Clustering Credit Card Customers for Targeted Marketing using Unsupervised Learning

CSML1000 Project #2, by Group 8

Tamer Hanna tamerh@my.yorku.ca (mailto:tamerh@my.yorku.ca) Pete Gray ptgray@my.yorku.ca (mailto:ptgray@my.yorku.ca) Xiaohai Lu yu271637@my.yorku.ca (mailto:yu271637@my.yorku.ca) Haofeng Zhou zhf85@my.yorku.ca (mailto:zhf85@my.yorku.ca)

```
library(dplyr);
library(ggplot2);
library(knitr);
library(validate);
library(tidyverse);  # data manipulation
library(cluster);    # clustering algorithms
library(clusterSim);
library(factoextra);
library(fpc);
```

OVERVIEW

Using data on the behaviour of credit card customers, we can use unsupervised learning to discover market segments that would be useful for targeting marketing strategies.

The dataset can be found here: https://www.kaggle.com/arjunbhasin2013/ccdata (https://www.kaggle.com/arjunbhasin2013/ccdata)

Our dataset has 19 columns of data on 9000 customers. Using K-means and PCA, we can determine an optimal number of market segments, discover the identifying properties of those segments, and be able to describe to the marketing department what the most significant behaviours of the people in those segments are. Marketing campaigns, then, can be targeted at, for example, impulse shoppers, or big spenders.

BUSINESS UNDERSTANDING

Applying specific marketing strategies to different types of customers can improve results and reduce costs. Credit card customers can be profiled by the way they use, and pay off, their card. The dataset contains data about how customers behave - how much they spend, the type of spending they do, and their frequency of each type of transaction. By understanding the patterns in the data, we will be able to select an optimal subset of our customer base for a targeted marketing campaign.

Business Objectives

Behavioural data can be mined to discover identifying features of clusters of customers who use their card in similar ways. The marketing department can use these insights to select target groups and optimize the marketing used to target them.

Data Mining Goals

Using shallow algorithm unsupervised learning, we will discover patterns in the data, and from those patterns we will discover an optimal set of clusters, or customer segments, with which we can associate certain behaviours. We would then like to be able to use our model to predict the appropriate segment, or target market, of customers who are not in our trianing data.

DATA UNDERSTANDING

Collect Initial Data

Load the data from the local filesystem:

```
#load data
df <- read.csv("credit-card-cust-behav-data.csv", stringsAsFactors = FALSE, header = TRUE, encoding  = "UTF-8")
```

Output a quick and dirty summary of the dataset, to ensure that we haven't loaded something scrambled, or the wrong thing:

```
str(df)
```

```
## 'data.frame':    8950 obs. of  18 variables:
##  $ CUST_ID                         : chr  "C10001" "C10002" "C10003" "C10004" ...
##  $ BALANCE                         : num  40.9 3202.5 2495.1 1666.7 817.7 ...
##  $ BALANCE_FREQUENCY               : num  0.818 0.909 1 0.636 1 ...
##  $ PURCHASES                       : num  95.4 0 773.2 1499 16 ...
##  $ ONEOFF_PURCHASES                : num  0 0 773 1499 16 ...
##  $ INSTALLMENTS_PURCHASES          : num  95.4 0 0 0 0 ...
##  $ CASH_ADVANCE                    : num  0 6443 0 206 0 ...
##  $ PURCHASES_FREQUENCY             : num  0.1667 0 1 0.0833 0.0833 ...
##  $ ONEOFF_PURCHASES_FREQUENCY      : num  0 0 1 0.0833 0.0833 ...
##  $ PURCHASES_INSTALLMENTS_FREQUENCY: num  0.0833 0 0 0 0 ...
##  $ CASH_ADVANCE_FREQUENCY          : num  0 0.25 0 0.0833 0 ...
##  $ CASH_ADVANCE_TRX                : int  0 4 0 1 0 0 0 0 0 0 ...
##  $ PURCHASES_TRX                   : int  2 0 12 1 1 8 64 12 5 3 ...
##  $ CREDIT_LIMIT                    : num  1000 7000 7500 7500 1200 1800 13500 2300 7000 11000 ...
##  $ PAYMENTS                        : num  202 4103 622 0 678 ...
##  $ MINIMUM_PAYMENTS                : num  140 1072 627 NA 245 ...
##  $ PRC_FULL_PAYMENT                : num  0 0.222 0 0 0 ...
##  $ TENURE                          : int  12 12 12 12 12 12 12 12 12 12 ...
```

Describe Data

The data describe 4 different spending behaviours:

- Purchases
- One-off Purchases
- Installment Purchases
- Cash Advances.

For each of these behaviours, there are 3 types of data:

- Dollar amount
- Number of transactions
- Frequency of transactions

There is also data about Credit Limit, Balance, and various behaviours related to payments.

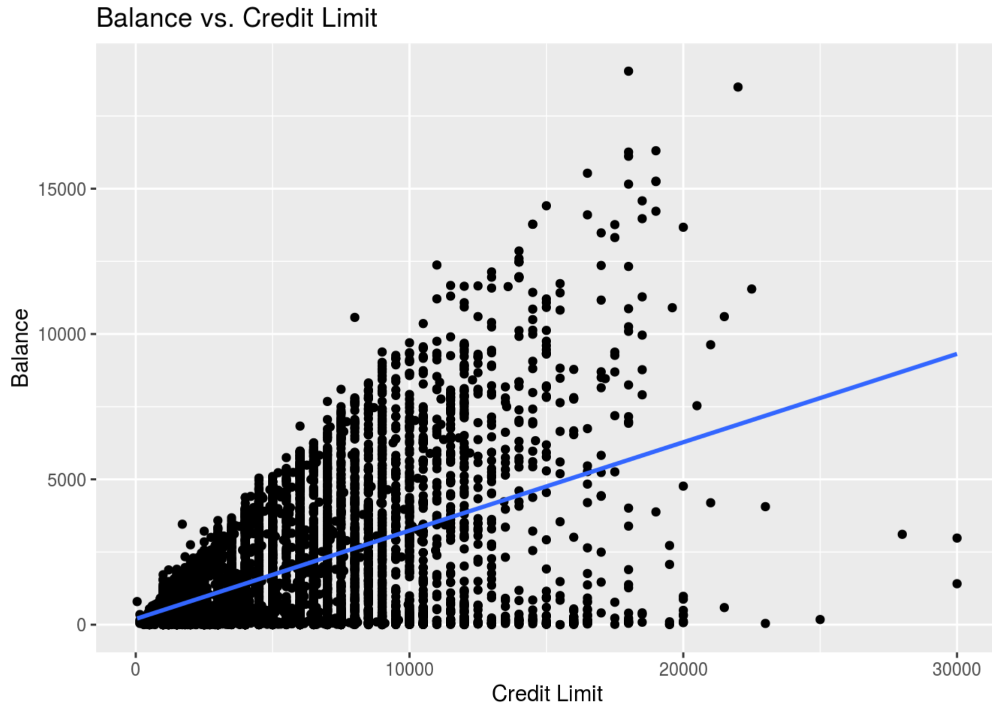
We are mostly interested in the spending behaviours, but will look at the predictive value of other data as well.

Explore Data

We'll plot some basic graphs, to ensure that the data conform to our limited domain understanding.

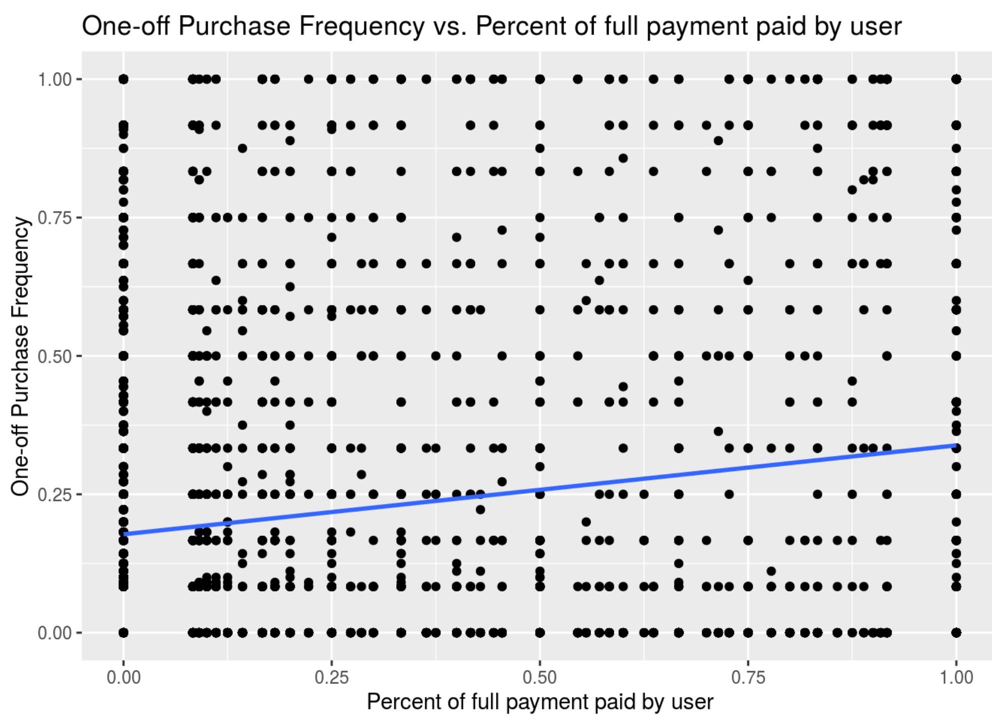
Balance vs. Credit Limit - we would expect to find a strong correlation here, even if only because those with small credit limits must have small balances. We certainly find it.

```
df %>% ggplot(aes(y=BALANCE, x=CREDIT_LIMIT)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="Balance vs. Credit Limit", x="Credit Limit", y="Balance")
```



Maybe those who pay higher portion of balance purchase more one-offs? No, this graph is pretty junky and doesn’t tell us anything.

```
df %>% ggplot(aes(y=ONEOFF_PURCHASES_FREQUENCY, x=PRC_FULL_PAYMENT)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="One-off Purchase Frequency vs. Percent of full payment paid by user", x="Percent of full payment paid by user"
, y="One-off Purchase Frequency")
```

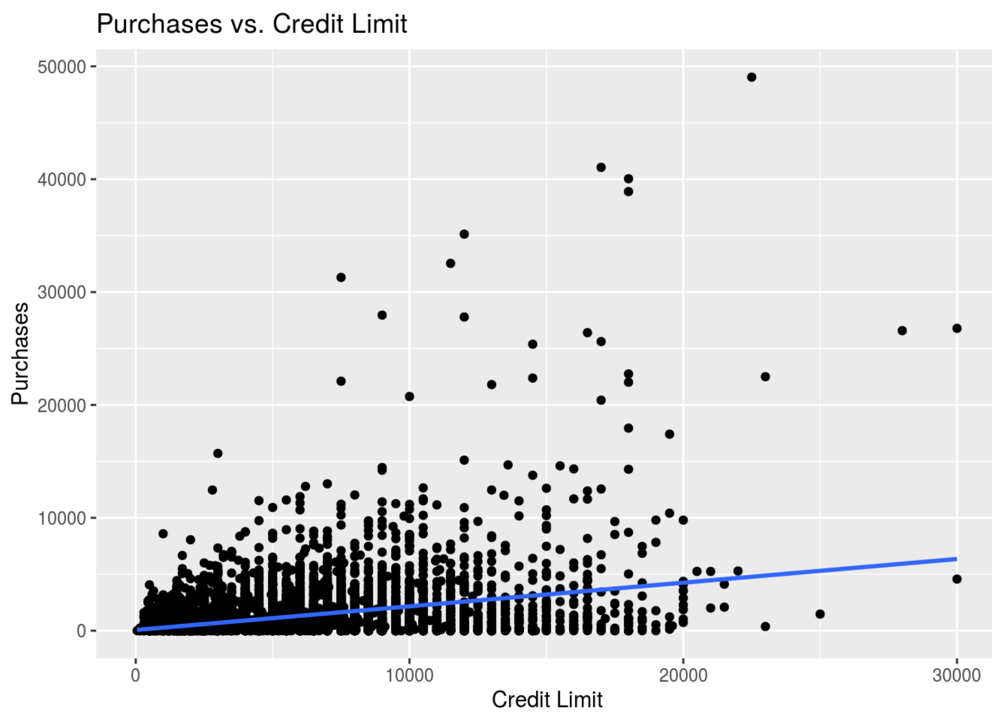


Here we can see that there appears to be a correlation between a customers credit limit, and the frequency of their purchases. Not surprising, given that frequent purchasing is one factor that leads to an increased credit limit.

```
df %>% ggplot(aes(y=PURCHASES_FREQUENCY, x=CREDIT_LIMIT)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="Purchase Frequency vs. Credit Limit", x="Credit Limit", y="Purchase Frequency")
```

