Survival Lung Cancer Modeling: Xgboost, Random Forest, GBM tentatives

Karim Mezhoud

2020-01-23

Contents

NB.In pdf version, some plot are crowded. Please follow this link to html format.

1 Description

This report is subset into 3 parts:

- Exploratory of tumor arrays slides using python code
- Exploratory radiomics, and clinical data (train) using R code
- Prediction Event by: xgboost, RF, GBM

1.1 Clinical Context

Computed Tomography scanner (CT scan) is a widely spread and popular exam in oncology: it reflects the density of the tissues of the human body. It is, then, adapted to the study of lung cancer because lungs are mostly filled with air (low density) while tumors are made of dense tissues.

1.2 Clinical context

Small Cell Lung Cancer can itself be split into four major subtypes based on histology observations: squamous cell carcinoma, large cell carcinoma, adenocarcinoma and a mixture of all

1.3 Goal

Predict the survival time of a patient (remaining days to live) from one three-dimensional CT scan (grayscale image) and a set of pre-extracted quantitative imaging features, as well as clinical data.

1.4 dataset

To each patient corresponds one CT scan, and one binary segmentation mask. The segmentation mask is a binary volume of the same size as the CT scan, except that it is composed of zeroes everywhere there is no tumour, and 1 otherwise. The CT scans and the associated segmentation masks are subsets of two public datasets:

- NSCLC Radiomics (subset of 285 patients)
- NSCLC RadioGenomics(subset of 141 patients)

Both training and validation contain for each patient, the time to event (days), as well as the censorship. Censorship indicates whether the event (death) was observed or whether the patient escaped the study: this can happen when the patient's track was lost, or if the patient died of causes not related to the disease.

2 Tumor Arrays Slides Exploration

2.1 Setting python version and anaconda environment for R :-)

```
reticulate::use_python("/Users/Mezhoud/anaconda3/bin/python3", required = TRUE)
reticulate::py_config()
```

```
## python:         /Users/Mezhoud/anaconda3/bin/python3
## libpython:      /Users/Mezhoud/anaconda3/lib/libpython3.7m.dylib
## pythonhome:     /Users/Mezhoud/anaconda3:/Users/Mezhoud/anaconda3
## version:        3.7.5 (default, Oct 25 2019, 10:52:18)  [Clang 4.0.1 (tags/RELEASE_401/final)]
## numpy:          /Users/Mezhoud/anaconda3/lib/python3.7/site-packages/numpy
## numpy_version:  1.17.3
##
## NOTE: Python version was forced by use_python function
```

```
knitr::opts_chunk$set(engine.path = list(
  python = '/Users/Mezhoud/anaconda3/bin/python3'
))
```

2.2 Load scans and masks of Tumor lung cancer

```
import numpy as np
from matplotlib import pyplot as plt
#from matplotlib import pyplot
from PIL import Image

img_array = np.load('train/images/patient_002.npz')
scan = img_array['scan']
mask = img_array['mask']

print("the dimension of scan array is: ", str(scan.shape))
```

```
## the dimension of scan array is:  (92, 92, 92)
```

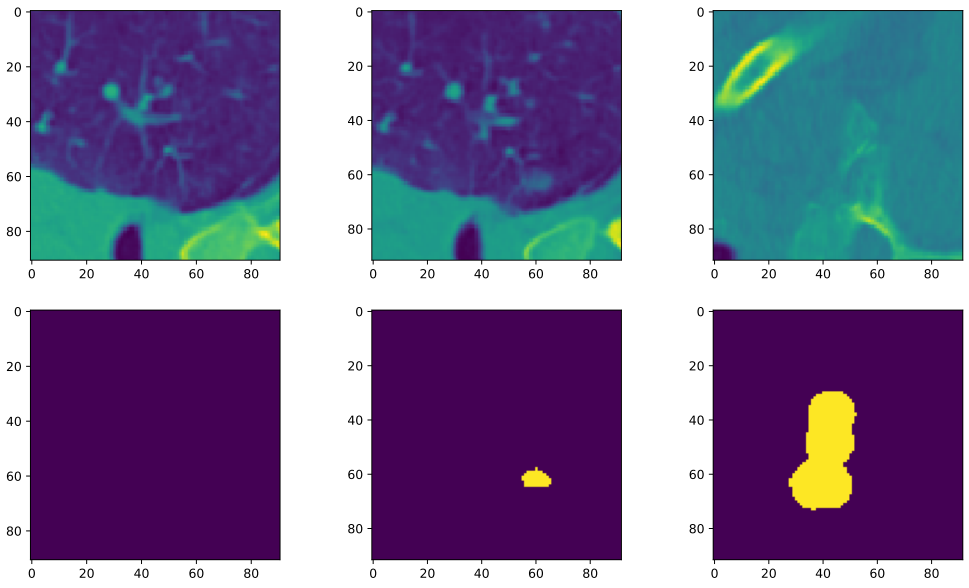
```
print("the dimension of mask array is: ", str(mask.shape))
```

```
## the dimension of mask array is:  (92, 92, 92)
```

```
print("plot some images from patient 002: ")
#plt.imshow(scan[:, :, 3])
```

```
## plot some images from patient 002:
```

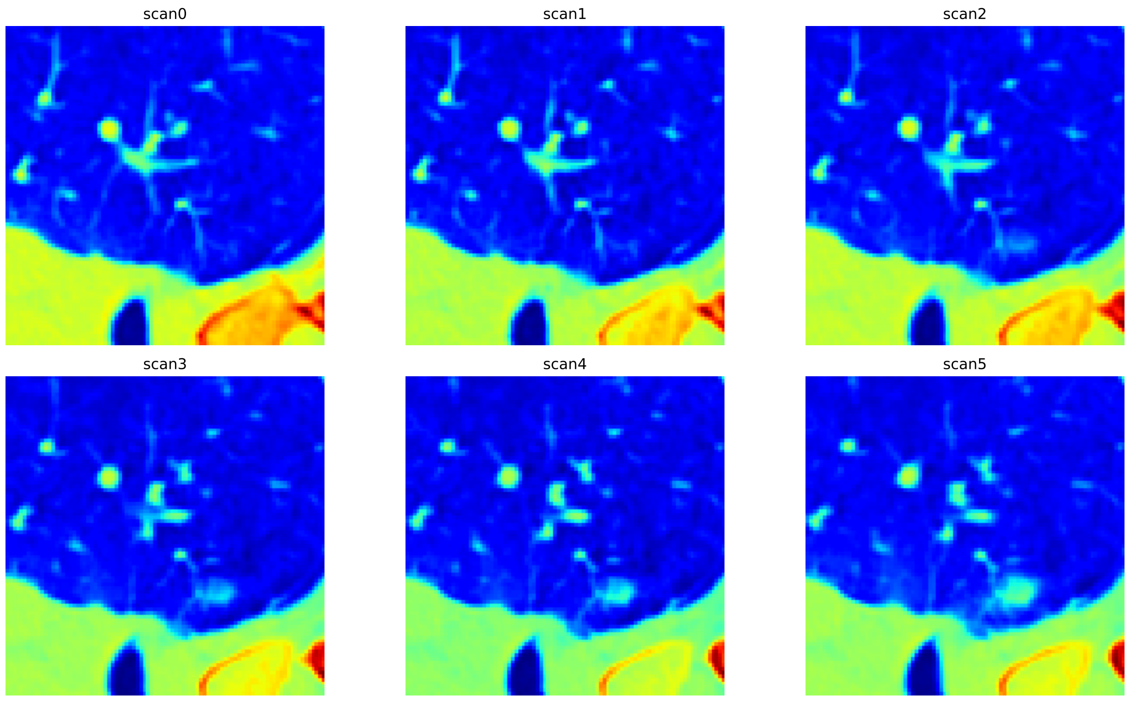
```
f, axarr = plt.subplots(2,3)
axarr[0,0].imshow(scan[1:92, 1:92, 0])
axarr[1,0].imshow(mask[1:92, 1:92, 0])
axarr[0,1].imshow(scan[:, :, 3])
axarr[1,1].imshow(mask[:, :, 3])
axarr[0,2].imshow(scan[:, :, 80])
axarr[1,2].imshow(mask[:, :, 80])
```



2.2.1 Function to plot multiple image from array

```
def plot_figures(figures, nrows = 1, ncols=1):
  """Plot a dictionary of figures.

  Parameters
  ----------
  figures : <title, figure> dictionary
  ncols : number of columns of subplots wanted in the display
  nrows : number of rows of subplots wanted in the figure
  """
  fig, axeslist = plt.subplots(ncols=ncols, nrows=nrows)
  for ind,title in zip(range(len(figures)), figures):
      axeslist.ravel()[ind].imshow(figures[title], cmap=plt.jet())
      axeslist.ravel()[ind].set_title(title)
      axeslist.ravel()[ind].set_axis_off()
  plt.tight_layout()


img_array = np.load('train/images/patient_002.npz')
scan = img_array['scan']
mask = img_array['mask']


# generation of a dictionary of (title, images)
number_of_im = 6
scan = {'scan'+str(i): scan[1:92, 1:92, i] for i in range(number_of_im)}

# plot of the images in a figure, with 5 rows and 4 columns
```

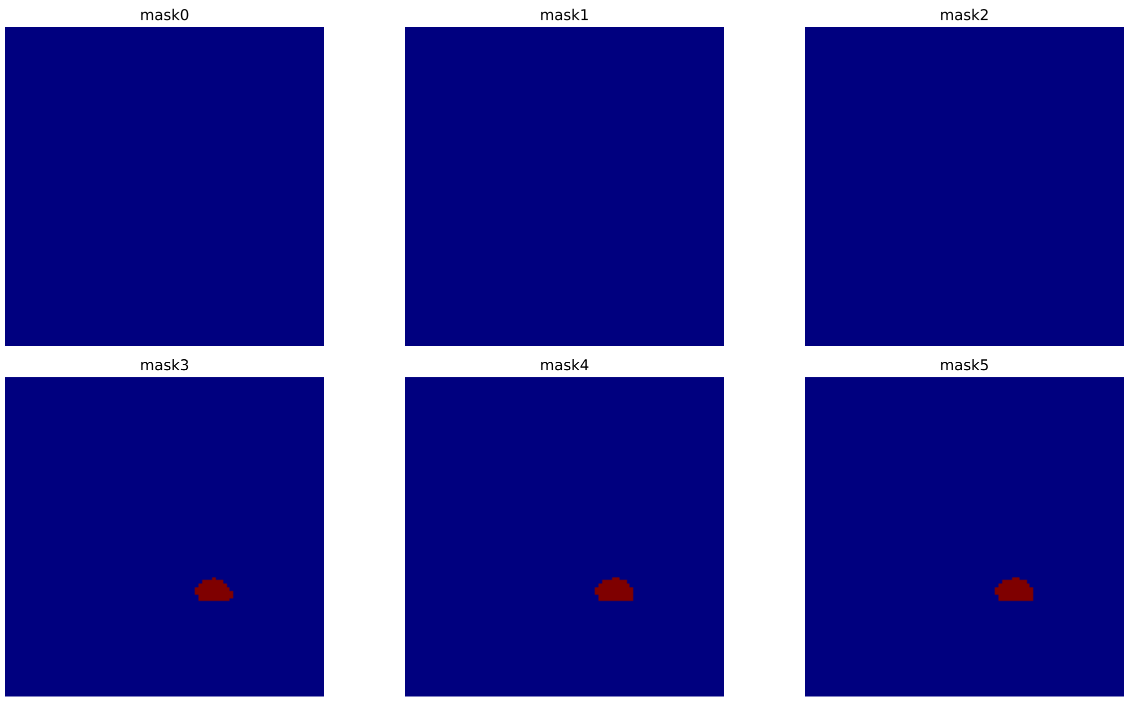
```
plot_figures(scan, 2, 3)
plt.show()
```



The plot shows colored images scan of 6 slides. At this step it is not easy to distinguish the tumor.

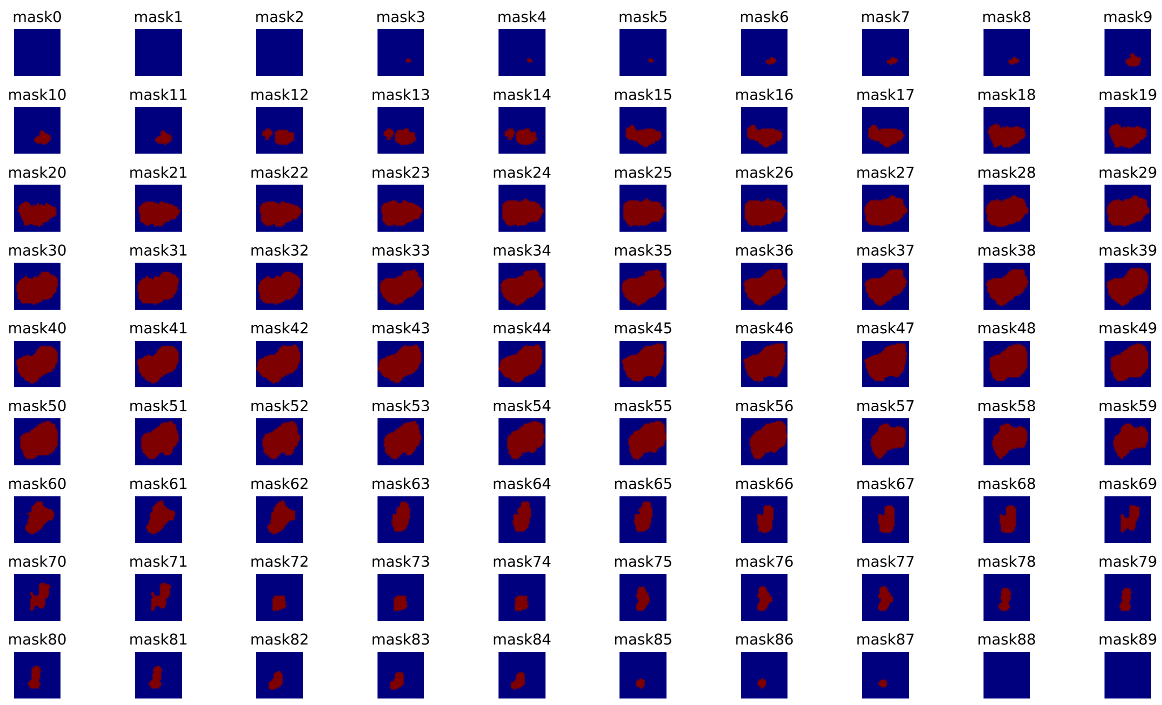
The dataset has aslo the masks for each scan slide which locate the position of the tumor in the scan.

```
mask = {'mask'+str(i): mask[1:92, 1:92, i] for i in range(number_of_im)}
# plot of the images in a figure, with 5 rows and 4 columns
plot_figures(mask, 2, 3)
plt.show()
```



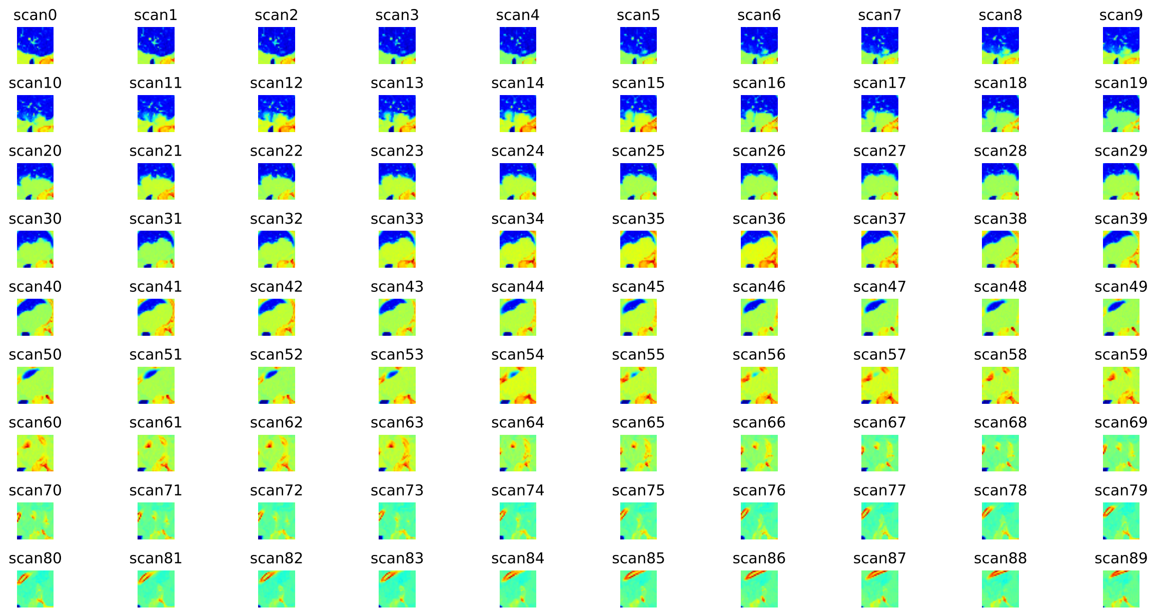
- The first 3 slides do not have tumor streak, however the next 3 ones indicate the position of the tumor in red color.
- If we plot more slides, we can observe the increase of the size of the tumor during plotting slides.
- At the end the size the Tumor is decreasing.
- We can note that the crop is adjusted to the size of the tumor

```
img_array = np.load('train/images/patient_002.npz')
scan = img_array['scan']
mask = img_array['mask']

mask = {'mask'+str(i): mask[1:92, 1:92, i] for i in range(90)}
# plot of the images in a figure, with 5 rows and 4 columns
plot_figures(mask, 9, 10)
plt.show()
```



