Data Wrangling

Anne-Fleur Hilbert

Pre-processing the data

We start by **importing** the data:

```
ingredients <- read_csv('ingredients.csv')
menu_items <- read_csv('menu_items.csv')
menuitem <- read_csv('menuitem.csv')
portion_uom_types <- read_csv('portion_uom_types.csv')
pos_ordersale <- read_csv('pos_ordersale.csv')
recipe_ingredient_assignments <- read_csv('recipe_ingredient_assignments.csv')
recipe_sub_recipe_assignments <- read_csv('recipe_sub_recipe_assignments.csv')
recipes <- read_csv('recipes.csv')
store_restaurant <- read_excel('store_restaurant.xlsx')
sub_recipe_ingr_assignments <- read_csv('sub_recipe_ingr_assignments.csv')
sub_recipes <- read_csv('sub_recipes.csv')
```

Merging all of the tables without regard for the essential information will result in a large data set that may slow down the analysis. A study of the ER diagram allows to accordingly select the relevant tables and disregard those that will only bring additional clutter. For example, the *pos_ordersale* and *store_restaurant* tables do not bring valuable information to our analysis. Indeed, we are only interested in the store the orders are coming from, which can be found in the *menuitem* table. Similarly, the *ingredients* and *portion_uom_types* tables are unnecessary as we can restrain our analysis to a specific ingredient by applying filters to alternative tables. Finally, we will not use the *recipes* table as information on the recipes themselves do not add to our analysis or our forecasts.

As mentioned previously, to limit the number of rows and ease our analysis, we must apply filters to only include the rows relating to lettuce. In this way, we keep only the relevant data for the forecast by filtering by ingredient ID when merging data. A data collection step is thus necessary prior to any further action or analysis. Additionally, only the quantity in ounces is relevant to our analysis because the restaurant branches studied are located in Berkeley, CA, and New York, NY. Therefore, we do not consider the *Lettuce - Metric* ingredient as this will give results in grams.

```
filter(ingredients, IngredientName == 'Lettuce')
```

```
## # A tibble: 1 x 4
##   IngredientName IngredientShortDescription IngredientId PortionUOMTypeId
##   <chr>          <chr>                             <dbl>            <dbl>
## 1 Lettuce        Lettuce                              27               15
```

We can further ease the merging of the data by standardising the column names relating to the menu item IDs between the *menuitem* and *menu_items* tables of the database.

```r
colnames(menuitem)[colnames(menuitem) == "Id"] <- "MenuItemId"
```

Now, we **merge** the data into a single data frame using the ER diagram to guide the relationships between each database table. We do not specify 'all = TRUE' in the merge functions below and keep the default of 'all = FALSE' to avoid merging unnecessary rows and perform an inner join. The merging process is broken down into different sub-processes as follows to regroup relevant information together:

- We merge information relating to the menu items so that the purchased menu items associated with a transaction and those associated with a recipe are in one data frame, *menu_data*.
- We look at the sub-recipes and associate them with their corresponding ingredient, filtering the data only to include the ones relating to lettuce. When associating the two, we need to multiply the factor by the quantity, that is, the number of times a sub-recipe is used in a recipe by the quantity of an ingredient used in each sub-recipe, such that we obtain the total amount of ingredients used in a recipe. This constitutes a second part of our process, the *sub_recipe_data* data frame.
- We create the *recipe_data* data frame where we filter the recipes only to include the ones related to lettuce.
- The *sub_recipe_data* and *recipe_data* are brought together in the *food_data* data frame by appending the rows of each data frame together to create a new data frame. This allows us to find the total quantity of lettuce in each recipe by summing the quantities corresponding to the same recipe.
- We merge the *food_data* and *menu_data* and find the total quantity of lettuce in each order by multiplying the quantity of the purchased item by the quantity of lettuce in each recipe.

```r
# Combine the menu data
menu_data <- menuitem %>%
  merge(menu_items, by = 'MenuItemId')

# Combine the sub-recipe data
sub_recipe_data <- sub_recipe_ingr_assignments %>%
  filter(IngredientId == '27') %>% # Filter to only include lettuce-related data
  merge(recipe_sub_recipe_assignments, by = 'SubRecipeId') %>%
  group_by(RecipeId) %>%
  summarise(Quantity = Factor * Quantity) # Find total ingredient quantity in each recipe

# Combine the sub-recipe data
recipe_data <- recipe_ingredient_assignments %>%
  filter(IngredientId == '27') %>% # Filter to only include lettuce-related data
  group_by(RecipeId) %>%
  select(RecipeId, Quantity)

# Combine data on the ingredients in the sub-recipes and recipes
food_data <- rbind(recipe_data, sub_recipe_data) %>%
  group_by(RecipeId) %>%
  summarise(Quantity = sum(Quantity)) # Find total quantity of lettuce in each recipe

# Join all the data together
data <- food_data %>%
  merge(menu_data, by = 'RecipeId') %>%
  mutate(Quantity = Quantity.x * Quantity.y) # Find total quantity of lettuce in each order
data$date <- as.Date(data$date, format="%y-%m-%d") # Convert character string to date
```

Now that the data is combined and adjusted to have the quantity of lettuce in ounces per day per restaurant, we subset the data for each restaurant and calculate the total quantity for each day.

```
restaurant_46673 <- data %>% filter(StoreNumber == '46673') %>%
  group_by(date) %>%
  summarise(`Quantity (ounces)` = sum(Quantity))

restaurant_4904 <- data %>% filter(StoreNumber == '4904') %>%
  group_by(date) %>%
  summarise(`Quantity (ounces)` = sum(Quantity))

restaurant_12631 <- data %>% filter(StoreNumber == '12631') %>%
  group_by(date) %>%
  summarise(`Quantity (ounces)` = sum(Quantity))

restaurant_20974 <- data %>% filter(StoreNumber == '20974') %>%
  group_by(date) %>%
  summarise(`Quantity (ounces)` = sum(Quantity))
```

When inspecting each data frame, we notice that the first six rows of the *restaurant_20974* data frame are not sequential. Therefore, we eliminate these first six rows to ease our analysis and avoid missing data, which may cause biased estimations.

```
restaurant_20974 <- restaurant_20974[-c(1:6),]
```

We can then write the data to csv files to be able to use them more easily at a later point in time.

```
write.csv(restaurant_46673, file ="restaurant_46673.csv")
write.csv(restaurant_4904, file ="restaurant_4904.csv")
write.csv(restaurant_12631, file ="restaurant_12631.csv")
write.csv(restaurant_20974, file ="restaurant_20974.csv")
```