Vizualizations

Dr. Marko Mitic

Problem 1: ELECTION FORECASTING REVISITED

In earlier work, we used logistic regression on polling data in order to construct US presidential election predictions. We separated our data into a training set, containing data from 2004 and 2008 polls, and a test set, containing the data from 2012 polls. We then proceeded to develop a logistic regression model to forecast the 2012 US presidential election.

In this problem, we'll revisit our logistic regression model from Unit 3, and learn how to plot the output on a map of the United States. Unlike what we did in the Crime lecture, this time we'll be plotting predictions rather than data!

First, load the ggplot2, maps, and ggmap packages using the library function. All three packages should be installed on your computer, but if not, you may need to install them too using the install.packages function.

Then, load the US map and save it to the variable statesMap, like we did during the Crime lecture:

```
# install.packages("ggplot2")
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.2.1
```

```
# install.packages("maps")
library(maps)
```

```
## Warning: package 'maps' was built under R version 3.2.1
```

```
#install.packages("ggmap")
library(ggmap)

statesMap = map_data("state")
```

The maps package contains other built-in maps, including a US county map, a world map, and maps for France and Italy.

If you look at the structure of the statesMap data frame using the str function, you should see that there are 6 variables. One of the variables, group, defines the different shapes or polygons on the map. Sometimes a state may have multiple groups, for example, if it includes islands. We observe that there is 63 different group types:

```
str(statesMap)
```

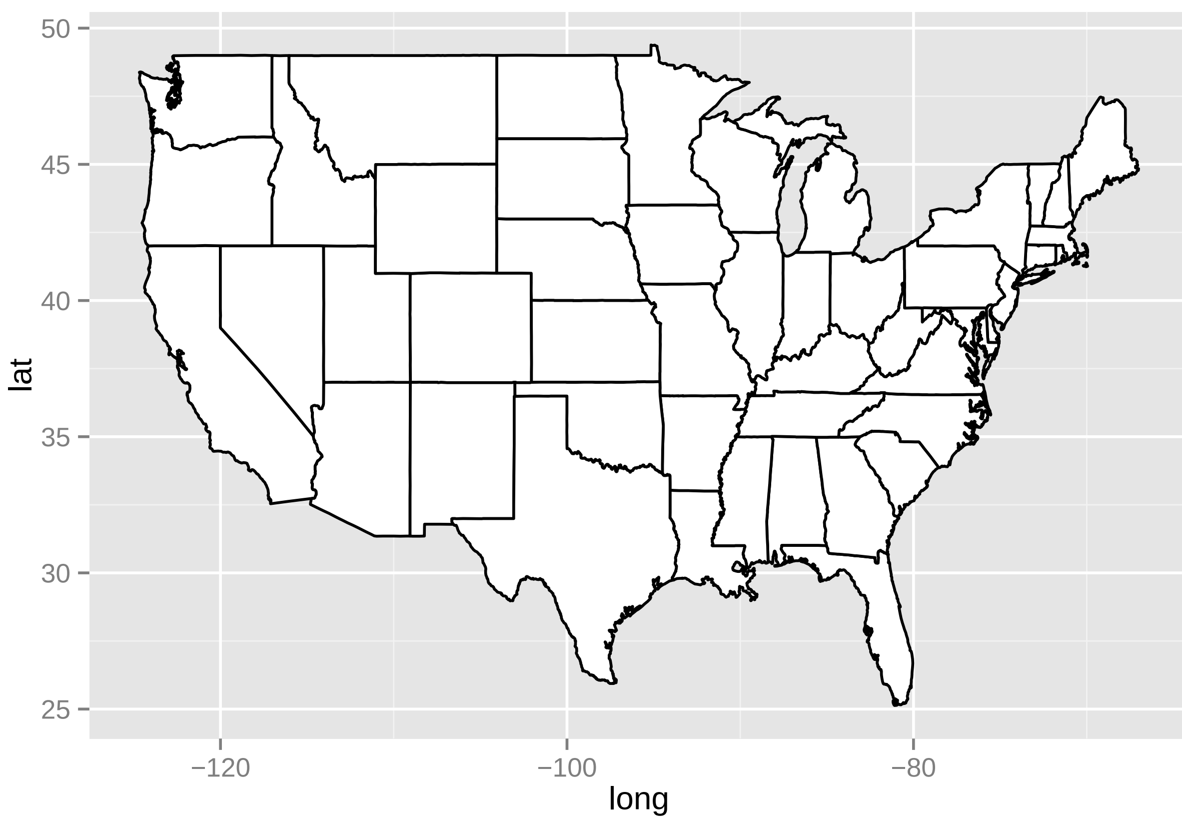
```
## 'data.frame':    15537 obs. of  6 variables:
##  $ long     : num  -87.5 -87.5 -87.5 -87.5 -87.6 ...
##  $ lat      : num  30.4 30.4 30.4 30.3 30.3 ...
##  $ group    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ order    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ region   : chr  "alabama" "alabama" "alabama" "alabama" ...
##  $ subregion: chr  NA NA NA NA ...
```

```r
table(statesMap$group)
```

```
##
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  202  149  312  516   79   91   94   10  872  381  233  329  257  256  113
##   16   17   18   19   20   21   22   23   24   25   26   27   28   29   30
##  397  650  399  566   36  220   30  460  370  373  382  315  238  208   70
##   31   32   33   34   35   36   37   38   39   40   41   42   43   44   45
##  125  205   78   16  290   21  168   37  733   12  105  238  284  236  172
##   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60
##   66  304  166  289 1088   59  129   96   15  623   17   17   19   44  448
##   61   62   63
##  373  388   68
```

We can now draw a map of the United States by typing the following in your R console:

```r
ggplot(statesMap, aes(x = long, y = lat, group = group)) + geom_polygon(fill = "white", color = "black")
```



ow, let's color the map of the US according to our 2012 US presidential election predictions from the Unit 3 Recitation. We'll rebuild the model here, using the dataset PollingImputed.csv (available in data folder).

```r
polling = read.csv("PollingImputed.csv")
```

Firstly w want to split the data using the subset function into a training set called "Train" that has observations from 2004 and 2008, and a testing set called "Test" that has observations from 2012.

```r
Train = subset(polling, polling$Year == 2004 | polling$Year == 2008)
Test = subset(polling, polling$Year == 2012)
```

Note that we only have 45 states in our testing set, since we are missing observations for Alaska, Delaware, Alabama, Wyoming, and Vermont, so these states will not appear colored in our map.

