

**SCHOOL OF COMPUTING SCIENCE & ENGINEERING**

**GALGOTIAS UNIVERSITY**

**PROJECT REPORT ON**

**Video Streaming App using Data Analysis**

**UNDER THE SUPERVISION OF**

**(Mr.** **Thirunavukkarasu K)**

**SUBMITTED BY:**

**ABHAY SHARMA (1413108001)**

**TAUSEEF AHMAD HASHMI (1413109002)**

**(B.TECH. CSE IBM, BATCH -7, SEM -7)**

**ACKNOWLEDGEMENT**

We would like to take this opportunity to express my profound sense of gratitude and respect to all those who helped us throughout the duration of this project. GALGOTIAS UNIVERSITY in particular has been the source of inspiration for us.

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to all of them.

We are highly indebted to Mr. Thirunavukkarasu K for his guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in completing the project.

We would also like to thank **Mr. Ajay Shankar**  Program Chair (CSE+IBM) & **Dr.Parma Nand**, Professor & Dean, School of Computing Science & Engineering for providing us the opportunity to work on this project.

**-ABHAY SHARMA**

**-**TAUSEEF AHMAD HASHMI

1. **Introduction**

Video Streaming app is recommender system based on Data Analytics by using MovieLens Data SetThis is a report on the movieLens dataset available "https://grouplens.org/datasets/movielens/”. MovieLens itself is a research site run by GroupLens Research group at the University of Minnesota. The first automated recommender system was developed there in 1993.

**2. Objectives**

The MovieLens dataset is most often used for the purpose of recommender systems which aim to predict user movie ratings based on other users’ ratings. In other words we expect that users with similar taste will tend to rate movies with high correlation.

We will make Android app which will use this Data Set for recommending movies to user.

However, in this analysis we will try to explore the movies themselves. Hopefully it will give us an interesting insight into the history of cinematography.

**3.** **Current Platforms**

* TasteKid
* Movielens
* Netflix

All current platform are limited to their obsolete services. Also there is no current platform which give the smart analysis like power analysis, session-time analysis, Data Analysis and Predictive Analysis.

**Limitations in current platforms**

* Currently there is no Analysis on movie dataset used by Customers
* No segment based analytics is available till now.
* A very few analytics platform are real time.

**Our Proposal**

We will provide the real analytical data to the customers about what type of movie they should watch or not.

In Other Words, We will recommend movies to users through our video Streaming App.

**4. Types of Analytics and Processing**

Working of this dataset needs to be various types of Analysis and Data Processing. We also need to setup platform for each. Some of them are-

**Data mining**

Data mining is basically be used to mined out usable data from Data Set.

**Data Cleaning**

We will also perform necessary cleaning and some transformations so that the data better suits our needs.

**Data Exploration**

Data Exploration to be done for particular type of analysis after Data Mining.

**Language/Tools used**

* R Programming
* Python
* Microsoft R Open distribution
* Windows & Linux(Ubuntu Preferred)
* Git-Github(svn-svn repository)
* Rapid Miner
* Google Cloud Prediction API
* R packages
* Java & Android SDK

**5. Bibliography**

**R Programming and its Package**

An introduction to R packages based on 11 of the most frequently asked user questions.

R packages are collections of functions and data sets developed by the community. They increase the power of R by improving existing base R functionalities, or by adding new ones. For example, if you are usually working with data frames, probably you will have heard about [dplyr](https://github.com/hadley/dplyr) or [data table](https://github.com/Rdatatable/data.table/wiki), two of the most popular R packages.

But imagine that you'd like to do some natural language processing of Korean texts, extract weather data from the web, or even estimate actual evapotranspiration using land surface energy balance models, R packages got you covered! Recently, the official repository (CRAN) reached 10,000 packages published, and many more are publicly available through the internet.

**Package Used**

For this analysis the Microsoft R Open distribution was used. The reason for this was its multithreaded performance as described here. Most of the packages that were used come from the tidyverse - a collection of packages that share common philosophies of tidy data. The tidy text and word cloud packages were used for some text processing. Finally, the doMC package was used to embrace the multithreading in some of the custom functions which will be described later.

