Practical ML in R - Coursera Assignment final

Paulo Pereira da Silva

2024-10-30

Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, the goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)

Objective

The goal of the project is to predict the manner in which they did the exercise. This is the **“classe”** variable in the training set.

Analysis

The first step of this exercise is to import the dataset and relevant packages.

```
df <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
validation_set <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
```

EDA

Next, some EDA is performed. First, the number of NAs is analyzed:

```
df %>% is.na() %>%
  colSums() %>%
  as.data.frame() %>%
  rename(missing_na = 1) %>%
  filter(missing_na > 0)
```

```
##                          missing_na
## max_roll_belt                 19216
## max_picth_belt                19216
## min_roll_belt                 19216
## min_pitch_belt                19216
## amplitude_roll_belt           19216
## amplitude_pitch_belt          19216
## var_total_accel_belt          19216
## avg_roll_belt                 19216
## stddev_roll_belt              19216
## var_roll_belt                 19216
## avg_pitch_belt                19216
## stddev_pitch_belt             19216
## var_pitch_belt                19216
## avg_yaw_belt                  19216
## stddev_yaw_belt               19216
## var_yaw_belt                  19216
## var_accel_arm                 19216
## avg_roll_arm                  19216
## stddev_roll_arm               19216
## var_roll_arm                  19216
## avg_pitch_arm                 19216
## stddev_pitch_arm              19216
## var_pitch_arm                 19216
## avg_yaw_arm                   19216
## stddev_yaw_arm                19216
## var_yaw_arm                   19216
## max_roll_arm                  19216
## max_picth_arm                 19216
## max_yaw_arm                   19216
## min_roll_arm                  19216
## min_pitch_arm                 19216
## min_yaw_arm                   19216
## amplitude_roll_arm            19216
## amplitude_pitch_arm           19216
## amplitude_yaw_arm             19216
## max_roll_dumbbell             19216
## max_picth_dumbbell            19216
## min_roll_dumbbell             19216
## min_pitch_dumbbell            19216
## amplitude_roll_dumbbell       19216
## amplitude_pitch_dumbbell      19216
## var_accel_dumbbell            19216
## avg_roll_dumbbell             19216
## stddev_roll_dumbbell          19216
## var_roll_dumbbell             19216
## avg_pitch_dumbbell            19216
## stddev_pitch_dumbbell         19216
## var_pitch_dumbbell            19216
## avg_yaw_dumbbell              19216
## stddev_yaw_dumbbell           19216
## var_yaw_dumbbell              19216
## max_roll_forearm              19216
## max_picth_forearm             19216
## min_roll_forearm              19216
## min_pitch_forearm             19216
## amplitude_roll_forearm        19216
## amplitude_pitch_forearm       19216
## var_accel_forearm             19216
## avg_roll_forearm              19216
## stddev_roll_forearm           19216
## var_roll_forearm              19216
## avg_pitch_forearm             19216
## stddev_pitch_forearm          19216
## var_pitch_forearm             19216
## avg_yaw_forearm               19216
## stddev_yaw_forearm            19216
## var_yaw_forearm               19216
```

There are lots of variables in the dataset with a significant number of NAs. Thus, I delete the missing values from numeric variables. Specifically, I delete all variables with more than 25% of missing values.

```
df %>%
  is.na() %>%
  {colSums(.)/NROW(.)} %>%
  Filter(function(x) x<0.25, .) %>%
  names() -> nonMissingVars

df %<>%
  select(all_of(nonMissingVars))
```

After cleaning missing values pertaining to numeric covariates, I explore some additional descriptive statistics using the skim function from skimr:

```r
#DataExplorer::create_report(training_set)
skimr::skim(df)
```

Data summary

Name	df
Number of rows	19622
Number of columns	93
_______________________	
Column type frequency:	
character	37
numeric	56
________________________	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
user_name	0	1	5	8	0	6	0
cvtd_timestamp	0	1	16	16	0	20	0
new_window	0	1	2	3	0	2	0
kurtosis_roll_belt	0	1	0	9	19216	397	0
kurtosis_picth_belt	0	1	0	9	19216	317	0
kurtosis_yaw_belt	0	1	0	7	19216	2	0
skewness_roll_belt	0	1	0	9	19216	395	0
skewness_roll_belt.1	0	1	0	9	19216	338	0
skewness_yaw_belt	0	1	0	7	19216	2	0
max_yaw_belt	0	1	0	7	19216	68	0
min_yaw_belt	0	1	0	7	19216	68	0
amplitude_yaw_belt	0	1	0	7	19216	4	0
kurtosis_roll_arm	0	1	0	8	19216	330	0
kurtosis_picth_arm	0	1	0	8	19216	328	0
kurtosis_yaw_arm	0	1	0	8	19216	395	0
skewness_roll_arm	0	1	0	8	19216	331	0
skewness_pitch_arm	0	1	0	8	19216	328	0
skewness_yaw_arm	0	1	0	8	19216	395	0
kurtosis_roll_dumbbell	0	1	0	7	19216	398	0
kurtosis_picth_dumbbell	0	1	0	7	19216	401	0
kurtosis_yaw_dumbbell	0	1	0	7	19216	2	0
skewness_roll_dumbbell	0	1	0	7	19216	401	0
skewness_pitch_dumbbell	0	1	0	7	19216	402	0
skewness_yaw_dumbbell	0	1	0	7	19216	2	0
max_yaw_dumbbell	0	1	0	7	19216	73	0
min_yaw_dumbbell	0	1	0	7	19216	73	0
amplitude_yaw_dumbbell	0	1	0	7	19216	3	0
kurtosis_roll_forearm	0	1	0	7	19216	322	0
kurtosis_picth_forearm	0	1	0	7	19216	323	0
kurtosis_yaw_forearm	0	1	0	7	19216	2	0
skewness_roll_forearm	0	1	0	7	19216	323	0
skewness_pitch_forearm	0	1	0	7	19216	319	0
skewness_yaw_forearm	0	1	0	7	19216	2	0
max_yaw_forearm	0	1	0	7	19216	45	0
min_yaw_forearm	0	1	0	7	19216	45	0

