Aplicación web creada con shiny para el análisis de la variabilidad de las materias primas en la fabricación de piensos

TRABAJO DE FINAL DE GRADO

AUTOR: Joaquín Casino Durán TUTORES: Mercè Farré, David Solà

Grado en Estadística Aplicada

30 de junio de 2017

RESUMEN

La variabilidad en las materias primas puede comprometer la calidad del producto final. En la fabricación de piensos en el sector porcino, este factor es determinante para la producción animal. Se realiza un proceso de manipulación de las bases de datos de los ingredientes principales de este pienso (trigo, soja, maíz y cebada) con la finalidad de conseguir una sola base de datos de todos los ingredientes y, todos los nutrientes y aminoácidos de estos. Mediante un shinydashboard *se crea un aplicativo para automatizar este proceso, generado en R, y con la posibilidad de tener un control de calidad sobre diferentes aminoácidos.*
Palabras clave: ingredientes, bases de datos, R, shinydashboard, algorítmo

La variabilitat en la matèries primeres pot comprometre la qualitat del producte final. En la fabricació del pinso en el sector porcí, aquest factor es determinant per la producció animal. Es realitza un procès de manipulació de les bases de dades dels ingredients principals d'aquest pinso (blat, soja, blat de moro i ordi) amb la finalitat d'aconseguir una base sola base de tots els ingredients, tots els seus nutrients i aminoàcids. Mitjançant un shinydashboard *es crea un aplicatiu per automatitzar aquest procès, generat amb R, i amb la posibilitat de tenir un control de qualitat sobre diferents aminoàcids.*
Paraules clau: ingredients, bases de dades, R, shinydhasboard, algorítme

Variability in raw materia can compromise the quality in the final product. In feed manufacture in the pig sector, this is a determining factor for animal manufacture. Handling process of the database was done over the main ingredients of the feed (wheat, soybean, corn and barley) in order to obtain a final database of all the ingredients, all their nutrients and amino acids. By means of a shinydashboard*, an app was made in order to automate the process,done with R software, with the possibility of exploring a quality control over differents amino acids.*
Key words: ingredients, database, R, shinydhasbord, algorithm

Índice

1. Introducción

Como en muchos procesos industriales, la variabilidad de las materias primas puede comprometer seriamente la calidad del producto final. En la fabricación de piensos, este problema puede tener consecuencias aún más importantes ya que, el producto, es uno de los factores determinantes para el correcto desarrollo de la producción animal. La variabilidad en la composición química de los nutrientes (proteína, energía, lisina, fibra...) de las materias primas (cereales entre ellas) es un factor limitante de la eficiencia global del proceso productivo.
Habitualmente, en el sector porcino, la estructura productiva es de integración vertical: todo el tejido encadenante esta dentro de una misma pirámide productiva, desde la fábrica de piensos hasta la elaboración de la carne. Por lo tanto, se puede incidir en diferentes estratos del ciclo productivo para mejorar la eficiencia global.

Las implicaciones en cadena son complejas de analizar para después decidir como optimizar (o mejorar significativamente) la eficiencia del proceso y garantizar unos estándares de calidad. En este sentido, los modelos estocásticos y la simulación proporcionan un marco apropiado para abordar el problema.
Recientemente (*Fabà et al., 2016*), se ha probado mediante simulación que, debido a la variabilidad de las concentraciones de nutrientes, determinadas composiciones químicas de las materias primas dan un pienso final por debajo de los niveles de calidad establecidos con *probabilidades no-negligentes*.

La dieta de los animales se fija con una fórmula óptima, és decir unos porcentajes de las materias primas $p_i, \ldots, p_k$ de manera que, en base a los nutrientes de las materias prima y al precio de estas, garantiza ciertas restricciones nutricionales y minimiza el precio del pienso resultante. La composición nutricional del pienso así formulado es, para cada nutriente y,

$$y = \sum_{i=1}^{k} w_i y_i \tag{1}$$

donde y_i es la cantidad de nutriente que aporta la materia prima i-ésima.

Llamamos formulación estática la determinación de la dieta en base a un valor nutricional nominal o teórico de los diversos ingredientes que lo componen. Las diferencias entre los valores nutricionales nominales y los valores reales pueden producir disfunciones en el producto final. Hoy en día, la concentración real de nutrientes se puede obtener con elevada precisión con la tecnología NIR (*Near Infra Red*) aplicada al análisis de las diversas materias primas. En la praxis, se suele realizar una formulación, que denominaremos semi-estática, de la dieta en base a los valores NIR de las últimas partidas de materias primas. Esta formulación se hace con una determinada periodicidad (mensual) que no se ajusta al ritmo de cambio de las materias primas las cuales se renuevan con una frecuencia mucho más alta y, por lo tanto, no puede garantizar completamente que los valores nutricionales del pienso fabricado cumplan con los estándares de calidad. Finalmente, una formulación dinámica se haría a tiempo real en el momento de fabricar el pienso, en base al análisis NIR de los ingredientes que se mezclarán en este instante. Este tipo de formulación garantizaría efectivamente los estándares de calidad a la vez que minimizaría el precio. Se dispone de datos de valores nutricionales de materias primas (trigo, maíz, soja y cebada) observados a lo largo de un año (2016, fuente SNIBA [1]). El objetivo es analizar las eventuales deficiencias en el grado de cumplimiento de los estándares de calidad cuando la formulación es estática. Un segundo objetivo, que no se incluye en este trabajo, seria la formulación dinámica, para lo cual ya se ha preparado la base de datos.

1.1. Base de datos

Como se ha expuesto, los datos utilizados serán los ingredientes principales en la fabricación de piensos para el sector porcino, estos ingredientes, separados en diversas bases de datos, tendrán añadidos algunos de sus componentes nutricionales y precio. Los datos disponibles para los 4 ingredientes principales(trigo, soja, cebada y maiz)son el número de muestra, desde cuando y hasta cuando se utilizaron y algunos de los nutrientes de estos. Como ejemplo, en la siguiente tabla del ingrediente `Trigo` se observa la estructura de los datos.

[1] Servei de Nutrició i Benestar Animal, Campus de Bellaterra. Datos, proporcionados por David Solà Oriol, y procedentes de una empresa ganadera.

	Muestra	DESDE	HASTA	ORIGENANALISIS	Humedad	Proteína_K	Ceniza	Almidon
1	16000389	4-1	4-1	FABRICA	11.10	10.60	1.50	60.00
2	16000390	4-1	4-1	FABRICA	11.20	11.30	1.50	59.70
3	16000388	4-1	4-1	FABRICA	11.00	10.70	1.50	60.00
4	16000391	4-1	4-1	FABRICA	11.70	11.50	1.50	59.60
5	16000392	4-1	4-1	FABRICA	11.40	11.60	1.50	59.60

Cuadro 1: Base de datos trigo, muestra de cómo son los datos.

El total de componentes nutricionales principales son: humedad, ceniza, almidón, grasa, proteína y fibra. Estos componentes nutricionales serán los que se encuentren de forma natural en la BBDD, pero aún quedaría pendiente de añadir todos los aminoácidos disponibles en estos nutrientes como la lisina, o los azucares que contienen.

En la tabla anterior, no se dispone de muchos de los nutrientes básicos ni de ningún aminoácido. Éste es el principal problema estructural de los datos. Se tratará este problema en la sección *procesamiento común*. Las variables de fecha, procedéncia y número de identificación se encuentran en todas las bases de datos.

