ICA: Artificial Intelligence Foundations(CIS4049-N)

Using the Application of Artificial Intelligence to Determine and Recommend Suitable Computer Operating Systems to Programmers

Chukwudi Onyema Ajoku

A Report Submitted in Partial Fulfilment of the Requirement for the Degree of
Master of Science

Module Leader: Dr. Alessandro Di Stefano

Applied Artificial Intelligence
Teesside University
Middlesbrough, England, United Kingdom
January 14, 2021

How this report was written

To meet up with the requirements and learning outcomes of this course, this report has been divided into three parts. The first part, I used a normal academic writing style to review relevant scientific literature, the basic AI concepts, history, evolution and shortcomings of AI.

In the second part, I applied AI to a given real world scenario. I used the personal pronouns throughout the rest of the book to address myself. This is with the imagination that I am teaching my self while performing the required tasks.

In the last part, I reflected on the chosen AI algorithms and on some of the challenges I encountered while doing the second part, my future goals and lastly my references.

Contents

Introduction

AI: An Overview

What is AI?

Artificial Intelligence (AI) has become the modern day electricity (Andrew Ng, Coursera) as our lives keep on revolving around it. AI is the science of making machines that can reason and act humanly and rationally. Humans have been able to simulate some amount of human intelligence in machines. The science of the techniques on how this is achieved is now know as Artificial Intelligence (AI).

"AI is the Science of making machines do things that require intelligence if done by men" (Minsky 1962)

AI has two main approaches:
1. The use of AI to understand how humans think and
2. To use AI to augment human thinking.

History and Development of AI

The concept of AI originated from fiction, imagination and philosophy, especially through efforts to understand and decipher processes of reasoning and the possibility of systematically replicating such processes in machines. Any aspect of learning/intelligence can be so systematically described that a machine can be made to simulate it as it is understood that given a set of rules, and putting few pieces of information together, an inference can be reached in a systematic way. This concept gave birth to logic. For Example, If we know that data is information, information is knowledge, and that knowledge is power, then we can infer that data is power.

It is almost difficult to tell the exact date AI became a field however, Involvement in AI can be traced up to the 1940s, specifically, 1942 when an American Fiction writer Isaac Asimov wrote a story called "Runaround" which centered around the three laws of robotics:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm;
2. A robot must obey the orders given to it by human beings except where such orders would conflict with the First Law; and
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws

This story of Asimov is believed to have sparked a lot of interest in AI.

Roughly the same time, came Alan Turing who designed the first electro-mechanical computer called "The Bombe" which was used to decipher the Enigma code used by the German army in the second world war. This was a task which even the best mathematicians at that time found very difficult to unravel. Turing with his research came up with a test on how to determine if a machine was intelligent or not. It was called the turing test which however, is more in the spheres of philosophy than in AI development.

Shortly after that, in 1956, Marvin Minsky (co-founder of MIT AI laboratory) and John McCarty (a computer scientist at Stanford) hosted a workshop at Dartmouth College which lasted for eight weeks. Among the great minds who gathered were Nathaniel Rochester (designer of IBM 701) and Claude Shannon (known for his contribution in information theory) and a host of many others.

Scientists at that time were so enthusiastic about AI. They kept believing and projecting the future possibilities of AI in a very optimistic manner. The field of AI got lots of funding for research until about 1973 when the US congress started criticising the excessive spending on AI research. These funding were withdrawn and it subsequently caused a decline in AI activities in those years.

In the 1980s some governments once more began funding AI research however, there was not much development. This stagnation was attributed to the methodology used which worked like a collective bunch of "if" and "else

if" statements. In fact, the machines then are better called "Expert systems" because they were only good at one task. Strictly speaking, one would argue that they were not truely AI although they were "smart" enough to defeat the world's chess champion at that time.

Can a machine perceive and differentiate between colours and shapes? Can it learn on its own without being hard-coded for a specific purpose? Can it be made more flexible and versatile? Such questions as the ones asked above birthed the neural networks in 1969 but at that time, there was insufficient computational power to delve into the field of Artificial Neural Networks until 2015 when Google designed AlphaGo which defeated the world champion on the board game called "Go" which is more complex than chess.

Branches and Applications of AI

Today AI can be viewed from a lot of different perspectives ranging from **cognitive sciences**, **philosophy**, **natural sciences**, **mathematics**, **linguistics**, **humanities** and even **art**.

Applications of AI:

Game playing
For games that require two or more people, AI has made it possible to play with the computer.

Speech recognition
Google assistant and Apple's Siri are able to considerably "understand" natural language.

Natural Language processing
Language translation is now being handled by computer programs. Keyboards on mobile devices can learn and predict patterns to users.

Computer Vision
Computers are now able to tell humans apart as people can now use facial lock on their mobile devices. Facebook also uses image recognition to recognise and automatically tag individuals to their photographs.

Expert Systems
Different AI systems are being developed for different purpose such as in medicine.

Business and in Advertisements:
AI is used to predict trends based on data. It is used to generate intelligent product suggestions to customers and on social media.

Future of AI

AI was built on fantasies so the expectations are quite high. AI agents achieving human-level intelligence is still very far however but is getting better with time. Many people fear that AI will be dangerous as presently, there are little or no moral compass guiding or regulating the use of AI technologies.

In terms of deep learning and neural networks, any outcome whether good or bad is possible given that it is like a black box, obscure even to fellow programmers. Perhaps there ought to be a regulatory body to standardise the way such AI systems are trained tested and deployed.

Problems and shortcomings of AI

AI has been improving our lives in diverse ways but despite that, lots of issues are of concern. For example;

1. Privacy violations:
   Data is now a huge deal and everywhere. Most times, such data could be sensitive, sold to other individuals and companies, etc.. What they decide to do with the data is completely up to them. Concerns about this has lead to measures being taken to protect data. For instance, the Data Protection Act of 1998 in the UK.

2. Discrimination:
   An AI algorithm can be gender or race biased in the sense that it can favour people of a particular gender than the opposite gender. For instance, in 2014, Amazon began developing an AI program that can identify the best resume in a screening process. It was observed that the program finally taught itself that resumes belonging to men were better ranked than those belonging to women.

3. Accidents
   In a study conducted, AI and a human were to drive a car. Unknown to the respondents who gave their verdict, more people complained of bad driving when AI drove than when a human did. The AI industry so far is being careful so as not to be blamed. If a self driving car has an accident, AI alone will likely get the whole blame.

4. Not yet truly intelligent:
   The human element of imperfection is still an important factor to as AI can pretend to but can never show true emotions. The imperfection though imperfect but is still loved as it introduces elements of surprise which is vital to human beings. This element of surprise is necessary to make and understand jokes. This is the reason why a lot of people enjoy live music performances than recorded ones. The use of AI to compose music have been explored. Although the feedbacks have been great, at least for now, humans are prefered as some decisions are better made by human beings than machines.

5. Loss of Jobs?:
   One would argue that the growth of AI would cause loss of jobs and wealth inequality. This is very understandable considering the fact that many processes are being automated and the top richest men in the world are all in AI. Some other people still believe that when AI takes up the menial jobs, humans will have the opportunity to explore jobs that require higher cognition.

6. Not foolproof:
   There is a concern that AI is being over-trusted. What if it makes a mistake? If machines are made to simulate how humans think and humans can make mistakes, why not AI? In 2016, Microsoft developed and released a chat box called Teka on twitter. The bot soon learnt to make racist remarks and Nazi propaganda. It was immediately pulled down because the company's reputation was at stake. Again, if humans can make mistakes and AI means making machines that can act like humans, this means that AI is not perfect.

Let us begin

What is the best Operating System?

In this section, I shall be illustrating how AI is applied in a real world scenario. I will be using my data to determine and suggest which Operating system is suitable for a programmer.

Data Source: I am going to be working on a set of data from a Stack Overflow Development Survey taken in 2018 which portrays the individual responses from members of stack overflow. Stack Overflow is an online community with lots of software developers who contribute to the community by sharing their programming knowledge, learning, as well as building their careers.

The particular dataset was downloaded from kaggle. It is a 186MB sized data which has 98,855 rows and 129 columns. The columns represent answers to different questions asked in the Stack Overflow 2018 survey and have been explained in this document.

My interest in this dataset is majorly fuelled by my curiosity to understand and identify trends in the software development community in Stack Overflow. Apart from that, the data is very rich and voluminous with plenty of data preprocessing and manipulations which would give me joy carrying out. Working on this dataset is an avenue for further practice and would give me the opportunity to solidify my skills in the use of R programming language for data cleaning, manipulation, visualisation and machine learning implementation.

Aim: My goal is to determine what factors that could be affecting the choice of Operating System (OS) of the survey participants and to predict what type of Operating System one is likely to use as a programmer, given that I know a few features about the programmer. It has always been an online war when topics comparing OS are discussed in online forums like reddit, quora, stackoverflow, etc. Those who use Linux like myself tend to value linux better over windows. However, I personally believe that the choice of programminmg languages you use, softwares, etc. should determine what type of OS to choose. For example, someone whose work is centered on the use of Power Bi Desktop application is more likely to choose windows over linux since the software is not yet available for Linux systems. This is really important to me as I try to put an end to this argument once and for all using my model.

Focus: This dataset has 129 columns. I do not desire to use all the columns and will therefore choose only those columns which will help me realise my goal. I shall focus on features that are most likely to affect the choice of programming languages. To identify these features, Firstly, I will need to visualise the programming language against a few chosen features in the columns in my dataset. These features will be chosen based on common sense. For instance, I expect a feature like *programming language worked with* to have a relationship with the the OS. I shall try to visualise these features with the OS and check the relationship. There is a function pair.panels() from the psch package that will allow me to do this. I will highlight it before beginning my visualisations.

For now, I will load the data in a variable called **full.dataset** Before I begin, let me invoke the packages I need.

```
library(dplyr)    # for data manipulation
library(ggplot2) # to plot and visualise data
library(scales)   # for graphs
library(caret)
library(naivebayes)
library(e1071)
library(forcats) # has a function fct_infreq() which can be used to sort graphs
# in ascending or descending order.
library(psych)    #will use pairs.panels() to visualise my variables
library(treemapify)  # for treemaps
library(randomForest)
library(doSNOW) # this library will allow me utilise my full computational power
# (parallel computing) when doing my cross validation which is likely to take up so much
```

```
# time
library(janitor) #to handle illegal names
```

Then read the CSV file.

```
full.dataset <- read.csv("survey_results_public.csv")
```

I could view the summary of my dataset and the first 6 lines using the code below.

```
print(summary(full.dataset)) # view summary
print(head(full.dataset)) # view first six lines on the data
```

However, I will try not to run it as its display will mess up my document. Printing 129 columns on an A4 document looks rough and will use up my allowable number of words (4000). As I go on and try to visualise my data, Wherever the display is rough, I will not display it here. Wherever the display is extremely important, I will attach a screenshot instead. If you have any doubts, Please kindly test my codes using the .Rmd file. If you are still viewing this file as a LaTeX pdf document, please revert to the .Rmd file and follow me.

Choosing my columns:

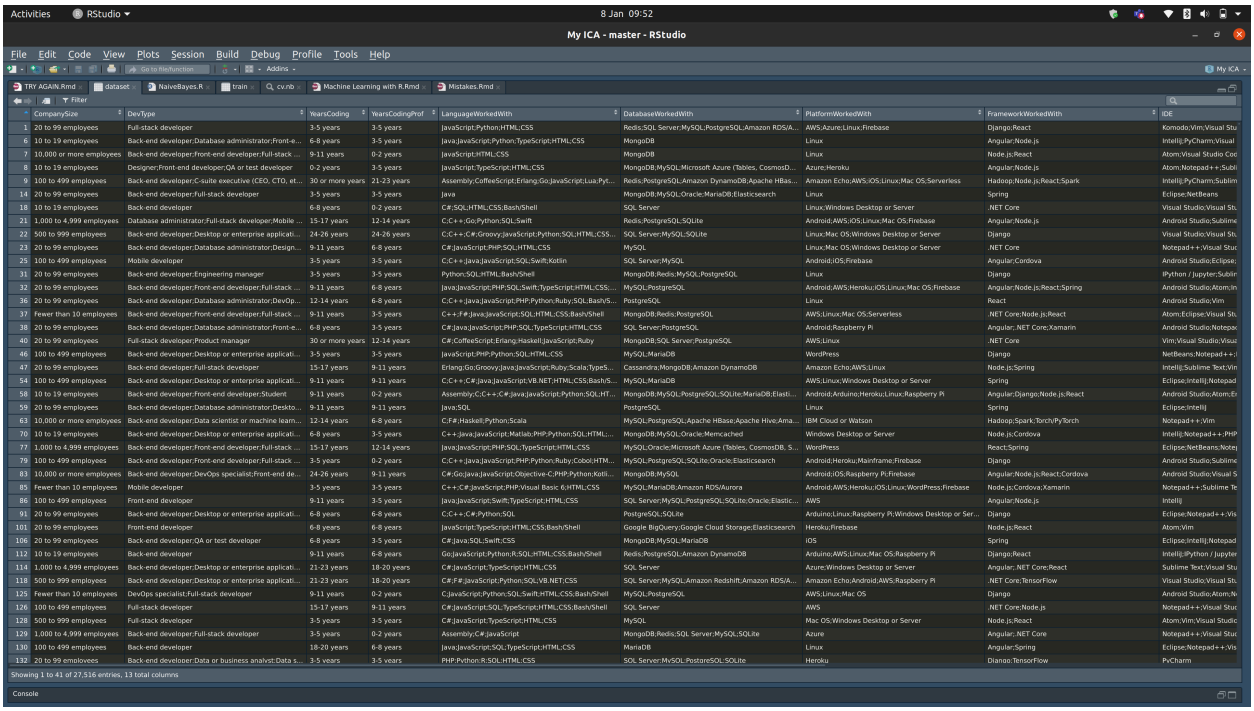
```
my.columns <- c(`CompanySize',
                `DevType',
                `YearsCoding',
                `YearsCodingProf',
                `LanguageWorkedWith',
                `DatabaseWorkedWith',
                `PlatformWorkedWith',
                `FrameworkWorkedWith',
                `IDE',
                `OperatingSystem',
                `NumberMonitors',
                `Methodology',
                `VersionControl'
                  )

dataset <- select(full.dataset, all_of(my.columns))
```

Let us remove all the rows containing the empty "NA" values from the selected dataset and in situations where one person might fill the survey twice, to reduce redundancy, let us make sure that no same row occurs more than once.

```
dataset <- na.omit(dataset) # eliminate the NAs
dataset <- unique(dataset)  # remove double entries.
```

Visualising our dataset table;

```
View(dataset)
```



As you can see, the data looks outrageously huge. they are all categorical values. I will therefore make them all a factor.

```
dataset[my.columns] <- lapply(dataset[my.columns], factor)
# Here, I select all columns and make them all factors.
str(dataset)
```

```
## `data.frame':    27516 obs. of  13 variables:
##  $ CompanySize        : Factor w/ 8 levels "1,000 to 4,999 employees",..: 5 2 3 2 4 5
 ↪  2 1 7 5 ...
##  $ DevType            : Factor w/ 4165 levels "Back-end developer",..: 4118 2158 2993
 ↪  3765 77 3050 1 3639 2743 1633 ...
##  $ YearsCoding        : Factor w/ 11 levels "0-2 years","12-14 years",..: 8 10 11 1 9
 ↪  8 10 3 6 11 ...
##  $ YearsCodingProf    : Factor w/ 11 levels "0-2 years","12-14 years",..: 8 8 1 8 5 8
 ↪  1 2 6 10 ...
##  $ LanguageWorkedWith : Factor w/ 12556 levels "Assembly","Assembly;C",..: 12107 11240
 ↪  11696 12345 1319 10532 7020 3048 2091 6532 ...
##  $ DatabaseWorkedWith : Factor w/ 4986 levels "Amazon DynamoDB",..: 4086 951 951 1084
 ↪  3851 1099 4320 3933 4729 2883 ...
##  $ PlatformWorkedWith : Factor w/ 6135 levels "Amazon Echo",..: 4265 5937 5937 5123
 ↪  675 5937 6039 2251 5972 5972 ...
##  $ FrameworkWorkedWith: Factor w/ 834 levels ".NET Core",".NET Core;Cordova",..: 661
 ↪  489 749 489 696 822 1 489 583 1 ...
##  $ IDE                : Factor w/ 4669 levels "Android Studio",..: 4248 4010 2853 2713
 ↪  3990 3270 4666 1875 4666 4471 ...
##  $ OperatingSystem    : Factor w/ 4 levels "BSD/Unix","Linux-based",..: 2 2 3 4 3 2 4
 ↪  3 3 4 ...
##  $ NumberMonitors     : Factor w/ 5 levels "1","2","3","4",..: 1 2 2 1 1 5 1 1 3 2 ...
##  $ Methodology        : Factor w/ 388 levels "Agile","Agile;Evidence-based software
 ↪  engineering",..: 272 1 272 187 7 1 1 143 272 388 ...
```

```
##  $ VersionControl     : Factor w/ 76 levels "Copying and pasting files to network
↪  shares",..: 5 5 5 5 5 5 5 5 29 1 ...
##  - attr(*, "na.action")= 'omit' Named int  2 3 4 5 10 11 12 13 15 16 ...
##   ..- attr(*, "names")= chr  "2" "3" "4" "5" ...
```

I will use the "summary()", "head()" and the "str()" command to narrow down on each of the chosen columns in order to have a rough idea of its contents. I shall also clean each column as I introduce them by narrating the question that was asked for that column in the survey.

Despite having converted the needed columns to factors, that is not enough. There are some specific columns which we will need to factor in levels, in an orderly manner as they appear. Columns like "CompanySize", "YearsCoding", "YearsCodingProf" and "NumberMonitors" need to be ordered in consecutive levels as it will help us in our visualisation.

CompanySize:

Approximately how many people are employed by the company or organization you work for?

```
print(summary(dataset$CompanySize))
```

```
## 1,000 to 4,999 employees       10 to 19 employees 10,000 or more employees
##                     2935                     2974                     3801
##     100 to 499 employees       20 to 99 employees 5,000 to 9,999 employees
##                     5558                     6737                     1095
##     500 to 999 employees  Fewer than 10 employees
##                     1800                     2616
```

This column needs to be ordered. let me properly order this in levels.

```
dataset$CompanySize <- factor(dataset$CompanySize,
                              order=TRUE,
                              levels = c("Fewer than 10 employees",
                                         "10 to 19 employees",
                                         "20 to 99 employees",
                                         "100 to 499 employees",
                                         "500 to 999 employees",
                                         "1,000 to 4,999 employees",
                                         "5,000 to 9,999 employees",
                                         "10,000 or more employees"))

str(dataset$CompanySize)
```

```
##  Ord.factor w/ 8 levels "Fewer than 10 employees"<..: 3 2 8 2 4 3 2 6 5 3 ...
```

The output showing "Ord.factor" shows that the factor levels have been ordered.

DevType:

Which of the following describes you? Please select all that apply.

```
print(head(dataset$DevType))
```

```
## [1] Full-stack developer
## [2] Back-end developer;Database administrator;Front-end developer;Full-stack developer
## [3] Back-end developer;Front-end developer;Full-stack developer
## [4] Designer;Front-end developer;QA or test developer
## [5] Back-end developer;C-suite executive (CEO, CTO, etc.);Data or business
↪  analyst;Database administrator;DevOps specialist;Engineering manager;Full-stack
↪  developer;System administrator
```

```
## [6] Back-end developer;Full-stack developer
## 4165 Levels: Back-end developer ...
```

The column is in a mess because it contains irregular number of items and will need real cleaning. The computer can not work with that.

