# ACSC/STAT 3740, Predictive Analytics

WINTER 2025
Toby Kenney

Homework Sheet 1

Model Solutions

[Note: all data in this homework are simulated.]

[Note: With many of these problems, there is no "correct" solution. These model solutions give a range of reasonable approaches, but there are many other good approaches that could be taken.]

## Basic Questions

1. *A former colleague has produced the code in the file `HW1Q1.R` to process the sports-analytics dataset in the file `HW1Q1.txt`, before leaving the company. The code is intended to remove all rows with `Balls.remaining` equal to zero. However, it does not work. Explain why the code does not work, and how to make it work, and how to restructure it in a better way.*

   Upon running the code, we get the error message "missing value where TRUE/FALSE needed". This indicates that `cricket.data$Balls.remaining[i]==0` is null. We can see that this happens when `i` is 8390, and the corresponding row is full of `NA` values. Looking at the new dimensions of the data frame, we see that it has grown significantly, instead of having rows removed. The problem comes from the precedence of the `:` operation. The line to remove the bad row should include parentheses.

   ```
   cricket.data<-rbind(cricket.data[1:(i-1),],cricket.data[(i+1):n,])
   ```

   After making this change, we still get the same error at the same value of `i`. However, the dimension of the data frame is now almost correct, but the last several rows of the data frame are all NA. This is because the for loop sets the indices at the start, and always runs for those indices, even after the data frame is shorter. This code can be fixed by running the loop backwards.

```
cricket.data<-read.table("HW1Q1.txt")

n<-dim(cricket.data)[1]

### Remove entries with zero balls remaining, as they must be mistakes.
for(i in n:1){
    if(cricket.data$Balls.remaining[i]==0){
        n<-dim(cricket.data)[1]
        cricket.data<-rbind(cricket.data[1:(i-1),],cricket.data[(i+1):n,])
    }
}
```

Note that we have also added the line to recalculate `n` every time a line is deleted.

This code successfully removes the lines with `Balls.remaining==0` in this dataset. However, it still has a bug. If the first or last row had `Balls.remaining==0`, then the code would not work. To make this more robust, we should use `seq_len`, and we can use negative indices in the second subset.

```
cricket.data<-read.table("HW1Q1.txt")

n<-dim(cricket.data)[1]

### Remove entries with zero balls remaining, as they must be mistakes.
for(i in n:1){
    if(cricket.data$Balls.remaining[i]==0){
        cricket.data<-rbind(cricket.data[seq_len(i-1),],cricket.data[-seq_len(i),])
    }
}
```

This fixes the bug, and produces working code. The code is however inefficient and it is easy to introduce bugs when modifying it. There are several better ways. The simplest is to directly use the subset operation to select the desired elements.

```
cricket.data<-read.table("HW1Q1.txt")

cricket.data.good<-cricket.data[cricket.data$Balls.remaining[i]>0,]
```

Alternatively, we can use the `dplyr` package and its `filter` command.

```
cricket.data<-read.table("HW1Q1.txt")

cricket.data.good<-cricket.data%>%filter(Balls.remaining>0)
```

Both the improved solutions use a new variable for the cleaned data. This is not essential, but is usually a good practice. It allows for easier debugging, as the original data is still available for comparison.

If we insist on using the loop to remove individual rows one at a time, the code can be improved by directly creating a vector of the rows to be removed. As in the previous code, running the loop in reverse avoids problems with the rows being renumbered. It is still somewhat inefficient as every removed row needs the table to be recopied.

```
cricket.data<-read.table("HW1Q1.txt")

rows.to.remove<-which(cricket.data$Balls.remaining[i]==0)

### Remove entries with zero balls remaining, as they must be mistakes.
for(i in rev(rows.to.remove){
    cricket.data<-rbind(cricket.data[seq_len(i-1),],cricket.data[-seq_len(i),])
}
```

2. *A government worker is investigating the effect of various parenting techniques on children's mental health. He has used the code in the file* `HW1Q2.R` *to process the data in the file* `HW1Q2.txt`. *Add comments to the code to make it easier to follow.*