Para el procesamiento de estos datos, la base de datos requerida debe recoger la información de las bases de datos de cada ingrediente. Así que para este procedimiento será indispensable la creación de un algorítmo capaz de alinear estas BBDD, sin la ayuda de ninguna variable que nos permita alinearlas de una manera natural.

Antes de poder alinear las BBDD con el fin de obtener un único fichero resultante, es indispensable que todas las bases de datos (trigo, cebada, maiz y soja) tengan los nutrientes principales. En caso de que no obtengamos de manera natural esta variable, se procederá a la imputación mediante una matriz *FEDNA*, que asigna a cada ingrediente valores teóricos para cada nutriente. Una vez todas las variables principales estén asignadas, una serie de regresiones permitirán obtener los aminoácidos y los precios.

1.2. Lenguaje utilizado

Para todo el procesamiento de datos, el lenguaje utilizado será `R`.

Dentro de este lenguaje, el código será una mezcla entre el lenguaje base de R para aplicarlo en funciones como lapply, bucles for, asignación de integers, entre otros. Y lenguaje `data.table`, una librería de R con lenguaje propio que optimiza el lenguaje bajando a capas de *C++*, la cual proporciona una manera más rápida de leer y manipular bases de datos. Para la herramienta *shiny*, se utiliza de manera marginal *html*.

2. Procesamiento de los datos

La preparación de los datos es el principal problema de este trabajo. Esta complejidad converge en la cantidad de variables faltantes e imposibilidad de juntar bases de datos basándonos en una variable de "merge".Para este procedimiento existen en dos métodos:

- **Asignación aleatoria**: Donde se simularan los valores faltantes mediante unas condiciones dinámicas.
- **Asignación estática**: Donde se asignarán estos valores mediante un valor fijo o nominal.

La asignación aleatoria presenta una mayor complejidad asumiendo una variabilidad mayor. Sin embargo, la asignación de valores para los datos faltantes de manera teórica, reduce la complejidad y a su vez la variabilidad . En este trabajo haremos especial incapié en la asignación aleatória y en el algorítmo para aplicarla.

2.1. Procesamiento común

Para un análisis es fundamental tener los datos limpios y preparados, de otra manera los resultados pueden no ser los deseados. En este apartado, se describirán todos los puntos y decisiones tomadas para preparar los datos. Una vez los datos estén preparados, se implementará el algorítmo que permite juntar los diferentes `data.frame` de manera aleatoria o estática.

Excel, formato original en el que los datos son compartidos, suele cambiar el formato de las variables una vez se abre el fichero. Por ello el primer punto a tener en cuenta es establecer todas las variables numéricas como tales con `as.numeric(X)`.
El siguiente punto será ordenar las BBDD mediante *DESDE* y *HASTA* y cambiar los nombres por defecto de las variables por un nombre mas cómodo. Ej : *Proteína K - proteina*.

En los datos facilitados por SNIBA, se encuentra la matriz FEDNA. En esta matriz encontramos los valores teóricos para los diferentes nutrientes y aminoácidos, en el caso de no obtenerlos con la tecnología NIR. Esta matriz cobra mucha importancia cuando se imputan valores de ingredientes de los cuales no se dispone información (manteca, sal, fosfato bicalcico...), y, a su vez, cuando se ejecuta la asignación estática de información faltante. Se recrea esta matriz en forma de lista en R para obtener los valores de los siguientes nutrientes:

- Humedad
- Ceniza
- Proteína
- Grasa
- Fibra
- Almidón
- Azúcares

De esta manera, se podrán hallar automáticamente qué variables faltan en cada base de datos:

```
f_cebadaIn <- fedna[[1]][,c(which(!(names(fedna[[1]]) %in% names(cebada))))]
f_maizIn   <- fedna[[2]][,c(which(!(names(fedna[[2]]) %in% names(maiz))))]
f_trigoIn  <- fedna[[3]][,c(which(!(names(fedna[[3]]) %in% names(trigo))))]
f_sojaIn   <- fedna[[4]][,c(which(!(names(fedna[[4]]) %in% names(soja))))]
```

Así para cada ingrediente, se podrán encontrar todas las variables relacionadas con los nutrientes base anteriormente enunciados. En el momento de que las 4 bases de datos se encuentren completas con las variables enunciadas, el orden en que encontraremos estas variables en cada base de datos tendrá que ser el mismo.

El siguiente punto, será añadir las cantidades de los diversos aminoácidos y, las cuales se pueden calcular a partir de ecuaciones de regresión lineal simple documentació de SNIBA) a partir de la cantidad de nitrógeno N, que a su vez, se obtiene como la proteína bruta dividida por 6.25:

$$y = (aN + b)/1000 = (a\frac{PB}{6{,}25} + b)/1000 \tag{2}$$

Para cada ingrediente, los coeficientes a y b de la recta de de regresión són diferentes, y se pueden ver en la siguiente tabla:

	Trigo			Cebada		
Aminoacido	a	b	Disponible	a	b	Disponible
Lys	129	86	83.33 %	130	160	76.32 %
Thr	156	36	84.38 %	154	87	74.29 %
Met	85	16	88.24 %	75	41	88.24 %
Cys	117	26	92.31 %	90	62	86.96 %
Trp	61	36	84.62 %	66	21	83.33 %
Val	229	55	85.11 %	234	110	80.77 %
Ile	204	0	89.47 %	211	0	80.56 %
Leu	385	34	90.14 %	413	0	83.82 %
Arg	245	82	88.68 %	231	107	81.63 %
Phe	297	-42	91.84 %	359	-89	85.71 %
Tyr	162	0	90.91 %	169	0	82.76 %
His	137	0	91.67 %	111	35	78.26 %
Ser	278	0	88.46 %	221	60	80.95 %
Ala	182	68	79.49 %	166	143	71.43 %
Asp	243	124	81.82 %	266	162	75 %
Glu	1980	-538	94.98 %	1698	-532	88.31 %
Gly	225	46	86.05 %	160	148	75.61 %
Pro	682	-242	95.19 %	807	-312	84.40 %

	Maiz			Soja		
Aminoacido	a	b	Disponible	a	b	Disponible
Lys	183	0	79.17 %	379	0	90.19 %
Thr	220	0	83.33 %	241	0	85.80 %
Met	81	50	94.12 %	82	0	91.94 %
Cys	86	55	90 %	86	0	86.15 %
Trp	16	40	80 %	84	0	89.28 %
Val	293	0	85.37 %	296	0	87.98 %
Ile	210	0	90 %	282	0	88.94 %
Leu	954	-266	93.14 %	473	0	89.03 %
Arg	134	173	92.11 %	447	0	93.77 %
Phe	297	0	90 %	316	0	90.32 %
Tyr	209	0	88.57 %	231	0	91.03 %
His	174	0	91.30 %	157	0	90.43 %
Ser	292	0	90.24 %	309	0	88.99 %
Ala	455	0	90.16 %	269	0	85.79 %
Asp	268	169	86.79 %	786	-631	88.98 %
Glu	1104	0	92.86 %	1103	0	90.01 %
Gly	129	132	83.87 %	218	325	83.98 %
Pro	722	-237	89.33 %	303	0	78.70 %

Cuadro 2: Tabla de coeficientes para la obtenci'on de aminoácidos totales y disponibles

Los aminoácidos disponibles se calcularán mediante la multiplicación del aminoácido resultante por la variable disponible. El cálculo total de aminoácidos se realiza con la siguiente función.

```
aminoacid_ing <- function(data,a,b,amino, disp){
  N <- data[["proteina"]]/6.25
  amino_trigo <- list()
  aminodisp_trigo <- list()
  for(i in 1:length(amino)){
    amino_trigo[[i]] <- data.table((a[i]*N + b[i])/1000)
    names(amino_trigo[[i]]) <- amino[i]
  }
  for(i in 1:length(amino)){
    aminodisp_trigo[[i]] <- amino_trigo[[i]]*(disp[[i]]/100)
    names(aminodisp_trigo[[i]]) <- paste(amino[i],"disp",sep="_")
  }
  amino_final <- do.call(cbind,amino_trigo)
  aminodisp_final <- do.call(cbind,aminodisp_trigo)
  amino_final <- cbind(amino_final, aminodisp_final)
  dt <- cbind(data, amino_final)
  return(dt)
}
```

Llegados a este punto, podremos encontrar todas las variables necesarias para cada uno de los ingredientes principales que formarán el compuesto final. Solo faltaría calcular la variable de la energía que aporta cada ingrediente y el precio. Para la energía, una serie de regresiones serán implementadas en R desde un fichero excel. En el anexo encontraremos todo el código necesario.

