Detecting Falls of Elderly Persons With Wearable Devices

Austin Murr

1/2/2021

Executive Summary

Motivation

Falls among the elderly population is a serious health concern. In fact, the CDC reports that the mortality rate of falls for people aged 65 and older rose by 30% between 2007 and 2016. Perhaps just as concerning is that while about 20% of geriatric falls results in a serious injury, less than half of elders who experience falls notify their doctors. The advent of wearbale smart devices, however, offers a chance to change this. If data from the accelerometers and vitals sensors in smart watches can be analyzed to distinguish falls from other daily activities, it would allow health care providers or family members to recevie instant alerts of an elderly person's fall. In turn, this would enable a more rapid response to falls that necessitate medical attention.

Data

The dataset used for this paper is publicly available on Kaggle as "Fall Detection Data from China".

The researchers who gathered this data derived it from 14 volunteers, each of whom recorded data for 20 falls and 16 activities of daily living (ADLs). For each of these 2,520 trials, the 4 seconds centered on the moment of greatest acceleration has been sampled from.

In total, there are 7 features with 16382 observations. The features are as follows:

- Activity (factor with levels **Standing, Walking, Sitting, Falling, Cramps,** and **Running**)
- Time of Observation (within 4 second window)
- Blood Sugar
- EEG
- Blood Pressure
- Heart Rate
- Circulation

Goal

The researchers who published this dataset claimed their ML algorithm could identify falls with "sensitivity, specificity, and accuracy all above 95%."

As these are academic researchers who are more skilled than I, the goal for this project will be attaining a model with measurements of sensitivity, specificity, and accuracy above 85%.

Analysis

Data Cleaning

The data given is already in a rather clean format:

ACTIVITY	TIME	SL	EEG	BP	HR	CIRCLUATION
3	4722.92	4019.64	-1600.00	13	79	317
2	4059.12	2191.03	-1146.08	20	54	165
2	4773.56	2787.99	-1263.38	46	67	224
4	8271.27	9545.98	-2848.93	26	138	554
4	7102.16	14148.80	-2381.15	85	120	809
5	7015.24	7336.79	-1699.80	22	95	427

One of the things that can be done, however, is changing the factor levels for `ACTIVITY` into meaningful labels. After referencing the data's documentation, we are able to recode the factors. Additionally, we will expand the column names from their abbreviated states so their meanings become more apparent.

Activity	Time	SugarLevel	EEG	BloodPressure	HeartRate	Circulation
Falling	4722.92	4019.64	-1600.00	13	79	317
Sitting	4059.12	2191.03	-1146.08	20	54	165
Sitting	4773.56	2787.99	-1263.38	46	67	224
Cramps	8271.27	9545.98	-2848.93	26	138	554
Cramps	7102.16	14148.80	-2381.15	85	120	809
Running	7015.24	7336.79	-1699.80	22	95	427

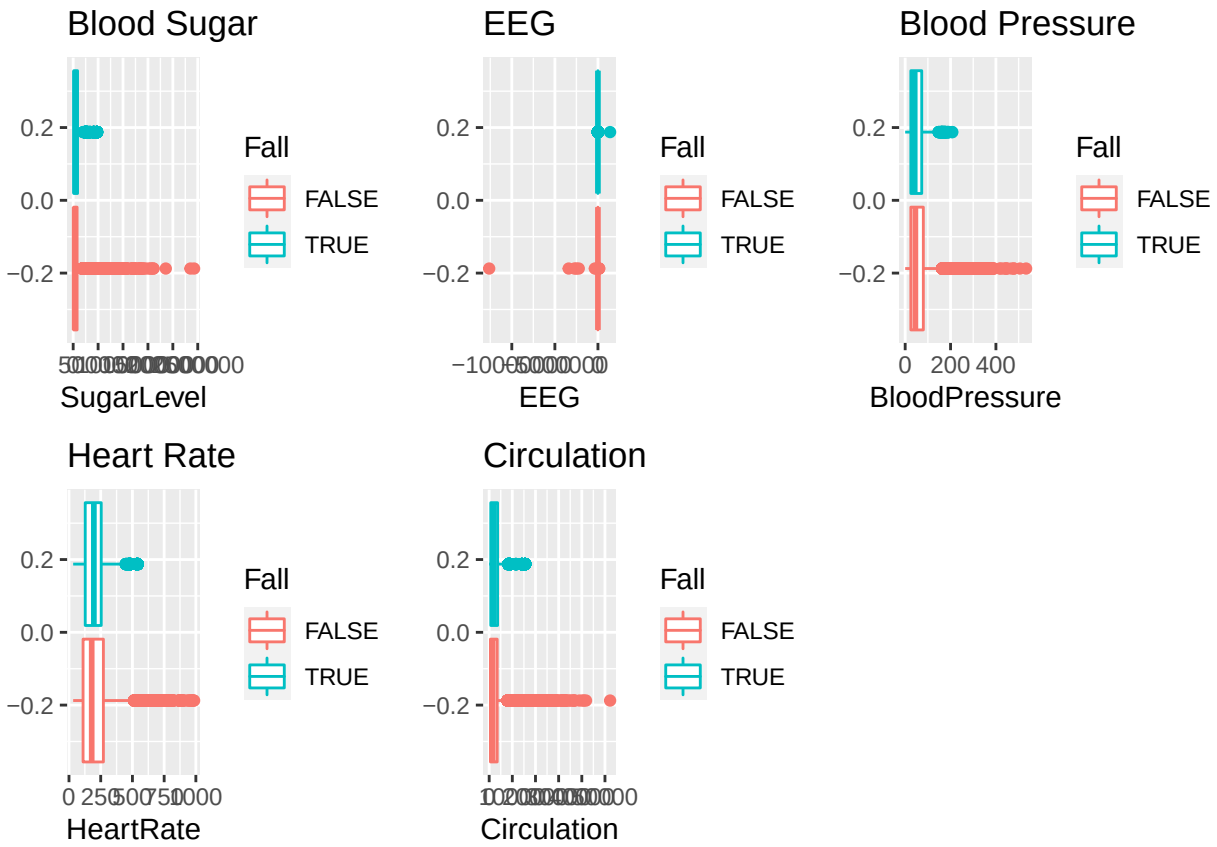
Another step in cleaning the data is simplifying the `Activity` column to a logical vector denoting whether the observation is a fall or not. This is not absolutely necessary, but it does simplify the problem from a multiclass calssification task to a binary classification one.

