Exam

Andreas, Kristoffer, Mikkel & Simon

22/9/2021

```
library(tidyverse)

library(lubridate)

library(magrittr)

library(FactoMineR)

library(factoextra)

library(uwot)

library(GGally)

library(rsample)

library(ggridges)

library(xgboost)

library(recipes)

library(parsnip)

library(glmnet)

library(tidymodels)

library(skimr)

library(VIM)

library(visdat)

library(ggmap)

library(ranger)

library(vip)
```

Loading the data

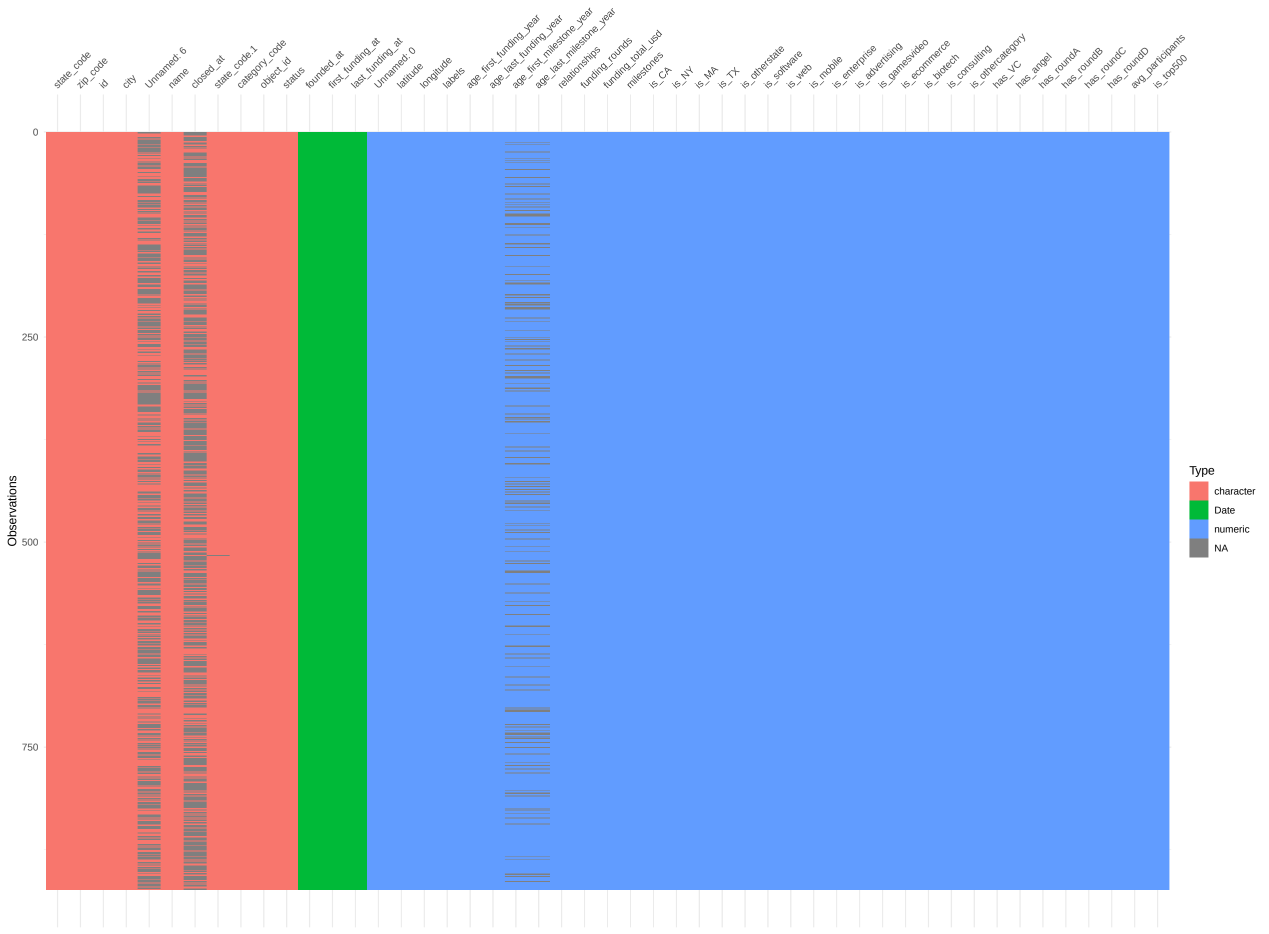
```r
data_start <- read_csv("startup data.csv",
col_types = cols(founded_at = col_date(format = "%m/%d/%Y"),
first_funding_at = col_date(format = "%m/%d/%Y"),
last_funding_at = col_date(format = "%m/%d/%Y")))
```

Data cleaning / EDA

Format data

First we look at what type of data we are dealing with.

```r
vis_dat(data_start)
```



We also show the amount of NA's in text, so that we can see how many NA's each variable contains.

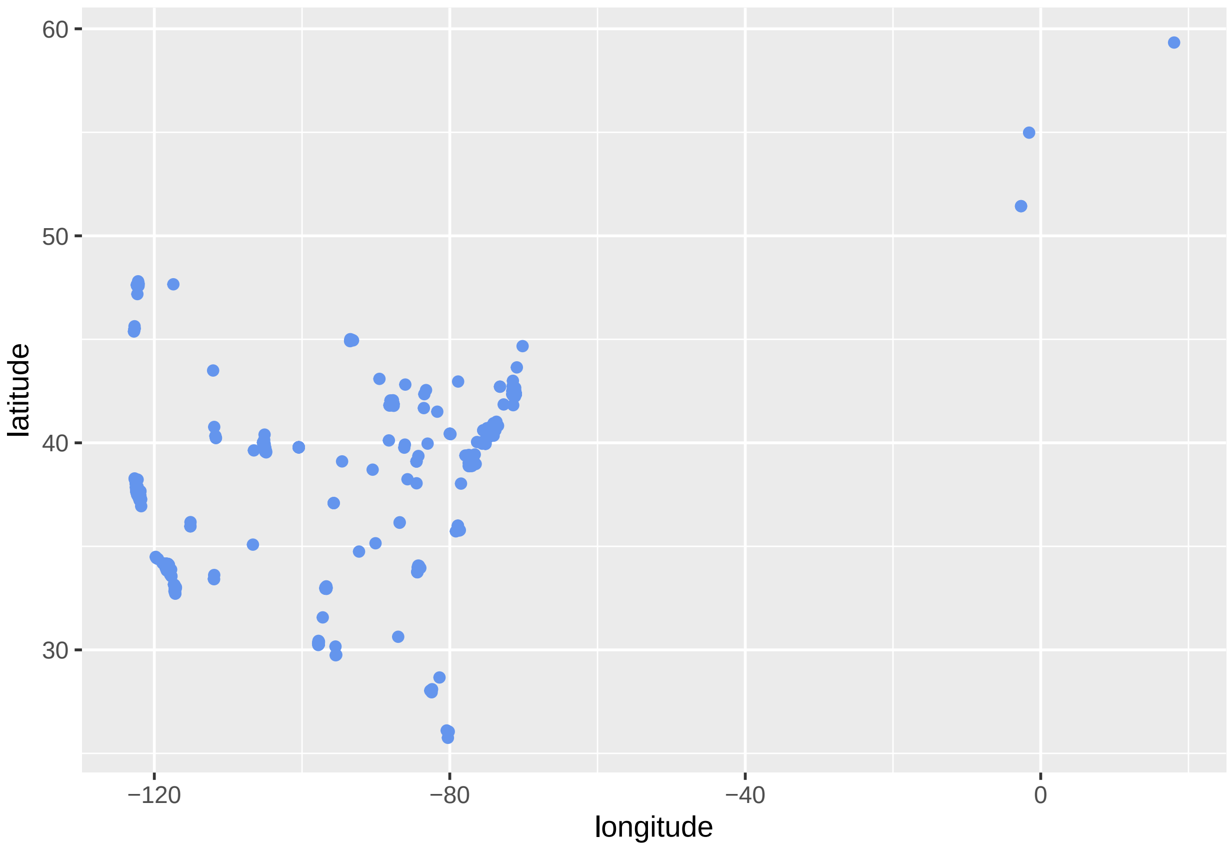
```r
is.na(data_start) %>% colSums()
```

```
##             Unnamed: 0             state_code               latitude
##                      0                      0                      0
##              longitude               zip_code                     id
##                      0                      0                      0
##                   city            Unnamed: 6                   name
##                      0                    493                      0
##                   labels               founded_at                closed_at 
##                        0                        0                      588 
##         first_funding_at          last_funding_at  age_first_funding_year 
##                        0                        0                        0 
##    age_last_funding_year age_first_milestone_year  age_last_milestone_year 
##                        0                      152                      152 
##            relationships           funding_rounds        funding_total_usd 
##                        0                        0                        0 
##               milestones             state_code.1                    is_CA 
##                        0                        1                        0 
##                    is_NY                    is_MA                    is_TX 
##                        0                        0                        0 
##            is_otherstate            category_code              is_software 
##                        0                        0                        0 
##                   is_web                is_mobile            is_enterprise 
##                        0                        0                        0 
##           is_advertising           is_gamesvideo             is_ecommerce 
##                        0                        0                        0 
##               is_biotech            is_consulting        is_othercategory 
##                        0                        0                        0 
##                object_id                   has_VC                has_angel 
##                        0                        0                        0 
##               has_roundA               has_roundB               has_roundC 
##                        0                        0                        0 
##               has_roundD         avg_participants                is_top500 
##                        0                        0                        0 
##                   status 
##                        0
```

Geographical visualisation

First we visualize the latitude and longitude of the observations.

```r
data_start %>%
  ggplot(aes(x = longitude, y = latitude)) +
  geom_point(color = "cornflowerblue")
```

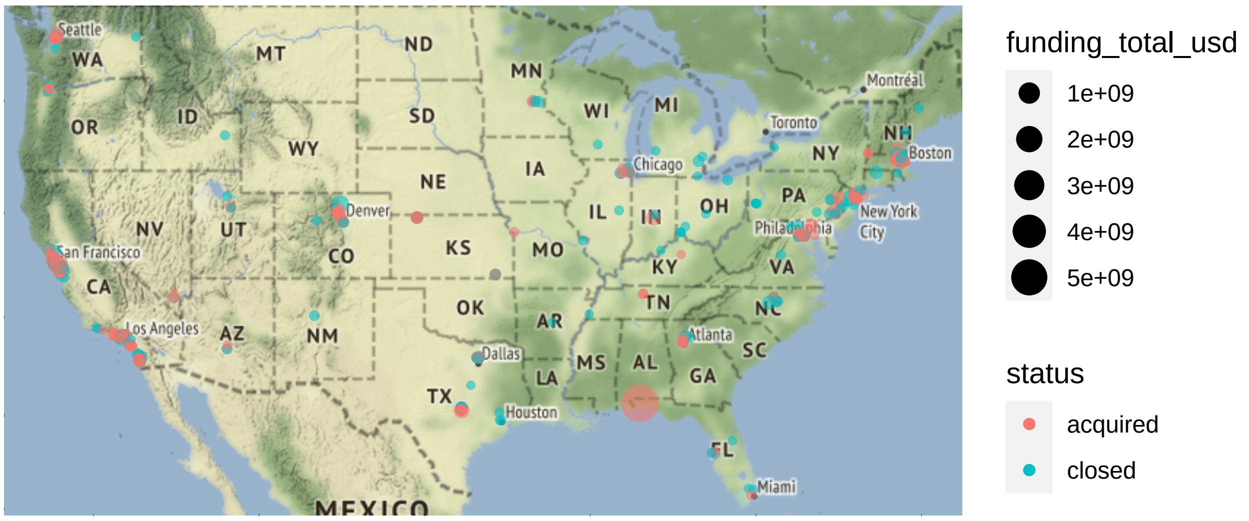


We can see that the dataset contain observations outside of the US. We remove these by filtering.

```
data_start %<>%
  filter(longitude < -40)
```

We can look how the startups perform geographically. We set the color equal to "status" to separate the startups by whether they're acquired or closed. Furthermore we set the size equal to "funding_total_usd" to make the size of the dots dependent on the amount of funding.

```
qmplot(x = longitude,
       y = latitude,
       data = data_start,
       geom = "point",
       color = status,
       size = funding_total_usd,
       alpha = 0.4) +
  scale_alpha(guide = 'none')
```



Data preparation process

We take a quick look at the data. First we divide the "funding_total_usd" by 1000 to make the tables more readable. The "funding_total_usd" will also be renamed to "total".

```
data_start %<>%
  mutate(total = funding_total_usd / 1000) %>%
  select(!funding_total_usd)

skim(data_start)
```

Table 1: Data summary

Name	data_start
Number of rows	919
Number of columns	49
Column type frequency:	
character	11
Date	3
numeric	35
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
state_code	0	1.00	2	2	0	35	0
zip_code	0	1.00	4	14	0	380	0
id	0	1.00	3	8	0	918	0
city	0	1.00	2	19	0	219	0
Unnamed: 6	492	0.46	11	28	0	249	0
name	0	1.00	3	39	0	918	0
closed_at	585	0.36	8	10	0	202	0
state_code.1	1	1.00	2	2	0	35	0
category_code	0	1.00	3	16	0	35	0

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
object_id	0	1.00	3	8	0	918	0
status	0	1.00	6	8	0	2	0

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
founded_at	0	1	1984-01-01	2013-04-16	2006-01-01	217
first_funding_at	0	1	2000-01-01	2013-11-20	2007-09-01	583
last_funding_at	0	1	2001-01-01	2013-11-20	2009-12-16	677

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75
Unnamed: 0	0	1.00	572.44	333.75	1.00	282.50	578.00	866.50
latitude	0	1.00	38.45	3.60	25.75	37.39	37.78	40.73
longitude	0	1.00	-104.00	21.30	-122.76	-122.20	-118.39	-77.31
labels	0	1.00	0.65	0.48	0.00	0.00	1.00	1.00
age_first_funding_year	0	1.00	2.24	2.51	-9.05	0.58	1.45	3.58
age_last_funding_year	0	1.00	3.94	2.97	-9.05	1.67	3.55	5.56
age_first_milestone_year	151	0.84	3.07	2.98	-14.17	1.00	2.58	4.70
age_last_milestone_year	151	0.84	4.77	3.21	-7.01	2.46	4.50	6.75
relationships	0	1.00	7.71	7.28	0.00	3.00	5.00	10.00
funding_rounds	0	1.00	2.31	1.39	1.00	1.00	2.00	3.00
milestones	0	1.00	1.84	1.32	0.00	1.00	2.00	3.00
is_CA	0	1.00	0.53	0.50	0.00	0.00	1.00	1.00
is_NY	0	1.00	0.12	0.32	0.00	0.00	0.00	0.00
is_MA	0	1.00	0.09	0.29	0.00	0.00	0.00	0.00
is_TX	0	1.00	0.05	0.21	0.00	0.00	0.00	0.00
is_otherstate	0	1.00	0.22	0.41	0.00	0.00	0.00	0.00
is_software	0	1.00	0.17	0.37	0.00	0.00	0.00	0.00
is_web	0	1.00	0.16	0.36	0.00	0.00	0.00	0.00
is_mobile	0	1.00	0.09	0.28	0.00	0.00	0.00	0.00
is_enterprise	0	1.00	0.08	0.27	0.00	0.00	0.00	0.00
is_advertising	0	1.00	0.07	0.25	0.00	0.00	0.00	0.00
is_gamesvideo	0	1.00	0.06	0.23	0.00	0.00	0.00	0.00
is_ecommerce	0	1.00	0.03	0.16	0.00	0.00	0.00	0.00
is_biotech	0	1.00	0.04	0.19	0.00	0.00	0.00	0.00
is_consulting	0	1.00	0.00	0.06	0.00	0.00	0.00	0.00
is_othercategory	0	1.00	0.32	0.47	0.00	0.00	0.00	1.00
has_VC	0	1.00	0.33	0.47	0.00	0.00	0.00	1.00
has_angel	0	1.00	0.25	0.44	0.00	0.00	0.00	1.00
has_roundA	0	1.00	0.51	0.50	0.00	0.00	1.00	1.00
has_roundB	0	1.00	0.39	0.49	0.00	0.00	0.00	1.00
has_roundC	0	1.00	0.23	0.42	0.00	0.00	0.00	0.00
has_roundD	0	1.00	0.10	0.30	0.00	0.00	0.00	0.00
avg_participants	0	1.00	2.84	1.88	1.00	1.50	2.50	3.78
is_top500	0	1.00	0.81	0.39	0.00	1.00	1.00	1.00
total	0	1.00	25403.73	190039.60	11.00	2725.00	10000.00	24605.27

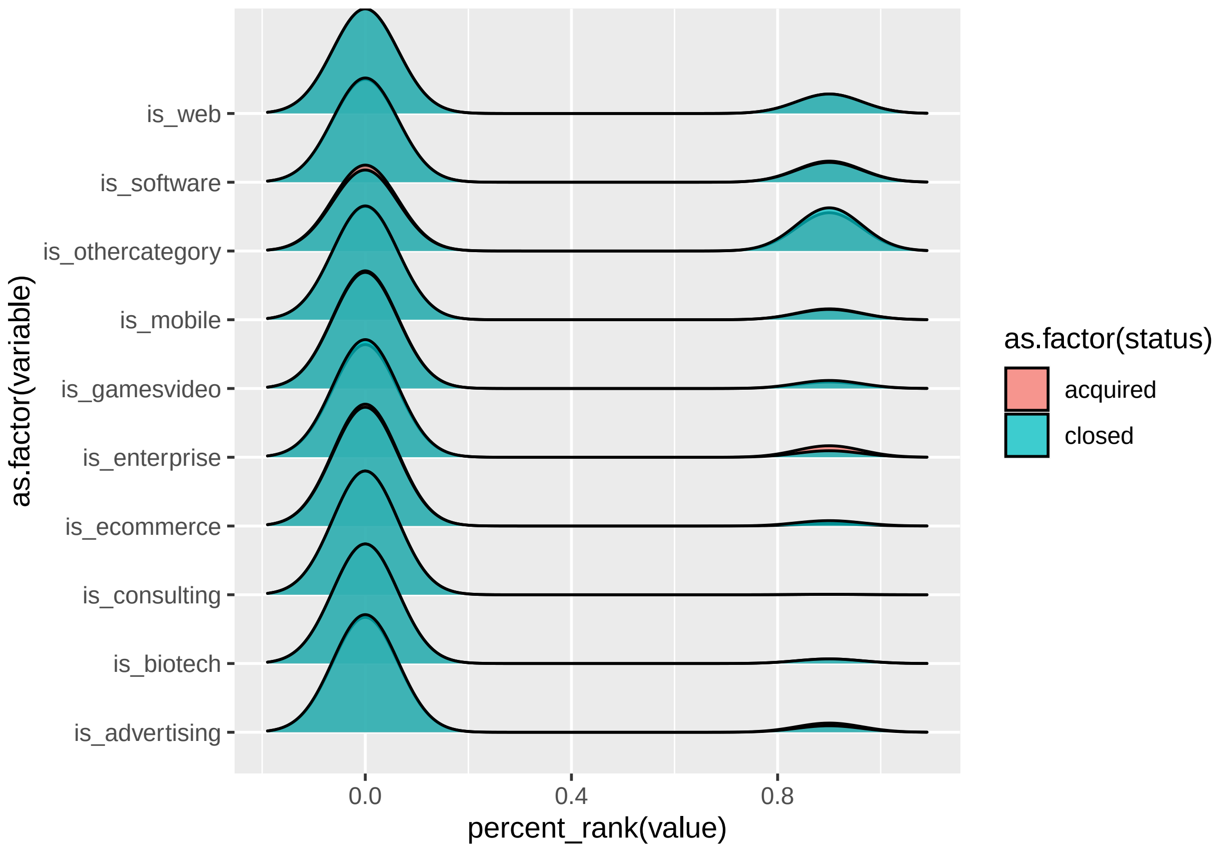
We start by removing columns that are undefined when loading the dataset and columns which show the same things. We also remove variables like longitude and latitude, which doesn't seem to bring much insight to the ongoing analysis.

```r
data = data_start %>%
  select(!c(`Unnamed: 0`, `Unnamed: 6`, state_code.1, object_id, avg_participants,
            has_roundA, has_roundB, has_roundC, has_roundD, zip_code, id, city, name,
            latitude, longitude, labels))
```

We divide the data into different groups and then look at the data to see where the distribution seem to differ depending on the status of the startup. We use geom_density_ridges to do this.

