# 華語／西洋音樂
# 歌詞情緒分析

G11 好rrrrr
黃耀霆
陳怡安
林鼎鈞
劉少桐
徐雋翔
2021.06.17

# 目錄

1. 簡介.....p.3
2. 方法.....p.4
   2.1. 資料取得
   2.2. 原始碼運作說明
3. 結果.....p.12
   3.1. 歌手進榜次數統計
   3.2. 頻率詞統計
   3.3. 歌詞情緒分析
   3.4. 歌詞正向比率分析
   3.5. 華語vs西洋
   3.6. 歌詞分級分析
4. 討論與貢獻.....p.27
   4.1. 結語
   4.2. 未來展望
5. 附錄.....p.28
   5.1. 組員分工
   5.2. 參考資料

# 簡介

## 研究介紹

音樂是大時代的縮影，每個年代的音樂展現其特有的思想、文化或歷史背景。而身為音樂愛好者的我們，決定以音樂為題材，縱向比較音樂在時代之間的變化趨勢和時代價值，橫向比較中西方（華語與西洋）空間上音樂展現的異同。

## 分析歌詞流程

### 收集

歌詞、歌名及歌手等資料，是我們透過網路爬蟲 R語言 httr和 rvest套件、存取 KKbox之 API及 Python lyricsgenius套件爬取 Billboard, Musixmatch, KKbox, Genius等網站取得。

### 清洗

對於收集到的文本資料我們透過 tidyr, tidytext, textclean, stringr, 及 textdata等套件進行清理。

### 分析

接著利用 jiebaR, dplyr, tibble等套件，整理dataframe以進行分析。

### 視覺化

分析結果的呈現，我們使用 R語言ggplot2, wodcloud2, tidytext等套件繪圖。

### 呈現

利用 R 的 blogdown 套件建立專案，接著從Hugo選擇主題模板，把所有的貼文利用 R markdown語法更新到網站上，最後透過Netlify免費架設出此次的專案靜態網頁，最後將所有成過放在靜態網站上呈現。

# 方法

## 資料取得

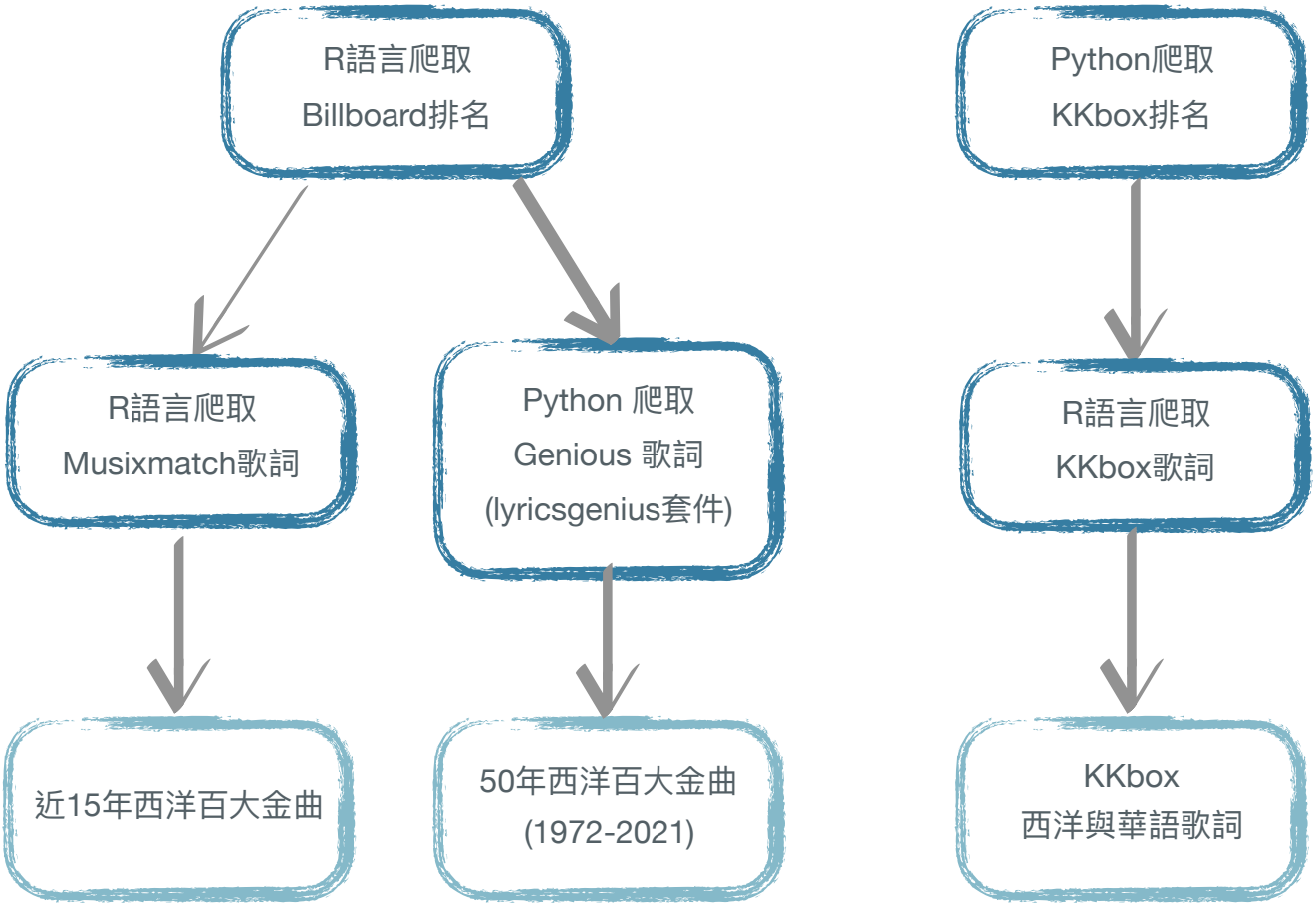
### 資料來源

歌曲排名網站：Billboard, KKbox

歌曲歌詞網站：Genius, Musixmatch, KKbox

### 取得方法

1. R爬取Billboard排名, KKBox歌詞
2. Python 爬取KKbox連結、使用 lyricsgenius 套件爬取英文歌詞



# 方法

## 原始碼

# 爬資料

## 爬取 Billboard排名前百的歌名、歌手

爬取歌名：
使用httr::GET向目標網址發出request，並用content()取得response，
接著透過rvest::html_nodes()抓取歌名所在的標籤，
建立以年份為欄位的 dataframe 存成 df_titles

```
library(httr)
library(rvest)
library(stringr)
```

```
year_websites <- c()
i = 0
for(year in 1972:2021){
    i <- i + 1
    year_websites[i] <- paste0("https://www.billboard.com/charts/hot-100/", year, "-06-05")
}
```

```
#title
df_titles <- data.frame(matrix(NA, ncol=0, nrow = 100))
for(year in 1972:2021 ){
    resp <- GET( paste0("https://www.billboard.com/charts/hot-100/", year, "-06-05") )
    while(resp$status_code != 200){
        resp <- GET( paste0("https://www.billboard.com/charts/hot-100/", year, "-06-05") )
    }
    html <- content(resp)
    songs <- html %>%
        html_nodes("ol.chart-list__elements")
    title <- songs %>%
        html_nodes(".chart-element__information__song") %>%
        html_text() %>%
        trimws()

    df_titles[ , ncol(df_titles) + 1] <- title
    colnames(df_titles)[ncol(df_titles)] <- year
}
```

