Tutorial Project – Weeks 4 and 5 (Assignment 2 – Parts A & B)

Part A – Data Wrangling and Data Tiding Prior to Instrument Calibration

Objective:

The purpose of this tutorial project is to tidy and wrangle data with methods from the "tidyr" and "dplyr" libraries so that the data can be used by clients, in an easy-to-use form. This project will focus on the methods:

- Filter() to remove missing data, out-of-bounds values and unexpected values
- Mutate() to calculate new variables
- Summarise() to concatenate information regarding replicate measurements
- Sub-sampling and stratification to make balanced distributions
- Re-scaling and other transformations
- Saving data sets in easy to read formats

Data

Sugar factories measure sugar cane juice at the start of the factory process to determine factory settings and to determine the economic value of the supplied sugar cane. To enable real time factory optimisation, a real-time measurement technology called near infrared spectroscopy (NIRS) is used. NIRS analyses the light spectrum that the sugar cane absorbs – the absorbance spectrum is correlated to the chemical composition of the sugar cane. However, the NIRS instruments need to be calibrated to measure specific components of the sugar cane. To do this, traditional laboratory measurements are collected and used to train (calibrate) the NIRS instruments.

The first step in training the NIRS instruments is to prepare laboratory measurements. Because laboratories use multiple assays (different measurement types), measurement information is typically stored in different files or databases. This information needs to be collated, cleaned and appropriately sub-sampled to make training datasets for NIRS instruments.

The laboratory measures are:

- **Pol**: the amount of optical rotation of light due to different sugars in sugar cane (there are three main types of sugar in sugar cane – sucrose, fructose and dextrose)
- **Brix**: the total amount of dissolved solids in the sugar cane juice (sugar is a dissolved solid!)
- **Fibre**: multiple weight measurements are taken to estimate the percentage of fibre (or cellulose) in sugar cane. Fibre is important as it is a key factor in optimising factory settings, like how much energy to use to squeeze out the juice from the sugar cane.

- **Ash**: multiple weight measurements are taken to estimate the percentage ash content. Ash is the amount of dirt that enters with the sugar cane (usually picked up by the sugar cane harvester). It is important to know the ash content as flocculants are added to the squeezed juice to remove it during the refining process (you don't want dirty sugar).

The different laboratory measurements are recorded on different software systems so that there is no single file that contains all the data. Additionally, because of the cost and times needed to make the lab measurement, not all laboratory measurement is taken for each sugar cane sample. Pol and Brix are relatively inexpensive ($25 per sample) and measurements are collected for most samples using the same database software program. Both Fibre and Ash are expensive ($150 p/s) and are taken very infrequently, and are recorded using two separate measurement programs. Consequently, the data is split into three data files. The first contains the Lab ID (a unique number for each sugar cane sample) and the corresponding Pol and Brix measurements. This data file contains many tens of thousands of samples.

The data files for both Fibre and Ash contain the Lab ID and a set of weight measurements that are used to calculate the Fibre and Ash values, respectively. These data files are very small and contain only a few hundred samples.

Tasks

The client requires the data to be cleaned, tidied, transformed, appropriately sub-sampled, then saved in a form that they can use to input into a third-party NIRS training/calibration software (a csv file). Also, the client wants all the data to be saved in a single csv file for future reference. So, in total, there will be five outputted files, a file for each of the four laboratory measurements, which will be used to train (calibrate) the NIRS instrument, and a single file containing all the cleaned and tidied lab data, to be archived.

In this project, we will make the four NIRS training files first, then join all the tidied data to make the final lab file.

Setting up the RStudio environment

We need to set-up a few variables, functions and libraries which will be used throughout the project. We will use the "dplyr", "tidyr" and "ggplot2" libraries. We will also use some custom functions to re-scale and z-transform data. There is a function "**ExpectedBrix()**" which uses the Pol value as the input. This function calculates the expected Brix given a Pol value. A difference between the measured Brix and expected Brix greater than one indicates that there was a problem in collecting either the Brix *or* Pol measurements. Lastly, we need to specify the out-of-bound thresholds for the laboratory measurements.

Table 1 ***Use** this R code to set-up the RStudio environment*

```
#~~~~~~~~~~~~
# Libraries
#~~~~~~~~~~~
library(dplyr)
library(ggplot2)
library(tidyr)

#~~~~~~~~~~~~
# Functions
#~~~~~~~~~~~

rescale_01 <-function(x) (x-min(x))/(max(x)-min(x)) -1/2
z_stand<-function(x) (x-mean(x))/sd(x)
ExpectedBrix <- function(x) (x*0.21084778699754 + 4.28455310831511)

#~~~~~~~~~~~~~
# Thresholds (out of bounds)
#~~~~~~~~~~~~
Thresh.Brix.min <- 15
Thresh.Brix.max <- 30

Thresh.Pol.min <- 50
Thresh.Pol.max <- 105
ExpectedBrix.delta <- 1

Thresh.Fibre.min <- 4
Thresh.Fibre.max <- 25
Thresh.Fibre.delta <- .25

Thresh.Ash.min <- 0
Thresh.Ash.max <- 8
```

Fibre Data

The Fibre data file (Lab_Fibre_Weights.csv) is a CSV file that: (a) contains the variable names in the first row; (b) uses comma (“,”) as field separator character; and (c) uses dot (“.”) as decimal point character.

