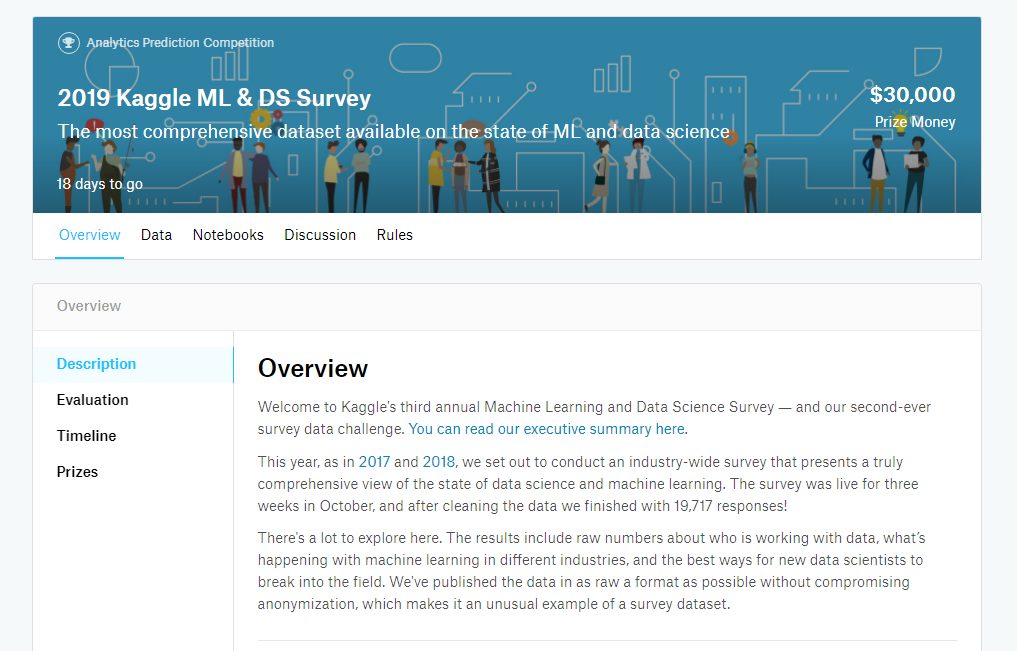
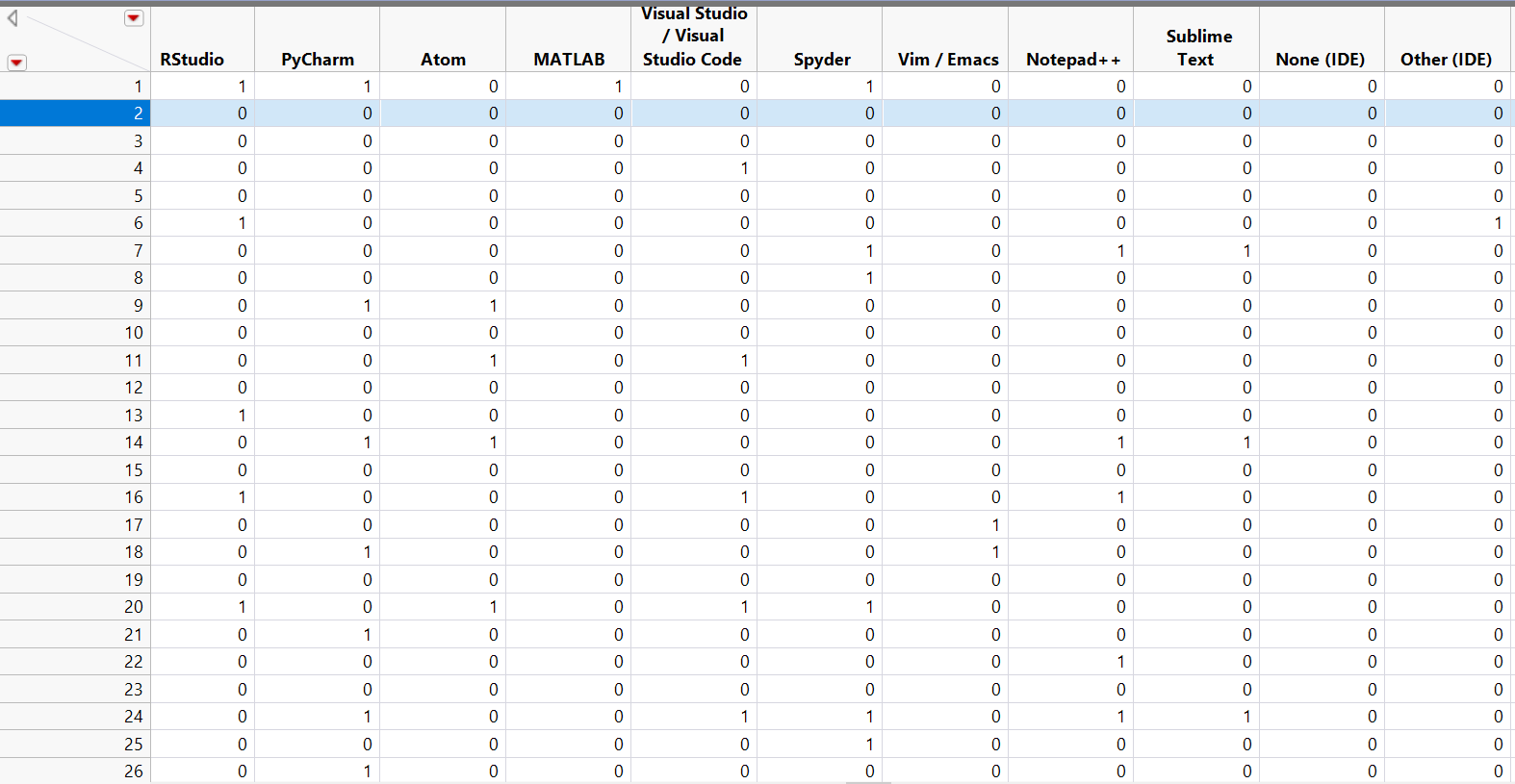
****

Goal:

The main goal of what Is hoped to be achieved here to get the various respondents to the Kaggle Survey somehow categorized through the answers they have given to the survey. Given the nature of the dataset, this may be tough to achieve, and our results may come up inconclusive going off what other people have already done with this, though it still is worth trying. The challenge has already finished by the time work on this project begun, but this was meant to be more of personal exercise.

Method #1 MCA:

Using the responses from the multiple-choice questions from the survey, we will attempt Multi-Correspondence Analysis. The first step is to turn our multiple-choice answers into a transactions list. This was done in JMP as its “Make Indicator Columns” function simplifies the process immensely. Certain columns in the transaction list were also re-labeled so that they could be understood better when looking at an MCA plot. A sample of what the data now looks like is shown below.



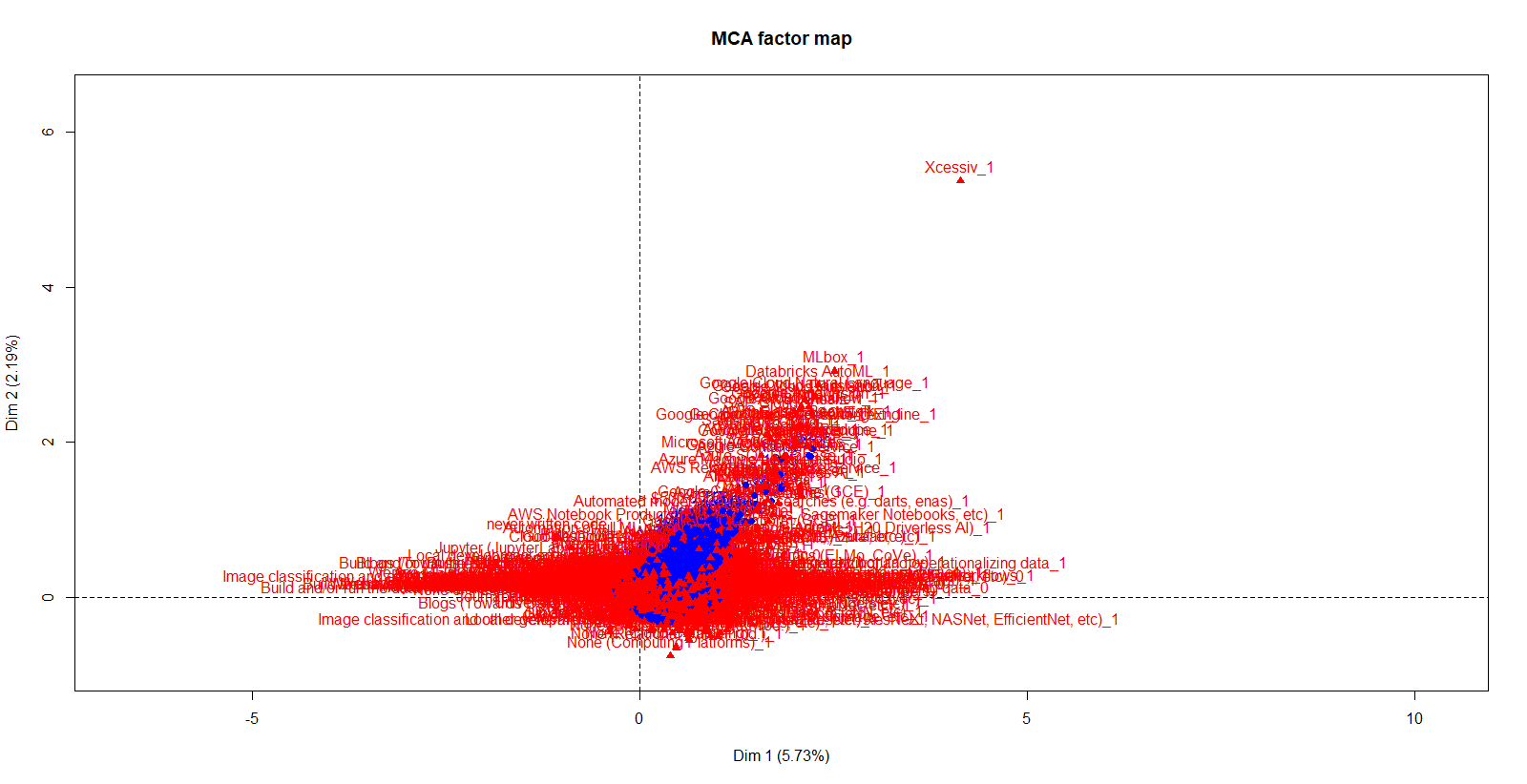
Next it will put it in R to complete the analysis.

Well start with everything and the kitchen sink just to see what there is to find.

Multi.mat = apply(MCA\_Ready,2,as.factor)

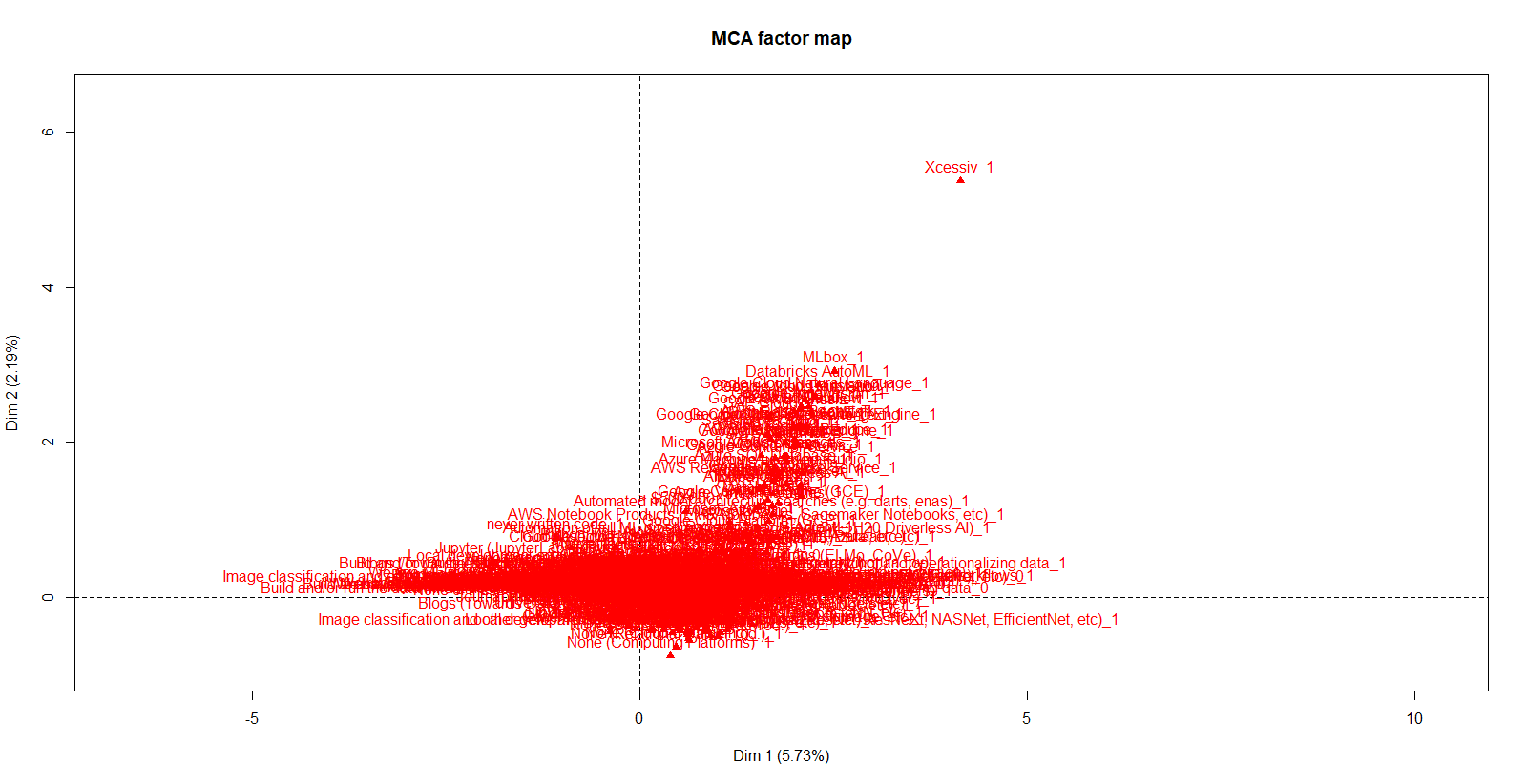
Mutli.MCA = MCA(Multi.mat)

plot(Mutli.MCA)

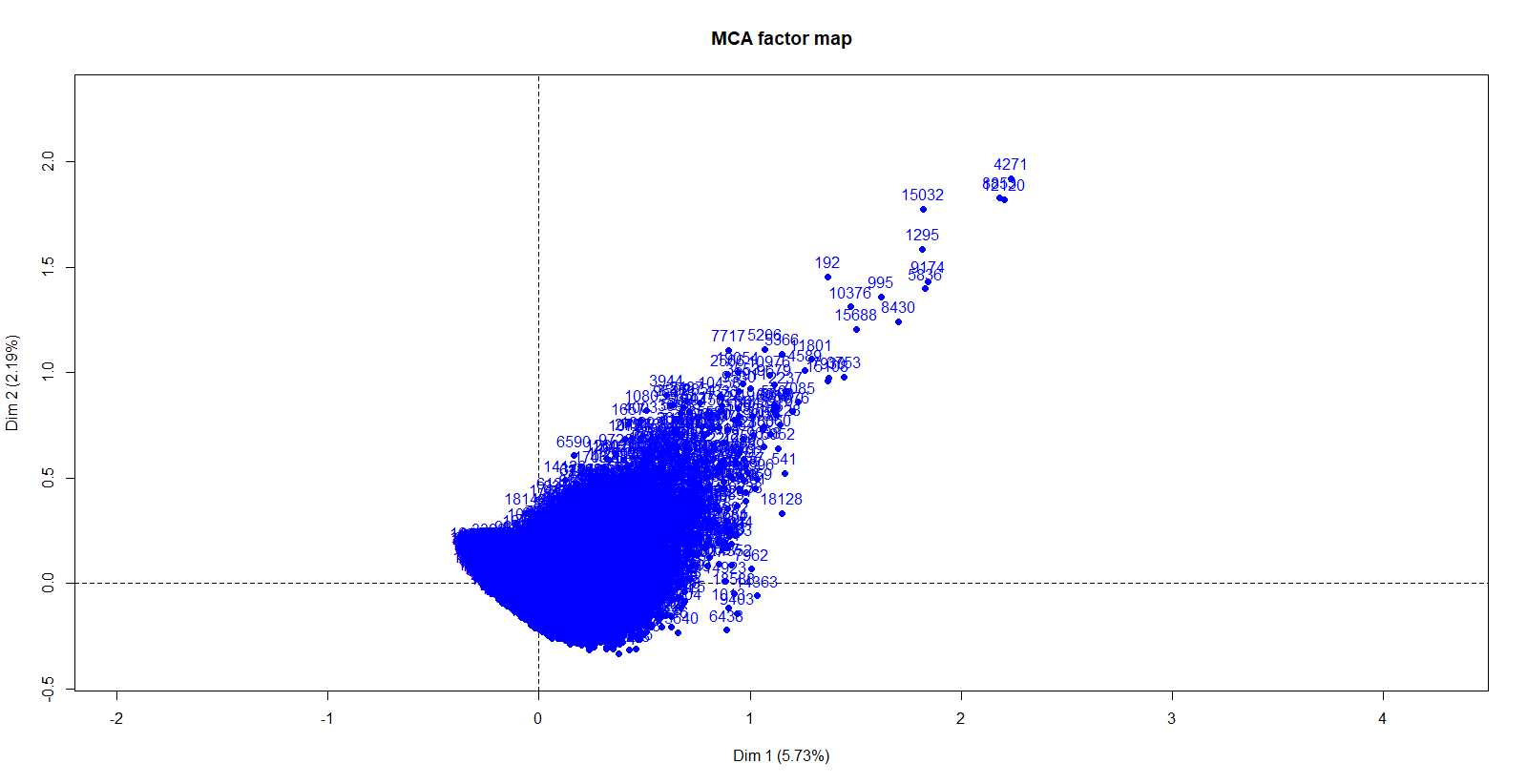


Users of Xcessiv are clear outliers, as only 28 users out of 19,717 say they used it. In addition, the overall structure of this plot is not optimal, although it probably isn’t bad enough to learn nothing from proceeding.

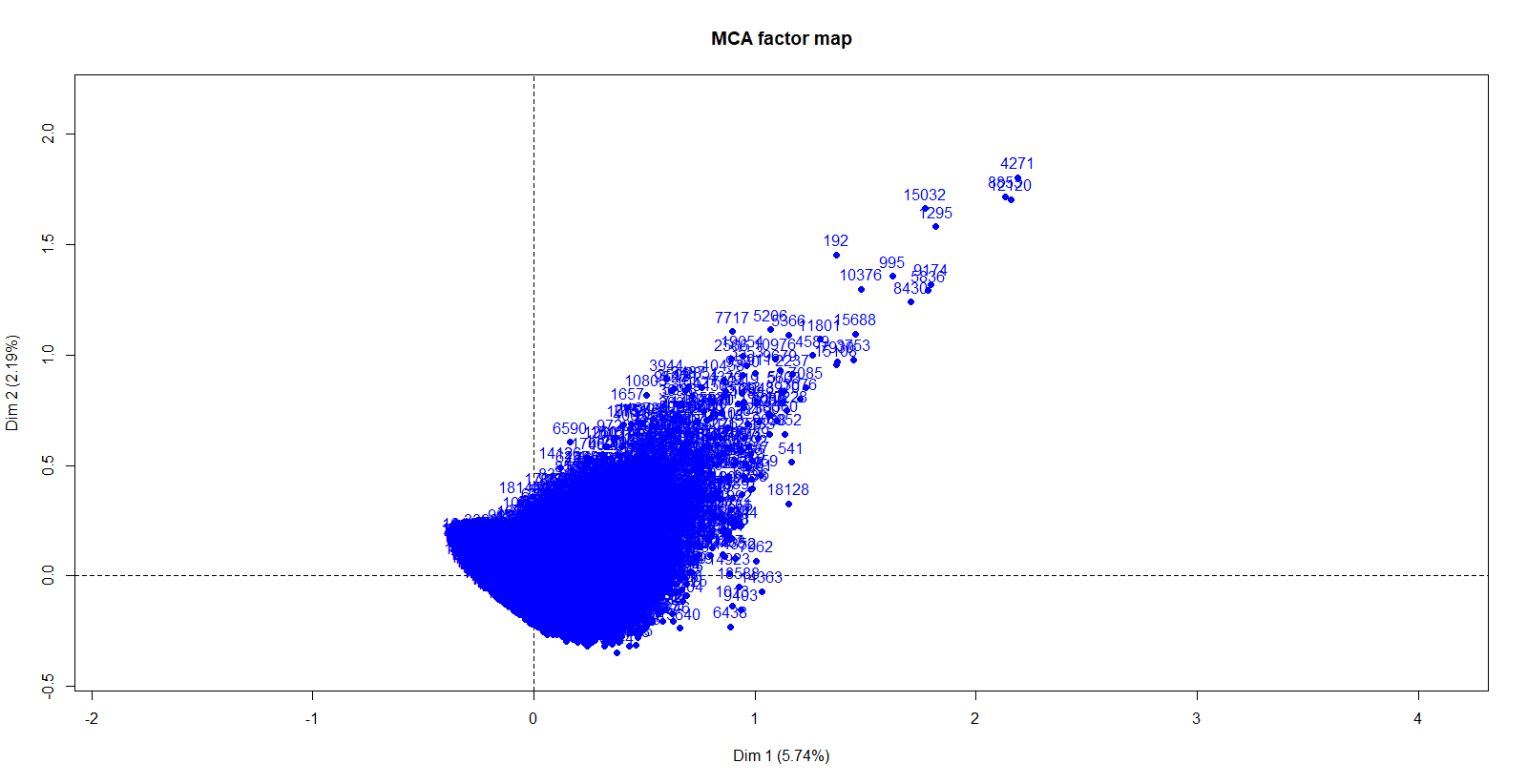
plot(Mutli.MCA, invisible = c("ind"))



plot(Mutli.MCA, invisible = c("var"))