   *The variables in the data set are explained in the following table:*

| *Variable* | *Meaning* |
|---|---|
| `Living.With` | *The child's household status — one of "Both", "Mother", "Father", "Joint custody", "Foster", "Other"* |
| `Family.Income` | *The combined annual household income* |
| `Age` | *The child's age* |
| `Siblings` | *The number of siblings the child has* |
| `Discipline.strict.rules` | *The extent to which the caregivers strictly enforce rules* |
| `Discipline.punishment` | *The extent to which the caregivers use punishment for misbehaviour* |
| `Parent.attention` | *The average number of hours per week that the caregivers spend with the child* |
| `Freedom` | *The extent to which the child is allowed to act without supervision.* |
| `Health.index` | *An index summarising the child's overall physicall health.* |
| `Programmes` | *The average number of hours per week that the child spends in extracurricular programmes.* |
| `Screentime` | *The average number of hours per week that the child spends using electronic devices.* |
| `Friends` | *The number of friends the child has.* |
| `School.Grades.Mathematics` | *The child's average grade in school mathematics.* |
| `School.Grades.English` | *The child's average grade in school english.* |
| `Depression.Score` | *A summary of various psychological surveys assessing the child's susceptibility to depression.* |

Here is one way the file could be commented.

```
Depression.data<-read.table("HW1Q2.txt")
library(dplyr) # for filter , mutate and select.

Depression.study.data<-Depression.data%>%
    filter(Age>10,
           ## The study focuses on children aged 11 and up
           Living.With%in%c("Both","Mother","Father","Joint Custody"),
           ## Only children living with their natural parents.
           Health.index>70) # Exclude children with very poor health.
Depression.study.data<-Depression.study.data%>%
    mutate(
        only.child=Siblings==0,
        ## to simplify analysis , we distinguish between only children
        ## and children with siblings.
        income.group=cut(Family.Income,
                         breaks=c(0,25000,40000,100000,Inf),
                         labels=c("poor","low-income","middle-income","high-income"))
        ## We divide income into 4 groups to deal with the
        ## heavy-tailed distribution.
    )%>%select(-c("Family.Income","Siblings"))
Depression.study.data.transformed<-Depression.study.data%>%
    mutate(
        log.prog=log(Programmes),
        log.attention=log(Parent.attention)
    )%>%select(-c("Programmes","Parent.attention"))
## We log-transform Programme time and Parent attention because they
## are heavy-tailed.

Screentime.pval<-rep(NA,4)
cut.offs<-c(5,10,20,30)
## We try four cut-off values for screentime.

for(i in seq_along(cut.offs)){
    cut.off<-cut.offs[i]
    temp.data<-Depression.study.data%>%
        mutate(screen.excess=Screentime>cut.off)%>%select(-c("Screentime"))
    ## We convert the Screentime to an indicator variable.
    model<-lm(Depression.Score~.,data=temp.data) # fit a linear model
    model.sum<-summary(model)
    Screentime.pval[i]<-model.sum$coefficients["screen.excessTRUE",4]
    ## extract the p-value for the coefficient of screen.excessTRUE
    ## (Naming conventions may be different on other systems, so you
    ## may need to modify this code).
}

best.cut.off<-cut.offs[Screentime.pval==min(Screentime.pval)]
### Select the cut-off that results in the smallest p-value.
```

3. *A scientist is studying crop growth. Their research assistant was analysing the data in the file `HW1Q3.txt`, and wrote the code in the file `HW1Q3.R` to process the data, before leaving suddenly. Upon reviewing the code, the scientist discovers that the code does not work. Fix the code.*

   There are several problems with the code. If we attempt to run the code, we get an error message and two warnings:

```
 Warning messages:
 1:  In distance[median, ] > cutoff :
 longer object length is not a multiple of shorter object length
 2:  In distance[median, ] > cutoff :
 longer object length is not a multiple of shorter object length
 Error in outliers[[i]]$x :  $ operator is invalid for atomic
vectors
```

   The error messages are typically cryptic. However, examining the `outliers` variable, we see that it consists of a single pair of variables `x` and `y`, instead of a list of 10 such pairs — one for each cut-off.

   [The error arises because `outliers[[1]]` refers to `outliers$x`, which is a matrix, so the entries do not have names.]