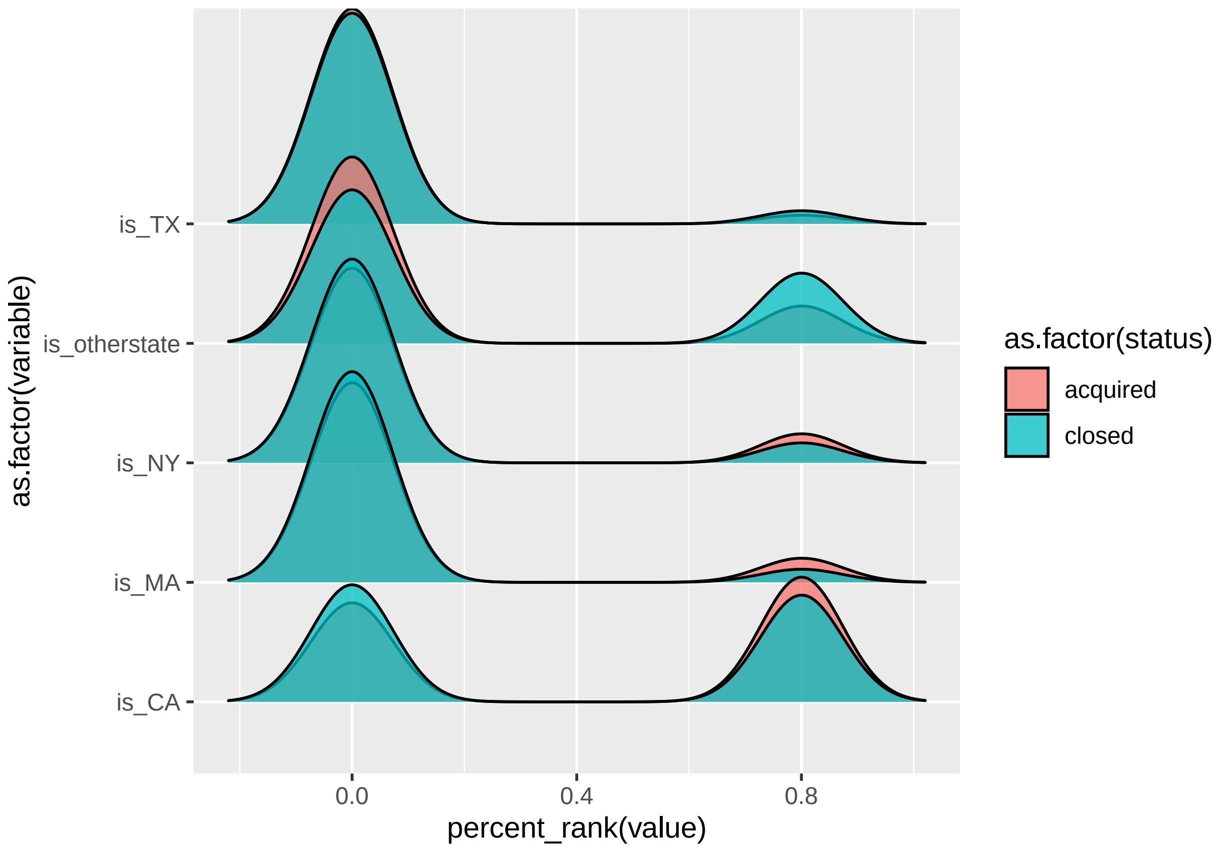
```r
jobs = data %>%
  select(is_software, is_web, is_mobile, is_enterprise, is_advertising, is_gamesvideo,
         is_ecommerce, is_biotech, is_consulting, is_othercategory, status)

state = data %>%
  select(is_CA, is_TX, is_MA, is_NY, is_otherstate, status)

dummies = data %>%
  select(is_top500, has_angel, has_VC, status)

par(mfrow=c(1,3))
jobs %>%
  select(status, is_numeric) %>%
  gather(variable, value, -status) %>%
  ggplot(aes(y = as.factor(variable),
             fill =  as.factor(status),
             x = percent_rank(value)) ) +
  ggridges::geom_density_ridges(alpha = 0.75)
```

```
## Picking joint bandwidth of 0.0633
```



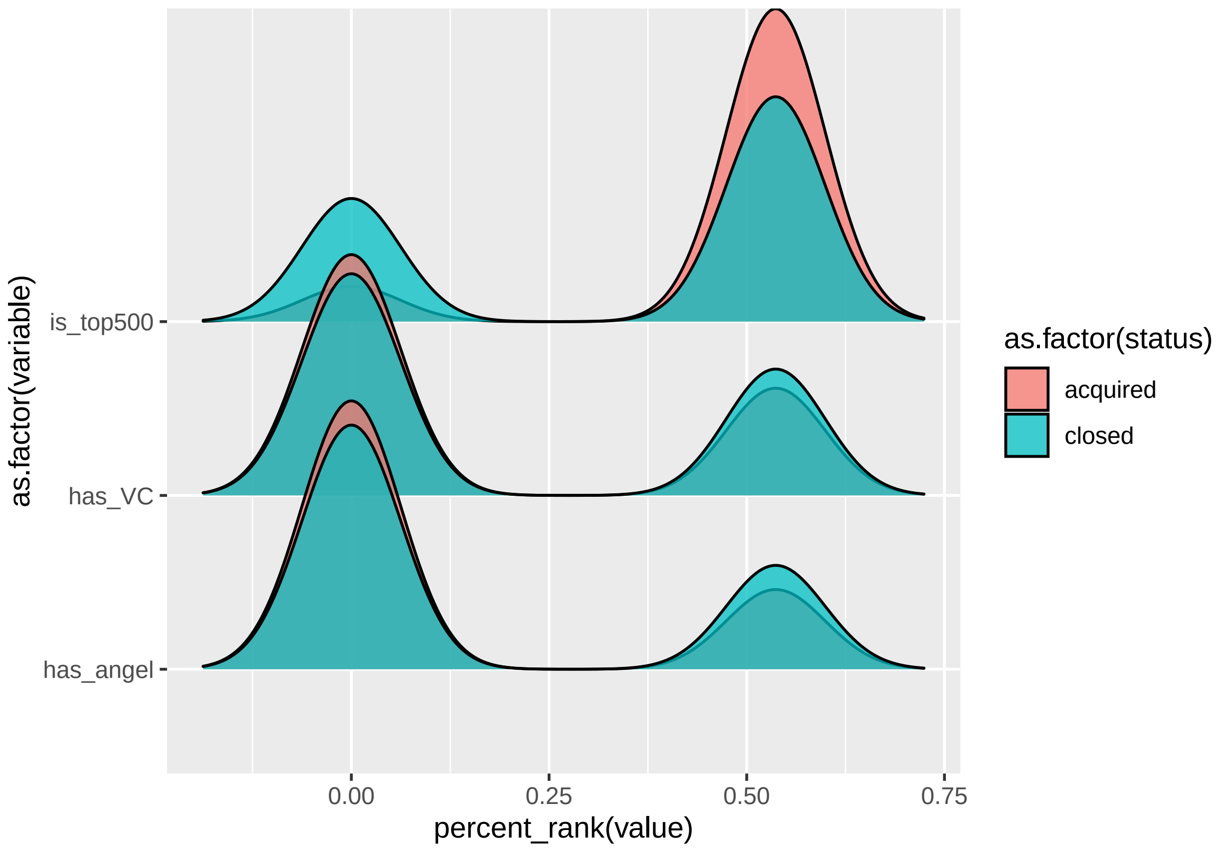
```
state %>%
  select(status, is_numeric) %>%
  gather(variable, value, -status) %>%
  ggplot(aes(y = as.factor(variable),
             fill =  as.factor(status),
             x = percent_rank(value)) ) +
  ggridges::geom_density_ridges(alpha = 0.75)
```

```
## Picking joint bandwidth of 0.0733
```



```r
dummies %>%
  select(status, is_numeric) %>%
  gather(variable, value, -status) %>%
  ggplot(aes(y = as.factor(variable),
             fill =  as.factor(status),
             x = percent_rank(value)) ) +
  ggridges::geom_density_ridges(alpha = 0.75)
```

```
## Picking joint bandwidth of 0.0625
```



By looking at the distribution plot we choose the variables that seem to be able to predict whether the startup will close or remain acquired.

Because of the above we drop the state and job variables plus the VC and angel dummies, but we keep is_top500 and then we skim the data again.

```
data %<>%
  select(-c(is_CA, is_TX, is_MA, is_NY, is_otherstate, is_software, is_web, is_mobile,
            is_enterprise, is_advertising, is_gamesvideo, is_ecommerce, is_biotech,
            is_consulting, is_othercategory, state_code, category_code, has_angel,
            has_VC))

skim(data)
```

Table 5: Data summary

Name	data
Number of rows	919
Number of columns	14
Column type frequency:	
character	2
Date	3
numeric	9
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
closed_at	585	0.36	8	10	0	202	0
status	0	1.00	6	8	0	2	0

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
founded_at	0	1	1984-01-01	2013-04-16	2006-01-01	217
first_funding_at	0	1	2000-01-01	2013-11-20	2007-09-01	583
last_funding_at	0	1	2001-01-01	2013-11-20	2009-12-16	677

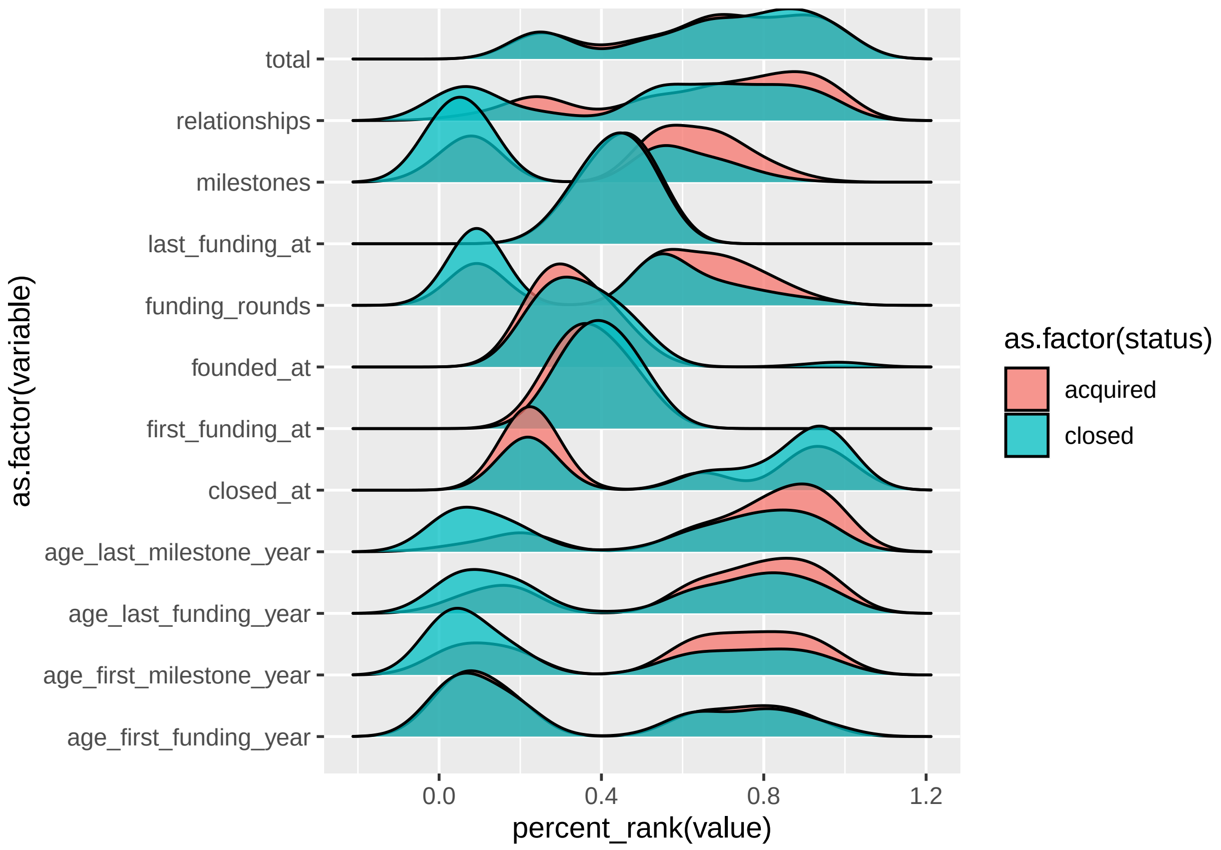
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75
age_first_funding_year	0	1.00	2.24	2.51	-9.05	0.58	1.45	3.58
age_last_funding_year	0	1.00	3.94	2.97	-9.05	1.67	3.55	5.56
age_first_milestone_year	151	0.84	3.07	2.98	-14.17	1.00	2.58	4.70
age_last_milestone_year	151	0.84	4.77	3.21	-7.01	2.46	4.50	6.75
relationships	0	1.00	7.71	7.28	0.00	3.00	5.00	10.00
funding_rounds	0	1.00	2.31	1.39	1.00	1.00	2.00	3.00
milestones	0	1.00	1.84	1.32	0.00	1.00	2.00	3.00
is_top500	0	1.00	0.81	0.39	0.00	1.00	1.00	1.00
total	0	1.00	25403.73	190039.60	11.00	2725.00	10000.00	24605.27

We still have 8 numerical variables and 3 date variables we haven't taken a closer look at so that is what we're gonna do now. We again use the geom_density_ridges to look at the relevant variables.

```r
data %>%
  select(status, is_numeric, !is_top500) %>%
  gather(variable, value, -status, -is_top500) %>%
  ggplot(aes(y = as.factor(variable),
             fill =  as.factor(status),
             x = percent_rank(value)) ) +
  ggridges::geom_density_ridges(alpha = 0.75)
```

```
## Picking joint bandwidth of 0.0708
```



The above plot shows, that relationships, milestones, funding_total_usd, funding_rounds, closed at, both first and last milestone_year and last_funding_year seems to impact whether or not a firm gets acquired or not. Closed_at will not be used because it contains almost 600 missing observations which is close to 66% of all observations. Founded at will not be used either simply because there doesn't seem to be that big of a difference in the density plots. And using both first and last milestone_year would maybe be irrelevant because they somehow nearly show the same thing. so we drop them. First and last funding_at doesn't show any significance either, so those also gets dropped.

```
data%<>%
  select(!c(age_last_milestone_year, closed_at, age_first_funding_year,founded_at,
          first_funding_at, last_funding_at))
```

Now we are left with only our status variable and our 7 numerical variables of interest. Now we check for missing values in our remaining variables.

```
data %>%
  summarise_all(funs(sum(is.na(.))))
```

```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
```

```
## # A tibble: 1 x 8
##   age_last_funding_~ age_first_mileston~ relationships funding_rounds milestones
##                <int>               <int>         <int>          <int>      <int>
## 1                  0                 151             0              0          0
## # ... with 3 more variables: is_top500 <int>, status <int>, total <int>
```

The above states that the age_first_milestone_year has 152 missing values. The problem with this variable is, that if we remove those rows, we lose 16.4% of our observations, and we cant just replace the NA's with something else like zero, because that would just manipulate our data. So the solution we have come up with is to make them into intervals and then just pick one of the intervals to be used in our analysis.

```
data %<>%
  mutate(age_first_milestone_year= ifelse(age_first_milestone_year <0,"before year 0",
      ifelse(age_first_milestone_year <= 3 , "[0-3]",
      ifelse(age_first_milestone_year <= 6, "]3-6]",
      ifelse(age_first_milestone_year >6, "over 6",""))))) %>%
  mutate(age_first_milestone_year= replace_na(age_first_milestone_year, "no milestone"))


data %>%
  filter(age_first_milestone_year == "before year 0")%>%
  count(status)
```

```
## # A tibble: 2 x 2
##   status       n
##   <chr>    <int>
## 1 acquired    25
## 2 closed      19
```

```
data %<>%
  mutate("is_before_start" = as.numeric(age_first_milestone_year == "before year 0")) %>%
  mutate("is_0:3" = as.numeric(age_first_milestone_year == "[0-3]")) %>%
  mutate("is_3:6" = as.numeric(age_first_milestone_year == "]3-6]")) %>%
  mutate("is_6<" = as.numeric(age_first_milestone_year == "over 6")) %>%
  mutate("is_no_milestone" = as.numeric(age_first_milestone_year == "no milestone"))


data %<>%
  select(!c(age_first_milestone_year, is_before_start, `is_3:6`, `is_6<`, is_no_milestone))
```

We pick the interval zero to three years, so we drop the rest, which means that firms achieved their first milestone when they were between 0 and 3 years old. We pick this interval because we wanna investigate whether it is positive or not in terms of getting acquired for a firm to get their first milestone quick eg. in the first couple of years.

And this conclude our variable selection which now will be explained more thoroughly in the stakeholder.

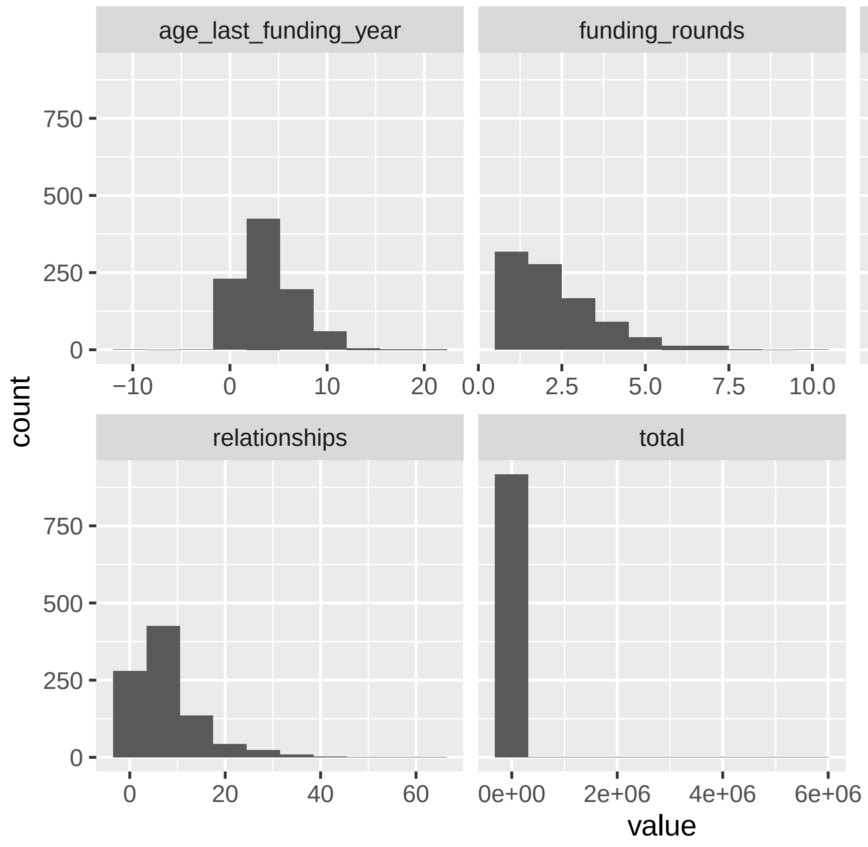
```
colnames(data)
```

```
## [1] "age_last_funding_year" "relationships"         "funding_rounds"
## [4] "milestones"            "is_top500"             "status"
## [7] "total"                 "is_0:3"
```

Then we check for outliers

```
data_plot = data %>%
  select(!c(`is_0:3`, is_top500, status))
```

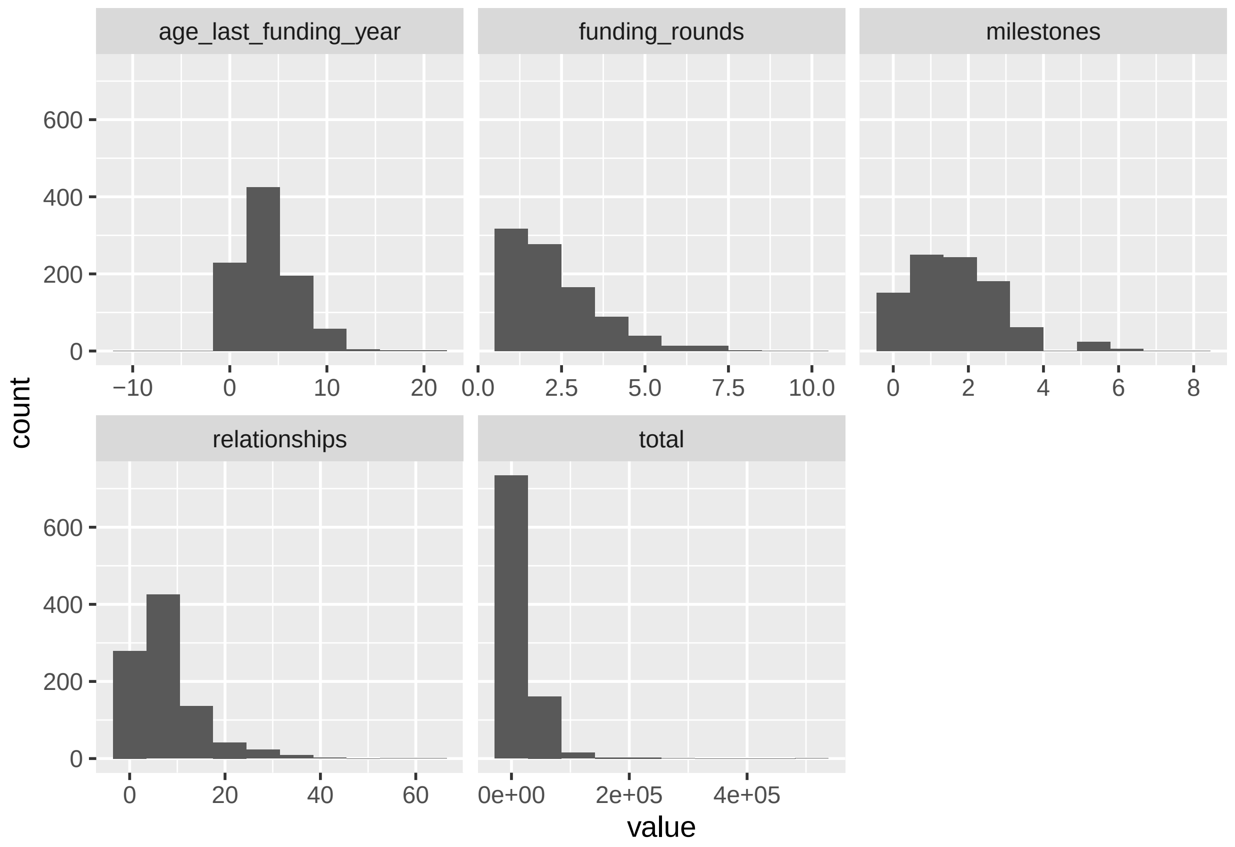
```
ggplot(gather(data_plot), aes(value)) +
  geom_histogram(bins = 10) +
  facet_wrap(~key, scales = 'free_x')
```



```
data%>% arrange(desc(total))
```

```
## # A tibble: 919 x 8
##    age_last_funding_y~ relationships funding_rounds milestones is_top500 status
##                  <dbl>         <dbl>          <dbl>      <dbl>     <dbl> <chr>
##  1                9.42            19              4          2         1 acquir~
##  2                3.96             5              2          3         1 closed
##  3                4.53            37              5          2         1 acquir~
##  4               11.9             29              7          3         1 acquir~
##  5                7.25             4              6          0         1 acquir~
##  6                6.42             6              8          2         1 closed
##  7               15.0              4              7          1         1 closed
##  8                7.16             1              3          0         1 closed
##  9                3.47            38              5          3         1 acquir~
## 10               11.2              3              4          2         1 acquir~
## # ... with 909 more rows, and 2 more variables: total <dbl>, is_0:3 <dbl>
```

```
data %<>%
  filter(total < 5700000)

data_plot = data %>%
  select(!c(`is_0:3`, is_top500, status))
  ggplot(gather(data_plot), aes(value)) +
    geom_histogram(bins = 10) +
  facet_wrap(~key, scales = 'free_x')
```



We removed the outlier in the variable "total" as this value was 10x larger than the second largest observation.

Now we move on to unsupervised machine learning.

Unsupervised ML

We start by performing dimensionality reduction on our data in the form of PCA, but before we can do that, we need to examine our data to figure out if we need to scale it. We do this by calculating the mean and standard deviations of the variables.

```
options(scipen = 999)

data_num = data %>%
  select_if(is.numeric)

s_deviation=apply(data_num,2, sd)

mean1=colMeans(data_num)

scale= as.data.frame(s_deviation, row.names = c("sd"))%>%
cbind(as.data.frame(mean1, row.names = "mean"))%>%
print()
```

```
##                          s_deviation        mean1
## age_last_funding_year      2.9620686    3.9290531
## relationships              7.2746264     7.7026144
## funding_rounds             1.3932926     2.3093682
## milestones                 1.3242545     1.8420479
## is_top500                  0.3930135     0.8093682
## total                  31631.2901824 19222.2531721
## is_0:3                     0.4939050     0.4204793
```

We can se the data is going to need scaling to perform the PCA, because our variables are not on the same scale. Namely "total" seems much larger. We also have some dummy variables which only take a value of either 0 or 1, so these will also be on a different scale then the rest.

