Report draft

Linh Tran

3/29/2021

```
# check the first few rows
head(stroke_data)
```

```
## # A tibble: 6 x 12
##      id gender   age hypertension heart_disease ever_married work_type
##   <dbl> <chr>  <dbl>        <dbl>         <dbl> <chr>        <chr>
## 1  9046 Male      67            0             1 Yes          Private
## 2 51676 Female    61            0             0 Yes          Self-emp~
## 3 31112 Male      80            0             1 Yes          Private
## 4 60182 Female    49            0             0 Yes          Private
## 5  1665 Female    79            1             0 Yes          Self-emp~
## 6 56669 Male      81            0             0 Yes          Private
## # ... with 5 more variables: Residence_type <chr>, avg_glucose_level <dbl>,
## #   bmi <chr>, smoking_status <chr>, stroke <dbl>
```

```
# summary of the data
summary(stroke_data)
```

```
##        id           gender               age         hypertension
##  Min.   :   67   Length:5110        Min.   : 0.08   Min.   :0.00000
##  1st Qu.:17741   Class :character   1st Qu.:25.00   1st Qu.:0.00000
##  Median :36932   Mode  :character   Median :45.00   Median :0.00000
##  Mean   :36518                      Mean   :43.23   Mean   :0.09746
##  3rd Qu.:54682                      3rd Qu.:61.00   3rd Qu.:0.00000
##  Max.   :72940                      Max.   :82.00   Max.   :1.00000
##  heart_disease     ever_married        work_type         Residence_type
##  Min.   :0.00000   Length:5110        Length:5110        Length:5110
##  1st Qu.:0.00000   Class :character   Class :character   Class :character
##  Median :0.00000   Mode  :character   Mode  :character   Mode  :character
##  Mean   :0.05401
##  3rd Qu.:0.00000
##  Max.   :1.00000
##  avg_glucose_level     bmi            smoking_status         stroke
##  Min.   : 55.12    Length:5110        Length:5110        Min.   :0.00000
##  1st Qu.: 77.25    Class :character   Class :character   1st Qu.:0.00000
##  Median : 91.89    Mode  :character   Mode  :character   Median :0.00000
##  Mean   :106.15                                          Mean   :0.04873
##  3rd Qu.:114.09                                          3rd Qu.:0.00000
##  Max.   :271.74                                          Max.   :1.00000
```

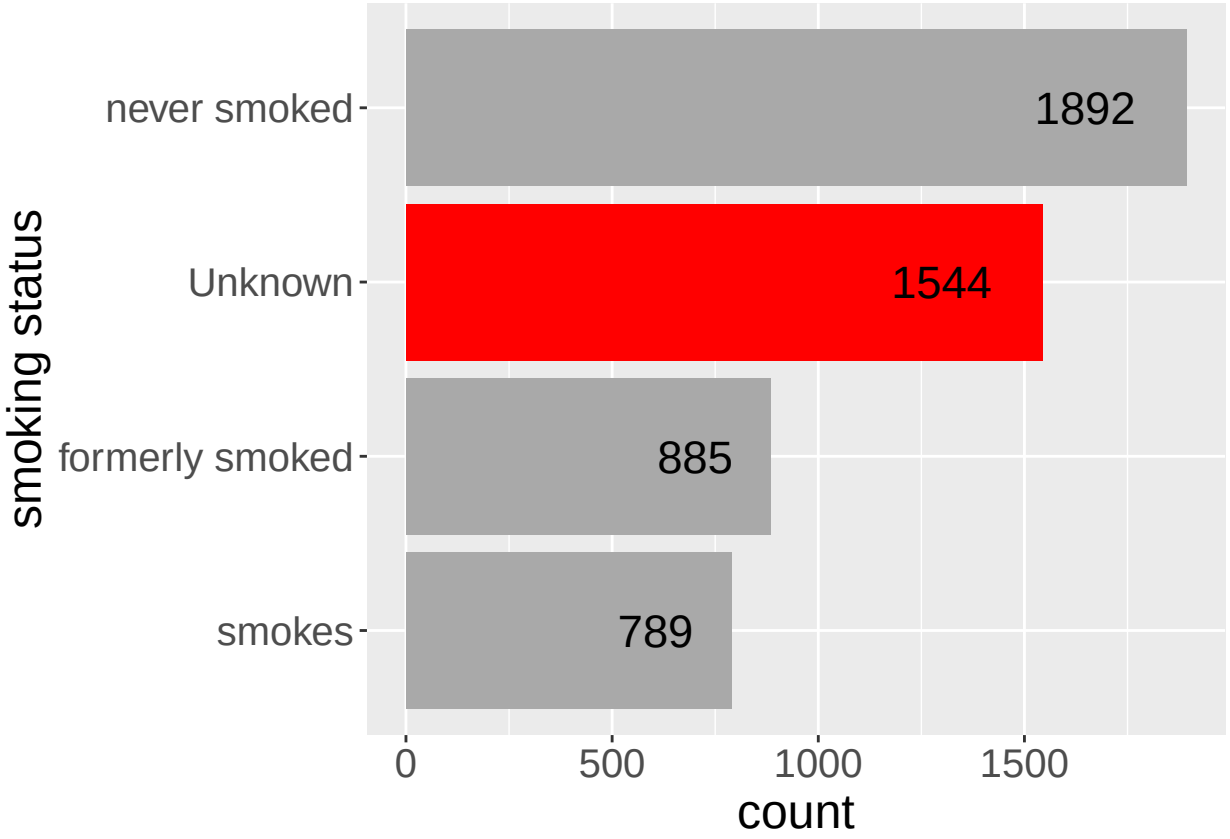
```
# how many "N/A" values are in my dataset per column?
miss_scan_count(data = stroke_data, search = list("N/A", "Unknown"))
```

```
## # A tibble: 12 x 2
```

```
##    Variable              n
##    <chr>             <int>
##  1 id                    0
##  2 gender                0
##  3 age                   0
##  4 hypertension          0
##  5 heart_disease         0
##  6 ever_married          0
##  7 work_type             0
##  8 Residence_type        0
##  9 avg_glucose_level     0
## 10 bmi                 201
## 11 smoking_status     1544
## 12 stroke                0
```

```
fig(15, 8)

stroke_data %>%
group_by(smoking_status) %>%
summarise(count = length(smoking_status)) %>%
mutate(smoking_status = factor(smoking_status)) %>%
ggplot(aes(x = fct_reorder(smoking_status, count), y = count, fill = factor(ifelse(smoking_status=="Unk
geom_col() +
geom_text(aes(label = count, x = smoking_status, y = count), size = 6, hjust = 1.5) +
coord_flip() +
scale_fill_manual(values = c("Unknown" = "red", "Known" = "darkgrey")) +
labs(x = "smoking status") +
theme(legend.position = "none") +
theme_bigfont
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
stroke_data_clean <- replace_with_na(data = stroke_data, replace = list(bmi = c("N/A"), smoking_status =
    # change bmi to numeric
    mutate(bmi = as.numeric(bmi))

# check
summary(stroke_data_clean)
```