El objetivo es crear una función que, pasados tres argumentos (base de datos, valores de digestibilidad y valor de eficiencia),se obtenga la variable energía. Para esto es importante crear otra función que sepa diferenciar qué base de datos se le está entrando a la función. Esto es necesario porque los valores de digestibilidad de la soja varían dependiendo de la cantidad de fibra y proteína que esta contiene. Por este motivo hay que aplicar una fórmula dinámica que no solo calcule la energía dependiendo de los valores de digestibilidad de cada ingrediente, sino que detecte si ese ingrediente es soja y aplique la fórmula con algunos cambios dependiendo de la fibra y la proteína.
Esta función se llamará `add.energy` que se encargará de diferenciar que ingrediente estamos introduciendo y aplicará la formula que da la energía.

```
trigo  <- add.energy(data=trigo, soja = F,  const = dig_trigo, efficiency = 0.783)
cebada <- add.energy(data=cebada, soja = F, const = dig_cebada, efficiency = 0.767)
maiz   <- add.energy(data=maiz, soja = F,   const = dig_maiz, efficiency = 0.801)
soja   <- add.energy(data=soja, soja = T,   const = dig_soja, efficiency = 0.605)
```

El precio es la última variable a añadir. En un fichero se puede obtener el precio para cada ingrediente para unas fechas concretas. En este caso también tenemos que diferenciar si el ingrediente es soja o no ya que por cada diferencia porcentual que haya entre la cantidad de proteína que hay y el precio para una cantidad de 47 % de proteína. En resumen, si la cantidad de proteína en soja es de 48 % el precio se incrementará un 1 % en cambio si fuese del 46 % el precio decrecería un 1 %.
Primero, de la misma manera que con los nombres de las variables, las fechas deberán tener un formato más cómodo para trabajar con ellas.

Ahora, se implementa una función que nos permitirá añadir los precios en función del día en que entraron los ingredientes.

```
  add.prices <- function(data,precios, ingrediente){
  col <- which(names(precios) == ingrediente)
  dates <- precios[["Fecha"]]
  for(i in length(dates):2){
    data[desde <  dates[i], prices := precios[i-1,col, with =F]]
    data[desde == dates[length(dates)], prices := precios[length(dates),col, with =F
```

```
      ]]
}
return(data)
}
```

Y aplicando esta función a Soja, la modificación vendrá dada dependiendo la proteína con el siguiente código:

```
soja   <- add.prices(data= soja  ,precios = precios, ingrediente="Soja")
soja[, prices := ifelse(proteina > 47, round(prices*(1+(proteina-47)/100),2),
                        ifelse(proteina < 47, round(prices*(1-(47- proteina)/100)
                            ,2),proteina))]
```

El último punto común en ambas formas de simulación es la creación de una variable auxiliar. Para crear esta variable se agrupa en un vector, las fechas de la variable `desde` de las bases de datos Trigo, Maíz y Soja, las bases de datos con más observaciones. De estas fechas solo se mantiene las que se encuentran repetidas en estas tres bases de datos. Es decir, si el día 1 de enero, hubo entradas de productos en las 3 bases de datos, esa fecha será seleccionada.
Una vez se ha el vector de fechas comunes, se ordena en orden natural y se crea un valor auxiliar desde 1 hasta el número de fechas comunes que diferencie los intervalos de tiempo entre una fecha y la siguiente en este vector de la siguiente manera.

first	second	aux
2016-01-04	2016-01-05	1
2016-01-05	2016-01-08	2
2016-01-08	2016-01-11	3

Cuadro 3: Tabla resumen de los intervalos de tiempo y el auxiliar que se le aplica

Una vez creada esta tabla, se añade este auxiliar para cada uno de los ingredientes principales teniendo en cuenta entre qué fechas se encuentra cada observación. Para discriminar entre variables pertenecientes a diferentes bases de datos, como por ejemplo la proteína que viene del trigo y la que viene de la soja, se añade el sufijo del ingrediente al que pertenece a cada una de las variables. Por ejemplo, en la base de datos de Trigo, `proteina` pasará a llamarse `proteina_trigo`.

El último procedimiento en común es convertir cada una de las bases de datos en listas, donde cada elemento de la lista sea una submuestra de las observaciones con el mismo auxiliar. Es decir, el primer elemento de cada una de las listas, serán las observaciones que el valor auxiliar sea 1.

2.2. Asignación aleatoria

En este apartado, se explica todo el procedimiento para juntar las bases de datos rellenando los datos faltantes aleatóriamente dentro de unas condiciones determinadas.

Esta función junta los `data.frame` de dos en dos y tiene como argumentos dos listas. El objetivo de esta función es, dado dos elementos de dos listas los cuales serán `data.frame`, obtener un solo `data.frame` que será un elemento de una nueva lista.

Trabajando sobre el primer elemento de las dos listas (dos `data.frame`), el primer paso es mirar cual de los dos tiene más observaciones. Se creará un auxiliar boleano que será 1 si el elemento de la primera lista tiene más observaciones que el elemento de la segunda lista, y dos auxiliares más, llamados `max_rows_list` y `min_rows_list`, contendrán el nombre de la lista que tenga más y menos elementos respectivamente.

Es importante calcular cuál es la diferencia en observaciones entre los dos elementos de las listas ya que serán el número de observaciones que se añadirán. Una vez se tiene este número n, se crea una muestra de tamaño n con reemplazamiento de los indicadores de filas que tiene la base de datos con menos observaciones. Esto quiere decir que, en el caso de que el `data.frame` con menos observaciones tenga 10 filas y la diferencia en filas sea n=3, una posible muestra seria { 3,8,4}, indicando que filas serán las elegidas para reemplazar los datos faltantes. Así pues, se haría un `rbind` del elemento con menos observaciones y las observaciones de la muestra anterior. Ahora que los dos elementos tienen la misma longitud en observaciones, se puede juntar uno con otro.

Una vez se tiene estos datos juntos, se guardarda en la posición adecuada dentro de la nueva lista.

Como esta función se aplica para un elemento de 2 listas, la forma de obtener el fichero con dos bases de datos juntas sería la siguiente:

```
new_list1 <- lapply(1:nrow(day_range), function(X)  merge_dt(list_trigo[[X]],
    list_maiz[[X]]))
```

En este caso, la función creada `merge_dt` junta la lista de la base de datos *trigo* con la lista de la base de datos *maiz*. Seguidamente se junta la nueva lista con la lista de la base de datos *cebada* y por último con *soja*.

