IRIS DATA WITH R

Nicole Landry

Analysing the Iris Data Set with R Language

Hello there! In this document, I will be sharing my run-through of the well-known machine learning data set, the Iris Data Set. The Iris Data Set refers to British statistician Ronald Fisher's data set on the dimensions for three iris species in his 1936 paper titled "The Use of Multiple Measurements in Taxonomic Problems". With this data, I will utilize the r language with packages in plotting and machine learning to explore the data and create a model for predicting iris species.

Installing Our Packages

```
INSTALL.PACKAGES('CARET', REPOS='HTTP://CRAN.RSTUDIO.COM/')

## PACKAGE 'CARET' SUCCESSFULLY UNPACKED AND MD5 SUMS CHECKED
##
## THE DOWNLOADED BINARY PACKAGES ARE IN
##  C:\USERS\NALAN\APPDATA\LOCAL\TEMP\RTMP04XKYF\DOWNLOADED_PACKAGES
```

The caret package stands for "Classification And REgression Training". It is a machine learning focused package, much like Sci-Kit Learn from Python. This package will contain the functions I will use for splitting the data for training and creating models later on. As I install this package in an IDE, it also installs other important libraries for us like dplyr, ggplot2, and tibble. It's a lot like tidyverse, except it has the added benefit of the machine learning features.

```
LIBRARY(GGPLOT2)
LIBRARY(DPLYR)
```

Here I'm loading ggplot2 and dplyr into the environment so they are ready to use. These two specific packages will do the leg work of getting to know the data.

Loading in the Iris data

```
DATA(IRIS)
```

Since the iris data set is included within R, all we have to is run data() to bring it into our environment

```
HEAD(IRIS)

##   SEPAL.LENGTH SEPAL.WIDTH PETAL.LENGTH PETAL.WIDTH SPECIES
## 1          5.1         3.5          1.4         0.2  SETOSA
## 2          4.9         3.0          1.4         0.2  SETOSA
## 3          4.7         3.2          1.3         0.2  SETOSA
## 4          4.6         3.1          1.5         0.2  SETOSA
## 5          5.0         3.6          1.4         0.2  SETOSA
## 6          5.4         3.9          1.7         0.4  SETOSA
```

Above we see the first few entries in our data as well as our given column names using the head() function. From this we can also tell that our data is already formatted into a data frame for us. To get to know the data, let's view the number of records(rows) and summarize the columns.

```
NROW(IRIS)
```

```
## [1] 150
```

We have 150 entries

The previous functions have either come from base R language or from Utils package (which comes with the standard installation of R, short for utility functions). Let's move on to using some of other packages!

```
SUMMARY(IRIS)
```

```
##   SEPAL.LENGTH    SEPAL.WIDTH     PETAL.LENGTH    PETAL.WIDTH
##  MIN.   :4.300   MIN.   :2.000   MIN.   :1.000   MIN.   :0.100
##  1ST QU.:5.100   1ST QU.:2.800   1ST QU.:1.600   1ST QU.:0.300
##  MEDIAN :5.800   MEDIAN :3.000   MEDIAN :4.350   MEDIAN :1.300
##  MEAN   :5.843   MEAN   :3.057   MEAN   :3.758   MEAN   :1.199
##  3RD QU.:6.400   3RD QU.:3.300   3RD QU.:5.100   3RD QU.:1.800
##  MAX.   :7.900   MAX.   :4.400   MAX.   :6.900   MAX.   :2.500
##        SPECIES
##  SETOSA    :50
##  VERSICOLOR:50
##  VIRGINICA :50
##
##
##
```

Here we see some spread information for each of our columns using the summary() function from dplyr. The dplyr package is a tool for data manipulation. It's somewhat similar to pandas from Python in that it can also allow us to filter and select from data frames very easily. From the use of summary(), we get our statistical averages for our numerical variables and counts for our categorical variables. We also see that we have 50 entries of each species.

```
COLSUMS(IS.NA(IRIS))
```

```
## SEPAL.LENGTH  SEPAL.WIDTH PETAL.LENGTH  PETAL.WIDTH      SPECIES
##            0           0            0            0            0
```