Table 2 ***Enter** your R code you used to import the Fibre data into a data table called “Lab_Fibre_Data”*

```
#~~~~~~~~~~~~~
# Input file
#~~~~~~~~~~~~
csv <- "Data/Raw1/Lab_Fibre_Weights.csv"
Lab_Fibre_Data <- read.csv(csv, header=T, sep=",", dec=".")
```

The Fibre data file contains the raw weight measurements used to calculate the percentage of fibre in a sugar cane sample, namely:

- SampleWeight, which is the initial tared weight of the sample
- InitialSampleCanWeight, which is the initial weight of the sample in a container
- FinalSampleCanWeight, which is the final weight of the sample in a container

The formula that is used to calculate the percentage fibre from the raw weight measurements is:

Fibre = 100 * (InitialSampleCanWeight – FinalSampleCanWeight) / SampleWeight

To ensure accuracy, raw weights are measured twice (columns 2 to 4 and 5 to 7 of the data) and the corresponding percentage fibre estimates (to be computed using the above equation separately for the two different measurements) can be averaged to provide the final result.

Calculate Percentage Fibre Variables

Table 3 ***Calculate*** *the fibre percentage of the first set of measurements (columns 2 to 4, named SampleWeight_1, InitialSampleCanWeight_1, and FinalSampleCanWeight_1). Name the resulting variable "Fibre1" and add this new variable to the "Lab_Fibre_Data" data table, using direct assignment (i.e. Lab_Fibre_Data$Fibre1 <- …).* ***Enter*** *your R code you used:*

```
#~~~~~~~~~~~~
# Calculate Percentage Fibre Variables
#~~~~~~~~~~~
Lab_Fibre_Data$Fibre1 <- 100 * (Lab_Fibre_Data$InitialSampleCanWeight_1 -
Lab_Fibre_Data$FinalSampleCanWeight_1) / Lab_Fibre_Data$SampleWeight_1
```

Table 4 ***Repeat*** *the above procedure for the second set of measurements (columns 5 to 7, named SampleWeight_2, InitialSampleCanWeight_2, and FinalSampleCanWeight_2), but now using the* ***mutate()*** *function from the dplyr package (rather than direct assignment) to calculate the corresponding fibre percentage, "Fibre2", and add it as a new variable to the "Lab_Fibre_Data" data table.* ***Enter*** *your R code you used:*

```
#~~~~~~~~~~~~
# Calculate Percentage Fibre Variables
#~~~~~~~~~~~
Lab_Fibre_Data <- mutate(Lab_Fibre_Data, Fibre2 = 100 * (InitialSampleCanWeight_2 -
FinalSampleCanWeight_2) / SampleWeight_2)
```

Filtering Fibre Variables

The Fibre data contains missing values, which are recorded **as zeros** in the data.

Table 5 ***Use** the **filter()** function to remove samples (rows) that contain a missing value in* **any** of the weight measurements. Since weights cannot be negative, do that by keeping only the rows that have positive values (> 0) for all the six raw weight measurements. Save the filtered data to a new data table called Lab_Fibre_Filtered. ***Enter** your R code you used:*

```
#~~~~~~~~~~~~
# Filter out values in all columns, except LabID, based on being greater than zero
#~~~~~~~~~~~
Lab_Fibre_Filter <- Lab_Fibre_Data %>%
  filter(SampleWeight_1 > 0, InitialSampleCanWeight_1 > 0, FinalSampleCanWeight_1
> 0, SampleWeight_2 > 0, InitialSampleCanWeight_2 > 0, FinalSampleCanWeight_2 >
0)
```

For the replicate fibre measurements, if there is an absolute difference between the two estimates (computed variables “Fibre1” and “Fibre2”) equal to or greater than 0.25 units, the sample is discarded.