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
amplitude_yaw_forearm	0	1	0	7	19216	3	0
classe	0	1	1	1	0	5	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	
X	0	1	9811.50	5664.53	1.00000e+00	4906.25	9811.50	14716.75	1.96220
raw_timestamp_part_1	0	1	1322827119.27	204927.68	1.32249e+09	1322673099.00	1322832920.00	1323084264.00	1.32309
raw_timestamp_part_2	0	1	500656.14	288222.88	2.94000e+02	252912.25	496380.00	751890.75	9.98801
num_window	0	1	430.64	247.91	1.00000e+00	222.00	424.00	644.00	8.64000
roll_belt	0	1	64.41	62.75	-2.89000e+01	1.10	113.00	123.00	1.62000
pitch_belt	0	1	0.31	22.35	-5.58000e+01	1.76	5.28	14.90	6.03000
yaw_belt	0	1	-11.21	95.19	-1.80000e+02	-88.30	-13.00	12.90	1.79000
total_accel_belt	0	1	11.31	7.74	0.00000e+00	3.00	17.00	18.00	2.90000
gyros_belt_x	0	1	-0.01	0.21	-1.04000e+00	-0.03	0.03	0.11	2.22000
gyros_belt_y	0	1	0.04	0.08	-6.40000e-01	0.00	0.02	0.11	6.40000
gyros_belt_z	0	1	-0.13	0.24	-1.46000e+00	-0.20	-0.10	-0.02	1.62000
accel_belt_x	0	1	-5.59	29.64	-1.20000e+02	-21.00	-15.00	-5.00	8.50000
accel_belt_y	0	1	30.15	28.58	-6.90000e+01	3.00	35.00	61.00	1.64000
accel_belt_z	0	1	-72.59	100.45	-2.75000e+02	-162.00	-152.00	27.00	1.05000
magnet_belt_x	0	1	55.60	64.18	-5.20000e+01	9.00	35.00	59.00	4.85000
magnet_belt_y	0	1	593.68	35.68	3.54000e+02	581.00	601.00	610.00	6.73000
magnet_belt_z	0	1	-345.48	65.21	-6.23000e+02	-375.00	-320.00	-306.00	2.93000
roll_arm	0	1	17.83	72.74	-1.80000e+02	-31.78	0.00	77.30	1.80000
pitch_arm	0	1	-4.61	30.68	-8.88000e+01	-25.90	0.00	11.20	8.85000
yaw_arm	0	1	-0.62	71.36	-1.80000e+02	-43.10	0.00	45.88	1.80000
total_accel_arm	0	1	25.51	10.52	1.00000e+00	17.00	27.00	33.00	6.60000
gyros_arm_x	0	1	0.04	1.99	-6.37000e+00	-1.33	0.08	1.57	4.87000
gyros_arm_y	0	1	-0.26	0.85	-3.44000e+00	-0.80	-0.24	0.14	2.84000
gyros_arm_z	0	1	0.27	0.55	-2.33000e+00	-0.07	0.23	0.72	3.02000
accel_arm_x	0	1	-60.24	182.04	-4.04000e+02	-242.00	-44.00	84.00	4.37000
accel_arm_y	0	1	32.60	109.87	-3.18000e+02	-54.00	14.00	139.00	3.08000
accel_arm_z	0	1	-71.25	134.65	-6.36000e+02	-143.00	-47.00	23.00	2.92000
magnet_arm_x	0	1	191.72	443.64	-5.84000e+02	-300.00	289.00	637.00	7.82000
magnet_arm_y	0	1	156.61	201.91	-3.92000e+02	-9.00	202.00	323.00	5.83000
magnet_arm_z	0	1	306.49	326.62	-5.97000e+02	131.25	444.00	545.00	6.94000
roll_dumbbell	0	1	23.84	69.93	-1.53710e+02	-18.49	48.17	67.61	1.53550
pitch_dumbbell	0	1	-10.78	36.99	-1.49590e+02	-40.89	-20.96	17.50	1.49400
yaw_dumbbell	0	1	1.67	82.52	-1.50870e+02	-77.64	-3.32	79.64	1.54950
total_accel_dumbbell	0	1	13.72	10.23	0.00000e+00	4.00	10.00	19.00	5.80000
gyros_dumbbell_x	0	1	0.16	1.51	-2.04000e+02	-0.03	0.13	0.35	2.22000
gyros_dumbbell_y	0	1	0.05	0.61	-2.10000e+00	-0.14	0.03	0.21	5.20000
gyros_dumbbell_z	0	1	-0.13	2.29	-2.38000e+00	-0.31	-0.13	0.03	3.17000
accel_dumbbell_x	0	1	-28.62	67.32	-4.19000e+02	-50.00	-8.00	11.00	2.35000
accel_dumbbell_y	0	1	52.63	80.75	-1.89000e+02	-8.00	41.50	111.00	3.15000
accel_dumbbell_z	0	1	-38.32	109.47	-3.34000e+02	-142.00	-1.00	38.00	3.18000
magnet_dumbbell_x	0	1	-328.48	339.72	-6.43000e+02	-535.00	-479.00	-304.00	5.92000
magnet_dumbbell_y	0	1	220.97	326.87	-3.60000e+03	231.00	311.00	390.00	6.33000
magnet_dumbbell_z	0	1	46.05	139.96	-2.62000e+02	-45.00	13.00	95.00	4.52000
roll_forearm	0	1	33.83	108.04	-1.80000e+02	-0.74	21.70	140.00	1.80000
pitch_forearm	0	1	10.71	28.15	-7.25000e+01	0.00	9.24	28.40	8.98000

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	
yaw_forearm	0	1	19.21	103.22	-1.80000e+02	-68.60	0.00	110.00	1.80000
total_accel_forearm	0	1	34.72	10.06	0.00000e+00	29.00	36.00	41.00	1.08000
gyros_forearm_x	0	1	0.16	0.65	-2.20000e+01	-0.22	0.05	0.56	3.97000
gyros_forearm_y	0	1	0.08	3.10	-7.02000e+00	-1.46	0.03	1.62	3.11000
gyros_forearm_z	0	1	0.15	1.75	-8.09000e+00	-0.18	0.08	0.49	2.31000
accel_forearm_x	0	1	-61.65	180.59	-4.98000e+02	-178.00	-57.00	76.00	4.77000
accel_forearm_y	0	1	163.66	200.13	-6.32000e+02	57.00	201.00	312.00	9.23000
accel_forearm_z	0	1	-55.29	138.40	-4.46000e+02	-182.00	-39.00	26.00	2.91000
magnet_forearm_x	0	1	-312.58	346.96	-1.28000e+03	-616.00	-378.00	-73.00	6.72000
magnet_forearm_y	0	1	380.12	509.37	-8.96000e+02	2.00	591.00	737.00	1.48000
magnet_forearm_z	0	1	393.61	369.27	-9.73000e+02	191.00	511.00	653.00	1.09000

As can be seen, there is also a number of empty character variable. NZV may help to identify them, and exclude them.

