MSDS692 Data Science Practicum

Frederick Pletz

6/20/2021

# Assignment 4 - Magic Power Curve

Pulled data from <https://mtgjson.com/>

Data Cleaning done in Excel for speed: o Fixed incomplete data o Removed cards with no availability in paper format o Removed cards where hasContentWarning = 1 o Removed any card where isAlternative = 1 o Removed all cards where type = [Vanguard, Scheme, Conspiracy, Plane, Phenomenon, Character, Planeswalker, Land, Basic Land, Land Creature, Artifact Land, Summon] o Removed all cards where borderColor = [Gold, Silver] o Removed all cards referencing “playing for ante” o Where asciiName was not blank, replaced name with asciiName (removed non ascii characters) o Removed all cards where layout was not equal to “normal” o Added “X” for color and color identity to represent no-color (resolved blank values) o Added date for each printing o Removed spaces from multi-word keywords o Changed “,” to “ ” to avoid issue with CSV import o Modified type field to better represent slimple data set o Created field for colored mana cost o Columns Kept: colorIdentiy, convertedManaCost, coloredManaCost, power, toughness, rarity, types, uuid, text, releaseDate

Final csv included in project for download.

#import the data - relative path, so if using make sure to download FinalCreatureData.csv and run the R script from the same location.  
data <- read.csv('FinalCreatureData.csv', strip.white=TRUE, header=TRUE, quote="\"")  
  
#convert data into a dataframe  
df <- data.frame(data)  
  
#turn keyword abilities in the text field and rarity into dummy variables for use in modeling  
df.dummy <- dummy\_cols(df, select\_columns = c("rarity", "text"), split=' ')  
df.dummy <- subset(df.dummy, select=-c(rarity,text,types,uuid,releaseDate,colorIdentity))

#test/train split  
train\_index <- createDataPartition(df.dummy$convertedManaCost, p=0.7, list=FALSE)  
trainingData <- df.dummy[train\_index,]  
testData <- df.dummy[-train\_index,]  
  
#general linear model using gaussian family  
mtg.glm <- glm(convertedManaCost ~ ., data = trainingData, family = "gaussian")  
  
#show summary of model  
summary(mtg.glm)