Table 6 ***UPDATE** your code in table 5 to include this maximum fibre difference limit as an <u>additional</u> filtering criteria. **Enter** your R code you used:*

```
#~~~~~~~~~~~~
# Remove any rows that are less than or equal to Thresh.Fibre.delta
#~~~~~~~~~~~
Lab_Fibre_Filter <- Lab_Fibre_Data %>%
  filter(SampleWeight_1 > 0, InitialSampleCanWeight_1 > 0, FinalSampleCanWeight_1
> 0, SampleWeight_2 > 0, InitialSampleCanWeight_2 > 0, FinalSampleCanWeight_2 >
0) %>%
  filter((abs(Fibre1 - Fibre2)) <= Thresh.Fibre.delta)
```

Table 7 ***Calculate** the final fibre estimates by averaging the replicate fibre measurements, “Fibre1” and “Fibre2”. Use **mutate()** to calculate the average fibre and add it as a new variable named “Fibre” to the Lab_Fibre_Filtered data table. **Enter** your R code you used:*

```
#~~~~~~~~~~~~
# Make new column with the average for columns “Fibre1” and “Fibre2
#~~~~~~~~~~~
Lab_Fibre_Filter <- mutate(Lab_Fibre_Filter, Fibre = (Fibre1 + Fibre2)/2)
```

Finally, we need to filter out any out-of-range measurements. The minimum and maximum values are specified in the environmental threshold variables we initially set-up. The threshold values for fibre are: Thresh.Fibre.min and Thresh.Fibre.max.

Table 8 ***Use** a **PIPE to** sequentially filter the measurements in Lab_Fibre_Filtered to remove the out-of-range fibre values, first keeping only the rows of the data table for which "Fibre" is greater than Thresh.Fibre.min, and then keeping only the resulting rows for which "Fibre" is less than Thresh.Fibre.max. Save the resulting data into the Lab_Fibre_Filtered table. **Enter** your R code you used:*

```
#~~~~~~~~~~~~
# Remove Fibre values that are outside min and max range
#~~~~~~~~~~~
Lab_Fibre_Filter <- Lab_Fibre_Filter %>%
  filter(Fibre > Thresh.Fibre.min) %>%
  filter(Fibre < Thresh.Fibre.max)
```

The resulting Lab_Fibre_Filtered data table contains many temporary variables which are no longer needed.

Table 9 ***Use select()** to save the LabID and Fibre variables (1st and last columns) from Lab_Fibre_Filtered to a new data table called Lab_Fibre. **Enter** your R code you used:*

```
#~~~~~~~~~~~~
# Assign two columns "LabID" and "Fibre" to Lab_Fibre
#~~~~~~~~~~~
Lab_Fibre <- Lab_Fibre_Filter %>%
  select(LabID, Fibre)
```

The data table Lab_Fibre contains the cleaned and tidied fibre measurements, alongside with their corresponding Lab IDs. We will need this data table later to make a NIRS calibration file and to join up with the other laboratory data to make a single output file.

Ash Data

The Ash data file (Lab_Ash_Weights.csv) is a CSV file that: (a) contains the variable names in the first row; (b) uses comma (",") as field separator character; and (c) uses dot (".") as decimal point character.

Table 10 ***Enter** your R code you used to import the Ash data into a data table called "Lab_Ash_Data"*

```
#~~~~~~~~~~~~
# Input File
#~~~~~~~~~~~
csv <- "Data/Raw1/Lab_Ash_Weights.csv"
Lab_Ash_Data <- read.csv(csv, header=T, sep=",", dec=".")
```

The Ash data file contains the raw weight measurements used to calculate the percentage of ash in a sugar cane sample, namely:

- TinWeight (column 2)
- InitialSampleInTinWeight (column 3)
- FinalSampleInTinWeight (column 4)

The formula that is used to calculate the percentage ash is:

Ash = 100 * FinalWeight / InitialWeight

where :

InitialWeight = InitialSampleInTinWeight - TinWeight

FinalWeight = FinalSampleInTinWeight - TinWeight

Calculate Ash Variables

The Ash data file contains missing values, which are recorded as zeros in the data. These zero values need to be filtered out before calculating the percentage ash.

Table 11 ***Use*** *a* ***PIPE*** *to (a) first filter out the missing values using filter(), then sequentially calculate (b) "InitialWeight", (c) "FinalWeight" and (d) "Ash" as new variables using mutate(). Save the pipe result to a new data table Lab_Ash_Calculated.* ***Enter*** *your R code you used:*

```
#~~~~~~~~~~~~
# a) For all columns that out rows that are zero or less
# b,c,d) Create three new columns "InitalWeight", "FinalWeight" and "Ash" and
# perform calculations
#~~~~~~~~~~~
Lab_Ash_Calculated <- Lab_Ash_Data %>%
  filter(LabID > 0, TinWeight > 0, InitialSampleInTinWeight > 0, FinalSampleInTinWeight
> 0) %>%
  mutate(InitialWeight  = InitialSampleInTinWeight - TinWeight) %>%
  mutate(FinalWeight  = FinalSampleInTinWeight - TinWeight) %>%
  mutate(Ash = 100 * FinalWeight / InitialWeight)
```

Filtering Ash Variables

We need to filter out any out-of-range measurements. The minimum and maximum value are specified in the environmental variables we initially set-up. The threshold values for ash are: Thresh.Ash.min and Thresh.Ash.max.