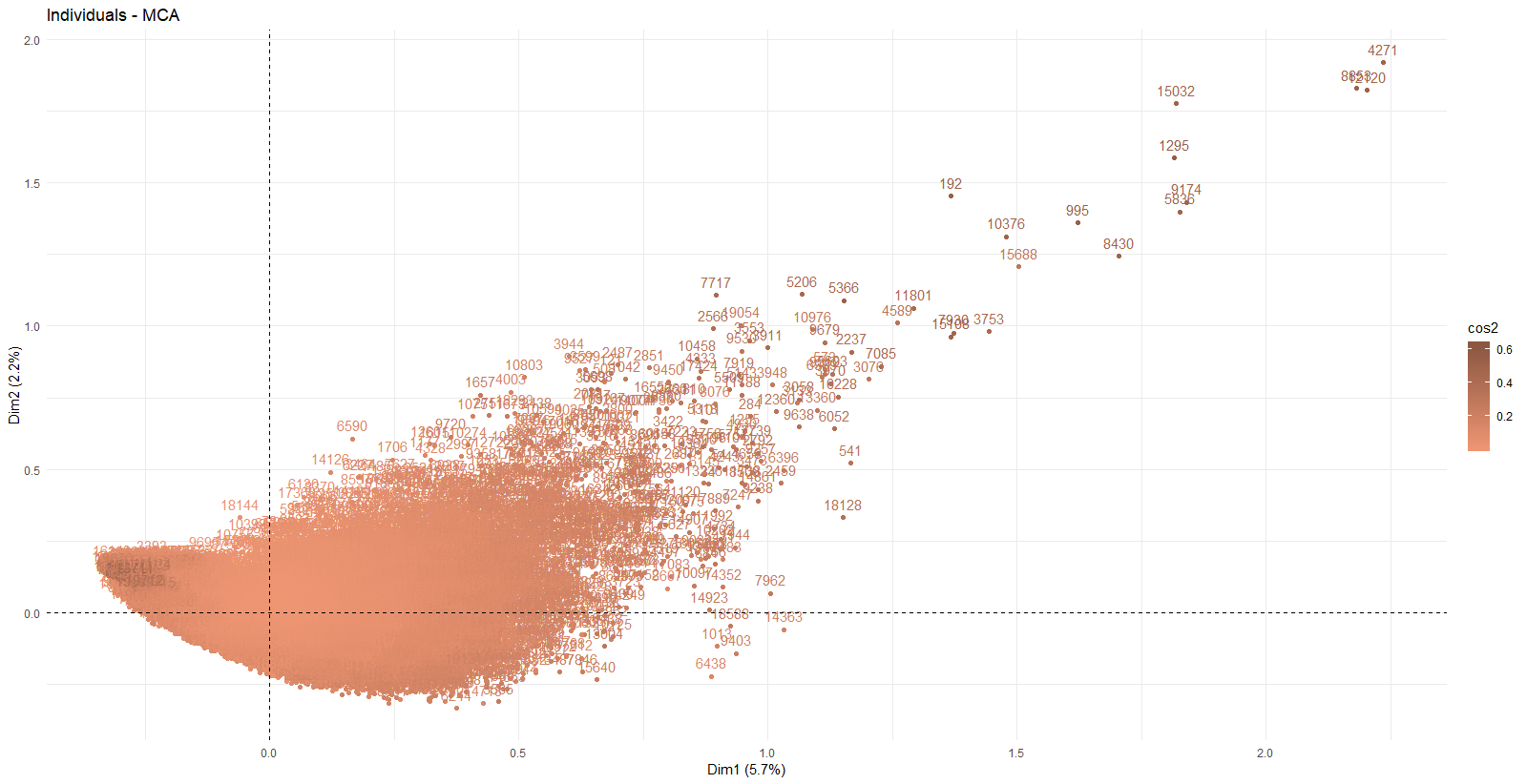
You can see how it (Xcessiv) pulls those users away. While one could remove all 28 of those individuals as outliers for that singular reason, because an overall trend in that direction beyond just them is seen, It was decided that at least for now I will leave it as is.



fviz\_mca\_ind(Mutli.MCA, col.ind="cos2") +

scale\_color\_gradient2(low="lightsalmon", mid = "lightsalmon2",

high="lightsalmon4")+theme\_minimal()

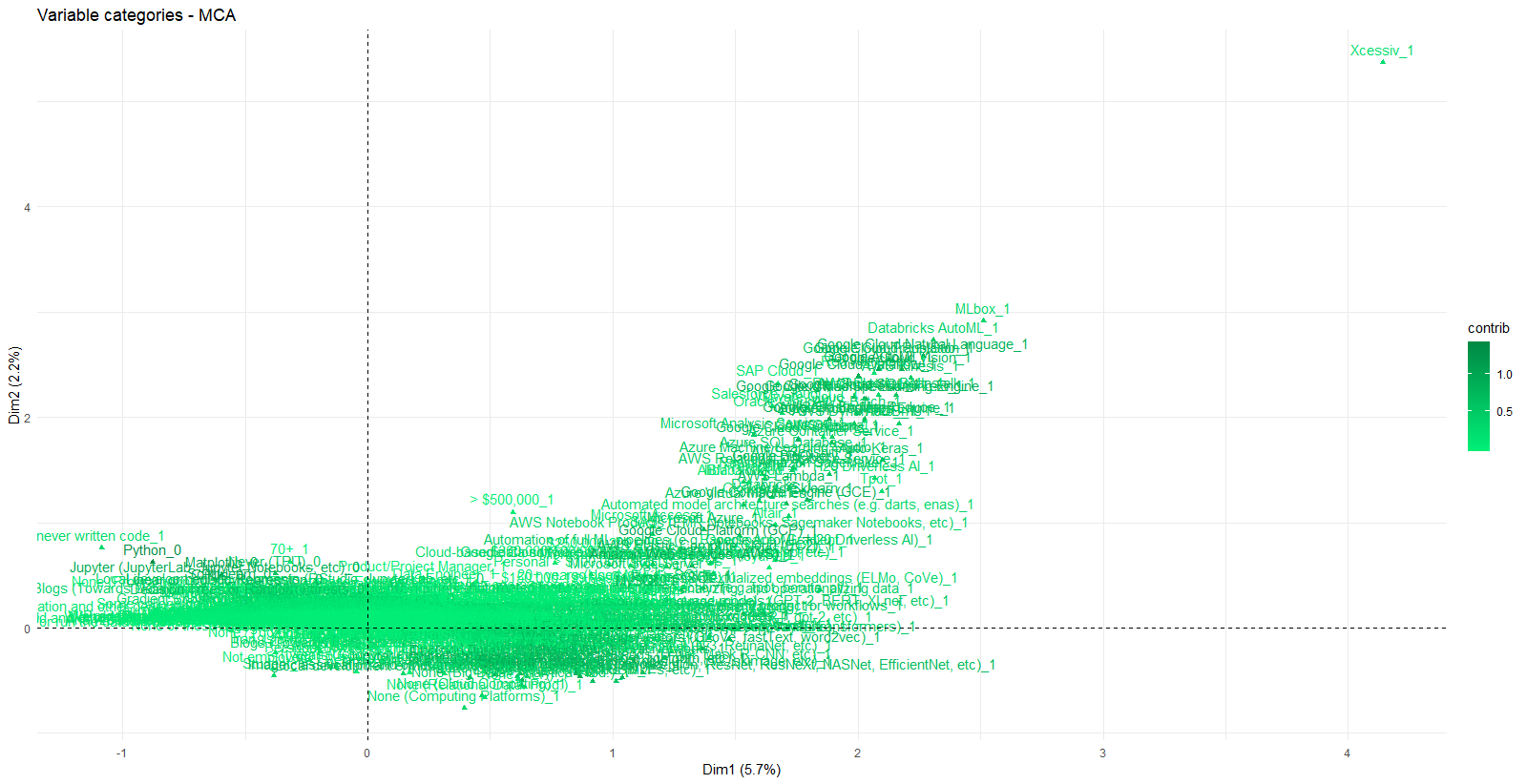


Points on the Fringes are much better represented then ones more in the middle. However, given that this survey looks at so much, there are bound to be massive groups of individuals separated by only trivial differences. The more interesting topic of discussion is likely to be what is pulling those people away from that mass. Furthermore, we should note that the entire bottom left quadrant is disregarded looking purely at this metric, but we do begin to see a pickup in the quadrant above. Additionally, Three groups are made distinct, with them being the trailing users in the top right, the thicker cluster on the bottom left, and point that emerges near the corner of the top right quadrant.

fviz\_mca\_var(Mutli.MCA, col.var="contrib") +

scale\_color\_gradient2(low="springgreen", mid = "springgreen2",

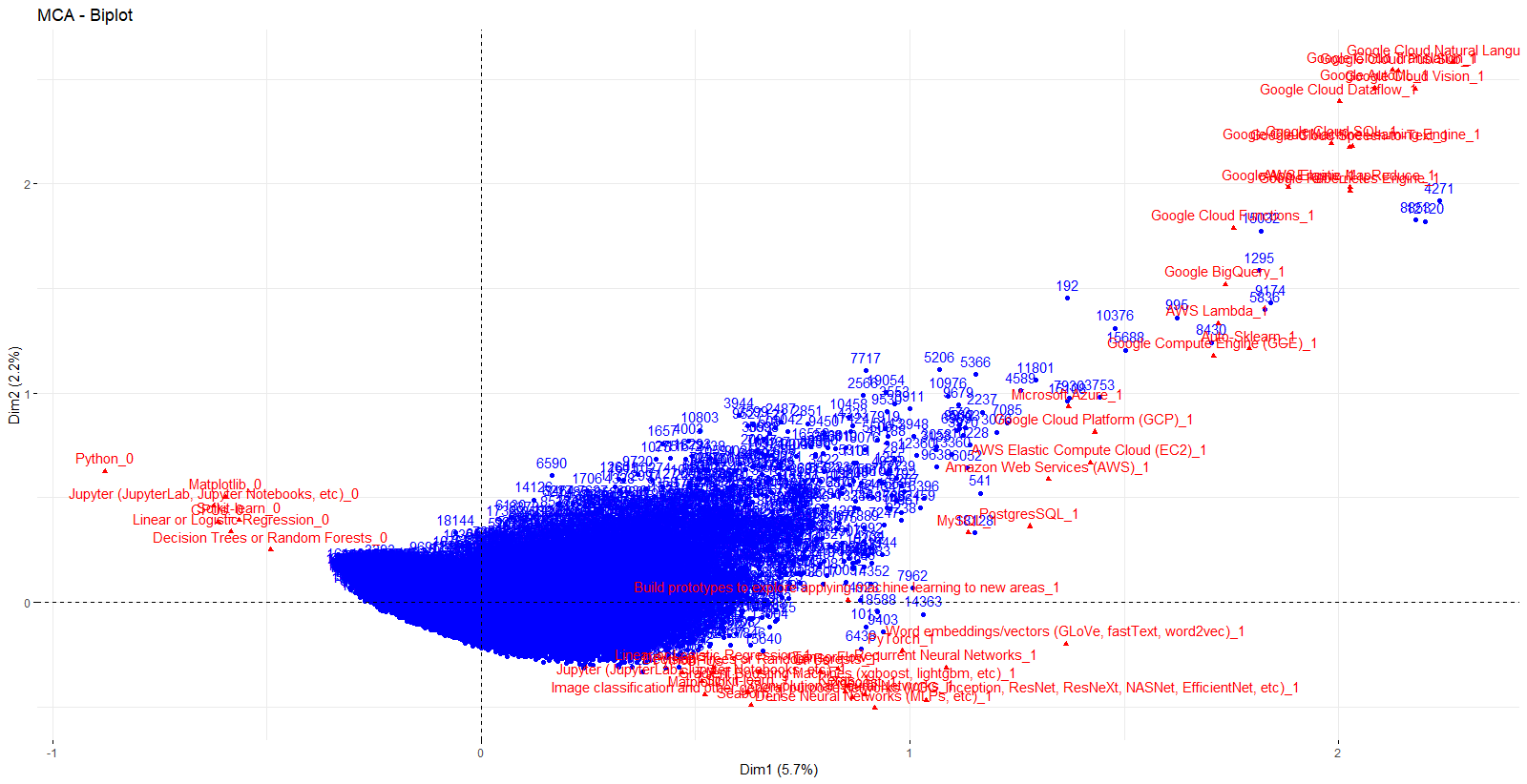
high="springgreen4")+theme\_minimal()



The contributions are similarly structured with the edges performing better, with some features of Kaggle users contributing much more than others. Also, The strength displayed by certain variables lines up with the main patterns seen in the individuals as expected, and we can already begin to see some features that place respondent in one camp or another, such as Python usage, google products usage, and users who work more surface level (i.e. don’t code, aren’t familiar with machine learning, etc.). To look at this more in-depth, we will create biplots that filter only on the top “x” number of variable contributions. Xcessiv also has a very low contribution score, so removing it likely would not change much, so It will be left it in.

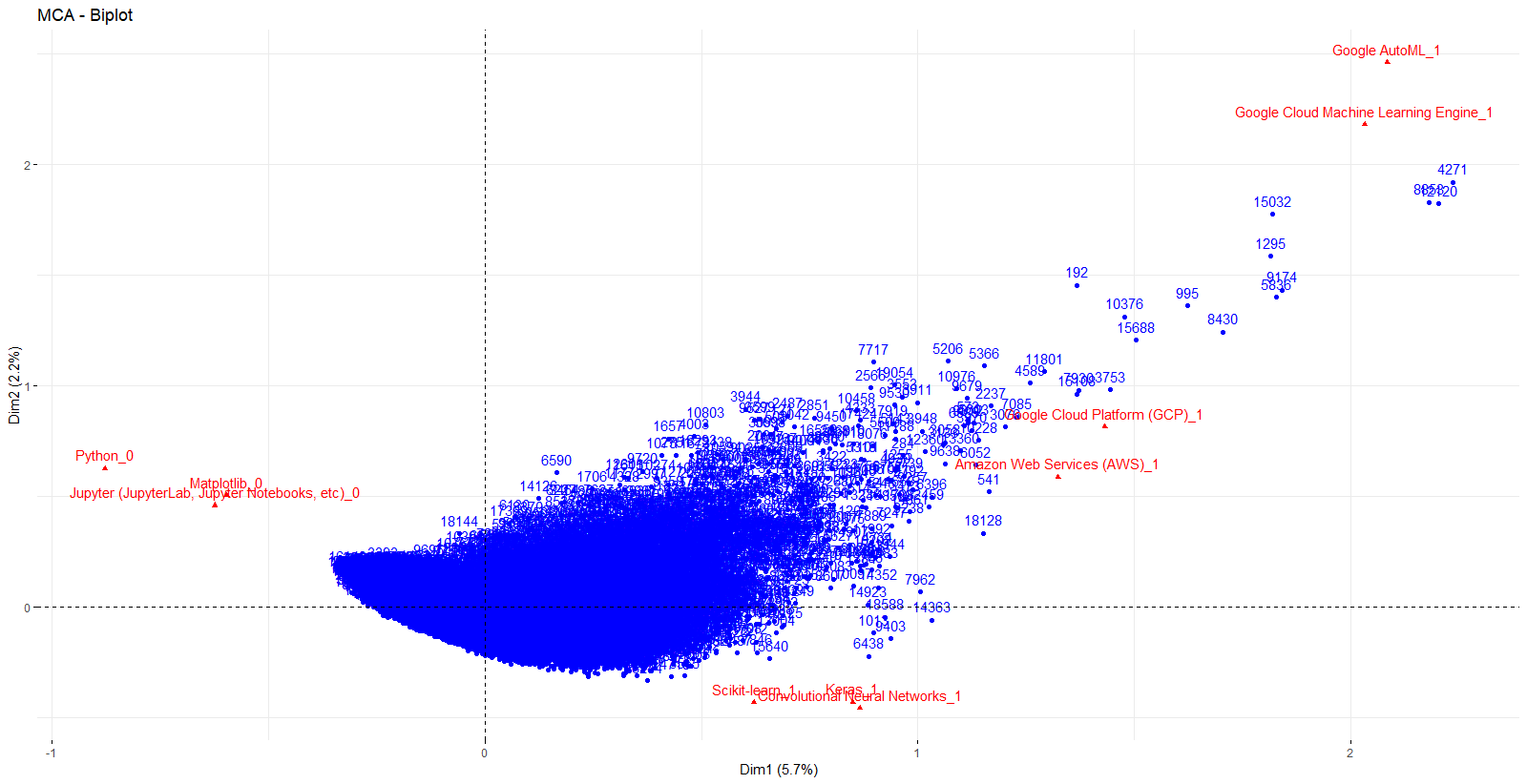
X = 50

fviz\_mca\_biplot(Mutli.MCA, select.var = list(contrib = 50) )



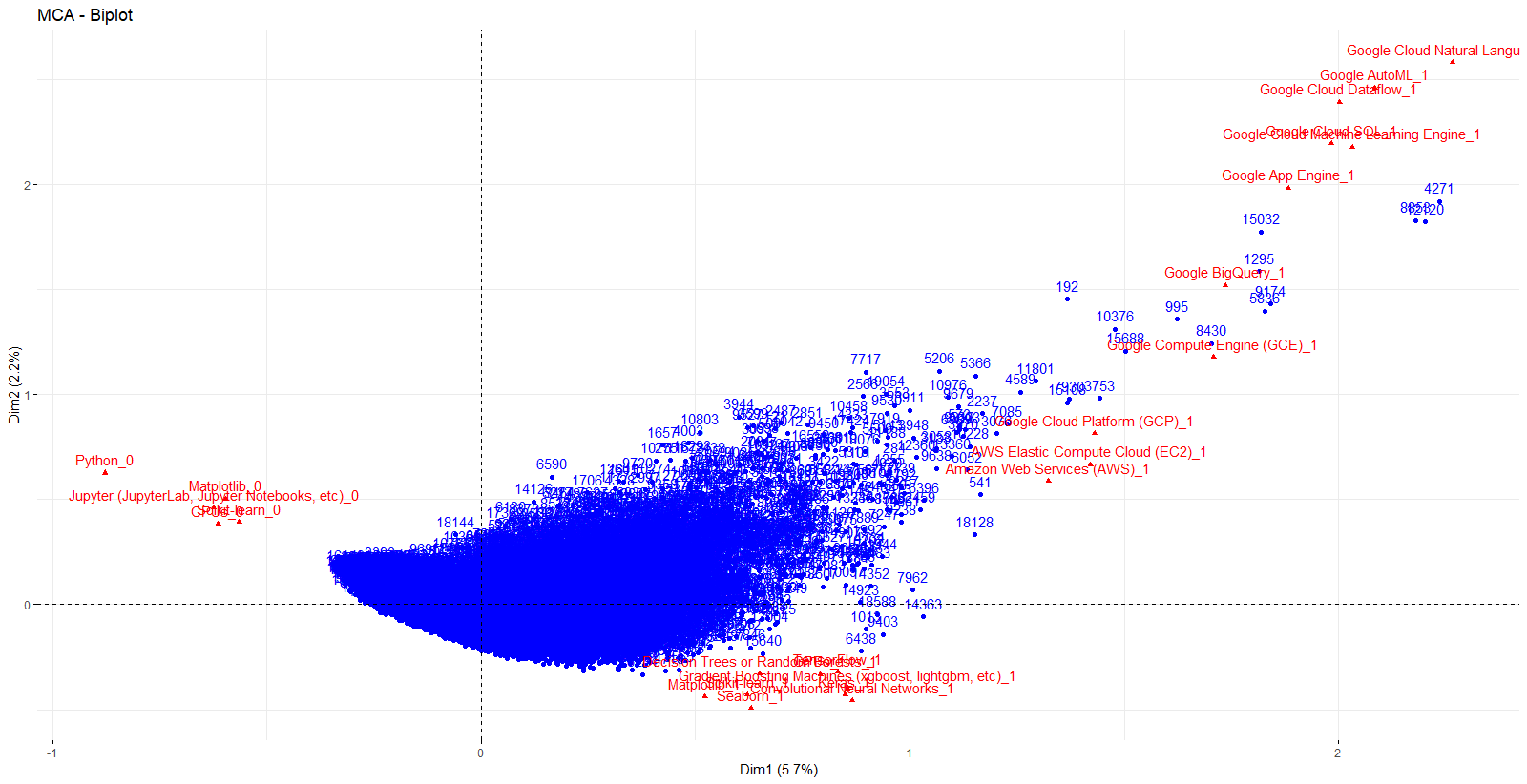
X = 10

fviz\_mca\_biplot(Mutli.MCA, select.var = list(contrib = 10) )



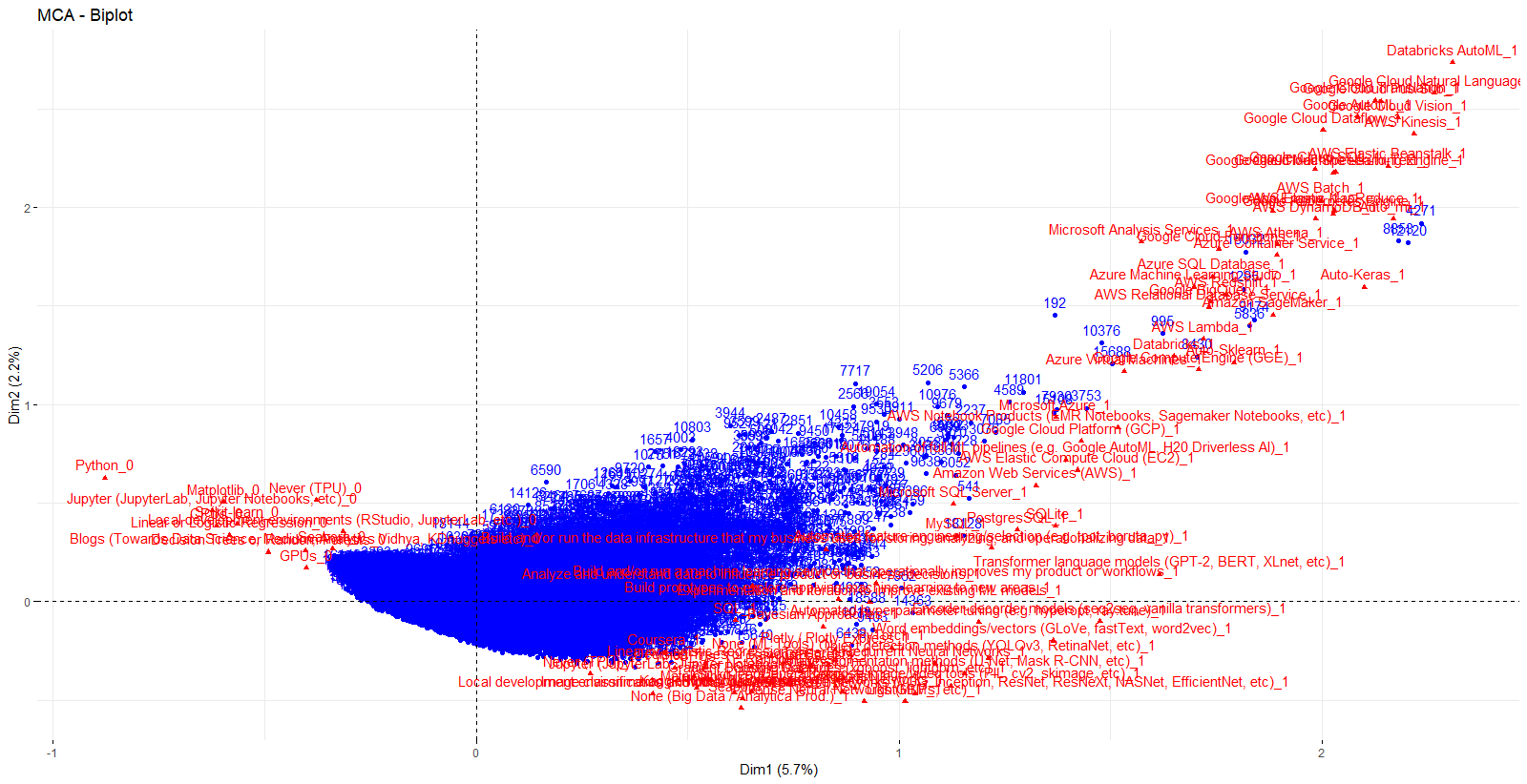
X = 25

fviz\_mca\_biplot(Mutli.MCA, select.var = list(contrib = 25) )