YearsCoding:

Including any education, for how many years have you been coding?

```
print(summary(dataset$YearsCoding))
```

```
##        0-2 years      12-14 years      15-17 years      18-20 years
##             1212             3071             2341             1771
##      21-23 years      24-26 years      27-29 years        3-5 years
##              984              607              335             5568
## 30 or more years        6-8 years       9-11 years
##              896             6380             4351
```

This Column needs to be ordered in levels.

```
dataset$YearsCoding <- factor(dataset$YearsCoding,
                              order=TRUE,
                              levels = c("0-2 years",
                                         "3-5 years",
                                         "6-8 years",
                                         "9-11 years",
                                         "12-14 years",
                                         "15-17 years",
                                         "18-20 years",
                                         "21-23 years",
                                         "24-26 years",
                                         "27-29 years",
                                         "30 or more years"))
str(dataset$YearsCoding)
```

```
##  Ord.factor w/ 11 levels "0-2 years"<"3-5 years"<..: 2 3 4 1 11 2 3 6 9 4 ...
```

YearsCodingProf:

For how many years have you coded professionally (as a part of your work)?

```
print(summary(dataset$YearsCodingProf))
```

```
##        0-2 years      12-14 years      15-17 years      18-20 years
##             5746             1881             1258             1108
##      21-23 years      24-26 years      27-29 years        3-5 years
##              525              271              149             8241
## 30 or more years        6-8 years       9-11 years
##              315             4785             3237
```

We also need to arrange it's factors in levels and give appropriate names.

```
dataset$YearsCodingProf <- factor(dataset$YearsCodingProf,
                                  order=TRUE,
                                  levels = c("0-2 years",
                                             "3-5 years",
                                             "6-8 years",
                                             "9-11 years",
```

```
                                            "12-14 years",
                                            "15-17 years",
                                            "18-20 years",
                                            "21-23 years",
                                            "24-26 years",
                                            "27-29 years",
                                            "30 or more years"))

str(dataset$YearsCodingProf)
```

```
##  Ord.factor w/ 11 levels "0-2 years"<"3-5 years"<..: 2 2 1 2 8 2 1 5 9 3 ...
```

LanguageWorkedWith:

Which of the following programming, scripting, and markup languages have you done extensive development work in over the past year, and which do you want to work in over the next year? (If you both worked with the language and want to continue to do so, please check both boxes in that row.)

```
print(head(dataset$LanguageWorkedWith))
```

```
## [1] JavaScript;Python;HTML;CSS
## [2] Java;JavaScript;Python;TypeScript;HTML;CSS
## [3] JavaScript;HTML;CSS
## [4] JavaScript;TypeScript;HTML;CSS
## [5] Assembly;CoffeeScript;Erlang;Go;JavaScript;Lua;Python;Ruby;SQL;HTML;CSS;Bash/Shell
## [6] Java
## 12556 Levels: Assembly ... VB.NET
```

This column is also a mess.

DatabaseWorkedWith:

Which of the following database environments have you done extensive development work in over the past year, and which do you want to work in over the next year? (If you both worked with the database and want to continue to do so, please check both boxes in that row.)

```
print(head(dataset$DatabaseWorkedWith))
```

```
## [1] Redis;SQL Server;MySQL;PostgreSQL;Amazon RDS/Aurora;Microsoft Azure (Tables,
↪  CosmosDB, SQL, etc)
## [2] MongoDB
## [3] MongoDB
## [4] MongoDB;MySQL;Microsoft Azure (Tables, CosmosDB, SQL, etc);Google Cloud Storage
## [5] Redis;PostgreSQL;Amazon DynamoDB;Apache HBase;Apache Hive;Amazon Redshift;Amazon
↪  RDS/Aurora;Elasticsearch
## [6] MongoDB;MySQL;Oracle;MariaDB;Elasticsearch
## 4986 Levels: Amazon DynamoDB ... SQLite;Oracle;Neo4j;Elasticsearch
```

This is also in a similar mess.

PlatformWorkedWith:

Which of the following platforms have you done extensive development work for over the past year? (If you both developed for the platform and want to continue to do so, please check both boxes in that row.)

```
print(head(dataset$PlatformWorkedWith))
```

```
## [1] AWS;Azure;Linux;Firebase
```

```
## [2] Linux
## [3] Linux
## [4] Azure;Heroku
## [5] Amazon Echo;AWS;iOS;Linux;Mac OS;Serverless
## [6] Linux
## 6135 Levels: Amazon Echo ... WordPress;Firebase
```

FrameworkWorkedWith:

Which of the following libraries, frameworks, and tools have you done extensive development work in over the past year, and which do you want to work in over the next year?

```
print(head(dataset$FrameworkWorkedWith))
```

```
## [1] Django;React                Angular;Node.js
## [3] Node.js;React               Angular;Node.js
## [5] Hadoop;Node.js;React;Spark Spring
## 834 Levels: .NET Core .NET Core;Cordova ... Xamarin;Torch/PyTorch
```

IDE:

Which development environment(s) do you use regularly? Please check all that apply.

```
print(head(dataset$IDE))
```

```
## [1] Komodo;Vim;Visual Studio Code
## [2] IntelliJ;PyCharm;Visual Studio Code
## [3] Atom;Visual Studio Code
## [4] Atom;Notepad++;Sublime Text;Visual Studio Code
## [5] IntelliJ;PyCharm;Sublime Text;Vim
## [6] Eclipse;NetBeans
## 4669 Levels: Android Studio Android Studio;Atom ... Xcode
```

OperatingSystem

What is the primary operating system in which you work?

```
print(summary(dataset$OperatingSystem))
```

```
##    BSD/Unix Linux-based       MacOS     Windows
##          35        5993        8456       13032
```

NumberMonitors:

How many monitors are set up at your workstation?

```
print(summary(dataset$NumberMonitors))
```

```
##           1           2           3           4 More than 4
##        6365       15376        5106         374         295
```

Let us order it in levels.

```
dataset$NumberMonitors <- factor(dataset$NumberMonitors,
                                 order=TRUE,
                                 levels = c( "1",
                                             "2",
                                             "3",
                                             "4",
```

```
                                                "More than 4"))
str(dataset$NumberMonitors)
```

```
##  Ord.factor w/ 5 levels "1"<"2"<"3"<"4"<..: 1 2 2 1 1 5 1 1 3 2 ...
```

Methodology:

Which of the following methodologies do you have experience working in?

```
print(head(dataset$Methodology))
```

```
## [1] Agile;Scrum
## [2] Agile
## [3] Agile;Scrum
## [4] Agile;Extreme programming (XP);Scrum
## [5] Agile;Evidence-based software engineering;Extreme programming (XP);Formal standard
↪  such as ISO 9001 or IEEE 12207 (aka "waterfall" methodologies);Kanban;Lean;Pair
↪  programming;Scrum
## [6] Agile
## 388 Levels: Agile ... Scrum
```

This column is also in a mess.

VersionControl:

What version control systems do you use regularly? Please select all that apply.

```
print(tail(dataset$VersionControl))
```

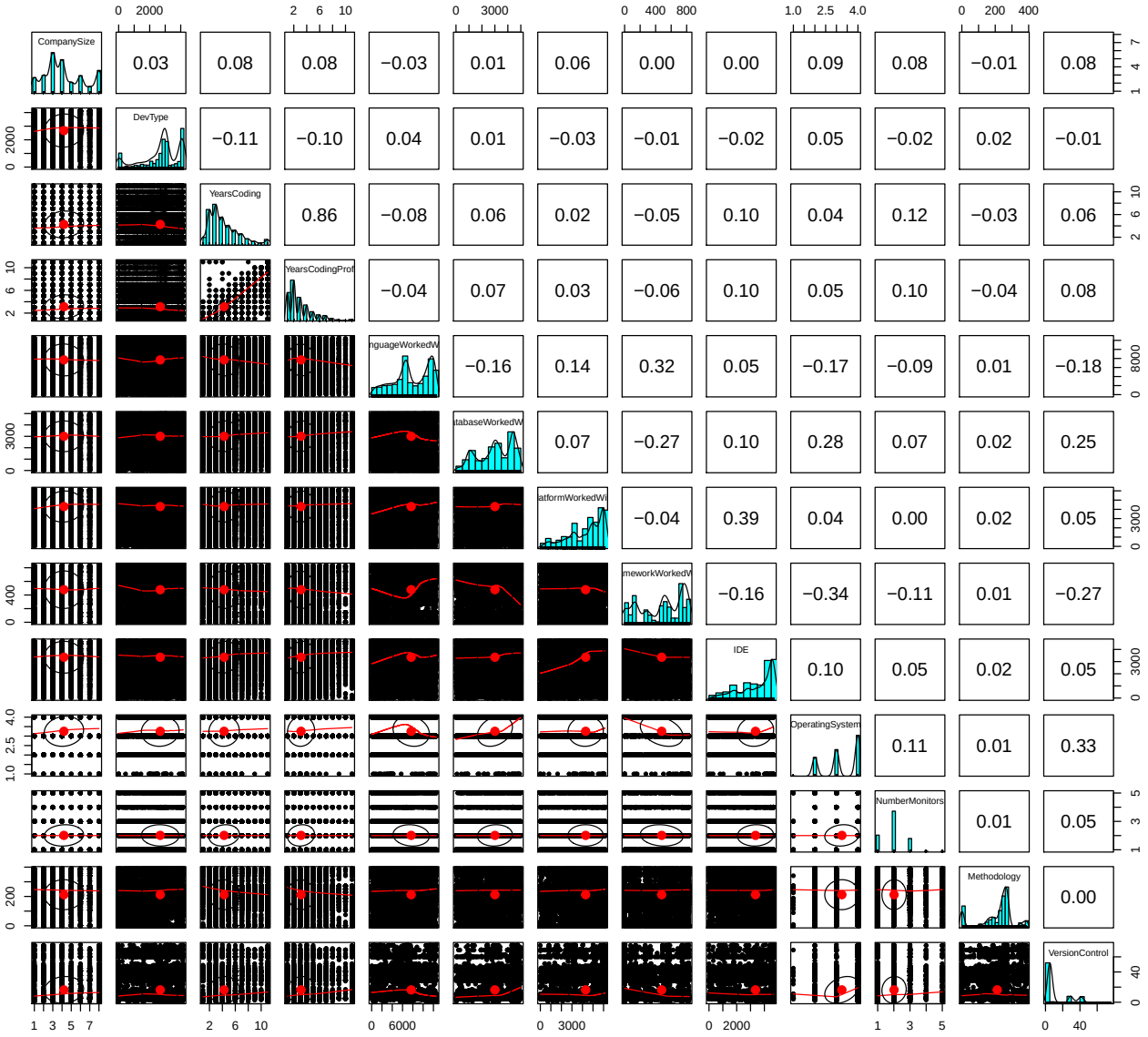
```
## [1] Git                                  Git
## [3] Git;Team Foundation Version Control Git
## [5] Git                                  Git
## 76 Levels: Copying and pasting files to network shares ...
```

Also messy.

Visualising the data.

Let me find out if there is any variance in my data.

```
pairs.panels(dataset)
```



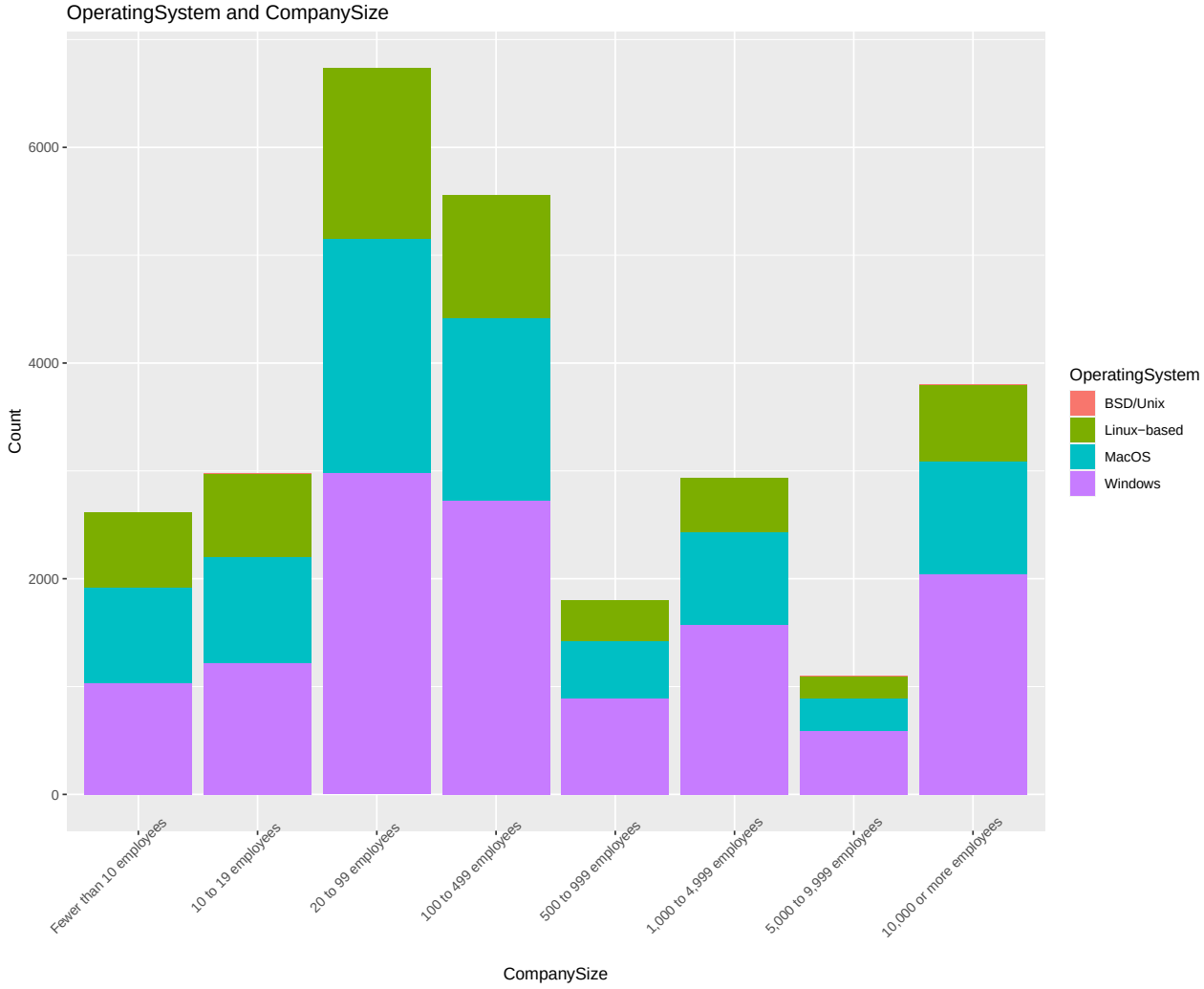
The above graph shows the relationship summary between all my variables. The diagonal is shows the name of the variable and its histogram. To see the corresponding graph or correlation coefficient of any two variables, imagine a vertical line on the cell of the first variable and also an imaginary horizontal line on the second variable. Where these two lines meet on the left is the relationship plot and wherever they meet on the right is the correlation coefficient.

From the plot above, It is quite clear that all my variables are categorical in nature.

I shall now carry out with indiidual visualisations. I prefer to use a stacked histogram as it really gives me an idea of proportion between two variables (frequency (as with histograms), the x axis as well as having another variable as legend)

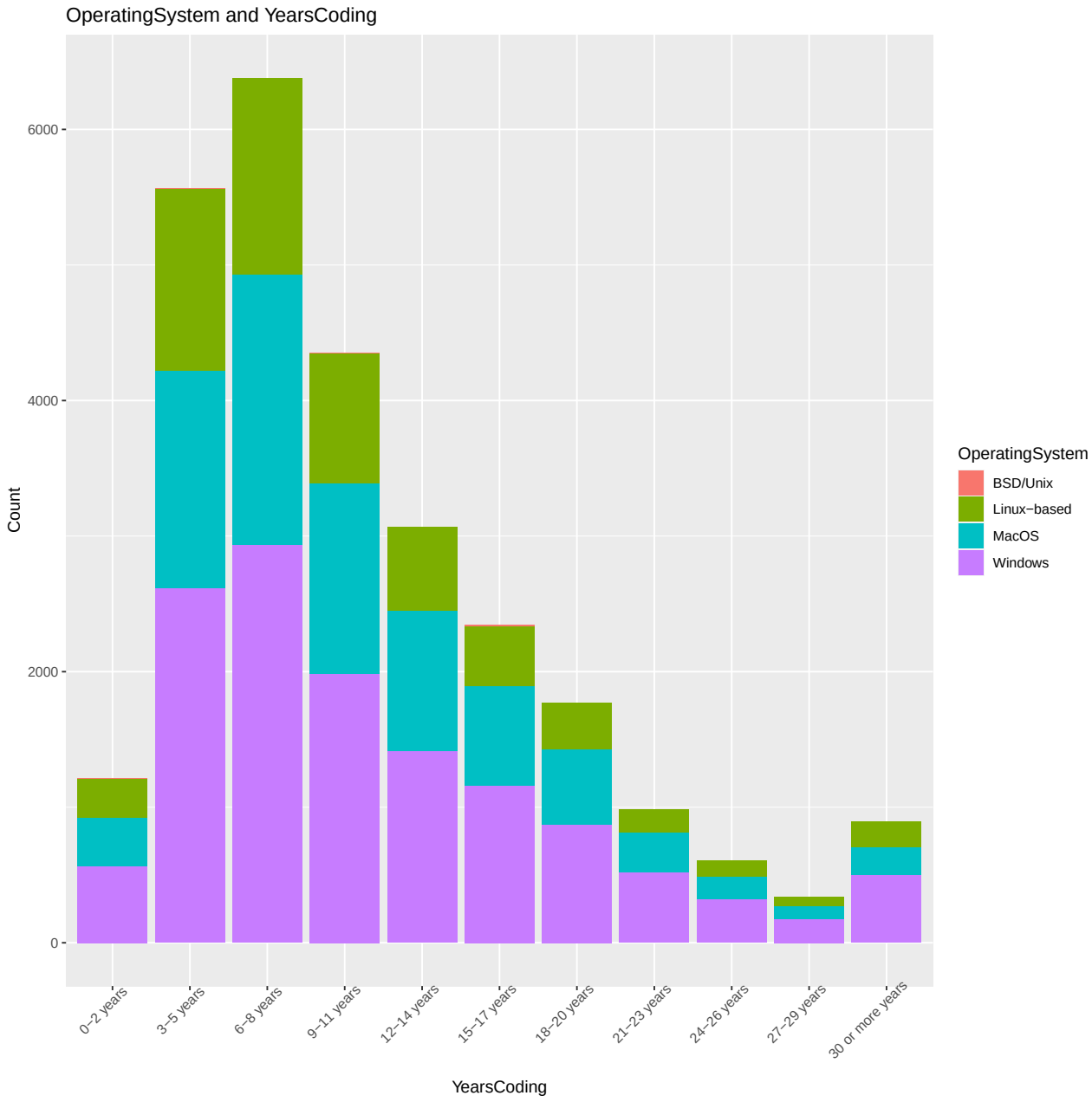
CompanySize vs. OperatingSystem

```
ggplot(dataset, aes(x = CompanySize, fill = OperatingSystem)) +
 geom_histogram(stat="count")+
 labs(x = "CompanySize",
      y = "Count",
      title = "OperatingSystem and CompanySize") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 0.8))
```



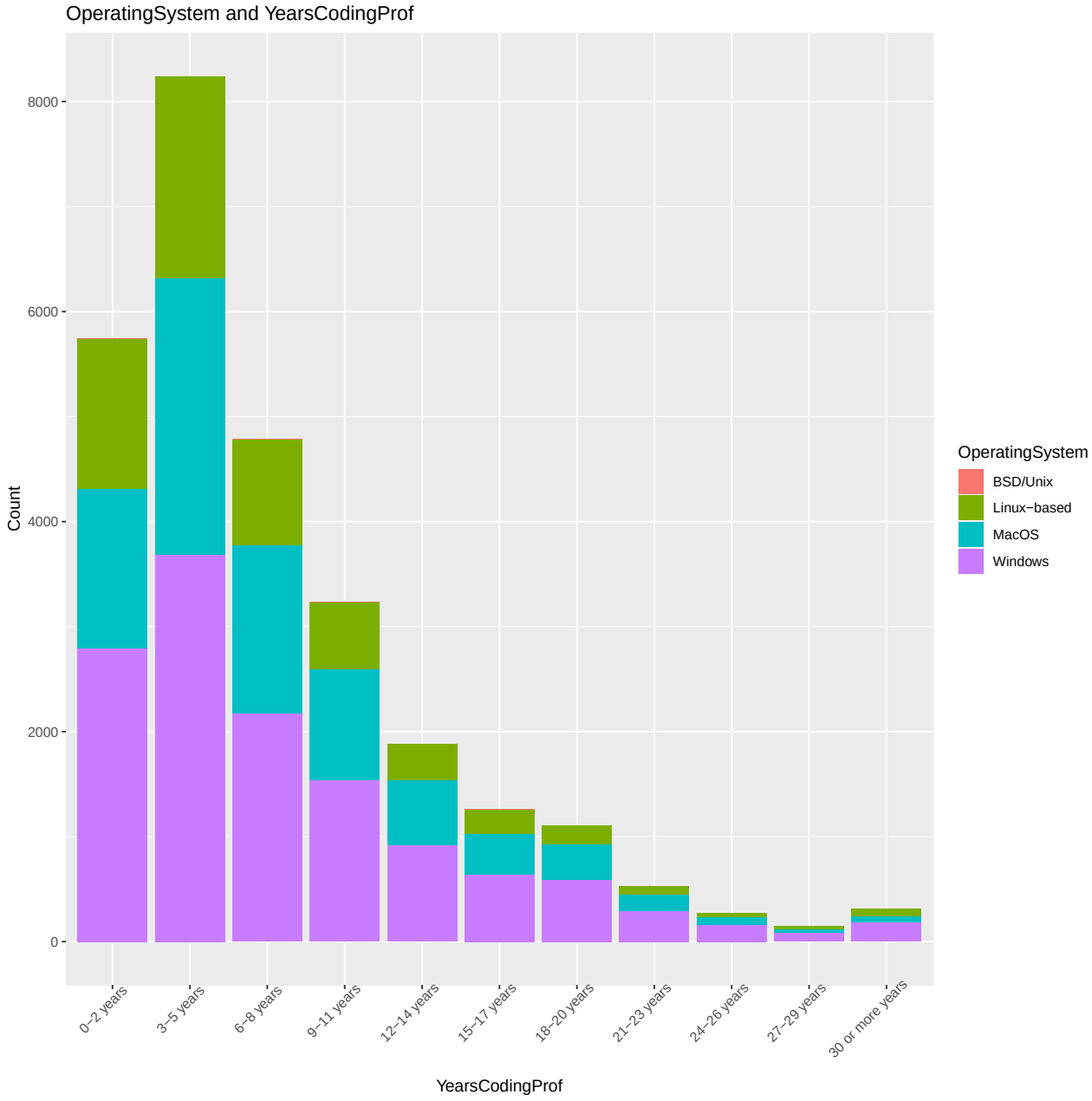
YearsCoding vs. OperatingSystem

```
ggplot(dataset, aes(x = YearsCoding, fill = OperatingSystem)) +
 geom_histogram(stat="count")+
 labs(x = "YearsCoding",
      y = "Count",
      title = "OperatingSystem and YearsCoding") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 0.8))
```