   The problem then becomes clear — the function `get.outliers` does not accept a vector of values for `cut.off`, whereas the later code assumes that it will. We can modify the function so that it can take a vector. [An alternative approach would be to call the function from the loop. This is slightly simpler, but if there is a danger that future users might expect it to handle vectors of cut-offs, then fixing the function is a better solution.]

```
Plant.data<-read.table("HW1Q3.txt")

get.outliers<-function(x,y,distance,median,cutoff){
    ## This function extracts all observations further from the median than the
    ## cut-off value, using the distance provided.
    ## If cutoff is a vector, then the function returns a list of the
    ## outliers for each cut-off value
    answer<-list(NA,length(cutoff))
    for(i in seq_along(cutoff)){
        answer[[i]]<-list("x"=x[distance[median,]>cutoff[i],],"y"=y[distance[median,]>cut
    }
    return(answer)
}

library(glmnet) # for LASSO
X<-model.matrix(yield~.,data=Plant.data)
# This creates a matrix of predictors to be used in LASSO, converting
# categorical variables to indicators.

### Calculate a distance matrix between predictors.
inv.cov<-solve(var(X[,-1]))
for(i in seq_len(n)){
    for(j in seq_len(n)){
        diff<-X[i,-1]-X[j,-1]
        distance[i,j]<-sqrt(t(diff)%*%inv.cov%*%diff)
    }
}

# The median point is the one that minimises total distance to other
# points.
median<-which(colSums(distance)==min(colSums(distance)))

LASSO<-cv.glmnet(X,Plant.data$yield,alpha=1)
# This performs cross-validation to select the best penalty parameter lambda.

coeffs<-LASSO$glmnet.fit$beta[,LASSO$lambda.1se]
### Using one standard deviation above the smallest is common practice
### to ensure a sparse model

### Extract the outliers at 10 cut-off values
cut.offs<-6+seq_len(10)/10
outliers<-get.outliers(X,Plant.data$yield,distance,median,cut.offs)

### For each cut-off, calculate the Mean Squared Error for the outliers
### Not including the intercept in the prediction
MSE<-rep(1,10)
for(i in seq_len(10)){
    predictions<-outliers[[i]]$x[,-1]%*%coeffs[-1]
    true<-outliers[[i]]$y
    MSE[i]<-mean((predictions-true)^2)
}
```

When we run this modified code, we get the error

```
Error in outliers[[i]]$x[, -1] :  incorrect number of dimensions
```

This is telling us that `outliers[[i]]$x` is not a matrix, but a vector. We find that `i==9` and `outliers[[i]]$x` only has one row. Because it only has one row, `R` reduces it to a vector, which makes the code not work. We need to further improve the `get.outliers` function by using the option `drop=FALSE` in the subset.

```
Plant.data<-read.table("HW1Q3.txt")

get.outliers<-function(x,y,distance,median,cutoff){
    ## This function extracts all observations further from the median than the
    ## cut-off value, using the distance provided.
    ## If cutoff is a vector, then the function returns a list of the
    ## outliers for each cut-off value
    answer<-list(NA,length(cutoff))
    for(i in seq_along(cutoff)){
        answer[[i]]<-list("x"=x[distance[median,]>cutoff[i],,drop=FALSE],
                          "y"=y[distance[median,]>cutoff[i]])
    }
    return(answer)
}

library(glmnet) # for LASSO
X<-model.matrix(yield~.,data=Plant.data)
# This creates a matrix of predictors to be used in LASSO, converting
# categorical variables to indicators.

### Calculate a distance matrix between predictors.
inv.cov<-solve(var(X[,-1]))
for(i in seq_len(n)){
    for(j in seq_len(n)){
        diff<-X[i,-1]-X[j,-1]
        distance[i,j]<-sqrt(t(diff)%*%inv.cov%*%diff)
    }
}

# The median point is the one that minimises total distance to other
# points.
median<-which(colSums(distance)==min(colSums(distance)))

LASSO<-cv.glmnet(X,Plant.data$yield,alpha=1)
# This performs cross-validation to select the best penalty parameter lambda.

coeffs<-LASSO$glmnet.fit$beta[,LASSO$lambda.1se]
### Using one standard deviation above the smallest is common practice
### to ensure a sparse model

### Extract the outliers at 10 cut-off values
cut.offs<-6+seq_len(10)/10
outliers<-get.outliers(X,Plant.data$yield,distance,median,cut.offs)

### For each cut-off, calculate the Mean Squared Error for the outliers
### Not including the intercept in the prediction
MSE<-rep(1,10)
for(i in seq_len(10)){
    predictions<-outliers[[i]]$x[,-1]%*%coeffs[-1]
    true<-outliers[[i]]$y
    MSE[i]<-mean((predictions-true)^2)
}
```

This code now runs without error and produces a vector of MSE values. The final value is `NaN` (Not a Number), which is because there are no outliers above the cut-off 7.0, so taking the mean of an empty set is undefined.