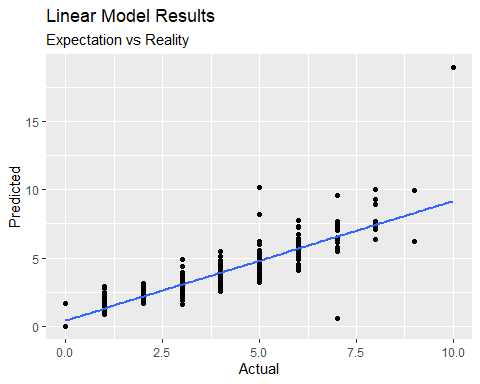
##   
## Call:  
## glm(formula = convertedManaCost ~ ., family = "gaussian", data = trainingData)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -5.1806 -0.3117 0.0056 0.3761 2.6314   
##   
## Coefficients: (7 not defined because of singularities)  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 0.021871 0.063489 0.344 0.730523   
## coloredManaCost -0.156458 0.017205 -9.094 < 2e-16 \*\*\*  
## power 0.642304 0.011830 54.294 < 2e-16 \*\*\*  
## toughness 0.476262 0.011231 42.405 < 2e-16 \*\*\*  
## rarity\_common 0.209174 0.040434 5.173 2.54e-07 \*\*\*  
## rarity\_mythic -1.363614 0.302729 -4.504 7.06e-06 \*\*\*  
## rarity\_rare -0.244643 0.063065 -3.879 0.000108 \*\*\*  
## rarity\_special -0.014027 0.227568 -0.062 0.950859   
## rarity\_uncommon NA NA NA NA   
## text\_afflict1 -0.311718 0.615901 -0.506 0.612833   
## text\_afterlife2 1.329413 0.436313 3.047 0.002344 \*\*   
## text\_amplify1 1.848514 0.356987 5.178 2.48e-07 \*\*\*  
## text\_flying 0.691526 0.035625 19.411 < 2e-16 \*\*\*  
## text\_trample 0.129558 0.067412 1.922 0.054772 .   
## text\_amplify2 NA NA NA NA   
## text\_provoke 1.726146 0.437458 3.946 8.24e-05 \*\*\*  
## text\_assist 1.041345 0.441700 2.358 0.018496 \*   
## text\_haste 0.515017 0.063098 8.162 5.94e-16 \*\*\*  
## text\_banding 0.341174 0.109529 3.115 0.001868 \*\*   
## text\_battlecry 0.613138 0.276343 2.219 0.026622 \*   
## text\_bloodthirst1 0.197639 0.209661 0.943 0.345975   
## text\_firststrike 0.600907 0.061573 9.759 < 2e-16 \*\*\*  
## text\_flanking 0.233373 0.257759 0.905 0.365373   
## text\_islandwalk 0.613332 0.152804 4.014 6.21e-05 \*\*\*  
## text\_bloodthirst2 0.932886 0.176495 5.286 1.40e-07 \*\*\*  
## text\_lifelink 0.106842 0.081792 1.306 0.191622   
## text\_menace 0.351459 0.114327 3.074 0.002141 \*\*   
## text\_bloodthirst3 1.849767 0.308956 5.987 2.55e-09 \*\*\*  
## text\_bloodthirst6 NA NA NA NA   
## text\_bushido1 0.642892 0.252620 2.545 0.011010 \*   
## text\_bushido2 1.499423 0.437404 3.428 0.000621 \*\*\*  
## text\_cascade 0.827808 0.163484 5.064 4.52e-07 \*\*\*  
## text\_changeling 0.150548 0.219300 0.686 0.492487   
## text\_deathtouch 0.217519 0.086263 2.522 0.011764 \*   
## text\_vigilance 0.254276 0.055695 4.565 5.30e-06 \*\*\*  
## text\_convoke 1.596827 0.198029 8.064 1.30e-15 \*\*\*  
## text\_devour2 1.011340 0.620790 1.629 0.103455   
## text\_hexproof 0.403971 0.128923 3.133 0.001754 \*\*   
## text\_indestructible 1.234649 0.215952 5.717 1.26e-08 \*\*\*  
## text\_afterlife1 0.293610 0.436419 0.673 0.501175   
## text\_renown1 0.095594 0.196079 0.488 0.625939   
## text\_unleash 0.328058 0.179872 1.824 0.068333 .   
## text\_defender -0.631431 0.076568 -8.247 3.01e-16 \*\*\*  
## text\_evolve 0.475013 0.234512 2.026 0.042952 \*   
## text\_exalted 0.578243 0.186876 3.094 0.002002 \*\*   
## text\_extort 0.264695 0.219651 1.205 0.228327   
## text\_shroud 0.033422 0.128288 0.261 0.794491   
## text\_wither 0.253748 0.191246 1.327 0.184731   
## text\_reach 0.353195 0.072137 4.896 1.06e-06 \*\*\*  
## text\_delve 1.729592 0.221925 7.794 1.07e-14 \*\*\*  
## text\_devoid -0.036104 0.619036 -0.058 