**List of Packages to be installed**

* library(checkpoint)
* checkpoint("2017-01-15", auto.install.knitr=T)
* library(tidyverse)
* library(lubridate)
* library(stringr)
* library(rvest)
* library(XML)
* library(tidytext)
* library(wordcloud)
* library(doMC)
* registerDoMC()
* set.seed(1234)

**Data Selection**

We will use MovieLens 20M datasets here updated October 2016.

Data Contain primary four CSV files i.e, movie.csv, tag.csv, link.csv, rating.csv

**6. Package Used**

Most of the packages that were used come from the tidyverse - a collection of packages that share common philosophies of tidy data. The tidytext and wordcloud packages were used for some text processing. Finally, the doMC package was used to embrace the multithreading in some of the custom functions which will be described later.

In [1]:

*# Load the packages -------------------------------------------------------*

library(tidyverse)

library(lubridate)

library(stringr)

library(rvest)

library(XML)

library(tidytext)

library(wordcloud)

library(doMC)

registerDoMC()

set.seed(1234)

Loading tidyverse: ggplot2

Loading tidyverse: tibble

Loading tidyverse: tidyr

Loading tidyverse: readr

Loading tidyverse: purrr

Loading tidyverse: dplyr

Conflicts with tidy packages ---------------------------------------------------

filter(): dplyr, stats

lag(): dplyr, stats

Attaching package: ‘lubridate’

The following object is masked from ‘package:base’:

date

Loading required package: xml2

Attaching package: ‘rvest’

The following object is masked from ‘package:readr’:

guess\_encoding

Attaching package: ‘XML’

The following object is masked from ‘package:rvest’:

xml

Loading required package: RColorBrewer

Loading required package: foreach

Attaching package: ‘foreach’

The following objects are masked from ‘package:purrr’:

accumulate, when

Loading required package: iterators

Loading required package: parallel

## 7. Dataset Description

The dataset is available in several snapshots. The ones that were used in this analysis were Latest Datasets - both full and small (for web scraping). They were last updated in October 2016.

### **Loading the Dataset**

The dataset is split into four files (genome-scores.csv and genome-tags.csv were omitted for this analysis)- movies.csv, ratings.csv, links.csv and tags.csv. We will iteratively load the files into the workspace using read\_csv() function and assign variable names accordingly. The read\_csv() function is very convenient because it automagically guesses column types based on the first 1000 rows. And more importantly it never converts strings to factors. Never.

Finally we will check object sizes to see how big is the dataset.

In [2]:

dataset\_files <- c("movie", "rating", "link", "tag")

suffix <- ".csv"

**for** (f **in** dataset\_files) {

path <- file.path("../input", paste0(f, suffix))

assign(f, read\_csv(path, progress = F))

print(paste(f, "object size is", format(object.size(get(f)),units="Mb")))

}

Parsed with column specification:

cols(

movieId = col\_integer(),

title = col\_character(),

genres = col\_character()

)

Warning message:

“10 parsing failures.

row col expected actual

7482 title delimiter or quote \

7482 title delimiter or quote 0

11678 title delimiter or quote \

11678 title delimiter or quote G

11678 title delimiter or quote \

..... ..... .................. ......

See problems(...) for more details.

”

[1] "movie object size is 2.7 Mb"

Parsed with column specification:

cols(

userId = col\_integer(),

movieId = col\_integer(),

rating = col\_double(),

timestamp = col\_datetime(format = "")

)

[1] "rating object size is 457.8 Mb"

Parsed with column specification:

cols(

movieId = col\_integer(),

imdbId = col\_integer(),

tmdbId = col\_integer()

)

[1] "link object size is 0.3 Mb"

Parsed with column specification:

cols(

userId = col\_integer(),

movieId = col\_integer(),

tag = col\_character(),

timestamp = col\_datetime(format = "")

)

Warning message:

“794 parsing failures.

row col expected actual

836 tag delimiter or quote \

836 tag delimiter or quote '

842 tag delimiter or quote \

842 tag delimiter or quote H

842 tag delimiter or quote \

... ... .................. ......

See problems(...) for more details.

”

[1] "tag object size is 13 Mb"

The biggest data frame is rating - 457.8 Mb - it contains movie ratings from movieLens users. Next we will see what kind of data we deal with.

