Chapter 3

Common statistical procedures

This chapter describes how to generate univariate summary statistics for continuous variables (such as means, variances, and quantiles), display and analyze frequency tables and cross-tabulations for categorical variables, and carry out a variety of one and two sample procedures.

3.1 Summary statistics

3.1.1 Means and other summary statistics

Example: See 3.6.1

```
xmean = mean(x)
```

The `mean()` function accepts a numeric vector or a numeric dataframe as arguments (date objects are also supported). Similar functions include `median()` (see 3.1.5 for more quantiles), `var()`, `sd()`, `min()`, `max()`, `sum()`, `prod()`, and `range()` (note that the latter returns a vector containing the minimum and maximum value). The `which.max()` and `which.min()` functions can be used to identify the observation containing the maximum and minimum values, respectively (see also `which()`, Section 1.5.2).

3.1.2 Means by group

See also 4.1.6 (fitting regression separately by group) *Example:* See 2.13.6

```
tapply(y, x, mean)
```

or

```
ave(y, as.factor(x), FUN=mean)
```

or

```
aggregate(y, list(x1, x2), mean)
```

The `tapply()` function applies the specified function given as the third argument (in this case `mean()`) to the vector `y` stratified by every unique set of values of the list of factors specified `x` (see also 1.6.3, the apply family of functions). It returns a vector with length equal to the number of unique set of values of `x`. Similar functionality is available using the `ave()` function (see `example(ave)`), which returns a vector of the same length as `x` with each element equal to the mean of the subset of observations with the factor level specified by `y`. The `aggregate()` function can be used in a similar manner, with a list of variables given as argument (see also 2.13.6).

3.1.3 Trimmed mean

```
mean(x, trim=frac)
```

The value `frac` can take on range 0 to 0.5, and specifies the fraction of observations to be trimmed from each end of `x` before the mean is computed (`frac=0.5` yields the median).

3.1.4 Five-number summary

Example: See 3.6.1

The five-number summary (minimum, 25th percentile, median, 75th percentile, maximum) is a useful summary of the distribution of observed values.

```
quantile(x)
fivenum(x)
summary(ds)
```

The `summary()` function calculates the five-number summary (plus the mean) for each of the columns of the vector or dataset given as arguments. The default output of the `quantile()` function is the min, 25th percentile, median, 75th percentile and the maximum. The `fivenum()` function reports the lower and upper hinges instead of the 25th and 75th percentiles, respectively.

3.1.5 Quantiles

Example: See 3.6.1

```
quantile(x, c(.025, .975))
quantile(x, seq(from=.95, to=.975, by=.0125))
```

Details regarding the calculation of quantiles in `quantile()` can be found using `help(quantile)`.

3.1.6 Centering, normalizing, and scaling

```
zscoredx = scale(x)
```

or

```
zscoredx = (x-mean(x))/sd(x)
```

The default behavior of `scale()` is to create a Z-score transformation. The `scale()` function can operate on matrices and dataframes, and allows the specification of a vector of the scaling parameters for both center and scale (see also `sweep()`, a more general function).

3.1.7 Mean and 95% confidence interval

Example: See 3.6.4

```
tcrit = qt(.975, length(x)-1)
ci95 = mean(x) + c(-1,1)*tcrit*sd(x)/sqrt(length(x))
```

or

```
t.test(x)$conf.int
```

While the appropriate 95% confidence interval can be generated in terms of the mean and standard deviation, it is more straightforward to use the t-test function to calculate the relevant quantities.

3.1.8 Maximum likelihood estimation of distributional parameters

Example: See 3.6.1

```
library(MASS)
fitdistr(x, densityfunction}
```

Options for `densityfunction` include `beta`, `cauchy`, `chi-squared`, `exponential`, `f`, `gamma`, `geometric`, `log-normal`, `lognormal`, `logistic`, `negative binomial`, `normal`, `Poisson`, `t` or `weibull`.

3.1.9 Bootstrapping a sample statistic

Bootstrapping is a powerful and elegant approach to estimation of sample statistics that can be implemented even in many situations where asymptotic results are difficult to find or otherwise unsatisfactory [11, 24]. Bootstrapping proceeds using three steps: First, resample the dataset (with replacement) a specified number of times (typically on the order of 10,000), calculate the desired statistic from each resampled dataset, then use the distribution of the resampled statistics to estimate the standard error of the statistic (normal approximation method), or construct a confidence interval using quantiles of that distribution (percentile method).