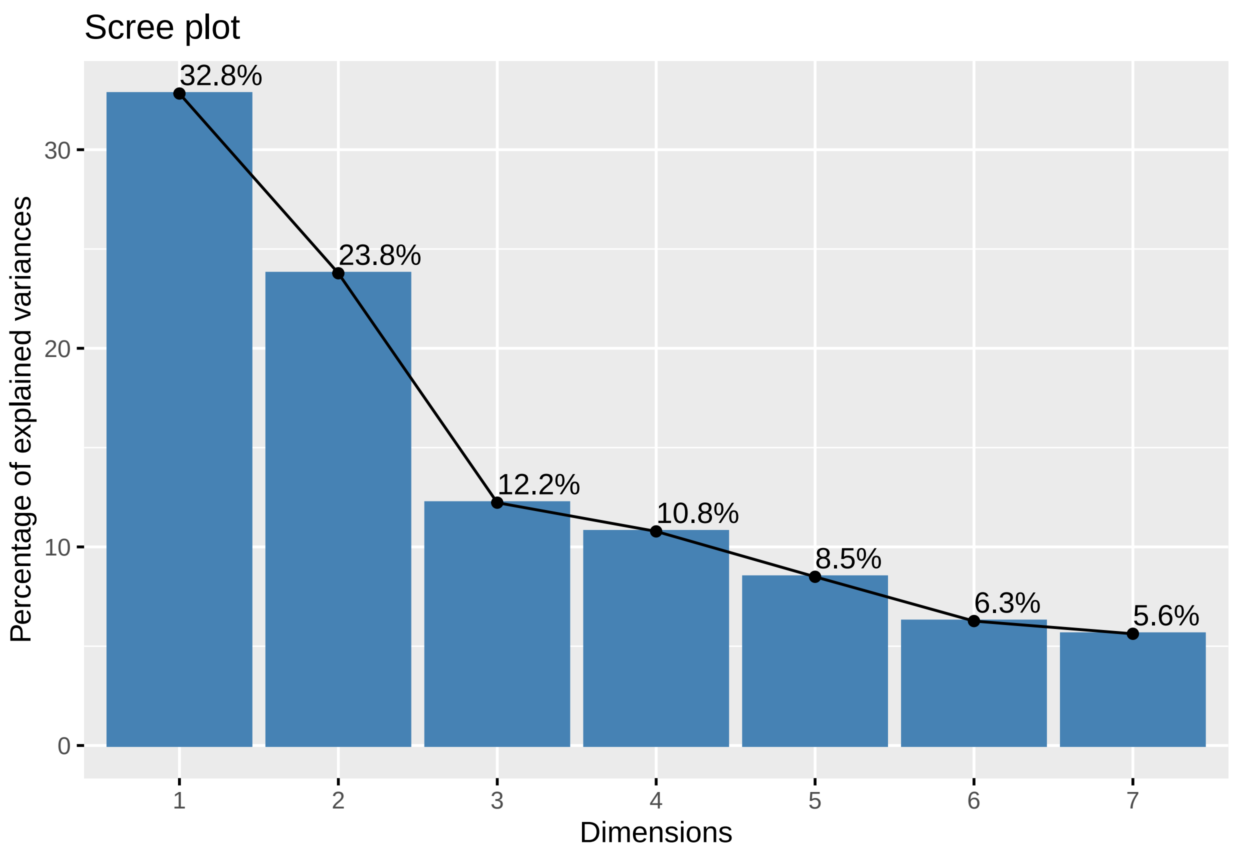
PCA

We scale the data by setting the argument scale.unit to TRUE and then we run the PCA. Further we can only run our PCA on numeric variables so we use the data subset just created above "data_num".

```
res_pca <- data_num %>%
  PCA(scale.unit = TRUE, graph =FALSE)
```

Now we have our PCA we can create a screeplot to pick the number of dimensions to use.

```
res_pca %>%
  fviz_screeplot(addlabels = TRUE,
                 ncp = 10,
                 ggtheme = theme_gray())
```

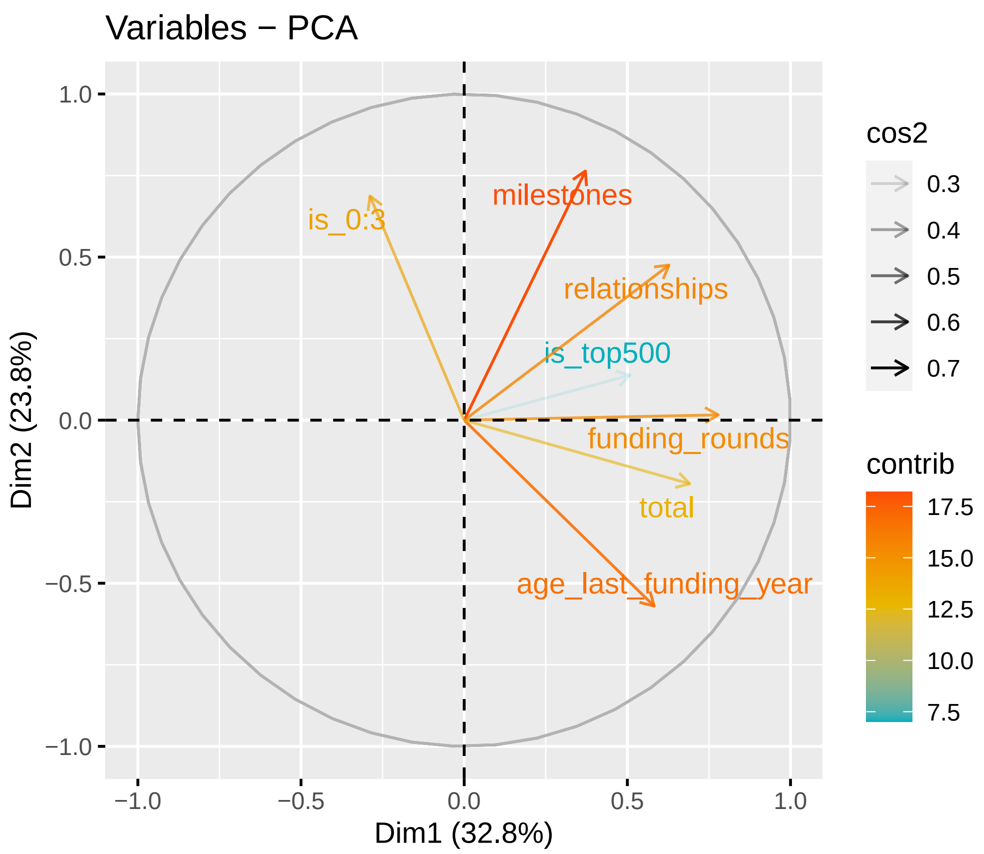


```
eig.val = get_eigenvalue(res_pca); eig.val
```

```
##       eigenvalue variance.percent cumulative.variance.percent
## Dim.1  2.2979309        32.827585                    32.82758
## Dim.2  1.6644850        23.778357                        56.60594
## Dim.3  0.8558298        12.226141                        68.83208
## Dim.4  0.7546476        10.780680                        79.61276
## Dim.5  0.5946997         8.495710                        88.10847
## Dim.6  0.4386274         6.266106                        94.37458
## Dim.7  0.3937795         5.625421                       100.00000
```

We can see the elbow shows the optimal dimensions are three dimensions with almost 69% explained variance. If we only look at the eigenvalues then our rule of thumb is to pick the dimensions with an eigenvalue >=1, which in this case is two dimensions. Those two dimensions only account for 57% of the total variance, which isn't that high. As it is hard to understand a plot with three dimensions we visualize our reduced data in two dimensions.

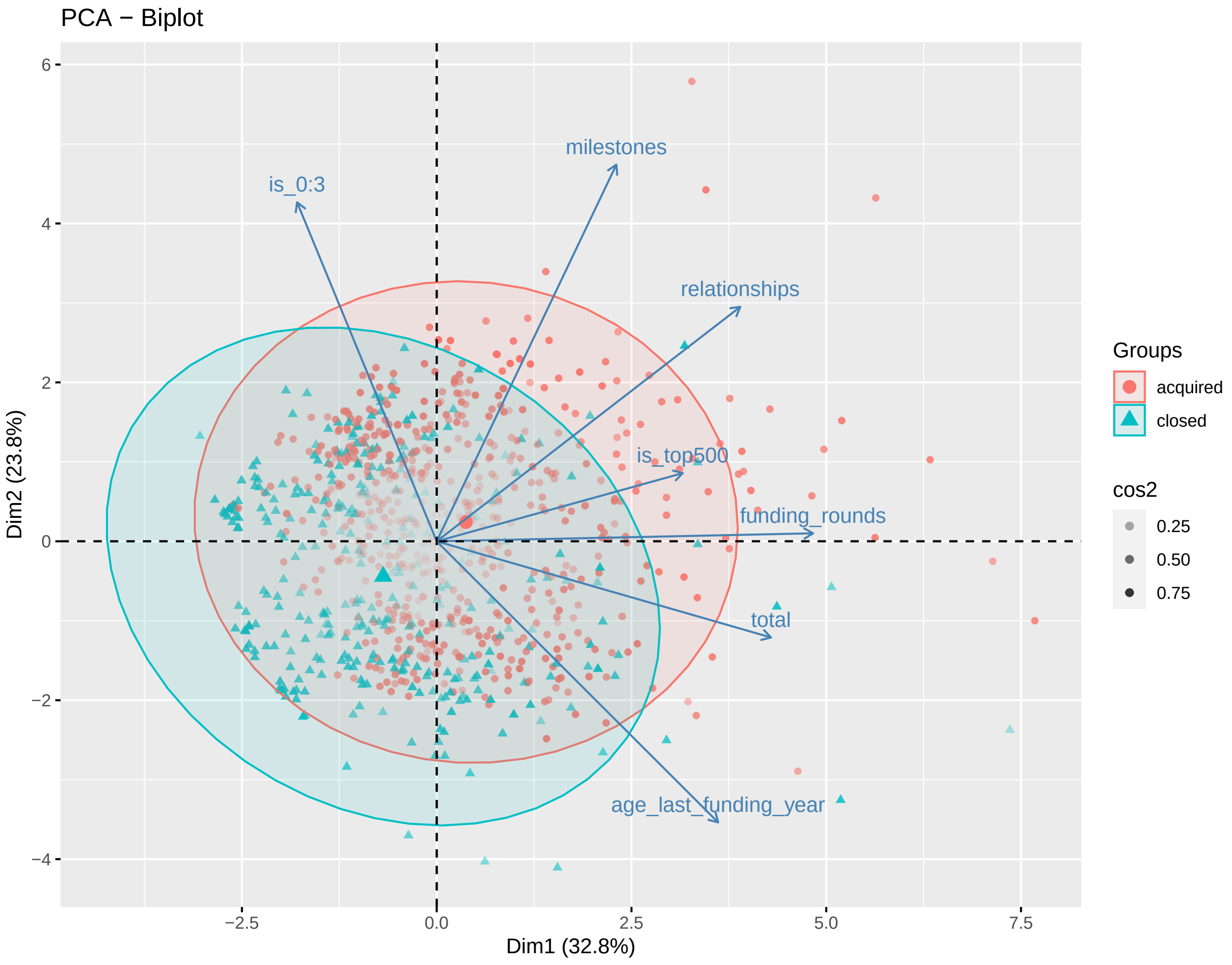
```
res_pca %>%
  fviz_pca_var(alpha.var = "cos2",
               col.var = "contrib",
               gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
               repel = TRUE,
               ggtheme = theme_gray())
```



From the plot we can see, that the x-axis split our seven variables into two groups. Where the ones involving funding or money is below or on the x_axis. funding_rounds lies on the x-axis which indicates that it is mostly explained by the first dimension. Variables like milestones, relationships, is_top500 and is_0:3 are above the x-axis. It can be said about our variables, that the less opaque the arrow the higher the cos2 and thereby a higher representation of the variable in the principal components we have used eg. is_top500 is not as well explained by the first two principal components as the other variables. Contrib shows almost the same as it is just (cos2*100)/(total cos2 of the component), that is why we see the one which are the most opaque also being the one with a blue/greenish color.

Now we plot all our observations in the two dimensional space and try to split them up by their status of either being acquired or closed.

```
res_pca %>%
fviz_pca_biplot(alpha.ind = "cos2",
habillage = data %>% pull(status) %>% factor(),
addEllipses = TRUE,
geom = "point",
ggtheme = theme_gray())
```



The plot shows that the red which is the firms who have been aquired tends to be more to the right, which indicates that they have reached more milestones, had more relationships and had a higher tendency to be a top500 startup. But to conclude the PCA doesn't really separate the firms by status.

We've tried doing the ellipses with other categories such as state of origin and type of industry, but neither one seems to show intuitive results.

Next we are going to use another dimensionality reduction method namely UMAP to see if it does a better job.

UMAP

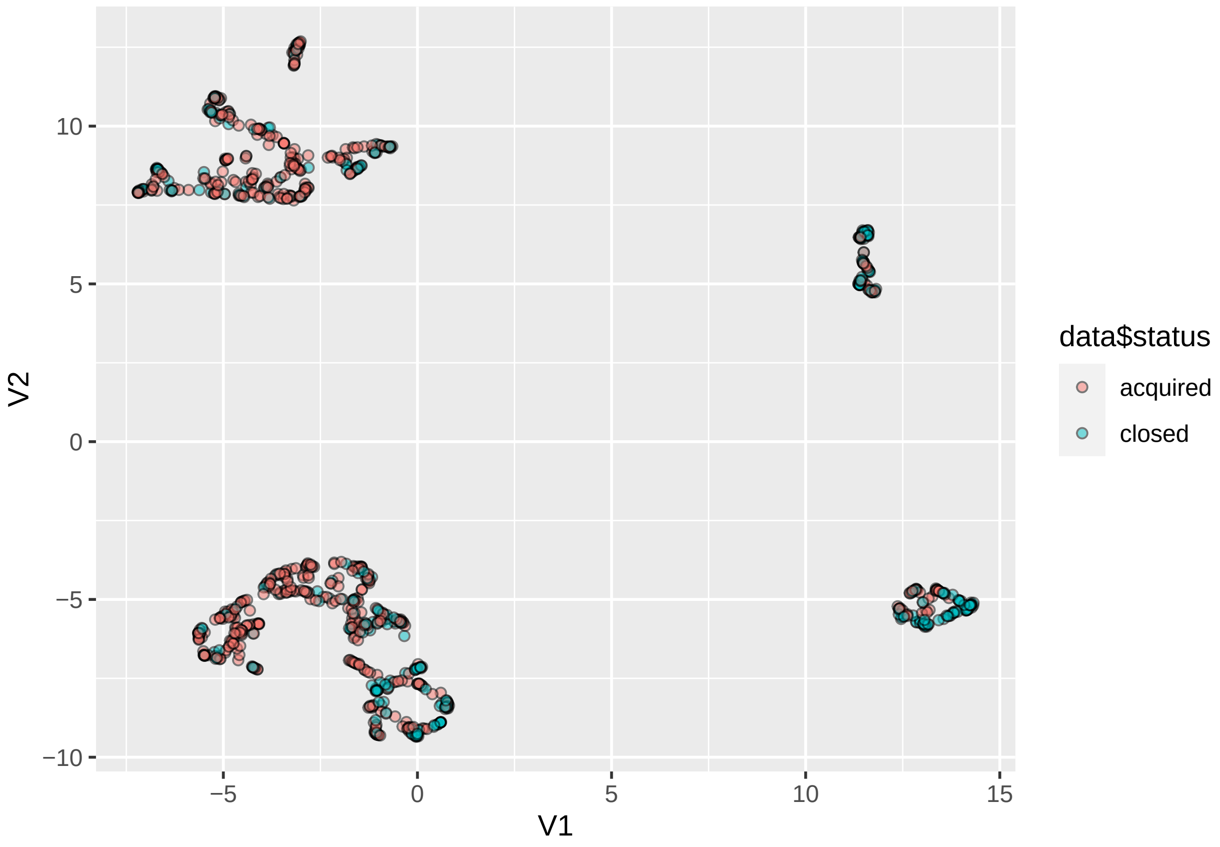
First we create the UMAP object and remember to scale it. n_neigbors shows the size of the local neighborhood. A smaller value will result in more data being preserved. We set metric to "cosine", which is just a method of calculating the distances between observations. min_dist just shows the minimum amount of distance allowed between the observations.

```
res_umap <- data_num  %>%
umap(n_neighbors = 15,
     metric = "cosine",
min_dist = 0.01,
scale = TRUE)
```

Then we plot it in a two dimensional space and fill our observations by status.

```
res_umap %>%
as_tibble() %>%
ggplot(aes(x = V1, y = V2, fill = data$status)) +
geom_point(shape = 21, alpha = 0.5)
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
```

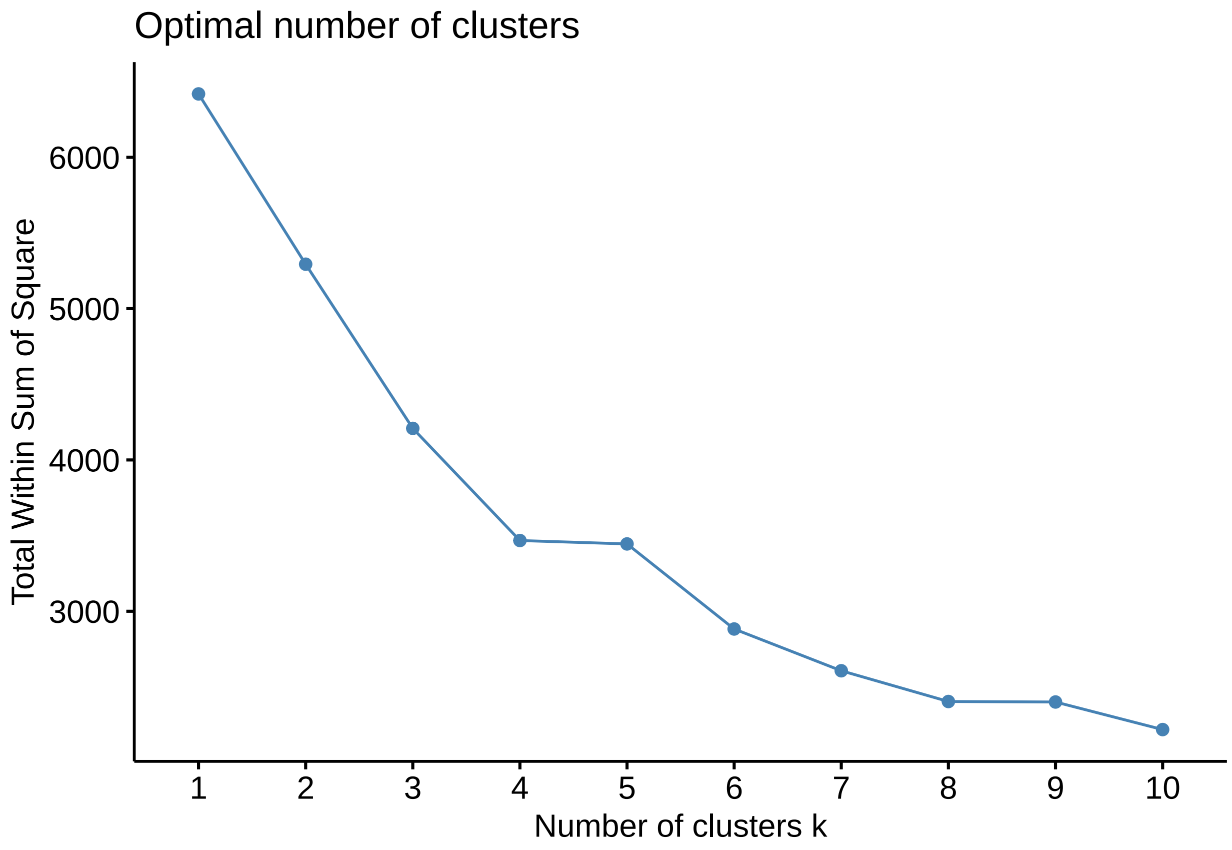


UMAP seems to separate the observations better as we kinda have 4 clusters here. But they are not separated that well between closed and acquired. Again we have used various categories in the "fill" of the function, but the others didn't seem to show a very meaningful separation.

K-means clustering

Now we wanna look at clustering of our data to see, whether our data are clustered by status or not, we make clusters both on our data and on our PCA data and we both make K-means clustering and hierarchical clustering. First K-means clustering on our dataset. We set the method to be used for estimating the optimal number of clusters to "wss" which is the total within sum of square.

```
data_num %>%
scale() %>%
fviz_nbclust(kmeans, method = "wss")
```



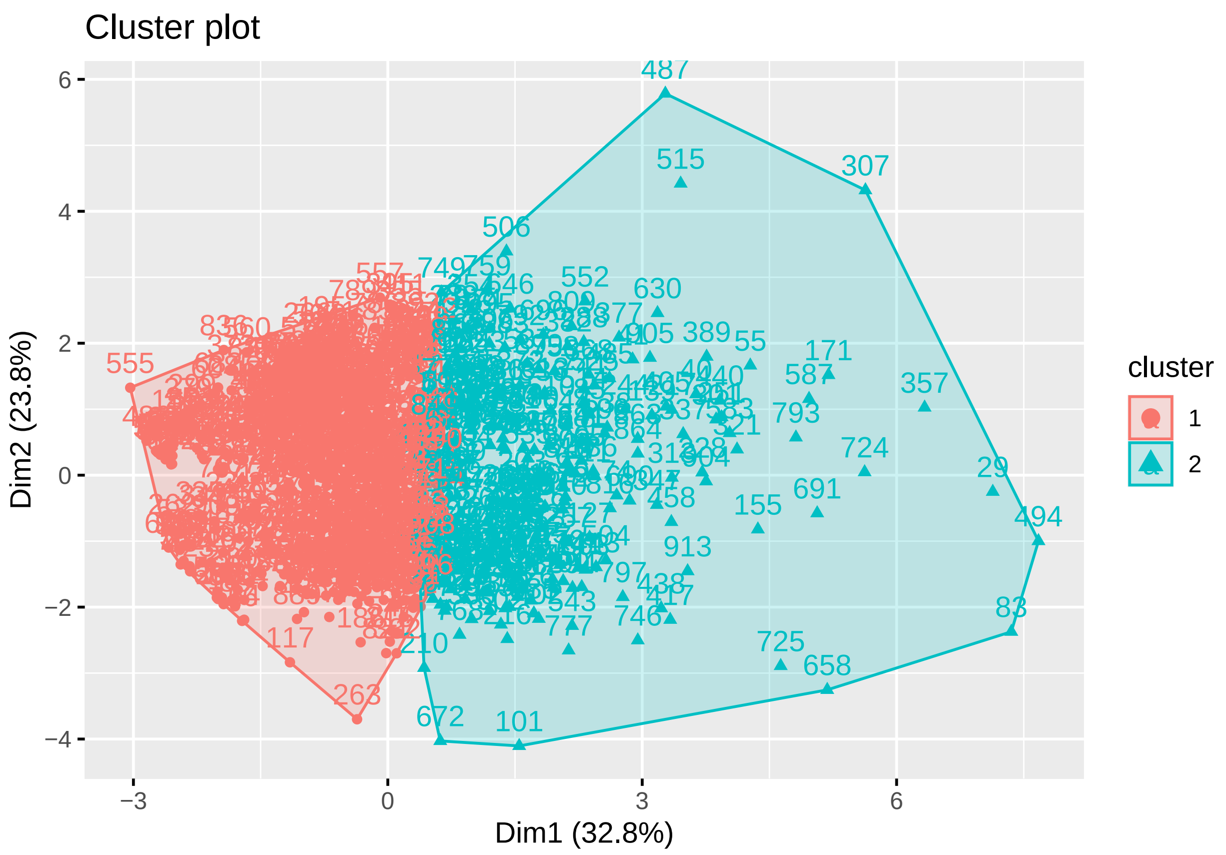
Seems like the optimal number of clusters is somewhere around 3 clusters, but as we want to show whether we can make two clusters which are separated by status which only has two different values "acquired" and "closed" we move on with two clusters.

We set the number of clusters (centers) to 2. We set nstart to 20, which indicates how many different random starting points the test will do in order to select the best one.

```
res_km2 <- data_num %>%
scale() %>%
kmeans(centers = 2, nstart = 20)
```

Now we plot the clusters.

```
res_km2 %>%
fviz_cluster(data = data_num ,
ggtheme = theme_gray())
```



K-means makes 2 clusters, but to observe the separation more clearly, we take a look at the result in a table by extracting the cluster number and putting into our data set.

```
data[,"cluster2"] <- res_km2$cluster

table(data$cluster2, data$status)
```

```
##
##     acquired closed
##   1      372    271
##   2      221     54
```

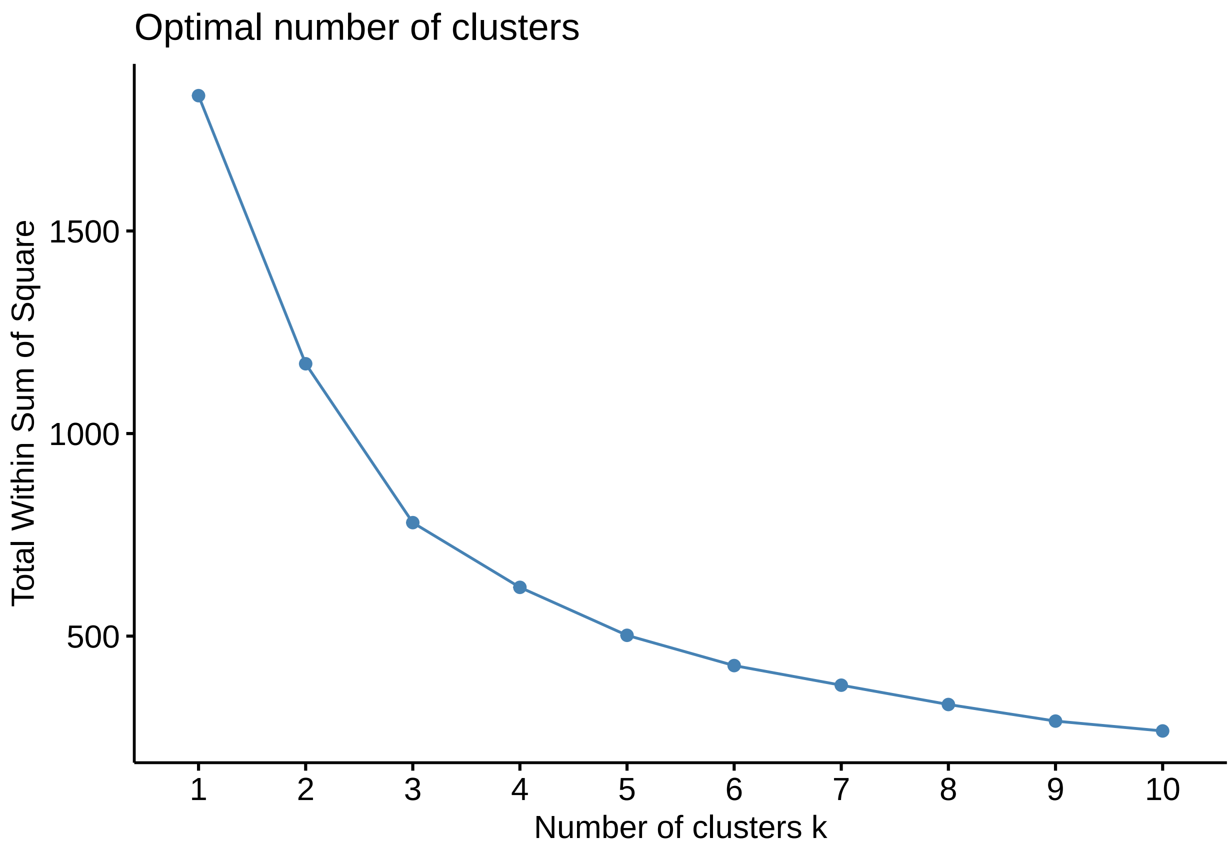
The clusters don't seem to be separated by the status of the startup at all. We can see that one of the clusters both has most startups who have been acquired and closed.

