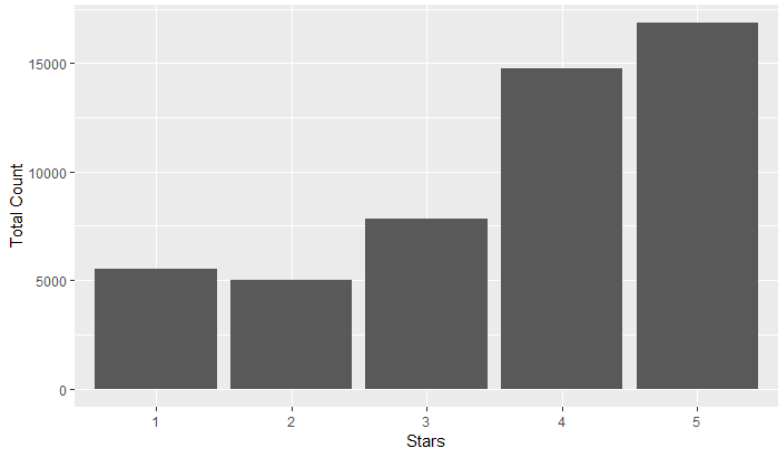
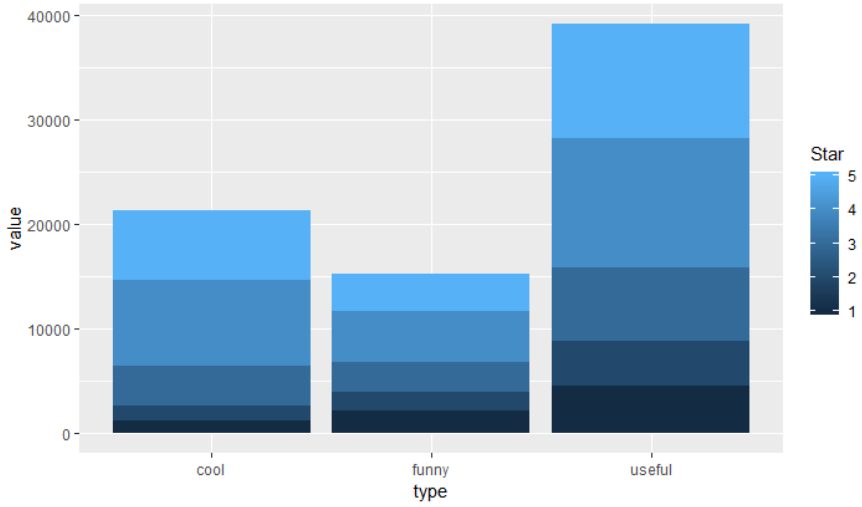
(a) Explore the data. How are star ratings distributed? How will you use the star ratings to obtain a label indicating 'positive' or 'negative' – explain using the data, graphs, etc.? Do star ratings have any relation to 'funny', 'cool', 'useful'? (Is this what you expected?)

The star ratings distribution in the data is shown in the graph below:



We observed that the average star rating observed 3.6, above which we will consider the ratings to be positive, while documents getting a star rating below it will be classed as negative.

'Funny', 'cool and 'useful' tags to the reviews could be applicable to both positive and negative reviews and as expected these tags are distributed across all star ratings. The slightly higher count of these labels for star 4 and star 5 ratings is a result of the data having more reviews of these ratings.



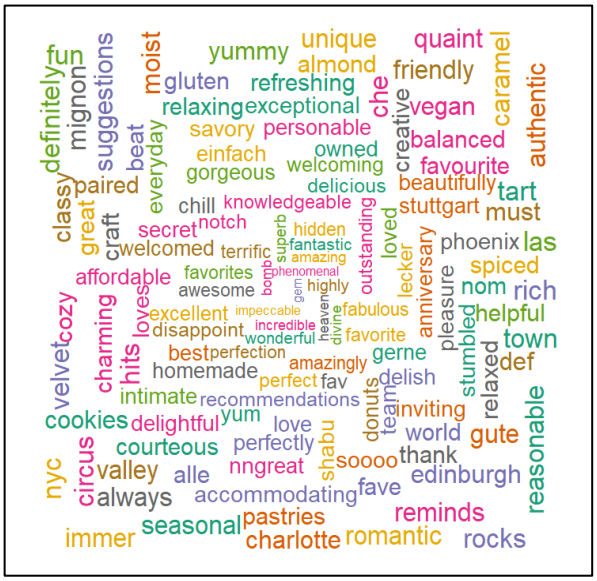
Question B

What are some words indicative of positive and negative sentiment? Do these 'positive' and 'negative' words make sense in the context of user reviews.

