Machine Learning in R

Copyright © 2022 Excel Consulting Solutions Pty Ltd. T/A Nexacu
Published by Nexacu

Brisbane | Melbourne | Sydney | Adelaide | Perth | Canberra | Parramatta
Australia
Phone: (+61) 1300 886 190
Web: www.nexacu.com.au
Email: info@nexacu.com.au

Product Code: MLR v 1.4

Build: 4/05/2022

Trademark Acknowledgments

Disclaimer

Copyright Notice

Contents

Trademark Acknowledgments i
Disclaimer i
Copyright Notice i
1. Introduction to Machine Learning 4
What is machine learning? 4
Prediction when the underlying relationship is known 4
Prediction when the underlying relationship is unknown 5
Supervised vs unsupervised techniques 6
Preparing data for analysis 6
Structured vs unstructured data 6
Inaccurate data 7
Missing values 7
Data transformation 7
2. Cluster analysis 8
Applications of Cluster Analysis 8
K-means 8
How the algorithm works 8
Data preparation and feature engineering 9
Variable selection 9
Data distribution and transformation 10
Scaling the data 10
How many clusters? 11
Elbow method 11
Silhouette method 13
The gap statistic 14
Performing k-means clustering in R 15
Customer segmentation analysis 17
RFM analysis 18
Examining the distributions 19
Scale the data 20
How many clusters? 20
Customer segmentation 21
Other clustering algorithms 22
3. Random forests 23
Basics of tree-based models 23
Strengths of decision trees 25
Limitations of decision trees 25
The bias-variance trade-off 25
From trees to (random) forests 26

Ensemble learning: bagging to reduce overfitting and improve predictive accuracy 26
How do we fit multiple models to the same dataset? 26
Bagging 27
The process of supervised machine learning 27
Feature engineering 27
Splitting data into training and test sets 28
Strategies for data splitting 28
Training the model 28
Improving the model 28
Evaluating the final model 29
Classification models 29
Regression models 30
The process of creating a random forest model 30
Random forests in R 31
Classification tree and random forest classification model 31
Regression tree and random forest regression model 35
Improving the model 38
4. R scripts in Power BI 39
Why bring a machine learning model into Power BI? 39
Setting up Power BI to use R scripts 39
Cluster analysis in Power BI 39
Bike-share forecasting in Power BI 43
5. Where to next? 45

1. Introduction to Machine Learning

What is machine learning?

Machine learning refers to a group of analysis techniques that can be used to extract knowledge from data. This usually involves building a model that can then be used to make predictions. Today we will also use a technique used to find patterns within data to group observations.

They are collectively referred to as machine-learning techniques because they use algorithms (step-by-step instructions written in a programming language) to "learn" from data. We use machine learning when we don't know the underlying relationship.

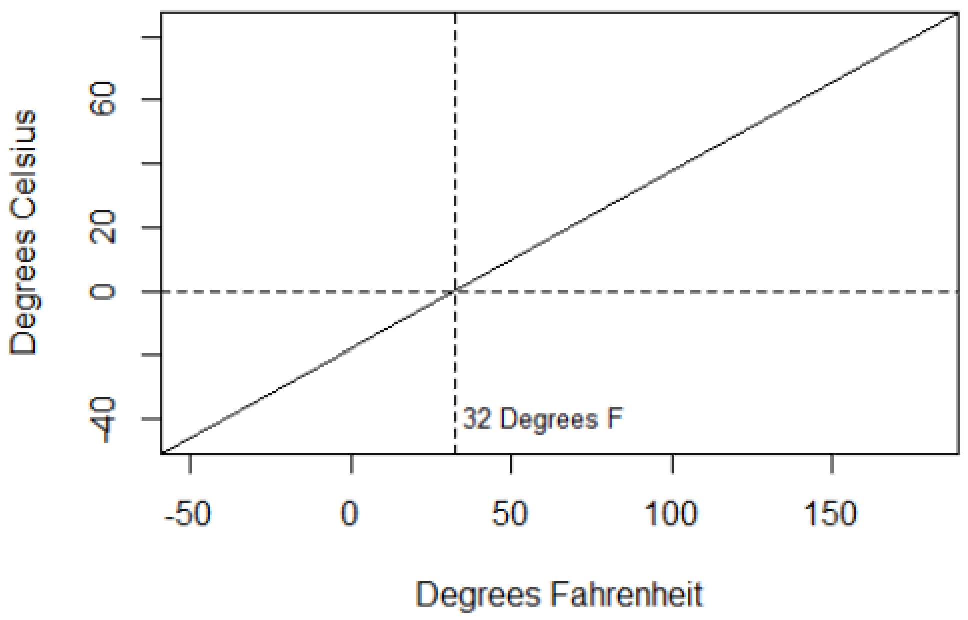
Prediction when the underlying relationship is known

We don't need to use machine learning when we know the underlying relationship. For example, to convert a temperature measured in degrees Fahrenheit to degrees Celscius, we can use the following equation:

(degrees Fahrenheit – 32) * 5/9 = degrees Celsius

This is because the relationship between these two units of measurement is known. It can also be represented graphically (Figure 1). The line is defined by its slope (gradient) and intercept (where it cuts the Y axis).

Figure 1. The relationship between degrees Celsius and degrees Fahrenheit



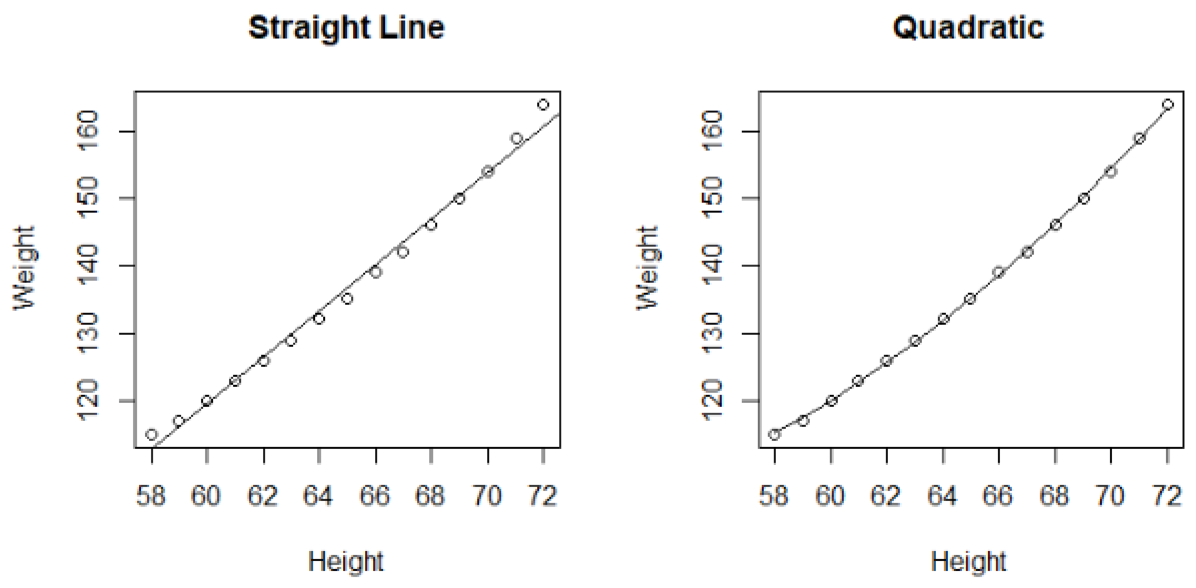
However, we don't always know if or how two values or variables are related. This is where machine learning can be helpful.

Prediction when the underlying relationship is unknown

If we plot the heights and weights in the *women* dataset that is distributed with R, we can see visually that there appears to be some relationship between weight and height. By fitting a curve to this data, we can create a predictive model.

In the world of machine learning, fitting the model is called *training* the model. The training process determines the model *parameters*, in this case the model coefficients. We can try a straight-line model first and then a quadratic model (Figure 2). Which fits the data better?

Figure 2. Predicting weight from height



In this case, the model we choose influences the general shape of the relationship uncovered but the algorithm *learns* from the data what the model coefficients should be.

```
Call:
lm(formula = weight ~ poly(height, 2), data = women)

Coefficients:
     (Intercept)  poly(height, 2)1  poly(height, 2)2
         136.733            57.730             5.335
```

We can then use the final model to make weight predictions when we only have height measurements.

In reality, the machine-learning process involves more than simply fitting a model. It starts with data collection and preparation which can take up to 80% of analysis time. Creating a model is usually an iterative process of training, testing and fine-tuning until the model produces sufficiently accurate predictions.

Today, we will go through the full machine-learning process using two very different machine learning techniques. We will perform cluster analysis using a k-means algorithm and we will create predictive models using a random forest algorithm.

Cluster analysis is used to identify natural groups or clusters within data. Random forests are used to create predictive models. They can be used to predict either numeric values or categories.

Supervised vs unsupervised techniques

An unsupervised technique is one in which we don't know the "correct" answer. Instead, we might use an algorithm to detect patterns within the data to reveal more about its structure. Cluster analysis is an unsupervised technique and we use it when we have unlabelled data. That is, we don't know what groupings exist in the data and which group each data point should belong to. For example, if I take a photo, the image is unlabelled. You can look at the photo and provide a label – e.g. is the photo of a person, a dog or a cat? If you have to do this for thousands of images, you may want to use machine learning to automate the process.

Supervised techniques are used when we know the actual values and we can compare the model predictions with these actual values. The random forest algorithm is a supervised technique. A subset of the data is used to train the algorithm. The remaining data is then used to test the accuracy of the predictions made by comparing actual measured values with predictions made by the model. For example, perhaps I want to predict ice-cream sales based on weather. If I have historical weather data and my ice-cream sales on those days, I may be able to create a predictive model. I can then use weather forecasts to predict sales. Predicting the number of ice-creams sold will create a regression model; whereas predicting "low", "average" or "high" sales will create a classification model.

Preparing data for analysis

Before any kind of analysis can be performed, data must be collected and processed. Pre-processing of raw data is necessary because machine learning algorithms expect data to be presented in a specific way. Some common data pre-processing operations are discussed below.

Structured vs unstructured data

Many machine learning algorithms work with data provided in a structured format. Structured data are organised into rows and columns, where each row represents a single observation and each column (field) represents a variable of interest.

Unstructured data are essentially all other data formats, including images, audio files, tweets and weather sensor data.

