**Here are some examples of Machine Learning or Data Science in R:**

The first article give the links to install R and R Studio in order to do quick analysis.

**Install R and RStudio on Windows 7, 8, and 10**

[**https://medium.com/@GalarnykMichael/install-r-and-rstudio-on-windows-5f503f708027**](https://medium.com/@GalarnykMichael/install-r-and-rstudio-on-windows-5f503f708027)

**Linear Regression using R**

[**https://medium.com/@GalarnykMichael/univariate-linear-regression-using-r-programming-3db499bdd1e3**](https://medium.com/@GalarnykMichael/univariate-linear-regression-using-r-programming-3db499bdd1e3)

Here is what you get more or less. The format is different in R Studio (Main Input Screen:

|  |
| --- |
| R version 3.5.1 (2018-07-02) -- "Feather Spray"  Copyright (C) 2018 The R Foundation for Statistical Computing  Platform: x86\_64-w64-mingw32/x64 (64-bit)  R is free software and comes with ABSOLUTELY NO WARRANTY.  You are welcome to redistribute it under certain conditions.  Type 'license()' or 'licence()' for distribution details.  R is a collaborative project with many contributors.  Type 'contributors()' for more information and  'citation()' on how to cite R or R packages in publications.  Type 'demo()' for some demos, 'help()' for on-line help, or  'help.start()' for an HTML browser interface to help.  Type 'q()' to quit R.  > # Linear Regression predicts linear relationship between two variables  >  > # Set path to Desktop  > set wd("~/Desktop")  Error: unexpected symbol in "set wd"  >  > download.file(url = 'https://raw.githubusercontent.com/mGalarnyk/Python\_Tutorials/master/Python\_Basics/Linear\_Regression/linear.csv', destfile = 'linear.csv')  trying URL 'https://raw.githubusercontent.com/mGalarnyk/Python\_Tutorials/master/Python\_Basics/Linear\_Regression/linear.csv'  Content type 'text/plain; charset=utf-8' length 2689 bytes  downloaded 2689 bytes  > rawData=read.csv("linear.csv", header=T)  >  > # Show first n entries of data.frame, notice NA values  > head(rawData, 10)  x y  1 82.58321982 134.907414  2 73.92246618 134.085180  3 34.88744536 NA  4 61.83998269 114.530638  5 16.77594025 31.376437  6 0.01673734 8.764634  7 44.45764646 73.285341  8 10.34490930 18.859865  9 42.76713229 72.946609  10 20.85663802 28.637286  >  > linModel <- lm(y~x, data = rawData)  >  > # Show attributes of linModel  > attributes(linModel)  $`names`  [1] "coefficients" "residuals" "effects" "rank" "fitted.values" "assign" "qr"  [8] "df.residual" "na.action" "xlevels" "call" "terms" "model"  $class  [1] "lm"  >  > # To show what happens with na.action, "omit" since data has NA  > linModel$na.action  3 52 59 61 63 64 91 93  3 52 59 61 63 64 91 93  attr(,"class")  [1] "omit"  >  > # Show coefficients of model  > linModel$coefficients  (Intercept) x  4.470197 1.583197  >  > # Predicting New Value based on our model  > predict(linModel, data.frame(x = 3))  1  9.219787  >  > plot(y ~ x, data = rawData,  + xlab = "This labels the x axis",  + ylab = "This labels the y axis",  + main = "Scatter Plot"  + )  >  > abline(linModel, col = "red", lwd = 3) |
|  |
| |  | | --- | | > | |

**#############################################################################**

**# Iris Flowers Analysis - Hello World Machine Learning**

**# From Machine Learning Mastery**

**# Modifications by Patrick Hagan**

**# Date: September 13, 2018**

**# CRAN R:** [**https://cran.r-project.org/**](https://cran.r-project.org/)

**# R Studio:** [**https://www.rstudio.com/products/rstudio/download/**](https://www.rstudio.com/products/rstudio/download/)

