Word count: 1975

DATA ANALYTICS 2

Table of Contents

1. The problem

1.1. Background

BottleRock Napa is a music festival located in California's Napa Valley.

It is characterised for its wine experience as well as top-tier pop and rock artists such as Bruno Mars or Red Hot Chilli Peppers. (1)

The festival takes place every year in May since 2013, when it had 120,000 attendees. The 2021 festival is scheduled in September, due to Coronavirus regulations. (2)

1.2. The problem

BottleRock Napa has 39K followers on Twitter, positioning itself in the top 10 festivals with the most followers in 2019:

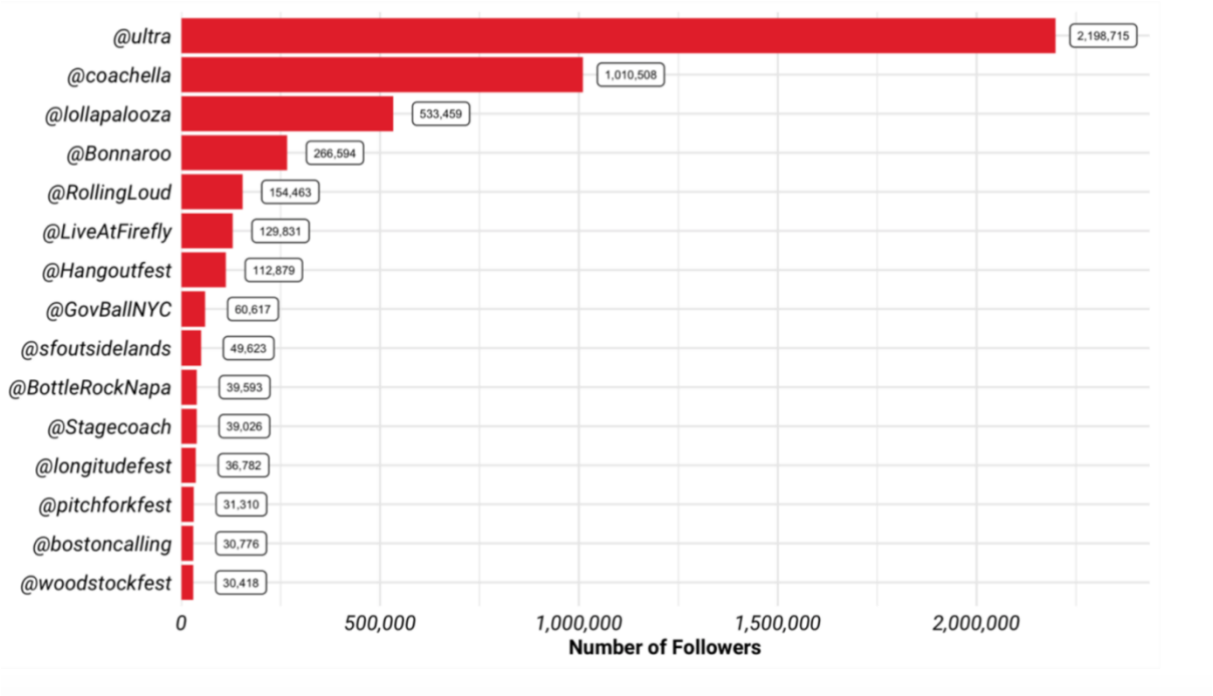


Figure 1 (3)

BottleRock Napa uses Twitter to sell tickets. See an example Tweet bellow:

Figure 2 (3)

The problem this report is addressing is increasing the attendance at the festival by increasing the engagement of people on Twitter and the reach of the Tweets.

The main questions the report is addressing are:

What are the most common words found in the Tweets mentioning BottleRock Napa?

What are the attributes that are the most important to maximise a tweet's reach and how can the festival use them in its advantage?

1.3 The data

The data analysed in this report was collected from the Twitter API, using Postman. It represents the Tweets where the festival username (@BottleRockNapa) was mentioned, starting from July 2020 until April 2021. This is the earliest date Twitter granted permission to retrieve Tweets. The maximum tweets for one request are 100 and the data in this report was retrieved separately for each month starting July 2020, so the data contains maximum 100 tweets per month. See the number of Tweets collected per month in the figure below:

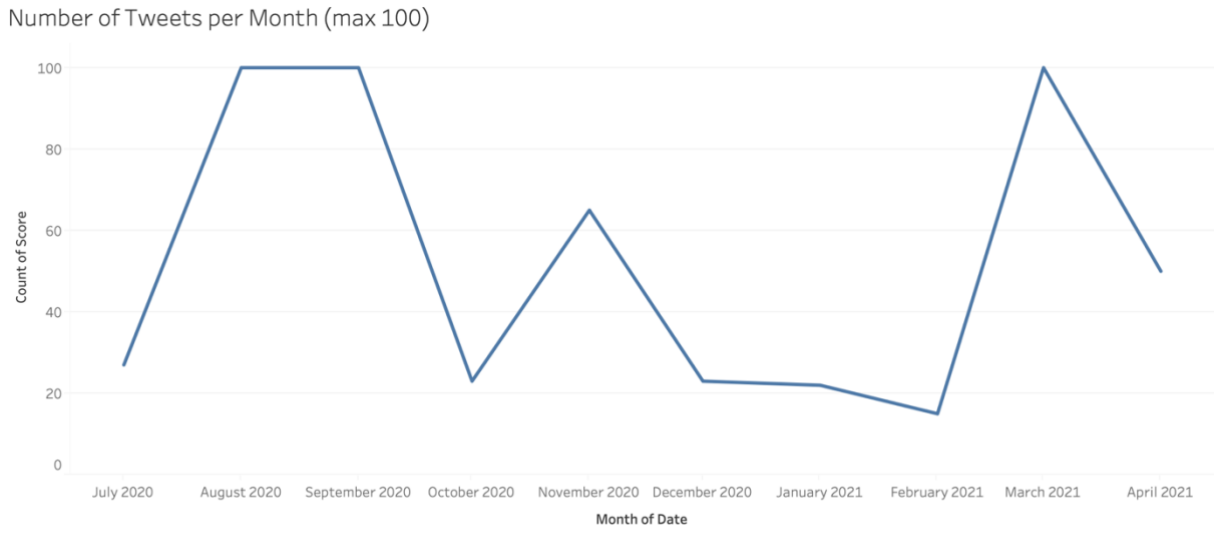


Figure 3 (Tableau)

The fields collected for each tweet can be seen in the example Tweet bellow, collected from Postman in json format:

```
{
    "public_metrics": {
        "retweet_count": 0,
        "reply_count": 0,
        "like_count": 0,
        "quote_count": 0
    },
    "created_at": "2020-08-27T23:03:23.000Z",
    "entities": {
        "urls": [
            {
                "start": 212,
                "end": 235,
                "url": "https://t.co/Z7cpnPrOZl",
                "expanded_url": "https://ul.ink/SXH2-8T9C5Y_T",
                "display_url": "ul.ink/SXH2-8T9C5Y_T"
            }
        ],
        "hashtags": [
            {
                "start": 6,
                "end": 16,
                "tag": "ButterDay"
            },
            {
                "start": 59,
                "end": 76,
                "tag": "ButterChardonnay"
            },
            {
                "start": 96,
```


Figure 4 – Tweet example (1)


```
                "end": 101,
                "tag": "Yeti"
            },
            {
                "start": 206,
                "end": 210,
                "tag": "WIN"
            }
        ],
        "mentions": [
            {
                "start": 158,
                "end": 173,
                "username": "BottleRockNapa"
            },
            {
                "start": 181,
                "end": 195,
                "username": "mattnathanson"
            }
        ]
    },
    "id": "1299120354980515840",
    "text": "Win a #ButterDay! For you and 3 besties. A prize pack with #ButterChardonnay, Govinos, a Butter #Yeti
        cooler and more swag! Plus a virtual JaMSession to meet @BottleRockNapa artist @MattNathanson. ENTER TO #WIN!
        https://t.co/Z7cpnPrOZl",
    "lang": "en",
    "author_id": "248270245",
    "source": "Twitter Web App",
    "reply_settings": "everyone"
}
```


Figure 4 – Tweet example (2)

The report will derive a new target attribute, “Score”, to measure a tweet’s reach, summing up the tweet’s number of likes, replies, retweets, and quotes (retweets containing text).

2. Understand the data

2.1. Nature and size of the data

After adding all the derived attributes (see section 3) the dataset has 22 attributes and 525 rows. The attributes are numerical, text, or date-format.

2.2. Relevant data attributes

The most relevant data attributes will be discussed using 3 Python visuals, 1 R visual, and 6 Tableau visuals. See section 3 for information about how these attributes were derived.

2.2.1. Text data

Figure 5 and 6 represent the text column in a word cloud before and after being pre-processed. It is clearer what the most used words are after removing URLs and the festival name. Some of the most used words are 'enter' or 'win'.

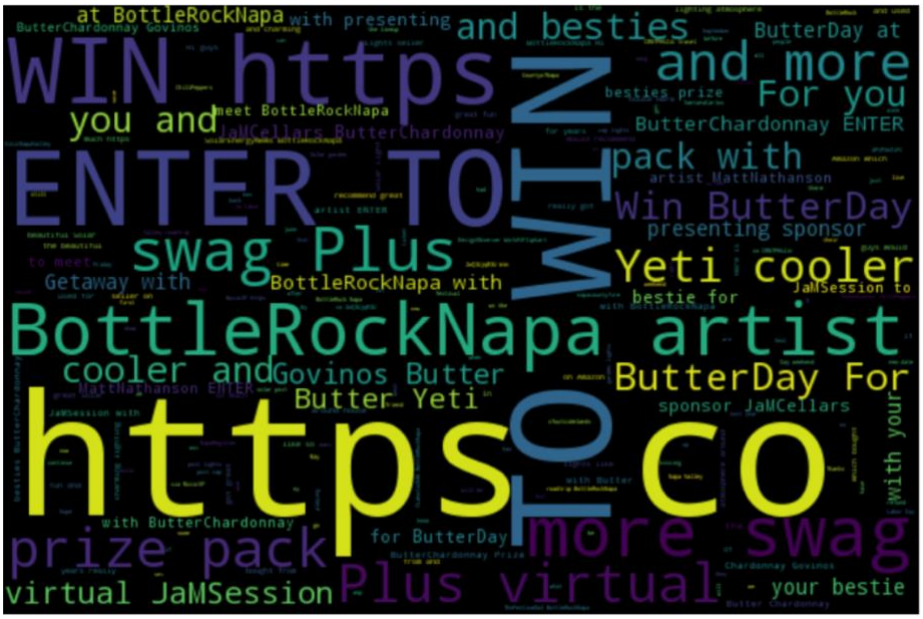


Figure 5: Python visual

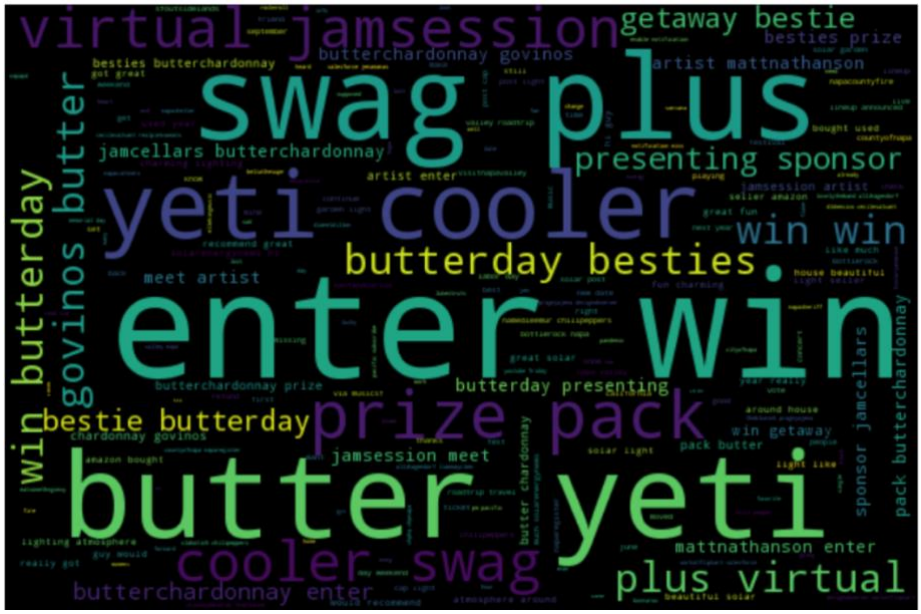


Figure 6: Python visual

The word cloud in *Figure 6* was better represented below – the number of occurrences for each of the most used words are visible. The words 'win', 'enter', 'pack', 'prize' appear in so many tweets because the festival is using Twitter competitions and promotions to sell tickets.