Inaccurate data

Data should be removed only if they can confidently be expected to be truly inaccurate, e.g. a negative weight value or a human age of 500. They should not be removed if they are simply unexpectely high or low. They may simply be *outliers*, rare but correct values, rather than incorrect figures.

Missing values

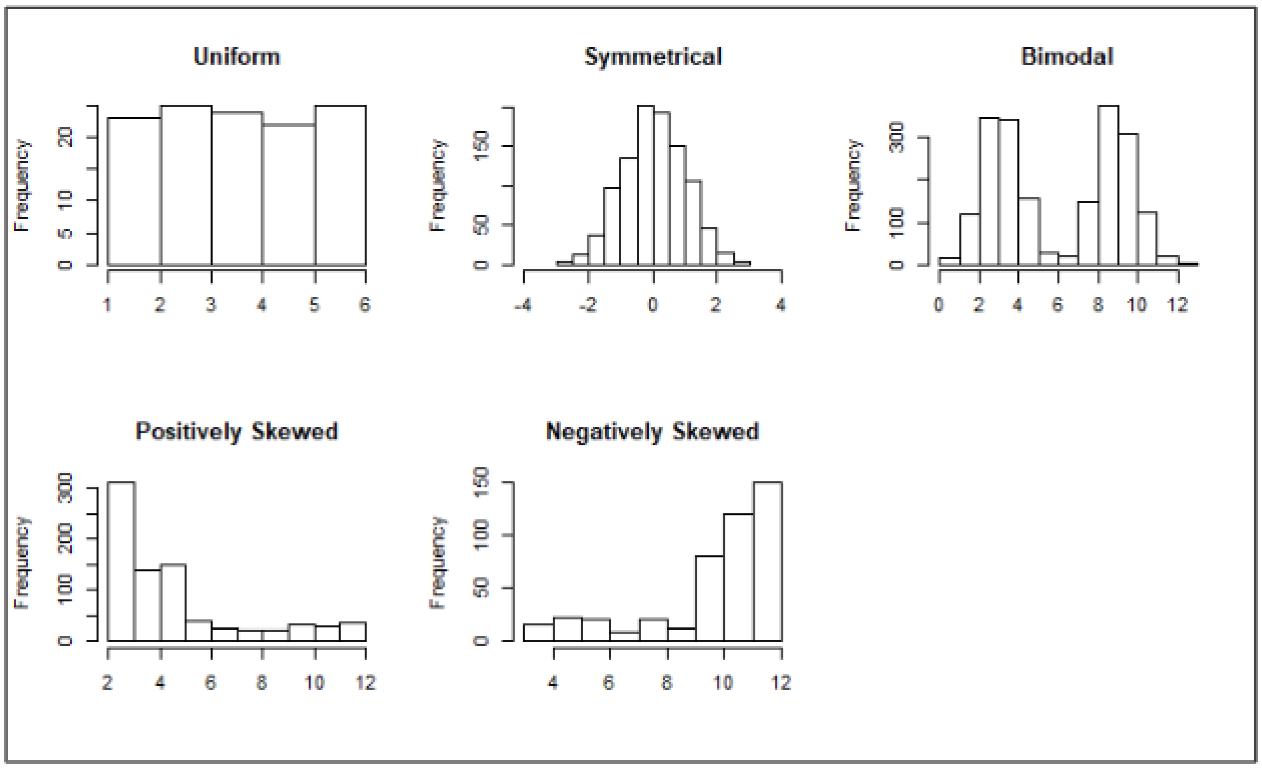
There are multiple ways to deal with missing values. You may choose to delete the entire observation but this will reduce the size of your dataset. Other approaches are to replace the value with the mean or to predict the missing values.

Data transformation

Some machine learning algorithms are sensitive to skewed data (Figure 3) or to the presence of outliers in the data. Outliers are extremely high or low values which are found outside the range within which most of the data lies.

Transforming data can reduce skew and reduce the impact of outliers.

Figure 3. Common data distributions



2. Cluster analysis

Cluster analysis is an unsupervised, exploratory data analysis technique. The technique is used to find natural groupings in the data, based on variable values. We would normally use it when we have data that are unlabelled. That is, data that have not been assigned to a group or category. The aim of cluster analysis is to determine which group each data point belongs to.

Applications of Cluster Analysis

Cluster analysis has many possible applications in diverse fields, including:

- Customer segmentation
- Detection of fraudulent credit card transactions
- Image processing
- Dietary analysis
- Performance evaluation of employees
- Classification of galaxies in astronomy
- Identifying associated species in biology
- Cancer diagnosis and prediction

We will perform customer segmentation analysis today. This is a process of dividing customers into groups based on their purchasing patterns or demographic information for the purpose of improving marketing to each group.

K-means

We will use the k-means algorithm. It is one of the most well known and widely-used clustering algorithms. Once you understand the basics of cluster analysis, there are many variants of the k-means algorithm to explore.

How the algorithm works

1. The user specifies how many clusters the algorithm should partition the data into.
2. Cluster centres (average values for each variable used in the analysis) can be provided, alternatively random observations (rows of data) are selected as the initial centres.
3. Randomly assign each data point to a cluster.
4. Calculate the new centroids (cluster centres).
5. Calculate the difference from each observation to the centroids and assign it to the closest cluster.
6. Calculate the new centroid values based on the cluster values.
7. Continue until all the clusters are stable.

This process minimises the within-cluster sum of squares.

For 3 variables, x, y and z, the withing-cluster sum of squares can be calculated by finding the sum of all of the Euclidean (straight-line) distances from each data point to its cluster centroid.

$$sum(\sqrt{(x-\bar{x})^2+(y-\bar{y})^2+(z-\bar{z})^2})$$

Data preparation and feature engineering

Data should be in a matrix (or data frame) with each variable in its own column. Each row should represent one observation (measurement).

Feature engineering is the term used to describe the transformation of raw data into the data that will be used as the inputs to the machine learning model. Sometimes we can use the raw data. Other times, we may need to transform it or derive new quantities, such as ratios.

Variable selection

Before performing cluster analysis we need to decide which variables to use. This decision will be influenced by the data available and any prior knowledge about which variables might be important in defining the groupings of interest. For example, when segmenting customers, it might be useful to separate between customers that spend small vs large amounts, new vs long-time customers and frequent vs infrequent customers.

Variables used to create k-means clusters must be numeric data, that can be appropriately summarised by their mean. Ideally, they should be continuous numeric data although in practice, it is often used with discrete numeric data, such as counts.

If your data are not numeric, you may be able to derive numeric values from the raw data to use when clustering.

We will illustrate the steps involved in performing cluster analysis using the *iris* dataset (distributed with the base R installation). In this example, we know the species or group each observation belongs to and we will be able to compare the k-means result to the actual groups. Normally, cluster analysis would be used if we did not know which species (group) each belonged to.

Ex 2.1 - Clustering on the iris dataset

1. Open the *iris* dataset in the data viewer.
2. Note that the first 4 columns contain numeric data. The 5th column contains the species name. In this initial exercise, we will use cluster analysis to see whether the algorithm can distinguish the different iris species based on their sepal and petal measurements.

Data distribution and transformation

Before performing k-means analysis, we need to examine the distributions of each variable that will be used to define the clusters. We can do this using histograms or boxplots and we can calculate skewness using the `DescTools::Skew()` function.

Ideally, the variables used by the k-means algorithm should not be skewed – they should not display long tails of unusually large or small values. Skewed data will disproportionately impact the centroid means and the within-cluster sum of squares.

Symmetric data have a skewness near zero:

- skewness < -1 or > 1 => highly skewed
- skewness between -0.5 and -1 or between 0.5 and 1 => moderately skewed
- skewness between -0.5 and 0.5 => approximately symmetric

Transformations can be used to reduce skewness.

Positively skewed data are commonly transformed with a log, square (or other) roots or reciprocal transformations.

Negatively skewed data are commonly transformed using squares, cubes or higher power transformations.

Ex 2.2 - Check for skewness in the predictor variables

1. Create histograms of each variable to be used in the cluster analysis.

```
par(mfrow=c(2,2))
hist(iris$Sepal.Length)
```

2. Use the Skew function from DescTools package to calculate the skewness of each variable.

```
library(DescTools)
lapply(iris[,-5], Skew)
```

3. Are the data skewed?
4. Do we need to transform the data?

Scaling the data

The k-means algorithm assumes that the variables used for clustering have similar means and variance.

Each variable to be used by the k-means algorithm should be scaled so that they all have the same magnitude. A variable that contains larger values than other variables will end up having a greater influence on the within-cluster variance and will therefore have a greater influence on selecting the appropriate cluster for the data point.

By scaling the data to the same range, they will have equal weight in determining allocation to clusters.

In practice, differences in magnitude are sometimes used to provide weightings on variables used in cluster allocation. This is generally based on domain knowledge that certain variables are more important than others. Today, we will assume all variables have equivalent importance.

The `scale()` function can be used to subtract the mean from each variable and divide it by its standard deviation. Resulting values will have a mean of 0 and a standard deviation / variance of 1.

Ex 2.3 - Scale the *iris* data

1. Use the `scale()` function to scale the 4 numeric variables in the *iris* dataset to have a mean of 0 and variance of 1. Save as a new data frame, *iris_c*.

```
iris_c <- data.frame(scale(iris[, -5]))
```

2. Check the mean and variance of each column in *iris_c*.

How many clusters?

Deciding the optimal number of clusters for grouping data is a crucial part of the analysis. However, it can also be the most difficult step, particularly if we have no prior information on which to base the decision.

However, there are a few graphical methods that can be used to assist in the selection of the number of clusters.

