Data Science - A practical Approach

Lorenz Feyen

Dec 09, 2021

CONTENTS

pdf version can be found here.

Part I

1. Introduction

CHAPTER

ONE

INTRODUCTION

this is an introduction

1.1 Structured vs Unstructured

When performing data preparation an important aspect is to consider with the type of data we are working with. In general there are 2 types of data, but you could consider a third.

1.1.1 Structured data

Structured data is data that adheres to a pre-defined data model and is therefore straightforward to analyze. This data model is the description of our data, each record has to be conform to the model. A table in a spreadsheet is a good example of the concept of structured data however often no data types are enforced, meaning a column can contain e.g. both numbers and text. Later we will see that a mixture of data types is often problematic therefor the need of a data model.

1.1.2 Unstructured data

In contrast to structured data, there is no apparent data model but this does not mean the data is unusable or cluttered. Usually it means either no data model has yet been applied or we are dealing with data that is difficult to confine in a model. A great example of this would be images, or more general (binary) files. These obviously are hard to sort yet often data structures also contain metadata from these files, with data describing things as when the file was uploaded, what is shown in the file, … In turn the metadata can be structured and a data model can be related to the unstructured data.

1.1.3 Semi-structured data

As an intermediate option, we have what is called semi-structured data. The reasoning behind this is that the concept of tables is not always applicable, in some occasions e.g. data lakes there is no complex structure present compared to a database. In a data lake files are stored similar to the folder structure in your computer, with no fancy infrastructure behind it, thus reducing operation costs. This implies that a data model can not be enforced and the data is stored in generic files.

1.2 Data Structures

There are several structures in which data can be stored and accessed, here we cover the 3 most important.

1.2.1 Data Lake

As mentioned earlier a data lake would be the most cost efficient method as it relies on the least infrastructure and can be serverless. The concept behind a data lake is straight-forward, the data is stored in simple files with a specific notation e.g. parquet, csv, xml,… What is important when designing a data lake would be partitioning, this can be achieved by using subfolders and saving parts of the data in different files. To make this more tangible, take a look at this symbolic example I provided. Instead of putting all data in one csv file, subfolder divide the data in Country, City and then the year. We could even further partition yet the data is here in daily frequency so that would create many small partitions. The difficulty for a data lake lies in the method of interacting, when adding new data one has to adhere to a agreed upon data model that is not enforced, meaning you could create incorrect data which then need to be cleaned. On the other hand efficiency of you data lake depends on good partitioning, as the order of divisioning of your folders. We could have also divided first on year and then on country and city. As a data scientist seeing the data lake might not be as common, as this is rather an engineering task, however using the concepts of a data lake in experimental projects can make a big difference.

1.2.2 Database

Another interesting data structure is the database, widely used for exceptional speeds and ease of use, yet costly in storage. Numerous implementations of servers using the SQL language are developed over the years with each their own dialect and advantages. The important take home message here is that you can easily perform queries on the database that pre-handles the data to retrieve the information you need. these operations include filtering, grouping categories, joining tables, ordering and much more, as SQL is a complete language on its own. As a data scientist these databases are much more common, so SQL is a good asset to learn!

1.2.3 Data Warehouse

A next step towards data analysis is the data warehouse, where a database is composed of the most pragmatic method of storing your data a data warehouse consist of multiple views on your data. Based upon the data of a dataset the data warehouse transforms this data into a new format that displays the data in a new way. Let me illustrate with with a simple example, we have a database with a table that contains the rentals of books from multiple libraries. This table has a few columns: a timestamp, the library, the action (rent, return, …), the client_id and the book_id. If you would want to know if a book is available this database is perfect for your needs as you just have to find the last event for that book and if its a return the book is (or should be) there. Now image we would want to know how many books are being rented per month this database is insufficient, yet our data warehouse might contain such a view! It is up to the data engineer/scientist to create a computation that displays the amount of books rented per month. If they also would like to subdivided it per category of books, you would need to incorporate another table of the database where information of the books is stored. More on these operations of a data warehouse will be seen in the data preprocessing chapter. One last remark about data warehousing, it is important to optimize between memory and computation. Tables in our data warehouse compared to database can be computed in place reducing memory costs yet increasing computation costs. If a visualization tool often queries a table in your warehouse it is favorable to create it as a table in your database.

1.3 OLTP and OLAP

From the previous section you might have deduced that a database and Data Warehouse serve 2 different purposes. These are denoted as OnLine Transaction Processing and OnLine Analytical Processing, as the names suggest these are used for transactional and analytical processes.

1.3.1 OLTP

For this method the database structure is optimal, let us review the example where we have libraries renting out books. Renting out a book would send a message to our OLTP system creating a new record stating that specific book is at this moment rented out from our library. OLTP handles day-to-day operational data that can be both written and read from our database.

1.3.2 OLAP

In the case we would like to analyse data from the libraries we would use the OLAP method, creating multi-dimensional views from our transactional data. Our dimensions would be the date (aggregated per month), the library and the category of book, the chapter of data preprocessing will use these operations practically. I could write a whole chapter on OLAP operations however they are well described in this wikipedia page.

Part II

2. Data Preparation

CHAPTER TWO

INTRODUCTION

When performing data science, we often do not elaborate about the preparation that went into the dataset. It is considered tedious and irrelevant to the story of the analysis, however it is often the most important part of data analysis. Data Preparation is the metaphorical foundation of your construction, if you fail to prepare data, you prepare to fail your analysis.

> Good data beats a fancy algorithm

If you would perform an analysis and insert unprepared data, you will mostly be disappointed with the result.

2.1 why Data Preparation?

Aside from metaphors let us make the reasoning behind this step more tangile, to explain the relevance of this step, we partitioned the answer into a few key points.

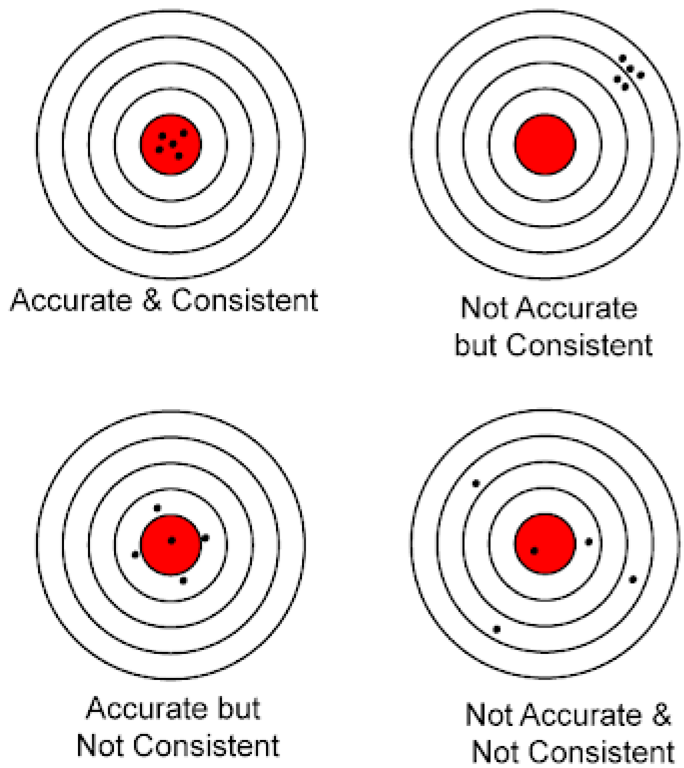
2.1.1 Accuracy

There is no excuse for incorrect data and accuracy is the most important attribute. Let us assume that we have a dataset where for some reason the result are not accurate. This would led us to an analysis where we conclude a result that contains a bias. An example would be a dataset of sold cars, where the listed price is that of the stock car without options. Options are not incorporated in the price and we are perhaps training an algorithm that predicts the stock price. If you as a data scientist fail to report/correct this, your predictions are making sense, but always underestimate!

2.1.2 Consistency

They usually say something such as 'consistency is key' and with data preparation that is likewise true. A dataset where we do not have consistent results will never converge towards a particular answer. Note however that it might not be a problem of consistency but rather you are missing crucial information. If we would have a dataset where local temperatures are logged, we would like to see a consistency each 24 hours. However we do see there are day to day fluctuations, so perhaps we need to keep track of cloud and rain data to make the dataset more complete. We could then see that the results are more consistent yet the possibility of outliers is still present. Equally possible would be that our temperature sensor is not sensitive enough or has large fluctuations in readings, it is the task of the data scientist to figure this out.

To get a visual about accuracy and consistency this picture might help:



2.1.3 Completeness

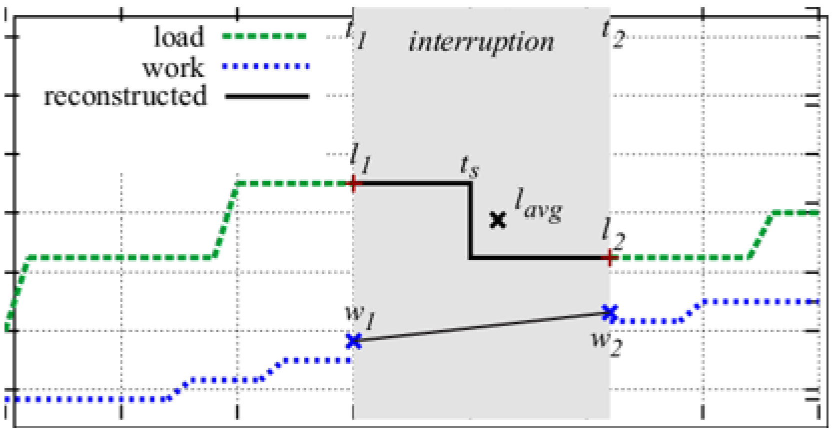
As hinted in the previous point, completeness is something you have to be aware of. Having 'complete' data is crucial for you narrative to give a correct answer, as you might otherwise lose detail. Note that you never will know if your data is complete as there might always be more data to mine. Yet you have to make a consideration between collecting more data and the effort required. This collecting can happen in multiple methods, as an example we use a survey where we asked several people 10 different questions, we could:

- gather new data, here our data grows 'longer' by asking the 10 question to more people. It might be that our sample of people were only students at a campus, so our data was not complete.
- gather new feature, by asking more questions to the same people (in case we could still find them). By doing this we get a better understanding of their opinion, again making our data more complete.
- fill missing values, by imputing the abstained questions with answers of similar records. When someone answered they did not want to answer we could figure out what they would have answered by looking at what persons answered that reply in a similar way.

2.1.4 Timeliness

For some datasets we are dealing with data that is time related. It can happen that data at specific timepoints is missing or delayed, resulting in a failure to use machine learning algorithms. A well-organised data pipeline utilises techniques of data preparation to circumvent these outages, usually this would be to retain the last successful datapoint. However in hindsight we could use more complex strategies to fill in these gaps or correct datetimes in our dataset,

In this example the data stream is interrupted and data preparation is there to handle these outages before we can perform analysis.



2.1.5 Believability

You could collect the most intricate dataset possible, but if the narrative that you are conducting contradicts itself, you will end up nowhere. During the process of data analytics it is important to apply a critical mind to what your dataset is telling you. Obviously this is not a reason to mask or mold the data so it agrees with your opinion. Rather you should be wary when conflicts happen and act accordingly, unfortunately it is impossible to write a generic tactic for this. As a data scientist your experience of the underlying subject should help create understanding of the topic, remember, gathering information from experts in the field is crucial here!

2.1.6 Interpretability

Another problem that might arise when you are diving deep into the data might be that you have created something no human could ever interpret. The Machine Learning algorithms outputs plausible and believable results, but it is impossible to understand the reasoning behind. For some this is perfectly acceptible, for some this is undesirable. It is your task as a data scientist to cater the wishes of the product operator and if they desire understanding as they would like to learn from the data driven process you need to unfold the process. Usually this comes down to which data transformations are used as some do produce an output that only makes mathematical sense.

2.1.7 In conclusion

There are multiple ways to deteriorate the quality of your data and raw formats of data often contain multiple. Before we can do anything with it these problems need to be resolved, if you fail to do so, the final output fails too.

2.2 Further reading

Towards Data Science

CHAPTER

THREE

MISSING DATA

In this notebook we will look at a few datasets where values from columns are missing. It is crucial for data science and machine learning to have a dataset where no values are missing as algorithms are usually not able to handle data with information missing.

For python, we will be using the pandas library to handle our dataset.

```
import pandas as pd
```

3.1 Kamyr digester

The first dataset we will be looking at is taken from a physical device equiped with numerous sensors, each timepoint (1 hour) these sensors are read out and the data is collected. Let's have a look at the general structure

```
kamyr_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c2_data_preparation/data/kamyr-digester.csv')
kamyr_df.head()
```

```
  Observation  Y-Kappa  ChipRate  BF-CMratio  BlowFlow  ChipLevel4   \
0    31-00:00    23.10    16.520     121.717  1177.607      169.805
1    31-01:00    27.60    16.810      79.022  1328.360      341.327
2    31-02:00    23.19    16.709      79.562  1329.407      239.161
3    31-03:00    23.60    16.478      81.011  1334.877      213.527
4    31-04:00    22.90    15.618      93.244  1334.168      243.131

   T-upperExt-2   T-lowerExt-2    UCZAA  WhiteFlow-4   ...  SteamFlow-4   \
0        358.282         329.545  1.443      599.253  ...        67.122
1        351.050         329.067  1.549      537.201  ...        60.012
2        350.022         329.260  1.600      549.611  ...        61.304
3        350.938         331.142  1.604      623.362  ...        68.496
4        351.640         332.709    NaN      638.672  ...        70.022

   Lower-HeatT-3  Upper-HeatT-3   ChipMass-4   WeakLiquorF   BlackFlow-2   \
0        329.432         303.099      175.964      1127.197      1319.039
1        330.823         304.879      163.202       665.975      1297.317
2        329.140         303.383      164.013       677.534      1327.072
3        328.875         302.254      181.487       767.853      1324.461
4        328.352         300.954      183.929       888.448      1343.424

   WeakWashF   SteamHeatF-3   T-Top-Chips-4   SulphidityL-4
0     257.325         54.612         252.077             NaN
1     241.182         46.603         251.406           29.11
```

(continues on next page)

(continued from previous page)

```
2      237.272          51.795          251.335            NaN
3      239.478          54.846          250.312          29.02
4      215.372          54.186          249.916          29.01

[5 rows x 23 columns]
```

Interesting, there seem to be 22 sensor values and 1 timestamp for each record. As mechanical devices are prone to noise and dropouts of sensors we would be foolish to assume no missing values are present.

```
kamyr_df.isna().sum().divide(len(kamyr_df)).round(4)*100
```

```
Observation          0.00
Y-Kappa              0.00
ChipRate             1.33
BF-CMratio           4.65
BlowFlow             4.32
ChipLevel4           0.33
T-upperExt-2         0.33
T-lowerExt-2         0.33
UCZAA                7.97
WhiteFlow-4          0.33
AAWhiteSt-4         46.84
AA-Wood-4            0.33
ChipMoisture-4       0.33
SteamFlow-4          0.33
Lower-HeatT-3        0.33
Upper-HeatT-3        0.33
ChipMass-4           0.33
WeakLiquorF          0.33
BlackFlow-2          0.33
WeakWashF            0.33
SteamHeatF-3         0.33
T-Top-Chips-4        0.33
SulphidityL-4       46.84
dtype: float64
```

As expected, the datapoint 'AAWhiteSt-4' even has 46% of data missing! It seems we only have 300 datapoints and presumably these missing values occur in different records our dataset will be decimated if we just drop all rows with missing values.

```
kamyr_df.shape
```

```
(301, 23)
```

```
kamyr_df.dropna().shape
```

```
(131, 23)
```

As we drop all rows with missing values, we are left with only 131 records. Whilst this might be good enough for some purposes, there are more viable options.

Perhaps we can first remove the column with the most missing values and then drop all remaining

```
kamyr_df.drop(columns=['AAWhiteSt-4 ','SulphidityL-4 ']).dropna().shape
```

```
(263, 21)
```

Significantly better, although we lost the information of 2 sensors we now have a complete dataset with 263 records. For purposes where those 2 sensors are irrelevant this is a viable option, keep in mind that this dataset is still 100% truthful, as we have not imputed any values.

Another option, where we retain all our records would be using the timely nature of our dataset, each record is a measurement with an interval of 1 hour. I have no knowledge of this dataset but one might make the assumption that the interval of 1 hour is taken as the state of the machine does not alter much in 1 hour. Therefore we could do what is called a forward fill, where we fill in the missing values with the same value of the sensor for the previous measurement.

This would solve nearly all nan values as there might be a problem where the first value is missing. This is shown below.

```
kamyr_df.fillna(method='ffill')['SulphidityL-4 ']
```

```
0        NaN
1      29.11
2      29.11
3      29.02
4      29.01
       ...
296    30.43
297    30.29
298    30.47
299    30.47
300    30.46
Name: SulphidityL-4 , Length: 301, dtype: float64
```

Although our dataset is not fully the truth, we can see that little to no changes occur in the sensor and using a forward fill is arguably the most suitable option.

3.2 Travel times

Another dataset from the same source contains a collection of recorded travel times and specific information about the travel itself as e.g.: the day of the week, where they were going, …

```
travel_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c2_data_preparation/data/travel-times.csv')
travel_df
```

```
          Date StartTime  DayOfWeek GoingTo  Distance  MaxSpeed  AvgSpeed  \
0     1/6/2012     16:37     Friday    Home     51.29     127.4      78.3
1     1/6/2012     08:20     Friday     GSK     51.63     130.3      81.8
2     1/4/2012     16:17  Wednesday    Home     51.27     127.4      82.0
3     1/4/2012     07:53  Wednesday     GSK     49.17     132.3      74.2
4     1/3/2012     18:57    Tuesday    Home     51.15     136.2      83.4
..         ...       ...        ...     ...       ...       ...       ...
200  7/18/2011     08:09     Monday     GSK     54.52     125.6      49.9
201  7/14/2011     08:03   Thursday     GSK     50.90     123.7      76.2
202  7/13/2011     17:08  Wednesday    Home     51.96     132.6      57.5
203  7/12/2011     17:51    Tuesday    Home     53.28     125.8      61.6
204  7/11/2011     16:56     Monday    Home     51.73     125.0      62.8

     AvgMovingSpeed FuelEconomy  TotalTime  MovingTime Take407All Comments
```

(continues on next page)

(continued from previous page)

```
0              84.8        NaN      39.3       36.3        No      NaN
1              88.9        NaN      37.9       34.9        No      NaN
2              85.8        NaN      37.5       35.9        No      NaN
3              82.9        NaN      39.8       35.6        No      NaN
4              88.1        NaN      36.8       34.8        No      NaN
..              ...        ...       ...        ...       ...      ...
200            82.4       7.89      65.5       39.7        No      NaN
201            95.1       7.89      40.1       32.1       Yes      NaN
202            76.7        NaN      54.2       40.6       Yes      NaN
203            87.6        NaN      51.9       36.5       Yes      NaN
204            92.5        NaN      49.5       33.6       Yes      NaN

[205 rows x 13 columns]
```

we have a total of 205 records and we can already see that the FuelEconomy column seems pretty bad, let's quantify that.

```
travel_df.isna().sum().divide(len(travel_df)).round(4)*100
```

```
Date                 0.00
StartTime            0.00
DayOfWeek            0.00
GoingTo              0.00
Distance             0.00
MaxSpeed             0.00
AvgSpeed             0.00
AvgMovingSpeed       0.00
FuelEconomy          8.29
TotalTime            0.00
MovingTime           0.00
Take407All           0.00
Comments            88.29
dtype: float64
```

In the end, it doesn't seem that bad, but there are comments and nearly none of them are filled in. Which in perspective is understandable. Let's see what the comments look like

```
travel_df[~travel_df.Comments.isna()].Comments
```

```
15                                  Put snow tires on
39                                         Heavy rain
49                                Huge traffic backup
50        Pumped tires up: check fuel economy improved?
52                                  Backed up at Bronte
54                                  Backed up at Bronte
60                                              Rainy
78                                   Rain, rain, rain
91                                   Rain, rain, rain
92          Accident: backup from Hamilton to 407 ramp
110                                           Raining
132                              Back to school traffic?
133                      Took 407 all the way (to McMaster)
150                               Heavy volume on Derry
156                            Start early to run a batch
158      Accident at 403/highway 6; detour along Dundas
165                                        Detour taken
166                                       Must be Friday
```

(continues on next page)

(continued from previous page)

```
172                              Medium amount of rain
174                                          New tires
182                               Turn around on Derry
184                                        Empty roads
187                             Police slowdown on 403
189                          Accident blocked 407 exit
Name: Comments, dtype: object
```

As you would expect, these comments are text based. Now imagine we would like to run some Natural Language Processing (NLP) on these, it would be a pain to perform string operations on it when it is riddled with missing values.

Here a simple example where we select all records containing the word 'rain', with no avail.

```
travel_df[travel_df.Comments.str.lower().str.contains('rain')]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_6376/1298831137.py in <module>
----> 1 travel_df[travel_df.Comments.str.lower().str.contains('rain')]

~/git/data-science-practical-approach/venv/lib/python3.8/site-packages/pandas/core/
↪frame.py in __getitem__(self, key)
   3446
   3447         # Do we have a (boolean) 1d indexer?
-> 3448         if com.is_bool_indexer(key):
   3449             return self._getitem_bool_array(key)
   3450

~/git/data-science-practical-approach/venv/lib/python3.8/site-packages/pandas/core/
↪common.py in is_bool_indexer(key)
    137                 # Don't raise on e.g. ["A", "B", np.nan], see
    138                 #  test_loc_getitem_list_of_labels_categoricalindex_with_
↪na
--> 139                 raise ValueError(na_msg)
    140             return False
    141         return True

ValueError: Cannot mask with non-boolean array containing NA / NaN values
```

The last line of the python error traceback gives us the reason it failed, because there were NaN values present.

Luckily the string variable has more or less it's on 'null' value, being an empty string, this way these operations are still possible, most of the comments will just contain nothing.

```
travel_df.Comments = travel_df.Comments.fillna('')
```

```
travel_df[travel_df.Comments.str.lower().str.contains('rain')]
```

```
           Date StartTime  DayOfWeek GoingTo  Distance  MaxSpeed  AvgSpeed  \
39   11/29/2011     07:23    Tuesday     GSK     51.74     112.2      55.3
60    11/9/2011     16:15  Wednesday    Home     51.28     121.4      65.9
78   10/25/2011     17:24    Tuesday    Home     52.87     123.5      65.1
91   10/12/2011     17:47  Wednesday    Home     51.40     114.4      59.7
110   9/27/2011     07:36    Tuesday     GSK     50.65     128.1      86.3
172    8/9/2011     08:15    Tuesday     GSK     49.08     134.8      60.5
```

(continues on next page)

(continued from previous page)

```
     AvgMovingSpeed FuelEconomy  TotalTime  MovingTime Take407All  \
39             61.0         NaN       56.2        50.9         No
60             71.8        9.35       46.7        42.1         No
78             72.4        8.97       48.7        43.8         No
91             65.8        8.75       51.7        46.9         No
110            88.6        8.31       35.2        34.3        Yes
172            67.2        8.54       48.7        43.8         No

                   Comments
39               Heavy rain
60                    Rainy
78         Rain, rain, rain
91         Rain, rain, rain
110                 Raining
172  Medium amount of rain
```

Fixed! now we can use the comments for analysis.

We still have to fix the FuelEconomy, let us take a look at the non NaN values

```
travel_df[~travel_df.FuelEconomy.isna()]
```

```
           Date StartTime  DayOfWeek GoingTo  Distance  MaxSpeed  AvgSpeed  \
6      1/2/2012     17:31     Monday    Home     51.37     123.2      82.9
7      1/2/2012     07:34     Monday     GSK     49.01     128.3      77.5
8    12/23/2011     08:01     Friday     GSK     52.91     130.3      80.9
9    12/22/2011     17:19   Thursday    Home     51.17     122.3      70.6
10   12/22/2011     08:16   Thursday     GSK     49.15     129.4      74.0
..          ...       ...        ...     ...       ...       ...       ...
197   7/20/2011     08:24  Wednesday     GSK     48.50     125.8      75.7
198   7/19/2011     17:17    Tuesday    Home     51.16     126.7      92.2
199   7/19/2011     08:11    Tuesday     GSK     50.96     124.3      82.3
200   7/18/2011     08:09     Monday     GSK     54.52     125.6      49.9
201   7/14/2011     08:03   Thursday     GSK     50.90     123.7      76.2

     AvgMovingSpeed FuelEconomy  TotalTime  MovingTime Take407All Comments
6              87.3           -       37.2        35.3         No
7              85.9           -       37.9        34.3         No
8              88.3        8.89       39.3        36.0         No
9              78.1        8.89       43.5        39.3         No
10             81.4        8.89       39.8        36.2         No
..              ...         ...        ...         ...        ...      ...
197            87.3        7.89       38.5        33.3        Yes
198           102.6        7.89       33.3        29.9        Yes
199            96.4        7.89       37.2        31.7        Yes
200            82.4        7.89       65.5        39.7         No
201            95.1        7.89       40.1        32.1        Yes

[188 rows x 13 columns]
```

It seems that aside NaN values there are also other intruders, a quick check on the data type (Dtype) reveils it is not recognised as a number!

```
travel_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            205 non-null    object
 1   StartTime       205 non-null    object
 2   DayOfWeek       205 non-null    object
 3   GoingTo         205 non-null    object
 4   Distance        205 non-null    float64
 5   MaxSpeed        205 non-null    float64
 6   AvgSpeed        205 non-null    float64
 7   AvgMovingSpeed  205 non-null    float64
 8   FuelEconomy     188 non-null    object
 9   TotalTime       205 non-null    float64
 10  MovingTime      205 non-null    float64
 11  Take407All      205 non-null    object
 12  Comments        205 non-null    object
dtypes: float64(6), object(7)
memory usage: 20.9+ KB
```

The column is noted as an object or string type, meaning that these numbers are given as ‘9.24’ instead of 9.24 and numerical operations are not possible. We can cast them to numeric but have to warn pandas to coerce errors, meaning errors will be converted to NaN values. Later we’ll handle the NaN’s.

```
travel_df.FuelEconomy = pd.to_numeric(travel_df.FuelEconomy, errors='coerce')
travel_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            205 non-null    object
 1   StartTime       205 non-null    object
 2   DayOfWeek       205 non-null    object
 3   GoingTo         205 non-null    object
 4   Distance        205 non-null    float64
 5   MaxSpeed        205 non-null    float64
 6   AvgSpeed        205 non-null    float64
 7   AvgMovingSpeed  205 non-null    float64
 8   FuelEconomy     186 non-null    float64
 9   TotalTime       205 non-null    float64
 10  MovingTime      205 non-null    float64
 11  Take407All      205 non-null    object
 12  Comments        205 non-null    object
dtypes: float64(7), object(6)
memory usage: 20.9+ KB
```

Wonderful, now the column is numerical and we can see 2 more missing values have popped up! We could easily drop these 19 records and have a complete dataset.

```
travel_df.dropna()
```

```
           Date StartTime  DayOfWeek GoingTo  Distance  MaxSpeed  AvgSpeed  \
8    12/23/2011     08:01     Friday     GSK     52.91     130.3      80.9
```

(continues on next page)

(continued from previous page)

```
9    12/22/2011     17:19   Thursday    Home      51.17     122.3      70.6
10   12/22/2011     08:16   Thursday     GSK      49.15     129.4      74.0
11   12/21/2011     07:45  Wednesday     GSK      51.77     124.8      71.7
12   12/20/2011     16:05    Tuesday    Home      51.45     130.1      75.2
..          ...       ...        ...     ...        ...       ...       ...
197   7/20/2011     08:24  Wednesday     GSK      48.50     125.8      75.7
198   7/19/2011     17:17    Tuesday    Home      51.16     126.7      92.2
199   7/19/2011     08:11    Tuesday     GSK      50.96     124.3      82.3
200   7/18/2011     08:09     Monday     GSK      54.52     125.6      49.9
201   7/14/2011     08:03   Thursday     GSK      50.90     123.7      76.2

     AvgMovingSpeed  FuelEconomy  TotalTime  MovingTime Take407All Comments
8              88.3         8.89       39.3        36.0         No
9              78.1         8.89       43.5        39.3         No
10             81.4         8.89       39.8        36.2         No
11             78.9         8.89       43.3        39.4         No
12             82.7         8.89       41.1        37.3         No
..              ...          ...        ...         ...        ...      ...
197            87.3         7.89       38.5        33.3        Yes
198           102.6         7.89       33.3        29.9        Yes
199            96.4         7.89       37.2        31.7        Yes
200            82.4         7.89       65.5        39.7         No
201            95.1         7.89       40.1        32.1        Yes

[186 rows x 13 columns]
```

However im leaving them as an excercise for you to apply a technique we will see in the next part

3.3 Material properties

Another dataset from the same source contains the material properties from 30 samples, this time there is not timestamp as the samples are not related in time with each other.

```
material_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c2_data_preparation/data/raw-material-properties.csv')
material_df
```

```
    Sample  size1  size2  size3  density1  density2  density3
0   X12558  0.696   2.69   6.38      41.8     17.18      3.90
1   X14728  0.636   2.30   5.14      38.1     12.73      3.89
2   X15468  0.841   2.85   5.20      37.6     13.58      3.98
3   X21364  0.609   2.13   4.62      34.2     11.12      4.02
4   X23671  0.684   2.16   4.87      36.4     12.24      3.92
5   X24055  0.762   2.81   6.36      38.1     13.28      3.89
6   X24905  0.552   2.34   5.03      41.3     16.71      3.86
7   X25917  0.501   2.17   5.09       NaN       NaN       NaN
8   X27871  0.619   2.11   5.13       NaN       NaN       NaN
9   X28690  0.610   2.10   4.18      35.0     12.15      3.86
10  X31385  0.532   2.09   4.93       NaN       NaN       NaN
11  X31813  0.738   2.29   5.47       NaN       NaN       NaN
12  X32807  0.779   2.62   5.59       NaN       NaN       NaN
13  X33943  0.537   2.23   5.41      35.2     11.34      3.99
14  X35035  0.702   2.05   5.10      34.2     10.54      4.02
15  X39223  0.768   2.51   5.09      34.9     12.55      3.90
```

(continues on next page)

(continued from previous page)

```
16  X40503  0.714   2.56   6.03       35.6      12.20       4.02
17  X41400  0.621   2.42   5.10       38.7      14.27       3.98
18  X42988  0.726   2.11   4.69       37.1      13.14       3.98
19  X44749  0.698   2.36   5.40       36.6      12.16       4.01
20  X45295    NaN    NaN    NaN       38.1      13.34       3.89
21  X46965  0.759   2.47   4.83       38.7      14.83       3.89
22  X49666  0.535   2.13   5.23        NaN        NaN        NaN
23  X50678  0.716   2.29   5.45       37.3      13.70       3.92
24  X52894  0.635   2.08   4.94        NaN        NaN        NaN
25  X53925  0.598   2.12   4.69       37.9      13.45       3.78
26  X54254  0.700   2.47   5.22       38.8      14.72       3.92
27  X54272  0.957   2.96   7.37       36.2      13.38       4.20
28  X54394  0.759   2.66   5.36       35.2      12.19       3.98
29  X55408  0.661   2.10   4.27        NaN        NaN        NaN
30  X56952  0.646   2.38   4.51       40.1      15.68       3.86
31  X57095  0.662   2.34   4.71       35.0      12.37       3.90
32  X57128  0.749   2.43   5.16       37.3      13.04       3.92
33  X61870  0.598   2.21   4.90        NaN        NaN        NaN
34  X61888  0.619   2.59   5.81        NaN        NaN        NaN
35  X72736  0.693   2.05   5.02       39.6      15.55       3.94
```

let us quantify the amount of missing data

```
material_df.isna().sum().divide(len(material_df)).round(4)*100
```

```
Sample        0.00
size1         2.78
size2         2.78
size3         2.78
density1     27.78
density2     27.78
density3     27.78
dtype: float64
```

Unfortunately that is a lot of missing data, covered in all records, dropping here seems almost impossible if we want to keep a healthy amount of records.

Here it would be wise to go for a more elaborate method of imputation, I opted for the K-nearest neighbours method, which looks at the K most similar records in the dataset to make an educated guess on what the missing value could be, this because we can assume that records with similar data are also similar over all the properties (columns).

Im using the sklearn library for this, which has more imputation techniques such as MICE. More info can be found here

```
from sklearn.impute import KNNImputer
```

im creating an imputer object and specify that i want to use the 5 most similar records and weigh them by distance from the to imputed record, meaning closer neighbours are more important.

```
imputer = KNNImputer(n_neighbors=5, weights="distance")
```

As the imputer only takes numerical values I had to do some pandas magic and drop the first column, which I then added again. The result is a fully filled dataset, you can recognise the new values as they are not rounded.

```
pd.DataFrame(
    imputer.fit_transform(material_df.drop(columns=['Sample'])),
    columns=material_df.columns.drop('Sample')
)
```

	size1	size2	size3	density1	density2	density3
0	0.696000	2.690000	6.380000	41.800000	17.180000	3.900000
1	0.636000	2.300000	5.140000	38.100000	12.730000	3.890000
2	0.841000	2.850000	5.200000	37.600000	13.580000	3.980000
3	0.609000	2.130000	4.620000	34.200000	11.120000	4.020000
4	0.684000	2.160000	4.870000	36.400000	12.240000	3.920000
5	0.762000	2.810000	6.360000	38.100000	13.280000	3.890000
6	0.552000	2.340000	5.030000	41.300000	16.710000	3.860000
7	0.501000	2.170000	5.090000	38.495282	14.029399	3.931180
8	0.619000	2.110000	5.130000	37.405275	13.157346	3.943667
9	0.610000	2.100000	4.180000	35.000000	12.150000	3.860000
10	0.532000	2.090000	4.930000	37.811132	13.646072	3.908364
11	0.738000	2.290000	5.470000	37.088833	13.255412	3.941654
12	0.779000	2.620000	5.590000	36.540567	12.889902	3.970973
13	0.537000	2.230000	5.410000	35.200000	11.340000	3.990000
14	0.702000	2.050000	5.100000	34.200000	10.540000	4.020000
15	0.768000	2.510000	5.090000	34.900000	12.550000	3.900000
16	0.714000	2.560000	6.030000	35.600000	12.200000	4.020000
17	0.621000	2.420000	5.100000	38.700000	14.270000	3.980000
18	0.726000	2.110000	4.690000	37.100000	13.140000	3.980000
19	0.698000	2.360000	5.400000	36.600000	12.160000	4.010000
20	0.733097	2.653959	5.881504	38.100000	13.340000	3.890000
21	0.759000	2.470000	4.830000	38.700000	14.830000	3.890000
22	0.535000	2.130000	5.230000	37.391815	13.089536	3.944335
23	0.716000	2.290000	5.450000	37.300000	13.700000	3.920000
24	0.635000	2.080000	4.940000	37.254724	13.206262	3.933904
25	0.598000	2.120000	4.690000	37.900000	13.450000	3.780000
26	0.700000	2.470000	5.220000	38.800000	14.720000	3.920000
27	0.957000	2.960000	7.370000	36.200000	13.380000	4.200000
28	0.759000	2.660000	5.360000	35.200000	12.190000	3.980000
29	0.661000	2.100000	4.270000	36.172345	12.755632	3.887375
30	0.646000	2.380000	4.510000	40.100000	15.680000	3.860000
31	0.662000	2.340000	4.710000	35.000000	12.370000	3.900000
32	0.749000	2.430000	5.160000	37.300000	13.040000	3.920000
33	0.598000	2.210000	4.900000	37.865882	13.826029	3.887021
34	0.619000	2.590000	5.810000	35.932339	12.318210	3.989911
35	0.693000	2.050000	5.020000	39.600000	15.550000	3.940000

This concludes the part of missing values, perhaps you can try yourself and impute the missing values for the FuelEconomy using the SimpleImputer or even the IterativeImputer.

CHAPTER

FOUR

CONCATENATION AND DEDUPLICATION

In this notebook we are going to investigate the concepts of stitching data files (concatenation) and verifying the integrity of our data concercing duplicates

4.1 Concatenation

When dealing with large amounts of data, fractioning is often the only solution. Not only does this tidy up your data space, but it also benefits computation. Aside from that, appending new data to your data lake is independent of the historical data. However if you want to perform historical analysis this means you will need to perform additional operations.

In this notebook we have a setup of a very small data lake containing daily minimal temperatures. If you would look closely in the url you would see the following structure.

> data/temperature/australia/melbourne/1981.csv

This is a straight-forward but perfect example on how fragmentation works, in our data lake we have: temperatures data fractioned by country, city and year. As we are working with daily temperatures further fractioning would not be interesting, but you could fraction e.g. per month.

In the cells below, we read our both 1981 and 1982 data and concatenate them using python.

```
import pandas as pd
```

```
melbourne_1981_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-
↪science-practical-approach/main/src/c2_data_preparation/data/temperatures/australia/
↪melbourne/1981.csv')
```

```
melbourne_1982_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-
↪science-practical-approach/main/src/c2_data_preparation/data/temperatures/australia/
↪melbourne/1982.csv')
```

```
df = pd.concat(
    [
        melbourne_1981_df,
        melbourne_1982_df,
    ]
)
```

```
df
```

```
            Date  Temp
0    1981-01-01  20.7
1    1981-01-02  17.9
2    1981-01-03  18.8
3    1981-01-04  14.6
4    1981-01-05  15.8
..          ...   ...
360  1982-12-27  15.3
361  1982-12-28  16.3
362  1982-12-29  15.8
363  1982-12-30  17.7
364  1982-12-31  16.3

[730 rows x 2 columns]
```

And there you have it! we now have a dataframe containing both data from 1981 as 1982. Can you figure out what I calculated in the next cell? Do you think there might be a more 'clean' solution?

```
df[df.Date.str[5:7]== '01'].Temp.mean()
```

```
17.140322580645158
```

As an exercise I would ask you now to create a small python script that given a begin and end year (between 1981 and 1990) can automatically concatenate all the necessary data

```
for i in range(1982,1987):
    print(i)
```

```
1982
1983
1984
1985
1986
```

4.2 Deduplication

Another important aspect of data cleaning is the removal of duplicates. Here we fragment of a dataset from activity on a popular games platform. We can see which user has either bought or played specific games and how often. Unfortunately for some reason, entries might have duplicates which we have to deal with as otherwise users might have e.g. bought a game twice.

```
df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-practical-
↪approach/main/src/c2_data_preparation/data/steam.csv')
df
```

```
        user_id                                   game    action  freq
0      11373749                Sid Meier's Civilization IV  purchase   1.0
1      11373749                Sid Meier's Civilization IV      play   0.1
2      11373749                Sid Meier's Civilization IV  purchase   1.0
3      11373749  Sid Meier's Civilization IV Beyond the Sword  purchase   1.0
4      11373749  Sid Meier's Civilization IV Beyond the Sword  purchase   1.0
...         ...                                        ...       ...   ...
1834  112845094                                     Arma 2  purchase   1.0
```

(continues on next page)

(continued from previous page)

```
1835  112845094                  Grand Theft Auto San Andreas  purchase   1.0
1836  112845094                    Grand Theft Auto Vice City  purchase   1.0
1837  112845094                    Grand Theft Auto Vice City  purchase   1.0
1838  112845094                          Grand Theft Auto III  purchase   1.0

[1839 rows x 4 columns]
```

We have a dataframe with 1839 interactions, you can see that the freq either notes the amount they bought (which always 1 as there is not use in buying it more) or the amount in hours they played.

Let us straightforward ask pandas to remove all rows that have an exact duplicate

```
df.drop_duplicates()
```

```
        user_id                                          game    action  freq
0      11373749                   Sid Meier's Civilization IV  purchase   1.0
1      11373749                   Sid Meier's Civilization IV      play   0.1
3      11373749  Sid Meier's Civilization IV Beyond the Sword  purchase   1.0
5      11373749          Sid Meier's Civilization IV Warlords  purchase   1.0
7      56038151                       Tom Clancy's H.A.W.X. 2  purchase   1.0
...         ...                                           ...       ...   ...
1831  112845094                  Grand Theft Auto San Andreas  purchase   1.0
1832  112845094                  Grand Theft Auto San Andreas      play   0.2
1833  112845094                          Grand Theft Auto III  purchase   1.0
1834  112845094                                        Arma 2  purchase   1.0
1836  112845094                    Grand Theft Auto Vice City  purchase   1.0

[1132 rows x 4 columns]
```

Alright! this seemed to have dropped 707 rows from our dataset, but we would like to know more about those. Let's ask which rows the algorithm has dropped:

```
df[df.duplicated()]
```

```
        user_id                                          game    action  freq
2      11373749                   Sid Meier's Civilization IV  purchase   1.0
4      11373749  Sid Meier's Civilization IV Beyond the Sword  purchase   1.0
6      11373749          Sid Meier's Civilization IV Warlords  purchase   1.0
10     56038151                  Grand Theft Auto San Andreas  purchase   1.0
12     56038151                    Grand Theft Auto Vice City  purchase   1.0
...         ...                                           ...       ...   ...
1827   39146470          Sid Meier's Civilization IV Warlords  purchase   1.0
1830   48666962                                      Crysis 2  purchase   1.0
1835  112845094                  Grand Theft Auto San Andreas  purchase   1.0
1837  112845094                    Grand Theft Auto Vice City  purchase   1.0
1838  112845094                          Grand Theft Auto III  purchase   1.0

[707 rows x 4 columns]
```

Here we can see the duplicates, no particular pattern seems to be present, we could just for curiosity count the games that are duplicated

```
df[df.duplicated()].game.value_counts()
```

```
Grand Theft Auto San Andreas                    172
Grand Theft Auto Vice City                      103
```

(continues on next page)

(continued from previous page)

```
Sid Meier's Civilization IV                     98
Grand Theft Auto III                            90
Sid Meier's Civilization IV Beyond the Sword    80
Sid Meier's Civilization IV Warlords            79
Sid Meier's Civilization IV Colonization        75
Crysis 2                                         7
Arma 2                                           1
Tom Clancy's H.A.W.X. 2                          1
TERA                                             1
Name: game, dtype: int64
```

It seems there are some games which are very prone to being duplicated, at this point we could go and ask the IT department why these games are acting weird.

Another thing im interested about is the perspective of a single gamer, here we took a single user_id and printed all his games

```
df[df.user_id == 11373749]
```

```
    user_id                                         game    action  freq
0  11373749                  Sid Meier's Civilization IV  purchase   1.0
1  11373749                  Sid Meier's Civilization IV      play   0.1
2  11373749                  Sid Meier's Civilization IV  purchase   1.0
3  11373749  Sid Meier's Civilization IV Beyond the Sword  purchase   1.0
4  11373749  Sid Meier's Civilization IV Beyond the Sword  purchase   1.0
5  11373749          Sid Meier's Civilization IV Warlords  purchase   1.0
6  11373749          Sid Meier's Civilization IV Warlords  purchase   1.0
```

Ah, you can see all of his three games are somehow duplicated in purchase, also it seems he only played one of them for only 0.1 hours. Looks like he fell to the bait of a tempting summer sale but didn't realise he had no time to actually play it.

Another thing I would like to mention here is that this dataset would make a fine recommender system as it contains user ids and hours played. Add game metadata (description) and reviews to the mix and your data preparation is done!

We can remove all duplicates now by overwriting our dataframe

```
df = df.drop_duplicates()
```

One thing still bothers me, as hours played can change over time it might be that different snapshots have produced different values, therefore more duplicates might be present with different hours_played.

Time to investigate this by using a subset of columns in the drop_duplicates algorithm

```
df.drop_duplicates(subset=['user_id', 'game', 'action'])
```

```
        user_id                                         game    action  freq
0      11373749                  Sid Meier's Civilization IV  purchase   1.0
1      11373749                  Sid Meier's Civilization IV      play   0.1
3      11373749  Sid Meier's Civilization IV Beyond the Sword  purchase   1.0
5      11373749          Sid Meier's Civilization IV Warlords  purchase   1.0
7      56038151                      Tom Clancy's H.A.W.X. 2  purchase   1.0
...         ...                                          ...       ...   ...
1831  112845094                   Grand Theft Auto San Andreas  purchase   1.0
1832  112845094                   Grand Theft Auto San Andreas      play   0.2
1833  112845094                           Grand Theft Auto III  purchase   1.0
1834  112845094                                         Arma 2  purchase   1.0
```

(continues on next page)

(continued from previous page)

```
1836  112845094                    Grand Theft Auto Vice City  purchase   1.0

[1120 rows x 4 columns]
```

Seems we have shaved off another 12 records, so our intuition was right, again lets see which the duplicates are:

```
df[df.duplicated(subset=['user_id', 'game', 'action'])]
```

```
        user_id                                          game action  freq
118   118664413                  Grand Theft Auto San Andreas   play   0.2
458    50769696                  Grand Theft Auto San Andreas   play   3.1
521    71411882                          Grand Theft Auto III   play   0.2
607    33865373                   Sid Meier's Civilization IV   play   2.0
898    71510748                  Grand Theft Auto San Andreas   play   0.2
908    28472068                    Grand Theft Auto Vice City   play   0.4
910    28472068                  Grand Theft Auto San Andreas   play   0.2
912    28472068                          Grand Theft Auto III   play   0.1
1506   59925638                       Tom Clancy's H.A.W.X. 2   play   0.3
1553  148362155                  Grand Theft Auto San Andreas   play  12.5
1709  176261926  Sid Meier's Civilization IV Beyond the Sword   play   0.4
1711  176261926                   Sid Meier's Civilization IV   play   0.2
```

As expected the duplicates are all in the ‘play’ action, to complete our view we extract the data of a single user

```
df[df.user_id==118664413]
```

```
       user_id                          game    action  freq
115  118664413  Grand Theft Auto San Andreas  purchase   1.0
116  118664413  Grand Theft Auto San Andreas      play   1.9
118  118664413  Grand Theft Auto San Andreas      play   0.2
```

It looks like we have a problem now, we know these are duplicates and should be removed, but which one? Personally I would argue here that we keep the highest value, as it is impossible to ‘unplay’ hours on the game. I will leave this as an exercise for you, but the solution is pretty tricky so i’ll give a hint:

The algorithm always keeps the first record in case of duplicates, so you could sort the rows making sure the higher value is always encountered first, good luck!

CHAPTER FIVE

OUTLIERS AND VALIDITY

When preparing data we have to be cautious with the accuracy of our set. Outliers and invalid data points are difficult to detect but should be handled with caution.

we start out by importing our most important library.

```
import pandas as pd
```

5.1 Silicon wafer thickness

Our first dataset contains information about the production of silicon wafers, each wafers thickness is measure on 9 different spots. More information on the dataset can be found here.

```
wafer_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c2_data_preparation/data/silicon-wafer-thickness.csv')
wafer_df.head()
```

```
      G1     G2     G3     G4     G5     G6     G7     G8     G9
0  0.175  0.188 -0.159  0.095  0.374 -0.238 -0.800  0.158 -0.211
1  0.102  0.075  0.141  0.180  0.138 -0.057 -0.075  0.072  0.072
2  0.607  0.711  0.879  0.765  0.592  0.187  0.431  0.345  0.187
3  0.774  0.823  0.619  0.370  0.725  0.439 -0.025 -0.259  0.496
4  0.504  0.644  0.845  0.681  0.502  0.151  0.404  0.296  0.260
```

we would like to investigate the distribution of measurements here, as we are early in this course using visualisation techniques would be too soon. This does not mean we can't use simple mathematics, introducing the InterQuartile Range. A reason for using IQR over standard deviation is that with IQR we do not assume a normal distribution. The IQR calculates the range between the bottom 'quart' or 25% and the top 25%, giving us an indication of the spread of our results, we calculate this IQR for each of the 9 measurements independently. For more info about IQR you can visit wikipedia.

```
iqr = wafer_df.quantile(0.75)-wafer_df.quantile(0.25)
iqr
```

```
G1    0.54425
G2    0.61000
G3    0.54075
G4    0.52475
G5    0.61175
G6    0.86750
G7    0.76175
```

(continues on next page)

(continued from previous page)

```
G8    0.87225
G9    0.86300
dtype: float64
```

you can see that the IQR spread for each measurement lays between 0.5 and 1 unit indicating that the 9 measurements of the wafer have a similar spread. With these IQR's we could calculate for each point relative to the spread of the measurement how far it is from the median.

```
relative_spread_df = (wafer_df-wafer_df.median())/iqr
relative_spread_df.head()
```

```
         G1        G2        G3        G4        G5        G6        G7  \
0 -0.011024 -0.077869 -0.819233 -0.367794  0.176543 -0.352738 -1.029865
1 -0.145154 -0.263115 -0.264448 -0.205812 -0.209236 -0.144092 -0.078110
2  0.782729  0.779508  1.100324  0.909004  0.532897  0.137176  0.586150
3  1.089573  0.963115  0.619510  0.156265  0.750306  0.427666 -0.012471
4  0.593477  0.669672  1.037448  0.748928  0.385779  0.095677  0.550706

         G8        G9
0 -0.130696 -0.254925
1 -0.229292  0.073001
2  0.083692  0.206257
3 -0.608770  0.564311
4  0.027515  0.290846
```

You can now see that some points are close to the median, whilst others are much higher, both positive as negative. By defining a threshold, we quantify what deviation has to be there to flag a reading as an outlier. The high outliers are seperated, note that only a single measurement of the 9 can trigger and render the total measurement as an outlier. Yet judging from the setup where we would want to find wafers with varying thickness that approach is desirable.

```
relative_spread_df[(relative_spread_df>2).any(axis='columns')]
```

```
            G1         G2         G3         G4         G5        G6  \
8     2.232430   2.009016   1.956542   1.589328   1.843890  1.544669
38   12.891135  12.827049  12.832178  13.913292  11.429506  9.500865
39    3.691318   3.981148   3.774387   4.081944   3.248059  3.729107
61    2.010106   2.153279   1.987980   1.863745   1.858602  1.274928
110   3.678457   2.841803   3.204808   3.180562   2.669391  0.518732
112   2.361047   2.086066   2.363384   2.107670   1.925623  1.238040
117   1.475425   1.043443   2.154415   2.582182   0.653862  1.823631
120   1.791456   1.484426   2.583449   1.440686   2.085819  0.990202
121   1.791456   1.484426   2.583449   1.440686   2.085819  0.990202
152   2.610932   2.102459   2.387425   2.549786   2.169187  1.730259
154  -0.529169  -0.538525  -0.404993  -0.331586  -0.552513  4.565994

            G7        G8        G9
8     1.233344  0.419604  1.582851
38   10.305875  9.927200  9.055620
39    3.304890  3.846374  3.149479
61    1.237283  0.825451  0.955968
110   0.700361  0.176555  0.727694
112   1.766328  0.890800  1.377752
117   1.581227  0.857552  1.188876
120   1.782081  1.034107  1.822711
121   1.782081  1.034107  1.822711
```

(continues on next page)

(continued from previous page)

```
152   2.241549  1.713958  1.592121
154  -0.051854 -0.382918 -0.536501
```

seems we have a few high outliers, you can clearly see the measurements are mostly all across the board high, but in some cases (e.g. id 154) only one measurement was an outlier. We can do the same for the low outliers.

```
relative_spread_df[(relative_spread_df<-2).any(axis='columns')]
```

```
           G1        G2        G3        G4        G5        G6        G7  \
54  -1.550758 -1.525410 -1.843736 -2.082897 -1.659174 -1.203458 -1.184772
56  -1.732660 -1.510656 -2.121128 -2.122916 -1.781774 -1.521614 -1.909419
59  -1.971520 -1.310656 -2.328248 -1.175798 -2.067838 -0.915274 -1.783394
64  -1.234727 -1.361475 -0.736015 -1.055741 -2.224765 -0.839193 -0.679357
65  -2.226918 -1.194262 -2.117429 -2.161029 -2.043318 -0.190202 -1.004923
102 -2.484153 -2.330328 -1.568192 -2.808957 -1.945239 -1.340634 -0.846078

           G8        G9
54  -1.650903 -1.245655
56  -1.782746 -1.159907
59  -1.304672 -1.514484
64  -0.865578 -0.663963
65  -0.270565 -0.794902
102 -1.691029 -0.887601
```

For a simple mathematical equation these result look promising, yet it can always be more sophisticated. Not going to deep into the subject we could perform some Machine Learning, using a unsupervised method. Here we use the sklearn library which contains the Isolation forest algorithm. More info about the algorithm here.

```
from sklearn.ensemble import IsolationForest
```

We first create the classifier and train (fit) it with the generic wafer data. Then for each record of the wafer data we make a prediction, if it thinks its an outlier, we keep them

```
clf = IsolationForest(random_state=0).fit(wafer_df)
wafer_df[clf.predict(wafer_df)==-1]
```

```
        G1     G2     G3     G4     G5     G6     G7     G8     G9
8    1.396  1.461  1.342  1.122  1.394  1.408  0.924  0.638  1.375
20  -0.558 -0.705 -0.526 -0.412 -0.753 -0.998 -0.270  0.598 -1.416
38   7.197  8.060  7.223  7.589  7.258  8.310  7.835  8.931  7.824
39   2.190  2.664  2.325  2.430  2.253  3.303  2.502  3.627  2.727
54  -0.663 -0.695 -0.713 -0.805 -0.749 -0.976 -0.918 -1.168 -1.066
56  -0.762 -0.686 -0.863 -0.826 -0.824 -1.252 -1.470 -1.283 -0.992
59  -0.892 -0.564 -0.975 -0.329 -0.999 -0.726 -1.374 -0.866 -1.298
61   1.275  1.549  1.359  1.266  1.403  1.174  0.927  0.992  0.834
65  -1.031 -0.493 -0.861 -0.846 -0.984 -0.097 -0.781  0.036 -0.677
102 -1.171 -1.186 -0.564 -1.186 -0.924 -1.095 -0.660 -1.203 -0.757
106 -0.659 -0.451 -0.692 -0.708 -0.595 -0.726 -1.031 -0.877 -1.080
110  2.183  1.969  2.017  1.957  1.899  0.518  0.518  0.426  0.637
112  1.466  1.508  1.562  1.394  1.444  1.142  1.330  1.049  1.198
117  0.984  0.872  1.449  1.643  0.666  1.650  1.189  1.020  1.035
120  1.156  1.141  1.681  1.044  1.542  0.927  1.342  1.174  1.582
121  1.156  1.141  1.681  1.044  1.542  0.927  1.342  1.174  1.582
152  1.602  1.518  1.575  1.626  1.593  1.569  1.692  1.767  1.383
```

Comparing the results with our IQR approach we see a lot of similarities, here the id 154 record did not show up as we already realised this was perhaps not a strong enough outlier. You could enhance our IQR technique by checking the amount of measurements that are above the threshold and respond accordingly, I will leave you a little hint.

```
(relative_spread_df>2).sum()
```

```
G1    7
G2    7
G3    8
G4    6
G5    6
G6    3
G7    3
G8    2
G9    2
dtype: int64
```

5.2 Distillation column

As an exercise you can try the same technique to this dataset and see what you would find, good luck! Be mindful that you do not incorporate the date as a variable in your outlier algorithm.

```
distil_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c2_data_preparation/data/distillation-tower.csv')
distil_df
```

```
           Date     Temp1    FlowC1     Temp2    TempC1     Temp3    TempC2  \
0    2000-08-21  139.9857  432.0636  377.8119  100.2204  492.1353  490.1459
1    2000-08-23  131.0470  487.4029  371.3060  100.2297  482.2100  480.3128
2    2000-08-26  118.2666  437.3516  378.4483  100.3084  488.7266  487.0040
3    2000-08-29  118.1769  481.8314  378.0028   95.5766  493.1481  491.1137
4    2000-08-30  120.7891  412.6471  377.8871   92.9052  490.2486  488.6641
..          ...       ...       ...       ...       ...       ...       ...
248  2003-01-26  130.8138  212.6385  341.5964  121.4354  468.3401  467.0299
249  2003-01-28  128.9673  225.1412  349.8965  118.8604  479.7665  478.4652
250  2003-01-31  130.5328  223.5965  345.9366  120.4027  474.5378  473.1145
251  2003-02-03  128.5248  213.5613  343.4950  119.6989  469.3802  467.9954
252  2003-02-04  131.0491  217.4117  346.1960  119.0825  474.6599  473.0381

       TempC3     Temp4  PressureC1  ...    Temp10  FlowC3   FlowC4   Temp11  \
0    180.5578  187.4331    215.0627  ...  513.9653  8.6279  10.5988  30.8983
1    172.6575  179.5089    205.0999  ...  504.5145  8.7662  10.7560  31.9099
2    165.9400  172.9262    205.0304  ...  508.9997  8.5319  10.5737  29.9165
3    167.2085  174.2338    205.2561  ...  514.1794  8.6260  10.6695  30.6229
4    167.0326  173.9681    205.0883  ...  511.0948  8.5939  10.4922  29.4977
..        ...       ...         ...  ...       ...     ...      ...      ...
248  174.7639  180.7649    229.7393  ...  479.0290  5.5590   6.4470  16.4131
249  176.2176  182.3646    230.5049  ...  491.2362  5.6342   6.4360  17.2385
250  176.3310  182.2578    230.6638  ...  485.8786  5.4810   6.3575  16.9866
251  174.6435  180.5093    230.5226  ...  480.2879  5.4727   6.4175  16.6778
252  177.1088  183.1810    225.6420  ...  486.0253  5.4597   6.3291  16.8766

       Temp12  InvTemp1  InvTemp2  InvTemp3  InvPressure1  VapourPressure
0    489.9900    2.0409    2.6468    2.1681        4.3524         32.5026
1    480.2888    2.0821    2.6932    2.2207        4.5497         34.8598
```

(continues on next page)

(continued from previous page)

```
2    486.6190    2.0550    2.6424    2.1796        4.5511         32.1666
3    491.1304    2.0361    2.6455    2.1620        4.5464         30.4064
4    487.6475    2.0507    2.6463    2.1704        4.5499         30.9238
..        ...       ...       ...       ...           ...             ...
248  466.3347    2.1444    2.9274    2.2127        4.0911         38.8507
249  477.8816    2.0926    2.8580    2.1620        4.0783         34.2653
250  472.3176    2.1172    2.8907    2.1855        4.0756         36.5717
251  467.0001    2.1413    2.9113    2.2090        4.0780         38.1054
252  472.2701    2.1174    2.8885    2.1844        4.1608         35.6298

[253 rows x 28 columns]
```

CHAPTER

SIX

STRING OPERATIONS

CHAPTER

SEVEN

DATETIME OPERATIONS

When our dataset contains time-related data, datetime operations are a great asset to our data science toolkit. For this exercise we obtain a public covid dataset containing A LOT of information on infection cases, deaths, tests and vaccinations.

Let's start by importing the data, as the dataset is about 60MB at the time of writing, this might take some time. Perhaps you could think of a method to make this more efficient, do we always need all of the data?

More info about the data can be found here

```
import pandas as pd
```

```
covid_df = pd.read_csv('https://raw.githubusercontent.com/owid/covid-19-data/master/
↪public/data/owid-covid-data.csv', on_bad_lines='skip')
covid_df.head()
```

```
  iso_code continent     location        date  total_cases  new_cases  \
0      AFG      Asia  Afghanistan  2020-02-24          5.0        5.0
1      AFG      Asia  Afghanistan  2020-02-25          5.0        0.0
2      AFG      Asia  Afghanistan  2020-02-26          5.0        0.0
3      AFG      Asia  Afghanistan  2020-02-27          5.0        0.0
4      AFG      Asia  Afghanistan  2020-02-28          5.0        0.0

   new_cases_smoothed  total_deaths  new_deaths  new_deaths_smoothed  ...  \
0                 NaN           NaN         NaN                  NaN  ...
1                 NaN           NaN         NaN                  NaN  ...
2                 NaN           NaN         NaN                  NaN  ...
3                 NaN           NaN         NaN                  NaN  ...
4                 NaN           NaN         NaN                  NaN  ...

   female_smokers  male_smokers  handwashing_facilities  \
0             NaN           NaN                  37.746
1             NaN           NaN                  37.746
2             NaN           NaN                  37.746
3             NaN           NaN                  37.746
4             NaN           NaN                  37.746

   hospital_beds_per_thousand  life_expectancy  human_development_index  \
0                         0.5            64.83                    0.511
1                         0.5            64.83                    0.511
2                         0.5            64.83                    0.511
3                         0.5            64.83                    0.511
4                         0.5            64.83                    0.511

   excess_mortality_cumulative_absolute  excess_mortality_cumulative  \
```

(continues on next page)

(continued from previous page)

```
0                                   NaN                            NaN
1                                   NaN                            NaN
2                                   NaN                            NaN
3                                   NaN                            NaN
4                                   NaN                            NaN

   excess_mortality  excess_mortality_cumulative_per_million
0               NaN                                      NaN
1               NaN                                      NaN
2               NaN                                      NaN
3               NaN                                      NaN
4               NaN                                      NaN

[5 rows x 65 columns]
```

As mentioned a lot of information is present here, about 65 columns. yet for this exercise my main objective is the 'date' column. If we would print out the data types using the info method, we can see that the date is recognized as an 'object' stating that it is an ordinary string, not a datetime.

```
covid_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 121744 entries, 0 to 121743
Data columns (total 65 columns):
 #   Column                                   Non-Null Count   Dtype
---  ------                                   --------------   -----
 0   iso_code                                 121744 non-null  object
 1   continent                                116202 non-null  object
 2   location                                 121744 non-null  object
 3   date                                     121744 non-null  object
 4   total_cases                              115518 non-null  float64
 5   new_cases                                115515 non-null  float64
 6   new_cases_smoothed                       114500 non-null  float64
 7   total_deaths                             104708 non-null  float64
 8   new_deaths                               104863 non-null  float64
 9   new_deaths_smoothed                      114500 non-null  float64
 10  total_cases_per_million                  114910 non-null  float64
 11  new_cases_per_million                    114907 non-null  float64
 12  new_cases_smoothed_per_million           113897 non-null  float64
 13  total_deaths_per_million                 104113 non-null  float64
 14  new_deaths_per_million                   104268 non-null  float64
 15  new_deaths_smoothed_per_million          113897 non-null  float64
 16  reproduction_rate                        98318 non-null   float64
 17  icu_patients                             14443 non-null   float64
 18  icu_patients_per_million                 14443 non-null   float64
 19  hosp_patients                            16504 non-null   float64
 20  hosp_patients_per_million                16504 non-null   float64
 21  weekly_icu_admissions                    1268 non-null    float64
 22  weekly_icu_admissions_per_million        1268 non-null    float64
 23  weekly_hosp_admissions                   2088 non-null    float64
 24  weekly_hosp_admissions_per_million       2088 non-null    float64
 25  new_tests                                52248 non-null   float64
 26  total_tests                              52352 non-null   float64
 27  total_tests_per_thousand                 52352 non-null   float64
 28  new_tests_per_thousand                   52248 non-null   float64
 29  new_tests_smoothed                       62816 non-null   float64
```

(continues on next page)

(continued from previous page)

```
 30   new_tests_smoothed_per_thousand          62816 non-null   float64
 31   positive_rate                            58959 non-null   float64
 32   tests_per_case                           58319 non-null   float64
 33   tests_units                              64746 non-null   object
 34   total_vaccinations                       28115 non-null   float64
 35   people_vaccinated                        26746 non-null   float64
 36   people_fully_vaccinated                  23714 non-null   float64
 37   total_boosters                           3057 non-null    float64
 38   new_vaccinations                         23298 non-null   float64
 39   new_vaccinations_smoothed                50221 non-null   float64
 40   total_vaccinations_per_hundred           28115 non-null   float64
 41   people_vaccinated_per_hundred            26746 non-null   float64
 42   people_fully_vaccinated_per_hundred      23714 non-null   float64
 43   total_boosters_per_hundred               3057 non-null    float64
 44   new_vaccinations_smoothed_per_million    50221 non-null   float64
 45   stringency_index                         101767 non-null  float64
 46   population                               120880 non-null  float64
 47   population_density                       112501 non-null  float64
 48   median_age                               107423 non-null  float64
 49   aged_65_older                            106229 non-null  float64
 50   aged_70_older                            106834 non-null  float64
 51   gdp_per_capita                           108055 non-null  float64
 52   extreme_poverty                          72482 non-null   float64
 53   cardiovasc_death_rate                    107695 non-null  float64
 54   diabetes_prevalence                      111063 non-null  float64
 55   female_smokers                           84078 non-null   float64
 56   male_smokers                             82858 non-null   float64
 57   handwashing_facilities                   54111 non-null   float64
 58   hospital_beds_per_thousand               97911 non-null   float64
 59   life_expectancy                          115458 non-null  float64
 60   human_development_index                  107790 non-null  float64
 61   excess_mortality_cumulative_absolute     4317 non-null    float64
 62   excess_mortality_cumulative              4317 non-null    float64
 63   excess_mortality                         4317 non-null    float64
 64   excess_mortality_cumulative_per_million  4317 non-null    float64
dtypes: float64(60), object(5)
memory usage: 60.4+ MB
```

We would like to change that, as we can only perform datetime operations if pandas recognises the datetime format used. Good for us, pandas has a method to automatically infer the date format, we do that now.

```
covid_df.date = pd.to_datetime(covid_df.date)
covid_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 121744 entries, 0 to 121743
Data columns (total 65 columns):
 #    Column                                   Non-Null Count   Dtype
---   ------                                   --------------   -----
 0    iso_code                                 121744 non-null  object
 1    continent                                116202 non-null  object
 2    location                                 121744 non-null  object
 3    date                                     121744 non-null  datetime64[ns]
 4    total_cases                              115518 non-null  float64
 5    new_cases                                115515 non-null  float64
 6    new_cases_smoothed                       114500 non-null  float64
```

(continues on next page)

(continued from previous page)

```
 7   total_deaths                              104708 non-null  float64
 8   new_deaths                                104863 non-null  float64
 9   new_deaths_smoothed                       114500 non-null  float64
 10  total_cases_per_million                   114910 non-null  float64
 11  new_cases_per_million                     114907 non-null  float64
 12  new_cases_smoothed_per_million            113897 non-null  float64
 13  total_deaths_per_million                  104113 non-null  float64
 14  new_deaths_per_million                    104268 non-null  float64
 15  new_deaths_smoothed_per_million           113897 non-null  float64
 16  reproduction_rate                         98318 non-null   float64
 17  icu_patients                              14443 non-null   float64
 18  icu_patients_per_million                  14443 non-null   float64
 19  hosp_patients                             16504 non-null   float64
 20  hosp_patients_per_million                 16504 non-null   float64
 21  weekly_icu_admissions                     1268 non-null    float64
 22  weekly_icu_admissions_per_million         1268 non-null    float64
 23  weekly_hosp_admissions                    2088 non-null    float64
 24  weekly_hosp_admissions_per_million        2088 non-null    float64
 25  new_tests                                 52248 non-null   float64
 26  total_tests                               52352 non-null   float64
 27  total_tests_per_thousand                  52352 non-null   float64
 28  new_tests_per_thousand                    52248 non-null   float64
 29  new_tests_smoothed                        62816 non-null   float64
 30  new_tests_smoothed_per_thousand           62816 non-null   float64
 31  positive_rate                             58959 non-null   float64
 32  tests_per_case                            58319 non-null   float64
 33  tests_units                               64746 non-null   object
 34  total_vaccinations                        28115 non-null   float64
 35  people_vaccinated                         26746 non-null   float64
 36  people_fully_vaccinated                   23714 non-null   float64
 37  total_boosters                            3057 non-null    float64
 38  new_vaccinations                          23298 non-null   float64
 39  new_vaccinations_smoothed                 50221 non-null   float64
 40  total_vaccinations_per_hundred            28115 non-null   float64
 41  people_vaccinated_per_hundred             26746 non-null   float64
 42  people_fully_vaccinated_per_hundred       23714 non-null   float64
 43  total_boosters_per_hundred                3057 non-null    float64
 44  new_vaccinations_smoothed_per_million     50221 non-null   float64
 45  stringency_index                          101767 non-null  float64
 46  population                                120880 non-null  float64
 47  population_density                        112501 non-null  float64
 48  median_age                                107423 non-null  float64
 49  aged_65_older                             106229 non-null  float64
 50  aged_70_older                             106834 non-null  float64
 51  gdp_per_capita                            108055 non-null  float64
 52  extreme_poverty                           72482 non-null   float64
 53  cardiovasc_death_rate                     107695 non-null  float64
 54  diabetes_prevalence                       111063 non-null  float64
 55  female_smokers                            84078 non-null   float64
 56  male_smokers                              82858 non-null   float64
 57  handwashing_facilities                    54111 non-null   float64
 58  hospital_beds_per_thousand                97911 non-null   float64
 59  life_expectancy                           115458 non-null  float64
 60  human_development_index                   107790 non-null  float64
 61  excess_mortality_cumulative_absolute      4317 non-null    float64
 62  excess_mortality_cumulative               4317 non-null    float64
 63  excess_mortality                          4317 non-null    float64
```

(continues on next page)

(continued from previous page)

```
 64  excess_mortality_cumulative_per_million  4317 non-null    float64
dtypes: datetime64[ns](1), float64(60), object(4)
memory usage: 60.4+ MB
```

now we are ready to perform datetime operations, however we can see that dates are appearing multiple times, this because we have records for multiple countries. I live in Belgium, so decided to isolate that subsection of the data. If they had used a data lake and partitioned into countries, reading out the data would have been much more efficient, but efficiency is not something I would expect from government as a Belgian.

```
covid_belgium_df = covid_df[covid_df.location=='Belgium'].set_index('date')
covid_belgium_df.head()
```

```
           iso_code continent location  total_cases  new_cases  \
date
2020-02-04      BEL    Europe  Belgium          1.0        1.0
2020-02-05      BEL    Europe  Belgium          1.0        0.0
2020-02-06      BEL    Europe  Belgium          1.0        0.0
2020-02-07      BEL    Europe  Belgium          1.0        0.0
2020-02-08      BEL    Europe  Belgium          1.0        0.0

            new_cases_smoothed  total_deaths  new_deaths  new_deaths_smoothed  \
date
2020-02-04                 NaN           NaN         NaN                  NaN
2020-02-05                 NaN           NaN         NaN                  NaN
2020-02-06                 NaN           NaN         NaN                  NaN
2020-02-07                 NaN           NaN         NaN                  NaN
2020-02-08                 NaN           NaN         NaN                  NaN

            total_cases_per_million  ...  female_smokers  male_smokers  \
date                                 ...
2020-02-04                    0.086  ...            25.1          31.4
2020-02-05                    0.086  ...            25.1          31.4
2020-02-06                    0.086  ...            25.1          31.4
2020-02-07                    0.086  ...            25.1          31.4
2020-02-08                    0.086  ...            25.1          31.4

            handwashing_facilities  hospital_beds_per_thousand  \
date
2020-02-04                     NaN                        5.64
2020-02-05                     NaN                        5.64
2020-02-06                     NaN                        5.64
2020-02-07                     NaN                        5.64
2020-02-08                     NaN                        5.64

            life_expectancy  human_development_index  \
date
2020-02-04            81.63                    0.931
2020-02-05            81.63                    0.931
2020-02-06            81.63                    0.931
2020-02-07            81.63                    0.931
2020-02-08            81.63                    0.931

            excess_mortality_cumulative_absolute  excess_mortality_cumulative  \
date
2020-02-04                                   NaN                          NaN
2020-02-05                                   NaN                          NaN
```

(continues on next page)

(continued from previous page)

```
2020-02-06                                          NaN                             NaN
2020-02-07                                          NaN                             NaN
2020-02-08                                          NaN                             NaN

            excess_mortality  excess_mortality_cumulative_per_million
date
2020-02-04               NaN                                      NaN
2020-02-05               NaN                                      NaN
2020-02-06               NaN                                      NaN
2020-02-07               NaN                                      NaN
2020-02-08               NaN                                      NaN

[5 rows x 64 columns]
```

Now that we have our dataset containing only Belgium I would like to emphasize another aspect, for features such as population density we would not expect a 'head count' to differ each day, and as we can see this number is steady over the whole line (results may vary for those who execute this in the future).

```
covid_belgium_df.population.value_counts()
```

```
11632334.0    611
Name: population, dtype: int64
```

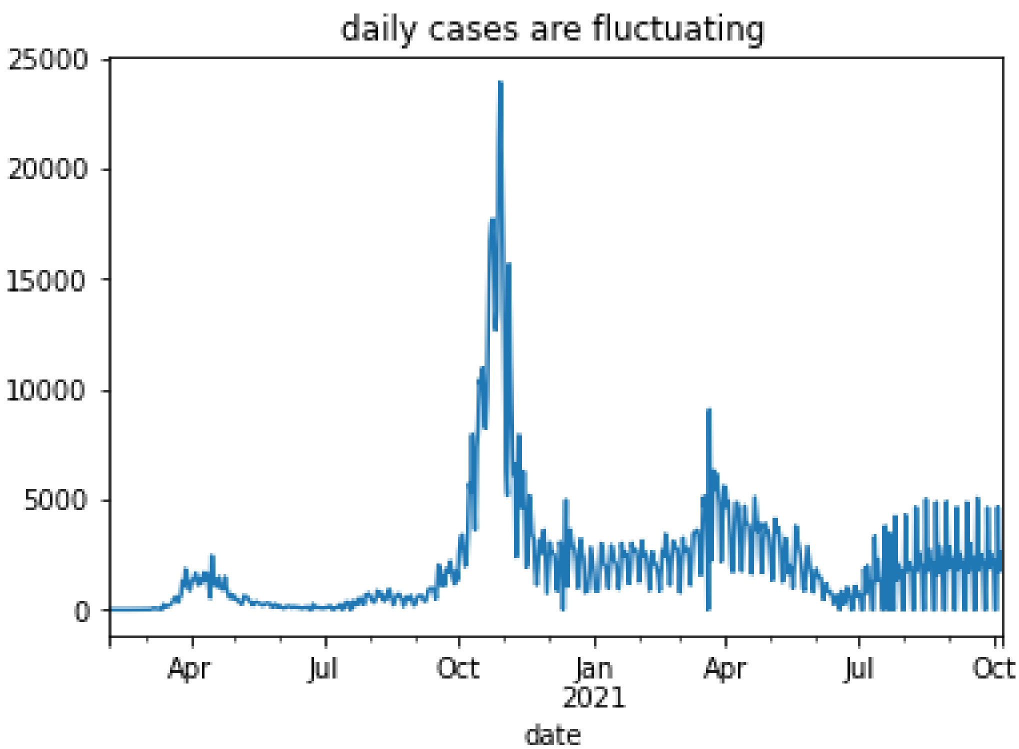
we only have a single value (in my case 11.6M) that is repeated over the whole dataset, would this look optimal to you? How would you perhaps approach this to improve data management? If you would like to go hands-on I left you a blank cell to experiment.

Optimalizations aside, we can not do that which we came for! Datetime operations, the first thing that I have in mind is that due to weekends, the cases might fluctuate a lot per day, so it is not optimal to view it on a daily basis.

First we create a simple line plot with the raw daily cases, then we perform a weekly sum to create a more smooth version of the new cases.

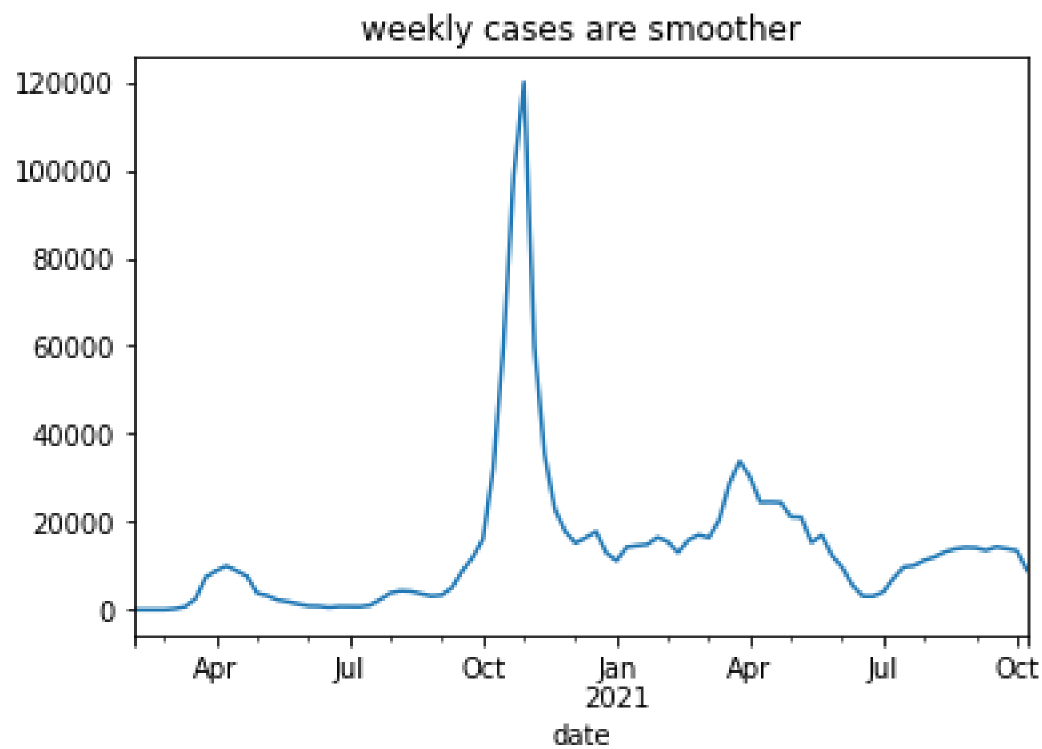
```
covid_belgium_df['new_cases'].plot(title='daily cases are fluctuating')
```

```
<AxesSubplot:title={'center':'daily cases are fluctuating'}, xlabel='date'>
```




```
weekly_cases_df = covid_belgium_df['new_cases'].resample('W').sum()
weekly_cases_df.plot(title='weekly cases are smoother')
```

```
<AxesSubplot:title={'center':'weekly cases are smoother'}, xlabel='date'>
```



That looks great! Those who inspected carefully saw that the x-axis was correclty identified as datetimes and that the y-axis for weekly sums have a much higher range.

In a next example we would like to have the relative changes from week to week, this can be done using the shift operation.

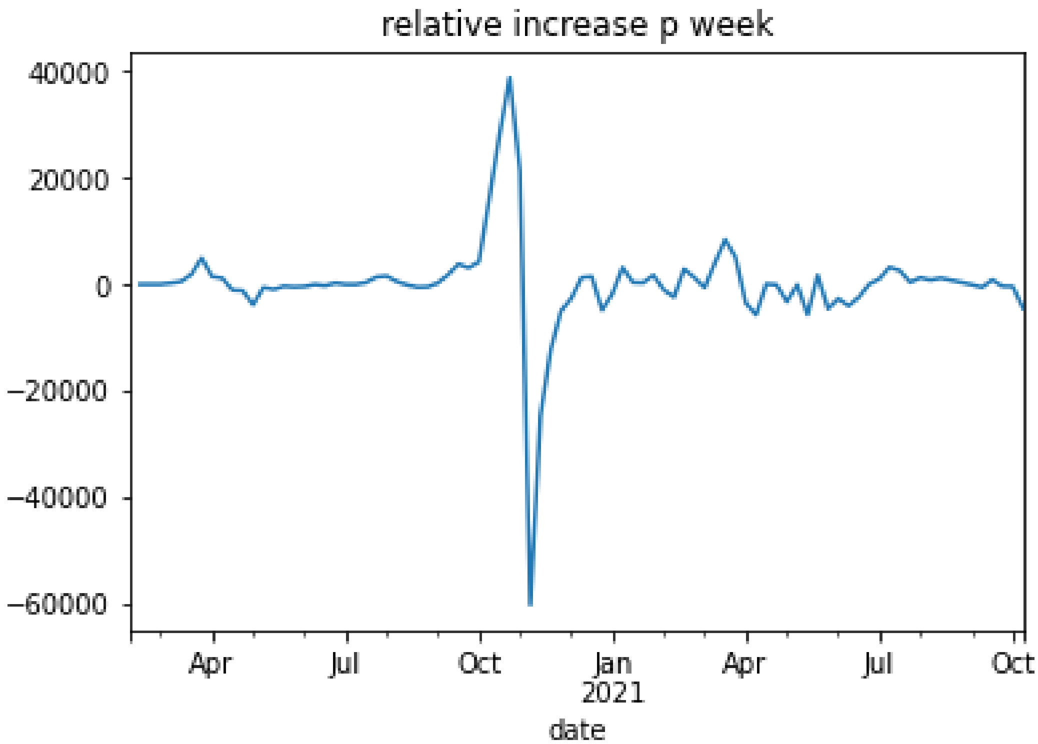
```
weekly_cases_df.shift(1)
```

```
date
2020-02-09        NaN
2020-02-16        1.0
2020-02-23        0.0
2020-03-01        0.0
2020-03-08        1.0
               ...
2021-09-12    14099.0
2021-09-19    13508.0
2021-09-26    14298.0
2021-10-03    13909.0
2021-10-10    13474.0
Freq: W-SUN, Name: new_cases, Length: 88, dtype: float64
```

This method shifted our data by 1 week forwards, this way we can subtract these results from our original data creating a relative increase (this_week_cases - last_week_cases).

```
(weekly_cases_df-weekly_cases_df.shift(1)).plot(title='relative increase p week')
```

```
<AxesSubplot:title={'center':'relative increase p week'}, xlabel='date'>
```



Another powerfull asset of datetimes is that we can utilize the concepts of days, weeks, months and years. In Belgium they speak about a phenomenon called 'the weekend effect' where a lot of reports are delayed and therefore Sundays have less cases whereas Mondays have more.

Do we see that in our data? let us seperate the Sundays and Mondays and take a mean!

```
print('mean deaths on Monday')
covid_belgium_df.loc[covid_belgium_df.index.dayofweek==0,"new_deaths"].mean()
```

```
mean deaths on Monday
```

```
39.02439024390244
```

```
print('mean deaths on Sunday')
covid_belgium_df.loc[covid_belgium_df.index.dayofweek==6,"new_deaths"].mean()
```

```
mean deaths on Sunday
```

```
36.646341463414636
```

It seems indeed that more people are reported to pass away no a Monday than on a Sunday, it would be optimal to verify this with statistics, but for now we keep it simple.

As a last example I would like to use slicing of our dataset to demonstrate we can also take a subset of our data and handle this, here we took the months of dec2020-jan2021 for belgium and calculated the total deaths.

```
covid_belgium_df.loc['2020-12-01':'2021-01-31'].new_deaths.sum()
```

```
4447.0
```

Now let's compare this to our neighbours, the Netherlands and France, we do exactly the same operations by selecting exaclty the same time window.

```
covid_netherlands_df = covid_df[covid_df.location=='Netherlands'].set_index('date')
covid_netherlands_df.loc['2020-12-01':'2021-01-31'].new_deaths.sum()
```

```
4655.0
```

```
covid_france_df = covid_df[covid_df.location=='France'].set_index('date')
covid_france_df.loc['2020-12-01':'2021-01-31'].new_deaths.sum()
```

```
23382.0
```

You can see that Belgium has the lowest of total deaths in that time interval, so you could assume we performed the best! However this approach is a bit simplified as there are not as many Belgians as French and Dutch. Could you perhaps think if an improvement to create a better understanding?

CHAPTER

EIGHT

CATEGORICAL ENCODING

Often we deal with categorical data and this kind of data is something computer algorithms are not able to understand. On the other hand long categorical features might take up unnecessary memory in our dataset, so converting to a categorical feature is optimal.

```
import pandas as pd
```

8.1 Raw Material Charaterization

In this dataset, we have a few numerical features describing characteristics of our material, next to that we also have an Outcome feature describing the state of our material in a category.

Let's have a look at the data

```
raw_material_df = pd.read_csv('./data/raw-material-characterization.csv')
raw_material_df.head()
```

```
  Lot number   Outcome  Size5  Size10  Size15    TGA   DSC   TMA
0       B370  Adequate   13.8     9.2    41.2  787.3  18.0  65.0
1       B880  Adequate   11.2     5.8    27.6  772.2  17.7  68.8
2       B452  Adequate    9.9     5.8    28.3  602.3  18.3  50.7
3       B287  Adequate   10.4     4.0    24.7  677.9  17.7  56.5
4       B576  Adequate   12.3     9.3    22.0  593.5  19.5  52.0
```

So we can see that the outcome is indeed a text field with a human interpretable value. The different values are:

```
raw_material_df.Outcome.unique()
```

```
array(['Adequate', 'Poor'], dtype=object)
```

Image that we would like to get all records where the Outcome is less than adequate, using strings this is not possible as the computer does not understand relations of Adequate and Poor when they are denoted as text.

```
raw_material_df[raw_material_df.Outcome<'Adequate']
```

```
Empty DataFrame
Columns: [Lot number, Outcome, Size5, Size10, Size15, TGA, DSC, TMA]
Index: []
```

To overcome this we can change the type of the feature from 'object' (string) to 'category' let us look at the data types of our data

```
raw_material_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24 entries, 0 to 23
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Lot number  24 non-null     object
 1   Outcome     24 non-null     object
 2   Size5       24 non-null     float64
 3   Size10      24 non-null     float64
 4   Size15      24 non-null     float64
 5   TGA         24 non-null     float64
 6   DSC         24 non-null     float64
 7   TMA         24 non-null     float64
dtypes: float64(6), object(2)
memory usage: 1.6+ KB
```

Now we can change that of Outcome to category using the astype method

```
raw_material_df.Outcome = raw_material_df.Outcome.astype('category')
raw_material_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24 entries, 0 to 23
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Lot number  24 non-null     object
 1   Outcome     24 non-null     category
 2   Size5       24 non-null     float64
 3   Size10      24 non-null     float64
 4   Size15      24 non-null     float64
 5   TGA         24 non-null     float64
 6   DSC         24 non-null     float64
 7   TMA         24 non-null     float64
dtypes: category(1), float64(6), object(1)
memory usage: 1.6+ KB
```

Our feature might be of categorical nature now, however we still have to define it is an ordinal category and has an order.

```
raw_material_df.Outcome = raw_material_df.Outcome.cat.as_ordered().cat.reorder_
↪categories(['Poor', 'Adequate'])
```

If we retry to effort to only take the records where the Outcome is less than Adequate, we now get an outcome! Since we only have 2 categories this is a bit unfortunate, but you should get the idea behind it.

```
raw_material_df[raw_material_df.Outcome<'Adequate']
```

```
   Lot number Outcome  Size5  Size10  Size15    TGA   DSC   TMA
5        B914    Poor   13.7     7.8    27.0  597.9  18.1  49.8
6        B404    Poor   15.5    10.7    34.3  668.5  19.6  55.7
7        B694    Poor   15.4    10.7    35.9  602.8  19.2  53.6
8        B875    Poor   14.9    11.3    41.0  614.6  18.5  50.0
10       B517    Poor   16.1    11.6    39.2  682.8  17.5  56.4
```

(continues on next page)

(continued from previous page)

```
13          B430    Poor    12.9       9.7     36.3   642.4   19.1   55.0
21          B745    Poor    10.2       5.8     24.7   575.9   18.5   46.2
```

Let's take this a step further, since computer algorithms still have no idea what the numerical relation is between Adequate and Poor, we could use a Label Encoder for that.

```
from sklearn.preprocessing import LabelEncoder
```

the label encoder is inputted with the Outcome feature and recognises 2 types, it chooses a numerical value for each while fitting.

```
le = LabelEncoder()
le.fit(raw_material_df.Outcome)
```

```
LabelEncoder()
```

After fitting we can use this encoder to transform our dataset!

```
raw_material_df['outcome_label'] = le.transform(raw_material_df.Outcome)
raw_material_df.head()
```

```
  Lot number   Outcome  Size5  Size10  Size15    TGA   DSC   TMA  \
0       B370  Adequate   13.8     9.2    41.2  787.3  18.0  65.0
1       B880  Adequate   11.2     5.8    27.6  772.2  17.7  68.8
2       B452  Adequate    9.9     5.8    28.3  602.3  18.3  50.7
3       B287  Adequate   10.4     4.0    24.7  677.9  17.7  56.5
4       B576  Adequate   12.3     9.3    22.0  593.5  19.5  52.0

   outcome_label
0              0
1              0
2              0
3              0
4              0
```

It seems something unfortunate has happened, the encoder gave the Adequate an outcome label of 0, which is lower than the label of Poor (1), this might be bad if we would like to give a score as our outcome.

There is in pandas another method of mapping a label to a category albeit less automated, as you would have to know the categories in your feature.

```
raw_material_df.outcome_label = raw_material_df.Outcome.map({'Poor': 0, 'Adequate':1})
raw_material_df.head()
```

```
  Lot number   Outcome  Size5  Size10  Size15    TGA   DSC   TMA outcome_label
0       B370  Adequate   13.8     9.2    41.2  787.3  18.0  65.0             1
1       B880  Adequate   11.2     5.8    27.6  772.2  17.7  68.8             1
2       B452  Adequate    9.9     5.8    28.3  602.3  18.3  50.7             1
3       B287  Adequate   10.4     4.0    24.7  677.9  17.7  56.5             1
4       B576  Adequate   12.3     9.3    22.0  593.5  19.5  52.0             1
```

Yes! This did the trick, now we can use that outcome label to predict an outcome for future samples.

CHAPTER

NINE

RESTAURANT TIPS

Now we are going to look at a dataset of tips, here a restaurant tracked the table bills and tips for a few days in the week whilst also noting the gender, smoking habit and time of day. This led to a small yet very interesting dataset, let's have a look!

```
tips_df = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/
↪tips.csv')
tips_df
```

```
     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female     No   Sun  Dinner     2
1         10.34  1.66    Male     No   Sun  Dinner     3
2         21.01  3.50    Male     No   Sun  Dinner     3
3         23.68  3.31    Male     No   Sun  Dinner     2
4         24.59  3.61  Female     No   Sun  Dinner     4
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]
```

We can see here that we have a lot of categorical variables: gender, smoker, the day and the time. In later sections we will see how we can aggregate on these categorical variables. Now however we would like to process them for a machine learning exercise, where we need numbers not text. For the features smoker and day, you could argue there is a clear numbering between them, smoking is 1 if the person was smoking and e.g. Sun relates to 7 as it is the seventh day of the week.

But for the gender this is different, we can't really say that women are 1 and Men are 0 or vice versa (although in this binary case it might work). The same theory applies for time, if we would say that breakfast, lunch and dinner equal to 0, 1 and 2 this would give our algorithm a bad impression as it would think dinner is twice lunch…

We use One Hot Encoding for this, the idea is that for each value of the feature we create a new column.

```
from sklearn.preprocessing import OneHotEncoder
```

First we create our encoder, then we give it the day column to learn and see which values of categories there are.

```
ohe = OneHotEncoder()
ohe.fit(tips_df[['day']])
```

```
OneHotEncoder()
```

Now we can perform an encoding, here we insert the day column and it returns a matrix with 4 columns corresponding to the 4 days in our feature.

```
ohe.transform(tips_df[['day']]).todense()
```

```
matrix([[0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
```

(continues on next page)

(continued from previous page)

```
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
```

(continues on next page)

(continued from previous page)

```
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
```

(continues on next page)

(continued from previous page)

```
       [0., 0., 1., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
```

(continues on next page)

(continued from previous page)

```
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 0., 1.]])
```

As this is a rather mathematical approach for this simple problem I prefer to use the pandas approach for this, which is the get_dummies method. The outcome is much more pleasing yet completely the same.

```
pd.get_dummies(tips_df.day)
```

```
     Fri  Sat  Sun  Thur
0      0    0    1     0
1      0    0    1     0
2      0    0    1     0
3      0    0    1     0
4      0    0    1     0
..   ...  ...  ...   ...
239    0    1    0     0
240    0    1    0     0
241    0    1    0     0
242    0    1    0     0
243    0    0    0     1

[244 rows x 4 columns]
```

As an exercise you could create a script that given a specific feature (e.g. day):

- extracts that feature
- creates dummies
- concattenates it to the dataframe

Good luck!

CHAPTER TEN

SCALING AND NORMALIZATION

In this notebook we are going to look into 2 rather mathematical concepts, Scaling and Normalization. The idea is that we can scale the values and shape the distribution of feature in our dataset.

As an example we take a dataset containing samples from a low density polyethylene production process, containing several numerical features such as temperatures, Forces, Pressure,…

The idea is that by using Scaling and normalization, the 'range of motion' for these sensors is equal and we can compare the fluxtuations not only inbetween records, but also inbetween sensors.

```
import pandas as pd
```

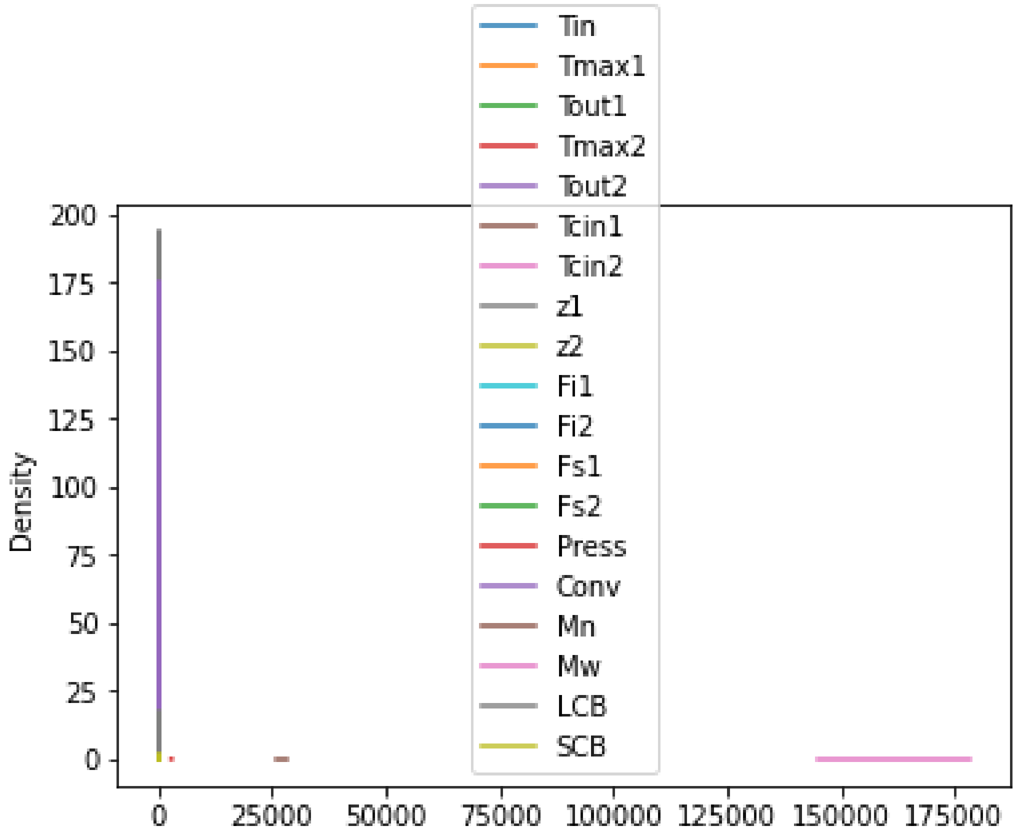
```
ldpe_df = pd.read_csv('https://openmv.net/file/LDPE.csv').drop(columns=['Unnamed: 0'])
ldpe_df.head()
```

```
      Tin   Tmax1   Tout1   Tmax2   Tout2   Tcin1   Tcin2     z1     z2  \
0  208.17  296.35  233.81  283.41  239.05  117.14  117.20  0.029  0.581
1  207.26  298.26  230.88  287.55  242.55  116.39  117.23  0.028  0.574
2  205.30  296.57  235.38  284.35  245.19  117.33  118.42  0.031  0.578
3  209.29  294.11  225.61  283.31  242.04  116.15  117.94  0.030  0.581
4  206.76  295.13  230.26  283.74  244.92  116.75  118.49  0.030  0.579

      Fi1     Fi2     Fs1     Fs2  Press    Conv     Mn      Mw    LCB    SCB
0  0.4507  0.4518  666.42  248.95   3021  0.1322  27379  160326  0.781  26.11
1  0.4765  0.5091  658.61  246.36   3033  0.1365  27043  165044  0.819  26.29
2  0.4744  0.4505  666.51  244.65   3004  0.1335  27344  165621  0.801  26.13
3  0.4429  0.4516  667.31  242.28   2980  0.1300  27502  160497  0.778  25.92
4  0.4394  0.4414  670.83  244.31   2997  0.1316  27518  165713  0.786  26.02
```

We can see that our features clearly have different ranges, but lets try to visualise these ranges using a density plot

```
ldpe_df.plot(kind='density')
```

```
<AxesSubplot:ylabel='Density'>
```

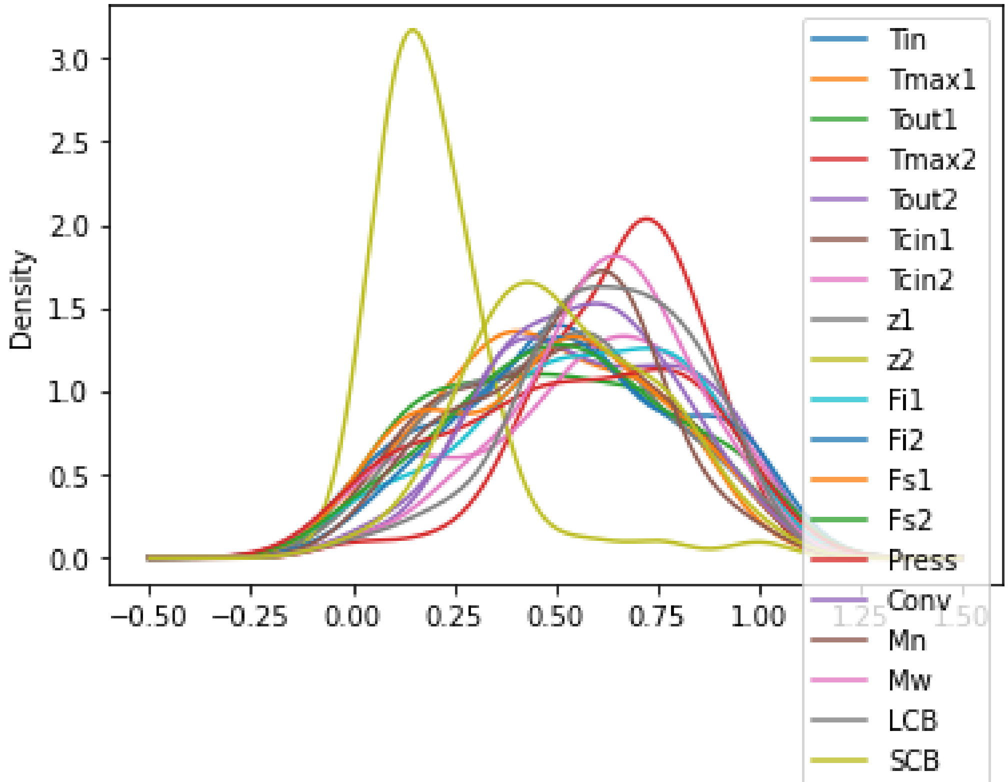


Ouch, this is clearly not working! Because the 'Mw' feature is in the range of 150k-175k our plot is so dilluted the rest are pinned to 0. We can use the sklearn library to perform a min max scaling, this scaling will shift the distribution of each feature between 0 and 1, but that can also be adjusted.

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
scaler.fit(ldpe_df)
pd.DataFrame(scaler.transform(ldpe_df), columns=ldpe_df.columns).plot(kind='density')
```

```
<AxesSubplot:ylabel='Density'>
```



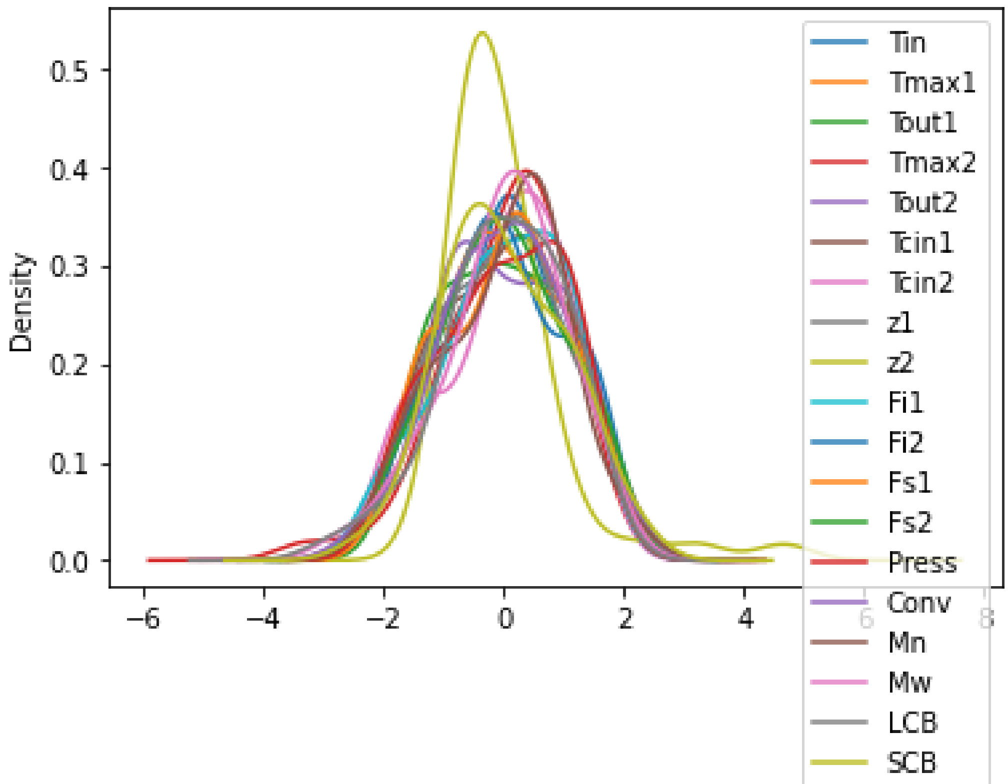
That makes a lot more sense, you can now see all of the distribution at once. Also there seems to be one (yellow) feature that has some outliers perhaps something weird is going on there…

Taking it a step further we could also alter the distributions by using a standard scaler instead of a min max scaler, redistributing the values mathematically into a normal distribution.

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaler.fit(ldpe_df)
pd.DataFrame(scaler.transform(ldpe_df), columns=ldpe_df.columns).plot(kind='density')
```

```
<AxesSubplot:ylabel='Density'>
```



You can see it had some trouble fitting our special feature into the normal distribution but it did work out in the end. With this we are ready to perform machine learning algorithms on this data, but first why not try and figure out where those outliers are I mentioned earlier?

CHAPTER

ELEVEN

BINNING AND RANKING

When dealing with numerical data the trouble can sometimes be that numbers can have a wide variety.

Here we apply 2 methods to deal with that, binning and ranking. With binning we change the numerical feature into a categorical/ordinal feature. Ranking is used when our numerical feature contains a non normal distribution that fails to be normalized.

For this example we use a food consumption dataset, where european countries are listed and the relative percentage of each country is given that consumes the type of food, e.g. a value of 67 means that 67% of that country eats that type of food.

```
import pandas as pd
pd.set_option('display.max_columns', None)
```

```
food_df = pd.read_csv('https://openmv.net/file/food-consumption.csv')
food_df
```

```
        Country  Real coffee  Instant coffee  Tea  Sweetener  Biscuits  \
0       Germany           90              49   88       19.0      57.0
1         Italy           82              10   60        2.0      55.0
2        France           88              42   63        4.0      76.0
3       Holland           96              62   98       32.0      62.0
4       Belgium           94              38   48       11.0      74.0
5    Luxembourg           97              61   86       28.0      79.0
6       England           27              86   99       22.0      91.0
7      Portugal           72              26   77        2.0      22.0
8       Austria           55              31   61       15.0      29.0
9   Switzerland           73              72   85       25.0      31.0
10       Sweden           97              13   93       31.0       NaN
11      Denmark           96              17   92       35.0      66.0
12       Norway           92              17   83       13.0      62.0
13      Finland           98              12   84       20.0      64.0
14        Spain           70              40   40        NaN      62.0
15      Ireland           30              52   99       11.0      80.0

    Powder soup  Tin soup  Potatoes  Frozen fish  Frozen veggies  Apples  \
0            51        19        21           27              21      81
1            41         3         2            4               2      67
2            53        11        23           11               5      87
3            67        43         7           14              14      83
4            37        23         9           13              12      76
5            73        12         7           26              23      85
6            55        76        17           20              24      76
7            34         1         5           20               3      22
```

(continues on next page)

(continued from previous page)

```
8           33        1         5           15             11      49
9           69       10        17           19             15      79
10          43       43        39           54             45      56
11          32       17        11           51             42      81
12          51        4        17           30             15      61
13          27       10         8           18             12      50
14          43        2        14           23              7      59
15          75       18         2            5              3      57

    Oranges  Tinned fruit  Jam  Garlic  Butter  Margarine  Olive oil  Yoghurt  \
0        75            44   71      22      91         85         74     30.0
1        71             9   46      80      66         24         94      5.0
2        84            40   45      88      94         47         36     57.0
3        89            61   81      15      31         97         13     53.0
4        76            42   57      29      84         80         83     20.0
5        94            83   20      91      94         94         84     31.0
6        68            89   91      11      95         94         57     11.0
7        51             8   16      89      65         78         92      6.0
8        42            14   41      51      51         72         28     13.0
9        70            46   61      64      82         48         61     48.0
10       78            53   75       9      68         32         48      2.0
11       72            50   64      11      92         91         30     11.0
12       72            34   51      11      63         94         28      2.0
13       57            22   37      15      96         94         17      NaN
14       77            30   38      86      44         51         91     16.0
15       52            46   89       5      97         25         31      3.0

    Crisp bread
0            26
1            18
2             3
3            15
4             5
5            24
6            28
7             9
8            11
9            30
10           93
11           34
12           62
13           64
14           13
15            9
```

Here you could do some data validity, where we check if all values are between 0 and 100, or we check for missing values. I will leave that up to you

11.1 Binning

the first thing we want to do is seperate the countries based on their coffee consumption, instead of creating arbitrary values we can perform a quantitative cut. This means we create a number of equally sized groups using the qcut function, we give them the labels low, medium and high.

```
food_df['bin_coffee'] = pd.qcut(food_df['Real coffee'], q=3, labels=['low', 'medium',
↪'high'])
food_df
```

```
        Country  Real coffee  Instant coffee  Tea  Sweetener  Biscuits  \
0       Germany           90              49   88       19.0      57.0
1         Italy           82              10   60        2.0      55.0
2        France           88              42   63        4.0      76.0
3       Holland           96              62   98       32.0      62.0
4       Belgium           94              38   48       11.0      74.0
5    Luxembourg           97              61   86       28.0      79.0
6       England           27              86   99       22.0      91.0
7      Portugal           72              26   77        2.0      22.0
8       Austria           55              31   61       15.0      29.0
9   Switzerland           73              72   85       25.0      31.0
10       Sweden           97              13   93       31.0       NaN
11      Denmark           96              17   92       35.0      66.0
12       Norway           92              17   83       13.0      62.0
13      Finland           98              12   84       20.0      64.0
14        Spain           70              40   40        NaN      62.0
15      Ireland           30              52   99       11.0      80.0

    Powder soup  Tin soup  Potatoes  Frozen fish  Frozen veggies  Apples  \
0            51        19        21           27              21      81
1            41         3         2            4               2      67
2            53        11        23           11               5      87
3            67        43         7           14              14      83
4            37        23         9           13              12      76
5            73        12         7           26              23      85
6            55        76        17           20              24      76
7            34         1         5           20               3      22
8            33         1         5           15              11      49
9            69        10        17           19              15      79
10           43        43        39           54              45      56
11           32        17        11           51              42      81
12           51         4        17           30              15      61
13           27        10         8           18              12      50
14           43         2        14           23               7      59
15           75        18         2            5               3      57

    Oranges  Tinned fruit  Jam  Garlic  Butter  Margarine  Olive oil  Yoghurt  \
0        75            44   71      22      91         85         74     30.0
1        71             9   46      80      66         24         94      5.0
2        84            40   45      88      94         47         36     57.0
3        89            61   81      15      31         97         13     53.0
4        76            42   57      29      84         80         83     20.0
5        94            83   20      91      94         94         84     31.0
6        68            89   91      11      95         94         57     11.0
7        51             8   16      89      65         78         92      6.0
8        42            14   41      51      51         72         28     13.0
9        70            46   61      64      82         48         61     48.0
```

(continues on next page)

(continued from previous page)

```
10       78            53  75       9      68         32         48     2.0
11       72            50  64      11      92         91         30    11.0
12       72            34  51      11      63         94         28     2.0
13       57            22  37      15      96         94         17     NaN
14       77            30  38      86      44         51         91    16.0
15       52            46  89       5      97         25         31     3.0

    Crisp bread bin_coffee
0            26     medium
1            18     medium
2             3     medium
3            15       high
4             5     medium
5            24       high
6            28        low
7             9        low
8            11        low
9            30        low
10           93       high
11           34       high
12           62     medium
13           64       high
14           13        low
15            9        low
```

a new column has appeared at the end of our dataframe, containing the labels of our binning, countries with low coffee consumption are put in the low category and vice versa. Now we can seperate the countries with low coffee consumption from the rest

```
food_df[food_df.bin_coffee == 'low']
```

```
        Country  Real coffee  Instant coffee  Tea  Sweetener  Biscuits  \
6       England           27              86   99       22.0      91.0
7      Portugal           72              26   77        2.0      22.0
8       Austria           55              31   61       15.0      29.0
9   Switzerland           73              72   85       25.0      31.0
14        Spain           70              40   40        NaN      62.0
15      Ireland           30              52   99       11.0      80.0

    Powder soup  Tin soup  Potatoes  Frozen fish  Frozen veggies  Apples  \
6            55        76        17           20              24      76
7            34         1         5           20               3      22
8            33         1         5           15              11      49
9            69        10        17           19              15      79
14           43         2        14           23               7      59
15           75        18         2            5               3      57

    Oranges  Tinned fruit  Jam  Garlic  Butter  Margarine  Olive oil  Yoghurt  \
6        68            89   91      11      95         94         57     11.0
7        51             8   16      89      65         78         92      6.0
8        42            14   41      51      51         72         28     13.0
9        70            46   61      64      82         48         61     48.0
14       77            30   38      86      44         51         91     16.0
15       52            46   89       5      97         25         31      3.0

    Crisp bread bin_coffee
```

(continues on next page)

(continued from previous page)

```
6              28        low
7               9        low
8              11        low
9              30        low
14             13        low
15              9        low
```

You can already see the England/Ireland stereotype here, note that those are the only 2 with really low coffee consumption, the others are only in this low binning because we requested equally spaced bins in our qcut function. using the cut function would result in a different outcome. Perhaps you could try that out?