K-means clustering after dimensionality reduction

Now we try to run K-means again - this time on the dimensionality reduced data, so we extracted the first two components as the eigenvalue limited the number of dimensions to two, even though the screeplot showed three dimensions (we performed K-means clustering with three dimensions, but didn't get separated clusters) and put them into a new data set called data_pca.

```
pca1 = res_pca$ind$coord[,1]
pca2 = res_pca$ind$coord[,2]
data_pca = data.frame(pca1, pca2)



data_pca %>%
  scale() %>%
```

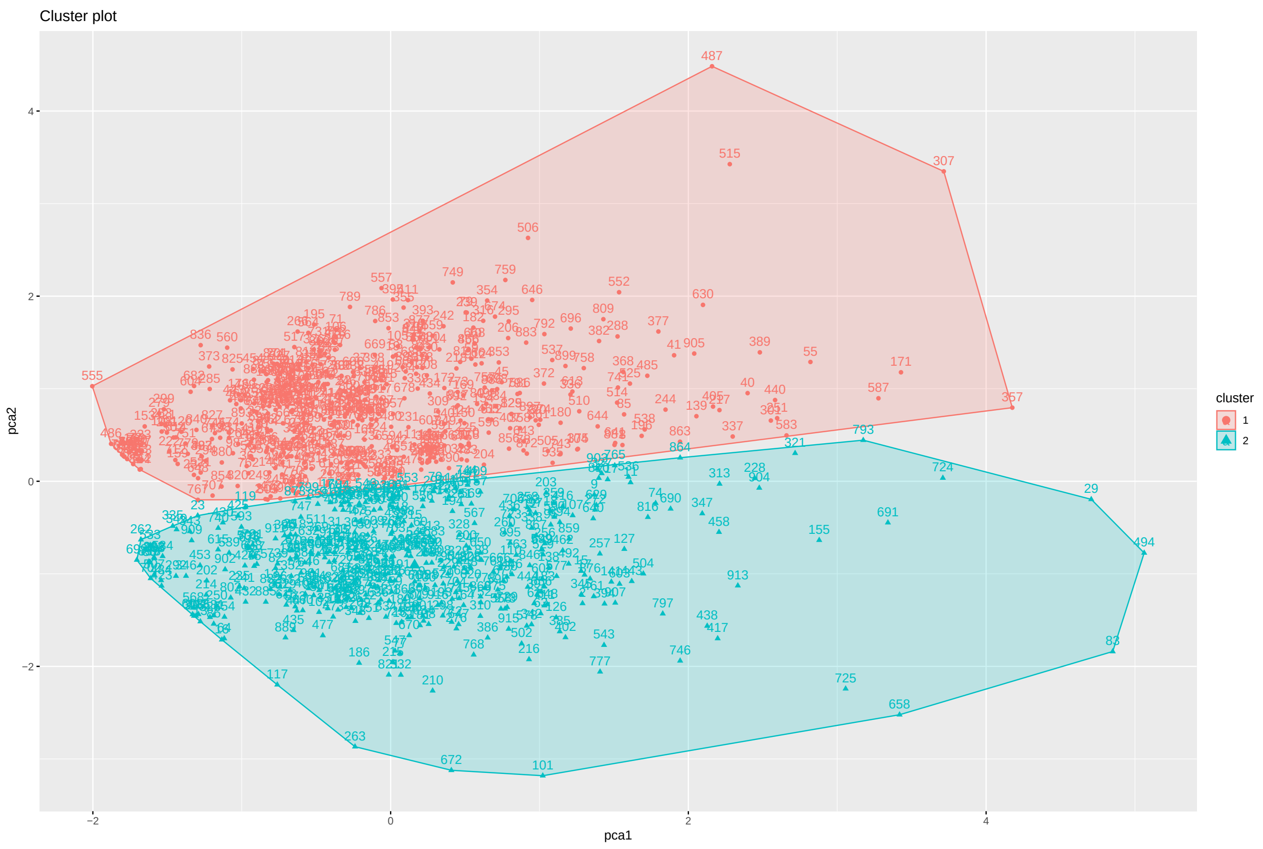
```r
fviz_nbclust(kmeans, method = "wss")
```



We can now see the elbow is formed at 3 clusters, but we still move on with 2 as we want to try and separate by status.

```r
res_km_pca1 <- data_pca %>%
  scale() %>%
  kmeans(centers = 2, nstart = 20)
```

```r
res_km_pca1 %>%
  fviz_cluster(data = data_pca,
               ggtheme = theme_gray())
```



Clustering with the pca data seems to make two clusters who are much more separated, so we again check in a table to to see how well they are separated by status.

```
data[,"cluster_pca2"] <- res_km_pca1$cluster

table(data$cluster_pca2, data$status); table(data$cluster2, data$status)
```

```
##
##     acquired closed
##   1      352    144
##   2      241    181
```

```
##
##     acquired closed
##   1      372    271
##   2      221     54
```

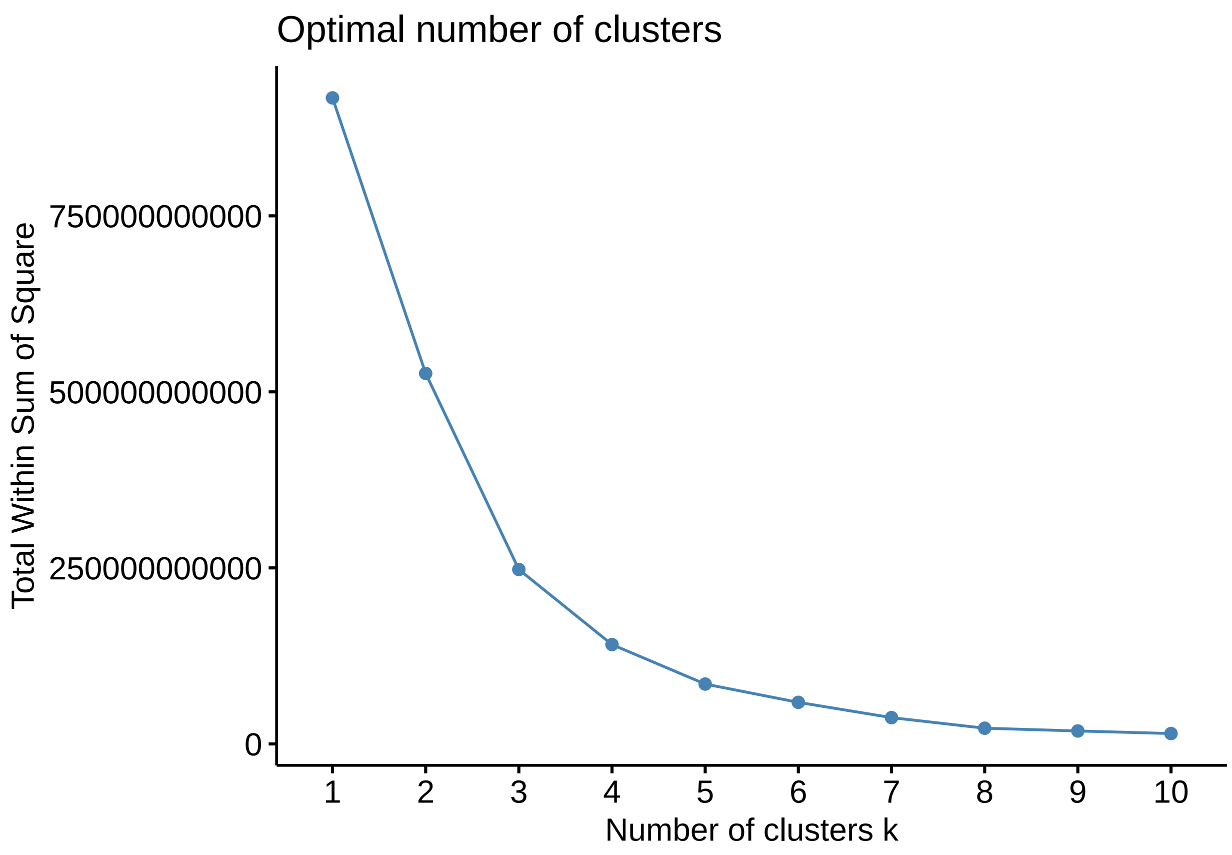
Running it on the dimensionality reduced data seems to have separated the firms by status a little worse then by running it on the entire data. We see the acquired firms are almost identical between the two, but the number of closed firm in each clusters has come closer to each other instead of farther away.

Now we will try to use a different clustering method namely hierarchical clustering.

Hierarchical clustering

We make the hclust object and put stand equal to true as it will scale our data.

```
fviz_nbclust(data_num, FUN = hcut, method = "wss")
```



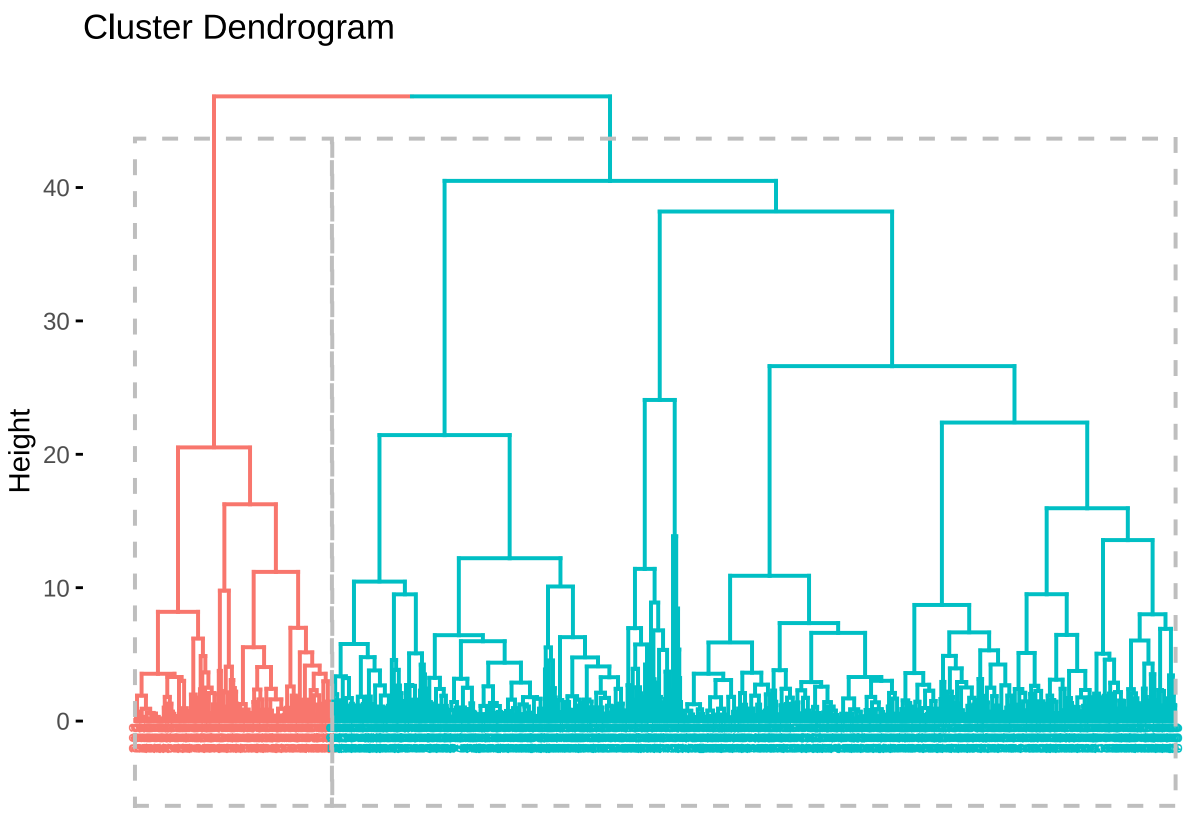
We see that the optimal amount of clusters is 3, but again we use 2.

```
res_hc = data_num %>%
  hcut(hc_func = "hclust", k = 2, stand = TRUE)
```

Then we make a dendogram.

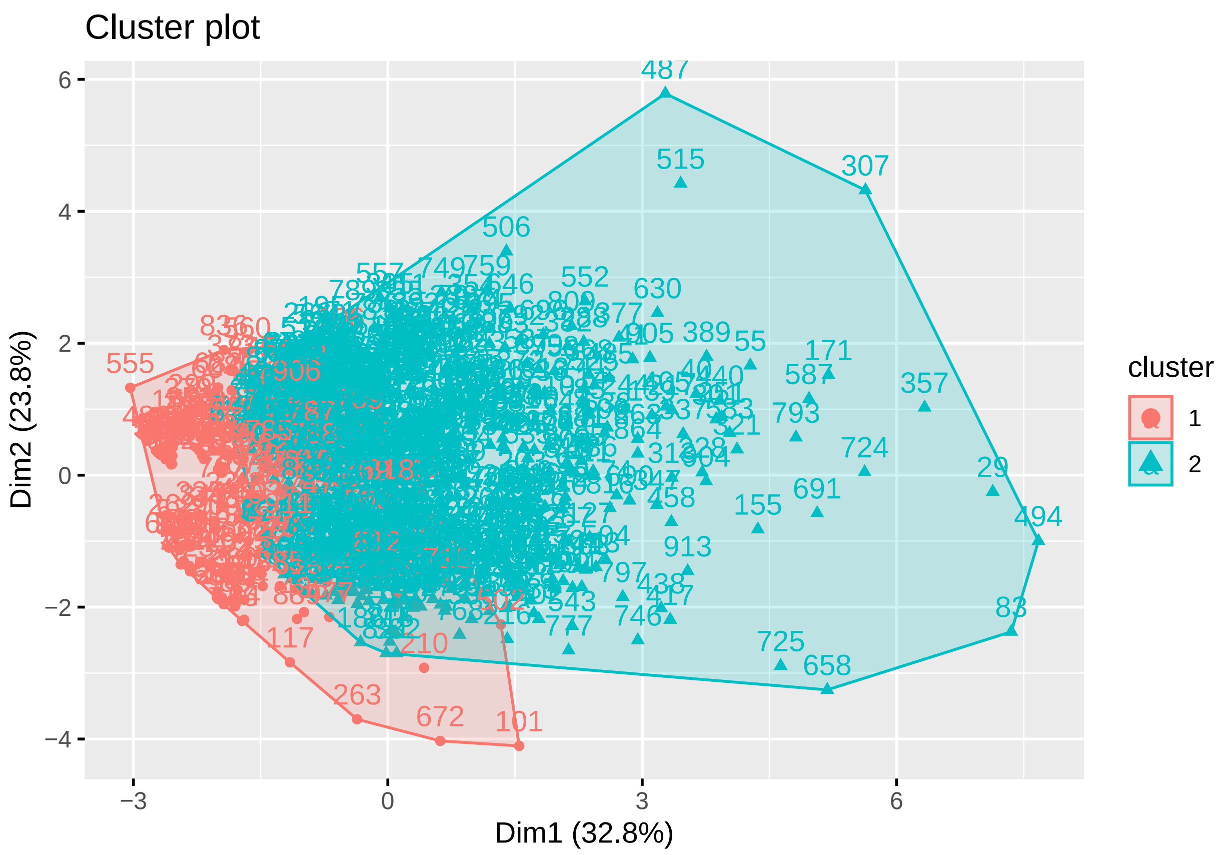
```
res_hc %>%
  fviz_dend(rect = TRUE, cex = 0.5)
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



The dendogram can be a bit of a mess when you deal with a high number of observations, but we can clearly see that by dealing with only two clusters we separate our data high up in the dendogram, which basically means that the two clusters are hard to distinguish between, which partly explains why we are having a hard time separating our startups by status. But as before we can plot our clusters in two dimensional space.

```
res_hc %>%
  fviz_cluster(data = data_num, ggtheme = theme_gray())
```



The plot in itself looks like the two clusters are fairly overlapped which might be due to us forcing 2 clusters on it instead 3 as the screeplot showed, but let us take a look at how well it separated the firms by status.

```
data[,"cluster_hclust2"] <- res_hc$cluster

table(data$cluster_hclust2, data$status)
```

```
##
##     acquired closed
##   1       57    117
##   2      536    208
```

Again, it seems that the the hierarchical clustering of the data doesn't really separate the startups in different clusters, although cluster 1 contains the largest amount of closed startups while cluster 2 contains the largest amount of acquired startups. Lastly let us try hclust on the PCA data.

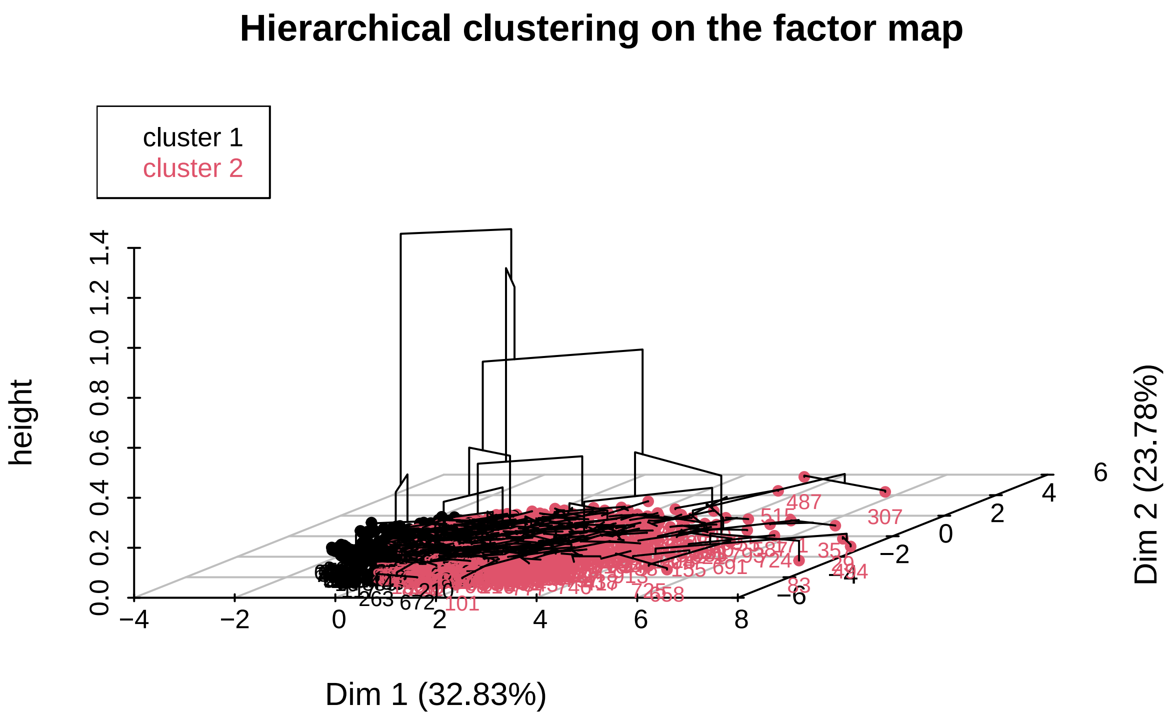
Hierarchical clustering after dimensionality reduction

We could do this as we did with K-means but hclust has this build in function which knows where to get the PCA's and then cluster them.

```
res_hcpc = res_pca %>%
  HCPC(nb.clust = 2, graph = FALSE)
```

Then we plot it

```
res_hcpc %>%
  plot(choice = "3D.map")
```



The plot is a bit of a mess because it also plots the trees and it is hard to distinguish where the black dots starts and where the red begins, so it seems like the clusters are not ideal, this might again be due to the fact stated above.

```
data[,"cluster_hclust_pca2"] <- res_hcpc$data.clust$clust

table(data$cluster_hclust_pca2, data$status); table(data$cluster_hclust2, data$status)
```

```
##
##     acquired closed
##   1       59    115
##   2      534    210
```

```
##
##     acquired closed
##   1       57    117
##   2      536    208
```

After using PCA it looks like "acquired" is separated more clearly into one cluster, but is still having trouble separating "closed" into one cluster. The variables used in this analysis wasnt able to capture the effects that are crucial to whether a start up is getting acquired or closed. This might be due to that fact that a lot more variables plays a part in this and maybe also a bit of luck, which cant be quantified.

Now we move on to supervised machine learning.

SML

We start by creating a new dataset and renaming our "status" variable to "y" for convenience, because this is the variable we want to predict.

