Monash University

FIT5145 ASSIGNMENT 3 | DATA SCIENCE EXERCISE

Sunil Cyriac 29003164

Contents

1. Task A - Analyze the top 20 emoticons in messages of tweets msgraw_sample.txt file provided.

1.1 Task A.1: To create the tweet2emo.sh script to generate the potential_emoticon.csv file and get the top 20 emoticons and store it in emoticon.csv file

By the careful analysis of the given file using the simple file read commands using bash it was clear that the file consists of numerous emoticons which should be extracted from the file and processed later for our analysis. For this a bash script named tweet2emo.sh is created and following are the methods or functionalities used in the scripts to generate most of all the potential emoticons list with its count in the file. (For writing the bash command in this assignment I have used the Ubuntu's terminal)

1. There are 47 columns in the twitter DB file from which we need only the first column which possess the twitter texts data containing our emoticons. This is cut by using the command **cut -f1 < msgraw_sample.txt.**
2. Now since we are in have isolated the tweet texts data and being aware that some of the HTML escapes for symbols like '< 'and '>' may be lost which will lead to dropping of some emojis made with it all this are converted to original form using the perl command as: **perl -p -e 's/&gt;/>/g; s/&lt;/</g;'**
3. Then the space characters are converted to the newline characters, so we can get the possible emotions characters alone in each line using the command **perl -p -e 's/\s+/\n/g;'**
4. Then the data is sorted and stored in an intermediate file for verifying if required. The first 4 steps are executed by using the pipe "|" function between the commands.
5. By reading the data from the intermediate file some of the unwanted blank lines are cleared by using the grep command.
6. Then by careful analysis and assumptions using the **sed command sed 's/[ACEFGHIJLKMNQRSTUWYabcdefghijklmnrsqtuwy1245670]*//g'** the lines consisting of alphabets and numbers which possibly won't make up for an emoticons are filtered off.
7. Then by using the awk command **awk 'length($0)>1 && length($0)<8'** the lines consisting of length < 1 and greater than 8 are filtered off which possibly won't make an emoticon because of the limit. Then this is stored in another intermediate file for verification. Step 5 to 7 are joined by using the pipe function
8. By analysing the intermediate file it was clear that most of the emoticons starts with some specific characters which were segregated and this is used in grep command **grep -E '^[:;=^<>(_*xXD0O8vVD\-]'** so we can obtain the right emoticons which starts with those exact characters.
9. By using the **sed** command some of the empty lines are again filtered off
10. Then the data is sorted, and count of each emoticon is taken using the **uniq -c** command.
11. The count and the corresponding emoticons are tab delimited using the **perl** command.
12. Finally, the entire data is sorted in descending order of the count, so it will be easier for us to get the top 20 emoticons if needed to check the file manually. This data is stored to potential_emoticon.csv file

Method followed to get the top 20 emoticons:

1. Since the data stored in the potential emoticon file was filtered based on the possible characters that could make up an emoticon and ordered in descending order of the count it was very easy to get the top 20 emoticons just by a quick eye balling of the data by referring to a list of most widely used emoticons from different sources.
2. In my assumption to differentiate the emoticons and the top ones I have considered that the alphabets are case sensitives used in emoticons so for example ':p' and ':P' are considered different. This provided the below list of top 20 emoticons:

Count	Emoticon
5240	:)
3280	:D
1645	:p
1525	:(
1089	;)
1040	-_-
1023	<3
510	:*
530	(^o^)
434	(^^)
392	:3
389	:-)
375	:/
340	xD
338	*-*
337	:'(
326	D:
288	-.-
235	=)
234	(>_<)

1.2 Task A.2: Compute the co-occurrence of words with emoticons

Method followed for finding co-occurrence of words with emoticons:

1. A bash script named 'emowords.sh' is created where the given python file 'emoword.py' is being called for extracting the co-occurrence of words matching with the emoticon. The msgraw_sample.txt file acts as the STDIN file for this which we will be feeding to it through the bash script.
2. In the bash script a for loop is initialized where a variable E takes each of the 20 top emoticons as values passed along with the for loop by hardcoding in the for loop "in" list.
3. By using the **./emoword.py $E < msgraw_sample.txt** command the given python code is executed and the $E parameter will take each emoticons as values as the for loop executes for each emoticon and progress so on. This value is passed to the variable 'emo' in the emoword.py code as arguments.
4. In this step the data is filtered for the **stop words** and the word **RT** so that the list of words generated will be meaningful words. There will be some single character words, and this is kept as is since it is tweet message and users are likely to use short words such **"K" for "OK"** etc. The word **RT** is filtered because it is a common word which is used with almost all the text messages in twitter so that is avoided hence we can find more interesting and meaningful words. This filtering is done using **sed -E** command where all the stop words are passed to it in bash script hardcoded right there. This is done because my assumption is stop words are static and will never change.
5. Then again, another filter is applied to get only the data which is containing only alphabets which makes meaningful words using **grep** command.
6. Then the records are sorted, and count of each word repeated in the data is calculated using **uniq -c** command.
7. Finally, the data is sorted in descending order of the count, so we can find the most occurring words easily by looking through the data which is loaded to the 'emowordresult.txt' file which is a tab delimited file using **perl** command.

8. In the final file for differentiating the words co-occurring with respect to each of the emoticon I have included heading as '**running this emoticon $E'** where $E will be replaced by the emoticon for which the for loop is running and this will be echoed in the terminal, so the user can see that for which emoticon the script is currently running.

List of selected interesting words per emoticon

Emoticon	Co-occuring words									
:)	follow	love	good	Hppy	bhdy	nght	Mng	lovely	school	frends
:D	kk	hppy	bhdy	good	hhhhh	love	jk	eh	y	sure
:p	hh	kok	hhhh	twter	lol	kkk	look			
:(	n't	love	kok	hhh	lol	hope	know	need	nght	los
;)	que	no	follow	hh	love	good	kk	thk	thg	lke
-_-	k	kk	jug	sms	flse	klo	buk	Gue	specl	tggl
<3	love	k	Hppy	LOVE	lke	follow	good	plee	hope	bck!
:*	hh	love	yg	lg	ng	kk	deh	ctk	hppy	jug
(^o^)	LOVE	ww								
(^^)	ku	jug	bu	smle	ko					
:3	yg	que	hh	ctk	sy	gue	kk	school	Yuhuuu	Lzy
:-)	lol	h	k	d	good	thg	love	ekend	Hppy	Greece
:/	que	yg	no	ku	hh	tu	eu	nk		
xD	que	no	y	lo	hh					
-	que	de	pr	eu	no	com	bom	hoje		
:'(	ku	y	gue	lg	yg	gu	lo	ng	jug	
D:	de	que	no	pr	nt	wt	hot	new	love	
-.-	k	d	lg	ku	y	hh	yg	gue		
=)	d	de	que	Bom	pr	no	yg			
(>_<)	w	ok								

From the above table we can convey the following observations:

- The words like **love, happy, birthday, morning, night, lovely, ok, lol etc** are most widely used along with the rest of all the top emoticons.
- Most of the words are written in 1 or 2 characters as short cuts and looks meaningless but after a brief research it was found that most of all the words here means something and for this online urban dictionary was referred (Mennie, n.d.)
- Some of the meaning of the words are: **klos: close, yg: young girl, hhh: showing happiness, gue: guy, que: question, smle: smile etc**

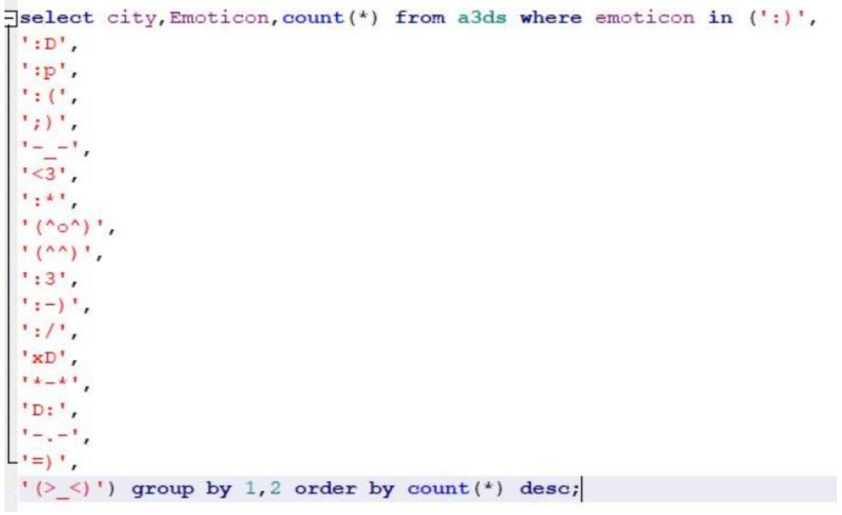
1.3 **Task A.3:** Interesting information about the top emoticons derived

