Diabetes Detection System Project

Xiaobin Hu

October 30, 2019

Contents

1 Overview

This project will use Pima Indian Diabetes dataset, to build a predective model, to detect how likely one can develop diabetes based on some feature, such as body mass index, diastolic blood pressure, etc.

This project is the requirement of requirement of the **HervardX: PH125.9x Data Science: Capstone** course.

1.1 Introduction

Pima Indian Diabetes dataset was collected from Pima Indians, a group of Native Americans living in Arizona. The data is originally from the National Insitude of Diabets and Digestive and Kidney Diseases. It has the following features:

- Number of times pregnant
- Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- Diastolic blood pressure (mm Hg)
- Triceps skin fold thickness (mm)
- 2-Hour serum insulin (mu U/ml)
- Body mass index (weight in kg/(height in m)^2)
- Diabetes pedigree function
- Age (years)

The last variable is the outcome categorical values

- diabetes (1 - yes, 0 - no)

1.2 Objective

In this project, I will carry out the fundamental data wrangling steps, to prepare data for machine learning:

- Data Aquisition
- Data Inspection
- Data Cleasing
- Data Transformation

I will use different algorithms to build machine learning models with default parameters, to choose the best candidate algorythms for further improvement. Next I will use Ensemble and Hyper Parameter Tuning strategies to improve my models.

2 Methods and Analysis

2.1 Data Preparation

Usally for a data science project, data preparation includes the following steps:

- Exploration - Find the right data.
- Aquisition - Collect the data, discover each dataset.
- Cleasing and Vlidation - Clean up data, and validate it by testing for errors.
- Transformation and Enrichment - Updating the format or value entries. Add and connect data with other related information.
- Store data - Once data is prepared, store it.

2.1.1 Data Aquisition

Pima Indian Diabetes dataset can be download from Github.

Download the data and load it to a data frame.

```
dl <- tempfile()
url <- "https://gist.githubusercontent.com/ktisha/c21e73a1bd1700294ef790c56c8aec1f/raw/819b69b5736821cc
download.file(url, dl)
pima_diabetes_dat <- data.frame(readr::read_csv(dl, skip = 9, col_names = FALSE))

# Remove uncessary object
rm(dl)
```

2.1.2 Cleasing and Validation

After data is downloaded, inspect and validate it. Give proper names to columns. The data is going to be splitted into two parts:

- **train_set** - This daset has 80% of the data randomly selected from the original Pima Indian Diabetes dataset. The machine learning alrorythms will be trained against this dataset.
- **test_set** - This dataset has the rest 20% data of original dataset. The performance of all algorythms will be evaluated with this dateset. Performance of modles will be measured by accuracy values.

Assigne proper names to each feature. Change the last variable to factor.

```
colnames(pima_diabetes_dat) <- c("num_Of_pregnance",
                                 "plasma_glucose_concentration",
                                 "diastolic_blood_pressure",
                                 "tricep_skin_fold_thickness",
```

```
                                  "serum_insulin",
                                  "bmi",
                                  "diabetes_pedigree_function",
                                  "age",
                                  "diabetes")
pima_diabetes_dat$diabetes <- factor(pima_diabetes_dat$diabetes)
```

Then explore the dataset to inspect the data. Visualize the data to understand it.

```
dim(pima_diabetes_dat)
```

```
## [1] 768   9
```

```
pima_diabetes_dat[1:6, ] %>%
  knitr::kable() %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

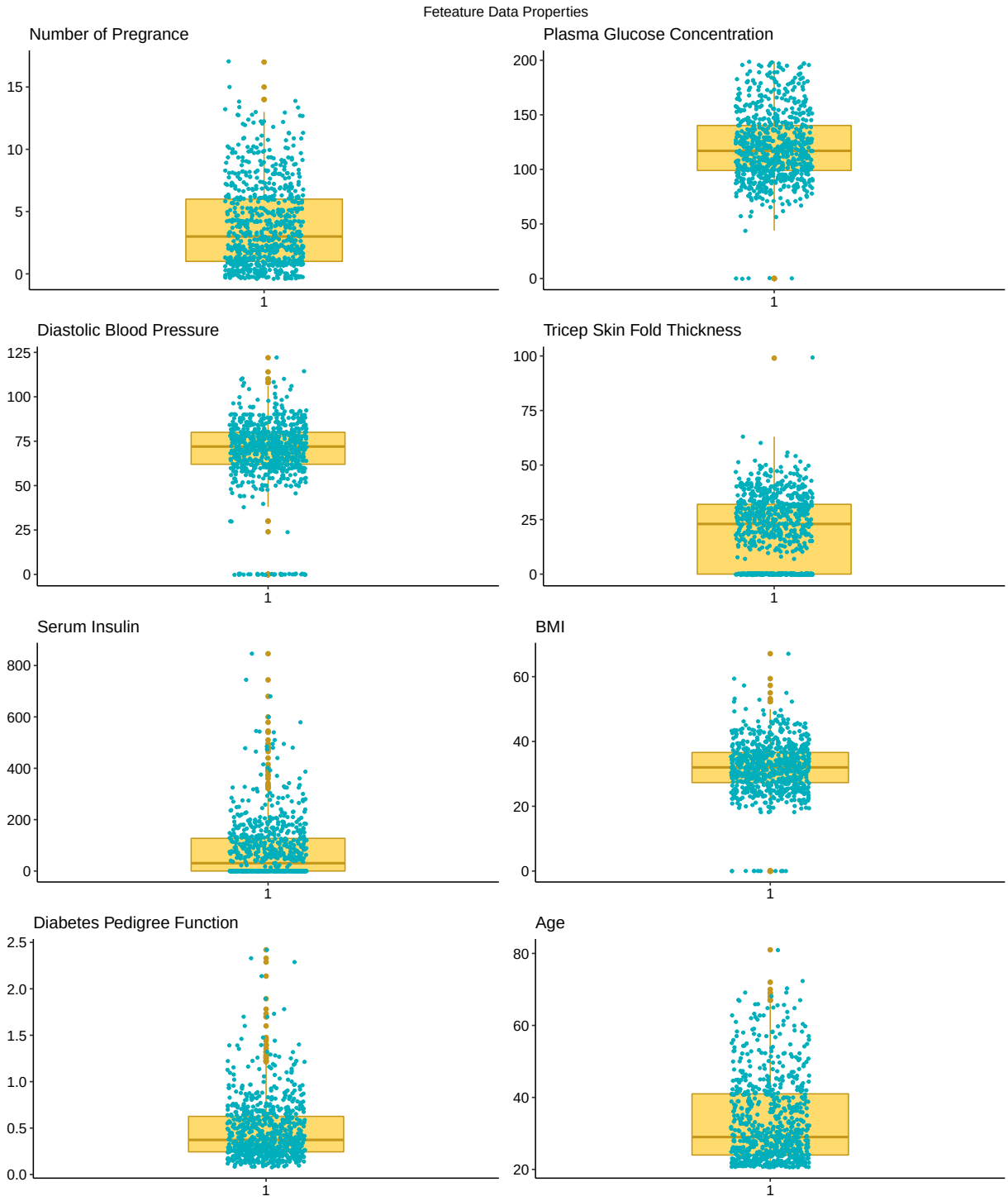
num_Of_pregnance	plasma_glucose_concentration	diastolic_blood_pressure	tricep_skin_fold_thickness	serum_insulin	bmi	diabetes_pedigree_function	age	diabetes
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0

Use boxplot charts to check the general properties of each features: percentiles, means, maximums,minimums, and inter quartile ranges (IQR). Identify any outliers and missing values.

