Data with Two Way Categorical Interactions

michael a. rieger

December 19, 2016

marieger@gmail.com

Washington University in St. Louis, Department of Genetics

Contents

Introduction **2**
To Do . . . 3
Materials . . . 3

Warm Up **3**
Linear Algebraic Operations . . . 3
The Dot Product . . . 4
The Element-wise Product (The Hadamard or the Schur Product) . . . 5
Matrix Multiplication . . . 7
The Identity and the Inverse . . . 9

Data Simulation **12**
Paper & findings . . . 12
Step 1: Use Allele Frequencies to Make Fake People . . . 13
Step 2: Make left skewed Emotional Support Seeking Score data . . . 14

Linear Modeling Review **16**
The Quadratic Loss Function and the Traditional Hypothesis Testing Framework . . . 16
Ordinary Least Squares (OLS) . . . 18
Using Wilkinson Notation and the lm function . . . 20
Design Matrix . . . 22
Contrast Coding for Categorical Variables . . . 23
Variance-Covariance Matrix & Cross-Correlation Between Predictors . . . 27
Dealing with Interaction terms to compute the Group Means and Standard Errors . . . 29

Model Fitting **33**
Fitting model with *lm* and omnibus ANOVA testing . . . 33
Type II F tests . . . 34
Type I F tests . . . 35
Type III F tests . . . 35

Identifying Interesting Post-Hoc Hypotheses **36**
Hypotheses as Linear Combinations of Model Coefficients 36
Computing Hypothesis Estimates & Confidence Intervals 39
Performing Multiple Hypothesis Testing Adjustments with *multcomp()* with single-step methods (e.g. Tukey's Honestly Signficant Difference, Dunnett) 40

Summary And Final Thoughts **43**
On p-values 43
What to report 44
On Multiple Testing Adjustments 44

Introduction

In this exercise you will learn how to ask typical questions for grouped data into multiple levels of categories. By the end, you will have comfort with:

- Basic linear algebraic operations in R on vectors & matrices
- Basic data simulation
- Data frame construction and specifying factors and factor levels
- Understanding the Loss Function
- Review of the Ordinary Least Squares (OLS) parameters and solution
- Understanding the OLS Design Matrix
- Understanding the OLS Covariance Matrix
- Using Wilkinson Notation to specify model components
- Fitting with lm()
- Omnibus testing with anova()
- Specifying hypotheses of interest as linear combinations of OLS coefficients
- Matrix summary of hypotheses of interest
- Computing hypothesis estimates and confidence intervals
- Adjusting for multiple hypothesis testing using single-step methods and the *multcomp* package
- Summary tables and reports

By the end of this exercise, you will also be comfortable using the following functions in R

- lm(), anova(), Anova() (from the *car* package), vcov(), cov2cor(), model.matrix(), pt(), rt(), rnorm(), expand.grid(), hist(), par(), tidy() (from the *broom* package), glht() (from the *multcomp* package), and some others

To Do

Before beginning this exercise, you must install the **car, broom,** and **multcomp** packages. To do so, make sure you are connected to the Internet. From the R Console:

```
> install.packages{"car"}
> install.packages{"broom"}
> install.packages{"multcomp"}
```

Materials

You need a computer with at least R installed, though Rstudio will be used as the main development environment. If you need more information about how to install these pieces of software, ask the great Answer Box called the Internet.

Warm Up

Linear Algebraic Operations

We will spend a bit of time before we begin making sure we are comfortable with all the various linear algebraic operations that are useful in R, as well as identifying some pitfalls where mistakes and bugs like to live.

In R all array-like objects ("lists of things") can be vectors or matrices. Matrices have two dimensions: rows and columns. Vectors only have 1 dimension, a length. Let's make a vector and a matrix:

```
> x = 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
# x is listed horizontally, but we will treat it as if it were a N x 1
    column.
```

This is probably a little too basic for some of you. That's fine. What I want you note is that R displays the contents of x (the numbers 1 through 10, here) as a list that stretches off horizontally. **In linear algebra we need to consider whether a vector is a row (1 x N) or a column (N x 1) because this matters for the way linear algebraic operations work**. For the most part, **R determines under the hood whether a vector should be a row or a column, depending on the application**. But sometimes letting R guess the orientation of the vector has unintended effects. For this reason:

> **R will list the contents of a vector horizontally, but we will assume under the hood that it is ALWAYS a N x 1 column**.

So a vector is always a column, which means it is a matrix with a single column. The length of a vector is given by the length function. If we want to force R to treat a vector as a row instead, we can transpose it:

```
> t(x)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    2    3    4    5    6    7    8    9    10
```

Now, unlike before, you can see we have both a row index and column indices as well, as now we have a 1x10 row vector.

As a corrolary to this is a description of how to make a matrix. There are a number of ways to specify a matrix in R. The simplest is using the matrix function:

```
> A = matrix(data=1:200, nrow = 20, ncol = 10)
> A       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
  [1,]     1   21   41   61   81  101  121  141  161   181
  [2,]     2   22   42   62   82  102  122  142  162   182
  [3,]     3   23   43   63   83  103  123  143  163   183
  [4,]     4   24   44   64   84  104  124  144  164   184
  [5,]     5   25   45   65   85  105  125  145  165   185
  [6,]     6   26   46   66   86  106  126  146  166   186
  [7,]     7   27   47   67   87  107  127  147  167   187
  [8,]     8   28   48   68   88  108  128  148  168   188
  [9,]     9   29   49   69   89  109  129  149  169   189
  [10,]   10   30   50   70   90  110  130  150  170   190
  [11,]   11   31   51   71   91  111  131  151  171   191
  [12,]   12   32   52   72   92  112  132  152  172   192
  [13,]   13   33   53   73   93  113  133  153  173   193
  [14,]   14   34   54   74   94  114  134  154  174   194
  [15,]   15   35   55   75   95  115  135  155  175   195
  [16,]   16   36   56   76   96  116  136  156  176   196
  [17,]   17   37   57   77   97  117  137  157  177   197
  [18,]   18   38   58   78   98  118  138  158  178   198
  [19,]   19   39   59   79   99  119  139  159  179   199
  [20,]   20   40   60   80  100  120  140  160  180   200
```

In this method, I gave a vector of the numbers 1:200, and I specified that I wanted a matrix that was 20 rows by 10 columns. As you see, each column is 1 sequence of 20 numbers, and they are assembled column-wise . Other ways include using the **rbind()** and **cbind**() functions, which you may have already used.

The Dot Product

The dot product is a simple kind of matrix multiplication. We describe it as the product of taking each element of a first vector and multiplying it by the corresponding element in the second vector, and then adding it all up. In math, it looks like:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = 1*1 + 2*2 + 3*3 + 4*4 = 30$$

In statistics lingo, the dot product of a vector with itself is a kind of **sum of squares.** In physics, the square root of the dot product of a vector with itself is called the **magnitude** of a vector. If **x** and **y** are two vectors, we write:

$$x \cdot y = x^T * y$$

In English we read this as, “The dot product of x and y is computed as matrix multiplication of the transpose of x by y”. One thing to note is that the **inner** dimensions match. I.e., if x and y are both N-long, we have:

$$1xN * Nx1$$

and you can see that the number of columns of the first term match the number of rows of the second term. The result of this operation is a “matrix” which has dimensions **1x1** (1xN*Nx1 = 1x1), or a single number.

In R, if we have two vectors of the same length, we can use the **transpose** (t()) and the matrix multiplication operator **%*%** to produce a dot product:

```
> x = 1:5
> y = 6:10
> xy = t(x)%*%y
> xy
     [,1]
[1,]  130
```

Because xy is the result of matrix multiplication, you can see R lists it as having 1 row and 1 column, rather than just a single number. If you want to remove the “matrix” nature of xy, you can use the as.numeric function:

```
> as.numeric(xy)
[1] 130
```

Though mostly this isn’t necessary.

You will run into an error if the inner dimensions do not match. I.e., if the vectors are different lengths:

```
> x = 1:5
> y = 6:12
> t(x)%*%y
  Error in t(x) %*% y : non-conformable arguments
```

The error **non-conformable arguments** means dimensions do not match the right way to do matrix multiplication. I.e., the number of columns of t(x) do not match the number of rows of y.

The Element-wise Product (The Hadamard or the Schur Product)

Unlike the dot product, an element-wise product just takes two vectors of the length N, and makes a third vector, where each element is just the product of corresponding elements:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 6 \end{bmatrix}$$

In R, we just take two vectors of the same length and use the normal scalar multiplication operator ***:**

```
> x = 1:3
> y = 2:4
> x*y
[1]  2  6 12
```

A simple kind of product of this sort is where you just multiply each element of a vector by some scalar α:

$$\alpha \begin{bmatrix} \dots \\ x_i \\ x_{i+1} \\ x_{i+2} \\ \dots \end{bmatrix}^{(1)}_{(N)} = \begin{bmatrix} \dots \\ \alpha x_i \\ \alpha x_{i+1} \\ \alpha x_{i+2} \\ \dots \end{bmatrix}^{(1)}_{(N)}$$

```
> x = 1:5
> 5*x [1]  5 10 15 20 25
```

R allows you to perform element-wise multiplication of two vectors of different lengths. It should actually feel weird to do this based upon what I described above. What does it mean to multiply two vectors of different lengths? There is no clear way to do this. Nevetherless, R does have a method**. In my opinion you should be careful not to use this method unless you're clear it will be helpful, because it can lead to errors in your code**.

> **Tip:** Build checks into your code to make sure two vectors (or matrices) are the same dimension before multiplying them element-wise. R will happily produce an answer without crashing, but you may not have anticipated this answer.

To understand what R does when vectors are of different lengths, consider a vector that is of length N, and a vector of length that is a multiple of N.

```
> x = 1:2
> y = 1:10
> x*y
  [1]  1  4  3  8  5 12  7 16  9 20
```

What has R done?

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{bmatrix} = \begin{bmatrix} 1*1 \\ 2*2 \\ 1*3 \\ 2*4 \\ 1*5 \\ 2*6 \\ 1*7 \\ 2*8 \\ 1*9 \\ 2*10 \end{bmatrix}$$

It creates a vector of length matching the longer of the two. Then it multiplies each element by an element in the shorter vector, alternating by which one it multiplies. In the case where lengths are exact multiples, you can see how sometimes this might be a convenient shortcut to repeating the elements using rep():

```
> x = 1:2
> y = 1:10
> repx=rep(x,5)
> repx
  [1] 1 2 1 2 1 2 1 2 1 2
> y
  [1]  1  2  3  4  5  6  7  8  9 10
> repx*y
  [1]  1  4  3  8  5 12  7 16  9 20
```

However, this shortcut can get you into trouble. If you expect the dimensions to match, but there is a bug in your code and somewhere data are lost and a vector becomes shorter than the other, you will get an answer at the end no matter what. There is at the very least a warning message, but if this is in a much larger script, that warning message might get buried:

```
> x=1:3
> y=1:10
> x*y
  [1]  1  4  9  4 10 18  7 16 27 10

Warning message:
In x * y : longer object length is not a multiple of shorter object
    length
```

So for a bug free life, make sure you double check the lengths of your vectors before multiplying them!

Matrix Multiplication

Matrix Multiplication is an extension of the dot product to matrices. If you have a matrix of dimension **M rows x N columns** and you multiply it by a matrix of **N rows x R columns** you will end up with a matrix that has **M rows X R columns**. As with the dot product, the inner dimensions cancel:

$$MxN * NxR = MxR$$

The dimensions are the most important thing to keep track of, as in real life we will never compute these values by hand. It works like this:

$$\begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix} * \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} = \begin{bmatrix} (1*1+5*2+9*3) & (1*4+5*5+9*6) \\ (2*1+6*2+10*3) & (2*4+6*5+10*6) \\ (3*1+7*2+11*3) & (3*4+7*5+11*6) \\ (4*1+8*2+12*3) & (4*4+8*5+12*6) \end{bmatrix}$$

It basically creates a new matrix which is a series of all the combinations of dot products (row*column, then sum). This is ugly but gets used an awful lot in mathematical modeling.

```
> A = matrix(1:12,nrow=4,ncol=3)
> B = matrix(1:6,nrow=3,ncol=2)
> A
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

> B
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

> dim(A)
[1] 4 3
> dim(B)
[1] 3 2
> dim(A %*% B)
[1] 4 2

> A %*% B
     [,1] [,2]
[1,]   38   83
[2,]   44   98
[3,]   50  113
[4,]   56  128
```

Why does this get used in mathematical modeling? It gets used because it is a very elegant summary of **systems of equations**. Let's say we have a model like:

$$y \sim ax_1 + bx_2 + cx_3 + dx_4$$

where x1,x2,x3, & x4 are some measured predictors of a variable y. We want to find the solutions for a,b,c, and d that best correspond to our data.