If we compare with the scan slides, we obtain:

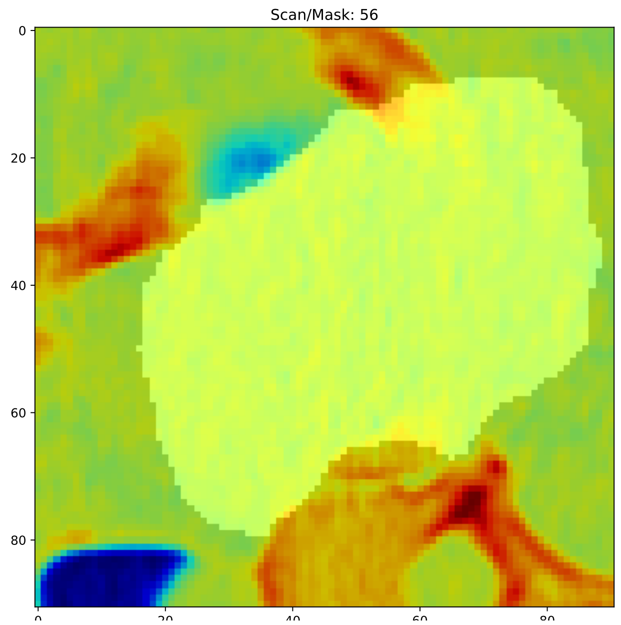
```
scan = {'scan'+str(i): scan[1:92, 1:92, i] for i in range(90)}
# plot of the images in a figure, with 5 rows and 4 columns
plot_figures(scan, 9, 10)
plt.show()
```



- It is always not easy to delimit the tumor in scan images
- Comparing to masks, we can note that, between scan 34 and scan 65, the slides have more yellow stain or less blue color.

2.2.2 Superimposing Scan and mask images

```
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image

img_array = np.load('train/images/patient_002.npz')
scan = img_array['scan']
mask = img_array['mask']


background = mask[1:92, 1:92, 56]
overlay = scan[1:92, 1:92, 56]

plt.title("Scan/Mask: 56")
plt.imshow(background, cmap='gray')
plt.imshow(overlay, cmap='jet', alpha=0.9)
```

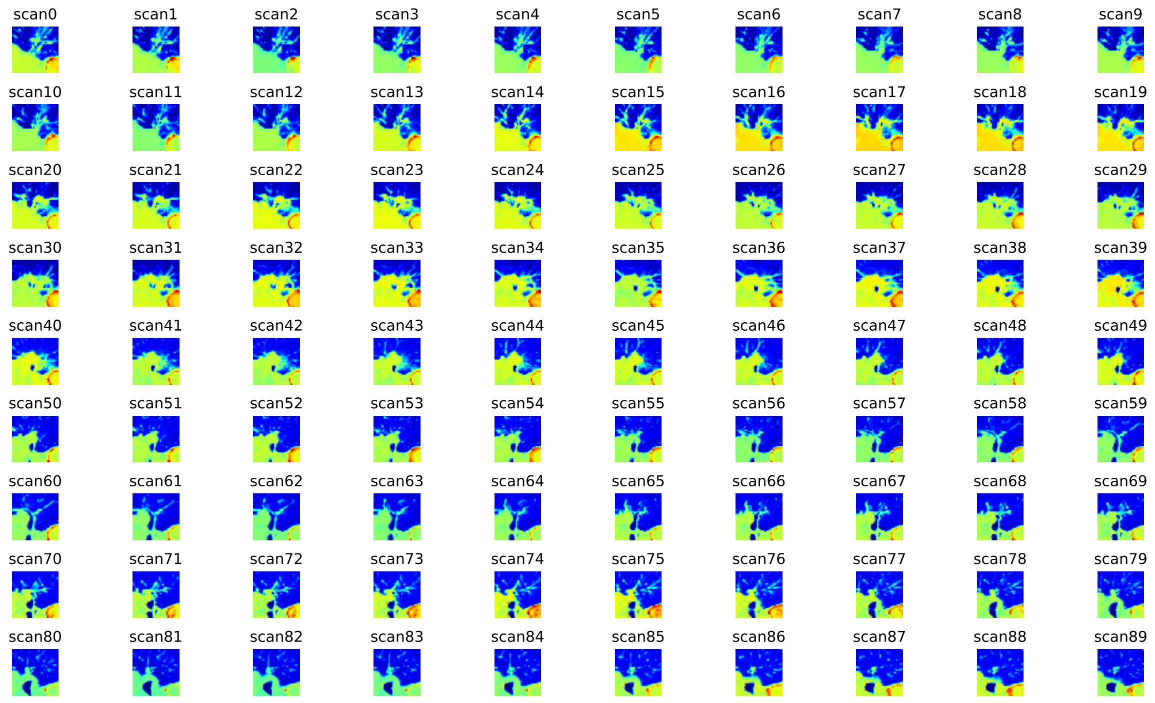


- It isnow clear that masks seems to be more useful that scans because the tumor in not visible in scan slides.

2.3 Load images from test dataset

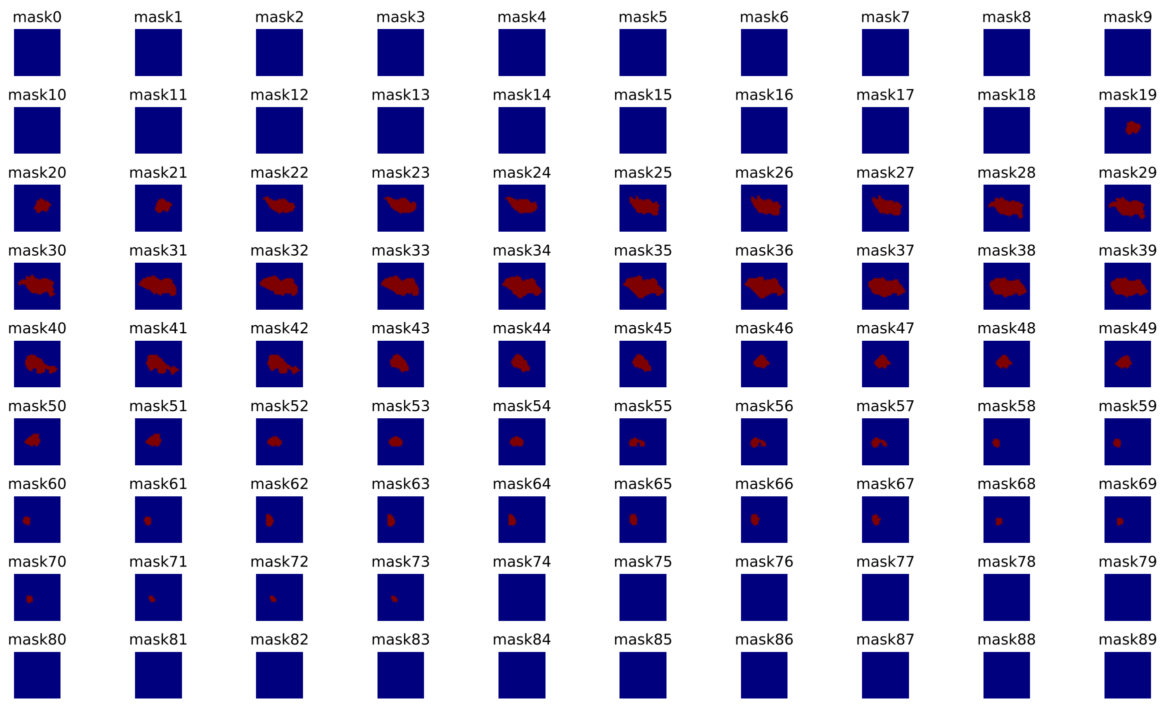
```
img_array = np.load('test/images/patient_001.npz')
scan = img_array['scan']
mask = img_array['mask']


# generation of a dictionary of (title, images)
number_of_im = 90
scan = {'scan'+str(i): scan[1:92, 1:92, i] for i in range(number_of_im)}
```

```
# plot of the images in a figure, with 5 rows and 4 columns
plot_figures(scan, 9, 10)
plt.show()
```

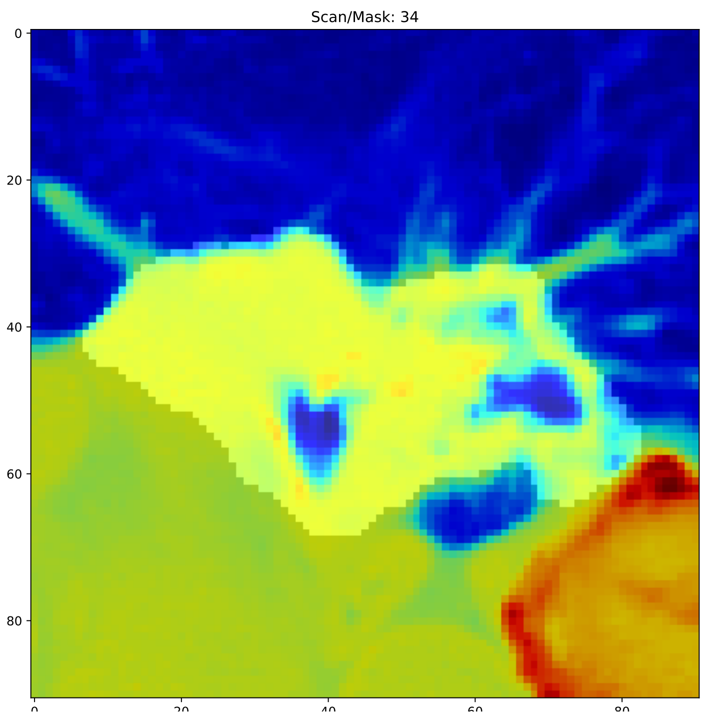


- Plot mask slides from test dataset

```
mask = {'mask'+str(i): mask[1:92, 1:92, i] for i in range(number_of_im)}
# plot of the images in a figure, with 5 rows and 4 columns
plot_figures(mask, 9, 10)
plt.show()
```



```python
img_array = np.load('test/images/patient_001.npz')
scan = img_array['scan']
mask = img_array['mask']

background = mask[1:92, 1:92, 34]
overlay = scan[1:92, 1:92, 34]

plt.title("Scan/Mask: 34")
plt.imshow(background, cmap='gray')
plt.imshow(overlay, cmap='jet', alpha=0.9)
```



- In the test images, we can also observe tumor slides like in train dataset.
- For training step, it maybe better to use masks slides than scan. But we need to explore variables in clinical data and radiomics and think how to associate images with numeric variables.
- One think we can do is the convert slides to dataframe (each slide in one row) and then we can obtain one matrix for each patient tumor.
- At this step I will switch from python to R :-)

2.4 Import image from python environment to R

The goal of this step is to convert image matrices as vector. So, each image can be ranged in one row. Finally, we can obtain one dataframe with 92 rows (images) ofr each sample (patient).

2.4.1 Import useful R packages

2.4.1.1 Useful python function

```
import numpy as np

def load_img_array(file):
  im_array = np.load(file)
  scan = im_array['scan']
  mask = im_array['mask']
  return scan,mask
```

2.4.1.2 Understanding the structure of the array of images

```r
patient_002 <- reticulate::py$load_img_array('train/images/patient_002.npz')

paste0("One image is a: ", class(patient_002[[1]][,,1]))
```

```
## [1] "One image is a: matrix"
```

```r
paste0("Two images are an: ", class(patient_002[[1]][,,1:2]))
```

```
## [1] "Two images are an: array"
```

```r
paste0("Print the first 10 pixels of Scan N°1: "); patient_002[[1]][,,1][1:10, 1:10]
```

```
## [1] "Print the first 10 pixels of Scan N°1: "

##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,] -777 -759 -707 -697 -749 -796 -826 -837 -858  -860
##  [2,] -783 -791 -774 -768 -787 -808 -827 -826 -829  -829
##  [3,] -804 -841 -827 -812 -820 -840 -831 -801 -792  -794
##  [4,] -830 -857 -839 -816 -805 -818 -801 -764 -734  -722
##  [5,] -844 -854 -843 -823 -799 -787 -771 -743 -704  -670
##  [6,] -844 -849 -844 -831 -821 -810 -809 -791 -760  -719
##  [7,] -848 -847 -848 -844 -847 -841 -849 -841 -829  -806
##  [8,] -847 -856 -854 -846 -840 -813 -826 -849 -848  -836
##  [9,] -840 -851 -836 -823 -835 -796 -803 -853 -857  -833
## [10,] -847 -841 -829 -817 -860 -832 -799 -842 -865  -838
```

```r
paste0("Print the first 10 pixels of Mask N°1: "); patient_002[[2]][,,1][1:10, 1:10]
```

```
## [1] "Print the first 10 pixels of Mask N°1: "

##        [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
##  [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [2,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [3,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [4,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [5,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [6,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [7,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [8,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [9,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [10,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

2.4.1.3 Convert the array of matrices to a list of matrices

```r
ls_scan_patient_002 <- lapply(seq(dim(patient_002[[1]])[3]), function(x) patient_002[[1]][ , , x])
ls_mask_patient_002 <- lapply(seq(dim(patient_002[[2]])[3]), function(x) patient_002[[1]][ , , x])

paste0("The dimension of the scan images is: ", length(ls_scan_patient_002))
```

```
## [1] "The dimension of the scan images is: 92"
```

```r
paste0("The dimension of the mask images is: ", length(ls_mask_patient_002))
```

```
## [1] "The dimension of the mask images is: 92"
```

2.4.1.4 Convert image matrix to vector

```r
mat2vec <- function(path){

  # Load patient CT scan
  patient <- py$load_img_array(path)

  # list scans
  scan <- lapply(seq(dim(patient[[1]])[3]), function(x) patient[[1]][ , , x])
  # list masks
  mask <- lapply(seq(dim(patient[[2]])[3]), function(x) patient[[1]][ , , x])

  # vectorize each matrix (image) into vector
  vec_scan <- lapply(scan, function(x) as.vector(x))

  # vectorise each mask (image) to vector
  vec_mask <- lapply(mask, function(x) as.vector(x))


  # bind vector into dataframe by row
  df_scan <-as.data.frame( do.call(rbind, vec_scan))
  df_mask <-as.data.frame( do.call(rbind, vec_mask))

  # extract patien_id from path
  scan_id <- paste0(tools::file_path_sans_ext(basename(path)), "_scan")
  mask_id <- paste0(tools::file_path_sans_ext(basename(path)), "_mask")

  # group in list the scan and the mask dataframes
  ls <- list(df_scan, df_mask)

  # Rename list
  names(ls) <- c(scan_id, mask_id)


  return(ls)
}

patient2 <- mat2vec('train/images/patient_002.npz')

paste0("The output is a: " ,class(patient2))
```

```
## [1] "The output is a: list"
```

```r
paste0("With length of: ", length(patient2))
```

```
## [1] "With length of: 2"
```

```r
paste0("The names of two elements are: ") ; names(patient2)
```

```
## [1] "The names of two elements are: "