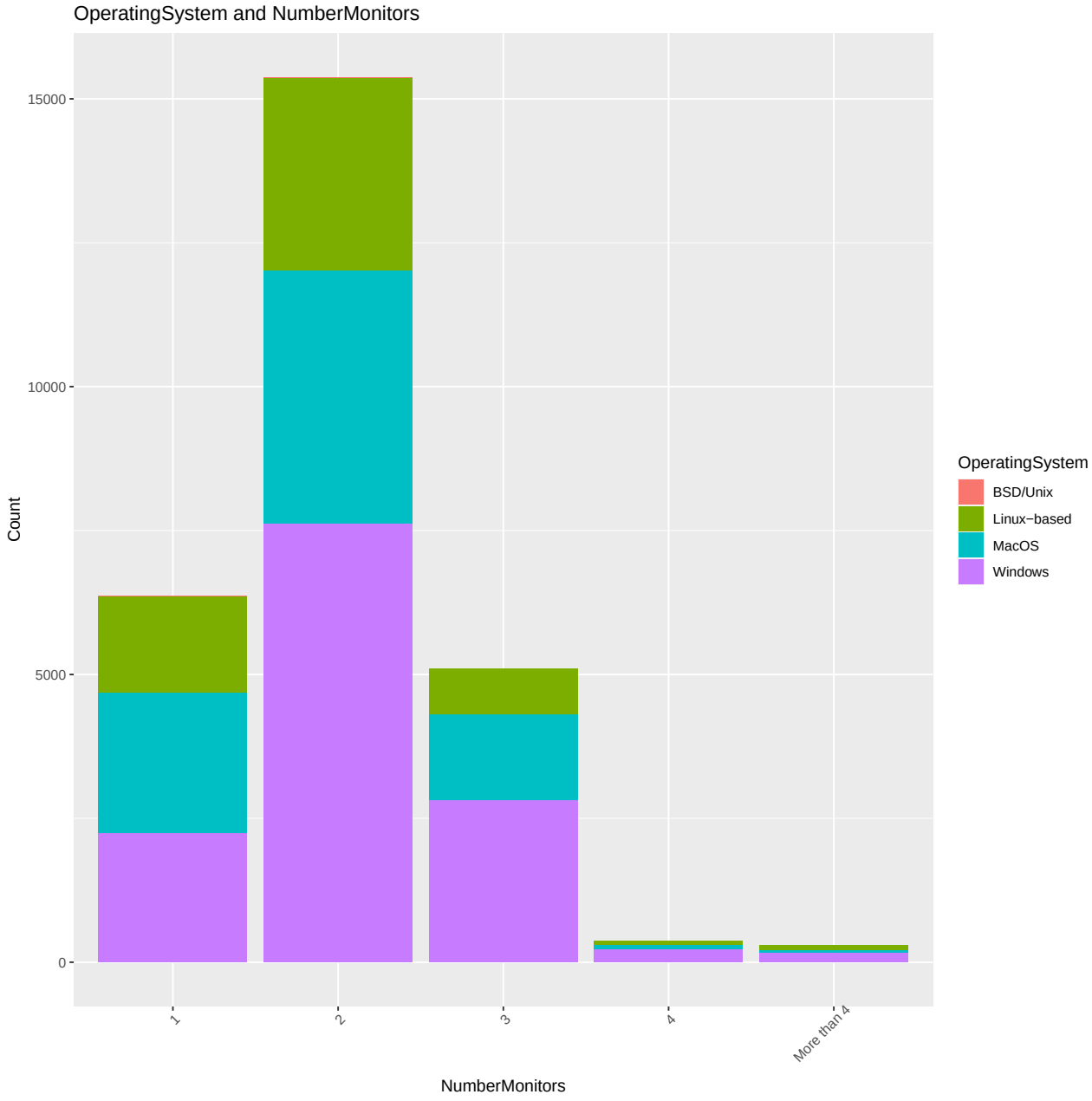
YearsCodingProf vs. OperatingSystem

```
ggplot(dataset, aes(x = YearsCodingProf, fill = OperatingSystem)) +
 geom_histogram(stat="count")+
 labs(x = "YearsCodingProf",
      y = "Count",
      title = "OperatingSystem and YearsCodingProf") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 0.8))
```



NumberMonitors vs OperatingSystem

```
ggplot(dataset, aes(x = NumberMonitors, fill = OperatingSystem)) +
 geom_histogram(stat="count")+
 labs(x = "NumberMonitors",
      y = "Count",
      title = "OperatingSystem and NumberMonitors") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 0.8))
```



Having visualised some of my columns, There are still more left untouched. They are; “DevType”, “LanguageWorkedWith”, “DatabaseWorkedWith”, “FrameworkWorkedWith”, “PlatformWorkedWith”, “IDE”, “Methodology” and “VersionControl”.

These Columns are quite problematic because they are not properly arranged. One column contains more than one variable, separated by a “;”. Below, Let us visualise two rows belonging to one of the columns and see what is happening.

```
head(dataset$DevType, 2)
```

```
## [1] Full-stack developer
## [2] Back-end developer;Database administrator;Front-end developer;Full-stack developer
## 4165 Levels: Back-end developer ...
```

The Second row contains 4 variables namely “Back-end developer”, “Database administrator”, “Front-end developer” and “Full-stack developer”. This means that a particular survey respondent knows those four DevTypes. How do we separate this?

I shall create some functions which will be used to convert columns of such nature to one-hot-encoding.

What is one-hot-encoding?

One-hot-encoding is just a way of representing nominal categorical variables into a continuous variable using zeros and ones.

I shall show you in detail but let us create our function below.

```
do.one.hot <- function(column, separator){
# This function takes a single column of a dataframe and a separator
# (in this case ";") and returns wider column of 1 and 0
# If the column has variables that look like "Java;JavaScript;CSS", it will perform
# one-hot-encoding on that column and return the encoded dataframe.


 variables <- list()
 # we initialise our list which will contain all possible options available under
 # our chosen column.

 columnnew <- data.frame()[1:NROW(column), ]
 # we intialise our columns as well. We are simply saying that we want an empty
 # dataframe with same length as whichever data that is passed into our function.

 for(rows in column){
  # just iterating through the column argument

  row.as.list <- strsplit(rows, toString(separator))[[1]]
  # here, we have converted the row into a list separated by the separator

  for(item in row.as.list){
    # for each item in the list from the row in the column

   if (!(item %in% variables)){
     # if item not in our originally created list called "variable"

    variables <-append(variables, item)
   # here, we append the items in the row.list into our initialised list while
   # eliminating repetitions.
```

```
 }
 }
}
# we now have all the possible variable in the column stored in "variables"


columnnew$Mod <- paste(column, toString(separator), sep="")
# Now, we create a new column appended to our empty initialised column which
# contains exactly the elements we are working with, except that this time, we
# want the rows to terminate with the separator. this is very necessary to
# enable us identify the last word. without doing this, Differentiating
# between "C", "Cobol", "C++", and "C#" will be difficult as "C" is contained
# in them as well. But we can differentiate "C;" from "Cobol;" and "Java;" from
# "JavaScript;" especially when this is the last item.
# including the separator shows where our variable ended.

for(i in variables){
 columnnew[i] <- ifelse(grepl(paste(toString(i), ";", sep=""),
                              columnnew$Mod, fixed = TRUE), 1, 0)
 # Now, I have created new columns to achieve my purpose as defined in my
 # function creation. The grepl will search through each row in the
 # columnnew$Mod and where there is a match, it will create a new column and
 # fill it with 1 otherwise 0.
}

# We return the new columns apart from the first one columnnew$Mod
return(columnnew[, -1])
}
```

I shall now test my newly created function.

```
DevType1 <- do.one.hot(dataset$DevType, ";")
#View(head(DevType1))
```

Now, that we have the biggest data cleaning problem out of the way, let us visualise those columns we left behind. To do that, we convert them back to long format with a function that will return the full data set converted in long format.

```
hot.long <- function(column){
 # This function will convert a mixed column, for example "Java;JavaScript;CSS"
 # to a long column
                        #Java
                        #Javascript
                        #CSS
 #side by side with the corresponding OperatingSystem of the respondent in another column


 #get one hot encoded data and save it as one.hot
 one.hot <- do.one.hot(column, ';')

 # binds os column to one hot encoded. dataset[10] stands for the OperatingSystem
 # column in the original dataset.

 binded <- cbind(dataset[10], one.hot)
```

```
 #initialise lists
 os = list()
 type = list()

 # iterate through the rows
 for(i in 1:nrow(binded)){

  #get name of Operating system per iteration
  os.name = binded[i,1]

  #iterate through the column names
  for(colum in colnames(binded)){

   #avoid the OS column
   if(colum != "OperatingSystem"){

    # any other cell that has the value 1
    if(binded[i, colum] == 1){

     # put name of operating system
    os <- unlist(append(os, toString(os.name) ))

    # put the name of the column
    type <- unlist(append(type, colum))
    }
   }
  }
 }
 #return a dataframe of the two vectors
 return(data.frame(os, type))
}
```

The function below takes about 30 seconds. My computer has intel core i7, 16gb of RAM and a speed of 2.6 gigahertz. Please, be patient with me.

```
start.time <- Sys.time()
devt <- hot.long(dataset$DevType)
#View(devd)
summary(devt)
```

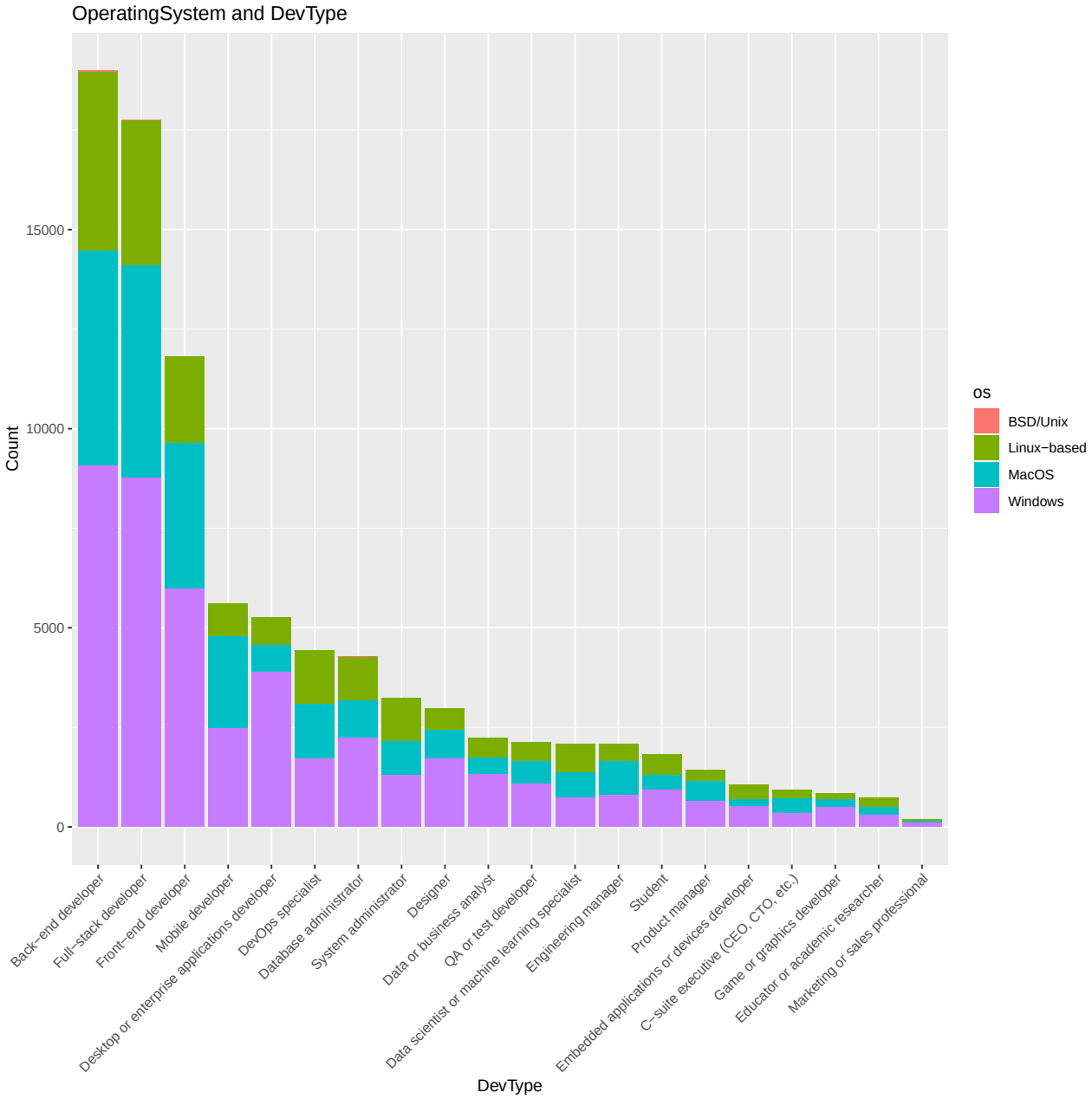
```
##            os                                                  type
##  BSD/Unix   :  110   Back-end developer                           :18990
##  Linux-based:19747   Full-stack developer                        :17749
##  MacOS      :25759   Front-end developer                         :11816
##  Windows    :44250   Mobile developer                            : 5604
##                      Desktop or enterprise applications developer: 5255
##                      DevOps specialist                           : 4436
##                      (Other)                                     :26016
```

```
end.time <- Sys.time()
print(end.time - start.time)
```

```
## Time difference of 35.00213 secs
```

Visualising DevType with OperatingSystem

```
ggplot(devt, aes(x = fct_infreq(type), fill = os)) +
 geom_histogram(stat="count")+
 labs(x = "DevType",
      y = "Count",
      title = "OperatingSystem and DevType") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 1))
```



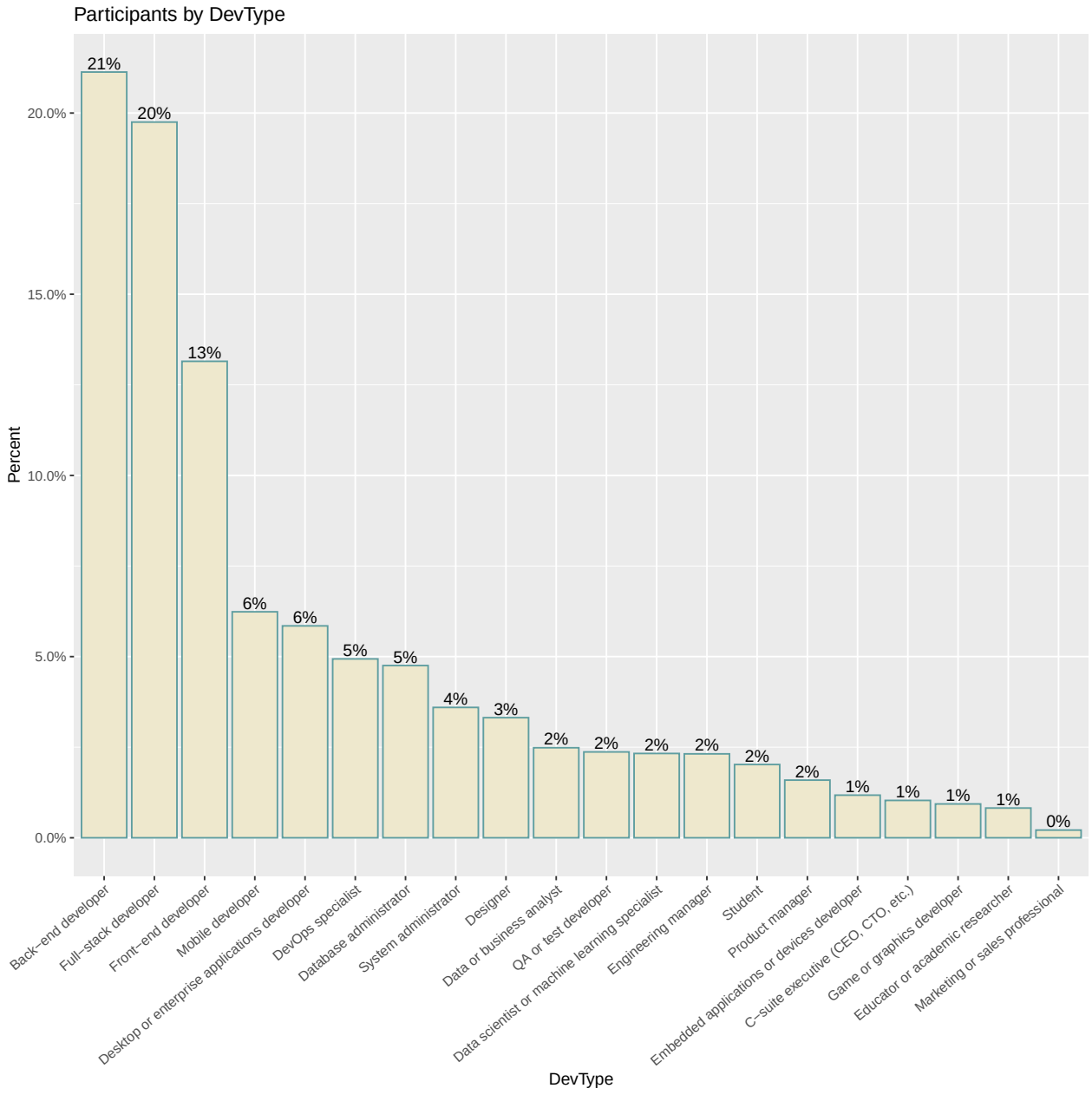
I really love the stacked column charts as they show me exactly what I want to see. However, I know you would want me to plot other different types of graphs. I will need another function that will give the frequencies of each occurrence. This will make it easier to show extra visualisations of my data.

