data622 homework 2

Daniel Sullivan

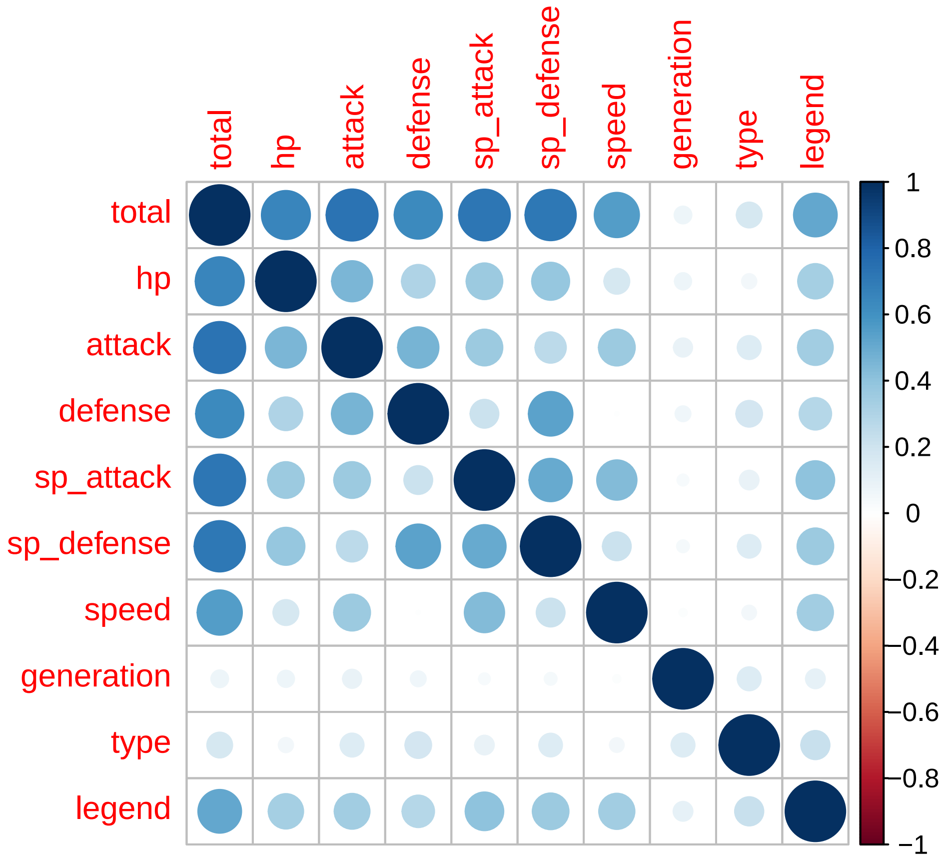
2023-03-26

Data download and exploratory graphs.

```
pokemon_df<-read.csv("https://raw.githubusercontent.com/TheSaltyCrab/Data-622/main/pokemon.csv?token=GHS

pokemon_df$type1[pokemon_df$type1=='Blastoise']<-'Water'
pokemon_df$type1[pokemon_df$type1=='Graass']<-'Grass'

pokemon_df<- pokemon_df%>%mutate(type=case_when(type1=="Grass"~1,type1=="Fire"~2,type1=="Water"~3, type
pokemon_df<- pokemon_df%>%mutate(legend=case_when(legendary=="False"~0,legendary=="True"~1))

pokemon_cor<-pokemon_df %>%
  select(!c(name,number,type1,type2,legendary))
#unique(pokemon_df$type1)
#length(unique(pokemon_df$type1))
#summary(pokemon_trim)
#head(pokemon_trim)
summary(pokemon_cor)
```

```
##      total             hp             attack          defense
##  Min.   : 175.0   Min.   :  1.00   Min.   :  5.00   Min.   :  5.00
##  1st Qu.: 330.0   1st Qu.: 50.00   1st Qu.: 56.00   1st Qu.: 52.00
##  Median : 460.5   Median : 68.00   Median : 80.00   Median : 70.00
##  Mean   : 440.9   Mean   : 70.49   Mean   : 80.94   Mean   : 74.97
##  3rd Qu.: 519.2   3rd Qu.: 84.00   3rd Qu.:100.00   3rd Qu.: 90.00
##  Max.   :1125.0   Max.   :255.00   Max.   :190.00   Max.   :250.00
##    sp_attack        sp_defense         speed          generation
##  Min.   : 10.00   Min.   : 20.00   Min.   :  5.00   Min.   :0.000
##  1st Qu.: 50.00   1st Qu.: 50.00   1st Qu.: 45.00   1st Qu.:2.000
##  Median : 65.00   Median : 70.00   Median : 65.00   Median :4.000
##  Mean   : 73.27   Mean   : 72.48   Mean   : 68.79   Mean   :4.295
##  3rd Qu.: 95.00   3rd Qu.: 90.00   3rd Qu.: 90.00   3rd Qu.:6.000
##  Max.   :194.00   Max.   :250.00   Max.   :200.00   Max.   :8.000
##       type            legend
##  Min.   : 1.000   Min.   :0.0000
##  1st Qu.: 3.000   1st Qu.:0.0000
##  Median : 6.000   Median :0.0000
##  Mean   : 7.718   Mean   :0.1101
##  3rd Qu.:13.000   3rd Qu.:0.0000
##  Max.   :18.000   Max.   :1.0000
```