## [1] "patient_002_scan" "patient_002_mask"
```

```r
paste0("which are: ", class(patient2$patient_002_scan))
```

```
## [1] "which are: data.frame"
```

```r
paste0("The dimension of each dataframe is: ") ; dim(patient2$patient_002_scan)
```

```
## [1] "The dimension of each dataframe is: "

## [1]   92 8464
```

- At this step we stop exploring scan and mask.
- We think to use only masks for modeling
- Potential method: keras, mxnet

3 Exploratory Data Analysis of radiomics and clinical data

```
#Load dataset
radiomics <- fread("train/features/radiomics.csv", quote = "")
clinical <- fread("train/features/clinical_data.csv")

# display only 8 columns and 5 rows
head(radiomics)[,1:8]
```

```
##          V1                           V2                           V3
## 1:                                 shape                        shape
## 2:           original_shape_Compactness1 original_shape_Compactness2
## 3: PatientID
## 4:       202                 0.027815034                  0.274891585
## 5:       371                  0.02301549                  0.188210005
## 6:       246                 0.027348106                  0.265739895
##                                 V4                                   V5
## 1:                           shape                                shape
## 2: original_shape_Maximum3DDiameter original_shape_SphericalDisproportion
## 3:
## 4:                     48.55924217                          1.537964054
## 5:                     75.70336849                          1.744961158
## 6:                     70.43436661                          1.555420243
##                           V6                          V7
## 1:                     shape                       shape
## 2: original_shape_Sphericity original_shape_SurfaceArea
## 3:
## 4:               0.650210255                  5431.33321
## 5:               0.573078659                 10369.56873
## 6:               0.642913068                 10558.81869
##                                  V8
## 1:                            shape
## 2: original_shape_SurfaceVolumeRatio
## 3:
## 4:                      0.275227763
## 5:                      0.240726824
## 6:                      0.200765988
```

```
head(clinical)
```

```
##    PatientID               Histology Mstage Nstage SourceDataset Tstage     age
## 1:       202          Adenocarcinoma      0      0            l2      2 66.0000
## 2:       371              large cell      0      2            l1      4 64.5722
## 3:       246 squamous cell carcinoma      0      3            l1      2 66.0452
## 4:       240                     nos      0      2            l1      3 59.3566
## 5:       284 squamous cell carcinoma      0      3             11       4 71.0554
## 6:       348 squamous cell carcinoma      0      2             11       2 65.0212
```

The radiomics features can be divided into 4 groups as follows (shown in row 1): - Group 1. First order statistics - Group 2. Shape and size based features - Group 3. Textural features - Group 4. Wavelet features

Each group can be subset into several sub-groups shown in row 2 of the radiomics dataset. To make the radiomics features numeric dataset we need to remove the two first rows and convert them to colnames.

```
groups <- radiomics[1:2,-1] %>%
  t() %>%
  as.data.frame() %>%
  rename("Groups" = V1, "Features" = V2) #%>%
#  remove_rownames()

head(groups)
```

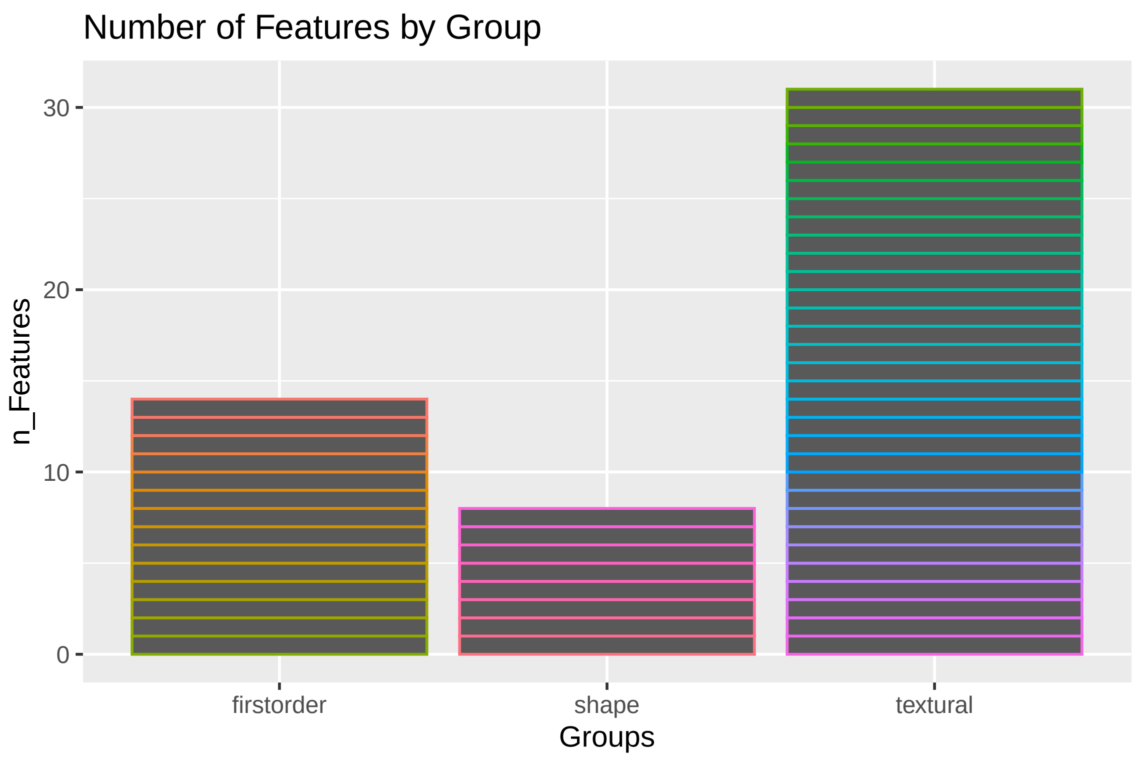
```
##    Groups                              Features
## V2  shape            original_shape_Compactness1
## V3  shape            original_shape_Compactness2
## V4  shape       original_shape_Maximum3DDiameter
## V5  shape original_shape_SphericalDisproportion
## V6  shape              original_shape_Sphericity
## V7  shape             original_shape_SurfaceArea
```

To improve the esthetic of the dataframe, we note:

- `original` is repetitive word. we can omit it.
- The group label is included in Feature label except `textural`
- We can remove `original` from Features and use use the rest as colnames of the radiomics dataset.

3.1 Plot the distribution of Goups and features

```
groups %>%
  group_by(Groups, Features) %>%
  summarise(n_Features = n()) %>%
  ggplot() +
  aes(x = Groups, y = n_Features, color = Features ) +
  geom_col() +
  theme(legend.position = "none") +
  ggtitle("Number of Features by Group")
```



3.1.1 Set New Colnames of radiomics

```
new_colnames_radiomics <- groups %>%
  mutate(Features = stringr::str_remove(Features,"original_")) %>%
  pull(Features)

new_colnames_radiomics %>% head()
```

```
## [1] "shape_Compactness1"           "shape_Compactness2"
## [3] "shape_Maximum3DDiameter"      "shape_SphericalDisproportion"
## [5] "shape_Sphericity"             "shape_SurfaceArea"
```

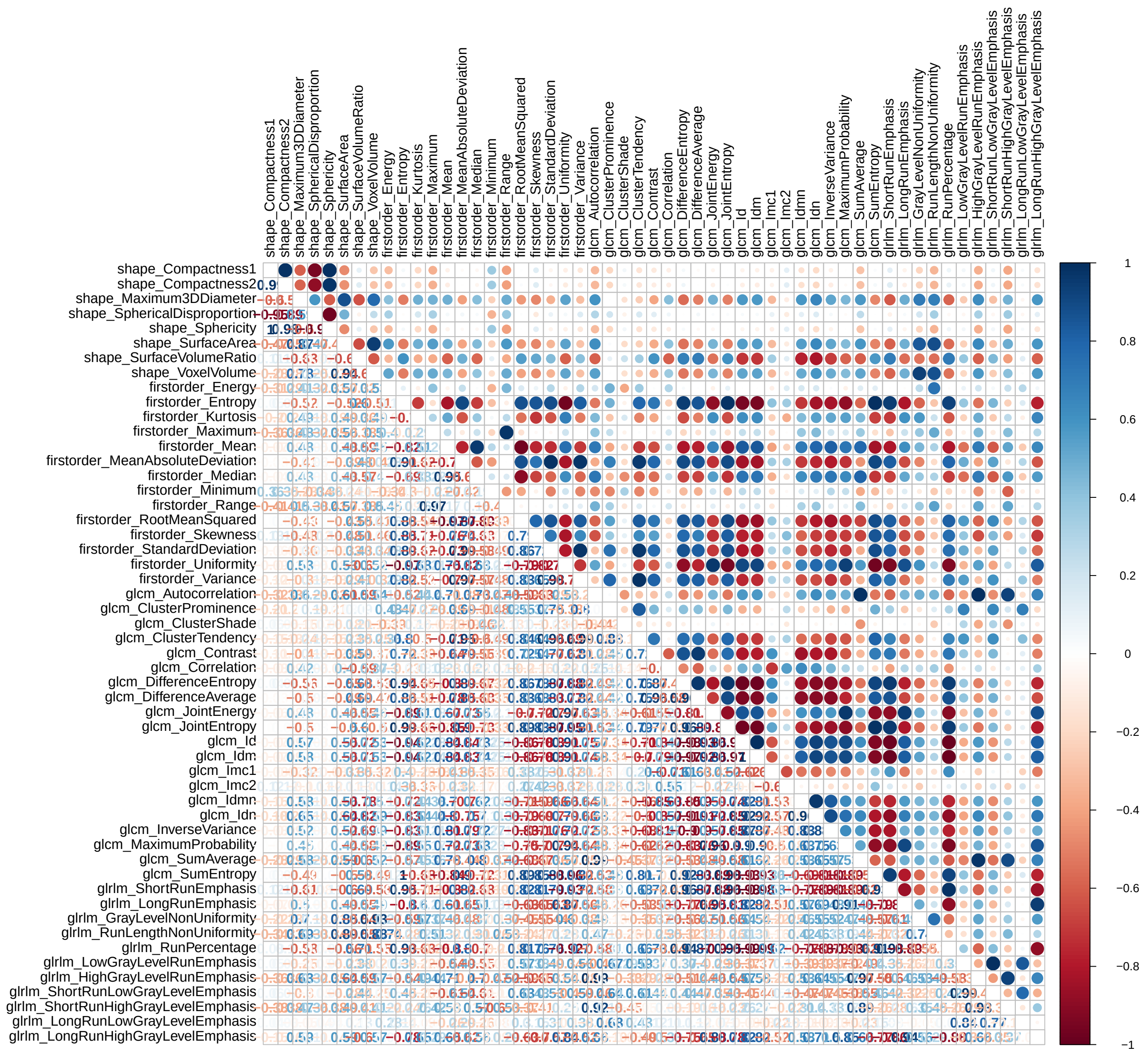
3.1.2 Get new radiomics style

```
old_names <- colnames(radiomics)
new_names <- c("PatientID", new_colnames_radiomics)

new_radiomics <- radiomics[-1:-3,] %>%
  rename_at(vars(old_names), ~ new_names) %>%
  mutate_if(is.character, as.numeric) #%>%
#as.matrix()


head(new_radiomics)[,1:8]
```

```
##   PatientID shape_Compactness1 shape_Compactness2 shape_Maximum3DDiameter
## 1       202         0.02781503          0.2748916                48.55924
## 2       371         0.02301549          0.1882100                75.70337
## 3       246         0.02734811          0.2657399                 70.43437
## 4       240         0.02681111          0.2554064                 46.81880
## 5       284         0.02369124          0.1994242                 53.79591
## 6       348         0.03098136          0.3410383                 63.74951
##   shape_SphericalDisproportion shape_Sphericity shape_SurfaceArea
## 1                     1.537964        0.6502103          5431.333
## 2                     1.744961        0.5730787         10369.569
## 3                     1.555420        0.6429131         10558.819
## 4                     1.576120        0.6344693          4221.412
## 5                     1.711620        0.5842418          5295.900
## 6                     1.431305        0.6986630          8493.134
##   shape_SurfaceVolumeRatio
## 1                0.2752278
## 2                0.2407268
## 3                0.2007660
## 4                0.3238780
## 5                0.3272407
## 6                0.1976017
```

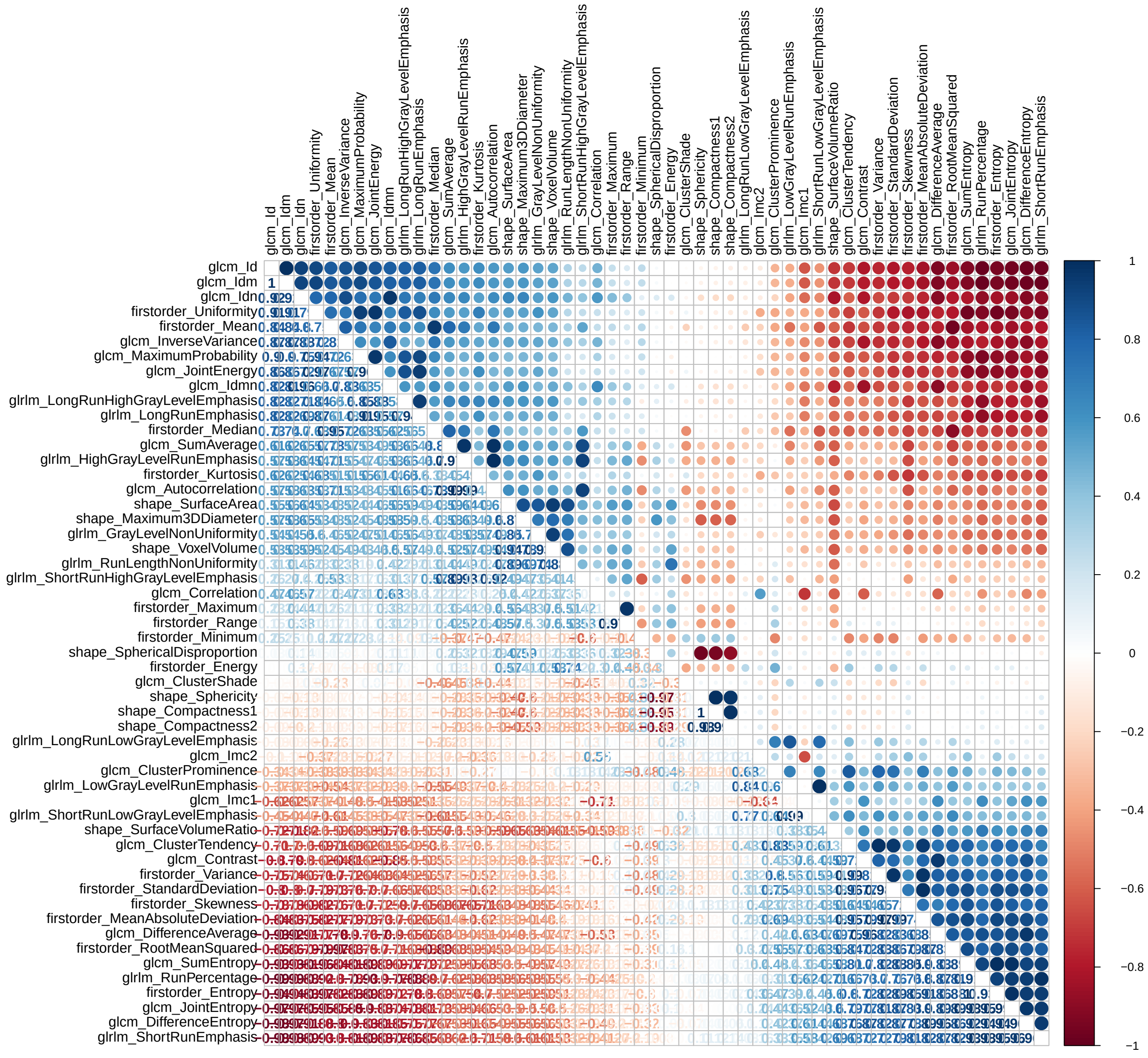
3.2 Glimpse correlation between features (default order)