X = 100

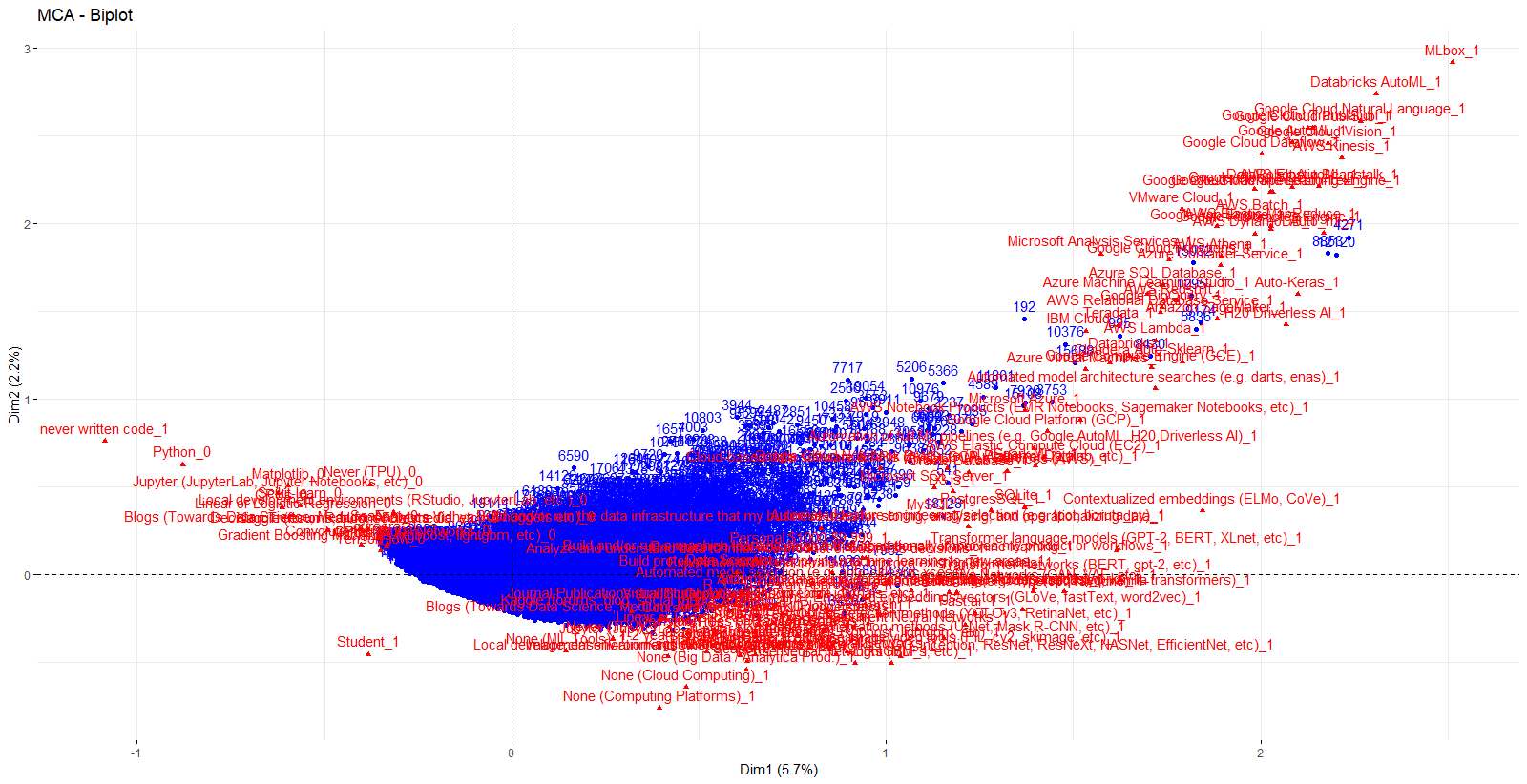
fviz\_mca\_biplot(Mutli.MCA, select.var = list(contrib = 100) )



None of the top 100 contributing variables are found in the areas marked above.

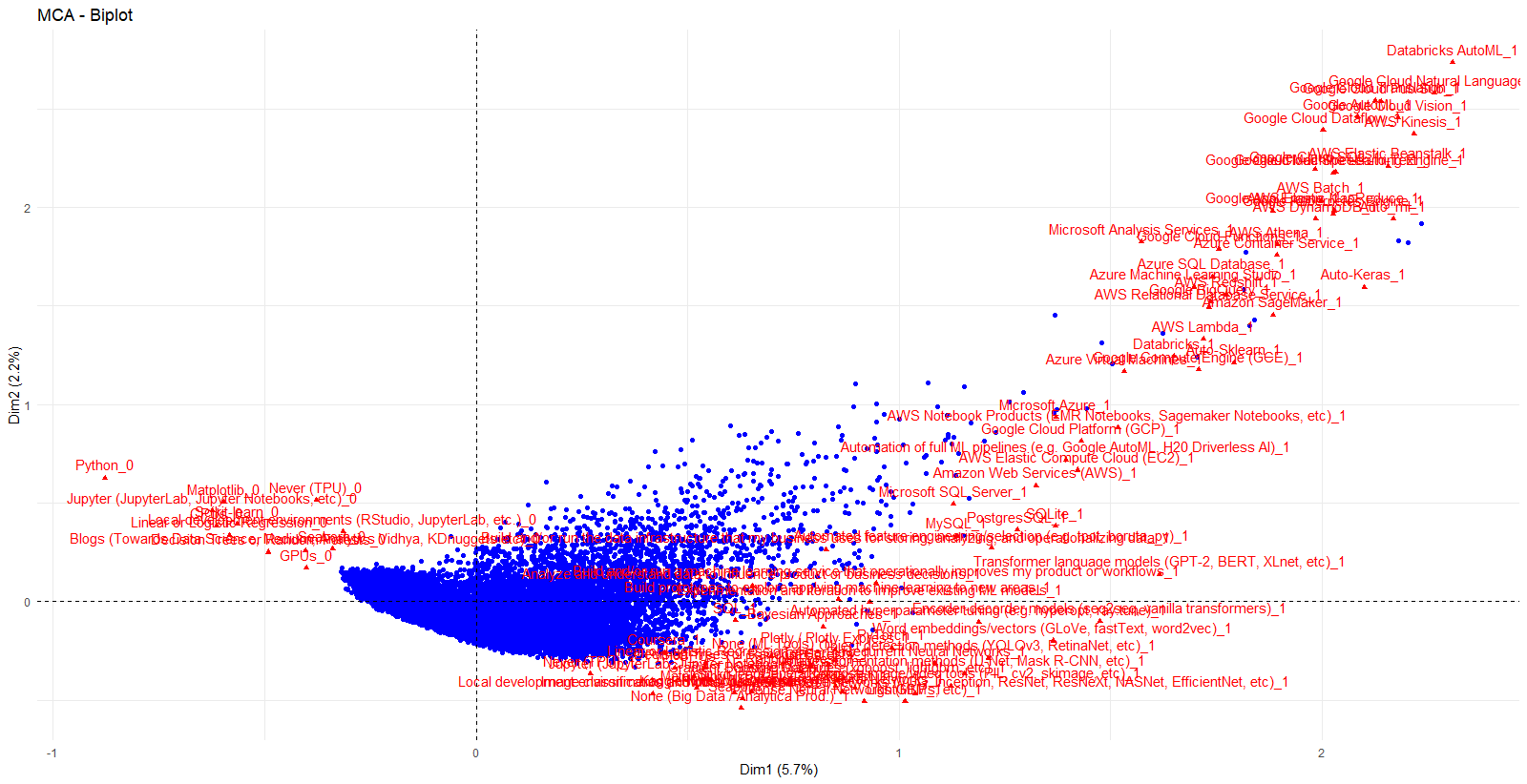
X = 150

fviz\_mca\_biplot(Mutli.MCA, select.var = list(contrib = 100) )



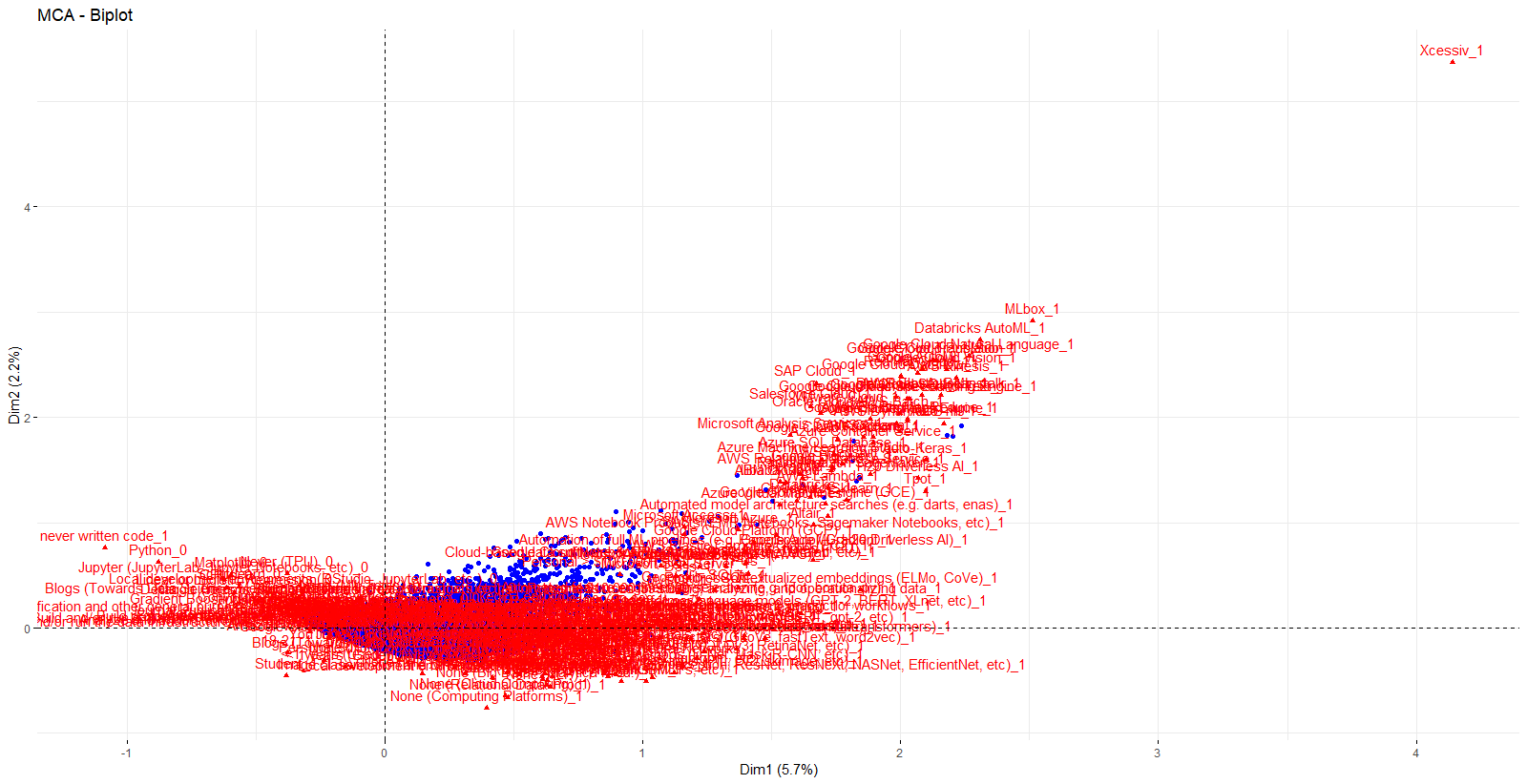
The previous observation still holds mostly true even with the top 150.

For characterizing this lower-dimensional representation, it was decided that going with the top 100 variables will be the most effective way to visualize the data whilst making sure that the project is only talking about aspects of Kaggle users that more meaningfully separate them from one another. The number labels for points was also removed to clean it up a bit.



Focusing on the top 100 contributing variables in 2 dimensions, one can three major groups develop within the quadrants. Starting in the group at the upper right quadrant, you see the smaller group of Kaggle users who tend to shy away from using programming languages directly to work with data, notably their lack of knowledge in Python, Rstudio visualization methods, Jupyter, Blogs, GPU systems and where unfamiliar with TPUs. Looking at the bottom right quadrant, you see the opposite with it being highlighted with people more familiar with neural networks, working with Jupyter, image classification, seaborn trees, and gradient boosting machines. In the quadrant above them, you get similar group that works more with company services like most of the Google packages, Microsoft services, databricks and AWS systems. This group is also by far the most distinct compared to their peers, with many seeming to only work within those environments. I would guess that this distinction likely separates those working for larger companies or institutions and those that work on a more individual level.

To view what influences the third quadrant, I will have to look at lower contributing variables. It is not until we look at up to the 250 highest where we start to see much of anything.

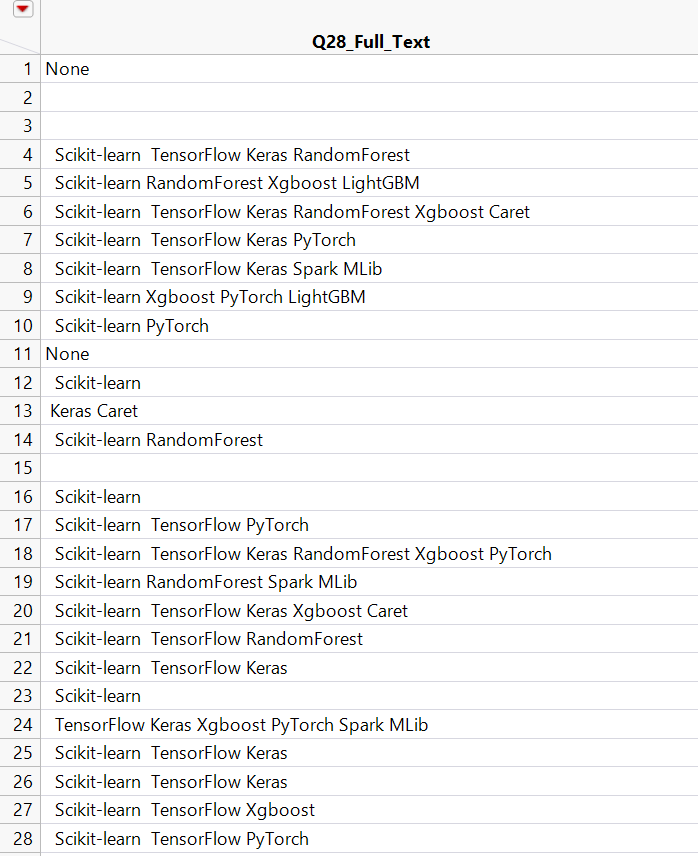


While difficult to read in Word the major pullers in this quadrant tend to be students and blog users, although based on the sheer lack of distance from the center of this group along with its lack of high contributors, it is likely more a combination of being between the top-right and bottom-left that puts people in this quadrant rather than anything overtly powerful other then perhaps the student indicator.

Overall, while the multiple-choice responses from the survey did not create the most beautiful of plots it does show a distinction among those who work with big data, showing a separation between more DSCI centric users, those who work within larger organization and those who tend to shy away from working with programming languages directly. However, in 2 dimensions we still find a vast number of users that are poorly represented, with most users being found in massive clump in the middle. Furthermore, it only can explain an abysmal 7% of the data this way. Perhaps this is indicative of the vast diversity in how one can approach, as well as the vast of problems wished to be solve that Kaggle users (and to an extent, those who work with data) that exist within such a group. From Python to Excel, there are no shortage of tools one can use to set themselves apart others even as patterns begin to form.

Method #2 Association Rules on Text Responses

Almost every question in this survey had some kind of optional text response, unfortunately most were underutilized by the respondents to the point where analysis would not be worth it, or would be at odds with results from the question it was tied to. Furthermore, for some questions the other text response was shuffled into a separate table, meaning that there is no way to tie it back to those who simply picked from a preselection of choices. However, for questions 14 and 28, respondents chose from a selection of text boxes that represented individual words pertaining to software and machine learning frameworks they use respectively in addition to having a free text response that was used more significantly. By concatenating all of the columns quickly in JMP, it can very easily make itself ready for finding association rules for both of those categories in R. A sample of what it looked like coming out of JMP is shown below.



Question 14 asked:

“What is the primary tool that you use at work or school to analyze data?”

Top 20 responses:

q14corpus = CleanCorpus(q14$ï..Q14\_full\_text)

q14tdm = TermDocumentMatrix(q14corpus)

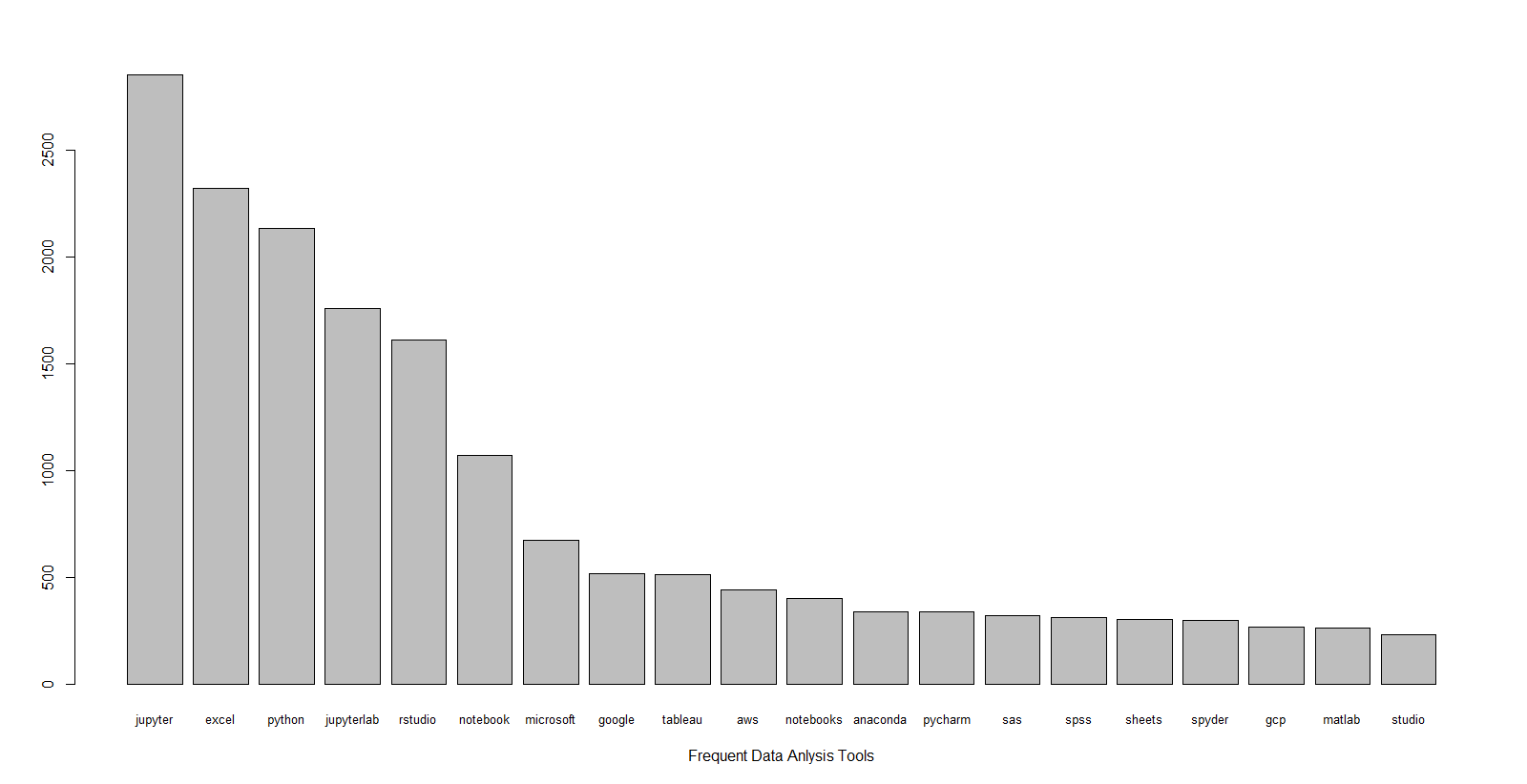
q14.mat = as.matrix(q14tdm)

q14.wfreq = sort(rowSums(q14.mat), decreasing = T)

q14.wfreq[1:20]

|  |
| --- |
| jupyter excel python jupyterlab rstudio notebook microsoft google tableau aws notebooks  2852 2324 2135 1758 1611 1070 673 517 515 441 401  anaconda pycharm sas spss sheets spyder gcp matlab studio  341 339 323 312 305 300 267 264 232 |
|  |
| |  | | --- | |  | |

barplot(q14.wfreq[1:20], xlab = "Frequent Data Anlysis Tools", cex.names = 0.75)



It appears that I have made an error, as it treats things such as “google” “sheets” and “Microsoft” “azure” as sperate tools. To remedy this, consider “Google” and “Microsoft” to be stopwords in the CleanCorpus function. From here on out, Sheets refers to Google Sheets and Azure means Microsoft Azure. Finally, I went back and removed missing rows as a result of individuals never getting this question in accordance with the way the survey was distributed to not otherwise overly penalize support values.