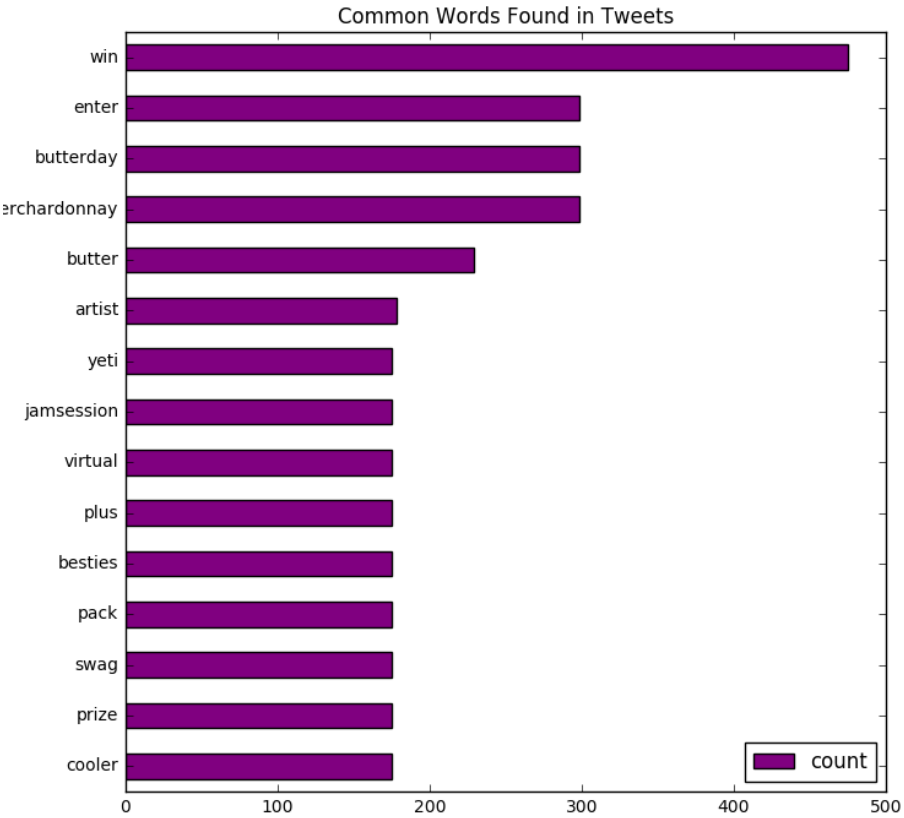


Figure 8: Python visual

2.2.2. Month

The data contains tweets from the last 10 months. The 'month' attribute is numerical. Section 3 explains how this attribute was derived from the full-date.

In the Figure below, the x-axis represents the month. The orange line is the average tweet score for each month. The best month to post seems to be July, with an average score of 5.296.

The blue line is the moving average of tweets score from the previous 3 months. There seems to be a decreasing trend – it might be due to the fact that organisers are forced to postpone the event due to COVID 19.

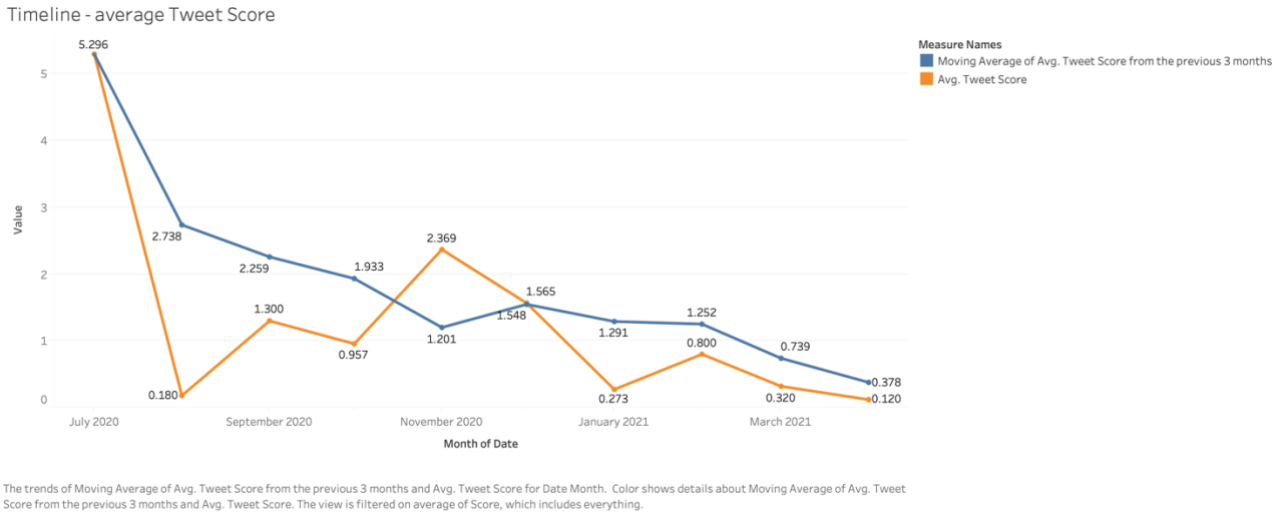


Figure 9: Tableau visual

2.2.3. Hour

Figure 10 represents the average tweets score by hour, in UTC time. The 'day' attribute is numerical. Section 3 explains how this attribute was derived from the full-date.

The best times to post are by far 15 and 14 UTC with an average Tweet Score of 5.5 and 5.

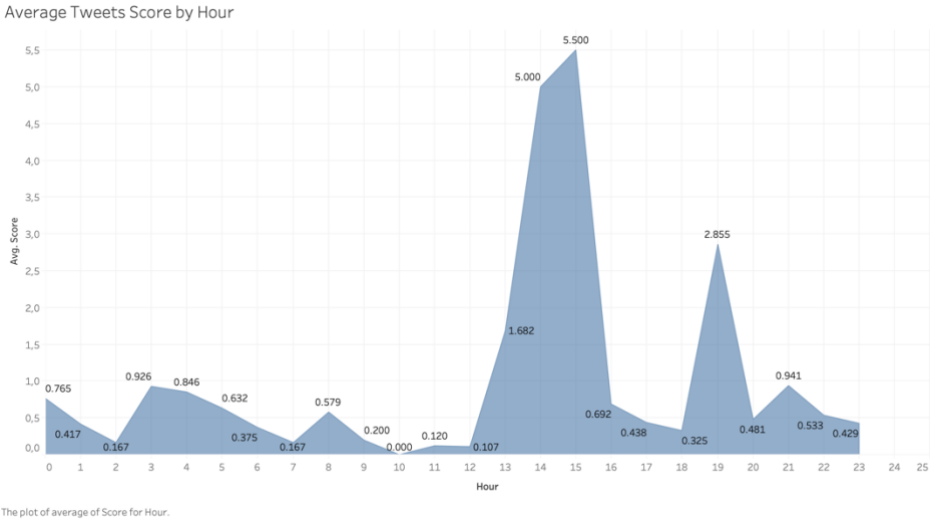


Figure 10: Tableau visual

2.2.4. Happiness score

The happiness score of a tweet is numerical and represents the mean of the happiness score of the words in the tweet. Each word's score is given in the Hedonometer.

The relation between the happiness score and the tweets' score is represented in the histogram below. The tweets with happiness score between 5-6 have the biggest average tweet score -2.978 – followed by the tweets with a happiness score between 7-8.

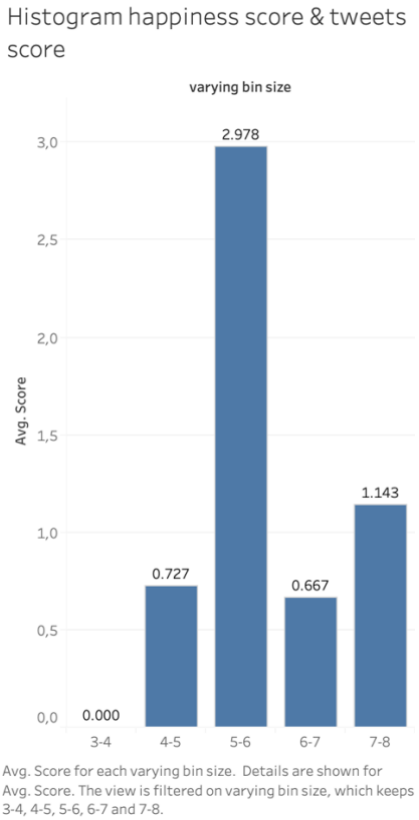


Figure 11: Tableau visual

2.2.5. Handle and Hashtag count

The handle count variable represents the number of user mentions '@' in each tweet. The first box plot shows how this variable is distributed – the average is 2 user mentions.

The hashtag count variable represents the number of hashtags in each tweet. The second box plot shows how this variable is distributed. The average is 3 hashtags, the 25th percentile is 0 hashtags and the 75th percentile is 4 hashtags.

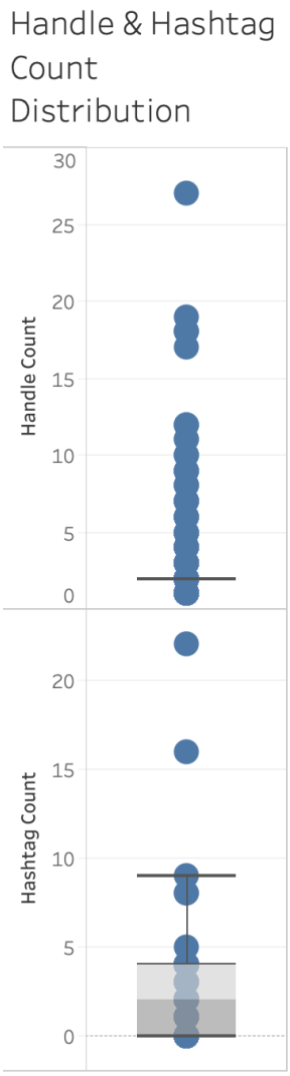


Figure 12: Tableau visual

Figure 13 and *Figure 14* show the relationship between the number of hashtags/user mentions and a tweet's score. Starting from 6, the bins are intervals of five, because it is visible from the box plots that there isn't so much data from 6 and above.

The tweets with 1 and 3 hashtags seem to have the best impact. The tweets with 6-10 mentions have the best impact with an average tweet score of 6.5.

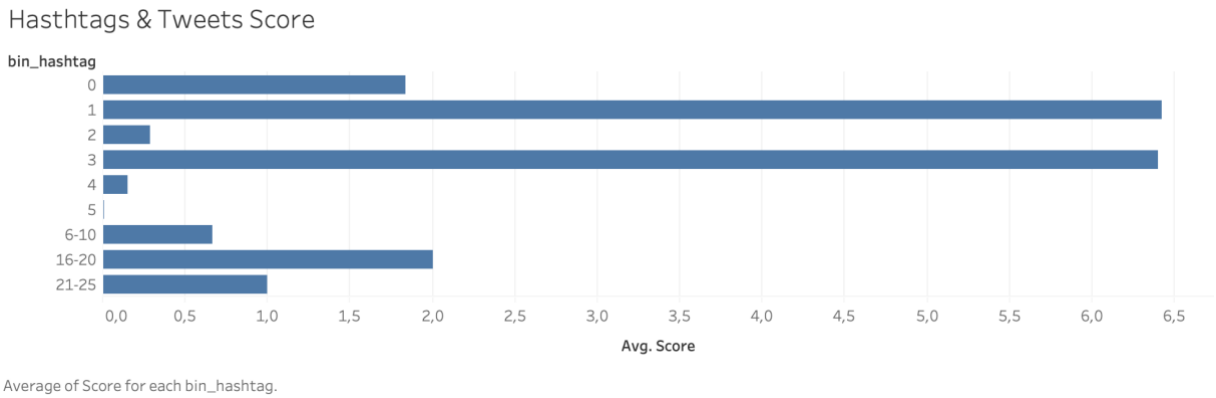


Figure 13: Tableau visual

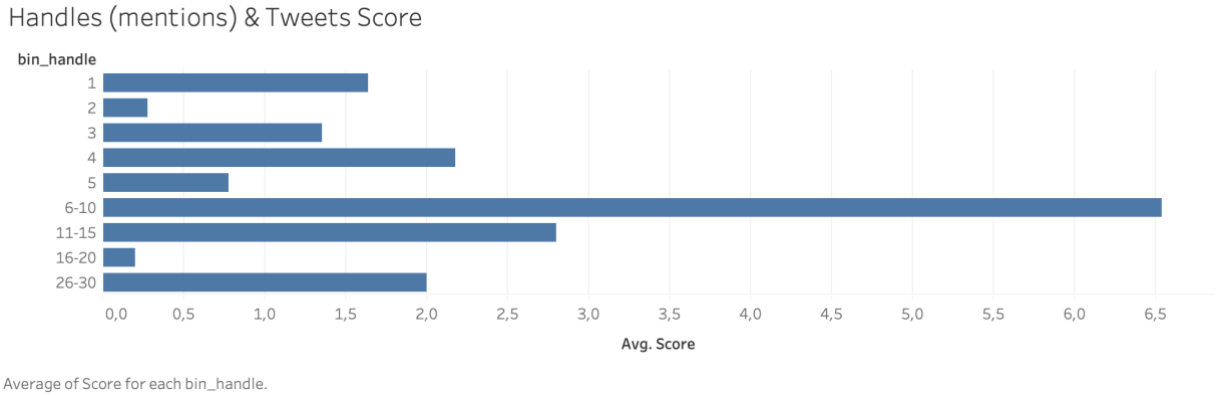


Figure 14: Tableau visual

2.3. Correlation between variables

Figure 15 shows the correlation between the numeric variables. *Year* is negatively correlated with *month,* because in 2021 there is data only for the first 4 months. This correlation helped in improving the prediction models by removing *year* from consideration. There is a strong positive correlation between the likes number and a tweet's score, as score represents the sum of the likes, replies, retweets, and quotes.

