Data Manipulation: World of Warcraft

Rolandas Leonovas

Project Aim

The aim of this project was to offer valuable insight into World of Warcraft in 2008 and how players continuously played through the game and evolved with it.
Not to mention the valuable insight into how players behaved throughout the game as well.

Data Source

www.kaggle.com. (2019). *World of Warcraft Avatar History*. [online] Available at: https://www.kaggle.com/datasets/mylesoneill/warcraft-avatar-history?datasetId=42&sortBy=voteCount&select=wowah_data.csv.

Cleaning The Data

```
rawwow <- tbl_df(fread("C:/Users/sekly/Desktop/year 4 uni/Coursework 2/data/wowah_data.csv"))

u.chars <- sample(unique(rawwow$char), 37354) # used to get a small sample (in this case we work with all data)
wow <- filter(rawwow, char %in% u.chars)
wow$timestamp <- mdy_hms(wow$timestamp)
wow <- arrange(wow, timestamp)
wow$current.date <- as.Date(wow$timestamp)
wow$hour <- hour(wow$timestamp)
wow$month <- format(wow$current.date, "%b")
wow$month.idx <- month(wow$current.date)
```

First you load the data and call it to see what is missing.

You then examine and check what you need to change to fit your needs, in my case the dates weren't ideal so those were changed to match the US format as that is where the data was from to avoid any data inconsistencies.

```
wow <- wow %>% group_by(char) %>% mutate(activation.date = min(current.date),

dsi = as.integer(difftime(current.date, activation.date, units = "days"))) %>% group_by()
```

You then need a way to distinguish of when the character first appeared in the game which is done by taking the first date for a unique character and inputting it into a new column

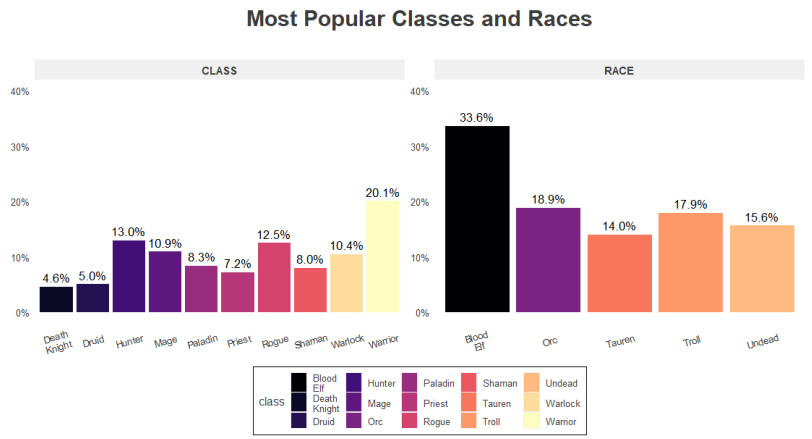
For consistencies sake a theme is very useful which allows for you to create nice and clean visualisations.

```
fte_theme <- function() {

  # Generate the colors for the chart procedurally with RColorBrewer
  palette <- brewer.pal("Greys", n=9)
  color.background = palette[2]
  color.grid.major = palette[3]
  color.axis.text = palette[6]
  color.axis.title = palette[7]
  color.title = palette[9]

  # Begin construction of chart
  theme_bw(base_size=9) +

    # Set the entire chart region to a light gray color
    theme(panel.background=element_rect(fill=color.background, color=color.background)) +
    theme(plot.background=element_rect(fill=color.background, color=color.background)) +
    theme(panel.border=element_rect(color=color.background)) +

    # Format the grid
    theme(panel.grid.major=element_line(color=color.grid.major,size=.50)) +
    theme(panel.grid.minor=element_blank()) +
    theme(axis.ticks=element_blank()) +

    # Format the legend, but hide by default
    theme(legend.position="none") +
    theme(legend.background = element_rect(fill=color.background)) +
    theme(legend.text = element_text(size=7,color=color.axis.title)) +

    # Set title and axis labels, and format these and tick marks
    theme(plot.title=element_text(color=color.title, size=10, vjust=1.25)) +
    theme(axis.text.x=element_text(size=7,color=color.axis.text)) +
    theme(axis.text.y=element_text(size=7,color=color.axis.text)) +
    theme(axis.title.x=element_text(size=8,color=color.axis.title, vjust=0)) +
    theme(axis.title.y=element_text(size=8,color=color.axis.title, vjust=1.25)) +
    theme(plot.title=element_text(hjust=0.5))+

    # Plot margins
    theme(plot.margin = unit(c(0.35, 0.2, 0.3, 0.35), "cm"))
}

gg_color_hue <- function(n) {
  hues = seq(15, 375, length = n + 1)
  hcl(h = hues, l = 65, c = 100)[1:n]
}
```