Elbow method

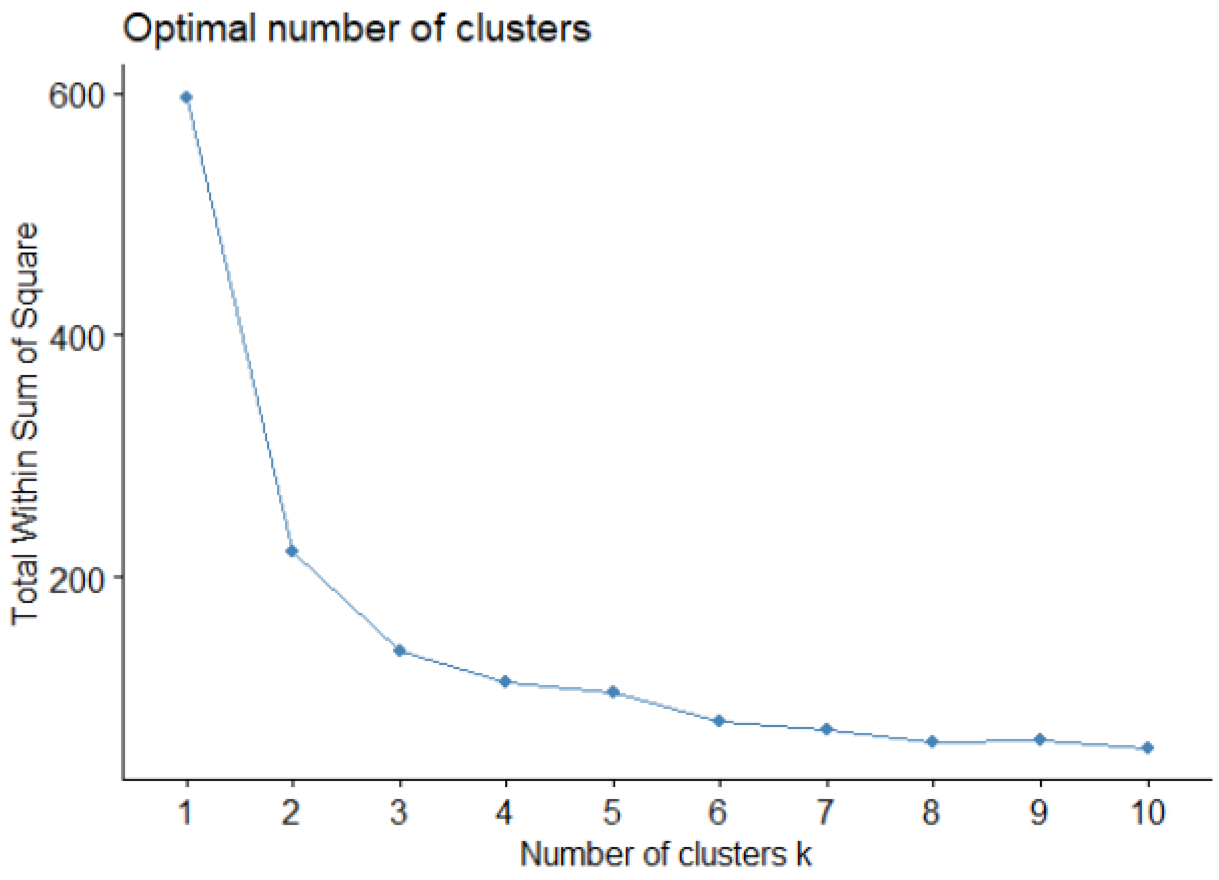
The elbow method is the most commonly used technique for selecting the optimal number of clusters.

When the k-means algorithm is run, the total within-cluster sum of squares (WSS) is calculated for each cluster. The total within-cluster sum of squares is the sum of each of these. Smaller values are desirable. They indicate that the clusters are more tightly grouped (less spread out).

By running the algorithm multiple times, for different numbers of clusters, we can compute the total within-cluster sum of squares (total WSS) for each. The results can then be plotted as total WSS vs number of clusters.

As the number of clusters increases, the total WSS decreases. The “elbow” refers to the point where the line chart bends, where increasing the number of clusters leads to only small decreases in the WSS (Figure 4).

Figure 4. Elbow graph for determining optimal number of clusters



The chart above shows a big decline in total WSS as the number of clusters increases from one to two. Much smaller declines in total WSS are evident after 3 clusters. From the above, the optimal number of clusters could be 2 or 3.

While this method can be helpful in determining optimal number of clusters, it remains somewhat subjective because the elbow location is not always unambiguous.

Ex 2.4 - Create an elbow chart for iris dataset clusters

1. We will use the `kmeans()` function from the base R packages. First we will use the `kmeans()` function to separate the data into 2 clusters.

```
iris_k2 <- kmeans(iris_c,2)
```

2. Print the contents of *iris_k2*. You will see a summary which includes:

- cluster means (centres)
- clustering vector – vector of values that indicate which cluster each row has been allocated to
- within cluster sum of squares
- a list of components available by name.

3. Print the value of the total within-cluster sum of squares.

```
iris_k2$tot.withinss
```

4. Repeat the above, this time separating the data into 3 clusters.

```
iris_k3 <- kmeans(iris_c,3)
iris_k3$tot.withinss
```

5. We could run the k-means algorithm multiple times to derive the total WSS but the *factoextra* package contains the function `fviz_nbclust()` that will do this for us and will produce a plot of the result.

```
install.packages("factoextra")
library(factoextra)
fviz_nbclust(iris_c, kmeans, method = "wss")
```

6. By default, `fviz_nbclust()` will produce results for between 1 and 10 clusters. The argument `k.max` can be used to specify the maximum number of clusters.

```
fviz_nbclust(iris_c, kmeans, method = "wss", k.max = 12)
```

Silhouette method

Another way to determine the optimal number of clusters is to compare distances between points within clusters to their distances from points in other clusters. Ideally, points within a cluster should be tightly grouped (compact) and clusters should be well-separated. The average silhouette width is used to measure of this.

Sihouette width is calculated by calculating the average distance between every point within a cluster and comparing to the minimum distance to a point from a different cluster.

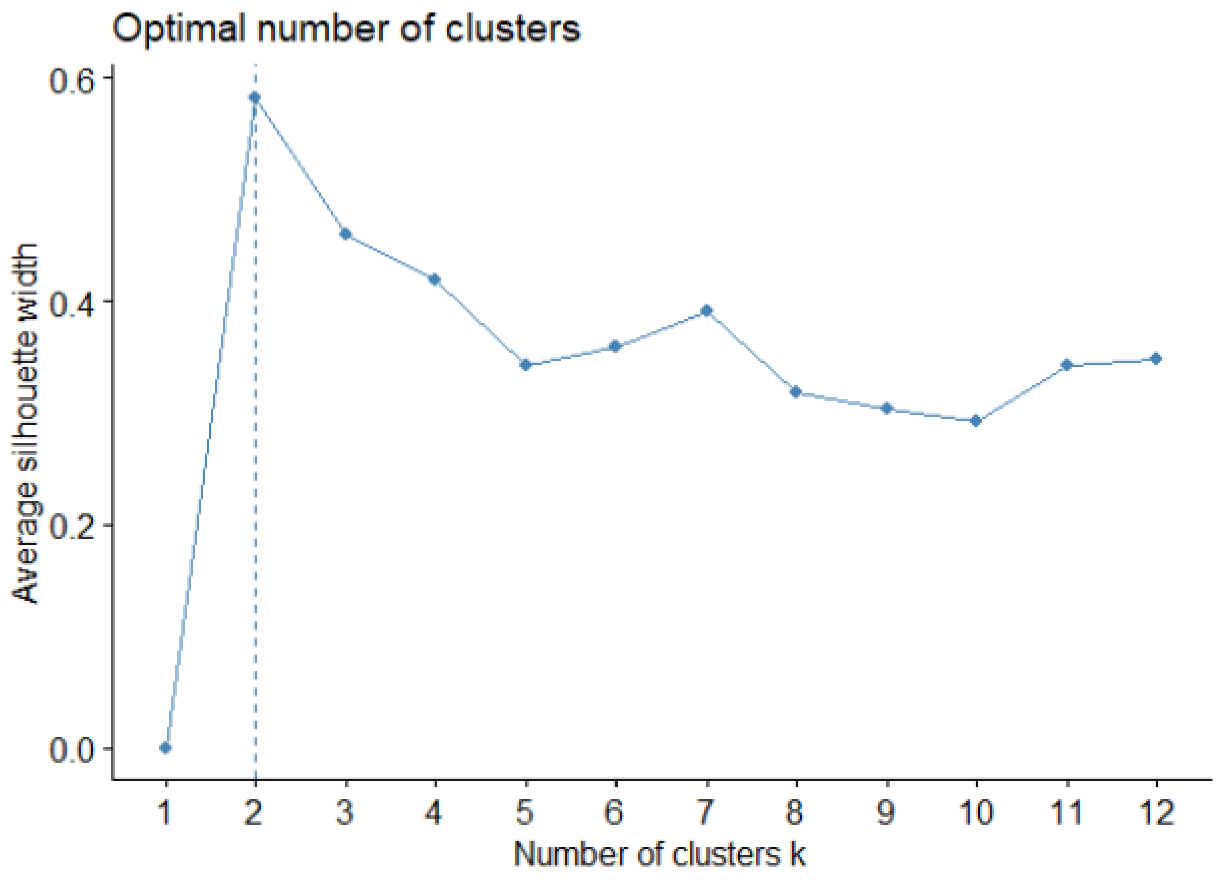
A silhouette width closer to 1 indicates that the observation is more similar to other observations in its cluster than to the most similar point in another cluster.

Values closer to 0 indicate that a point lies a similar distance to points from two different clusters. In this case, it is not clear which cluster the point should be assigned to.

The average silhouette width can be calculated and plotted against the number of clusters to determine the optimal cluster number for the dataset (Figure 5). The number of clusters that results in the largest average silhouette width is considered optimal. The larger the average silhouette width, the better the clustering.

The silhouette method can be very informative but it is very computationally demanding.

Figure 5. Silhouette method for determining optimal number of clusters



In this case, the silhouette graph indicates that the optimal number of clusters is 2.

