

```
suppressMessages(library(tidyverse))
suppressMessages(library(pROC))
suppressMessages(library(caret))
```

```
# Set seed for reproducibility
set.seed(1123)
```

```
setwd("~/Downloads/UM - Fall 22/STATS 504/HW5")
test <- read.csv("test_df.csv")
train <- read.csv("train_df.csv")
drop <- c("X")
train = train[,!(names(train) %in% drop)]
test = test[,!(names(test) %in% drop)]
```

Logistic Regression:

```
fullmod <- glm(dv.hypertension1 ~ ., train, family = binomial)
summary(fullmod)
```

```
##
## Call:
## glm(formula = dv.hypertension1 ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.4780  -0.2401  -0.1581  -0.1075   3.6277
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.452e+01  1.365e+00 -10.638 < 2e-16 ***
## age           6.253e-02  1.678e-02   3.727 0.000194 ***
## racehispanic  -8.583e-01  3.140e-01  -2.733 0.006274 **
## racenative    -1.397e+00  1.033e+00  -1.352 0.176382
## raceother     -1.013e+00  2.956e-01  -3.427 0.000609 ***
## racewhite     -7.234e-01  2.301e-01  -3.144 0.001668 **
## emosupportTRUE  1.884e-01  5.658e-01   0.333 0.739118
## financialsupportTRUE -1.501e-01  3.632e-01  -0.413 0.679335
## prenatalsupportTRUE -2.869e-01  2.959e-01  -0.970 0.332171
## deliverysupportTRUE -5.727e-02  5.930e-01  -0.097 0.923067
## psstotal      -1.895e-02  2.416e-02  -0.784 0.432930
## anxtotal       9.532e-04  1.261e-02   0.076 0.939732
## worryfambaby   8.660e-02  7.851e-02   1.103 0.270032
## exerciseTRUE   7.940e-02  1.825e-01   0.435 0.663523
## systolic       6.043e-02  8.419e-03   7.178 7.08e-13 ***
## diastolic      4.321e-02  1.036e-02   4.173 3.01e-05 ***
## worryhealthcare -5.098e-02  9.171e-02  -0.556 0.578334
## worrysymptoms  5.357e-02  4.309e-02   1.243 0.213802
## ssqmean       -2.937e-02  6.878e-02  -0.427 0.669372
## prepreglbs     6.418e-03  1.660e-03   3.867 0.000110 ***
## familypreeclampsia -1.055e-01  1.433e-01  -0.736 0.461704
## income        -1.740e-02  2.341e-02  -0.743 0.457231
## kidney1TRUE    1.097e+00  4.317e-01   2.541 0.011059 *
```

```
## lupus1TRUE          1.355e+00  1.092e+00  1.241 0.214638
## collagen1TRUE       -8.619e-01  7.532e-01 -1.144 0.252513
## crohns1TRUE          8.775e-01  6.484e-01  1.353 0.175919
## pcos1TRUE            5.615e-01  2.762e-01  2.033 0.042074 *
## discrimination       8.732e-02  5.701e-02  1.532 0.125580
## bornearly           -2.529e-01  1.350e-01 -1.873 0.061059 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1609.0 on 5553 degrees of freedom
## Residual deviance: 1270.9 on 5525 degrees of freedom
## AIC: 1328.9
##
## Number of Fisher Scoring iterations: 7
```

```
backwards = step(fullmod, trace = 0)
summary(backwards)
```

```
##
## Call:
## glm(formula = dv.hypertension1 ~ age + race + systolic + diastolic +
##   worrysymptoms + prepreglbs + kidney1 + pcos1 + discrimination +
##   bornearly, family = binomial, data = train)
##
## Deviance Residuals:
##   Min       1Q   Median       3Q      Max
## -1.4357  -0.2412  -0.1620  -0.1088   3.5674
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -15.278740   1.056563 -14.461 < 2e-16 ***
## age           0.055829   0.014501   3.850 0.000118 ***
## racehispanic  -0.896660   0.306258  -2.928 0.003414 **
## racenative    -1.551804   1.029600  -1.507 0.131762
## raceother     -1.104829   0.275278  -4.014 5.98e-05 ***
## racewhite     -0.837490   0.209449  -3.999 6.37e-05 ***
## systolic       0.061305   0.008375   7.320 2.49e-13 ***
## diastolic      0.040909   0.010341   3.956 7.62e-05 ***
## worrysymptoms  0.063910   0.036076   1.772 0.076476 .
## prepreglbs     0.006513   0.001628   4.001 6.30e-05 ***
## kidney1TRUE    1.099539   0.428573   2.566 0.010300 *
## pcos1TRUE      0.567517   0.273550   2.075 0.038020 *
## discrimination 0.090650   0.056509   1.604 0.108679
## bornearly     -0.285002   0.130037  -2.192 0.028401 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1609.0 on 5553 degrees of freedom
## Residual deviance: 1280.5 on 5540 degrees of freedom
## AIC: 1308.5
```

```
##
## Number of Fisher Scoring iterations: 7

nothing <- glm(dv.hypertension1 ~ 1, train, family = binomial)
forwards = step(nothing, trace = 0,
                scope=list(lower=formula(nothing),upper=formula(fullmod)),
                direction="forward")
summary(forwards)
```

```
##
## Call:
## glm(formula = dv.hypertension1 ~ systolic + prepreglbs + diastolic +
##      race + age + kidney1 + bornearly + pcos1 + worryfambaby +
##      discrimination, family = binomial, data = train)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.4717  -0.2413  -0.1614  -0.1088   3.5732
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -15.216202   1.044475 -14.568 < 2e-16 ***
## systolic      0.061229   0.008384   7.303 2.82e-13 ***
## prepreglbs    0.006651   0.001626   4.091 4.30e-05 ***
## diastolic     0.041355   0.010306   4.013 6.00e-05 ***
## racehispanic  -0.907287   0.306433  -2.961 0.003068 **
## racenative    -1.565584   1.030295  -1.520 0.128624
## raceother     -1.127447   0.276369  -4.079 4.51e-05 ***
## racewhite     -0.853459   0.209441  -4.075 4.60e-05 ***
## age           0.055162   0.014440   3.820 0.000133 ***
## kidney1TRUE   1.105035   0.426701   2.590 0.009606 **
## bornearly     -0.288030   0.129970  -2.216 0.026682 *
## pcos1TRUE     0.563157   0.273768   2.057 0.039680 *
## worryfambaby  0.111072   0.062944   1.765 0.077630 .
## discrimination 0.086584   0.056691   1.527 0.126687
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 1609.0  on 5553  degrees of freedom
## Residual deviance: 1280.5  on 5540  degrees of freedom
## AIC: 1308.5
##
## Number of Fisher Scoring iterations: 7
```

```
# note: Backwards model has one extra variable.
```

```
predLOG <- predict(fullmod, test, type = "response")
predtrainLOG <- predict(fullmod, train, type = "response")
predLOG = as.numeric(predLOG >= 0.5)
predtrainLOG = as.numeric(predtrainLOG >= 0.5)
truthTest <- ifelse(test$dv.hypertension1 == "TRUE", 1, 0)
truthTrain <- ifelse(train$dv.hypertension1 == "TRUE", 1, 0)
```

```
table(predicted = predLOG, actual=truthTest)
```

```
##          actual
## predicted    0    1
##          0 2303   69
##          1     6    2
```

```
testErrorLOG <- mean(predLOG!=truthTest)
testErrorLOG
```

```
## [1] 0.03151261
```

```
table(predicted = predtrainLOG, actual = truthTrain)
```

```
##          actual
## predicted    0    1
##          0 5365  165
##          1     6   18
```

```
trainErrorLOG <- mean(predtrainLOG != truthTrain)
trainErrorLOG
```

```
## [1] 0.03078862
```

Backwards Model:

```
predLOG.b <- predict(backwards, test, type = "response")
predtrainLOG.b <- predict(backwards, train, type = "response")
predLOG.b = as.numeric(predLOG.b >= 0.5)
predtrainLOG.b = as.numeric(predtrainLOG.b >= 0.5)
truthTest.b <- ifelse(test$dv.hypertension1 == "TRUE", 1, 0)
truthTrain.b <- ifelse(train$dv.hypertension1 == "TRUE", 1, 0)
```

```
table(predicted = predLOG.b, actual=truthTest.b)
```

```
##          actual
## predicted    0    1
##          0 2302   69
##          1     7    2
```

```
testErrorLOG.b <- mean(predLOG.b!=truthTest.b)
testErrorLOG.b
```

```
## [1] 0.03193277
```

```
table(predicted = predtrainLOG.b, actual = truthTrain.b)
```

```
##          actual
## predicted    0    1
##          0 5365  168
##          1     6   15
```

```
trainErrorLOG.b <- mean(predtrainLOG.b != truthTrain.b)
trainErrorLOG.b
```

```
## [1] 0.03132877
```

Forwards Model:

```
predLOG.f <- predict(forwards, test, type = "response")
predtrainLOG.f <- predict(forwards, train, type = "response")
predLOG.f = as.numeric(predLOG.f >= 0.5)
predtrainLOG.f = as.numeric(predtrainLOG.f >= 0.5)
truthTest.f <- ifelse(test$dv.hypertension1 == "TRUE", 1, 0)
truthTrain.f <- ifelse(train$dv.hypertension1 == "TRUE", 1, 0)
```

```
table(predicted = predLOG.f, actual=truthTest.f)
```

```
##          actual
## predicted    0    1
##          0 2302   69
##          1     7    2
```

```
testErrorLOG.f <- mean(predLOG.f!=truthTest.f)
testErrorLOG.f
```

```
## [1] 0.03193277
```

```
table(predicted = predtrainLOG.f, actual = truthTrain.f)
```

```
##          actual
## predicted    0    1
##          0 5366  167
##          1     5   16
```

```
trainErrorLOG.f <- mean(predtrainLOG.f != truthTrain.f)
trainErrorLOG.f
```

```
## [1] 0.03096867
```

All testing errors:

```
testErrorLOG
```

```
## [1] 0.03151261
```

```
# AIC full mod: 1328.863
fullmod$aic
```

```
## [1] 1328.863
```

```
# AIC backwards model: 1308.461
backwards$aic
```

```
## [1] 1308.461
```

```
testErrorLOG.b
```

```
## [1] 0.03193277
```

```
# AIC forwards model: 1308.489
forwards$aic
```

```
## [1] 1308.489
```

```
testErrorLOG.f
```

```
## [1] 0.03193277
```

Test Errors very similar. Going to look at lower AIC.

```
summary <- summary(backwards)
exp(summary$coefficients[,1])
```

```
##      (Intercept)          age  racehispanic    racenative    raceother
## 2.314876e-07  1.057417e+00  4.079297e-01  2.118655e-01  3.312675e-01
##      racewhite    systolic    diastolic  worrysymptoms  prepreglbs
## 4.327954e-01  1.063223e+00  1.041758e+00  1.065996e+00  1.006534e+00
##      kidney1TRUE  pcos1TRUE  discrimination  bornearly
## 3.002782e+00  1.763882e+00  1.094886e+00  7.520128e-01
```

Confusion Matrices:

```
print("Backwards model: ")
```

```
## [1] "Backwards model: "
```

```
table(predicted = predLOG.b, actual=truthTest.b)
```

```
##      actual
## predicted  0    1
##      0 2302  69
##      1    7    2
```

```
testErrorLOG.b <- mean(predLOG.b!=truthTest.b)
testErrorLOG.b
```

```
## [1] 0.03193277
```

```
print("Full model: ")
```

```
## [1] "Full model: "
```

```
table(predicted = predLOG, actual=truthTest)
```

```
##          actual
## predicted    0    1
##          0 2303   69
##          1    6    2
```

```
testErrorLOG <- mean(predLOG!=truthTest)
testErrorLOG
```

```
## [1] 0.03151261
```

```
print("Forwards model: ")
```

```
## [1] "Forwards model: "
```

```
table(predicted = predLOG.f, actual=truthTest.f)
```

```
##          actual
## predicted    0    1
##          0 2302   69
##          1    7    2
```

```
testErrorLOG.f <- mean(predLOG.f!=truthTest.f)
testErrorLOG.f
```

```
## [1] 0.03193277
```