```
M <- cor(new_radiomics[-1])
#corrplot(M,  method = "circle")
corrplot.mixed(M, tl.col="black", tl.pos = "lt")
```



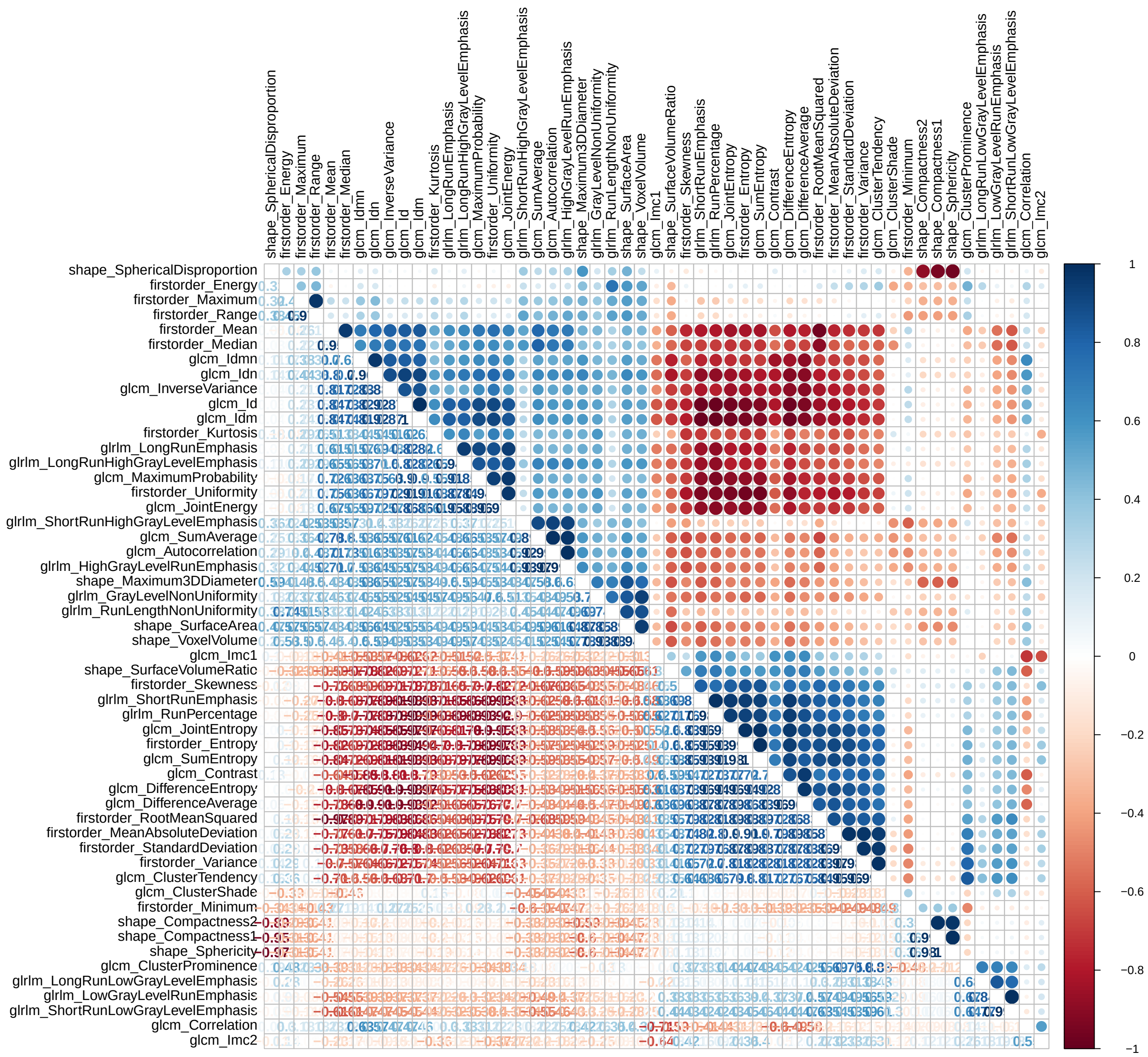
3.2.1 Set the first principal component order of the features

```
corrplot.mixed(M, tl.col="black", tl.pos = "lt", order = "FPC")
```



3.2.2 Set the hierarchical clustering order of the features

```
corrplot.mixed(M, tl.col="black", tl.pos = "lt", order = "hclust")
```

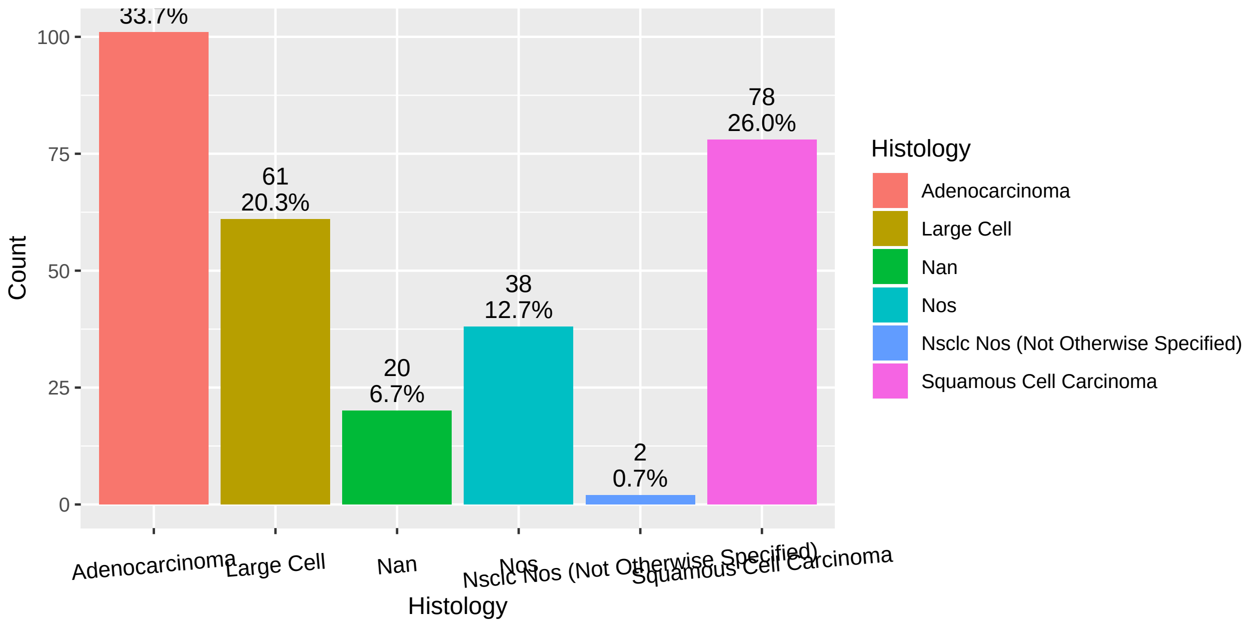


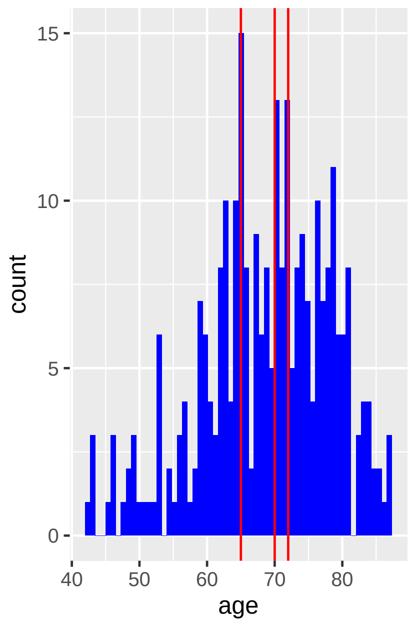
- We can return to these heatmap when we predict the most importante features using modeling.

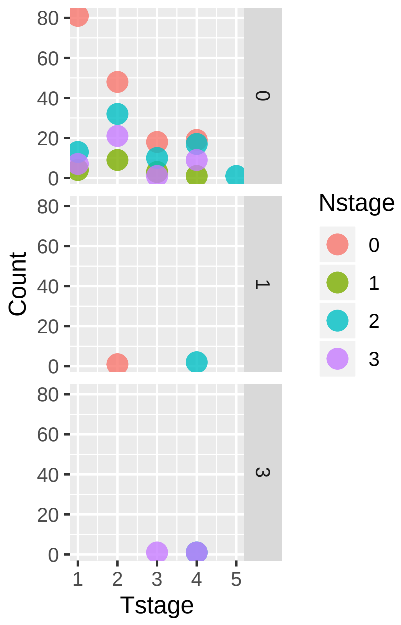
3.3 Explore Clinical data

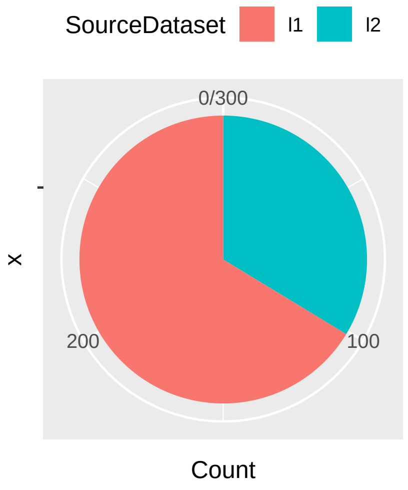
```
p1 <- clinical %>%
  group_by(Histology = stringi::stri_trans_totitle(Histology)) %>% # case insensitive of adenocarcinoma
  group_by(Histology) %>%
  summarise(Count = n()) %>%
  ggplot()+
  aes(x = Histology, y = Count, fill= Histology) +
  geom_col()+
  geom_text(aes(label = percent(Count/sum(Count))), vjust = -0.5)+
  geom_text(aes(label = Count), vjust = -2) +
  theme(axis.text.x = element_text(color="black",size=10,hjust=.5,vjust=.5, angle=5))
```

```r
p2 <- ggplot(data=clinical[!is.na(clinical$age),]) +
  aes(x= age) +
  geom_histogram(fill="blue", bins = 60)  +
  geom_vline(xintercept = c(65,70, 72 ), color = "red")
#coord_flip()

p3 <- clinical %>%
  mutate(Nstage = as.factor(Nstage)) %>%
  group_by(Mstage, Nstage, Tstage) %>%
  summarise(Count = n()) %>%
  ggplot() +
  aes(x = Tstage, y = Count, color = Nstage) +
  facet_grid(Mstage~ .) +
  geom_point(size=4, alpha = 0.8)

p4 <- clinical %>%
  group_by(SourceDataset) %>%
  summarise(Count = n()) %>%
  ggplot()+
  aes(x = "", y = Count, fill = SourceDataset) +
  geom_bar(width = 1, stat = "identity") +
  coord_polar("y", start=0) +
  theme(legend.position = "top")

grid.arrange(p1,p2,p3,p4, layout_matrix = rbind(c(1),c(2, 3, 4)), nrow = 2)
```









- The most frequente cases is `Adenocarcinoma` followed by `Aquamous Cell Carcinoma`.
- NOS: not otherwise specified
- It seems `NOS` and `Nsclc Nos` corespond to the same category
- `Nan` is not available ?
- The density plot shows that the must frequent cases are `65, 70, 72` years old.

_ The most frequent `Nstage` class is also `0`, followed by `2`, `3`, and `1`.

- The third plots shows that the most cases are in `Mstage == 0`. We can focus only in this class.
- There are two sources of dataset.

3.4 Explore output_train and output_test

```
output_train <- fread("output_train.csv")
output_test <- fread("output_test.csv")
head(output_train)
```

```
##    PatientID SurvivalTime Event
## 1:       202         1378     0
## 2:       371          379     1
## 3:       246          573     1
## 4:       240          959     0
## 5:       284         2119     0
## 6:       348          706     1
```

```
head(output_test)
```

```
##    PatientID SurvivalTime Event
## 1:        13     788.4177   NaN
## 2:       155     427.6501   NaN
## 3:       404     173.5872   NaN
## 4:       407     389.8780   NaN
## 5:         9    1580.7672   NaN
## 6:        49     472.5234   NaN
```

- The goal is the fill `Event` variable in output_test by `0` or `1`.

4 Preprocessing of Train and Test dataset

The output of this section is to clean and unify variables and merge clinical, radimoics, and output_train dataset.

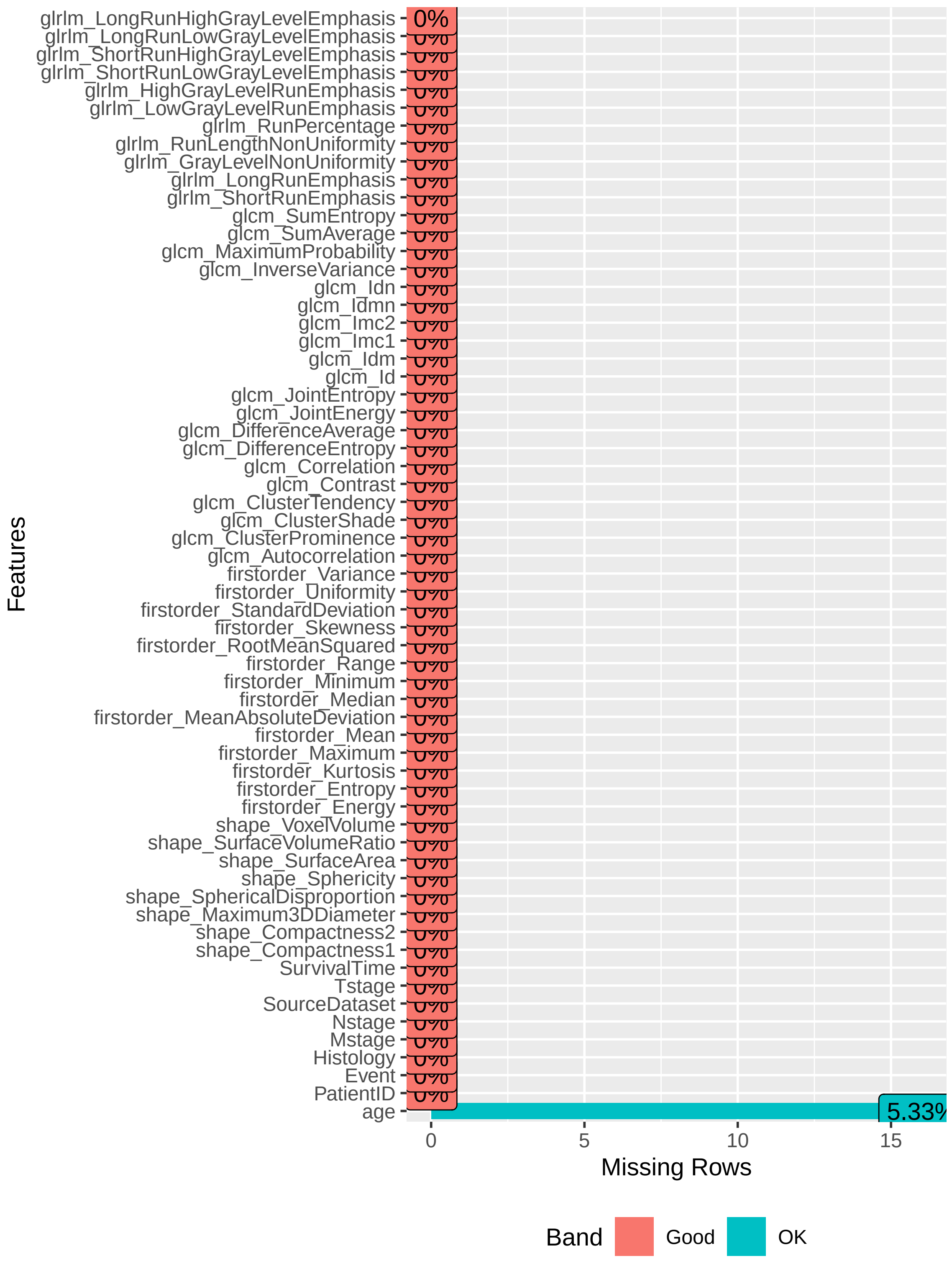
4.1 Train wrangling