One thing that you should notice immediately is that this can be reorganized as the dot product of a vector of data times a vector of coefficients:

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} * \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Let us call **y** our **response** and the vector of **x**s our predictors. For each measured response $\mathbf{y}^{(i)}$ we have data on all four **x**s. Maybe they are {*weight, height, blood pressure*, *age*}. If we have 10 people in our study, we can summarize their data in a table like:

$$\overset{PREDICTORS}{\begin{bmatrix} w^{(1)} & h^{(1)} & bp^{(1)} & ag^{(1)} \\ w^{(2)} & h^{(2)} & bp^{(2)} & ag^{(2)} \\ w^{(3)} & h^{(3)} & bp^{(3)} & ag^{(3)} \\ w^{(4)} & h^{(4)} & bp^{(4)} & ag^{(4)} \\ w^{(5)} & h^{(5)} & bp^{(5)} & ag^{(5)} \\ w^{(6)} & h^{(6)} & bp^{(6)} & ag^{(6)} \\ w^{(7)} & h^{(7)} & bp^{(7)} & ag^{(7)} \\ w^{(8)} & h^{(8)} & bp^{(8)} & ag^{(8)} \\ w^{(9)} & h^{(9)} & bp^{(9)} & ag^{(9)} \\ w^{(10)} & h^{(10)} & bp^{(10)} & ag^{(10)} \end{bmatrix}} * \overset{COEFFICIENTS}{\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}} \sim \overset{RESPONSE}{\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ y^{(4)} \\ y^{(5)} \\ y^{(6)} \\ y^{(7)} \\ y^{(8)} \\ y^{(9)} \\ y^{(10)} \end{bmatrix}}$$

Each row of PREDICTORS & RESPONSE represents a single individual's data for those variables. By matrix multiplication, you can see that this simplifies the system of equations resulting from multiplying each row of data by the coefficients, then taking the sum. Since we don't have to (nor want to) multiply this by hand, you can see that the dimensions of PREDICTORS are 10x4 (10 people, 4 predictor variables), COEFFICIENTS is 4x1 (4 coefficients), and thus RESPONSE = 10x4*4x1=10x1, a vector of 10 values. Summarizing this as a model is usually written like this:

$$y \sim \boldsymbol{X} * \beta$$

where X is a matrix of predictors, y is a vector of responses, and β is a vector of coefficients. In modeling, we try to find the best values of the coefficients that predict our data. Our hope is that these values are **non-zero** (meaning our predictors actually are correlated to our response in some way), and that they are generalizable, meaning I could take an 11th data point and arrive at a good prediction based on the values of a,b,c,&d.

The Identity and the Inverse

The last concepts I want to review before we continue are the concepts of the identity matrix and the inverse. Consider a simple algebra problem:

$$a * b = c$$

Now, we happen to know that in the special case, b=1:

$$a * 1 = a$$

Any number multiplied by 1 is just itself. 1 is called the "identity" operator for multiplication.

Matrices also have a special operator called an identity. If a matrix is square dimension (nrow=ncol), then the identity is a matrix with 1s down the diagonal and 0s everywhere else. We call this **I** and in R we can compute it using diag().

```
> A = matrix(1:16,nrow=4,ncol=4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16

> I = diag(4)
> I
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1

> A %*% I
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

As you can see, if we multiply A * I we just get A back.

Why do we need to define an operator like this?

Let's return to the algebra problem:

$$a * b = c$$

In order to solve for b:

$$b = \frac{c}{a}$$

Let's expand on this. This is the same thing as multiplying both sides of the equation by $\frac{1}{a}$, which I will denote as a^{-1}or the **inverse** of a:

$$a * b = c$$

$$a^{-1} * a * b = a^{-1} * c$$

Now, we know that:

$$a^{-1} * a = \frac{1}{a}a = 1$$

and so:

$$1 * b = a^{-1} * c$$

and

$$b = a^{-1} * c = \frac{1}{a} * c = \frac{c}{a}$$

With numbers, solving the inverse is easy. You just take 1/number. With matrices, we can define a similar problem:

$$A * B = C$$

where A, B, and C are matrices. Suppose we know A and C but not B? How do we solve for B?

$$(A^{-1} * A) * B = A^{-1} * C$$

This seems reasonable based on usual numbers. But unfortunately just taking 1/A for all elements of A doesn't give us an inverse that solves the problem. Instead, just like regular numbers, we analogize that:

$$A^{-1} * A = I$$

We say, the inverse of A times A equals the identity. If we can find a matrix A^{-1} that satisfies this, then:

$$(A^{-1} * A) * B = A^{-1} * C$$

$$I * B = A^{-1} * C$$

and because I is the identity:

$$B = A^{-1} * C$$

Solving this problem is NOT trivial. It is a solvable problem under certain conditions, such as when **A is square and it is full rank**.

Full rank means that none of the columns are redundant (i.e., no column is a linear transformation of another column. Column 2 doesn't equal 3*Column 1 and so forth). Square means nrow=ncol.

Let's try this out. Note the matrices we've been working with have columns that are linear transforms of other columns. Let's draw some random numbers from a normal distribution instead:

```
> A = matrix(rnorm(n=16,mean=6,sd=2),nrow=4,ncol=4)
> A
     [,1]     [,2]     [,3]     [,4]
[1,] 4.582060 5.082515 7.013341 5.121340
[2,] 3.546969 5.985602 3.360024 7.629071
[3,] 5.122954 5.644100 9.218038 7.772231
[4,] 8.135094 2.121487 5.516783 9.222128
```

The **rnorm** function pulls 16 samples from a distribution with a mean of 6 and standard deviation of 2, and matrix() organizes them into a 4x4. R actually does not use exact solutions to solve this problem, but relies on an algorithm called Cholesky decomposition. We won't explain this, but the function in R is the **solve** function:

```
> solve(A)
     [,1]        [,2]       [,3]        [,4]
[1,]  0.5105755 -0.05553007 -0.4560398  0.14674041
[2,]  0.3541780  0.15207789 -0.2661270 -0.09820708
[3,] -0.1052538 -0.14942809  0.2705080 -0.04591268
[4,] -0.4689049  0.10338974  0.3016850  0.02904843
> solve(A)%*%A
     [,1]          [,2]          [,3]          [,4]
[1,]  1.000000e+00 -2.775558e-16 1.110223e-16  2.220446e-16
[2,] -2.220446e-16  1.000000e+00 3.330669e-16 -5.551115e-16
[3,]  1.110223e-16  8.326673e-17 1.000000e+00  3.885781e-16
[4,]  8.326673e-17  7.632783e-16 2.775558e-16  1.000000e+00
```

You can see that the result of solve(A) %*% A is not exactly the identity. However it is **pretty darn close**. A 64-bit operating system thinks that about 2E-16 is the smallest floating point number that is non-zero. Out to 16 decimal spaces then, we basically have the identity. Here it is rounded to 10 decimal spaces:

```
> round(solve(A)%*%A,10)
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
```

Note that finding the inverse of small matrices does have some exact solutions, but these do not scale to large matrices. You can read more about it here :

https://en.wikipedia.org/wiki/Invertible_matrix

Look what happens if you do have a column which is an exact multiple of another. Let's take column 4 of A and set it equal to column 3 of A*2:

```
> A[,4]=A[,3]*2
> solve(A)
  Error in solve.default(A) :    Lapack routine dgesv: system is exactly
      singular: U[4,4] = 0
```

The error is called **singularity**. A matrix for which we cannot solve the inverse is said to be singular. In our case, by making column 4 exactly equal to a multiple of column 3, we have made a matrix which is **rank deficient**. This means that even though we have 4 columns, we really only need to know 3 columns to have all the information that is stored. If you encounter errors related to a matrix being singular or rank deficient, this means that your input data has some strange **collinearity** problems where columns are redundant. You should remove or deal with these redundant columns before continuing. This error can occur if two or more columns are also very highly correlated such that they are practically multiples of one another.

Data Simulation

We will take a break from math to work on a set of data that we'd like to model. I was trying to hunt down a good dataset for our lab with some interesting cases of interaction and I couldn't find one that was appropriate for an introduction, so instead we will simulate data from the literature.

Paper & findings

You will find the paper here: http://www.pnas.org/content/107/36/15717.full . This looks at the interaction of cultural environment and oxytocin receptor genotype on seeking emotional support from others by Kim and colleagues [1]. They genotyped 274 American and Korean participants (also including a number of Korean Americans who were genotypically more like the Korean population, but raised in an American cultural environment). They found that the presence of a G allele in the OXTR under individuals experiencing high psychological distress was correlated to increased levels of seeking emotional support in individuals in an American cultural context. (For those who are interested, the SNP is rs53576.)

This is actually a three-way interaction of categories:

[1] Kim, H.S. et al. Culture, distress, and oxytocin receptor polymorphism (OXTR) interact to influence emotional support seeking. Proceedings of the National Academy of Sciences (2010), Vol. 107, No. 36, pp. 15717-15721. DOI:10.1073/pnas.1010830107

- Genotype (having A or G at this SNP. AG/GG were similar compared to AA, thus this is a dominant effect of the G allele)
- Cultural context (American or Korean)
- Having experienced high psychological distress (Categorized as High or Low using surveys)

To score whether or not an individual was higher or lower in terms of how they sought emotional support from others, there were some multipart questionnaire (so this is self-reported data). These resulted in a score on a scale of 0 to 6, where 0 is the least seeking of emotional support from others, and 6 was the highest.

Unfortunately we don't have their data. But we have some statistics about their data. In this exercise, we will only consider the group under High Psychological Distress since we are only interested in learning about two-way interactions of data (In this case Cultural Environment x Genotype).

Cultural Context	Genotype	Mean Score	SD
American	AA	3.44	1.47
	AG/GG	4.24	1.46
Korean	AA	3.55	1.38
	AG/GG	3.01	1.45
Allele Frequency			
	G allele (AG or GG genotype)		
American (Korean)	0.5		
American (European)	0.88		
Korean	0.57		

They also tell us that Emotional Support Seeking scores were left skewed (remember, this means the peak of the histogram is towards the right, with a heavy tail to the left). European Americans and Korean Americans were approximately 50%:50% of their American data.

To model this, I used some computational magic with the help of the handy **Beta distribution,** a very flexible family of probability distributions, to model slightly skewed data, constrained to have a certain mean and standard deviation, and on a certain interval. I won't directly teach you this today, but you're welcome to inspect the code. To load this function ("rgbeta"), make sure you have the rgbeta.R file in your working directory and execute:

```
> source("rgbeta.R")
```

Ok! Let's simulate!

Step 1: Use Allele Frequencies to Make Fake People

We know the allele frequencies from their sample, so we can make any sample size we want (assuming these frequencies are close to the real frequencies in the population). Actually in their paper, I couldn't find reported values for the breakdown of subgroup (e.g., G allele,Korean environment), just the total Ns for genotypes (from which I computed the allele frequencies above). Let's presume we have sample size of 1000 for each group (which is more generous than what they have).

We use the binomial distribution to model the allele proportion. **The binomial distribution considers the # of failures ("0") vs. successes ("1") in *n* trials based on some probability of success *p*.** In this case, we will consider a "success" to be having a G allele at rs53576, and we will only consider the dominant effect of having G (i.e. not modeling both heterozygotes and homozygotes separately).

Tip: It may be useful to open the Editor in Rstudio (File > New File > R script, or CTL-SHIFT-N on a PC, maybe Cmd Shift N on a Mac?).

```
# First, let's create 50 koreans who have been raised in Korea:

kor.kor = data.frame(culture = rep("korean",times=1000),
            genotype = rbinom(n=1000, size=1, prob=0.5))
# Now, lets make 25 European Americans

am.am = data.frame(culture = rep("american",times=500),
            genotype = rbinom(n=500,size=1,prob=0.88))

# And finally, 25 Korean Americans

am.kor = data.frame(culture = rep("american",times=500),
          genotype = rbinom(n=500,size=1,prob=0.57))
```

If you take the sum(*whichever dataframe*$genotype), you will see the numbers of 1s are roughly similar to the expected probabilities, but these are random samples so the sample proportions may be a little different from run to run.

Step 2: Make left skewed Emotional Support Seeking Score data

Ok, now given each mean and standard deviation, let's make fake scores on the interval 0:6.

