Descriptive analysis

Descriptive analysis (sometimes referred to as Exploratory analysis) is a crucial part in any kind of analysis. The goal is to “get to know your dataset” – to find outliers, irregularities and identify what transformations to do for the further stages (e.g. for modelling). Exploratory phase is largely an iterative process, where you develop understanding of the data by:

1. collecting questions about the data
2. answering them by transforming the data, summarizing and plotting it,
3. which leads to new questions :)

Prerequisites (recap)

The power of R comes from the vast universe of packages. Every time you need a new package you install it:

```r
install.packages("tidyverse")
```

In your script you will load packages by writing:

```r
library(tidyverse)
```

It is important to properly comment your code. If you use many new packages (libraries), it is wise to comment why you need them.

```r
library(data.table) # package to load big datasets much faster
```

Other useful functions

```r
getwd() # shows the directory you are in now.
setwd("path") # change the directory
```

Loading datasets

Now we load a dataset into R. Today we are analyzing data (artificial) of total sales on a weekly basis for two particular products in different stores.
There are many ways things can be done in R. For example, you can read in dataset using base functionality:

```r
dt <- read.table("/path/intro_dataset.csv", header=TRUE, sep=',')
```

However, you can also print with `fread` which is like `read.table` but faster and more convenient, which tries to be as automatic as possible:

```r
dt <- fread("/path/intro_dataset.csv")
```

Always check how the dataset looks like by 1) printing the header:

```r
head(dt) # alternatively use tail()
```

```r
## storeNum Year Week p1sales p2sales p1price p2price p1prom p2prom
country
## 1: 101 1 1 127 106 2.29 2.29 0 0
## 2: 101 1 2 137 105 2.49 2.49 0 0
## 3: 101 1 3 156 97 2.99 2.99 1 0
## 4: 101 1 4 117 106 2.99 3.19 0 0
## 5: 101 1 5 138 100 2.49 2.59 0 1
## 6: 101 1 6 115 127 2.79 2.49 0 0
## 1: US
## 2: US
## 3: US
## 4: US
## 5: US
## 6: US
dim(dt) # how many rows and columns
```

```r
## [1] 2080 10
```

2. Checking the structure:

```r
str(dt)
```

```r
## Classes 'data.table' and 'data.frame': 2080 obs. of 10 variables:
## $ storeNum: chr "101" "101" "101" "101" ...
## $ Year : int 1 1 1 1 1 1 1 1 1 1 ...
## $ Week : int 1 2 3 4 5 6 7 8 9 10 ...
## $ p1sales : int 127 137 156 117 138 115 116 106 116 145 ...
## $ p2sales : int 106 105 97 106 100 127 90 126 94 91 ...
## $ p1price : num 2.29 2.49 2.99 2.99 2.49 2.79 2.99 2.99 2.99 2.29 ...
## $ p2price : num 2.29 2.49 2.99 3.19 2.59 2.49 3.19 2.29 2.29 2.99 ...
```
Descriptive statistics

Summary statistics is displayed by:

```
summary(dt)
```

We see that `storeNum` is a character vector and represents store id. Let's check how many unique stores we have:

```
unique(dt$storeNum)  # displays all unique values of a vector
## [1] "101" "102" "103" "104" "105" "106" "107" "108" "109" "110" "111"
## [12] "112" "113" "114" "115" "116" "117" "118" "119" "120"
length(unique(dt$storeNum))  # number of unique stores in our dataset
## [1] 20
```
Discrete variables

Another useful inspection for discrete features (variables) is using frequency counts. This can be achieved by `table` function:

```r
table(dt$p1prom) # one-way frequency table
##
## 0  1
## 1872 208
table(promotion_p1=dt$p1prom, promotion_p2=dt$p2prom) # two-way frequency table
##
## promotion_p2
## promotion_p1  0  1
##     0 1616 256
##     1  176  32
```

The next question we might ask is how much varies price of the first and second product:

```r
summary(dt$p1price)
##
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
##  2.1900  2.2900  2.4900  2.5440  2.7900  2.9900
summary(dt$p2price)
##
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
##  2.2900  2.4900  2.5900  2.7000  2.9900  3.1900
```

We also would like to know what are the prices when the product’s price is reduced and is not. Price is a continuous variable. However, we already checked that it does not vary much.