For the first visualisation which is just the percentages of what class and races the players were you need to separate the data into manageable pieces by creating a separate data frame with just the data you need.

```
pop.raceclass <- wow %>% group_by(race, charclass) %>% summarise(num_chars = n_distinct(char)) %>% group_by()
pop.races <- pop.raceclass %>% group_by(race) %>% summarise(num_chars = sum(num_chars)) %>% mutate(type = "RACE") %>% rename(class = race)
pop.classes <- pop.raceclass %>% group_by(charclass) %>% summarise(num_chars = sum(num_chars)) %>% mutate(type = "CLASS") %>% rename(class = charclass)
pop.raceclass.sparse <- bind_rows(pop.races, pop.classes)
pop.raceclass.sparse <- pop.raceclass.sparse %>% group_by(type) %>% mutate(idx = num_chars / sum(num_chars)) %>% group_by()
pop.raceclass.sparse$class <- gsub(" ", "\n", pop.raceclass.sparse$class)
```

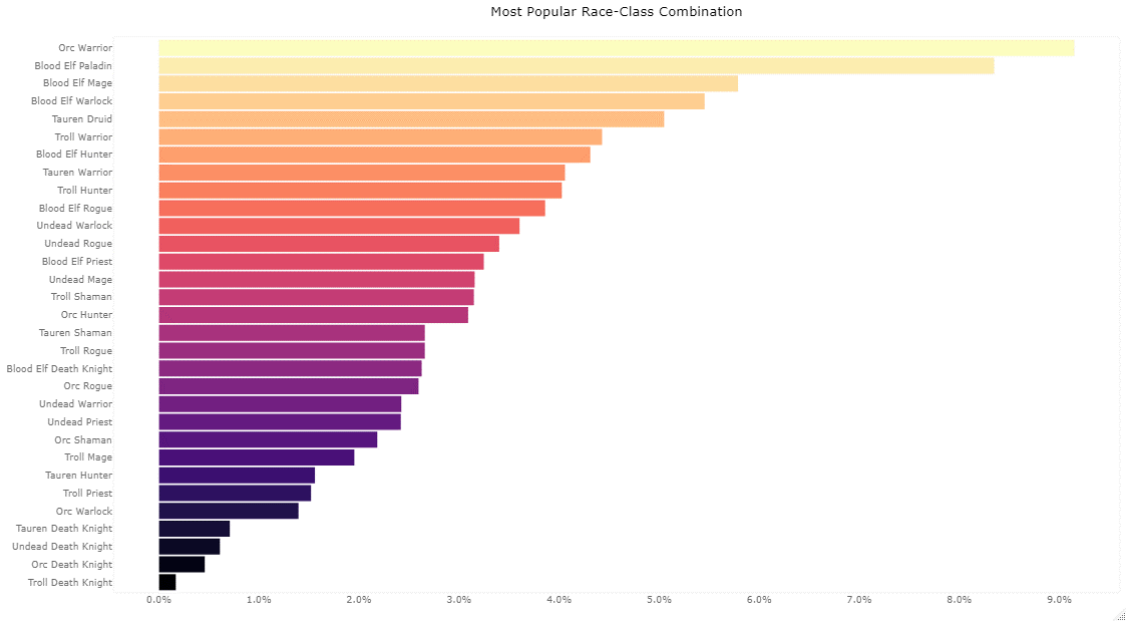
First and Second Visualisation



Second Visualisation's Cleaning

```
pop.raceclass.combine <- mutate(pop.raceclass, class = paste0(race, " ", charclass), idx = num_chars / sum(num_chars)) %>% arrange(desc(idx)) %>% mutate(rank = 1:nrow(pop.raceclass))
pop.raceclass.combine <- mutate(pop.raceclass.combine, percent = idx*100)
pop.raceclass.combine$class <- factor(pop.raceclass.combine$class, levels = pop.raceclass.combine$class[order(pop.raceclass.combine$rank, decreasing= TRUE)])
```

For the second one it is a kind of continuation of the previous visualisation that will allow me to gather more in-depth information that would help me expand on my thoughts. So in this case you need to combine both the character race and class combinations into one data frame along with the number of characters used to allow for a precise number in the visualisation.



