one. I'm assuming this dataset was created, when "Die with a smile" just came out. Here it was already clear to see that the dataset had duplicates and why it had them: They come from different playlists. Maybe that is how they retrieved the samples, by looking at the most popular playlists?
I was also wondering if just using the track_id I could reliably detect and remove all outliers. What if the same track is actually stored multiple times under different ids (single versions, remasters etc.). Where does it make sense to draw the line here? Should e.g. a live version of a song should be treated as a duplicate or not?

4.2 Cleaning and merging the data

Before getting ahead of myself though, I first wanted to combine the two datasets, as I see no point in keeping them seperate. If I want to classify them again as high/low popularity, I could just do that with the original threshold of 68 again.

4.2.1 Merging

Looking manually at the two set's columns they seem to be identical (but in a different order for some reason), but I wanted to quickly confirm it anyway with `print(sorted(high.columns) == sorted(low.columns))`, which returned *True*.
So I could just go ahead and merge them without any further manipulation of their columns:

```
songs = pd.concat([low, high], ignore_index=True).sort_values(by='track_popularity', ascending=False)
# Check if correctly concatenated & sorted
print(songs.head(5)["track_popularity"],"\n", songs.tail(5)["track_popularity"])
```

```
>
3145    100
3147    100
3146    100
3149     98
3148     98
```

```
Name: track_popularity, dtype: int64
3143    11
3129    11
3130    11
3128    11
3141    11
Name: track_popularity, dtype: int64
```

4.2.2 Removing Duplicates

As indicated before, I was not sure, whether detecting dupliactes by their id would be sufficient enough. So I also wanted to check for duplicates by title and artist. (I could have also gone more granularly, as this does not filter e.g. a "(Remaster)" at the the end of a song title, but for the sake of simplicity I decided against that. The discussion of what to count as a duplicate and what not would go far more into the musical domain at some point, I think)
First I wanted to count the amount of duplicates:

```
idDupes = songs.duplicated(subset=['track_id'], keep=False)
nameDupes = songs.duplicated(subset=['track_name', 'track_artist'], keep=False)


print("Duplicates by track_id:", idDupes.sum())
print("Duplicates by title and artist:", nameDupes.sum())
```

```
>
Duplicates by track_id: 631
Duplicates by title and artist: 671
```

So there is a good amount of duplicates not indicated by their ID. After taking a look at some of them with

```
display(songs.index[nameDupes].difference(songs.index[idDupes]))
```

I noticed a lot of Remaster, Deluxe etc. Album versions, but songs with the same musical characteristics. So I decided to go with the "title and artist" defintion for duplicates.
This created however the problem of vastly different popularity and genre values for each version of a duplicated song.
As especially the genre was only tracked thru the playlist, not the song itself I didn't want to just take one of them. There was a big "gaming playlist", which a lot of songs came from, which then hat the "playlist_genre" of "gaming", which I don't really recognize to be a proper music genre.
This is a real bummer, as I know that Spotify does definitely keep track of the song's genre, but it isn't included in the data here. This makes the genre values a bit unreliable. But because I liked the challenge of coming of with a solution for different values in multiple versions of the same song, I wanted to continue anyway:
I though it would be best to just take the average popularity of each duplicate and do a majority vote on the genre to leave in the one version, that will not get removed.
To do this I created the following routine:

```
songs['track_genre'] = songs['playlist_genre']

dupes = songs.loc[nameDupes].sort_values(by='track_name')

genres = []
popularities = []
current = {
    'artist': -1,
    'track_name': -1,
    'index': -1
}

def maj_vote():
    if(current["index"] < 0):
        return

    songs.loc[current["index"], 'track_genre'] = Counter(genres).most_common()[0][0]
    songs.loc[current["index"], 'track_popularity'] = sum(popularities) / len(popularities)

for i in dupes.index:
    title = songs.loc[i, 'track_name']
    artist = songs.loc[i, 'track_artist']

    # song is not a dupe of current
    if(current["artist"] !=  artist and current["track_name"] != title):
        # Assign popularity and genre values for last dupe
        maj_vote()

        # Continue with this song as new
        current["artist"] = artist
        current["track_name"] = title
        current["index"] = i
        genres = [songs.loc[i, 'track_genre']]
        popularities = [songs.loc[i, 'track_popularity']]
    else: # song is a dupe of current
        genres.append(songs.loc[i, 'playlist_genre'])
        popularities.append(songs.loc[i, 'track_popularity'])

dupes = songs.loc[nameDupes].sort_values(by='track_name')
display(dupes[["track_name", "track_popularity", "track_genre"]])
```

I could have gone for the much more elegant chatgpt solution I got, which leveraged panda's mighty SQL style declarative functionality much more, but I didn't want to just copy something blindly, without really understanding it. So I did it myself.
In the majority vote above, if there is no clear winner, simply the first encounter wins. I could have chosen the one with the highest popularity here or something similar, but again I wanted to keep it simple.
After scrolling thru the results a bit, I was happy with them.
(excerpt from the results:)

	track_name	track_popularity	track_genre	playlist_genre
3489	we can't be friends (wait for your love)	83.222222	pop	gaming
3444	we can't be friends (wait for your love)	81.000000	pop	pop
3263	we can't be friends (wait for your love)	86.000000	pop	pop

After this I could finally go ahead and remove the duplicates, as my routine saved the aggregations of all duplicates in the first encounter of the duplicated song.

```
songs = songs.drop_duplicates(subset=['track_name', 'track_artist'], keep='first')
# Check if removal was successful
print(songs.duplicated(subset=['track_name', 'track_artist'], keep=False)[lambda x : x].sum())
```

```
>
0
```

I got a zero, that means success. :)

4.3 Simplifying the data

As I wanted to only look at the musical properties and the popularity of the tracks, I could drop a lot of columns, mostly specific to album, playlist, spotify ids etc.:

```
cols_dropped = [
    'playlist_genre', 'track_href', 'uri', 'track_album_name',
    'playlist_name', 'analysis_url', 'track_id', 'track_album_id',
    'id', 'playlist_subgenre','type', 'playlist_id'
]

songs = songs.drop(columns=cols_dropped)
```

4.4 Test and Visualisations

One of my main research questions was, what properties of a song would have an influence on their popularity. So I wanted to do a quick correlation check on the data. As I also wanted to include genre, I wanted to make it numeric. In hindsight I should have known already that this would be a bad idea, as the order of which I would assign numbers to the genre would drastically influence the correlation. After having done this with

```
genres = list(songs['track_genre'].unique())
songs['genre_num'] = songs['track_genre'].apply(lambda g : genres.index(g))
```

I found a (relatively) significant correlation of -0.358979, which of course just came from the fact that above I took the genres from the data that was sorted by popularity. Later I shuffled the genre's order with `random.shuffle(genres)`, which had correlation values much closer to just zero.
(That's why later I just created a Boxplot for genres and popularity)
The correlation check `songs.corr(numeric_only=True)` returned pretty disenchanted values for the track_popularity:

	track_popularity
instrumentalness	-0.263717
acousticness	-0.232541
time_signature	0.003822
mode	0.004517
genre_num	0.015106
speechiness	0.018749
liveness	0.021718
duration_ms	0.021744
key	0.028142
tempo	0.059147
valence	0.096289
danceability	0.127116
energy	0.193995
loudness	0.216811
track_popularity	1.000000

Instrumental and acoustic songs tend to be less popular than loud and energetic ones in the samples from the dataset, but not by a very significant amount.
There was however some significant correlation ($|cor| \geq 0.5$) between especially the properties from spotify's musical analysis. The highest I could find was between energy and acousticness; -0.751047.

4.4.1 Genre Boxplot

To get a better glimpse of the realtionship between genre and popularity, I wanted to create a Boxplot, to compare the genres. To keep it more staightforward I extracted the 10 most popular genres with:

```
popularities =
    songs.groupby('track_genre'['track_popularity'].mean().reset_index()
    .sort_values(by='track_popularity', ascending=False)

pop_genres = popularities['track_genre'].to_numpy()[:10]

print(pop_genres)

>
['r&b' 'k-pop' 'gaming' 'metal' 'punk' 'rock' 'j-pop' 'indie' 'pop' 'hip-hop']
```

(Again, I don't like this 'gaming' genre in here, it does pose the question of how reliablethe playlist_genre really is for the individual track...)
Then I created the Boxplot:

```
data = songs[songs["track_genre"].apply(lambda g : g in pop_genres)][
        ["track_popularity", "track_genre"]
    ]

fig = plt.figure(figsize=(20, 10))

sns.boxplot(data=data, x='track_genre', y='track_popularity', order=pop_genres)

plt.xlabel('Genre', fontsize=20)
plt.ylabel('Popularity', fontsize=20)

plt.show()
```

