Working With Data Final Project: Dota Science

Jonathan Ludolph

12/12/2019

Contents

1 - Introduction to DotA 2

Defense of the Ancients 2 (better known as DotA 2) is one of the most played online multiplayer battle arena game in the world right now. Because of the amount of practice, strategy and skill involved, it makes for one of the most exciting esports to watch. In fact, last year, the 9th world cup of DotA (also kown as The International 9) reached an unprecedented prize pool of over 30 Million dollars.
The game consists of matches which typically last around 40 minutes in which two teams of 5 players battle each other to reach the enemy's base and destroy their 'Ancient' (a large structure within the base). Each of the 10 players selects a 'Hero' amongst the 115 Heroes available that they will play over the course of the match. Each hero can only be picked by a single player during the Match. As the match progresses, players earn Money (Gold) which allows them to buy items (weapons, armor and magical items which give bonuses and essentially advantages over the enemy) and Experience (XP) which allows them to gain levels for more powerful abilities. Gold is earned by killing enemy Heroes or Creeps (which are neutral creatures around the map). Experience is earned when an enemy or a Creep dies in a radius near your Hero. Once you have earned enough experience you gain levels which allow you to choose which of your abilities you want to upgrade.
There is a lot of Data to be found about DotA 2, as several types of data is being generated for every game played (typically about 600000 at all time). There is data about in-match statistics, post-match statistics, player statistics, overall accounts statistics and so on. The data is open and available to anyone on 'opendota.com'. There are a lot of parameters which can be taken into account when trying to predict a team's victory but the three main factors in DotA 2 remain Experience, Gold and what is done with it (Items bought). As a matter of fact, itemization is one of the most complex aspects of the game. It requires in-depth understanding of all items and all heroes in order to know which items would fit which heroes and which items would counter them.
DotA 2 is a game that constantly changes, as it has updates about once a week which changes statistics about all sorts of things within the game such as Heroes statistics and abilities and item changes. Therefore, it is difficult to keep track of what Heroes are best at which time and which items are most useful for which hero. That is where the data become useful: one can build models about current statistics and create functions that could potentially adapt to each new update through machine learning. Where would we start?

2 - Objectives

In order to begin with creating a model of how DotA 2 would be played best, one must investigate Hero behaviour: the key to winning a match of DotA 2, is understanding how to play the Hero and what to do with him. One of the biggest questions that people ask themselves when a new update comes out, is what to prioritize: Experience or Gold?
It has been an ongoing debate for a while now and is a question of utmost importance.
Secondly, 'item builds'. Item builds is an expression refering to which items the player is buying for his hero. This is the second biggest change that comes with each update: the Hero and item statistics change fluctuate constantly and therefore, item builds must adapt. It isn't easy to constantly adapt item builds to recent updates without being a pro player. How can we choose items accordingly with the help of data?
I will investigate the overall relative importance of Gold and Experience first and secondly, estimate which items are best for the most played Hero for the update that we will be looking at.