```
# Convert character variables to numeric
new_clinical <- clinical %>%
                mutate(Histology = stringi::stri_trans_totitle(Histology)) %>%
                mutate_if(is.character, as.factor) %>%
                mutate_if(is.factor, as.numeric)
                #mutate(Histo = as.numeric(as.factor(Histology))) %>%
                #mutate(Source = as.numeric(as.factor(SourceDataset))) %>%
                #select(everything(), - Histology, -SourceDataset)

train <- new_clinical %>%
  mutate_if(is.character, as.factor) %>%
  left_join(y = output_train, by = "PatientID") %>%
  left_join(y = new_radiomics, by = "PatientID") %>%
  select(PatientID, Event, everything()) %>%
  setDT()



train[,1:10] %>% head()
```

```
##    PatientID Event Histology Mstage Nstage SourceDataset Tstage     age
## 1:       202     0         1      0      0             2      2 66.0000
## 2:       371     1         2      0      2             1      4 64.5722
## 3:       246     1         6      0      3             1      2 66.0452
## 4:       240     0         4      0      2             1      3 59.3566
## 5:       284     0         6      0      3             1      4 71.0554
## 6:       348     1         6      0      2             1      2 65.0212
##    SurvivalTime shape_Compactness1
## 1:         1378         0.02781503
## 2:          379         0.02301549
## 3:          573         0.02734811
## 4:          959         0.02681111
## 5:         2119         0.02369124
## 6:          706         0.03098136
```

4.1.1 Explore missing value in train

```r
library(DataExplorer)
DataExplorer::plot_missing(train)
```



- There are 18 missing `age` from 300.

4.2 Test wrangling

```
radiomics_test <- fread("test/features/radiomics.csv", quote = "")
clinical_test <- fread("test/features/clinical_data.csv")
output_test <- fread("output_test.csv")
```

4.2.1 Transform radiomics test dataset

```
groups_test <- radiomics_test[1:2,-1] %>%
  t() %>%
  as.data.frame() %>%
  rename("Groups" = V1, "Features" = V2)

new_colnames_radiomics_test <- groups_test %>%
  mutate(Features = stringr::str_remove(Features,"original_")) %>%
  pull(Features)

old_names_test <- colnames(radiomics_test)
new_names_test <- c("PatientID", new_colnames_radiomics_test)

new_radiomics_test <- radiomics_test[-1:-3,] %>%
  rename_at(vars(old_names), ~ new_names) %>%
  mutate_if(is.character, as.numeric) #%>%
#as.matrix()


head(new_radiomics_test)[,1:8]
```

```
##   PatientID shape_Compactness1 shape_Compactness2 shape_Maximum3DDiameter
## 1        13         0.02888522         0.29645143               106.90182
## 2       155         0.03194837         0.36266005                18.81489
## 3       404         0.01599883         0.09094503               105.08092
## 4       407         0.03135766         0.34937318                46.96807
## 5         9         0.01781454         0.11275905                56.54202
## 6        49         0.03816202         0.51744596                20.12461
##   shape_SphericalDisproportion shape_Sphericity shape_SurfaceArea
## 1                     1.499738        0.6667830        29085.5414
## 2                     1.402276        0.7131265          629.4436
## 3                     2.223687        0.4497036        12509.2654
## 4                     1.419832        0.7043089         4067.6574
## 5                     2.069901        0.4831149         7093.3657
## 6                     1.245599        0.8028264          844.2344
##   shape_SurfaceVolumeRatio
## 1                0.1145278
## 2                0.7038788
## 3                0.3152977
## 4                0.2821040
## 5                0.3760316
## 6                0.5088176
```

4.2.2 Transform clinical test dataset

```
new_clinical_test <- clinical_test %>%
                mutate(Histology = stringi::stri_trans_totitle(Histology)) %>%
                mutate_if(is.character, as.factor) %>%
                mutate_if(is.factor, as.numeric)
```

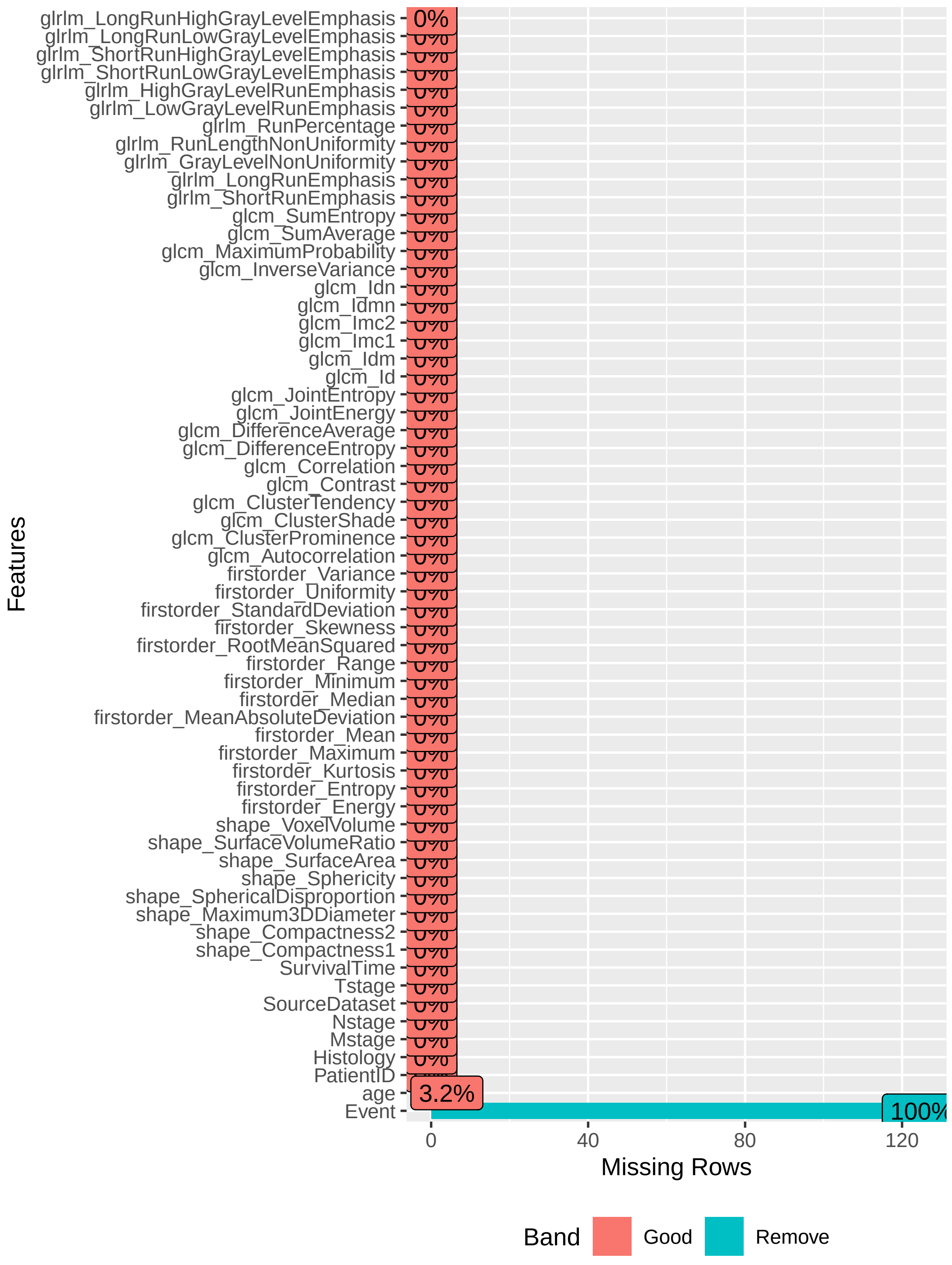
4.2.3 Merge clinical, radimoics, and output_test dataset

```
test <- new_clinical_test %>%
  mutate_if(is.character, as.factor) %>%
  left_join(y = output_test, by = "PatientID") %>%
  left_join(y = new_radiomics_test, by = "PatientID") %>%
  select(PatientID, Event, everything()) %>%
  setDT() # convert to data.table

test[,1:10] %>% head()
```

```
##    PatientID Event Histology Mstage Nstage SourceDataset Tstage     age
## 1:        13   NaN         4      0      0             1      4 44.3970
## 2:       155   NaN         1      0      3             1      1 63.3183
## 3:       404   NaN         2      0      2             1      2 64.7255
## 4:       407   NaN         4      0      0             1      2 65.3635
## 5:         9   NaN         1      0      0             2      2 50.0000
## 6:        49   NaN         6      0      0             1      2 86.1410
##    SurvivalTime shape_Compactness1
## 1:     788.4177         0.02888522
## 2:     427.6501         0.03194837
## 3:     173.5872         0.01599883
## 4:     389.8780         0.03135766
## 5:    1580.7672         0.01781454
## 6:     472.5234         0.03816202
```

4.2.3.1 Explore missing value in test

```
library(DataExplorer)
DataExplorer::plot_missing(test)
```



- There are 4 missing `age` from 125.

5 Xgboost modeling

5.1 Scaling Train and Test dataset

```
trainremoveCols <- c('PatientID','Event')
testremoveCols <- c('PatientID', 'Event')

Event <- train$Event
PatientID <- test$PatientID

train[,(trainremoveCols) := NULL]
test[,(testremoveCols) := NULL]

# Do scaling
dt <- rbind(train, test)
scale.cols <- colnames(dt)
dt[, (scale.cols) := lapply(.SD, scale), .SDcols = scale.cols]
train <- cbind(Event, head(dt,nrow(train)))
test  <- cbind(PatientID, tail(dt, nrow(test)))
rm(dt)
gc()
```

```
##           used (Mb) gc trigger  (Mb) limit (Mb) max used  (Mb)
## Ncells 1315712 70.3    2390648 127.7         NA  2390648 127.7
## Vcells 4792747 36.6   10146329  77.5     102400  7728233  59.0
```

5.2 Split Train dataset into Train & Valid sets

```
library(rsample)

set.seed(100)
train_valid_split <- rsample::initial_split(train, prop = 0.8)
train_valid_split
```

```
## <241/59/300>
```

- We can retrieve our training and testing sets using training() and testing() functions.

```
# Retrieve train and test sets
train_8 <- rsample::training(train_valid_split)
valid_2  <- rsample::testing(train_valid_split)
train_8[1:10, 1:10]
```

```
##     Event  Histology     Mstage     Nstage SourceDataset     Tstage         age
##  1:     0 -1.0235305 -0.1207812 -0.8392371     1.4175490 -0.1386218 -0.2504355
##  2:     1 -0.5217999 -0.1207812  0.8392371    -0.7037831  1.7024495 -0.3972837
##  3:     1  1.4851227 -0.1207812  1.6784742    -0.7037831 -0.1386218 -0.2457867
##  4:     0  1.4851227 -0.1207812  1.6784742    -0.7037831  1.7024495  0.2695086
##  5:     1  1.4851227 -0.1207812  0.8392371    -0.7037831 -0.1386218 -0.3511044
##  6:     1 -1.0235305 -0.1207812 -0.8392371     1.4175490 -1.0591575  0.1609615
##  7:     1 -1.0235305 -0.1207812 -0.8392371    -0.7037831  1.7024495  0.6186818
##  8:     1 -1.0235305 -0.1207812  0.8392371    -0.7037831  1.7024495 -1.5788159
##  9:     0 -0.5217999 -0.1207812  0.8392371    -0.7037831 -0.1386218 -2.0372357
## 10:     1  1.4851227 -0.1207812  0.8392371     1.4175490 -0.1386218  1.1894541
##     SurvivalTime shape_Compactness1 shape_Compactness2
##  1:    0.7487397          0.3273910          0.2206572
##  2:   -0.7068626         -0.4508739         -0.5451008
##  3:   -0.4241931          0.2516768          0.1398098
##  4:    1.8284207         -0.3412982         -0.4460329
##  5:   -0.2304042          0.8408228          0.8050072
##  6:    0.7356262         -0.8618943         -0.8911634
##  7:   -0.9720474          0.3619137          0.2579747
##  8:   -0.9735045         -1.1924952         -1.1402475
##  9:    0.3815607          0.9177119          0.8979347
## 10:   -0.7345467         -0.0934834         -0.2114097
```

5.3 Format train and test to DMatrix

```
library(Matrix)
```

```
##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```
options(na.action='na.pass')
train_8_sparse <- sparse.model.matrix(Event ~., data=train_8)
dtrain_8 <- xgb.DMatrix(data=train_8_sparse, label = train_8$Event)

options(na.action='na.pass')
valid_2_sparse <- sparse.model.matrix(Event ~., data=valid_2)
dvalid_2 <- xgb.DMatrix(data=valid_2_sparse, label = valid_2$Event)
```

5.4 Optimize features with Cross validation

Here, we can see after how many rounds, we achieved the smallest test error.

```
params <- list(booster = "gbtree",
              tree_method = "auto",
              objective = "binary:logistic",
              eval_metric = "auc",          #  for Binary classification error rate
              max_depth = 2,        # 6 makes training heavy, there is no correlation between features
              eta = 0.01,                     # learning rate
              subsample = 0.8,              # prevent overfitting
              colsample_bytree = 0.1         # specify the fraction of columns to be subsampled. # 0.5
             )
```

```
tme <- Sys.time()
cv_model <- xgb.cv(params = params,
                   data = dtrain_8,
                   nthread = parallel::detectCores(all.tests = FALSE, logical = TRUE),  #2,
                   nrounds = 25000,
                   verbose = TRUE,
                   nfold = 5,
                   print_every_n = 100,
                   early_stopping_rounds = 1000,
                   maximize = TRUE,
                   prediction = TRUE) # prediction of cv folds
```