The following process was carried out to get the interesting information about the emoticons and the tweet messages:

1. By using the given python file the command **./emodata.py < msgraw_sample.txt** was executed.
2. The generated text file was converted to .csv file for further processing and extracting of required information
3. The .csv file was named as **a3ds** and this was imported to the SQLite database and then the following interesting information was taken out:

1. Top City/Country and their count of using the respective emoticons

City/Country	Emoticon	Count(*)
	:)	1016
	:D	550
	(^o^)	357
	:(	298
	(^^)	246
	<3	229
	:p	207
	;)	187
	(>_<)	161
	_	136
	-_-	115
	:/	98
Indonesia	:D	79
	xD	78
Indonesia	:)	77
	:*	74
	:-)	71
	-.-	67
	=)	65
Philippines	:)	54
Indonesia	:p	49
	:3	42
indonesia	:D	37
	D:	30
Jakarta	:D	28
indonesia	:)	27
Singapore	:)	26
Indonesia	:(	25
Australia	:)	23

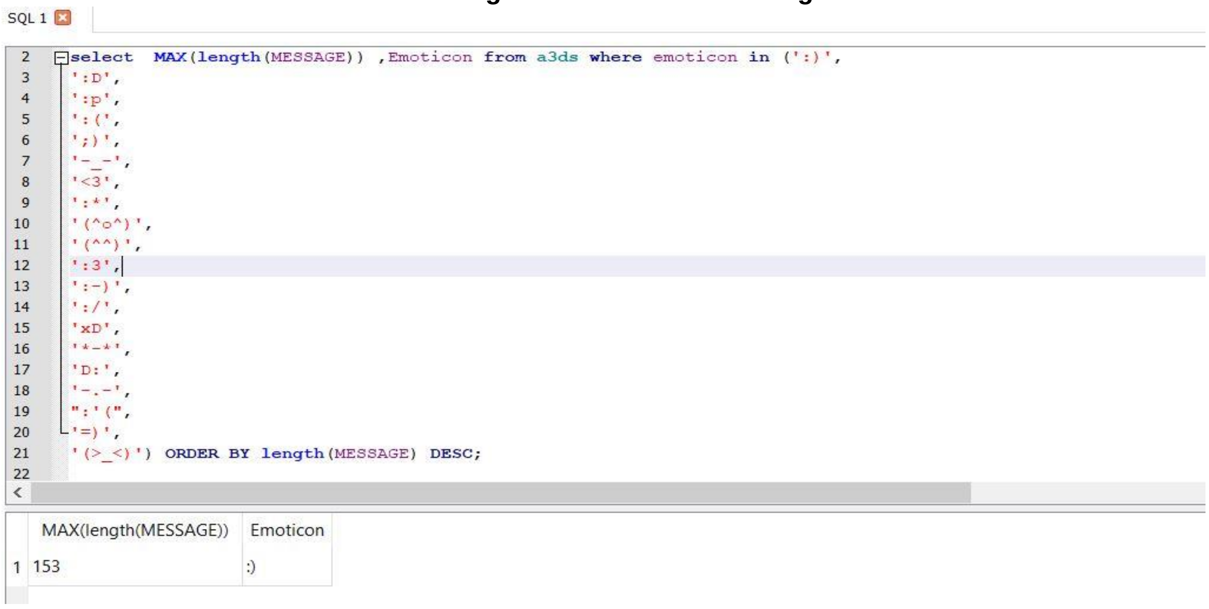


```
select city,Emoticon,count(*) from a3ds where emoticon in (':)',
':D',
':p',
':(',
';)',
'-_-',
'<3',
':*',
'(^o^)',
'(^^)',
':3',
':-)',
':/',
'xD',
'*-*',
'D:',
'-.-',
'=)',
'(>_<)') group by 1,2 order by count(*) desc;
```

Fig1: Top City/country count with respect to emoticons

- From the above table which we obtained from the query it is evident that some of the cities or countries data was not fed into the file but among the rest it is clear that most of the top emoticons are used by the Asian countries like Indonesia, Philippines, Singapore and most used emojis are ':D' and '😆'