Next, we'll create a logistic regression model and make predictions on the test set using the following commands:

```r
mod2 = glm(Republican~SurveyUSA+DiffCount, data=Train, family="binomial")
TestPrediction = predict(mod2, newdata=Test, type="response")
```

`TestPrediction` gives the predicted probabilities for each state, but let's also create a vector of Republican/Democrat predictions by using the following command:

```r
TestPredictionBinary = as.numeric(TestPrediction > 0.5)
```

Now, put the predictions and state labels in a data.frame so that we can use ggplot:

```r
predictionDataFrame = data.frame(TestPrediction, TestPredictionBinary, Test$State)
```

Average predicted probability on test set is:

```r
mean(TestPrediction)
```

```
## [1] 0.4852626
```

Next, we need to merge "predictionDataFrame" with the map data "statesMap". Before doing so, we need to convert the Test.State variable to lowercase, so that it matches the region variable in statesMap:

```r
predictionDataFrame$region = tolower(predictionDataFrame$Test.State)
```

Merging is done as follows:

```r
predictionMap = merge(statesMap, predictionDataFrame, by = "region")
```

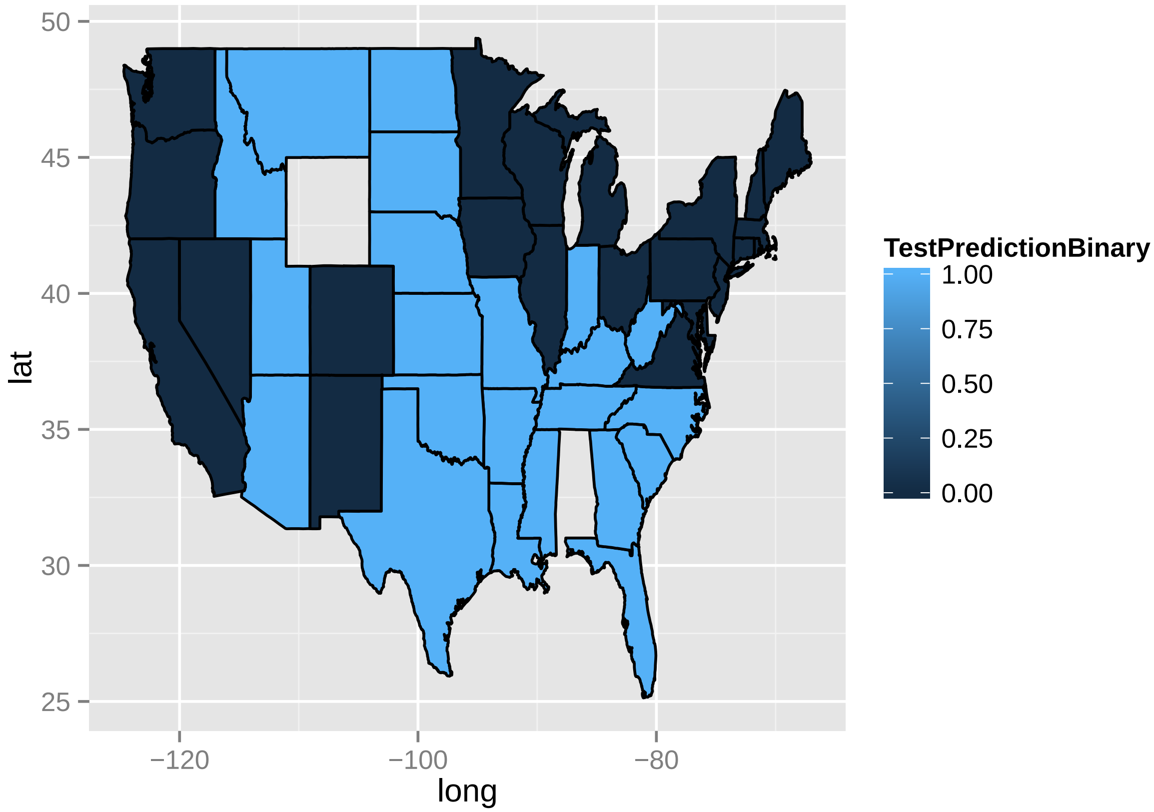
Lastly, we need to make sure the observations are in order so that the map is drawn properly, by typing the following:

```r
predictionMap = predictionMap[order(predictionMap$order),]
```

You an note that when we merge data, it only merged the observations that exist in both data sets. So since we are merging based on the region variable, we will lose all observations that have a value of "region" that doesn't exist in both data frames. You can change this default behavior by using the all.x and all.y arguments of the merge function. For more information, look at the help page for the merge function by typing ?merge in your R console.

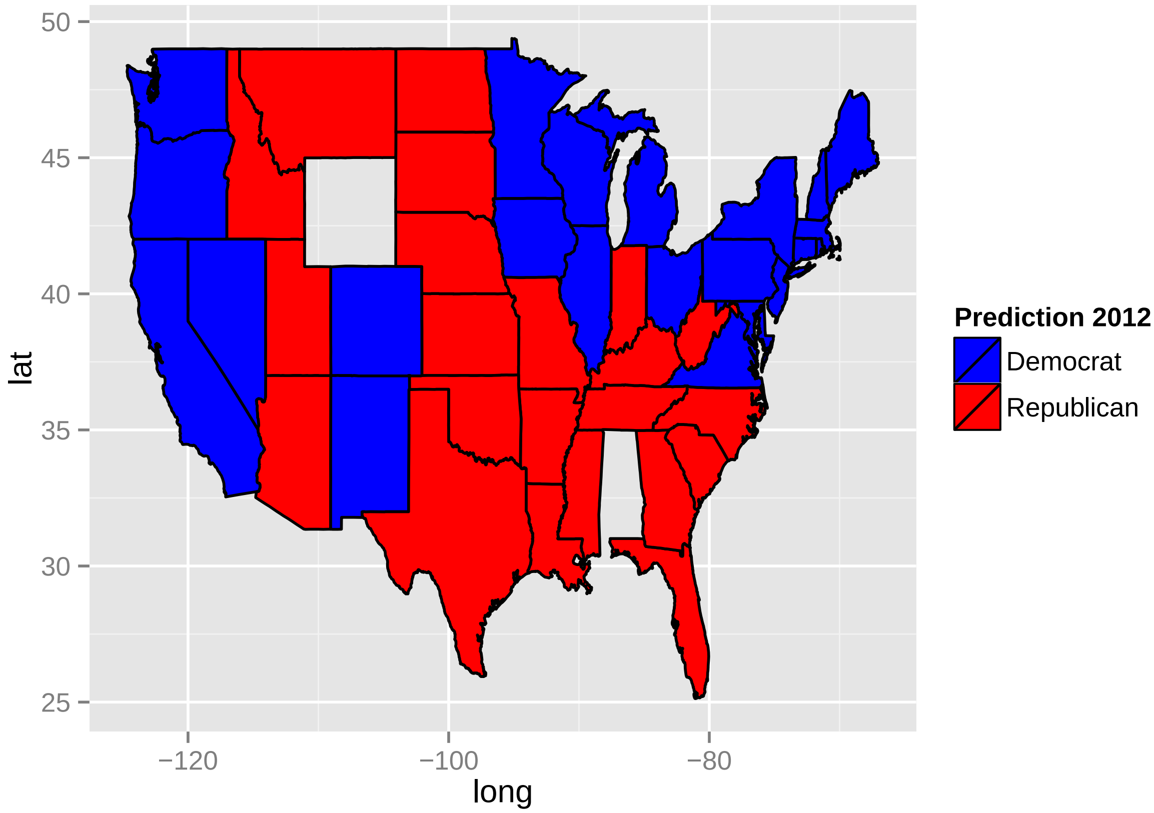
Now we are ready to color the US map with our predictions:

```
ggplot(predictionMap, aes(x = long, y = lat, group = group, fill = TestPredictionBinary)) + geom_polygon
```