There is still an error in the code, which does not give any obvious signs. However, if we check the vector `coeffs`, we see that it is all zeros. The reason is that we used `LASSO$lambda.1se` as the index, whereas, we need to use the index for the corresponding value of `lambda`. That is, we can use

```
coeffs<-LASSO$glmnet.fit$beta[,LASSO$lambda==LASSO$lambda.1se])
```

or

```
coeffs<-LASSO$glmnet.fit$beta[,which(LASSO$lambda==LASSO$lambda.1se)])
```

The full corrected code is

```
Plant.data<-read.table("HW1Q3.txt")

get.outliers<-function(x,y,distance,median,cutoff){
    ## This function extracts all observations further from the median than the
    ## cut-off value, using the distance provided.
    ## If cutoff is a vector, then the function returns a list of the
    ## outliers for each cut-off value
    answer<-list(NA,length(cutoff))
    for(i in seq_along(cutoff)){
        answer[[i]]<-list("x"=x[distance[median,]>cutoff[i],,drop=FALSE],
                          "y"=y[distance[median,]>cutoff[i]])
    }
    return(answer)
}

library(glmnet) # for LASSO
X<-model.matrix(yield~.,data=Plant.data)
# This creates a matrix of predictors to be used in LASSO, converting
# categorical variables to indicators.

### Calculate a distance matrix between predictors.
inv.cov<-solve(var(X[,-1]))
for(i in seq_len(n)){
    for(j in seq_len(n)){
        diff<-X[i,-1]-X[j,-1]
        distance[i,j]<-sqrt(t(diff)%*%inv.cov%*%diff)
    }
}

# The median point is the one that minimises total distance to other
# points.
median<-which(colSums(distance)==min(colSums(distance)))

LASSO<-cv.glmnet(X,Plant.data$yield,alpha=1)
# This performs cross-validation to select the best penalty parameter lambda.

coeffs<-LASSO$glmnet.fit$beta[,which(LASSO$lambda==LASSO$lambda.1se)]
### Using one standard deviation above the smallest is common practice
### to ensure a sparse model

### Extract the outliers at 10 cut-off values
cut.offs<-6+seq_len(10)/10
outliers<-get.outliers(X,Plant.data$yield,distance,median,cut.offs)

### For each cut-off, calculate the Mean Squared Error for the outliers
### Not including the intercept in the prediction
MSE<-rep(1,10)
for(i in seq_len(10)){
    predictions<-outliers[[i]]$x[,-1]%*%coeffs[-1]
    true<-outliers[[i]]$y
    MSE[i]<-mean((predictions-true)^2)
}
```

4. *A government researcher is studying the effect of news coverage on elections. Their research assistant was analysing the data in the file `HW1Q4.txt`, and wrote the code in the file `HW1Q4.R` to process the data, before leaving suddenly. Upon reviewing the code, the researcher discovers that the code does not work. Fix the code and improve it to reduce the risk of this type of mistake happenning in future.*

   The error message here is somewhat cryptic. Examining it carefully, it is saying that we have referred to the variable `MSE` from the data frame `Folds`, but `Folds` is not a data frame, it is a vector. Looking at Folds confirms that it is a vector of "1"s and "2"s. The problem is that Folds is the name of the global variable used to communicate between the `create.folds` and `cross.validate` functions. The problem can be fixed by renaming the data frame used to save the results. However, using a global variable to communicate implicitly between functions is dangerous, and it is likely that this problem will happen again. To avoid this, the `Folds` variable should be explicitly passed between the functions.

```
create.folds<-function(nf,length){
### Make folds for cross-validation.
### The caret package has better functions to implement this,
### but we needed to avoid the dependency.

    nfold<-nf
    Folds<-sample(seq_len(nfold),length,replace=TRUE,prob=rep(1,nfold)/nfold)
    #This variable is local.
    return(Folds)
}

cross.validate<-function(formula,data,Folds,nfold){
### First call the create.folds function to set up the folds.
### Then call this function to make cross-validated predictions.
### Folds is now explicitly passed as a parameter.

    predictions<-rep(NA,dim(data)[1])

    for(i in seq_len(nfold)){
        training.data<-data[Folds!=i,]
        model<-lm(formula,data=training.data)
        test.data<-data[Folds==i,]
        predictions[Folds==i]<-predict(model,newdata=test.data)
    }
    return(predictions)
}




election.data<-read.table("HW1Q4.txt")
n<-dim(election.data)[1]

### Try different numbers of folds and compare the cross-validated MSE

### Prepare a table for the results:
Folds<-data.frame("folds"=c(2,3,4,5,10,20),"MSE"=rep(NA,6))

for(i in seq_len(6)){
### Now we can use a different name for the variable to pass folds
### between the functions, using a lower-case "f".
    folds<-create.folds(Folds$folds[i],n)
    cv.pred<-cross.validate(Outcome~.,election.data,folds,Folds$folds[i])
### calculate mean-squared error.
    Folds$MSE[i]<-mean((cv.pred-election.data$Outcome)^2)
}
```