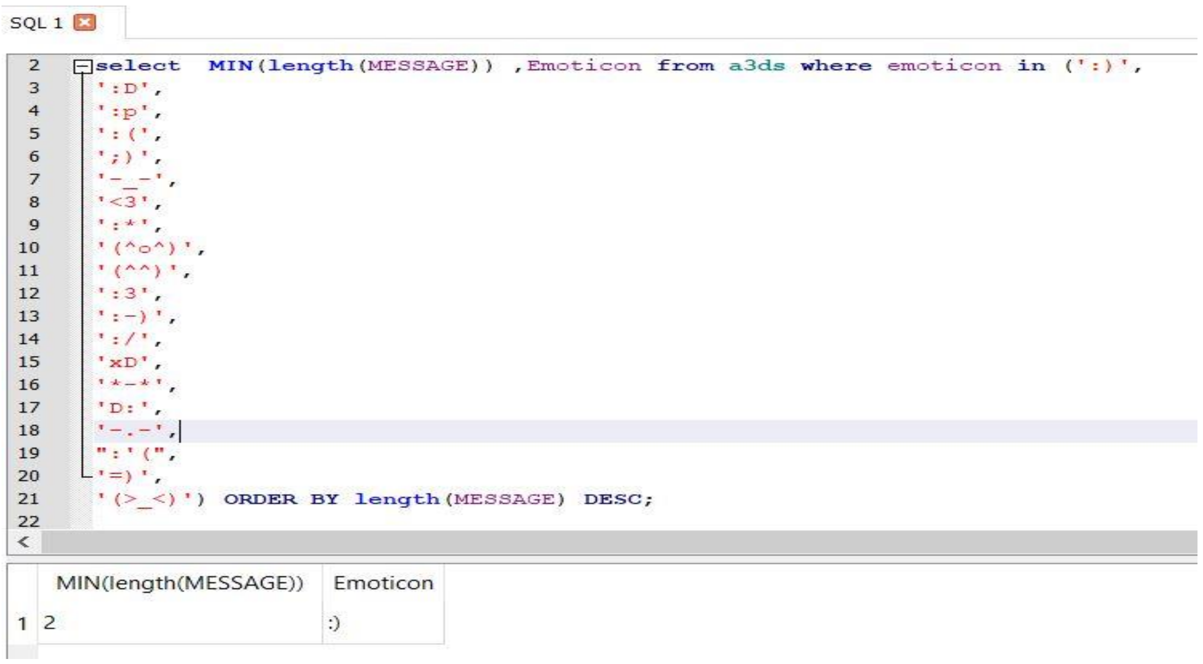
2. Emoticons with the longest and shortest message


SQL 1

```
select  MAX(length(MESSAGE)) ,Emoticon from a3ds where emoticon in (':)',
':D',
':p',
':(',
';)',
'-_-',
'<3',
':*',
'(^o^)',
'(^^)',
':3',
':-)',
':/',
'xD',
'*-*',
'D:',
'-.-',
":'(",
'=)',
'(>_<)') ORDER BY length(MESSAGE) DESC;
```

	MAX(length(MESSAGE))	Emoticon
1	153	:)

Fig2: Longest message emoticon


SQL 1

```
select  MIN(length(MESSAGE)) ,Emoticon from a3ds where emoticon in (':)',
':D',
':p',
':(',
';)',
'-_-',
'<3',
':*',
'(^o^)',
'(^^)',
':3',
':-)',
':/',
'xD',
'*-*',
'D:',
'-.-',
":'(",
'=)',
'(>_<)') ORDER BY length(MESSAGE) DESC;
```