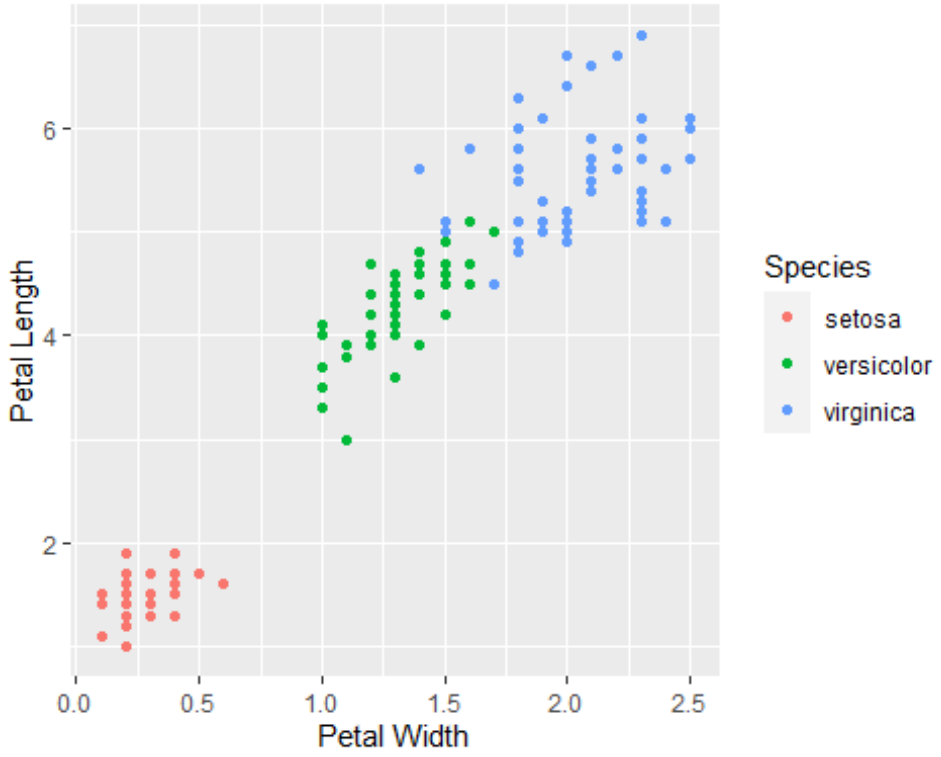
And just to make sure, we also have no NA values! If we did have any NA values, then we would consider using tidyr package's drop_na() or replace_na(). Otherwise we would run into lots of issues with our machine learning. Tidyr package is a good compliment to dplyr and together they work just about as well as pandas.

Visualizing the Data

We now know a little bit about our iris data. It's also already in a nice data frame with comprehensible columns names, so let's map some variables into plots!

