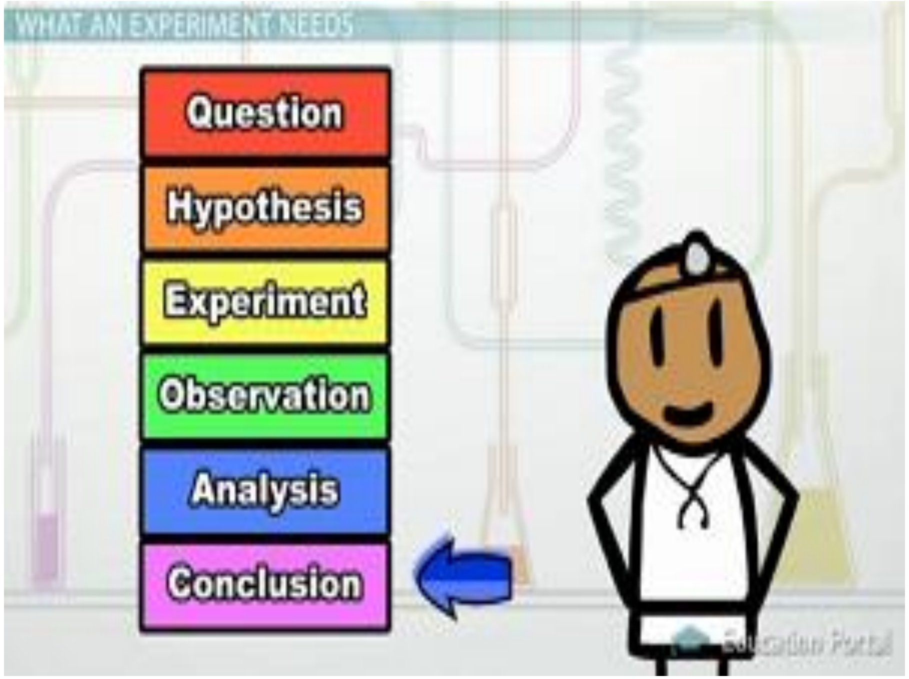
HANDS-ON: MAKING SENSE OF BIG DATA, MACHINE LEARNING, AND MODELING

Jameson Brennan and Hector Menendez

Dept. Animal Science, South Dakota State University

EXPERIMENTAL DESIGN

- When beginning an experiment
  - Study Design
  - Data Collection
  - Statistical Analysis
- What's missing?
  - Data processing



OVERVIEW

- Process raw data
- Train ML Models
- Assess Accuracy
- Predict Behavior
- Real-world Animal Science Example
  - Animal science data is getting bigger
  - More research on ML

WHY UTILIZE OPEN SOURCE PROGRAMS?

- Automate many processes
- Reproducible research
- Submit with publication
- Build on research better

HOUSEKEEPING

- Need to download
- Program R
  - https://www.r-project.org/
- R Studio
  - https://www.rstudio.com/products/rstudio/download/

HOUSEKEEPING

- **Helpers**
  - Anna Dagel
  - Logan Vandermark
  - Lily McFadden
  - Hector Menendez

HOUSEKEEPING

- Presentation Materials
  - Google Drive
  - **User Name:** nanp.2022@gmail.com
  - **Password:** ASAS_NANP_2022
  - **Download Folder:** Brennan_NANP_2022
  - Unzip to documents

HOUSEKEEPING

- Download folder
  - .RMD file
  - HTML Document
  - Example datasets

Name	Date modified	Type	Size
ASAS NANP 2022	5/16/2022 10:45 AM	RMD File	26 KB
ASAS-NANP-2022	5/16/2022 10:45 AM	Chrome HTML Do...	6,287 KB
DATA-021	5/16/2022 10:45 AM	Microsoft Excel C...	25,035 KB
DATA-022	5/16/2022 10:45 AM	Microsoft Excel C...	24,969 KB
DATA-023	5/16/2022 10:45 AM	Microsoft Excel C...	25,044 KB
DATA-024	5/16/2022 10:45 AM	Microsoft Excel C...	25,005 KB
DATA-025	5/16/2022 10:45 AM	Microsoft Excel C...	24,858 KB
Model_Training_Data	5/16/2022 10:45 AM	Microsoft Excel C...	2,908 KB

HOUSEKEEPING

- Set Working Directory
- Load Packages

Line 32

```
#Needed packages
  list.of.packages <- c("lubridate","ggplot2",'dplyr','randomForest','plotly','class','caret','MASS','knitr')
  new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
  if(length(new.packages)) install.packages(new.packages)
  library(lubridate)
  library(ggplot2)
  library(dplyr)
  library(randomForest)
  library(plotly)
  library(caret)
  library(class)
  library(MASS)
  library(rpart)
  library(e1071)
  library(knitr)
```

MACHINE LEARNING PROCESS

Step 1: Collecting data

Step 2: Preparing data

Step 3: Choosing a model

Step 4: Training the model

Step 5: Parameter tuning

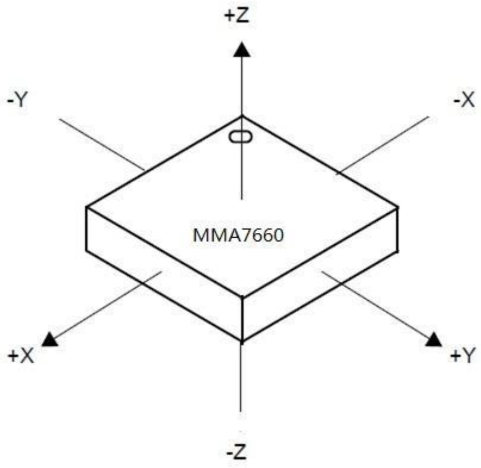
Step 6: Evaluate the model

Step 7: Make predictions

Goal use accelerometer data to predict livestock behavior (grazing, resting, walking)

STEP 1: COLLECTING DATA

- Accelerometers measure gravity
  - Axis Based Motion Sensing
  - X, Y, and Z Direction
- Used in Cell Phones
- Fitbits
- Animal movement and behavior
- ‘Livestock Accelerometer’ Web of Science
  - 165 articles



STEP 1: COLLECTING DATA

Gulf Coast Data Concepts Accelerometer

- X16-mini
- Records X, Y, and Z position
- Set at 12 Hz (~12 records per second)

5200mah Li-ion battery

Field Observations to train ML models

STEP 1: COLLECTING DATA

- **Big Data**
  - 32 Steers * 30 files/month * 3 Months =
  - 2880 Files *1,000,000 Records a file

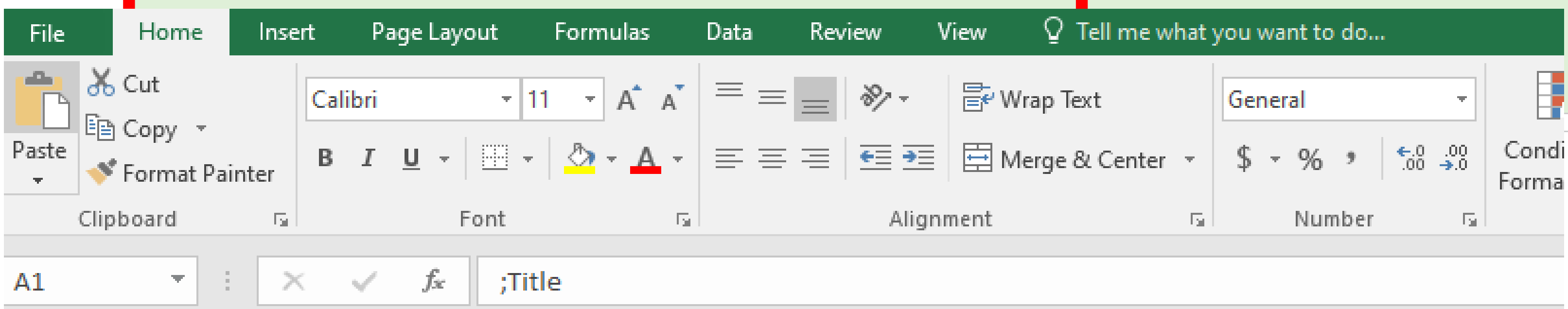
STEP 2: PREPARING DATA

- Write down common steps
  - Convert date time
  - Convert units
  - Drop unnecessary columns
  - Merge data files
  - Export to desired format

STEP 2: PREPARING DATA

Line 64

```
#Set working directory
#setwd("~/Conferences/NANP_2022/Workshopt")
```



	A	B	C	D	E	F
1	;Title	http://www.gcdataconcepts.com	X16-mini	Analog Dev ADXL345		
2	;Version	1113	Build date	Jan 6 2016	SN:CCDC1016DEE0180	
3	;Start_time	2017-06-10	14:22:08.011			
4	;Temperature	-999	deg C	Vbat	4062	mv
5	;SampleRate	12	Hz			
6	;Deadband	0	counts			
7	;DeadbandTimeout	0	sec			
8	;Time	Ax	Ay	Az		
9	0.045	789	-1404	1451		
10	0.128	1096	-1843	1685		
11	0.211	1060	-1116	1576		
12	0.294	719	-429	1701		
13	0.377	749	-720	1587		
14	0.46	1012	-1050	1028		

```
## 9          1451
## 10         1685
## 11         1576
## 12         1701
## 13         1587
## 14         1028
## 15         1190
```

STEP 2: PREPARING DATA

- Clean up header/calculate time

Line 79

```
#Extract start date and time and convert to date time object
start_time = paste(Accel_df[3,2],Accel_df[3,3])
start_time=as.POSIXct(start_time,format ="%Y-%m-%d %H:%M:%S")
```

```
#delete first 8 rows from dataframe and remove unneeded blank rows
Accel_df=Accel_df[-c(1:8),]
Accel_df$V5=NULL
Accel_df$V6=NULL
#rename columns and convert to numeric
colnames(Accel_df)=c("Time","Ax","Ay","Az")
Accel_df$Time=as.numeric(as.character(Accel_df$Time))
Accel_df$Ax=as.nu
Accel_df$Ay=as.nu
Accel_df$Az=as.nu
```

```
#add start time t
Accel_df$Time= st
rownames(Accel_df
head(Accel_df)
```

```
##                  Time   Ax    Ay   Az
## 1 2017-06-10 14:22:08  789 -1404 1451
## 2 2017-06-10 14:22:08 1096 -1843 1685
## 3 2017-06-10 14:22:08 1060 -1116 1576
## 4 2017-06-10 14:22:08  719  -429 1701
## 5 2017-06-10 14:22:08  749  -720 1587
## 6 2017-06-10 14:22:08 1012 -1050 1028
```

STEP 2: PREPARING DATA

- Convert Units

Line 117

```
#convert to g
Accel_df$Ax=Accel_df$Ax/2048
Accel_df$Ay=Accel_df$Ay/2048
Accel_df$Az=Accel_df$Az/2048
#Calculte MI and SMA
Accel_df$MI=sqrt(Accel_df$Ax^2 + Accel_df$Ay^2 + Accel_df$Az^2)
Accel_df$SMA=abs(Accel_df$Ax) + abs(Accel_df$Ay) + abs(Accel_df$Az)

head(Accel_df)
```

```
##                  Time        Ax         Ay        Az        MI      SMA
## 1 2017-06-10 14:22:08 0.3852539 -0.6855469 0.7084961 1.0584714 1.779297
## 2 2017-06-10 14:22:08 0.5351562 -0.8999023 0.8227539 1.3315932 2.257812
## 3 2017-06-10 14:22:08 0.5175781 -0.5449219 0.7695312 1.0756418 1.832031
## 4 2017-06-10 14:22:08 0.3510742 -0.2094727 0.8305664 0.9257281 1.391113
## 5 2017-06-10 14:22:08 0.3657227 -0.3515625 0.7749023 0.9261873 1.492188
## 6 2017-06-10 14:22:08 0.4941406 -0.5126953 0.5019531 0.8711994 1.508789
```