```
# make a vector to hold the scores:
kor.kor$ess = rep(NA,times=nrow(kor.kor))

# make the scores based on genotype:
kor.kor$ess[kor.kor$genotype==0] = rgbeta(n=sum(kor.kor$genotype==0),
                        mu=3.55,
                        sd=1.38, lims=c(0,6))$r

kor.kor$ess[kor.kor$genotype==1] = rgbeta(n=sum(kor.kor$genotype==1),
                        mu=3.01,
                        sd=1.45, lims=c(0,6))$r
# pool the americans
am = rbind(am.am,am.kor)

# make a vector to hold the scores
am$ess = rep(NA,times=nrow(am))

# make the scores based on genotype)
am$ess[am$genotype==0] = rgbeta(n=sum(am$genotype==0),
                 mu=3.44,
                 sd=1.47, lims=c(0,6))$r

am$ess[am$genotype==1] = rgbeta(n=sum(am$genotype==1),
                 mu=4.24,
                 sd=1.46, lims=c(0,6))$r

# Pool the dataframes:

ESSData = rbind(kor.kor,am)
```

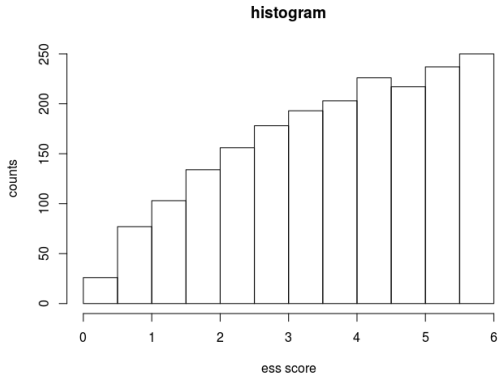
The extra "$r" after the calls to function rgbeta extracts the simulated data (the output of rgbeta is a data frame). In our simulation, I will allow the **ess** value to be fractional (i.e. 3.1 is an ok score). I don't

know that in the original data the scores were not integers (0,1,2,3..). They might have been. But if they were averaged over several questionnaire or computed some other way then maybe they were fractional.

To look at the distribution, use the hist function:

```
>hist(ESSData$ess,breaks="FD",main="histogram",xlab="ess score",ylab="counts")
```

(The option breaks="FD" specifies using the Friedman-Diaconis optimal breakpoints for the histogram's x axis (the bin size). It's my favorite.) The following should appear in the plotting window. Your histogram may look a little different from mine. These are random draws from a distribution, so every person in the room right now may have a slightly different answer:



The peak is towards the right as expected by the magic of the beta distribution. To look at how we did constraining the sample means in our random sample, use the aggregate function to create a summary table:

```
cultureXgenotype=paste(ESSData$culture,ESSData$genotype,sep=":")

summaryTable = aggregate(ESSData$ess, list(cultureXgenotype),function(a){mn=
    mean(a),sd=sd(a)})
summaryTable$expectedmean = c(3.44,4.24,3.55,3.01)
summaryTable$expectedsd = c(1.47,1.46,1.38,1.45)

colnames(summaryTable) = c("Group","sample mean","sample sd","expected mean","
    expected sd")

# From the console:
> summaryTable
     Group sample mean   sample sd  expected mean  expected sd
1 american:0     3.259498   1.490958           3.44           1.47
2 american:1     4.215379   1.444537           4.24           1.46
3   korean:0     3.561900   1.344456           3.55           1.38
4   korean:1     3.147342   1.407902       3.01           1.45
```

Not too shabby! The American AAs are actually a bit lower than the expected mean, and the Korean AG/GGs are a little higher. The nice thing about simulation, is we can re-run draws from our Beta distribution many times, and compute the output of many models, to get a sense of what "could" happen in many theoretical population samples. This the is main principle behind techniques such as Monte Carlo sampling.

Now, depending on where we are on any session we might have to end here. Make sure to save your code in the editor. Or to work on these particular simulated data again, write ESSData to a tab delimited text file:

```
>write.table(ESSData,file="ESSData.tab",sep="\t",quote="FALSE",col.names=TRUE,row.names
    = FALSE)
```

We can return to this file in the future.

Linear Modeling Review

The Quadratic Loss Function and the Traditional Hypothesis Testing Framework

In order to evaluate whether any model of data is better or worse than some other model of data, we use various criteria of model fitness. One of the simplest such criteria is something called the **Loss Function**. (This is sometimes called a Cost Function). There are different kinds of Loss functions, but the kind we will consider here is called **quadratic Loss**:

$$J(y) = \sum (y_{observed} - y_{predicted})^2$$

The function J computes a value which is the sum of squared differences between observed and predicted values (a dot product). If predicted values and observed values are very close to one another, the loss is close to 0. If they are far apart, then the loss becomes some larger positive number.

It seems intuitive that if we compare two losses computed from predictions from two different models, the better model will result in a smaller loss. The best model available will be the one that minimizes the loss.

By the way, there's no good reason to assume that the quadratic loss function is always the right metric for the problem, but it is chosen for normally distributed data. A traditional hypothesis testing framework presumes that there is some:

M_0 : Null Model

M_{alt} : Alternative Model

ANOVA is a method by which two models can be compared, if they are nested (i.e., the alternative model contains additional terms compared to the null model). Within that framework, the F test compares:

$$F = \frac{(\#observations - \#predictors_{alt})}{\#predictors_{alt} - \#predictors_0} \cdot \frac{J(M_0) - J(M_{alt})}{J(M_{alt})}$$

The numerator of this ratio is the difference in loss between the null model & an alternative model over the loss in the alternative model. If this ratio is greater than 1, this means something is gained by using the alternative model. It is weighted by the additional number of predictors in the alternative model and the number of data points, since increasing the number of data points, or increasing the number of predictors, can make an alternative model better. These two quantities are sometimes called the **degrees of freedom. Degrees of freedom** are sort of an ugly term that no one ever really explains well. These two numbers are parameters for something called an F distribution, and they control the shape of this distribution. If you know F, (#obs-#pred), & (#pred_alt - #pred_null), you can find the reference F distribution and associated p-value used to determine whether you should reject the null model in favor of the alternative based upon some arbitrary threshold (e.g., 0.05). It more strongly penalizes for additional predictors since those will artificially boost a model's fitness, and favors models with large N and few predictors.

Suppose the loss in $M_0$is 1000, and the loss in $M_{alt}$is 100. Then F is proportional to $\frac{1000-100}{100} = \frac{900}{100} = 9$. If we have 50 data points and 3 predictors in the alternative model compared to just 1 in the null, we weight this by a factor of $\frac{50-3}{2} = \frac{47}{2}$. You can see this makes the F ratio very large. We only have a few predictors and many more data points, and we have a large recovery of loss. We presume this means our predictors are doing a good job at predicting the data so it makes sense to consider the alternative model more favorably. However, if we have nearly as many predictors as we have data points, then this weakens F: $\frac{50-48}{47} = \frac{2}{47}$. You see, as you will learn, if you have many many predictors, you can over inflate a model's favorability. This is called "overfitting". In fact, if you have exactly an equal number of predictors and data points, a linear model finds a perfect solution (a perfectly terrible solution), since you are basically saying that every data point can have its own predictor. This is unlikely to generalize well to new data and an F statistic cannot be defined for such a model as $J(M_{alt})$ will equal 0.

A second thought experiment: Let's suppose our null model is that y is a function of age and that age spans from birth (year 0) to age 12 for a human. Let's say we know that age significantly is able to predict y.

$$M_0 : y \sim age|_0^{12}$$

Now, let's say we want to test whether any benefit is gained by adding a new factor. Let this new factor be height at those same ages:

$$M_{alt} : y \sim age|_0^{12} + height|_{age=0}^{age=12}$$

The problem should be obvious. Height and age are very correlated for humans between these ages. If they were perfectly correlated, we would probably get a singularity error fitting this model. But linear models can tolerate some degree of correlation. If they are somewhat strongly correlated, then we may expect that very little information is gained. Thus perhaps:

$$\frac{J(M_0) - J(M_{alt})}{J(M_{alt})}$$

will be some small number. If J(M_0) = 10.2 and J(M_{alt}) = 10, then we find that F $= \frac{n-2}{1}\frac{0.2}{10} = (n-2)*0.002$. If N were very very large, we might observe a significant effect but with small N we will not. However, because age and height are correlated, the model:

$$y \sim age$$

and another model:

$$y \sim height$$

are both probably decent models. Both age and height significantly predict our response. There are different approaches to the problem of multiple models with similar fitness. If you enumerate all such models and compute their predictions separately, you can average the final values of the predictions. This avoids having to choose which is better, and is a type of **ensemble** method. We will explore ensemble methods such as Bayesian Model Averaging in future workshops.

To summarize:

- Specifying the null model can greatly impacts judging the significance of competing models.
- F tests are only defined if models are nested with increasing complexity
- F tests judge models with a high N and low # of predictors favorably, and penalize models with large numbers of predictors. Increasing your N and reducing your numbers of predictors is a strategy to "gain statistical power". This is in quotes because usually people say this without really knowing

what they mean. Here it means boosting the ability of the F test to find any of your predictors to be a significant predictor of the response.

- When predictors have strong correlation, a predictor can be judged insignificant when in fact it may be equally a significant predictor of a response.

In typical linear models, the null model for F tests is specified as one in which the best predictor of the data is just the mean of the data. This is often abbreviated as:

$$M_0 : y \sim 1$$

where the "1" indicates that only a constant should predict y, rather than another set of data. In this case the constant is the mean ($y \sim \mu$).

Although we cannot compute a true F ratio unless the models are nested (because of the weighting term for the number of additional predictors), we can use the Loss function to compare different possible constant predictors of a set of data. In R, let us randomly sample some normally distributed data and see whether or not the median or the mean is a better predictor of our sample:

```
> j = function(x,p){sum((x-p)^2)}
> x = rnorm(n=25,mean=5,sd=1)
> meanx=mean(x)
> medianx=median(x)
> j(x,meanx)
  [1] 25.85405
> j(x,medianx)
  [1] 26.21998
```

You can see that using the mean or the median results in a similar value for the loss function J, though the mean is slightly smaller. If instead we used some totally random other numbers to predict the response:

```
> p = rnorm(n=25,mean=2,sd=0.5)
> j(x,p)
  [1] 311.6864
```

In this case, I generated a new vector p of the same length of x from a different distribution. You can see that the loss associated with using p as the predicted values for x is an order of magnitude higher than using either the mean or the median. This suggests that either of those is a better predictor than totally different random numbers.

Ordinary Least Squares (OLS)

We have discussed the Loss function with how it relates to comparing model fitness. However, it is also used to solve the ordinary least squares problem. If we want to fit some coefficients β that satisfy a model:

$$y \sim X\beta$$

and we want to find these β such that they minimize the quadratic loss:

$$\beta = \beta_{min(\sum(y-X\beta)^2)}$$

(where y are the observed values, and Xβ are the predicted values), then the OLS solution is the solution that minimizes the quadratic loss. That solution is:

$$\beta_{min(J(y))} = (X^T * X)^{-1} * X^T * y$$

In R code, this is:

```
> beta = solve(t(X) %*% X) %*% t(X) %*% y
```

This is the case assuming that the quadratic loss function was the thing we wanted to minimize in the first place. The values $y - X\beta$ are called **residuals** are assumed to be normally distributed with a mean of 0, and a variance σ^2. This in math abbreviation is written:

$$\varepsilon \sim \mathcal{N}(0, \sigma^2)$$

In computer simulation, it is equivalent to drawing random numbers from a normal distribution: rnorm(n,mean=0,sd=σ
Once we have solved for β, the value of σ^2 is computed as:

$$\sigma^2 = \frac{1}{n-k}\sum(y - X\beta)^2 = \frac{1}{n-k}\varepsilon^T\varepsilon$$

I.e., the variance term is the loss function given y,X, and β weighted by the number of rows of X (the number of data points) minus the number of columns of X (the number of predictors). This should look familiar: it is one of the weighting factors in the F test. In R code we can compute this as:

```
> eps=y-X%*%beta
> j.eps = sum(eps^2)
> s2 = j.eps/(nrow(X)-ncol(X))
```

Although OLS has an exact answer, you can imagine that finding the values of β could be solved computationally:

```
# Below some pseudocode:
initialize beta values
# maybe equal to all zeros to start

compute J(y)

compute first and second derivatives of J(y) with respect to beta (J_1(y), J_2(y)) to determine whether we are at a minimum

# Differentiating dJ/dbeta when there are multiple beta values requires computing a gradient
# as well as using a simultaneous updating rule

# J_1(y)=0 is a min or a max, J_2(y) > 0 is concave up.

# In practice, we may never reach exactly zero, so set "tolerance"
tol = 1E-6 # or some small number

while |J_1(y)| > tol & J_2(y) < tol
  update beta by some rule to approach the minimum (e.g., Newton Raphson algorithm)
end

# If while loop exits, we have converged on the final values of beta

# If we have a bad update rule, the while loop may diverge instead.
```

This is useful because minimum/maximum finding algorithms are well established. For non-linear types of model fitting, this approach is employed. For linear modeling, the converged values of β should be identical to or within a very small amount of the OLS estimates.

Using Wilkinson Notation and the lm function

By now, you have probably used the lm function to fit a linear model at least sometimes. To pass a model function to lm, we use a notation written by Wilkinson and Rogers [2]. In this notation, we consider the variable designations for **X** and we don't explicitly write out β.

Let's consider a model predicting some response **y** using an intercept, age, and weight. In matrix form:

$$X * \beta = \begin{bmatrix} \dots & \dots & \dots \\ 1 & age^{(i)} & weight^{(i)} \\ 1 & age^{(i+1)} & weight^{(i+1)} \\ \dots & \dots & \dots \end{bmatrix} * \begin{bmatrix} \beta_{intercept} \\ \beta_{age} \\ \beta_{weight} \end{bmatrix} \sim \begin{bmatrix} \dots \\ y^{(i)} \\ y^{(i+1)} \\ \dots \end{bmatrix}$$

Note the way that I have written X. The first column is a column of all 1s. This is because after matrix multiplication, each 1 will be multiplied by the intercept, such that the full linear form is:

$$y = \beta_{intercept} + \beta_{age} age + \beta_{weight} weight|_{\text{ith row}}$$

Because summarizing the intercept along with the other predictors in matrix form requires using a column of 1s, the number "1" in Wilkinson notation refers to the intercept. In Wilkinson notation, we write an abbreviated form of the linear equation omitting the βs and leaving only the predictors. The equation above might be written:

$$y \sim 1 + age + weight$$

[2] Wilkinson, G.N., and Rogers, C.E. Symbolic Description of Factorial Models for Analysis of Variance. Journal of the Royal Statistical Society. Series C (Applied Statistics) Vol. 22, No. 3 (1973), pp. 392-399. DOI: 10.2307/2346786

This is how a model form can be passed to R using the lm function. R actually assumes that an intercept is always fitted, so equivalently:

$$y \sim age + weight$$

means the same thing. This is important to realize, because sometimes you want to assume an intercept is zero. For example, if I am fitting a protein standard curve and I presume that 0 mg/mL of protein means 0 absorbance, I may choose to fit a model like:

$$protein.conc \sim absorbance - 1$$

where the "-1" tells R to remove the intercept from the model fit and assume the intercept is 0.

You may recognize at this point that I said earlier that a typical null hypothesis model is a model that fits only an intercept:

$$M_0 : y \sim 1$$

It turns out that the least squares solution for $\beta_{intercept}$ for such a model is always the **mean of the data** μ. Thus, in the absence of any other predictors being fitted, the predictor of any random vector of data that minimizes the quadratic loss is in fact the mean of the data. If the data are skewed or otherwise not "well behaved", it turns out the mean will still minimize the quadratic loss function, it's just that the quadratic loss function may not be the best criterion for optimization. It all goes back to the loss function. The OLS solution always minimizes the quadratic loss. However, only if the data are normally distributed do we think that (a) the quadratic loss is a good criterion, and (b) that the minimum is a global minimum.

To show that this is the case, let's fit M_0 for a vector of random data. Load the **broom** package so we can use the **tidy** and **glance** functions to quickly pull out model components:

```
> library(broom)
> A = data.frame(y=rnorm(n=100,mean=10,sd=2))
> mdl = lm(y~1,data=A)
> tidy(mdl)
     term estimate std.error statistic      p.value 1
(Intercept) 9.969101  0.205206  48.58096 7.11555e-71
```

The tidy function from the broom package is more handy than the summary() function you may be used to, because it always outputs a data frame. This means all the columns are accessible by normal **$** notation, or using indexing, and can be easily exported using the write.table function. Note the estimate for the intercept 9.691. Also note that the standard error is the same as would be computed by $\frac{sd}{\sqrt{n}}$, and the t statistic 48.58 is from the hypothesis test that the estimate is equal to 0. The estimate for the intercept should be identical to the mean of the data:

```
> coef(mdl)
  (Intercept)
  9.969101
> mean(A$y)
  [1] 9.969101
```

Wilkinson notation is flexible for other types of models. For example, we can also define polynomial terms:

$$y \sim 1 + x + x\text{^}2 + x\text{^}3$$

in order to fit a polynomial function, since polynomials are still linear forms. A separate coefficient and an intercept will be determined for each term.

You can also model functions that are linear for two predictors as well as considering the multiplication of those two predictors. This is called an "interaction":

$$y \sim 1 + x + q + x : q$$

using the ":". Shorthand for this is using the asterisk *:

$$y \sim x * q$$

This notation will fit the same model as 1+x+x:q. Using an F test, we can compare:

$$y \sim 1 + x + q$$

to the model:

$$y \sim 1 + x + x : q$$

to determine whether or not the interaction term reduces the associated Loss.

Design Matrix

In summary form, the matrix of data values used as predictors + a column of 1s for an intercept **(X)** is called the **Design Matrix**. It is called that because at least under the hood, it is the way that we solve the OLS problem. We have to specify X in order to solve for β. If we want an intercept in our model, we add a column of 1s. If we don't, we leave it off. If we want to model an interaction of two predictors, we create a column that multiplies them together. If not, we leave this column out. Thus, our problem is solved based up on how we design **X**.

In the case of numerical data, we can imagine that each predictor of the response is a single column of X. When we fit a polynomial such as:

$$y \sim \beta_{intercept} + \beta_{linear} x + \beta_{quad} x^2$$

we create a matrix like:

$$\boldsymbol{X} = \begin{bmatrix} ... & ... & ... \\ 1 & {}^{(i)}x & {}^{(i)}x^2 \\ 1 & {}^{(i+1)}x & {}^{(i+1)}x^2 \\ ... & ... & ... \end{bmatrix}$$

Note that this does not cause a singularity error because x^2 is a non-linear transformation of x, thus this is still a full-rank matrix and the inverse $(X^T X)^{-1}$ can still be computed. We may choose to fit this model because we believe that our data are a parabola with respect to the predictor x. However, x & x^2 are two parts of the same underlying predictor. We would plot x on the x-axis and predicted values on the y. It's not like x and x^2 are independent predictors. Thus, if our question is "Is x a significant predictor of y" and we believe that "Either x significally predicts y quadratically (i.e., a parabola) or x doesn't predict y at all", then we want to make sure for our F test, that we compare:

$$M_0 : y \sim 1$$

$$M_{alt} : y \sim 1 + x + x\text{^2}$$

Thus factor x is composed of two columns of the design matrix: the linear and quadratic forms. I'm considering the combined effect of adding x and x^2 as a single predictor, even though for solving OLS both terms must be entered into the design matrix as if they were two independent predictors.

If instead my question is "Is the quadratic form better at predicting y than the linear form?" then I want to compare each term separately as:

$$M_0 : y \sim 1 + x$$

$$M_{alt} : y \sim 1 + x + x\text{^}2$$

Again, our research question guides our choice of null model for comparison.

Grouping together different columns of X for a test is in fact the way that we use linear models to deal with non-numeric data, such as categories. In a moment we will discuss this method, but for a categorical factor **group**, fitting the model:

$$y \sim 1 + group$$

is equivalent to fitting a model with an intercept and a column of the design matrix for each of the total number of groups minus 1. Nevertheless, we can group these columns together in an F test to consider them a single predictor.

Contrast Coding for Categorical Variables

Using the OLS framework for dealing with categories is useful. If you recall, fitting an intercept-only model results in using the mean of the data to predict the data. When we code categorical variables appropriately, OLS can return to us interesting quantities, such comparing using group means for prediction, rather than a pooled mean. By the way, we can compute that anyway without using OLS. However, when categories interact or otherwise get complicated, OLS provides a convenient notation and system that would otherwise be difficult to deal with. Because of this, nearly every statistical package I can think of uses OLS to compute the F ratios associated with using categories to predict data, rather than using classical ANOVA formulas which get very ugly as the number of categorical predictors increases. OLS provides a unified framework for both numerical and categorical modeling that has become extremely widespread.

Categorical variables in R are called **factors**. Factors have **levels** which represent the number of groups in that factor. Now suppose we have a factor with three levels, A, B, and C. You might do something that assigns a value of 1,2,and 3 for each category. However, this assumes mathematically that group C is "greater" than group A, which is meaningless (unless our variables are **ordinal**). This can do weird things to the interpretation of the β values. Furthermore, why use 1, 2, and 3 instead of 10,20, 30? Or 2,158,and 34?

Because of the ambiguity of such schemes and the difficulty in interpreting the output of the model, a common method for coding variables is to use **dummy variable coding**. Dummy variables create a column of data for each level of the factor. Then we assign a value of **0** if the data point is not a member of that group, and a **1** if it is:

Original Labels		Group A	Group B	Group C
...		...	...	...
B		0	1	0
A	$\rightarrow Dummy.Coded =$	1	0	0
A		1	0	0
B		0	1	0
C		0	0	1

As you see, each of these forms a nice variable, and no column is a direct linear transformation of any other column. If we were to use Dummy.Coded as our design matrix, the OLS estimates for β_{groupA}, β_{groupB}, and β_{groupC} would be the mean of each of these three groups. You can think of this as each group getting its own intercept.

However usually our research question is: **Are we better off splitting the data into groups (i.e., is there a significant effect of the groups) than just predicting by the pooled mean (no splitting)**. To do so, as you recall, requires that we fit a null model that contains only an intercept, and that our alternative model adds the group data. Let us add an intercept to the Dummy.Coded matrix:

Intercept	Group A	Group B	Group C
1	...	...	...
1	0	1	0
1	1	0	0
1	1	0	0
1	0	1	0
1	0	0	1

When we do this, now we have a singularity problem. You see, if we take Column 1 and subtract columns 3 & 4 we get column 2. Similarly, if we take Column 1 and subtract columns 2 and 3 we get column 4. This is a matrix of rank 3 but there are 4 columns.

The design matrix for such a problem removes a single column. The column you end up removing is called the **reference** and its choice is arbitrary. Let us call group A the reference:

$$X = \begin{array}{|c|c|c|} \hline \text{Intercept} & \text{Group B} & \text{Group C} \\ \hline 1 & ... & ... \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ \hline \end{array}$$

This type of scheme is call **contrast coding.** The reason for this will become apparent. When we solve the OLS estimates for each of these predictors, this very specific result occurs:

$$\beta_{intercept} = \mu_{reference}$$

and

$$\beta_B = \mu_{groupB} - \mu_{reference}$$

and

$$\beta_C = \mu_{groupC} - \mu_{reference}$$

Thus each of the OLS coefficient estimates (aside the intercept) is a difference between means. The full equation is:

$$y \sim X * \beta = \mu_{reference} + \beta_B B\{^{=1 \text{if member}}_{=0 \text{ if non-member}} + \beta_C C\{^{=1 \text{if member}}_{=0 \text{ if non-member}}$$

When we perform an F test, we compare:

$$M_0 = y \sim 1 : \text{"Best predictor is the pooled mean"}$$

to

$$M_{alt} = y \sim 1 + group$$: "Best predictor of each group is its own mean"

If we reject M_0 in favor of M_{alt} we believe that at least 1 group must be different from all the others. In Wilkinson notation, we just write "y~1+group" (or just "y~group" since R knows to add an intercept by default). In R, when a variable is coded as a factor, it knows to perform dummy coding, and it uses the first level as the reference level. In R, that level is assigned by whichever is first encountered in the data. You as the researcher may specifically want to consider one of the groups as the reference instead of some other.

Before we continue, let's make sure we understand why this linear model works. If a data point is in Group A, then the B and C columns both = 0. Then we have:

$$\text{If Group A: } y \sim \beta_{intercept} + \beta_B \cdot 0 + \beta_C \cdot 0 = \beta_{intercept} = \mu_A$$

$$\text{If Group B: } y \sim \beta_{intercept} + \beta_B \cdot 1 + \beta_C \cdot 0 = \mu_A + (\mu_B - \mu_A) = \mu_B$$

$$\text{If Group C: } y \sim \beta_{intercept} + \beta_B \cdot 0 + \beta_C \cdot 1 = \mu_A + (\mu_C - \mu_A) = \mu_C$$

Thus using our coding scheme, we return the expected predicted values. The bonus is that the t-test that each β_B or β_C is different from 0 is the same as determining computing a t-test to determine whether a difference between two means is non-zero.

What about estimating the difference between B and C? This is actually just a combination of the other beta values:

$$\mu_C - \mu_B = (\beta_{intercept} + \beta_B \cdot 0 + \beta_C \cdot 1) - (\beta_{intercept} + \beta_B \cdot 1 + \beta_C \cdot 0)$$

$$= \beta_{intercept} + \beta_C - \beta_{intercept} - \beta_B = \beta_C - \beta_B$$

So if we define a new term $\beta* = \beta_C - \beta_B$ we can estimate the difference and other associated statistics for this linear combination of beta values. (This becomes the principle behind post hoc comparison testing). Even if we didn't mean to choose A as the reference, we can obviously compute all possible pairwise comparisons as linear combinations of different β values. Thus the OLS model provides an environment within which we can easily ask any question of the data. Also because of this, the **F test returns the same answer no matter which group we call the reference**.

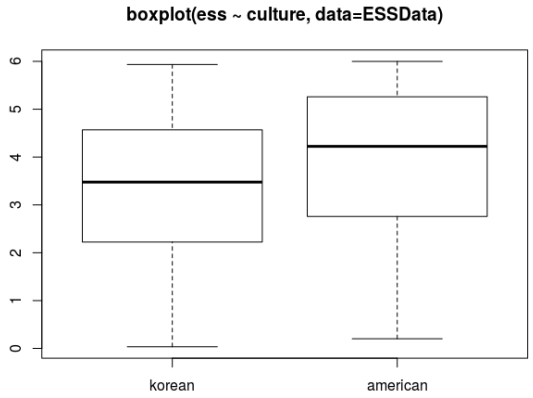
Let us return to our data frame **ESSData** from our emotional support seeking score for the Kim et al paper. Inspect the **culture** column using:

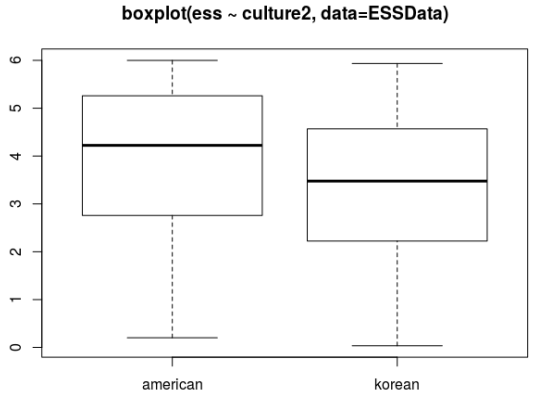
```
> head(ESSData$culture,n=20)
[1] korean korean korean korean korean korean korean korean korean
    korean korean korean korean
[14] korean korean korean korean korean korean korean
Levels: korean american