As an example, we consider estimating the standard error and 95% confidence interval for the coefficient of variation (COV), defined as σ/μ, for a random variable X. Note that for both packages, the user must provide code (as a function) to calculate the statistic of interest.

```
library(boot)
covfun = function(x, i) {sd(x[i])/mean(x[i])}
res = boot(x, covfun, R=10000)
print(res)
plot(res)
quantile(res$t, c(.025, .975))  # percentile method
```

The first argument to the `boot()` function specifies the data to be bootstrapped (in this case a vector, though a dataframe can be set up if more than one variable is needed for the calculation of the sample statistic) as well as a function to calculate the statistic for each resampling iteration. Here the function `covfun()` takes two arguments: The first is the original data (as a vector) and the second a set of indices into that vector (that represent a given bootstrap sample).

The `boot()` function returns an object of class `boot`, with an associated `plot()` function that provides a histogram and QQ-plot (see `help(plot.boot)`). The return value object (`res`, above) contains the vector of resampled statistics (`res$t`), which can be used to estimate the quantiles or standard error. The `boot.ci()` function can be used to generate bias-corrected and accelerated intervals.

3.1.10 Proportion and 95% confidence interval

Example: See 7.1.2

```
binom.test(sum(x), length(x))
prop.test(sum(x), length(x))
```

The `binom.test()` function calculates an exact Clopper–Pearson confidence interval based on the F distribution [4] using the first argument as the number of successes and the second argument the number of trials, while `prop.test()` calculates an approximate confidence interval by inverting the score test. Both allow specification of `p` for the null hypothesis. The `conf.level` option can be used to change the default confidence level.

3.1.11 Tests of normality

```
library(nortest)
ad.test(x)      # Anderson-Darling test
cvm.test(x)     # Cramer-von Mises test
lillie.test(x)  # Lilliefors (KS) test
pearson.test(x) # Pearson chi-square
sf.test(x)      # Shapiro-Francia test
```

3.2 Contingency tables

3.2.1 Display counts for a single variable

Example: See 3.6.3

Frequency tables display counts of values for a single variable (see also 3.2.2, cross-classification tables).

```
count = table(x)
percent = count/sum(count)*100
rbind(count, percent)
```

Additional marginal displays (in this case the percentages) can be added and displayed along with the counts.

3.2.2 Display cross-classification table

Example: See 3.6.3

Contingency tables display group membership across categorical (grouping) variables. They are also known as cross-classification tables, cross-tabulations, and two-way tables.

```
mytab = table(y, x)
addmargins(mytab)
prop.table(mytab, 1)
```

or

```
xtabs(~ y + x)
```

or

```
library(prettyR)
xtab(y ~ x, data=ds)
```

The `addmargins()` function adds (by default) the row and column totals to a table, while `prop.table()` can be used to calculate row totals (with option 1) and column totals (with option 2). The `colSums()`, `colMeans()` functions (and their equivalents for rows) can be used to efficiently calculate sums and means for numeric arrays. Missing values can be displayed using `table()` by specifying `exclude=NULL`.

The `xtabs()` function can be used to create a contingency table from cross-classifying factors. Much of the process of displaying tables is automated in the `prettyR` library `xtab()` function (which requires specification of a dataframe to operate on).

3.2.3 Pearson chi-square statistic

Example: See 3.6.3

```
chisq.test(x, y)
```

The `chisq.test()` command can accept either two factor vectors or a matrix with counts. By default a continuity correction is used (this can be turned off using the option `correct=FALSE`).

3.2.4 Cochran–Mantel–Haenszel test

The Cochran–Mantel–Haenszel test provides an assessment of the relationship between X_2 and X_3, stratified by (or controlling for) X_1. The analysis provides a way to adjust for the possible confounding effects of X_1 without having to estimate parameters for them.

```
mantelhaen.test(x2, x3, x1)
```

3.2.5 Fisher's exact test

Example: See 3.6.3

```
fisher.test(y, x)
```

or

```
fisher.test(ymat)
```

The `fisher.test()` command can accept either two class vectors or a matrix with counts (here denoted by `ymat`). For tables with many rows and/or columns, p-values can be computed using Monte Carlo simulation using the `simulate.p.value` option.