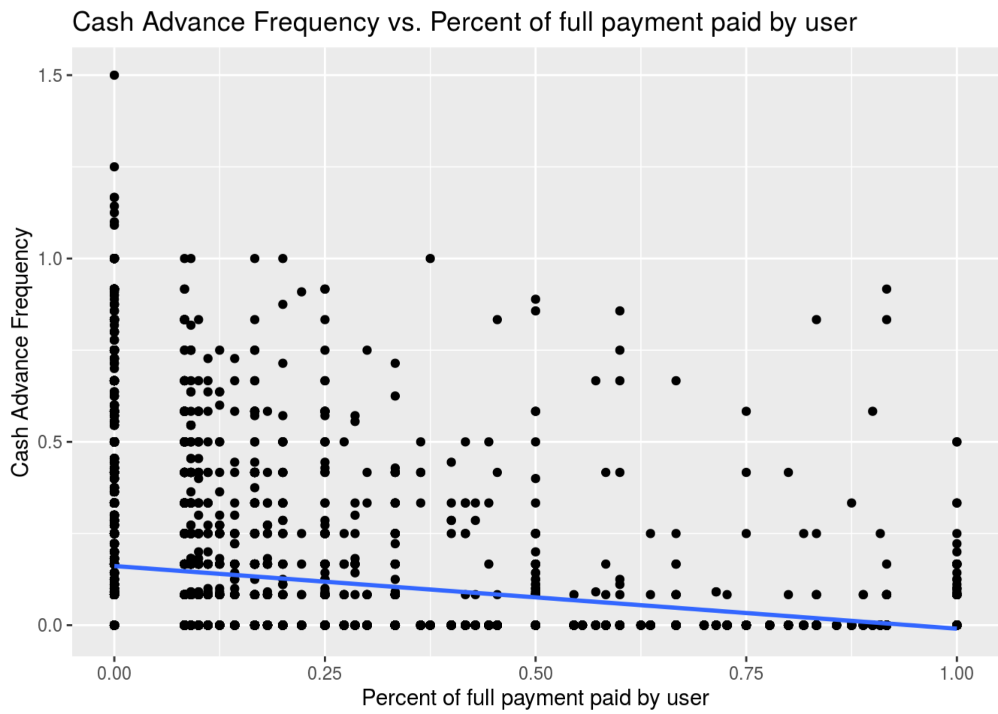
Higher credit limit, more purchases? Maybe. This graph is an excellent showcase of “outliers”. Only a small handful of the 9,000 records show a credit limit, or purchases, greater than $20,000. This will be one of our guiding intuitions when we get to the Data Cleaning phase.

```
df %>% ggplot(aes(y=PURCHASES, x=CREDIT_LIMIT)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="Purchases vs. Credit Limit", x="Credit Limit", y="Purchases")
```



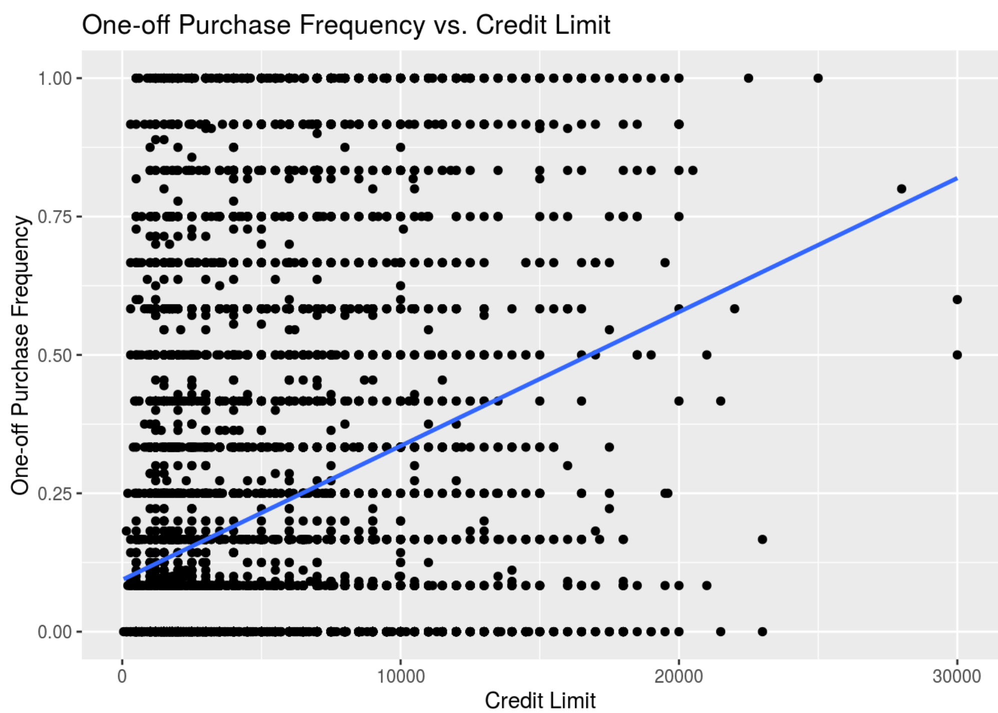
It would appear that those who pay off thier balance, do not take cash advances. No surprises, but a little hard to read because of the scale. One thing that appears in this graphs is a large “column” of people who are completely unlikely to pay off their entire balance, and take cash advances with very high frequency. (It is likely that the marketing department may have less interest in these people, than in others. But we’ll see!)

```
df %>% ggplot(aes(y=CASH_ADVANCE_FREQUENCY, x=PRC_FULL_PAYMENT)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="Cash Advance Frequency vs. Percent of full payment paid by user", x="Percent of full payment paid by user", y=
"Cash Advance Frequency")
```



One last graph, as part of our data exploration and sanity check. This one shows that there is a clear corrleation between a customer’s credit limit, and their frequency of one-off purchases. By now, we can be comfortable that our data isn’t erratic, and that our limited domain knowledge isn’t out to lunch.

```
df %>% ggplot(aes(y=ONEOFF_PURCHASES_FREQUENCY, x=CREDIT_LIMIT)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="One-off Purchase Frequency vs. Credit Limit", x="Credit Limit", y="One-off Purchase Frequency")
```



Verify Data Quality

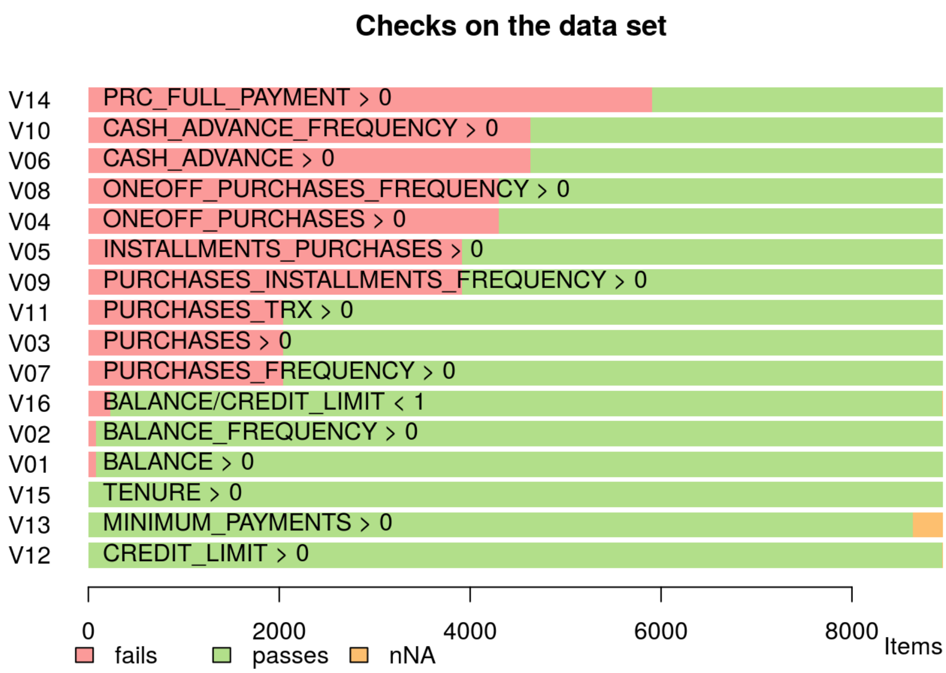
Check if data greater than zero, look for missing data

```
cf <- check_that(df, BALANCE > 0, BALANCE_FREQUENCY > 0, PURCHASES > 0, ONEOFF_PURCHASES > 0, INSTALLMENTS_PURCHASES > 0, CASH_ADVANCE > 0, PURCHASES_FREQUENCY > 0, ONEOFF_PURCHASES_FREQUENCY > 0, PURCHASES_INSTALLMENTS_FREQUENCY > 0, CASH_ADVANCE_FREQUENCY > 0, PURCHASES_TRX > 0, CREDIT_LIMIT > 0, MINIMUM_PAYMENTS > 0, PRC_FULL_PAYMENT > 0, TENURE > 0, BALANCE/CREDIT_LIMIT < 1)
summary(cf)
```

```
##    name items passes fails nNA error warning
## 1   V01  8950   8870    80   0 FALSE   FALSE
## 2   V02  8950   8870    80   0 FALSE   FALSE
## 3   V03  8950   6906  2044   0 FALSE   FALSE
## 4   V04  8950   4648  4302   0 FALSE   FALSE
## 5   V05  8950   5034  3916   0 FALSE   FALSE
## 6   V06  8950   4322  4628   0 FALSE   FALSE
## 7   V07  8950   6907  2043   0 FALSE   FALSE
## 8   V08  8950   4648  4302   0 FALSE   FALSE
## 9   V09  8950   5035  3915   0 FALSE   FALSE
## 10  V10  8950   4322  4628   0 FALSE   FALSE
## 11  V11  8950   6906  2044   0 FALSE   FALSE
## 12  V12  8950   8949     0   1 FALSE   FALSE
## 13  V13  8950   8637     0 313 FALSE   FALSE
## 14  V14  8950   3047  5903   0 FALSE   FALSE
## 15  V15  8950   8950     0   0 FALSE   FALSE
## 16  V16  8950   8722   227   1 FALSE   FALSE
##                              expression
## 1                           BALANCE > 0
## 2                 BALANCE_FREQUENCY > 0
## 3                         PURCHASES > 0
## 4                  ONEOFF_PURCHASES > 0
## 5            INSTALLMENTS_PURCHASES > 0
## 6                      CASH_ADVANCE > 0
## 7               PURCHASES_FREQUENCY > 0
## 8        ONEOFF_PURCHASES_FREQUENCY > 0
## 9  PURCHASES_INSTALLMENTS_FREQUENCY > 0
## 10           CASH_ADVANCE_FREQUENCY > 0
## 11                    PURCHASES_TRX > 0
## 12                     CREDIT_LIMIT > 0
## 13                 MINIMUM_PAYMENTS > 0
## 14                 PRC_FULL_PAYMENT > 0
## 15                           TENURE > 0
## 16             BALANCE/CREDIT_LIMIT < 1
```

There appear to be a great number of zeroes. Let’s look at that a little more closely:

```
barplot(cf,main="Checks on the data set")
```



Lots of zero! Though, using our limitied knowledge of this domain, we can understand that this could be perfectly reasonable. In the first few lines of the above chart, we can see that a lot of people never make the full payment, a lot of people never get a cash advance, and some people didn't even make a purchase. Seems entirely reasonable. We can note too that the "zeroes" in PURCHASES_TRX, PURCHASES, and PURCHASES_FREQUENCY appear to be identical. In line with what we'd expect!

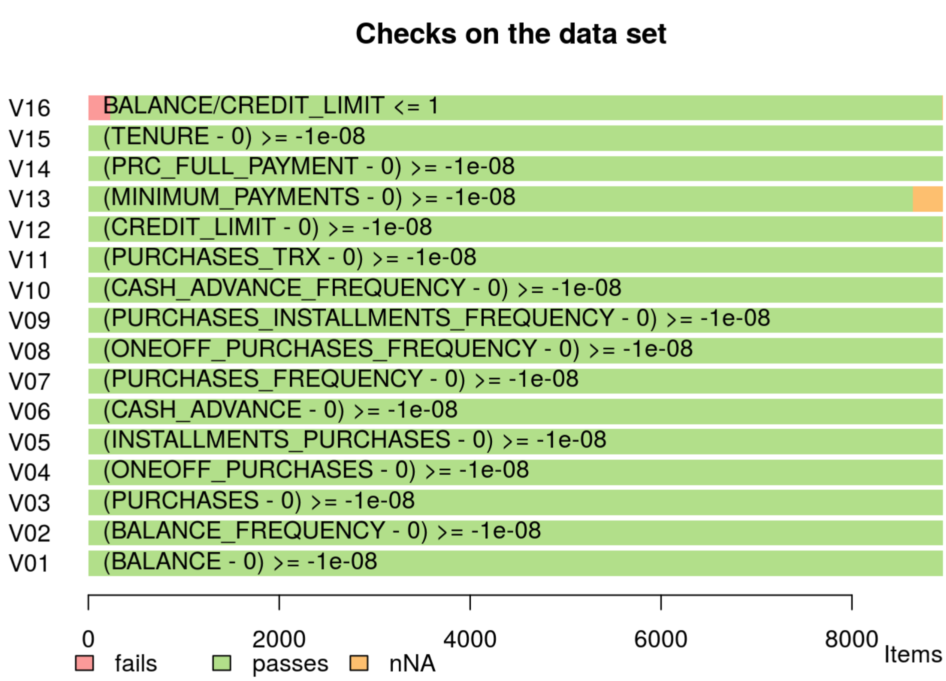
Now let's include zero, and see how it all stacks up for Greater OR Equal to zero:

```
cf <- check_that(df, BALANCE >= 0, BALANCE_FREQUENCY >= 0, PURCHASES >= 0, ONEOFF_PURCHASES >= 0, INSTALLMENTS_PURCHASES >= 0, CASH_ADVANCE >= 0, PURCHASES_FREQUENCY >= 0, ONEOFF_PURCHASES_FREQUENCY >= 0, PURCHASES_INSTALLMENTS_FREQUENCY >= 0, CASH_ADVANCE_FREQUENCY >= 0, PURCHASES_TRX >= 0, CREDIT_LIMIT >= 0, MINIMUM_PAYMENTS >= 0, PRC_FULL_PAYMENT >= 0, TENURE >= 0, BALANCE/CREDIT_LIMIT <= 1)
summary(cf)
```

```
##    name items passes fails nNA error warning
## 1   V01  8950   8950     0   0 FALSE   FALSE
## 2   V02  8950   8950     0   0 FALSE   FALSE
## 3   V03  8950   8950     0   0 FALSE   FALSE
## 4   V04  8950   8950     0   0 FALSE   FALSE
## 5   V05  8950   8950     0   0 FALSE   FALSE
## 6   V06  8950   8950     0   0 FALSE   FALSE
## 7   V07  8950   8950     0   0 FALSE   FALSE
## 8   V08  8950   8950     0   0 FALSE   FALSE
## 9   V09  8950   8950     0   0 FALSE   FALSE
## 10  V10  8950   8950     0   0 FALSE   FALSE
## 11  V11  8950   8950     0   0 FALSE   FALSE
## 12  V12  8950   8949     0   1 FALSE   FALSE
## 13  V13  8950   8637     0 313 FALSE   FALSE
## 14  V14  8950   8950     0   0 FALSE   FALSE
## 15  V15  8950   8950     0   0 FALSE   FALSE
## 16  V16  8950   8722   227   1 FALSE   FALSE
##                                     expression
## 1                         (BALANCE - 0) >= -1e-08
## 2               (BALANCE_FREQUENCY - 0) >= -1e-08
## 3                       (PURCHASES - 0) >= -1e-08
## 4                (ONEOFF_PURCHASES - 0) >= -1e-08
## 5          (INSTALLMENTS_PURCHASES - 0) >= -1e-08
## 6                    (CASH_ADVANCE - 0) >= -1e-08
## 7             (PURCHASES_FREQUENCY - 0) >= -1e-08
## 8      (ONEOFF_PURCHASES_FREQUENCY - 0) >= -1e-08
## 9  (PURCHASES_INSTALLMENTS_FREQUENCY - 0) >= -1e-08
## 10         (CASH_ADVANCE_FREQUENCY - 0) >= -1e-08
## 11                  (PURCHASES_TRX - 0) >= -1e-08
## 12                   (CREDIT_LIMIT - 0) >= -1e-08
## 13               (MINIMUM_PAYMENTS - 0) >= -1e-08
## 14               (PRC_FULL_PAYMENT - 0) >= -1e-08
## 15                         (TENURE - 0) >= -1e-08
## 16                     BALANCE/CREDIT_LIMIT <= 1
```