## 8. Data Cleaning

In this section we will take the first look at the loaded data frames. We will also perform necessary cleaning and some transformations so that the data better suits our needs. First, let’s look at the ratings table.

In [3]:

*# Clean ratings*

glimpse(rating)

Observations: 20,000,263

Variables: 4

$ userId <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

$ movieId <int> 2, 29, 32, 47, 50, 112, 151, 223, 253, 260, 293, 296, 318...

$ rating <dbl> 3.5, 3.5, 3.5, 3.5, 3.5, 3.5, 4.0, 4.0, 4.0, 4.0, 4.0, 4....

$ timestamp <dttm> 2005-04-02 23:53:47, 2005-04-02 23:31:16, 2005-04-02 23:...

We have 24 million rows and 4 columns. It seems that only timestamp column need to be converted. We will create new data frame that we will work on and preserve the original data frame (treat it as read-only).

In [4]:

ratings\_df <- rating %>%

mutate(timestamp = as\_datetime(timestamp))

summary(ratings\_df)

userId movieId rating

Min. : 1 Min. : 1 Min. :0.500

1st Qu.: 34395 1st Qu.: 902 1st Qu.:3.000

Median : 69141 Median : 2167 Median :3.500

Mean : 69046 Mean : 9042 Mean :3.526

3rd Qu.:103637 3rd Qu.: 4770 3rd Qu.:4.000

Max. :138493 Max. :131262 Max. :5.000

timestamp

Min. :1995-01-09 11:46:44

1st Qu.:2000-08-20 18:55:45

Median :2004-12-20 15:18:06

Mean :2004-11-20 02:32:01

3rd Qu.:2008-11-02 16:11:57

Max. :2015-03-31 06:40:02

Ok, looks like there is no missing data. We can also see that the ratings range from 0.5 to 5 and that they are timestamped. Now, let’s look into the movies data frame.

In [5]:

glimpse(movie)

Observations: 27,278

Variables: 3

$ movieId <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ...

$ title <chr> "Toy Story (1995)", "Jumanji (1995)", "Grumpier Old Men (19...

$ genres <chr> "Adventure|Animation|Children|Comedy|Fantasy", "Adventure|C...

There are over 40 thousand movies and 3 columns. Most of the movies have their debut year added to their names - we want to extract this into separate columns. Genres columns contains multiple categories per row - we want to have them separated into one category per row. We will deal with this later.

In [6]:

movies\_df <- movie %>%

*# trim whitespaces*

mutate(title = str\_trim(title)) %>%

*# split title to title, year*

extract(title, c("title\_tmp", "year"), regex = "^(.\*) \\(([0-9 \\-]\*)\\)$", remove = F) %>%

*# for series take debut date*

mutate(year = if\_else(str\_length(year) > 4, as.integer(str\_split(year, "-", simplify = T)[1]), as.integer(year))) %>%

*# replace title NA's with original title*

mutate(title = if\_else(is.na(title\_tmp), title, title\_tmp)) %>%

*# drop title\_tmp column*

select(-title\_tmp) %>%

*# generic function to turn (no genres listed) to NA*

mutate(genres = if\_else(genres == "(no genres listed)", `is.na<-`(genres), genres))

Warning message in replace\_with(out, !condition & !is.na(condition), false, "`false`"):

“NAs introduced by coercion”

Here we extracted the movie debut year using extract() function from tidyr package. For the case of movie series where year has “yyyy-yyyy” format we take the first date. In the last line we replaced the string “(no genres listed)” with NA value to make further processing easier. There are also some warnings suggesting that missing values appeared. We’ll check that now.

In [7]:

*# Check NA's*

na\_movies <- movies\_df %>%

filter(is.na(title) | is.na(year))

glimpse(na\_movies)

Observations: 28

Variables: 4

$ movieId <int> 8359, 26815, 40697, 79607, 87442, 89932, 89971, 89973, 9045...

$ title <chr> "Skokie (1981)", "Deadly Advice(1994)", "Babylon 5", "Milli...

$ year <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...

$ genres <chr> "Drama", "Comedy|Drama", "Sci-Fi", "Action|Drama|Sci-Fi|Thr...