Table 12 ***Update*** *your* ***PIPE*** *from table 11 to filter out any out-of-range Ash values. Hint: Add the additional filter at the end of the pipe.* ***Enter*** *your R code you used:*

```
#~~~~~~~~~~~~
# Remove any rows outside the min max threshold for "Ash"
#~~~~~~~~~~~
  filter(Ash > Thresh.Ash.min) %>%
  filter(Ash < Thresh.Ash.max)
```

Summarising Ash Variables

To ensure accuracy, ash is also measured twice, and the two measurements are supposed to be averaged to provide the final result. However, unlike the fibre data, replicates occur as two separate rows (rather than columns) in the data, with the same Lab ID. We need to summarise the replicate values to a single, averaged value.

Table 13 ***Use a PIPE*** *with the* ***grouped_by()*** *and* ***summarize()*** *functions to produce a data table called Lab_Ash that is grouped by LabID and summarises variable Ash by taking its grouped mean values. The resulting table, Lab_Ash, must then have two variables, LabID and Ash, where LabID now contains unique values (no replicates).* ***Enter*** *your R code you used:*

```
#~~~~~~~~~~~~
# Group all calculation of mean by “LabID”
#~~~~~~~~~~~
Lab_Ash <- Lab_Ash_Calculated %>%
  group_by(LabID) %>%
  summarise(Ash = mean(Ash)) %>%
  select(LabID, Ash)
```

The data table Lab_Ash contains the cleaned and tidied ash measurements. We will need this data table later to make a NIRS calibration file and to join up with the other laboratory data to make a single output file.

Pol and Brix Data

The Pol and Brix data file (Lab_Pol_Brix.csv) is a CSV file that: (a) contains the variable names in the first row; (b) uses comma (“,”) as field separator character; and (c) uses dot (“.”) as decimal point character.

Table 14 ***Enter*** *your R code you used to import the Pol and Brix data into a data table called “Lab_PB_Data”*

```
#~~~~~~~~~~~~
# Input File
#~~~~~~~~~~~
csv <- "Data/Raw1/Lab_Pol_Brix.csv"
Lab_PB_Data <- read.csv(csv, header=T, sep=",", dec=".")
```

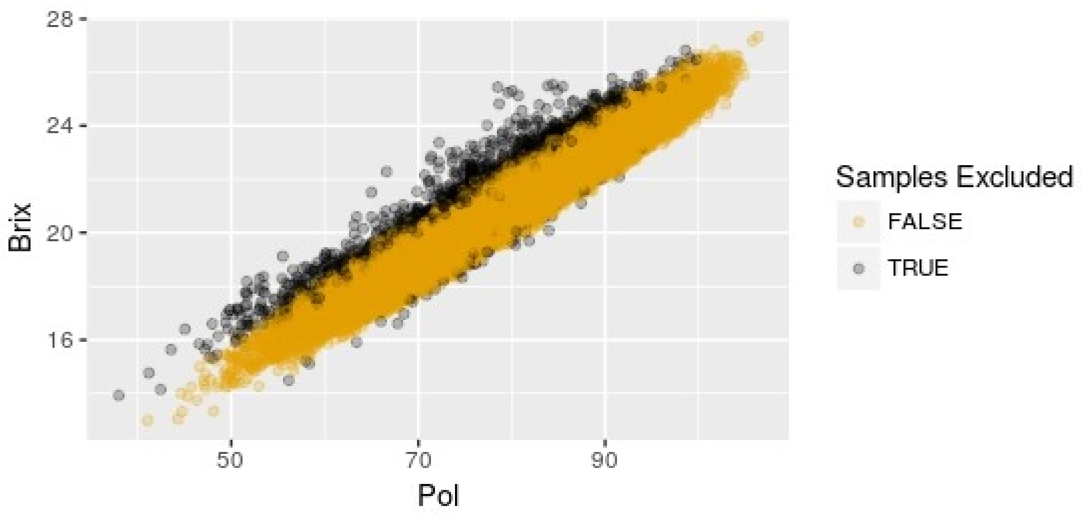
The Pol and Brix dataset does not contain any missing values because the database that exported the records had already handled missing values. However, the dataset does contain anomalous measurements. There is a function “ExpectedBrix” which uses the Pol value as the input. This function calculates the expected Brix given a Pol value. A difference between the measured Brix and expected Brix greater than one indicates that there was a problem in collecting either the Brix *or* Pol measurements.