AUC:

```
auc(test$dv.hypertension1, predLOG.f)
```

```
## Setting levels: control = FALSE, case = TRUE
```

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

```
## Area under the curve: 0.5126
```

```
auc(test$dv.hypertension1, predLOG.b)
```

```
## Setting levels: control = FALSE, case = TRUE
```

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

```
## Area under the curve: 0.5126
```

```
auc(test$dv.hypertension1, predLOG)
```

```
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```

```
## Area under the curve: 0.5128
```

```
test$dv.hypertension1 <- ifelse(test$dv.hypertension1 == "TRUE", 1, 0)
conf_mat = table("truth" = test$dv.hypertension1, "pred" = predLOG)
conf_mat = confusionMatrix(conf_mat, mode = "everything", positive = "1")
conf_mat$byClass
```

##	Sensitivity	Specificity	Pos Pred Value
##	0.2500000000	0.9709106239	0.0281690141
##	Neg Pred Value	Precision	Recall
##	0.9974014725	0.0281690141	0.2500000000
##	F1	Prevalence	Detection Rate
##	0.0506329114	0.0033613445	0.0008403361
##	Detection Prevalence	Balanced Accuracy	
##	0.0298319328	0.6104553120	

```
conf_mat
```

```
## Confusion Matrix and Statistics
```

```
##
```

```
##      pred
```

```
## truth  0    1
```

```
##      0 2303    6
```

```
##      1    69    2
```

```
##
```

```
##              Accuracy : 0.9685
```

```
##              95% CI : (0.9607, 0.9751)
```

```
##      No Information Rate : 0.9966
```

```
##      P-Value [Acc > NIR] : 1
```

```
##
```

```
##              Kappa : 0.0449
```

```
##
```

```
##      McNemar's Test P-Value : 8.118e-13
```

```
##
```

```
##              Sensitivity : 0.2500000
```

```
##              Specificity : 0.9709106
```

```
##              Pos Pred Value : 0.0281690
```

```
##              Neg Pred Value : 0.9974015
```

```
##              Precision : 0.0281690
```

```
##              Recall : 0.2500000
```

```
##              F1 : 0.0506329
```

```
##              Prevalence : 0.0033613
```

```
##              Detection Rate : 0.0008403
```

```
##      Detection Prevalence : 0.0298319
```

```
##      Balanced Accuracy : 0.6104553
```

```
##
```

```
##      'Positive' Class : 1
```

```
##
```



```
conf_mat.b = table("truth" = test$dv.hypertension1, "pred" = predLOG.b)
conf_mat.b = confusionMatrix(conf_mat.b, mode = "everything", positive = "1")
conf_mat.b$byClass
```

```
##          Sensitivity          Specificity      Pos Pred Value
##          0.222222222          0.9708983551      0.0281690141
##      Neg Pred Value          Precision          Recall
##          0.9969683846          0.0281690141      0.2222222222
##              F1          Prevalence      Detection Rate
##          0.0500000000          0.0037815126      0.0008403361
## Detection Prevalence      Balanced Accuracy
##          0.0298319328          0.5965602887
```

```
conf_mat.b
```

```
## Confusion Matrix and Statistics
##
##      pred
## truth  0    1
##    0 2302    7
##    1    69    2
##
##              Accuracy : 0.9681
##              95% CI : (0.9602, 0.9748)
##      No Information Rate : 0.9962
##      P-Value [Acc > NIR] : 1
##
##              Kappa : 0.0436
##
##      McNemar's Test P-Value : 2.612e-12
##
##              Sensitivity : 0.2222222
##              Specificity : 0.9708984
##              Pos Pred Value : 0.0281690
##              Neg Pred Value : 0.9969684
##              Precision : 0.0281690
##              Recall : 0.2222222
##              F1 : 0.0500000
##              Prevalence : 0.0037815
##              Detection Rate : 0.0008403
##      Detection Prevalence : 0.0298319
##      Balanced Accuracy : 0.5965603
##
##      'Positive' Class : 1
##
```

Import

In [1]:

```
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None)
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import StratifiedKFold, cross_validate, learning_curve, RandomizedSearchCV, GridSearchCV, train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, plot_confusion_matrix, make_scorer, accuracy_score, auc, precision_recall_curve, average_precision_score
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import PowerTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from imblearn.over_sampling import SMOTE

import xgboost as xgb
import lightgbm as lgb

import os
import sys
```

Input

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
file_path = '/content/drive/MyDrive/stats504/'
```

Mounted at /content/drive

In [3]:

```
df = pd.read_csv(file_path+'nuMoM2bsubset.csv')
df.drop(columns=['dv.gestweeks', 'dv.v3epdstotal', 'dv.preeclampsia', 'dv.diabetes1', 'v1epdstotal'], inplace=True)
# df.drop_duplicates(inplace=True)
df.head(2)
```

Out[3]:

	age	race	emosupport	financialsupport	prenatalsupport	deliverysupport	psstotal	anxtotal	worryfambaby	exercise	sys
0	31.0	white	1.0	1.0	1.0	1.0	26	36.0	5.0	1.0	1
1	26.0	black	1.0	1.0	1.0	1.0	30	34.0	5.0	2.0	1



Data preprocessing

In [5]:

```
In [5]:
```

```
def data_preporcess(df, cat):
    y = df.loc[:, 'dv.hypertension1']
    x = df.drop(columns='dv.hypertension1')
    print('Total shape: ', x.shape, y.shape)

    # x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=test_size, random_state=random_state, shuffle=shuffle)

    train_df = pd.read_csv(file_path+'train_df.csv', index_col=[0])
    test_df = pd.read_csv(file_path+'test_df.csv', index_col=[0])

    train_df = train_df.drop_duplicates()
    train_df[["age", "income", "prepreglbs"]] = train_df[["age", "income", "prepreglbs"]].replace({'0': np.nan, 0: np.nan})

    y_train = train_df.loc[:, 'dv.hypertension1']
    y_test = test_df.loc[:, 'dv.hypertension1']

    if cat==True:
        x_train = train_df.drop(columns=['dv.hypertension1'])
        x_test = test_df.drop(columns=['dv.hypertension1'])
        x_train = pd.get_dummies(x_train)
        x_test = pd.get_dummies(x_test)
    else:
        x_train = train_df.drop(columns=['dv.hypertension1', 'race'])
        x_test = test_df.drop(columns=['dv.hypertension1', 'race'])

    print('Train shape: ', x_train.shape, y_train.shape)
    print('Test shape: ', x_test.shape, y_test.shape)

    return x, y, x_train, y_train, x_test, y_test
```

Imbalance Data

```
In [6]:
```

```
def smote_balance(x_train, y_train, r, k):
    oversample = SMOTE(r, k_neighbors=k)
    print(f'Shape of the training before SMOTE: {x_train.shape, y_train.shape}')

    x_tr_resample, y_tr_resample = oversample.fit_resample(x_train, y_train)
    print(f'Shape of the training after SMOTE: {x_tr_resample.shape, y_tr_resample.shape}')

    # Target distribution before SMOTE
    non_fraud = 0
    fraud = 0

    for i in y_train:
        if i == 0:
            non_fraud +=1
        else:
            fraud +=1

    # Target distribution after SMOTE
    no = 0
    yes = 1

    for j in y_tr_resample:
        if j == 0:
            no +=1
        else:
            yes +=1

    print(f'BEFORE OVERSAMPLING \n \tNon-frauds: {non_fraud} \n \tFauds: {fraud}')
    print(f'AFTER OVERSAMPLING \n \tNon-frauds: {no} \n \tFauds: {yes}')
```

```
return x_tr_resample, y_tr_resample
```

Model selection

In [7]:

```
def evaluate_models(X, y, models, cv):
    f1_scores = dict()
    roc_auc_scores = dict()
    acc_scores = dict()

    for i, model in enumerate(models):
        clf_pipeline = make_pipeline(preprocessing_pipeline, model)
        # clf_pipeline = make_pipeline(model)
        results = cross_validate(clf_pipeline, X, y, cv=cv, scoring=['f1', 'accuracy', '
roc_auc'], n_jobs=-1)
        avg_f1 = np.mean(results['test_f1'])
        avg_acc = np.mean(results['test_accuracy'])
        avg_roc = np.mean(results['test_roc_auc'])

        model_name = model.__class__.__name__
        f1_scores[model_name] = avg_f1
        acc_scores[model_name] = avg_acc
        roc_auc_scores[model_name] = avg_roc
        print('{}-of-{}: {} f1={}, acc={}, roc_auc={}'.format(i+1, len(models), model_na
me, avg_f1, avg_acc, avg_roc))
    return f1_scores, acc_scores, roc_auc_scores

def visualize_scores(f1_scores, acc_scores, roc_auc_scores):
    x = np.arange(len(f1_scores))
    width = 0.3

    f1_values = list(f1_scores.values())
    acc_values = list(acc_scores.values())
    roc_values = list(roc_auc_scores.values())

    plt.figure(figsize=(20, 8)).tight_layout()
    plt.bar(x - width, f1_values, width, label='f1 score')
    plt.bar(x, acc_values, width, label='accuracy')
    plt.bar(x + width, roc_values, width, label='roc_auc')

    for index, value in enumerate(x - width / 2):
        plt.text(value, f1_values[index], '{:.3}'.format(f1_values[index]),
                 verticalalignment='bottom', horizontalalignment='center', fontsize=10)

    for index, value in enumerate(x + width / 2):
        plt.text(value, acc_values[index], '{:.3}'.format(acc_values[index]),
                 verticalalignment='bottom', horizontalalignment='center', fontsize=10)

    for index, value in enumerate(x + width / 2):
        plt.text(value, roc_values[index], '{:.3}'.format(roc_values[index]),
                 verticalalignment='bottom', horizontalalignment='center', fontsize=10)

    classifiers_names = f1_scores.keys()
    plt.xticks(x, classifiers_names, rotation=40, horizontalalignment='right', fontsize=
10)
    plt.legend()

def model_select(X, y, models, cv):
    f1_scores, acc_scores, roc_auc_scores = evaluate_models(X, y, models, cv)
    visualize_scores(f1_scores, acc_scores, roc_auc_scores)
```

Implementation

Without smote

Best model on MLP

In [30]:

```
preprocessing_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('nomalize', MinMaxScaler())
    # ('standard', StandardScaler())
])
```

