Business Analytics with Power BI

Module 3 – Predictive Analytics with

Power BI and R

Student Lab Manual – Lab 2 – Creating Machine

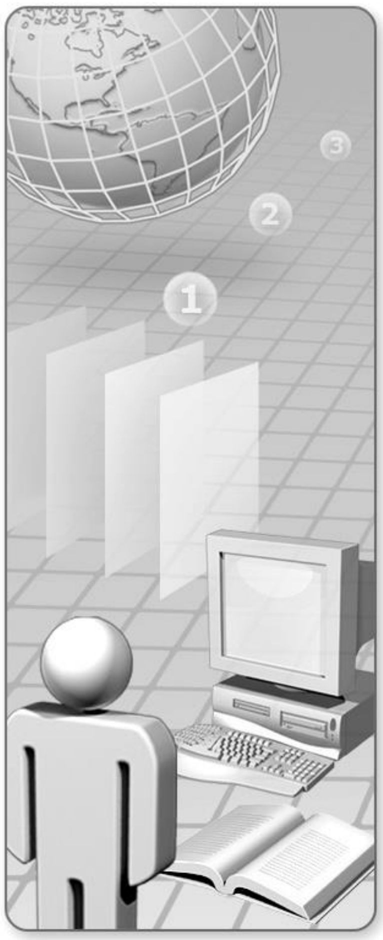
Learning Models by Using R

Version 1.0

Module 3

Predictive Analytics with Power BI and R

Lab 2 – Creating Machine Learning Models by Using R



Lab 2: Creating Machine Learning Models by Using R

Introduction

In this lab, you will create two machine learning models by using R language. One model will be to predict gas consumption of a car (regression model). The other model will be to classify flowers in three different species using linear discriminant analysis (LDA) and random forests (classification models).

Objectives

After completing this lab, you will be able to:

- Create machine learning models by using R language.

Estimated time to complete this lab

45 minutes (depends on experience)

Resources

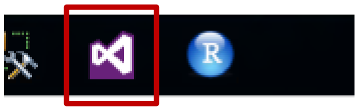
Virtual machine (VM) Name	**Business Analytics with Power BI - Module 1**
Domain	**POWERBI-WIN10**
User	**POWERBI-WIN10\LabUser**
Password	**P@ssw0rd1!**
Lab Files	**E:\Labs\**
Asset Files	**E:\Assets\**

Exercise 1: Creating Machine Learning Models in R

Creating a Regression Model to Predict Consumption

In this task, you will create a regression model to predict oil consumption based on cars stored in the **mtcars** data frame.