爬取歌手（同爬取歌名的步驟）：
建立以年份為欄位的 dataframe 存成 df_artist

```
#artist
df_artists <- data.frame(matrix(NA, ncol=0, nrow = 100))
for(year in 1972:2021 ){
    resp <- GET( paste0("https://www.billboard.com/charts/hot-100/", year, "-06-05") )
    while(resp$status_code != 200){
        resp <- GET( paste0("https://www.billboard.com/charts/hot-100/", year, "-06-05") )
    }
    html <- content(resp)
    songs <- html %>%
        html_nodes("ol.chart-list__elements")
    artist <- songs %>%
        html_nodes(".chart-element__information__artist") %>%
        html_text() %>%
        trimws()

    df_artists[ , ncol(df_artists) + 1] <- artist
    colnames(df_artists)[ncol(df_artists)] <- year
}
```

# 方法

## 原始碼運作說明

### 爬取Genius歌詞

將原有的 df_titles 和 df_artists 轉換成
以歌手、歌名為欄位的新 dataframe: df_all

```
#用gather轉換
df_artists %>%
    mutate(Rank = 1:100) %>%
    gather("Year", "Artist", 1:50) -> df_Artist

df_titles %>%
    mutate(Rank = 1:100) %>%
    gather("Year", "Title", 1:50) -> df_Title
```

```
#最終的Dataframe
df_All <- df_Title %>%
    mutate(Artist = df_Artist$Artist)
```

利用前頁爬取的歌名、歌手資料透過Python lyricsgenius 套件爬取歌詞

```
import lyricsgenius
import pandas as pd
import csv
import pandas as pd
from bs4 import BeautifulSoup
import time
import requests
import json
import re
import csv
import string
import random
```

```
token = "TTZReCNV_4kBUIEwI2P350OLnvckm68yKdCGgdOVOJaJa4ABud3BCMQVf_nGoKVE"
genius = lyricsgenius.Genius(token)
chart = pd.read_csv("/Users/spacegray0315/Desktop/R&Data_science/billboard/All.csv")
lyrics = []
song_invalid_no = []
for i in range(800, 1800):
    print(i)
    artist_name = chart["Artist"][i]
    song_title = chart["Title"][i]
    try:
        song = genius.search_song(song_title, artist = artist_name)
        lyric = song.lyrics
    except:
        song = "null"
        lyric = "null"
    print(lyric)
    lyrics.append(lyric)
    time.sleep(random.uniform(1, 5))
```

### 爬取 KKbox 歌手、歌名

使用 Python 首先取得憑證的API網址、標頭及參數，
並透過request.post()取得結果

```
from bs4 import BeautifulSoup
import requests
import json
import re
```

# 方法

## 原始碼
## 運作說明

```
def get_access_token():
  #API網址
  url = "https://account.kkbox.com/oauth2/token"
  #標頭
  headers = {
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "account.kkbox.com"
  }
  #參數
  data = {
    "grant_type": "client_credentials",
    "client_id": "86b4143aa500ae41e800ceef0e5d7437",
    "client_secret": "940af67c66f936703966539f364fe7b6"
  }
  access_token = requests.post(url, headers=headers, data=data)
  return access_token.json()["access_token"]
```

使用 Python 的 request.get 進行API呼叫
並使用 response.jason 將存取網頁內容轉為字典，
接著取得歌手資料（歌名資料亦是如此）

```
def get_charts_tracks(chart_id):
  #存取憑證
  access_token = get_access_token()
  #取得音樂排行榜列表中的歌曲API網址
  url = "https://api.kkbox.com/v1.1/charts/" + chart_id + "/tracks"
  #標頭
  headers = {
    "accept": "application/json",
    "authorization": "Bearer " + access_token
  }
  #參數
  params = {
    "territory": "TW"  #台灣領域
  }
  response = requests.get(url, headers=headers, params=params)
  result = response.json()["data"]
  songs = []
  for item in result:
    artist = item["album"]["artist"]["name"].split(", ")
    if len(artist) > 1:
        name = ""
        for i in range(len(artist)-1):
            name += artist[i] + " "
        name += artist[len(artist)-1]
    else:
        name = artist[0]
    songs.append(([item["name"], name, item["url"]]))
  return songs

chart_id = "X_uXaLNUGc_ahqgH22"
song_list = get_charts_tracks(chart_id)
link_list = []
lyrics = []
```

將取得的歌手與歌名資料做成下一步爬取歌詞之網址（也就是欄位”link”），
接著再使用R語言同前頁透過httr套爬取KKbox網站歌詞資料

```
singer = requests.get('https://www.kkbox.com/tw/tc/charts/artist_song_charts-oiUmCge9WONO2zyWC0F0E08H.html')
soup = BeautifulSoup(singer.text, 'html.parser')
songs = soup.find_all('a', class_="song-title")
name = soup.find('div', class_="song-artist-album").a.string
links = []
names = []
for song in songs[0:20]:
    link = "https://www.kkbox.com" + song["href"]
    links.append(link)
    names.append(name)
print(links)
```

# 方法

## 原始碼運作說明

### 清理、斷詞

將歌詞（歌名亦是如此）清理，其中包括拆解縮寫、轉為小寫後斷詞，
並剔除特殊符號以及stopwords，
最後透過 tidytext::unnest_tokens()將已斷詞資料轉成 tidytext format

```
#合併、清除NULL
all_songs_raw <- bind_rows( song_1970,
                            song_1980,
                            song_1990,
                            song_2000,
                            song_2010,
                            song_2020)
all_songs <- all_songs_raw %>% filter(!Lyrics == "null")
```

```
#cleaning
cleand <- all_songs
#還原縮寫
clean <- function(x){
  x = gsub("won't", "will not", x)
  x = gsub("can't", "can not", x)
  x = gsub("n't", " not", x)
  x = gsub("'ll", " will", x)
  x = gsub("'re", " are", x)
  x = gsub("'ve", " have", x)
  x = gsub("'m", " am", x)
  x = gsub("'d", " would", x)
  x = gsub("'s", "", x)
  x = gsub('feelin|feelingg', 'feeling', x)
  x = gsub('tryin', 'trying', x)
  x = gsub('mothafucka', 'motherfucker', x)
  x = gsub('wanna', 'want to', x)
  x = gsub('dat', 'that', x)
  return(x)
}
#清除歌詞資料裡面[Verse]、[Bridge]之類的非歌詞元素
removeSpecialChars <- function(x) gsub("[^a-zA-Z0-9 ]", " ", x)

cleand$Lyrics = all_songs$Lyrics %>%
    gsub(pattern = '\\[[^][]*]', replacement = ' ') %>%
    tolower() %>%
    clean() %>%
    removeSpecialChars() %>%
    gsub(pattern = '[[:punct:]|[:digit:]]', replacement = ' ')
```

```
#tidying-lyrics
#making stopwords list
myStopwords = c('ooh', 'oooh', 'oh', 'ow', 'uh', 'baby', 'babi', 'bebe',
                'yeah', 'yeh', 'ye', 'yes', 'ya', 'eh', 'da', 'cardi', 'se',
                'ayy', 'ah', 'yo', 'o', 'bum', 'na', 'la', 'ai', 'ba', 'hey','chorus',
                'da','yo','dr','aah','mckenzie','2006','yuhh','yurr','aaaaaaaaaaaaahhhhhhh',
                'aaahhs','aaaaahh', 'aaaaaaaaaaaaaah', 'uhuh',
                'gon','gonna','woo','badoomboom','badoom','boom','bass','hmm', 'mmm', 'mmhmm')
df_tidy <- cleand %>%
  ungroup() %>%
  unnest_tokens(word, Lyrics) %>%
  distinct() %>%
  filter(!word %in% myStopwords & decade != 'NA') %>%
  anti_join(stop_words) %>%
  filter(nchar(word) > 2) %>%
  select(year, Artist, Title, word, decade)
```