In [28]:

```
def best_model_select_MLP(x_train, y_train, x_test, y_test):
    MLP_parameters = {
        'mlpclassifier__hidden_layer_sizes': [2, 10, 2],
        'mlpclassifier__solver': ['sgd', 'adam'],
        'mlpclassifier__learning_rate': ['adaptive', 'constant'],
        'mlpclassifier__max_iter': [1000],
        'mlpclassifier__activation': ['logistic', 'tanh'],
        'mlpclassifier__alpha': [1e-5, 1e-4, 1e-3]
    }

    MLP_pipeline = make_pipeline(preprocessing_pipeline, MLPClassifier(random_state=42))

    MLP_grid_search = GridSearchCV(
        MLP_pipeline,
        param_grid=MLP_parameters,
        scoring = 'recall',
        n_jobs = -1,
        cv = 5
    )

    MLP_grid_search.fit(x_train, y_train)

    display(MLP_grid_search.best_score_)
    display(MLP_grid_search.best_params_)
    dict1 = MLP_grid_search.best_params_

    model_dict = {k.replace('mlpclassifier__', ''):v for k, v in dict1.items()}

    X_train = preprocessing_pipeline.fit_transform(x_train)
    X_test = preprocessing_pipeline.transform(x_test)

    best_model_MLP = MLPClassifier(**model_dict)

    best_model_MLP.fit(X_train, y_train)
    predictions = best_model_MLP.predict(X_test)

    precision, recall, _ = precision_recall_curve(y_test, predictions)
    auc_score = auc(recall, precision)

    print("f1 score = {0:.4f}".format(f1_score(y_test, predictions)))
    print("Precision score = {0:.4f}".format(precision_score(y_test, predictions)))
    print("Recall score = {0:.4f}".format(recall_score(y_test, predictions)))
    print("ROC AUC score = {0:.4f}".format(roc_auc_score(y_test, predictions)))
    print("PR AUC score = {0:.4f}".format(auc_score))
    print("accuracy score = {0:.4f}".format(accuracy_score(y_test, predictions)))

    display(plot_confusion_matrix(best_model_MLP, X_test, y_test))

    return best_model_MLP
```

In [31]:

```
x, y, x_train, y_train, x_test, y_test = data_preporcess(df, cat=True)
best_model_MLP = best_model_select_MLP(x_train, y_train, x_test, y_test)
```

Total shape: (7626, 25) (7626,)

```
Train shape: (5516, 29) (5516,)
Test shape: (2380, 29) (2380,)
```

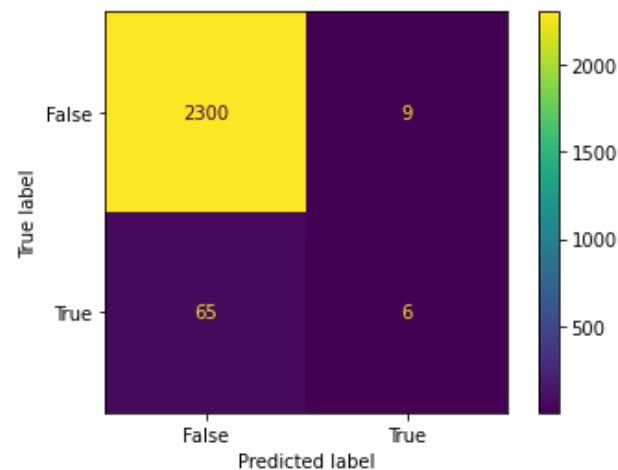
```
0.10111111111111111
```

```
{'mlpclassifier__solver': 'adam',
 'mlpclassifier__max_iter': 1000,
 'mlpclassifier__learning_rate': 'constant',
 'mlpclassifier__hidden_layer_sizes': 10,
 'mlpclassifier__activation': 'tanh'}
```

```
f1 score = 0.1395
Precision score = 0.4000
Recall score = 0.0845
ROC AUC score = 0.5403
PR AUC score = 0.2559
accuracy score = 0.9689
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f261521c0d0>
```



With smote

```
In [10]:
```

```
preprocessing_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('nomalize', MinMaxScaler())
    # ('standard', StandardScaler())
])
```

Best model on MLP

```
In [14]:
```

```
def best_model_select_MLP(x_train, y_train, x_test, y_test):
    MLP_parameters = {
        'mlpclassifier__hidden_layer_sizes': [2, 10, 2],
        'mlpclassifier__solver': ['sgd', 'adam'],
        'mlpclassifier__learning_rate': ['adaptive', 'constant'],
        'mlpclassifier__max_iter': [1000],
        'mlpclassifier__activation': ['logistic', 'tanh'],
        'mlpclassifier__alpha': [1e-5, 1e-4, 1e-3]
    }

    MLP_pipeline = make_pipeline(preprocessing_pipeline, MLPClassifier(random_state=42))

    MLP_grid_search = GridSearchCV(
        MLP_pipeline,
```

```

        param_grid=MLP_parameters,
        scoring = 'recall',
        n_jobs = -1,
        cv = 5
    )

MLP_grid_search.fit(x_train, y_train)

display(MLP_grid_search.best_score_)
display(MLP_grid_search.best_params_)
dict1 = MLP_grid_search.best_params_

model_dict = {k.replace('mlpclassifier__', ''):v for k, v in dict1.items()}

X_train = preprocessing_pipeline.fit_transform(x_train)
X_test = preprocessing_pipeline.transform(x_test)

best_model_MLP = MLPClassifier(**model_dict)

best_model_MLP.fit(X_train, y_train)
predictions = best_model_MLP.predict(X_test)

precision, recall, _ = precision_recall_curve(y_test, predictions)
auc_score = auc(recall, precision)

print("f1 score = {0:.4f}".format(f1_score(y_test, predictions)))
print("Precision score = {0:.4f}".format(precision_score(y_test, predictions)))
print("Recall score = {0:.4f}".format(recall_score(y_test, predictions)))
print("ROC AUC score = {0:.4f}".format(roc_auc_score(y_test, predictions)))
print("PR AUC score = {0:.4f}".format(auc_score))
print("accuracy score = {0:.4f}".format(accuracy_score(y_test, predictions)))

display(plot_confusion_matrix(best_model_MLP, X_test, y_test))

return best_model_MLP

```

In [19]:

```

x, y, x_train, y_train, x_test, y_test = data_preporcess(df, cat=True)
x_train_balance, y_train_balance = smote_balance(x_train, y_train, r=1.0, k=10)
best_model_MLP = best_model_select_MLP(x_train_balance, y_train_balance, x_test, y_test)

```

```

Total shape: (7626, 25) (7626,)
Train shape: (5516, 29) (5516,)
Test shape: (2380, 29) (2380,)
Shape of the training before SMOTE: ((5516, 29), (5516,))
Shape of the training after SMOTE: ((10678, 29), (10678,))
BEFORE OVERSAMPLING
  Non-frauds: 5339
  Fauds: 177
AFTER OVERSAMPLING
  Non-frauds: 5339
  Fauds: 5340

```

```

/usr/local/lib/python3.7/dist-packages/imblearn/utils/_validation.py:591: FutureWarning:
Pass sampling_strategy=1.0 as keyword args. From version 0.9 passing these as positional
arguments will result in an error
  FutureWarning,

```

0.8615852138903222

```

{'mlpclassifier__activation': 'logistic',
 'mlpclassifier__alpha': 0.0001,
 'mlpclassifier__hidden_layer_sizes': 6,
 'mlpclassifier__learning_rate': 'adaptive',
 'mlpclassifier__max_iter': 1000,
 'mlpclassifier__solver': 'adam'}

```

```

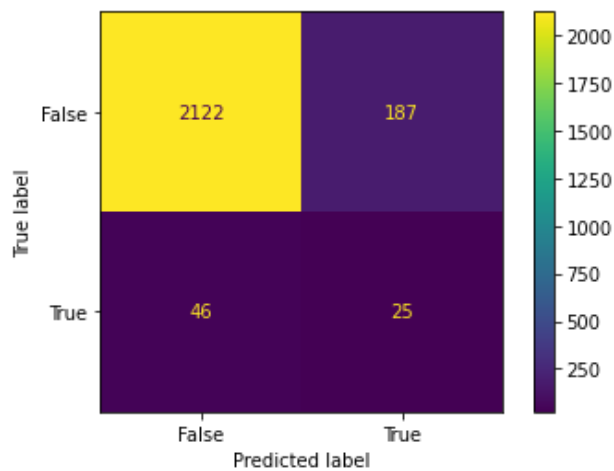
f1 score = 0.1767
Precision score = 0.1179
Recall score= 0.3521
ROC AUC score = 0.6356

```

PR AUC score = 0.2447
accuracy score = 0.9021

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.  
warnings.warn(msg, category=FutureWarning)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f2615191390>
```



appendix

11/9/2022

```
library(tidyverse)
library(class)
library(kknn)
library(ggplot2)
library(caret)
```

```
# can only use numerical ones in the knn model
train <- read.csv("train_df.csv") %>% select(-X, -race)
test  <- read.csv("test_df.csv") %>% select(-X, -race)
```

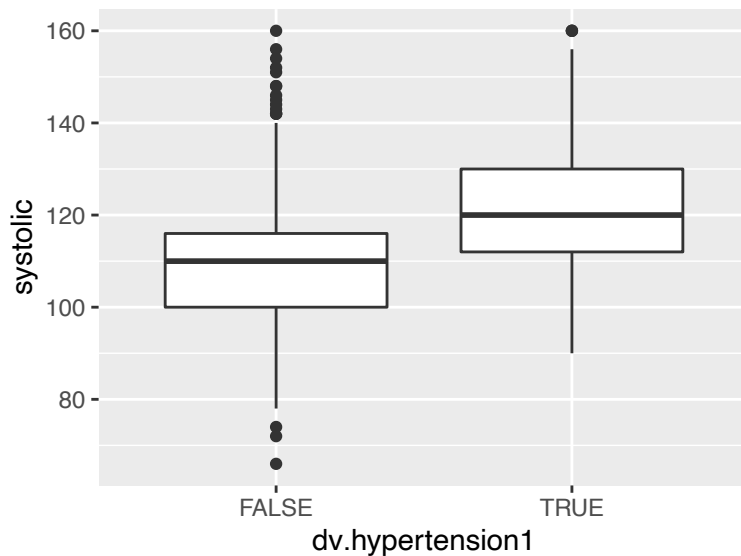
EDA

```
summary(train)
```

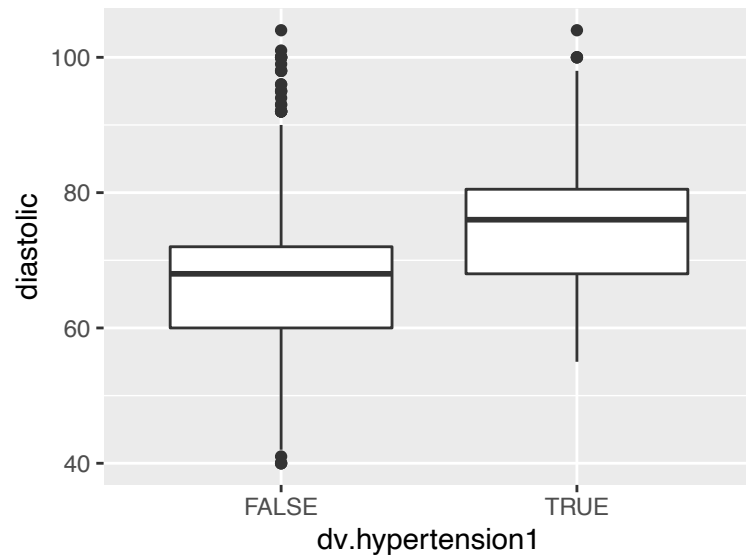
```
##      age      emosupport  financialsupport prenatalsupport
## Min.   : 0.00   Mode :logical   Mode :logical   Mode :logical
## 1st Qu.:23.00   FALSE:323   FALSE:546     FALSE:633
## Median :28.00   TRUE :5231   TRUE :5008     TRUE :4921
## Mean    :27.11
## 3rd Qu.:31.00
## Max.     :52.00
## deliverysupport  psstotal      anxtotal      worryfambaby  exercise
## Mode :logical   Min.   : 0.00   Min.   :20.00   Min.   :0.0   Mode :logical
## FALSE:327       1st Qu.:28.00   1st Qu.:30.00   1st Qu.:4.0   FALSE:1568
## TRUE :5227       Median :30.00   Median :34.00   Median :5.0   TRUE :3986
##                Mean    :29.74   Mean    :35.34   Mean    :4.7
##                3rd Qu.:32.00   3rd Qu.:40.00   3rd Qu.:5.0
##                Max.     :50.00   Max.     :72.00   Max.     :9.0
##      systolic      diastolic  worryhealthcare worrysymptoms
## Min.   : 66.0   Min.   : 40.0   Min.   :0.000   Min.   : 5.000
## 1st Qu.:100.0   1st Qu.: 60.0   1st Qu.:2.000   1st Qu.: 7.000
## Median :110.0   Median : 68.0   Median :2.000   Median : 9.000
## Mean    :109.2   Mean    : 67.2   Mean    :2.692   Mean    : 9.078
## 3rd Qu.:118.0   3rd Qu.: 72.0   3rd Qu.:3.000   3rd Qu.:10.000
## Max.     :160.0   Max.     :104.0   Max.     :6.000   Max.     :18.000
##      ssqmean      prepreglbs  familypreeclampsia  income
## Min.   :0.000   Min.   : 0.0   Min.   :1.000   Min.   : 0.000
## 1st Qu.:6.000   1st Qu.:125.0   1st Qu.:3.000   1st Qu.: 4.000
## Median :6.583   Median :140.0   Median :3.000   Median :10.000
## Mean    :6.198   Mean    :150.8   Mean    :2.786   Mean    : 7.899
## 3rd Qu.:7.000   3rd Qu.:168.0   3rd Qu.:3.000   3rd Qu.:12.000
```