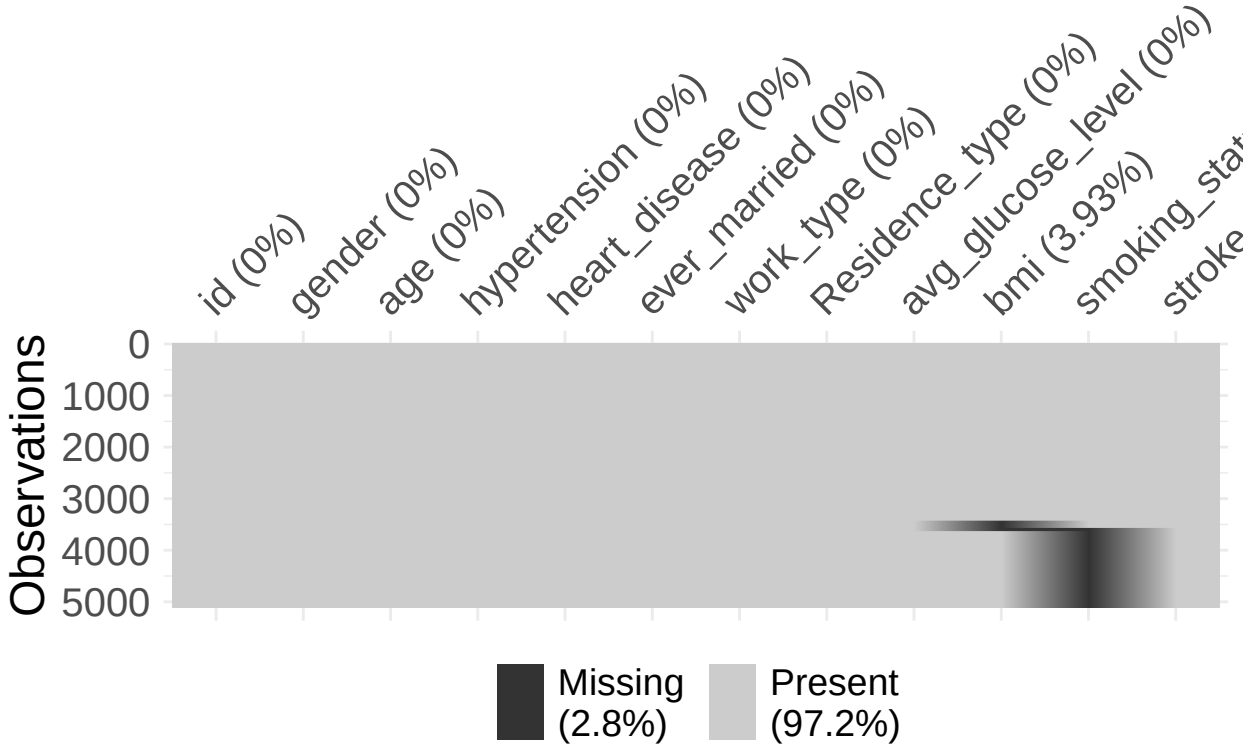
```
##        id           gender               age         hypertension
##  Min.   :   67   Length:5110        Min.   : 0.08   Min.   :0.00000
##  1st Qu.:17741   Class :character   1st Qu.:25.00   1st Qu.:0.00000
##  Median :36932   Mode  :character   Median :45.00   Median :0.00000
##  Mean   :36518                      Mean   :43.23   Mean   :0.09746
##  3rd Qu.:54682                      3rd Qu.:61.00   3rd Qu.:0.00000
##  Max.   :72940                      Max.   :82.00   Max.   :1.00000
##
##  heart_disease     ever_married        work_type         Residence_type
##  Min.   :0.00000   Length:5110        Length:5110        Length:5110
##  1st Qu.:0.00000   Class :character   Class :character   Class :character
##  Median :0.00000   Mode  :character   Mode  :character   Mode  :character
##  Mean   :0.05401
##  3rd Qu.:0.00000
##  Max.   :1.00000
##
##  avg_glucose_level      bmi         smoking_status         stroke
##  Min.   : 55.12    Min.   :10.30   Length:5110        Min.   :0.00000
##  1st Qu.: 77.25    1st Qu.:23.50   Class :character   1st Qu.:0.00000
##  Median : 91.89    Median :28.10   Mode  :character   Median :0.00000
##  Mean   :106.15    Mean   :28.89                      Mean   :0.04873
```

```
##  3rd Qu.:114.09    3rd Qu.:33.10                      3rd Qu.:0.00000
##  Max.   :271.74    Max.   :97.60                      Max.   :1.00000
##                    NA's   :201
```