STEP 2: PREPARING DATA

Line 133

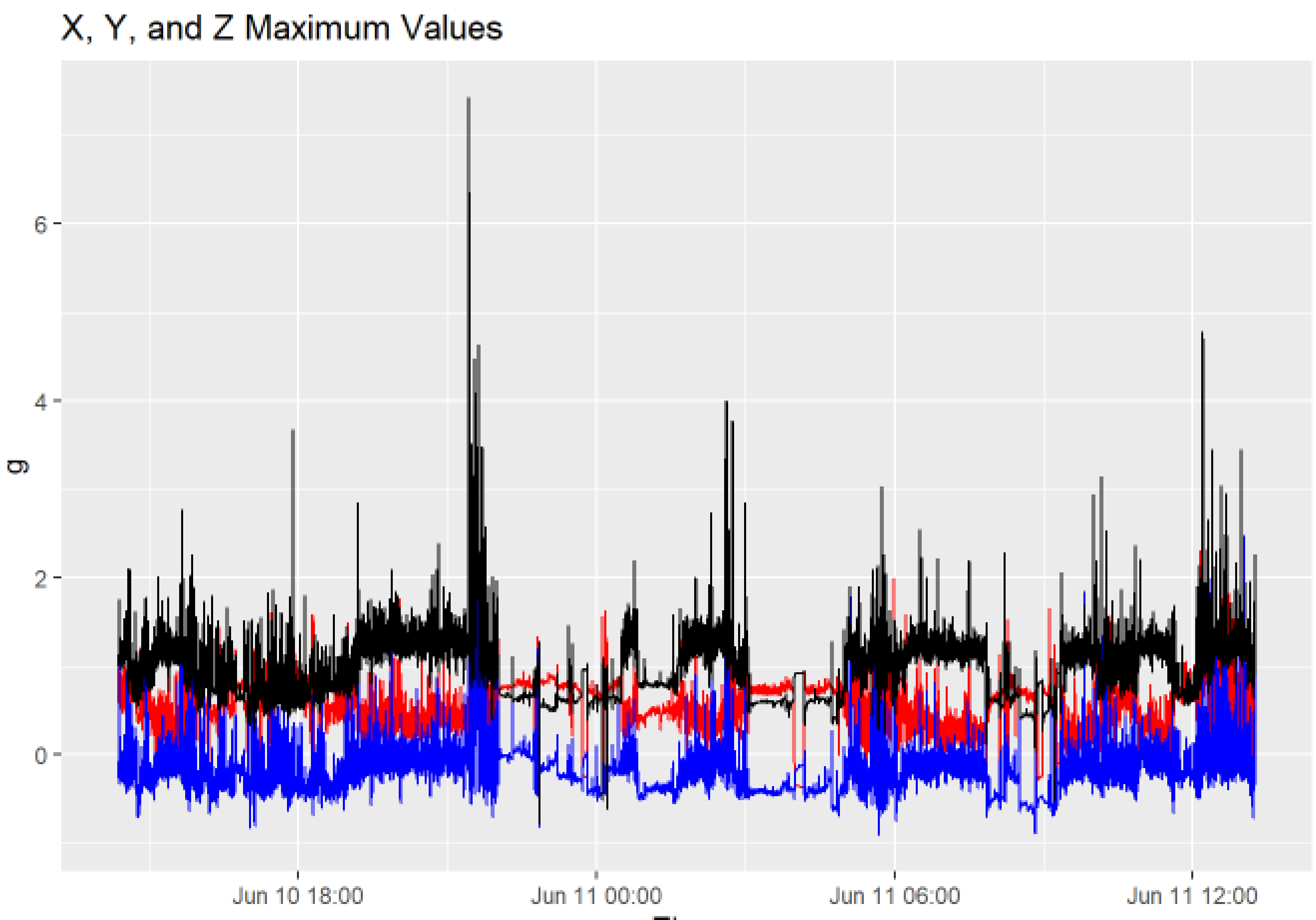
```
#round time to 5 s
Accel_df$Time=lubr
#Create Standard E
standard_error <- 
#Calculate Mean, M
Accel_mean=aggrega
colnames(Accel_mea
Accel_min=aggregat
colnames(Accel_min
Accel_max=aggregat
colnames(Accel_max
Accel_SE=aggregate
colnames(Accel_SE)

#Combine into one 

Accel_df=list(Acce
Accel_df=Reduce(fu
head(Accel_df)
```

```
##                  Time    X_Mean     Y_Mean    Z_Mean  MI_Mean SMA_Mean
## 1 2017-06-10 14:22:10 0.4301961 -0.5261841 0.7205200 1.009853 1.676900
## 2 2017-06-10 14:22:15 0.4179036 -0.5146240 0.7266846 1.002592 1.659212
## 3 2017-06-10 14:22:20 0.3803467 -0.5254639 0.7467855 1.004776 1.652596
## 4 2017-06-10 14:22:25 0.4224513 -0.5035781 0.7137071 0.986538 1.639736
## 5 2017-06-10 14:22:30 0.3697428 -0.5133952 0.7609701 1.002796 1.644108
## 6 2017-06-10 14:22:35 0.4598307 -0.4775228 0.7189290 1.002290 1.656283
##       X_Max      Y_Max     Z_Max   MI_Max  SMA_Max     X_Min      Y_Min
## 1 0.7519531 -0.2094727 1.0322266 1.335298 2.282715 0.2153320 -0.8999023
## 2 0.7592773 -0.1806641 1.0336914 1.364981 2.311523 0.1557617 -0.8886719
## 3 0.6918945 -0.1572266 1.1235352 1.414516 2.330078 0.2045898 -0.9643555
## 4 0.6606445 -0.1582031 0.9902344 1.305036 2.239258 0.2246094 -0.8476562
## 5 0.5810547 -0.2685547 1.1215820 1.381742 2.234375 0.2109375 -0.7763672
## 6 0.8901367 -0.1738281 1.0405273 1.373862 2.300293 0.2275391 -0.9233398
##       Z_Min    MI_Min  SMA_Min       X_SE       Y_SE       Z_SE      MI_SE
## 1 0.4555664 0.8391981 1.348633 0.02507613 0.03454523 0.03050527 0.03270258
## 2 0.4599609 0.8359342 1.314941 0.01742707 0.01874789 0.01774744 0.01822701
## 3 0.4584961 0.7883861 1.252441 0.01233284 0.01987365 0.01925812 0.01976474
## 4 0.4262695 0.7809516 1.207520 0.01507468 0.01756646 0.01633678 0.01650232
## 5 0.3808594 0.6734779 1.093262 0.01211075 0.01572578 0.02093040 0.01967968
## 6 0.3720703 0.8271793 1.252930 0.01891206 0.02135721 0.01849351 0.01838327
##       SMA_SE
## 1 0.05946504
## 2 0.03348271
## 3 0.03364648
## 4 0.03047485
## 5 0.03114984
## 6 0.03254150
```

STEP 2: PREPARING DATA

Line 160 Line 168

```
ggplot2::ggplot(Accel_df)+
  geom_line(aes(x=Time,y=X_Max),color='red')+
  geom_line(aes(x=Time,y=Y_Max),color='blue')+
  geom_line(aes(x=Time,y=Z_Max),color='black')+
  ylab("g")+
  ggtitle('X, Y, and Z Maximum Values')
```



STEP 2: PREPARING DATA

Line 183

```
Accel_function=function (datafile){
  #Load in the raw data file and view first 15 records
  Accel_df=read.csv(datafile,header=F)
  start_time = paste(Accel_df[3,2],Accel_df[3,3])
  start_time=as.POSIXct(start_time,format ="%Y-%m-%d %H:%M:%S")

  #delete first 8 rows from dataframe and remove unneeded blank rows
  Accel_df=Accel_df[-c(1:8),]
  Accel_df$V5=NULL
  Accel_df$V6=NULL
  #rename columns and convert to numeric
  colnames(Accel_df)=c("Time","Ax","Ay","Az")
  Accel_df$Time=as.numeric(as.character(Accel_df$Time))
  Accel_df$Ax=as.numeric(as.character(Accel_df$Ax))
  Accel_df$Ay=as.numeric(as.character(Accel_df$Ay))
  Accel_df$Az=as.numeric(as.character(Accel_df$Az))

  #add start time to time and display cleaned up header dataframe
  Accel_df$Time= start_time+Accel_df$Time
  rownames(Accel_df) <- NULL
```

Create a function with input 'datafile' name

Bunch of steps

• • •

```
 #Combine into one dataframe

 Accel_df=list(Accel_mean,Accel_max,Accel_min,Accel_SE)
 Accel_df=Reduce(function(x, y) merge(x, y, all=TRUE), Accel_df)

 return(Accel_df)

}
```

Return the processed data

STEP 2: PREPARING DATA

Line 244

```
Accel_data=Accel_function('DATA-022.csv')
head(Accel_data)
```

```
##                  Time   X_Mean     Y_Mean    Z_Mean   MI_Mean SMA_Mean
## 1 2017-06-11 13:15:20 0.3492635 -0.4703878 0.7804871 0.9853950 1.600138
## 2 2017-06-11 13:15:25 0.5361654 -0.3397054 0.7276042 0.9803144 1.603475
## 3 2017-06-11 13:15:30 0.6746971 -0.2567697 0.6084895 1.0119527 1.580492
## 4 2017-06-11 13:15:35 0.5690267 -0.4343913 0.6648600 1.0152054 1.668278
## 5 2017-06-11 13:15:40 0.5214844 -0.4165853 0.7055583 0.9847082 1.643628
## 6 2017-06-11 13:15:45 0.4854818 -0.4696533 0.7023600 0.9894400 1.657495
```

STEP 2: PREPARING DATA

Line 254

```
#extract the names of all files that match the string 'DATA-'
filenames=list.files(getwd(),pattern = "DATA-",all.files = FALSE)
#Process the list of files to create on dataframe with all five datafiles merged together
Accel_Merged <- dplyr::bind_rows(lapply(filenames[1:length(filenames)], Accel_function))
dim(Accel_Merged)
```

List all files with string matching 'DATA-'

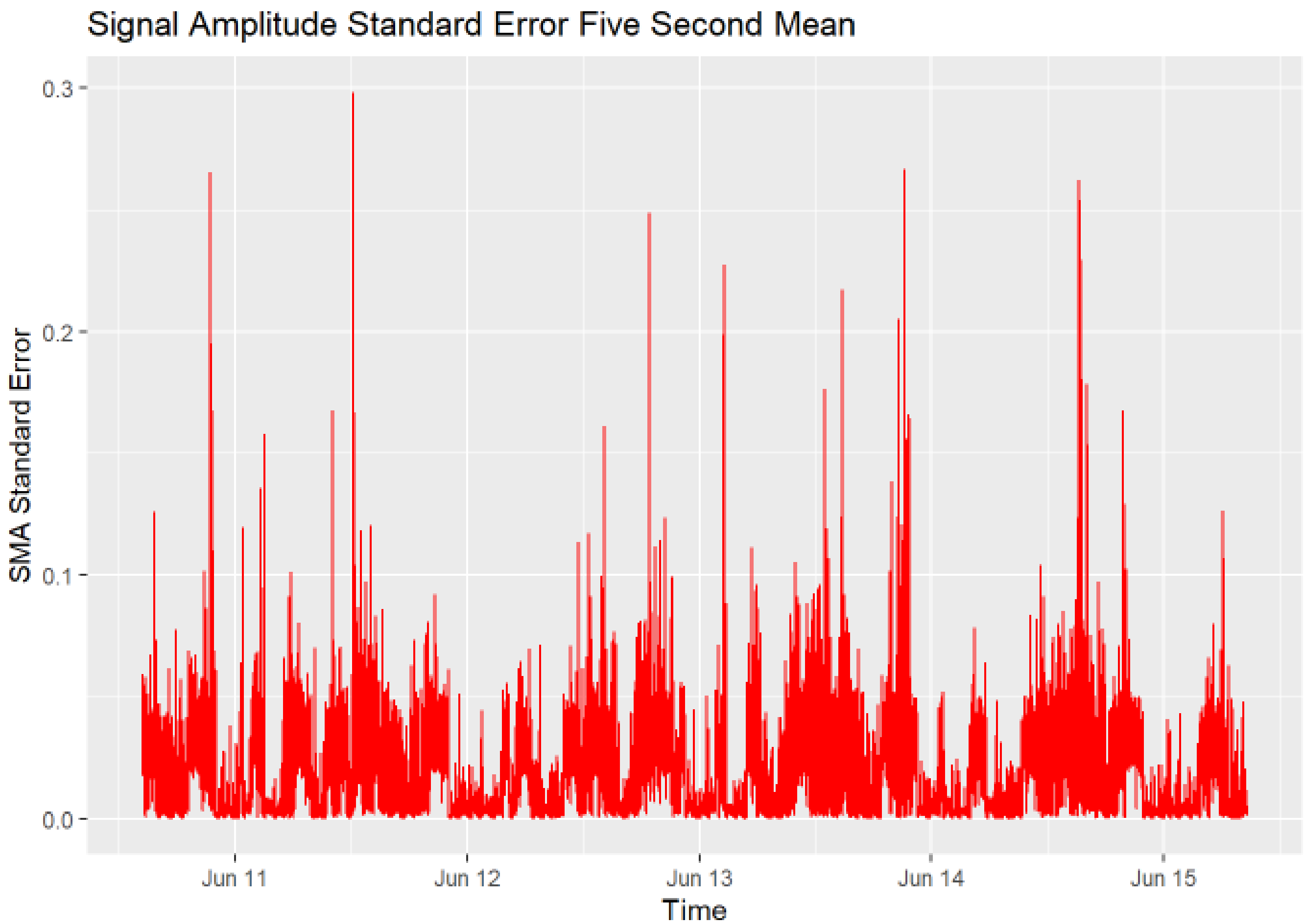
Apply our function to file list and bind rows together

```
## [1] 82281    21
```