Table 15 ***Use** the **mutate()** function to add to the data table Lab_PB_Data a new variable, PredBrix, which uses the ExpectedBrix function with Pol as input. **Enter** your R code you used:*

```
#~~~~~~~~~~~~
# Make new column called PredBrix that holds the value calculated using the functions
# ExpectedBrix() sourced from assess_func_lib.R
#~~~~~~~~~~~
source("R/assess_func_lib.R")
Lab_PB_Data <- Lab_PB_Data %>%
  mutate(PredBrix = ExpectedBrix(Pol))
```

Table 16 ***Use ggplot()** to visually display a scatter (i.e. point) plot of variables Brix and Pol. Colour code the points according to if the absolute difference between the measured Brix (variable Brix, 3rd column) and the predicted Brix (variable PredBrix, 4th column) is greater than one. **Enter** your R code **and** the jpeg of the plot:*

```
#~~~~~~~~~~~~
# Graph a scatter plot for Pol and Brix that shows the included/excluded Pol and Brix
# based on the calculation |Brix-Pred| > 1.
#~~~~~~~~~~~
PB_Scatter_Map <- ggplot(Lab_PB_Data, aes(x=Pol, y=Brix, colour=(abs(Brix-
PredBrix) > ExpectedBrix.delta))) +
  scale_colour_manual(name = 'Samples Excluded', values =
setNames(c("#000000","#E69F00"),c(T, F)))
PB_Scatter_Map  + geom_point(shape=19,
position=position_jitter(width=1,height=.5), alpha=0.25)
```



We have used PIPES to filter data. Now we will use a PIPE to filter data and select variables.

Table 17 ***Use** a **PIPE** to sequentially filter out undesirable rows from Lab_PB_Data, and then select only a subset of its variables, as follows: first, filter out samples (rows) where (a) the absolute difference between the measured Brix and predicted Brix is greater than one, and/or (b) any value for Pol or Brix are out of range (the min. and max. values are specified in the threshold variables we initially set-up, namely, Thresh.Brix.min, Thresh.Brix.max, Thresh.Pol.min, and Thresh.Pol.max). Then, (c) select only the variables LabID, Pol and Brix to constitute a new data table, called Lab_PB. **Enter** your R code you used:*

```
 #~~~~~~~~~~~~
 # Remove all rows that are out of ranges
 # absolute values of Brix – PredBrix is Less than 1
 # Min and Max for Brix and Pol
 # Make new table with columns “LabID”,“Pol” and “Brix”
 #~~~~~~~~~~~
Lab_PB <- Lab_PB_Data %>%
 filter((abs(Brix-PredBrix) < ExpectedBrix.delta)) %>%
 filter(Pol > Thresh.Pol.min) %>%
 filter(Pol < Thresh.Pol.max) %>%
 filter(Brix > Thresh.Brix.min) %>%
 filter(Brix < Thresh.Brix.max) %>%
 select(LabID, Pol, Brix)
```

The data table Lab_PB contains the cleaned and tidied Pol and Brix measurements. We will need this data table later to make a NIRS calibration file and to join up with the other laboratory data to make a single output file.

A Single Lab File

We have tidied and cleaned laboratory data for the Fibre, Ash and Pol and Brix samples in the three data tables Lab_Fibre, Lab_Ash and Lab_PB respectively. The common variable that links all the samples together is LabID. To join tables together using a common key variable, we can use the **full_join()** function from the **dplyr** library.

Table 18 ***Use** the following R code to join the Fibre and Ash tables together.*

```
Lab <- full_join(Lab_Ash, Lab_Fibre, by=c("LabID" = "LabID"))
```

The last clause in the above full_join() statement uses the input parameter by=c(“LabID” = “LabID”). This parameter specifies which variable names to link the samples (rows) to between the two data tables. Also, because not every LabID has an Ash and a Fibre measurement (i.e., some Lab IDs in one file may not be present in the other file and vice-versa), NA (missing values) are inserted where the sample does not have a respective Ash or Fibre measurement.

Now, let’s add the Pol and Brix data to the Lab data table

Table 19 ***Join*** *the existing Lab data table to the Lab_PB data table.* ***Enter*** *you R code* ***and*** *the first eleven rows of the combined Lab data table. Hint: Use Lab[1:11,] to display the top 11 rows:*

```
#~~~~~~~~~~~~
# Unsure why the output is different to Lab_Out.csv file given to use.  Essentially it is
# the Lab_Out minus the NA's rows
#~~~~~~~~~~~
Lab <- full_join(Lab, Lab_PB, by=c("LabID" = "LabID"))
Lab[1:11, ]
> Lab[1:11, ]
   LabID       Ash    Fibre    Pol   Brix
1     27 2.3355689 15.57110 78.776 20.759
2    448 4.4466623 14.40024 65.056 18.108
3   1041 3.7434827 14.80769 79.970 20.881
4   1059 1.6638742 15.09137 81.652 21.519
5   1183 2.7893071 17.80312 86.610 22.838
6   2080 2.2814858       NA 71.285 19.091
7   2205 2.1112700 13.87420 79.269 21.254
8   2812 0.9625988 17.18891 86.378 23.225
9   3288 1.0409728 15.13368 75.734 20.560
10  3916 2.3825928 14.77926 78.043 20.998
11  4047 0.8751971 19.72404 87.263 22.214
```

To save the Lab data table, we can use the **write.table()** function in the R base library.