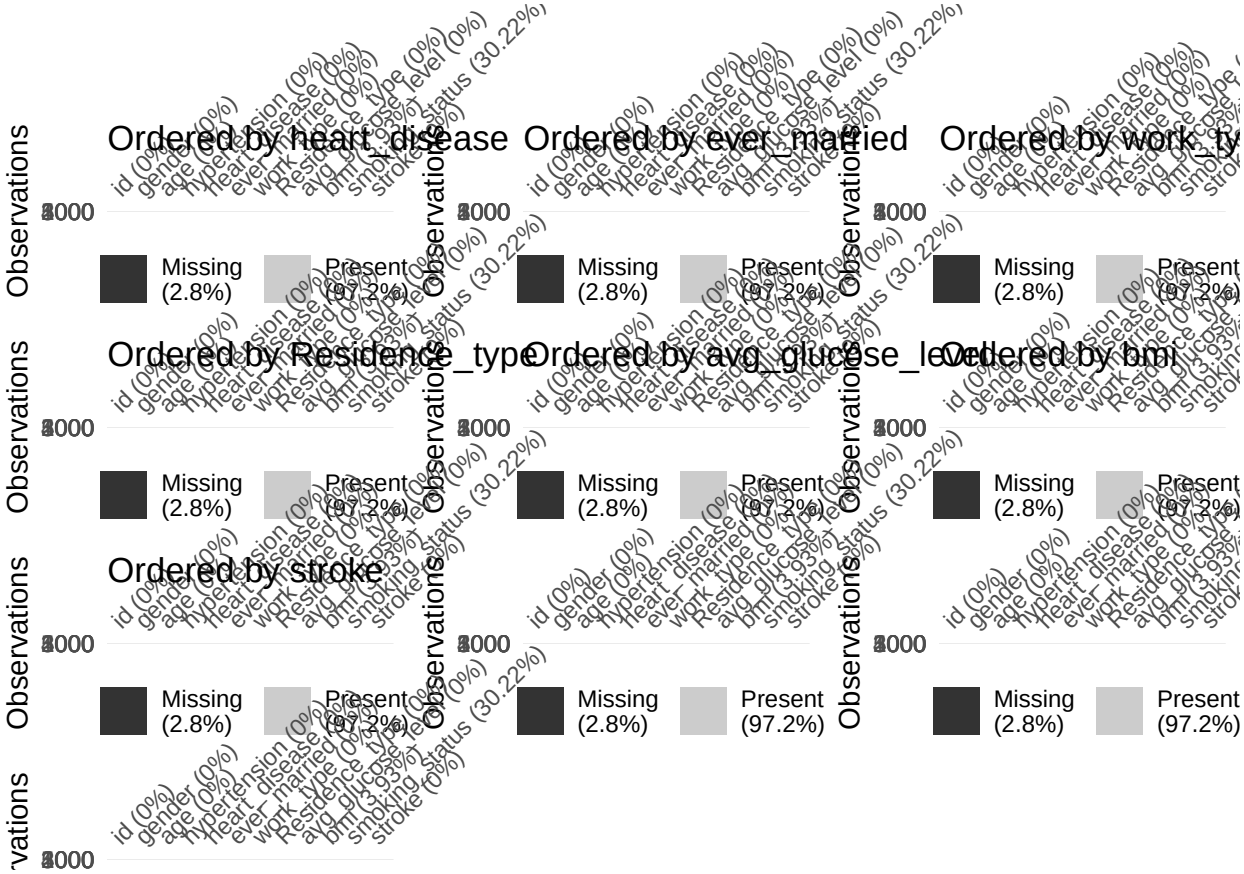
```
unique(stroke_data_clean$smoking_status)
```

```
## [1] "formerly smoked" "never smoked"    "smokes"          NA
```

```
fig(15, 8)

# visualize the missing values
vis_miss(stroke_data_clean, cluster = TRUE) +
theme_bigfont
```



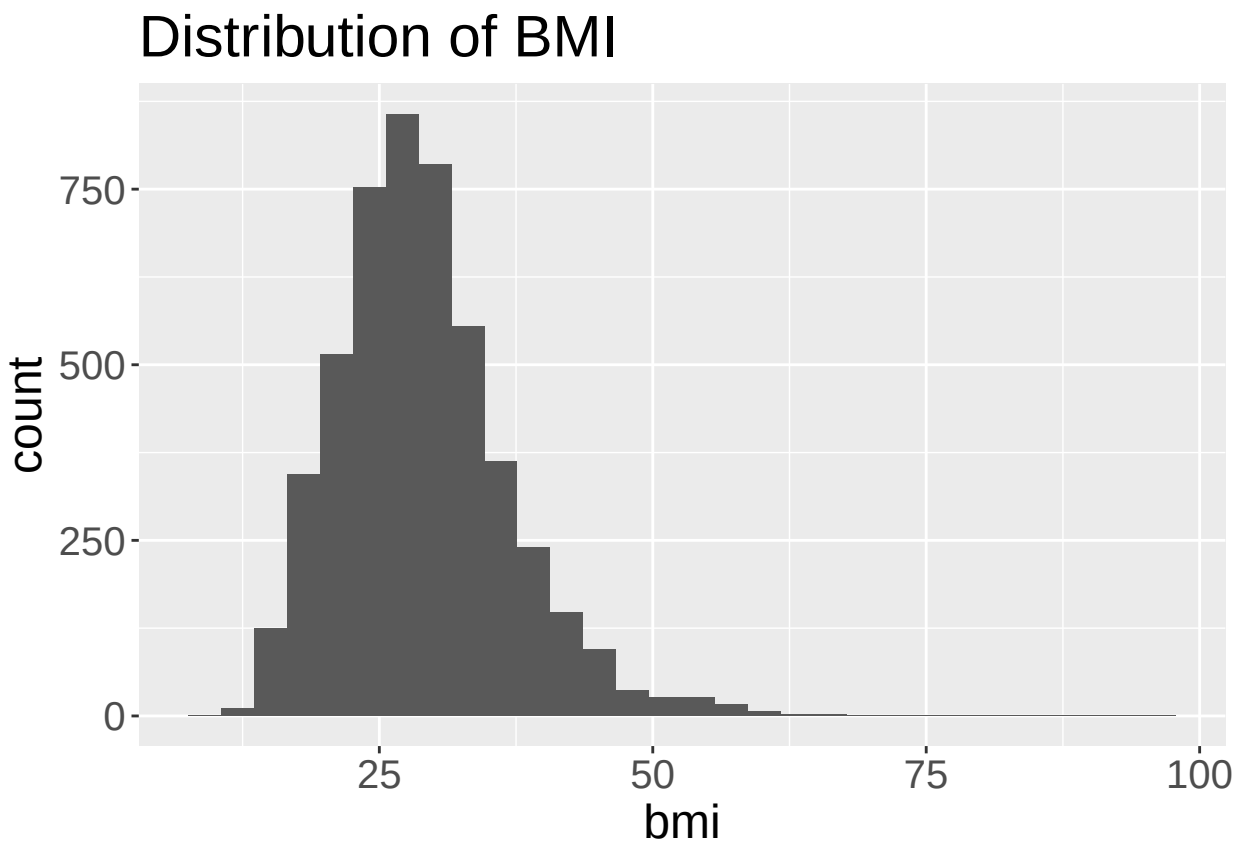

```
fig(20, 30)

# create vector of column names with
cols <- stroke_data_clean %>%
  dplyr::select(-id, -smoking_status) %>%
  names()

vis_plots_list <- list()

for (i in 1:length(cols)) {
    vis_plots_list[[i]] <- stroke_data_clean %>% arrange_at(cols[i]) %>% vis_miss() + labs(title = paste
}

n <- length(vis_plots_list)
nCol <- floor(sqrt(n))
do.call("grid.arrange", c(vis_plots_list, ncol = nCol))
```