Seems that warnings appeared, because some of the movies do not have their debut year. We will ignore those movies in further analysis as there aren’t many of them.

In [8]:

summary(movies\_df)

movieId title year genres

Min. : 1 Length:27278 Min. :1891 Length:27278

1st Qu.: 6931 Class :character 1st Qu.:1976 Class :character

Median : 68068 Mode :character Median :1998 Mode :character

Mean : 59855 Mean :1989

3rd Qu.:100293 3rd Qu.:2008

Max. :131262 Max. :2015

NA's :28

Let’s check the tags data frame now.

In [9]:

glimpse(tag)

Observations: 465,426

Variables: 4

$ userId <int> 18, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 6...

$ movieId <int> 4141, 208, 353, 521, 592, 668, 898, 1248, 1391, 1617, 169...

$ tag <chr> "Mark Waters", "dark hero", "dark hero", "noir thriller",...

$ timestamp <dttm> 2009-04-24 18:19:40, 2013-05-10 01:41:18, 2013-05-10 01:...

Seems that only timestamp needs to be converted.

In [10]:

tags\_df <- tag %>%

mutate(timestamp = as\_datetime(timestamp))

summary(tags\_df)

userId movieId tag

Min. : 18 Min. : 1 Length:465426

1st Qu.: 28780 1st Qu.: 2571 Class :character

Median : 70201 Median : 7373 Mode :character

Mean : 68717 Mean : 32626

3rd Qu.:107322 3rd Qu.: 62249

Max. :138472 Max. :131258

timestamp

Min. :2005-12-24 13:00:10

1st Qu.:2009-06-14 19:21:10

Median :2011-04-08 21:07:20

Mean :2011-02-26 09:55:46

3rd Qu.:2013-04-17 18:29:50

Max. :2015-03-31 03:09:12

No missing values, we can continue to the links data frame.

In [11]:

glimpse(link)

Observations: 27,278

Variables: 3

$ movieId <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ...

$ imdbId <int> 114709, 113497, 113228, 114885, 113041, 113277, 114319, 112...

$ tmdbId <int> 862, 8844, 15602, 31357, 11862, 949, 11860, 45325, 9091, 71...

We have 40,000 rows with ids to imdb and tmdb websites. We will use them later for some web scraping.

Ok, we are now done with data cleaning. Let’s go deeper into the data exploration.

## 9. Data Exploration

In this part we will try to explore the dataset and reveal some interesting facts about the movie business.

### **How many movies were produced per year?**

The first question that may be asked is how many movies were produced year by year. We can easily extract this information from the movies\_df data frame.

In [12]:

*# Number of movies per year/decade*

movies\_per\_year <- movies\_df %>%

na.omit() %>% *# omit missing values*

select(movieId, year) %>% *# select columns we need*

group\_by(year) %>% *# group by year*

summarise(count = n()) %>% *# count movies per year*

arrange(year)

print(movies\_per\_year)

# A tibble: 116 × 2

year count

<int> <int>

1 1894 2

2 1895 2

3 1896 2

4 1898 2

5 1899 1

6 1900 1

7 1901 1

8 1902 1

9 1903 1

10 1905 1

# ... with 106 more rows

There are some years that are missing, probably there were no movies produced in the early years. We can easily fix missing values using complete() function from the tidyr package.

In [13]:

*# fill missing years*

movies\_per\_year <- movies\_per\_year %>%

complete(year = full\_seq(year, 1), fill = list(count = 0))

print(movies\_per\_year)

