Exam 1 Study Guide

Midterm exam 1 will be open on the days **September 22nd, 23rd, 24th, 25th, and 26th**. You will take it under the supervision of a proctor approved by UAF eCampus. The exam will cover information in **Modules 1, 2, 3, and 4**. You will be given **120 minutes** for the exam and you will be allowed to use any **notes** (hard copy or electronic), a calculator, and **R/RStudio**. To that end, please find a way to take the exam in a location with a computer that runs R. If you are going to use your own laptop, you will need to show your proctor that you have disabled its WiFi capability. ***Bring said laptop to the exam fully charged, as not all areas have convenient outlet access.***

To study for the exam, you might want to make sure you can perform the tasks listed in the learning objectives for these modules, since the test will be built out of these objectives. Said another way, every task listed in the learning objectives is fair game for the test. But typically exams put special emphasis on the following short list of tasks:

- Inputting a given data set into R for further analysis
- Reading the R summary report from a fitted linear model and knowing what the various numbers mean
- Interpreting the slope and intercept of a fitted model
- Constructing and interpreting confidence intervals for the mean response and prediction intervals
- Performing inference (confidence intervals and hypothesis tests) for regression coefficients

The following pages comprise a compiled glossary of notable R functions from the first four labs.

R function glossary

- `abline()`: This function superimposes a straight line on a plot. The standard way to use it is to specify `a`, which is the line's y-intercept, and `b`, its slope. For instance, to superimpose a line that tracks the trend evident in the scatter plot of the `Orange` data, you could ask for a line with an intercept of -100 and a slope of 9:

```
plot(age ~ circumference, data = Orange)
abline(a = 16, b = 0.1)
```

This function has special utility when working with linear models. To draw the fitted SLR model on top of a scatter plot, simply call this function with the saved model object inside:

```
plot(age ~ circumference, data = Orange)
abline(Orange.model)
```

- `anova()`: The `anova()` function produces the ANOVA table summary of the fitted model. It does not give the "Total" row, but otherwise the table matches those discussed in lecture. The only required argument is the fitted model object.

- `arrows()`: This function is used to superimpose arrows on a plot. Many arrow styles are available.

- `boxplot()`: This function produces a boxplot of one or more numeric vectors. For instance:

```
boxplot(Orange$circumference, Orange$age)
```

- `c()`: This function produces a vector of the given arguments:

```
c(1, 2, 3, 5, 8, 13, 21)
```

- `confint()`: To get 95% confidence intervals on the coefficients of a SLR model, use this function. The first row is an interval for β_0 and the second row is for β_1. If desired, the confidence level can be changed using one of the function's arguments.

```
confint(Orange.model)
```

- `cor()`: This function calculates the sample correlation of two vectors of numbers.

- `cov()`: This function calculates the sample covariance of two vectors of numbers.

- `data.frame()`: This function produces a data frame. Each vector in the data frame can be named and specified as an argument. For instance:

```
data.frame(Students = c("Paul", "Diana", "Cindy"), Dish = c("Eggs benedict",
    "Oatmeal", "French toast"), Grade = c(89, 90, 96))
```

- `dnorm()`, `pnorm()`, `qnorm()`, `rnorm()`: These are four R functions related to the normal distribution.

Every distribution pre-programmed into R will provide four similar functions, similarly preceded by the letters "d", "p", "q", and "r". Thus the chi-squared distribution functions are accessible as:

```r
dchisq()
pchisq()
qchisq()
rchisq()
```

The functions that begin with `d` evaluate the distribution's probability mass function or probability density function at some value. This is a topic that falls outside the scope of this course.

The function that begins with `p` calculates cumulative probabilities for a random variable with the given distribution. Specifically, if x is some fixed number and X is a normally distributed random variable with mean 0 and standard deviation 1, then `pnorm(`x`)` calculates

$$P(X \le x)$$

For example, let $x = 1$. The empirical rule from STAT 200 predicts that $P(X \le 1)$ is approximately 0.84. R provides a more precise answer:

```r
pnorm(1)
```

The principal reason for using distribution functions beginning with `p` in this course will be to calculate p-values, since p-values—whether one-tailed or two-tailed—are nothing but cumulative probabilities.