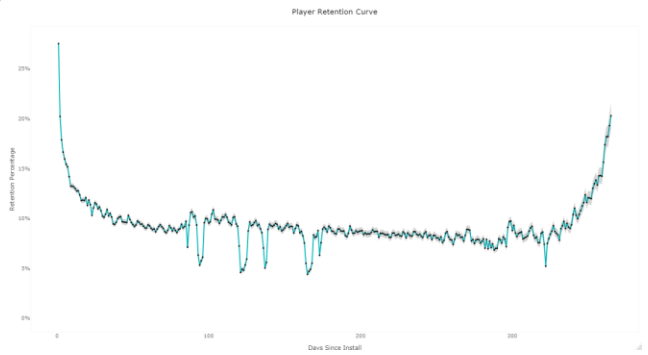
Third and Fourth Visualisation

Data Manipulation

```
cols <- gg_color_hue(4)

min.date <- min(wow$activation.date)
max.date <- max(wow$activation.date)

base <- tbl_df(data.frame(activation.date = seq.Date(min.date, max.date, "days")))
u.chars <- unique(wow$char)
```

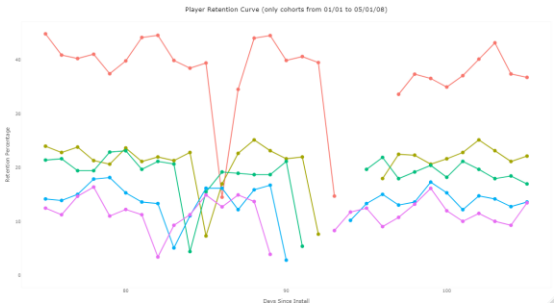
For the retention curve I had to again change the data accordingly to so I could make it viewable and easily understandable.
Here I separate the activation dates so offer minimum and maximum values for the entire X axis and then input into a data frame along with a new data frame for unique WoW characters.

```
retention.cohortly <- wow %>% filter(char %in% u.chars) %>% group_by(activation.date, dsi) %>% summarise(yes = n_distinct(char)) %>% mutate(no = yes[dsi == 0] - yes, size = yes + no, idx = yes / size) %>% group_by()

retention <- retention.cohortly %>% group_by(dsi) %>% summarise(yes = sum(yes), no = sum(no), size = yes + no, idx = 100*yes/size) %>% filter(dsi > 0)

retention.cohortly <- rename(retention.cohortly, Retention = idx, DSI = dsi) %>% mutate(Retention = 100*Retention)
retention <- rename(retention, DSI = dsi, Retention = idx)

hpdi <- sapply(retention$DSI, function(dsi) {
  row <- filter(retention, DSI == dsi)
  x <- rbeta(1e5, row$yes+1, row$no+1)
  HPDinterval(as.mcmc(x), prob = .9)*100
})
hpdi <- tbl_df(data.frame(t(hpdi)))
colnames(hpdi) <- c("Lower", "Upper")
retention <- tbl_df(cbind(retention, hpdi))
```



Then all you need to do is setup the retention curve axis according to the your data and create your ggplotly and the third one is done.

```
retentionRange <- function(dates, base, min.dsi, max.dsi, point=T) {
  df <- left_join(base, retention.cohortly, c("activation.date", "dsi" = "DSI"))
  df <- filter(df, dsi >= min.dsi, dsi <= max.dsi)
  df <- filter(df, activation.date %in% dates) %>% mutate(activation.date = factor(as.character(activation.date)))

  from <- format(min(dates), "%d/%m")
  to <- format(max(dates), "%d/%m/%y")

  g <- ggplot(data = df, aes(dsi, Retention, colour = activation.date, label = yes)) +
    fte_theme() +

    theme(
      panel.background = element_rect(fill='transparent'),
      plot.background = element_rect(fill='transparent', color=NA),
      panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      legend.background = element_rect(fill='transparent'),
      legend.box.background = element_rect(fill='transparent')
    )+
    expand_limits(y = 0) +
    labs(title = paste0("Player Retention Curve (only cohorts from ", from, " to ", to, ")"), x = "Days Since Install", y = "Retention Percentage") +
    geom_line()
  if (point) {
    g <- g + geom_point()
  }
  ggplotly(g)
}
```