NZV

Now, I eliminate near zero variance variables.

```
nsv_obj <- nearZeroVar(df, saveMetrics = TRUE)
nsv_obj
```

```
##                         freqRatio percentUnique zeroVar   nzv
## X                        1.000000  100.00000000   FALSE FALSE
## user_name                1.100679    0.03057792   FALSE FALSE
## raw_timestamp_part_1     1.000000    4.26562022   FALSE FALSE
## raw_timestamp_part_2     1.000000   85.53154622   FALSE FALSE
## cvtd_timestamp           1.000668    0.10192641   FALSE FALSE
## new_window              47.330049    0.01019264   FALSE  TRUE
## num_window               1.000000    4.37264295   FALSE FALSE
## roll_belt                1.101904    6.77810621   FALSE FALSE
## pitch_belt               1.036082    9.37722964   FALSE FALSE
## yaw_belt                 1.058480    9.97349913   FALSE FALSE
## total_accel_belt         1.063160    0.14779329   FALSE FALSE
## kurtosis_roll_belt    1921.600000    2.02323922   FALSE  TRUE
## kurtosis_picth_belt    600.500000    1.61553358   FALSE  TRUE
## kurtosis_yaw_belt       47.330049    0.01019264   FALSE  TRUE
## skewness_roll_belt    2135.111111    2.01304658   FALSE  TRUE
## skewness_roll_belt.1   600.500000    1.72255631   FALSE  TRUE
## skewness_yaw_belt       47.330049    0.01019264   FALSE  TRUE
## max_yaw_belt           640.533333    0.34654979   FALSE  TRUE
## min_yaw_belt           640.533333    0.34654979   FALSE  TRUE
## amplitude_yaw_belt      50.041667    0.02038528   FALSE  TRUE
## gyros_belt_x             1.058651    0.71348486   FALSE FALSE
## gyros_belt_y             1.144000    0.35164611   FALSE FALSE
## gyros_belt_z             1.066214    0.86127816   FALSE FALSE
## accel_belt_x             1.055412    0.83579655   FALSE FALSE
## accel_belt_y             1.113725    0.72877383   FALSE FALSE
## accel_belt_z             1.078767    1.52379982   FALSE FALSE
## magnet_belt_x            1.090141    1.66649679   FALSE FALSE
## magnet_belt_y            1.099688    1.51870350   FALSE FALSE
## magnet_belt_z            1.006369    2.32901845   FALSE FALSE
## roll_arm                52.338462   13.52563449   FALSE FALSE
## pitch_arm               87.256410   15.73234125   FALSE FALSE
## yaw_arm                 33.029126   14.65701763   FALSE FALSE
## total_accel_arm          1.024526    0.33635715   FALSE FALSE
## gyros_arm_x              1.015504    3.27693405   FALSE FALSE
## gyros_arm_y              1.454369    1.91621649   FALSE FALSE
## gyros_arm_z              1.110687    1.26388747   FALSE FALSE
## accel_arm_x              1.017341    3.95984099   FALSE FALSE
## accel_arm_y              1.140187    2.73672409   FALSE FALSE
## accel_arm_z              1.128000    4.03628580   FALSE FALSE
## magnet_arm_x             1.000000    6.82397309   FALSE FALSE
## magnet_arm_y             1.056818    4.44399144   FALSE FALSE
## magnet_arm_z             1.036364    6.44684538   FALSE FALSE
## kurtosis_roll_arm      246.358974    1.68178575   FALSE  TRUE
## kurtosis_picth_arm     240.200000    1.67159311   FALSE  TRUE
## kurtosis_yaw_arm      1746.909091    2.01304658   FALSE  TRUE
## skewness_roll_arm      249.558442    1.68688207   FALSE  TRUE
## skewness_pitch_arm     240.200000    1.67159311   FALSE  TRUE
## skewness_yaw_arm      1746.909091    2.01304658   FALSE  TRUE
## roll_dumbbell            1.022388   84.20650290   FALSE FALSE
## pitch_dumbbell           2.277372   81.74498012   FALSE FALSE
## yaw_dumbbell             1.132231   83.48282540   FALSE FALSE
## kurtosis_roll_dumbbell  3843.200000    2.02833554   FALSE  TRUE
## kurtosis_picth_dumbbell 9608.000000    2.04362450   FALSE  TRUE
## kurtosis_yaw_dumbbell     47.330049    0.01019264   FALSE  TRUE
## skewness_roll_dumbbell  4804.000000    2.04362450   FALSE  TRUE
## skewness_pitch_dumbbell 9608.000000    2.04872082   FALSE  TRUE
## skewness_yaw_dumbbell     47.330049    0.01019264   FALSE  TRUE
## max_yaw_dumbbell         960.800000    0.37203139   FALSE  TRUE
## min_yaw_dumbbell         960.800000    0.37203139   FALSE  TRUE
## amplitude_yaw_dumbbell    47.920200    0.01528896   FALSE  TRUE
## total_accel_dumbbell       1.072634    0.21914178   FALSE FALSE
## gyros_dumbbell_x           1.003268    1.22821323   FALSE FALSE
## gyros_dumbbell_y           1.264957    1.41677709   FALSE FALSE
## gyros_dumbbell_z           1.060100    1.04984201   FALSE FALSE
## accel_dumbbell_x           1.018018    2.16593619   FALSE FALSE
## accel_dumbbell_y           1.053061    2.37488533   FALSE FALSE
## accel_dumbbell_z           1.133333    2.08949139   FALSE FALSE
## magnet_dumbbell_x          1.098266    5.74864948   FALSE FALSE
## magnet_dumbbell_y          1.197740    4.30129447   FALSE FALSE
## magnet_dumbbell_z          1.020833    3.44511263   FALSE FALSE
## roll_forearm              11.589286   11.08959331   FALSE FALSE
## pitch_forearm             65.983051   14.85577413   FALSE FALSE
## yaw_forearm               15.322835   10.14677403   FALSE FALSE
## kurtosis_roll_forearm    228.761905    1.64101519   FALSE  TRUE
## kurtosis_picth_forearm   226.070588    1.64611151   FALSE  TRUE
## kurtosis_yaw_forearm      47.330049    0.01019264   FALSE  TRUE
## skewness_roll_forearm    231.518072    1.64611151   FALSE  TRUE
## skewness_pitch_forearm   226.070588    1.62572623   FALSE  TRUE
## skewness_yaw_forearm      47.330049    0.01019264   FALSE  TRUE
## max_yaw_forearm          228.761905    0.22933442   FALSE  TRUE
## min_yaw_forearm          228.761905    0.22933442   FALSE  TRUE
## amplitude_yaw_forearm     59.677019    0.01528896   FALSE  TRUE
## total_accel_forearm        1.128928    0.35674243   FALSE FALSE
```

```
## gyros_forearm_x          1.059273    1.51870350   FALSE FALSE
## gyros_forearm_y          1.036554    3.77637346   FALSE FALSE
## gyros_forearm_z          1.122917    1.56457038   FALSE FALSE
## accel_forearm_x          1.126437    4.04647844   FALSE FALSE
## accel_forearm_y          1.059406    5.11160942   FALSE FALSE
## accel_forearm_z          1.006250    2.95586586   FALSE FALSE
## magnet_forearm_x         1.012346    7.76679238   FALSE FALSE
## magnet_forearm_y         1.246914    9.54031189   FALSE FALSE
## magnet_forearm_z         1.000000    8.57710733   FALSE FALSE
## classe                   1.469581    0.02548160   FALSE FALSE
```

```
df <- df[, !nsv_obj$nzv]
```