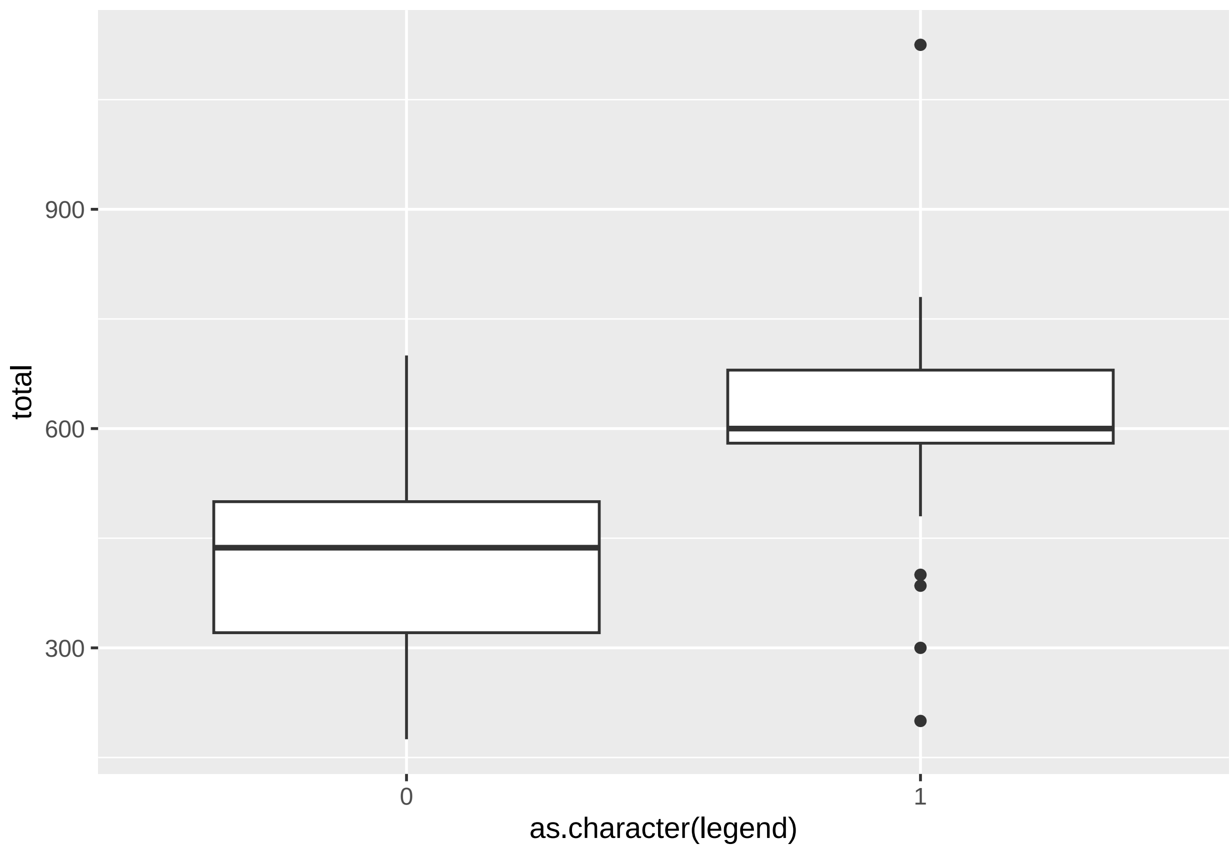
```
corrplot(cor(pokemon_cor))
```



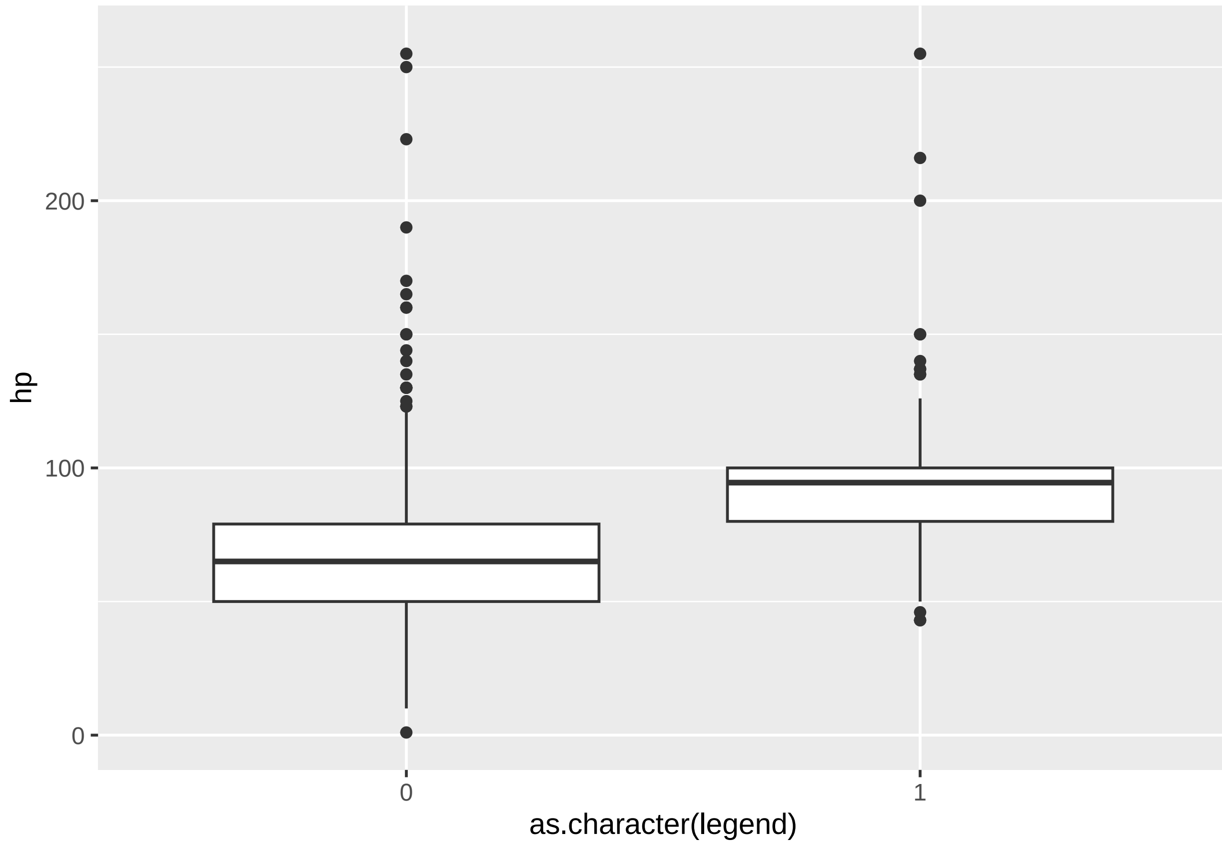

```
#pokemon_df$type
```

With this data my goal was to try and classify my data into legendary pokemon and non-legendary so i began focusing in on that specific column and how it relates to the data.