# 方法

## 原始碼
## 運作說明

# 分析、繪圖

## 頻率詞統計

將清洗、斷詞過的歌詞計算詞彙頻率，並且用 group_by() 以十年為一個區間，
接著slice()出每個十年前7個頻率詞，
使用 ggplot2 套件的 geom_col() 繪製長條圖呈現

```
#timeless-lyrics
timeless_words <- df_tidy %>%
  group_by(decade) %>%
  count(word, decade, sort = TRUE) %>%
  slice(seq_len(7)) %>%
  ungroup() %>%
  arrange(decade, n) %>%
  mutate(row = row_number())

plt_timeless <- timeless_words %>%
  ggplot(aes(row, n, fill = decade)) +
  geom_col() +
  labs(title = "Timeless words",
       x = NULL,
       y = NULL) +
  theme_bw() +
  theme(legend.position = "None") +
  facet_wrap(~decade, scales = "free", ncol = 5) +
  scale_x_continuous(breaks = timeless_words$row,
                     labels = timeless_words$word) +
  theme(axis.text.x = element_blank()) +
  coord_flip()
```

另外我們也使用同上方法分析近15年的百大金曲，
取出前20個頻率詞

```
#create word frequency charts
frequency_words <- freq_terms(
    song_data$text, top = 20, at.least = 3, stopwords =
        c(stopwords("english"), "dont", "can", "get", "got"))

frequency_words %>%
    ggplot( aes(x=WORD, y=FREQ)) +
    geom_segment( aes(x= reorder(WORD,desc(FREQ)), xend=WORD, y=0, yend=FREQ)
                  ,color="skyblue", size=1) +
    geom_point(color="blue", size=4, alpha=0.6) +
    theme_minimal()+
    theme(text = element_text(size = 10),
          plot.title = element_text(size = 16,color = "#ff5a5f", face = "bold"
                                    ,margin = margin(b = 7)),
          plot.subtitle = element_text(size = 10, color = "darkslategrey"
                                       , margin = margin(b = 7))) +
    labs(x = "歌詞用字", y = "頻率", title = "2006年-2020年前百大金曲常用詞統計")
```

接著統計50年每年前百大歌曲的前100個最常使用詞彙，
使用 wordcloud2套件以文字雲方式呈現

```
#算詞頻
unigram_tidy <- df_tidy %>%
  group_by(word) %>%
  count() %>%
  ungroup () %>%
  arrange(desc(n))

plt_unigram <- wordcloud2(data = unigram_tidy[1:100, ], size = 1, color = brewer.pal(7, 'Dark2'))
```

# 方法

## 原始碼
## 運作說明

針對詞頻，最後我們依據TF-IDF演算法使用 bind_if_idf()
找出在那個年代歌詞中出現平率較高但在其他則沒那麼高的詞，
透過geom_col()展現不同時代的前8個unique words

```
#TF-IDF
tfidf_words_decade <- df_tidy %>%
  count(decade, word, sort = TRUE) %>%
  ungroup() %>%
  bind_tf_idf(word, decade, n) %>%
  arrange(desc(tf_idf))

top_tfidf_words_decade <- tfidf_words_decade %>%
  group_by(decade) %>%
  slice(seq_len(8)) %>%
  ungroup() %>%
  arrange(decade, tf_idf) %>%
  mutate(row = row_number())

plt_tfidf <- top_tfidf_words_decade %>%
  ggplot(aes(x = row, tf_idf, fill = decade)) +
  geom_col(show.legend = NULL) +
  labs(x = NULL, y = "TF-IDF") +
  ggtitle("Important Words using TF-IDF by Decade") +
  theme_bw() +
  facet_wrap(~decade,
             ncol = 2, nrow = 3,
             scales = "free") +
  scale_x_continuous(
    breaks = top_tfidf_words_decade$row,
    labels = top_tfidf_words_decade$word) +
  coord_flip()
```

### 百大傳唱歌曲頻率詞統計

我們將百大傳唱歌曲歌詞資料放入自行定義的 group函數，
並計算每首個的分級平均，
透過 geom_bar()呈現分級分佈

```
#用來分類的函數
group <-function(x){
  if(x%in%level1)return(1)
  else if(x%in%level2)return(2)
  else if(x%in%level3)return(3)
  else if(x%in%level4)return(4)
  else if(x%in%level5)return(5)
  else if(x%in%level6)return(6)
  else return(7)
}
```

```
final_data <-data%>%
  mutate(level=mean_grouping2)
final_data1 <-final_data%>%
  group_by(level)%>%
  summarise(n=n())%>%
  arrange(desc(n))
final_data1$level <-unlist(final_data1$level )
ggplot(final_data1)+
  geom_bar(aes(x = level, y = n),
           stat = "identity")+
  labs(title="歌名詞彙使用等級")
```

# 方法

## 原始碼
## 運作說明

針對詞頻，最後我們依據TF-IDF演算法使用 bind_if_idf()
找出在那個年代歌詞中出現平率較高但在其他則沒那麼高的詞，
透過geom_col()展現不同時代的前8個unique words

```
#TF-IDF
tfidf_words_decade <- df_tidy %>%
  count(decade, word, sort = TRUE) %>%
  ungroup() %>%
  bind_tf_idf(word, decade, n) %>%
  arrange(desc(tf_idf))

top_tfidf_words_decade <- tfidf_words_decade %>%
  group_by(decade) %>%
  slice(seq_len(8)) %>%
  ungroup() %>%
  arrange(decade, tf_idf) %>%
  mutate(row = row_number())

plt_tfidf <- top_tfidf_words_decade %>%
  ggplot(aes(x = row, tf_idf, fill = decade)) +
  geom_col(show.legend = NULL) +
  labs(x = NULL, y = "TF-IDF") +
  ggtitle("Important Words using TF-IDF by Decade") +
  theme_bw() +
  facet_wrap(~decade,
             ncol = 2, nrow = 3,
             scales = "free") +
  scale_x_continuous(
    breaks = top_tfidf_words_decade$row,
    labels = top_tfidf_words_decade$word) +
  coord_flip()
```

### 百大傳唱歌曲頻率詞統計

我們將百大傳唱歌曲歌詞資料放入自行定義的 group函數，
並計算每首個的分級平均，
透過 geom_bar()呈現分級分佈

```
#用來分類的函數
group <-function(x){
  if(x%in%level1)return(1)
  else if(x%in%level2)return(2)
  else if(x%in%level3)return(3)
  else if(x%in%level4)return(4)
  else if(x%in%level5)return(5)
  else if(x%in%level6)return(6)
  else return(7)
}
```