> str(ESSData$culture)
Factor w/ 2 levels "korean","american": 1 1 1 1 1 1 1 1 1 1 ...

> contrasts(ESSData$culture)
         american
korean          0
american        1
```

If instead we wanted to change the level order, we can use the **factor** function with the levels option:

```
> ESSData$culture2 = factor(ESSData$culture,
          levels=c("american","korean"))

> contrasts(ESSData$culture2)
        korean
american      0
korean        1
```

This also affects how things are ordered in plots. To plot a boxplot showing ESS score by culture or culture2, the same data are shown but with a different ordering of the x-axis:





To compute the design matrix for a model in Wilkinson notation, without manually assembling it ourselves, we use the **model.matrix** function. Here is what the design matrix looks like for the factors of genotype & culture and their interaction for the ESS data:

```
# recode genotype as a factor. Original labels are 0, 1, but replace with "AA" and "AG/GG"

ESSData$genotype = factor(ESSData$genotype, levels=c("0","1"), labels=c("AA","AG/GG"))

# Generate a Design Matrix fitting y~genotype*culture (same as y~1+genotype + culture +
    genotype:culture)

X = model.matrix(~genotype*culture,data=ESSData)



#From Console:
> head(X)
  (Intercept) genotypeAG/GG cultureamerican genotypeAG/GG:cultureamerican
1           1             1               0                             0
2           1             1               0                             0
3           1             1               0                             0
4           1             1               0                             0
5           1             1               0                             0
6           1             0               0                             0
```

The first column contains all 1s for the intercept. The second takes a 1 if the genotype has the G allele, and 0 if only the A allele. The third gets 1 if american, otherwise 0 if korean. The last column is simply the 2nd column multiplied by the 3rd column. The interaction column gets a 1 if you have both the G allele & are american, since otherwise the values for columns 2 & 3 are 0. Note that multiplying columns or raising columns to powers does not result in singularity errors. Only addition or subtraction of columns, or multiplying columns by scalar amounts will result in such an error.

Variance-Covariance Matrix & Cross-Correlation Between Predictors

When we have determined that we have predictors that seem to improve a model fit, we would then like to know something about the error associated with the model β estimates. This allows us to determine a coefficient's confidence bounds.

The coefficient errors are computed from the Variance Covariance matrix. The diagonal terms of this matrix represent the variance in the estimate of each coefficient in the model (in the case of our genotype*culture, this is 4 coefficients total), the square root of which represents the standard error in these coefficients. The co-variance terms represent the degree to which different predictors covary with their ability to predict the response. If the value is near 0, this means predictors are mostly independent of one another. If the value is large and positive, they positively co-vary and if negative then they negatively co-vary (when one is related to y going up, the other is related to y going down, etc.).

The Variance Covariance matrix is computed as:

$$V = \sigma^2 (X^T * X)^{-1}$$

where remember that $\sigma^2$is equal to:

$$\sigma^2 = \frac{(y - X\beta)^T (y - X\beta)}{n - k}$$

for n rows and k columns of X.

We will fit this with **lm** in R, but to show you the calculations, here's what it looks like to solve for V in full:

```
y = ESSData$ess
X = model.matrix(~genotype*culture,data=ESSData)
beta = solve(t(X)%*%X)%*%t(X)%*%y

# From the console:
> beta
                                    [,1]
(Intercept)                      3.6451663
genotypeAG/GG                   -0.5650124
cultureamerican                 -0.2203197
genotypeAG/GG:cultureamerican    1.3118887

# We will interpret these coefficients later, don't worry.

> eps = y-X%*%beta
> s2 = as.numeric((t(eps)%*%eps)/(nrow(X)-ncol(X)))

> V = s2*solve(t(X)%*%X)

             (Intercept) genotypeAG/GG cultureamerican genotypeAG/GG:cultureamerican
(Intercept)       0.004166979   -0.004166979     -0.004166979                      0.004166979
genotypeAG/GG    -0.004166979    0.008435181      0.004166979                     -0.008435181
cultureamerican  -0.004166979    0.004166979      0.011834221                     -0.011834221
genotypeAG/GG     0.004166979   -0.008435181     -0.011834221                      0.019010687
 :cultureamerican
```

Whenever predictors are on different scales (such as with numerical variables), or when predictors are binary (0s and 1s here), this matrix on its own can be hard to interpret. You can normalize a variance-covariance matrix to scale everything between -1 and 1 (the Pearson linear correlation). You can look up how to do this, but in R there is simple cov2cor function:

```
> cov2cor(V)
                (Intercept) genotypeAG/GG cultureamerican genotypeAG/GG:cultureamerican
(Intercept)       1.0000000    -0.7028513       -0.5933908                      0.4681788
genotypeAG/GG    -0.7028513     1.0000000        0.4170655                     -0.6661136
cultureamerican  -0.5933908     0.4170655        1.0000000                     -0.7889890
genotypeAG/GG     0.4681788    -0.6661136       -0.7889890                      1.0000000
  :cultureamerican
```

We see that the second beta (genotypeAG/GG) is negatively correlated with the intercept. This is likely because when korean/AA is associated with y going up, korean/AG/GG is associated with y going down.

The standard errors for the coefficients are the square root of the diagonal of V:

```
> sebeta = sqrt(diag(V))
> sebeta
                    (Intercept)                   genotypeAG/GG
                                                cultureamerican
                     0.06455214                      0.09184324
                                                      0.10878521
genotypeAG/GG:cultureamerican
                     0.13787925
```

To summarize the model estimates:

1. Intercept: this is the mean of the reference “Korean&AA”: 3.65 with a standard error of 0.064.

2. Beta genotypeAG/GG: this is the difference between “Korean&AG/GG” and “Korean&AA”, which has a value of -0.565 with a standard error of 0.092.

3. Beta cultureamerican: this is difference between “American&AA” and “Korean&AA”, which has a value of -0.220 with a standard error of 0.109.

4. Beta (interaction term): this is the difference between “american&AG/GG” and “korean&AG/GG”, which has a value of 1.312 with standard error of 0.138.

When there are interactions between categories, the beta values can be a little difficult to interpret on their own. However, we can use linear combinations of the beta values to evaluate nearly any operation or quantity of interest related to combinations of group means.

Dealing with Interaction terms to compute the Group Means and Standard Errors

Let’s compute the actual group means. This will provide the framework to understand how to perform post-hoc comparisons:

$$\mu_{AA\&Korean} = \beta_{intercept} + \beta_{AG/GG} \cdot 0 + \beta_{american} \cdot 0 + \beta_{american\&AG/GG} \cdot 0 = \beta_{intercept}$$

$$\mu_{AG/GG\&Korean} = \beta_{intercept} + \beta_{AG/GG} \cdot 1 + \beta_{american} \cdot 0 + \beta_{american\&AG/GG} \cdot 0 = \beta_{intercept} + \beta_{AG/GG}$$

$$\mu_{AA\&American} = \beta_{intercept} + \beta_{AG/GG} \cdot 0 + \beta_{american} \cdot 1 + \beta_{american\&AG/GG} \cdot 0 = \beta_{intercept} + \beta_{american}$$

$$\begin{aligned} \mu_{AG/GG\&American} &= \beta_{intercept} + \beta_{AG/GG} \cdot 1 + \beta_{american} \cdot 1 + \beta_{american\&AG/GG} \cdot 1 \\ &= \beta_{intercept} + \beta_{AG/GG} + \beta_{american} + \beta_{AG/GG\&american} \end{aligned}$$

To summarize this, we can use a lil matrix and do matrix multiplication to extract all the means. This is just a design matrix of exemplar cases for each combination of culture and genotype. You can do this manually by creating the matrix:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

In large data, this might be error prone to do this by hand. So instead, let’s do it semi-automatically in R with the expand.grid function.