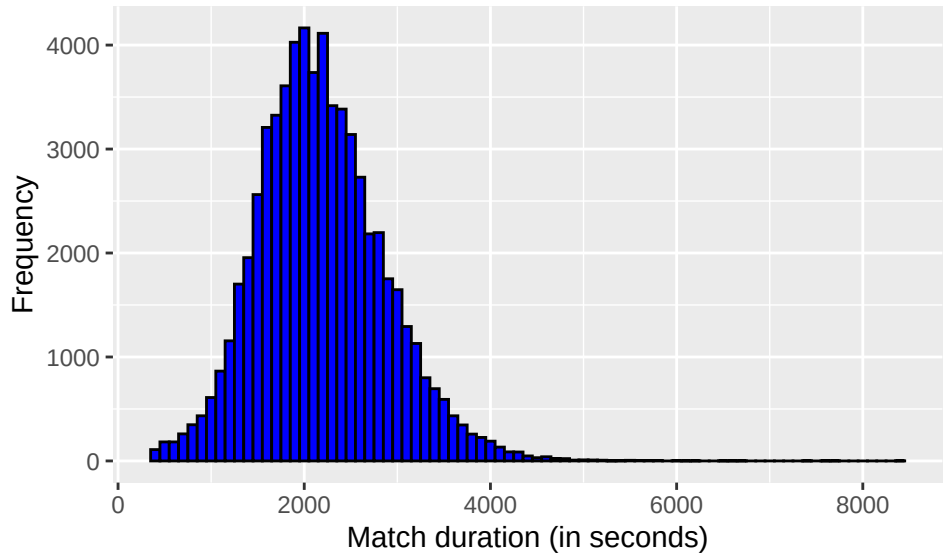

Figure 1: Frequency of Match durations

3 - Data

Initializing the data and getting it into a fomat that would allow me to analyze it was the most challenging part. The data was obtained from www.kaggle.com and found through the Google Dataset Search tool. It was assembled by the kaggle user from www.opendota.com. This data set contains over 63000 observations. Each observation is a unique match. There were originally 7 columns. The first six columns contained over all information about the match and the seventh was a column with nested data frames (each row was a data frame on it's own). The nested data frames contained information about each player, which is why it was a data frame on it's own: there were ten observations per data frame with 14 variables containing information for each player.
I encountered two major difficulties when attempting data wrangling: first of all, the original file was in .json format as well badly formatted (there weren't any commas between each observation which R interpreted as being a file of different objects and not a single data frame). This was solved by text editing in Bash with the following command:

```
sed -e 's/$/,/' matches.json > matches_reworked.json
```

This added commas to every line and wrote the result to a new file entitled 'matches_reworked.json' which was then loaded to R with the 'jsonlite' package and the function 'fromJSON()'.
Once I overcame this obstacle, I had to flatten the player nested data frame. In order to do so I created a function which took the data frame as argument and created new columns by combining the player ID with the name of the columns from the nested data frame . This newly formed, neatly formatted data frame was then fused with the original data frame, replacing the player column with the newly created variables.
This final data frame contained then the same amount of observations as the original one but had 136 variables. Over the analysis, a lot of data wrangling was executed in different ways depending what was need to be analyzed, because of the size of the data. The columns were reduced according to the needs.
Some general data wrangling was carried out, however, before the start of the actual analysis. After summarizing all columns, I noticed that the variables 'dire_score' and 'radiant_score' had a lot of missing values. These variables weren't used in this analysis, therefore they were be removed later on. I initially had a look at the distribution of the 'duration' variable (which represents the amount of time the matches lasted in seconds) in order to rule out any outliers with extreme values that could have tempered with our results (Figure 1).

On Figure 1, we can observe that most of the matches lasted around 40 minutes but very rarely would it be longer than 6000 seconds, which is about an hour and a half (only 9 times to be precise). Therefore I deleted those rows in order not to have outliers.

4 - Results/Discussion

Gold, XP and Victory

As mentioned before, the aim of this analysis was to evaluate the best and most effective way of playing DotA 2 in order to have a theoretical advantage over your opponents with the help of match data. The first piece of analysis that was carried out was to investigate the importance of Gold and Experience regarding match outcome - win or loss.
After having created a separate data frame containing the variables 'radiant_win', 'XP_radiant', 'XP_dire', 'Gold_radiant' and 'Gold_dire' ('Victory'), I could start working on the model. The variables 'XP_radiant', 'XP_dire', 'Gold_radiant' and 'Gold_dire' were obtained by taking 'xp_per_minute' and 'gold_per_minute' from each player from each faction, adding them up respectively and finally multiplying them by the 'duration' in minutes as follows:

$$\text{XP\_radiant/dire} = \text{total(xp\_per\_minute)\_radiant/dire} \times \frac{\text{duration}}{60}$$