La ventaja de este procedimiento es que la imputación de datos faltantes es en función de datos registrados en las mismas fechas, por lo que se contempla variabilidad, pero esta variabilidad estará condicionada a las fechas en las que se encuentre la observación. De esta manera, se consigue una lista con todas las bases de datos juntas y con la función `rbindlist` se obtiene el dataset deseado, objetivo del trabajo. El tiempo aproximado de ejecución es de 5.21 segundos, que incluyendo el tiempo de guardado de los ficheros sube a 9 segundos.

2.3. Asignación estática

De una manera similar al ejercicio de asignación aleatória, en este caso se aplica una asignación estática sobre los valores teóricos de estos nutrientes (fuente: FEDNA) en cada uno de los ingredientes. Con asignación estática se hace referencia a que las observaciones añadidas son constantes, es decir, la misma observación añadida n veces.

Para este procedimiento, el camino a recorrer es algo diferente al del apartado anterior. Como no replica valores que se encuentran en la base de datos, hay que modificar la matriz **fedna** para tener todos los aminoácidos y energía de los valores teóricos de cada ingrediente. Para eso utilizaremos las funciones `add.energy` y `aminoacid_ingrediente`. Aún así sigue haciendo falta asignar un número de muestra, fecha desde, fecha hasta y orígen. Estos campos vendrán fijados en el Cuadro 4 4.

Variable	Valor
Muestra	11111111
Desde	2000-01-01
Hasta	2000-01-01
Origen	TEORICO

Cuadro 4: Variables fijas en modificación de matriz fedna

Una vez se tienen todos esos valores para cada uno de los ingredientes podemos empezar con la función que permetirá juntar las bases de datos. En esta ocasión, a diferencia de la simulación aleatoria, es necesario juntar los ingredientes 2 a 2 por separado, y las dos listas resultantes juntarlas con otro procedimiento.

El input para esta función serán dos listas y el nombre del ingrediente que forman estas listas (Ej. lista_trigo, ingrediente: trigo). Como en la simulación aleatoria el primer paso es calcular cual del primer elemento de ambas listas tiene más observaciones y que magnitud tiene esta diferencia en valor absoluto. En este caso, se asigna un `data.table` auxiliar con el siguiente formato:

Ingrediente	Cebada	Maíz	Trigo	Soja
Valor	1	2	3	4

Cuadro 5: Tabla auxiliar del algorítmo de asignación estática

Una vez se crea esta tabla se puede obtener el indicador de elemento que contiene los valores de la lista fedna del mismo ingrediente que la lista con menos observaciones. Esto quiere decir que si la lista 1 es la más pequeña y es la lista trigo, veremos la posición del trigo en la lista fedna, es decir, que elemento es. En este caso este valor irá del 1 al 4 ya que solo tenemos 4 ingredientes. Así pues, se puede guardar el ingrediente que tiene menos observaciones y así quedarnos con el elemento de **fedna** adecuado para añadir. Una vez añadidos n veces esta observación se puede agregar los precios y el auxiliar por la media de estas variables en el subset ya que no tenemos fechas para estas. En este momento ya es posible juntar los elementos de ambas listas.

```
new_list1 <- lapply(1:nrow(day_range), function(X)  merge_dt(list_trigo[[X]],list_
    maiz[[X]], fedna = fedna, ing1 = "trigo", ing2 = "maiz"))
new_list2 <- lapply(1:nrow(day_range), function(X)  merge_dt(list_soja[[X]],list_
    cebada[[X]], fedna = fedna, ing1 = "soja", ing2 = "cebada"))
```

Como se ve en el código anterior, se aplica la función a trigo:maiz, soja:cebada separadamente.

A continuación, es necesario hacer una modificación de este código para juntar estas dos listas, ya que hasta ahora, se añadía de la matriz **fedna** solamente un ingrediente. En este caso, las nuevas combinaciones serán de dos ingredientes. Para ello se tomará la siguiente consideración: Se crea una nueva lista fedna, con esta vez 2 elementos. El primer elemento será el `cbind` de los elementos trigo y maiz de la matriz fedna, en el mismo orden que las columnas de `new_list1`. Este mismo procedimiento se replica para soja y cebada. Con un procedimiento parecido al anterior se juntan las bases de datos de la siguiente manera:

- Se obtiene el `data.table` con más observaciones.
- Se escoge que elemento de la nueva lista fedna se añadirá.
- Se junta n veces este elemento, donde n es la diferencia absoluta entre el número de observaciones de cada lista.
- En variables como prices, sustituimos NA por el valor más repetido en la correspondiente columna de ese elemento de la lista.
- Se juntan los elementos de cada una de las listas.
- Aplicamos la nueva función a las dos listas y obtenemos el dataset final.

Con estos procedimientos, obtenemos un dataset completo, con todos los ingredientes, valores nutricionales y precios, el cual es el objetivo principal de este trabajo.

3. Shiny

Shiny es un paquete libre de R que proporciona un entorno web elegante y potente para implementar aplicaciones web usando R. Shiny convierte los análisis, en aplicaciones webs interactivas, sin ningún tipo de conocimiento HTML CSS o Javascript.
En este trabajo, la aplicación de shiny será a partir de `shinydashboard`, una extensión de shiny para hacer dashboards interactivos con capacidad de modificaciones mediante código CSS o HTML para la personalización de esta.

3.1. Shinydashboard

Shinydashboard, tiene un lenguaje propio que permite crear todo el entorno gráfico. Para esto, el código R vendrá separado en 2 partes:

- **ui** Donde encontraremos todo el código referente al *User Interface*, esto engloba todos los comandos que crean o bien el entorno gráfico, creación de espacios reservados para tablas, gráficos y demás, o lo relacionado con sidebar, header o cuerpo de la aplicación
- **server** Donde encontraremos todo el código de cálculos, creación de gráficos, tablas...

De esta manera, si en **server** se crea un data.frame o tabla, para mostrarlo en la aplicación web se creará un marco en **ui** para poder visualizarlo.
El **ui** tiene tres partes: `dashboardHeader`, `dashboardSidebar` y `dashboardBody`. Un entorno con estas partes vacías y un server vacío generaría un output como el siguiente:

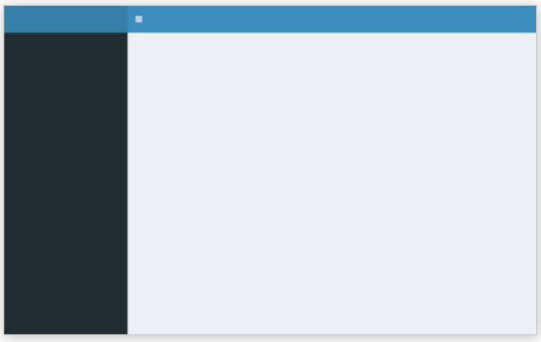

Figura 1: Shinydashboard vacío.

El aspecto básico de este dashboard puede modificarse desde R pero solo permite modificar el color principal (azul, morado, blanco, amarillo, rojo o verde), las pestañas de la barra lateral, o los iconos de la barra superior. Para cualquier otro cambio relacionado con el diseño, la herramienta principal sería CSS que es un lenguaje de hojas de estilos creado para controlar el aspecto de documentos HTML.
La cabecera de la aplicación puede contener notificaciones, información desplegable o menus de tareas. Es una forma cómoda de compartir recordatorios, accesos principales a webs o correos electrónicos de contacto, o simplemente poner la fecha. En la barra lateral se encuentran todas las pestañas y subpestañas en forma desplegable. Es una forma sencilla para navegaciones rápidas. La ventaja que ofrece respecto a `shiny` base es que puedes acumular más contenido dentro de una misma aplicación de forma ordenada. Por último, el cuerpo de la aplicación se define creando tablas que contengan información. Si se necesitan 3 pestañas en la barra lateral, se crean 3 tablas independientes. Para repartir el espacio dentro de estas tablas utilizaremos cajas, que contendrán la información que se quiera mostrar. De la misma manera, una caja puede contener varias de estas en criterio del usuario.