```
> tmp = expand.grid(genotype=levels(ESSData$genotype),culture=levels(ESSData$culture))
> tmp
  genotype  culture
1       AA   korean
2    AG/GG   korean
3       AA american
4    AG/GG american
> tmpX = model.matrix(~genotype*culture,data=tmp)
> tmpX
  (Intercept) genotypeAG/GG cultureamerican genotypeAG/GG:cultureamerican
1           1             0               0                             0
2           1             1               0                             0
3           1             0               1                             0
4           1             1               1                             1
> rownames(tmpX)=c("Korean&AA","Korean&AG/GG","American&AA","American&AG/GG")
> tmpX
               (Intercept) genotypeAG/GG cultureamerican genotypeAG/GG:cultureamerican
Korean&AA                1             0               0                             0
Korean&AG/GG             1             1               0                             0
American&AA              1             0               1                             0
American&AG/GG           1             1               1                             1
```

Setting the rownames right now is a great idea! Because of matrix multiplication, these will accompany the result of multiplication, and will make a very nice summary table:

```
> tmpX%*%beta
                    [,1]
Korean&AA       3.645166
Korean&AG/GG    3.080154
American&AA     3.424847
American&AG/GG  4.171723
```

Finally, what are the standard errors associated with these means? To do this, we use the fact that any linear combination of β with factors a,b,c,&d can be expressed as:

$$\beta* = \begin{bmatrix} a & b & c & d \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$$

Let's call the vector [a,b,c,d] weights W. The Variance-Covariance associated with this is:

$$W * V * W^T$$

Our weights are a 1x4 vector. So this is (1x4*4x4*4*1) which is a 1x1 matrix, i.e., a number which is the variance associated with the linear combination. If you combine all your linear combinations in a matrix, then this generalizes to a matrix of weights. So to compute the standard errors with these means, that's simply our exemplar matrix tmpX wrapping around V:

$$tmpX * V * tmpX^T$$

The square root of the diagonal of this matrix is the vector of standard errors for these quantities:

```
# Transform V to new linear combinations of betas:
tmpV = tmpX %*% V %*% t(tmpX)
# Note, the row names move along with everything so you can tell you're doing it correctly

se.mu=sqrt(diag(tmpV))

# Compute the group means using tmpX
muSummary = cbind(tmpX %*% beta, se.mu)

colnames(muSummary) = c("mean","se")

> muSummary
                    mean          se
Korean&AA       3.645166 0.06455214
Korean&AG/GG    3.080154 0.06533147
American&AA     3.424847 0.08756279
American&AG/GG 4.171723 0.05392832
```

Hopefully you can see that any comparisons you might want to make between any two means can be computed as a linear combinations of the rows in tmpX. For example, want the statistics associated with the difference between Americans/AA and Americans/AG/GG? Just take tmpX[3,] - tmpX[4,]. What if you want the average of the Koreans across genotype vs. the average of the Americans across genotype? Just take the average of tmpX[2,]&tmpX[1,] and the average the tmpX[3,]&tmpX[4,] and so forth. We will explore these types of calculations in our post-hoc hypothesis testing section.

Although the differences between means have some cross correlation, the means for each group should be independent of one another. To show that that is the case, inspect the correlation matrix of the Variance-Covariance matrix of the linear combinations we generated:

```
> cov2cor(tmpX %*% V %*% t(tmpX))
                    Korean&AA Korean&AG/GG  American&AA American&AG/GG
Korean&AA        1.000000e+00 0.000000e+00 1.534511e-16   0.000000e+00
Korean&AG/GG     0.000000e+00 1.000000e+00 0.000000e+00   2.461846e-16
American&AA     -1.534511e-16 0.000000e+00 1.000000e+00  -7.347232e-16
American&AG/GG   2.491568e-16 2.461846e-16 1.836808e-16   1.000000e+00
> round(cov2cor(tmpX %*% V %*% t(tmpX)),5)
                Korean&AA Korean&AG/GG American&AA American&AG/GG
Korean&AA               1             0           0              0
Korean&AG/GG            0             1           0              0
American&AA             0             0           1              0
American&AG/GG          0             0           0              1
```

The second line I've rounded to 5 decimal spaces. You can see that essentially the correlation matrix is only 1s down the diagonal and zeros elsewhere, which shows that the group means are independent of one another. No group mean is a multiple or linear combination of any other group means. Knowing the mean of Korean/AA doesn't help me figure out the mean of Korean/AG/GG

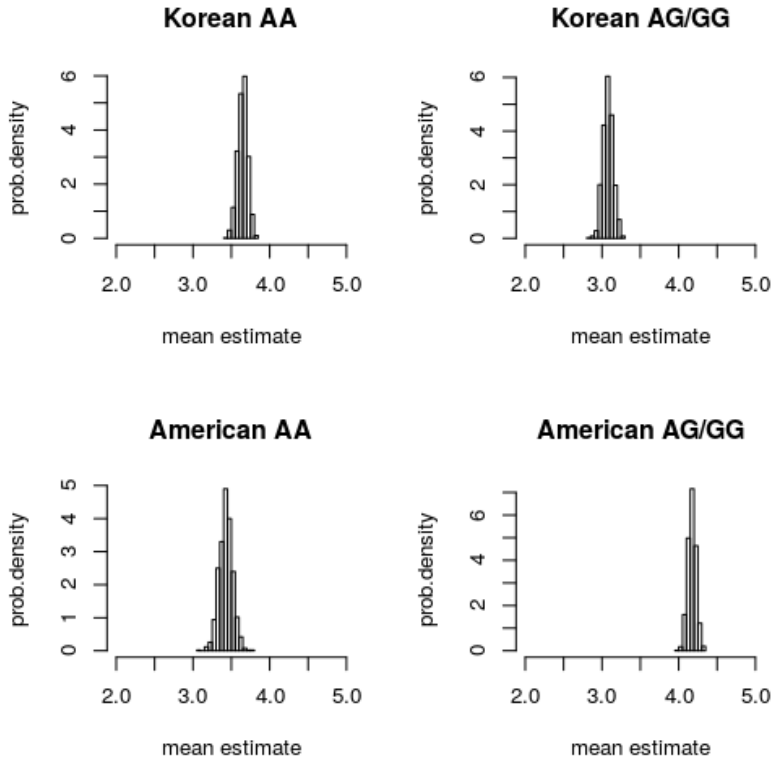
You may want to know which means are "significantly different" from one other. We will discuss this more but it's worth mentioning that the probability distributions associated with your means are:

$$estimate + se * t_{n-k}$$

This means the probability associated with predicting any of these means is a t distribution with parameter "n-k" or the rows minus the columns of X. This parameter is again called "degrees of freedom" as mysterious as that sounds. Here is a quick way to plot histograms showing the expected probability distributions for each of the group means. This by the way isn't the best way to do this, but I wanted to

introduce the rt function to draw random numbers from the t distribution. In real life you would use the **dt** function to calculate the known t distribution probability densities.

```
# make plot
rKA = muSummary[1,1] + muSummary[1,2]*rt(n=1000,df = nrow(X)-ncol(X))
rKG = muSummary[2,1] + muSummary[2,2]*rt(n=1000,df = nrow(X)-ncol(X))
rAA = muSummary[3,1] + muSummary[3,2]*rt(n=1000,df = nrow(X)-ncol(X))
rAG = muSummary[4,1] + muSummary[4,2]*rt(n=1000,df = nrow(X)-ncol(X))
par(mfrow=c(2,2))
hist(rKA,main="Korean AA",probability = TRUE,ylab="prob.density",xlab="mean estimate",
     xlim=c(2,5))
hist(rKG,main="Korean AG/GG",probability = TRUE,ylab="prob.density",xlab="mean estimate",
     xlim=c(2,5))
hist(rAA,main="American AA",probability = TRUE,ylab="prob.density",xlab="mean estimate",
     xlim=c(2,5))
hist(rAG,main="American AG/GG",probability = TRUE,ylab="prob.density",xlab="mean estimate",
     xlim=c(2,5))
```



This graph shows you the probability density (probability per unit change in x) associated with each of the mean estimates and standard errors, which are drawn from a t distribution with n=2000 - k=4 = 1996 degrees of freedom. From these we see that the American AA & Korean AA mean ESS scores are close to one another. Compared to the Korean AA, the Korean AG/GG mean is predicted to be slightly lower, and the American AG/GG is predicted to be slightly higher. Thus in the American context, it appears that the G allele influences an increase in seeking emotional support, while in the Korean context a slightly opposite effect is observed. It is important to stress at this point that these are the distributions associated **with the mean and the error associated with estimating the mean from the data.** These are not the distributions of the actual data. You can use the boxplot function with Wilkinson notation to depict the data, such as: boxplot(ess ~ culture*genotype, data=ESSData).

Model Fitting

Now that we have a thorough understanding of the mathematical framework necessary to both (a) compute model coefficients, and (b) compute quantities of interest and associated errors, we will here on out rely on built-in R functions.

Fitting model with *lm* and omnibus ANOVA testing

To fit the model from the preceeding section, we use a simple call to lm, specifying the model in Wilkinson notation:

```
# fit model
mdl = lm(ess ~ genotype*culture, data = ESSData)

# If broom library is not loaded, run library(broom)

#From console:
> glance(mdl)
  r.squared adj.r.squared    sigma statistic      p.value df    logLik      AIC    BIC
1 0.08104917    0.07966798 1.452065  58.68074 2.369688e-36  4 -3581.848 7173.696 7201.7
  deviance df.residual
1 4208.549        1996
```

The **glance** function from the broom package shows us several model statistics:

- r.squared: This is the R-squared of the model. This is defined as $R^2 = 1 - \frac{J(y)_{M_{full}}}{J(y)_{M_0}}$ where J(y)$_{M_{full}}$is the Loss function associated with the full model. J(y)$_{M_0}$is the loss with the intercept only model (null model). We interpret R-squared as the "fraction of variance in the data explained by the model". **0.08** is nothing to shake a stick at, however we see that the F test (F=58.68, p $\approx$0) indicates we should reject the null model. So although we can see a significant effect of include our genotype & culture terms, it is of low explanatory value.
- adj.r.squared: Quite like the F ratio, we can penalize R-squared by the number of data points & number of predictors, since if we have many predictors we can artificially have a very large value of R-squared. It is defined as $adj.R^2 = R^2 - \frac{k-1}{n-k}(1 - R^2)$. This is the exact inverse of the weight for F, where k is the columns of X, and k-1 is the additional predictors given by the model, and n-k the usual rows of X - the columns of X. When k is very large, R^2 is reduced by a larged amount. Here adj.r.squared is very close to r.squared because the number of data points is high and the number of predictors is low.
- sigma: the standard error of the model, or $\sqrt{\sigma^2}$.
- statistic: The F statistic comparing to the null model with only an intercept. Here it is large (58.68) and the p-value from the F distribution with parameters (k-1, n-k) is almost zero. This is the probability that an F statistic $\geq$ the one observed would occur in the F distribution under the null model.
- df: ncol(X). I think this is a bug in the broom package, because summary(mdl) clearly shows that the F statistic is computed based on ncol(X)-1, not ncol(X) as the degrees of freedom.
- logLik: the natural logarithm of the likelihood function associated with this model. We will discuss Likelihood functions at a later date.
- AIC, BIC, Deviance: Different criteria for understanding the likelihood of the model. We will discuss these in reference to model selection at a later date.

- df.residual: The “second” degrees of freedom for F, n-k or nrow(X) - ncol(X).

To understand whether the single factors of **genotype** or **culture** or their interaction impact the model fitness, we can compute F statistics comparing models lacking these terms to models containing them. These F tests comprise what is called an Analysis of Variance Table (ANOVA). This is a test which is used to determine ahead of time whether any of our factors in the model are worth considering in the first place. This method is used to limit the number of pairwise comparisons in the data that are subsequently explored. The test we will use primarily is the **Type II** ANOVA. You can compute this using the Anova() function from the **car** package. The **tidy** function from the **broom** package takes the output of Anova and makes a nice data frame:

```
# load libraries
library(broom)
library(car)

