## Introduction

Hypertension (high blood pressure) is a condition where the pressure of the blood pumping against the walls of the arteries is elevated beyond normal levels. The goal of this analysis is to predict the history of hypertension in pregnant women with no previous pregnancy lasting 20 weeks or more (nulliparas). Specifically, there is an importance to identifying those who have a history of hypertension (high blood pressure). A history of hypertension might be an indication of a higher risk for developing gestational hypertension. For some women the development of gestational hypertension may increase their risk for more serious complications later in pregnancy, like preeclampsia.

Our preliminary results indicate that predicting history of hypertension is a difficult, if not impossible, problem to solve, but our results highlight what approaches are possible. Adaboost (which is an ensemble technique further discussed in the methods section) is the best performing model using variables like systolic, diastolic, and race, which were deemed important, but even this comes with its set of limitations and nuances which are discussed in this report.

### Methods

The goal of the analysis is to predict the history of hypertension in pregnant women. The information provided to tackle this problem includes many items from the medical history of the patient. We want to train a statistical model that can use this information to predict whether or not the person has a history of hypertension, which could potentially serve as a proxy for predicting whether the person will develop hypertension during pregnancy.

Since our data is imbalanced (only 3% of individuals in the data have a history of hypertension), we upsampled the training set in an attempt to have our model learn from more examples of hypertension cases. Synthetic Minority Over-sampling Technique (SMOTE) was used to upsample the training data so that the number of observations with hypertension matched the number of observations without hypertension. The algorithm is similar to K-Nearest Neighbors, where the algorithm generates examples from the predictors which are neighbors of the predictors from the minority class. Since SMOTE requires the computation of numeric distances between neighboring points, it is necessary for the data to be numeric.

One-hot encoding was used to create numeric representations of categorical variables. One-hot encoding transforms a categorical variable into several columns where each categorical level has a column with entries of 1 or 0, with 1 indicating that the column has the given property. For example, the race variable was split into multiple columns like race.white, race.black, race.hispanic etc where exactly one of these columns had a 1 and others were 0. Several categories had very few responses, so columns with low variance were dropped from the training

of some of our models. However, some variables with low variance were retained in the model due to the underlying social factors associated with hypertension, such as including native american women in the dataset.

Before the models were fitted with the preprocessed data, we had to identify appropriate evaluation metrics to enable comparison of different models. A confusion matrix allows us to understand how well the model is doing at predicting different kinds of outcomes i.e true positives, true negatives, false positives and false negatives. We use metrics such as Recall, Precision, and F1 score to provide a more holistic view of model performance. Recall represents the percentage of women with history of hypertension correctly classified by the model as having a history of hypertension, while Precision represents the percentage of women who actually had a history of hypertension amongst all women classified as having a history of hypertension. Finally, the F1 score is the weighted combination of recall and precision, generally described as the harmonic mean of two. These metrics are clearly more powerful in describing the usefulness of our models, as simply looking at accuracy (percentage classified correctly) can be misleading due to the data being imbalanced.

Another aspect of model training was to be cautious about overfitting. To tackle this, we used 5-fold cross validation for all models that were fitted. Cross-validation is a resampling procedure used to evaluate models on a limited data sample. The k-fold cross validation splits data into k groups, where k-1 groups are trained and the last is used for validation. This helps models utilize all the information in a limited data set and avoid overfitting. We use this approach to tune parameters which cannot be done strictly using the training set because otherwise we risk overfitting.

Given the combination of categorical and numerical variables, and our goal of predicting a binary outcome (presence or absence of history of hypertension), we settled on the following models that are suitable for binary classification problems:

**Logistic regression**

Building a logistic regression model will allow us to determine how strong the association is between two or more factors and the patient's condition. Logistic regression can be used to predict whether a patient will have hypertension. Advantages to using logistic regression include that it is easy to interpret. It not only measures the association between variables, but also measures the direction of the association (positive or negative). A limitation is the assumption of linearity between the outcome and independent variables. With logistic regression, it is difficult to model complex relationships.

Since there are many variables in the data set, we will consider using backward and forward selection to obtain the best model.

**Multi-Layer Perceptron**

Multi-Layer Perceptron is a relatively simple form of Neural Network, where the information is feedforward from input to hidden and then to output layer. It utilizes backpropagation, an algorithm through which one feeds forward the values, calculates the errors and propagates it back to the earlier layers. Advantage comes that its multiple layers and non-linear activation function can help to classify data that is not linearly separable. A limitation for this approach is the shortage of interpretation, that it relies on complex co-adaptations of weights during the training phase instead of measuring and comparing quality of splits, and thus we are not able to easily access information other than prediction scores.

**K-nearest neighbors (KNN)**

This technique uses an integer k and a similarity measure (typically euclidean distance) to find the k nearest points determined by the similarity measure, and makes the prediction based on the classification of the majority of the neighbors. If the variables have very different ranges and units, all variables will be standardized. Generally, a smaller value of k gives a more flexible model. For example, when k = 1 each new point would be classified by only its nearest neighbor, thus, a small k can lead to overfitting when training the model. Higher dimensions (more predictors) mean that points are generally farther apart which means that neighbors cover a larger area which increases the model's bias. Reducing the number of neighbors reduces the bias, but the predictions become noisier because there are not as many neighbors to balance leading to noisier predictions and a higher variance. Cross-validation will be used to choose the value of k, the number of neighbors. The results of the model are not very interpretable and the method does not offer a form of variable selection. Further, it cannot use categorical variables as predictors.

**AdaBoost**

AdaBoost is an ensemble type method where a collection of "weak learners" (e.g. stumps of decision trees) are fit in a sequential manner, and each of these learners makes a prediction. Points which are misclassified are upweighted to penalize the misclassification, and points which are correctly classified are down weighted. The final classification is decided by aggregating the decision of those weighted classifiers. AdaBoost was chosen as boosting type models are typically comparable to Random Forest models, and allows us to access variable importance, so some of the mysteries of black box models are avoided. Limitations of the model are twofold: hyperparameter tuning is computationally expensive, and there is still some inaccessibility as to why the final classifier is making certain decisions. As the model is fit in a sequential order, parallel processing is not possible. Completing a small grid search for optimal hyperparameters took between 12 to 30 minutes, and larger grid searches seem to yield diminishing returns in terms of increases in precision.

**Random Forest**

Similar to AdaBoost, Random Forest is another ensemble technique which aggregates multiple learners in order to make a prediction. The key difference between AdaBoost and a Random Forest is in the sequentiality during the model training phase. A Random Forest is just a collection of decision trees (or stumps), and each tree's prediction can be obtained in parallel. For a classification problem like this one, the final prediction of the Random Forest is whatever the majority of trees predicted as the outcome (hypertension, True or False). Similar to AdaBoost, Random Forests also provide information about variable importance so the model is not entirely a black box. Furthermore, an arbitrarily picked decision tree from the forest is presented in an attempt to aid interpretability. However, a Random Forest is still not inherently interpretable in terms of inference of causal relationships which is one of its primary relationships. It cannot provide information about how each variable of the dataset affects the outcome (like Logistic Regression for example). Since the goal was to optimize on the prediction accuracy and a compromise on inference and interpretation was warranted, this limitation did not necessarily need to be addressed.

## Results

The provided data is a subset of data from the NuMoM2b study, collected by interviews, questionnaires, clinical measurements, and so on. Each row in the dataset contains a patient's medical records and information including demographic, psychosocial, dietary, physiological, health, and pregnancy outcomes. The outcome variable (dv.hypertension1) our analysis focuses on is the history of hypertension recorded on the first visit. The data contains 7,934 observations, with 26 variables (25 predictors), and was split into a testing and training set of 2,380 and 5,554 observations, respectively. The imbalance of the data, where only 3% patients are recorded to have hypertension history, is the main issue.

Several variables had a level that encompassed all of the reasons that someone would not have an answer to a question (i.e. they did not know or refused to answer or the value is just missing). For example, in the age column, there are values that are zero when the patients in this dataset should all be older than zero.

All of the data variables are further summarized in the baseline table presented in the Appendix as requested by the client.

We present the results of each of the models along with some interpretation of them where necessary. Adaboost was the best performing model, but it is useful to look at the results from the other models as well.

**Logistic regression**

There is not a notable difference between the test error of the full model and the models chosen using forward and backward selection. Considering AIC as another model metric, the model chosen using backward selection is the best model as it has a lower AIC score. It is also a simpler model with less variables. Variables included in the model are age, race, systolic blood pressure, diastolic blood pressure, worry symptoms score, pre-pregnancy weight, history of kidney disease, history of PCOS, whether a patient has been discriminated against and whether the mother was born early. Results indicate that age, systolic blood pressure, diastolic blood pressure, pre-pregnancy weight, the mother being born early, history of kidney disease and history of PCOS are all significantly associated with having hypertension at a $\alpha = 0.05$ significance level. Further, the odds of having hypotension is .43 times lower if you are white compared to other races. On the other hand, the odds of having hypertension are 1.76 times higher among those with a history of PCOS.