Visualising further:

```
get.freq.table <- function(dataset){
  # this function takes in a one-hot-encoded dataset and return the frequencies
  # of each variable as a dataframe.

  # I will add a new column based on the counts from the parent column




  Type <- list() # we initialise our list where we will store the column names
  Freq <- list() # we initialise another list to store the frequency .

  for (i in colnames(dataset)){
    # we iterate through our column names

    Type <- unlist(append(Type, i))
    # we append the names of the columns to our initialised list
    # we use the unlist() function to make it a vector


    Freq <- unlist(append(Freq, sum(dataset[, i])))
    # we append the frequency of occurence to our Freq list by summing the columns

  }

  return(data.frame(Type, Freq)) # we return the contents of our lists as a dataframe.
}
```

Now, for the rest of our unvisualised columns, We shall utilise the two functions above to do so. These functions will also come in handy when we finally one-hot-encode our whole data before training it using a suiltable machine learning algorithm.

```
DevType1 <- do.one.hot(dataset$DevType, ";")
DevTypeFreq <- get.freq.table(DevType1)

# Let us add a new column to show percentages
PlotDevType <- DevTypeFreq %>%
  mutate(pct  = Freq / sum(Freq), pctlabel = paste0(round(pct*100), "%"))

# we plot the bars as percentages, in descending order with bar labels
ggplot(PlotDevType, aes(x = reorder(Type, -pct),
                        y = pct)) +
  geom_bar(stat = "identity",
           fill = "cornsilk2",
           color = "cadetblue") +
  geom_text(aes(label = pctlabel),
            vjust = -0.25) +
  scale_y_continuous(labels = percent) +
  labs(x = "DevType",
       y = "Percent",
       title  = "Participants by DevType") +
  theme(axis.text.x = element_text(angle = 40,
                                   hjust = 1))
```


Participants by DevType
Percent
20.0%
15.0%
10.0%
5.0%
0.0%
21%
20%
13%
6%
6%
5%
5%
4%
3%
2%
2%
2%
2%
2%
2%
1%
1%
1%
1%
0%
Back−end developer
Full−stack developer
Front−end developer
Mobile developer
Desktop or enterprise applications developer
DevOps specialist
Database administrator
System administrator
Designer
Data or business analyst
QA or test developer
Data scientist or machine learning specialist
Engineering manager
Student
Product manager
Embedded applications or devices developer
C−suite executive (CEO, CTO, etc.)
Game or graphics developer
Educator or academic researcher
Marketing or sales professional
DevType

Visualising LanguageWorkedWith with OperatingSystem

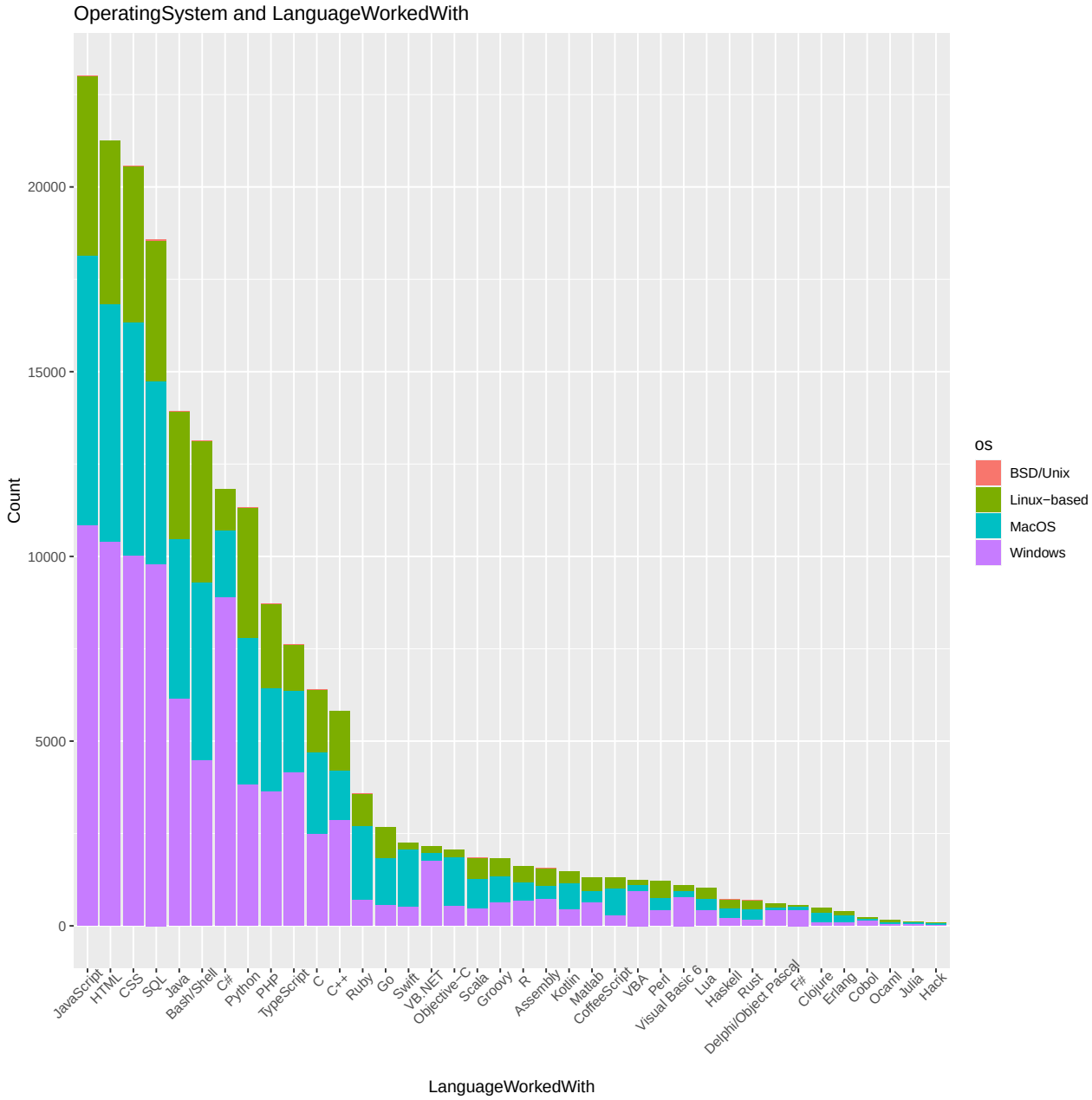
```
LanguageWorkedWith1 <- do.one.hot(dataset$LanguageWorkedWith, ";")
LanguageWorkedWithFreq <- get.freq.table(LanguageWorkedWith1)

# create a treemap with tile labels

ggplot(LanguageWorkedWithFreq,aes(fill = Type, area = Freq, label = Type)) +
  geom_treemap() +  geom_treemap_text(color = "white", place = "centre") +
  labs(title = "Visualising LanguageWorkedWith") + theme(legend.position = "none")
```

Visualising LanguageWorkedWith

My stacked Column Chart: Please be patient with me as this will take time. Took me 2.5 minutes.

Visualising further:

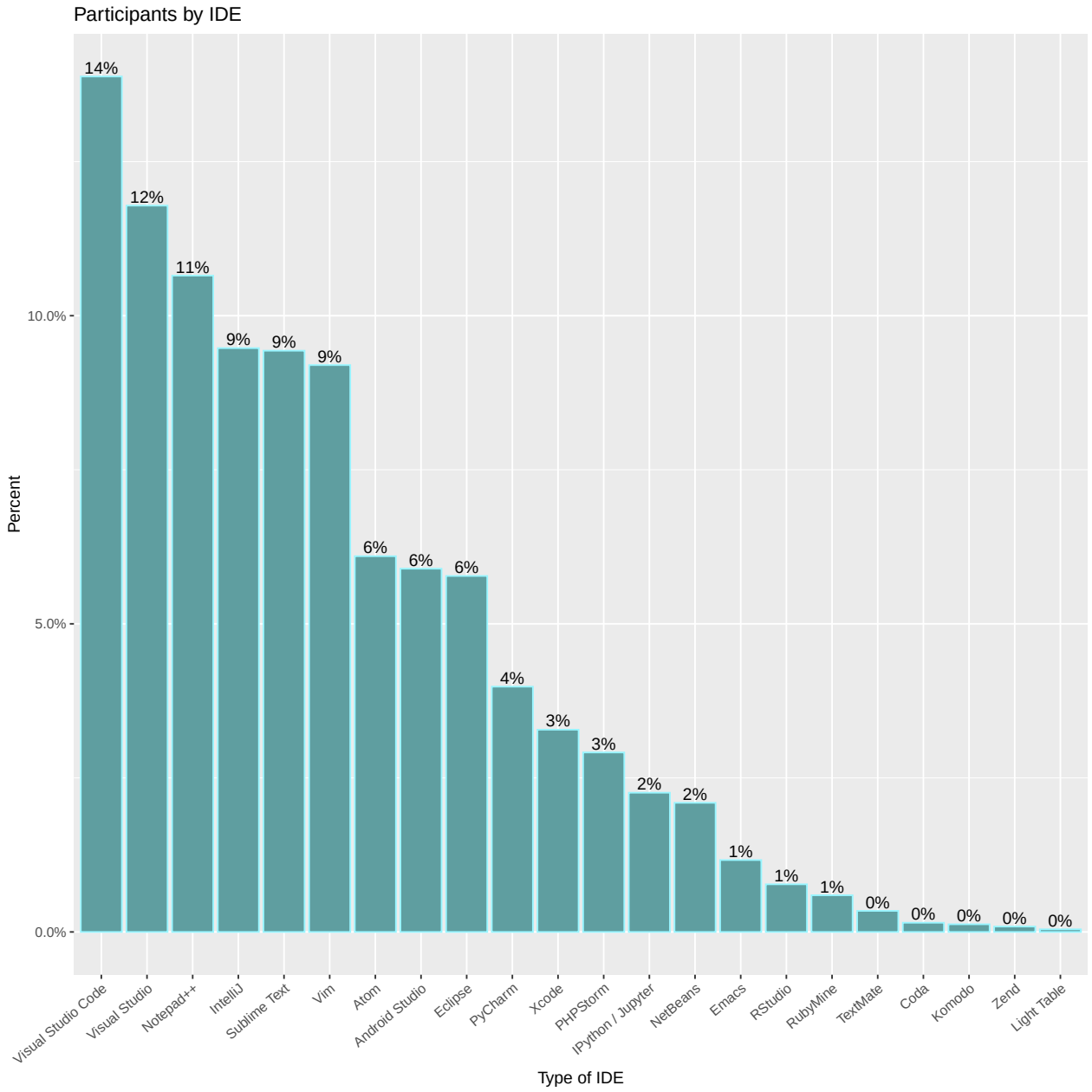
```
start.time <- Sys.time()

lww <- hot.long(dataset$LanguageWorkedWith)
ggplot(lww, aes(x = fct_infreq(type), fill = os)) +
 geom_histogram(stat="count")+
 labs(x = "LanguageWorkedWith",
      y = "Count",
      title = "OperatingSystem and LanguageWorkedWith") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 0.8))
end.time <- Sys.time()
print(end.time - start.time)
```




```
## Time difference of 2.458954 mins
```

Visualising IDE with OperatingSystem

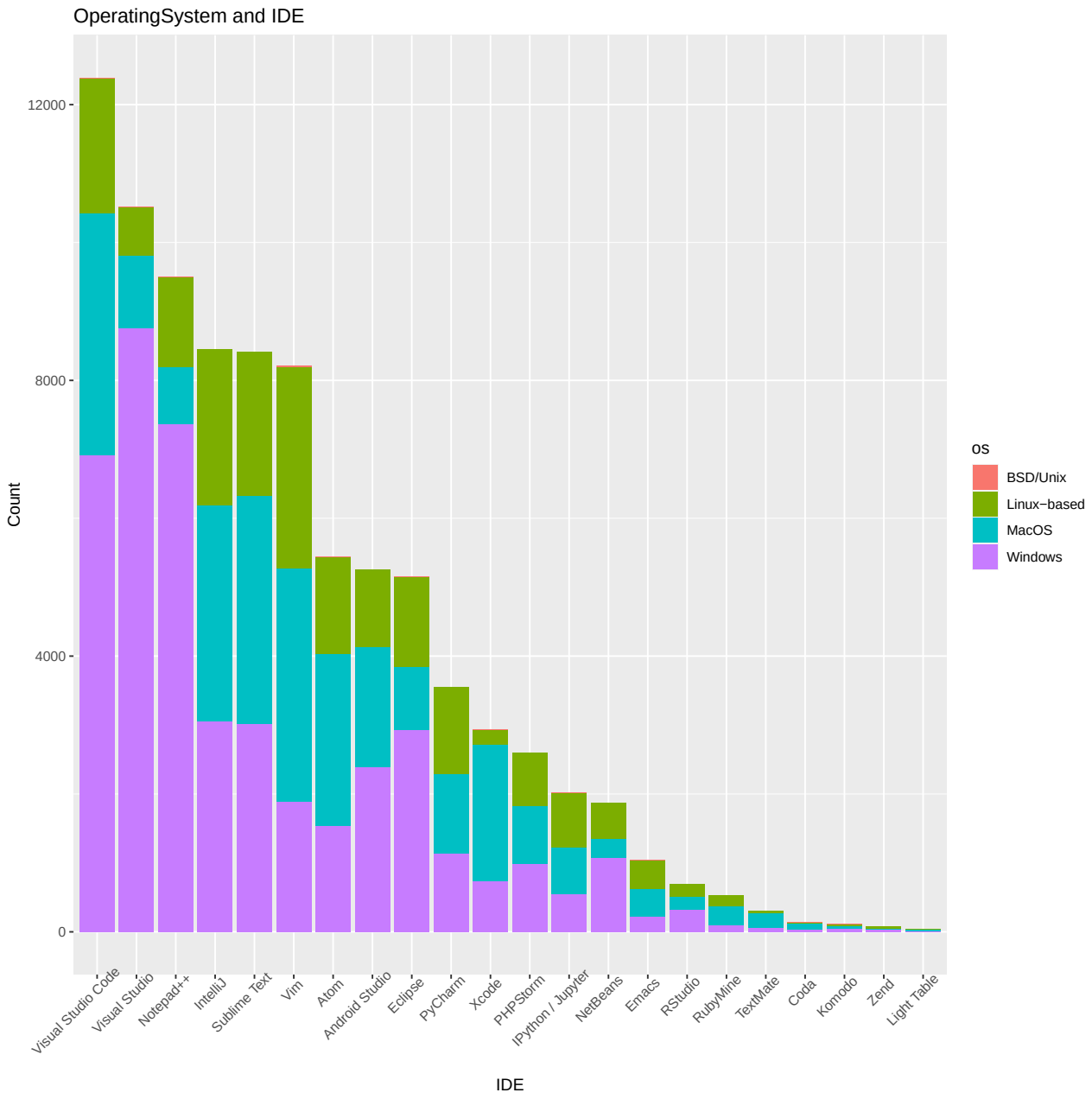
```
IDE1 <- do.one.hot(dataset$IDE, ";")
IDEFreq <- get.freq.table(IDE1)
# Let us add a new column to show percentages
PlotIDE <- IDEFreq %>%
  mutate(pct  = Freq / sum(Freq), pctlabel = paste0(round(pct*100), "%"))

# we plot the bars as percentages, in descending order with bar labels
ggplot(PlotIDE, aes(x = reorder(Type, -pct),
                        y = pct)) +
  geom_bar(stat = "identity",
           fill = "cadetblue",
           color = "cadetblue1") +
  geom_text(aes(label = pctlabel),
            vjust = -0.25) +
  scale_y_continuous(labels = percent) +
  labs(x = "Type of IDE", y = "Percent", title  = "Participants by IDE")+
  theme(axis.text.x = element_text(angle = 40,
                                   hjust = 1))
```



What a pity! My favorite rstudio has just 1%.

Visualising further:

```
start.time <- Sys.time()
ide <- hot.long(dataset$IDE)
ggplot(ide, aes(x = fct_infreq(type), fill = os)) +
 geom_histogram(stat="count")+
 labs(x = "IDE",
      y = "Count",
      title = "OperatingSystem and IDE") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 0.8))
```



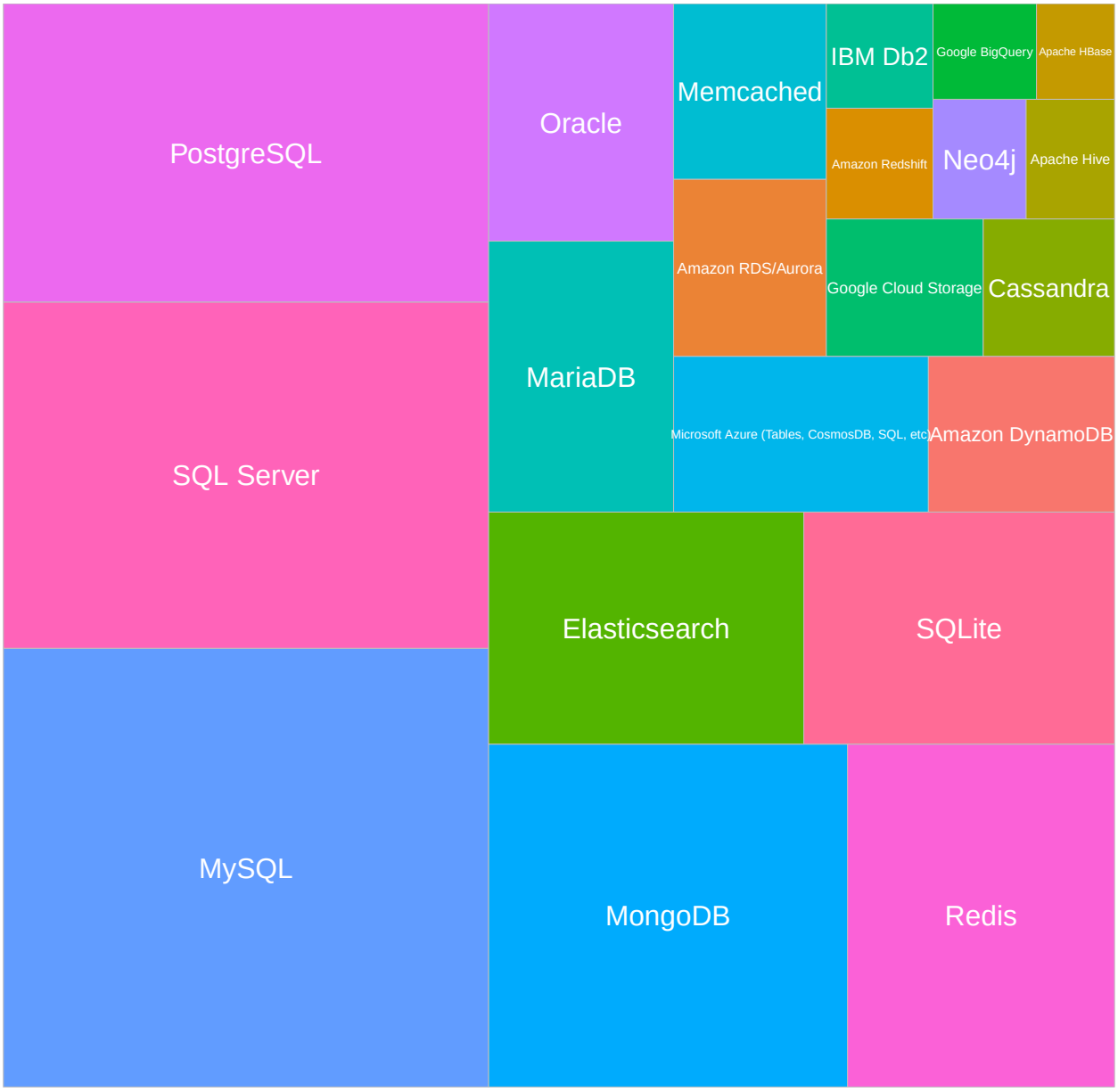

```
end.time <- Sys.time()
print(end.time - start.time)
```

```
## Time difference of 35.70292 secs
```