3.2.6 McNemar's test

McNemar's test tests the null hypothesis that the proportions are equal across matched pairs, for example, when two raters assess a population.

```
mcnemar.test(y, x)
```

The `mcnemar.test()` command can accept either two class vectors or a matrix with counts.

3.3 Bivariate statistics

3.3.1 Epidemiologic statistics

Example: See 3.6.3

It is straightforward to calculate summary measures such as the odds ratio, relative risk and attributable risk (see also 5.1, generalized linear models).

```
sum(x==0&y==0)*sum(x==1&y==1)/(sum(x==0&y==1)*sum(x==1&y==0))
```

or

```
tab1 = table(x, y)
tab1[1,1]*tab1[2,2]/(tab1[1,2]*tab1[2,1])
```

or

```
glm1 = glm(y ~ x, family=binomial)
exp(glm1$coef[2])
```

or

```
library(epitools)
oddsratio.fisher(x, y)
oddsratio.wald(x, y)
riskratio(x, y)
riskratio.wald(x, y)
```

The `epitab()` function in `library(epitools)` provides a general interface to many epidemiologic statistics, while `expand.table()` can be used to create individual level data from a table of counts.

3.3.2 Test characteristics

The sensitivity of a test is defined as the probability that someone with the disease (D=1) tests positive (T=1), while the specificity is the probability that someone without the disease (D=0) tests negative (T=0). For a dichotomous screening measure, the sensitivity and specificity can be defined as $P(D = 1, T = 1)/P(D = 1)$ and $P(D = 0, T = 0)/P(D = 0)$, respectively (see also 6.1.17, receiver operating characteristic curves).

```
sens = sum(D==1&T==1)/sum(D==1)
spec = sum(D==0&T==0)/sum(D==0)
```

Sensitivity and specificity for an outcome D can be calculated for each value of a continuous measure T using the following code.

```
library(ROCR)
pred = prediction(T, D)
diagobj = performance(pred, "sens", "spec")
spec = slot(diagobj, "y.values")[[1]]
sens = slot(diagobj, "x.values")[[1]]
cut = slot(diagobj, "alpha.values")[[1]]
diagmat = cbind(cut, sens, spec)
head(diagmat, 10)
```

The ROCR package facilitates the calculation of test characteristics, including sensitivity and specificity. The `prediction()` function takes as arguments the continuous measure and outcome. The returned object can be used to calculate quantities of interest (see `help(performance)` for a comprehensive list). The `slot()` function is used to return the desired sensitivity and specificity values for each cut score, where `[[1]]` denotes the first element of the returned list (see Section 1.5.4, `help(list)`, and `help(Extract)`).

3.3.3 Correlation

Example: See 3.6.2 and 6.6.9

```
pearsoncorr = cor(x, y)
spearmancorr = cor(x, y, method="spearman")
kendalltau = cor(x, y, method="kendall")
```

or

```
cormat = cor(cbind(x1, ..., xk))
```

Tests and confidence intervals for correlations can be generated using the function `cor.test()`. Specifying `method="spearman"` or `method="kendall"` as an option to `cor()` or `cor.test()` generates the Spearman or Kendall correlation coefficients, respectively. A matrix of variables (created with `cbind()`, see 2.9.1) can be used to generate the correlation between a set of variables. Subsets of the returned correlation matrix can be selected, as demonstrated in Section 3.6.2. This can save space by avoiding replicating correlations above and below the diagonal of the correlation matrix. The `use` option for `cor()` specifies how missing values are handled (either `"all.obs"`, `"complete.obs"`, or `"pairwise.complete.obs"`).

3.3.4 Kappa (agreement)

```
library(irr)
kappa2(data.frame(x, y))
```

The `kappa2()` function takes a dataframe (see 1.5.6) as argument. Weights can be specified as an option.