and the same for gold. Next, I applied a Machine Learning model with the help of the *caret* and *pROC* packages. This type of model doesn't accept any missing values, therefore I double-checked that there weren't any. It turns out that the variable 'radiant_win' contained five missing values. Five missing values compared to the total amount of values wasn't going to impede on the accuracy of my model so I decided to remove those rows.
After having modified the 'radiant_win' values to factors 'win' and 'loss' instead of 1s and 0s, I did a standard train/test split of the data where I used 75% for training and 25% for testing. The outcome variable is categorical so instead of using RMSE as a metric, I used ROC. When running the model, we could see that there was a clear distinction (Figure 2).

```
##                        var    rel.inf
## Gold_radiant Gold_radiant 52.0207233
## Gold_dire       Gold_dire 47.4098172
## XP_dire           XP_dire  0.3060599
## XP_radiant     XP_radiant  0.2633995
```

On Figure 2, we can see that the experience (XP) barely plays a role in the winning outcome: the amount of gold gathered by the radiant side and the amount of gold gathered by the dire side combined, add up to about 99.3% of relative relevance. We could therefore argue that for the specific update at which the data was collected, gold was a major game-winning factor: a team could completely disregard gaining experience, focus on gathering gold, and the win was almost certain.
In order to back up my arguments, I checked the model accuracy by comparing predictions made with the model from the test data frame and the actual values. As a matter of fact, the accuracy of this model was excellent: about 98% of predictions were correct (as depicted in the table below).

Accuracy	Kappa
0.9831758	0.9663277

Since gold was the most important factor, it meant that items were therefore the most important factor of the game. Which brings us to the second part of the analysis: which hero was picked the most and what were the best items to buy for him.

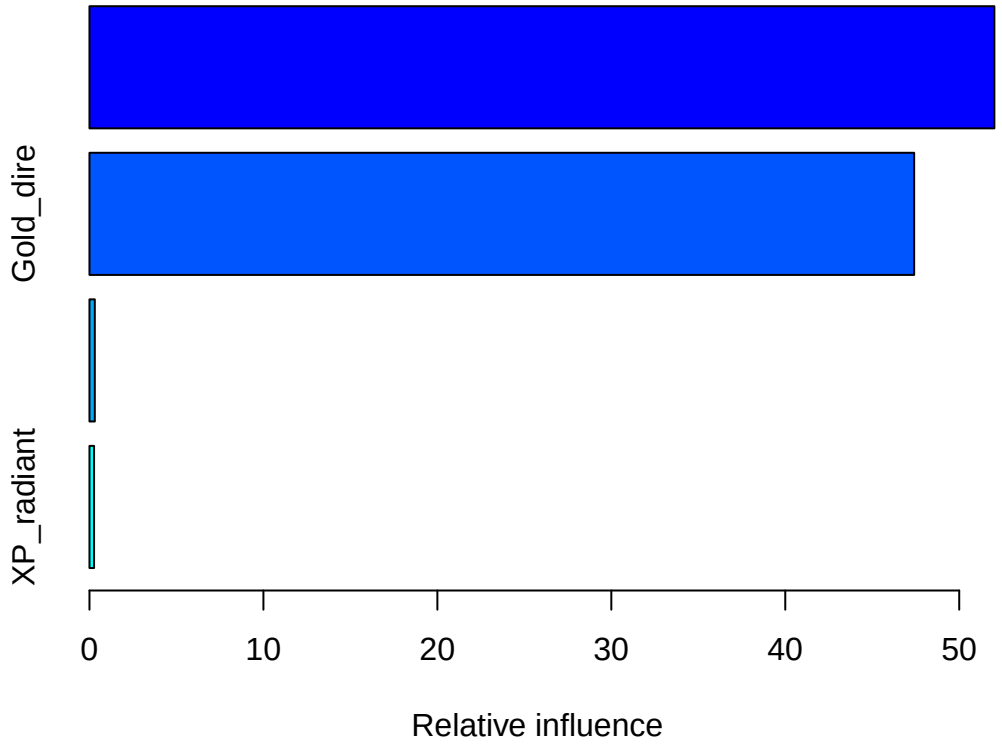

Figure 2: ML model: relevance of Gold and XP

Most Picked Hero and His Favourite Items

In order to process the Heroes correctly, the second data set that came with the matches was the heroes.json data set. Similarly to the 'matches.json' dataset, the data file was processed in bash and commas were added (as it was badly formatted as well). Once the data was loaded into R, it was kept aside, as it was only going to be of use later on.
A new data frame was created to filter the variables which contained IDs for heroes. By performing a count through a table, I found which were the 10 most picked heroes (table below).