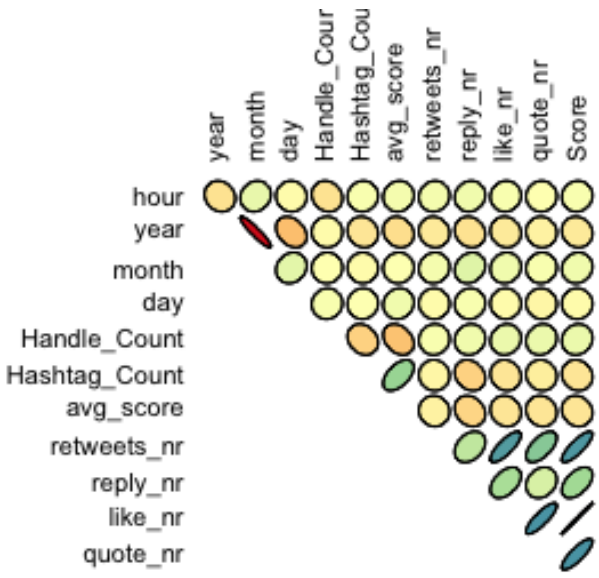


Figure 15: R visual

3. Data Preparation

Firstly, the json files for the different months were converted into one CSV file in Python. (see code Appendix)

3.1. Adding derived attributes

3.1.1. Python

Tweet score	Date	Text
• The target attribute, *Score,* has been created by summing up the tweets' number of likes, replies, retweets and quotes (retweets containing text). • See *figure 16*	• In *Figure 17*, a datetime object was created from the string in column 'created_at'. • In *Figure 18,* four new attributes were created: hour, month, day, year.	• For text pre-processing, a new attribute, *textCleanLem,* was created. For every tweet in the *Text* column, the words were lemmatized, the stopwords, the URLs, and the festival name were removed. See appendix for the code.

```
tweetsDf["Score"]=tweetsDf["retweets_nr"]+tweetsDf["reply_nr"]+tweetsDf["like_nr"]+tweetsDf['quote_nr']
```

Figure 16

```
from datetime import datetime
tweetsDf['created_at'] = tweetsDf['created_at'].apply(lambda x: datetime.strptime(x,'%Y-%m-%dT%H:%M:%S.000Z'))
```

Figure 17

```
tweetsDf['hour'] = tweetsDf['created_at'].apply(lambda x: x.hour)
tweetsDf['month'] = tweetsDf['created_at'].apply(lambda x: x.month)
tweetsDf['day'] = tweetsDf['created_at'].apply(lambda x: x.day)
tweetsDf['year'] = tweetsDf['created_at'].apply(lambda x: x.year)
```

Figure 18

3.1.2. R

Handle & Hashtag Count	Sentiment Score
• Str_count was used to derive two attributes: the number of hashtags and user mentions in the tweets. • See *Figure 19*	• Every tweet was split into words and represented as a data frame. An inner join was performed to merge this data frame with the happiness dictionary. The sentiment score of a tweet was then represented by the mean of the happiness score for each word in a tweet. • See *Figure 20*

```
# str_count is from package "stringr"
train.df$Handle_Count <- str_count(train.df$text, "@") # Returns the number of "@" in a tweet.

train.df$Hashtag_Count <- str_count(train.df$text, "#") # Returns the number of "#" in a tweet.
```

Figure 19

```
ntweets <- length(train.df$textCleanLem)
for (i in 1:ntweets) {
  this_tweet <- tolower(as.character(train.df$textCleanLem[i]))
  this_tweet_words <- strsplit(this_tweet, split = " ")
  this_tweet_words_vector <- unlist(this_tweet_words)
  this_tweet_words_df <- data.frame("word" = this_tweet_words_vector)
  single_tweet_merged <- merge(this_tweet_words_df , happiness_dictionary_happs, by = "word", type = "inner")
  train.df$avg_score[i] <- mean(single_tweet_merged$happs)
}


summary(train.df["avg_score"])
```

Figure 20

3.1.3. Tableau

- Bins for histograms

  In order to create histograms, 3 new attributes were created using the same logic: a conditional statement that specifies each bin's size. See the figures below.

bin_handle

```
if [Handle Count]=0  then '0'
elseif [Handle Count]=1  then '1'
elseif [Handle Count]=2  then '2'
elseif [Handle Count]=3 then '3'
elseif [Handle Count]=4 then '4'
elseif [Handle Count]=5 then '5'
elseif [Handle Count]>5 and [Handle Count]<=10 then '6-10'
elseif [Handle Count]>10 and [Handle Count]<=15 then '11-15'
elseif [Handle Count]>=16 and [Handle Count]<=20 then '16-20
elseif [Handle Count]>=21 and [Handle Count]<=25 then '21-25
elseif [Handle Count]>=26 and [Handle Count]<=30 then '26-30
END
```

Figure 21

bin_hashtag

```
if [Hashtag Count]=0  then '0'
elseif [Hashtag Count]=1  then '1'
elseif [Hashtag Count]=2  then '2'
elseif [Hashtag Count]=3 then '3'
elseif [Hashtag Count]=4 then '4'
elseif [Hashtag Count]=5 then '5'
elseif [Hashtag Count]>5 and [Hashtag Count]<=10 then '6-10'
elseif [Hashtag Count]>10 and [Hashtag Count]<=15 then '11-1
elseif [Hashtag Count]>=16 and [Hashtag Count]<=20 then '16-
elseif [Hashtag Count]>=21 and [Hashtag Count]<=25 then '21-
END
```

Figure 22

bin_happiness

```
if [Avg Score]>=0 and [Avg Score]<=1 then '0-1'
elseif [Avg Score]>=1 and [Avg Score]<=2 then '1-2'
elseif [Avg Score]>=2 and [Avg Score]<=3 then '2-3'
elseif [Avg Score]>=3 and [Avg Score]<=4 then '3-4'
elseif [Avg Score]>=4 and [Avg Score]<=5 then '4-5'
elseif [Avg Score]>=5 and [Avg Score]<=6 then '5-6'
elseif [Avg Score]>=6 and [Avg Score]<=7 then '6-7'
elseif [Avg Score]>=7 and [Avg Score]<=8 then '7-8'
elseif [Avg Score]>=8 and [Avg Score]<=9 then '8-9'
END
```

Figure 23

- Date

  A *date* object was created using the DATEPARSE function

Date

```
DATEPARSE('ddMMyyyy',[Day2]+[Month2]+STR([Year]))
```

Figure 24

3.2. Missing data

The 23 missing values (only 4% of the data) from the new attribute, *sentiment score*, were replaced with the mean of the column in R.

3.3. Data splitting

The data was split into training and validation sets in R. 75% of the rows were randomly sampled in the training set and the other 25% in the validation set.

```
]:
# Split data.
set.seed(201)
small.train.df <- train.df[sample(nrow(train.df), 0.75 * nrow(train.df)) ,]
valid.df <- train.df[-as.numeric(row.names(small.train.df)), ]
```

Figure 25

The testing set was generated from the 2 tweets in *Figure 26*. Both of these tweets' purpose is to sell tickets.

```
tweet1 <- "Go for a day or mix and match to see all your favourite artists! 1-day @Sonoma_Harvest  #SonomaHarvest Music Festival
tickets go on sale today at 10am PDT!. See your favourite artists including @BrunoMars, @Imaginedragons, @Mylie_Cyrus, @foals, @thegreatkhalid
View lineups + buy tickets >>bit.ly/SHMF19Tix"

tweet2 <- " 2.  One weekend isn't enough
We wanted more unforgettable music moments in wine country, so we created @Sonoma_Harvest #SonomaHarvest. Celebrate and smile with us! #celebration
Tickets ON SALE NOW! Snag them for your crew before we sell out: >>bit.ly/SHMF19Tix "
```

Figure 26

Four observations were generated from the tweets above, as they were both set to be posted at different times to test the hypotheses of the best times to post discussed in Section 2. *Avg_score* represents the sentiment score.

```
xtest <- rbind(xtest_1, xtest_2, xtest_3, xtest_4)
xtest
```

hour	month	Handle_Count	Hashtag_Count	avg_score
21	8	6	1	5.896
14	7	6	1	5.896
21	8	2	2	6.011
14	7	2	2	6.011

Figure 27

4. Prediction models

This section discusses 4 prediction models and 2 ensemble models. The accuracy of the models was measured with root mean square log error, to not penalize huge differences in the predicted and true values. 71% of the tweets have the score 0, thus we consider the log-transformation of the tweet's score for every prediction model.

```
> # Calculate the percentage of the tweets that have 0 retweets.
> length(which(train.df$Score == 0))/length(train.df$Score)
[1] 0.7180952
```

Figure 28

5 variables were considered as a starting point for all the models*: hour, month, handle_count, hashtag_count, happines_score.* These were the only numeric variables in the dataset except for *year, day, minutes, seconds*. *Year* was unconsidered because there was not enough data – only 6 months from 2020 and 4 months from 2021, so it was highly correlated with *month.*

4.1. Linear Regression

4.1.1. Model insights

A linear regression with interaction was fitted. The model output below offers insights into the significance, estimate, standard error, and t-value of the variables. We can see that some significant variables are the number of hashtags and mentions, or the interaction between month and happiness score.

```
Call:
lm(formula = log(1 + ytrain) ~ . + .^2, data = xtrain[, variablesSelected])

Residuals:
    Min      1Q  Median      3Q     Max
-0.9345 -0.1823 -0.1584  0.0713  3.8547

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)               -0.369329   1.535120  -0.241  0.81001
hour                      -0.068520   0.067091  -1.021  0.30777
month                      0.441676   0.154278   2.863  0.00443 **
Handle_Count              -0.532607   0.179109  -2.974  0.00313 **
Hashtag_Count             -0.309340   0.293503  -1.054  0.29258
avg_score                  0.038319   0.250366   0.153  0.87844
hour:month                 0.001298   0.001231   1.055  0.29232
hour:Handle_Count          0.001588   0.001765   0.900  0.36886
hour:Hashtag_Count        -0.004808   0.002733  -1.759  0.07935 .
hour:avg_score             0.011553   0.010945   1.056  0.29186
month:Handle_Count        -0.006717   0.003103  -2.165  0.03103 *
month:Hashtag_Count       -0.010258   0.005937  -1.728  0.08484 .
month:avg_score           -0.064359   0.024687  -2.607  0.00949 **
Handle_Count:Hashtag_Count -0.002543   0.005926  -0.429  0.66812
Handle_Count:avg_score     0.100627   0.030786   3.269  0.00118 **
Hashtag_Count:avg_score    0.058634   0.045694   1.283  0.20021
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5463 on 377 degrees of freedom
Multiple R-squared:  0.2042,    Adjusted R-squared:  0.1726
F-statistic: 6.451 on 15 and 377 DF,  p-value: 2.796e-12
```

Figure 29 – model output

4.1.2. Model improvements

The interaction between variables was considered to improve the model from a RMSLE of 0.597412612110352 to a RMSLE of 0.58696948184094. This means that the interaction between the terms has an impact on the target variable.

4.2. XGBoost

4.2.1. Model insights

An XGBoost model was fitted. The model offers insights about variables' importance, this time happiness score offering the most gain (the most insight on the target variable). See figure below.

Feature	Gain	Cover	Frequency
avg_score	0.46013526	0.51770550	0.34021305
hour	0.20192999	0.21578368	0.30559254
Handle_Count	0.16077300	0.11037577	0.11984021
month	0.11511404	0.10524461	0.17376831
Hashtag_Count	0.06204771	0.05089043	0.06058589

Figure 30 – variable importance

4.2.2. Model improvements

The best version of the model was constructed by trying different values for the *nround* parameter – different maximum number of trees. The values tried were 40,50,75,100. *Nround*=50 had the lowest error, RMSLE=0.556433932153292. Moreover, 2 more models were built by removing the least important variables, one by one. The XGBoost with all 5 variables had the lowest error.

4.3. Random forest

4.3.1. Model insights

A random forest was fitted using the 5 predictors below. *Hashtag_Count* is the most important variable for this model – removing it gives an 11% increase in the mean squared error.