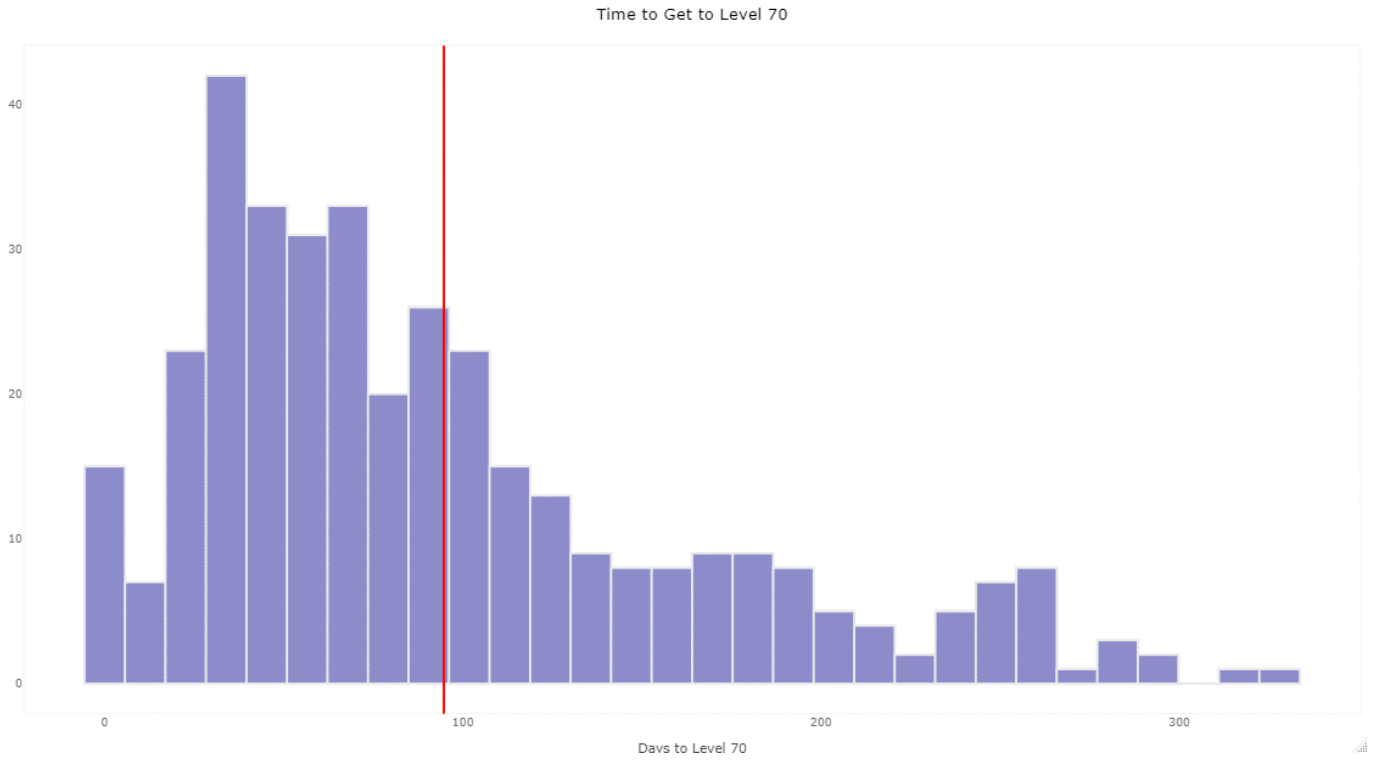


The fourth one is similar to the last one the only difference being you're pinpointing a specific date range.

Fifth Visualisation

```
u.70chars <- unique(filter(wow, level == 70)$char)
d70 <- filter(wow, char %in% u.70chars) %>%
  group_by(char) %>%
  summarise(
    max.date = max(current.date),
    min.date = min(current.date),
    min.level = min(level),
    date70 = min(current.date[level == 70]),
    days_to_70 = date70-min.date) %>%
  filter(min.level == 1) %>%
  group_by(days_to_70) %>%
  summarise(users = n_distinct(char)) %>%
  arrange(days_to_70)

a70 <- data.frame(days = rep(d70$days_to_70, d70$users))
mean_time_to_70 <- sum(d70$users*d70$days_to_70)/sum(d70$users)
```

For the fifth one I had to filter out the unique level 70 characters, again do a similar thing to the dates in the way it was done in retention except you are now calculating the amount of time it took to hit 70.