```
## Max. :7.000 Max. :368.0 Max. :3.000 Max. :14.000
## dv.hypertension1 kidney1 lupus1 collagen1
## Mode :logical Mode :logical Mode :logical Mode :logical
## FALSE:5371 FALSE:5451 FALSE:5544 FALSE:5460
## TRUE :183 TRUE :103 TRUE :10 TRUE :94
##
##
##
## crohns1 pcos1 discrimination bornearly
## Mode :logical Mode :logical Min. : 0.000 Min. :1.000
## FALSE:5503 FALSE:5309 1st Qu.: 1.000 1st Qu.:3.000
## TRUE :51 TRUE :245 Median : 1.000 Median :3.000
##
## Mean : 1.626 Mean :2.796
##
## 3rd Qu.: 2.000 3rd Qu.:3.000
##
## Max. :11.000 Max. :3.000
```

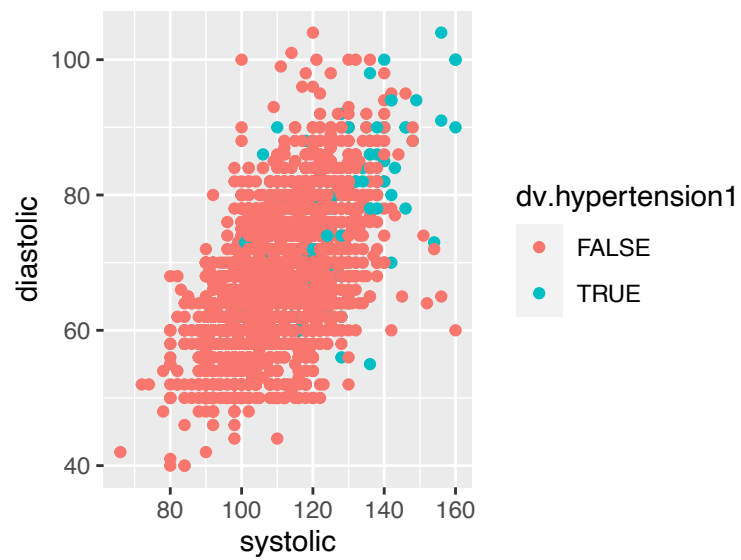
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = systolic))
```



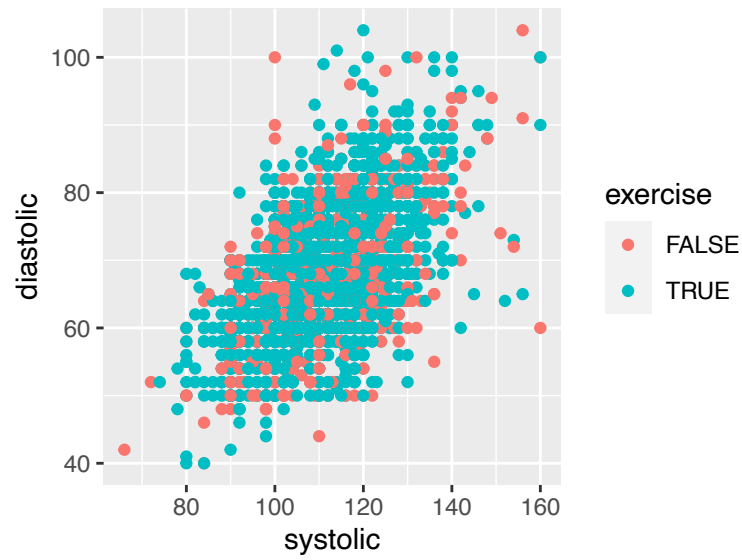
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = diastolic))
```



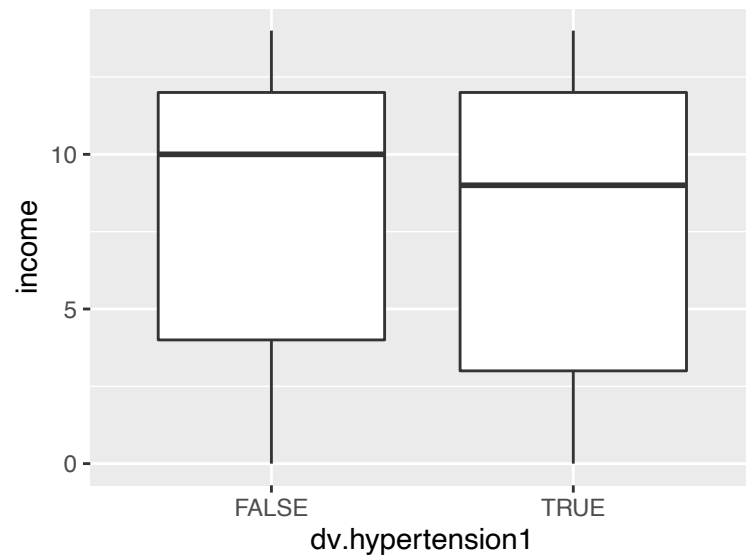
```
ggplot(data = train) + geom_point(mapping = aes(x = systolic, y = diastolic, color = dv.hypertension1))
```



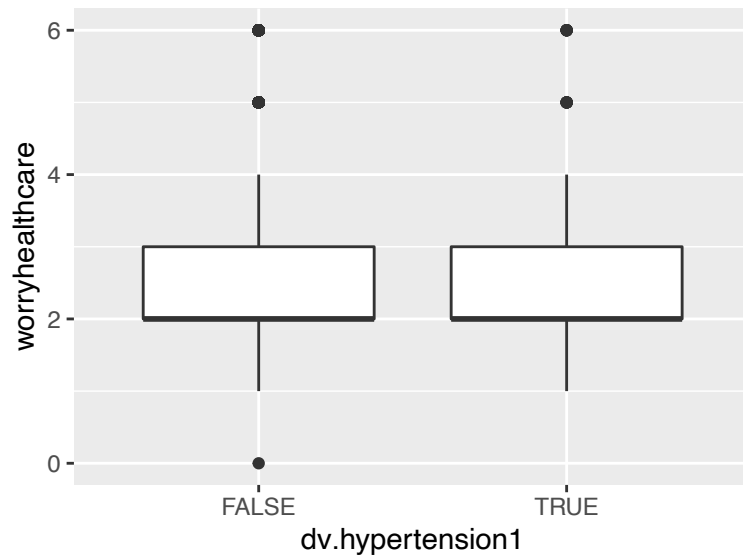
```
ggplot(data = train) + geom_point(mapping = aes(x = systolic, y = diastolic, color = exercise))
```



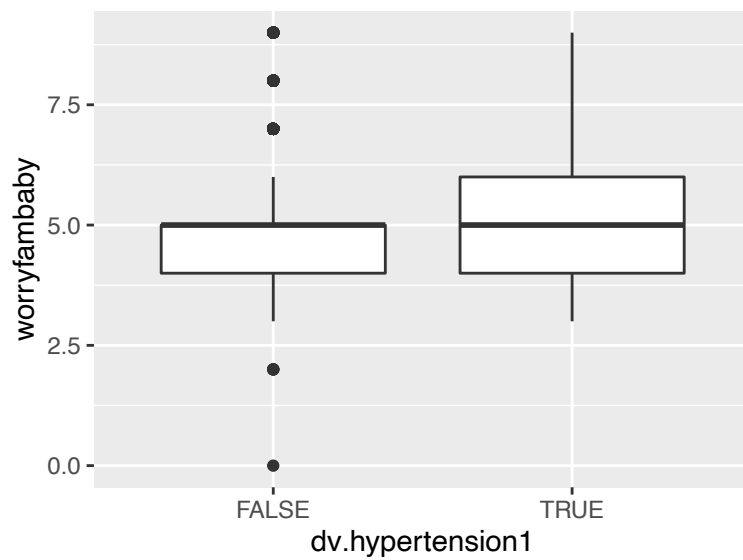
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = income))
```



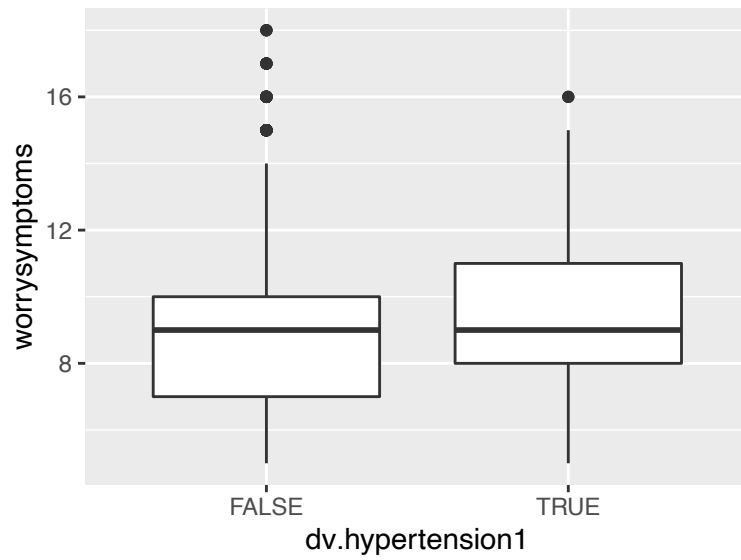
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = worryhealthcare))
```



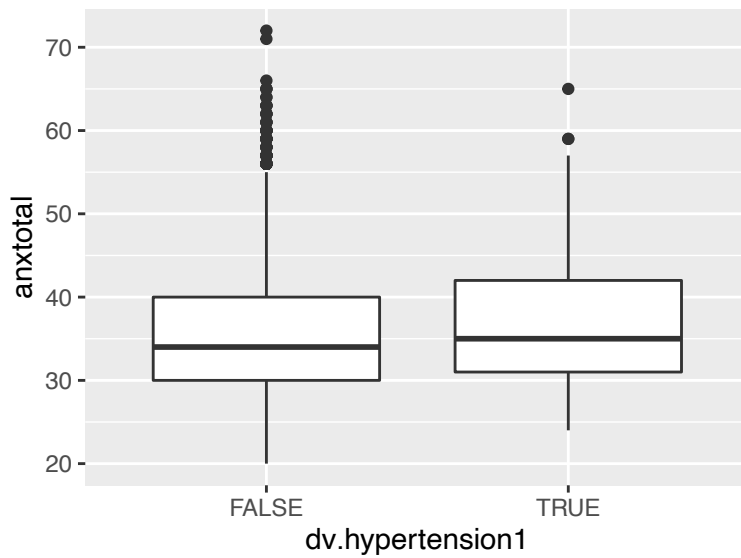
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = worryfambaby))
```



