3 Basic Analyses

If you were starting to get impatient about when we will actually start to analyze some data, the wait is over! By this point we can now manually enter data, navigate through folders on our computer (via MATLAB), and load and save data files. Now comes the good part!

Before we begin, we should get you acquainted with the single function that you will use the most often.

3.1 Don't Be Afraid to Ask for Help!

MATLAB has a great built-in help function that displays information about a function in the main command window (**`help`**). `help` displays the syntax required to use the function, as well as listing some related functions. If you're not sure what the function you're looking for is called, and guessing doesn't seem to work, MATLAB also has a function called `lookfor`. Using **`lookfor`**, we just tell MATLAB what we are trying to do (in the form of keywords), and MATLAB searches the descriptions of all of the functions and tells us where our keyword comes up.

For example, if you want to find the average of a list of values, you might think that you just need to use a function called "average," right? If you were in Microsoft Excel, you would be correct. However, this is not Excel. In MATLAB (and in statistics!), we use the function called "mean." If you didn't know this yet, this would be a great opportunity to try `lookfor`. Even though you know now, I suggest you try `lookfor` and `help` anyway.

```
>> lookfor average
MEAN   Average or mean value.
>> help  mean
 MEAN    Average or mean value.
    For vectors,  MEAN(X)  is the mean value of the elements in X. For
    matrices, MEAN(X) is a row vector containing the mean value of
    each column. For N-D arrays, MEAN(X) is the mean value of the
    elements along the first non-singleton dimension of X.

    MEAN(X,DIM) takes the mean along the dimension DIM of X.

    Example: If X = [0 1 2
                     3 4 5]

    then mean(X,1) is [1.5 2.5 3.5] and mean(X,2) is [1
                                                      4]

    See also median, std, min, max, var, cov, mode.

    Reference page in Help browser
       doc mean
```

A second, more detailed help command also exists, called `doc`. Using the `doc` function opens a separate help window and looks up the function in question. The `doc` function differs from `help` in that it provides a more detailed description of the function, as well as more examples of how to use the function. `doc` also may include tables and figures in its examples, rather than just text.

```
>> doc mean
```

★ TIP #18

Don't underestimate the usefulness of `help` and `doc` when you aren't sure what function to use. Though they may be cryptic at times, they often can give you some direction as to how to solve your problem! ■

3.2 Descriptive Statistics

Now, we'll need to start doing some simple analyses. Usually some things we need to do are average values across, find the median, find standard deviations, and find out how many values we have.

- `mean`: calculates the average/mean of the given numbers.
- `median`: calculates the median from a data set.
- `std`: calculates the standard deviation.
- `var`: calculates the variance.
- `min`: reports the smallest value in the specified data set.
- `max`: reports the largest value in the specified data set.
- `sum`: sums up all of the values in the data set.
- `length`: calculates how "large" a matrix is (only outputs the largest dimension of the matrix).
- `size`: gets the lengths of *all* dimensions of a matrix.
- `sort`: rearranges the rows to be "sort"-ed by their value.

Of course, you would need to also include the variable you want to do the operation on. For example, the mean of matrix `M` would be `mean(M)`.

For now, let's revisit our `iqbrain` data set.

```
>> iqbrain = load('data.txt');
```

Let's try our new functions to check out the size of our iqbrain matrix.

```
>> length(iqbrain)
ans =
    40
>> size(iqbrain)
ans =
    40     5
```

To calculate the average IQ for the participants in this study, we simply ask MATLAB using the `mean`.

```
>> mean(iqbrain)
ans =
   1.0e+05 *
     0.0000    0.0011        NaN        NaN        9.0876
```

Hmm . . . that doesn't quite work. If we use the `mean` function, it calculates the mean for each column. However, the IQs are only in one of our columns. If you recall from Chapter 1, we learned how to select only portions of our data, which is our second column. For your reference, I'll also show you how to select the fifth row (here, participant), rather than column.

```
>> iqbrain(:,2)
ans =
    133
    140
    139
    133
    137
    ...
>> iqbrain(5,:)
ans =
              2         137         147          65         951545
```

Now, for the mean . . .

```
>> mean(iqbrain(:,2))
ans =
  113.4500
```

★ **TIP #19**

Never name a variable with the same name as a function! For example, `mean = mean(iqbrain)`. While this command will work, from this point on `mean` will refer to your new variable rather than the MATLAB function. If you do make this mistake, using `clear` to clear the variable from your workspace should get you back on track! To do this, type `clear('mean')`. ■

Similarly, we can also calculate the minimum, maximum, and standard deviation.

```
>> min(iqbrain(:,2))
ans =
    77
>> max(iqbrain(:,2))
ans =
   144
>> std(iqbrain(:,2))
ans =
   24.0821
```

★ TIP #20

When using some of these functions, primarily `mean` and `std`, you may realize MATLAB is doing the analysis along a dimension other than the one you want. In these cases, you want to specify what dimension for MATLAB to do the function on. However, the developers of MATLAB decided not to make both functions work quite the same way. If you want to do the mean across the second dimension (across columns) you would do `mean([variable name],2)`. However, the respective standard deviation would be `std([variable name],[],2)`. This is because `std` takes a different variable in the second position, rather than the dimension to do the calculation across. A dimension can also be specified in the `size` function, simply as `size([variable name],[dimension number])`. ■

3.3 Comparing Values

Next, we need to be able to compare some numbers. The main relational operators in MATLAB are as follows:

`=`	sets values[†]
`~`	not
`==`	is equal to
`~=`	is not equal to
`<`	is lesser than
`>`	is greater than
`=<`	is less than or equal to
`=>`	is greater than or equal to

When we give MATLAB two values, with one of these operators in the middle, it tells us if that statement is true or not! MATLAB tells us this by responding with either 1 (true) or 0 (false). This kind of response is called "Boolean" logic.