Further, results indicate the model is good at predicting cases when individuals do not have a history of hypertension. However, the focus on our analysis is to be able to correctly predict those who do have a history of hypertension. Out of the 71 patients in the data set who do have a history of hypertension, only 2 of these individuals were correctly classified. The precision of this model is only 2.82%.

**Multi-Layer Perceptron**

All variables were included in the model, with One Hot encoding on variable race, MinMaxScalar (normalization) on numeric features, and dropping duplicate samples. After that, the training data size was 5516 samples and 29 feature columns. Income, age and pre-pregnancy weight with zero entries that were considered to be missing value, and were replaced with their mean values. Through grid search and 5-fold cross validation, the model was tuned to find optimal hidden layer size, activation function, solver, learning rate type, and l2 penalty parameter based on recall score criteria. MLP was tuned in two scenarios: the one fitted on the original imbalanced data only had 9 patients to be errorly predicted hypertension yet only 6 out of 71 history hypertension cases were found. Its precision score was up to 0.4 while poor behavior in recall score (0.0845) and f1 score (0.1395). The other one fitted on the upsampled data, which contains the same size for each class. It turned out to correctly predict 46 out of 71 hypertension cases, yet 187 patients were errorly predicted to have history of hypertension, and this led to a higher recall score (0.3521) and lower precision score (0.1179). However, the f1 score increased to 0.1767 after upsampled.

**KNN**

After trying different versions of KNN, including a weighted version using different kernels and one that was trained on the upsampled data, it was determined that the regular KNN performed

just as well as the other versions in terms of overall accuracy and performed very close to the weighted model when comparing recall scores (difference of about .01). Choosing the simpler model provides a more straightforward methodology. These models were tuned using 5 fold cross validation and both the overall accuracy and recall were calculated. Recall was a determinant in the model selection because correctly identifying the women with hypertension is important. The most accurate models did not correspond to the best recall models. After tuning, the optimal parameter value found was k = 2. This result feels as though it is overfitting because of the very small value of k. However, considering the KNN methodology, classifying based on the majority of the neighbors' classifications, given that by far most of the data is patients without hypertension, adding more neighbors will lead to nearly always predicting not having hypertension which undermines the most important goal, accurately identify as many people with hypertension as possible. The overall accuracy of the model was about .95, which was about .02 lower than the most accurate regular KNN models, but the much higher recall of .0845 for this model made this the best KNN model overall.

**AdaBoost**

Similar to what was done in the MLP pre-processing, predictors which were factors were converted to one-hot encoding, and numeric columns were centered and scaled, and numeric columns which had NA entries (indicated by 0s), were imputed to be the mean of the non-zero entries of that column (e.g. age, pre-pregnancy weight). Of the 82 predictors in the one-hot encoded predictor matrix, 37 of those predictors had nearly zero variance. Consideration was taken as to not remove variables that had low variance, but might be important to include for equity reasons, such as the race variable. The ten most important variables as determined gain in Gini index is given in Figure xxx. Both diastolic and systolic blood pressure taken at the time of first visit were the two most important by variables in predicting history of hypertension, this make some sense as the definition of hypertension is a systolic pressure greater than or equal to 130, or a diastolic pressure greater than or equal to 80.

The optimal models were found using a grid search over the number of trees used, and max depth of those trees where the target metric was to maximize the area under the precision-recall curve (PRAUC). Two models were fit: one trained on the one-hot encoded training data as described in the previous paragraph, and one on the upsampled SMOTE training set, which contained the same number instances for each class.

The model trained on the original unbalanced data yielded correctly predicted a woman having a history of hypertension 0.0845 of the time (precision), and the model was only able to identify about a third of women predicted to have had hypertension correctly (recall). The F1 score is 0.1348, which is the harmonic mean of the precision and recall. In contrast the model trained on balanced upsampled data correctly predicted a woman having a history of hypertension 0.1408 of the time, and similarly, the model was only able to correctly predict if a woman had hypertension

sometime in the past correctly about a third of the time. The F1 score increased to 0.1923 is the harmonic mean of the precision and recall. The overall test accuracy of both models was nearly 97%, which is mostly the byproduct of the unbalanced classes. These results demonstrate how upsampling can be used to train a model to increase the precision of its prediction, although the gains are marginal at best.

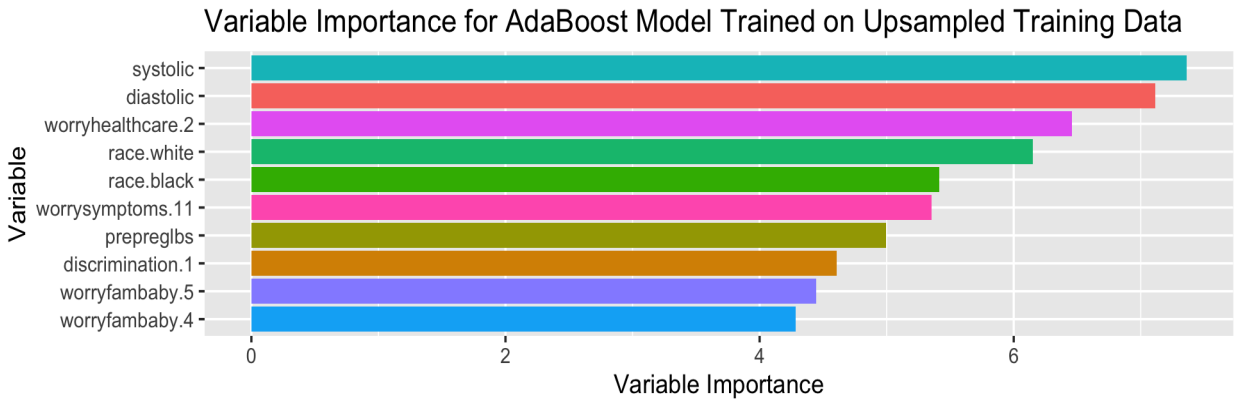



Figure 1: Variable Importance of AdaBoost Model

**Random Forest**

The data preprocessing was performed similarly to that of the other models with categorical variables being one-hot-encoded. Variables for the model were selected based on the variable-importance that Random Forests provide. Once the variables were selected, models were fitted on both the original imbalanced data as well as the SMOTE-balanced data. The model that was trained on the imbalanced data was quite unusable as it predicted 100% of test samples to not have hypertension. While this is still a high accuracy model (97%) it is not ideal when comparing more meaningful metrics like the F1 Score, Precision, and Recall, all of which were 0. However, the random forest that was fitted on the balanced data performed much better. While the overall accuracy of predictions dropped a little, this was acceptable as other metrics improved. This model had a really high recall score (0.59) and a precision score of 0.09, to give an F1 metric of 0.15.

As for the details about the training itself, a Cross Validation approach with Grid Search was used, with varying the values of two hyperparameters (number of estimators, and maximum depth of each tree). The best model selected as per Grid Search had 100 estimators and a maximum depth of 3. It must be noted that the model prefers high values of tree depth, but this would only lead to more overfitting and less interpretability (each tree is very dense) so the possible values were constrained.

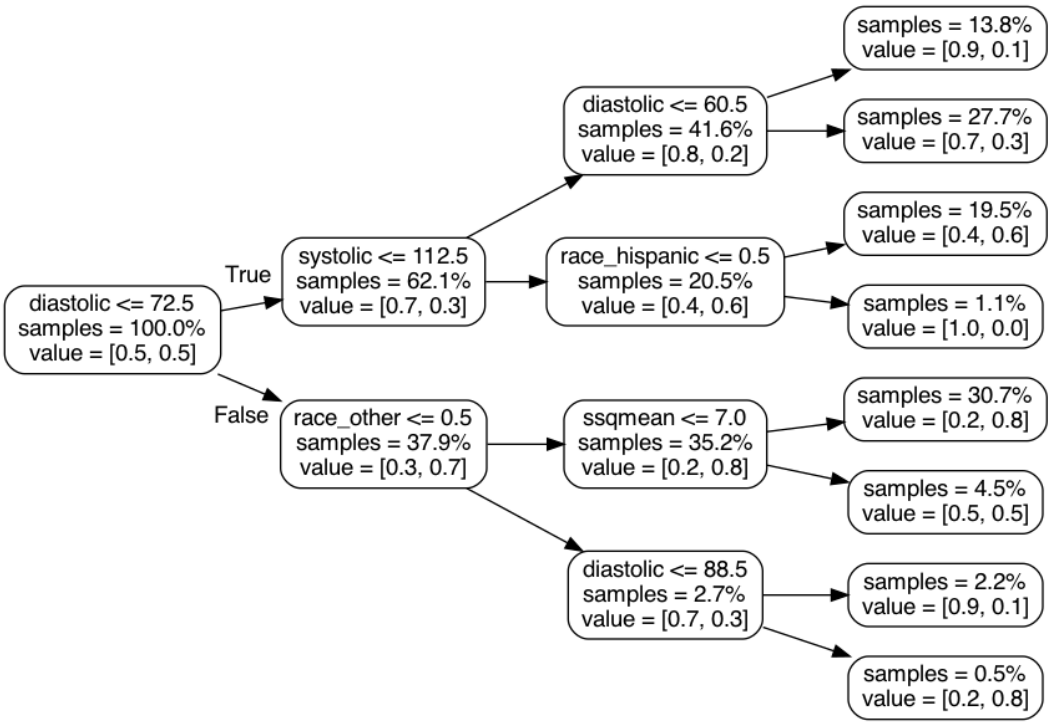



Figure 2: One of the 100 decision trees used by the Random Forest model. Sample represents percentage of data present in that node, and values is the False/True split of having history of hypertension.

