Assignment 1 Data analytics & communication

Micky Labreche s4021290 & Pavel Katsitadze s3240932

25-11-2021

1 Finding the gaps in Bachelor projects

1.1 Brief summary of the research

The considered thesis is: A tableau prover for GL provability logic by Tim van Loo, (https://fse.studentheses.ub.rug.nl/15402/1/AI_BA_2017_VanLoo.pdf). The bachelor project concerns itself with the implementation of a tableau prover using the C programming language by method of a priority queue. Three methods for computing the provability were compared for length of the produced tableaux and computing time.

Making a semantic tableaux is a computable method for proving statements about logical formulas. For this project, the GL logic is chosen. GL (Gödel-Löb) is a provability logic, which means it concerns itself with the "square" operator (meaning: "it is provable that"). The research question itself is: "In a tableau prover for provability logic that runs in PSPACE, does modifying the input to a logically equivalent formula using less operators before constructing the tableau influence run time?" The focus for the program design was to make the output as close to human as possible, and a high level overview for the program structure was given and pseudocode functions were added in the appendix. To measure the computing time, the clock function from the standard c library (time.h) was used. It was mentioned how the test formulae were selected. In the results section it was shown that the different methods do produce different results in both tableaux length and computing time, and the differences were discussed. The thesis concluded that "It seems that for some input formulae, the form greatly influences tableau sizes as well as run time", but that there was no consistent pattern among the compared methods for this, as increases in complexity of the tableaux happened independently of the methods used. It is mentioned that the findings were not generalizable, due to lack of consistent benchmarks in GL.

1.2 Flaws in the research

The flaws that can be found in this thesis mostly stem from reproducibility issues, some of which are inherent to the subject matter and are alluded to in the discussion section of the paper. Namely that some test formulae were not computed because of RAM limitations. It is not mentioned which formulae these were, and so it is not possible to know upon which formulae the conclusions are based. This is further compounded because the origins of the selected test formulae were given, but the specific formulae themselves were not. Secondly, the lack of printing of complete tableaux, which was due to them being too long to be readable. Sadly, this also means there is no way of knowing whether the same intermediate results are obtained when reproducing this research. There is also a lack of mention of specifics of the system, other than the system having 16GB of RAM, a 4 GHz CPU and running Linux mint 17.3. Furthermore, only a high-level overview of the implemented functions is given. It is possible that in independent replication slightly different implementations of the functions are used, resulting in different outcomes.

1.3 Improvements

Most of the flaws underlined can be solved with good record keeping and disclosure. The lack of information given in the thesis could be due to either of the two options. It should also be considered that adding all of

this information to the paper itself would make it too dense to read. A possibility would be to store it with the Open Science Framework. Other than that, no generalized claims are made, nor is there a statistical analysis or data which could have been manipulated.

2 Replicable and reproducible Bachelor project

2.1 Potential bachelor project

A possible bachelor project would be to test whether working memory in students can be improved through repeated training. The idea is to ask students to participate in two spaced working memory tests (n-back, for example), with a semester of time between the tests. Half of these students will be asked to train their working memory with an app at regular intervals in between these two tests. Assignment would happen at random.

2.2 Reproducibility and replicability

Reproducibility is the possibility for independent researchers to repeat a study in the same way that it was done the first time. Replicability is the ability for the researchers to then obtain the same (or significantly similar) results. Reproducibility is dependent on the transparency of the methods, program (versions), and scripts used, while replicability has more to do with bias and data selection. If you can't figure out how a cake was made, it wasn't reproducible. If you have the instructions, follow them exactly, and the cake turns out completely different from the pictures it wasn't replicable.

2.3 Reproducible methods

The best method for improving reproducibility would be to keep track of how the data is gathered, and which training method is used. Both the testing and training information should be archived. (e.g. the questions used, spacing of training, which application and version is used for training)

2.4 Replicable methods

The replicability of such a project would largely depend on obtaining a large sample, so it is less dependent on individual variances, and not generalizing results to larger populations. The alternative would be to ask a large number of people over the globe to participate in such a study.