We see that the legend displays a blue gradient for outcomes between 0 and 1. However, when plotting the binary predictions there are only two possible outcomes: 0 or 1. Let's replot the map with discrete outcomes. We can also change the color scheme to blue and red, to match the blue color associated with the Democratic Party in the US and the red color associated with the Republican Party in the US. This can be done with the following command:

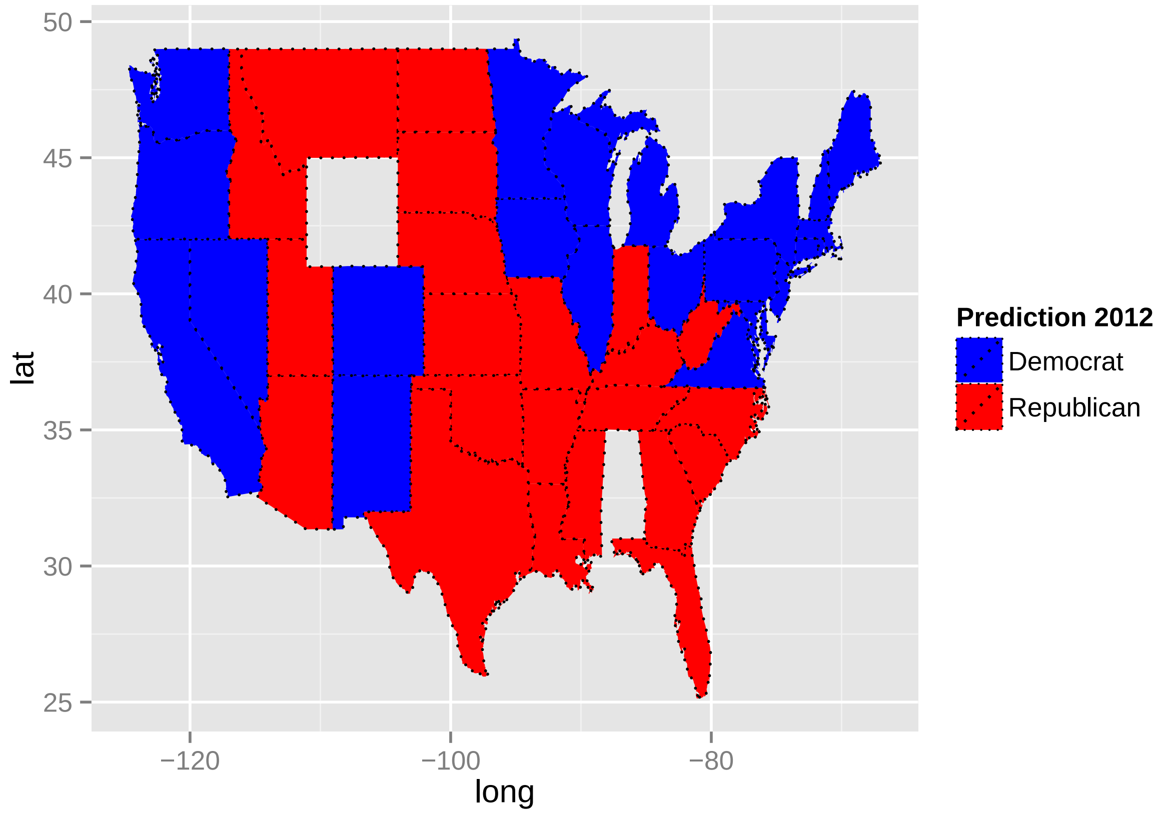
```
ggplot(predictionMap, aes(x = long, y = lat, group = group, fill = TestPredictionBinary))+ geom_polygon
```