CleanCorpus = function(tweets) {

tweetCorpus = Corpus(VectorSource(tweets))

tweetCorpus = tm\_map(tweetCorpus,content\_transformer(tolower))

tweetCorpus = tm\_map(tweetCorpus,removePunctuation)

tweetCorpus = tm\_map(tweetCorpus,removeNumbers)

tweetCorpus = tm\_map(tweetCorpus,removeWords,stopwords("english"))

tweetCorpus = tm\_map(tweetCorpus, removeWords, c("google","microsoft"))

return(tweetCorpus)

}

q14corpus = CleanCorpus(q14$ï..Q14\_full\_text)

q14tdm = TermDocumentMatrix(q14corpus)

q14.mat = as.matrix(q14tdm)

q14.wfreq = sort(rowSums(q14.mat), decreasing = T)

jupyter excel python jupyterlab rstudio notebook tableau aws notebooks anaconda pycharm

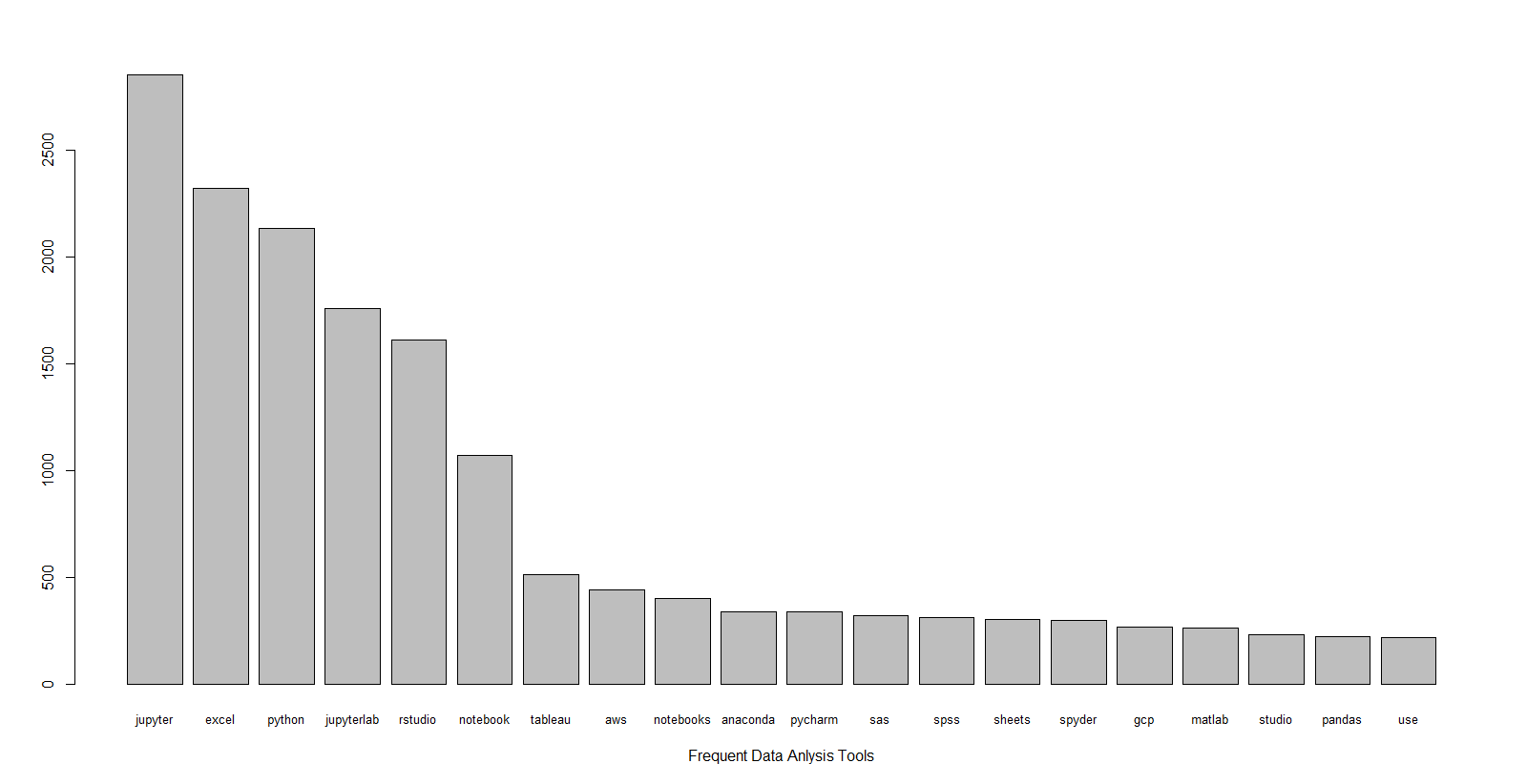
2852 2324 2135 1758 1611 1070 515 441 401 341 339

sas spss sheets spyder gcp matlab studio pandas use

323 312 305 300 267 264 232 224 219

q14.wfreq[1:20]

barplot(q14.wfreq[1:20], xlab = "Frequent Data Anlysis Tools", cex.names = 0.8)

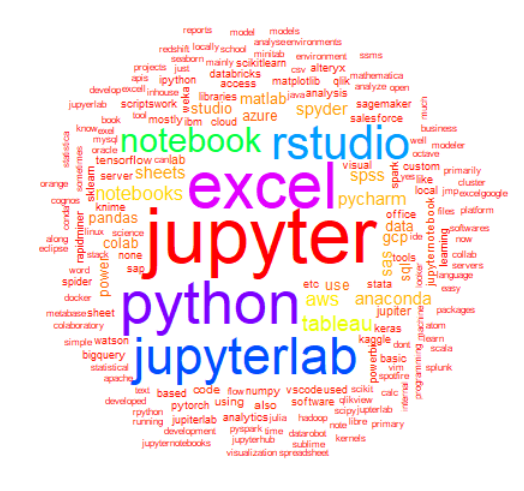


Now with that taken care of, we can create a few word clouds to try and visualize data in a more exciting manner.

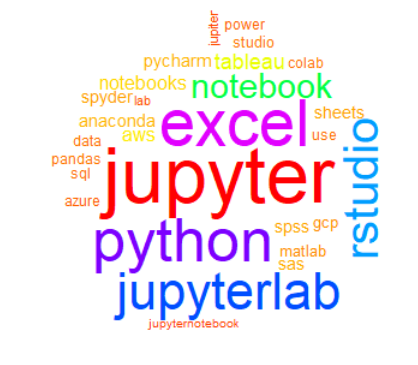
wordcloud(words = names(q14.wfreq), freq = q14.wfreq, random.order = F, col = rainbow(1000), min.freq = 50)



wordcloud(words = names(q14.wfreq), freq = q14.wfreq, random.order = F, col = rainbow(1000), min.freq = 10)



wordcloud(words = names(q14.wfreq), freq = q14.wfreq, random.order = F, col = rainbow(1000), min.freq = 100)



The results, while noteworthy, are not terribly surprising. Python and Rstudio makes sense being in the top given that they have both been around for a while are community-library driven function like R. Jupyter is a go to for cooperative programming, so it makes sense for it to be the top result. Excel, unfortunately, is ubiquitous and the default option for almost every institution in the western world. Finally, notebook and tableau are both powerful constructs for their particular niches that are easy to use. Starting with tableau though, the drop-off in frequency is a mess, with the next series of runners-up consisting mostly of suites or packages that work well in tandem with the tools described earlier.

Word clouds, as profound as they can be, are at the end of the day merely glorified bar plots. With association rules we can find more specific insights, and with this question, figure out if there are any sorts of patterns among the tools Kaggle users utilize.

> findAssocs(q14tdm, "jupyter", 0.1)

$`jupyter`

notebook notebooks lab

0.51 0.29 0.20

While Jupyter was an option in the checkboxes, many users would go on to specify in text whether they used just notebooks or lab. Notebook and notebooks were not combined as one though, as sometimes it seemed as though people were referring to Google notebook, or just simply responded with “notebook” in the other text field having picked both or none of the others. Still, one can take away that while a lot of Jupyter users use both, there is still a significant group of Kaggle users who don’t

> findAssocs(q14tdm, "excel", 0.1)

$`excel`

sheets

0.1

Both Excel and Google sheets are a lot of peoples “go to” for simple data analysis, so it should not come as to surprising that we find some association here. Although I suspect the usage is mostly to convert one to another so one can work in there preferred environment.

> findAssocs(q14tdm, "sheets", 0.1)

$`sheets`

excel excelgoogle

0.1 0.1

The pattern is certainly reversible. I’m not sure what “excelgoogle” is, but Ii is believed to be someone attempting to say “google sheets”.

> findAssocs(q14tdm, "python", 0.1)

$`python`

libraries

0.14

Often it seems as though python users seemed to use it for a variety of tasks. Often times they would specify in the custom text field what libraries they used in addition to just “python”.

> findAssocs(q14tdm, "tableau", 0.05)

$`tableau`

alteryx histogramsbargraphs salesforce power dashboard experience

0.10 0.09 0.07 0.06 0.06 0.06

Tableau is interesting because while it lacks associations as strong as others near the top, its sole dedication to visualizations has left it wider variety of correlations with other specialized software.

On the whole, just looking at singular word associations we don’t find anything major. Surprisingly, at least according to this survey, most individuals primarily focus on one or two tools. This may be because when working for larger companies, people tend to get more specialized when working in larger teams, leading to them mastering only a couple of key utilities. Nevertheless, I will still use apriori to try to find other interesting rules, though we may have to loosen our support requirements a bit more than usual.

> q14.arules = apriori(q14.trans, parameter = list(supp = 0.001, conf = 0.5, target = "rules"),

+ appearance = list(none = c("jupyter","learning"))

+ )

Apriori

Parameter specification:

confidence minval smax arem aval originalSupport maxtime support minlen maxlen target ext

0.5 0.1 1 none FALSE TRUE 5 0.001 1 10 rules FALSE

Algorithmic control:

filter tree heap memopt load sort verbose

0.1 TRUE TRUE FALSE TRUE 2 TRUE

Absolute minimum support count: 15

set item appearances ...[2 item(s)] done [0.00s].

set transactions ...[2200 item(s), 15673 transaction(s)] done [0.00s].

sorting and recoding items ... [124 item(s)] done [0.00s].

creating transaction tree ... done [0.00s].

checking subsets of size 1 2 3 done [0.00s].

writing ... [19 rule(s)] done [0.00s].

creating S4 object ... done [0.00s].

> inspect(q14.arules)

lhs rhs support confidence lift count

[1] {access} => {excel} 0.001403688 0.7333333 4.945582 22

[2] {sagemaker} => {aws} 0.001212276 0.5588235 19.905548 19

[3] {watson} => {studio} 0.001403688 0.6470588 43.712728 22

[4] {sklearn} => {pandas} 0.001212276 0.6129032 42.884073 19

[5] {scripts} => {python} 0.001403688 0.7857143 5.778742 22

[6] {analyze} => {data} 0.001339884 0.9130435 72.640256 21

[7] {keras} => {python} 0.001467492 0.6052632 4.451567 23

[8] {science} => {data} 0.001467492 0.9583333 76.243443 23

[9] {server} => {sql} 0.001850316 0.5471698 40.837107 29

[10] {matplotlib} => {pandas} 0.002105532 0.6600000 46.179375 33

[11] {visual} => {code} 0.001658904 0.5000000 92.194118 26

[12] {visual} => {studio} 0.002998788 0.9038462 61.060262 47

[13] {libraries} => {python} 0.003509220 0.8461538 6.223261 55

[14] {numpy} => {pandas} 0.003126396 0.8032787 56.204406 49

[15] {analysis} => {data} 0.001850316 0.5087719 40.477068 29

[16] {code,visual} => {studio} 0.001531296 0.9230769 62.359416 24

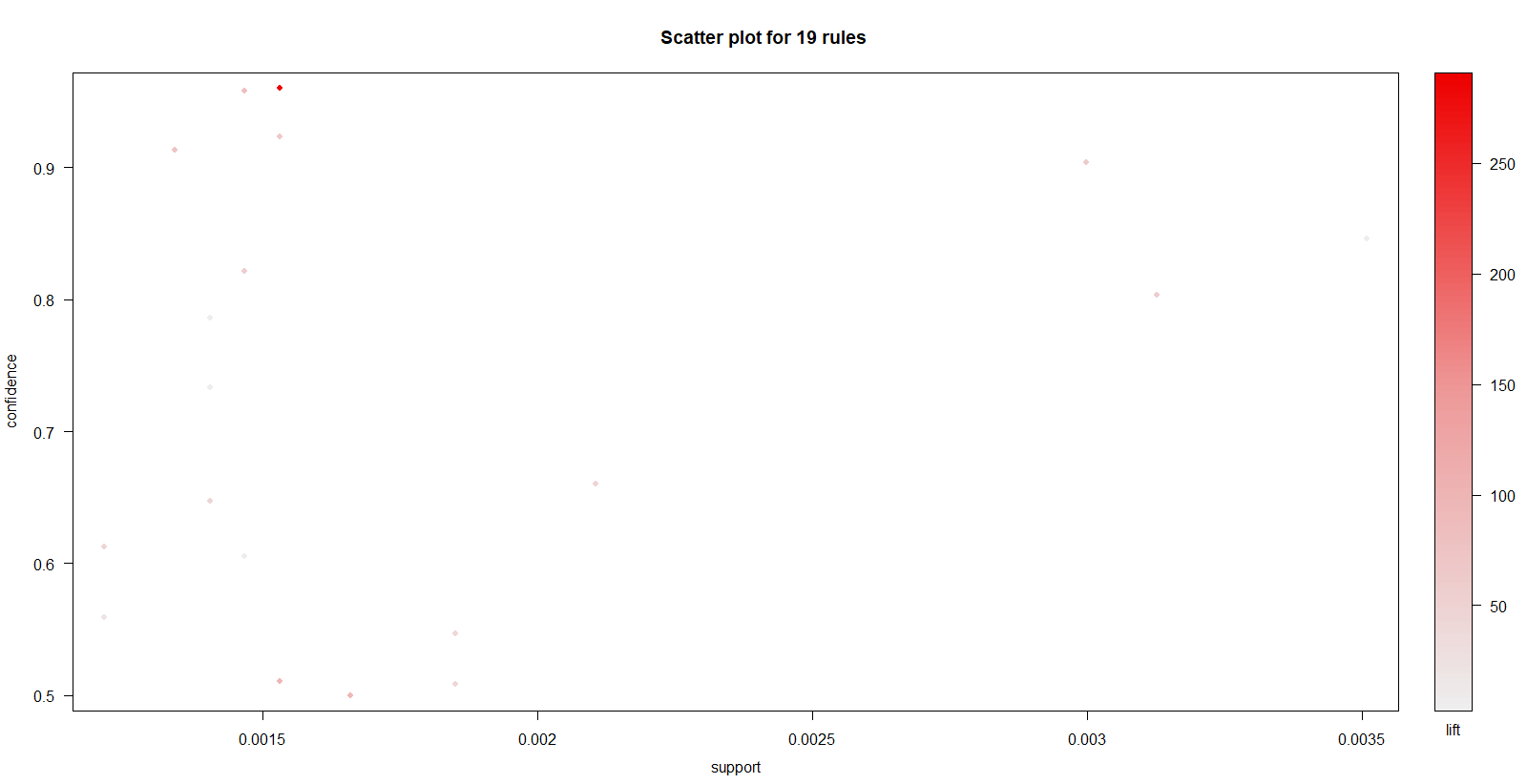
[17] {studio,visual} => {code} 0.001531296 0.5106383 94.155695 24

[18] {code,studio} => {visual} 0.001531296 0.9600000 289.347692 24

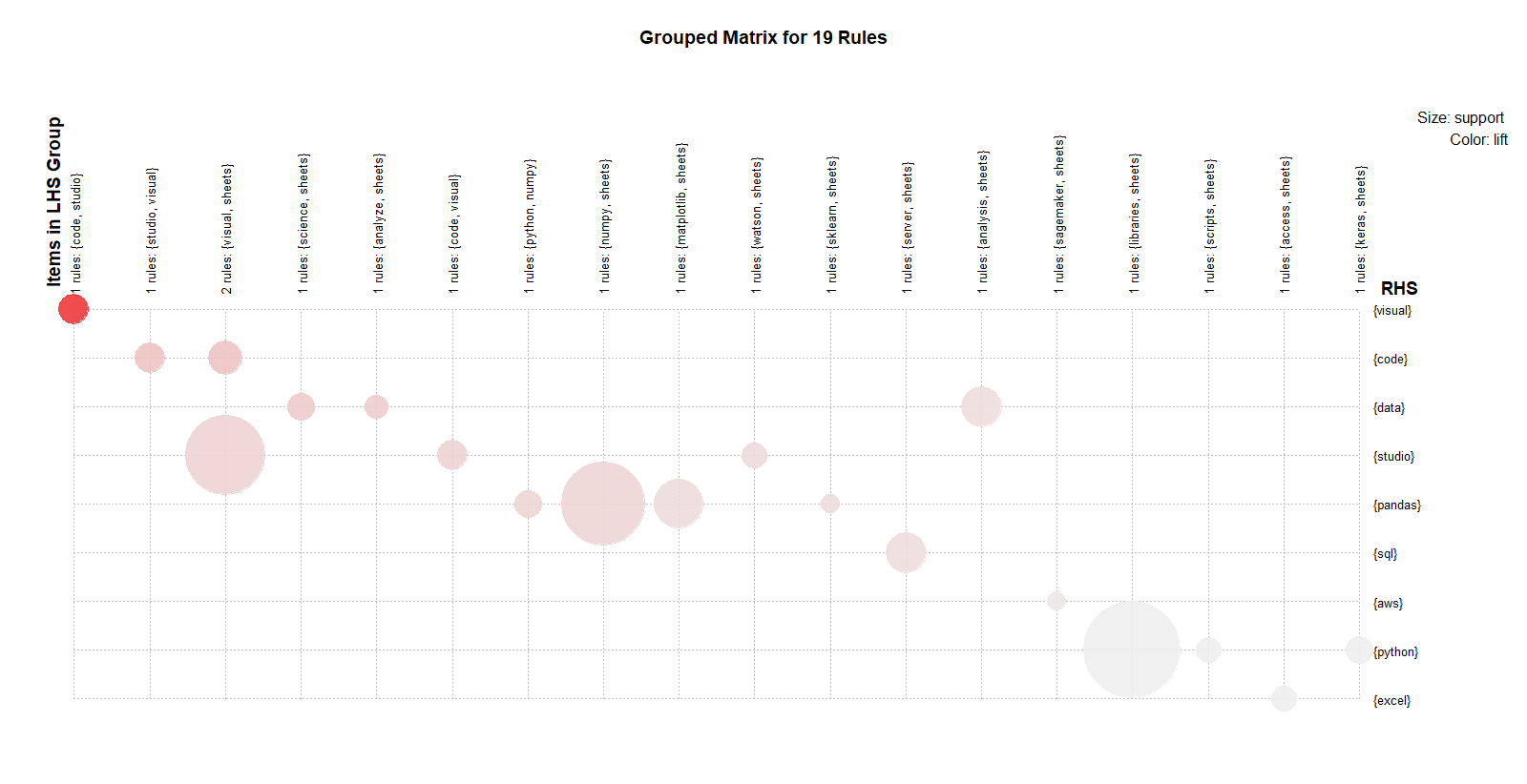
[19] {python,numpy} => {pandas} 0.001467492 0.8214286 57.474330 23

This certainly is scraping the barrel in terms of support to find interesting rules, although given what was found above that isn’t to surprising. It should be noted that the best rules to come out of this seem to be from those who use more specialized software like access, watson, and sklearn, likely as their narrow focus forcing their users to be more likely to also work with something else. In fact, some of these rules have incredibly great lift values even with counts upwards past 40, which isn’t nothing. This point is best illustrated in the scatterplot and correlation plot below.

plot(q14.arules)

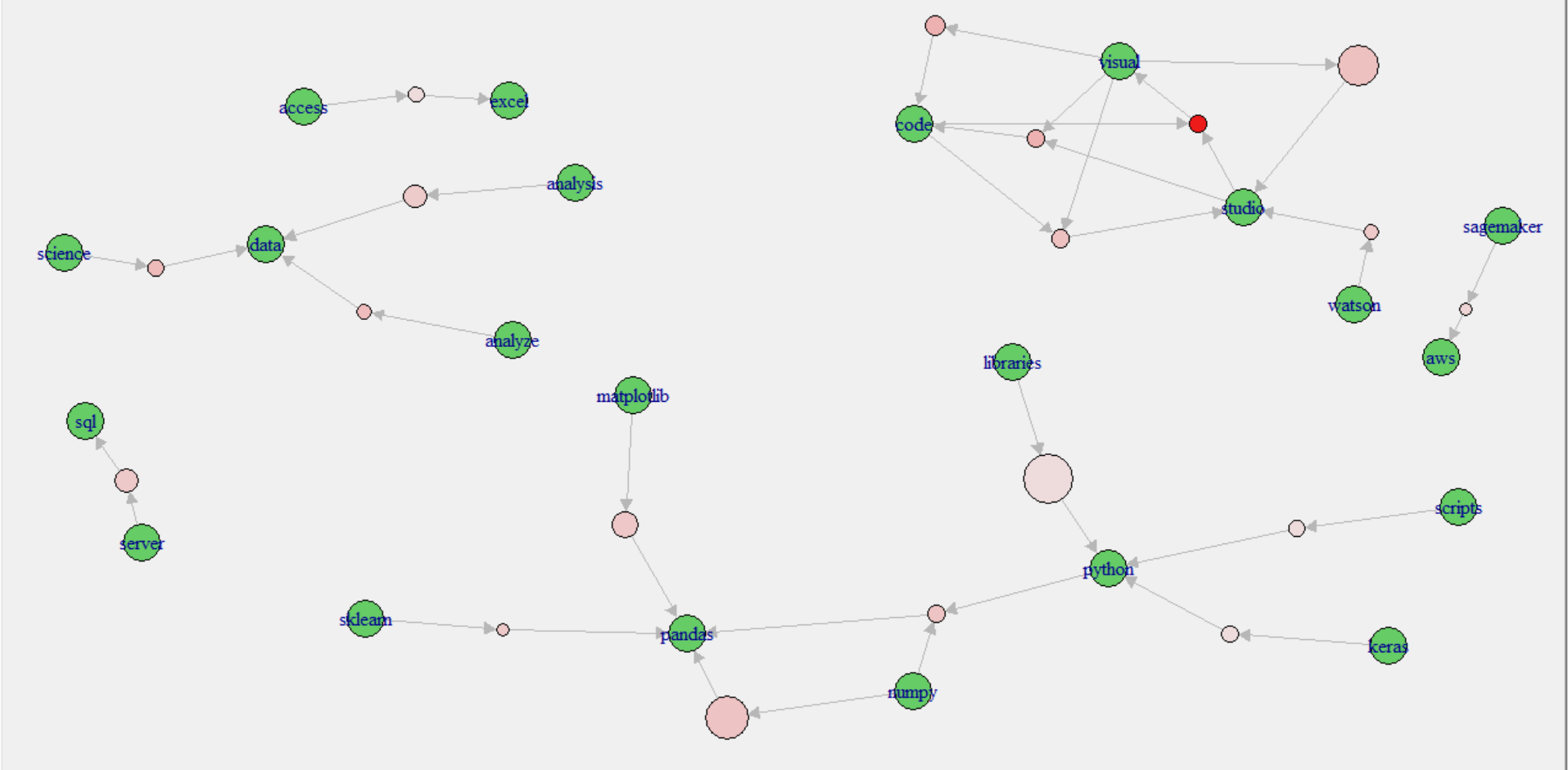


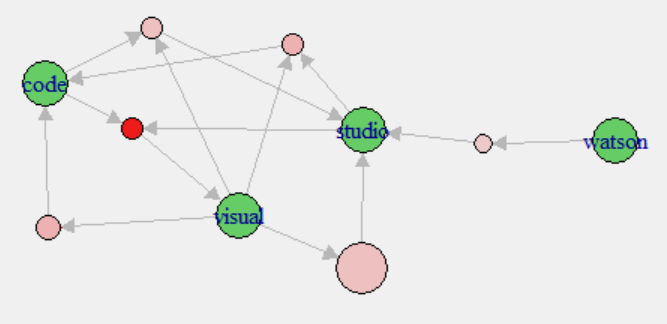
plot(q14.arules, method = "grouped")



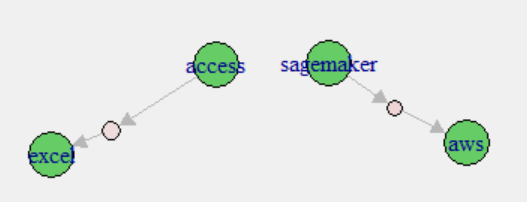
plot(q14.arules, method = "graph", engine = "interactive")

ruleExplorer was providing many issues, so It was decided instead to just make plots using engine = interactive.

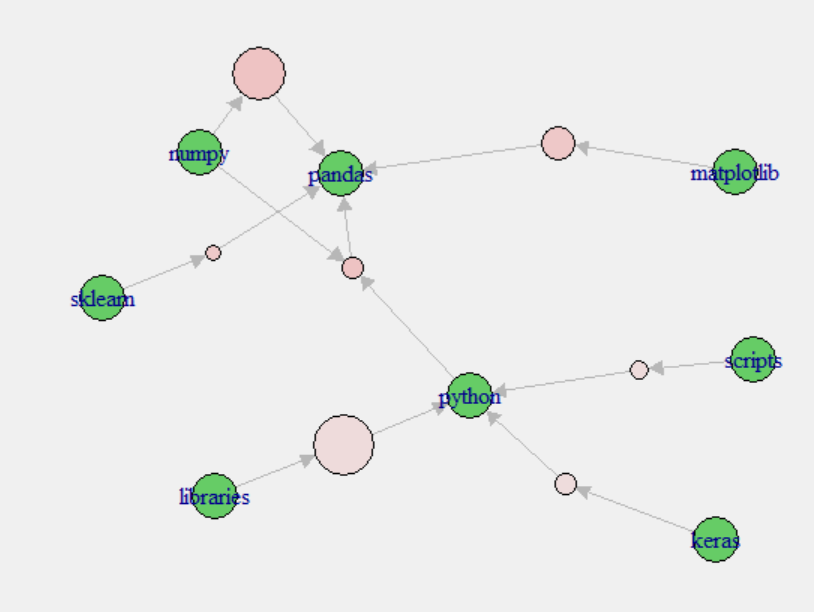




Here one can see some patterns among users of less all-encompassing programs being used in tandem to cover each other’s weaknesses.