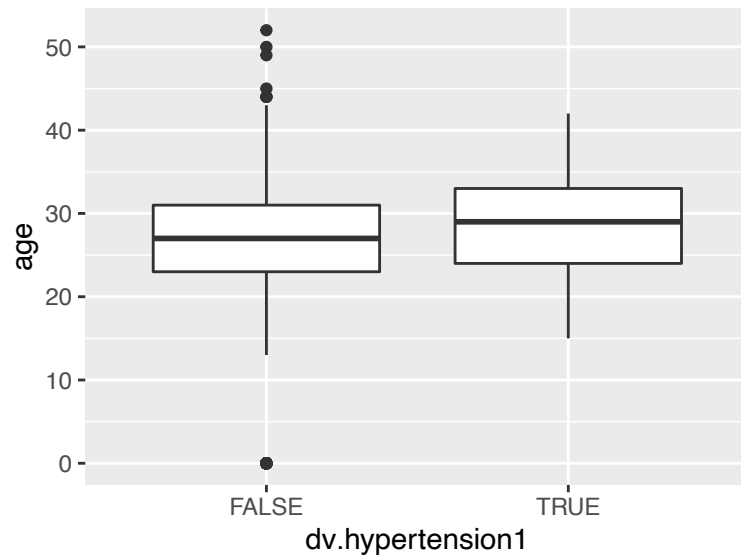
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = worrysymptoms))
```



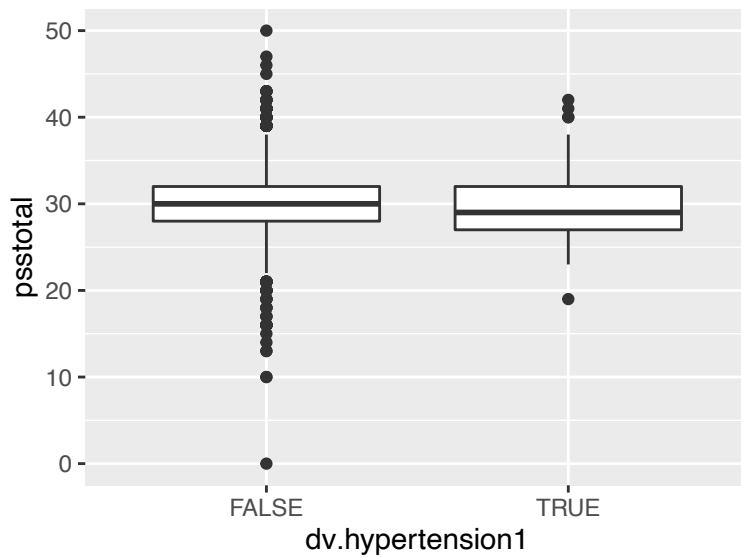
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = anxtotal))
```



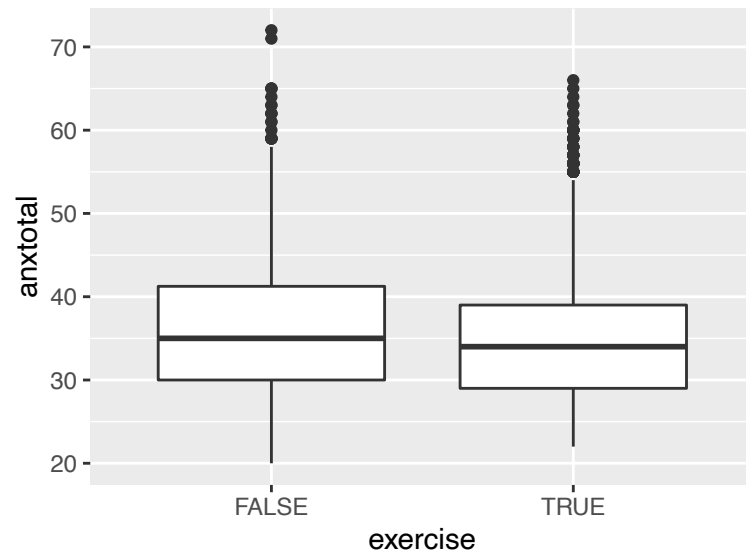
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = age))
```



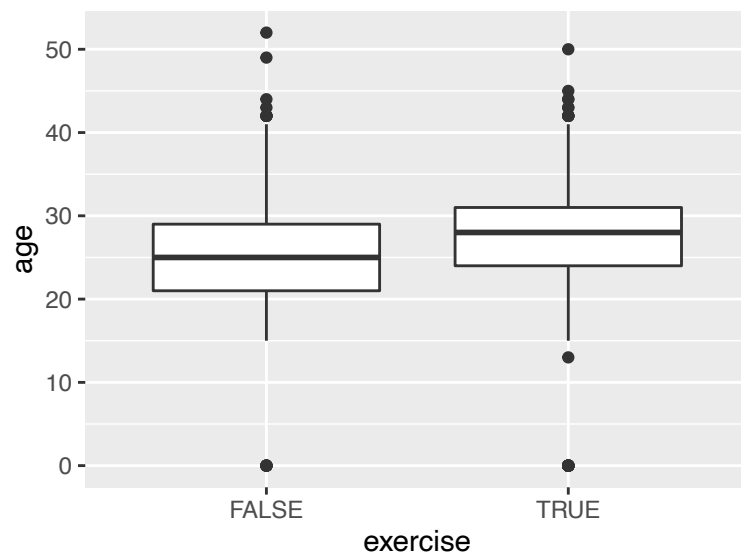
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = psstotal))
```



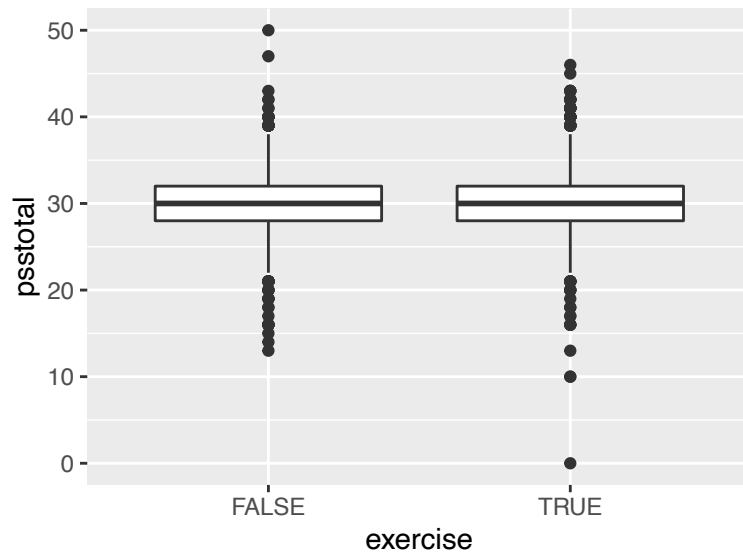
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = exercise, y = anxttotal))
```



```
ggplot(data = train) + geom_boxplot(mapping = aes(x = exercise, y = age))
```



```
ggplot(data = train) + geom_boxplot(mapping = aes(x = exercise, y = pssttotal))
```

scale the data for knn

```
train_x <- train %>% select(-dv.hypertension1)
train_label <- train %>% .$dv.hypertension1
test_x <- test %>% select(-dv.hypertension1)
test_label <- test %>% .$dv.hypertension1
```

```
mean_train = colMeans(train_x)
std_train = sqrt(diag(var(train_x)))
# training data
train_x = scale(train_x, center = mean_train, scale = std_train)
# test data
test_x = scale(test_x, center = mean_train, scale = std_train)
```

K-Fold CV

```
Kfold_CV_knn <- function(K,K_knn,train,train_label){
  fold_size <- floor(nrow(train)/K)
  cv_error <- rep(0,K)
  sensitives <- rep(0,K)
  for(i in 1:K){
    # select K-1 folds
    if(i!=K){
      CV_test_rows = ((i-1)*fold_size+1):(i*fold_size)
    }else{
      CV_test_rows = ((i-1)*fold_size+1):nrow(train)
    }
    CV_train <- train[-CV_test_rows,]
    CV_test <- train[CV_test_rows,]
```

```

# normalize training and testing using mean and sd
mean_CV_train <- colMeans(CV_train)
sd_CV_train <- apply(CV_train,2,sd)

CV_train <- scale(CV_train,center = mean_CV_train,scale = sd_CV_train)
CV_test <- scale(CV_test,center = mean_CV_train,scale = sd_CV_train)
# Fit
pred_CV_test <- knn(CV_train,CV_test,train_label[-CV_test_rows],k = K_knn)
# Calculate CV error
cv_error[i] <- mean(pred_CV_test!=train_label[CV_test_rows])
cm <- confusionMatrix(data = as.factor(pred_CV_test), reference = as.factor(train_label[CV_test_rows]),
                      positive = "TRUE")
sensitives[i] = cm$byClass["Sensitivity"]
}
senses[i] = mean(sensitives)
return(mean(cv_error))
}

```

```

K_fold <- 10
K_knn <- 1:50
cv_error <- rep(0,length(K_knn))
senses <- rep(0,length(K_knn))
for(i in 1:length(K_knn)){
  cv_error[i] <- Kfold_CV_knn(K = K_fold, K_knn = K_knn[i],train = train_x,train_label = train_label)
}

```

```
min(cv_error)
```

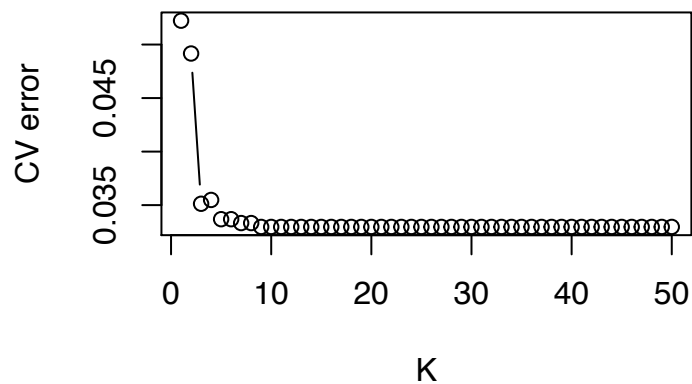
```
## [1] 0.03296266
```

```
best_k = which(cv_error == min(cv_error))
best_k
```

```
## [1] 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
## [26] 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

```
plot(cv_error~K_knn,type='b',main = '10-Fold CV error v.s. choice of k in KNN',xlab = 'K',ylab = 'CV error')
```