0.953498   
## text\_ingest -0.275615 0.872892 -0.316 0.752228   
## text\_devour1 -0.027481 0.317796 -0.086 0.931098   
## text\_devour3 2.016021 0.438348 4.599 4.52e-06 \*\*\*  
## text\_doublestrike 1.124974 0.114393 9.834 < 2e-16 \*\*\*  
## text\_prowess 0.165819 0.130869 1.267 0.205289   
## text\_undying 0.800755 0.233530 3.429 0.000619 \*\*\*  
## text\_dredge3 0.569716 0.356051 1.600 0.109745   
## text\_fabricate1 0.794595 0.187420 4.240 2.35e-05 \*\*\*  
## text\_fabricate2 2.530175 0.617560 4.097 4.36e-05 \*\*\*  
## text\_fading2 NA NA NA NA   
## text\_fear 1.061608 0.190288 5.579 2.77e-08 \*\*\*  
## text\_persist 0.866196 0.238911 3.626 0.000296 \*\*\*  
## text\_soulshift5 1.885831 0.280020 6.735 2.17e-11 \*\*\*  
## text\_forestwalk 0.247512 0.115625 2.141 0.032431 \*   
## text\_infect 0.332614 0.196036 1.697 0.089918 .   
## text\_partner -1.580056 0.635270 -2.487 0.012960 \*   
## text\_mentor 0.299180 0.276435 1.082 0.279265   
## text\_modular4 NA NA NA NA   
## text\_ripple4 NA NA NA NA   
## text\_flash 0.763778 0.097990 7.794 1.06e-14 \*\*\*  
## text\_fading3 -1.345389 0.293650 -4.582 4.92e-06 \*\*\*  
## text\_melee 0.515871 0.437137 1.180 0.238105   
## text\_modular1 0.846074 0.310239 2.727 0.006447 \*\*   
## text\_phasing -1.092226 0.277111 -3.941 8.39e-05 \*\*\*  
## text\_soulshift3 0.996756 0.616350 1.617 0.106004   
## text\_soulshift4 1.377885 0.308726 4.463 8.55e-06 \*\*\*  
## text\_swampwalk 0.773331 0.097088 7.965 2.82e-15 \*\*\*  
## text\_rampage4 NA NA NA NA   
## text\_myriad -0.173022 0.311648 -0.555 0.578834   
## text\_graft1 0.925413 0.616743 1.500 0.133656   
## text\_horsemanship 1.309497 0.219840 5.957 3.06e-09 \*\*\*  
## text\_modular2 3.253938 0.619308 5.254 1.65e-07 \*\*\*  
## text\_rampage1 0.959282 0.309637 3.098 0.001976 \*\*   
## text\_dethrone 1.284311 0.628831 2.042 0.041252 \*   
## text\_improvise 1.335844 0.277728 4.810 1.63e-06 \*\*\*  
## text\_intimidate 0.930201 0.219070 4.246 2.28e-05 \*\*\*  
## text\_soulshift6 2.099691 0.625965 3.354 0.000811 \*\*\*  
## text\_modular3 4.768955 0.616819 7.732 1.71e-14 \*\*\*  
## text\_mountainwalk 0.323686 0.143787 2.251 0.024490 \*   
## text\_afflict2 -0.001275 0.629369 -0.002 0.998384   
## text\_rampage2 1.658464 0.261195 6.350 2.70e-10 \*\*\*  
## text\_rampage3 2.257847 0.617706 3.655 0.000264 \*\*\*  
## text\_riot 0.520701 0.309859 1.680 0.093036 .   
## text\_shadow 0.116630 0.309030 0.377 0.705914   
## text\_vanishing4 -1.335206 0.629611 -2.121 0.034078 \*   
## text\_skulk 0.747481 0.621025 1.204 0.228885   
## text\_soulshift7 0.693249 0.435956 1.590 0.111961   
## text\_sunburst 3.423192 0.310208 11.035 < 2e-16 \*\*\*  
## text\_renown2 0.322484 0.311008 1.037 0.299916   
## text\_soulshift8 1.641747 0.620699 2.645 0.008237 \*\*   
## text\_vanishing2 -3.219112 0.617773 -5.211 2.09e-07 \*\*\*  
## text\_bushido5 2.691066 0.654894 4.109 4.14e-05 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for gaussian family taken to be 0.378719)  
##   
## Null deviance: 5921.92 on 1985 degrees of freedom  
## Residual deviance: 715.78 on 1890 degrees of freedom  
## AIC: 3803.3  
##   
## Number of Fisher Scoring iterations: 2