A similar idea is seen here, but only with two specific pairing types. Access with Excel makes sense given that they’re both MS products. Same with Sagemaker and AWS, as those are the Amazon equivalent.



Probably the most interesting group of associations is found here, as it shows Pythons and Pandas to be epicenters that are used by respondents in addition to other software like Google Scripts, Keras, sklearn, numpy and matplotlib. This seems reasonable given that both Python and Pandas function like R in the sense that while you can do almost anything you need to within them, sometimes more focused software can get you better results more easily. Finally, libraries were left in even though it isn’t software because if someone wanted to see what language people more associate using external libraries vs. base with they could, and the result ended up being Python by a significant margin.

However, it should also be kept in mind that with everything that we have learned with these rules, the overall support is low, so we should be hesitant to make any sweeping claims about data Kaggle Users. Although, there may be enough here to perhaps make an argument that any future studies like this done by Kaggle should include some more extensive text sections on this subject, as patterns would more likely be found.

Question 28 asks:

“Which of the following machine learning frameworks do you use on a regular basis?”

q28 <- read.csv("Q28\_full\_textcsv\_null\_removed.csv")

CleanCorpus = function(tweets) {

tweetCorpus = Corpus(VectorSource(tweets))

tweetCorpus = tm\_map(tweetCorpus,content\_transformer(tolower))

tweetCorpus = tm\_map(tweetCorpus,removePunctuation)

tweetCorpus = tm\_map(tweetCorpus,removeNumbers)

tweetCorpus = tm\_map(tweetCorpus,removeWords,stopwords("english"))

tweetCorpus = tm\_map(tweetCorpus, removeWords, c("google","microsoft"))

return(tweetCorpus)

}

q28corpus = CleanCorpus(q28$Q28\_Full\_Text)

q28tdm = TermDocumentMatrix(q28corpus)

q28.mat = as.matrix(q28tdm)

q28.wfreq = sort(rowSums(q28.mat), decreasing = T)

q28.wfreq[1:20]

scikitlearn tensorflow keras randomforest xgboost pytorch lightgbm none

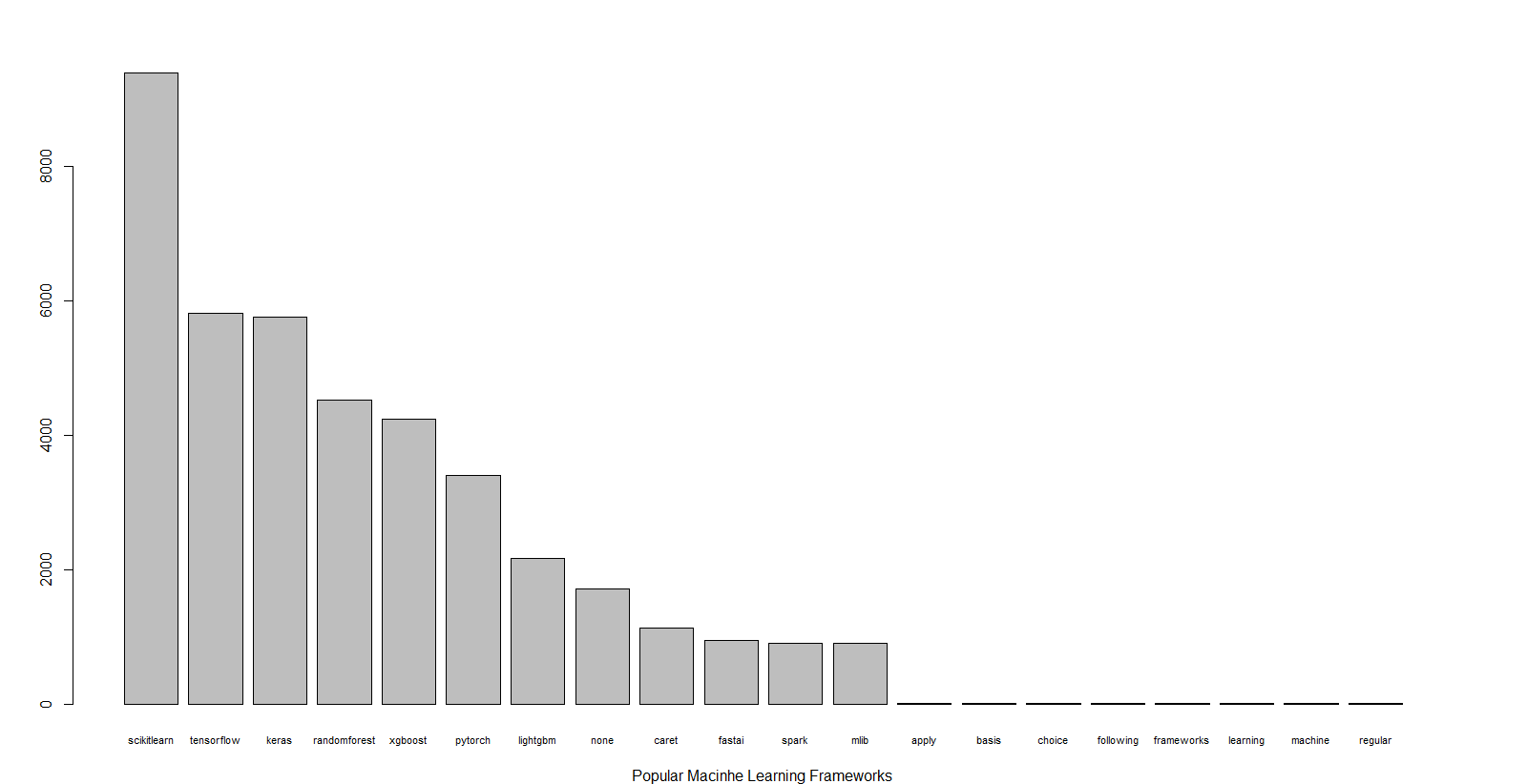
9390 5822 5756 4524 4243 3412 2166 1721

caret fastai spark mlib apply basis choice following

1139 949 911 910 11 11 11 11

frameworks learning machine regular

11 11 11 11



Now I must admit that machine learning frameworks are not something I would call myself overtly knowledgeable about, but these answers seem distinct and recognizable enough. While after mlib we quickly find ourselves left with negligible support, the high count for other the rest of the frameworks will likely allow us to get better association rules than the previous question. Like before, word clouds were constructed next as an alternative visualization.

wordcloud(words = names(q28.wfreq), freq = q28.wfreq, random.order = F, col = rainbow(100), min.freq = 50)



wordcloud(words = names(q28.wfreq), freq = q28.wfreq, random.order = F, col = rainbow(1000))



wordcloud(words = names(q28.wfreq), freq = q28.wfreq, random.order = T, col = rainbow(20), min.freq = 20)



Next, some quick association peeking on words can give us a rough idea of what’s to come.

> findAssocs(q28tdm, "scikitlearn", 0.1)

$`scikitlearn`

keras xgboost randomforest tensorflow lightgbm pytorch

0.27 0.27 0.22 0.21 0.20 0.11

>

> findAssocs(q28tdm, "tensorflow", 0.1)

$`tensorflow`

keras pytorch scikitlearn xgboost

0.57 0.22 0.21 0.10

>

> findAssocs(q28tdm, "keras", 0.1)

$`keras`

tensorflow scikitlearn xgboost pytorch lightgbm randomforest fastai

0.57 0.27 0.20 0.20 0.16 0.10 0.10

>

> findAssocs(q28tdm, "randomforest", 0.1)

$`randomforest`

xgboost lightgbm caret scikitlearn mlib spark keras

0.43 0.26 0.26 0.22 0.11 0.10 0.10

>

> findAssocs(q28tdm, "xgboost", 0.1)

$`xgboost`

lightgbm randomforest scikitlearn keras caret tensorflow

0.51 0.43 0.27 0.20 0.17 0.10

These are much better across the board compared to question 14. While all of these are likely worth looking at with apriori, the relationship keras-tensorflow should definitely be one to follow carefully.

temp = t(q28.mat)

temp[temp>1] = 1

q28.trans = as(temp,"transactions")

q28.arules = apriori(q28.trans, parameter = list(supp = 0.2, conf = 0.5, target = "rules", maxlen = 3, minlen = 1)

, appearance = list(none = c(" ")

))

q28.arules = sort(q28.arules, by="confidence", decreasing=TRUE)

inspect(q28.arules)

lhs rhs support confidence lift count

[1] {xgboost} => {scikitlearn} 0.2677039 0.8677822 1.271084 3682

[2] {keras,tensorflow} => {scikitlearn} 0.2679221 0.8481013 1.242256 3685

[3] {randomforest} => {scikitlearn} 0.2735204 0.8315650 1.218035 3762

[4] {keras} => {scikitlearn} 0.3476807 0.8307853 1.216892 4782

[5] {tensorflow} => {scikitlearn} 0.3375745 0.7974923 1.168127 4643

[6] {scikitlearn,tensorflow} => {keras} 0.2679221 0.7936679 1.896475 3685

[7] {keras,scikitlearn} => {tensorflow} 0.2679221 0.7705981 1.820475 3685

[8] {keras} => {tensorflow} 0.3159081 0.7548645 1.783306 4345

[9] {tensorflow} => {keras} 0.3159081 0.7463071 1.783306 4345

[10] {scikitlearn} => {keras} 0.3476807 0.5092652 1.216892 4782

Conveniently, exactly 10 rules were returned through these parameters. Scikitlearn is shown as a dominating force within used machine learning frameworks, as using most other major frameworks pushes you towards that. One can also notice that the keras-tensorflow relationship is present, but by far not the strongest, although it is after all of the rules implying scikitlearn.

If we loosen our parameters we can get many more rules.

q28.arules = apriori(q28.trans, parameter = list(supp = 0.1, conf = 0.5, target = "rules", maxlen = 3, minlen = 1)

)

q28.arules = sort(q28.arules, by="confidence", decreasing=TRUE)

inspect(q28.arules)

lhs rhs support confidence lift count

[1] {tensorflow,xgboost} => {scikitlearn} 0.1414134 0.9213643 1.349568 1945

[2] {randomforest,tensorflow} => {scikitlearn} 0.1374146 0.9139265 1.338674 1890

[3] {keras,randomforest} => {scikitlearn} 0.1470845 0.9092135 1.331770 2023

[4] {keras,xgboost} => {scikitlearn} 0.1592264 0.9090909 1.331591 2190

[5] {lightgbm,xgboost} => {scikitlearn} 0.1217100 0.9083017 1.330435 1674

[6] {lightgbm} => {scikitlearn} 0.1408318 0.8942752 1.309889 1937

[7] {randomforest,xgboost} => {scikitlearn} 0.1704959 0.8746736 1.281178 2345

[8] {xgboost} => {scikitlearn} 0.2677039 0.8677822 1.271084 3682

[9] {scikitlearn,lightgbm} => {xgboost} 0.1217100 0.8642230 2.801443 1674

[10] {keras,pytorch} => {scikitlearn} 0.1246183 0.8561439 1.254036 1714

[11] {lightgbm} => {xgboost} 0.1339974 0.8508772 2.758182 1843

[12] {tensorflow,xgboost} => {keras} 0.1305075 0.8503079 2.031816 1795

[13] {keras,tensorflow} => {scikitlearn} 0.2679221 0.8481013 1.242256 3685

[14] {randomforest} => {scikitlearn} 0.2735204 0.8315650 1.218035 3762

[15] {keras} => {scikitlearn} 0.3476807 0.8307853 1.216892 4782

[16] {tensorflow,pytorch} => {scikitlearn} 0.1236731 0.8170029 1.196705 1701

[17] {randomforest,tensorflow} => {keras} 0.1217100 0.8094778 1.934252 1674

[18] {keras,pytorch} => {tensorflow} 0.1164025 0.7997003 1.889227 1601

[19] {tensorflow} => {scikitlearn} 0.3375745 0.7974923 1.168127 4643

[20] {scikitlearn,tensorflow} => {keras} 0.2679221 0.7936679 1.896475 3685

[21] {pytorch} => {scikitlearn} 0.1912898 0.7711020 1.129471 2631

[22] {keras,scikitlearn} => {tensorflow} 0.2679221 0.7705981 1.820475 3685

[23] {tensorflow,pytorch} => {keras} 0.1164025 0.7689721 1.837464 1601

[24] {keras} => {tensorflow} 0.3159081 0.7548645 1.783306 4345

[25] {keras,randomforest} => {tensorflow} 0.1217100 0.7523596 1.777388 1674

[26] {tensorflow} => {keras} 0.3159081 0.7463071 1.783306 4345

[27] {keras,xgboost} => {tensorflow} 0.1305075 0.7451225 1.760291 1795

[28] {keras,randomforest} => {xgboost} 0.1108041 0.6849438 2.220296 1524

[29] {scikitlearn,pytorch} => {keras} 0.1246183 0.6514633 1.556676 1714

[30] {scikitlearn,pytorch} => {tensorflow} 0.1236731 0.6465222 1.527356 1701

[31] {scikitlearn,xgboost} => {randomforest} 0.1704959 0.6368821 1.936268 2345

[32] {keras,xgboost} => {randomforest} 0.1108041 0.6326276 1.923333 1524

[33] {xgboost} => {randomforest} 0.1949251 0.6318642 1.921013 2681

[34] {randomforest,scikitlearn} => {xgboost} 0.1704959 0.6233386 2.020599 2345

[35] {pytorch} => {tensorflow} 0.1513741 0.6101993 1.441546 2082

[36] {scikitlearn,xgboost} => {keras} 0.1592264 0.5947854 1.421244 2190

[37] {randomforest} => {xgboost} 0.1949251 0.5926172 1.921013 2681

[38] {pytorch} => {keras} 0.1455577 0.5867526 1.402049 2002

[39] {randomforest,xgboost} => {keras} 0.1108041 0.5684446 1.358302 1524

[40] {xgboost} => {keras} 0.1751490 0.5677587 1.356663 2409

[41] {randomforest,scikitlearn} => {keras} 0.1470845 0.5377459 1.284947 2023

[42] {scikitlearn,xgboost} => {tensorflow} 0.1414134 0.5282455 1.247937 1945

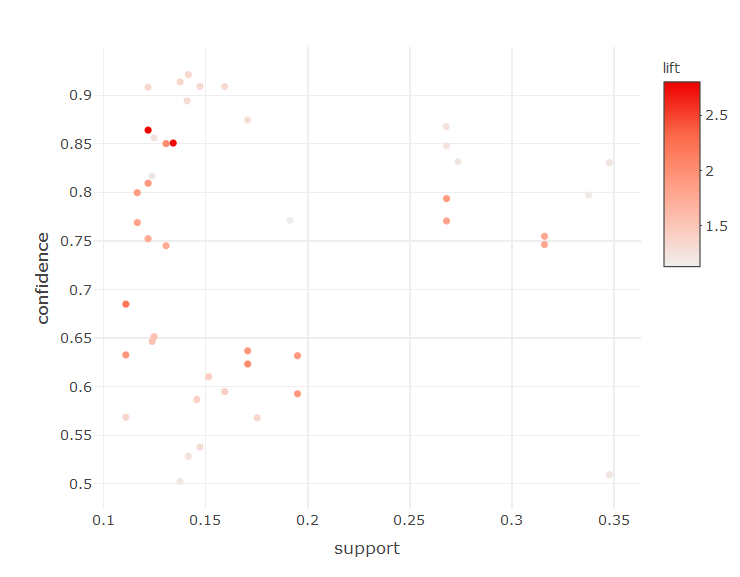
[43] {scikitlearn} => {keras} 0.3476807 0.5092652 1.216892 4782

[44] {randomforest,scikitlearn} => {tensorflow} 0.1374146 0.5023923 1.186861 1890

At 10% support, we now get a much greater variety of rules, often with more favor shown towards rules of length 2, which will certainly make for a more interesting visual. Our lift values are also greater than 1 throughout, which means we have rules of at least theoretical use.

For some odd reason, ruleExplorer was ok to work with this ruleset, so I will use if for the following visualizations.

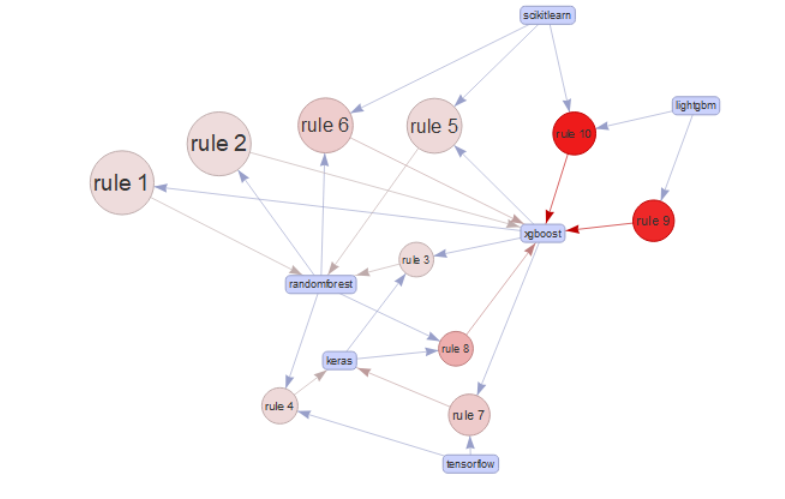
ruleExplorer(q28.arules)



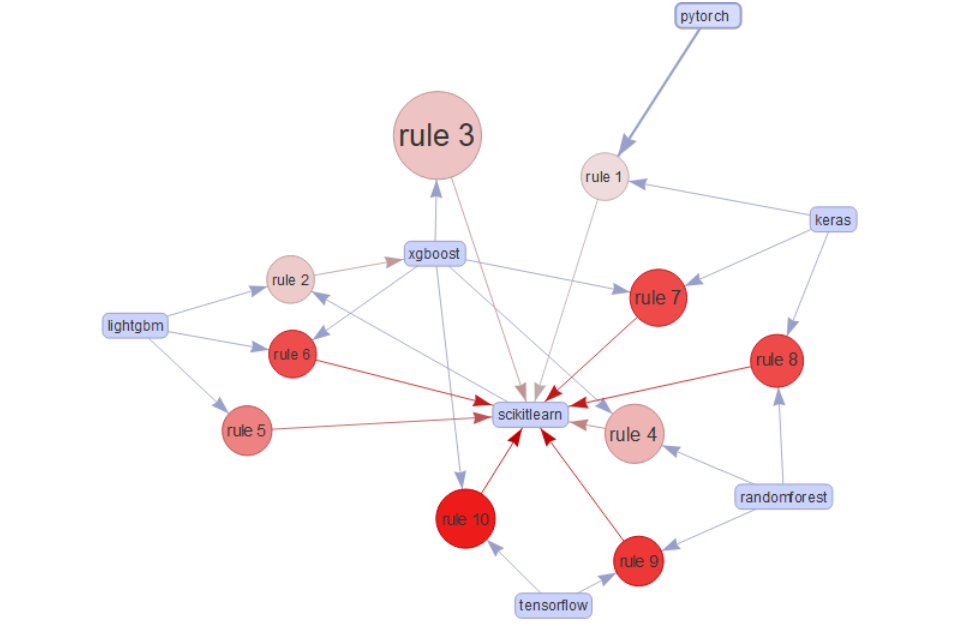
Higher lift rules appear to have lower support, but the highest all-around points still seem to maintain at least a decent amount of lift. I will examine those points in addition to just the top ten by each metric.



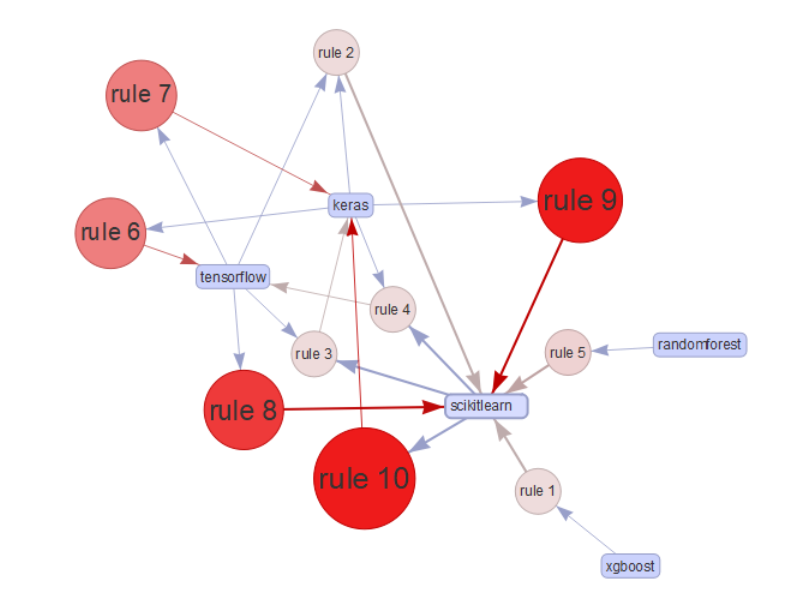
Top ten rules by lift:



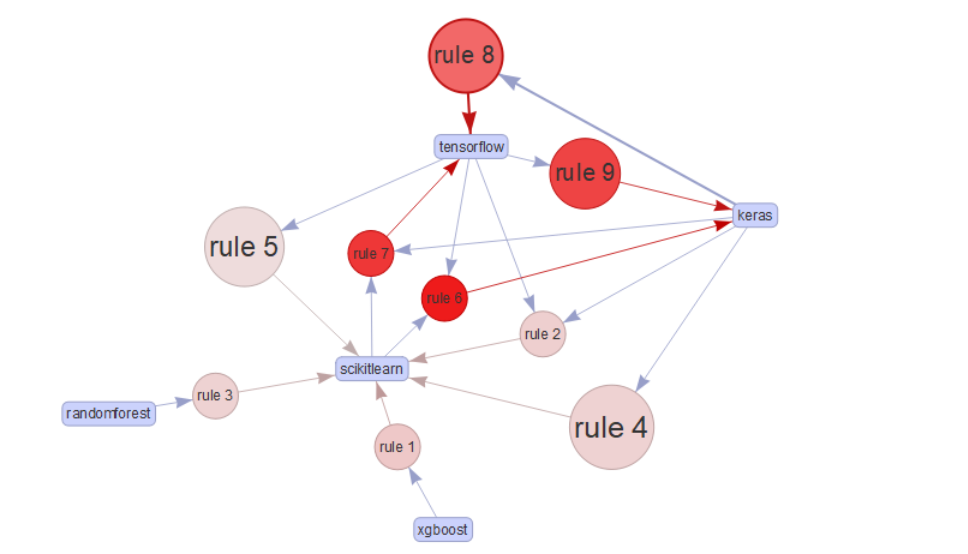
Top ten rules by confidence:



Top ten rules by support:



The “best” rules based on our scatterplot:



In the end, much greater success was had with question 28 versus 14, due to more respondents picking more than one value. While I may not know enough about the differences in the frameworks to confidently make logical conjecture as to why these patterns occur, there is still some ideas to convey. For instance, scikitlearn seems to be both the groundwork where everyone starts, and one that is continued to use as individuals become familiar with others. One also can formulate a top group of mainstays that appear to be the standard for most repsondents, such as the aforementioned scikitlearn in addition to randomforest, tensorflow, xgboost, keras, and pytorch. Finally, we are able to draw connections with high associations that either indicate a similarity between frameworks resulting in individuals picking up both, necessary to use in tandem to work on certain projects leading to people learning both, or that knowledge of certain frameworks include principles of others meaning knowing one means you likely know the other. Keras and Tensorflow likely fall into a relationship this, or something else I’m not aware of. Xgboost and Lightdm also could be, but perhaps in only one direction. Otherwise, I would say a lot of the rules speak for themselves and are best described in context looking at one of the various visualizations above.