Visualising DatabaseWorkedWith

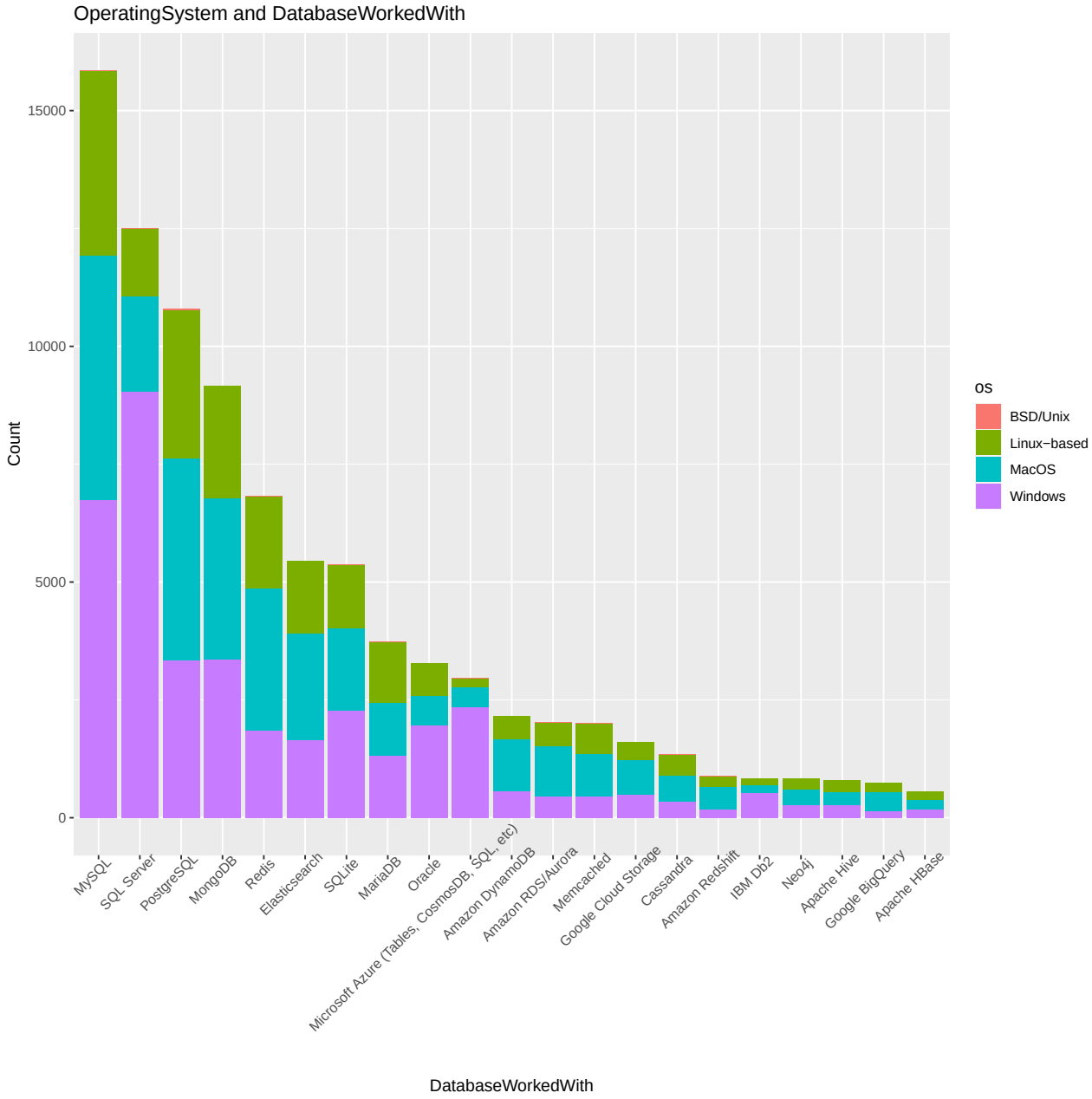
```
DatabaseWorkedWith1 <- do.one.hot(dataset$DatabaseWorkedWith, ";")
DatabaseWorkedWithFreq <- get.freq.table(DatabaseWorkedWith1)
# create a treemap with tile labels

ggplot(DatabaseWorkedWithFreq,aes(fill = Type, area = Freq, label = Type)) +
  geom_treemap() +  geom_treemap_text(color = "white", place = "centre") +
  labs(title = "Visualising DatabaseWorkedWith") + theme(legend.position = "none")
```

Visualising DatabaseWorkedWith



Visualising further:

```
dbww <- hot.long(dataset$DatabaseWorkedWith)
ggplot(dbww, aes(x = fct_infreq(type), fill = os)) +
 geom_histogram(stat="count")+
 labs(x = "DatabaseWorkedWith",
      y = "Count",
      title = "OperatingSystem and DatabaseWorkedWith") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 0.8))
```

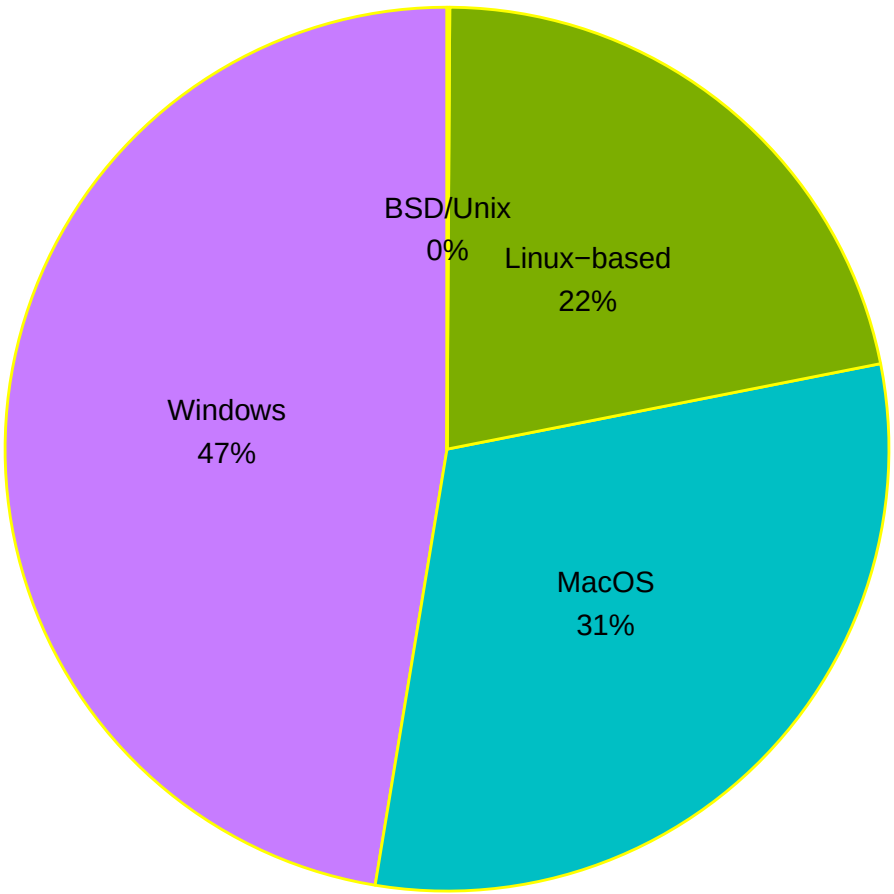


Visualising OperatingSystem

```
OperatingSystem1 <- do.one.hot(dataset$OperatingSystem, ";")
OperatingSystemFreq <- get.freq.table(OperatingSystem1)
```

```r
# create a pie  chart with slice labels and percentages
PlotOperatingSystemFreq <- OperatingSystemFreq %>%
   arrange(desc(Type)) %>%
   mutate(prop = round(Freq * 100 / sum(Freq), 1),
          lab.ypos = cumsum(prop) - 0.5 * prop)

# we append our percentages to the names
PlotOperatingSystemFreq$label <- paste0(PlotOperatingSystemFreq$Type, "\n",
                                        round(PlotOperatingSystemFreq$prop),"%")

# we plot
ggplot(PlotOperatingSystemFreq, aes(x = "", y = prop, fill = Type)) +
  geom_bar(width = 1, stat = "identity", color = "yellow") +
  geom_text(aes(y = lab.ypos, label = label), color = "black") +
  coord_polar("y",start = 0,  direction = -1) +
  theme_void() +
  theme(legend.position = "TRUE") +
  labs(title = "Participants by Operating System")
```

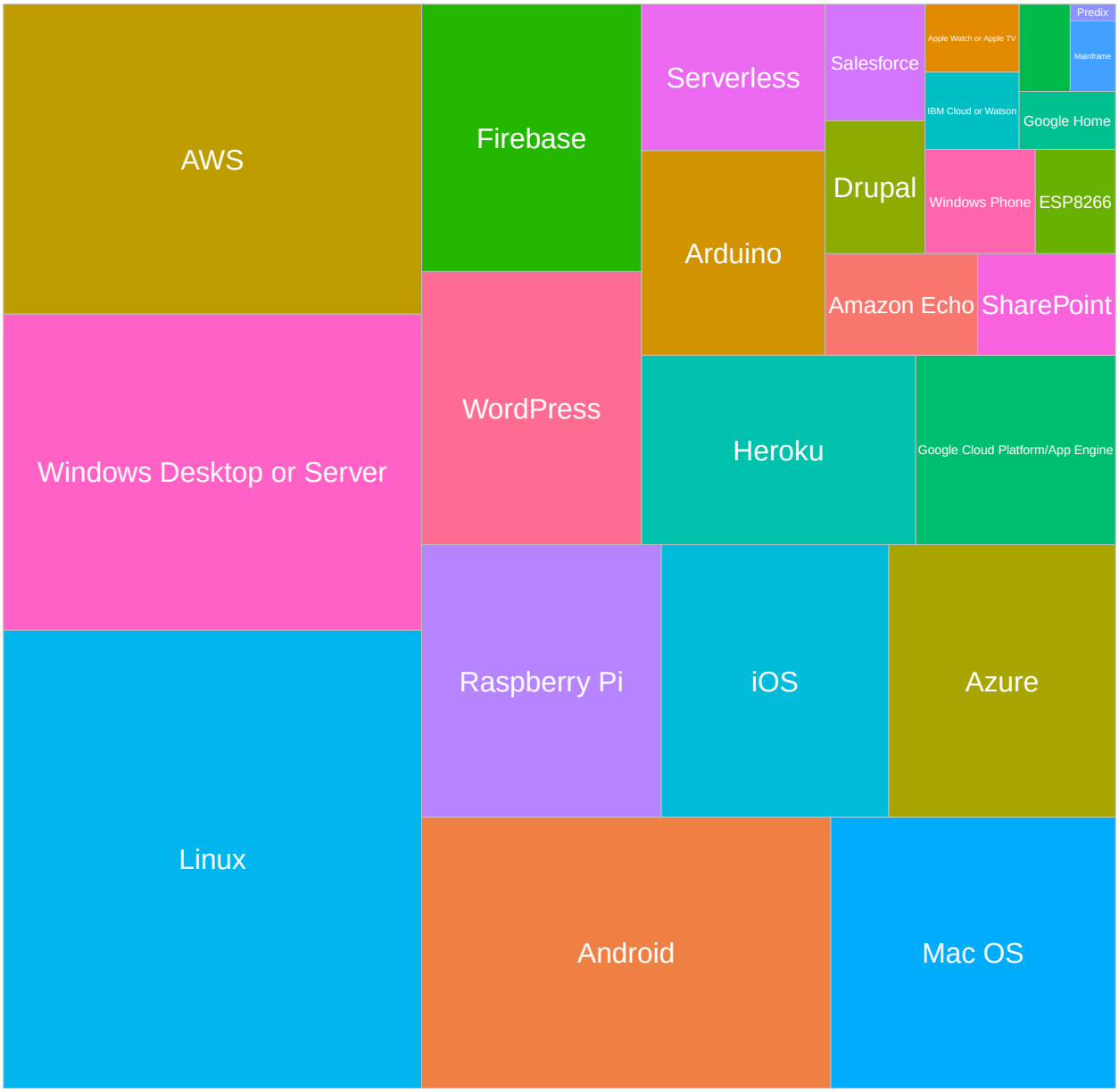
Participants by Operating System



Visualising PlatformWorkedWith

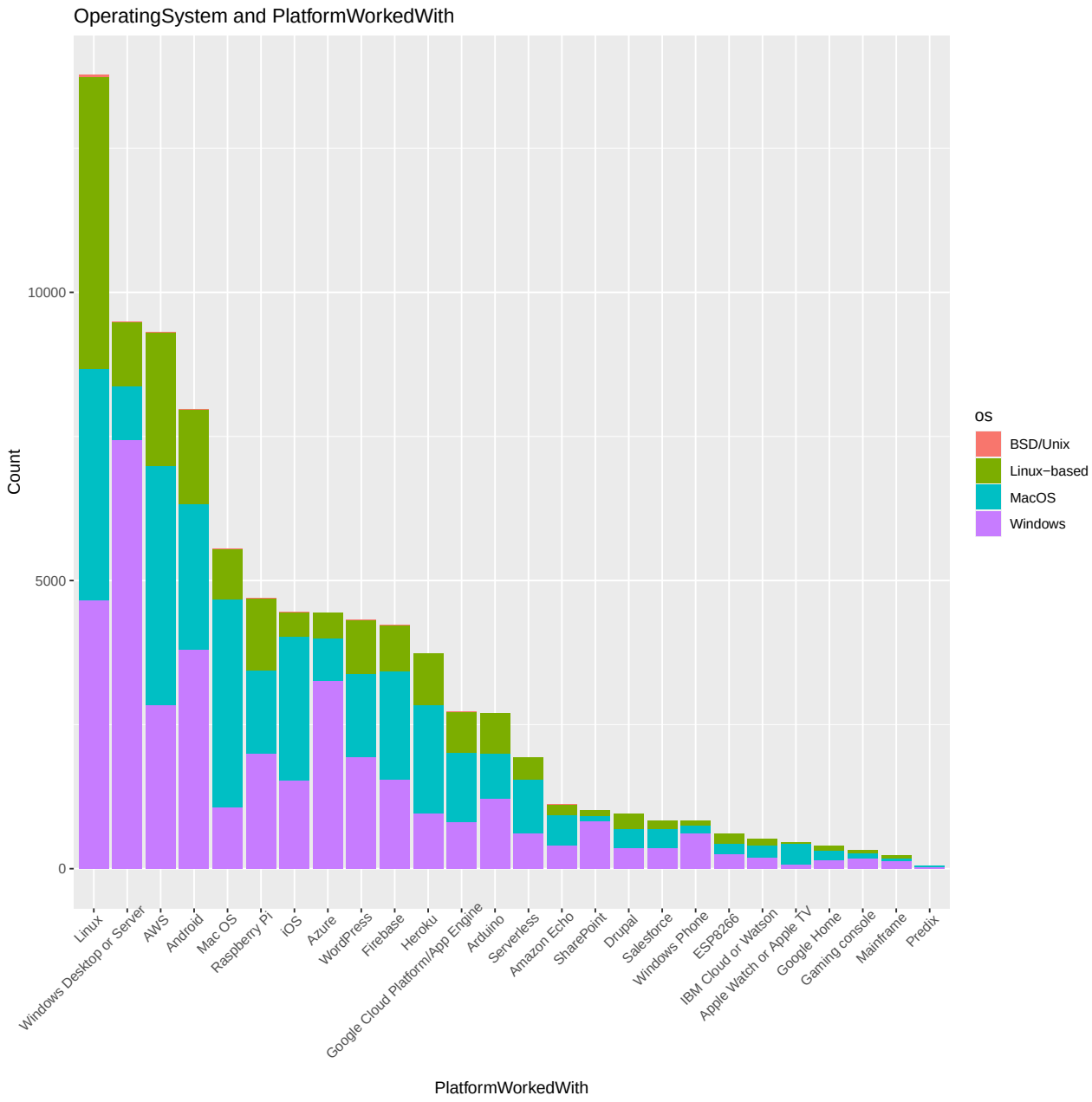
```
PlatformWorkedWith1 <- do.one.hot(dataset$PlatformWorkedWith, ";")
PlatformWorkedWithFreq <- get.freq.table(PlatformWorkedWith1)
# create a treemap with tile labels

ggplot(PlatformWorkedWithFreq,aes(fill = Type, area = Freq, label = Type)) +
  geom_treemap() +  geom_treemap_text(color = "white", place = "centre") +
  labs(title = "Visualising PlatformWorkedWith") + theme(legend.position = "none")
```

Visualising PlatformWorkedWith



Visualising further:

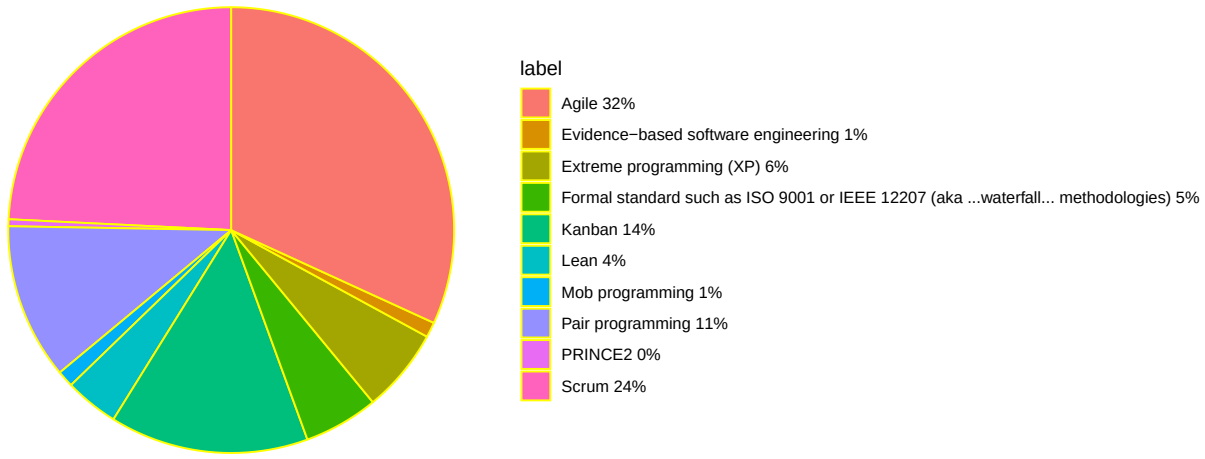
```
pww <- hot.long(dataset$PlatformWorkedWith)
ggplot(pww, aes(x = fct_infreq(type), fill = os)) +
 geom_histogram(stat="count")+
 labs(x = "PlatformWorkedWith",
      y = "Count",
      title = "OperatingSystem and PlatformWorkedWith") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 0.9))
```



Visualising Methodology

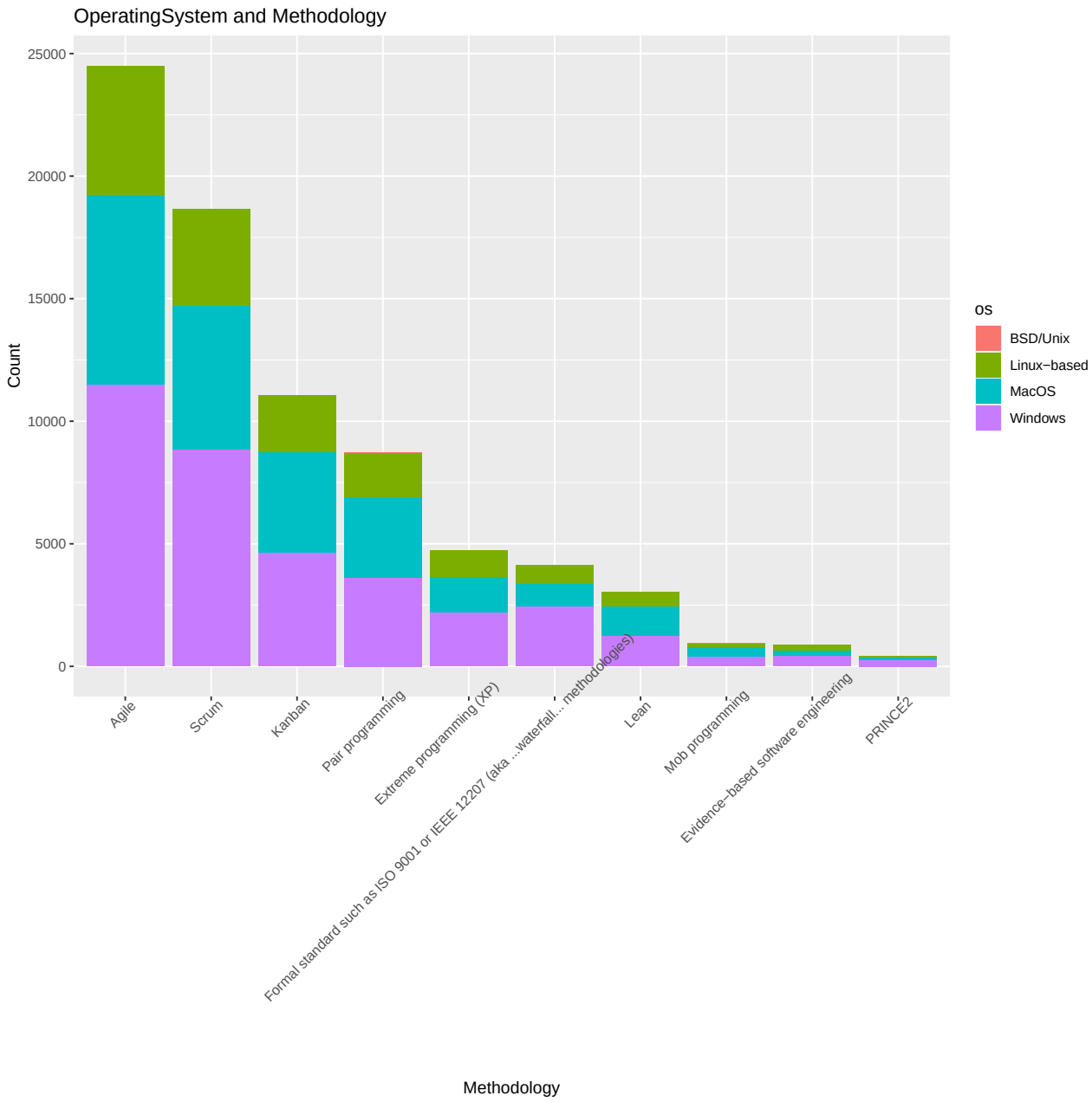
```
Methodology1 <- do.one.hot(dataset$Methodology, ";")
MethodologyFreq <- get.freq.table(Methodology1)

# create a pie  chart with slice labels and percentages
PlotMethodologyFreq <- MethodologyFreq %>%
   arrange(desc(Type)) %>%
   mutate(prop = round(Freq * 100 / sum(Freq), 1),
          lab.ypos = cumsum(prop) - 0.5 * prop)

# we append our percentages to the names
PlotMethodologyFreq$label <- paste0(PlotMethodologyFreq$Type, " ",
                                    round(PlotMethodologyFreq$prop),"%  ")

# we plot
ggplot(PlotMethodologyFreq, aes(x = "", y = prop, fill = label)) +
  geom_bar(width = 1, stat = "identity", color = "yellow") +
  coord_polar("y",start = 0,  direction = -1) +
  theme_void() +
  labs(title = "Participants by Methodology")
```