#use model to predict the test values  
prediction <- predict(mtg.glm, testData)

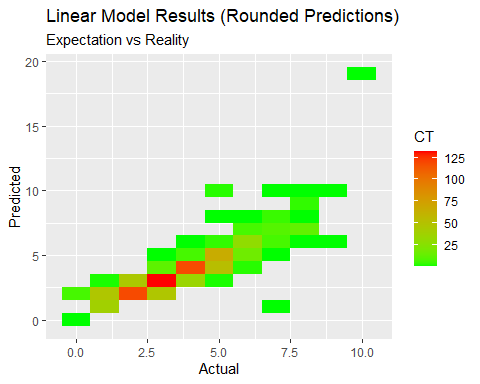
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

#create a data frame containing actual values and predicted values  
predictions <- data.frame(testData$convertedManaCost, prediction, testData$convertedManaCost-prediction)  
names(predictions) <- c("Actual", "Predicted", "Difference")  
  
#visualize Actual vs Predicted results with a linear model line to show fit  
g <- ggplot(predictions, aes(x=Actual, y=Predicted))  
  
g + geom\_point() +   
 geom\_smooth(method="lm", se=F) +   
 labs(subtitle="Expectation vs Reality",   
 y="Predicted",   
 x="Actual",   
 title="Linear Model Results")

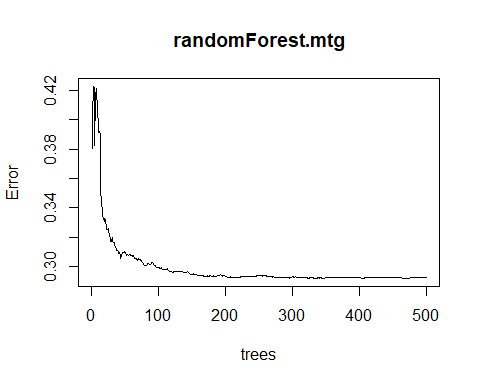
## `geom\_smooth()` using formula 'y ~ x'



#round predicted values (convertedManacost is an Int), then do a group count Actual vs Prediction, then format for visualization  
predictions.rounded <- data.frame(Grouped=interaction(testData$convertedManaCost, round(prediction), sep='x'), CT=1)  
predictions.rounded <- aggregate(predictions.rounded$CT, by=list(Category=predictions.rounded$Grouped), FUN=sum)  
predictions.rounded <- separate(data=predictions.rounded, col=Category, into=c("Actual", "Predicted"), sep="x")  
predictions.rounded <- transform(predictions.rounded, Actual = as.numeric(Actual), Predicted = as.numeric(Predicted))  
names(predictions.rounded) <- c("Actual", "Predicted", "CT")  
  
  
#visualize Actual vs Predicted using a heatmap  
g <- ggplot(predictions.rounded, aes(x=Actual, y=Predicted, fill=CT))  
  
g + geom\_point() + geom\_tile() +  
 scale\_fill\_gradient(low="green", high="red") +  
 labs(subtitle="Expectation vs Reality",   
 y="Predicted",   
 x="Actual",   
 title="Linear Model Results (Rounded Predictions)")

 Interesting results; some outliers, but generally the heatmap shows strong predictive capabilities.

#generate random forest model  
randomForest.mtg <- randomForest(convertedManaCost~., data=trainingData)  
  
#plot to determine best number of trees  
plot(randomForest.mtg)



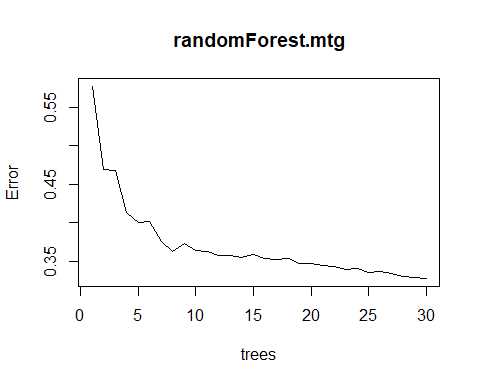
#plot shows best number of trees around 30, remodel with ntree=30  
randomForest.mtg <- randomForest(convertedManaCost~., data=trainingData, ntree=30)  
  
#show model results  
summary(randomForest.mtg)

## Length Class Mode   
## call 4 -none- call   
## type 1 -none- character  
## predicted 1986 -none- numeric   
## mse 30 -none- numeric   
## rsq 30 -none- numeric   
## oob.times 1986 -none- numeric   
## importance 102 -none- numeric   
## importanceSD 0 -none- NULL   
## localImportance 0 -none- NULL   
## proximity 0 -none- NULL   
## ntree 1 -none- numeric   
## mtry 1 -none- numeric   
## forest 11 -none- list   
## coefs 0 -none- NULL   
## y 1986 -none- numeric   
## test 0 -none- NULL   
## inbag 0 -none- NULL   
## terms 3 terms call

print(randomForest.mtg)