STEP 2: PREPARING DATA

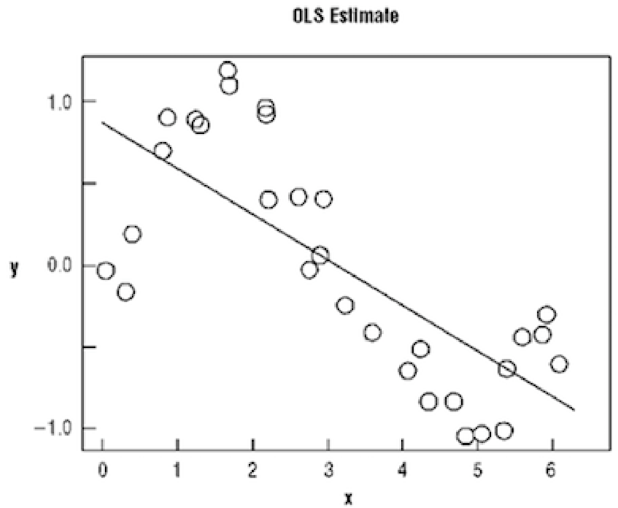
Line 263

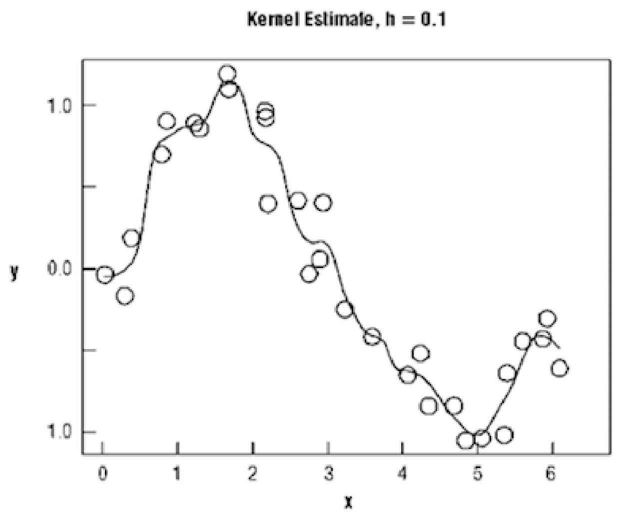
```
ggplot2::ggplot(Accel_Merged,aes(x=Time,y=SMA_SE))+
  geom_line(color='Red')+
  ylab("SMA Standard Error")+
  ggtitle('Signal Amplitude Standard Error Five Second Mean')
```



STEP 3: CHOOSING A MODEL

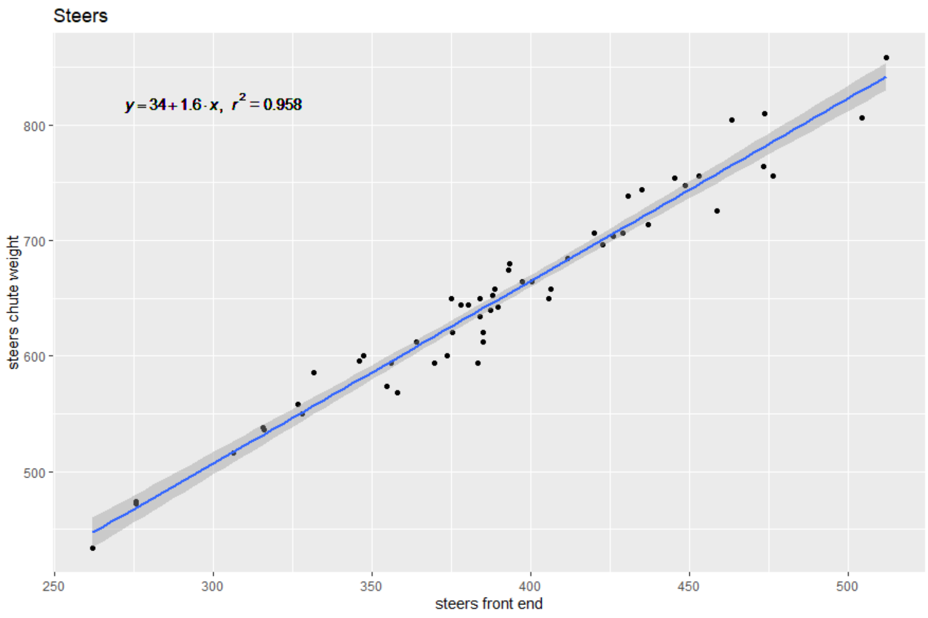
- Type of data
  - Parametric vs Non-Parametric
  - Continuous vs Categorical





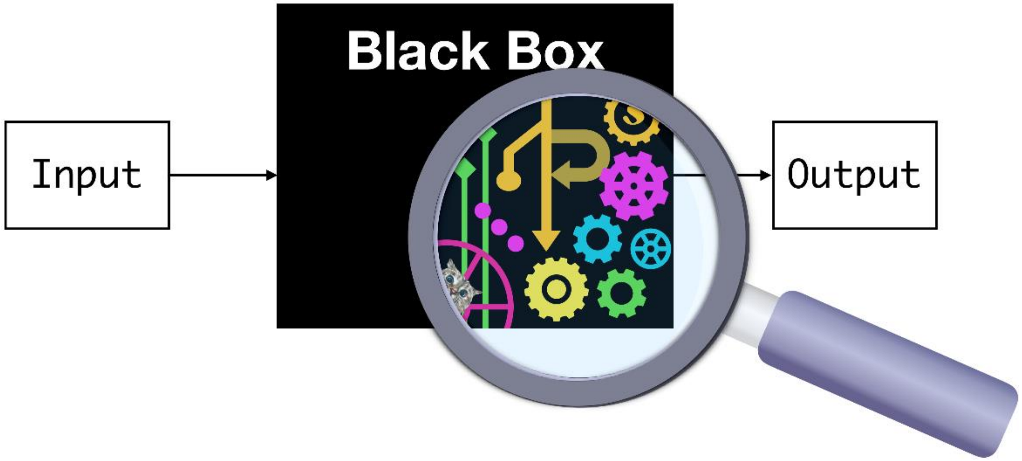
STEP 3: CHOOSING A MODEL

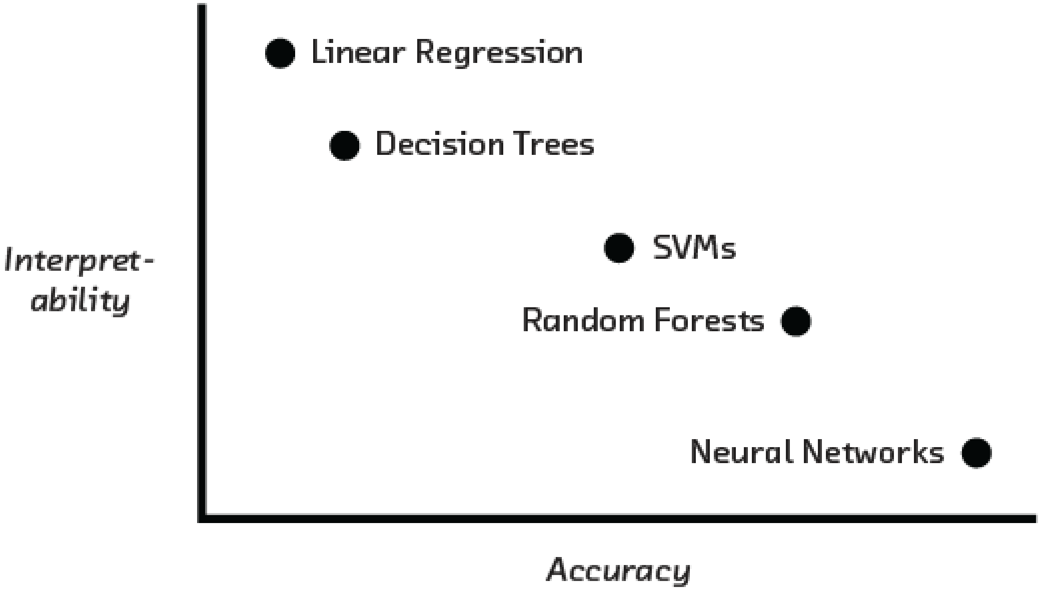
- Regression (Continuous Data)
  - Linear Model
  - Ridge Regression
  - Lasso Regression
  - Regression Trees
  - Splines
  - Neural Networks
- Classification (Categorical)
  - Logistic Regression (binary)
  - Discriminant analysis
  - KNN
  - Decision Trees
  - Support Vector Machines
  - Neural Networks



STEP 3: CHOOSING A MODEL

- Interpretability vs Accuracy





STEP 3: CHOOSING A MODEL

- Training dataset
  - “Model_Training_Data.csv”
  - Animal behavior observed in the field
  - Accelerometer data aggregated to 5 second intervals

Line 278

```
observed_df=read.csv('Model_Training_Data.csv')
#print column names
colnames(observed_df)
```

```
##  [1] "Time"      "X_Mean"    "Y_Mean"    "Z_Mean"    "MI_Mean"   "SMA_Mean"
##  [7] "X_Max"     "Y_Max"     "Z_Max"     "MI_Max"    "SMA_Max"   "X_Min"
## [13] "Y_Min"     "Z_Min"     "MI_Min"    "SMA_Min"   "MI_SE"     "SMA_SE"
## [19] "X_SE"      "Y_SE"      "Z_SE"      "Behavior"
```

STEP 3: CHOOSING A MODEL

- What kind of data?
- Is it balanced?

Line 278

```
observed_df=read.csv('Model_Training_Data.csv')
#Set Behavior as factor
observed_df$Behavior=as.factor(observed_df$Behavior)
#print column names
colnames(observed_df)
```

```
##  [1] "Time"     "X_Mean"   "Y_Mean"   "Z_Mean"   "MI_Mean"  "SMA_Mean"
##  [7] "X_Max"    "Y_Max"    "Z_Max"    "MI_Max"   "SMA_Max"  "X_Min"
## [13] "Y_Min"    "Z_Min"    "MI_Min"   "SMA_Min"  "MI_SE"    "SMA_SE"
## [19] "X_SE"     "Y_SE"     "Z_SE"     "Behavior"
```

```
#print count of each behavior
table(observed_df$Behavior)
```

```
##
##    G    R    W
## 6738 4520  220
```

STEP 3: CHOOSING A MODEL

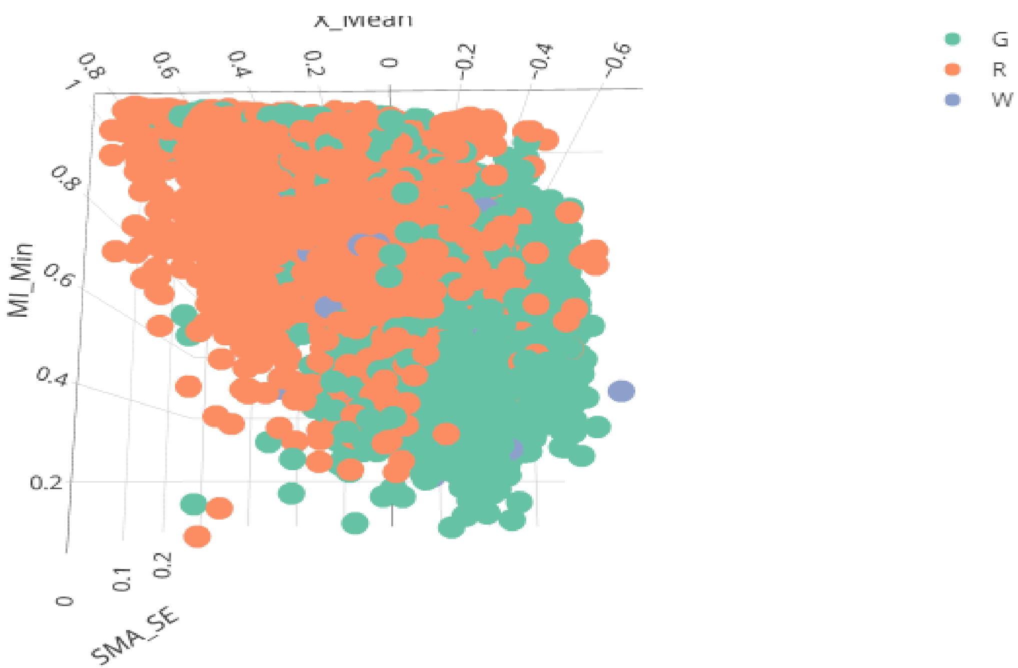
- Plot your data

Line 291 **Line 299** **Line 308** **Line 319**

```
plotly:: plot_ly(observed_df, x=~SMA_SE, y=~X_Mean,
         z=~MI_Min, color=~Behavior)
```