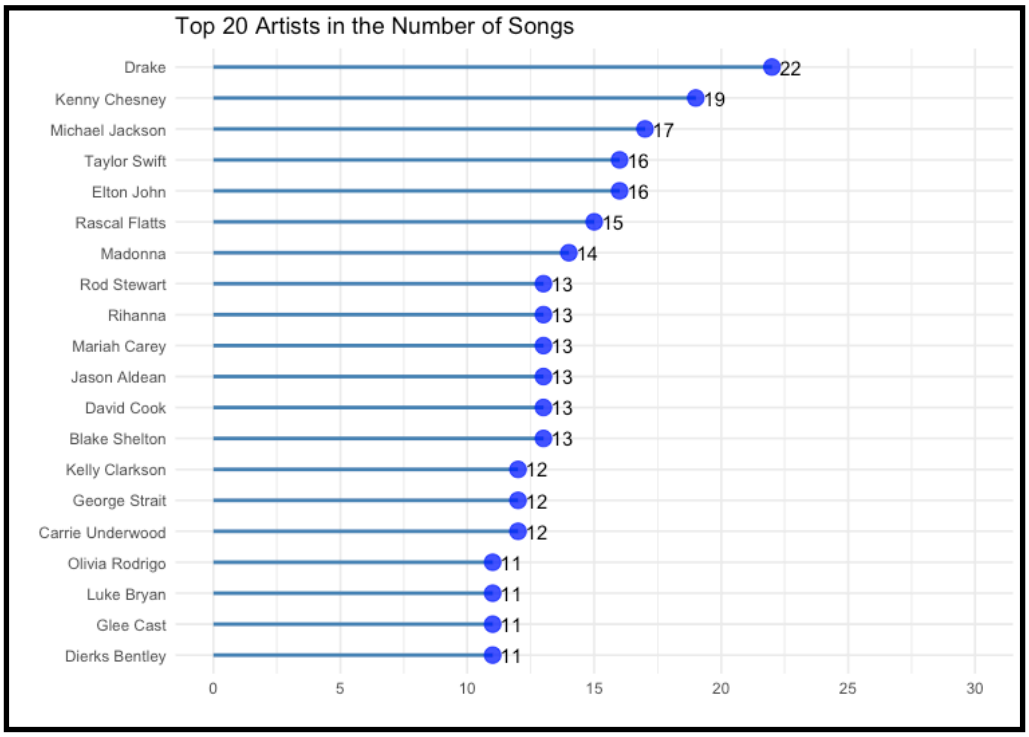
```
final_data <-data%>%
  mutate(level=mean_grouping2)
final_data1 <-final_data%>%
  group_by(level)%>%
  summarise(n=n())%>%
  arrange(desc(n))
final_data1$level <-unlist(final_data1$level )
ggplot(final_data1)+
  geom_bar(aes(x = level, y = n),
           stat = "identity")+
  labs(title="歌名詞彙使用等級")
```

# 結果

## 歌手進榜次數統計

### 前20位進榜 Billboard Hot 100歌手

（50年--趨勢）



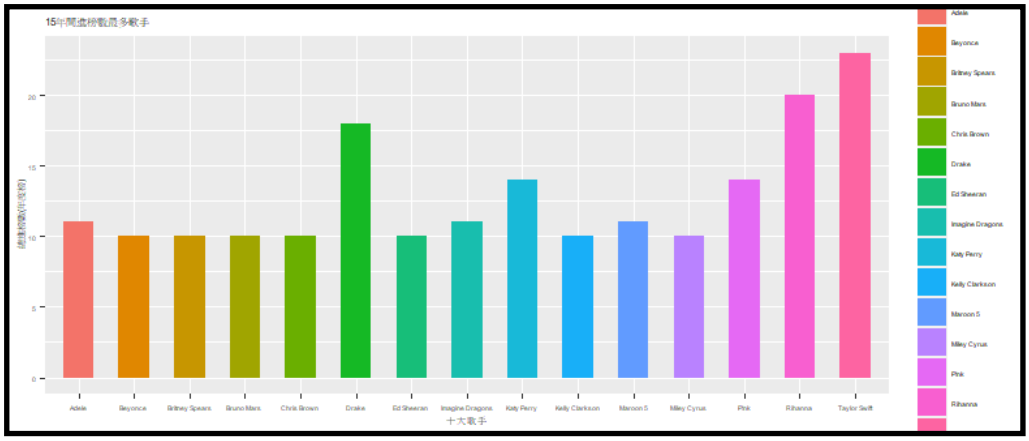

### 文字雲展現

# 結果

## 歌手進榜次數統計

### 前20位進榜 Billboard Hot 100歌手

**（15年--近期統計）**

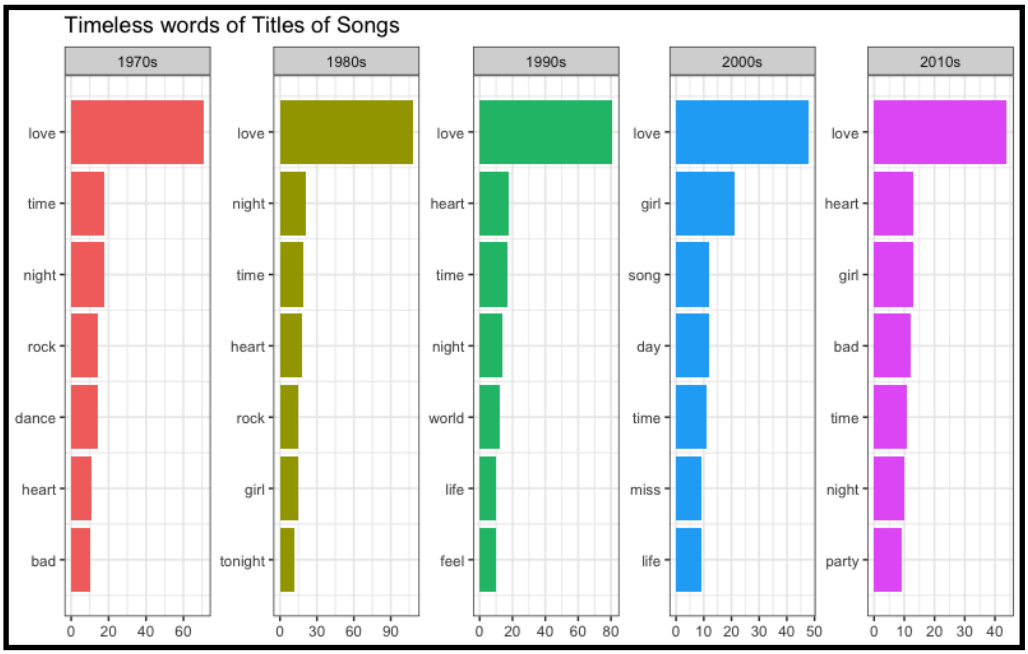


# 結果

## 頻率詞統計
## —歌名

### 歌名詞頻統計

（50年--趨勢）




### 文字雲展現

# 結果

## 頻率詞統計—歌名

### 分析：

透過分析近５０年的英文歌歌名，我們發現到「ｌｏｖｅ」這個詞彙始終處在第一名，且與第二名皆有斷層性的差距，推論可能是因為「愛」這個元素是普遍歌曲會採納的。其他詞彙諸如