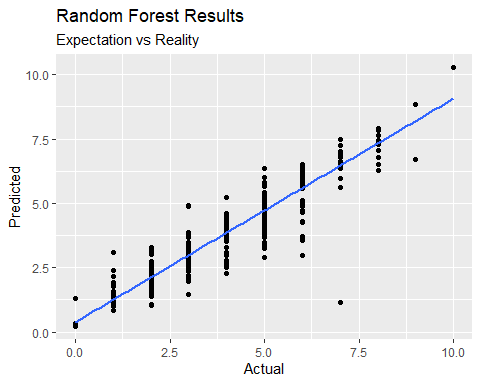
##   
## Call:  
## randomForest(formula = convertedManaCost ~ ., data = trainingData, ntree = 30)   
## Type of random forest: regression  
## Number of trees: 30  
## No. of variables tried at each split: 34  
##   
## Mean of squared residuals: 0.327657  
## % Var explained: 89.01

plot(randomForest.mtg)

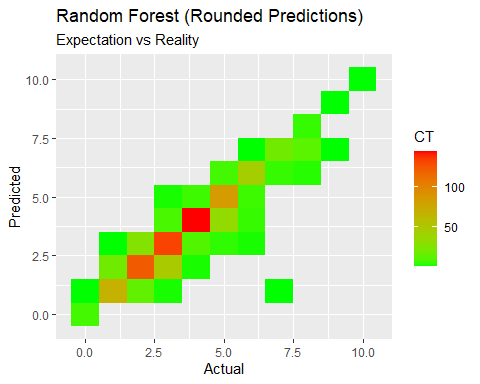


#predict against test data using random forest model  
prediction <- predict(randomForest.mtg, testData)  
  
#create dataframe from predicted values  
predictions <- data.frame(testData$convertedManaCost, prediction)  
names(predictions) <- c("Actual", "Predicted")  
  
#plot random forest results with linear model to show fit  
g <- ggplot(predictions, aes(x=Actual, y=Predicted))  
  
g + geom\_point() +   
 geom\_smooth(method="lm", se=F) +   
 labs(subtitle="Expectation vs Reality",   
 y="Predicted",   
 x="Actual",   
 title="Random Forest Results")

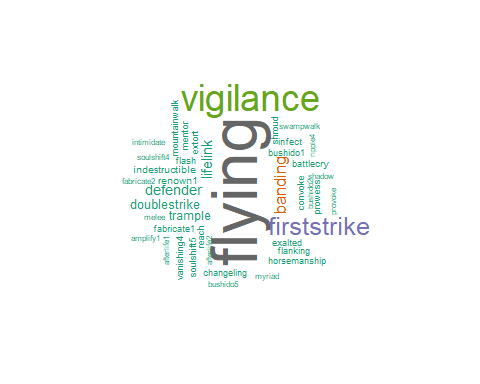
## `geom\_smooth()` using formula 'y ~ x'



#create heat map grouping dataframe for random forest model  
predictions.rounded <- data.frame(Grouped=interaction(testData$convertedManaCost, round(prediction), sep='x'), CT=1)  
predictions.rounded <- aggregate(predictions.rounded$CT, by=list(Category=predictions.rounded$Grouped), FUN=sum)  
predictions.rounded <- separate(data=predictions.rounded, col=Category, into=c("Actual", "Predicted"), sep="x")  
predictions.rounded <- transform(predictions.rounded, Actual = as.numeric(Actual), Predicted = as.numeric(Predicted))  
names(predictions.rounded) <- c("Actual", "Predicted", "CT")  
  
#plot heatmap  
g <- ggplot(predictions.rounded, aes(x=Actual, y=Predicted, fill=CT))  
  
g + geom\_point() + geom\_tile() +  
 scale\_fill\_gradient(low="green", high="red") +  
 labs(subtitle="Expectation vs Reality",   
 y="Predicted",   
 x="Actual",   
 title="Random Forest (Rounded Predictions)")



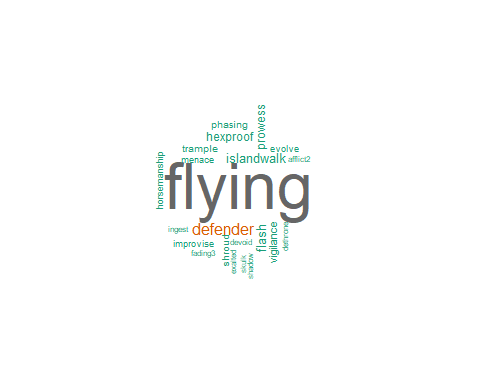
#Each section is the same except for the initial filter.  
#The steps are as follows:  
# 1 - filter to single color identity  
# 2 - generate text corpus from text & types  
# 3 - turn that into a wordcloud and visualize  
  