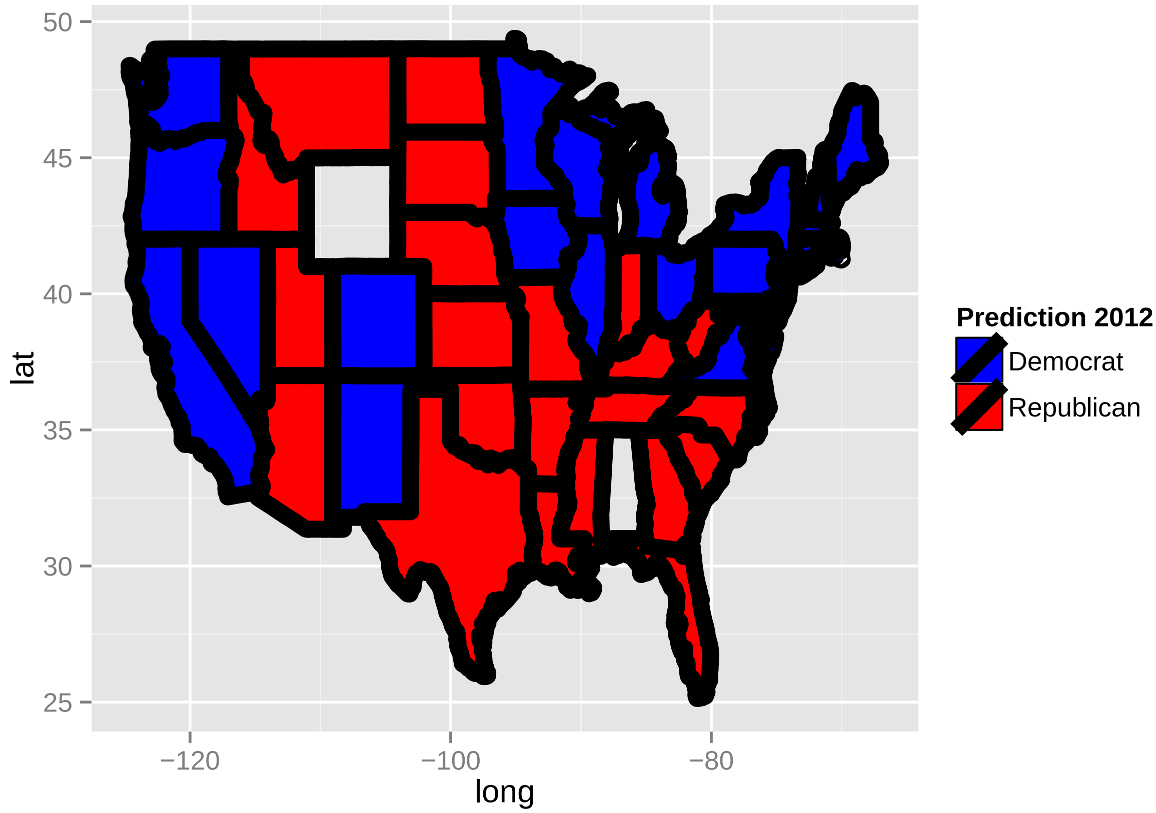
It is interestingly to observe the predicted and actual voting results in state of Florida. We predicted that Republican will won (0.9640395 prediction value), but was actually won by Democrat. We predicted Republican for the state of Florida with high probability, meaning that we were very confident in our incorrect prediction! Historically, Florida is usually a close race, but our model doesn't know this. The model only uses polling results for the particular year. For Florida in 2012, Survey USA predicted a tie, but other polls predicted Republican, so our model predicted Republican.

`ggplot` has manu options. Some of them we'll see in the next few figures. You can change linetype, size or fill type of the polygons:

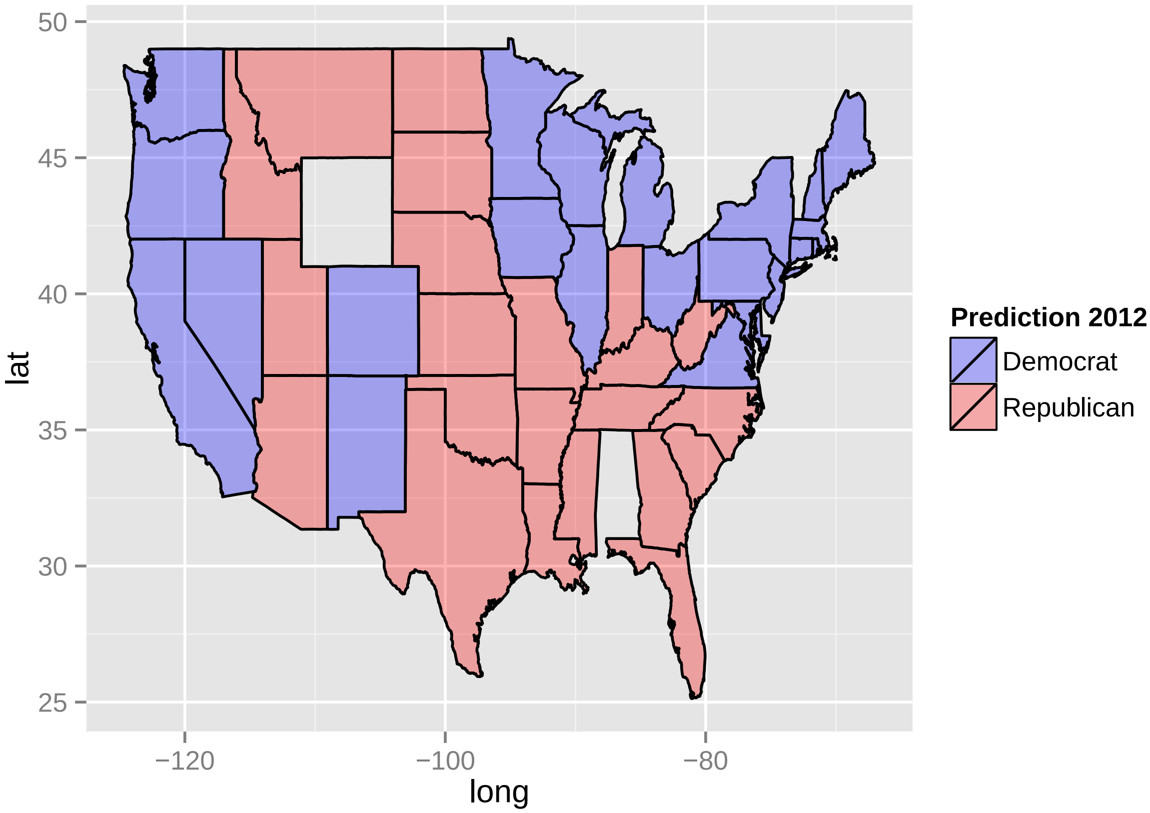
```
ggplot(predictionMap, aes(x = long, y = lat, group = group, fill = TestPredictionBinary))+ geom_polygon
```




```
ggplot(predictionMap, aes(x = long, y = lat, group = group, fill = TestPredictionBinary))+ geom_polygon
```




```
ggplot(predictionMap, aes(x = long, y = lat, group = group, fill = TestPredictionBinary))+ geom_polygon
```



More of this visualization options will be explored in next problems.

Problem 2: VISUALIZING NETWORK DATA

The cliche goes that the world is an increasingly interconnected place, and the connections between different entities are often best represented with a graph. Graphs are comprised of vertices (also often called "nodes") and edges connecting those nodes. In this assignment, we will learn how to visualize networks using the igraph package in R.

For this assignment, we will visualize social networking data using anonymized data from Facebook; this data was originally curated in a recent paper about computing social circles in social networks. In our visualizations, the vertices in our network will represent Facebook users and the edges will represent these users being Facebook friends with each other.