1. If Microsoft Visual Studio is not open, open it. On the taskbar, click the **Visual Studio 2015** shortcut.



2. Click the **File** tab, select **New**, and then select **Project**.

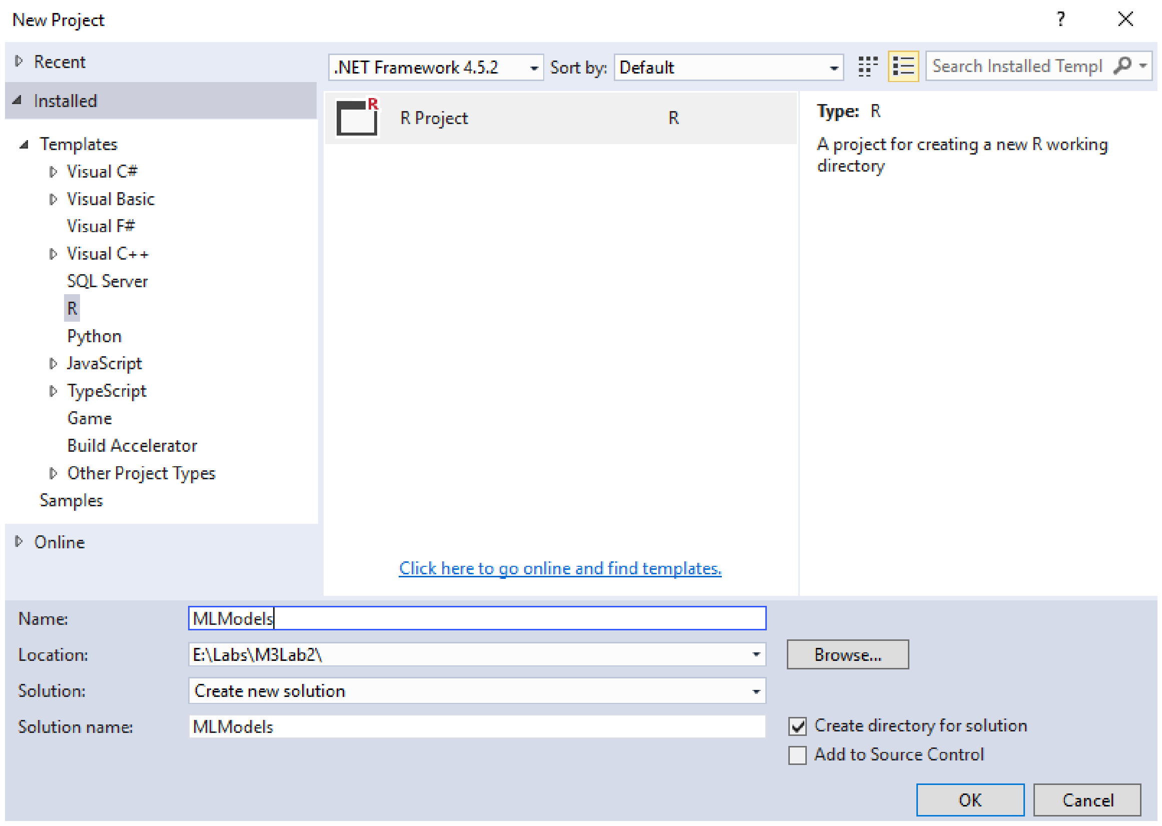
3. In the **New Project** window, click on the left side, **Templates** >> **R**.

4. Select **R Project**.

5. In the **Name** box, type **MLModels**, and in the **Location** box, browse to **E:\Labs\M3Lab2\**.

6. In the **Solution name** box, type **MLModels**.

7. Leave the **Create directory for solution** check box selected. You should see the following screen:


New Project

- Recent
- Installed
  - Templates
    - Visual C#
    - Visual Basic
    - Visual F#
    - Visual C++
    - SQL Server
    - R
    - Python
    - JavaScript
    - TypeScript
    - Game
    - Build Accelerator
    - Other Project Types
  - Samples
- Online

.NET Framework 4.5.2 Sort by: Default

R Project R

Type: R

A project for creating a new R working directory

Click here to go online and find templates.

Name: MLModels

Location: E:\Labs\M3Lab2\

Solution: Create new solution

Solution name: MLModels

Create directory for solution [✓]

Add to Source Control [ ]