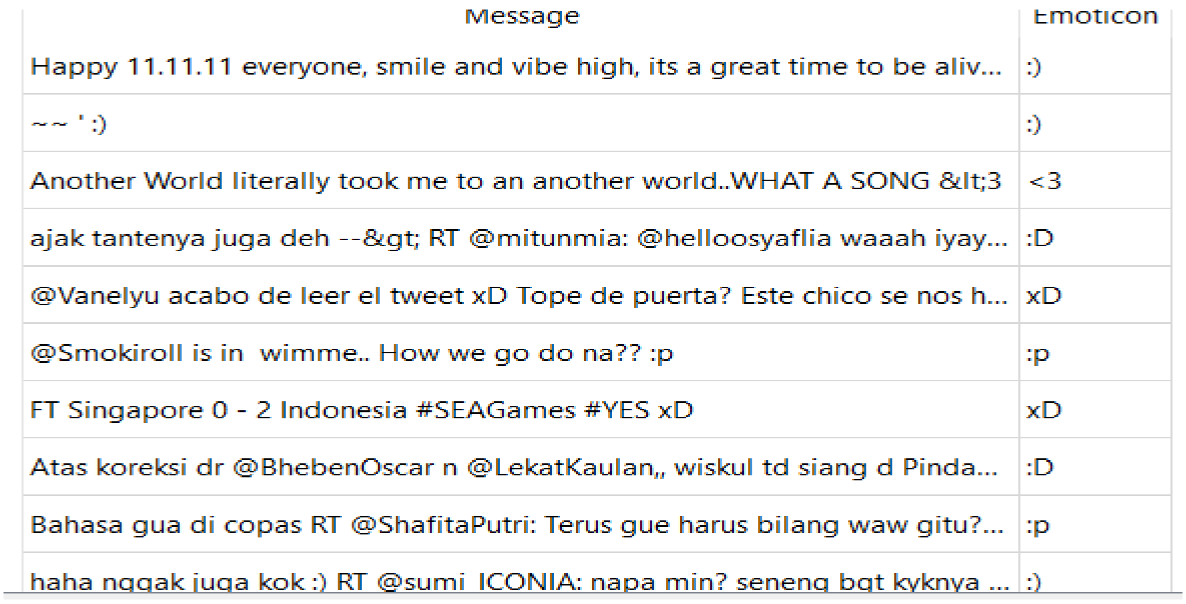
	MIN(length(MESSAGE))	Emoticon
1	2	:)

Fig3: Shortest message emoticon

- By analysing the results, it is quite interesting that the emoticon with the most count in the file is the same emoticon which is having the longest as well as shortest message

3. Message types attached to emoticons

By analysing the messages in the database, it became clearer that many of the messages are also in **different languages** and most of all the messages have a **happy tone** and some of the users are **retweeting** many tweets in the website.



Message	Emoticon
Happy 11.11.11 everyone, smile and vibe high, its a great time to be aliv...	:)
~~ ' :)	:)
Another World literally took me to an another world..WHAT A SONG <3	<3
ajak tantenya juga deh --> RT @mitunmia: @helloosyaflia waaah iyay...	:D
@Vanelyu acabo de leer el tweet xD Tope de puerta? Este chico se nos h...	xD
@Smokiroll is in wimme.. How we go do na?? :p	:p
FT Singapore 0 - 2 Indonesia #SEAGames #YES xD	xD
Atas koreksi dr @BhebenOscar n @LekatKaulan,, wiskul td siang d Pinda...	:D
Bahasa gua di copas RT @ShafitaPutri: Terus gue harus bilang waw gitu?...	:p
haha nggak juga kok :) RT @sumi_ICONIA: napa min? seneng bgt kyknya ...	:)

Fig4: Sample message types

2. Task B - Histogram Plots, Multiple R-squared values and MSE

2.1 Task B.1: To plot the histograms for X1, X2, X3 and X4 from train.csv

By observing the histogram plots for X1, X2, X3 and X4 it is very evident that the normally distributed data samples are:

1. Histogram with variable X1
2. Histogram with Variable X4

The reasons which supports the above answers are:

- The graphs for the X1 histogram and X4 histogram are bell shaped.
- Mean and median values of both X1 and X4 histograms are almost equal.
- When we observe the curve for the X1 and X4 histograms considering their variables we can see that they are symmetrical.