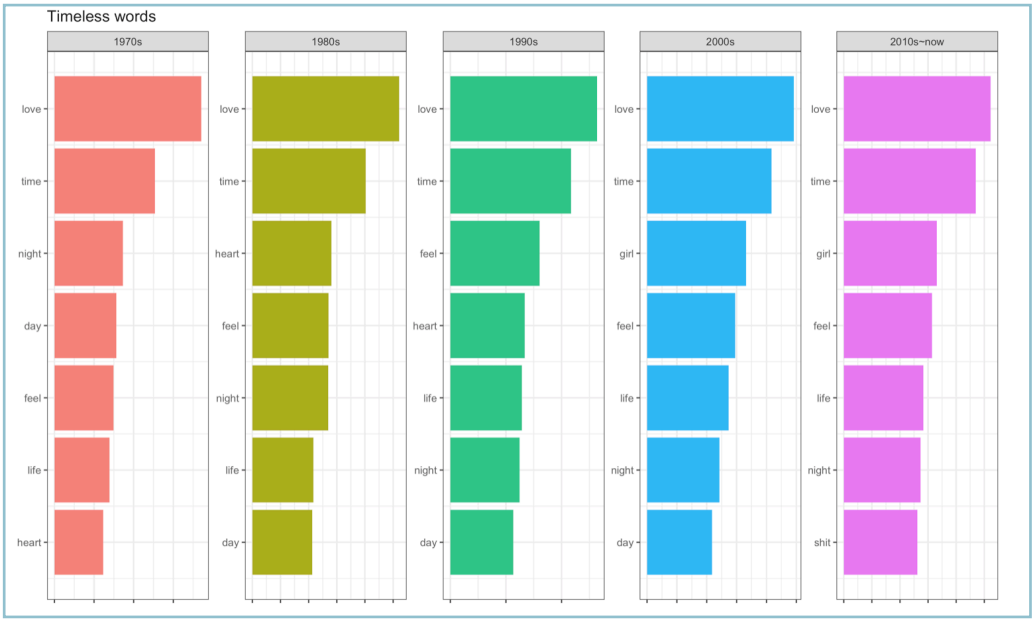
「ｈｅａｒｔ」「ｎｉｇｈｔ」「ｔｉｍｅ」等等也都與這樣的推論有一定程度的關聯性。

# 結果

## 頻率詞統計
## —歌詞

### 歌詞詞頻統計

（50年--趨勢）



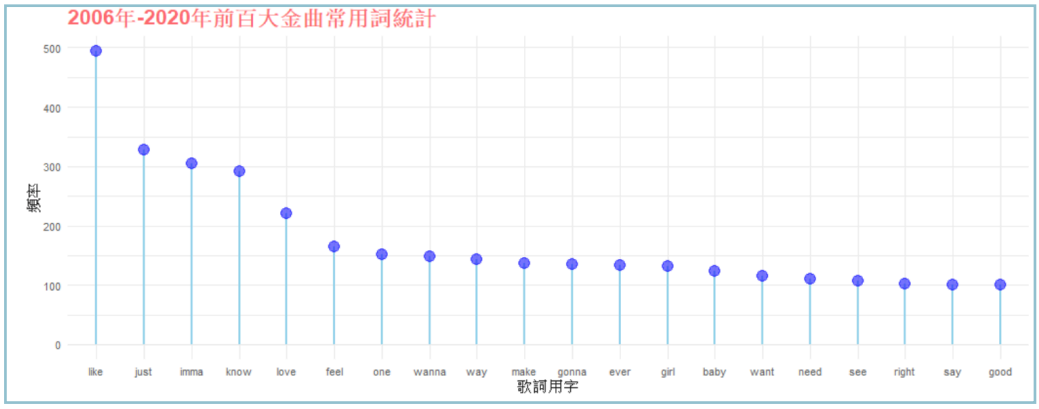

### 文字雲展現

# 結果

## 頻率詞統計
## —歌詞

### 歌詞詞頻統計

**（15年--近期統計）**




## 分析：

接著我們將近５０年以及近１５年的歌詞做為分析對象，分別代表趨勢性以及近代性的歌詞發展，顯示出「ｌｏｖｅ」（或近義詞

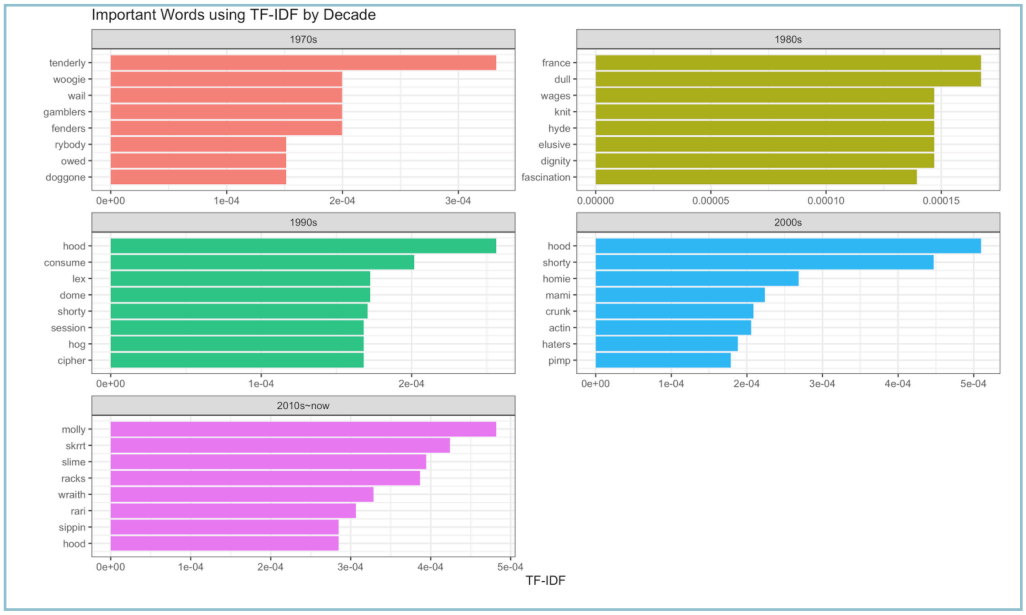
「ｌｉｋｅ」）依然是第一名。也發現出其實歌詞與歌名常出現的單詞都有一定程度的重疊性。

# 結果

## 頻率詞統計—歌詞

## 不同年代下的unique words(by TF-IDF)

**（50年--趨勢）**




## 分析：

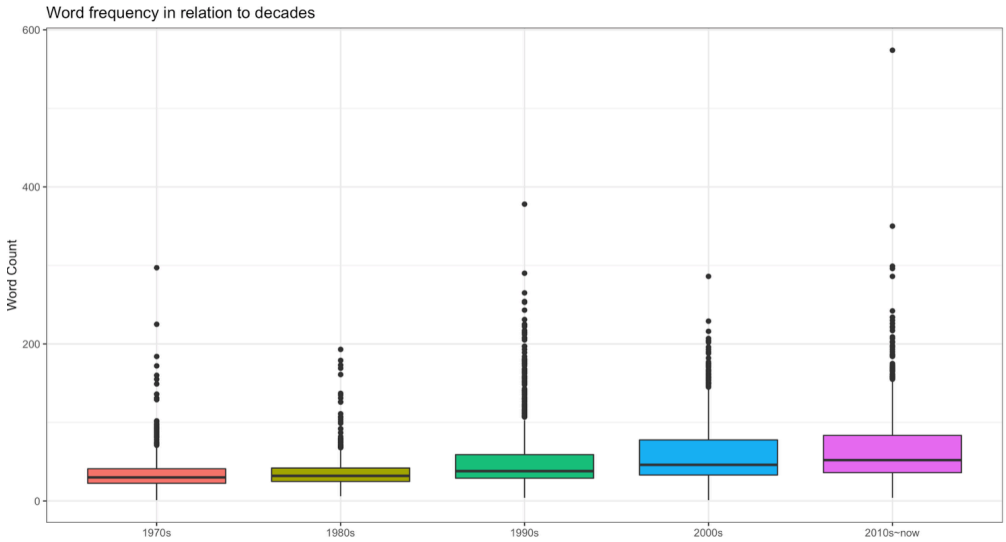
利用演算法ＴＦ－ＩＤＦ，我們發現出不同時代下的unique words，這對於跨時代性的比較有相當大的意義，其充分體現出在當代文化價值下每個時代歌詞所表現出的意涵。而表現出的結果也如預期一樣，每個時代的unique words幾乎不怎麼相似。