3.4 Two sample tests for continuous variables

3.4.1 Student's t-test

Example: See 3.6.4

```
t.test(y1, y2)
```

or

```
t.test(y ~ x)
```

The first example for the `t.test()` command displays how it can take two vectors (`y1` and `y2`) as arguments to compare, or in the latter example a single vector corresponding to the outcome (`y`), with another vector indicating group membership (`x`) using a formula interface (see Sections 1.5.7 and 4.1.1). By default, the two-sample t-test uses an unequal variance assumption. The option `var.equal=TRUE` can be added to specify an equal variance assumption. The command `var.test()` can be used to formally test equality of variances.

3.4.2 Nonparametric tests

Example: See 3.6.4

```
wilcox.test(y1, y2)
ks.test(y1, y2)

library(coin)
median_test(y ~ x)
```

By default, the `wilcox.test()` function uses a continuity correction in the normal approximation for the p-value. The `ks.test()` function does not calculate an exact p-value when there are ties. The median test shown will generate an exact p-value with the `distribution="exact"` option.

3.4.3 Permutation test

Example: See 3.6.4

```
library(coin)
oneway_test(y ~ as.factor(x), distribution=approximate(B=bnum))
```

The `oneway_test` function in the `coin` library implements a variety of permutation-based tests (see also the `exactRankTests` package). An empirical p-value is generated if `distribution=approximate` is specified. This is asymptotically equivalent to the exact p-value, based on bnum Monte Carlo replicates.

3.4.4 Logrank test

Example: See 3.6.5

See also 6.1.18 (Kaplan–Meier plot) and 5.3.1 (Cox proportional hazards model)

```
library(survival)
survdiff(Surv(timevar, cens) ~ x)
```

If cens is equal to 0, then Surv() treats timevar as the time of censoring, otherwise the time of the event. Other tests within the G-rho family of Fleming and Harrington [18] are supported by specifying the rho option.

3.5 Further resources

Verzani [77] and Everitt and Hothorn [13] present comprehensive introductions for the use of R to fit a common statistical model. Efron and Tibshirani [11] provide a comprehensive overview of bootstrapping. A readable introduction to permutation-based inference can be found in [22]. Collett [5] presents an accessible introduction to survival analysis.

3.6 HELP examples

To help illustrate the tools presented in this chapter, we apply many of the entries to the HELP data. The code for these examples can be downloaded from http://www.math.smith.edu/r/examples.

3.6.1 Summary statistics and exploratory data analysis

We begin by reading the dataset.

```
> options(digits=3)
> options(width=68)  # narrows output to stay in the gray box
> ds = read.csv("http://www.math.smith.edu/r/data/help.csv")
> attach(ds)
```

A first step would be to examine some univariate statistics (3.1.1) for the baseline CESD (Center for Epidemiologic Studies–Depression measure of depressive symptoms) score.

We can use functions which produce a set of statistics, such as fivenum(), or request them singly.

```
> fivenum(cesd)

[1]  1 25 34 41 60

> mean(cesd); median(cesd)

[1] 32.8

[1] 34

> range(cesd)

[1]  1 60

> sd(cesd)

[1] 12.5

> var(cesd)

[1] 157
```

We can also generate desired statistics. Here, we find the deciles (3.1.5).

```
> quantile(cesd, seq(from=0, to=1, length=11))

  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
 1.0 15.2 22.0 27.0 30.0 34.0 37.0 40.0 44.0 49.0 60.0
```

Graphics can allow us to easily review the whole distribution of the data. Here we generate a histogram (6.1.9) of CESD, overlaid with its empirical PDF (6.1.21) and the closest-fitting normal distribution (see Figure 3.1).

```
> library(MASS)
> hist(cesd, main="distribution of CESD scores", freq=FALSE)
> lines(density(cesd), lty=2, lwd=2)
> xvals = seq(from=min(cesd), to=max(cesd), length=100)
> param = fitdistr(cesd, "normal")
> lines(xvals, dnorm(xvals, param$estimate[1],
+    param$estimate[2]), lwd=2)
```

3.6.2 Bivariate relationships

We can calculate the correlation (3.3.3) between CESD and MCS and PCS (mental and physical component scores). First, we show the default correlation matrix.