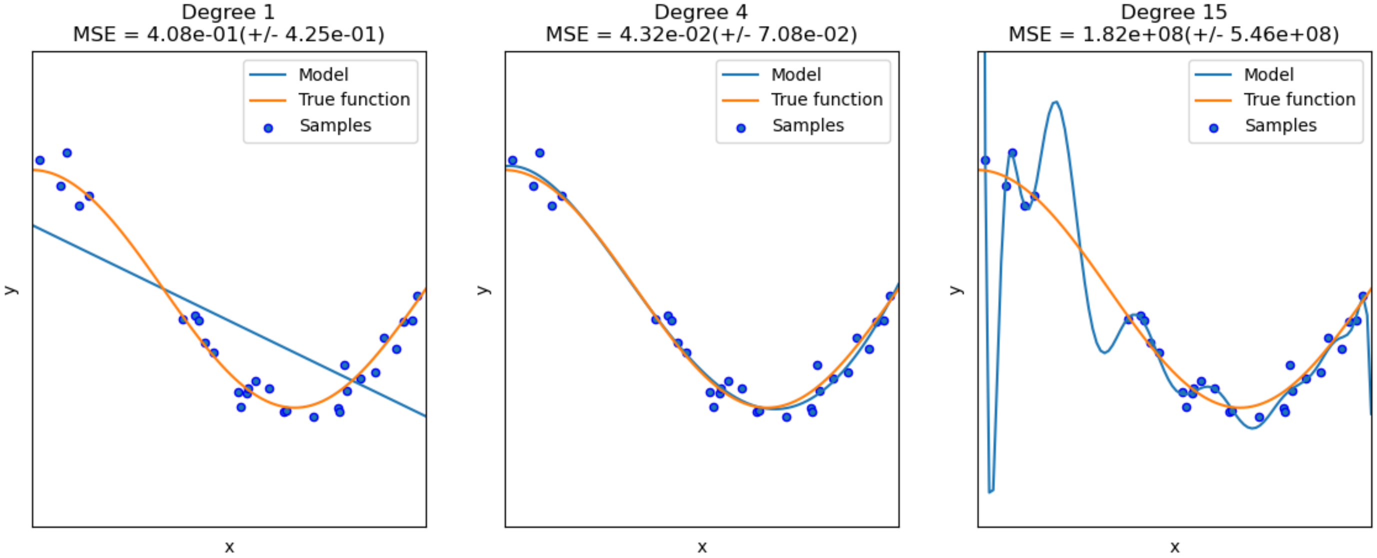
```
## No trace type specified:
##   Based on info supplied, a 'scatter3d' trace seems appropriate.
##   Read more about this trace type -> https://plotly.com/r/reference/#scatter3d
```

```
## No scatter3d mode specifed:
##   Setting the mode to markers
##   Read more about this attribute -> https://plotly.com/r/reference/#scatter-mode
```



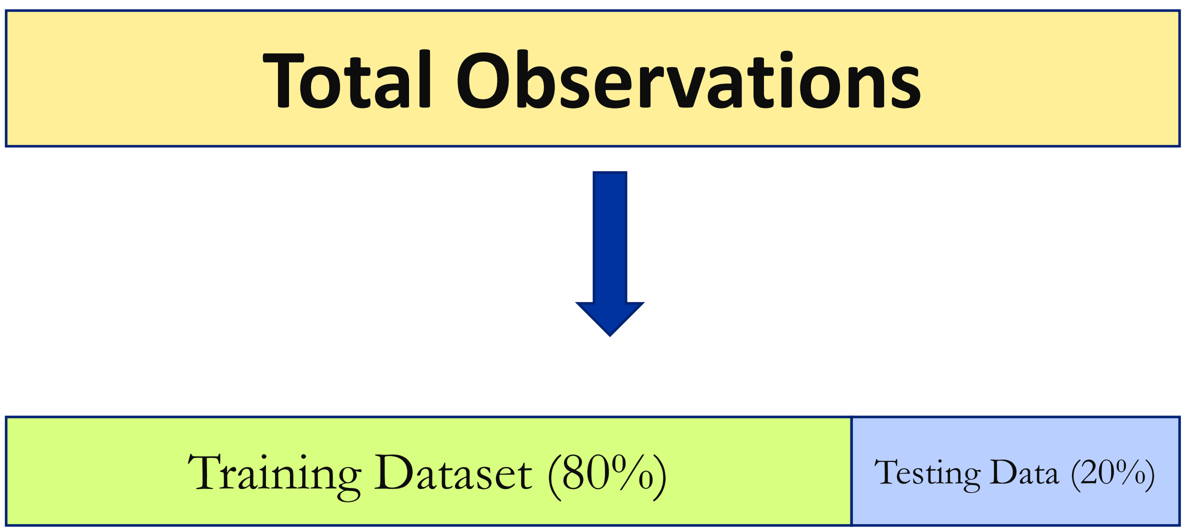
STEP 6: MODEL EVALUATIONS

- Model evaluations
  - Estimate performance accuracy of model
  - Need unbiased estimate
  - Goal predict on un-observed data
  - Evaluate **under-fitting** or **over-fitting** models



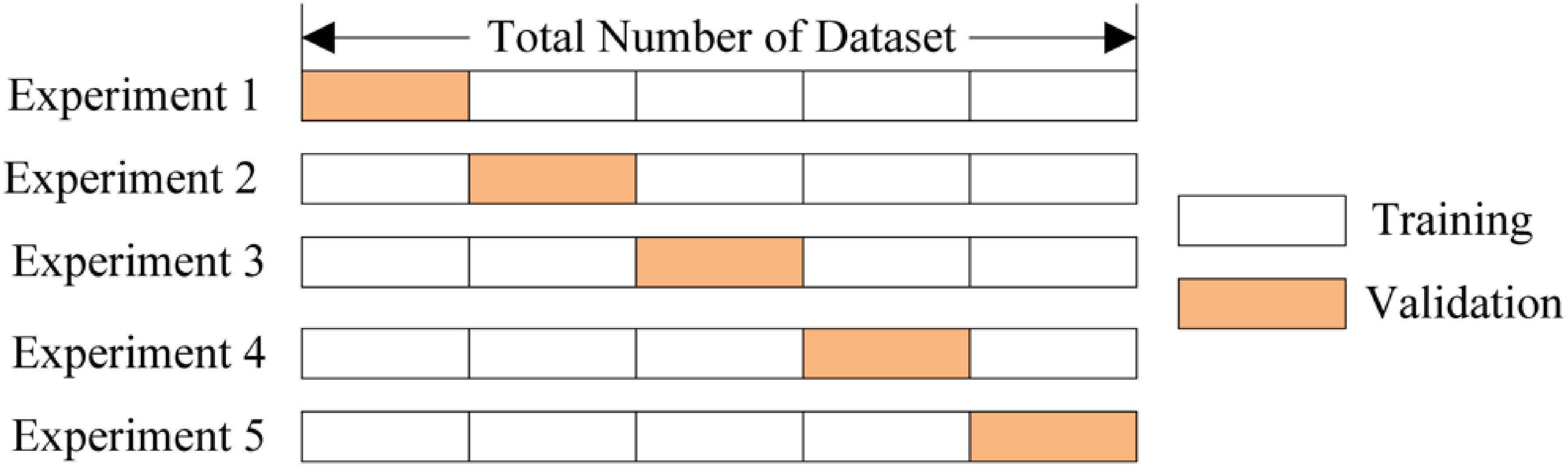
STEP 6: MODEL EVALUATIONS

- Validation Set Approach
  - Randomly split data into training/testing dataset
  - Easy to deploy
  - Dependent on subset of observations
  - Less data to train your model



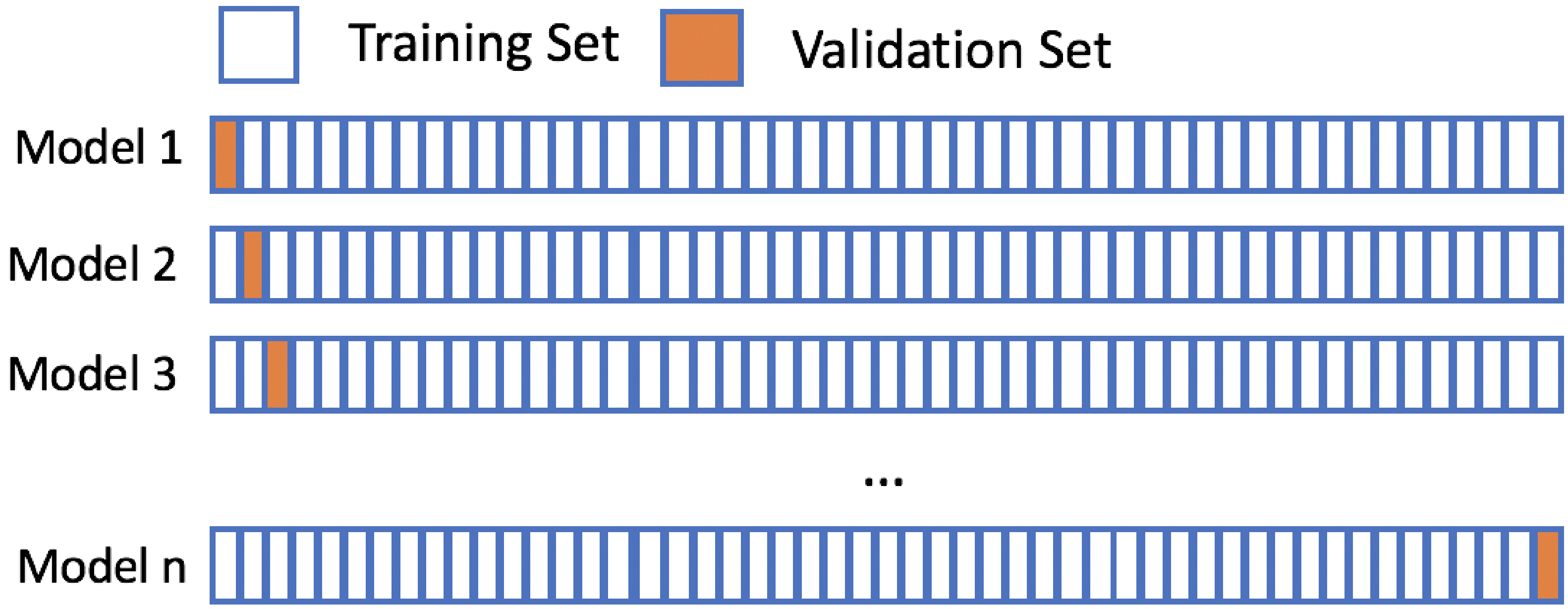
STEP 6: MODEL EVALUATIONS

- Cross Validation
  - Uses all the data
  - Range of accuracies
  - Reduces overfitting



STEP 6: MODEL EVALUATIONS

- Leave One Out Cross Validation (LOOCV)
  - Same as CV
  - K-Fold = Number of Observations
  - Modified examples
  - Computationally expensive



STEP 6: MODEL EVALUATIONS

- How to assess accuracy

$$\text{Overall } Accuracy = \frac{\#\ Correct}{Total\ Number}$$

Test	Has Disease	Does not have disease
Positive	True Positive 100	False Positive 112
Negative	False Negative 64	True Negative 1000