4. Implementación de Shiny

En el link `https://jcasinonq.shinyapps.io/tfgjoaquin/` se encuentra el dashboard donde se implementan los procedimientos anteriormente comentados. Este dashboard consta de 3 partes esenciales:

- **Subida de ficheros:** En este conjunto de pestañas se encuentra una subpestaña para cada uno de los ingredientes principales. En estas, se puede subir un fichero con una barra de carga y la opción de modificar a tiempo real la forma de subir el fichero (*separadores, formato y cabecera entre otros*). Además, en la parte inferior, hay un botón de descarga de un fichero de ejemplo, para que el fichero que subido tenga las mismas características.
- **Datos completos:** En esta pestaña se puede visualizar el fichero completo, eligiendo si queremos verlo mediante una asignación aleatoria o una asignación estática.
- **Gráficos:** En esta última pestaña podremos ver algunos gráficos de control de calidad en proteínas y aminoácidos.

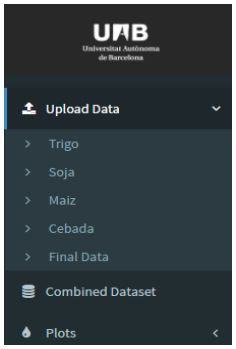

Figura 2: Barra lateral del dashboard.

Como se ve en la imagen anterior, las partes esenciales anteriormente comentadas se muestran en forma de accesos rápidos. De esta manera podremos acceder a cualquiera de estas pestañas indiferentemente de donde se encuentre el usuario. Así pues, el aspecto final del dashboard es el siguiente:

Figura 3: Barra lateral del dashboard.

Respecto a la imagen base de shiny mostrado en la figura 2, se muestra un cambio en el grafísmo en la figura 3. Esto es gracias a añadir código CSS [2] personalizado para este dashboard. En este código se define lo siguiente:

- **Definición del background:** Se establece un degradado para el fondo de la aplicación.

[2] El código se encuentra en el anexo *Código CSS*

- **Header:** Se modifica el color de la barra superior, del logo y de la barra lateral de navegación.
- **Box header:** Establece un color específico para las cajas de contenido.
- **Header strings:** Se modifica el color, fuente y tipo de letra para los títulos de tablas, gráficos...
- **Botones:** Los botones de selección de columnas a mostrar en las tablas se modifican para tener un grosor específico.

Profundizando más en el código del dashboard, y como antes se definió, en el script `ui.R` definimos el Header, Sidebar y Body. Se crean estas partes por separado y acaban llamándose al final de este script de la siguiente manera:

```
ui <- dashboardPage(
  dbHeader,
  sidebar,
  body
)
```

En el header se definen los iconos que se encuentran en la parte superior derecha en forma de menú desplegable. En el caso de este dashboard no añade información adicional más que algunos links de interés o la capacidad de contactar vía email en caso de errores.
El sidebar es una parte importante del dashboard, en este se definen los diferentes menús y se vinculan las diferentes tablas. Se pueden definir diferentes menus y submenus con tablas referenciadas que se encontrarán en el Body. El sidebar usado en este dashboard es el siguiente:

```
sidebar <- dashboardSidebar(
  sidebarMenu(
  menuItem("Behavioral Dashboard", tabName = "main", icon = icon("th")),
  menuItem("Upload Data", tabName = "aggregate", icon = icon("upload"),
           menuSubItem("Trigo", tabName = "trigo" ,icon = icon("angle-right")),
           menuSubItem("Soja", tabName = "soja" ,icon = icon("angle-right")),
           menuSubItem("Maiz", tabName = "maiz" ,icon = icon("angle-right")),
           menuSubItem("Cebada", tabName = "cebada" ,icon = icon("angle-right")),
           menuSubItem("Precios", tabName = "precios" ,icon = icon("angle-right")),
           menuSubItem("Final Data", tabName = "fdata" ,icon = icon("angle-right")))
              ,
  menuItem("Combined Dataset", tabName = "combo", icon = icon("database")),
  menuItem("Plots", tabName = "ranking", icon = icon("tint"), collapsible = T,
           menuSubItem("Distributions", tabName = "plot1" ,icon = icon("area-chart")
               ),
           menuSubItem("Check", tabName = "plot2" ,icon = icon("check-square-o")))
           ))
```

La última sección a definir es el body de `ui.R`. En esta parte del dashboard se crean las secciones referenciadas en el sidebar. En estas tablas o secciones se establece un nombre único para esta tabla y todo el contenido. Botones, tablas, imágenes, cajas de contenido, texto... Són algunas de las cosas que puedes incluir en esta sección. A continuación, un snippet de código de la tabla *Combined dataset*.

```
tabItem(tabName = "combo"
          ,fluidRow(
            h2("Top desktop domains"),
            box( p(class = "text-muted",paste("Choose the method to obtain de data
                :",
                                              "Random Sampling for a dynamic
                                                  assignation of the needed values
                                                  ",
                                              "or Static Assignation for an
                                                  imputation of theorical values")
                                                  ),
```

```
                radioButtons('method', 'Method',
                             c('None'="",
                               'Random Sampling'='random',
                               'Static Assignation'="static")),p(class = "text-
                                   muted",paste("Column Visibility help us to
                                   filter only the desired columns, it is an
                                   extra tool to help us to avoid the problem of
                                   large columns.")),
            DT::dataTableOutput("finaldata"),
            status = "primary",
            solidHeader = TRUE,
            width = "100%",
            height = "100%")),
        downloadButton('download_fdata', 'Download'))
```

El script `server.R` incluye todo el código `R` especificado de diferentes maneras. Cuando se necesita crear una base de datos o procedimientos que respondan a estímulos externos, por ejemplo, cuando en el dashboard se define si los datos requieren una simulación aleatoria o una asignación estática. Este tipo de datos se les llama reactivos, haciendo referencia a que reaccionan a un estímulo o modificación. En el dashboard, el gráfico de control de calidad contiene una tabla que va cambiando en función de dónde ponemos el margen superior e inferior. En el siguiente código se muestra como se define este dato reactivo.

```
output$Reactive1 <- DT::renderDataTable({
   names <- colnames(dt_final())
   colsIn <- names[names%like%input$variable2 & !(names%like%"disp")]
  dt <- dt_final()[, colsIn, with=F]
  dt <- dt[,lapply(.SD, as.numeric)]
  vec_pesos <- c
      (0.30397,0.15,0.24403,0.2,0.05261,0.00858,0.01109,0.00462,0.00283,0.00838,0.00224,0.00032,

  dt[, total := apply(dt,1, function(X) sum(X*vec_pesos))]
  dt <- cbind(dt_final()[, .(desde)], dt)
  dt[, Month := as.factor(months(as.Date(desde)))]
  dt <- dt[ total > input$threshold1 | total < input$threshold2]
  datatable(dt,options = list(pageLength = 8,keys = TRUE, fixedHeader = TRUE,dom =
       'Bfrtip',
                                      buttons = I('colvis'), scrollX = TRUE,
                                          deferRender = TRUE),
            extensions = c('Buttons', 'FixedHeader','KeyTable','Scroller'))
})
```

Como se observa, va cambiando en función de input$variable2 que es el tipo de componente que observamos en el gráfico (proteína, lisina o energía)y, input$threshold1 y input&threshold2 que serán los cortes superiores e inferiores. Cualquier tipo de dato que este relacionado con un dato reactivo, tambieén será reactivo. Por ejemplo, la caja que nos informa del número de observaciones que quedan por encima y por debajo de los límites. En este caso, aparte de ser un dato reactivo, es un dato renderizable, que se asigna a datos reactivos que van a mostrarse por pantalla (rederPlot, renderDataTable, renderSankey...).

