LAB100

Week 17: Further Data Summarisation

This workshop introduces builds on the data summary techniques learned in Week 15 by introducing additional functions useful in creating further summaries of datasets.

- Be able to use the functions `apply` and `tapply` to obtain summary statistics from datasets.
- Produce plots of multivariate data in order to visually explore possible associations
- Perform more advanced data manipulation to get data into a more usable format for analysis

Remember that you need to change the working directory and open a new R script at the start of each workshop.

For this Workshop we will again use the airquality dataset used in Week 15. Ensure the file `airquality.txt` is in your working directory. Reload the data into RStudio using

```
airquality <- read.table("airquality.txt", header=TRUE)
```

Ensure you have the correct data by checking the dimension of the data frame

```
dim(airquality)
```

which should return

```
[1] 153   6
```

informing us that the data has 153 rows and 6 columns.

In addition, we will also use a dataset containing the monthly rainfall in mm for Durham, UK in the years 1911 to 1990. Download the file `durham.txt` from the Data directory on the LAB110 Moodle page and save it into your current working directory. Load the data into RStudio using the `read.table` command that was introduced in Week 15.

```
rainfall <- read.table("durham.txt",header=TRUE)
```

This will load a data frame called `rainfall` with 80 data columns corresponding to the years 1911 to 1990 and 12 rows corresponding to the rainfall in each month ordered January in row 1 to December in row 12.

1 Functions for summarising data: tapply

In Week 15 we can across the function `table` which allowed us to perform cross-tabulations of discrete variables. However, `table` is not very useful when the variables of interest can take a large number of possible values, or are continuous, because `table` will count the frequency of each unique value in the vector or unique combination of values if used in conjuction with multiple variables.

For instance, for the airquality dataset, we can use `table` to cross tabulated the number of days with each temperature for each month in the dataset using

```
attach(airquality)
table(Temp,Month)
```

however, this returns a rather sparse table where many of the counts are zero. We might instead want to determine the mean temperature in each month. One way of doing this, following what we learned in Week 15, is to use subscripting. For instance, we can get the mean temperature in May using

```
mean(Temp[Month==5])
```

However, a quicker way of obtaining the mean for each month in the data is the use the function `tapply`.

The function `tapply` takes three main arguments, `X` - a data vector, `INDEX` a second data vector defining categories and `FUN` a function. `tapply` groups the data in `X` by the unique values in `INDEX` and then applies the function in `FUN` to each of the groups. For instance if we take `X=Temp`, `INDEX=Month` and `FUN=mean` it will return the mean temperature in each month in the dataset

```
tapply(Temp,Month,mean)
```

This returns

```
       5        6        7        8        9
65.54839 79.10000 83.90323 83.96774 76.90000
```

which shows that the mean temperature in May was 65.55

If we change the function used in `FUN` we can get a different summary measure. For instance

```
tapply(Temp,Month,sd)
```

will give the standard deviation (computed using `sd`) of temperatures in each month.

If the function specified in `FUN` takes additional arguments these can specified within the call to `tapply`. For instance, recall that the `Solar.R` variable within the data contains missing observations. To calculate the monthly means, removing the missing observations, we need to apply `mean` with `na.rm=TRUE` hence we can use

```
tapply(Solar.R,Month,mean,na.rm=TRUE)
```

2 Functions for summarising data: apply

Now consider the Durham rainfall dataset. The data are arrange such that the rows correspond to months and the columns correspond to years. Suppose we again wish to obtain the mean rainfall in each month. In order to do this we need to find the mean of each row of the data. One way of doing this would be to separately use subscripting on each row, e.g. to find the mean for May we could use

```
mean(rainfall[5,])
```

However, to compute the means for each month simultaneously we can use the function `apply`. This is a function to allow an operation to be applied separately to each row or to each column of a matrix, or more generally it can also be used with arrays (defined in Week 16).

`apply` takes three main arguments: `X` - the matrix or array containing the data, `MARGIN` - the dimension of the matrix or array that we wish to apply the operations, `FUN` - the function to be applied. For instance, to compute the mean rainfall in each month of the year, we are calculating the row means so will take `MARGIN = 1`.

```
monthmean <- apply(rainfall, 1, mean)
```

If instead, we want to calculate the mean rainfall in each year (weighting each month equally) we would be summing over columns and would therefore take `MARGIN = 2`.

```
yearmean <- apply(rainfall, 2, mean)
```

As with `tapply`, changing `FUN` to a different function will allow other quantities to be computed. For instance, taking a year mean by weighting each month equally is somewhat crude. A slightly more accurate method would be to take a weighted mean of the form

$$\text{Weighted Mean} \quad = \quad \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

for some weights w_i. The obvious choice of weights would be the number of days in each month. The presence of leap years complicates things rather more than we would like, so let us assume all Februarys have 28 days. A function called `weighted.mean` exists to compute the weighted mean with a second argument `w` being the vector of weights. Hence we may use

```
weighted.yearmean <- apply(rainfall, 2, weighted.mean,
        w=c(31,28,31,30,31,30,31,31,30,31,30,31))
```

to compute a slightly more accurate yearly mean.

3 Re-arranging a dataset

The airquality dataset is arranged with each row corresponding to a separate set of measurements collected on a single day. In contrast, the rainfall dataset has data from completely separate months arranged on the same row. For analysis, it might be more convenient to have the data in a 'long' format such that each row just gives a separate individual month's data.