```
p_nog_1 <- pima_diabetes_dat %>%
  ggplot(aes(x = factor(1), y = num_Of_pregnance)) +
  geom_boxplot(width = 0.4, fill = "#FFDB6D", color = "#C4961A") +
  geom_jitter(width = 0.1, size = 1, color = "#00AFBB") +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
  labs(title = "Number of Pregrance",
    x = NULL,
    y = NULL) +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme_pubr()

p_pgc_1 <- pima_diabetes_dat %>%
  ggplot(aes(x = factor(1), y = plasma_glucose_concentration)) +
  geom_boxplot(width = 0.4, fill = "#FFDB6D", color = "#C4961A") +
  geom_jitter(width = 0.1, size = 1, color = "#00AFBB") +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
  labs(title = "Plasma Glucose Concentration",
       x = NULL,
       y = NULL) +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme_pubr()

p_dbp_1 <- pima_diabetes_dat %>%
  ggplot(aes(x = factor(1), y = diastolic_blood_pressure)) +
  geom_boxplot(width = 0.4, fill = "#FFDB6D", color = "#C4961A") +
  geom_jitter(width = 0.1, size = 1, color = "#00AFBB") +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
  labs(title = "Diastolic Blood Pressure",
       x = NULL,
       y = NULL) +
```

```
  theme(plot.title = element_text(hjust = 0.5)) +
  theme_pubr()

p_tsft_1 <- pima_diabetes_dat %>%
  ggplot(aes(x = factor(1), y = tricep_skin_fold_thickness)) +
  geom_boxplot(width = 0.4, fill = "#FFDB6D", color = "#C4961A") +
  geom_jitter(width = 0.1, size = 1, color = "#00AFBB") +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
  labs(title = "Tricep Skin Fold Thickness",
       x = NULL,
       y = NULL) +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme_pubr()

p_si_1 <- pima_diabetes_dat %>%
  ggplot(aes(x = factor(1), y = serum_insulin)) +
  geom_boxplot(width = 0.4, fill = "#FFDB6D", color = "#C4961A") +
  geom_jitter(width = 0.1, size = 1, color = "#00AFBB") +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
  labs(title = "Serum Insulin",
       x = NULL,
       y = NULL) +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme_pubr()

p_bmi_1 <- pima_diabetes_dat %>%
  ggplot(aes(x = factor(1), y = bmi)) +
  geom_boxplot(width = 0.4, fill = "#FFDB6D", color = "#C4961A") +
  geom_jitter(width = 0.1, size = 1, color = "#00AFBB") +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
  labs(title = "BMI",
       x = NULL,
       y = NULL) +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme_pubr()


p_dpf_1 <- pima_diabetes_dat %>%
  ggplot(aes(x = factor(1), y = diabetes_pedigree_function)) +
  geom_boxplot(width = 0.4, fill = "#FFDB6D", color = "#C4961A") +
  geom_jitter(width = 0.1, size = 1, color = "#00AFBB") +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
  labs(title = "Diabetes Pedigree Function",
       x = NULL,
       y = NULL) +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme_pubr()

p_age_1 <- pima_diabetes_dat %>%
  ggplot(aes(x = factor(1), y = age)) +
  geom_boxplot(width = 0.4, fill = "#FFDB6D", color = "#C4961A") +
  geom_jitter(width = 0.1, size = 1, color = "#00AFBB") +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
```

```
  labs(title = "Age",
       x = NULL,
       y = NULL) +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme_pubr()

grid.arrange(p_nog_1, p_pgc_1, p_dbp_1, p_tsft_1, p_si_1, p_bmi_1,p_dpf_1, p_age_1,
             nrow = 4,
             top = "Feteature Data Properties",
             left = "")
```



The boxplots tell us that many features have zero values, which may be treated as missing values. There also may be some outliers, such as Diastolic Blood Pressoure lower around 25.

Next, let's validate the data to find:

- Duplicate observations.
- Any missing values
- Any feature has zero or near-zero variance.
- Features highly correlated.

- Any linear combinations betwee features.