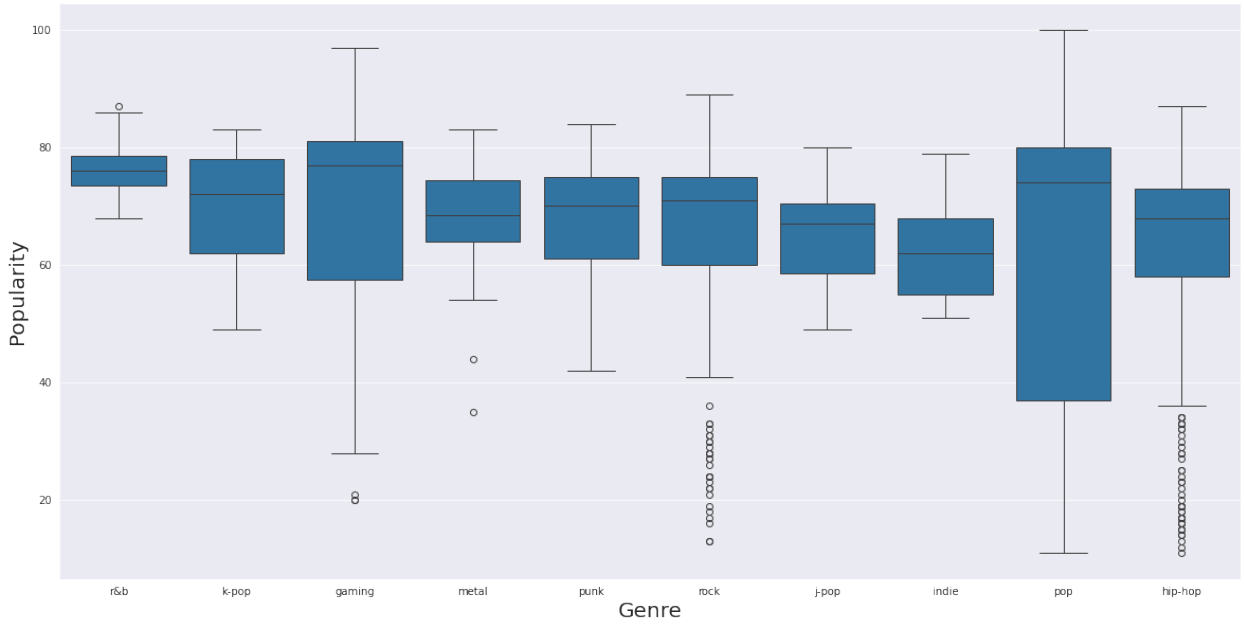

Figure 12: Boxplot of the 10 most popular genres, displaying their popularities' ditributions

Interesting, so pop music varies a lot (which makes sense I would say), while rock and hip-hop music tend to be very popular, but have a lot of outliers, and r&b songs are all very close to each other in terms of popularity.
How reliable is that statement though? To evaluate this I wanted to count the times genres actually occured:

```
print(data["track_genre"].value_counts())
```

```
>
track_genre
pop          441
hip-hop      350
rock         304
gaming       111
punk          71
r&b           47
metal         30
j-pop         23
k-pop         17
indie         17
Name: count, dtype: int64
```

Well, besides the top 3 genres here I would definitely not call that enough data.
Along the way I also tried running `display(songs[songs['track_genre'] == 'blues'])`,
which returned a lot of Justin Bieber, Eminem, Rihanna etc. from a playlist called "Classic Blues".
I didn't know if this should make me laugh or cry to be honest. (On the domain level and the data level...)
So I would say the genre data in this dataset can simply not be trusted. What a shame...

4.4.2 Energy & Acousticness Scatterplot

After this huge disappointment i wanted to treat myself with a very pretty chart. So as again mentioned, energy and acousticness had a high correlation, I wanted to create a scatterplot with the two:

```
x = songs["acousticness"]
y = songs["energy"]

fig1 = plt.figure(figsize=(10, 10))
plt.scatter(x, y)
plt.xlabel('Acousticness', fontsize=20)
plt.ylabel('Energy', fontsize=20)
plt.show()
```

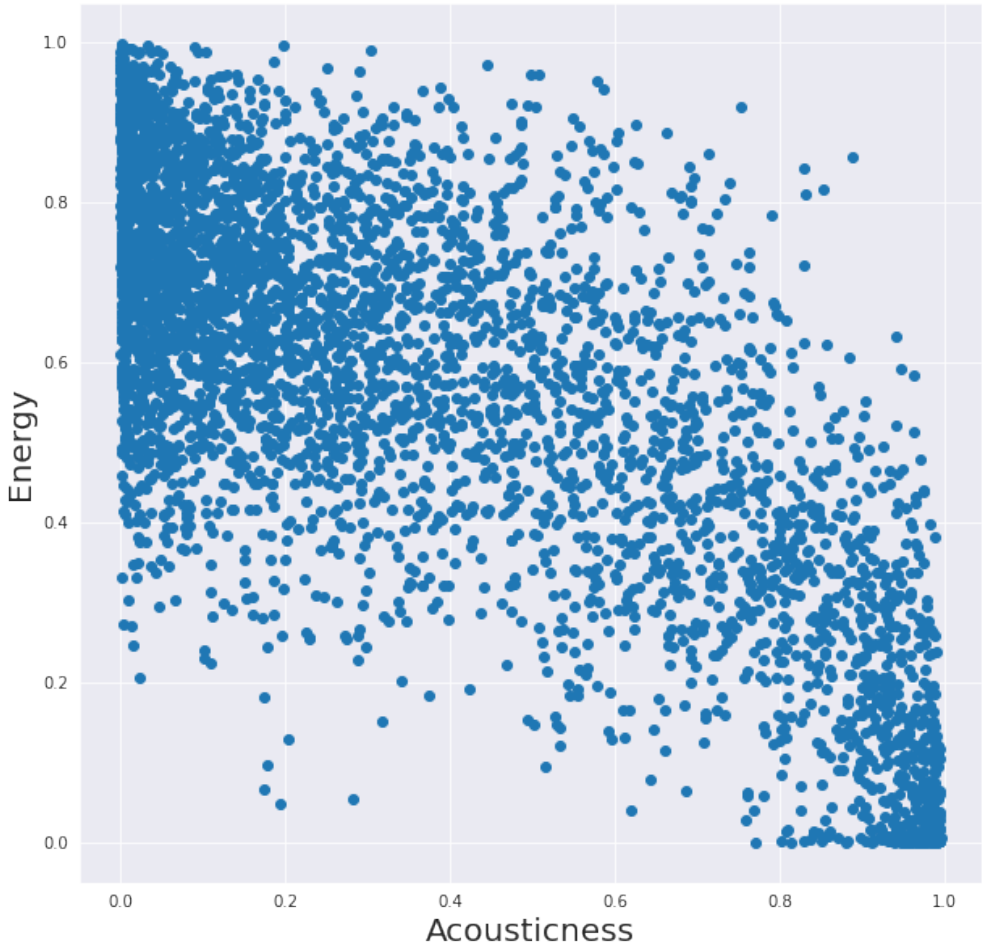

Figure 13: Scatterplot displaying the correlation of energy and acousticness

This allows for a lot of nice insights:

- The data contains mostly songs with low acousticness and high energy
- There's clear trend of more acoustic songs having less energy
- There are some outliers, but most of the data follows this pattern

4.5 Questionnaire

1. **Can you find any flaws with your dataset of choice that are not already covered by the material?**
   Not really flaws we haven't talked about, but a problem; How to aggregate different values of rows that one wants to merge (duplicates in this example).
   In this case I was happy with the solution I came up with, but I think that in other domains it might be harder to tell whether a majority vote, taking the values' mean / median or something completely different is the best solution

2. **Do you know what the different columns of data represent and can you verify that they are decently correct?**
   Mostly yes, the columns from spotify's musical analysis saved values between 0 and 1, so it is easy to interpret them as percentages. To understand their true meaning and how Spotify retrieves those values it would be a good idea to check their API reference.
   Also the "mode"- and "key"-columns safes numbers, instead of musical description like "c major", but again this mapping should be to find in spotify's api reference.

3. **Is there a way to tell beforehand if a visualization will be informative?**
   Sometimes yes, e.g. by checking the covariance of two properties can tell you wheter a Scatterplot will be helpful or not.
   In this case I could have also counted the amount of songs for all genres to see beforehand that the Boxplot will not be very meaningful for most genres, that only had about 20 songs in the whole dataset.

4. **Is there something to learn from your visualization that is difficult to understand by only looking at the tabular data and its aggregations?**
   I think so. The Scatterplot not only shows the correlation, but also the density of energetic/acoustic songs present in the data.
   If the genre values could actually be trusted and if each genre had more songs in the dataset, the Boxplot could also give you a much better understanding of each genre's distribution of popularities.

4.6 Conclusion

I really enjoyed this exercise and seeing the differences between how data analysis can be done in R and Python. As I am rather familiar with traditional programming languages I would probably prefer to use Python for the pre-processing, but definitely R for the visualisations; With Pandas, Python also gets a lot of declarative functionality, without sacrificing the 'classical' approaches more familiar to most programmers. The amount of effort it takes to create just one graph though is really frustrating, after having experienced how easy it can be in R.
As now I also worked with a "real" dataset (vs. the 3 columns and 182 observation I chose for my R exercise) it was interesting to see and experience this "real data is messy", one always hears, in reality. Even a dataset that looks somewhat clean by just quickly glimpsing over a table representation of it can hide a lot of problems, that need to be solved.
Here I stumbled on most problems by accident, which is a very bad approach. This goes to show how important a proper understanding of data pre-processing and common problems with data and possible solutions is.
In hindsight I also noticed that I clang on to the genre too much, as this was something I wanted to analyse, before looking at the data. After having seen, how unreliable the genre values are in this dataset, I should have rather dropped my interest in it and worked with other values, that were far more reliable.
What I take from this is, that our excitement / interest before looking at the data should not influence the trust you put into it. One should be honest with themselves if they feel that the data is not reliable and leave emotions aside.