Our only serious problems remaining are some N/As in Minimum Payment. We also see that a few people have a balance greater than their credit limit - this would not constitue a problem with the data. Simply a problem the customer in question is having. (Their balance is higher than their credit limit)

```
barplot(cf,main="Checks on the data set")
```



As a last look at the data, let’s look at the variance of the individual columns.

```
var(df$BALANCE)
```

```
## [1] 4332775
```

```
var(df$BALANCE_FREQUENCY)
```

```
## [1] 0.05612351
```

```
var(df$PURCHASES)
```

```
## [1] 4565208
```

```
var(df$ONEOFF_PURCHASES)
```

```
## [1] 2755228
```

```
var(df$INSTALLMENTS_PURCHASES)
```

```
## [1] 817827.4
```

```
var(df$CASH_ADVANCE)
```

```
## [1] 4398096
```

```
var(df$PURCHASES_FREQUENCY)
```

```
## [1] 0.1610985
```

```
var(df$ONEOFF_PURCHASES_FREQUENCY)
```

```
## [1] 0.08900441
```

```
var(df$PURCHASES_INSTALLMENTS_FREQUENCY)
```

```
## [1] 0.1579647
```

```
var(df$CASH_ADVANCE_FREQUENCY)
```

```
## [1] 0.04004857
```

```
var(df$CASH_ADVANCE_TRX)
```

```
## [1] 46.5758
```

```
var(df$PURCHASES_TRX)
```

```
## [1] 617.9027
```

```
var(df$CREDIT_LIMIT)
```

```
## [1] NA
```

```
var(df$PAYMENTS)
```

```
## [1] 8381394
```

```
var(df$MINIMUM_PAYMENTS)
```

```
## [1] NA
```

```
var(df$PRC_FULL_PAYMENT)
```

```
## [1] 0.08555578
```

```
var(df$TENURE)
```

```
## [1] 1.791129
```

As expected, there is a big difference in the variance, between the different types of data. Dollar amounts are huge, and have huge variance. Transaction frequencies are small, and have small variance. As our ideal investigation into this dataset would involve clustering and predicting based on both types of values, we anticipate scaling some or all of the data columns we will be using, so that our algorithm can give comparable consideration to all of them.

DATA PREPARATION

Select Data

We saw that MINIMUM_PAYMENTS is riddled with N/As, so let's remove that column. This should be compatible with our Business Objectives, as we're hoping to get people to spend, rather than optimize their repayments.

```
df$MINIMUM_PAYMENTS <- NULL
```

And let's check that worked:

```
cf <- check_that(df, BALANCE >= 0, BALANCE_FREQUENCY >= 0, PURCHASES >= 0, ONEOFF_PURCHASES >= 0, INSTALLMENTS_PURCHASES >=
0, CASH_ADVANCE >= 0, PURCHASES_FREQUENCY >= 0, ONEOFF_PURCHASES_FREQUENCY >= 0, PURCHASES_INSTALLMENTS_FREQUENCY >= 0, CASH
_ADVANCE_FREQUENCY >= 0, PURCHASES_TRX >= 0, CREDIT_LIMIT >= 0, MINIMUM_PAYMENTS >= 0, PRC_FULL_PAYMENT >= 0, TENURE >= 0, B
ALANCE/CREDIT_LIMIT <= 1)
summary(cf)
```

```
##    name items passes fails nNA error warning
## 1   V01  8950   8950     0   0 FALSE   FALSE
## 2   V02  8950   8950     0   0 FALSE   FALSE
## 3   V03  8950   8950     0   0 FALSE   FALSE
## 4   V04  8950   8950     0   0 FALSE   FALSE
## 5   V05  8950   8950     0   0 FALSE   FALSE
## 6   V06  8950   8950     0   0 FALSE   FALSE
## 7   V07  8950   8950     0   0 FALSE   FALSE
## 8   V08  8950   8950     0   0 FALSE   FALSE
## 9   V09  8950   8950     0   0 FALSE   FALSE
## 10  V10  8950   8950     0   0 FALSE   FALSE
## 11  V11  8950   8950     0   0 FALSE   FALSE
## 12  V12  8950   8949     0   1 FALSE   FALSE
## 13  V13     0      0     0   0  TRUE   FALSE
## 14  V14  8950   8950     0   0 FALSE   FALSE
## 15  V15  8950   8950     0   0 FALSE   FALSE
## 16  V16  8950   8722   227   1 FALSE   FALSE
##                                          expression
## 1                           (BALANCE - 0) >= -1e-08
## 2                 (BALANCE_FREQUENCY - 0) >= -1e-08
## 3                         (PURCHASES - 0) >= -1e-08
## 4                  (ONEOFF_PURCHASES - 0) >= -1e-08
## 5            (INSTALLMENTS_PURCHASES - 0) >= -1e-08
## 6                      (CASH_ADVANCE - 0) >= -1e-08
## 7               (PURCHASES_FREQUENCY - 0) >= -1e-08
## 8        (ONEOFF_PURCHASES_FREQUENCY - 0) >= -1e-08
## 9  (PURCHASES_INSTALLMENTS_FREQUENCY - 0) >= -1e-08
## 10           (CASH_ADVANCE_FREQUENCY - 0) >= -1e-08
## 11                    (PURCHASES_TRX - 0) >= -1e-08
## 12                     (CREDIT_LIMIT - 0) >= -1e-08
## 13                 (MINIMUM_PAYMENTS - 0) >= -1e-08
## 14                 (PRC_FULL_PAYMENT - 0) >= -1e-08
## 15                           (TENURE - 0) >= -1e-08
## 16                        BALANCE/CREDIT_LIMIT <= 1
```

There were some N/As in CREDIT_LIMIT, but we'd really like to use that column in our clusterings and predicitons. let's remove those rows so we can use that column.

```
df <- na.omit(df)
```

Let's check that we haven't misunderstood that N/A, or the code we used to remove it, and accidentally ransacked the data we're using with that move - looks good.

```
cf <- check_that(df, BALANCE >= 0, BALANCE_FREQUENCY >= 0, PURCHASES >= 0, ONEOFF_PURCHASES >= 0, INSTALLMENTS_PURCHASES >=
0, CASH_ADVANCE >= 0, PURCHASES_FREQUENCY >= 0, ONEOFF_PURCHASES_FREQUENCY >= 0, PURCHASES_INSTALLMENTS_FREQUENCY >= 0, CASH
_ADVANCE_FREQUENCY >= 0, PURCHASES_TRX >= 0, CREDIT_LIMIT >= 0, MINIMUM_PAYMENTS >= 0, PRC_FULL_PAYMENT >= 0, TENURE >= 0, B
ALANCE/CREDIT_LIMIT <= 1)
summary(cf)
```

```
##    name items passes fails nNA error warning
## 1   V01  8949   8949     0   0 FALSE   FALSE
## 2   V02  8949   8949     0   0 FALSE   FALSE
## 3   V03  8949   8949     0   0 FALSE   FALSE
## 4   V04  8949   8949     0   0 FALSE   FALSE
## 5   V05  8949   8949     0   0 FALSE   FALSE
## 6   V06  8949   8949     0   0 FALSE   FALSE
## 7   V07  8949   8949     0   0 FALSE   FALSE
## 8   V08  8949   8949     0   0 FALSE   FALSE
## 9   V09  8949   8949     0   0 FALSE   FALSE
## 10  V10  8949   8949     0   0 FALSE   FALSE
## 11  V11  8949   8949     0   0 FALSE   FALSE
## 12  V12  8949   8949     0   0 FALSE   FALSE
## 13  V13     0      0     0   0  TRUE   FALSE
## 14  V14  8949   8949     0   0 FALSE   FALSE
## 15  V15  8949   8949     0   0 FALSE   FALSE
## 16  V16  8949   8722   227   0 FALSE   FALSE
##                                            expression
## 1                             (BALANCE - 0) >= -1e-08
## 2                   (BALANCE_FREQUENCY - 0) >= -1e-08
## 3                           (PURCHASES - 0) >= -1e-08
## 4                    (ONEOFF_PURCHASES - 0) >= -1e-08
## 5              (INSTALLMENTS_PURCHASES - 0) >= -1e-08
## 6                        (CASH_ADVANCE - 0) >= -1e-08
## 7                 (PURCHASES_FREQUENCY - 0) >= -1e-08
## 8          (ONEOFF_PURCHASES_FREQUENCY - 0) >= -1e-08
## 9   (PURCHASES_INSTALLMENTS_FREQUENCY - 0) >= -1e-08
## 10             (CASH_ADVANCE_FREQUENCY - 0) >= -1e-08
## 11                      (PURCHASES_TRX - 0) >= -1e-08
## 12                       (CREDIT_LIMIT - 0) >= -1e-08
## 13                   (MINIMUM_PAYMENTS - 0) >= -1e-08
## 14                   (PRC_FULL_PAYMENT - 0) >= -1e-08
## 15                             (TENURE - 0) >= -1e-08
## 16                        BALANCE/CREDIT_LIMIT <= 1
```

Clean Data

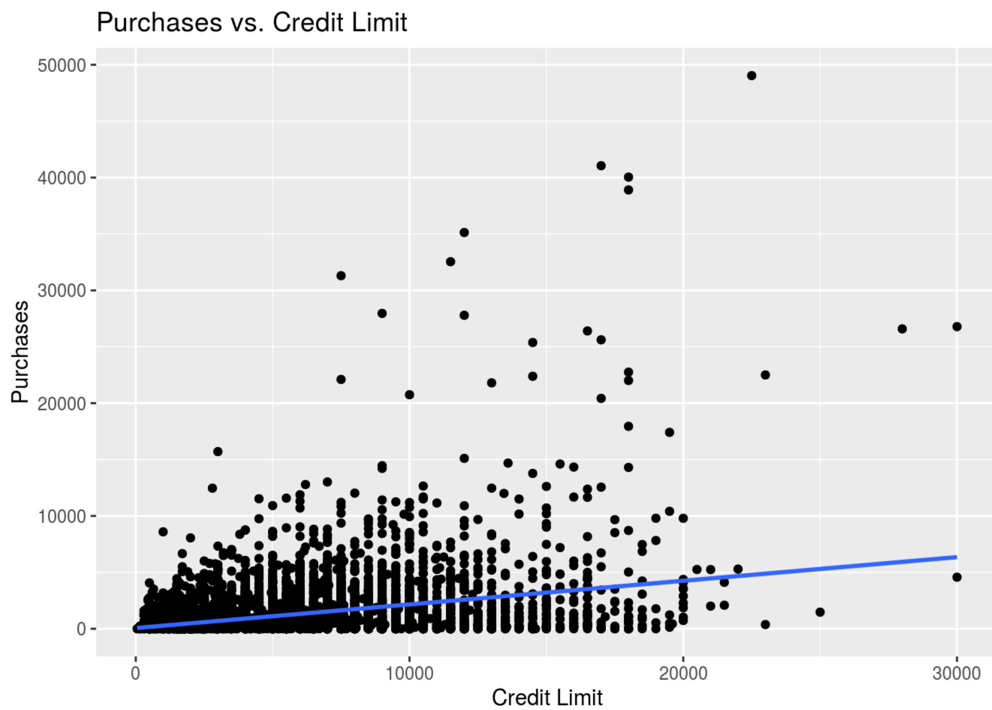
During the Data Exploration phase, we discovered that of the 9,000 rows of data, there are a handful that have much higher values that all the rest. While we are certainly interested in these customers, they can be easily found without expensive machine learning. Simple code can be used to scrape off the customers with a credit limit, or purchases, higher than some amount.