```
data_sml= data %>%
  rename(y = status) %>%
  select(y, total, funding_rounds, relationships, milestones, is_top500, `is_0:3`,
         age_last_funding_year)
```

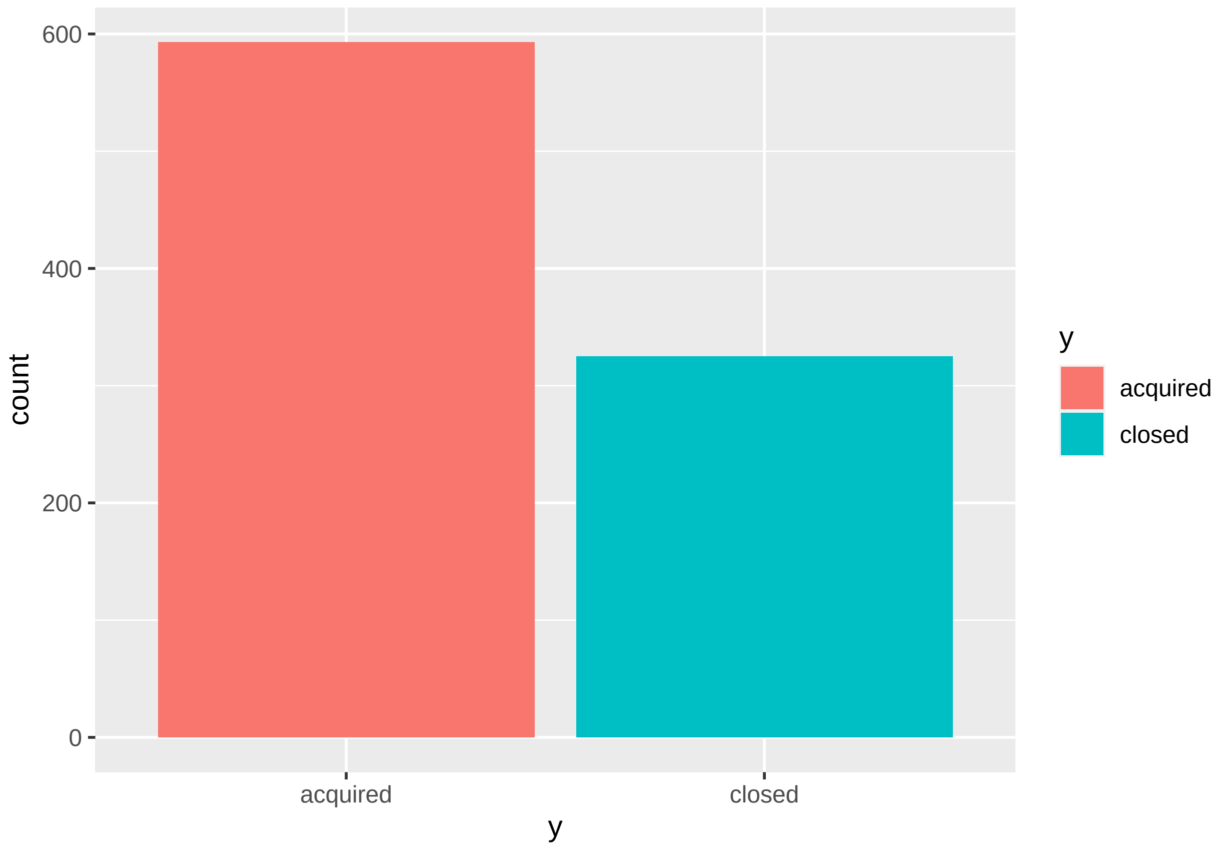
```r
glimpse(data_sml)
```

```
## Rows: 918
## Columns: 8
## $ y                     <chr> "acquired", "acquired", "acquired", "acquired", ~
## $ total                 <dbl> 375, 40100, 2600, 40000, 1300, 7500, 26000, 3410~
## $ funding_rounds        <dbl> 3, 4, 1, 3, 2, 1, 3, 3, 3, 3, 3, 5, 1, 3, 5, 1, ~
## $ relationships         <dbl> 3, 9, 5, 5, 2, 3, 6, 25, 13, 14, 22, 8, 0, 15, 1~
## $ milestones            <dbl> 3, 1, 2, 1, 1, 1, 2, 3, 4, 4, 3, 2, 0, 3, 1, 0, ~
## $ is_top500             <dbl> 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, ~
## $ `is_0:3`              <dbl> 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, ~
## $ age_last_funding_year <dbl> 3.0027, 9.9973, 1.0329, 5.3151, 1.6685, 4.5452, ~
```

Training & Test split

Just to get an idea how the y variable is split between acquired and closed we can do a histogram

```r
data_sml %>%
  ggplot(aes(y, fill= y)) +
  geom_bar()
```



We can see there are more observations under acquired than closed, so to be certain that the training and test data will be as similar as possible we set the strata argument equal to y when we split the data.

```r
set.seed(123)

data_split <- initial_split(data_sml, prop = 0.75, strata = y)
```

```
data_train <- data_split  %>%  training()
data_test <- data_split %>% testing()
```

Preprocessing recipe

To automate the process we create a recipe that will procces the data using: Step_log, which will log transform data (usefull as many of our variables are right-skewed) step_center, Centers all numeric variables to mean = 0. step_scale, scales all numeric variables to sd = 1. step_dummy converts categorical/factor variables into binary dummies.

```
data_recipe <- data_train %>%
  recipe(y ~.) %>%
  step_log(total, funding_rounds) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes()) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  prep()


summary(data_recipe)
```

```
## # A tibble: 8 x 4
##   variable              type    role      source
##   <chr>                 <chr>   <chr>     <chr>
## 1 total                 numeric predictor original
## 2 funding_rounds        numeric predictor original
## 3 relationships         numeric predictor original
## 4 milestones            numeric predictor original
## 5 is_top500             numeric predictor original
## 6 is_0:3                numeric predictor original
## 7 age_last_funding_year numeric predictor original
## 8 y                     nominal outcome   original
```

Defining the models

We will now start specifying our models as follows:

1. pick a model type

- Logistic regression
- Decision tree
- XGBoost
- K- nearest neighboors
- Random forrest

2. set the engine: the softwear used to fit the model

3. set the mode: which in this case will be classification.

Logistic Regression

```
model_lg <- logistic_reg(mode = 'classification') %>%
  set_engine('glm', family = binomial)
```

Decision tree

```r
model_dt <- decision_tree(mode = 'classification',
                          cost_complexity = tune(),
                          tree_depth = tune(),
                          min_n = tune()
                          ) %>%
  set_engine('rpart')
```

Extreme Gradient Boosted Tree (XGBoost)

```r
model_xg <- boost_tree(mode = 'classification',
                       trees = 100,
                       mtry = tune(),
                       min_n = tune(),
                       tree_depth = tune(),
                       learn_rate = tune()
                       ) %>%
  set_engine("xgboost")
```

K-nearest neighbor

```r
model_knn <-
  nearest_neighbor(neighbors = 4) %>% # we can adjust the number of neighbors
  set_engine("kknn") %>%
  set_mode("classification")
```

Random forest

```r
model_rf <-
  rand_forest() %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")
```

Define workflow

We will now define the workflow of the model by adding first the recipe to a general workflow, and then using this to create a workflow for each model.

```r
workflow_general <- workflow() %>%
  add_recipe(data_recipe)

workflow_lg <- workflow_general %>%
  add_model(model_lg)

workflow_dt <- workflow_general %>%
  add_model(model_dt)

workflow_xg <- workflow_general %>%
  add_model(model_xg)

workflow_knn <- workflow_general %>%
  add_model(model_knn)
```

```r
workflow_rf <- workflow_general %>%
  add_model(model_rf)
```

Hyperparameter Tuning

As the parameters in the decision tree and XGBoost model are set to tune(), we will now find the optimal values for the parameters.

Validation Sampling (N-fold crossvlidation)

We use k-fold crossvalidation to build a set of 5 validation folds with the function vfold_cv. We also use stratified sampling by setting the strata argument to y. We set repeats equal to 3. We dont have to use boosttraps as we have enogh observations.

```r
set.seed(100)

data_resample <- data_train %>%
  vfold_cv(strata = y,
           v = 3,
           repeats = 3)
```

Hyperparameter Tuning: Decision Tree

First we tune the decision tree, using the tune_grid function where we first specify the workflow, next we give it the resampled data, and last the grid means give us 10 different versions of every tuneable parameters.

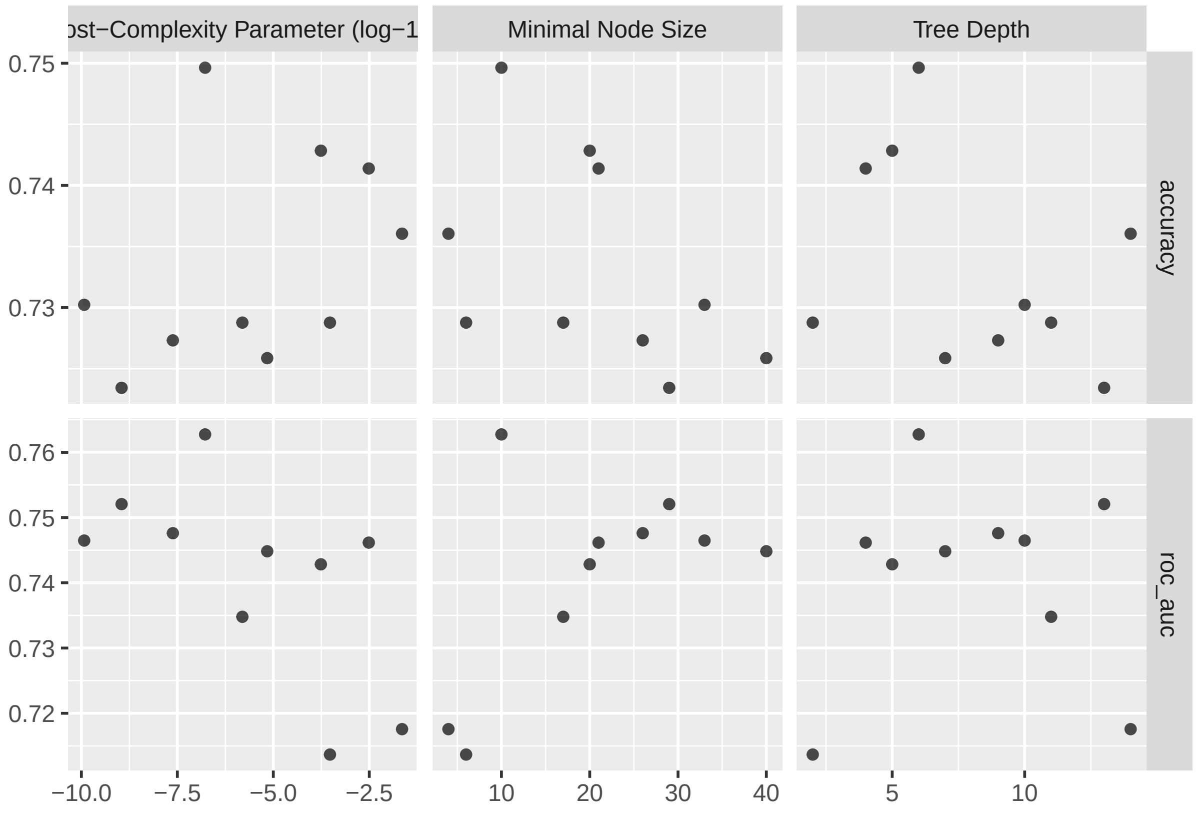
```r
tune_dt <-
  tune_grid(
    workflow_dt,
    resamples = data_resample,
    grid = 10
  )
```

```
## Warning: package 'rlang' was built under R version 4.0.2
```

```
## Warning: package 'vctrs' was built under R version 4.0.2
```

We can now see that the tuned parameters are plotted with different values compared to the accuracy and roc-auc values.

```r
tune_dt %>% autoplot()
```



We will now use the select_best function to select the parameters that maximize the area under the roc curve.

```r
best_param_dt <- tune_dt %>% select_best(metric = 'roc_auc')
best_param_dt
```

```
## # A tibble: 1 x 4
##   cost_complexity tree_depth min_n .config
##             <dbl>      <int> <int> <chr>
## 1     0.000000167          6    10 Preprocessor1_Model08
```

```r
tune_dt %>% show_best(metric = 'roc_auc', n = 1)
```

```
## # A tibble: 1 x 9
##   cost_complexity tree_depth min_n .metric .estimator  mean     n std_err
##             <dbl>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl>
## 1     0.000000167          6    10 roc_auc binary     0.763     9  0.0113
## # ... with 1 more variable: .config <chr>
```

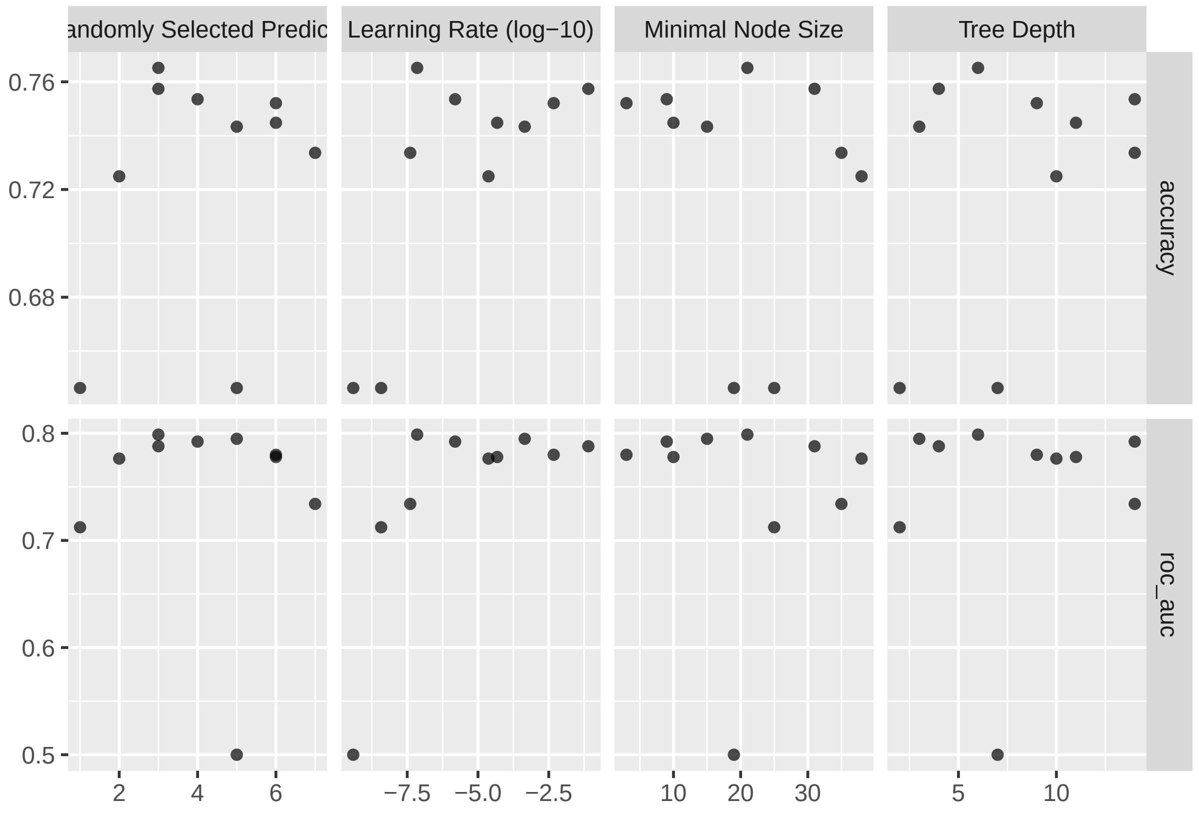
Hyperparameter Tuning: Random Forest

We now do the same for the Random Forest model again using the tune_grid function where we first specify the workflow, next we give it the resampled data, and last the grid means give us 10 different versions of every tuneable parameters.

```r
tune_xg <-
  tune_grid(
    workflow_xg,
    resamples = data_resample,
    grid = 10
  )
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```
tune_xg %>% autoplot()
```



```
best_param_xg <- tune_xg %>% select_best(metric = 'roc_auc')
best_param_xg
```

```
## # A tibble: 1 x 5
##    mtry min_n tree_depth   learn_rate .config
##   <int> <int>      <int>        <dbl> <chr>
## 1     3    21          6 0.0000000709 Preprocessor1_Model08
```

```
tune_xg %>% show_best(metric = 'roc_auc', n = 1)
```

```
## # A tibble: 1 x 10
##    mtry min_n tree_depth   learn_rate .metric .estimator  mean     n std_err
##   <int> <int>      <int>        <dbl> <chr>   <chr>      <dbl> <int>   <dbl>
## 1     3    21          6 0.0000000709 roc_auc binary     0.799     9 0.00785
## # ... with 1 more variable: .config <chr>
```

Fit models with tuned hyperparameters

We now fit the best parameters into the workflow of the two models.

```
workflow_final_dt <- workflow_dt %>%
  finalize_workflow(parameters = best_param_dt)

workflow_final_xg <- workflow_xg %>%
  finalize_workflow(parameters = best_param_xg)
```

Evaluate models

here we us the resampled data to evaluate the models.

Logistic regression

We use our workflow object to perform resampling. Furthermore, we use metric_set() to choose some common classification performance metrics provided by the yardstick package. Visit yardsticks reference to see the complete list of all possible metrics.

Note that Cohen's kappa coefficient ( ) is a similar measure to accuracy, but is normalized by the accuracy that would be expected by chance alone and is very useful when one or more classes have large frequency distributions. The higher the value, the better.

```r
log_res <-
  workflow_lg %>%
  fit_resamples(
    resamples = data_resample,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE)
    )

log_res %>% collect_metrics(summarize = TRUE)
```

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  binary     0.768     9 0.00768 Preprocessor1_Model1
## 2 f_meas    binary     0.831     9 0.00526 Preprocessor1_Model1
## 3 kap       binary     0.464     9 0.0198  Preprocessor1_Model1
## 4 precision binary     0.786     9 0.00796 Preprocessor1_Model1
## 5 recall    binary     0.883     9 0.00915 Preprocessor1_Model1
## 6 roc_auc   binary     0.804     9 0.0108  Preprocessor1_Model1
## 7 sens      binary     0.883     9 0.00915 Preprocessor1_Model1
## 8 spec      binary     0.558     9 0.0229  Preprocessor1_Model1
```

Model coefficients we save model coefficients for a fitted model object from a workflow using the same code as before only with one exception using extract and our new function.

```r
get_model <- function(x) {
  pull_workflow_fit(x) %>% tidy()
}



log_res_2 <-
  workflow_lg %>%
  fit_resamples(
    resamples = data_resample,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE,
      extract = get_model)
    )
```

```
## ! Fold1, Repeat1: internal: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Pleas...

## ! Fold2, Repeat1: internal: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Pleas...

## ! Fold3, Repeat1: internal: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Pleas...

## ! Fold1, Repeat2: internal: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Pleas...

## ! Fold2, Repeat2: internal: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Pleas...

## ! Fold3, Repeat2: internal: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Pleas...

## ! Fold1, Repeat3: internal: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Pleas...

## ! Fold2, Repeat3: internal: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Pleas...

## ! Fold3, Repeat3: internal: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Pleas...
```

We can now see the estimate std.error and t-statistic/p-value for each variable.

```
log_res_2$.extracts[[1]][[1]]
```

```
## [[1]]
## # A tibble: 8 x 5
##   term                  estimate std.error statistic  p.value
##   <chr>                    <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)            -1.05       0.146    -7.20  6.00e-13
## 2 total                  -0.331      0.168    -1.97  4.90e- 2
## 3 funding_rounds         -0.147      0.147    -1.00  3.16e- 1
## 4 relationships          -1.46       0.261    -5.62  1.95e- 8
## 5 milestones             -0.388      0.162    -2.40  1.64e- 2
## 6 is_top500              -0.355      0.128    -2.78  5.36e- 3
## 7 `is_0:3`                0.0519     0.140     0.370 7.11e- 1
## 8 age_last_funding_year   0.176      0.157     1.13  2.60e- 1
```

```
all_coef <- map_dfr(log_res_2$.extracts, ~ .x[[1]][[1]])
```

Show all of the resample coefficients for a single predictor:

```
filter(all_coef, term == "relationships")
```

```
## # A tibble: 9 x 5
##   term          estimate std.error statistic      p.value
##   <chr>            <dbl>     <dbl>     <dbl>        <dbl>
## 1 relationships   -1.46      0.261     -5.62 0.0000000195
## 2 relationships   -1.12      0.245     -4.58 0.00000471
## 3 relationships   -0.982     0.256     -3.84 0.000121
## 4 relationships   -1.24      0.260     -4.79 0.00000168
## 5 relationships   -1.33      0.268     -4.97 0.000000664
## 6 relationships   -0.997     0.235     -4.23 0.0000230
## 7 relationships   -1.09      0.243     -4.48 0.00000732
## 8 relationships   -1.10      0.255     -4.29 0.0000175
## 9 relationships   -1.38      0.258     -5.35 0.0000000863
```

Decision tree

```
dt_res <-
  workflow_final_dt %>%
  fit_resamples(
    resamples = data_resample,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(save_pred = TRUE)
    )

dt_res %>% collect_metrics(summarize = TRUE)
```

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  binary     0.750     9 0.00418 Preprocessor1_Model1
## 2 f_meas    binary     0.818     9 0.00360 Preprocessor1_Model1
## 3 kap       binary     0.420     9 0.0105  Preprocessor1_Model1
## 4 precision binary     0.770     9 0.00521 Preprocessor1_Model1
## 5 recall    binary     0.874     9 0.0104  Preprocessor1_Model1
## 6 roc_auc   binary     0.763     9 0.0113  Preprocessor1_Model1
## 7 sens      binary     0.874     9 0.0104  Preprocessor1_Model1
## 8 spec      binary     0.523     9 0.0182  Preprocessor1_Model1
```

XGboost

```
xgb_res <-
  workflow_final_xg %>%
  fit_resamples(
    resamples = data_resample,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(save_pred = TRUE)
    )

xgb_res %>% collect_metrics(summarize = TRUE)
```

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  binary     0.760     9 0.00645 Preprocessor1_Model1
## 2 f_meas    binary     0.828     9 0.00452 Preprocessor1_Model1
## 3 kap       binary     0.440     9 0.0157  Preprocessor1_Model1
## 4 precision binary     0.773     9 0.00524 Preprocessor1_Model1
## 5 recall    binary     0.890     9 0.00604 Preprocessor1_Model1
## 6 roc_auc   binary     0.797     9 0.00739 Preprocessor1_Model1
## 7 sens      binary     0.890     9 0.00604 Preprocessor1_Model1
## 8 spec      binary     0.523     9 0.0136  Preprocessor1_Model1
```

KNN

```
knn_res <-
  workflow_knn %>%
  fit_resamples(
    resamples = data_resample,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(save_pred = TRUE)
    )
```

```
## Warning: package 'kknn' was built under R version 4.0.2
```

```
knn_res %>% collect_metrics(summarize = TRUE)
```

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  binary     0.690     9 0.00619 Preprocessor1_Model1
## 2 f_meas    binary     0.765     9 0.00469 Preprocessor1_Model1
## 3 kap       binary     0.311     9 0.0150  Preprocessor1_Model1
## 4 precision binary     0.750     9 0.00588 Preprocessor1_Model1
## 5 recall    binary     0.782     9 0.00730 Preprocessor1_Model1
## 6 roc_auc   binary     0.728     9 0.00748 Preprocessor1_Model1
## 7 sens      binary     0.782     9 0.00730 Preprocessor1_Model1
## 8 spec      binary     0.524     9 0.0158  Preprocessor1_Model1
```

Random forrest

```
rf_res <-
  workflow_rf %>%
  fit_resamples(
    resamples = data_resample,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(save_pred = TRUE)
    )

rf_res %>%  collect_metrics(summarize = TRUE)
```

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  binary     0.778     9 0.00542 Preprocessor1_Model1
## 2 f_meas    binary     0.840     9 0.00332 Preprocessor1_Model1
## 3 kap       binary     0.483     9 0.0150  Preprocessor1_Model1
## 4 precision binary     0.788     9 0.00675 Preprocessor1_Model1
## 5 recall    binary     0.899     9 0.00579 Preprocessor1_Model1
## 6 roc_auc   binary     0.808     9 0.00625 Preprocessor1_Model1
## 7 sens      binary     0.899     9 0.00579 Preprocessor1_Model1
## 8 spec      binary     0.556     9 0.0195  Preprocessor1_Model1
```

Compare performance

We get a summary for the performed models. We add the model name to each metric to keep the models appart from each other later on.