Activity	Fall
Falling	TRUE
Sitting	FALSE
Sitting	FALSE
Cramps	FALSE
Cramps	FALSE
Running	FALSE

Now it is time to see if we have any outliers that would make any meaningful form of analysis difficult. If so, we should remove those anomalies. Typically, we would use the standard Tukey Outlier Definition:

$$[Q_1 - 1.5 * (Q_3 - Q_1), Q_3 + 1.5 * (Q_3 - Q_1)]$$

This is what the raw data looks like:



As is quite clear, there are quite a few outliers that impact the data, most notably in the `SugarLevel`, `EEG`, and `Circulation` categories.

This table further illustrates the extent of noise in this data, as the Standard Deviation is absurdly large, and the extremes of our data often are unrealistic readings, suggesting questionable sensor performance:

```
## # A tibble: 5 x 7
##   Feature              Min     Max    Mean       SD    IQR SDtoIQR_Ratio
##   <chr>              <dbl>   <dbl>   <dbl>    <dbl>  <dbl>         <dbl>
## 1 SugarLevel          42.2 2426140 75272.  127502.  70820.         1.80
## 2 EEG           -12626000 1410000 -5621.  108221.   3480         31.1
## 3 HeartRate            33     986   212.     130.    152          0.855
## 4 BloodPressure         0     533    58.3     48.3    53          0.911
## 5 Circulation           5   52210  2894.    3826.   2952          1.30
```

The minimum and maximum values most certainly qualify as outliers (a Heart Rate of 986 would be beyond lethal, and same with Blood Pressure reading at 0), but the more alarming issue is comparing the standard deviation to the interquartile range. For normally distributed data, this tends to settle at around 0.74, as shown by this code:

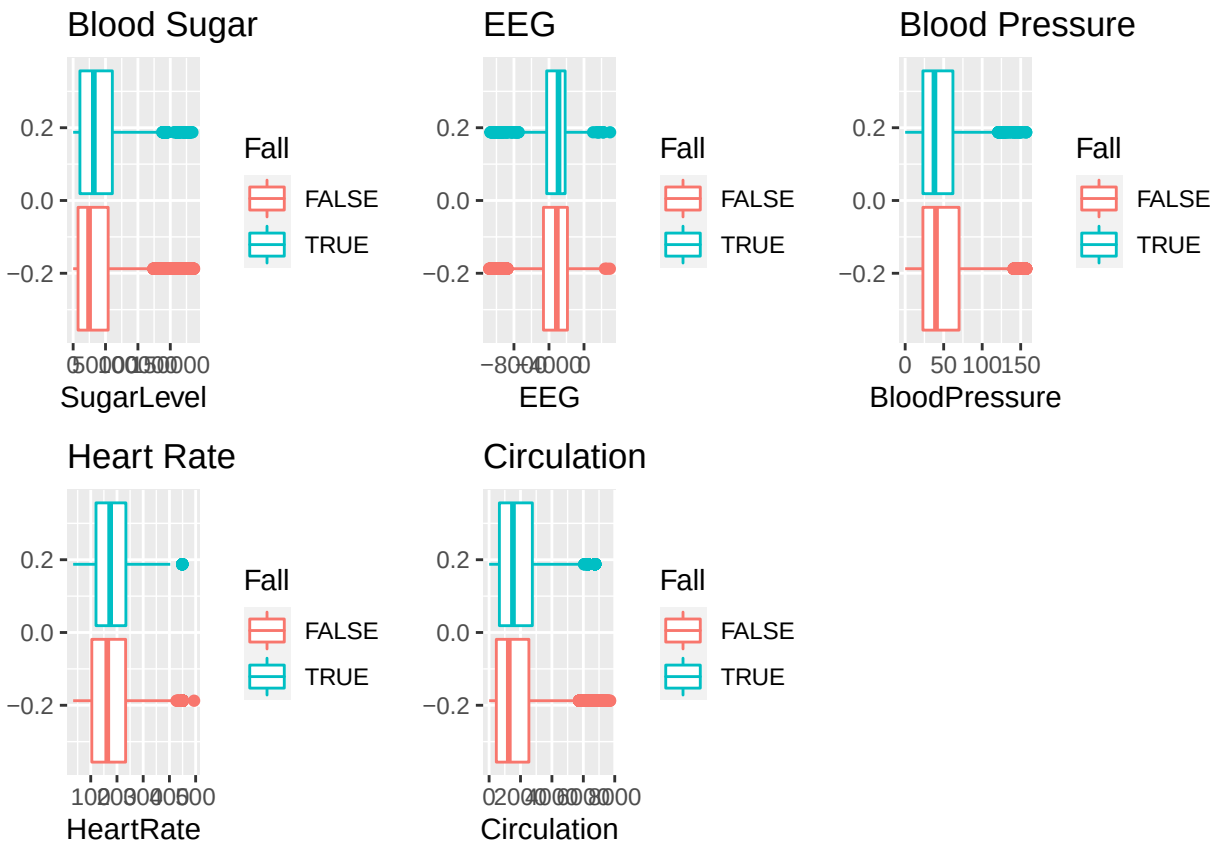
```
d = rnorm(1000000)

tibble(SDtoIQR_Ratio = sd(d) / IQR(d))
```

```
## # A tibble: 1 x 1
##   SDtoIQR_Ratio
```

```
##            <dbl>
## 1          0.742
```

Of course, we would expect this dataset to not follow the normal distribution entirely because there is a physiological reponse occuring in the body over Time, but such large ratios are a red flag to be cautious about the number of outliers.

Simply applying the Tukey outlier filter helps significantly, making our data look like the following:




```
## # A tibble: 5 x 7
##   Feature            Min    Max    Mean      SD    IQR SDtoIQR_Ratio
##   <chr>            <dbl>  <dbl>   <dbl>   <dbl>  <dbl>         <dbl>
## 1 SugarLevel        42.2 186768 39820.  42263.  47889.         0.883
## 2 EEG           -10829     2980 -3441.   2038.   2630.         0.775
## 3 HeartRate         33      494   176.    86.2   127          0.679
## 4 BloodPressure      0      157    48.2   32.7    46          0.711
## 5 Circulation        5     7704  1798.   1656.   2082          0.795
```

The data looks much more usable now.