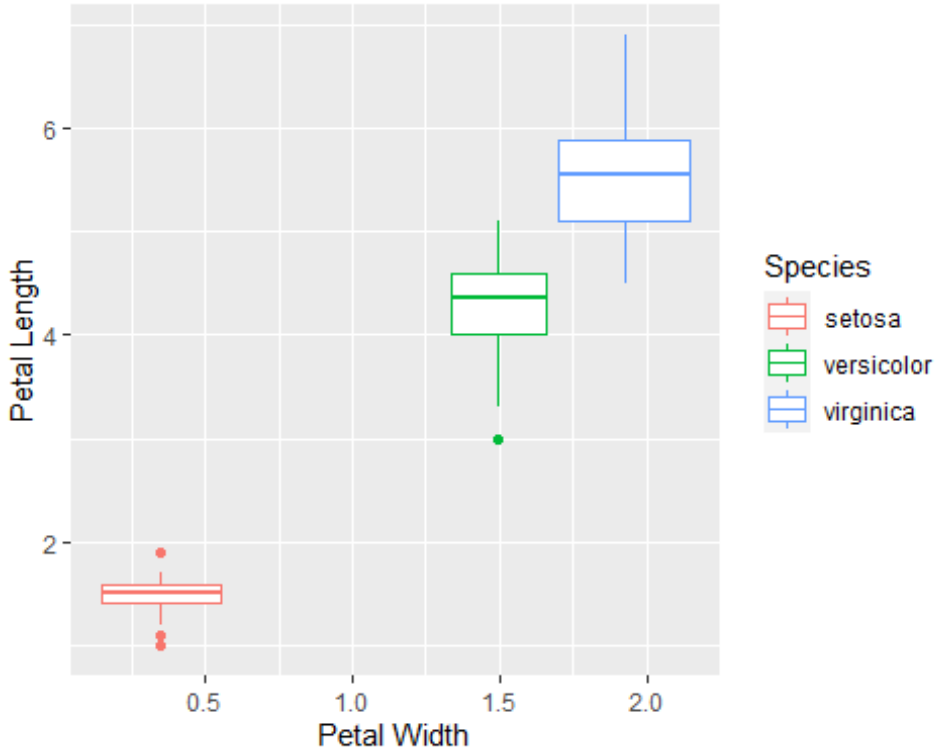
Now we are going to start using another important library, ggplot2 to represent our data visually. Ggplot2 acts as our matplotlib from Python. The "gg" stands for grammar of graphics which makes sense because ggplot2 is all about creating charts, plots, and other variable defined visuals.

```
OPTIONS(REPR.PLOT.WIDTH = 5, REPR.PLOT.HEIGHT = 4)
IRIS_PLOT <-GGPLOT(IRIS,AES(Y=PETAL.LENGTH,X=PETAL.WIDTH,COL=SPECIES))+GEOM_POINT()+XLAB
("PETAL WIDTH")+YLAB("PETAL LENGTH")
IRIS_PLOT
```



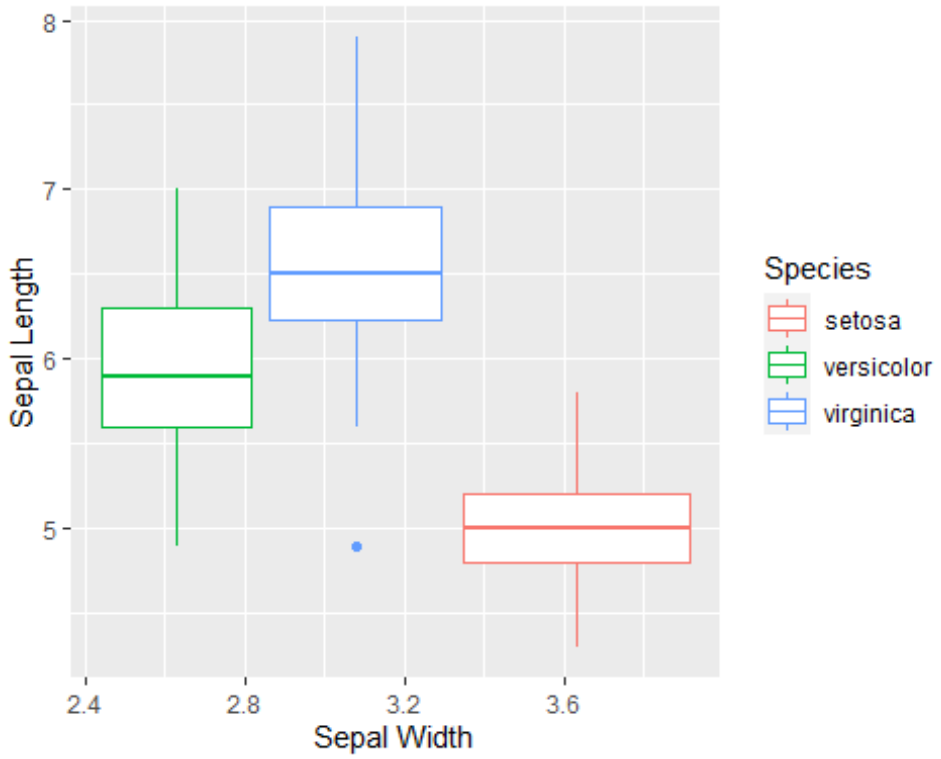
With this first plot, I placed petal width on the x and sepal length on the y and colored by species. Notice how distinctively the species group for these values!

```
OPTIONS(REPR.PLOT.WIDTH = 5, REPR.PLOT.HEIGHT = 4)
IRIS_BOX <- GGPLOT(IRIS,AES(Y=PETAL.LENGTH,X=PETAL.WIDTH,COL=SPECIES))+GEOM_BOXPLOT()+XLA
B("PETAL WIDTH")+YLAB("PETAL LENGTH")
IRIS_BOX
```