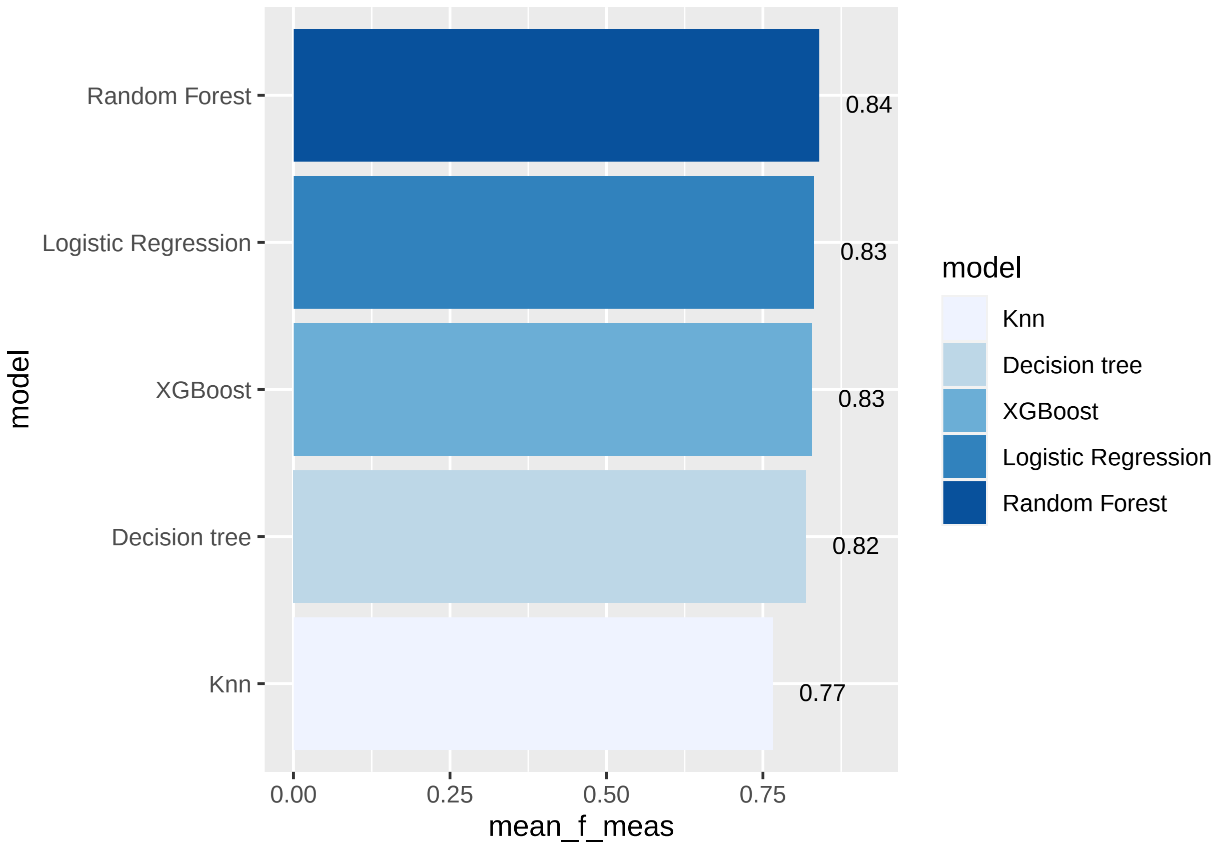
```
log_metrics <-
  log_res %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Logistic Regression")

rf_metrics <-
  rf_res %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Random Forest")

xgb_metrics <-
  xgb_res %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "XGBoost")

knn_metrics <-
  knn_res %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Knn")

dt_metrics <-
  dt_res %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Decision tree")
```

We now bind the rows for the above metricies and create a dataframe, we then change the data structure and show the mean_f_meas score for each model which include the precision and recall score.

Precision quantifies the number of positive class predictions that actually belong to the positive class. Recall quantifies the number of positive class predictions made out of all positive examples in the dataset. F-Measure provides a single score that balances both the concerns of precision and recall in one number.

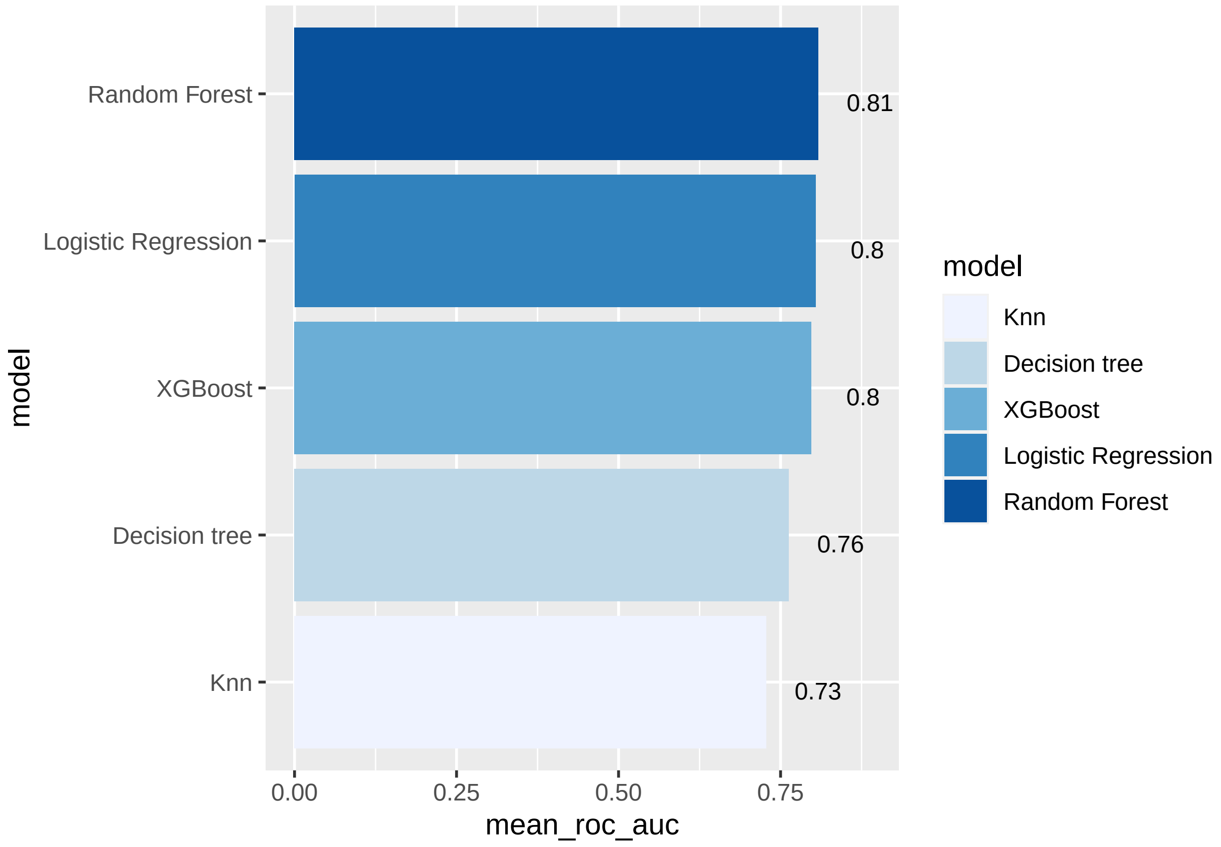
```
model_compare <- bind_rows(
                          log_metrics,
                           rf_metrics,
                           xgb_metrics,
                           knn_metrics,
                          dt_metrics,
                           )



model_comp <-
```

```
  model_compare %>%
  select(model, .metric, mean, std_err) %>%
  pivot_wider(names_from = .metric, values_from = c(mean, std_err))


model_comp %>%
  arrange(mean_f_meas) %>%
  mutate(model = fct_reorder(model, mean_f_meas)) %>%
  ggplot(aes(model, mean_f_meas, fill=model)) +
  geom_col() +
  coord_flip() +
  scale_fill_brewer(palette = "Blues") +
   geom_text(
     size = 3,
     aes(label = round(mean_f_meas, 2), y = mean_f_meas + 0.08),
     vjust = 1
  )
```



We will now show the mean area under the curve for each of the models.

```
model_comp %>%
  arrange(mean_roc_auc) %>%
  mutate(model = fct_reorder(model, mean_roc_auc)) %>%
  ggplot(aes(model, mean_roc_auc, fill=model)) +
  geom_col() +
  coord_flip() +
  scale_fill_brewer(palette = "Blues") +
     geom_text(
     size = 3,
     aes(label = round(mean_roc_auc, 2), y = mean_roc_auc + 0.08),
     vjust = 1
  )
```



Choose model

Log-reg model

Performance metrics Show average performance over all folds (note that we use log_res):

```
log_res %>%  collect_metrics(summarize = TRUE)
```

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  binary     0.768     9 0.00768 Preprocessor1_Model1
## 2 f_meas    binary     0.831     9 0.00526 Preprocessor1_Model1
## 3 kap       binary     0.464     9 0.0198  Preprocessor1_Model1
## 4 precision binary     0.786     9 0.00796 Preprocessor1_Model1
## 5 recall    binary     0.883     9 0.00915 Preprocessor1_Model1
## 6 roc_auc   binary     0.804     9 0.0108  Preprocessor1_Model1
## 7 sens      binary     0.883     9 0.00915 Preprocessor1_Model1
## 8 spec      binary     0.558     9 0.0229  Preprocessor1_Model1
```

Show performance for every single fold:

```
log_res %>%  collect_metrics(summarize = FALSE)
```

```
## # A tibble: 72 x 6
##    id      id2   .metric   .estimator .estimate .config
##    <chr>   <chr> <chr>     <chr>          <dbl> <chr>
##  1 Repeat1 Fold1 recall    binary         0.872 Preprocessor1_Model1
##  2 Repeat1 Fold1 precision binary         0.796 Preprocessor1_Model1
##  3 Repeat1 Fold1 f_meas    binary         0.832 Preprocessor1_Model1
##  4 Repeat1 Fold1 accuracy  binary         0.773 Preprocessor1_Model1
##  5 Repeat1 Fold1 kap       binary         0.483 Preprocessor1_Model1
##  6 Repeat1 Fold1 sens      binary         0.872 Preprocessor1_Model1
##  7 Repeat1 Fold1 spec      binary         0.593 Preprocessor1_Model1
##  8 Repeat1 Fold1 roc_auc   binary         0.789 Preprocessor1_Model1
##  9 Repeat1 Fold2 recall    binary         0.845 Preprocessor1_Model1
## 10 Repeat1 Fold2 precision binary         0.796 Preprocessor1_Model1
## # ... with 62 more rows
```

Collect model predictions To obtain the actual model predictions, we use the function collect_predictions and save the result as log_pred:

```
log_pred <-
  log_res %>%
  collect_predictions()
```

Confusion Matrix

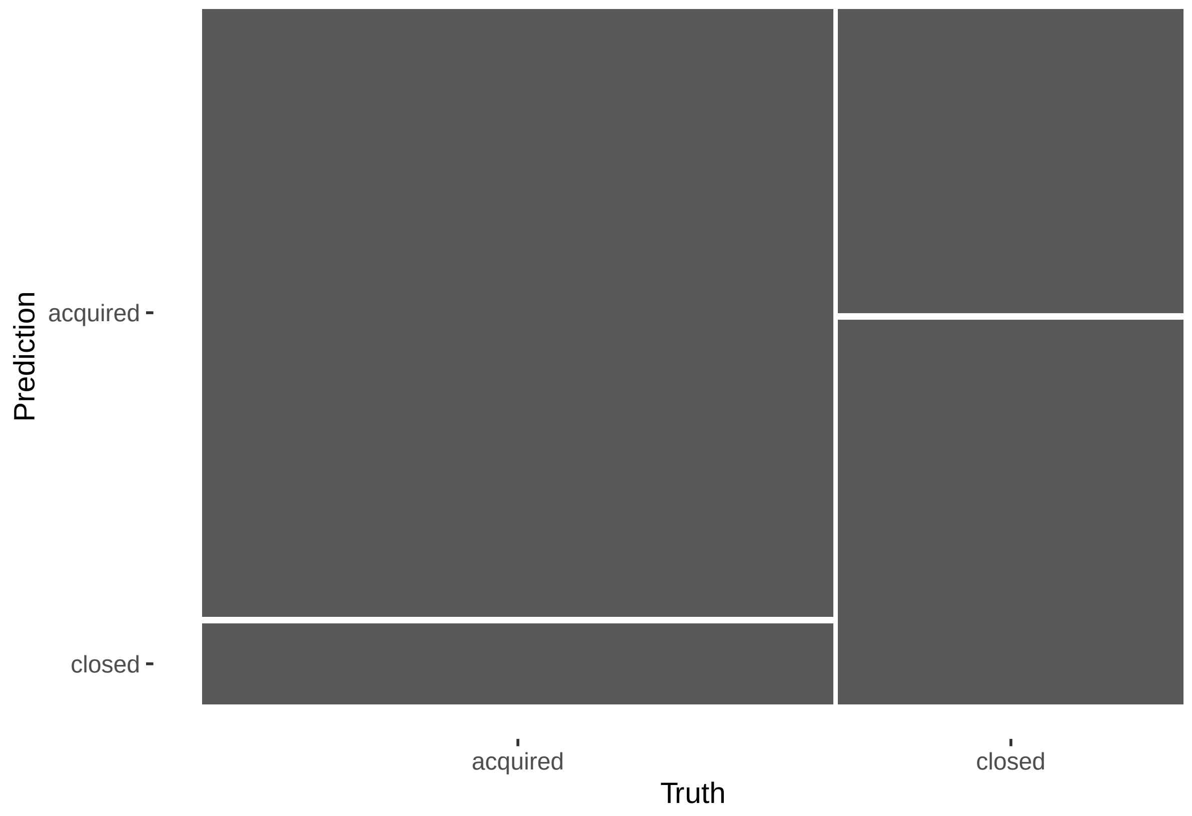
We can now use our collected predictions to make a confusion matrix

```
log_pred %>%
  conf_mat(y, .pred_class)
```

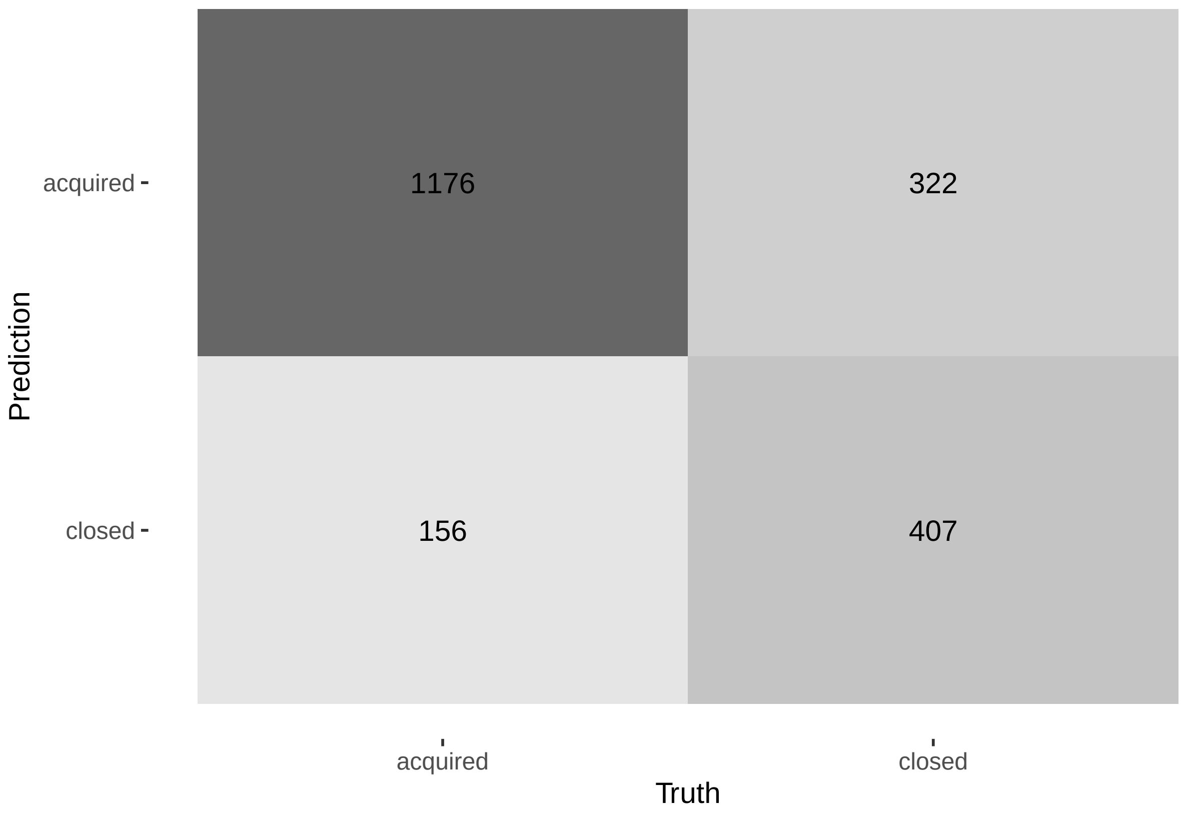
```
##           Truth
## Prediction acquired closed
##   acquired     1176    322
##   closed        156    407
```

And we can also visualize

```
log_pred %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "mosaic")
```



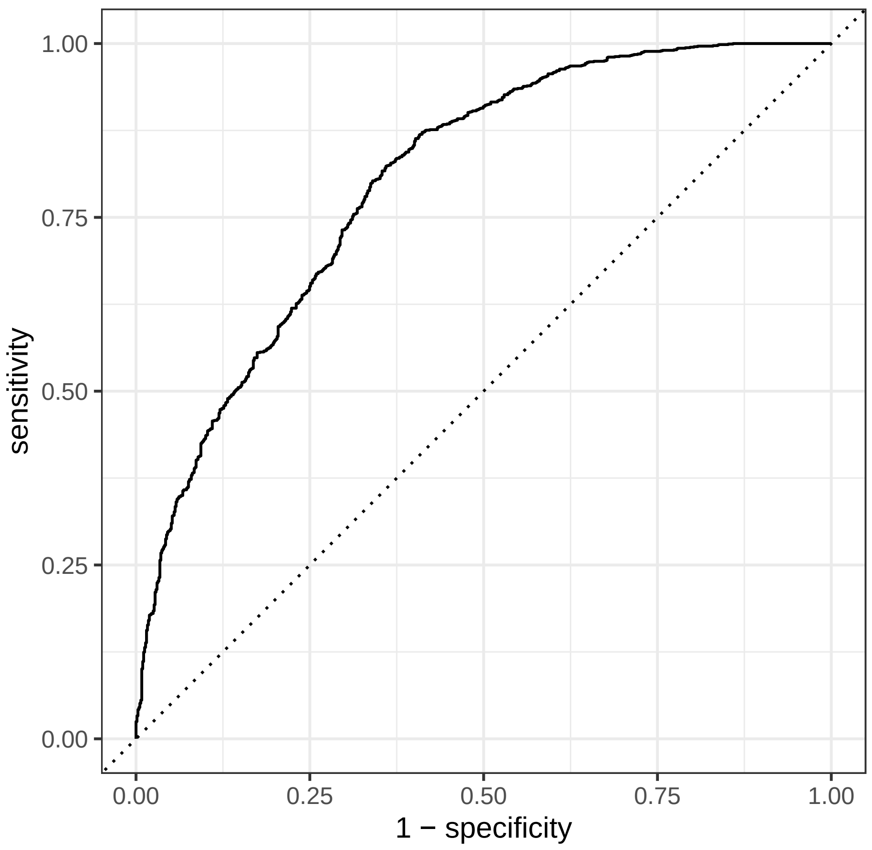
```r
log_pred %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "heatmap")
```



ROC curve

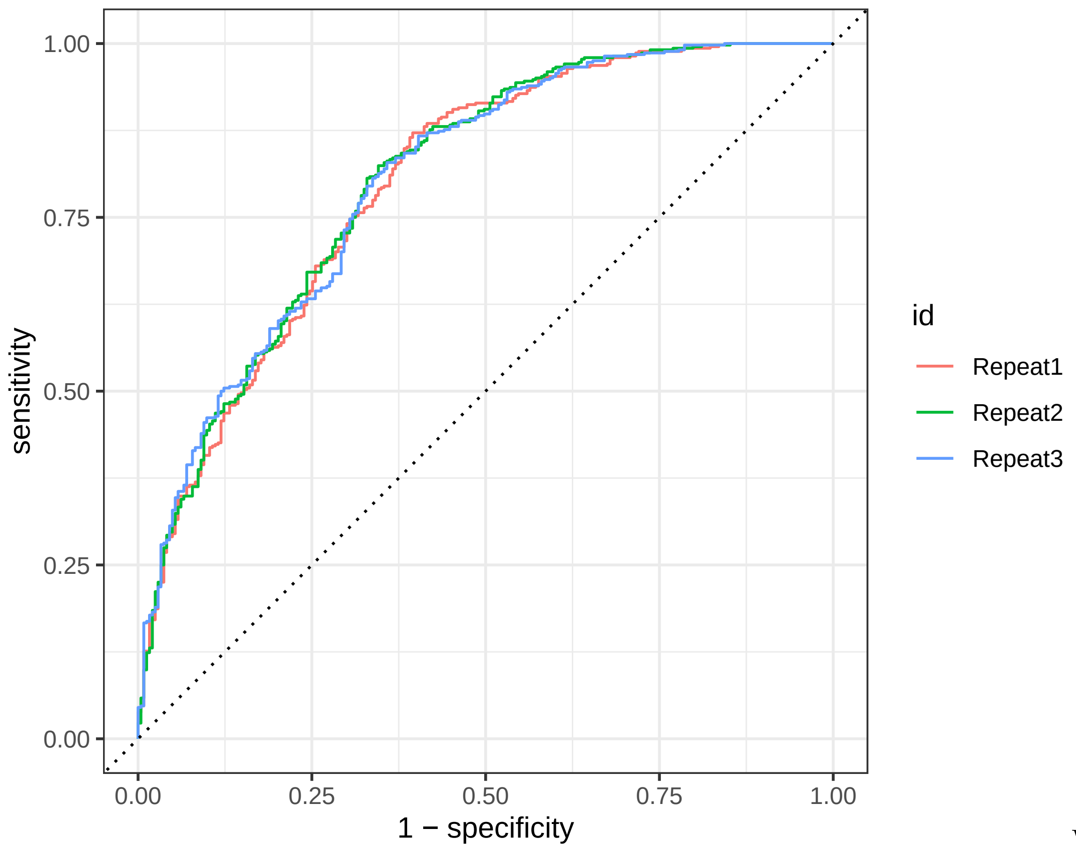
We will now create the ROC curve with 1 - specificity on the x-axis (false positive fraction = FP/(FP+TN)) and sensitivity on the y axis (true positive fraction = TP/(TP+FN)).

```
log_pred %>%
  roc_curve(y, .pred_acquired) %>%
  autoplot()
```



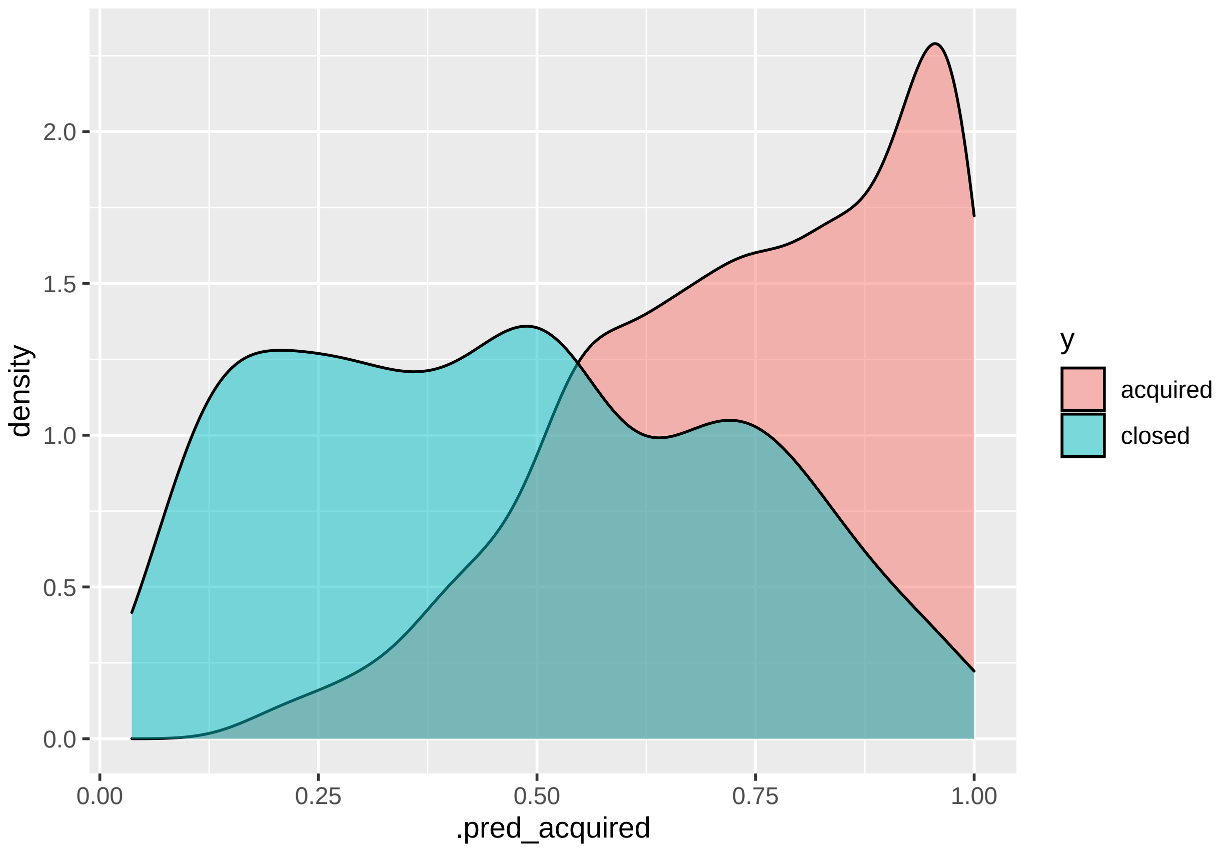
We can create the same curve for each of the folds used from the resampled data.