Hero ID	Frequency
86	13108
74	12532
9	12148
7	11075
14	10807
21	10449
8	10295
62	9997
11	9635
39	9384

Comparing the IDs with the Heroes data frame containing all of the Hero information and data, I found out that the most picked hero was 'Rubick'.
Once I knew which hero was the most picked, it was time to investigate the itemization (also known as item builds). This part was tricky as the heroes weren't always picked by the players which had the same player IDs. In fact, it was with tidyverse that the rows were analyzed to pin point which matches had a player on a team that picked Rubick.

However, to select only the items for that specific player, I had to create a new data frame which contained only the columns for the items and filter it with the player ID in order to grab his six picked items through a for-loop.
With the item selections acquired, I could examine which ones were the most picked and pick out the top 6 most picked items. In order to obtain the actual names of the items, I had to find the list of IDs for the items. The package 'RDota2' allows users to grab data directly from the opendota API and I managed to retrieve a list of items with their respective IDs. The top 6 items were the following:

Frequency	Item
8586	Arcane Boots
8088	Magic Wand
6795	Town Portal Scroll
5914	Blink Dagger
4258	Force Staff
1922	Observer Ward

On this table, we can see that more than half of the players that picked Rubick, also picked Arcane Boots and Magic Wand. That is an extremely high rate of item pick. We could think that those items must work really well with Rubick's abilities and needs. In order to verify this statement, we are going to have a look at the win rates for the most picked item builds and see if they really are the best way to itemize when playing a Hero like Rubick.
In order to compare the win rates of the different item builds, I had to grab the items, put them into a list in order to be able to compare them and see how many builds were exactly the same (a single differing item changes the whole build and its results). To carry out those counts, the list of items for each match where Rubick was played was compared to the rest of them (for each row, the list was taken and compared to the rest of the rows and counted as identical or not and the outcome of the game was checked). Finally, we ended up with a data frame with the counts of how many times the specific builds were picked, and how many games were won, in order to calculate the relative win rate.
The win rates allowed us to compare the most picked items with their relative success and relevance. I took the 10 most picked item builds and ranked them according to their relative win rates. The build with the most success was the following:

ID	Item
0	Empty Slot
0	Empty Slot
0	Empty Slot
36	Magic Wand
102	Force Staff
180	Arcane Boots

Here we see Arcane Boots and Magic Wand again, which proves that the most picked items by players were actually correct. Even though, the build with the most success is one with three empty item slots. It could result from the fact that if the teams had a big advantage and were simply playing better, the player may not have had the time to finish buying all of his items, since they could have won the match very quickly.

5 - Limitations / Future work

Over the course of this analysis, I ran into some major difficulties. As I stated earlier, the format of the data set was a major issue to start with with many modifications to be made in order to be able to merely begin to analyse it. In order to improve accuracy throughout this analysis, one could have made some