**#############################################################################**

**# Load Libraries**

**library(caret)**

**# Load iris dataset into R Studio environment running R**

**data(iris)**

**# rename the dataset as dataframe**

**df <- iris**

**# Split out validation dataset create a list of 80% of the**

**# rows in the original dataset we can use to train and test**

**validation\_index <- createDataPartition(df$Species, p=0.80, list=FALSE)**

**# select 20% of the data for validation**

**validation <- df[-validation\_index,]**

**# use the remaining 80% of data to training and testing the models**

**df <- df[validation\_index,]**

**validation\_index <- createDataPartition(df$Species, p=0.80, list=FALSE)**

**# select 20% of the data for validation**

**validation <- df[-validation\_index,]**

**# use the remaining 80% of data to training and testing the models**

**df <- df[validation\_index,]**

**# dataframe dimensions**

**dim(df)**

**# list types for each attribute**

**sapply(df, class)**

**# display first 5 rows of the data**

**head(df)**

**# list the levels for the class**

**levels(df$Species)**

**# split input and output**

**x <- df[,1:4]**

**y <- df[,5]**

**# summarize the class distribution**

**percentage <- prop.table(table(df$Species)) \* 100**

**cbind(freq=table(df$Species), percentage=percentage)**

**# summarize attribute distributions**

**summary(df)**

**# boxplots for numeric**

**par(mfrow=c(1,4))**

**for(i in 1:4) {**

**boxplot(x[,i], main=names(df)[i])**

**}**

**# barplot for class breakdown**

**plot(y)**

**# scatterplot matrix**

**featurePlot(x=x, y=y, plot="ellipse")**

**# box and whisker plots for each attribute**

**featurePlot(x=x, y=y, plot="box")**

**# density plots for each attribute by class value**

**scales <- list(x=list(relation="free"), y=list(relation="free"))**

**featurePlot(x=x, y=y, plot="density", scales=scales)**

**# Run algorithms using 10-fold cross validation**

**control <- trainControl(method="cv", number=10)**

**metric <- "Accuracy"**

**# LDA**

**set.seed(7)**

**fit.lda <- train(Species~., data=df, method="lda", metric=metric, trControl=control)**

**# CART**

**set.seed(7)**

**fit.cart <- train(Species~., data=df, method="rpart", metric=metric, trControl=control)**

**# kNN**

**set.seed(7)**

**fit.knn <- train(Species~., data=df, method="knn", metric=metric, trControl=control)**

**# SVM**

**set.seed(7)**

**fit.svm <- train(Species~., data=df, method="svmRadial", metric=metric, trControl=control)**

**# RANDOM FOREST**

**set.seed(7)**

**fit.rf <- train(Species~., data=df, method="rf", metric=metric, trControl=control)**

**# compare algorithms**

**results <- resamples(list(lda=fit.lda, cart=fit.cart, knn=fit.knn, svm=fit.svm, rf=fit.rf))**

**summary(results)**

**dotplot(results)**

**print(fit.cart)**

**print(fit.knn)**

**print(fit.lda)**

**print(fit.rf)**

**print(fit.svm)**

**# Estimate Skill on Validation dataframe**

**set.seed(7)**

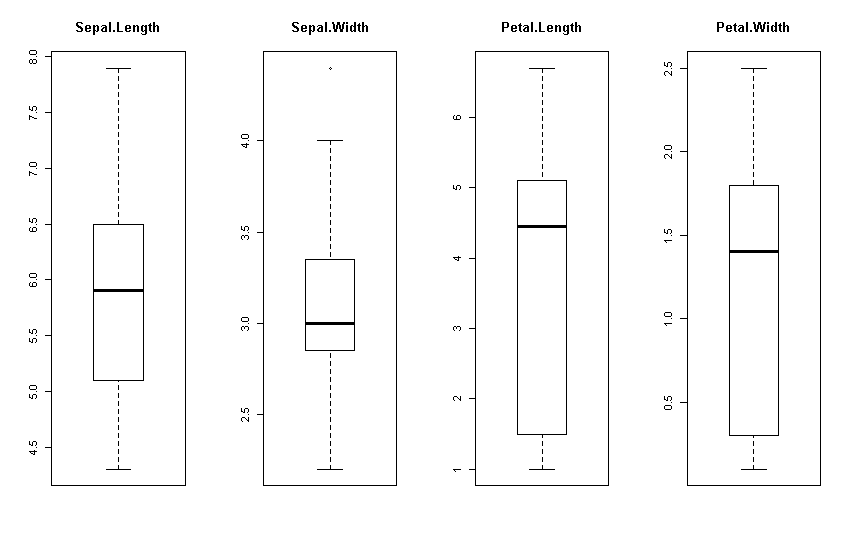
**predictions <- predict(fit.lda, newdata=validation)**

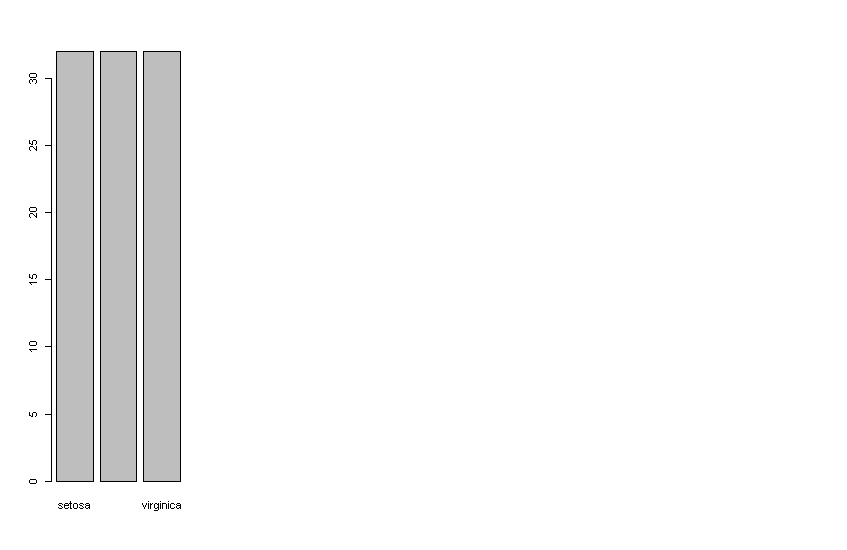
**confusionMatrix(predictions, validation$Species)**