Participants by Methodology



Visualising further:

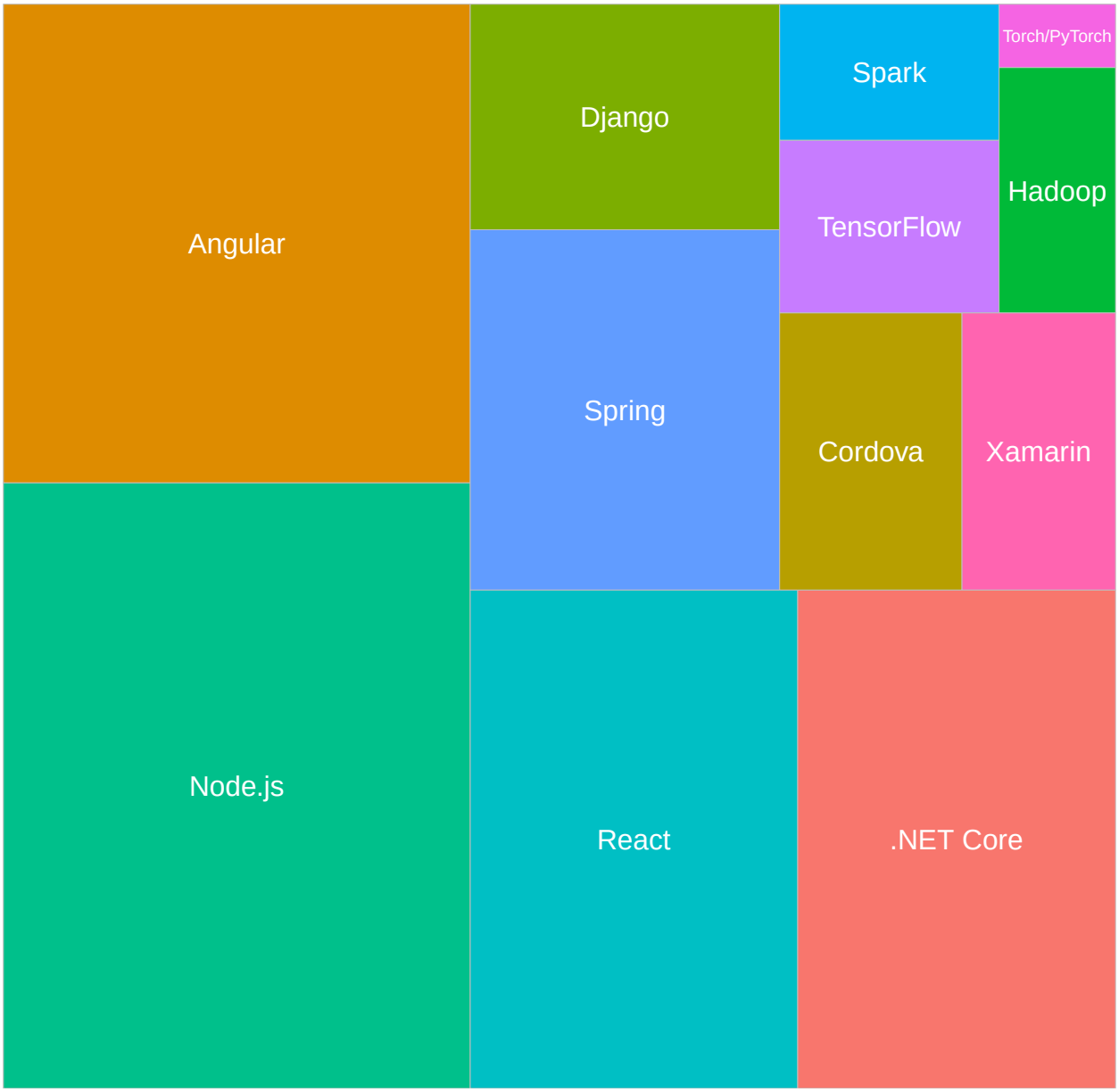
```
methodologyy <- hot.long(dataset$Methodology)
ggplot(methodologyy, aes(x = fct_infreq(type), fill = os)) +
 geom_histogram(stat="count")+
 labs(x = "Methodology",
      y = "Count",
      title = "OperatingSystem and Methodology") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 0.8))
```



Visualising FrameworkWorkedWith

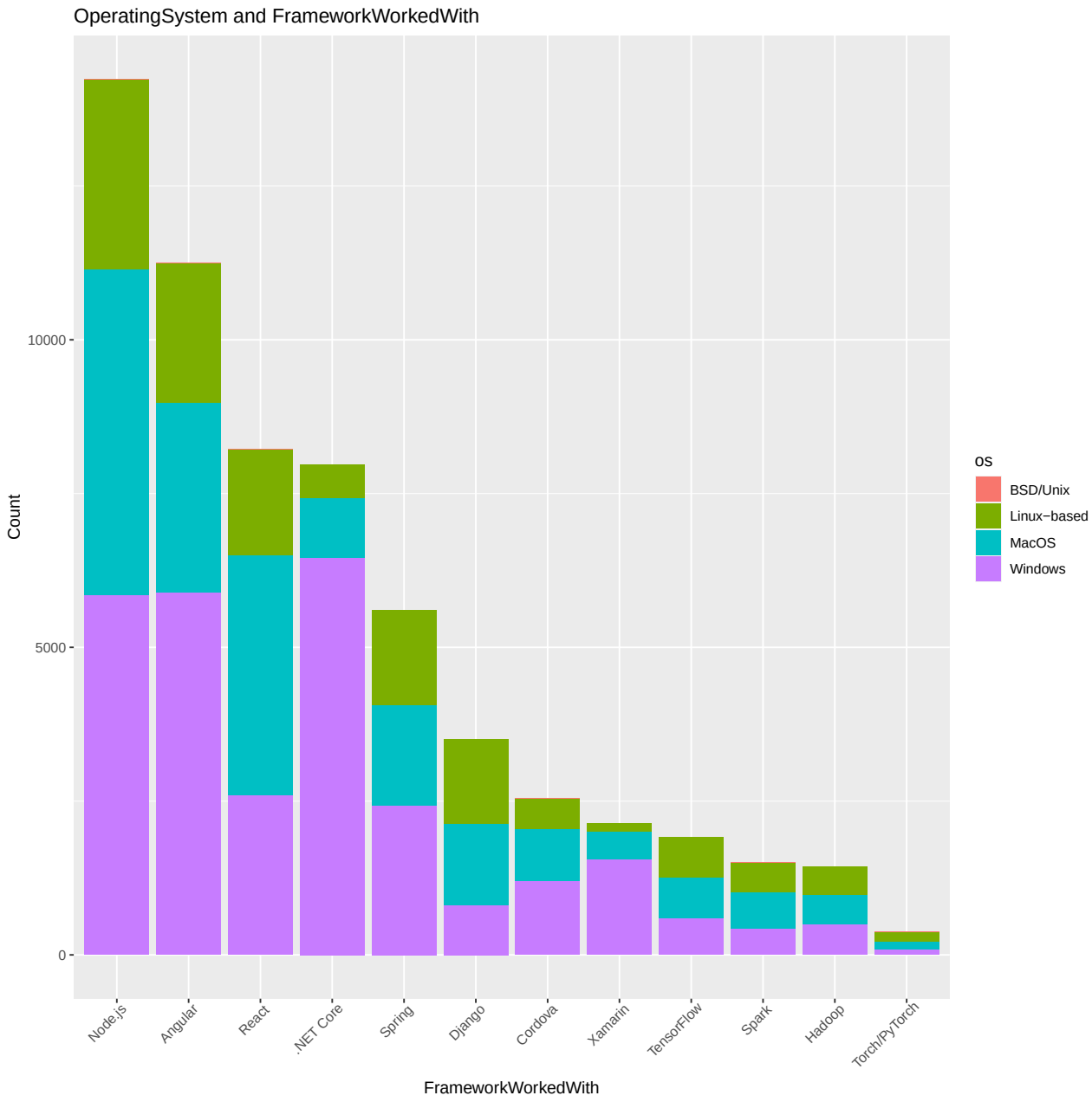
```
FrameworkWorkedWith1 <- do.one.hot(dataset$FrameworkWorkedWith, ";")
FrameworkWorkedWithFreq <- get.freq.table(FrameworkWorkedWith1)
# create a treemap with tile labels

ggplot(FrameworkWorkedWithFreq,aes(fill = Type, area = Freq, label = Type)) +
  geom_treemap() +  geom_treemap_text(color = "white", place = "centre") +
  labs(title = "Visualising FrameworkWorkedWith") + theme(legend.position = "none")
```

Visualising FrameworkWorkedWith



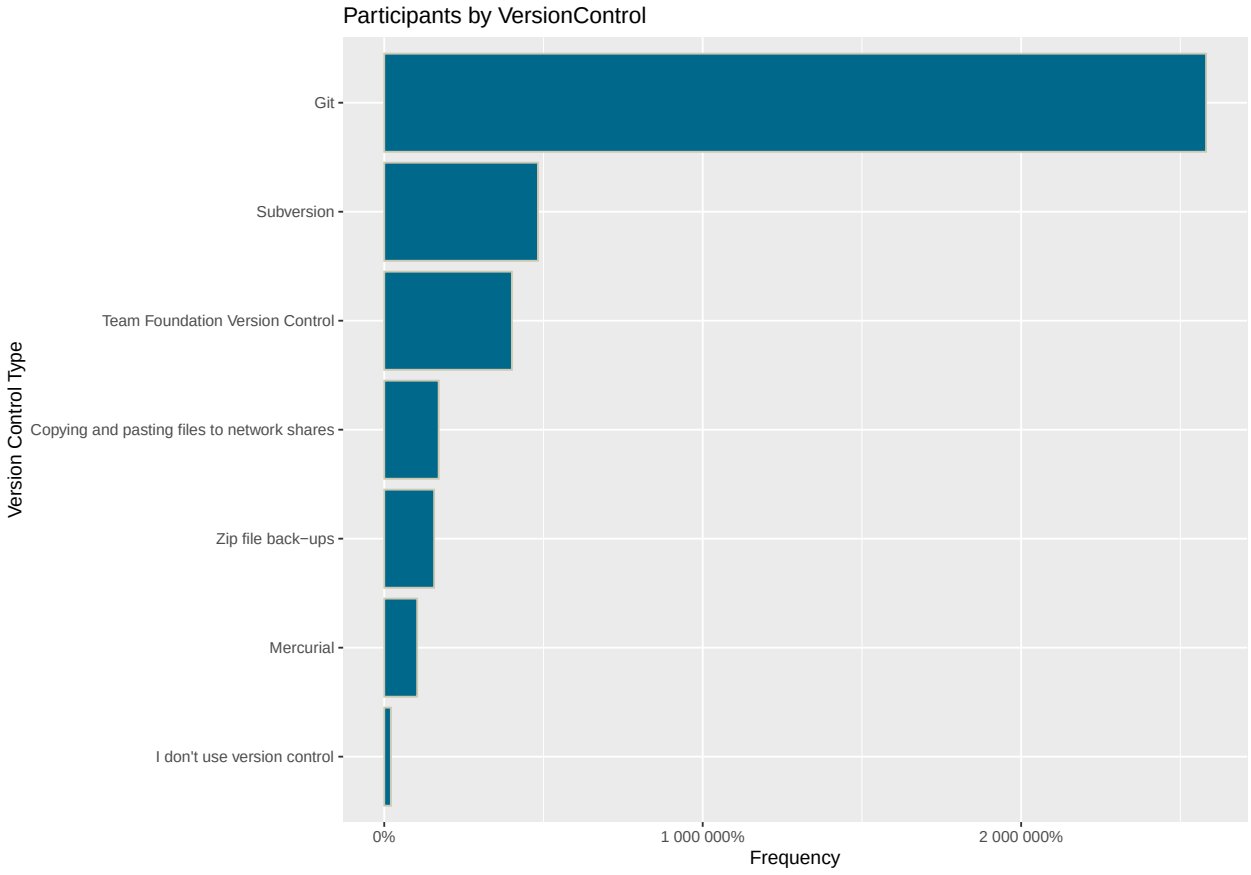
Visualising further:

```
fww <- hot.long(dataset$FrameworkWorkedWith)
ggplot(fww, aes(x = fct_infreq(type), fill = os)) +
 geom_histogram(stat="count")+
 labs(x = "FrameworkWorkedWith",
y = "Count",
title = "OperatingSystem and FrameworkWorkedWith") +
 theme(axis.text.x = element_text(angle = 45,
hjust = 0.9))
```



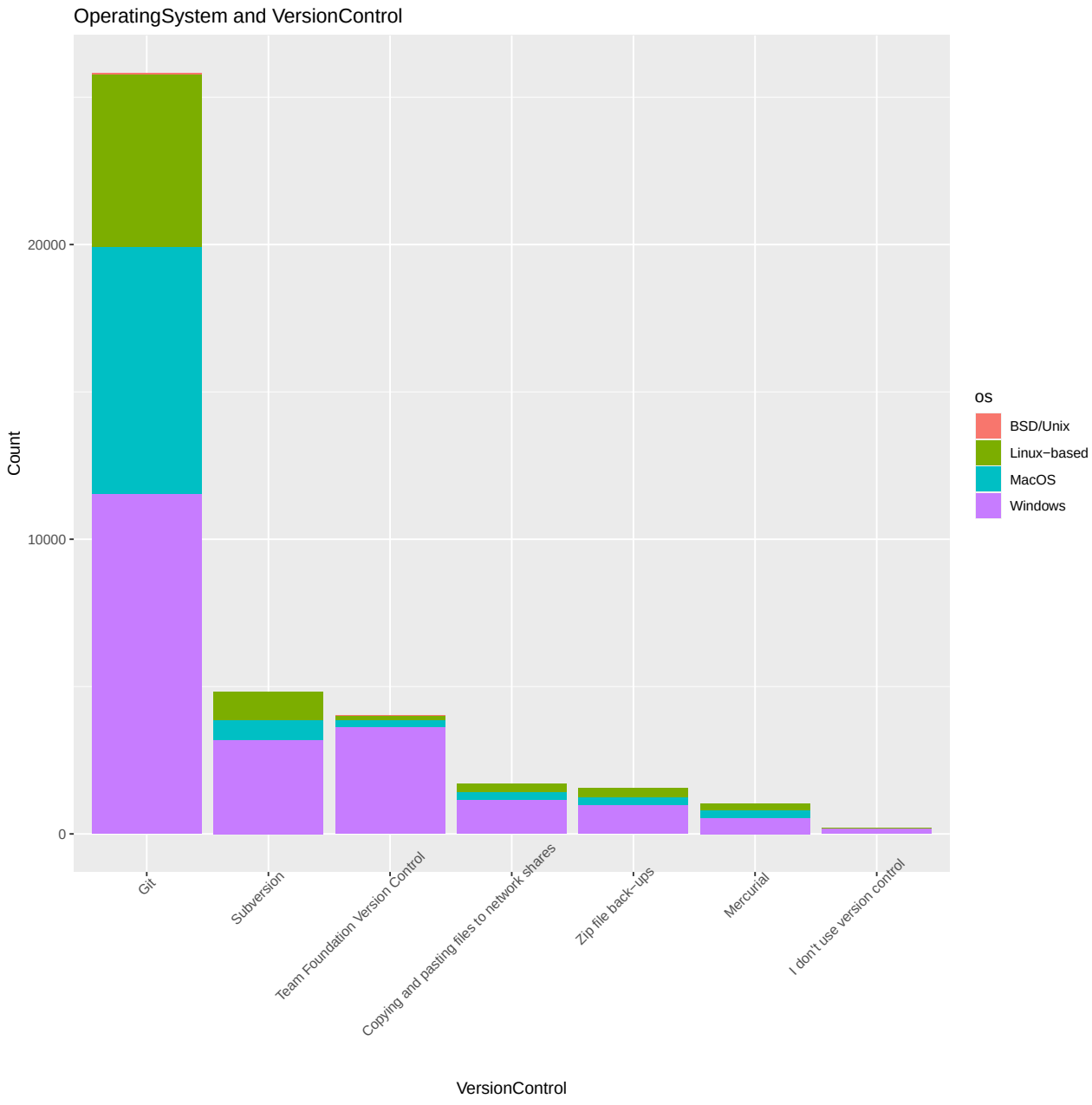
Visualising VersionControl of choice

```
VersionControl1 <- do.one.hot(dataset$VersionControl, ";")
VersionControlFreq <- get.freq.table(VersionControl1)

# Let us add a new column to show percentages
PlotVersionControl <- VersionControlFreq %>%
  mutate(pct  = Freq / sum(Freq), pctlabel = paste0(round(pct*100), "%"))

# we plot the bars as percentages, in descending order with bar labels
ggplot(VersionControlFreq, aes(x = reorder(Type, Freq),
                        y = Freq)) +
  geom_bar(stat = "identity",
           fill = "deepskyblue4",
           color = "cornsilk3") +
  scale_y_continuous(labels = percent) +
  labs(x = "Version Control Type", y = "Frequency", title  = "Participants by VersionControl")+
  coord_flip()
```



Visualising further:

```
vc <- hot.long(dataset$VersionControl)
ggplot(vc, aes(x = fct_infreq(type), fill = os)) +
 geom_histogram(stat="count")+
 labs(x = "VersionControl",
      y = "Count",
      title = "OperatingSystem and VersionControl") +
 theme(axis.text.x = element_text(angle = 45,
                                  hjust = 0.8))
```



Machine Learning

Now that we are visualising our data, I shall now prepare the data to make it suitable for machine learning.

Preparing our table for Machine Learning.

There are a handful of powerful machine learning algorithms. However, to make the best use of these algorithms, it is very important that we transform the data into the desired format. One of the common steps for doing this is encoding the data, which enhances the computational power and the efficiency of the algorithms.

To prepare our table for machine learning, I would like to examine my choices. My Target variable (OperatingSystem) which I want to predict is a categorical variable as well as the rest of my variables. To prepare my data, I am going to categorise them into 2. The Ordinal variables and The Nominal variables.

My Ordinal variables are those variable with ordered factor. They are CompanySize, YearsCoding, YearsCodingProf, NumberMonitors. I am going to leave them the way they are.

The Nominal variables are my variables which do not have order. The rest of my data are nominal. With my nominal data, I will use one hot encoding to encode them. I have addressed this technique earlier and I also created a function for that.

Now, let us encode the column that contains the Nominal data. We will begin by creating a function in order to eliminate code repetition. We will look at those columns that are formatted in a terrible but specific way such as the "DevType" column. Let us look at the first 2 rows.

```
print(head(dataset$DevType, 2))
```

```
## [1] Full-stack developer
## [2] Back-end developer;Database administrator;Front-end developer;Full-stack developer
## 4165 Levels: Back-end developer ...
```

I will use the function "do.one.hot" which I initially created during my visualisation process to encode my dataset. Here is a function to encode, combine, remove column and return a one hot encoded dataset.

```
# define the funtion parameters
One.hotter <- function(dataset, column, dataset.with.column){

  # one hot encoding
  onehot <- do.one.hot(column = dataset.with.column, separator = ";")

  # add the encoded column to the pre-existing data
  new.data <- cbind(dataset, onehot)

  # remove the column that has just been one hot encoded.
  new.data <- select(new.data, -c(toString(column)))

  # return the original dataset but with the selected column converted to one hot
  # encoding
  return(new.data)
}
```

Now, let us encode the rest of our data.

```
dataset <- One.hotter(dataset = dataset,
                      column = "DevType",
                      dataset.with.column = dataset$DevType)
dataset <- One.hotter(dataset = dataset,
                      column = "LanguageWorkedWith",
```

```
                        dataset.with.column = dataset$LanguageWorkedWith)
dataset <- One.hotter(dataset = dataset,
                        column = "FrameworkWorkedWith",
                        dataset.with.column = dataset$FrameworkWorkedWith)
dataset <- One.hotter(dataset = dataset,
                       column = "DatabaseWorkedWith",
                       dataset.with.column = dataset$DatabaseWorkedWith)
dataset <- One.hotter(dataset = dataset,
                        column = "PlatformWorkedWith",
                        dataset.with.column = dataset$PlatformWorkedWith)
dataset <- One.hotter(dataset = dataset,
                       column = "IDE",
                       dataset.with.column = dataset$IDE)
dataset <- One.hotter(dataset = dataset,
                       column = "Methodology",
                       dataset.with.column = dataset$Methodology)
dataset <- One.hotter(dataset = dataset,
                       column = "VersionControl", dataset.with.column = dataset$VersionControl)
```

```
#View(dataset)
```

Now we have 161 columns of data and the one hot encoded columns are all numeric. I want them to be factors as they will give me better prediction since this is a classification problem.