Here I've taken the same petal values for x and y as in the first plot and created box plots showing the spread and outliers by species. We can tell so far that setosa is obviously the smallest for petal sizes while virginica gives us the largest.

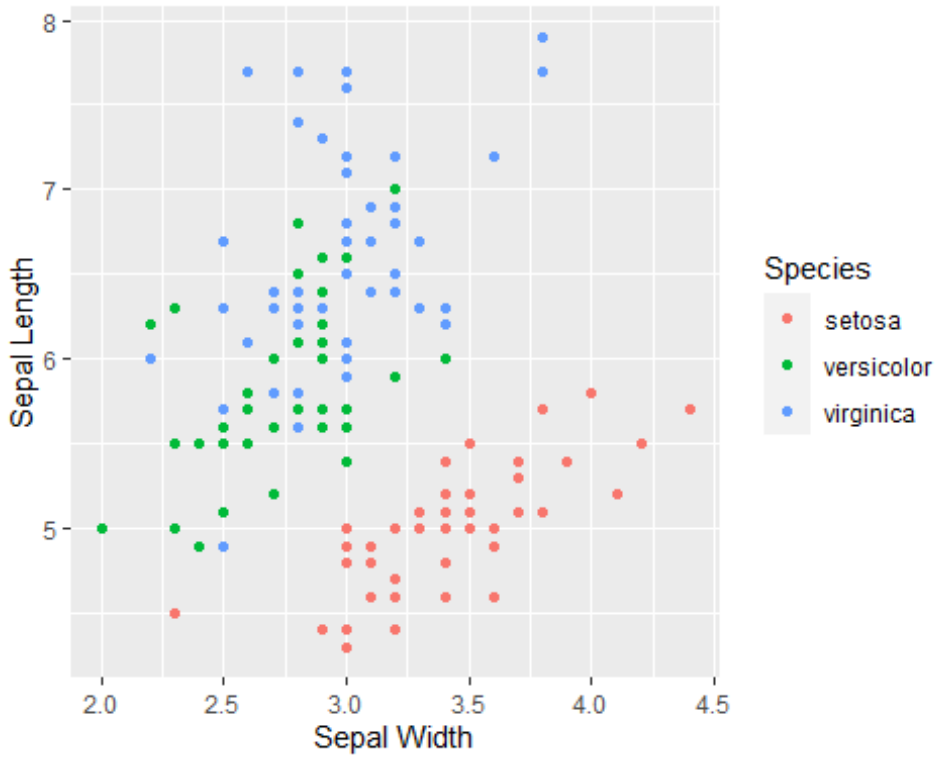
```
OPTIONS(REPR.PLOT.WIDTH = 5, REPR.PLOT.HEIGHT = 4)
IRIS_BOX2 <- GGPLOT(IRIS,AES(Y=SEPAL.LENGTH,X=SEPAL.WIDTH,COL=SPECIES))+GEOM_BOXPLOT()+X
LAB("SEPAL WIDTH")+YLAB("SEPAL LENGTH")
IRIS_BOX2
```



Here we have a similar box plot as before, except now we have sepal width versus sepal length. Notice how the sizing order has been somewhat flipped; setosa has the widest sepal on average. Virginica has the longest sepal length and a slighter greater width than veriscolor, but these two species are much closer in value and with greater spread than setosa.