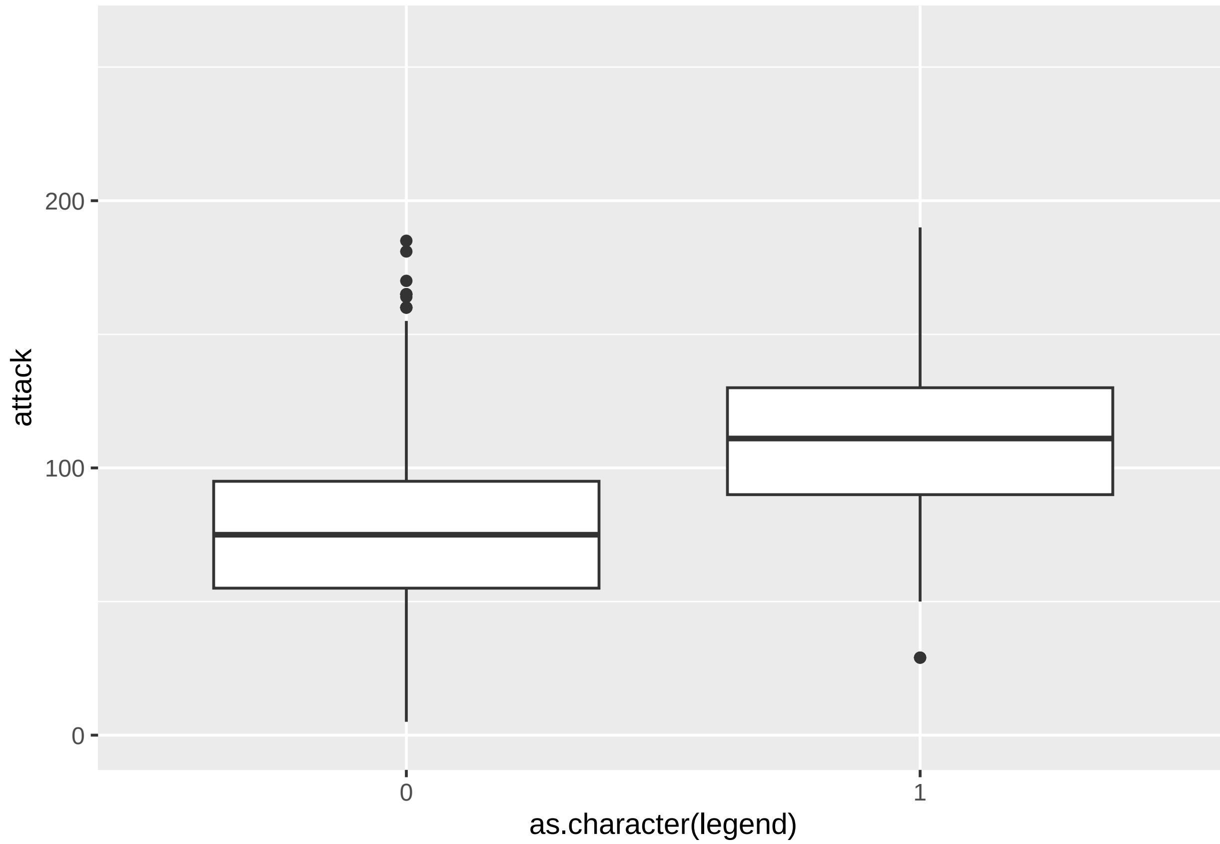
```
ggplot(pokemon_cor, aes(x=as.character(legend), y=total)) +
  geom_boxplot()
```



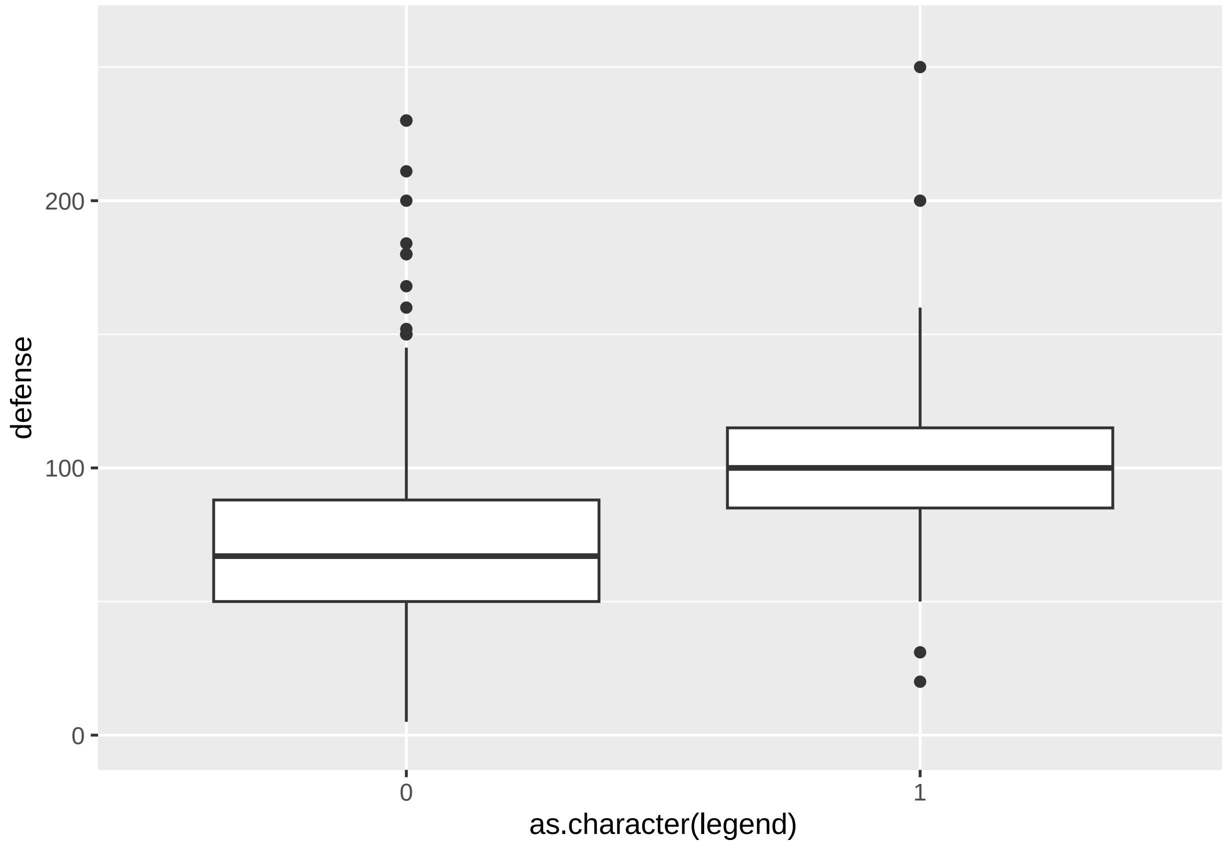

```
ggplot(pokemon_cor, aes(x=as.character(legend), y=hp)) +
  geom_boxplot()+coord_cartesian(ylim = c(0, 260))
```