```
dupes <- duplicated(pima_diabetes_dat)
table(dupes)
```

```
## dupes
## FALSE
##   768
```

There is 0 duplicate observation has been found. Let's check if there is any zero or near-zero variance feature in the dataset.

```
feature_variance <- caret::nearZeroVar(pima_diabetes_dat, saveMetrics = TRUE)
feature_variance %>%
  knitr::kable()
```

	freqRatio	percentUnique	zeroVar	nzv
num_Of_pregnance	1.216216	2.2135417	FALSE	FALSE
plasma_glucose_concentration	1.000000	17.7083333	FALSE	FALSE
diastolic_blood_pressure	1.096154	6.1197917	FALSE	FALSE
tricep_skin_fold_thickness	7.322581	6.6406250	FALSE	FALSE
serum_insulin	34.000000	24.2187500	FALSE	FALSE
bmi	1.083333	32.2916667	FALSE	FALSE
diabetes_pedigree_function	1.000000	67.3177083	FALSE	FALSE
age	1.142857	6.7708333	FALSE	FALSE
diabetes	1.865672	0.2604167	FALSE	FALSE

There is 0 zero or near-zero variance feature(s).

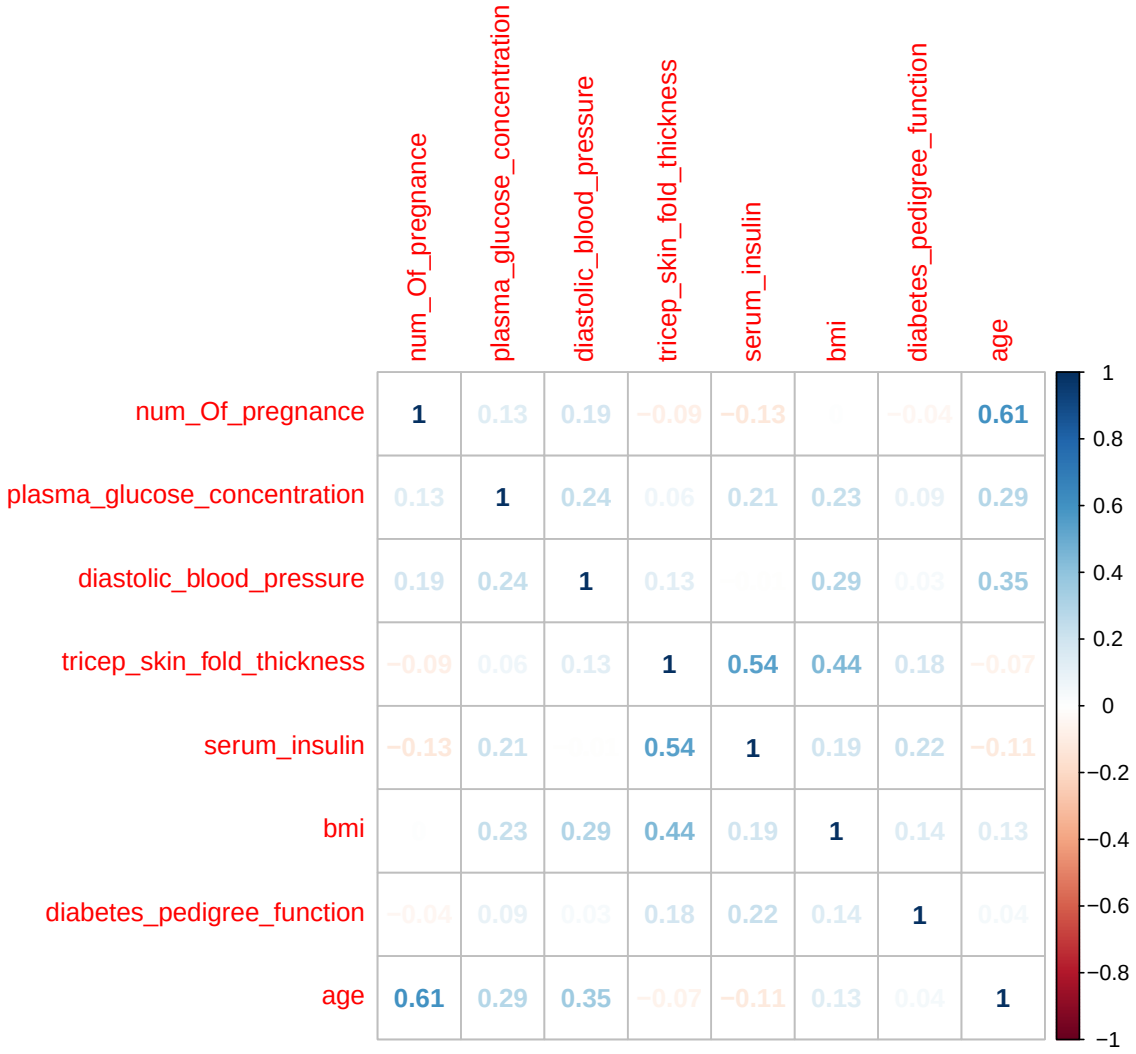
Now let's check correlations between features.

```
df_corr <- cor(pima_diabetes_dat[!colnames(pima_diabetes_dat)
                                 %in% c("diabetes")], method = "spearman")
high_corr <- caret::findCorrelation(df_corr, cutoff = 0.9)
length(high_corr)
```

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

There is 0 high correlation(s) found between the features. Let's visualize the correlations in the following chart.

```
corrplot(df_corr, method = "number")
```



I also want to check if there is any linear combinations among the features.

```
linear_combos <- caret::findLinearCombos(pima_diabetes_dat[!colnames(pima_diabetes_dat) %in% c("diabetes
length(linear_combos$linearCombos)
```

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

There is 0 linear combinations found within the features. Collect the basic descripive statistic information about this dataset.