As we're hoping to come up with customer segments that are a significant portion of our customer base, perhaps containing hundreds or thousands of customers, let's look at "capping" some of these extreme values, to see if it moderates these few extreme outliers and suggests more patterns at a smaller scale.

```
df_capped <- df
```

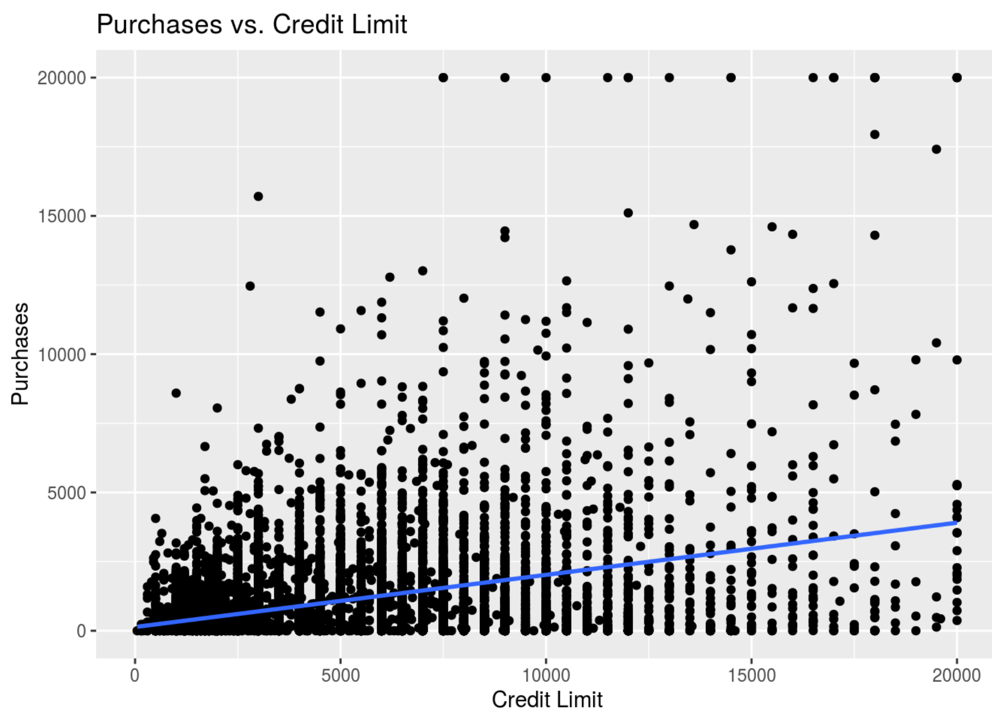
First, here's the graph that drew this to our attention:

```
df %>% ggplot(aes(y=PURCHASES, x=CREDIT_LIMIT)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="Purchases vs. Credit Limit", x="Credit Limit", y="Purchases")
```



Let’s cap those two variables at $20,000 and see if it seems a little less crazy:

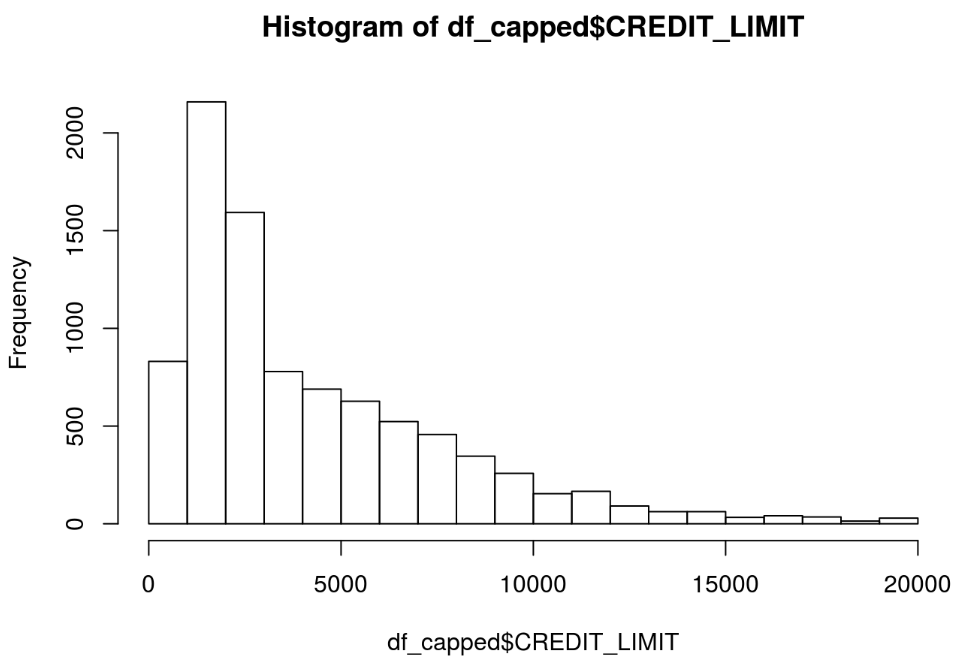
```
df_capped$CREDIT_LIMIT[df$CREDIT_LIMIT > 20000 ] <- 20000
df_capped$PURCHASES[df$PURCHASES > 20000 ] <- 20000
df_capped %>% ggplot(aes(y=PURCHASES, x=CREDIT_LIMIT)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="Purchases vs. Credit Limit", x="Credit Limit", y="Purchases")
```



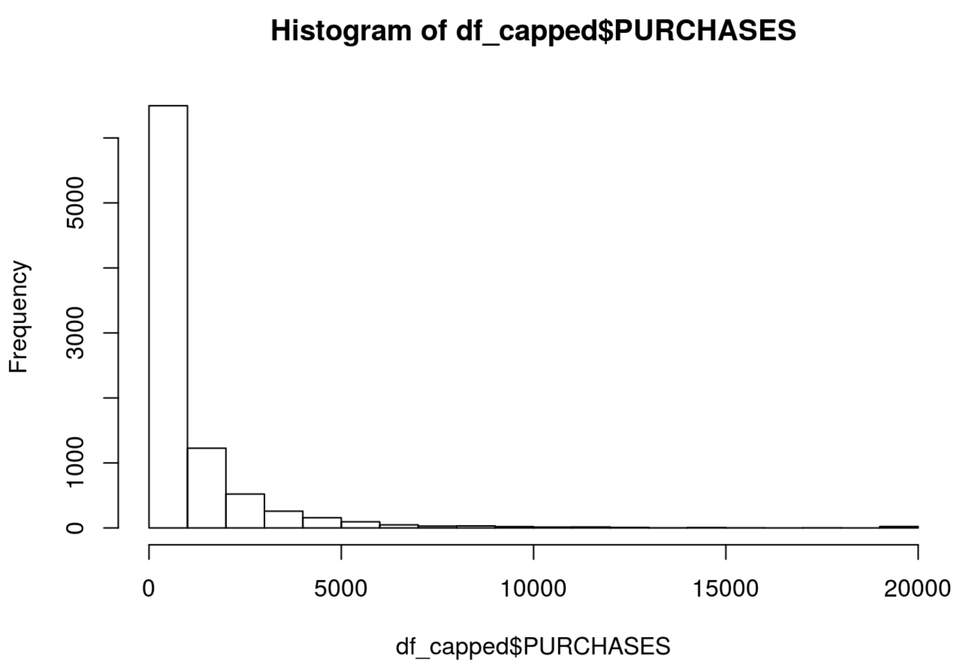
That seems much better, both for the purchases, and the credit limit.

Let’s look at one variable at a time, to ensure we haven’t created a big distortion out at the high end:

```
hist(df_capped$CREDIT_LIMIT)
```

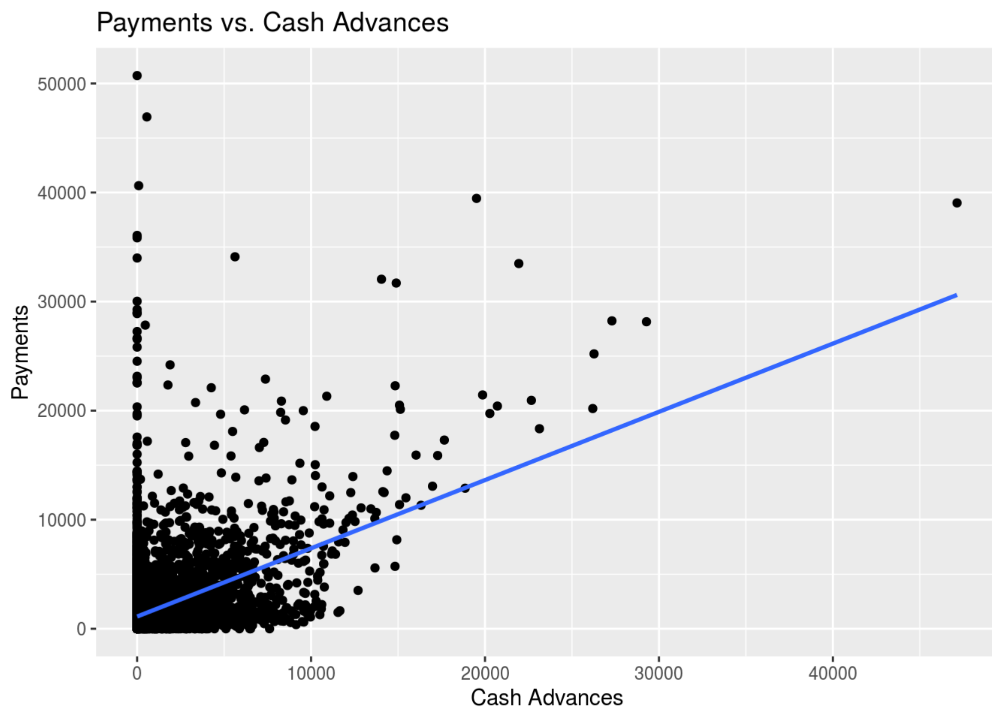



```
hist(df_capped$PURCHASES)
```



Seems okay. The distortions appear small. Let's look at the distribution of some of the other big-value columns that we're most interested in, to see how they look.

```
df %>% ggplot(aes(y=PAYMENTS, x=CASH_ADVANCE)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="Payments vs. Cash Advances", x="Cash Advances", y="Payments")
```

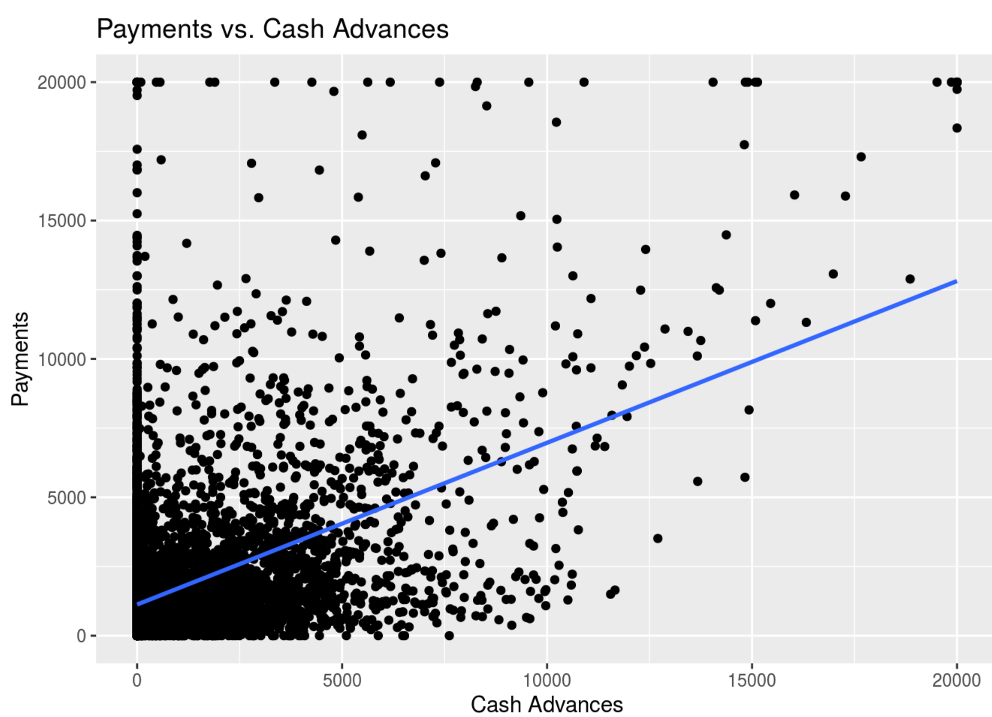


Wow, that's some serious outliers. Let's cap them.

We'll continue to use 20,000, to keep things simple, unless something suggests either that we're affecting large numbers of data points, or not budging even a handful.