Sensitivity: TP/ TP+FN

Specificity: TN/ FP+TN

STEP 6: MODEL EVALUATION

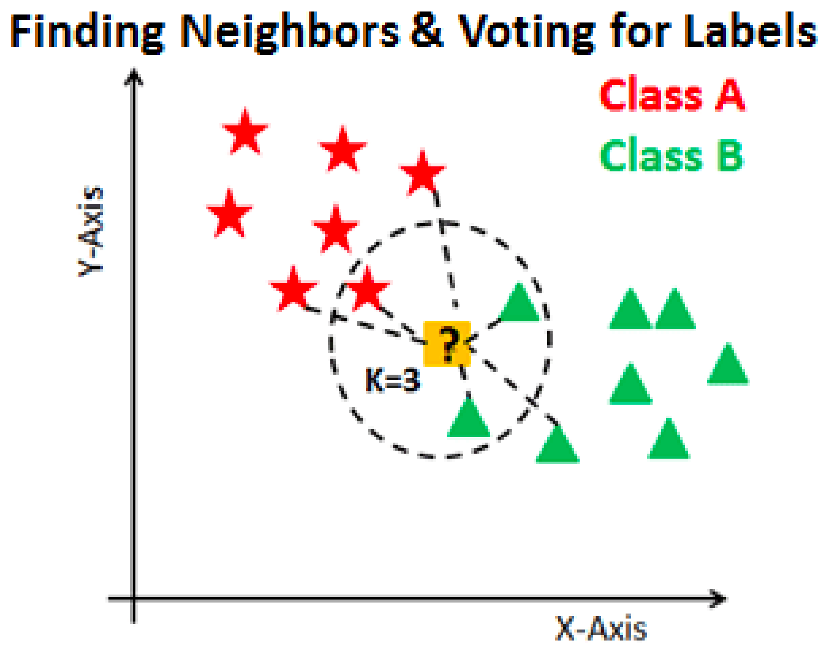
- Validation Set Approach (VSA)

Line 335

```r
#setting seed allows us to reproduce the exact results
set.seed(314)
observed_df$Behavior=as.factor(observed_df$Behavior)
#This example will do a 80/20 train/test. You can change the 0.8 to alter this ratio
train_data_index <- sample(1:nrow(observed_df), 0.8 * nrow(observed_df))
test_data_index <- setdiff(1:nrow(observed_df), train_data_index)

# Build train and test datset
train_data <- observed_df[train_data_index,]
test_data <- observed_df[test_data_index, ]
```

STEP 4: TRAINING THE MODEL

- **K Nearest Neighbors**
- non-parametric algorithm
- Classification based on distance to nearest neighbor



STEP 4: TRAINING THE MODEL

- KNN Validation Set Approach

Line 354

```
#KNN VSA method
#create train dataset with only predictors
train.knn=cbind.data.frame(train_data[,2:21])
#test dataset only predictors
test.knn=cbind.data.frame(test_data[,2:21])
#fit knn model with three nearest neighbors, assign prediction to test_data column KNN
test_data$KNN=knn(train.knn,test.knn,train_data$Behavior,k=3)
#compare prediction with observed on test dataset
caret::confusionMatrix(test_data$KNN,test_data$Behavior)
```

STEP 6: MODEL EVALUATION

- KNN VSA Output

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    G    R    W
##          G 1277   81   19
##          R   65  809    6
##          W   14    3   22
##
## Overall Statistics
##
##                Accuracy : 0.9181
##                  95% CI : (0.9061, 0.929)
##     No Information Rate : 0.5906
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.835
##
##  Mcnemar's Test P-Value : 0.3193
##
## Statistics by Class:
##
##                      Class: G Class: R Class: W
## Sensitivity            0.9417   0.9059 0.468085
## Specificity            0.8936   0.9494 0.992441
## Pos Pred Value         0.9274   0.9193 0.564103
## Neg Pred Value         0.9140   0.9407 0.988923
## Prevalence             0.5906   0.3889 0.020470
## Detection Rate         0.5562   0.3524 0.009582
## Detection Prevalence   0.5997   0.3833 0.016986
## Balanced Accuracy      0.9177   0.9277 0.730263
```

Overall Accuracy 91.8%

How about walking predictions

STEP 4: TRAINING THE MODEL

- KNN CV

Line 370

```
#KNN 10 fold Cross validation
#library(caret) #already l[illegible]et above? HMM##
set.seed(314)
#Train model [illegible] fold cv
knn.cv=train(Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max + SMA_Max
+ X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
              tuneGrid=expand.grid(k=3),
              method='knn',
              trControl=trainControl(method = "cv",number=5), #cha[illegible]ber of folds
              metric="Accuracy",
              data = observed_df)
knn.cv
```

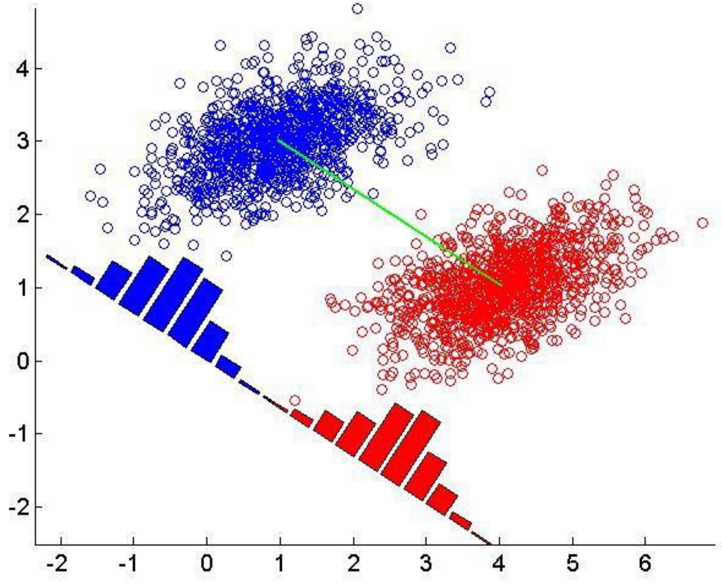
STEP 6: MODEL EVALUATION

- KNN CV Output

```
## k-Nearest Neighbors
##
## 11478 samples
##    20 predictor
##     3 classes: 'G', 'R', 'W'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 9182, 9182, 9183, 9183, 9182
## Resampling results:
##
##   Accuracy   Kappa
##   0.9214148  0.8409239
##
## Tuning parameter 'k' was held constant at a value of 3
```

STEP 4: TRAINING THE MODEL

- **Linear Discriminant Analysis (LDA)**
- LDA is a parametric algorithm used for classification.
- LDA takes the variance between the classes of the predictor variables and the variance within each class and compares them (in the form of a ratio).



STEP 4: TRAINING THE MODEL

- LDA VSA

Line 392

```
#####LDA VSA Approach

lda.vsa=lda(Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max + SMA_Max
+ X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
            data=train_data)
#predict behavior on test dataset using the model
test_data$LDA_VSA=predict(lda.vsa,test_data,type="response")$class
caret::confusionMatrix(test_data$LDA_VSA,test_data$Behavior)
```

STEP 4: TRAINING THE MODEL

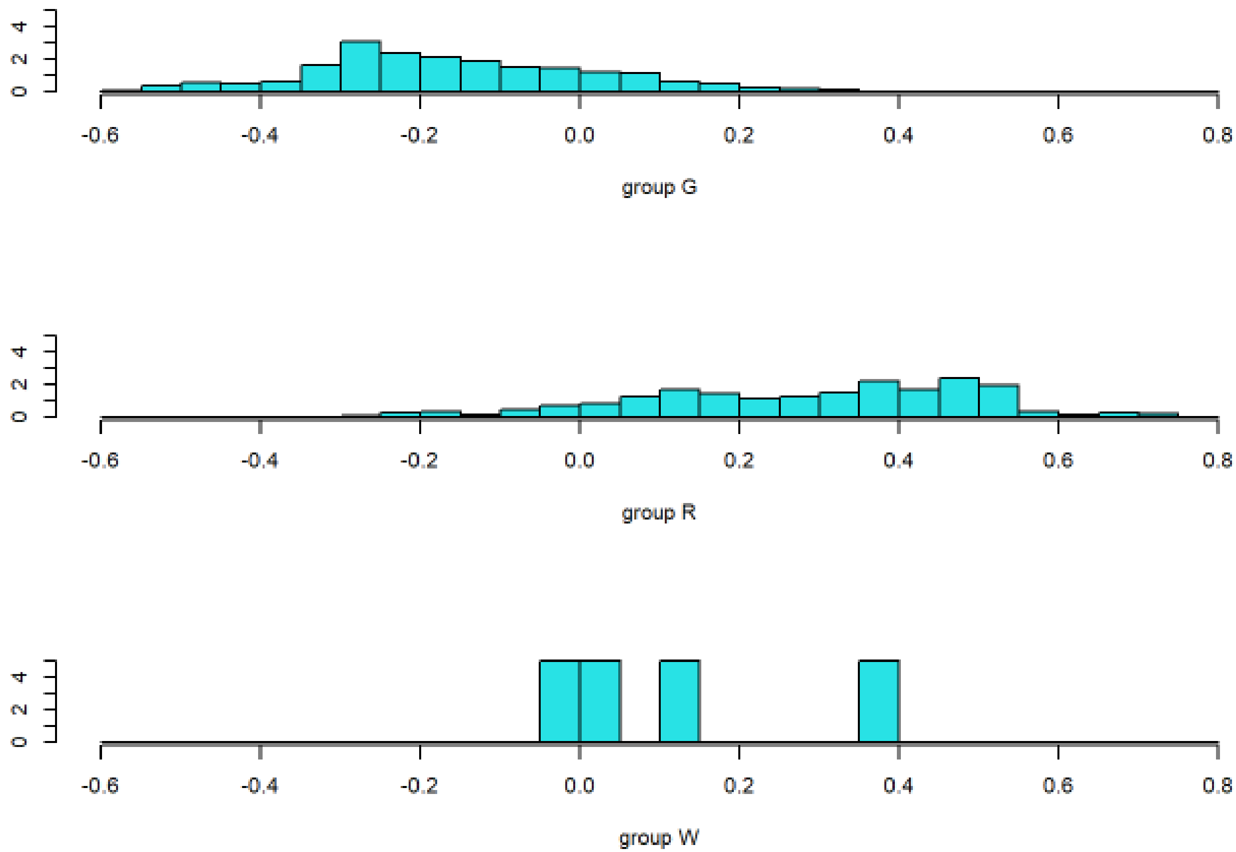
- LDA VSA Output

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    G    R    W
##          G 1289   85   36
##          R   64  808   10
##          W    3    0    1
##
## Overall Statistics
##
##                Accuracy : 0.9138
##                  95% CI : (0.9015, 0.9249)
##     No Information Rate : 0.5906
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8232
##
##  Mcnemar's Test P-Value : 6.924e-09
##
## Statistics by Class:
##
##                      Class: G Class: R  Class: W
## Sensitivity            0.9506   0.9048 0.0212766
## Specificity            0.8713   0.9473 0.9986661
## Pos Pred Value         0.9142   0.9161 0.2500000
## Neg Pred Value         0.9244   0.9399 0.9799302
## Prevalence             0.5906   0.3889 0.0204704
## Detection Rate         0.5614   0.3519 0.0004355
## Detection Prevalence   0.6141   0.3841 0.0017422
## Balanced Accuracy      0.9109   0.9260 0.5099713
```

STEP 4: TRAINING THE MODEL

Line 405

```
ldahist(test_data$X_Mean,test_data$LDA_VSA)
```



STEP 4: TRAINING THE MODEL

- LDA CV

Line 412

```
cv.lda <-function (data, model=origin~., yname="origin", K=5, seed=314) {
    n <- nrow(data)
    set.seed(seed)
    datay=data[,yname] #response variable
    #partition the data into K subsets
    f <- ceiling(n/K)
    s <- sample(rep(1:K, f), n)
    #generate indices 1:10 and sample n of them
    # K fold cross-validated error
    CV=NULL
    for (i in 1:K) { #i=1
      test.index <- seq_len(n)[(s == i)] #test data
      train.index <- seq_len(n)[(s != i)] #training data

      #model with training data
      lda.fit=lda(model, data=data[train.index,])
      #observed test set y
      lda.y <- data[test.index, yname]
      #predicted test set y
      lda.predy=predict(lda.fit, data[test.index,])$class

      #observed - predicted on test data
      error= mean(lda.y!=lda.predy)
      #error rates
      CV=c(CV,error)
    }
    #Output
    list(call = model, K = K,error=CV,
         lda_error_rate = mean(CV), seed = seed)
  }
```

```
#Use our function to run the CV
lda.kfold=cv.lda(data=observed_df,
                 model = Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max +  MI_Max + SMA_Max
+ X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
                 yname="Behavior",
                 K=5,
                 seed = 314)
#Show output and store accuracy
lda.kfold
```

STEP 6: MODEL EVALUATION

- LDA CV Output

```
## $call
## Behavior ~ X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max +
##     Y_Max + Z_Max + MI_Max + SMA_Max + X_Min + Y_Min + Z_Min +
##     MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE
##
## $K
## [1] 10
##
## $error
##  [1] 0.09581882 0.09930314 0.09059233 0.07578397 0.09067132 0.08362369
##  [7] 0.07578397 0.09407666 0.07142857 0.08456844
##
## $lda_error_rate
## [1] 0.08616509
##
## $seed
## [1] 314
```