The first file we will use, edges.csv, contains variables V1 and V2, which label the endpoints of edges in our network. Each row represents a pair of users in our graph who are Facebook friends. For a pair of friends A and B, edges.csv will only contain a single row – the smaller identifier will be listed first in this row. From this row, we will know that A is friends with B and B is friends with A.

The second file, users.csv, contains information about the Facebook users, who are the vertices in our network. This file contains the following variables:

- **id**: A unique identifier for this user; this is the value that appears in the rows of edges.csv
- **gender**: An identifier for the gender of a user taking the values A and B. Because the data is anonymized, we don't know which value refers to males and which value refers to females.

- **school**: An identifier for the school the user attended taking the values A and AB (users with AB attended school A as well as another school B). Because the data is anonymized, we don't know the schools represented by A and B.
- **locale**: An identifier for the locale of the user taking the values A and B. Because the data is anonymized, we don't know which value refers to what locale.

First, we'll load the data from edges.csv into a data frame called edges, and load the data from users.csv into a data frame called users. The `str` command can give us elementary structures of datasets.

```
edges = read.csv("edges.csv")
str(edges)
```

```
## 'data.frame':    146 obs. of  2 variables:
##  $ V1: int  4019 4023 4023 4027 3988 3982 3994 3998 3993 3982 ...
##  $ V2: int  4026 4031 4030 4032 4021 3986 3998 3999 3995 4021 ...
```

```
users = read.csv("users.csv")
str(users)
```

```
## 'data.frame':    59 obs. of  4 variables:
##  $ id    : int  3981 3982 3983 3984 3985 3986 3987 3988 3989 3990 ...
##  $ gender: Factor w/ 3 levels "","A","B": 2 3 3 3 3 3 2 3 3 2 ...
##  $ school: Factor w/ 3 levels "","A","AB": 2 1 1 1 1 2 1 1 2 1 ...
##  $ locale: Factor w/ 3 levels "","A","B": 3 3 3 3 3 3 2 3 3 2 ...
```

We will be using the igraph package to visualize networks; install and load this package using the install.packages and library commands.

We can create a new graph object using the graph.data.frame() function. Please not4 that a directed graph is one where the edges only go one way – they point from one vertex to another. The other option is an undirected graph, which means that the relations between the vertices are symmetric.

```
#install.packages("igraph")
library(igraph)
```

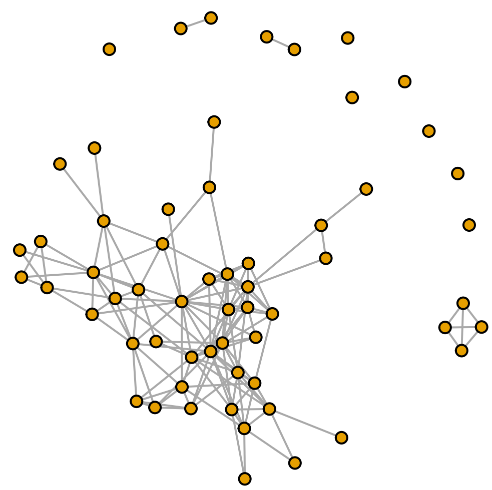
```
## Warning: package 'igraph' was built under R version 3.2.1
```

```
##
## Attaching package: 'igraph'
##
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
##
## The following object is masked from 'package:base':
##
##     union
```

```
g = graph.data.frame(edges, FALSE, users)
```

Now, we want to plot our graph. By default, the vertices are large and have text labels of a user's identifier. Because this would clutter the output, we will plot with no text labels and smaller vertices:

```r
plot(g, vertex.size=5, vertex.label=NA)
```



In this graph, there are a number of groups of nodes where all the nodes in each group are connected but the groups are disjoint from one another, forming "islands" in the graph. Such groups are called "connected components," or "components" for short. We observe that In addition to the large connected component, there is a 4-node component and two 2-node components. Also, there are 7 nodes that are not connected to any other nodes. Each forms a 1-node connected component.
In our graph, the "degree" of a node is its number of friends. We have already seen that some nodes in our graph have degree 0 (these are the nodes with no friends), while others have much higher degree. We can use `degree(g)` to compute the degree of all the nodes in our graph g.

```r
degree(g)
```

```
## 3981 3982 3983 3984 3985 3986 3987 3988 3989 3990 3991 3992 3993 3994 3995
##    7   13    1    0    5    8    1    6    5    3    2    2    5   10    8
##  594 3996 3997 3998 3999 4000 4001 4002 4003 4004 4005 4006 4007 4008 4009
##    3    3   10   13    3    8    1    6    4    9    2    1    3    0    9
## 4010 4011 4012 4013 4014 4015 4016 4017 4018 4019 4020 4021 4022 4023 4024
##    0    3    1    5   11    0    3    8    6    7    7   10    0   17    0
## 4025 4026 4027 4028 4029 4030 4031 4032 4033 4034 4035 4036 4037 4038
##    3    8    6    1    1   18   10    1    2    1    0    1    3    8
```

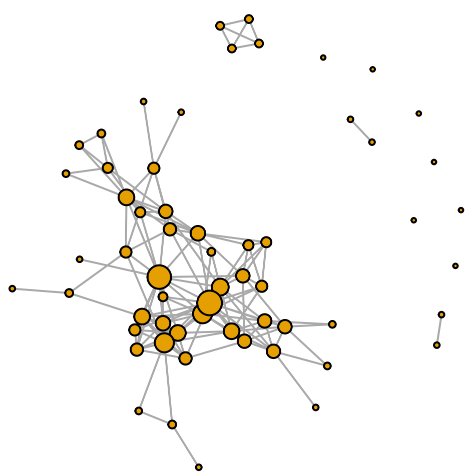
In a network, it's often visually useful to draw attention to "important" nodes in the network. While this might mean different things in different contexts, in a social network we might consider a user with a large number of friends to be an important user. From the previous problem, we know this is the same as saying that nodes with a high degree are important users.

To visually draw attention to these nodes, we will change the size of the vertices so the vertices with high degrees are larger. To do this, we will change the "size" attribute of the vertices of our graph to be an increasing function of their degrees:

```
V(g)$size = degree(g)/2+2
```

Now that we have specified the vertex size of each vertex, we will no longer use the vertex.size parameter when we plot our graph:

```
plot(g, vertex.label=NA)
```



we have changed the "size" attributes of our vertices. However, we can also change the colors of vertices to capture additional information about the Facebook users we are depicting.