# 結果

## 頻率詞統計—歌詞

### 歌詞長度

**（50年--趨勢）**



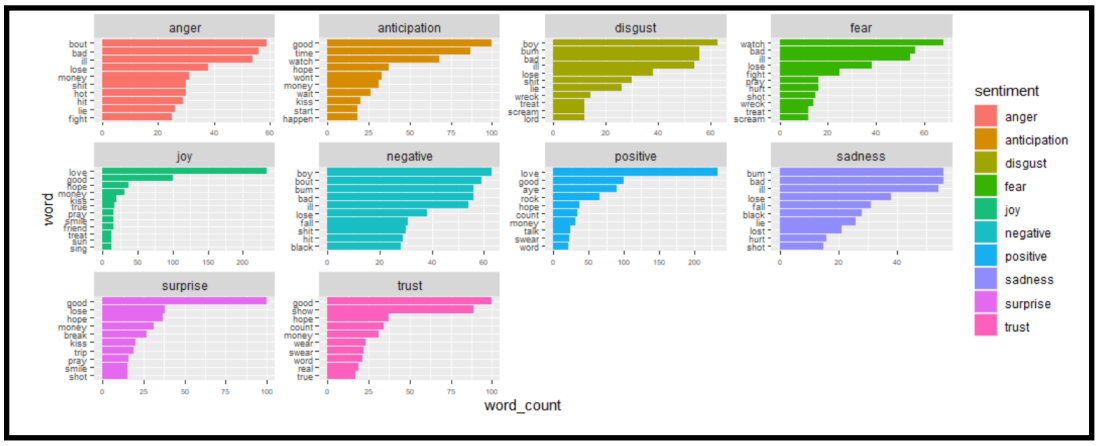

### 分析：

觀察近50年的歌詞長度，發現歌詞有逐步增長的趨勢→推論可能是近代饒舌歌曲的增加提升了整體平均。

# 結果

## 歌詞
## 情緒分析

### 百大金曲十種情緒分析
（15年）




## 分析：

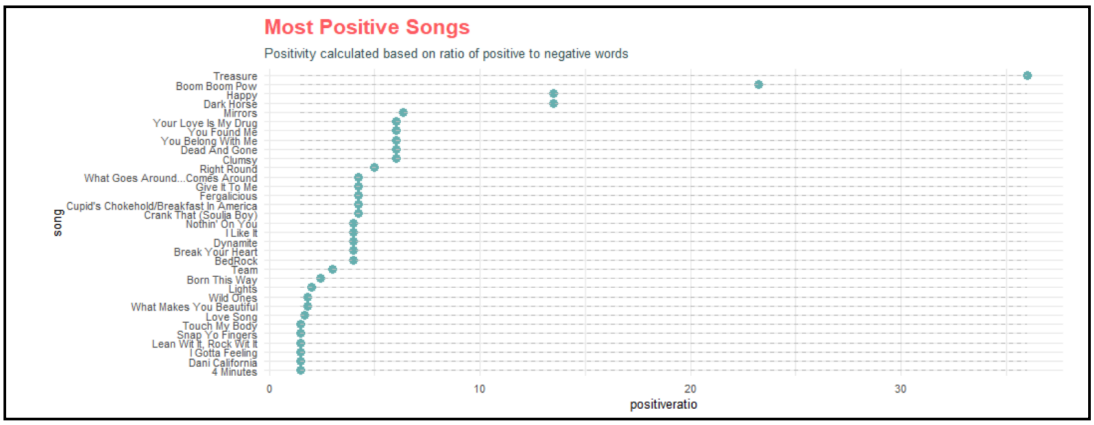
分析在不同的情緒下，每個情緒下常出現的歌詞→在相似的情緒下，出現的字詞相似性較高，ex: anticipation/joy/positive: hope, love, good等詞皆有出現。而anger/disgust/sadness: bad, ill皆有出現。

# 結果

## 歌曲情緒分析

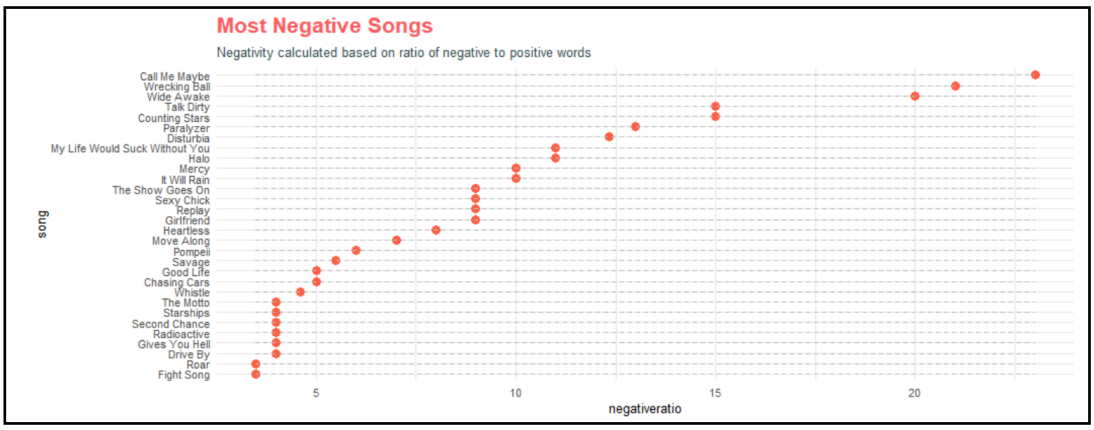
### 最正面歌曲

（15年）




### 最負面歌曲

（15年）



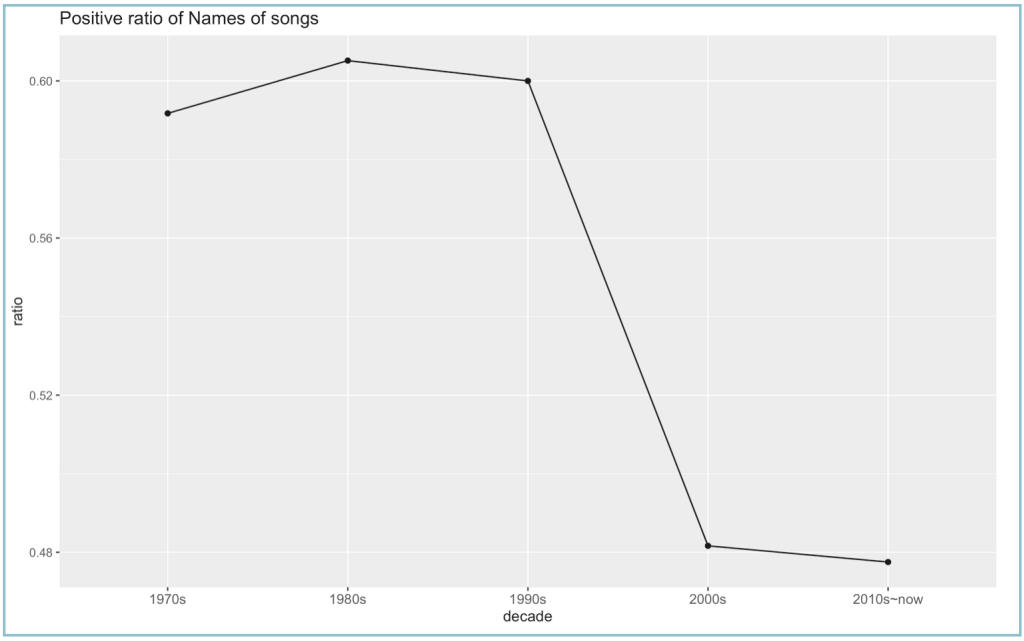

### 分析

在進行這個分析時，我們認為負面歌曲的結果有點出乎意料：call me maybe竟然排名負面歌曲第一名→進而認為應是我們忽略了同樣的歌詞重複唱可能會影響這首歌的正負向比率。