5 Mathematical methods for data (pre-) processing, filtering, basic analysis

Lectures from: 18.02.2025 & 21.02.2025
Held by: Per Arnqvist

5.1 PCA vs LDA vs. K-means

While PCA and LDA are both multivariate techniques that try to reduce high-dimensional data to 2 or sometimes 3 dimensions in order to allow human-understandable visualisations, there are fundamental differences to there design and goal in mind.
While the goal of PCA is mainly to capture the maximum variance within the data to build it's new axis (the "principal components") on, LDA tries to provide the best possible seperation between different given classes of the data. This makes LDA a supervised method, as classes for the whole dataset must be provided, while PCA is unsupervised on the other hand, as it doesn't need more than the unlabeled data.
The resulting principal components are linear combinations of the original dimensions, so it is possible to retrieve which dimensions influence them the most.
On the other hand, LDA rather tries to maximize the distance between the means of each class and to minimize the variation within each category. From that it draws a transformation from all dimensions to 2-3 new ones, that maximizes the separability between the classes.
K-means clustering is more different from the two. It's goal is simply to cluster given data into a predefined amount of clusters. While it an unsupervised method, it does need the amount of clusters to create.
The way it works is much simpler than PCA and LDA, yet it is still a powerful technique:
It simply places n "centroids" (middle point for each class) randomly within the data. Then all datapoints are assigned to the class of their nearest centroids. (measured in euclidean distance).
After that the centroids are moved to the middle of the new clustered data. If now all datapoints still have the same centroid closest to them, the algorithm is finished. If not, the process of classifying each point to the class of it's closest centroid and moving the centroids gets repeated until the before mentioned situation is reached.

5.2 Principal Component Analysis

5.2.1 On the original data

When using PCA on the raw data and checking it's summary and loadings, in this case the first PCA actually describes basically the whole variance within the data and ends up being just identical to the *total.sulfur.dioxide* column of the data:

```
raw_pca <- princomp(wine5)
summary(raw_pca)
raw_pca$loadings
```

```
>
Importance of components:
                          Comp.1      Comp.2      Comp.3       Comp.4       Comp.5
Standard deviation     34.4464783 1.283816603 0.729816501 1.704154e-01 1.476545e-01
Proportion of Variance  0.9981227 0.001386437 0.000448045 2.442935e-05 1.833952e-05
Cumulative Proportion   0.9981227 0.999509186 0.999957231 9.999817e-01 1.000000e+00

Loadings:
                   Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
volatile.acidity                            0.640  0.763
total.sulfur.dioxide  1.000
sulphates                                  -0.761  0.646
alcohol                     -0.769  0.639
```

```
quality                     -0.635 -0.765  0.105

                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
SS loadings        1.0    1.0    1.0    1.0    1.0
Proportion Var     0.2    0.2    0.2    0.2    0.2
Cumulative Var     0.2    0.4    0.6    0.8    1.0
```

When looking at the dataset it quickly becomes obvious why: the range of this column with a maximum value of 289 is much higher than the ones form the other columns.
Plotting the PCA then will basically just show the total.sulfur.dioxide on the first axis and basically irrelevant data on the second, as PCA2 just has a 0.1% proportion of the variance.
For a later comparison I did it anyway:

```
ggplot(data.frame(raw_pca$scores), aes(x = Comp.1, y = Comp.2, color=paste0(wine5$quality) )) +
  geom_point(size = 1) +
  labs(x = "PCA 1", y = "PCA 2")
```

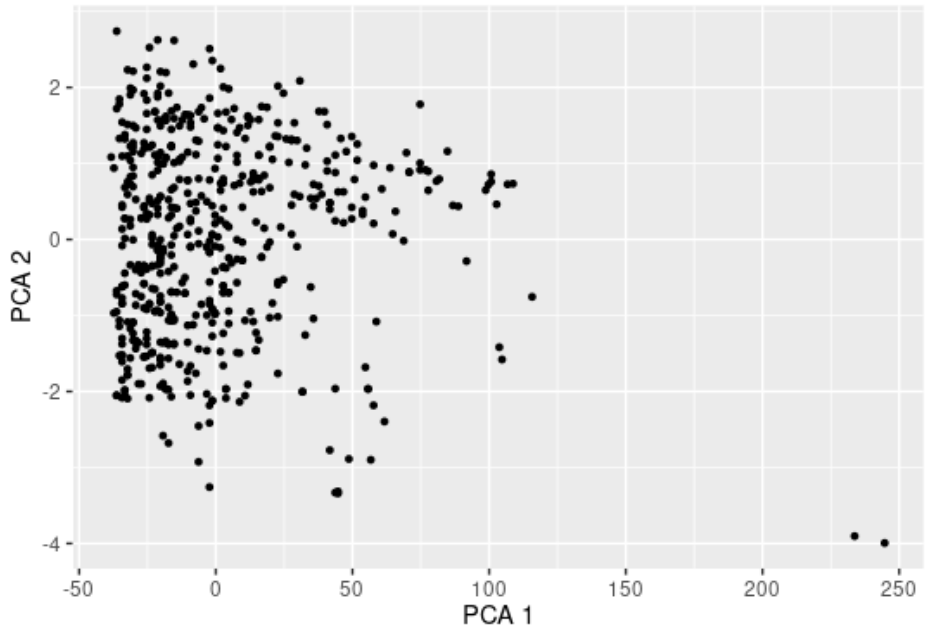

Figure 14: PCA performed on the raw dataset

We can see that are few outliers for the total sulfur dioxide (PCA 1), but this is basically the only real helpful insight to gain from this.
To get a comparison to the LDA, later I also did a PCA with the wine quality removed from the data before and it being color coded in the chart:

```
raw_pca <- princomp(wine5[, 1:4])
ggplot(data.frame(raw_pca$scores), aes(x = Comp.1, y = Comp.2, color=paste0(wine5$quality) )) +
  geom_point(size = 1) +
  labs(x = "PCA 1", y = "PCA 2")
```

Figure 15: PCA performed on the raw dataset with 'quality' removed and being indicated by color

As the colors are all mingled, it is hard to draw any conclusions on wine quality.
One last thing I did before standardizing the dataset was creating the PCA with the correlation matrix instead of the covariance matrix, as this is supposed to also be a way to get a reliable PCA graph on unstandardized data:

```
raw_pca <- princomp(wine5, cor = TRUE)
```

Interestingly this gave me the exact same PCA as the one I did on the standardized dataset. As I know that both approaches would lead to the scaling of the data being basically ignored, I expected similar results, but not the same. This goes to show how unintuitive (yet interesting) I find statistics.

5.2.2 On the standardized data

The summary and loadings of the PCA standardized data (or as mentioned on the raw data with the correlation matrix for PCA) looks much more promising:

```
scaled_wine <- scale(wine5)
scaled_pca <- princomp(scaled_wine)
summary(scaled_pca)
scaled_pca$loadings
```

```
>
Importance of components:
                          Comp.1    Comp.2    Comp.3    Comp.4     Comp.5
Standard deviation     1.4485746 1.0415768 0.8739523 0.8053877 0.62868857
Proportion of Variance 0.4204353 0.2173703 0.1530358 0.1299653 0.07919333
Cumulative Proportion  0.4204353 0.6378056 0.7908414 0.9208067 1.00000000

Loadings:
                   Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
volatile.acidity      0.500  0.220  0.167  0.723  0.387
total.sulfur.dioxide  0.196 -0.767 -0.573  0.208
```

```
sulphates            -0.346 -0.533  0.698  0.306 -0.123
alcohol              -0.493  0.280 -0.365  0.583 -0.453
quality              -0.591        -0.152         0.792

                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
SS loadings        1.0    1.0    1.0    1.0    1.0
Proportion Var     0.2    0.2    0.2    0.2    0.2
Cumulative Var     0.2    0.4    0.6    0.8    1.0
```

Even though we can see that the variance is actually nicely represented in PCA 1 and 2 and, when looking at the loadings, actually comes from all columns, the graph still ends up being pretty disappointing:

```
ggplot(data.frame(scaled_pca$scores), aes(x = Comp.1, y = Comp.2))+
  geom_point(size = 1) +
  labs(x = "PCA 1", y = "PCA 2")
```

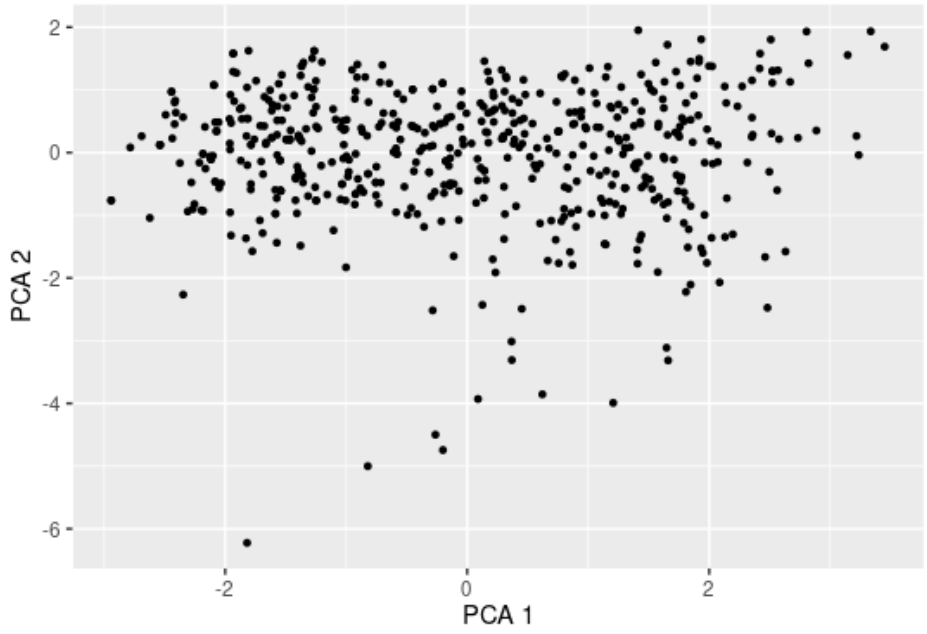

Figure 16: PCA performed on the standardized dataset

Besides a few outliers with lower PCA 2 values it all just looks like one big cluster. Altough it is apparent that besides those outliers all data lies mostly above -2 PCA 2, is however spread all over the more important PCA 1.
So when looking at the loading of PCA 2 it can be said that most of the data tends to have lower total sulfur dioxide and lower sulphates, as those columns have a big negative loading on PCA2.
Again out of curiosity I dropped the quality column and indicated it trough color in the graph. This time I also added ellipses around them for some "mock clusters":

```
scaled_wine <- scale(wine5[, 1:4])
ggplot(data.frame(scaled_pca$scores), aes(x = Comp.1, y = Comp.2, color=paste0(wine5$quality)  )) +
  geom_point(size = 1) +
  stat_ellipse(aes(fill = factor(wine5$quality))) +
  labs(x = "PCA 1", y = "PCA 2")
```