improvements that would have been extensively complex and difficult to run with a single node.
First of all, for the Gold and Experience modelling, the fact that the matches were over when the data was taken, influenced greatly the accuracy of the model. The winning team was bound to have more Gold and Experience than the losing team, thus rendering the outcome predictions highly accurate. In order to look more closely at the effects of both of these factors on the match outcome, one could take a look at more in-depth data about values over the course of a match. The gold_per_min and xp_per_min variables are taken as an average from the final total amount of Gold and XP earned by the players. A time-series analysis of Gold and XP quantities over several matches would be a more relevant approach.
Furthermore, I had some interest in clustering the results by position. In DotA 2, there are two major types of positions: Supports and Cores. The Cores are the Heroes that need to gather a maximum amount of Gold and purchase items in order to reach their power spikes, whereas Supports are Heroes that don't necessarily require to purchase items in order to be impactful over the game; they usually have strong abilities that allow them to grow over the course of a match simply by levelling up with experience and upgrade their abilities. The major issue with this is that there are secondary roles. In fact, when trying to separate Heroes into their respective positions, I realized that some Heroes weren't tagged as either support nor core. On top of that, some Heroes can be played as both, which makes positionning fairly subjective. But it could potentially be a relevant piece of analysis to separate the needs for Gold and Experience according to Heroes played and not as a whole.
Secondly, the item analysis was only carried out for one Hero. Unfortunately, applying the method to all Heroes would have required a much more important computing power, which I wasn't able to access. With the win rates of item builds one could create an AI which suggests which items are best to pick for which Hero. Moreover, items can be relevant to choose according to the Heroes picked by other players as well. A model which takes into account items chosen, win_rate as well as Hero picks by all players (teammates and opponents) would be a much more sensible approach. 'DotA 2 Plus' is a premium service on DotA 2 which provides such information. Every time a match begins, it analyses which Heroes were picked by each player and suggests a build for your Hero, based on all the factors that I mentionned previously.
Finally, the number of observations was limited compared to the amount of data that actually exists. With more computing power one could analyse all of the matches played every day and train machine learning models to adapt to the new ways of playing the Heroes. An other interesting way that one could analyse the data would be to have the actual update indicator, as well as the levels of expertise of players (professional players play Heroes differently than beginners). There are many factors that could be taken into account when trying to model DotA 2 data and my own analysis is only scratching the surface of it, due to the lack of ressources.

6 - Conclusion

As we have seen, DotA 2 can be a versatile game and quite complex to understand: it isn't for everyone. The high level of skill and strategy required to be a good DotA player make professional DotA 2 players among the smartest esport athletes of all. When it comes down to it, winning a game depends as much on the skill as understanding the way Heroes are played and how well they work together as a team. The data can be relevant however, for non-professional players, a simple hint of an advantage before even starting the game can throw the game off-course for the adversaries.
Throughout this journey within the depth of DotA, some distinctions were made, that could potentially help players seek out concrete evidence of statistical winning advantages. Experience and Gold were compared and analyzed through a Machine Learning model with a categorical outcome ('radiant_win'). It was discovered that over all, Gold appeared to be much more important as a predictor of match outcome as XP (more than 99% relevance). It was discussed however, that the analysis could've required some additional factors to be taken into account, which would have greatly improved the reasoning behind our results.
Furthermore, from the importance of Gold and the fact that Gold is used to purchase items for additional power and advantage over the opponent, itemization was investigated. The most picked Hero among the players in the dataset was found (Rubick) and his item builds were evaluated. I found out that the items purchased by most players who played Rubick were 'Arcane Boots' and 'Magic Wand' with a purchase rate of over 50% of Rubick players: a major preference.

It wasn't enough to simply evaluate which items were most frequently bought, one also had to see if those purchases were justified. In order to judge whether or not these items were genuinely the best items to get for the most popular Hero, the win rates of the different builds had to be calculated. It was shown that as a matter of fact, the most successful build contained both 'Arcane Boots' and 'Magic Wand'. Which proved that popular opinions were indeed justfied by it's success.