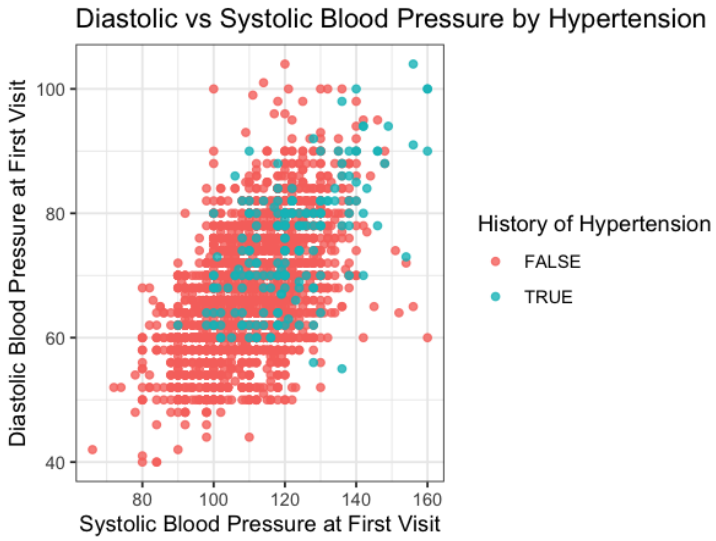



Figure 3: This figure plots the diastolic versus systolic blood pressure of each patient at the time of their first visit, it also displays whether or not the patient has a history of hypertension.

Finally, we present in Table 1, the performance of each model for easy comparison. While there is no single model that holistically outperforms the rest, Adaboost, Random Forest, and MultiLayer Perceptron are more promising than the others. Adaboost has the best F1 Score which is the metric of interest, and as described in Figure 1, deems the diastolic, systolic, and race variables among others to be most important in the prediction. This is reiterated by the Random Forest model as well, as Figure 2 shows decisions being based on these same variables. Moreover, Figure 3 depicts the

relationships between the systolic and diastolic variables while also depicting the history of hypertension. Finally, we also learned that upsampling the data is particularly helpful in building a better predictive model.

Table 1: Model Metrics on the Test Data

| **Model** | **Recall** | **Precision** | **F1 Score=2*Precision*Recall/ (Precision+Recall)** |
|---|---|---|---|
| AdaBoost | 0.3333 | 0.0845 | 0.1348 |
| AdaBoost w/ minority class upsampled to match majority | 0.3030 | 0.1408 | 0.1923 |
| Random Forest w/ upsampling | 0.59 | 0.09 | 0.15 |
| MLP | 0.0845 | 0.4000 | 0.1395 |
| MLP with minority class upsampled to match majority | 0.3521 | 0.1179 | 0.1767 |
| Logistic Regression (Full Model) | 0.25 | 0.0281 | 0.0506 |
| Logistic Regression (Forward/Backward Selection) | 0.2222 | 0.0282 | 0.05 |
| KNN | 0.08451 | 0.10 | 0.09160 |
| KNN with upsampling | 0.09859 | .09836 | 0.09091 |

## Conclusion

The goal of this analysis was to be able to predict the history of hypertension in pregnant women with no previous pregnancy lasting 20 weeks-0 days or more estimated gestational age (nulliparas). While trying to predict the history of hypertension, we provided how various

models and approaches perform in different evaluations, and were able to get an understanding of what factors may contribute to  a history of hypertension.

Results from the logistic regression model (which we treat as baseline) indicate that age, systolic blood pressure, diastolic blood pressure, pre-pregnancy weight,  the mother being born early, history of kidney disease and history of PCOS are all associated with having a history of hypertension. Further, the odds of having hypertension is lower if you are white or hispanic compared to other races. The best performing Adaboost model is mostly in agreement with these findings, as it picks systolic, diastolic, and race as some of its most important variables in predicting the outcome.

A limitation of this analysis is the class imbalance of our data. The majority of the patients in this data set do not have a history of hypertension. This leads to issues as our primary focus is to predict whether a patient has a history of hypertension. While this limitation is addressed by the upsampling techniques, it is definitely not comparable to having balanced data in the first place.

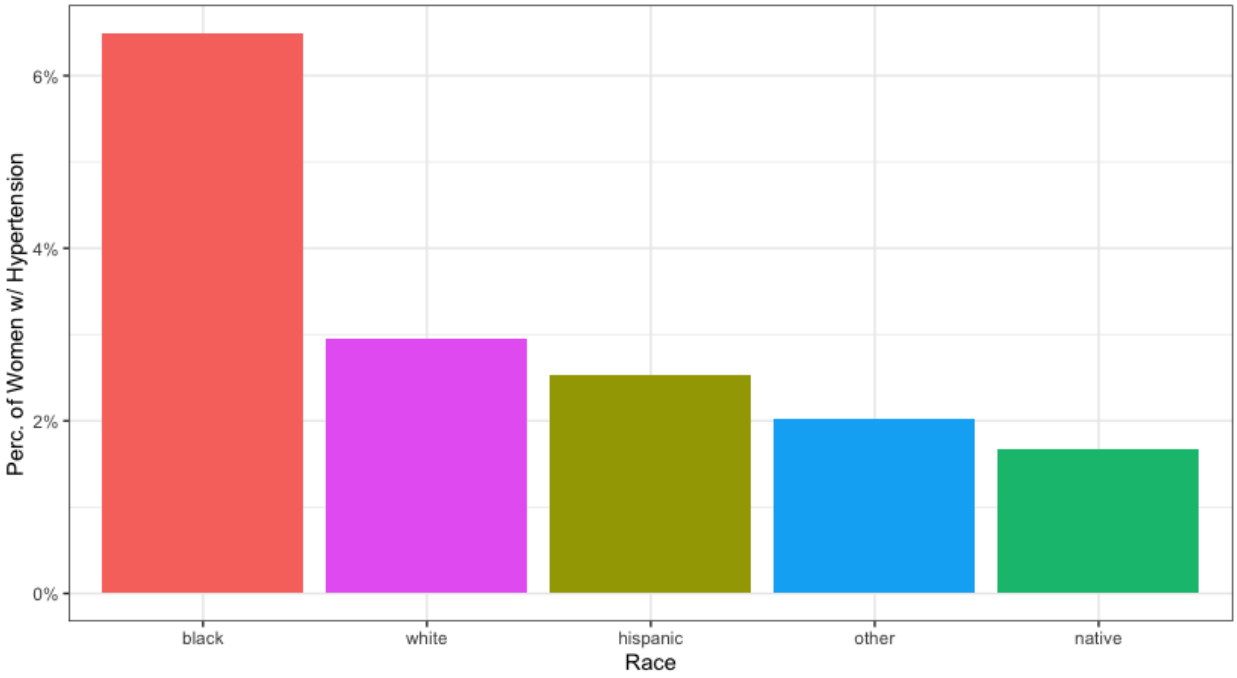
## **Appendix**:

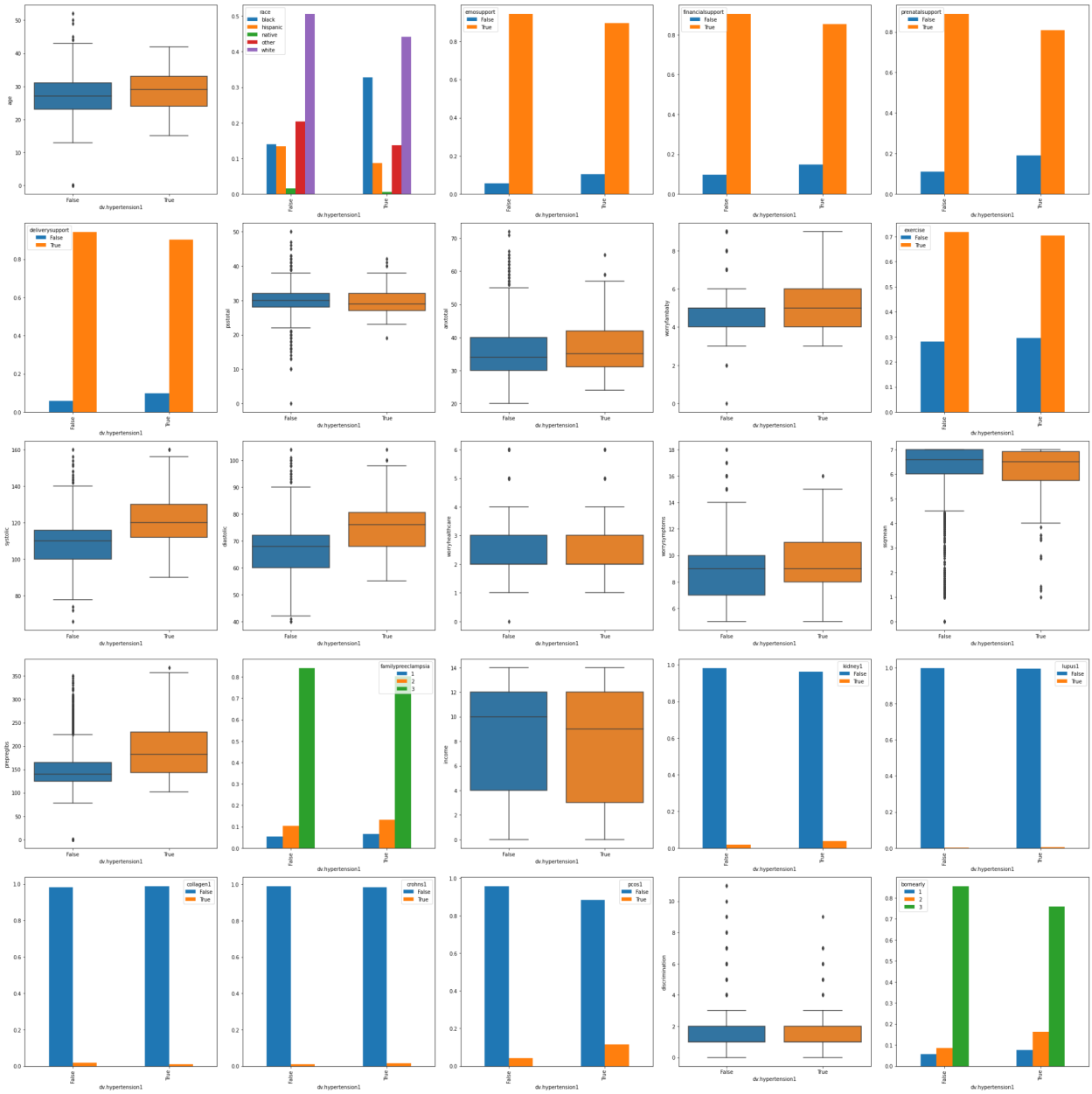
Table 2: Baseline Table of Descriptives

| Variable | Number of Unique Factors | Top Factor Counts or IQR | Mean or Proportion |
|---|---|---|---|
| race | 5 | whi: 4003, oth: 1588, bla: 1155, his: 1069 | NA |
| worryfambaby | 9 | 4: 2663, 5: 2224, 3: 1208, 6: 1117 | NA |
| worryhealthcare | 7 | 2: 4464, 3: 1983, 4: 986, 5: 342 | NA |
| worrysymptoms | 15 | 8: 1516, 9: 1416, 7: 1209, 10: 1102 | NA |
| familypreeclampsia | 3 | 3: 6681, 2: 838, 1: 415 | NA |
| discrimination | 12 | 1: 4493, 2: 2036, 3: 524, 0: 303 | NA |
| bornearly | 3 | 3: 6791, 2: 699, 1: 444 | NA |
| emosupport | 2 | TRU: 7470, FAL: 464 | 0.9415 |
| financialsupport | 2 | TRU: 7137, FAL: 797 | 0.8995 |
| prenatalsupport | 2 | TRU: 7026, FAL: 908 | 0.8856 |
| deliverysupport | 2 | TRU: 7461, FAL: 473 | 0.9404 |
| exercise | 2 | TRU: 5703, FAL: 2231 | 0.7188 |
| dv.hypertension1 | 2 | FAL: 7680, TRU: 254 | 0.032 |
| kidney1 | 2 | FAL: 7798, TRU: 136 | 0.0171 |
| lupus1 | 2 | FAL: 7917, TRU: 17 | 0.0021 |
| collagen1 | 2 | FAL: 7800, TRU: 134 | 0.0169 |
| crohns1 | 2 | FAL: 7865, TRU: 69 | 0.0087 |
| pcos1 | 2 | FAL: 7575, TRU: 359 | 0.0452 |
| age | NA | (23,31) | 27.2328 |
| psstotal | NA | (28,32) | 29.7545 |

| | | | |
|---|---|---|---|
| anxtotal | NA | (30,40) | 35.4239 |
| systolic | NA | (100,118) | 109.1274 |
| diastolic | NA | (60,72) | 67.1273 |
| ssqmean | NA | (6,7) | 6.2057 |
| prepreglbs | NA | (125,168) | 151.7918 |
| income | NA | (4,12) | 7.95 |

## Exploratory Data Analysis:





age
dv.hypertension1
False
True
race
black
hispanic
native
other
white
emosupport
financialsupport
prenatalsupport
deliverysupport
pristotal
anxiotal
worryfambaby
exercise
systolic
diastolic
worryhealthcare
worrysymptoms
sixminwk
prepregbs
familypreeclampsia
income
kidney1
lupus1
collagen1
crohns1
pcos1
discrimination
bornearly

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

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

```r
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:

```r
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
```

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

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

## Area under the curve: 0.5128
```

```r
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
```

```r
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
##
```

```r
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
##         0.2222222222         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
```

```r
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, Rand
omizedSearchCV, 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_preci
sion_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, AdaBoost
Classifier
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', 'v
1epdstotal'], 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]:

```
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, rand
om_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"]].re
place({'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([
    ('impoter', 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]:

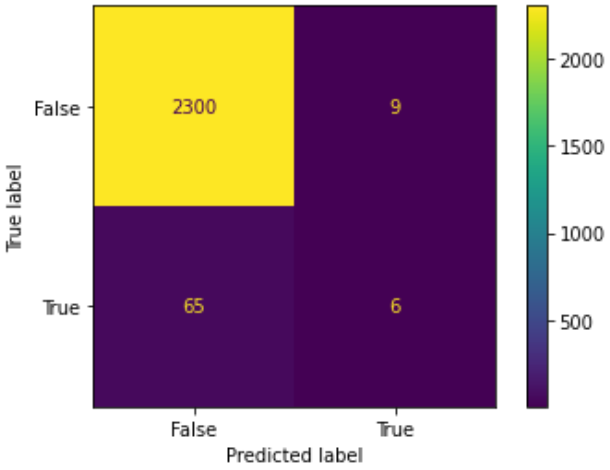
```
x, y, x_train, y_train, x_test, y_test = data_preporcess(df, cat=True)
best_model_MLP = best_model_select_MLP(x_train, y_train, x_test, y_test)
```

```
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: Fu
nction plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecate
d 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([
    ('impoter', 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
```

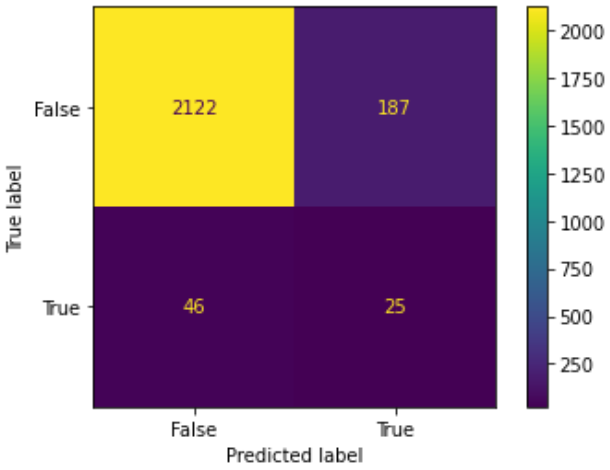
```
/usr/local/lib/python3.7/dist-packages/imblearn/utils/_validation.py:591: FutureWarning:
Pass sampling_strategy=1.0 as keyword args. From version 0.9 passing these as positional
arguments will result in an error
  FutureWarning,
```

```
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)
```