```
log_pred %>%
  group_by(id) %>%
    roc_curve(y, .pred_acquired) %>%
  autoplot()
```



We can plot the predicted probability distributions for our two classes.

```r
log_pred %>%
  ggplot() +
  geom_density(aes(x = .pred_acquired,
                   fill = y),
               alpha = 0.5)
```



XGboost model

Performance metrics

Show average performance over all folds.

```
xgb_res %>%  collect_metrics(summarize = TRUE)
```

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  binary     0.760     9 0.00645 Preprocessor1_Model1
## 2 f_meas    binary     0.828     9 0.00452 Preprocessor1_Model1
## 3 kap       binary     0.440     9 0.0157  Preprocessor1_Model1
## 4 precision binary     0.773     9 0.00524 Preprocessor1_Model1
## 5 recall    binary     0.890     9 0.00604 Preprocessor1_Model1
## 6 roc_auc   binary     0.797     9 0.00739 Preprocessor1_Model1
## 7 sens      binary     0.890     9 0.00604 Preprocessor1_Model1
## 8 spec      binary     0.523     9 0.0136  Preprocessor1_Model1
```

Show performance for every single fold:

```
xgb_res %>%  collect_metrics(summarize = FALSE)
```

```
## # A tibble: 72 x 6
##    id      id2   .metric   .estimator .estimate .config
##    <chr>   <chr> <chr>     <chr>          <dbl> <chr>
##  1 Repeat1 Fold1 recall    binary         0.885 Preprocessor1_Model1
##  2 Repeat1 Fold1 precision binary         0.771 Preprocessor1_Model1
##   3 Repeat1 Fold1 f_meas    binary         0.824 Preprocessor1_Model1
##   4 Repeat1 Fold1 accuracy  binary         0.755 Preprocessor1_Model1
##   5 Repeat1 Fold1 kap       binary         0.430 Preprocessor1_Model1
##   6 Repeat1 Fold1 sens      binary         0.885 Preprocessor1_Model1
##   7 Repeat1 Fold1 spec      binary         0.519 Preprocessor1_Model1
##   8 Repeat1 Fold1 roc_auc   binary         0.785 Preprocessor1_Model1
##   9 Repeat1 Fold2 recall    binary         0.872 Preprocessor1_Model1
##  10 Repeat1 Fold2 precision binary         0.759 Preprocessor1_Model1
## # ... with 62 more rows
```

Collect model predictions

To obtain the actual model predictions, we use the function collect_predictions and save the result as xgb_pred:

```
xgb_pred <-
  xgb_res %>%
  collect_predictions()
```

Confusion Matrix

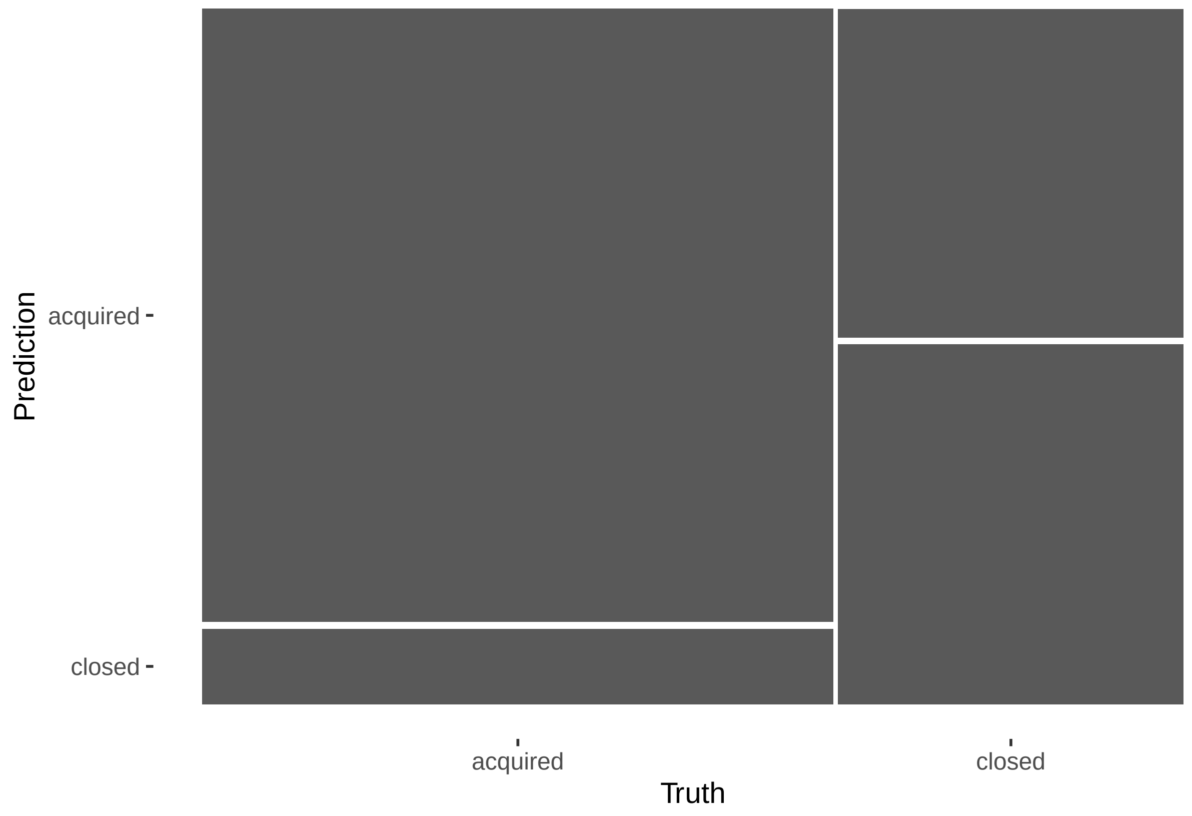
We can now use our collected predictions to make a confusion matrix

```
xgb_pred %>%
  conf_mat(y, .pred_class)
```

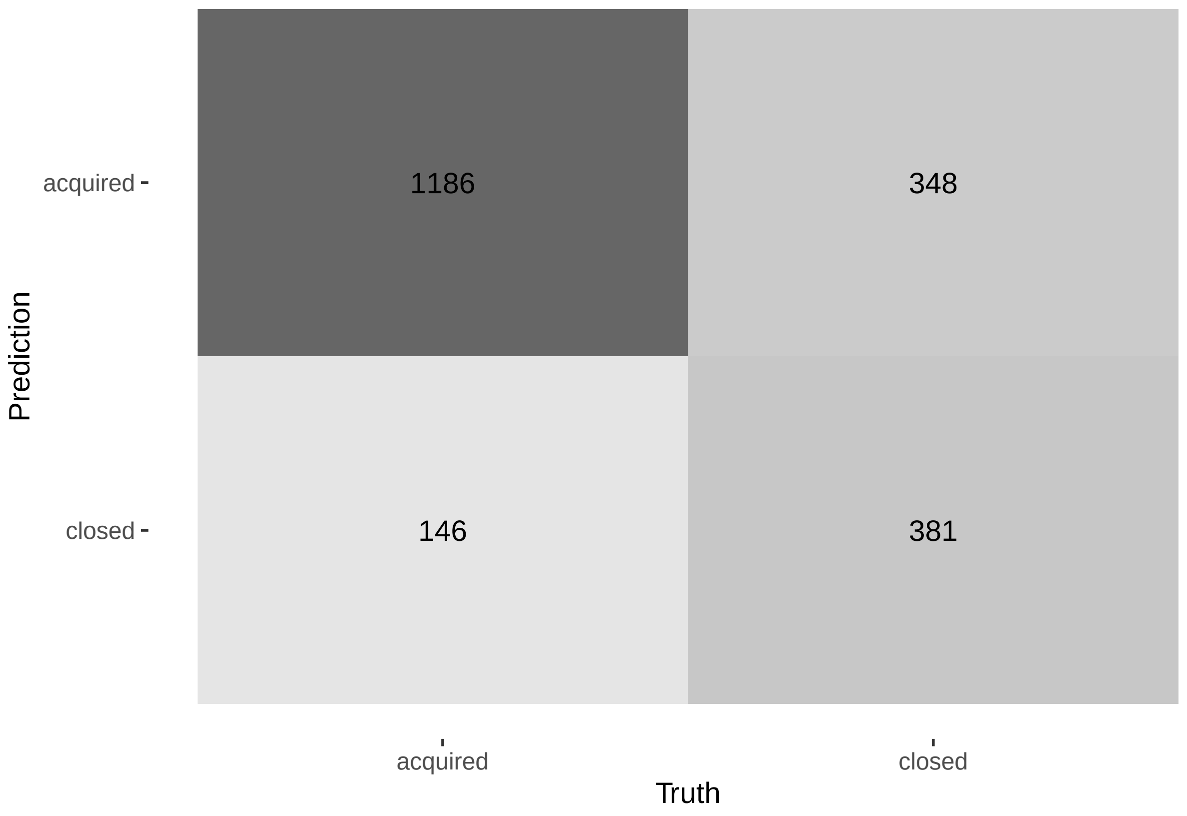
```
##             Truth
## Prediction acquired closed
##   acquired     1186    348
##   closed        146    381
```

And visualize it again

```
xgb_pred %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "mosaic")
```



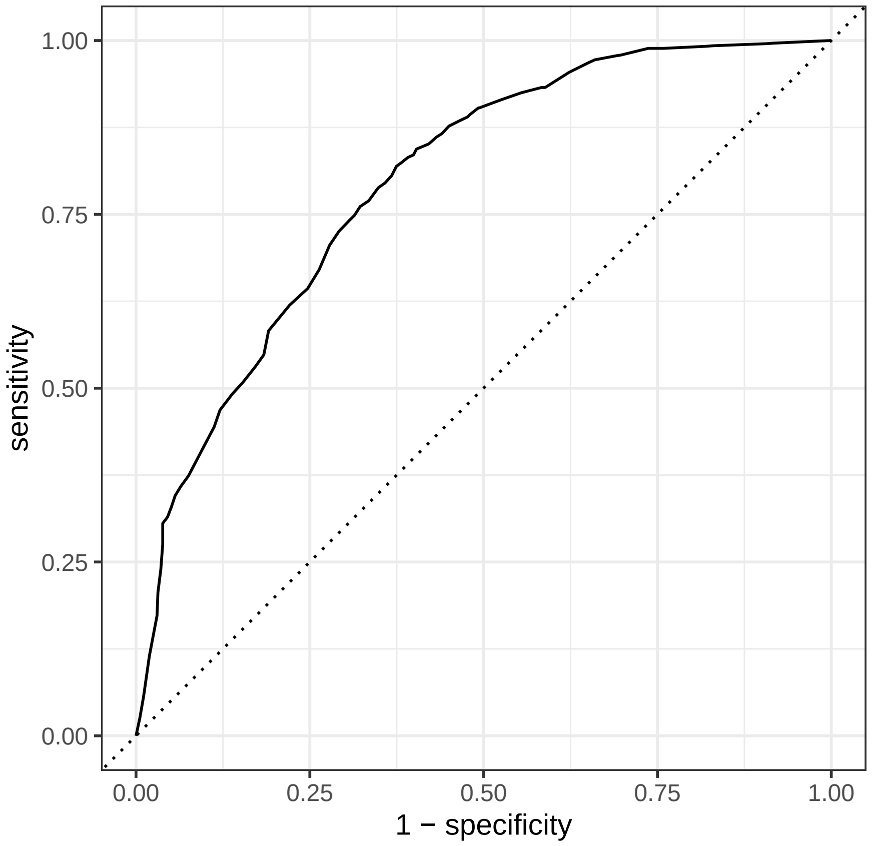
```
xgb_pred %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "heatmap")
```



ROC curve

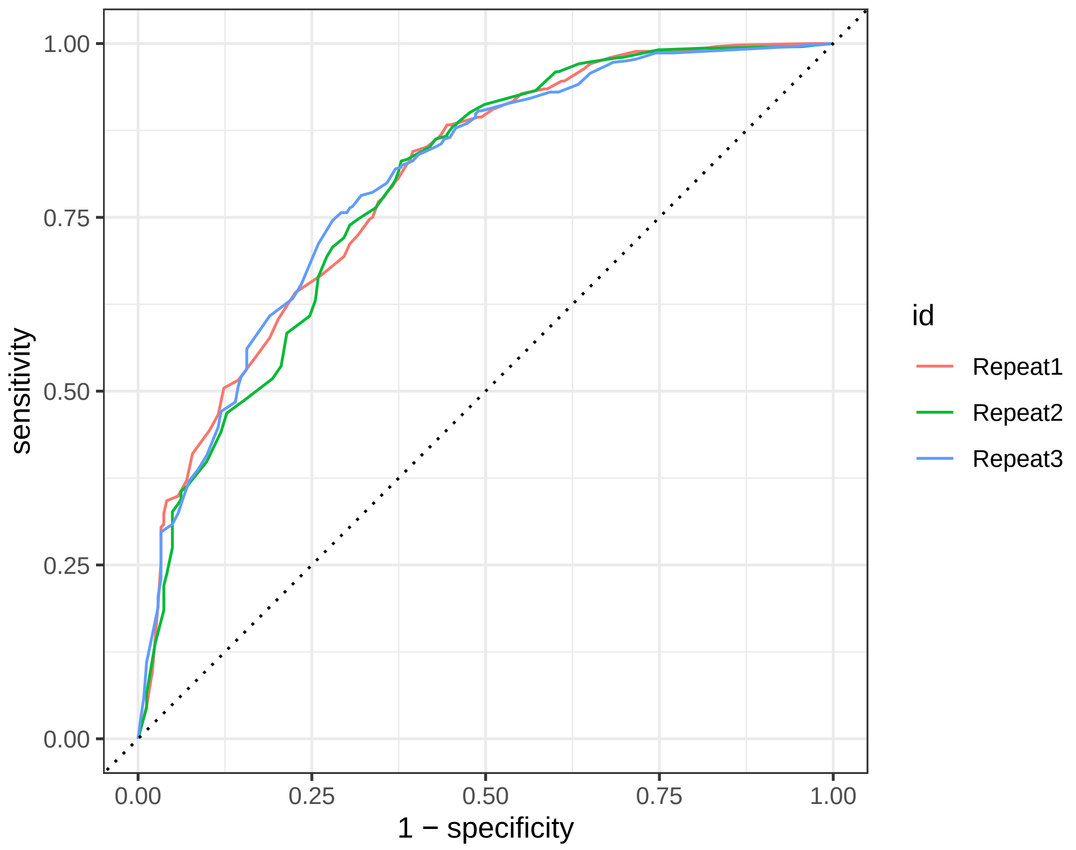
We will now create the ROC curve with 1 - specificity on the x-axis (false positive fraction = FP/(FP+TN)) and sensitivity on the y axis (true positive fraction = TP/(TP+FN)).

```
xgb_pred %>%
  roc_curve(y, .pred_acquired) %>%
  autoplot()
```



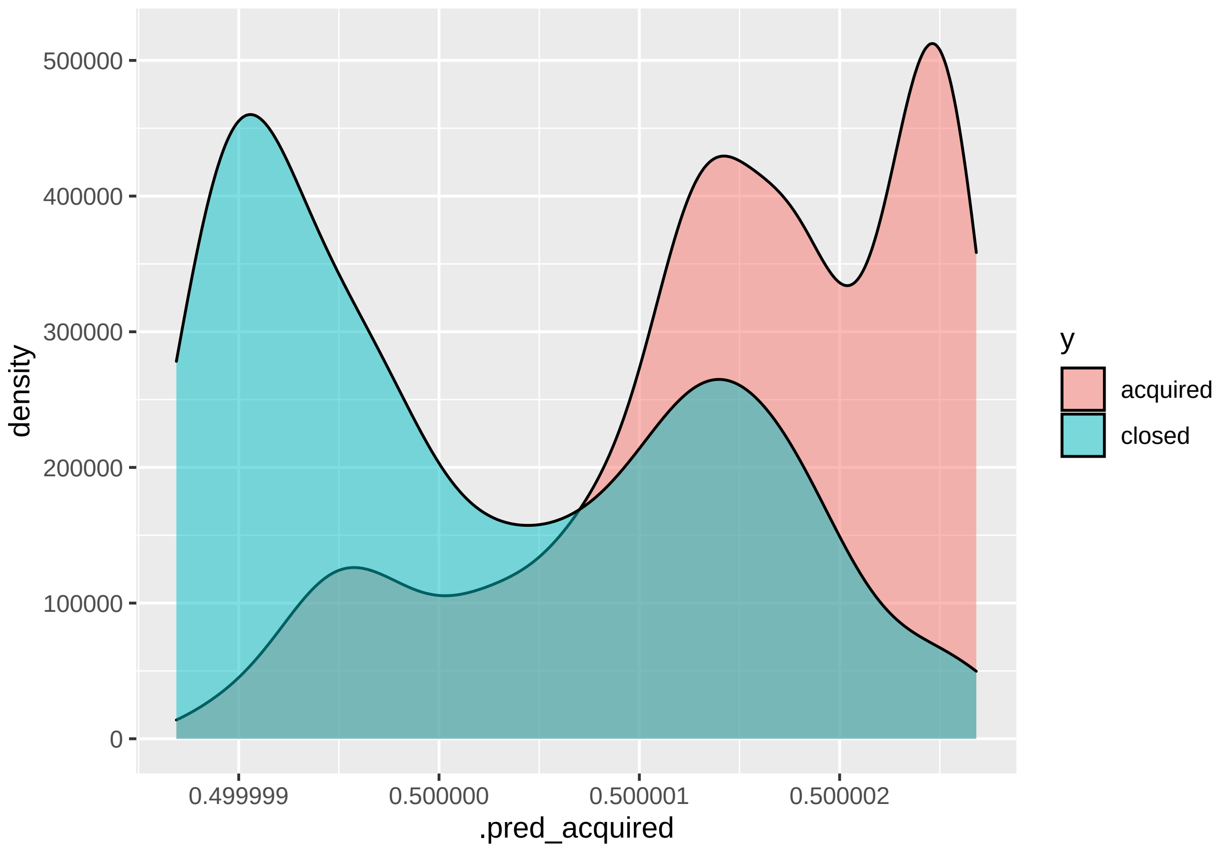
And now using the folds again.

```
xgb_pred %>%
  group_by(id) %>%
    roc_curve(y, .pred_acquired) %>%
  autoplot()
```



We again show the probability distributions for our two classes.

```
xgb_pred %>%
  ggplot() +
  geom_density(aes(x = .pred_acquired,
                   fill = y),
               alpha = 0.5)
```



Random forrest model

Performance metrics

Show average performance over all folds:

```
rf_res %>%  collect_metrics(summarize = TRUE)
```

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  binary     0.778     9 0.00542 Preprocessor1_Model1
## 2 f_meas    binary     0.840     9 0.00332 Preprocessor1_Model1
## 3 kap       binary     0.483     9 0.0150  Preprocessor1_Model1
## 4 precision binary     0.788     9 0.00675 Preprocessor1_Model1
## 5 recall    binary     0.899     9 0.00579 Preprocessor1_Model1
## 6 roc_auc   binary     0.808     9 0.00625 Preprocessor1_Model1
## 7 sens      binary     0.899     9 0.00579 Preprocessor1_Model1
## 8 spec      binary     0.556     9 0.0195  Preprocessor1_Model1
```

Show performance for every single fold:

```
rf_res %>%  collect_metrics(summarize = FALSE)
```

```
## # A tibble: 72 x 6
##    id      id2   .metric   .estimator .estimate .config
##    <chr>   <chr> <chr>     <chr>          <dbl> <chr>
##  1 Repeat1 Fold1 recall    binary         0.899 Preprocessor1_Model1
##  2 Repeat1 Fold1 precision binary         0.764 Preprocessor1_Model1
##   3 Repeat1 Fold1 f_meas    binary         0.826 Preprocessor1_Model1
##   4 Repeat1 Fold1 accuracy  binary         0.755 Preprocessor1_Model1
##   5 Repeat1 Fold1 kap       binary         0.423 Preprocessor1_Model1
##   6 Repeat1 Fold1 sens      binary         0.899 Preprocessor1_Model1
##   7 Repeat1 Fold1 spec      binary         0.494 Preprocessor1_Model1
##   8 Repeat1 Fold1 roc_auc   binary         0.783 Preprocessor1_Model1
##   9 Repeat1 Fold2 recall    binary         0.905 Preprocessor1_Model1
##  10 Repeat1 Fold2 precision binary         0.807 Preprocessor1_Model1
## # ... with 62 more rows
```

Collect model predictions

To obtain the actual model predictions, we use the function collect_predictions and save the result as log_pred:

```r
rf_res <-
  log_res %>%
  collect_predictions()
```

Confusion Matrix

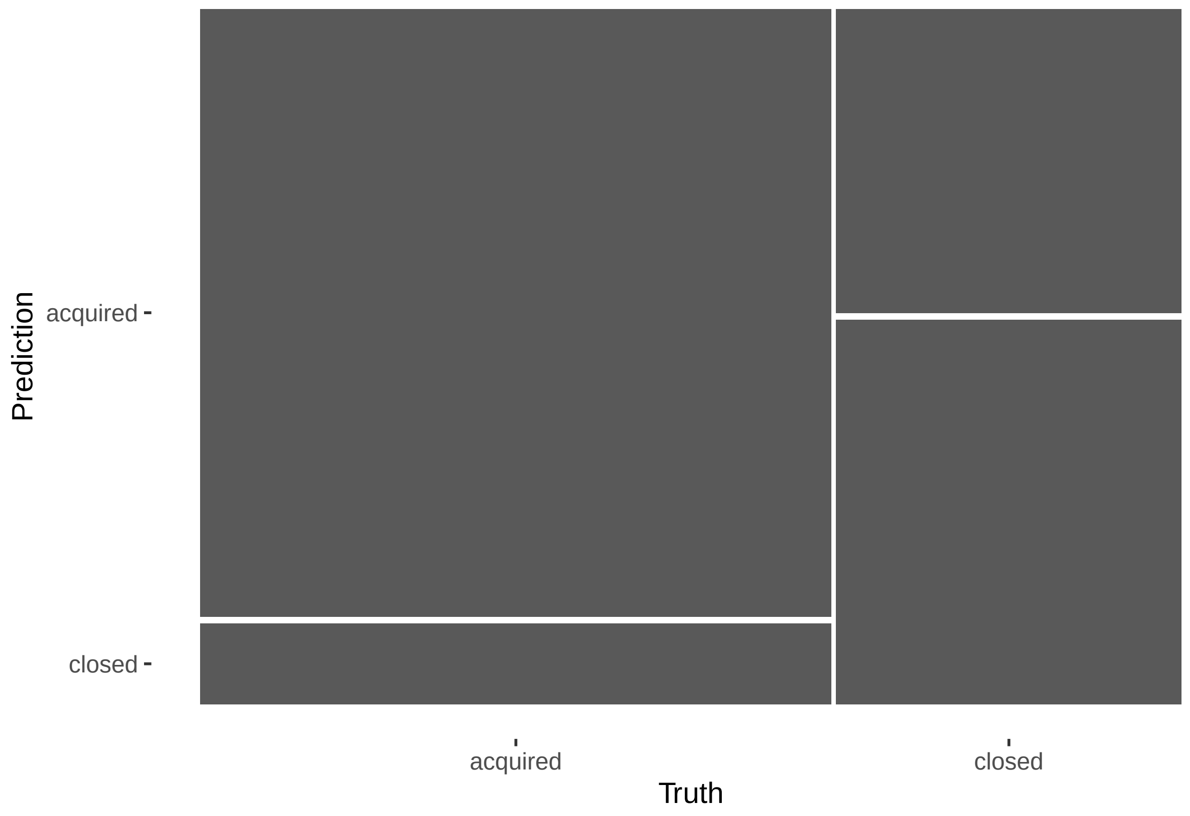
Create the confusion matrix for the predicted values.