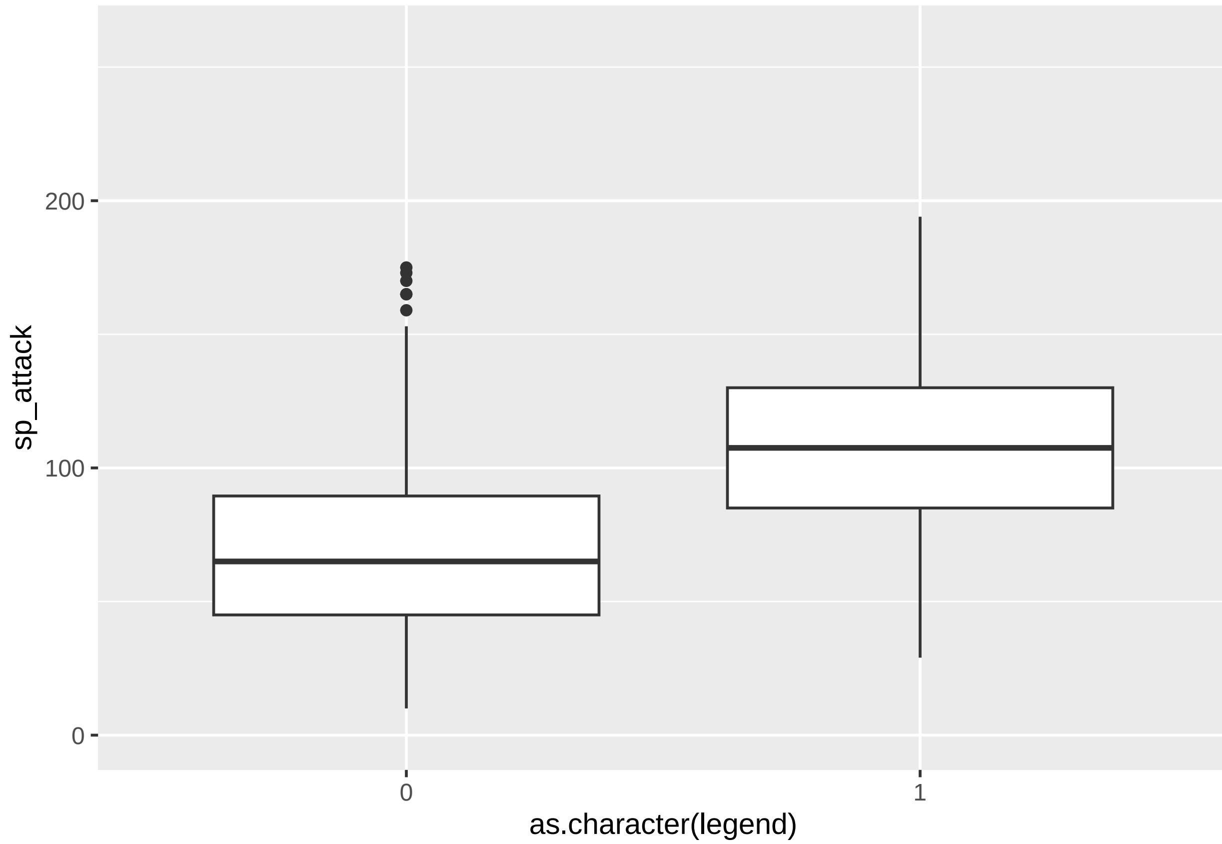

```
ggplot(pokemon_cor, aes(x=as.character(legend), y=attack)) +
  geom_boxplot()+coord_cartesian(ylim = c(0, 260))
```



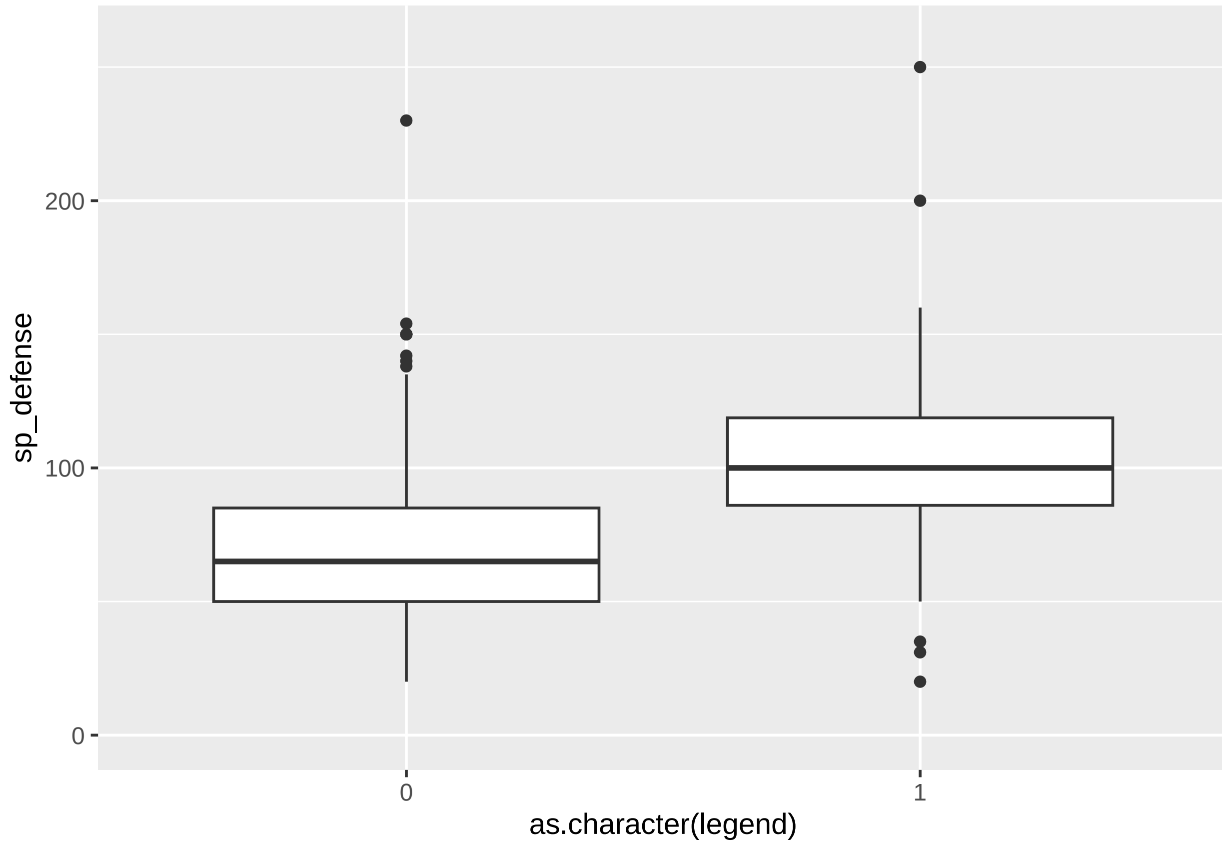

```
ggplot(pokemon_cor, aes(x=as.character(legend), y=defense)) +
  geom_boxplot()+coord_cartesian(ylim = c(0, 260))
```