# A tibble: 122 × 2

year count

<dbl> <dbl>

1 1894 2

2 1895 2

3 1896 2

4 1897 0

5 1898 2

6 1899 1

7 1900 1

8 1901 1

9 1902 1

10 1903 1

# ... with 112 more rows

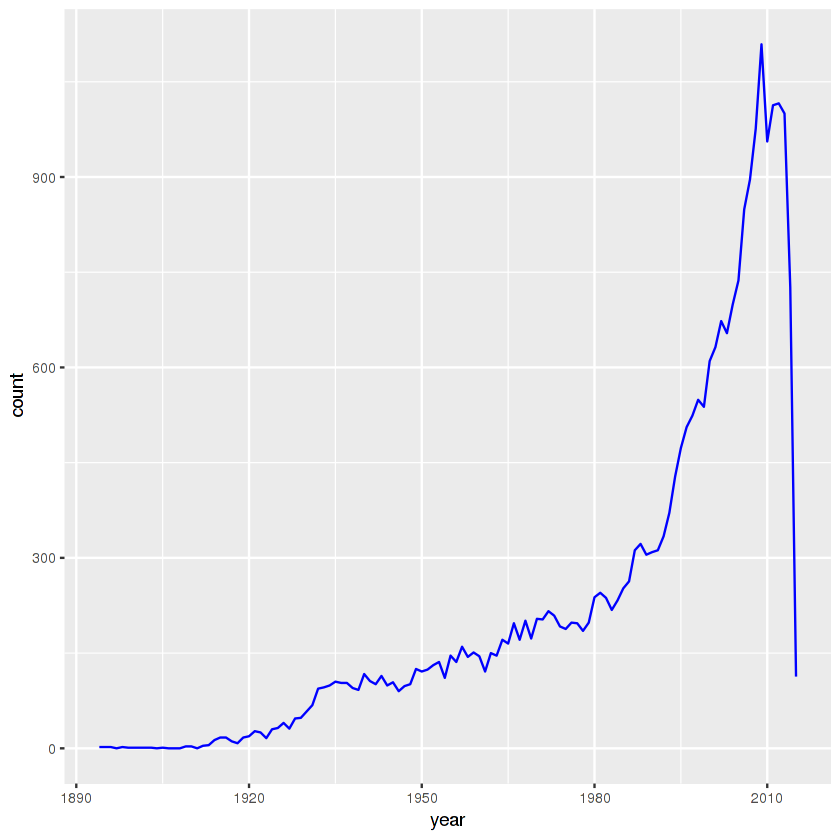
That’s better. Now let’s plot what we have.

In [14]:

movies\_per\_year %>%

ggplot(aes(x = year, y = count)) +

geom\_line(color="blue")



We can see an exponential growth of the movie business and a sudden drop in 2016. The latter is caused by the fact that the data is collected until October 2016 so we don’t have the full data on this year. As for the former, perhaps it was somewhat linked to the beginning of the information era. Growing popularity of the Internet must have had a positive impact on the demand for movies. That is certainly something worthy of further analysis.

### **What were the most popular movie genres year by year?**

We know how many movies were produced, but can we check what genres were popular? We might expect that some events in history might have influenced the movie creators to produce specific genres. First we will check what genres are the most popular in general.

In [15]:

genres\_df <- movies\_df %>%

separate\_rows(genres, sep = "\\|") %>%

group\_by(genres) %>%

summarise(number = n()) %>%

arrange(desc(number))

print(genres\_df)

# A tibble: 20 × 2

genres number

<chr> <int>

1 Drama 13344

2 Comedy 8374

3 Thriller 4178

4 Romance 4127

5 Action 3520

6 Crime 2939

7 Horror 2611

8 Documentary 2471

9 Adventure 2329

10 Sci-Fi 1743

11 Mystery 1514

12 Fantasy 1412

13 War 1194

14 Children 1139

15 Musical 1036

16 Animation 1027

17 Western 676

18 Film-Noir 330

19 <NA> 246

20 IMAX 196

No suprise here. Dramas and comedies are definitely the most popular genres.

In [16]:

*# Genres popularity per year*

genres\_popularity <- movies\_df %>%

na.omit() %>% *# omit missing values*

select(movieId, year, genres) %>% *# select columns we are interested in*

separate\_rows(genres, sep = "\\|") %>% *# separate genres into rows*

mutate(genres = as.factor(genres)) %>% *# turn genres in factors*

group\_by(year, genres) %>% *# group data by year and genre*

summarise(number = n()) %>% *# count*

complete(year = full\_seq(year, 1), genres, fill = list(number = 0)) *# add missing years/genres*

Now we are able to plot the data. For readability we choose 4 genres: animation, sci-fi, war and western movies.