```r
rf_res %>%
  conf_mat(y, .pred_class)
```

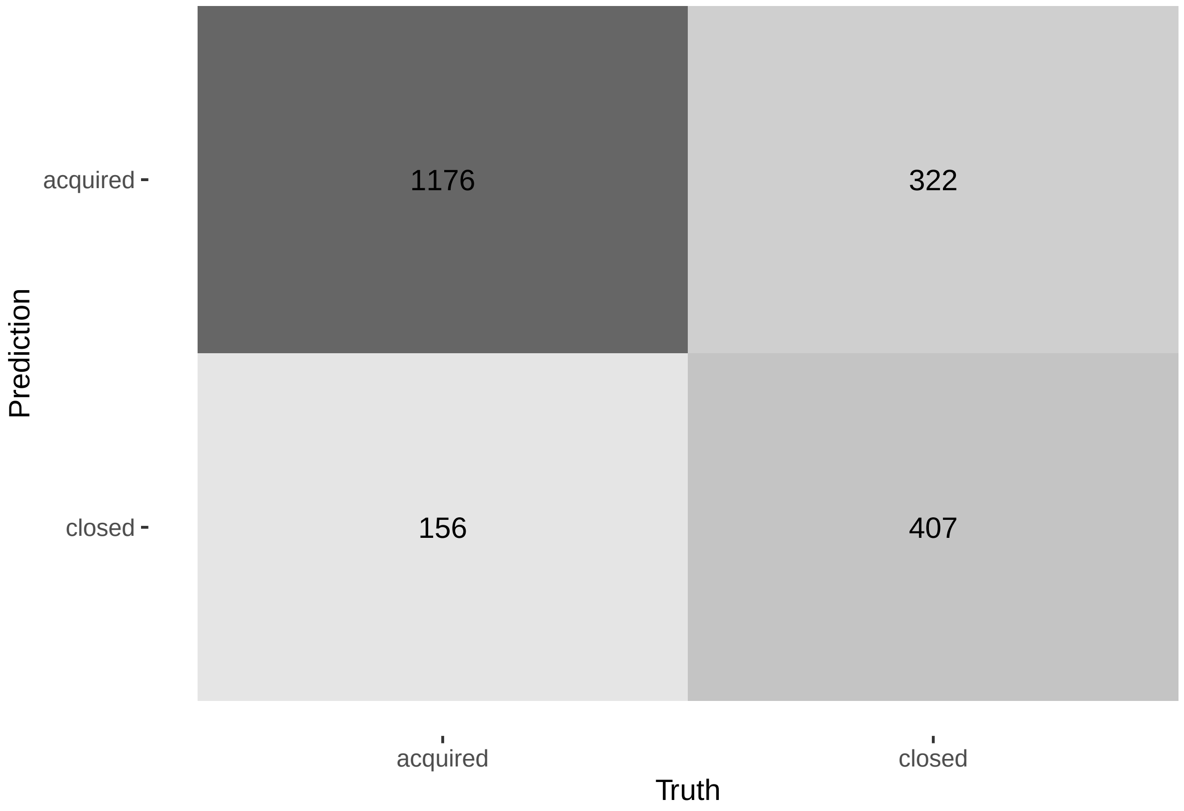
```
##           Truth
## Prediction acquired closed
##   acquired     1176    322
##   closed        156    407
```

Visualize again

```r
rf_res %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "mosaic")
```



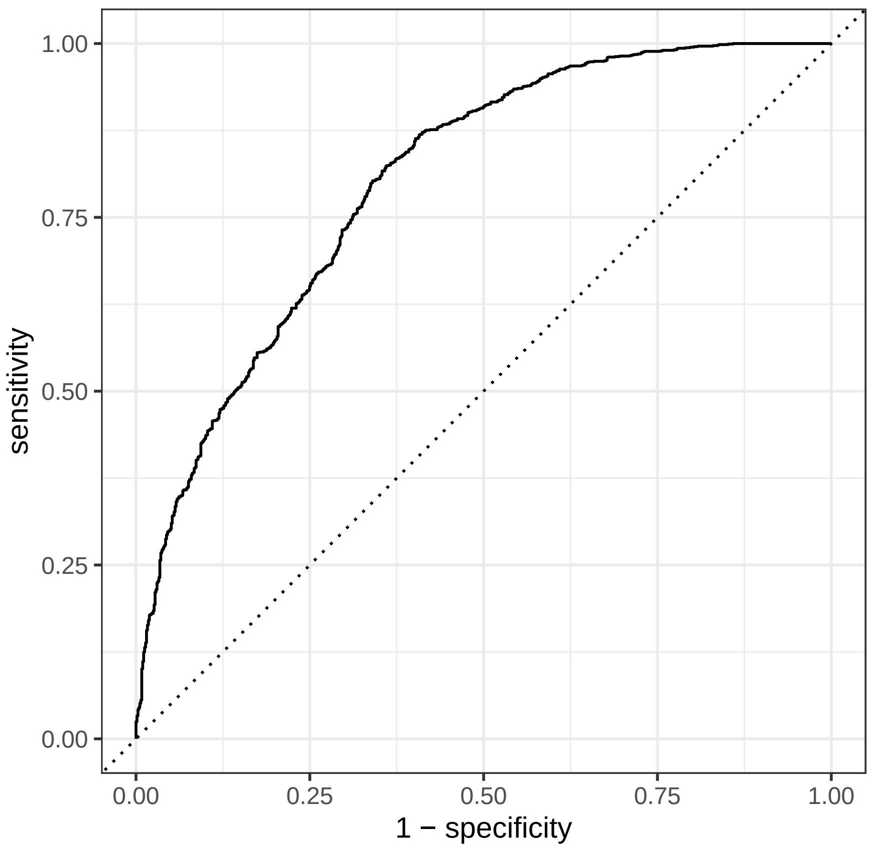
```r
rf_res %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "heatmap")
```



ROC curve

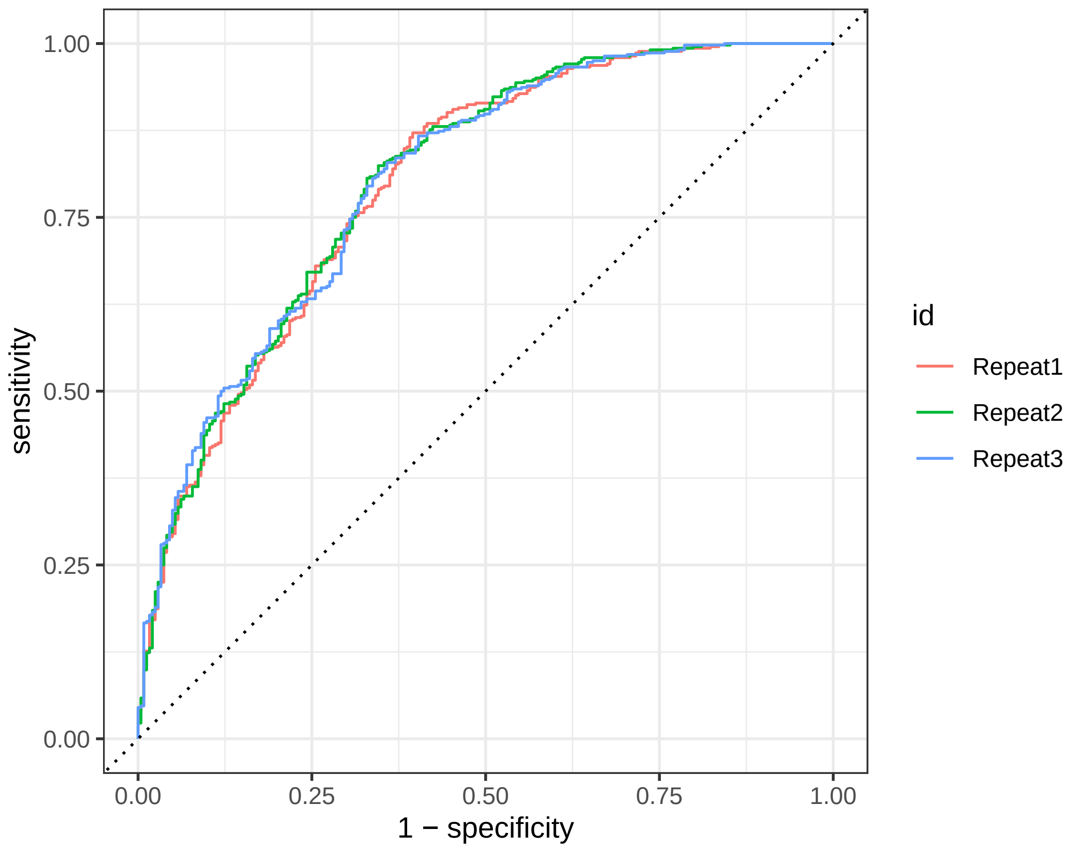
We will now create the ROC curve with 1 - specificity on the x-axis (false positive fraction = FP/(FP+TN)) and sensitivity on the y axis (true positive fraction = TP/(TP+FN)).

```
rf_res %>%
  roc_curve(y, .pred_acquired) %>%
  autoplot()
```



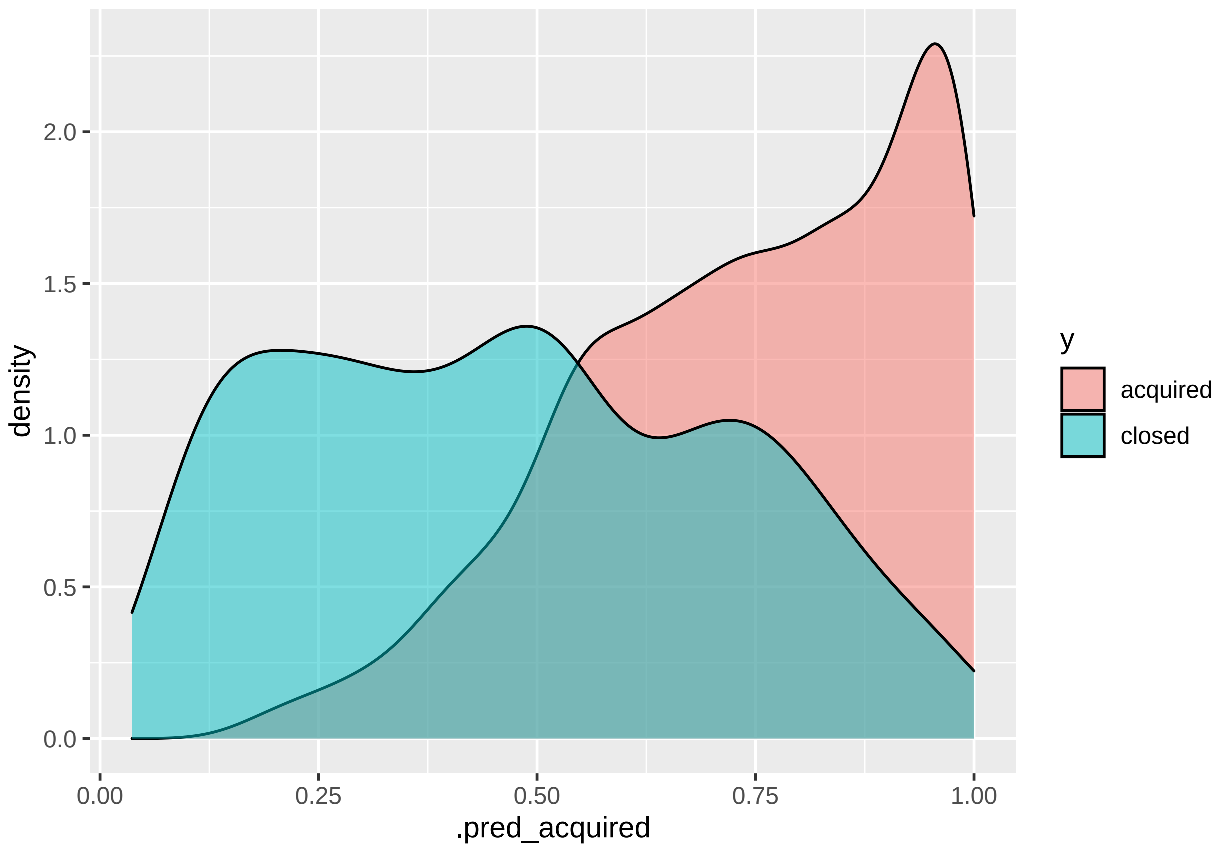
With folds.

```
rf_res %>%
  group_by(id) %>%
    roc_curve(y, .pred_acquired) %>%
  autoplot()
```



We create the predicted probability distributions for our two classes.

```
rf_res %>%
  ggplot() +
  geom_density(aes(x = .pred_acquired,
                   fill = y),
               alpha = 0.5)
```



Random Forest on test data

We now use the test data by setting the split argument equal to data_split.

```
last_fit_rf <- last_fit(workflow_rf,
                        split = data_split,
                        metrics = metric_set(
                          recall, precision, f_meas,
                          accuracy, kap,
                          roc_auc, sens, spec)
                        )
```

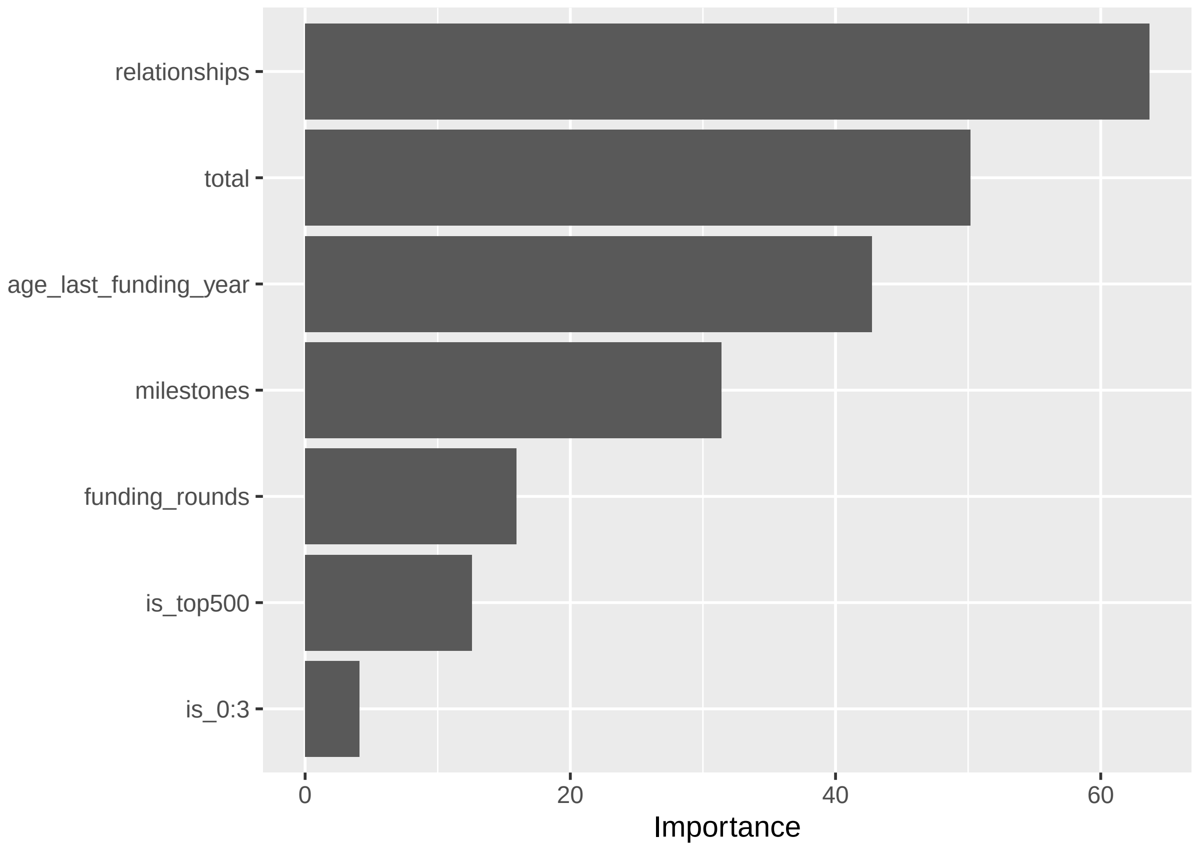
```
last_fit_rf %>%
  collect_metrics()
```

```
## # A tibble: 8 x 4
##   .metric   .estimator .estimate .config
##   <chr>     <chr>          <dbl> <chr>
## 1 recall    binary         0.886 Preprocessor1_Model1
## 2 precision binary         0.810 Preprocessor1_Model1
## 3 f_meas    binary         0.846 Preprocessor1_Model1
## 4 accuracy  binary         0.792 Preprocessor1_Model1
## 5 kap       binary         0.528 Preprocessor1_Model1
## 6 sens      binary         0.886 Preprocessor1_Model1
## 7 spec      binary         0.622 Preprocessor1_Model1
## 8 roc_auc   binary         0.826 Preprocessor1_Model1
```

We use the pluck function to calculate the variables with the biggest importance for the predicted variable.

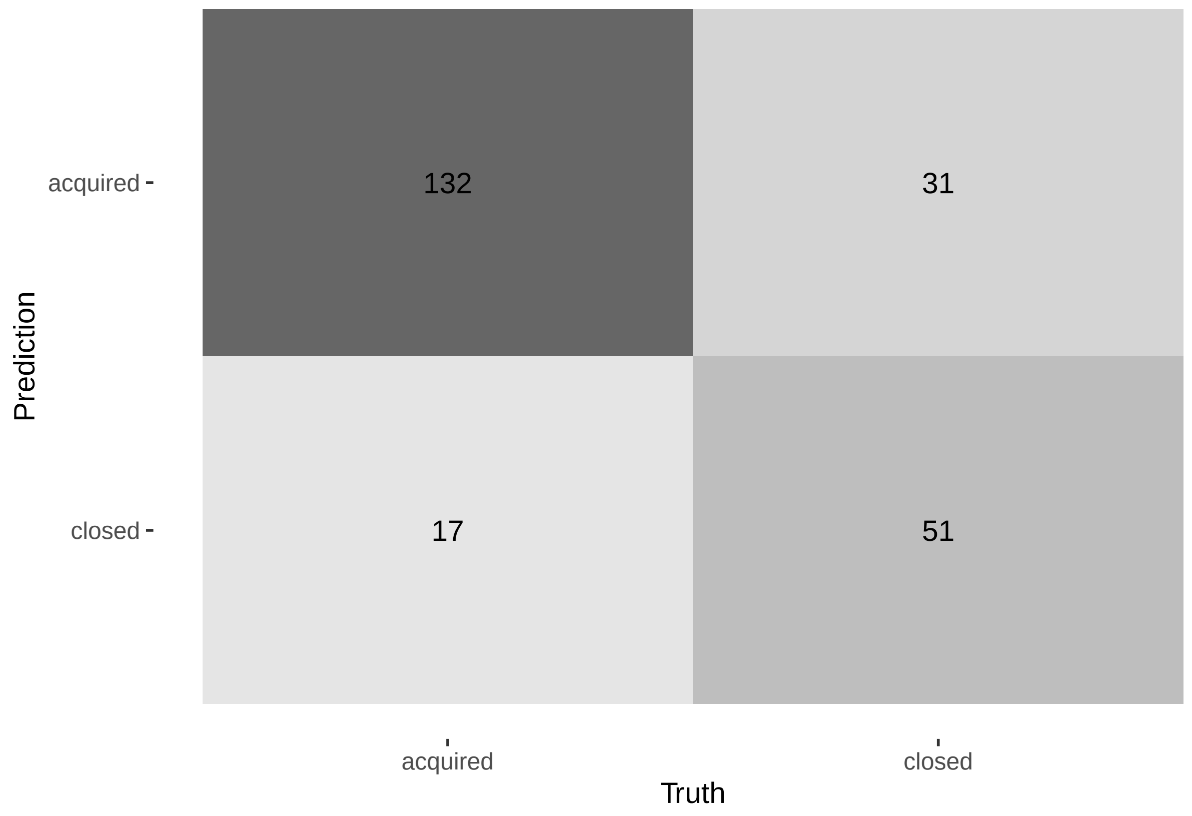
```r
last_fit_rf %>%
  pluck(".workflow", 1) %>%
  pull_workflow_fit() %>%
  vip(num_features = 7)
```

```
## Warning: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Please use `extract_fit_parsnip()` instead.
```



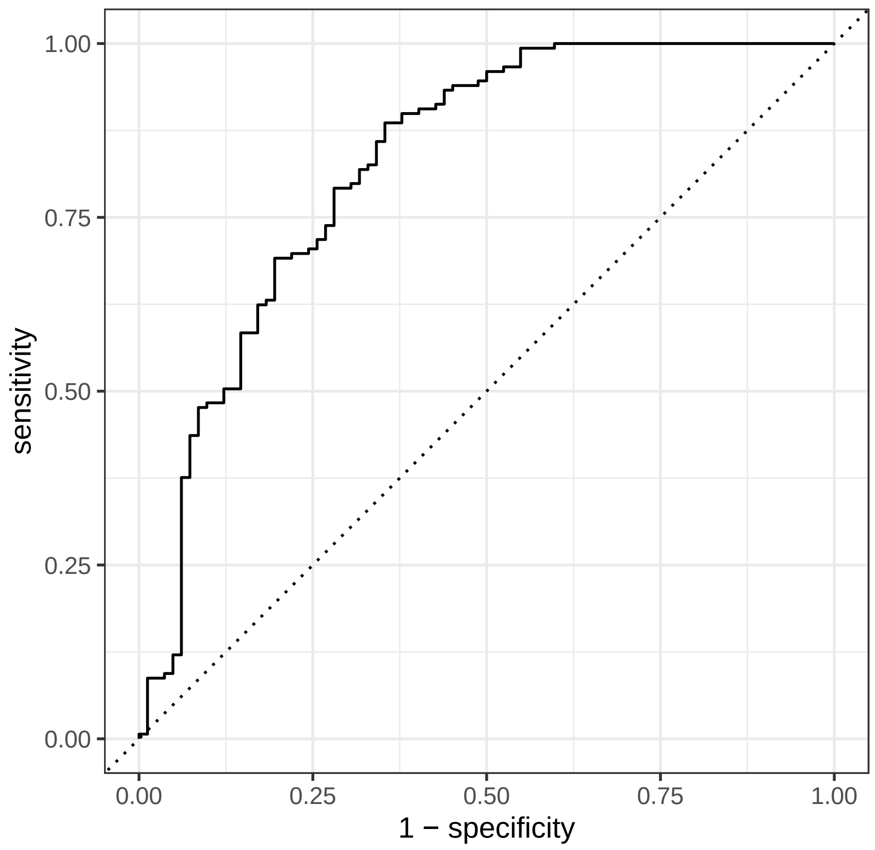
We can also make the confusion matrix using the test data

```r
last_fit_rf %>%
  collect_predictions() %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "heatmap")
```



And lastly show the ROC curve using the test data.

```
last_fit_rf %>%
  collect_predictions() %>%
  roc_curve(y, .pred_acquired) %>%
  autoplot()
```


sensitivity
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1 − specificity