Make all one hot encoded columns factors

```
f.dataset <- dataset %>% mutate_if(is.numeric, as.factor)
```

check

```
str(f.dataset)
```

```
## 'data.frame':    27516 obs. of  161 variables:
##  $ CompanySize                                                          :
 ↪  Ord.factor w/ 8 levels "Fewer than 10 employees"<..: 3 2 8 2 4 3 2 6 5 3 ...
##  $ YearsCoding                                                          :
 ↪  Ord.factor w/ 11 levels "0-2 years"<"3-5 years"<..: 2 3 4 1 11 2 3 6 9 4 ...
##  $ YearsCodingProf                                                      :
 ↪  Ord.factor w/ 11 levels "0-2 years"<"3-5 years"<..: 2 2 1 2 8 2 1 5 9 3 ...
##  $ OperatingSystem                                                      :
 ↪  Factor w/ 4 levels "BSD/Unix","Linux-based",..: 2 2 3 4 3 2 4 3 3 4 ...
##  $ NumberMonitors                                                       :
 ↪  Ord.factor w/ 5 levels "1"<"2"<"3"<"4"<..: 1 2 2 1 1 5 1 1 3 2 ...
##  $ Full-stack developer                                                 :
 ↪  Factor w/ 2 levels "0","1": 2 2 2 1 2 2 1 2 1 2 ...
##  $ Back-end developer                                                   :
 ↪  Factor w/ 2 levels "0","1": 1 2 2 1 2 2 2 1 2 2 ...
##  $ Database administrator                                               :
 ↪  Factor w/ 2 levels "0","1": 1 2 1 1 2 1 1 2 1 2 ...
##  $ Front-end developer                                                  :
 ↪  Factor w/ 2 levels "0","1": 1 2 2 2 1 1 1 1 1 2 ...
##  $ Designer                                                             :
 ↪  Factor w/ 2 levels "0","1": 1 1 1 2 1 1 1 1 1 2 ...
##  $ QA or test developer                                                 :
 ↪  Factor w/ 2 levels "0","1": 1 1 1 2 1 1 1 1 2 1 ...
```

```
##   [list output truncated]
```

161 is a large number of columns. I still need to reduce it be removing columns with near zero variance from my dataset. Columns with near zero variance will contribute little or nothing to my overall performance and removing them will reduce my training time as there will be fewer columns to work with. Remove zero variance

```
# if there are actually non zero variance in my data
if (length(nearZeroVar(f.dataset)) > 0) {

 # remove it.
  f.dataset <- f.dataset[, -nearZeroVar(f.dataset)]
}
print(dim(f.dataset))
```

```
## [1] 27516   107
```

Now our 161 columns have been reduced to 107 without loosing much information in our data. We will now handle the illegal column names and name them properly otherwise, It will be impossible to feed in the data into a machine learning algorithm without having legal names.

With illegal names containing negative signs in-between, or positive signs (as with C++) I get the error. "Error in eval(predvars, data, env) : object 'Back-end developer' not found" This is accomplished using the "clean_names()" function from the Janitor package and "make.names()" from base R.

```
summary(f.dataset$OperatingSystem)
```

```
##    BSD/Unix Linux-based       MacOS     Windows
##          35        5993        8456       13032
```

```
f.dataset$OperatingSystem <- as.factor(make.names(f.dataset$OperatingSystem,
                                                  unique = FALSE,
                                                  allow_ = TRUE))
```

```
summary(f.dataset$OperatingSystem)
```

```
##    BSD.Unix Linux.based       MacOS     Windows
##          35        5993        8456       13032
```

```
g.dataset <- clean_names(f.dataset, "upper_camel")
paste(colnames(f.dataset), " = ", colnames(g.dataset))
```

```
##   [1] "CompanySize  =  CompanySize"
##   [2] "YearsCoding  =  YearsCoding"
##   [3] "YearsCodingProf  =  YearsCodingProf"
##   [4] "OperatingSystem  =  OperatingSystem"
##   [5] "NumberMonitors  =  NumberMonitors"
##   [6] "Full-stack developer  =  FullStackDeveloper"
##   [7] "Back-end developer  =  BackEndDeveloper"
##   [8] "Database administrator  =  DatabaseAdministrator"
##   [9] "Front-end developer  =  FrontEndDeveloper"
##  [10] "Designer  =  Designer"
##  [11] "QA or test developer  =  QaOrTestDeveloper"
##  [12] "Data or business analyst  =  DataOrBusinessAnalyst"
##  [13] "DevOps specialist  =  DevOpsSpecialist"
##  [14] "Engineering manager  =  EngineeringManager"
##  [15] "System administrator  =  SystemAdministrator"
##  [16] "Mobile developer  =  MobileDeveloper"
```

```
##  [17] "Desktop or enterprise applications developer  =
↪  DesktopOrEnterpriseApplicationsDeveloper"
##  [18] "Product manager  =  ProductManager"
##  [19] "Student  =  Student"
##  [20] "Data scientist or machine learning specialist  =
↪  DataScientistOrMachineLearningSpecialist"
##  [21] "JavaScript  =  JavaScript"
##  [22] "Python  =  Python"
##  [23] "HTML  =  Html"
##  [24] "CSS  =  Css"
##  [25] "Java  =  Java"
##  [26] "TypeScript  =  TypeScript"
##  [27] "Assembly  =  Assembly"
##  [28] "Go  =  Go"
##  [29] "Ruby  =  Ruby"
##  [30] "SQL  =  Sql"
##  [31] "Bash/Shell  =  BashShell"
##  [32] "C#  =  CNumber"
##  [33] "C  =  C"
##  [34] "C++  =  C_2"
##  [35] "Swift  =  Swift"
##  [36] "Groovy  =  Groovy"
##  [37] "PHP  =  Php"

##   [output truncated]
```

Our data is ready to be used for machine learning. I will not need PCA (This is not an unsupervised learning) nor to scale my data as they are all categorical values. Moreso, I will be using a tree based model known as RandomForest which does not need scaling when working with categorical variables.

Split Data

```
# Data Partition
# I set seed to be able to reproduce same results.
set.seed(123)

# share it into 80% training data and 20% testing data.
s.data <- sample(2, nrow(g.dataset), replace = TRUE, prob = c(0.8, 0.2))

train <- g.dataset[s.data==1,]
test <- g.dataset[s.data==2,]
```

Train Model (randomForest)

Using RandomForest (Training time is 28 seconds)

```
start.time <- Sys.time()
# Random Forest
set.seed(222)
rf <- randomForest(OperatingSystem~., data=train)
end.time <- Sys.time()
print(end.time - start.time)
```

```
## Time difference of 27.74237 secs
```

View our model: We can view our model by using the print function. For specifics, the attributes can be used to narrow down further.

```
print(rf)
```

```
##
## Call:
##  randomForest(formula = OperatingSystem ~ ., data = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 23.59%
## Confusion matrix:
##             BSD.Unix Linux.based MacOS Windows class.error
## BSD.Unix           0          10     9      10   1.0000000
## Linux.based        0        2501  1195    1154   0.4843299
## MacOS              0         677  5191     887   0.2315322
## Windows            0         558   697    9139   0.1207427
```

```
attributes(rf)
```

```
## $names
##  [1] "call"            "type"            "predicted"       "err.rate"
##  [5] "confusion"       "votes"           "oob.times"       "classes"
##  [9] "importance"      "importanceSD"    "localImportance" "proximity"
## [13] "ntree"           "mtry"            "forest"          "y"
## [17] "test"            "inbag"           "terms"
##
## $class
## [1] "randomForest.formula" "randomForest"
```

ntree:
The number of trees (default valus is 500) this means that the random forest model I created used 500 different trees and
mtry:
The number of samples randomly tried per split.

For regression models, the value is the number of variables divided by 3 while for classification models, the value is closest square root of the number of variables. In my case, as shown above, mtry = 10

Explaining my atttributes

Any of the attributes listed can be viewed using the syntax "rf$" for example "rf$confusion" to see the confusion metrics.

```
rf$confusion
```

```
##             BSD.Unix Linux.based MacOS Windows class.error
## BSD.Unix           0          10     9      10   1.0000000
## Linux.based        0        2501  1195    1154   0.4843299
## MacOS              0         677  5191     887   0.2315322
## Windows            0         558   697    9139   0.1207427
```

I am not very happy with the model as 48% of the linux users are greatly misclassified. I shall try to improve on the model in order to make it better. For now, let us try predicting by using the model on the training data. This is to help give an idea on the fitting.

randomForest Model and Training Data

```
set.seed(232)
# Prediction & Confusion Matrix - train data
p1 <- predict(rf, train)
confusionMatrix(p1, train$OperatingSystem)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    BSD.Unix Linux.based MacOS Windows
##   BSD.Unix          29           0     0       0
##   Linux.based        0        4850     0       0
##   MacOS              0           0  6755       0
##   Windows            0           0     0   10394
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9998, 1)
##     No Information Rate : 0.4719
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BSD.Unix Class: Linux.based Class: MacOS
## Sensitivity                 1.000000             1.0000       1.0000
## Specificity                 1.000000             1.0000       1.0000
## Pos Pred Value              1.000000             1.0000       1.0000
## Neg Pred Value              1.000000             1.0000       1.0000
## Prevalence                  0.001317             0.2202       0.3067
## Detection Rate              0.001317             0.2202       0.3067
## Detection Prevalence        0.001317             0.2202       0.3067
## Balanced Accuracy           1.000000             1.0000       1.0000
##                      Class: Windows
## Sensitivity                  1.0000
## Specificity                  1.0000
## Pos Pred Value               1.0000
## Neg Pred Value               1.0000
## Prevalence                   0.4719
## Detection Rate               0.4719
## Detection Prevalence         0.4719
## Balanced Accuracy            1.0000
```

When we predict with the training data, there is no misclassification. The 95% Confidence Interval is quite tight. Sensitivity and Specificity are maximum. This is because it is the data that was used to build the model. One has to be careful as sometimes, 100% accuracy on training data might be an overfitting. The problem of machine learning is the ability to tune modeling parameters so appropriately that you do not overfit the model neither will you have a low accuracy with the testing data.

randomForest Model and Testing Data

Now, let us look at the test data and predict. This is where we will have the true accuracy performance of the model.

```
# # Prediction & Confusion Matrix - test data
p2 <- predict(rf, test)
confusionMatrix(p2, test$OperatingSystem)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    BSD.Unix Linux.based MacOS Windows
##   BSD.Unix           0           0     0       0
##   Linux.based        3         574   180     118
##   MacOS              0         305  1318     194
##   Windows            3         264   203    2326
##
## Overall Statistics
##
##                Accuracy : 0.7686
##                  95% CI : (0.7572, 0.7797)
##     No Information Rate : 0.4807
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6263
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BSD.Unix Class: Linux.based Class: MacOS
## Sensitivity                 0.000000             0.5022       0.7748
## Specificity                 1.000000             0.9307       0.8682
## Pos Pred Value                   NaN             0.6560       0.7254
## Neg Pred Value              0.998907             0.8767       0.8957
## Prevalence                  0.001093             0.2083       0.3099
## Detection Rate              0.000000             0.1046       0.2402
## Detection Prevalence        0.000000             0.1594       0.3311
## Balanced Accuracy           0.500000             0.7165       0.8215
##                      Class: Windows
## Sensitivity                  0.8817
## Specificity                  0.8351
## Pos Pred Value               0.8319
## Neg Pred Value               0.8841
## Prevalence                   0.4807
## Detection Rate               0.4238
## Detection Prevalence         0.5095
## Balanced Accuracy            0.8584
```

From the look of things, there is no true positively classified BSD/Unix user. In general, The BSD/Unix users are too few. I suspect that during my splitting, all of them went into the training set and not one was left in the testing set. we can improve this through Cross validation.

The accuracy of our model is 77% Kappa is 0.6263 which is substantial. Kappa rule of tumb is:
0.81 - 1.00 Almost perfect

0.61 - 0.80 Substantial
0.41 - 0.60 Moderate
0.21 - 0.40 Fair
0.00 - 0.20 Slight
0.00 Poor

95% Confidence interval is 0.76 up to 0.78.

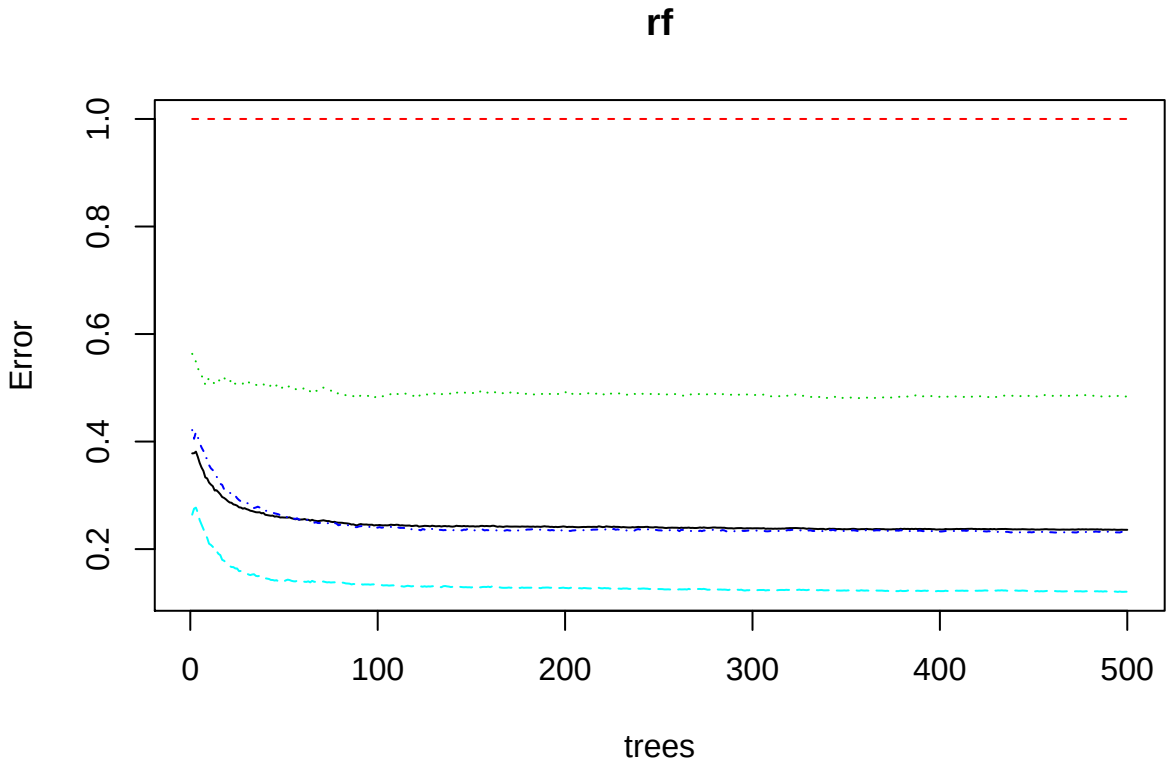
Improving Random Forest

Can this be improved upon?

Let us try.

I shall begin by finding the most suitable mtry and ntree values. To find the best ntrees, we will plot the model to see the error graph.

```
# Error rate of Random Forest
plot(rf)
```



From the graph above, the number of trees become really steady as from about 300. this means that more number of trees will not reduce the error any further.

Let us tune the model. First, I will find the column number of my target variable.

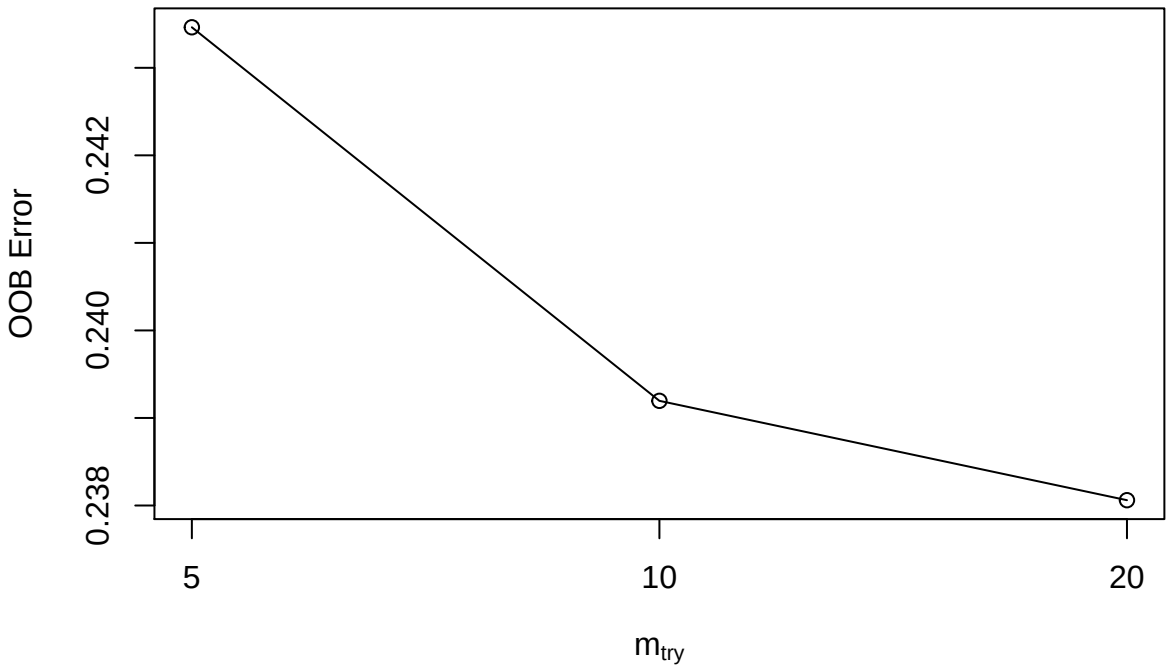
```
iv <- which( colnames(train)=="OperatingSystem" )
iv
```

```
## [1] 4
```

```
set.seed(232)
t <- tuneRF(train[, -iv], train[, iv],
```

```
       stepFactor = 0.5,
       plot = TRUE,
       ntreeTry = 300,
       trace = TRUE,
       improve = 0.5)
```

```
## mtry = 10  OOB error = 23.92%
## Searching left ...
## mtry = 20    OOB error = 23.81%
## 0.004744733 0.5
## Searching right ...
## mtry = 5     OOB error = 24.35%
## -0.0178402 0.5
```



From the graph, we get the least errors when mtry is 20. This does not look so different from the default 10. At mtry = 10, Out of Bag error becomes 0.239 and at mtry = 20, out of bag error becomes 0.240 which amounts to only 0.001 improvement. This is too small.

I should point out that before using the "set.seed()", I was getting a lesser error at mtry = 10. I believe different results will come up, depending on which tree the algorithm begins from. This does not matter so much as the error improvement is even too little. mtry = 10 or mtry = 20 almost yield the same result!.

Looks like the the Random Forest algorithm already chose the best parameters. It does not look like our model will improve by passing in my mtry and ntree parameters. Let me try.

```
rf <- randomForest(OperatingSystem~., data = train,
                   ntree = 300,
                   mtry = 10,
                   importance = TRUE,
                   proximity = TRUE)
rf
```

```
##
## Call:
##  randomForest(formula = OperatingSystem ~ ., data = train, ntree = 300,      mtry = 10, importance =
```

```
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 23.73%
## Confusion matrix:
##             BSD.Unix Linux.based MacOS Windows class.error
## BSD.Unix           0          11     9       9   1.0000000
## Linux.based        0        2496  1188    1166   0.4853608
## MacOS              0         675  5181     899   0.2330126
## Windows            0         576   694    9124   0.1221859
```

My accuracy did not improve, The only thing left to try is cross-validation.

I have just tried the cross validation and have waited over 4 hours trying to train the model. I can not continue any further as I wonder if you would have the patience to let it train when you try running my code.

I will try using a different algorithm which is extremely fast when compaired with random forest. It is called the naive bayes.