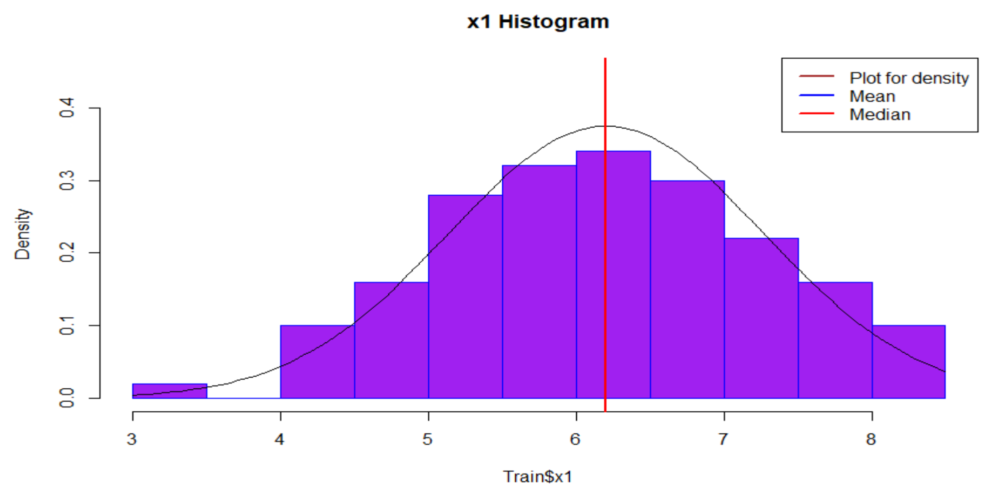


Fig1: X1 histogram

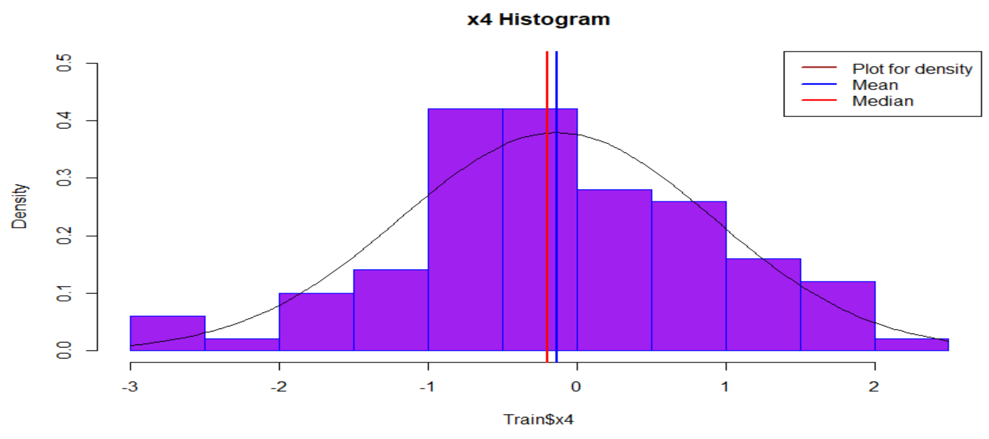
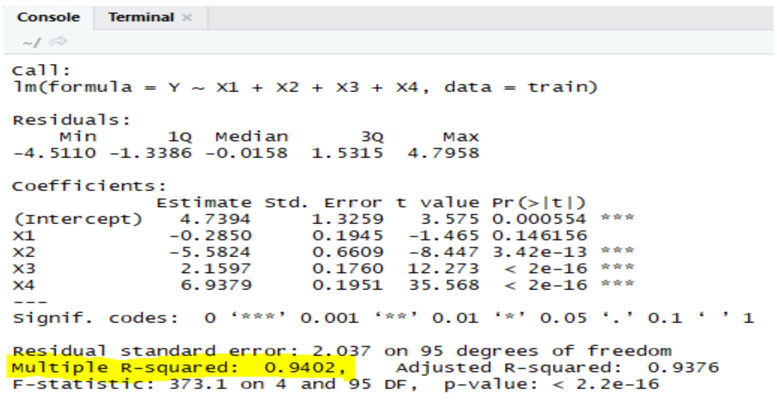


Fig2: X4 Histogram

2.2 Task B.2 : Fit the linear regression model using the file train.csv to find the higher multiple R squared value

Referring the 2 linear fit models model1 and model 2 developed using the R studio using the train.csv file we can see that the multiple R -squared value for **model1** is **higher** than **model2** and its value is **0.9402**. The below screenshots show the values obtained from the code in R studio which will give clear insights to the results obtained and supports the answer.