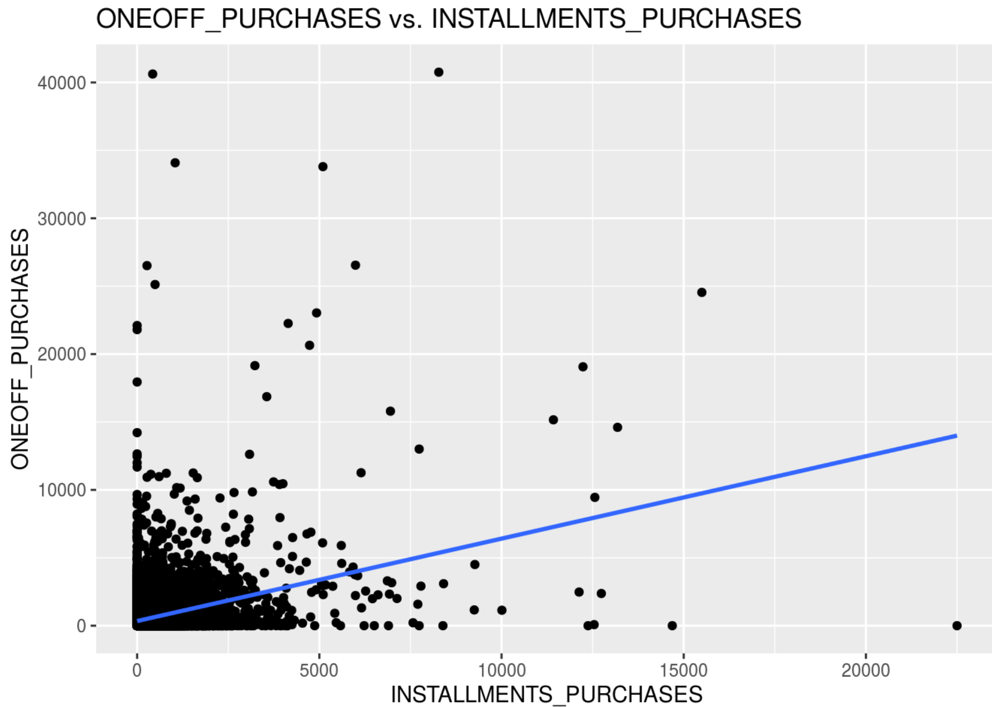
```
df_capped$PAYMENTS[df$PAYMENTS > 20000 ] <- 20000
df_capped$CASH_ADVANCE[df$CASH_ADVANCE > 20000 ] <- 20000

df_capped %>% ggplot(aes(y=PAYMENTS, x=CASH_ADVANCE)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="Payments vs. Cash Advances", x="Cash Advances", y="Payments")
```



Two more columns we're very interested in using need to be check for this:

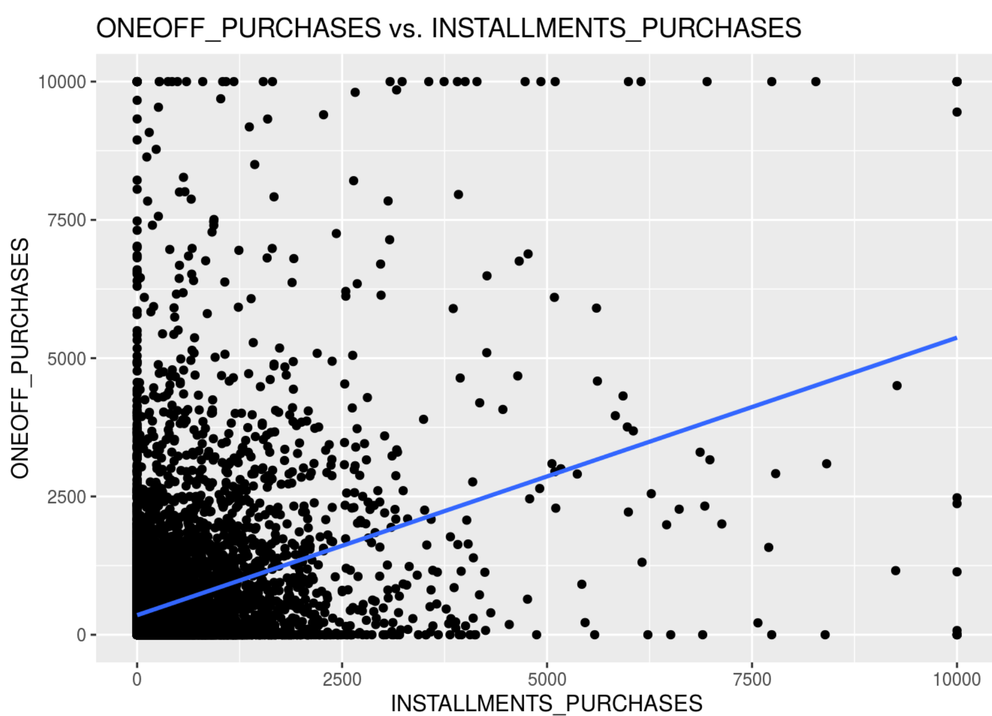
```
df %>% ggplot(aes(y=ONEOFF_PURCHASES, x=INSTALLMENTS_PURCHASES)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="ONEOFF_PURCHASES vs. INSTALLMENTS_PURCHASES", x="INSTALLMENTS_PURCHASES", y="ONEOFF_PURCHASES")
```



A very few extreme outliers. In this case, 20,000 will hardly touch these - let's cap these two at 10,000.

```
df_capped$ONEOFF_PURCHASES[df$ONEOFF_PURCHASES > 10000 ] <- 10000
df_capped$INSTALLMENTS_PURCHASES[df$INSTALLMENTS_PURCHASES > 10000 ] <- 10000

df_capped %>% ggplot(aes(y=ONEOFF_PURCHASES, x=INSTALLMENTS_PURCHASES)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title="ONEOFF_PURCHASES vs. INSTALLMENTS_PURCHASES", x="INSTALLMENTS_PURCHASES", y="ONEOFF_PURCHASES")
```



Let's look at the range of the parameters, to make sure they've wound up how we were expecting:

```
range(df_capped$BALANCE)
```

```
## [1]     0.00 19043.14
```

```
range(df_capped$BALANCE_FREQUENCY)
```

```
## [1] 0 1
```

```
range(df_capped$PURCHASES)
```

```
## [1]     0 20000
```

```
range(df_capped$ONEOFF_PURCHASES)
```

```
## [1]     0 10000
```

```
range(df_capped$INSTALLMENTS_PURCHASES)
```

```
## [1]     0 10000
```

```
range(df_capped$CASH_ADVANCE)
```

```
## [1]     0 20000
```

```
range(df_capped$PURCHASES_FREQUENCY)
```

```
## [1] 0 1
```

```
range(df_capped$ONEOFF_PURCHASES_FREQUENCY)
```

```
## [1] 0 1
```

```
range(df_capped$PURCHASES_INSTALLMENTS_FREQUENCY)
```

```
## [1] 0 1
```

```
range(df_capped$CASH_ADVANCE_FREQUENCY)
```

```
## [1] 0.0 1.5
```

```
range(df_capped$CASH_ADVANCE_TRX)
```

```
## [1]   0 123
```

```
range(df_capped$PURCHASES_TRX)
```

```
## [1]   0 358
```

```
range(df_capped$CREDIT_LIMIT)
```

```
## [1]    50 20000
```

```
range(df_capped$PAYMENTS)
```

```
## [1]     0 20000
```

```
#range(df$MINIMUM_PAYMENTS)    - doesn't work as "null".
range(df_capped$PRC_FULL_PAYMENT)
```

```
## [1] 0 1
```

```
range(df_capped$TENURE)
```

```
## [1]  6 12
```

Construct Data

We wish to try three different approaches to the inputs for our model.

1. A simple model, using only dollar values, capped to reduce the impact of outliers. (df_capped)
2. A preliminary “hack” of scaling the data, for easy interpretation of our scaled values, that will allow training and prediction based on both dollar amounts and frequencies of transactions.
3. More rigourously scaled data, which we may or may not use in our Shiny app due to time constraints and the thrill of mapping our sample input values into the distribution of the scaled data.

Let’s do the basic, hackish scaling:

```
hackish_scaled_df <- df_capped
hackish_scaled_df$BALANCE <- df_capped$BALANCE/20000
hackish_scaled_df$PURCHASES <- df_capped$PURCHASES/20000
hackish_scaled_df$ONEOFF_PURCHASES <- df_capped$ONEOFF_PURCHASES/10000
hackish_scaled_df$INSTALLMENTS_PURCHASES <- df_capped$INSTALLMENTS_PURCHASES/10000
hackish_scaled_df$CASH_ADVANCE <- df_capped$CASH_ADVANCE/20000
hackish_scaled_df$CREDIT_LIMIT <- df_capped$CREDIT_LIMIT/20000
```

```
range(hackish_scaled_df$BALANCE)
```

```
## [1] 0.0000000 0.9521569
```

```
range(hackish_scaled_df$PURCHASES)
```

```
## [1] 0 1
```

```
range(hackish_scaled_df$ONEOFF_PURCHASES)
```

```
## [1] 0 1
```

```
range(hackish_scaled_df$INSTALLMENTS_PURCHASES)
```

```
## [1] 0 1
```

```
range(hackish_scaled_df$CASH_ADVANCE)
```

```
## [1] 0 1
```

```
range(hackish_scaled_df$CREDIT_LIMIT)
```

```
## [1] 0.0025 1.0000
```

```
# And the frequency variables, which are pre-scaled, for comparison:
range(hackish_scaled_df$PURCHASES_FREQUENCY)
```

```
## [1] 0 1
```

```
range(hackish_scaled_df$ONEOFF_PURCHASES_FREQUENCY)
```

```
## [1] 0 1
```

```
range(hackish_scaled_df$PURCHASES_INSTALLMENTS_FREQUENCY)
```

```
## [1] 0 1
```

```
range(hackish_scaled_df$CASH_ADVANCE_FREQUENCY)
```

```
## [1] 0.0 1.5
```

Now we will do a formal scaling. This can be compared to the hackish one above, both in terms of accuracy of clustering and prediction, and the ease with which we can map new, arbitrary samples into its distribution for future predictions.

```
scaled_df <- df_capped

scaled_df$BALANCE <-scale(scaled_df$BALANCE)[, 1]
scaled_df$BALANCE_FREQUENCY <-scale(df$BALANCE_FREQUENCY)[, 1]
scaled_df$PURCHASES <- scale(df$PURCHASES)[, 1]
scaled_df$ONEOFF_PURCHASES <- scale(df$ONEOFF_PURCHASES)[, 1]
scaled_df$INSTALLMENTS_PURCHASES<- scale(df$INSTALLMENTS_PURCHASES)[, 1]
scaled_df$CASH_ADVANCE<- scale(df$CASH_ADVANCE)[, 1]
scaled_df$PURCHASES_FREQUENCY<- scale(df$PURCHASES_FREQUENCY)[, 1]
scaled_df$ONEOFF_PURCHASES_FREQUENCY <- scale(df$ONEOFF_PURCHASES_FREQUENCY)[, 1]
scaled_df$PURCHASES_INSTALLMENTS_FREQUENCY<- scale(df$PURCHASES_INSTALLMENTS_FREQUENCY)[, 1]
scaled_df$CASH_ADVANCE_FREQUENCY <- scale(df$CASH_ADVANCE_FREQUENCY)[, 1]
scaled_df$CASH_ADVANCE_TRX <- scale(df$CASH_ADVANCE_TRX)[, 1]
scaled_df$PURCHASES_TRX   <- scale(df$PURCHASES_TRX)[, 1]
scaled_df$CREDIT_LIMIT<- scale(df$CREDIT_LIMIT)[, 1]
scaled_df$PAYMENTS  <- scale(df$PAYMENTS)[, 1]
# scaled_df$MINIMUM_PAYMENTS <- scale(df$MINIMUM_PAYMENTS)[, 1]      # We've nullified this column.
scaled_df$PRC_FULL_PAYMENT <- scale(df$PRC_FULL_PAYMENT)[, 1]
scaled_df$TENURE<- scale(df$TENURE)[, 1]
str(scaled_df)
```

```
## 'data.frame':    8949 obs. of  17 variables:
##  $ CUST_ID                         : chr  "C10001" "C10002" "C10003" "C10004" ...
##  $ BALANCE                         : num  -0.732 0.787 0.447 0.049 -0.359 ...
##  $ BALANCE_FREQUENCY               : num  -0.25 0.134 0.518 -1.018 0.518 ...
##  $ PURCHASES                       : num  -0.425 -0.47 -0.108 0.232 -0.462 ...
##  $ ONEOFF_PURCHASES                : num  -0.357 -0.357 0.109 0.546 -0.347 ...
##  $ INSTALLMENTS_PURCHASES          : num  -0.349 -0.455 -0.455 -0.455 -0.455 ...
##  $ CASH_ADVANCE                    : num  -0.467 2.605 -0.467 -0.369 -0.467 ...
##  $ PURCHASES_FREQUENCY             : num  -0.807 -1.222 1.27 -1.014 -1.014 ...
##  $ ONEOFF_PURCHASES_FREQUENCY      : num  -0.679 -0.679 2.673 -0.399 -0.399 ...
##  $ PURCHASES_INSTALLMENTS_FREQUENCY: num  -0.707 -0.917 -0.917 -0.917 -0.917 ...
##  $ CASH_ADVANCE_FREQUENCY          : num  -0.675 0.574 -0.675 -0.259 -0.675 ...
##  $ CASH_ADVANCE_TRX                : num  -0.476 0.11 -0.476 -0.33 -0.476 ...
##  $ PURCHASES_TRX                   : num  -0.511 -0.592 -0.109 -0.552 -0.552 ...
##  $ CREDIT_LIMIT                    : num  -0.96 0.689 0.826 0.826 -0.905 ...
##  $ PAYMENTS                        : num  -0.529 0.819 -0.384 -0.599 -0.364 ...
##  $ PRC_FULL_PAYMENT                : num  -0.526 0.234 -0.526 -0.526 -0.526 ...
##  $ TENURE                          : num  0.361 0.361 0.361 0.361 0.361 ...
##  - attr(*, "na.action")= 'omit' Named int 5204
##   ..- attr(*, "names")= chr "5204"
```

```
range(scaled_df$BALANCE)
```

```
## [1] -0.751662  8.396726
```

```
range(scaled_df$PURCHASES)
```

```
## [1] -0.4695577 22.4812220
```

```
range(scaled_df$ONEOFF_PURCHASES)
```

```
## [1] -0.3569366 24.1984941
```

```
range(scaled_df$INSTALLMENTS_PURCHASES)
```

```
## [1] -0.4545815 24.4243905
```

```
range(scaled_df$CASH_ADVANCE)
```

```
## [1] -0.4667793 22.0087908
```

```
range(scaled_df$CREDIT_LIMIT)
```

```
## [1] -1.2214  7.0093
```

```
# And the frequency variables, which are pre-scaled, for comparison:
range(scaled_df$PURCHASES_FREQUENCY)
```

```
## [1] -1.221860  1.269671
```

```
range(scaled_df$ONEOFF_PURCHASES_FREQUENCY)
```

```
## [1] -0.6786783  2.6731453
```

```
range(scaled_df$PURCHASES_INSTALLMENTS_FREQUENCY)
```

```
## [1] -0.9170383  1.5989932
```

```
range(scaled_df$CASH_ADVANCE_FREQUENCY)
```

```
## [1] -0.6752567  6.8197856
```

Oh, goodie! I'm having trouble interpreting that already. If we have trouble getting to the bottom of all that - we'll stick with the basic, student-model hackish version for our predictions.