# From Console:

>tidy(Anova(mdl, type=2))
               term         sumsq    df    statistic       p.value
1          genotype     0.1311175     1   0.06218547 8.031001e-01
2           culture   167.8421648     1 79.60295913 1.011223e-18
3 genotype:culture   190.8833302     1 90.53075522 5.043449e-21
4         Residuals  4208.5490858  1996           NA            NA
```

Type II F tests

The Type II F test tests models with our terms as follows:

- y~genotype vs. y~genotype+culture : Assesses the influence of “culture” term
- y~culture vs. y~genotype + culture: Assesses the influence of the “genotype” term
- y~genotype+culture vs. y~genotype+culture+genotype:culture: Assesses the influence of the interaction term.

The tidy function shows us the following quantities:

- sumsq: The change in Loss between the competing models. The Residual sumsq term is the loss function for the full model, which serves as a denominator for all the respective F tests in an ANOVA.
- df: the change in # of predictors between competing models. The Residual df term is the nrow(X)-ncol(X) (“n-k”) from the full model.
- statistic: The F ratio
- p.value: The probability of observing F$\geq$the F ratio computed, from a distribution with parameters df and residual df under the null model (the model of lower complexity).

From this we see that there is a significant interaction term, as well as a main effect of culture. The significant main effect of culture implies that were we to average across genotypes, we would see some influence of culture alone. However, in the presence of the significant interaction, we need to first understand the respective between-subgroup relationships to understand the main effect. Averaging across genotype might reveal something interesting, or it might be something artificial to that manipulation. **Whenever an interaction is significant, you must explore it before you can interpret the main effects**. It doesn’t mean the main effect isn’t “real”, but you simply have to graph and look at parameter confidence intervals to make a subjective decision of what you think is going on.

Type I F tests

Type I tests are a little different. They assess the influence on the model by adding terms to generate increasingly complex models.

1. Compare y~1 to y~genotype to assess genotype.
2. Compare y~1+genotype to y~1+genotype+culture to assess culture.
3. Compare y~1+genotype + culture + genotype:culture to assess the interaction.

The thing is, if your predictors are at all correlated, the order in which they enter such a scheme may change the F test, as we saw trying to compare age and height in a toy example earlier. In our case we know that the prevalence of the G allele is correlated to ethnic identity. Because 50% of the Americans are of Korean descent, and 100% of the Koreans are of Korean descent, there will be more Gs in the American cultural sample where European Americans have a higher frequency of G. Thus whichever is a stronger effect (culture or genotype) may dominate depending on whether it is entered first. Type I tests are the default of the R anova() function (notice the function name all lowercase):

```
> anova(mdl)
Analysis of Variance Table

Response: ess
                  Df Sum Sq Mean Sq F value  Pr(>F)
genotype           1   12.5  12.458  5.9085 0.01516 *
culture            1  167.8 167.842 79.6030 < 2e-16 ***
genotype:culture   1  190.9 190.883 90.5308 < 2e-16 ***
Residuals       1996 4208.5   2.108


> anova(lm(ess~culture*genotype,data=ESSData))
Analysis of Variance Table

Response: ess
                  Df Sum Sq Mean Sq F value Pr(>F)
culture            1  180.2 180.169 85.4493 <2e-16 ***
genotype           1    0.1   0.131  0.0622 0.8031
culture:genotype   1  190.9 190.883 90.5308 <2e-16 ***
Residuals       1996 4208.5   2.108
```

Look at these two ANOVA tables, first for the mdl with Genotype first, and Culture second. And the second where the order is flipped. In the first, genotype is shown to have a significant effect, as we are comparing to an intercept only model. In the second, the culture term is a stronger effect and adding in genotype to a model already containing culture adds relatively little improvement to the model and the F ratio is small.

Sometimes you want to ask exactly this sort of question. In that case, the Type I test is actually useful. But the Type II test is more conservative. You compare single terms to models containing the other single terms already. So culture added to genotype still exerts a strong effect, but genotype added to culture does not (until you consider the interaction).

You can see the F statistic in the second Type I for genotype is the same as the F statistic for the genotype term in the Type II. That is because both of these are comparing y~1+culture to y~1+culture+genotype.

Type III F tests

Type III tests we will not discuss, but they are the default output of certain programs such as SPSS or MATLAB. These tests compare full models to models containing every other term, without considering

whether terms are interactions. This is not that useful for a model with interactions. To assess genotype, for example, a Type III test compares:

- y~1+culture+genotype:culture VS y~1+culture+genotype+genotype:culture

A "genotype:culture" term is sort of meaningless in the absence of a genotype term, so this method can produce wacky results for models containing interactions. Type II tests are the most appropriate for this kind of model. In models without interaction terms, Type III and Type II tests produce the same results.

Identifying Interesting Post-Hoc Hypotheses

At this point we have all the tools we need to ask the questions we want to to explore the interaction in the Kim et al. simulated data about the interaction of genotype and culture.

The ANOVA tells us that there is a significant effect of a genotype X culture interaction. This means that:

- A vs. G allele in each culture has a different ESS score (may be in unchanged in one cultural environment and different in another, or may be positive in one environment, and negative in the other), or
- Within each genotype, there is a change between cultures, or
- Something else is going on that we find "uninteresting": A genotype in Americans vs G genotype in Koreans is different from the G genotype in Americans vs. the A genotype in Koreans. This could be a source of interaction however we would have difficulty interpreting what it means if it were the only difference between the groups.

We primarily care about:

- Within genotypes, across cultures (AA: K vs. A, AG/GG: K vs. A)
- Within cultures, across genotypes (A: AA vs. AG/GG, K: AA vs. AG/GG)

This is a space of four mean differences we'd like to test. Since we also have a main effect of culture but not a robust main effect of genotype, we may also desire to add the following test:

- Averaging genotypes, across cultures (averageGeno: K vs. A)

But for now, we will leave this out and just explore the interaction. This is a space of 4 comparisons out of a total possible of 6. This is because we don't care so much about AA,K vs. AG/GG,Am and AG/GG,K vs. AA,Am as those comparisons don't help us understand our data very well.

Hypotheses as Linear Combinations of Model Coefficients

To compute these, we create a matrix like before that computes linear combinations of the coefficients. Recall our exemplar cases to compute group means:

```
> tmp = expand.grid(genotype=levels(ESSData$genotype),culture=levels(ESSData$culture))
> tmp
  genotype  culture
1       AA   korean
2    AG/GG   korean
3       AA american
4    AG/GG american
> tmpX = model.matrix(~genotype*culture,data=tmp)
> rownames(tmpX)=c("Korean&AA","Korean&AG/GG","American&AA","American&AG/GG")
> tmpX
               (Intercept) genotypeAG/GG cultureamerican genotypeAG/GG:cultureamerican
Korean&AA                1             0                0                             0
Korean&AG/GG             1             1                0                             0
American&AA              1             0                1                             0
American&AG/GG           1             1                1                             1
```

To create a matrix to define pairwise comparisons, we simply subtract the corresponding rows of this matrix from each other. For example, to compute AA:A vs. K, we would take row 3 and subtract row 1. Again, we could define all of these manually, but it is perhaps a bit nicer to do it with some built in functions.

```
# Let's use a Tukey matrix to define a set of contrasts of interest using the multcomp
    package:

library(multcomp)

# In case you didn't do it before, fit the model and make the exemplar design matrix:

mdl = lm(ess ~ genotype*culture, data=ESSData)

tmp = expand.grid(genotype=levels(ESSData$genotype),culture=levels(ESSData$culture))
tmpX = model.matrix(~genotype*culture,data=tmp)
rownames(tmpX)=c("Korean&AA","Korean&AG/GG","American&AA","American&AG/GG")

# First, define a variable that mixes genotype & culture in the order defined in our
    model (genotype*culture) and add it to our tmp variable
tmp$mixed = paste(tmp$genotype,tmp$culture,sep=":")
tmp$mixed = factor(tmp$mixed,
      levels = c("AA:korean","AG/GG:korean","AA:american","AG/GG:american"))

# Now create a Tukey matrix of all possible pairwise comparisons:

Tukey = contrMat(table(tmp$mixed),type="Tukey")

# Now subselect the comparisons of interest

Tukey = Tukey[c("AA:american - AA:korean",
                "AG/GG:american - AG/GG:korean",
                "AG/GG:american - AA:american",
                "AG/GG:korean - AA:korean"),]

# Transform the matrix tmpX by the Tukey matrix:

MDiffs = Tukey %*% tmpX

# Inspect the output of MDiffs:

> MDiffs
                    (Intercept) genotypeAG/GG cultureamerican genotypeAG/GG:
                        cultureamerican
AA:american-AA:korean         0             0                1
                                  0
AG/GG:american-AG/GG:korean   0             0                1
                                  1
AG/GG:american-AA:american    0             1                0
                                  1
AG/GG:korean-AA:korean        0             1                0
                                  0
```

You should verify that these contrasts are the correct ones. AA:american - AA:korean should be equal to tmpX[3,] - tmpX[1,]. If it is not, something is wrong:

```
# Define contrasts manually:

MDiffs2 = rbind(tmpX[3,] - tmpX[1,], tmpX[4,] - tmpX[2,], tmpX[4,] - tmpX[3,], tmpX[2,]-
    tmpX[1,])
rownames(MDiffs2) = rownames(MDiffs)

# Verify that MDiffs2 & MDiffs are the same.
```

Computing Hypothesis Estimates & Confidence Intervals

To compute the estimates for our hypotheses and the associated confidence intervals, we can use our normal linear algebra functions which we learned about before. We can use coef(mdl) to extract the β values, and vcov(mdl) to extract the Variance-Covariance matrix, rather than solving them manually. We can perform t-tests that these differences are equal to 0, and report 95% confidence intervals on the t-distributions associated with these estimates using the pt() and qt() functions.

```
# Compute Mean Differences:

MuDiffEst = MDiffs %*% coef(mdl)
mdl.vc = vcov(mdl)
MuDiffVC = MDiffs %*% mdl.vc %*% t(MDiffs)
MuDiffSE = sqrt(diag(MuDiffVC))

# Prepare a summary table:

MuDiffSummary = data.frame(mudiff = MuDiffEst, se = MuDiffSE)

# Compute t stat

MuDiffSummary$t = MuDiffSummary$mudiff/MuDiffSummary$se
# Compute the 2-tailed p-value
MuDiffSummary$p.value = 2*pt(abs(MuDiffSummary$t),df=nrow(X)-ncol(X),lower.tail=FALSE)

# Compute 95% confidence interval
MuDiffSummary$lower95 = MuDiffSummary$mudiff + MuDiffSummary$se*qt(p=0.025,df=nrow(X)-
    ncol(X))
MuDiffSummary$upper95 = MuDiffSummary$mudiff + MuDiffSummary$se*qt(p=0.975,df=nrow(X)-
    ncol(X))

# Inspect the output (rounded to 3 decimal spaces)
> round(MuDiffSummary,3)
                              mudiff    se      t p.value lower95 upper95
AA:american - AA:korean       -0.220 0.109 -2.025   0.043  -0.434  -0.007
AG/GG:american - AG/GG:korean  1.092 0.085 12.885   0.000   0.925   1.258
AG/GG:american - AA:american   0.747 0.103  7.263   0.000   0.545   0.949
AG/GG:korean - AA:korean      -0.565 0.092 -6.152   0.000  -0.745  -0.385
```

Looking within culture (rows 3 & 4), we see that G allele Americans have a higher ESS score by 0.75 points within the interval 0.55 - 0.95. Thus G allele Americans compared to A only Americans have a nearly 0.5 to 1 point added to their ESS score, implying that they are more likely to seek emotional support.

By contrast, the G allele Koreans are estimated to be between -0.75 to -0.4 lower than their A only compatriots in ESS score, meaning they are less likely to seek emotional support under high psychological distress.

This indicates how the interaction works. The same allele in different contexts produces opposite results.

The A only allele genotypes across cultures show similar ESS scores. The p-value is 0.043 because the confidence interval is approaching zero but does not overlap it. Thus we detect a smaller difference between these groups. The G allele genotypes across cultures naturally have a wider separation, since we know within genotype the cultures are different.

Since the interaction shows that genotypes in different cultural contexts move in **opposite** directions, we could consider that averaging across them to interpret the main effect is not that interesting. Nevertheless, we can compute it:

```
# Average across genotypes:

tmpXam = colMeans(tmpX[3:4,])
tmpXkor = colMeans(tmpX[1:2,])

AvGenoDiff = matrix(tmpXam - tmpXkor,nrow=1,ncol=4)
rownames(AvGenoDiff) = "AvGeno:american - korean"
colnames(AvGenoDiff) = names(coef(mdl))

AvDiffSummary = data.frame( mudiff = AvGenoDiff %*% coef(mdl),
              se = sqrt(diag(AvGenoDiff %*% vcov(mdl) %*% t(AvGenoDiff))))
AvDiffSummary$t = AvDiffSummary$mudiff / AvDiffSummary$se
AvDiffSummary$p.value = 2*pt(abs(AvDiffSummary$t),df=nrow(X)-ncol(X),lower.tail=FALSE)
AvDiffSummary$lower95 = AvDiffSummary$mudiff + AvDiffSummary$se*qt(p=0.025,df=nrow(X)-
    ncol(X))
AvDiffSummary$upper95 = AvDiffSummary$mudiff + AvDiffSummary$se*qt(p=0.975,df=nrow(X)-
    ncol(X))

MuDiffSummary = rbind(MuDiffSummary, AvDiffSummary)