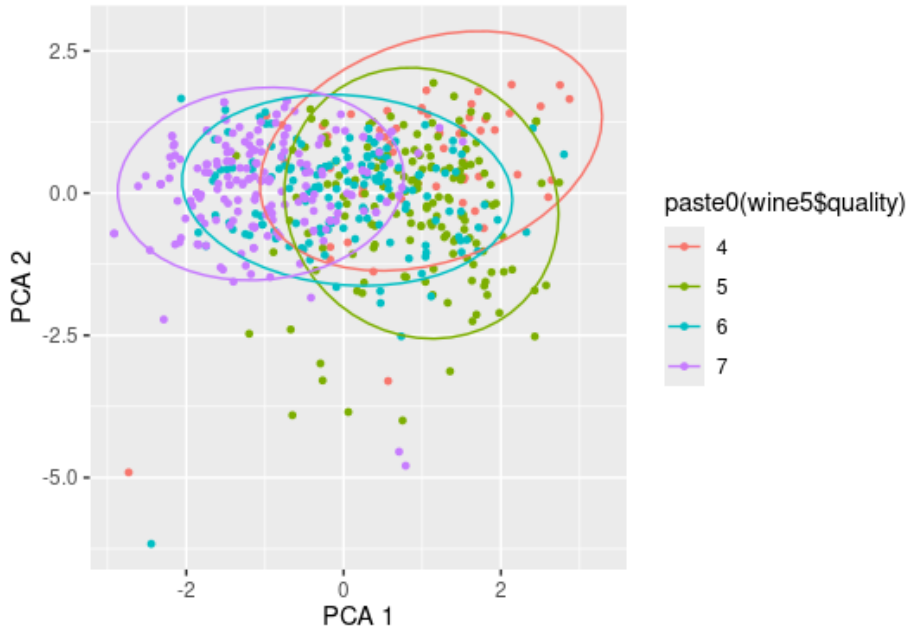

Figure 17: PCA performed on the stanardized dataset with quality removed, indicated by color and encircled

While the "clusters" are overlapping a lot, nevertheless a trend for the wine quality to get lower for higher PCA1 values is visible. When looking at the summary and loadings of the pca this finally allows for some insight into the data:

```
scaled_wine <- scale(wine5[, 1:4])
scaled_pca <- princomp(scaled_wine)
summary(scaled_pca)
scaled_pca$loadings
```

```
>
Importance of components:
                          Comp.1    Comp.2    Comp.3    Comp.4
Standard deviation     1.2291886 1.0414841 0.8652407 0.8053064
Proportion of Variance 0.3784117 0.2716644 0.1875001 0.1624238
Cumulative Proportion  0.3784117 0.6500761 0.8375762 1.0000000

Loadings:
                    Comp.1 Comp.2 Comp.3 Comp.4
volatile.acidity     0.612  0.208  0.280  0.710
total.sulfur.dioxide 0.262 -0.775 -0.535  0.212
sulphates           -0.468 -0.522  0.646  0.303
alcohol             -0.581  0.290 -0.468  0.600

               Comp.1 Comp.2 Comp.3 Comp.4
SS loadings      1.00   1.00   1.00   1.00
Proportion Var   0.25   0.25   0.25   0.25
Cumulative Var   0.25   0.50   0.75   1.00
```

PCA1 does hold a little less proportion of the variance than before, but still a significant amount. When looking at biggest absolute values of this loading we can say that the quality tends to be higher for wines with a lower volatile acidity (big positive value) and more sulphates and alcohol (big negative values). So again, as with Whisky, people just like wine more, when it gets them drunk more quickly.

5.3 Linear Discriminant Analysis

5.3.1 On the original data

As LDA is a supervised learning task, the data must be provided with labels. For this I again used the wine quality and ignored it in the lda itself. Checking its results looked good:

```
linear <- lda(quality~. - quality, data = wine5)
linear
```

```
>
Prior probabilities of groups:
         4          5          6          7
0.09601449 0.27173913 0.27173913 0.36050725

Group means:
  volatile.acidity total.sulfur.dioxide sulphates  alcohol
4        0.6939623             36.24528 0.5964151 10.26509
5        0.5829333             59.30333 0.6262000  9.86000
6        0.4896000             44.38000 0.6696667 10.52767
7        0.4039196             35.02010 0.7412563 11.46591

Coefficients of linear discriminants:
                             LD1         LD2        LD3
volatile.acidity     -3.15207152  4.62436062 2.85090417
total.sulfur.dioxide -0.00753553 -0.01616900 0.02288596
sulphates             1.98242633  0.04497137 2.10512493
alcohol               0.79227004  0.42446975 0.49687829

Proportion of trace:
   LD1    LD2    LD3
0.9089 0.0855 0.0056
```

While most of the trace is presented on LD1, at least it's not completely determined by one column, like it was the case for the PCA on the raw dataset.
The resulting graph however looks less promising:

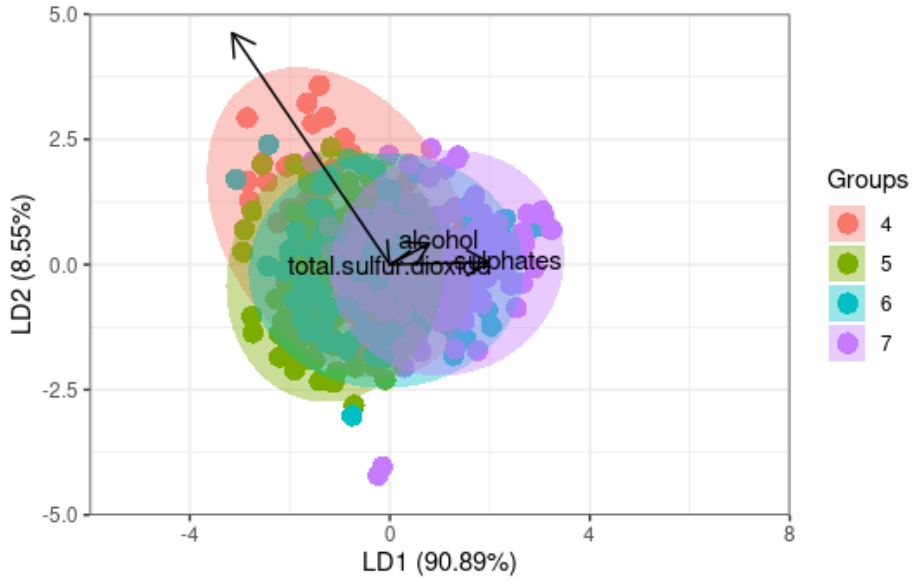

Figure 18: LDA performed on the raw dataset, with wine quality removed and set as class label

While there is a subtle trend visible of quality increasing with higher LD1 values, which when looking at the coefficients again comes from low volatile acidity (big negative value) and higher sulphates and alcohol (big positive values), there are no nice seperated clusters visible as expected.
At least it can provide a confirmation of the trend already seen in the last PCA. Also the length of the *total.sulfur.dioxide* vector clearly shows it's much bigger domain in the raw dataset.

5.3.2 On the standardized data

I was hoping to get much nicer clusters with the standardized data now. interestingly when looking at the proportions of trace of the three linear discriminants, they stayed the same, but the coefficients of the columns changed:

```
scaled_wine <-  as.data.frame(scale(wine5))
scaled_linear <- lda(quality~. - quality, data = scaled_wine)
scaled_linear
```

```
>
Prior probabilities of groups:
 -1.88969738530301 -0.893409938610308  0.102877508082399   1.09916495477511
        0.09601449         0.27173913         0.27173913         0.36050725

Group means:
                  volatile.acidity total.sulfur.dioxide   sulphates    alcohol
-1.88969738530301        1.03204985         -0.233045689 -0.46979381 -0.3530071
-0.893409938610308       0.42980340          0.435758444 -0.29535381 -0.7157932
0.102877508082399       -0.07645806          0.002903676 -0.04078441 -0.1178580
1.09916495477511        -0.54120826         -0.268582394  0.37849148  0.7223973

Coefficients of linear discriminants:
                            LD1          LD2       LD3
volatile.acidity     -0.5811095  0.852537723 0.5255869
total.sulfur.dioxide -0.2597990 -0.557451160 0.7890285
```

```
sulphates             0.3384911  0.007678675 0.3594413
alcohol               0.8846649  0.473971607 0.5548245

Proportion of trace:
   LD1    LD2    LD3
0.9089 0.0855 0.0056
```

Besides the length of the *total.sulfur.dioxide* vector however, the LDA graph stayed the same:

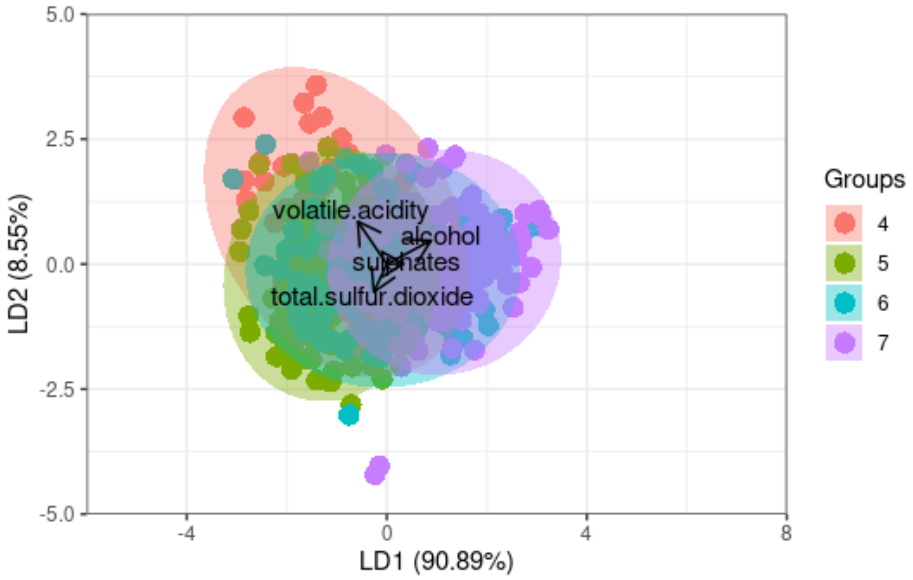

Figure 19: LDA performed on the raw dataset, with wine quality removed and set as class label

As the whole design of LDA is aimed at providing maximal seperation between clusters, I was really disapointed with the results here. They did indicate the small trend for the wine quality depended on 3 factors, but those I had already disocverd through PCA. I guess, when LDA can not seperate them better, there may simply not lay the foundation for a good seperation within the data.
But I will put this to the test with K-means!

5.4 K-means Clustering

5.4.1 On the original data

As there are 4 distinct wine quality values within the data, I wanted the clustering to create 4 clusters, hoping that it could identify the characteristics that contribute to the wine quality. For that I of course removed the quality column for the clustering:

```
km <- kmeans(wine5[, 1:4], 4, 100)
fviz_cluster(km, data = wine5[, 1:4],
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw()
)
```

Figure 20: K-means clustering performed on the raw dataset with quality column removed

While especially clusters 2 and 3 are mingled a lot and all of them lay within the whole domain of dimension 1, there is a clear trend on the second dimension visible. To interpret this result, I printed the centers of the clusters:

```
print(km$centers)
```

```
>
  volatile.acidity total.sulfur.dioxide sulphates   alcohol
1        0.5367857             86.24405 0.6622619 10.307937
2        0.4917481             19.63910 0.6693609 10.887594
3        0.5052247             47.30337 0.6856742 10.587360
4        0.5089583            148.08333 0.7404167  9.891667
```

This shows that besides the total sulfur dioxide all clusters have similar means for the rest of the columns. As before I'm assuming that this column just influences the clustering way to much as it has a much higher domain than the rest and influences the variance displayed in dimension 2 unproportionally.
But before addressing this by normalizing the data for the k-means, I wanted to render a second diagram that also displays the actual quality class each point belong to.
At first I wanted to display the quality by different shapes, which however made the diagramm hard to interpret. So instead I chose color to display it, knowing that it might harm the understandility with the clusters a bit, but that's the best result I was able to get. I chose to use the same colors for each quality that R had automatically assignend before in the PCA and LDA:

```r
get_color <- function(quality) {
  colors <- rep(NA, length(quality))
  colors[quality == 4] <- "red"
  colors[quality == 5] <- "green"
  colors[quality == 6] <- "blue"
  colors[quality == 7] <- "purple"

  return(colors)
}

km <- kmeans(wine5[, 1:4], 4, 100)
fviz_cluster(km, data = wine5[, 1:4],
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw()
) +
  geom_point(aes(color = get_color(wine5$quality))) +
  scale_color_identity()
```

Figure 21: K-means clustering on the raw dataset with color indicating wine quality

Now, while this is really ugly and the clusters are a bit hard to assign to their respective index, a bit of structure seems to be visible:
Clusters 2 and 3 (the lowest ones on dimension 2) seem to mostly capture most of the higher quality wine (blue and purple dots). So when looking at the summary from the kmeans above again, we can see that those were the clusters with the lower total sulfur dioxide.
So this observation, we had already made before can be done even on the raw dataset with kmeans.

5.4.2 On the standardized data

Performing k-means on the normalized data gives a much prettier diagramm with nicely seperated clusters:

```
km_scaled <- kmeans(scale(wine5[, 1:4]), 4, 100)
fviz_cluster(km_scaled, data = wine5[, 1:4],
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw()
)
```

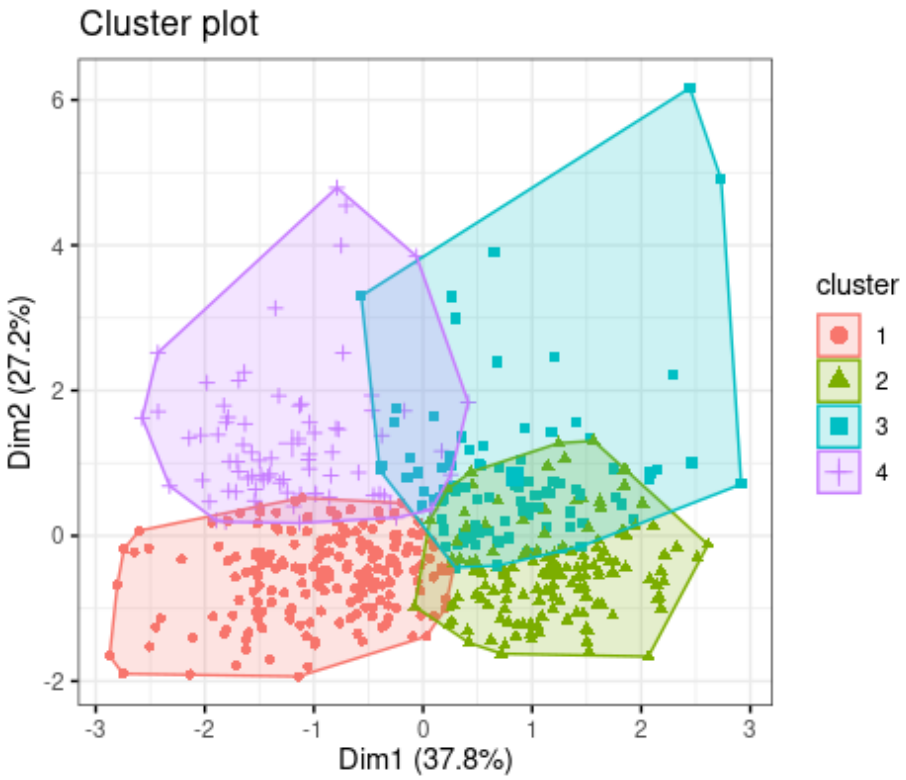

Figure 22: K-means clustering performed on the standardized dataset with quality column removed

When looking at the centers of all columns in the clusters, we can clearly characterize every cluster:

```
print(km_scaled$centers)
```

```
>
  volatile.acidity total.sulfur.dioxide  sulphates    alcohol
1       0.75055967          -0.31671726 -0.5214433 -0.4730880
2      -0.57197010          -0.40883393  0.1148585  1.2016547
3      -0.68225414          -0.04813479  1.2057076 -0.2612139
4       0.09324481           1.71218576 -0.5036581 -0.7853974
```

So wines in cluster 1 tend to have a higher volatile acidity and lower sulphates and alcohol, while those in cluster 2 contain higher alcohol and have less volatile acidity. So it is fair to assume that the variance of those columns is most likely presented mostly in dimension 1, as that's where the first two clusters differ. Total sulfur dioxide however seems to be represented in the other dimension as here the two clusters have similar values and have roughly the same domain in the second dimension.

This is also visible in cluster 4, which stretches the furthest in dimension 2 to a few outliers and has the highest average total sulfur dioxide of all clusters. Wines in there also tend to have less alcohol, and sulphates, while in cluster 3 the sulphate levels tend to be higher and the overall volatile acidity much lower.
With those insights, let's indicate the wine qualities again by color in another very ugly chart:

```
km_scaled <- kmeans(scale(wine5[, 1:4]), 4, 100)
fviz_cluster(km_scaled, data = wine5[, 1:4],
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw()
) +
  geom_point(aes(color = get_color(wine5$quality))) +
  scale_color_identity()
```