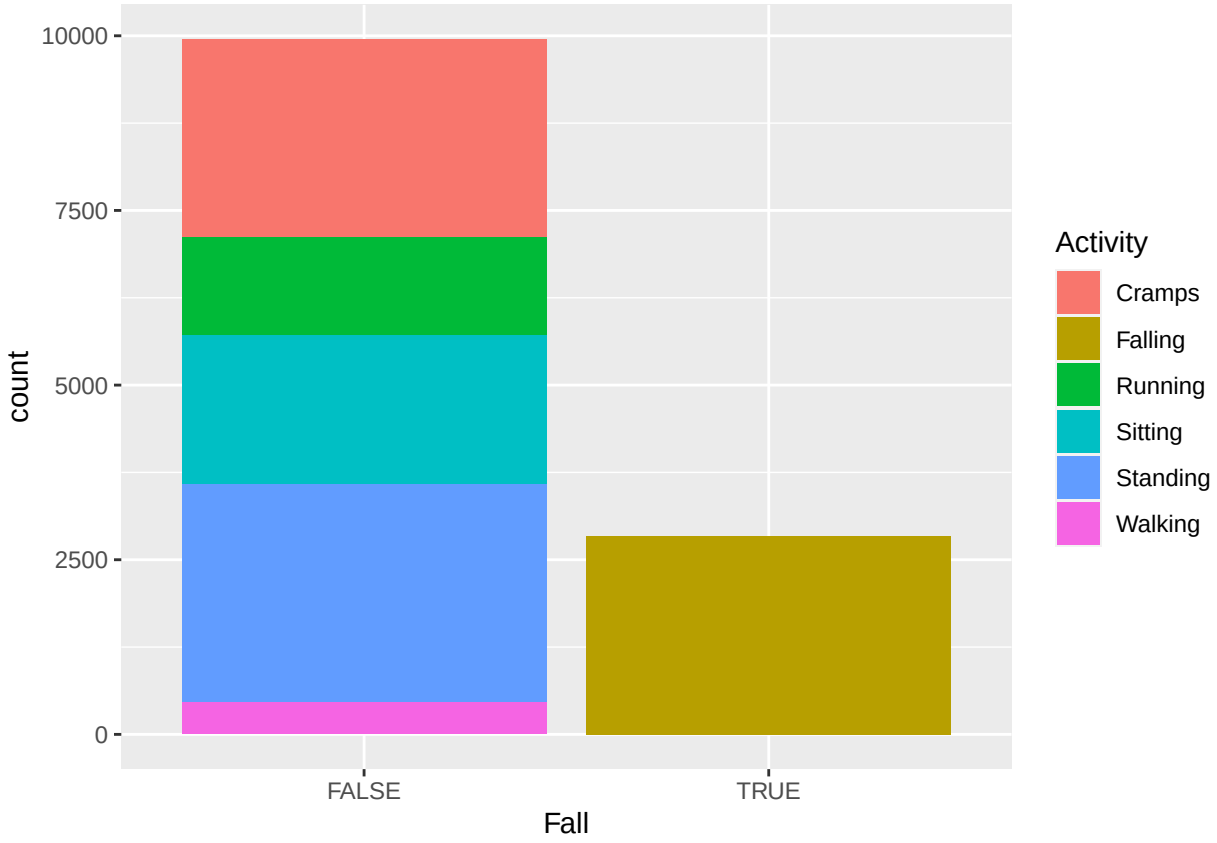
Finally, the dataset needs to be separated into a training set and a test set before continuing onto Exploratory Data Analysis and Visualization. The data will be partitioned so 90% is used for training and 10% is used for testing:

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(falls$Fall, times = 1, p = 0.1, list = FALSE)
```

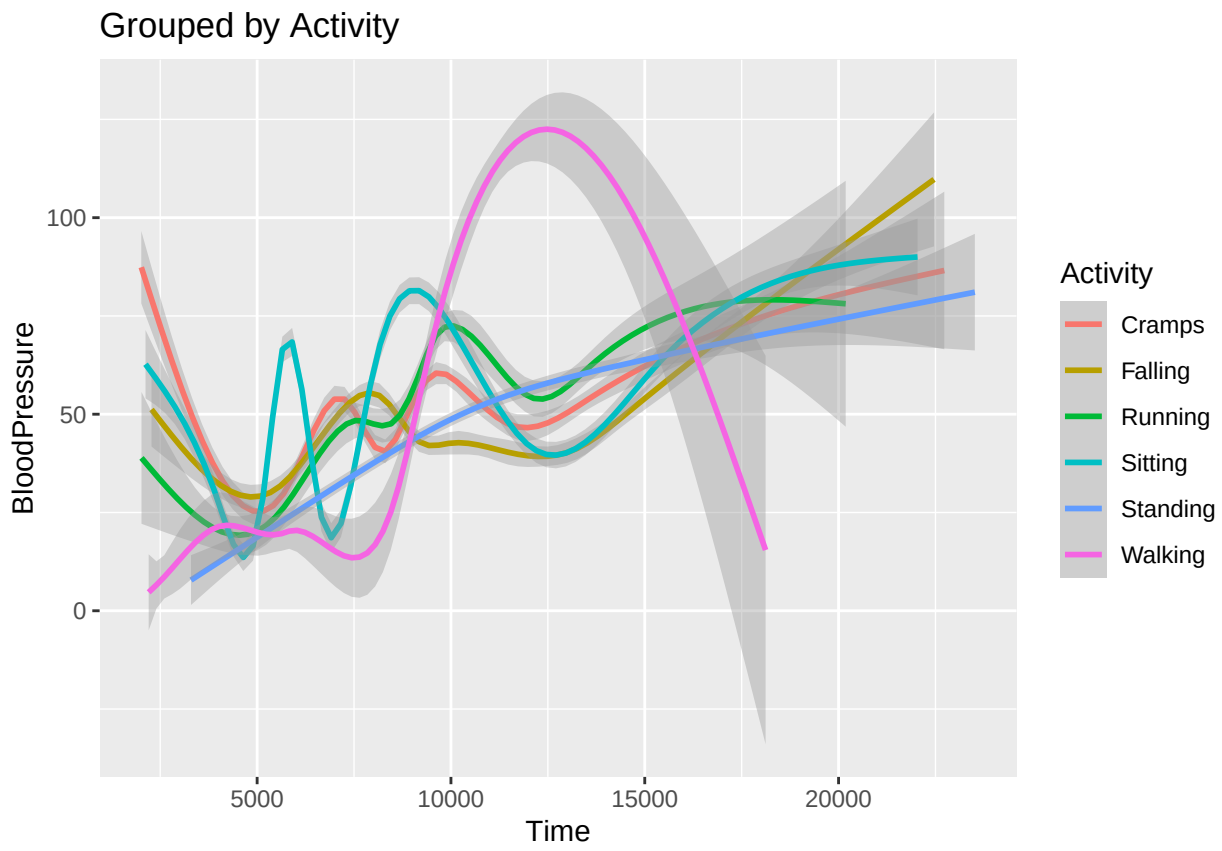
```
train = falls[-test_index,]
test = falls[test_index,]