If we feel it is necessary, or at least convenient to share variables implicitly, we need to create an environment for these functions that will allow us to pass variables that cannot clash with global variables. This can be done by creating the interacting functions with another function.

```
make.functions<-function(){
    ## These variables are local to the make.functions function
    ## Thus, only functions defined inside this function can access them.

    nfold<-0
    Folds<-NULL

    create.folds<-function(nf,length){
### Make folds for cross-validation.
### The caret package has better functions to implement this,
### but we needed to avoid the dependency.

        nfold<<-nf
        Folds<<-sample(seq_len(nfold),length,replace=TRUE,prob=rep(1,nfold)/nfold)

    }

    cross.validate<-function(formula,data){
### First call the create.folds function to set up the folds.
### Then call this function to make cross-validated predictions.

        predictions<-rep(NA,dim(data)[1])

        for(i in seq_len(nfold)){
            training.data<-data[Folds!=i,]
            model<-lm(formula,data=training.data)
            test.data<-data[Folds==i,]
            predictions[Folds==i]<-predict(model,newdata=test.data)
        }
        return(predictions)
    }

    return(list("create.folds"=create.folds,"cross.validate"=cross.validate))
}

### To make them accessible functions, we need a few extra lines of code.
functions<-make.functions()
create.folds<-functions$create.folds
cross.validate<-functions$cross.validate



election.data<-read.table("HW1Q4.txt")
n<-dim(election.data)[1]

### Try different numbers of folds and compare the cross-validated MSE
```

```
### Prepare a table for the results:
Folds<-data.frame("folds"=c(2,3,4,5,10,20),"MSE"=rep(NA,6))

for(i in seq_len(6)){
    create.folds(Folds$folds[i],n)
    cv.pred<-cross.validate(Outcome~.,election.data)
### calculate mean-squared error.
    Folds$MSE[i]<-mean((cv.pred-election.data$Outcome)^2)
```

Another approach would be to make the functions into a package. This would mean that unless the global variable `Folds` is exported, it would remain in the package's namespace and thus avoid any clashes with other variables with the same name. Approaches that use global variables are a bad idea for this simple code where only a single vector of folds needs to be passed between functions, but might make things simpler if the functions need to pass a large number of variables.

5. *The code in the file* `HW1Q5.R` *is a script for processing a reinsurance company's contract records. Improve the code to make it more reusable and less error-prone.*

   Examining the code, we identify several issues that cause the code not to be reusable or robust.

   - The exchange rates are hard-coded into the code. If an exchange rate changes, it will need to be updated in multiple places. If any one is missed, it will cause errors.
   - The code is almost the same for all cases, but is repeated in each case. Any updates to the methods need to be changed in every branch, leading to the possibility of mistakes. The code should be redesigned to use a single piece of code, either using a function or by using variables to prepare before the code. Indeed, we can see a mistake in the code — there is no checking the policy limit for Quota-sharing contracts in Germany. This is almost certainly a mistake caused by the bad code design.
   - The branching code assumes that all entries are from the available set of entries, and are correctly input. If any entry is misformatted or if a new country or contract type is added, the entry will be incorrectly processed. The code should check for this and produce an error.
   - The filename automatically uses the current date. It may be necessary to process records from a previous day. The code should be modified to allow that, probably using a function.
   - The main loop over transactions uses the `:` operator. If the transaction list is empty, it will cause an error.

   We first make a more general function to open the data. The function defaults to using todays date, but that can be overriden, or the full filename can be given.

```
get.contracts<-function(dat=as.character(
                                as.Date(
                                    date(),
                                    format="%a %b %d %H:%M:%S %Y")),
                        filename=NULL){
### By default, this loads today's contracts, but another date can be provided.
### A full filename can be provided to override the default.
    if(is.null(filename)){
        ### if filename not given, use the default for the given date.
        return(read.table(paste("Contracts",dat,".txt",sep="_")))
    }else{
        return(read.table(filename))
    }
}

contracts<-get.contracts()
```