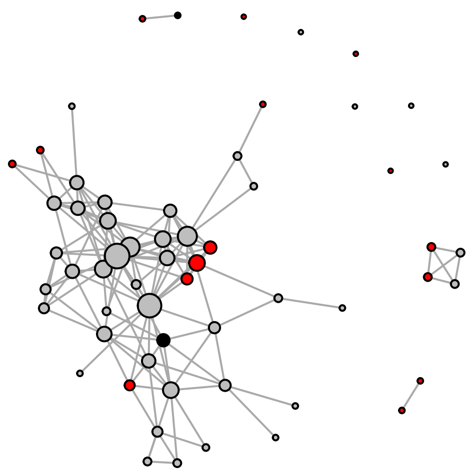
When changing the size of nodes, we first obtained the vertices of our graph with V(g) and then accessed the the size attribute with `V(g)$size`. To change the color, we will update the attribute V(g)$color.

To color the vertices based on the gender of the user, we will need access to that variable. When we created our graph g, we provided it with the data frame users, which had variables gender, school, and locale. These are now stored as attributes `V(g)$gender`, `V(g)$school`, and `V(g)$locale`.

We can update the colors by setting the color to black for all vertices, than setting it to red for the vertices with gender A and setting it to gray for the vertices with gender B:

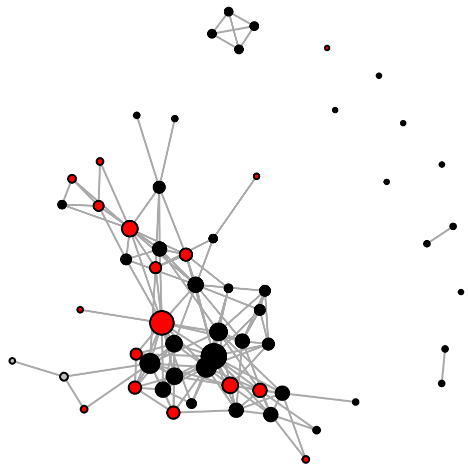
```
V(g)$color = "black"
V(g)$color[V(g)$gender == "A"] = "red"
V(g)$color[V(g)$gender == "B"] = "gray"
```

```r
plot(g, vertex.label=NA)
```



Similarly, we can color the vertices based on the school that each user in our network attended.

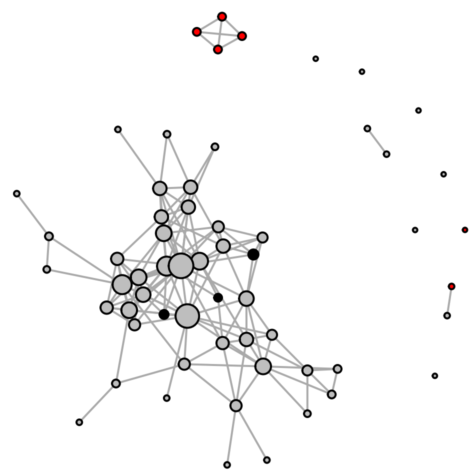
```r
V(g)$color = "black"
V(g)$color[V(g)$school == "A"] = "red"
V(g)$color[V(g)$school == "AB"] = "gray"

plot(g, vertex.label=NA)
```



The two students who attended schools A and B are colored gray; we can see from the graph that they are Facebook friends (aka they are connected by an edge). The high-degree users (depicted by the large nodes) are a mixture of red and black color, meaning some of these users attended school A and other did not.

Next, we can color the vertices based on the locale of the user.

```
V(g)$color = "black"
V(g)$color[V(g)$locale == "A"] = "red"
V(g)$color[V(g)$locale == "B"] = "gray"

plot(g, vertex.label=NA)
```



Nearly all of the vertices from the large connected component are colored gray, indicating users from Locale B. Meanwhile, all the vertices in the 4-user connected component are colored red, indicating users from Locale A.

Finally, few more things are important to note. The three functions to plot the igraph are plot.igraph (the function we used through the command "plot"), tkplot, and rglplot. rglplot makes 3-D plots – you can try one with rglplot(g, vertex.label=NA). Once you've made the plot, you can click and drag to rotate the graph. To use this function, you will need to install and load the "rgl" package.

To change the edge width, you need to change the edge parameter called "width". From ?igraph.plotting, we read that we need to append the prefix "edge." to the beginning for our call to plot, so the full parameter is called "edge.width". For instance, we could plot with edge width 2 with the command plot(g, edge.width=2, vertex.label=NA).

Problem 3: VISUALIZING TEXT DATA USING WORD CLOUDS

Earlier in the problems, we used text analytics as a predictive tool, using word frequencies as independent variables in our models. However, sometimes our goal is to understand commonly occurring topics in text data instead of to predict the value of some dependent variable. In such cases, word clouds can be a visually appealing way to display the most frequent words in a body of text.

A word cloud arranges the most common words in some text, using size to indicate the frequency of a word. For instance, this is a word cloud for the complete works of Shakespeare, removing English stopwords:

While we could generate word clouds using free generators available on the Internet, we will have more flexibility and control over the process if we do so in R. We will visualize the text of tweets about Apple, a dataset we used earlier in the course. As a reminder, this dataset (which can be downloaded from tweets.csv) has the following variables:

- **Tweet** – the text of the tweet
- **Avg** – the sentiment of the tweet, as assigned by users of Amazon Mechanical Turk. The score ranges on a scale from -2 to 2, where 2 means highly positive sentiment, -2 means highly negative sentiment, and 0 means neutral sentiment.

First we'll download the dataset "tweets.csv", and load it into a data frame called "tweets" using the read.csv() function, remembering to use stringsAsFactors=FALSE when loading the data.

```
tweets = read.csv("tweets.csv", stringsAsFactors = FALSE)
str(tweets)
```

```
## 'data.frame':    1181 obs. of  2 variables:
##  $ Tweet: chr  "I have to say, Apple has by far the best customer care service I have ever received!
##  $ Avg  : num  2 2 1.8 1.8 1.8 1.8 1.8 1.6 1.6 1.6 ...
```

Next, perform the following pre-processing tasks, noting that we don't stem the words in the document or remove sparse terms:

1) Create a corpus using the Tweet variable

```
#install.packages("tm")
library(tm)
```

```
## Warning: package 'tm' was built under R version 3.2.1

## Loading required package: NLP

## Warning: package 'NLP' was built under R version 3.2.1

##
## Attaching package: 'NLP'
##
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
corpus = Corpus(VectorSource(tweets$Tweet))
```