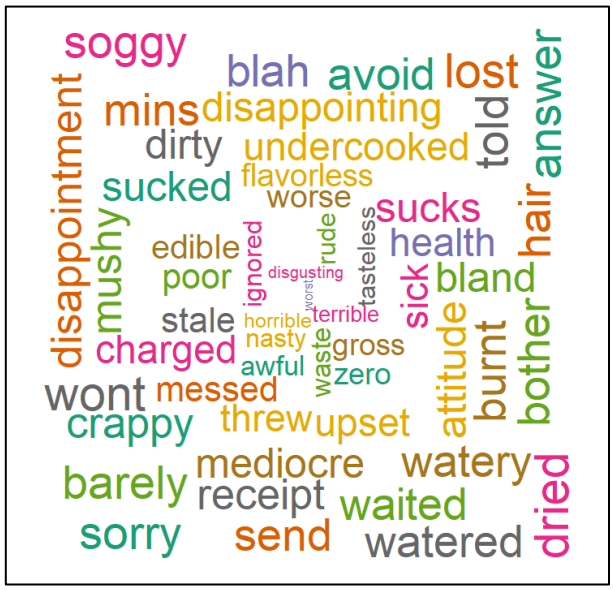
Methodology to clean the dataset–

- We cleaned the data by removing numbers, punctuation, symbols. We also converted all the words to lowercase and removed stop-words
- We created a Document Term matrix using the *term frequency* (# occurrence of word in the document/total no. of words in the document). We filtered for only those words that occur in at-least 0.5% of the documents and had min = 3 or max = 20 characters. After applying this filter, we obtained *~2300* words in the final document term matrix. (Essentially Document term matrix with 50k rows and 2300 columns).
- For each word in each document, we multiply the star rating of the document with the term frequency
- Take a total of the word star rating and average it by the total term frequency. The corresponding value gives the total positive/negative score of the word using the star rating.

Positive Words



Negative Words



Top positive words

gem	4.679253
phenomenal	4.673739
impeccable	4.648564
incredible	4.636565
heaven	4.624803
bomb	4.619899
divine	4.609689
amazing	4.594094
superb	4.580221
wonderful	4.555276
highly	4.554844
fantastic	4.541669
perfection	4.534500
hidden	4.512815
terrific	4.509699
favorites	4.507637
amazingly	4.500253
awesome	4.499664
fabulous	4.490146
outstanding	4.486104
perfect	4.473016
knowledgeable	4.465503

Top negative words

worst	1.380236
disgusting	1.467742
horrible	1.542541
terrible	1.551901
tasteless	1.642773
ignored	1.670572
rude	1.673134
nasty	1.674728
waste	1.682479
awful	1.712974
gross	1.771127
zero	1.777992
worse	1.833819
flavorless	1.853833
edible	1.885937
poor	1.908823
messed	1.920465
stale	1.921582
threw	2.030901
sick	2.055526
undercooked	2.065904
charged	2.080112

We can see that the positive and negative words and inline with our expectation with words like phenomenal, amazing, terrific occurring has the highest scores positive words while worst, disgusting, horrible as negative words

Question C

We will use consider three dictionaries – the Harvard IV dictionary of positive and negative terms, the extended sentiment lexicon developed by Prof Bing Liu of UIC-CS, and the AFINN dictionary which includes words commonly used in user-generated content in the web.

How many matching terms are there for each of the dictionaries? Consider using the dictionary based positive and negative terms to predict sentiment (positive or negative based on star rating) of a movie. Are you able to predict review sentiment based on these aggregated scores, and how do they perform? Does any dictionary perform better?

Dictionary	# of Positive words	# of Negative Words
Bing Liu	2006	4783
Harvard	1636	2006
Afinn Dictionary	878	1597

* *Afinn also has a word with score = 0 i.e. neutral*

# of Common Positive Words

Dictionary	Bing Liu	Harvard	Afinn Dictionary
Bing Liu	-	824	433
Harvard		-	349
Afinn Dictionary			-

# of Common Negative Words

Dictionary	Bing Liu	Harvard	Afinn Dictionary
Bing Liu	-	1556	865
Harvard		-	511
Afinn Dictionary			-

Sentiment Prediction using individual dictionary

General Methodology

1. Filter the initial document term matrix with all the words using the corresponding positive and negative dictionaries based on common words.
2. Calculate a cumulative positive score and negative score for each document based on presence the similar words from the dictionary by taking a sum of the term frequency calculated earlier
3. Predict the sentiment of the document as positive if the positive score > negative score. If not, flag it as negative
4. Obtain actual sentiment as positive if star rating >=4 and negative if star rating <=2. Flag neutral is star rating = 3
5. Compare the actual sentiment with predicted sentiment to compare performance of each dictionary

1. Bing Liu

We filter the initial document term matrix for the bing liu positive and negative dictionaries and obtained a resultant matrix with around ~1300 similar words for positive matrix and ~2500 words for negative dictionary.