Rather than using `if` statements to select the exchange rate, we can create a function to lookup the exchange rate from a table. We create a lookup table from two vectors. We create lookup tables for country and currency. We could create a single table to give the exchange rate for each country. However, this would include the Euro exchange rate twice, making it possible that it could be updated incorrectly. Alternatively, it might be desirable to have different rates, even for countries that use the same currency if the transactions are processed at different times.

```
### Easy to update list of all countries
### With relevant information
exchange_rate_data<-list(
    "countries"=c("Canada","USA","France","Germany","China"),
    "currencies"=c("CAD","USD","EUR","EUR","RMB"),
    "currency_list"=c("CAD","USD","EUR","RMB"),
    "exchange_rates"=c(1,1.24032,1.64720,1.80394,0.189401)
)

get.rate<-function(country,data){
### This only works for a single country.
### Would need to be modified to handle a vector.

### We pass exchange rate data as a parameter to ensure that the user
### is aware it is used in the function. Leaving it as a global
### variable would not be terrible, but keeping it as a parameter
### reduces any danger of the user incorrectly specifying it.

    if(country%in% data$countries){
        country.no=which(country==data$countries)[1]
### If multiple matches, this takes the first
        currency=data$currencies[country.no]
    }else{
        stop(paste("Country \"",country,"\" not in database.",sep=""))
    }
    if(currency %*% in data$currencies){
        currency.no=which(currency==data$currency_list)[1]
        return(data$exchange_rates[currency.no])
    }else{
        stop(paste("Currency \"",currency,"\" not in database.",sep=""))
    }
}
```

The following modified loop avoids repeating the same code.

```
day.records<-NULL


for( i in seq_along(contracts)){
    record<-contracts[i,]
    exch.rate<-get_rate(record$country,exchange_rate_data)

    if(record$contract=="Excess of Loss"){
        record$claim=record$loss-record$attachment
    }else if(record$contract=="Quota Share"){
        record$claim=record$loss*record$percentage
    }else if(record$contract=="Catastrophe Cover"){
        record$claim=record$catastrophe.loss-record$attachment
    }else{
        ## Give a clear error message
        stop(paste("Contract type \"",record$contract,"\" not known.",sep=""))
        ## It is important to put quotations around the error, as
        ## trailing spaces can cause errors.
    }
### These lines appear to be almost the same in all cases
### It is better to put them only once.
    if(record$claim<0){
        record$claim=0
    }
    if(record$claim>record$limit){
        record$claim=record$limit
    }
    record$profit=record$premium-record$claim
    if(record$contract=="Quota Share"){
        record$profit=record$profit-record$ceding.commission
    }
    record$converted.profit<-record$profit*exch.rate
}
```

For the exchange rate lookup table, we can use the names attribute to create slightly neater lookup tables and functions.

```
### Use names attribute to make better lookup tables

currency.lookup.table<-c("CAD","USD","EUR","EUR","RMB"),
names(currency.lookup.table)<-c("Canada","USA","France","Germany","China"),

exchange.rate.lookup.table=c(1,1.24032,1.64720,1.80394,0.189401)
names(exchange.rate.lookup.table)=c("CAD","USD","EUR","RMB"),

exchange_rate_data<-list(
    "currency"=currency.lookup.table,
    "exchange_rates"=exchange.rate.lookup.table
)

get.rate<-function(country,data){
### This only works for a single country.
### Would need to be modified to handle a vector.

    currency<-data$currency[country]
    if(is.null(currency)){
        stop(paste("Country \"",country,"\" not in database.",sep=""))
    }
    rate<-data$exchange_rates[currency]
    if(is.null(rate)){
        stop(paste("Currency \"",currency,"\" not in database.",sep=""))
    }
    return(rate)
}
```

One slightly awkward problem with the exchange rate lookup function is that we need to either use the exchange rate data as a global variable, or pass it as a parameter for every call to `get.rate`.

A more advanced solution uses something called a “closure” to build the fixed values into the `get.rate` function.

```
### Use a closure to fix constant parameters in a function.
### We do this by using one function that returns a new function.

make.rate.lookup<-function(currencies,exchange.rates){
    #define these local variables to fix the values.
    currency=currencies
    exchange_rates=exchange.rates

    return(function(country){
### This only works for a single country.
### Would need to be modified to handle a vector.

### Now this function only needs a single parameter.

        curr<-currency[country]
        if(is.null(curr)){
            stop(paste("Country \"",country,"\" not in database.",sep=""))
        }
        rate<-exchange_rates[curr]
        if(is.null(rate)){
            stop(paste("Currency \"",currency,"\" not in database.",sep=""))
        }
        return(rate)
    })
}

### Define the same lookup tables as the previous example
currency.lookup.table<-c("CAD","USD","EUR","EUR","RMB"),
names(currency.lookup.table)<-c("Canada","USA","France","Germany","China"),

exchange.rate.lookup.table=c(1,1.24032,1.64720,1.80394,0.189401)
names(exchange.rate.lookup.table)=c("CAD","USD","EUR","RMB"),

### And use them to make the look-up function.
get.rate<-make.rate.lookup(currency.lookup.table,exchange.rate.lookup.table)
```

In this code, the look-up tables are built into the get.rate function when it is created. Now the look-up tables are fixed. By calling `make.rate.lookup` with different look-up tables, we could change them, but this creates a new function.