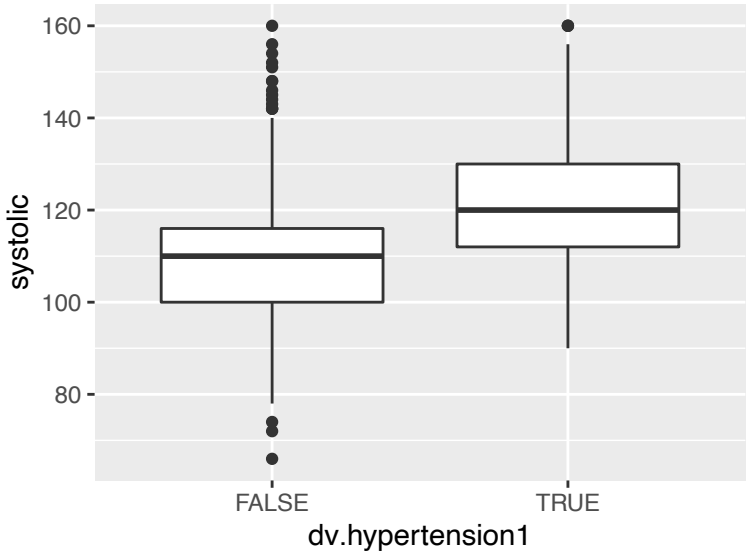
## EDA

```
summary(train)
```

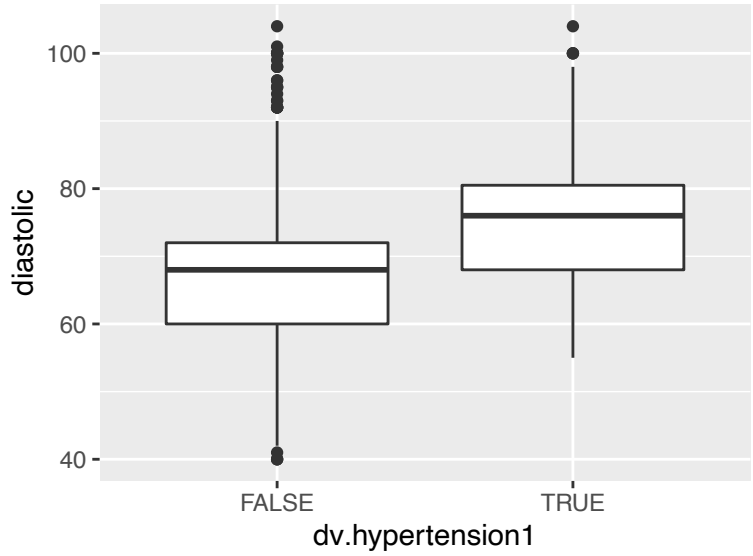
```
##       age         emosupport      financialsupport prenatalsupport
##  Min.   : 0.00   Mode :logical   Mode :logical    Mode :logical
##  1st Qu.:23.00   FALSE:323       FALSE:546        FALSE:633
##  Median :28.00   TRUE :5231      TRUE :5008       TRUE :4921
##  Mean   :27.11
##  3rd Qu.:31.00
##  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
```

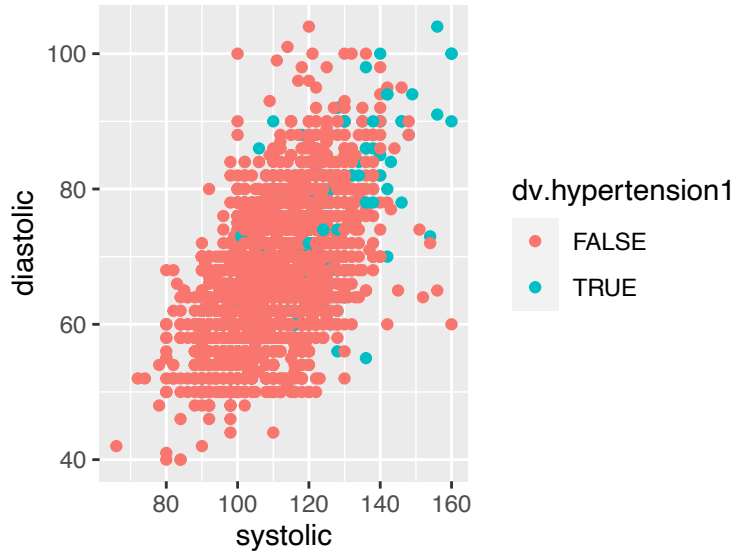
```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = systolic))
```



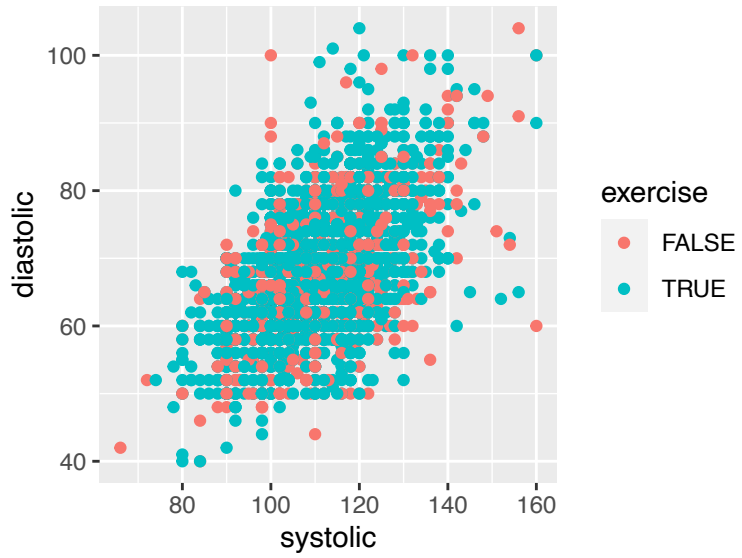

```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = diastolic))
```



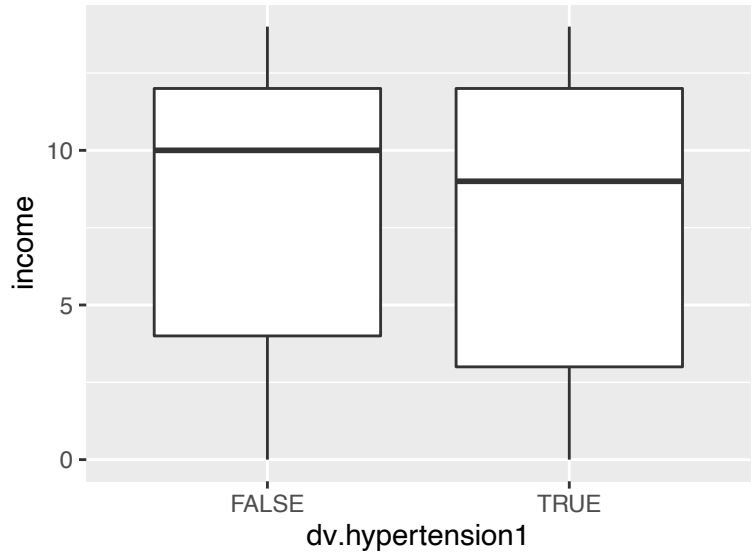

```
ggplot(data = train) + geom_point(mapping = aes(x = systolic, y = diastolic, color = dv.hypertension1))
```



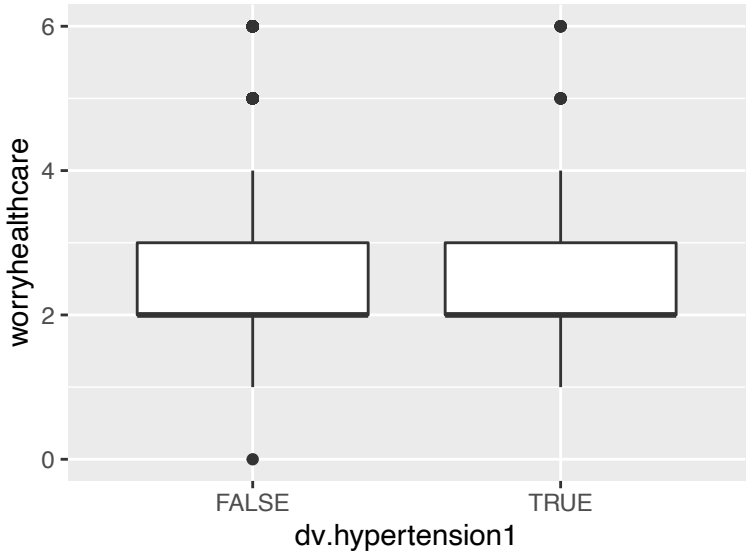

```
ggplot(data = train) + geom_point(mapping = aes(x = systolic, y = diastolic, color = exercise))
```



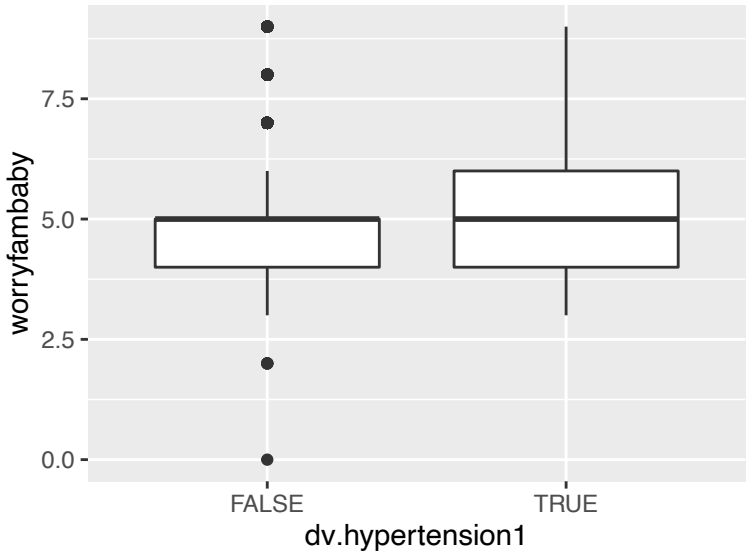

```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = income))
```



