R-code Dirty Dozen

1. Name R objects with similiar, misleading, or uninformative names.

2. Ignore the type of a vector.

```
a= 1:8
a[2] = TRUE

flat = unlist(data1)
odd = seq(1,6145, by=2)
even = seq(2,6146, by=2)
```

3. Make multiple copies of a large object.

```
countyNames = countyVotes[,1]
bushVotes = countyVotes[,2]
kerryVotes = countyVotes[,3]
votingDiff = data.frame(countyNames, diff = bushVotes-kerryVotes)
```

   Now there are two copies of *countyVotes*. If the object is large, a lot of space has been wasted.

4. Initialize a vector, and do it one element at a time.

```
for (i in (1:n)) {
  x[i] = 0
}
```

   Rarely do we need to initialize a vector in R. If we do, there is no need for a loop. To declare a new vector, we can issue the command *numeric(length=n)*.

5. Rely on globally set variables inside functions.

```
myFunc = function(x) {
  x + K
}
```

the variable K has not been passed into the function via its parameters. If possible, it would be better to make this dependence explicit by including K in the argument list *function(x,K)*.

6. Apply a vectorized function to a vector one element at a time.

```
for(i in 1: length(votes[,2])) scales = 3*(sqrt(abs(votes[i,2])))

for (i in 1:length(x)) {
   y[i] = floor(x[i])+ 2
}
```

   There is no need for a loop or an apply when the R function operates on vectors of any length as is the case in this example.

7. Put an operation that needs to be done once inside a loop.

```
for (i in 1:length(x)) {
   z = load(data)
   y[i] = x[i]+ sum(data)
}
```

8. Call a function several times over, each time tweaking the input arguments slightly.

```
text(0.28965, -1.225, "+500,000", cex=.33)
text(0.28965, -1.245, "+100,000", cex=.33)
text(0.28965, -1.26, "+50,000", cex=.33)
text(0.28965, -1.27, "+10,000", cex=.33)
text(0.28965, -1.28, "+1,000", cex=.33)

z1 = myFunc(x,1)
z2 = myFunc(x,2)
z3 = myFunc(x,3)
```

   This type of code lends itself to errors. A simple mistyping of the name will accidentally assign the results of MyFunc to the wrong variable. If there is concern about saving intermediate results within a function then a loop over the elements in a list will take care of this problem. If the concern is over the length of time it takes to run the code then running it in the background solves the problem.

9. Keep every possible intermediate result, just in case.

```
z1 = myFunc(x)
z2 = z1 * 2
z3 = log(z2)
z4 = z3+10

size = function(vector1, vector2){
 x  =vector1
 y = vector2
 z = x-y
 s = (sqrt(abs(z)))
}
```

   When first writing code, it can be helpful to assign intermediate results to different values in order to check your code. But once it is working, it should be cleaned up.

10. Leave big objects in your workspace rather than saving them to a separate rda file.

    Saving the R objects in a separate file to be loaded only when needed and cleaning up the workspace can make your code run faster.

11. Document your code incompletely. Keep similiar looking functions around with nearly the same names, some of which work and some of which don't.

    Analagous to turning in your final essay in English, there is no need to include the intermediate drafts (especially if they are mixed in with the final version).

12. Use cat() and print() rather than trace() and browser() or adding a verbose option to your functions.