To see this visualized in a different format, I'm going to plot these same sepal x and y values into a scatter plot like we did with petal values.

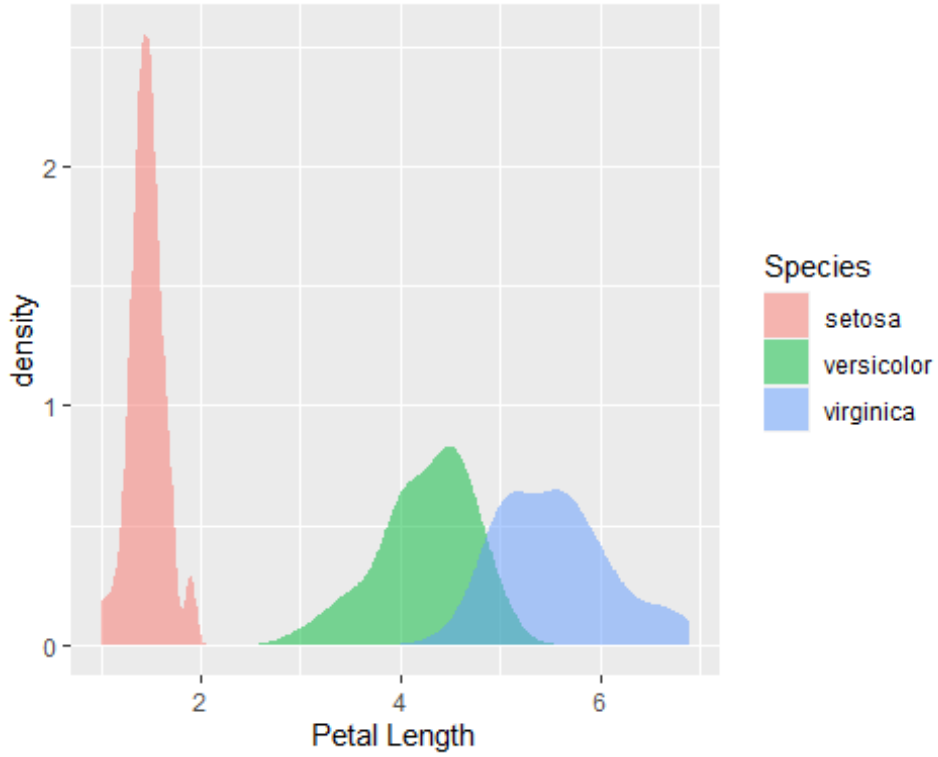
```
OPTIONS(REPR.PLOT.WIDTH = 5, REPR.PLOT.HEIGHT = 4)
IRIS_PLOT2 <-GGPLOT(IRIS,AES(Y=SEPAL.LENGTH,X=SEPAL.WIDTH,COL=SPECIES))+GEOM_POINT()+XLA
B("SEPAL WIDTH")+YLAB("SEPAL LENGTH")
IRIS_PLOT2
```