white <- df[ which(df$colorIdentity =='W'), ]  
  
white.text <- Corpus(VectorSource(white$text))  
tdm.white.text <- TermDocumentMatrix(white.text)  
matrix.white.text <- as.matrix(tdm.white.text)  
count.white.text <- sort(rowSums(matrix.white.text), decreasing=TRUE)  
df.white.text <- data.frame(word = names(count.white.text), freq=count.white.text)  
  
white.types <- Corpus(VectorSource(white$types))  
tdm.white.types <- TermDocumentMatrix(white.types)  
matrix.white.types <- as.matrix(tdm.white.types)  
count.white.types <- sort(rowSums(matrix.white.types), decreasing=TRUE)  
df.white.types <- data.frame(word = names(count.white.types), freq=count.white.types)  
  
wordcloud(words = df.white.text$word, freq = df.white.text$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



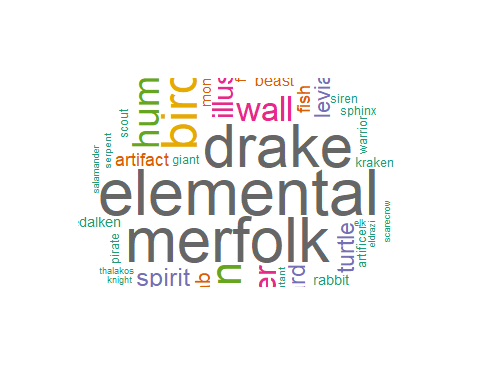
wordcloud(words = df.white.types$word, freq = df.white.types$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



blue <- df[ which(df$colorIdentity =='U'), ]  
  
blue.text <- Corpus(VectorSource(blue$text))  
tdm.blue.text <- TermDocumentMatrix(blue.text)  
matrix.blue.text <- as.matrix(tdm.blue.text)  
count.blue.text <- sort(rowSums(matrix.blue.text), decreasing=TRUE)  
df.blue.text <- data.frame(word = names(count.blue.text), freq=count.blue.text)  
  
blue.types <- Corpus(VectorSource(blue$types))  
tdm.blue.types <- TermDocumentMatrix(blue.types)  
matrix.blue.types <- as.matrix(tdm.blue.types)  
count.blue.types <- sort(rowSums(matrix.blue.types), decreasing=TRUE)  
df.blue.types <- data.frame(word = names(count.blue.types), freq=count.blue.types)  
  
wordcloud(words = df.blue.text$word, freq = df.blue.text$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



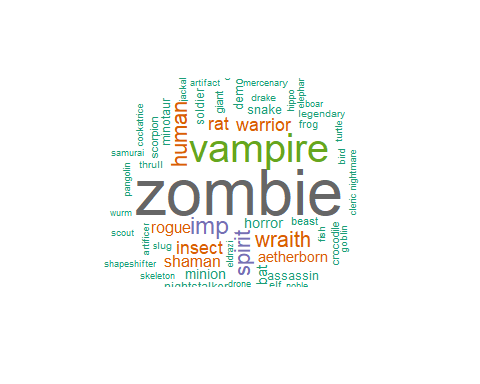
wordcloud(words = df.blue.types$word, freq = df.blue.types$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



black <- df[ which(df$colorIdentity =='B'), ]  
  
black.text <- Corpus(VectorSource(black$text))  
tdm.black.text <- TermDocumentMatrix(black.text)  
matrix.black.text <- as.matrix(tdm.black.text)  
count.black.text <- sort(rowSums(matrix.black.text), decreasing=TRUE)  
df.black.text <- data.frame(word = names(count.black.text), freq=count.black.text)  
  
black.types <- Corpus(VectorSource(black$types))  
tdm.black.types <- TermDocumentMatrix(black.types)  
matrix.black.types <- as.matrix(tdm.black.types)  
count.black.types <- sort(rowSums(matrix.black.types), decreasing=TRUE)  
df.black.types <- data.frame(word = names(count.black.types), freq=count.black.types)  
  
wordcloud(words = df.black.text$word, freq = df.black.text$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