Ex 2.5 - Create a silhouette graph for iris dataset clusters

1. The `fviz_nbclust()` function also has an option to produce a silhouette graph.

```
fviz_nbclust(iris_c, kmeans, method = "silhouette")
```

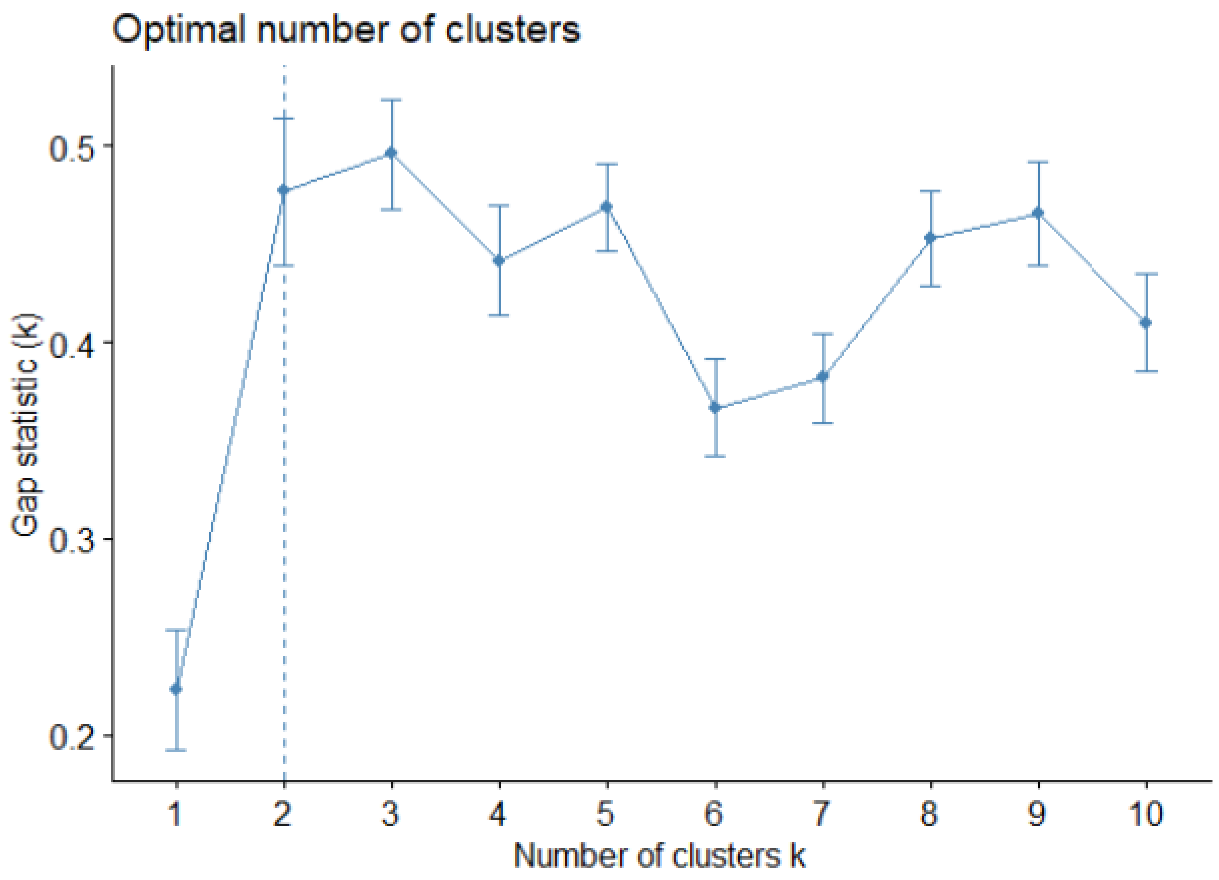
The gap statistic

Another popular method for determining optimal number of clusters using the `fviz_nbclust()` function is the gap statistic. This method compares the within-cluster variation with the variation that would be expected if the data were randomly, uniformly distributed.

Calculated for different cluster numbers, we can plot the results. The `fviz_nbclust()` function will plot a line graph with error bars representing one standard error (Figure 6). There are several ways of using the chart to determine the optimal number of clusters. These include choosing the smallest number of clusters that gives a first maximum gap statistic or simply the number that yields the maximum overall gap statistic.

More commonly, a smaller number of clusters is preferred. For example, choosing the optimal number of clusters (k) based on the gap being within one standard error of the gap statistic of k+1 clusters.

Figure 6. Gap statistic for determining optimal number of clusters



In this case, the gap statistic indicates the optimal number of clusters is 2.

Ex 2.6 - Create a gap-statistic plot for iris dataset clusters

1. The `fviz_nbclust()` function also has an option to produce a gap-statistic plot.

```
fviz_nbclust(iris_c, kmeans, method = "gap_stat")
```

Performing k-means clustering in R

Once the data have been transformed and scaled as necessary and you have decided on how many clusters to create, the final cluster analysis can be performed.

Running the algorithm multiple times can produce different results due to differences in the starting centres. If you do not specify what these starting values should be, the `kmeans()` function will randomly select rows from the dataset as the initial centres.

The `nstart` argument enables us to specify how many sets of initial centres to use. The function will return the best results. For example, if we set `nstart = 20`, the function will run 20 times with different initial centres each time. The function will return the result that has the smallest total within-cluster sum of squares.

Ex 2.7 - k-means cluster analysis

1. Perform the cluster analysis on the *iris_c* dataset.

```
set.seed(123)
iris_clust <- kmeans(iris_c, centers = 3, nstart = 50)
```

2. Take a look at the cluster means and the cluster allocations. What do the cluster means represent?
3. To calculate means for each cluster:

```
iris_res <- data.frame(iris, cluster =
                         factor(iris_clust$cluster))
tapply(iris_res$Petal.Length, iris_res$cluster,mean)
tapply(iris_res$Petal.Width, iris_res$cluster,mean)
tapply(iris_res$Sepal.Length, iris_res$cluster,mean)
tapply(iris_res$Sepal.Width, iris_res$cluster,mean)
```

4. We have an unusual case here because we have used a labelled dataset. We already know which species each observation belongs to. Usually, we would use cluster analysis when we have unlabelled data – when we don't know which species each observation belongs to. But for the purposes of this exercise, it makes it possible to compare the cluster to the actual species groups. We'll use the `table()` function to compare actual groups vs the allocated clusters.

```
table(iris_res$Species, iris_res$cluster)
```

5. Create plots to visualise differences in variables between species and clusters.

```
# How did measurements vary between species?
par(mfrow = c(1,2))
plot(iris_res$Petal.Length,iris_res$Sepal.Length,
          col = iris_res$Species)
legend("topleft", legend = levels(iris_res$Species),
          col = 1:3, lty = 1, cex = 0.8)
#How did measurements vary between clusters?
plot(iris_res$Petal.Length,iris_res$Sepal.Length,
          col = iris_res$cluster)
legend("topleft", legend = levels(iris_res$cluster),
          col = 1:3, lty = 1, cex = 0.8)
```

6. Note that cluster analysis is better able to detect the group that has values very distinct from the other groups. It is less good at detecting groups that have more similar / overlapping values.
7. To add the cluster means to the plot, we can use the `tapply()` function to calculate means for each cluster.

```
points(tapply(iris_res$Petal.Length, iris_res$cluster,mean),
       tapply(iris_res$Sepal.Length, iris_res$cluster,mean),
       col = 4, pch = "X")
```

8. We can also use a scatterplot matrix to visualise results.

```
pairs(iris_res[,1:4], col = iris_res$cluster)
```

Customer segmentation analysis

Now that we have worked through the cluster analysis process we will apply it to a customer segmentation problem. Customer segmentation is used in marketing to divide customers into groups (or *segments*) based on common characteristics. This can enable better marketing to customers, targeted product development and identification of particularly valuable customers. Segmentation can be based on geographic or demographic information. In this case we will base it on customer buying behaviour.

We will be using a dataset that contains transaction information. Each row of the dataset records the Sale ID, Date, Store ID, Customer ID, Product ID, Quantity and Paid amounts. Each row represents the sale of a particular product in a transaction; transactions that include multiple products are recorded over multiple rows.

Ex 2.8 - Import the sales transaction dataset

1. Import the Excel file named *SalesData*.
2. Rename the column names to remove spaces.

```
names(SalesData) <-
                    c("Sale_ID","Date","Store_ID",
                    "Customer_ID","Product_ID","Qty", "Paid")
```

Remember that we need numeric data for cluster analysis. In this case, we will do some feature engineering and derive some values from the raw data. We will use RFM analysis to calculate values that characterise the behaviour of our customers, based on their transaction histories.

RFM analysis

RFM stands for recency, frequency and monetary. Recency is number of days since customer last purchased, frequency is the total number of transactions they have made and monetary is the total amount of money they have spent.

All three of these calculated values are numeric. They can be calculated manually, however, the R package *rfm* contains a function that will compute these values for us.

The `rfm::rfm_table_order()` function can be used to calculate RFM values from transaction data. The first argument to this function is a data frame that contains the customer ID, order date and spend amount.

However, the data that we have includes extra information related to the store, product and quantity. The data for some transactions are also captured across multiple rows when multiple products have been purchased.

We must group and summarise the data before using it to calculate RFM.

Ex 2.9 - Calculating values to be used for cluster analysis

1. Group and summarise the sales data to create transaction data.

```
# install tidyverse, if required
install.packages("tidyverse")
# load tidyverse
library(tidyverse)

# group the sales data
grpSalesData<-group_by(SalesData, Store_ID, Date, Customer_ID)
# summarise the sales data
smrySalesData<-summarise(grpSalesData, Paid = sum(Paid))
```

2. We now have a summary of the transactions but have an extra column containing the store number. Let's drop this column.

```
 # drop the store id column
 transactData <- smrySalesData[,-1]
```

3. We can now calculate the RFM values.

```
# install and load the rfm package:
install.packages("rfm")
library(rfm)
```

```
# Days since last transaction will be calculated from the
# last transaction date
lastDate <- max(transactData$Date)
resultRFM <- rfm_table_order(transactData, customer_id =
Customer_ID, order_date = Date, revenue = Paid, analysis_dat =
lastDate)
```

4. This creates a list which includes a data frame named rfm, containing the calculated RFM values.

```
View(resultRFM$rfm)
```

5. We can save this into a data frame of its own.

```
salesRFM <- resultRFM$rfm
```

Examining the distributions

We now have the numeric values that we will use to perform the cluster analysis:

- *recency_days*
- *transaction_count*
- *amount*

We must prepare these values for cluster analysis.

Ex 2.10 - Data distribution and transformation

1. Examine the distributions of the 3 variables listed above. Do they need to be transformed before performing cluster analysis?

   Use histograms and the `DescTools::Skew()` function to evaluate the skewness of each. Are the data skewed?
2. Some of the data are highly skewed so we will transform the values to reduce the skewness. We can try taking square roots or logarithms of the raw values.

```
# Examine the skewness of square-root transformed variables
hist(sqrt(salesRFM$transaction_count))
Skew(sqrt(salesRFM$transaction_count))

hist(sqrt(salesRFM$amount))
Skew(sqrt(salesRFM$amount))
```

3. The square root transformation reduces the skewness of both variables. We can compare this with a logarithmic transformation.

```
# Examine the skewness of logarithmically transformed
# variables
hist(log10(salesRFM$transaction_count))
Skew(log10(salesRFM$transaction_count))
hist(log10(salesRFM$amount))
Skew(log10(salesRFM$amount))
```

We will use the square root transformation of *amount*. We will use the logarithmic transformation of *transaction_count*.