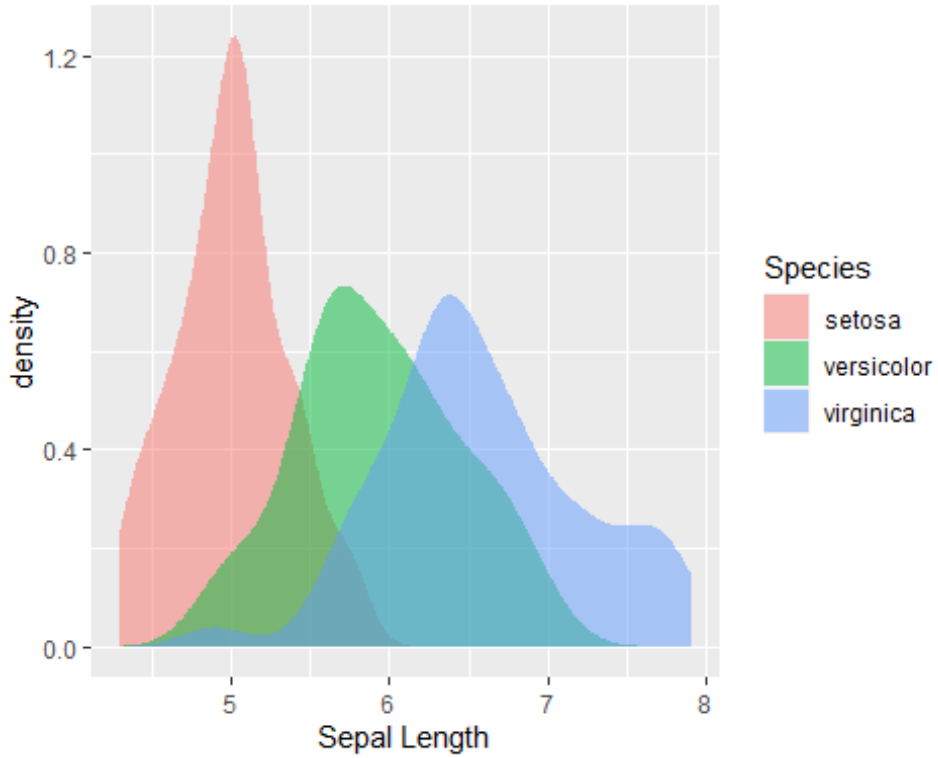
From this plot we see that unlike with petal dimensions, the species start to get much more muddled when viewing sepal dimensions, especially with virginica and setosa.

To further illustrate how the iris species overlap, I'm going to create two density plots, one for petal length and one for sepal length.

```
IRIS_DEN1 <- GGPLOT(IRIS,AES(PETAL.LENGTH,FILL=SPECIES))+GEOM_DENSITY(LINETYPE=0, ALPHA=.
5)+XLAB("PETAL LENGTH")
IRIS_DEN1
```




```
IRIS_DEN2 <- GGPLOT(IRIS,AES(SEPAL.LENGTH, FILL=SPECIES))+GEOM_DENSITY(LINETYPE=0,ALPHA=.
5)+XLAB("SEPAL LENGTH")
IRIS_DEN2
```



With these two plots, we are seeing the distribution using geom_density() which acts like a histogram expect it computes a smoothed estimated curve instead of bins. Like we saw in the scatter plots, virginica and versicolor most frequently overlap while setosa tends to be more segregated. With sepal length in particular, all three species are much harder to distinguish.

Machine Learning

So far, we've loaded in our data, looked at it's structure as a data frame with some dplyr, and now compared the iris species visually with ggplot2 charts. We've seen there is some natural clustering within certain iris measurements and certain species, but this is not universally true across the entire data set of variables. If we want to predict iris species with the greatest accuracy, we would likely want to factor in all our variables. This isn't something that could easily be done manually or with the help of two dimensional plots, so instead we look to more advanced machine learning algorithms to do the work for us.