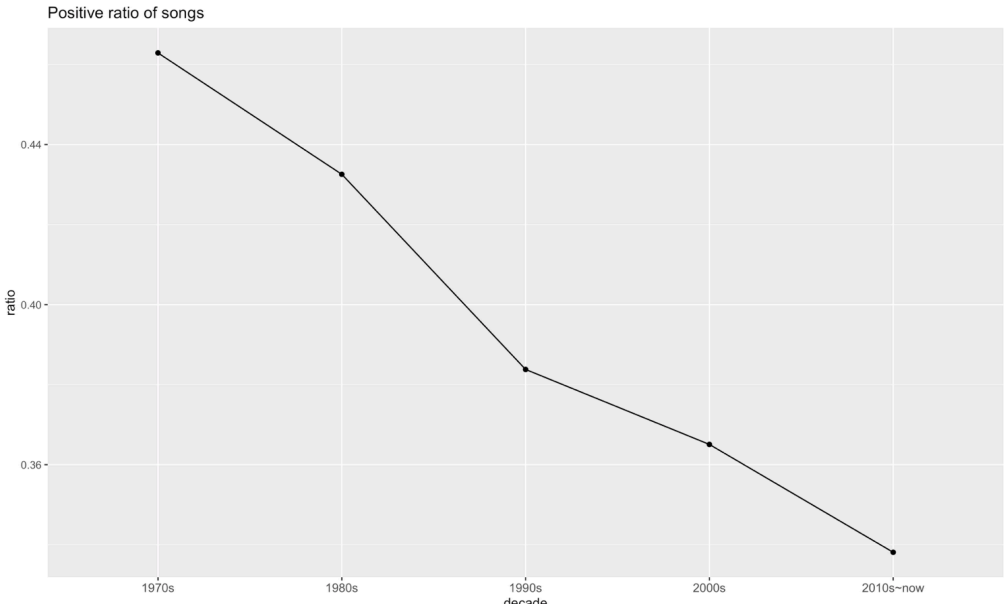
# 結果

歌曲
正向比率分析

## 分時代歌名正向比率



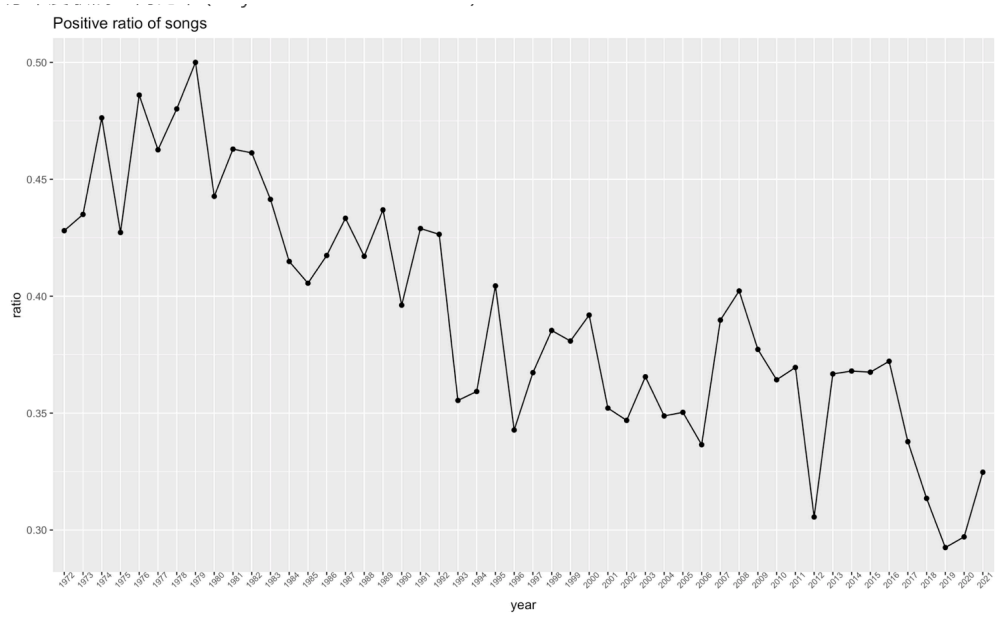

## 分時代歌詞正向比率



# 分年度歌詞正向比率：

# 結果

## 歌曲
## 正向比率分析



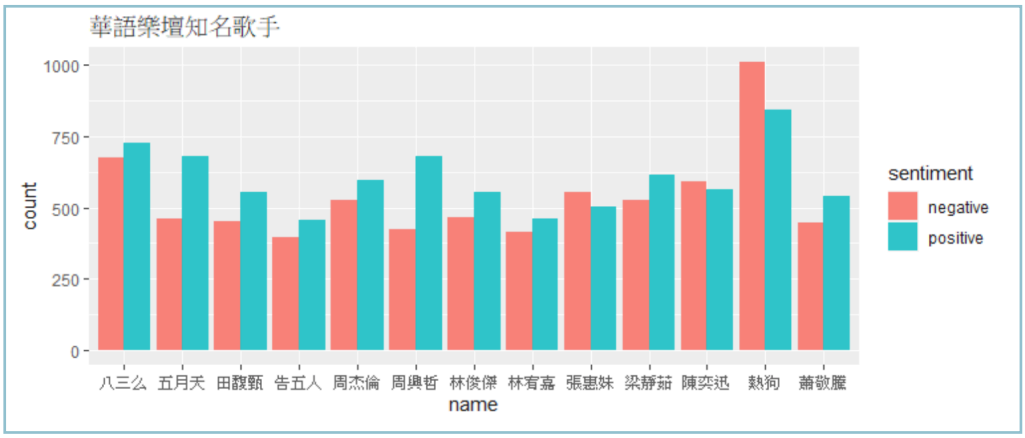

## 分析

藉由上面的圖表，我們發現隨著時間的演進，歌詞越來越不正向，似乎有「厭世化」的趨勢。

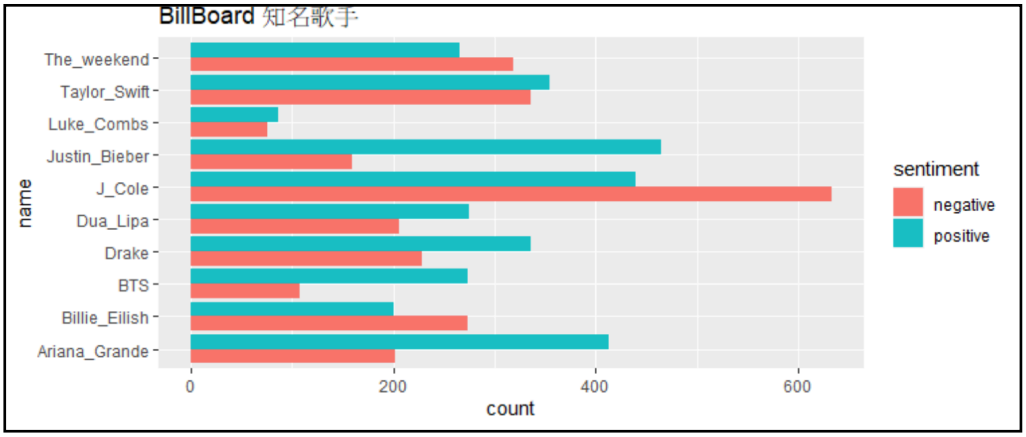
# 結果

## 華語VS西洋
## 一正負向分析

### 華語歌手正負頻率
（近4年）




### 西洋歌手正負頻率
（近4年）

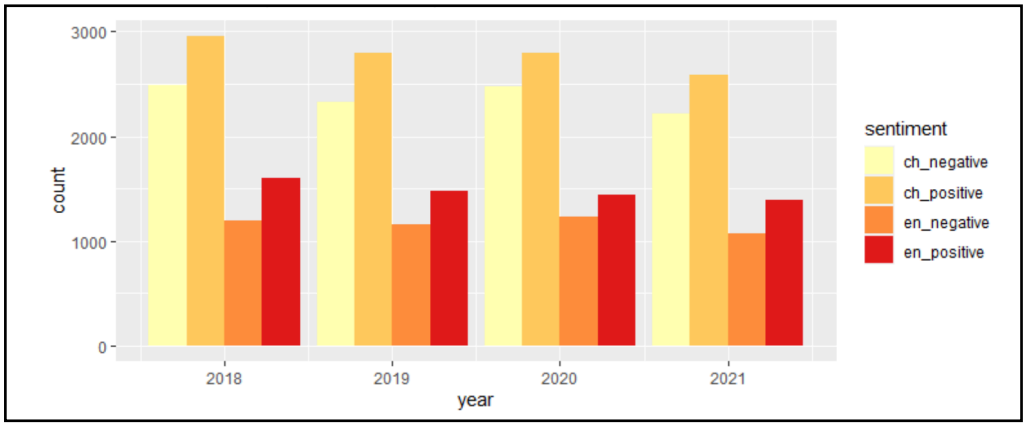


# 結果

## 華語vs西洋
## 一正負向分析

## 華語/西洋歌詞正負頻率

（近4年）




## 分析：

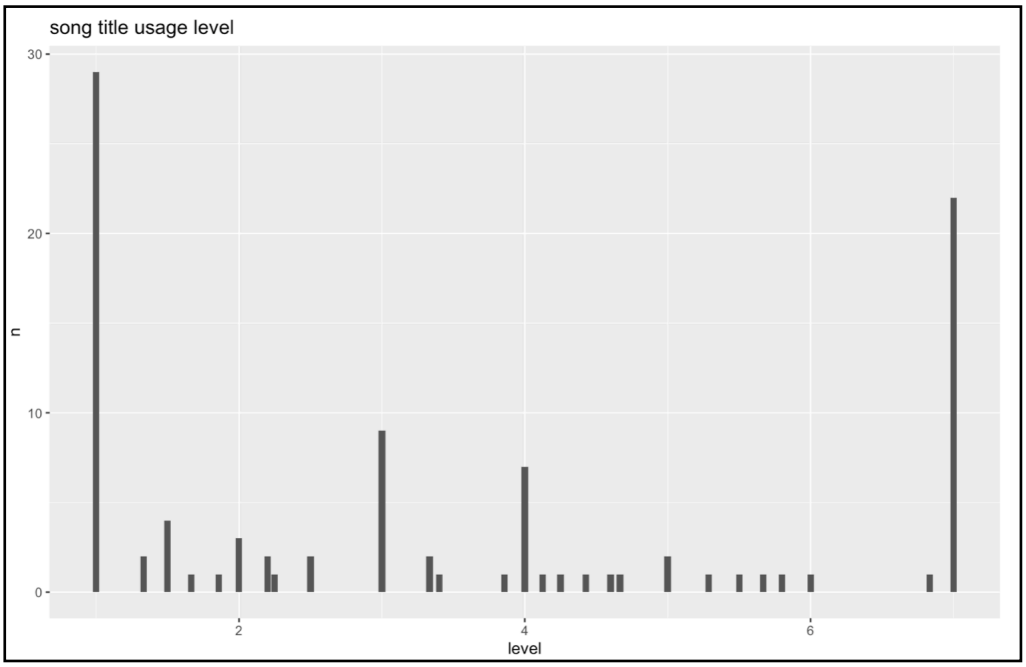
華語歌手中，普遍呈現正向>負向，唯一的例外是熱狗，推論應該是饒舌歌手的作品中較多會diss他人的元素使得歌曲較為負面。

在華語西洋歌詞正負頻率中，同樣呈現出正向歌曲比率較高的現象。

# 結果

## 百大傳唱歌曲歌名分級

## 2020most hummed songs歌名單字分類




## 分析：

分類方法藉由分析billboard 2020前100首被哼唱歌曲的歌名，搭配教育部公布的7000單字分級，（有的詞彙不在這些分類裡面將其歸類為7級）根據每個單詞的分級再取平均數，即為該首歌的分類。

結果發現：一級的比例是最高的，但七級比例如此高的原因推測是使用方法不夠嚴謹。