The target variable is defined as character. It will be re-classified as factor. The user name is classified as character. Next, one hot encoding is applied to this variable. Finally, cvtd_timestamp is coded as character, but it is actually a date variable.

```
sapply(df, class) |>
  as.data.frame() |>
  rename(type = 1 ) |>
  arrange(type)
```

```
##                             type
## user_name              character
## cvtd_timestamp         character
## classe                 character
## X                        integer
## raw_timestamp_part_1     integer
## raw_timestamp_part_2     integer
## num_window               integer
## total_accel_belt         integer
## accel_belt_x             integer
## accel_belt_y             integer
## accel_belt_z             integer
## magnet_belt_x            integer
## magnet_belt_y            integer
## magnet_belt_z            integer
## total_accel_arm          integer
## accel_arm_x              integer
## accel_arm_y              integer
## accel_arm_z              integer
## magnet_arm_x             integer
## magnet_arm_y             integer
## magnet_arm_z             integer
## total_accel_dumbbell     integer
## accel_dumbbell_x         integer
## accel_dumbbell_y         integer
## accel_dumbbell_z         integer
## magnet_dumbbell_x        integer
## magnet_dumbbell_y        integer
## total_accel_forearm      integer
## accel_forearm_x          integer
## accel_forearm_y          integer
## accel_forearm_z          integer
## magnet_forearm_x         integer
## roll_belt                numeric
## pitch_belt               numeric
## yaw_belt                 numeric
## gyros_belt_x             numeric
## gyros_belt_y             numeric
## gyros_belt_z             numeric
## roll_arm                 numeric
## pitch_arm                numeric
## yaw_arm                  numeric
## gyros_arm_x              numeric
## gyros_arm_y              numeric
## gyros_arm_z              numeric
## roll_dumbbell            numeric
## pitch_dumbbell           numeric
## yaw_dumbbell             numeric
## gyros_dumbbell_x         numeric
## gyros_dumbbell_y         numeric
## gyros_dumbbell_z         numeric
## magnet_dumbbell_z        numeric
## roll_forearm             numeric
## pitch_forearm            numeric
## yaw_forearm              numeric
## gyros_forearm_x          numeric
## gyros_forearm_y          numeric
## gyros_forearm_z          numeric
## magnet_forearm_y         numeric
## magnet_forearm_z         numeric
```

```
df %<>%
  mutate(classe = factor(classe),
         cvtd_timestamp = lubridate::parse_date_time(cvtd_timestamp, "%d/%m/%Y %H:%M")) %>%
  select(-X) #, raw_timestamp_part_1, raw_timestamp_part_2
```

```
classe <- df$classe
df <- dummyVars(classe ~ ., df) %>%
  predict(df) %>%
  as.data.frame() %>%
  add_column(classe = classe)
```

```
## Warning in model.frame.default(Terms, newdata, na.action = na.action, xlev =
## object$lvls): variable 'classe' is not a factor
```

After the application of on-hot-encoding, there are still no near zero variables.

```
nearZeroVar(df)
```

```
## integer(0)
```

Let's proceed with preliminary data pre-processing.

Correlations and multicolinearity

Next, let's gauge whether predictors are highly correlated.

```
correl <- findCorrelation(na.omit(df) %>% select(where(is.numeric)) %>%  cor(), cutoff = .75)
correl
```

```
##  [1] 20 11 19 14 46 13  6 18 21 12 45 47 48  9  7 49  4 44 31 33 22 35 58 41 55
## [26] 43 28
```

Almost one third of the dataset is redundant. These columns will be dropped from the dataset. We are left with 35 predictors.

```
correl_cols<-df %>%
  select(where(is.numeric)) %>%
  select(all_of(correl)) %>%
  colnames()

df %<>%
  select(-all_of(correl_cols))
```

Apparently, there are no linear combinations in the dataset. So, let's continue.

```
linear_comb <- findLinearCombos(df %>% select_if(is.numeric))
linear_comb
```

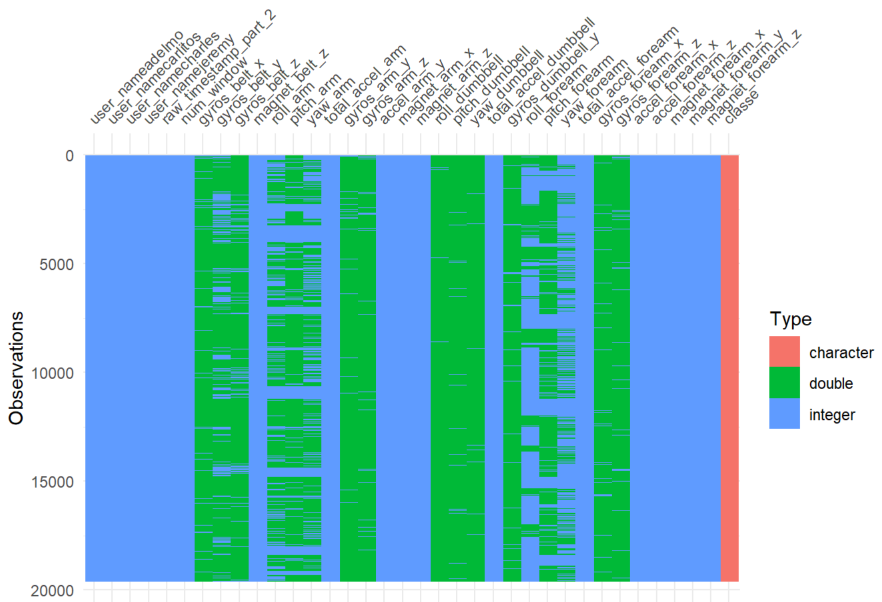
```
## $linearCombos
## list()
##
## $remove
## NULL
```

Addtional plots

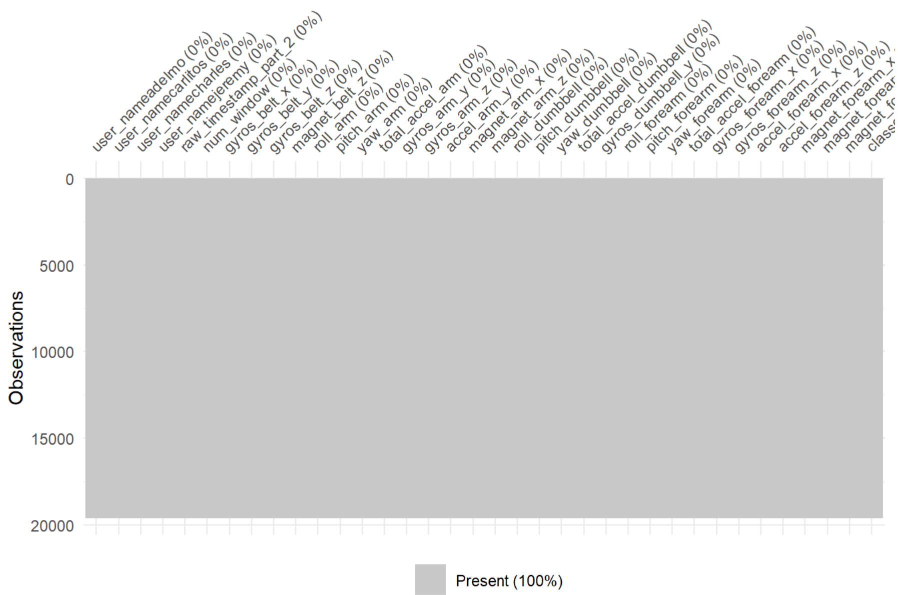
All variables are numeric, and we have no missing values among the predictors. Nice!!!

```
visdat::vis_dat(df)
```