Scale the data

The amount data are orders of magnitude larger than *transaction_count* values (raw and transformed). Left unscaled, the amount values will have a larger impact on the allocation into classes than the other variables.

Ex 2.11 - Scale the data

1. Create a new column in *salesRFM* to hold the transformed and scaled variables.

```
salesRFM$ts_recency_days <- scale(salesRFM$recency_days)
salesRFM$ts_transaction_count <-
                   scale(log10(salesRFM$transaction_count))
salesRFM$ts_amount <- scale(sqrt(salesRFM$amount))
```

How many clusters?

Use the elbow, silhouette and gap statistic methods to help select the number of clusters for this analysis.

Ex 2.12 - How many clusters?

1. Use the `fviz_nbclust()` function to support the choice of number of clusters.

```
x <- 10:12
fviz_nbclust(salesRFM[,x], kmeans, method = "wss")
fviz_nbclust(salesRFM[,x], kmeans, method = "silhouette")
fviz_nbclust(salesRFM[,x], kmeans, method = "gap_stat")
```

2. What is the optimal number of clusters for this dataset?

When the optimal number of clusters is not clear, it is worth considering the practical application of the cluster information. How many clusters will be manageable or usable? What can you do with this data? These statistics can help your decision about how many clusters but ultimately it will be decided by domain knowledge or the business case.

For the rest of this exercise, we'll use 6 clusters. This might correspond to high/low combinations of the cluster variables.

Customer segmentation

Now that we have derived our numeric variables, transformed and scaled our data and chosen the number of clusters, we are ready to segment our data.

Ex 2.13 - Segment the data

1. Use the k-means algorithm to partition the data into 6 clusters.

```
set.seed(123)
rfm_clust <- kmeans(salesRFM[,x], centers = 6,
               nstart = 50)
```

2. Visualise the results with a scatterplot matrix.

```
pairs(salesRFM[,3:5], col = rfm_clust$cluster)
```

3. Create a scatterplot. Jittering can help.

```
plot(salesRFM$recency_days, salesRFM$amount,
          col = rfm_clust$cluster)
plot(jitter(salesRFM$transaction_count, factor = 2),
          salesRFM$amount, col = rfm_clust$cluster)
plot(jitter(salesRFM$transaction_count, factor = 2),
          salesRFM$recency_days, col = rfm_clust$cluster)
legend(x="topright",
          legend = levels(factor(rfm_clust$cluster)),
          fill = levels(factor(rfm_clust$cluster)), cex = 0.8)
```

4. Create a summary of the mean values for each cluster.

```
rfmMeans <- data.frame(
               recency_days = tapply(salesRFM$recency_days,
               rfm_clust$cluster,mean),
               transaction_count =
               tapply(salesRFM$transaction_count,
               rfm_clust$cluster,mean),
               amount = tapply(salesRFM$amount,
               rfm_clust$cluster,mean))
```

5. Add the cluster centres to the last plot.

```
points(rfmMeans$transaction_count, rfmMeans$recency_days,
                      col = 1, pch = "x", cex = 2)
```

Other clustering algorithms

The R package *cluster* contains a number of algorithms that can be used to perform cluster analysis.

The `pam()` function is a version of k-means that is a little bit more robust because instead of using calculated means as centroids, it uses medoids. Medoids are actual data points in the dataset (middle points) and this makes it less sensitive to outliers. One drawback is that it does not work so well with larger datasets because it is more computationally intensive.

Ex 2.14 - Use the `pam()` function to segment the RFM data

1. Install and load the cluster package and cluster using the `pam()` function.

```
install.packages("cluster")
library(cluster)
rfm_pam_clust <- pam(salesRFM[,10:12], k = 6)
```

2. Compare results to kmeans.

```
table(rfm_pam_clust$clustering, rfm_clust$cluster)
plot(salesRFM$recency_days, salesRFM$amount,
          col = rfm_pam_clust$clustering)
plot(jitter(salesRFM$transaction_count, factor = 2),
     salesRFM$amount, col = rfm_pam_clust$clustering)
plot(jitter(salesRFM$transaction_count, factor = 2),
     salesRFM$recency_days, col = rfm_pam_clust$clustering)
```

3. Random forests

The random forest method is a supervised learning algorithm that is used for prediction. It is a flexible technique that is used for both classification and regression problems.

Classification involves predicting a category or a discrete group while regression involves predicting continuous, numeric values.

A random forest model is created by constructing many individual tree models and aggregating the results.

Basics of tree-based models

Random forests evolved from decision trees. Decision trees work by generating a set of rules to make predictions. Like random forests, decision trees are based on a supervised learning algorithm and can be used for both classification and regression. The resulting model (and rules) can be visualised in an easy-to-understand tree (hence the name).

The decision tree in Figure 7 shows the classification rules for predicting which species an iris flower belongs to, based on petal and sepal measurements. The decision tree in Figure 8 shows the regression rules for predicting petal length based on species and other measurements in the iris dataset.

The rules for this tree were generated by binary recursive partitioning – that is by repeatedly splitting the data in two. The subsets are called nodes and are indicated by the lines on the decision tree. The root node is found at the top of the tree. Nodes that are split are termed parent nodes and produce child nodes. In determining how to split the data, the algorithm decides which predictor variable to use and its value.

The split is chosen in a way that minimises the amount of variability in the dependent variable (the class or quantity being predicted) in the nodes. That is, splitting the data in a way that will reduce the prediction error. The splitting process continues until a stopping criterion is reached. These final subsets are called the terminal nodes and on the decision tree are found at the bottom of the tree. These are the final predictions and they represent either the majority class or the average of the response values in that terminal node.

By following the tree diagram top-down, it is possible to use the rules generated to find the prediction. If the splitting criteria is true, the child node on the left is followed down the tree. If the splitting criteria is false, the right-hand child node is followed.

Figure 7. Classification tree for the iris dataset

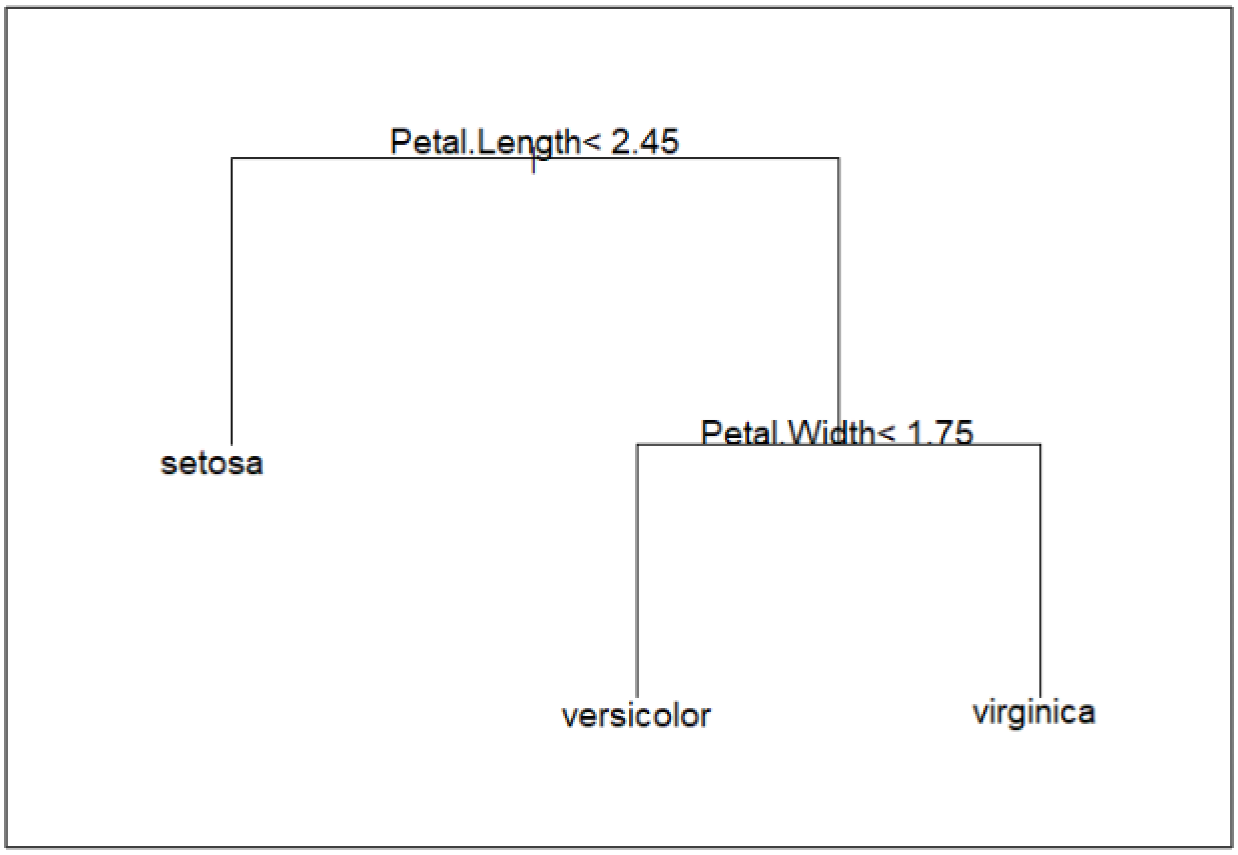
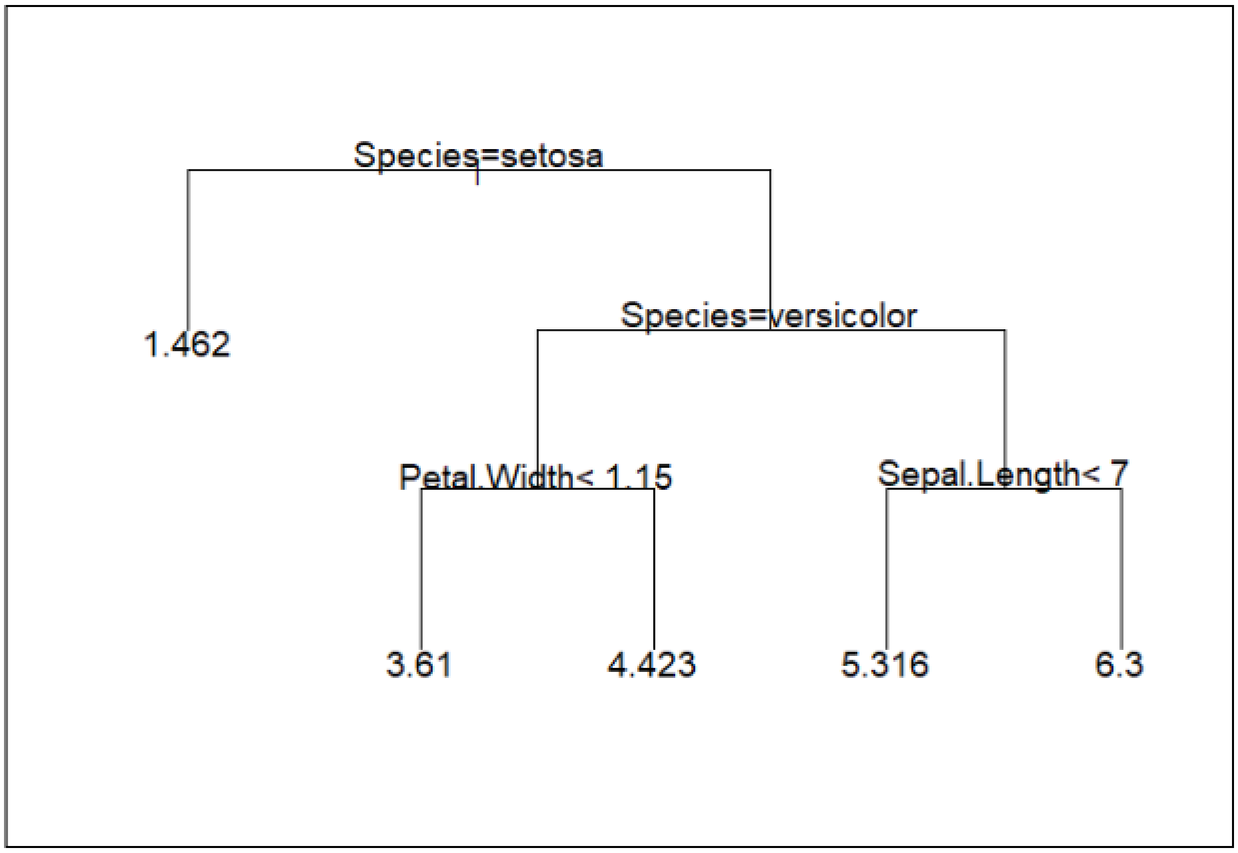