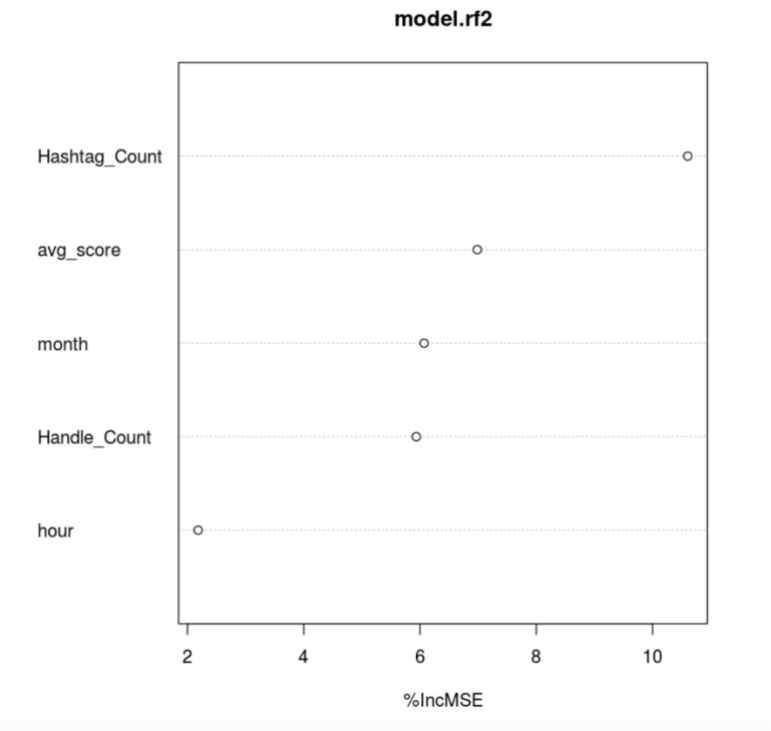


Figure 31 – variable importance

4.3.2. Model improvements

The model was tuned by modifying the parameter mtry – the number of variables randomly sampled for each tree. Mtry=2 gave lower error than mtry=3 (RMSLE= 0.534204606275396 < 0.535697017933687). Moreover, 3 more random forests were built by removing the least important variables, one by one. The random forest with all 5 variables had the lowest error.

4.4. Regression tree

4.4.1. Model insights

A regression tree was fitted using the 4 variables below. *Happiness score* is the most important predictor again – like in the xgboost model.

avg_score	38.916577690616
Handle_Count	20.1823712297963
month	11.5874191314517
hour	11.2217771454299

Figure 32 – variable importance

4.4.2. Model improvements

Removing the least important variable from the first tree, *Hashtag_Count*, improved the error to RMSLE= 0.622733749382179.

4.5. Ensemble models

The table below was created to select the best prediction models.

RMSLE	Model
0.5974126	linear_regression1
0.5869695	linear_regression_interaction
0.5564339	xgboost1
0.5669623	xgboost2
0.5701218	xgboost3
0.5356970	rf1
0.5342046	rf2
0.5643846	rf3
0.5441818	rf3
0.5630676	rf4
0.6227337	rtree1
0.6227337	rtree2
0.6316206	rtree3

Figure 33 – model results

4.5.1. Simple average model

Three average models were created – with the best 4,3 and 2 models. The one that performed the best was averaging the second random forest and first xgboost from *Figure 35*. The RMSLE was 0.537430513260946.

4.5.2. Weighted average

This final ensemble model is a function of the 3 best models from *Figure 35.* Different weights were tried for all the models – weights in seq(0.1,0.9,0.1). The matrix below captures the RMSLE of all these functions. A cell's value is 0 when the sum of the weights is higher than 1.

```
m1_weight <- seq(0.1,0.9,0.1)
m2_weight <- seq(0.1,0.9,0.1)

# set up a matrix to store the Brier skill scores
RMSLE_matrix <- matrix(0,9,9)
for (i in 1:9) {
    for (j in 1:9){
        if (m2_weight[j]+ m1_weight[i] > 1) next
        ensemble_pred <- m1_weight[i]*rf.pred.unlogged +xgb.pred.unlogged*m2_weight[j] + lm.2.pred.unlogged*(1-
                                                                                    m1_weight[i] - m2_weight[i])
        RMSLE_matrix[i,j] <- RMSLE(ensemble_pred, valid.df$Score)
    }
}
```

Figure 34

```
RMSLE_matrix
```

0.5637825	0.5572505	0.5545717	0.5546927	0.5569155	0.5607490	0.5658334	0.5718968	0.5787302
0.5558011	0.5491001	0.5462316	0.5461753	0.5482454	0.5519540	0.5569406	0.5629313	0.0000000
0.5508474	0.5436868	0.5403838	0.5399393	0.5416724	0.5450942	0.5498401	0.0000000	0.0000000
0.5495987	0.5416530	0.5376345	0.5365543	0.5377310	0.5406701	0.0000000	0.0000000	0.0000000
0.5534752	0.5443029	0.5391811	0.5371199	0.5374305	0.0000000	0.0000000	0.0000000	0.0000000
0.5664573	0.5551486	0.5481305	0.5443909	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.6287228	0.6012818	0.5843025	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.5337317	0.5331524	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.5332131	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

Figure 35

The element in row 8, column 2 corresponds to the lowest error. According to the table, the weights put on the predictions from the random forest, xgboost and linear regression should be 0.8,0.2,and 0. The lowest RMSLE error achieved by the weighted average ensemble is 0.5331524.

4.6. Best prediction model

The weighted average model has the lowest RMSLE. This model was used to make predictions on the testing set generated in Section 3. The second option is predicted to have the highest reach. (see figures below).

```
xtest <- rbind(xtest_1, xtest_2, xtest_3, xtest_4)
xtest
```

hour	month	Handle_Count	Hashtag_Count	avg_score
21	8	6	1	5.896
14	7	6	1	5.896
21	8	2	2	6.011
14	7	2	2	6.011

Figure 27 – test set

1	2.55478907346168
2	5.46222663789033
3	1.47947038157358
4	1.37576411089721

Figure 36 - predictions

5. Conclusions and recommendations

The best prediction model was the weighted average with weights of 0.8 on the random forest and 0.2 on the XGBoost. Therefore, it can be concluded that the most important variables are the 5 variables used in the order of the random forest output in *Figure 32:*

- Hashtags number
- Happiness score
- Month
- User mentions number
- Hour

The first two observations in the test set were predicted to have higher score than the others – *Figure 13* shows that tweets containing one hashtag have an average tweet score of 6.5 while tweets containing two hashtags have an average tweet score of 0.4. Moreover, *Figure 11* shows that tweets with happiness score between 5-6 have an average tweet score of 2.9 while tweets with happiness score between 6-7 have an average tweet score of 0.6.

Taking into account all 5 variables, there was a percentage increase in a tweet score of 298% from tweet 4 to tweet 2. Increasing a tweet's reach by 298% means there will also be an increase in the tickets bought through Twitter for the festival taking place in September.

BottleRock Napa is recommended to prioritise these 5 variables in the ordered mentioned and look at the figures in Section 2 when choosing the values of these. In order to maximise a tweet's reach, a tweet should:

- Have 1 hashtags
- Have happiness score between 5-6
- Be posted in July
- Have between 6-10 user mentions
- Be posted between 15:00-16:00 UTC

References:

1. BottleRock Doubles Revenue With Using atVenu [Internet]. Atvenu.com. 2020 [cited 13 April 2021]. Available from: https://www.atvenu.com/case-studies/bottlerock
2. BottleRock Napa Valley - Wikipedia [Internet]. En.wikipedia.org. 2021 [cited 15 April 2021]. Available from: https://en.wikipedia.org/wiki/BottleRock_Napa_Valley
3. Harrington R. Tweet to the Rhythm: What Twitter Tells Us About Music Festivals [Internet]. Medium. 2019 [cited 14 April 2021]. Available from: https://medium.com/compassred-data-blog/tweet-to-the-rhythm-what-twitter-tells-us-about-music-festivals-965020872f3a

Code Appendix

Three code files were used: one Python file, one R file, and one Jupyter Notebook R file. Two R files were used because some libraries were not working on the server and some functions were not working in R.

2. Understand the data

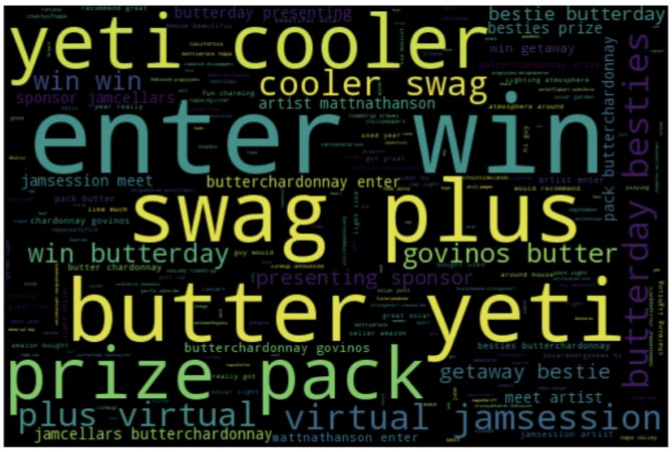
2.2.1. Text data – Python

```
In [ ]: ################################ SECTION 2: UNDERSTAND THE DATA  ######################################
        #2.2.1. Text data
```

```
In [76]: # Generate the wordcloud
corpus = ' '.join(tweetsDf.textCleanLem.tolist())
def cloud():
    wordcloud = WordCloud(background_color='black',height=400,width=600,stopwords=(keyWord)).generate(corpus)
    pylab.figure(figsize=(18,7))
    pylab.axis("off")
    pylab.imshow(wordcloud)
```

```
In [77]: # Plot the wordcloud
%pylab inline
cloud()
fig.savefig('words_clean.png')
```

```
Populating the interactive namespace from numpy and matplotlib
WARNING: pylab import has clobbered these variables: ['datetime']
`%matplotlib` prevents importing * from pylab and numpy
```

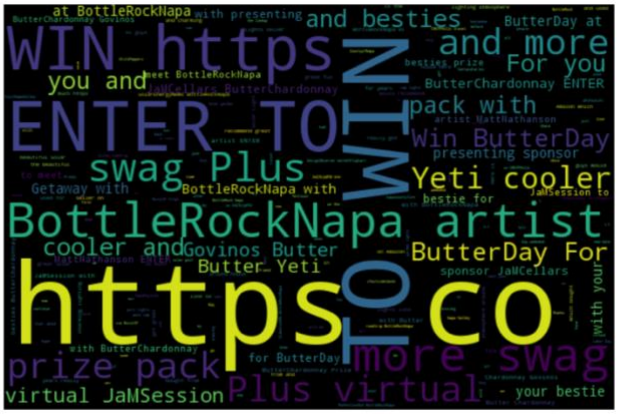



```
In [78]: # Generate the wordcloud
corpus = ' '.join(tweetsDf.text.tolist())
def cloud():
    wordcloud = WordCloud(background_color='black',height=400,width=600,stopwords=(keyWord)).generate(corpus)
    pylab.figure(figsize=(18,7))
    pylab.axis("off")
    pylab.imshow(wordcloud)
```

```
In [45]: # Plot the wordcloud
%pylab inline
cloud()
fig.savefig('words.png')
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
Populating the interactive namespace from numpy and matplotlib
```



In [79]:
```
# Create a list of lists containing lowercase words for each tweet
words_in_tweet = [tweet.lower().split() for tweet in tweetsDf["textCleanLem"]]
words_in_tweet[2]
```

Out[79]:
```
['interview',
 'twin',
 'xl',
 'wearetwinxl',
 'ride',
 'good',
 'vibe',
 'way',
 'new',
 'act',
 'career',
 'musician',
 'play',
 'next',
 'may',
 'cameronwalker']
```

In [81]:
```
all_words = list(itertools.chain(*words_in_tweet))
all_words[2]
```

Out[81]:
```
'cut'
```

In [83]:
```
# Count each word across all tweets
counts = collections.Counter(all_words)
counts.most_common(15)
```

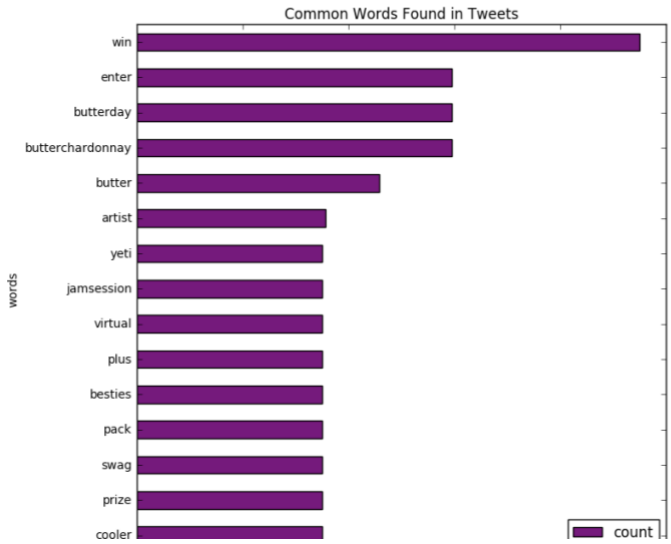
Out[83]:
```
[('win', 475),
 ('butterchardonnay', 298),
 ('butterday', 298),
 ('enter', 298),
 ('butter', 229),
 ('artist', 178),
 ('cooler', 175),
 ('prize', 175),
 ('swag', 175),
 ('pack', 175),
 ('besties', 175),
 ('plus', 175),
 ('virtual', 175),
 ('jamsession', 175),
 ('yeti', 175)]
```

n [84]:
```
# Process for word counts


clean_tweets_nsw = pd.DataFrame(counts.most_common(15),
                                columns=['words', 'count'])
clean_tweets_nsw
```

ut[84]:

	words	count
0	win	475
1	butterchardonnay	298
2	butterday	298
3	enter	298
4	butter	229
5	artist	178
6	cooler	175
7	prize	175
8	swag	175
9	pack	175
10	besties	175
11	plus	175
12	virtual	175
13	jamsession	175
14	yeti	175

n [88]:
```
fig, ax = plt.subplots(figsize=(8, 8))

# Plot horizontal bar graph
clean_tweets_nsw.sort_values(by='count').plot.barh(x='words',
                      y='count',
                      ax=ax,
                      color="purple")

ax.set_title("Common Words Found in Tweets")

plt.show()
fig.savefig('words_clean_count.png')
```