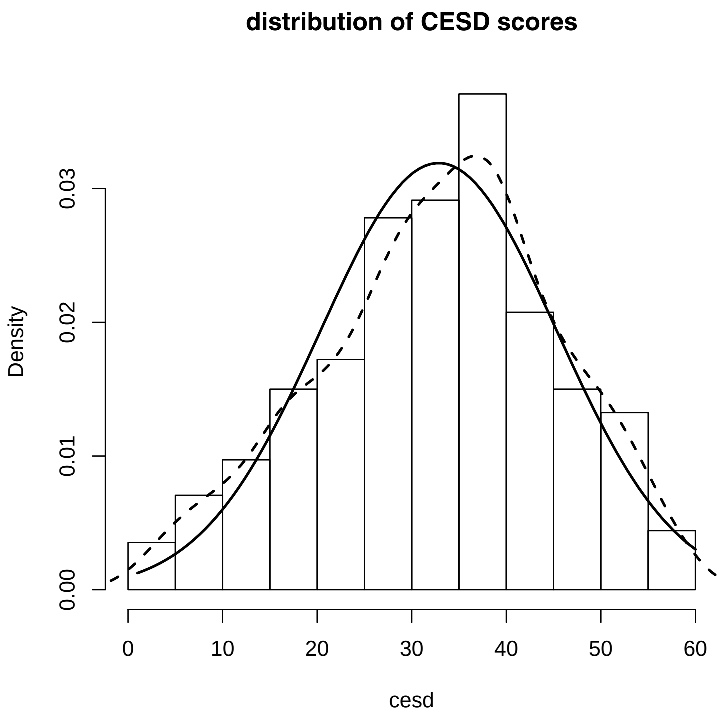

Figure 3.1: Density plot of depressive symptom scores (CESD) plus superimposed histogram and normal distribution.

```
> cormat = cor(cbind(cesd, mcs, pcs))
> cormat

       cesd    mcs    pcs
cesd  1.000 -0.682 -0.293
mcs  -0.682  1.000  0.110
pcs  -0.293  0.110  1.000
```

To save space, we can just print a subset of the correlations.

```
> cormat[c(2, 3), 1]

   mcs    pcs
-0.682 -0.293
```

Figure 3.2 displays a scatterplot (6.1.1) of CESD and MCS, for the female subjects. The plotting character (6.2.1) is the primary substance (Alcohol, Cocaine, or Heroin). We add a rug plot (6.2.9) to help demonstrate the marginal distributions.

```
> plot(cesd[female==1], mcs[female==1], xlab="CESD", ylab="MCS",
+    type="n", bty="n")
> text(cesd[female==1&substance=="alcohol"],
+    mcs[female==1&substance=="alcohol"],"A")
> text(cesd[female==1&substance=="cocaine"],
+    mcs[female==1&substance=="cocaine"],"C")
> text(cesd[female==1&substance=="heroin"],
+    mcs[female==1&substance=="heroin"],"H")
> rug(jitter(mcs[female==1]), side=2)
> rug(jitter(cesd[female==1]), side=3)
```

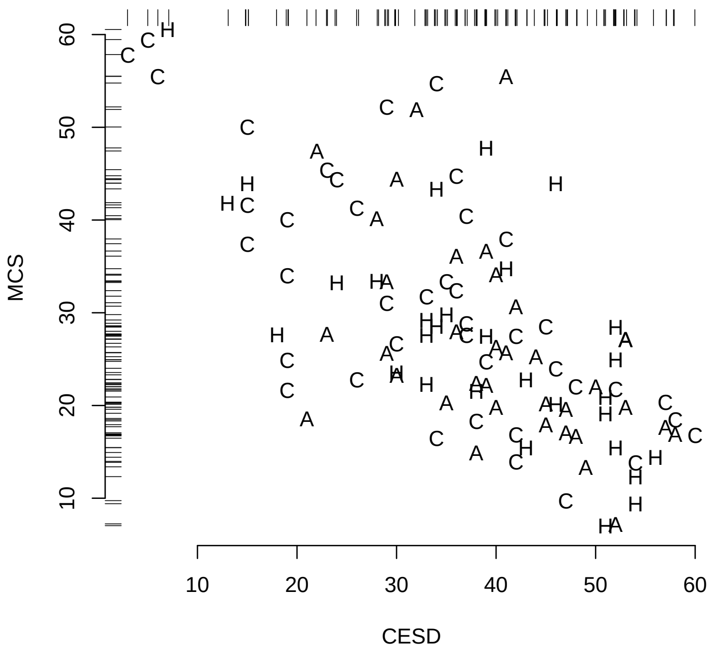

Figure 3.2: Scatterplot of CESD and MCS for women, with primary substance shown as the plot symbol.