Figure 8. Regression tree for predicting petal length in the iris dataset



back to top

Strengths of decision trees

Decision trees can be used in a wide range of applications, since they can deal with both regression and classification problems and can handle both numeric and categorical predictor variables.

While many machine learning algorithms require certain assumptions to be met, e.g. assumptions about the distribution of variables or about the nature of the relationship between predictors and dependent variables, decision trees rely on few if any, assumptions.

Another strength of decision trees is the ease with which they can be interpreted. For example, in Figure 7, an observation with a petal length less than 2.45 would be predicted to belong to the *setosa* species. Those with a petal length of at least 2.45 and a petal width less than 1.75 would be predicted to belong to *versicolor* species; while those with a petal width of 1.75 or more would be predicted to belong to *virginica* species.

Importantly, decision trees select the most important variables for prediction and model interactions between variables. The visualisation identifies the important variables for prediction which can be helpful in understanding and potentially managing the response variable.

Limitations of decision trees

However, decision trees do have certain limitations. Figure 8 contains a regression tree which predicts petal length based on the species and other measurements in the dataset. The tree contains 5 terminal nodes which means that any prediction of this continuous, numeric variable will be one of these 5 values. This can limit the accuracy of predictions.

There is also a danger of *overfitting* decision trees. Overfitting happens when the model produced captures too much of the variability in the training dataset. A decision tree can keep growing (splitting the data) until there is only 1 observation in each terminal node. This model will be too flexible.

Instead of just capturing the underlying relationship, it will capture some of the noise or scatter in the training dataset. It will represent the pattern in the training dataset very well but will not generalise well when predicting with new data. This reduces the model accuracy when used to predict with new data (data not included in the training dataset).

The bias-variance trade-off

Predictive models will always contain some inaccuracies. These can be due to "noise" or random irregularity in the data (usually due to other factors). Errors can also be due to *overfitting* which increases the variance explained by the model. Overfit models are very sensitive to small changes in the training dataset which alter the predictions made from the model.

The bias-variance trade-off occurs whenever we fit a model to data. A model that is too flexible will have higher variance but lower bias. Models that have been overfit to the training data have higher variance and poor predictive accuracy. If we trained the model on different data, we would be likely to get very different predictions.

But a less flexible model may *underfit* the data and introduce more *bias*. This is the error associated with assumptions made about how the model should fit to the data.

Bias essentially measures the difference between actual and predicted values. A more biased model will have bigger differences between actual and predicted values but will be less sensitive to small changes in the training data.

In most predictive modelling problems, we have a trade-off between bias and variance. Decision trees have low bias but high variance.

From trees to (random) forests

Random forests were designed to overcome the limitations of decision trees. Instead of growing a single tree, we grow many – a forest! By creating many trees and aggregating results to create a single model, overfitting is reduced and prediction accuracy improves.

When individual trees are created by the random forest algorithm, only a random subset of predictor variables are considered for each tree model. This leads to the production of tree predictions that are not highly correlated and results in a greater reduction in the variance of the model.

As variance decreases with the creation of the random forest, bias increases. The aim is to find a trade-off that improves the model overall.

Unlike individual decision trees, random forest models are not easily visualised or interpreted but they have much better predictive accuracy. We can also obtain a summary of the most important variables for prediction from the random forest model.

Ensemble learning: bagging to reduce overfitting and improve predictive accuracy

Random forests are described as an *ensemble* method because they combine several models (decision trees) to produce a single, final model. Ensemble methods are used to improve model accuracy.

How do we fit multiple models to the same dataset?

Random forests use a technique called *bootstrapping* to select new data subsets from the training data. This involves randomly selecting observations from the training data, with replacement. This creates a new dataset, which is usually the same size as the original dataset.

A decision tree is then grown on the bootstrapped dataset. The tree is grown very large and will overfit the data.

This process of creating bootstrap samples and fitting a tree to each is repeated hundreds or thousands of times.

The two random processes of selecting predictors and bootstrapping the dataset result in independent predictions between each tree model.

Bagging

Bagging is the process of averaging the predictions from each of these trees to come up with the final model prediction. In a classification problem, the prediction is the majority class predicted from all trees. This process reduces the variance (overfitting) of the model.

The process of supervised machine learning

There is a general sequence of steps that are followed when creating predictive models using machine learning.

This involves:

- Data collection
- Data preparation / feature engineering
- Splitting of the dataset - creating training and test sets
- Algorithm selection
- Training the model on the training set
- Hyperparameter tuning
- Evaluation of the final model by comparison with the test set
- Deployment of the final model.

Feature engineering

Feature engineering is the process of extracting "features" or variables from raw data that will be used as inputs to the machine learning model. It includes the calculation of new quantities, data transformations and scaling (as in the customer segmentation example), dealing with missing values, recoding categories and binning continuous values into groups. Feature engineering is an essential part of achieving good model accuracy.

Good feature engineering requires that you have a good understanding of the problem you are trying to solve. Domain knowledge should guide the process of selecting data to collect, use and derive based on understanding the factors that are likely to be related to the outcome.

Data should be in a data frame with each variable in its own column. Each row should represent one observation (measurement). Random forests can use continuous numeric or discrete (category) predictors.

Splitting data into training and test sets

The dataset must be split into training and test sets. This is done so that the model can be trained on one portion of the data. Then its predictive accuracy can be evaluated using data that were not used to build the model.

Sometimes the data are split into 3, to also create a validation set, used to optimise hyperparameters. However, with the random forest model we can use out of bag samples instead (see *Improving the Model*).

Common ratios for splitting the data into training and test sets are 80:20 or 70:30.

Strategies for data splitting

When we split the data, we want each dataset to be representative of the entire dataset. We do not want to miss low or high predicted values or entire categories. We may want to ensure that the same range of predictor variable values are sampled in both data sets. There are many approaches to ensuring that this occurs, usually involving some form of stratified sampling.

Fortunately, the *caret* package in R was created to make the process of creating predictive models easier. It has several methods for splitting datasets. See https://topepo.github.io/caret/data-splitting.html for further details.

Training the model

Training the model involves fitting the selected algorithm to the training dataset.

Improving the model

Hyperparameters are parameters of the algorithm that are set before training the model. By varying the hyperparameters we can "tune" the model to improve its accuracy.

The optimal hyperparameters will be different for each dataset. Random forest algorithms are much-loved in part because the default hyperparameter settings often provide good results but improvements may be possible through tuning.

Hyperparameters of random forests include:

- number of trees to be grown
- number of features (variables) considered at each split during training,
- minimum number of observations in the terminal node.

Research has shown that more trees are better and that this and the number of features considered are most important for predictive accuracy.

After adjusting the hyperparameters, the model is retrained on the training set and the accuracy of the new predictions are assessed. Assessing accuracy at this stage often uses the out-of-bag samples – these are the observations that do not get selected in the bootstrap sample.

The goal is to produce a model which is as accurate as possible while remaining computationally manageable.

Evaluating the final model

The test dataset is used to test the model's predictive accuracy. Actual, measured values from the test set are compared with predictions made by the model and used to calculate measures of model quality.

Classification models

Classification model quality can be evaluated by calculating *accuracy* and by creating a *confusion matrix* (Figure 9).

Accuracy = number of correct predictions / total number of predictions

The confusion matrix provides more information, by creating a matrix comparing actual to predicted classes:

Figure 9. A confusion matrix

	Predicted A	Predicted B	Predicted C
True A	50	7	11
True B	10	40	9
True C	5	8	35

For any class,

- grey cells are true positives (TP)
- sum of white cells in a row are the false negatives (FN)
- sum of white cells in a column are the false positives (FP)
- true negatives (TN) are sum of all cells, excluding those in the row or column labelled with that class. The TN for class A are calculated by summing the values in the rounded rectangle.

From this matrix, other measures such as *sensitivity*, *specificity* and *precision* can be calculated for each class.

Sensitivity (also referred to as *recall*) is the ability of the model to correctly predict when an observation is in the class. Specificity is the ability of the predictor to correctly predict when an observation is *not* in that class. Precision is the ratio of true positives to all predicted positives and indicates how confident you can be that predicted positives are correct. How well does the model avoid labelling negative samples positive? There is often a trade-off in these metrics.

e.g., for class A,

sensitivity / recall = TP/(TP + FN) = 50/(50 + 7 + 11) = 0.74

specificity = TN/(TN + FP) = (40 + 8 + 9 + 35)/(40 + 8 + 9 + 35 + 10 + 5) = 0.86

precision = TP/(TP+FP)

Other metrics

Regression models

Measures of model quality for regression models include mean absolute error (MAE), root mean square error (RMSE) and R-squared values.