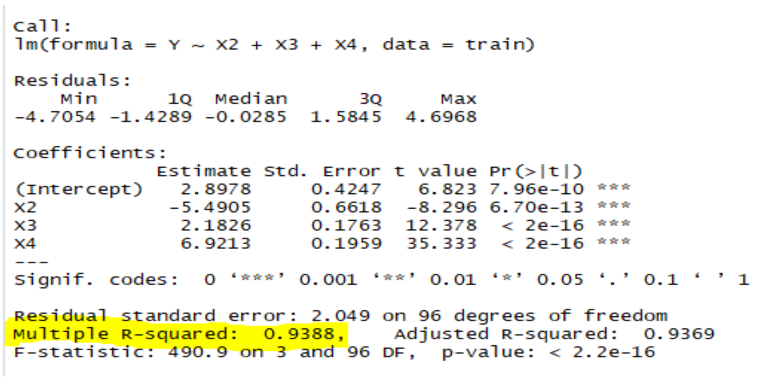
```
Call:
lm(formula = Y ~ X1 + X2 + X3 + X4, data = train)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.5110 -1.3386 -0.0158  1.5315  4.7958 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   4.7394     1.3259   3.575 0.000554 ***
X1           -0.2850     0.1945  -1.465 0.146156    
X2           -5.5824     0.6609  -8.447 3.42e-13 ***
X3            2.1597     0.1760  12.273  < 2e-16 ***
X4            6.9379     0.1951  35.568  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.037 on 95 degrees of freedom
Multiple R-squared:  0.9402,	Adjusted R-squared:  0.9376 
F-statistic: 373.1 on 4 and 95 DF,  p-value: < 2.2e-16
```

Fig3: Multiple R-squared value results for model1



```
Call:
lm(formula = Y ~ X2 + X3 + X4, data = train)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.7054 -1.4289 -0.0285  1.5845  4.6968 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   2.8978     0.4247   6.823 7.96e-10 ***
X2           -5.4905     0.6618  -8.296 6.70e-13 ***
X3            2.1826     0.1763  12.378  < 2e-16 ***
X4            6.9213     0.1959  35.333  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.049 on 96 degrees of freedom
Multiple R-squared:  0.9388,	Adjusted R-squared:  0.9369 
F-statistic: 490.9 on 3 and 96 DF,  p-value: < 2.2e-16
```

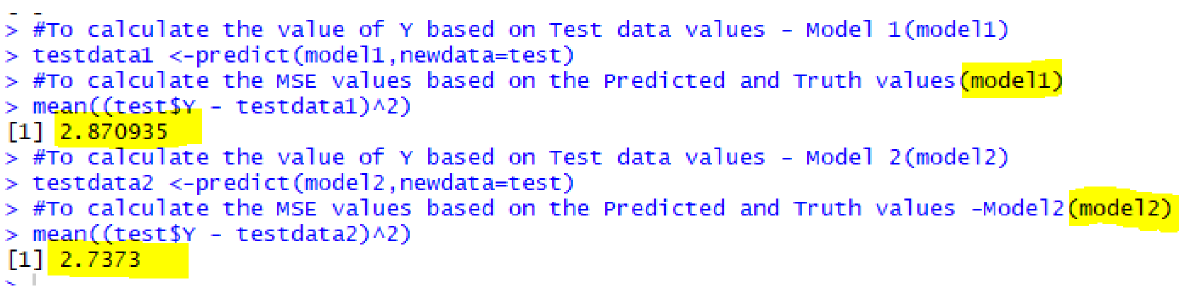
Fig4: Multiple R-squared value results for model2

2.3 Task B.3: Use the coefficients of Model 1 and 2 and predict values of Y using test.csv file, calculate MSE between the predictions and the true values to find which model is better

The following steps are carried out to find the answers for the task B.3

Step 1: Using the coefficients of model 1 and model 2 respectively the values for Y of test data are predicted.

Step 2: The mean squared errors for model 1 and model 2 are calculated in different instances using the predicted values from the step 1 and its truth values we have taken from the test data given.



```
> #To calculate the value of Y based on Test data values - Model 1(model1)
> testdata1 <-predict(model1,newdata=test)
> #To calculate the MSE values based on the Predicted and Truth values(model1)
> mean((test$Y - testdata1)^2)
[1] 2.870935
> #To calculate the value of Y based on Test data values - Model 2(model2)
> testdata2 <-predict(model2,newdata=test)
> #To calculate the MSE values based on the Predicted and Truth values -Model2(model2)
> mean((test$Y - testdata2)^2)
[1] 2.7373
```

Fig5: MSE of model 1 and model 2

The above figure shows the mean square values for model 1 and model 2 respectively. This can be obtained even by using the MSE functions from the R library but in this case, I have considered using the mean function and both functions provide the exact same results. The MSE for model 1 is **2.870935** which is higher than the MSE of model 2 at **2.7373.**

From the known fact that the more complex models have the higher R square value we can see that Model 1 which the complex model is have got the higher R square value.