```
## [1]  train-auc:0.744572+0.064073 test-auc:0.702438+0.082770
## Multiple eval metrics are present. Will use test_auc for early stopping.
## Will train until test_auc hasn't improved in 1000 rounds.
##
## [101]    train-auc:0.870199+0.010406 test-auc:0.719389+0.054983
## [201]    train-auc:0.908392+0.009288 test-auc:0.738096+0.067038
## [301]    train-auc:0.937980+0.009603 test-auc:0.742856+0.073792
## [401]    train-auc:0.958381+0.007234 test-auc:0.745781+0.075328
## [501]    train-auc:0.971117+0.005513 test-auc:0.748186+0.078245
## [601]    train-auc:0.981932+0.003549 test-auc:0.748662+0.073363
## [701]    train-auc:0.989035+0.002636 test-auc:0.751144+0.073269
## [801]    train-auc:0.993115+0.002251 test-auc:0.752468+0.075049
## [901]    train-auc:0.995830+0.001267 test-auc:0.752491+0.073045
## [1001]   train-auc:0.997458+0.000986 test-auc:0.750383+0.075110
## [1101]   train-auc:0.998436+0.000822 test-auc:0.752745+0.074776
## [1201]   train-auc:0.999304+0.000382 test-auc:0.754863+0.075870
## [1301]   train-auc:0.999609+0.000281 test-auc:0.757606+0.074092
## [1401]   train-auc:0.999652+0.000233 test-auc:0.758671+0.077658
## [1501]   train-auc:0.999847+0.000149 test-auc:0.757243+0.077075
## [1601]   train-auc:0.999913+0.000128 test-auc:0.760429+0.080661
## [1701]   train-auc:0.999978+0.000043 test-auc:0.761804+0.078747
## [1801]   train-auc:1.000000+0.000000 test-auc:0.761781+0.079641
## [1901]   train-auc:1.000000+0.000000 test-auc:0.764569+0.077992
## [2001]   train-auc:1.000000+0.000000 test-auc:0.763607+0.077886
## [2101]   train-auc:1.000000+0.000000 test-auc:0.764313+0.077829
## [2201]   train-auc:1.000000+0.000000 test-auc:0.765058+0.078533
## [2301]   train-auc:1.000000+0.000000 test-auc:0.764752+0.080319
## [2401]   train-auc:1.000000+0.000000 test-auc:0.765500+0.082560
## [2501]   train-auc:1.000000+0.000000 test-auc:0.765848+0.082679
## [2601]   train-auc:1.000000+0.000000 test-auc:0.765853+0.083729
## [2701]   train-auc:1.000000+0.000000 test-auc:0.767584+0.084333
## [2801]   train-auc:1.000000+0.000000 test-auc:0.767564+0.083388
## [2901]   train-auc:1.000000+0.000000 test-auc:0.768261+0.082905
## [3001]   train-auc:1.000000+0.000000 test-auc:0.767957+0.082808
## [3101]   train-auc:1.000000+0.000000 test-auc:0.767907+0.083715
## [3201]   train-auc:1.000000+0.000000 test-auc:0.767216+0.083291
## [3301]   train-auc:1.000000+0.000000 test-auc:0.767566+0.083038
## [3401]   train-auc:1.000000+0.000000 test-auc:0.766488+0.083823
## [3501]   train-auc:1.000000+0.000000 test-auc:0.765769+0.085249
## [3601]   train-auc:1.000000+0.000000 test-auc:0.765775+0.084047
## [3701]	train-auc:1.000000+0.000000 test-auc:0.766821+0.083965
## [3801]	train-auc:1.000000+0.000000 test-auc:0.768238+0.083915
## [3901]	train-auc:1.000000+0.000000 test-auc:0.768253+0.083865
## [4001]	train-auc:1.000000+0.000000 test-auc:0.769293+0.084696
## [4101]	train-auc:1.000000+0.000000 test-auc:0.767554+0.085023
## [4201]	train-auc:1.000000+0.000000 test-auc:0.767911+0.085678
## [4301]	train-auc:1.000000+0.000000 test-auc:0.769295+0.084663
## [4401]	train-auc:1.000000+0.000000 test-auc:0.768966+0.083995
## [4501]	train-auc:1.000000+0.000000 test-auc:0.769337+0.084664
## [4601]	train-auc:1.000000+0.000000 test-auc:0.770063+0.084524
## [4701]	train-auc:1.000000+0.000000 test-auc:0.770762+0.086182
## [4801]	train-auc:1.000000+0.000000 test-auc:0.770762+0.086182
## [4901]	train-auc:1.000000+0.000000 test-auc:0.770733+0.085839
## [5001]	train-auc:1.000000+0.000000 test-auc:0.771431+0.085685
## [5101]	train-auc:1.000000+0.000000 test-auc:0.772436+0.084950
## [5201]	train-auc:1.000000+0.000000 test-auc:0.771077+0.084316
## [5301]	train-auc:1.000000+0.000000 test-auc:0.771434+0.084976
## [5401]	train-auc:1.000000+0.000000 test-auc:0.771427+0.085900
## [5501]	train-auc:1.000000+0.000000 test-auc:0.772086+0.084872
## [5601]	train-auc:1.000000+0.000000 test-auc:0.772449+0.084947
## [5701]	train-auc:1.000000+0.000000 test-auc:0.772082+0.085432
## [5801]	train-auc:1.000000+0.000000 test-auc:0.773126+0.085734
## [5901]	train-auc:1.000000+0.000000 test-auc:0.773796+0.084919
## [6001]	train-auc:1.000000+0.000000 test-auc:0.775868+0.085087
## [6101]	train-auc:1.000000+0.000000 test-auc:0.775889+0.086328
## [6201]	train-auc:1.000000+0.000000 test-auc:0.775875+0.086345
## [6301]	train-auc:1.000000+0.000000 test-auc:0.775168+0.085948
## [6401]	train-auc:1.000000+0.000000 test-auc:0.775168+0.085948
## [6501]	train-auc:1.000000+0.000000 test-auc:0.775860+0.086386
## [6601]	train-auc:1.000000+0.000000 test-auc:0.775527+0.086600
## [6701]	train-auc:1.000000+0.000000 test-auc:0.775838+0.086254
## [6801]	train-auc:1.000000+0.000000 test-auc:0.776898+0.085454
## [6901]	train-auc:1.000000+0.000000 test-auc:0.777580+0.085361
## [7001]	train-auc:1.000000+0.000000 test-auc:0.778599+0.084680
## [7101]	train-auc:1.000000+0.000000 test-auc:0.777915+0.085147
## [7201]	train-auc:1.000000+0.000000 test-auc:0.778264+0.084891
## [7301]	train-auc:1.000000+0.000000 test-auc:0.778264+0.084891
## [7401]	train-auc:1.000000+0.000000 test-auc:0.778236+0.085371
## [7501]	train-auc:1.000000+0.000000 test-auc:0.778251+0.084939
## [7601]	train-auc:1.000000+0.000000 test-auc:0.778565+0.083907
## [7701]	train-auc:1.000000+0.000000 test-auc:0.778585+0.085129
## [7801]	train-auc:1.000000+0.000000 test-auc:0.778897+0.084486
## [7901]	train-auc:1.000000+0.000000 test-auc:0.778585+0.085129
## [8001]	train-auc:1.000000+0.000000 test-auc:0.779602+0.084841
## [8101]	train-auc:1.000000+0.000000 test-auc:0.778905+0.085364
## [8201]	train-auc:1.000000+0.000000 test-auc:0.779254+0.085127
## [8301]	train-auc:1.000000+0.000000 test-auc:0.779268+0.085049
## [8401]	train-auc:1.000000+0.000000 test-auc:0.778200+0.084589
## [8501]	train-auc:1.000000+0.000000 test-auc:0.777856+0.083875
## [8601]	train-auc:1.000000+0.000000 test-auc:0.778190+0.083670
## [8701]	train-auc:1.000000+0.000000 test-auc:0.778190+0.083670
## [8801]	train-auc:1.000000+0.000000 test-auc:0.777508+0.084157
## [8901]	train-auc:1.000000+0.000000 test-auc:0.777158+0.084397
## [9001]	train-auc:1.000000+0.000000 test-auc:0.777158+0.084397
## [9101]	train-auc:1.000000+0.000000 test-auc:0.777174+0.084371
## [9201]	train-auc:1.000000+0.000000 test-auc:0.776489+0.084827
## Stopping. Best iteration:
## [8221]	train-auc:1.000000+0.000000 test-auc:0.779952+0.084599
```

```r
Sys.time() - tme
```

```
## Time difference of 29.97534 secs
```

5.5 Train the model

```r
watchlist <- list(train = dtrain_8, eval = dvalid_2)
tme <- Sys.time()
xgboost_tree <- xgb.train(data = dtrain_8,
                          params = params,
                          watchlist = watchlist,
                          nrounds = cv_model$best_iteration, # more than 12000 ~0.897
                          print_every_n = 500,
                          verbose = TRUE)
```

```
## [1]	train-auc:0.693182	eval-auc:0.586207
## [501]	train-auc:0.957117	eval-auc:0.841379
## [1001]	train-auc:0.993397	eval-auc:0.843678
## [1501]	train-auc:0.999096	eval-auc:0.866667
## [2001]	train-auc:0.999931	eval-auc:0.872414
## [2501]	train-auc:1.000000	eval-auc:0.878161
## [3001]	train-auc:1.000000	eval-auc:0.881609
## [3501]	train-auc:1.000000	eval-auc:0.880460
## [4001]	train-auc:1.000000	eval-auc:0.880460
## [4501]	train-auc:1.000000	eval-auc:0.881609
## [5001]	train-auc:1.000000	eval-auc:0.877011
## [5501]	train-auc:1.000000	eval-auc:0.879310
## [6001]	train-auc:1.000000	eval-auc:0.880460
## [6501]	train-auc:1.000000	eval-auc:0.879310
## [7001]	train-auc:1.000000	eval-auc:0.877011
## [7501]	train-auc:1.000000	eval-auc:0.874713
## [8001]	train-auc:1.000000	eval-auc:0.872414
## [8221]	train-auc:1.000000	eval-auc:0.871264
```

```r
Sys.time() - tme
```

```
## Time difference of 8.718807 secs
```

5.6 Predict valid_2 dataset

```r
pred_valid <- predict(xgboost_tree, dvalid_2)

summary(pred_valid)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max. 
## 0.0008307 0.0456546 0.5756907 0.4909866 0.8893077 0.9995762
```

- We suppose that if Prob $>$ 0.575 (Median), the Event is 1, else 0

5.7 Transform propability to binary classification

```r
pred_bin <- as.numeric(pred_valid >= 0.5)
table(pred_bin)
```

```
## pred_bin
##  0  1
## 28 31
```

5.8 Confusion matrix for Tree model

```r
data.frame(prediction = as.numeric(pred_bin),
         label = as.numeric(valid_2$Event)) %>%
         count(prediction, label)
```

```
## # A tibble: 4 x 3
##   prediction label     n
##        <dbl> <dbl> <int>
## 1          0     0    22
## 2          0     1     6
## 3          1     0     7
## 4          1     1    24
```

5.9 Extract the most important features from tree xgboost model

5.9.1 List the most important features

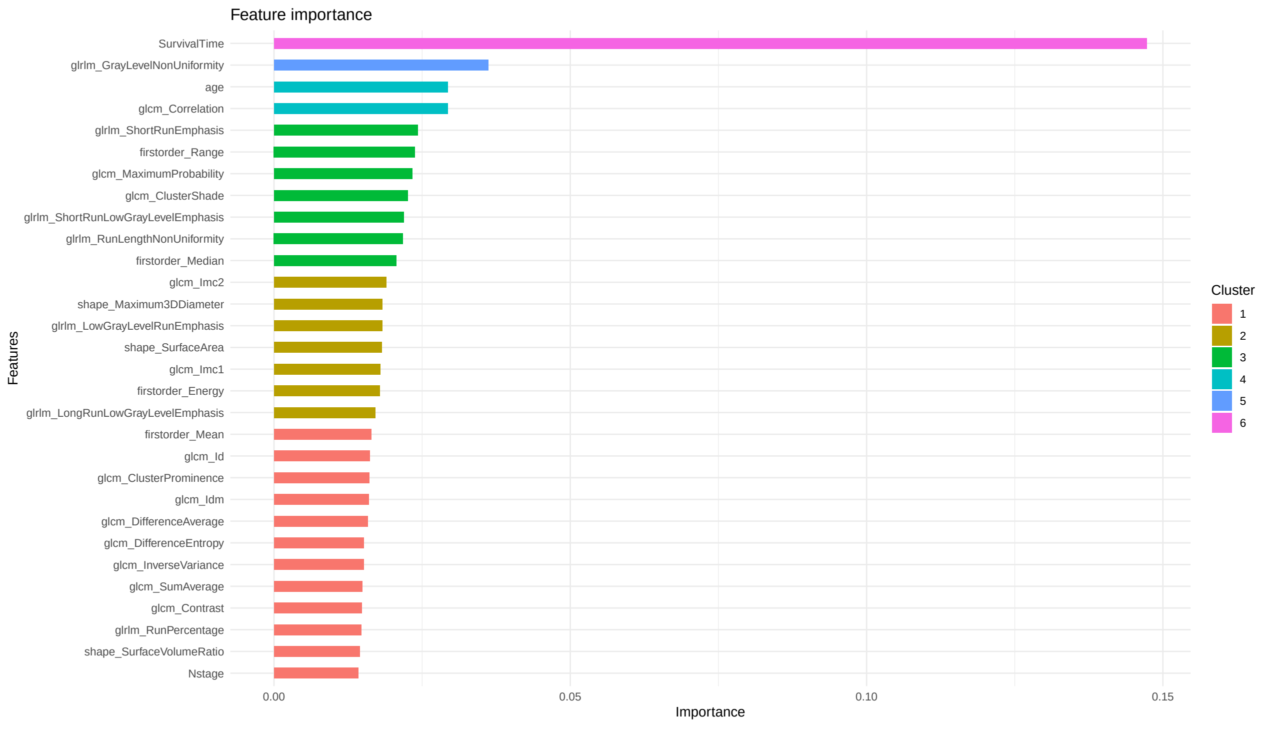
```r
features <- colnames(train_8)
importance_matrix_tree <- xgb.importance(features, model = xgboost_tree)
importance_matrix_tree
```

```
##                                  Feature         Gain        Cover    Frequency
##  1:                         SurvivalTime 0.1473016576 0.0606032617 0.0443522942
##  2:          glrlm_GrayLevelNonUniformity 0.0361461906 0.0329921725 0.0303795186
##  3:                                  age 0.0293497900 0.0344337912 0.0280356982
##  4:                     glcm_Correlation 0.0293344404 0.0316390293 0.0316415758
##  5:                glrlm_ShortRunEmphasis 0.0242824351 0.0211755591 0.0200126206
##  6:                    firstorder_Range 0.0237971560 0.0273691845 0.0287118002
##  7:              glcm_MaximumProbability 0.0233000328 0.0240000228 0.0250608492
##  8:                    glcm_ClusterShade 0.0226094231 0.0247564691 0.0221310737
##  9:  glrlm_ShortRunLowGrayLevelEmphasis 0.0218925972 0.0223398825 0.0200126206
## 10:         glrlm_RunLengthNonUniformity 0.0217701604 0.0247601084 0.0232579104
## 11:                    firstorder_Median 0.0206514365 0.0202149659 0.0173983593
## 12:                            glcm_Imc2 0.0189165892 0.0241984645 0.0246551880
## 13:              shape_Maximum3DDiameter 0.0183020510 0.0208021222 0.0216803390
## 14:         glrlm_LowGrayLevelRunEmphasis 0.0182646503 0.0207184040 0.0198323267
## 15:                    shape_SurfaceArea 0.0181659458 0.0176602938 0.0182096818
## 16:                            glcm_Imc1 0.0179234187 0.0181160355 0.0160461552
## 17:                    firstorder_Energy 0.0178565413 0.0203080998 0.0206887226
## 18:   glrlm_LongRunLowGrayLevelEmphasis 0.0170534424 0.0207731122 0.0201027675
## 19:                      firstorder_Mean 0.0164444669 0.0193970561 0.0187956369
## 20:                              glcm_Id 0.0161800000 0.0137571279 0.0128910124
## 21:               glcm_ClusterProminence 0.0160688179 0.0204573920 0.0203732083
## 22:                             glcm_Idm 0.0160089908 0.0146376942 0.0150094654
## 23:               glcm_DifferenceAverage 0.0157883970 0.0146849868 0.0157306409
## 24:               glcm_DifferenceEntropy 0.0151487822 0.0169302018 0.0162264491
## 25:                 glcm_InverseVariance 0.0151286284 0.0193457245 0.0237987920
## 26:                      glcm_SumAverage 0.0149163557 0.0180440007 0.0180744614
## 27:                        glcm_Contrast 0.0147771600 0.0168717437 0.0160461552
## 28:                  glrlm_RunPercentage 0.0147585440 0.0126227537 0.0126656450
## 29:             shape_SurfaceVolumeRatio 0.0144804480 0.0133940707 0.0146488777
## 30:                               Nstage 0.0142376516 0.0173331222 0.0131163797
## 31:                            glcm_Idmn 0.0142023419 0.0121121202 0.0114486613
## 32:  glrlm_LongRunHighGrayLevelEmphasis 0.0141824721 0.0155633959 0.0180293879
## 33:                   shape_Compactness1 0.0134119067 0.0166033858 0.0192463716
## 34:                    shape_VoxelVolume 0.0130344702 0.0124567191 0.0123050572
## 35:                 glcm_Autocorrelation 0.0129387816 0.0142384976 0.0139277022
## 36: glrlm_ShortRunHighGrayLevelEmphasis 0.0128974180 0.0186926076 0.0205985757
## 37:    firstorder_MeanAbsoluteDeviation 0.0125793229 0.0142261158 0.0152348328
## 38:        shape_SphericalDisproportion 0.0124878567 0.0146265292 0.0167222573
## 39:                     shape_Sphericity 0.0123218032 0.0149262125 0.0159560083
## 40:                   firstorder_Maximum 0.0123047946 0.0179208467 0.0198323267
## 41:                             glcm_Idn 0.0121079516 0.0126515402 0.0126656450
## 42:                  firstorder_Skewness 0.0120363070 0.0131467075 0.0149193185
## 43:                    glcm_JointEnergy 0.0118810705 0.0091120524 0.0094203552
## 44:           firstorder_RootMeanSquared 0.0112659617 0.0140719255 0.0155052736
## 45:                   shape_Compactness2 0.0112622210 0.0149530392 0.0173082124
## 46:                   firstorder_Entropy 0.0100879556 0.0120711812 0.0137023348
## 47:                   firstorder_Minimum 0.0098321660 0.0125515927 0.0144235103
## 48:                  firstorder_Kurtosis 0.0093894713 0.0116019425 0.0141079960
## 49:                    glcm_JointEntropy 0.0091521962 0.0096012586 0.0107725593
## 50:                  firstorder_Variance 0.0090218473 0.0101489612 0.0100063103
## 51:      glrlm_HighGrayLevelRunEmphasis 0.0089822955 0.0115605924 0.0135671144
## 52:                glrlm_LongRunEmphasis 0.0079809619 0.0083208852 0.0089245470
## 53:        firstorder_StandardDeviation 0.0079070265 0.0101078093 0.0107274858
## 54:                      glcm_SumEntropy 0.0075818420 0.0094132818 0.0102767511
## 55:                firstorder_Uniformity 0.0075086249 0.0090427880 0.0108176327
## 56:                 glcm_ClusterTendency 0.0073854350 0.0089882640 0.0105922654
## 57:                            Histology 0.0028452880 0.0044211127 0.0059496980
## 58:                               Tstage 0.0018435905 0.0019519711 0.0030199225
## 59:                        SourceDataset 0.0007104185 0.0005798108 0.0004056612
##                                Feature         Gain        Cover    Frequency
```

- Survival Time is the most important feature, followed by age, and 8 texture description.

5.9.2 Plot the most important features (Tree model)