Overall success with text associations had their highs and lows, but even at their lowest one can muse that information of at least some substance was obtained, even if more shows what can be improved in the design of the survey.

Method #3 Sentiment Analysis

While most users went through the entirety of the survey without writing a text response, roughly 3,000 of them did, often writing quite a bit of substance on their job, what they recommend aspiring Data Scientists, their preferred (or lack thereof) media sources, their education, and how they would sum up their job title. Answers such as these are found in questions 12, 13, 19, 5 and 9, which were concatenated into a single in JMP so that we can look at the overall sentiment of these more personal responses. Empty rows were then removed. A sample of what the data looks like post conversion is found below.



While this may have a lot of more dry, descriptive material, an individual might be curious as to what results will find in what one would expect to lack any sort of emotional impact. It can’t hurt to check, and furthermore it will be an opportunity to practice a method I otherwise haven’t ever used before.

Starting out, basic analysis of the text was performed like last time with word clouds and frequencies, although this is likely to be more scattered then usual.

sent <- read.csv("Subset of other\_text\_responses 34.csv")

CleanCorpus = function(tweets) {

tweetCorpus = Corpus(VectorSource(tweets))

tweetCorpus = tm\_map(tweetCorpus,content\_transformer(tolower))

tweetCorpus = tm\_map(tweetCorpus,removePunctuation)

tweetCorpus = tm\_map(tweetCorpus,removeNumbers)

tweetCorpus = tm\_map(tweetCorpus,removeWords,stopwords("english"))

return(tweetCorpus)

}

sentcorpus = CleanCorpus(sent$ï..Full\_Other\_Text\_12\_13\_19\_5\_9)

sentTDM = TermDocumentMatrix(sentcorpus)

sent.mat = as.matrix(sentTDM)

sent.wfreq = sort(rowSums(sent.mat), decreasing = T)

sent.wfreq[1:50]

engineer data learning science manager linkedin machine professor youtube medium

280 221 120 115 102 94 77 73 70 70

consultant analyst academy books analytics research architect university course developer

63 59 58 56 55 54 54 51 47 45

mlcourseai courses coursera nptel online business cognitive ibm software facebook

43 42 39 38 34 34 33 33 32 32

director pluralsight simplilearn teacher julia datacamp applied newsletters computer class

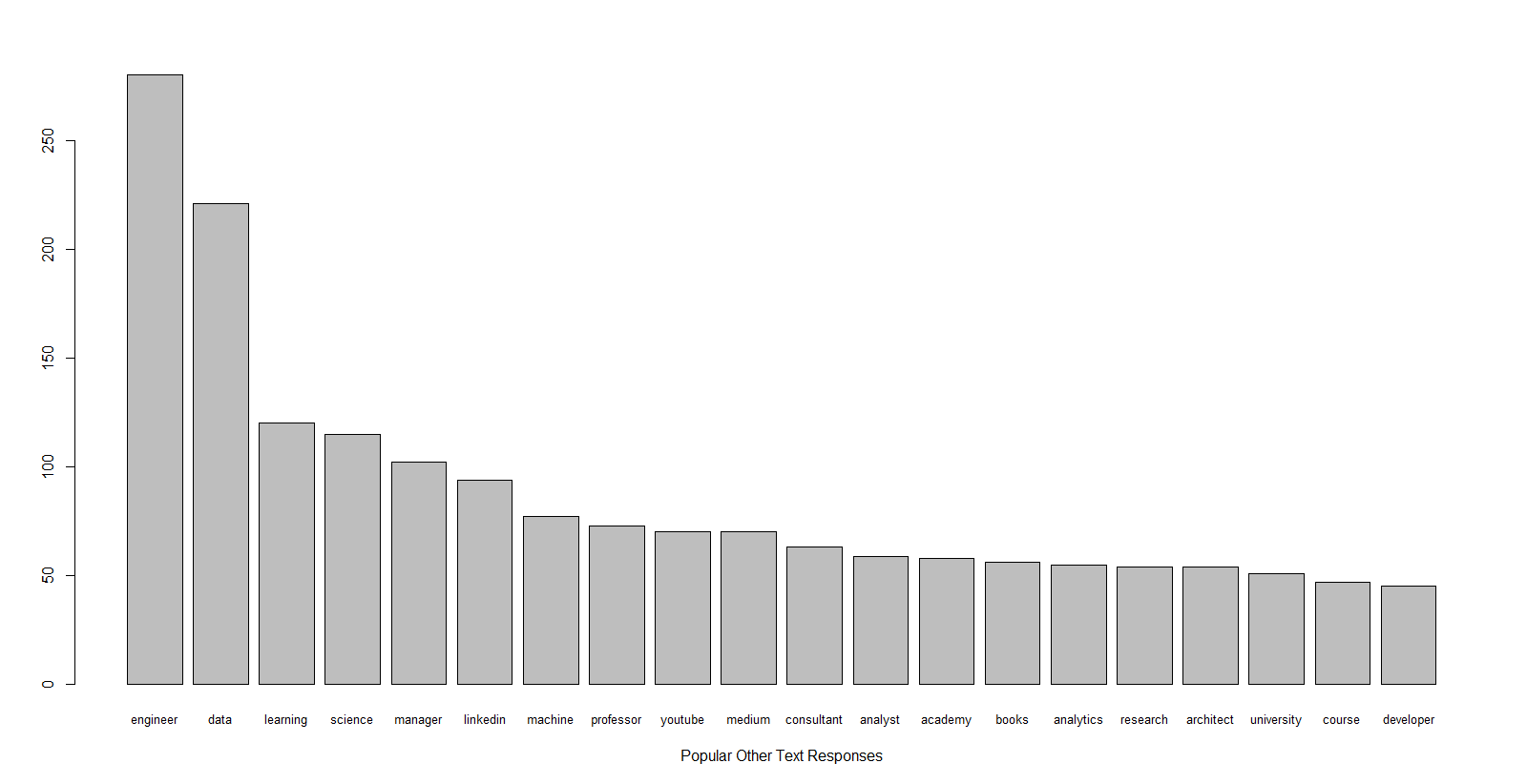
31 31 30 29 29 28 28 27 26 26

stepik python school github lecturer udemy support senior upgrad scientist

24 24 23 23 23 23 23 21 21 20

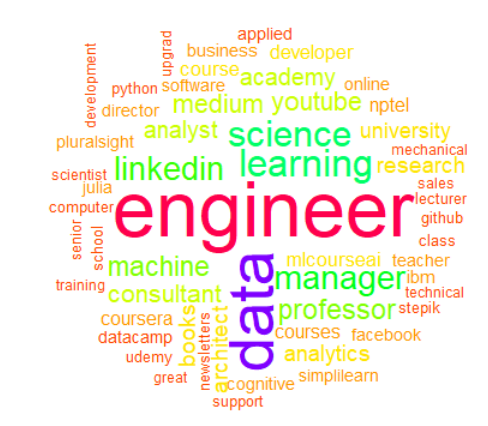
What’s seen is a large variety of similarly themed proper nouns in addition to general terms as expected, however, variations in what someone calls their title can matter to certain sentiment measures. Furthermore, a combination of smaller supported words will likely get us a better picture then more popular words anyway given the nature of this survey’s intent.

> barplot(sent.wfreq[1:20], xlab = "Popular Other Text Responses", cex.names = 0.8)

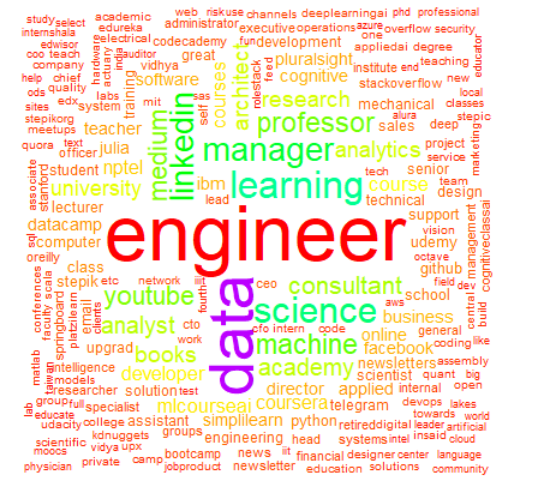


Note that the drop-off is very quick and then relatively consistent.

wordcloud(words = names(sent.wfreq), freq = sent.wfreq, random.order = F, col = rainbow(100), min.freq = 50)



Even in the word minimum word frequency word cloud, we can begin to see some words with greater sentiment such as great and support, although it is not until we widen the gates that we get something more substantial.



Here we being to get additional words like solutions, team, open, like, help, lead, and quality. With that being said it should be noted that context may affect the true nature of these words, so one should be careful to make a sweeping claim even if we do return a strong result.

temp = sent$ï..Full\_Other\_Text\_12\_13\_19\_5\_9

temp = str\_replace\_all(temp,"&amp", replacement = "and")

temp\_sente = get\_sentences(temp)

sent.sen = sentiment(temp\_sente, polarity\_dt = hash\_sentiment\_huliu)

To start out hash\_sentiment\_huliu was used, which the r library Lexicon describes as, “A data.table dataset containing an augmented version of Hu & Liu’s (2004) positive/negative word list as sentiment lookup values.” This list was originally intended for customer reviews and is meant to pick up on specific strings of words that indicate positive or negative features. The idea here is that it might be able to score true sentiments higher than just singular descriptions which might help mitigate getting unintended positives.

> mean(sent.sen$sentiment)

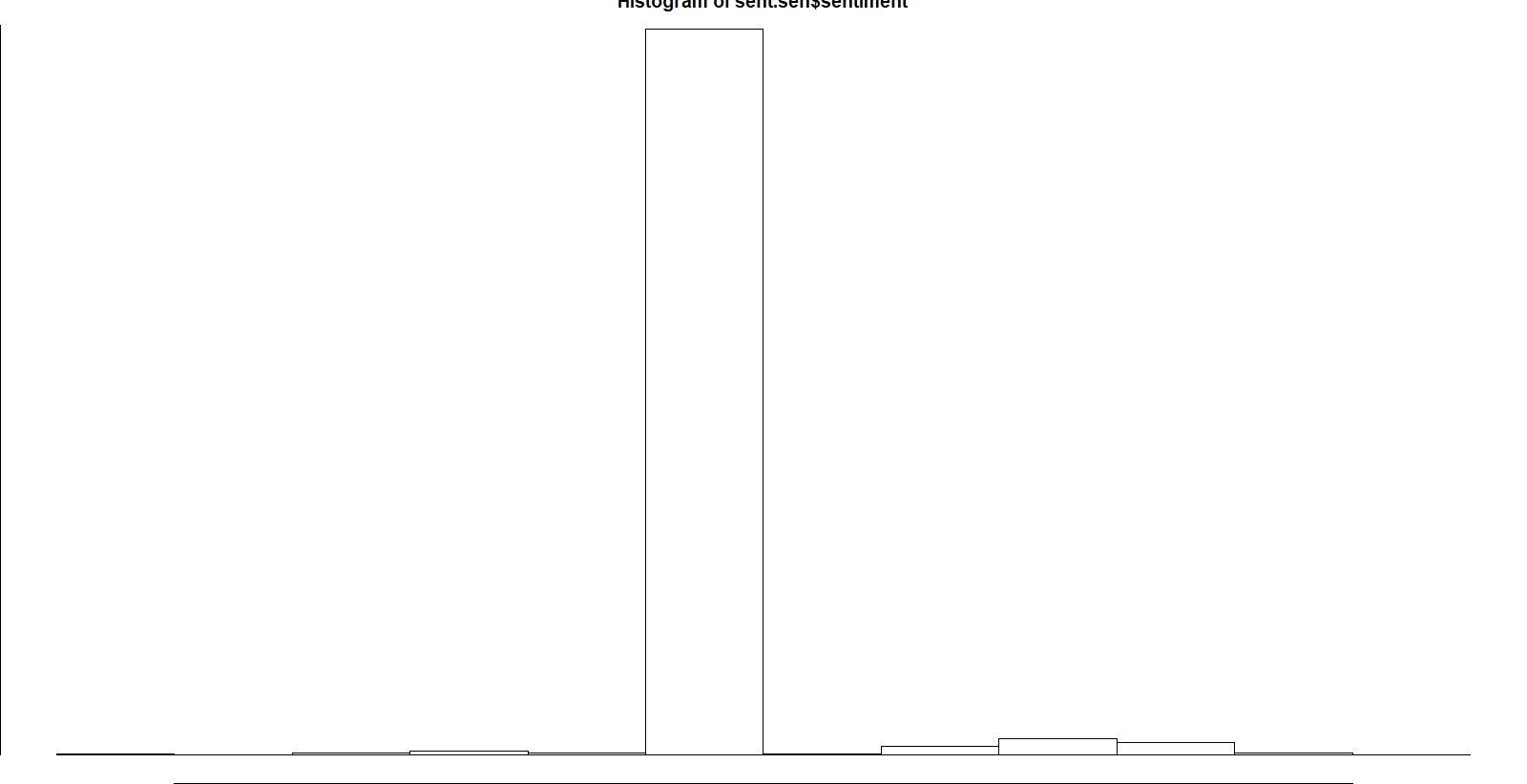
[1] 0.02184088

> mean(sent.sen$sentiment[sent.sen$sentiment!=0])

[1] 0.3396524

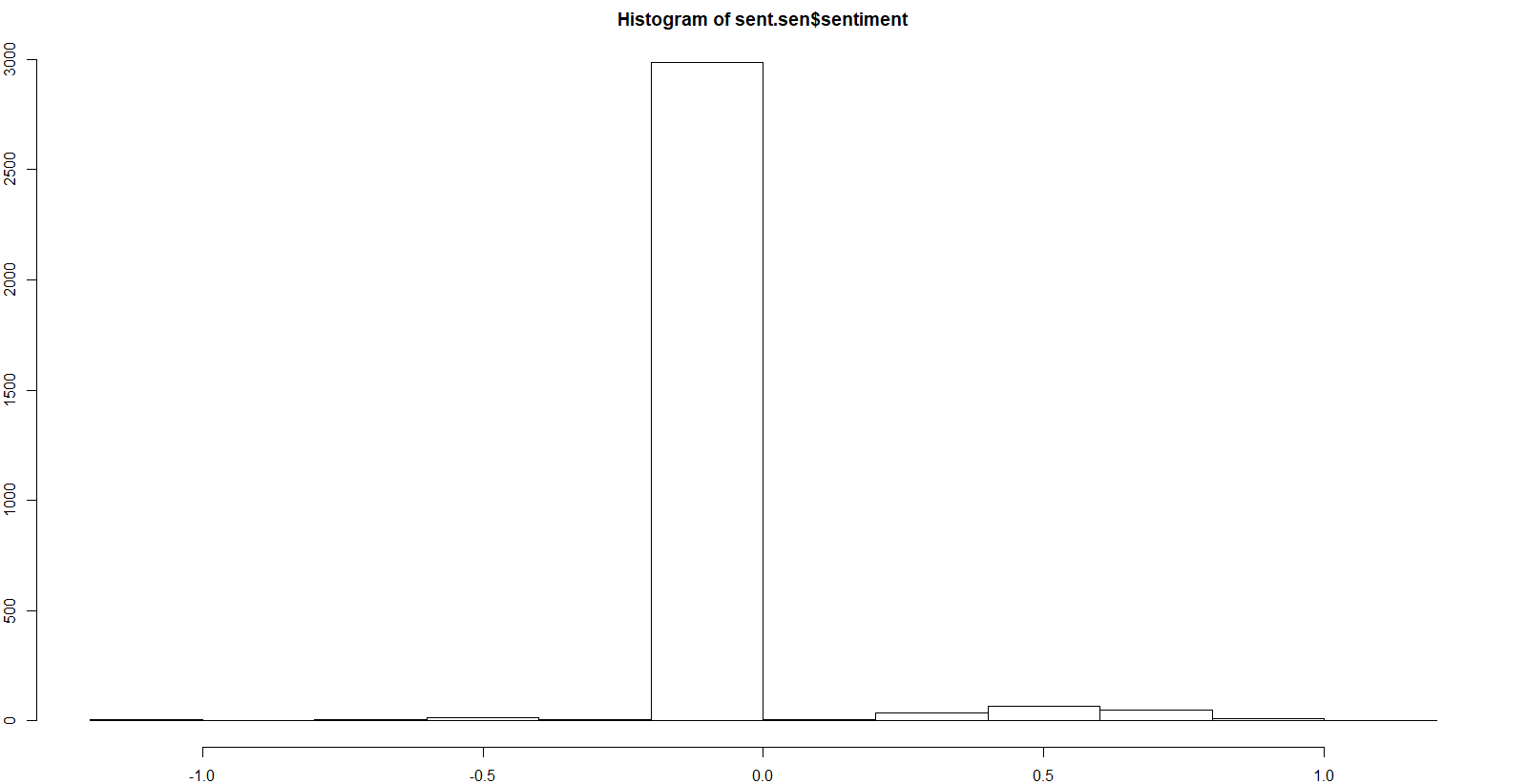
Not very high as expected.

hist(sent.sen$sentiment)



Unfortunately, it appears that is has returned enough zero sentiment values to break Rstudio’s plot window. This was fixed by manually changing the margins.

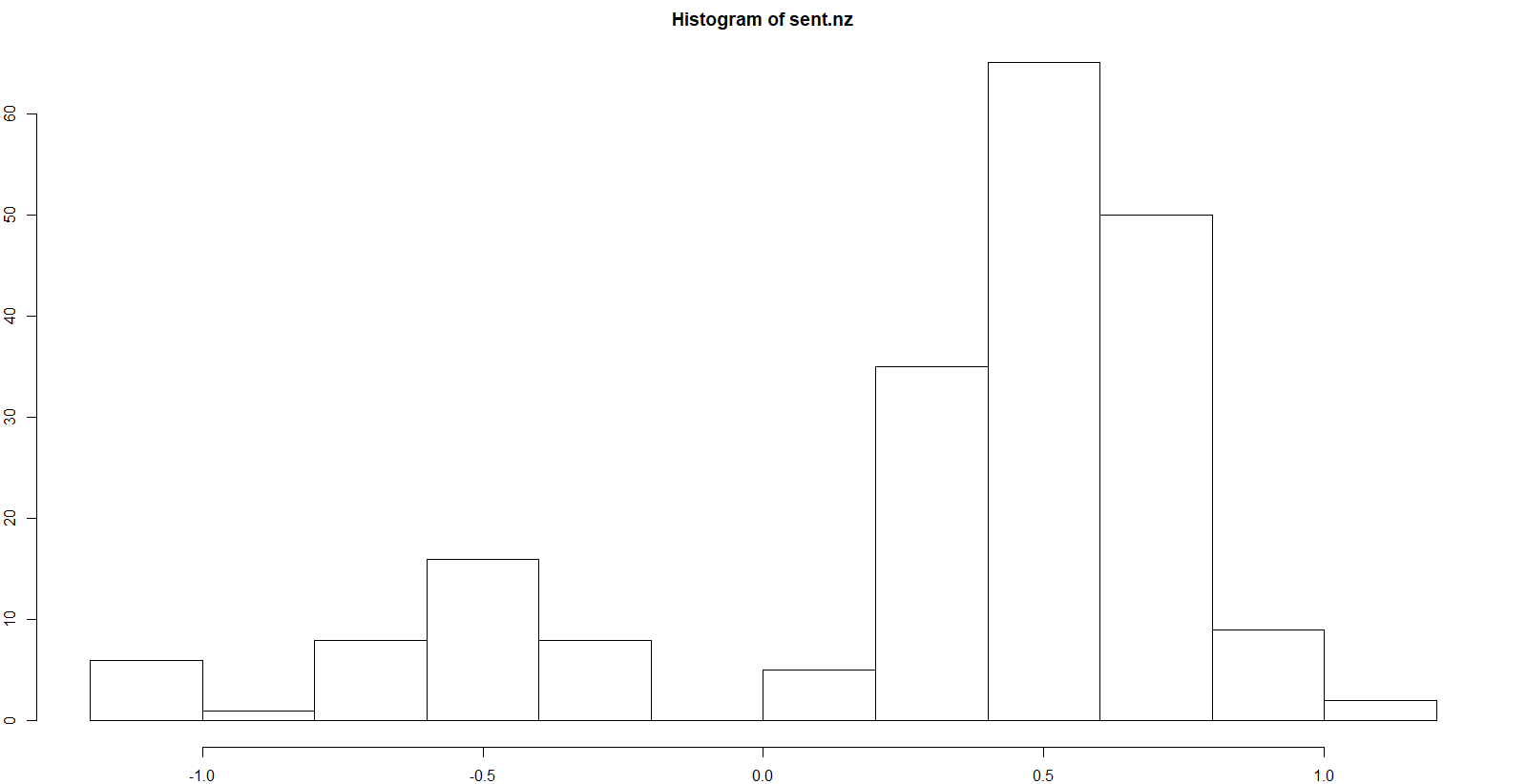
par(mar = c(2, 2, 2, 2))



Flat zeros are shown to dominate. Although, one still might want to get an idea of what people thought other than straight neutral values. Therefore, the zero values where removed for the construction of an additional histogram that might allow onlookers to get a better idea of what everyone else is thinking.

sent.nz = sent.sen$sentiment[sent.sen$sentiment!=0]

hist(sent.nz)



One can definitely spot a clear skewness leading into positivity, which follows the casual observation made earlier. However, even if there is something greater to that, the level is only a little less than 1.0 at the most, although we should consider that most of these responses are rather short. Disregarding all of that though, there is still enough lack of a sample of nonzero sentiments to say anything other than failing to reject the hypothesis the project had going into this. There are still other sentiments left to try though.

sent.sen2 = sentiment(temp\_sente, polarity\_dt = hash\_sentiment\_senticnet)

Next, hash\_sentiment\_senticnet, was tried which is described in Lexicon’s documentation as, “A data.table dataset containing an augmented version of Cambria, Poria, Bajpai,& Schuller’s (2016) positive/negative word list as sentiment lookup values.” This list seems more focused on individual words as opposed to larger phrases. Though that to pitfalls described previously, it could also allow individuals to get something of a bit more substance.

> mean(sent.sen2$sentiment)

[1] 0.1997935

> mean(sent.sen2$sentiment[sent.sen$sentiment!=0])

[1] 0.3298224

An improvement, as this had way less zeros and many more positives. Although adjusting for zeros the overall sentiment is slightly less.