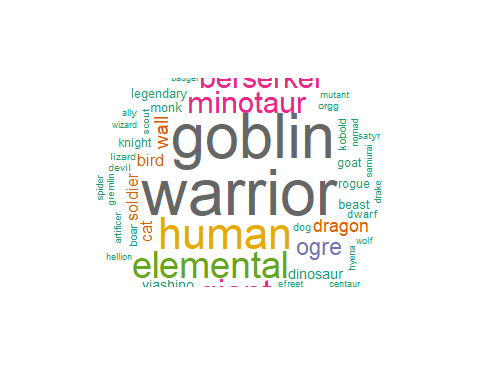
wordcloud(words = df.black.types$word, freq = df.black.types$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



red <- df[ which(df$colorIdentity =='R'), ]  
  
red.text <- Corpus(VectorSource(red$text))  
tdm.red.text <- TermDocumentMatrix(red.text)  
matrix.red.text <- as.matrix(tdm.red.text)  
count.red.text <- sort(rowSums(matrix.red.text), decreasing=TRUE)  
df.red.text <- data.frame(word = names(count.red.text), freq=count.red.text)  
  
red.types <- Corpus(VectorSource(red$types))  
tdm.red.types <- TermDocumentMatrix(red.types)  
matrix.red.types <- as.matrix(tdm.red.types)  
count.red.types <- sort(rowSums(matrix.red.types), decreasing=TRUE)  
df.red.types <- data.frame(word = names(count.red.types), freq=count.red.types)  
  
wordcloud(words = df.red.text$word, freq = df.red.text$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



wordcloud(words = df.red.types$word, freq = df.red.types$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



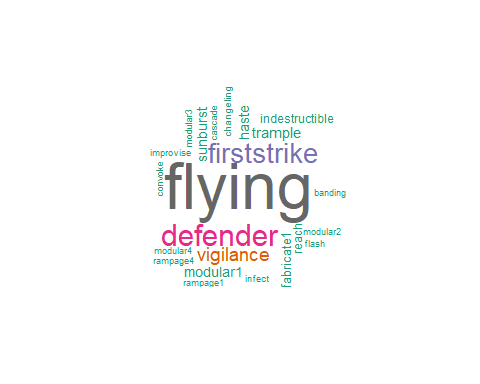
green <- df[ which(df$colorIdentity =='G'), ]  
  
green.text <- Corpus(VectorSource(green$text))  
tdm.green.text <- TermDocumentMatrix(green.text)  
matrix.green.text <- as.matrix(tdm.green.text)  
count.green.text <- sort(rowSums(matrix.green.text), decreasing=TRUE)  
df.green.text <- data.frame(word = names(count.green.text), freq=count.green.text)  
  
green.types <- Corpus(VectorSource(green$types))  
tdm.green.types <- TermDocumentMatrix(green.types)  
matrix.green.types <- as.matrix(tdm.green.types)  
count.green.types <- sort(rowSums(matrix.green.types), decreasing=TRUE)  
df.green.types <- data.frame(word = names(count.green.types), freq=count.green.types)  
  
wordcloud(words = df.green.text$word, freq = df.green.text$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



wordcloud(words = df.green.types$word, freq = df.green.types$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



colorless <- df[ which(df$colorIdentity =='X'), ]  
  
colorless.text <- Corpus(VectorSource(colorless$text))  
tdm.colorless.text <- TermDocumentMatrix(colorless.text)  
matrix.colorless.text <- as.matrix(tdm.colorless.text)  
count.colorless.text <- sort(rowSums(matrix.colorless.text), decreasing=TRUE)  
df.colorless.text <- data.frame(word = names(count.colorless.text), freq=count.colorless.text)  
  
colorless.types <- Corpus(VectorSource(colorless$types))  
tdm.colorless.types <- TermDocumentMatrix(colorless.types)  
matrix.colorless.types <- as.matrix(tdm.colorless.types)  
count.colorless.types <- sort(rowSums(matrix.colorless.types), decreasing=TRUE)  
df.colorless.types <- data.frame(word = names(count.colorless.types), freq=count.colorless.types)  
  
wordcloud(words = df.colorless.text$word, freq = df.colorless.text$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))



wordcloud(words = df.colorless.types$word, freq = df.colorless.types$freq, min.freq = 1,  
 max.words=200, random.order=FALSE, rot.per=0.35,   
 colors=brewer.pal(8, "Dark2"))