We wish to create a dataset with three columns each of length 960 corresponding to the Month in which the observation occurred, the Year the observation occurred and the Rainfall observed in that year and month. The conversion is relatively simple because we can use the fact that the 'combine' function `c` will convert a data frame into a list.

```
rain <- c(rainfall)
```

This creates a list with 80 elements each of length 12. We then wish to convert this list into a single vector. A function called `unlist` exists to do just this.

```
rainvec <- unlist(rain)
```

We now have a vector of length 960 whose first 12 elements correspond to the rainfall in each month of 1911, whose second 12 elements correspond to rainfall in 1912 and so on. It therefore just remains to create the Month and Year vectors by using the `rep` function (revisit the Workshop from Week 7 if you have forgotten how this function works).

```
monthvec <- rep(1:12,80)
yearvec <- rep(1911:1990,each=12)
```

Finally, we can put these together into a data frame. Data frames can be specified in the same way as lists, but using a function called `data.frame`:

```
rainfall.long <- data.frame(month = monthvec, year = yearvec, rain = rainvec)
```

Note that R has given the rows of the data frame slightly odd names automatically (by combining the names of the rows and columns of the original data frame). To remove these we can use

```
row.names(rainfall.long) <- NULL
```

Now that we have the data in long form we can use `tapply` to compute monthly or year means. Clearly, if we have done the data manipulation correctly, we should obtain the same answers as we did using `apply` on the original data.

```
tapply(rainfall.long$rain, rainfall.long$month, mean)
tapply(rainfall.long$rain, rainfall.long$year, mean)
```

Verify that these commands give the same month and year means as in Section 2.

4 Plotting data

Finally, a useful alternative way of summarising data is to produce some kind of graph or chart. We can use the plotting introduced in Week 10. For instance, we can explore the relationship between `Temp` and `Ozone` in the `airquality` data by using the `plot` function

```
plot(Temp,Ozone)
```

This produces a simple scatterplot. The pattern of points will give some indication of the correlation between the variables. Here there appears to be a tendancy for Ozone to be high if temperature is high, implying a positive correlation.

If we attempt to use `plot` on a data frame or matrix, a **pairs** plot will be produced. This is a matrix of scatterplots corresponding to all the pairs of variables in the data. This can be useful in quickly establishing which variables in the data may be related.

```
plot(airquality[,1:4])
```

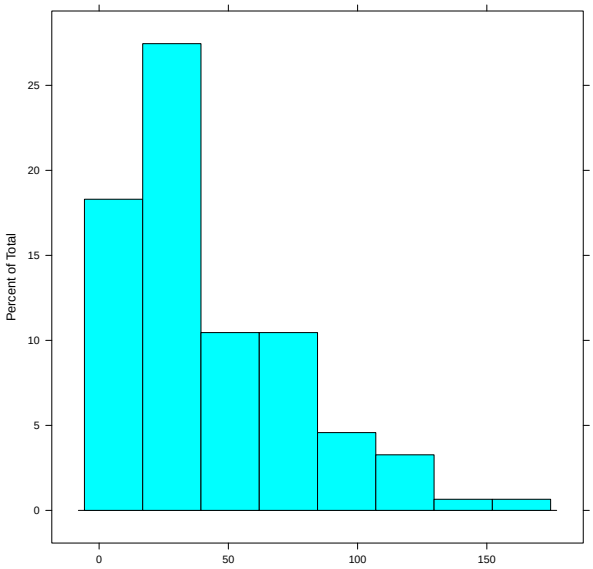
If we wish to get an idea of the distribution of a particular variable we can use a **histogram**. This bins the observed data into intervals and plots the frequency (if the bins are the same width) or density (if the bins are differing widths) in each bin. In R the function `histogram` will produce a histogram of a given. We supply the vector of data from which we wish to plot. For instance, to produce a histogram for the temperatures in the airquality data we use

```
histogram(Temp)
```

The resulting plot shows that the temperatures are roughly symmetrically distributed about a mode of around 80 fahrenheit.

Quiz 1: Histograms

Produce histograms of each of variables in the airquality dataset and hence determine which of the variables is depicted in the following histogram:



(**A**) Temp (**B**) Month (**C**) Wind (**D**) Solar.R (**E**) Ozone

---

Quiz 2: Annual Rainfall 1

By using `sum` in combination with either `apply` or `tapply` determine which year in the Durham rainfall dataset had the greatest total rainfall.

Quiz 3: Annual Rainfall 2

Again using `sum` in combination with either `apply` or `tapply` determine the median yearly rainfall in the Durham rainfall dataset.

Quiz 4: Annual Rainfall 3

A meterologist suspects that Durham has become wetter over the period of the dataset. Consider whether there is any evidence of this by calculating the Pearson correlation between total annual rainfall and calendar year. Give your answer to 2 decimal places.
[See Week 15 for how to compute the Pearson correlation].

Quiz 5: Seasonal Rainfall

Define winter to be December, January and February, spring to be March, April and May, summer to June, July and August and autumn to be September, October, November. By creating an additional variable in the `rainfall.long` dataset, use `tapply` to calculate the mean rainfall in an average month in each of winter, spring, summer and autumn.
State which season has the highest monthly rainfall. Give that season's mean monthly rainfall to 2 decimal places.
[Note: You should not attempt to correct for differing lengths of months in this question].

Quiz 6: Wettest month

Determine the number of calendar years in the period 1911-1990 for which the wettest month was in summer (June, July or August).