10-Fold CV error v.s. choice of k in KNN



```
pred_train <- knn(train_x, train_x, train_label,
                  k = 2)
pred_test  <- knn(train_x, test_x, train_label, k = 2)
```

```
tp <- 6
fn <- 65
fp <- 54

(recall <- tp/(tp+fn))
```

```
## [1] 0.08450704
```

```
(precision <- tp/(tp + fp))
```

```
## [1] 0.1
```

```
(f1 <- 2*precision*recall/(precision+recall))
```

```
## [1] 0.09160305
```

```
#confusionMatrix(pred_train, as.factor(train_label), positive = "TRUE")
#mean(pred_train == train_label)
```

```
confusionMatrix(pred_test, as.factor(test_label), positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  2259   63
##      TRUE    50    8
```

```
##
##          Accuracy : 0.9525
##          95% CI : (0.9432, 0.9607)
##    No Information Rate : 0.9702
##    P-Value [Acc > NIR] : 1.000
##
##          Kappa : 0.0999
##
## Mcnemar's Test P-Value : 0.259
##
##          Sensitivity : 0.112676
##          Specificity : 0.978346
##    Pos Pred Value : 0.137931
##    Neg Pred Value : 0.972868
##          Prevalence : 0.029832
##    Detection Rate : 0.003361
##    Detection Prevalence : 0.024370
##    Balanced Accuracy : 0.545511
##
##    'Positive' Class : TRUE
##
```

```
mean(pred_test == test_label)
```

```
## [1] 0.952521
```

try weighted KNN

<https://search.r-project.org/CRAN/refmans/kknn/html/kknn.html>

```
Kfold_CV_kknn <- function(K,K_knn,train,train_label, kern){
  fold_size <- floor(nrow(train)/K)
  cv_error <- rep(0,K)
  sensitive <- rep(0,K)
  for(i in 1:K){
    # select K-1 folds
    if(i!=K){
      CV_test_rows = ((i-1)*fold_size+1):(i*fold_size)
    }else{
      CV_test_rows = ((i-1)*fold_size+1):nrow(train)
    }
    CV_train = train[-CV_test_rows,]
    CV_test = train[CV_test_rows,]
    # Fit knn
    fit.kknn = kknn(dv.hypertension1 ~., train = CV_train, test = CV_test,k = K_knn,
                    kernel = kern, distance = 2)
    pred_CV_test <- fit.kknn$fitted.values
    # Calculate error
    cv_error[i] = mean(pred_CV_test!=train_label[CV_test_rows])
    cm <- confusionMatrix(data = pred_CV_test, reference = train_label[CV_test_rows],
                          positive = "TRUE")
    sensitive[i] = cm$byClass["Sensitivity"]
  }
}
```

```

}
return(mean(sensitive))
}

```

```

K_fold <- 5
K_knn <- 3:25
kernels <- c("triangular", "epanechnikov", "optimal", "gaussian", "rectangular")
sensitives <- rep(0,length(K_knn))
train$dv.hypertension1 <- as.factor(train$dv.hypertension1)
for(kerns in kernels) {
  sensitives <- rep(0,length(K_knn))
  for(i in 1:length(K_knn)){
    kval<-K_knn[i]
    sensitives[i] <- Kfold_CV_kknn(K = K_fold, K_knn = kval,train = train,
                                  train_label = train$dv.hypertension1, kern=kerns)
  }
  best_k <- which(sensitives == max(sensitives))
}

```

```

knn.fit <- kknn(dv.hypertension1~., train, test,k=3, kernel = "optimal", distance = 2)
confusionMatrix(data = knn.fit$fitted.values, reference = as.factor(test$dv.hypertension1),
                 positive = "TRUE")

```

```

## Confusion Matrix and Statistics
##
##              Reference
## Prediction FALSE TRUE
##      FALSE  2265   63
##      TRUE    44    8
##
##              Accuracy : 0.955
##              95% CI : (0.9459, 0.963)
##      No Information Rate : 0.9702
##      P-Value [Acc > NIR] : 0.99998
##
##              Kappa : 0.1076
##
##  Mcnemar's Test P-Value : 0.08184
##
##      Sensitivity : 0.112676
##      Specificity : 0.980944
##      Pos Pred Value : 0.153846
##      Neg Pred Value : 0.972938
##      Prevalence : 0.029832
##      Detection Rate : 0.003361
##      Detection Prevalence : 0.021849
##      Balanced Accuracy : 0.546810
##
##      'Positive' Class : TRUE
##

```

upsampled data

```
train_one_hot <- read.csv("train_hot_X_y.csv") %>% select(-X)
test_one_hot <- read.csv("test_hot_X_y.csv") %>% select(-X)

data <- train_one_hot %>% select(-dv.hypertension)
test_data <- test_one_hot %>% select(-dv.hypertension)
```

```
K_knn <- 1:50
senses <- rep(0, length(K_knn))
Kfold_CV_knn1 <- function(K,K_knn,train,train_label){
  fold_size <- floor(nrow(train)/K)
  cv_error <- rep(0,K)
  sensitives <- rep(0,K)
  for(i in 1:K){
    # select K-1 folds
    if(i!=K){
      CV_test_rows <- ((i-1)*fold_size+1):(i*fold_size)
    }else{
      CV_test_rows <- ((i-1)*fold_size+1):nrow(train)
    }
    CV_train = train[-CV_test_rows,]
    CV_test = train[CV_test_rows,]
    # normalize the CV_train and CV_test
    mean_CV_train <- colMeans(CV_train)
    sd_CV_train <- apply(CV_train,2,sd)

    CV_train <- scale(CV_train,center = mean_CV_train,scale = sd_CV_train)
    CV_test <- scale(CV_test,center = mean_CV_train,scale = sd_CV_train)
    # Fit knn
    pred_CV_test <- knn(CV_train,CV_test,train_label[-CV_test_rows],k = K_knn)
    # Calculate CV error
    cv_error[i] <- mean(pred_CV_test!=train_label[CV_test_rows])
    cm <- confusionMatrix(data = as.factor(pred_CV_test),
                          reference = as.factor(train_label[CV_test_rows]), positive = "yes")
    sensitives[i] <- cm$byClass["Sensitivity"]
  }
  senses[i] <- mean(sensitives)
  return(mean(cv_error))
}
```

```
K_fold <- 10
K_knn <- 1:50
cv_error <- rep(0,length(K_knn))
for(i in 1:length(K_knn)){
  cv_error[i] <- Kfold_CV_knn1(K = K_fold, K_knn = K_knn[i],train = data,
                              train_label = train_one_hot$dv.hypertension)
}
```

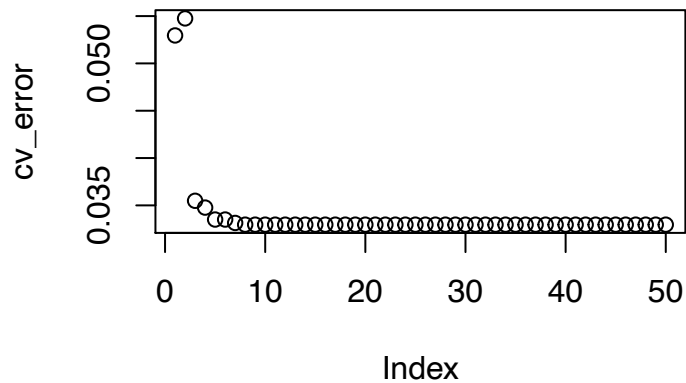
```
print(min(cv_error))
```

```
## [1] 0.03296266
```

```
best_k = which(cv_error == min(cv_error))
print(best_k)
```

```
## [1] 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
## [26] 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

```
plot(cv_error)
```



```
kfit <- knn(train = data, test = test_data, cl = train_one_hot$dv.hypertension, k = 2)
confusionMatrix(data = kfit, reference = test_one_hot$dv.hypertension,
                 positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no  yes
##          no 2250  65
##          yes  59   6
##
##              Accuracy : 0.9479
##              95% CI : (0.9382, 0.9565)
##      No Information Rate : 0.9702
##      P-Value [Acc > NIR] : 1.0000
##
##              Kappa : 0.0615
##
##  Mcnemar's Test P-Value : 0.6534
##
##              Sensitivity : 0.084507
##              Specificity : 0.974448
##      Pos Pred Value : 0.092308
##      Neg Pred Value : 0.971922
##      Prevalence : 0.029832
```

```
##          Detection Rate : 0.002521
## Detection Prevalence : 0.027311
##      Balanced Accuracy : 0.529477
##
##      'Positive' Class : yes
##
```



```
1 #####LIBRARY AND FUNCTIONS#####
2 #For cleaning and reading data
3 library(tidyverse)
4 library(caret)
5 library(themis)
6 library(EZtune)
7 library(MLmetrics)
8
9 #
10 # library(tidymodel)
11 #for plotting
12 theme_set(theme_bw())
13
14
15 #change wd, and import data
16 # setwd("~/Documents/Box Sync/Statistics Master/Fall 2022/STATS504/hw5")
17 setwd("/Users/brad/Downloads/hw5")
18 df <- read.csv("data/nuMoM2bsubset.csv")
19 # load("brad_models.RData") # Saved my global enviornment just in case...
20
21 ##### Groups 4, 9, 14 outcome: dv.hypertension1
22
23 null_outcomes <-c("dv.diabetes1",
24                   "dv.v1epdstotal",
25                   "dv.gestweeks",
26                   "dv.preeclampsia")
27
28 df <- df[,!(names(df) %in% null_outcomes)]
29
30 ##### DATA CLEANING #####
31 test_df = df %>% summarise(across(everything(), list(min,max)))
32 test_df = t(test_df)
33
34 #Convert booleans to "TRUE"/"FALSE"
35 df$emosupport <- df$emosupport == 1
36 df$financialsupport <- df$financialsupport==1
37 df$prenatalsupport <- df$prenatalsupport == 1
38 df$financialsupport <- df$financialsupport ==1
39 df$deliverysupport <- df$deliverysupport ==1
40 df$exercise <- df$exercise == 1
41 df$dv.hypertension1 <- df$dv.hypertension1==1
42 df$kidney1 <- df$kidney1==1
43 df$lupus1 <- df$lupus1 == 1
44 df$collagen1 <- df$collagen1==1
45 df$crohns1 <- df$crohns1 == 1
46 df$pcos1 <- df$pcos1 == 1
47
48
49 #Three level factors....
50 df$familypreeclampsia <- as.factor(df$familypreeclampsia)
51 df$bornearly <- as.factor(df$bornearly)
52
53
54
55 #Higher level factors....
56 df$worryfambaby <- as.factor(df$worryfambaby)
```

```

57 df$worryhealthcare <- as.factor(df$worryhealthcare)
58 df$worrysymptoms <- as.factor(df$worrysymptoms) #what are the levels for this? There should be a codebook.
59 df$discrimination <- as.factor(df$discrimination)
60 df$race <- as.factor(df$race)
61
62 # skimr::skim(df)
63 #####SUMMARY TABLE#####
64 skimmed_df = skimr::skim(df)
65 skimmed_df$n_missing = NULL
66 skimmed_df$complete_rate= NULL
67 skimmed_df$numeric.hist = NULL
68 skimmed_df$factor.ordered = NULL
69
70 #For factors
71 skimmed_df$factor.top_counts[8:18] <- skimmed_df$logical.count[8:18]
72 skimmed_df$factor.top_counts[8:18] <- skimmed_df$logical.count[8:18]
73 skimmed_df$factor.n_unique[8:18] <- 2
74 skimmed_df$numeric.mean[8:18] <- skimmed_df$logical.mean[8:18]
75 skimmed_df$numeric.mean <- round(skimmed_df$numeric.mean, digits= 4)
76 skimmed_df$logical.mean <- NULL
77 skimmed_df$logical.count = NULL
78 skimmed_df$factor.top_counts[19:26] <- paste0(
  ("",skimmed_df$numeric.p25[19:26],",",skimmed_df$numeric.p75[19:26],")")
79 skimmed_df$numeric.sd <- NULL
80 skimmed_df$numeric.p0 <- NULL
81 skimmed_df$numeric.p25 <- NULL
82 skimmed_df$numeric.p50 <- NULL
83 skimmed_df$numeric.p75 <- NULL
84 skimmed_df$numeric.p100 <- NULL
85 # write.csv(skimmed_df, "data/baseline.csv")
86
87
88 rm(test_df, null_outcomes) #drop unused items
89
90 #####IMPUTE VALUES#####
91 colSums(df == 0)
92 #age
93 #psstotla
94 #ssqmean
95 # prepreglbs
96
97 df$age[df$age==0] <- mean(df$age[df$age!=0])
98 df$prepreglbs[df$prepreglbs==0] <- mean(df$prepreglbs[df$prepreglbs!=0])
99
100 #####TEST TRAIN#####
101 set.seed(1123)
102 size = floor(0.3*dim(df))
103
104 id = sample(c(1:7934),replace=F, size = floor(0.3*dim(df)))[1])
105 train<-df[-id,]
106 test<-df[id,]
107
108 # write.csv(test,"test_df.csv")
109 # write.csv(train,"train_df.csv")
110
111 #####ENCODE TRAIN#####
112 dmy <- dummyVars(" ~ .", data = train)
113 train_hot <- data.frame(predict(dmy, newdata = train))
114
115 train_hot_X = train_hot[,!(names(train_hot) %in% c("dv.hypertension1FALSE","dv.hypertension1TRUE"))]

```