```r
library(Ckmeans.1d.dp)
xgb.ggplot.importance(importance_matrix_tree[1:30,]) +
ggplot2::theme_minimal()
```



5.10 Test Xgboost Prediction

5.10.1 Load test data and format to DMatrix

```
test_sparse <- sparse.model.matrix(PatientID ~., data=test)
 dtest <- xgb.DMatrix(data=test_sparse, label = test$PatientID)
```

```
pred_tree <- predict(xgboost_tree, dtest)
head(pred_tree)
```

```
## [1] 0.22829722 0.94555700 0.93745029 0.98050964 0.01068046 0.93734211
```

```
summary(pred_tree)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00421 0.17358 0.63923 0.53808 0.89808 0.99944
```

5.11 submission

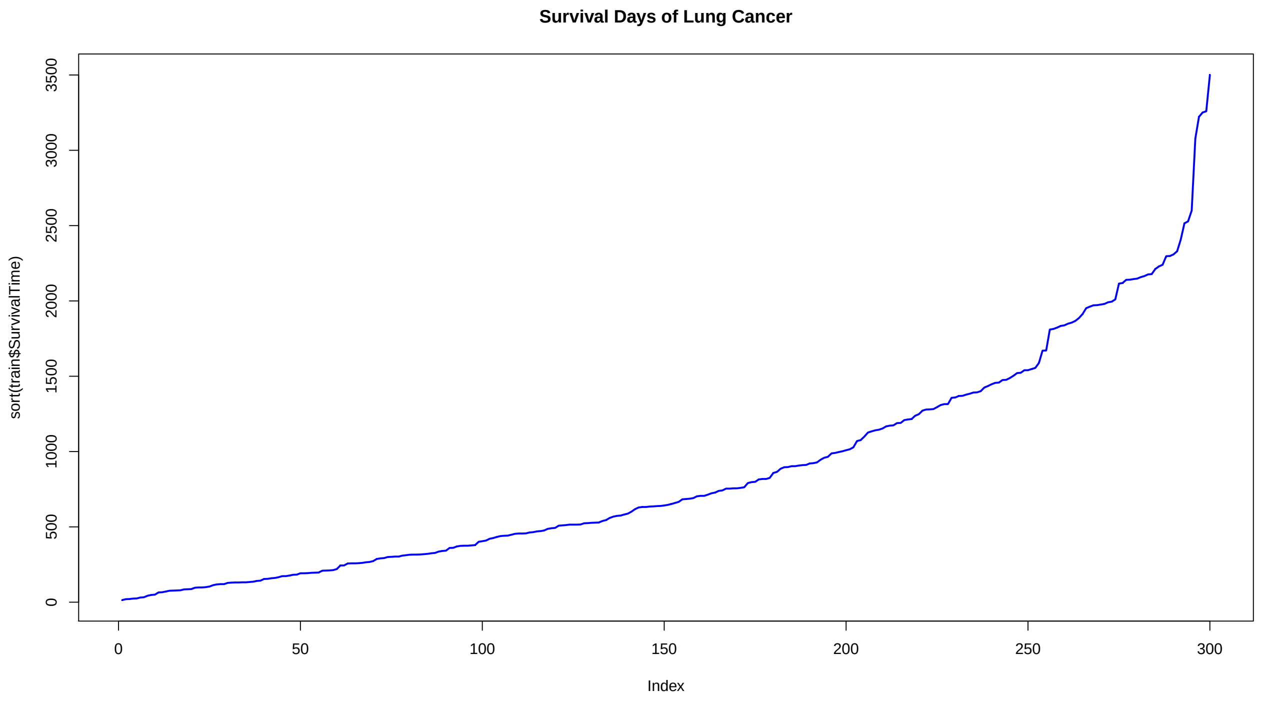
```
pred <- data.frame(
  PatientID = PatientID,
  Event = pred_tree
)

submission <- output_test %>%
  select(PatientID, SurvivalTime) %>%
  left_join(pred, by = "PatientID")

fwrite(submission, "submission.csv")
```

6 Random Forest Survival model (ranger package)

This method will give us an outcome probability over a time continuum (flipping the non-event to event probability)

Let's take a quick look at the time period range in the training portion of our data set:

```r
# reset train and test dataset
train <-new_clinical %>%
  mutate_if(is.character, as.factor) %>%
  left_join(y = output_train, by = "PatientID") %>%
  left_join(y = new_radiomics, by = "PatientID") %>%
  select(PatientID, Event, everything()) %>%
  setDT() # convert to data.table


 test <- new_clinical_test %>%
  mutate_if(is.character, as.factor) %>%
  left_join(y = output_test, by = "PatientID") %>%
  left_join(y = new_radiomics_test, by = "PatientID") %>%
  select(PatientID, Event, everything()) %>%
  setDT() # convert to data.table

plot(sort(train$SurvivalTime), pch='.', type='o',
     col='blue', lwd=2 ,
     main = 'Survival Days of Lung Cancer')
```



6.1 Dealing with missing Age

6.1.1 Check the data for missing values.

```r
sapply(train, function(x) sum(is.na(x)))
```

```
##                          PatientID                              Event
##                                  0                                  0
##                          Histology                             Mstage
##                                  0                                  0
##                             Nstage                      SourceDataset
##                                  0                                  0
##                             Tstage                                age
##                                  0                                 16
##                       SurvivalTime                shape_Compactness1
##                                  0                                  0
##                 shape_Compactness2           shape_Maximum3DDiameter
##                                  0                                  0
##       shape_SphericalDisproportion                  shape_Sphericity
##                                  0                                  0
##                  shape_SurfaceArea         shape_SurfaceVolumeRatio
##                                  0                                  0
##                  shape_VoxelVolume                 firstorder_Energy
##                                  0                                  0
##                 firstorder_Entropy               firstorder_Kurtosis
##                                  0                                  0
##                 firstorder_Maximum                   firstorder_Mean
##                                  0                                  0
##    firstorder_MeanAbsoluteDeviation                 firstorder_Median
##                                  0                                  0
##                 firstorder_Minimum                  firstorder_Range
##                                  0                                  0
##           firstorder_RootMeanSquared               firstorder_Skewness
##                                  0                                  0
##       firstorder_StandardDeviation             firstorder_Uniformity
##                                  0                                  0
##                firstorder_Variance             glcm_Autocorrelation
##                                  0                                  0
##            glcm_ClusterProminence                 glcm_ClusterShade
##                                  0                                  0
##             glcm_ClusterTendency                     glcm_Contrast
##                                  0                                  0
##                   glcm_Correlation           glcm_DifferenceEntropy
##                                  0                                  0
##            glcm_DifferenceAverage                 glcm_JointEnergy
##                                  0                                  0
##                 glcm_JointEntropy                          glcm_Id
##                                  0                                  0
##                          glcm_Idm                         glcm_Imc1
##                                  0                                  0
##                         glcm_Imc2                         glcm_Idmn
##                                  0                                  0
##                          glcm_Idn              glcm_InverseVariance
##                                  0                                  0
##             glcm_MaximumProbability                        glcm_SumAverage
##                                   0                                      0
##                     glcm_SumEntropy              glcm_ShortRunEmphasis
##                                   0                                      0
##               glrlm_LongRunEmphasis        glrlm_GrayLevelNonUniformity
##                                   0                                      0
##         glrlm_RunLengthNonUniformity                 glrlm_RunPercentage
##                                   0                                      0
##        glrlm_LowGrayLevelRunEmphasis      glrlm_HighGrayLevelRunEmphasis
##                                   0                                      0
##   glrlm_ShortRunLowGrayLevelEmphasis glrlm_ShortRunHighGrayLevelEmphasis
##                                   0                                      0
##    glrlm_LongRunLowGrayLevelEmphasis  glrlm_LongRunHighGrayLevelEmphasis
##                                   0                                      0
```

6.1.2 Imputation processing for train data

```r
library(mice)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'mice'
```

```
## The following object is masked from 'package:tidyr':
##
##     complete
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```r
init = mice(train, maxit=0)
```

```
## Warning: Number of logged events: 1
```

```r
meth = init$method
predM = init$predictorMatrix
```

- We may not want to use a certain variable as predictors. For example, the PatitenID variable does not have any predictive value.

```r
predM[, c("PatientID")] <- 0
```

- If we want to skip a variable from imputation use the code below. This variable will be used for prediction.

```r
#colnames(train)[train[, !names(train) %in% c("PatientID")]]

meth[c("PatientID")]=""
```

- Now let specify the methods for imputing the missing values. There are specific methods for continues, binary and ordinal variables. I set different methods for each variable. You can add more than one variable in each method.

```r
meth[c("age")]="cart"  # pmm (Predictive Mean Matching suitable for numeric variables )

# pmm generate error. it is seems working with cart
# (https://stackoverflow.com/questions/48355250/
#do-imputation-in-r-when-mice-returns-error-that-system-is-computationally-singu)
```

- Now it is time to run the multiple (m=5) imputation.

```
set.seed(103)
imputed = mice(train, method=meth, predictorMatrix=predM, m=5)
```

```
##
##  iter imp variable
##   1   1  age
##   1   2  age
##   1   3  age
##   1   4  age
##   1   5  age
##   2   1  age
##   2   2  age
##   2   3  age
##   2   4  age
##   2   5  age
##   3   1  age
##   3   2  age
##   3   3  age
##   3   4  age
##   3   5  age
##   4   1  age
##   4   2  age
##   4   3  age
##   4   4  age
##   4   5  age
##   5   1  age
##   5   2  age
##   5   3  age
##   5   4  age
##   5   5  age
```

```
## Warning: Number of logged events: 25
```

- Create a dataset after imputation.

```
imputed <- complete(imputed)
```

- Check for missings in the imputed dataset.

```
sapply(imputed, function(x) sum(is.na(x)))
```

```
##                            PatientID                                Event
##                                    0                                    0
##                            Histology                               Mstage
##                                    0                                    0
##                               Nstage                        SourceDataset
##                                    0                                    0
##                               Tstage                                  age
##                                    0                                    0
##                         SurvivalTime                    shape_Compactness1
##                                    0                                    0
##                   shape_Compactness2               shape_Maximum3DDiameter
##                                   0                                   0
##        shape_SphericalDisproportion                    shape_Sphericity
##                                   0                                   0
##                   shape_SurfaceArea            shape_SurfaceVolumeRatio
##                                   0                                   0
##                   shape_VoxelVolume                   firstorder_Energy
##                                   0                                   0
##                  firstorder_Entropy                 firstorder_Kurtosis
##                                   0                                   0
##                  firstorder_Maximum                     firstorder_Mean
##                                   0                                   0
##    firstorder_MeanAbsoluteDeviation                   firstorder_Median
##                                   0                                   0
##                  firstorder_Minimum                    firstorder_Range
##                                   0                                   0
##          firstorder_RootMeanSquared                 firstorder_Skewness
##                                   0                                   0
##        firstorder_StandardDeviation               firstorder_Uniformity
##                                   0                                   0
##                 firstorder_Variance                glcm_Autocorrelation
##                                   0                                   0
##              glcm_ClusterProminence                   glcm_ClusterShade
##                                   0                                   0
##                glcm_ClusterTendency                       glcm_Contrast
##                                   0                                   0
##                    glcm_Correlation              glcm_DifferenceEntropy
##                                   0                                   0
##              glcm_DifferenceAverage                    glcm_JointEnergy
##                                   0                                   0
##                   glcm_JointEntropy                             glcm_Id
##                                   0                                   0
##                            glcm_Idm                           glcm_Imc1
##                                   0                                   0
##                           glcm_Imc2                           glcm_Idmn
##                                   0                                   0
##                            glcm_Idn                glcm_InverseVariance
##                                   0                                   0
##             glcm_MaximumProbability                     glcm_SumAverage
##                                   0                                   0
##                     glcm_SumEntropy              glrlm_ShortRunEmphasis
##                                   0                                   0
##               glrlm_LongRunEmphasis        glrlm_GrayLevelNonUniformity
##                                   0                                   0
##        glrlm_RunLengthNonUniformity                 glrlm_RunPercentage
##                                   0                                   0
##       glrlm_LowGrayLevelRunEmphasis      glrlm_HighGrayLevelRunEmphasis
##                                   0                                   0
##  glrlm_ShortRunLowGrayLevelEmphasis glrlm_ShortRunHighGrayLevelEmphasis
##                                   0                                   0
##   glrlm_LongRunLowGrayLevelEmphasis  glrlm_LongRunHighGrayLevelEmphasis
##                                   0                                   0
```

- Accuracy

```
print("train$age") ; summary(train$age)
```

```
## [1] "train$age"
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   42.51   62.98   69.95   68.77   76.20   87.13      16
```

```
print("imputed$age") ; summary(imputed$age)
```

```
## [1] "imputed$age"
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   42.51   63.00   70.00   68.88   76.35   87.13
```

- Well done :-)

6.1.3 Do the same imputation for test

```
init = mice(test, maxit=0)
```

```
## Warning: Number of logged events: 2
```

```
meth = init$method
predM = init$predictorMatrix

predM[, c("PatientID")] <- 0
meth[c("PatientID")]=""
meth[c("age")]="cart"

set.seed(103)
imputed_test = mice(test, method=meth, predictorMatrix=predM, m=5)
```

```
##
##  iter imp variable
##   1   1  age
##   1   2  age
##   1   3  age
##   1   4  age
##   1   5  age
##   2   1  age
##   2   2  age
##   2   3  age
##   2   4  age
##   2   5  age
##   3   1  age
##   3   2  age
##   3   3  age
##   3   4  age
##   3   5  age
##   4   1  age
##   4   2  age
##   4   3  age
##   4   4  age
##   4   5  age
##   5   1  age
##   5   2  age
##   5   3  age
##   5   4  age
##   5   5  age
```

```
## Warning: Number of logged events: 25
```

```r
imputed_test <- complete(imputed_test)

imputed_test[1:10,1:10]
```

```
##    PatientID Event Histology Mstage Nstage SourceDataset Tstage     age
## 1         13    NA         4      0      0             1      4 44.3970
## 2        155    NA         1      0      3             1      1 63.3183
## 3        404    NA         2      0      2             1      2 64.7255
## 4        407    NA         4      0      0             1      2 65.3635
## 5          9    NA         1      0      0             2      2 50.0000
## 6         49    NA         6      0      0             1      2 86.1410
## 7         55    NA         3      0      0             1      1 75.2663
## 8        200    NA         3      0      0             1      1 85.4511
## 9        170    NA         2      0      3             1      1 69.8727
## 10       387    NA         2      0      3             1      2 52.8569
##    SurvivalTime shape_Compactness1
## 1      788.4177         0.02888522
## 2      427.6501         0.03194837
## 3      173.5872         0.01599883
## 4      389.8780         0.03135766
## 5     1580.7672         0.01781454
## 6      472.5234         0.03816202
## 7     1970.9725         0.03699879
## 8      530.4248         0.03373779
## 9     1067.4630         0.01929334
## 10     378.3248         0.02531470
```

6.2 Preprocessing Train and Test

In order to measure the AUC of each model we need to split the data randomly (with seed) into two equal parts:

```r
set.seed(1234)
random_splits <- runif(nrow(imputed))
train_df_official <- imputed[random_splits < .5,]
dim(train_df_official)
```

```
## [1] 151  62
```

```r
validate_df_official <- imputed[random_splits >= .5,]
dim(validate_df_official)
```

```
## [1] 149  62
```

In order to align the survival and the classification models, we will focus on the probability of reaching event over the first `quantile(test$SurvivalTime)[[2]]` days.