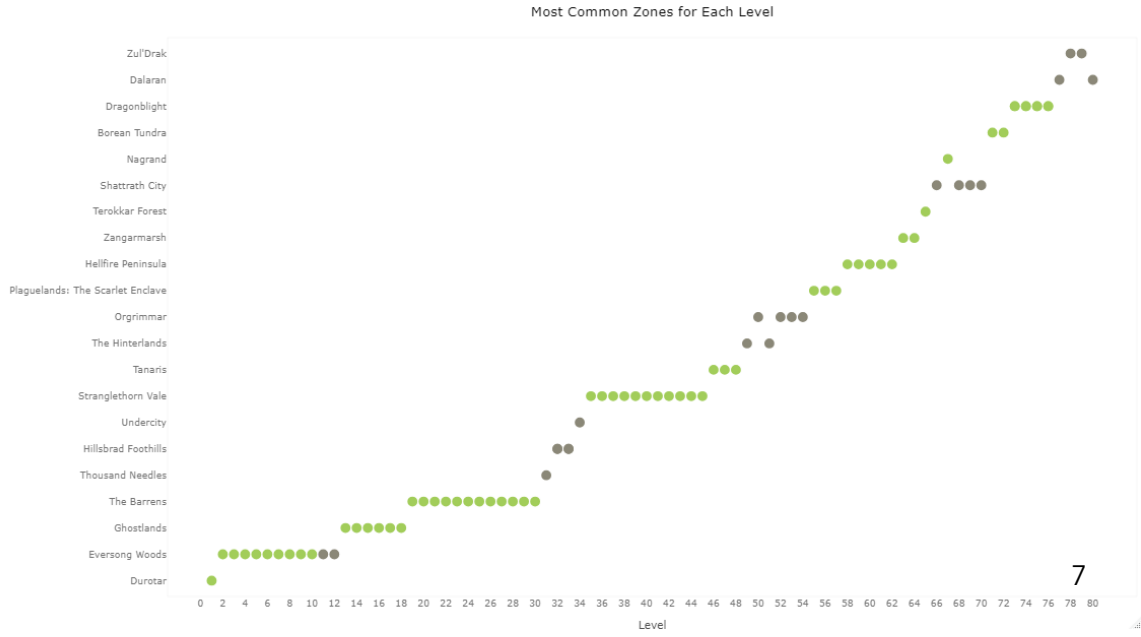
Sixth Visualisation

```
# Popular Zones
common.zones <- tbl_df(data.frame(rawzones))
#common.zones$lv <- common.zones$level
#common.zones <- separate(common.zones, lv, c("start", "end"), "-")
knitr::kable(head(common.zones))

common.zones
common.zones <- subset(common.zones, Type!="City" & Type!="Dungeon" & Type!="Battleground" & Type!="Sea" & Type!="Arena")


# Alliance data is insufficient so we don't want to use them
cz <- filter(common.zones, Controlled %in% c("Horde", "Contested", "PvP"))
zones <- wow %>% group_by(level, zone) %>% summarise(num_chars = n_distinct(char)) %>% group_by() %>% arrange(level, desc(num_chars))
f.zones <- zones %>% group_by(level) %>% filter(row_number() == 1) %>% group_by()

cz$level <-  paste(cz$Min_rec_level, "-", cz$Max_rec_level)

names(cz) <- tolower(names(cz))
```

This was a difficult one as you needed to separate and filter out the zones you did not need to receive accurate minimum and maximum recommended levels and the condense it only to the Horde faction.

```
f.zones

f.zones$zone.assert <- as.character(apply(f.zones, 1, function(x) {
  zone <- as.character(x[2])
  level <- as.numeric(x[1])

  tmp <- cz[cz$zone_name == zone,]
  start <- as.numeric(tmp$min_rec_level)
  end <- as.numeric(tmp$max_rec_level)
  if (nrow(tmp) == 0) {
    0
  } else {
    ifelse((level >= start) & (level <= end), 1, 0)
  }
}))



f.zones <- left_join(f.zones, select(cz, zone, LevelRange = level), by = "zone")
f.zones$LevelRange[is.na(f.zones$LevelRange)] <- "-"

f.zones
names(f.zones) <- tolower(names(f.zones))


f.zones$Recommended <- ifelse(f.zones$zone.assert == 1, "Yes", "No")
f.zones$zone <- factor(f.zones$zone, levels = unique(f.zones$zone[order(f.zones$level)]))
f.zones <- rename(f.zones, Level = level, Zone = zone)
```