Functions that begin with `q` do exactly the inverse of what functions that begin with `p` do. In `q` functions, you specify the probability that sits to the left of some number and the function tells you what the number is. For instance, you might want to know what number x in a standard normal distribution is such that

$$P(X \le x) = 0.84.$$

You would then ask for

```r
qnorm(0.84)
```

and the answer is about 1. The principal reason for using distribution functions beginning with `q` in this course will be to get the multipliers that are used in calculating confidence intervals.

Finally, the function that begins with `r` is a random number generator. The function `rnorm()` generates realizations from a normal distribution, optionally with its mean and *standard deviation* (NOT VARIANCE) specified. If these are left unspecified, the function's default is assume a mean of 0 and standard deviation of 1. You also need to specify `n`, the number of realizations you want. Hence,

```r
rnorm(n = 15, mean = 5, sd = 2)
```

produces 15 realizations from a normal distribution with mean 5 and standard deviation 2. As stated above, if you don't write the names of function arguments, R assumes you are specifying them in the same order that they appear in the function's help file, meaning you can simply write

```r
rnorm(15, 5, 2)
```

to sample from the same distribution.

- `Effect()`: This function draws effect plots as discussed in lecture. The function resides in the `effects()` library, which is automatically opened when `alr4` is opened.

For now, only two arguments are necessary for using `Effect()`: the "focal" predictor (which is the one and only predictor in SLR models) and the fitted model object. These should be specified in an ordering that puts the focal predictor first, followed by the fitted model object.

```
Effect("circumference", Orange.model)
```

This line of code does not produce a plot. To get the plot, put the whole function inside `plot()`:

```
plot(Effect("circumference", Orange.model))
```

- Elementary functions: Naturally, R can perform simple calculations. For instance:

```
3 + 5
2 - 5
4 * 96
7/2
sqrt(19)
2^5
```

An important fact to always remember is that R is case-sensitive. Thus, none of the commands

```
Sqrt(19)
SQRT(19)
sQrT(19)
```

do anything but throw errors. Keep this in mind when you see functions or objects with names that include capital letters. They must always be typed exactly the same way in order for R to understand.

- `file.choose()`: This function opens an operating system navigation dialogue box. Once a file is selected, the function returns the path for that file. No arguments are necessary to run the function:

```
file.choose()
```

- `head()`: This function allows you to look at the "head" of a long object. The data frame `Heights` in `alr4` is long, but sometimes all you want to see is the first few rows:

```
head(Heights)
```

- `hist()`: This function produces a histogram of a numeric vector. For instance:

```
hist(Orange$age)
```

- `list()`: This function produces a list of its arguments.

- `lm()`: This function is used to fit linear models, such as SLR models. It uses formula notation and a `data=` argument. To fit a SLR to the `Orange` data, do:

```
Orange.model <- lm(age ~ circumference, data = Orange)
```

A very simple report of the model-fitting results is given by printing `Orange.model`:

```
Orange.model
```

Sometimes you want a linear model without an intercept (i.e. where the fitted line passes through the point (0,0)). The default setting in `lm()` is to give you a non-zero intercept, but you can override the default and *force* the intercept to be zero.

For example, in the `Orange` data set, it might seem odd that a standard simple linear model wants to estimate a non-zero intercept, which is the mean age of all trees with zero circumference. If the tree has no circumference, doesn't that mean the tree's age is also zero?

The intercept is forced to be zero using a `0` in the model formula in `lm()`, as in

```
lm(age ~ 0 + circumference, data = Orange)
```

Note that the fitted model still reports a slope estimate, but not an intercept estimate.

Even though the default is to include an intercept, if you should ever want to assure yourself that R will give you one, you can use a `1` in the model formula:

```
lm(age ~ 1 + circumference, data = Orange)
```

This fact will be helpful later on when we discuss random effect and mixed models.

- `log()`: This function computes the natural logarithm (base e) of its argument. For any other base, use the `base` argument.

```
log(10)
log(10, base = 10)
```

- `mean()`: This function calculates the sample mean of a vector of numbers. For example, to get the mean of the first column from the `Burt` data frame, you could write:

```
mean(Burt$IQbio)
```

The dollar sign allows you to specify which column of the `Burt` data frame you want.

- `names()`: You can extract the names from an object with `names()`.