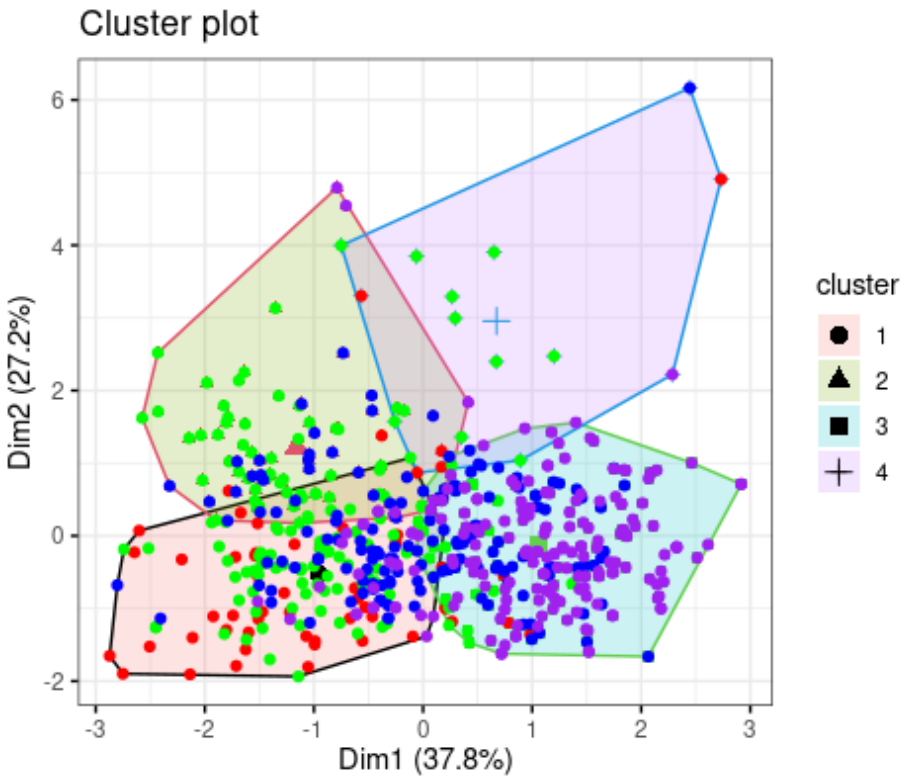

Figure 23: K-means clustering on the standardized dataset with color indicating wine quality

Despite the diagram's atrociousness, we can see that the best quality wines (purple) seem to be almost all within cluster 2, while the second best wines are spread among clusters 1 and 2.
The lower quality wines (red and green) seem to spread between clusters 1 and 4.
So, again the result is that the best ranked wines in this dataset are the ones with more alcohol and less volatile acidity.

5.5 Conclusion

So in conclusion, all three introduced methods to handle high dimensional data and transform it into nicely visualizable two dimensions work reasonably well, even on a pretty scattered dataset. Even though they all follow slightly different approaches and have different goals in mind, I could draw the same conclusions

regarding the wine quality from all of them. As they are methods made to create visualisation they can not be used to draw final conclusions for a thorough data analysis, but are great to get a first intuition for the data being worked with. For PCA and k-means clustering the data should definitley be scaled however.
When looking at especially cluster 3 from Figure 22, a outlier removal would probably also have been good to get even better insights into the data. So while all the introduced methods might seem like magic, they do need some amount of pre-processing.

6 Interactive graphs and pages, Image Manipulation and 3D imaging

Lecture from: 04.03.2025
Held by: Kary Främling

6.1 3D Visualisations

6.1.1 Scatterplot

I decided that I wanted to start with some examples inspired by the *Visualisation3D.Rmd* file. As I could not get *devtools* installed however, I also had problems with a lot of other packages, even when trying to install and build them directly from source. To compensate for that I also took some inspiration from the further resources provided in Canvas.
The scatterplot and histogram I was able to render in the file, so I decided to recreate them first. With `data(package = .packages(all.available = TRUE))`, I again outputted all directly available datasets and found an interesting one in the "math"-dataset from the "doBy"-package, which contains grades from 88 students in the subjects mechanics, vectors, algebra, analysis and statistics. (I'm assuming "vectors" here just means linear algebra)
I started with a first rudimentary scatterplot for the algebra, analysis and statistics grades:

```
x <- math$al
y <- math$an
z <- math$st

scatter3D(x, y, z,
      xlab = "Algebra",
      ylab = "Analysis",
      zlab = "Statistics",
      bty="b2",
      pch=19
)
```

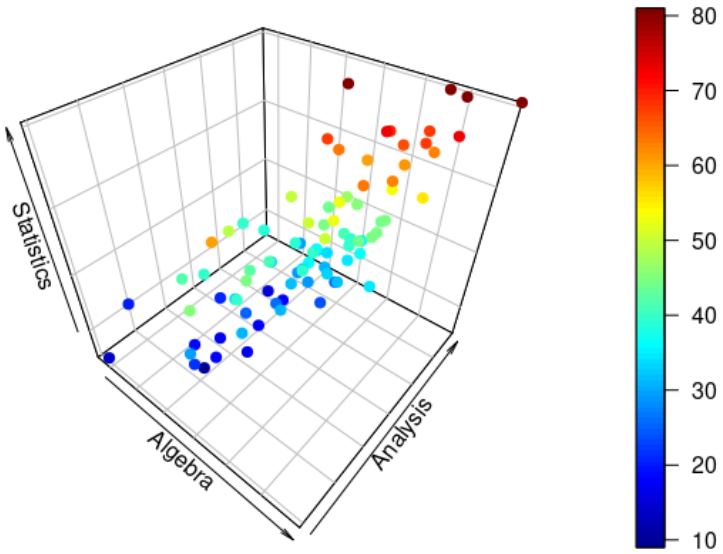

Figure 24: 3D scatterplot displaying the algebra, analysis and statistics grade from the math dataset

Here we can see a pretty clear trend in students with good grades in one subject also passing better in the others too. Some however also manage to get good statistics grade with more mediocre algebra grades, while algebra and analysis grades seem to be a bit more correlated. When thinking about the similarities between the 3 subjects, that seems reasonable.
Taking a quick look at the actual correlation confirms this observation:

```
cor(math)
```

```
>
          me        ve        al        an        st
me 1.0000000 0.5534052 0.5467511 0.4093920 0.3890993
ve 0.5534052 1.0000000 0.6096447 0.4850813 0.4364487
al 0.5467511 0.6096447 1.0000000 0.7108059 0.6647357
an 0.4093920 0.4850813 0.7108059 1.0000000 0.6071743
st 0.3890993 0.4364487 0.6647357 0.6071743 1.0000000
```

After realizing that I don't have to use the colors for just another representation of the z-axis, I wanted to use it for a fourth dimension, which provided me with a good small research question: "How do the grades in different mathematical disciplines affect the mechanics grade of a student?"
So with a bit of trial and error getting a good colored representation for the mechanics grade of a student I ended up with this graph:

```r
colors <- colorRampPalette(c("blue", "red"))(10)[as.numeric(cut(math$me, breaks = 10))]

scatter3D(x, y, z,
      clab = c("Mechanics"),
      col = colors,
      xlab = "Algebra",
      ylab = "Analysis",
      zlab = "Statistics",
      bty="b2",
      pch=19,
      ticktype = "detailed"
)
```

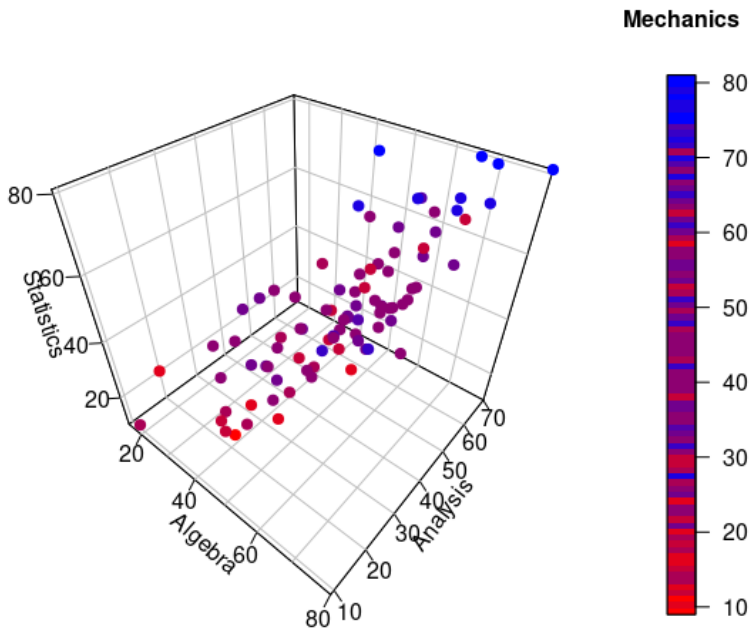

Figure 25: 3D scatterplot displaying the algebra, analysis and statistics grade from the math dataset, with mechanics being additionally displayed by color

Here it is quite easy to see that the mechanics grade also tends to follow the overall trend of students being good in one subject also being good in the rest. There are however a certain amount of blue points (indicating a good mechanics grade), that are significantly lower on the z-axis, indicating a worse statistics grade. When taking a look at the correlation again, those two dimensions are the ones with the lowest value of 0.3890993.
When thinking of the relevance of all mathematical subjects for mechanics this does also seem quite reasonable.