Accuracy	70%			
	Actual			
Prediction	0	1	neutral	Precision
0	5977	2848	1623	57.21%
1	4568	28773	6211	72.75%
Recall	56.68%	90.99%		

2. Harvard Dictionary

Obtained a resultant matrix with around ~1200 similar words for positive matrix and ~1300 words for negative dictionary.

Accuracy	63.2%			
	TRUE			
pred	0	1	neutral	Precision
0	5021	5054	1931	41.82%
1	5524	26567	5903	69.92%
Recall	41.82%	84.02%		

3. Afin Dictionary

Obtained a resultant matrix with around ~1200 similar words for positive matrix and ~1300 words for negative dictionary.

Accuracy	67.8%			
	TRUE			
pred	0	1	neutral	Precision
0	4493	2220	1092	57.57%
1	6052	29401	6742	69.68%
Recall	57.57%	92.98%		

Compare performance of dictionaries:

We observe the ***bing liu*** dictionary performs the best with the highest accuracy of ~70% overall. It also has good performance for the positive class with both high recall and precision value while performs moderately on the negative class.

The ***afinn*** dictionary gives the next best results with similar performance on positive and negative class.

Question D

Develop models to predict review sentiment. For this, split the data randomly into training and test sets. To make run times manageable, you may take a smaller sample of reviews (minimum should be 10,000). One may seek a model built using only the terms matching any or all of the sentiment dictionaries, or by using a broader list of terms (the idea here being, maybe words other than only the dictionary terms can be useful). You should develop at least three different types of models (Naïve Bayes, and two others of your choice ….Lasso logistic regression (why Lasso?), knn, SVM, random forest,…?)

(i) Develop models using only the sentiment dictionary terms (you can try individual dictionaries or combine all dictionary terms). Do you use term frequency, tfidf, or other measures? What is the size of the document-term matrix?

(ii) Develop models using a broader list of terms – how do you obtain these terms? Will you use stemming here? Report on performance of the models. Compare performance with that in part (c) above.

For models in (i) and (ii): Do you use term frequency, tfidf, or other measures, and why? Do you prune terms, and how (also, why?). What is the size of the document-term matrix?

(i) Models using sentiment dictionary terms

We used the Bing Liu dictionary as well as a collation of all three dictionaries to build the models. We chose Bing Liu since among the individual dictionaries, it gave the best results w.r.t. accuracy, precision and recall

General Methodology to create dataset for modeling

1. We filtered the original document term matrix with all words with the Bing Liu/collated dictionary using the positive and negative words. For the process of model building, ***we used tfidf instead of tf*** since while building model we need the relative importance of the within as well across all the documents
2. We then subset this dataset only for those documents where we had actual star rating either >=4 (positive) or <=2 (negative). After filtering, we were left with *42166 documents* (remainder had star rating = 3)
3. We performed random sampling on document to obtain a ***training dataset with 10395 documents*** which was then used to build model. ***The rest was used as a test dataset***
   Below is the size of the train and test document term matrix

	Rows(# documents)	Columns(# of words)
Train	10395	3865 (1359 Positive words, 2506 negative words)
Test	31771	3865

Note: Here we did not prune the words based on the min/max document occurrence since we were using tfidf as the measure and the word count after filtering with dictionaries was considerable that we needed to build the model. We also did not stem the words since the dictionary consists of terms like accomplish/accomplishments and stemming would reduce the root word

Naïve Bayes Model

Naïve Bayes with Laplace smoothing factor = 1

Result for Bing Liu Dictionary

	Train	Test
Accuracy	80.60%	80.29%
Precision (Positive Class)	81.51%	81.00%
Precision (Negative Class)	73.94%	74.36%
Recall (Positive Class)	95.81%	96.32%
Recall (Negative Class)	35.36%	32.07%