I tried to think of some metric to quantify the status of coffee drinkers, since we also have the instant coffee consumption we could create a metric where we subtract the amount of instant coffe drinkers from the amount of real coffee drinkers. This way we can measure that difference between them, I already went ahead and made equal quantity bins for them with labels low, medium and high 'quality coffee'.

```
food_df['bin_qual_coffee'] = pd.qcut(food_df['Real coffee'] - food_df['Instant coffee
↪'], q=3, labels=['low', 'medium', 'high'])
```

```
food_df[food_df.bin_qual_coffee=='high']
```

```
    Country  Real coffee  Instant coffee  Tea  Sweetener  Biscuits  \
1     Italy           82              10   60        2.0      55.0
10   Sweden           97              13   93       31.0       NaN
11  Denmark           96              17   92       35.0      66.0
12   Norway           92              17   83       13.0      62.0
13  Finland           98              12   84       20.0      64.0

    Powder soup  Tin soup  Potatoes  Frozen fish  Frozen veggies  Apples  \
1            41         3         2            4               2      67
10           43        43        39           54              45      56
11           32        17        11           51              42      81
12           51         4        17           30              15      61
13           27        10         8           18              12      50

    Oranges  Tinned fruit  Jam  Garlic  Butter  Margarine  Olive oil  Yoghurt  \
1        71             9   46      80      66         24         94      5.0
10       78            53   75       9      68         32         48      2.0
11       72            50   64      11      92         91         30     11.0
12       72            34   51      11      63         94         28      2.0
13       57            22   37      15      96         94         17      NaN

    Crisp bread bin_coffee bin_qual_coffee
1            18     medium            high
10           93       high            high
11           34       high            high
12           62     medium            high
13           64       high            high
```

Aha! you can see here which countries prefer the real coffee over the instant version. It seems the scandinavian countries together with obviously Italy are the true Caffeine connoisseur of Europe. Another intersting thing we can do now is take the mean for each product for both group high and low and take the difference for high - low. We can see the result below

```
food_df[food_df.bin_qual_coffee=='high'].mean()-food_df[food_df.bin_qual_coffee=='low
↪'].mean()
```

```
/tmp/ipykernel_16521/3908782487.py:1: FutureWarning: Dropping of nuisance columns in␣
↪DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version␣
↪this will raise TypeError.  Select only valid columns before calling the reduction.
  food_df[food_df.bin_qual_coffee=='high'].mean()-food_df[food_df.bin_qual_coffee==
↪'low'].mean()
```

```
Real coffee       34.500000
Instant coffee   -43.366667
Tea                2.066667
Sweetener         -0.800000
Biscuits           2.583333
Powder soup      -18.200000
Tin soup          -9.600000
Potatoes           5.066667
Frozen fish       15.400000
Frozen veggies    10.866667
Apples            -4.166667
Oranges            3.666667
Tinned fruit     -14.066667
Jam              -12.233333
Garlic           -13.466667
Butter            10.333333
Margarine          2.500000
Olive oil         -3.433333
Yoghurt          -19.000000
Crisp bread       36.533333
dtype: float64
```

It seems a preference for quality coffee also pairs with crisp bread, who knew? Do you think scaling/normalization might be interesting here? why (not)?

11.2 Ranking

In case normalization fails us and we are for some reason not able to get a normal distribution out of a feature, we can still resort to ranking. Note that non linear machine learning techniques often use a ranking functionality under the hood, therefore this technique is often not required, yet for educational purposes we are going to use it here anyway. Let's see how the distribution for Real coffee consumption looks like.

```
food_df.sort_values('Real coffee')
```

```
        Country  Real coffee  Instant coffee  Tea  Sweetener  Biscuits  \
6       England           27              86   99       22.0      91.0
15      Ireland           30              52   99       11.0      80.0
8       Austria           55              31   61       15.0      29.0
14        Spain           70              40   40        NaN      62.0
7      Portugal           72              26   77        2.0      22.0
9   Switzerland           73              72   85       25.0      31.0
1         Italy           82              10   60        2.0      55.0
2        France           88              42   63        4.0      76.0
0       Germany           90              49   88       19.0      57.0
12       Norway           92              17   83       13.0      62.0
4       Belgium           94              38   48       11.0      74.0
3       Holland           96              62   98       32.0      62.0
11      Denmark           96              17   92       35.0      66.0
```

(continues on next page)

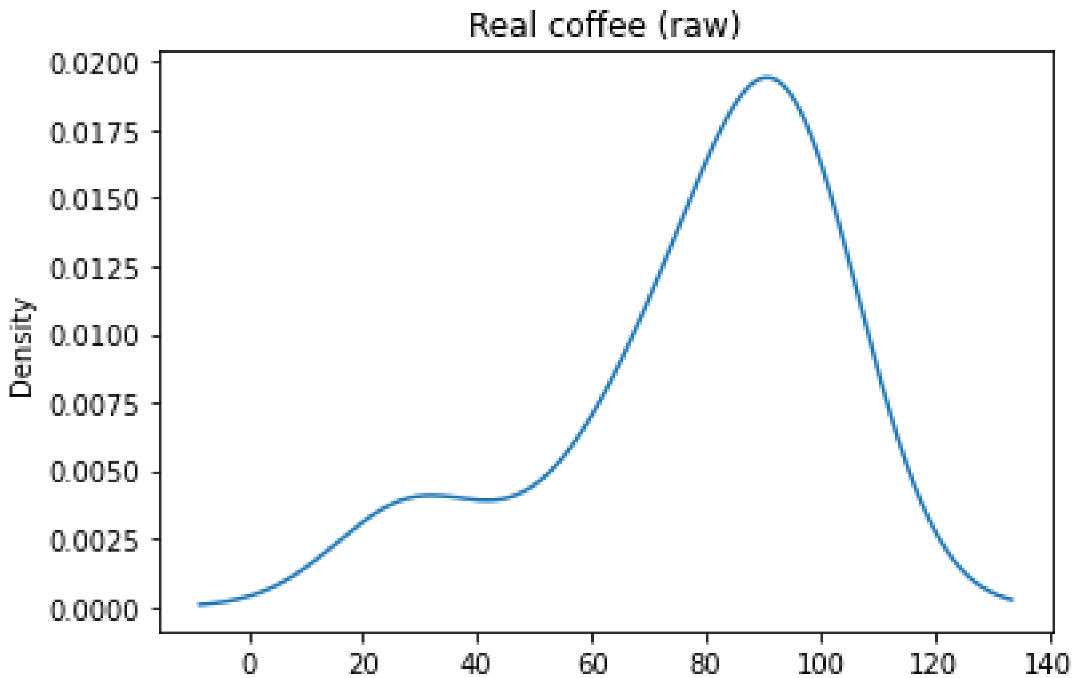
(continued from previous page)

```
5    Luxembourg           97              61   86       28.0      79.0
10       Sweden           97              13   93       31.0       NaN
13      Finland           98              12   84       20.0      64.0

    Powder soup  Tin soup  Potatoes  Frozen fish  Frozen veggies  Apples  \
6            55        76        17           20              24      76
15           75        18         2            5               3      57
8            33         1         5           15              11      49
14           43         2        14           23               7      59
7            34         1         5           20               3      22
9            69        10        17           19              15      79
1            41         3         2            4               2      67
2            53        11        23           11               5      87
0            51        19        21           27              21      81
12           51         4        17           30              15      61
4            37        23         9           13              12      76
3            67        43         7           14              14      83
11           32        17        11           51              42      81
5            73        12         7           26              23      85
10           43        43        39           54              45      56
13           27        10         8           18              12      50

    Oranges  Tinned fruit  Jam  Garlic  Butter  Margarine  Olive oil  Yoghurt  \
6        68            89   91      11      95         94         57     11.0
15       52            46   89       5      97         25         31      3.0
8        42            14   41      51      51         72         28     13.0
14       77            30   38      86      44         51         91     16.0
7        51             8   16      89      65         78         92      6.0
9        70            46   61      64      82         48         61     48.0
1        71             9   46      80      66         24         94      5.0
2        84            40   45      88      94         47         36     57.0
0        75            44   71      22      91         85         74     30.0
12       72            34   51      11      63         94         28      2.0
4        76            42   57      29      84         80         83     20.0
3        89            61   81      15      31         97         13     53.0
11       72            50   64      11      92         91         30     11.0
5        94            83   20      91      94         94         84     31.0
10       78            53   75       9      68         32         48      2.0
13       57            22   37      15      96         94         17      NaN

    Crisp bread bin_coffee bin_qual_coffee
6            28        low             low
15            9        low             low
8            11        low             low
14           13        low             low
7             9        low          medium
9            30        low             low
1            18     medium            high
2             3     medium          medium
0            26     medium          medium
12           62     medium            high
4             5     medium          medium
3            15       high             low
11           34       high            high
5            24       high          medium
10           93       high            high
13           64       high            high
```

Ah yes, about half of the values are 90 or higher, not really optimal as the range is between 0 and 100! We can also view this in a visual way using a density plot.

```
food_df['Real coffee'].plot(kind='density', title='Real coffee (raw)')
```

```
<AxesSubplot:title={'center':'Real coffee (raw)'}, ylabel='Density'>
```



For larger datasets this would be more useful as we cannot see our whole dataset, it is clear we have to do something about this, now imagine we can not use regular normalization techniques. The rank method now comes in handy!

```
food_df['rank_coffee'] = food_df['Real coffee'].rank()
food_df
```

```
        Country  Real coffee  Instant coffee  Tea  Sweetener  Biscuits  \
0       Germany           90              49   88       19.0      57.0
1         Italy           82              10   60        2.0      55.0
2        France           88              42   63        4.0      76.0
3       Holland           96              62   98       32.0      62.0
4       Belgium           94              38   48       11.0      74.0
5    Luxembourg           97              61   86       28.0      79.0
6       England           27              86   99       22.0      91.0
7      Portugal           72              26   77        2.0      22.0
8       Austria           55              31   61       15.0      29.0
9   Switzerland           73              72   85       25.0      31.0
10       Sweden           97              13   93       31.0       NaN
11      Denmark           96              17   92       35.0      66.0
12       Norway           92              17   83       13.0      62.0
13      Finland           98              12   84       20.0      64.0
14        Spain           70              40   40        NaN      62.0
15      Ireland           30              52   99       11.0      80.0

    Powder soup  Tin soup  Potatoes  Frozen fish  Frozen veggies  Apples  \
```

(continues on next page)

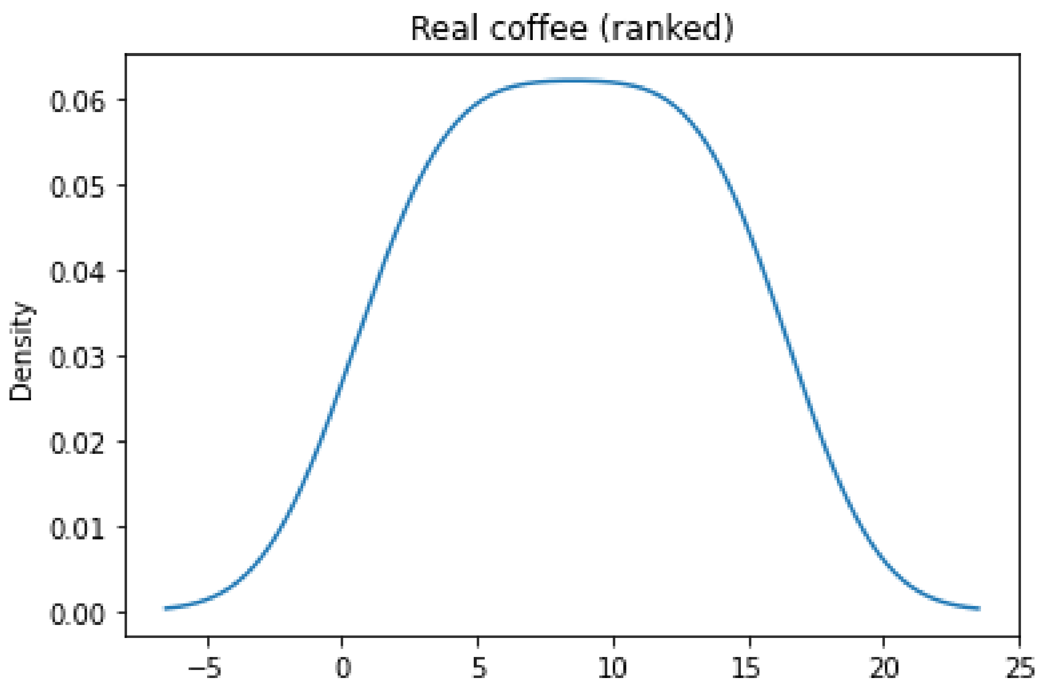
(continued from previous page)

```
0           51        19        21           27              21       81
1           41         3         2            4               2       67
2           53        11        23           11               5       87
3           67        43         7           14              14       83
4           37        23         9           13              12       76
5           73        12         7           26              23       85
6           55        76        17           20              24       76
7           34         1         5           20               3       22
8           33         1         5           15              11       49
9           69        10        17           19              15       79
10          43        43        39           54              45       56
11          32        17        11           51              42       81
12          51         4        17           30              15       61
13          27        10         8           18              12       50
14          43         2        14           23               7       59
15          75        18         2            5               3       57

    Oranges  Tinned fruit  Jam  Garlic  Butter  Margarine  Olive oil  Yoghurt  \
0        75            44   71      22      91         85         74     30.0
1        71             9   46      80      66         24         94      5.0
2        84            40   45      88      94         47         36     57.0
3        89            61   81      15      31         97         13     53.0
4        76            42   57      29      84         80         83     20.0
5        94            83   20      91      94         94         84     31.0
6        68            89   91      11      95         94         57     11.0
7        51             8   16      89      65         78         92      6.0
8        42            14   41      51      51         72         28     13.0
9        70            46   61      64      82         48         61     48.0
10       78            53   75       9      68         32         48      2.0
11       72            50   64      11      92         91         30     11.0
12       72            34   51      11      63         94         28      2.0
13       57            22   37      15      96         94         17      NaN
14       77            30   38      86      44         51         91     16.0
15       52            46   89       5      97         25         31      3.0

    Crisp bread bin_coffee bin_qual_coffee  rank_coffee
0            26     medium          medium          9.0
1            18     medium            high          7.0
2             3     medium          medium          8.0
3            15       high             low         12.5
4             5     medium          medium         11.0
5            24       high          medium         14.5
6            28        low             low          1.0
7             9        low          medium          5.0
8            11        low             low          3.0
9            30        low             low          6.0
10           93       high            high         14.5
11           34       high            high         12.5
12           62     medium            high         10.0
13           64       high            high         16.0
14           13        low             low          4.0
15            9        low             low          2.0
```

At the end of our data a new column was appended, containing the ranking of each country with the lowest being 1 and the highest equal to the amount of countries. When we visualise this distribution we get a uniform distribution, not normal but still better than before!

```
food_df['rank_coffee'].plot(kind='density', title='Real coffee (ranked)')
```

```
<AxesSubplot:title={'center':'Real coffee (ranked)'}, ylabel='Density'>
```



CHAPTER TWELVE

SOME PRACTICE

Now that you have learned techniques in data preparation, why don't you put them to use in this wonderfully horrifying dataset. Good luck!

```
import os
import json

import pandas as pd
```

```
kaggle_dir = os.path.expanduser("~/.kaggle")
if not os.path.exists(kaggle_dir):
    os.mkdir(kaggle_dir)

with open(f'{kaggle_dir}/kaggle.json', 'w') as f:
    json.dump(
        {
            "username":"lorenzf",
            "key":"7a44a9e99b27e796177d793a3d85b8cf"
        }
        , f)
```

```
import kaggle
kaggle.api.dataset_download_files(dataset='PromptCloudHQ/us-jobs-on-monstercom', path=
↪'./data', unzip=True)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_25600/39646943.py in <module>
----> 1 import kaggle
      2 kaggle.api.dataset_download_files(dataset='PromptCloudHQ/us-jobs-on-monstercom
↪', path='./data', unzip=True)

ModuleNotFoundError: No module named 'kaggle'
```

```
df = pd.read_csv('./data/monster_com-job_sample.csv')
```

```
df.head()
```

```
                    country country_code date_added has_expired  \
0  United States of America           US        NaN          No
1  United States of America           US        NaN          No
2  United States of America           US        NaN          No
```

(continues on next page)

(continued from previous page)

```
3  United States of America           US        NaN          No
4  United States of America           US        NaN          No

          job_board                                    job_description  \
0  jobs.monster.com  TeamSoft is seeing an IT Support Specialist to...
1  jobs.monster.com  The Wisconsin State Journal is seeking a flexi...
2  jobs.monster.com  Report this job About the Job DePuy Synthes Co...
3  jobs.monster.com  Why Join Altec? If you’re considering a career...
4  jobs.monster.com  Position ID#  76162 # Positions  1 State  CT C...

                                           job_title             job_type  \
0             IT Support Technician Job in Madison   Full Time Employee
1           Business Reporter/Editor Job in Madison           Full Time
2  Johnson & Johnson Family of Companies Job Appl...  Full Time, Employee
3                     Engineer - Quality Job in Dixon           Full Time
4      Shift Supervisor - Part-Time Job in Camphill   Full Time Employee

                                            location  \
0                                  Madison, WI 53702
1                                  Madison, WI 53708
2  DePuy Synthes Companies is a member of Johnson...
3                                         Dixon, CA
4                                       Camphill, PA

                      organization  \
0                              NaN
1          Printing and Publishing
2  Personal and Household Services
3                  Altec Industries
4                           Retail

                                            page_url salary  \
0  http://jobview.monster.com/it-support-technici...    NaN
1  http://jobview.monster.com/business-reporter-e...    NaN
2  http://jobview.monster.com/senior-training-lea...    NaN
3  http://jobview.monster.com/engineer-quality-jo...    NaN
4  http://jobview.monster.com/shift-supervisor-pa...    NaN

                      sector                           uniq_id
0    IT/Software Development  11d599f229a80023d2f40e7c52cd941e
1                        NaN  e4cbb126dabf22159aff90223243ff2a
2                        NaN  839106b353877fa3d896ffb9c1fe01c0
3   Experienced (Non-Manager)  58435fcab804439efdcaa7ecca0fd783
4  Project/Program Management  64d0272dc8496abfd9523a8df63c184c
```

Need some inspiration? perhaps this might help!

Part III

3. Data Preprocessing

CHAPTER THIRTEEN

DATA PREPROCESSING

this is an introduction

CHAPTER

FOURTEEN

INDEXING AND SLICING

In

```
import pandas as pd
```

```
min_temp_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c2_data_preparation/data/temperatures/australia/
↪melbourne/1981.csv')
min_temp_df
```

```
           Date  Temp
0    1981-01-01  20.7
1    1981-01-02  17.9
2    1981-01-03  18.8
3    1981-01-04  14.6
4    1981-01-05  15.8
..          ...   ...
360  1981-12-27  15.5
361  1981-12-28  13.3
362  1981-12-29  15.6
363  1981-12-30  15.2
364  1981-12-31  17.4

[365 rows x 2 columns]
```

```
min_temp_df.Date = pd.to_datetime(min_temp_df.Date)
```

```
min_temp_df = min_temp_df.set_index('Date')
```

```
min_temp_df.loc['1981-06-01':'1981-06-30']
```

```
            Temp
Date
1981-06-01  11.6
1981-06-02  10.6
1981-06-03   9.8
1981-06-04  11.2
1981-06-05   5.7
1981-06-06   7.1
1981-06-07   2.5
1981-06-08   3.5
1981-06-09   4.6
```

(continues on next page)

(continued from previous page)

```
1981-06-10  11.0
1981-06-11   5.7
1981-06-12   7.7
1981-06-13  10.4
1981-06-14  11.4
1981-06-15   9.2
1981-06-16   6.1
1981-06-17   2.7
1981-06-18   4.3
1981-06-19   6.3
1981-06-20   3.8
1981-06-21   4.4
1981-06-22   7.1
1981-06-23   4.8
1981-06-24   5.8
1981-06-25   6.2
1981-06-26   7.3
1981-06-27   9.2
1981-06-28  10.2
1981-06-29   9.5
1981-06-30   9.5
```

```
min_temp_df.loc['1989-06-01':'1989-06-30'].mean()
```

```
Temp   NaN
dtype: float64
```

```
min_temp_df.resample('MS').mean()
```

```
                 Temp
Date
1981-01-01  17.712903
1981-02-01  17.678571
1981-03-01  13.500000
1981-04-01  12.356667
1981-05-01   9.490323
1981-06-01   7.306667
1981-07-01   7.577419
1981-08-01   7.238710
1981-09-01  10.143333
1981-10-01  10.087097
1981-11-01  11.890000
1981-12-01  13.680645
```

```
import seaborn as sns
```

```
tip_df = sns.load_dataset('tips')
tip_df.head()
```

```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
```

(continues on next page)

(continued from previous page)

```
4        24.59   3.61   Female      No   Sun   Dinner     4
```

```
tip_index_df = tip_df.set_index('day')
```

```
tip_index_df.loc['Sun']
```

```
     total_bill   tip     sex smoker    time  size
day
Sun       16.99  1.01  Female     No  Dinner     2
Sun       10.34  1.66    Male     No  Dinner     3
Sun       21.01  3.50    Male     No  Dinner     3
Sun       23.68  3.31    Male     No  Dinner     2
Sun       24.59  3.61  Female     No  Dinner     4
..          ...   ...     ...    ...     ...   ...
Sun       20.90  3.50  Female    Yes  Dinner     3
Sun       30.46  2.00    Male    Yes  Dinner     5
Sun       18.15  3.50  Female    Yes  Dinner     3
Sun       23.10  4.00    Male    Yes  Dinner     3
Sun       15.69  1.50    Male    Yes  Dinner     2

[76 rows x 6 columns]
```

```
tip_index_df = tip_df.set_index(['day','time'])
```

```
tip_index_df.loc[('Thur','Lunch')].tip.mean()
```

```
/tmp/ipykernel_25625/2537502835.py:1: PerformanceWarning: indexing past lexsort depth␣
↪may impact performance.
  tip_index_df.loc[('Thur','Lunch')].tip.mean()
```

```
2.767704918032786
```

```
pd.pivot_table(tip_df, values='total_bill', index='day', columns='time', aggfunc=
↪'median')
```

```
time  Lunch  Dinner
day
Thur  16.00  18.780
Fri   13.42  18.665
Sat     NaN  18.240
Sun     NaN  19.630
```

```
tip_df.set_index(['sex', 'time','smoker']).loc[('Male', 'Dinner','Yes')]['tip'].mean()
```

```
/tmp/ipykernel_25625/3467525553.py:1: PerformanceWarning: indexing past lexsort depth␣
↪may impact performance.
  tip_df.set_index(['sex', 'time','smoker']).loc[('Male', 'Dinner','Yes')]['tip'].
↪mean()
```

```
3.123191489361702
```

CHAPTER

FIFTEEN

MERGE

When data becomes multi-dimensional - covering multiple aspects of information - it usually happens that a lot of information is redundant. Take for example the next dataset, we have collected ratings of restaurants from users, when a single user rates 2 restaurants the information of the user relates to both rows, yet it would be wasteful to keep this info twice. The same can happen when we have a restaurant with 2 ratings, the location of the restaurant is kept twice in our data, which is not scalable.

We solve this problem using relational data, the idea is that we have a common key column in 2 of our tables which we can use to join the data for further processing.

In our example we use a dataset with consumers, restaurants and ratings between those, you can find more information here.

```
import pandas as pd
```

```
rating_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c3_data_preprocessing/data/cuisine/rating_final.csv')
rating_df
```

```
     userID  placeID  rating  food_rating  service_rating
0     U1077   135085       2            2               2
1     U1077   135038       2            2               1
2     U1077   132825       2            2               2
3     U1077   135060       1            2               2
4     U1068   135104       1            1               2
...     ...      ...     ...          ...             ...
1156  U1043   132630       1            1               1
1157  U1011   132715       1            1               0
1158  U1068   132733       1            1               0
1159  U1068   132594       1            1               1
1160  U1068   132660       0            0               0

[1161 rows x 5 columns]
```

this first table we read contains the userID from whom the rating came, the placeID is the restaurant he/she rated and the numerical values of the 3 different ratings.

Perhaps you can find out what the min and max values for the ratings are?

to know the type of restaurant, we can not read another table

```
cuisine_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c3_data_preprocessing/data/cuisine/chefmozcuisine.csv')
cuisine_df
```

```
     placeID         Rcuisine
0     135110          Spanish
1     135109          Italian
2     135107   Latin_American
3     135106          Mexican
4     135105        Fast_Food
..       ...              ...
911   132005          Seafood
912   132004          Seafood
913   132003    International
914   132002          Seafood
915   132001     Dutch-Belgian

[916 rows x 2 columns]
```

This table also contains the placeID, so we should be able to merge/join these 2 tables and create a new table with info of both. Notice how we specify the 'on' parameter where we denote placeID as our common key.

```
merged_df = pd.merge(rating_df, cuisine_df, on='placeID', how='inner')
merged_df
```

```
     userID  placeID  rating  food_rating  service_rating   Rcuisine
0     U1077   135085       2            2               2  Fast_Food
1     U1108   135085       1            2               1  Fast_Food
2     U1081   135085       1            2               1  Fast_Food
3     U1056   135085       2            2               2  Fast_Food
4     U1134   135085       2            1               2  Fast_Food
...     ...      ...     ...          ...             ...        ...
1038  U1061   132958       2            2               2   American
1039  U1025   132958       1            0               0   American
1040  U1097   132958       2            1               1   American
1041  U1096   132958       1            2               2   American
1042  U1136   132958       2            2               2   American

[1043 rows x 6 columns]
```

Great! now we have more info about the rating that were given, being the type of cuisine that they rated. We could figure out which cuisines are available in our dataset and do a comparison, let us count the occurences of each cuisine.

```
merged_df.Rcuisine.value_counts()
```

```
Mexican              238
Bar                  140
Cafeteria            102
Fast_Food             91
Seafood               62
Bar_Pub_Brewery       59
Pizzeria              51
Chinese               41
American              39
International         37
Contemporary          32
Burgers               31
Japanese              29
Italian               26
Family                14
```

(continues on next page)

(continued from previous page)

```
Cafe-Coffee_Shop     12
Breakfast-Brunch      9
Game                  7
Vietnamese            6
Bakery                5
Mediterranean         4
Armenian              4
Regional              4
Name: Rcuisine, dtype: int64
```

A lot of mexican, which is not surpising as this dataset comes from Mexico. I wonder if there is a difference between 'Bar' and 'Bar_Pub_Brewery', we can see if the average rating for those 2 differ.

```
for cuisine in ['Bar', 'Bar_Pub_Brewery']:
    print(cuisine)
    print(merged_df[merged_df.Rcuisine==cuisine][['rating', 'food_rating', 'service_
↪rating']].mean())
    print()
```

```
Bar
rating            1.200000
food_rating       1.135714
service_rating    1.085714
dtype: float64

Bar_Pub_Brewery
rating            1.305085
food_rating       1.169492
service_rating    1.203390
dtype: float64
```

just looking at the averages we can deduces that while food ratings do not change a lot, the service seems a lot better at the Brewery.

```
merged_df[merged_df.Rcuisine=='Cafeteria'][['rating', 'food_rating', 'service_rating
↪']].mean()
```

```
rating            1.205882
food_rating       1.127451
service_rating    1.078431
dtype: float64
```

```
merged_df[merged_df.Rcuisine=='Cafe-Coffee_Shop'][['rating', 'food_rating', 'service_
↪rating']].mean()
```

```
rating            1.583333
food_rating       1.333333
service_rating    1.416667
dtype: float64
```

As easy as it looks, we can now merge information of different tables in our dataset and perform some simple comparisons, in later sections we will see how we can improve on those.

As an exercise I already read in the table containing the info about which type of payment the user has opted for. Could you find out if the type of payment could have an influence on the rating?

```
user_payment_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c3_data_preprocessing/data/cuisine/userpayment.csv')
user_payment_df
```

```
    userID          Upayment
0    U1001              cash
1    U1002              cash
2    U1003              cash
3    U1004              cash
4    U1004  bank_debit_cards
..     ...               ...
172  U1134              cash
173  U1135              cash
174  U1136              cash
175  U1137              cash
176  U1138              cash

[177 rows x 2 columns]
```

CHAPTER

SIXTEEN

GROUPBY

In the previous section we saw how to combine information of multiple tables from our dataset. Here we are going to build further on that by using the merged information to group on categorical variables.

```
import pandas as pd
```

```
rating_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c3_data_preprocessing/data/cuisine/rating_final.csv')
rating_df
```

```
     userID  placeID  rating  food_rating  service_rating
0     U1077   135085       2            2               2
1     U1077   135038       2            2               1
2     U1077   132825       2            2               2
3     U1077   135060       1            2               2
4     U1068   135104       1            1               2
...     ...      ...     ...          ...             ...
1156  U1043   132630       1            1               1
1157  U1011   132715       1            1               0
1158  U1068   132733       1            1               0
1159  U1068   132594       1            1               1
1160  U1068   132660       0            0               0

[1161 rows x 5 columns]
```

Again we have our rating data containing the users, places and ratings they gave. As a simple example we could just group by the placeID column and take the mean, this would give us the mean rating for each restaurant

```
grouped_rating_df = rating_df.groupby('placeID').mean().sort_values('rating')
grouped_rating_df
```

```
           rating  food_rating  service_rating
placeID
132654   0.250000         0.25        0.250000
135040   0.250000         0.25        0.250000
132560   0.500000         1.00        0.250000
132663   0.500000         0.50        0.666667
135069   0.500000         0.50        0.750000
...           ...          ...             ...
132755   1.800000         2.00        1.600000
132922   1.833333         1.50        1.833333
134986   2.000000         2.00        2.000000
135034   2.000000         2.00        1.600000
```

(continues on next page)

(continued from previous page)

```
132955   2.000000         1.80        1.800000

[130 rows x 3 columns]
```

Keep in mind that this might be tricky, as we do not always have as much records per group, we could count the amount per records using a groupby operation and count.

```
rating_df.groupby('placeID').rating.count()
```

```
placeID
132560     4
132561     4
132564     4
132572    15
132583     4
          ..
135088     6
135104     7
135106    10
135108    11
135109     4
Name: rating, Length: 130, dtype: int64
```

Taking an average of 4 ratings might not be ideal, so we should keep in mind that our groups have a good sample size.

Let's make things more interesting and insert some location data.

```
geo_df = pd.read_csv('./data/cuisine/geoplaces2.csv').set_index('placeID')
geo_df
```

```
          latitude   longitude  \
placeID
134999   18.915421  -99.184871
132825   22.147392 -100.983092
135106   22.149709 -100.976093
132667   23.752697  -99.163359
132613   23.752903  -99.165076
...            ...         ...
132866   22.141220 -100.931311
135072   22.149192 -101.002936
135109   18.921785  -99.235350
135019   18.875011  -99.159422
132877   22.135364 -100.934948

                                            the_geom_meter  \
placeID
134999   0101000020957F000088568DE356715AC138C0A525FC46...
132825   0101000020957F00001AD016568C4858C1243261274BA5...
135106   0101000020957F0000649D6F21634858C119AE9BF528A3...
132667   0101000020957F00005D67BCDDED8157C1222A2DC8D84D...
132613   0101000020957F00008EBA2D06DC8157C194E03B7B504E...
...                                                    ...
132866   0101000020957F000013871838EC4A58C1B5DF74F8E396...
135072   0101000020957F0000E7B79B1DB94758C1D29BC363D8AA...
135109   0101000020957F0000A6BF695F136F5AC1DADF87B20556...
135019   0101000020957F0000B49B2E5C6E785AC12F9D58435241...
```

(continues on next page)

(continued from previous page)

```
132877   0101000020957F000090735015B84B58C1AF0DC0414698...

                                   name  \
placeID
134999                  Kiku Cuernavaca
132825                  puesto de tacos
135106       El Rinc�n de San Francisco
132667   little pizza Emilio Portes Gil
132613                     carnitas_mata
...                                 ...
132866                          Chaires
135072                       Sushi Itto
135109                        Paniroles
135019      Restaurant Bar Coty y Pablo
132877                  sirloin stockade

                                                address             city  \
placeID
134999                                       Revolucion       Cuernavaca
132825             esquina santos degollado y leon guzman           s.l.p.
135106                                  Universidad 169  San Luis Potosi
132667                           calle emilio portes gil        victoria
132613                            lic. Emilio portes gil         victoria
...                                                 ...              ...
132866                                 Ricardo B. Anaya  San Luis Potosi
135072               Venustiano Carranza 1809 C Polanco  San Luis Potosi
135109                                                ?                ?
135019   Paseo de Las Fuentes 24 Pedregal de Las Fuentes         Jiutepec
132877                                                ?                ?

                   state country fax    zip           alcohol   smoking_area  \
placeID
134999           Morelos  Mexico   ?      ?  No_Alcohol_Served           none
132825            s.l.p.  mexico   ?  78280  No_Alcohol_Served           none
135106   San Luis Potosi  Mexico   ?  78000          Wine-Beer    only at bar
132667        tamaulipas       ?   ?      ?  No_Alcohol_Served           none
132613        Tamaulipas  Mexico   ?      ?  No_Alcohol_Served      permitted
...                  ...     ...  ..    ...                ...            ...
132866   San Luis Potosi  Mexico   ?      ?  No_Alcohol_Served  not permitted
135072               SLP  Mexico   ?  78220  No_Alcohol_Served           none
135109                 ?       ?   ?      ?          Wine-Beer  not permitted
135019           Morelos  Mexico   ?      ?  No_Alcohol_Served           none
132877                 ?       ?   ?      ?  No_Alcohol_Served           none

        dress_code     accessibility   price                    url Rambience  \
placeID
134999    informal  no_accessibility  medium  kikucuernavaca.com.mx  familiar
132825    informal        completely     low                      ?  familiar
135106    informal         partially  medium                      ?  familiar
132667    informal        completely     low                      ?  familiar
132613    informal        completely  medium                      ?  familiar
...            ...               ...     ...                    ...       ...
132866    informal        completely  medium                      ?  familiar
135072    informal  no_accessibility  medium     sushi-itto.com.mx  familiar
135109    informal  no_accessibility  medium                      ?     quiet
135019    informal        completely     low                      ?  familiar
132877    informal        completely     low                      ?  familiar
```

(continues on next page)

(continued from previous page)

```
       franchise    area other_services
placeID
134999         f  closed           none
132825         f    open           none
135106         f    open           none
132667         t  closed           none
132613         t  closed           none
...          ...     ...            ...
132866         f  closed           none
135072         f  closed           none
135109         f  closed       Internet
135019         f  closed           none
132877         f  closed           none

[130 rows x 20 columns]
```

Here we have for each restaurant information about its location, I mentioned earlier that grouping per restaurant might be dangerous as some restaurants have nearly no reviews. By adding information such as city, state and country we have other categorical variables to group by. Notice how we use the merge operation from previous section, but this time specify our common key is the index.

```
geo_rating_df = pd.merge(grouped_rating_df, geo_df, left_index=True, right_index=True)
geo_rating_df
```

```
           rating  food_rating  service_rating   latitude   longitude  \
placeID
132654   0.250000         0.25        0.250000  23.735523  -99.129588
135040   0.250000         0.25        0.250000  22.135617 -100.969709
132560   0.500000         1.00        0.250000  23.752304  -99.166913
132663   0.500000         0.50        0.666667  23.752511  -99.166954
135069   0.500000         0.50        0.750000  22.140129 -100.944872
...           ...          ...             ...        ...         ...
132755   1.800000         2.00        1.600000  22.153324 -101.019546
132922   1.833333         1.50        1.833333  22.151135 -100.982311
134986   2.000000         2.00        2.000000  18.928798  -99.239513
135034   2.000000         2.00        1.600000  22.140517 -101.021422
132955   2.000000         1.80        1.800000  22.147622 -101.010275

                                            the_geom_meter  \
placeID
132654   0101000020957F000040E8F628488557C18224E8B94845...
135040   0101000020957F00001B552189B84A58C15A2AAEFD2CA2...
132560   0101000020957F0000FC60BDA8E88157C1B2C357D6DA4E...
132663   0101000020957F0000FDF8D26EE08157C1FEDB6A1FDB4E...
135069   0101000020957F000038E5D546B74A58C18FD29AD0D29A...
...                                                    ...
132755   0101000020957F000026CADE45A14658C1F011EBCA55AF...
132922   0101000020957F000060A98A38FF4758C146718E41D9A4...
134986   0101000020957F00002A0D05E2D96D5AC1AB058CB1EC56...
135034   0101000020957F000026D92BB4894858C161A7552DA2B0...
132955   0101000020957F000068BE7C87C24758C1920A360A08AD...

                                          name  \
placeID
132654   Carnitas Mata  Calle 16 de Septiembre
```

(continues on next page)

(continued from previous page)

```
135040               Restaurant los Compadres
132560                     puesto de gorditas
132663                              tacos abi
135069              Abondance Restaurante Bar
...                                       ...
132755                    La Estrella de Dimas
132922                    cafe punta del cielo
134986                Restaurant Las Mananitas
135034              Michiko Restaurant Japones
132955                              emilianos

                                        address            city  \
placeID
132654                       16 de Septiembre        victoria
135040            Camino a Simon Diaz 155 Centro  San Luis Potosi
132560                     frente al tecnologico         victoria
132663                                         ?         victoria
135069               Industrias 908 Valle Dorado  San Luis Potosi
...                                          ...              ...
132755                      Av. de los Pintores  San Luis Potosi
132922                                         ?                ?
134986                      Ricardo Linares 107       Cuernavaca
135034   Cordillera de Los Alpes 160 Lomas 2 Seccion  San Luis Potosi
132955                      venustiano carranza   san luis potos

              state  ...           alcohol smoking_area dress_code  \
placeID              ...
132654   tamaulipas  ...  No_Alcohol_Served         none   informal
135040          SLP  ...          Wine-Beer         none   informal
132560   tamaulipas  ...  No_Alcohol_Served    permitted   informal
132663   tamaulipas  ...  No_Alcohol_Served         none   informal
135069          SLP  ...          Wine-Beer         none   informal
...             ...  ...                ...          ...        ...
132755       S.L.P.  ...  No_Alcohol_Served         none   informal
132922            ?  ...  No_Alcohol_Served    permitted     formal
134986      Morelos  ...          Wine-Beer         none     formal
135034          SLP  ...  No_Alcohol_Served         none   informal
132955       mexico  ...          Wine-Beer         none   informal

           accessibility   price                  url Rambience franchise  \
placeID
132654        completely     low                    ?  familiar         f
135040   no_accessibility    high                    ?  familiar         f
132560   no_accessibility     low                    ?  familiar         f
132663        completely     low                    ?  familiar         f
135069   no_accessibility     low                    ?  familiar         f
...                   ...     ...                  ...       ...       ...
132755         partially  medium                    ?  familiar         f
132922        completely  medium                    ?  familiar         f
134986   no_accessibility    high  lasmananitas.com.mx  familiar         f
135034   no_accessibility  medium                    ?  familiar         f
132955        completely     low                    ?  familiar         t

           area other_services
placeID
132654   closed           none
135040   closed           none
```

(continues on next page)

(continued from previous page)

```
132560     open          none
132663   closed          none
135069   closed          none
...         ...           ...
132755   closed       variety
132922   closed          none
134986   closed          none
135034   closed          none
132955   closed       variety

[130 rows x 23 columns]
```

By adding this amount of data, things are getting a bit cluttered, thankfully we can use pandas to get a list of all our columns.

```
geo_rating_df.columns
```

```
Index(['rating', 'food_rating', 'service_rating', 'latitude', 'longitude',
       'the_geom_meter', 'name', 'address', 'city', 'state', 'country', 'fax',
       'zip', 'alcohol', 'smoking_area', 'dress_code', 'accessibility',
       'price', 'url', 'Rambience', 'franchise', 'area', 'other_services'],
      dtype='object')
```

How about we try and see if we can find a difference between countries for the ratings?

```
geo_rating_df.groupby('country')[['rating', 'food_rating', 'service_rating']].mean()
```

```
           rating  food_rating  service_rating
country
?        1.166045     1.232946        1.069169
Mexico   1.200977     1.229093        1.118162
mexico   1.062660     1.069006        0.900064
```

Ah, it seems we forgot to do some data cleaning here, perhaps you could jump in and fix this string problem, might as well tackle the missing value while we are at it. Aside from that, we can see that lower-case Mexico is not doing very well, perhaps the food was so bad they forgot how to write Mexico?

Jokes aside, do you see the ressemblance between this and our rudimentary approach of comparing different categories? We are slowly getting more and more efficient using these operations, how about the difference between alcohol consumption?

```
geo_rating_df.groupby('alcohol')[['rating', 'food_rating', 'service_rating']].mean()
```

```
                     rating  food_rating  service_rating
alcohol
Full_Bar           1.287124     1.218315        1.170311
No_Alcohol_Served  1.148075     1.194730        1.042417
Wine-Beer          1.231887     1.261840        1.174437
```

Something we can remark here is that the food rating for no alcohol locations seems to be holding up, whilst the general rating and service rating fall behind. This would suggest that the food rating indeed is for the food, where the type of drinks served have no influence.

As a last we look at the difference between accessibility, does that influences our ratings?

```
geo_rating_df.groupby('accessibility')[['rating', 'food_rating', 'service_rating']].
↪mean()
```

```
                    rating  food_rating  service_rating
accessibility
completely        1.132494     1.203597        1.049709
no_accessibility  1.196189     1.206242        1.091278
partially         1.275356     1.330294        1.219991
```

It seems having partial accessibility is the way to go here, performing better than complete accessibility. We can however find that is due to a low sample size of 9 restaurants, making it prone to variation.

```
geo_rating_df.accessibility.value_counts()
```

```
no_accessibility    76
completely          45
partially            9
Name: accessibility, dtype: int64
```

You should get the hang of it by now, perhaps you can play some more with the other categories.

There is one thing I still would like to address, you perhaps have notices that in the beginning I first took the average rating per restaurant and later again took the average per category. This is a bad practice as a bad restaurant with one review has equal influence as a good restaurant with 100 reviews, perhaps you can think of a way to group all reviews from a category instead of the average for each restaurant?

In the previous section we added the cuisine type, perhaps you could do some groupby operations on that too here?

CHAPTER SEVENTEEN

PIVOT

When using the groupby operation we used 1 categorical variable to seperate/group our data into those categories. Here we go a step further and use 2 categories to aggregate our data, resulting in a comparison matrix.

Aside from that, the pivot operation can in general be used to go from a long data format, to a wide data format. To keep things uniform we stick with the same cuisine dataset.

```
import pandas as pd
```

```
rating_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c3_data_preprocessing/data/cuisine/rating_final.csv')
rating_df
```

```
     userID  placeID  rating  food_rating  service_rating
0     U1077   135085       2            2               2
1     U1077   135038       2            2               1
2     U1077   132825       2            2               2
3     U1077   135060       1            2               2
4     U1068   135104       1            1               2
...     ...      ...     ...          ...             ...
1156  U1043   132630       1            1               1
1157  U1011   132715       1            1               0
1158  U1068   132733       1            1               0
1159  U1068   132594       1            1               1
1160  U1068   132660       0            0               0

[1161 rows x 5 columns]
```

And again we merge with the geolocations data, I feel that it becomse obvious here how these operations are very related to eachother.

```
geo_df = pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c3_data_preprocessing/data/cuisine/geoplaces2.csv')
```

A subtle difference between last time is that I did not first group per restaurant, however this leads to a dataframe that has a lot of redundant information! Try to look in the merged dataframe and spot the copies of data.

```
geo_rating_df = pd.merge(rating_df, geo_df, on='placeID')
geo_rating_df
```

```
     userID  placeID  rating  food_rating  service_rating   latitude  \
0     U1077   135085       2            2               2  22.150802
1     U1108   135085       1            2               1  22.150802
2     U1081   135085       1            2               1  22.150802
```

(continues on next page)

(continued from previous page)

```
3     U1056   135085       2           2            2  22.150802
4     U1134   135085       2           1            2  22.150802
...     ...      ...     ...         ...          ...        ...
1156  U1061   132958       2           2            2  22.144979
1157  U1025   132958       1           0            0  22.144979
1158  U1097   132958       2           1            1  22.144979
1159  U1096   132958       1           2            2  22.144979
1160  U1136   132958       2           2            2  22.144979

       longitude                                   the_geom_meter  \
0    -100.982680  0101000020957F00009F823DA6094858C18A2D4D37F9A4...
1    -100.982680  0101000020957F00009F823DA6094858C18A2D4D37F9A4...
2    -100.982680  0101000020957F00009F823DA6094858C18A2D4D37F9A4...
3    -100.982680  0101000020957F00009F823DA6094858C18A2D4D37F9A4...
4    -100.982680  0101000020957F00009F823DA6094858C18A2D4D37F9A4...
...          ...                                              ...
1156 -101.005683  0101000020957F000049095EB34A4858C15CB4BD1EE1AB...
1157 -101.005683  0101000020957F000049095EB34A4858C15CB4BD1EE1AB...
1158 -101.005683  0101000020957F000049095EB34A4858C15CB4BD1EE1AB...
1159 -101.005683  0101000020957F000049095EB34A4858C15CB4BD1EE1AB...
1160 -101.005683  0101000020957F000049095EB34A4858C15CB4BD1EE1AB...

                        name                         address  ...  \
0     Tortas Locas Hipocampo  Venustiano Carranza 719 Centro  ...
1     Tortas Locas Hipocampo  Venustiano Carranza 719 Centro  ...
2     Tortas Locas Hipocampo  Venustiano Carranza 719 Centro  ...
3     Tortas Locas Hipocampo  Venustiano Carranza 719 Centro  ...
4     Tortas Locas Hipocampo  Venustiano Carranza 719 Centro  ...
...                      ...                             ...  ...
1156      tacos los volcanes          avenida hivno nacional  ...
1157      tacos los volcanes          avenida hivno nacional  ...
1158      tacos los volcanes          avenida hivno nacional  ...
1159      tacos los volcanes          avenida hivno nacional  ...
1160      tacos los volcanes          avenida hivno nacional  ...

                alcohol   smoking_area dress_code     accessibility   price  \
0     No_Alcohol_Served  not permitted   informal  no_accessibility  medium
1     No_Alcohol_Served  not permitted   informal  no_accessibility  medium
2     No_Alcohol_Served  not permitted   informal  no_accessibility  medium
3     No_Alcohol_Served  not permitted   informal  no_accessibility  medium
4     No_Alcohol_Served  not permitted   informal  no_accessibility  medium
...                 ...            ...        ...               ...     ...
1156  No_Alcohol_Served           none   informal        completely     low
1157  No_Alcohol_Served           none   informal        completely     low
1158  No_Alcohol_Served           none   informal        completely     low
1159  No_Alcohol_Served           none   informal        completely     low
1160  No_Alcohol_Served           none   informal        completely     low

     url Rambience franchise    area other_services
0      ?  familiar         f  closed           none
1      ?  familiar         f  closed           none
2      ?  familiar         f  closed           none
3      ?  familiar         f  closed           none
4      ?  familiar         f  closed           none
...   ..       ...       ...     ...            ...
1156   ?     quiet         t  closed           none
1157   ?     quiet         t  closed           none
```

(continues on next page)

(continued from previous page)

```
1158   ?     quiet          t  closed           none
1159   ?     quiet          t  closed           none
1160   ?     quiet          t  closed           none

[1161 rows x 25 columns]
```

Now that we have our workable data, we can choose 2 categories and create a comparison matrix using the pivot operation. Yet there might be a problem that we still have to resolve, can you figure out the problem reading the error at the end of the stack trace below?

```
geo_rating_df.pivot(index='alcohol', columns='smoking_area', values='rating')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_20513/1351770208.py in <module>
----> 1 geo_rating_df.pivot(index='alcohol', columns='smoking_area', values='rating')

~/git/data-science-practical-approach/venv/lib/python3.8/site-packages/pandas/core/
↪frame.py in pivot(self, index, columns, values)
   7791         from pandas.core.reshape.pivot import pivot
   7792
-> 7793         return pivot(self, index=index, columns=columns, values=values)
   7794
   7795     _shared_docs[

~/git/data-science-practical-approach/venv/lib/python3.8/site-packages/pandas/core/
↪reshape/pivot.py in pivot(data, index, columns, values)
    515         else:
    516             indexed = data._constructor_sliced(data[values]._values,␣
↪index=multiindex)
--> 517     return indexed.unstack(columns_listlike)
    518
    519

~/git/data-science-practical-approach/venv/lib/python3.8/site-packages/pandas/core/
↪series.py in unstack(self, level, fill_value)
   4079         from pandas.core.reshape.reshape import unstack
   4080
-> 4081         return unstack(self, level, fill_value)
   4082
   4083     # ----------------------------------------------------------------------

~/git/data-science-practical-approach/venv/lib/python3.8/site-packages/pandas/core/
↪reshape/reshape.py in unstack(obj, level, fill_value)
    458         if is_1d_only_ea_dtype(obj.dtype):
    459             return _unstack_extension_series(obj, level, fill_value)
--> 460         unstacker = _Unstacker(
    461             obj.index, level=level, constructor=obj._constructor_expanddim
    462         )

~/git/data-science-practical-approach/venv/lib/python3.8/site-packages/pandas/core/
↪reshape/reshape.py in __init__(self, index, level, constructor)
    131             raise ValueError("Unstacked DataFrame is too big, causing int32␣
↪overflow")
    132
--> 133         self._make_selectors()
```

(continues on next page)

(continued from previous page)

```
    134
    135     @cache_readonly

~/git/data-science-practical-approach/venv/lib/python3.8/site-packages/pandas/core/
↪reshape/reshape.py in _make_selectors(self)
    183
    184         if mask.sum() < len(self.index):
--> 185             raise ValueError("Index contains duplicate entries, cannot reshape
↪")
    186
    187         self.group_index = comp_index

ValueError: Index contains duplicate entries, cannot reshape
```

It says: 'Index contains duplicate entries, cannot reshape' meaning that some combinations of our 2 categories, alcohol and smoking area have duplicates, which is understandable. I opted to solve this by grouping over the 2 categories and taking the mean for each combination, then i take this grouped data and pivot by setting the alcohol consumption as index and the smoking are as columns.

```
grouped_geo_rating_df = geo_rating_df.groupby(['alcohol','smoking_area'])[['rating',
↪'food_rating', 'service_rating']].mean().reset_index()
grouped_geo_rating_df.pivot(index='alcohol', columns='smoking_area', values='rating')
```

```
smoking_area             none  not permitted  only at bar  permitted   section
alcohol
Full_Bar             1.305556       0.857143          NaN   1.500000  1.272727
No_Alcohol_Served    1.186788       1.124402          NaN   1.114286  1.265823
Wine-Beer            1.217391       1.000000     1.368421   1.300000  1.275000
```

Wonderful! Now we have for each combination an average rating, notice however that not every combination has the same sample size, so comparing might be tricky if you only have a few ratings.

To figure that out I counted the ratings per combination.

```
geo_rating_df.groupby(['alcohol','smoking_area']).count().reset_index().pivot(index=
↪'alcohol', columns='smoking_area', values='rating')
```

```
smoking_area          none  not permitted  only at bar  permitted  section
alcohol
Full_Bar              36.0            7.0          NaN        4.0     33.0
No_Alcohol_Served    439.0          209.0          NaN       35.0     79.0
Wine-Beer            161.0            9.0         19.0       10.0    120.0
```

It seems that there might e a correlation between the 2 categories, as a lot of place where smoking is not permitted/none, there is no alcohol served, which makes sense. Comparing the ratings with alcohol allowance for places where smoking is not permitted is not a good idea, the counts are 7, 209 and 9, very unbalanced.

```
geo_df.columns
```

```
Index(['placeID', 'latitude', 'longitude', 'the_geom_meter', 'name', 'address',
       'city', 'state', 'country', 'fax', 'zip', 'alcohol', 'smoking_area',
       'dress_code', 'accessibility', 'price', 'url', 'Rambience', 'franchise',
       'area', 'other_services'],
      dtype='object')
```

I printed the columns above, perhaps you could figure out a relation between the price category and the (R)ambience of the restaurant? Perhaps there are other combinations of which I did not think of, try some out!

CHAPTER

EIGHTEEN

USING SQL

In this notebook we are going to do things different, instead of using python and pandas for data wrangling/processing we outsource them to the SQL, a language used for databases.

As it would be complicated to setup a complete SQL server, I opted to create a local database using SQLite which is built-in the sqlalchemy library used by python to interact with a database.

We start by importing or necessary libraries

```
import pandas as pd
import sqlalchemy
```

As mentioned we are going to create a local SQL database and dump it to a .db file. In order to do that we first have to read our data from comma seperated value (CSV) files that were provided within the repository.

We use pandas to read them and collect them into an object data

```
data = {
    'ratings': pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c3_data_preprocessing/data/cuisine/rating_final.csv'),
    'cuisine': pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c3_data_preprocessing/data/cuisine/chefmozcuisine.csv'),
    'parking': pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-science-
↪practical-approach/main/src/c3_data_preprocessing/data/cuisine/chefmozparking.csv'),
    'user_cuisine': pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-
↪science-practical-approach/main/src/c3_data_preprocessing/data/cuisine/usercuisine.
↪csv'),
    'user_payment': pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-
↪science-practical-approach/main/src/c3_data_preprocessing/data/cuisine/userpayment.
↪csv'),
    'user_profile': pd.read_csv('https://raw.githubusercontent.com/LorenzF/data-
↪science-practical-approach/main/src/c3_data_preprocessing/data/cuisine/userprofile.
↪csv', na_values='?'),
}
```

Before we can act with our database we need to create an engine by setting up the connection. In a more complex situation this would need an url to the server running the database, a userid and password for loging and some other configurations.

In this case we only need the location of our .db file, which i will put in the same location as the notebook.

```
engine = sqlalchemy.create_engine('sqlite:///ratings.db')
```

Great! we now have an engine that can run our SQL queries, yet for now our database is empty, let us fill it with all the data we collected earlier!

We use the .to_sql method of pandas to easily convert the pandas dataframe to a table in our database, each name in our data object will be a table with the corresponding data.

```
for table_name, df in data.items():
    df.to_sql(table_name, engine, if_exists='replace')
```

And with this our migration to SQL has been completed, we now have a SQL server running locally that has several tables containing data. Instead of using python to do the processing we can instruct our server to handle this, usually resulting is faster compute times, yet results may vary!

Let's start with a simple example, I saw that we have a table with ratings, to see how it looks by selecting all columns.

```
df = pd.read_sql(
    """
    SELECT * FROM ratings
    """,
    engine
)
df
```

```
      index userID  placeID  rating  food_rating  service_rating
0         0  U1077   135085       2            2               2
1         1  U1077   135038       2            2               1
2         2  U1077   132825       2            2               2
3         3  U1077   135060       1            2               2
4         4  U1068   135104       1            1               2
...     ...    ...      ...     ...          ...             ...
1156   1156  U1043   132630       1            1               1
1157   1157  U1011   132715       1            1               0
1158   1158  U1068   132733       1            1               0
1159   1159  U1068   132594       1            1               1
1160   1160  U1068   132660       0            0               0

[1161 rows x 6 columns]
```

It looks that an index has been copied too, we skipped preparation and it already shows. For now we are going to ignore these steps yet we should clean that later. If you want to save some time, you can LIMIT your search to a number of rows, next I put a limit of 5 to only retrieve the first 5 results

```
df = pd.read_sql(
    """
    SELECT * FROM ratings
    LIMIT 5
    """,
    engine
)
df
```

```
   index userID  placeID  rating  food_rating  service_rating
0      0  U1077   135085       2            2               2
1      1  U1077   135038       2            2               1
2      2  U1077   132825       2            2               2
3      3  U1077   135060       1            2               2
4      4  U1068   135104       1            1               2
```

Great! Here it does not matter as our database is local and not at all large in size, but this trick might save you a lot of time when exploring.

Next we would like to only select specific columns, by changing the asterisk to the wanted columns the server knowns which columns to retrieve.

```
df = pd.read_sql(
    """
    SELECT userID, rating FROM ratings
    """,
    engine
)
df.head()
```

```
  userID  rating
0  U1077       2
1  U1077       2
2  U1077       2
3  U1077       1
4  U1068       1
```

Aside from less traffic, this tidies up your data as usually most columns are not needed.

Just like columns, entries can also be filtered, in the next example we use an equation to filter only the ratings with a general rating of 2.

```
df = pd.read_sql(
    """
    SELECT userID, rating FROM ratings
    WHERE ratings.rating = 2
    """,
    engine
)
df.head()
```

```
  userID  rating
0  U1077       2
1  U1077       2
2  U1077       2
3  U1067       2
4  U1103       2
```

Similarly you can also filter based on text fields, for this example I retrieve data from another table, cuisine. No particular columns are selected yet we want to only retrieve the entries where the column Rcuisine contains a text ending on 'food' the percent sign is a wildcard indicating that any text can be present here.

```
df = pd.read_sql(
    """
    SELECT * FROM cuisine
    WHERE Rcuisine LIKE '%food'
    """,
    engine
)
df.head()
```

```
   index  placeID   Rcuisine
0      4   135105  Fast_Food
1      8   135103  Fast_Food
2     40   135089    Seafood
3     43   135086  Fast_Food
4     44   135085  Fast_Food
```

It looks that the server has found 2 types of entries that satisfy my filter, both 'Fast_Food' and 'Seafood' were results as they both end in 'food', the percent sign in this case filled for 'Fast_' and 'Sea'.

A third method of filtering entries can be a range of numbers, using the BETWEEN and AND statements.

```
df = pd.read_sql(
    """
    SELECT userID, placeID, rating FROM ratings
    WHERE placeID BETWEEN 132000 AND 135000
    """,
    engine
)
df.head()
```

```
  userID  placeID  rating
0  U1077   132825       2
1  U1068   132740       0
2  U1068   132663       1
3  U1068   132732       0
4  U1068   132630       1
```

Another method would be to use the IN statement and supply a list/tuple of possible entries, in the example we filter on 2 users that placed ratings.

```
df = pd.read_sql(
    """
    SELECT userID, placeID, rating FROM ratings
    WHERE userID IN ('U1077', 'U1103')
    """,
    engine
)
df.head()
```

```
  userID  placeID  rating
0  U1077   135085       2
1  U1077   135038       2
2  U1077   132825       2
3  U1077   135060       1
4  U1103   132584       1
```

It is also possible to filter on NULL values (NaN or missing values in SQL), this way we can easily see we again forgot to do our data preparation.

```
df = pd.read_sql(
    """
    SELECT * FROM user_profile
    WHERE smoker is NULL
    """,
    engine
)
df
```

```
   index userID   latitude   longitude smoker drink_level dress_preference  \
0     23  U1024  22.154021 -100.976028   None  abstemious             None
1    121  U1122  22.169601 -100.991821   None  abstemious             None
2    129  U1130  23.733000  -99.133000   None  abstemious             None

  ambience transport marital_status hijos  birth_year interest  personality  \
```

(continues on next page)

(continued from previous page)

```
0      None       None           None   None        1930     none   hard-worker
1      None       None           None   None        1930     none   hard-worker
2      None       None           None   None        1989     none   hard-worker

  religion activity   color  weight budget   height
0      none      None  yellow      40    None      1.2
1      none      None  yellow      40    None      1.2
2      none      None  yellow      40    None      1.2
```

We can quickly fix this by just removing all users that have missing values for smoker, as there are only 3. The syntax is a bit different as we are not using pandas, but the idea is the same, we just dont parse the result into pandas.

```
conn = engine.connect()
conn.execute(
    """
    DELETE FROM user_profile
    WHERE smoker is NULL
    """
)
```

```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x7fc53609e1c0>
```

Before we check if they are removed, think about the impact of removing users, do you think we can just do this without consequences? what about the ratings they gave? Perhaps you could remove them too here? Is it still possible?

We do a quick check to see if the users with missing values are gone.

```
df = pd.read_sql(
    """
    SELECT * FROM user_profile
    WHERE smoker is NULL
    """,
    engine
)
df
```

```
Empty DataFrame
Columns: [index, userID, latitude, longitude, smoker, drink_level, dress_preference,␣
↪ambience, transport, marital_status, hijos, birth_year, interest, personality,␣
↪religion, activity, color, weight, budget, height]
Index: []
```

Thus far we used 2 tables, ratings and cuisine, yet always seperate. Here we combine the information of both by joining them on a common column; the placeID.

Using the JOIN keyword together with the ON keyword we here perform an inner join.

```
df = pd.read_sql(
    """
    SELECT ratings.placeID, cuisine.Rcuisine, ratings.rating
    FROM ratings JOIN cuisine
    ON ratings.placeID == cuisine.placeID
    """,
    engine
)
df.head()
```

```
   placeID   Rcuisine  rating
0   135085  Fast_Food       2
1   132825    Mexican       2
2   135060    Seafood       1
3   135104    Mexican       1
4   132740    Mexican       0
```

Now we can see per rating, not only which placeID is related but also the cuisine of that place. This way we can create new views on our data without having overly complicated structures with redundant data.

Next to joining we can also aggregate data, here I created a query that counts the ratings in the ratings table, giving us the total amount of ratings.

```
count_df = pd.read_sql(
    """
    SELECT COUNT(rating) FROM ratings
    """,
    engine
)
count_df
```

```
   COUNT(rating)
0           1161
```

The strengh of aggregation becomes useful when using the GROUP BY keyword, where we can group our data based on columns. The next query calculates the average rating from the rating table grouped on the placeID, note when using grouping all other selected columns need to have an aggregation function in order to work.

```
avg_df = pd.read_sql(
    """
    SELECT placeID, AVG(rating) FROM ratings
    GROUP BY placeID
    """,
    engine
)
avg_df.head()
```

```
   placeID  AVG(rating)
0   132560         0.50
1   132561         0.75
2   132564         1.25
3   132572         1.00
4   132583         1.00
```

We can go further and combine joining and grouping, with this we can join the cuisine type from the cuisine table and group on that column, we then take both average and count of ratings.

```
cuisine_df = pd.read_sql(
    """
    SELECT cuisine.Rcuisine, AVG(ratings.rating), COUNT(ratings.rating)
    FROM ratings JOIN cuisine
    ON ratings.placeID == cuisine.placeID
    GROUP BY cuisine.Rcuisine
    """,
    engine
)
cuisine_df
```

```
            Rcuisine  AVG(ratings.rating)  COUNT(ratings.rating)
0           American             1.153846                     39
1           Armenian             1.250000                      4
2             Bakery             1.400000                      5
3                Bar             1.200000                    140
4    Bar_Pub_Brewery             1.305085                     59
5    Breakfast-Brunch            1.000000                      9
6            Burgers             1.032258                     31
7    Cafe-Coffee_Shop            1.583333                     12
8          Cafeteria             1.205882                    102
9            Chinese             1.219512                     41
10      Contemporary             1.250000                     32
11            Family             1.571429                     14
12         Fast_Food             1.164835                     91
13              Game             1.428571                      7
14     International             1.513514                     37
15           Italian             1.038462                     26
16          Japanese             1.344828                     29
17     Mediterranean             1.750000                      4
18           Mexican             1.189076                    238
19          Pizzeria             1.117647                     51
20          Regional             0.500000                      4
21           Seafood             1.241935                     62
22        Vietnamese             1.166667                      6
```

For an American cuisine we have an average rating of 1.15 and a count of 39 ratings. Keeping track of the count makes sure we known how many ratings are behind the average score.

Let's say we want to know the type with the highest average rating, we could use the ORDER BY keyword to order our results.

```
cuisine_df = pd.read_sql(
    """
    SELECT cuisine.Rcuisine, AVG(ratings.rating), COUNT(ratings.rating)
    FROM ratings JOIN cuisine
    ON ratings.placeID == cuisine.placeID
    GROUP BY cuisine.Rcuisine
    ORDER BY AVG(ratings.rating) DESC
    """,
    engine
)
cuisine_df
```

```
            Rcuisine  AVG(ratings.rating)  COUNT(ratings.rating)
0      Mediterranean             1.750000                      4
1   Cafe-Coffee_Shop             1.583333                     12
2             Family             1.571429                     14
3      International             1.513514                     37
4               Game             1.428571                      7
5             Bakery             1.400000                      5
6           Japanese             1.344828                     29
7    Bar_Pub_Brewery             1.305085                     59
8       Contemporary             1.250000                     32
9           Armenian             1.250000                      4
10           Seafood             1.241935                     62
11           Chinese             1.219512                     41
12         Cafeteria             1.205882                    102
```

(continues on next page)

(continued from previous page)

```
13             Bar          1.200000             140
14         Mexican          1.189076             238
15      Vietnamese          1.166667               6
16       Fast_Food          1.164835              91
17        American          1.153846              39
18        Pizzeria          1.117647              51
19         Italian          1.038462              26
20         Burgers          1.032258              31
21 Breakfast-Brunch         1.000000               9
22        Regional          0.500000               4
```

So, mediterranean cuisine has the highest rating, yet only 4 ratings are present, not a representable amount. What we could do is create a query that filters all the places with 5 or more ratings, we can use the HAVING keyword to filter groups whilst performing a GROUP BY operation.

```
place_df = pd.read_sql(
    """
    SELECT placeID, COUNT(rating)
    FROM ratings
    GROUP BY ratings.placeID
    HAVING COUNT(rating) > 4
    """,
    engine
)
place_df.head()
```

```
   placeID  COUNT(rating)
0   132572             15
1   132584              6
2   132594              5
3   132608              6
4   132609              5
```

With this query qe only keep the places with 5 or more ratings, as we chosen 5 as an arbitrary value of statistical significance here.

As a last query I would like to combine the last 2, where we use the filter as a subquery in our query to find the average of each cuisine type. This means that we take the average of each cuisine type, but only take into account places with 5 or more reviews.

```
cuisine_df = pd.read_sql(
    """
    SELECT cuisine.Rcuisine, AVG(ratings.rating), COUNT(ratings.rating)
    FROM ratings JOIN cuisine
    ON ratings.placeID == cuisine.placeID
    WHERE ratings.placeID in (
        SELECT placeID
        FROM ratings
        GROUP BY ratings.placeID
        HAVING COUNT(rating) > 4
    )
    GROUP BY cuisine.Rcuisine
    ORDER BY AVG(ratings.rating) DESC
    """,
    engine
)
```

(continues on next page)

(continued from previous page)

```
cuisine_df
```

```
             Rcuisine  AVG(ratings.rating)  COUNT(ratings.rating)
0              Family             1.600000                     10
1    Cafe-Coffee_Shop             1.583333                     12
2       International             1.513514                     37
3                Game             1.428571                      7
4            Japanese             1.423077                     26
5              Bakery             1.400000                      5
6     Bar_Pub_Brewery             1.290909                     55
7             Seafood             1.241935                     62
8             Chinese             1.216216                     37
9           Cafeteria             1.205882                    102
10                Bar             1.181818                    132
11            Mexican             1.181395                    215
12       Contemporary             1.178571                     28
13           American             1.171429                     35
14         Vietnamese             1.166667                      6
15          Fast_Food             1.159091                     88
16           Pizzeria             1.117647                     51
17   Breakfast-Brunch             1.000000                      9
18            Burgers             0.925926                     27
19            Italian             0.857143                     14
```

You can see that Mediterranean now is missing as it only had 4 ratings, yet the Family cuisine still has 10 out of 12 reviews and it's average even increased.

Although we used SQL which can only perform simple mathematics we were able to manipulate our dataset before even going into the data exploration phase. When dealing with larger datasets using SQL can drastically improve your data analytical experience and is therefore an essential tool for a data scientist

I'll leave a blank cell here for you to experiment, for more inspiration you could also check out this cheat sheet

Part IV

4. Data Visualisation

CHAPTER

NINETEEN

INTRODUCTION

this is an introduction

CHAPTER

TWENTY

LINE PLOT

The most straight-forward yet very useful plotting graph is the line plot. With the line plot we achieve the visualisation of a single feature organized in a usually time based reference.

The line plot is ideal if you want to achieve a time critical pattern residing within your data. In this example we use the prepared taxi dataframe that comes with our plotting library seaborn.

From all possible plotting libraries in Python we opted for the seaborn as it has an optimal combination of simplicity and beaty, yet other libraries are equally powerful.

We begin by importing our neccesary libraries

```
import pandas as pd
import seaborn as sns
sns.set_theme()
```

For aestetic reasons we change the figure size to something a bit larger

```
sns.set(rc={'figure.figsize':(16,12)})
```

We load our dataset, this dataset contains the trip of taxi's in regions of New York City with timestamps of pickup and dropoff.

```
taxi_df = sns.load_dataset('taxis')
taxi_df.head()
```

```
               pickup             dropoff  passengers  distance  fare   tip  \
0  2019-03-23 20:21:09  2019-03-23 20:27:24           1      1.60   7.0  2.15
1  2019-03-04 16:11:55  2019-03-04 16:19:00           1      0.79   5.0  0.00
2  2019-03-27 17:53:01  2019-03-27 18:00:25           1      1.37   7.5  2.36
3  2019-03-10 01:23:59  2019-03-10 01:49:51           1      7.70  27.0  6.15
4  2019-03-30 13:27:42  2019-03-30 13:37:14           3      2.16   9.0  1.10

   tolls  total   color      payment            pickup_zone  \
0    0.0  12.95  yellow  credit card        Lenox Hill West
1    0.0   9.30  yellow         cash  Upper West Side South
2    0.0  14.16  yellow  credit card          Alphabet City
3    0.0  36.95  yellow  credit card              Hudson Sq
4    0.0  13.40  yellow  credit card           Midtown East

            dropoff_zone pickup_borough dropoff_borough
0    UN/Turtle Bay South      Manhattan       Manhattan
1  Upper West Side South      Manhattan       Manhattan
2           West Village      Manhattan       Manhattan
3         Yorkville West      Manhattan       Manhattan
4         Yorkville West      Manhattan       Manhattan
```

As we saw earlier, it is important to prepare the data, due to storage specification they did not parse the dates into a datetime format, which we do here.

```
taxi_df.pickup = pd.to_datetime(taxi_df.pickup)
taxi_df.dropoff = pd.to_datetime(taxi_df.dropoff)
```

Before we can do anything with this dataset, we need to format it into a proper format, for our first graph I would like to view the total amount of passengers per day. This means we have to take our data and resample on the pickup date, taking the sum.

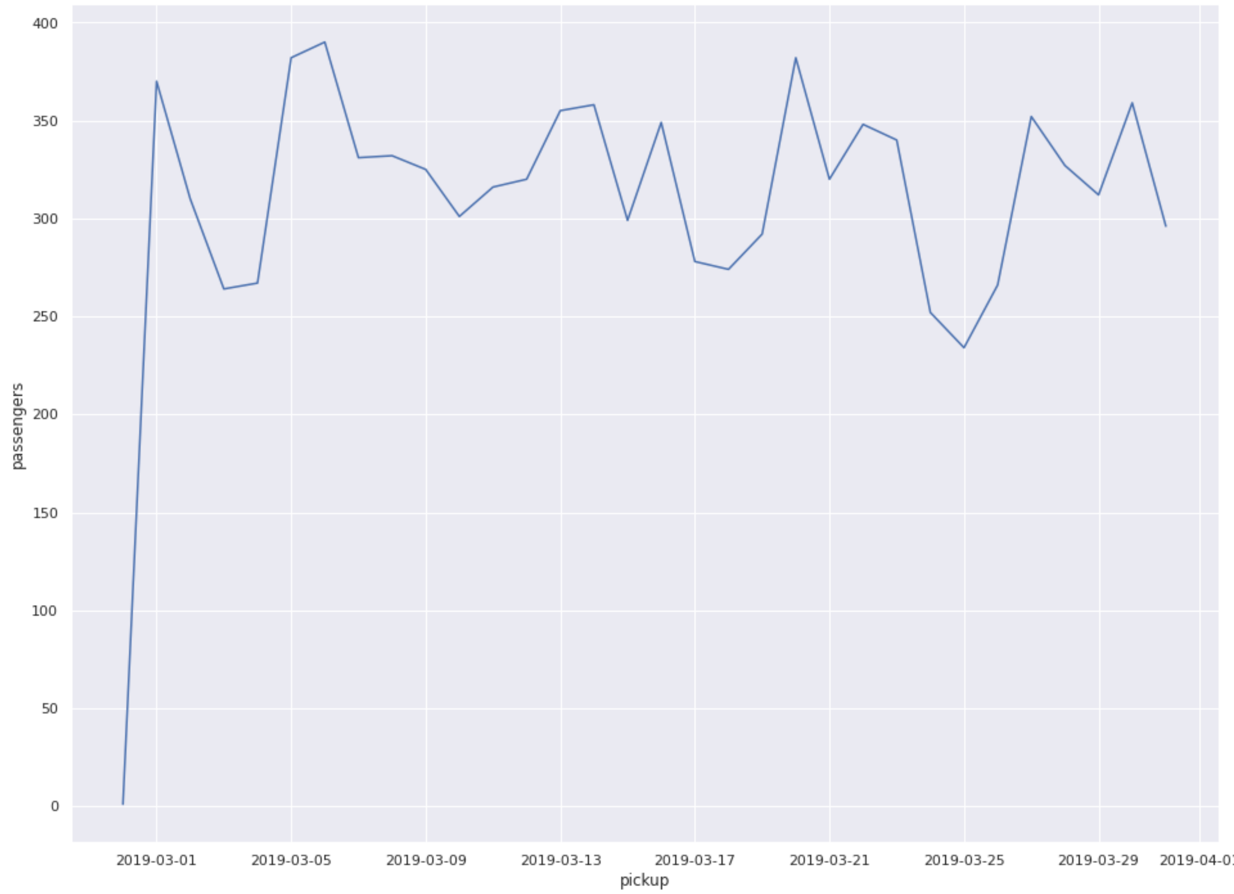
```
pass_df = taxi_df.set_index('pickup').resample('D').sum()
pass_df.head()
```

```
            passengers  distance     fare     tip  tolls    total
pickup
2019-02-28           1      0.90     5.00    0.00   0.00     6.30
2019-03-01         370    640.29  2946.97  442.47  60.34  4213.83
2019-03-02         310    548.70  2358.00  333.97  28.80  3319.02
2019-03-03         264    554.04  2187.89  307.47  34.56  3027.32
2019-03-04         267    583.81  2335.74  334.98  63.36  3269.08
```

You can almost see the plot here, we have an index of dates and a feature ‘passengers’, these two will make the backbone of our visualisation.

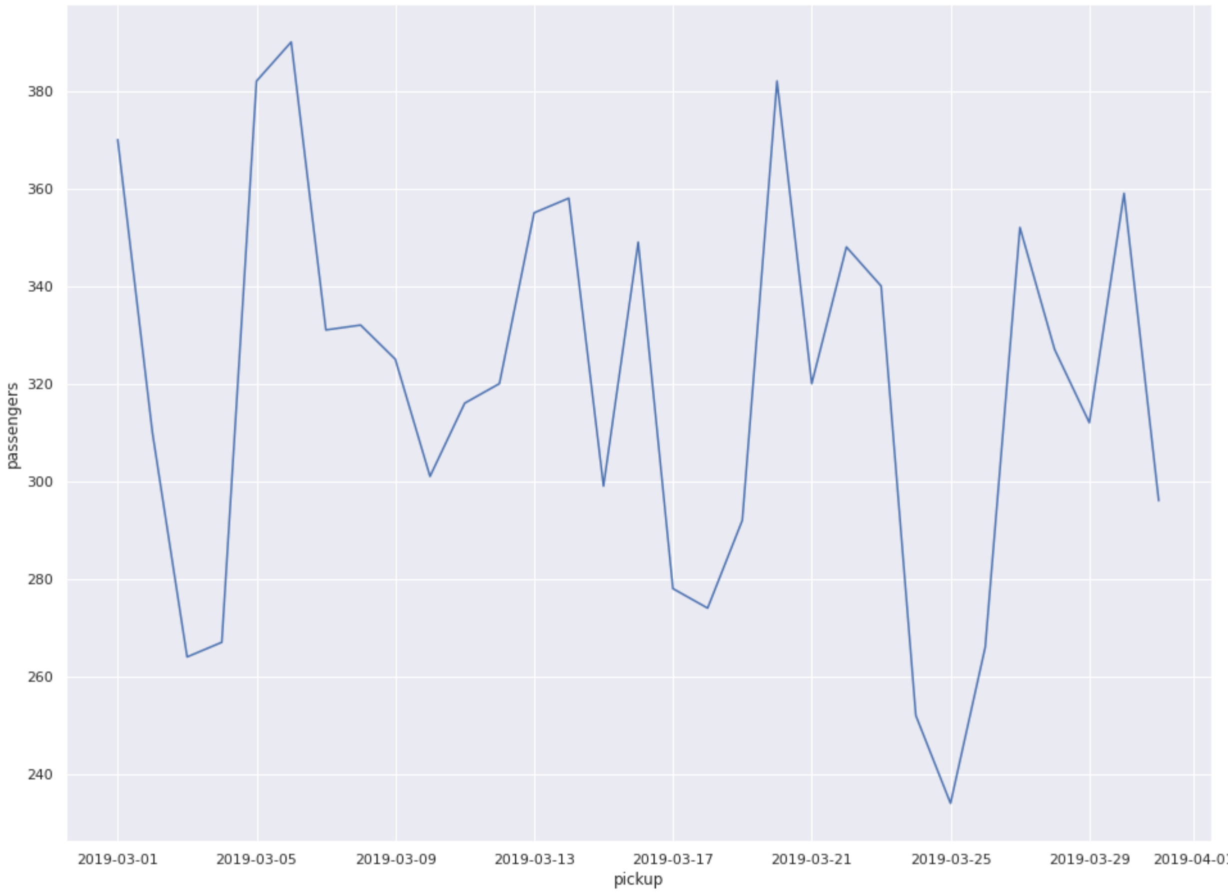
```
sns.lineplot(x=pass_df.index, y=pass_df.passengers)
```

```
<AxesSubplot:xlabel='pickup', ylabel='passengers'>
```



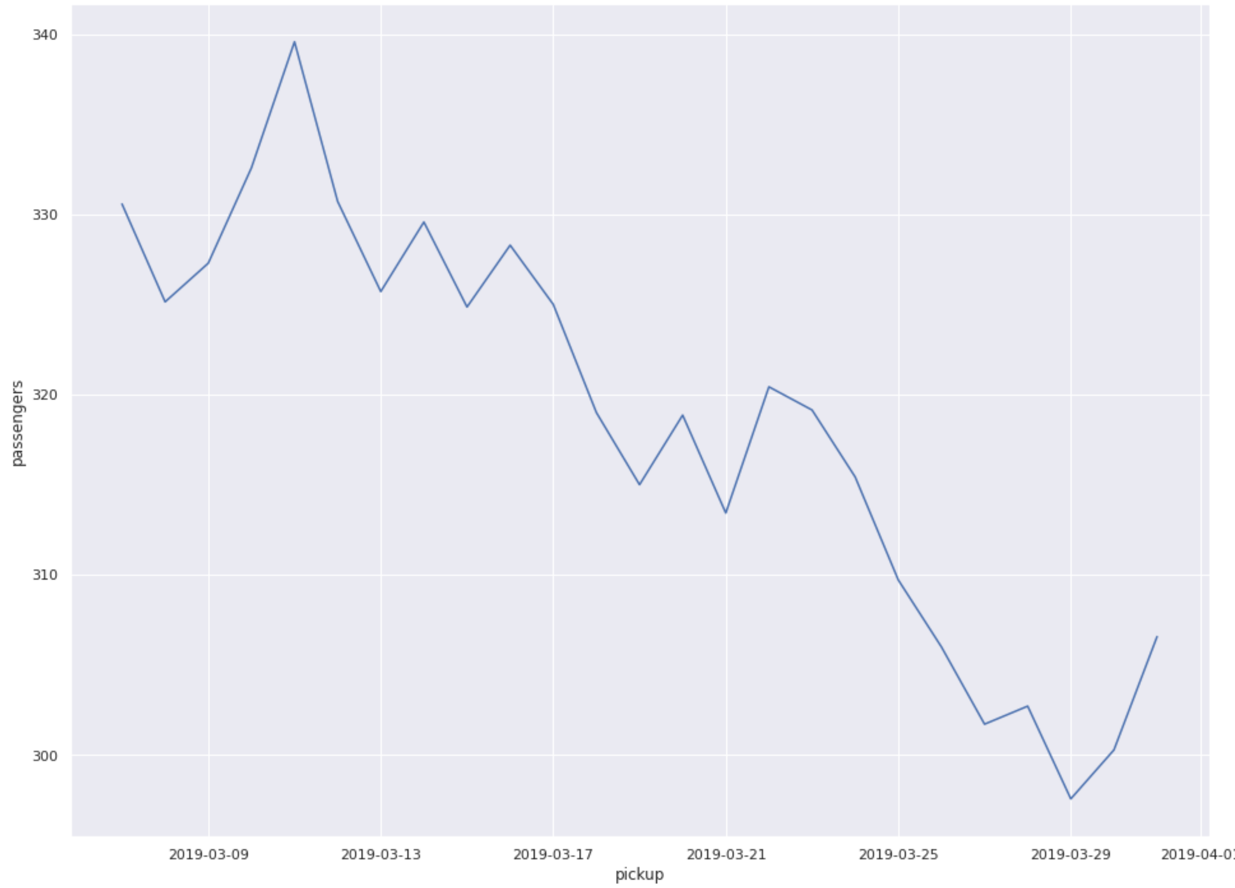
Looks about right, however I don't like the start of it, the data started late on that first day, resampling shows we only have 1 passenger for that day. This is not representable, so we remove that record.

```
pass_df = pass_df.loc['2019-03-01':]
ax = sns.lineplot(x=pass_df.index, y=pass_df.passengers)
```



Much better, however the plot feels like there is a lot of fluctuations, so it would be practical to apply a rolling sum or mean. This rolling operation takes the last x values and applies an operation (sum, mean,…) to it, creating a smoother graph and is visually more sensitive to trends.

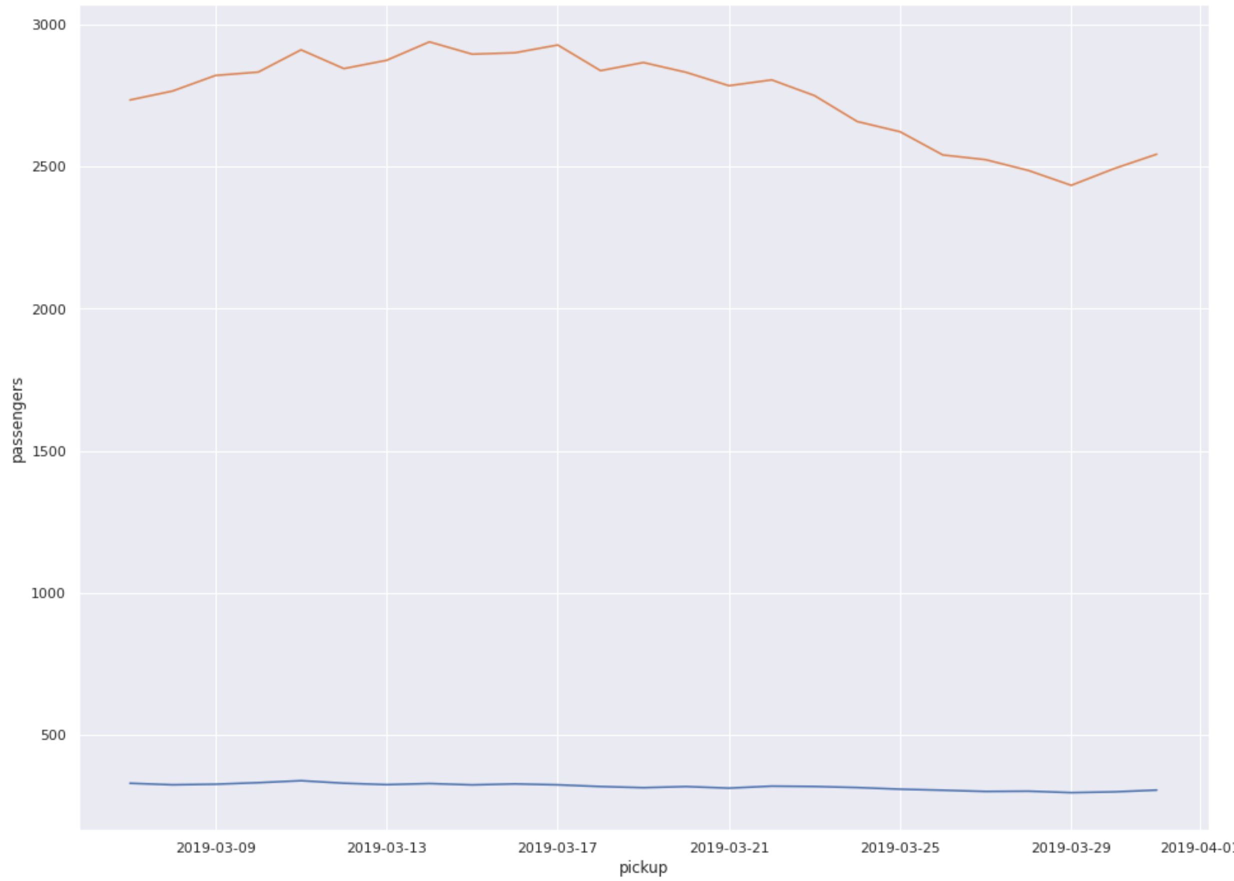
```
rolling_pass_df = pass_df.rolling(7).mean()
ax = sns.lineplot(x=rolling_pass_df.index, y=rolling_pass_df.passengers)
```



By applying a rolling mean, we can see that the average amount of passengers per day is decreasing. I feel there is no need to panick, as this is only 1 month of data and seasonal fluxtuations do happen.

Something else that triggers my curiosity is the amount these passengers paid, can we perhaps see a trend there? It would be ideal to plot these together so the comparison is simple.

```
ax = sns.lineplot(x=rolling_pass_df.index, y=rolling_pass_df.passengers)
ax = sns.lineplot(x=rolling_pass_df.index, y=rolling_pass_df.fare, ax=ax)
```



As we only have a few passengers per trip, yet trips can be costly the ranges of these 2 features are completely different. Before we think about scaling, we actually do want to know the scale here, we just cant fit them in the same graph.

A first approach would be to use a secondary axis, where the right side of the y-axis is used to show the fare scale. You can see that the graph is already getting more complicated code-wise, this is where using the right library is key as they usually have built in features for that.

```
ax = sns.lineplot(x=rolling_pass_df.index, y=rolling_pass_df.passengers, label=
↪'passengers', legend=False)
ax2 = ax.twinx()
ax = sns.lineplot(x=rolling_pass_df.index, y=rolling_pass_df.fare, ax=ax2, color='r',␣
↪label='fare', legend=False)
ax.figure.legend()
```

```
<matplotlib.legend.Legend at 0x7fc9f08c1fd0>
```

Interesting! It shows that there was a period where they did not follow eachother perfect, yet the trend is almost exact for these features.

Another method where you can compare them would require feature engineering, where we calculate the fare per passenger per day, apply the rolling window and plot. Perhaps you could figure that out? create a new feature that divides the fare by the passengers, recreate the rolling dataframe and use seaborn to plot the results.

At the start we used the sum of passengers per day, however we could also visualise the average amount of passengers per ride. The reason why I would like to do this is because earlier I saw a difference in trend for the fare and the amount of passengers, an explanation for this could be that the average amount of passengers dropped, resulting in lower passengers, yet the total expenditure of fares would remain constant.

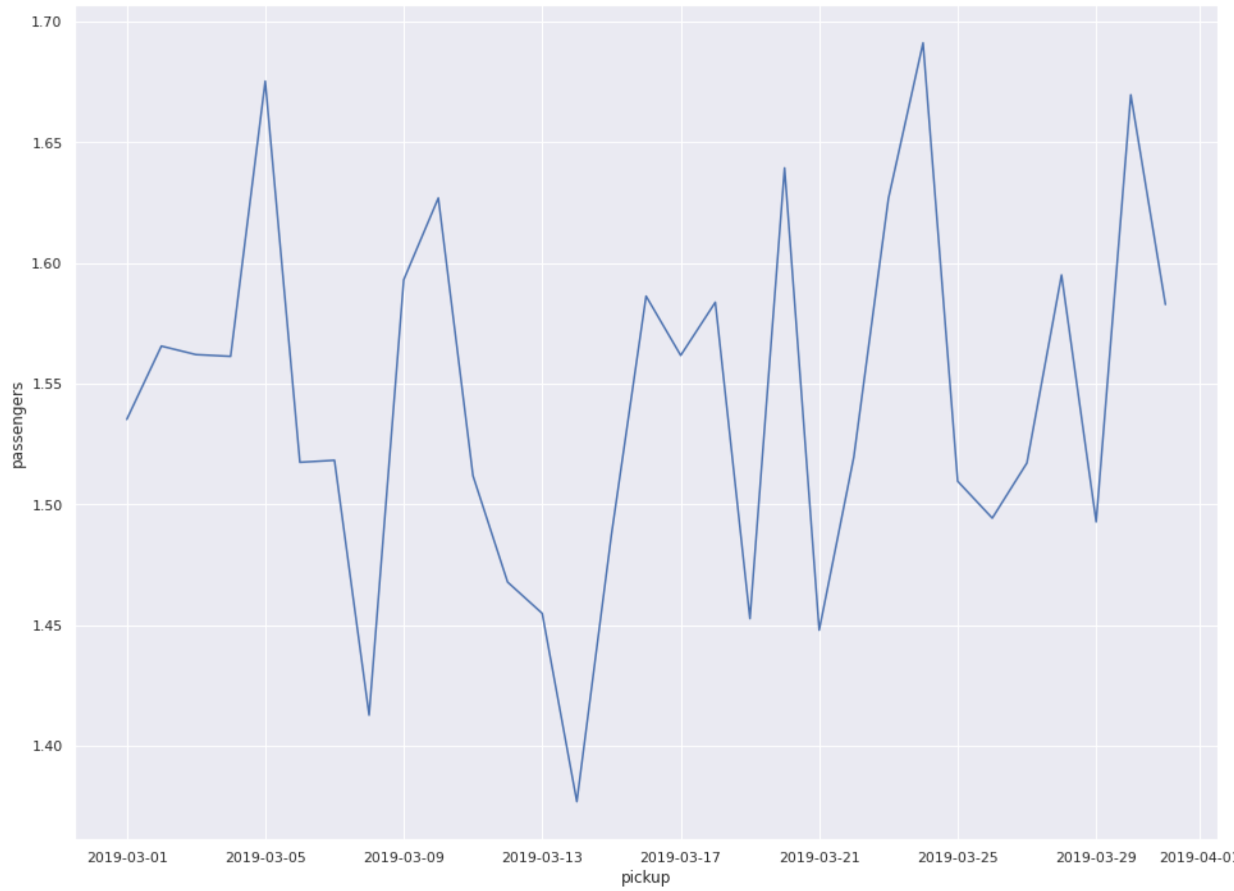
Let us figure this out, we here calculate the average (mean) of the passengers per day.

```
avg_pass_df = taxi_df.set_index('pickup').resample('D').mean()
avg_pass_df.head()
```

```
            passengers  distance       fare       tip     tolls      total
pickup
2019-02-28    1.000000  0.900000   5.000000  0.000000  0.000000   6.300000
2019-03-01    1.535270  2.656805  12.228091  1.835975  0.250373  17.484772
2019-03-02    1.565657  2.771212  11.909091  1.686717  0.145455  16.762727
2019-03-03    1.562130  3.278343  12.946095  1.819349  0.204497  17.913136
2019-03-04    1.561404  3.414094  13.659298  1.958947  0.370526  19.117427
```

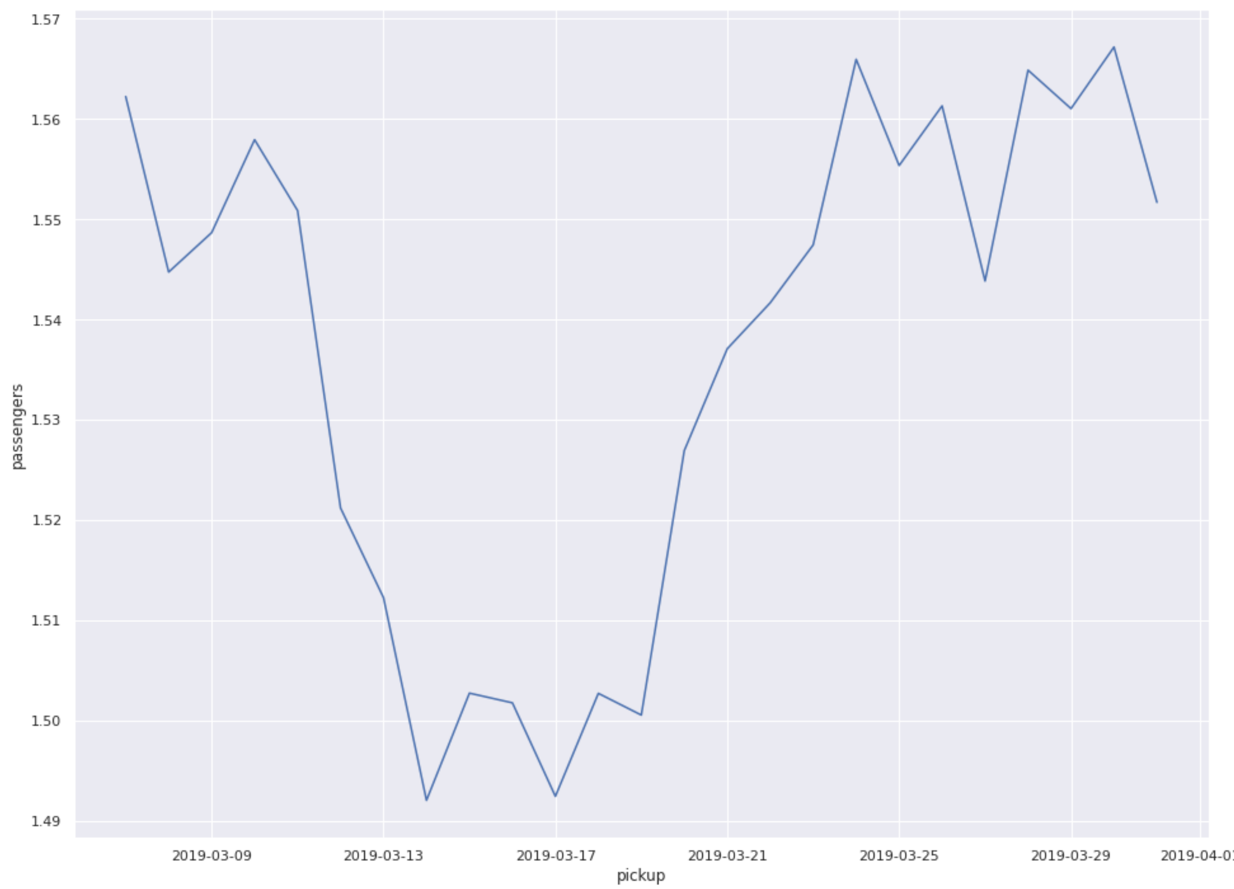
Doing more or less exactly the same we can create a simple plot with the average amount of passengers in a taxi.

```
avg_pass_df = avg_pass_df[1:]
ax = sns.lineplot(x=avg_pass_df.index, y=avg_pass_df.passengers)
```



For the same reasons, this plot is not suitable as it has too much variance. We apply a rolling mean of 7 days and re-evaluate.

```
rolling_avg_pass_df = avg_pass_df.rolling(7).mean()
ax = sns.lineplot(x=rolling_avg_pass_df.index, y=rolling_avg_pass_df.passengers)
```



We find a dip in passengers per ride that looks to be in the same time interval, therefore we could conclude here that fares did not get more expensive, rather the sharing of cabs was less. You could try and find a method to add the data of these two graphs together, yet this is already advanced visualisation.

Another question that I have for you, do you think that the dip is relevant? Not specifically from a business point of view, rather from a statistical view, Perhaps if you look at the range of the y-axis you might feel that our plot is a bit magnified. This is a good example of how you can use ranges of your axi to make data more dramatic. Be weary of these malpractices!

We are not done yet, as our dataset contains much more information. Harnessing the powers of the preprocessing we learned, we could include other (mostly categorical) feature into our line plot.

Here we take the payment option (either cash or card) and use it to create 2 time series in long format (2 datasets below each other).

```
pass_payment_df = taxi_df.groupby('payment').apply(
    lambda x: x.set_index('pickup').resample('D').sum()
)
pass_payment_df
```

```
                        passengers  distance    fare   tip  tolls   total
payment     pickup
cash        2019-02-28           1      0.90    5.00  0.00   0.00    6.30
            2019-03-01         104    112.31  571.50  0.00   5.76  748.76
            2019-03-02          86    159.46  690.50  0.00   5.76  863.96
            2019-03-03          67    172.34  641.50  0.00  17.28  782.18
            2019-03-04          71    130.60  571.50  0.00   0.00  710.95
```

(continues on next page)

(continued from previous page)

```
...                                 ...       ...      ...      ...     ...      ...
credit card 2019-03-27              263    532.61  2260.64   485.63   69.12  3342.29
            2019-03-28              227    403.41  1886.07   404.45   40.32  2802.94
            2019-03-29              211    404.61  1831.98   410.13   23.04  2747.85
            2019-03-30              268    540.71  2211.10   487.97   78.62  3249.49
            2019-03-31              202    376.78  1632.93   345.83   29.16  2408.42

[63 rows x 6 columns]
```

Seaborn does not like this long format type, therefore we unstack the first index and create a wide format. For those wo are punctilious, you can notice we created a missing value, with wat should we fill it? (Our luck that seaborn can handle missing values!)

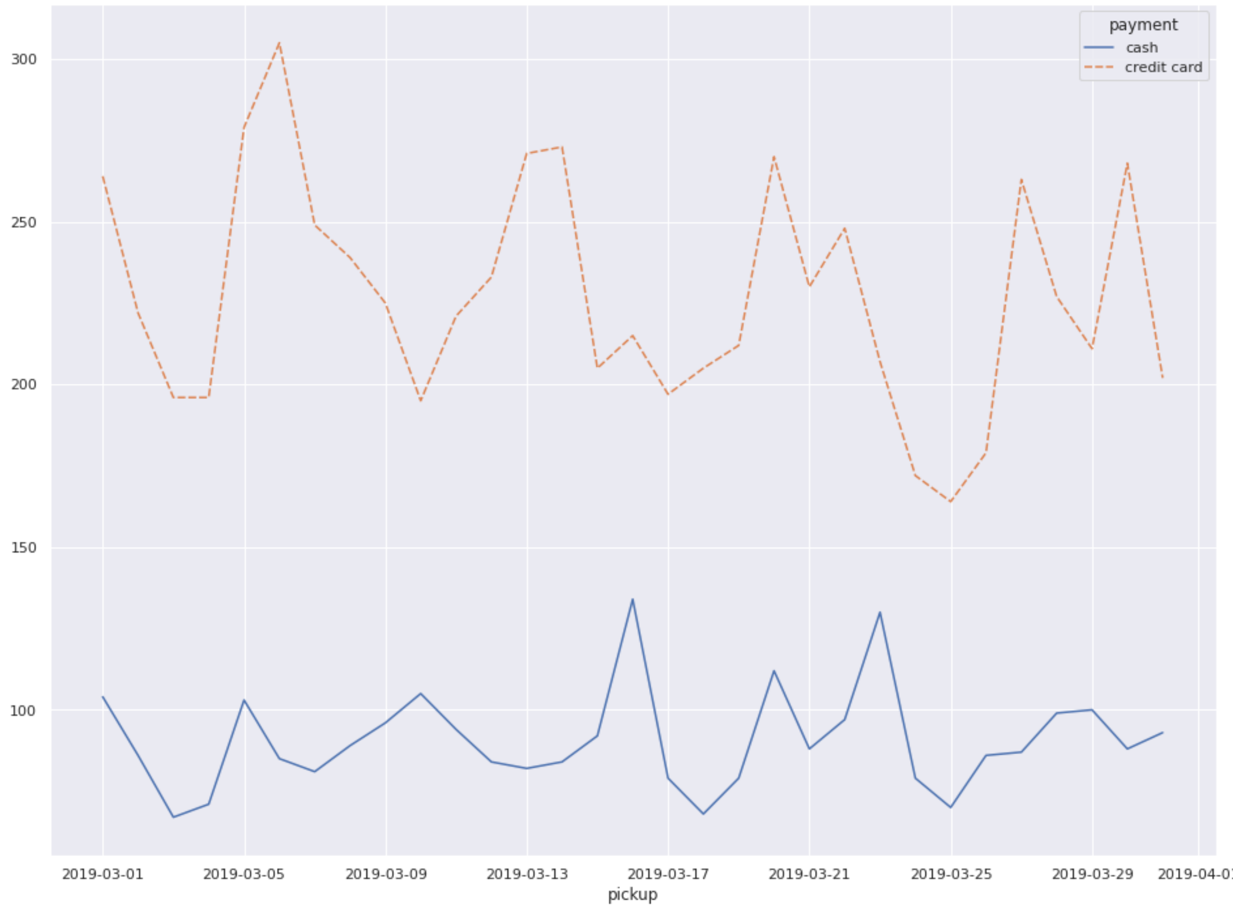
```
pass_payment_df.unstack(0).head()
```

```
           passengers             distance              fare                 \
payment          cash credit card     cash credit card   cash credit card
pickup
2019-02-28        1.0         NaN     0.90         NaN    5.0         NaN
2019-03-01      104.0       264.0   112.31      527.08  571.5     2363.97
2019-03-02       86.0       222.0   159.46      377.74  690.5     1651.50
2019-03-03       67.0       196.0   172.34      381.60  641.5     1526.39
2019-03-04       71.0       196.0   130.60      453.21  571.5     1764.24

            tip              tolls               total
payment    cash credit card   cash credit card    cash credit card
pickup
2019-02-28  0.0         NaN   0.00         NaN    6.30         NaN
2019-03-01  0.0      442.47   5.76       54.58  748.76     3446.47
2019-03-02  0.0      333.97   5.76       23.04  863.96     2430.36
2019-03-03  0.0      307.47  17.28       17.28  782.18     2224.34
2019-03-04  0.0      334.98   0.00       63.36  710.95     2558.13
```

Same data, different structure, now seaborn understands the format and we can go back to visualisation.

For simplicity we start with a simple passengers line plot

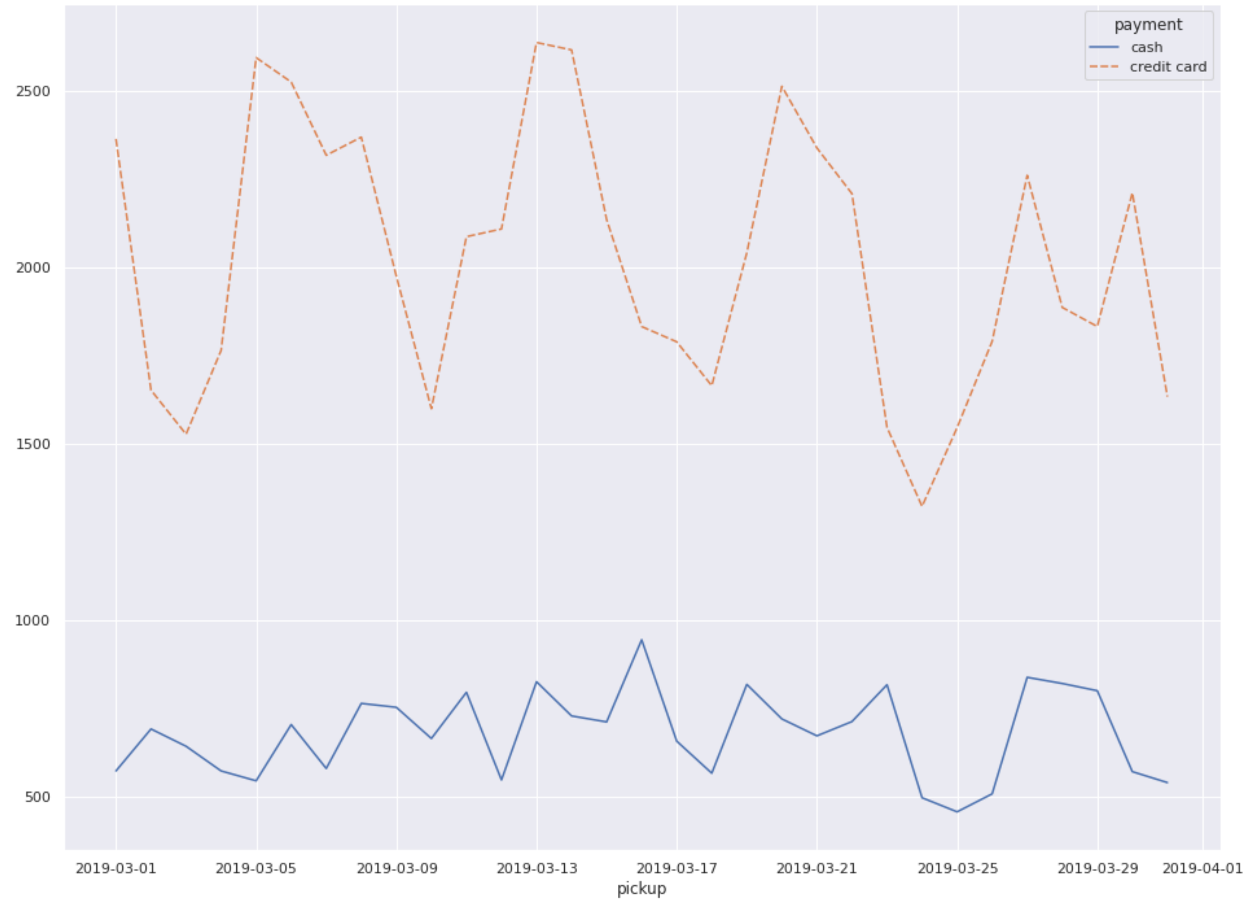
```
ax = sns.lineplot(data=pass_payment_df.passengers.unstack(0)[1:])
```



You can see that there are generally more people paying by card, which is more convenient in such an occasion. Note that here we should not use a seperate y-axis as we are comparing 2 sets of data that are similar by origin.

We do the same for fares.

```
ax = sns.lineplot(data=pass_payment_df.fare.unstack(0)[1:])
```



This is more or less a no-brainer, as more people pay by card, the fares by card are also more. So we can't really compare fares with this plot, we have to be creative.