We can display frequencies using `table` function:

```r
price_table <- table(promotion=dt$p1prom, price=dt$p1price) # how many times each product was promoted at each price level
```

But let’s see the percentages and margins instead of the counts. This way, we can compare it to the second product.

```r
# promotions are in rows, prices are in columns
# summing over rows
margin.table(price_table, 1)
##
##     promotion
## 0   1
##     0  1
```
## 1872 208
# summing over columns
\textbf{margin.table}(price_table, 2)

## price
## 2.19 2.29 2.49 2.79 2.99
## 395 444 423 443 375
\textbf{prop.table}(price_table) # cell %

## price
## promotion 2.19 2.29 2.49 2.79 2.99
## 0 0.17019231 0.19134615 0.18317308 0.19038462 0.16490385
## 1 0.01971154 0.02211538 0.02019231 0.02259615 0.01538462

\textbf{prop.table}(price_table, 1) # row percentages

## price
## promotion 2.19 2.29 2.49 2.79 2.99
## 0 0.18910260 0.21260680 0.20352558 0.21153852 0.18322652
## 1 0.19711544 0.22115385 0.20192311 0.22596153 0.15384621

\textbf{prop.table}(price_table, 2) # column percentages

## price
## promotion 2.19 2.29 2.49 2.79 2.99
## 0 0.89620253 0.89639640 0.90070922 0.89390519 0.91466667
## 1 0.10379747 0.10360360 0.09929078 0.10609481 0.08533333

### Continuous variables

For continuous variables, where data varies a lot, table is not feasible nor meaningful. Try for example running \texttt{table(dt$psales)}. The following table summarizes the functions of descriptive statistics that we discussed during the lecture.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min(x)</td>
<td>Minimum value</td>
</tr>
<tr>
<td>max(x)</td>
<td>Maximum value</td>
</tr>
<tr>
<td>mean(x)</td>
<td>Mean</td>
</tr>
<tr>
<td>median(x)</td>
<td>Median</td>
</tr>
<tr>
<td>var(x)</td>
<td>Variance</td>
</tr>
<tr>
<td>sd(x)</td>
<td>Standard deviation</td>
</tr>
<tr>
<td>quantile(x, probs=c(0.25, 0.5, 0.75))</td>
<td>Percentiles/Quartiles</td>
</tr>
</tbody>
</table>
Optional. There are recently some attempts to automate some of the descriptive procedures. One of such recent examples is the package `dataMaid`. Install and load the library and run command `clean(dt, output='html')`. Check the results.

### Data Transformations

Often exploring data involves a narrower approach. For example, instead of average sales of the first product we want to calculate average sales for each country, or for each shop. Or we want to filter the data and take a look at average sales when the product was promoted. If you have experience with other programming languages, your first approach for such analysis would be to run for loops. However, the power of R comes from the **vectorized** approach. In [StackOverflow](https://stackoverflow.com) we can find the following explanation:

Vectorization is the process of rewriting a loop so that instead of processing a single element of an array N times, it processes (say) 4 elements of the array simultaneously N/4 times.