```
pima_diabetes_dat %>%
  sjmisc::descr() %>%
  knitr::kable() %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

	var	type	label	n	NA.prc	mean	sd	se	md	trimmed	range	skew
6	num_Of_pregnance	numeric	num_Of_pregnance	768	0	3.8450521	3.3695781	0.1215892	3.0000	3.4610390	17 (0-17)	0.9016740
7	plasma_glucose_concentration	numeric	plasma_glucose_concentration	768	0	120.8945312	31.9726182	1.1537125	117.0000	119.3798701	199 (0-199)	0.1737535
5	diastolic_blood_pressure	numeric	diastolic_blood_pressure	768	0	69.1054688	19.3558072	0.6984425	72.0000	71.3571429	122 (0-122)	-1.8436080
9	tricep_skin_fold_thickness	numeric	tricep_skin_fold_thickness	768	0	20.5364583	15.9522176	0.5756261	23.0000	19.9366883	99 (0-99)	0.1093725
8	serum_insulin	numeric	serum_insulin	768	0	79.7994792	115.2440024	4.1585097	30.5000	56.7451299	846 (0-846)	2.2722509
2	bmi	numeric	bmi	768	0	31.9925781	7.8841603	0.2844951	32.0000	31.9592532	67.1 (0-67.1)	-0.4289816
4	diabetes_pedigree_function	numeric	diabetes_pedigree_function	768	0	0.4718763	0.3313286	0.0119558	0.3725	0.4215536	2.34 (0.08-2.42)	1.9199111
1	age	numeric	age	768	0	33.2408854	11.7602315	0.4243608	29.0000	31.5438312	60 (21-81)	1.1295967
3	diabetes	categorical	diabetes	768	0	0.3489583	0.4769514	0.0172105	0.0000	0.3116883	1 (0-1)	0.6350166

For the summary statistic information, we know that there is no NA values, but there are many numeric features which have zero values. Many of these cases, exception 'number of pregnance' feature, should be treated as missing values.

First, check how many observations with plasma glucose concentration equal to zero.

```
sum(pima_diabetes_dat$plasma_glucose_concentration == 0)
```

```
## [1] 5
```

There are 5 observations which have missing values for Plasm Glucose Concentration feature. Imputation can be implemented for these missing values using mean of this feature.

```
pima_diabetes_dat <- pima_diabetes_dat%>%
  mutate(plasma_glucose_concentration_clean = ifelse(plasma_glucose_concentration == 0,
                                                     mean(plasma_glucose_concentration),
                                                     plasma_glucose_concentration))
```

Secondly, check how many observations which have Diastolic Blood Pressure value missing.

```
sum(pima_diabetes_dat$diastolic_blood_pressure == 0)
```

```
## [1] 35
```

There are 35 observations which have missing values for Diastolic Blood Presure feature. Imputation can be implemented for these missing values using mean of this feature.

```
pima_diabetes_dat <- pima_diabetes_dat%>%
  mutate(diastolic_blood_pressure_clean = ifelse(diastolic_blood_pressure == 0,
                                                 mean(diastolic_blood_pressure),
                                                 diastolic_blood_pressure))
```

Thirdly, check how many observations which have Tricep Skin Fold Thickness value missing.

```
sum(pima_diabetes_dat$tricep_skin_fold_thickness == 0)
```

```
## [1] 227
```

There are 227 observations which have missing values for Diastolic Blood Presure feature, which is 29.56% of total data. Because of the large amount of missing values for this feature, imputation does not make sense, so my strategy for this case is dropping this feature from machine learning.

Next, check how many observations which have Serum Insulin value missing.

```
sum(pima_diabetes_dat$serum_insulin == 0)
```

```
## [1] 374
```

There are 374 observations which have missing values for Diastolic Blood Presure feature, which is 48.7% of total data. Because of the large amount of missing values for this feature, imputation does not make sense, so my strategy for this case is dropping this feature from machine learning.

Finally, check how many observeations which have zero values for BMI feature.

```
sum(pima_diabetes_dat$bmi == 0)
```

```
## [1] 11
```

There are 11 observations which have missing values for BMI feature. Imputation can be implemented for these missing values. Two types of impuations will be used: mean and median.

```
pima_diabetes_dat <- pima_diabetes_dat%>%
  mutate(bmi_clean = ifelse(bmi == 0, mean(bmi), bmi))
```