3.6.3 Contingency tables

Here we display the cross-classification (contingency) table (3.2.2) of homeless at baseline by gender, calculate the observed odds ratio (OR, Section 3.3.1), and assess association using the Pearson χ^2 test (3.2.3) and Fisher's exact test (3.2.5).

```
> count = table(substance)
> percent = count/sum(count)*100
> rbind(count, percent)

        alcohol cocaine heroin
count     177.0   152.0  124.0
percent    39.1    33.6   27.4
```

For cross-classification tables, similar syntax is used.

```
> table(homeless, female)

        female
homeless   0   1
       0 177  67
       1 169  40
```

The `prettyR` library provides a way to display tables with additional statistics.

```
> library(prettyR)
> xtres = xtab(homeless ~ female, data=ds)

Crosstabulation of homeless by female
         female
homeless        0        1
0             177       67      244
            72.54    27.46    53.86
            51.16    62.62

1             169       40      209
            80.86    19.14    46.14
            48.84    37.38

              346      107      453
            76.38    23.62

odds ratio = 0.63
relative risk (homeless-1) = 0.7
```

We can easily calculate the odds ratio directly.

```
> or = (sum(homeless==0 & female==0)*
+         sum(homeless==1 & female==1))/
+        (sum(homeless==0 & female==1)*
+         sum(homeless==1 & female==0))
> or

[1] 0.625
```

```
> library(epitools)
> oddsobject = oddsratio.wald(homeless, female)
> oddsobject$measure

         odds ratio with 95% C.I.
Predictor estimate lower upper
        0    1.000    NA    NA
        1    0.625 0.401 0.975

> oddsobject$p.value

         two-sided
Predictor midp.exact fisher.exact chi.square
        0         NA           NA         NA
        1     0.0381       0.0456     0.0377
```

The χ^2 and Fisher's exact tests assess independence between gender and homelessness.

```
> chisqval = chisq.test(homeless, female, correct=FALSE)
> chisqval

        Pearson's Chi-squared test

data:  homeless and female
X-squared = 4.32, df = 1, p-value = 0.03767
```

```
> fisher.test(homeless, female)

        Fisher's Exact Test for Count Data

data:  homeless and female
p-value = 0.04560
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.389 0.997
sample estimates:
odds ratio
     0.626
```

The `fisher.test()` command returns the conditional MLE for the odds ratio, which is attenuated towards the null value.

3.6.4 Two sample tests of continuous variables

We can assess gender differences in baseline age using a t-test (3.4.1) and non-parametric procedures.

```
> ttres = t.test(age ~ female, data=ds)
> print(ttres)

        Welch Two Sample t-test

data:  age by female
t = -0.93, df = 180, p-value = 0.3537
alternative hypothesis:
true difference in means is not equal to 0

95 percent confidence interval:
 -2.45  0.88
sample estimates:
mean in group 0 mean in group 1
           35.5            36.3
```

The `names()` function can be used to identify the objects returned by the `t.test()` function.

```
> names(ttres)

[1] "statistic"   "parameter"   "p.value"     "conf.int"
[5] "estimate"    "null.value"  "alternative" "method"
[9] "data.name"

> ttres$conf.int

[1] -2.45  0.88
attr(,"conf.level")
[1] 0.95
```

A permutation test (3.4.3) can be run to generate a Monte Carlo p-value.

```
> library(coin)
> oneway_test(age ~ as.factor(female),
+    distribution=approximate(B=9999), data=ds)

        Approximative 2-Sample Permutation Test

data:  age by as.factor(female) (0, 1)
Z = -0.92, p-value = 0.3623
alternative hypothesis: true mu is not equal to 0
```