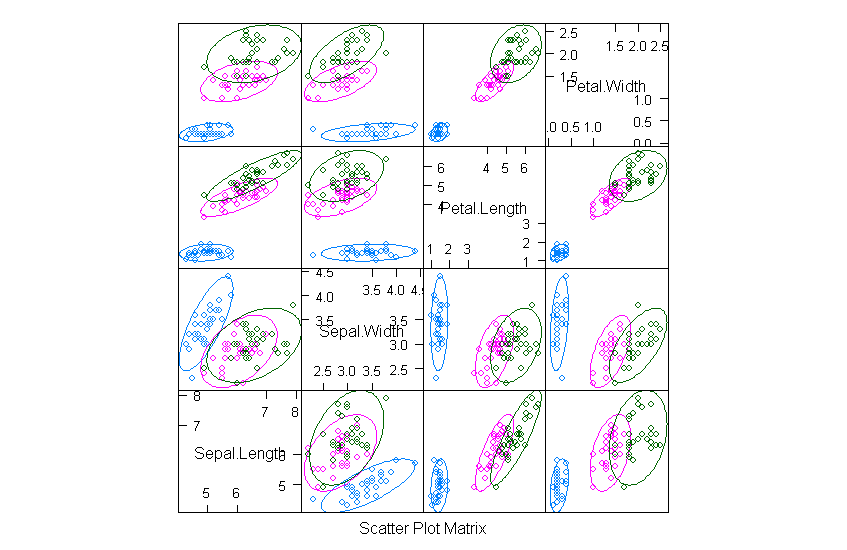
**Actual R Run:**

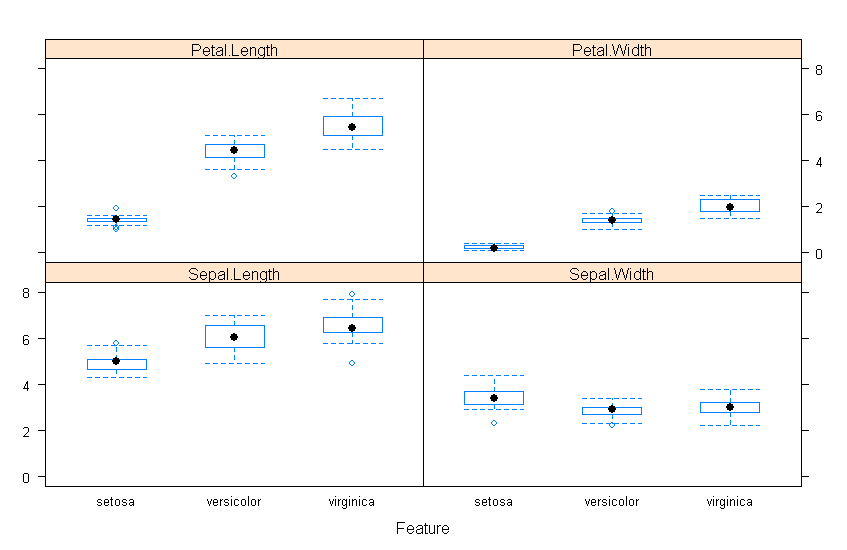
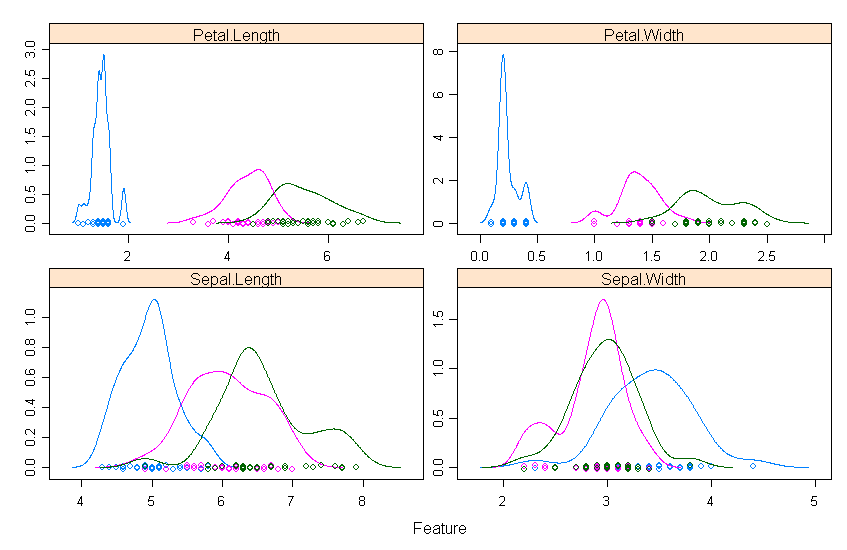
|  |
| --- |
| R version 3.5.1 (2018-07-02) – “Feather Spray”  Copyright © 2018 The R Foundation for Statistical Computing  Platform: x86\_64-w64-mingw32/x64 (64-bit)  R is free software and comes with ABSOLUTELY NO WARRANTY.  You are welcome to redistribute it under certain conditions.  Type ‘license()’ or ‘licence()’ for distribution details.  R is a collaborative project with many contributors.  Type ‘contributors()’ for more information and  ‘citation()’ on how to cite R or R packages in publications.  Type ‘demo()’ for some demos, ‘help()’ for on-line help, or  ‘help.start()’ for an HTML browser interface to help.  Type ‘q()’ to quit R.  [Workspace loaded from ~/.Rdata]  > #############################################################################  > # Iris Flowers Analysis – Hello World Machine Learning  > # From Machine Learning Mastery  > # Modifications by Patrick Hagan  > # Date: September 13, 2018  > #############################################################################  > # Load Libraries  > library(caret)  Loading required package: lattice  Loading required package: ggplot2  > # Load iris dataset into R Studio environment running R  > data(iris)  > # rename the dataset as dataframe  > df <- iris  > # Split out validation dataset create a list of 80% of the  > # rows in the original dataset we can use to train and test  > validation\_index <- createDataPartition(df$Species, p=0.80, list=FALSE)  > # select 20% of the data for validation  > validation <- df[-validation\_index,]  > # use the remaining 80% of data to training and testing the models  > df <- df[validation\_index,]  > validation\_index <- createDataPartition(df$Species, p=0.80, list=FALSE)  > # select 20% of the data for validation  > validation <- df[-validation\_index,]  > # use the remaining 80% of data to training and testing the models  > df <- df[validation\_index,]  > # dataframe dimensions  > dim(df)  [1] 96 5  > # list types for each attribute  > sapply(df, class)  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  “numeric” “numeric” “numeric” “numeric” “factor”  > # display first 5 rows of the data  > head(df)  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  7 4.6 3.4 1.4 0.3 setosa  9 4.4 2.9 1.4 0.2 setosa  > # list the levels for the class  > levels(df$Species)  [1] "setosa" "versicolor" "virginica"  > # split input and output  > x <- df[,1:4]  > y <- df[,5]  > # summarize the class distribution  > percentage <- prop.table(table(df$Species)) \* 100  > cbind(freq=table(df$Species), percentage=percentage)  freq percentage  setosa 32 33.33333  versicolor 32 33.33333  virginica 32 33.33333  > # summarize attribute distributions  > summary(df)  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  Min. :4.300 Min. :2.200 Min. :1.000 Min. :0.100 setosa :32  1st Qu.:5.100 1st Qu.:2.875 1st Qu.:1.500 1st Qu.:0.300 versicolor:32  Median :5.900 Median :3.000 Median :4.450 Median :1.400 virginica :32  Mean :5.868 Mean :3.083 Mean :3.781 Mean :1.206  3rd Qu.:6.500 3rd Qu.:3.325 3rd Qu.:5.100 3rd Qu.:1.800  Max. :7.900 Max. :4.400 Max. :6.700 Max. :2.500  > # boxplots for numeric  > par(mfrow=c(1,4))  > for(i in 1:4) {  + boxplot(x[,i], main=names(df)[i])  + }  > # barplot for class breakdown  > plot(y)  > # scatterplot matrix  > featurePlot(x=x, y=y, plot="ellipse")  > # box and whisker plots for each attribute  > featurePlot(x=x, y=y, plot="box")  > # density plots for each attribute by class value  > scales <- list(x=list(relation="free"), y=list(relation="free"))  > featurePlot(x=x, y=y, plot="density", scales=scales)  > # Run algorithms using 10-fold cross validation  > control <- trainControl(method="cv", number=10)  > metric <- "Accuracy"  > # LDA  > set.seed(7)  > fit.lda <- train(Species~., data=df, method="lda", metric=metric, trControl=control)  > # CART  > set.seed(7)  > fit.cart <- train(Species~., data=df, method="rpart", metric=metric, trControl=control)  > # kNN  > set.seed(7)  > fit.knn <- train(Species~., data=df, method="knn", metric=metric, trControl=control)  > # SVM  > set.seed(7)  > fit.svm <- train(Species~., data=df, method="svmRadial", metric=metric, trControl=control)  > # RANDOM FOREST  > set.seed(7)  > fit.rf <- train(Species~., data=df, method="rf", metric=metric, trControl=control)  > # compare algorithms  > results <- resamples(list(lda=fit.lda, cart=fit.cart, knn=fit.knn, svm=fit.svm, rf=fit.rf))  > summary(results)  Call:  summary.resamples(object = results)  Models: lda, cart, knn, svm, rf  Number of resamples: 10  Accuracy  Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  lda 0.8888889 0.9250000 1.0000000 0.9688889 1.0000000 1 0  cart 0.8888889 0.9000000 0.9545455 0.9486869 1.0000000 1 0  knn 0.8888889 0.9000000 1.0000000 0.9577778 1.0000000 1 0  svm 0.8888889 0.8916667 0.9045455 0.9375758 1.0000000 1 0  rf 0.8888889 0.9000000 0.9000000 0.9286869 0.9772727 1 0  Kappa  Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  lda 0.8333333 0.8880597 1.0000000 0.9532564 1.0000000 1 0  cart 0.8333333 0.8490502 0.9320988 0.9230095 1.0000000 1 0  knn 0.8333333 0.8490502 1.0000000 0.9365898 1.0000000 1 0  svm 0.8333333 0.8376866 0.8574719 0.9065690 1.0000000 1 0  rf 0.8333333 0.8490502 0.8507463 0.8931588 0.9660494 1 0 |
|  |
| |  | | --- | | > | |