2.3. Correlation between variables – R

```
################################### SECTION 2: UNDERSTAND THE DATA  ########################################

#2.3. Correlation between the variables

#summary stats for the first 12 census data columns

summary(train.df[,2:20])

#numeric columns
numeric.df<-train.df[, c("hour","year","month","day","Handle_Count","Hashtag_Count","avg_score","retweets_nr","reply_nr","like_nr","quote_nr","Score")]

#correlation plot

library(ellipse);library(RColorBrewer);    options(repr.plot.height=10)
my_colors=colorRampPalette(brewer.pal(5, "Spectral"))(100)

data=cor(numeric.df,use="complete.obs")

plotcorr(data , col=my_colors[data*50+50], mar = c(0,0,0,0),
         cex.lab=0.7, type = "upper" , diag=FALSE)
```

3. Date preparation

Python:

```
In [ ]: #-------------------------------------# # Library importations # #-------------------------------------#

In [58]: import os
         import tweepy as tw
         import pandas as pd
         import itertools
         import collections
         from datetime import datetime

In [59]: import pandas as pd
         import numpy as np

         # JSON is short for JavaScript Object Notation, and is a way to store information in an
         # organized, easy-to-access manner.
         import json
         # The re module provides regular expression matching operations.
         import re

         # NLTK is a leading platform for building Python programs to work with natural language data.
         from nltk.corpus import stopwords
         from nltk.tokenize import WordPunctTokenizer
         from nltk.stem.wordnet import WordNetLemmatizer

         # tweepy is a library for accessing the Twitter API.
         import tweepy
         from tweepy import OAuthHandler
         from tweepy.streaming import StreamListener

         # wordcloud is a library for generating wordcloud figures.
         from wordcloud import WordCloud
         import string
         import pickle

         # The time function is used to set the time limit of "listening" the tweets.
         import time

In [60]: import nltk
         nltk.download('stopwords') # NOTE: ONCE YOU HAVE DOWNLOAD THESE CORPORA, YOU CAN COMMENT OUT THIS LINE.
         nltk.download('wordnet') # NOTE: ONCE YOU HAVE DOWNLOAD THESE CORPORA, YOU CAN COMMENT OUT  THIS LINE.

         [nltk_data] Downloading package stopwords to
         [nltk_data]     /home/tan6tur/nltk_data...
         [nltk_data]   Package stopwords is already up-to-date!
         [nltk_data] Downloading package wordnet to /home/tan6tur/nltk_data...
         [nltk_data]   Package wordnet is already up-to-date!
Out[60]: True

In [ ]: ################################## SECTION 3: PREPARE THE DATA  ########################################

In [61]:
         # merging the data files and changing the format to CSV

         tweetList = []

         files=["July.json","august.json","September.json","October.json","November.json","December.json","January.json","Februa
         for i in range (0,10):
             with open(files[i]) as in_file:  # change the name of your json file.
                 for line in in_file:
                     if line.strip():
                         tweets = json.loads(line)
                         data = tweets['data']
                         print(data[0].keys())
                         for tweet in data:
                             keys = tweet.keys()

                             mentions = list(map(lambda x: x['username'],tweet['entities']['mentions']))
                             row = (
                             tweet['author_id'],                     # author id
                             tweet['created_at'],                    # created at
                             tweet['public_metrics']['retweet_count'],# retweets
                             tweet['public_metrics']['reply_count'],  #replies
                             tweet['public_metrics']['like_count'],   #likes
                             tweet['public_metrics']['quote_count'],  #quotes
                             str(mentions).strip('[]'),              #mentions
                             tweet['id'],                            #id
                             tweet['text'],                          #text
                             tweet['lang'],                          #language
                             tweet['source'],                        #source
                             tweet['reply_settings']                 #reply settings
                             )
                             values = [value for value in row]
                             tweetList.append(values)

         colNames = 'author_id,created_at,retweets_nr,reply_nr,like_nr,quote_nr,mentions_usernames,id,text,language,source,reply

         # If the account does not share location information, we replace it as NaN.
         tweetsDf = pd.DataFrame(tweetList,columns=colNames)
```

```
dict_keys(['entities', 'lang', 'created_at', 'text', 'in_reply_to_user_id', 'public_metrics', 'id', 'author_id', 'rep
ly_settings', 'source'])
dict_keys(['id', 'lang', 'created_at', 'text', 'public_metrics', 'entities', 'author_id', 'reply_settings', 'sourc
e'])
dict_keys(['reply_settings', 'id', 'created_at', 'text', 'in_reply_to_user_id', 'public_metrics', 'lang', 'author_i
d', 'entities', 'source'])
dict_keys(['lang', 'id', 'created_at', 'text', 'in_reply_to_user_id', 'public_metrics', 'entities', 'author_id', 'rep
ly_settings', 'source'])
dict_keys(['lang', 'id', 'created_at', 'text', 'in_reply_to_user_id', 'public_metrics', 'entities', 'author_id', 'rep
ly_settings', 'source'])
dict_keys(['lang', 'id', 'created_at', 'text', 'in_reply_to_user_id', 'public_metrics', 'entities', 'author_id', 'rep
ly_settings', 'source'])
dict_keys(['lang', 'reply_settings', 'id', 'created_at', 'text', 'in_reply_to_user_id', 'public_metrics', 'entities',
'author_id', 'source'])
dict_keys(['lang', 'id', 'created_at', 'text', 'public_metrics', 'entities', 'author_id', 'reply_settings', 'sourc
e'])
dict_keys(['lang', 'created_at', 'entities', 'text', 'public_metrics', 'id', 'author_id', 'reply_settings', 'sourc
e'])
dict_keys(['lang', 'created_at', 'entities', 'text', 'public_metrics', 'id', 'author_id', 'reply_settings', 'sourc
e'])
```

2]:
```
tweetsDf.head()
```

2]:

	author_id	created_at	retweets_nr	reply_nr	like_nr	quote_nr	mentions_usernames	id	
0	20123238	2020-07-27T13:50:12.000Z	0	0	0	0	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1287747118891843587	@BottleRockN @JaMCellars @ElleKingMusi Why...
1	2358609954	2020-07-25T21:07:09.000Z	0	0	1	0	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1287132303257477120	@BottleRockN @JaMCellars @ElleKingMusi w...
2	59311659	2020-07-25T19:30:00.000Z	0	0	1	0	'wearetwinxl', 'BottleRockNapa', 'CameronWalker'	1287107853761875970	INTERVIEW: Tw XL (@wearetwi rides 'Good'...
3	1044741391803404289	2020-07-25T17:56:38.000Z	0	0	0	0	'BottleRockNapa', 'ChiliPeppers'	1287084356385767426	@BottleRockN @ChiliPeppers Talento BRASILEI...
4	17606101	2020-07-25T05:30:49.000Z	0	0	1	0	'BottleRockNapa', 'NRateliff', 'SantanaCarlos'	1286896667921276929	@BottleRockN @NRateliff @SantanaCarl Love...

4]:
```
index = tweetsDf.index
len(index)
```

4]: 525

3.1. Adding derived attributes

3.1.1. Python

In [ ]:
```
#3.1.1. Addind derived attributes in Python
```

In [65]:
```
#create a datetime object
tweetsDf['created_at'] = tweetsDf['created_at'].apply(lambda x: datetime.strptime(x,'%Y-%m-%dT%H:%M:%S.000Z'))
```

In [66]:
```
tweetsDf.head()
```

Out[66]:

	author_id	created_at	retweets_nr	reply_nr	like_nr	quote_nr	mentions_usernames	id	text	la
0	20123238	2020-07-27 13:50:12	0	0	0	0	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1287747118891843587	@BottleRockNapa @JaMCellars @ElleKingMusic Why...	e
1	2358609954	2020-07-25 21:07:09	0	0	1	0	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1287132303257477120	@BottleRockNapa @JaMCellars @ElleKingMusic I w...	e
2	59311659	2020-07-25 19:30:00	0	0	1	0	'wearetwinxl', 'BottleRockNapa', 'CameronWalker'	1287107853761875970	INTERVIEW: Twin XL (@wearetwinxl) rides 'Good'...	e
3	1044741391803404289	2020-07-25 17:56:38	0	0	0	0	'BottleRockNapa', 'ChiliPeppers'	1287084356385767426	@BottleRockNapa @ChiliPeppers Talento BRASILEI...	p
4	17606101	2020-07-25 05:30:49	0	0	1	0	'BottleRockNapa', 'NRateliff', 'SantanaCarlos'	1286896667921276929	@BottleRockNapa @NRateliff @SantanaCarlos Love...	e

In [67]:
```
#add date derived attribures
tweetsDf['hour'] = tweetsDf['created_at'].apply(lambda x: x.hour)
tweetsDf['month'] = tweetsDf['created_at'].apply(lambda x: x.month)
tweetsDf['day'] = tweetsDf['created_at'].apply(lambda x: x.day)
tweetsDf['year'] = tweetsDf['created_at'].apply(lambda x: x.year)
```

In [68]:
```
tweetsDf.head()
```

Out[68]:

like_nr	quote_nr	mentions_usernames	id	text	language	source	reply_settings	hour	month	day	year
0	0	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1287747118891843587	@BottleRockNapa @JaMCellars @ElleKingMusic Why...	en	Twitter for Android	everyone	13	7	27	2020
1	0	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1287132303257477120	@BottleRockNapa @JaMCellars @ElleKingMusic I w...	en	Twitter for iPhone	everyone	21	7	25	2020
1	0	'wearetwinxl', 'BottleRockNapa', 'CameronWalker'	1287107853761875970	INTERVIEW: Twin XL (@wearetwinxl) rides 'Good'...	en	TweetDeck	everyone	19	7	25	2020
0	0	'BottleRockNapa', 'ChiliPeppers'	1287084356385767426	@BottleRockNapa @ChiliPeppers Talento BRASILEI...	pt	Twitter for Android	everyone	17	7	25	2020
1	0	'BottleRockNapa', 'NRateliff', 'SantanaCarlos'	1286896667921276929	@BottleRockNapa @NRateliff @SantanaCarlos Love...	en	Twitter for Android	everyone	5	7	25	2020

In [69]:
```
# This function is based on a function provided by
# https://github.com/AndreyKarasev/Data_Science_Class
def cleanLem (text, keyWord):
    '''Function
    --------
    Return a tweet after removing stop words, urls, RTs, the keyword,
    and characters that are not letters.
    Also lemmitizes words in tweets

    Parameters
    ----------
    text : str
        Original tweet
    keyWord : str
        The keyword you want to exclude in the wordcloud

    Returns
    -------
    clean tweet : str'''
    #  Tokenize a text into a sequence of alphabetic and non-alphabetic characters
    tokenizer = WordPunctTokenizer()

    # Lemmatize a word
    lemmatizer = WordNetLemmatizer()

    # Remove stopwords
    stopWords = stopwords.words('english')

    # Remove words in customised list
    customList = ['amp']
    exclude = stopWords + customList + keyWord.split()

    # Remove strings that match the pattern of '\s*RT\s\S*\s'
    # \s*: any number whitespace characters (include 0 whitespace)
    # RT: RT
    # \s: one whitespace
    # \S*: any non-whitespace characters (include punctuation)
    text = re.sub(r'\s*RT\s\S*\s', '', text)  # delete any number of RTs
    text = re.sub(r'http\S*', '', text)  # remove url

    # tokenizer will seperate alphabetic and non-alphabetic characters.
    # Need to covert to lowercase characters, or later the algorithm will treat lowercase
    # and uppercase characters as different characters when we generate wordclouds.
    words = tokenizer.tokenize(text.lower())
    cleanWords = [word for word in words if word not in exclude]
    lemWords = []
    for word in cleanWords:
        # keeps only alphabetic characters
        letters=''.join([letter for letter in word if letter.isalpha()])
        lemWords.append(lemmatizer.lemmatize(letters))
        continue
    return  ' '.join(lemWords)
```

In [70]:
```
keyWord = 'bottlerocknapa'
tweetsDf['textCleanLem'] = tweetsDf.text.apply(cleanLem, keyWord = 'bottlerocknapa')
```

In [71]:
```
tweetsDf.head()
```

Out[71]:

_nr	mentions_usernames	id	text	language	source	reply_settings	hour	month	day	year	textCleanLem
	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1287747118891843587	@BottleRockNapa @JaMCellars @ElleKingMusic Why...	en	Twitter for Android	everyone	13	7	27	2020	jamcellars ellekingmusic cut santana song ...
	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1287132303257477120	@BottleRockNapa @JaMCellars @ElleKingMusic I w...	en	Twitter for iPhone	everyone	21	7	25	2020	jamcellars ellekingmusic set peppered heavy...
	'wearetwinxl', 'BottleRockNapa', 'CameronWalker'	1287107853761875970	INTERVIEW: Twin XL (@wearetwinxl) rides 'Good'...	en	TweetDeck	everyone	19	7	25	2020	interview twin xl wearetwinxl ride good v...
	'BottleRockNapa', 'ChiliPeppers'	1287084356385767426	@BottleRockNapa @ChiliPeppers Talento BRASILEI...	pt	Twitter for Android	everyone	17	7	25	2020	chilipeppers talento brasileiro
	'BottleRockNapa', 'NRateliff', 'SantanaCarlos'	1286896667921276929	@BottleRockNapa @NRateliff @SantanaCarlos Love...	en	Twitter for Android	everyone	5	7	25	2020	nrateliff santanacarlos loved

```
dtype: int64
```

[73]:
```
#creating a derived target attribute
tweetsDf["Score"]=tweetsDf["retweets_nr"]+tweetsDf["reply_nr"]+tweetsDf["like_nr"]+tweetsDf['quote_nr']
```

[74]:
```
tweetsDf.head()
```

t[74]:

	author_id	created_at	retweets_nr	reply_nr	like_nr	quote_nr	mentions_usernames	id	text	la
0	20123238	2020-07-27 13:50:12	0	0	0	0	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1287747118891843587	@BottleRockNapa @JaMCellars @ElleKingMusic Why...	e
1	2358609954	2020-07-25 21:07:09	0	0	1	0	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1287132303257477120	@BottleRockNapa @JaMCellars @ElleKingMusic I w...	e
2	59311659	2020-07-25 19:30:00	0	0	1	0	'wearetwinxl', 'BottleRockNapa', 'CameronWalker'	1287107853761875970	INTERVIEW: Twin XL (@wearetwinxl) rides 'Good'...	e
3	1044741391803404289	2020-07-25 17:56:38	0	0	0	0	'BottleRockNapa', 'ChiliPeppers'	1287084356385767426	@BottleRockNapa @ChiliPeppers Talento BRASILEI...	p
4	17606101	2020-07-25 05:30:49	0	0	1	0	'BottleRockNapa', 'NRateliff', 'SantanaCarlos'	1286896667921276929	@BottleRockNapa @NRateliff @SantanaCarlos Love...	e

[75]:
```
# Save the preprocessed tweets into a CSV file
tweetsDf.to_csv('tweetsFestival.csv')
```

3.1.2. R

```
## Step1: Load the packages and read in the data

library(stringr)
library(jsonlite)
library(plyr)
library(lubridate)
library(parallel)
library(foreach)

library(doSNOW)
library(foreach)
library(parallel)

library(plyr); library(dplyr)

################################### SECTION 3: PREPARE THE DATA  ########################################

# Set up your multiple cores as separate workers and then make them a cluster.
workers <- detectCores()
workers # How many cores do you have?
cluster <- makeCluster(workers, type = "SOCK")
registerDoSNOW(cluster)

# Load in the training set.
train.df<-tweetsFestival


dim(train.df)

```

```r
# 3.1.2. Adding derived attributes in R

# str_count is from package "stringr"

train.df$Handle_Count <- str_count(train.df$text, "@") # Returns the number of "@" in a tweet.

train.df$Hashtag_Count <- str_count(train.df$text, "#") # Returns the number of "#" in a tweet.


# Read in the "Happiness Dictionary".
happiness_dictionary <- read.csv("Hedonometer.csv")
head(happiness_dictionary)

happiness_dictionary_happs <- happiness_dictionary[ , c(2, 3)]

# We use the column "happs" to score the words.
head(happiness_dictionary_happs)

ntweets <- length(train.df$textCleanLem)
for (i in 1:ntweets) {
  this_tweet <- tolower(as.character(train.df$textCleanLem[i]))
  this_tweet_words <- strsplit(this_tweet, split = " ")
  this_tweet_words_vector <- unlist(this_tweet_words)
  this_tweet_words_df <- data.frame("word" = this_tweet_words_vector)
  single_tweet_merged <- merge(this_tweet_words_df , happiness_dictionary_happs, by = "word", type = "inner")
  train.df$avg_score[i] <- mean(single_tweet_merged$happs)
}
```

3.2. Missing values (R)

```r
#3.2. Missing data

summary(train.df["avg_score"])


train.df$avg_score<-ifelse(is.na(train.df$avg_score),
                           mean(train.df$avg_score,na.rm=TRUE),
                           train.df$avg_score)

summary(train.df["avg_score"])


# Write train.df into a CSV file.
write.csv(train.df, '/Users/taniaturdean/Desktop/data analytics 2 twitter_train_allVariables.csv', row.names=FALSE)
```

3.3. Data splitting (R)

In [ ]:
```
# 3.3. Data Splitting
```

In [9]:
```
# Split data.
set.seed(201)
small.train.df <- train.df[sample(nrow(train.df), 0.75 * nrow(train.df)) ,]
valid.df <- train.df[-as.numeric(row.names(small.train.df)), ]
```

In [10]:
```
# Pull apart the Xs and Ys into separate frames.
xtrain <- small.train.df[ , -c(4,5,6,7,19)]
xvalid <- valid.df[ , -c(4,5,6,7,19)]
ytrain <- small.train.df[ , 19]
yvalid <- valid.df[ , 19]
```

In [11]:
```
# Highly skewed data
summary(train.df$Score)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.000   0.000   0.000   1.065   1.000 102.000 
```

In [12]:
```
# Calculate the percentage of the tweets that have 0 retweets.
length(which(train.df$Score == 0))/length(train.df$Score)
```

0.718095238095238

In [ ]:
```
# create the test set
```

In [1]:
```
# Option 1:
xtest_1 <- data.frame(hour=21,month=8,Handle_Count=6,Hashtag_Count=1,avg_score=5.896) #good+bad

# Option 2:
xtest_2 <- data.frame(hour=14,month=7,Handle_Count=6,Hashtag_Count=1,avg_score=5.896) #good+good

# Option 3:
xtest_3 <- data.frame(hour=21,month=8,Handle_Count=2,Hashtag_Count=2,avg_score=6.011 ) #bad+bad

# Option 4:
xtest_4 <- data.frame(hour=14,month=7,Handle_Count=2,Hashtag_Count=2,avg_score=6.011 )#bad + good
```

In [2]:
```
xtest <- rbind(xtest_1, xtest_2, xtest_3, xtest_4)
xtest
```

hour	month	Handle_Count	Hashtag_Count	avg_score
21	8	6	1	5.896
14	7	6	1	5.896
21	8	2	2	6.011
14	7	2	2	6.011

4. Prediction models (R)

4.1. Linear Regression

In [ ]:
```
#-----------------------------------# # Library importations # #-----------------------------------#
```

In [2]:
```
.libPaths("/usr/local/lib/R/site-library") # You need to run this line if you are using Jupyter server
library(stringr)
library(jsonlite)
library(plyr)
library(lubridate)
library(parallel)
library(rpart)
```

```
Error in library(stringr): Package 'stringr' version 1.2.0 cannot be unloaded
Traceback:

1. library(stringr)
2. stop(gettextf("Package %s version %s cannot be unloaded", sQuote(package),
 .     oldversion), domain = NA)
```

In [3]:
```
library(xgboost)
```

In [4]:
```
library(foreach)
library(randomForest)
```

```
randomForest 4.6-12
Type rfNews() to see new features/changes/bug fixes.
```

In [5]:
```
library(doSNOW)
library(foreach)
library(parallel)

# Set up your multiple cores as separate workers and then make them a cluster.
workers <- detectCores()
workers # How many cores do you have?
cluster <- makeCluster(workers, type = "SOCK")
registerDoSNOW(cluster)
```

```
Loading required package: iterators
Loading required package: snow

Attaching package: 'parallel'

The following objects are masked from 'package:snow':

    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
    clusterExport, clusterMap, clusterSplit, makeCluster, parApply,
    parCapply, parLapply, parRapply, parSapply, splitIndices,
    stopCluster
```

```
8
```

In [ ]:
```
################################## SECTION 3: PREPARE THE DATA  ########################################
```

In [6]:
```
# Load in the training set.
train.df <- read.csv("data analytics 2 twitter_train_allVariables.csv")
train.df[1,]
```

X1	author_id	created_at	retweets_nr	reply_nr	like_nr	quote_nr	mentions_usernames	id	text	...	reply_settings	ho
0	20123238	2020-07-27 13:50:12	0	0	0	0	'BottleRockNapa', 'JaMCellars', 'ElleKingMusic'	1.287747e+18	@BottleRockNapa @JaMCellars @ElleKingMusic Why did you cut Santana down to a 2 song set (slight exaggeration)? At least Muse didn't	...	everyone	13

14	7	2	2	6.011

In [ ]:
```
################################## SECTION 4: PREDICTION MODELS ######################################
```

In [ ]:
```
# Root Mean Squared Logarithmic Error (RMSLE)
RMSLE <- function(predictions, realizations){
    sqrt(sum((log(predictions+1) - log(realizations+1))^2)/length(predictions))
}
```

In [13]:
```
# 4.1. Linear Regression
# Select variables.
# hour, month, handle_count, hashtag_count, happines_score
variablesSelected <- c(10,11,15,16,17)
```

In [14]:
```
# Fit the linear regression
lm.1 <- lm(log(1 + ytrain) ~ ., data = xtrain[, variablesSelected])
summary(lm.1)
```

```
Call:
lm(formula = log(1 + ytrain) ~ ., data = xtrain[, variablesSelected])

Residuals:
    Min      1Q  Median      3Q     Max
-0.9406 -0.2445 -0.1758  0.0919  3.8302

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.194189   0.481704   0.403 0.687076
hour           0.007405   0.004157   1.781 0.075662 .
month          0.033873   0.008803   3.848 0.000139 ***
Handle_Count   0.027636   0.011280   2.450 0.014726 *
Hashtag_Count -0.071340   0.017699  -4.031  6.7e-05 ***
avg_score     -0.022806   0.077524  -0.294 0.768775
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5658 on 387 degrees of freedom
Multiple R-squared:  0.1241,    Adjusted R-squared:  0.1128
F-statistic: 10.97 on 5 and 387 DF,  p-value: 6.923e-10
```

In [15]:
```
lm.1.pred <- predict(lm.1, xvalid)

lm.1.pred.unlogged <- exp(lm.1.pred) - 1  # Unlog the y variable.
RMSLE(lm.1.pred.unlogged, yvalid)  #Score the lm model.
```

0.597412612110352

In [16]:
```

# The code below saves the results in a data frame model.results so that
# we can show and compare the error rates for all the models later.

model.results <- data.frame(RMSLE=0.597412612110352, Model="linear_regression1")
```

In [17]:
```
# Fit the linear regression with interaction
lm.2 <- lm(log(1 + ytrain) ~ . + .^2, data = xtrain[, variablesSelected])
summary(lm.2)
```

```
Call:
lm(formula = log(1 + ytrain) ~ . + .^2, data = xtrain[, variablesSelected])

Residuals:
    Min      1Q  Median      3Q     Max
-0.9345 -0.1823 -0.1584  0.0713  3.8547

Coefficients:
                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                  -0.369329   1.535120  -0.241  0.81001
hour                         -0.068520   0.067091  -1.021  0.30777
month                         0.441676   0.154278   2.863  0.00443 **
Handle_Count                 -0.532607   0.179109  -2.974  0.00313 **
Hashtag_Count                -0.309340   0.293503  -1.054  0.29258
avg_score                     0.038319   0.250366   0.153  0.87844
hour:month                    0.001298   0.001231   1.055  0.29232
hour:Handle_Count             0.001588   0.001765   0.900  0.36886
hour:Hashtag_Count           -0.004808   0.002733  -1.759  0.07935 .
hour:avg_score                0.011553   0.010945   1.056  0.29186
month:Handle_Count           -0.006717   0.003103  -2.165  0.03103 *
month:Hashtag_Count          -0.010258   0.005937  -1.728  0.08484 .
month:avg_score              -0.064359   0.024687  -2.607  0.00949 **
Handle_Count:Hashtag_Count   -0.002543   0.005926  -0.429  0.66812
Handle_Count:avg_score        0.100627   0.030786   3.269  0.00118 **
Hashtag_Count:avg_score       0.058634   0.045694   1.283  0.20021
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5463 on 377 degrees of freedom
```

F-statistic: 6.491 on 15 and 377 DF, p-value: 2.796e-12

8]:
```
lm.2.pred <- predict(lm.2, xvalid)
lm.2.pred.unlogged <- exp(lm.2.pred) - 1  # Unlog the y variable.
RMSLE(lm.2.pred.unlogged, yvalid)  #Score the lm model.
```

0.58696948184094

9]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE = 0.58696948184094, Model="linear_regression_interaction"))
```

4.2. XGBoost

```
]: # 4.2. XGBoost
```

```
)]: # Create matrix for xgboost.
xtrain.xgb <- model.matrix(~ 0 + ., data = xtrain[, variablesSelected])
xvalid.xgb <- model.matrix(~ 0 + ., data = xvalid[, variablesSelected])