In [17]:

genres\_popularity %>%

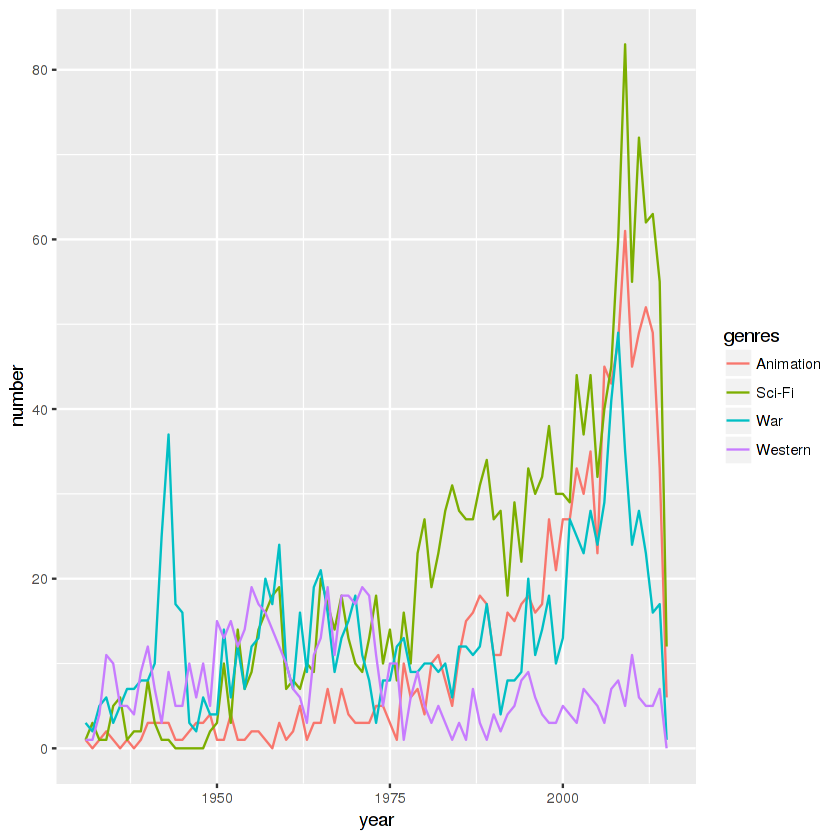
filter(year > 1930) %>%

filter(genres %in% c("War", "Sci-Fi", "Animation", "Western")) %>%

ggplot(aes(x = year, y = number)) +

geom\_line(aes(color=genres)) +

scale\_fill\_brewer(palette = "Paired")



Here we have some interesting observations. First we can notice a rapid growth of sci-fi movies shortly after 1969, the year of the first Moon landing. Secondly, we notice high number of westerns in 1950s and 1960s that was the time when westerns popularity was peaking. Next, we can see the rise of popularity of animated movies, the most probable reason might be the computer animation technology advancement which made the production much easier. War movies were popular around the time when big military conflicts occured - World War II, Vietnam War and most recently War in Afghanistan and Iraq. It’s interesting to see how the world of cinematography reflected the state of the real world.

### **What tags best summarize a movie genre?**

Looking at how each movie genre is tagged by users is a great way to see if a movie genre can be described using just a few words. We’ll explore a selection of movie genres and see if anything interesting pops out.

In [18]:

*# Tags for genres*

genres\_tags <- movies\_df %>%

na.omit() %>%

select(movieId, year, genres) %>%

separate\_rows(genres, sep = "\\|") %>%

inner\_join(tags\_df, by = "movieId") %>%

select(genres, tag) %>%

group\_by(genres) %>%

nest()

### **Action**

In [19]:

*# plot wordcloud per genre*

genre<-"Action"

genre\_words <- genres\_tags %>%

filter(genres == genre) %>%

unnest() %>%

mutate(tag = str\_to\_lower(tag, "en")) %>%

anti\_join(tibble(tag=c(tolower(genre)))) %>%

count(tag)

wordcloud(genre\_words$tag, genre\_words$n, max.words = 50, colors=brewer.pal (8, "Dark2"))

Joining, by = "tag"



### **Comedy**

In [20]:

*# plot wordcloud per genre*

genre<-"Comedy"

genre\_words <- genres\_tags %>%

filter(genres == genre) %>%

unnest() %>%

mutate(tag = str\_to\_lower(tag, "en")) %>%

anti\_join(tibble(tag=c(tolower(genre)))) %>%

count(tag)

wordcloud(genre\_words$tag, genre\_words$n, max.words = 50, colors=brewer.pal(8, "Dark2"))