I opted to go for an average fare per passenger, as this is in my opinion more relevant than the amount of rides

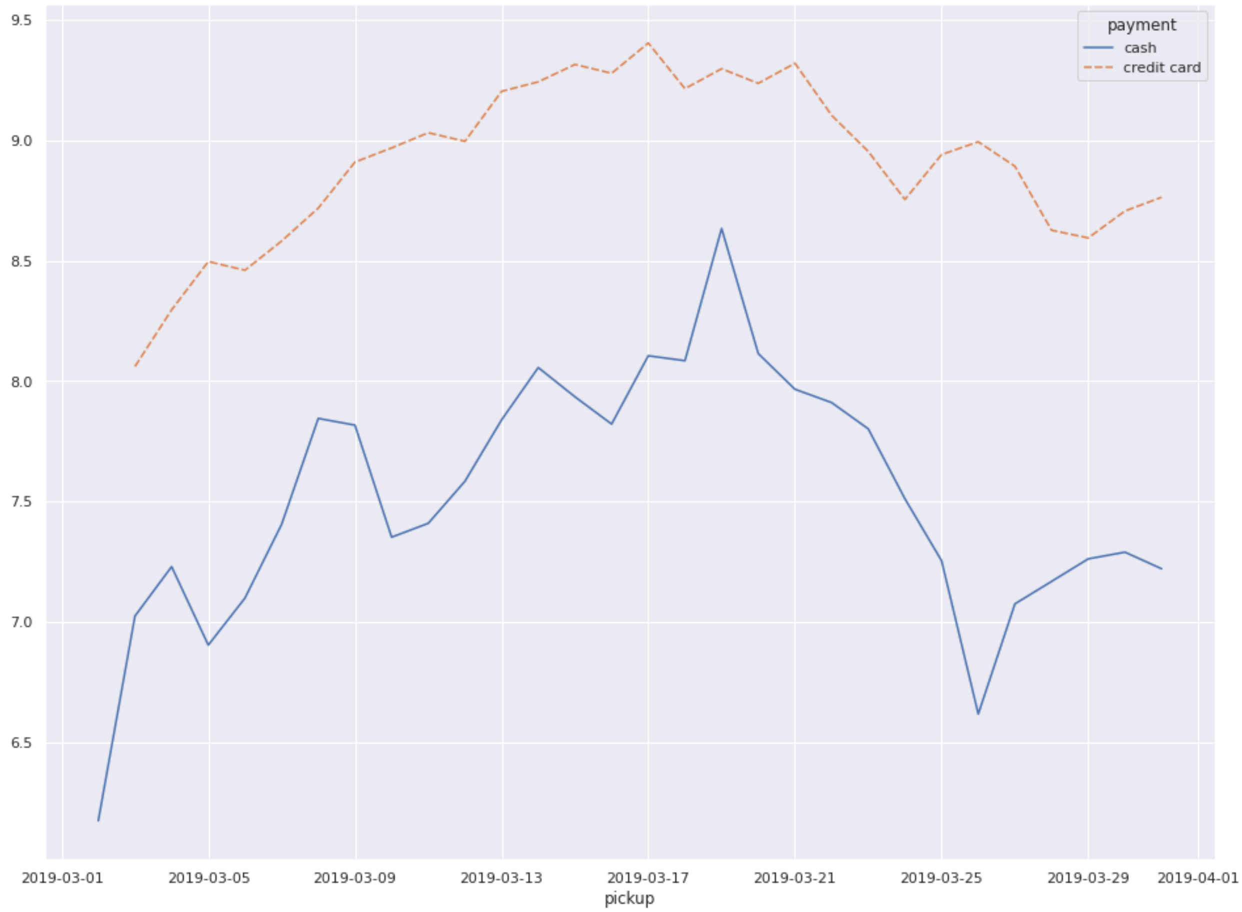
```
pass_payment_df['fare_pass'] = pass_payment_df.fare/pass_payment_df.passengers
pass_payment_df.head()
```

```
                        passengers  distance   fare  tip  tolls   total  fare_pass
payment pickup
cash    2019-02-28               1      0.90    5.0  0.0   0.00    6.30   5.000000
        2019-03-01             104    112.31  571.5  0.0   5.76  748.76   5.495192
        2019-03-02              86    159.46  690.5  0.0   5.76  863.96   8.029070
        2019-03-03              67    172.34  641.5  0.0  17.28  782.18   9.574627
        2019-03-04              71    130.60  571.5  0.0   0.00  710.95   8.049296
```

We created a new feature both containing info of fares and passengers, using this we create a new visualisations.

In this visualisation we show for both payment options the average fare amount per passenger in the cab.

```
ax = sns.lineplot(data=pass_payment_df.fare_pass.unstack(0).rolling(7, min_periods=3).
↪mean())
```



We can conclude that the average amount that has to be paid per person is lower for cash, indicating that people jump to their debit card as soon as the amount gets too high.

As a last I would like to emphasise that the x-axis, being time does not have to be linear. To illustrate this we create a weekly passenger rate and impose each week over the others.

```
pass_df.groupby(pd.Grouper(freq='W')).apply(
    lambda x: sns.lineplot(x=x.index.day_name(), y=x.passengers)
)
```

```
pickup
2019-03-03    AxesSubplot(0.125,0.125;0.775x0.755)
2019-03-10    AxesSubplot(0.125,0.125;0.775x0.755)
2019-03-17    AxesSubplot(0.125,0.125;0.775x0.755)
2019-03-24    AxesSubplot(0.125,0.125;0.775x0.755)
2019-03-31    AxesSubplot(0.125,0.125;0.775x0.755)
Freq: W-SUN, dtype: object
```

Here we can see there is a weekly trend occuring, where Sundays and Mondays are usually less busy days. The origin of this is hard to argue, as it might be less traffic, less taxi drivers working,…

Perhaps you could complete this visualisation by investigating the distance and/or tips?

CHAPTER

TWENTYONE

HISTOGRAM PLOT

When visualising one dimensional data without relating it to other information an option would be histograms. Histograms are used when describing distributions in your data, it is not the values itself you are visualising, rather the counts/frequencies of each value.

We again start with importing our libraries

```
import pandas as pd
import seaborn as sns
sns.set_theme()
sns.set(rc={'figure.figsize':(16,8)})
```

For this example we will be using the prepared dataset from seaborn containing mileages of several cars. Information about the cars is also given.

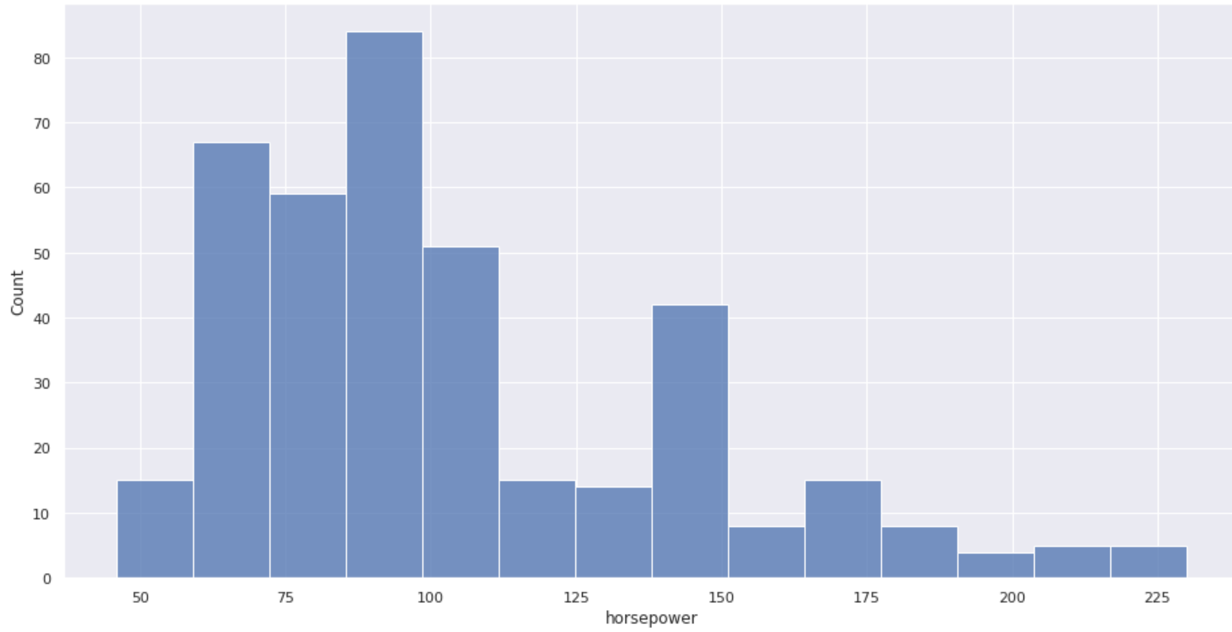
```
mpg_df = sns.load_dataset('mpg')
mpg_df.head()
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0    3504          12.0   
1  15.0          8         350.0       165.0    3693          11.5   
2  18.0          8         318.0       150.0    3436          11.0   
3  16.0          8         304.0       150.0    3433          12.0   
4  17.0          8         302.0       140.0    3449          10.5   

   model_year origin                       name  
0          70    usa  chevrolet chevelle malibu  
1          70    usa          buick skylark 320  
2          70    usa         plymouth satellite  
3          70    usa              amc rebel sst  
4          70    usa                ford torino  
```

We start of simple by plotting the distribution of horsepower in our dataset.

```
sns.histplot(data=mpg_df, x='horsepower')
```

```
<AxesSubplot:xlabel='horsepower', ylabel='Count'>
```

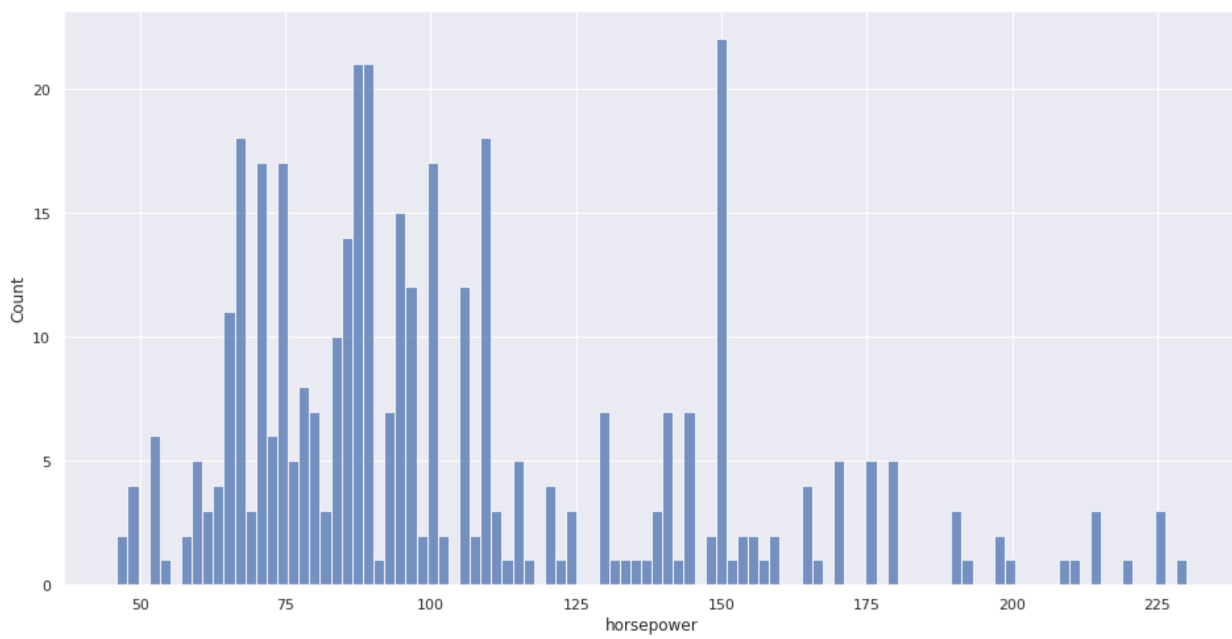


A first thing that is visible is that our feature is not normally distributed, we have a long tail to the higer end.

For histograms we can specify the amount of bins in which we seperate the counts, seaborn selects a suitable number yet we can change this.

```
sns.histplot(data=mpg_df, x='horsepower', bins=100)
```

```
<AxesSubplot:xlabel='horsepower', ylabel='Count'>
```

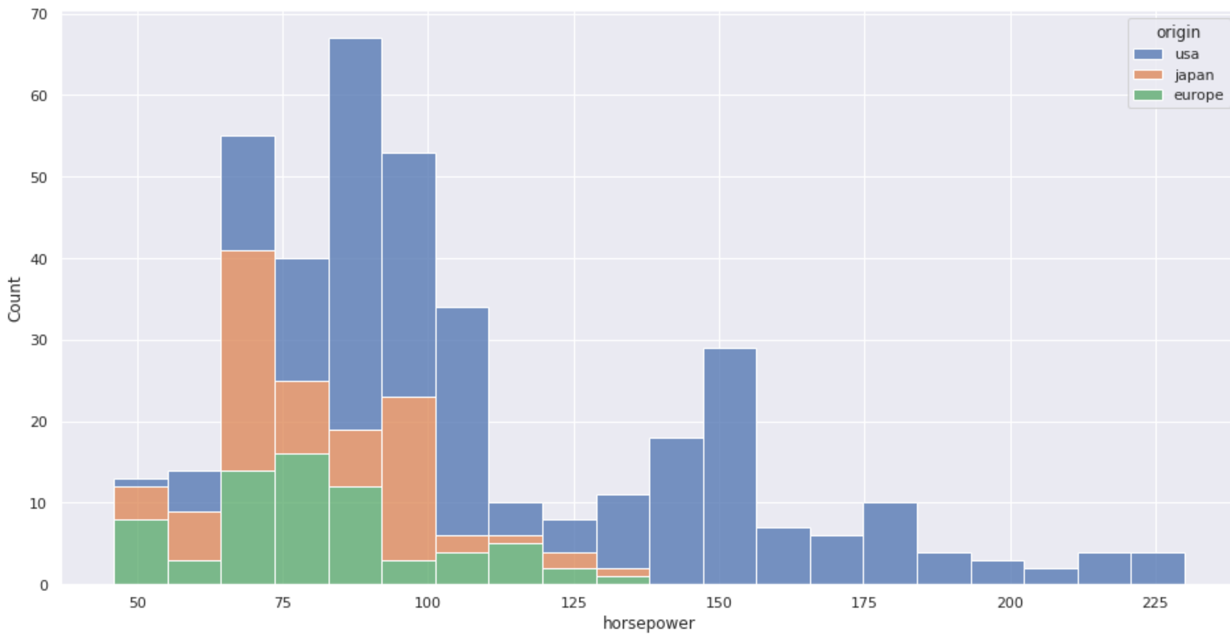


As you can see, the previous option looks a lot better. Taking the right amount of bins is important.

In order to add more information to our plot, we can use categorical data to split our data into multiple histograms. Here we used the origin of the cars to split into 3 categories, notice how each of them has their own area, japan and europe are on the lower end whilst usa is centered in higher horsepower.

```
sns.histplot(data=mpg_df, x='horsepower', hue='origin', bins=20, multiple='stack')
```

```
<AxesSubplot:xlabel='horsepower', ylabel='Count'>
```

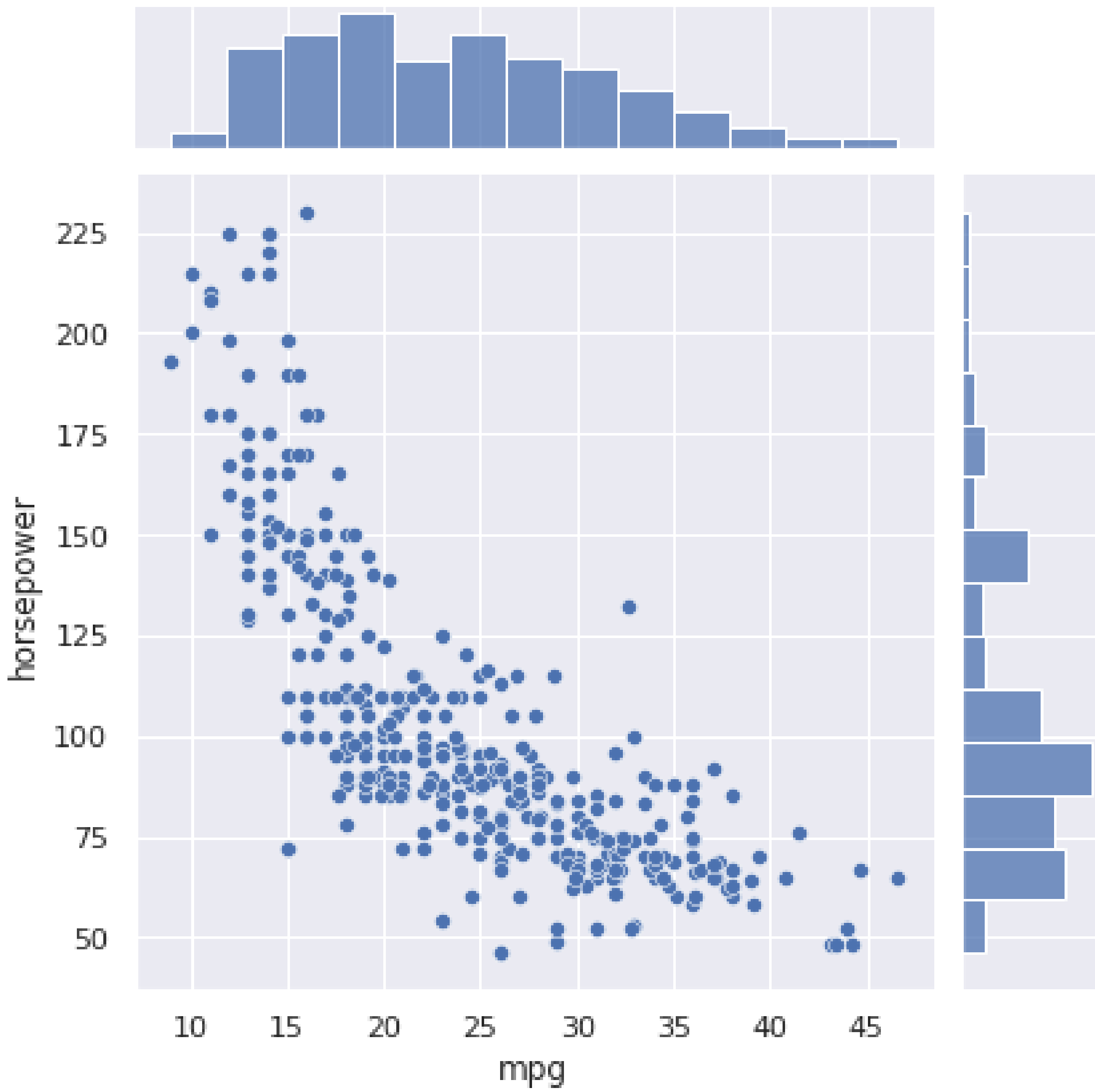


A neat feature of seaborn is that it can join histograms and scatter plots (in the next section) together.

Here we see how the visualisations of 2 one dimensional histograms perfectly combine together into a scatter plot, where 2 dimensional data is shown (both mileage and horsepower).

```
sns.jointplot(data=mpg_df, x='mpg', y='horsepower')
```

```
<seaborn.axisgrid.JointGrid at 0x7ff9e42c8fa0>
```



Histograms are a really powerfull tool when it comes to validating your data, we can easily the distribution of each feature, see if they are normally distributed and visualise distributions of subgroups.

Yet for final visualisations they are often not interesting enough.

CHAPTER

TWENTYTWO

BOX PLOT

In the previous section we looked into visualising the distributions of 1 dimensional data. We used histograms for this, but there is a second more statistical option for this, the Boxplot.

To be brief, the boxplot shows a box containing the InterQuartile Data that we already talked about and also has 2 whiskers, showing the threshold for outliers. Actual outliers are then printed seperately, making this plot ideal for outlier detection aswel as distributions.

I personally think this option is more suited for multiple categories compared to histograms, yet your mileage may vary.

```
import pandas as pd
import seaborn as sns
sns.set_theme()
sns.set(rc={'figure.figsize':(16,12)})
```

For this section we will look into the discovery of extrasolar planets, or planets that are ourside our own solar system. For each planet they listed the method of discovery, orbital period, mass, distance and year of discovery.

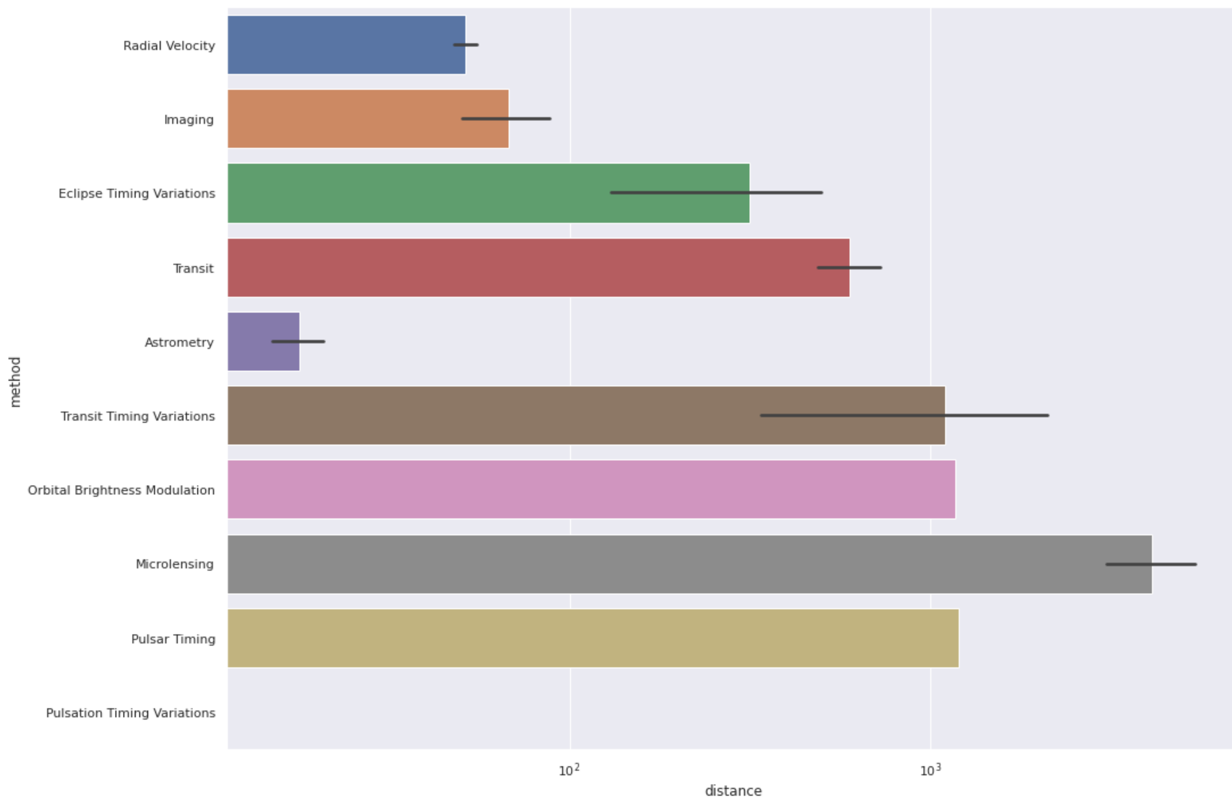
```
planet_df = sns.load_dataset('planets')
planet_df.head()
```

```
            method  number  orbital_period   mass  distance  year
0  Radial Velocity       1         269.300   7.10     77.40  2006
1  Radial Velocity       1         874.774   2.21     56.95  2008
2  Radial Velocity       1         763.000   2.60     19.84  2011
3  Radial Velocity       1         326.030  19.40    110.62  2007
4  Radial Velocity       1         516.220  10.50    119.47  2009
```

Let's say we would like to show the distances of each discovery method, if we would use a bar plot, the results might be hard to interpret.

```
ax = sns.barplot(data=planet_df, x='distance', y='method')
ax.set(xscale="log")
```

```
[None]
```



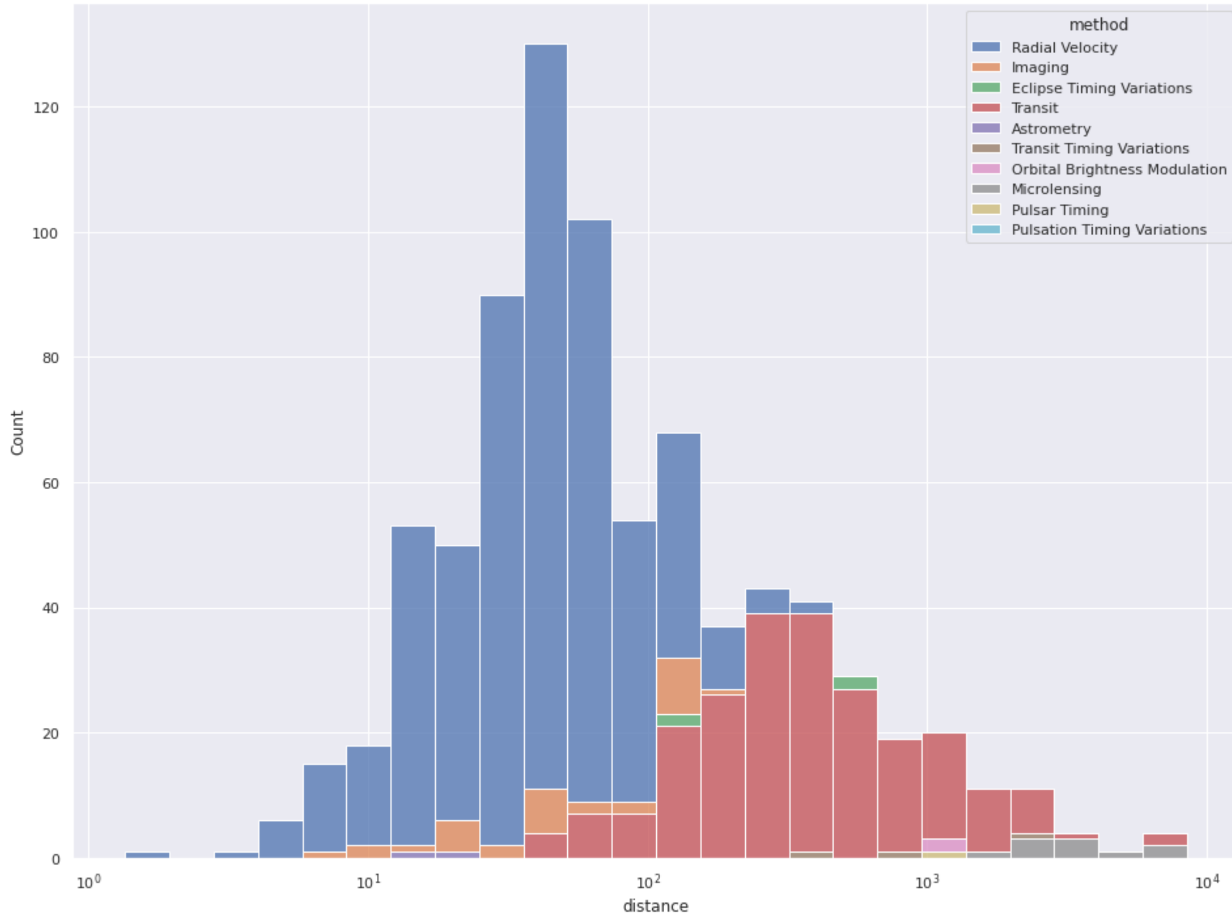
Whilst bar plots can be a good idea, here they are not.

Only use bar plots when visualising singular data points who are related to zero, not aggregations of multiple data points. Bar plots do not work if:

- your datapoints have no relation to zero
- your categories are related with different intervals
- you are dealing with groups of datapoints, not single datapoints (this case)

anyway, we could use a histogram similar to previous section, let's see how that turns out.

```
ax = sns.histplot(data=planet_df, x='distance', hue='method', multiple='stack', log_
↪scale=True)
```

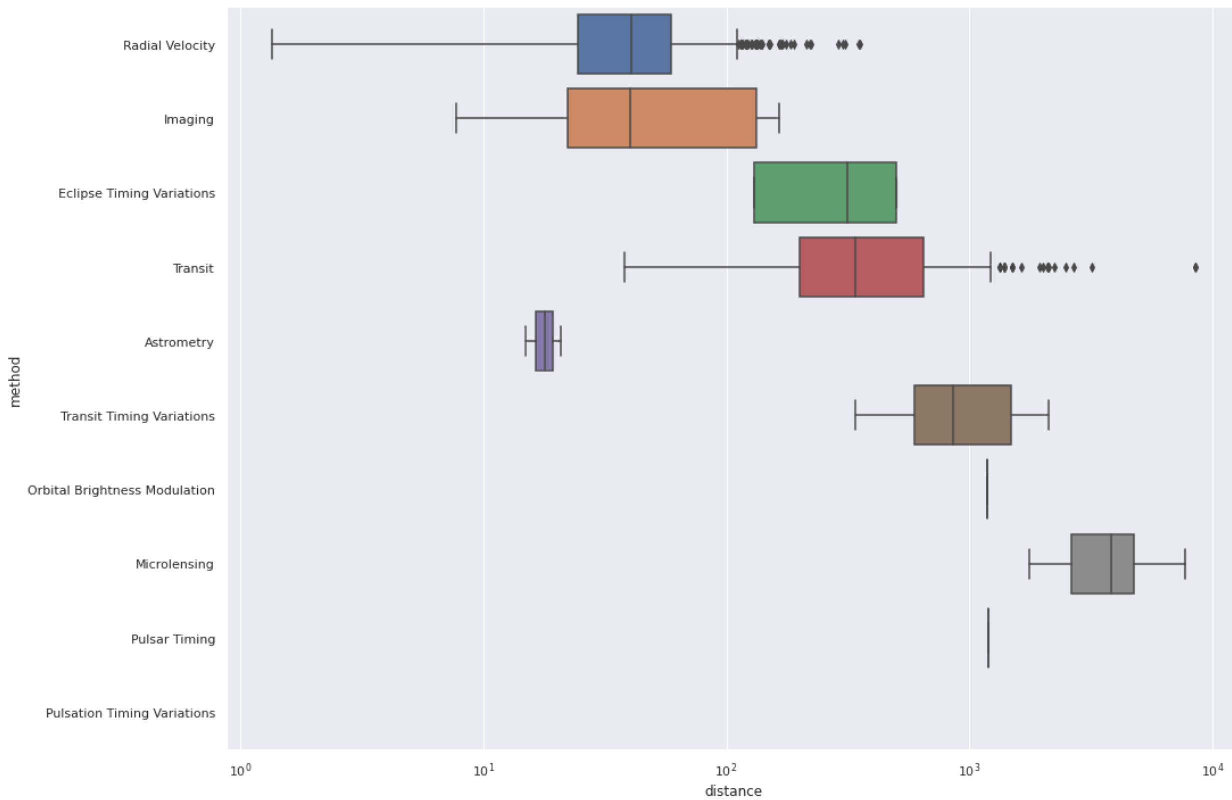


The histogram seems to be working, yet the methods with lower count are suppressed. A boxplot can overcome this and we can also compare medians of each method with eachother.

Take a few minutes to understand the next plot, at first it is very confusing, yet when adapted this is the most powerful visualisation of data exploration.

```
ax = sns.boxplot(data=planet_df, x='distance', y='method')
ax.set(xscale="log")
```

```
[None]
```



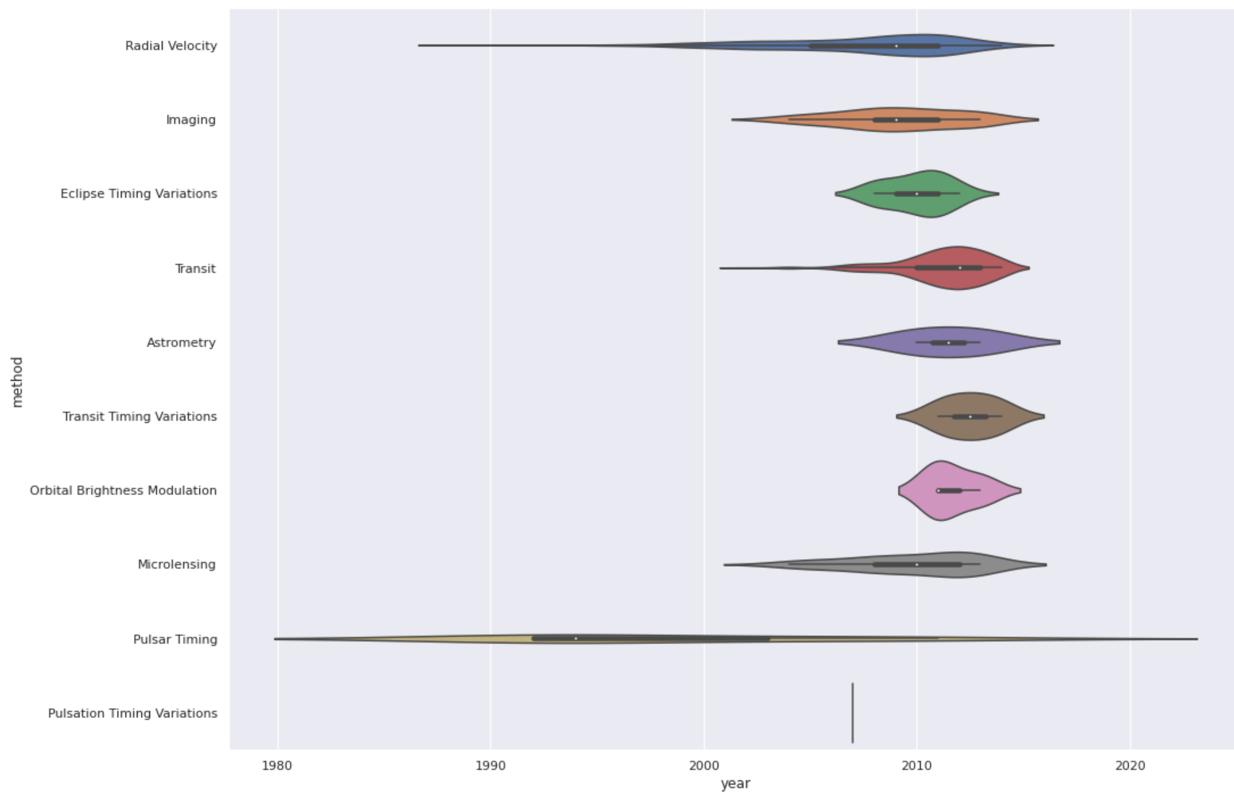
Can you see now why the bar plot here is a bad idea? Some methods have a broader distribution and relating our data to zero makes no real sense. With financial data this is different as budgets always start with 0.

Here we can conclude that some methods of detecting a planet requires a further or closer distance. You could say that if you want to discover a far extrasolar plant pick one of the last methods

An addition to the boxplot, where we focus more on distribution instead of statistics, would be the violin plot. Can you see why they would call it like that?

```
sns.violinplot(data=planet_df, x='year', y='method')
```

```
<AxesSubplot:xlabel='year', ylabel='method'>
```



As an exercise calculate the median, Q1 and Q3 of the distance per method and see if you come to the same conclusion as the boxplot

CHAPTER

TWENTYTHREE

SCATTER PLOT

Thus far we dealt with one dimensional data in our visualisations, sometimes adding a category to divide our data. Here we take it a step further, scatter plots are to visualise the relation between 2 numerical features.

One remark that I would like to make here is that discrete numerical features (age, n_persons,…) are possible to use, yet when dealing with a small range (e.g. 0-10) the results are skewed.

```
import pandas as pd
import seaborn as sns
sns.set_theme()
sns.set(rc={'figure.figsize':(16,8)})
```

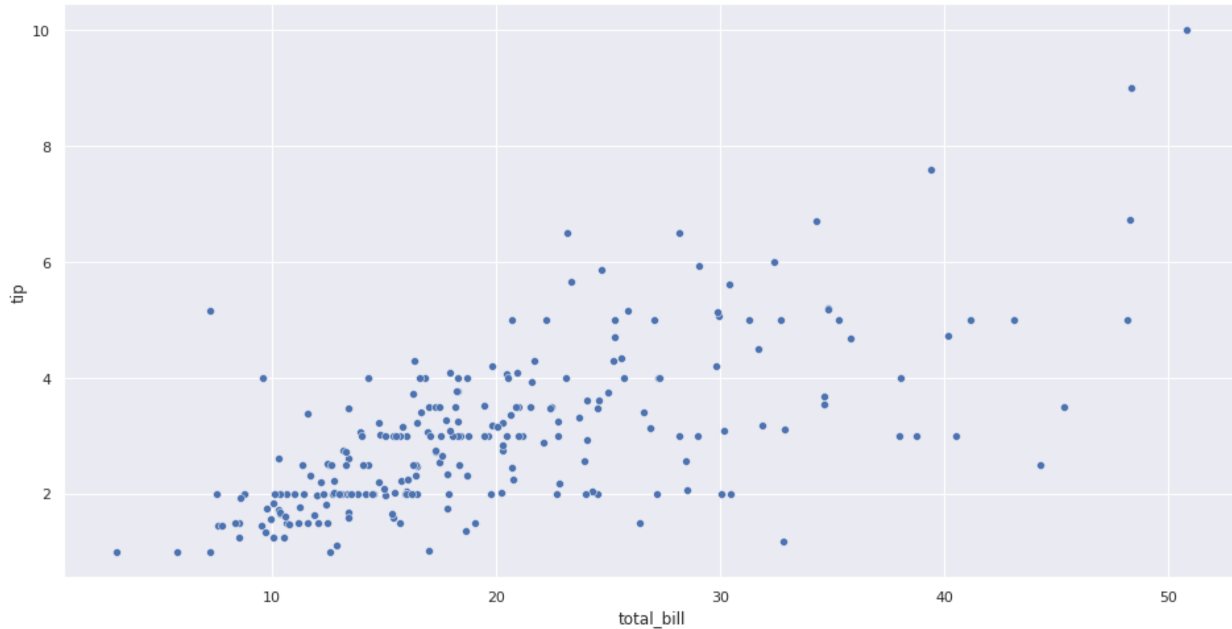
For scatter plots I opted to use a dataset containing tips from a restaurant, the tips are divided in gender, smoker, time of day and day of week.

```
tips_df = sns.load_dataset('tips')
tips_df.head()
```

```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

The most simple scatter plot we can make would be showing the relation between the total bill and the tip, we would assume the tip is proportional to the size of the bill.

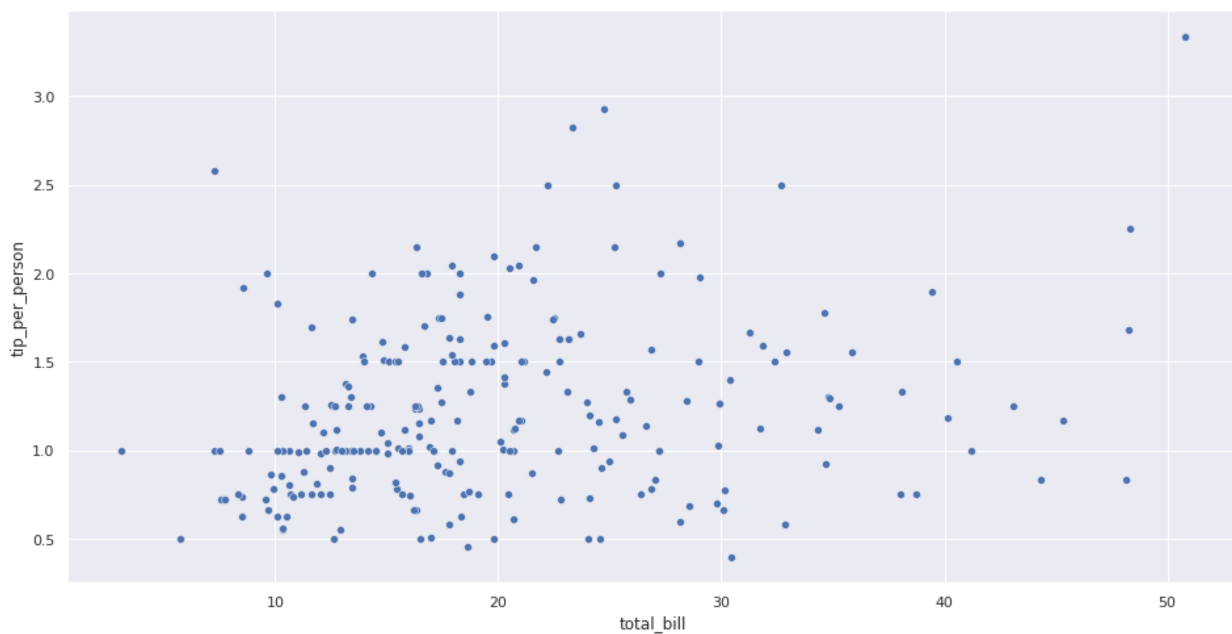
```
ax = sns.scatterplot(data=tips_df, x="total_bill", y="tip")
```



Just as expected, when the total bill rises, the tip grows too, we have some generous persons, and some less generous, but nothing out of the ordinary.

To get a better idea of the tipping habits we could calculate the tip per person in the bill, which is noted by size. We divide the tip by the amount of people and plot again.

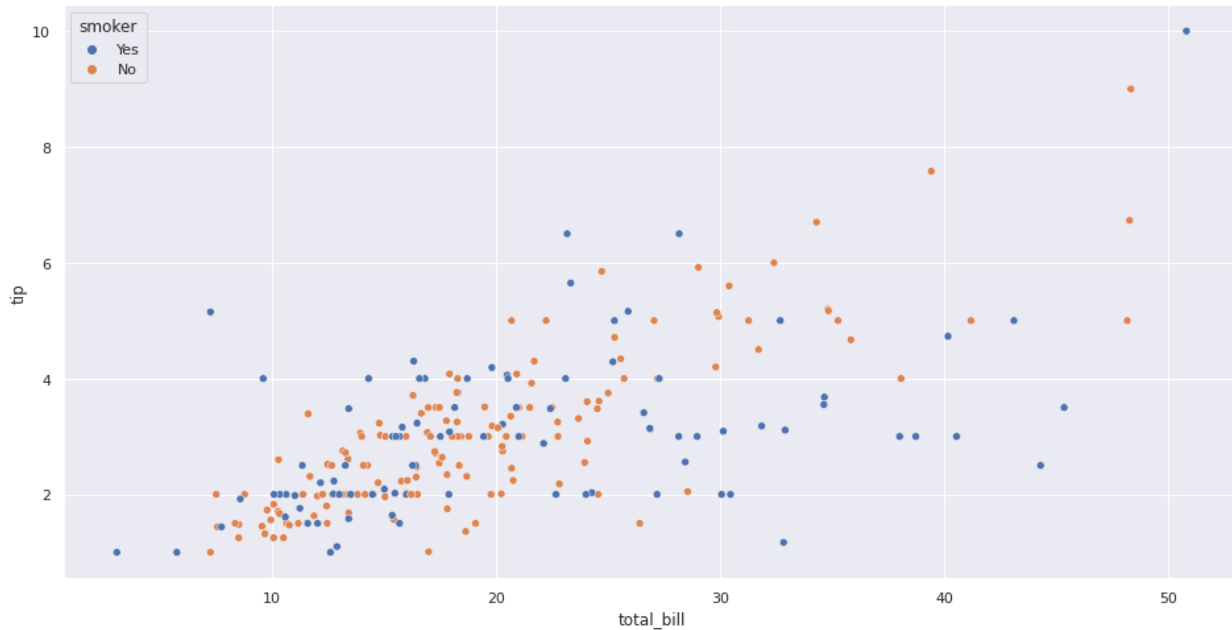
```
tips_df['tip_per_person'] = tips_df.tip/tips_df['size']
ax = sns.scatterplot(data=tips_df, x="total_bill", y="tip_per_person")
```



It is much harder to see a relation now, so we could argue that depending of the service everyone gives a specific amount. So it is not the size of the bill that is defining the tip, rather the amount of persons (although this is very similar) in the bill.

Aside from feature engineering, we can also add categorical features, using different colors for each feature. Here we added if they smoked or not.

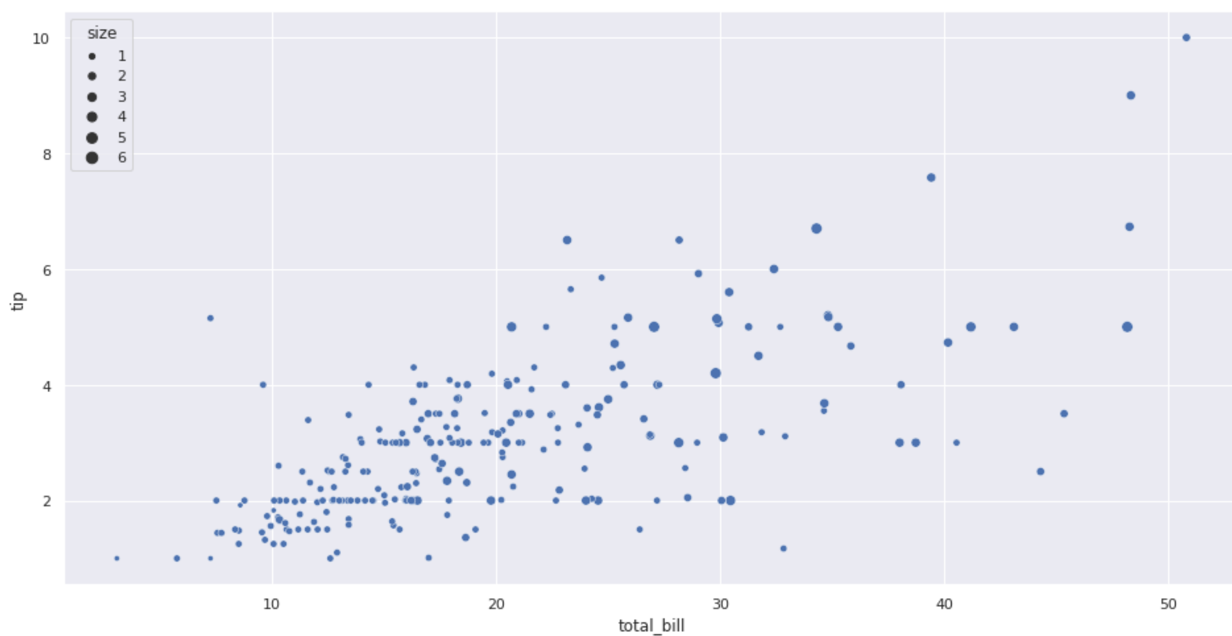
```
ax = sns.scatterplot(data=tips_df, x="total_bill", y="tip", hue='smoker')
```



It is hard to see if smoking had an effect on either the bill or the tip, which indicates that your plot is not that useful. This is not true if you wanted to prove that there is no effect of smoking obviously!

We can also add a numerical feature into the scatter plot, by using sizes of our dots in the scatter plot. The size of the group now influences the size of our dots.

```
ax = sns.scatterplot(data=tips_df, x="total_bill", y="tip", size="size")
```

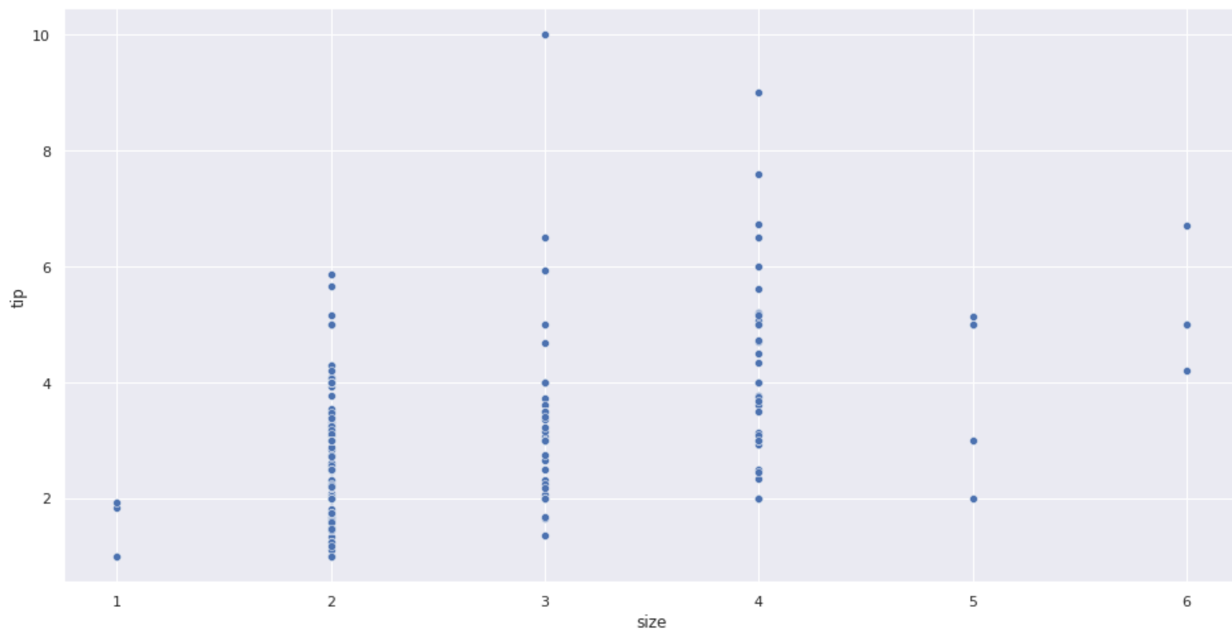


Whilst it might not be really visible because of the linear nature of the size - it is only going from 1 to 6 - the relation is not obvious. Perhaps you could do some feature engineering where you artificially increase the size by taking the square?

You could argue if that is still representable, but for the sake of the exercise let's say it is.

In the beginning I talked about numerical features with a low range, the size of our group is one of them. See what happens when i would use it in a scatter plot.

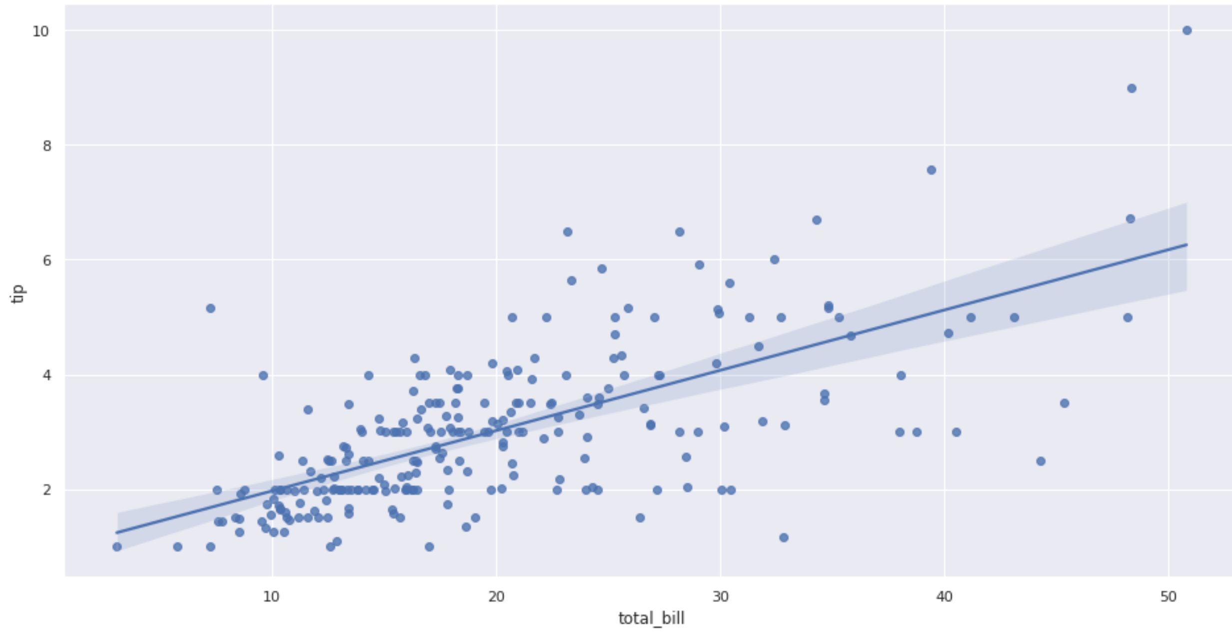
```
ax = sns.scatterplot(data=tips_df, x="size", y="tip")
```



It clearly shows that a higher size means statistical higher tips, up to a cut-off of 5 appearantly. Yet do you feel this is an aesthetically satisfying plot?

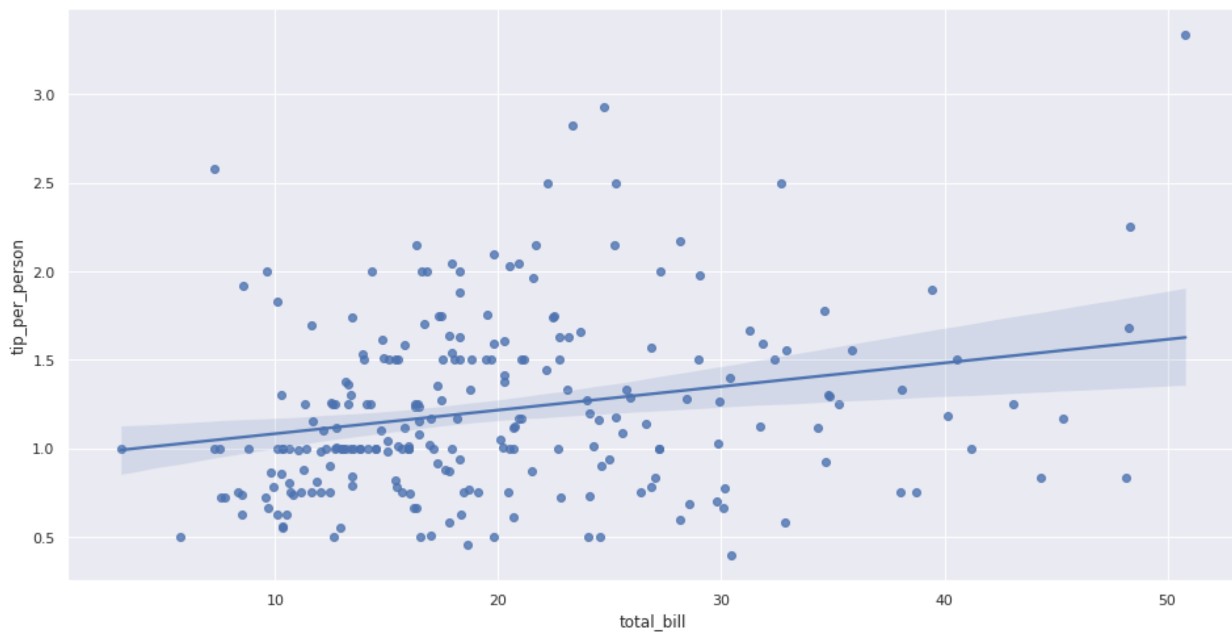
Not going to much in the field of machine learning, seaborn has an interesting feature built-in. They offer a regression plot, where a linear regression is draw with a confidence interval (the light blue area). Not wanting to give mathematical number it shows what it thinks is the relation between the 2 variables.

```
ax = sns.regplot(data=tips_df, x="total_bill", y="tip")
```



It seems to be very confident about the relation, how about where we corrected for group size?

```
ax = sns.regplot(data=tips_df, x="total_bill", y="tip_per_person")
```



Less confident, less appearent. Keep in mind that it will always see a relations, the question is how confident!

CHAPTER

TWENTYFOUR

HEATMAP PLOT

A heatmap also deals with 2 dimensional data and cares about the relation. Here instead of numerical data with dots, we are using categorical data where every combination of the 2 categories has a singular value.

This results into a matrix that we visualize where each index of the matrix has its own color based on a color gradient. This plot got its name as it is used to find 'hot spots' between combinations of 2 categorical features.

```
import pandas as pd
import seaborn as sns
sns.set_theme()
sns.set(rc={'figure.figsize':(16,12)})
```

To make optimal use of this plot, we are going to take on a rather complex dataset, where we have measurements of brain networks. The idea is that we have several networks with several nodes in 2 hemispheres, the content of the data is not as important here, what matters is that we want to find correlations between different nodes in the brain.

```
brain_df = sns.load_dataset("brain_networks", header=[0, 1, 2], index_col=0)
brain_df.head()
```

```
network          1                     2                     3              \
node             1                     1                     1
hemi            lh         rh         lh         rh         lh         rh
0        56.055744  92.031036   3.391576  38.659683  26.203819 -49.715569
1        55.547253  43.690075 -65.495987 -13.974523 -28.274963 -39.050129
2        60.997768  63.438793 -51.108582 -13.561346 -18.842947  -1.214659
3        18.514868  12.657158 -34.576603 -32.665958  -7.420454  17.119448
4        -2.527392 -63.104668 -13.814151 -15.837989 -45.216927   3.483550

network          4                     5             ...         16  \
node             1                     1             ...          3
hemi            lh         rh         lh         rh  ...         rh
0        47.461037  26.746613 -35.898861  -1.889181  ...   0.607904
1        -1.210660 -19.012897  19.568010  15.902983  ...  57.495071
2       -65.575806 -85.777428  19.247454  37.209419  ...  28.317369
3       -41.800869 -58.610184  32.896915  11.199619  ...  71.439629
4       -62.613335 -49.076508  18.396759   3.219077  ...  95.597565

network                                17                                \
node             4                      1                     2
hemi            lh          rh         lh         rh         lh         rh
0       -70.270546   77.365776 -21.734550   1.028253   7.791784  68.903725
1       -76.393219  127.261360 -13.035799  46.381824 -15.752450  31.000332
2         9.063977   45.493263  26.033442  34.212200   1.326110 -22.580757
3        65.842979  -10.697547  55.297466   4.255006  -2.420144  12.098393
```

(continues on next page)

(continued from previous page)

```
4        50.960453  -23.197300  43.067562  52.219875  28.232882 -11.719750

network
node            3                     4
hemi           lh          rh         lh
0       -10.520872  120.490463 -39.686432
1       -39.607521   24.764011 -36.771008
2        12.985169  -75.027451   6.434262
3       -15.819172  -37.361431  -4.650954
4         5.453649    5.169828  87.809135

[5 rows x 62 columns]
```

luckily for us, the pandas library has an easy method of finding out what the correlation is between different columns of numerical data. These correlations are denoted between -1 (completely opposite) to 1 (completely related). Take a minute to understand how the columns and index changed using the operation, you can see that a node in a network and hemisphere has a correlation of 1.00 with itself.

```
brain_df.corr()
```

```
network                   1                   2                   3           \
node                      1                   1                   1
hemi                     lh        rh        lh        rh        lh        rh
network node hemi
1       1    lh    1.000000  0.881516 -0.042699 -0.074437 -0.342849 -0.169498
             rh    0.881516  1.000000  0.013073  0.033733 -0.351509 -0.162006
2       1    lh   -0.042699  0.013073  1.000000  0.813394 -0.006940 -0.039375
             rh   -0.074437  0.033733  0.813394  1.000000 -0.027324 -0.023608
3       1    lh   -0.342849 -0.351509 -0.006940 -0.027324  1.000000  0.553183
...                     ...       ...       ...       ...       ...       ...
17      2    lh   -0.206379 -0.273370 -0.151724 -0.224447  0.026579 -0.056687
             rh   -0.212601 -0.266456 -0.124508 -0.172704 -0.089109 -0.144020
        3    lh   -0.142770 -0.174222 -0.179912 -0.250455 -0.012675 -0.047434
             rh   -0.204326 -0.223572 -0.044706 -0.090798 -0.024644 -0.103875
        4    lh   -0.219283 -0.273626 -0.209557 -0.216674  0.013747 -0.058838

network                   4                   5            ...        16  \
node                      1                   1            ...         3
hemi                     lh        rh        lh        rh  ...        rh
network node hemi                                          ...
1       1    lh   -0.373050 -0.361726  0.431619  0.418708  ... -0.106642
             rh   -0.333244 -0.337476  0.431953  0.519916  ... -0.173530
2       1    lh   -0.019773  0.007099 -0.147374 -0.104164  ... -0.215429
             rh   -0.017577 -0.014632 -0.173501 -0.094717  ... -0.184458
3       1    lh    0.528787  0.503403 -0.157154 -0.185008  ... -0.146451
...                     ...       ...       ...       ...  ...       ...
17      2    lh    0.020064  0.084837 -0.359879 -0.394522  ...  0.173117
             rh    0.007278  0.029909 -0.299152 -0.295150  ...  0.299440
        3    lh    0.070114  0.100063 -0.245179 -0.303354  ... -0.055529
             rh    0.101791  0.128318 -0.302654 -0.277378  ...  0.079460
        4    lh   -0.069100 -0.031653 -0.282767 -0.279381  ...  0.418857

network                                       17                           \
node                      4                   1                   2
hemi                     lh        rh        lh        rh        lh        rh
network node hemi
```

(continues on next page)

(continued from previous page)

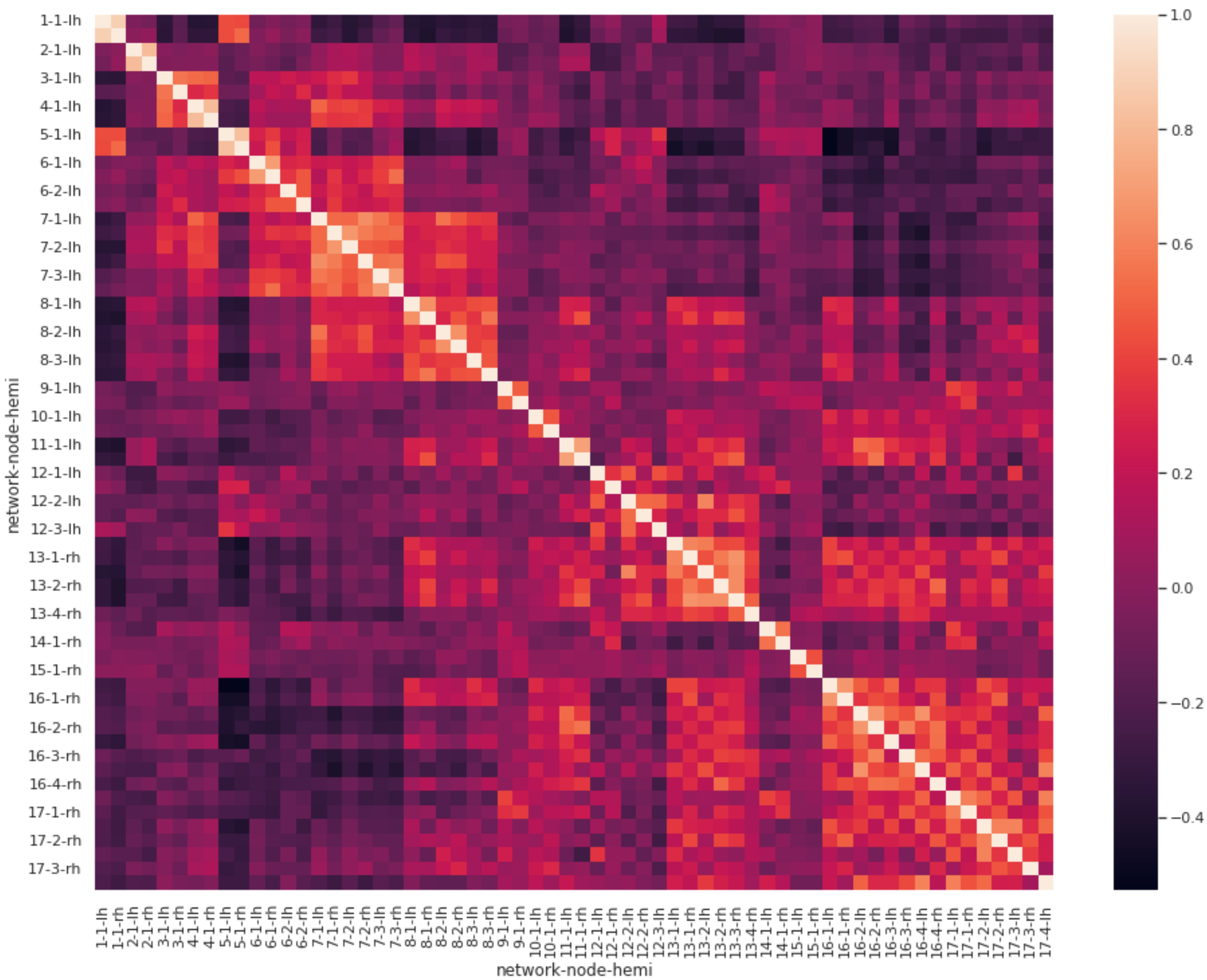
```
1       1    lh   -0.162254 -0.232501 -0.099781 -0.161649 -0.206379 -0.212601
             rh   -0.224436 -0.277954 -0.212964 -0.262915 -0.273370 -0.266456
2       1    lh   -0.239876 -0.093679 -0.240455 -0.190721 -0.151724 -0.124508
             rh   -0.244956 -0.061151 -0.255101 -0.169402 -0.224447 -0.172704
3       1    lh   -0.033931 -0.156972 -0.015964 -0.149944  0.026579 -0.089109
...                     ...       ...       ...       ...       ...       ...
17      2    lh    0.478606  0.258958  0.499351  0.319184  1.000000  0.597620
             rh    0.204444  0.453497  0.272868  0.440901  0.597620  1.000000
        3    lh    0.259191  0.046663  0.454838  0.188905  0.601382  0.345253
             rh    0.005291  0.296318  0.087061  0.224760  0.319382  0.456019
        4    lh    0.603491  0.172167  0.589364  0.451264  0.517481  0.256544

network
node                     3                   4
hemi                    lh        rh        lh
network node hemi
1       1    lh   -0.142770 -0.204326 -0.219283
             rh   -0.174222 -0.223572 -0.273626
2       1    lh   -0.179912 -0.044706 -0.209557
             rh   -0.250455 -0.090798 -0.216674
3       1    lh   -0.012675 -0.024644  0.013747
...                     ...       ...       ...
17      2    lh    0.601382  0.319382  0.517481
             rh    0.345253  0.456019  0.256544
        3    lh    1.000000  0.379705  0.264381
             rh    0.379705  1.000000  0.090302
        4    lh    0.264381  0.090302  1.000000

[62 rows x 62 columns]
```

This result is way to much to see a pattern, yet if we add a color scale and give each a gradation, we can see some correlations.

Can you see how nodes from the same network are related with a more whitish color? The heatmap might be fairly intimidating at first but is a powerful tool when handling bigger datasets.

```
sns.heatmap(data=brain_df.corr())
```

```
<AxesSubplot:xlabel='network-node-hemi', ylabel='network-node-hemi'>
```



Without going into the medical details we can also apply some machine learning to it and create a clustermap. This map is a way to group nodes from similar networks into clusters, an advances technique!

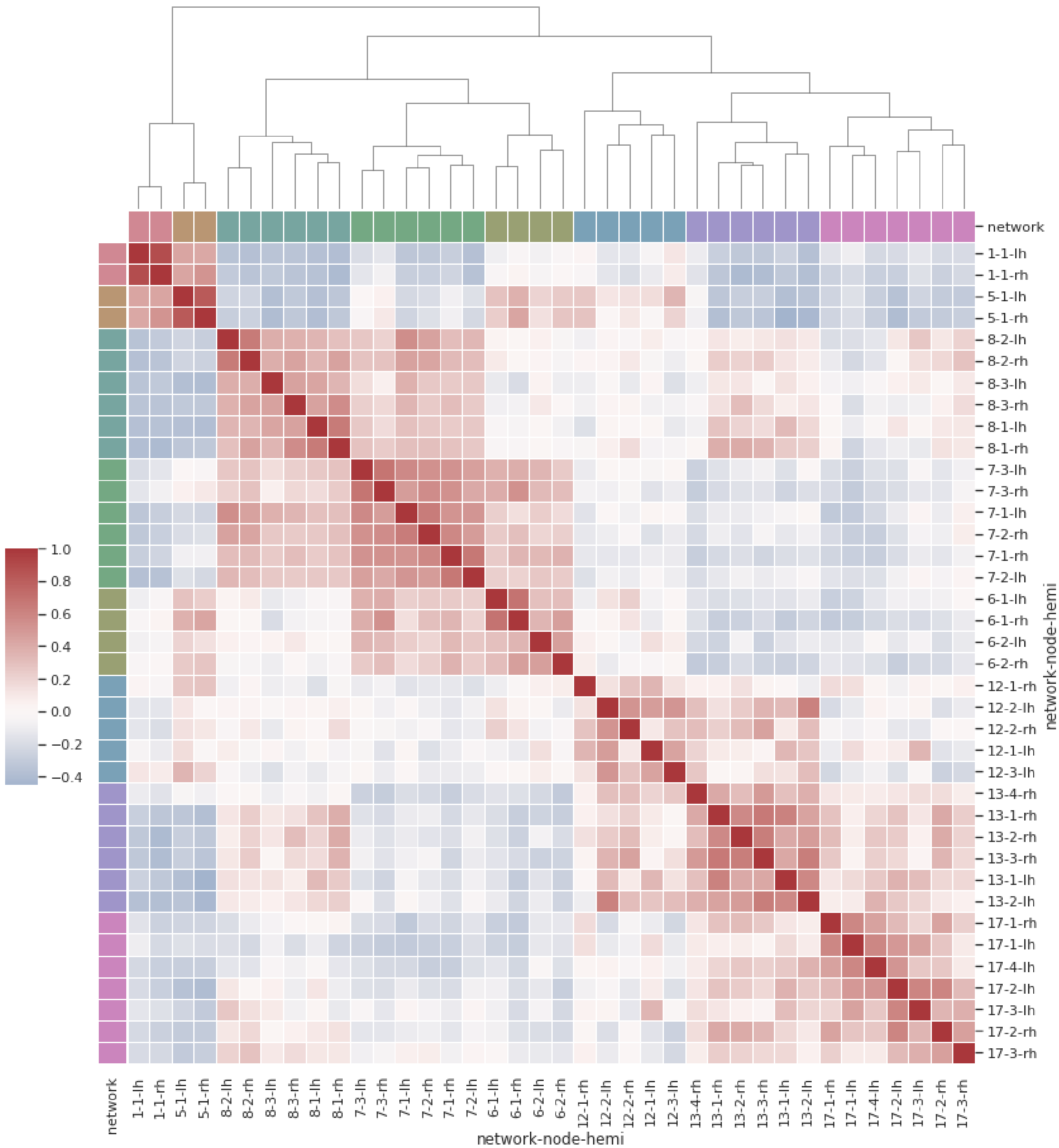
Gaze over the colors and look at the axi, notice how the computer figured out how to group the most similar nodes from networks. Also, I did not create this by myself, so don't give me credit for this!

```
# Select a subset of the networks
used_networks = [1, 5, 6, 7, 8, 12, 13, 17]
used_columns = (brain_df.columns.get_level_values("network")
                          .astype(int)
                          .isin(used_networks))
brain_df = brain_df.loc[:, used_columns]

# Create a categorical palette to identify the networks
network_pal = sns.husl_palette(8, s=.45)
network_lut = dict(zip(map(str, used_networks), network_pal))

# Convert the palette to vectors that will be drawn on the side of the matrix
networks = brain_df.columns.get_level_values("network")
network_colors = pd.Series(networks, index=brain_df.columns).map(network_lut)

# Draw the full plot
g = sns.clustermap(brain_df.corr(), center=0, cmap="vlag",
```

(continues on next page)

(continued from previous page)

```
                   row_colors=network_colors, col_colors=network_colors,
                   dendrogram_ratio=(.1, .2),
                   cbar_pos=(.02, .32, .03, .2),
                   linewidths=.75, figsize=(12, 13))

g.ax_row_dendrogram.remove()
```



Part V

5. Data Exploration

CHAPTER TWENTYFIVE

INTRODUCTION

this is an introduction

CHAPTER

TWENTYSIX

VARIABLE IDENTIFICATION

in this notebook we are going to look into a few simple but interesting techniques about getting to know more about what is inside the dataset you are given. Whenever you start out on a new project these steps are usually the first that are performed in order to know how to proceed.

We start out by loading the titanic dataset from seaborn

```
import seaborn as sns
titanic_df = sns.load_dataset('titanic')
sns.set_theme()
sns.set(rc={'figure.figsize':(16,12)})
```

26.1 description

Let us start out simple and retrieve information about each column, using the .info method we can get non-null counts (giving us an idea if there are nans) and the type of each column (to see if we need to change types).

```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

it looks like all types are already correctlyaddressed, but we can see a lot of nans are present for age and deck, this might be a problem!

For numerical columns we can get a bunch of information using the .describe method. this can also be used for categories but has less info

```
titanic_df.describe()
```

```
         survived      pclass         age       sibsp       parch        fare
count  891.000000  891.000000  714.000000  891.000000  891.000000  891.000000
mean     0.383838    2.308642   29.699118    0.523008    0.381594   32.204208
std      0.486592    0.836071   14.526497    1.102743    0.806057   49.693429
min      0.000000    1.000000    0.420000    0.000000    0.000000    0.000000
25%      0.000000    2.000000   20.125000    0.000000    0.000000    7.910400
50%      0.000000    3.000000   28.000000    0.000000    0.000000   14.454200
75%      1.000000    3.000000   38.000000    1.000000    0.000000   31.000000
max      1.000000    3.000000   80.000000    8.000000    6.000000  512.329200
```

```
titanic_df.describe(include=['category', 'object'])
```

```
         sex embarked  class  who deck  embark_town alive
count    891      889    891  891  203          889   891
unique     2        3      3    3    7            3     2
top     male        S  Third  man    C  Southampton    no
freq     577      644    491  537   59          644   549
```

26.2 Uniques, frequencies and ranges

the describe method is a bit lacklusting for categorical features, so we use some good old data wrangling to get more info, asking for unique values gives us all the possible values for a column. Aside from the uniques, we can also get the value counts or frequencies and the range of a column.

```
titanic_df['embark_town'].unique()
```

```
array(['Southampton', 'Cherbourg', 'Queenstown', nan], dtype=object)
```

```
titanic_df['embark_town'].value_counts()
```

```
Southampton    644
Cherbourg      168
Queenstown      77
Name: embark_town, dtype: int64
```

```
titanic_df['age'].min(), titanic_df['age'].max()
```

```
(0.42, 80.0)
```

26.3 mean and deviation

to get more information about a numerical range, we calculate the mean and deviation. Note that these statistics imply that our column is normally distributed!

You can also see that I applied the dropna method, this because the calculations cannot handle nan values, but this means our outcome might be distorted from the truth, thread carefuly.

```
import statistics
```

```
titanic_df['age'].dropna().mean()
```

```
29.69911764705882
```

```
titanic_df['age'].dropna().median()
```

```
28.0
```

26.4 median and interquantile range

When our distribution is not normal, using the median and IQR is advised. First we apply the shapiro wilk test and it has a very low p-value (the second value) which means we can reject the null-hypothesis that there is a normal distribution. more info about shapiro-wilk can be found on wikipedia

```
from scipy.stats import shapiro
shapiro(titanic_df['age'].dropna())
```

```
ShapiroResult(statistic=0.9814548492431641, pvalue=7.322165629375377e-08)
```

```
titanic_df['age'].dropna().median()
```

```
28.0
```

```
from scipy.stats import iqr
iqr(titanic_df['age'].dropna())
```

```
17.875
```

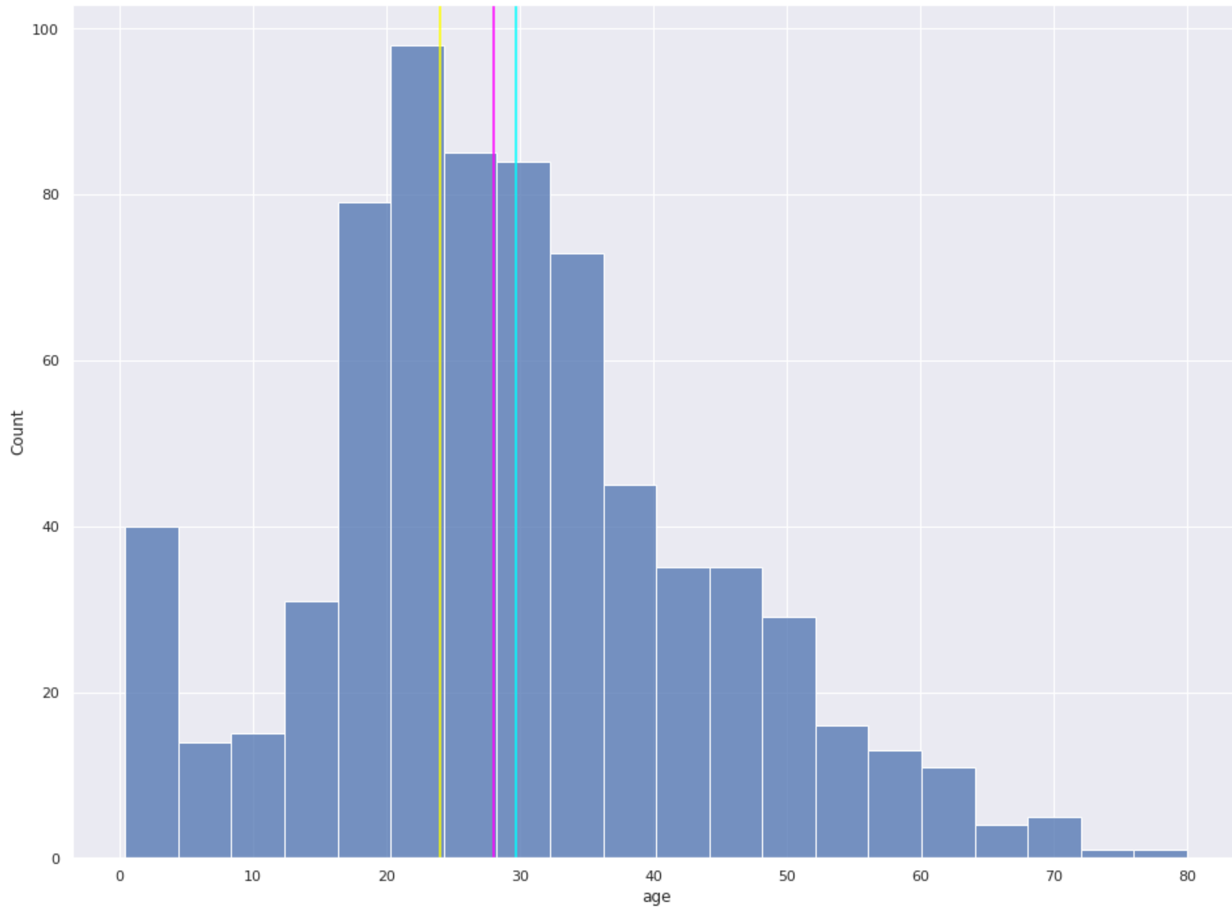
```
from scipy.stats.mstats import mquantiles
mquantiles(titanic_df['age'].dropna())
```

```
array([20., 28., 38.])
```

Appearently the average of 29.70 is fairly higher than the median at 28, meaning that there is a shift towards older people. You can also see this on the following plot, where we note the mean, median and mode.

```
ax = sns.histplot(data=titanic_df, x='age')

ax.axvline(titanic_df.age.mean(), color='cyan')
ax.axvline(titanic_df.age.median(), color='magenta')
ax.axvline(titanic_df.age.mode()[0], color='yellow')
```

```
<matplotlib.lines.Line2D at 0x7fa1b3657e50>
```



26.5 modes and frequencies

When we don't have numerical data we can still find some interesting results, here we use the mode (most frequent value) and the proporties of each value to deduce the proporties of people that embarked in the 3 different towns. Nearly 3/4 people embarked in one harbour.

```
titanic_df['embark_town'].mode()
```

```
0    Southampton
dtype: object
```

```
titanic_df['embark_town'].value_counts()/len(titanic_df)
```

```
Southampton    0.722783
Cherbourg      0.188552
Queenstown     0.086420
Name: embark_town, dtype: float64
```

CHAPTER TWENTYSEVEN

UNI-VARIATE ANALYSIS

In this notebook we will go a bit deeper into the analysis of a single column or variable of our dataset. This means we will be looking into how visualisations might be useful to attain more information. We start out again by loading the titanic dataset and obtaining the same info as before.

```
import seaborn as sns
titanic_df = sns.load_dataset('titanic')
sns.set_style()
sns.set(rc={'figure.figsize':(16,12)})
```

```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

27.1 Nominal data

Lets take a look into a norminal column, the embark town has 3 different options and we already saw how to count the values and calculate proportions.

```
titanic_df['embark_town'].value_counts()
```

```
Southampton    644
Cherbourg      168
Queenstown      77
Name: embark_town, dtype: int64
```

```
titanic_df['embark_town'].value_counts()/len(titanic_df)
```

```
Southampton    0.722783
Cherbourg      0.188552
Queenstown     0.086420
Name: embark_town, dtype: float64
```

```
import statistics
statistics.mode(titanic_df['embark_town'])
```

```
'Southampton'
```

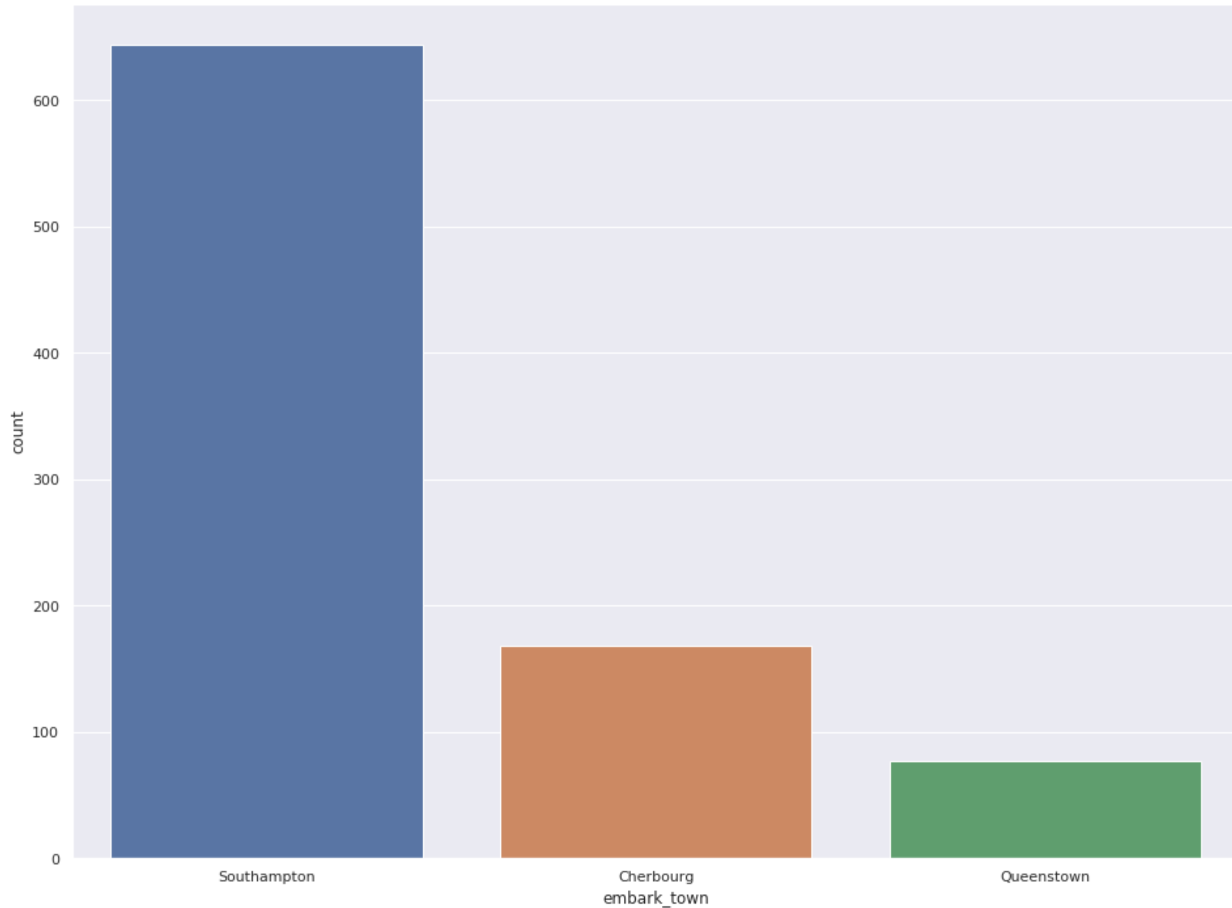
```
statistics.median(titanic_df['embark_town'])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_11541/1735617235.py in <module>
----> 1 statistics.median(titanic_df['embark_town'])

/usr/lib/python3.8/statistics.py in median(data)
    425
    426     """
--> 427     data = sorted(data)
    428     n = len(data)
    429     if n == 0:

TypeError: '<' not supported between instances of 'float' and 'str'
```

Hmmm, it seems we can not take the median because python does not know the order of the categories. Let's kick it up a notch and use some plots to make these proportions more clear, we'll use a bar chart to do this.

```
sns.countplot(data=titanic_df, x='embark_town')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e7f61d100>
```

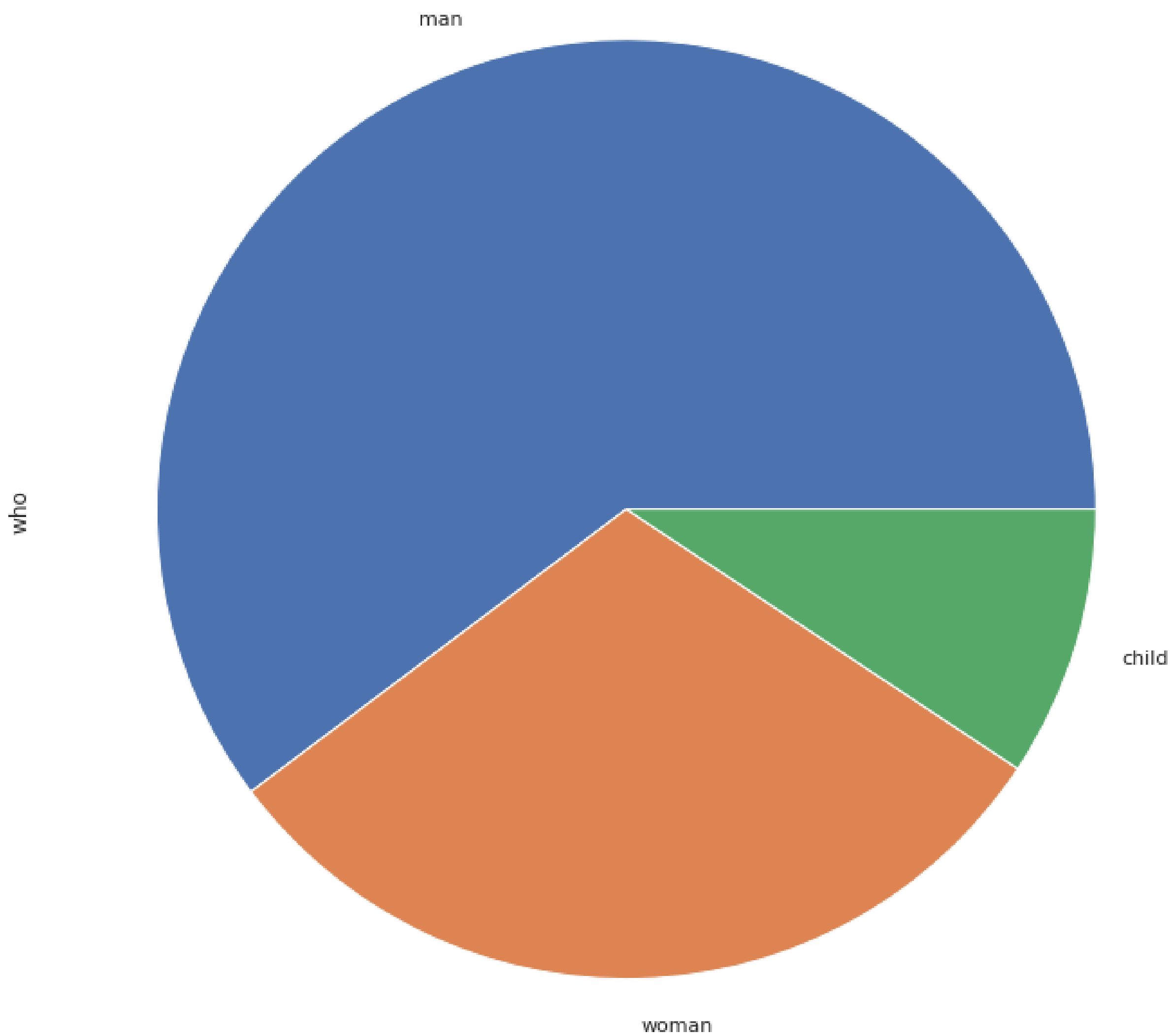


Something important that I would like to mention here is that this serves as a method to validate sample size, if e.g. only a handful persons would embark on a location, the statistics in this group will have a high variance which will not always shot in your visualisations. Be mindful to check sample sizes of categories when applying statistics.

The bar chart is ideal to compare the values to eachother, yet if we would like to visualise the proportions to eachother, we need a pie plot. Here we use the 'who' feature containing information about the person itself, we have 3 categories: man, woman, child.

```
titanic_df.who.value_counts().plot.pie()
```

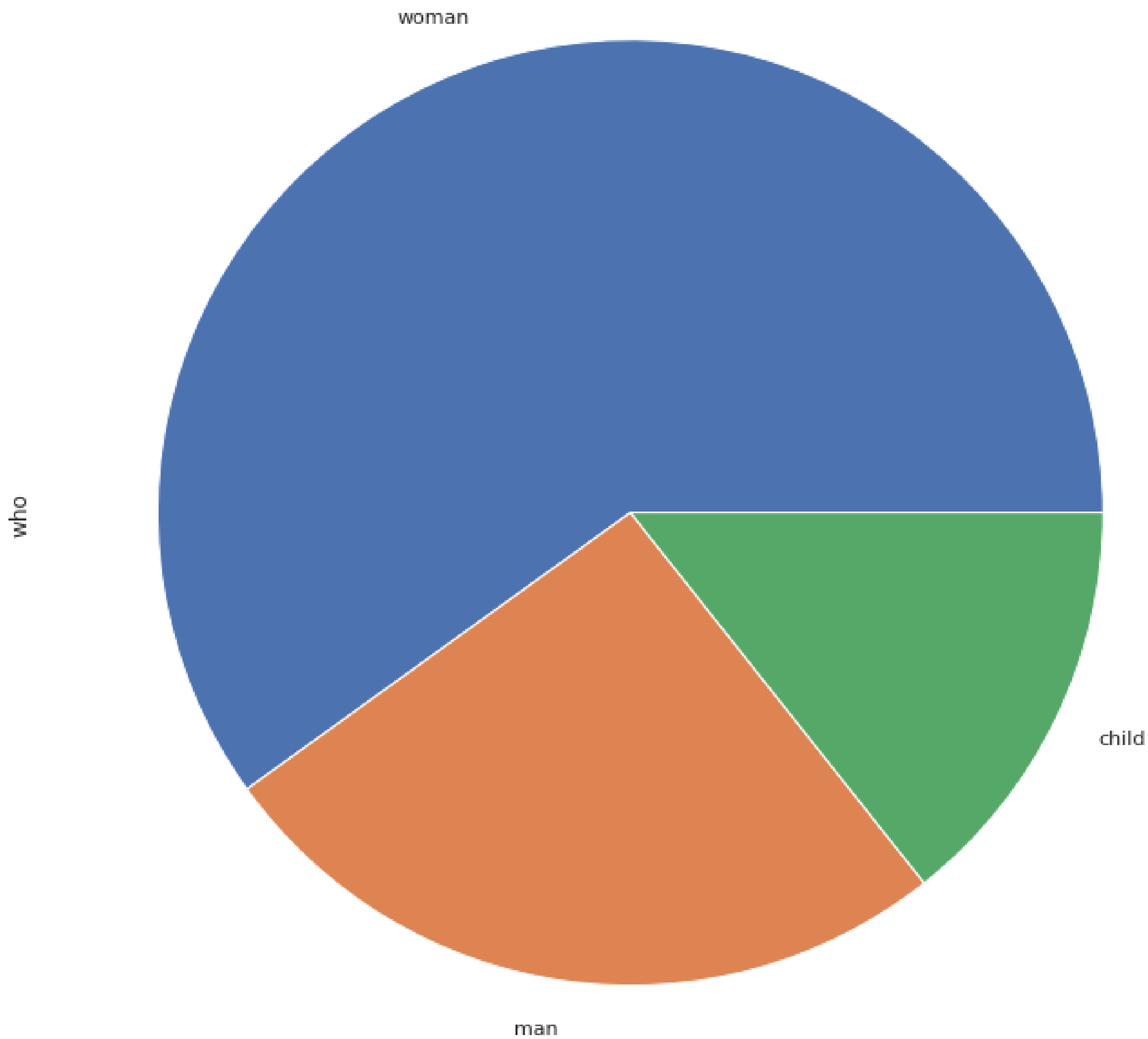
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e7d2ca430>
```



The saying goes 'Woman and children first' which would mean the survivors are mainly those 2 groups, let us confirm that by subselecting only the survivors and recreate the pie plot.

```
titanic_df[titanic_df.survived==1].who.value_counts().plot.pie()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e7d285dc0>
```



You can see that the groups are now reversed, where men are proportionally less represented. By using a pie plot we circumvent the problem where we have a bias towards size of our dataset, the pie plot applies scaling by itself.

27.2 Ordinal data

Whilst there was no order in the town where passengers embarked, there is in the class of the ticket they bought. So we need to keep this in mind when exploring. We can not just say they belonged to any class as there is a difference in these classes! However the same statistics apply, but with a different twist.

```
titanic_df['class'].value_counts()
```

```
Third     491
First     216
```

(continues on next page)

(continued from previous page)

```
Second    184
Name: class, dtype: int64
```

```
titanic_df['class'].value_counts()/len(titanic_df)
```

```
Third     0.551066
First     0.242424
Second    0.206510
Name: class, dtype: float64
```

it seems more people travelled on the titanic in first class than second class! nothing you would see nowadays.

```
statistics.mode(titanic_df['class'])
```

```
'Third'
```

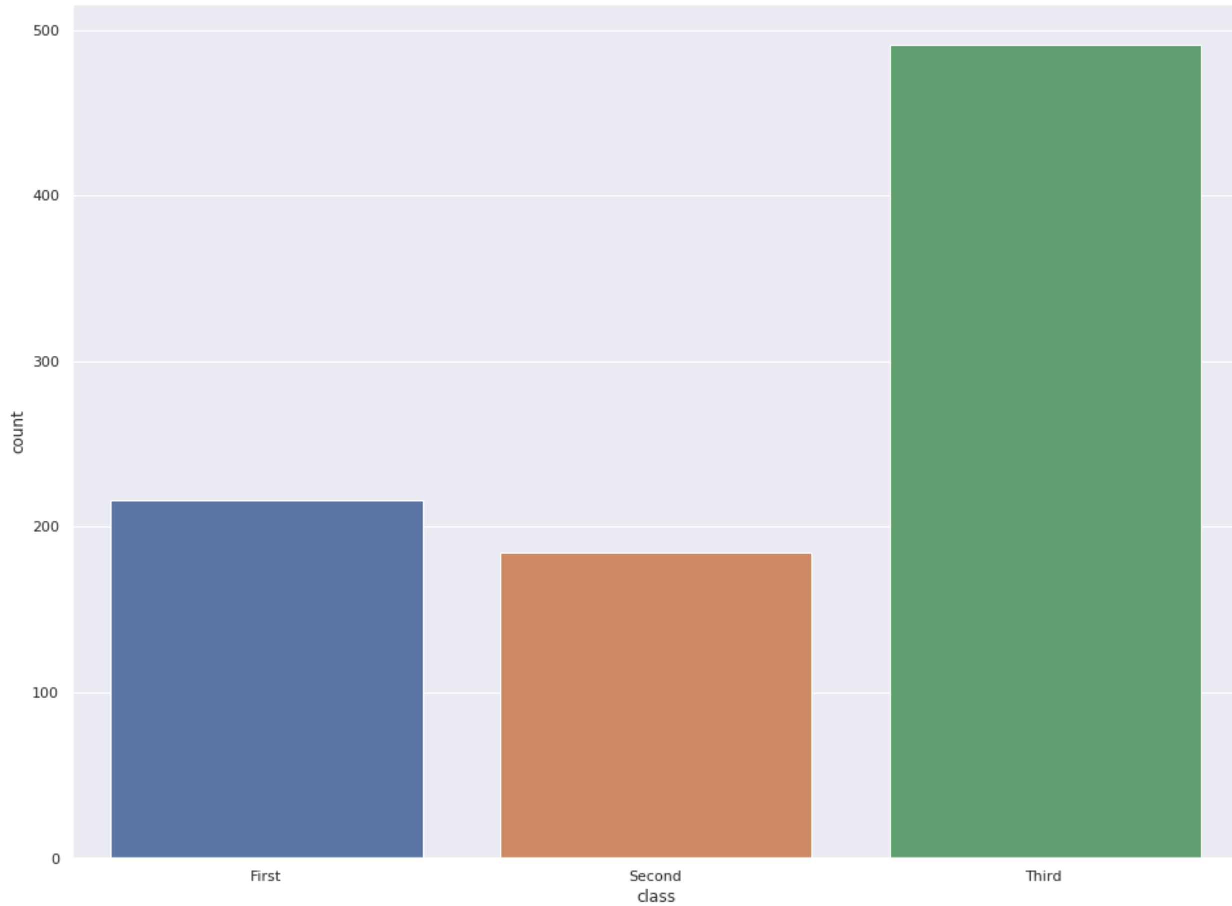
```
statistics.median(titanic_df['class'])
```

```
'Third'
```

Here we can use the median, as there is an order in the classes! By using a bar plot we can visualise the distribution, because the graphing library knows the order of the categories, they will also be properly displayed, how convenient.

```
sns.countplot(data=titanic_df, x='class')
```

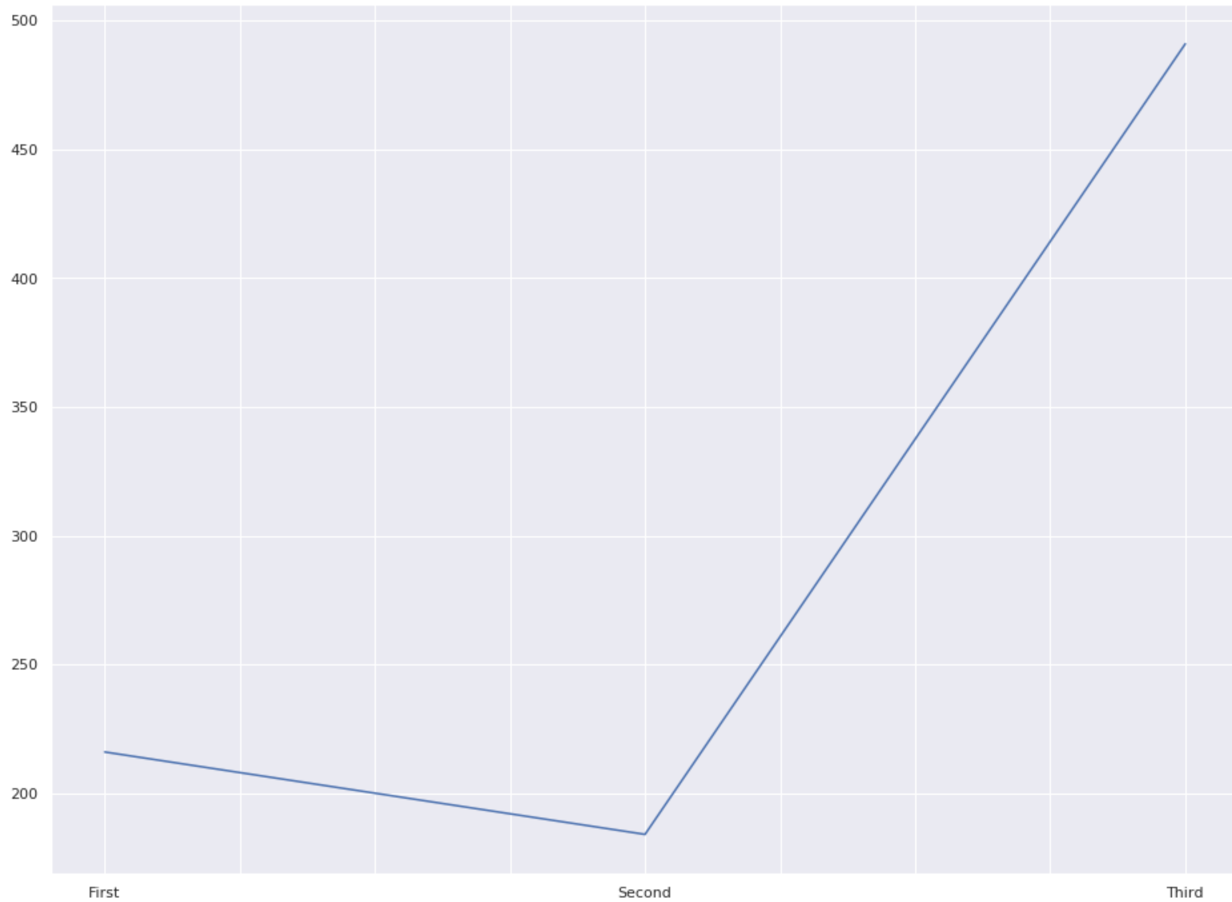
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e7d250580>
```



You could however also create a line plot with this, as there is a relation between the classes, as shown below.

```
titanic_df['class'].value_counts()[['First', 'Second', 'Third']].plot()
```

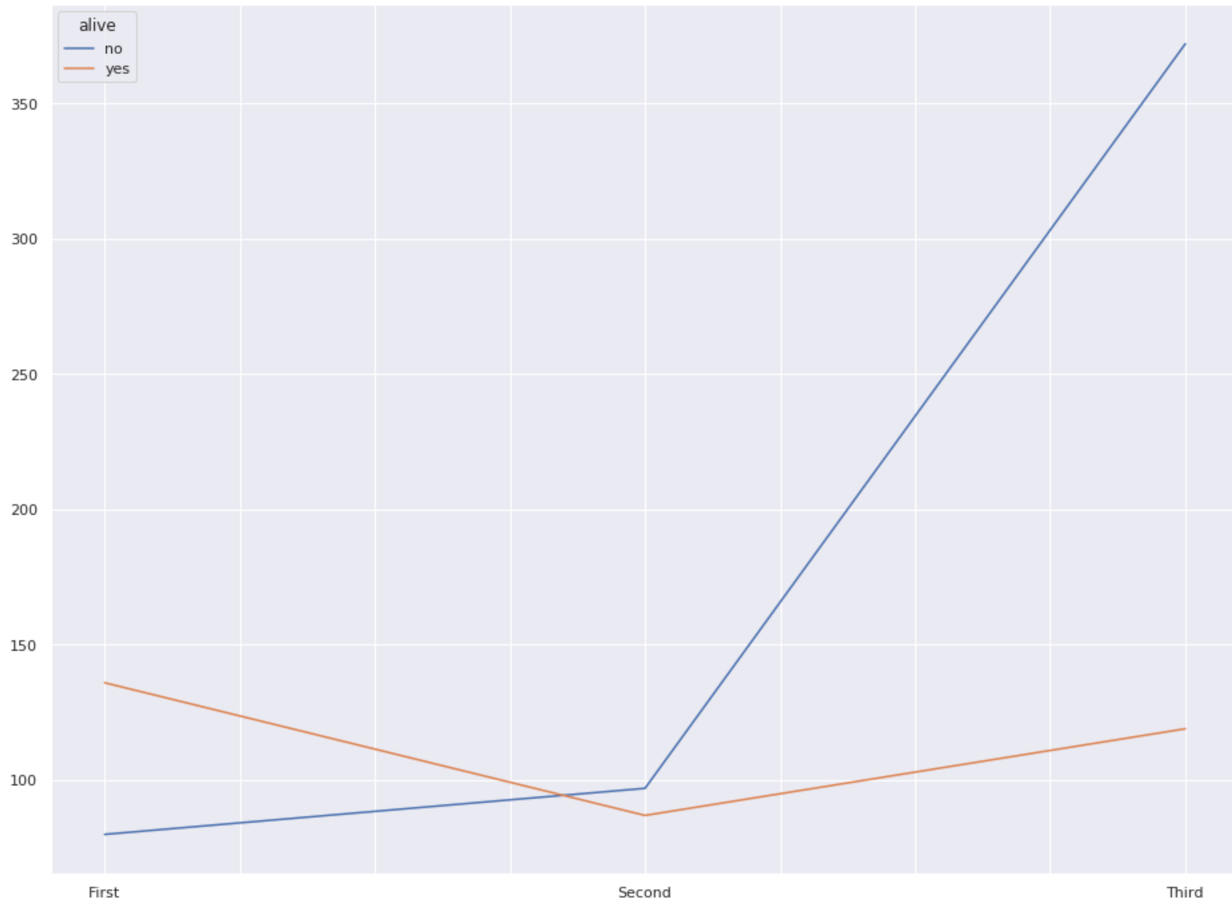
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e7d224370>
```



This plot feels underwhelming with only 3 points, but we could make it more interesting, we divide our data on who survived and count the amount of persons per class that survived or not. It is clear to say the a higher class meant higher chances of survival.

```
titanic_df['class'].groupby(titanic_df.alive).apply(lambda x: x.value_counts()).
↪unstack(0).reindex(['First', 'Second', 'Third']).plot()
```

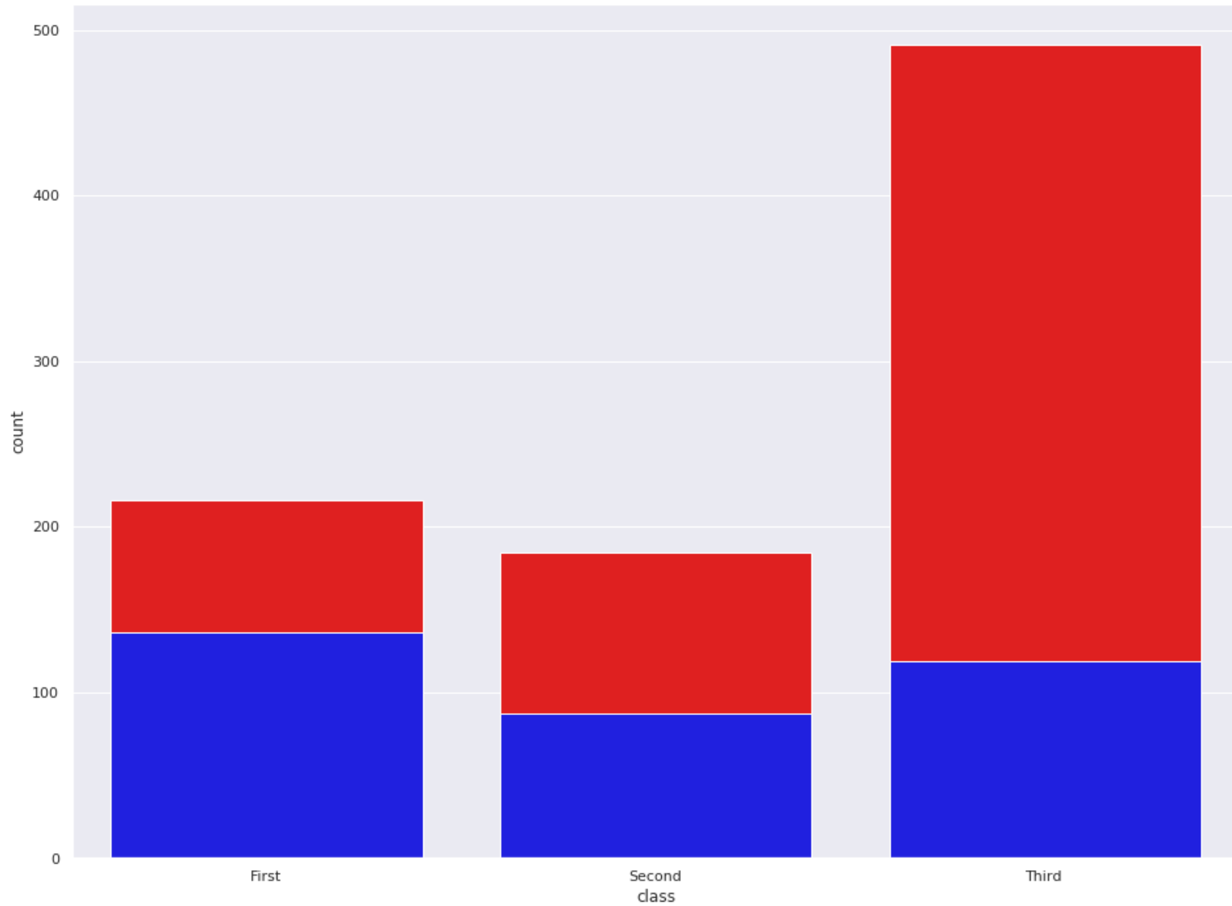
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e7d1f9550>
```



Personally as there is no time related factor in our x-axis, the line or parallel plot here is not as convenient. Since we have a situation where there is a confinement that the amount of survived can not be more that the total, I would opt for a bar plot, which is show below.