En cambio, cuando queremos crear un dato reactivo que no se tiene que mostrar, es suficiente definirlo como $x < -reactive()$.

5. Resultados

Llegado a este punto, el objetivo del trabajo está cumplido. Es posible cumplir el objetivo mediante dos técnicas diferentes. Para compararlos se utilizarán los gráficos de control de calidad añadidos en la sección de gráficos del dashboard.
En este gráfico de puntos se muestra para algunas variables (proteína, lisina y energía), la cantidad de estas variables en el producto final para cada muestra. La contribución de cada ingrediente al producto final viene determinada por la siguiente tabla:

Ingrediente	Contribución en %
Cebada	20 %
Maiz	15 %
Trigo	30,397 %
Soja	24,403 %
Manteca	5,261 %
Carbonato Calcico	0,858 %
Fosfato Bicalcio	1,109 %
Sal	0,462 %
Hidroxy	0,283 %
Lisina Liq.	0,838 %
L-Treonina	0,22 %
L-Triptofano	0,032 %
L-Valina	0,053 %

Cuadro 6: Tabla con contribución de cada ingrediente al producto final.

En este ejemplo, se revisarán los gráficos sobre la proteína final con los dos métodos disponibles.

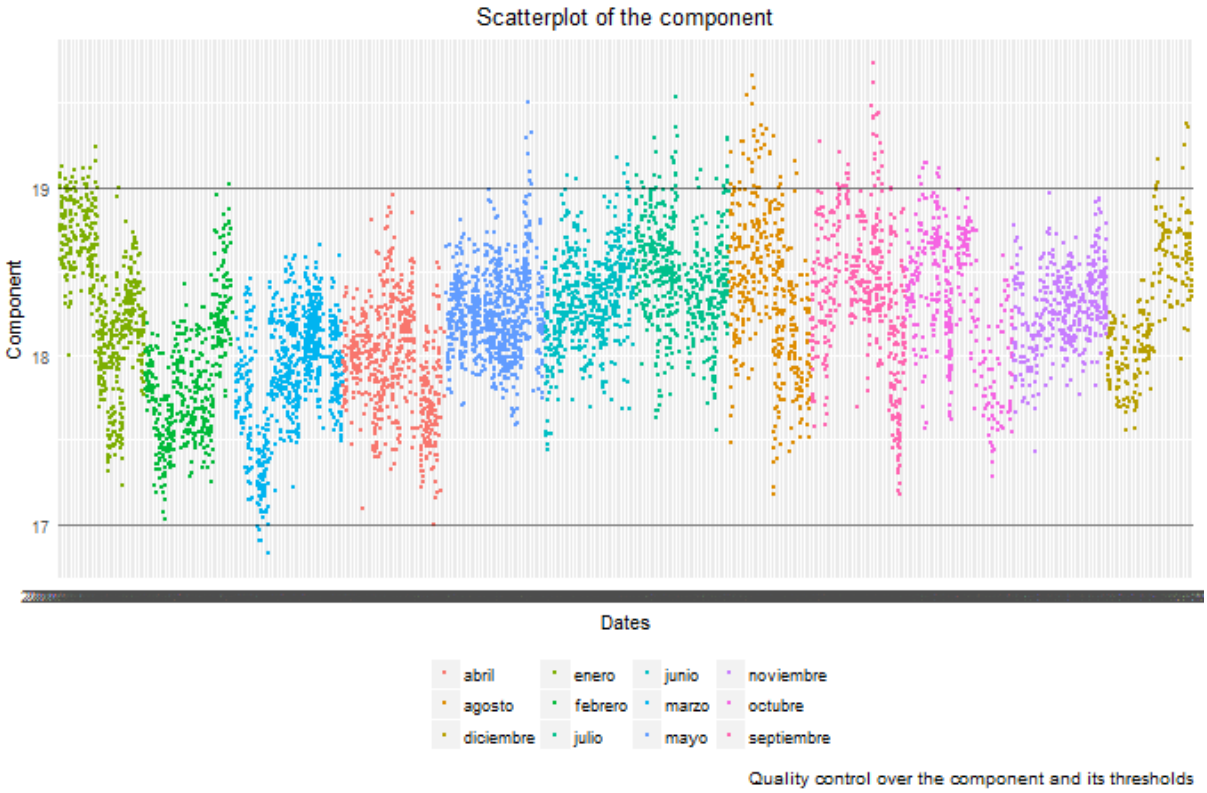

Figura 4: Scatterplot de proteína mediante el método de simulación aleatoria.

En el gráfico anterior se muestra la evolución de la proteína en el producto final diferenciando cada mes por colores. Las franjas horizontales ponen unos lindares entre los cuales deberá estar el producto final. En el dashboard es posible ver cuantas de estas observaciones quedan fuera de los límites.
El movimiento sinusoidal que se aprecia en el gráfico es un resultado que esperable.