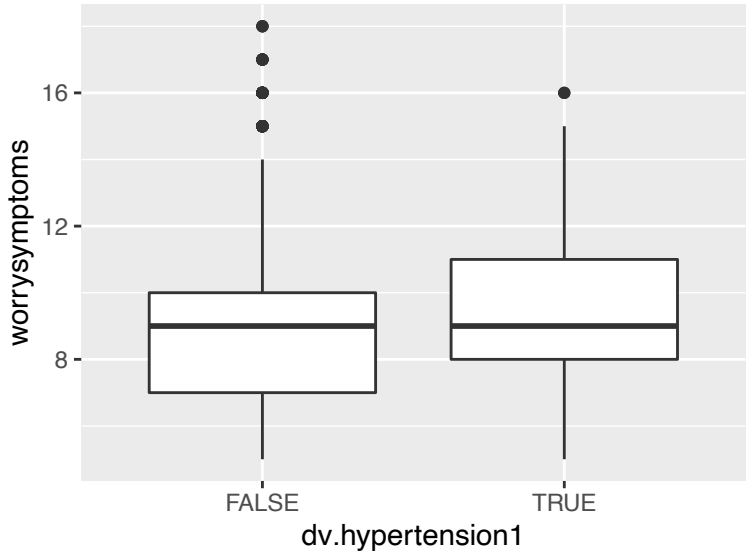

```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = worryhealthcare))
```



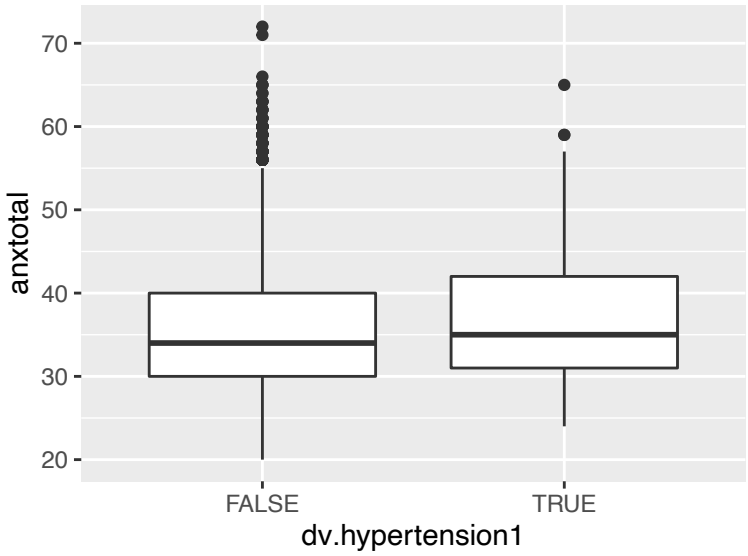

```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = worryfambaby))
```



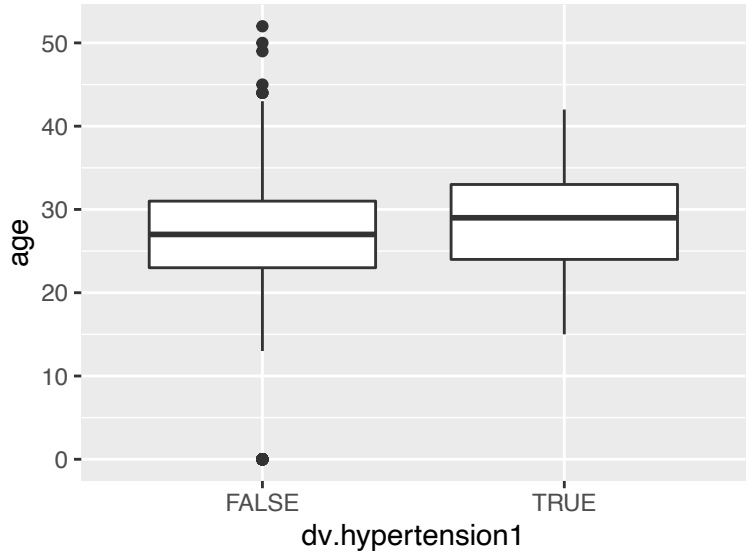

```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = worrysymptoms))
```



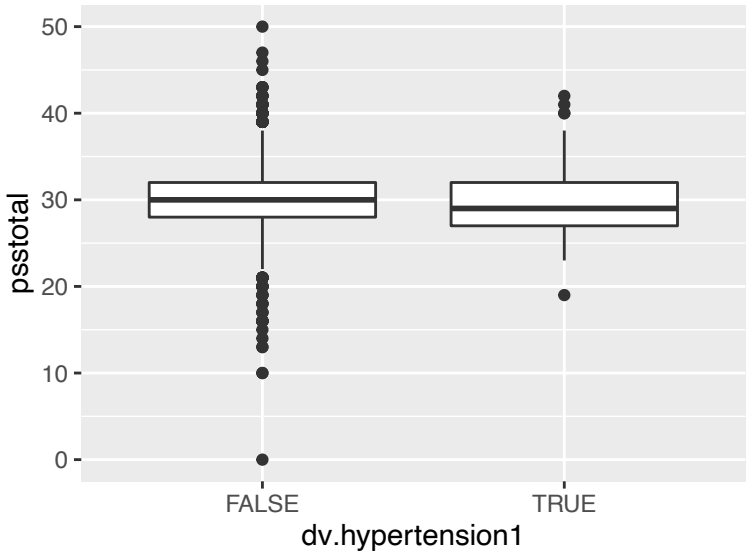

```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = anxtotal))
```



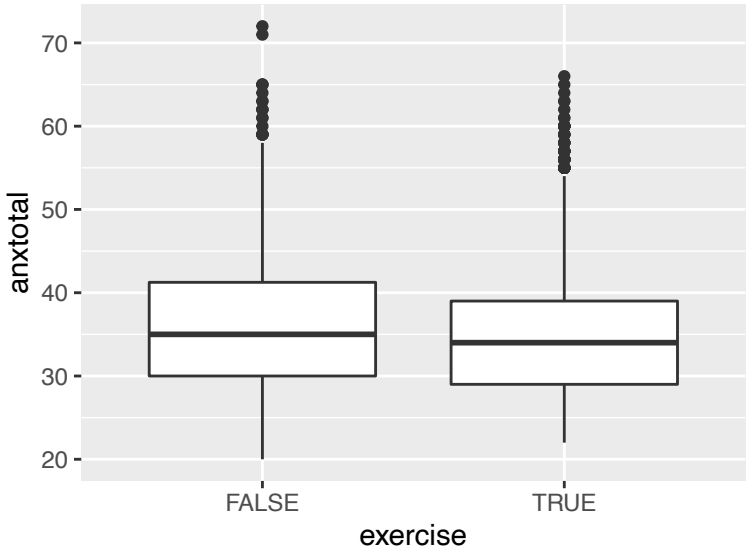

```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = age))
```



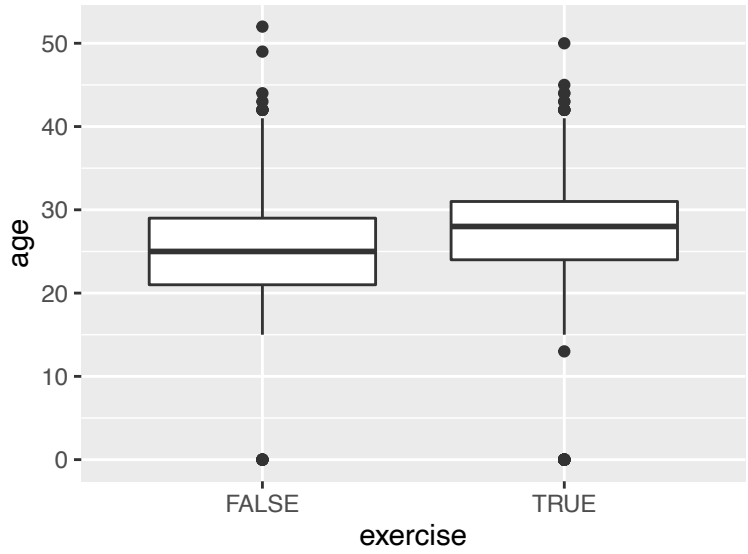

```
ggplot(data = train) + geom_boxplot(mapping = aes(x = dv.hypertension1, y = psstotal))
```



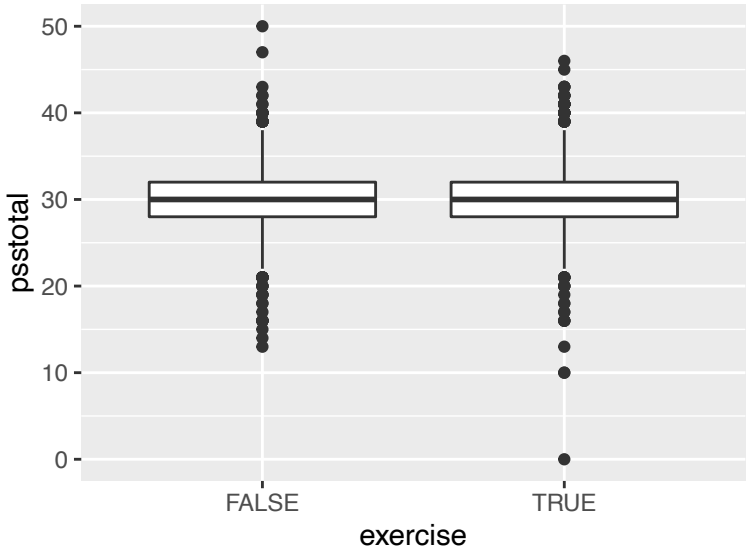

```
ggplot(data = train) + geom_boxplot(mapping = aes(x = exercise, y = anxtotal))
```