Result for Combined Dictionary

	Train	Test
Accuracy	82.90%	81.70%
Precision (Positive Class)	82.78%	81.60%
Precision (Negative Class)	83.74%	82.65%
Recall (Positive Class)	97.40%	97.64%
Recall (Negative Class)	39.76%	33.77%

Random Forest

We ran the random Forest model using 200 trees. Below the results for Bing Liu Dictionary and Combined Dictionaries

Result for Bing Liu Dictionary

	Train	Test
Accuracy	94.96%	87.79%
Precision (Positive Class)	94.36%	88.01%
Precision (Negative Class)	97.16%	86.75%
Recall (Positive Class)	99.19%	96.94%
Recall (Negative Class)	82.38%	60.30%

Result for Combined Dictionary

	Train	Test
Accuracy	97.21%	87.93%
Precision (Positive Class)	96.68%	87.39%
Precision (Negative Class)	98.99%	90.78%
Recall (Positive Class)	99.69%	98.03%
Recall (Negative Class)	89.83%	57.84%

SVM

We used a linear Kernel and optimized the cost parameter using the tune function in R. We obtained the lowest cross validation error at C = 128 for the Bing Liu Dictionary and C = 64 for the combined dictionary.

```
> tune_cost=tune(svm ,train.x = model_data_train, train.y = as.factor(train_label), type = 'C-classification', kernel ="linear",
+                ranges =list(cost=c(0.125,    0.25,   0.5,    1,      2,      4,      8,      16,     32,     64,     128,    256,    512) ))
> summary(tune_cost)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  128

- best performance: 0.1083233

- Detailed performance results:
      cost     error  dispersion
1    0.125 0.2516590 0.011068172
2    0.250 0.2515628 0.010957872
3    0.500 0.2485807 0.011082770
4    1.000 0.2379022 0.013066877
5    2.000 0.2143332 0.012237141
6    4.000 0.1859536 0.013341563
7    8.000 0.1554579 0.012400427
8   16.000 0.1275619 0.008862276
9   32.000 0.1141919 0.008966580
10  64.000 0.1084193 0.007012394
11 128.000 0.1083233 0.007189870
12 256.000 0.1094774 0.005511786
13 512.000 0.1137098 0.005814654
```

** Code snippet for finding optimal cost parameter*

Result for Bing Liu Dictionary

	Train	Test
Accuracy	93.20%	88.94%
Precision (Positive Class)	93.11%	90.12%
Precision (Negative Class)	93.52%	84.29%
Recall (Positive Class)	98.17%	95.76%
Recall (Negative Class)	78.40%	68.42%

Result for Combined Dictionary

	Train	Test
Accuracy	94.07%	89.66%
Precision (Positive Class)	94.04%	90.75%
Precision (Negative Class)	94.21%	85.47%
Recall (Positive Class)	98.32%	96.01%
Recall (Negative Class)	81.46%	70.56%

For all three models, we observe that the combined dictionary performs better than Bing Liu dictionary since it has a more exhaustive list of words

(ii) Models using a broader list of terms:

The broader list of terms were obtained by lemmatizing the text using the 'textstem' package in R. After lemmatization, we had a total of 64,000 terms, processing which was beyond the capability of our machines, therefore we pruned our features to make it more manageable. We pruned the features further by selecting only those words which appeared in at-least 0.01% of the documents. We were left with a total of 5490 features. We are using tfidf again here as we wanted relative importance within as well as across the documents.

The performance of this set of features on the three previously run models are shown below:

Naïve Bayes:

Naïve Bayes with Laplace smoothing factor = 1

	Train	Test
Accuracy	80.73%	82.94%
Precision (Positive Class)	80.35%	82.92%
Precision (Negative Class)	81.89%	83.01%
Recall (Positive Class)	93.07%	94.39%
Recall (Negative Class)	57.94%	58.50%

Random Forest:

Total number of trees = 100

	Train	Test
Accuracy	99.93%	87.36%
Precision (Positive Class)	99.91%	86.11%
Precision (Negative Class)	100.00%	95.08%
Recall (Positive Class)	100.00%	99.08%
Recall (Negative Class)	99.73%	52.64%

SVM

We used the same procedure as mentioned for the previous SVM model where we obtained the optimum cost parameter of C =128.

	Train	Test
Accuracy	99.67%	89.99%
Precision (Positive Class)	99.71%	93.11%
Precision (Negative Class)	99.53%	80.58%
Recall (Positive Class)	99.85%	93.53%
Recall (Negative Class)	99.11%	79.48%

All our models above have far superior performance compared to what we noticed with part (c) where we used a simple method using the positive and negative dictionaries.