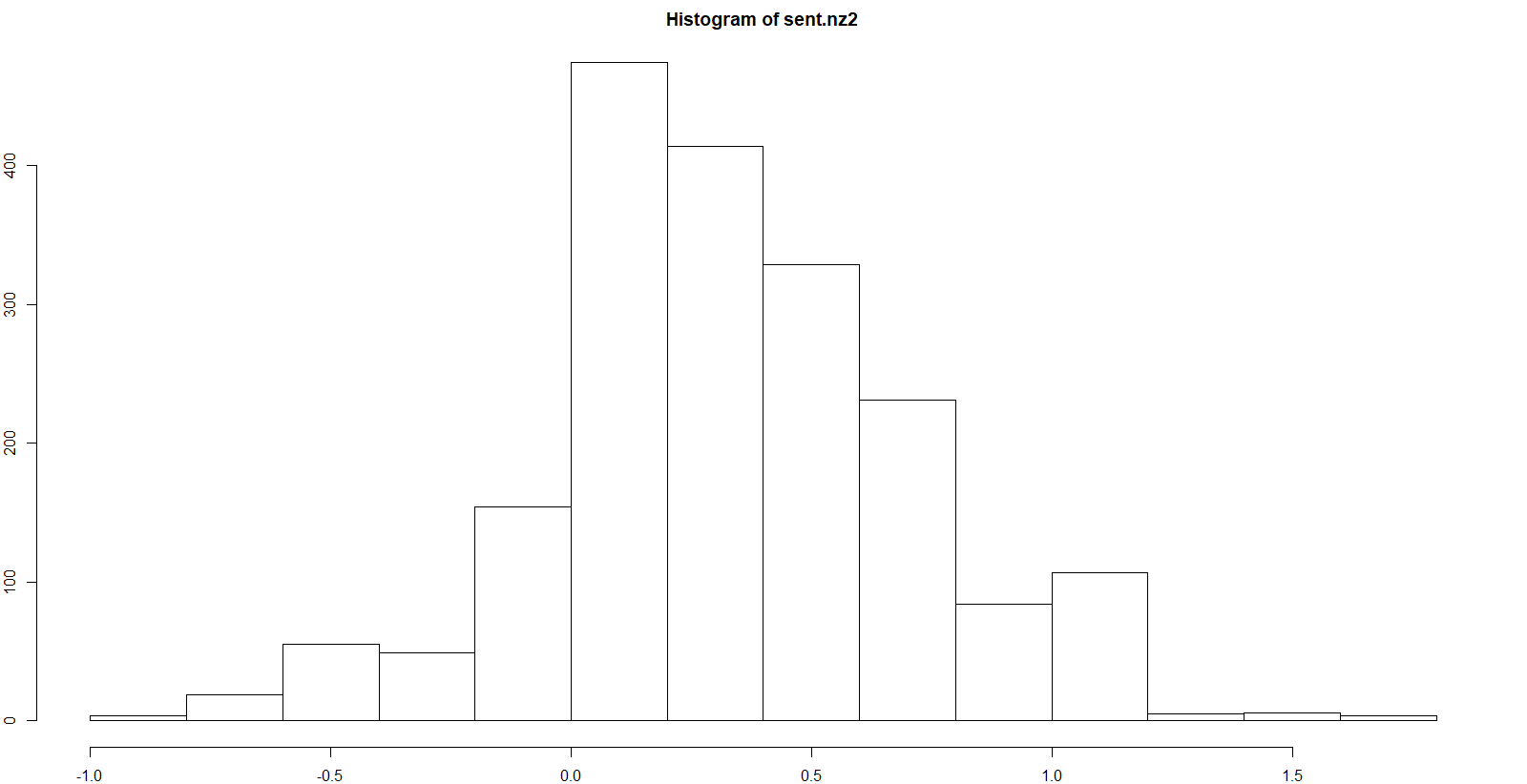
sent.sen2 = sentiment(temp\_sente, polarity\_dt = hash\_sentiment\_senticnet)



While it still seems to have a lot of neutral responses, one can see an improvement in the variety. The maximum positive amount is also stronger, as better seen down below.

sent.nz2 = sent.sen2$sentiment[sent2.sen$sentiment!=0]

hist(sent.nz2)



Compared to our last plot, we have much more support for values without zero sentiment, although with the way this scaling works, there is still a lot close to it. Although, there is favoritism shown positive sentiments even more than the previous metric, with it now outnumbering the negatives by much more. One should hesitate to draw to much from these results for the same reasons as before.

sent.sen3 = sentiment(temp\_sente, polarity\_dt = hash\_sentiment\_socal\_google)

Trying out one last sentiement list, hash\_sentiment\_socal\_google was used, which lexicon has documented as, “A data.table dataset containing a version of Taboada, Brooke, Tofiloski, Voll, & Stede’s (2011) positive/negative word list as sentiment lookup values.“ Back tracking the source of the list to <https://dl.acm.org/citation.cfm?id=2000518>, it appears that while it focuses on singular words like our attempt preceding this, except that it also be able to tell orientation of the word based on those around it. For example, “Not great” is treated as a singular negative compared to counting it as, say, a positive and negative or just positive. Furthermore, it appears as though it is meant to better handles intensification and negation, so that disarming or fueling comments may be better counted. Depending on how it executes that process, it may perform better or worse then what we have thus far.

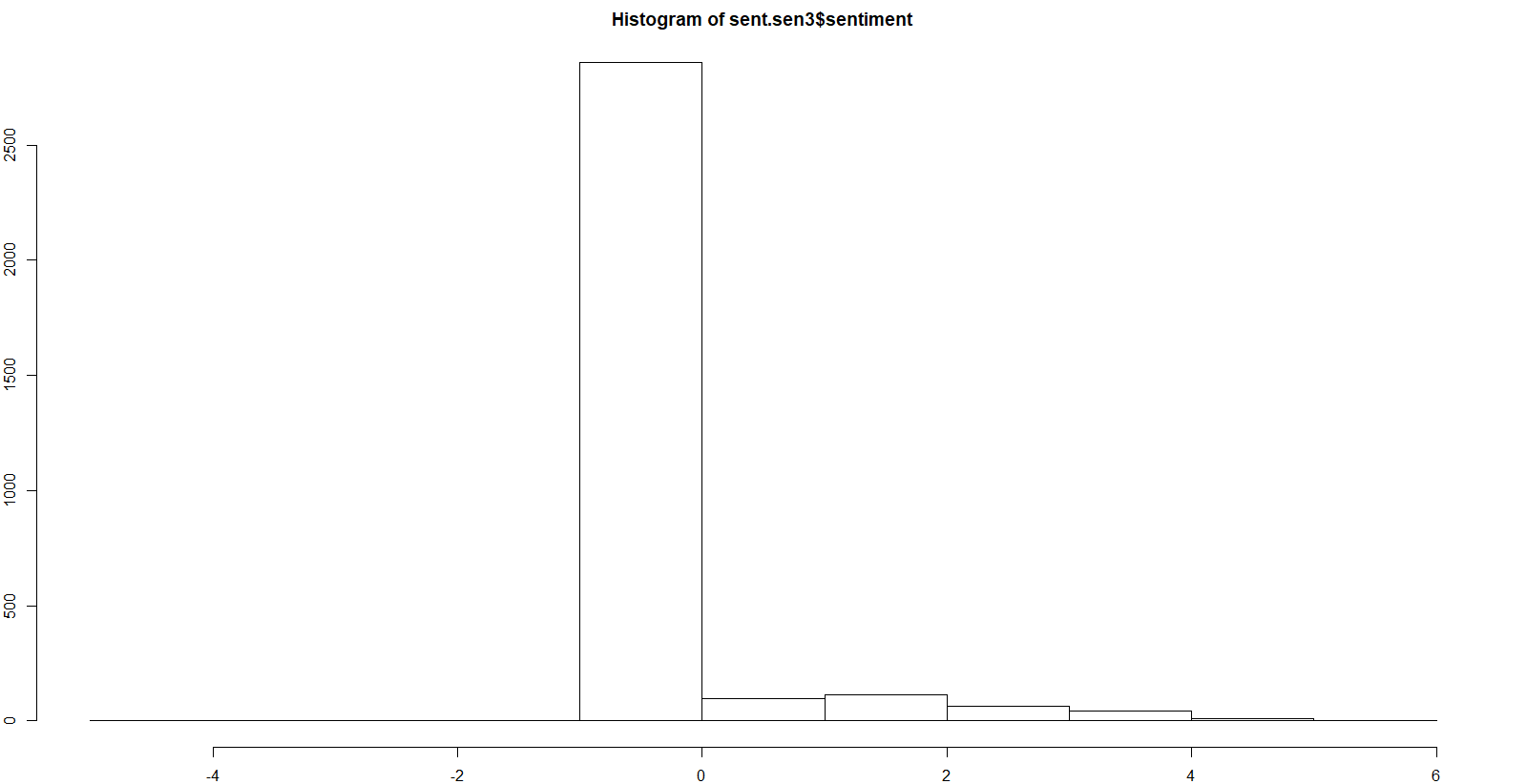
> mean(sent.sen3$sentiment)

[1] 0.1737993

> mean(sent.sen3$sentiment[sent.sen$sentiment!=0])

[1] 0.2402188

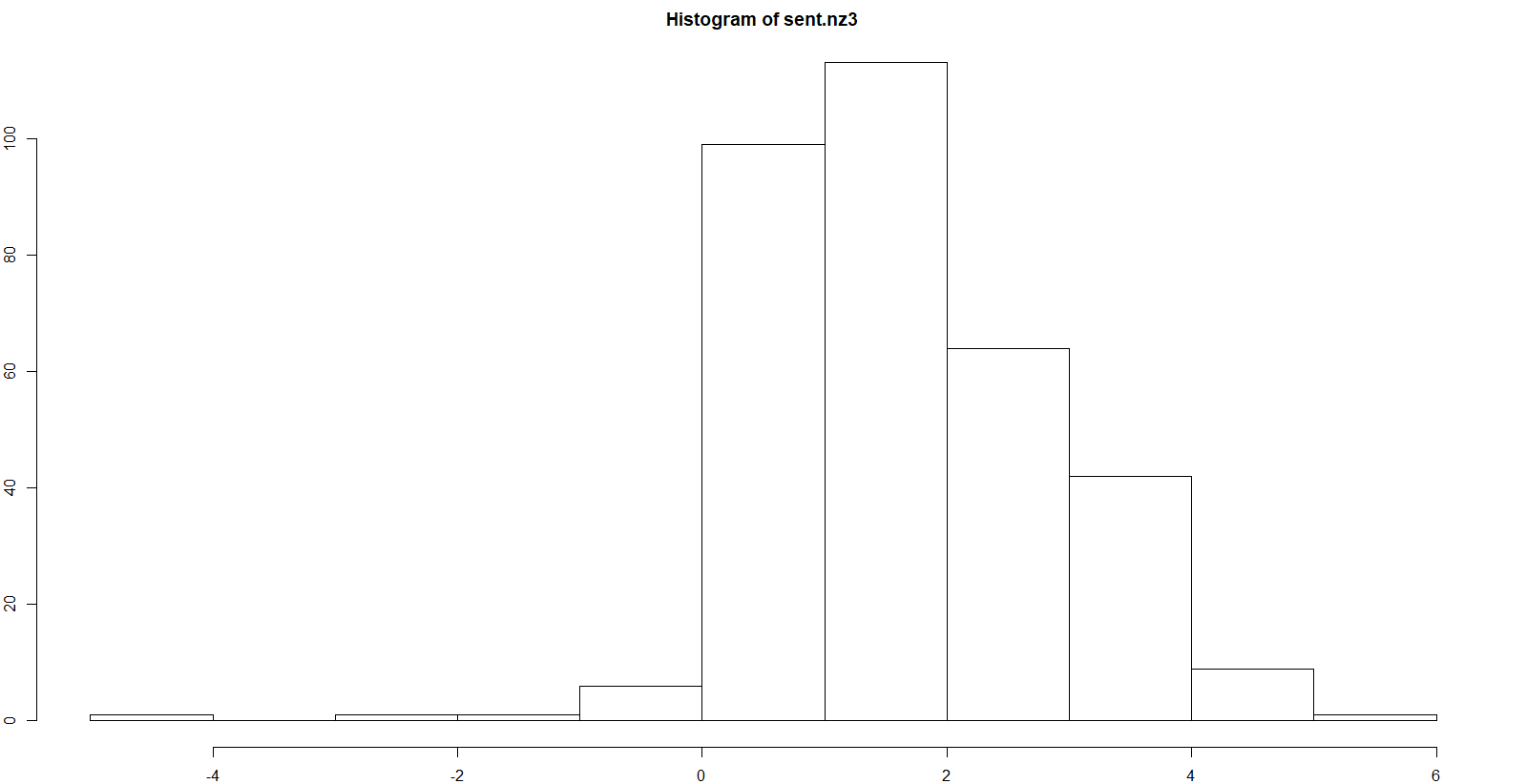
For sure the best unadjusted for zeros, but the worst once they are considered.



In terms of values of zeros, this performed by far the worst, having even more then the hash\_sentiment\_huliu. However, it appears as though the negative sentiments have disappeared entirely.

sent.nz3 = sent.sen3$sentiment[sent.sen3$sentiment!=0]

hist(sent.nz3)



Removing the zero values leaves us with the most positively oriented histogram of sentiments thus far, not only in total support, but in raw score as well. Even the negative values, being rarer, tend to be more pronounced than anything previous. This sentiment method certainly produces the most profound results in the zero-adjusted graphs, and is likely the most interesting one to be found. The previous method still produces a better summary though.

Despite this surprising turn though, one likely still should consider the sentiment analysis to fail to result in any major return that isn’t what one likely would already assume. Even with the “best” sentiment list, it still is only talking about a couple hundred people having positive sentiment in their responses compared to the 1,000s+ individuals who returned sentiment-less informational responses. One also should account for the fact that we already are only looking at a subset of roughly 3,000 individuals from a list of over 19,000 (~16,000 removing nulls) originally. Adding to the pile, because none of these text responses where mandatory, there is already a bias in favor of the type of individual who would take the time to write something of sentiment.

The good news is that one can say with greater confidence that, for the written responses from this survey, the emotions and feelings of the individuals has not greatly impacted the information they provided, as depending on the metric the distribution of scores is either almost entirely a flat zero, or mostly zero with a hint of positivity. This in itself adds credibility to dataset as a whole, and is something we likely could not have had any evidence to back up had we not done this.

Method #4 Principle Component Analysis:

The Kaggle responses included no true continuous default variables other than the duration a user took to complete the survey itself. However, many multiple-choice questions offered narrow numeric ranges for details such as age, salary, number of fellow employees, years spent coding, etc. These responses where recoded into the average value for each range (ex. Age 20-23 became 21.5). This process was done mostly within Openrefine. A sample of what it looks like after that process is below. In addition, not every individual who took the survey was given all of the questions, but rather subset of them, resulting in missing information that was dealt with accordingly mostly through the use of the missMDA package in R. After all is set and done, there will be left with only 7 continuous data columns converted from ordinal columns originally with missing values that will have to be imputed. The odds for a against great PCA output are stacked fairly high from the start, but it may prove useful to look anyway.



Normally, one would first like to look at correlations within this data, but because of the way this data was gathered, we have missing values for some columns in some rows. Trying to get correlations normally results in something like this:

Kpca <- read.csv("Cluster\_columns\_Recoded\_Relabeled2.csv")

Kpca.mat <- Kpca[,-c(1)] #get rid of the time to complete column

Kpca.mat <- scale(Kpca.mat)

cor(Kpca.mat)

Approximate.Age Approximate.Empolyees Approximate.Wage Approximate.Spending

Approximate.Age 1.0000000 NA 0.3572123 NA

Approximate.Empolyees NA 1 NA NA

Approximate.Wage 0.3572123 NA 1.0000000 NA

Approximate.Spending NA NA NA 1

Coding.Years NA NA NA NA

TPU.Times NA NA NA NA

ML.Years NA NA NA NA

Coding.Years TPU.Times ML.Years

Approximate.Age NA NA NA

Approximate.Empolyees NA NA NA

Approximate.Wage NA NA NA

Approximate.Spending NA NA NA

Coding.Years 1 NA NA

TPU.Times NA 1 NA

ML.Years NA NA 1

Using the library missMDA, we can impute missing values using what is described as an algorithm that iterates a for a set number of dimensions before a PCA is performed.

Kpca.ms <- estim\_ncpPCA(Kpca.mat,method.cv = "Kfold", verbose = FALSE)

Kpca.ms$ncp

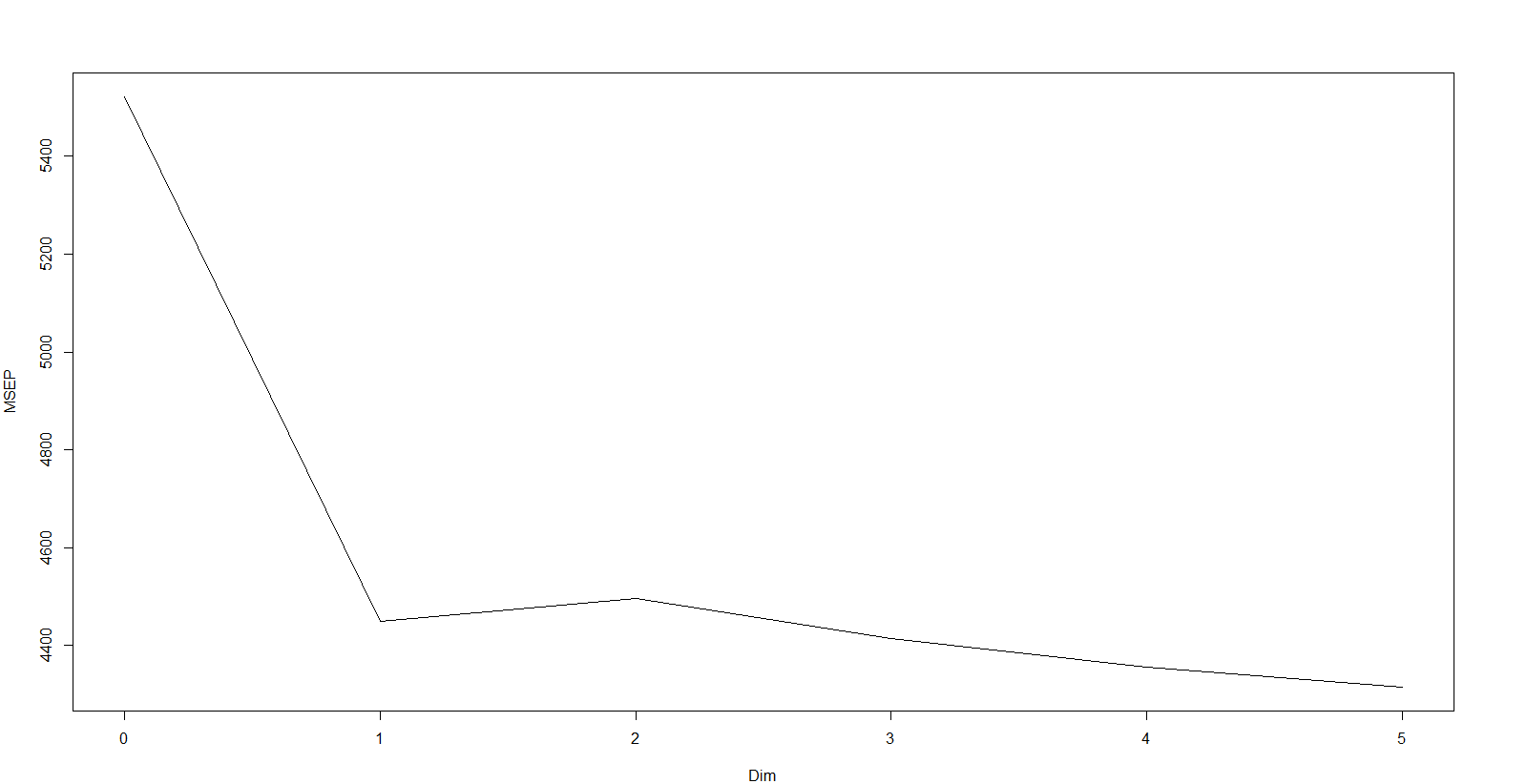
[1] 5

Set if for 5 components for data imputation.

Kpca.comp <- imputePCA(Kpca.mat, ncp = Kpca.ms$ncp)

Kpca.comp$completeObs[1:10,]

plot(0:5, Kpca.ms$criterion, xlab = "Dim", ylab = "MSEP", type = "line")



Skree plot. I would surmise that we probably are going to want to keep 2 of these. However, the function for estimating the number of PCA’s claimed that 5 would optimal. Both will be attempted starting with PCs =5

> Kpca.comp$completeObs[1:10,] # the imputed data set

Approximate.Age Approximate.Empolyees Approximate.Wage Approximate.Spending Coding.Years TPU.Times

[1,] -0.8167014 0.4203963 0.03288408 -0.4838038 -0.47307367 -0.24976335

[2,] 0.9719555 1.6523870 -0.43996848 2.8959329 -0.88952401 -0.03264443

[3,] 2.3840530 -0.1635513 -0.54276251 -0.4326516 0.59195490 -0.15863983

[4,] 0.9719555 1.6523870 3.98017501 1.2060646 -0.47307367 0.08186130

[5,] -0.8167014 -0.8054343 -0.46875081 -0.4838038 -0.68129884 -0.24976335

[6,] 1.9133538 -0.8054343 0.52629545 1.2060646 3.27497937 -0.24976335

[7,] -0.8167014 -0.7807945 -0.33717444 -0.4669051 -0.05662334 3.72973247

[8,] -0.8167014 1.6523870 0.85523636 -0.4838038 -0.05662334 0.08186130

[9,] -0.8167014 -0.0696041 -0.54276251 -0.2743169 -0.05662334 -0.24976335

[10,] 2.3840530 -0.8054343 -0.53453899 -0.4669051 0.67216476 -0.24976335

ML.Years

[1,] -0.32841736

[2,] -0.03592886

[3,] 0.36822243

[4,] -0.02275946

[5,] -0.63407525

[6,] 2.72816155

[7,] -0.02275946

[8,] 0.28289843

[9,] -0.32841736

[10,] -0.63407525

Now there is a completely imputed data set and can analysis can proceed mostly as normal.

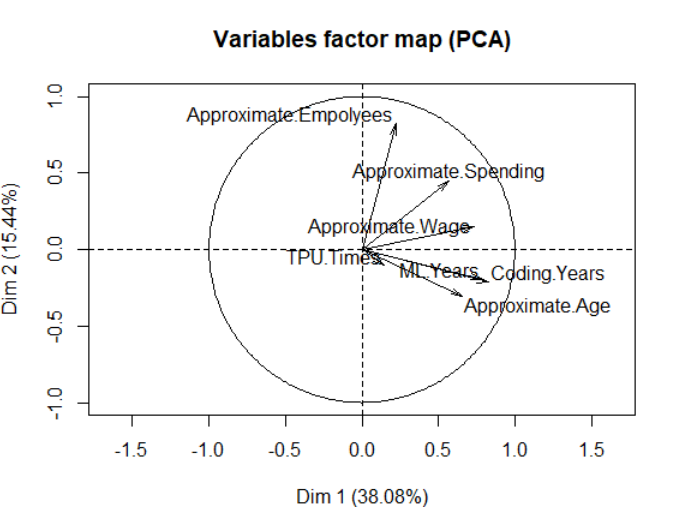
res.pca <- PCA(Kpca.comp$completeObs, ncp = Kpca.ms$ncp, graph=FALSE)

par(mar = c(4, 4, 4, 4))

plot(res.pca, lab="quali")



Now one can see a biplot reducing the data into 2-dimensions which together explain roughly 53% of the total variation. While not optimal, given the process that was used to get the data into a form usable for PCA, it could be considered impressive. Interestingly, the largest mass of points appears near the center with points slowly falling away from that clump to the right as you leave the center. Perhaps the variables will explain this pattern.



