Class test II (2016) – Model answers

1. (a)

```
stations <- read.csv("stations.csv")
trips <- read.csv("trips.csv")
```

(b)

```
sum(stations$docks)
sum(subset(stations, city=="Mountain View")$docks)
```

(c)

```
mean(trips$subscription_type=="Subscriber")
mean(trips$subscription_type=="Customer")
```

(d)

```
stations[which.min(stations$docks),]
```

(e)

```
sfstations <- subset(stations, city=="San Francisco")
plot(lat~long, data=sfstations, xlab="Longitude", ylab="Latitude", pch=16, cex=2)
title("Bicycle trips in San Francisco")
```

(f)

```
od <- matrix(NA, nrow(stations), nrow(stations))
for (i in 1:nrow(od))
  for (j in 1:ncol(od))
    od[i,j] <- nrow(subset(trips, start_id==i & end_id==j))
```

or

```
od <- matrix(0, nrow(stations), nrow(stations))
for (i in 1:nrow(trips))
  od[trips$start_id[i], trips$end_id[i]] <- od[trips$start_id[i], trips$end_id[i]]+1
```

(g)

```
for (i in 1:nrow(sfstations))
  for (j in 1:nrow(sfstations))
    if (i<j) {
      n.trips <- od[sfstations$id[i], sfstations$id[j]] + od[sfstations$id[j], sfstations$id[i]]
      if (n.trips>=5) {
        lwd <- 1
        if (n.trips>=10)
          lwd <- 2
        if (n.trips>=20)
          lwd <- 3
          segments(sfstations$long[i], sfstations$lat[i], sfstations$long[j], sfstations$lat[j], col="grey", lwd
          }
    }
points(sfstations$long, sfstations$lat, pch=16, cex=2)
```

2. (a)

```
approx.permut <- function(N, n) {
  sqrt(N/(N-n)) * (N/(N-n))^N * (N-n)^n * exp(-n)
}
```

(b)

```
exact.permut <- function(N, n) {
  result <- N
  for (i in 1:(n-1))
    result <- result * (N-i)
  result
}
```

or

```r
exact.permut <- function(N, n) {
  prod((N-n+1):N)
}
```

(c)

```r
approx.permut(200,10)
exact.permut(200,10)
exp(lgamma(200+1)-lgamma(200-10+1))
# factorial(200)/factorial(190) won't work
```

3. Using a loop …

```r
pascal <- function(n) {
  triangle <- list(1)
  if (n>1)
    for (k in 2:n)
      triangle[[k]] <- c(0,triangle[[k-1]])+c(triangle[[k-1]],0)
  triangle
}
```

Using a recursive function (i.e. a function which calls itself) …

```r
pascal <- function(n) {
  if (n==1) return(list(1))
  triangle <- pascal(n-1)
  triangle[[n]] <- c(0,triangle[[n-1]])+c(triangle[[n-1]],0)
  triangle
}
```

We can now compute the first 10 rows using

```r
pascal(10)
```

The figure in the question sheet can be drawn as follows *(not required)*.

```r
draw.hexagon <- function(x, y, r=0.5/sin(pi/3)) {
  t <- seq(0,2*pi, len=7)
  lines(x+r*sin(t), y+r*cos(t))
}

draw.pascal <- function(pa) {
  par(mar=rep(0.1,4))
  n <- length(pa)
  yscale <- sin(pi/3)
  plot(NULL, xlim=c(0,n+1), ylim=yscale*c(-n-1,0), bty="n", xaxt="n", yaxt="n")
  x.centre <- (n+1)/2
  for (i in 1:n) {
    p <- length(pa[[i]])
    text(x.centre-p/2+1:p,-i*yscale,pa[[i]])
    for (j in 1:p)
      draw.hexagon(x.centre-p/2+j,-i*yscale)
  }
}

draw.pascal(pascal(13))
```

4. (a)

```r
simulate.sir <- function(N=1000, initial.infected=round(0.01*N), alpha=0.1, beta=0.25, T=100) {
  if ((initial.infected<0) | (initial.infected>N))
    stop("Invalid number of initial.infected (needs to be >=0 and <=N)")
  if  ((alpha<=0) | (alpha>=1))
    stop("Invalid value of alpha (needs to ne in (0,1))")
  if (beta<=0)
    stop("Invalid value of beta (needs to be positive)")
  result <- data.frame(susceptible=numeric(T+1), infected=numeric(T+1), recovered=numeric(T+1))
  result$susceptible[1] <- N - initial.infected
  result$infected[1] <- initial.infected
  result$recovered[1] <- 0
  for (t in 1:T) {
    Delta.S <- rbinom(1, result$susceptible[t], 1-exp(-beta*result$infected[t]/N))
    Delta.I <- rbinom(1, result$infected[t], alpha)
```

```
    result$susceptible[t+1] <- result$susceptible[t] - Delta.S
    result$infected[t+1] <- result$infected[t] + Delta.S - Delta.I
    result$recovered[t+1] <- result$recovered[t] + Delta.I
  }
  result
}
```

(b)

```
data <- simulate.sir()

time <- 1:nrow(data)-1
matplot(time, data, type="l",ylim=c(0,1000), xlab="Time", ylab="Population")
legend("right", col=1:3, lty=1:3, c("Susceptible", "Infected", "Recovered"))
```

Instead of `matplot` we could have used

```
plot(time, data[,1], ylim=c(0,1000), xlab="Time" , ylab="Population",
     type="l", lty=1, col=1)
lines(time, data[,2], lty=2, col=2)
lines(time, data[,3], lty=3, col=3)
```

(c)

```
results <- numeric(50)
for (i in 1:length(results)) {
  data <- simulate.sir()
  results[i] <- simulate.sir()$recovered[101]/1000
}
mean(results)
```