Another approach we could take is to make a function to calculate the claim for the different contract types so the main loop is shorter. This does have the advantage of separating the code that deals with these types, making it easier to add a new contract type or some similar modification.

```
get.claim.amount(record){
    if(record$contract=="Excess of Loss"){
        ans<-record$loss-record$attachment
    }else if(record$contract=="Quota Share"){
        ans<-record$loss*record$percentage
    }else if(record$contract=="Catastrophe Cover"){
        ans<-record$catastrophe.loss-record$attachment
    }else{
        ## Give a clear error message
        stop(paste("Contract type \"",record$contract,"\" not known.",sep=""))
    }
    if(ans<0){
        ans=0
    }
    if(ans>record$limit){
        ans=record$limit
    }
    return(ans)
}


day.records<-NULL


for( i in seq_along(contracts)){
    record<-contracts[i,]
    exch.rate<-get_rate(record$country,exchange_rate_data)

    record$claim<-get.claim.amount(record)

    record$profit=record$premium-record$claim
    if(record$contract=="Quota Share"){
        record$profit=record$profit-record$ceding.commission
    }
    record$converted.profit=record$profit*exch.rate

}
```

6. *A data scientist has produced the code in the file* `HW1Q6.R` *to process a company's data. Testing the code on a small subset of the data, she finds that it takes 7 hours to process a dataset with 200,000 records, each with 400 predictors.*

   *(a) Approximately how long would the program be expected to take for the company's whole database of 4,000,000 records with 1,200 variables each?*

We see that the code has two nested loops, both of which concatenate lists. Concatenating lists of length $n$ is $O(n)$ complexity, so building a list of length $n$ is $O(n^2)$ complexity. Thus if $n$ is the number of records, and $p$ is the number of predictors, then the complexity of this program is $O(n^2p^2)$. Thus, if we have 20 times as many records, and 3 times as many predictors, then program execution will take $20^2 \times 3^2 = 3600$ times as long, or 25200 hours, often referred to as 1050 days.

*(b) Management deems the time required unacceptable. Rewrite the code to run more efficiently for big datasets.*

Apparently three years is too long. To make the code faster, we can reorganise the code to create the lists first.

```
records<-list()

for(i in seq_len(num_records)){
    formatted.record<-rep(NA,num_variables)
    for(j in seq_len(num_variables)){
        rec<-record[i,j]
        rec<-strsplit(rec,"-")[[1]]
        rec<-mean(as.numeric(rec)) #take mean of range
        formatted.record[j]<-rec #add to the list
    }
    records[[i]]<-formatted.record
}
```

This should be easily fast enough. However, the code can be made even more efficient using vectorisation — the `strsplit` function can take a vector as input, and will return a list of outputs. We can use the `lapply` function to more efficiently process this.

```
records<-list()

for(i in seq_len(num_records)){
    records[[i]]<-unlist(
        # lapply produces a list of vectors of length 1. unlist turns
        # it into a single vector as required.
        lapply( # When the input to strsplit is a vector, it returns a list
            strsplit(record[i,],"-"),
            function(x){mean(as.numeric(x))}
        ))
}
```

This has the same complexity $O(np)$, but a slightly faster running time.

7. *The file* ***HW1Q7.txt*** *contains data from an entertainment company about electricity usage. The data are not formatted in a very convenient way. Read the data into* **R** *and reformat into a more convenient way, and use it to create a plot showing electricity used per hour (y-axis) vs number of people (x-axis) with colour showing age group and size showing company size, with a facet grid of type of event versus time of day. Make a list of all corrections made to the data.*

   We first read and clean the customers table:

```
customers<-read.table("HW1Q7.txt",skip=2,nrow=52)
### could use stringsAsFactors, but will need to fix factors manually
### anyway to correct mistakes.

customers$ID<-as.integer(customers$ID)
str(customers)
summary(customers)
### Numerical values all look OK. Next we check the categorical
### variable "Sector".
table(customers$Sector)
### Merge "Live performance" and "Live Performance"
customers$Sector[customers$Sector=="Live performace"]<-"Live Performance"

### Now convert to factor
customers$Sector<-as.factor(customers$Sector)
```