STEP 4: TRAINING THE MODEL

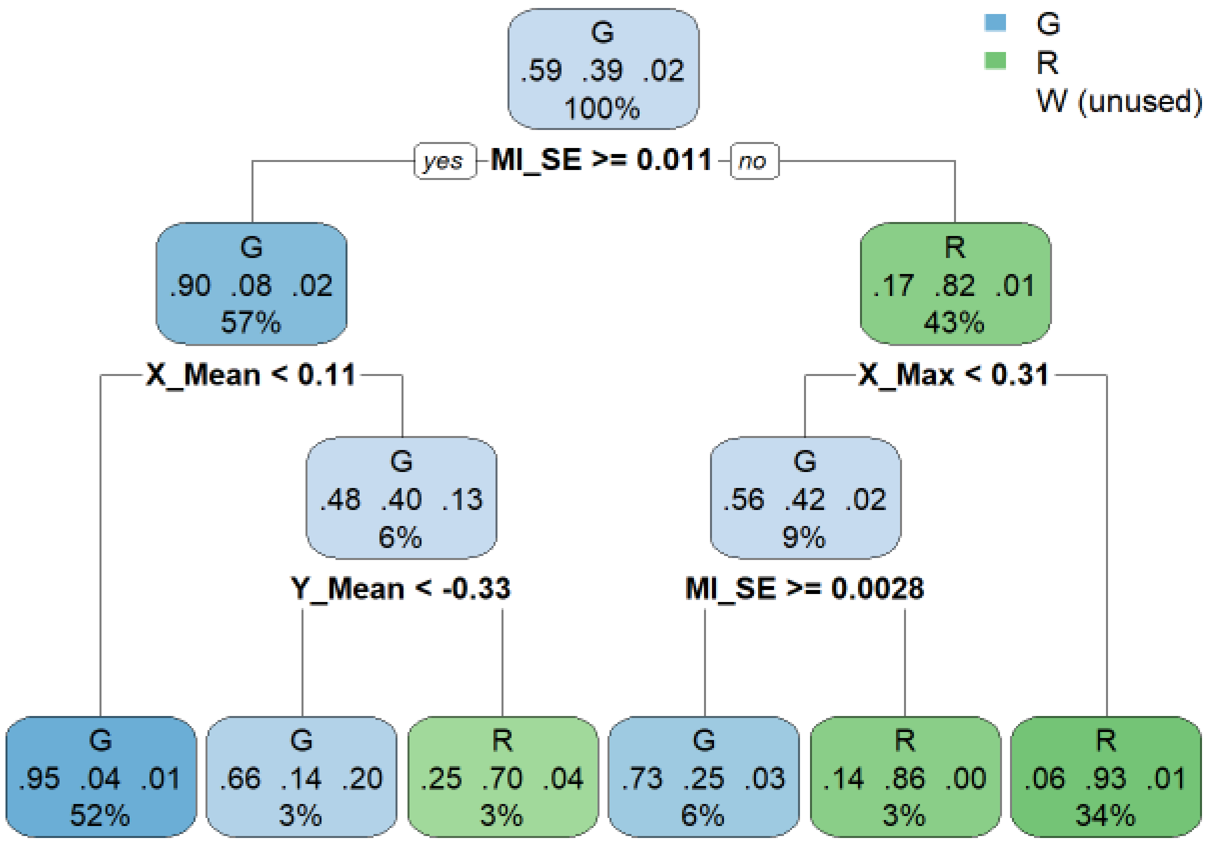
- **Random Forest**
  - Constructs multiple decision trees
  - Uses bootstrapping to select a random sample of data
  - Feature bagging to create a random subset of features (reduces overfitting).

STEP 4: TRAINING THE MODEL

- Decision Tree simple example

Line 462

```
#Example of recursi
rpart_vsa=rpart(Beh
+ Y_Min + Z_Min + MI
              ,da

#Plot the decision
rpart.plot:: rpart.
```




```
Min
```

STEP 4: TRAINING THE MODEL

- RF VSA

Line 473

```
#Set Seed
set.seed(314)
#Fit Random Forest Model
rf_vsa=randomForest( Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max +
SMA_Max + X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE
                    ,ntree=1000,data=train_data)
#Predict Behavior on test dataset
test_data$rf_vsa=predict(rf_vsa,newdata=test_data)
#Generate confusion matrix and save accuracy
caret::confusionMatrix(test_data$rf_vsa,test_data$Behavior)
```

Fit Model with 1000 trees

Use model to predict test data

Evaluate model

STEP 6: MODEL EVALUATION

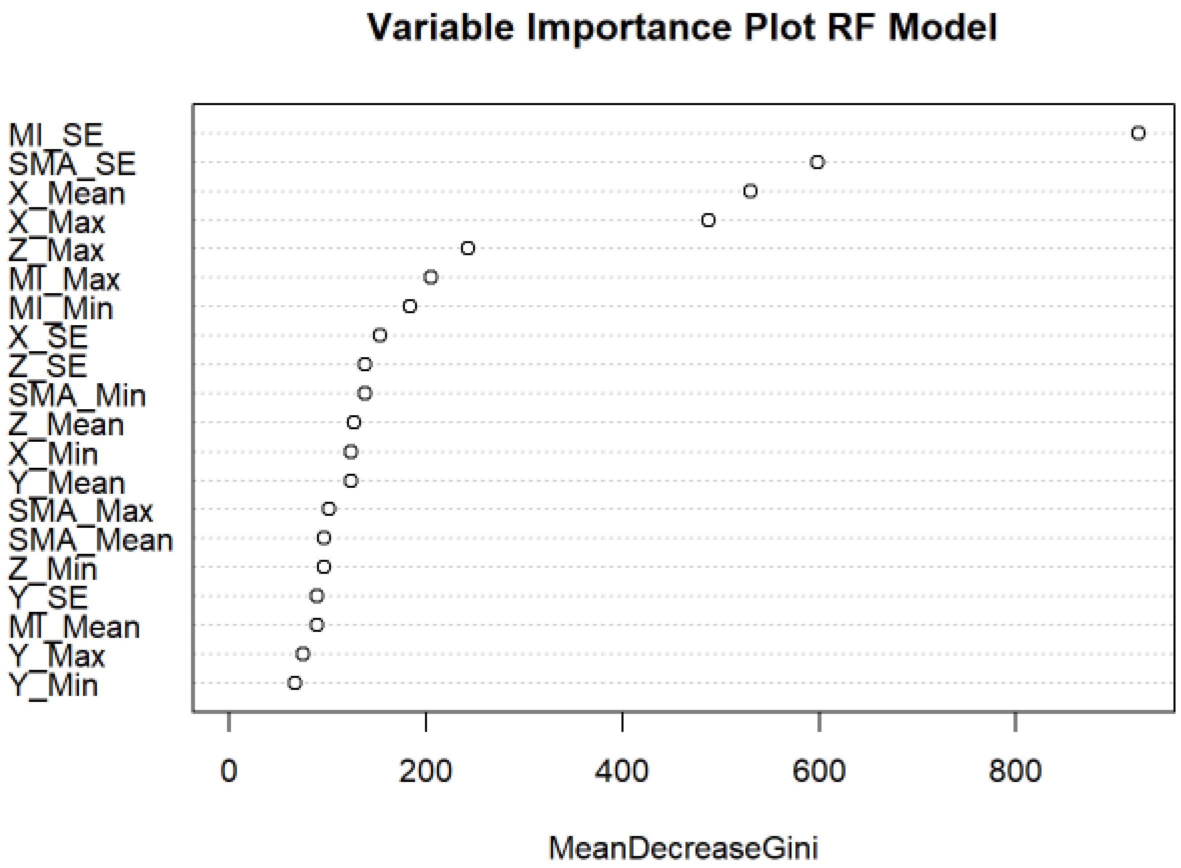
- RF VSA Output

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    G    R    W
##          G 1301   41   24
##          R   54  852    9
##          W    1    0   14
##
## Overall Statistics
##
##                Accuracy : 0.9438
##                  95% CI : (0.9336, 0.9529)
##     No Information Rate : 0.5906
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8861
##
##  Mcnemar's Test P-Value : 5.391e-07
##
## Statistics by Class:
##
##                      Class: G Class: R Class: W
## Sensitivity            0.9594   0.9541 0.297872
## Specificity            0.9309   0.9551 0.999555
## Pos Pred Value         0.9524   0.9311 0.933333
## Neg Pred Value         0.9409   0.9703 0.985533
## Prevalence             0.5906   0.3889 0.020470
## Detection Rate         0.5666   0.3711 0.006098
## Detection Prevalence   0.5949   0.3985 0.006533
## Balanced Accuracy      0.9451   0.9546 0.648714
```

STEP 6: EVALUATE THE MODEL

- RF Variable Importance

Line 488



STEP 4: TRAINING THE MODEL

- RF CV

Line 495

```
rf.cv=function (data, model=origin~., yname="origin", K=10, seed=314) {
  n <- nrow(data)
  set.seed(seed)
  datay=data[,yname] #response variable
  #partition the data into K subsets
  f <- ceiling(n/K)
  s <- sample(rep(1:K, f), n)
  #generate indices 1:10 and sample n of them
  # K fold cross-validated error

  CV=NULL
  #i=3
  for (i in 1:K) { #i=1
    test.index <- seq_len(n)[(s == i)] #test data
    train.index <- seq_len(n)[(s != i)] #training data

    #model with training data
    rf.fit=randomForest(model, data=data[train.index,])
    #observed test set y
    rf.y <- data[test.index, yname]
    #predicted test set y
    rf.predy=predict(rf.fit, data[test.index,])

    #observed - predicted on test data
    error= mean(rf.y!=rf.predy)
    #error rates
    CV=c(CV,error)
  }
  #Output
  list(call = model, K = K,error=CV,
       rf_error_rate = mean(CV), seed = seed)
}
#Run function for cross validation using random forest
cv_rf=rf.cv(data = observed_df,
            model =  Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max +
SMA_Max + X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
            yname="Behavior",
            K=10,
            seed = 314)
#Cross validation output
cv_rf
```

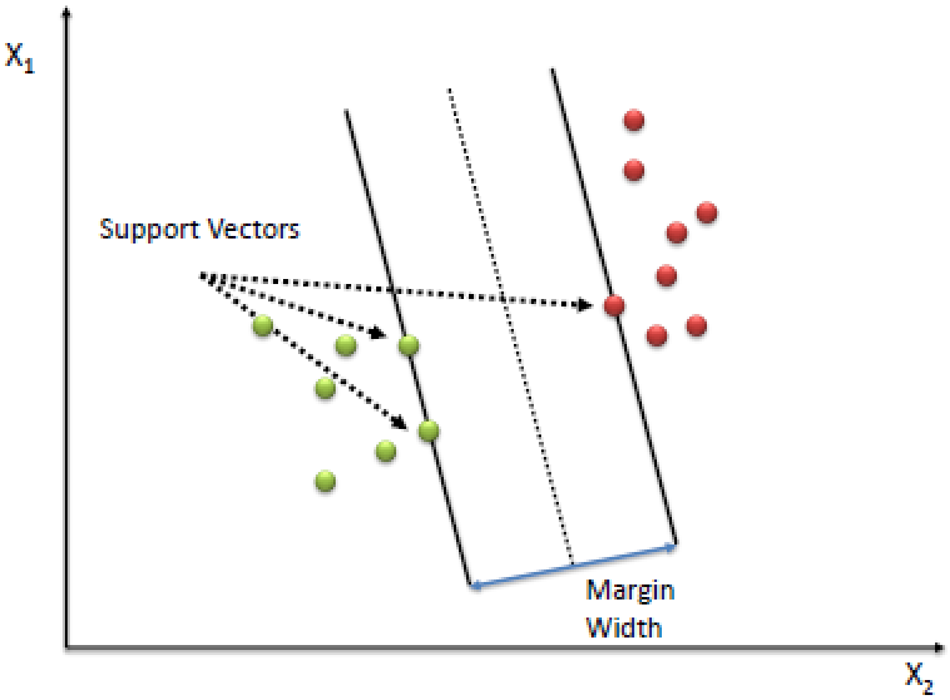
STEP 6: EVALUATE THE MODEL

- RF CV Output

```
## $call
## Behavior ~ X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max +
##     Y_Max + Z_Max + MI_Max + SMA_Max + X_Min + Y_Min + Z_Min +
##     MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE
##
## $K
## [1] 10
##
## $error
##  [1] 0.05836237 0.06445993 0.04965157 0.04442509 0.04795118 0.05574913
##  [7] 0.04703833 0.06097561 0.04094077 0.04620750
##
## $rf_error_rate
## [1] 0.05157615
##
## $seed
## [1] 314
```

STEP 4: TRAINING THE MODEL

- Support Vector Machine (SVM)
- SVM discriminative classifier that constructs separating hyperplanes



STEP 6: EVALUATE THE MODEL

- SVM VSA

Line 544