Joining, by = "tag"



### **Thriller**

In [21]:

*# plot wordcloud per genre*

genre<-"Thriller"

genre\_words <- genres\_tags %>%

filter(genres == genre) %>%

unnest() %>%

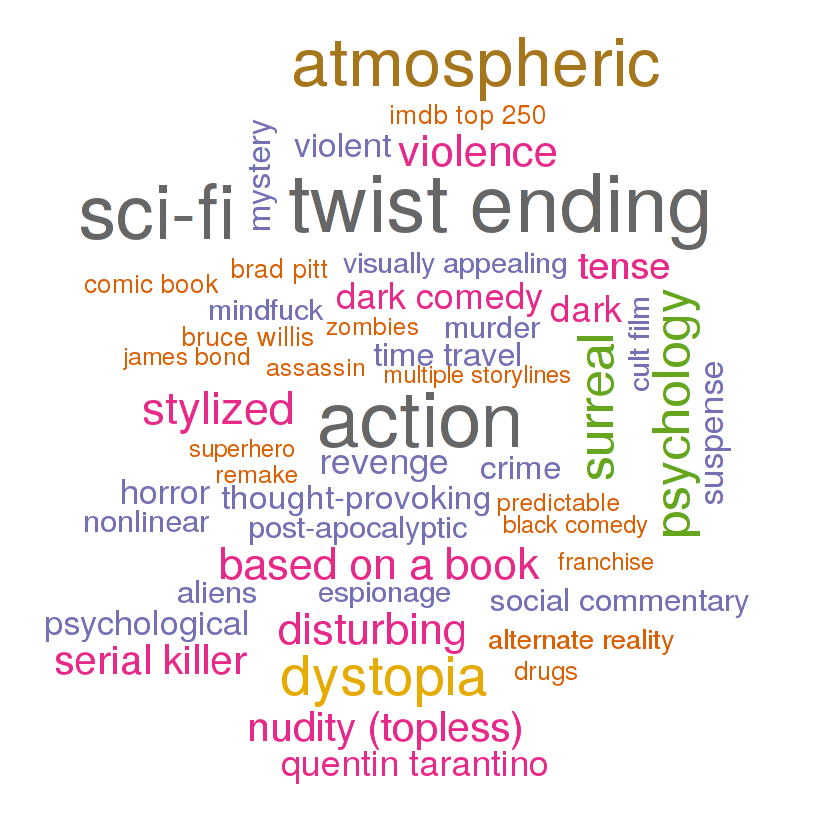
mutate(tag = str\_to\_lower(tag, "en")) %>%

anti\_join(tibble(tag=c(tolower(genre)))) %>%

count(tag)

wordcloud(genre\_words$tag, genre\_words$n, max.words = 50, colors=brewer.pal(8, "Dark2"))

Joining, by = "tag"



### **What were the best movies of every decade (based on users’ ratings)?**

We may wish to see what were the highest rated movies in every decade. First, let’s find average score for each movie.

In [22]:

*# average rating for a movie*

avg\_rating <- ratings\_df %>%

inner\_join(movies\_df, by = "movieId") %>%

na.omit() %>%

select(title, rating, year) %>%

group\_by(title, year) %>%

summarise(count = n(), mean = mean(rating), min = min(rating), max = max(rating)) %>%

ungroup() %>%

arrange(desc(mean))

print(avg\_rating)

# A tibble: 26,468 × 6

title year count mean min max

<chr> <int> <int> <dbl> <dbl> <dbl>

1 1971 2014 1 5 5 5

2 A Blank on the Map 1971 1 5 5 5

3 A Gun for Jennifer 1997 1 5 5 5

4 A Night for Dying Tigers 2010 1 5 5 5

5 Abendland 2011 1 5 5 5

6 Afstiros katallilo 2008 1 5 5 5

7 Al otro lado 2004 1 5 5 5

8 Argentina latente 2007 1 5 5 5

9 B-Side 2013 1 5 5 5

10 Bandaged 2009 1 5 5 5

# ... with 26,458 more rows

That doesn’t look too good. If we sort by average score our ranking will be polluted by movies with low count of reviews. To deal with this issue we will use a weighted average used on IMDB website for their Top 250 ranking. Head [here](https://districtdatalabs.silvrback.com/computing-a-bayesian-estimate-of-star-rating-means) for more details.