   We identify and fix several misspellings and abbreviations. We then do the same for the events table.

```
events<-read.table("HW1Q7.txt",skip=57)

str(events)
### We note that Number.of.people is character, when it should be numeric.
### This indicates there are probably some errors that need to be fixed.

which(is.na(as.numeric(events$Number.of.People)))
events$Number.of.People[c(83,134)]
### Commas should clearly be removed.
events$Number.of.People[c(83,134)]<-c(3216,5691)
### Now we can convert it to numeric
events$Number.of.People<-as.integer(events$Number.of.People)
### Check that it converted correctly
which(is.na(events$Number.of.People))

### Now Check the levels of factor variables:
table(events$Event.type)
### Seems OK
table(events$Age.group)
### "Adlut" is a misspelling of "Adult" and "YoungChild" should be
### "Young Child".
events$Age.group[events$Age.group=="Adlut"]<-"Adult"
events$Age.group[events$Age.group=="YoungChild"]<-"Young Child"


table(events$Time.of.day)
### "Late" probably means the same as "Late Evening".
events$Time.of.day[events$Time.of.day=="Late"]<-"Late Evening"


summary(events)
### Change strings to factors
events$Event.type<-as.factor(events$Event.type)
events$Age.group<-as.factor(events$Age.group)
events$Time.of.day<-as.factor(events$Time.of.day)
```

Now that each table is cleaned, we join the two tables.

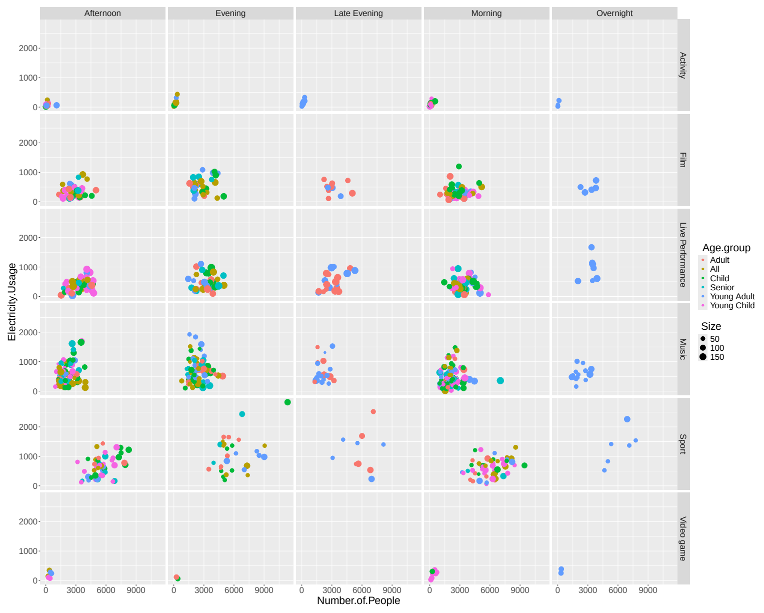
```
library(dplyr)

events.full<-events%>%left_join(customers,by=c("Host.ID"="ID"))
```

Finally, we can make the plot requested.

```
library(ggplot2)
ggplot(data=events.full,
       mapping=aes(x=Number.of.People,
                   y=Electricity.Usage,
                   colour=Age.group,
                   size=Size))+
    geom_point()+
    facet_grid(Event.type~Time.of.day)+
    theme(plot.title=element_text(size=20,hjust=0.5),
                   axis.title=element_text(size=20,hjust=0.5),
                   axis.text=element_text(size=16),
                   legend.title=element_text(size=20,hjust=0.5),
                   legend.text=element_text(size=16),
                   strip.text=element_text(size=16))
```

This gives the following plot.



[This is not a particularly great figure. It could certainly be improved in many ways.]

During this process, we fixed the following errors in the data:

## Customers

### Sector

- Customers 2 and 10 have sector "Live performance", while customers 7, 12, 15, 34, 50 and 51 have sector "Live Performance". These have been merged.

## Events

### Age.group

- Events 185, 612 and 616 have age group “Adlut”, which is almost certainly a misspelling of “Adult”.
- Events 14, 85, 184, 275, 300, 726 and 775 have age group “YoungChild”, which should be “Young Child”.

### Time.of.day

- Events 97, 214 and 457 have time of day “Late” which is presumably the same as “Late Evening”.

### Number.of.People

- Events 83 and 134 have commas in the number of people, causing them to be interpreted as character.