```
sns.countplot(x = 'class', data = titanic_df, color = 'red')
sns.countplot(x = 'class', data = titanic_df[titanic_df.survived==1], color = 'blue')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e7d278b50>
```



27.3 Continuous data

After categories which are discrete we also have continuous data, which is by nature always ordered. Here we can perform all the other statistical methods along with the mean, but again keep in mind that using the mean does come with a lot of responsibility.

```
statistics.mode(titanic_df['age'])
```

```
24.0
```

```
statistics.median(titanic_df['age'].dropna())
```

```
28.0
```

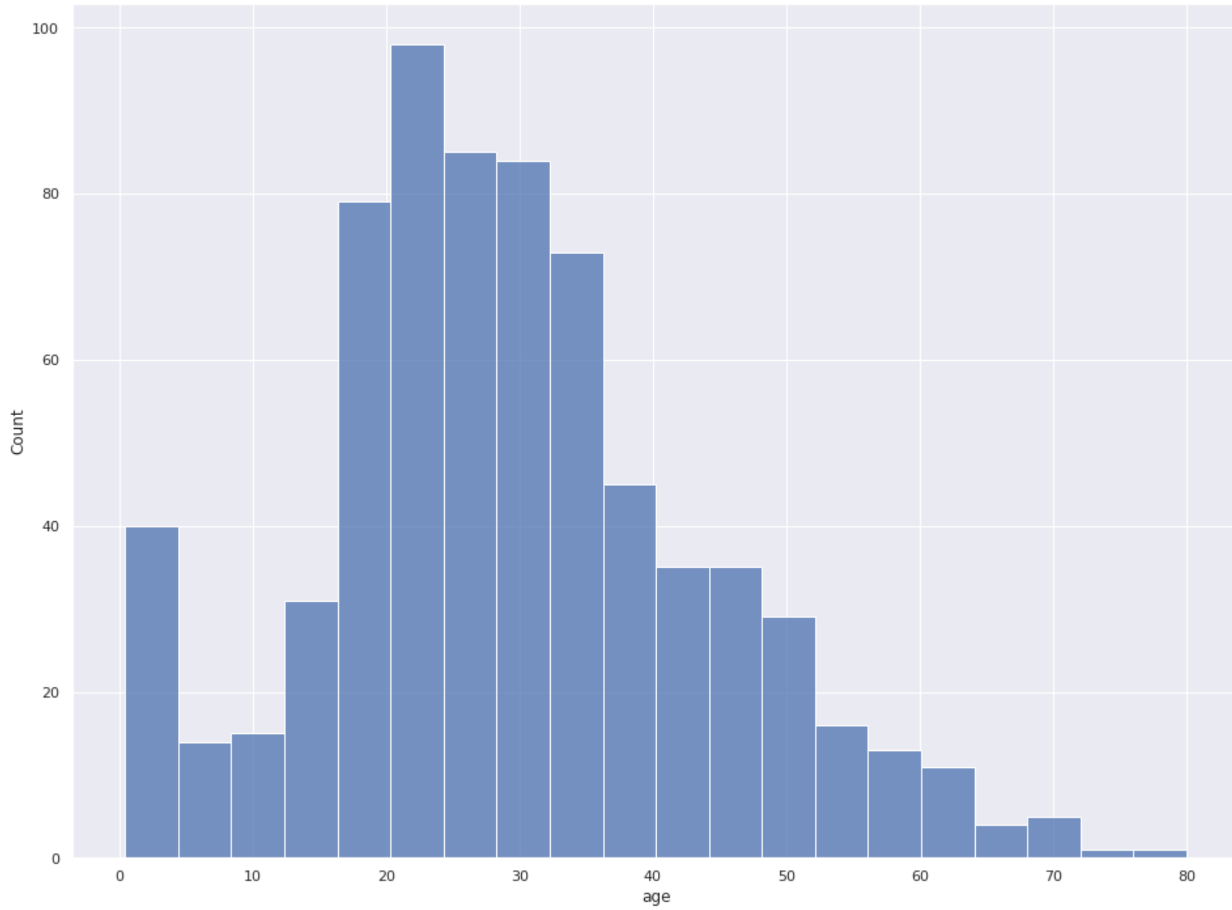
```
statistics.mean(titanic_df['age'].dropna())
```

```
29.699117647058824
```

A very potent method of showing the distribution is a histogram or distribution plot as shown below, here we can see the long tail on the right which we correctly predicted earlier when we saw that the mean was slightly higher than the median.

```
sns.histplot(titanic_df['age'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e7ce39f10>
```



going into more mathematical calculations, we can calculate the interquartile ranges, the upper and lower bounds and therefore find any outliers

```
q1, q3 = titanic_df['age'].quantile([0.25, 0.75])
q3-q1
```

```
17.875
```

```
lower_bound = q1 - (1.5 * q1)
upper_bound = q3 + (1.5 * q3)
lower_bound, upper_bound
```

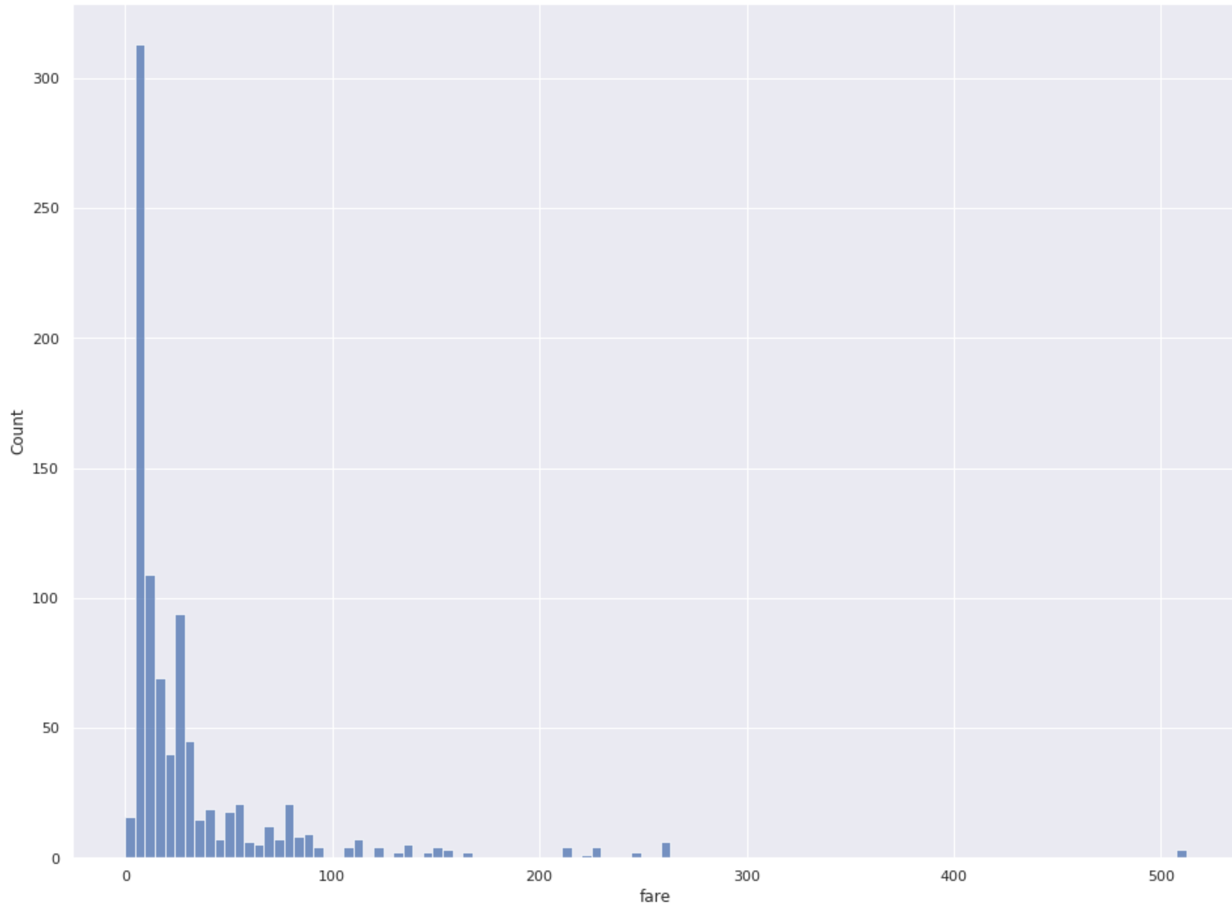
```
(-10.0625, 95.0)
```

It seems that for age, no outliers have been found, which is not really suprising as you don't have any control over your age, unfortunately…

Another numerical feature they had control over was the fare, we give a visualisation of the distrubition here.

```
sns.histplot(titanic_df['fare'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e7cddbf10>
```



This distribution looks horrific, we could also look at the mean and median differences to see this tremendous shift towards higher fares.

```
print('median')
print(titanic_df.fare.median())
print('mean')
print(titanic_df.fare.mean())
```

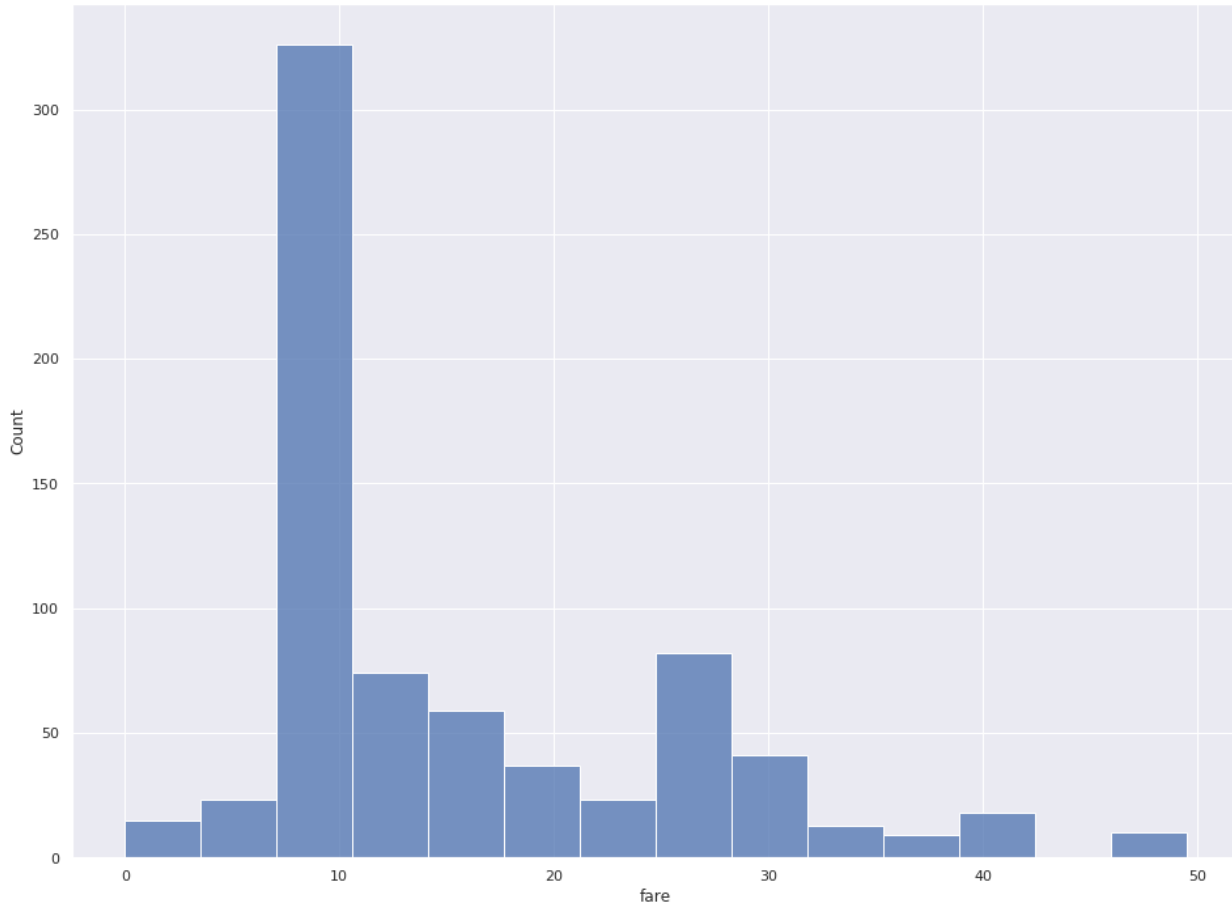
```
median
14.4542
mean
32.204207968574636
```

Perhaps you can use the outlier detection of above to find the upper outlier treshold?

Let us assume that the upper bound for fares is about 50, which is lower than some tickets. By removing these values we can correct our distribution and get a more evened out result. This is especially useful in cases of machine learning where we would not not our algorithm to be biased due to a few extraordinary values, we would have to seperate these specific cases to ensure higher accuracy.

```
sns.histplot(titanic_df[titanic_df.fare<50].fare)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e7cbf1220>
```



Much better, here we can clearly see our values, keep the records with outliers seperate for other purposes. Again looking at the new mean and median we see a lot less difference, indicating a better distribution.

```
print('median')
print(titanic_df[titanic_df.fare<50].fare.median())
print('mean')
print(titanic_df[titanic_df.fare<50].fare.mean())
```

```
median
11.1333
mean
15.500598493150687
```

CHAPTER

TWENTYEIGHT

BI-VARIATE ANALYSIS

In this notebook we are going to look at correlations between two columns in our dataset, this is were it becomes interesting as it opens more opportunities to explore our dataset. We start out by importing necessary libraries and loading the titanic dataset.

```
import seaborn as sns
import pandas as pd
from scipy import stats
titanic_df = sns.load_dataset('titanic')
sns.set_style()
sns.set(rc={'figure.figsize':(16,12)})
```

```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

28.1 Categorical vs categorical

The first comparison we can do is between 2 categorical variables, in this dataset we can use the class of the passenger and the town they embarked the titanic, let's make a contingency table first.

```
contingency_table = pd.crosstab(titanic_df['embark_town'], titanic_df['class'])
contingency_table
```

```
class        First  Second  Third
embark_town
Cherbourg       85      17     66
Queenstown       2       3     72
Southampton    127     164    353
```

With all these numbers it is fairly hard to find if there is a correlation between these 2 variables. Let statistics do the work and get the chi squared test involved, we do not apply a continuity correction as the embarkment is a nominal variable.

The results of the Cramer V test (simplified chi squared test).

```
chi, p, dof, exp = stats.chi2_contingency(contingency_table, correction=False)
chi, p, dof, exp
```

```
(123.75190952951289,
 8.435267819894384e-26,
 4,
 array([[ 40.44094488,  34.77165354,  92.78740157],
        [ 18.53543307,  15.93700787,  42.52755906],
        [155.02362205, 133.29133858, 355.68503937]]))
```

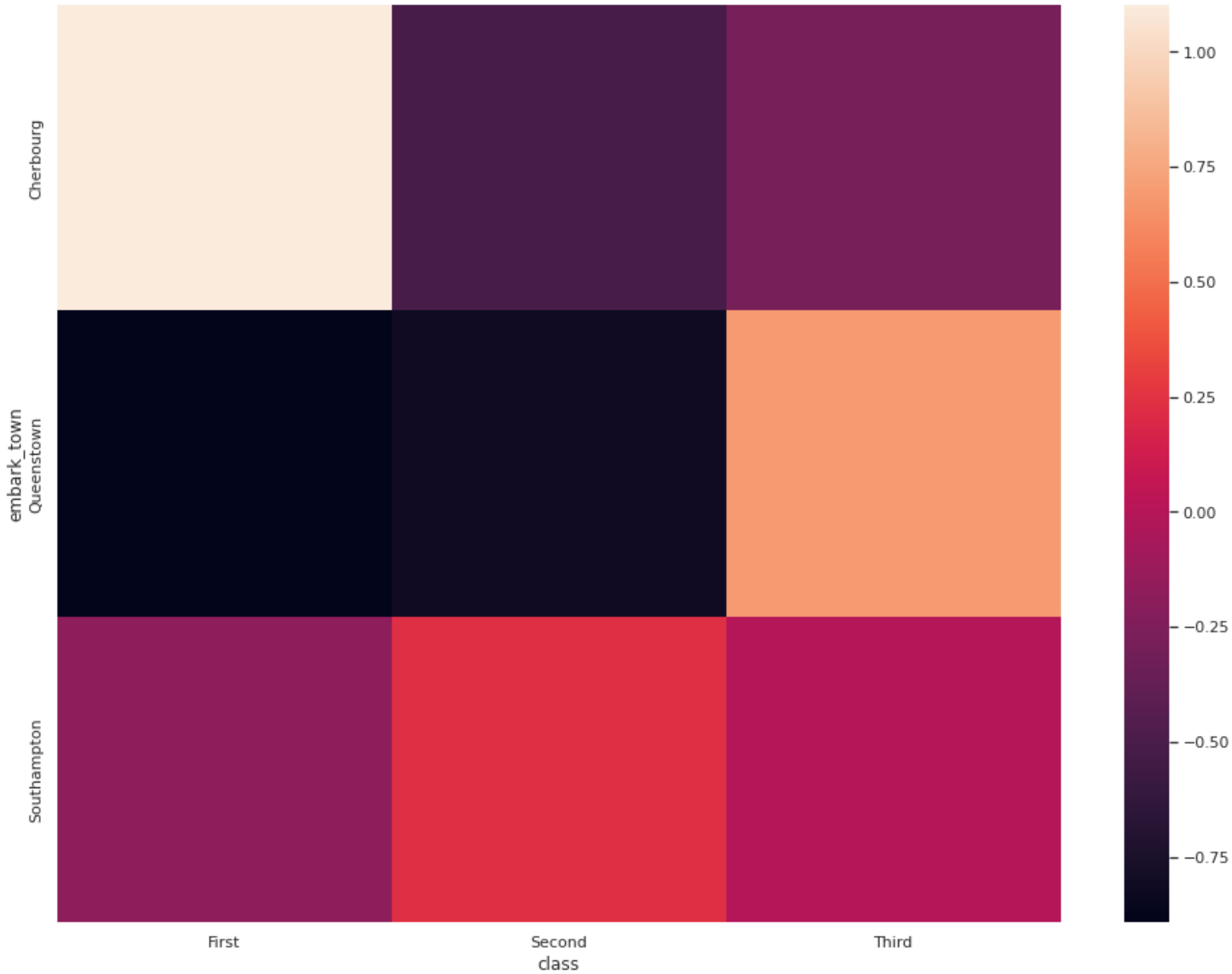
in order of appearance:

- the test statistic chi is very high, indicating a correlation
- the p value is low, so this is definitely not by chance
- there are 4 'degrees of freedom'
- the expected frequency table shows what it thinks the proporties should look like

What we could do now is create a heatmap with the contingency table but subtract the expected non-biased values and scale using the expected values (real - expected)/expected. This gives us the biggest changes in respect with 'random' values.

```
sns.heatmap(
    pd.DataFrame((contingency_table-exp)/exp, index=contingency_table.index,␣
↪columns=contingency_table.columns)
)
```

```
<AxesSubplot:xlabel='class', ylabel='embark_town'>
```



There seems to be much more people from first class that have embarked in Cherbourg, and the lower classes are more represented from Queenstown. The population from southampton only sees a positive deviation in second class.

To demonstrate that there can also be no correlation we now calculate the proportions of survival for each town and class combination.

```
survived_df = titanic_df.groupby(['embark_town', 'class']).survived.sum().unstack(
↪'class')/contingency_table
survived_df
```

```
class           First    Second     Third
embark_town
Cherbourg    0.694118  0.529412  0.378788
Queenstown   0.500000  0.666667  0.375000
Southampton  0.582677  0.463415  0.189802
```

If we would do a Cramer V test now, we assume there would be no significance, as it would not make sense that the embarked town has no influence on the chances (proportion of survived persons) of survival.

```
chi, p, dof, exp = stats.chi2_contingency(survived_df, correction=True)
p
```

```
0.9989353452702686
```

As you can see, the p value is 0.99, indicating that the differences in embarkment are purely coincidental!

28.2 Categorical vs continuous

The most interesting exploration (in my opinion) happens when we combine categorical and continuous data, as more graphing opportunities are present. When doing this comparison, we usually use the student t-test or Z-test, you can spend hours arguing the difference and which to use, yet I will stick for simplicity with the t-test for robuustness.

we can use the t-test to check if a continuous variable changes between 2 categories of a categorical variable.

let us seperate the men from the women and see if they had to pay a different fare amount

```
t, p = stats.ttest_ind(
    titanic_df.fare[titanic_df.who=='man'],
    titanic_df.fare[titanic_df.who=='woman']
)
t, p
```

```
(-5.817465335062089, 8.614583735152227e-09)
```

Our p-value again is very low, indicating there is a difference in the groups. The t statistic is -5.82, meaning that the second group (women) are paying more for fares.

We print out the means to verify

```
print('mean male fare')
print(titanic_df.fare[titanic_df.who=='man'].mean())
print('mean female fare')
print(titanic_df.fare[titanic_df.who=='woman'].mean())
```

```
mean male fare
24.864181750465548
mean female fare
46.570711070110704
```

By the looks of this, the fares are heavily gender biased. To put this into more detail, we pivot the means of each group including class into a table, as female might be more in the upper classes.

```
titanic_df.groupby(['who', 'class']).fare.mean().unstack('class')
```

```
class        First      Second       Third
who
child   139.382633  28.323905  23.220190
man      65.951086  19.054124  11.340213
woman   104.317995  20.868624  15.354351
```

This already makes more sense, it is mainly the first class difference that drives up the prices, yet the difference seems to be still present.

Can you perform a t-test on the gender fare gap in the third class, is it still significant?

A t-test is ideal if you would like to compare 2 groups, yet often we have multiple groups. For this we can use a (one_way) ANOVA or ANalysis Of VAriance.

We seperate on class and check if the fare is significantly different.

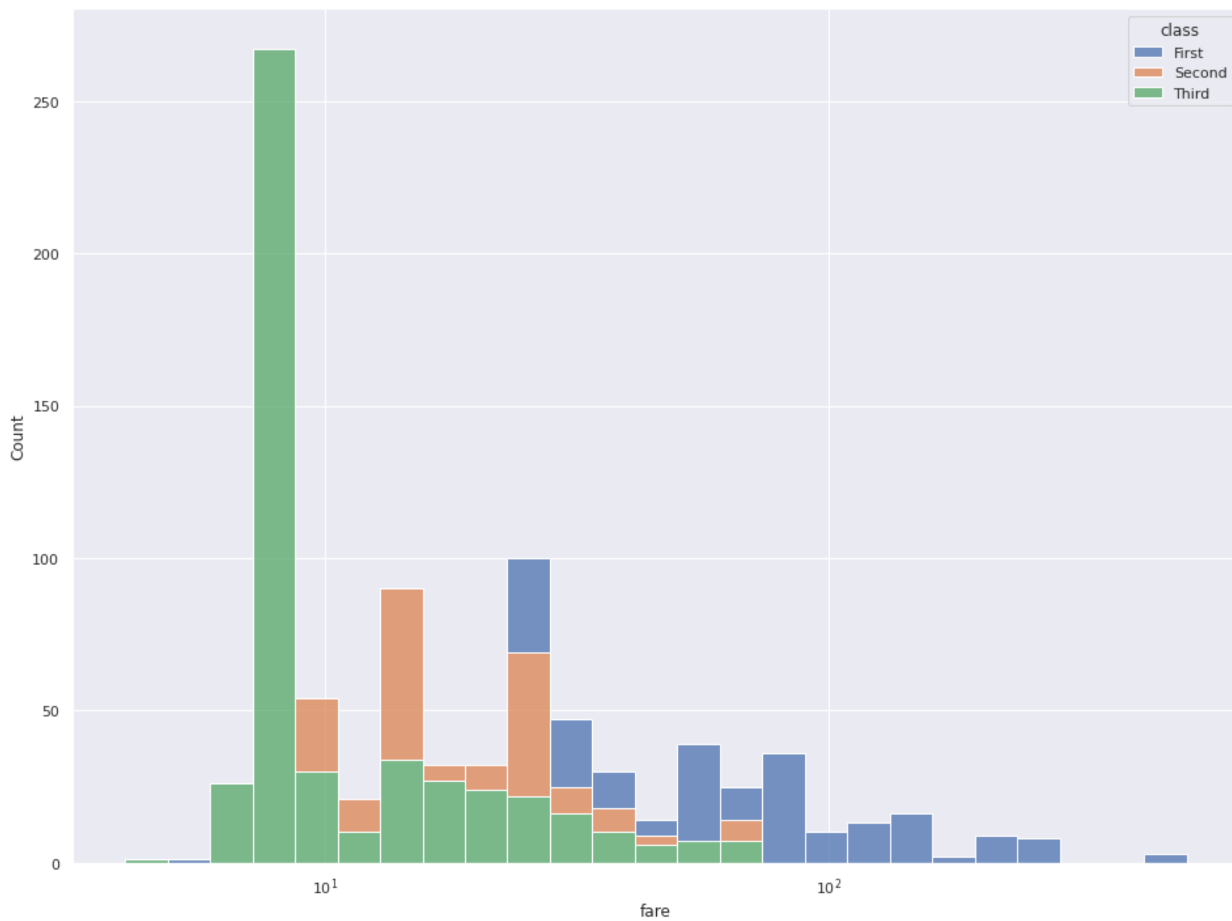
```
F, p = stats.f_oneway(
    titanic_df.fare[titanic_df.pclass==1],
    titanic_df.fare[titanic_df.pclass==2],
    titanic_df.fare[titanic_df.pclass==3]
)
F, p
```

```
(242.34415651744814, 1.0313763209141171e-84)
```

This was more or less a no-brainer, as it is advertised that higher classes come with a higher pricetag. We can use a nice histogram to show this division of class.

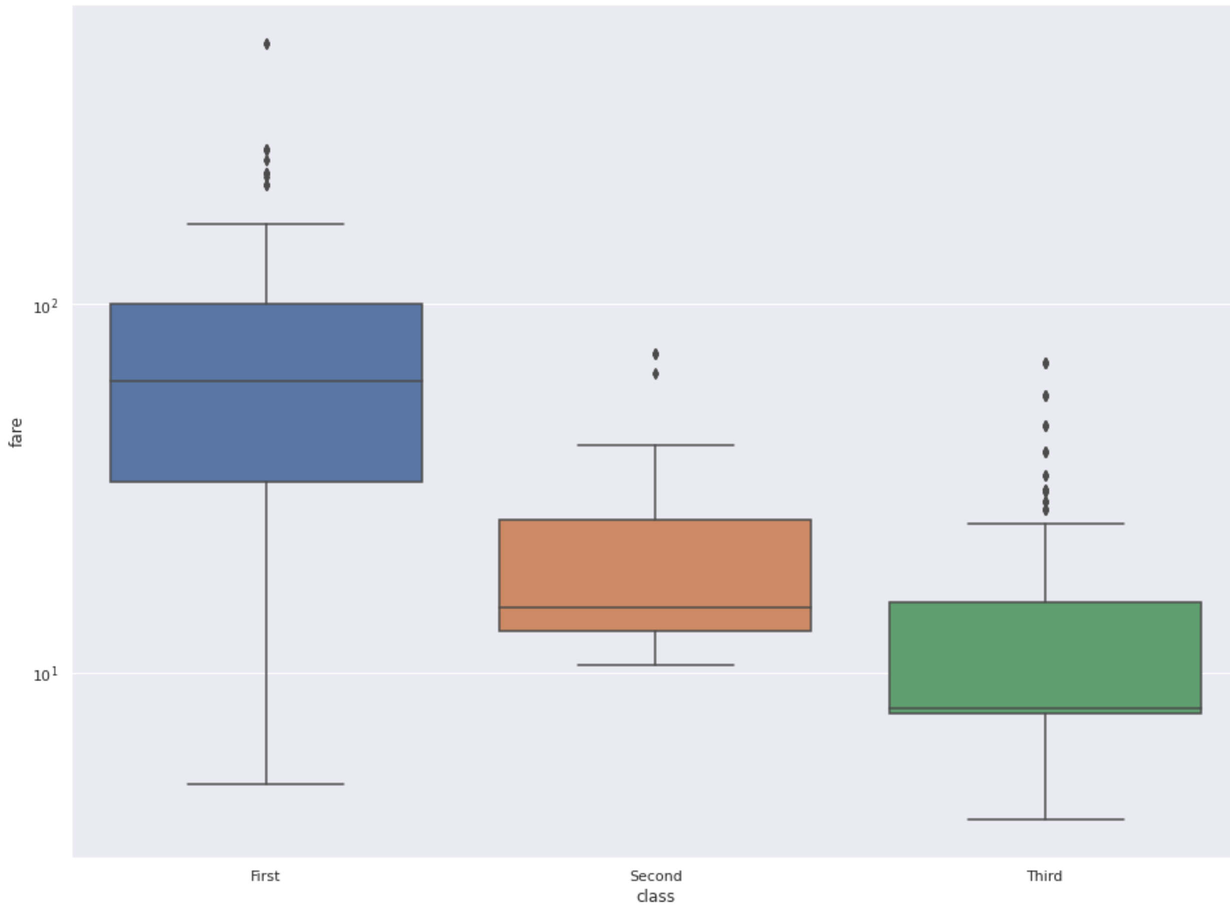
```
sns.histplot(data=titanic_df[titanic_df.fare!=0], x='fare', hue='class', log_
↪scale=True, multiple='stack', bins=25)
```

```
<AxesSubplot:xlabel='fare', ylabel='Count'>
```



A less cluttered plot would be to use a boxplot, containing less information about the distribution, yet still showing simple statistics.

```
ax = sns.boxplot(data=titanic_df[titanic_df.fare!=0], x='class', y='fare')
ax.set_yscale("log")
```



We could do something similar, but taking the age instead of the fare, giving us the following result.

```
F, p = stats.f_oneway(
    titanic_df.age[titanic_df.pclass==1].dropna(),
    titanic_df.age[titanic_df.pclass==2].dropna(),
    titanic_df.age[titanic_df.pclass==3].dropna()
)
F, p
```

```
(57.443484340676214, 7.487984171959904e-24)
```

The p value indicates there is surely a difference in age between classes, how about we look at the means for each class.

```
titanic_df.groupby('pclass').age.mean()
```

```
pclass
1    38.233441
2    29.877630
3    25.140620
Name: age, dtype: float64
```

What about any statistical significant differences in ages for the groups that survived and didn't, could you perform this analysis? Report your findings in a histogram.

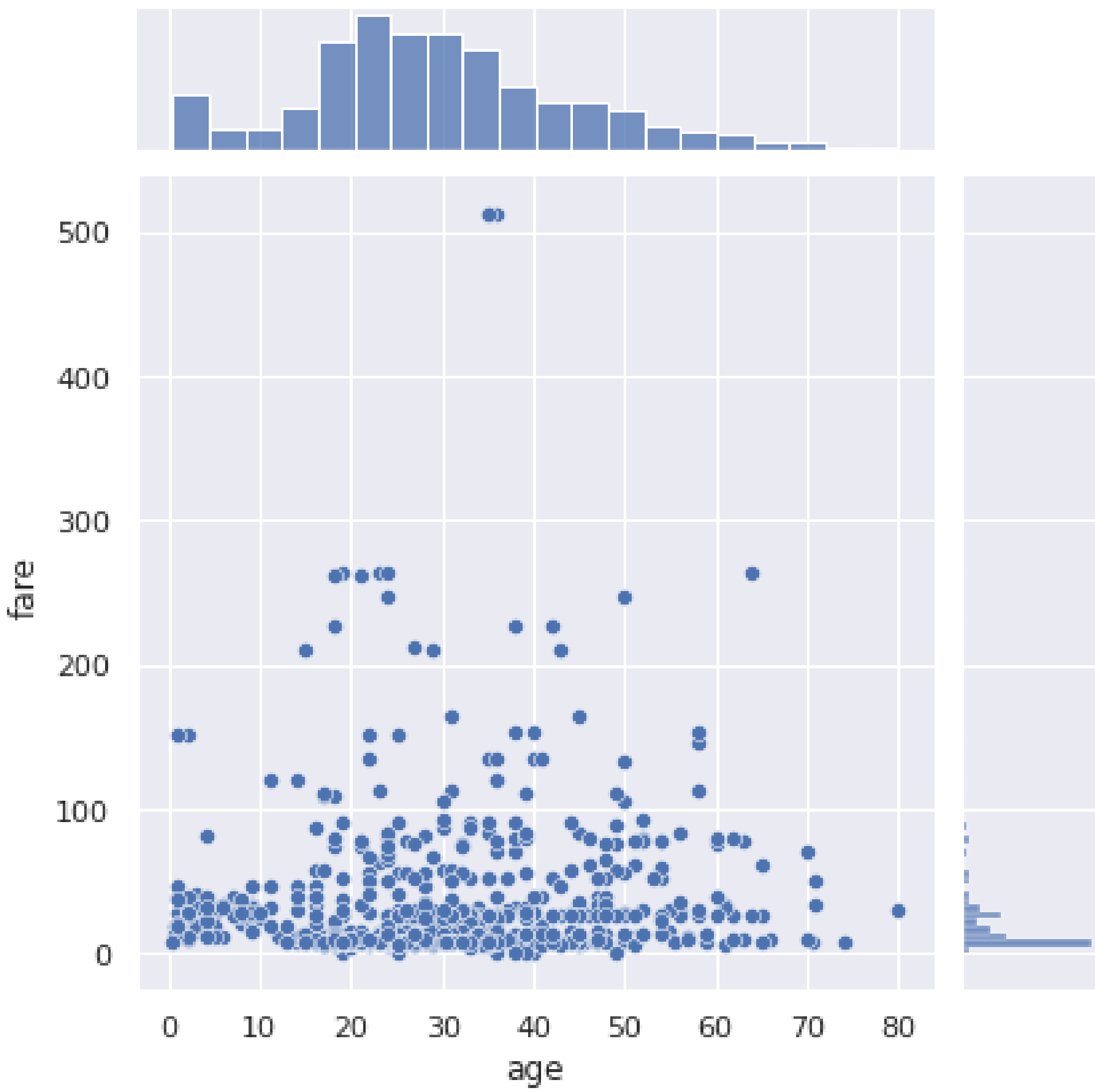
28.3 Continuous vs continuous

A thirds option to explore the interactions within your dataset is by comparing 2 continuous variables.

Seaborn has a nice functionality where can perform a jointplot that not only shows us the scatter plot but also the distributions, When we perform this plot we notice the inbalanced distribution of the fares.

```
sns.jointplot(data=titanic_df, x='age', y='fare')
```

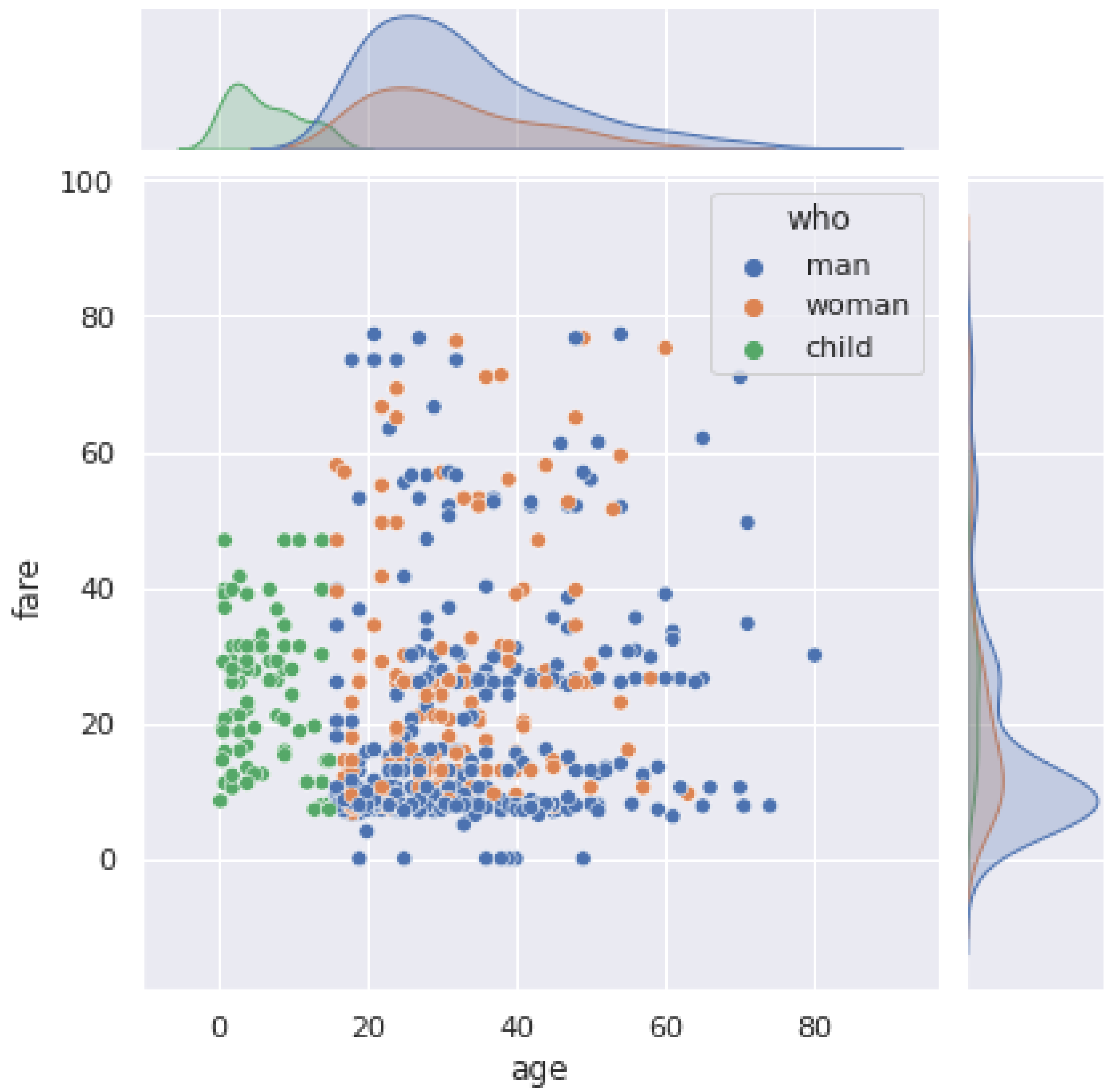
```
<seaborn.axisgrid.JointGrid at 0x7f9fa6f080d0>
```



What we could do is remove outliers, if I recall correclty we set a upper bound of 77.5, let's do that here and replot. I've also added the type of person as a color, you can here see that women and children pay more as we saw earlier.

```
sns.jointplot(data=titanic_df[titanic_df.fare<77.5], x='age', y='fare', hue='who')
```

```
<seaborn.axisgrid.JointGrid at 0x7f9fa6ac27f0>
```



To make this more mathematically sound, we are using the spearman rank correlation test, not the pearson as we are dealing with non normal data. You could check that with a shapiro wilk test but i'll leave that up to you!

```
corr, p = stats.spearmanr(a=titanic_df[['age','fare']].dropna())
corr, p
```

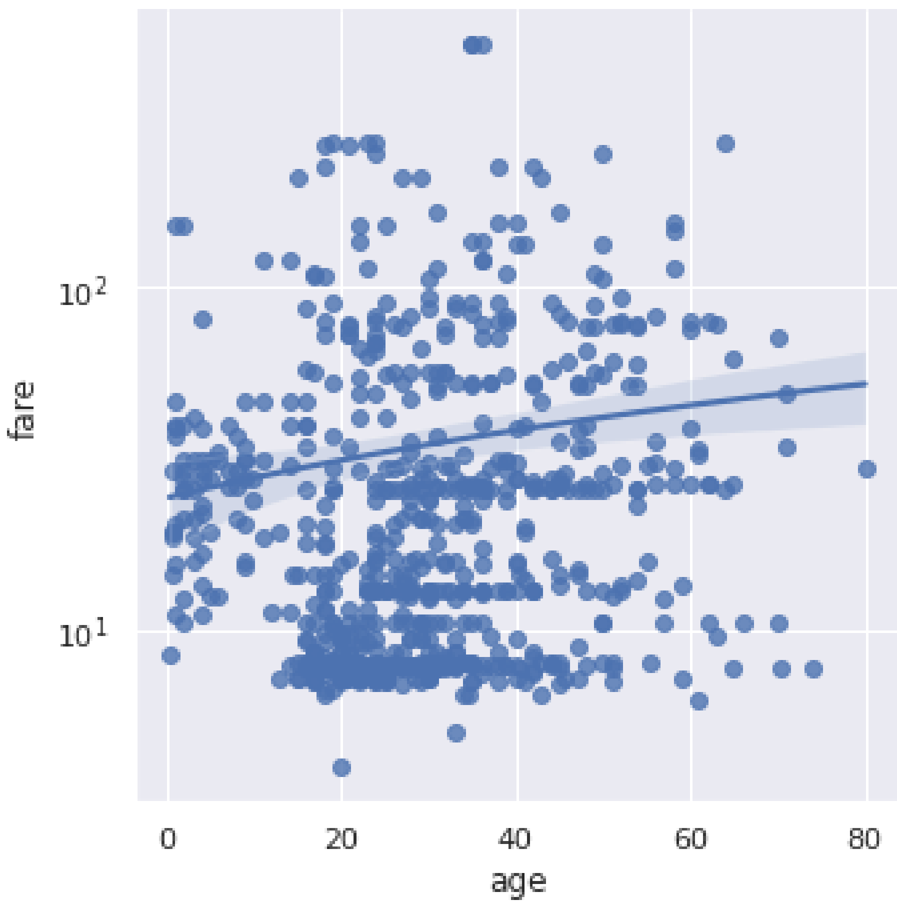
```
(0.13505121773428 78, 0.00029580903243060916)
```

with a p-value of only 0.000296 we can safely reject the null-hypothesis, meaning there is a correlation. The correlation coefficient here is only 0.135, meaning for any person each year of age would make their fare about 0.135 dollars more expensive on average, which in that time was a fair amount of money.

To make this more visual, I added a lmplot that performs a linear regression, you can see how the line goes up in fare as the age goes up. I had to use a logarithmic y-scale as the distribution is still not normal.

```
ax = sns.lmplot(data=titanic_df, x='age', y='fare')
ax.set(yscale='log')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9fa4916100>
```



Now this correlation of 0.135 dollar is relevant for ANY person, man, female, child, first class, second,…

Perhaps we could find several subgroups with a higher or lower correlation, I will perform the correlation with the outliers removed.

```
corr, p = stats.spearmanr(a=titanic_df[titanic_df.fare<77.5][['age','fare']].dropna())
corr, p
```

```
(0.09269934275477329, 0.019762193968013368)
```

When we remove outliers, we have a less strong correlations, indicating that the outliers - with high fares - are in general older persons.

Try to experiment with subsetting the data and find a group where age matters more for the correlation.

CHAPTER

TWENTYNINE

NEW DATA SOURCES

In this notebook we are going to look into adding new data to your dataset. We start out with a taxi dataset describing all pickup points from taxis in a specific date interval, notice that the dataset is divided up into months. Each month has their specific csv file saved in an AWS location.

```
import pandas as pd
import seaborn as sns
from urllib.request import urlopen
```

```
data_url_files = urlopen('https://raw.githubusercontent.com/toddwschneider/nyc-taxi-
↪data/master/setup_files/raw_data_urls.txt')
data_urls = data_url_files.read().decode('utf-8').split('\n')
data_urls[0:12]
```

```
['https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-01.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-02.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-03.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-04.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-05.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-06.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-07.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-08.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-09.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-10.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-11.csv',
 'https://s3.amazonaws.com/nyc-tlc/trip+data/fhv_tripdata_2015-12.csv']
```

Due to slow parsing of data we will here only parse the uber data from jan-mar 2015

```
datasets = [pd.read_csv(url) for url in data_urls[0:3]]
```

```
cab_df = pd.concat(datasets)
```

```
print('shape: ' + str(cab_df.shape))
cab_df.head()
```

```
shape: (9153861, 3)
```

```
  Dispatching_base_num          Pickup_date  locationID
0               B00013  2015-01-01 00:30:00         NaN
1               B00013  2015-01-01 01:22:00         NaN
2               B00013  2015-01-01 01:23:00         NaN
```

(continues on next page)

(continued from previous page)

```
3              B00013  2015-01-01 01:44:00         NaN
4              B00013  2015-01-01 02:00:00         NaN
```

We would like to find out how many uber rides were performed each day so we:

- parse the date string to a datetime format
- set the date as index
- resample to '1D' or one day (and chose count as aggregation)

```
cab_df['datetime'] = pd.to_datetime(cab_df['Pickup_date'], format="%Y/%m/%d %H:%M:%S")
```

```
cab_df = cab_df.set_index('datetime')
```

```
cab_df.head()
```

```
                    Dispatching_base_num          Pickup_date  locationID
datetime
2015-01-01 00:30:00               B00013  2015-01-01 00:30:00         NaN
2015-01-01 01:22:00               B00013  2015-01-01 01:22:00         NaN
2015-01-01 01:23:00               B00013  2015-01-01 01:23:00         NaN
2015-01-01 01:44:00               B00013  2015-01-01 01:44:00         NaN
2015-01-01 02:00:00               B00013  2015-01-01 02:00:00         NaN
```

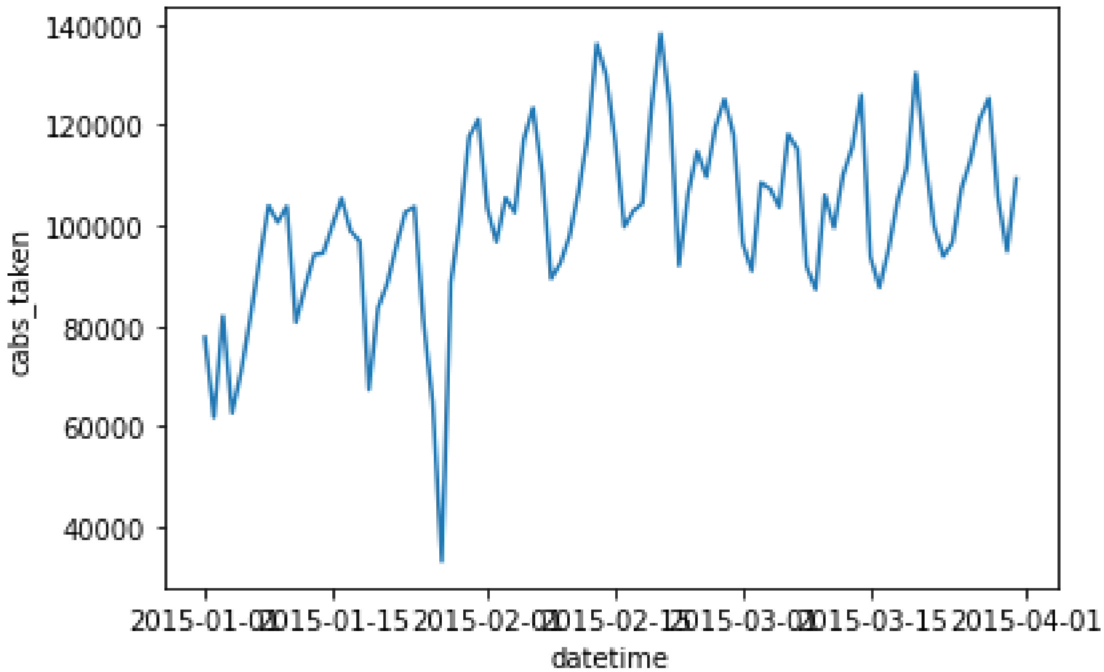
```
cabs_taken = cab_df['Dispatching_base_num'].resample('1D').count().rename('cabs_taken
↪')
cabs_taken.head()
```

```
datetime
2015-01-01    77789
2015-01-02    61832
2015-01-03    81955
2015-01-04    62691
2015-01-05    71063
Freq: D, Name: cabs_taken, dtype: int64
```

great! now we have an idea on how many ubers were taken each day, let us use a simple line plot to show the results.

```
sns.lineplot(data=cabs_taken)
```

```
<AxesSubplot:xlabel='datetime', ylabel='cabs_taken'>
```



This dataset is nice, but by itself pretty useless, why don't we look up some weather information to see if this influences our traffic.

```
url = 'https://raw.githubusercontent.com/toddwschneider/nyc-taxi-data/master/data/
↪central_park_weather.csv'
weather = pd.read_csv(url)
```

```
weather.head()
```

```
       STATION                          NAME        DATE   AWND  PRCP  SNOW  \
0  USW00094728  NY CITY CENTRAL PARK, NY US  2009-01-01  11.18   0.0   0.0
1  USW00094728  NY CITY CENTRAL PARK, NY US  2009-01-02   6.26   0.0   0.0
2  USW00094728  NY CITY CENTRAL PARK, NY US  2009-01-03  10.07   0.0   0.0
3  USW00094728  NY CITY CENTRAL PARK, NY US  2009-01-04   7.61   0.0   0.0
4  USW00094728  NY CITY CENTRAL PARK, NY US  2009-01-05   6.93   0.0   0.0

   SNWD  TMAX  TMIN
0   0.0    26    15
1   0.0    34    23
2   0.0    38    29
3   0.0    42    25
4   0.0    43    38
```

you can see a variaty of information, more info on the column names can be found hereagain we need to:

- parse the date
- set it to the index
- resampling is not needed as it is already in day-to-day intervals

```
weather['DATE'] =  pd.to_datetime(weather['DATE'], format="%Y/%m/%d")
weather = weather.set_index('DATE')
```

```
weather.head()
```

```
                STATION                          NAME   AWND  PRCP  SNOW  SNWD  \
DATE
2009-01-01  USW00094728  NY CITY CENTRAL PARK, NY US  11.18   0.0   0.0   0.0
2009-01-02  USW00094728  NY CITY CENTRAL PARK, NY US   6.26   0.0   0.0   0.0
2009-01-03  USW00094728  NY CITY CENTRAL PARK, NY US  10.07   0.0   0.0   0.0
2009-01-04  USW00094728  NY CITY CENTRAL PARK, NY US   7.61   0.0   0.0   0.0
2009-01-05  USW00094728  NY CITY CENTRAL PARK, NY US   6.93   0.0   0.0   0.0

            TMAX  TMIN
DATE
2009-01-01    26    15
2009-01-02    34    23
2009-01-03    38    29
2009-01-04    42    25
2009-01-05    43    38
```

Having 2 dataset, now we need to merge them. Since we already prepared the date as index, this should be easy.

```
merged_df = pd.merge(cabs_taken, weather, left_index=True, right_index=True)
```

```
merged_df.head()
```

```
            cabs_taken      STATION                          NAME   AWND  PRCP  \
2015-01-01       77789  USW00094728  NY CITY CENTRAL PARK, NY US   7.16  0.00
2015-01-02       61832  USW00094728  NY CITY CENTRAL PARK, NY US   7.16  0.00
2015-01-03       81955  USW00094728  NY CITY CENTRAL PARK, NY US   6.49  0.71
2015-01-04       62691  USW00094728  NY CITY CENTRAL PARK, NY US   6.49  0.30
2015-01-05       71063  USW00094728  NY CITY CENTRAL PARK, NY US  10.51  0.00

            SNOW  SNWD  TMAX  TMIN
2015-01-01   0.0   0.0    39    27
2015-01-02   0.0   0.0    42    35
2015-01-03   0.0   0.0    42    33
2015-01-04   0.0   0.0    56    41
2015-01-05   0.0   0.0    49    21
```

One would assume that when it is a rainy day, people would use more cabs. so let us seperate based on precipitation.

```
rained = merged_df[merged_df['PRCP']>0]
no_rain = merged_df[merged_df['PRCP']==0]
```

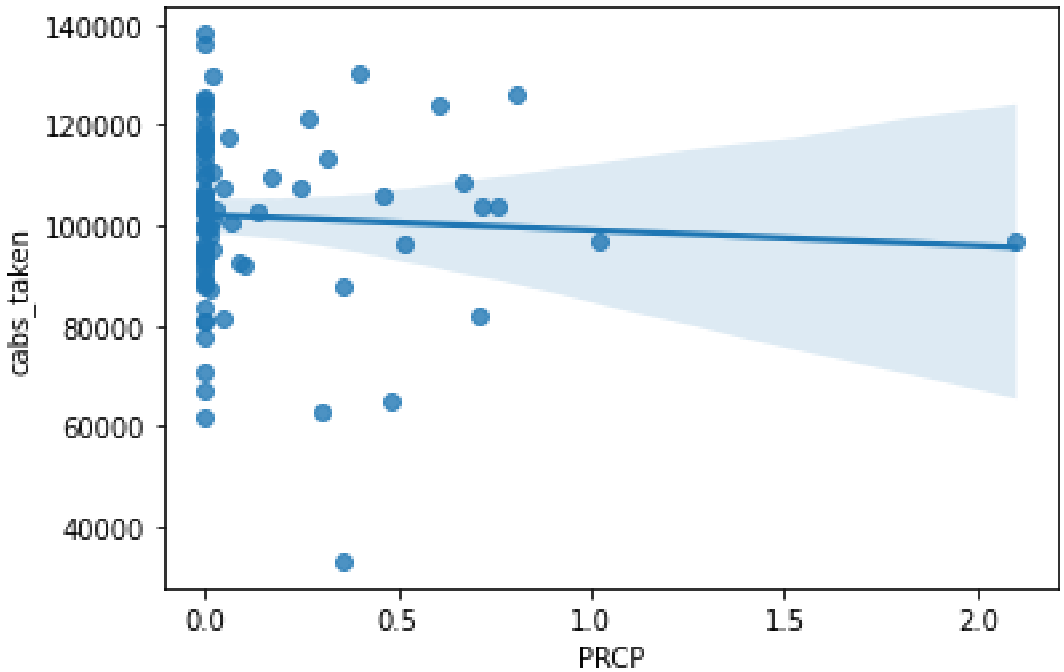
```
print('average uber rides on a rainy day')
print(rained['cabs_taken'].mean())
print('average uber rides on a dry day')
print(no_rain['cabs_taken'].mean())
```

```
average uber rides on a rainy day
99837.29411764706
average uber rides on a dry day
102846.30357142857
```

ouch! it looks like the average new yorker doesn't mind getting wet, or they take a cab any day…using a regression plot we can see it more clear

```
sns.regplot(data=merged_df, x='PRCP', y='cabs_taken')
```

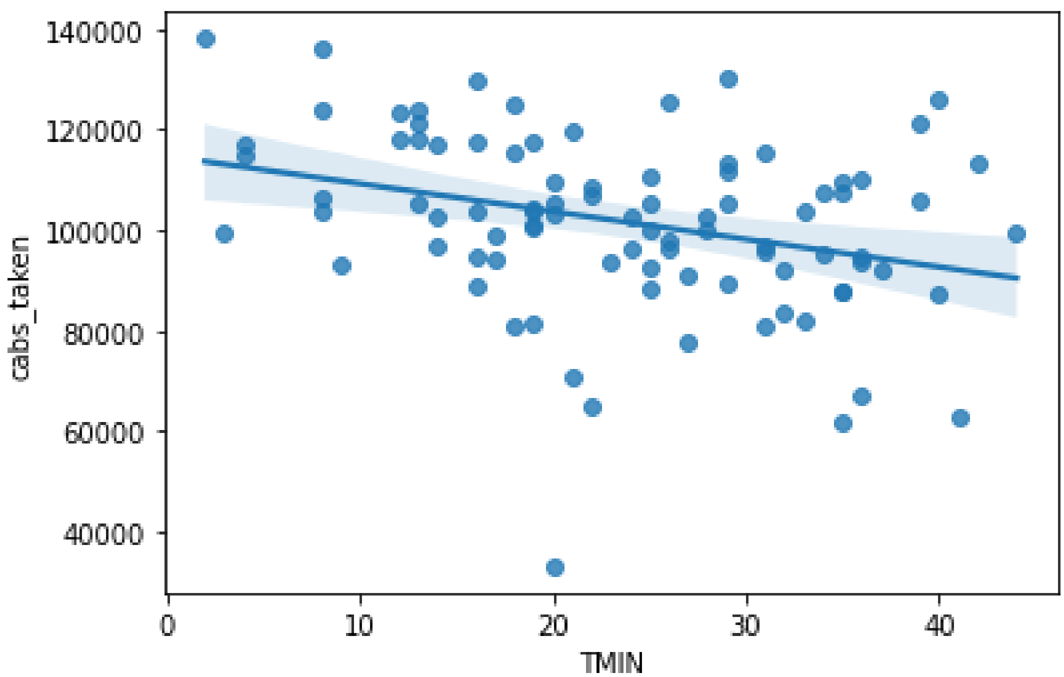
```
<AxesSubplot:xlabel='PRCP', ylabel='cabs_taken'>
```



Ok, here we see that it might just be because a lot of days are dry and the dataset is skewed. Not reliable info.What about temperatures, can we see a difference if the lowest temperature changes?

```
sns.regplot(data=merged_df, x='TMIN', y='cabs_taken')
```

```
<AxesSubplot:xlabel='TMIN', ylabel='cabs_taken'>
```



Appearantly when the temperature lowers, yorkers seem to be taking more cab rides. So global warming might be disastrous for capitalism after all?

CHAPTER

THIRTY

FEATURE ENHANCING

This rather simple notebook is a small illustration how feature enhancing might work in specific cases, we have a dataset containing cars and their fuel efficiency. What we will try to illustrate here is that sometimes combinations or formulas using the original data might display patterns not visible with the previous data.

```
import pandas as pd
import seaborn as sns
from scipy.stats import spearmanr
```

we load the mpg dataset and have a look at it.

```
mpg = sns.load_dataset('mpg')
```

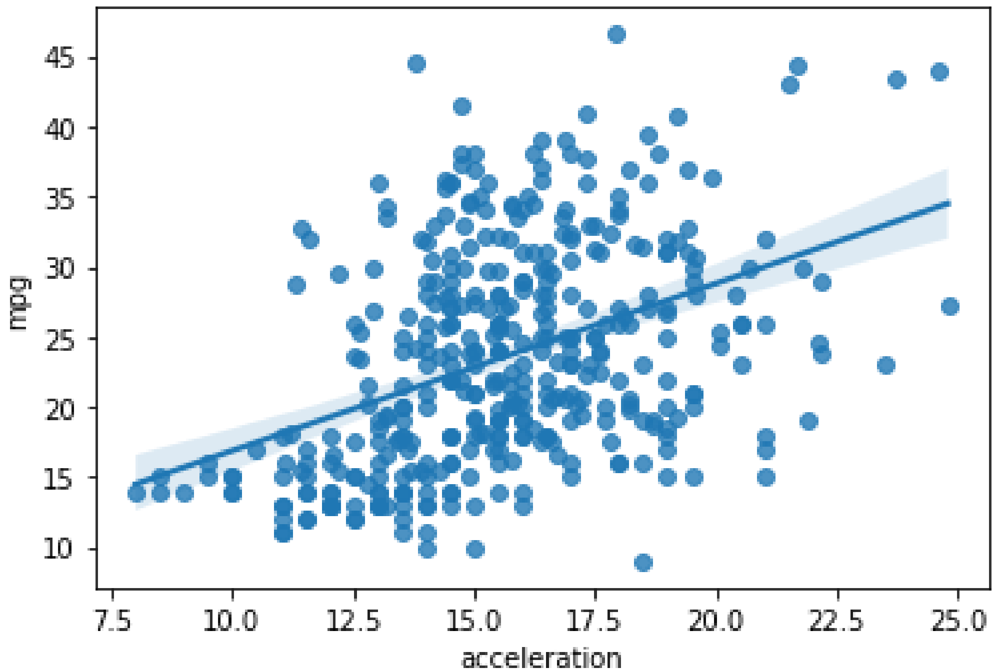
```
mpg.head()
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0    3504          12.0   
1  15.0          8         350.0       165.0    3693          11.5   
2  18.0          8         318.0       150.0    3436          11.0   
3  16.0          8         304.0       150.0    3433          12.0   
4  17.0          8         302.0       140.0    3449          10.5   

   model_year origin                       name  
0          70    usa  chevrolet chevelle malibu  
1          70    usa          buick skylark 320  
2          70    usa         plymouth satellite  
3          70    usa              amc rebel sst  
4          70    usa                ford torino  
```

We'll try to explore our dataset by printing out some regression plots between features of the car and the mileage per gallon.

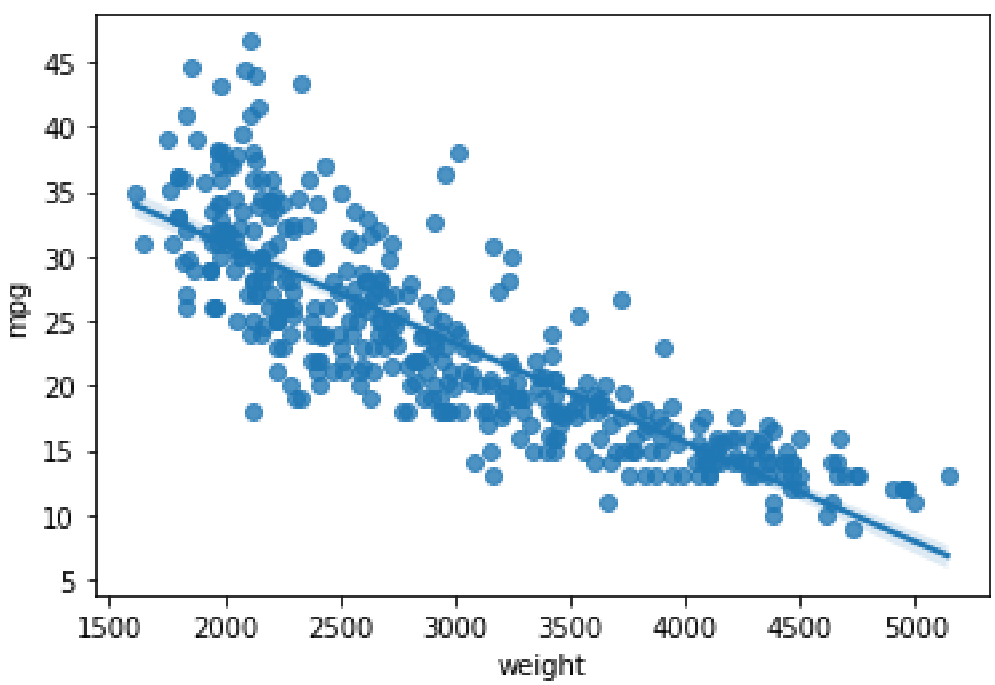
```
sns.regplot(x=mpg['acceleration'], y=mpg['mpg'])
corr, p = spearmanr(mpg['mpg'], mpg['acceleration'])
print('acceleration correlation: ' + str(100*round(corr,4)) + "%")
```

```
acceleration correlation: 43.87%
```




```
sns.regplot(x=mpg['weight'], y=mpg['mpg'])
corr, p = spearmanr(mpg['mpg'], mpg['weight'])
print('weight correlation: ' + str(100*round(corr,4)) + "%")
```

```
weight correlation: -87.49%
```



We can see that the acceleration has a positive influence on the miles per gallon, whilst the weight has a negative influence, what about the acceleration per weight?

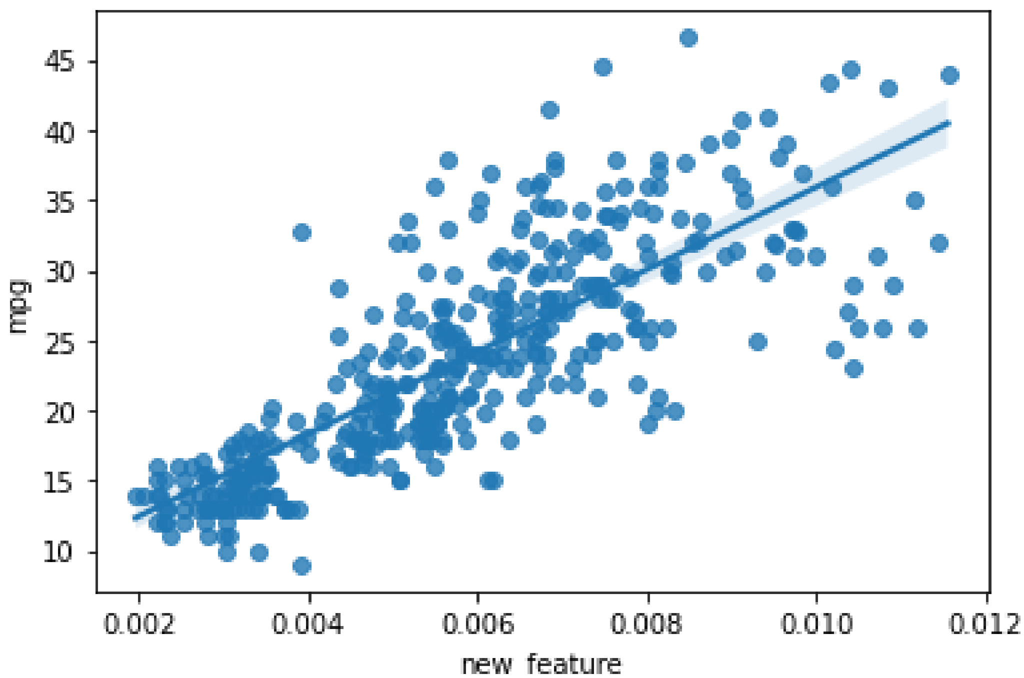
```
mpg['new_feature'] = mpg['acceleration']/mpg['weight']
```

```
mpg.head()
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0    3504          12.0
1  15.0          8         350.0       165.0    3693          11.5
2  18.0          8         318.0       150.0    3436          11.0
3  16.0          8         304.0       150.0    3433          12.0
4  17.0          8         302.0       140.0    3449          10.5

   model_year origin                       name  new_feature
0          70    usa  chevrolet chevelle malibu     0.003425
1          70    usa          buick skylark 320     0.003114
2          70    usa         plymouth satellite     0.003201
3          70    usa              amc rebel sst     0.003495
4          70    usa                ford torino     0.003044
```

```
sns.regplot(x=mpg['new_feature'], y=mpg['mpg'])
corr, p = spearmanr(mpg['mpg'], mpg['new_feature'])
print('new feature correlation: ' + str(100*round(corr,4)) + "%")
```

```
new feature correlation: 84.19%
```



It seems we are not able to create a new feature with even more correlation, not every story has to be a success. We can report this to our boss and explain the results.

CHAPTER THIRTYONE

CLUSTER ANALYSIS

Before starting this notebook I would like to state that what is explained here will be elaborated later in the course and might look complicated at this point. If you do not feel familiar with these concepts that is perfectly fine.

```
import pandas as pd
import seaborn as sns
```

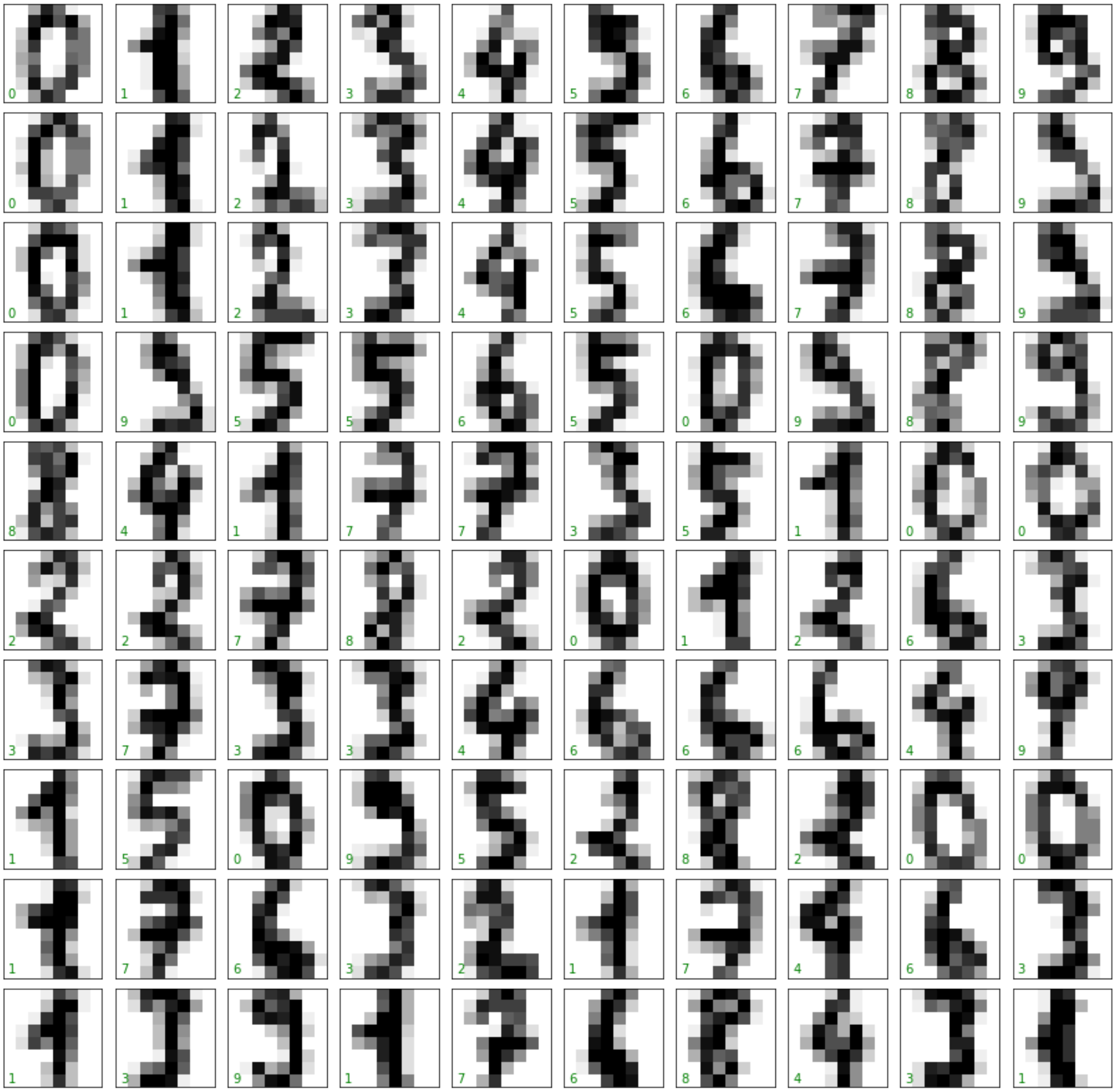
We will load a digits dataset from sklearn, the machine learning library, these are 8x8 pixel images showing handwritten digits with the correct answer.In the dataset there are 1797 images giving the dataset a dimension of (1797, 8*8)

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

```
(1797, 64)
```

Before we start, let's print out a few of them, the following cell will do that. Again, plotting is not yet seen, so the following cells might be overwhelming.

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(10, 10, figsize=(16, 16),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(digits.target[i]),
            transform=ax.transAxes, color='green')
```



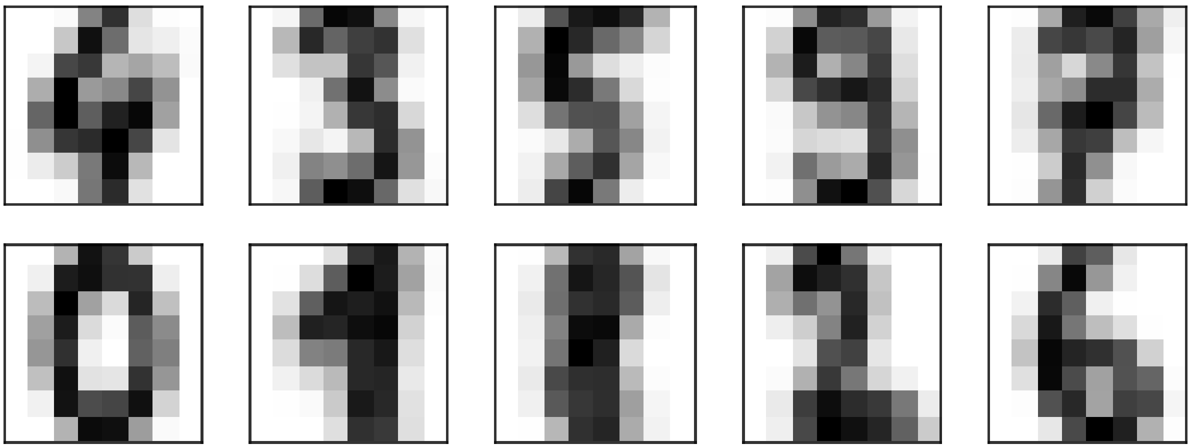
In cluster analysis we will try to figure out clusters within the dataset, keep in mind that these cluster are constructed without knowning the correct answer. Here we use the Isomap algorithm to create clusters, by using fit and transform methods we can create the clusters

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=10, random_state=0)
clusters = kmeans.fit_predict(digits.data)
kmeans.cluster_centers_.shape
```

```
(10, 64)
```

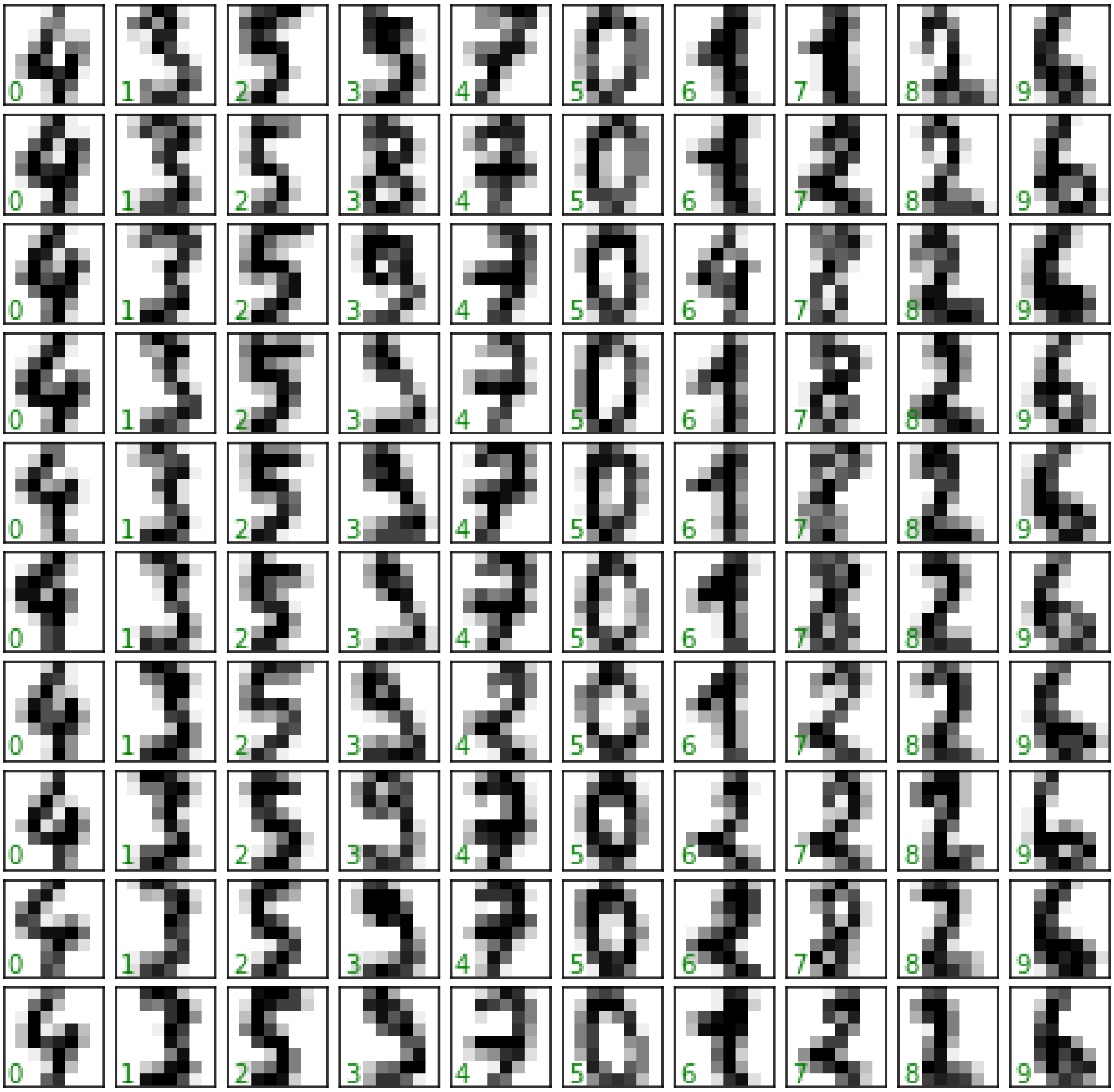
Now that the algorithm seperated the dataset into 10 clusters, we can ask it to print the center of each cluster.This gives us an idea how the average digit in that cluster looks like.

```
fig, ax = plt.subplots(2, 5, figsize=(8, 3))
centers = kmeans.cluster_centers_.reshape(10, 8, 8)
for axi, center in zip(ax.flat, centers):
    axi.set(xticks=[], yticks=[])
    axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)
```



Those look similar to the actual numbers, confirming that arabic numbers have good visual seperation inbetween.Aside from the centers we can also print a few examples from the clusters.

```
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[clusters==i%10][int(i/10)], cmap='binary', interpolation=
↪'nearest')
    ax.text(0.05, 0.05, str(i%10),
            transform=ax.transAxes, color='green')
```



You can see that the cluster number does not match the actual number, that's because our algorithm does not understand which numbers there are.It does however understand the differences between the numbers! This technique can also be used for other datasets where no outcome is given, but we would like to separate our dataset into clusters.

To make this more visible we will use another example of a dataset about the leafs of 3 types of iris flowers.

```
iris_df = sns.load_dataset('iris')
iris_df.head()
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

What we could do here is ask the algorithm to create 3 clusters of records, as the dataset contains 3 types of iris flowers.

We do not supply the algorithm with the information of the species, yet it has to figure out by itself how to seperate the records.

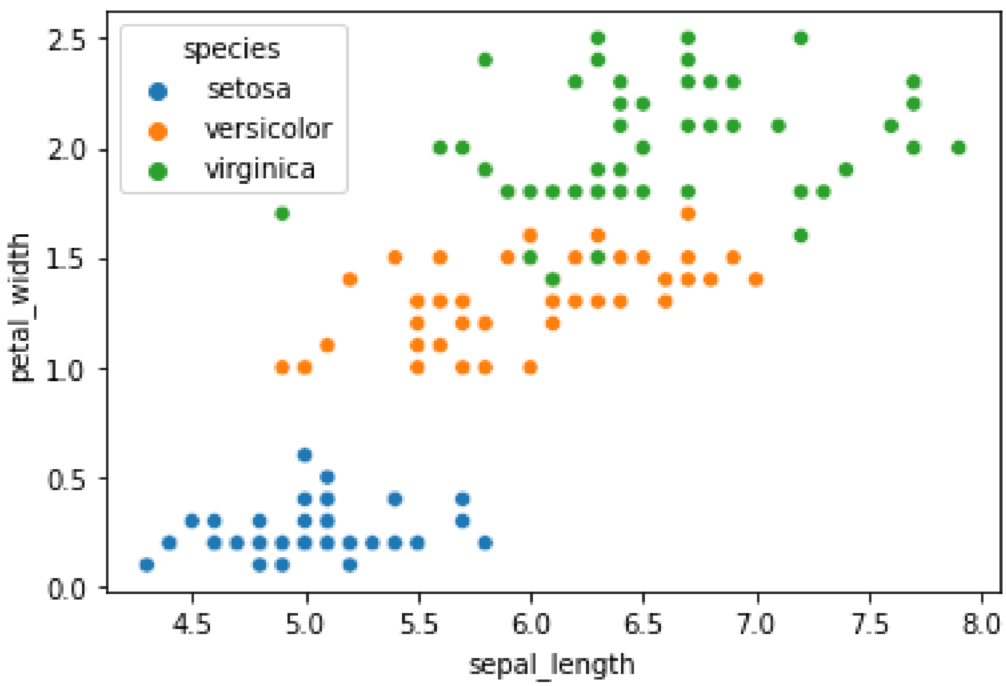
```
kmeans = KMeans(n_clusters=3, random_state=0)
iris_df['cluster'] = kmeans.fit_predict(iris_df.drop(columns='species'))
iris_df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	cluster
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3.0	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5.0	3.6	1.4	0.2	setosa	1

We can see our data now has an additional feature cluster which contains either 0, 1 or 2. If the clustering has been performed as expected, the clusters should coincide with the species. Using a plot we can find out.

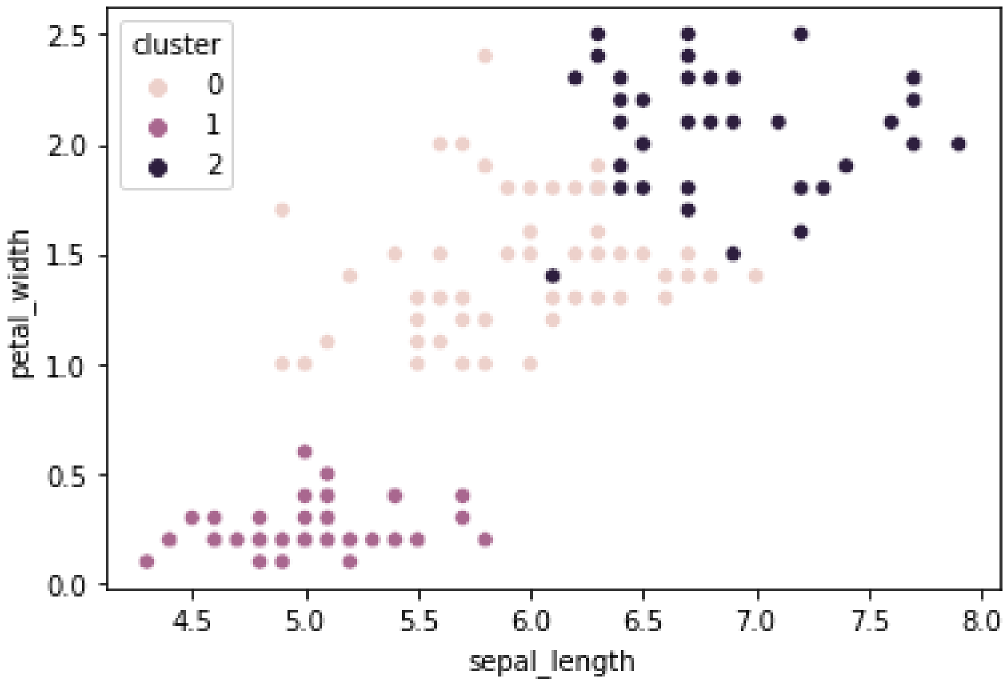
```
sns.scatterplot(data=iris_df, x='sepal_length', y='petal_width', hue='species')
```

```
<AxesSubplot:xlabel='sepal_length', ylabel='petal_width'>
```




```
sns.scatterplot(data=iris_df, x='sepal_length', y='petal_width', hue='cluster')
```

```
<AxesSubplot:xlabel='sepal_length', ylabel='petal_width'>
```



For some reason seaborn thinks it is useful to change color scheme, yet you can see that there is an uncanny similarity between the clusters and the species, the algorithm was succesful in finding the different species.

Without giving the information we were able to cluster the different species of iris flowers yet we have no idea which cluster belongs to which species. It is the reasers responsibility to take conclusion in what the different clusters mean!

CHAPTER

THIRTYTWO

VIF: VARIANCE INFLATION FACTOR

in this notebook we will investigate the variance inflation which can occur in a dataset. As an example here, we will use the 'Mile Per Gallon' dataset contianing a set of cars and their fuel efficiency. Some columns in the dataset might

```
import pandas as pd
import seaborn as sns
from statsmodels.stats.outliers_influence import variance_inflation_factor
mpg = sns.load_dataset('mpg')
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_13569/3881140987.py in <module>
      1 import pandas as pd
      2 import seaborn as sns
----> 3 from statsmodels.stats.outliers_influence import variance_inflation_factor
      4 mpg = sns.load_dataset('mpg')

ModuleNotFoundError: No module named 'statsmodels'
```

```
mpg.head()
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0    3504          12.0
1  15.0          8         350.0       165.0    3693          11.5
2  18.0          8         318.0       150.0    3436          11.0
3  16.0          8         304.0       150.0    3433          12.0
4  17.0          8         302.0       140.0    3449          10.5

   model_year origin                       name
0          70    usa  chevrolet chevelle malibu
1          70    usa          buick skylark 320
2          70    usa         plymouth satellite
3          70    usa              amc rebel sst
4          70    usa                ford torino
```

as you can see, we also imported a function 'variance_inflation_factor' which will help us calculate this, more information can be found on wikipedia.

to use the function, we refer to the documentation. The function is a bit stubborn and requires the following:

- only numerical values (so we to drop the categories)
- no nan values (dropping nans)
- as a numpy array instead of a pandas dataframe

```
cols_to_keep = ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
 ↪'model_year']
vif_compatible_df = mpg[cols_to_keep]
vif_compatible_df = vif_compatible_df.dropna(axis='index')
vif_compatible_df = vif_compatible_df.values
vif_compatible_df
```

```
array([[   8. ,  307. ,  130. , 3504. ,   12. ,   70. ],
       [   8. ,  350. ,  165. , 3693. ,   11.5,   70. ],
       [   8. ,  318. ,  150. , 3436. ,   11. ,   70. ],
       ...,
       [   4. ,  135. ,   84. , 2295. ,   11.6,   82. ],
       [   4. ,  120. ,   79. , 2625. ,   18.6,   82. ],
       [   4. ,  119. ,   82. , 2720. ,   19.4,   82. ]])
```

this looks a lot different! we don't know anymore what all of that means, but the computer does, now we run it through the function. Notice how we have to specify a specific column, the resulting inflation factor is that for the chosen column

```
# we pick column 0 which is 'cylinders' according to cols_to_keep
variance_inflation_factor(vif_compatible_df, 0)
```

```
115.97777160980726
```

```
for idx, col in enumerate(cols_to_keep):
  print(col + ": \t" + str(variance_inflation_factor(vif_compatible_df, idx)))
```

```
cylinders:         115.97777160980726
displacement:         86.48595590611876
horsepower:         60.25657462146676
weight:        137.4717563697324
acceleration:         69.40087667701684
model_year:         109.3200159587966
```

32.1 TODO

The variance inflation gives a numerical value to how little variation there is between one column and the others in a dataset, you will see how the numbers will gradually go down as you remove more and more columns.This way we have a quantifyable method of removing data from our dataset in case there is too much 'duplicate' information.There is no real cut-off value that specifies of a column should or should not be removed, so make sure you can argument your decision.

- experiment with removing columns in the cols_to_keep list
- What do you think would be the ideal dataset here? we would like to predict the fuel economy (mpg) of a car.

CHAPTER

THIRTYTHREE

PRINCIPLE COMPONENT ANALYSIS

In this notebook we will not try to remove data from our dataset, but transform the variation in our features (columns) into less features.We will do this using the concept of PCA (principle component analysis). The dataset we will be using here is about the dimensions of iris flowers, in total 150 flowers were measured of 3 species.

```
import pandas as pd
import seaborn as sns
from sklearn.decomposition import PCA
iris = sns.load_dataset('iris')
```

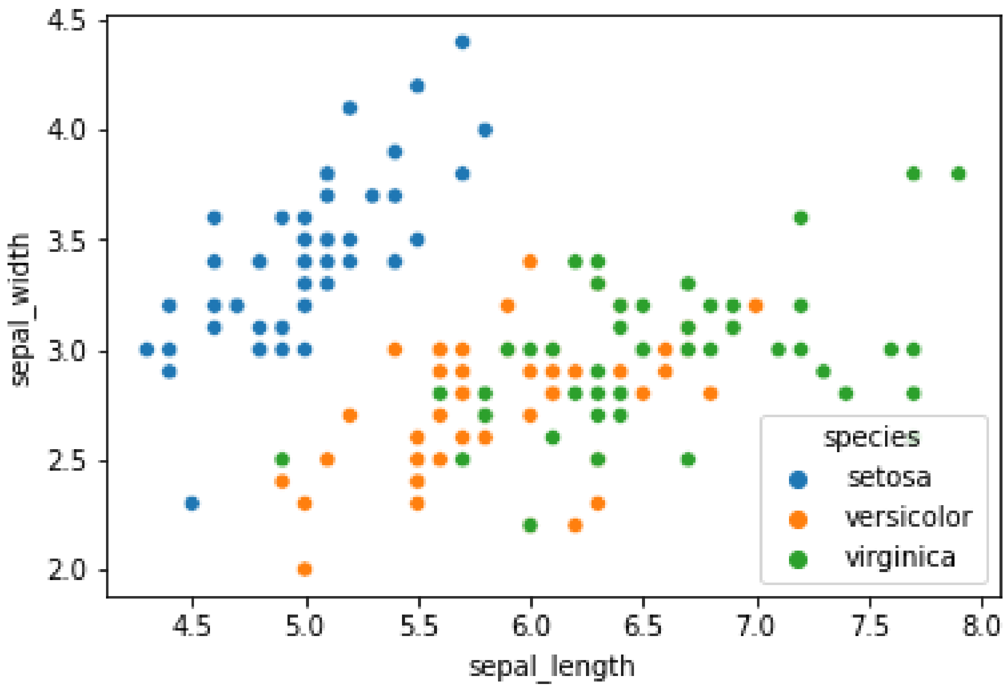
you can see that we imported a function PCA from sklearn, this will do the calculations for us, but we still need to specify some parameters.Before we do that, let us use the first 2 columns of the dataset to plot a scatter and see if we can distinguish the different species of flowers.

```
iris.head()
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

```
sns.scatterplot(x=iris['sepal_length'], y=iris['sepal_width'], hue=iris['species'])
```

```
<AxesSubplot:xlabel='sepal_length', ylabel='sepal_width'>
```



That already looks pretty good, but versicolor and virginica are still hard to differentiate. Let's see if we can compress the variation of all 4 columns into 2 axi.We do this by creating a PCA transformer and specifying we want only 2 output components

```
pca = PCA(n_components=2)
```

We also need to prepare our dataframe, we do this by only dropping our outcome (that which we do not need for the transform)

```
X = iris.drop(columns='species')
X.head()
```

```
   sepal_length  sepal_width  petal_length  petal_width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
```

```
iris_pca = pca.fit_transform(X)
pd.DataFrame(iris_pca, columns=['PC1', 'PC2'])
```

```
          PC1       PC2
0   -2.684126  0.319397
1   -2.714142 -0.177001
2   -2.888991 -0.144949
3   -2.745343 -0.318299
4   -2.728717  0.326755
..        ...       ...
145  1.944110  0.187532
146  1.527167 -0.375317
147  1.764346  0.078859
```

(continues on next page)

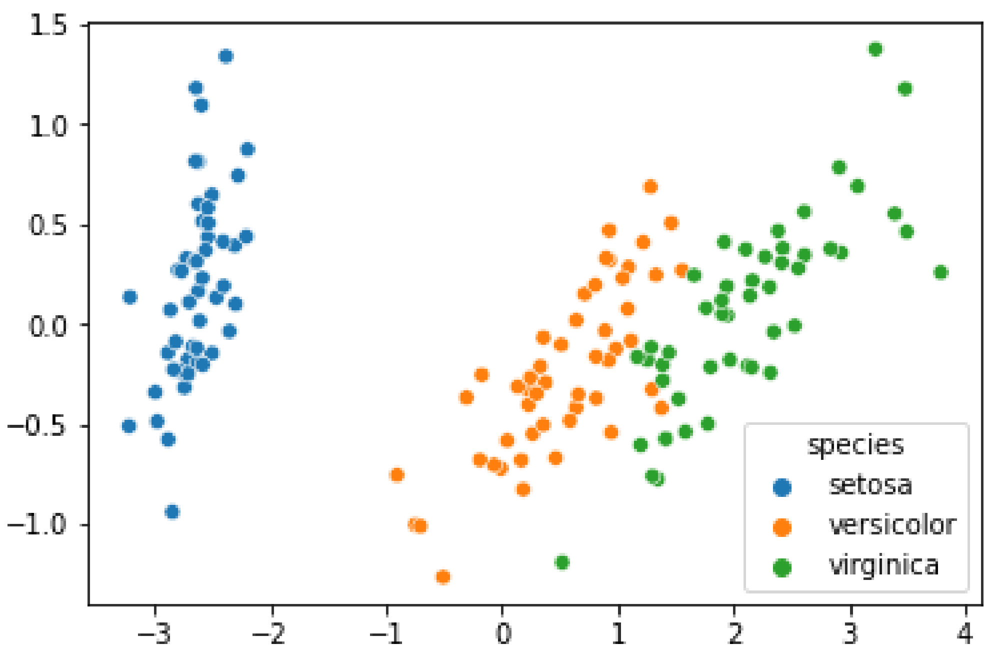
(continued from previous page)

```
148  1.900942  0.116628
149  1.390189 -0.282661

[150 rows x 2 columns]
```

Running it through the PCA transformer using the fit_transform function gives us a numpy 2 dimensional array (which is similar to a pandas dataframe) with 2 columns.When inserted into a scatter plot they show us (nearly) all variance of 4 columns compressed into a 2 dimensional plot.

```
sns.scatterplot(x=iris_pca[:,0], y=iris_pca[:,1], hue=iris['species'])
```

```
<AxesSubplot:>
```



33.1 TODO

it is clear that this function is very potent concerning data visualisation, do you think you can improve on the mpg dataset?

- experiment with the PCA transformer using the mpg dataset

```
mpg = sns.load_dataset('mpg')
mpg.head()
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0    3504          12.0
1  15.0          8         350.0       165.0    3693          11.5
2  18.0          8         318.0       150.0    3436          11.0
3  16.0          8         304.0       150.0    3433          12.0
4  17.0          8         302.0       140.0    3449          10.5

   model_year origin                       name
```

(continues on next page)

(continued from previous page)

```
0          70    usa  chevrolet chevelle malibu
1          70    usa          buick skylark 320
2          70    usa         plymouth satellite
3          70    usa              amc rebel sst
4          70    usa                ford torino
```

Part VI

6. Machine Learning

CHAPTER

THIRTYFOUR

MACHINE LEARNING

this is an introduction

Part VII

7. Case Studies

CHAPTER THIRTYFIVE

CASE STUDY: OLIST

In this case study we will create an overview on how a generic Data Analysis study on a dataset works.

The case study is divided into several parts:

- Goals
- Parsing
- Preparation (cleaning)
- Processing
- Exploration
- Visualization
- Conclusion

35.1 Goals

In this section we will state the goals we try to obtain by analyzing this dataset. Here are the questions that our customer had:

- Can we predict prices for products?
- Do customers behave predictable, can we recommend specific items to specific customers?
- Sellers with more/better reviews seem to do better, can you quantify this?
- Are there items with a specific time pattern?
- Are products related to geographical information?
- Is there anything else remarkable in our data?

We'll (try to) keep these question in mind when performing the case study.

35.2 Parsing

we start out by importing all necessary libraries

```
import os
import json
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats
import matplotlib.pyplot as plt
from IPython.display import set_matplotlib_formats
%matplotlib inline
set_matplotlib_formats('svg')
```

```
/tmp/ipykernel_6945/4057771804.py:10: DeprecationWarning: `set_matplotlib_formats` is␣
↪deprecated since IPython 7.23, directly use `matplotlib_inline.backend_inline.set_
↪matplotlib_formats()`
  set_matplotlib_formats('svg')
```

in order to download datasets from kaggle, we need an API key to access their API, we'll make that here

```
if not os.path.exists(os.path.expanduser('~/.kaggle')):
    os.mkdir(os.path.expanduser('~/.kaggle'))

with open(os.path.expanduser('~/.kaggle/kaggle.json'), 'w') as f:
    json.dump(
        {
            "username":"lorenzf",
            "key":"7a44a9e99b27e796177d793a3d85b8cf"
        }
        , f)
```

now we can import kaggle too and download the datasets

```
import kaggle
kaggle.api.dataset_download_files(dataset='olistbr/brazilian-ecommerce', path='./data
↪', unzip=True)
kaggle.api.dataset_download_files(dataset='olistbr/marketing-funnel-olist', path='./
↪data', unzip=True)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_6945/819357782.py in <module>
----> 1 import kaggle
      2 kaggle.api.dataset_download_files(dataset='olistbr/brazilian-ecommerce', path=
↪'./data', unzip=True)
      3 kaggle.api.dataset_download_files(dataset='olistbr/marketing-funnel-olist',␣
↪path='./data', unzip=True)

ModuleNotFoundError: No module named 'kaggle'
```

the csv files are now in the './data' folder, we can now read them using pandas, here is the list of all csv files in our folder

```
os.listdir('./data')
```

```
['olist_order_reviews_dataset.csv',
 'olist_order_items_dataset.csv',
 'product_category_name_translation.csv',
 'olist_products_dataset.csv',
 'olist_closed_deals_dataset.csv',
 'olist_order_payments_dataset.csv',
 'olist_marketing_qualified_leads_dataset.csv',
 'olist_sellers_dataset.csv',
 'olist_customers_dataset.csv',
 'olist_orders_dataset.csv',
 'olist_geolocation_dataset.csv']
```

we will now parse interesting dataframes.

```
customers = pd.read_csv('./data/olist_customers_dataset.csv')
print('shape: ' + str(customers.shape))
customers.head()
```

```
shape: (99441, 5)
```

```
                         customer_id  ... customer_state
0  06b8999e2fba1a1fbc88172c00ba8bc7  ...             SP
1  18955e83d337fd6b2def6b18a428ac77  ...             SP
2  4e7b3e00288586ebd08712fdd0374a03  ...             SP
3  b2b6027bc5c5109e529d4dc6358b12c3  ...             SP
4  4f2d8ab171c80ec8364f7c12e35b23ad  ...             SP

[5 rows x 5 columns]
```

```
sellers = pd.read_csv('./data/olist_sellers_dataset.csv')
print('shape: ' + str(sellers.shape))
sellers.head()
```

```
shape: (3095, 4)
```

```
                           seller_id  ...  seller_state
0  3442f8959a84dea7ee197c632cb2df15  ...            SP
1  d1b65fc7debc3361ea86b5f14c68d2e2  ...            SP
2  ce3ad9de960102d0677a81f5d0bb7b2d  ...            RJ
3  c0f3eea2e14555b6faeea3dd58c1b1c3  ...            SP
4  51a04a8a6bdcb23deccc82b0b80742cf  ...            SP

[5 rows x 4 columns]
```

```
products = pd.read_csv('./data/olist_products_dataset.csv')
print('shape: ' + str(products.shape))
products.head()
```

```
shape: (32951, 9)
```

```
                          product_id  ... product_width_cm
0  1e9e8ef04dbcff4541ed26657ea517e5  ...             14.0
1  3aa071139cb16b67ca9e5dea641aaa2f  ...             20.0
2  96bd76ec8810374ed1b65e291975717f  ...             15.0
```

(continues on next page)

(continued from previous page)

```
3  cef67bcfe19066a932b7673e239eb23d  ...              26.0
4  9dc1a7de274444849c219cff195d0b71  ...              13.0

[5 rows x 9 columns]
```

```
translation = pd.read_csv('./data/product_category_name_translation.csv')
print('shape: ' + str(translation.shape))
translation.head()
```

```
shape: (71, 2)
```

```
    product_category_name product_category_name_english
0           beleza_saude                 health_beauty
1  informatica_acessorios         computers_accessories
2             automotivo                          auto
3        cama_mesa_banho                bed_bath_table
4       moveis_decoracao               furniture_decor
```

```
orders = pd.read_csv('./data/olist_order_items_dataset.csv')
print('shape: ' + str(orders.shape))
orders.head()
```

```
shape: (112650, 7)
```

```
                           order_id  order_item_id  ...   price  freight_value
0  00010242fe8c5a6d1ba2dd792cb16214              1  ...   58.90          13.29
1  00018f77f2f0320c557190d7a144bdd3              1  ...  239.90          19.93
2  000229ec398224ef6ca0657da4fc703e              1  ...  199.00          17.87
3  00024acbcdf0a6daa1e931b038114c75              1  ...   12.99          12.79
4  00042b26cf59d7ce69dfabb4e55b4fd9              1  ...  199.90          18.14

[5 rows x 7 columns]
```

```
order_reviews = pd.read_csv('./data/olist_order_reviews_dataset.csv')
print('shape: ' + str(order_reviews.shape))
order_reviews.head()
```

```
shape: (99224, 7)
```

```
                          review_id  ... review_answer_timestamp
0  7bc2406110b926393aa56f80a40eba40  ...     2018-01-18 21:46:59
1  80e641a11e56f04c1ad469d5645fdfde  ...     2018-03-11 03:05:13
2  228ce5500dc1d8e020d8d1322874b6f0  ...     2018-02-18 14:36:24
3  e64fb393e7b32834bb789ff8bb30750e  ...     2017-04-21 22:02:06
4  f7c4243c7fe1938f181bec41a392bdeb  ...     2018-03-02 10:26:53

[5 rows x 7 columns]
```

35.3 Preparation

here we perform tasks to prepare the data in a more pleasing format.

35.3.1 Data Types

Before we do anything with our data, it is good to see if our data types are in order

```
customers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 5 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   customer_id               99441 non-null  object
 1   customer_unique_id        99441 non-null  object
 2   customer_zip_code_prefix  99441 non-null  int64 
 3   customer_city             99441 non-null  object
 4   customer_state            99441 non-null  object
dtypes: int64(1), object(4)
memory usage: 3.8+ MB
```

```
customers['customer_city'] = customers['customer_city'].astype('category')
customers['customer_state'] = customers['customer_state'].astype('category')
customers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 5 columns):
 #   Column                    Non-Null Count  Dtype   
---  ------                    --------------  -----   
 0   customer_id               99441 non-null  object  
 1   customer_unique_id        99441 non-null  object  
 2   customer_zip_code_prefix  99441 non-null  int64   
 3   customer_city             99441 non-null  category
 4   customer_state            99441 non-null  category
dtypes: category(2), int64(1), object(2)
memory usage: 2.7+ MB
```

```
sellers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3095 entries, 0 to 3094
Data columns (total 4 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   seller_id               3095 non-null   object
 1   seller_zip_code_prefix  3095 non-null   int64 
 2   seller_city             3095 non-null   object
 3   seller_state            3095 non-null   object
dtypes: int64(1), object(3)
memory usage: 96.8+ KB
```

```
sellers['seller_city'] = sellers['seller_city'].astype('category')
sellers['seller_state'] = sellers['seller_state'].astype('category')
sellers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3095 entries, 0 to 3094
Data columns (total 4 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   seller_id               3095 non-null   object
 1   seller_zip_code_prefix  3095 non-null   int64
 2   seller_city             3095 non-null   category
 3   seller_state            3095 non-null   category
dtypes: category(2), int64(1), object(1)
memory usage: 83.1+ KB
```

```
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32951 entries, 0 to 32950
Data columns (total 9 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   product_id                  32951 non-null  object
 1   product_category_name       32341 non-null  object
 2   product_name_lenght         32341 non-null  float64
 3   product_description_lenght  32341 non-null  float64
 4   product_photos_qty          32341 non-null  float64
 5   product_weight_g            32949 non-null  float64
 6   product_length_cm           32949 non-null  float64
 7   product_height_cm           32949 non-null  float64
 8   product_width_cm            32949 non-null  float64
dtypes: float64(7), object(2)
memory usage: 2.3+ MB
```

```
products['product_category_name'] = products['product_category_name'].astype('category
↪')
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32951 entries, 0 to 32950
Data columns (total 9 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   product_id                  32951 non-null  object
 1   product_category_name       32341 non-null  category
 2   product_name_lenght         32341 non-null  float64
 3   product_description_lenght  32341 non-null  float64
 4   product_photos_qty          32341 non-null  float64
 5   product_weight_g            32949 non-null  float64
 6   product_length_cm           32949 non-null  float64
 7   product_height_cm           32949 non-null  float64
 8   product_width_cm            32949 non-null  float64
dtypes: category(1), float64(7), object(1)
memory usage: 2.0+ MB
```

```
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112650 entries, 0 to 112649
Data columns (total 7 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   order_id             112650 non-null  object
 1   order_item_id        112650 non-null  int64
 2   product_id           112650 non-null  object
 3   seller_id            112650 non-null  object
 4   shipping_limit_date  112650 non-null  object
 5   price                112650 non-null  float64
 6   freight_value        112650 non-null  float64
dtypes: float64(2), int64(1), object(4)
memory usage: 6.0+ MB
```

```
orders['shipping_limit_date']= pd.to_datetime(orders['shipping_limit_date'])
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112650 entries, 0 to 112649
Data columns (total 7 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   order_id             112650 non-null  object
 1   order_item_id        112650 non-null  int64
 2   product_id           112650 non-null  object
 3   seller_id            112650 non-null  object
 4   shipping_limit_date  112650 non-null  datetime64[ns]
 5   price                112650 non-null  float64
 6   freight_value        112650 non-null  float64
dtypes: datetime64[ns](1), float64(2), int64(1), object(3)
memory usage: 6.0+ MB
```

35.3.2 Missing values

for each dataframe we apply a few checks in order to see the quality of data

```
print('customer missing values: ')
print(customers.isna().any())
```

```
customer missing values:
customer_id                 False
customer_unique_id          False
customer_zip_code_prefix    False
customer_city               False
customer_state              False
dtype: bool
```

```
print('sellers missing values: ')
print(sellers.isna().any())
```

```
sellers missing values:
seller_id                 False
seller_zip_code_prefix    False
seller_city               False
seller_state              False
dtype: bool
```

```
print('products missing values: ')
print(products.isna().any())
```

```
products missing values:
product_id                    False
product_category_name          True
product_name_lenght            True
product_description_lenght     True
product_photos_qty             True
product_weight_g               True
product_length_cm              True
product_height_cm              True
product_width_cm               True
dtype: bool
```

we can see that there are missing values for products, let's see how many!

```
products.isna().sum()
```

```
product_id                      0
product_category_name         610
product_name_lenght           610
product_description_lenght    610
product_photos_qty            610
product_weight_g                2
product_length_cm               2
product_height_cm               2
product_width_cm                2
dtype: int64
```

as there are not 'that many' products with missing information, I opted to drop them out. But maybe later i'll come back to that decision if these products seem crucial.