2) Convert the corpus to lowercase (don't forget to type "corpus = tm_map(corpus, PlainTextDocument)" in your R console right after this step)

```
corpus=tm_map(corpus, tolower)
corpus = tm_map(corpus, PlainTextDocument)
```

3) Remove punctuation from the corpus

```
corpus = tm_map(corpus, removePunctuation)
```

4) Remove all English-language stopwords

```
corpus = tm_map(corpus, removeWords, stopwords("english"))
```

5) Build a document-term matrix out of the corpus

```
dtm = DocumentTermMatrix(corpus)
dtm
```

```
## <<DocumentTermMatrix (documents: 1181, terms: 3780)>>
## Non-/sparse entries: 10273/4453907
## Sparsity           : 100%
## Maximal term length: 115
## Weighting          : term frequency (tf)
```

6) Convert the document-term matrix to a data frame called allTweets

```
allTweets = as.data.frame(as.matrix(dtm))
```

We can observe that there are 3780 unique words are there across all the documents. Note that we skipped `stem` process, because it will be easier to read and understand the word cloud if it includes full words instead of just the word stems.

Next, we need to install and load the "wordcloud" package, which is needed to build word clouds:

```
# install.packages("wordcloud")
library(wordcloud)
```

```
## Warning: package 'wordcloud' was built under R version 3.2.1
```

```
## Loading required package: RColorBrewer
```

```
## Warning: package 'RColorBrewer' was built under R version 3.2.1
```

We know that `colnames`, and `colSums` give us a vector of words and therfrequencies, respectivelt, so our cloud can be generated as follows:

```
wordcloud(colnames(allTweets), colSums(allTweets), scale=c(4, .5))
```

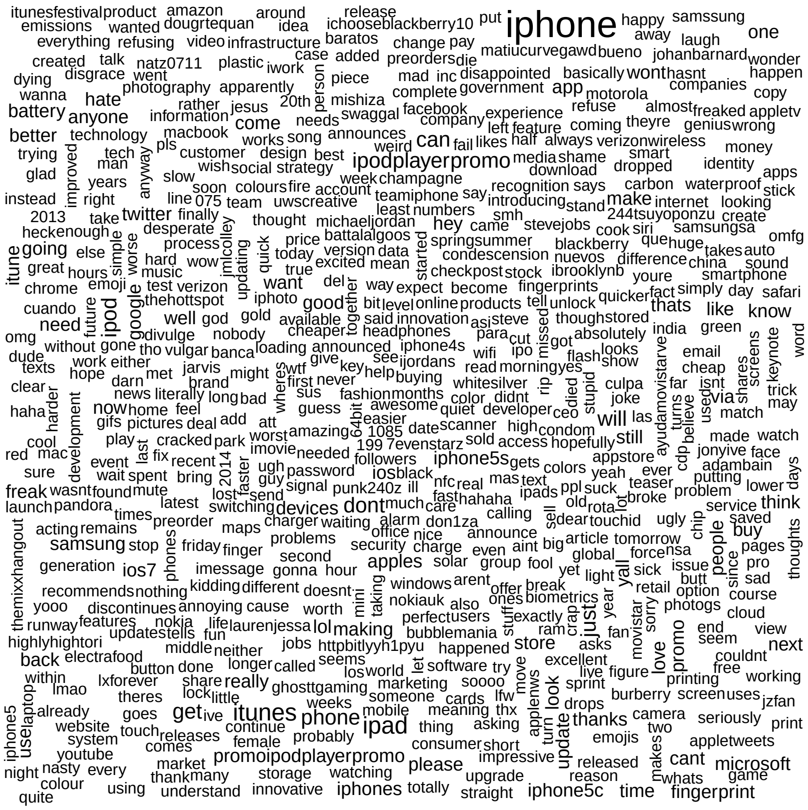
```
## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : important could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : windowsphone could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : apple could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : pretty could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : creativefutur could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : page could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : announcement could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : chargers could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : joconfino could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : things could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : support could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : ready could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : android could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : switch could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : actually could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : white could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : new could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : httpbitly18xc8dk could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : guys could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : thepartycow could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : freaking could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : business could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : another could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : something could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : hurt could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : wants could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : maybe could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : sagarkamesh could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : acciones could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : hello could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : getting could not be fit on page. It will not be plotted.

## Warning in wordcloud(colnames(allTweets), colSums(allTweets), scale =
## c(4, : follow could not be fit on page. It will not be plotted.
```

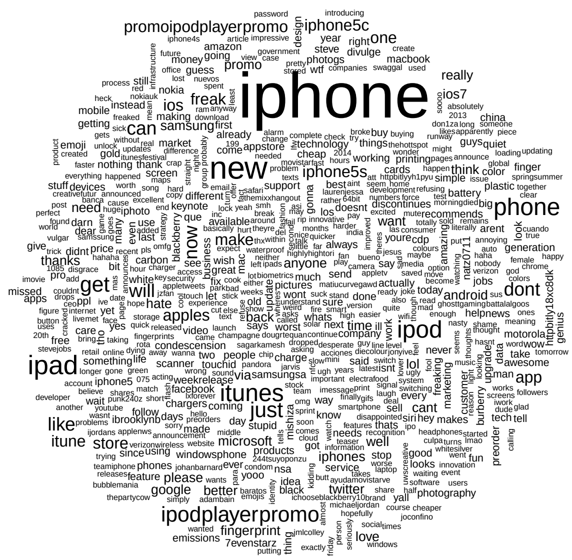


apple is by far most frequend terms in our dataset.In order to perform fair analysis, we'll eliminate this word from our dataset, repeating the whole process for generating allTweets data frame:

```
#install.packages("tm")
library(tm)

corpus = Corpus(VectorSource(tweets$Tweet))

corpus=tm_map(corpus, tolower)
corpus = tm_map(corpus, PlainTextDocument)

corpus = tm_map(corpus, removePunctuation)

corpus = tm_map(corpus, removeWords, c("apple", stopwords("english")))

dtm = DocumentTermMatrix(corpus)
dtm
```

```
## <<DocumentTermMatrix (documents: 1181, terms: 3779)>>
## Non-/sparse entries: 9121/4453878
## Sparsity           : 100%
## Maximal term length: 115
## Weighting          : term frequency (tf)
```

```r
allTweets = as.data.frame(as.matrix(dtm))

wordcloud(colnames(allTweets), colSums(allTweets), scale=c(2, .25))
```



Most frequent word in the new data frame is `iphone`. So far, the word clouds we've built have not been too visually appealing – they are crowded by having too many words displayed, and they don't take advantage of color. One important step to building visually appealing visualizations is to experiment with the parameters available, which in this case can be viewed by typing ?wordcloud in your R console.