```
#Fit SVM model
svm_mod=svm(Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max + SMA_Max
+ X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
            data = train_data,kernel='linear')
#Use model to predict on test dataset
test_data$SVM_VSA=predict(svm_mod,test_data)
#Calculate confusion matrix and save accuracy
svm_vsa=as.numeric(caret::confusionMatrix(test_data$Behavior,test_data$SVM_VSA)$overall[1])
caret::confusionMatrix(test_data$Behavior,test_data$SVM_VSA)
```

STEP 6: EVALUATE THE MODEL

- SVM Output

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    G    R    W
##          G 1285   71    0
##          R   71  822    0
##          W   39    8    0
##
## Overall Statistics
##
##                Accuracy : 0.9177
##                  95% CI : (0.9057, 0.9286)
##     No Information Rate : 0.6076
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8315
##
##  Mcnemar's Test P-Value : 3.476e-10
##
## Statistics by Class:
##
##                      Class: G Class: R Class: W
## Sensitivity            0.9211   0.9123       NA
## Specificity            0.9212   0.9491  0.97953
## Pos Pred Value         0.9476   0.9205       NA
## Neg Pred Value         0.8830   0.9437       NA
## Prevalence             0.6076   0.3924  0.00000
## Detection Rate         0.5597   0.3580  0.00000
## Detection Prevalence   0.5906   0.3889  0.02047
## Balanced Accuracy      0.9212   0.9307       NA
```

STEP 6: EVALUATE THE MODEL

- SVM CV

Line 557

```
svm.cv=function (data, model=origin~., yname="origin", K=10, seed=314) {
  n <- nrow(data)
  set.seed(seed)
  datay=data[,yname] #response variable
  #partition the data into K subsets
  f <- ceiling(n/K)
  s <- sample(rep(1:K, f), n)
  #generate indices 1:10 and sample n of them
  # K fold cross-validated error

  CV=NULL
  #i=3
  for (i in 1:K) { #i=1
    test.index <- seq_len(n)[(s == i)] #test data
    train.index <- seq_len(n)[(s != i)] #training data

    #model with training data
    svm.fit=svm(model, data=data[train.index,],kernel='linear')
    #observed test set y
    svm.y <- data[test.index, yname]
    #predicted test set y
    svm.predy=predict(svm.fit, data[test.index,])

    #observed - predicted on test data
    error= mean(svm.y!=svm.predy)
    #error rates
    CV=c(CV,error)
  }
  #Output
  list(call = model, K = K,error=CV,
      svm_error_rate = mean(CV), seed = seed)
}
#Run function for cross validation using random forest
cv_svm=svm.cv(data = observed_df,
            model =  Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max +
SMA_Max + X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
            yname="Behavior",
            K=10,
            seed = 314)
#store accuracy
cv_svm=1-cv_svm$svm_error_rate
#Cross validation output
cv_svm
```

STEP 6: EVALUATE THE MODEL

Line 606

Display Table of Results

Now that we have run our selected machine learning models, we can display the accuracy for each model and testing sceme into a table.

```
#create dataframe of accuracy and models
final_table=as.data.frame(rbind(c(knn.vsa*100,knn.cv*100),c(LDA_VSA*100,lda.cv*100),c(vsa_rf*100,cv_rf*100),c(svm_vsa*100,cv
_svm*100)))
final_table$Model=c("KNN","LDA","RF","SVM")
colnames(final_table)=c("VSA Accuracy","10-Fold CV Accuracy","Model")
final_table<- final_table[, c(3,1,2)]

rownames(final_table)=NULL


knitr::kable(final_table,digits=1,caption="Model Accuracy (%) for Validation and CV Approaches")
```

Model Accuracy (%) for Validation and CV Approaches

Model	VSA Accuracy	10-Fold CV Accuracy
KNN	91.8	92.2
LDA	91.4	91.4
RF	94.4	94.8
SVM	91.8	91.7

STEP 5: PARAMETER TUNING

- Tune models to get the best fit
- Improve accuracy of models
- Different for each ML model

STEP 5: PARAMETER TUNING

- Example: KNN parameter tuning

Line 626

```
#KNN 10 fold Cross validation
library(caret)
set.seed(314)
#Train model using 10 fold cv
knn.cv=train(Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max + SMA_Max
+ X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
              tuneGrid=expand.grid(k=c(1,3,5,7)),
              method='knn',
              trControl=trainControl(method = "cv",number=10), #change number to change number of folds
              metric="Accuracy",
              data = observed_df)
knn.cv
```

STEP 5: PARAMETER TUNING

- KNN output

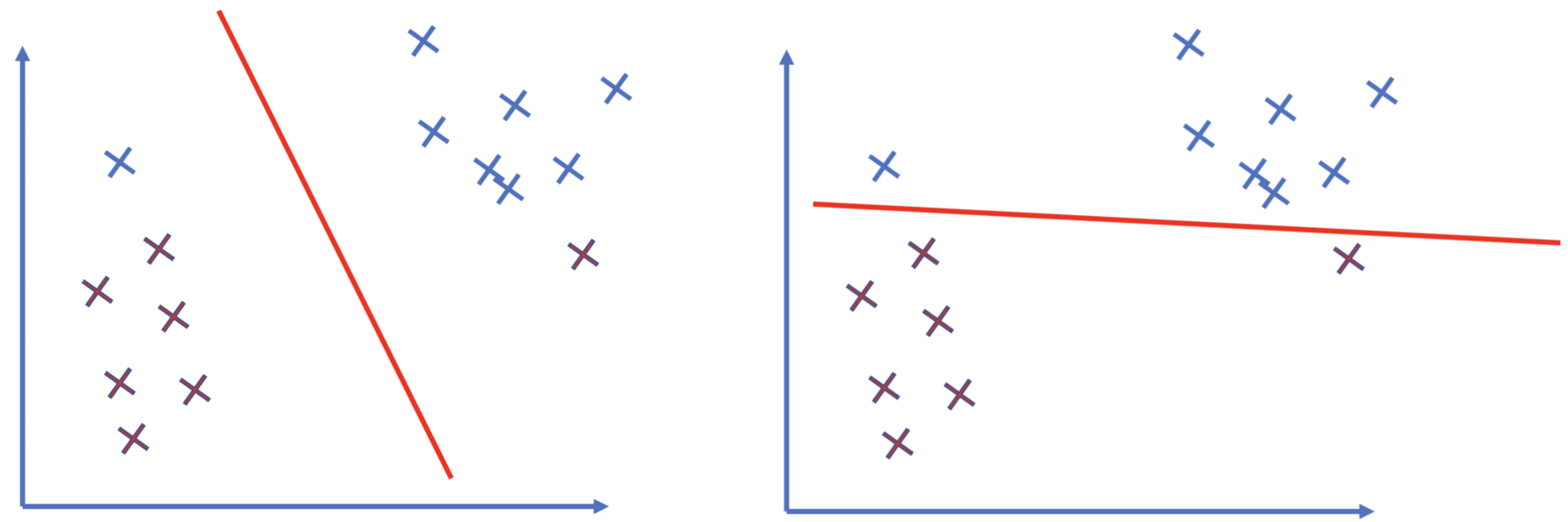
```
## k-Nearest Neighbors
##
## 11478 samples
##    20 predictor
##     3 classes: 'G', 'R', 'W'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 10330, 10331, 10330, 10330, 10330, 10330, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   1  0.9088694  0.8174106
##   3  0.9223727  0.8428673
##   5  0.9242023  0.8459400
##   7  0.9254220  0.8481722
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

STEP 5: PARAMETER TUNING

- SVM Tune Cost Function

Line 648

```
set.seed(314)

tune_svm=tune(svm,Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max + SMA_Max
+ X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
              data = observed_df,kernel='linear',ranges = list(cost=c(.01,10)))
summary(tune_svm)
```



Line 659

STEP 5: PARAMETER TUNING

```
#Fit SVM model
svm_lin=svm(Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max + SMA_Max
+ X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
            data = train_data,kernel='linear')
#Use model to predict on test dataset
test_data$SVM_lin=predict(svm_lin,test_data)
#Calculate confusion matrix and save accuracy
svm_lin=as.numeric(caret::confusionMatrix(test_data$Behavior,test_data$SVM_lin)$overall[1])

svm_rad=svm(Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max + SMA_Max
+ X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
            data = train_data,kernel='radial')
#Use model to predict on test dataset
test_data$SVM_rad=predict(svm_rad,test_data)
#Calculate confusion matrix and save accuracy
svm_rad=as.numeric(caret::confusionMatrix(test_data$Behavior,test_data$SVM_rad)$overall[1])

svm_poly=svm(Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max + SMA_Max
+ X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE,
            data = train_data,kernel='polynomial')
#Use model to predict on test dataset
test_data$SVM_poly=predict(svm_poly,test_data)
#Calculate confusion matrix and save accuracy
svm_poly=as.numeric(caret::confusionMatrix(test_data$Behavior,test_data$SVM_poly)$overall[1])

print(paste('Linear:',svm_lin,',', "Radial:",svm_rad,',', "Polynomial:",svm_poly))
```

```
## [1] "Linear: 0.917682926829268 , Radial: 0.937282229965157 , Polynomial: 0.930749128919861"
```

STEP 6: EVALUATE THE MODEL

Display Table of Results

Now that we have run our selected machine learning models, we can display the accuracy for each model and testing sceme into a table.

```
#create dataframe of accuracy and models
final_table=as.data.frame(rbind(c(knn.vsa*100,knn.cv*100),c(LDA_VSA*100,lda.cv*100),c(vsa_rf*100,cv_rf*100),c(svm_vsa*100,cv_svm*100)))
final_table$Model=c("KNN","LDA","RF","SVM")
colnames(final_table)=c("VSA Accuracy","10-Fold CV Accuracy","Model")
final_table<- final_table[, c(3,1,2)]

rownames(final_table)=NULL



knitr::kable(final_table,digits=1,caption="Model Accuracy (%) for Validation and CV Approaches")
```

Model Accuracy (%) for Validation and CV Approaches

Model	VSA Accuracy	10-Fold CV Accuracy
KNN	91.8	92.2
LDA	91.4	91.4
RF	94.4	94.8
SVM	91.8	91.7

STEP 7: MAKE PREDICTIONS

- Re-fit model using all available data

Line 694

```
set.seed(314)
rf_deploy=randomForest( Behavior~X_Mean + Y_Mean + Z_Mean + MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max + SMA_Max +
                   X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE + X_SE + Y_SE + Z_SE
                            ,ntree=1000,data=observed_df)
rf_deploy
```

```
##
## Call:
##  randomForest(formula = Behavior ~ X_Mean + Y_Mean + Z_Mean +      MI_Mean + SMA_Mean + X_Max + Y_Max + Z_Max + MI_Max +
SMA_Max +      X_Min + Y_Min + Z_Min + MI_Min + SMA_Min + MI_SE + SMA_SE +      X_SE + Y_SE + Z_SE, data = observed_df, ntre
e = 1000)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 5.07%
## Confusion matrix:
##      G    R  W class.error
## G 6535  195  8  0.03012763
## R  218 4302  0  0.04823009
## W  121   40 59  0.73181818
```

STEP 7: MAKE PREDICTIONS

- Use model to predict unobserved behavior

Line 705

```
Accel_Merged$Behavior=predict(rf_deploy,Accel_Merged)
```

STEP 7: MAKE PREDICTIONS

Line 715

```
#convert date time to only date
  Accel_Merged$Date= as.Date(Accel_Merged$Time, format =  "%m/%d/%Y")

#subset out walk predictions, count the number per day and convert back to minutes
  df_pred_walk=subset(Accel_Merged,Behavior=='W')
  df_pred_walk=aggregate(df_pred_walk$Behavior,by=list(c(df_pred_walk$Date)),FUN=length)
  colnames(df_pred_walk)=c("Date","walk_Min")
  df_pred_walk$walk_Min=(df_pred_walk$walk_Min*5)/60
#subset out graze predictions, count the number per day and convert back to minutes
  df_pred_graze=subset(Accel_Merged,Behavior=='G')
  df_pred_graze=aggregate(df_pred_graze$Behavior,by=list(c(df_pred_graze$Date)),FUN=length)
  colnames(df_pred_graze)=c("Date","graze_Min")
  df_pred_graze$graze_Min=(df_pred_graze$graze_Min*5)/60