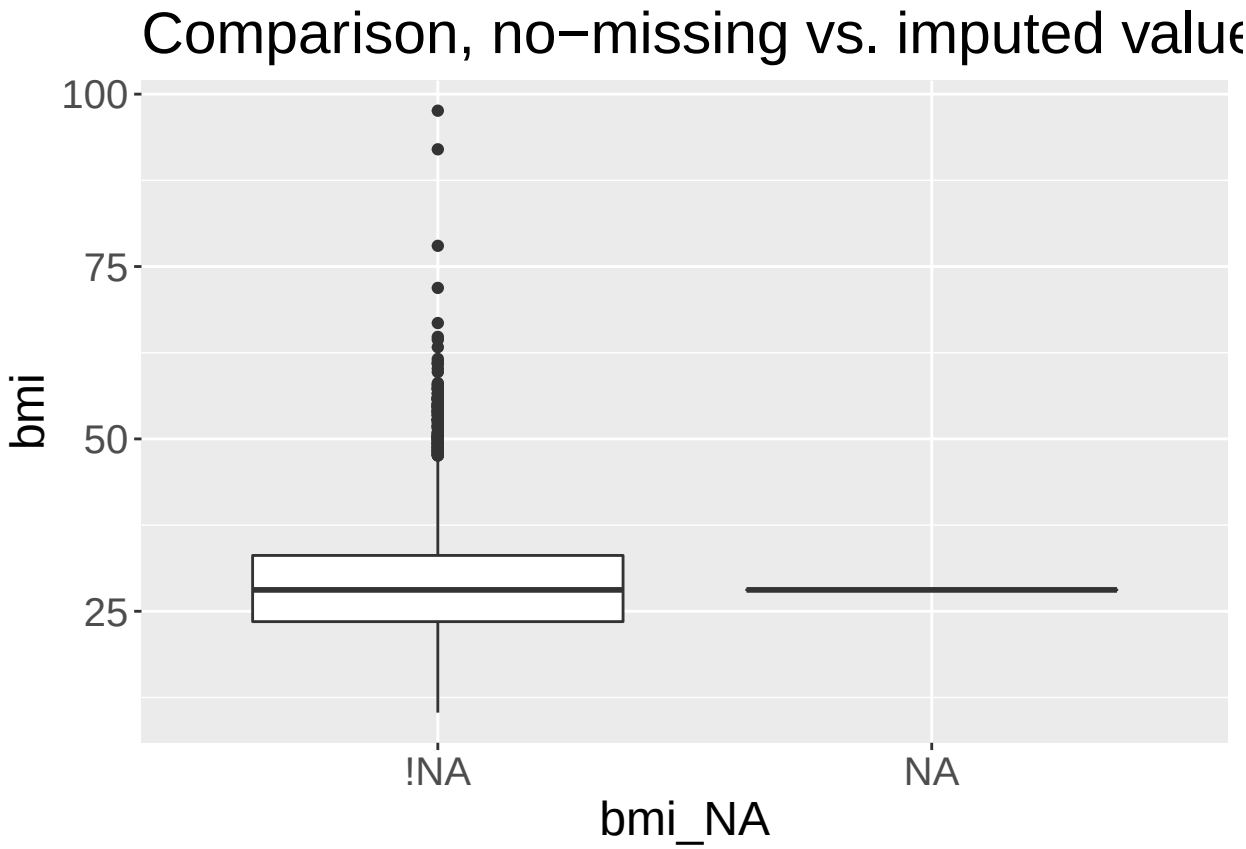

```
fig(10, 8)

# check distribution of bmi
ggplot(stroke_data_clean, aes(x = bmi)) +
geom_histogram() +
labs(title = "Distribution of BMI") +
theme_bigfont
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 201 rows containing non-finite values (stat_bin).
```

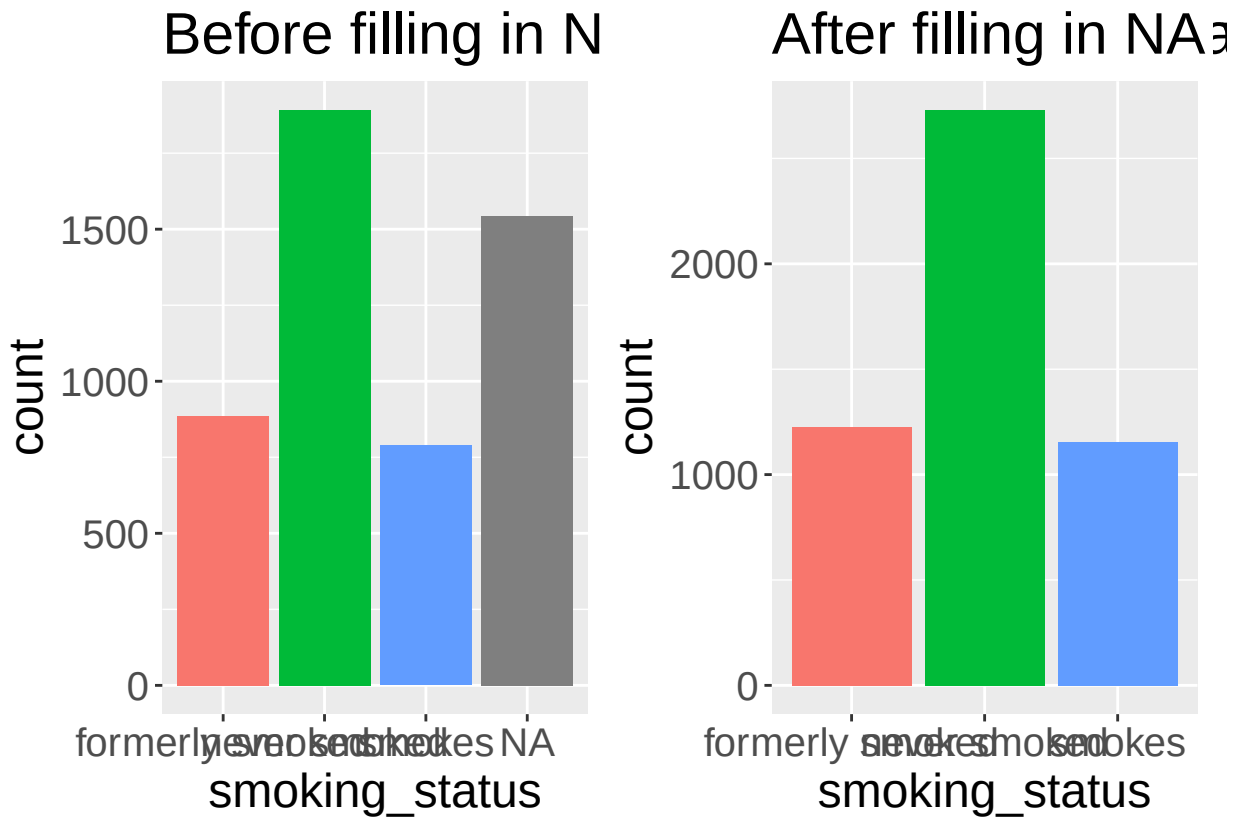



```
fig(10,8)

# impute median and bind shadow to evaluate imputation
stroke_data_imp <- bind_shadow(stroke_data_clean) %>%
impute_median_at(.vars = c("bmi")) %>%
add_label_shadow()

# Explore the median values in bmi in the imputed dataset
ggplot(stroke_data_imp,
       aes(x = bmi_NA, y = bmi)) +
geom_boxplot() +
labs(title = "Comparison, no-missing vs. imputed values for BMI") +
theme_bigfont
```




```
stroke_data_imp <- impute_median_at(stroke_data_clean, .vars = c("bmi"))
```