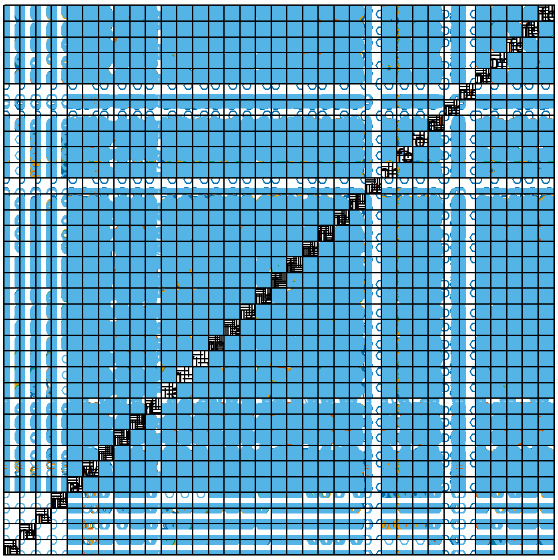

```
visdat::vis_guess(df)
```




```
visdat::vis_miss(df)
```



```
featurePlot(x = df[, -which(names(df) == "classe")], y =df$classe, plot = 'pairs')
```



Scatter Plot Matrix

```
#featurePlot(x = df[, setdiff(names(df), "classe")], y =df$classe, plot = 'box')
```

As there are no NAs in the dataset, there is no need for NA imputation.

```
df |> is.na() |> colSums() |> as_tibble() %>% arrange(value, desc = TRUE)
```

```
## # A tibble: 36 × 1
##    value
##    <dbl>
##  1     0
##  2     0
##  3     0
##  4     0
##  5     0
##  6     0
##  7     0
##  8     0
##  9     0
## 10     0
## # ℹ 26 more rows
```

The dependent variable is multinomial. There is some imbalance, but it is not strong.

```
table(df$classe) |> prop.table() |> knitr::kable()
```

Var1	Freq
A	0.2843747
B	0.1935073
C	0.1743961
D	0.1638977
E	0.1838243

Modelling

Let's split the dataset into training set and test set.

```
set.seed(123)
trainIndex <- createDataPartition(df$classe, p = 0.75, list = FALSE)
trainingSet <- df[trainIndex,]
testSet <- df[-trainIndex,]
```

The next step is to define a trainControl function. CV with 5 folds is defined, for the sake of time. I save predictions of each subfold and class probabilities.

```
ctrl <- trainControl(
  method = "cv",#repeatedcv
  number = 5,
  #repeats = 5,
  savePredictions = "final",
  #multiClassSummary = TRUE,
  classProbs = TRUE,
  verboseIter = FALSE,
  allowParallel = TRUE
)
```

Given the complexity of the models, the estimation is “parallelized”.

```
library(doParallel)
```

```
## Warning: package 'doParallel' was built under R version 4.3.3
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
# Check how many cores you have to work with
detectCores()
```

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

```
# Set the number of clusters caret has to work with. Creates number of clusters = cores-1
cl <- makePSOCKcluster(detectCores() - 2)

#cl <- makePSOCKcluster(8)
registerDoParallel(cl)
getDoParWorkers()
```

```
## [1] 10
```

Nine different models are tested. A wrapping function is used to pass all the models at once.

```
# define models to try
models <- c("multinom", "lda", "naive_bayes", "svmRadial", "knn", "rpart", "ranger", "glmnet", "nnet")

# set CV control for knn, k-folds
# control <- trainControl(method = "cv", number = 10, p = .9) # 10 fold, 10%

# fit models

train_models <- lapply(models, function(model){
    print(model)
    set.seed(123)
    train(classe ~ .,
          method = model,
          data = trainingSet,
          trControl = ctrl,
          preProcess = c("center","scale"),
          metric = "Kappa",
          tuneLength = 10)
})
```

```
## [1] "multinom"
## # weights:  185 (144 variable)
## initial  value 23687.707195
## iter  10 value 17244.732625
## iter  20 value 16607.200984
## iter  30 value 16540.454452
## iter  40 value 16445.315999
## iter  50 value 16295.365342
## iter  60 value 15982.515470
## iter  70 value 15910.543293
## iter  80 value 15883.916703
## iter  90 value 15849.866030
## iter 100 value 15828.685612
## final  value 15828.685612
## stopped after 100 iterations
## [1] "lda"
## [1] "naive_bayes"
## [1] "svmRadial"
## [1] "knn"
## [1] "rpart"
## [1] "ranger"
## [1] "glmnet"
```

```
## Warning: from glmnet C++ code (error code -6); Convergence for 6th lambda value
## not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
## [1] "nnet"
## # weights:  784
## initial  value 26845.148382
## iter  10 value 18435.408117
## iter  20 value 12202.334868
## iter  30 value 8939.874706
## iter  40 value 7394.847830
## iter  50 value 6856.260234
## iter  60 value 6227.252150
## iter  70 value 5697.484242
## iter  80 value 5367.323423
## iter  90 value 5084.698500
## iter 100 value 4643.167780
## final  value 4643.167780
## stopped after 100 iterations
```

```
names(train_models) <- models


stopCluster(cl)
registerDoSEQ()
```