#subset out rest predictions, count the number per day and convert back to minutes
  df_pred_rest=subset(Accel_Merged,Behavior=='R')
  df_pred_rest=aggregate(df_pred_rest$Behavior,by=list(c(df_pred_rest$Date)),FUN=length)
  colnames(df_pred_rest)=c("Date","rest_Min")
  df_pred_rest$rest_Min=(df_pred_rest$rest_Min*5)/60
#Combine into one data frame column and calculate total
  df_Total=as.data.frame(cbind(as.character(df_pred_graze$Date),df_pred_graze$graze_Min,df_pred_rest$rest_Min,df_pred_walk$w
alk_Min))
  colnames(df_Total)=c("Date","Graze_Min","Rest_Min","Walk_Min")
  df_Total$Graze_Min=as.numeric(df_Total$Graze_Min)
  df_Total$Rest_Min=as.numeric(df_Total$Rest_Min)
  df_Total$Walk_Min=as.numeric(df_Total$Walk_Min)
  df_Total$Total_Minutes=df_Total$Graze_Min+df_Total$Rest_Min+df_Total$Walk_Min

#print table as output
  knitr::kable(df_Total,digits=0,caption="Daily Behavior Estimates")
```

Subset out Walking Behavior

Count number per day

Convert count to minutes

Combine into one dataframe

STEP 7: MAKE PREDICTIONS

- Daily Behavior Estimate

Date	Graze_Min	Rest_Min	Walk_Min	Total_Minutes
2017-06-10	67	136	15	218
2017-06-11	598	740	102	1440
2017-06-12	526	827	87	1440
2017-06-13	595	735	110	1440
2017-06-14	587	716	136	1440
2017-06-15	276	565	38	879

WHAT CAN WE USE THIS FOR?

- Combine with GPS to identify grazing selection
- Low and High RFI animals
- Changes in behavior
- Incorporate into additional models

APPLICATION EXAMPLE: NET ENERGY FOR ACTIVITY

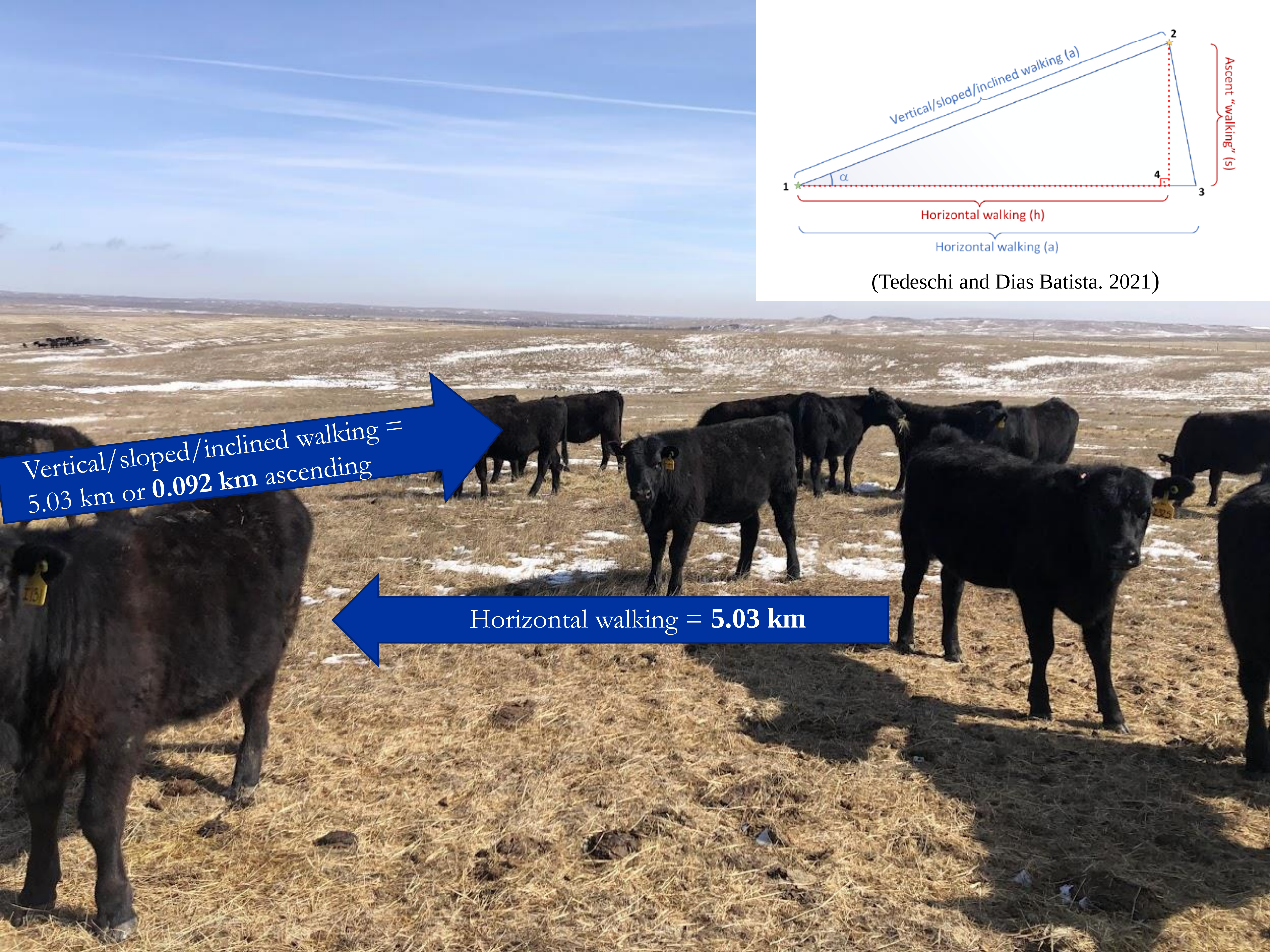
$$\text{Physical Activity} = (0.1 \times Standing + 0.062 \times Position\ Changes +$$

(Agricultural, Research Council, 1980)

Minutes Grazing = 576

Average Daily Distance Traveled (~10.6 km)

Minutes Walking = 108

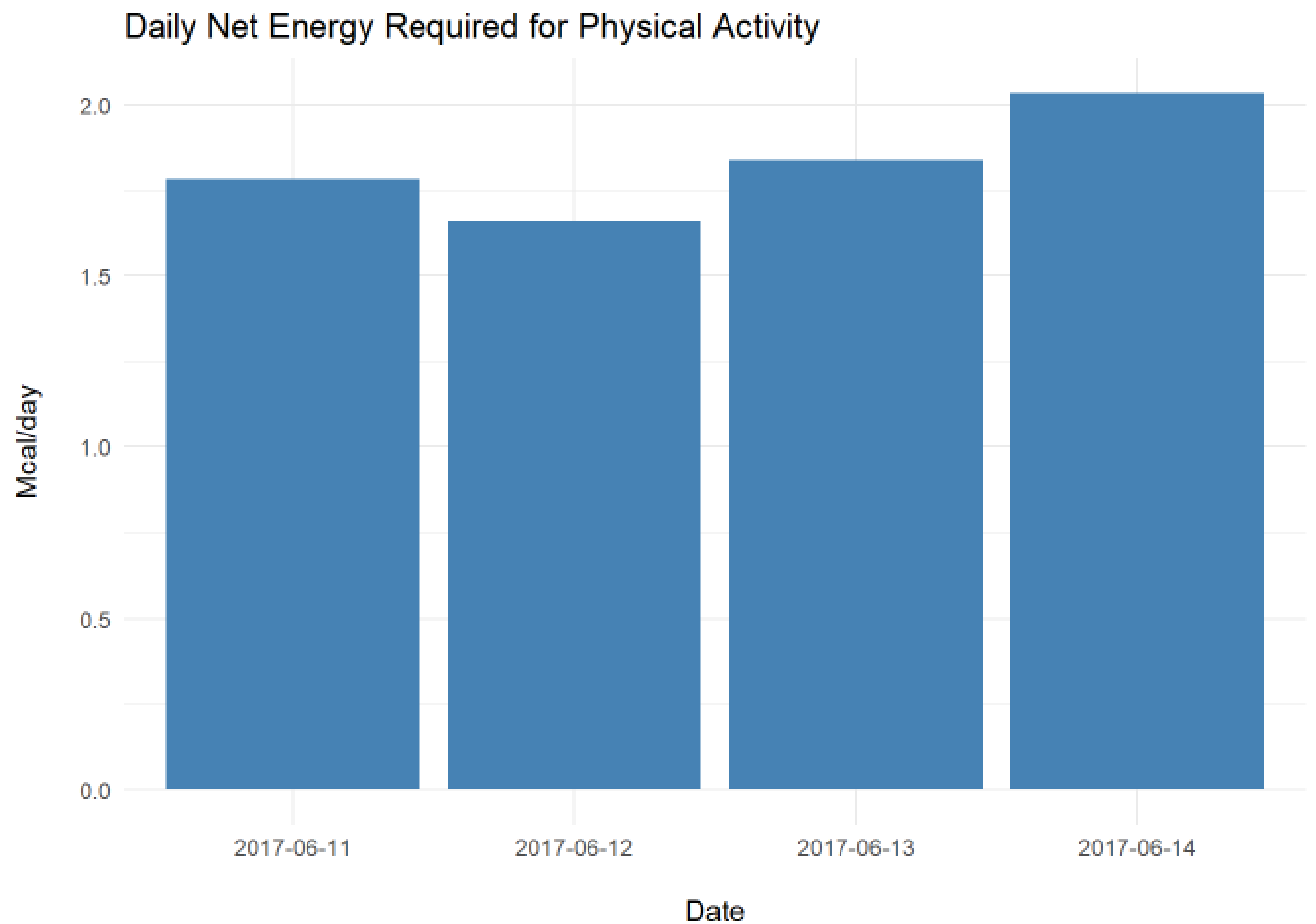


(Tedeschi and Dias Batista. 2021)

Line 754

```
#head(df_NEm_Total$Graze_Min)
df_NEm<-df_Total[2:5,] #whole days only

#### CONVERT REST MINUTES TO HOURS

#df_NEm$Graze_Min=df_NEm$Graze_Min/60
df_NEm$Rest_Min=df_NEm$Rest_Min/60
#df_NEm$Walk_Min=df_NEm$Walk_Min/60



#####COVERT Walk minutes to distance in kilometers per day

Walking_Rate<-1.05/1000 #km per second


Walk_Distance_Per_Min<-Walking_Rate*60 #in Kilometers per minute (i.e, 4 meters/minute * 60 seconds)

df_NEm$Avg_Distance_Walk<-df_NEm$Walk_Min*Walk_Distance_Per_Min
###Avg_Distance_Walk

####
Grazing_Walking_Rate<-0.093/1000 # km per second
Graze_Distance_Per_Min<-Grazing_Walking_Rate*60 #in kilometers/minute
df_NEm$Avg_Distance_Grazed<-df_NEm$Graze_Min*Graze_Distance_Per_Min

###Avg_Distance_Grazed

Fraction_Distance_Flat<-0.5 #Assume that have the distance is traveled on flat ground
df_NEm$Distance_Slope_Km<-(df_NEm$Avg_Distance_Walk+df_NEm$Avg_Distance_Grazed)*(1-Fraction_Distance_Flat)
df_NEm$Distance_flat_Km<-(df_NEm$Avg_Distance_Walk+df_NEm$Avg_Distance_Grazed)*Fraction_Distance_Flat

####################
```

$$\mathbf{NEmr}_{\,act} = \frac{(0.1 \times Standing + 0.062 \times Position\ Change + 0.621 \times Distance_{Flat} + Distance_{Sloped}) \times FWB}{1000}$$



WHY DID WE DO THIS?

- Streamline data processing
  - Can be very labor intensive
- Developing models and processes in code
  - Free up time to spend on research questions
- Jumping off point
  - Animal science examples

THANK YOU

CONTACT INFORMATION

Jameson Brennan

Assistant Professor

Dept. Animal Science

West River Research and Extension Center

Jameson.Brennan@sdstate.edu

REFERENCES

- **Agricultural Research Council. 1980.** The Nutrient Requirements of Ruminant Livestock. Agricultural Research Council. The Gresham Press, London, UK.
- Tedeschi, L. O., and L. F. Dias Batista. 2021. Precision determination of energy and protein requirements of grazing and feedlot animals. Pages 177-204 in Feeding the Future: Precision Nutrition for Tomorrow's Animal. D. Kumar, M.-P. Létourneau-Montminy, L. McKnight, I. Parenteau, R. Petri, S. Robinson, G. Widyaratne and S. Hopkins, eds. Virtual. Animal Nutrition Association of Canada
- Tedeschi and Fox, 2020. BLUE BOOK