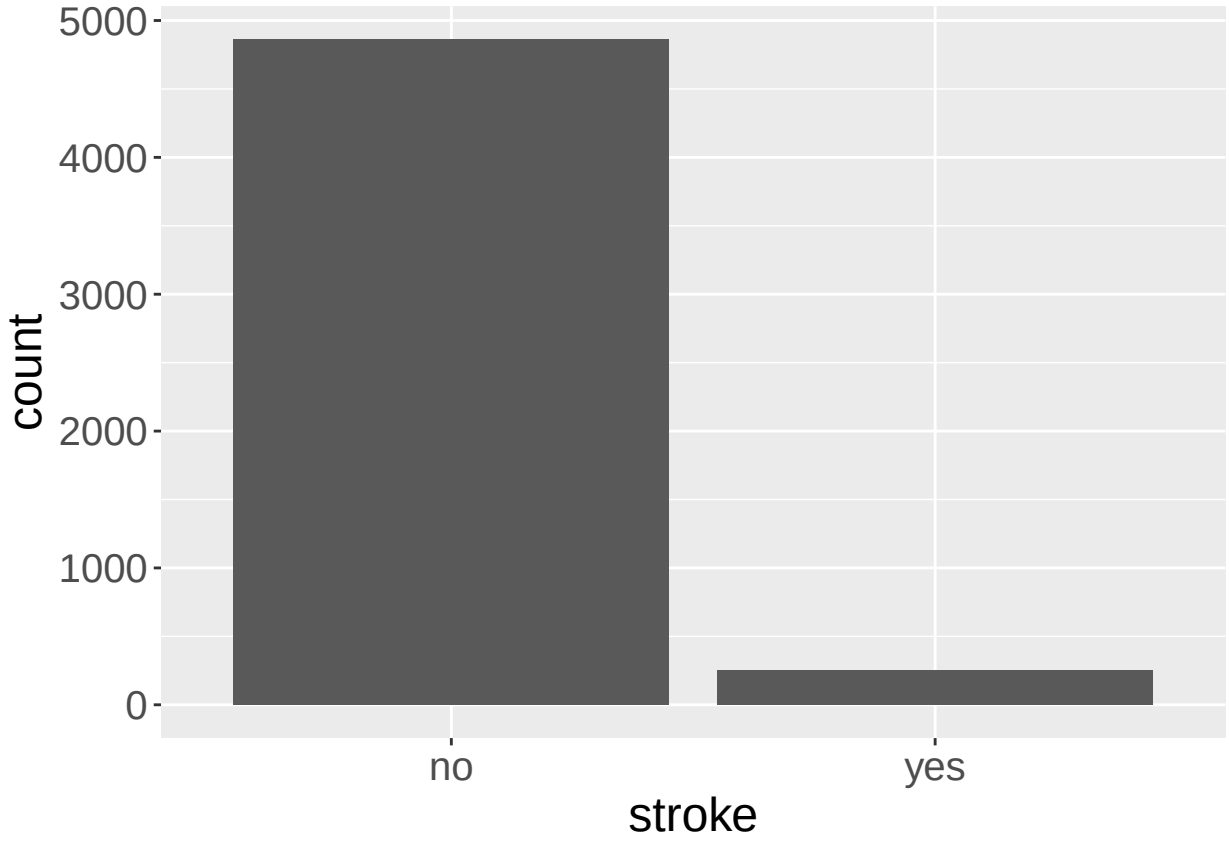
```
fig(16,8)

p1 <- ggplot(stroke_data_imp,
       aes(x = smoking_status, fill = smoking_status)) +
geom_bar() +
labs(title = "Before filling in NA values in smoking_status") +
theme(legend.position = "none") +
theme_bigfont

# fill imputation based on previous unique value in "smoking_status" column
after <- stroke_data_imp %>%
fill(smoking_status)
# mode imputation which leads to worse performance of models:
#mutate(across(c(smoking_status)), replace(., is.na(.), "never smoked"))

# Explore the median values in bmi in the imputed dataset
p2 <- ggplot(after,
       aes(x = smoking_status, fill = smoking_status)) +
geom_bar() +
labs(title = "After filling in NA values in smoking_status") +
theme(legend.position = "none") +
theme_bigfont

p1 + p2
```




```
stroke_data_imp2 <- stroke_data_imp %>%
  fill(smoking_status) %>%
  mutate(across(c(hypertension, heart_disease), factor),
      across(where(is.character), as.factor),
      across(where(is.factor), as.numeric),
      stroke = factor(ifelse(stroke == 0, "no", "yes")))
```

```
stroke_data_imp2 <- stroke_data_imp2 %>%
mutate(bmi = case_when(bmi < 18.5 ~ "underweight",
                       bmi >= 18.5 & bmi < 25 ~ "normal weight",
                       bmi >= 25 & bmi < 30 ~ "overweight",
                       bmi >= 30 ~ "obese"),
      bmi = factor(bmi, levels = c("underweight", "normal weight", "overweight", "obese"), order = TRUE)
```

```
fig(10, 8)

# plot prop of people who had a stroke
stroke_data_imp2 %>%
dplyr::select(stroke) %>%
ggplot(aes(x = stroke)) +
geom_bar() +
theme_bigfont
```




```
# count how many people had a stroke and the prop
stroke_data_imp2 %>%
group_by(stroke) %>%
summarize(n = n()) %>%
mutate(prop = round(n / sum(n), 2))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 3
##   stroke     n  prop
##   <fct>  <int> <dbl>
## 1 no      4861  0.95
## 2 yes      249  0.05
```