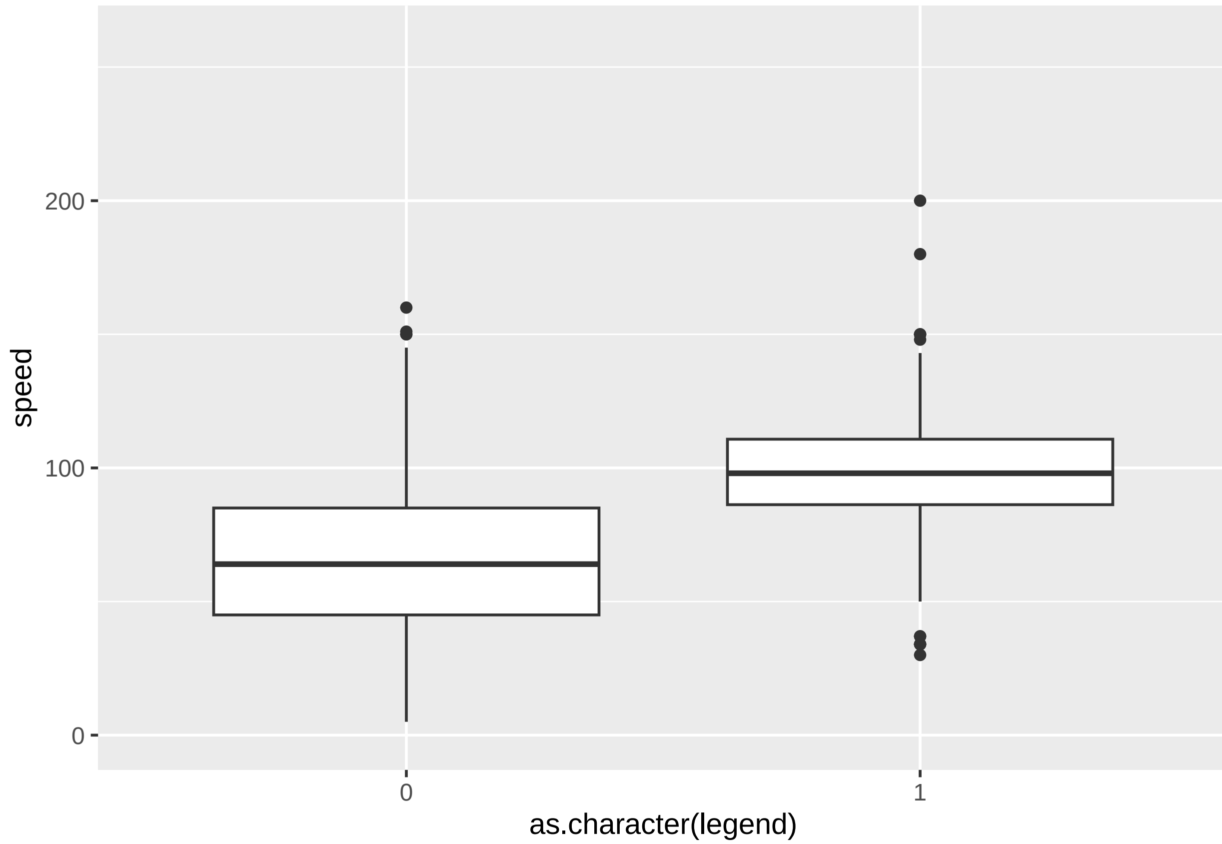

```
ggplot(pokemon_cor, aes(x=as.character(legend), y=sp_attack)) +
  geom_boxplot()+coord_cartesian(ylim = c(0, 260))
```




```
ggplot(pokemon_cor, aes(x=as.character(legend), y=sp_defense)) +
  geom_boxplot()+coord_cartesian(ylim = c(0, 260))
```




```
ggplot(pokemon_cor, aes(x=as.character(legend), y=speed)) +
  geom_boxplot()+coord_cartesian(ylim = c(0, 260))
```