Table 20 ***Use*** *the following R code to save the Lab data table to disk*

```
 #~~~~~~~~~~~~
 # Lab_Out saved to disk for further use later
 #~~~~~~~~~~~
write.table(Lab, file = "Lab_Out.csv", append = FALSE, quote = TRUE,
sep = ",",
            eol = "\n", na = "NA", dec = ".", row.names = FALSE, col.names
= TRUE, qmethod = c("escape", "double"), fileEncoding = "")
```

Making Calibration files

Because the Lab_Fibre and Lab_Ash data tables have only a few hundred samples, the calibration data files will contain all of the tidied and cleaned samples. However, to help the user with the third-party NIRS training (calibration) software, both the Fibre and Ash measurements need to be transformed before saving to file.

The Fibre values need to be transformed (rescaled) using the z_stand() function provided in the environment set-up, while the Ash values need to be log transformed and then rescaled using the z_stand() function.

Table 21 ***Use*** *transform() to transform the Fibre measurements from the Lab_Fibre table using the provided z_stand() function. Then* ***save*** *the resulting table (containing variables LabID and Fibre) to a file on disk.* ***Enter*** *the R code you use to perform both actions.*

```
 #~~~~~~~~~~~~
 # Load source file assess_func_lib.R which contains functions needed for calculations
 #~~~~~~~~~~~
source("R/assess_func_lib.R")
 #~~~~~~~~~~~~
 # Use z_stand() from assess_func_lib.R to produce values for “File_Fibre_Out”
 #~~~~~~~~~~~
File_Fibre_Out <- transform(Lab_Fibre, Fibre = z_stand(Fibre))
 #~~~~~~~~~~~~
 # Output to disk “File_Fibre_Out” for later use
 #~~~~~~~~~~~
write.table(File_Fibre_Out, file = "output/File_Fibre_Out.csv", append = FALSE, quote =
TRUE, sep = ",",
            eol = "\n", na = "NA", dec = ".", row.names = FALSE, col.names = TRUE,
            qmethod = c("escape", "double"), fileEncoding = "")
```

Table 22 ***Use a pipe*** *with two subsequent transform() operations to transform the Ash measurements from the Lab_Ash table, first using log10(), and then using z_stand().* ***Save*** *the resulting table (containing variables LabID and Ash) to a file on disk.* ***Enter*** *the R code you use to perform both actions.*

```
 #~~~~~~~~~~~~
 # Load source file assess_func_lib.R which contains functions needed for
 calculations
 #~~~~~~~~~~~
source("R/assess_func_lib.R")
 #~~~~~~~~~~~~
 # Use z_stand() from assess_func_lib.R and log10 from R Base to produce values for
 # “File_Ash_Out”
 #~~~~~~~~~~~
File_Ash_Out <- Lab_Ash %>%
  transform(Ash = log10(Ash)) %>%
  transform(Ash = z_stand(Ash))
 #~~~~~~~~~~~~
 # Output to disk “File_Ash_Out” for later use
 #~~~~~~~~~~~
write.table(File_Ash_Out, file = "output/File_Ash_Out.csv", append = FALSE, quote =
TRUE, sep = ",",
            eol = "\n", na = "NA", dec = ".", row.names = FALSE, col.names = TRUE,
            qmethod = c("escape", "double"), fileEncoding = "")
```

The Pol and Brix samples are a bit tricker because the third-party software is limited to use no more than 2000 samples. To produce calibration files for Brix and Pol, we will need to sub-sample. Additionally, the third-party software produces optimised calibrations when the calibration data has a box distribution. Therefore, to make the calibration files, we need to perform stratified sub-sampling to (a) limit the number of samples and (b) to produce a box distribution.

However, the Lab_PB data does not have an existing variable to make stratified sub-samples; we have to make one ourselves. The base library function cut(x) divides the range of x into intervals and codes the values in x according to which interval they fall. The leftmost interval corresponds to level one, the next leftmost to level two and so on.

Table 23 ***Use*** *the following R code to create a variable which can be subsequently used for stratified sub-sampling:*

```
#~~~~~~~~~~
# Function cut() divides the rows containing Brix values into 40
# intervals.  The interval numbers are stored in column Bbin. This allows
# the user to stratify by the Bbin number.
#~~~~~~~~~
Lab_PB$Bbin <- cut(Lab_PB$Brix, 40, labels = FALSE)
```

This creates an auxiliary variable, Bbin, that splits the range of Brix values in the Lab_PB table into forty sections. The resulting Bbin variable is an integer variable and, as such, it cannot be readily used for stratification because the stratification variable needs to be an ordinal variable for grouping. To change the variable type, we will re-cast the integer variable into an ordinal factor variable using the as.factor() function

Table 24 ***Use*** *the following R code to re-cast the Bbin variable as ordinal, so it can be used for stratified sub-sampling:*

```
#~~~~~~~~~~
# Stratification can not work with integers so the values in Bbin are
# change to factor
#~~~~~~~~~
Lab_PB$Bbin <- as.factor(Lab_PB$Bbin)
```

Now Bbin can be used as a grouping variable for sub-sampling.