MODELING

Select Modeling Technique

We have decided to try a few different models based on the K-means algorithm. Our models will differ by the inputs they take, and how those inputs are cleaned and scaled.

This will allow us to compare the resulting models easily, and determine which data preparation techniques are best suited to the knowledge discovery we seek.

Generate Test Design

Model #1 - clean data with capped outliers.

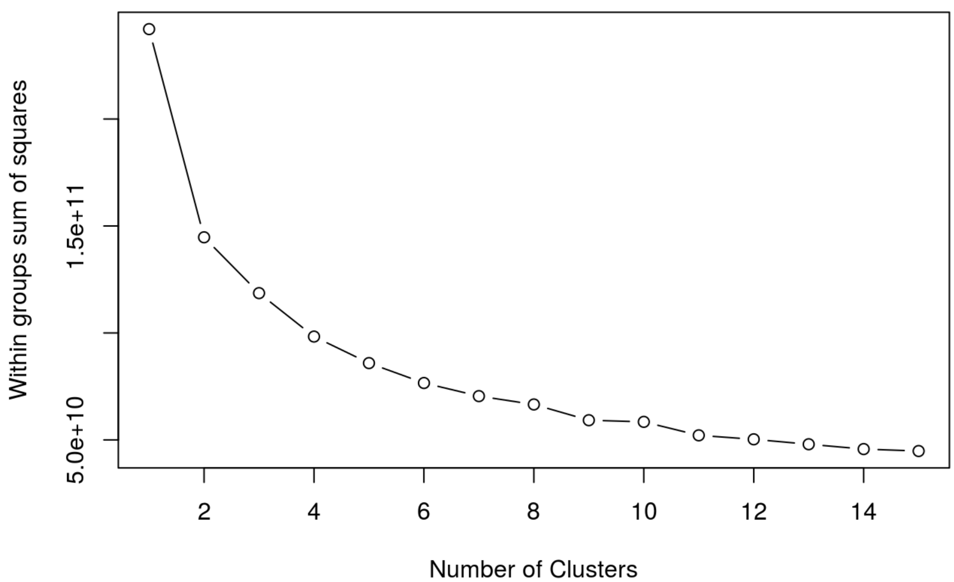
The best thing about this model was that we were able to generate it quickly, and use it to build up the first implementations of our Shiny App. We do have high hopes that our more advanced models will yeild better results. We shall check that assumption!

Determine number of clusters

Run tests on various numbers of clusters to look for an “elbow” which will suggest how many useful clusters we can hope to get from this data.

```
cluster_data_1 <- df_capped[c(2,4:7, 14)]

wss <- (nrow(cluster_data_1)-1)*sum(apply(cluster_data_1,2,var))
for (i in 2:15) wss[i] <- sum(kmeans(cluster_data_1,
   centers=i)$withinss)
plot(1:15, wss, type="b", xlab="Number of Clusters",
  ylab="Within groups sum of squares")
```



In terms of how many clusters we should choose, these graphs could be suggesting anywhere from 4 to 7, depending on how we read them.

Here’s one with 6 clusters:

```
# K-Means Cluster Analysis
fit <- kmeans(cluster_data_1, 6) # 6 cluster solution
# get cluster means
aggregate(cluster_data_1,by=list(fit$cluster),FUN=mean)
```

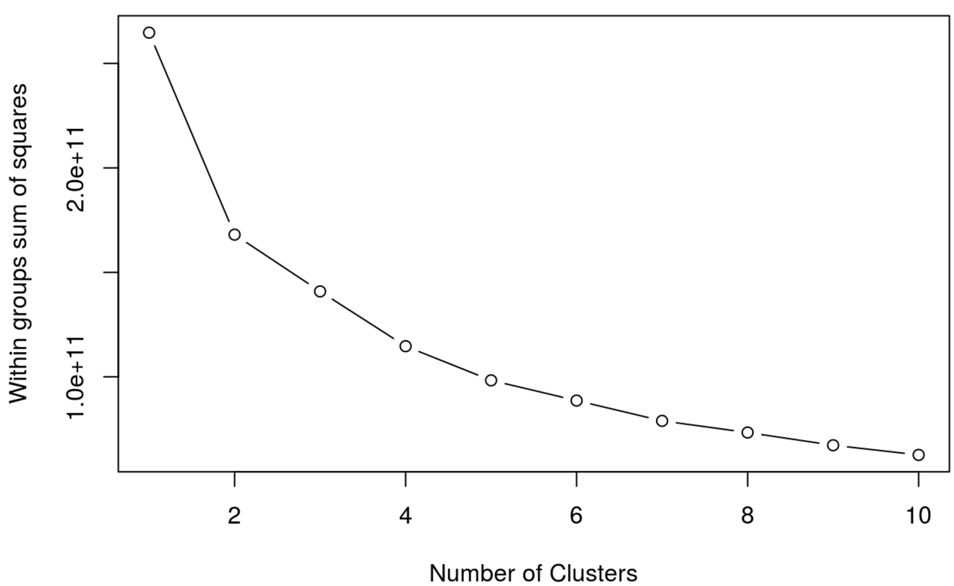
```
##   Group.1   BALANCE  PURCHASES ONEOFF_PURCHASES INSTALLMENTS_PURCHASES
## 1       1 5213.1376 10809.0679        6368.3055              3373.6342
## 2       2 1770.9971   610.2233         320.7073               289.9409
## 3       3 1927.6654  1560.1893         982.0957               578.1333
## 4       4  759.5318   461.3438         211.7375               249.8228
## 5       5 1551.4645  3981.2722        2616.4733              1365.9083
## 6       6 6361.0780   674.6616         402.3061               272.4666
##   CASH_ADVANCE CREDIT_LIMIT
## 1     863.8139    12483.775
## 2    1067.2790     5997.419
## 3     452.7800    12120.541
## 4     462.3361     2031.427
## 5     357.9748     5681.709
## 6    6004.7943     9772.893
```

```r
# append cluster assignment
cluster_data_1 <- data.frame(cluster_data_1, fit$cluster)
## Assess Model
```

Sure, but how does this look WITHOUT all that capping?

```r
df0 <- read.csv("credit-card-cust-behav-data.csv", stringsAsFactors = FALSE, header = TRUE, encoding  = "UTF-8")
df0 <- na.omit(df0)
cld2 <- df0[c(2,4:7,14)]

wss <- (nrow(cld2)-1)*sum(apply(cld2,2,var))
for (i in 2:10) wss[i] <- sum(kmeans(cld2,
   centers=i)$withinss)
plot(1:10, wss, type="b", xlab="Number of Clusters",
  ylab="Within groups sum of squares")
```



And a clustering into 6, with the uncapped data, yeilds almost identical results.

```r
# K-Means Cluster Analysis
fit <- kmeans(cld2, 6) # 6 cluster solution
# get cluster means
aggregate(cld2,by=list(fit$cluster),FUN=mean)
```

```
##   Group.1   BALANCE  PURCHASES ONEOFF_PURCHASES INSTALLMENTS_PURCHASES
## 1       1  794.8300   529.7977         257.4535               272.6593
## 2       2 6936.7833   710.1741         417.8344               292.5266
## 3       3 5390.3896 27690.8658       21422.8846              6267.9812
## 4       4  839.2454  1547.2875         938.6637               609.2029
## 5       5 3847.4119   573.6719         314.8448               258.8591
## 6       6 3580.8141  4469.6711        2936.4292              1533.2903
##   CASH_ADVANCE CREDIT_LIMIT
## 1     448.7942     2095.824
## 2    7625.3394    11092.598
## 3     929.6892    16333.333
## 4     184.9120     6882.282
## 5    2799.0910     6135.212
## 6     506.1849    12864.396
```

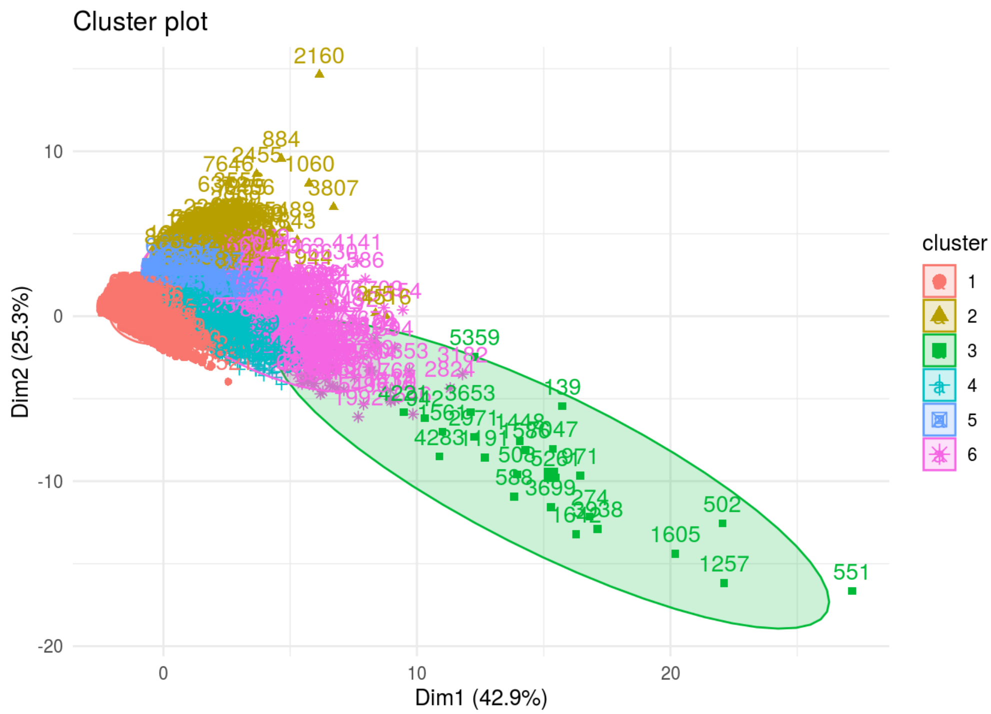
```r
# append cluster assignment
cld2 <- data.frame(cld2, fit$cluster)
## Assess Model
```

This 6-cluster solution clearly nets us 6 groups we can identify - Big Spenders, Cash Advancers, Smaller Cash Advancers, Balance Carriers, Balance Payers, and Light Users.

It is interesting that whether or not we cap, with 6 clusters, we get two groups of “cash advancers”, which differ only in terms of credit limit.

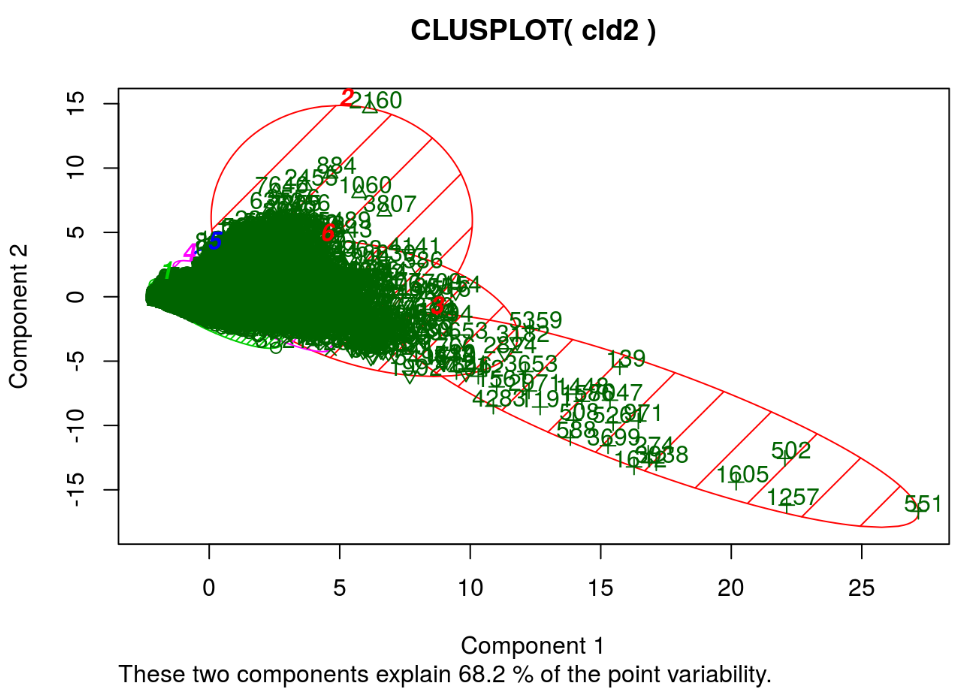
Let’s have a look at a cluster plot for that.

```
fviz_cluster(fit,cld2, ellipse.type = "norm")+ theme_minimal()
```



And now a cluster plot with a more primitive library, just to be sure we might not be missing patterns with this colourful approach.

```
clusplot(cld2, fit$cluster, color=TRUE, shade=TRUE,
         labels=2, lines=0)
```



While not all of the groups stand out clearly in this plot, we can easily line up two of them with the numerical data above - the Big Spenders, and the Balance Carriers.