8. Click **OK**. Make sure you save your progress as you progress through the lab.

9. In the **Solution Explorer** pane, rename **script.R** to **Consumption.R**.

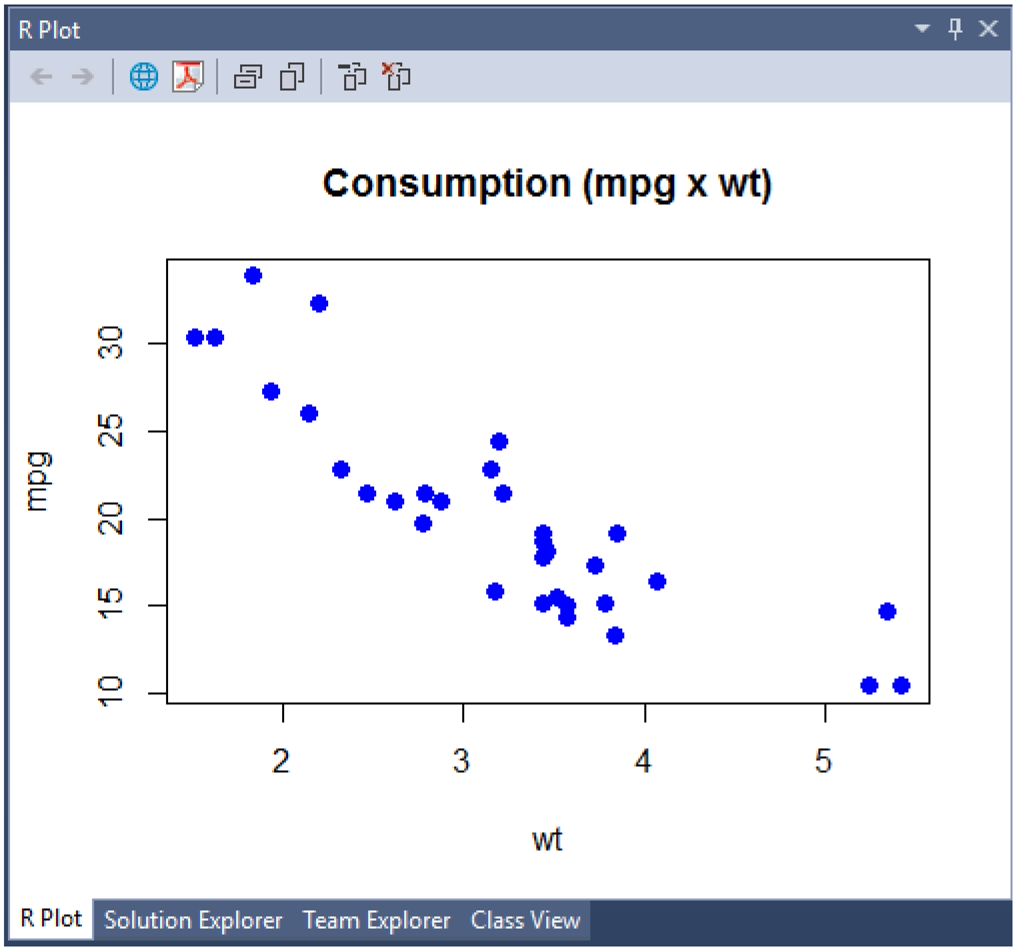
10. Add the following code to the **Consumption.R** script, and then run the code. Notice that these functions help you to understand your data.

```
tail(mtcars)
str(mtcars)
summary(mtcars)
```

11. Add the following lines and run it. Notice that there is a correlation between weight (wt) and consumption (mpg).

```
mpg <- mtcars$mpg
wt <- mtcars$wt
plot(wt, mpg, pch = 16, cex = 1.3, col = “blue”, main =
“Consumption (mpg x wt)”, xlab = “wt”, ylab = “mpg”)
```

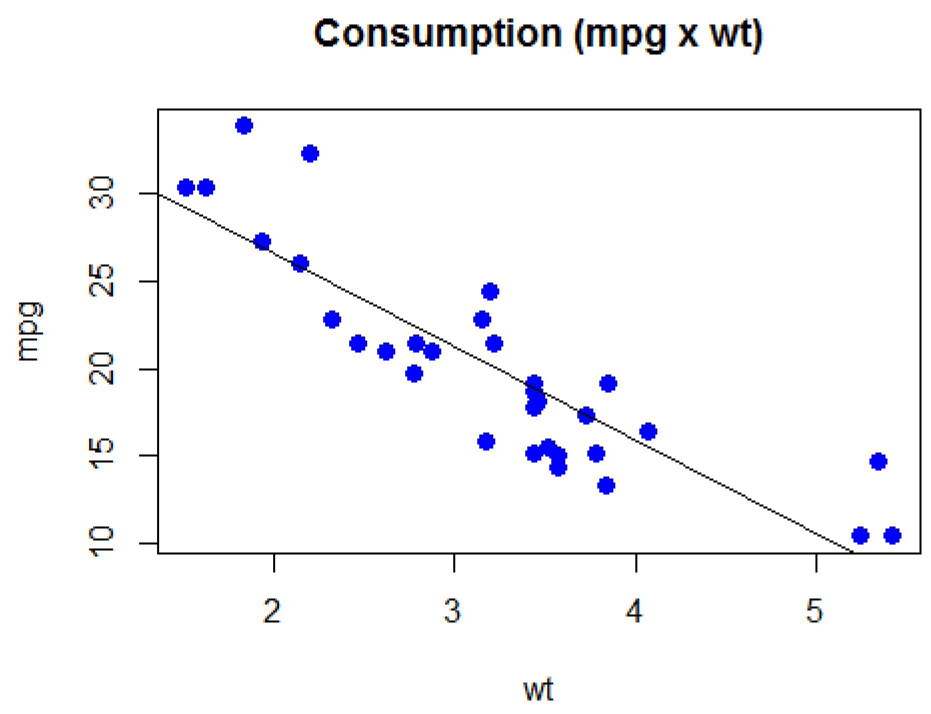
12. You should have the following plot in your **R Plot pane**:



13. Add the following code to the same file, and then run it. This code will create a linear model in which **mpg** is calculated based on **wt** only.

```
m <- lm(mpg ~ wt)
m
```

14. Use the calculated coefficients to plot a line over the last plot you made. Add the following line and then run it. Your plot should look like the following screenshot:



15. Test the model you have created. Add the following code to your **Consumption.R** script file and run it.

```
s <- mtcars[1:5,] #get the first 5 lines
data.frame(actual = s$mpg, predict = predict(m,s))
```

16. Your output (**R Interactive** pane) should have similar information. Notice that the predicted values are not exactly the same as the actual ones. This is due to the generalization that is needed in the model. We need to avoid overfitting. Generic models usually work better with real world examples.

```
                  actual  predict
Mazda RX4           21.0 23.28261
Mazda RX4 Wag       21.0 21.91977
Datsun 710          22.8 24.88595
Hornet 4 Drive      21.4 20.10265
Hornet Sportabout   18.7 18.90014
```

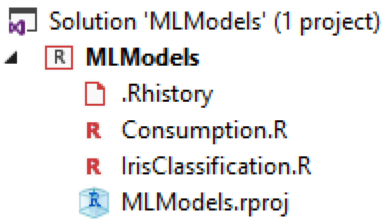
17. Save your script file.

Creating a Classification Model to Predict a Class of the Iris Plant

In this task, you will create a classification model that will classify one of the three different classes of the Iris plant: Iris Setosa, Iris Versicolour, and Iris Virginica. We will use two different algorithms: Linear Discriminant Analysis (LDA) and Random Forests. Remember that there are several other algorithms that would be helpful in this task as well.

1. If Visual Studio is not open, open it. On the taskbar, click the **Visual Studio 2015** shortcut. Open the project created in the last task (**MLModels**).

2. Add a new script file (right-click the project's name, and then click **Add R Script**). Rename the file name to **IrisClassification.R**. You should have something similar to the following screenshot:



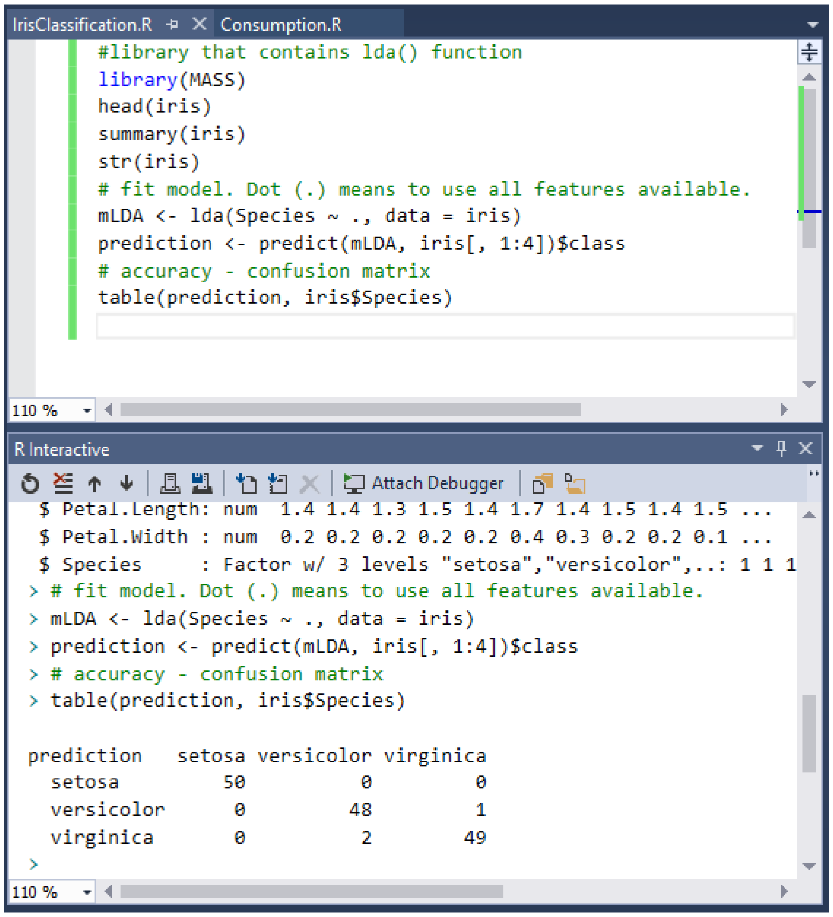
3. Add the following code to your **IrisClassification.R** script. You do not need to copy the comments. Run all the code.

```
#library that contains lda() function
library(MASS)
head(iris)
summary(iris)
str(iris)
# fit model. Dot (.) means to use all features available in iris
# dataframe.
mLDA <- lda(Species ~ ., data = iris)
prediction <- predict(mLDA, iris[, 1:4])$class
# accuracy - confusion matrix
table(prediction, iris$Species)
```

Note: We usually create at least two datasets from our data: one to train the model and the other to validate it. This example was simplified for the purposes of this lab. We will see a more complex example in the last lab of this module.

4. You should have the following result. This shows a very good classification, which is not necessarily true, because we used the same dataset to fit the model and to validate it.

In real-world cases, you should always validate a model with elements that were not used to fit/train your model.



```
#library that contains lda() function
library(MASS)
head(iris)
summary(iris)
str(iris)
# fit model. Dot (.) means to use all features available.
mLDA <- lda(Species ~ ., data = iris)
prediction <- predict(mLDA, iris[, 1:4])$class
# accuracy - confusion matrix
table(prediction, iris$Species)
```

```
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1
> # fit model. Dot (.) means to use all features available.
> mLDA <- lda(Species ~ ., data = iris)
> prediction <- predict(mLDA, iris[, 1:4])$class
> # accuracy - confusion matrix
> table(prediction, iris$Species)

prediction   setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         48         1
  virginica       0          2        49
>
```

5. Add the following code to your script and then run it. This code simulates a new case to test against our model:

```
new_case <- data.frame(Sepal.Length = 5.1, Sepal.Width = 3.5,
Petal.Length = 1.4, Petal.Width = 0.2, Specie = "setosa")
new_case
data.frame(actual = new_case$Specie,
   predict = predict(mLDA, new_case[, 1:4])$class)
```

6. Now that you already have a model using LDA, you will create another one using random forests. It is part of a data scientist's job to determine which algorithm and parameters are good enough for the problem.
   Add the following code to your script (**IrisClassification.R**), and then run it. Notice that for this algorithm, we will create two datasets (train and test) to validate our model.

```
install.packages("randomForest")
library(randomForest)

#split in two groups (~70% train | ~30% test)
i <- sample(2,nrow(iris),replace=TRUE,prob=c(0.7,0.3))
train <- iris[i==1,]
test <- iris[i==2,]

#create model
rf <-
randomForest(Species~.,data=train,ntree=100,proximity=TRUE)
print(rf)

#test model against test data
rf.pred <-predict(rf,newdata=test)
table(rf.pred, test$Species)
```

7. See the results created by your script. Save your project and close Visual Studio.