I will drop the following features from the diabetes dataset,

- tricep_skin_fold_thickness - There are too many missing values.
- serum_insulin - There are too many missing values

```
diabetes_dat_filtd <- pima_diabetes_dat %>%
  select(-c("tricep_skin_fold_thickness",
            "serum_insulin"))

diabetes_dat_filtd[1:6, ] %>%
  knitr::kable() %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

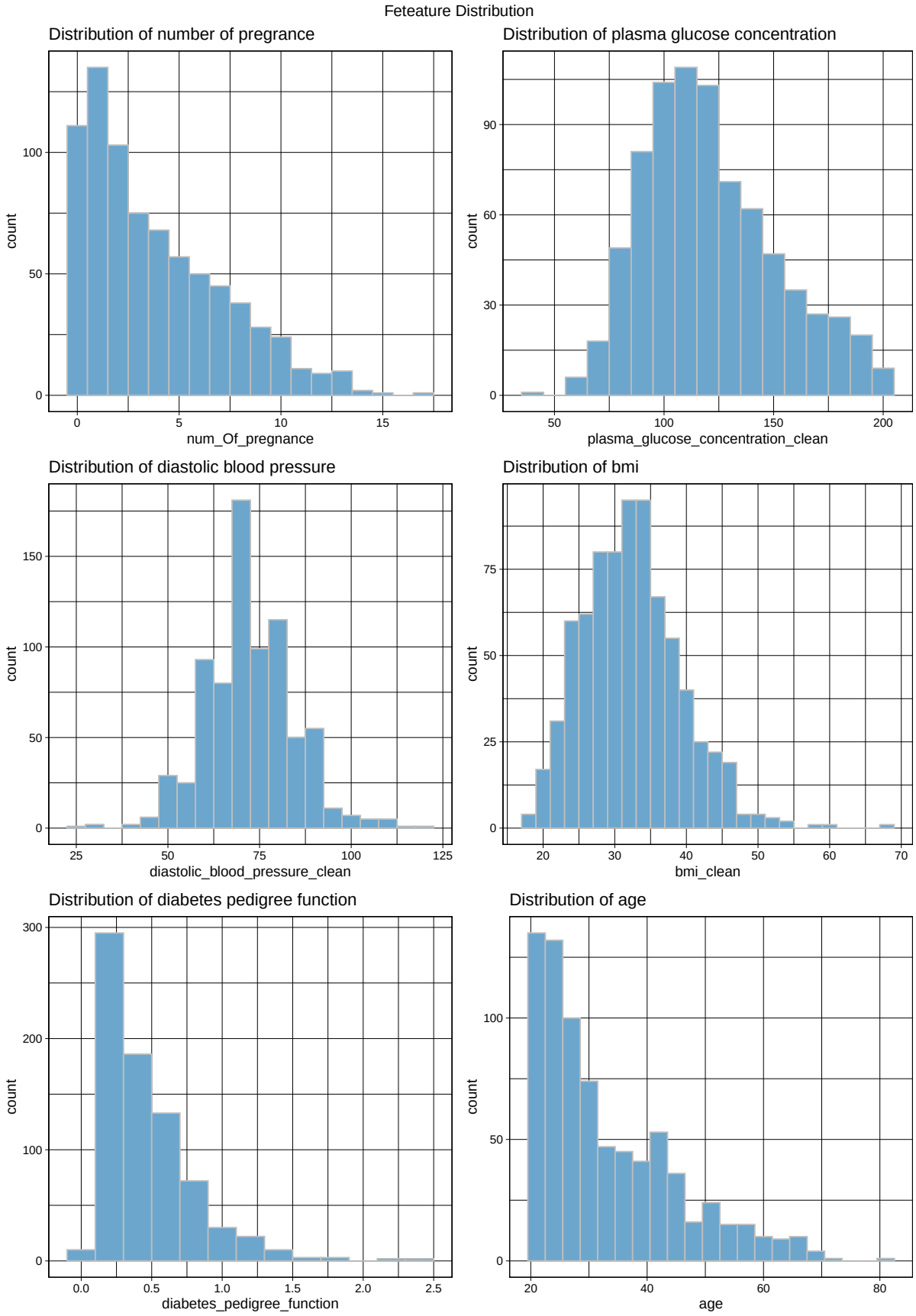
num_Of_pregnance	plasma_glucose_concentration	diastolic_blood_pressure	bmi	diabetes_pedigree_function	age	diabetes	plasma_glucose_concentration_clean	diastolic_blood_pressure_clean	bmi_clean
6	148	72	33.6	0.627	50	1	148	72	33.6
1	85	66	26.6	0.351	31	0	85	66	26.6
8	183	64	23.3	0.672	32	1	183	64	23.3
1	89	66	28.1	0.167	21	0	89	66	28.1
0	137	40	43.1	2.288	33	1	137	40	43.1
5	116	74	25.6	0.201	30	0	116	74	25.6

Plot distribution charts for all features, to understand the data.

```
fill <- 'skyblue3'
color <- 'grey'

# Plot distribution of number of pregrance
bw <- 1
plot_nog <- diabetes_dat_filtd %>%
  ggplot(aes(num_Of_pregnance)) +
  geom_histogram(
    aes(fill = I(fill),
        color = I(color)),
    binwidth = bw,
    show.legend = FALSE)  +
  labs( title = "Distribution of number of pregrance") +
  theme_linedraw()

bw <- 10
plot_pgc <- diabetes_dat_filtd %>%
  ggplot(aes(plasma_glucose_concentration_clean)) +
  geom_histogram(
    aes(fill = I(fill),
        color = I(color)),
    binwidth = bw,
    show.legend = FALSE)  +
  labs( title = "Distribution of plasma glucose concentration") +
  theme_linedraw()
```

```r
# plot_pgc

bw <- 5
plot_dbp <- diabetes_dat_filtd %>%
  ggplot(aes(diastolic_blood_pressure_clean)) +
  geom_histogram(
    aes(fill = I(fill),
        color = I(color)),
    binwidth = bw,
    show.legend = FALSE)  +
  labs( title = "Distribution of diastolic blood pressure") +
  theme_linedraw()

# plot_dbp

bw <- 2
plot_bmi <- diabetes_dat_filtd %>%
  ggplot(aes(bmi_clean)) +
  geom_histogram(
    aes(fill = I(fill),
        color = I(color)),
    binwidth = bw,
    show.legend = FALSE)  +
  labs( title = "Distribution of bmi") +
  theme_linedraw()

# plot_bmi

bw <- 0.2
plot_dpf <- diabetes_dat_filtd %>%
  ggplot(aes(diabetes_pedigree_function)) +
  geom_histogram(
    aes(fill = I(fill),
        color = I(color)),
    binwidth = bw,
    show.legend = FALSE)  +
  labs( title = "Distribution of diabetes pedigree function") +
  theme_linedraw()

# plot_dpf
bw <- 3
plot_age <- diabetes_dat_filtd %>%
  ggplot(aes(age)) +
  geom_histogram(
    aes(fill = I(fill),
        color = I(color)),
    binwidth = bw,
    show.legend = FALSE)  +
  labs( title = "Distribution of age") +
  theme_linedraw()