Now we are moving on into using the caret package. The caret package has hundreds of machine learning algorithms plus the tools to tune and compare the machine learning models. As I mentioned before, I will use this package in the same I would use sci-kit learn in Python. We know that for our iris data we are hoping to predict the species, which is categorical variable, so our models should fall under the category of classification models. But before we try using any specific model, we need to partition are data for model training.

```
LIBRARY(CARET)
```

Splitting the Data

```
INDEX<- CREATEDATAPARTITION(IRIS$SPECIES, P=0.70, LIST=FALSE)
```

With the createDataPartition() function, we are splitting our data by a desired percent overall, which I have chosen to be 70%, and defining which column we want to maintain the same proportions of data within each split, which in this case is Species. We have equal parts of each iris species in the overall data, so we want to preserve that in order to have the most accurately predicting model.

```
TESTSET <- IRIS[-INDEX,]
TRAINSET <- IRIS[INDEX,]
```

The createDataPartition() function returns a matrix that we can then use to index the overall iris data. Here I'm renaming these splits into test and train sets. Splitting and training a model with a select portion of the data is necessary for machine learning in order to compare how the model predicts data it has not seen before. Otherwise, the model could predict the given data set well, but it could perform horribly when predicting new data in another context, otherwise known as "overfitting".

Fitting and Predicting the First Model

For our first model, I've decide to try using a random forest model. Random Forest is a type of classification that starts with a decision tree that makes branches by choosing between variable values. The branches eventually narrow to a particular predicted class for the data points. With a

random forest model, many decision trees are made, then the predictions from all the trees are aggregated to create a final class prediction.

```
set.seed(123)
rf_1<- train(Species~.,data=trainset,method ="rf")
rf_1

## Random Forest
##
## 105 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 105, 105, 105, 105, 105, 105, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9499706  0.9242589
##   3     0.9500517  0.9243389
##   4     0.9500260  0.9243752
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 3.
```

First, we set a seed to control the randomization in our fitting, so we can rerun this function and get the same output. Here I am using train() to predict Species using all other variables from the trainset data and establishing the method to be a random forest model with "rf". We could modify the parameters, but first let's see how it does with the default.

If we call rf_1, we can see the resampling info, the sample sizes, and the accuracy results for varying levels of mtry. The mtry represents how many variables the random forest model will split on, or how many "tree nodes" are on the decision tree. The final mtry for this model is 3 since it had the highest accuracy score. We're past 90%, so so far the accuracy and Kappa are looking pretty good for this model.

```
pred_1<- predict(rf_1,testset)
```

Now to see how the model does on new data, I am creating predictions with our trained model on the testset data. Then, I'm going to compare the predictions to the actual values with a confusion matrix

```
confusionMatrix(pred_1,testset$Species)

## Confusion Matrix and Statistics
##
##            Reference
## Prediction   setosa versicolor virginica
```

```
##   setosa         15          0         0
##   versicolor      0         14         0
##   virginica       0          1        15
##
## Overall Statistics
##
##                Accuracy : 0.9778
##                  95% CI : (0.8823, 0.9994)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9667
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.9333           1.0000
## Specificity                 1.0000            1.0000           0.9667
## Pos Pred Value              1.0000            1.0000           0.9375
## Neg Pred Value              1.0000            0.9677           1.0000
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3111           0.3333
## Detection Prevalence        0.3333            0.3111           0.3556
## Balanced Accuracy           1.0000            0.9667           0.9833
```

With the confusion matrix, we can see what species was predicted for each entry and what species the entries actually were. We do see that we misclassified versicolor and virginica. This isn't surprising since we saw earlier how close versicolor and virginica aligned in size. Our accuracy for these predictions was also very high. We could call it and say we are happy with this model, but let's see how other model algorithms would fair with this data.