6.1.2 Linear regression

In one of the additional resources from Canvas, I found a simple linear regression to put into a 3D Scatterplot, So I wanted to try it out too:

```
fit <- lm(z ~ x + y)

grid.lines = 26
x.pred <- seq(min(x), max(x), length.out = grid.lines)
y.pred <- seq(min(y), max(y), length.out = grid.lines)
xy <- expand.grid(x = x.pred, y = y.pred)
z.pred <- matrix(predict(fit, newdata = xy),
                 nrow = grid.lines, ncol = grid.lines)

scatter3D(x, y, z,
      clab = c("Mechanics"),
      col = colors,
      xlab = "Algebra",
      ylab = "Analysis",
      zlab = "Statistics",
      bty="b2",
      pch=19,
      cex=0.5,
      ticktype = "detailed",
      surf = list(x = x.pred, y = y.pred, z = z.pred, alpha=0.65)
)
```

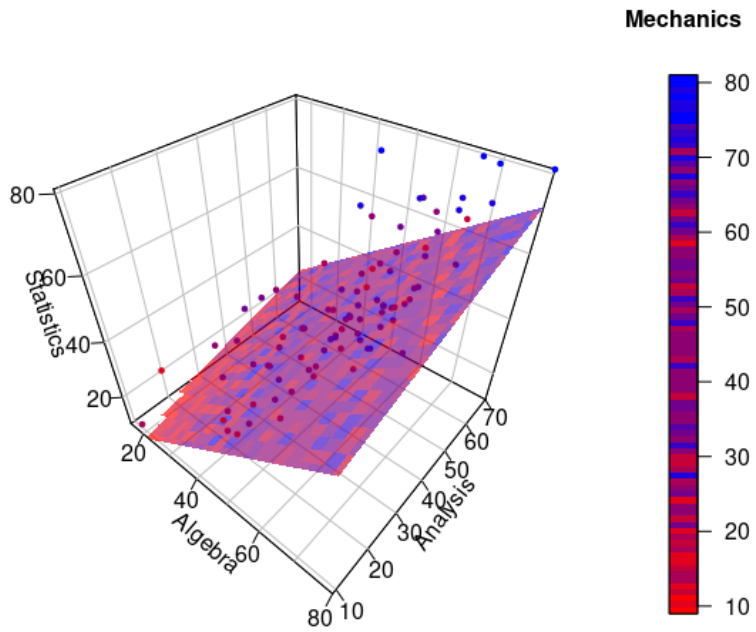

Figure 26: 3D Scatterplot of the math dataset with a linear regression plane

I have tried graphs with both the color coding for the mechanics grade and without. I included the one with it here and decreased the opacity of the plane a bit to make the rest a bit more visible.
This is a lot of information packed into a small 3D plot, but I think it is still viable.
We can nicely see the already discovered trend of good students being good across the board and that it is

quite strong for algebra and analysis, but less strong for statistics and either of the other two. When trying to imagine just the line the plane creates when ignoring the third dimension, those observations are being possible when concentrating a bit.
I would however not try to cram even more information into this chart, as already for the mechanics grade we can again see that there are some blue square on the plane, indicating some mechanics students with good grades and mediocre math grades, but it is hard to tell on which of the 3 dimensions this is dependent on.
I have tried to use the "plot3Drgl"-librabry to make the 3D plots dragable, but I was sadly not able to get it to work.

6.1.3 Histogram

I also had a few problems getting the method from *Visualisation3D.Rmd* to work, that computes the number of classes for the histogram to work, so I just tried out a couple of values manually and settled on 15 as a good sweetspot between too much data being summed up in one class or too many classes to really understand the distribution of the data. I think the resulting histogram allows for quite good insights:

```
x.breaks <- seq(min(math$ve), max(math$ve), length.out = 15)
y.breaks <- seq(min(math$me), max(math$me), length.out = 15)

xy.table <- table(cut(math$ve, x.breaks), cut(math$me, y.breaks))

z <- as.matrix(xy.table)

x.mids <- (head(x.breaks, -1) + tail(x.breaks, -1)) / 2
y.mids <- (head(y.breaks, -1) + tail(y.breaks, -1)) / 2

hist3D(
  x = x.mids,
  y = y.mids,
  z = z,
  scale = TRUE,
  border = "black",
  xlab = "vectors",
  ylab = "mechanics",
  zlab = "frequency",
  ticktype = "detailed"
)
```

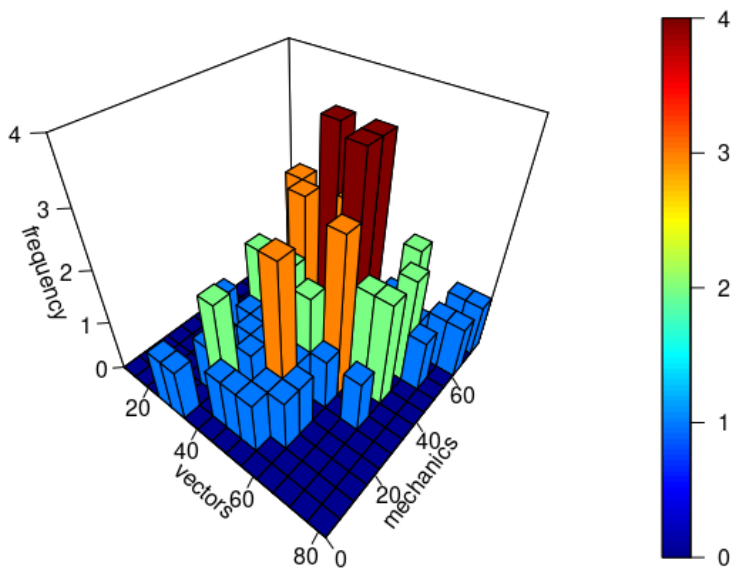

Figure 27: 3D Histogram of the mechanics and vectors grades from the math dataset

When looking at both dimensions individually it can first be said, that the grade distribution of the vectors grades look quite like a normal distribution, with the majority of the students getting mediocre grades and increasingly less getting very bad or very good grades.
In mechanics the grades are more skewed towards better grades (the dark red classes), but still a lot of grades within midrange (the orange classes).
Both dimensions combined, this histogram visualizes nicely, that most of the data lies on the "diagonal" between the two dimension, meaning that there are no/just a few students with drastically better or worse grades in one than in the other subject. The majority however tends to be good in mechanics and mediocre in vectors.
One downside of such a 3D Histogramm is, that big classes can completely hide classes behind them; It is hard to see the all classes high on the mechanics and low on the vectors dimension. From most of what is visible, we can assume that those classes have very low frequencies, but we can't tell for sure. There could be a significant amount of student with an A+ in mechanics and and an F in vectors, we just can't see it.
Here I would have really liked to get *plot3Drgl* to work, to observe this better, but as mentioned before, I sadly could not.

6.2 Working with images

For this section I took some inspiration from the *WorkWithImages.Rmd*-Demo and tried out a few things with mainly *Magick* and *Tesseract*.

6.2.1 Quantizing images

I really liked the color histograms they did to compare the soil differences of denmark. Here I did something slightly different: I compared my hometown Opladen with Umeå.
To do so I took a quick google earth screenshot from both of them and followed the approach from the slides.

I tried to have roughly the same zoom level when taking the screenshots for a fair comparison. Sadly all the google earth pictures seem to be from may, so I could not get an image of Umeå covered in snow, as I was hoping to in the beginning.
For Opladen I started with this image:

Figure 28: Picture of Opladen from google earth

So mostly residential area, with a a few fields and small forests outside and a small river (the Wupper). Had I chosen to take the whole of city of Leverkusen around Opladen, you could also see the Rhine, which would take up much more of the picture then the Wupper does here.
Quantizing it to 6 colors gave this result:

```
quantized <- image_quantize(opladen, 6, colorspace = 'YCbCr')
quantized
```

Figure 29: Picture of Opaden quantized into 6 colors

I did the same thing with this image from Umeå:

Figure 30: Picture of Umeå from google earth

So here more forest in between, all in all brighter colors and of course the river and Nydala lake. The quantized picture than looked like this:

```
quantized <- image_quantize(umea, 6, colorspace = 'YCbCr')
quantized
```

Figure 31: Picture of Umeå quantized into 6 colors

Then to get the histograms displaying the frequency of colors for each picture I followed the slides as such:

```
orig1 <- image_crop(opladen, '500x500')
hist1 <- image_crop(quantized, '500x500') %>% count_colors %>% plot_hist()
opladen_final <- image_append(c(orig1, hist1))
```

```
opladen_final

orig1 <- image_crop(umea, '500x500')
hist1 <- image_crop(quantized, '500x500') %>% count_colors %>% plot_hist()
umea_final <- image_append(c(orig1, hist1))
umea_final
```

Resulting in:

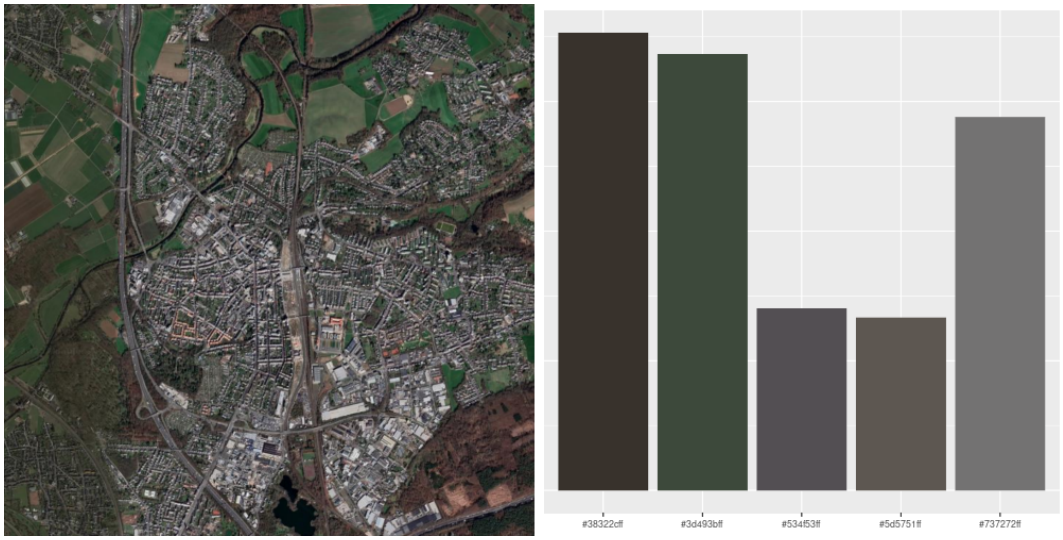

Figure 32: Histogram of color frequencies in quantized picture of Opladen

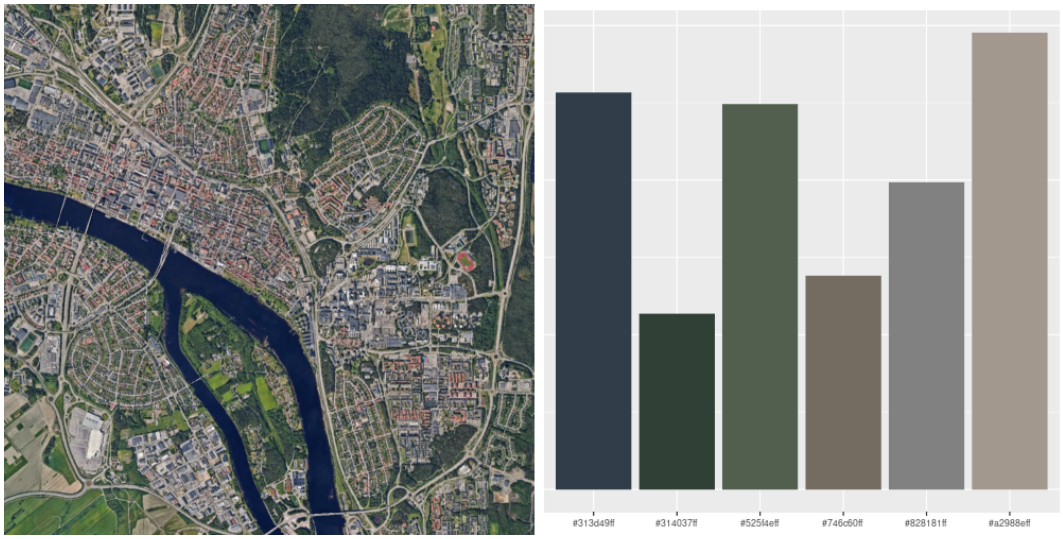

Figure 33: Histogram of color frequencies in quantized picture of Umeå

A proper scientific comparison would probably need much more work being done. As an example in the quantized picture of Umeå, some of the forest ended up with the same color as the water from the river and the lake. So experimenting with the amount of colors to quantize the images, as well as some other steps might be necessary.
Nevertheless, I think the results are quite good and represent broadly what I would have expected, having

experienced both cities:
Blue is of course much more represented in the picture of Umeå as to be expected for a city at the coast. It is also a tad greener and the housing area seems to be in brighter colors than in Opladen. However this might also have to do with the position of the sun, when the pictures have been taking.
I also only noticed at the end that the histogram of Opladen only contains 5 colors, even though I definitley used 6 as parameter for the quantizing of the image. So it could be assumed, that the there is just very little variety in the colors of Opladen. (Having lived there for 18 years, I can confirm that. It is not the most pretty town.)

6.2.2 OCR

Lastly, after failing to get OpenCV installed, I decided to mess around with tesseract and its OCR capabilites instead.
Being drawn away a bit too much by the example of deciphering a medieval manuscript from the beginning of the slides, I wanted to try OCR on a handwritten letter with a very cursive font I found online:

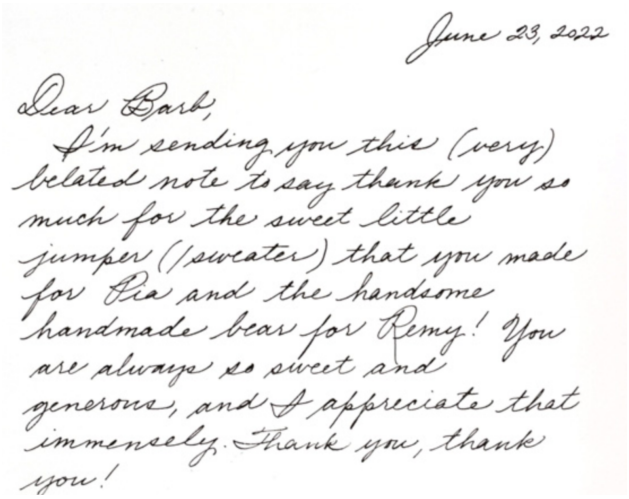

Figure 34: Handwritten letter for OCR

As I have a very hard time to even read this myself, I can in hindsight not understand, how I could be so optimistic with this.
I imported the image anyway, converted it to a grayscale as shown in the slides and parsed it to tesseract's OCR:

```
letter %>%
  image_convert(type = 'grayscale') %>%
  image_ocr() %>%
  cat()
```

```
>
Por ee
hoa Lut
; lewtater/) thal priaole'
Vinwonsthy ak yar, thawte
ste 40 tr ormedag./
Meg
Fpencleeg.
```

I would have expected at least a couple of words. With that I would have maybe tried a bit more pre-processing to improve the results, but it is just plain obvious that the OCR is completely overwhelmed with

this image, so I gave it an easier challenge instead:
I looked up one of my favorite music album's CD back covers and imported it into R:

Figure 35: Back cover of the Van Halen Best of Volume 1 CD for OCR

I imagined this to be a fair challenge for the OCR; Much easier than the letter, but harder than a simple image of a few printed words.
I cropped it a bit with Magick, so that the image would only contain the song list and the big "VAN HALEN"-text underneath and put it through the same processing as the letter:

```
cd <- image_crop(cd, "460x550+200+80")

cd %>%
  image_convert(type = 'grayscale') %>%
  image_ocr() %>%
  cat()

>
Pao tury CoNGEr Te)
Pe UNAS hi CLE Reg
POOR Ven eer ag tn tcTs
Pee arta Cee NT Ca ICY!
3. AND THE GRADLE WILL ROCK... 3:3
Peel ruta etry
5 Pat Crecy
PI TPCT S
Pere amor ima TtCe Som Rohs rey
Fm Sow eee
poem 2°82. eee eat L
12, POUNDCAKE 5:22
Pe Coram Coated
14, CAN'T STOP LOVIN' YOU 4:08
1. HUMANS BEING 5:10
```

```
16. CAN'T GET THIS STUFF NO MORE 5:14
17, ME WISE MAGIC 6:05
```

Ah yes, "Pe UNAS hi CLE Reg", that's my favorite song!
Interestingly enough for a couple of songs this worked perfectly, but for the rest it did not at all. The result is however much more motivating than the one from the letter.
To improve the performance I had the idea of inverting the image's color before running the OCR, so that it would be dark text on a light background, as I assumed that that's rather what the OCR model has been trained on:

```
image_negate(cd) %>%
  image_convert(type="grayscale") %>%
  image_ocr() %>%
  cat()
```

```
>
1. ERUPTION 1:42
2. AIN'T TALKIN' "BOUT LOVE 3:47
3. RUNNIN' WITH THE DEVIL 3:32
4. DANCE THE NIGHT AWAY 3:04
3. AND THE GRADLE WILL ROCK... 3:3
6. UNCHAINED 3:27
5 7. JUMP 4:04
8. PANAMA 3:31
9. WHY CAN'T THIS BE LOVE 3:45
10. DREAMS 4:54
11, WHEN IT'S LOVE 5:36
12. POUNDCAKE 5:22
1g. RIGHT NOW 5:21
14, CAN'T STOP LOVIN' YOU 4:08
1. HUMANS BEING 5:10
16. CAN'T GET THIS STUFF NO MORE 5:14
17, ME WISE MAGIC 6:05
```

Besides some punctuation marks and the big "VAN HALEN"-lettering underneath completely missing this worked out perfectly, all the song names have been detected. So with just a tiny bit of pre-processing it is really easy to improve the OCR performance drastically.
If we were to ignore that Spotify and the internet are a thing, I could easily imagine this being part of my old routine of copying CDs to my mp3-player, so that I don't have to type out all the song titles myself.

6.3 Conclusion

There is so much more to do with R and data visualisation and images. I liked that last outlook on what is more to do there than just creating a few simple 2D plots, especially with 3D graphics, the possibilities just become more in amount and complexity.
While 3D graphics can be very useful to analyze higher dimensional data one has to always be careful to not overdo it. It happens quite easily that the resulting graph looks really pretty, but makes it hard for humans to interpret. Then using tools to drag, turn and zoom around the plot can be very useful to fully grasp the data. But when in doubt, creating multiple simpler graphs with less information is always the better alternative I think.
And I was quite surprised on what can all be down comparatively easy with images. Before looking at the slides from Münster, I was quite unsure on what to do with images. The map example really struck with me, I think quantizing images like this has even much more potential than just doing a quick ad-hoc analysis of a city's areas.
And while calling OCR "easy" in this way is of course pretty naive, applying it with such a nice library

as tesseract and tweaking the image with a bit of processing to better fit to the model, was a very nice introduction to this of course much more complicated and topic.

7 Conclusion

What do I think I have learned from this course as a whole?
I would say that, given a dataset and a research question, I know feel comfortable to 1.) analyze the characteristics of the dataset, 2.) find out, whether the dataset contains the necessary information to answer the question, 3.) prepare the dataset to then 4.) answer the question properly and then finally 5.), visualize the answer appropriately in different ways.
However I also learned to use visualisations of the data not just for the end result but also along the way during the process to understand the characteristics of the data and identifying the pre-processing steps needed to continue working with it.
As an example, I could of course find out through various data summaries that removing outliers is necessary, but had I just done a quick scatterplot or histogram I would have seen that immediately.
In all chapters of this learning I wanted to follow the order of steps mentioned above by finding small interesting question to answer with the introduced tools, instead of just going through all methods quickly just to tick all the boxes.
It was also by first university course with a learning diary instead of an exam and I have to say I really enjoyed working on it. It took quite some time sometimes, but I liked the freedom of the exercise and always felt like I had really learned something after finishing a chapter. Afterwards I always started to wonder how I could used the applied methods for personal projects.
I am finishing this course now with a solid toolkit for simple data analysis and two new programming languages to write on my resumes.
Especially during the project I came, like many of my fellow students, to the insight that pre-processing is much more fun with python, but for visuals I would choose R any time of the day.