Because we have imbalanced data, I decided to use oversampling method to increase the number of 'stroke' instances to balance the distribution of classes.

```
imbalanceRatio(as.data.frame(stroke_data_imp2), classAttr = "stroke")
```

```
## [1] 0.05122403
```

```
stroke_test <- stroke_data_imp2 %>%
mutate(
    stroke = as.character(stroke),
    across(where(is.factor), as.numeric),
    stroke = factor(stroke)
)

stroke_oversampled <- oversample(as.data.frame(stroke_test), classAttr = "stroke", ratio = 1, method =
```

```r
head(stroke_oversampled)
```

```
##      id gender age hypertension heart_disease ever_married work_type
## 1  9046      2  67            1             2            2         4
## 2 51676      1  61            1             1            2         5
## 3 31112      2  80            1             2            2         4
## 4 60182      1  49            1             1            2         4
## 5  1665      1  79            2             1            2         5
## 6 56669      2  81            1             1            2         4
##   Residence_type avg_glucose_level bmi smoking_status stroke
## 1              2            228.69   4              1    yes
## 2              1            202.21   3              2    yes
## 3              1            105.92   4              2    yes
## 4              2            171.23   4              3    yes
## 5              1            174.12   2              2    yes
## 6              2            186.21   3              1    yes
```

```r
stroke_oversampled %>%
  group_by(stroke) %>%
  summarize(n = n()) %>%
  mutate(prop = round(n / sum(n), 2))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 3
##   stroke     n  prop
##   <fct>  <int> <dbl>
## 1 no      4861   0.5
## 2 yes     4861   0.5
```

```r
stroke_oversampled <-
  stroke_oversampled %>%
  dplyr::select(-id)
```

Train/Test split for oversampled data

```r
set.seed(2021)

trRow = createDataPartition(y = stroke_oversampled$stroke, p = 0.7, list = F)
train_oversamp = stroke_oversampled[trRow, ]
test_oversamp = stroke_oversampled[-trRow, ]
```

Train/Test split for original data

```r
set.seed(2021)

trRow = createDataPartition(y = stroke_test$stroke, p = 0.7, list = F)
train_original = stroke_test[trRow, ]
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
test_original = stroke_test[-trRow, ]

test_original_no_stroke = test_original %>%
  dplyr::select(-id, - stroke)
```

Model building

GLMnet

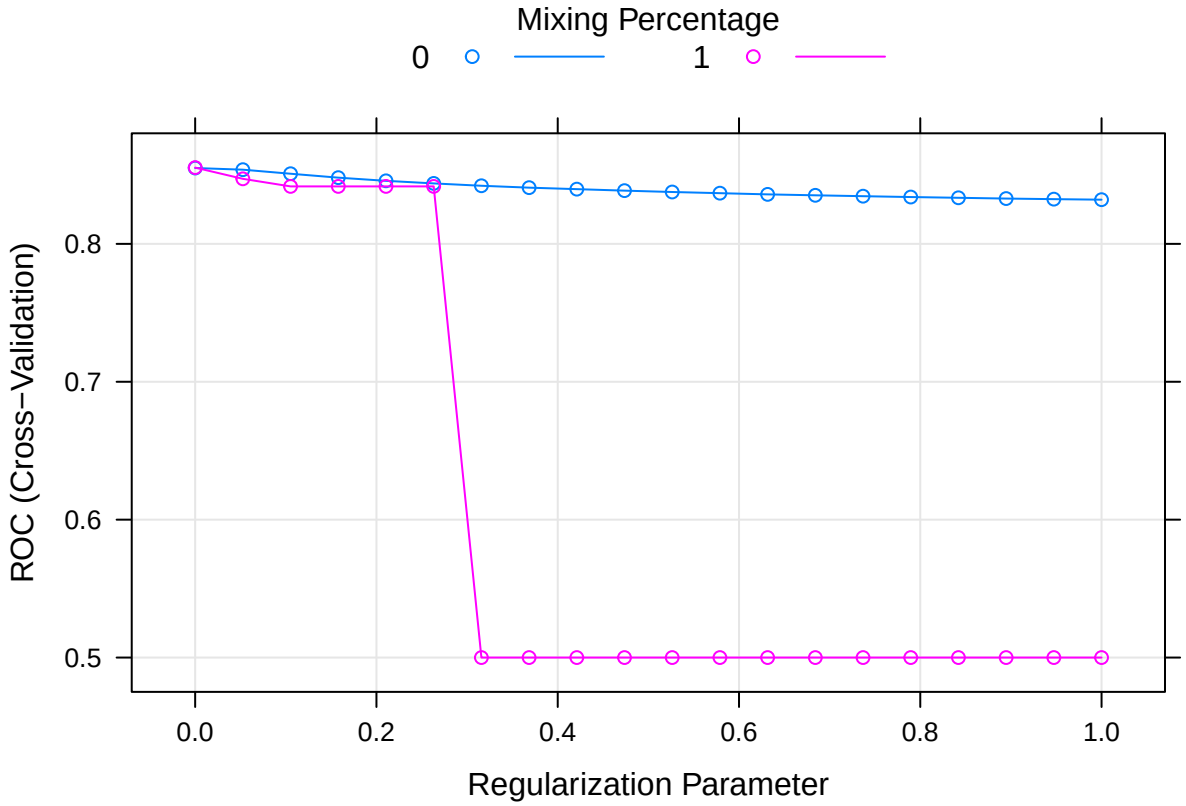
```
# custom train control
myControl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = twoClassSummary,
  classProbs = TRUE,
  verboseIter = TRUE
)

myGrid <- expand.grid(
    alpha = c(0,1),
    lambda = seq(0.00001, 1, length = 20)
)

set.seed(42)
glmnet_model <- train(
    stroke ~ .,
    train_oversamp,
    method = "glmnet",
    tuneGrid = myGrid,
    trControl = myControl
)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.
```

```
## + Fold01: alpha=0, lambda=1
## - Fold01: alpha=0, lambda=1
## + Fold01: alpha=1, lambda=1
## - Fold01: alpha=1, lambda=1
## + Fold02: alpha=0, lambda=1
## - Fold02: alpha=0, lambda=1
## + Fold02: alpha=1, lambda=1
## - Fold02: alpha=1, lambda=1
## + Fold03: alpha=0, lambda=1
## - Fold03: alpha=0, lambda=1
## + Fold03: alpha=1, lambda=1
## - Fold03: alpha=1, lambda=1
## + Fold04: alpha=0, lambda=1
## - Fold04: alpha=0, lambda=1
## + Fold04: alpha=1, lambda=1
## - Fold04: alpha=1, lambda=1
## + Fold05: alpha=0, lambda=1
## - Fold05: alpha=0, lambda=1
## + Fold05: alpha=1, lambda=1
## - Fold05: alpha=1, lambda=1
## + Fold06: alpha=0, lambda=1
## - Fold06: alpha=0, lambda=1
```

```
## + Fold06: alpha=1, lambda=1
## - Fold06: alpha=1, lambda=1
## + Fold07: alpha=0, lambda=1
## - Fold07: alpha=0, lambda=1
## + Fold07: alpha=1, lambda=1
## - Fold07: alpha=1, lambda=1
## + Fold08: alpha=0, lambda=1
## - Fold08: alpha=0, lambda=1
## + Fold08: alpha=1, lambda=1
## - Fold08: alpha=1, lambda=1
## + Fold09: alpha=0, lambda=1
## - Fold09: alpha=0, lambda=1
## + Fold09: alpha=1, lambda=1
## - Fold09: alpha=1, lambda=1
## + Fold10: alpha=0, lambda=1
## - Fold10: alpha=0, lambda=1
## + Fold10: alpha=1, lambda=1
## - Fold10: alpha=1, lambda=1
## Aggregating results
## Selecting tuning parameters
## Fitting alpha = 1, lambda = 1e-05 on full training set
```

```
plot(glmnet_model)
```




```
max(glmnet_model[["results"]]$ROC)
```

```
## [1] 0.8553074
```

```
mm_test <- test_oversamp %>%
  dplyr::select(-stroke)
```

```r
glmnet_pred <- predict(glmnet_model, newdata = mm_test)

confusionMatrix(glmnet_pred, factor(test_oversamp[["stroke"]]), positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1090  214
##        yes  368 1244
##
##                Accuracy : 0.8004
##                  95% CI : (0.7854, 0.8148)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6008
##
##  Mcnemar's Test P-Value : 2.267e-10
##
##             Sensitivity : 0.8532
##             Specificity : 0.7476
##          Pos Pred Value : 0.7717
##          Neg Pred Value : 0.8359
##              Prevalence : 0.5000
##          Detection Rate : 0.4266
##    Detection Prevalence : 0.5528
##       Balanced Accuracy : 0.8004
##
##        'Positive' Class : yes
##
```