Appendix

```
  ### DOTA PROJECT ###

## Obtaining Dataframe

library(tidyverse)
library(ggplot2)
library(jsonlite)
library(reshape2)
library(caret)
library(pROC)
library(broom)
library(Amelia)
# Reading the data from JSON file
# The issues with the file: it was badly formatted; had to add commas to every line,
# as well as square brackets at the beginning and the end of the file.
# added commas with bash: 'sed -e 's/$/,/' matches.json > matches_reworked.json'. Similary,
# the same was done to modify the 'heroes.json' file and was renamed 'heroes_reworked.json'.

JSON_list_matches <- fromJSON('matches_reworked.json',flatten=TRUE)
# The heroes_reworked.json file contains information about the heroes, mostly we will only
# use it to identify the hero IDs from the player data.
JSON_list_heroes <- fromJSON('heroes_reworked.json',flatten=TRUE)

MatchData <- as.data.frame(JSON_list_matches)
Heroes <- as.data.frame(JSON_list_heroes)

head(Heroes)

# function to flatten the nested player data frames
Player.flatten <- function(player_list){
  flat <- dcast(melt(as.data.frame(player_list),
                     id.var="player_slot"),
                1~player_slot+variable)
  return(flat)
}

head(MatchData)
# Extracting the list of nested data frame from the global data
Player_stats <- MatchData$players


# Flatten the columns to obtain the data structure that we want
Player_df <- as.data.frame(do.call(rbind, lapply(Player_stats,FUN=Player.flatten)))[-1]
# [-1] because the first column is a column of 1s that comes from the Player.flatten function
head(Player_df)
```

```r
# Now lets combine the Player_df with Match Data to replace the original 'players' column
# that contained the nested data frames.

MatchDataClean <- cbind(MatchData[-7],Player_df) # [-7] to remove the old 'players' variable

## Wrangling / Cleaning

# We are interested in the Experience (XP) a player gets and Gold. Therefore, we can
# go ahead and remove any variables that we won't be needing.

summary(MatchDataClean)

# We can see that the dire_score and radiant_score have 31454 NA's each. which is almost
# half of the rows. Fortunately for us, we won't be using the columns for our analysis,
# therefore we can keep these rows but remove the columns later.

colnames(MatchDataClean)

write_csv(MatchDataClean,'MatchDataClean.csv')
# saving the data to a separate file, for ease of use
MatchDataClean <- read.csv('MatchDataClean.csv')

# Let's look at potential outliers from the duration column and visualise them
summary(MatchDataClean$duration)

ggplot(MatchDataClean,aes(x=duration))+
  geom_histogram(binwidth=100,fill='blue',colour='black')+
  labs(x='Match duration (in seconds)',y='Frequency')

# We can see that the matches are mostly of the same duration and that a very little
# amount of matches actually last longer than 6000 seconds. Let's check how many
# matches actually were longer than 6000 seconds, because these are very long matches and will
# our analyses.

long_games <- MatchDataClean %>%
  filter(duration >= 6000)

long_games$duration
# There are only 9 games which are seriously long, we can delete these rows, deleting them
# won't affect the accuracy of our analysis.

MatchDataShort <- MatchDataClean %>%
  filter(duration <= 6000)

# We can now start selecting the columns that we will need for our initial analysis.

XP_advantage_cols <- MatchDataShort %>%
  select(ends_with('xp_per_min'))

Gold_advantage_cols <- MatchDataShort %>%
  select(ends_with('gold_per_min'))

#### XP, Gold and victory
```

```r
Victory <- MatchDataShort%>%
  select(radiant_win)


# changing from boolean to actual factors for modelling
Victory$radiant_win <- as.factor(ifelse(Victory$radiant_win==TRUE,'win','loss'))



# Now let's create columns of Gold and XP advantage for Radiant side



Victory$XP_radiant <- rowSums(XP_advantage_cols[,1:5])*(MatchDataShort$duration/60)