Interestingly, it appears that every single one of the variables is pulling away from the center to one side. While this doesn’t bode well for any potential clustering operations, it does makes sense as almost all of these variables could be used to measure an Kaggle users over all knowledge in working with large datasets, which would pull a more knowledge group of users away from a larger casual base. The one that pulls the most in a different direction is Approximate Employees, which would support this idea. Although, we only have seven variables and we only reduced that to 5 components as recommended by the algorithm. Perhaps if one goes with the intuition of the skree plot it could be improved by using 2.

Kpca.comp <- imputePCA(Kpca.mat, ncp = 2)

Kpca.comp$completeObs[1:10,] # the imputed data set

#imp <- cbind.data.frame(res.comp$completeObs,WindDirection)

res.pca <- PCA(Kpca.comp$completeObs, ncp = 2, graph=FALSE)

res.pca$var$coord

Dim.1 Dim.2

Approximate.Age 0.6720469 0.09081792

Approximate.Empolyees 0.2356185 -0.75035922

Approximate.Wage 0.7267909 -0.07517661

Approximate.Spending 0.6029538 -0.23677151

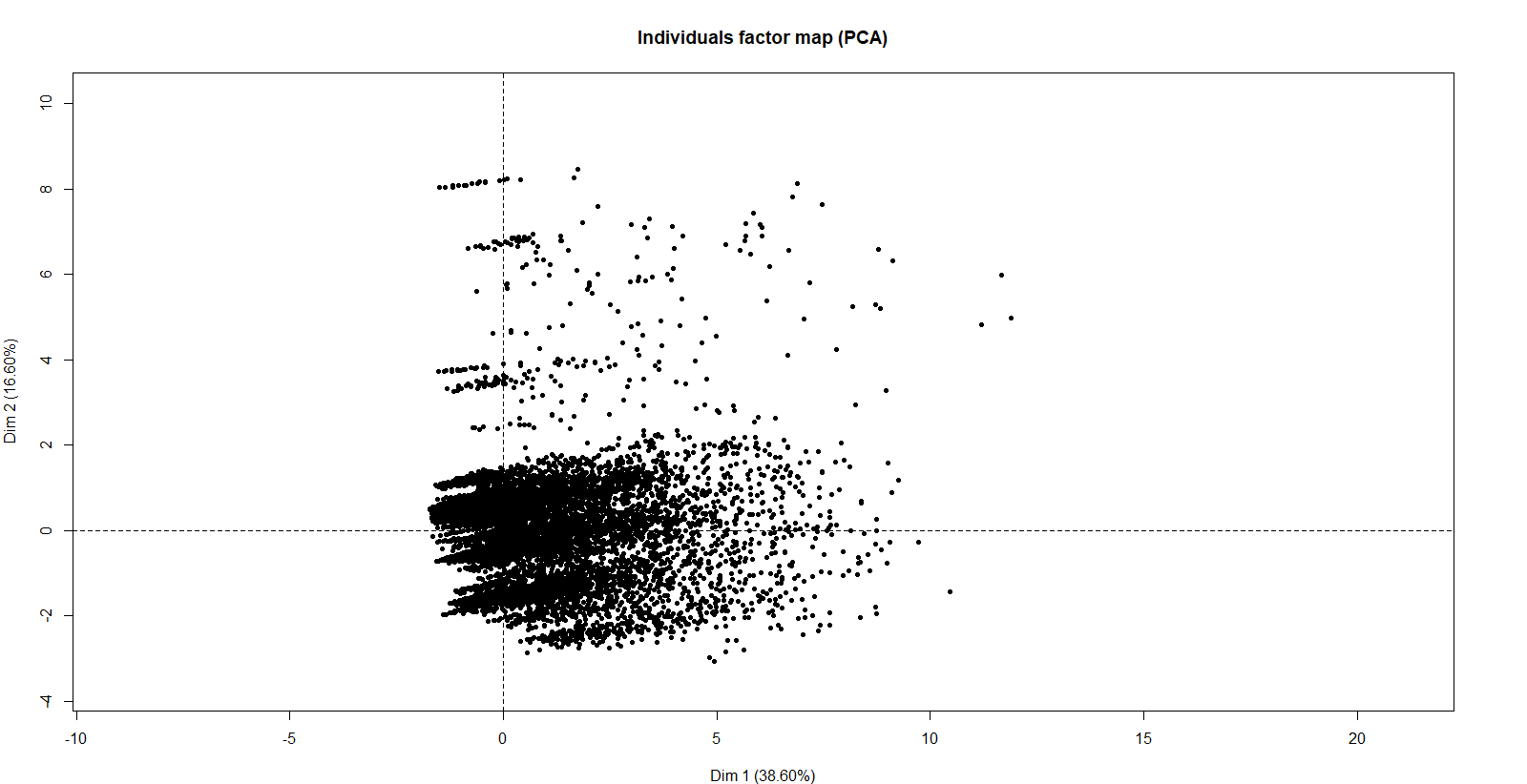
Coding.Years 0.8213063 0.11683020

TPU.Times 0.1248717 0.69856597

ML.Years 0.7830031 0.16600204

par(mar = c(4, 4, 4, 4))

plot(res.pca, lab="quali")



The major difference here going from 5 is that the separation from the middle now goes into two directions, although with way less going north then going west. The variable plot will surely provide additional context.

plot(res.pca, choix="var")



Overall, a notable improvement. Just to check, one should look at retaining and imputing with 3 and 4 PCs. We will look just at loadings to determine which is better first. It does explain about 64% of the variation, putting it as firmly better than using 5. Its patterns for the variables are pretty much the same as using 5, other than more firmly separating approximate employees and TPU experience away from everything else.

Using 3:

> Kpca.comp <- imputePCA(Kpca.mat, ncp = 3)

ML.Years

[1,] -0.32841736

[2,] 0.19884143

[3,] 0.48160547

[4,] -0.02275946

[5,] -0.63407525

[6,] 2.72816155

[7,] -0.02275946

[8,] 0.28289843

[9,] -0.32841736

[10,] -0.63407525

>

>

> res.pca <- PCA(Kpca.comp$completeObs, ncp = 3, graph=FALSE)

>

> res.pca$var$coord

Dim.1 Dim.2 Dim.3

Approximate.Age 0.6671509 -0.1299904 -0.38595223

Approximate.Empolyees 0.2058220 0.8320755 0.21408377

Approximate.Wage 0.7309051 0.1109459 0.10794291

Approximate.Spending 0.5860603 0.2816806 0.39505203

Coding.Years 0.8244091 -0.1459466 -0.17686825

TPU.Times 0.1407627 -0.4947557 0.79136674

ML.Years 0.7886289 -0.1784693 -0.07935067

> Kpca.comp <- imputePCA(Kpca.mat, ncp = 4)

>

>

>

> res.pca <- PCA(Kpca.comp$completeObs, ncp = 4, graph=FALSE)

>

> res.pca$var$coord

Dim.1 Dim.2 Dim.3 Dim.4

Approximate.Age 0.6698504 -0.29979592 -0.26300504 0.23276972

Approximate.Empolyees 0.2208176 0.82441440 -0.13118036 0.50377703

Approximate.Wage 0.7315472 0.15341210 0.03176053 -0.23947777

Approximate.Spending 0.5702768 0.44083061 0.18861136 -0.54764697

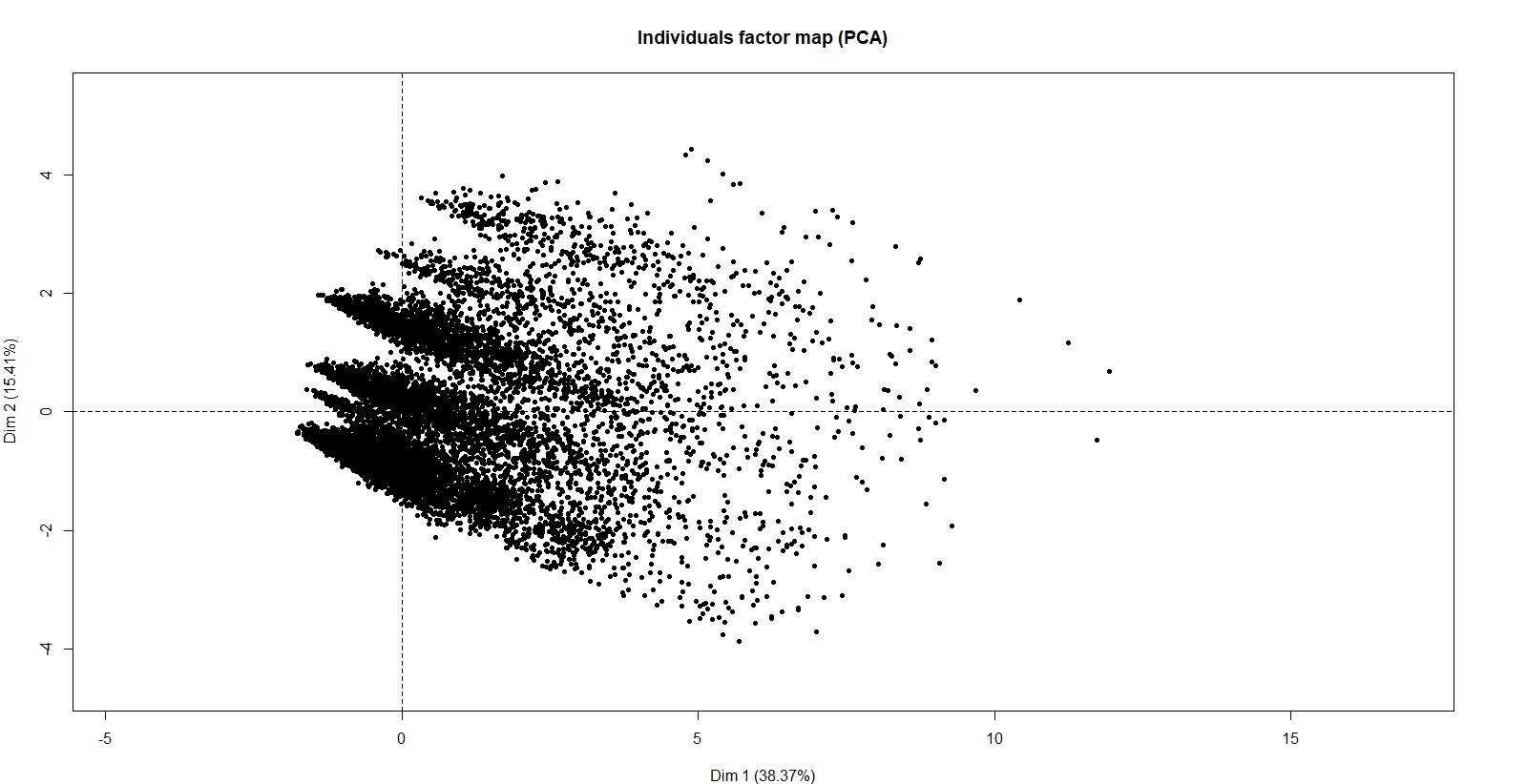
Coding.Years 0.8265511 -0.21318852 -0.08121660 0.13055103

TPU.Times 0.1363603 -0.07708865 0.94499394 0.26165312

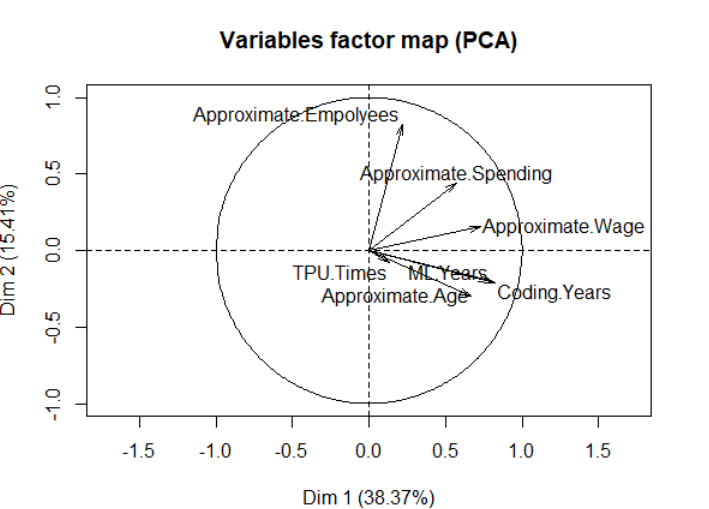
ML.Years 0.7914085 -0.19980504 0.01594110 0.09697864

Working with 3 PCs resulted in the loadings being pretty distinct for each, so well look a bit more in depth to see if it fairs any better then 2.

plot(res.pca, lab="quali")



plot(res.pca, choix="var")



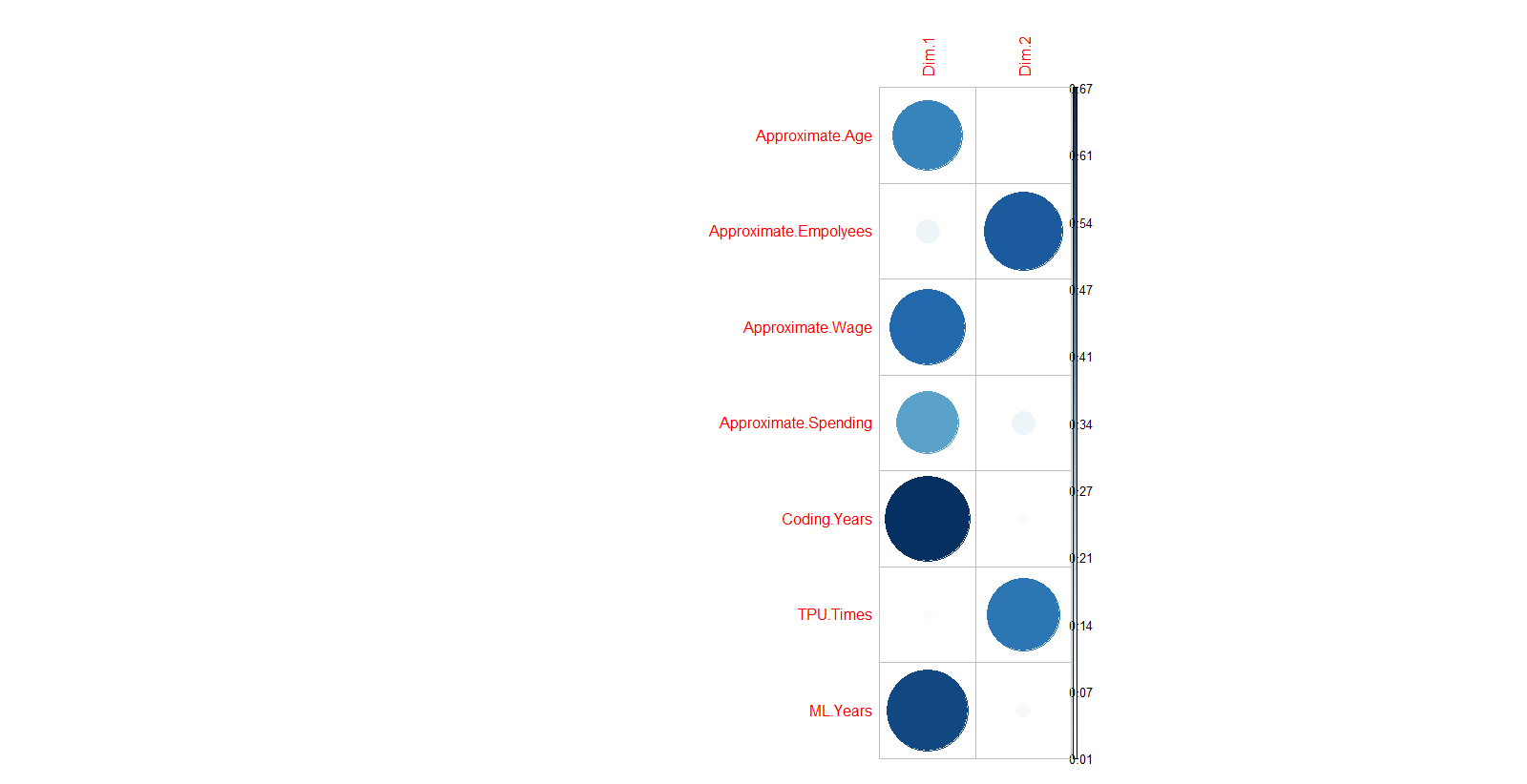
While better than using 5 like the algorithm recommended, this still fails on both fronts to be an improvement. Going any higher with more PCs then 5 would only exacerbate the issues found, and going less than 2 is simply unreasonable. Moving forward, 2 shall be the amount retained.

Corplot:

vars <- get\_pca\_var(res.pca = res.pca)

corrplot(vars$cos2, is.corr = F, )

The correlations shown here aren’t the greatest, but are the best gotten with the tested PC amounts.

Unfortunately, I am unable to take advantage of some of the more advanced PCA visualization methods included as part of Facto series of libraries as my memory tends to run out when trying to produce them. In leu of that, I will instead display a sample of values for the cos2 of individuals and the contribution of variables as raw values. Though I have no good way to display, the points on the edges are better represented then the points on the insides as expected. Particularly, the trailing points off the center have the highest measures of representation.

> res.pca$var

$`coord`

Dim.1 Dim.2

Approximate.Age 0.6720469 0.09081792

Approximate.Empolyees 0.2356185 -0.75035922

Approximate.Wage 0.7267909 -0.07517661

Approximate.Spending 0.6029538 -0.23677151

Coding.Years 0.8213063 0.11683020

TPU.Times 0.1248717 0.69856597

ML.Years 0.7830031 0.16600204

$cor

Dim.1 Dim.2

Approximate.Age 0.6720469 0.09081792

Approximate.Empolyees 0.2356185 -0.75035922

Approximate.Wage 0.7267909 -0.07517661

Approximate.Spending 0.6029538 -0.23677151

Coding.Years 0.8213063 0.11683020

TPU.Times 0.1248717 0.69856597

ML.Years 0.7830031 0.16600204

$cos2

Dim.1 Dim.2

Approximate.Age 0.45164697 0.008247895

Approximate.Empolyees 0.05551608 0.563038961

Approximate.Wage 0.52822500 0.005651522

Approximate.Spending 0.36355334 0.056060750

Coding.Years 0.67454402 0.013649295

TPU.Times 0.01559294 0.487994415

ML.Years 0.61309379 0.027556678

$contrib

Dim.1 Dim.2

Approximate.Age 16.714219 0.7096798

Approximate.Empolyees 2.054498 48.4459813

Approximate.Wage 19.548162 0.4862781

Approximate.Spending 13.454115 4.8236770

Coding.Years 24.963029 1.1744365

TPU.Times 0.577052 41.9888674

ML.Years 22.688924 2.3710798

> res.pca$ind$cos2

Dim.1 Dim.2

1 4.276566e-01 2.504592e-01

2 1.112778e-01 3.243746e-01

3 3.042650e-01 2.728803e-02

4 3.375617e-01 1.547861e-01

5 7.983469e-01 3.712510e-02

6 7.456332e-01 3.891612e-02

7 8.951800e-03 5.856098e-01

8 2.029718e-02 2.891218e-01

9 5.940401e-01 3.252560e-02

10 3.092667e-02 7.143011e-02

11 8.814439e-01 2.682701e-02

12 4.602373e-02 7.266793e-01

13 2.650036e-01 1.575196e-01

14 6.037871e-02 4.437813e-01

15 4.808532e-01 3.018012e-01

16 5.347962e-01 5.534114e-04

17 5.294793e-01 6.832984e-02

18 5.740560e-01 1.813937e-02

19 1.975165e-01 2.539699e-01

20 4.499744e-02 1.856846e-01

21 2.511491e-02 1.541919e-01

22 2.432756e-01 4.900872e-02

23 1.737413e-01 4.779958e-01

24 6.267622e-02 4.210483e-01

25 6.799302e-02 1.184228e-01

26 3.677118e-01 1.299867e-01

27 5.301151e-01 8.386688e-03

28 6.509157e-01 3.352416e-02

29 1.259847e-01 1.986048e-01

30 7.398799e-01 6.790443e-02

31 2.048889e-02 4.531537e-01

32 2.915343e-03 2.356506e-01

33 6.804994e-01 2.159027e-01

34 4.376021e-01 2.391759e-01

35 4.481176e-02 3.246486e-01

36 6.100916e-02 2.130591e-01

37 2.373302e-01 1.365823e-01

38 3.935896e-01 1.150640e-01

39 1.458835e-02 4.772745e-01

40 6.732545e-01 2.264990e-02

I later did manage to get a basic biplot of variables and individuals to work after playing with garbage collector functions and optimizing my memory usage.

fviz\_pca\_biplot(res.pca, label = "var", col.var = "black", col.ind = "grey")



While the ability to explain over 60% of the variation in 2 dimensions, despite everything that the data had to go through to get to that point, is admiral, the results should likely still be taken with a grain of salt. Though our pattern did arguably fall in line with the MCA analysis and we ended up with mostly distinct PC’s, the fact that almost all the variables seemed to measure the same thing could cause one to concede that perhaps the survey fell a few variables short of being able to truly separate out their respondents. The variables that ended up being used were all converted ordinal data, even if that data should have debatably started out as continuous to begin with.

Still, one can learn a bit more about the respondents, and one can see a divergence between the more experienced data analysts, scientists, and engineer’s vs the more casual and intermediate level users. Kaggle may appear from the outside to only attract veterans of the field, but more than one might initially surmise use it to train themselves or do things more basic.

Regardless, for any conjecture beyond that, I would recommend to try and utilize the MCA results or attempt to do advanced clustering techniques using a computer with more RAM, though that was known to likely happen going into it. The “Kaggle Survey challenge” lives up to its name, as it was no easy feat to work many common unsupervised learning techniques to it, PCA especially, but it is through our attempts that we become better data scientists and learn that every problem has its unique streaks that can’t always be resolved neatly.

Results:

It was known going into this task that useful results may be tough to come by, and all said and done that hasn’t been disproven. While both the MCA and PCA showed some indication that there may be a distinction between Kaggle users that are more familiar with Data Science work and users who don’t, both suffer from enough of a variety of issues that calling that claim confirmed would be naïve. Limited success was found in text association, as groups of individuals using particular suites did form their own groups in question 14’s response, though support was for such rules was detrimentally low. Question 28’s rules showed a distinction between machine learning users and not, although that could have been seen with a simple count. In terms of groups, all the frameworks were very intertwined, making it difficult to draw any significant lines. Finally, our sentiment analysis was grasping at straws to make any major claim beyond that responses lacked sentiment, which at most is only useful to say that written answers were not influenced by emotion.

Additional methods not utilized here could be applied as well, with the most notable one being a thorough cluster analysis of the same continues variables made in this report out of ordinal data. This was original attempted, although problems were quickly found due to computational devices lacking memory resulting in most R methods related to clustering crashing the program or refusing to run (In fact, certain PCA visuals could not even be successfully created). Should someone with more powerful machines at their disposal wanted to try it, they might be able to come up with something of note.

While the main goal of this project may not have been met, throughout it I have shown what perhaps a future survey could change about its structure to improve the quality of future analysis should Kaggle ever do one in the future. To say that nothing was gained from this in its entirety is also false, as I personally have walked away with a much more understanding of the patterns and structures that make up this data, which in itself is valuable to anybody with questions pertaining to it.