```r
period_choice <- round(quantile(test$SurvivalTime)[[2]])
period_choice <- 825
```

- We also need to create a classification-centric outcome variable. This will measure how many patients reached event or not within the chosen period. Here we look for a censor feature of 1 (i.e. the event happened) under the chosen period to set the outcome to 1, everything else is set to 0:

```
# classification data set
train_df_classificaiton  <- train_df_official

train_df_classificaiton$ReachedEvent <- ifelse((train_df_classificaiton$Event == 1 &
                                         train_df_classificaiton$SurvivalTime <= period_choice), 1, 0)

summary(train_df_classificaiton$ReachedEvent)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   0.000   0.457   1.000   1.000
```

```
validate_df_classification  <- validate_df_official

validate_df_classification$ReachedEvent <- ifelse((validate_df_classification$Event == 1 &
                                        validate_df_classification$SurvivalTime <= period_choice), 1, 0

summary(validate_df_classification$ReachedEvent)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.4228  1.0000  1.0000
```

- Now we can easily get an AUC score on the probability of reaching event within our allotted period choice

6.3 Survival Model (ranger)

```
# omit PatientID from variable importance
var <- paste(colnames(train)[train[, !names(train) %in% c("PatientID",
                                                           "Event",
                                                           "SurvivalTime")]],
              collapse ="+")

survival_formula <- formula(paste('Surv(', 'SurvivalTime', ',', 'Event', ') ~ ', var))


survival_formula
```

```
## Surv(SurvivalTime, Event) ~ Histology + Mstage + Nstage + SourceDataset +
##     Tstage + age + shape_Compactness1 + shape_Compactness2 +
##     shape_Maximum3DDiameter + shape_SphericalDisproportion +
##     shape_Sphericity + shape_SurfaceArea + shape_SurfaceVolumeRatio +
##     shape_VoxelVolume + firstorder_Energy + firstorder_Entropy +
##     firstorder_Kurtosis + firstorder_Maximum + firstorder_Mean +
##     firstorder_MeanAbsoluteDeviation + firstorder_Median + firstorder_Minimum +
##     firstorder_Range + firstorder_RootMeanSquared + firstorder_Skewness +
##     firstorder_StandardDeviation + firstorder_Uniformity + firstorder_Variance +
##     glcm_Autocorrelation + glcm_ClusterProminence + glcm_ClusterShade +
##     glcm_ClusterTendency + glcm_Contrast + glcm_Correlation +
##     glcm_DifferenceEntropy + glcm_DifferenceAverage + glcm_JointEnergy +
##     glcm_JointEntropy + glcm_Id + glcm_Idm + glcm_Imc1 + glcm_Imc2 +
##     glcm_Idmn + glcm_Idn + glcm_InverseVariance + glcm_MaximumProbability +
##     glcm_SumAverage + glcm_SumEntropy + glrlm_ShortRunEmphasis +
##     glrlm_LongRunEmphasis + glrlm_GrayLevelNonUniformity + glrlm_RunLengthNonUniformity +
##     glrlm_RunPercentage + glrlm_LowGrayLevelRunEmphasis + glrlm_HighGrayLevelRunEmphasis +
##     glrlm_ShortRunLowGrayLevelEmphasis + glrlm_ShortRunHighGrayLevelEmphasis +
##     glrlm_LongRunLowGrayLevelEmphasis + glrlm_LongRunHighGrayLevelEmphasis
```

```
survival_model <- ranger(survival_formula,
                data = train_df_official,
                seed = 1234,
                importance = 'permutation',
                mtry = 2,
                verbose = TRUE,
                num.trees = 50,
                write.forest=TRUE)

# print out coefficients
sort(survival_model$variable.importance, decreasing = TRUE) %>% head(20)
```

```
##                            glcm_Idm                            glcm_Idn
##                         0.010874783                         0.008679476
##             glcm_MaximumProbability             glrlm_ShortRunEmphasis
##                         0.006445334                         0.005767017
##                 glrlm_RunPercentage                           glcm_Imc1
##                         0.005115244                         0.004855692
##                     firstorder_Mean                   glcm_JointEntropy
##                         0.004413386                         0.004139470
##                     glcm_SumEntropy        glrlm_RunLengthNonUniformity
##                         0.003978832                         0.003897940
##                              Tstage                           glcm_Idmn
##                         0.003583321                         0.003427321
##                   firstorder_Median                                 age
##                         0.003387350                         0.003266305
##                 firstorder_Variance                  firstorder_Entropy
##                         0.003228114                         0.003219570
## glrlm_ShortRunLowGrayLevelEmphasis                       SourceDataset
##                         0.003183818                         0.003181861
##   firstorder_MeanAbsoluteDeviation                     glcm_SumAverage
##                         0.003039612                         0.002837739
```

- The orange PatientID has more probability to survive.

```
print("Orange PatientID: "); imputed[1,c(2,7,8,9)]
```

```
## [1] "Orange PatientID: "
```

```
##   Event Tstage age SurvivalTime
## 1     0      2  66         1378
```

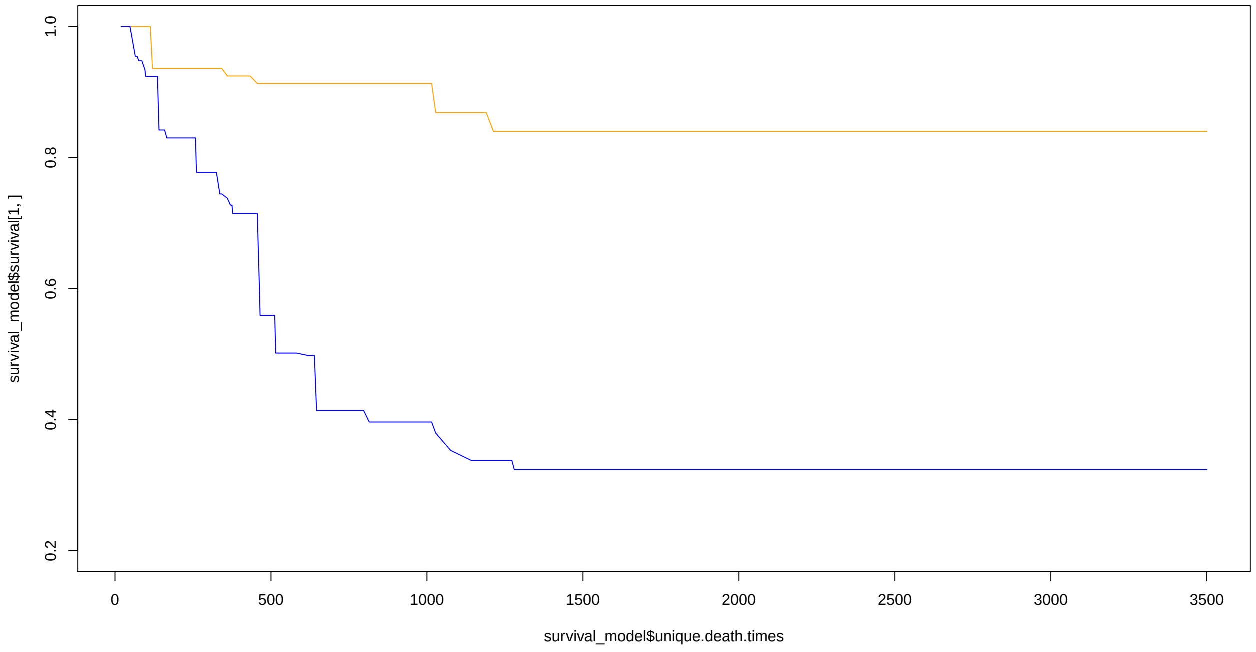
```
print("Blue PatientID: "); imputed[56,c(2,7,8,9)]
```

```
## [1] "Blue PatientID: "
```

```
##    Event Tstage     age SurvivalTime
## 56     1      4 62.4476          303
```

- Once we have our survival_model model object, we can take a look at some probabilities of survival (this is just for illustrative purposes as we haven't split our data set yet). Let's look at two patients - row 1 and row 56:

```r
plot(survival_model$unique.death.times, survival_model$survival[1,],
     type='l', col='orange', ylim=c(0.2,1))

lines(survival_model$unique.death.times,
      survival_model$survival[56,], col='blue')
```



6.3.1 Scoring the Random Forest Survival Model

First we get the basic survival prediction using our validation split set and then we flip the probability of the period of choice and get the AUC score:

```r
feature_names <- setdiff(names(train_df_classificaiton),
                         c('PatientID', 'ReachedEvent',
                           'SurvivalTime', 'Event'))


suvival_predictions <- predict( survival_model,
                                validate_df_official[, feature_names])

roc(response = validate_df_classification$ReachedEvent,
    predictor = 1 - suvival_predictions$survival[,which(suvival_predictions$unique.death.times
                                                        == period_choice)])
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

##
## Call:
## roc.default(response = validate_df_classification$ReachedEvent,     predictor = 1 - suvival_predicti
##
## Data: 1 - suvival_predictions$survival[, which(suvival_predictions$unique.death.times == period_choi
## Area under the curve: 0.7171
```

6.3.2 RF survival Prediction

```
survival_predictions_test <- predict( survival_model,imputed_test[, feature_names])

predictor <- 1 -
  survival_predictions_test$survival[,which(survival_predictions_test$unique.death.times
                                                          == period_choice)]

pred <- data.frame(
  PatientID = PatientID,
  Event = predictor
)

submission_surv <- output_test %>%
  select(PatientID, SurvivalTime) %>%
  left_join(pred, by = "PatientID")

fwrite(submission_surv, "submission_surv.csv")
```

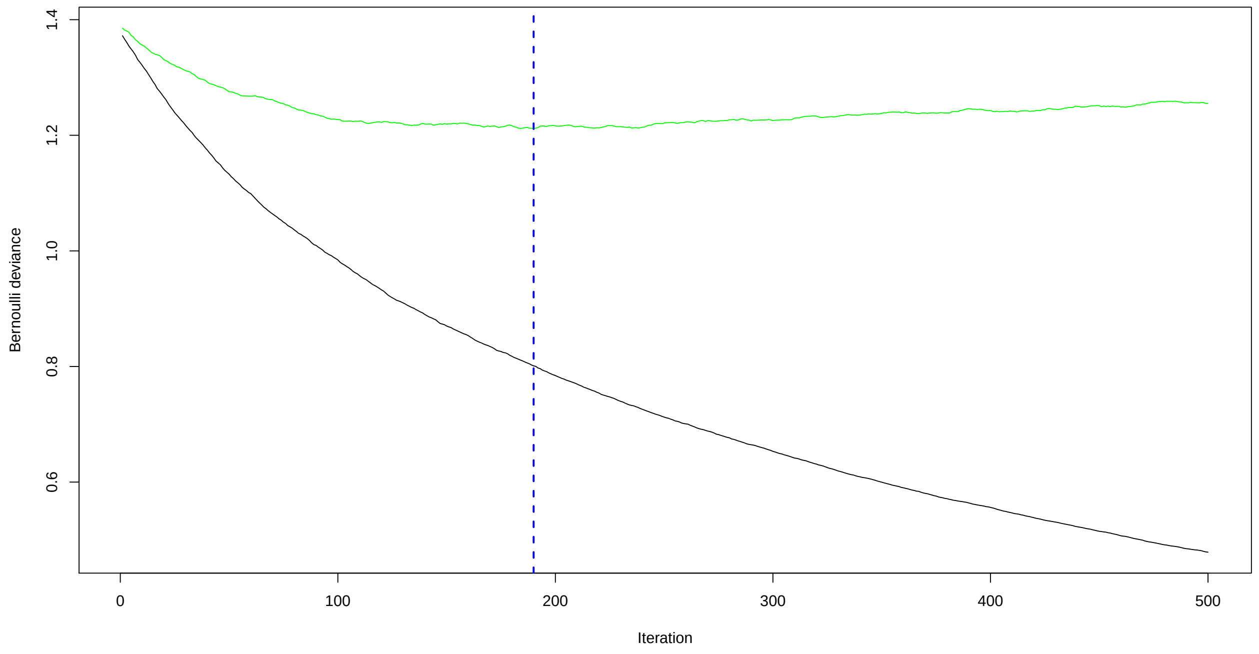
7 GBM Classification modeling

- Let's run and score our classification GBM model:

```
require(gbm)



var <- paste(colnames(train_df_classificaiton)[ !(names(train_df_classificaiton) %in%
                             c("PatientID", 'SurvivalTime' , "ReachedEvent", "Event"))],
             collapse ="+")


classification_formula <- formula(paste('ReachedEvent ~ ', var))

classification_formula
```

```
## ReachedEvent ~ Histology + Mstage + Nstage + SourceDataset +
##     Tstage + age + shape_Compactness1 + shape_Compactness2 +
##     shape_Maximum3DDiameter + shape_SphericalDisproportion +
##     shape_Sphericity + shape_SurfaceArea + shape_SurfaceVolumeRatio +
##     shape_VoxelVolume + firstorder_Energy + firstorder_Entropy +
##     firstorder_Kurtosis + firstorder_Maximum + firstorder_Mean +
##     firstorder_MeanAbsoluteDeviation + firstorder_Median + firstorder_Minimum +
##     firstorder_Range + firstorder_RootMeanSquared + firstorder_Skewness +
##     firstorder_StandardDeviation + firstorder_Uniformity + firstorder_Variance +
##     glcm_Autocorrelation + glcm_ClusterProminence + glcm_ClusterShade +
##     glcm_ClusterTendency + glcm_Contrast + glcm_Correlation +
##     glcm_DifferenceEntropy + glcm_DifferenceAverage + glcm_JointEnergy +
##     glcm_JointEntropy + glcm_Id + glcm_Idm + glcm_Imc1 + glcm_Imc2 +
##     glcm_Idmn + glcm_Idn + glcm_InverseVariance + glcm_MaximumProbability +
##     glcm_SumAverage + glcm_SumEntropy + glrlm_ShortRunEmphasis +
##     glrlm_LongRunEmphasis + glrlm_GrayLevelNonUniformity + glrlm_RunLengthNonUniformity +
##     glrlm_RunPercentage + glrlm_LowGrayLevelRunEmphasis + glrlm_HighGrayLevelRunEmphasis +
##     glrlm_ShortRunLowGrayLevelEmphasis + glrlm_ShortRunHighGrayLevelEmphasis +
##     glrlm_LongRunLowGrayLevelEmphasis + glrlm_LongRunHighGrayLevelEmphasis
```

```
set.seed(1234)
gbm_model = gbm(classification_formula,
                data =  train_df_classificaiton,
                distribution='bernoulli',
                n.trees=500,
                interaction.depth=3,
                shrinkage=0.01,
                bag.fraction=0.5,
                keep.data=FALSE,
                cv.folds=5)

nTrees <- gbm.perf(gbm_model)
```



```
validate_predictions <- predict(gbm_model,
                                newdata = validate_df_classification[,feature_names],
                                type="response", n.trees = nTrees)
```

```
require(pROC)
roc(response=validate_df_classification$ReachedEvent, predictor=validate_predictions)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

##
## Call:
## roc.default(response = validate_df_classification$ReachedEvent,     predictor = validate_predictions
##
## Data: validate_predictions in 86 controls (validate_df_classification$ReachedEvent 0) < 63 cases (va
## Area under the curve: 0.7213
```

Now that both models can predict the same period and the probability of reaching the event, we average them together and see how they help each other (straight 50/50 here which may not be the best mix)

7.0.1 GBM Classifiaction Prediction

```
gbm_predictions_test <- predict(gbm_model, imputed_test[,feature_names])
```

```
## Using 190 trees...
```

```
PatientID <- imputed_test$PatientID

pred <- data.frame(
  PatientID = PatientID,
  Event = gbm_predictions_test
)

submission_gbm <- output_test %>%
  select(PatientID, SurvivalTime) %>%
  left_join(pred, by = "PatientID")

fwrite(submission_gbm, "submission_gbm.csv")
```

7.1 Blend both models (Survival and Classification) together

```
roc(predictor = (validate_predictions + (1 - suvival_predictions$survival[,which(suvival_predictions$uni
          response = validate_df_classification$ReachedEvent)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
##
## Call:
## roc.default(response = validate_df_classification$ReachedEvent,     predictor = (validate_prediction
##
## Data: (validate_predictions + (1 - suvival_predictions$survival[, which(suvival_predictions$unique.d
## Area under the curve: 0.7239
```