**Plots:**

****

****

****

****

**Machine Learning Example with Iris Dataset in Python: (Iris-Python-Example.py)**

**#############################################################################**

**# Iris Flowers Analysis - Hello World Machine Learning**

**# From Machine Learning Mastery**

**# Modifications by Patrick Hagan**

**# Date: September 14, 2018**

**# Anaconda Python: https://www.anaconda.com/download/**

**#############################################################################**

**# Hello World Classification: Iris flowers prediction**

**# Prepare Problem**

**# Load libraries**

**from pandas import read\_csv**

**from pandas.tools.plotting import scatter\_matrix**

**from matplotlib import pyplot**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.model\_selection import KFold**

**from sklearn.model\_selection import cross\_val\_score**

**from sklearn.metrics import classification\_report**

**from sklearn.metrics import confusion\_matrix**

**from sklearn.metrics import accuracy\_score**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.neighbors import KNeighborsClassifier**

**from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis**

**from sklearn.naive\_bayes import GaussianNB**

**from sklearn.svm import SVC**

**# Load dataset**

**# filename = 'iris.data.csv'**

**Filename=** ‘**C:\Users\Students\Downloads\IRIS.csv’**

**names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']**

**dataset = read\_csv(filename, names=names)**

**# Summarize Data**

**# Descriptive statistics**

**# shape**

**print(dataset.shape)**

**# head**

**print(dataset.head(20))**

**# descriptions**

**print(dataset.describe())**

**# class distribution**

**print(dataset.groupby('class').size())**

**# Data visualizations**

**# box and whisker plots**

**dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)**

**pyplot.show()**

**# histograms**

**dataset.hist()**

**pyplot.show()**

**# scatter plot matrix**

**scatter\_matrix(dataset)**

**pyplot.show()**

**# Prepare Data**

**# Split-out validation dataset**

**array = dataset.values**

**X = array[:,0:4]**

**Y = array[:,4]**

**validation\_size = 0.20**

**seed = 7**

**X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, Y, test\_size=validation\_size, random\_state=seed)**

**# Spot-Check Algorithms**

**models = []**

**models.append(('LR', LogisticRegression()))**

**models.append(('LDA', LinearDiscriminantAnalysis()))**

**models.append(('KNN', KNeighborsClassifier()))**

**models.append(('CART', DecisionTreeClassifier()))**

**models.append(('NB', GaussianNB()))**

**models.append(('SVM', SVC()))**

**# evaluate each model in turn**

**results = []**

**names = []**

**for name, model in models:**

**kfold = KFold(n\_splits=10, random\_state=seed)**

**cv\_results = cross\_val\_score(model, X\_train, Y\_train, cv=kfold, scoring='accuracy')**

**results.append(cv\_results)**

**names.append(name)**

**msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())**

**print(msg)**

**# Compare Algorithms**

**fig = pyplot.figure()**

**fig.suptitle('Algorithm Comparison')**

**ax = fig.add\_subplot(111)**

**pyplot.boxplot(results)**

**ax.set\_xticklabels(names)**

**pyplot.show()**

**# Make predictions on validation dataset**