With caretList(), we can specify a number of models and run them on the data all at once! But we also need to install caretEnsemble to use it.

```
install.packages("caretEnsemble", repos='http://cran.rstudio.com/')

## package 'caretEnsemble' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\nalan\AppData\Local\Temp\RtmpO4xkyF\downloaded_packages

library(caretEnsemble)

algorithmList<-c('rpart','knn','svmPoly')
```

With caretEnsemble loaded, I'm going to try three more models: "rpart" recursive partitioning with regression trees, "knn" k-nearest neighbors, and a "svmPoly" support vector model with polynomial

kernel. All three of these are types of classification algorithms. Rpart is a recursive decision tree model but with only one tree, unlike random forest. KNN is a means of predicting a data point's class type with a formula to determine the class of the next closest data point. And finally, support vector models classify by attempting to partition data into hyper-planes by class, in this case with a polynomial kernel.

```
set.seed(123)
models <- caretList(Species~.,data=trainset,methodList=algorithmList)
```

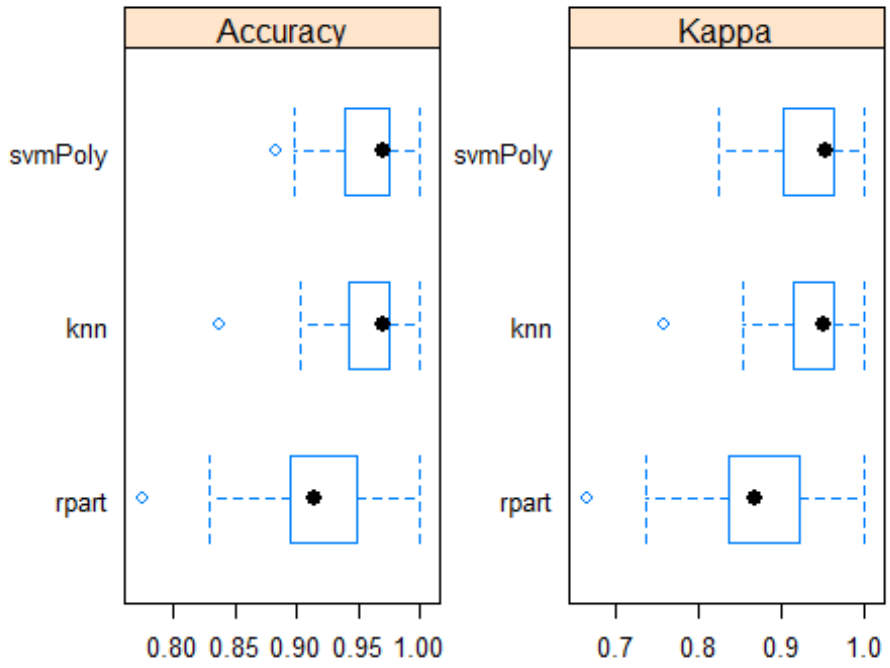
We set our seed again then call caretList() with our list of algorithms. We acquire and save our results over 25 samples of our data with resamples(), and then finally we summarize them, giving us the new spread of accuracy and kappa values.

```
set.seed(123)
results <- resamples(models)
summary(results)

##
## Call:
## summary.resamples(object = results)
##
## Models: rpart, knn, svmPoly
## Number of resamples: 25
##
## Accuracy
##            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## rpart   0.7750000 0.8947368 0.9142857 0.9164939 0.9487179    1    0
## knn     0.8372093 0.9428571 0.9696970 0.9565680 0.9756098    1    0
## svmPoly 0.8837209 0.9393939 0.9696970 0.9552013 0.9750000    1    0
##
## Kappa
##            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## rpart   0.6660482 0.8376068 0.8685857 0.8733438 0.9212121    1    0
## knn     0.7580386 0.9135802 0.9519774 0.9341578 0.9630631    1    0
## svmPoly 0.8254870 0.9022989 0.9544828 0.9318167 0.9624765    1    0
```

Let's view the results in a simple box plot.

```
scales <- list(x=list(relation="free"),y=list(relation="free"))
bwplot(results, scales=scales)
```



And now we see how the spreads of the scores compare visually! All four of the different models performed quite well lucky for us. From here, we could try fine tuning one of these models, but we might not get higher results. There is even a chance different parameters could weaken the models! However, for the sake of classifying flower species, I think we've certainly done the job with enough accuracy.