```
products = products.dropna()
```

```
print('orders missing values: ')
print(orders.isna().any())
```

```
orders missing values:
order_id               False
order_item_id          False
product_id             False
seller_id              False
shipping_limit_date    False
price                  False
freight_value          False
dtype: bool
```

35.3.3 Duplicates

```
print('customer duplicates: ')
print(customers.duplicated().any())
```

```
customer duplicates:
False
```

```
print('seller duplicates: ')
print(sellers.duplicated().any())
```

```
seller duplicates:
False
```

```
print('products duplicates: ')
print(products.duplicated().any())
```

```
products duplicates:
False
```

```
print('orders duplicates: ')
print(orders.duplicated().any())
```

```
orders duplicates:
False
```

No duplicates, that's a good sign, it means that each customer, seller and product is unique!

35.3.4 Indexing

It is more convenient to work with an index, usually we can use ids as index

```
customers = customers.set_index('customer_id')
customers.head()
```

```
                                                customer_unique_id  ...  customer_
 ↪state
customer_id                                                         ...
06b8999e2fba1a1fbc88172c00ba8bc7  861eff4711a542e4b93843c6dd7febb0  ...         ␣
 ↪SP
18955e83d337fd6b2def6b18a428ac77  290c77bc529b7ac935b93aa66c333dc3  ...         ␣
 ↪SP
4e7b3e00288586ebd08712fdd0374a03  060e732b5b29e8181a18229c7b0b2b5e  ...         ␣
 ↪SP
b2b6027bc5c5109e529d4dc6358b12c3  259dac757896d24d7702b9acbbff3f3c  ...         ␣
 ↪SP
4f2d8ab171c80ec8364f7c12e35b23ad  345ecd01c38d18a9036ed96c73b8d066  ...         ␣
 ↪SP

[5 rows x 4 columns]
```

```
sellers = sellers.set_index('seller_id')
sellers.head()
```

```
                                  seller_zip_code_prefix  ... seller_state
seller_id                                                 ...
3442f8959a84dea7ee197c632cb2df15                   13023  ...           SP
d1b65fc7debc3361ea86b5f14c68d2e2                   13844  ...           SP
ce3ad9de960102d0677a81f5d0bb7b2d                   20031  ...           RJ
c0f3eea2e14555b6faeea3dd58c1b1c3                    4195  ...           SP
51a04a8a6bdcb23deccc82b0b80742cf                   12914  ...           SP

[5 rows x 3 columns]
```

```
products = products.set_index('product_id')
products.head()
```

```
                                  product_category_name  ...  product_width_cm
product_id                                               ...
1e9e8ef04dbcff4541ed26657ea517e5             perfumaria  ...              14.0
3aa071139cb16b67ca9e5dea641aaa2f                  artes  ...              20.0
96bd76ec8810374ed1b65e291975717f          esporte_lazer  ...              15.0
cef67bcfe19066a932b7673e239eb23d                  bebes  ...              26.0
9dc1a7de274444849c219cff195d0b71  utilidades_domesticas  ...              13.0

[5 rows x 8 columns]
```

```
orders = orders.set_index('order_id')
orders.head()
```

```
                                  order_item_id  ... freight_value
order_id                                         ...
00010242fe8c5a6d1ba2dd792cb16214              1  ...         13.29
00018f77f2f0320c557190d7a144bdd3              1  ...         19.93
000229ec398224ef6ca0657da4fc703e              1  ...         17.87
00024acbcdf0a6daa1e931b038114c75              1  ...         12.79
00042b26cf59d7ce69dfabb4e55b4fd9              1  ...         18.14

[5 rows x 6 columns]
```

35.3.5 Translation

for the products we have a specific dataset that contains the translations, we can apply that to the products dataframe

```
 translation_dict = translation.set_index('product_category_name')['product_category_
↪name_english'].to_dict()
products['product_category_name'] = products['product_category_name'].cat.rename_
↪categories(translation_dict)
products.head()
```

```
                                 product_category_name  ...  product_width_cm
product_id                                              ...
1e9e8ef04dbcff4541ed26657ea517e5             perfumery  ...              14.0
3aa071139cb16b67ca9e5dea641aaa2f                   art  ...              20.0
96bd76ec8810374ed1b65e291975717f        sports_leisure  ...              15.0
cef67bcfe19066a932b7673e239eb23d                  baby  ...              26.0
9dc1a7de274444849c219cff195d0b71            housewares  ...              13.0
```

(continues on next page)

(continued from previous page)

```
[5 rows x 8 columns]
```

35.4 Processing

35.4.1 Product pricing

if we want to find out if there is a correlation between pricing and products, we need to match each product with a price, let's see what happens when we merge orders and products

```
orders.head()
```

```
                                  order_item_id  ... freight_value
order_id                                         ...
00010242fe8c5a6d1ba2dd792cb16214              1  ...         13.29
00018f77f2f0320c557190d7a144bdd3              1  ...         19.93
000229ec398224ef6ca0657da4fc703e              1  ...         17.87
00024acbcdf0a6daa1e931b038114c75              1  ...         12.79
00042b26cf59d7ce69dfabb4e55b4fd9              1  ...         18.14

[5 rows x 6 columns]
```

it seems that we only have prices of complete orders, which makes things more complicated. Below you can see that some orders contain multiple unique products, therefore we cannot easily deduce the price of a single item…

```
orders.groupby(level=0).apply(lambda x: x.product_id.nunique()).value_counts()
```

```
1    95430
2     2846
3      298
4       70
6       10
5        8
7        3
8        1
dtype: int64
```

well, let us see if we can find all orders with one item, these prices should agree with the price of the product

```
multi_item_orders = orders[orders['order_item_id']!=1].index.unique().values
single_item_orders = orders.drop(index=multi_item_orders)
```

```
products_w_price = products.merge(single_item_orders[['product_id', 'price', 'freight_
↪value']], how='left', left_index=True, right_on='product_id').drop(columns='product_
↪id')
```

```
products_w_price
```

```
                                  product_category_name  ...  freight_value
e17e4f88e31525f7deef66779844ddce              perfumery  ...           7.39
5236307716393b7114b53ee991f36956                    art  ...          17.99
```

(continues on next page)

(continued from previous page)

```
01f66e58769f84129811d43eefd187fb          sports_leisure  ...           7.82
143d00a4f2dde4e0364ee1821577adb3                    baby  ...           9.54
86cafb8794cb99a9b1b77fc8e48fbbbb              housewares  ...           8.29
...                                                  ...  ...            ...
6e4008bddce63615856554f94e5233db          bed_bath_table  ...          11.91
7c8a032bb75e0e4d524b14ba147d4ba5          bed_bath_table  ...          17.14
fc957026f2482ab3bddf91ebc9d0dfc5          bed_bath_table  ...          12.39
NaN                                computers_accessories  ...            NaN
f3a47ba087f05d39a74ed1b653f0be1b          bed_bath_table  ...          27.05

[90991 rows x 10 columns]
```

35.4.2 grouped per category

It would be interesting to have the averages of each feature grouped per category.

```
avg_category_product = products_w_price.groupby('product_category_name').mean()
avg_category_product
```

```
                            product_name_lenght  ...  freight_value
product_category_name                            ...
agro_industry_and_commerce            46.189349  ...      28.733963
food                                  48.781022  ...      14.680448
food_drink                            45.186916  ...      17.074249
art                                   47.687179  ...      19.120052
arts_and_craftmanship                 46.791667  ...      16.152500
...                                         ...  ...            ...
signaling_and_security                49.641221  ...      22.465238
tablets_printing_image                55.444444  ...      15.205278
telephony                             52.207986  ...      15.705825
fixed_telephony                       47.950000  ...      16.911832
housewares                            48.442928  ...      21.907430

[73 rows x 9 columns]
```

35.4.3 seller reviews

Another thing that says a lot about sales is the seller rating, we combine orders with order reviews for this

```
seller_review_df = pd.merge(
    orders,
    order_reviews,
    left_index=True,
    right_on='order_id'
).merge(
    sellers,
    left_on='seller_id',
    right_index=True
)
seller_review_df.head()
```

```
       order_item_id  ... seller_state
51963              1  ...           SP
53184              1  ...           SP
81465              1  ...           SP
25922              1  ...           SP
82616              1  ...           SP

[5 rows x 16 columns]
```

We can do a lot of things with this, an option is to get the average review per seller

```
seller_review_df.groupby('seller_id')['review_score'].mean().sort_values()
```

```
seller_id
6d04126aba80df143fd038e711b8fd96    1.0
b6c6854d4d92a5f6f46be8869da3fa1a    1.0
34aefe746cd81b7f3b23253ea28bef39    1.0
b7ba853e9551f4558440881fd3e5c815    1.0
17adeba047385fb0c67d8e90b4296d21    1.0
                                   ...
d7827b2af99326a03b0ed9c7a24db0d3    5.0
4aba6a02a788d3ec81c03137144d9a80    5.0
94ca168e8bcb407ab85c5da308863027    5.0
95cca791657aabeff15a07eb152d7841    5.0
186cdd1b2df32caa72cfb410bba768d3    5.0
Name: review_score, Length: 3090, dtype: float64
```

or the average review per seller state

```
seller_review_df.groupby('seller_state')['review_score'].mean().sort_values()
```

```
seller_state
AC    1.000000
AM    2.333333
RO    3.857143
PB    3.864865
SE    3.900000
MA    4.002506
SP    4.005078
ES    4.005450
DF    4.033333
PR    4.072292
PI    4.083333
BA    4.090202
SC    4.093865
RJ    4.101670
MG    4.105868
PE    4.132584
CE    4.138298
MT    4.165517
RS    4.214351
GO    4.254826
RN    4.267857
MS    4.469388
PA    4.500000
Name: review_score, dtype: float64
```

35.5 Exploration

35.5.1 Product pricing

for the product pricing we created a dataframe that contained the single item price for most products, lets review the dataframe

```
products_w_price.info()
products_w_price.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 90991 entries, e17e4f88e31525f7deef66779844ddce to
↪f3a47ba087f05d39a74ed1b653f0be1b
Data columns (total 10 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   product_category_name       90991 non-null  category
 1   product_name_lenght         90991 non-null  float64
 2   product_description_lenght  90991 non-null  float64
 3   product_photos_qty          90991 non-null  float64
 4   product_weight_g            90991 non-null  float64
 5   product_length_cm           90991 non-null  float64
 6   product_height_cm           90991 non-null  float64
 7   product_width_cm            90991 non-null  float64
 8   price                       87575 non-null  float64
 9   freight_value               87575 non-null  float64
dtypes: category(1), float64(9)
memory usage: 7.0+ MB
```

```
                                 product_category_name  ...  freight_value
e17e4f88e31525f7deef66779844ddce             perfumery  ...           7.39
5236307716393b7114b53ee991f36956                   art  ...          17.99
01f66e58769f84129811d43eefd187fb        sports_leisure  ...           7.82
143d00a4f2dde4e0364ee1821577adb3                  baby  ...           9.54
86cafb8794cb99a9b1b77fc8e48fbbbb            housewares  ...           8.29

[5 rows x 10 columns]
```

```
products_w_price.describe()
```

```
       product_name_lenght  ...  freight_value
count         90991.000000  ...   87575.000000
mean             48.847600  ...      20.405906
std              10.009026  ...      16.052020
min               5.000000  ...       0.000000
25%              42.000000  ...      13.440000
50%              52.000000  ...      16.500000
75%              57.000000  ...      21.400000
max              76.000000  ...     409.680000

[8 rows x 9 columns]
```

normal distribution

When we would want to predict the price of an item, it means the the other information of that item should correlate with said price. we can do that for all numerical values with a correlation plot. Before we do that let us use shapiro wilk to test normality

```
for name, col in products_w_price.loc[:,(products_w_price.dtypes == float).values].
↪iteritems():
    print(name)
    print(scipy.stats.shapiro(col.dropna()))
```

```
product_name_lenght
(0.9154905080795288, 0.0)
product_description_lenght
(0.8121932148933411, 0.0)
product_photos_qty
(0.743693470954895, 0.0)
product_weight_g
(0.5443710088729858, 0.0)
product_length_cm
(0.8115382194519043, 0.0)
product_height_cm
(0.8004813194274902, 0.0)
product_width_cm
(0.8457856774330139, 0.0)
price
(0.4680249094963074, 0.0)
freight_value
(0.5769327282905579, 0.0)
```

```
/usr/local/lib/python3.7/dist-packages/scipy/stats/morestats.py:1676: UserWarning: p-
↪value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

Numerical correlation

hmm it seems that we are dealing with very non normal data, which is usually the case if human behaviour is involved. We should be careful when using linear or parametric methods, so instead of calculating the pearson correlation coefficients, I opt to go for spearman rank correlations

```
pricing_corr = products_w_price.loc[:,(products_w_price.dtypes == float).values].
↪corr(method='spearman')
pricing_corr
```

```
                            product_name_lenght  ...  freight_value
product_name_lenght                    1.000000  ...       0.033853
product_description_lenght             0.082110  ...       0.123991
product_photos_qty                     0.165681  ...       0.007767
product_weight_g                       0.077482  ...       0.460155
product_length_cm                      0.055458  ...       0.293482
product_height_cm                     -0.042872  ...       0.295279
product_width_cm                       0.062193  ...       0.283687
price                                  0.026564  ...       0.445154
freight_value                          0.033853  ...       1.000000
```

(continues on next page)

(continued from previous page)

```
[9 rows x 9 columns]
```

Variance inflation

it looks like there seem to be some interesting correlations, the price is (slightly) correlated with things as product description, weight, length, height, width and freight value, indicating that bigger items are priced higher. We have to take into account that freight value is on itself correlating with the latter and therefore might be inflating our results, lets use VIF to check this

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19:␣
↪FutureWarning: pandas.util.testing is deprecated. Use the functions in the public␣
↪API at pandas.testing instead.
  import pandas.util.testing as tm
```

```
cols_to_keep = ['product_name_lenght', 'product_description_lenght', 'product_photos_
↪qty', 'product_weight_g', 'product_length_cm', 'product_height_cm', 'product_width_
↪cm', 'freight_value']
vif_compatible_price = products_w_price[cols_to_keep]
vif_compatible_price = vif_compatible_price.dropna(axis='index')
vif_compatible_price = vif_compatible_price.values
vif_price = {}
for idx, col in enumerate(cols_to_keep):
    vif_price[col] = variance_inflation_factor(vif_compatible_price, idx)
    print(col + ": \t" + str(variance_inflation_factor(vif_compatible_price, idx)))
```

```
product_name_lenght:         9.31669476790696
product_description_lenght:         2.56022139812164
product_photos_qty:         2.7618344464628604
product_weight_g:         3.041859125336769
product_length_cm:         6.820907759918533
product_height_cm:         3.686708040653942
product_width_cm:         7.7388710828170835
freight_value:         4.1851224676888155
```

As mentioned earlier, the values here are hard to interpret, however the values seem to be lower than my experience expected. If infinite values arise we know that we need to do things different. Let's assume the collinearity between these columns is ok and they don't interfere with eachother enough to make a difference in the outcome.

Categorical correlation

Something interesting we haven't looked into yet is the product category, we could try an ANOVA, but knowing at least one category is different is just a beginning.

```
str(list(products_w_price.dtypes[(products_w_price.dtypes == float)].index))
```

```
"['product_name_lenght', 'product_description_lenght', 'product_photos_qty', 'product_
↪weight_g', 'product_length_cm', 'product_height_cm', 'product_width_cm', 'price',
↪'freight_value']"
```

```
products_w_price_p_category = [products_w_price.loc[products_w_price['product_
↪category_name']==category,(products_w_price.dtypes == float).values].dropna() for␣
↪category in products_w_price['product_category_name'].unique()]
result = scipy.stats.f_oneway(*products_w_price_p_category)

anova_price = {}
for name, test, p in zip(list(products_w_price.dtypes[(products_w_price.dtypes ==␣
↪float)].index), result[0], result[1]):
    anova_price[name] = [test, p]

anova_price = pd.DataFrame.from_dict(anova_price, columns=['test', 'p'], orient='index
↪')
anova_price
```

```
                                  test    p
product_name_lenght          75.721841  0.0
product_description_lenght  188.961232  0.0
product_photos_qty          119.219872  0.0
product_weight_g            296.729258  0.0
product_length_cm           387.834682  0.0
product_height_cm           394.784176  0.0
product_width_cm            450.042245  0.0
price                       155.185233  0.0
freight_value                98.666293  0.0
```

it seems that every continuous column has at least one category that differs from the rest, aside from order item id, which is always 1.

Grouping by category

Now comes the tricky part, we would like to know if specific categories perform better on the correlations, but this is impossible to do by hand! However python gives us the opportunity to automate this. To do this properly we have to set a rule:

- correlations should be better than the original one without separation of categories

Look closely how we do almost exactly the same, however we aggregate (groupby) based on the category name

```
pricing_corr
```

```
                            product_name_lenght  ...  freight_value
product_name_lenght                    1.000000  ...       0.033853
product_description_lenght             0.082110  ...       0.123991
product_photos_qty                     0.165681  ...       0.007767
product_weight_g                       0.077482  ...       0.460155
product_length_cm                      0.055458  ...       0.293482
product_height_cm                     -0.042872  ...       0.295279
product_width_cm                       0.062193  ...       0.283687
price                                  0.026564  ...       0.445154
freight_value                          0.033853  ...       1.000000

[9 rows x 9 columns]
```

```
pricing_rel_corr = products_w_price.groupby('product_category_name').apply(
    lambda x: x.loc[:,(x.dtypes == float).values].corr(method='spearman') - pricing_
↪corr
```

(continues on next page)

(continued from previous page)

```
    )
pricing_rel_corr
```

```
                                                    product_name_lenght  ... ␣
↪freight_value
product_category_name                                                    ...
agro_industry_and_commerce product_name_lenght                0.000000  ...       ␣
↪0.036661
                           product_description_lenght         0.542952  ...       ␣
↪0.057213
                           product_photos_qty                 0.187918  ...       -
↪0.009524
                           product_weight_g                   0.177061  ...       ␣
↪0.171163
                           product_length_cm                 -0.053490  ...       ␣
↪0.308595
...                                                                ...  ...       ␣
↪     ...
housewares                 product_length_cm                  0.018919  ...       ␣
↪0.121902
                           product_height_cm                  0.022937  ...       -
↪0.016564
                           product_width_cm                   0.031462  ...       ␣
↪0.085940
                           price                              0.069826  ...       ␣
↪0.098247
                           freight_value                      0.068276  ...       ␣
↪0.000000

[657 rows x 9 columns]
```

for those who are already proficient with python can read that I opted to take the absolute correlation (meaning negatives become positives), this way both negative and positive correlations mean the same thing. Then I subtracted with the overall absolute correlation and divided that whole with the overall correlation giving me a relative change. When this relative change is positive, that category has an increased correlation

```
pricing_corr_stacked = pricing_rel_corr.stack()
pricing_corr_stacked.sort_values(ascending=False)
```

```
product_category_name
security_and_services                product_width_cm           product_description_
↪lenght    1.063551
                                     product_description_lenght  product_width_cm       ␣
↪       1.063551
                                     product_height_cm          product_name_lenght     ␣
↪       1.042872
                                     product_name_lenght        product_height_cm       ␣
↪       1.042872
pc_gamer                             product_name_lenght        product_height_cm       ␣
↪       1.042872
                                                                                        ␣
↪         ...
                                     product_weight_g           product_width_cm        ␣
↪      -1.536737
furniture_mattress_and_upholstery    product_length_cm          product_weight_g        ␣
↪      -1.542839
```

(continues on next page)

(continued from previous page)

```
                                   product_weight_g            product_length_cm      ␣
↪      -1.542839
pc_gamer                           product_width_cm            product_length_cm      ␣
↪      -1.558266
                                   product_length_cm           product_width_cm       ␣
↪      -1.558266
Length: 5836, dtype: float64
```

wow! we seem to be having very strong correlation increases up to 99%!? Is this possible? We should be very suspicious about these results, lets us find out why there are this high increases by calculating the initial correlation of 'security_and_services'

```
pricing_p_cat_corr = products_w_price.groupby('product_category_name').apply(
    lambda x: x.loc[:,(x.dtypes == float).values].corr(method='spearman')
    )
```

```
pricing_p_cat_corr.loc[('security_and_services','price')]
```

```
product_name_lenght           1.0
product_description_lenght    1.0
product_photos_qty           -1.0
product_weight_g              1.0
product_length_cm             1.0
product_height_cm             1.0
product_width_cm              1.0
price                         1.0
freight_value                 1.0
Name: (security_and_services, price), dtype: float64
```

```
pricing_corr.loc['price']
```

```
product_name_lenght           0.026564
product_description_lenght    0.218892
product_photos_qty            0.026766
product_weight_g              0.524087
product_length_cm             0.260411
product_height_cm             0.356680
product_width_cm              0.274180
price                         1.000000
freight_value                 0.445154
Name: price, dtype: float64
```

This is not normal, a perfect correlation might indicate a category with only one record, let us print the subset of data belonging to this category

```
products_w_price[products_w_price['product_category_name']=='security_and_services']
```

```
                                  product_category_name  ...  freight_value
bede3503afed051733eeb4a84d1adcc5  security_and_services  ...          15.45
2c4ada2e75c2ad41dd93cebb5df5f023  security_and_services  ...          25.77

[2 rows x 10 columns]
```

Dealing with small subsets in data

as expected, we only have 2 item here making things a lot more complicated. We can solve this by making a compromise, since predicting prices for categories (of there is a difference in categories) with little to no examples is inaccurate, we can choose to drop all small categories. This means that our prediction is not capable for certain items however.

```
category_sizes = products_w_price.groupby('product_category_name').size().sort_
↪values()
small_categories = list(category_sizes[category_sizes<50].index.values)
small_categories
```

```
['security_and_services',
 'fashion_childrens_clothes',
 'pc_gamer',
 'cds_dvds_musicals',
 'la_cuisine',
 'portateis_cozinha_e_preparadores_de_alimentos',
 'home_comfort_2',
 'flowers',
 'arts_and_craftmanship',
 'diapers_and_hygiene',
 'fashion_sport',
 'party_supplies',
 'music',
 'fashio_female_clothing',
 'furniture_mattress_and_upholstery']
```

We opted for a minimum of 50 items per category, let's see how that improves our relative correlations:

```
pricing_corr_stacked.drop(index=small_categories).sort_values(ascending=False)
```

```
product_category_name
fashion_underwear_beach  product_photos_qty          product_height_cm            0.
↪849303
                         product_height_cm           product_photos_qty           0.
↪849303
christmas_supplies       product_width_cm            product_description_lenght   0.
↪838264
                         product_description_lenght  product_width_cm             0.
↪838264
                         product_length_cm           product_description_lenght   0.
↪789431
                                                                                    .
↪..
fashion_underwear_beach  product_weight_g            product_length_cm           -1.
↪037104
books_imported           price                       product_width_cm            -1.
↪136276
                         product_width_cm            price                       -1.
↪136276
fashion_shoes            product_length_cm           product_width_cm            -1.
↪273546
                         product_width_cm            product_length_cm           -1.
↪273546
Length: 4698, dtype: float64
```

Now we filtered out smaller categories that might have high fluctuations, however we are not interested into correlations between any 2 columns (keep your goals in mind!) so we are going to filter only the price. I even found a method (xs) which I never use myself, google is your friend!

```
pricing_corr_stacked.drop(index=small_categories).xs('price', level=1, drop_
↪level=False).sort_values(ascending=False)
```

```
product_category_name
small_appliances_home_oven_and_coffee  price  product_photos_qty            0.705878
computers                              price  product_photos_qty            0.703822
furniture_bedroom                      price  product_photos_qty            0.638002
home_confort                           price  product_name_lenght           0.631313
fashion_shoes                          price  product_name_lenght           0.604471
                                                                              ...
computers                              price  product_height_cm            -0.810059
construction_tools_lights              price  product_description_lenght   -0.812678
computers                              price  product_length_cm            -0.819188
                                              product_weight_g             -0.984040
books_imported                         price  product_width_cm             -1.136276
Length: 522, dtype: float64
```

Ok, here I personally believe we have something we can work with! We can clearly see a relative change for correlation with certain columns. One thing that still remains is to filter per category the most important change compared to the average correlation

```
pricing_most_important = pricing_corr_stacked.drop(index=small_categories).xs('price',
↪ level=1, drop_level=True).sort_values(ascending=False).reset_index().drop_
↪duplicates(subset=['product_category_name']).set_index('product_category_name')
pricing_most_important.columns = ['parameter', 'relative_correlation']
pricing_most_important.head(10)
```

```
                                                        parameter  relative_
↪correlation
product_category_name
small_appliances_home_oven_and_coffee          product_photos_qty              0.
↪705878
computers                                      product_photos_qty              0.
↪703822
furniture_bedroom                              product_photos_qty              0.
↪638002
home_confort                                  product_name_lenght              0.
↪631313
fashion_shoes                                 product_name_lenght              0.
↪604471
fashion_underwear_beach                        product_photos_qty              0.
↪577173
cine_photo                             product_description_lenght              0.
↪554634
electronics                                    product_photos_qty              0.
↪547695
fixed_telephony                        product_description_lenght              0.
↪524150
christmas_supplies                              product_length_cm              0.
↪513941
```

```
pricing_least_important = pricing_corr_stacked.drop(index=small_categories).xs('price
↪', level=1, drop_level=True).sort_values(ascending=False).reset_index().drop_
↪duplicates(subset=['product_category_name'], keep='last').set_index('product_
↪category_name')
```

(continues on next page)

(continued from previous page)

```
pricing_least_important.columns = ['parameter', 'relative_correlation']
pricing_least_important.tail(10)
```

```
                                                 parameter  relative_correlation
product_category_name
home_confort                             product_length_cm             -0.444956
furniture_bedroom                      product_name_lenght             -0.472171
fashion_shoes                           product_photos_qty             -0.491136
furniture_living_room           product_description_lenght             -0.507893
audio                                  product_name_lenght             -0.530846
industry_commerce_and_business           product_height_cm             -0.596886
fashion_underwear_beach                  product_length_cm             -0.629531
construction_tools_lights       product_description_lenght             -0.812678
computers                                 product_weight_g             -0.984040
books_imported                            product_width_cm             -1.136276
```

What we can distill here:

- the quantity of photo's is important for small applicances, computers, furniture,… which is to be expected because you are willing to pay more if you are sure it looks like you want it to look
- the weight of fasion accessories and 'industry commerce' is not as important compared to other categories, as these things are always light, expensive or not

Anyway, now it is up to you to further interpret these values, but I think this should already give a nice idea on how we can estimate prices and how this changes per category.

35.6 Visualization

35.6.1 Product pricing

Now that we done the exploration, we can back our hypothesi up with some visual representations, many plots you will make will not end up in the final product but are meant to give you a more clear view on the situation itself

Normal distribution

In the exploration we talked about the non normal distribution of our dataset, let us plot the numerical columns into histograms to verify this. Fortunately, pandas has a built-in hist method that works perfect.

```
products_w_price.hist(figsize=(16,8), layout=(2,5));
```

```
<Figure size 1152x576 with 10 Axes>
```

ouch! this doesn't look normally distributed at all, we can also put it into a boxplot and compare with a bar plot

```
ax = sns.boxplot(data=products_w_price.loc[:,(products_w_price.dtypes == float).
↪values].stack().reset_index(), x='level_1', y=0)
ax.set_yscale('log')
ax.set_xticklabels(ax.get_xticklabels(),rotation=-20,horizontalalignment='left');
```

```
Output hidden; open in https://colab.research.google.com to view.
```

```
ax = sns.barplot(data=products_w_price.loc[:,(products_w_price.dtypes == float).
↪values].stack().reset_index(), x='level_1', y=0)
ax.set_yscale('log')
ax.set_xticklabels(ax.get_xticklabels(),rotation=-20,horizontalalignment='left');
```

```
<Figure size 432x288 with 1 Axes>
```

These 2 plots look alike, but in my opinion the first clearly shows that the peak consists out of outliers, hence the non normal distribution. Can you find the column responsible for this peak using the histograms?

Numerical correlation

We saw there were some numerical correlations within the dataset, let us try to visualize these, the first thing that pops into my mind is the pairplot.

```
#sns.pairplot(data=products_w_price.loc[:,(products_w_price.dtypes == float).values].
↪dropna())
```

hmm it seems that in this case the pairplot doesn't seem to be that conclusive, but we already knew that the correlations werent that appearent. Let us keep it simple and make a heatmap of the correlation statistic!

```
sns.heatmap(products_w_price.loc[:,(products_w_price.dtypes == float).values].
↪corr(method='spearman'), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5352774a10>
```

```
<Figure size 432x288 with 2 Axes>
```

ok this is basically the same as in the exploration but with colors, these colors however give us a good way to group correlations, we can see that the width, height, length and weight create a nice block, and are also correlated with the price.

Variance Inflation

We looked into the inflation inbetween those correlated columns, because it might be that they are telling the same story. To illustrate this information we can use a bar chart.

```
pd.Series(vif_price).plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f53542a3410>
```

```
<Figure size 432x288 with 1 Axes>
```

This way we can see that both the product length and width are highly correlated with other columns in the dataset i.e. their variation is explainable by other columns in the dataset. We opted to not remove any parameters here.

Categorical correlation

We performed anova tests to know if and how much variance there is between categories for each numerical column. we can use a bar plot to visualize.

```
anova_price.plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5354828950>
```

```
<Figure size 432x288 with 1 Axes>
```

This might not be my best plot I ever made, but it quantifies the amount of variation of a numerical column compared to the category column 'category' item specs as size vary more whilst the name and description remain much more the same. In this plot, you can see I made a crucial mistake by using the same axis range for the test statistic and the p-value, which is much smaller (between 0-1). Don't do this yourself! (I didn't bother as all p-values are 0 except for order_item_id)

Grouping by category

As last we grouped by category and recalculated the numerical correlation for each category apart. Note that we removed lowly populated categories as the prediction of the price might be not representative. I will use a boxplot to show any variation

```
products_w_price_sorted_price = products_w_price.groupby('product_category_name').
↪median().sort_values('price').index
products_w_price_sorted_price
```

```
CategoricalIndex(['home_comfort_2', 'dvds_blu_ray', 'electronics', 'flowers',
                  'telephony', 'portateis_cozinha_e_preparadores_de_alimentos',
                  'diapers_and_hygiene', 'fixed_telephony', 'food_drink',
                  'books_general_interest', 'drinks', 'home_appliances',
                  'fashion_bags_accessories', 'arts_and_craftmanship',
                  'fashio_female_clothing', 'cds_dvds_musicals', 'food',
                  'books_technical', 'christmas_supplies',
                  'costruction_tools_garden', 'fashion_underwear_beach',
                  'fashion_male_clothing', 'garden_tools', 'fashion_sport',
                  'books_imported', 'music', 'housewares', 'market_place',
                  'consoles_games', 'party_supplies', 'furniture_decor',
                  'costruction_tools_tools', 'stationery', 'baby',
                  'signaling_and_security', 'computers_accessories',
                  'bed_bath_table', 'auto', 'fashion_shoes', 'cine_photo',
                  'health_beauty', 'furniture_mattress_and_upholstery', 'toys',
                  'sports_leisure', 'audio', 'home_confort', 'perfumery',
                  'fashion_childrens_clothes', 'pet_shop',
                  'construction_tools_construction', 'tablets_printing_image',
                  'art', 'musical_instruments', 'luggage_accessories',
                  'industry_commerce_and_business', 'furniture_living_room',
                  'small_appliances', 'home_construction',
                  'kitchen_dining_laundry_garden_furniture',
                  'construction_tools_safety', 'pc_gamer', 'cool_stuff',
                  'la_cuisine', 'air_conditioning', 'watches_gifts',
                  'security_and_services', 'office_furniture',
                  'construction_tools_lights', 'furniture_bedroom',
                  'home_appliances_2', 'agro_industry_and_commerce',
                  'small_appliances_home_oven_and_coffee', 'computers'],
                 categories=['agro_industry_and_commerce', 'food', 'food_drink', 'art
↪', 'arts_and_craftmanship', 'party_supplies', 'christmas_supplies', 'audio', ...],
↪ordered=False, name='product_category_name', dtype='category')
```

(continues on next page)

(continued from previous page)

```

```

```
ax = sns.boxplot(data=products_w_price, x='product_category_name', y='price',␣
↪order=products_w_price_sorted_price)
ax.set(yscale="log")
ax.set_xticklabels(ax.get_xticklabels(),rotation=-20,horizontalalignment='left');
```

```
<Figure size 432x288 with 1 Axes>
```

cool! here we can see the variation in groups for the price column, this way we can deduce wich categories are highly priced and which are lowly priced. Our machine learning solution later will use this information to help decide the price (if we of course use it to train the model). We can conclude that while the variation in each category can be high, there is a trend in price between categories.

We also calculated relative changes of correlation between price and other numerical columns inbetween categories. Let's see if we can visualize that information, my best guess would be a bar chart

```
pricing_most_important.head()
```

```
                                                parameter  relative_correlation
product_category_name
small_appliances_home_oven_and_coffee   product_photos_qty              0.705878
computers                               product_photos_qty              0.703822
furniture_bedroom                       product_photos_qty              0.638002
home_confort                           product_name_lenght              0.631313
fashion_shoes                          product_name_lenght              0.604471
```

```
top_n = 10
ax = sns.barplot(x=pricing_most_important.head(top_n).index.to_list(), y=pricing_most_
↪important.head(top_n)['relative_correlation'], alpha=0.7, palette='colorblind')
for idx, p in enumerate(ax.patches):
    ax.annotate(pricing_most_important.head(top_n)['parameter'][idx],
                   (p.get_x() + p.get_width() / 2., 0),
                   ha = 'center', va = 'bottom',
                   xytext = (0, 9),
                rotation = 90,
                  color='white',
                   textcoords = 'offset points')
    ax.annotate(format(p.get_height(), '.1f'),
                   (p.get_x() + p.get_width() / 2., p.get_height()*0.9),
                   ha = 'center', va = 'center',
                   xytext = (0, 9),
                   textcoords = 'offset points')
ax.set_xticklabels(ax.get_xticklabels(),rotation=-20,horizontalalignment='left');
```

```
<Figure size 432x288 with 1 Axes>
```

You can see I put a little bit more effort in this last graph as I think this is the nice visualisation to show others. We can also make a similar plot but with the relatively least important features.

```
top_n = 10
ax = sns.barplot(x=pricing_least_important.tail(top_n).index.to_list(), y=pricing_
↪least_important.tail(top_n)['relative_correlation'], alpha=0.7, palette='colorblind
↪')
for idx, p in enumerate(ax.patches):
```

(continues on next page)

(continued from previous page)

```
    ax.annotate(pricing_least_important.tail(top_n)['parameter'][idx],
                   (p.get_x() + p.get_width() / 2., 0),
                   ha = 'center', va = 'top',
                   xytext = (0, -9),
                rotation = -90,
                  color='white',
                   textcoords = 'offset points')
    ax.annotate(format(p.get_height(), '.1f'),
                   (p.get_x() + p.get_width() / 2., p.get_height()*0.9),
                   ha = 'center', va = 'center',
                   xytext = (0, 9),
                   textcoords = 'offset points')
ax.set_xticklabels(ax.get_xticklabels(),rotation=-20,horizontalalignment='left');
```

```
<Figure size 432x288 with 1 Axes>
```

35.7 Summary

35.7.1 Product pricing

To conclude the product pricing analysis, we checked for normal distributions which werent present, so we had to opt for non-parametric/non-linear methods (although in many cases these will still do fine). We checked for numerical correlations but these were not really interesting, which led to the idea that perhaps per category our price could be predicted more accurate. This was proven by the fact that our price surely differs inbetween categories.

We split up our dataset by grouping per category and removing small categories, now we could see that a relative change in correlation - meaning that the correlation of a column in our dataset with the price was different in that category compared to the overall correlation of this column with the price - was present for all categories. For each category we selected both the highest increase in correlation - meaning a 'spike' in importance - for that category and the highest decrease - meaning a 'drop' in importance - for that category.

These plots hence show the most important and least important attributes for an item concerning the price e.g. if we want to increase the price of an item in the computers category, we need to make sure it has enough pictures and not try to decrease the weight value.

CHAPTER THIRTYSIX

CASE STUDY: CHURN

In this case study we try to create an answer why customers have left our service, a telecom operator.

The case study is divided into several parts:

- Goals
- Parsing
- Preparation (cleaning)
- Processing
- Exploration
- Visualization
- Conclusion

36.1 Goals

In this section we define questions that will be our guideline througout the case study

- Why are customers leaving us?
- Can we cluster types of customers?

We'll (try to) keep these question in mind when performing the case study.

36.2 Parsing

we start out by importing all libraries

```
import os
import json
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats
import sklearn
import matplotlib.pyplot as plt
from IPython.display import set_matplotlib_formats
%matplotlib inline
```

in order to download datasets from kaggle, we need an API key to access their API, we'll make that here

```
if not os.path.exists(os.path.expanduser('~/.kaggle')):
    os.mkdir(os.path.expanduser('~/.kaggle'))

with open(os.path.expanduser('~/.kaggle/kaggle.json'), 'w') as f:
    json.dump(
        {
            "username":"lorenzf",
            "key":"7a44a9e99b27e796177d793a3d85b8cf"
        }
        , f)
```

now we can import kaggle too and download the datasets

```
import kaggle
kaggle.api.dataset_download_files(dataset='blastchar/telco-customer-churn', path='./
↪data', unzip=True)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_8931/3288988394.py in <module>
----> 1 import kaggle
      2 kaggle.api.dataset_download_files(dataset='blastchar/telco-customer-churn',␣
↪path='./data', unzip=True)

ModuleNotFoundError: No module named 'kaggle'
```

the csv files are now in the './data' folder, we can now read them using pandas, here is the list of all csv files in our folder

```
os.listdir('./data')
```

```
['WA_Fn-UseC_-Telco-Customer-Churn.csv']
```

This dataset only contains 1 file, in it each row has all the information about a single customer and which services he or she has or had before churning.

```
churn_df = pd.read_csv('./data/WA_Fn-UseC_-Telco-Customer-Churn.csv')
print('shape: ' + str(churn_df.shape))
churn_df.head()
```

```
shape: (7043, 21)
```

```
   customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No
1  5575-GNVDE    Male              0      No         No      34          Yes
2  3668-QPYBK    Male              0      No         No       2          Yes
3  7795-CFOCW    Male              0      No         No      45           No
4  9237-HQITU  Female              0      No         No       2          Yes

      MultipleLines InternetService OnlineSecurity  ... DeviceProtection  \
0  No phone service             DSL             No  ...               No
1                No             DSL            Yes  ...              Yes
2                No             DSL            Yes  ...               No
3  No phone service             DSL            Yes  ...              Yes
4                No     Fiber optic             No  ...               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
```

(continues on next page)

(continued from previous page)

```
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

              PaymentMethod MonthlyCharges  TotalCharges Churn
0          Electronic check          29.85         29.85    No
1              Mailed check          56.95        1889.5    No
2              Mailed check          53.85        108.15   Yes
3  Bank transfer (automatic)          42.30       1840.75    No
4          Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
```

Looks like there is some personal info and the configuration of the service, such as if they had an internet service, with or without options such as security, backup,… By the lookds of it these Yes/No answers are not booleans (i.e. 2 options) but rather categories as they have a third option, 'No … service'.

36.3 Preparation

here we perform tasks to prepare the data in a more pleasing format.

36.3.1 Data Types

Before we do anything with our data, it is good to see if our data types are in order

```
churn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
```

(continues on next page)

(continued from previous page)

```
 18   MonthlyCharges    7043 non-null   float64
 19   TotalCharges      7043 non-null   object
 20   Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

I am opting to change the sernior citizan from 0/1 to No/Yes and convert them all to categories, let's do that right now.

```
churn_df.SeniorCitizen = churn_df.SeniorCitizen.map({0: 'No', 1:'Yes'})
churn_df[['gender', 'SeniorCitizen', 'Partner','Dependents', 'PhoneService',
↪'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
↪'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
↪'PaperlessBilling', 'PaymentMethod', 'Churn']] = churn_df[['gender', 'SeniorCitizen
↪', 'Partner','Dependents', 'PhoneService', 'MultipleLines', 'InternetService',
↪'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
↪'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn']].
↪astype('category')
churn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   category
 2   SeniorCitizen     7043 non-null   category
 3   Partner           7043 non-null   category
 4   Dependents        7043 non-null   category
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   category
 7   MultipleLines     7043 non-null   category
 8   InternetService   7043 non-null   category
 9   OnlineSecurity    7043 non-null   category
 10  OnlineBackup      7043 non-null   category
 11  DeviceProtection  7043 non-null   category
 12  TechSupport       7043 non-null   category
 13  StreamingTV       7043 non-null   category
 14  StreamingMovies   7043 non-null   category
 15  Contract          7043 non-null   category
 16  PaperlessBilling  7043 non-null   category
 17  PaymentMethod     7043 non-null   category
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   category
dtypes: category(17), float64(1), int64(1), object(2)
memory usage: 339.4+ KB
```

Now our yes/no answers are configured as categories, for numbers we see that there are 2: 'MontlyCharges' and 'TotalCharges'. I'm going to make them floating numbers

```
churn_df[['MonthlyCharges', 'TotalCharges']] = churn_df[['MonthlyCharges',
↪'TotalCharges']].astype('float')
churn_df.info()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_6003/2494845660.py in <module>
----> 1 churn_df[['MonthlyCharges', 'TotalCharges']] = churn_df[['MonthlyCharges',
↪'TotalCharges']].astype('float')
      2 churn_df.info()

~/.local/lib/python3.8/site-packages/pandas/core/generic.py in astype(self, dtype,␣
↪copy, errors)
   5813         else:
   5814             # else, only a single dtype is given
-> 5815             new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   5816             return self._constructor(new_data).__finalize__(self, method=
↪"astype")
   5817 

~/.local/lib/python3.8/site-packages/pandas/core/internals/managers.py in astype(self,
↪ dtype, copy, errors)
    416 
    417     def astype(self: T, dtype, copy: bool = False, errors: str = "raise") ->␣
↪T:
--> 418         return self.apply("astype", dtype=dtype, copy=copy, errors=errors)
    419 
    420     def convert(

~/.local/lib/python3.8/site-packages/pandas/core/internals/managers.py in apply(self,␣
↪f, align_keys, ignore_failures, **kwargs)
    325                     applied = b.apply(f, **kwargs)
    326                 else:
--> 327                     applied = getattr(b, f)(**kwargs)
    328             except (TypeError, NotImplementedError):
    329                 if not ignore_failures:

~/.local/lib/python3.8/site-packages/pandas/core/internals/blocks.py in astype(self,␣
↪dtype, copy, errors)
    590         values = self.values
    591 
--> 592         new_values = astype_array_safe(values, dtype, copy=copy,␣
↪errors=errors)
    593 
    594         new_values = maybe_coerce_values(new_values)

~/.local/lib/python3.8/site-packages/pandas/core/dtypes/cast.py in astype_array_
↪safe(values, dtype, copy, errors)
   1307 
   1308     try:
-> 1309         new_values = astype_array(values, dtype, copy=copy)
   1310     except (ValueError, TypeError):
   1311         # e.g. astype_nansafe can fail on object-dtype of strings

~/.local/lib/python3.8/site-packages/pandas/core/dtypes/cast.py in astype_
↪array(values, dtype, copy)
   1255 
   1256     else:
-> 1257         values = astype_nansafe(values, dtype, copy=copy)
   1258 
   1259     # in pandas we don't store numpy str dtypes, so convert to object
```

(continues on next page)

(continued from previous page)

```
~/.local/lib/python3.8/site-packages/pandas/core/dtypes/cast.py in astype_nansafe(arr,
↪ dtype, copy, skipna)
   1093     if arr.ndim > 1:
   1094         flat = arr.ravel()
-> 1095         result = astype_nansafe(flat, dtype, copy=copy, skipna=skipna)
   1096         # error: Item "ExtensionArray" of "Union[ExtensionArray, ndarray]"␣
↪has no
   1097         # attribute "reshape"

~/.local/lib/python3.8/site-packages/pandas/core/dtypes/cast.py in astype_nansafe(arr,
↪ dtype, copy, skipna)
   1199     if copy or is_object_dtype(arr.dtype) or is_object_dtype(dtype):
   1200         # Explicit copy, or required since NumPy can't view from / to object.
-> 1201         return arr.astype(dtype, copy=True)
   1202
   1203     return arr.astype(dtype, copy=copy)

ValueError: could not convert string to float: ''
```

Looks like we have encountered some problems, there are strings in the Total charges that are not able to be converted to a decimal number. We print out the rows that create an error and observe.

```
churn_df[pd.to_numeric(churn_df.TotalCharges,errors='coerce').isna()]
```

```
      customerID  gender SeniorCitizen Partner Dependents  tenure  \
488   4472-LVYGI  Female            No     Yes        Yes       0
753   3115-CZMZD    Male            No      No        Yes       0
936   5709-LVOEQ  Female            No     Yes        Yes       0
1082  4367-NUYAO    Male            No     Yes        Yes       0
1340  1371-DWPAZ  Female            No     Yes        Yes       0
3331  7644-OMVMY    Male            No     Yes        Yes       0
3826  3213-VVOLG    Male            No     Yes        Yes       0
4380  2520-SGTTA  Female            No     Yes        Yes       0
5218  2923-ARZLG    Male            No     Yes        Yes       0
6670  4075-WKNIU  Female            No     Yes        Yes       0
6754  2775-SEFEE    Male            No      No        Yes       0

     PhoneService     MultipleLines InternetService       OnlineSecurity  ...  \
488            No  No phone service             DSL                  Yes  ...
753           Yes                No              No  No internet service  ...
936           Yes                No             DSL                  Yes  ...
1082          Yes               Yes              No  No internet service  ...
1340           No  No phone service             DSL                  Yes  ...
3331          Yes                No              No  No internet service  ...
3826          Yes               Yes              No  No internet service  ...
4380          Yes                No              No  No internet service  ...
5218          Yes                No              No  No internet service  ...
6670          Yes               Yes             DSL                   No  ...
6754          Yes               Yes             DSL                  Yes  ...

         DeviceProtection          TechSupport          StreamingTV  \
488                   Yes                  Yes                  Yes
753   No internet service  No internet service  No internet service
936                   Yes                   No                  Yes
1082  No internet service  No internet service  No internet service
```

(continues on next page)

(continued from previous page)

```
1340                 Yes                 Yes                 Yes
3331  No internet service  No internet service  No internet service
3826  No internet service  No internet service  No internet service
4380  No internet service  No internet service  No internet service
5218  No internet service  No internet service  No internet service
6670                 Yes                 Yes                 Yes
6754                  No                 Yes                  No

          StreamingMovies  Contract PaperlessBilling  \
488                    No  Two year              Yes
753   No internet service  Two year               No
936                   Yes  Two year               No
1082  No internet service  Two year               No
1340                   No  Two year               No
3331  No internet service  Two year               No
3826  No internet service  Two year               No
4380  No internet service  Two year               No
5218  No internet service  One year              Yes
6670                   No  Two year               No
6754                   No  Two year              Yes

                  PaymentMethod MonthlyCharges  TotalCharges Churn
488   Bank transfer (automatic)          52.55                  No
753                Mailed check          20.25                  No
936                Mailed check          80.85                  No
1082               Mailed check          25.75                  No
1340    Credit card (automatic)          56.05                  No
3331               Mailed check          19.85                  No
3826               Mailed check          25.35                  No
4380               Mailed check          20.00                  No
5218               Mailed check          19.70                  No
6670               Mailed check          73.35                  No
6754  Bank transfer (automatic)          61.90                  No

[11 rows x 21 columns]
```

Seems that there are some customers being so new they have no total charges, for convenience i'm going to change the space to a 0.

```
churn_df.TotalCharges = churn_df.TotalCharges.replace(' ', '0')
```

```
churn_df[['MonthlyCharges', 'TotalCharges']] = churn_df[['MonthlyCharges',
↪'TotalCharges']].astype('float')
churn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   category
 2   SeniorCitizen     7043 non-null   category
 3   Partner           7043 non-null   category
 4   Dependents        7043 non-null   category
 5   tenure            7043 non-null   int64
```

(continues on next page)

(continued from previous page)

```
 6   PhoneService      7043 non-null   category
 7   MultipleLines     7043 non-null   category
 8   InternetService   7043 non-null   category
 9   OnlineSecurity    7043 non-null   category
 10  OnlineBackup      7043 non-null   category
 11  DeviceProtection  7043 non-null   category
 12  TechSupport       7043 non-null   category
 13  StreamingTV       7043 non-null   category
 14  StreamingMovies   7043 non-null   category
 15  Contract          7043 non-null   category
 16  PaperlessBilling  7043 non-null   category
 17  PaymentMethod     7043 non-null   category
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   float64
 20  Churn             7043 non-null   category
dtypes: category(17), float64(2), int64(1), object(1)
memory usage: 339.4+ KB
```

36.3.2 Missing values

for each dataframe we apply a few checks in order to see the quality of data

```
print(100*churn_df.isna().sum()/churn_df.shape[0])
```

```
customerID          0.0
gender              0.0
SeniorCitizen       0.0
Partner             0.0
Dependents          0.0
tenure              0.0
PhoneService        0.0
MultipleLines       0.0
InternetService     0.0
OnlineSecurity      0.0
OnlineBackup        0.0
DeviceProtection    0.0
TechSupport         0.0
StreamingTV         0.0
StreamingMovies     0.0
Contract            0.0
PaperlessBilling    0.0
PaymentMethod       0.0
MonthlyCharges      0.0
TotalCharges        0.0
Churn               0.0
dtype: float64
```

No missing values (if we do not count the ones we solved earlier), sometimes luck is on our side.

36.3.3 Duplicates

For any reason, our dataset might be containing duplicates that would be counted twice and will introduce a bias we would not want. On the other hand, duplicates can be subjected to interpretation, here we would say that if 2 records are completely the same they are duplicates.

```
churn_df.duplicated().any()
```

```
False
```

36.3.4 Indexing

It is more convenient to work with an index, our dataset already contains an id which we can use as index

```
churn_df = churn_df.set_index('customerID')
churn_df.head()
```

```
            gender SeniorCitizen Partner Dependents  tenure PhoneService  \
customerID
7590-VHVEG  Female            No     Yes         No       1           No
5575-GNVDE    Male            No      No         No      34          Yes
3668-QPYBK    Male            No      No         No       2          Yes
7795-CFOCW    Male            No      No         No      45           No
9237-HQITU  Female            No      No         No       2          Yes

               MultipleLines InternetService OnlineSecurity OnlineBackup  \
customerID
7590-VHVEG  No phone service             DSL             No          Yes
5575-GNVDE                No             DSL            Yes           No
3668-QPYBK                No             DSL            Yes          Yes
7795-CFOCW  No phone service             DSL            Yes           No
9237-HQITU                No     Fiber optic             No           No

           DeviceProtection TechSupport StreamingTV StreamingMovies  \
customerID
7590-VHVEG               No          No          No              No
5575-GNVDE              Yes          No          No              No
3668-QPYBK               No          No          No              No
7795-CFOCW              Yes         Yes          No              No
9237-HQITU               No          No          No              No

                  Contract PaperlessBilling              PaymentMethod  \
customerID
7590-VHVEG  Month-to-month              Yes           Electronic check
5575-GNVDE        One year               No               Mailed check
3668-QPYBK  Month-to-month              Yes               Mailed check
7795-CFOCW        One year               No  Bank transfer (automatic)
9237-HQITU  Month-to-month              Yes           Electronic check

            MonthlyCharges  TotalCharges Churn
customerID
7590-VHVEG           29.85         29.85    No
5575-GNVDE           56.95       1889.50    No
3668-QPYBK           53.85        108.15   Yes
7795-CFOCW           42.30       1840.75    No
9237-HQITU           70.70        151.65   Yes
```

36.4 Processing

36.4.1 Churn vs no churn

I would like to compare between persons that have churned and others, therefore a function that calculates the counts between churn and a given column would be convenient. By using functions I keep things dynamic without having to store a dataframe for each column, but static dataframes work equally well!

```
def count_matrix(col_name):
    return churn_df.groupby(['Churn', col_name]).size().unstack()
```

```
count_matrix('DeviceProtection')
```

```
DeviceProtection    No  No internet service   Yes
Churn
No                1884                 1413  1877
Yes               1211                  113   545
```

aside from the counts I would also like to know the mean, as some groups have a smaller population yet their proportion of churned persons might be higher.

```
def mean_matrix(col_name):
    df = churn_df.groupby(['Churn', col_name]).size().unstack()
    return df.divide(df.sum(axis='index'),axis='columns')
```

```
mean_matrix('DeviceProtection')
```

```
DeviceProtection        No  No internet service       Yes
Churn
No                0.608724              0.92595  0.774979
Yes               0.391276              0.07405  0.225021
```

out of curiosity, let's print all those 'mean matrices'

```
for col in churn_df.columns.drop('Churn'):
  print(mean_matrix(col))
  print()
```

```
gender    Female      Male
Churn
No      0.730791  0.738397
Yes     0.269209  0.261603

SeniorCitizen        No       Yes
Churn
No             0.763938  0.583187
Yes            0.236062  0.416813

Partner        No       Yes
Churn
No        0.67042  0.803351
Yes       0.32958  0.196649

Dependents         No       Yes
```

(continues on next page)

(continued from previous page)

```
Churn
No          0.687209  0.845498
Yes         0.312791  0.154502

tenure   0         1         2     3         4         5         6         7   \
Churn
No      1.0  0.380098  0.483193  0.53  0.528409  0.518797  0.636364  0.610687
Yes     0.0  0.619902  0.516807  0.47  0.471591  0.481203  0.363636  0.389313

tenure        8         9   ...        63    64        65        66        67  \
Churn                       ...
No      0.658537  0.613445  ...  0.944444  0.95  0.881579  0.853933  0.897959
Yes     0.341463  0.386555  ...  0.055556  0.05  0.118421  0.146067  0.102041

tenure    68        69        70        71        72
Churn
No      0.91  0.915789  0.907563  0.964706  0.983425
Yes     0.09  0.084211  0.092437  0.035294  0.016575

[2 rows x 73 columns]

PhoneService        No       Yes
Churn
No            0.750733  0.732904
Yes           0.249267  0.267096

MultipleLines        No  No phone service       Yes
Churn
No             0.749558          0.750733  0.713901
Yes            0.250442          0.249267  0.286099

InternetService       DSL  Fiber optic       No
Churn
No               0.810409     0.581072  0.92595
Yes              0.189591     0.418928  0.07405

OnlineSecurity         No  No internet service       Yes
Churn
No              0.582333              0.92595  0.853888
Yes             0.417667              0.07405  0.146112

OnlineBackup        No  No internet service       Yes
Churn
No            0.600712              0.92595  0.784685
Yes           0.399288              0.07405  0.215315

DeviceProtection        No  No internet service       Yes
Churn
No                0.608724              0.92595  0.774979
Yes               0.391276              0.07405  0.225021

TechSupport        No  No internet service       Yes
Churn
No           0.583645              0.92595  0.848337
Yes          0.416355              0.07405  0.151663

StreamingTV        No  No internet service       Yes
```

(continues on next page)

(continued from previous page)

```
Churn
No          0.664769              0.92595  0.699298
Yes         0.335231              0.07405  0.300702

StreamingMovies        No  No internet service       Yes
Churn
No               0.663196              0.92595  0.700586
Yes              0.336804              0.07405  0.299414

Contract  Month-to-month  One year  Two year
Churn
No              0.572903  0.887305  0.971681
Yes             0.427097  0.112695  0.028319

PaperlessBilling        No       Yes
Churn
No                0.836699  0.664349
Yes               0.163301  0.335651

PaymentMethod  Bank transfer (automatic)  Credit card (automatic)  \
Churn
No                              0.832902                 0.847569
Yes                             0.167098                 0.152431

PaymentMethod  Electronic check  Mailed check
Churn
No                     0.547146      0.808933
Yes                    0.452854      0.191067

MonthlyCharges  18.25   18.40   18.55   18.70   18.75   18.80   18.85   \
Churn
No                 1.0     1.0     1.0     1.0     1.0     1.0     0.8
Yes                0.0     0.0     0.0     0.0     0.0     0.0     0.2

MonthlyCharges  18.90     18.95     19.00   ...  117.35  117.45  117.50  \
Churn                                       ...
No                 1.0  0.833333  0.857143  ...     1.0     0.0     1.0
Yes                0.0  0.166667  0.142857  ...     0.0     1.0     0.0

MonthlyCharges  117.60  117.80  118.20  118.35  118.60  118.65  118.75
Churn
No                 1.0     0.0     1.0     0.0     1.0     1.0     1.0
Yes                0.0     1.0     0.0     1.0     0.0     0.0     0.0

[2 rows x 1585 columns]

TotalCharges  0.00     18.80    18.85    18.90    19.00    19.05     19.10    \
Churn
No                1.0      1.0      0.5      1.0      1.0      1.0  0.666667
Yes               0.0      0.0      0.5      0.0      0.0      0.0  0.333333

TotalCharges  19.15    19.20     19.25    ...  8477.70  8496.70  8529.50  \
Churn                                     ...
No                1.0      1.0  0.666667  ...      1.0      1.0      1.0
Yes               0.0      0.0  0.333333  ...      0.0      0.0      0.0

TotalCharges  8543.25  8547.15  8564.75  8594.40  8670.10  8672.45  8684.80
```

(continues on next page)

(continued from previous page)

```
Churn
No                  1.0       1.0       1.0       1.0       1.0       1.0       0.0
Yes                 0.0       0.0       0.0       0.0       0.0       0.0       1.0

[2 rows x 6531 columns]
```

We already see some big differences between populations of churn and no churn for some of these features, promising!

36.4.2 one hot encoding

I would also like to run the data into an algorithm, yet computers don't like categories, so I 'one hot encode' the categories and get a column/feature for each category in my categorical variables.

```
churn_ohe_df = pd.concat(
    [
     pd.get_dummies(churn_df.drop(columns=['Churn'])),
     churn_df.Churn.eq('Yes').astype(int)
    ], axis='columns'
)
churn_ohe_df.head()
```

```
            tenure  MonthlyCharges  TotalCharges  gender_Female  gender_Male  \
customerID
7590-VHVEG       1           29.85         29.85              1            0
5575-GNVDE      34           56.95       1889.50              0            1
3668-QPYBK       2           53.85        108.15              0            1
7795-CFOCW      45           42.30       1840.75              0            1
9237-HQITU       2           70.70        151.65              1            0

            SeniorCitizen_No  SeniorCitizen_Yes  Partner_No  Partner_Yes  \
customerID
7590-VHVEG                 1                  0           0            1
5575-GNVDE                 1                  0           1            0
3668-QPYBK                 1                  0           1            0
7795-CFOCW                 1                  0           1            0
9237-HQITU                 1                  0           1            0

            Dependents_No  ...  Contract_Month-to-month  Contract_One year  \
customerID                 ...
7590-VHVEG              1  ...                        1                  0
5575-GNVDE              1  ...                        0                  1
3668-QPYBK              1  ...                        1                  0
7795-CFOCW              1  ...                        0                  1
9237-HQITU              1  ...                        1                  0

            Contract_Two year  PaperlessBilling_No  PaperlessBilling_Yes  \
customerID
7590-VHVEG                  0                    0                     1
5575-GNVDE                  0                    1                     0
3668-QPYBK                  0                    0                     1
7795-CFOCW                  0                    1                     0
9237-HQITU                  0                    0                     1

            PaymentMethod_Bank transfer (automatic)  \
customerID
```

(continues on next page)

(continued from previous page)

```
7590-VHVEG                                       0
5575-GNVDE                                       0
3668-QPYBK                                       0
7795-CFOCW                                       1
9237-HQITU                                       0

            PaymentMethod_Credit card (automatic)  \
customerID
7590-VHVEG                                     0
5575-GNVDE                                     0
3668-QPYBK                                     0
7795-CFOCW                                     0
9237-HQITU                                     0

            PaymentMethod_Electronic check  PaymentMethod_Mailed check  Churn
customerID
7590-VHVEG                               1                           0      0
5575-GNVDE                               0                           1      0
3668-QPYBK                               0                           1      1
7795-CFOCW                               0                           0      0
9237-HQITU                               1                           0      1

[5 rows x 47 columns]
```

36.4.3 correlation

I went ahead and already calculated the correlation matrix for this dataset, with the ohe version of the data we can figure out which categories are related. In the next cell I printed out all correlations with the churn feature.

```
churn_corr_df = churn_ohe_df.corr()
churn_corr_df['Churn']
```

```
tenure                                  -0.352229
MonthlyCharges                           0.193356
TotalCharges                            -0.198324
gender_Female                            0.008612
gender_Male                             -0.008612
SeniorCitizen_No                        -0.150889
SeniorCitizen_Yes                        0.150889
Partner_No                               0.150448
Partner_Yes                             -0.150448
Dependents_No                            0.164221
Dependents_Yes                          -0.164221
PhoneService_No                         -0.011942
PhoneService_Yes                         0.011942
MultipleLines_No                        -0.032569
MultipleLines_No phone service          -0.011942
MultipleLines_Yes                        0.040102
InternetService_DSL                     -0.124214
InternetService_Fiber optic              0.308020
InternetService_No                      -0.227890
OnlineSecurity_No                        0.342637
OnlineSecurity_No internet service      -0.227890
OnlineSecurity_Yes                      -0.171226
```

(continues on next page)

(continued from previous page)

```
OnlineBackup_No                             0.268005
OnlineBackup_No internet service           -0.227890
OnlineBackup_Yes                           -0.082255
DeviceProtection_No                         0.252481
DeviceProtection_No internet service       -0.227890
DeviceProtection_Yes                       -0.066160
TechSupport_No                              0.337281
TechSupport_No internet service            -0.227890
TechSupport_Yes                            -0.164674
StreamingTV_No                              0.128916
StreamingTV_No internet service            -0.227890
StreamingTV_Yes                             0.063228
StreamingMovies_No                          0.130845
StreamingMovies_No internet service        -0.227890
StreamingMovies_Yes                         0.061382
Contract_Month-to-month                     0.405103
Contract_One year                          -0.177820
Contract_Two year                          -0.302253
PaperlessBilling_No                        -0.191825
PaperlessBilling_Yes                        0.191825
PaymentMethod_Bank transfer (automatic)    -0.117937
PaymentMethod_Credit card (automatic)      -0.134302
PaymentMethod_Electronic check              0.301919
PaymentMethod_Mailed check                 -0.091683
Churn                                       1.000000
Name: Churn, dtype: float64
```

We can see that complementary categories show an inverse correlation, indicating that we are dealing with a excess of information. Logical as when option A is not chosen, option B is. However in this case, as some categoricals have 3 options I opt to keep all info, although it would be a good idea to remove 1 option for each category, this should become appearent in data exploration.

36.5 Exploration

Here we start with the exploration of our dataset, we look into normal distribution of numerical data, categorical correlations, numerical and categorical correlation, cluster results, and a simple machine learning implementation.

36.5.1 Normal distribution

As a precaution I will check the normality of our numerical data. Although most probably not essential for further analysis it might be useful later.

```
for name, col in churn_df[['tenure', 'MonthlyCharges', 'TotalCharges']].iteritems():
    print(name)
    print(scipy.stats.shapiro(col.dropna()))
```

```
tenure
ShapiroResult(statistic=0.9037491083145142, pvalue=0.0)
MonthlyCharges
ShapiroResult(statistic=0.9208902716636658, pvalue=0.0)
TotalCharges
ShapiroResult(statistic=0.8601524233818054, pvalue=0.0)
```

```
/home/lorenzf/.local/lib/python3.8/site-packages/scipy/stats/morestats.py:1760:
↪UserWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

It is clear that our numerical data is not normally distributed, as mentioned not essential, therefore I will not be transforming the data and keeping it as it is. This is useful later because we keep the meaning of the values.

36.5.2 Categorical correlations

We have a lot of categorical features that could correlate with our Churn parameter, for each of those we would like to know how strong their correlation is. We can use the count_matrix function we created earlier for this.

```
count_matrix('DeviceProtection')
```

```
DeviceProtection    No  No internet service   Yes
Churn
No                1884                 1413  1877
Yes               1211                  113   545
```

Using the Chi Squared Contingency test we can find out if any category of the chosen feature correlates with our Churn feature. it returns the test statistic F (strength of correlation), the p-value (chance of correlation) and expected values if no correlation is present.

```
F, p, df, exp = scipy.stats.chi2_contingency(count_matrix('DeviceProtection'))
```

Something I find interesting is to subtract the expected values from the true values, this case we see where the surplusses are.

```
count_matrix('DeviceProtection') - exp
```

```
DeviceProtection         No  No internet service        Yes
Churn
No               -389.68025           291.954423  97.725827
Yes               389.68025          -291.954423 -97.725827
```

To make our lives simpler, we extract all the categorical columns that we want to test against the Churn feature.

```
cat_cols = churn_df.columns.drop(['Churn', 'tenure', 'MonthlyCharges', 'TotalCharges
↪'])
cat_cols
```

```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
       'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'Contract', 'PaperlessBilling', 'PaymentMethod'],
      dtype='object')
```

Here I've written a small script that for each of those columns performs the Chi Squared test and writes the results down.

```
significant_cols = []
chi2_results = {}
for col in cat_cols:
    counts = count_matrix(col)
    F, p, df, exp = scipy.stats.chi2_contingency(counts)
```

(continues on next page)

(continued from previous page)

```
    if p<0.05:
        significant_cols.append(col)
        chi2_results[col] = {
            'F': F,
            'p': p,
            'real': counts,
            'exp': exp,
            'diff': counts - exp,
        }

# sort in descending F value
chi2_results = {x[0]: x[1] for x in sorted(chi2_results.items(), key=lambda x: x[1]['F
↪'], reverse=True)}
```

The features that are significant have a p-value less than 0.05, indicating only a 5% chance that this occurs randomly. We list them here

```
significant_cols
```

```
['SeniorCitizen',
 'Partner',
 'Dependents',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod']
```

Lets zoom into one of them, here we print the difference of the true values and the expected

```
chi2_results['SeniorCitizen']['diff']
```

```
SeniorCitizen          No         Yes
Churn
No             172.947608 -172.947608
Yes           -172.947608  172.947608
```

We can see that there are about 173 persons more in the group of SeniorCitizen that have Churned than was expected. Perhaps the provided service was not Senior friendly?

```
mean_matrix('SeniorCitizen')*100
```

```
SeniorCitizen         No        Yes
Churn
No             76.393832  58.318739
Yes            23.606168  41.681261
```

We can see the same pattern in our mean matrix, from the Senior Citizens about 18% more have churned than the non SeniorCitizen group! To make things more easier on the eye I've put it into a dataframe that is sorted by correlation strength

```
corr_df = pd.DataFrame(
    {
        'p': [chi2_results[col]['p'] for col in significant_cols],
        'F': [chi2_results[col]['F'] for col in significant_cols]
    }
    , index=significant_cols).sort_values('F', ascending=False)
```

We can see that features such as Contract type, OnlineSecurity and TechSupport have a strong correlation with Churning.

36.5.3 Numerical vs Categorical correlation

Next we would like to know if numerical features have a correlation with our Churn, using ANOVA we can mathematically calculate this. First let's look at the averages of tenure between Yes and No Churn.

```
churn_df.groupby('Churn').tenure.mean()
```

```
Churn
No     37.569965
Yes    17.979133
Name: tenure, dtype: float64
```

This is already a clear difference, but let's not jump to conclusion, ANOVA also takes into account group sizes and variation.

```
scipy.stats.f_oneway(
    churn_df[churn_df.Churn=='Yes'].tenure,
    churn_df[churn_df.Churn=='No'].tenure
)
```

```
F_onewayResult(statistic=997.2680104991438, pvalue=7.999057960610892e-205)
```

That p-values sure does speak for itself, there is a clear difference in tenures for users that have churned and others!

36.5.4 unsupervised clustering

Our customers also asked us to find out if we can find specific clusters of users in their dataset, so we perform a clustering analysis.

```
from sklearn.cluster import KMeans
```

We create a clustering algorithm and specify that we would like to have 2 clusters, perhaps they will overlap with churn and nochurn. Then we fit the algorithm with our dataset without the churn feature.

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(churn_ohe_df.drop(columns=['Churn']))
```

```
KMeans(n_clusters=2)
```

After training on the dataset we can ask it to give us the labels for each record that it assigned, [0, 1] are the 2 clusters that it used to seperate our data.

```
kmeans.labels_
```

```
array([0, 0, 0, ..., 0, 0, 1], dtype=int32)
```

Great! Now we just have to do some data manipulation by adding the labels as a new feature to a new dataframe. We end up with churn_cluster_df, the same as churn_df but with an unsupervised clustering label.

```
churn_cluster_df = churn_df.copy()
churn_cluster_df['cluster'] = kmeans.labels_
#churn_cluster_df[['dist_0', 'dist_1']] = kmeans.transform(churn_ohe_df.drop(columns=
↪'Churn'))
```

We can calculate a comparison matrix, where for each combination of churn and cluster we count how many records there are.

```
churn_cluster_df.groupby(['Churn','cluster']).size().unstack()
```

```
cluster     0     1
Churn
No       3404  1770
Yes      1547   322
```

Looks like the overlap is not as clear as we would have expect it, this is common in unsupervised techniques as we did not specify the Churn feature to the algorithm. This does not imply our work is useless as it might give other insight to our data.

Same for the regular data we create 2 functions that aggregate our data based on a specific column name.

```
def count_cluster_matrix(col_name):
    return churn_cluster_df.groupby(['cluster', col_name]).size().unstack()

def mean_cluster_matrix(col_name):
    df = churn_cluster_df.groupby(['cluster', col_name]).size().unstack()
    return df.divide(df.sum(axis='index'),axis='columns')
```

As an example we count the occurences of Device Protection with our clusters

```
count_cluster_matrix('DeviceProtection')
```

```
DeviceProtection    No  No internet service   Yes
cluster
0                 2432                 1526   993
1                  663                    0  1429
```

Cluster 1 seems to not contains any users that did not have internet access, so we can already see that this cluster only contains users with internet and mostly have device protection.

To automate results, we again perform the contingency analysis, this time on the cluster feature instead of the churn feature.

```
cl_significant_cols = []
cl_chi2_results = {}
for col in churn_cluster_df.columns.drop('Churn'):
    counts = count_cluster_matrix(col)
    F, p, df, exp = scipy.stats.chi2_contingency(counts)
    if p<0.05:
        cl_significant_cols.append(col)
        cl_chi2_results[col] = {
```

(continues on next page)

(continued from previous page)

```
            'F': F,
            'p': p,
            'real': counts,
            'exp': exp,
            'diff': counts - exp,
        }

# sort in descending F value
cl_chi2_results = {x[0]: x[1] for x in sorted(cl_chi2_results.items(), key=lambda x:␣
 ↪x[1]['F'], reverse=True)}
```

The significant columns can be completely different, yet seem fairly similar

```
cl_significant_cols
```

```
['SeniorCitizen',
 'Partner',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod']
```

We ask for the difference, which only seems to be the PhoneService, this feature is important for the clusters but not the churning.

```
set(cl_significant_cols).difference(significant_cols)
```

```
{'PhoneService'}
```

To get a better picture I opted to print all the significant results in order of correlation strenght. Both for Churn as for cluster.

```
print('Churn significant features')
{col: result['F'] for col, result in chi2_results.items()}
```

```
Churn significant features
```

```
{'Contract': 1184.5965720837926,
 'OnlineSecurity': 849.9989679615962,
 'TechSupport': 828.1970684587393,
 'InternetService': 732.309589667794,
 'PaymentMethod': 648.1423274814,
 'OnlineBackup': 601.8127901134089,
 'DeviceProtection': 558.419369407389,
 'StreamingMovies': 375.6614793452656,
 'StreamingTV': 374.20394331098134,
```

(continues on next page)

(continued from previous page)

```
 'PaperlessBilling': 258.27764906707307,
 'Dependents': 189.12924940423474,
 'SeniorCitizen': 159.42630036838742,
 'Partner': 158.7333820309922,
 'MultipleLines': 11.33044148319756}
```

```
print('Cluster significant features')
{col: result['F'] for col, result in cl_chi2_results.items()}
```

```
Cluster significant features
```

```
{'DeviceProtection': 1742.0880663243113,
 'OnlineBackup': 1665.2730646044615,
 'StreamingTV': 1655.5343860608277,
 'StreamingMovies': 1643.5321794643069,
 'TechSupport': 1391.976678378673,
 'OnlineSecurity': 1333.6884498651216,
 'MultipleLines': 1115.4765363222418,
 'Contract': 1041.6388959111168,
 'InternetService': 998.482344451734,
 'PaymentMethod': 578.5875906673851,
 'Partner': 480.3441523872099,
 'PaperlessBilling': 145.83172959071203,
 'PhoneService': 89.14446552423011,
 'SeniorCitizen': 54.438061283034386,
 'Dependents': 17.631250785385838}
```

I meantioned PhoneService earlier, when we print the difference between truth and expected, we see that a lot more persons that have a phone service are in cluster 1. We already knew cluster 1 has the users with internet service, now it seems users with phone services are also more present in cluster 1. It seems to be filled with customers that have most services…

```
cl_chi2_results['PhoneService']['diff']
```

```
PhoneService           No         Yes
cluster
0              107.576175 -107.576175
1             -107.576175  107.576175
```

Another this that caught my attention is the payment method, cluster 1 uses way more often an automatic payment method. Perhaps these are sleeping customers that have no idea about what they pay.

```
cl_chi2_results['PaymentMethod']['diff']
```

```
PaymentMethod  Bank transfer (automatic)  Credit card (automatic)  \
cluster
0                            -206.381798              -194.916513
1                             206.381798               194.916513

PaymentMethod  Electronic check  Mailed check
cluster
0                     75.481897    325.816413
1                    -75.481897   -325.816413
```

For numerical features we can see that cluster 1 usually has much higher values. This cluster consist of customers that are loyal, pay more per month and therefore also in total.

```
churn_cluster_df.groupby('cluster')[['tenure', 'MonthlyCharges', 'TotalCharges']].
↪mean()
```

```
            tenure  MonthlyCharges  TotalCharges
cluster
0        21.144617       53.591820    977.746748
1        58.940249       91.196702   5361.063360
```

The tenure and total charges reverses in case of grouping per Churn, yet the monthly charges on average are still higher, customers churn early as they have high monthly charges.

```
churn_cluster_df.groupby('Churn')[['tenure', 'MonthlyCharges', 'TotalCharges']].mean()
```

```
          tenure  MonthlyCharges  TotalCharges
Churn
No     37.569965       61.265124   2549.911442
Yes    17.979133       74.441332   1531.796094
```

36.5.5 Nearest Neighbour classification

Our client asked if we could predict future churning, we could solve this with a classification algorithm. I chose for KNN as it is simple and explainable. we start by importing.

```
from sklearn.neighbors import KNeighborsClassifier
```

To classify users between churn and nochurn we create a knn classifier, I opted to go for 5 neighbours so it will look at the 5 most similar users in our dataset and see if they churned.

```
knn = KNeighborsClassifier(n_neighbors=5)
```

We train the algorithm by fitting on the churn data, notice how we both supply input (all columns but churn) and output (only churn column) so the algorithm knows the outcome.

```
knn.fit(churn_ohe_df.drop(columns='Churn'), churn_ohe_df.Churn)
```

```
KNeighborsClassifier()
```

Now that the algorithm is trained, we create a new dataframe that not only contains the truth (Churn) but also the prediction as new feature (predict).

```
churn_predicted_df = churn_df.copy()
churn_predicted_df['predict'] = knn.predict(churn_ohe_df.drop(columns='Churn'))
```

To evaluate the results, we create a confusion matrix, where all 4 combinations are counted.

```
conf_matrix = churn_predicted_df[['Churn', 'predict']].value_counts().unstack()
conf_matrix
```

```
predict     0     1
Churn
No       4778   396
Yes       796  1073
```

Of all churners, (1869) we found 1073, which is not bad, yet us calculate accuracy (amount of flagged users that is actually a churner) and recall (amount of churners that is found by the algorithm).

```
f"accuracy: {(conf_matrix[1]['Yes']/conf_matrix[1].sum()*100).round(2)}%"
```

```
'accuracy: 73.04%'
```

```
f"recall: {(conf_matrix[1]['Yes']/conf_matrix.loc['Yes'].sum()*100).round(2)}%"
```

```
'recall: 57.41%'
```

36.6 Visualisation

Now that we have explored the content of our data, we need to create an appealing visualisation to demonstrate the relations.

36.6.1 Categorical correlation

We deduced earlier that features such as Contract and OnlineSecurity are good predictors for churning, I can think of 2 ways to visualise categorical correlations, heatmaps and stacked bar charts. First again our results, both the contingency result as the mean matrix.

```
chi2_results['Contract']['diff']
```

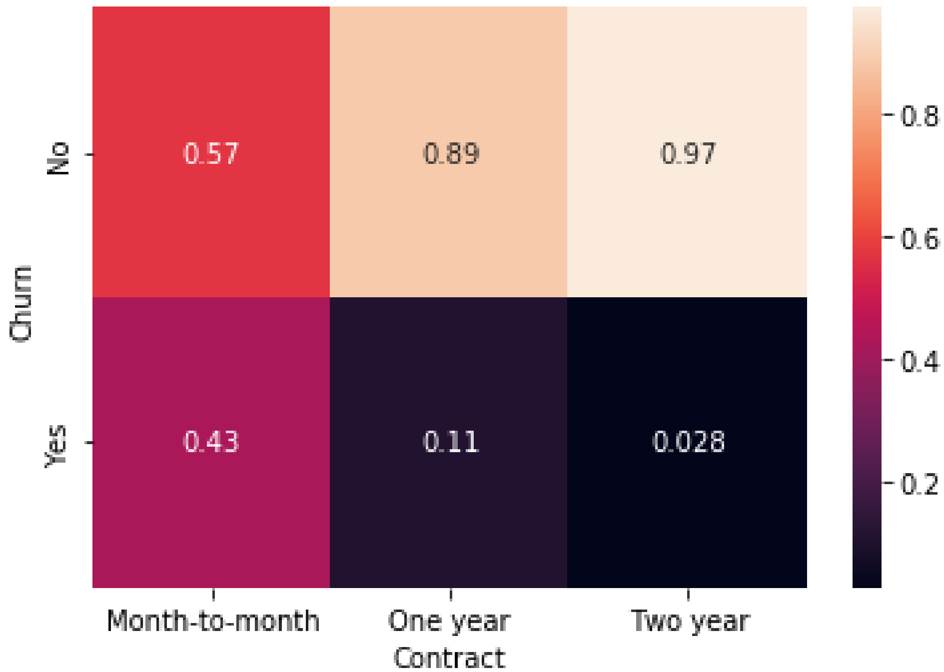
```
Contract  Month-to-month   One year    Two year
Churn
No           -626.691751  224.88982  401.801931
Yes           626.691751 -224.88982 -401.801931
```

```
mean_matrix('Contract')
```

```
Contract  Month-to-month  One year  Two year
Churn
No              0.572903  0.887305  0.971681
Yes             0.427097  0.112695  0.028319
```

What we would like to do now is turn this dataframe into a color coded version, a heatmap. Our Seaborn library makes this very easy and we can even annotate this

```
ax = sns.heatmap(mean_matrix('Contract'), annot=True)
```



This plot shows that for Churn==Yes (lower row) the most of them come from the Month-to-month category, indicating that user who pay month-to-month are more susceptible to churn. We could make a bold claim and say that if ONLY the Contract was the determining factor and not other features, we could save about 30% of the month-to-month group if our services would improve in that category similarly to other groups. Or if we would be able to convert all users in that category to the one-year contract.

```
churn_df.Contract.value_counts()['Month-to-month']*(0.427-0.1126)
```

```
1218.3
```

About 1200 Churners would have been prevented! that is a whole lot! obviously to mention that this is only true if the Contract was the ONLY feature that would make a change.

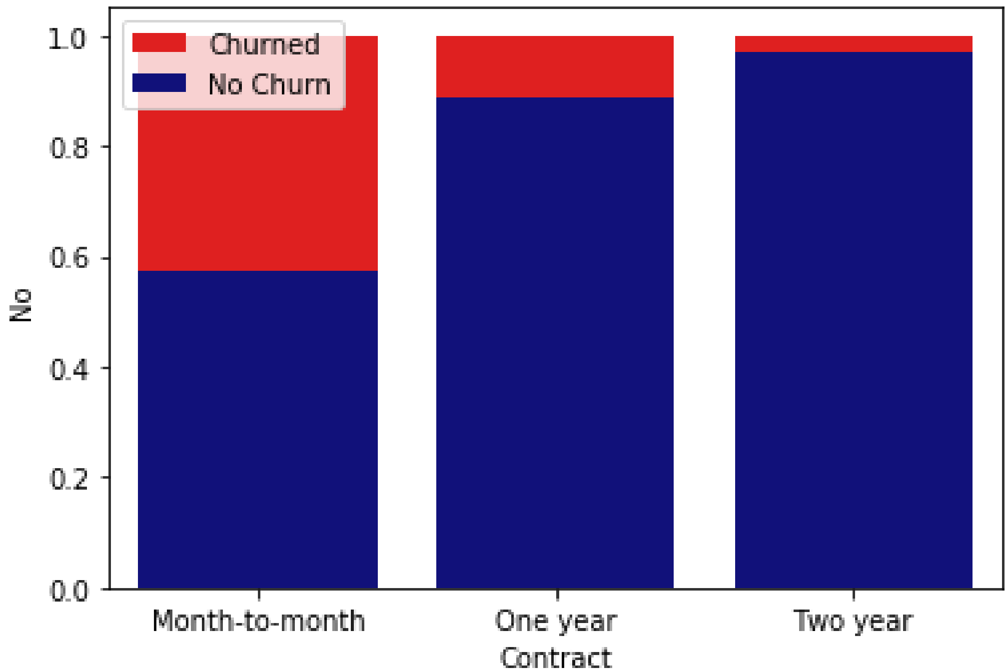
To make a bar plot we first need some more data wrangling, we create the following view so seaborn can create the stacked bar plot.

```
vis_matrix = mean_matrix('Contract').T.reset_index()
vis_matrix['sum'] = 1
vis_matrix
```

```
Churn         Contract        No       Yes  sum
0       Month-to-month  0.572903  0.427097    1
1             One year  0.887305  0.112695    1
2             Two year  0.971681  0.028319    1
```

With this visualisation matrix we have not only no and yes for churn as features, but also the sum. There are other methods to obtain the stacked bar chart but the result is the same.

```
sns.barplot(x="Contract", y="sum", data=vis_matrix, color='red', label='Churned')
sns.barplot(x="Contract",  y="No", data=vis_matrix, color='darkblue', label='No Churn
↪')
plt.legend()
plt.show()
```



I like to think that this graph clearly displays the disparity between different contracts and the relation to Churning, the red portion indications the percentage of churned customers, keep in mind that some categories might not be large so a larger portion of churners is not as detrimental in that case, but as we saw earlier about 1200 churners could have been prevented if the proportions for month-to-month contract would be the same.

We can perform a similar result for online security.

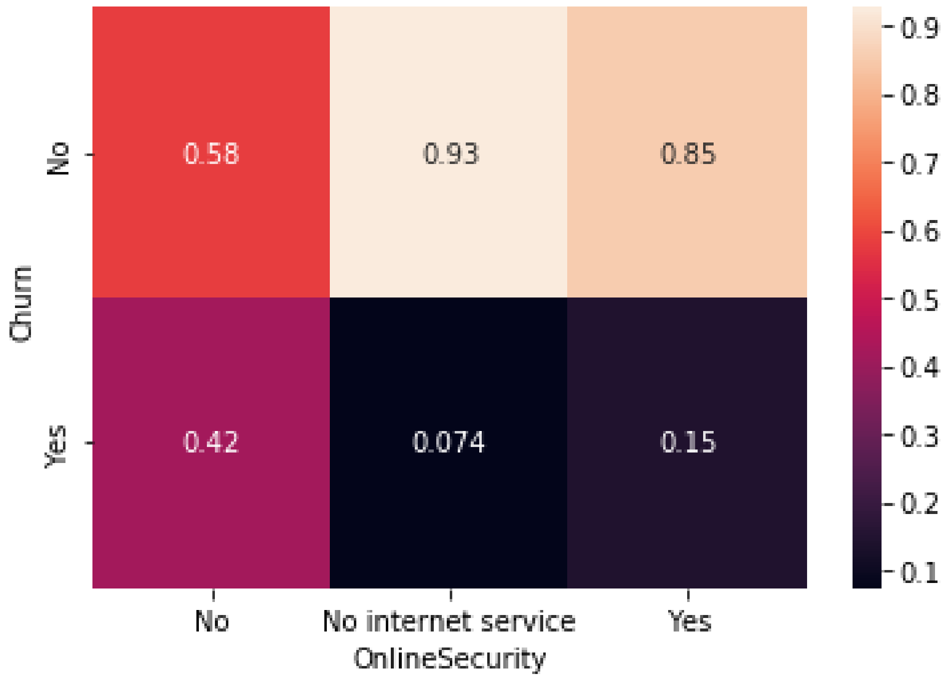
```
chi2_results['OnlineSecurity']['diff']
```

```
OnlineSecurity          No  No internet service         Yes
Churn
No              -532.736192           291.954423  240.781769
Yes              532.736192          -291.954423 -240.781769
```

```
mean_matrix('OnlineSecurity')
```

```
OnlineSecurity        No  No internet service       Yes
Churn
No              0.582333              0.92595  0.853888
Yes             0.417667              0.07405  0.146112
```

```
ax = sns.heatmap(mean_matrix('OnlineSecurity'), annot=True)
```



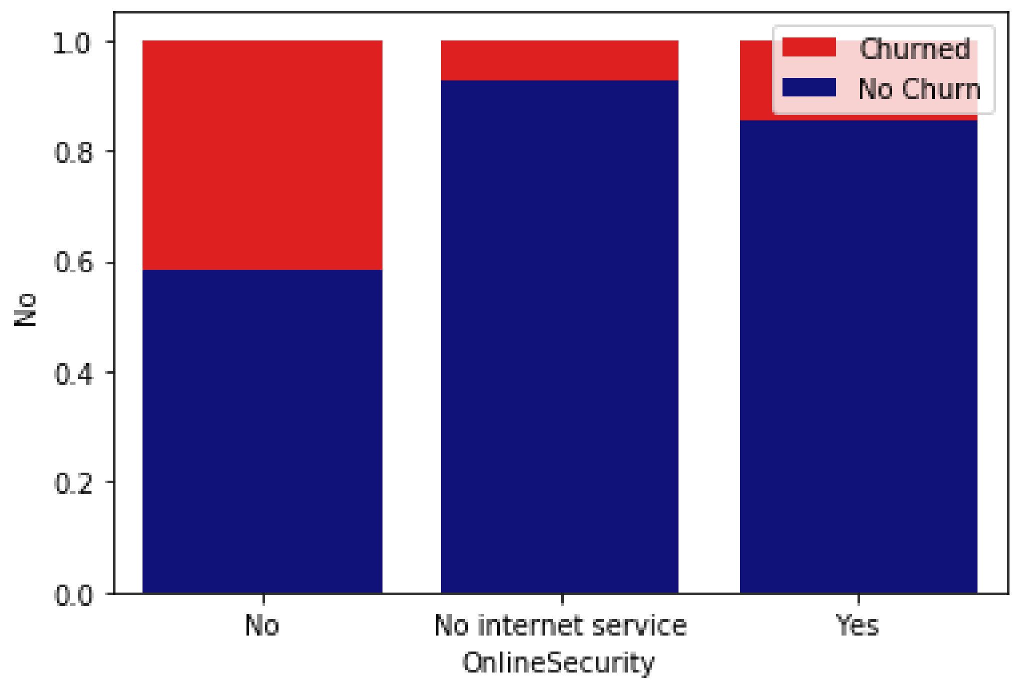

```
churn_df.OnlineSecurity.value_counts()['No']*(0.417-0.146)
```

```
947.9580000000001
```

```
vis_matrix = mean_matrix('OnlineSecurity').T.reset_index()
vis_matrix['sum'] = 1
vis_matrix
```

```
Churn       OnlineSecurity        No       Yes  sum
0                       No  0.582333  0.417667    1
1      No internet service  0.925950  0.074050    1
2                      Yes  0.853888  0.146112    1
```

```
sns.barplot(x="OnlineSecurity", y="sum", data=vis_matrix, color='red', label='Churned
↪')
sns.barplot(x="OnlineSecurity",  y="No", data=vis_matrix, color='darkblue', label='No␣
↪Churn')
plt.legend()
plt.show()
```



Again a big difference in groups, this time we could have saved about 950 churners if we would have convinced users that no online security is a bad idea.

36.6.2 Numerical vs Categorical correlation

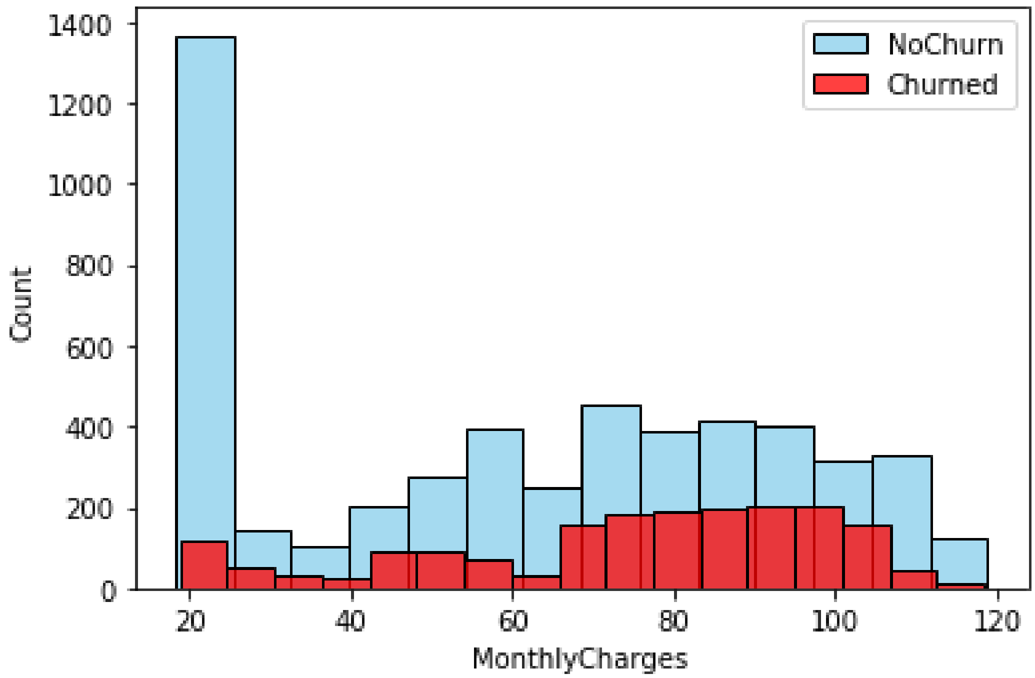
When visualising numerical and categorical correlation it usually comes down to histograms. Here I will look into MonthlyCharges and tenure. For a refreshment we group per churn and print the averages.

```
churn_df.groupby('Churn')[['tenure', 'MonthlyCharges', 'TotalCharges']].mean()
```

```
          tenure  MonthlyCharges  TotalCharges
Churn
No     37.569965       61.265124   2549.911442
Yes    17.979133       74.441332   1531.796094
```

The trick for histograms with different categories is to overlap multiple histograms, we seperate our dataset into churned and nochurn and plot both results.

```
sns.histplot(x=churn_df[churn_df.Churn=='No'].MonthlyCharges, color="skyblue", label=
 ↪"NoChurn")
sns.histplot(x=churn_df[churn_df.Churn=='Yes'].MonthlyCharges, color="red", label=
 ↪"Churned")
plt.legend()
plt.show()
```



For monthly charges we can see that although there was a significant difference found by ANOVA and the means are different, the distributions look alike. The culprit behind this is probably the long peak of no churn in the beginning, the dataset seems to have a lot of small customers that are happy with their services as the price is low. A good example how with non normal data we should not simply rely on mathematics to say something is significant!

Perhaps to overcome non normality we could opt for the median instead of the mean.

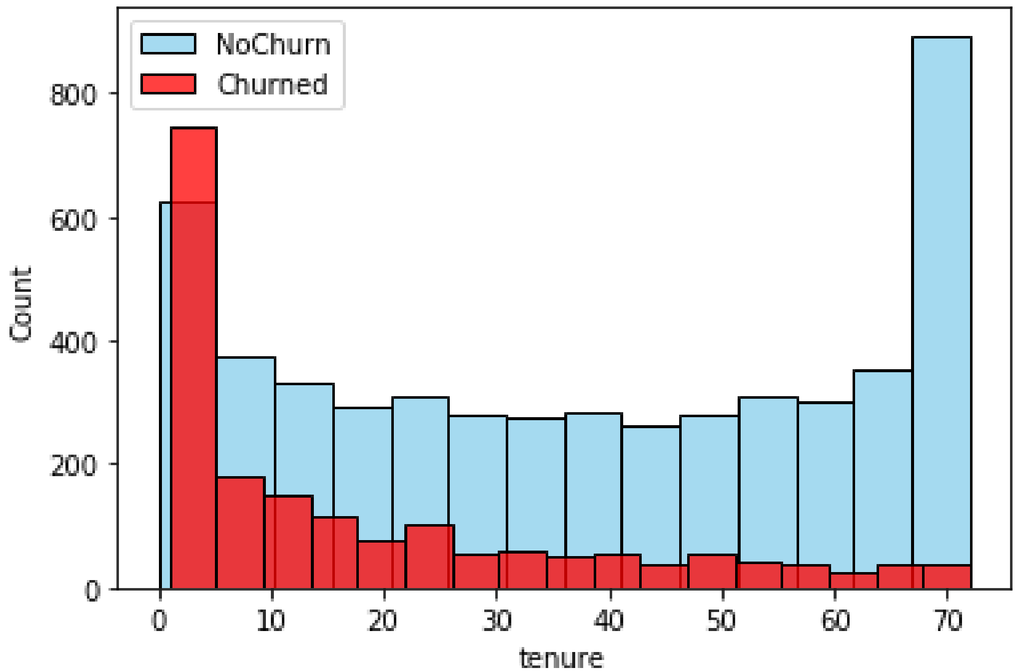
```
churn_df.groupby('Churn')[['tenure', 'MonthlyCharges', 'TotalCharges']].median()
```

```
       tenure  MonthlyCharges  TotalCharges
Churn
No       38.0          64.425      1679.525
Yes      10.0          79.650       703.550
```

Although the values have changed (indicating again non normal data) we see that the difference is still present, so our non normality has not been 'solved'. We are warned.

Similar to the previous plot, we create a histogram for tenure

```
sns.histplot(x=churn_df[churn_df.Churn=='No'].tenure, color="skyblue", label="NoChurn
 ↪")
sns.histplot(x=churn_df[churn_df.Churn=='Yes'].tenure, color="red", label="Churned")
plt.legend()
plt.show()
```

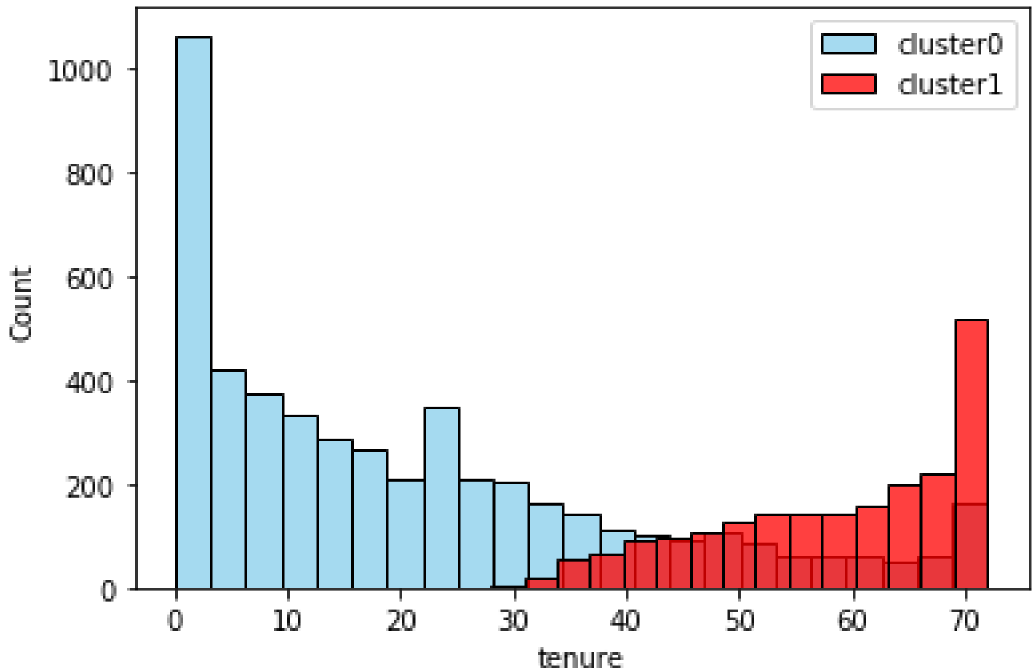


This looks really great! we can see that churned users usually have a lower tenure, perhaps onboarding of new customers is a problem?

36.6.3 Unsupervised clustering

Similar to the churn feature, we can also use the cluster feature, basically the same method, but a different outcome.

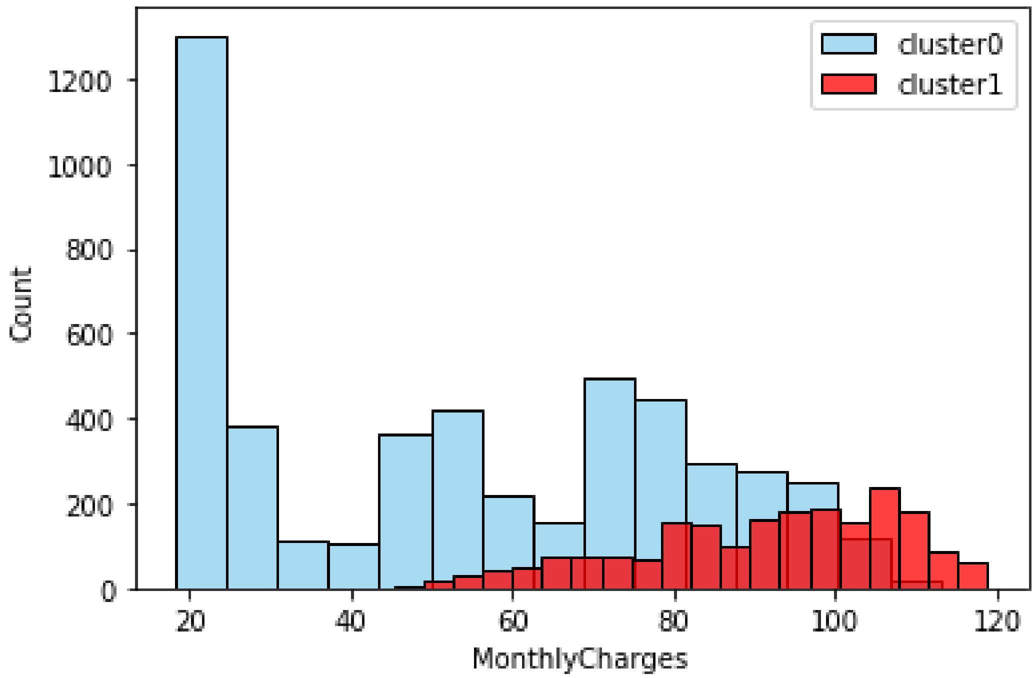
```
sns.histplot(x=churn_cluster_df[churn_cluster_df.cluster==0].tenure, color="skyblue",␣
↪label="cluster0")
sns.histplot(x=churn_cluster_df[churn_cluster_df.cluster==1].tenure, color="red",␣
↪label="cluster1")
plt.legend()
plt.show()
```



Here you can see the power of clustering, the algorithm clearly used the tenure as a input to determine the clusters. cluster 1 contains most of the longer customers (that all have internet and most of them phone service).

In case of montly charges we also see a big difference.

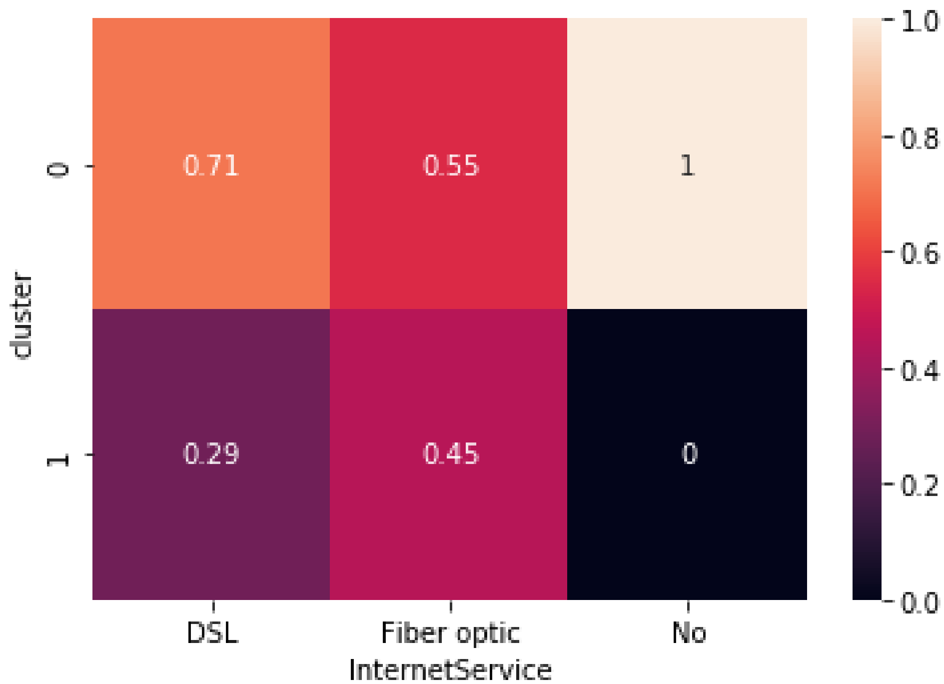
```
sns.histplot(x=churn_cluster_df[churn_cluster_df.cluster==0].MonthlyCharges, color=
 ↪"skyblue", label="cluster0")
sns.histplot(x=churn_cluster_df[churn_cluster_df.cluster==1].MonthlyCharges, color=
 ↪"red", label="cluster1")
plt.legend()
plt.show()
```



cluster 1 again contains the higher paying customers, which is explainable as they mostly all have phone and internet. These customers might be 'sleeping' as they are not aware of higher charges.

To show the phone services I created a simple heatmap.

```
ax = sns.heatmap(mean_cluster_matrix('InternetService'), annot=True)
```



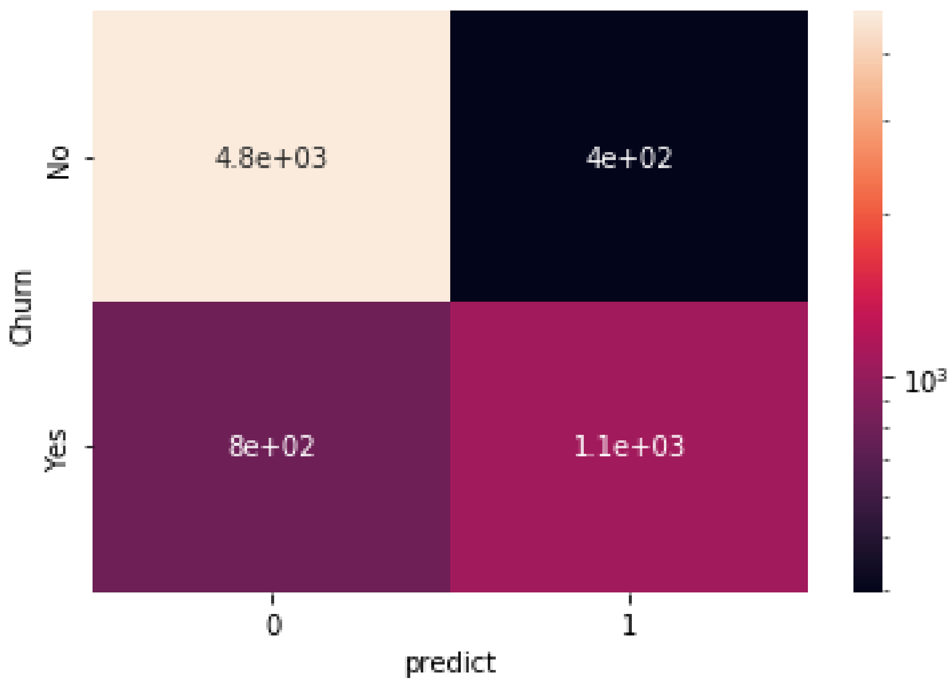
It shows in which cluster the internet users are, again all users that have no internet are in cluster 0.

36.6.4 K Nearest Neighbours

Illustrating a machine learning algorithm is always difficult, a we are dealing with categorical variables it is exceptionally hard.

The only thing I can think of here is to create a heatmap from the confusion matrix, with a logarithmic scale.

```
from matplotlib.colors import LogNorm
ax = sns.heatmap(conf_matrix, annot=True,  norm=LogNorm())
```



Not great, but shows that the false positives (no churnes that are flagged) and false negatives (churners that not have been flagged) are fairly low.

36.6.5 Summary

At this point it would be a good idea to reconnect with our client and discuss our results.

In our analysis we found some significant difference for churners, being:

- A short tenure
- Having a month-to-month contract
- Not having additional options on services
- Senior Citizenship

To prevent this they could for example:

- Give attention to new customers, create a better onboarding
- Create promotion/discount for longer subscription plans
- Create promotion/discount on additional services
- Improve elpdesk for less technology abled persons

When we cluster the customers in 2 groups, we did not find a clear overlap with the churn parameter, however it seems the second cluster found customers that have higher tenures and more additional services. Looking at Charges, this cluster had a significantly higher amount, indicating that the most profitable customers belong to this cluster.

A (simple) machine learning exercise has shown there is a possibility of having a 75% accuracy (amount of flagged users that is actually a churner) and a recall of 57% (amount of churners that is found by the algorithm). These results are not great, but not bad either, further improvements might be needed but this implementation is not critical, i.e. flagging a user as a churner whilst he/she is not, is not necessary crucial for operation.

CHAPTER

THIRTYSEVEN

CASE STUDY: OLYMPIC MEDALS

In this case study we explore the history of medals in the summer and winter olympics

The case study is divided into several parts:

- Goals
- Parsing
- Preparation (cleaning)
- Processing
- Exploration
- Visualization
- Conclusion

37.1 Goals

In this section we define questions that will be our guideline througout the case study

- Which countries are over-/underperforming?
- Are some countries exceptional in some sports?
- Do physical traits have an influence on some sports?

We'll (try to) keep these question in mind when performing the case study.

37.2 Parsing

we start out by importing all necessary libraries

```
import os
import json
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats
import matplotlib.pyplot as plt
from IPython.display import set_matplotlib_formats
%matplotlib inline
```

(continues on next page)

(continued from previous page)

```
#set_matplotlib_formats('svg')
plt.rcParams['figure.figsize'] = [10, 10]
```

in order to download datasets from kaggle, we need an API key to access their API, we'll make that here

```
if not os.path.exists(os.path.expanduser('~/.kaggle')):
    os.mkdir(os.path.expanduser('~/.kaggle'))

with open(os.path.expanduser('~/.kaggle/kaggle.json'), 'w') as f:
    json.dump(
        {
            "username":"lorenzf",
            "key":"7a44a9e99b27e796177d793a3d85b8cf"
        }
        , f)
```

now we can import kaggle too and download the datasets

```
import kaggle
kaggle.api.dataset_download_files(dataset='heesoo37/120-years-of-olympic-history-
↪athletes-and-results', path='./data', unzip=True)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_9005/183238531.py in <module>
----> 1 import kaggle
      2 kaggle.api.dataset_download_files(dataset='heesoo37/120-years-of-olympic-
↪history-athletes-and-results', path='./data', unzip=True)

ModuleNotFoundError: No module named 'kaggle'
```

the csv files are now in the './data' folder, we can now read them using pandas, here is the list of all csv files in our folder

```
os.listdir('./data')
```

```
['WA_Fn-UseC_-Telco-Customer-Churn.csv',
 'API_NY.GDP.PCAP.CD_DS2_en_csv_v2_3358201.csv',
 'noc_regions.csv',
 'freeFormResponses.csv',
 'SurveySchema.csv',
 'jester_ratings.csv',
 'multipleChoiceResponses.csv',
 'one-million-reddit-jokes.csv',
 'jester_items.csv',
 'athlete_events.csv',
 'API_SP.POP.TOTL_DS2_en_csv_v2_3358390.csv']
```

The file of our interest is 'athlete_events.csv', it contains every contestant in every sport since 1896. Let's print out the top 5 events.

```
athlete_events = pd.read_csv('./data/athlete_events.csv')
print('shape: ' + str(athlete_events.shape))
athlete_events.head()
```

```
shape: (271116, 15)
```

```
   ID                      Name Sex   Age  Height  Weight            Team  \
0   1                 A Dijiang   M  24.0   180.0    80.0           China
1   2                  A Lamusi   M  23.0   170.0    60.0           China
2   3       Gunnar Nielsen Aaby   M  24.0     NaN     NaN         Denmark
3   4      Edgar Lindenau Aabye   M  34.0     NaN     NaN  Denmark/Sweden
4   5  Christine Jacoba Aaftink   F  21.0   185.0    82.0     Netherlands

   NOC        Games  Year  Season       City          Sport  \
0  CHN  1992 Summer  1992  Summer  Barcelona     Basketball
1  CHN  2012 Summer  2012  Summer     London           Judo
2  DEN  1920 Summer  1920  Summer  Antwerpen       Football
3  DEN  1900 Summer  1900  Summer      Paris     Tug-Of-War
4  NED  1988 Winter  1988  Winter    Calgary  Speed Skating

                              Event Medal
0       Basketball Men's Basketball   NaN
1      Judo Men's Extra-Lightweight   NaN
2           Football Men's Football   NaN
3        Tug-Of-War Men's Tug-Of-War  Gold
4  Speed Skating Women's 500 metres   NaN
```

Seems we have a name, gender, age, height and weight of the contestant, as wel as the country they represent, the games they attended located in which city. The last 3 columns specify the sport, event within the sport and a possible medal. Presumably the keeping of their score would have been difficult as different sports use different score metrics which would be hard to compare.

```
noc_regions = pd.read_csv('./data/noc_regions.csv')
print('shape: ' + str(noc_regions.shape))
noc_regions.head()
```

```
shape: (230, 3)
```

```
   NOC       region                 notes
0  AFG  Afghanistan                   NaN
1  AHO      Curacao  Netherlands Antilles
2  ALB      Albania                   NaN
3  ALG      Algeria                   NaN
4  AND      Andorra                   NaN
```

37.3 Preparation

here we perform tasks to prepare the data in a more pleasing format.