But to answer the question if Model 1 is the better model than Model 2 we have to also consider the factors like adjusted R squared value and sometimes the predicted R squared values the reasons being:

1. R squared values sometimes can be inaccurate (minitab blog editor, 2016).
2. Models with too many variables and if there is a new data sometimes won't fit the model and this will lead to the usage of the predicted R squared values (minitab blog editor, 2016).
3. In some cases, the models with less MSE will be a better model which will make the model 2 a better model.

R studio Code

The .rmd file is also supplied with the report in the submission

```
#Task B1: Histogram plots for X1,X2,X3 and X4 respectively

#Histogram Plot for X1
hist(train$X1, main="x1 Histogram",xlab="Train$x1",border="blue", col="purple",ylim=c(0.00,0.45),prob=TRUE)
curve(dnorm(x, mean=mean(train$X1), sd=sd(train$X1)), add=TRUE)
abline(v = mean(train$X1),col = "blue",lwd = 2)
abline(v = median(train$X1),col = "red",lwd = 2)
legend(x = "topright", c("Plot for density", "Mean", "Median"), col = c("brown", "blue", "red"), lwd = c(2, 2, 2))

#Histogram Plot for X2
hist(train$X2, main="x2 Histogram",xlab="Train$x2",border="blue", col="purple",ylim=c(0.00,1.9),prob=TRUE)
curve(dnorm(x, mean=mean(train$X2), sd= sd(train$X2)), add=TRUE)
abline(v = mean(train$X2),col = "blue",lwd = 2)
abline(v = median(train$X2),col = "red",lwd = 2)
legend(x = "topright", c("Plot for density", "Mean", "Median"), col = c("brown", "blue", "red"), lwd = c(2, 2, 2))

#Histogram Plot for X3
hist(train$X3, main="x3 Histogram",xlab="Train$x3",border="blue", col="purple",ylim=c(0.00,0.5),prob=TRUE)
curve(dnorm(x, mean=mean(train$X3), sd=sd(train$X3)), add=TRUE)
abline(v = mean(train$X3),col = "blue",lwd = 2)
abline(v = median(train$X3),col = "red",lwd = 2)
legend(x = "topright", c("Plot for density", "Mean", "Median"), col = c("brown", "blue", "red"), lwd = c(2, 2, 2))

#Histogram Plot for X4
hist(train$X4, main="x4 Histogram",xlab="Train$x4",border="blue", col="purple",ylim=c(0.00,0.5),prob=TRUE)
curve(dnorm(x, mean=mean(train$X4), sd=sd(train$X4)), add=TRUE)
abline(v = mean(train$X4),col = "blue",lwd = 2)
abline(v = median(train$X4),col = "red",lwd = 2)

legend(x = "topright", c("Plot for density", "Mean", "Median"), col = c("brown", "blue", "red"), lwd = c(2,
2, 2))



#Task B.2 - Using train.csv to fit the Linear regression models


#Model1 - Linear Regression :


model1<-lm(formula = Y~X1+X2+X3+X4,data=train)

summary(model1)


#Model2 - Linear Regression:


model2<-lm(formula = Y~X2+X3+X4,data=train)

summary(model2)




#Task B.3- To predict the Y values of test.csv then find the mean squared value between predictions
and true value


#To calculate the value of Y based on Test data values - Model 1(model1)

testdata1 <-predict(model1,newdata=test)


#To calculate the MSE values based on the Predicted and Truth values(model1)

mean((test$Y - testdata1)^2)


#To calculate the value of Y based on Test data values - Model 2(model2)

testdata2 <-predict(model2,newdata=test)


#To calculate the MSE values based on the Predicted and Truth values -Model2(model2)

mean((test$Y - testdata2)^2)
```

References

Mennie, K. (n.d.). *Urban Dictionary*. Retrieved from urbandictionary.com: https://www.urbandictionary.com

minitab blog editor. (2016, february). *Five Reasons Why Your R-squared Can Be Too High*. Retrieved from http://blog.minitab.com: http://blog.minitab.com/blog/adventures-in-statistics-2/five-reasons-why-your-r-squared-can-be-too-high

Appendix

Software used: SQLite database for Task A.3

1. Took the tab delimited file generated from the task A.3 bash script and python file then it was converted to csv file
2. That was imported to the SQLite database as table named a3ds and the column names were modified as TimeZone, Emoticon, Message, Created_date, User and City
3. Queries used are attached as screenshot in the 1.3 section answers