Victory$XP_dire <- rowSums(XP_advantage_cols[,6:10])*(MatchDataShort$duration/60)
# xp per min * minutes played

Victory$Gold_radiant <- rowSums(Gold_advantage_cols[,1:5])*(MatchDataShort$duration/60)
Victory$Gold_dire <- rowSums(Gold_advantage_cols[,6:10])*(MatchDataShort$duration/60)
# gold per min * minutes played

head(Victory)

# We still have 5 missing values for radiant_win, which a very small amount
# compared to the number of observations. We can go ahead and remove them.

Victory <- na.omit(Victory)

Victory[is.na(Victory),] # 0 missing values
## ML model

##  Data Partition with seed to be able to reproduce
set.seed(10122019)
# train test splitting: getting our train rows from createDataPartition and
# using them to split our data frame

splitRows <- createDataPartition(Victory[,'radiant_win'],p=.75,list=FALSE,times=1)
trainVictory <- Victory[splitRows,]
testVictory <- Victory[-splitRows,]

objControl <- trainControl(method='cv', number=3, returnResamp='none',
                           summaryFunction = twoClassSummary, classProbs = TRUE)

# Categorical model so we use ROC

objModel <- train(trainVictory[,c('XP_radiant','Gold_radiant','XP_dire','Gold_dire')],
                  trainVictory[,'radiant_win'],
                  method='gbm',
                  trControl=objControl,
                  metric = "ROC",
                  preProc = c("center", "scale"))

summary(objModel)
```

```r
# Here we see that our model argues that XP is close to irrelevant when it comes to victory.
# (approx. 99% vs. less than 1%). Let's evaluate the model:

predictions <- predict(object=objModel,
                       testVictory[,c('XP_radiant','Gold_radiant','XP_dire','Gold_dire')],
                       type='raw')

print(postResample(pred=predictions,obs=testVictory[,'radiant_win']))

# Our model has an accuracy of 98.3% - incredible.
# We can deduce that Gold is more important than Experience for this patch.


##### Let's try the items and see how much item picks for the top 3 most picked
##### heroes influence the outcome of the game

Hero_picks <- MatchDataClean %>%
  select(ends_with('hero_id'))

Hero_picks_vector <- c(
  Hero_picks$`0_hero_id`,
  Hero_picks$`1_hero_id`,
  Hero_picks$`2_hero_id`,
  Hero_picks$`3_hero_id`,
  Hero_picks$`4_hero_id`,
  Hero_picks$`128_hero_id`,
  Hero_picks$`129_hero_id`,
  Hero_picks$`130_hero_id`,
  Hero_picks$`131_hero_id`,
  Hero_picks$`132_hero_id`
)



Top_picks <- as.data.frame(table(Hero_picks_vector))%>%
  arrange(desc(Freq))

most_picked_id <- Top_picks[1:10,] # Most picked id = 86
most_picked_id

head(MatchDataClean)

MatchDataMostPicked <- MatchDataClean %>%
  select(ends_with('hero_id'),
         contains('item'),
         radiant_win
         ) %>%
  filter(`0_hero_id`==86|
         `1_hero_id`==86|
         `2_hero_id`==86|
         `3_hero_id`==86|
         `4_hero_id`==86|
```

```
         `128_hero_id`==86|
         `129_hero_id`==86|
         `130_hero_id`==86|
         `131_hero_id`==86|
         `132_hero_id`==86
    )

head(MatchDataMostPicked)


# get the cell which contains the value, find how far in columns, items are and grab them

ItemBuilds86 <- data.frame(radiant_win=MatchDataMostPicked$radiant_win)

dim(MatchDataMostPicked)

Items <- MatchDataMostPicked %>%
  select(contains('item'))

for(i in 1:dim(MatchDataMostPicked)[1]){
  col_id <- which(MatchDataMostPicked[i,1:10]==86)
  # get column number for which player is playing hero 86
  # obtaining the items for that specific player:
  # each player has 6 items
  ItemBuilds86[i,2] <- Items[i,(col_id-1)*6+1]
  ItemBuilds86[i,3] <- Items[i,(col_id-1)*6+2]
  ItemBuilds86[i,4] <- Items[i,(col_id-1)*6+3]
  ItemBuilds86[i,5] <- Items[i,(col_id-1)*6+4]
  ItemBuilds86[i,6] <- Items[i,(col_id-1)*6+5]
  ItemBuilds86[i,7] <- Items[i,(col_id-1)*6+6]