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




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




## 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)
}
```

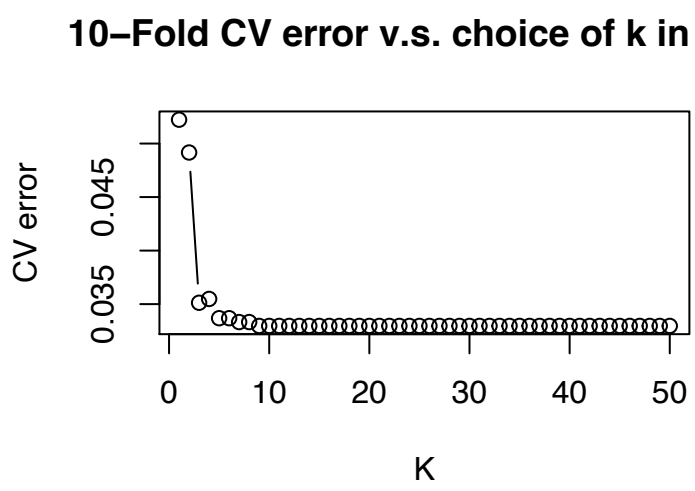
```
min(cv_error)
```

```
## [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 er
```




```
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)
}
```

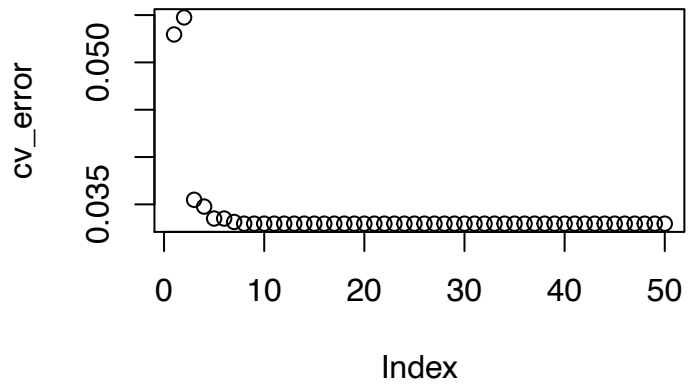
```
print(min(cv_error))
```

```
## [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
##
```

**Created by Bradley Velasquez on Friday, November 11, 2022 at 11:49 AM**
https://umich.enterprise.slack.com/files/U02CR94U41E/F04AYP9SKRP/model_script.r

```
##############LIBRARY AND FUNCTIONS##############
#For cleaning and reading data
library(tidyverse)
library(caret)
library(themis)
library(EZtune)
library(MLmetrics)

#
# library(tidymodel)
#for plotting
theme_set(theme_bw())


#change wd, and import data
# setwd("~/Documents/Box Sync/Statistics Master/Fall 2022/STATS504/hw5")
setwd("/Users/brad/Downloads/hw5")
df <- read.csv("data/nuMoM2bsubset.csv")
# load("brad_models.RData") # Saved my global enviornment just in case...

############ Groups 4, 9, 14 outcome: dv.hypertension1

null_outcomes <-c("dv.diabetes1",
                  "dv.v1epdstotal",
                  "dv.gestweeks",
                  "dv.preeclampsia")

df <- df[,!(names(df) %in% null_outcomes)]

######## DATA CLEANING #########
test_df = df %>% summarise(across(everything(), list(min,max)))
test_df = t(test_df)

#Convert booleans to "TRUE"/"FALSE"
df$emosupport <- df$emosupport == 1
df$financialsupport <- df$financialsupport==1
df$prenatalsupport <- df$prenatalsupport == 1
df$financialsupport <- df$financialsupport ==1
df$deliverysupport <- df$deliverysupport ==1
df$exercise <- df$exercise == 1
df$dv.hypertension1 <- df$dv.hypertension1==1
df$kidney1 <- df$kidney1==1
df$lupus1 <- df$lupus1 == 1
df$collagen1 <- df$collagen1==1
df$crohns1 <- df$crohns1 == 1
df$pcos1 <- df$pcos1 == 1


#Three level factors....
df$familypreeclampsia <- as.factor(df$familypreeclampsia)
df$bornearly <- as.factor(df$bornearly)



#Higher level factors....
df$worryfambaby <- as.factor(df$worryfambaby)
```

```r
df$worryhealthcare <- as.factor(df$worryhealthcare)
df$worrysymptoms <- as.factor(df$worrysymptoms) #what are the levels for this? There should be a codebook.
df$discrimination <- as.factor(df$discrimination)
df$race <- as.factor(df$race)

# skimr::skim(df)
############SUMMARY TABLE###########
skimmed_df = skimr::skim(df)
skimmed_df$n_missing = NULL
skimmed_df$complete_rate= NULL
skimmed_df$numeric.hist = NULL
skimmed_df$factor.ordered = NULL

#For factors
skimmed_df$factor.top_counts[8:18] <- skimmed_df$logical.count[8:18]
skimmed_df$factor.top_counts[8:18] <- skimmed_df$logical.count[8:18]
skimmed_df$factor.n_unique[8:18] <- 2
skimmed_df$numeric.mean[8:18] <- skimmed_df$logical.mean[8:18]
skimmed_df$numeric.mean <- round(skimmed_df$numeric.mean, digits= 4)
skimmed_df$logical.mean <- NULL
skimmed_df$logical.count = NULL
skimmed_df$factor.top_counts[19:26] <- paste0("
(",skimmed_df$numeric.p25[19:26],",",skimmed_df$numeric.p75[19:26],")")
skimmed_df$numeric.sd <- NULL
skimmed_df$numeric.p0 <- NULL
skimmed_df$numeric.p25 <- NULL
skimmed_df$numeric.p50 <- NULL
skimmed_df$numeric.p75 <- NULL
skimmed_df$numeric.p100 <- NULL
# write.csv(skimmed_df, "data/baseline.csv")


rm(test_df, null_outcomes) #drop unused items

######IMPUTE VALUES########
colSums(df == 0)
#age
#psstotla
#ssqmean
# prepreglbs

df$age[df$age==0] <- mean(df$age[df$age!=0])
df$prepreglbs[df$prepreglbs==0] <- mean(df$prepreglbs[df$prepreglbs!=0])

########TEST TRAIN###############
set.seed(1123)
size = floor(0.3*dim(df))

id = sample(c(1:7934),replace=F, size = floor(0.3*dim(df))[1])
train<-df[-id,]
test<-df[id,]

# write.csv(test,"test_df.csv")
# write.csv(train,"train_df.csv")

#########ENCODE TRAIN#########
dmy <- dummyVars(" ~ .", data = train)
train_hot <- data.frame(predict(dmy, newdata = train))

train_hot_X = train_hot[,!(names(train_hot) %in% c("dv.hypertension1FALSE","dv.hypertension1TRUE"))]
```

```

#Outcome variable needs to have a valid name. Use make.names() or use
train_hot_y = factor(train_hot$dv.hypertension1TRUE,
                     levels = c(1,0),
                     labels = c("yes","no"))



#Combined data frame .....
train_hot_X_y<- train_hot_X
train_hot_X_y$dv.hypertension <- train_hot_y


#####UP SAMPLE TRAINING DAT########
#minority class has 50% of observations as majoirty
train_up50<- smote(train_hot_X_y, var ="dv.hypertension", over=0.5, k=10)
table(train_up50$dv.hypertension)
#majority class has 100% of observations as majority
train_up100<- smote(train_hot_X_y, var ="dv.hypertension",  over=1, k=10)
table(train_up100$dv.hypertension)

#Export
# write.csv(train_up50, "data/train_one_hot_50.csv")
# write.csv(train_up100, "data/train_one_hot_100.csv")




#########columns with no variance in training data ############
X_no_var = nearZeroVar(train_hot_X_y)
X_no_var_names = names(train_hot_X_y)[X_no_var]
X_no_var_names = X_no_var_names[-c(1,37)] #hypertension has low variance, as does race == native

#Drop columns with no variance....
train_hot_X_y <- train_hot_X_y[,!(names(train_hot_X) %in% X_no_var_names)]
train_up50 <- train_up50[,!(names(train_up50) %in% X_no_var_names)]
train_up100 <- train_up100[,!(names(train_up100) %in% X_no_var_names)]

# Check the distribution of these variables....
# train_hot_X[,X_no_var_names] %>% skimr::skim()

############DOWNSAMPLE#############
set.seed(1123)
train_hot_down = downSample(x=train_hot_X_y,
                            y=train_hot_X_y$dv.hypertension)
train_hot_down$Class <- NULL

##############ENCODE TEST##########
dmy <- dummyVars(" ~ .", data = test)
test_hot <- data.frame(predict(dmy, newdata = test))

test_hot_X = test_hot[,!(names(test_hot) %in% c("dv.hypertension1FALSE","dv.hypertension1TRUE"))]

#Outcome variable needs to have a valid name. Use make.names() or use
test_hot_y = factor(test_hot$dv.hypertension1TRUE,
                    levels = c(1,0),
                    labels = c("yes","no"))

#combine into one data frame....
test_hot_X_y = test_hot_X
```

```r
test_hot_X_y$dv.hypertension = test_hot_y

#Drop columns with no variance
test_hot_X_y <- test_hot_X_y[,!(names(test_hot_X) %in% X_no_var_names)]


########CLEAN WORKSPACE############
rm(list = c(#"test_hot_X_sub",
            "test_hot_X",
            "test_hot",
            "test_hot_y",
            # "train_hot_X_sub",
            "train_hot_X",
            "train_hot",
            "train_hot_y",
            "dmy"))

############ HYPERPARAMETER TUNING##############

models <- caret::modelLookup() #what models are in the caret package?