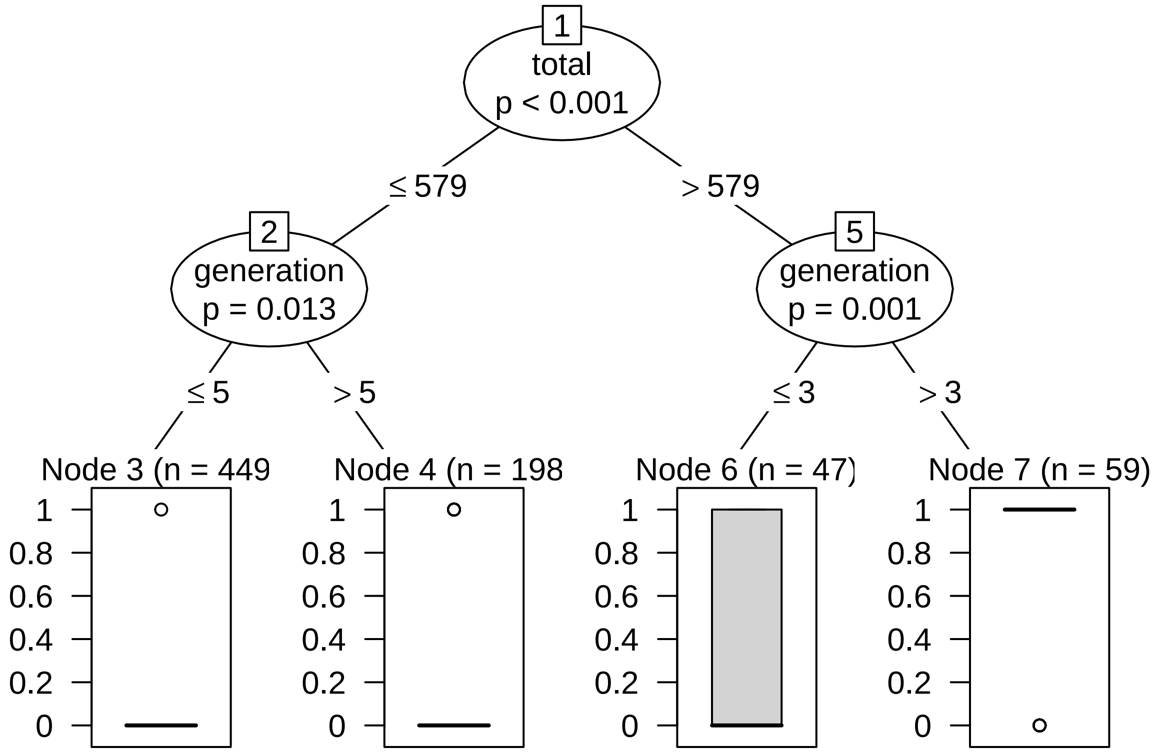


Models

Decision Tree With all variables

created a data partition in order to make my test and train data sets and modeled a decision tree with all variables.

```
set.seed(9)
p = createDataPartition(pokemon_cor$type, p = .7, list = F)
train_p =pokemon_cor[p, ]
#print(train_p$type)
test_p = pokemon_cor[-p, ]

model_allvar<-ctree(legend~.,train_p)
plot(model_allvar)
```




```
prediction1<-round(predict(model_allvar, test_p))
#prediction1[1]
#test_p$legend
cm1<-(confusionMatrix(data = factor(prediction1), reference = factor(test_p$legend)))
cm1
```

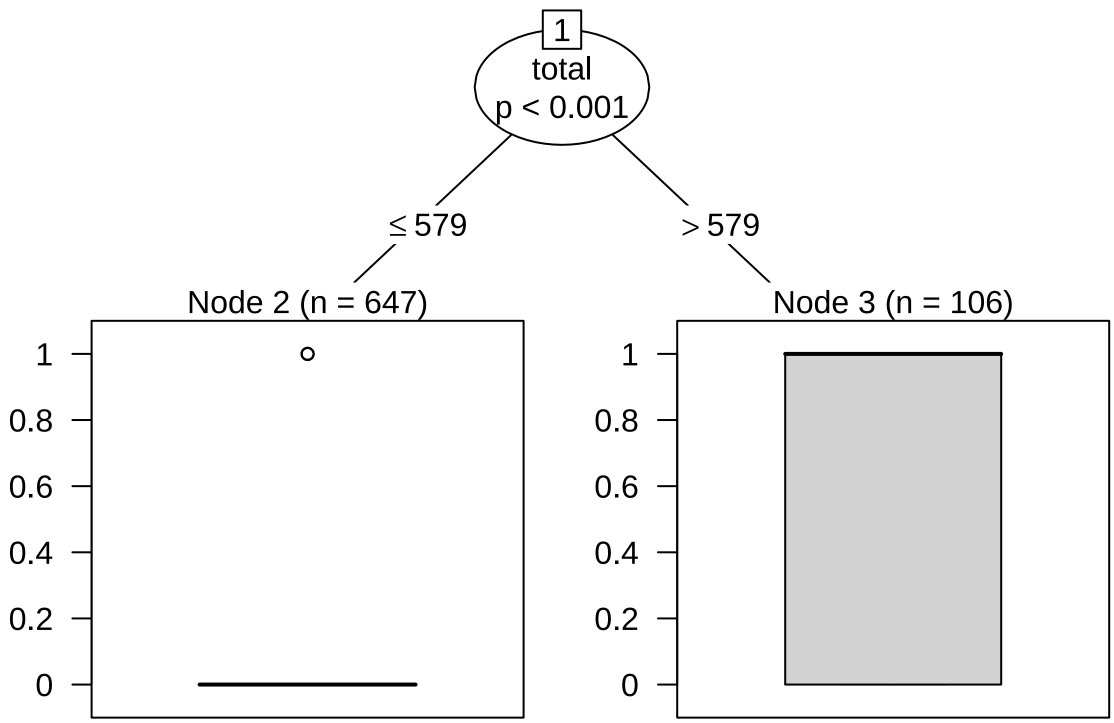
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 279  17
##          1   3  20
##
##                Accuracy : 0.9373
##                  95% CI : (0.9048, 0.9613)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : 0.0009837
##
##                   Kappa : 0.6341
##
##  Mcnemar's Test P-Value : 0.0036504
##
##             Sensitivity : 0.9894
##             Specificity : 0.5405
##          Pos Pred Value : 0.9426
##          Neg Pred Value : 0.8696
```

```
##               Prevalence : 0.8840
##           Detection Rate : 0.8746
##     Detection Prevalence : 0.9279
##        Balanced Accuracy : 0.7650
##
##         'Positive' Class : 0
##
```

Decision Tree With restricted variables

Seeing a weird decision node where it was classifying off of generation which their should not really be any relationship between the two i decided to strip the variables down to total stats, special attack, and attack.