train = train %>% arrange(Time)
test = test %>% arrange(Time)
```

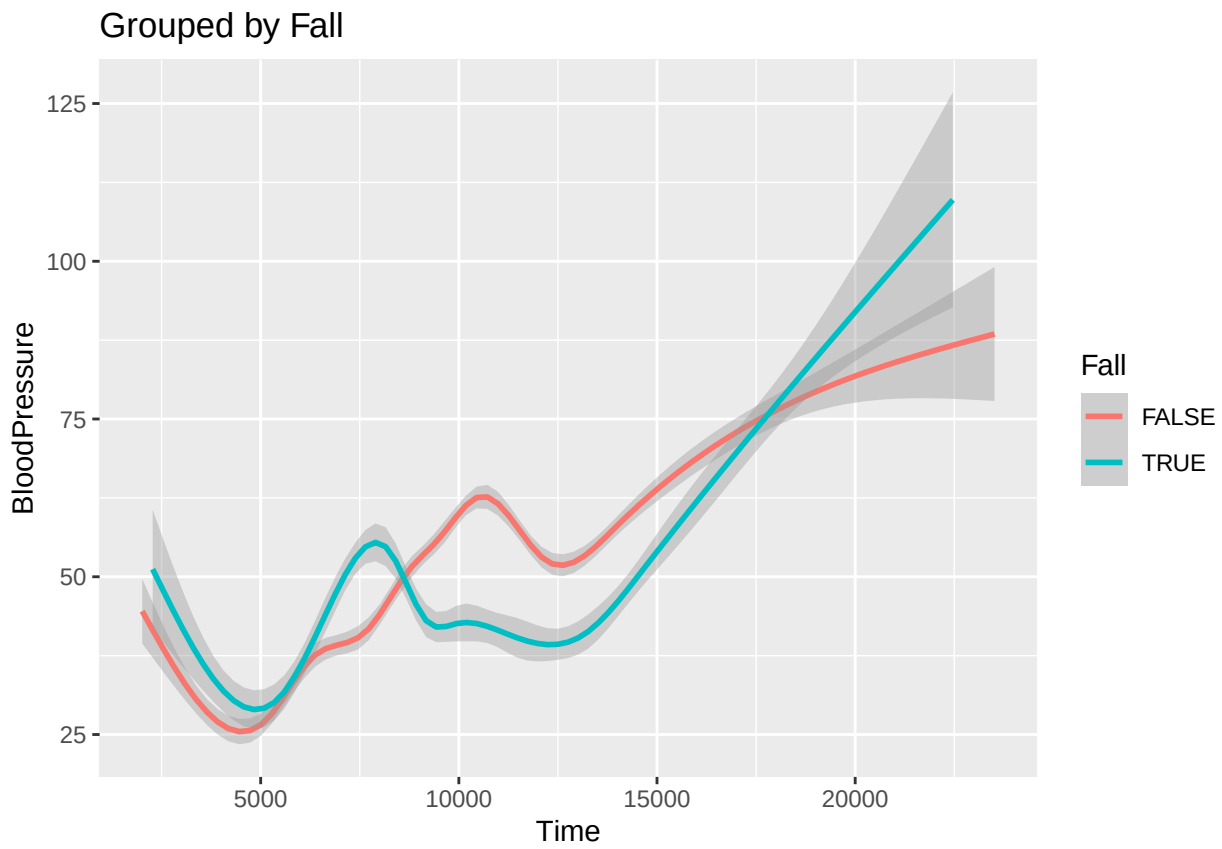
Exploratory Data Analysis

In our training set, there are roughly 4 times the number of non-falls as there are recorded falls. In the figure below, the breakdown of specific activities is given for the broader `Fall` classifications.



As a way to visualize how grouping activities into the binary `Fall` category simplifies our task, the figures below compare the difference between grouping by the 6 factor `Activity` column and grouping according to the binary `Fall` column, respectively. The figures chart the recorded `BloodPressure` versus the `Time`.


Grouped by Activity
BloodPressure
100
50
0
5000
10000
15000
20000
Time
Activity
Cramps
Falling
Running
Sitting
Standing
Walking



As is clearly demonstrated, binary classification reveals trends that are easier to decipher.