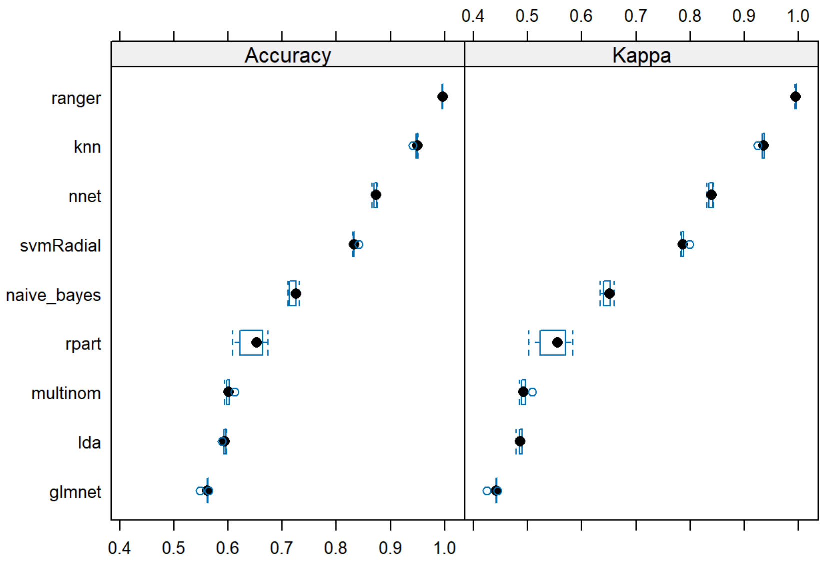
Now, let’s see the summary results for each method:

```
resamples(train_models) |> summary()
```

```
##
## Call:
## summary.resamples(object = resamples(train_models))
##
## Models: multinom, lda, naive_bayes, svmRadial, knn, rpart, ranger, glmnet, nnet
## Number of resamples: 5
##
## Accuracy
##                  Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## multinom    0.5942275 0.5974864 0.6009517 0.6015088 0.6024465 0.6124321    0
## lda         0.5886549 0.5931339 0.5942275 0.5938307 0.5959905 0.5971467    0
## naive_bayes 0.7111791 0.7130730 0.7252038 0.7212941 0.7256968 0.7313179    0
## svmRadial   0.8297655 0.8308424 0.8325976 0.8336060 0.8332201 0.8416044    0
## knn         0.9408767 0.9473148 0.9493886 0.9475466 0.9497453 0.9504076    0
## rpart       0.6083560 0.6222826 0.6523955 0.6442484 0.6641766 0.6740313    0
## ranger      0.9952462 0.9955812 0.9962636 0.9961272 0.9966021 0.9969429    0
## glmnet      0.5486064 0.5616299 0.5616718 0.5596533 0.5625000 0.5638587    0
## nnet        0.8664174 0.8695209 0.8730051 0.8719925 0.8750000 0.8760190    0
##
## Kappa
##                  Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## multinom    0.4852997 0.4893654 0.4931222 0.4944000 0.4962871 0.5079256    0
## lda         0.4794062 0.4847898 0.4868927 0.4861762 0.4895549 0.4902375    0
## naive_bayes 0.6346709 0.6399725 0.6516901 0.6477652 0.6524509 0.6600414    0
## svmRadial   0.7830249 0.7844947 0.7867240 0.7880824 0.7876580 0.7985104    0
## knn         0.9252704 0.9333455 0.9359710 0.9336552 0.9364361 0.9372530    0
## rpart       0.5024526 0.5228816 0.5548358 0.5466357 0.5698626 0.5831457    0
## ranger      0.9939878 0.9944104 0.9952737 0.9951015 0.9957021 0.9961333    0
## glmnet      0.4242584 0.4417969 0.4419617 0.4391091 0.4428442 0.4446842    0
## nnet        0.8309991 0.8349822 0.8395231 0.8381342 0.8419477 0.8432191    0
```

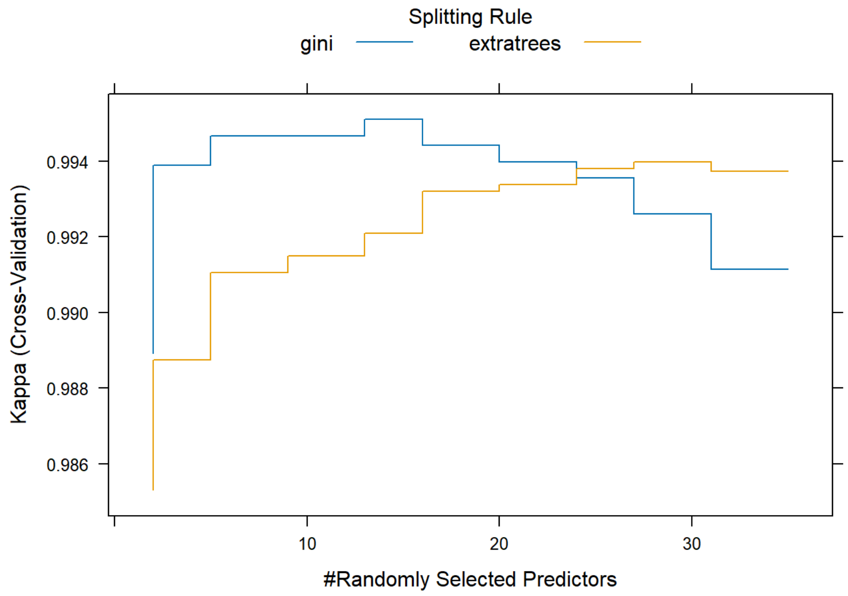
Nice, our best models are random forests (ranger), knn and neural nets. But, these results pertain to the training set.

```
bwplot(resamples(train_models))
```



The random forest model has an outstanding performance. Let's see its behavior during the tunning process:

```
plot(train_models$ranger, print.thres = 0.5, type="S")
```



Below, one may find the results of the tunning process of the ranger model:

```
train_models$ranger$results
```

```
##    mtry min.node.size  splitrule  Accuracy     Kappa   AccuracySD      KappaSD
## 1     2             1       gini 0.9912349 0.9889121 0.0014713192 0.0018619494
## 2     2             1 extratrees 0.9883814 0.9853022 0.0018529431 0.0023444055
## 3     5             1       gini 0.9951761 0.9938983 0.0012793425 0.0016177325
## 4     5             1 extratrees 0.9910992 0.9887410 0.0024901563 0.0031493596
## 5     9             1       gini 0.9957875 0.9946719 0.0009175563 0.0011603605
## 6     9             1 extratrees 0.9929339 0.9910621 0.0011860542 0.0014995927
## 7    13             1       gini 0.9957874 0.9946717 0.0010082295 0.0012750976
## 8    13             1 extratrees 0.9932735 0.9914916 0.0011372907 0.0014383622
## 9    16             1       gini 0.9961272 0.9951015 0.0007042430 0.0008906403
## 10   16             1 extratrees 0.9937492 0.9920932 0.0009784504 0.0012371935
## 11   20             1       gini 0.9955838 0.9944142 0.0008649619 0.0010937553
## 12   20             1 extratrees 0.9946324 0.9932105 0.0010850181 0.0013720383
## 13   24             1       gini 0.9952438 0.9939841 0.0009307749 0.0011773089
## 14   24             1 extratrees 0.9947684 0.9933826 0.0007807620 0.0009870496
## 15   27             1       gini 0.9949041 0.9935543 0.0014218839 0.0017987446
## 16   27             1 extratrees 0.9951079 0.9938120 0.0010639936 0.0013457479
## 17   31             1       gini 0.9941566 0.9926092 0.0011878478 0.0015026340
## 18   31             1 extratrees 0.9952439 0.9939841 0.0009608578 0.0012152244
## 19   35             1       gini 0.9930016 0.9911483 0.0013729492 0.0017368401
## 20   35             1 extratrees 0.9950400 0.9937262 0.0011670257 0.0014759902
```

So, our best tunning parameters are as follows:

```
train_models$ranger$bestTune
```

```
##   mtry splitrule min.node.size
## 9   16      gini             1
```

Also remarkable is that results are consistent across folds.