This model is okay for a start, but we're missing some good data still. We would like more information about how frequently a customer performs various transaction types, in addition to how much money they spend. For example, Big Spenders could be the type to do massive one-off purchases - or they may be the type to do many small purchases every week.

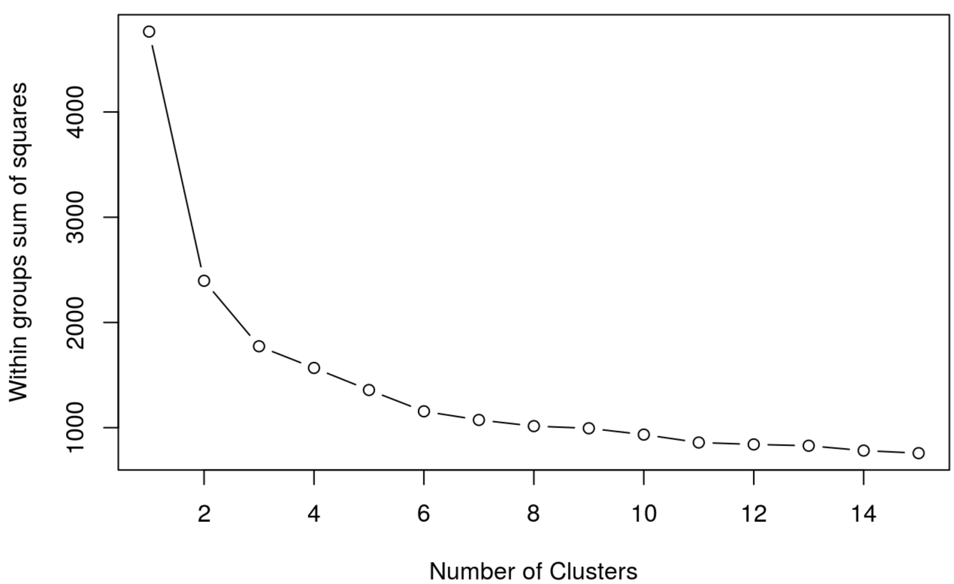
This data exists in the dataset, but we would need to employ some Feature Scaling for our model to be able to learn from it, and not just the very large dollar amounts.

For our first attempt, we will scale the larger dollar-denominated data by simply dividing it by the amount we capped it to earlier, resulting in a range of values from zero to one.

Model 2 - With hackishly scaled data

```
cluster_data_2 <- hackish_scaled_df[c(2,4:11, 14)]

wss <- (nrow(cluster_data_2)-1)*sum(apply(cluster_data_2,2,var))
for (i in 2:15) wss[i] <- sum(kmeans(cluster_data_2,
   centers=i)$withinss)
plot(1:15, wss, type="b", xlab="Number of Clusters",
  ylab="Within groups sum of squares")
```



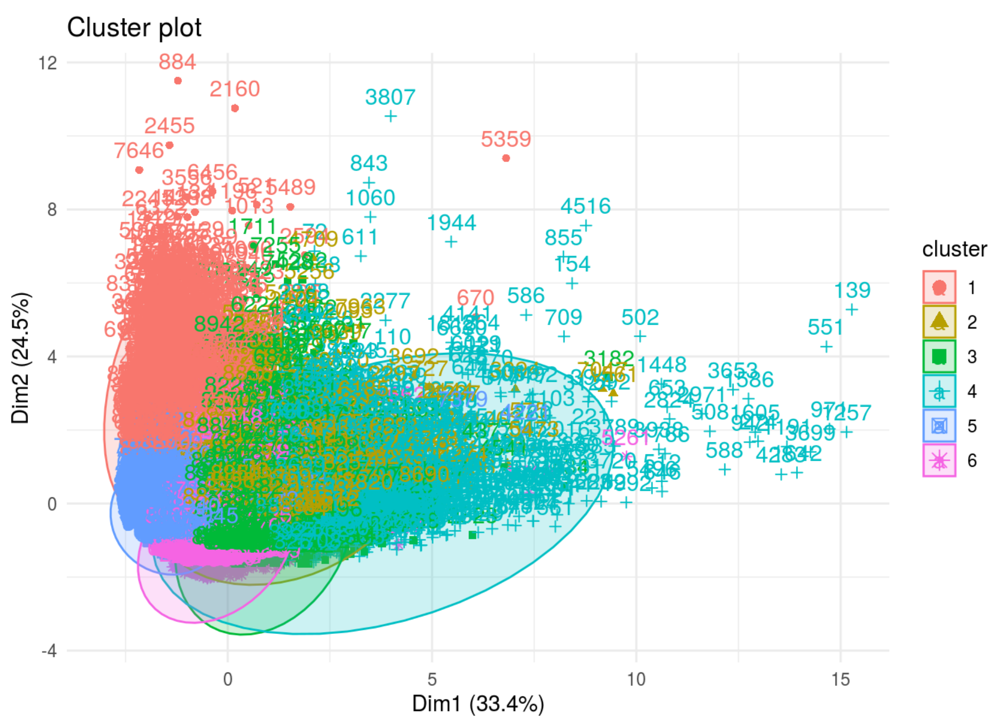
We can see right away that our Sum of Squares is much, much lower than the previous example - though this may not tell us much, given that we have dramatically change the variance of most of our features.

```
# K-Means Cluster Analysis
fit_2 <- kmeans(cluster_data_2, 6) # 6 cluster solution
# get cluster means
aggregate(cluster_data_2,by=list(fit_2$cluster),FUN=mean)
```

```
##   Group.1    BALANCE   PURCHASES ONEOFF_PURCHASES INSTALLMENTS_PURCHASES
## 1       1 0.19512014 0.009984959       0.01586191            0.002919724
## 2       2 0.07382462 0.090582823       0.16521636            0.012295215
## 3       3 0.05433250 0.051144899       0.01926841            0.082717717
## 4       4 0.12200038 0.211244629       0.26018529            0.151636150
## 5       5 0.05384008 0.011814957       0.02111173            0.002193311
## 6       6 0.05013546 0.034025443       0.02419485            0.042988239
##   CASH_ADVANCE PURCHASES_FREQUENCY ONEOFF_PURCHASES_FREQUENCY
## 1   0.19885101          0.08473351                 0.05893067
## 2   0.02637644          0.78297722                 0.75179043
## 3   0.02520970          0.95370860                 0.07994499
## 4   0.03875876          0.97007256                 0.77365825
## 5   0.03230485          0.09768933                 0.07382961
## 6   0.01950801          0.54666717                 0.09492730
##   PURCHASES_INSTALLMENTS_FREQUENCY CASH_ADVANCE_FREQUENCY CREDIT_LIMIT
## 1                       0.02598998             0.49208639    0.3356844
## 2                       0.11038313             0.08302058    0.2782671
## 3                       0.91403511             0.06851220    0.1836167
## 4                       0.84570220             0.10124346    0.3904970
## 5                       0.02312163             0.10783468    0.1624301
## 6                       0.45271620             0.06290382    0.1977867
```

```
# append cluster assignment
cluster_data_2 <- data.frame(cluster_data_2, fit_2$cluster)
## Assess Model
```

```
fviz_cluster(fit_2,cluster_data_2, ellipse.type = "norm")+ theme_minimal()
```



In trying to interpret these clusters, we can see that we have less interpretable results now. The average Purchases of the Big Spenders is around 0.19 - as is the average Cash Advances of the Big Cash Advancers. While we know we can restore these to dollar amounts by multiplying by our scale factor - 20,000 - as we go deeper and deeper into the features, while we can tell which are highest and which are lowest, it is more and more difficult to interpret what these averages refer to.

And Let's try that with 8 clusters, so we can compare it to the six:

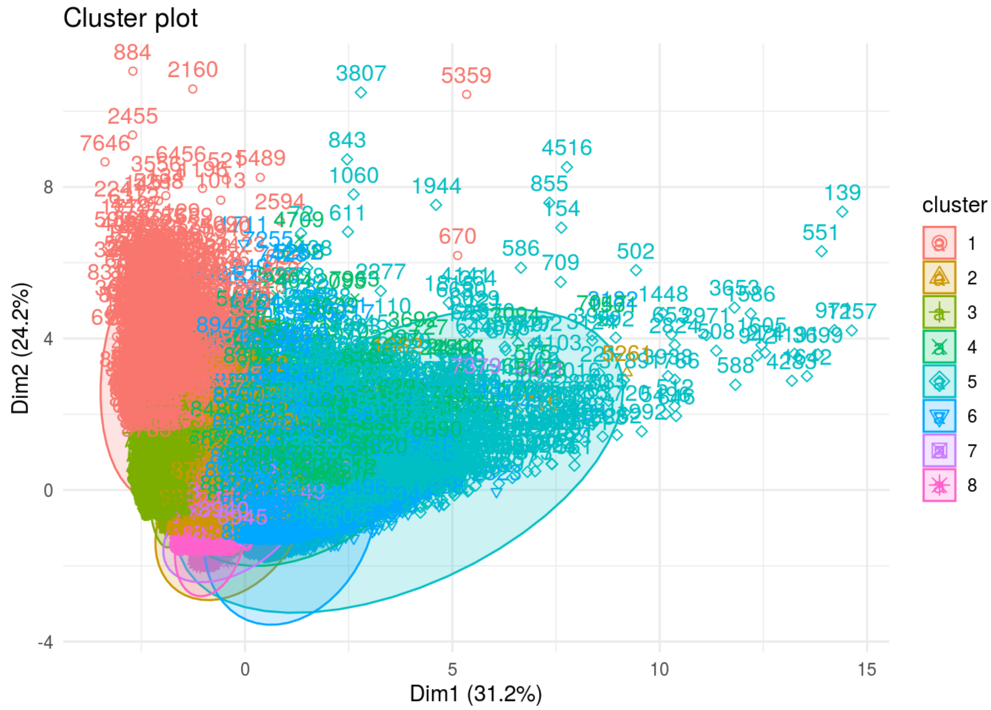
```
# K-Means Cluster Analysis
fit_2_8 <- kmeans(cluster_data_2, 8) # 8 cluster solution
# get cluster means
aggregate(cluster_data_2,by=list(fit_2_8$cluster),FUN=mean)
```

```
##   Group.1    BALANCE   PURCHASES ONEOFF_PURCHASES INSTALLMENTS_PURCHASES
## 1       1 0.19512014 0.009984959      0.015861905           0.0029197244
## 2       2 0.05013546 0.034025443      0.024194849           0.0429882389
## 3       3 0.06426810 0.001351675      0.002440061           0.0002646231
## 4       4 0.07382462 0.090582823      0.165216356           0.0122952151
## 5       5 0.12200038 0.211244629      0.260185286           0.1516361500
## 6       6 0.05433250 0.051144899      0.019268408           0.0827177170
## 7       7 0.05021738 0.043381816      0.082095911           0.0023013502
## 8       8 0.04069421 0.014248240      0.023661845           0.0048411014
##   CASH_ADVANCE PURCHASES_FREQUENCY ONEOFF_PURCHASES_FREQUENCY
## 1   0.19885101          0.08473351                 0.05893067
## 2   0.01950801          0.54666717                 0.09492730
## 3   0.05276792          0.01682928                 0.01335546
## 4   0.02637644          0.78297722                 0.75179043
## 5   0.03875876          0.97007256                 0.77365825
## 6   0.02520970          0.95370860                 0.07994499
## 7   0.01597445          0.30961785                 0.29388567
## 8   0.01005986          0.12882496                 0.07340872
##   PURCHASES_INSTALLMENTS_FREQUENCY CASH_ADVANCE_FREQUENCY CREDIT_LIMIT
## 1                      0.025989980            0.49208639    0.3356844
## 2                      0.452716201            0.06290382    0.1977867
## 3                      0.003061081            0.18067547    0.1182494
## 4                      0.110383128            0.08302058    0.2782671
## 5                      0.845702200            0.10124346    0.3904970
## 6                      0.914035108            0.06851220    0.1836167
## 7                      0.022218546            0.06882853    0.1757488
## 8                      0.051442231            0.02128604    0.2189071
##   fit_2.cluster
## 1             1
## 2             6
## 3             5
## 4             2
## 5             4
## 6             3
## 7             5
## 8             5
```

```
# append cluster assignment
cluster_data_2 <- data.frame(cluster_data_2, fit_2_8$cluster)
## Save Model
save(fit_2_8 , file = 'CreditCardBehaviour8Clusters.rda')
```

As suggested by the elbow chart, we aren’t really getting more useful clusters, by going in this direction.

```
fviz_cluster(fit_2_8,cluster_data_2, ellipse.type = "norm")+ theme_minimal()
```