```

116
117 #Outcome variable needs to have a valid name. Use make.names() or use
118 train_hot_y = factor(train_hot$dv.hypertension1TRUE,
119                       levels = c(1,0),
120                       labels = c("yes","no"))
121
122
123
124 #Combined data frame .....
125 train_hot_X_y<- train_hot_X
126 train_hot_X_y$dv.hypertension <- train_hot_y
127
128
129 #####UP SAMPLE TRAINING DAT#####
130 #minority class has 50% of observations as majority
131 train_up50<- smote(train_hot_X_y, var ="dv.hypertension", over=0.5, k=10)
132 table(train_up50$dv.hypertension)
133 #majority class has 100% of observations as majority
134 train_up100<- smote(train_hot_X_y, var ="dv.hypertension", over=1, k=10)
135 table(train_up100$dv.hypertension)
136
137 #Export
138 # write.csv(train_up50, "data/train_one_hot_50.csv")
139 # write.csv(train_up100, "data/train_one_hot_100.csv")
140
141
142
143
144 #####columns with no variance in training data #####
145 X_no_var = nearZeroVar(train_hot_X_y)
146 X_no_var_names = names(train_hot_X_y)[X_no_var]
147 X_no_var_names = X_no_var_names[-c(1,37)] #hypertension has low variance, as does race == native
148
149 #Drop columns with no variance....
150 train_hot_X_y <- train_hot_X_y[!(names(train_hot_X) %in% X_no_var_names)]
151 train_up50 <- train_up50[!(names(train_up50) %in% X_no_var_names)]
152 train_up100 <- train_up100[!(names(train_up100) %in% X_no_var_names)]
153
154 # Check the distribution of these variables....
155 # train_hot_X[,X_no_var_names] %>% skimr::skim()
156
157 #####DOWNSAMPLE#####
158 set.seed(1123)
159 train_hot_down = downSample(x=train_hot_X_y,
160                             y=train_hot_X_y$dv.hypertension)
161 train_hot_down$Class <- NULL
162
163 #####ENCODE TEST#####
164 dmy <- dummyVars(" ~ .", data = test)
165 test_hot <- data.frame(predict(dmy, newdata = test))
166
167 test_hot_X = test_hot[!(names(test_hot) %in% c("dv.hypertension1FALSE", "dv.hypertension1TRUE"))]
168
169 #Outcome variable needs to have a valid name. Use make.names() or use
170 test_hot_y = factor(test_hot$dv.hypertension1TRUE,
171                     levels = c(1,0),
172                     labels = c("yes","no"))
173
174 #combine into one data frame....
175 test_hot_X_y = test_hot_X

```

```

176 test_hot_X_y$dv.hypertension = test_hot_y
177
178 #Drop columns with no variance
179 test_hot_X_y <- test_hot_X_y[,!(names(test_hot_X) %in% X_no_var_names)]
180
181
182 #####CLEAN WORKSPACE#####
183 rm(list = c("#test_hot_X_sub",
184             "test_hot_X",
185             "test_hot",
186             "test_hot_y",
187             # "train_hot_X_sub",
188             "train_hot_X",
189             "train_hot",
190             "train_hot_y",
191             "dmy"))
192
193 ##### HYPERPARAMETER TUNING#####
194
195 models <- caret::modellookup() #what models are in the caret package?
196
197 ##### Adaboost.M1#####
198
199 #!!!!!!!!!!!!!! WARNING THE FOLLOWING CHUNKS TAKE ~40 Minutes to run!!!!!!!!!!!!!!
200
201 fitGrid_ada <- expand.grid(mfinal = c(1,6,9,100),
202                           # mfinal = (1:3)*3,
203                           # maxdepth = c(1:3),
204                           maxdepth = c(1,2,4),
205                           coeflearn = c("Breiman"))
206
207 fitControl_ada <- trainControl(method = "repeatedcv",
208                                repeats = 5,
209                                classProbs = T,
210                                # summaryFunction = twoClassSummary,
211                                summaryFunction = prSummary)
212 #on up sampled
213
214 # using the adaboost.m1 package...
215 set.seed(1123)
216 start_time = Sys.time()
217 ada.mod <- train(x=train_hot_X_y[,-48],
218                 y= train_hot_X_y$dv.hypertension,
219                 method = 'AdaBoost.M1',
220                 trControl = fitControl_ada,
221                 tuneGrid = fitGrid_ada,
222                 metric = "AUC",
223                 verbose = TRUE)
224 total_time <- Sys.time() - start_time
225 total_time
226
227 # Upsampled to be 50% majority class
228 set.seed(1123)
229 start_time <- Sys.time()
230 ada50.mod <- train(x=train_up50[,-48],
231                   y= train_up50$dv.hypertension,
232                   method = 'AdaBoost.M1',
233                   trControl = fitControl_ada,
234                   tuneGrid = fitGrid_ada,
235                   metric = "AUC",

```

```

236         verbose = TRUE)
237 total_time <- Sys.time() - start_time
238 total_time
239
240
241 # upsampled to Matched classes - change metric to ROC.
242 set.seed(1123)
243 total_time <- Sys.time()
244 ada100.mod <- train(x=train_up100[,-48],
245                    y= train_up100$dv.hypertension,
246                    method = 'AdaBoost.M1',
247                    trControl = fitControl_ada,
248                    tuneGrid = fitGrid_ada,
249                    metric = "ROC",
250                    verbose = TRUE)
251 total_time <- Sys.time() - start_time
252 total_time
253
254 # On downsampled - use ROC
255 set.seed(1123)
256 start_time <- Sys.time()
257 adadown.mod <- train(x=train_hot_down[,-48],
258                    y= train_hot_down$dv.hypertension,
259                    method = 'AdaBoost.M1',
260                    trControl = fitControl_ada,
261                    tuneGrid = fitGrid_ada,
262                    metric = "ROC",
263                    verbose = TRUE)
264 total_time <- Sys.time() - start_time
265 total_time
266
267
268 ##### USING GBM PACKAGE #####3
269
270 # set.seed(1123)
271 # start_time <- Sys.time()
272 # ada.mod <- train(x=train_hot_X_y[,-48],
273 #                 y= train_hot_X_y$dv.hypertension,
274 #                 distribution = 'adaboost',
275 #                 method="gbm",
276 #                 trControl = fitControl_ada,
277 #                 tuneGrid = fitGrid_ada,
278 #                 metric = "AUC",
279 #                 verbose = TRUE)
280 # total_time <- Sys.time() - start_time
281 # total_time
282
283 # set.seed(1123)
284 # start_time <- Sys.time()
285 # ada50.mod <- train(x=train_up50[,-48],
286 #                   y= train_up50$dv.hypertension,
287 #                   distribution = 'adaboost',
288 #                   method="gbm",
289 #                   trControl = fitControl_ada,
290 #                   tuneGrid = fitGrid_ada,
291 #                   metric = "AUC",
292 #                   verbose = TRUE)
293 # total_time <- Sys.time() - start_time
294 # total_time
295

```

```

296
297 # set.seed(1123)
298 # total_time <- Sys.time()
299 # ada100.mod <- train(x=train_up100[,-83],
300 #                    y= train_up100$dv.hypertension,
301 #                    distribution = 'adaboost',
302 #                    method="gbm",
303 #                    trControl = fitControl_ada,
304 #                    tuneGrid = fitGrid_ada,
305 #                    metric = "ROC",
306 #                    verbose = TRUE)
307 # total_time <- Sys.time() - start_time
308 # total_time
309
310
311 #Fit downsampled data on finer grid...
312 # fitGrid_ada <- expand.grid(interaction.depth = c(1, 3, 6, 9),
313 #                             n.trees = c(1,10,20,50,100),
314 #                             shrinkage = seq(.0005, .05,.0005),
315 #                             n.minobsinnode = 10)
316 #
317 # fitControl_ada <- trainControl(method = "repeatedcv",
318 #                                repeats = 5,
319 #                                classProbs = T,
320 #                                summaryFunction = twoClassSummary)
321 #on up sampled
322 # set.seed(1123)
323 # start_time <- Sys.time()
324 # adadown.mod <- train(x=train_hot_down[,-83],
325 #                     y= train_hot_down$dv.hypertension,
326 #                     distribution = 'adaboost',
327 #                     method="gbm",
328 #                     trControl = fitControl_ada,
329 #                     tuneGrid = fitGrid_ada,
330 #                     metric = "ROC",
331 #                     verbose = TRUE)
332 # total_time <- Sys.time() - start_time
333 # total_time
334
335
336
337 #####PREDICTION#####
338 # Class Predictions
339 test_predada <- predict(object = ada.mod,newdata = test_hot_X_y[,-48])
340 test_predada50 <- predict(object = ada50.mod,newdata = test_hot_X_y[,-48])
341 test_predada100 <- predict(object = ada100.mod,newdata = test_hot_X_y[,-48])
342 test_predadadown <- predict(object = adadown.mod,newdata = test_hot_X_y[,-48])
343
344 # Probabilities
345 test_predada_p <- predict(object = ada.mod,newdata = test_hot_X_y[,-48],type="prob")
346 test_predada50_p <- predict(object = ada50.mod,newdata = test_hot_X_y[,-48],type = "prob")
347 test_predada100_p <- predict(object = ada100.mod,newdata = test_hot_X_y[,-48],type = "prob")
348 test_predadadown_p <- predict(object = adadown.mod,newdata = test_hot_X_y[,-48],type = "prob")
349
350 #####PRAUC#####
351 ada_prauc = MLmetrics::PRAUC(test_predada_p$yes, test_hot_X_y$dv.hypertension)
352 ada_prauc
353
354 adaup_prauc = MLmetrics::PRAUC(test_predada100_p$yes, test_hot_X_y$dv.hypertension)
355 adaup_prauc

```