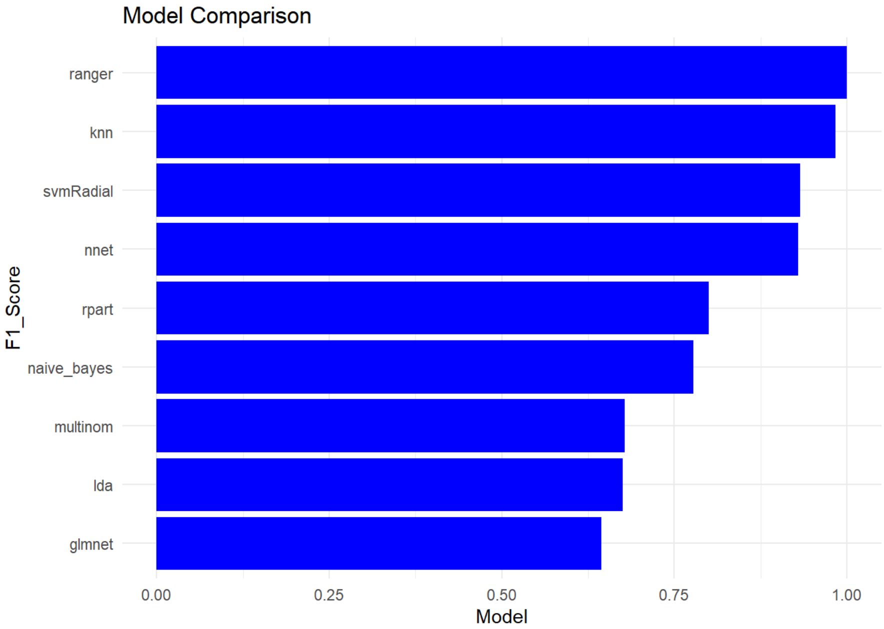
```
train_models$ranger$resample
```

```
##    Accuracy     Kappa Resample
## 1 0.9952462 0.9939878    Fold2
## 2 0.9962636 0.9952737    Fold3
## 3 0.9955812 0.9944104    Fold5
## 4 0.9969429 0.9961333    Fold4
## 5 0.9966021 0.9957021    Fold1
```

But, how does each model behave in the testing set?

Now, let's see how the different models behaved:

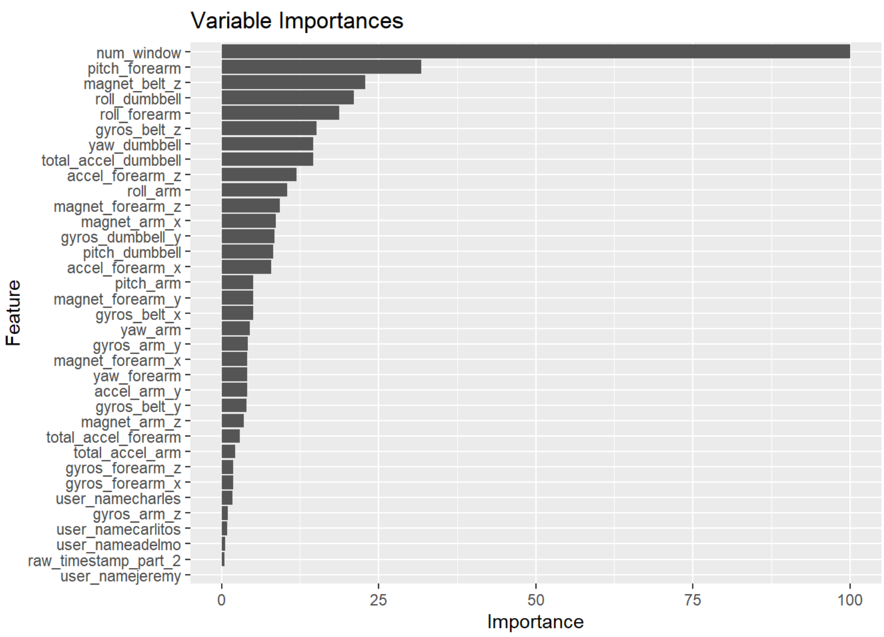
```
sapply(train_models,
       function(x){
         pred <- predict(x, testSet)
         MLmetrics::F1_Score(y_true = testSet$classe, y_pred = pred)

       }) |> reshape2::melt() %>%
#rowid_to_column() %>%
  rownames_to_column(var = "model")|>
  ggplot(aes(x=reorder(model, value), y=value))+
  geom_bar(stat="identity", fill = "blue")+
  ggtitle("Model Comparison")+
  labs(x="F1_Score", y="Model")+
  coord_flip() +
  theme_minimal()
```



Ok, these results also point to the prevalence of the random forest model as the one displaying larger out of sample accuracy.

What about variable importance? For that, one has to retrain our model and define “importance = impurity”

```
train(classe ~ .,
          method = "ranger",
          data = trainingSet,
          trControl = ctrl,
          preProcess = c("center","scale"),
#          metric = "Kappa",
          importance='impurity',
          tuneGrid = train_models$ranger$bestTune) %>%
          varImp(.) %>%
          ggplot() +
          ggtitle("Variable Importances")
```



Now, let's see some additional out of sample prediction stats:

```
# Calculate performance on the Test. Pass two vector to calculate performance metrics
ranger_fit <- train_models$ranger
pred <- predict(ranger_fit, testSet) |> as_data_frame()
```

```
## Warning: `as_data_frame()` was deprecated in tibble 2.0.0.
## ℹ Please use `as_tibble()` (with slightly different semantics) to convert to a
##   tibble, or `as.data.frame()` to convert to a data frame.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
confusionMatrix(factor(testSet$classe), factor(pred$value))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    0  949    0    0    0
##          C    0    2  853    0    0
##          D    0    1    3  800    0
##          E    0    0    0    0  901
##
## Overall Statistics
##
##                Accuracy : 0.9988
##                  95% CI : (0.9973, 0.9996)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9985
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9968   0.9965   1.0000   1.0000
## Specificity            1.0000   1.0000   0.9995   0.9990   1.0000
## Pos Pred Value         1.0000   1.0000   0.9977   0.9950   1.0000
## Neg Pred Value         1.0000   0.9992   0.9993   1.0000   1.0000
## Prevalence             0.2845   0.1941   0.1746   0.1631   0.1837
## Detection Rate         0.2845   0.1935   0.1739   0.1631   0.1837
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      1.0000   0.9984   0.9980   0.9995   1.0000
```

```
postResample(factor(testSet$classe), factor(pred$value))
```

```
##  Accuracy     Kappa
## 0.9987765 0.9984524
```

```
## or predicting using the probabilities (nice because you can get ROC)
probs <- extractProb(list(model=ranger_fit), testX = testSet, testY = testSet$classe)

#predict(ranger_fit, testSet, type = "prob")
```

```
## Make sure the levels are appropriate for multiClassSummary(), ie case group is first level
levs <- LETTERS[1:5]
probs$obs <- factor(probs$obs, levels = levs)
probs$pred <- factor(probs$pred, levels = levs)
```

```
## Calculating Accuracy
mean(probs$obs==probs$pred)
```

```
## [1] 0.9996942
```

```
## pred column shows model predicted label if cutoff for calling label = 0.5
table(probs$obs, probs$pred)
```

```
##
##        A    B    C    D    E
##   A 5580    0    0    0    0
##   B    0 3797    0    0    0
##   C    0    2 3420    0    0
##   D    0    1    3 3212    0
##   E    0    0    0    0 3607
```

```
multiClassSummary(probs, lev = levels(probs$obs))
```

```
##               logLoss                   AUC                 prAUC
##            0.03410196            0.99999940            0.86723207
##              Accuracy                 Kappa               Mean_F1
##            0.99969422            0.99961324            0.99965047
##      Mean_Sensitivity      Mean_Specificity   Mean_Pos_Pred_Value
##            0.99963435            0.99992505            0.99966682
##   Mean_Neg_Pred_Value        Mean_Precision           Mean_Recall
##            0.99992656            0.99966682            0.99963435
##   Mean_Detection_Rate Mean_Balanced_Accuracy
##            0.19993884            0.99977970
```

```
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.3.3
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
rangerROC <- roc(testSet$classe |> factor(), probs$A[probs$dataType== "Test"])
```

```
## Warning in roc.default(factor(testSet$classe), probs$A[probs$dataType == :
## 'response' has more than two levels. Consider setting 'levels' explicitly or
## using 'multiclass.roc' instead
```