# 討論與貢獻

## 結語 & 未來展望

透過這次專案，我們分析歌了跨時代以及跨區域的歌曲，結合所學的知識，在不斷地的嘗試之後，也更加理解音樂的箇中奧妙，從多個面向切入，也得到了許多有趣的結果，希望未來我們能將程式語言更融入我們的生活，探討更多不同領域的事物。

另外關於歌詞分析的部分，期許有一天能做的如連結所呈現的，擴大更多可以分析的面向。

（https://www.kaggle.com/deepshah16/song-lyrics-dataset）

# 附錄

## 組員分工

### 黃耀霆

投影片製作、紀錄討論、網路爬蟲、歌手進榜數統計、頻率詞統計、歌詞情緒分析

### 陳怡安

書面報告製作、網路爬蟲、歌手進榜數統計、頻率詞統計、歌曲正負向年度趨勢

### 林鼎鈞

書面報告製作、網路爬蟲、頻率詞統計、歌詞分級分析

### 劉少桐

網站架設、主題發想、網路爬蟲、歌詞彙整（50個年度）

### 徐雋翔

網站架設、網路爬蟲、華語／西洋樂壇正負相比較、歌手正負向用詞統計

# 附錄

## 參考資料

### Rpubs

https://rpubs.com/Nush12/textmining

https://rpubs.com/rafrys/723764

### Github

https://github.com/johnwmillr/LyricsGenius

https://github.com/sweslo17/chinese_sentiment/tree/master/dict

### 其他網站

https://www.learncodewithmike.com/2020/02/python-kkbox-open-api.html

https://suyenting.netlify.app/blog/create-my-website/?fbclid=IwAR1tbeKNrFDHS-X-HINBkbRAYzTvxVcAZ1y9tpfEXf-0D8y-5QH5revlNi4

https://www.managertoday.com.tw/eightylife/article/view/166