The best model using GLMnet is a ridge regression classifier. However, it has lower accuracy than the baseline model (0.78 vs 0.95), however it's recall (sensitivity) is of course larger than baseline (0.74) meaning it has at least some predictive power to classify true positives correctly.

Now redo for original data (before oversampling)

```r
glmnet_pred_original <- predict(glmnet_model, newdata = test_original_no_stroke)

confusionMatrix(glmnet_pred_original, factor(test_original[["stroke"]]), positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1090   22
##        yes  368   52
##
##                Accuracy : 0.7454
##                  95% CI : (0.7228, 0.7671)
##     No Information Rate : 0.9517
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1399
```

```
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.70270
##             Specificity : 0.74760
##          Pos Pred Value : 0.12381
##          Neg Pred Value : 0.98022
##              Prevalence : 0.04830
##          Detection Rate : 0.03394
##    Detection Prevalence : 0.27415
##       Balanced Accuracy : 0.72515
##
##        'Positive' Class : yes
##
```

Using original data (no oversampling)

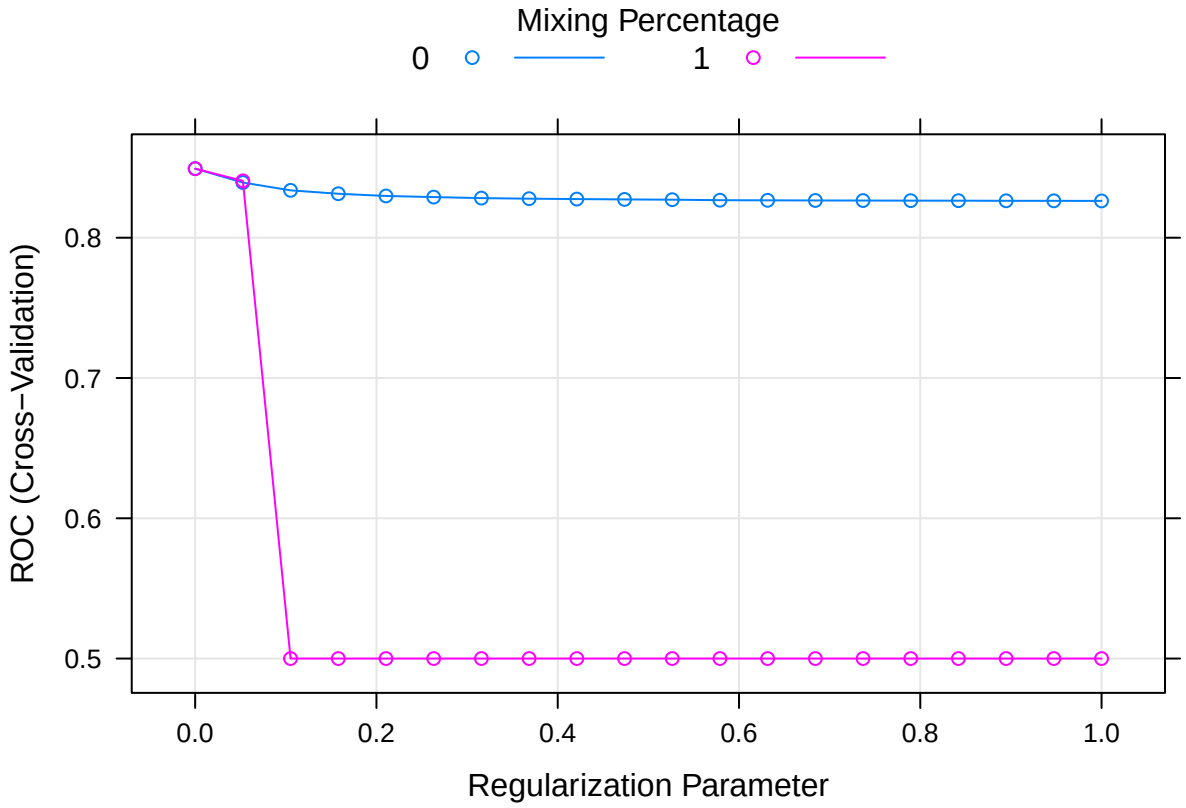
```
# custom train control
myControl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = twoClassSummary,
  classProbs = TRUE,
  verboseIter = TRUE
)

myGrid <- expand.grid(
    alpha = c(0,1),
    lambda = seq(0.00001, 1, length = 20)
)

set.seed(2021)
glmnet_model <- train(
    stroke ~ .,
    train_original,
    method = "glmnet",
    tuneGrid = myGrid,
    trControl = myControl
)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.
```

```
## + Fold01: alpha=0, lambda=1
## - Fold01: alpha=0, lambda=1
## + Fold01: alpha=1, lambda=1
## - Fold01: alpha=1, lambda=1
## + Fold02: alpha=0, lambda=1
## - Fold02: alpha=0, lambda=1
## + Fold02: alpha=1, lambda=1
## - Fold02: alpha=1, lambda=1
## + Fold03: alpha=0, lambda=1
## - Fold03: alpha=0, lambda=1
## + Fold03: alpha=1, lambda=1
## - Fold03: alpha=1, lambda=1
## + Fold04: alpha=0, lambda=1
```

```
## - Fold04: alpha=0, lambda=1
## + Fold04: alpha=1, lambda=1
## - Fold04: alpha=1, lambda=1
## + Fold05: alpha=0, lambda=1
## - Fold05: alpha=0, lambda=1
## + Fold05: alpha=1, lambda=1
## - Fold05: alpha=1, lambda=1
## + Fold06: alpha=0, lambda=1
## - Fold06: alpha=0, lambda=1
## + Fold06: alpha=1, lambda=1
## - Fold06: alpha=1, lambda=1
## + Fold07: alpha=0, lambda=1
## - Fold07: alpha=0, lambda=1
## + Fold07: alpha=1, lambda=1
## - Fold07: alpha=1, lambda=1
## + Fold08: alpha=0, lambda=1
## - Fold08: alpha=0, lambda=1
## + Fold08: alpha=1, lambda=1
## - Fold08: alpha=1, lambda=1
## + Fold09: alpha=0, lambda=1
## - Fold09: alpha=0, lambda=1
## + Fold09: alpha=1, lambda=1
## - Fold09: alpha=1, lambda=1
## + Fold10: alpha=0, lambda=1
## - Fold10: alpha=0, lambda=1
## + Fold10: alpha=1, lambda=1
## - Fold10: alpha=1, lambda=1
## Aggregating results
## Selecting tuning parameters
## Fitting alpha = 0, lambda = 1e-05 on full training set
```

```
plot(glmnet_model)
```




```
max(glmnet_model[["results"]]$ROC)
```

```
## [1] 0.8492836
```

```
mm_test <- test_original %>%
  dplyr::select(-stroke)

glmnet_pred <- predict(glmnet_model, newdata = mm_test)

confusionMatrix(glmnet_pred, factor(test_original[["stroke"]]), positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1458   74
##        yes    0    0
##
##                Accuracy : 0.9517
##                  95% CI : (0.9397, 0.9619)
##     No Information Rate : 0.9517
##     P-Value [Acc > NIR] : 0.5309
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
```

```
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.9517
##              Prevalence : 0.0483
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : yes
##
```