3 Tidyverse exercise

Now we explore the functions inside the libraries of tidyverse. We use R Studio in combination with R Markdown to create the code and the document.

3.1 Dataset

The dataset I chose is a Pokemon dataset. Pokemon are fictional creatures who can fight eachother with special abilities. In this dataset all different Pokemon are described as they all have different properties. Examples of variables are happiness, weight, attack values and health points.

3.2 Mutate command

```
pokemon %>%
  mutate(hp2 = hp*2) %>%
  select(name, hp, hp2) %>%
  head
```

```
##         name hp hp2
## 1  Bulbasaur 45  90
## 2    Ivysaur 60 120
## 3   Venusaur 80 160
## 4 Charmander 39  78
## 5 Charmeleon 58 116
## 6  Charizard 78 156
```

By using the mutate command, we can add a new variable and fill the cells based on the existing the values of existing variables in de dataset. In this case we use the following command to get a new column filled with two times the health points of each pokemon. This column is not present in the old data set. This would result in the following error: "undefined columns selected".

3.3 Count command

```
pokemon %>%
  count(type1) %>%
  head
```

```
##      type1  n
## 1      bug 72
## 2     dark 29
## 3   dragon 27
## 4 electric 39
## 5    fairy 18
## 6 fighting 28
```

When using this command we can select a certain categorical variable an count how many instances of a category are in the column. This returns a dataframe with in the left column the categories and in the right column the counts. In this case we can see the pokemon type. Before, each category was written in each cell, now the "type" data is more clear to analyze.

3.4 Filter command

```
filtered <- pokemon %>%
  filter(generation > 6)
```

This method is used for filtering out observations. Only the rows that fulfill the boolean condition in the filter argument will remain in the resulting dataframe. In this case we only want to keep the pokemon that are higher than sixth generation. Before we had a lot of rows, now there is only some left.

```
nrow(pokemon)
```

```
## [1] 801
```

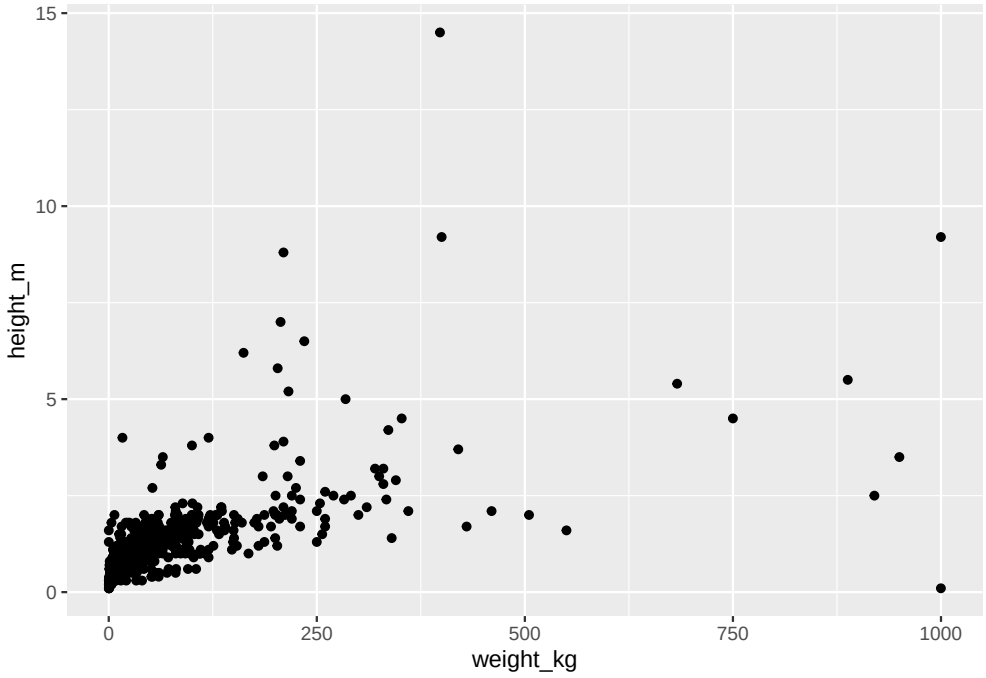
```
nrow(filtered)
```

```
## [1] 80
```