- `pairs()`: This function produces a scatter plot matrix, like those discussed in the textbook. All that is needed for argument is the name of a data frame whose columns are all numeric vectors. For instance:

```
pairs(Orange)
```

- `par()`: In addition to providing documentation to all of R's plotting parameters, this function can be used to specify certain parameters before calling the `plot()` function. For instance, the following command shrinks the top margin of whatever plots are made afterwards.

```
par(mar = c(5.1, 4.1, 2.1, 2.1))
```

- `plot()`: This function produces scatter plots when two quantitative variables are given as arguments, either in sequence (x followed by y) or in formula (y ~ x). Use the `data=` argument to specify the data frame where the variables are found. Other possible arguments include `xlim=`, `ylim=`, `pch=`, `xlab=`, `ylab=`, and `main=`.

- `polygon()`: This function is used to superimpose polygons on a plot, and can be shaded different colors. The first argument is a vector of x-coordinates of each vertex of the polygon, and the second argument is a vector of the vertices' y-coordinates. For instance, return to the `Orange` data and suppose there was a need to draw a blue box on top of the lowest five points. This could be done with:

```
plot(age ~ circumference, data = Orange)
polygon(c(28, 28, 35, 35), c(100, 136, 136, 100), col = "blue")
```

- `predict()`: This function produces predictions for new values of the predictor(s). For instance, in the `Orange` data set, we may want to predict the circumference of a tree with trunk circumference of 175 mm. This is done with `predict()`, where the first argument is a fitted model object (in this case, `Orange.model`). The second argument should be a data frame that contains the predictor values for which predictions are desired (eg 175). The full syntax would look like

```
predict(Orange.model, data.frame(circumference = 175))
```

This function is simply plugging 175 into the fitted model and calculating the result:

$$\tilde{y} = \hat{\beta}_0 + \hat{\beta}_1 x = 16.6036 + 7.8160 * 175 = 1384.404.$$

Through the `interval=` argument, you can additionally ask for either:

1. a 95% confidence interval for the mean response at some value(s) of the predictor(s), or
2. a 95% prediction interval for a new response at some value(s) of the predictor(s)

The syntax requires you to specify whether you want a confidence interval or a prediction interval. If the tree age of interest is 175, then the two intervals are given by

```
predict(Orange.model, data.frame(circumference = 175), interval = "confidence")
predict(Orange.model, data.frame(circumference = 175), interval = "prediction")
```

As with `confint()`, the confidence/prediction level can be adjusted inside the function.

- `read.csv()`: This function can be used to import data from Excel into R Studio. Suppose a data frame exists in an Excel spreadsheet where each variable is a different column and each observation is a row. The spreadsheet should be "Saved As" a CSV (comma delimitted) file format. Let the file be called "Unicorn data" and reside in the Windows directory "C:/Users/sdgoddard/Documents". Then the file could be imported with the following command:

```
unicorns <- read.csv("C:/Users/sdgoddard/Documents/Unicorn data.csv")
```

This example assumes that there are not variable names in the first row of the spreadsheet. If the spreadsheet contains a row of variable names, then you should tell R of it:

```
unicorns <- read.csv("C:/Users/sdgoddard/Documents/Unicorn data.csv", header = TRUE)
```

If you ever need help finding the file path for a given file, use the `file.choose` function.

- `sd()`: This function calculates the sample standard deviation of a vector of numbers.

- `seq()`: This function creates a vector of sequential numbers between two endpoints, as in:

```r
seq(7, 13)
```

The same sequence can be made with `:`, as in:

```r
7:13
```

- `str()`: This function allows the user to see the structure of an object.

- `summary()`: This function produces a summary of an object. Different types of objects receive different kinds of summaries. For instance, a five-number (well, technically six-number) summary is produced for numeric vectors:

```r
summary(c(0, 1, 0, 0, 1, 615))
```

Importantly, this function allows for the generation of very helpful reports when a linear model is fit with the `lm()` function. Consider `Orange.model`:

```r
summary(Orange.model)
```

- `table()`: This function produces a frequency table of an R object.

- `tail()`: This function is similar to `head()`, but displays only the last few rows of an object.

- `typeof()`: This function checks a vector's type.

- `var()`: This function calculates the sample variance of a vector of numbers. For example, to get the variance of the second column from the `Burt` data frame, you could write:

```r
var(Burt$IQfoster)
```

- `vcov()`: This function returns the variance-covariance matrix of the parameter estimates in a fitted linear model. The main diagonal of the following output reports the variances of $\hat{\beta}_0$ and $\hat{\beta}_1$, respectively, while the off-diagonals report the covariance between them.

```r
Orange.model <- lm(age ~ circumference, data = Orange)
vcov(Orange.model)
```