In [23]:

*# R = average for the movie (mean) = (Rating)*

*# v = number of votes for the movie = (votes)*

*# m = minimum votes required to be listed in the Top 250*

*# C = the mean vote across the whole report*

weighted\_rating <- **function**(R, v, m, C) {

**return** (v/(v+m))\*R + (m/(v+m))\*C

}

avg\_rating <- avg\_rating %>%

mutate(wr = weighted\_rating(mean, count, 500, mean(mean))) %>%

arrange(desc(wr)) %>%

select(title, year, count, mean, wr)

print(avg\_rating)

# A tibble: 26,468 × 5

title year count mean wr

<chr> <int> <int> <dbl> <dbl>

1 Pulp Fiction 1994 67310 4.174231 0.9926265

2 Forrest Gump 1994 66172 4.029000 0.9925006

3 Shawshank Redemption, The 1994 63366 4.446990 0.9921711

4 Silence of the Lambs, The 1991 63299 4.177057 0.9921629

5 Jurassic Park 1993 59715 3.664741 0.9916964

6 Star Wars: Episode IV - A New Hope 1977 54502 4.190672 0.9909094

7 Braveheart 1995 53769 4.042534 0.9907866

8 Terminator 2: Judgment Day 1991 52244 3.931954 0.9905202

9 Matrix, The 1999 51334 4.187186 0.9903538

10 Schindler's List 1993 50054 4.310175 0.9901096

# ... with 26,458 more rows

That’s better. Movies with more good reviews got higher score. Now let’s findthe best movie for every decade since the beginning of cinematography.

In [24]:

*# find best movie of a decade based on score*

*# heavily dependent on the number of reviews*

best\_per\_decade <- avg\_rating %>%

mutate(decade = year %/% 10 \* 10) %>%

arrange(year, desc(wr)) %>%

group\_by(decade) %>%

summarise(title = first(title), wr = first(wr), mean = first(mean), count = first(count))

print(best\_per\_decade)

# A tibble: 13 × 5

decade title

<dbl> <chr>

1 1890 Dickson Experimental Sound Film

2 1900 The Kiss

3 1910 Frankenstein

4 1920 Cabinet of Dr. Caligari, The (Cabinet des Dr. Caligari., Das)

5 1930 All Quiet on the Western Front

6 1940 Pinocchio

7 1950 Cinderella

8 1960 Psycho

9 1970 M\*A\*S\*H (a.k.a. MASH)

10 1980 Star Wars: Episode V - The Empire Strikes Back

11 1990 Dances with Wolves

12 2000 Gladiator

13 2010 Inception

# ... with 3 more variables: wr <dbl>, mean <dbl>, count <int>

Here we can notice the disadvantage of weighted ratings - low score for old movies. That’s not necessarily caused by movies quality, rather small number of viewers.

### **What were the best years for a genre (based on users’ ratings)?**

In [25]:

genres\_rating <- movies\_df %>%

na.omit() %>%

select(movieId, year, genres) %>%

inner\_join(ratings\_df, by = "movieId") %>%

select(-timestamp, -userId) %>%

mutate(decade = year %/% 10 \* 10) %>%

separate\_rows(genres, sep = "\\|") %>%

group\_by(year, genres) %>%

summarise(count = n(), avg\_rating = mean(rating)) %>%

ungroup() %>%

mutate(wr = weighted\_rating(mean, count, 5000, mean(mean))) %>%

arrange(year)

In [26]:

genres\_rating %>%

filter(genres %in% c("Action", "Romance", "Sci-Fi", "Western")) %>%

ggplot(aes(x = year, y = wr)) +

geom\_line(aes(group=genres, color=genres)) +

geom\_smooth(aes(group=genres, color=genres)) +

facet\_wrap(~genres)

`geom\_smooth()` using method = 'loess' and formula 'y ~ x'

It seems that most of the movie genres are actually getting better and better. That is also influenced by the fact that more movies are produced.