MAE is simply the average of the residuals (prediction errors).

MAE = mean(abs(predicted values – actual values))

$$MAE = mean(|predicted\ values - actual\ values|)$$

RMSE is the standard deviation of the residuals

$$RMSE = \sqrt{\text{mean}((\text{predicted value} - \text{actual value})^2})$$

RMSE is a widely-used metric but can be sensitive to outliers. Larger errors are punished more by this metric.

Both MAE and RMSE have the same units as the predictor variable and the closer they are to 0, the more accurate the model.

The R-squared value represents the proportion of variance explained by the model and ranges in value from 0 to 100. An R-squared value of 100 explains all of the variance in the response variable. R-squared values should be reported for the training data not for the test set.

The root mean squared logarithmic error (RMSLE) is less common but can also be used to evaluate the quality of your prediction. This penalises under-estimation more than over-estimation. In a business example like this one, we may prefer to over-estimate rather than under-estimate demand. The closer this value is to 0, the more accurate the model.

$$RMSLE = \sqrt{\text{mean}((\ln(\text{predicted value} + 1) - \ln(\text{actual value} + 1))^2})$$

In this formula, 1 is added to predicted and actual values to avoid taking a logarithm of 0 (actual and predicted values must be ≥ 1).

The process of creating a random forest model

Many machine learning algorithms require the data to meet certain assumptions and for the best variables for prediction to have been pre-selected. Random forests require few (if any) assumptions to be met and are generally not sensitive to outliers. The model fitting process includes a method of variable selection and automatically accounts for interactions between variables.

As stated earlier, there is no need to create a validation set for optimising hyperparameters, since the out-of-bag samples can be used at this step. This allows more of the data to be used for training. Hyperparameter tuning can improve model performance but using default values is often sufficient.

For the reasons above, it is often referred to as an "out-of-the-box" solution.

The random forest algorithm creates a predictive classification or regression model and provides measures of variable performance.

Random forests in R

Random forests can be created in R using functions from a range of contributed packages. The two functions we will use today are `randomForest()` and `ranger()`.

The *randomForest* package provides an implementation of the original random forest algorithm (developed by Breiman and Cutler in Fortran).

The *ranger* package provides a faster implementation of the original random forest algorithm.

Both of these functions can be used with the *caret* package, designed to simplify the process of creating predictive models. We will use *caret* functions to split datasets, train and tune models and evaluate performance.

When used with *caret*, more hyperparameters can be tuned for the `ranger()` function compared to the `randomForest()` function.

Classification tree and random forest classification model

Ex 3.1 - Create a random forest classification model to predict weight status

In this exercise we will create a classification tree and random forest model to predict weight classes based on dietary, lifestyle and other factors.

1. Import the file *PredictObesity.csv*. Import the data using the *base* function. Note that importing the data using *readr* will create a tibble rather than an ordinary data frame. Tibbles do not convert character data to factors by default and these columns will need to be converted manually.
2. The file contains a number of columns, each containing a variable that may be related to an individual's weight, such as dietary and physical activity data. The last column, *weight_status* indicates whether that person's weight is classified as healthy, overweight or obese.
3. We don't need to do any feature engineering with this dataset. We'll create and plot a decision tree first so that we can visualise some of the relationships in this dataset.

```
library(rpart)
library(rpart.plot)
ob_tree <- rpart(weight_status ~ ., data = PredictObesity)
rpart.plot(ob_tree)
```

4. Before we fit our random forest, we will split the data into test and training datasets using the caret package.

```
library(caret)
obsplit <- createDataPartition(PredictObesity$weight_status,
               p = 0.8, list = FALSE)
obtrain <- PredictObesity[obsplit,]
obtest <- PredictObesity[-obsplit,]
# fit a random forest
library(randomForest)
set.seed(123)
ob_rf <- randomForest(weight_status ~ ., data = obtrain,
          importance = TRUE)
```

5. The accuracy of the model produced can be tested by predicting *weight_status* using the test dataset.

```
ob_rf_pr <- predict(ob_rf, newdata = obtest)
```

6. The accuracy of the model can be evaluated by using a confusion matrix to compare the predicted values to the *weight_status* values in the test dataset.

```
confusionMatrix(ob_rf_pr, obtest$weight_status)
```

7. Explore the named components of *ob_rf*.

```
ob_rf$importance
```

8. Plot variable importance.

```
varImpPlot(ob_rf)
```

9. Is this the best model we can fit to this dataset? We can use the `caret::train()` function to see if modifying some of the hyperparameters will improve the accuracy of the predictive model.

```
# Set up a grid of values to try
tgrid <- expand.grid(mtry = 2:12)
set.seed(123)
ob_rf_tr <- train(weight_status ~ ., data = obtrain,
               method = "rf",
               trControl = trainControl(method = "oob"),
               tuneGrid = tgrid,
               num.trees = 500,
               importance = TRUE)
```

10. We can plot the results to see how the accuracy of the model changes as we vary the *mtry* (number of predictors) hyperparameter. Note that the caret package only allows us to tune this hyperparameter for *randomForest* models. We can use other random forest algorithms to tune multiple parameters as shown in the next step.

```
plot(ob_rf_tr)
ob_rf_tr$bestTune
```

```
ob_rf_tpr <- predict(ob_rf_tr, newdata = obtest)
confusionMatrix(ob_rf_tpr, obtest$weight_status)
```

11. If we use the `ranger()` function instead of `randomForest()`, we have the ability to also tune the node sizes.

```
library(ranger)
tgrid <- expand.grid(
               .mtry = 3:8,
               .min.node.size = c(5,6,7,8,9,10),
               .splitrule = "gini"
               )
set.seed(123)
ob_rf_tr <- train(weight_status ~ ., data = obtrain,
               method = "ranger",
               trControl = trainControl(method = "oob"),
               tuneGrid = tgrid,
               num.trees = 500)
ob_rf_tpr <- predict(ob_rf_tr, newdata = obtest)
confusionMatrix(ob_rf_tpr, obtest$weight_status)
```

12. Did you see much improvement in the model when you tuned the hyperparameters? One of the reasons that random forests are popular is that they often work well using the default hyperparameters.
13. What would the accuracy be if we predicted using the most common class? How much does the random forest improve upon this?

```
# how many predictions were made?
length(obtest$weight_status)

# how many observations are there for each weight status in
# the test set?
table(obtest$weight_status)

# Divide the count of the most common class by the total
# number of observations.
table(obtest$weight_status)["Overweight"] /
               length(obtest$weight_status)
```

Regression tree and random forest regression model

Ex 3.2 - Regression tree and random forest prediction model

In this exercise we will predict bike-share demand based on date, time and weather information.

1. Download the dataset from https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset . Go to *Data Folder* and download the zip file. We will be working with the *hour* file.
2. Import the dataset *hour.csv*.
3. Explore relationships between the variables. Are there any relationships between variables and the number of casual, registered and total users?
4. Summarise or visualise the spread of casual, registered and total users.
5. Create regression trees to visualise some of the relationships in the data.

```
library(rpart)
library(rpart.plot)
# create a tree for total users
bk_tree <- rpart(cnt ~ season + yr + mnth + hr + holiday +
weekday + workingday + weathersit + temp + atemp + hum +
windspeed, data = hour)
rpart.plot(bk_tree)


# create a tree for registered users
bk_tree_r <- rpart(registered ~ season + yr + mnth + hr +
holiday + weekday + workingday + weathersit + temp + atemp +
hum + windspeed, data = hour)
rpart.plot(bk_tree_r)


# create a tree for casual users
bk_tree_c <- rpart(casual ~ season + yr + mnth + hr + holiday
+ weekday + workingday + weathersit + temp + atemp + hum +
windspeed, data = hour)
rpart.plot(bk_tree_c)
```

6. Split the data into training and test sets. This problem is slightly different, in that we are looking at changes over time. The data show yearly, monthly and daily trends. We will use data from the first 19 days of each month for training the model and the remaining data for our test set.

```
library(lubridate)
bk_train <- hour[day(as_date(hour$dteday)) < 20,]
bk_test <- hour[day(as_date(hour$dteday)) >= 20,]
```

7. Create a random forest regression model to predict total number of users.

```
set.seed(123)
bk_rf <- randomForest(cnt ~ season + yr + mnth + hr +
                        holiday + weekday + workingday +
                        weathersit + temp + atemp + hum +
                        windspeed, data = bk_train,
                        importance = TRUE)
```

8. We can use the `caret::postResample()` function to calculate RMSE and MAE values.

```
bk_rf_preds <- predict(bk_rf, newdata = bk_test)
postResample(bk_rf_preds, bk_test$cnt)
```

9. We can also calculate the RMSLE

```
library(Metrics)
rmsle(bk_test$cnt, bk_rf_preds)
```

10. Which are the most important variables in the model?

```
varImpPlot(bk_rf)
```

11. Repeat the steps above but increase the number of trees to 1000. Does this improve the model?

```
set.seed(123)
bk_rf <- randomForest(cnt ~ season + yr + mnth + hr +
                         holiday + weekday + workingday +
                         weathersit + temp + atemp + hum +
                         windspeed, data = bk_train,
                         ntrees = 1000)
```

12. Let's vary the number of predictor variables considered at each split.

```
tgrid <- expand.grid( mtry = 3:10 )
set.seed(123)
bk_rft <- train(cnt ~ season + yr + mnth + hr +
                    holiday + weekday + workingday +
                    weathersit + temp + atemp + hum +
                    windspeed, data = bk_train,
                    method = "rf",
                    trControl = trainControl(method="oob"),
                    tuneGrid = tgrid,
                    num.trees = 500,
                    importance = TRUE)


bk_rft_preds <- predict(bk_rft, newdata = bk_test)
postResample(bk_rft_preds, bk_test$cnt)
rmsle(bk_test$cnt, bk_rft_preds)
```

13. Plot to see how accuracy changes as number of predictors tried varies.

```
plot(bk_rft)
```

14. What are the most important predictors?

```
plot(varImp(bk_rft))
```

15. Let's vary the number of predictor variables considered at each split and the minimum node size.

```
tgrid <- expand.grid(
.mtry = 2:12,
.min.node.size =  10,
.splitrule = "variance"
)
set.seed(123)
bk_rfr <- train(cnt ~ season + yr + mnth + hr + holiday +
                    weekday + workingday + weathersit + temp +
                    atemp + hum + windspeed,
                    data = bk_train,
                    method = "ranger",
                    trControl = trainControl(method="oob"),
                    tuneGrid = tgrid,
                    num.trees = 500,
                    importance = "impurity")

bk_rfr_preds <- predict(bk_rfr, newdata = bk_test)
postResample(bk_rfr_preds, bk_test$cnt)
rmsle(bk_test$cnt, bk_rfr_preds)
```

16. We saw that casual and registered users have different behaviour. Perhaps we can improve predictions by modelling each separately?