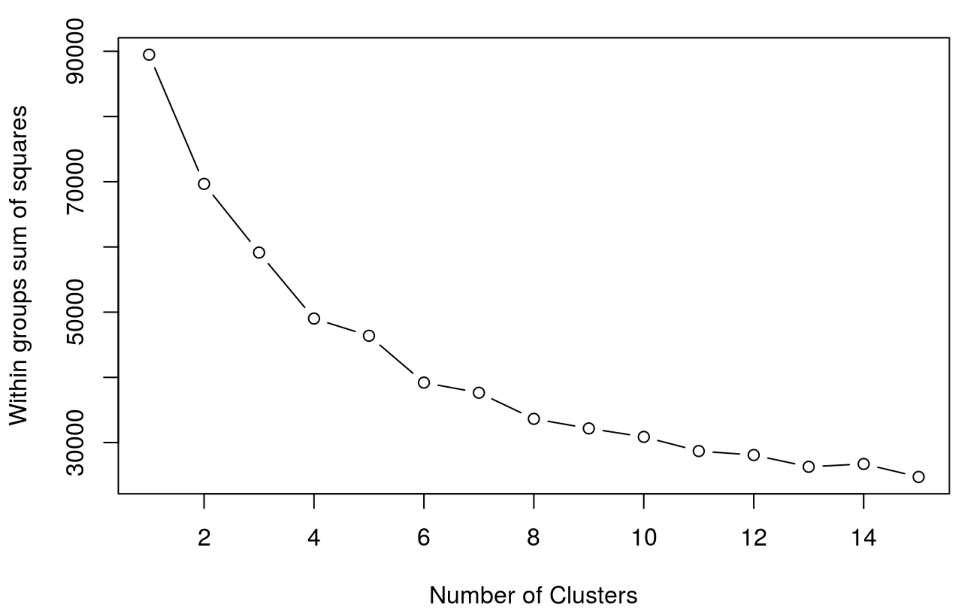
Model 3 - With formally scaled data

For our third, and final, model, we will use classical scaling with the scale() function. While this should lead to better results, it also increases the challenge of interpreting the clusters, as well as the challenge of making predictions based on raw data, when our model has been trained using scaled data. How do we scale the raw data, into the same distribution as the scaled data our model was trained on? We explore that in order to use models like these in our Shiny App.

So here's a K means clustering, with our properly scaled data:

```
cluster_data_3 <- scaled_df[c(2,4:11, 14)]

wss <- (nrow(cluster_data_3)-1)*sum(apply(cluster_data_3,2,var))
for (i in 2:15) wss[i] <- sum(kmeans(cluster_data_3,
   centers=i)$withinss)
plot(1:15, wss, type="b", xlab="Number of Clusters",
  ylab="Within groups sum of squares")
```

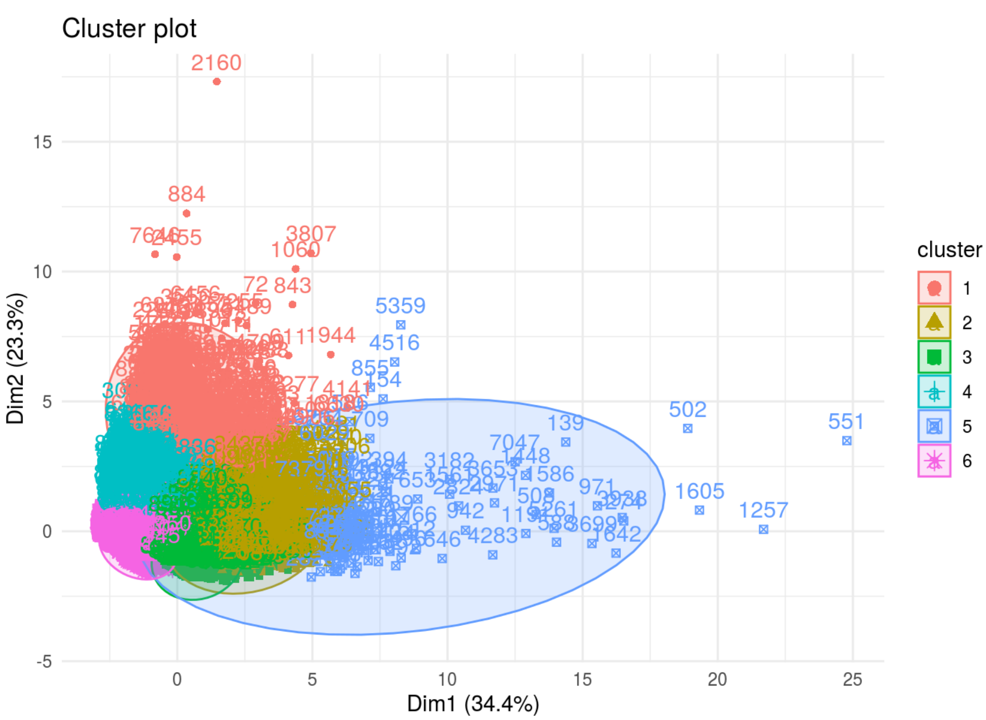


```
# K-Means Cluster Analysis
fit_3 <- kmeans(cluster_data_3, 6) # 6 cluster solution
# get cluster means
aggregate(cluster_data_3,by=list(fit_3$cluster),FUN=mean)
```

```
##   Group.1     BALANCE   PURCHASES ONEOFF_PURCHASES INSTALLMENTS_PURCHASES
## 1       1  2.31693977 -0.15066335       -0.1177826             -0.1399840
## 2       2  0.08632681  0.81083582        0.8489361              0.3571823
## 3       3 -0.38579833 -0.05100305       -0.2564292              0.3507857
## 4       4  0.32020707 -0.40590951       -0.2947753             -0.4182716
## 5       5  1.44920698  6.29886631        5.3772644              5.0118097
## 6       6 -0.37841659 -0.31725325       -0.2042073             -0.3748722
##   CASH_ADVANCE PURCHASES_FREQUENCY ONEOFF_PURCHASES_FREQUENCY
## 1    2.4389530          -0.3444347                 -0.1602980
## 2   -0.2827983           1.0388032                  1.8849275
## 3   -0.3539232           0.9840109                 -0.4061849
## 4    0.5769575          -0.9851256                 -0.4903844
## 5   -0.0267621           1.0851564                  1.7680189
## 6   -0.3298670          -0.7219366                 -0.3244293
##   PURCHASES_INSTALLMENTS_FREQUENCY CASH_ADVANCE_FREQUENCY CREDIT_LIMIT
## 1                       -0.2870573              1.7257380   1.52748837
## 2                        0.4116095             -0.3358930   0.62756559
## 3                        1.1809963             -0.4498844  -0.31627166
## 4                       -0.8177847              1.2206201  -0.08751552
## 5                        1.0580248             -0.3256453   2.11280216
## 6                       -0.6820577             -0.3774079  -0.35665152
```

```
# append cluster assignment
cluster_data_3 <- data.frame(cluster_data_3, fit_3$cluster)
## Assess Model
```

```
fviz_cluster(fit_3,cluster_data_3, ellipse.type = "norm")+ theme_minimal()
```



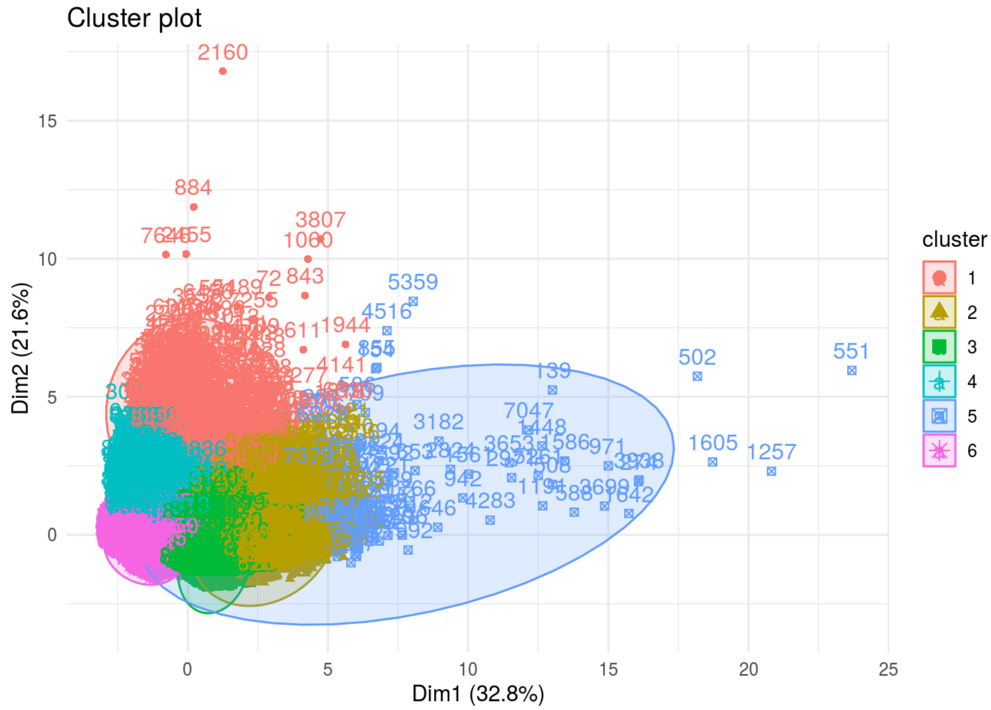
And Let's try that with 11 clusters, to see if we can make use of that:

```
# K-Means Cluster Analysis
fit_3_11 <- kmeans(cluster_data_3, 11) # 11 cluster solution
# get cluster means
aggregate(cluster_data_3,by=list(fit_3_11$cluster),FUN=mean)
```

```
##    Group.1     BALANCE  PURCHASES ONEOFF_PURCHASES INSTALLMENTS_PURCHASES
## 1        1  1.70291649 13.1473920      14.11411571             5.15614584
## 2        2 -0.24206498  0.3895410       0.52799419            -0.04910658
## 3        3  1.71168897 -0.1718793      -0.11610822            -0.19330783
## 4        4 -0.48014593 -0.1674507      -0.28269191             0.12373791
## 5        5  0.80744661  2.4056091       2.58042920             0.94694857
## 6        6  0.85872411  0.2362848       0.15228030             0.27841853
## 7        7  0.32593832 -0.4120702      -0.29888938            -0.42527544
## 8        8 -0.37841659 -0.3172533      -0.20420726            -0.37487220
## 9        9  1.57904654  4.5371499       1.88301786             7.27475468
## 10      10  2.78780277 -0.2449533      -0.17836539            -0.25147392
## 11      11  0.01121353  0.7312179       0.07373337             1.59194270
##    CASH_ADVANCE PURCHASES_FREQUENCY ONEOFF_PURCHASES_FREQUENCY
## 1    0.06516462           0.9976785                  2.1368537
## 2   -0.38179473           1.0011356                  2.0495372
## 3    4.90858044          -0.6020396                 -0.2866216
## 4   -0.39274449           0.9607700                 -0.4548290
## 5   -0.24081999           1.0997288                  2.0648398
## 6    0.99908191           1.0142880                  0.6157547
## 7    0.57098337          -1.0161626                 -0.5057960
## 8   -0.32986702          -0.7219366                 -0.3244293
## 9   -0.05186813           1.1130394                  1.2908078
## 10   1.46541981          -0.5448773                 -0.2793797
## 11  -0.35909273           1.1042146                  0.1599011
##    PURCHASES_INSTALLMENTS_FREQUENCY CASH_ADVANCE_FREQUENCY CREDIT_LIMIT
## 1                        0.75612251             -0.4462417   3.18250536
## 2                        0.06928503             -0.4911029   0.34547677
## 3                       -0.49919533              2.1948262   1.40247533
## 4                        1.15189985             -0.5062689  -0.48840638
## 5                        0.82753338             -0.3677699   1.36714660
## 6                        0.94941315              1.7290689   0.50465881
## 7                       -0.84137485              1.1935295  -0.08829445
## 8                       -0.68205766             -0.3774079  -0.35665152
## 9                        1.34852700             -0.2807808   1.95921841
## 10                      -0.48548879              1.3816522   1.70264389
## 11                       1.33485746             -0.5223970   0.66376508
##    fit_3.cluster
## 1       5.000000
## 2       2.003509
## 3       1.000000
## 4       3.000000
## 5       2.329897
## 6       2.176056
## 7       4.000000
## 8       6.000000
## 9       4.842105
## 10      1.014205
## 11      2.690196
```

```
# append cluster assignment
cluster_data_3 <- data.frame(cluster_data_3, fit_3_11$cluster)
## Save Model
#save(fit_3_11 , file = 'CreditCardBehaviour11Clusters.rda')
```

```
fviz_cluster(fit_3,cluster_data_3, ellipse.type = "norm")+ theme_minimal()
```



As suggested by the “elbow” graph, going to a higher numbers of clusters doesn’t appear to yield any more useful information.

CONCLUSIONS

We were able to cluster the data. We were able to create clusters that gave us better insight into customer behaviour by scaling the bigger features and including transaction frequencies. We were able to gain insights into the spending behaviours that most define the market segments we discovered.

RECOMMENDATIONS

Our intention had been to use a model that was trained on scaled data in our Shiny App. (https://iwis-zhou.shinyapps.io/CreditCardDataset/ (https://iwis-zhou.shinyapps.io/CreditCardDataset/))

To do this, we would have needed to transform the input fields on the Shiny user interface into the scaled distribution that we trained the model on.

Also, we would have liked to have relayed our interpretation of the clusters (Big Spenders, Frequent Impulse Shoppers, Cash Advancers, etc.) as that would make it interpretable to a person.

Future versions would be able to tell us which cluster, what the probability of that sample being in that cluster are, and provide a more detailed description of the spending habits that dominate that market segment.