# plot_age
```

```r
grid.arrange(plot_nog, plot_pgc, plot_dbp, plot_bmi,plot_dpf, plot_age,
             nrow = 3,
             top = "Feteature Distribution",
             left = ""
)
```


Feteature Distribution
Distribution of number of pregrance
count
num_Of_pregnance
Distribution of plasma glucose concentration
count
plasma_glucose_concentration_clean
Distribution of diastolic blood pressure
count
diastolic_blood_pressure_clean
Distribution of bmi
count
bmi_clean
Distribution of diabetes pedigree function
count
diabetes_pedigree_function
Distribution of age
count
age

Now the dataset is tidy and ready for builing machine learning models. Split it to two parts: 80% of the randomly selecte data will be used for training. The rest of the data will be used for testing.

Pleaset note: it is important to keep the ratios of diabetes positive in traing and testing datasets to be very close.

```
set.seed(2007, sample.kind = "Rounding")
```

```
## Warning in set.seed(2007, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
train_index <- createDataPartition(diabetes_dat_filtd$diabetes,
                                   times = 1, p = 0.8, list = FALSE)
train_set <- diabetes_dat_filtd[train_index, ]
test_set <- diabetes_dat_filtd[-train_index, ]
```

Training dataset has 80.08% data randomly selected from the original dataset. The ratio of diabetes positive in original dataset is 34.9%. The ratio in training dataset is 34.96%. The ratio in testing dataset is 34.64%. The ratioes are very close, which is the excellent setting for machine learing.

2.1.3 Save Data and Load Data from Files

This project may take some time to finish. It is the best pracice to save the datasets into **data** subdirectory of current working directory in RData format. Nextime datasets can be populated again by loading data from local data files.

```
# Create a sudirectory to save the data file
if(!dir.exists(file.path(getwd(), "data"))){
  dir.create(file.path(getwd(), "data"), recursive = TRUE)
}
saveRDS(pima_diabetes_dat, "data/pima_diabetes_dat.rds")
saveRDS(diabetes_dat_filtd, "data/diabetes_dat_filtd.rds")
saveRDS(train_set, "data/diabetes_train_set.rds")
saveRDS(test_set, "data/diabetes_test_set.rds")
```

When you resume your work, you can uncomment the following code to populate datasets from local files. It can save you a lot of time for data preparation.

```
#   to repopulate datasets by loading data from local rdata files.
# train_set <- readr::read_rds("data/diabetes_train_set.rds")
# test_set <- readr::read_rds("data/diabetes_test_set.rds")
```

3 Models and Results

Machine learing model builings can have many iterations: feature engineering - algorythm selection - model training - model testing.

In order to find the ideal predictive models, I plan to follow these steps:

1. Try some of the most common claasifier algorythms. Compare their performance. Get the candidates for further tuning.
2. Use Ensemble strategy to try some combinations of the algorythms selected from step 1, to build stacked models. Compare the performance.
3. Tune hyper parameters for the algoryths with the best performance determined in step 1. Find the final model with the best performance.

3.1 Select Algorythm Candidates

I will use a list of the very common classification algorythms, to build models with default parameters. Then compare their accuracy values, and find the candidates for further improvement.

```
set.seed(1, sample.kind = "Rounding")
models <- c("glm", "lda", "naive_bayes", "svmLinear",
            "knn", "gamLoess", "multinom", "qda", "rf", "adaboost")
fits <- lapply(models, function(model){
  print(model)
  caret::train(diabetes ~ ., method = model, data = train_set)
})
names(fits) <- models
diabetes_hat_all <- sapply(fits, function(fit){
  diabetes_hat <- predict(fit, test_set)
})
diabetes_hat_all_df <- data.frame(diabetes_hat_all)
accuracy_all <- sapply(diabetes_hat_all_df, function(y_hat){
  diabetes_hat <- factor(y_hat)
  confusionMatrix(diabetes_hat, reference = test_set$diabetes)$overall["Accuracy"]
})
```

Here is the performance of all models

```
accuracy_all %>%
  knitr::kable()
```

	x
glm.Accuracy	0.7450980
lda.Accuracy	0.7385621
naive_bayes.Accuracy	0.7516340
svmLinear.Accuracy	0.7581699
knn.Accuracy	0.7385621
gamLoess.Accuracy	0.7843137
multinom.Accuracy	0.7450980
qda.Accuracy	0.7581699
rf.Accuracy	0.7647059
adaboost.Accuracy	0.7320261

The accuracy values of naive_bayes, svmLinear, gamLoess, qda, and rf models are all above 0.75. They will be the candidates for further tuning.

3.2 Ensemble and Stacked Models

Ensemble can combine a number of classifiers and use their predicted class probabilities as input features to another classifier. This method can usually result it improved accuracy.

3.2.1 Ensemble Iteration One

In this iteration, Random Forest and Naive Bayes classifiers will be used as base learners. The gbm classifier will be used as the super learner.

```
diabetes_task <- mlr::makeClassifTask(data = train_set, target = "diabetes")
base <- c("classif.randomForest", "classif.naiveBayes")

learns <- lapply(base, makeLearner)
learns <- lapply(learns, setPredictType, "prob")
```

```
sl <-
  mlr::makeStackedLearner(
    base.learners = learns,
    super.learner = "classif.gbm",
    predict.type = "prob",
    method = "stack.cv"
  )

stacked_fit <- mlr::train(sl, diabetes_task)
pred_stacked <- predict(stacked_fit, newdata = test_set)
```

The accuracy of this stacked model is:

```
mlr::calculateConfusionMatrix(pred_stacked)
```

```
##         predicted
## true      0  1 -err.-
##   0      83 17     17
##   1      20 33     20
##   -err.- 20 17     37
```

```
mlr::performance(pred_stacked, measures = list(acc, logloss))
```

```
##       acc   logloss
## 0.7581699 0.4502152
```

3.2.2 Ensemble Iteration Two

In this iteration, **Random Forest** and **LDA** classifiers will be used as base learners. The **GLMNET** classifier will be used as the super learner.