37.3.1 Data Types

Before we do anything with our data, it is good to see if our data types are in order

```
athlete_events.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271116 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   ID      271116 non-null  int64
 1   Name    271116 non-null  object
 2   Sex     271116 non-null  object
 3   Age     261642 non-null  float64
 4   Height  210945 non-null  float64
 5   Weight  208241 non-null  float64
 6   Team    271116 non-null  object
 7   NOC     271116 non-null  object
 8   Games   271116 non-null  object
 9   Year    271116 non-null  int64
 10  Season  271116 non-null  object
 11  City    271116 non-null  object
 12  Sport   271116 non-null  object
 13  Event   271116 non-null  object
 14  Medal   39783 non-null   object
dtypes: float64(3), int64(2), object(10)
memory usage: 31.0+ MB
```

```
athlete_events[['Sex', 'Team', 'Season', 'City', 'Sport', 'Event']] = athlete_events[[
↪'Sex', 'Team', 'Season', 'City', 'Sport', 'Event']].astype('category')
athlete_events.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271116 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   ID      271116 non-null  int64
 1   Name    271116 non-null  object
 2   Sex     271116 non-null  category
 3   Age     261642 non-null  float64
 4   Height  210945 non-null  float64
 5   Weight  208241 non-null  float64
 6   Team    271116 non-null  category
 7   NOC     271116 non-null  object
 8   Games   271116 non-null  object
 9   Year    271116 non-null  int64
 10  Season  271116 non-null  category
 11  City    271116 non-null  category
 12  Sport   271116 non-null  category
 13  Event   271116 non-null  category
 14  Medal   39783 non-null   object
dtypes: category(6), float64(3), int64(2), object(4)
memory usage: 20.8+ MB
```

37.3.2 Missing values

for each dataframe we apply a few checks in order to see the quality of data

```
print(100*athlete_events.isna().sum()/athlete_events.shape[0])
```

```
ID          0.000000
Name        0.000000
Sex         0.000000
Age         3.494445
Height     22.193821
Weight     23.191180
Team        0.000000
NOC         0.000000
Games       0.000000
Year        0.000000
Season      0.000000
City        0.000000
Sport       0.000000
Event       0.000000
Medal      85.326207
dtype: float64
```

Age, 3.5% missing:

Here we can't do much about it, we could impute using mean or median by looking at other contestants from the same sport/event, however I have a feeling that missing ages might be prevalent in the same sports.

```
athlete_events.groupby('Year')['Age'].apply(lambda x: x.isna().sum()).sort_
↪values(ascending=False).head(25)
```

```
Year
1948    1176
1924    1142
1928     963
1920     845
1900     790
1906     743
1908     649
1956     638
1932     330
1952     277
1904     274
1960     221
1984     216
1936     213
1980     187
1896     163
1912     156
1968     118
1988     110
1972      96
1964      56
1976      52
1992      44
1996       8
1994       2
Name: Age, dtype: int64
```

```
athlete_events.groupby('Sport')['Age'].apply(lambda x: x.isna().sum()).sort_
↪values(ascending=False).head(25)
```

```
Sport
Gymnastics           1179
Athletics            1117
Shooting              821
Fencing               715
Cycling               678
Rowing                526
Swimming              524
Art Competitions      507
Wrestling             491
Football              375
Boxing                318
Sailing               285
Weightlifting         206
Hockey                204
Water Polo            200
Equestrianism         193
Basketball            186
Tennis                124
Diving                121
Archery                80
Alpine Skiing          78
Bobsleigh              72
Modern Pentathlon      53
Rugby                  48
Tug-Of-War             44
Name: Age, dtype: int64
```

Although some sports and years are more problematic, we cannot pinpoint a specific group where ages are missing. Imputing with mean or median would drasticly influence the distribution and standard deviation later on. I opt to leave the missing values as is and drop rows with NaN's when using age in calculations.

Height & Weight, 22 & 23 % missing:

Similar to the Age, yet much more are missing, to a point where dropping would become problematic. Let's see if we can find a hotspot of missing data.

```
athlete_events.groupby('Year')[['Height', 'Weight']].apply(lambda x: x.isna().sum()).
↪sort_values(by='Height', ascending=False).head(25)
```

```
      Height  Weight
Year
1952    7170    7171
1948    6311    6329
1936    6209    6414
1924    4719    5003
1928    4599    4856
1956    3748    3754
1920    3525    3821
1912    3319    3444
1992    3175    3157
1908    2626    2618
1932    2108    2771
1996    1871    1821
```

(continues on next page)

(continued from previous page)

```
1900    1820    1857
1906    1476    1528
1904    1088    1154
1960     961    1048
1988     933     928
1976     876     920
1964     681     708
1984     598     603
1980     588     596
1896     334     331
1972     301     389
1994     187     189
2016     176     223
```

```
athlete_events.groupby('Sport')[['Height', 'Weight']].apply(lambda x: x.isna().sum()).
↪sort_values(by='Height', ascending=False).head(25)
```

```
                      Height  Weight
Sport
Gymnastics              8045    8372
Athletics               5717    6023
Swimming                4045    4391
Shooting                3779    4148
Fencing                 3773    4195
Art Competitions        3519    3523
Cycling                 2883    3029
Rowing                  2675    2662
Alpine Skiing           2435    2479
Football                2098    2212
Wrestling               1808    1849
Equestrianism           1742    1791
Sailing                 1647    1716
Boxing                  1469    1497
Cross Country Skiing    1464    1596
Hockey                  1102    1150
Speed Skating           1090    1192
Water Polo              1058    1122
Weightlifting            929     134
Ice Hockey               905     923
Tennis                   820     854
Bobsleigh                786     846
Diving                   764     828
Basketball               655     858
Figure Skating           631     786
```

Again, no hotspots. For the same reason (distribution) we will not be imputing values, although for machine learning reasons this might be useful to increase the training pool. We will drop the rows with missing values whenever we use the height/weight columns. It would be wise here to inform our audience that conclusions on this data might be skewed by a possible bias - there might be a reason the data is missing - which might in turn cause us to make a wrongful conclusion!

Medal, 85% Missing:

Lastly we see that most are missing the medal, this is obviously that they did not win one. We could boldly assume that since each event has 3 medals, there must be an average of 20 contestants (17/20 = 85%). But this might be deviating over time and sport.

37.3.3 Duplicates

For any reason, our dataset might be containing duplicates that would be counted twice and will introduce a bias we would not want. On the other hand, duplicates can be subjected to interpretation, here we would say that if 2 records share a name, gender, NOC, Games and event, the rows would be identical. This would mean that the person would have performed twice in the same event for the same games under the same flag. The illustration below demonstrates a duplicate.

```
athlete_events[athlete_events.Name == 'Jacques Doucet']
```

```
          ID            Name Sex  Age  Height  Weight         Team  NOC  \
57956  29661  Jacques Doucet   M  NaN     NaN     NaN  Favorite-17  FRA
57957  29661  Jacques Doucet   M  NaN     NaN     NaN   Favorite-1  FRA
57958  29661  Jacques Doucet   M  NaN     NaN     NaN   Favorite-1  FRA

             Games  Year  Season   City    Sport                  Event  \
57956  1900 Summer  1900  Summer  Paris  Sailing     Sailing Mixed Open
57957  1900 Summer  1900  Summer  Paris  Sailing  Sailing Mixed 2-3 Ton
57958  1900 Summer  1900  Summer  Paris  Sailing  Sailing Mixed 2-3 Ton

        Medal
57956     NaN
57957  Silver
57958  Silver
```

We can se that Jacques for some reason is listed twice for the Sailing Mixed 2-3 Ton event. He won silver, but coming in second is no excused to be listed a second time! Perhaps we can find out where things went wrong by investigating in which year the duplicates appear.

```
duplicate_events = athlete_events[athlete_events.duplicated(['Name', 'Sex', 'NOC',
 ↪'Games', 'Event'])]
duplicate_events.groupby(['Year'])['Name'].count()
```

```
Year
1900    110
1908     35
1924    126
1928    347
1932    504
1936    258
1948    100
1968      2
1996      2
1998      3
2002      3
2012      1
Name: Name, dtype: int64
```

Seems most of them happen before 1948, perhaps due to errors in manual entries, it feels safe to delete them.

```
athlete_events = athlete_events.drop_duplicates(['Name', 'Sex', 'NOC', 'Games', 'Event
 ↪'])
```

37.3.4 Indexing

It is more convenient to work with an index, our dataset already contains an id which we can use as index

```
athlete_events = athlete_events.set_index('ID')
athlete_events.head()
```

```
                        Name Sex   Age  Height  Weight            Team  NOC  \
ID
1                  A Dijiang   M  24.0   180.0    80.0           China  CHN
2                   A Lamusi   M  23.0   170.0    60.0           China  CHN
3        Gunnar Nielsen Aaby   M  24.0     NaN     NaN         Denmark  DEN
4       Edgar Lindenau Aabye   M  34.0     NaN     NaN  Denmark/Sweden  DEN
5   Christine Jacoba Aaftink   F  21.0   185.0    82.0     Netherlands  NED

          Games  Year  Season       City          Sport  \
ID
1   1992 Summer  1992  Summer  Barcelona     Basketball
2   2012 Summer  2012  Summer     London           Judo
3   1920 Summer  1920  Summer  Antwerpen       Football
4   1900 Summer  1900  Summer      Paris     Tug-Of-War
5   1988 Winter  1988  Winter    Calgary  Speed Skating

                               Event Medal
ID
1        Basketball Men's Basketball   NaN
2       Judo Men's Extra-Lightweight   NaN
3            Football Men's Football   NaN
4        Tug-Of-War Men's Tug-Of-War  Gold
5   Speed Skating Women's 500 metres   NaN
```

37.4 Processing

37.4.1 Medals per country per sport

To find out which country (NOC) performs the best, we would like to have a dataframe with 3 columns ['Gold', 'Silver', 'Bronze'] containing the count of each, as row index, we would have the games and the NOC, thus a multiindex. An important detail is that team sports are given multiple medals, as indicated by the exampe below. Be careful as bias might not always as visible.

```
athlete_events[(athlete_events.Event == "Basketball Men's Basketball")&(athlete_
↪events.Games=='1992 Summer')&(athlete_events.Medal=='Gold')]
```

```
                                   Name Sex   Age  Height  Weight  \
ID
7901                Charles Wade Barkley   M  29.0   198.0   114.0
11668                     Larry Joe Bird   M  35.0   205.0   100.0
30009                Clyde Austin Drexler   M  30.0   200.0   101.0
33553               Patrick Aloysius Ewing   M  29.0   213.0   109.0
55424          Earvin "Magic" Johnson, Jr.   M  32.0   205.0   100.0
55881               Michael Jeffrey Jordan   M  29.0   198.0    90.0
65809            Christian Donald Laettner   M  22.0   211.0   107.0
74176                          Karl Malone   M  29.0   205.0   116.0
```

(continues on next page)

(continued from previous page)

```
83179   Christopher Paul "Chris" Mullin   M  28.0   200.0    98.0
95105            Scottie Maurice Pippen   M  26.0   200.0   102.0
101428           David Maurice Robinson   M  26.0   216.0   107.0
115325            John Houston Stockton   M  30.0   185.0    79.0

                 Team  NOC        Games  Year  Season       City       Sport  \
ID
7901    United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
11668   United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
30009   United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
33553   United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
55424   United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
55881   United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
65809   United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
74176   United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
83179   United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
95105   United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
101428  United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball
115325  United States  USA  1992 Summer  1992  Summer  Barcelona  Basketball

                              Event Medal
ID
7901    Basketball Men's Basketball  Gold
11668   Basketball Men's Basketball  Gold
30009   Basketball Men's Basketball  Gold
33553   Basketball Men's Basketball  Gold
55424   Basketball Men's Basketball  Gold
55881   Basketball Men's Basketball  Gold
65809   Basketball Men's Basketball  Gold
74176   Basketball Men's Basketball  Gold
83179   Basketball Men's Basketball  Gold
95105   Basketball Men's Basketball  Gold
101428  Basketball Men's Basketball  Gold
115325  Basketball Men's Basketball  Gold
```

The preprocessing for this dataframe seem complex but is combination of several operations:

- drop all records with no medals
- drop duplicates based on 'Games', 'NOC' , 'Event', 'Medal' to correct for team sports
- group per 'Games', 'NOC' , 'Medal'
- aggregate groups by calculating their size

At this point, we have a single column containing the amount of medals and 3 indices: 'Games' , 'NOC' and 'Medal'

- unstack the 'Medal' column to obtain 3 columns 'Gold', 'Silver', 'Bronze'
- make sure the order of columns is 'Gold', 'Silver', 'Bronze'
- drop rows where no medals are won, as we do not need those rows

This operation looks like the following:

```
medals_country_df = athlete_events.dropna(subset=['Medal']).drop_duplicates(['Games',
↪'NOC', 'Event']).groupby(['Games', 'NOC', 'Medal', 'Sport']).size().unstack('Medal
↪')[['Gold', 'Silver', 'Bronze']]#.dropna(how='all')#.fillna(0)
medals_country_df = medals_country_df[medals_country_df.sum(axis='columns')>0]
medals_country_df
```

```
Medal                      Gold  Silver  Bronze
Games        NOC Sport
1896 Summer AUS Athletics     2       0       0
                Tennis        0       0       1
            AUT Cycling       1       0       2
                Swimming      1       1       0
            DEN Fencing       0       0       1
...                         ...     ...     ...
2016 Summer UZB Wrestling     0       0       3
            VEN Athletics     0       1       0
                Boxing        0       0       1
                Cycling       0       0       1
            VIE Shooting      1       1       0

[6915 rows x 3 columns]
```

37.4.2 average statistics per year, country and sport

```
avg_stats_df = athlete_events.groupby(['Sex', 'NOC', 'Games', 'Sport'])[['Age',
 ↪'Height', 'Weight']].mean().dropna()
avg_stats_df
```

```
                                 Age  Height  Weight
Sex NOC Games       Sport
F   AFG 2004 Summer Athletics   18.0   180.0    56.0
                    Judo        18.0   165.0    70.0
        2008 Summer Athletics   22.0   180.0    56.0
        2012 Summer Athletics   23.0   160.0    52.0
        2016 Summer Athletics   20.0   165.0    55.0
...                              ...     ...     ...
M   ZIM 2016 Summer Archery     37.0   186.0    78.0
                    Athletics   29.6   167.6    63.2
                    Rowing      27.0   191.0    87.0
                    Shooting    42.0   182.0    80.0
                    Swimming    22.0   181.0    84.0

[31329 rows x 3 columns]
```

37.5 Exploration

At first we would like to know which countries are performing well, we could simply do a sum of all medals for each country as shown below

```
medals_agg_df = medals_country_df.groupby('NOC').sum().sort_values(by='Gold',␣
 ↪ascending=False)
medals_agg_df.head(20)
```

```
Medal  Gold  Silver  Bronze
NOC
USA     871     653     585
URS     405     298     285
GER     287     308     315
```

(continues on next page)

(continued from previous page)

```
GBR     239     279     272
ITA     234     209     229
FRA     224     244     299
CHN     208     152     140
RUS     181     167     186
SWE     170     194     219
GDR     165     128     134
HUN     161     152     168
NOR     148     126     124
AUS     142     163     184
JPN     140     135     159
FIN     121     123     157
CAN     118     153     182
NED     113     119     135
KOR     106      95      88
SUI      95     110     105
ROU      81      90     114
```

As expected, USA leads the charts, interestingly although disbanded over 30 years ago, the soviet are still second in amount of medals, this leads me to several questions:

- does every country have the same resources?
- are some sports easier to obtain medals?
- is the type of medal important?

To create a simple answer on the last one, we could for each country calculate the percentage of gold/silver/bronze medals they obtained, meaning that not the amount but the ratio is important.

```
medals_perc_df = medals_agg_df[medals_agg_df.sum(axis='columns')>20].apply(lambda x:
↪x/x.sum(), axis='columns').sort_values(by='Gold', ascending=False)
medals_perc_df.head(20)
```

```
Medal       Gold     Silver     Bronze
NOC
ETH    0.476190  0.142857  0.380952
CHN    0.416000  0.304000  0.280000
USA    0.412992  0.309625  0.277383
URS    0.409919  0.301619  0.288462
TUR    0.402174  0.293478  0.304348
EUN    0.387931  0.310345  0.301724
NZL    0.387931  0.232759  0.379310
GDR    0.386417  0.299766  0.313817
NOR    0.371859  0.316583  0.311558
KOR    0.366782  0.328720  0.304498
CUB    0.363208  0.306604  0.330189
ITA    0.348214  0.311012  0.340774
IND    0.344828  0.241379  0.413793
SVK    0.343750  0.406250  0.250000
CRO    0.340909  0.363636  0.295455
RUS    0.338951  0.312734  0.348315
HUN    0.334719  0.316008  0.349272
KEN    0.329114  0.367089  0.303797
JPN    0.322581  0.311060  0.366359
GER    0.315385  0.338462  0.346154
```

```
medals_agg_df.loc['ETH']
```

```
Medal
Gold      20
Silver     6
Bronze    16
Name: ETH, dtype: int64
```

Out of nowhere Ethiopia seems to be the highest achiever when it comes to gold medals, but this might be an anomaly as their total medal count is rather low, but still impressive! Also China steps up showing that they don't take second best.

I also mentioned resources, some countries are not as big as USA an China and therefore send less athletes. We could have checked for the amount of athlete's yet opted to go for each countries population. If a country has a bigger population it means it has a bigger pool of genetically favored persons for a sport.

To investigate this I searched for a dataset containing the data, coming from the worldbank API, in the next section we download the data.

```
from io import BytesIO
from zipfile import ZipFile
from urllib.request import urlopen
```

```
resp = urlopen("https://api.worldbank.org/v2/en/indicator/SP.POP.TOTL?
↪downloadformat=csv")
zipfile = ZipFile(BytesIO(resp.read()))
print(f"found files: {zipfile.namelist()}")
```

```
found files: ['Metadata_Indicator_API_SP.POP.TOTL_DS2_en_csv_v2_3358390.csv', 'API_SP.
↪POP.TOTL_DS2_en_csv_v2_3358390.csv', 'Metadata_Country_API_SP.POP.TOTL_DS2_en_csv_
↪v2_3358390.csv']
```

```
file_name = 'API_SP.POP.TOTL_DS2_en_csv_v2_3358390.csv'
zipfile.extract(file_name, './data')
pop_df = pd.read_csv('./data/'+file_name, encoding='', skiprows=4)
pop_df.head()
```

```
                  Country Name Country Code     Indicator Name Indicator Code  \
0                        Aruba          ABW  Population, total    SP.POP.TOTL
1  Africa Eastern and Southern          AFE  Population, total    SP.POP.TOTL
2                  Afghanistan          AFG  Population, total    SP.POP.TOTL
3   Africa Western and Central          AFW  Population, total    SP.POP.TOTL
4                       Angola          AGO  Population, total    SP.POP.TOTL

          1960         1961         1962         1963         1964  \
0      54208.0      55434.0      56234.0      56699.0      57029.0
1  130836765.0  134159786.0  137614644.0  141202036.0  144920186.0
2    8996967.0    9169406.0    9351442.0    9543200.0    9744772.0
3   96396419.0   98407221.0  100506960.0  102691339.0  104953470.0
4    5454938.0    5531451.0    5608499.0    5679409.0    5734995.0

          1965  ...         2012         2013         2014         2015  \
0      57357.0  ...     102565.0     103165.0     103776.0     104339.0
1  148769974.0  ...  547482863.0  562601578.0  578075373.0  593871847.0
2    9956318.0  ...   31161378.0   32269592.0   33370804.0   34413603.0
3  107289875.0  ...  370243017.0  380437896.0  390882979.0  401586651.0
4    5770573.0  ...   25107925.0   26015786.0   26941773.0   27884380.0
```

(continues on next page)

(continued from previous page)

```
          2016          2017          2018          2019          2020  \
0     104865.0      105361.0      105846.0      106310.0      106766.0
1  609978946.0   626392880.0   643090131.0   660046272.0   677243299.0
2   35383028.0    36296111.0    37171922.0    38041757.0    38928341.0
3  412551299.0   423769930.0   435229381.0   446911598.0   458803476.0
4   28842482.0    29816769.0    30809787.0    31825299.0    32866268.0

   Unnamed: 65
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN

[5 rows x 66 columns]
```

You can see that for each year from 1960 the population for each country is given, we first have to stack/unpivot the data to obtain a view that is useful for our purpose.

```
pop_df = pop_df.drop(columns=['Country Name', 'Indicator Name', 'Indicator Code'] +
 ↪pop_df.columns[pop_df.columns.str.contains('Unnamed')].tolist()).set_index('Country
 ↪Code').stack()
pop_df = pop_df.rename('population')
pop_df.head(5)
```

```
Country Code
ABW           1960    54208.0
              1961    55434.0
              1962    56234.0
              1963    56699.0
              1964    57029.0
Name: population, dtype: float64
```

Now we have to match this with our medals dataset we created earlier

```
medals_country_df.head()
```

```
Medal                       Gold  Silver  Bronze
Games        NOC Sport
1896 Summer AUS Athletics      2       0       0
                Tennis         0       0       1
            AUT Cycling        1       0       2
                Swimming       1       1       0
            DEN Fencing        0       0       1
```

There seems to be a problem, our medals dataset does not indicate the year, we can solve this by adding a column

```
medals_country_df['year'] = medals_country_df.index.get_level_values('Games').str[:4]
medals_country_df.head()
```

```
Medal                       Gold  Silver  Bronze  year
Games        NOC Sport
1896 Summer AUS Athletics      2       0       0  1896
                Tennis         0       0       1  1896
```

(continues on next page)

(continued from previous page)

```
            AUT Cycling       1       0       2  1896
                Swimming      1       1       0  1896
            DEN Fencing       0       0       1  1896
```

Great! now we can merge the population data with our medals data

```
medals_country_pop_df = pd.merge(medals_country_df, pop_df, left_on=[medals_country_
↪df.index.get_level_values('NOC'), 'year'], right_index=True, how='left')
medals_country_pop_df
```

```
                           Gold  Silver  Bronze  year  population
Games       NOC Sport
1896 Summer AUS Athletics     2       0       0  1896         NaN
                Tennis        0       0       1  1896         NaN
            AUT Cycling       1       0       2  1896         NaN
                Swimming      1       1       0  1896         NaN
            DEN Fencing       0       0       1  1896         NaN
...                         ...     ...     ...   ...         ...
2016 Summer UZB Wrestling     0       0       3  2016  31847900.0
            VEN Athletics     0       1       0  2016  29851249.0
                Boxing        0       0       1  2016  29851249.0
                Cycling       0       0       1  2016  29851249.0
            VIE Shooting      1       1       0  2016         NaN

[6915 rows x 5 columns]
```

As our population data only contained data from 1960 onwards, we need to discard some of our rows, we do this with the dropna method

```
medals_country_pop_df = medals_country_pop_df.dropna()
medals_country_pop_df
```

```
                               Gold  Silver  Bronze  year  population
Games       NOC Sport
1960 Summer ARG Boxing            0       0       1  1960  20481781.0
                Sailing           0       1       0  1960  20481781.0
            AUS Athletics         1       2       1  1960  10276477.0
                Boxing            0       0       2  1960  10276477.0
                Equestrianism     1       1       0  1960  10276477.0
...                             ...     ...     ...   ...         ...
2016 Summer UZB Weightlifting     1       0       0  2016  31847900.0
                Wrestling         0       0       3  2016  31847900.0
            VEN Athletics         0       1       0  2016  29851249.0
                Boxing            0       0       1  2016  29851249.0
                Cycling           0       0       1  2016  29851249.0

[3710 rows x 5 columns]
```

In order to use our population information, we need to be creative, I decided to keep things simple and for each type of medal divide the amount with the population, therefore the value is changed from:

- the amount of medals earned for a country

to

- the amount of medals earned per person for a country

Which will be much lower for countries with a higher population

```
medals_pop_df = medals_country_pop_df[['Gold', 'Silver', 'Bronze']].div(medals_
↪country_pop_df.population,axis='index')
medals_pop_df
```

```
                                      Gold        Silver        Bronze
Games       NOC Sport
1960 Summer ARG Boxing        0.000000e+00  0.000000e+00  4.882388e-08
                Sailing       0.000000e+00  4.882388e-08  0.000000e+00
            AUS Athletics     9.730961e-08  1.946192e-07  9.730961e-08
                Boxing        0.000000e+00  0.000000e+00  1.946192e-07
                Equestrianism 9.730961e-08  9.730961e-08  0.000000e+00
...                                    ...           ...           ...
2016 Summer UZB Weightlifting 3.139924e-08  0.000000e+00  0.000000e+00
                Wrestling     0.000000e+00  0.000000e+00  9.419773e-08
            VEN Athletics     0.000000e+00  3.349944e-08  0.000000e+00
                Boxing        0.000000e+00  0.000000e+00  3.349944e-08
                Cycling       0.000000e+00  0.000000e+00  3.349944e-08

[3710 rows x 3 columns]
```

You can see that these values are much lower as populations are very high. Now we can do exactly the same as before and sort per highest total amount.

```
medals_pop_df.groupby('NOC').sum().sort_values(by='Gold', ascending=False).head(20)
```

```
         Gold    Silver    Bronze
NOC
LIE  0.000077  0.000077  0.000193
NOR  0.000022  0.000020  0.000018
NZL  0.000011  0.000007  0.000011
FIN  0.000010  0.000012  0.000014
HUN  0.000010  0.000010  0.000010
SWE  0.000008  0.000010  0.000010
CUB  0.000007  0.000006  0.000006
CHI  0.000007  0.000014  0.000013
AUS  0.000007  0.000008  0.000009
JAM  0.000005  0.000010  0.000007
AUT  0.000005  0.000009  0.000009
EST  0.000004  0.000004  0.000006
ROU  0.000004  0.000004  0.000005
CAN  0.000003  0.000004  0.000004
ITA  0.000003  0.000002  0.000003
SUR  0.000003  0.000000  0.000002
TTO  0.000002  0.000003  0.000008
BRN  0.000002  0.000002  0.000003
KOR  0.000002  0.000002  0.000002
USA  0.000002  0.000002  0.000002
```

Our data is now completely different, for the reason that Liechtenstein is very small it scores very high. You could argue that being small is an advantage here, yet it also means you have less chance to have highly athletic persons. Just to make sure that they did not by accident get a gold medal let's get all of their medals.

```
athlete_events[(athlete_events.NOC=='LIE') & ~(athlete_events.Medal.isna())]
```

```
                                        Name Sex   Age  Height  Weight  \
ID
```

(continues on next page)

(continued from previous page)

```
37329                          Paul Frommelt   M  30.0   178.0    70.0
37330                         Willi Frommelt   M  23.0   180.0    78.0
62609                   Ursula Konzett (-Gregg)   F  24.0   164.0     NaN
129663                    Andreas "Andi" Wenzel   M  21.0   175.0    70.0
129663                    Andreas "Andi" Wenzel   M  25.0   175.0    70.0
129665  Hannelore "Hanni" Wenzel (-Weirather)   F  19.0   165.0    57.0
129665  Hannelore "Hanni" Wenzel (-Weirather)   F  23.0   165.0    57.0
129665  Hannelore "Hanni" Wenzel (-Weirather)   F  23.0   165.0    57.0
129665  Hannelore "Hanni" Wenzel (-Weirather)   F  23.0   165.0    57.0

                 Team  NOC        Games  Year  Season         City  \
ID
37329   Liechtenstein  LIE  1988 Winter  1988  Winter      Calgary
37330   Liechtenstein  LIE  1976 Winter  1976  Winter    Innsbruck
62609   Liechtenstein  LIE  1984 Winter  1984  Winter     Sarajevo
129663  Liechtenstein  LIE  1980 Winter  1980  Winter  Lake Placid
129663  Liechtenstein  LIE  1984 Winter  1984  Winter     Sarajevo
129665  Liechtenstein  LIE  1976 Winter  1976  Winter    Innsbruck
129665  Liechtenstein  LIE  1980 Winter  1980  Winter  Lake Placid
129665  Liechtenstein  LIE  1980 Winter  1980  Winter  Lake Placid
129665  Liechtenstein  LIE  1980 Winter  1980  Winter  Lake Placid

                Sport                             Event   Medal
ID
37329   Alpine Skiing          Alpine Skiing Men's Slalom  Bronze
37330   Alpine Skiing          Alpine Skiing Men's Slalom  Bronze
62609   Alpine Skiing        Alpine Skiing Women's Slalom  Bronze
129663  Alpine Skiing    Alpine Skiing Men's Giant Slalom  Silver
129663  Alpine Skiing    Alpine Skiing Men's Giant Slalom  Bronze
129665  Alpine Skiing        Alpine Skiing Women's Slalom  Bronze
129665  Alpine Skiing      Alpine Skiing Women's Downhill  Silver
129665  Alpine Skiing  Alpine Skiing Women's Giant Slalom    Gold
129665  Alpine Skiing        Alpine Skiing Women's Slalom    Gold
```

In my opinion this looks about right, 2 gold medals, 2 silver and 5 bronze is impressive for a country with less than 40k inhabitants.

Also a lot of scandinavian countries seem to have taken the lead, this might be indicating that there is less competition in winter sports as they are known to excel there.

Most remarkable is the fall of the USA, which falls to the 20th place, indicating that if we correct for the amount of persons in the country it does not perform that well.

In a same method we could also account for the Gross Domestic Product per Capita, indicating the wealth of a country, again we download data from worldbank

```
resp = urlopen("https://api.worldbank.org/v2/en/indicator/NY.GDP.PCAP.CD?
↪downloadformat=csv")
zipfile = ZipFile(BytesIO(resp.read()))
print(f"found files: {zipfile.namelist()}")
```

```
found files: ['Metadata_Indicator_API_NY.GDP.PCAP.CD_DS2_en_csv_v2_3358201.csv', 'API_
↪NY.GDP.PCAP.CD_DS2_en_csv_v2_3358201.csv', 'Metadata_Country_API_NY.GDP.PCAP.CD_DS2_
↪en_csv_v2_3358201.csv']
```

```
file_name = 'API_NY.GDP.PCAP.CD_DS2_en_csv_v2_3358201.csv'
zipfile.extract(file_name, './data')
```

(continues on next page)

(continued from previous page)

```
gdp_cap_df = pd.read_csv('./data/'+file_name, encoding='', skiprows=4)
gdp_cap_df.head()
```

```
                 Country Name Country Code                Indicator Name  \
0                       Aruba          ABW  GDP per capita (current US$)
1  Africa Eastern and Southern          AFE  GDP per capita (current US$)
2                 Afghanistan          AFG  GDP per capita (current US$)
3   Africa Western and Central          AFW  GDP per capita (current US$)
4                      Angola          AGO  GDP per capita (current US$)

   Indicator Code        1960        1961        1962        1963        1964  \
0  NY.GDP.PCAP.CD         NaN         NaN         NaN         NaN         NaN
1  NY.GDP.PCAP.CD  147.836769  147.238537  156.426780  182.521139  162.594548
2  NY.GDP.PCAP.CD   59.773234   59.860900   58.458009   78.706429   82.095307
3  NY.GDP.PCAP.CD  107.963779  113.114697  118.865837  123.478967  131.892939
4  NY.GDP.PCAP.CD         NaN         NaN         NaN         NaN         NaN

         1965  ...          2012          2013          2014          2015  \
0         NaN  ...  24712.493263  26441.619936  26893.011506  28396.908423
1  180.489043  ...   1672.363658   1653.188436   1658.650062   1507.800256
2  101.108325  ...    641.871438    637.165464    613.856505    578.466353
3  138.566819  ...   1936.390962   2123.392433   2166.743309   1886.248158
4         NaN  ...   5100.097027   5254.881126   5408.411700   4166.979833

           2016          2017          2018         2019         2020  \
0  28452.170615  29350.805019  30253.279358          NaN          NaN
1   1404.953164   1540.232473   1534.171767  1485.307425  1330.140232
2    509.220100    519.888913    493.756581   507.103392   508.808409
3   1666.422406   1606.978332   1695.959215  1772.339155  1714.426800
4   3506.073128   4095.810057   3289.643995  2809.626088  1895.770869

   Unnamed: 65
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN

[5 rows x 66 columns]
```

```
gdp_cap_df = gdp_cap_df.drop(columns=['Country Name', 'Indicator Name', 'Indicator␣
↪Code'] + gdp_cap_df.columns[gdp_cap_df.columns.str.contains('Unnamed')].tolist()).
↪set_index('Country Code').stack()
gdp_cap_df = gdp_cap_df.rename('gdp')
gdp_cap_df.head(5)
```

```
Country Code
ABW           1986     6472.398709
              1987     7885.158927
              1988     9765.909207
              1989    11392.269150
              1990    12306.717679
Name: gdp, dtype: float64
```

Again data from 1960 untill recent that we can use, we merge this with our original medals data.

```
medals_country_gdp_df = pd.merge(medals_country_df, gdp_cap_df, left_on=[medals_
↪country_df.index.get_level_values('NOC'), 'year'], right_index=True, how='left').
↪dropna()
medals_country_gdp_df
```

```
                                  Gold  Silver  Bronze  year           gdp
Games       NOC Sport
1960 Summer AUS Athletics            1       2       1  1960   1807.785710
                Boxing               0       0       2  1960   1807.785710
                Equestrianism        1       1       0  1960   1807.785710
                Swimming             4       4       3  1960   1807.785710
            AUT Rowing               0       1       0  1960    935.460427
...                                ...     ...     ...   ...           ...
2016 Summer USA Wrestling            2       0       1  2016  58021.400500
            UZB Boxing               3       2       2  2016   2567.799207
                Judo                 0       0       2  2016   2567.799207
                Weightlifting        1       0       0  2016   2567.799207
                Wrestling            0       0       3  2016   2567.799207

[3459 rows x 5 columns]
```

And again we recompute our metric, by dividing the amount of medals by the GDP, indicating not how many medals but how many medals per dollar of weight per person obtained

```
medals_country_gdp_df = medals_country_gdp_df[['Gold', 'Silver', 'Bronze']].
↪div(medals_country_gdp_df.gdp,axis='index')
medals_country_gdp_df
```

```
                                      Gold    Silver    Bronze
Games       NOC Sport
1960 Summer AUS Athletics         0.000553  0.001106  0.000553
                Boxing            0.000000  0.000000  0.001106
                Equestrianism     0.000553  0.000553  0.000000
                Swimming          0.002213  0.002213  0.001659
            AUT Rowing            0.000000  0.001069  0.000000
...                                    ...       ...       ...
2016 Summer USA Wrestling         0.000034  0.000000  0.000017
            UZB Boxing            0.001168  0.000779  0.000779
                Judo              0.000000  0.000000  0.000779
                Weightlifting     0.000389  0.000000  0.000000
                Wrestling         0.000000  0.000000  0.001168

[3459 rows x 3 columns]
```

In order to compare we calulcate again the total medal/wealth metric for each country

```
medals_country_gdp_df.groupby('NOC').sum().sort_values(by='Gold', ascending=False).
↪head(20)
```

```
         Gold    Silver    Bronze
NOC
CHN  0.190328  0.182633  0.153561
ETH  0.082480  0.019586  0.040711
KEN  0.064936  0.082503  0.075461
RUS  0.048552  0.040582  0.041834
USA  0.047245  0.033450  0.033275
```

(continues on next page)

(continued from previous page)

```
JPN  0.041464  0.025717  0.034149
ITA  0.035705  0.032462  0.040074
CUB  0.030598  0.021835  0.027075
TUR  0.026663  0.016649  0.008150
UKR  0.026470  0.027748  0.044867
PAK  0.021715  0.016054  0.022192
ROU  0.017914  0.020278  0.021639
IND  0.013408  0.015722  0.028250
GBR  0.013026  0.020645  0.021438
AUS  0.012907  0.013433  0.015555
KOR  0.011429  0.034608  0.027044
UZB  0.010776  0.008233  0.015982
FRA  0.010533  0.015412  0.017983
FIN  0.009890  0.012106  0.013962
NOR  0.009418  0.009189  0.006486
```

As expected China performs well, but also Etheopia again scores high together with Kenia, I'm assuming a lot of runners come from this region. Remarkable is that countries such as USA and Japan, which are known to have a high GDP are still performing outstanding.

Now that we have 3 versions of the same analysis it debatable which one is 'more accurate', I personally believe that good athlete's depend more on the countries population than wealth, as talent will always emerge from a pool and GDP is not a great indicator if the country has the resources to support an athlete.

37.5.1 Medals per group (season, sport,...)

I mentioned earlier that Scandinavian countries are good at winter sports, let's prove it, we divide our dataset in 'Summer' and 'Winter'.

```
medals_country_df['season'] = medals_country_df.index.get_level_values('Games').
↪str[5:]
medals_country_df.head()
```

```
Medal                      Gold  Silver  Bronze  year  season
Games        NOC Sport
1896 Summer AUS Athletics     2       0       0  1896  Summer
                Tennis        0       0       1  1896  Summer
            AUT Cycling       1       0       2  1896  Summer
                Swimming      1       1       0  1896  Summer
            DEN Fencing       0       0       1  1896  Summer
```

By grouping per season and country and counting the total amount of medals (here gold, silver or bronze does not matter) we get 2 values for each country. We first sum all types of medals, then group by season and country and last pivot the season feature to create columns for each season

```
medals_season_df = medals_country_df.set_index('season', append=True)[['Gold', 'Silver
↪', 'Bronze']].sum(axis='columns').unstack('NOC').groupby('season').sum().T
medals_season_df
```

```
season  Summer  Winter
NOC
AFG        2.0     0.0
AHO        1.0     0.0
ALG       17.0     0.0
```

(continues on next page)

(continued from previous page)

```
ANZ        11.0     0.0
ARG        74.0     0.0
..          ...     ...
VIE         4.0     0.0
WIF         2.0     0.0
YUG        83.0     4.0
ZAM         2.0     0.0
ZIM         8.0     0.0

[149 rows x 2 columns]
```

Using our contingengy table chi squared test we can easily find out if for certain rows the distribution of our 2 columns (Summer and Winter) is skewed.

```
F, p, df, exp = scipy.stats.chi2_contingency(medals_season_df)
F, p
```

```
(2662.3291002167407, 0.0)
```

with a p-value of 0.0 we know there is a definite shift for certain countries, using the expected values we calculate the diff and sort it by descending order on summer

```
medals_season_diff_df = medals_season_df-exp
medals_season_diff_df.sort_values(by='Summer', ascending=False)
```

```
season      Summer      Winter
NOC
GBR      85.965542  -85.965542
USA      68.575099  -68.575099
HUN      62.780286  -62.780286
AUS      56.924241  -56.924241
ROU      39.753392  -39.753392
..             ...         ...
SUI     -78.671749   78.671749
FIN     -82.659263   82.659263
CAN     -90.223556   90.223556
AUT    -133.247569  133.247569
NOR    -193.088246  193.088246

[149 rows x 2 columns]
```

Although having bad weather, the british do not fancy some snow at all, similar for the United States. In contrast, countries as Norway, Austria, Canada, Finland, Switzerland, … really excel in winter sports!

Similarly to this analysis we can to the same for countries and types of sports, we do the same manipulation and obtain the next view of our data.

```
medals_sport_df = medals_country_df.sum(axis='columns').unstack('Sport').groupby('NOC
↪').sum()
medals_sport_df.index = medals_sport_df.index.astype('str')
medals_sport_df
```

```
/tmp/ipykernel_14702/611435394.py:1: FutureWarning: Dropping of nuisance columns in␣
↪DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version␣
↪this will raise TypeError.  Select only valid columns before calling the reduction.
```

(continues on next page)

(continued from previous page)

```
medals_sport_df = medals_country_df.sum(axis='columns').unstack('Sport').groupby(
↪'NOC').sum()
```

```
Sport  Aeronautics  Alpine Skiing  Alpinism  Archery  Art Competitions  \
NOC
AFG            0.0            0.0       0.0      0.0               0.0
AHO            0.0            0.0       0.0      0.0               0.0
ALG            0.0            0.0       0.0      0.0               0.0
ANZ            0.0            0.0       0.0      0.0               0.0
ARG            0.0            0.0       0.0      0.0               0.0
..             ...            ...       ...      ...               ...
VIE            0.0            0.0       0.0      0.0               0.0
WIF            0.0            0.0       0.0      0.0               0.0
YUG            0.0            2.0       0.0      0.0               0.0
ZAM            0.0            0.0       0.0      0.0               0.0
ZIM            0.0            0.0       0.0      0.0               0.0

Sport  Athletics  Badminton  Baseball  Basketball  Basque Pelota  ...  \
NOC                                                               ...
AFG          0.0        0.0       0.0         0.0            0.0  ...
AHO          0.0        0.0       0.0         0.0            0.0  ...
ALG          9.0        0.0       0.0         0.0            0.0  ...
ANZ          1.0        0.0       0.0         0.0            0.0  ...
ARG          5.0        0.0       0.0         2.0            0.0  ...
..           ...        ...       ...         ...            ...  ...
VIE          0.0        0.0       0.0         0.0            0.0  ...
WIF          2.0        0.0       0.0         0.0            0.0  ...
YUG          2.0        0.0       0.0         7.0            0.0  ...
ZAM          1.0        0.0       0.0         0.0            0.0  ...
ZIM          0.0        0.0       0.0         0.0            0.0  ...

Sport  Table Tennis  Taekwondo  Tennis  Trampolining  Triathlon  Tug-Of-War  \
NOC
AFG             0.0        2.0     0.0           0.0        0.0         0.0
AHO             0.0        0.0     0.0           0.0        0.0         0.0
ALG             0.0        0.0     0.0           0.0        0.0         0.0
ANZ             0.0        0.0     1.0           0.0        0.0         0.0
ARG             0.0        1.0     5.0           0.0        0.0         0.0
..              ...        ...     ...           ...        ...         ...
VIE             0.0        1.0     0.0           0.0        0.0         0.0
WIF             0.0        0.0     0.0           0.0        0.0         0.0
YUG             2.0        0.0     0.0           0.0        0.0         0.0
ZAM             0.0        0.0     0.0           0.0        0.0         0.0
ZIM             0.0        0.0     0.0           0.0        0.0         0.0

Sport  Volleyball  Water Polo  Weightlifting  Wrestling
NOC
AFG           0.0         0.0            0.0        0.0
AHO           0.0         0.0            0.0        0.0
ALG           0.0         0.0            0.0        0.0
ANZ           0.0         0.0            0.0        0.0
ARG           1.0         0.0            2.0        0.0
..            ...         ...            ...        ...
VIE           0.0         0.0            1.0        0.0
WIF           0.0         0.0            0.0        0.0
YUG           0.0         7.0            0.0       16.0
```

(continues on next page)

(continued from previous page)

```
ZAM           0.0         0.0            0.0         0.0
ZIM           0.0         0.0            0.0         0.0

[149 rows x 66 columns]
```

So instead of knowing which countries are performing different on summer and winter games, we can not figure out which sports are excelled by a nation.

```
F, p, df, exp = scipy.stats.chi2_contingency(medals_sport_df)
F, p
```

```
(44987.26611756852, 0.0)
```

Again a p-value of 0.0 indicate the correlation is not a coincidence, so we should investigate with the differences, as we have a lot of sports and countries, it would be wise to select a single country or sport.

```
medals_sport_diff_df = medals_sport_df-exp
medals_sport_diff_df.loc['NED'].sort_values(ascending=False).head(10)
```

```
Sport
Speed Skating       62.077260
Cycling             33.908452
Swimming            25.759395
Hockey              14.862949
Rowing              14.234062
Equestrianism       14.231480
Sailing             11.050366
Judo                10.992856
Art Competitions     3.047785
Football             1.774823
Name: NED, dtype: float64
```

As you can see I took the Dutch which clearly have a favorite. Speed Skating and Hockey where 2 sports where I thought they would be scoring well, but they also perform well on cycling and swimming!

It also works the other way around if we select a sport and see which countries are good, I wanted to known which countries are good at sailing.

```
medals_sport_diff_df['Sailing'].sort_values(ascending=False).head()
```

```
NOC
GBR    32.429944
DEN    23.764377
NZL    18.539080
SWE    16.605895
ESP    14.554508
Name: Sailing, dtype: float64
```

Looks like Great Britain is good at sailing, all those years of colonialism still seem to pay of…

37.5.2 Athlete attributes

In this section we will be looking at attributes from athletes, age, height and weight are all given in the dataset, yet with a lot of missing values. To make our life easier I created 2 functions that retrieves groups of athletes based on a grouping and the mean of each groups for the grouping, also you can set if we only take athletes that received a medal.

```
def group_athletes(grouping=['Sex'], agg=False, medals=False):
    df = athlete_events.drop_duplicates(subset=['Name', 'Age', 'NOC'])
    df = df.dropna(subset=['Age', 'Height', 'Weight'])
    if medals:
        df = df[~df.Medal.isna()]
    return [x[1][['Age', 'Height', 'Weight']] for x in df.groupby(grouping) if␣
↪len(x[1])>5]
```

```
def median_athletes(grouping=['Sex'], medals=False):
    df = athlete_events.dropna(subset=['Age', 'Height', 'Weight'])
    if medals:
        df = df[~df.Medal.isna()]
    return df.groupby(grouping)[['Age', 'Height', 'Weight']].median()
```

To give an example, here is the result of the mean for athletes grouped per gender. I want to remark here that I did not perform a non-normal test as a fact that I always know data such as this is not normal distributed. A mean is not the perfect indicator for this!

```
median_athletes(['Sex'])
```

```
      Age  Height  Weight
Sex
F    23.0   168.0    59.0
M    25.0   179.0    74.0
```

Now for each attribute we would like to perform an ANOVA with the initial values, we can do this with the scipy library, where we supply the data from the (in this case) 2 groups.

```
F, p = scipy.stats.f_oneway(*group_athletes(['Sex']))
print(f'F: {F}')
print(f'p: {p}')
```

```
F: [ 2099.07936998 41302.70098716 44472.78629524]
p: [0. 0. 0.]
```

You can See that the p-values are all less that 0.05 indicating no chance this happend by accident, so there is a clear difference for Age, Height and Weight for Male and Female Athletes. Which was also visible in the earlier table we created, yet we know it is not by random coincidence.

How about we only take athletes that have obtained a medal? do we see a difference then?

```
F, p = scipy.stats.f_oneway(*group_athletes(['Sex'], medals=True))
print(f'F: {F}')
print(f'p: {p}')
```

```
F: [ 162.38167465 5281.5001145  6522.92324657]
p: [4.62479657e-37 0.00000000e+00 0.00000000e+00]
```

Again the results are very clear, yet we can see that the F-Values are much lower, indicating the difference is much lower, let's look at medians

```
median_athletes(['Sex'], medals=True)
```

```
      Age  Height  Weight
Sex
F    24.0   170.0    63.0
M    25.0   181.0    78.0
```

Although no big differences most values have shifted upwards indicating being taller and heavier gives you more chance on a medal?

Instead of focussing on gender, let's look at sports, as I assume not every sports prefers the same athlete.

```
F, p = scipy.stats.f_oneway(*group_athletes(['Sport'], medals=True))
print(f'F: {F}')
print(f'p: {p}')
```

```
F: [ 94.89968089 172.60170532 106.28595443]
p: [0. 0. 0.]
```

F values are much less, yet we should not compare as we changed our grouping, the p-values as usually are so low there is no chance of randomness.

As we have too many sports, I decided to sort them by Height and only show the shortest.

```
median_sport_df = median_athletes(['Sport']).dropna().sort_values(by='Height')
median_sport_df.head()
```

```
                        Age  Height  Weight
Sport
Gymnastics             21.0   164.0    58.0
Diving                 22.0   167.0    60.0
Trampolining           24.0   167.0    58.0
Figure Skating         22.0   168.0    57.0
Synchronized Swimming  22.0   168.0    55.0
```

Clearly there are some sports that favor being small, there are probably numerous arguments why that would be, but I'm not going to go there.

Now that we are here, let's look at the sports with the heaviest athletes.

```
median_sport_df.sort_values(by='Weight', ascending=False).head()
```

```
             Age  Height  Weight
Sport
Tug-Of-War  24.5   183.5    90.0
Bobsleigh   28.0   182.0    90.0
Basketball  25.0   191.0    85.0
Baseball    26.0   183.0    85.0
Water Polo  25.0   185.0    84.0
```

Although not a sport anymore, Tug-Of-War still has the heaviest contestants, which indicates that weight sure is a way to win an old-fashioned tug of war.

To give it some more insight, we could divide each row with it's mean, this would give a differential compared to the mean.

```python
median_sport_df.apply(lambda x: x-x.mean())
```

```
                               Age     Height  Weight
Sport
Gymnastics               -4.464286 -11.544643   -13.0
Diving                   -3.464286  -8.544643   -11.0
Trampolining             -1.464286  -8.544643   -13.0
Figure Skating           -3.464286  -7.544643   -14.0
Synchronized Swimming    -3.464286  -7.544643   -16.0
Weightlifting            -0.464286  -7.544643     4.0
Rhythmic Gymnastics      -7.464286  -7.544643   -22.0
Table Tennis              0.535714  -5.544643    -7.0
Softball                  0.535714  -5.544643    -5.0
Short Track Speed Skating -2.464286  -5.544643    -8.0
Freestyle Skiing         -1.464286  -5.544643    -6.0
Wrestling                -0.464286  -3.544643     1.0
Archery                  -0.464286  -3.544643    -2.0
Boxing                   -2.464286  -3.544643    -8.0
Snowboarding             -1.464286  -2.544643    -3.0
Hockey                   -0.464286  -2.544643    -2.0
Triathlon                 2.535714  -2.544643   -10.0
Golf                      3.035714  -2.544643    -1.0
Rugby                    -2.464286  -2.544643     5.0
Alpine Skiing            -2.464286  -2.544643     0.0
Cross Country Skiing      0.535714  -2.544643    -5.0
Shooting                  5.535714  -1.544643     3.0
Art Competitions         16.535714  -1.544643     7.5
Badminton                 0.535714  -1.544643    -3.0
Speed Skating            -1.464286  -1.544643    -1.0
Biathlon                  0.535714  -1.544643    -4.0
Rugby Sevens              0.535714  -1.544643     6.0
Lacrosse                  2.535714  -1.544643    -4.5
Equestrianism             7.535714  -1.544643    -3.0
Judo                     -0.464286  -1.544643     2.0
Football                 -2.464286  -0.544643     0.0
Curling                   4.535714  -0.544643     1.0
Athletics                -0.464286   0.455357    -4.0
Taekwondo                -1.464286   0.455357    -4.0
Luge                     -1.464286   1.455357     6.0
Nordic Combined          -1.464286   1.455357    -4.0
Cycling                  -1.464286   1.455357    -1.0
Ski Jumping              -2.464286   1.455357    -6.0
Skeleton                  3.535714   1.455357     5.0
Fencing                   0.535714   2.455357     1.0
Modern Pentathlon         0.535714   2.455357     0.0
Swimming                 -5.464286   3.455357    -1.0
Canoeing                 -0.464286   3.455357     6.0
Sailing                   2.535714   3.455357     3.0
Ice Hockey               -0.464286   4.455357    11.0
Tennis                   -0.464286   4.455357    -1.0
Motorboating              1.535714   5.455357     6.0
Bobsleigh                 2.535714   6.455357    19.0
Baseball                  0.535714   7.455357    14.0
Tug-Of-War               -0.964286   7.955357    19.0
Handball                  0.535714   8.455357    11.0
Rowing                   -0.464286   9.455357    11.0
Water Polo               -0.464286   9.455357    13.0
```

(continues on next page)

(continued from previous page)

```
Beach Volleyball            3.535714  10.455357     7.0
Volleyball                 -0.464286  11.455357     7.0
Basketball                 -0.464286  15.455357    14.0
```

This way you can see that the median basketball player is 15.5 cm taller than an average athlete.

Aside from grouping on 1 attribute (Gender or Sport) we can also combine them, but this makes things more complicated. Here we group on Gender and Sport type and only select medal wining athletes.

```
sport_gender_df = median_athletes(['Sex', 'Sport'], medals=True).dropna().unstack('Sex
↪')
sport_gender_df.head()
```

```
                Age         Height          Weight
Sex               F     M        F      M        F     M
Sport
Alpine Skiing  24.0  25.0  169.0  180.0   64.0  83.0
Archery        23.5  24.0  168.0  180.0   62.5  76.0
Athletics      25.0  24.0  170.0  182.0   60.0  74.0
Badminton      25.0  26.0  171.0  180.0   62.0  73.0
Basketball     25.0  25.0  183.0  198.0   73.0  94.0
```

The options of comparison grow exponentially with every grouping level, therefore I selected one which I thought might be interesting, we are comparing per sport the height of males and females. so a negative value means females are higher than males.

```
(sport_gender_df['Height']['M']-sport_gender_df['Height']['F']).rename('height_
↪difference').sort_values(ascending=False).dropna()
```

```
Sport
Golf                        21.5
Figure Skating              17.0
Handball                    15.0
Basketball                  15.0
Volleyball                  15.0
Swimming                    14.0
Speed Skating               14.0
Water Polo                  13.0
Biathlon                    13.0
Triathlon                   13.0
Cross Country Skiing        13.0
Skeleton                    12.5
Ski Jumping                 12.5
Snowboarding                12.0
Shooting                    12.0
Curling                     12.0
Judo                        12.0
Cycling                     12.0
Archery                     12.0
Trampolining                12.0
Athletics                   12.0
Sailing                     12.0
Taekwondo                   11.5
Alpine Skiing               11.0
Rowing                      11.0
Rugby Sevens                11.0
```

(continues on next page)

(continued from previous page)

```
Ice Hockey                   11.0
Hockey                       11.0
Fencing                      11.0
Canoeing                     11.0
Bobsleigh                    10.0
Weightlifting                10.0
Freestyle Skiing             10.0
Tennis                       10.0
Table Tennis                  9.5
Diving                        9.0
Gymnastics                    9.0
Badminton                     9.0
Equestrianism                 9.0
Beach Volleyball              9.0
Wrestling                     8.0
Luge                          8.0
Football                      8.0
Short Track Speed Skating     8.0
Boxing                        7.0
Modern Pentathlon             6.0
Name: height_difference, dtype: float64
```

Here you can read that e.g. basketbalplayers in general have a taller height, yet difference between male and female is also 15cms so the height advantage is not that appearant in female basketball. On the other side, Boxing has a lower height difference, yet boxing already was a sport that benefits smaller athletes than average.

To end this section I would like to take a grouping where the difference is not that obvious, by grouping per medal.

```
F, p = scipy.stats.f_oneway(*group_athletes(['Medal']))
print(f'F: {F}')
print(f'p: {p}')
```

```
F: [0.39329622 6.66008411 5.73133235]
p: [0.67483359 0.00128364 0.00324762]
```

You can see that for age we have a p-value of 0.67, indicating no difference in age for athletes that have obtained different types of medals, yet for height and weight the p-value is significant. However if we look at the median values we see nearly no difference.

```
median_athletes(['Medal'])
```

```
         Age  Height  Weight
Medal
Bronze  25.0   178.0    72.0
Gold    25.0   178.0    73.0
Silver  25.0   178.0    73.0
```

This is a great example of how significance does not imply relevance, the differences here are so small they are irrelevant.

37.6 Visualization

Before we start creating graphics, a little recall we started out with a view of our data for each games, NOC and sport the amount of medals

```
medals_country_df.head()
```

```
Medal                          Gold  Silver  Bronze  year  season
Games       NOC Sport
1896 Summer AUS Athletics         2       0       0  1896  Summer
                Tennis            0       0       1  1896  Summer
            AUT Cycling           1       0       2  1896  Summer
                Swimming          1       1       0  1896  Summer
            DEN Fencing           0       0       1  1896  Summer
```

What I would be interested in is the evoluation of amount of medals for the highest achieving countries, therefore we need a list of the best countries, I selected the top 10 countries with most medals.

```
most_medals = medals_country_df.groupby('NOC')[['Gold','Silver','Bronze']].sum().
↪sum(axis='columns').sort_values(ascending=False).head(10).index.values
most_medals
```

```
array(['USA', 'URS', 'GER', 'GBR', 'FRA', 'ITA', 'SWE', 'RUS', 'CHN',
       'AUS'], dtype=object)
```

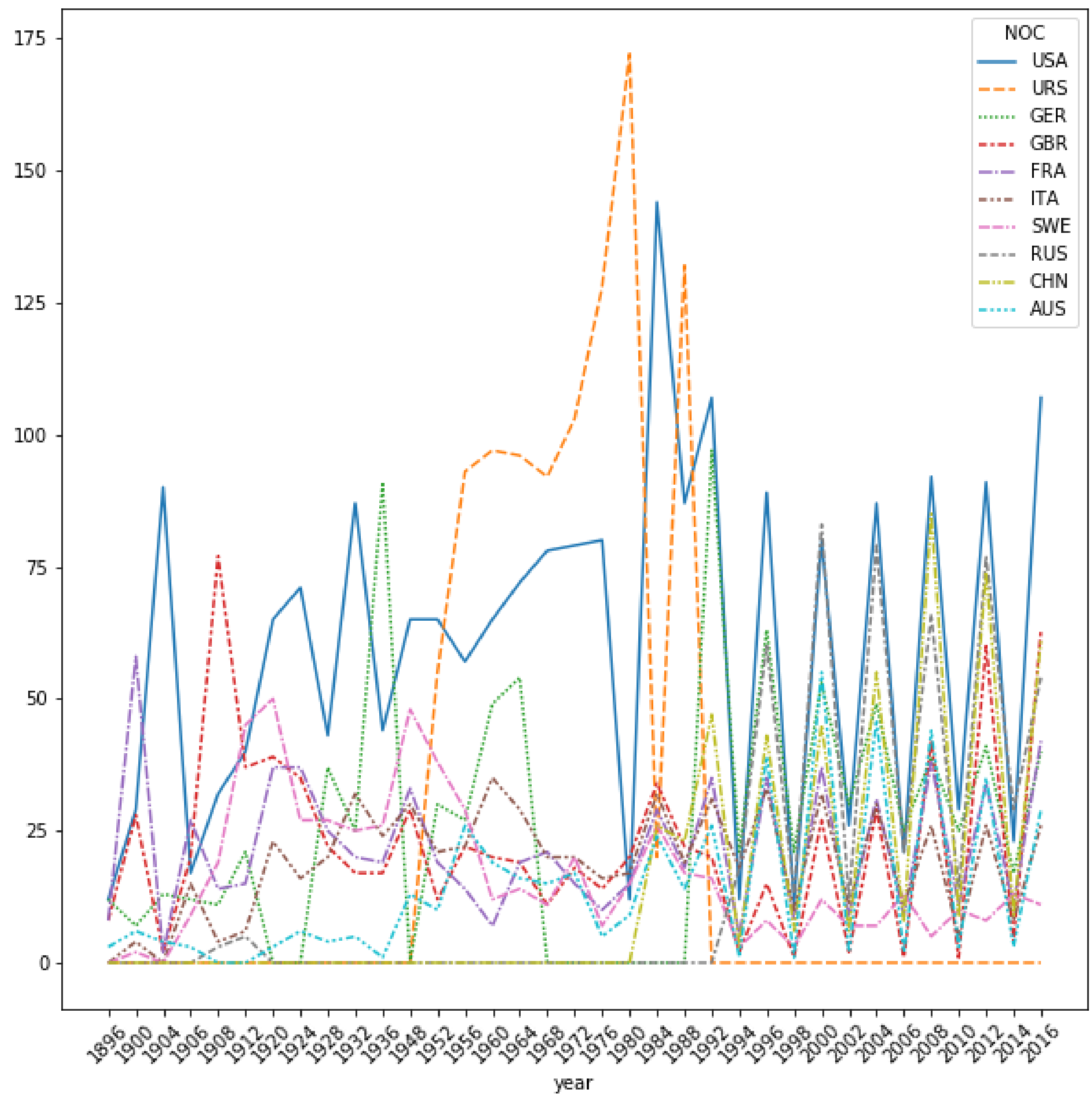
Now for those countries we create a new view on our data that contains the won medals for each of those countries.

```
medals_country_wide_df = medals_country_df.reset_index().groupby(['year','NOC'])[[
↪'Gold', 'Silver', 'Bronze']].sum().sum(axis='columns').unstack()
medals_country_wide_df = medals_country_wide_df[most_medals].fillna(0)
medals_country_wide_df.tail()
```

```
NOC     USA  URS   GER   GBR   FRA   ITA   SWE   RUS   CHN   AUS
year
2008   92.0  0.0  39.0  42.0  38.0  26.0   5.0  66.0  85.0  44.0
2010   29.0  0.0  25.0   1.0  11.0   5.0  10.0  14.0   9.0   3.0
2012   91.0  0.0  41.0  60.0  34.0  26.0   8.0  77.0  74.0  35.0
2014   23.0  0.0  17.0   4.0  11.0   8.0  13.0  29.0   9.0   3.0
2016  107.0  0.0  40.0  63.0  42.0  26.0  11.0  54.0  61.0  29.0
```

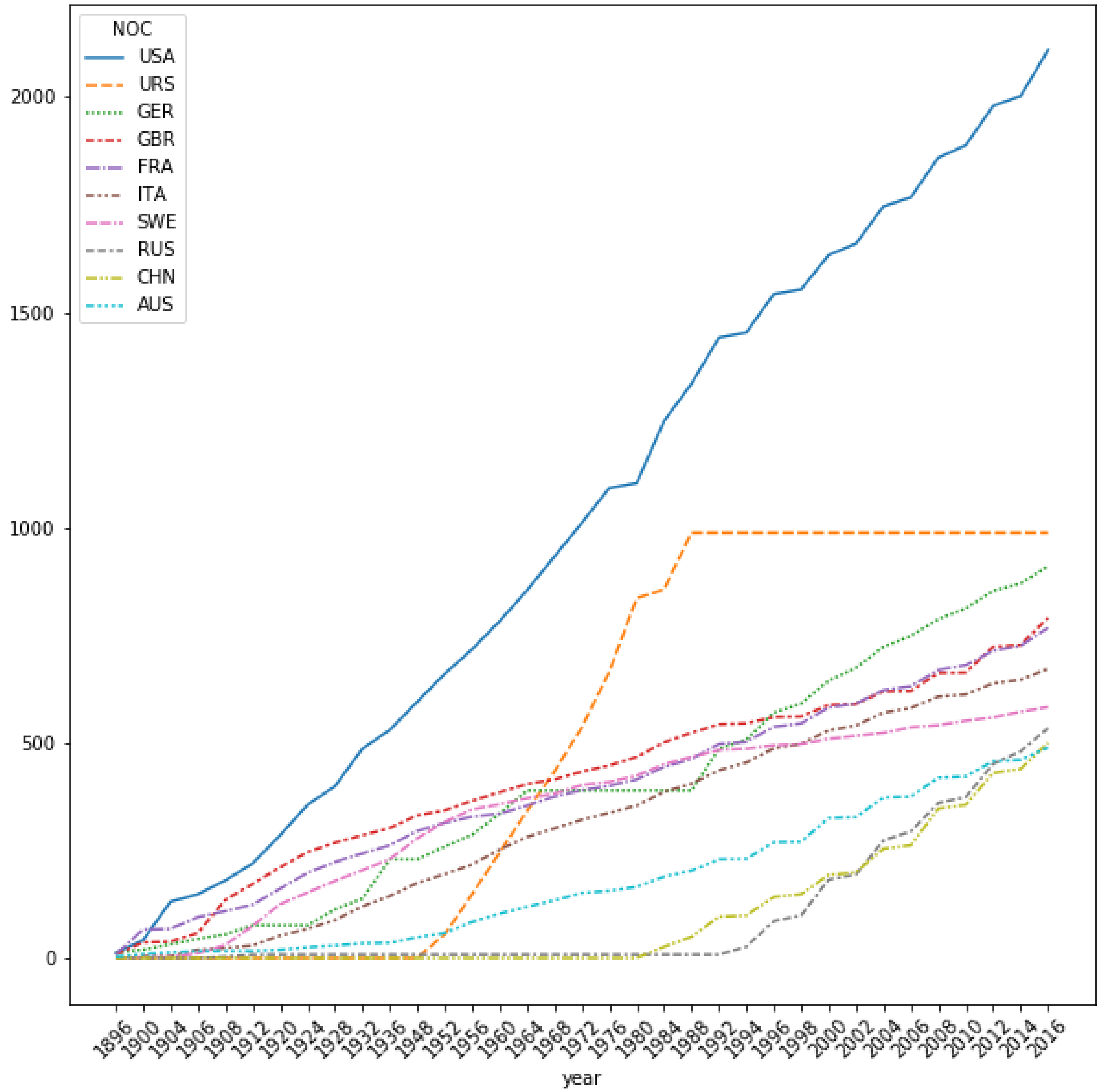
We can create a simple line plot for this, where the x-axis is the chronological years of each games and y is the amount of medals

```
sns.lineplot(data=medals_country_wide_df)
plt.xticks(rotation=45)
plt.show()
```



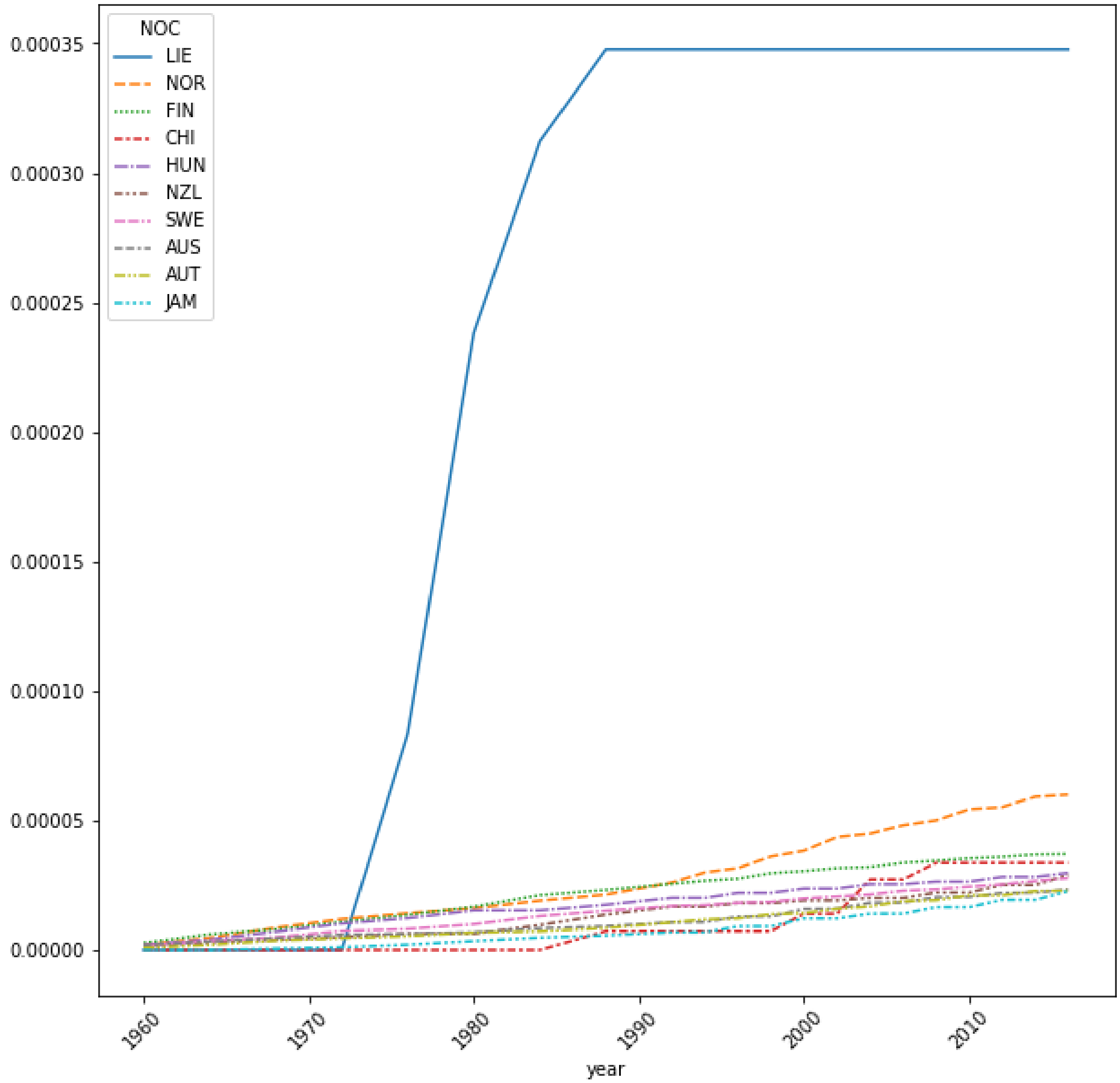
Looks like we forgot something, we are plotting the amount of medals per year and not cumulative, fortunately a builtin method can solve this

```
sns.lineplot(data=medals_country_wide_df.cumsum())
plt.xticks(rotation=45)
plt.show()
```



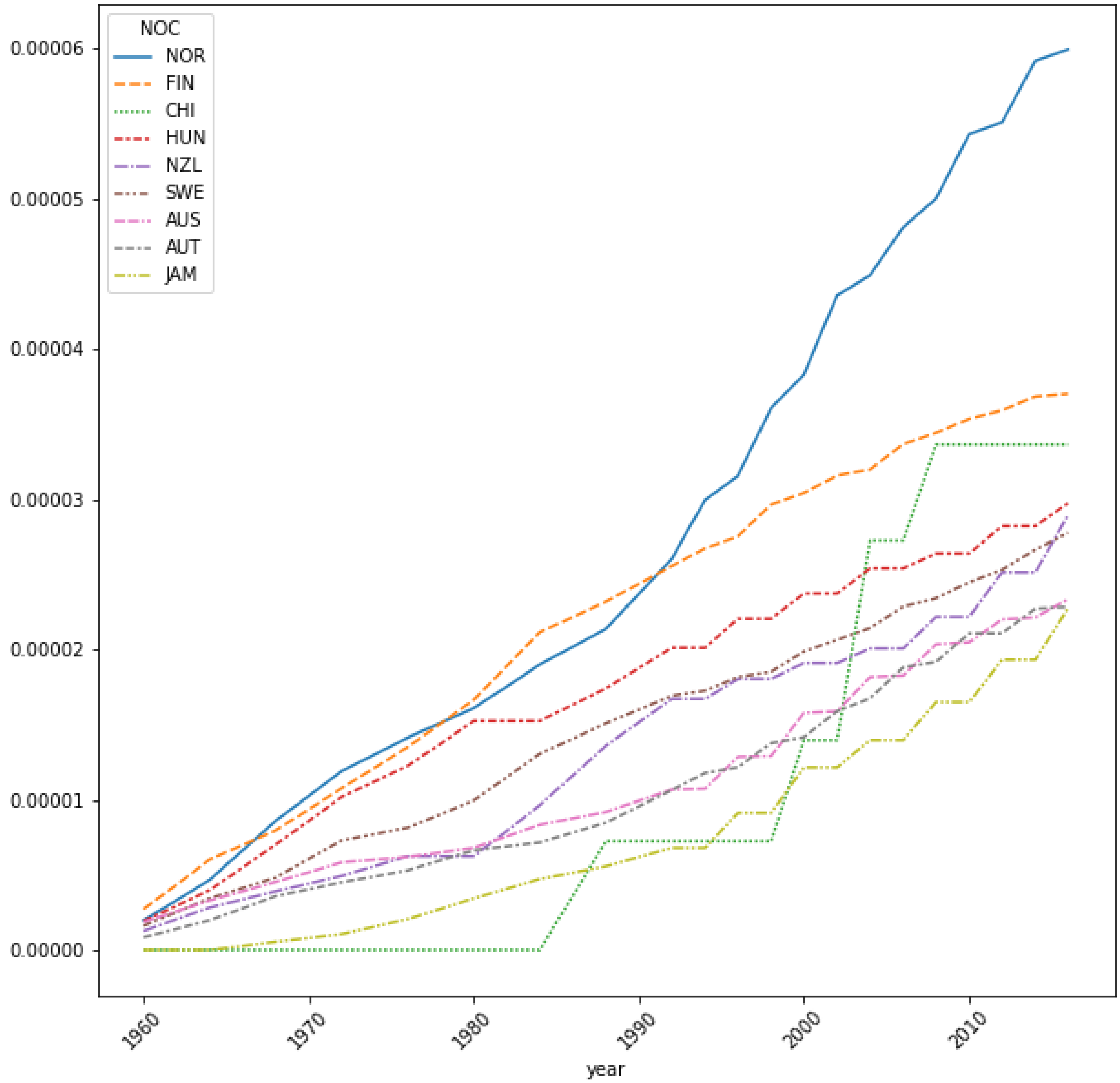
I did the same for the population corrected data, creating a line plot, this is in my opinion more interesting as it gives a more honest take on the competition.

```
most_medals_pop = medals_pop_df.groupby('NOC')[['Gold','Silver','Bronze']].sum().
↪sum(axis='columns').sort_values(ascending=False).head(10).index.values
medals_pop_df['year'] = medals_pop_df.index.get_level_values('Games').str[:4].astype(
↪'int')
medals_country_wide_pop_df = medals_pop_df.reset_index().groupby(['year','NOC'])[[
↪'Gold', 'Silver', 'Bronze']].sum().sum(axis='columns').unstack()
medals_country_wide_pop_df = medals_country_wide_pop_df[most_medals_pop].fillna(0)
sns.lineplot(data=medals_country_wide_pop_df.cumsum())
plt.xticks(rotation=45)
plt.show()
```



There seems to have been a golden age for Liechtenstein, as they are taking up a lot of space I opted to remove them and plot again

```
sns.lineplot(data=medals_country_wide_pop_df.drop(columns=['LIE']).cumsum())
plt.xticks(rotation=45)
plt.show()
```

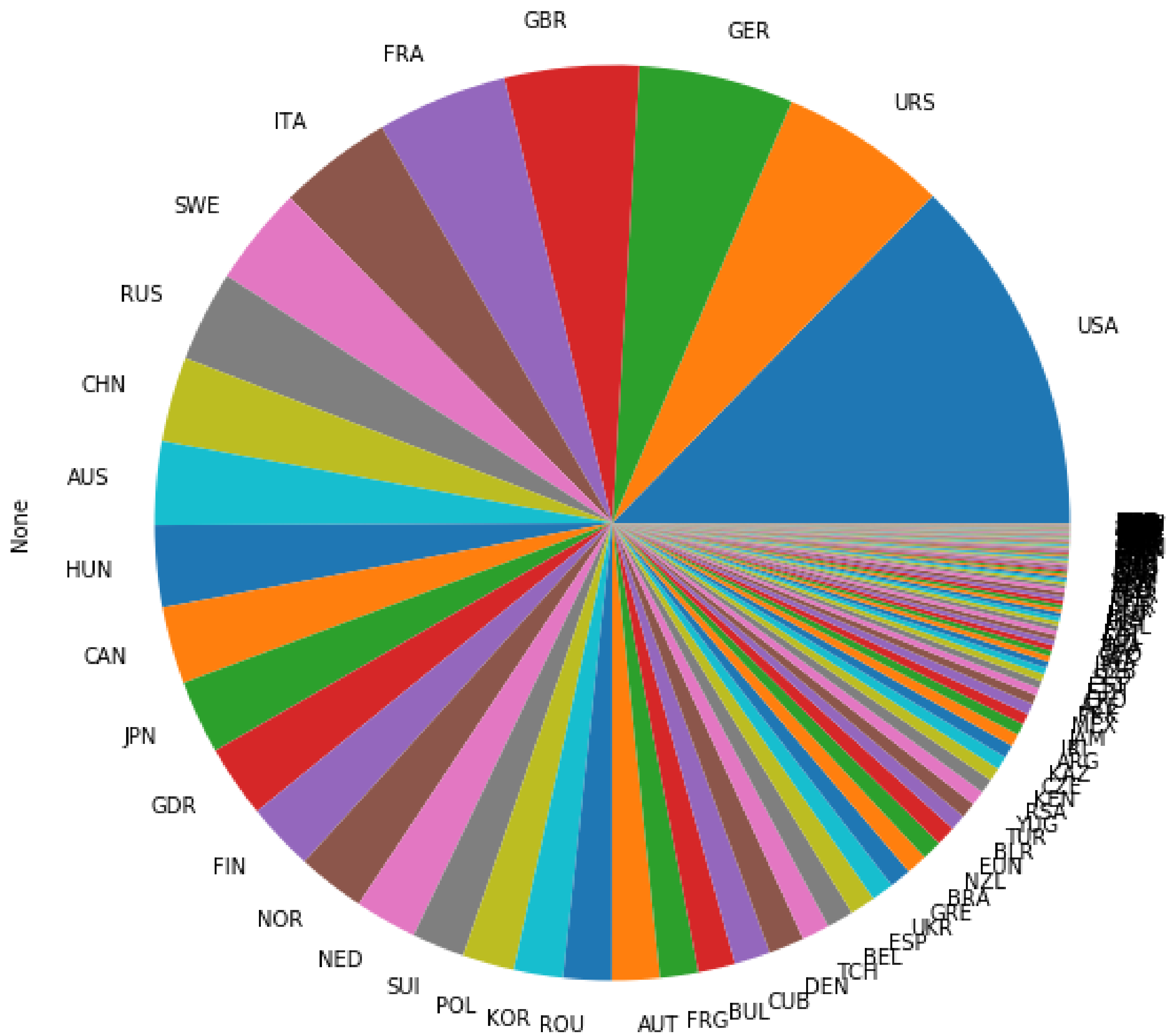


Great! a lot of other interesting countries performances, note that CHI stands for Chile which catches up phenomenally.

Another take would be a pie chart, although not my favorite it would make a good option in this situation, as we want to compare the relative portions of countries. When we use the regular data we obtain the following.

```
medals_country_df.groupby(level='NOC').sum().sum(axis='columns').sort_
↪values(ascending=False).plot.pie()
```

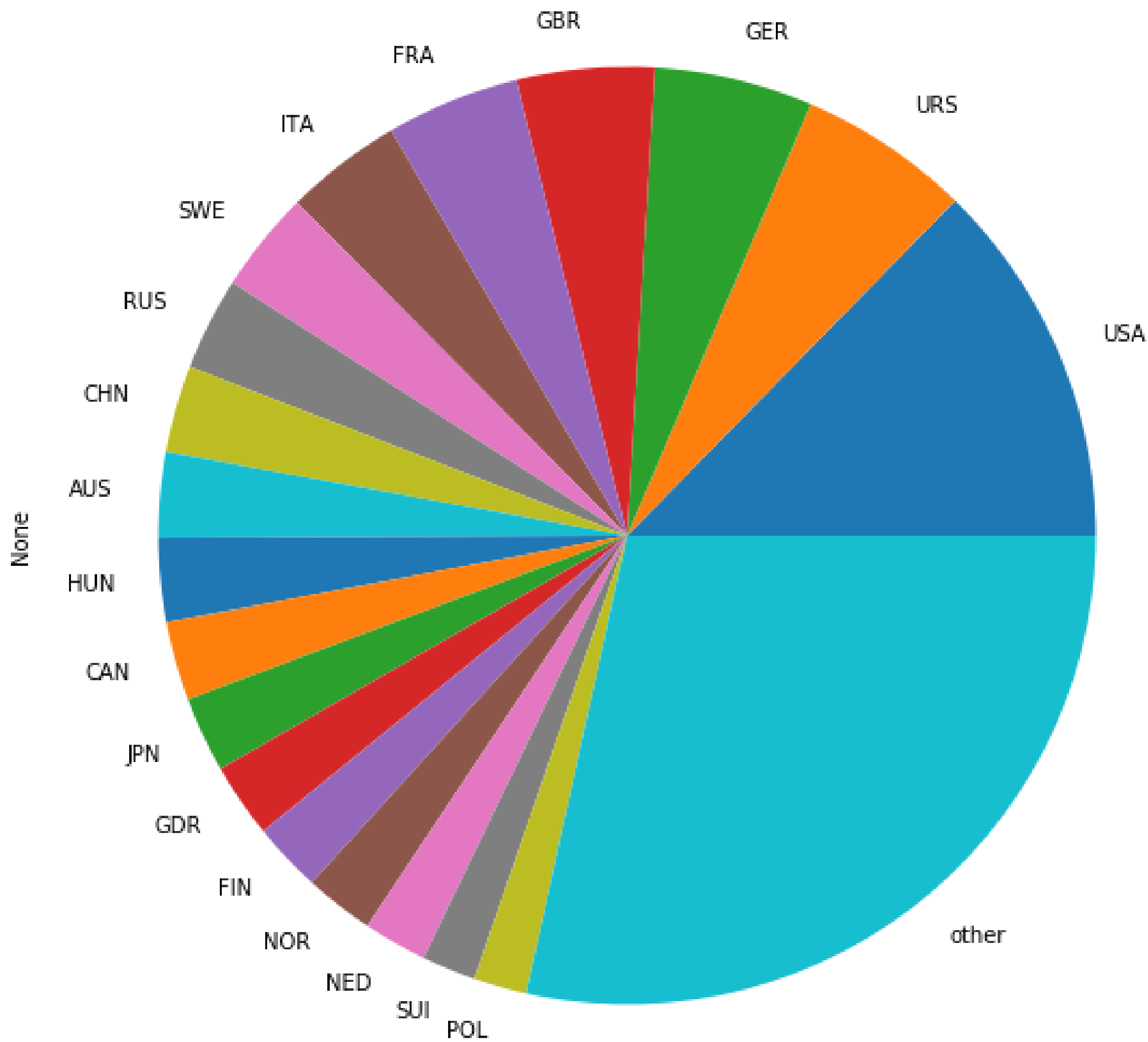
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba75f21af0>
```



Verry messy, as most countries are not visible on the pie chart, a good option would be to only take the top 20 countries and put the others in a 'other' category.

```
medals_country_vis_df = medals_country_df.groupby(level='NOC').sum().sum(axis='columns
↪').sort_values(ascending=False)[:19]
medals_country_vis_df['other'] = medals_country_df.groupby(level='NOC').sum().
↪sum(axis='columns').sort_values(ascending=False)[19:].sum()
medals_country_vis_df.plot.pie()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba75cff940>
```

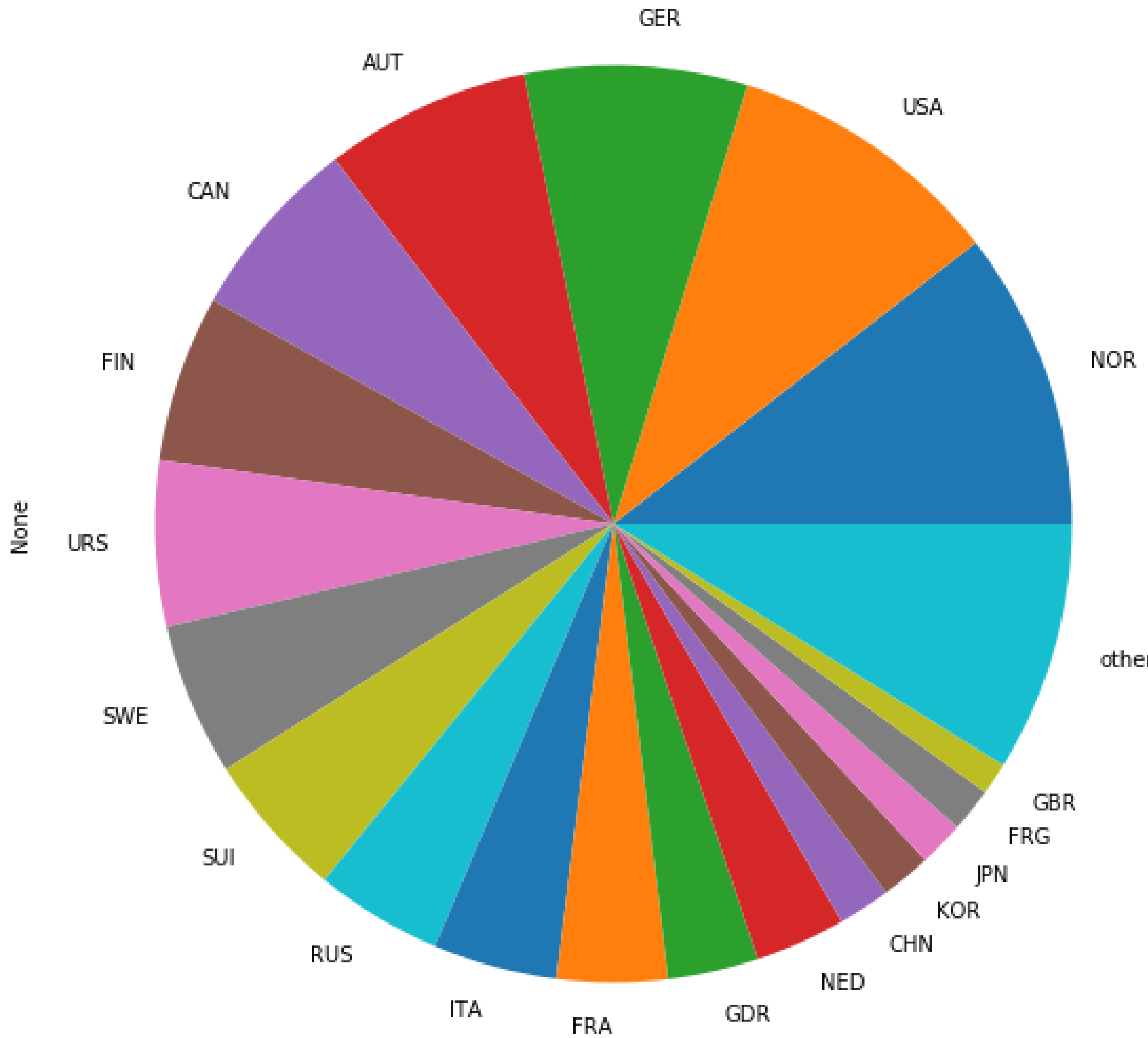


Much better, with this pie plot we can see that 10 countries obtained about half of all medals and the next 10 have about 25%, the other 130 countries are in the botton quarter.

Now to add more depth we can divide our dataset, something we mentioned earlier is the dominance in winter sports, here we create the same pie chart but only take events from winter games.

```
medals_winter_df = medals_country_df[medals_country_df.season=='Winter'].
↪groupby(level='NOC').sum().sum(axis='columns').sort_values(ascending=False)[:19]
medals_winter_df['other'] = medals_country_df[medals_country_df.season=='Winter'].
↪groupby(level='NOC').sum().sum(axis='columns').sort_values(ascending=False)[19:].
↪sum()
medals_winter_df.plot.pie()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba75c709d0>
```

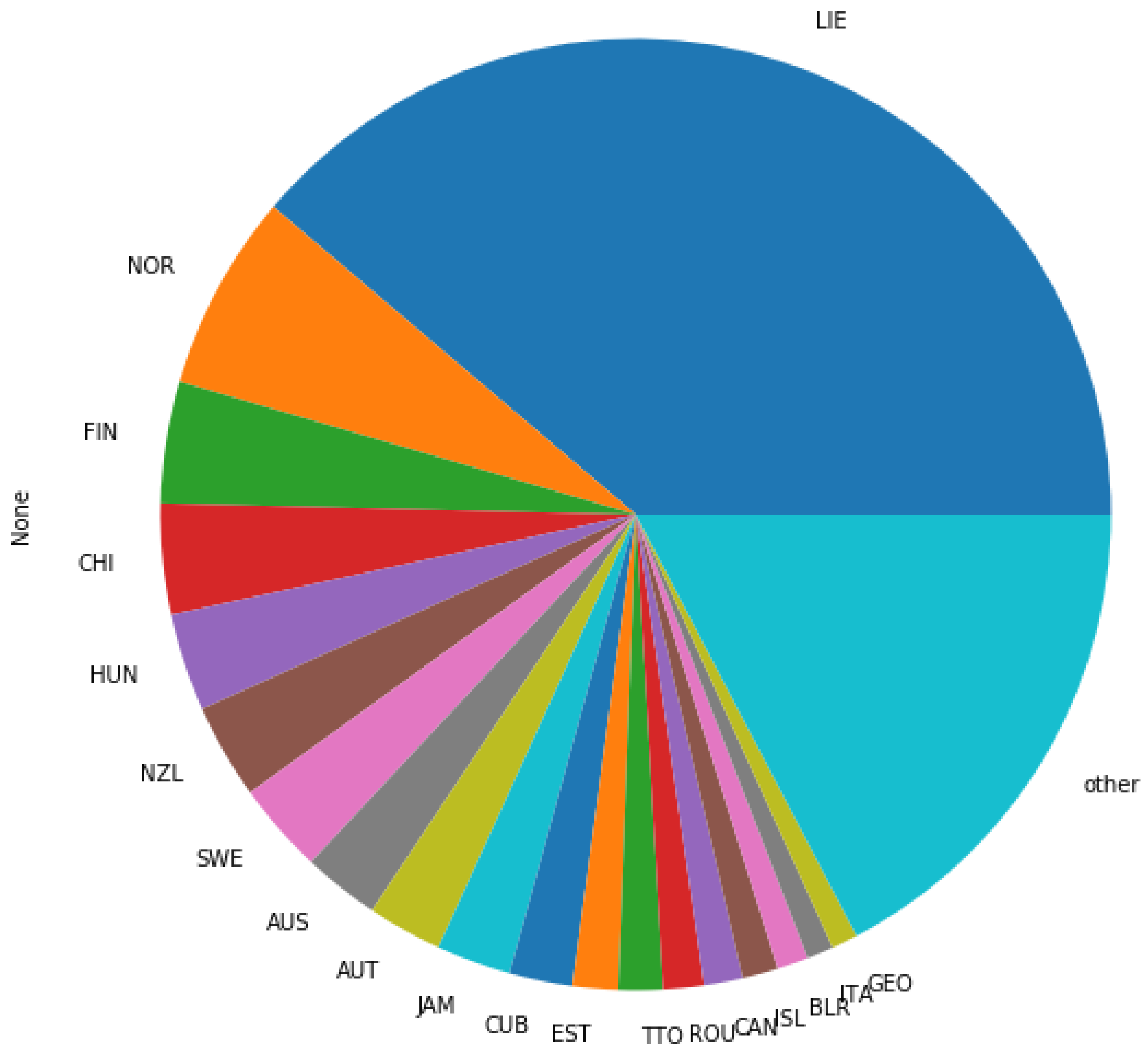


You can compare them and see that some countries fall and some rise, indicating that countries definitely have a preference.

Again we can do the same with population corrected data.

```
medals_pop_vis_df = medals_pop_df.groupby(level='NOC').sum()[['Gold','Silver','Bronze
↪']].sum(axis='columns').sort_values(ascending=False)[:19]
medals_pop_vis_df['other'] = medals_pop_df.groupby(level='NOC').sum()[['Gold','Silver
↪','Bronze']].sum(axis='columns').sort_values(ascending=False)[19:].sum()
(medals_pop_vis_df*1200).plot.pie()
```

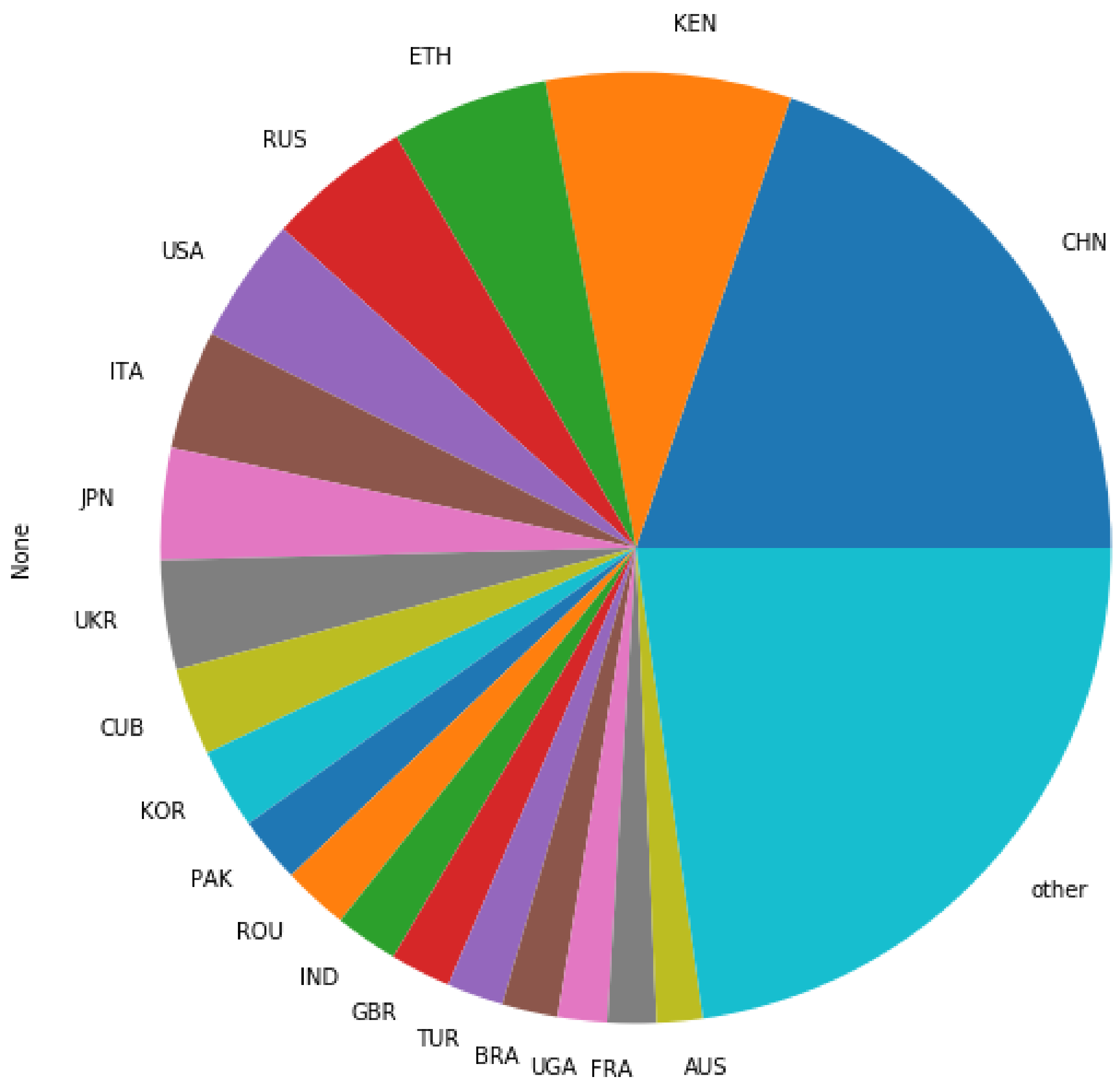
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba75bef760>
```



Or GDP corrected data

```
medals_gdp_vis_df = medals_country_gdp_df.groupby(level='NOC').sum()[['Gold','Silver',
↪'Bronze']].sum(axis='columns').sort_values(ascending=False)[:19]
medals_gdp_vis_df['other'] = medals_country_gdp_df.groupby(level='NOC').sum()[['Gold',
↪'Silver','Bronze']].sum(axis='columns').sort_values(ascending=False)[19:].sum()
medals_gdp_vis_df.plot.pie()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba75b6e1f0>
```



37.6.1 best performing per sport

To visualise the best performing country per sport we first need the country that won the most medals per sport. we do this with the following code

```
best_country_sport_df = pd.concat(
    [
        medals_country_df.groupby(level=['NOC', 'Sport']).sum().sum(axis='columns').
↪groupby(level='Sport').apply(lambda x: x.idxmax()[0]),
        medals_country_df.groupby(level=['NOC', 'Sport']).sum().sum(axis='columns').
↪groupby(level='Sport').apply(lambda x: x.max())
    ], axis='columns', keys=['country', 'medals']
)
```

(continues on next page)

(continued from previous page)

```
best_country_sport_df.head()
```

```
                  country  medals
Sport
Aeronautics           SUI       1
Alpine Skiing         AUT      80
Alpinism              AUS       1
Archery               KOR      30
Art Competitions      GER      20
```

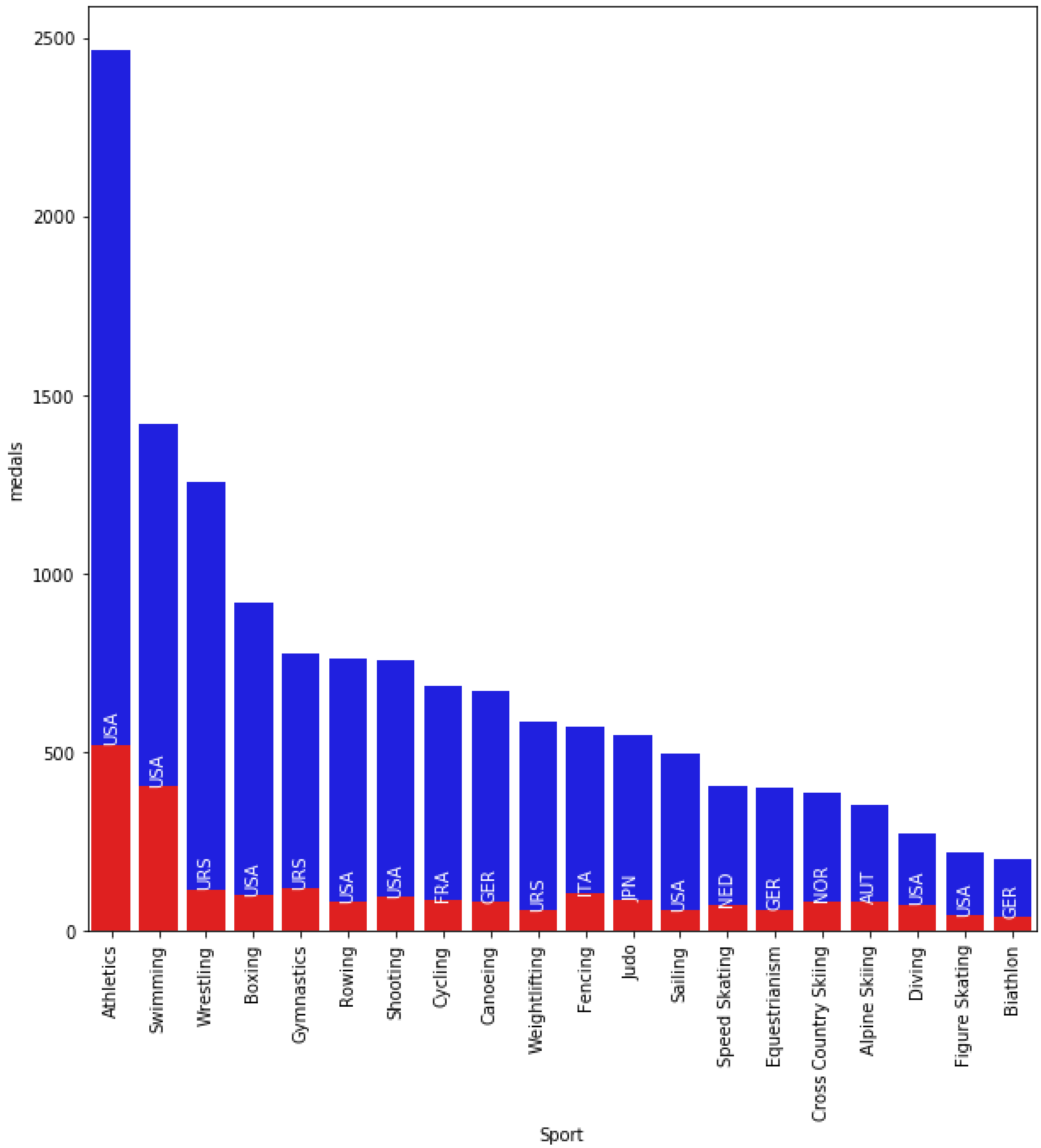
As there are to many sports, I opted to only visualise the top 20 most popular sports, by the amount of medals

```
total_medals_sport = medals_country_df.groupby(level='Sport').sum().sum(axis='columns
↪').rename('medals').sort_values(ascending=False).reset_index().head(20)
popular_sports = list(total_medals_sport.Sport)
best_country_sport_df.loc[popular_sports].medals
```

```
Sport
Athletics              521
Swimming               404
Wrestling              116
Boxing                 100
Gymnastics             118
Rowing                  81
Shooting                94
Cycling                 84
Canoeing                80
Weightlifting           55
Fencing                105
Judo                    84
Sailing                 58
Speed Skating           71
Equestrianism           59
Cross Country Skiing    83
Alpine Skiing           80
Diving                  73
Figure Skating          41
Biathlon                38
Name: medals, dtype: int64
```

Now we can create a bar plot, where the portion of each best performing country is shown together with the region name.

```
sns.barplot(x=total_medals_sport.Sport.astype('str'), y=total_medals_sport.medals,␣
↪color='b')
sns.barplot(x=popular_sports, y=best_country_sport_df.loc[popular_sports].medals,␣
↪color='r')

for idx, sport in enumerate(popular_sports):
    plt.text(idx, best_country_sport_df.loc[sport].medals+10, best_country_sport_df.
↪loc[sport].country, horizontalalignment='center', size='medium', color='white',␣
↪rotation=90)

plt.xticks(rotation=90)
plt.show()
```



This both indicates the popularity of the sport (by amount of total medals) and the amount of medals won by the best performing country.

Another approach would be to use the difference between truth and expected values, we calculated the difference earlier.

```
medals_sport_diff_df.head()
```

```
Sport   Aeronautics  Alpine Skiing  Alpinism   Archery  Art Competitions  \
NOC
AFG       -0.000120      -0.042382 -0.000720 -0.017649         -0.016088
```

(continues on next page)

(continued from previous page)

```
AHO      -0.000060      -0.021191 -0.000360 -0.008825         -0.008044
ALG      -0.001021      -0.360247 -0.006123 -0.150018         -0.136751
ANZ      -0.000660      -0.233101 -0.003962 -0.097070         -0.088486
ARG      -0.004442      -1.568135 -0.026654 -0.653020         -0.595270

Sport   Athletics  Badminton  Baseball  Basketball  Basque Pelota  ...  \
NOC                                                                ...
AFG    -0.295714  -0.009845 -0.001801   -0.010806      -0.000120  ...
AHO    -0.147857  -0.004923 -0.000900   -0.005403      -0.000060  ...
ALG     6.486433  -0.083684 -0.015308   -0.091848      -0.001021  ...
ANZ    -0.626426  -0.054148 -0.009905   -0.059431      -0.000660  ...
ARG    -5.941410  -0.364269 -0.066635    1.600192      -0.004442  ...

Sport   Table Tennis  Taekwondo    Tennis  Trampolining  Triathlon  Tug-Of-War  \
NOC
AFG       -0.009125   1.982711 -0.022212     -0.003122  -0.003002   -0.001801
AHO       -0.004562  -0.008644 -0.011106     -0.001561  -0.001501   -0.000900
ALG       -0.077560  -0.146956 -0.188798     -0.026534  -0.025513   -0.015308
ANZ       -0.050186  -0.095089  0.877836     -0.017169  -0.016509   -0.009905
ARG       -0.337616   0.360307  4.178173     -0.115500  -0.111058   -0.066635

Sport   Volleyball  Water Polo  Weightlifting  Wrestling
NOC
AFG     -0.010085   -0.011406      -0.070477  -0.151039
AHO     -0.005043   -0.005703      -0.035238  -0.075519
ALG     -0.085725   -0.096950      -0.599052  -1.283828
ANZ     -0.055469   -0.062733      -0.387622  -0.830712
ARG      0.626846   -0.422019      -0.607636  -5.588426

[5 rows x 66 columns]
```

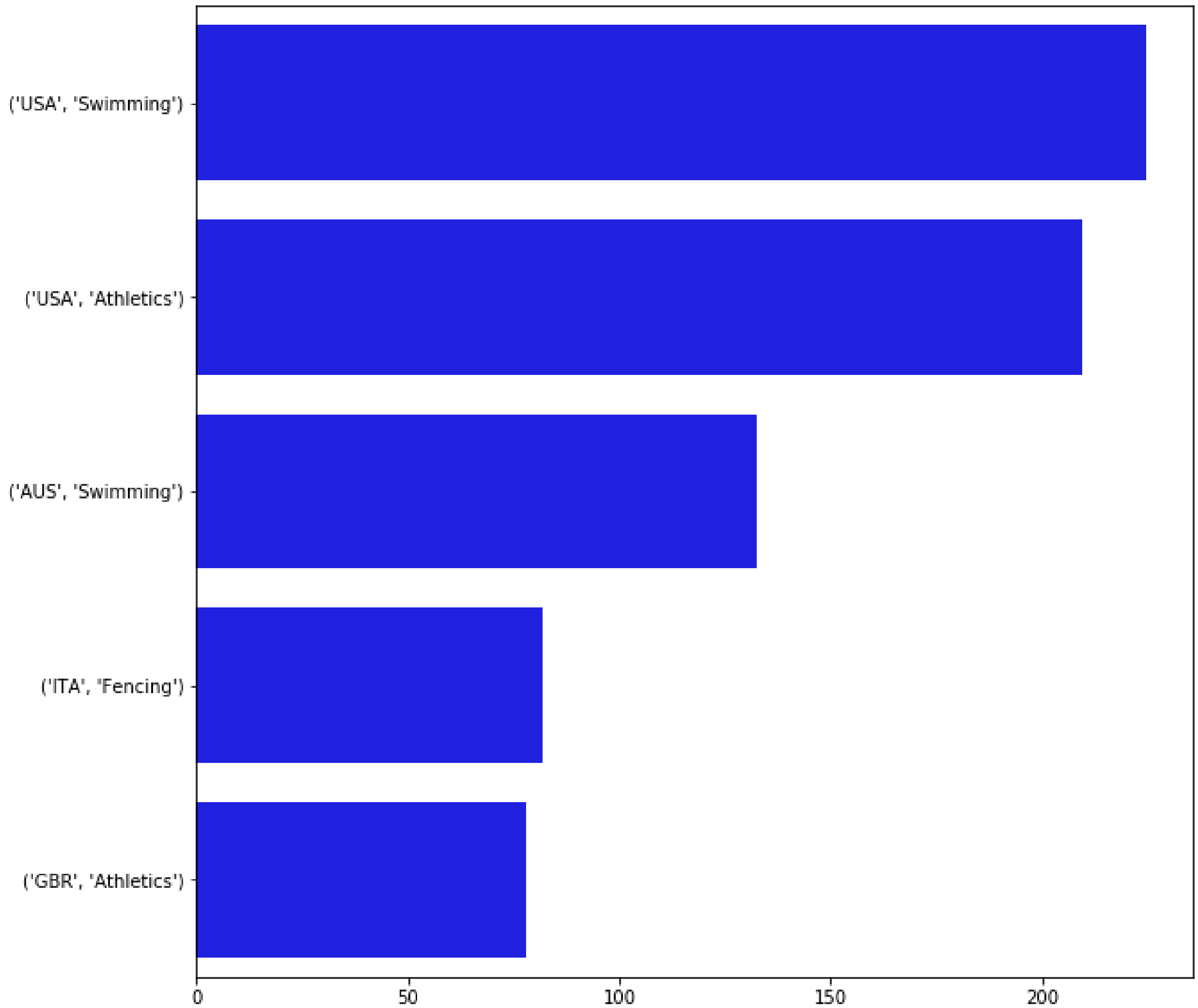
By sorting on the values in this matrix, we find the combination of region and sport that are most extreme, meaning either much more medals then expected, or much less medals than expected.

```
medals_diff_df = medals_sport_diff_df.stack().sort_values(ascending=False)
medals_diff_df.head()
```

```
NOC  Sport
USA  Swimming     224.472926
     Athletics    209.169828
AUS  Swimming     132.374235
ITA  Fencing       82.005643
GBR  Athletics     78.193060
dtype: float64
```

So now we know that USA has aboutn 224 more medals in Swimming than expected, we could put this in a bar chart

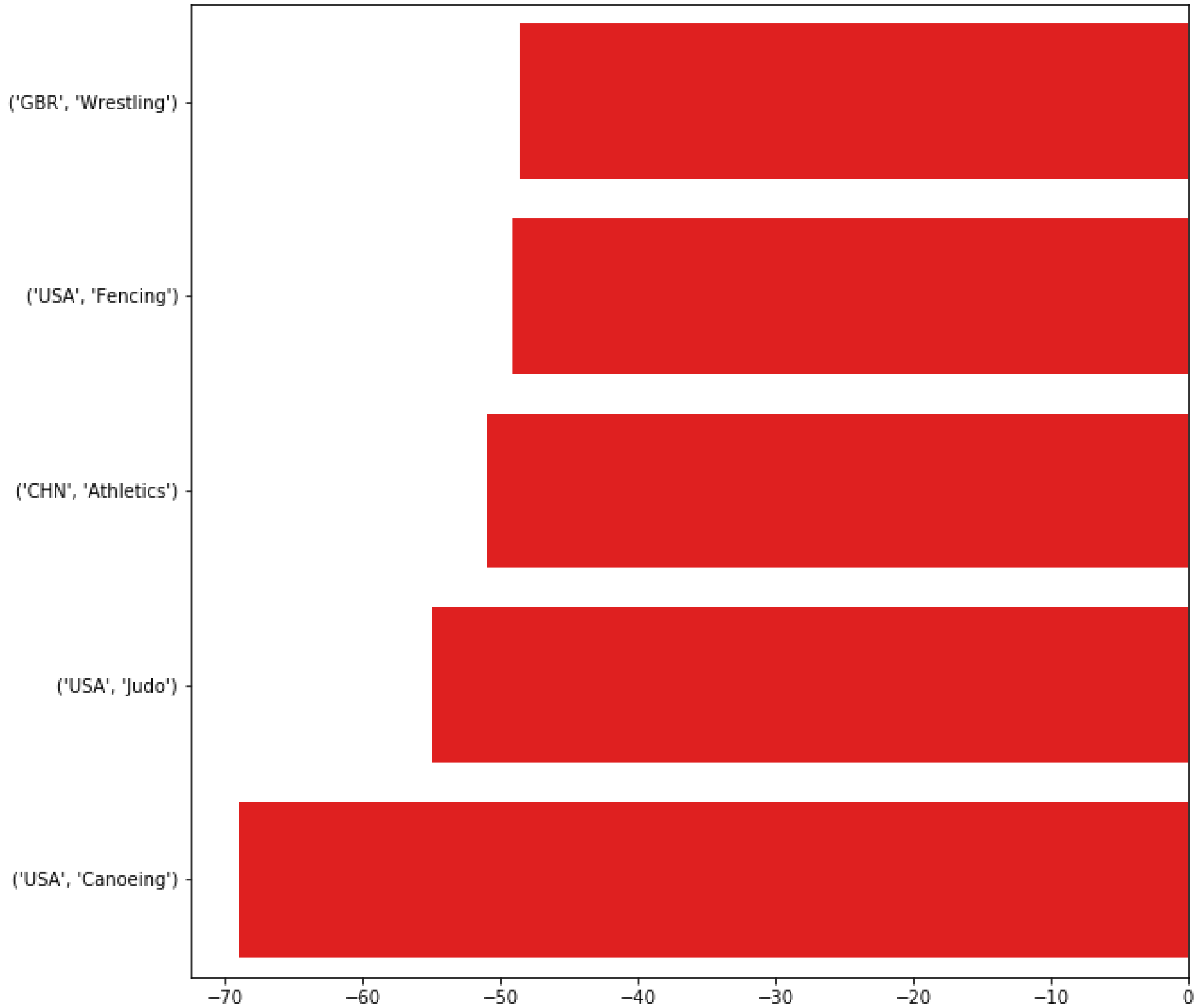
```
sns.barplot(x=medals_diff_df.head(), y=medals_diff_df.head().index.values, color='b')
plt.show()
```



This reveals that USA seems to be investing a lot in Swimming or Athletics sports, which are by coincidence sports that have the most medals. You could argue that due to the cold war show-off they have fallen prey to the cobra effect where they used the amount of medals they could get as a target instead of a measure of performance, shifting them towards sports where more medals can be obtained.