Similarly, a Wilcoxon nonparametric test (3.4.2) can be requested,

```
> wilcox.test(age ~ as.factor(female), correct=FALSE)

        Wilcoxon rank sum test

data:  age by as.factor(female)
W = 17512, p-value = 0.3979
alternative hypothesis: true location shift is not equal to 0
```

as well as a Kolmogorov–Smirnov test.

```
> ksres = ks.test(age[female==1], age[female==0], data=ds)
> print(ksres)

        Two-sample Kolmogorov-Smirnov test

data:  age[female == 1] and age[female == 0]
D = 0.063, p-value = 0.902
alternative hypothesis: two-sided
```

We can also plot estimated density functions (6.1.21) for age for both groups, and shade some areas (6.2.14) to emphasize how they overlap (Figure 3.3). We create a function (see 1.6) to automate this task.

```
> plotdens = function(x,y, mytitle, mylab) {
+    densx = density(x)
+    densy = density(y)
+    plot(densx, main=mytitle, lwd=3, xlab=mylab, bty="l")
+    lines(densy, lty=2, col=2, lwd=3)
+    xvals = c(densx$x, rev(densy$x))
+    yvals = c(densx$y, rev(densy$y))
+    polygon(xvals, yvals, col="gray")
+ }
```

The `polygon()` function is used to fill in the area between the two curves.

```
> mytitle = paste("Test of ages: D=", round(ksres$statistic,3),
+    " p=", round(ksres$p.value, 2), sep="")
> plotdens(age[female==1], age[female==0], mytitle=mytitle,
+    mylab="age (in years)")
> legend(50, .05, legend=c("Women", "Men"), col=1:2, lty=1:2,
+    lwd=2)
```

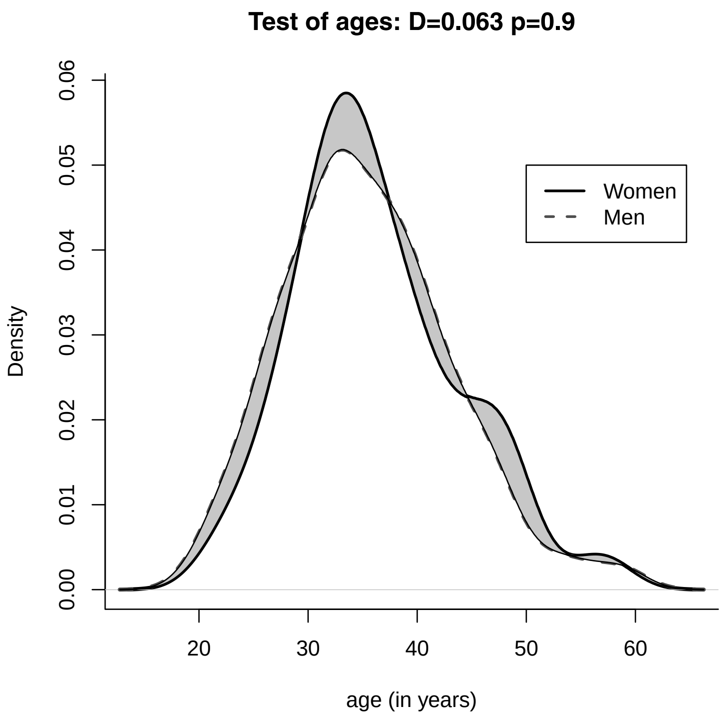

Figure 3.3: Density plot of age by gender.

3.6.5 Survival analysis: Logrank test

The logrank test (3.4.4) can be used to compare estimated survival curves between groups in the presence of censoring. Here we compare randomization groups with respect to `dayslink`, where a value of 0 for `linkstatus` indicates that the observation was censored, not observed, at the time recorded in `dayslink`.

```
> library(survival)
> survobj = survdiff(Surv(dayslink, linkstatus) ~ treat,
+    data=ds)
> print(survobj)

Call:
survdiff(formula = Surv(dayslink, linkstatus) ~ treat, data = ds)

n=431, 22 observations deleted due to missingness.

          N Observed Expected (O-E)^2/E (O-E)^2/V
treat=0 209       35     92.8      36.0      84.8
treat=1 222      128     70.2      47.6      84.8

 Chisq= 84.8  on 1 degrees of freedom, p= 0

> names(survobj)

[1] "n"         "obs"       "exp"       "var"       "chisq"
[6] "na.action" "call"
```