# Fit the xgboost model
model.xgb <- xgboost(xtrain.xgb, log(ytrain+1), nround = 50, verbose = 0, nthred = workers)  #nround=50 (0.556433932153
                                                                                                #=100(0.55771946680
                                                                                              #=75 (0.557078972676131)
                                                                                     #=40  (0.55689691697435)

# Generate predictions
xgb.pred <- predict(model.xgb, xvalid.xgb)
xgb.pred.unlogged <- exp(xgb.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(xgb.pred.unlogged, yvalid)  #Score the model.
```

0.556433932153292

```
]: # Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE = 0.556433932153292, Model="xgboost1"))
```

```
]: # Print out the variale importance
xgb.importance(colnames(xtrain[, variablesSelected]), model.xgb)
colnames(xtrain.xgb)
```

Feature	Gain	Cover	Frequency
avg_score	0.46013526	0.51770550	0.34021305
hour	0.20192999	0.21578368	0.30559254
Handle_Count	0.16077300	0.11037577	0.11984021
month	0.11511404	0.10524461	0.17376831
Hashtag_Count	0.06204771	0.05089043	0.06058589

'hour' 'month' 'Handle_Count' 'Hashtag_Count' 'avg_score'

```
]: xtrain.xgb
```

	hour	month	Handle_Count	Hashtag_Count	avg_score
322	8	12	1	2	6.417143
88	19	8	1	4	6.608333
192	13	9	2	4	6.593333
21	8	7	2	0	6.372500
328	13	12	4	0	5.247273
73	19	8	1	4	6.608333
24	19	7	1	1	6.095000
191	16	9	2	4	6.593333
52	23	8	2	4	6.593333
149	1	9	2	4	6.593333
243	4	10	6	0	5.720000

150	19	9	2	4	6.593333

In [24]:
```
# Fit the xgboost model
model2.xgb <- xgboost(xtrain.xgb[,-4], log(ytrain+1), nround = 50, verbose = 0, nthred = workers)

# Generate predictions
xgb.pred <- predict(model2.xgb, xvalid.xgb[,-4])
xgb.pred.unlogged2 <- exp(xgb.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(xgb.pred.unlogged2, yvalid)  #Score the model.
```

0.566962307748351

In [25]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE = 0.566962307748351, Model="xgboost2"))
```

In [26]:
```
model.results
```

RMSLE	Model
0.5974126	linear_regression1
0.5869695	linear_regression_interaction
0.5564339	xgboost1
0.5669623	xgboost2

In [27]:
```
# Select variables.
# hour, month, handle_count, happines_score
variablesSelected <- c(10,11,15,17)
```

In [28]:
```
# Print out the variale importance
xgb.importance(colnames(xtrain[, variablesSelected]), model2.xgb)
colnames(xtrain.xgb)
```

Feature	Gain	Cover	Frequency
avg_score	0.5147359	0.58782161	0.3941411
hour	0.2000775	0.22754857	0.3069241
Handle_Count	0.1730727	0.08824770	0.1111851
month	0.1121139	0.09638212	0.1877497

'hour' 'month' 'Handle_Count' 'Hashtag_Count' 'avg_score'

In [29]:
```
# Fit the xgboost model
model3.xgb <- xgboost(xtrain.xgb[,-c(2,4)], log(ytrain+1), nround = 50, verbose = 0, nthred = workers)

# Generate predictions
xgb.pred <- predict(model3.xgb, xvalid.xgb[,-c(2,4)])
xgb.pred.unlogged3 <- exp(xgb.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(xgb.pred.unlogged3, yvalid)  #Score the model.
```

0.57012181459039

In [30]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE = 0.57012181459039, Model="xgboost3"))
```

In [31]:
```
#the error is higher so I will not cut other variables
```

4.3. Random forest

]:
```
# 4.3. Random forest
```

]:
```
# Select variables.
# hour, month, handle_count, hashtag_count, happines_score
variablesSelected <- c(10,11,15,16,17)
```

]:
```
dim(xtrain)
```

```
393  17
```

]:
```
model.rf <- foreach(ntreePerWorker = rep(ceiling(500/workers), workers), .combine=combine, .multicombine=TRUE, .inorder
  randomForest(xtrain[, variablesSelected], log(ytrain+1), ntree = ntreePerWorker, mtry = 3,  importance = TRUE,set.see
}

# Generate predictions
rf.pred <- predict(model.rf, xvalid[, variablesSelected])
rf.pred.unlogged <- exp(rf.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(rf.pred.unlogged, yvalid)  #Score the model.
```

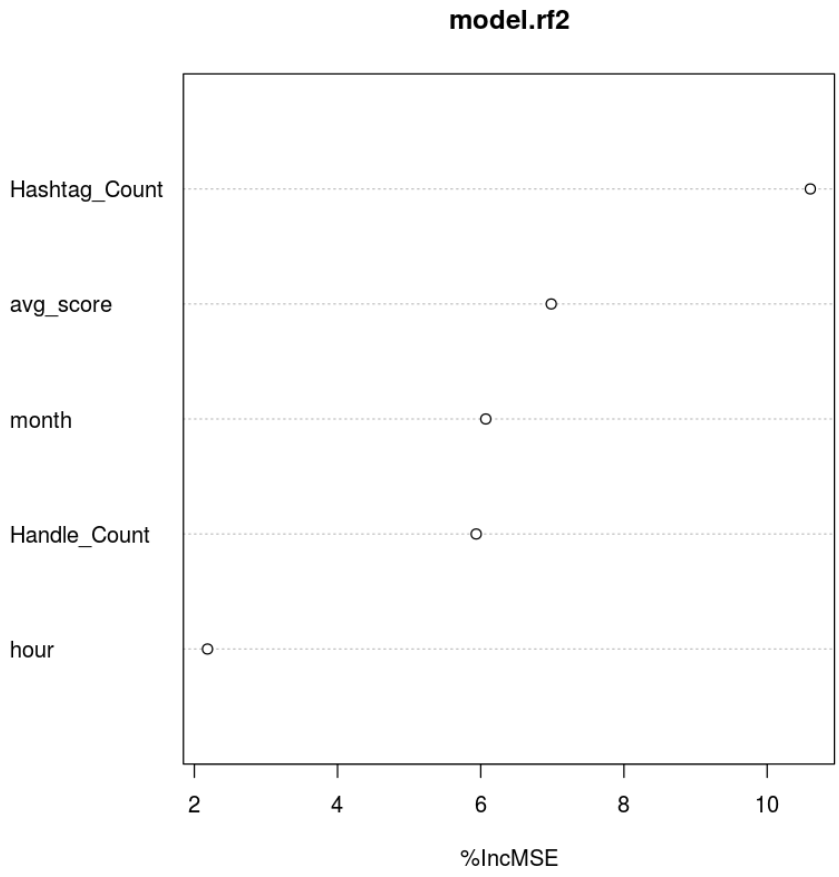
0.535697017933687

]:
```
model.rf2 <- foreach(ntreePerWorker = rep(ceiling(500/workers), workers), .combine=combine, .multicombine=TRUE, .inorde
  randomForest(xtrain[, variablesSelected], log(ytrain+1), ntree = ntreePerWorker, mtry = 2, importance = TRUE,set.seed
}

# Generate predictions
rf.pred <- predict(model.rf2, xvalid[, variablesSelected])
rf.pred.unlogged <- exp(rf.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(rf.pred.unlogged, yvalid)  #Score the model.
```

0.534204606275396

]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE = 0.535697017933687, Model="rf1"))
model.results <- rbind(model.results, data.frame( RMSLE = 0.534204606275396, Model="rf2"))
```

]:
```
#variable importance
varImpPlot(model.rf2, type = 1)
```



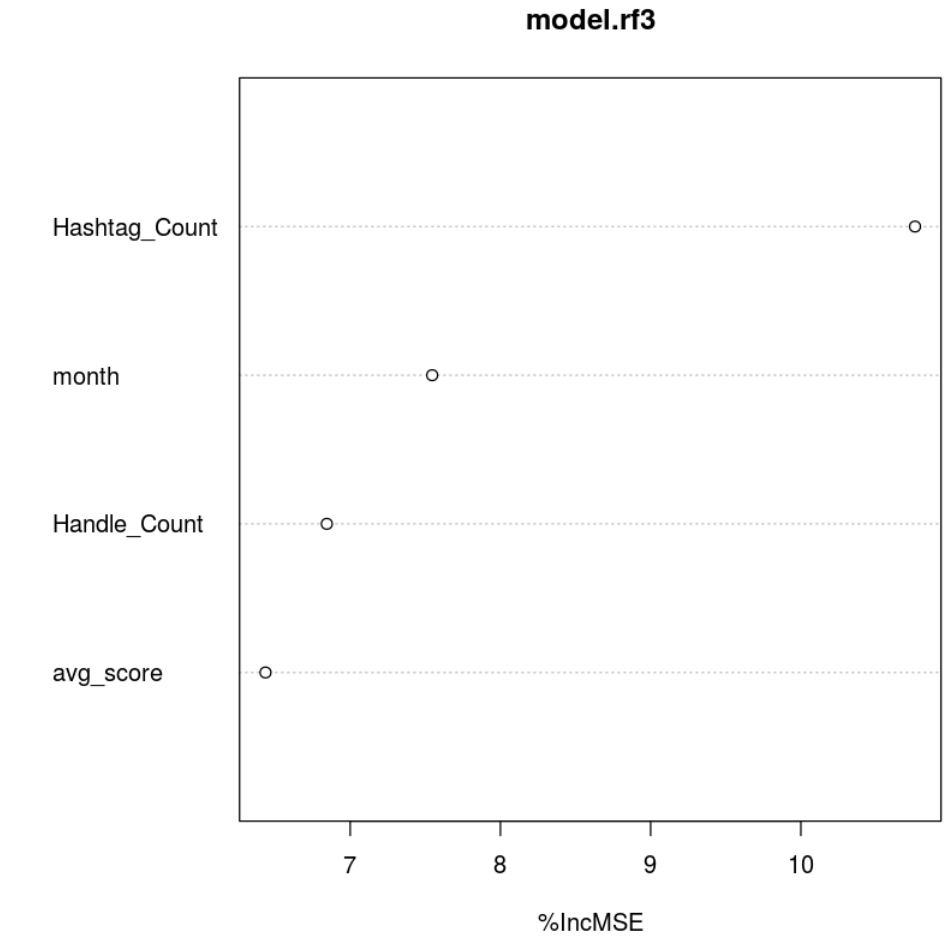
In [38]:
```
# Select variables.
#  month, handle_count, hashtag_count, happines_score
variablesSelected <- c(11,15,16,17)
```

In [39]:
```
model.rf3 <- foreach(ntreePerWorker = rep(ceiling(500/workers), workers), .combine=combine, .multicombine=TRUE, .inorde
  randomForest(xtrain[, variablesSelected], log(ytrain+1), ntree = ntreePerWorker, mtry = 2,  importance = TRUE,set.see
}

# Generate predictions
rf.pred <- predict(model.rf3, xvalid[, variablesSelected])
rf.pred.unlogged3 <- exp(rf.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(rf.pred.unlogged3, yvalid)  #Score the model.
```

```
0.564384590477743
```

In [40]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE =0.564384590477743 , Model="rf3"))
```

In [41]:
```
#variable importance
varImpPlot(model.rf3, type = 1)
```



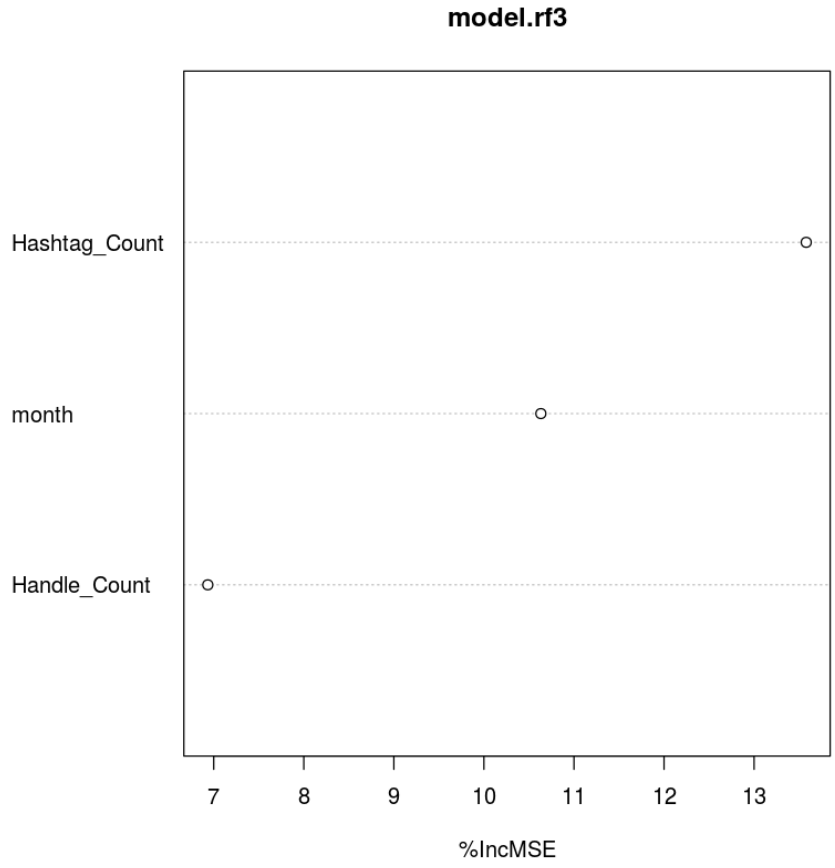