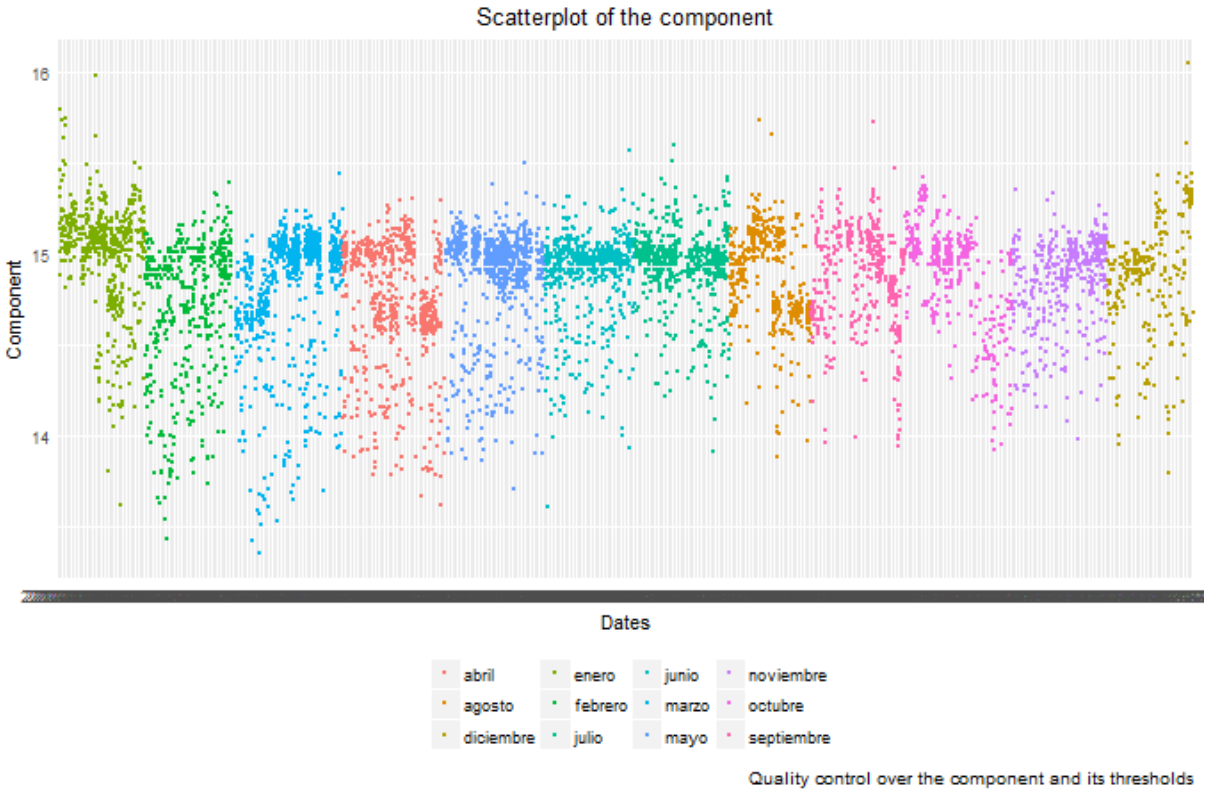

Figura 5: Scatterplot de proteína mediante el método de simulación estática.

En el gráfico anterior se muestra una variabilidad mucho menos definida. Esto se debe a la cantidad de asignaciones teóricas que hacen tender esta volatilidad hacía un punto concreto. Otro hecho remarcable es que todas las observaciones se encuentran fuera de los lindares, esto se debe a que los valores teóricos son generalmente inferiores a los valores que podemos observar en las muestras, eso hará bajar el producto final.
Este comportamiento se repite para las diferentes variables.

En conclusión, el método que se establecía como método correcto de asignación antes de la realización de este trabajo, asignación estática de valores teóricos, se ha demostrado que no es el mejor para obtener valores realistas de las observaciones que no están disponibles. Esto puede deberse en gran medida a que estos coeficientes de la matriz fedna no sean adecuados y reduzcan este producto final. Esta conclusión abre un estudio de estos coeficientes por parte de la industria y el planteamiento del uso de la asignación aleatoria por veterinarios, matemáticos y estadísticos que utilicen estos datos para modelizar un algorítmo de optimización.

6. Conclusiones

El impacto positivo que podría ocasionar una optimización en las dietas del sector porcino es suficiente para plantear una automatización y modelización del método en el que se calculan estas dietas. Esta aplicación está un paso más cerca a este objetivo y los resultados se traducírian tanto monetaria como cualitativamente.
La base de datos obtenida tras la aplicación de los algoritmos anteriormente descritos recogen toda la información necesaria para la realización de estas dietas, y el formato obtenido optimiza la capacidad de aplicar un algoritmo de optimización sobre estos datos. Esto repercute en el tiempo de ejecución de los procedimientos actuales ya que se han automatizado todos los archivos Excel que se utilizan para el cálculo de aminoácidos y energía.

La aplicación creada mediante `shinydashboard` consigue los objetivos del trabajo: Automatizar el proceso de obtención de cada una de las dietas, crear un algorítmo capaz de igualar las bases de datos para tener un solo fichero final y capacitar al algorítmo para mostrar la evolución temporal del producto final para componentes como la proteína o la lisina.

La ventaja de esta aplicación es que no requiere conocimientos técnicos en programación para poder usarla. Este hecho tiene un valor añadido ya que en ocasiones la persona que trabaje con ella no tiene que contar con esos conocimientos. Aún así, la aplicación tiene limitaciones. Dentro de la automatización, contiene parámetros fijos que no pueden modificarse mediante la interfaz. Esto repercute en que cambios en coeficientes teóricos no puedan actualizarse si no es entrando directamente al código.

6.1. Comentarios personales

En este trabajo, se hace uso de dos herramientas que durante el transcurso del grado no han sido estudiadas. Estas son las librerías `data.table` y `shiny` con su posterior aplicación en `shinydashboard`. En el caso de la librería `data.table`, que mejora el rendimiento del procesamiento de bases de datos, los conocimientos eran previos a la realización de este trabajo. Al ser un lenguaje aparte del lenguaje base de R, disponía de autonomía para la realización del código. Por este motivo, el tiempo invertido en realizar el código R ha sido ligeramente superior a uno realizado bajo supervisión ya que con la aplicación de esta librería, busco optimizar el tiempo de ejecución de los respectivos algorítmos.

En el caso de `shiny` y `shinydashboard`, la falta de conocimientos previos me llevó a invertir una gran cantidad de tiempo en conocer y dominar ligeramente este lenguaje. La aplicación se ha realizado de manera **autónoma** invirtiendo aproximadamente 150 horas de trabajo entre el estudio de la técnica y la realización de la aplicación. La falta de conocimientos previos y la autonomía influyeron directamente en este tiempo.

6.2. Perspectivas futuras

La limitación de la apliación en lo relacionado a la capacidad de modificaciones en valores teóricos vía interfaz, es uno de los temas principales que podrían seguir mejorándose. Cualquier valor teórico podría aplicarse en forma de fichero externo de la misma manera en la que se cargan bases de datos en la aplicación. Este cambio supondría más horas de trabajo pero repercutiría en una aplicación más robusta en cuanto aútonomía para trabajar vía interfaz.
De igual manera, una vez el proceso de optimización fuese realizado podría añadirse a la aplicación con todo lo que eso suponga.
Respecto a la parte gráfica, más diagramas y gráficos podrían ser añadidos si así se requiriese.|| En resumen, el presente trabajo son las bases de un proyecto que podría llevar mucho más tiempo y que sin duda tendría una gran repercusión en el sector.

7. Bibliografía

DataCamp - Learn R, Python & Data Science Online
`https://www.datacamp.com`

data.table cheatsheet - Data analysis, the `data.table` way. The official cheat sheet for the DataCamp course.
`https://s3.amazonaws.com/assets.datacamp.com/img/blog/data+table+cheat+sheet.pdf`

Shinydashboard - Official R-Studio Shinydashboard GitHub.
`https://rstudio.github.io/shinydashboard/structure.html`

DT - An R interface to the DataTables JavaScript library, official R-Studio DT GitHub.
`https://rstudio.github.io/DT/`

w3schools - The world's larget web developer site - Tutorial on CSS, a lenguage that describes the style of an HTML document.
`https://www.w3schools.com/css/`

CRAN data.table - data.table package, the fastest way to manipulate.
`https://cran.r-project.org/web/packages/data.table/data.table.pdf`

CRAN ShinyDashboard - shinydashboard package, create dashboard with Shiny.
`https://cran.r-project.org/web/packages/shinydashboard/shinydashboard.pdf`

R data.table - Official R-studio data.table GitHub.
`https://github.com/Rdatatable/data.table/wiki`

Davison, A. C., and Hinkley D. V. (1997). - Bootstrap methods and their application. Cambridge University Press.

Good, Phillip I. (2006). Birkhäuser, Boston. - Resampling methods: a practical guide to data Analysis.