We can now experiment with different options in `wordcloud` command. For example, the cloud with all negative tweets can be easily generated:

```r
negativeTweets = subset(allTweets, tweets$Avg <= -1)
wordcloud(colnames(negativeTweets), colSums(negativeTweets))
```

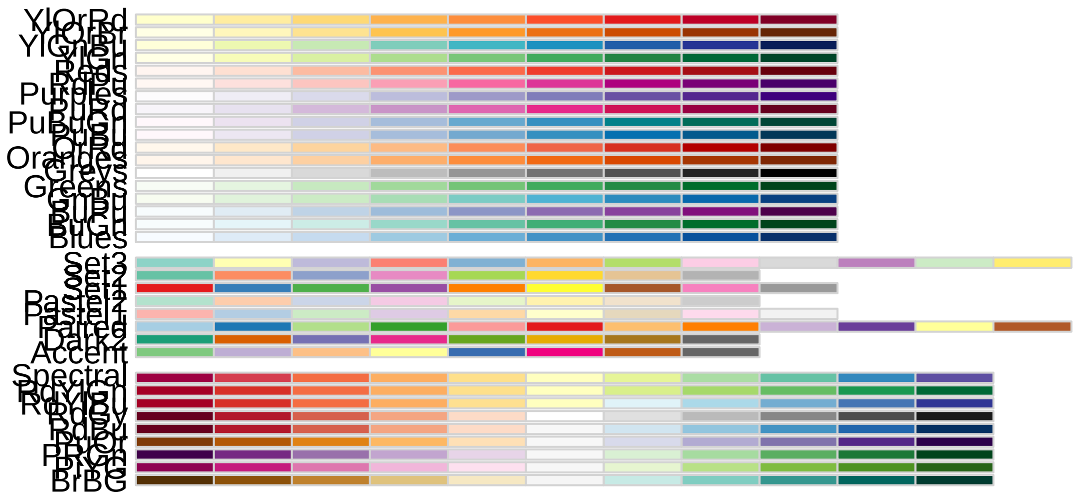
The use of a palette of colors can often improve the overall effect of a visualization. We can easily select our own colors when plotting; for instance, we could pass c("red", "green", "blue") as the colors parameter to wordcloud(). The RColorBrewer package, which is based on the ColorBrewer project (colorbrewer.org), provides pre-selected palettes that can lead to more visually appealing images. Though these palettes are designed specifically for coloring maps, we can also use them in our word clouds and other visualizations.

The "RColorBrewer" package may have already been installed and loaded when you installed and loaded the "wordcloud" package, in which case you don't need to go through this additional installation step. If you obtain errors (for instance, "Error: lazy-load database 'P' is corrupt") after installing and loading the RColorBrewer package and running some of the commands, try closing and re-opening R.

```
# install.packages("RColorBrewer")
library(RColorBrewer)
```

In this package two functions are importnant: the function brewer.pal() returns color palettes from the ColorBrewer project when provided with appropriate parameters, and the function display.brewer.all() displays the palettes we can choose from.

```
display.brewer.all()
```

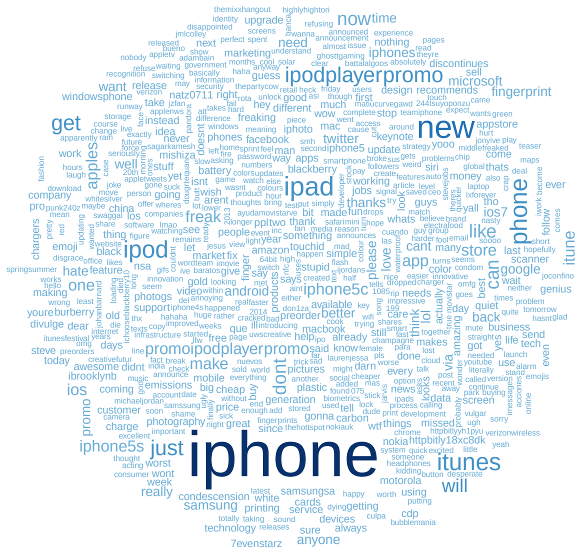


From ?brewer.pal we read that Accent and Set2 are both “qualitative palettes,” which means color changes don’t imply a change in magnitude (we can also see this in the output of display.brewer.all). As a result, the colors selected would not visually identify the least and most frequent words.

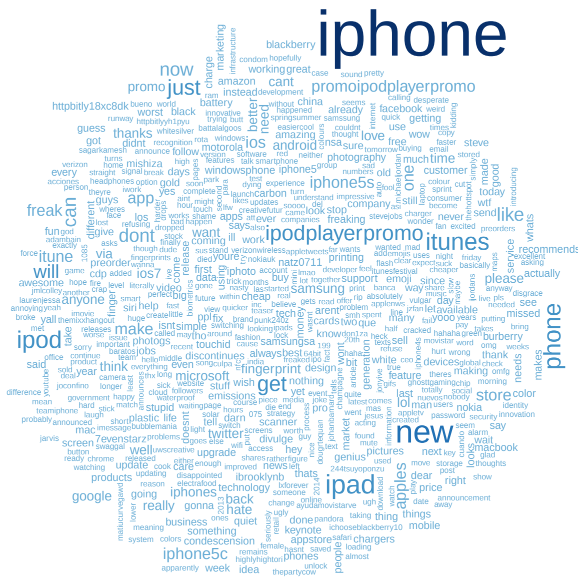
On the other hand, YlOrRd is a “sequential palette,” with earlier colors begin lighter and later colors being darker. Therefore, it is a good palette choice for indicating low-frequency vs. high-frequency words.

In sequential palettes, sometimes there is an undesirably large contrast between the lightest and darkest colors. You can see this effect when plotting a word cloud for allTweets with parameter colors=brewer.pal(9, “Blues”), which returns a sequential blue palette with 9 colors. We can change this by removing first four elements in the pallete:

```
wordcloud(colnames(allTweets), colSums(allTweets), scale=c(2, .25), colors = brewer.pal(9, "Blues")[c(-
```




```
## OR

wordcloud(colnames(allTweets), colSums(allTweets), scale=c(2, .25), colors = brewer.pal(9, "Blues")[c(5
```



That's it. Hope something from abovementioned is useful!