```
base <- c("classif.randomForest", "classif.lda")

learns <- lapply(base, makeLearner)
learns <- lapply(learns, setPredictType, "prob")
sl <-
  mlr::makeStackedLearner(
    base.learners = learns,
    super.learner = "classif.glmnet",
    predict.type = "prob",
    method = "stack.cv"
  )

stacked_fit <- mlr::train(sl, diabetes_task)
pred_stacked <- predict(stacked_fit, newdata = test_set)
```

The accuracy of this stacked model is:

```
mlr::calculateConfusionMatrix(pred_stacked)
```

```
##         predicted
## true      0  1 -err.-
##   0      86 14     14
##   1      20 33     20
##   -err.- 20 14     34
```

```
mlr::performance(pred_stacked, measures = list(acc, logloss))
```

```
##       acc   logloss
## 0.7777778 0.4597710
```

3.2.3 Ensemble Iteration Three

In this iteration, **GBM** and **Random Forest** classifiers will be used as base learners. The **GLMNET** classifier will be used as the super learner.

```
base <- c("classif.gbm", "classif.randomForest")

learns <- lapply(base, makeLearner)
learns <- lapply(learns, setPredictType, "prob")
sl <-
  mlr::makeStackedLearner(
    base.learners = learns,
    super.learner = "classif.lda",
    predict.type = "prob",
    method = "stack.cv"
  )
stacked_fit <- mlr::train(sl, diabetes_task)
pred_stacked <- predict(stacked_fit, newdata = test_set)
```

The accuracy of this stacked model is:

```
mlr::calculateConfusionMatrix(pred_stacked)
```

```
##         predicted
## true      0  1 -err.-
##   0      82 18     18
##   1      20 33     20
##   -err.- 20 18     38
```

```
mlr::performance(pred_stacked, measures = list(acc, logloss))
```

```
##       acc   logloss
## 0.7516340 0.4539501
```

3.3 Tune Hyper Parameters for GBM and Random Forest Models

Because the accuracy values of the previous models are not satisfactory, I will try to improve the performance of some models by tuning their hyper parameters.

3.3.1 Tune GBM model

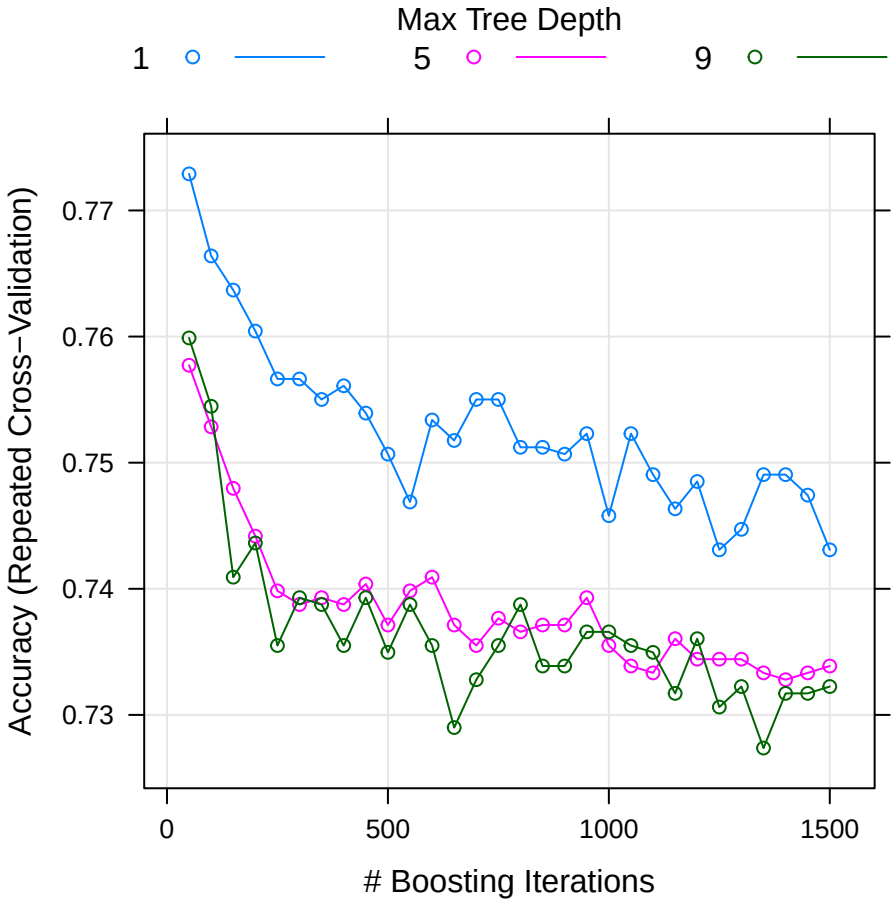
Let's start with **GBM** model.

```
gbmGrid <-  expand.grid(interaction.depth = c(1, 5, 9),
                        n.trees = (1:30)*50,
                        shrinkage = 0.1,
                        n.minobsinnode = 20)

fitControl <- trainControl(method = "repeatedcv",
                           number = 5,
                           repeats = 3)
```

```
set.seed(42, sample.kind = "Rounding")

gbm_model_diabetes_grid <- caret::train(diabetes ~ .,
                                        data = train_set,
                                        method = "gbm",
                                        trControl = fitControl,
                                        verbose = FALSE,
                                        tuneGrid = gbmGrid)
```

The following chart demonstrates the tuning process.

```
plot(gbm_model_diabetes_grid)
```



Let's evaluate the model with testing dataset, and calculate the accuracy.