}

head(ItemBuilds86)

ItemBuilds86 <- ItemBuilds86%>%
  rename(Item_1 = 'V2',
         Item_2 = 'V3',
         Item_3 = 'V4',
         Item_4 = 'V5',
         Item_5 = 'V6',
         Item_6 = 'V7')



# Now we want to see if players play the Hero in the most successful way: comparing
# the most played items, as well as the most successful items.

TopItems86 <- as.data.frame(table(c(ItemBuilds86$Item_1,
      ItemBuilds86$Item_2,
      ItemBuilds86$Item_3,
      ItemBuilds86$Item_4,
      ItemBuilds86$Item_5,
      ItemBuilds86$Item_6)
```

```
      ))

# I used RDota2 to find the ids for items
install.packages('RDota2')
library(RDota2)
key_actions(action = 'register_key', value = 'F377D81D0C9F9534A8C74FE065B4A495')
item_names <- get_game_items()$content # list df of items and relevant information
head(item_names)

item_names <- item_names %>%
  select(id,localized_name) %>%
  filter(id<278) # Newly added items that weren't available at the time our data was taken

TopItems86 <- TopItems86%>%
  arrange(desc(Freq)) # ranking items

item_names[item_names$id == 3,2]

top6 <- TopItems86[2:7,] # Top 6 most picked items on this hero - 0 just means item slot empty

# Now let's reveal which are the actual items:
for(i in 1:6){
  top6[i,3] <- item_names[item_names$id == top6$Var1[i],2]
}

top6 <- top6 %>%
  rename(Name = 'V3',
         ID = 'Var1')

top6

## Comparing How well the Item Builds did

ItemBuilds86[is.na(ItemBuilds86),]
# Theres 1 missing value for radiant_win, it's 1 out of 13000 observations,
# we can remove it.


ItemBuilds86 <- na.omit(ItemBuilds86)
ItemBuilds86

# Creating a list of vectors containing the items, fusing the builds
build <- c()
build_list <- list()
for(i in 1:dim(ItemBuilds86)[1]){
  for(n in 2:7){
    build[n-1] <- ItemBuilds86[i,n]
  }
  build_list[[i]] <- c(build[order(build)],ItemBuilds86$radiant_win[i])
  # adding the outcome of match at the end of the vector for following analysis
}
```

```r
# initializing vectors
skips <- c()
unique_builds <- c()
build_count <- c()
win <- c()
for(i in 1:dim(ItemBuilds86)[1]){
  if(i %in% skips){# skipping the row if build is the same
    next()
  }
  counts <- 0 # counting how many times build was chosen
  success <- 0 # counting how many wins for the i-th build
  for(n in 1:dim(ItemBuilds86)[1]){
    same <- setequal(build_list[[i]][1:6],build_list[[n]][1:6]) # comparing vectors (builds)
    if(same==TRUE){
      counts<-counts+1
      skips <- c(skips,n) # creating  a vector of already matched builds
    }
    ifelse((same==TRUE & build_list[[n]][7]==1),success <- success+1,success <- success+0)
  }
  unique_builds <- c(unique_builds,i)
  build_count <- c(build_count,counts)
  win <- c(win,success)
}# finally, creating the vectors that will make up the final data frame

Build_success_df <- data.frame(build_ID=unique_builds,build_count=build_count,win_count=win)

write.csv(Build_success_df,'Build_success.csv')
# took a long time to run this loop, no need to do it again
Build_success_df <- read.csv('Build_success.csv')

Build_success_df$win_rate <- Build_success_df$win_count/Build_success_df$build_count
Build_success_df$pick_rate <- Build_success_df$build_count/sum(Build_success_df$build_count)


Most_picked_builds <- Build_success_df %>%
  arrange(desc(pick_rate))

Best_builds_20 <- Most_picked_builds[1:10,c('build_ID','win_rate')]%>%
  arrange(desc(win_rate))
  # out of the most picked builds, I checked
# which were the most successful.
ID <- Best_builds_20[1,1]

items <- build_list[[ID]][1:6]

best_items <- data.frame(ID=items)

for(i in 1:6){
  if(best_items$ID[i]==0){
    best_items[i,2] <- 'Empty Slot' # the item slot is empty (i.e. no items)
    next()
  }
  best_items[i,2] <- item_names[item_names$id == best_items$ID[i],2]
```

```
}

best_items <- best_items%>%
  rename(Item='V2')
```