Training Model (Naive Bayes)

```
# Naive Bayes Model
nbayes <- naive_bayes(OperatingSystem ~ ., data = train, laplace = 1)
nbayes
```

```
##
## ================================== Naive Bayes ==================================
##
##  Call:
## naive_bayes.formula(formula = OperatingSystem ~ ., data = train,
##     laplace = 1)
##
## ---------------------------------------------------------------------------------
##
## Laplace smoothing: 1
##
## ---------------------------------------------------------------------------------
##
##  A priori probabilities:
##
##    BSD.Unix Linux.based       MacOS     Windows
## 0.001316506 0.220174324 0.306655166 0.471854004
##
## ---------------------------------------------------------------------------------
##
##  Tables:
##
## ---------------------------------------------------------------------------------
##  ::: CompanySize (Categorical)
## ---------------------------------------------------------------------------------
##
## CompanySize                  BSD.Unix Linux.based      MacOS    Windows
##   Fewer than 10 employees  0.08108108  0.11589131 0.10409582 0.07729283
##   10 to 19 employees       0.10810811  0.13071223 0.11518557 0.09296289
```

```
##   20 to 99 employees       0.18918919  0.26327707 0.25521218 0.22505287
##   100 to 499 employees     0.13513514  0.18958419 0.20419932 0.21082484
##   500 to 999 employees     0.08108108  0.06319473 0.06239834 0.06844838
##   1,000 to 4,999 employees 0.13513514  0.08439687 0.10099068 0.12161123
##   5,000 to 9,999 employees 0.05405405  0.03437629 0.03430430 0.04624111
##   10,000 or more employees 0.21621622  0.11856731 0.12361378 0.15756585
##
## ------------------------------------------------------------------------------
##  ::: YearsCoding (Categorical)
## ------------------------------------------------------------------------------
##
## YearsCoding          BSD.Unix Linux.based      MacOS    Windows
##   0-2 years        0.05000000  0.04669821 0.04241797 0.04420951
##   3-5 years        0.15000000  0.22176507 0.19302394 0.19971168
##   6-8 years        0.10000000  0.24439416 0.23411174 0.22546852
##   9-11 years       0.10000000  0.15758074 0.16597694 0.15031235
##   12-14 years      0.15000000  0.10615100 0.12562814 0.10840942
##   15-17 years      0.15000000  0.07323596 0.08380136 0.08793849
##   18-20 years      0.12500000  0.05780704 0.06606562 0.06660259
##   21-23 years      0.02500000  0.03003497 0.03355010 0.04007689
##   24-26 years      0.02500000  0.01954330 0.02054390 0.02566074
##   27-29 years      0.02500000  0.01090311 0.01064144 0.01335896
##   30 or more years 0.10000000  0.03188644 0.02423884 0.03825084
##
## ------------------------------------------------------------------------------
##  ::: YearsCodingProf (Categorical)
## ------------------------------------------------------------------------------
##
## YearsCodingProf       BSD.Unix Linux.based       MacOS     Windows
##   0-2 years        0.125000000 0.235342522 0.181347916 0.212878424
##   3-5 years        0.125000000 0.325447439 0.310671002 0.280634310
##   6-8 years        0.225000000 0.166632380 0.188885604 0.169726093
##   9-11 years       0.150000000 0.102448056 0.124297960 0.117251321
##   12-14 years      0.075000000 0.058218474 0.074490098 0.069293609
##   15-17 years      0.075000000 0.039909484 0.045078333 0.049783758
##   18-20 years      0.050000000 0.032297881 0.039905409 0.045074483
##   21-23 years      0.025000000 0.013577453 0.016405557 0.022969726
##   24-26 years      0.050000000 0.008023041 0.008572273 0.012782316
##   27-29 years      0.025000000 0.005554413 0.003694945 0.006246997
##   30 or more years 0.075000000 0.012548858 0.006650902 0.013358962
##
## ------------------------------------------------------------------------------
##  ::: NumberMonitors (Categorical)
## ------------------------------------------------------------------------------
##
## NumberMonitors    BSD.Unix Linux.based       MacOS     Windows
##    1           0.235294118 0.276828012 0.290828402 0.172228099
##    2           0.411764706 0.559835221 0.515680473 0.583613809
##    3           0.205882353 0.133058702 0.178698225 0.215982306
##    4           0.029411765 0.013800206 0.009023669 0.015963073
##    More than 4 0.117647059 0.016477858 0.005769231 0.012212713
##
## ------------------------------------------------------------------------------
##  ::: FullStackDeveloper (Bernoulli)
```

```
## ------------------------------------------------------------------------------
##
## FullStackDeveloper  BSD.Unix Linux.based     MacOS   Windows
##                  0 0.5483871   0.3994229 0.3671748 0.3287803
##                  1 0.4516129   0.6005771 0.6328252 0.6712197
##
## ------------------------------------------------------------------------------
##
## # ... and 101 more tables
##
## ------------------------------------------------------------------------------
```

```
plot(nbayes)
```

The above function shows numerous graphs (equal to the number of variables). The graph shows how each variable contributes to the dependent variable.

Naive Bayes and Training Data

```
# Confusion Matrix - train data
p1 <- predict(nbayes, train)
```

```
(tab1 <- table(p1, train$OperatingSystem))
```

```
##
## p1            BSD.Unix Linux.based MacOS Windows
##   BSD.Unix           2          67    37      42
##   Linux.based       15        3109  1444    1585
##   MacOS              5        1075  4582     840
##   Windows            7         599   692    7927
```

```
accuracy <- sum(diag(tab1)) * 100 / sum(tab1)
 paste("Accuracy of the model with the training data is ", round(accuracy, 2), "%", sep = "")
```

```
## [1] "Accuracy of the model with the training data is 70.91%"
```

Naive Bayes and Testing Data

```
# Confusion Matrix - test data
p2 <- predict(nbayes, test)
```

```
(tab2 <- table(p2, test$OperatingSystem))
```

```
##
## p2            BSD.Unix Linux.based MacOS Windows
##   BSD.Unix           0          15    13      15
##   Linux.based        3         714   383     342
##   MacOS              0         282  1149     234
##   Windows            3         132   156    2047
```

```
 accuracy <- sum(diag(tab2)) * 100 / sum(tab2)
 paste("Accuracy of the model with the testing data is ", round(accuracy, 2), "%", sep = "")
```

```
## [1] "Accuracy of the model with the testing data is 71.25%"
```

Improving Naive Bayes

Let me see how I can improve on this.

```
attributes(nbayes)
```

```
## $names
## [1] "data"       "levels"     "laplace"    "tables"     "prior"
## [6] "usekernel"  "usepoisson" "call"
##
## $class
## [1] "naive_bayes"
```

Using ?naive_bayes, I read up the documentary on the attributes. Sadly, tweaking the parameters of the naive_bayes function mostly favour numeric variables. Since all my variables are categorical in nature, I am only left with cross validation. I will gladly do it this time because naive bayes is very fast. I hope to get a better accuracy by so doing.

Sadly, my solution for cross validation does not work!!! I would gladly appreciate it if I am able to get a feedback on this.

```
set.seed(121)

# create control object for cross validation.
ctrl <- trainControl(method='cv',
                     number=10,
                     savePredictions = TRUE)

cv.nb <- train(OperatingSystem ~., data = train, method = "nb", trControl= ctrl)
```

```
## predictions failed for Fold01: usekernel= TRUE, fL=0, adjust=1 Error in
↪  predict.NaiveBayes(modelFit, newdata) :
##   Not all variable names used in object found in newdata
## model fit failed for Fold01: usekernel=FALSE, fL=0, adjust=1 Error in
↪  NaiveBayes.default(x, y, usekernel = FALSE, fL = param$fL, ...) :
##   Zero variances for at least one class in variables: Groovy1, Kotlin1, R1, Xamarin1,
↪  Git1
```

Explanation as to why it did not work

Here is what I think. Cross-validation works by sharing my training data into k folds (in my case, k=10). Perhaps, during the demarcations, some of my data were not duly represented. For instance if I have 1250 observations and I remove the near zero variance variables. Then I share my data into 80:20 ratio (training and testing respectively). This further reduces my training observation to 1000 and 250 for testing. with 10 fold cross validation, my training data for each training further reduces to 900.

I suspect that reduction in number of observations will once more reintroduce near zero variance in some columns and this is not good for naive bayes since it is built on probability.

There are two solutions to this. The first is to be more brutal at the point where I removed near zero variance columns just before splitting my data into training and testing. However, I am going to loose more columns which are vital to have a good prediction model.

Already, my dependent variable (OperatingSystem) has a very small category of BSD/Unix users which is less than 3% of the entire respondents. In the end, when cross validation will be splitting, training and doing its job, the chances of certain splits not having BSD/Unix users is very high. The more columns I remove, the lesser the means of teaching the model to explain the dependent variables.

The next solution will be to convert the one hot encoded columns into long format. But this will certainly increase the length of my dataset by over 200%

It is worthy of note that cross validation with random forest takes longer but works fine because it is a tree based model. However, it is very slow as it has to learn all the data on all the trees. This is unlike the fast naive bayes algorithm (naive indeed) which refused to do my cross validation for me simply because it is a probability based model and perhaps some of my data were not well represented during the cross validation.

Critical evaluation and discussion of the significance of the applied AI techniques

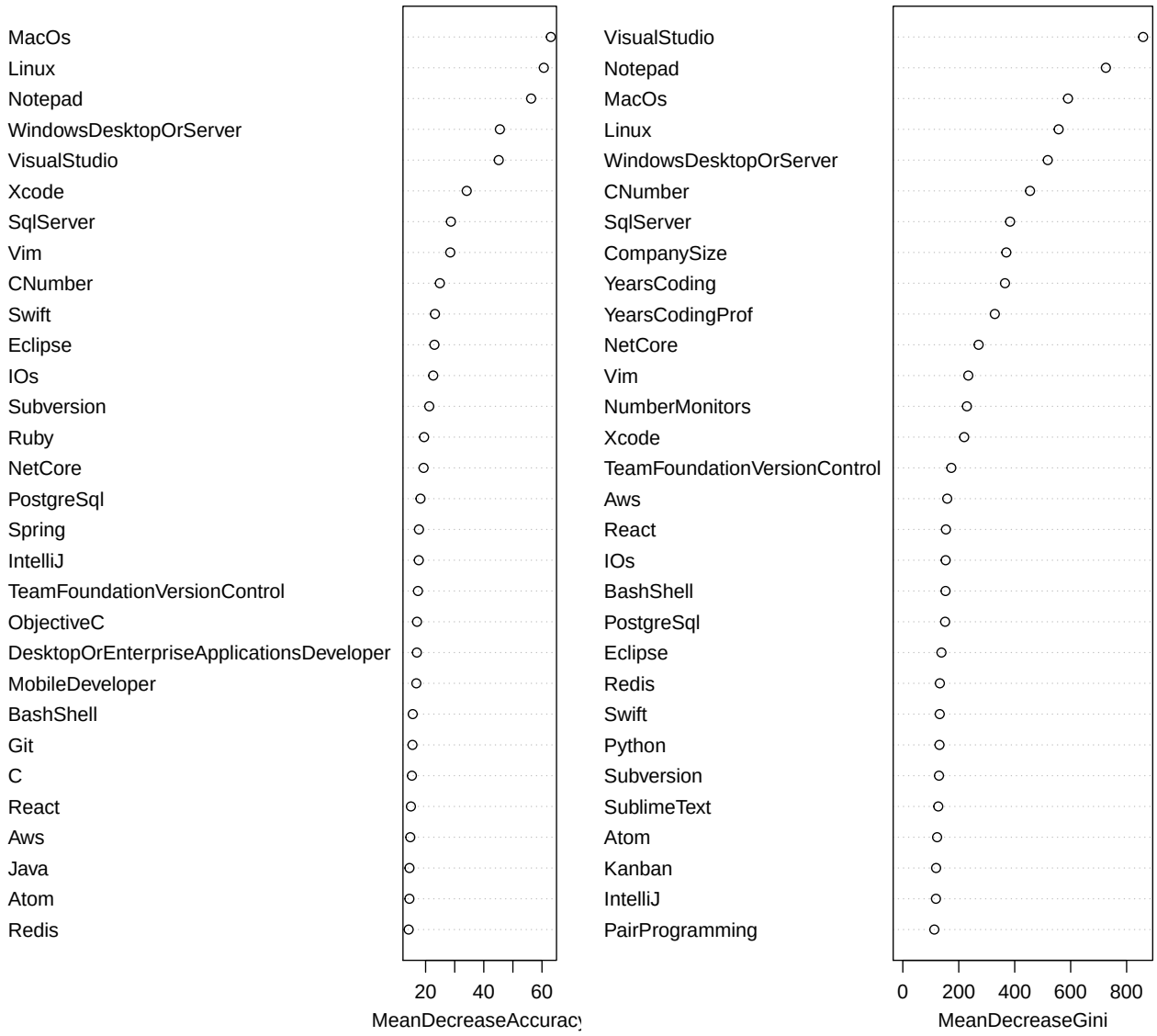
There are several techniques in AI such as Machine Learning, Natural Language processing, Automation and Robotics, and Computer Vision and I have used only one in this project.

You will find below the justifications of the machine learning algorithms I used in accomplishing my task. Most of the knitty-gritties have already been discussed in my R code above.

Why Random Forest?

Random forest is a tree based model with great versatility. It can be used on both classification and regression problems. The main reason why I went straight to it is because it would allow me to view my variables based based on scale of importance. With Random Forest, how much the features contribute towards prediction can easily be seen as shown below.

```
varImpPlot(rf)
```



Random forest is user friendly and straightforward, it has just few tunable hyperparameters of which their dynamic default values are designed to give the best model.

Random forest is also a very handy algorithm since its default hyperparameters often produce a good prediction result. Understanding the hyperparameters is pretty straightforward, and there's also not that many of them. (mtry and ntrees).

One of the biggest problems in machine learning is overfitting, but most of the time this won't happen thanks to the random forest classifier. If there are enough trees in the forest, the classifier won't overfit the model.

Overfitting is a huge machine learning problem. A random forest model if trained with enough trees is most likely not to overfit the data.

Randon forest however comes with its own disadvantages and limitations. It was very slow to train. When I tried to improve on my model via cross validaion, It failed me as it took plenty of runtime. The random forest being a slow algorithm is ineffective for creating solutions for problems that require speed. This was

the reason why I moved on to the very fast naive bayes.

Why Naive Bayes?

I fell back on naive bayes for its simplicity and speed after toying with Decision trees and Support Vector Machines (Yes, I did try those too). It was so fast such that I began to wonder if my code actually ran. I was filled with disbelief until I used the model to predict the testing data. It did try to match the random forest but the random forest had more accuracy. Accuracy for random forest was 77% while that of Naive Bayes was 71%. I must confess, although it had a lesser accuracy, Naive bayes did a good job with so much speed and has won my love.

The bayesian classifier as seen in the module material, Week 6, Lecture 6, part 2 (Di Stephano, A. 2020) perfectly explains the algorithm. Naive bayes is known to work best with large datasets because of its speed. It can as well perform well with little training data. Although in some cases, Naive bayes has proven to have comparable performance with decision trees and selected neural networks classifiers however, it makes a very huge assumption which makes it naive. It assumes that all features on my dataset are independent of each other. I wish It never made this assumption, It would have been the best machine learning algorithm as its accuracy will increase. Real data always have dependent features. As with my dataset, It is common knowledge that people who are into data modelling frameworks could use Power Bi and in turn are more likely to use the windows operating system than a mac or linux. Naive bayes does not take this into account because it handles each column separately. This is why those who apply it try to do so only when the feature importance is the same for all variables as usually is the case with text data. email classification, etc.

Reflections

This ICA was interestingly very challenging. Despite the numerous materials available on the dashboard, I found myself searching for solutions and asking a lot of questions on online platforms like reddit and stackoverflow just to know how to go about the steps. The course on its own gave me an idea of how to go about AI and machine learning. I became aware of the principles however, It was so tough replicating what was learnt on a real dataset.

For instance, during lab sessions, built-in data in R such as cars dataset, iris dataset, etc. are always used. I felt as though I understood all that was taught during the lab sessions. I was very mistaken because, real data was different. It had more cleaning and more manipulation. It is bigger, more disorganised than the built-in datasets we practised with. This made me research a lot, having had an idea of what was required of me in the course, I knew exactly what to do so I had to search and find how it was done whilst consulting my lecture materials as well.

The online community support is huge and quite helpful. Most of my questions have been asked and have been answered before by different people. Sometimes, I would have difficulty finding a function to do what I want so I had to learn to write functions myself. I regret that the functions I wrote were slow because it entailed iterating through a dataset one by one. I wished there was a way I could make them run faster and more memory efficient. I did learn about functional tests in my python module however, I did not have enough time to implement that here. Using the knowledge obtained from my Python module, I was able to backup my progress using git. You can clone the repository using this link https://github.com/Chuukwudi/My-ICA-on-R.git

One of my biggest challenge so far in this ICA was in data cleaning and visualisation. Some columns in my data was formatted in a very terrible manner. Writing functions to clean the data made me to re-visit and familiarise myself with the basics. I came to a lot of dead-ends. Initially, My plan in this ICA was to predict salaries and incomes based on other variables. I discovered late that that would have been a disaster as there are lots of different factors affecting salaries but not reflecting in the data. I found out how difficult it was to predict continuous variable using categorical data. Even after I one hot encoded all my variables and made them as continuous as they can be, I then found out that there was little or no correlation between the salaries and the variables. I stopped and restarted this ICA more than once. I had the option to encode the salaries as categorical variables using intervals but then..an idea occurred to me. Why not put an end to

the online battle of Operating systems once and for all? Could there be anything causing people to prefer one operating system over another? Could it be their years of experience, the platforms, frameworks and languages they work with? Could it be just a trend? This furthermore kindled my interest.

Some of those functions took me days to write. There were challenges and as soon as I solved one, another immediately sprung up. It felt so rewarding after solving each challenge I encountered. I remain very grateful for this experience as I am more confident with kaggle as opposed to the first few moments I started visiting the website. When I first heard about kaggle, it looked so alien to me. But now, I have the ability to go through notebooks of other people and watch how some problems have been approached.

My greatest help came from reading documentations on functions by using the question mark followed by the function name. I could read about a whole library by using double question marks instead. This served as a dictionary and made using the libraries easy.

I really appreciate my teacher Dr. Alessandro Di Stefano for teaching us research methodologies (Use of google scholar) and very helpful and interesting tools such as LaTeX. I did not just learn AI and R but LaTeX as well.

Future Plans

After having spent some time with R, I have fallen in love with the language. I plan to participate in kaggle competitions and earn myself a good reputation. This is my first activity in AI and Machine learning and I believe it will only get better with time. My forthcoming advanced practice will be an avenue to properly familiarise myself with industrial standards and I really look up to it.

I am also very curious and excited about the modules in the next semester where I will be learning “Deep Learning”, “Machine Learning” and “AI Ethics and Applications”. There is still a lot to learn and do in this field as it is still growing. I look forward to contributing and collaborating with like-minded people to make AI simpler and better. Having an AI hobby group with an advanced mentor will be wonderful as we collectively exchange ideas and collaborate under the auspices of our mentor.

As soon as I graduate, I would like to continue to Ph.D. in the field of AI. I gladly would not fail to take up any real world application or opportunity to further sharpen my skills and get my hands dirty.

References

Buchanan, B.G., 2005. A (very) brief history of artificial intelligence. Ai Magazine, 26(4), pp.53-53.

Caughlin D. 2020, K-Fold Cross-Validation in R, Available at: https://www.youtube.com/watch?v=BQ1VAZ7jNYQ (Accessed: 9th January 2021)

Di Stephano A. 2020, Week 6–Machine Learning –Classification I: Naïve Bayes and K-Nearest Neighbour, Teesside University, pp.2-14

Does your AI discriminate?, 2020. Available at: https://theconversation.com/does-your-ai-discriminate-132847. (Accessed 9th January 2021)

Haenlein, M. and Kaplan, A., 2019. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. California management review, 61(4), pp.5-14

Hong, J.W., 2020. Why Is Artificial Intelligence Blamed More? Analysis of Faulting Artificial Intelligence for Self-Driving Car Accidents in Experimental Settings. International Journal of Human–Computer Interaction, 36(18), pp.1768-1774.

Jackson, P.C., 2019. Introduction to artificial intelligence. Courier Dover Publications.

Kambria. 2019, The 7 Most Pressing Ethical Issues in Artificial Intelligence. Available at: https://kambria.io/blog/the-7-most-pressing-ethical-issues-in-artificial-intelligence/ (Accessed: 9th January 2021)

McCarthy, J., 1998. What is artificial intelligence?.

Ng, A., 2017, January. Artificial intelligence is the new electricity. In presentation at the Stanford MSx Future Forum.

Plant, R., 2011, March. An introduction to artificial intelligence. In 32nd Aerospace Sciences Meeting and Exhibit (p. 294).

Rai D., 2017, Random Forest in R - Classification and Prediction Example with Definition & Steps. Available at: https://www.youtube.com/watch?v=dJclNIN-TPo&t=1256s (Accessed: 22nd December 2020)