```
model_smallvar<-ctree(legend~total+sp_attack+attack,train_p)
plot(model_smallvar)
```




```
test_small_p<-test_p%>%select(c(total,sp_attack,attack))

prediction2<-round(predict(model_smallvar, test_small_p))
#prediction1[1]
#test_p$legend
for (i in 1:length(prediction2+1)){
  if (prediction2[i] == 1){
    prediction2[i]<-0
  }
```

```r
  else{
    prediction2[i]<-1
  }

}


#prediction2
cm2<-(confusionMatrix(data = factor(prediction2), reference = factor(test_p$legend)))
cm2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  10  30
##          1 272   7
##
##                Accuracy : 0.0533
##                  95% CI : (0.0313, 0.084)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.2019
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.03546
##             Specificity : 0.18919
##          Pos Pred Value : 0.25000
##          Neg Pred Value : 0.02509
##              Prevalence : 0.88401
##          Detection Rate : 0.03135
##    Detection Prevalence : 0.12539
##       Balanced Accuracy : 0.11233
##
##        'Positive' Class : 0
##
```

Random Forest With All Variables

implemented ensemble bagging(random forest) in order to see if their was an improvement with this method.

```r
train_p$legend<-as.factor(train_p$legend)
train_x<-train_p %>% select(!legend)
train_y<-as.factor(train_p$legend)

test_x<-test_p %>% select(!legend)
test_y<-as.factor(test_p$legend)


set.seed(9)
```

```r
model_forest <- randomForest(
  formula = legend ~ .,
  x=train_x,y=train_y, xtest = test_x, ytest = test_y
)

min<-which.min(model_forest$err.rate)

model_forest$confusion
```

```
##     0  1 class.error
## 0 659 13  0.01934524
## 1  16 65  0.19753086
```

```r
model_forest <- randomForest(
  formula = legend ~ .,
  data=train_p, ntree = min
)

predictionT<-predict(model_forest, test_x)
#print(prediction)
#predictionT<-round(predictionT)
#prediction1[]
#test_air$month[]
#test_air$month
cmT<-(confusionMatrix(data = factor(predictionT), reference = factor(test_y)))
cmT
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 278  12
##          1   4  25
##
##                Accuracy : 0.9498
##                  95% CI : (0.9198, 0.9711)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : 3.917e-05
##
##                   Kappa : 0.7301
##
##  Mcnemar's Test P-Value : 0.08012
##
##             Sensitivity : 0.9858
##             Specificity : 0.6757
##          Pos Pred Value : 0.9586
##          Neg Pred Value : 0.8621
##              Prevalence : 0.8840
##          Detection Rate : 0.8715
##    Detection Prevalence : 0.9091
##       Balanced Accuracy : 0.8307
##
##        'Positive' Class : 0
```

```
##
```

```r
#print(model_forest$err.rate)
#plot(model_forest)
#model_forest$forest

#print(min)
```

AdaBoost model All Variables

seeing the random forest hardly improved the classification i wanted to test some boosting tree methods.

```r
model_adaboost <- boosting(legend~., data=train_p, boos=TRUE, mfinal=50)
summary(model_adaboost)
```

```
##            Length Class   Mode
## formula       3   formula call
## trees        50   -none-  list
## weights      50   -none-  numeric
## votes      1506   -none-  numeric
## prob       1506   -none-  numeric
## class       753   -none-  character
## importance    9   -none-  numeric
## terms         3   terms   call
## call          5   -none-  call
```

```r
predict_ada = predict(model_adaboost, test_p)
predict_ada$confusion
```

```
##                Observed Class
## Predicted Class   0   1
##               0 277  10
##               1   5  27
```

```r
print("accuracy")
```

```
## [1] "accuracy"
```

```r
print(1-predict_ada$error)
```

```
## [1] 0.9529781
```

```r
print("sensitivity")
```

```
## [1] "sensitivity"
```

```r
print(27/(27+5))
```

```
## [1] 0.84375
```

```r
print("specificity")
```

```
## [1] "specificity"
```

```r
print(277/(277+10))
```

```
## [1] 0.9651568
```

```r
print("precision")
```

```
## [1] "precision"
```

```r
print(27/(27+10))
```

```
## [1] 0.7297297
```

```r
print("balanced accuracy")
```

```
## [1] "balanced accuracy"
```

```r
print((0.9651568+0.84375)/2)
```

```
## [1] 0.9044534
```

```r
#predict_ada$prob
#cmT<-(confusionMatrix(data = factor(predict_ada), #reference = factor(test_y)))
#cmT
```

Homework 3 SVM vs decision trees

```r
#train_p$legend<-as.factor(train_p$legend)
#train_x<-train_p %>% select(!legend)%>%scale()
#train_y<-as.factor(train_p$legend)

#test_x<-test_p %>% select(!legend)%>%scale()
#test_y<-as.factor(test_p$legend)

train_p$legend<-as.numeric(train_p$legend)

scale_tr_p = scale(train_p)
scale_ts_p = scale(test_p)
scale_tr_p$legend<-as.factor(train_p$legend)
```

```
## Warning in scale_tr_p$legend <- as.factor(train_p$legend): Coercing LHS to a
## list
```

```r
scale_ts_p$legend<-as.factor(test_p$legend)
```

```
## Warning in scale_ts_p$legend <- as.factor(test_p$legend): Coercing LHS to a list
```

```r
#train_p
```

```r
svm_model_lin = svm(formula = legend ~ .,data = train_p,type = 'C-classification',kernel = 'linear')

svm_predict<-as.numeric(predict(svm_model_lin, test_p))
#svm_predict
#svm_predict[0]
for (i in 1:length(svm_predict+1)){
  if (svm_predict[i] == 1){
    svm_predict[i]<-0
  }
  else{
    svm_predict[i]<-1
  }

}
#svm_predict
cm_svm<-(confusionMatrix(data = factor(svm_predict), reference = factor(test_y)))
cm_svm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 279  22
##          1   3  15
##
##                Accuracy : 0.9216
##                  95% CI : (0.8865, 0.9486)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : 0.0182207
##
##                   Kappa : 0.5081
##
##  Mcnemar's Test P-Value : 0.0003182
##
##             Sensitivity : 0.9894
##             Specificity : 0.4054
##          Pos Pred Value : 0.9269
##          Neg Pred Value : 0.8333
##              Prevalence : 0.8840
##          Detection Rate : 0.8746
##    Detection Prevalence : 0.9436
##       Balanced Accuracy : 0.6974
##
##        'Positive' Class : 0
##
```