```
# Select variables.
#  month, handle_count, hashtag_count
variablesSelected <- c(11,15,16)
```

```
model.rf3 <- foreach(ntreePerWorker = rep(ceiling(500/workers), workers), .combine=combine, .multicombine=TRUE, .inorde
  randomForest(xtrain[, variablesSelected], log(ytrain+1), ntree = ntreePerWorker, mtry = 3,  importance = TRUE,set.see


# Generate predictions
rf.pred <- predict(model.rf3, xvalid[, variablesSelected])
rf.pred.unlogged3 <- exp(rf.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(rf.pred.unlogged3, yvalid)  #Score the model.
```

```
.544181843634976
```

```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE =0.544181843634976 , Model="rf3"))
```

```
#variable importance
varImpPlot(model.rf3, type = 1)
```



In [46]:
```
# Select variables.
#  month,  hashtag_count
variablesSelected <- c(11,16)
```

In [47]:
```
model.rf4 <- foreach(ntreePerWorker = rep(ceiling(500/workers), workers), .combine=combine, .multicombine=TRUE, .inorde
  randomForest(xtrain[, variablesSelected], log(ytrain+1), ntree = ntreePerWorker, mtry = 2,  importance = TRUE,set.see
}

# Generate predictions
rf.pred <- predict(model.rf4, xvalid[, variablesSelected])
rf.pred.unlogged4 <- exp(rf.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(rf.pred.unlogged4, yvalid)  #Score the model.
```

0.5630676447418

In [48]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE =0.5630676447418, Model="rf4"))
```

In [49]:
```
#the error is higher so I will not cut other variables
```

4.4. Regression tree

In [ ]:
```
# 4.4. Regression Tree
```

In [50]:
```
# Select variables.
# hour, month, handle_count, hashtag_count, happines_score
variablesSelected <- c(10,11,15,16,17)
```

In [51]:
```
# Fit the regression tree.
model.rpart <- rpart(log(ytrain+1) ~., data = xtrain[, variablesSelected], control = rpart.control(cp = 0.001))

# Generate predictions
rpart.pred <- predict(model.rpart, xvalid[, variablesSelected])
rpart.pred.unlogged <- exp(rpart.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(rpart.pred.unlogged, yvalid)  #Score the model.
```

0.622733749382179

In [52]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE =0.622733749382179, Model="rtree1"))
```

In [53]:
```
# Which attributes are most important?
model.rpart$variable.importance
```

avg_score	38.916577690616
Handle_Count	20.1823712297963
month	11.5874191314517
hour	11.2217771454299
Hashtag_Count	4.98769234740603

In [54]:
```
# Select variables.
# hour, month, handle_count, happines_score
variablesSelected <- c(10,11,15,17)
```

In [55]:
```
# Fit the regression tree.
model.rpart2 <- rpart(log(ytrain+1) ~., data = xtrain[, variablesSelected], control = rpart.control(cp = 0.001))

# Generate predictions
rpart.pred <- predict(model.rpart2, xvalid[, variablesSelected])
rpart.pred.unlogged2<- exp(rpart.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(rpart.pred.unlogged2, yvalid)  #Score the model.
```

0.622733749382179

In [56]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE =0.622733749382179, Model="rtree2"))
```

In [57]:
```
# Which attributes are most important?
model.rpart2$variable.importance
```

avg_score	38.916577690616
Handle_Count	20.1823712297963
month	11.5874191314517
hour	11.2217771454299

In [58]:
```
# Select variables.
# month, handle_count, happines_score
variablesSelected <- c(11,15,17)
```

In [59]:
```
# Fit the regression tree.
model.rpart3 <- rpart(log(ytrain+1) ~., data = xtrain[, variablesSelected], control = rpart.control(cp = 0.001))

# Generate predictions
rpart.pred <- predict(model.rpart3, xvalid[, variablesSelected])
rpart.pred.unlogged <- exp(rpart.pred) - 1  # Unlog the y variable.

# Score the predictions.
RMSLE(rpart.pred.unlogged, yvalid)  #Score the model.
```

0.631620557932791

```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE =0.631620557932791, Model="rtree3"))
```

```
#the error is higher so I will not cut other variables
```

```
model.results
```

RMSLE	Model
0.5974126	linear_regression1
0.5869695	linear_regression_interaction
0.5564339	xgboost1
0.5669623	xgboost2
0.5701218	xgboost3
0.5356970	rf1
0.5342046	rf2
0.5643846	rf3
0.5441818	rf3
0.5630676	rf4
0.6227337	rtree1
0.6227337	rtree2
0.6316206	rtree3

4.5.1. Simple Average Models

In [ ]:
```
# 4.5.1. Simple average models
```

In [63]:
```
# Ensemble model.
ensemble_pred_avg <- ( rf.pred.unlogged+xgb.pred.unlogged+lm.2.pred.unlogged+rpart.pred.unlogged2)/4
RMSLE(ensemble_pred_avg, valid.df$Score)
```

0.555010811744055

n [101]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE =0.555010811744055, Model="avg_1"))
```

In [64]:
```
# Ensemble model.
ensemble_pred_avg2 <- ( rf.pred.unlogged+xgb.pred.unlogged+lm.2.pred.unlogged)/3
RMSLE(ensemble_pred_avg2, valid.df$Score)
```

0.538590894737329

n [102]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE =0.538590894737329, Model="avg_2"))
```

In [65]:
```
# Ensemble model.
ensemble_pred_avg3 <- ( rf.pred.unlogged+xgb.pred.unlogged)/2
RMSLE(ensemble_pred_avg3, valid.df$Score)
```

0.537430513260946

n [103]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE =0.537430513260946, Model="avg_3"))
```

4.5.2. Weighted average models

In [ ]:
```
# 4.5.2. Weighted Average
```

In [88]:
```
m1_weight <- seq(0.1,0.9,0.1)
m2_weight <- seq(0.1,0.9,0.1)

# set up a matrix to store the RMSLE errors
RMSLE_matrix <- matrix(0,9,9)
for (i in 1:9) {
    for (j in 1:9){
        if (m2_weight[j]+ m1_weight[i] > 1) next
        ensemble_pred <- m1_weight[i]*rf.pred.unlogged +xgb.pred.unlogged*m2_weight[j] + lm.2.pred.unlogged*(1-
                                                                                    m1_weight[i] - m2_weight[i])
        RMSLE_matrix[i,j] <- RMSLE(ensemble_pred, valid.df$Score)
    }
}
```

```
Warning message in log(predictions + 1):
“NaNs produced”Warning message in log(predictions + 1):
“NaNs produced”Warning message in log(predictions + 1):
“NaNs produced”
```

In [89]:
```
RMSLE_matrix
```

0.5637825	0.5572505	0.5545717	0.5546927	0.5569155	0.5607490	0.5658334	0.5718968	0.5787302
0.5558011	0.5491001	0.5462316	0.5461753	0.5482454	0.5519540	0.5569406	0.5629313	0.0000000
0.5508474	0.5436868	0.5403838	0.5399393	0.5416724	0.5450942	0.5498401	0.0000000	0.0000000
0.5495987	0.5416530	0.5376345	0.5365543	0.5377310	0.5406701	0.0000000	0.0000000	0.0000000
0.5534752	0.5443029	0.5391811	0.5371199	0.5374305	0.0000000	0.0000000	0.0000000	0.0000000
0.5664573	0.5551486	0.5481305	0.5443909	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.6287228	0.6012818	0.5843025	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
NaN	NaN	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
NaN	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

In [90]:
```
ensemble_pred<-m1_weight[8]*rf.pred.unlogged +xgb.pred.unlogged*m2_weight[1] + lm.2.pred.unlogged*(1-m1_weight[8] - m2_
```

In [91]:
```
RMSLE_matrix[8,1] <- RMSLE(ensemble_pred, valid.df$Score)
```

In [92]:
```
RMSLE_matrix[8,1]
```

0.53373173279302

In [93]:
```
ensemble_pred<-m1_weight[8]*rf.pred.unlogged +xgb.pred.unlogged*m2_weight[2] + lm.2.pred.unlogged*(1-m1_weight[8] - m2_
```

In [94]:
```
RMSLE_matrix[8,2]<- RMSLE(ensemble_pred, valid.df$Score)
```

In [95]:
```
RMSLE_matrix[8,2]
```

0.533152443308139

In [96]:
```
ensemble_pred<-m1_weight[9]*rf.pred.unlogged +xgb.pred.unlogged*m2_weight[1] + lm.2.pred.unlogged*(1-m1_weight[9] - m2_
```

In [97]:
```
RMSLE_matrix[9,1]<- RMSLE(ensemble_pred, valid.df$Score)
```

In [98]:
```
RMSLE_matrix[9,1]
```

0.533213109538991

In [99]:
```
RMSLE_matrix
```

0.5637825	0.5572505	0.5545717	0.5546927	0.5569155	0.5607490	0.5658334	0.5718968	0.5787302
0.5558011	0.5491001	0.5462316	0.5461753	0.5482454	0.5519540	0.5569406	0.5629313	0.0000000
0.5508474	0.5436868	0.5403838	0.5399393	0.5416724	0.5450942	0.5498401	0.0000000	0.0000000
0.5495987	0.5416530	0.5376345	0.5365543	0.5377310	0.5406701	0.0000000	0.0000000	0.0000000
0.5534752	0.5443029	0.5391811	0.5371199	0.5374305	0.0000000	0.0000000	0.0000000	0.0000000
0.5664573	0.5551486	0.5481305	0.5443909	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.6287228	0.6012818	0.5843025	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.5337317	0.5331524	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

[100]:
```
# Find the element in the matrix corresponds to the lowest error?
which(RMSLE_matrix == min(RMSLE_matrix[RMSLE_matrix>0]), arr.ind = TRUE)
```

row	col
8	2

n [ ]:
```
# According to the table, the weights put on the predictions from the random forest,
# xgboost and linear regression should be 0.8,0.2,and 0.
# The lowest RMSLE erros acheived by the weighted average ensemble is 0.5331524.
```

[104]:
```
# Save the results for later ...

model.results <- rbind(model.results, data.frame( RMSLE =0.5331524, Model="weighted_avg"))
```

[105]:
```
model.results
```

RMSLE	Model
0.5974126	linear_regression1
0.5869695	linear_regression_interaction
0.5564339	xgboost1
0.5669623	xgboost2
0.5701218	xgboost3
0.5356970	rf1
0.5342046	rf2
0.5643846	rf3
0.5441818	rf3
0.5630676	rf4
0.6227337	rtree1
0.6227337	rtree2
0.6316206	rtree3
0.5550108	avg_1
0.5385909	avg_2
0.5374305	avg_3
0.5331524	weighted_avg

4.6. Best prediction model

0.5331524 | weighted_avg

In [ ]:
```
# 4.6. Best prediction model
```

In [85]:
```
#train the best model on the whole dataset
```

In [106]:
```
# Pull apart the Xs and Ys into separate frames.
xtrainf <- train.df[ , -c(4,5,6,7,19)]
ytrainf <- train.df[ , 19]
```

In [107]:
```
# Select variables.
# hour, month, handle_count, hashtag_count, happines_score
variablesSelected <- c(10,11,15,16,17)
```

In [108]:
```
# Create matrix for xgboost.
xtrain.xgb <- model.matrix(~ 0 + ., data = xtrainf[, variablesSelected])


# Fit the xgboost model
model.xgb <- xgboost(xtrain.xgb, log(ytrainf+1), nround = 50, verbose = 0, nthred = workers)  #nround=50 (0.55643393215
                                                                                                #=100(0.55771946680
                                                                                                #=75 (0.557078972676131)
                                                                                        #=40  (0.556896916974350)

# Fit the random forest

model.rf2 <- foreach(ntreePerWorker = rep(ceiling(500/workers), workers), .combine=combine, .multicombine=TRUE, .inorde
  randomForest(xtrainf[, variablesSelected], log(ytrainf+1), ntree = ntreePerWorker, mtry = 2, importance = TRUE,set.se
}
```

In [111]:
```
#xgboost1

xtrain.xgb <- model.matrix(~ 0 + ., data = xtest)

# Generate predictions
xgb.pred <- predict(model.xgb, xtrain.xgb)
xgb.pred.unlogged_test <- exp(xgb.pred) - 1  # Unlog the y variable.


#random forest 2
# Generate predictions
rf.pred <- predict(model.rf2, xtest)
rf.pred.unlogged_test <- exp(rf.pred) - 1  # Unlog the y variable.
```

In [112]:
```
ensemble_pred<-0.8*rf.pred.unlogged_test +xgb.pred.unlogged_test*0.2
```

In [113]:
```
ensemble_pred
```

1	2.55478907346168
2	5.46222663789033
3	1.47947038157358
4	1.37576411089721