Table 25 ***Use*** *the group_by() function then the sample_n() function to perform stratified sampling on the Brix meansurements, using Bbin as the grouping variable and size=50 for the number of samples in each stratification.* ***Name*** *the resulting data table as Lab_B_Stratified_Balanced. Note: sample_n() will need to use attribute replace = TRUE, as not all groups have fifty samples.* ***Enter*** *the R code you use to perform both actions.*

```
#~~~~~~~~~~~
# Group the Lab_PB table by newly created column Bbin and stratify sample by size
# 50
#~~~~~~~~~~
```

```
Lab_B_Stratified_Balanced <- Lab_PB %>%
  group_by(Bbin) %>%
  sample_n(size = 50, replace = TRUE)
```

Now that we have a stratified sub-sample, to make the clients work easier, the stratified sub-sample Brix measurements need to be rescaled using the rescale_01() function provided in the environment set-up. Then, we need to select the LabID and Brix variables form the stratified sub-sampled data table, so we can write them as a separate data table to a csv file.

Table 26 ***Use*** *transform() with rescale_01() to rescale the Brix measurements in Lab_B_Stratified_Balanced. Then,* ***use*** *select() to select only the LabID and Brix variables, and* ***write*** *the resulting data table to a csv file.* ***Enter*** *the R code you use to perform the three actions.*

```
  #~~~~~~~~~~~~
  # Load source file assess_func_lib.R which contains functions needed for calculations
  #~~~~~~~~~~~
source("R/assess_func_lib.R")
  #~~~~~~~~~~~~
  # Use function rescale_01 from assess_func_lib.R to produce the new value of Brix
  # Select only columns “LabID” and “Brix” to be assigned to “File_Brix_Out”
  #~~~~~~~~~~~
File_Brix_Out <- Lab_B_Stratified_Balanced %>%
  transform(Brix = rescale_01(Brix)) %>%
  select(LabID, Brix)
  #~~~~~~~~~~~~
  # Output to disk “File_Brix_Out.csv”
  #~~~~~~~~~~~
write.table(File_Brix_Out, file = "output/File_Brix_Out.csv", append = FALSE, quote = 
TRUE, sep = ",",
            eol = "\n", na = "NA", dec = ".", row.names = FALSE, col.names = TRUE,
            qmethod = c("escape", "double"), fileEncoding = "")
```

We can repeat the method used to make the Brix stratified sub-sample for the Pol measurement, but stream line it by using a pipe.

Table 27 ***Use*** *a pipe to repeat the stratified sub-sampling method used for Brix, but now for Pol. Then write the LabID and the (stratified, sub-sampled, rescaled) Pol vales to a file.* ***Enter*** *the R code you use to perform both actions.*

```
  #~~~~~~~~~~~~
  # Load source file assess_func_lib.R which contains functions needed for calculations
  #~~~~~~~~~~~
source("R/assess_func_lib.R")
  #~~~~~~~~~~~~
  # Use function rescale_01 from assess_func_lib.R to produce the new value of Brix
  # Select only columns “LabID” and “Pol” to be assigned to “File_Pol_Out”
  #~~~~~~~~~~
File_Pol_Out <- Lab_B_Stratified_Balanced %>%
  transform(Pol = rescale_01(Pol)) %>%
```

```
  select(LabID, Pol)
  #~~~~~~~~~~~~~
  # Output to disk
  #~~~~~~~~~~~~
write.table(File_Pol_Out, file = "output/File_Pol_Out.csv", append = FALSE, quote =
TRUE, sep = ",",
            eol = "\n", na = "NA", dec = ".", row.names = FALSE, col.names = TRUE,
            qmethod = c("escape", "double"), fileEncoding = "")
```

Part B – Processing NIRS Predictions

In part A of this assignment, we processed raw laboratory data and produced a number of files to be used as input into a third-party calibration software than trains NIRS instruments to make accurate measurements of sugar cane samples. Once an NIRS instrument has been calibrated with our files using such a third-party software, the instrument can then be used to make measurements of new sugar cane samples. These measurements are initially stored into a raw data file, which also needs to be processed for further analysis.

Data

The set of NIRS measurements are: Pol, Brix, Fibre and Ash. Because NIRS is fast, multiple NIRS measurement sets are taken per sugar cane sample. However, because it is fast, not all measurements are high quality. NIRS measurements need to be screened for quality then aggregated before it is used by other controlling systems for factory control.

For each NIRS measurement, a set of distance values, GH (Global neighbourhood) and NH (Nearest Neighbourhood), are calculated and used for quality control. GH and NH values less than 3.5 and 2, respectively, are high quality measurements. If either the GH or NH are greater than 3.5 and 2, respectively, then the NIRS measurement cannot be trusted, and is subsequently discarded.