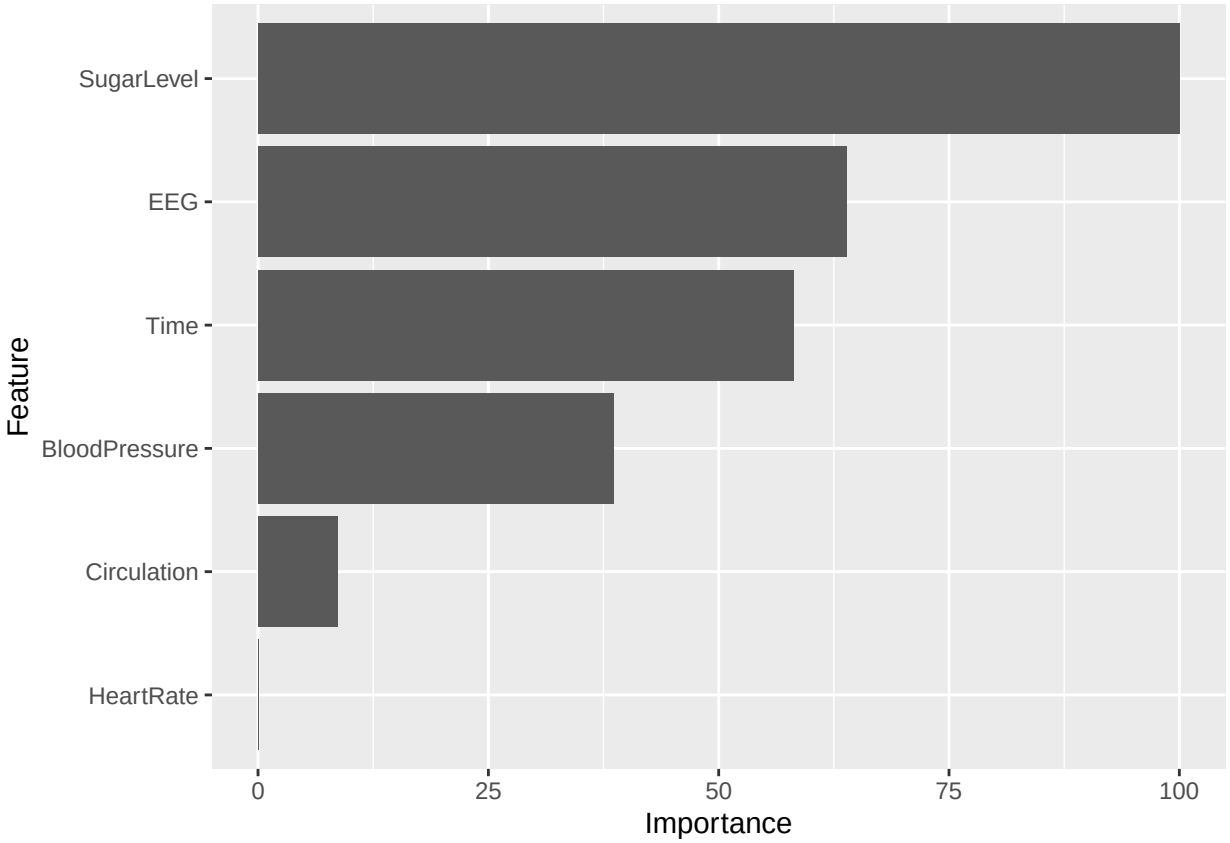
These graphs depict the general patterns for each feature plotted against time. For the most part, there are no extraordinary differences between the trends for both Falls and non-Falls. Predictions, then, will likely have to rely on a number of features.

To ascertain a quantitative value of each feature's predictive power, we will train a Random Forest and analyze the variable importance assigned to each feature.

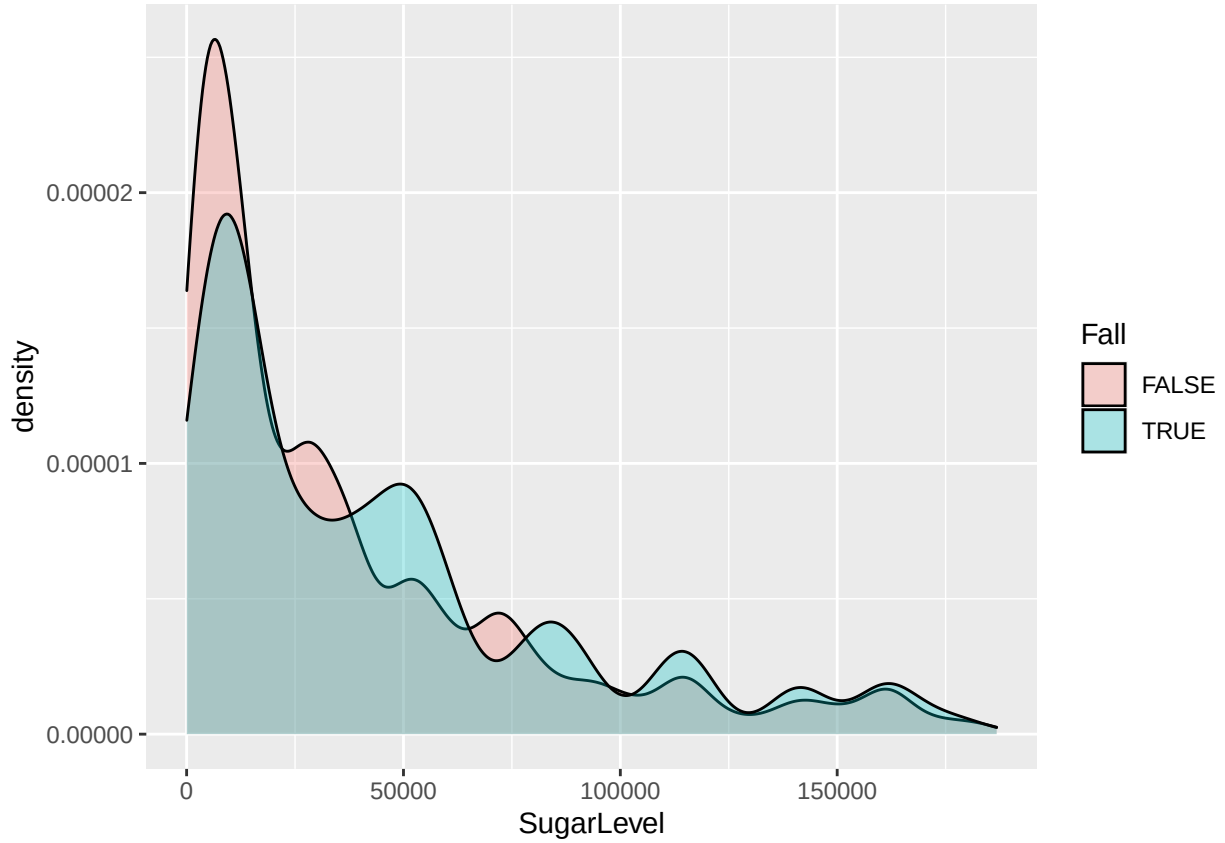
```
set.seed(2021, sample.kind = "Rounding")
train_control = trainControl(method="cv", number=10, savePredictions = TRUE)
rf_fit = train(
  factor(Fall) ~ .,
  data = train,
  method = "rf",
  trControl = train_control
  )
```

The best performing model trained had an overall accuracy of 86.7%.

The variable most important to the random forest's success was blood's `SugarLevel`. This seems rather surprising given the more variant chart of `BloodPressure`. Even more surprising is *how* important it actually was. To illustrate, here is a graph charting the relative importance of each variable:



Given the shocking importance of `SugarLevel`, it might be insightful to train another model with this as the primary predictor. Let's take a deeper look at the distribution of `SugarLevel`.



It appears that measurements of Blood Sugar tend to coalesce into density groups, as testified to by the humps in the density plot. Moreover, when grouped by `Fall` type, there appear to be some regions that are more likely to host a certain type of Fall as opposed to the other.

This information will be important for fine-tuning our models.

Model Building

So far, our first Random Forest model had an accuracy of 86.7%, which is rather good for a first shot.

Let's take a look at where it struggled the most.