**knn = KNeighborsClassifier()**

**knn.fit(X\_train, Y\_train)**

**predictions = knn.predict(X\_validation)**

**print(accuracy\_score(Y\_validation, predictions))**

**print(confusion\_matrix(Y\_validation, predictions))**

**print(classification\_report(Y\_validation, predictions))**

**Machine Learning Example with Iris Dataset in Go: (Iris\_GO\_Example.go)**

**#############################################################################**

**# Iris Flowers Analysis - Hello World Machine Learning**

**# From: Machine Learning with GO from Packt Publishing**

**# Modifications by Patrick Hagan**

**# Date: September 18, 2018**

**#############################################################################**

// Name: Iris\_GO\_Example.go

// Date: September 18, 2018

// Original Source: Machine Learning with Go

// Description: Combined various small GO programs into this one

// to analyze the Hello World Iris dataset. All errors are mine.

// Modified by: Patrick Hagan

//---------------------------------------------------------------------

import (

    "encoding/csv"

    "fmt"

    "io"

    "log"

    "os"

    "strconv"

    "github.com/gonum/floats"

    "github.com/gonum/stat"

    "github.com/kniren/gota/dataframe"

    "github.com/montanaflynn/stats"

    "github.com/gonum/plot"

    "github.com/gonum/plot/plotter"

    "github.com/gonum/plot/vg"

)

func main() {

// Open the IRIS CVS file

irisFile, err := os.Open("../data/iris.csv")

if err != nil {

log.Fatal(err)

}

defer irisFile.Close()

// Create a dataframe from the CVS file.

// The types of the columns will be inferred.

irisDF := dataframe.ReadCSV(irisFile)

// As a sanity check, display the records to stdout.

// Gota will format the dataframe for pretty printing.

fmt.Printf(irisDF)

// Create a filter for the dataframe.

filter := dataframe.F{

    Colname: "species"

    Comparator: "=="

    Comparando: "Iris-versicolor"

}

// Filter the dataframe to see only the rows where

// the iris species is "Iris-versicolor".

versicolorDF := irisDF.Filter(filter)

if versicolorDF.Err != nil {

    log.Fatal(versicolorDF.Err)

}

// Filter the dataframe again, but only select out the

// sepal\_width and species columns

versicolorDF = irisDF.Filter(filter).

Select([]string{"sepal\_width", "species"})

// Filter the dataframe again, but only display

// the first three results

versicolorDF = irisDF.Filter(filter).

Select([]string{"sepal\_width", "species"}).

Subset([]int{0, 1, 2})

// Get the float values from the "sepal\_length" column

// because we want the measures for this variable

sepal\_Length := irisDF.Col("sepal\_length").Float()

// Calculate the Mean of the variable

meanVal := stat.Mean(sepal\_Length, nil)

// Calculate the Mode of the variable

modeVal, modeCount := stat.Mode(sepal\_Length, nil)

// Calculate the Median of the variable

medianVal, err := stat.Median(sepal\_Length)

if err != nil {

    log.Fatal(err)

}

// Calculate the Min of the variable

minVal := floats.Min(sepal\_Length)

// Calculate the Max of the variable

maxVal := floats.Max(sepal\_Length)

// Calculate the Range of the variable

rangeVal := maxVal - minVal

// Calculate the Variance of the variable

varianceVal := stat.Variance(sepal\_Length, nil)