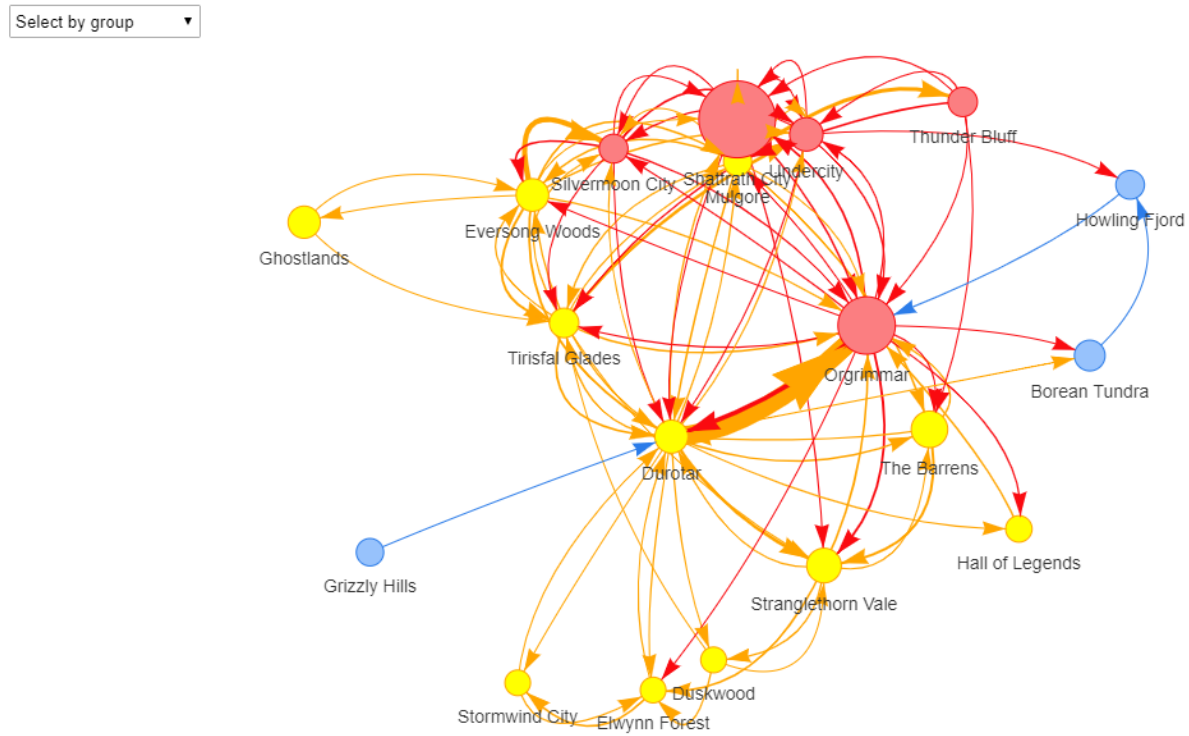
Which you then need to create a separate column to count if a zone is recommended to a player or not.



Seventh Visualisation

```
# Popular Zones
common.zones <- tbl_df(data.frame(rawzones))
#common.zones$lv <- common.zones$level
#common.zones <- separate(common.zones, lv, c("start", "end"), "-")
knitr::kable(head(common.zones))

capital <- c("Shattrath City","Orgrimmar","Silvermoon City","Thunder Bluff","Undercity","Dalaran")

battleground <- c("Warsong Gulch", "Arathi Basin", "Alterac Valley", "Eye of the Storm","Wintergrasp", "Strand of the

Northrend <- c("Howling Fjord","Borean Tundra","Coldarra","Dragonblight","Grizzly Hills","Zul'Drak","Sholazar Basin",

Outland <- c("Hellfire Peninsula","Zangarmarsh","Terokkar Forest","Nagrand","Blade's Edge Mountains","Netherstorm","Sh
```

Same way you clean up the recommended levels you do with the zones by picking out the most popular ones and relevant ones to avoid visualisation bloat.



Eight
Visualisation

```
playTime <- function(facet.num) {
  playtime <- wow %>% group_by(current.date, hour) %>% summarise(num=n_distinct(char)) %>% group_by()

  min.date <- min(wow$current.date)
  max.date <- max(wow$current.date)

  base <- tbl_df(data.frame(current.date = rep(seq.Date(min.date, max.date, "days"), each=24), hour = 0:23))
  playtime <- left_join(base, playtime, by = c("current.date", "hour"))
  playtime <- mutate(playtime, facet = ifelse(current.date <= as.Date("2008-06-30"), 1, 2))
```

For the final one you have already done all the ground work so by looking back you can just setup the dates that fit your needs and create a new data frame for hours that will allow you to check the most played hour slots.