```
## # A tibble: 2 x 8
##   Fall  Count  Time SugarLevel    EEG BloodPressure HeartRate Circulation
##   <lgl> <int> <dbl>      <dbl>  <dbl>         <dbl>     <dbl>       <dbl>
## 1 FALSE  1998 9869.     33057. -2887.            43       172        1519
## 2 TRUE   3148 9284.     25859. -2808.            43       158        1364
```

The Random Forest had significant difficulties identifying true falls, instead heavily favoring FALSE Falls. In the summary statistics, the median value is reported in order to resist the effects of outliers. The largest gap between the TRUE and FALSE Fall values is the `SugarLevel`. It just so happens that the deficit of TRUE Falls occurs where there is a dip in the density of TRUE Falls. In other words, the model is biased toward FALSE results because of the number advantage they have.

To correct this error, we can add simulated normal data to reduce the favor FALSE Falls currently enjoy.

```r
# Using medians as the means for rnorm() to resist outliers
means = train[row_index,] %>% filter(Fall == TRUE) %>% summarize(Time = median(Time), SugarLevel = medi

iqrs = train[row_index,] %>% filter(Fall == TRUE) %>% summarize(Time = IQR(Time), SugarLevel = IQR(Suga

# SDs are calculated from the IQR to ensure the SDs are not biased from outliers
sds = iqrs * 0.35

new_data = tibble(Time = vector(length = 100), SugarLevel = vector(length = 100), EEG = vector(length =

for(feature in colnames(means)){
  mean = means[[feature]] %>% as.numeric()
  sd = sds[[feature]] %>% as.numeric()
  data = rnorm(100, mean = mean, sd = sd)
  new_data[[feature]] = data
}

new_data$Fall = rep(TRUE, 100)
new_data %>% head() %>% kable()
```

Time	SugarLevel	EEG	BloodPressure	HeartRate	Circulation	Fall
12232.902	49370.97	-3146.551	49.34647	182.60881	2193.0169	TRUE
11795.492	29837.64	-3043.599	48.29762	145.88737	875.6936	TRUE
9284.308	29029.37	-3239.402	53.60912	236.74732	1863.7249	TRUE
11152.187	40486.64	-2575.712	34.64350	165.37108	1060.3237	TRUE
8525.513	33482.66	-3450.280	67.68997	82.25699	206.7929	TRUE
11231.996	18494.33	-4579.071	55.41196	219.68014	2041.7178	TRUE

As we can see this data more closely matches the data of the mistakes made. While this will likely cause disruptions to the model's overall performance, it will likely perform better in this specific region.

```r
added = rbind(train, new_data)
exp_fit = train(
  factor(Fall) ~ .,
  data = added,
  method = "rf",
  trControl = train_control
)
```

While this model performs roughly the same overall, it performs well on our previous mistakes.

```r
mistakes = train[row_index,]
mean(predict(exp_fit, mistakes) == mistakes$Fall)
```

```
## [1] 0.9885348
```

The general idea, now, is to put these two random forest models in ensemble, but we need a tie breaker. For this, we will train a k-Nearest Neighbors model. This gives a different perspective as the tie-breaking opinion. We will train this on the original training set.

```r
knn_fit = train(
  factor(Fall) ~ .,
  data = train,
  model = "knn",
  trControl = train_control
)
```

Results

We will run an ensemble of our three models on the test set with the following code.

```r
knn_predict = predict(knn_fit, test)
exp_predict = predict(exp_fit, test)
rf_predict = predict(rf_fit, test)

combined = tibble(knn = as.logical(knn_predict),
                  exp = as.logical(exp_predict),
                  rf = as.logical(rf_predict))
```

Here is the full confusion matrix:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  1033  127
##      TRUE     73  189
##
##                Accuracy : 0.8594
##                  95% CI : (0.8402, 0.877)
##     No Information Rate : 0.7778
##     P-Value [Acc > NIR] : 0.000000000000004435
##
##                   Kappa : 0.5667
##
##  Mcnemar's Test P-Value : 0.0001785
##
##             Sensitivity : 0.5981
##             Specificity : 0.9340
##          Pos Pred Value : 0.7214
##          Neg Pred Value : 0.8905
##              Prevalence : 0.2222
##          Detection Rate : 0.1329
##    Detection Prevalence : 0.1842
##       Balanced Accuracy : 0.7660
##
##        'Positive' Class : TRUE
##
```

The 2 of the 3 goals were met, with overall accuracy and specificity above 85%. However, the sensitivity fell far short of 85%, at around 60%.

While the intent was to be able to predict when a fall actually occurred, we were better able to determine when a patient did not fall. This is definitely a first step in detecting falls of elderly persons with wearable health devices, but the low sensitivity means work still needs to be done.

Among the things that would be most helpful going forward is better quality data. While it appears the model performed poorly, it actually reached the upper limit of what could be done with the given data and non-specialized machine learning algorithms. This is simply because of the amount of unreliable measurements and noisy data points. I tried to combat this by adding in normalized data to increase sensitivity, but that is a less-than-ideal option.

Additionally, having access to a test subject number would have been helpful for comparing baseline activites with falling, customized to that individual. That way, the machine learning algorithms would focus on *changes* in vitals that signify a fall, not merely assuming everybody's vitals are roughly the same.

Conclusion

While I would have liked to switched to a different project with better data, I found that this project forced me to remember things from this Professional Certificate Series and apply it that other projects would not have. It was challenging, but by sticking with it, I was able to think creatively and actually *apply* the knowledge I learned, not simply plugging in prepackaged algorithms. For example, I was stuck for a long time, forgetting about the importance of cleaning outliers out of the data. That is what led me to remembering Tukey's formula, which greatly helped me. I used the basic idea of PCA and dimensionality reduction to simplify the complex data set. Finally, I used my knowledge of the normal distribution to add additional data points while having a minimal effect on the integrity of the training set.

Overall, this was a phenomenal way to complete my journey through this Data Science Professional Certificate series. It gave me the opportunity to think outside the box and persist, qualities that are essential in a good data scientist. That is the greatest skill this project cemented in me as a data scientist: I can think for myself and think creatively, rather than relying on pre-fabricated tools.