```
y_hat_by_glm_cv <- predict(gbm_model_diabetes_grid, test_set)
tuned_gbm_acc <- confusionMatrix(y_hat_by_glm_cv, reference = test_set$diabetes)$overall["Accuracy"]
```

Accuracy of tuned gbm model is 0.7712418.

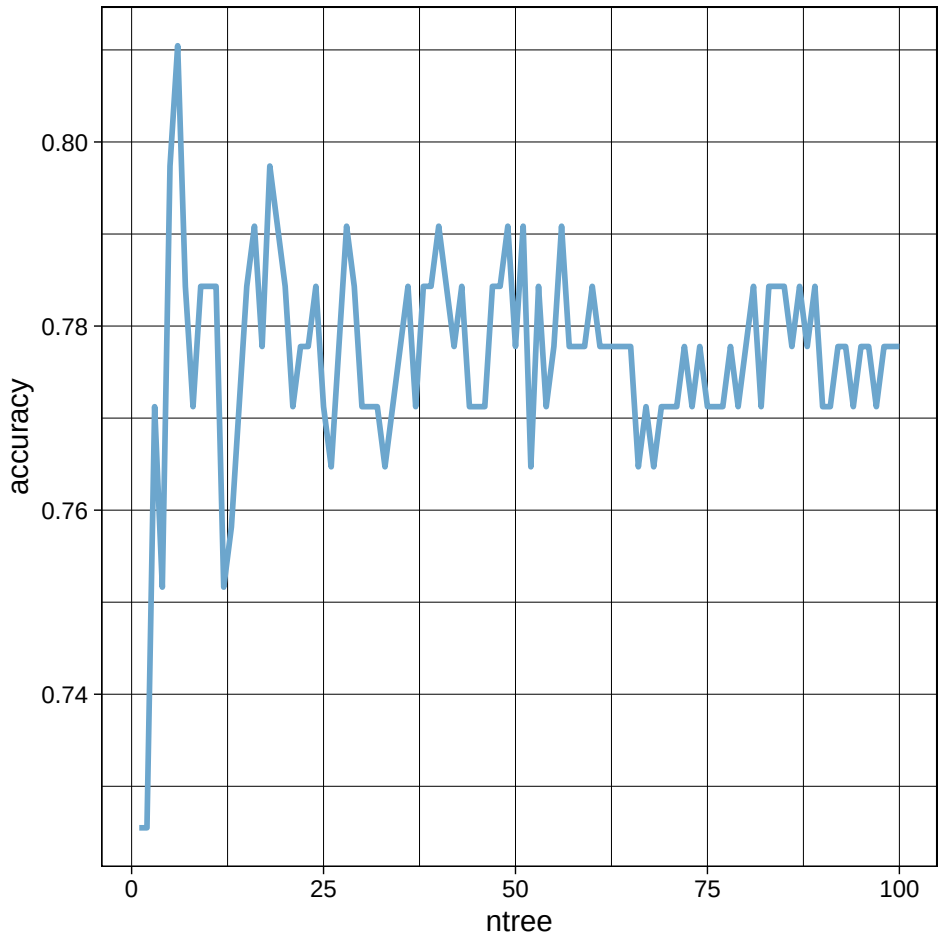
3.3.2 Use Manual Search to Tune Random Forest Model

```
control <- trainControl(method="repeatedcv", number=5, repeats=3, search="grid")
tunegrid <- expand.grid(.mtry=c(sqrt(ncol(train_set))))
```

```
modellist <- list()
acc_List <- list()
metric <- "Accuracy"

for (ntree in c(1:100)) {
  set.seed(7, sample.kind = "Rounding")
  fit <- caret::train(diabetes ~ ., data=train_set, method="rf", metric=metric, tuneGrid=tunegrid, trCo
  y_hat <- predict(fit, test_set)
  acc <- confusionMatrix(y_hat, reference = test_set$diabetes)$overall["Accuracy"]
  key <- toString(ntree)
  modellist[[key]] <- fit
  acc_List <- c(acc_List, acc)
}
temp_acc_df <- as_vector(acc_List)
acc_df <- data.frame("ntree" = c(1:100),
                     "accuracy" = temp_acc_df)
best_acc <- max(acc_df$accuracy)
best_acc
best_acc <- max(acc_df$accuracy)
best_ntree <- acc_df[which.max(acc_df$accuracy),"ntree"]
```

Let's plot the tuning process to visulize how the number of trees impact on the performance of Random Forest model.

```
acc_df %>%
  ggplot(aes(x = ntree, y = accuracy)) +
  geom_line(color = "skyblue3", size = 1 ) +
  theme_linedraw()
```



The final mdoel is the tuned **Randome Forest** model with ntree = 6, which can give us the accuracy equal to 0.8104575.

4 Conclusion

Through this project, I have demonstrated my data science knowledge and skills, learned throughout the **HarvardX Professional Certificate in Data Science** courses, in the steps of data anquisition, data wrangling, data analysis, data visualization, and machine learning.

After many iterations of building machine learning models, a tuned **Random Forest** model is elected as the final model, which can provide us accuracy equal to **0.8104575**.

Due to a lot of missing values in two features, which should have important influence on the outcome class, I have to drop them from dataset before feed it to build machine learing models. Further research should be conducted to find the best way to handle missing values for those two features, so they can be used to train models. If the missing values can be taclked correctly, the accuracy of the model may be better.

Also, there are many outliers in features such as Diastolic Blood Pressure, and BMI. Because of the small size of the data, every piece of data is precious, I do not exclude those outliers. If the outliers are treated properly, the performance can be improved.

For production, I shall deploy the final predictive model as an inference service, for example, a REST API service. A GUI application, like a mobile app, shall be created to interact with the inference servce, for users to check how likely he/she can have diabetes based on his/her health parameters.