```

356 ##### AUROC #####
357 # library(pROC)
358 # ada.roc <- roc(test_hot_X_y$dv.hypertension, test_predada_p$yes)
359 # # plot(ada.roc, print.thres="best", print.thres.best.method="closest.topleft")
360 # ada50.roc <- roc(test_hot_X_y$dv.hypertension, test_predada50_p$yes)
361 # ada100.roc <- roc(test_hot_X_y$dv.hypertension, test_predada100_p$yes)
362 #
363 #
364 # plot(ada50.roc, print.thres="best", print.thres.best.method="closest.topleft")
365 # result.coords <- coords(ada.roc, "best", best.method="closest.topleft", ret=c("ppv", "tpr"))
366 # print(result.coords)#to get threshold and accuracy
367
368
369 #####CONFUSION METRICS#####
370 conf_matada = table("truth"=test_hot_X_y$dv.hypertension,"pred"= test_predada)
371 conf_matada=confusionMatrix(conf_matada, mode = "everything", positive = "yes")
372 conf_matada
373
374
375 conf_matada50 = table("truth"=test_hot_X_y$dv.hypertension,"pred"= test_predada50)
376 conf_matada50=confusionMatrix(conf_matada50, mode = "everything", positive = "yes")
377 conf_matada50
378
379 conf_matada100 = table("truth"=test_hot_X_y$dv.hypertension,"pred"= test_predada100) #Adaboost looks a bit better
380 conf_matada100=confusionMatrix(conf_matada100, mode = "everything", positive ="yes")
381 conf_matada100
382
383 conf_matadadown= table("truth"=test_hot_X_y$dv.hypertension,"pred"= test_predadadown) #Adaboost looks a bit better
384 conf_matadadown=confusionMatrix(conf_matadadown, mode = "everything", positive ="yes")
385 conf_matadadown
386
387 metrics_df = data.frame(ada = conf_matada$byClass,
388                          ada50 = conf_matada50$byClass,
389                          ada.tune100 = conf_matada100$byClass)
390
391 # save.image(file='brad_models.RData')
392 # load("brad_svm_env.RData")
393
394 #####PLOTS #####
395 hyp_race_p = df %>% group_by(race) %>% summarise(p = mean(dv.hypertension1)) %>%
396   ggplot(aes(x=reorder(race, -p), y = p, fill = race))+
397   geom_bar(stat='identity')+
398   scale_y_continuous(labels = scales::percent)+
399   xlab("Race")+
400   ylab("Perc. of Women w/ Hypertension")+
401   guides(fill="none")
402
403
404 ggplot(data = df, aes(x=prepreglbs, fill = race, group = race))+
405   # geom_density(alpha=0.4)+
406   geom_histogram(aes(y=stat(density)))+
407   # scale_y_continuous(labels = scales::percent)+
408   # xlab("Frequency")+
409   # scale_y_continuous(labels = percent )
410   facet_grid(rows = vars(race))
411   # ylab("Perc. w/ Hypertension")+
412   # guides(fill="none")
413
414

```

```
415 ggplot(data=df)+
416   geom_jitter(aes(x=diastolic,
417                   y = systolic,
418                   col = dv.hypertension1),
419               alpha=0.5)+
420   facet_wrap(~dv.hypertension1)+
421   scale_x_continuous(sec.axis = sec_axis(~ . ,
422                                           name = "Has Hypertension",
423                                           breaks = NULL,
424                                           labels = NULL))+
425   # scale_y_continuous(labels = scales::percent)+
426   xlab("Systolic Blood Pressure At First Visit")+
427   ylab("Diastolic Blood Pressure At First Visit")+
428   guides(col="none")
429
430
431
```



```
In [331... import pandas as pd
from imblearn.over_sampling import SMOTE
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
```

Read in Data and perform SMOTE to handle class imbalance

```
In [332... df = pd.read_csv("train_df.csv", index_col=0)
X, y = pd.get_dummies(df.drop("dv.hypertension1", axis=1)), df["dv.hypertension1"]
```

```
In [333... X_orig, y_orig = X.copy(), y.copy()
```

```
In [334... X.columns
```

```
Out[334... Index(['age', 'emosupport', 'financialsupport', 'prenatalsupport',
      'deliverysupport', 'psstotal', 'anxttotal', 'worryfambaby', 'exercise',
      'systolic', 'diastolic', 'worryhealthcare', 'worrysymptoms', 'ssqmean',
      'prepreglbs', 'familypreeclampsia', 'income', 'kidney1', 'lupus1',
      'collagen1', 'crohns1', 'pcos1', 'discrimination', 'bornearly',
      'race_black', 'race_hispanic', 'race_native', 'race_other',
      'race_white'],
      dtype='object')
```

```
In [335... oversample = SMOTE()
X, y = oversample.fit_resample(X, y)
```

```
In [336... X.shape
```

```
Out[336... (10742, 29)
```

```
In [337... test_df = pd.read_csv("test_df.csv", index_col=0)
X_test, y_test = pd.get_dummies(test_df.drop("dv.hypertension1", axis=1)), test_df["dv.hypertension1"]
```

Model fitting and variable selection

```
In [338... rf = RandomForestClassifier(max_depth=3)
rf.fit(X,y)
```

```
Out[338... ▼      RandomForestClassifier
RandomForestClassifier(max_depth=3)
```

```
In [339... rf.score(X_test, y_test)
```

```
Out[339... 0.8336134453781513
```

```
In [340... from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
```

```
In [341... f1_score(y_test, y_hat)
```

```
Out[341... 0.027397260273972605
```

```
In [342... recall_score(y_test, y_hat)
```

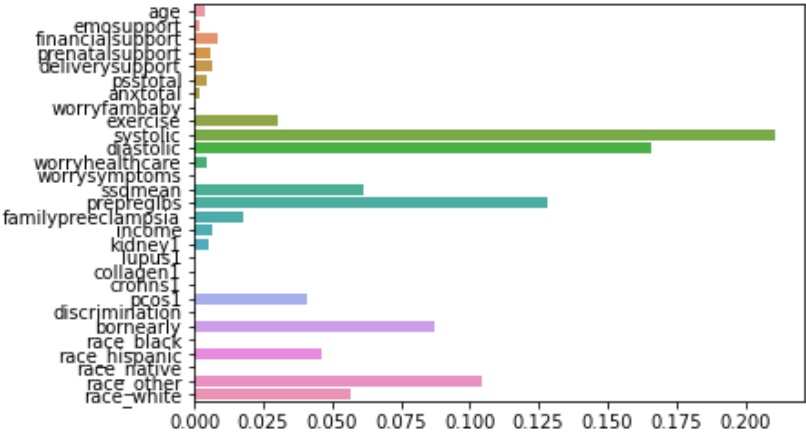
Out[342... 0.014084507042253521

```
In [343... precision_score(y_test, y_hat)
```

Out[343... 0.5

```
In [344... sns.barplot(x = rf.feature_importances_, y = rf.feature_names_in_)
```

Out[344... <AxesSubplot: >



```
In [345... rf.feature_importances_
```

Out[345... array([3.86155030e-03, 1.95458504e-03, 8.59539108e-03, 5.69465183e-03, 6.24187987e-03, 4.53219561e-03, 1.71176302e-03, 7.10330658e-04, 3.00137305e-02, 2.10912060e-01, 1.65669615e-01, 4.26874348e-03, 1.28535094e-04, 6.15763318e-02, 1.28450732e-01, 1.75898461e-02, 6.34511987e-03, 5.35490712e-03, 0.00000000e+00, 1.60465339e-04, 2.18437448e-05, 4.07629791e-02, 5.77988718e-04, 8.75051851e-02, 0.00000000e+00, 4.61412072e-02, 0.00000000e+00, 1.04521376e-01, 5.66969855e-02])

Use the variable importances from the full model to decide which variables to include

Model Selection (Grid Search)

```
In [346... cols = list(map(lambda t: t[1], filter(lambda t: t[0] > 0.02, zip(rf.feature_importances_, rf.feature_names_in_))))
X_train = X[cols + ["race_black", "race_native"]]
X_train_orig = X_orig[cols + ["race_black", "race_native"]]
X_train
```

		exercise	systolic	diastolic	ssqmean	prepreglbs	pcos1	bornearly	race_hispanic	race_other	race_white	race
	0	True	126	80	7.000000	145.000000	False	3	0	0	1	
	1	False	136	82	6.833333	220.000000	False	3	0	0	0	
	2	True	100	72	4.000000	98.000000	False	3	0	0	0	
	3	True	128	70	6.916667	335.102240	False	2	0	0	1	
	4	False	128	78	2.666667	262.000000	False	3	0	0	0	
	
10737	True	113	61	6.373740	136.732354	False	3	0	0	0	0	
10738	True	102	65	6.308870	103.279340	False	2	0	1	0	0	
10739	True	103	65	5.789501	184.412336	False	2	0	0	0	0	

	exercise	systolic	diastolic	ssqmean	prepreglbs	pcos1	bornearly	race_hispanic	race_other	race_white	race
10740	False	118	71	6.684672	299.838241	False	3	0	0	0	
10741	True	102	66	6.718174	122.618094	False	3	0	0	1	

10742 rows × 12 columns

```
In [388... rf = RandomForestClassifier(random_state=1234)
clf = GridSearchCV(rf, {"n_estimators": [10, 50, 100, 500], "max_depth": [1, 2, 3]})
clf.fit(X_train, y)
```

```
Out[388... ▸ GridSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

```
In [389... X_test = X_test[cols + ["race_black", "race_native"]]
```

```
In [390... clf.score(X_test, y_test)
```

```
Out[390... 0.8067226890756303
```

```
In [391... best_rf = clf.best_estimator_
best_rf
```

```
Out[391... ▼ RandomForestClassifier
RandomForestClassifier(max_depth=3, random_state=1234)
```

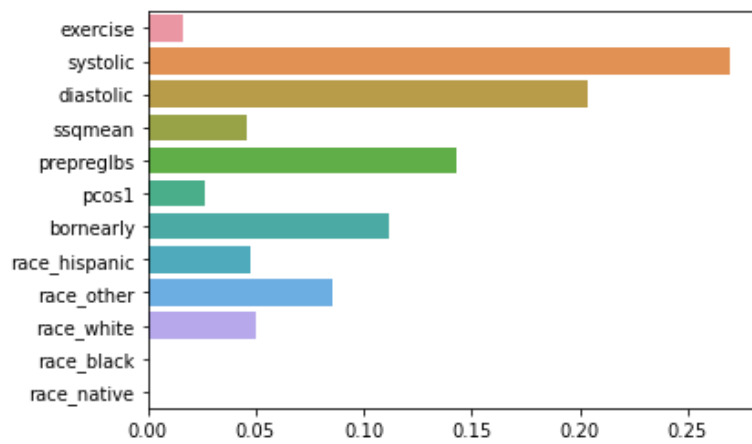
```
In [392... from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
```

```
In [393... y_hat = clf.predict(X_test)
print(f"F1 Score: {f1_score(y_test, y_hat)}",
      f"Precision: {precision_score(y_test, y_hat)}",
      f"Recall: {recall_score(y_test, y_hat)}",
      f"Accuracy: {accuracy_score(y_test, y_hat)}",
      sep="\n"
    )
```

```
F1 Score: 0.15441176470588236
Precision: 0.08879492600422834
Recall: 0.5915492957746479
Accuracy: 0.8067226890756303
```

```
In [383... sns.barplot(x = best_rf.feature_importances_, y = best_rf.feature_names_in_)
```

```
Out[383... <AxesSubplot: >
```



In [402...

```
from sklearn.tree import export_graphviz
import pydot
tree = best_rf.estimators_[0] # pick first tree in the ensemble for visualization

export_graphviz(
    tree,
    out_file = 'tree.dot',
    feature_names = best_rf.feature_names_in_,
    rounded = True,
    precision = 1,
    impurity=False,
    proportion=True,
    rotate=True
)
# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('tree.dot')
# Write graph to a png file
graph.write_png('tree.png')
```

In [403...

```
best_rf.n_estimators
```

Out[403...

100

In [385...

```
best_rf.fit(X_train_orig, y_orig)
```

Out[385...

```
▼ RandomForestClassifier
RandomForestClassifier(max_depth=3, random_state=1234)
```

In [386...

```
y_hat = best_rf.predict(X_test)
print(f"F1 Score: {f1_score(y_test, y_hat)}",
      f"Precision: {precision_score(y_test, y_hat)}",
      f"Recall: {recall_score(y_test, y_hat)}",
      f"Accuracy: {accuracy_score(y_test, y_hat)}",
      sep="\n"
    )
```

```
F1 Score: 0.0
Precision: 0.0
Recall: 0.0
Accuracy: 0.9701680672268908
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```