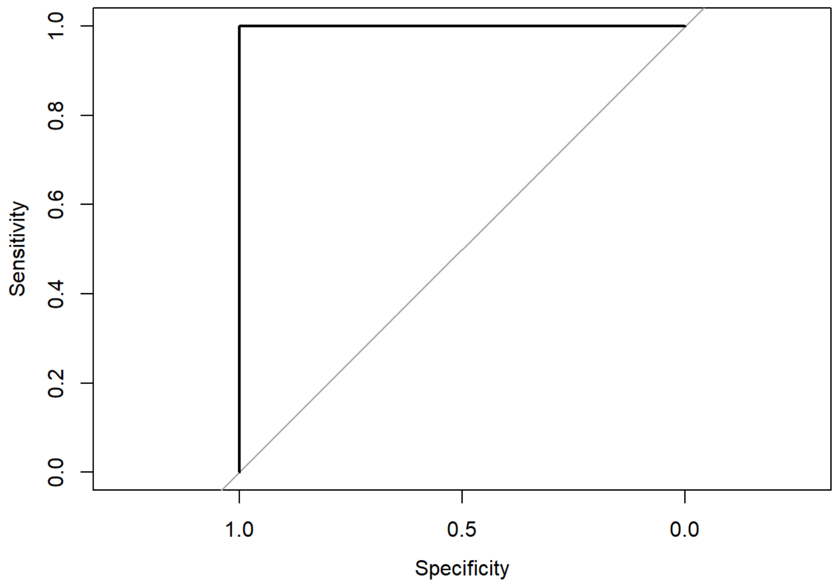
```
## Setting levels: control = A, case = B
```

```
## Setting direction: controls > cases
```

```
rangerROC
```

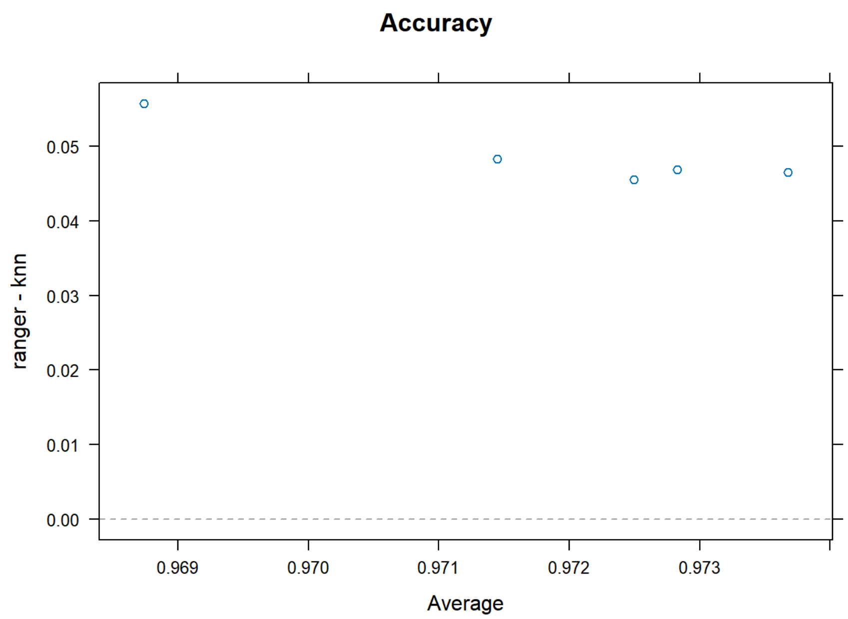
```
##
## Call:
## roc.default(response = factor(testSet$classe), predictor = probs$A[probs$dataType ==     "Test"])
##
## Data: probs$A[probs$dataType == "Test"] in 1395 controls (factor(testSet$classe) A) > 949 cases (factor(testSet$classe) B).
## Area under the curve: 1
```

```
plot(rangerROC, metric = "ROC")
```



Let's compare our two best models:

```
train_models[c("ranger", "knn")] |> resamples() |>
  xyplot(., what = "BlandAltman")
```




```
train_models[c("ranger", "knn")] |>
  resamples() |>
  diff() |>
  summary()
```

```
##
## Call:
## summary.diff.resamples(object =
##  diff(resamples(train_models[c("ranger", "knn")])))
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## Accuracy
##        ranger    knn
## ranger           0.04858
## knn    1.223e-05
##
## Kappa
##        ranger    knn
## ranger           0.06145
## knn    1.199e-05
```

Random forest seems to clearly outperform KNN.

Final Predictions

```
validation_set2 <- validation_set %>%
  select(any_of(nonMissingVars))

validation_set2 <- validation_set2 %>%
  select(!names(validation_set2)[nsv_obj$nzv])


validation_set2 <- dummyVars( ~ ., validation_set2) %>%
  predict(validation_set2) %>%
  as.data.frame()



validation_set2 %<>%
  select(-any_of(correl_cols))

predict(ranger_fit, validation_set2) |> as_data_frame()
```

```
## # A tibble: 20 × 1
##    value
##    <fct>
##  1 B
##  2 A
##  3 B
##  4 A
##  5 A
##  6 E
##  7 D
##  8 B
##  9 A
## 10 A
## 11 B
## 12 C
## 13 B
## 14 A
## 15 E
## 16 E
## 17 A
## 18 B
## 19 B
## 20 B
```