Task

The client requires us to clean and tidy the NIRS data, specifically, filter out any low quality measurements and any out-of-range values. Secondly, the client needs the cleaned data to be averaged according to the sample ID. Then, lastly, they require a single csv file containing the averaged, cleaned data.

First, we need to set-up a few variables and libraries which will be used throughout the task. We will use the “dplyr”, “tidyr” and “ggplot2” libraries

Table 28 ***Use*** *this R code to set-up the RStudio environment*

```
#~~~~~~~~~~~~~
# Libraries
#~~~~~~~~~~~~
library(dplyr)
```

```
library(ggplot2)
library(tidyr)

#~~~~~~~~~~~~~~
# Thresholds
#~~~~~~~~~~~~~
Thresh.Brix.min <- 15
Thresh.Brix.max <- 30

Thresh.Pol.min <- 50
Thresh.Pol.max <- 105

Thresh.Fibre.min <- 4
Thresh.Fibre.max <- 25

Thresh.Ash.min <- 0
Thresh.Ash.max <- 8
```

Now we need to import the NIRS data (NIRPred.csv), a CSV file with variable names in the first row, comma (",") as field separator character, and dot (".") as decimal point character.

Table 29 ***Enter*** *your R code you used to import the NIR data into a data table called "NIRData"*

Enter your answer here

Look at the summary of the NIRData table using the summary() function and look at the first fifteen rows of this table typing NIRData[1:15,]. You will see a few things. Firstly, from the summary, we see that there are measurements that are out-of-range (according to the min. and max. thresholds provided above) and that there are ScanID values of -1. The ScanID of -1 is used in the NIRS system when the sample number is not recorded properly. In these cases, the scan (row) needs to be discarded.

Secondly, from the summary, there is a variable called DateTime, which refers to the date and time which the NIRS scanned the sample. However, the DateTime variable was automatically assigned as a factor variable when inputted using the read.table() function above. This is not correct as it makes it extremely hard to do anything with it. So, to get this DateTime variable into the correct format and data type, we will re-cast it as a POSIXct data type. The POSIXct data type is the way that RStudio handles date time variables.

Table 30 ***Use*** *the following R code to correctly assign the DateTime variable to the POSIXct data type. Note that the POSIXct uses input arguments that specify the format which our DateTime uses.*

```
NIRData$DateTime <- as.POSIXct(NIRData$DateTime, format = "%Y-%m-%d
%H:%M:%S")
```

Lastly, when looking at the first fifteen sample rows in the data table, the ScanID refers to the LabID with a decimal suffix (the two digit fractional part) that is the scan number for that LabID. In order to extract the LabID by itself, so we can use it later for grouping, we need to round ScanID down to the lowest integer, which can be achieved with the floor() function in the R base library.

Table 31 ***Use transform()*** *with the floor() function applied to ScanID to create a new variable called LabID in the NIRData table.* ***Enter*** *your R code you used to create the new variable,* ***then*** *enter the first fifteen rows of the updated NIRData table.*

Enter your answer here

In Part A of this assignment, we looked at using pipes to filter out samples with unexpected values and samples with out-of-range measurements. We will use the same methods using a single pipe to filter the NIR data.

Table 32 ***Use*** *a* ***PIPE*** *to sequentially filter the NIR data by filtering out any (a) GH values greater than 3.5, (b) NH values greater than 2, (c) any out-of-range values for Pol, Brix, Fibre and Ash and (d) any sample that has a ScanID equal to -1. Save the filtered data to a new data table called NIRData_Filtered.* ***Enter*** *your R code:*

Enter your answer here

Now we have a cleaned data set, where there are multiple rows per LabID, a similar case to the Ash data in Part A. We will now finish the tidying by using another pipe to group, summarise and select the NIR data so that we have one averaged result per LabID for each Pol, Brix, Fibre and Ash measurement type. Note, we don’t need to save the GH or NH values, but we do need the DateTime of the *first* time the LabID was scanned with the NIR.

Table 33 ***Use a PIPE*** *with the* ***grouped_by()*** *and* ***summarize()*** *functions on the NIRData_Filtered table to produce a data table called NIR_Final which is grouped by LabID and contains, in addition to the grouped variable, the first DateTime for each group as well as the corresponding mean values for Pol, Brix, Fibre and Ash (i.e. the group means). Hint: the min() function returns the earliest date/time when applied to a date/time type variable.* ***Enter*** *your R code you used* ***then*** *enter the first fifteen rows of the updated NIR_Final table:*

Enter your answer here

Note, because we have pipes that filter and summarise data, we could have piped both pipes together to make a single pipe.

Table 34 ***Discuss*** *the advantages of combining PIPES when handling very large datasets*

Enter your answer here

Finally, save the NIR_Final data table to a csv file.

Table 35 ***Enter*** *your R code to save the NIR_Final data table to a csv file on disk.*

Enter your answer here