Anyway, the same analysis can be done for the worst combinations.

```
sns.barplot(x=medals_diff_df.tail(), y=medals_diff_df.tail().index.values, color='r')
plt.show()
```



This analysis can also be performed on Country level, here we see that Belgium is good at

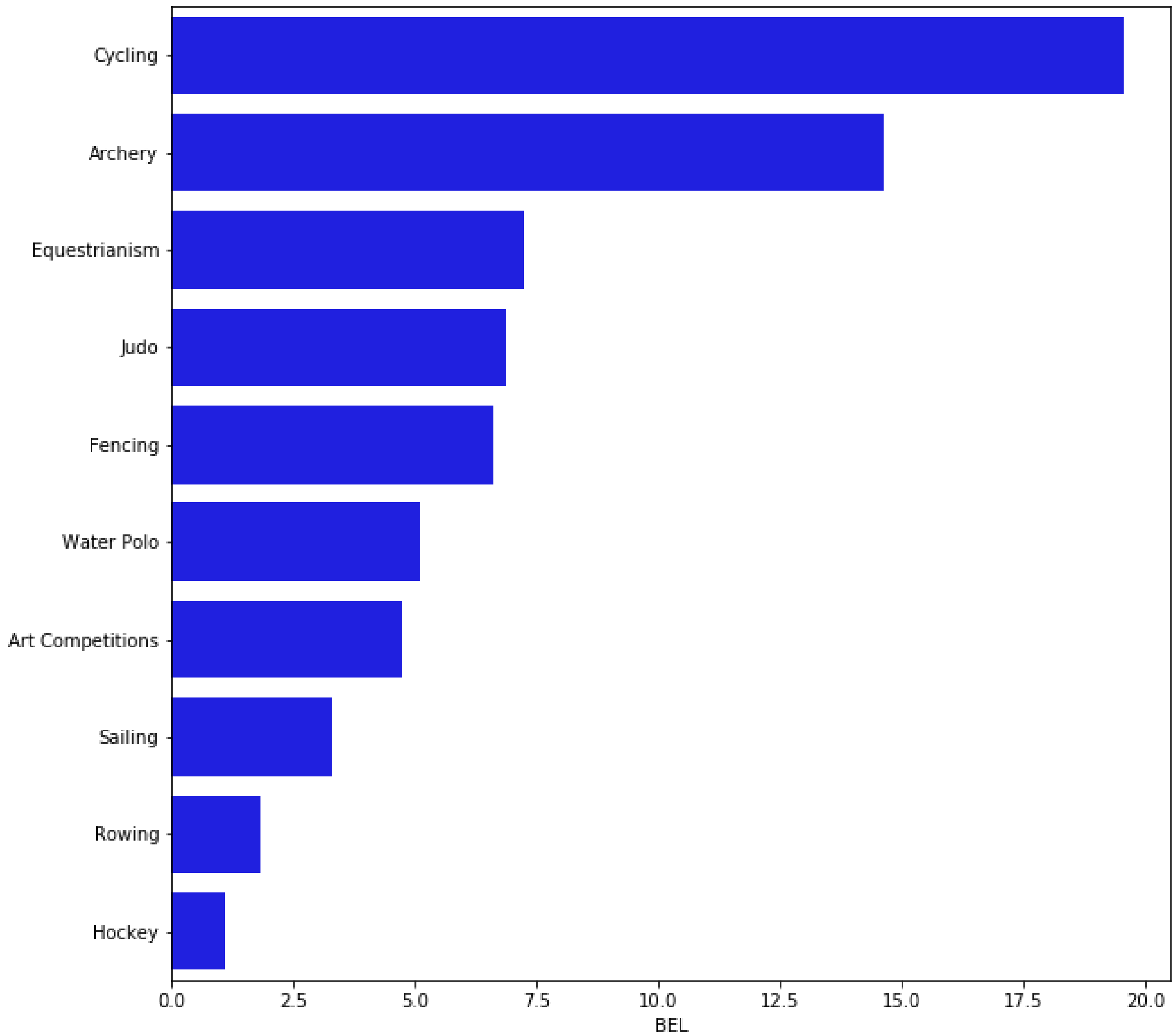
```
medals_sport_diff_df.loc['BEL']
```

```
Sport
Aeronautics          -0.009425
Alpine Skiing        -3.326990
Alpinism             -0.056549
Archery              14.614540
Art Competitions      4.737063
                        ...
Tug-Of-War            0.858626
Volleyball           -0.791692
Water Polo            5.104634
Weightlifting        -2.532417
Wrestling            -7.856525
Name: BEL, Length: 66, dtype: float64
```

And we can put this in the same type of barchart to make it comparible with the previous chart

```
sns.barplot(x=medals_sport_diff_df.loc['BEL'].sort_values(ascending=False).head(10),
↪y=medals_sport_diff_df.loc['BEL'].sort_values(ascending=False).head(10).index.
↪astype('str').values, color='b')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba766ca1c0>
```



37.6.2 Athlete attributes

We also investigated athlete specific attributes, to refresh our memory a printout of how the dataset looks

```
df = athlete_events.drop_duplicates(subset=['Name', 'Age', 'NOC'])
df = df.dropna(subset=['Age', 'Height', 'Weight'])[['Sex', 'Sport', 'Medal', 'Age',
↪'Height', 'Weight']].reset_index(drop=True)
df.head()
```

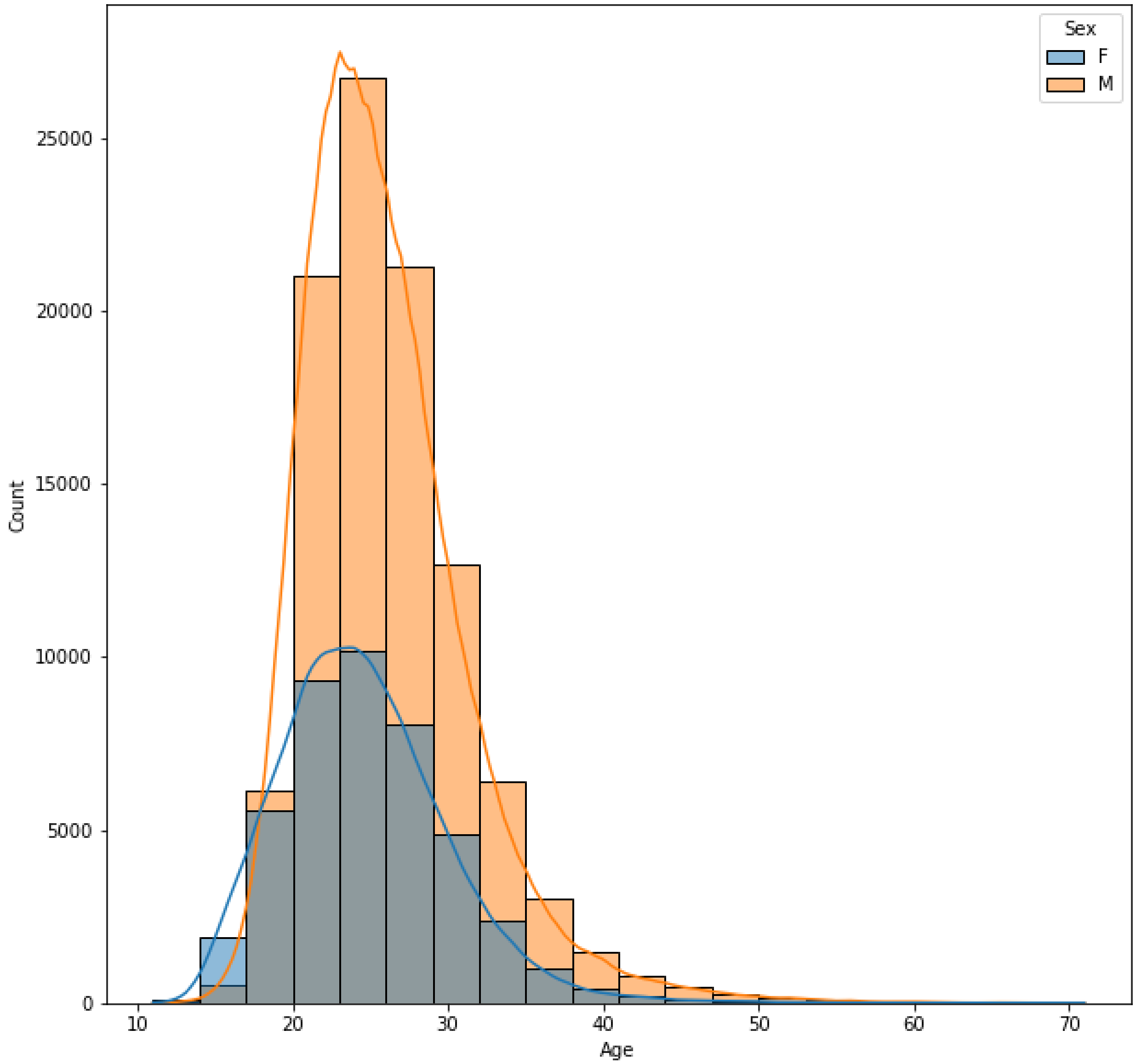
```
  Sex          Sport Medal   Age  Height  Weight
0   M     Basketball   NaN  24.0   180.0    80.0
1   M           Judo   NaN  23.0   170.0    60.0
2   F  Speed Skating   NaN  21.0   185.0    82.0
3   F  Speed Skating   NaN  25.0   185.0    82.0
4   F  Speed Skating   NaN  27.0   185.0    82.0
```

I kept features such as gender, Sport, … as these were attributes on which the physical appearance was different, we can use these features to group our athletes and visualise the distribution with a histogram.

```
sns.histplot(data = df, x='Age', hue='Sex', bins=20, kde=True)
```

```
/usr/lib/python3/dist-packages/matplotlib/cbook/__init__.py:1402: FutureWarning:
↪Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will
↪be removed in a future version.  Convert to a numpy array before indexing instead.
  ndim = x[:, None].ndim
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:276: FutureWarning: Support
↪for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be
↪removed in a future version.  Convert to a numpy array before indexing instead.
  x = x[:, np.newaxis]
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:278: FutureWarning: Support
↪for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be
↪removed in a future version.  Convert to a numpy array before indexing instead.
  y = y[:, np.newaxis]
```

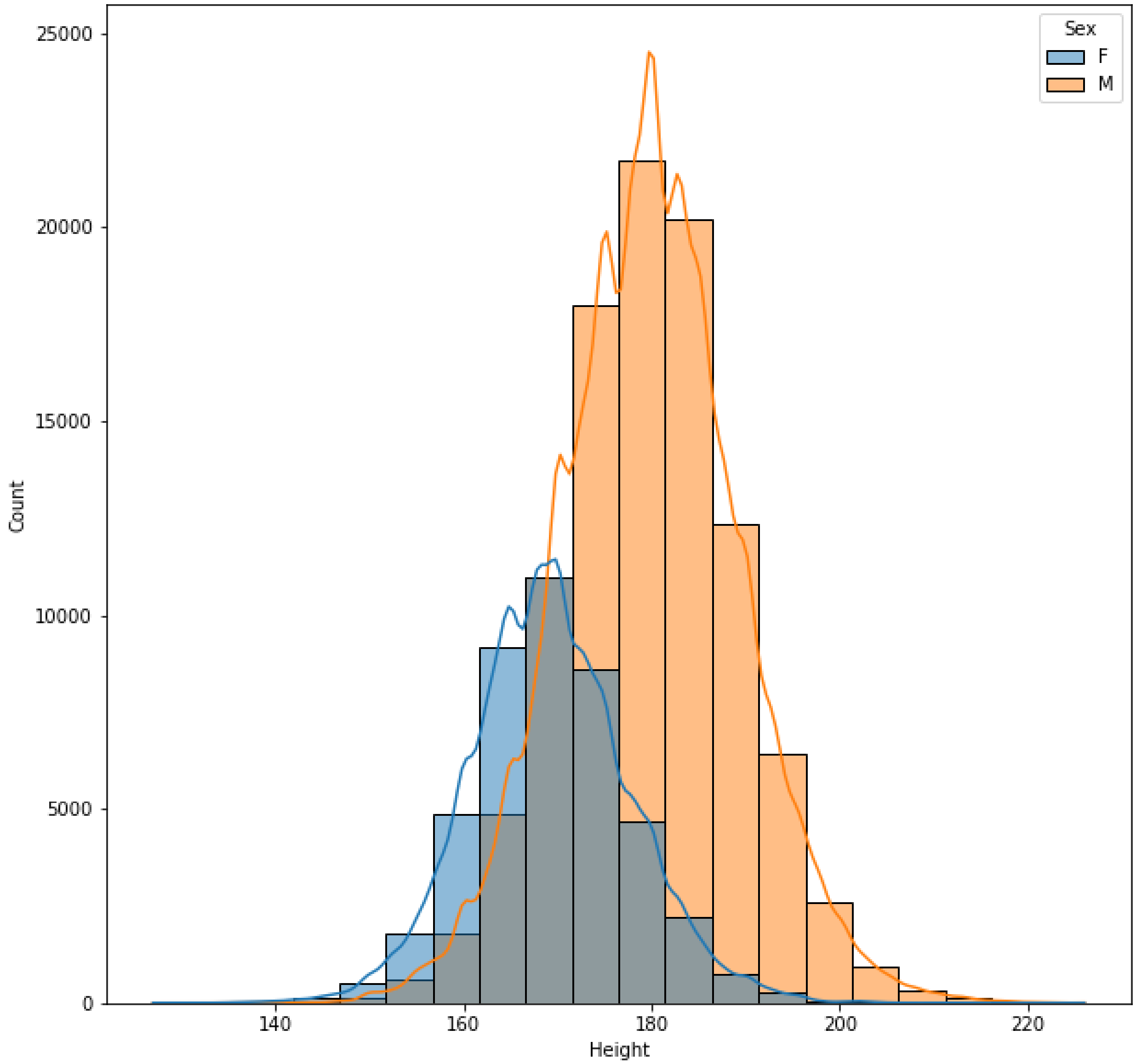
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba75833190>
```




```
sns.histplot(data = df, x='Height', hue='Sex', bins=20, kde=True)
```

```
/usr/lib/python3/dist-packages/matplotlib/cbook/__init__.py:1402: FutureWarning:␣
↪Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will␣
↪be removed in a future version.  Convert to a numpy array before indexing instead.
  ndim = x[:, None].ndim
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:276: FutureWarning: Support␣
↪for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be␣
↪removed in a future version.  Convert to a numpy array before indexing instead.
  x = x[:, np.newaxis]
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:278: FutureWarning: Support␣
↪for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be␣
↪removed in a future version.  Convert to a numpy array before indexing instead.
  y = y[:, np.newaxis]
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba75824820>
```



For gender, the difference in age is not that appearent, yet the shift in height is, women are in general less tall as men.

When grouping per sport we saw significant differences.

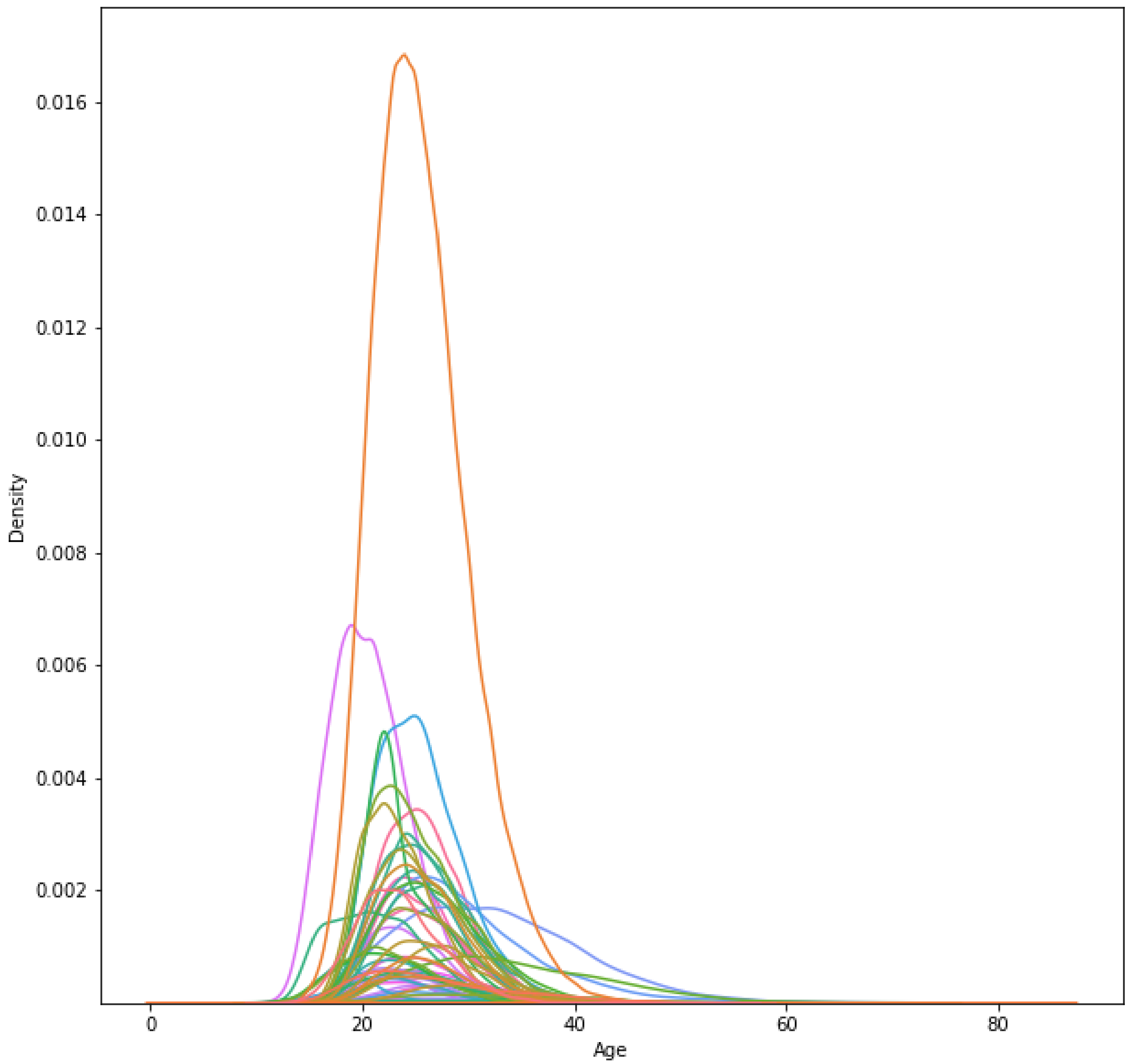
```
ax = sns.kdeplot(data=df, x='Age', hue='Sport')
plt.legend().remove()
```

```
/home/lorenzf/.local/lib/python3.8/site-packages/seaborn/distributions.py:316:
↪UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_
↪singular=False` to disable this warning.
  warnings.warn(msg, UserWarning)
/usr/lib/python3/dist-packages/matplotlib/cbook/__init__.py:1402: FutureWarning:
↪Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will
↪be removed in a future version.  Convert to a numpy array before indexing instead.
  ndim = x[:, None].ndim
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:276: FutureWarning: Support
↪for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be
↪removed in a future version.  Convert to a numpy array before indexing instead.
```

(continues on next page)

(continued from previous page)

```
  x = x[:, np.newaxis]
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:278: FutureWarning: Support␣
↪for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be␣
↪removed in a future version.  Convert to a numpy array before indexing instead.
  y = y[:, np.newaxis]
No handles with labels found to put in legend.
```



If we put all sports like this in a distribution plot, it becomes a big mess, I had to remove the legend as there are a lot of sports and the bins all overlap. It seems not a good idea to make such a plot.

For medals we only have 3 different groups.

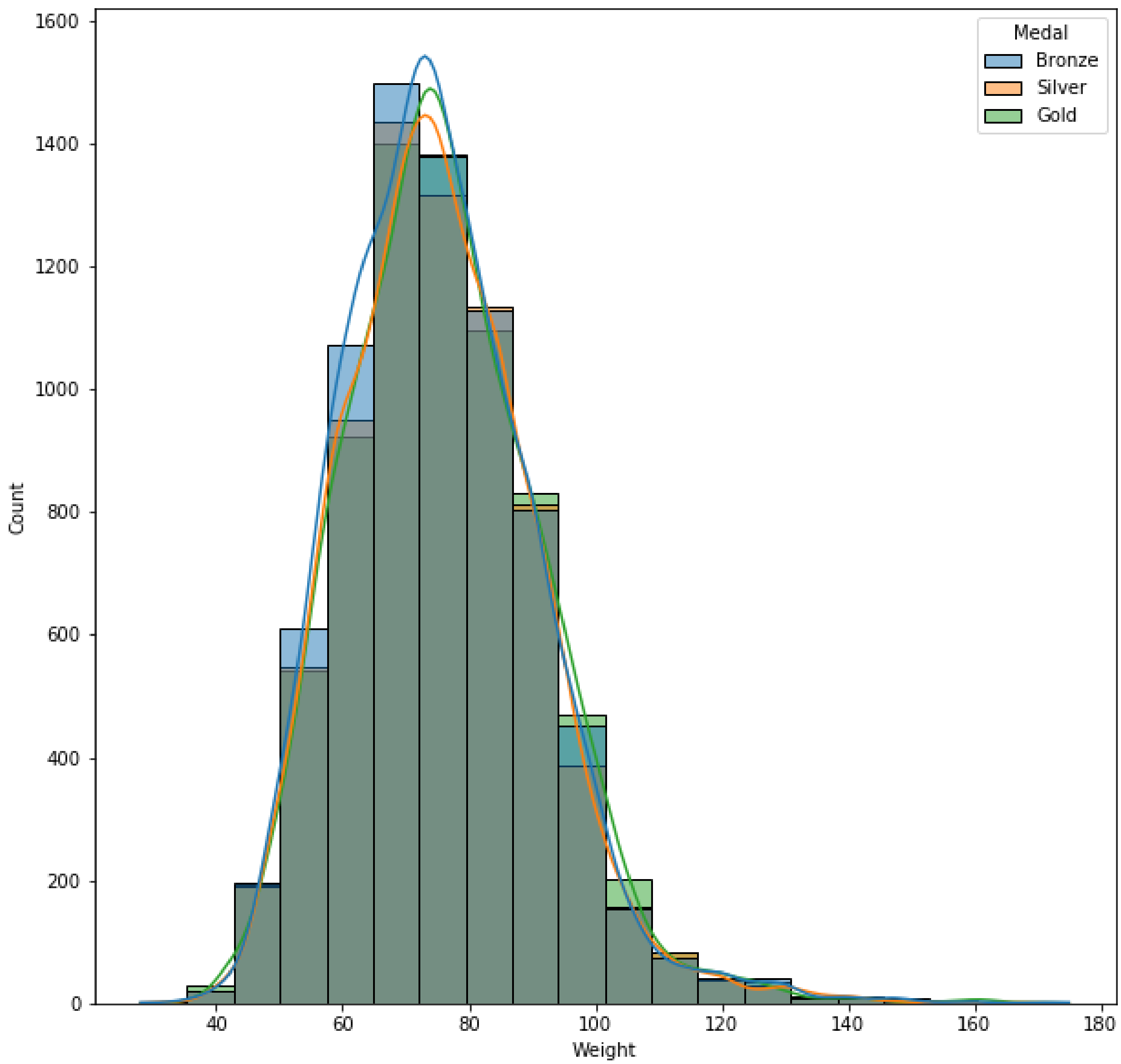
```
ax = sns.histplot(data=df, x='Weight', hue='Medal', bins=20, kde=True)
```

```
/usr/lib/python3/dist-packages/matplotlib/cbook/__init__.py:1402: FutureWarning:␣
↪Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will␣
↪be removed in a future version.  Convert to a numpy array before indexing instead.
```

(continues on next page)

(continued from previous page)

```
  ndim = x[:, None].ndim
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:276: FutureWarning: Support
↪for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be
↪removed in a future version.  Convert to a numpy array before indexing instead.
  x = x[:, np.newaxis]
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:278: FutureWarning: Support
↪for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be
↪removed in a future version.  Convert to a numpy array before indexing instead.
  y = y[:, np.newaxis]
```

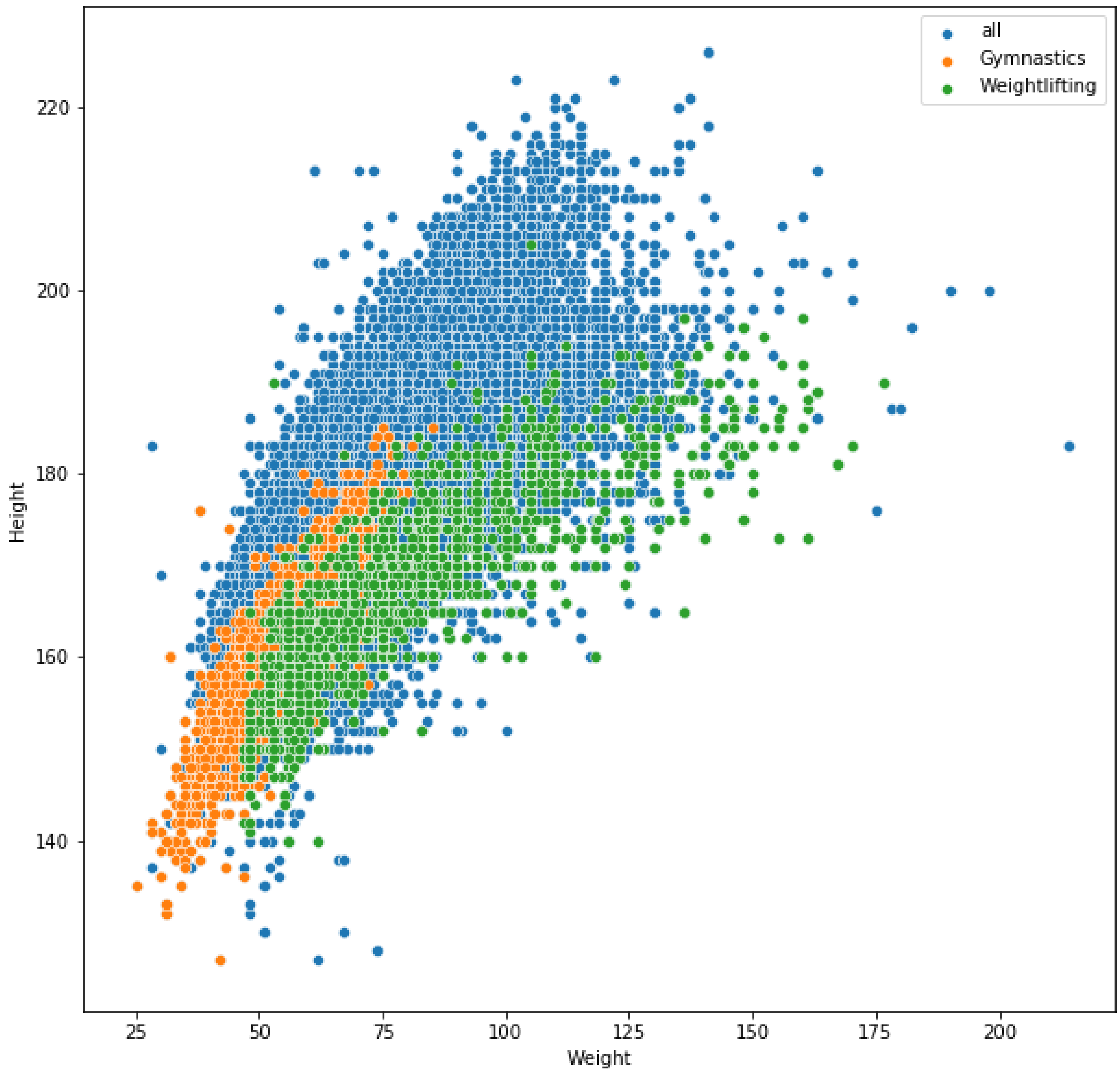


Obviously for each ceremony we have 1 gold, 1 silver and 1 bronze, so distributions are equal in size. We saw erlier that the groups do not have significant differences and this is confirmed with the histogram, although you can see some small differences that perhaps show a pattern?

Lastly I would like to add another dimension to the plots by using scatterplots, it will be messy but creates a new perspective. For the scatter plot I would first plot all athlete's height and weight (you could add lines of equal BMI here) and superpose in other colors subgroups of athletes based on groups. Here I use the sport to show all athletes, gymnastics and weightlifting.

```
sns.scatterplot(data=df, x='Weight', y='Height', label='all')
sns.scatterplot(data=df[df.Sport=='Gymnastics'], x='Weight', y='Height', label=
↪'Gymnastics')
sns.scatterplot(data=df[df.Sport=='Weightlifting'], x='Weight', y='Height', label=
↪'Weightlifting')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba75061fd0>
```



you can clearly see how gymnastics are the smallest athletes and whilst weightlifting are also fairly small, they have a much higher weight, as they need muscles to perform their sport.

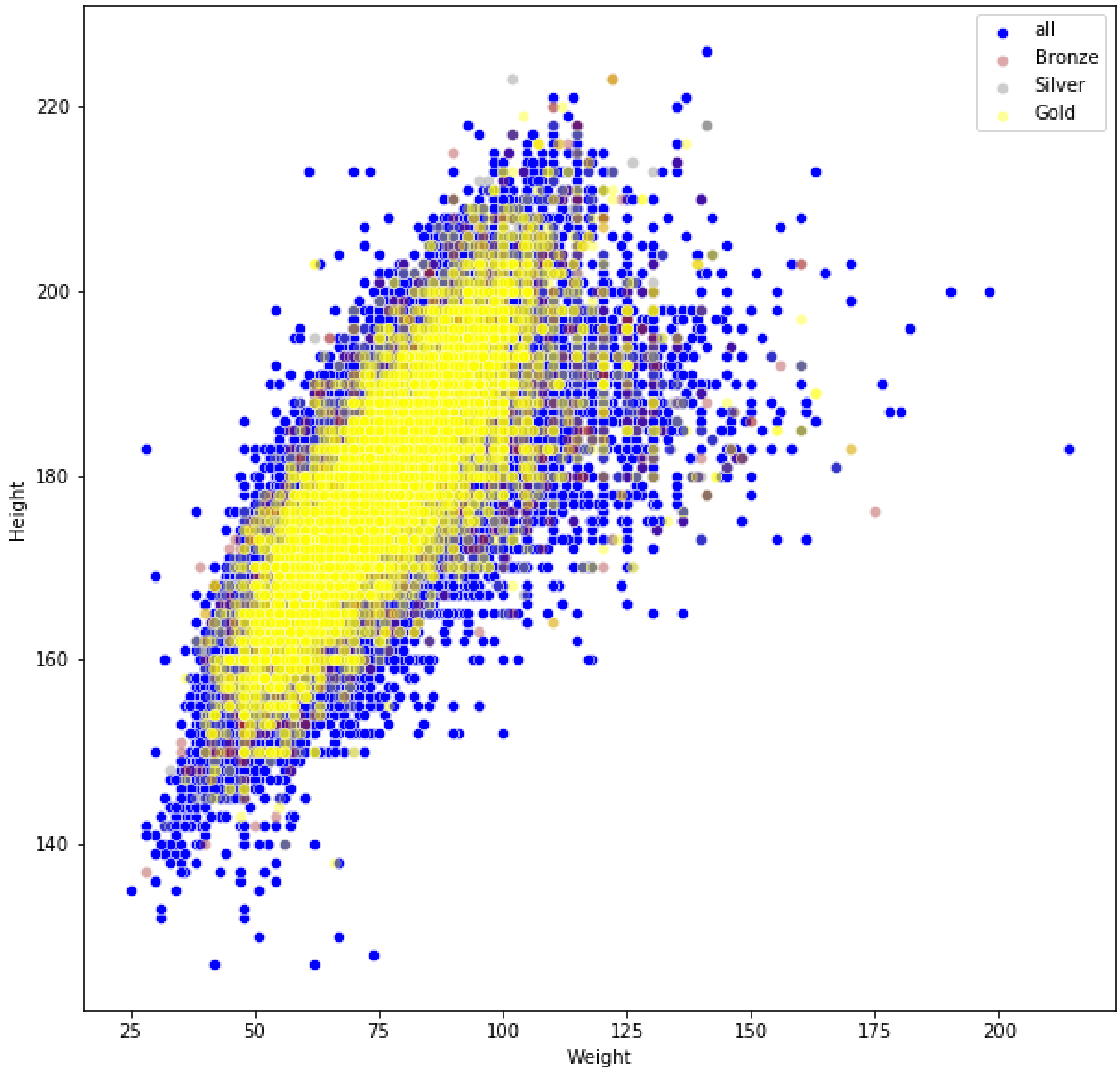
```
sns.scatterplot(data=df[df.Medal.isna()], x='Weight', y='Height', label='all', color=
↪'blue')
sns.scatterplot(data=df[df.Medal=='Bronze'], x='Weight', y='Height', label='Bronze',␣
↪color='brown', alpha=0.4)
```

(continues on next page)

(continued from previous page)

```
sns.scatterplot(data=df[df.Medal=='Silver'], x='Weight', y='Height', label='Silver',␣
↪color='grey', alpha=0.4)
sns.scatterplot(data=df[df.Medal=='Gold'], x='Weight', y='Height', label='Gold',␣
↪color='yellow', alpha=0.4)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba74f313d0>
```



Looking at this graph we can see that while there is no difference for athlete that achieves different types of medals, there is a clear area in which you should be in order to be a medal winner, outside that area clearly dimishishes your chances.

Also there seems to be an athlete that is more than 200kgs?

```
athlete_events[athlete_events.Weight==athlete_events.Weight.max()]
```

```
                    Name Sex   Age  Height  Weight  Team  NOC        Games  \
ID
12177  Ricardo Blas, Jr.   M  21.0   183.0   214.0  Guam  GUM  2008 Summer
12177  Ricardo Blas, Jr.   M  25.0   183.0   214.0  Guam  GUM  2012 Summer

       Year  Season     City Sport                   Event Medal
ID
12177  2008  Summer  Beijing  Judo  Judo Men's Heavyweight   NaN
12177  2012  Summer   London  Judo  Judo Men's Heavyweight   NaN
```

37.7 Summary

- Best performing depends on metric
- Some countries focus on different sports due to multiple reasons (# medals, heritage, …)
- Your sport and physical attributes are related, there is a ideal weight and height

CHAPTER

THIRTYEIGHT

CASE STUDY: USER SURVEY

In this case study we figure out how to analyse the responses from a user survey form kaggle

The case study is divided into several parts:

- Goals
- Parsing
- Preparation (cleaning)
- Processing
- Exploration
- Visualization
- Conclusion

38.1 Goals

In this section we define questions that will be our guideline througout the case study

- What influences salary?
- Can we deduce common skills for job titles?
- Do higher paid jobs spend time differently?
- Important: education or experience?

We'll (try to) keep these question in mind when performing the case study.

38.2 Parsing

we start out by importing all necessary libraries

```
import os
import json
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats
import matplotlib.pyplot as plt
from IPython.display import set_matplotlib_formats
```

(continues on next page)

(continued from previous page)

```
%matplotlib inline
set_matplotlib_formats('svg')
plt.rcParams['figure.figsize'] = [10, 10]
```

```
/tmp/ipykernel_9037/2151882340.py:10: DeprecationWarning: `set_matplotlib_formats` is␣
↪deprecated since IPython 7.23, directly use `matplotlib_inline.backend_inline.set_
↪matplotlib_formats()`
  set_matplotlib_formats('svg')
```

in order to download datasets from kaggle, we need an API key to access their API, we'll make that here

```
if not os.path.exists(os.path.expanduser('~/.kaggle')):
    os.mkdir(os.path.expanduser('~/.kaggle'))

with open(os.path.expanduser('~/.kaggle/kaggle.json'), 'w') as f:
    json.dump(
        {
            "username":"lorenzf",
            "key":"7a44a9e99b27e796177d793a3d85b8cf"
        }
        , f)
```

now we can import kaggle too and download the datasets

```
import kaggle
kaggle.api.dataset_download_files(dataset='kaggle/kaggle-survey-2018', path='./data',␣
↪unzip=True)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_9037/707767778.py in <module>
----> 1 import kaggle
      2 kaggle.api.dataset_download_files(dataset='kaggle/kaggle-survey-2018', path='.
↪/data', unzip=True)

ModuleNotFoundError: No module named 'kaggle'
```

the csv files are now in the './data' folder, we can now read them using pandas, here is the list of all csv files in our folder

```
os.listdir('./data')
```

```
['WA_Fn-UseC_-Telco-Customer-Churn.csv',
 'API_NY.GDP.PCAP.CD_DS2_en_csv_v2_3358201.csv',
 'noc_regions.csv',
 'freeFormResponses.csv',
 'SurveySchema.csv',
 'jester_ratings.csv',
 'multipleChoiceResponses.csv',
 'one-million-reddit-jokes.csv',
 'jester_items.csv',
 'athlete_events.csv',
 'API_SP.POP.TOTL_DS2_en_csv_v2_3358390.csv']
```

The file of our interest is 'multipleChoiceResponses.csv', it contains the multiple choice responses of our subjects. Let's print out the top 5 events.

```
choice_df = pd.read_csv('./data/multipleChoiceResponses.csv')
print('shape: ' + str(choice_df.shape))
choice_df.head()
```

```
shape: (23860, 395)
```

```
/home/lorenzf/.local/lib/python3.8/site-packages/IPython/core/interactiveshell.
↪py:3441: DtypeWarning: Columns (0,2,8,10,21,23,24,25,26,27,28,44,56,64,83,85,87,107,
↪109,123,125,150,157,172,174,194,210,218,219,223,246,249,262,264,276,277,278,279,280,
↪281,282,283,284,285,286,287,288,289,290,304,306,325,326,329,341,368,371,384,385,389,
↪390,391,393,394) have mixed types.Specify dtype option on import or set low_
↪memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

```
  Time from Start to Finish (seconds)                                   Q1  \
0              Duration (in seconds)  What is your gender? - Selected Choice
1                                710                                Female
2                                434                                  Male
3                                718                                Female
4                                621                                  Male

                                  Q1_OTHER_TEXT  \
0  What is your gender? - Prefer to self-describe...
1                                             -1
2                                             -1
3                                             -1
4                                             -1

                         Q2                                          Q3  \
0  What is your age (# years)?  In which country do you currently reside?
1                     45-49                    United States of America
2                     30-34                                   Indonesia
3                     30-34                    United States of America
4                     35-39                    United States of America

                                                  Q4  \
0  What is the highest level of formal education ...
1                                    Doctoral degree
2                                  Bachelor's degree
3                                    Master's degree
4                                    Master's degree

                                                  Q5  \
0  Which best describes your undergraduate major?...
1                                              Other
2                     Engineering (non-computer focused)
3        Computer science (software engineering, etc.)
4  Social sciences (anthropology, psychology, soc...

                                                  Q6  \
0  Select the title most similar to your current ...
1                                         Consultant
2                                              Other
3                                     Data Scientist
4                                       Not employed
```

(continues on next page)

(continued from previous page)

```
                                       Q6_OTHER_TEXT  \
0  Select the title most similar to your current ...
1                                                 -1
2                                                  0
3                                                 -1
4                                                 -1

                                                  Q7  ...  \
0  In what industry is your current employer/cont...  ...
1                                              Other  ...
2                          Manufacturing/Fabrication  ...
3                                     I am a student  ...
4                                                NaN  ...

                                      Q49_OTHER_TEXT  \
0  What tools and methods do you use to make your...
1                                                 -1
2                                                 -1
3                                                 -1
4                                                 -1

                                         Q50_Part_1  \
0  What barriers prevent you from making your wor...
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN

                                         Q50_Part_2  \
0  What barriers prevent you from making your wor...
1                                                NaN
2                                                NaN
3                                 Too time-consuming
4                                                NaN

                                         Q50_Part_3  \
0  What barriers prevent you from making your wor...
1                                                NaN
2                                                NaN
3                                                NaN
4               Requires too much technical knowledge

                                         Q50_Part_4  \
0  What barriers prevent you from making your wor...
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN

                                         Q50_Part_5  \
0  What barriers prevent you from making your wor...
1                                                NaN
2                                                NaN
3                                                NaN
4              Not enough incentives to share my work

                                         Q50_Part_6  \
```

(continues on next page)

(continued from previous page)

```
0  What barriers prevent you from making your wor...
1                                                 NaN
2                                                 NaN
3                                                 NaN
4                                                 NaN

                                          Q50_Part_7  \
0  What barriers prevent you from making your wor...
1                                                 NaN
2                                                 NaN
3                                                 NaN
4                                                 NaN

                                          Q50_Part_8  \
0  What barriers prevent you from making your wor...
1                                                 NaN
2                                                 NaN
3                                                 NaN
4                                                 NaN

                                      Q50_OTHER_TEXT
0  What barriers prevent you from making your wor...
1                                                  -1
2                                                  -1
3                                                  -1
4                                                  -1

[5 rows x 395 columns]
```

```
free_form_df = pd.read_csv('./data/freeFormResponses.csv')
print('shape: ' + str(free_form_df.shape))
free_form_df.head()
```

```
shape: (23860, 35)
```

```
/home/lorenzf/.local/lib/python3.8/site-packages/IPython/core/interactiveshell.
↪py:3441: DtypeWarning: Columns (25) have mixed types.Specify dtype option on import␣
↪or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

```
                                      Q11_OTHER_TEXT  \
0  Select any activities that make up an importan...
1                                                 NaN
2                                                 NaN
3                                                 NaN
4                                                 NaN

                                      Q12_OTHER_TEXT  \
0  What is the primary tool that you use at work ...
1                                                 NaN
2                                                 NaN
3                                                 NaN
4                                                 NaN

                                     Q12_Part_1_TEXT  \
```

(continues on next page)

(continued from previous page)

```
0  What is the primary tool that you use at work ...
1                                              NaN
2                                              NaN
3                                              NaN
4                                              NaN

                                  Q12_Part_2_TEXT  \
0  What is the primary tool that you use at work ...
1                                              NaN
2                                              NaN
3                                              NaN
4                                              NaN

                                  Q12_Part_3_TEXT  \
0  What is the primary tool that you use at work ...
1                                              NaN
2                                              NaN
3                                              NaN
4                                              NaN

                                  Q12_Part_4_TEXT  \
0  What is the primary tool that you use at work ...
1           Jupyter Notebooks, Pycharm, Intelijidea
2                                              NaN
3                                         anaconda
4                                              NaN

                                  Q12_Part_5_TEXT  \
0  What is the primary tool that you use at work ...
1                                              NaN
2                                              NaN
3                                              NaN
4                                              NaN

                                   Q13_OTHER_TEXT  \
0  Which of the following integrated development ...
1                                              NaN
2                                              NaN
3                                              NaN
4                                              NaN

                                   Q14_OTHER_TEXT  \
0  Which of the following hosted notebooks have y...
1                                              NaN
2                                              NaN
3                                              NaN
4                                              NaN

                                   Q15_OTHER_TEXT  ...  \
0  Which of the following cloud computing service...  ...
1                                              NaN  ...
2                                              NaN  ...
3                                              NaN  ...
4                                              NaN  ...

                                   Q34_OTHER_TEXT  \
0  During a typical data science project at work ...
```

(continues on next page)

(continued from previous page)

```
1                                                0.0
2                                                NaN
3                                                  0
4                                                NaN

                                     Q35_OTHER_TEXT  \
0  What percentage of your current machine learni...
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN

                                     Q36_OTHER_TEXT  \
0  On which online platforms have you begun or co...
1                                        mlcourse.ai
2                                                NaN
3                                                NaN
4                                                NaN

                                     Q37_OTHER_TEXT  \
0  On which online platform have you spent the mo...
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN

                                     Q38_OTHER_TEXT  \
0  Who/what are your favorite media sources that ...
1                                             ods.ai
2                                                NaN
3                                                NaN
4                                                NaN

                                     Q42_OTHER_TEXT  \
0  What metrics do you or your organization use t...
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN

                                     Q49_OTHER_TEXT  \
0  What tools and methods do you use to make your...
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN

                                     Q50_OTHER_TEXT  \
0  What barriers prevent you from making your wor...
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN

                                      Q6_OTHER_TEXT  \
0  Select the title most similar to your current ...
1                                                NaN
```

(continues on next page)

(continued from previous page)

```
2                                                   NaN
3                                                   NaN
4                                                   NaN

                                          Q7_OTHER_TEXT
0  In what industry is your current employer/cont...
1                                                   NaN
2                                                   NaN
3                                                   NaN
4                                                   NaN

[5 rows x 35 columns]
```

I saw that the first row of our choice dataframe contains the questions, to let's extract that.

```
questions = choice_df.iloc[0]
choice_df = choice_df.drop(0)
```

```
questions.head(20)
```

```
Time from Start to Finish (seconds)                                Duration (in␣
↪seconds)
Q1                                                  What is your gender? - Selected␣
↪Choice
Q1_OTHER_TEXT                           What is your gender? - Prefer to self-describe.
↪..
Q2                                                              What is your age (#␣
↪years)?
Q3                                                In which country do you currently␣
↪reside?
Q4                                      What is the highest level of formal education .
↪..
Q5                                      Which best describes your undergraduate major?.
↪..
Q6                                      Select the title most similar to your current .
↪..
Q6_OTHER_TEXT                           Select the title most similar to your current .
↪..
Q7                                      In what industry is your current employer/cont.
↪..
Q7_OTHER_TEXT                           In what industry is your current employer/cont.
↪..
Q8                                      How many years of experience do you have in yo.
↪..
Q9                                      What is your current yearly compensation (appr.
↪..
Q10                                     Does your current employer incorporate machine.
↪..
Q11_Part_1                              Select any activities that make up an importan.
↪..
Q11_Part_2                              Select any activities that make up an importan.
↪..
Q11_Part_3                              Select any activities that make up an importan.
↪..
Q11_Part_4                              Select any activities that make up an importan.
↪..
```

(continues on next page)

(continued from previous page)

```
Q11_Part_5                          Select any activities that make up an importan.
↪..
Q11_Part_6                          Select any activities that make up an importan.
↪..
Name: 0, dtype: object
```

38.3 Preparation

here we perform tasks to prepare the data in a more pleasing format.

38.3.1 Data Types

Before we do anything with our data, it is good to see if our data types are in order

```
choice_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23859 entries, 1 to 23859
Columns: 395 entries, Time from Start to Finish (seconds) to Q50_OTHER_TEXT
dtypes: object(395)
memory usage: 72.1+ MB
```

Seems there are to many too show, so we have to do some manual work, The first 10 questions seem to be about personal info, where the first one is about gender

```
print(questions.Q1)
choice_df.Q1.value_counts()
```

```
What is your gender? - Selected Choice
```

```
Male                       19430
Female                      4010
Prefer not to say            340
Prefer to self-describe       79
Name: Q1, dtype: int64
```

```
print(questions.Q1_OTHER_TEXT)
choice_df.Q1_OTHER_TEXT.unique()
```

```
What is your gender? - Prefer to self-describe - Text
```

```
array(['-1', '2', '3', '4', '5', '6', -1, 7, 8, 9, 10, 11, 12, 13, 14, 15,
       16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 4,
       32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
       49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
       66, 67], dtype=object)
```

Hmm the self-describe seems to already been encoded, as there are so many different answers I would opt to ignore those results as they only take up 79 answers of all 24k. For the second question I am going to convert it to an ordinal value, this way we know the order of the categories.

```
choice_df.Q2 = choice_df.Q2.astype(pd.api.types.CategoricalDtype(categories=['18-21',
↪'22-24', '25-29', '30-34', '35-39', '40-44', '45-49', '50-54', '55-59', '60-69',
↪'70-79', '80+'], ordered=True))
print(questions.Q2)
choice_df.Q2
```

```
What is your age (# years)?
```

```
1        45-49
2        30-34
3        30-34
4        35-39
5        22-24
         ...
23855    45-49
23856    25-29
23857    22-24
23858    25-29
23859    25-29
Name: Q2, Length: 23859, dtype: category
Categories (12, object): ['18-21' < '22-24' < '25-29' < '30-34' ... '55-59' < '60-69'
↪< '70-79' < '80+']
```

Next we have a few very important questions that signify the situation of each user in our survey. I chose for nominal categories as I don't want to be biased.

```
print(questions.Q6)
choice_df.Q6.value_counts()
```

```
Select the title most similar to your current role (or most recent title if retired):␣
↪- Selected Choice
```

```
Student                    5253
Data Scientist             4137
Software Engineer          3130
Data Analyst               1922
Other                      1322
Research Scientist         1189
Not employed                842
Consultant                  785
Business Analyst            772
Data Engineer               737
Research Assistant          600
Manager                     590
Product/Project Manager     428
Chief Officer               360
Statistician                237
DBA/Database Engineer       145
Developer Advocate          117
Marketing Analyst           115
Salesperson                 102
Principal Investigator       97
Data Journalist              20
Name: Q6, dtype: int64
```

```
print(questions[['Q3', 'Q4', 'Q5', 'Q6', 'Q7']])
choice_df[['Q3', 'Q4', 'Q5', 'Q6', 'Q7']] = choice_df[['Q3', 'Q4', 'Q5', 'Q6', 'Q7']].
↪astype('category')
```

```
Q3             In which country do you currently reside?
Q4    What is the highest level of formal education ...
Q5    Which best describes your undergraduate major?...
Q6    Select the title most similar to your current ...
Q7    In what industry is your current employer/cont...
Name: 0, dtype: object
```

Question 8 is about experience, or as they call it tenure. Not as a numerical value but in categories, so again I create an ordinal category from it.

```
print(questions.Q8)
choice_df.Q8.value_counts()
```

```
How many years of experience do you have in your current role?
```

```
0-1      5898
1-2      3745
2-3      2577
5-10     2524
3-4      1751
10-15    1512
4-5      1488
15-20     854
20-25     384
30 +      197
25-30     171
Name: Q8, dtype: int64
```

```
choice_df.Q8 = choice_df.Q8.astype(pd.api.types.CategoricalDtype(categories=['0-1',
↪'1-2', '2-3', '3-4', '4-5', '5-10', '10-15', '15-20', '20-25', '25-30', '30 +'],␣
↪ordered=True))
print(questions.Q8)
choice_df.Q8
```

```
How many years of experience do you have in your current role?
```

```
1          NaN
2         5-10
3          0-1
4          NaN
5          0-1
         ...
23855     5-10
23856      NaN
23857      0-1
23858      NaN
23859      NaN
Name: Q8, Length: 23859, dtype: category
Categories (11, object): ['0-1' < '1-2' < '2-3' < '3-4' ... '15-20' < '20-25' < '25-30
↪' < '30 +']
```

And not to forget we have the salary, again as a category, which is unfortunate since we could have been able to create a more accurate prediction in the end. Here I opt for an ordinal category.

```
choice_df.Q9.value_counts()
```

```
I do not wish to disclose my approximate yearly compensation    4756
0-10,000                                                        4398
10-20,000                                                       1937
20-30,000                                                       1395
30-40,000                                                       1119
40-50,000                                                        965
50-60,000                                                        919
100-125,000                                                      843
60-70,000                                                        729
70-80,000                                                        677
90-100,000                                                       566
125-150,000                                                      533
80-90,000                                                        506
150-200,000                                                      457
200-250,000                                                      172
250-300,000                                                       75
500,000+                                                          63
300-400,000                                                       52
400-500,000                                                       23
Name: Q9, dtype: int64
```

```
choice_df.Q9 = choice_df.Q9.astype(pd.api.types.CategoricalDtype(categories=['0-10,000
↪', '10-20,000', '20-30,000', '30-40,000', '40-50,000', '50-60,000', '60-70,000',
↪'70-80,000', '80-90,000', '90-100,000', '100-125,000', '125-150,000', '150-200,000',
↪ '200-250,000', '250-300,000', '300-400,000', '400-500,000', '500,000+',],␣
↪ordered=True))
choice_df.Q9
```

```
1                NaN
2          10-20,000
3           0-10,000
4                NaN
5           0-10,000
            ...
23855    250-300,000
23856            NaN
23857      10-20,000
23858            NaN
23859            NaN
Name: Q9, Length: 23859, dtype: category
Categories (18, object): ['0-10,000' < '10-20,000' < '20-30,000' < '30-40,000' ...
↪'250-300,000' < '300-400,000' < '400-500,000' < '500,000+']
```

38.3.2 Missing values

for each dataframe we apply a few checks in order to see the quality of data

```
print(100*choice_df.isna().sum().head(20)/choice_df.shape[0])
```

```
Time from Start to Finish (seconds)     0.000000
Q1                                      0.000000
Q1_OTHER_TEXT                           0.000000
Q2                                      0.000000
Q3                                      0.000000
Q4                                      1.764533
Q5                                      3.822457
Q6                                      4.019448
Q6_OTHER_TEXT                           0.000000
Q7                                      9.111866
Q7_OTHER_TEXT                           0.000000
Q8                                     11.559579
Q9                                     35.332579
Q10                                    13.370217
Q11_Part_1                             60.048619
Q11_Part_2                             77.027537
Q11_Part_3                             78.066977
Q11_Part_4                             69.684396
Q11_Part_5                             79.320173
Q11_Part_6                             85.452031
dtype: float64
```

You can clearly see that there are a lot of missing values, for questions 11 and onwards this is just because they did not check that answer on a question, but for 1-10 this is a problem as these are 'mandatory' questions. I have no idea how to fill this in and salary is missing about 35%, pretty disastrous, but this is to be expected with user surveys.

Another problem we have here is trolls, there might have been persons that would just fill this in to mess with our data collection, I thought they might have been funny and answered a high salary.

```
choice_df[choice_df.Q9=='500,000+'].Q2.value_counts()
```

```
25-29    13
35-39    10
30-34     7
80+       7
50-54     6
45-49     5
18-21     4
22-24     4
55-59     4
60-69     3
40-44     0
70-79     0
Name: Q2, dtype: int64
```

you can see there are 13 persons between 25-29 that earn more than 500k annually, which i think is near impossible. Let us see what they are upto.

```
choice_df[(choice_df.Q9=='500,000+') & (choice_df.Q2=='25-29')]
```

```
      Time from Start to Finish (seconds)                        Q1  \
2322                                  561  Prefer to self-describe
8899                                 2116                     Male
12092                                1607                     Male
13468                                5487                     Male
14367                              359331                     Male
15469                                  68                     Male
15825                                  94                   Female
16404                                  78        Prefer not to say
18120                                 281                   Female
20576                                 197  Prefer to self-describe
21122                                 183                     Male
22264                                 520                     Male
22591                                 426                   Female

      Q1_OTHER_TEXT     Q2                        Q3  \
2322              7  25-29                    France
8899             -1  25-29               Philippines
12092            -1  25-29                     China
13468            -1  25-29                     India
14367            -1  25-29                     Kenya
15469            -1  25-29  United States of America
15825            -1  25-29  United States of America
16404            -1  25-29  United States of America
18120            -1  25-29                  Colombia
20576            65  25-29                   Belgium
21122            -1  25-29                     India
22264            -1  25-29                     India
22591            -1  25-29                 Indonesia

                                      Q4  \
2322               I prefer not to answer
8899                    Bachelor’s degree
12092                     Doctoral degree
13468                   Bachelor’s degree
14367                   Bachelor’s degree
15469                     Master’s degree
15825                     Master’s degree
16404  No formal education past high school
18120                     Doctoral degree
20576                     Master’s degree
21122                   Bachelor’s degree
22264                     Master’s degree
22591                   Bachelor’s degree

                                                      Q5                  Q6  \
2322                                               Other               Other
8899                 Engineering (non-computer focused)        Data Analyst
12092  Information technology, networking, or system ...      Data Scientist
13468      Computer science (software engineering, etc.)      Data Scientist
14367  Medical or life sciences (biology, chemistry, ...      Data Scientist
15469                           Mathematics or statistics  Research Scientist
15825      Computer science (software engineering, etc.)    Business Analyst
16404                                                NaN          Consultant
18120      Computer science (software engineering, etc.)      Data Scientist
20576      Computer science (software engineering, etc.)             Student
21122      Computer science (software engineering, etc.)   Software Engineer
```

(continues on next page)

(continued from previous page)

```
22264                Engineering (non-computer focused)    Business Analyst
22591  Information technology, networking, or system ...  Research Assistant

      Q6_OTHER_TEXT                                    Q7  ... Q49_OTHER_TEXT  \
2322            113                       I am a student  ...             -1
8899             -1                   Accounting/Finance  ...             -1
12092            -1                 Computers/Technology  ...             -1
13468            -1                 Computers/Technology  ...             -1
14367            -1                 Computers/Technology  ...            166
15469            -1                   Accounting/Finance  ...             -1
15825            -1                 Computers/Technology  ...             -1
16404            -1     Hospitality/Entertainment/Sports  ...             -1
18120            -1                 Computers/Technology  ...             -1
20576            -1                       I am a student  ...             -1
21122            -1                 Computers/Technology  ...             -1
22264            -1  Online Business/Internet-based Sales  ...             -1
22591            -1                 Academics/Education  ...             -1

          Q50_Part_1          Q50_Part_2  \
2322             NaN                 NaN
8899             NaN                 NaN
12092            NaN  Too time-consuming
13468  Too expensive  Too time-consuming
14367            NaN                 NaN
15469            NaN                 NaN
15825            NaN                 NaN
16404            NaN                 NaN
18120            NaN                 NaN
20576            NaN                 NaN
21122            NaN                 NaN
22264            NaN                 NaN
22591            NaN                 NaN

                                Q50_Part_3  \
2322                                   NaN
8899                                   NaN
12092  Requires too much technical knowledge
13468  Requires too much technical knowledge
14367                                  NaN
15469                                  NaN
15825                                  NaN
16404                                  NaN
18120                                  NaN
20576                                  NaN
21122                                  NaN
22264                                  NaN
22591                                  NaN

                                          Q50_Part_4  \
2322                                             NaN
8899                                             NaN
12092                                            NaN
13468  Afraid that others will use my work without gi...
14367                                            NaN
15469                                            NaN
15825                                            NaN
16404                                            NaN
```

(continues on next page)

(continued from previous page)

```
18120                                               NaN
20576                                               NaN
21122                                               NaN
22264                                               NaN
22591                                               NaN

                                   Q50_Part_5  \
2322                                      NaN
8899                                      NaN
12092                                     NaN
13468  Not enough incentives to share my work
14367                                     NaN
15469                                     NaN
15825                                     NaN
16404                                     NaN
18120                                     NaN
20576                                     NaN
21122                                     NaN
22264                                     NaN
22591                                     NaN

                                              Q50_Part_6  \
2322                                                 NaN
8899                                                 NaN
12092                                                NaN
13468  I had never considered making my work easier f...
14367                                                NaN
15469                                                NaN
15825                                                NaN
16404                                                NaN
18120                                                NaN
20576                                                NaN
21122                                                NaN
22264                                                NaN
22591                                                NaN

                               Q50_Part_7 Q50_Part_8 Q50_OTHER_TEXT
2322                                  NaN        NaN             -1
8899                                  NaN      Other            260
12092                                 NaN        NaN             -1
13468                                 NaN        NaN             -1
14367  None of these reasons apply to me        NaN             -1
15469                                 NaN        NaN             -1
15825                                 NaN        NaN             -1
16404                                 NaN        NaN             -1
18120                                 NaN        NaN             -1
20576                                 NaN        NaN             -1
21122                                 NaN        NaN             -1
22264                                 NaN        NaN             -1
22591                                 NaN        NaN             -1

[13 rows x 395 columns]
```

No way they are this succesfull, i'm not yet going to remove them, but i'm definitely going to keep this in mind, this might break our predictions!

Later on I will remove the entries without salaries, but im going to keep them in a prediction dataframe, so we could perhaps predict their salary, we don't have a reference but still might be interesting. For the rest of the preparation im going to keep them in here so the final format of both train and prediction are the same.

38.3.3 Duplicates

It is very highly unlikely but just to check if no one has entered the same survey twice, we check for duplicates

```
choice_df[choice_df.duplicated()]
```

```
      Time from Start to Finish (seconds)    Q1 Q1_OTHER_TEXT     Q2  \
15278                                  36  Male            -1  18-21
15865                                  23  Male            -1  18-21
17521                                  36  Male            -1  25-29
18257                                  27  Male            -1  25-29
18320                                  46  Male            -1  35-39
18966                                  43  Male            -1  18-21
21214                                 106  Male            -1  18-21
21916                                  45  Male            -1  22-24
22049                                  46  Male            -1  25-29
22638                                  60  Male            -1  25-29
22816                                  41  Male            -1  25-29
23683                                  70  Male            -1  25-29

                                          Q3                 Q4  \
15278                                  China                NaN
15865               United States of America                NaN
17521               United States of America    Master's degree
18257                                 Brazil                NaN
18320               United States of America                NaN
18966                                  India  Bachelor's degree
21214                                  India  Bachelor's degree
21916                                  China                NaN
22049                                  China                NaN
22638                                  China                NaN
22816  I do not wish to disclose my location                NaN
23683                                 France    Master's degree

                                               Q5       Q6 Q6_OTHER_TEXT  \
15278                                         NaN      NaN            -1
15865                                         NaN      NaN            -1
17521                                         NaN      NaN            -1
18257                                         NaN      NaN            -1
18320                                         NaN      NaN            -1
18966                                         NaN      NaN            -1
21214  Computer science (software engineering, etc.)  Student            -1
21916                                         NaN      NaN            -1
22049                                         NaN      NaN            -1
22638                                         NaN      NaN            -1
22816                                         NaN      NaN            -1
23683                                         NaN      NaN            -1

                  Q7  ... Q49_OTHER_TEXT Q50_Part_1 Q50_Part_2 Q50_Part_3  \
15278            NaN  ...             -1        NaN        NaN        NaN
15865            NaN  ...             -1        NaN        NaN        NaN
17521            NaN  ...             -1        NaN        NaN        NaN
18257            NaN  ...             -1        NaN        NaN        NaN
```

(continues on next page)

(continued from previous page)

```
18320              NaN  ...          -1        NaN        NaN        NaN
18966              NaN  ...          -1        NaN        NaN        NaN
21214  I am a student  ...          -1        NaN        NaN        NaN
21916              NaN  ...          -1        NaN        NaN        NaN
22049              NaN  ...          -1        NaN        NaN        NaN
22638              NaN  ...          -1        NaN        NaN        NaN
22816              NaN  ...          -1        NaN        NaN        NaN
23683              NaN  ...          -1        NaN        NaN        NaN

      Q50_Part_4 Q50_Part_5 Q50_Part_6 Q50_Part_7 Q50_Part_8 Q50_OTHER_TEXT
15278        NaN        NaN        NaN        NaN        NaN             -1
15865        NaN        NaN        NaN        NaN        NaN             -1
17521        NaN        NaN        NaN        NaN        NaN             -1
18257        NaN        NaN        NaN        NaN        NaN             -1
18320        NaN        NaN        NaN        NaN        NaN             -1
18966        NaN        NaN        NaN        NaN        NaN             -1
21214        NaN        NaN        NaN        NaN        NaN             -1
21916        NaN        NaN        NaN        NaN        NaN             -1
22049        NaN        NaN        NaN        NaN        NaN             -1
22638        NaN        NaN        NaN        NaN        NaN             -1
22816        NaN        NaN        NaN        NaN        NaN             -1
23683        NaN        NaN        NaN        NaN        NaN             -1

[12 rows x 395 columns]
```

I take back my words, seems there are some faulty entries, perhaps we should even improve our bad entry detection? For now im just going to remove duplicates

```
choice_df = choice_df.drop_duplicates()
```

At this point im going to seperate the non salary entries from our training dataframe. resulting in 2 partitions:

- train_df
- prediction_df

```
prediction_df = choice_df[(choice_df.Q9.isna()) | (choice_df.Q9=='I do not wish to␣
↪disclose my approximate yearly compensation')]
train_df = choice_df.drop(prediction_df.index)
print('prediction shape:' + str(prediction_df.shape))
print('remaining shape:' + str(train_df.shape))
```

```
prediction shape:(8418, 395)
remaining shape:(15429, 395)
```

38.4 Processing

For other questions I selected a few that caught my interest, here is the list that made it. Notice that I did not perform any preparation on these question as they mostly are checkmarks on a survey, yet in processing I am going to create a more convenient method to store them.

```
print(questions.Q11_Part_1)
#print(questions.Q12_Part_1_TEXT)
print(questions.Q13_Part_1)
```

(continues on next page)

(continued from previous page)

```
print(questions.Q16_Part_1)
print(questions.Q17)
print(questions.Q19_Part_1)
print(questions.Q21_Part_1)
print(questions.Q31_Part_1)
print(questions.Q34_Part_1)
print(questions.Q42_Part_1)
print(questions.Q49_Part_1)
```

```
Select any activities that make up an important part of your role at work: (Select␣
↪all that apply) - Selected Choice - Analyze and understand data to influence␣
↪product or business decisions
Which of the following integrated development environments (IDE's) have you used at␣
↪work or school in the last 5 years? (Select all that apply) - Selected Choice -␣
↪Jupyter/IPython
What programming languages do you use on a regular basis? (Select all that apply) -␣
↪Selected Choice - Python
What specific programming language do you use most often? - Selected Choice
What machine learning frameworks have you used in the past 5 years? (Select all that␣
↪apply) - Selected Choice - Scikit-Learn
What data visualization libraries or tools have you used in the past 5 years? (Select␣
↪all that apply) - Selected Choice - ggplot2
Which types of data do you currently interact with most often at work or school?␣
↪(Select all that apply) - Selected Choice - Audio Data
During a typical data science project at work or school, approximately what␣
↪proportion of your time is devoted to the following? (Answers must add up to 100%) -
↪ Gathering data
What metrics do you or your organization use to determine whether or not your models␣
↪were successful? (Select all that apply) - Selected Choice - Revenue and/or␣
↪business goals
What tools and methods do you use to make your work easy to reproduce? (Select all␣
↪that apply) - Selected Choice - Share code on Github or a similar code-sharing␣
↪repository
```

38.4.1 One hot encoding questions

What I will do here is create a makeshift database, not in SQL as usually just to keep it simple, but in a dictionary of dataframes. For each question I will take the answers and create a one hot encoded table from them, for each user we will know which checkmarks they marked and which they didn't. This view makes it easier to apply statistics and machine learning to the data.

```
answer_dfs = {}
for question in ['Q11', 'Q13', 'Q16', 'Q19', 'Q21', 'Q31', 'Q34', 'Q42', 'Q49']:

    choices = train_df[train_df.columns[train_df.columns.str.contains(question)][:-
↪1]].notnull().astype(int)
    choices.columns = questions[questions.index.str.contains(question)][:-1].str.
↪split(' - ').apply(lambda x: x[-1]).values
    answer_dfs[question] = choices
```

an example of a question, Q13: Which IDE's have you used in the last 5 years?

```
answer_dfs['Q13']
```

```
       Jupyter/IPython  RStudio  PyCharm  Visual Studio Code  nteract  Atom  \
2                    0        0        0                   0        0     0
3                    0        0        0                   0        0     0
5                    0        1        0                   0        0     0
7                    0        1        0                   0        0     0
8                    1        0        1                   0        0     1
...                ...      ...      ...                 ...      ...   ...
23844                1        0        1                   0        0     0
23845                0        0        0                   0        0     0
23854                0        0        0                   0        0     0
23855                1        1        1                   0        0     0
23857                0        0        0                   0        0     0

       MATLAB  Visual Studio  Notepad++  Sublime Text  Vim  IntelliJ  Spyder  \
2           0              0          0             0    0         0       0
3           1              0          0             0    0         0       0
5           0              0          0             0    0         0       0
7           0              0          0             0    0         0       0
8           0              1          1             1    0         1       1
...       ...            ...        ...           ...  ...       ...     ...
23844       0              0          0             1    1         1       0
23845       0              0          0             0    0         0       0
23854       0              0          0             0    0         0       0
23855       1              0          0             1    0         1       0
23857       0              0          0             0    0         0       0

       None  Other
2         1      0
3         0      0
5         0      0
7         0      0
8         0      0
...     ...    ...
23844     0      0
23845     0      0
23854     0      0
23855     0      0
23857     0      0

[15429 rows x 15 columns]
```

for some reason they did Q17 differently, so we have to one hot encode it in another method.

```
answer_dfs['Q17'] = pd.get_dummies(train_df[train_df.columns[train_df.columns.str.
 ↪contains('Q17')][:-1]])
answer_dfs['Q17']
```

```
       Q17_Bash  Q17_C#/.NET  Q17_C/C++  Q17_Go  Q17_Java  \
2             0            0          0       0         0
3             0            0          0       0         1
5             0            0          0       0         0
7             0            0          0       0         0
8             0            0          0       0         0
...         ...          ...        ...     ...       ...
23844         0            0          0       0         1
23845         0            0          0       0         0
23854         0            0          0       0         0
```

(continues on next page)

(continued from previous page)

```
23855          0             0           1        0          0
23857          0             0           0        0          0

       Q17_Javascript/Typescript  Q17_Julia  Q17_MATLAB  Q17_Other  Q17_PHP  \
2                              0          0           0          0        0
3                              0          0           0          0        0
5                              0          0           0          0        0
7                              0          0           0          0        0
8                              0          0           0          0        0
...                          ...        ...         ...        ...      ...
23844                          0          0           0          0        0
23845                          0          0           0          0        0
23854                          0          0           0          0        0
23855                          0          0           0          0        0
23857                          0          0           0          0        0

       Q17_Python  Q17_R  Q17_Ruby  Q17_SAS/STATA  Q17_SQL  Q17_Scala  \
2               0      0         0              0        0          0
3               0      0         0              0        0          0
5               0      0         0              0        1          0
7               0      0         0              0        0          0
8               0      0         0              0        0          0
...           ...    ...       ...            ...      ...        ...
23844           0      0         0              0        0          0
23845           0      0         0              0        0          0
23854           0      0         0              0        0          0
23855           0      0         0              0        0          0
23857           0      0         0              0        0          0

       Q17_Visual Basic/VBA
2                         0
3                         0
5                         0
7                         0
8                         0
...                     ...
23844                     0
23845                     0
23854                     0
23855                     0
23857                     0

[15429 rows x 17 columns]
```

That was for our choices data, where the questions are based on choices, for generic info we do it a bit different, we create a general dataframe containing all info.

```
info_df = train_df[['Q1', 'Q2', 'Q3', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9', 'Q10']]
#info_df.columns = questions[['Q1', 'Q2', 'Q3', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9',
↪'Q10']]
```

```
info_df
```

```
           Q1     Q2                        Q3                  Q4  \
2        Male  30-34                 Indonesia   Bachelor's degree
3      Female  30-34  United States of America     Master's degree
```

(continues on next page)

(continued from previous page)

```
5        Male  22-24                    India    Master’s degree
7        Male  35-39                    Chile    Doctoral degree
8        Male  18-21                    India    Master’s degree
...       ...    ...                      ...                ...
23844    Male  30-34              Netherlands    Master’s degree
23845    Male  22-24                  Romania    Master’s degree
23854    Male  30-34                   Turkey    Doctoral degree
23855    Male  45-49                   France    Doctoral degree
23857    Male  22-24                   Turkey    Master’s degree

                                                      Q5                  Q6  \
2                    Engineering (non-computer focused)                Other
3         Computer science (software engineering, etc.)       Data Scientist
5                           Mathematics or statistics         Data Analyst
7      Information technology, networking, or system ...                Other
8      Information technology, networking, or system ...                Other
...                                                  ...                  ...
23844     Computer science (software engineering, etc.)   Software Engineer
23845                           Mathematics or statistics              Student
23854     Computer science (software engineering, etc.)  Research Assistant
23855     Computer science (software engineering, etc.)       Chief Officer
23857     Computer science (software engineering, etc.)   Software Engineer

                               Q7     Q8           Q9  \
2      Manufacturing/Fabrication   5-10    10-20,000
3                 I am a student    0-1     0-10,000
5                 I am a student    0-1     0-10,000
7            Academics/Education  10-15    10-20,000
8                          Other    0-1     0-10,000
...                          ...    ...          ...
23844       Computers/Technology  10-15   90-100,000
23845             I am a student    0-1     0-10,000
23854        Academics/Education   5-10    10-20,000
23855       Computers/Technology   5-10  250-300,000
23857       Computers/Technology    0-1    10-20,000

                                                     Q10
2                          No (we do not use ML methods)
3                                          I do not know
5                                          I do not know
7                          No (we do not use ML methods)
8      We recently started using ML methods (i.e., mo...
...                                                  ...
23844  We are exploring ML methods (and may one day p...
23845                                                NaN
23854                                                NaN
23855  We recently started using ML methods (i.e., mo...
23857  We recently started using ML methods (i.e., mo...

[15429 rows x 10 columns]
```

38.4.2 Mean choice Matrix

As we have so much information to process, I opted to keep it dynamic, the following function helps in that, it calculates for a question from our choice database the mean occurence for each group in a feature of the info dataframe. Let's say we want to know the average amount of persons that know a specific language for each role/job title. We would have to match Q16 (known languages) with Q6 (job description). This is performed below, notice how it both performs a merge (join) and a groupby to get the result.

```
def mean_choice_matrix(info, question):
    return info_df[[info]].join(answer_dfs[question]).groupby(info).mean()
```

```
mean_choice_matrix('Q6','Q16')
```

```
                             Python         R       SQL      Bash      Java  \
Q6
Business Analyst           0.605085  0.401695  0.547458  0.049153  0.079661
Chief Officer              0.717131  0.274900  0.430279  0.191235  0.183267
Consultant                 0.692845  0.413613  0.472949  0.116928  0.136126
DBA/Database Engineer      0.666667  0.307692  0.717949  0.188034  0.213675
Data Analyst               0.647059  0.484594  0.586134  0.079132  0.093137
Data Engineer              0.801418  0.248227  0.510638  0.223404  0.246454
Data Journalist            0.600000  0.400000  0.400000  0.200000  0.200000
Data Scientist             0.860265  0.429671  0.511542  0.193906  0.100031
Developer Advocate         0.611765  0.211765  0.482353  0.094118  0.341176
Manager                    0.681416  0.384956  0.455752  0.117257  0.126106
Marketing Analyst          0.505747  0.471264  0.505747  0.022989  0.022989
Not employed                    NaN       NaN       NaN       NaN       NaN
Other                      0.682041  0.287537  0.314033  0.116781  0.145240
Principal Investigator     0.759036  0.433735  0.277108  0.144578  0.144578
Product/Project Manager    0.720365  0.276596  0.455927  0.100304  0.161094
Research Assistant         0.776286  0.310962  0.192394  0.154362  0.161074
Research Scientist         0.780541  0.340541  0.196757  0.188108  0.142703
Salesperson                0.612500  0.275000  0.250000  0.025000  0.125000
Software Engineer          0.737179  0.123504  0.405128  0.167949  0.334188
Statistician               0.500000  0.822222  0.327778  0.077778  0.033333
Student                    0.697710  0.231679  0.214885  0.083969  0.225573

                           Javascript/Typescript  Visual Basic/VBA     C/C++  \
Q6
Business Analyst                        0.084746          0.189831  0.069492
Chief Officer                           0.306773          0.083665  0.139442
Consultant                              0.165794          0.111693  0.078534
DBA/Database Engineer                   0.128205          0.068376  0.102564
Data Analyst                            0.098039          0.110644  0.079132
Data Engineer                           0.184397          0.053191  0.132979
Data Journalist                         0.400000          0.000000  0.100000
Data Scientist                          0.098492          0.043398  0.105571
Developer Advocate                      0.400000          0.094118  0.141176
Manager                                 0.137168          0.139381  0.097345
Marketing Analyst                       0.045977          0.068966  0.034483
Not employed                                 NaN               NaN       NaN
Other                                   0.131501          0.087341  0.144259
Principal Investigator                  0.216867          0.096386  0.265060
Product/Project Manager                 0.206687          0.088146  0.133739
Research Assistant                      0.102908          0.038031  0.255034
Research Scientist                      0.088649          0.038919  0.301622
Salesperson                             0.137500          0.075000  0.137500
```

(continues on next page)

(continued from previous page)

```
Software Engineer                 0.354274         0.036325  0.257692
Statistician                      0.050000         0.072222  0.094444
Student                           0.108397         0.027481  0.330153

                           MATLAB     Scala     Julia        Go   C#/.NET  \
Q6
Business Analyst         0.030508  0.016949  0.003390  0.005085  0.042373
Chief Officer            0.091633  0.067729  0.015936  0.059761  0.103586
Consultant               0.062827  0.024433  0.010471  0.012216  0.082024
DBA/Database Engineer    0.051282  0.025641  0.000000  0.017094  0.179487
Data Analyst             0.067227  0.028711  0.005602  0.006303  0.039916
Data Engineer            0.081560  0.161348  0.008865  0.033688  0.086879
Data Journalist          0.200000  0.100000  0.000000  0.000000  0.200000
Data Scientist           0.076023  0.068637  0.014158  0.018159  0.040320
Developer Advocate       0.035294  0.035294  0.000000  0.023529  0.211765
Manager                  0.050885  0.024336  0.011062  0.011062  0.075221
Marketing Analyst        0.011494  0.000000  0.000000  0.000000  0.011494
Not employed                  NaN       NaN       NaN       NaN       NaN
Other                    0.100098  0.020608  0.005888  0.012758  0.073602
Principal Investigator   0.240964  0.024096  0.060241  0.024096  0.096386
Product/Project Manager  0.063830  0.039514  0.003040  0.012158  0.106383
Research Assistant       0.308725  0.013423  0.020134  0.000000  0.053691
Research Scientist       0.265946  0.024865  0.019459  0.003243  0.061622
Salesperson              0.087500  0.025000  0.000000  0.000000  0.025000
Software Engineer        0.073077  0.050855  0.008547  0.052137  0.185470
Statistician             0.166667  0.000000  0.016667  0.000000  0.038889
Student                  0.198473  0.015649  0.011832  0.008397  0.054198

                              PHP      Ruby  SAS/STATA      None     Other
Q6
Business Analyst         0.035593  0.006780   0.091525  0.054237  0.027119
Chief Officer            0.103586  0.043825   0.035857  0.023904  0.055777
Consultant               0.052356  0.012216   0.073298  0.031414  0.041885
DBA/Database Engineer    0.042735  0.008547   0.025641  0.008547  0.017094
Data Analyst             0.032213  0.014006   0.105042  0.011905  0.018908
Data Engineer            0.054965  0.015957   0.033688  0.003546  0.028369
Data Journalist          0.000000  0.000000   0.100000  0.000000  0.000000
Data Scientist           0.024007  0.008618   0.058172  0.001847  0.023084
Developer Advocate       0.141176  0.023529   0.011765  0.000000  0.058824
Manager                  0.044248  0.022124   0.077434  0.044248  0.028761
Marketing Analyst        0.034483  0.000000   0.057471  0.080460  0.000000
Not employed                  NaN       NaN        NaN       NaN       NaN
Other                    0.056919  0.012758   0.051030  0.056919  0.037291
Principal Investigator   0.096386  0.012048   0.060241  0.012048  0.072289
Product/Project Manager  0.091185  0.027356   0.024316  0.045593  0.027356
Research Assistant       0.071588  0.006711   0.046980  0.011186  0.031320
Research Scientist       0.043243  0.011892   0.043243  0.014054  0.043243
Salesperson              0.062500  0.012500   0.037500  0.100000  0.037500
Software Engineer        0.110684  0.038034   0.003419  0.001709  0.051282
Statistician             0.005556  0.016667   0.355556  0.016667  0.022222
Student                  0.051527  0.010305   0.035878  0.006489  0.015267
```

We can see that for each combination of job title and programming language an average between 0 and 1 persons have checked this option, e.g. the combination of data scientist and python equals 0.86, meaning that 86% of data scientists know python.

Similarly we can also calculate correlation between choices from our choice database, here we did it again for Question

16.

```
answer_dfs['Q16'].corr()
```

```
                          Python         R       SQL      Bash      Java  \
Python                  1.000000  0.077293  0.191304  0.188435  0.141813
R                       0.077293  1.000000  0.223527  0.032511 -0.034205
SQL                     0.191304  0.223527  1.000000  0.161086  0.135890
Bash                    0.188435  0.032511  0.161086  1.000000  0.078031
Java                    0.141813 -0.034205  0.135890  0.078031  1.000000
Javascript/Typescript   0.125164 -0.039002  0.192323  0.146723  0.254773
Visual Basic/VBA        0.007550  0.098949  0.159062 -0.026907  0.024432
C/C++                   0.183621 -0.049046 -0.034188  0.082853  0.227691
MATLAB                  0.131117  0.030446 -0.047761  0.010577  0.064536
Scala                   0.095374  0.033991  0.117620  0.116862  0.165490
Julia                   0.039096  0.063129  0.013813  0.058785  0.005821
Go                      0.053180 -0.016114  0.048231  0.104544  0.073318
C#/.NET                 0.046182 -0.039415  0.134615  0.000396  0.137888
PHP                     0.048012 -0.006318  0.157483  0.054351  0.177413
Ruby                    0.031308  0.014921  0.056242  0.103550  0.067257
SAS/STATA              -0.009036  0.198183  0.120958 -0.032248 -0.040728
None                   -0.206520 -0.085536 -0.101752 -0.049637 -0.056888
Other                   0.009641 -0.001240  0.001674  0.056457  0.026739

                        Javascript/Typescript  Visual Basic/VBA     C/C++  \
Python                               0.125164          0.007550  0.183621
R                                   -0.039002          0.098949 -0.049046
SQL                                  0.192323          0.159062 -0.034188
Bash                                 0.146723         -0.026907  0.082853
Java                                 0.254773          0.024432  0.227691
Javascript/Typescript                1.000000          0.049290  0.095921
Visual Basic/VBA                     0.049290          1.000000  0.020796
C/C++                                0.095921          0.020796  1.000000
MATLAB                              -0.004829          0.019424  0.260311
Scala                                0.060073          0.005185  0.004697
Julia                                0.025601          0.037063  0.049980
Go                                   0.118897          0.002042  0.048903
C#/.NET                              0.222775          0.122287  0.134850
PHP                                  0.307413          0.077126  0.111623
Ruby                                 0.127312          0.011487  0.041745
SAS/STATA                           -0.048567          0.093618 -0.052866
None                                -0.052961         -0.032002 -0.058636
Other                                0.034497         -0.004965  0.019235

                          MATLAB     Scala     Julia        Go   C#/.NET  \
Python                  0.131117  0.095374  0.039096  0.053180  0.046182
R                       0.030446  0.033991  0.063129 -0.016114 -0.039415
SQL                    -0.047761  0.117620  0.013813  0.048231  0.134615
Bash                    0.010577  0.116862  0.058785  0.104544  0.000396
Java                    0.064536  0.165490  0.005821  0.073318  0.137888
Javascript/Typescript -0.004829  0.060073  0.025601  0.118897  0.222775
Visual Basic/VBA        0.019424  0.005185  0.037063  0.002042  0.122287
C/C++                   0.260311  0.004697  0.049980  0.048903  0.134850
MATLAB                  1.000000  0.003116  0.056403  0.003529  0.029772
Scala                   0.003116  1.000000  0.049863  0.077110  0.006505
Julia                   0.056403  0.049863  1.000000  0.091513  0.013769
Go                      0.003529  0.077110  0.091513  1.000000  0.042659
C#/.NET                 0.029772  0.006505  0.013769  0.042659  1.000000
```

(continues on next page)

(continued from previous page)

```
PHP                    0.046232  0.021619  0.010427  0.043077  0.108940
Ruby                   0.014027  0.066129  0.065590  0.092767  0.039859
SAS/STATA              0.015261  0.012633  0.045429 -0.008502 -0.026410
None                  -0.044376 -0.025780 -0.013031 -0.017048 -0.035878
Other                 -0.000043  0.014499  0.006897  0.012141  0.006953

                            PHP      Ruby  SAS/STATA      None     Other
Python                 0.048012  0.031308  -0.009036 -0.206520  0.009641
R                     -0.006318  0.014921   0.198183 -0.085536 -0.001240
SQL                    0.157483  0.056242   0.120958 -0.101752  0.001674
Bash                   0.054351  0.103550  -0.032248 -0.049637  0.056457
Java                   0.177413  0.067257  -0.040728 -0.056888  0.026739
Javascript/Typescript  0.307413  0.127312  -0.048567 -0.052961  0.034497
Visual Basic/VBA       0.077126  0.011487   0.093618 -0.032002 -0.004965
C/C++                  0.111623  0.041745  -0.052866 -0.058636  0.019235
MATLAB                 0.046232  0.014027   0.015261 -0.044376 -0.000043
Scala                  0.021619  0.066129   0.012633 -0.025780  0.014499
Julia                  0.010427  0.065590   0.045429 -0.013031  0.006897
Go                     0.043077  0.092767  -0.008502 -0.017048  0.012141
C#/.NET                0.108940  0.039859  -0.026410 -0.035878  0.006953
PHP                    1.000000  0.080351  -0.011212 -0.029713  0.031095
Ruby                   0.080351  1.000000   0.007042 -0.015859  0.034500
SAS/STATA             -0.011212  0.007042   1.000000 -0.029014 -0.005546
None                  -0.029713 -0.015859  -0.029014  1.000000 -0.021852
Other                  0.031095  0.034500  -0.005546 -0.021852  1.000000
```

Here we see thich answers are checked usually together or not, as an example we see that python and SQL have a correlation of 19% whilst Python and R have a correlation of 7.7% which is logical as Python and R have a similar purpose and SQL is complementary. Obviously None is always negatively correlated, a good example of obsolete information!

38.4.3 Count matrix

to correlate information between 2 questions of the info dataframe, we create a function that counts the occurence of each combination. An example is given for question 2 (age) and Question 7 (industry). With this information we can find out if there is a correlation between information of our users in the survey, not specifically their choices on the multiple choice answers.

```
def count_matrix(q1, q2):
    return info_df[[q1, q2]].groupby([q1, q2]).size().unstack()
def mean_matrix(q1, q2):
    return info_df[[q1, q2]].groupby([q1, q2]).size().unstack().apply(lambda x: x/x.
 ↪sum(), axis='columns')
```

```
count_matrix('Q2', 'Q7')
```

```
Q7      Academics/Education  Accounting/Finance  Broadcasting/Communications  \
Q2
18-21                   117                  27                            4
22-24                   322                 198                           38
25-29                   539                 361                           86
30-34                   364                 252                           66
35-39                   243                 141                           56
40-44                   146                  82                           33
45-49                    91                  47                           24
```

(continues on next page)

(continued from previous page)

```
50-54                   71                  38                           5
55-59                   35                  18                           4
60-69                   40                  14                           3
70-79                   11                   2                           0
80+                      2                   1                           0

Q7      Computers/Technology  Energy/Mining  Government/Public Service  \
Q2
18-21                    194              8                          9
22-24                    786             38                         37
25-29                   1250             97                        118
30-34                    795             81                        125
35-39                    454             44                         74
40-44                    279             40                         48
45-49                    195             13                         40
50-54                    123              8                         34
55-59                     64              7                         17
60-69                     32              8                         20
70-79                      5              1                          0
80+                        2              0                          2

Q7      Hospitality/Entertainment/Sports  I am a student  \
Q2
18-21                                  3             869
22-24                                 22             811
25-29                                 39             424
30-34                                 34              89
35-39                                 25              32
40-44                                 14              12
45-49                                  5               5
50-54                                  3               2
55-59                                  5               3
60-69                                  1               0
70-79                                  0               0
80+                                    1               1

Q7      Insurance/Risk Assessment  Manufacturing/Fabrication  Marketing/CRM  \
Q2
18-21                          12                          7             11
22-24                          56                         41             71
25-29                         131                        111            124
30-34                         109                         96             83
35-39                          73                         62             50
40-44                          30                         40             22
45-49                          15                         32             12
50-54                          16                         21              4
55-59                           7                         12              6
60-69                           4                         12              1
70-79                           1                          0              2
80+                             0                          1              0

Q7      Medical/Pharmaceutical  Military/Security/Defense  Non-profit/Service  \
Q2
18-21                       16                          9                   2
22-24                       76                         33                  16
25-29                      179                         31                  50
30-34                      127                         25                  40
```

(continues on next page)

(continued from previous page)

```
35-39                      76                        22                  17
40-44                      37                        14                  13
45-49                      27                         4                   9
50-54                      21                         6                   5
55-59                      17                         2                   1
60-69                      14                         3                   0
70-79                       1                         0                   0
80+                         1                         1                   0

Q7     Online Business/Internet-based Sales  \
Q2
18-21                                    12
22-24                                    43
25-29                                   107
30-34                                    62
35-39                                    23
40-44                                    14
45-49                                     5
50-54                                     4
55-59                                     1
60-69                                     1
70-79                                     0
80+                                       0

Q7     Online Service/Internet-based Services  Other  Retail/Sales  \
Q2
18-21                                      20     27             9
22-24                                     123     98            59
25-29                                     227    227           110
30-34                                     162    169            81
35-39                                      86     79            44
40-44                                      51     69            25
45-49                                      24     34            14
50-54                                      10     18             5
55-59                                       4     10             6
60-69                                       8     13             2
70-79                                       0      3             0
80+                                         0      3             1

Q7     Shipping/Transportation
Q2
18-21                        7
22-24                       26
25-29                       67
30-34                       59
35-39                       36
40-44                       20
45-49                        6
50-54                        9
55-59                        3
60-69                        4
70-79                        0
80+                          0
```

38.5 Exploration

To start of our exploration I would like to know what influences our salary, to do so I created a count_matrix function that counts the occurences of each option with information questions, to illustrate an example with Q4: which degree?

```
count_matrix('Q4', 'Q9')
```

```
Q9                                                    0-10,000  10-20,000  \
Q4
Bachelor's degree                                         1790        567
Doctoral degree                                            343        307
I prefer not to answer                                      38         32
Master's degree                                           1812        909
No formal education past high school                        32         20
Professional degree                                        112         61
Some college/university study without earning a...        271         41

Q9                                                    20-30,000  30-40,000  \
Q4
Bachelor's degree                                          341        259
Doctoral degree                                            255        223
I prefer not to answer                                       9          8
Master's degree                                            672        546
No formal education past high school                         8         16
Professional degree                                         56         34
Some college/university study without earning a...          54         33

Q9                                                    40-50,000  50-60,000  \
Q4
Bachelor's degree                                          223        183
Doctoral degree                                            183        179
I prefer not to answer                                       6          9
Master's degree                                            491        491
No formal education past high school                        10         15
Professional degree                                         21         18
Some college/university study without earning a...          31         24

Q9                                                    60-70,000  70-80,000  \
Q4
Bachelor's degree                                          176        159
Doctoral degree                                            157        120
I prefer not to answer                                       7          3
Master's degree                                            353        353
No formal education past high school                         1          7
Professional degree                                         15         11
Some college/university study without earning a...          20         24

Q9                                                    80-90,000  90-100,000  \
Q4
Bachelor's degree                                          118         124
Doctoral degree                                            102         110
I prefer not to answer                                       1           2
Master's degree                                            260         304
No formal education past high school                         4           4
Professional degree                                          6          11
Some college/university study without earning a...          15          11
```

(continues on next page)

(continued from previous page)

```
Q9                                                      100-125,000  125-150,000  \
Q4
Bachelor's degree                                               166          115
Doctoral degree                                                 185          124
I prefer not to answer                                            5            1
Master's degree                                                 449          265
No formal education past high school                              4            2
Professional degree                                              11            7
Some college/university study without earning a...               23           19

Q9                                                      150-200,000  200-250,000  \
Q4
Bachelor's degree                                                89           23
Doctoral degree                                                 120           55
I prefer not to answer                                            2            1
Master's degree                                                 223           77
No formal education past high school                              3            2
Professional degree                                               8            9
Some college/university study without earning a...               12            5

Q9                                                      250-300,000  300-400,000  \
Q4
Bachelor's degree                                                15           12
Doctoral degree                                                  25           18
I prefer not to answer                                            1            0
Master's degree                                                  32           20
No formal education past high school                              0            0
Professional degree                                               0            2
Some college/university study without earning a...                2            0

Q9                                                      400-500,000  500,000+
Q4
Bachelor's degree                                                 4        19
Doctoral degree                                                   7         9
I prefer not to answer                                            1         4
Master's degree                                                   9        20
No formal education past high school                              1         6
Professional degree                                               0         2
Some college/university study without earning a...                1         3
```

By using a contingency chi squared test we can find out which degrees are under- and overrepresented for which salary ranges.

```
F, p, df, exp = scipy.stats.chi2_contingency(count_matrix('Q4','Q9'))
F, p
```

```
(1065.634531227411, 3.0827908345529236e-160)
```

With such significance we already know this is not a coincidence and the correlation will propably be large. comparing true and expected values we can see where the difference is.

```
degree_diff = count_matrix('Q4', 'Q9')-exp
degree_diff
```

```
Q9                                                        0-10,000  10-20,000  \
Q4
```

(continues on next page)

(continued from previous page)

```
Bachelor’s degree                                    540.636205  16.745868
Doctoral degree                                     -375.890142  -9.618964
I prefer not to answer                                 0.943807  15.679435
Master’s degree                                     -264.857087  -5.704906
No formal education past high school                  -6.481431   3.051721
Professional degree                                    2.541707  12.791561
Some college/university study without earning a...  103.106941 -32.944714

Q9                                                   20-30,000  30-40,000  \
Q4
Bachelor’s degree                                   -55.285242 -58.880420
Doctoral degree                                      26.975501  40.090025
I prefer not to answer                               -2.753840  -1.428349
Master’s degree                                      13.242465  17.577290
No formal education past high school                 -4.205911   6.209022
Professional degree                                  21.280964   6.150107
Some college/university study without earning a...   0.746063  -9.717675

Q9                                                   40-50,000  50-60,000  \
Q4
Bachelor’s degree                                   -51.132802 -78.065332
Doctoral degree                                      25.262622  28.781710
I prefer not to answer                               -2.130793   1.256789
Master’s degree                                      35.300344  57.022814
No formal education past high school                  1.556485   6.958973
Professional degree                                  -3.017111  -4.872254
Some college/university study without earning a...  -5.838745 -11.082701

Q9                                                   60-70,000  70-80,000  \
Q4
Bachelor’s degree                                   -31.090997 -33.319074
Doctoral degree                                      37.838810   9.338648
I prefer not to answer                                0.857671  -2.704193
Master’s degree                                       8.746063  33.301899
No formal education past high school                 -5.378573   1.076415
Professional degree                                  -3.143496  -5.849310
Some college/university study without earning a...  -7.829477  -1.844384

Q9                                                   80-90,000  90-100,000  \
Q4
Bachelor’s degree                                   -25.742174  -36.786700
Doctoral degree                                      19.290038   17.482533
I prefer not to answer                               -3.263400   -2.768942
Master’s degree                                      21.052823   36.719165
No formal education past high school                 -0.427377   -0.952362
Professional degree                                  -6.593428   -3.086720
Some college/university study without earning a...  -4.316482  -10.606974

Q9                                                   100-125,000  125-150,000  \
Q4
Bachelor’s degree                                     -73.475598   -36.412211
Doctoral degree                                        47.204550    36.876661
I prefer not to answer                                 -2.102858    -3.490894
Master’s degree                                        50.912114    13.302677
No formal education past high school                   -3.376045    -2.663620
Professional degree                                    -9.980751    -6.265409
Some college/university study without earning a...     -9.181412    -1.347203
```

(continues on next page)

(continued from previous page)

```
Q9                                                 150-200,000  200-250,000  \
Q4
Bachelor’s degree                                   -40.822477   -25.860976
Doctoral degree                                      45.299501    26.885151
I prefer not to answer                               -1.850541    -0.449219
Master’s degree                                       7.191976    -4.223151
No formal education past high school                 -0.998639     0.495042
Professional degree                                  -3.373906     4.719230
Some college/university study without earning a...   -5.445914    -1.566077

Q9                                                 250-300,000  300-400,000  \
Q4
Bachelor’s degree                                    -6.305658    -2.771923
Doctoral degree                                      12.740618     9.500162
I prefer not to answer                                0.368073    -0.438136
Master’s degree                                      -3.417072    -4.555836
No formal education past high school                 -0.656232    -0.454987
Professional degree                                  -1.866615     0.705814
Some college/university study without earning a...   -0.863115    -1.985093

Q9                                                 400-500,000  500,000+
Q4
Bachelor’s degree                                    -2.533735  1.103247
Doctoral degree                                       3.240456 -1.297881
I prefer not to answer                                0.806209  3.469181
Master’s degree                                      -1.861235 -9.750340
No formal education past high school                  0.798756  5.448765
Professional degree                                  -0.572429  0.432044
Some college/university study without earning a...    0.121978  0.594983
```

It would be very hard to analyse this difference using the complete matrix, I propose we take the sum of differences for the high paying jobs and compare those. As a threshold of high-paying I chose to go for those who ‘earn six figures’.

```
degree_diff.loc[:,'100-125,000':'400-500,000'].sum(axis='columns').sort_values()
```

```
Q4
Bachelor’s degree                                                 -188.182578
Some college/university study without earning a bachelor’s degree  -20.266835
Professional degree                                                -16.634066
I prefer not to answer                                              -7.157366
No formal education past high school                                -6.855726
Master’s degree                                                     57.349472
Doctoral degree                                                    181.747100
dtype: float64
```

By the looks of it, it pays of to study longer and get more degrees, as a Masters degree is overrepresented by 57 persons and Doctoral degrees even by 181 persons. On the other side Bachelors or Professional degrees are underrepresented whilst no formal education is not particularly underperforming.

We can do the same for Q5: which field? were we analyse in which field the person works compared with their salary.

```
df = count_matrix('Q5', 'Q9')
df = df.loc[~(df==0).all(axis=1)]
F, p, deg, exp = scipy.stats.chi2_contingency(df)
print(f'F: {F}, p: {p}')
```

(continues on next page)

(continued from previous page)

```
diff = df-exp
diff.loc[:,'100-125,000':'400-500,000'].sum(axis='columns').sort_values()
```

```
F: 1090.8622667456245, p: 4.440076551687432e-121
```

```
Q5
Computer science (software engineering, etc.)                        -226.894272
Information technology, networking, or system administration          -48.957238
I never declared a major                                               -1.528181
Environmental science or geology                                       -1.521054
Medical or life sciences (biology, chemistry, medicine, etc.)           5.826010
Other                                                                  10.079966
Fine arts or performing arts                                           12.732902
Humanities (history, literature, philosophy, etc.)                     28.574735
Social sciences (anthropology, psychology, sociology, etc.)            31.130313
Mathematics or statistics                                              39.390480
Physics or astronomy                                                   47.878580
Engineering (non-computer focused)                                     51.042958
A business discipline (accounting, economics, finance, etc.)           52.244802
dtype: float64
```

We have a clear loser here, for some reason the computer science department seems to be underpayed or either not worth their money. on the other side there is a more gradual increase and most fields are over represented in the region of highly paid jobs.

What about Q6: your job description?

```
df = count_matrix('Q6', 'Q9')
df = df.loc[~(df==0).all(axis=1)]
F, p, deg, exp = scipy.stats.chi2_contingency(df)
print(f'F: {F}, p: {p}')
prof_diff = df-exp
prof_diff.loc[:,'100-125,000':'400-500,000'].sum(axis='columns').sort_values()
```

```
F: 5430.1040630013995, p: 0.0
```

```
Q6
Student                       -342.940761
Data Analyst                   -97.451682
Research Assistant             -58.433405
Business Analyst               -45.406507
Research Scientist             -15.196643
DBA/Database Engineer           -2.341629
Salesperson                     -2.173764
Statistician                    -2.140968
Developer Advocate              -0.872124
Marketing Analyst               -0.151468
Data Journalist                  2.603280
Software Engineer                4.167412
Data Engineer                   20.224966
Principal Investigator          22.407220
Other                           28.674185
Product/Project Manager         30.047897
Consultant                      56.967918
Chief Officer                   70.942316
```

(continues on next page)

(continued from previous page)

```
Manager                      101.868235
Data Scientist               229.205522
dtype: float64
```

Highly expected students score very bad here, which is a good confirmation. Something remarkable here is the difference between Data Analyst and Data Scientist, two jobs that seem to be similar have such a difference in representation in the high paid region.

To complete the analysis we also chose Q7: which sector?

```
df = count_matrix('Q7', 'Q9')
df = df.loc[~(df==0).all(axis=1)]
F, p, deg, exp = scipy.stats.chi2_contingency(df)
print(f'F: {F}, p: {p}')
diff = df-exp
diff.loc[:,'100-125,000':'400-500,000'].sum(axis='columns').sort_values()
```

```
F: 4518.06867284469, p: 0.0
```

```
Q7
I am a student                              -290.982760
Academics/Education                         -160.690323
Non-profit/Service                            -6.369823
Online Business/Internet-based Sales          0.009203
Broadcasting/Communications                   3.444617
Manufacturing/Fabrication                     4.242660
Military/Security/Defense                     8.049193
Retail/Sales                                 10.276752
Government/Public Service                    11.811848
Shipping/Transportation                      11.897725
Marketing/CRM                                12.086590
Energy/Mining                                13.813144
Hospitality/Entertainment/Sports             15.769849
Other                                        24.245965
Insurance/Risk Assessment                    33.588891
Medical/Pharmaceutical                       50.314149
Online Service/Internet-based Services       57.134487
Accounting/Finance                           74.047314
Computers/Technology                        127.310519
dtype: float64
```

Here we can again see the students, this time acompanied by the Academics/Education sector, which is understandable as it usually is a non-profit governmental oranization. Leading the charts we have the Computers/Technology sector which is currently booming.

38.5.1 Common skills

Aside from salary we also are interested in most common skills for a specific job title, therefore I took the averages of each choice for a multiple choice question. Here as example the combination of Q6: which job and Q16: what languages?

```
mean_choice_matrix('Q6', 'Q16')
```

```
                          Python         R       SQL      Bash      Java  \
Q6
Business Analyst        0.605085  0.401695  0.547458  0.049153  0.079661
Chief Officer           0.717131  0.274900  0.430279  0.191235  0.183267
Consultant              0.692845  0.413613  0.472949  0.116928  0.136126
DBA/Database Engineer   0.666667  0.307692  0.717949  0.188034  0.213675
Data Analyst            0.647059  0.484594  0.586134  0.079132  0.093137
Data Engineer           0.801418  0.248227  0.510638  0.223404  0.246454
Data Journalist         0.600000  0.400000  0.400000  0.200000  0.200000
Data Scientist          0.860265  0.429671  0.511542  0.193906  0.100031
Developer Advocate      0.611765  0.211765  0.482353  0.094118  0.341176
Manager                 0.681416  0.384956  0.455752  0.117257  0.126106
Marketing Analyst       0.505747  0.471264  0.505747  0.022989  0.022989
Not employed                 NaN       NaN       NaN       NaN       NaN
Other                   0.682041  0.287537  0.314033  0.116781  0.145240
Principal Investigator  0.759036  0.433735  0.277108  0.144578  0.144578
Product/Project Manager 0.720365  0.276596  0.455927  0.100304  0.161094
Research Assistant      0.776286  0.310962  0.192394  0.154362  0.161074
Research Scientist      0.780541  0.340541  0.196757  0.188108  0.142703
Salesperson             0.612500  0.275000  0.250000  0.025000  0.125000
Software Engineer       0.737179  0.123504  0.405128  0.167949  0.334188
Statistician            0.500000  0.822222  0.327778  0.077778  0.033333
Student                 0.697710  0.231679  0.214885  0.083969  0.225573

                        Javascript/Typescript  Visual Basic/VBA     C/C++  \
Q6
Business Analyst                     0.084746          0.189831  0.069492
Chief Officer                        0.306773          0.083665  0.139442
Consultant                           0.165794          0.111693  0.078534
DBA/Database Engineer                0.128205          0.068376  0.102564
Data Analyst                         0.098039          0.110644  0.079132
Data Engineer                        0.184397          0.053191  0.132979
Data Journalist                      0.400000          0.000000  0.100000
Data Scientist                       0.098492          0.043398  0.105571
Developer Advocate                   0.400000          0.094118  0.141176
Manager                              0.137168          0.139381  0.097345
Marketing Analyst                    0.045977          0.068966  0.034483
Not employed                              NaN               NaN       NaN
Other                                0.131501          0.087341  0.144259
Principal Investigator               0.216867          0.096386  0.265060
Product/Project Manager              0.206687          0.088146  0.133739
Research Assistant                   0.102908          0.038031  0.255034
Research Scientist                   0.088649          0.038919  0.301622
Salesperson                          0.137500          0.075000  0.137500
Software Engineer                    0.354274          0.036325  0.257692
Statistician                         0.050000          0.072222  0.094444
Student                              0.108397          0.027481  0.330153

                          MATLAB     Scala     Julia        Go   C#/.NET  \
Q6
```

(continues on next page)

(continued from previous page)

```
Business Analyst          0.030508  0.016949  0.003390  0.005085  0.042373
Chief Officer             0.091633  0.067729  0.015936  0.059761  0.103586
Consultant                0.062827  0.024433  0.010471  0.012216  0.082024
DBA/Database Engineer     0.051282  0.025641  0.000000  0.017094  0.179487
Data Analyst              0.067227  0.028711  0.005602  0.006303  0.039916
Data Engineer             0.081560  0.161348  0.008865  0.033688  0.086879
Data Journalist           0.200000  0.100000  0.000000  0.000000  0.200000
Data Scientist            0.076023  0.068637  0.014158  0.018159  0.040320
Developer Advocate        0.035294  0.035294  0.000000  0.023529  0.211765
Manager                   0.050885  0.024336  0.011062  0.011062  0.075221
Marketing Analyst         0.011494  0.000000  0.000000  0.000000  0.011494
Not employed                   NaN       NaN       NaN       NaN       NaN
Other                     0.100098  0.020608  0.005888  0.012758  0.073602
Principal Investigator    0.240964  0.024096  0.060241  0.024096  0.096386
Product/Project Manager   0.063830  0.039514  0.003040  0.012158  0.106383
Research Assistant        0.308725  0.013423  0.020134  0.000000  0.053691
Research Scientist        0.265946  0.024865  0.019459  0.003243  0.061622
Salesperson               0.087500  0.025000  0.000000  0.000000  0.025000
Software Engineer         0.073077  0.050855  0.008547  0.052137  0.185470
Statistician              0.166667  0.000000  0.016667  0.000000  0.038889
Student                   0.198473  0.015649  0.011832  0.008397  0.054198

                               PHP      Ruby  SAS/STATA      None     Other
Q6
Business Analyst          0.035593  0.006780   0.091525  0.054237  0.027119
Chief Officer             0.103586  0.043825   0.035857  0.023904  0.055777
Consultant                0.052356  0.012216   0.073298  0.031414  0.041885
DBA/Database Engineer     0.042735  0.008547   0.025641  0.008547  0.017094
Data Analyst              0.032213  0.014006   0.105042  0.011905  0.018908
Data Engineer             0.054965  0.015957   0.033688  0.003546  0.028369
Data Journalist           0.000000  0.000000   0.100000  0.000000  0.000000
Data Scientist            0.024007  0.008618   0.058172  0.001847  0.023084
Developer Advocate        0.141176  0.023529   0.011765  0.000000  0.058824
Manager                   0.044248  0.022124   0.077434  0.044248  0.028761
Marketing Analyst         0.034483  0.000000   0.057471  0.080460  0.000000
Not employed                   NaN       NaN        NaN       NaN       NaN
Other                     0.056919  0.012758   0.051030  0.056919  0.037291
Principal Investigator    0.096386  0.012048   0.060241  0.012048  0.072289
Product/Project Manager   0.091185  0.027356   0.024316  0.045593  0.027356
Research Assistant        0.071588  0.006711   0.046980  0.011186  0.031320
Research Scientist        0.043243  0.011892   0.043243  0.014054  0.043243
Salesperson               0.062500  0.012500   0.037500  0.100000  0.037500
Software Engineer         0.110684  0.038034   0.003419  0.001709  0.051282
Statistician              0.005556  0.016667   0.355556  0.016667  0.022222
Student                   0.051527  0.010305   0.035878  0.006489  0.015267
```

This does give us a lot of information, e.g. that 86% of all data scientists use python, yet it does not show correlation between answers, therefore we would need to go back to our original one hot encoded data and merge with the necessary info, it look like this.