Improving the model

The random forest is known for working well with many datasets, out of the box. We have seen that hyperparameter tuning can sometimes increase prediction accuracy.

Other ways of improving predictions include:

- Collecting more data
- Measuring other variables
- Better feature engineering – deriving features (predictor variables) that better predict the response
- Choose a better algorithm – there is no single algorithm that works best in every situation. Often, this is simply a case of trial and error.

back to top

4. R scripts in Power BI

Why bring a machine learning model into Power BI?

Power BI makes it easy to visualise data through its point-and-click interface. It allows data and visualisations to be shared within an organisation as reports or dashboards. And it streamlines the process of connecting to data sources and updating data.

Power BI also allows R scripts to be run for both data transformation and visualisation. This enables you to bring the machine learning models that you create in R into Power BI. There are two ways to do this. You can bring in the code to recreate the model each time. We will do this for the cluster analysis. Or you can create a model, export it as an R object and then load it for use and prediction in Power BI. We will do this for the random forest model.

When data refreshes, your model will automatically refresh – retraining on newly-available data or predicting with new predictor datasets as they become available.

In this section, we'll use our k-means algorithm in an R script in Power BI and see how it can be used to dynamically update and re-segment customers as more data becomes available.

We'll also use our bike share model, to see how it can be used to provide on-going forecasts of bike demand.

Setting up Power BI to use R scripts

If R is installed on your machine, Power BI should detect this. We will check the Power BI options to ensure that it detects the presence of R and RStudio.

Ex 4.1 - Checking R settings in Power BI

1. Click on **File** > **Options and settings** > **Options** > **GLOBAL** > **R Scripting**
2. Under **Detected R home directories:** there should be a folder path. If you have multiple versions of R installed, you can select the one you want Power BI to use, via the dropdown.
3. Under **Detected R IDEs:** you should see RStudio and any other integrated development environments you have installed for R.

Cluster analysis in Power BI

In this exercise, we will load multiple, related tables of data related to our sales, stores, products and customers. Customer segmentation can be performed dynamically: as data are added, customers are re-segmented. Customer segments can then be related back to any other data we hold about our customers in the *Customer* table.

Ex 4.2 - Dynamically perform cluster analysis in Power BI

1. Connect to the data source. Click on **Get Data** > **Excel**. Navigate to the file *SalesDataTables.xls* and click **Open**. Tick the boxes to select all the tables – *Customer, Product, SaleFact* and *Store*. Click **Load**.
2. Select the data view to see the four tables of data that have been imported.
3. Select the model view to see how those tables are related. Power BI simplifies the process of working with data organised in multiple, related tables.
4. Click on **Transform Data** to open Power Query. This is where we will run the cluster analysis.
5. Create a copy of the *SaleFact* query and rename it *Cluster*.
6. Before using an R script on the data in *Cluster*, change the type of the *Date* column to text. R doesn't recognise the Power BI date formats.
7. With *Cluster* selected, click **Transform** > **Run R script**.
8. The R script box will open and contains the following text:

   # 'dataset' holds the input data for this script

   This means that the data in the query has been put into a data frame named *dataset*. You need to use *dataset* in your R code to refer to the data in the query.
9. Copy the RFM cluster analysis code that you created earlier and paste it under the existing text in the **Run R script** box. We will need to make a few modifications.
10. Make the following changes:
    - **a.** Assign the contents of dataset into *SalesData*.
    - **b.** Move any `library()` commands to the top of the script.
    - **c.** Remove any `install()` commands.
    - **d.** Remove all data frames except for *rfm_clust* by adding the following to the end of the code.

11. Your final code should look like this:

```
# 'dataset' holds the input data for this script
library(tidyverse)
library(rfm)
library(lubridate)

SalesData <- dataset
SalesData$Date <- dmy(SalesData$Date)
names(SalesData) <-
c("Sale_ID","Date","Store_ID","Customer_ID","Product_ID","Qty"
, "Paid")
# group and summarise sales data
grpSalesData<-group_by(SalesData, Store_ID, Date, Customer_ID)
smrySalesData<-summarise(grpSalesData, Paid = sum(Paid))

# drop the store id column
transactData <- smrySalesData[,-1]

lastDate <- max(transactData$Date)
resultRFM <- rfm_table_order(transactData,
               customer_id = Customer_ID, order_date = Date,
               revenue = Paid, analysis_dat = lastDate)

salesRFM <- resultRFM$rfm
salesRFM$ts_recency_days <- scale(salesRFM$recency_days)
salesRFM$ts_transaction_count <-
               scale(log10(salesRFM$transaction_count))
salesRFM$ts_amount <- scale(sqrt(salesRFM$amount))

rfm_clust <- kmeans(salesRFM[,10:12], centers = 6,
               nstart = 50)
salesRFM$cluster <- rfm_clust$cluster

rm(grpSalesData)
rm(resultRFM)
rm(SalesData)
rm(smrySalesData)
rm(transactData)
```

12. If you are asked about privacy settings, set them to Public or check the box for **Ignore Privacy Levels** and **Save**.
13. Click on **Table** to see the results if they do not expand automatically. The *Cluster* query will now contain the RFM values and the cluster number they have been allocated to.
14. Select the *Customer* query. Click **Home** > **Merge Queries** and merge with *Cluster*, on customer ID in each query.
15. Select *recency_days, transaction_count, amount* and *cluster* as columns to expand. Do not use the original column name as prefix.
16. We can write the *Customer* query to a file with another R script.

```
library(lubridate)
td <- today()
write.csv(dataset, paste("D:\\Customer", day(td), month(td),
                year(td), ".csv", sep =""))
dataset2 <- dataset
```

17. Close and apply and create the following visualisations:
    a. Clustered column chart of count of *customer ID* vs *cluster*, *gender* as legend.
    b. Clustered column of count of *Customer ID* vs *cluster*, *Source* as legend.
    c. Clustered column of count of *Customer ID* vs *cluster*, *Age Grp* as legend
    d. Table of cluster, median amount, median transaction count, median recency and count of customer ID.
    e. On a new report page, create a line graph to show *Paid* vs *Year* and *Month*. Create a slicer based on cluster. See how the trends differ between clusters.
18. We can also create an R script visual which will allow us to use some of the built-in RFM plotting functions.
    a. Click on **R script visual** in the **Visualisations** pane.
    b. The R script editor will open. Select the *SaleFact* fields *Customer ID*, *Store ID*, *Date* and *Paid*. Only the *Paid* field should be summarised (sum).
    c. In the R script editor, enter the following code and then click **Run script**.

```
library(rfm)
library(lubridate)
dataset$Date = as.Date(dataset$Date)
analysis_date<-max(dataset$Date)
dataset$Date = as.Date(dataset$Date)
names(dataset)<-c("Customer_ID","Store_ID","Date","Paid")
SalesRFG <- rfm_table_order(dataset, Customer_ID, Date, Paid,
                analysis_date,recency_bins=3,frequency_bins=3,
                monetary_bins=3)
rfm_bar_chart(SalesRFG)
```

19. You can also experiment with the following plotting functions:

```
rfm_histograms(SalesRFG,hist_bins=8)
rfm_order_dist(SalesRFG)
rfm_rm_plot(SalesRFG)
rfm_heatmap(SalesRFG)
```

20. Go into the *SalesDataTables* file and resize the *SaleFact* table to include all of the data. Go back into Power BI and click on refresh to see how the clusters update.

Bike-share forecasting in Power BI

In the following example, the model created in R is used to predict 5 days into the future.

Ex 4.3 - Use bike-share model to make a forecast in Power BI

1. In R, run the following code:

```
# Use all data prior to 18/9/2012 to create the model

bike_data <- filter(hour, dteday < "2012-09-18")

final_model <- train(cnt ~ season + yr + mnth + hr + holiday +
                    weekday + workingday + weathersit + temp +
                    atemp + hum + windspeed,
                data = bk_train, method = "ranger",
                trControl = trainControl(method="oob"),
                tuneGrid = bk_rfr$bestTune,
                num.trees = 500,
                importance = "impurity")

# Export the model
saveRDS(final_model, "bike_model")
```

2. Open the Power BI file named Random Forest. It contains just one table which contains weather data from 18/9/2012 – 22/9/2012.
3. Click on **Transform data** to open the Power Query Editor.
4. Before we do the analysis, change the type of *dteday* to **text**.
5. Click on **Transform** > **Run R script** to create an R script. Copy in the code below (providing the location of the model, in the working directory or elsewhere):

```
bike_model <- readRDS(paste0([insert working directory here],
                              "/bike_model"))
result <- dataset
result$predictions <- predict(bike_model, newdata = dataset)
```

6. Note that an R script can only return a single data frame / query so we will save the predictions in a column in the original query.
7. Change the type of the *dteday* column back to **Date**.
8. Select **Home** > **Close & Apply**.
9.
    a. Create a table and add *dteday, hr* and *predictions*.
    b. Create a line chart. Add *dteday* and *hr* to the axis and *prediction* to **Values**.

5. Where to next?

This course is a starting point for your machine-learning journey. Machine learning is an enormous field and if you use these tools regularly, you are likely to be continually learning.

Depending on your goals, you may need to learn more about:

- data pre-processing and feature extraction,
- other machine learning algorithms,
- the various arguments to the R functions and how to use them to better train or tune your tree
- and other ways of evaluating model performance.

The following resources may be helpful:

R Project website - https://www.r-project.org/help.html

The Comprehensive R Archive Network - https://cran.r-project.org/

Both of these sites contain more detailed documentation for machine-learning functions in R, found under *Packages*. For example, documentation for the *randomForest* package -

https://cran.r-project.org/web/packages/randomForest/randomForest.pdf.

Cheat sheets for machine learning packages available in R (e.g. caret, MLR, Machine Learning Modelling) can be found at https://rstudio.com/resources/cheatsheets/ .