```r
svm_model_rad = svm(formula = legend ~ .,data = train_p,type = 'C-classification',kernel = 'radial')

svm_predict_rad<-as.numeric(predict(svm_model_rad, test_p))
#svm_predict
#svm_predict[0]
for (i in 1:length(svm_predict_rad+1)){
  if (svm_predict_rad[i] == 1){
    svm_predict_rad[i]<-0
  }
  else{
    svm_predict_rad[i]<-1
  }

}
#svm_predict
cm_svm_rad<-(confusionMatrix(data = factor(svm_predict_rad), reference = factor(test_y)))
cm_svm_rad
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 280  21
##          1   2  16
##
##                Accuracy : 0.9279
##                  95% CI : (0.8938, 0.9537)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : 0.0064643
##
##                   Kappa : 0.5475
##
##  Mcnemar's Test P-Value : 0.0001746
##
##             Sensitivity : 0.9929
##             Specificity : 0.4324
##          Pos Pred Value : 0.9302
##          Neg Pred Value : 0.8889
##              Prevalence : 0.8840
##          Detection Rate : 0.8777
##    Detection Prevalence : 0.9436
##       Balanced Accuracy : 0.7127
##
##        'Positive' Class : 0
##
```

```r
svm_model_pol = svm(formula = legend ~ .,data = train_p,type = 'C-classification',kernel = 'polynomial'

svm_predict_pol<-as.numeric(predict(svm_model_pol, test_p))
#svm_predict
#svm_predict[0]
for (i in 1:length(svm_predict_pol+1)){
  if (svm_predict_pol[i] == 1){
```

```r
    svm_predict_pol[i]<-0
  }
  else{
    svm_predict_pol[i]<-1
  }

}
#svm_predict
cm_svm_pol<-(confusionMatrix(data = factor(svm_predict_pol), reference = factor(test_y)))
cm_svm_pol
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 280  22
##          1   2  15
##
##                Accuracy : 0.9248
##                  95% CI : (0.8901, 0.9512)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : 0.0110779
##
##                   Kappa : 0.5205
##
##  Mcnemar's Test P-Value : 0.0001052
##
##             Sensitivity : 0.9929
##             Specificity : 0.4054
##          Pos Pred Value : 0.9272
##          Neg Pred Value : 0.8824
##              Prevalence : 0.8840
##          Detection Rate : 0.8777
##    Detection Prevalence : 0.9467
##       Balanced Accuracy : 0.6992
##
##        'Positive' Class : 0
##
```

```r
svm_model_sig = svm(formula = legend ~ .,data = train_p,type = 'C-classification',kernel = 'sigmoid')

svm_predict_sig<-as.numeric(predict(svm_model_sig, test_p))
#svm_predict
#svm_predict[0]
for (i in 1:length(svm_predict_sig+1)){
  if (svm_predict_sig[i] == 1){
    svm_predict_sig[i]<-0
  }
  else{
    svm_predict_sig[i]<-1
  }

}
```

```r
#svm_predict
cm_svm_sig<-(confusionMatrix(data = factor(svm_predict_sig), reference = factor(test_y)))
cm_svm_sig
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 270  14
##          1  12  23
##
##                Accuracy : 0.9185
##                  95% CI : (0.8829, 0.9461)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : 0.02882
##
##                   Kappa : 0.593
##
##  Mcnemar's Test P-Value : 0.84452
##
##             Sensitivity : 0.9574
##             Specificity : 0.6216
##          Pos Pred Value : 0.9507
##          Neg Pred Value : 0.6571
##              Prevalence : 0.8840
##          Detection Rate : 0.8464
##    Detection Prevalence : 0.8903
##       Balanced Accuracy : 0.7895
##
##        'Positive' Class : 0
##
```

```r
svm_model_tot = svm(formula = legend ~ total,data = train_p,type = 'C-classification',kernel = 'linear')

svm_predict_tot<-as.numeric(predict(svm_model_tot, test_p))
#svm_predict
#svm_predict[0]
for (i in 1:length(svm_predict_tot+1)){
  if (svm_predict_tot[i] == 1){
    svm_predict_tot[i]<-0
  }
  else{
    svm_predict_tot[i]<-1
  }

}
#svm_predict
cm_svm_tot<-(confusionMatrix(data = factor(svm_predict_tot), reference = factor(test_y)))
cm_svm_tot
```

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

```
## Prediction   0   1
##          0 276  30
##          1   6   7
##
##                Accuracy : 0.8871
##                  95% CI : (0.8472, 0.9197)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : 0.4740597
##
##                   Kappa : 0.2338
##
##  Mcnemar's Test P-Value : 0.0001264
##
##             Sensitivity : 0.9787
##             Specificity : 0.1892
##          Pos Pred Value : 0.9020
##          Neg Pred Value : 0.5385
##              Prevalence : 0.8840
##          Detection Rate : 0.8652
##    Detection Prevalence : 0.9592
##       Balanced Accuracy : 0.5840
##
##        'Positive' Class : 0
##
```

```r
svm_model_spe = svm(formula = legend ~ total,data = train_p,type = 'C-classification',kernel = 'linear'

svm_predict_spe<-as.numeric(predict(svm_model_spe, test_p))
#svm_predict
#svm_predict[0]
for (i in 1:length(svm_predict_spe+1)){
  if (svm_predict_spe[i] == 1){
    svm_predict_spe[i]<-0
  }
  else{
    svm_predict_spe[i]<-1
  }

}
#svm_predict
cm_svm_spe<-(confusionMatrix(data = factor(svm_predict_spe), reference = factor(test_y)))
cm_svm_spe
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 276  30
##          1   6   7
##
##                Accuracy : 0.8871
##                  95% CI : (0.8472, 0.9197)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : 0.4740597
```

```
##
##                   Kappa : 0.2338
##
##  Mcnemar's Test P-Value : 0.0001264
##
##             Sensitivity : 0.9787
##             Specificity : 0.1892
##          Pos Pred Value : 0.9020
##          Neg Pred Value : 0.5385
##              Prevalence : 0.8840
##          Detection Rate : 0.8652
##    Detection Prevalence : 0.9592
##       Balanced Accuracy : 0.5840
##
##        'Positive' Class : 0
##
```