> round(MuDiffSummary,4)
                                mudiff     se       t p.value lower95 upper95
AA:american - AA:korean        -0.2203 0.1088 -2.0253   0.043 -0.4337 -0.0070
AG/GG:american - AG/GG:korean   1.0916 0.0847 12.8853   0.000  0.9254  1.2577
AG/GG:american - AA:american    0.7469 0.1028  7.2627   0.000  0.5452  0.9486
AG/GG:korean - AA:korean       -0.5650 0.0918 -6.1519   0.000 -0.7451 -0.3849
AvGeno:american - korean        0.4356 0.0689  6.3189   0.000  0.3004  0.5708
```

Adding it to our summary as an additional hypothesis test, we see that even averaging across genotypes, the Americans have a higher ESS score. However, we should take that with a grain of salt, since it like is driven by the larger gap between the G-allele in Americans vs. Koreans, so this conclusion may not be very informative.

Performing Multiple Hypothesis Testing Adjustments with *multcomp()* with single-step methods (e.g. Tukey's Honestly Signficant Difference, Dunnett)

At the 95% confidence level, we suspect that 5% of null hypothesis rejections are false positives. This amounts to 1 in 20 hypothesis tests. By simulation, you can show this is the case, and we have to believe that 1 in 20 significant findings throughout the literature based on this criterion are false positives.

To deal with this issue, there are number of multiple hypothesis testing procedures. One you may have heard of is Bonferroni. In Bonferroni correction, the significance criterion ($p < 0.05$) is adjusted by the number of tests performed to yield a more stringent significance criterion.

This is often an extremely conservative estimate. In the context of linear model hypothesis tests, we also know the tests are not independent of one another. We already know for example, from the ANOVA, that an interaction is present in our data. Thus if we observe American Gs to have higher ESS scores than American As, we already expect that the Korean raised counterparts will show a smaller change, or an opposite change. Furthermore, we would like to make some interesting decisions about what to do next in our data. If we raise the bar too high, we may miss something that is worth following up on and reproducing.

Thus, a less conservative approach is comprised of the **single-step** methods. Single-step methods were designed in different ways, but reduce to the same type of method:

> Given N Hypothesis tests with associated estimates and standard errors, the **single step method** evaluates the parameter t distribution arising from a **multivariate t distribution of dimension equal to the number of hypothesis tests**.

This is sort of ethereal to understand. But a simple way to understand it is as a **joint probability**. Multivariate probability distributions are the distribution of joint probabilities. In the discrete case, we'd like to know what the probability of observing a=value1 is when b=value2. For example, if you flip two fair coins, what is the probability of observing Heads and Heads on both. While each has a probability of 1/2, the joint probability of both is 1/4. In the continuous case, we ask what is the probability of observing **a vector of t statistics (or greater), given a space of N such distributions**. This joint probability will be different than conducting each test as if it existed alone. The adjusted p-values and confidence intervals are the conditional values after accounting for all the tests.

The multcomp package determines the associated multivariate distribution and conditional probabilities by calling a second package, the **mvtnorm** package that implements both multivariate normal distributions and multivariate t distributions for R. It uses the associated pmvt() and qmvt() functions to determine the conditional p-values and quantiles needed for inference on the hypothesis tests.

Even though different single-step methods use different criteria classically, they converge on this single method. Thus, a space of any number of hypotheses can be explored. Classically two tests are commonly used:

1. Tukey's Honestly Significant Difference:
   This test adjusts for a space of a number of hypothesis tests equal to the **total number of possible pairwise comparisons in the data**. In our case, that would be 6 total tests.

2. Dunnett's Test:
   This test adjusts for a space of number of hypothesis tests equal to **the number of group means -1** as it compares each mean to a single reference mean (the "many to one" problem). In our case, there would be 3 such tests with 4 possible means.

Because both of these tests are single-step methods, they compute conditional quantiles and p-values dependent on a space of either 6 or 3 tests specifically. But this is generalizable to any number of tests. In our case, considering our 4 tests of interest, we should expect a level of adjustment somewhere in between these two. If we had a matrix computing all 6 tests, multcomp() would return to us the Tukey's HSD. If we had the tests formulated all with respect to a single control, we are returned Dunnett's values. In our case, we have a custom design. I have seen this reported in the literature as "adjustment for multiple comparisons was performed using the single-step method as part of the multcomp() package in R" referencing the associated publication [3].

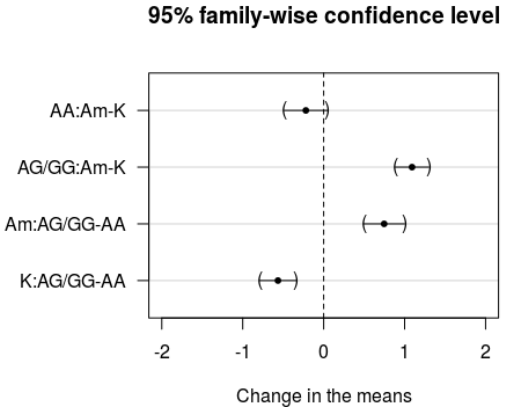
To return the associated values using the multcomp package, we use the **glht** function with the linfct option, again using **tidy** from the broom package to make a clean data frame:

[3] Hothorn, T., Bretz, F., Westfall, P. Simultaneous Inference in General Parametric Models. Biometrical Journal (2008), Vol. 50, No. 3, pp. 346-363

```
# Using MDiffs, our matrix of 4 hypothesis tests:

posthoc.custom = glht(mdl, linfct=MDiffs)
customSummary = tidy(summary(posthoc.custom))
rownames(customSummary) = customSummary$lhs
# This takes a column from the tidy output to make the row names

customInt = tidy(confint(posthoc.custom))
customSummary = cbind(customSummary[,c("estimate","std.error","statistic","p.value")],
          customInt[,c("conf.low","conf.high")])


# Inspect the output, rounded to 4 decimal places:

> round(customSummary,4)
                              estimate std.error statistic p.value conf.low conf.high
AA:american - AA:korean         -0.2203    0.1088   -2.0253  0.1386  -0.4867    0.0461
AG/GG:american - AG/GG:korean    1.0916    0.0847   12.8853  0.0000   0.8841    1.2990
AG/GG:american - AA:american     0.7469    0.1028    7.2627  0.0000   0.4950    0.9987
AG/GG:korean - AA:korean        -0.5650    0.0918   -6.1519  0.0000  -0.7899   -0.3401
```

After correcting for 4 hypotheses, you can see the adjusted interval for the first comparison overlaps zero with p.value = 0.1386. The bulk of the confidence interval stretches below zero, so it is not appropriate to say that AA:american and AA:korean are “not different”. But they have a smaller level of difference that is not considered significant after correction. This comparison would be lower priority for replication, but replication would allow us to say further what we believe. The other intervals are slightly widened. You can plot this using the plot function:

```
#Change back to a single grid
par(mfrow=c(1,1))
#so the labels don't get cutoff:
rownames(MDiffs)=c("AA:Am-K","AG/GG:Am-K","Am:AG/GG-AA","K:AG/GG-AA")
plot(confint(glht(mdl,linfct=MDiffs)),xlab="Change in the means",xlim = c(-2,2))
```



If we wanted to run a full Tukey’s set of comparisons, we could use our unmodified Tukey matrix from before:

```
# Using MDiffs3, our matrix of all possible comparisons:

Tukey = contrMat(table(tmp$mixed),type="Tukey")
MDiffs3 = Tukey %*% tmpX
posthoc.tukey = glht(mdl, linfct=MDiffs3)
tukeySummary = tidy(summary(posthoc.tukey))
rownames(tukeySummary) = tukeySummary$lhs
# This takes a column from the tidy output to make the row names

tukeyInt = tidy(confint(posthoc.tukey))
tukeySummary = cbind(tukeySummary[,c("estimate","std.error","statistic","p.value")],
                     tukeyInt[,c("conf.low","conf.high")])


# Inspect the output, rounded to 4 decimal places:

> round(tukeySummary,4)
                                estimate std.error statistic p.value conf.low conf.high
AG/GG:korean - AA:korean         -0.5650    0.0918   -6.1519  0.0000  -0.8008   -0.3293
AA:american - AA:korean          -0.2203    0.1088   -2.0253  0.1766  -0.4996    0.0589
AG/GG:american - AA:korean        0.5266    0.0841    6.2600  0.0000   0.3106    0.7425
AA:american - AG/GG:korean        0.3447    0.1092    3.1551  0.0088   0.0643    0.6251
AG/GG:american - AG/GG:korean     1.0916    0.0847   12.8853  0.0000   0.8741    1.3090
AG/GG:american - AA:american      0.7469    0.1028    7.2627  0.0000   0.4829    1.0108
```

You can see that it has slightly widened the confidence interval for AA:american - AA:korean and increased the adjusted p-value to 0.1766. However, because there are only 6 hypothesis tests, the adjustments do not particularly change our priorities.

Summary And Final Thoughts

On p-values

The p-value is a decision criterion, though many consider it a bad one. It means the probability of observing a particular test statistic equal to or more extreme than a particular value (here, a t statistic) when the null hypothesis is true.

The null hypothesis for F tests are that a model is best predicted by a model lacking a particular term.

The null hypothesis for mean contrasts (t tests) is that the contrast is equal to 0.

One thing to consider about mean contrasts is that it actually might be relatively easy with a large enough sample size to observe a difference that is non-zero. If the sample size is increased enough, this can make the p-value very small. That said, perhaps a better null hypothesis is that the mean difference exists within some range around zero. Maybe for this ESS score, a difference of -0.1 to 0.1 is considered essentially no difference at all. This is called **practical significance** and it requires the researcher to have intuition or prior knowledge about what sorts of differences are meaningful. Is a single unit on the ESS score likely to amount to any effective difference in a person's life? The p-value criterion does not have a way to take this into account. Bayesian methods allow the researcher to incorporate prior information in order to compute the actual probabilities of different hypotheses being true.

If you **fail to reject the null hypothesis it does not mean that two means are equal**. In the case of our adjusted p-values, we would say that the contrast between the AA genotypes across cultures is not significantly different from zero. Nevertheless, you can see from the confidence interval that the difference is **mostly** less than zero. You need to inspect graphs of data points and confidence intervals around means in order to interpret whether or not you think failure to reject the null hypothesis means the means are equal, or whether it means you have a weak effect and not enough of a sample size. If the confidence interval were **symmetric** around 0, we might conclude that the AA genotypes had equivalent scores across cultures.

What to report

This is subjective, so here comes a bunch of my own opinion. Traditionally you report the results of the F tests, and then the post-hoc summary. Most people just report the p-values. **The p-value is a decision criterion, but is not particularly informative on its own**. You should report the:

1. Actual value of a mean difference (This tells the reader the magnitude of the effect. Equivalently, there are a number of other metrics of "effect size". Still the actual value of the mean difference is in a tangible unit for readability).
2. The confidence interval and/or the standard error associated with the mean difference.
3. Your interpretation based on other data or prior literature as to whether you think this much of a difference is practically important to anything.

On Multiple Testing Adjustments

People get very excited when they find that they are able to reject a null hypothesis. However, certainly many such differences are not particularly meaningful. Your decision to reject a null hypothesis should be a platform to run another experiment that explores the implication of that decision. If not, you made a decision for no reason. There is no amount of multiple test adjustment that can really make you certain that a decision is not a false positive, and if you correct for everything ever reported, only the largest and most robust effects would remain.

Google Scholar contains roughly 160 million documents as of May 2014. Suppose each document runs statistical tests and reports 5 significant findings at the $p<0.05$ level. This means that our new criterion for significance is $p < 1.56E\text{-}9$ by Bonferroni. This is actually not so bad right? In our simulation study here, the uncorrected p-value for AG/GG American vs. AA American was 5.4 E -13. So we would still accept this finding. In fact, we would accept all the findings in our table except the AA comparison across cultures. However, in large genomic studies, where many tests are performed, this would start to become onerous and would impede our ability to follow interesting lines of inquiry. Don't consider multiple testing correction as a means to being "more certain about the truth". Instead, consider it a rank prioritization scheme for further experimentation. Only further experimentation (and maybe different decision criteria) can bring us to a higher degree of certainty, but rank prioritizing using multiple testing adjustment gives us a way to determine which follow-up experiments are worth doing first (/end opinions rant).