########## Adaboost.M1############

#!!!!!!!!!!!!!! WARNING THE FOLLOWING CHUNKS TAKE ~40 Minutes to run!!!!!!!!!!!!!!!!

fitGrid_ada <- expand.grid(mfinal = c(1,6,9,100),
                           # mfinal = (1:3)*3,
                           # maxdepth = c(1:3),
                           maxdepth = c(1,2,4),
                           coeflearn = c("Breiman"))

fitControl_ada <- trainControl(method = "repeatedcv",
                               repeats = 5,
                               classProbs = T,
                               # summaryFunction = twoClassSummary,
                               summaryFunction = prSummary)
#on up sampled

# using the adaboost.m1 package....
set.seed(1123)
start_time = Sys.time()
ada.mod <- train(x=train_hot_X_y[,-48],
                 y= train_hot_X_y$dv.hypertension,
                 method = 'AdaBoost.M1',
                 trControl = fitControl_ada,
                 tuneGrid = fitGrid_ada,
                 metric = "AUC",
                 verbose = TRUE)
total_time <- Sys.time() - start_time
total_time

# Upsampled to be 50% majority class
set.seed(1123)
start_time <- Sys.time()
ada50.mod <- train(x=train_up50[,-48],
                   y= train_up50$dv.hypertension,
                   method = 'AdaBoost.M1',
                   trControl = fitControl_ada,
                   tuneGrid = fitGrid_ada,
                   metric = "AUC",
```

```
                   verbose = TRUE)
total_time <- Sys.time() - start_time
total_time


# upsampled to Matched classes - change metric to ROC.
set.seed(1123)
total_time <- Sys.time()
ada100.mod <- train(x=train_up100[,-48],
                    y= train_up100$dv.hypertension,
                    method = 'AdaBoost.M1',
                    trControl = fitControl_ada,
                    tuneGrid = fitGrid_ada,
                    metric = "ROC",
                    verbose = TRUE)
total_time <- Sys.time() - start_time
total_time

# On downsampled - use ROC
set.seed(1123)
start_time <- Sys.time()
adadown.mod <- train(x=train_hot_down[,-48],
                     y= train_hot_down$dv.hypertension,
                     method = 'AdaBoost.M1',
                     trControl = fitControl_ada,
                     tuneGrid = fitGrid_ada,
                     metric = "ROC",
                     verbose = TRUE)
total_time <- Sys.time() - start_time
total_time


############# USING GBM PACKAGE ############3

# set.seed(1123)
# start_time <- Sys.time()
# ada.mod <- train(x=train_hot_X_y[,-48],
#                    y= train_hot_X_y$dv.hypertension,
#                    distribution = 'adaboost',
#                    method="gbm",
#                    trControl = fitControl_ada,
#                    tuneGrid = fitGrid_ada,
#                    metric = "AUC",
#                    verbose = TRUE)
# total_time <- Sys.time() - start_time
# total_time

# set.seed(1123)
# start_time <- Sys.time()
# ada50.mod <- train(x=train_up50[,-48],
#                     y= train_up50$dv.hypertension,
#                     distribution = 'adaboost',
#                     method="gbm",
#                     trControl = fitControl_ada,
#                     tuneGrid = fitGrid_ada,
#                     metric = "AUC",
#                     verbose = TRUE)
# total_time <- Sys.time() - start_time
# total_time

```

```
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
357 ######## AUROC #######
358 # library(pROC)
359 # ada.roc <- roc(test_hot_X_y$dv.hypertension, test_predada_p$yes)
360 # # plot(ada.roc, print.thres="best", print.thres.best.method="closest.topleft")
361 # ada50.roc <- roc(test_hot_X_y$dv.hypertension, test_predada50_p$yes)
362 # ada100.roc <- roc(test_hot_X_y$dv.hypertension, test_predada100_p$yes)
363 #
364
365 # plot(ada50.roc, print.thres="best", print.thres.best.method="closest.topleft")
366 # result.coords <- coords(ada.roc, "best", best.method="closest.topleft", ret=c("ppv","tpr"))
367 # print(result.coords)#to get threshold and accuracy
368
369
370 #########CONFUSION METRICS############
371 conf_matada = table("truth"=test_hot_X_y$dv.hypertension,"pred"= test_predada)
372 conf_matada=confusionMatrix(conf_matada, mode = "everything", positive = "yes")
373 conf_matada
374
375 conf_matada50 = table("truth"=test_hot_X_y$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$dv.hypertension,"pred"= test_predada100) #Adaboost looks a bit bette
380 conf_matada100=confusionMatrix(conf_matada100, mode = "everything", positive ="yes")
381 conf_matada100
382
383 conf_matadadown= table("truth"=test_hot_X_y$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
```

```
ggplot(data=df)+
  geom_jitter(aes(x=diastolic,
                  y = systolic,
                  col = dv.hypertension1),
              alpha=0.5)+
  facet_wrap(~dv.hypertension1)+
  scale_x_continuous(sec.axis = sec_axis(~ . ,
                                         name = "Has Hypertension",
                                         breaks = NULL,
                                         labels = NULL))+
  # scale_y_continuous(labels = scales::percent)+
  xlab("Systolic Blood Pressure At First Visit")+
  ylab("Diastolic Blood Pressure At First Visit")+
  guides(col="none")
```

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', 'anxtotal', '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)
```

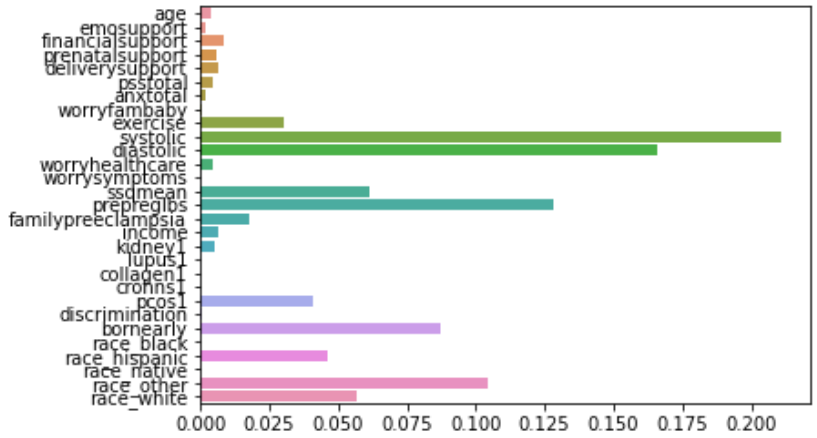
Out[342…
```
0.014084507042253521
```

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.featur
X_train = X[cols + ["race_black", "race_native"]]
X_train_orig = X_orig[cols + ["race_black", "race_native"]]
X_train
```

Out[346…

| | 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 | |
| **10738** | True | 102 | 65 | 6.308870 | 103.279340 | False | 2 | 0 | 1 | 0 | |
| **10739** | True | 103 | 65 | 5.789501 | 184.412336 | False | 2 | 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)
```

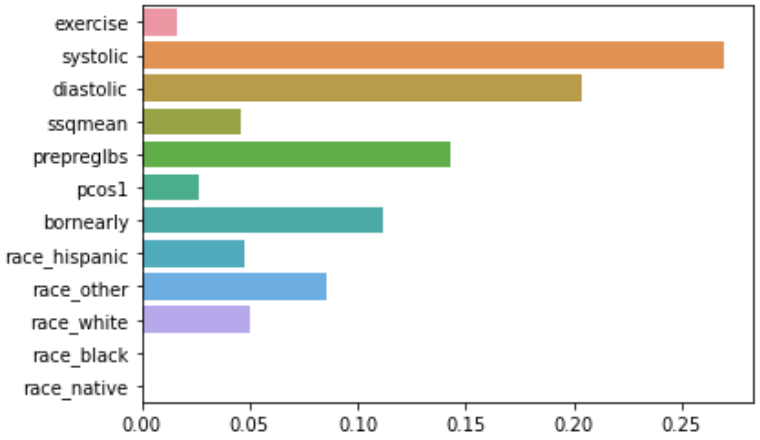
In [392…]
```
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
```

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: UndefinedMetricW
arning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_divisio
n` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```