3.5 Ggplot

Now we make a plot that shows the relation between the height and the weight of the pokemon. This can be easily done with ggplot and plotting the points on a graph. As you can see most pokemon have a small height and a small weight. However there seems to be a correlation: the higher the pokemon, the more they weigh. The furher we go to the extremes the less dense the cloud becomes.

```
ggplot(data = pokemon) +
  geom_point(mapping = aes(x = weight_kg, y = height_m))
```



3.6 Summarizing

```
pokemon %>%
  group_by(type1) %>%
  summarise(test=mean(attack)) %>%
  head
```

```
## # A tibble: 6 x 2
##   type1     test
##   <chr>    <dbl>
## 1 bug       70.1
## 2 dark      87.8
## 3 dragon   106.
## 4 electric  70.8
## 5 fairy     62.1
## 6 fighting  99.2
```

With this we can group a category just like before. Instead of counting we can choose our own summarization function. In this case we can view the mean attack points for each type of pokemon.

3.7 Spreading and gathering data

```
pokemon %>%
  pivot_wider(names_from = type1, values_from = defense) %>%
  select(name, 40:45) %>%
  head
```

```
## # A tibble: 6 x 7
##   name       grass  fire water   bug normal poison
##   <chr>      <int> <int> <int> <int>  <int>  <int>
## 1 Bulbasaur     49    NA    NA    NA     NA     NA
## 2 Ivysaur       63    NA    NA    NA     NA     NA
## 3 Venusaur     123    NA    NA    NA     NA     NA
## 4 Charmander    NA    43    NA    NA     NA     NA
## 5 Charmeleon    NA    58    NA    NA     NA     NA
## 6 Charizard     NA    78    NA    NA     NA     NA
```

This command makes the data frame wider based on two variables. Extra columns are added to the right for every category of a variable. In this case we choose the pokemon type as the names_from parameter and the defence points as the values_from parameter. In cases that pokemon are not a certain category, there is no value to display in the column corresponding to that category. Then the NA value is put in.

3.8 Seperating and uniting

We first have a column in which abilities are combines into a string which represents a list. This is visible in the original dataset below.

```
pokemon %>%
  select(name, abilities) %>%
  head
```

```
##         name                   abilities
## 1  Bulbasaur ['Overgrow', 'Chlorophyll']
## 2    Ivysaur ['Overgrow', 'Chlorophyll']
## 3   Venusaur ['Overgrow', 'Chlorophyll']
## 4 Charmander    ['Blaze', 'Solar Power']
## 5 Charmeleon    ['Blaze', 'Solar Power']
## 6  Charizard    ['Blaze', 'Solar Power']
```

Now we use the separate method to seperate these values into different columns. The columns are named according to the ability number (a1, a2, ...). Not every pokemon has two abilities, some have less. These values are noted as NA. Some have more, these values are merged into the last ability. You can customize the separating behaviour with a regular expression.

```
pokemon %>%
  separate(abilities, c("none", "a1", "a2"), extra = "merge") %>%
  select(name, a1, a2) %>%
  head
```

```
##         name       a1            a2
## 1  Bulbasaur Overgrow Chlorophyll']
## 2    Ivysaur Overgrow Chlorophyll']
## 3   Venusaur Overgrow Chlorophyll']
## 4 Charmander    Blaze Solar Power']
## 5 Charmeleon    Blaze Solar Power']
## 6  Charizard    Blaze Solar Power']
```