A bit too technical, but let's see what it means on practice. Let's first try to calculate average sales per country using **classical** approach:

```r
countries = unique(dt$country)  # choose unique countries in our dataset
for(cn in countries){  # create a loop to pick a country from our country vector
  country_dt <- filter(dt, country==cn)  # filter the data, so that only data for this country is chosen
  country_mean <- mean(country_dt$p1sales)  # calculate the mean for this country
  print(c(cn, country_mean))  # print the result one-by-one
}
```

Now the vectorized approach would be:
Let’s measure the time taken for each of two approaches by writing `system.time` around our functions (the first is ‘for loop’ and the second is vectorized approach):

```
##    user  system elapsed
## 0.112   0.006   0.119
##    user  system elapsed
## 0.004   0.001   0.006
```

And this is on a very tiny dataset! On bigger sizes the difference becomes much more crucial. Now, let's train our data manipulation skills a bit more:

```
dt %>%
group_by(country) %>%
filter(p1prom == 1) %>%
summarise(counts = n())
```
```
dt %>%
group_by(country) %>%
select(contains("p")) %>%
filter(p1prom == 1 & p2prom == 1) %>%
summarise_all(funs(mean, sd), na.rm=TRUE)
```

I often use for the reference the following [dplyr cheatsheet](https://dplyr.tidyverse.org/).

## Visualization

Let’s try to practice how to do charts for our later analysis, as we will do a lot of graphs along the course. It is important to remember what is the purpose of your plots. There are plots with the main goal to communicate the message for the decision-makers or consumers. These plots have to clearly convey the message that was found and investigated earlier. For the exploratory analysis the graphs facilitate the discovery. We have to learn how to detect interesting and unexpected patterns and what questions to ask next.
Bar chart. Multi-set bar chart

The following chart displays how many transactions per country were made.

```r
ggplot(data = dt) + geom_bar(mapping = aes(x = country))
```

Compare it to the results of the table:

<table>
<thead>
<tr>
<th>country</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>AU</td>
<td>104</td>
</tr>
<tr>
<td>BR</td>
<td>208</td>
</tr>
<tr>
<td>CN</td>
<td>208</td>
</tr>
<tr>
<td>DE</td>
<td>520</td>
</tr>
<tr>
<td>GB</td>
<td>312</td>
</tr>
<tr>
<td>JP</td>
<td>416</td>
</tr>
<tr>
<td>US</td>
<td>312</td>
</tr>
</tbody>
</table>
The `ggplot2` plots are very flexible. Let's try to beautify it a bit:

```r
ggplot(data = dt) +
  geom_bar(mapping = aes(x = country), fill='#f1e8ca',
           color='#745151', alpha=0.8) + theme_bw(base_size=28)
```

The code `'#f1e8ca'` or `'#745151'` are color codes that can be found, for example, here:  [color codes](http://example.com/color-codes).

Multi-chart bar:

```r
dt %>%
  mutate(p1prom = as.factor(p1prom)) %>%
  ggplot() +
  geom_bar(mapping = aes(x = country, fill=p1prom),
           position='dodge')
```
Histogram. Density

Histogram shows the distribution of continuous variable. In a histogram the x-axis is partitioned into equal bins. The height of a bar shows how many observations fall into each bin. The results of the histogram heavily depend on the chosen bin size.

```r
p1 <- ggplot(data = dt) + geom_histogram(mapping = aes(x = p1sales))
p2 <- ggplot(data = dt) + geom_histogram(mapping = aes(x = p1sales), binwidth=50)
p3 <- ggplot(data = dt) + geom_histogram(mapping = aes(x = p1sales), binwidth=1)
grid.arrange(p1,p2,p3, ncol=3) # for this function you will need gridExtra package (install.packages("gridExtra"), library(gridExtra))
```
Another plot for the distribution is density plot. It is a smoothed histogram that provides a general estimate of the distribution. The area under the distribution is always sums to 1. It helps to estimate general tendency, however it comes at a cost, as y-axis is difficult to interpret.

```r
ggplot(data = dt) + geom_density(mapping = aes(x = p1sales), fill='#f1e8ca')
```
It is more helpful to compare several distributions via density plots:

```r
ggplot(data = dt) +
  geom_density(mapping = aes(x = p1sales), fill='#f1e8ca', alpha=0.3) +
  geom_density(aes(x=p2sales), fill = '#745151', alpha=0.3) +
  xlab('sales')
```
While histogram is a good way to show the distribution of one continuous variable, if we want to compare several distributions, it may become tricky. Densities are good, if there are not too many either. We could overlay several histograms, but if the scale between two histograms vary a lot, it is difficult to analyze:

**Boxplot**

While histogram is a good way to show the distribution of one continuous variable, if we want to compare several distributions, it may become tricky. Densities are good, if there are not too many either. We could overlay several histograms, but if the scale between two histograms vary a lot, it is difficult to analyze:
A better approach would be to use boxplots.

```r
ggplot(data = dt, mapping = aes(x = storeNum, y = p1sales)) + geom_boxplot() + theme_bw()
```
Boxplots help to quickly grasp the information about several distributions and compare it.

**Line chart**

Line charts are often the easiest to understand :). Usually, the most common way is to use it for time-series, e.g. we want to observe a behavior over time.

```r
dt %>%
  group_by(country) %>%
  summarise(p1sales=sum(p1sales)) %>%
  ggplot(aes(x=country, y=p1sales, group=1)) + geom_line() + theme_bw()
```
Scatter plot. Correlation

Two continuous variables can be visualized using scatterplot. It shows the relationship between these variables. We will later observe, how such relationship helps in models. Let’s first observe what are the basic patterns we want (or don’t want) to see in the data:
The relationship between two variables can be expressed with the correlation coefficient. However, it only shows whether there is a linear relationship. For example, the correlation coefficients for the figures above are correspondingly:

\[
\text{cor}(\text{df1}$x$, \text{df1}$y); \text{ cor}(\text{df2}$x$, \text{df2}$y); \text{ cor}(\text{df3}$x$, \text{df3}$y)
\]

```
## [1] 0.9988615
## [1] 0.01823327
## [1] 0.5416792
```

**Melting/Casting**

There is a particular concept of long and wide data formats, as we discussed during the lecture. For example, when we made plots of two density plots for product 1 and product 2, it was not the optimal way to do it. We can tidy our data. The following figure from *R for Data Science* illustrates it very well:
<table>
<thead>
<tr>
<th>storeNum</th>
<th>Year</th>
<th>Week</th>
<th>p1sales</th>
<th>p2sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1</td>
<td>1</td>
<td>127</td>
<td>106</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>2</td>
<td>137</td>
<td>105</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>3</td>
<td>156</td>
<td>97</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>4</td>
<td>117</td>
<td>106</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>5</td>
<td>138</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>6</td>
<td>115</td>
<td>127</td>
</tr>
</tbody>
</table>

```
gathered_dt <- initial_data %>%
gather(p1sales, p2sales, key = "product", value = "sales")
head(gathered_dt)
```

<table>
<thead>
<tr>
<th>storeNum</th>
<th>Year</th>
<th>Week</th>
<th>product</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1</td>
<td>1</td>
<td>p1sales</td>
<td>127</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>2</td>
<td>p1sales</td>
<td>137</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>3</td>
<td>p1sales</td>
<td>156</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>4</td>
<td>p1sales</td>
<td>117</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>5</td>
<td>p1sales</td>
<td>138</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>6</td>
<td>p1sales</td>
<td>115</td>
</tr>
</tbody>
</table>

Now we can use it to plot the densities in a more natural way:

```
 ggplot(data = gathered_dt) +
 geom_density(mapping = aes(x = sales, fill=product),
 alpha=0.3) +
```
A heat map (or heatmap) is a graphical representation of data where the individual values contained in a matrix are represented as colors. It is really useful to display a general view of numerical data, not to extract specific data point.

library(ggplot2)

df.team_data <- expand.grid(teams = c("Team A", "Team B", Team C", "Team D"), metrics = c("Metric 1", "Metric 2", "Metric 3", "Metric 4", "Metric 5"))
```r
# add variable: performance
set.seed(41)
df.team_data$performance <- rnorm(nrow(df.team_data))

# inspect
head(df.team_data)

# PLOT: heatmap
ggplot(data = df.team_data, aes(x = metrics, y = teams)) + geom_tile(aes(fill = performance))
```