```
df = info_df[['Q6','Q9']].join(answer_dfs['Q16'])
df.head()
```

```
              Q6         Q9  Python  R  SQL  Bash  Java  \
2          Other  10-20,000       0  0    1     0     0
3  Data Scientist   0-10,000       0  1    0     0     1
```

(continues on next page)

(continued from previous page)

```
5    Data Analyst   0-10,000       0  0     1      0      1
7           Other  10-20,000       0  1     0      0      0
8           Other   0-10,000       1  0     0      0      0

   Javascript/Typescript  Visual Basic/VBA  C/C++  MATLAB  Scala  Julia  Go  \
2                      0                 0      0       0      0      0   0
3                      0                 0      0       1      0      0   0
5                      0                 0      0       0      0      0   0
7                      0                 0      0       0      0      0   0
8                      0                 0      0       0      0      0   0

   C#/.NET  PHP  Ruby  SAS/STATA  None  Other
2        0    0     0          0     0      0
3        0    0     0          0     0      0
5        0    0     0          0     0      0
7        0    0     0          0     0      0
8        0    0     0          0     0      0
```

To deduce the correlation for all persons, we would not need Q6 or Q9, this will become necessary when we want to select subgroups. For now we calculate the percentage of all persons that have chosen each option

```
df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
```

```
Python                   0.735563
SQL                      0.403072
R                        0.323028
C/C++                    0.183162
Java                     0.174282
Javascript/Typescript    0.154644
Bash                     0.138441
MATLAB                   0.113812
C#/.NET                  0.077452
Visual Basic/VBA         0.062609
PHP                      0.054443
SAS/STATA                0.052045
Scala                    0.041545
Other                    0.030203
Go                       0.018601
Ruby                     0.016138
None                     0.015101
Julia                    0.010953
dtype: float64
```

We can see that options as Python, SQL and R are very popular, yet how do they correlate? Are the same persons who pick python also those who pick R? We use the numerical correlation to calculate this. Notice that I use the Spearman Rank as our data consists of 0 and 1, being non-normal distributed.

```
all_jobs_corr = df.corr('spearman')
all_jobs_corr
```

```
                         Python         R       SQL      Bash      Java  \
Python                 1.000000  0.077293  0.191304  0.188435  0.141813
R                      0.077293  1.000000  0.223527  0.032511 -0.034205
SQL                    0.191304  0.223527  1.000000  0.161086  0.135890
Bash                   0.188435  0.032511  0.161086  1.000000  0.078031
Java                   0.141813 -0.034205  0.135890  0.078031  1.000000
```

(continues on next page)

(continued from previous page)

```
Javascript/Typescript  0.125164 -0.039002  0.192323  0.146723  0.254773
Visual Basic/VBA       0.007550  0.098949  0.159062 -0.026907  0.024432
C/C++                  0.183621 -0.049046 -0.034188  0.082853  0.227691
MATLAB                 0.131117  0.030446 -0.047761  0.010577  0.064536
Scala                  0.095374  0.033991  0.117620  0.116862  0.165490
Julia                  0.039096  0.063129  0.013813  0.058785  0.005821
Go                     0.053180 -0.016114  0.048231  0.104544  0.073318
C#/.NET                0.046182 -0.039415  0.134615  0.000396  0.137888
PHP                    0.048012 -0.006318  0.157483  0.054351  0.177413
Ruby                   0.031308  0.014921  0.056242  0.103550  0.067257
SAS/STATA             -0.009036  0.198183  0.120958 -0.032248 -0.040728
None                  -0.206520 -0.085536 -0.101752 -0.049637 -0.056888
Other                  0.009641 -0.001240  0.001674  0.056457  0.026739

                       Javascript/Typescript  Visual Basic/VBA     C/C++  \
Python                              0.125164          0.007550  0.183621
R                                  -0.039002          0.098949 -0.049046
SQL                                 0.192323          0.159062 -0.034188
Bash                                0.146723         -0.026907  0.082853
Java                                0.254773          0.024432  0.227691
Javascript/Typescript               1.000000          0.049290  0.095921
Visual Basic/VBA                    0.049290          1.000000  0.020796
C/C++                               0.095921          0.020796  1.000000
MATLAB                             -0.004829          0.019424  0.260311
Scala                               0.060073          0.005185  0.004697
Julia                               0.025601          0.037063  0.049980
Go                                  0.118897          0.002042  0.048903
C#/.NET                             0.222775          0.122287  0.134850
PHP                                 0.307413          0.077126  0.111623
Ruby                                0.127312          0.011487  0.041745
SAS/STATA                          -0.048567          0.093618 -0.052866
None                               -0.052961         -0.032002 -0.058636
Other                               0.034497         -0.004965  0.019235

                         MATLAB     Scala     Julia        Go   C#/.NET  \
Python                 0.131117  0.095374  0.039096  0.053180  0.046182
R                      0.030446  0.033991  0.063129 -0.016114 -0.039415
SQL                   -0.047761  0.117620  0.013813  0.048231  0.134615
Bash                   0.010577  0.116862  0.058785  0.104544  0.000396
Java                   0.064536  0.165490  0.005821  0.073318  0.137888
Javascript/Typescript -0.004829  0.060073  0.025601  0.118897  0.222775
Visual Basic/VBA       0.019424  0.005185  0.037063  0.002042  0.122287
C/C++                  0.260311  0.004697  0.049980  0.048903  0.134850
MATLAB                 1.000000  0.003116  0.056403  0.003529  0.029772
Scala                  0.003116  1.000000  0.049863  0.077110  0.006505
Julia                  0.056403  0.049863  1.000000  0.091513  0.013769
Go                     0.003529  0.077110  0.091513  1.000000  0.042659
C#/.NET                0.029772  0.006505  0.013769  0.042659  1.000000
PHP                    0.046232  0.021619  0.010427  0.043077  0.108940
Ruby                   0.014027  0.066129  0.065590  0.092767  0.039859
SAS/STATA              0.015261  0.012633  0.045429 -0.008502 -0.026410
None                  -0.044376 -0.025780 -0.013031 -0.017048 -0.035878
Other                 -0.000043  0.014499  0.006897  0.012141  0.006953

                            PHP      Ruby  SAS/STATA      None     Other
Python                 0.048012  0.031308  -0.009036 -0.206520  0.009641
R                     -0.006318  0.014921   0.198183 -0.085536 -0.001240
```

(continues on next page)

(continued from previous page)

```
SQL                      0.157483  0.056242   0.120958 -0.101752  0.001674
Bash                     0.054351  0.103550  -0.032248 -0.049637  0.056457
Java                     0.177413  0.067257  -0.040728 -0.056888  0.026739
Javascript/Typescript    0.307413  0.127312  -0.048567 -0.052961  0.034497
Visual Basic/VBA         0.077126  0.011487   0.093618 -0.032002 -0.004965
C/C++                    0.111623  0.041745  -0.052866 -0.058636  0.019235
MATLAB                   0.046232  0.014027   0.015261 -0.044376 -0.000043
Scala                    0.021619  0.066129   0.012633 -0.025780  0.014499
Julia                    0.010427  0.065590   0.045429 -0.013031  0.006897
Go                       0.043077  0.092767  -0.008502 -0.017048  0.012141
C#/.NET                  0.108940  0.039859  -0.026410 -0.035878  0.006953
PHP                      1.000000  0.080351  -0.011212 -0.029713  0.031095
Ruby                     0.080351  1.000000   0.007042 -0.015859  0.034500
SAS/STATA               -0.011212  0.007042   1.000000 -0.029014 -0.005546
None                    -0.029713 -0.015859  -0.029014  1.000000 -0.021852
Other                    0.031095  0.034500  -0.005546 -0.021852  1.000000
```

As an example you can see that for those who chose python (the column called python) there is a 7.7% correlation with R and 19.1% with SQL, so a person who uses python is more likely to also know SQL (or Bash) rather than R. This is understandable as those languages are similar in usage.

Now we want to change things so we don't look towards all persons, but only data scientists, as I am a data scientist and want to know which languages I should learn more about.

```
df = df[df['Q6']=='Data Scientist']
data_science_corr = df.corr('spearman')
df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
```

```
Python                     0.860265
SQL                        0.511542
R                          0.429671
Bash                       0.193906
C/C++                      0.105571
Java                       0.100031
Javascript/Typescript      0.098492
MATLAB                     0.076023
Scala                      0.068637
SAS/STATA                  0.058172
Visual Basic/VBA           0.043398
C#/.NET                    0.040320
PHP                        0.024007
Other                      0.023084
Go                         0.018159
Julia                      0.014158
Ruby                       0.008618
None                       0.001847
dtype: float64
```

For the case of percentage that have chosen languages, things do not drastically change, although all percentages are up much more. You can see that the shell scripting language Bash has shifted upwards so a basic knowledge in Bash would not hurt.

For the correlation I opted to show the difference with the all persons correlation.

```
data_science_corr-all_jobs_corr
```

```
                         Python         R       SQL      Bash      Java  \
Python                 0.000000 -0.053854  0.018683 -0.035673 -0.042951
R                     -0.053854  0.000000 -0.002263 -0.030455  0.047379
SQL                    0.018683 -0.002263  0.000000  0.020711 -0.076892
Bash                  -0.035673 -0.030455  0.020711  0.000000 -0.052135
Java                  -0.042951  0.047379 -0.076892 -0.052135  0.000000
Javascript/Typescript -0.024721  0.027537 -0.086304 -0.039736 -0.051694
Visual Basic/VBA       0.008582  0.003023 -0.047632 -0.004962  0.025389
C/C++                 -0.091380  0.040190 -0.020833 -0.028407 -0.051798
MATLAB                -0.065754  0.020863  0.049269 -0.010267  0.033358
Scala                 -0.007031 -0.026163  0.006402 -0.009835  0.032070
Julia                 -0.020853  0.022313  0.030326 -0.005540  0.015005
Go                    -0.024962  0.005169 -0.035231 -0.031319 -0.049516
C#/.NET                0.000325  0.016383 -0.056398 -0.009901  0.038876
PHP                   -0.013800  0.032662 -0.084676 -0.044813  0.018255
Ruby                  -0.012939 -0.008402 -0.025087 -0.073477 -0.020655
SAS/STATA              0.021968 -0.063263  0.021932  0.000158  0.019228
None                   0.099795  0.048202  0.057734  0.028540  0.042548
Other                 -0.006802  0.033430  0.021428 -0.022983 -0.009677

                       Javascript/Typescript  Visual Basic/VBA     C/C++  \
Python                             -0.024721          0.008582 -0.091380
R                                   0.027537          0.003023  0.040190
SQL                                -0.086304         -0.047632 -0.020833
Bash                               -0.039736         -0.004962 -0.028407
Java                               -0.051694          0.025389 -0.051798
Javascript/Typescript               0.000000         -0.003094  0.005652
Visual Basic/VBA                   -0.003094          0.000000 -0.020233
C/C++                               0.005652         -0.020233  0.000000
MATLAB                              0.046422          0.027772  0.000168
Scala                               0.005440         -0.003260  0.024847
Julia                              -0.012755          0.014130 -0.014849
Go                                 -0.040080         -0.008382  0.039396
C#/.NET                            -0.022725         -0.020037 -0.001594
PHP                                -0.096084          0.037497 -0.028092
Ruby                               -0.057557          0.017683  0.012913
SAS/STATA                           0.023788          0.027697  0.027393
None                                0.038744          0.022840  0.043858
Other                              -0.016523          0.012460 -0.012017

                         MATLAB     Scala     Julia        Go   C#/.NET  \
Python                -0.065754 -0.007031 -0.020853 -0.024962  0.000325
R                      0.020863 -0.026163  0.022313  0.005169  0.016383
SQL                    0.049269  0.006402  0.030326 -0.035231 -0.056398
Bash                  -0.010267 -0.009835 -0.005540 -0.031319 -0.009901
Java                   0.033358  0.032070  0.015005 -0.049516  0.038876
Javascript/Typescript  0.046422  0.005440 -0.012755 -0.040080 -0.022725
Visual Basic/VBA       0.027772 -0.003260  0.014130 -0.008382 -0.020037
C/C++                  0.000168  0.024847 -0.014849  0.039396 -0.001594
MATLAB                 0.000000  0.010878  0.046837  0.018341  0.029507
Scala                  0.010878  0.000000 -0.030876  0.004490  0.005925
Julia                  0.046837 -0.030876  0.000000  0.028763 -0.011845
Go                     0.018341  0.004490  0.028763  0.000000 -0.023663
C#/.NET                0.029507  0.005925 -0.011845 -0.023663  0.000000
PHP                    0.045341 -0.000573  0.004817 -0.019231  0.053129
Ruby                   0.047175 -0.038760  0.007792 -0.055572 -0.008187
SAS/STATA              0.012680 -0.007288  0.002699 -0.005601  0.008893
```

(continues on next page)

(continued from previous page)

```
None                    0.032038  0.014104  0.007876  0.011198  0.027062
Other                  -0.005384  0.016729  0.009378  0.013002  0.013636

                             PHP      Ruby  SAS/STATA      None     Other
Python                 -0.013800 -0.012939   0.021968  0.099795 -0.006802
R                       0.032662 -0.008402  -0.063263  0.048202  0.033430
SQL                    -0.084676 -0.025087   0.021932  0.057734  0.021428
Bash                   -0.044813 -0.073477   0.000158  0.028540 -0.022983
Java                    0.018255 -0.020655   0.019228  0.042548 -0.009677
Javascript/Typescript -0.096084 -0.057557   0.023788  0.038744 -0.016523
Visual Basic/VBA        0.037497  0.017683   0.027697  0.022840  0.012460
C/C++                  -0.028092  0.012913   0.027393  0.043858 -0.012017
MATLAB                  0.045341  0.047175   0.012680  0.032038 -0.005384
Scala                  -0.000573 -0.038760  -0.007288  0.014104  0.016729
Julia                   0.004817  0.007792   0.002699  0.007876  0.009378
Go                     -0.019231 -0.055572  -0.005601  0.011198  0.013002
C#/.NET                 0.053129 -0.008187   0.008893  0.027062  0.013636
PHP                     0.000000  0.035548   0.015185  0.022966  0.011745
Ruby                    0.035548  0.000000  -0.001761  0.011849  0.017689
SAS/STATA               0.015185 -0.001761   0.000000  0.018324  0.037394
None                    0.022966  0.011849   0.018324  0.000000  0.015240
Other                   0.011745  0.017689   0.037394  0.015240  0.000000
```

In the Python column we can see that generic non Data Science languages such as C/C++ and Java are falling, yet the correlation with Bash is also negative, this indicates by selecting Data Science profiles we have on average more people choosing for Bash, but NOT in combination with Python. Although results are somewhat expected, there do not seem to be any drastic changes.

To shake things up more, we apply a second filter, where we only take the persons who earn more than 100k.

```
df = df.loc[('100-125,000'<df.Q9) & (df.Q9<'500,000+')]
high_paying_job_corr = df.corr('spearman')
df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
```

```
Python                   0.896635
SQL                      0.608173
R                        0.459135
Bash                     0.274038
Scala                    0.105769
Java                     0.093750
Javascript/Typescript    0.088942
C/C++                    0.067308
SAS/STATA                0.062500
MATLAB                   0.055288
Visual Basic/VBA         0.043269
C#/.NET                  0.043269
Other                    0.031250
PHP                      0.021635
Julia                    0.019231
Go                       0.019231
Ruby                     0.009615
None                     0.002404
dtype: float64
```

For the average of choices we can now see that Scala - a language used for big data - shootsj up the ranks, indicating that having a data engineering language in your knowledge base is good for your salary.

To compare the correlation of high paying data science jobs I took the difference with correlation of all jobs.

```
high_paying_job_corr-all_jobs_corr
```

```
                         Python         R       SQL      Bash      Java  \
Python                 0.000000 -0.065523 -0.027102 -0.032937 -0.140966
R                     -0.065523  0.000000  0.031806 -0.014573  0.052306
SQL                   -0.027102  0.031806  0.000000  0.011898 -0.063562
Bash                  -0.032937 -0.014573  0.011898  0.000000 -0.016784
Java                  -0.140966  0.052306 -0.063562 -0.016784  0.000000
Javascript/Typescript -0.046816  0.090043 -0.131819 -0.016832 -0.065553
Visual Basic/VBA      -0.012961 -0.010383 -0.060971  0.055178  0.028768
C/C++                 -0.155440  0.032570 -0.064647 -0.032800 -0.149517
MATLAB                -0.083528  0.021053  0.048020  0.005863  0.038082
Scala                  0.021397 -0.052843  0.030335  0.022321  0.126133
Julia                  0.008448  0.053728  0.062726  0.012154 -0.050859
Go                    -0.005636 -0.042652 -0.007548  0.005638 -0.058305
C#/.NET               -0.051593 -0.014271 -0.133335 -0.078079 -0.003621
PHP                   -0.051796  0.068248 -0.105817 -0.034582  0.114871
Ruby                   0.002147 -0.006841 -0.027615 -0.108861  0.070073
SAS/STATA              0.031462 -0.057440 -0.035771  0.007200 -0.008248
None                   0.061944  0.040308  0.040596  0.019477  0.041100
Other                  0.005959  0.057555  0.029281  0.019045  0.010292

                       Javascript/Typescript  Visual Basic/VBA     C/C++  \
Python                             -0.046816         -0.012961 -0.155440
R                                   0.090043         -0.010383  0.032570
SQL                                -0.131819         -0.060971 -0.064647
Bash                               -0.016832          0.055178 -0.032800
Java                               -0.065553          0.028768 -0.149517
Javascript/Typescript               0.000000         -0.032728  0.022367
Visual Basic/VBA                   -0.032728          0.000000 -0.030771
C/C++                               0.022367         -0.030771  0.000000
MATLAB                              0.040091         -0.019176 -0.157380
Scala                              -0.030239         -0.001491  0.027699
Julia                              -0.007864          0.019187 -0.017737
Go                                 -0.039671         -0.031821 -0.016661
C#/.NET                             0.001309         -0.051377 -0.050517
PHP                                -0.179745          0.053665 -0.019681
Ruby                                0.014972         -0.032441  0.030100
SAS/STATA                           0.037665          0.046707  0.102410
None                                0.037624          0.021562  0.045449
Other                              -0.042081          0.034672 -0.012343

                         MATLAB     Scala     Julia        Go   C#/.NET  \
Python                -0.083528  0.021397  0.008448 -0.005636 -0.051593
R                      0.021053 -0.052843  0.053728 -0.042652 -0.014271
SQL                    0.048020  0.030335  0.062726 -0.007548 -0.133335
Bash                   0.005863  0.022321  0.012154  0.005638 -0.078079
Java                   0.038082  0.126133 -0.050859 -0.058305 -0.003621
Javascript/Typescript  0.040091 -0.030239 -0.007864 -0.039671  0.001309
Visual Basic/VBA      -0.019176 -0.001491  0.019187 -0.031821 -0.051377
C/C++                 -0.157380  0.027699 -0.017737 -0.016661 -0.050517
MATLAB                 0.000000  0.016286 -0.013690 -0.037404 -0.029523
Scala                  0.016286  0.000000 -0.041107 -0.068354 -0.041228
Julia                 -0.013690 -0.041107  0.000000  0.016330 -0.043548
Go                    -0.037404 -0.068354  0.016330  0.000000 -0.072438
C#/.NET               -0.029523 -0.041228 -0.043548 -0.072438  0.000000
```

(continues on next page)

(continued from previous page)

```
PHP                      0.062386 -0.019036 -0.031250 -0.063900  0.021852
Ruby                     0.069920 -0.019920  0.099978  0.072801 -0.060814
SAS/STATA                0.009181 -0.069142  0.063037  0.044657 -0.028500
None                     0.032500  0.008898  0.006157  0.010174  0.025439
Other                    0.017045  0.013578 -0.032047 -0.037291  0.022755

                              PHP      Ruby  SAS/STATA      None     Other
Python                  -0.051796  0.002147   0.031462  0.061944  0.005959
R                        0.068248 -0.006841  -0.057440  0.040308  0.057555
SQL                     -0.105817 -0.027615  -0.035771  0.040596  0.029281
Bash                    -0.034582 -0.108861   0.007200  0.019477  0.019045
Java                     0.114871  0.070073  -0.008248  0.041100  0.010292
Javascript/Typescript   -0.179745  0.014972   0.037665  0.037624 -0.042081
Visual Basic/VBA         0.053665 -0.032441   0.046707  0.021562  0.034672
C/C++                   -0.019681  0.030100   0.102410  0.045449 -0.012343
MATLAB                   0.062386  0.069920   0.009181  0.032500  0.017045
Scala                   -0.019036 -0.019920  -0.069142  0.008898  0.013578
Julia                   -0.031250  0.099978   0.063037  0.006157 -0.032047
Go                      -0.063900  0.072801   0.044657  0.010174 -0.037291
C#/.NET                  0.021852 -0.060814  -0.028500  0.025439  0.022755
PHP                      0.000000  0.074312  -0.027184  0.022413 -0.057803
Ruby                     0.074312  0.000000  -0.032483  0.011022 -0.052197
SAS/STATA               -0.027184 -0.032483   0.000000  0.016340  0.016248
None                     0.022413  0.011022   0.016340  0.000000  0.013036
Other                   -0.057803 -0.052197   0.016248  0.013036  0.000000
```

Again as I mainly use python I will be looking at the Python column, you can see that Scala is indeed correlated with Python and Java or C/C++ is not a must at all.

In a similar fashion we evaluate the influence of machine learning toolkits, where we first see the average choice of all persons.

```
df = info_df[['Q6','Q9']].join(answer_dfs['Q19'])
df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
```

```
Scikit-Learn    0.588113
TensorFlow      0.476635
Keras           0.393026
randomForest    0.290168
Xgboost         0.278761
PyTorch         0.183226
None            0.128071
Caret           0.114395
Spark MLlib     0.108108
lightgbm        0.105256
H20             0.069674
Fastai          0.055350
Caffe           0.053730
catboost        0.051656
Prophet         0.039406
CNTK            0.035647
Mxnet           0.030916
mlr             0.027027
Other           0.025731
dtype: float64
```

Scikit-learn or sklearn (the one we sometimes use) is chosen the most often, problably because of it’s ease of use and effectiveness. Now we would like to see the choice of data scientists

```
df = df[df['Q6']=='Data Scientist']
df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
```

```
Scikit-Learn     0.791936
TensorFlow       0.607879
Keras            0.567251
Xgboost          0.534626
randomForest     0.474608
PyTorch          0.251154
Caret            0.224685
lightgbm         0.224685
Spark MLlib      0.219452
H20              0.157279
catboost         0.114805
Fastai           0.101878
Prophet          0.098492
Caffe            0.063096
CNTK             0.060634
Mxnet            0.055094
mlr              0.052324
Other            0.029240
None             0.023392
dtype: float64
```

No particular shifts although None has dropped to the last place, indicating that knowledge of Machine Learning is essential for a Data Scientist.

What happens when we only ask the high paying data scientists?

```
df = df.loc[('100-125,000'<df.Q9) & (df.Q9<'500,000+')]
df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
```

```
Scikit-Learn     0.838942
TensorFlow       0.639423
Xgboost          0.598558
Keras            0.596154
randomForest     0.504808
Spark MLlib      0.310096
PyTorch          0.278846
Caret            0.245192
lightgbm         0.245192
H20              0.216346
Prophet          0.120192
Fastai           0.115385
catboost         0.110577
Mxnet            0.086538
Caffe            0.086538
CNTK             0.079327
mlr              0.055288
Other            0.036058
None             0.021635
dtype: float64
```

Nothing in particular, except that all percentages have increased, to conclude your choice of machine learning library is not that important!

38.5.2 Time spend

I would also like to know how other data scientists spend their time, in the same fashion we analyse this

```
df = info_df[['Q6','Q9']].join(answer_dfs['Q34'])
df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
```

```
Gathering data                                                      0.762525
Cleaning data                                                       0.762525
Visualizing data                                                    0.762525
Model building/model selection                                      0.762525
Putting the model into production                                   0.762525
Finding insights in the data and communicating with stakeholders    0.762525
dtype: float64
```

Looks like we made a mistake, we one hot encoded all questions but this is a numerical question, we need som more manipulations.

```
df = train_df[train_df.columns[train_df.columns.str.contains('Q34')][:-1]].fillna(0).
↪astype(float)
df = info_df[['Q6','Q9']].join(df).rename(columns=questions[questions.index.str.
↪contains('Q34')].apply(lambda x: x.split(' - ')[-1]).to_dict())
df
```

```
                       Q6           Q9  Gathering data  Cleaning data  \
2                   Other    10-20,000             0.0            0.0
3          Data Scientist     0-10,000             2.0            3.0
5            Data Analyst     0-10,000            10.0           10.0
7                   Other    10-20,000             0.0           30.0
8                   Other     0-10,000            20.0           30.0
...                   ...          ...             ...            ...
23844   Software Engineer   90-100,000            10.0           30.0
23845             Student     0-10,000             0.0            0.0
23854  Research Assistant    10-20,000             0.0            0.0
23855       Chief Officer  250-300,000             0.0            0.0
23857   Software Engineer    10-20,000             0.0            0.0

       Visualizing data  Model building/model selection  \
2                   0.0                             0.0
3                  20.0                            50.0
5                  20.0                            10.0
7                  50.0                             0.0
8                  20.0                            20.0
...                 ...                             ...
23844               5.0                            40.0
23845               0.0                             0.0
23854               0.0                             0.0
23855               0.0                             0.0
23857               0.0                             0.0

       Putting the model into production  \
2                                    0.0
3                                   20.0
5                                   20.0
7                                    0.0
8                                    5.0
...                                  ...
```

(continues on next page)

(continued from previous page)

```
23844                          10.0
23845                           0.0
23854                           0.0
23855                           0.0
23857                           0.0

       Finding insights in the data and communicating with stakeholders
2                                                    0.0
3                                                    0.0
5                                                   23.0
7                                                   20.0
8                                                    5.0
...                                                  ...
23844                                                5.0
23845                                                0.0
23854                                                0.0
23855                                                0.0
23857                                                0.0

[15429 rows x 8 columns]
```

This looks better, now for each answer we have a value between 0 and 100%, we need to check if they have filled in this answer though

```
df = df[~(df.drop(columns=['Q6', 'Q9']).sum(axis='columns')==0)]
df
```

```
                            Q6           Q9  Gathering data  Cleaning data  \
3               Data Scientist     0-10,000             2.0            3.0
5                 Data Analyst     0-10,000            10.0           10.0
7                        Other    10-20,000             0.0           30.0
8                        Other     0-10,000            20.0           30.0
10           Software Engineer    20-30,000            55.0           10.0
...                        ...          ...             ...            ...
23823        Software Engineer     0-10,000            20.0           20.0
23824       Research Scientist    70-80,000            10.0           10.0
23836   Product/Project Manager   10-20,000            10.0            0.0
23841                  Student    10-20,000            20.0            5.0
23844        Software Engineer   90-100,000            10.0           30.0

       Visualizing data  Model building/model selection  \
3                  20.0                            50.0
5                  20.0                            10.0
7                  50.0                             0.0
8                  20.0                            20.0
10                 20.0                             5.0
...                 ...                             ...
23823              20.0                             5.0
23824              40.0                            10.0
23836              10.0                            10.0
23841              15.0                            40.0
23844               5.0                            40.0

       Putting the model into production  \
3                                   20.0
5                                   20.0
```

(continues on next page)

(continued from previous page)

```
7                                  0.0
8                                  5.0
10                                 0.0
...                                ...
23823                             20.0
23824                              0.0
23836                             20.0
23841                             20.0
23844                             10.0

       Finding insights in the data and communicating with stakeholders
3                                                                 0.0
5                                                                23.0
7                                                                20.0
8                                                                 5.0
10                                                               10.0
...                                                               ...
23823                                                            10.0
23824                                                            10.0
23836                                                            50.0
23841                                                             0.0
23844                                                             5.0

[11535 rows x 8 columns]
```

much better! we have the percentages and dropped the rows where nothing was filled in

```
time_all = df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
time_all
```

```
Cleaning data                                                       23.743813
Model building/model selection                                      21.038863
Gathering data                                                      17.262623
Visualizing data                                                    13.837350
Finding insights in the data and communicating with stakeholders    11.864940
Putting the model into production                                    9.054319
dtype: float64
```

In the beginning of my course I showed a graph on how a data scientists time is divided, this should give another view on it, most of it is data cleaning and model selection, visualization and insights are equally important but get more hands-on time.

How are these relations when looking at Data Scientists?

```
df = df[df['Q6']=='Data Scientist']
time_scientist = df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
time_scientist
```

```
Cleaning data                                                       25.341431
Model building/model selection                                      20.309071
Gathering data                                                      16.010219
Visualizing data                                                    12.916368
Finding insights in the data and communicating with stakeholders    12.675780
Putting the model into production                                   10.362675
dtype: float64
```

I would not say things have changed much, as expected as many of the persons are data scientists. Does this stay when we filter on the higher paid jobs?

```
df = df.loc[('100-125,000'<df.Q9) & (df.Q9<'500,000+')]
time_high_pay = df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
time_high_pay
```

```
Cleaning data                                                       25.435135
Model building/model selection                                      20.500000
Gathering data                                                      16.502703
Finding insights in the data and communicating with stakeholders    13.413514
Visualizing data                                                    11.916216
Putting the model into production                                   10.729730
dtype: float64
```

There seems to be a little change, we can see that data visualization is less important, this is understandable as this is rather a task for a data analyst that creates reports using graphs.

So if I want to specialize myself in Data Science I should not put the focus on data visualizations.

To end this analysis I would like to pick Q42: Quality control of products. Again we do the same analysis

```
df = info_df[['Q6','Q9']].join(answer_dfs['Q42'])
df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
```

```
Metrics that consider accuracy                                                0.
↪419924
Revenue and/or business goals                                                 0.
↪290492
Not applicable (I am not involved with an organization that builds ML models) 0.
↪136107
Metrics that consider unfair bias                                             0.
↪133126
Other                                                                         0.
↪013740
dtype: float64
```

```
df = df[df['Q6']=='Data Scientist']
df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
```

```
Metrics that consider accuracy                                                0.
↪620499
Revenue and/or business goals                                                 0.
↪517082
Metrics that consider unfair bias                                             0.
↪184057
Not applicable (I am not involved with an organization that builds ML models) 0.
↪035703
Other                                                                         0.
↪019083
dtype: float64
```

```
df = df.loc[('100-125,000'<df.Q9) & (df.Q9<'500,000+')]
df.drop(columns=['Q6','Q9']).mean().sort_values(ascending=False)
```

```
Metrics that consider accuracy                                                0.
↪673077
```

(continues on next page)

(continued from previous page)

```
Revenue and/or business goals                                         0.
↪634615
Metrics that consider unfair bias                                     0.
↪201923
Not applicable (I am not involved with an organization that builds ML models)    0.
↪024038
Other                                                                 0.
↪021635
dtype: float64
```

You can see that Data Scientists focus more on Metrics that consider unfair bias, as this is often an issue in Data Science, when reporting data biases might not be that critical (or might even help you) but in Data Science - when exploring new ideas - it is important to not have a bias that might disrupt your machine learning algorithm.

38.5.3 Age vs experience

Something we can really do much about, but it would be nice to see if it is never too late to change careers. For both age and experience we create a cross-tabulation and calculate a contingency test.

```
age_crosstab_df = info_df.groupby(['Q9', 'Q2']).size().unstack()
age_crosstab_df
```

```
Q2            18-21  22-24  25-29  30-34  35-39  40-44  45-49  50-54  55-59  \
Q9
0-10,000       1119   1574   1057    360    149     69     36     19      7
10-20,000       100    427    747    329    154     94     41     25      9
20-30,000        40    220    539    302    145     73     42     19      8
30-40,000        20    168    371    262    137     83     37     19      7
40-50,000        18    112    318    223    134     72     44     24      8
50-60,000        16     93    291    214    131     80     42     27     15
60-70,000         3     71    201    189    110     73     36     25     11
70-80,000         6     63    166    173     91     66     52     29     17
80-90,000         7     44    140    119     72     49     35     23      8
90-100,000       11     34    112    146    108     65     28     30     19
100-125,000      13     40    154    209    168     89     56     43     37
125-150,000       1     20     85    128    101     68     52     41     18
150-200,000       2      9     49    101     84     69     49     41     28
200-250,000       1      2     17     37     18     23     27     18     18
250-300,000       2      6      8      9      9      8     12      9      5
300-400,000       0      4      6      7     11      5      6      5      2
400-500,000       0      3      4      4      5      3      2      0      1
500,000+          4      4     13      7     10      0      5      6      4

Q2            60-69  70-79  80+
Q9
0-10,000          5      2    1
10-20,000         8      2    1
20-30,000         6      0    1
30-40,000        11      4    0
40-50,000         8      3    1
50-60,000         8      2    0
60-70,000         9      1    0
70-80,000        12      2    0
80-90,000         8      0    1
90-100,000       11      1    1
```

(continues on next page)

(continued from previous page)

```
100-125,000      31       2    1
125-150,000      17       1    1
150-200,000      21       4    0
200-250,000      11       0    0
250-300,000       7       0    0
300-400,000       4       1    1
400-500,000       0       1    0
500,000+          3       0    7
```

just by looking at it you can see a correlation, but just for significance we do the statistics

```
F, p, deg, exp = scipy.stats.chi2_contingency(age_crosstab_df)
print(f'F: {F}, p: {p}')
diff = age_crosstab_df-exp
age_diff = diff.loc['100-125,000':'400-500,000'].sum()#.sort_values()
age_diff
```

```
F: 6918.154466333605, p: 0.0
```

```
Q2
18-21   -171.372999
22-24   -320.210902
25-29   -274.517013
30-34    101.264502
35-39    167.356860
40-44    126.864346
45-49    119.917428
50-54    100.712165
55-59     77.992806
60-69     65.859032
70-79      5.368527
80+        0.765247
dtype: float64
```

Again I took the high paying jobs and you can see that from the age of 30 there is an overrepresentation in high paying jobs. We can safely say that by increasing age you are more likely to end up in the high paying salary sector although it reverts back around the age of 55.

Now for the experience

```
exp_crosstab_df = info_df.groupby(['Q9', 'Q8']).size().unstack()
exp_crosstab_df
```

```
Q8              0-1  1-2  2-3  3-4  4-5  5-10  10-15  15-20  20-25  25-30  30 +
Q9
0-10,000       1806  852  503  339  236   272    170    135     14      4     5
10-20,000       432  431  289  186  148   259    114     36     19     12     5
20-30,000       262  290  215  128  114   225     99     31     20      6     3
30-40,000       246  215  155  105   81   169     82     39     14      3     7
40-50,000       191  193  125   83  100   144     69     33     18      5     2
50-60,000       221  157  115   91   73   118     66     49     11      4    14
60-70,000       151  139  104   65   57   103     64     22     13      3     8
70-80,000       128  129   92   59   50    92     55     35     21      8     8
80-90,000       103   84   68   42   45    85     40     17      9      5     8
90-100,000       89   94   68   50   44    97     64     24     21      9     6
100-125,000     113  106  109   58   76   162    105     49     28     17    19
```

(continues on next page)

(continued from previous page)

```
125-150,000    55   72   63   43   51    92     63     45     26     12    11
150-200,000    47   38   33   37   46    85     72     37     26     20    16
200-250,000    11   12   12   10   14    32     23     25     17      7     9
250-300,000     8    6    4    2    3    11     17     11      6      3     4
300-400,000     9    4    1    4    4     9      6      2      5      2     6
400-500,000     0    1    5    2    2     4      7      0      1      0     1
500,000+        7    3    4    4    1    14      7      5      2      4    12
```

A less obvious correlation, we can use the F values to compare.

```
F, p, deg, exp = scipy.stats.chi2_contingency(exp_crosstab_df)
print(f'F: {F}, p: {p}')
diff = exp_crosstab_df-exp
exp_diff = diff.loc['100-125,000':'400-500,000'].sum() #.sort_values()
exp_diff
```

```
F: 2522.63511999856, p: 0.0
```

```
Q8
0-1     -301.217156
1-2     -157.483033
2-3      -48.686185
3-4      -27.510193
4-5       35.358432
5-10     118.191428
10-15    135.444994
15-20     85.522504
20-25     70.979157
25-30     43.603009
30 +      45.797043
dtype: float64
```

The F value is indeed lower, indicating that the correlation between age and salary is stronger than age and experience. The expected experience level to reach the high paying jobs seems to be around the 5 year mark.

38.6 Visualisation

Although data scientists spend less time visualizing, I'm still going to make the effort here, a little refreshment, we created a mean matrix between 2 informative questions.

```
mean_matrix('Q6','Q9')
```

```
Q9                        0-10,000  10-20,000  20-30,000  30-40,000  40-50,000  \
Q6
Business Analyst          0.161017   0.176271   0.108475   0.079661   0.077966
Chief Officer             0.083665   0.087649   0.059761   0.027888   0.035857
Consultant                0.109948   0.095986   0.089005   0.094241   0.082024
DBA/Database Engineer     0.205128   0.153846   0.170940   0.051282   0.076923
Data Analyst              0.260504   0.151961   0.094538   0.079132   0.070028
Data Engineer             0.182624   0.145390   0.097518   0.070922   0.081560
Data Journalist           0.300000   0.200000   0.100000   0.000000   0.000000
Data Scientist            0.141890   0.106802   0.083102   0.079717   0.080640
Developer Advocate        0.223529   0.188235   0.047059   0.141176   0.082353
```

(continues on next page)

(continued from previous page)

```
Manager                 0.068584   0.061947   0.070796   0.086283   0.055310
Marketing Analyst       0.183908   0.114943   0.137931   0.080460   0.091954
Not employed                 NaN        NaN        NaN        NaN        NaN
Other                   0.205103   0.131501   0.091266   0.076546   0.064769
Principal Investigator  0.048193   0.060241   0.048193   0.024096   0.084337
Product/Project Manager 0.100304   0.100304   0.103343   0.057751   0.079027
Research Assistant      0.382550   0.203579   0.149888   0.098434   0.051454
Research Scientist      0.140541   0.139459   0.100541   0.112432   0.102703
Salesperson             0.237500   0.150000   0.125000   0.050000   0.050000
Software Engineer       0.252991   0.144017   0.113675   0.080342   0.059829
Statistician            0.227778   0.161111   0.100000   0.038889   0.033333
Student                 0.759924   0.101527   0.057634   0.033969   0.014885

Q9                      50-60,000  60-70,000  70-80,000  80-90,000  \
Q6
Business Analyst         0.091525   0.077966   0.054237   0.050847
Chief Officer            0.039841   0.079681   0.051793   0.023904
Consultant               0.069808   0.068063   0.036649   0.045375
DBA/Database Engineer    0.059829   0.042735   0.051282   0.017094
Data Analyst             0.069328   0.067227   0.055322   0.046218
Data Engineer            0.060284   0.065603   0.040780   0.044326
Data Journalist          0.000000   0.000000   0.000000   0.000000
Data Scientist           0.070483   0.059095   0.063712   0.045860
Developer Advocate       0.058824   0.023529   0.023529   0.035294
Manager                  0.070796   0.046460   0.055310   0.046460
Marketing Analyst        0.080460   0.034483   0.057471   0.022989
Not employed                  NaN        NaN        NaN        NaN
Other                    0.076546   0.058881   0.053974   0.036310
Principal Investigator   0.096386   0.048193   0.036145   0.048193
Product/Project Manager  0.072948   0.066869   0.057751   0.063830
Research Assistant       0.049217   0.020134   0.015660   0.011186
Research Scientist       0.099459   0.063784   0.047568   0.031351
Salesperson              0.062500   0.062500   0.075000   0.025000
Software Engineer        0.061538   0.038034   0.044444   0.026068
Statistician             0.050000   0.061111   0.083333   0.061111
Student                  0.007634   0.003435   0.004198   0.002290

Q9                      90-100,000  100-125,000  125-150,000  150-200,000  \
Q6
Business Analyst          0.052542     0.037288     0.010169     0.013559
Chief Officer             0.035857     0.079681     0.079681     0.119522
Consultant                0.062827     0.101222     0.048866     0.055846
DBA/Database Engineer     0.051282     0.025641     0.059829     0.017094
Data Analyst              0.032213     0.046919     0.013305     0.005602
Data Engineer             0.035461     0.065603     0.046099     0.042553
Data Journalist           0.000000     0.100000     0.200000     0.000000
Data Scientist            0.055402     0.082179     0.058787     0.044629
Developer Advocate        0.047059     0.047059     0.035294     0.035294
Manager                   0.066372     0.110619     0.081858     0.097345
Marketing Analyst         0.057471     0.057471     0.068966     0.011494
Not employed                   NaN          NaN          NaN          NaN
Other                     0.030422     0.063788     0.039254     0.039254
Principal Investigator    0.072289     0.132530     0.084337     0.132530
Product/Project Manager   0.066869     0.075988     0.063830     0.057751
Research Assistant        0.004474     0.004474     0.004474     0.000000
Research Scientist        0.036757     0.043243     0.029189     0.023784
Salesperson               0.025000     0.037500     0.050000     0.012500
```

(continues on next page)

(continued from previous page)

```
Software Engineer          0.034615     0.061111     0.032051     0.026068
Statistician               0.055556     0.050000     0.033333     0.016667
Student                    0.004198     0.004198     0.002290     0.001145

Q9                        200-250,000  250-300,000  300-400,000  400-500,000  \
Q6
Business Analyst             0.001695     0.000000     0.000000     0.000000
Chief Officer                0.067729     0.035857     0.023904     0.015936
Consultant                   0.022688     0.006981     0.001745     0.001745
DBA/Database Engineer        0.008547     0.008547     0.000000     0.000000
Data Analyst                 0.001401     0.000700     0.001401     0.002101
Data Engineer                0.012411     0.003546     0.005319     0.000000
Data Journalist              0.000000     0.100000     0.000000     0.000000
Data Scientist               0.014158     0.004925     0.003693     0.001847
Developer Advocate           0.011765     0.000000     0.000000     0.000000
Manager                      0.046460     0.017699     0.008850     0.002212
Marketing Analyst            0.000000     0.000000     0.000000     0.000000
Not employed                      NaN          NaN          NaN          NaN
Other                        0.012758     0.007851     0.003925     0.000981
Principal Investigator       0.036145     0.012048     0.012048     0.000000
Product/Project Manager      0.018237     0.009119     0.006079     0.000000
Research Assistant           0.000000     0.000000     0.000000     0.000000
Research Scientist           0.012973     0.006486     0.006486     0.001081
Salesperson                  0.012500     0.000000     0.000000     0.000000
Software Engineer            0.010684     0.005556     0.003846     0.002137
Statistician                 0.005556     0.011111     0.005556     0.005556
Student                      0.000763     0.000000     0.000382     0.000000

Q9                        500,000+
Q6
Business Analyst          0.006780
Chief Officer             0.051793
Consultant                0.006981
DBA/Database Engineer     0.000000
Data Analyst              0.002101
Data Engineer             0.000000
Data Journalist           0.000000
Data Scientist            0.003078
Developer Advocate        0.000000
Manager                   0.006637
Marketing Analyst         0.000000
Not employed                   NaN
Other                     0.006869
Principal Investigator    0.024096
Product/Project Manager   0.000000
Research Assistant        0.004474
Research Scientist        0.002162
Salesperson               0.025000
Software Engineer         0.002991
Statistician              0.000000
Student                   0.001527
```

What I was thinking about would be a bar chart where each job title is a row and the distribution of each salary is shown, below the example

```
df = mean_matrix('Q6', 'Q9').dropna().cumsum(axis='columns')
```

(continues on next page)

(continued from previous page)

```
for idx, col in enumerate(df.columns[::-1]):
    sns.barplot(x=df[col], y=df.index, color=sns.color_palette('colorblind')[idx%10])

plt.xlabel('distribution of salary')
print(df.columns.tolist())
plt.show()
```

```
['0-10,000', '10-20,000', '20-30,000', '30-40,000', '40-50,000', '50-60,000', '60-70,
↪000', '70-80,000', '80-90,000', '90-100,000', '100-125,000', '125-150,000', '150-
↪200,000', '200-250,000', '250-300,000', '300-400,000', '400-500,000', '500,000+']
```

```
<Figure size 720x720 with 1 Axes>
```

The colors are awful but it displays the salary distribution, you can see that students clearly are in the lower parts similar to research assistants, notice that jobs with low statistical count can create a distortion as e.g. data journalist only has about 20 records. Jobs such as Manager and Principal Investigator seem to have a very even distribution indicating a faster climbing up the salary ladder.

In a similar fashion for other questions you could construct the same graph.

Another way to look at these things would be to use the difference between true and expected values, we already created the degree for differences, let's turn this into a bar plot.

```
df = degree_diff.loc[:,'100-125,000':'400-500,000'].sum(axis='columns').sort_values()
df
```

```
Q4
Bachelor's degree                                                  -188.182578
Some college/university study without earning a bachelor's degree  -20.266835
Professional degree                                                 -16.634066
I prefer not to answer                                               -7.157366
No formal education past high school                                 -6.855726
Master's degree                                                      57.349472
Doctoral degree                                                     181.747100
dtype: float64
```

```
sns.barplot(x = df.index.astype('str'), y=df, color='b')
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 720x720 with 1 Axes>
```

There are a lot of things you can still do to beautify this graph, but that's not our main interest, it shows the under- and overrepresented groups in high paying jobs. It would be wise however to create a relative version of this, as e.g. bachelor's degrees might be much more prevalent than others.

The same can be done with groupings of profession/job

```
df = prof_diff.loc[:,'100-125,000':'400-500,000'].sum(axis='columns').sort_values()
sns.barplot(x = df.index.astype('str'), y=df, color='b')
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 720x720 with 1 Axes>
```

To keep things consistent and because people love bar charts, we can use them to also display the disparity of choices of programming languages between high paying data scientists and all persons

```
df = (high_paying_job_corr-all_jobs_corr).Python.sort_values()
sns.barplot(x = df.index.astype('str'), y=df, color='b')
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 720x720 with 1 Axes>
```

you can see that the correlation between python and C/C++ is 15% less likely for high paid data scientists, indicating that it is not a good choice to learn next, in contrast languages such as Scala and SAS are a good option!

As far as my knowledge goes, the increase in correlation with None is because they are both negative and Python is more often chosen for data scientists, therefore the option 'not Python, not None' (but another language) is less often chosen, resulting in a higher correlation.

If you would want to make things a bit more fancy, you could use a clustermap, underlaying an algorithm will cluster your parameters into groups, here we cluster the correlation between common languages.

```
df = info_df[['Q6','Q9']].join(answer_dfs['Q16'])
sns.clustermap(df.corr('spearman'))
plt.show()
```

```
<Figure size 720x720 with 4 Axes>
```

The algorithm was able to group languages such as Python, Bash, SQL and Scala, indicating that there is some correlation, but this graph makes things rather complicated in my opinion.

Now about time spending, we could visualize this by showing the difference between high paid scientists and regular persons

```
df = time_high_pay-time_all
sns.barplot(x = df.index.astype('str'), y=df, color='b')
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 720x720 with 1 Axes>
```

we can see they spend more time on cleaning data, communication and production readiness, but less on visualization. Efficient time handling can be crucial for a good career!

At last we discussed age vs experience, as we cannot use histograms and overlapping is not possible with different categories (age vs exp) we are stuck with a bar chart. The repetivity of our dataset is reflected in our visualization.

```
df = age_diff
sns.barplot(x = df.index, y=df, color='b')
plt.show()
```

```
<Figure size 720x720 with 1 Axes>
```

Although simple it clearly shows the surplus of older persons in the high paying jobs.

```
df = exp_diff
sns.barplot(x = df.index, y=df, color='b')
plt.show()
```

```
<Figure size 720x720 with 1 Axes>
```

And as known before, experience gets your salary going from the 5 years and onwards

38.7 Summary

- Degrees and Job title strongly influences job salary
- Job sectors as Academics are underpayed
- For a data Scientist using python aim for other skills such as Scala and forget C/C++
- Your choice of Machine Learning library is of no importance
- Data Scientists spend less time visualizing and more cleaning, communicating and production
- Data Scientists are more worried about biases in their analysis
- Although both relevant, Age is more an indicator of a higher salary than experience, never to late to chase your dreams!

CHAPTER

THIRTYNINE

CASE STUDY: JOKES

In this case study we find out if we can make ourselves funnier by analysing jokes from a database.

The case study is divided into several parts:

- Goals
- Parsing
- Preparation (cleaning)
- Processing
- Exploration
- Visualization
- Conclusion

39.1 Goals

In this section we define questions that will be our guideline througout the case study

- What jokes are funny?
- Can we find types of jokes?
- Would a joke recommender work?

We'll (try to) keep these question in mind when performing the case study.

39.2 Parsing

we start out by importing all necessary libraries

```
import os
import json
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats
import matplotlib.pyplot as plt
from IPython.display import set_matplotlib_formats
%matplotlib inline
set_matplotlib_formats('svg')
```

```
/tmp/ipykernel_8969/4057771804.py:10: DeprecationWarning: `set_matplotlib_formats` is␣
↪deprecated since IPython 7.23, directly use `matplotlib_inline.backend_inline.set_
↪matplotlib_formats()`
  set_matplotlib_formats('svg')
```

in order to download datasets from kaggle, we need an API key to access their API, we'll make that here

```
if not os.path.exists(os.path.expanduser('~/.kaggle')):
    os.mkdir(os.path.expanduser('~/.kaggle'))

with open(os.path.expanduser('~/.kaggle/kaggle.json'), 'w') as f:
    json.dump(
        {
            "username":"lorenzf",
            "key":"7a44a9e99b27e796177d793a3d85b8cf"
        }
        , f)
```

now we can import kaggle too and download the datasets

```
import kaggle
kaggle.api.dataset_download_files(dataset='pavellexyr/one-million-reddit-jokes', path=
↪'./data', unzip=True)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_8969/2979186203.py in <module>
----> 1 import kaggle
      2 kaggle.api.dataset_download_files(dataset='pavellexyr/one-million-reddit-jokes
↪', path='./data', unzip=True)

ModuleNotFoundError: No module named 'kaggle'
```

the csv files are now in the './data' folder, we can now read them using pandas, here is the list of all csv files in our folder

```
os.listdir('./data')
```

```
['WA_Fn-UseC_-Telco-Customer-Churn.csv',
 'API_NY.GDP.PCAP.CD_DS2_en_csv_v2_3358201.csv',
 'noc_regions.csv',
 'freeFormResponses.csv',
 'SurveySchema.csv',
 'jester_ratings.csv',
 'multipleChoiceResponses.csv',
 'one-million-reddit-jokes.csv',
 'jester_items.csv',
 'athlete_events.csv',
 'API_SP.POP.TOTL_DS2_en_csv_v2_3358390.csv']
```

With only one file in the dataset, we import it.

```
reddit_jokes_df = pd.read_csv('./data/one-million-reddit-jokes.csv')
print('shape: ' + str(reddit_jokes_df.shape))
reddit_jokes_df.head()
```

```
shape: (1000000, 12)
```

```
   type      id subreddit.id subreddit.name  subreddit.nsfw  created_utc  \
0  post  ftbp1i        2qh72          jokes           False   1585785543
1  post  ftboup        2qh72          jokes           False   1585785522
2  post  ftbopj        2qh72          jokes           False   1585785508
3  post  ftbnxh        2qh72          jokes           False   1585785428
4  post  ftbjpg        2qh72          jokes           False   1585785009

                                          permalink      domain  url  \
0  https://old.reddit.com/r/Jokes/comments/ftbp1i...  self.jokes  NaN
1  https://old.reddit.com/r/Jokes/comments/ftboup...  self.jokes  NaN
2  https://old.reddit.com/r/Jokes/comments/ftbopj...  self.jokes  NaN
3  https://old.reddit.com/r/Jokes/comments/ftbnxh...  self.jokes  NaN
4  https://old.reddit.com/r/Jokes/comments/ftbjpg...  self.jokes  NaN

                                           selftext  \
0  My corona is covered with foreskin so it is no...
1                       It's called Google Sheets.
2  The vacuum doesn't snore after sex.\n\n&amp;#x...
3                                         [removed]
4                                         [removed]

                                              title  score
0             I am soooo glad I'm not circumcised!      2
1  Did you know Google now has a platform for rec...      9
2  What is the difference between my wife and my ...     15
3                             My last joke for now.      9
4            The Nintendo 64 turns 18 this week...    134
```

Already we can see a lot of unnecessary information, so cleanup is important. It seems the joke is divided in a title and selftext where often the punchline is present.

39.3 Preparation

here we perform tasks to prepare the data in a more pleasing format.

39.3.1 Cleanup

First thing I would like to do see which columns are useless, by printing the amount of unique values

```
for col in reddit_jokes_df.columns:
    print(col)
    print(reddit_jokes_df[col].nunique())
    print()
```

```
type
1

id
1000000

subreddit.id
```

(continues on next page)

(continued from previous page)

```
1

subreddit.name
1

subreddit.nsfw
1

created_utc
996373

permalink
1000000

domain
364

url
4410

selftext
520567

title
861254

score
8913
```

a few columns only have 1 value, also the links are not important for our case, so we drop them too.

```
reddit_jokes_df = reddit_jokes_df.drop(columns=['type', 'id', 'subreddit.id',
↪'subreddit.name', 'subreddit.nsfw', 'permalink', 'url'])
reddit_jokes_df.head()
```

```
   created_utc      domain                                          selftext  \
0   1585785543  self.jokes  My corona is covered with foreskin so it is no...
1   1585785522  self.jokes                        It's called Google Sheets.
2   1585785508  self.jokes  The vacuum doesn't snore after sex.\n\n&amp;#x...
3   1585785428  self.jokes                                         [removed]
4   1585785009  self.jokes                                         [removed]

                                               title  score
0             I am soooo glad I'm not circumcised!      2
1  Did you know Google now has a platform for rec...      9
2  What is the difference between my wife and my ...     15
3                              My last joke for now.      9
4            The Nintendo 64 turns 18 this week...    134
```

much cleaner already!

39.3.2 Data Types

Before we do anything with our data, it is good to see if our data types are in order

```
reddit_jokes_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 5 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   created_utc  1000000 non-null  int64
 1   domain       1000000 non-null  object
 2   selftext     995525 non-null   object
 3   title        1000000 non-null  object
 4   score        1000000 non-null  int64
dtypes: int64(2), object(3)
memory usage: 38.1+ MB
```

the created_utc feature is encoded in an unix timestamp, it would be more usefull to transform it to a timestamp

```
reddit_jokes_df['created'] = pd.to_datetime(reddit_jokes_df['created_utc'], unit='s')
del reddit_jokes_df['created_utc']
reddit_jokes_df.head()
```

```
       domain                                           selftext  \
0  self.jokes  My corona is covered with foreskin so it is no...
1  self.jokes                             It's called Google Sheets.
2  self.jokes  The vacuum doesn't snore after sex.\n\n&amp;#x...
3  self.jokes                                          [removed]
4  self.jokes                                          [removed]

                                               title  score  \
0                I am soooo glad I'm not circumcised!      2
1  Did you know Google now has a platform for rec...      9
2  What is the difference between my wife and my ...     15
3                               My last joke for now.      9
4               The Nintendo 64 turns 18 this week...    134

              created
0 2020-04-01 23:59:03
1 2020-04-01 23:58:42
2 2020-04-01 23:58:28
3 2020-04-01 23:57:08
4 2020-04-01 23:50:09
```

39.3.3 Missing values

for each dataframe we apply a few checks in order to see the quality of data

```
print(100*reddit_jokes_df.isna().sum()/reddit_jokes_df.shape[0])
```

```
domain      0.0000
selftext    0.4475
title       0.0000
score       0.0000
created     0.0000
dtype: float64
```

it looks like some jokes are missing the selftext field, we show a few here.

```
reddit_jokes_df[reddit_jokes_df.selftext.isna()].sort_values(by='score',␣
↪ascending=False)
```

```
            domain selftext  \
625315   imgur.com      NaN
971313  self.jokes      NaN
942471  self.jokes      NaN
926550  self.jokes      NaN
919422  self.jokes      NaN
...            ...      ...
929807  self.jokes      NaN
959394  self.jokes      NaN
929809  self.jokes      NaN
959338  self.jokes      NaN
999984  self.jokes      NaN

                                                     title  score  \
625315                The funniest /r/jokes has ever been  67950
971313                                 Ellen Pao's career  36918
942471  If a woman sleeps with a bunch of guys, she's ...  17486
926550     One in every 2 and a half men is HIV positive.  17456
919422  Accordion to a recent survey, replacing words ...  12580
...                                                   ...    ...
929807                                               9gag      0
959394                     Like flaming globes of Sigmund      0
929809  On a scale of 10 to 10, how good am I at givin...      0
959338  Who is Julius Caesar's favorite singer? Mark A...      0
999984      One direction should be renamed 0.8 Direction      0

                    created
625315 2017-05-20 15:41:28
971313 2015-07-03 15:41:05
942471 2015-10-05 16:09:09
926550 2015-11-18 04:29:54
919422 2015-12-07 18:55:27
...                     ...
929807 2015-11-09 03:33:22
959394 2015-08-14 13:40:21
929809 2015-11-09 03:26:55
959338 2015-08-14 17:03:55
999984 2015-03-26 19:57:54
```

(continues on next page)

(continued from previous page)

```
[4475 rows x 5 columns]
```

as far as I can see here the jokes are so short they are only one sentence, so we can fill in the missing values with an empty text.

```
reddit_jokes_df.selftext = reddit_jokes_df.selftext.fillna('')
```

This does not mean we are done, earlier I noticed the words [removed] and [deleted] in the selftext feature, indicating the joke was removed or deleted, these are missing values!

```
reddit_jokes_df[reddit_jokes_df.selftext.isin(['[removed]', '[deleted]'])].head()
```

```
       domain   selftext                                          title  \
3  self.jokes  [removed]                           My last joke for now.
4  self.jokes  [removed]              The Nintendo 64 turns 18 this week...
5  self.jokes  [removed]                               Sex with teacher.
6  self.jokes  [removed]                               Another long one.
8  self.jokes  [removed]  A Priest takes a walk down to the docks one day

   score             created
3      9 2020-04-01 23:57:08
4    134 2020-04-01 23:50:09
5      1 2020-04-01 23:49:55
6      8 2020-04-01 23:44:11
8     88 2020-04-01 23:39:27
```

I am going to remove these jokes as they are not complete anymore, it might have been that these jokes have been removed as they have already been posted.

```
reddit_jokes_df = reddit_jokes_df[~reddit_jokes_df.selftext.isin(['[removed]',
 ↪'[deleted]'])]
reddit_jokes_df.shape
```

```
(578637, 5)
```

seems we have kept about 578k jokes, not bad!

39.3.4 Duplicates

As formatting of text might be different i'm not expecting a lot of duplicates, let's see what we can find.

```
reddit_jokes_df[reddit_jokes_df.duplicated(subset=['title', 'selftext'])]
```

```
            domain                                          selftext  \
211     self.jokes                                   An academia nut..
4452    self.jokes                                      Reposssssssst
6349    self.jokes  “To Japan,” replies her husband. \n\n“Oh my! T...
6881    self.jokes  Fortunately, I belong to the 1% of intelligent...
8299    self.jokes                            You tell it a shitty joke.
...            ...                                               ...
999779  self.jokes                                              Dam.
999851  self.jokes                                    He tractor down.
999882  self.jokes
```

(continues on next page)

(continued from previous page)

```
999936  self.jokes   Don't be stupid, feminists can't change anything
999979  self.jokes                                                 Smoke

                                                    title  score  \
211          What do you call a nut that gets good grades?      5
4452    If a snake who is on reddit has to comment a r...      0
6349    A woman asks her husband where he’s taking the...      4
6881                          99.9% of people are idiots.  45135
8299                        How do you get a toilet to laugh?      0
...                                                   ...    ...
999779        What did the fish say when he hit the wall?     25
999851                     How did the farmer find his wife?     58
999882                                     women's rights      0
999936  How many feminists does it take to change a li...     24
999979                           What do you call a flying Jew?      0

                   created
211    2020-04-01 18:54:06
4452   2020-03-27 09:16:20
6349   2020-03-25 00:48:09
6881   2020-03-24 09:40:14
8299   2020-03-22 07:49:45
...                    ...
999779 2015-03-27 10:33:12
999851 2015-03-27 02:42:29
999882 2015-03-27 00:48:36
999936 2015-03-26 22:00:06
999979 2015-03-26 20:16:34

[12867 rows x 5 columns]
```

A fair amount of jokes are reposted, so we keep the ones with the highest score.

```
reddit_jokes_df = reddit_jokes_df.sort_values('score').drop_duplicates(subset=['title
↪', 'selftext'], keep='last').reset_index()
```

39.3.5 Text formatting

Before we can analyze the text in the jokes we have to format it. We can do this by removing all special character and changing it all to lowercase

```
for col in ['selftext', 'title']:
    reddit_jokes_df[col] = reddit_jokes_df[col].replace(to_replace="[^a-zA-Z,.!? ]",␣
↪value="", regex=True).str.lower()

reddit_jokes_df.head()
```

```
    index      domain                                          selftext  \
0  630580  self.jokes                              those who need closure,
1  187066  self.jokes                so when someone asks you can say its .
2  437464  self.jokes                                         tooth hurty!
3  714598  self.jokes  where did you get a phone that works in spaini...
4  187072  self.jokes  me how many am i allowed?guy only one me well ...
```

(continues on next page)

(continued from previous page)

```
                                              title  score  \
0        there are two kinds of people in the world.      0
1                          set your wifi password to      0
2                at what time do you see your dentist?      0
3  john and juan are on lunch break when juans ph...      0
4  a guy is handing out free fake mustaches on th...      0

              created
0 2017-05-12 17:01:44
1 2019-05-28 00:30:46
2 2018-03-28 10:17:26
3 2017-01-13 02:37:59
4 2019-05-28 00:20:01
```

Next we create a single joke by combining the title and selftext, this makes it easier to operate.

```
reddit_jokes_df['joke'] = reddit_jokes_df.title + ' ' + reddit_jokes_df.selftext
reddit_jokes_df = reddit_jokes_df.drop(columns=['title', 'selftext'])
reddit_jokes_df.head()
```

```
    index      domain  score             created  \
0  630580  self.jokes      0 2017-05-12 17:01:44
1  187066  self.jokes      0 2019-05-28 00:30:46
2  437464  self.jokes      0 2018-03-28 10:17:26
3  714598  self.jokes      0 2017-01-13 02:37:59
4  187072  self.jokes      0 2019-05-28 00:20:01

                                                joke
0  there are two kinds of people in the world. th...
1  set your wifi password to  so when someone ask...
2  at what time do you see your dentist? tooth hu...
3  john and juan are on lunch break when juans ph...
4  a guy is handing out free fake mustaches on th...
```

39.4 Processing

39.4.1 Timing of joke

I would like to know if the timing of the jokes makes an impact on how funny the joke is, so i grouped based on both the weekday as well as the hour of day.

```
reddit_jokes_weekday = reddit_jokes_df.groupby(reddit_jokes_df.created.dt.weekday).
↪score.agg(['mean', 'count'])
reddit_jokes_weekday
```

```
               mean  count
created
0        226.871773  79866
1        228.808886  82940
2        222.802165  84793
3        215.771594  84932
4        222.888666  82634
```

(continues on next page)

(continued from previous page)

```
5        232.752534  75089
6        241.322581  75516
```

```
reddit_jokes_hour = reddit_jokes_df.groupby(reddit_jokes_df.created.dt.hour).score.
↪agg(['mean', 'count'])
reddit_jokes_hour
```

```
                mean  count
created
0        189.177767  25646
1        189.383726  25440
2        172.406772  25368
3        140.741126  23637
4        144.066960  21162
5        137.355467  19006
6        168.542319  16671
7        214.903014  15198
8        271.710558  14217
9        398.431366  14009
10       456.262600  15952
11       446.946555  18056
12       404.759640  21447
13       318.348451  24342
14       263.100078  26899
15       227.382529  28322
16       204.701879  29327
17       185.274719  29623
18       210.557595  29369
19       194.320871  29342
20       194.809965  29063
21       179.245679  29099
22       198.723083  27817
23       179.602399  26758
```

39.4.2 Bag of words

To be able to work with the words in our joke, we create a bag of words dataframe, where for each word and joke combination a count is kept of how many times the word is present in that joke. Notice that stopwords are removed.

First we split each joke up in words

```
joke_words = reddit_jokes_df.joke.str.split(' ')
joke_words.head()
```

```
0    [there, are, two, kinds, of, people, in, the, ...
1    [set, your, wifi, password, to, , so, when, so...
2    [at, what, time, do, you, see, your, dentist?,...
3    [john, and, juan, are, on, lunch, break, when,...
4    [a, guy, is, handing, out, free, fake, mustach...
Name: joke, dtype: object
```

Next we use the nltk toolkit to get a list of english stopwords.

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stopwords.words('english')[:5]
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/lorenzf/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
['i', 'me', 'my', 'myself', 'we']
```

We remove all the stopwords from the jokes, now the jokes have a handicapped grammar.

```
joke_words = joke_words.head().apply(lambda x : [word for word in x if word not in␣
↪stopwords.words('english')])
joke_words.head()
```

```
0          [two, kinds, people, world., need, closure,]
1        [set, wifi, password, , someone, asks, say, .]
2                  [time, see, dentist?, tooth, hurty!]
3     [john, juan, lunch, break, juans, phone, rings...
4     [guy, handing, free, fake, mustaches, street, ...
Name: joke, dtype: object
```

Finally we are going to use sklearn and the CountVectorizer to create the BoW vector, this is a sparse matrix as most words are not appearing in most jokes. This means we cannot visualise the matrix, or our computer would explode.

```
from sklearn.feature_extraction.text import CountVectorizer

cnt_vect = CountVectorizer(analyzer="word", stop_words=stopwords.words('english'),␣
↪max_features=20000)

bow_jokes = cnt_vect.fit_transform(reddit_jokes_df.joke.values)
```

```
bow_jokes
```

```
<565770x20000 sparse matrix of type '<class 'numpy.int64'>'
        with 9101120 stored elements in Compressed Sparse Row format>
```

But we can fetch the vocabulary of our bag, which starts with a lot of weird words, indicating we might have chosen too many features

```
cnt_vect.get_feature_names_out()[:10]
```

```
array(['aa', 'aaa', 'aaah', 'aah', 'aardvark', 'aaron', 'ab', 'aback',
       'abacus', 'abandon'], dtype=object)
```

39.4.3 Term Frequency - Inverse Document Frequency

Another interesting method is the tf-idf matrix, where each occurence is weighted by the overall frequency of that word. If a word is used often over all jokes, it won't be as important, but if a word is used infrequent it is more important.

Again we use sklearn to vectorize our jokes

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect = TfidfVectorizer()
tfidf_jokes = tfidf_vect.fit_transform(reddit_jokes_df.joke.values)
tfidf_jokes
```

```
<565770x196601 sparse matrix of type '<class 'numpy.float64'>'
	with 15153814 stored elements in Compressed Sparse Row format>
```

we can create a quick dataframe to interpret the result, for each word in our dataset we retrieve the inverse document frequency, a high idf means a unique word.

```
idf = pd.DataFrame(
    {
      'term': tfidf_vect.get_feature_names_out(),
      'idf': tfidf_vect.idf_,
    }
)
idf.head()
```

```
     term        idf
0      aa  10.026437
1     aaa  10.275653
2    aaaa  12.454185
3   aaaaa  13.147332
4  aaaaaa  13.552798
```

When we sort them by idf we can find the most unique words, yet it doesn't seem to be useful at the moment.

```
idf.sort_values(by='idf', ascending=False).head(10)
```

```
                                                   term        idf
196600  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzthe  13.552798
110080                              misterunderstanding  13.552798
110074                                     misterectomy  13.552798
110075                                       misterious  13.552798
110076                                     misterjmyers  13.552798
110077                                        misterlee  13.552798
110078                                      misterogyny  13.552798
110079                                          misters  13.552798
110081                                misterunderstood  13.552798
110072                                misterapproximate  13.552798
```

39.5 Exploration

```
good_jokes = reddit_jokes_df[reddit_jokes_df.score>10000].copy()
good_jokes
```

```
         index      domain   score             created  \
562291  766392  self.jokes   10003 2016-11-05 12:19:21
562292  817060  self.jokes   10013 2016-08-15 23:23:22
562293  956485  self.jokes   10017 2015-08-23 14:31:29
562294  672977  self.jokes   10018 2017-03-12 10:30:11
562295  207962  self.jokes   10019 2019-04-24 23:06:14
...        ...         ...     ...                 ...
565765  329338  self.jokes   98257 2018-10-08 13:53:47
565766  141894  self.jokes  103652 2019-08-10 15:03:25
565767  596220  self.jokes  106412 2017-07-05 18:01:05
565768  511072  self.jokes  136359 2017-11-21 20:15:27
565769   29360  self.jokes  142733 2020-02-20 01:51:00

                                                     joke
562291  a joke my grandma told me before she passed. s...
562292  if a woman sleeps with  men shes a slut, but i...
562293  how many germans does it take to change a ligh...
562294  a man and his wife are awakened at  oclock in ...
562295  my least favorite color is purple. i hate it m...
...                                                   ...
565765  a new navy recruit has his first day on the su...
565766  if your surprised that jeffrey epstein commite...
565767  v vedit seems like the ctrl key on my keyboard...
565768  calm down about the net neutrality thing... pa...
565769  sad news the founder of rjokes has passed away...

[3479 rows x 5 columns]
```

```
good_jokes_word_cnt = pd.concat(
    [
        pd.Series(cnt_vect.get_feature_names_out()),
        pd.Series(np.asarray(bow_jokes[good_jokes.index].sum(axis=0)).squeeze()),
    ], axis='columns', keys=['word', 'count']
)
good_jokes_word_cnt
```

```
             word  count
0              aa      1
1             aaa      3
2            aaah      0
3             aah      0
4        aardvark      0
...           ...    ...
19995        zoos      0
19996          zs      0
19997    zucchini      0
19998  zuckerberg      1
19999        zwei      1

[20000 rows x 2 columns]
```

```python
good_jokes_word_cnt.sort_values('count', ascending=False).head(20)
```

```
        word  count
15199   says   1647
10528    man   1405
15050   said   1196
12100    one   1057
8673      im    833
7278     get    717
19559   wife    669
10101   like    656
5190    dont    632
19416   well    626
1211    back    626
953     asks    607
947    asked    586
19762  would    564
9692    know    562
7779     guy    518
7388      go    492
4463     day    481
19689  woman    479
7473     got    461
```

```python
for joke in good_jokes[good_jokes.joke.str.contains(' man ')].tail(5).joke:
    print(joke)
    print()
```

```
a man in an interrogation room says im not saying a word without my lawyer present.␣
↪cop you are the lawyer. lawyer exactly, so wheres my present?

christmas joke nsfw a  year old male walks into a drug store. he says ive been␣
↪invited to christmas dinner at my new girlfriends house. afterwards i hope there is␣
↪a chance i get lucky, you know what i mean clerk how about condoms then? they could␣
↪come in handy. heres a pack. the young man after paying walks to the door, stops,␣
↪smiles, comes back you know what, the mom is also smoking hot, i think ill take␣
↪another pack, just in case i get extra lucky.christmas eve comes around, the boy␣
↪sits at the dinner table and doesnt say a word. after a while his girlfriend says␣
↪if i had known you were so quiet, i wouldnt have invited you. the young man replies␣
↪if you had told me your dad works at a drug store, i wouldnt have come.

my favorite joke everyone knows dave dave was bragging to his boss one day, you know,␣
↪i know everyone there is to know. just name someone, anyone, and i know them.tired␣
↪of his boasting, his boss called his bluff, ok, dave, how about tom cruise?no␣
↪dramas boss, tom and i are old friends, and i can prove it.so dave and his boss fly␣
↪out to hollywood and knock on tom cruises door, and tom cruise shouts, dave! whats␣
↪happening? great to see you! come on in for a beer!although impressed, daves boss␣
↪is still skeptical. after they leave cruises house, he tells dave that he thinks␣
↪him knowing cruise was just lucky.no, no, just name anyone else, dave says.
↪president obama, his boss quickly retorts.yup, dave says, old buddies, lets fly out␣
↪to washington, and off they go. at the white house, obama spots dave on the tour␣
↪and motions him and his boss over, saying, dave, what a surprise, i was just on my␣
↪way to a meeting, but you and your friend come on in and lets have a beer first and␣
↪catch up.well, the boss is very shaken by now but still not totally convinced.after␣
↪they leave the white house grounds he expresses his doubts to dave, who again␣
↪implores him to name anyone else.pope francis, his boss replies.sure! says dave.␣
↪ive known the pope for years. so off they fly to rome.dave and his boss are␣
↪assembled with the masses at the vaticans st. peters square when dave says, this␣
↪will never work. i cant catch the popes eye among all these people. tell you what,␣
↪i know all the guards so let me just go upstairs and ill come out on the balcony␣
↪with the pope.he disappears into the crowd headed towards the vatican.sure enough,␣
↪half an hour later dave emerges with the pope on the balcony, but by the time dave␣
↪returns, he finds that his boss has had a heart attack and is surrounded by␣
↪paramedics.making his way to his boss side, dave asks him, what happened?his boss␣
```

(continues on next page)

(continued from previous page)

```
by legalizing cannabis and samesex marriage we finally interpreted the bible␣
↪correctly a man who lays with another man should be stoned. leviticus  esv edit a␣
↪typo edit thanks for the gold humorous stranger!

a man walks into a bar... the bartender asks why the long face? the man replies i␣
↪just found out my wife is sleeping with another man. ive decided im going to drink␣
↪myself to death. the bartender looks shocked and says im sorry i cant help you kill␣
↪yourself. the man asks well what would you do in my situation?the bartender puffs␣
↪himself up a bit and says if i found out a guy was sleeping with my wife i wouldnt␣
↪sit around feeling sorry for myself, id kill the guy.the man jumps up from his␣
↪stool and shouts thats a great idea! thanks! and runs out of the bar.a couple hours␣
↪goes by and the bartender is starting to get nervous when the man walks back into␣
↪the bar with a smile on his face.did you kill the guy? the bartender asks nervously.
↪nope! i slept with your wife. whiskey please.
```

```
good_jokes_word_idf = pd.concat(
    [
        pd.Series(tfidf_vect.get_feature_names_out()),
        pd.Series(np.asarray(tfidf_jokes[good_jokes.index].sum(axis=0)).squeeze()),
    ], axis='columns', keys=['word', 'count']
)
good_jokes_word_idf
```

```
                                                     word     count
0                                                      aa  0.218800
1                                                     aaa  0.337332
2                                                    aaaa  0.000000
3                                                   aaaaa  0.000000
4                                                  aaaaaa  0.000000
...                                                   ...       ...
196596                                   zzzzzzzzzzzzzzzz  0.000000
196597                              zzzzzzzzzzzzzzzzzzzzzz  0.000000
196598                    zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzthe  0.000000
196599         zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzthe  0.000000
196600   zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzthe  0.000000

[196601 rows x 2 columns]
```

```
good_jokes_word_idf[~good_jokes_word_idf.word.isin(stopwords.words('english'))].sort_
↪values('count', ascending=False).head(20)
```

```
          word      count
147669    said  67.537998
149360    says  64.319957
103769     man  57.291711
85108       im  51.772732
190313    wife  51.119043
121680     one  50.363516
99600     like  44.929305
69195      get  43.785308
53561     edit  41.610126
50077     dont  41.495084
11555    asked  39.825113
95238     know  34.931383
```

(continues on next page)

(continued from previous page)

```
192780   would  33.093493
71679      got  32.481152
188359    well  31.715616
74211      guy  31.570352
14528     back  30.231193
153190     sex  28.914655
191722   woman  28.747487
128249  people  28.045112
```

```
tfidf_good_vect = TfidfVectorizer()
tfidf_good_jokes = tfidf_good_vect.fit_transform(good_jokes.joke.values)
tfidf_good_jokes
```

```
<3479x13866 sparse matrix of type '<class 'numpy.float64'>'
	with 158145 stored elements in Compressed Sparse Row format>
```

```
good_jokes_word_idf = pd.concat(
    [
        pd.Series(tfidf_good_vect.get_feature_names_out()),
        pd.Series(np.asarray(tfidf_good_jokes.sum(axis=0)).squeeze()),
    ], axis='columns', keys=['word', 'count']
)
good_jokes_word_idf
```

```
             word     count
0              aa  0.217575
1             aaa  0.387436
2           aaaah  0.085922
3       aaarrrghh  0.316697
4          aaaway  0.318949
...           ...       ...
13861    zoobooks  0.237976
13862   zookeeper  0.409659
13863    zoophile  0.203000
13864  zuckerberg  0.352931
13865        zwei  0.142169

[13866 rows x 2 columns]
```

```
good_jokes_word_idf[~good_jokes_word_idf.word.isin(stopwords.words('english'))].sort_
↪values('count', ascending=False).head(20)
```

```
        word      count
10299   said  61.751454
10407   says  60.323491
7268     man  58.135122
8314     one  49.662388
6025      im  49.230105
13533   wife  47.633648
6978    like  45.431302
5084     get  43.641674
3600    dont  40.811579
717    asked  36.849412
6697    know  34.773909
```

(continues on next page)

(continued from previous page)

```
5422      guy  32.166281
5227      got  31.952154
13694   would  31.815714
13404    well  29.399329
8758   people  29.100171
3846     edit  29.063515
10666     sex  28.859416
918      back  28.349291
3092      day  28.040213
```

```
from sklearn.cluster import KMeans
```

```
kmeans = KMeans(n_clusters=100)
kmeans.fit(tfidf_good_jokes)
```

```
KMeans(n_clusters=100)
```

```
good_jokes['label'] = kmeans.labels_
```

```
good_jokes.head()
```

```
         index      domain  score             created  \
562291  766392  self.jokes  10003 2016-11-05 12:19:21
562292  817060  self.jokes  10013 2016-08-15 23:23:22
562293  956485  self.jokes  10017 2015-08-23 14:31:29
562294  672977  self.jokes  10018 2017-03-12 10:30:11
562295  207962  self.jokes  10019 2019-04-24 23:06:14

                                                    joke  label
562291  a joke my grandma told me before she passed. s...     10
562292  if a woman sleeps with  men shes a slut, but i...     35
562293  how many germans does it take to change a ligh...     30
562294  a man and his wife are awakened at  oclock in ...     19
562295  my least favorite color is purple. i hate it m...     73
```

```
jokes_cluster_counts = good_jokes.label.value_counts()
jokes_cluster_counts
```

```
22    356
19    312
5     211
10    166
65     95
     ...
33      8
71      8
55      7
13      7
27      6
Name: label, Length: 100, dtype: int64
```

```
for joke in good_jokes[good_jokes.label==jokes_cluster_counts.index[0]].sort_values(
↪'score', ascending=False).joke.head():
```

(continues on next page)

(continued from previous page)

```
    print(joke)
    print()
```

```
calm down about the net neutrality thing... paying additional money to access certain␣
↪sites will give you a sense of pride and accomplishment.

why was the antivaxxers  year old child crying? midlife crisis

all countries eventually got coronavirus but china got it right off the bat.

as an aussie, americans are always asking me where in australia there isnt something␣
↪trying to kill you... school is my answer

a feminist told me about the dwayne johnson rule. the rule, as she explained it, was␣
↪that in order to determine if a particular comment was appropriate to say to a␣
↪woman, first ask yourself, would i be comfortable saying this to dwayne johnson? if␣
↪not, dont say it.i thought this sounded like a good rule. so i told heryour chest␣
↪is fucking epic.
```

```
for joke in good_jokes[good_jokes.label==jokes_cluster_counts.index[-1]].sort_values(
↪'score', ascending=False).joke.head():
    print(joke)
    print()
```

```
i take viagra for my sun burn... it doesnt cure it, but it keeps the sheets off my␣
↪legs when i sleep.ampxb

what rhymes with orange no it doesnt

im taking viagra for my sunburn. it doesnt cure it, but it keeps the sheets off of my␣
↪legs

ive been taking viagra for my sunburn doesnt cure it, but it keeps the sheets off my␣
↪legs at night.

im taking viagra for my sunburn it doesnt cure it, but it keeps the sheets off my legs
```

```
kaggle.api.dataset_download_files(dataset='vikashrajluhaniwal/jester-17m-jokes-
↪ratings-dataset', path='./data', unzip=True)
```

```
jester_jokes_df = pd.read_csv('./data/jester_items.csv')
print('shape: ' + str(jester_jokes_df.shape))
jester_jokes_df.head()
```

```
shape: (150, 2)
```

```
   jokeId                                           jokeText
0       1  A man visits the doctor. The doctor says "I ha...
1       2  This couple had an excellent relationship goin...
2       3  Q. What's 200 feet long and has 4 teeth? \n\nA...
3       4  Q. What's the difference between a man and a t...
4       5  Q.\tWhat's O. J. Simpson's Internet address? \...
```

```
jester_ratings_df = pd.read_csv('./data/jester_ratings.csv')
print('shape: ' + str(jester_ratings_df.shape))
jester_ratings_df.head()
```

```
shape: (1761439, 3)
```

```
   userId  jokeId  rating
0       1       5   0.219
1       1       7  -9.281
2       1       8  -9.281
3       1      13  -6.781
4       1      15   0.875
```

```
jester_ratings_df.groupby('jokeId').rating.mean()
```

```
jokeId
5     -1.756331
7     -1.809230
8     -0.672010
13    -0.590224
15    -1.377098
         ...
146    0.178280
147    1.783395
148    3.061760
149    2.399796
150    2.810758
Name: rating, Length: 140, dtype: float64
```

```
jester_sorted = jester_ratings_df.groupby('jokeId').rating.mean().to_frame().
↪join(jester_jokes_df).sort_values('rating', ascending=False)
jester_sorted.head()
```

```
          rating  jokeId                                          jokeText
jokeId
53      3.714381    54.0  The Pope dies and, naturally, goes to heaven. ...
105     3.711223   106.0  An engineer dies and reports to the pearly gat...
89      3.606506    90.0  Q: How many programmers does it take to change...
129     3.583496   130.0  An old man goes to the doctor for his yearly p...
35      3.560305    36.0  A guy walks into a bar, orders a beer and says...
```

```
for joke in jester_sorted.head().jokeText:
    print(joke)
    print('---')
```

```
The Pope dies and, naturally, goes to heaven. He's met by the reception
committee, and after a whirlwind tour he is told that he can enjoy any
of the myriad of recreations available.
He decides that he wants to read all of the ancient original text of the
Holy Scriptures, so he spends the next eon or so learning languages.
After becoming a linguistic master, he sits down in the library and
begins to pour over every version of the Bible, working back from most
recent "Easy Reading" to the original script.
All of a sudden there is a scream in the library. The Angels come
```

(continues on next page)

(continued from previous page)

```
running in only to find the Pope huddled in his chair, crying to himself
and muttering, "An 'R'! The scribes left out the 'R'."
A particularly concerned Angel takes him aside, offering comfort, asks
him what the problem is and what does he mean.
After collecting his
wits, the Pope sobs again, "It's the letter 'R'. They left out the 'R'.
The word was supposed to be CELEBRATE!"

---
An engineer dies and reports to the pearly gates. St. Peter checks his dossier and␣
↪says, "Ah, you''re an engineer--you're in the wrong place." So, the engineer␣
↪reports to the gates of hell and is let in. Pretty soon, the engineer gets␣
↪dissatisfied with the level of comfort in hell, and starts designing and building␣
↪improvements. After awhile, they've got air conditioning, flush toilets and␣
↪escalators, and the engineer is a pretty popular guy. One day, God calls Satan up␣
↪on the telephone and says with a sneer, "So, how's it going down there in hell?"␣
↪Satan replies, "Hey, things are going great. We've got air conditioning, flush␣
↪toilets and escalators, and there's no telling what this engineer is going to come␣
↪up with next." God replies, "What?? You've got an engineer? That's a mistake--he␣
↪should never have gotten down there; send him up here." Satan says, "No way." I␣
↪like having an engineer on the staff, and I'm keeping him." God says, "Send him␣
↪back up here or I'll sue." Satan laughs uproariously and answers, "Yeah, right. And␣
↪just where are YOU going to get a lawyer?"
---
Q: How many programmers does it take to change a lightbulb?

A: NONE!   That's a hardware problem....

---
An old man goes to the doctor for his yearly physical, his wife tagging along. When␣
↪the doctor enters the examination room, he tells the old man, "I need a urine␣
↪sample, a stool sample and a sperm sample." The old man, being hard of hearing,␣
↪looks at his wife and yells: "WHAT? What did he say? What's he want?" His wife␣
↪yells back, "He needs your underwear."
---
A guy walks into a bar, orders a beer and says to the bartender,
"Hey, I got this great Polish Joke..."

The barkeep glares at him and says in a warning tone of voice:
"Before you go telling that joke you better know that I'm Polish, both
bouncers are Polish and so are most of my customers"

"Okay" says the customer,"I'll tell it very slowly."

---
```

```
jester_ratings_pivot_df = jester_ratings_df.pivot(index='userId', columns='jokeId',␣
↪values='rating')
jester_ratings_pivot_df.head()
```

```
jokeId     5      7      8      13     15     16     17     18     19     20   \
userId
1       0.219 -9.281 -9.281 -6.781  0.875 -9.656 -9.031 -7.469 -8.719 -9.156
2      -9.688  9.938  9.531  9.938  0.406  3.719  9.656 -2.688 -9.562 -9.125
3      -9.844 -9.844 -7.219 -2.031 -9.938 -9.969 -9.875 -9.812 -9.781 -6.844
4      -5.812 -4.500 -4.906    NaN    NaN    NaN    NaN    NaN    NaN    NaN
```

(continues on next page)

(continued from previous page)

```
5       6.906  4.750 -5.906 -0.406 -4.031  3.875  6.219  5.656  6.094  5.406

jokeId  ...  141  142  143  144  145  146  147  148  149  150
userId  ...
1       ...  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
2       ...  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
3       ...  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
4       ...  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
5       ...  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN

[5 rows x 140 columns]
```

```
kmeans = KMeans(n_clusters=100)
kmeans.fit(jester_ratings_pivot_df.fillna(0))
```

```
KMeans(n_clusters=100)
```

```
user_clusters = pd.Series(kmeans.labels_, index=jester_ratings_pivot_df.index)
user_clusters
```

```
userId
1        18
2        42
3        64
4        37
5        15
         ..
63974    68
63975    56
63976    98
63977    85
63978    14
Length: 59132, dtype: int32
```

```
user_cluster_counts = user_clusters.value_counts()
user_cluster_counts
```

```
35    3806
98    2301
81    1884
37    1672
78    1655
      ...
26      47
4       45
31       1
62       1
97       1
Length: 100, dtype: int64
```

```
users_set = set(user_clusters[user_clusters==user_cluster_counts.index[0]].index)
print(users_set)
```

```
{8193, 16389, 7, 24583, 57354, 16395, 32779, 57356, 24592, 57363, 32791, 24601, 16414,
↪ 24606, 32801, 49185, 8232, 16425, 24618, 43, 24619, 24621, 49193, 41012, 24631, 58,
↪ 24634, 8252, 24638, 24642, 16453, 24647, 49223, 8265, 57421, 49230, 49233, 82,
↪24659, 24660, 49235, 24663, 24664, 57431, 24666, 32861, 41057, 24678, 57448, 32873,
↪24683, 116, 16501, 41080, 24700, 125, 16510, 32894, 32895, 8321, 24705, 32898, 8324,
↪ 8328, 24714, 16523, 49294, 49295, 24720, 41104, 49297, 49300, 32917, 41109, 57496,
↪8347, 32923, 41115, 32926, 49309, 8352, 49312, 24746, 32938, 24749, 24750, 32945,
↪8370, 24756, 49335, 57527, 41149, 57537, 41156, 24778, 24780, 41168, 8406, 49367,
↪41176, 57559, 49371, 41188, 41189, 57572, 41194, 33007, 24823, 24825, 41211, 49408,
↪33029, 262, 49416, 33034, 41227, 57613, 16656, 57619, 16660, 16661, 8470, 33045,
↪24856, 33049, 49428, 283, 49434, 57629, 24865, 8482, 41251, 16676, 49448, 24877,
↪41264, 8499, 16691, 311, 49467, 16701, 24895, 8515, 324, 49475, 41288, 8524, 41296,
↪49488, 33110, 49503, 24935, 41320, 24937, 57704, 8555, 364, 24939, 24941, 24943,
↪57714, 33142, 24951, 24954, 49530, 382, 57729, 24963, 16773, 16777, 41354, 57745,
↪57747, 8601, 24986, 33178, 33190, 16807, 57766, 427, 33201, 16818, 8628, 33206,
↪25017, 41401, 57790, 8640, 49601, 8645, 33221, 49606, 41417, 465, 8657, 16849,
↪49619, 41431, 16856, 33242, 49627, 479, 41443, 57827, 16870, 33258, 16878, 8689,
↪41458, 8691, 16884, 8693, 49651, 57847, 49659, 33278, 33281, 8708, 25093, 33284,
↪41478, 49671, 25097, 49672, 57861, 49677, 16910, 8719, 25103, 33294, 33295, 25107,
↪8724, 16916, 41491, 25111, 41494, 537, 540, 25117, 41500, 57889, 57892, 57893, 556,
↪25132, 8761, 41529, 25148, 33341, 25154, 582, 25158, 41544, 33354, 41549, 16976,
↪33361, 594, 8787, 8788, 33364, 49749, 8796, 41565, 25182, 16992, 16996, 615, 8809,
↪8813, 25198, 33390, 41584, 626, 632, 633, 49787, 25223, 41610, 41611, 49803, 17038,
↪656, 25236, 661, 49814, 33439, 17057, 675, 17062, 681, 688, 8881, 17075, 25275,
↪49852, 17089, 8900, 33477, 711, 712, 8905, 33481, 17099, 33482, 25294, 17103, 17104,
↪ 17105, 58065, 17107, 25305, 17114, 49882, 58073, 33503, 17121, 58084, 33509, 25320,
↪ 746, 8941, 41712, 25330, 8950, 58103, 761, 25340, 25341, 25342, 33536, 41729,
↪41731, 41732, 49923, 25350, 49926, 49928, 33548, 49934, 17168, 8980, 789, 17179,
↪25372, 8989, 17182, 25374, 25376, 41760, 41762, 41763, 49950, 805, 33574, 17192,
↪41782, 17210, 17212, 58174, 58175, 9028, 17220, 17222, 17223, 841, 9033, 25417,
↪41804, 25422, 50000, 33617, 17234, 33620, 17239, 50009, 17242, 41819, 33629, 9054,
↪50014, 41824, 17251, 33639, 17256, 9065, 9066, 58215, 41840, 25458, 888, 25464, 890,
↪ 891, 25466, 33656, 58238, 25472, 25473, 33664, 33667, 25478, 25479, 41863, 58249,
↪9098, 9099, 25483, 33674, 9103, 41875, 25493, 41877, 9112, 17305, 25502, 17311,
↪17312, 25503, 25504, 25505, 33696, 41893, 58273, 25512, 50090, 943, 17327, 17329,
↪25519, 25523, 17332, 25527, 17345, 17346, 17347, 41925, 33734, 33735, 58315, 9164,
↪17362, 17363, 58325, 41942, 17367, 41945, 41946, 25567, 25568, 9185, 50143, 17379,
↪25571, 50145, 41967, 33776, 33781, 17398, 17399, 9208, 9209, 58360, 25595, 33787,
↪17405, 25599, 58367, 9218, 41986, 41987, 17415, 33799, 17419, 1037, 9229, 25615,
↪17424, 25617, 42000, 25623, 1048, 25625, 33815, 42007, 1054, 42014, 33827, 17448,
↪42029, 25648, 33840, 58418, 17460, 33846, 17466, 9276, 25660, 9287, 25674, 1102,
↪17486, 17487, 50256, 50260, 1109, 33880, 17497, 25690, 17499, 50268, 9309, 1119,
↪33888, 50272, 58467, 33896, 9322, 50282, 42093, 42094, 33903, 25712, 50288, 58479,
↪50291, 58482, 9336, 50297, 42106, 42113, 50305, 9348, 17541, 25737, 9354, 58508,
↪9360, 25744, 42130, 25747, 17556, 25755, 25761, 17573, 33958, 42153, 25770, 25772,
↪17581, 25773, 17583, 25776, 33969, 25778, 17587, 33971, 25783, 17592, 42173, 17598,
↪58564, 17607, 9416, 50379, 34005, 34007, 42200, 58587, 58592, 25825, 42215, 58599,
↪42217, 58600, 42219, 58602, 1261, 9454, 42221, 34032, 50417, 34036, 42230, 42231,
↪34042, 9467, 50426, 17663, 17666, 58627, 42246, 34056, 9481, 34058, 17675, 42250,
↪50441, 58633, 42255, 34065, 50452, 34069, 25878, 50455, 17689, 25881, 50457, 50458,
↪34081, 9506, 9507, 25894, 9512, 58664, 25898, 1328, 25904, 9526, 50487, 34108,
↪25921, 9540, 50500, 58698, 25931, 42319, 25937, 9555, 1364, 25939, 17750, 42323,
↪50518, 9564, 25949, 17759, 25951, 50527, 34146, 34147, 50531, 58723, 17768, 50536,
↪17771, 42349, 17774, 17776, 1394, 17779, 50550, 25975, 58746, 17787, 42364, 17789,
↪42369, 17798, 50570, 1421, 1422, 1423, 9615, 25997, 42386, 1431, 17816, 34204,
↪50589, 34207, 26018, 1443, 17827, 34212, 58786, 58789, 26025, 9646, 42415, 26042,
↪34234, 9660, 34236, 42426, 26047, 26053, 42439, 17864, 17865, 1482, 26062, 17872,
↪34256, 42450, 26070, 42456, 58846, 17887, 17888, 26079, 34272, 9699, 26083, 42466,
```

(continues on next page)

```
↪42472, 58857, 34282, 58858, 26092, 26093, 58862, 26097, 17908, 26100, 26102, 58874,
↪26110, 34305, 26116, 9733, 50694, 58886, 17928, 26121, 34315, 17932, 17933, 58898,
↪26132, 17941, 26134, 34326, 50708, 50709, 9755, 58910, 26144, 17953, 58918, 50727,
↪17960, 58927, 34361, 34363, 50750, 17983, 50752, 9795, 58950, 50762, 17995, 1613,
↪1617, 26193, 26194, 58968, 26202, 18012, 18013, 58976, 26210, 18024, 42603, 9836,
↪34412, 26222, 42604, 58991, 34417, 42610, 1651, 18035, 26231, 1657, 42624, 42625,
↪18050, 26242, 26247, 1677, 26253, 42640, 26257, 59027, 42645, 26262, 34455, 9880,
```

(continued from previous page)

```
jester_group_sorted = jester_ratings_df[jester_ratings_df.userId.isin(users_set)].
↪groupby('jokeId').rating.mean().to_frame().join(jester_jokes_df).sort_values('rating
↪', ascending=False)
jester_group_sorted.head()
```

```
            rating  jokeId                                          jokeText
jokeId
53        1.457240    54.0  The Pope dies and, naturally, goes to heaven. ...
114       1.289011   115.0  A lady bought a new Lexus. It cost a bundle. T...
50        1.147600    51.0  Did you hear that Clinton has announced there ...
126       1.140983   127.0  A little boy goes to his dad and asks, "What i...
89        1.136812    90.0  Q: How many programmers does it take to change...
```

```
users_set = set(user_clusters[user_clusters==user_cluster_counts.index[-5]].index)
jester_group_sorted = jester_ratings_df[jester_ratings_df.userId.isin(users_set)].
↪groupby('jokeId').rating.mean().to_frame().join(jester_jokes_df).sort_values('rating
↪', ascending=False)
jester_group_sorted.head()
```

```
            rating  jokeId                                          jokeText
jokeId
80        9.469000    81.0  An Asian man goes into a New York CityBank to ...
73        8.937500    74.0  Q: How many stalkers does it take to change a ...
116       7.938000   117.0  A man joins a big corporate empire as a traine...
106       7.290574   107.0  (A) The Japanese eat very little fat and suffe...
63        7.135891    64.0  What is the rallying cry of the International ...
```

```
kmeans.inertia_
```

```
30322545.978758477
```

```
kmeans = KMeans(n_clusters=200)
kmeans.fit(jester_ratings_pivot_df.fillna(0))
```

```
KMeans(n_clusters=200)
```

```
kmeans.inertia_
```

```
28653640.838373728
```

```
user_clusters = pd.Series(kmeans.labels_, index=jester_ratings_pivot_df.index)
user_clusters.value_counts()
```

```
43     3134
110    2009
7      1161
1      1059
118    1038
       ...
116       1
```

(continues on next page)

(continued from previous page)

```
115        1
80         1
158        1
125        1
Length: 200, dtype: int64
```

```
users_set = set(user_clusters[user_clusters==49].index)
jester_group_sorted = jester_ratings_df[jester_ratings_df.userId.isin(users_set)].
↪groupby('jokeId').rating.mean().to_frame().join(jester_jokes_df).sort_values('rating
↪', ascending=False)
jester_group_sorted.head()
```

```
          rating  jokeId                                           jokeText
jokeId
27      4.927000    28.0  A mechanical, electrical and a software engine...
116     4.562000   117.0  A man joins a big corporate empire as a traine...
8       1.609726     9.0  A country guy goes into a city bar that has a ...
50      1.509039    51.0  Did you hear that Clinton has announced there ...
53      1.463231    54.0  The Pope dies and, naturally, goes to heaven. ...
```

```
elbow_dict = {}
for k in [5, 10, 50, 100, 200, 500]:
    print(k)
    elbow_dict[k] = {}
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(jester_ratings_pivot_df.fillna(0))

    elbow_dict[k]['kmeans'] = kmeans
    elbow_dict[k]['inertia'] = kmeans.inertia_
    elbow_dict[k]['user_cluster'] = pd.Series(kmeans.labels_, index=jester_ratings_
↪pivot_df.index)
```

```
5
10
50
100
200
500
```

```
for k, clustering in elbow_dict.items():
    print(clustering['inertia'])
```

```
39282887.116419956
36456133.44415126
31997079.132014535
30295914.351647645
28679506.336334243
26367649.15034368
```

```
inertia = pd.Series([clustering['inertia'] for k, clustering in elbow_dict.items()],␣
↪index=elbow_dict.keys())
sns.lineplot(x=inertia.index, y=inertia)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa6ec815250>
```

```
<Figure size 432x288 with 1 Axes>
```

```
elbow_dict[100]['user_cluster'].value_counts()
```

```
87    3840
41    2218
55    1817
20    1703
63    1670
      ...
36      74
56      60
45      59
88      38
77       1
Length: 100, dtype: int64
```

```
elbow_dict[500]['user_cluster'].value_counts()
```

```
336    2090
15     1384
179     850
362     765
297     747
       ...
462       1
356       1
435       1
209       1
487       1
Length: 500, dtype: int64
```

```
from sklearn.neighbors import NearestNeighbors
```

```
nbrs = NearestNeighbors(n_neighbors=5)
nbrs.fit(jester_ratings_pivot_df.fillna(0))
```

```
NearestNeighbors()
```

```
jester_ratings_pivot_df.fillna(0).loc[[1]]
```

```
jokeId    5      7      8      13     15     16     17     18     19     20   \
userId
1       0.219 -9.281 -9.281 -6.781  0.875 -9.656 -9.031 -7.469 -8.719 -9.156

jokeId  ...  141  142  143  144  145  146  147  148  149  150
userId  ...
1       ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[1 rows x 140 columns]
```

```
dist, neighbours = nbrs.kneighbors(jester_ratings_pivot_df.fillna(0).loc[[1]])
neighbours[0].tolist()
```

```
[0, 44456, 100, 4214, 51129]
```

```
neighbours_ratings = jester_ratings_pivot_df.iloc[neighbours[0].tolist()[1:]]
neighbours_ratings
```

```
jokeId     5      7      8      13     15     16     17     18     19     20   \
userId
47727     NaN -5.938 -5.938 -6.188 -8.594 -7.844 -8.031 -7.562 -7.750    NaN
114     8.438 -5.594 -3.344 -3.750  2.594 -8.312 -5.469 -4.469 -2.531 -3.969
4641      NaN -4.531 -6.188 -2.375 -2.750 -1.938 -5.250 -3.625  1.156    NaN
55103     NaN -6.875 -6.875 -6.875 -5.750 -5.719 -5.719 -5.719 -5.719    NaN

jokeId  ...  141  142    143    144    145  146  147    148    149    150
userId  ...
47727   ...  NaN  NaN  6.438    NaN  5.594  NaN  NaN    NaN    NaN    NaN
114     ...  NaN  NaN    NaN    NaN    NaN  NaN  NaN    NaN    NaN    NaN
4641    ...  NaN  NaN    NaN  6.031    NaN  NaN  NaN  7.125  6.719    NaN
55103   ...  NaN  NaN  2.375    NaN  5.125  NaN  NaN  1.625  4.156  2.375

[4 rows x 140 columns]
```

```
approriate_jokes = neighbours_ratings.mean()[neighbours_ratings.mean()>7].index.
↪tolist()
approriate_jokes
```

```
[5, 27, 29, 50, 69, 105, 121, 122, 123, 125]
```

```
recommended_jokes = jester_ratings_pivot_df.loc[1,approriate_jokes]
recommended_jokes
```

```
jokeId
5      0.219
27     8.781
29     8.781
50     9.906
69     8.688
105    2.000
121    8.781
122      NaN
123    8.781
125      NaN
Name: 1, dtype: float64
```

```
recommended_jokes[recommended_jokes.isna()].index.tolist()
```

```
[122, 125]
```

```
for joke in jester_jokes_df[jester_jokes_df.jokeId.isin(recommended_jokes[~
↪recommended_jokes.isna()].index.tolist())].jokeText:
    print(joke)
    print('---')
```

```
Q.        What's O. J. Simpson's Internet address?
A.        Slash, slash, backslash, slash, slash, escape.

---
Clinton returns from a vacation in Arkansas and walks down  the
steps of Air Force One with two pigs under his arms.  At the bottom
of the steps, he says  to the honor guardsman, "These are genuine
Arkansas Razor-Back Hogs.  I got this one for Chelsea and this one for
Hillary."

The guardsman replies, "Nice trade, Sir."

---
An old Scotsmen is sitting with a younger Scottish gentleman and says the boy.
"Ah, lad look out that window. You see that stone wall there, I built it with
me own bare hands, placed every stone meself.  But do they call me MacGregor the
wall builder? No!

He Takes a few sips of his beer then says, "Aye, and look out on that lake and
eye that beautiful pier. I built it meself, laid every board and hammered each
nail but do they call me MacGregor the pier builder? No!

He continues..."And lad, you see that road? That too I build with me own bare
hands. Laid every inch of pavement meself, but do they call MacGregor the road
builder? No!"

Again he returns to his beer for a few sips, then says,
"Agh, but you screw one sheep..."

---
A guy goes into confession and says to the priest, "Father, I'm 80 years
old, widower, with 11 grandchildren. Last night I met two beautiful flight
attendants. They took me home and I made love to both of them. Twice."

The priest said: "Well, my son, when was the last time you were in
confession?"
 "Never Father, I'm Jewish."
 "So then, why are you telling me?"
 "I'm telling everybody."

---
This guys wife asks, "Honey if I died would you remarry?" and he replies,
"Well, after a considerable period of grieving, we all need
companionship, I guess I would."

She then asks, "If I died and you remarried, would she live in this
house?" and he replies, "We've spent a lot of time and money getting this
house just the way we want it. I'm not going to get rid of my house, I
guess she would."

"If I died and you remarried, and she lived in this house, would she
sleep in our bed?" and he says, "That bed is brand new, we just paid two
thousand dollars for it, it's going to last a long time, I guess she
would."

So she asks, "If I died and you remarried, and she lived in this house,
and slept in our bed, would she use my golf clubs?"
```

(continues on next page)

(continued from previous page)

```
"Oh no, she's left handed."

---
A couple of hunters are out in the woods in the deep south when one of them falls to␣
↪the ground. He doesn't seem to be breathing, and his eyes are rolled back in his␣
↪head. The other guy whips out his cell phone and calls 911. He gasps to the␣
↪operator, "My friend is dead! What can I do?" The operator, in a calm and soothing␣
↪voice, says, "Alright, take it easy. I can help. First, let's make sure he's dead."␣
↪There is silence, and then a gun shot is heard. The hunter comes back on the line.
↪"Okay. Now what??"
---
A drunk staggers into a Catholic Church, enters a confessional booth, sits down, but␣
↪says nothing. The Priest coughs a few times to get his attention but the drunk just␣
↪sits there. Finally, the Priest pounds three times on the wall. The drunk mumbles,
↪"Ain't no use knockin, there's no paper on this side either."
---
When most people claim to be "killing time", it's only an expression. When Chuck␣
↪Norris kills time, the minutes actually cease to exist.
---
```

```
for joke in jester_jokes_df[jester_jokes_df.jokeId.isin(recommended_jokes[recommended_
↪jokes.isna()].index.tolist())].jokeText:
    print(joke)
    print('---')
```

```
An astronomer, a physicist and a mathematician (it is said) were holidaying in␣
↪Scotland. Glancing from a train window, they observed a black sheep in the middle␣
↪of a field. "How interesting," observed the astronomer, "All Scottish sheep are␣
↪black!" To which the physicist responded, "No, no! Some Scottish sheep are black!"␣
↪The mathematician gazed heavenward in supplication, and then intoned, "In Scotland␣
↪there exists at least one field, containing at least one sheep, at least one side␣
↪of which is black."
---
An American tourist goes into a restaurant in Spain and orders the specialty of the␣
↪house. When his dinner arrives, he asks the waiter what it is. "These, senor,"␣
↪replied the waiter in broken English, "are the testicles of the bull killed in the␣
↪ring today." The tourist swallowed hard but tasted the dish and thought it was␣
↪delicious. So he comes back the next evening and orders the same item. When it is␣
↪served, he says to the waiter, "These testicles...are much smaller than the ones I␣
↪had last night." "Yes, senor," replied the waiter, "You see...the bull, he does not␣
↪always lose.
---
```