// Calculate the Standard Deviation of the variable

stdDevVal := stat.StdDev(sepal\_Length, nil)

// Sort the values

floats.Argsort(sepal\_Length, inds)

// Get the Quantiles

quant25 := stat.Quantile(0.25, stat.Empirical, sepal\_Length, nil)

quant50 := stat.Quantile(0.50, stat.Empirical, sepal\_Length, nil)

quant75 := stat.Quantile(0.75, stat.Empirical, sepal\_Length, nil)

// Output the results to standard out

fmt.Printf("\n Sepal Length Summary Statistics: \n")

fmt.Printf(" Mean Value: %0.2f\n", meanVal)

fmt.Printf(" Mode Value: %0.2f\n", modeVal)

fmt.Printf(" Mode Count: %0.2f\n", modeCount)

fmt.Printf(" Median Value: %0.2f\n\n", medianVal)

fmt.Printf(" Max Value: %0.2f\n\n", maxVal)

fmt.Printf(" Min Value: %0.2f\n\n", minVal)

fmt.Printf(" Range Value: %0.2f\n\n", rangeVal)

fmt.Printf("Variance Value: %0.2f\n\n", varianceVal)

fmt.Printf(" Std Dev Value: %0.2f\n\n", stdDevVal)

fmt.Printf(" 25 Quantile: %0.2f\n\n", quant25)

fmt.Printf(" 50 Quantile: %0.2f\n\n", quant50)

fmt.Printf(" 75 Quantile: %0.2f\n\n", quant75)

// Create a histogram for each of the feature columns in the dataset

for \_, colName := range irisDF.Names() {

    // If the column is one of the feature columns, create histogram

    if colName != "species" {

        // Create a plotter.Values value and fill it with the

        // values from the respective column of the dataframe

        v := make(plotter.Values, irisDF.Col(colName).Float())

        for i, floatVal := range irisDF.Col(colName).Float() {

            v[i] = floatVal

        }

        // Make a plot and set its title

        p, err := plot.New()

        if err != nil {

            log.Fatal(err)

        }

        p.Title.Text = fmt.Sprintf("Histogram of a %s", colName)

        // Create a histogram of our values drawn

        // from the standard normal.

        h, err := plotter.NewHist(v, 16)

        if err != nil {

            log.Fatal(err)

        }

        // Normalize the histogram

        h.normalize(1)

        // Add the histogram to the plot

        p.Add(h)

        // Save the plot to a PNG file

        if err := p.Save(4\*vg.Inch, 4\*vg.Inch, colName + "\_hist.png"); err != nil {log.Fatal(err)}

        // Make a second plot and set its title and axis label

        p2, err := plot.New()

        if err != nil {

            log.Fatal(err)

        }

        p2.Title.Text = "Box Plots"

        p2.Y.Label.Text = "Values"

        // Create the box for our data

        w := vg.Points(50)

        // Create a box plot for each of the feature columns in the dataset

for idx, colName := range irisDF.Names() {

            // If the column is one of the feature columns, create box plot

            if colName != "species" {

                // Create a plotter.Values value and fill it with the

         // values from the respective column of the dataframe

         v2 := make(plotter.Values, irisDF.Nrow())

         for i, floatVal := range irisDF.Col(colName).Float() {

             v2[i] = floatVal

                }

                // Add the data to the plot

                b, err := plotter.NewBoxPlot(w, float64(idx), v2)

                if err != nil {

                    log.Fatal(err)

                }

                // Add the histogram to the plot

         p2.Add(h)

                // Set the x axis of the plot to nominal width

                // the given names for x=0, x=1, etc.

p2.NominalX("sepal\_length", "sepal\_width", "petal\_length", "petal\_width")

        // Save the plot to a PNG file

                if err := p2.Save(4\*vg.Inch, 4\*vg.Inch, colName + "\_boxplots.png");

                 err != nil {log.Fatal(err)

                }

            }

        }

    }

}

}