The values for accuracy improve a little for the oversampled data (0.7898 vs 0.7448). The sensitivity also improved for the oversampled data (0.8320 vs 0.68919).

Random Forest

Original data

```
rfGrid <- data.frame(
  .mtry = c(2,3,5,6),
  .splitrule = "gini",
  .min.node.size = 5
)

rfControl <- trainControl(
    method = "oob",
    number = 5,
    verboseIter = TRUE
)

rf_model_original <- train(
    stroke ~ .,
    train_original,
    method = "ranger",
    tuneLength = 3,
    tuneGrid = rfGrid,
    trControl = rfControl
)
```

```
## + : mtry=2, splitrule=gini, min.node.size=5
## - : mtry=2, splitrule=gini, min.node.size=5
## + : mtry=3, splitrule=gini, min.node.size=5
## - : mtry=3, splitrule=gini, min.node.size=5
## + : mtry=5, splitrule=gini, min.node.size=5
## - : mtry=5, splitrule=gini, min.node.size=5
## + : mtry=6, splitrule=gini, min.node.size=5
## - : mtry=6, splitrule=gini, min.node.size=5
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 2, splitrule = gini, min.node.size = 5 on full training set
```

```
rf_model_original
```

```
## Random Forest
##
## 3578 samples
##   11 predictor
```

```
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9505310  -0.001106534
##   3     0.9496926   0.007253995
##   5     0.9494131  -0.003253201
##   6     0.9499721   0.017657146
##
## Tuning parameter 'splitrule' was held constant at a value of gini
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 2, splitrule = gini
##  and min.node.size = 5.
```

```
mm_test_original = test_original %>%
  dplyr::select(-stroke)

rf_pred_original <- predict(rf_model_original, newdata = mm_test_original)

confusionMatrix(rf_pred_original, factor(test_original[["stroke"]]), positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1458   74
##        yes    0    0
##
##                Accuracy : 0.9517
##                  95% CI : (0.9397, 0.9619)
##     No Information Rate : 0.9517
##     P-Value [Acc > NIR] : 0.5309
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.9517
##              Prevalence : 0.0483
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : yes
##
```

Oversampled data

```
rfGrid <- data.frame(
  .mtry = c(2,3,5,6),
  .splitrule = "gini",
  .min.node.size = 5
)

rfControl <- trainControl(
    method = "oob",
    number = 5,
    verboseIter = TRUE
)

rf_model_oversamp <- train(
    stroke ~ .,
    train_oversamp,
    method = "ranger",
    tuneLength = 3,
    tuneGrid = rfGrid,
    trControl = rfControl
)
```

```
## + : mtry=2, splitrule=gini, min.node.size=5
## - : mtry=2, splitrule=gini, min.node.size=5
## + : mtry=3, splitrule=gini, min.node.size=5
## - : mtry=3, splitrule=gini, min.node.size=5
## + : mtry=5, splitrule=gini, min.node.size=5
## - : mtry=5, splitrule=gini, min.node.size=5
## + : mtry=6, splitrule=gini, min.node.size=5
## - : mtry=6, splitrule=gini, min.node.size=5
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 5, splitrule = gini, min.node.size = 5 on full training set
```

```
rf_model_oversamp
```

```
## Random Forest
##
## 6806 samples
##   10 predictor
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9626800  0.9253600
##   3     0.9709080  0.9418160
##   5     0.9717896  0.9435792
##   6     0.9710550  0.9421099
##
## Tuning parameter 'splitrule' was held constant at a value of gini
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 5, splitrule = gini
```

```
##  and min.node.size = 5.
```

```
mm_test_oversamp = test_oversamp %>%
  dplyr::select(-stroke)

rf_pred_oversamp <- predict(rf_model_oversamp, newdata = mm_test_oversamp)

confusionMatrix(rf_pred_oversamp, factor(test_oversamp[["stroke"]]), positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1428   65
##        yes   30 1393
##
##                Accuracy : 0.9674
##                  95% CI : (0.9603, 0.9736)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9348
##
##  Mcnemar's Test P-Value : 0.0004861
##
##             Sensitivity : 0.9554
##             Specificity : 0.9794
##          Pos Pred Value : 0.9789
##          Neg Pred Value : 0.9565
##              Prevalence : 0.5000
##          Detection Rate : 0.4777
##    Detection Prevalence : 0.4880
##       Balanced Accuracy : 0.9674
##
##        'Positive' Class : yes
##
```

Looks like the random forest model is great at classifying true negative cases, but performs poorly on classifying true positive cases which is what we are interested in (we want to detect people with stroke, so we can be confident in telling a patient they are at risk of stroke when we supply his/her information to the model).