8. Anexos

8.1. Función merge_dt para asignación aleatoria

La función merge_dt nos permite juntar dos bases de datos en la simulación aleatoria.

```
 merge_dt <- function(list1,list2){

 nrow_l1 <- nrow(list1)
 nrow_l2 <- nrow(list2)
 bolean <- nrow_l1 > nrow_l2

max_rows_list <- ifelse(nrow_l1 > nrow_l2,"list1","list2")#lista que tiene mas
    filas
min_rows_list <- ifelse(nrow_l1 > nrow_l2,"list2","list1")

nrows_add <- abs(nrow_l1 - nrow_l2)#diferencia de filas
which_rows <- sample(1:nrow(get(min_rows_list)),nrows_add, replace = T)#obtenemos
    un data set con las observaciones que a?adiremos

new_min_dt <- rbind(get(min_rows_list),get(min_rows_list)[which_rows,]) #juntamos
    las nuevas observaciones
new_min_dt <- new_min_dt[,4:sum(!(names(new_min_dt) %like %"aux"))] #nos quedamos
    solo con las variables referentes a los nutrientes
new_max_dt <- get(max_rows_list) #el data set mas grande
info <- new_max_dt[,1:3, with=F]
new_max_dt <- new_max_dt[,4:sum(!(names(new_max_dt) %like %"aux"))]#quitamos el
    auxiliar


if(bolean ==T){
  binded <- cbind(info,new_max_dt,new_min_dt)
} else{
  binded <- cbind(info,new_min_dt,new_max_dt)
}
return(binded)
}
```

8.2. Función merge_dt para asignación estática

La siguiente función sirve para hacer el merge en las bases de datos originales 2 a 2 en simulación estática.

```
 merge_dt <- function(list1,list2, fedna, ing1,ing2){

 nrow_l1 <- nrow(list1)
 nrow_l2 <- nrow(list2)
 bolean <- nrow_l1 > nrow_l2

max_rows_list <- ifelse(nrow_l1 > nrow_l2,"list1","list2")#lista que tiene mas
    filas
min_rows_list <- ifelse(nrow_l1 > nrow_l2,"list2","list1")

nrows_add <- abs(nrow_l1 - nrow_l2)#diferencia de filas
dt_ingnum <- data.table(cebada = 1, maiz = 2, trigo = 3, soja = 4)
val_aux <- ifelse(bolean == FALSE, which(names(fedna) %like %ing1),which(names(fedna)
    %like %ing2))

ing_min <- names(dt_ingnum)[val_aux]
dt_add <- fedna[[val_aux]]#obtenemos un data set con las observaciones que a?
    adiremos
```

```
 dt_add_aux <- data.table(prices = mean(get(min_rows_list)[[paste("prices",ing_min,
     sep="_")]]), aux = mean(get(min_rows_list)[[paste("aux",ing_min,sep="_")]]))
 names(dt_add_aux) <- paste(names(dt_add_aux),ing_min,sep="_")
 dt_add <- cbind(dt_add, dt_add_aux)



 new_min_dt <- rbind(get(min_rows_list),dt_add[rep(1,nrows_add),])
 new_min_dt <- new_min_dt[,4:sum(!(names(new_min_dt) %like%"aux"))]
 new_max_dt <- get(max_rows_list) #el data set mas grande
 info <- new_max_dt[,1:3, with=F]
 new_max_dt <- new_max_dt[,4:sum(!(names(new_max_dt) %like%"aux"))]#quitamos el
     auxiliar


 if(bolean ==F){
   binded <- cbind(info,new_max_dt,new_min_dt)
 } else{
   binded <- cbind(info,new_min_dt,new_max_dt)
 }
 return(binded)
}


new_list1 <- lapply(1:nrow(day_range), function(X)  merge_dt(list_trigo[[X]],list_
    maiz[[X]], fedna = fedna, ing1 = "trigo", ing2 = "maiz"))
new_list2 <- lapply(1:nrow(day_range), function(X)  merge_dt(list_soja[[X]],list_
    cebada[[X]], fedna = fedna, ing1 = "soja", ing2 = "cebada"))
```

Con el siguiente código se puede juntar dos bases de datos obtenidas con la función anterior.

```
if((names(new_list1[[1]])[1] %like%"maiz")==T){
  bind1 = cbind(fedna[[2]],fedna[[3]][,5:ncol(fedna[[3]])])
}else{  bind1 = cbind(fedna[[3]],fedna[[2]][,5:ncol(fedna[[2]])])}

if((names(new_list2[[1]])[1] %like%"soja")==T){
  bind2 = cbind(fedna[[4]],fedna[[1]][,5:ncol(fedna[[3]])])
}else{  bind2 =  cbind(fedna[[1]],fedna[[4]][,5:ncol(fedna[[4]])])}

fedna2 <- list(bind1,bind2)

merge_dt2 <- function(list1,list2, fedna2){

  nrow_l1 <- nrow(list1)
  nrow_l2 <- nrow(list2)
  bolean <- nrow_l1 > nrow_l2

  max_rows_list <- ifelse(nrow_l1 > nrow_l2,"list1","list2")
  min_rows_list <- ifelse(nrow_l1 > nrow_l2,"list2","list1")

  nrows_add <- abs(nrow_l1 - nrow_l2)#diferencia de filas
  val_aux <- ifelse(bolean == FALSE, 1,2)
  dt_add <- fedna2[[val_aux]]
  names(dt_add)[1:3] <- names(get(min_rows_list))[1:3]

  new_min_dt <- rbind(get(min_rows_list),dt_add[rep(1,nrows_add),], fill = T)
  most_repeated <- function(X){
    class <- class(X)
    if(class(X) == "numeric"){
      value <- as.numeric(names(sort(table(new_min_dt[[X]]),decreasing =T))[1])
    }else{value <- names(sort(table(new_min_dt[[X]]),decreasing =T))[1]}
```

```
  }


  colNas <- c(colnames(new_min_dt)[colSums(is.na(new_min_dt)) > 0])
  if(length(colNas)!= 0){
  vals <- lapply(1:length(colNas), function(X)  most_repeated(new_min_dt[,colNas[X
      ]]))
  for(i in 1:length(colNas)){
    new_min_dt[is.na(get(colNas[i])), colNas[i] := vals[[i]]]
  }}


  new_min_dt <- new_min_dt[,4:sum(!(names(new_min_dt) %like% "aux"))]
  new_max_dt <- get(max_rows_list) #el data set mas grande
  info <- new_max_dt[,1:3, with=F]
  new_max_dt <- new_max_dt[,4:sum(!(names(new_max_dt) %like% "aux"))]


  if(bolean ==F){
    binded <- cbind(info,new_max_dt,new_min_dt)
  } else{
    binded <- cbind(info,new_min_dt,new_max_dt)
  }
  names(binded)[1:3] <- c("muestra","desde","hasta")
  return(binded)
}
new_list3 <- lapply(1:nrow(day_range), function(X)  merge_dt2(new_list1[[X]],new_
    list2[[X]], fedna2 = fedna2))
```

8.3. Código CSS

```
.content-wrapper,
.right-side {
  background: #FFFFFF;
  background: linear-gradient( #282828,#73A1CC);
  border-radius: 0px;
}


.skin-blue .main-header .logo {
                            background-color: #282828;
                            height: 80px;
                           }
                             .skin-blue .main-header .logo:hover {
                             background-color: #282828;
                             }
                             .skin-blue .main-header .navbar {
                             background-color: #282828;
                             }

.box.box-solid.box-primary {
                border: #282828;
}

.box.box-solid.box-primary>.box-header {
                color: #A7CAEB;
                background: #282828;
}

h2 {
                color: white;
                font-weight: bold;
}

div.dt-button-collection{
                width: 330px;
}


div.dt-button-collection a.dt-button{
        display: inline-block;
        margin-right: 5px;
        width: 150px;
        background: white;
}
```