[†] a single equal sign alone does *not* compare values, but was still included here for completion, and to remind you of this fact.

```
>> 1 == 1
ans =
     1
>> 1 == 0
ans =
     0
>> 42 > 10
ans =
     1
>> iqbrain(:,2)' > 100
ans =
  Columns 1 through 10
          1     1     1     1     1     0     1     0     0     1
  Columns 11 through 20
          1     1     1     1     0     0     1     0     1     0
  Columns 21 through 30
          0     0     1     1     0     1     0     1     0     1
  Columns 31 through 40
          1     1     1     0     0     1     1     0     0     0
```

Soon you will learn to

- Select specific subsets of data based on logical statements (p. 53)
- Calculate descriptive statistics for experimental conditions (p. 45)

3.4 Working With Logical Operators

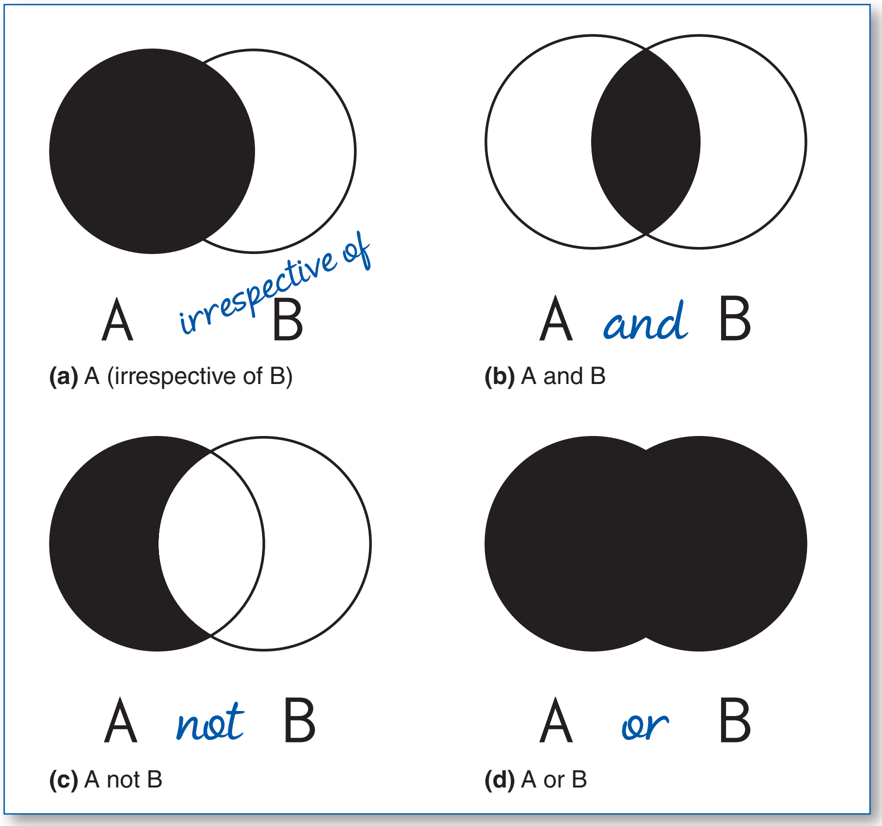
Okay, so you can compare numbers and see if they're greater than, lesser than, or equal to each other. We can also see where one number is `not` the same as another set. Let's not stop there though! To do more complicated comparisons we can use `and` and `or` operations as well.

While combinations of these operators may seem trivial, or even irrelevant, they will lay the foundation of your data analysis. As such, I will explain their use through several different approaches.

First, consider that we have one data set of values, called "A," as well as a second data set, labeled "B." If we are only interested in A, then we can stop there (Figure 3.1a).

```
>> A = [ 1 1 1 1 0 0 0 0 ];
>> B = [ 1 0 1 0 1 0 1 0 ];
>> Aonly = A
Aonly =
       1     1     1     1     0     0     0     0
```

Figure 3.1 Venn diagrams of Boolean operators.



If we are interested in values that are present in both A *and* B, then we use *and* (Figure 3.1b). In MATLAB, *and* is represented by an & (ampersand) symbol.

```
>> AandB = A & B
AandB =
      1      0      1      0      0      0      0      0
```

If we only want to consider cases where values are present in A, but *not* in B, we use the Boolean operator *not* (Figure 3.1c). In MATLAB, *not* is represented by a ~ (tilde) symbol.

```
>> AnotB = A & ~B
AnotB =
      0      1       0       1       0       0       0       0
```

Last, if we are satisfied if any value is a part of either A *or* B, then *or* is the appropriate operator (Figure 3.1d). In MATLAB, *or* is represented by a | (bar or pipe) symbol.

```
>> AorB = A | B
AorB =
      1      1      1      1      1      0      1      0
```

Of potentially similar usefulness is when A is not equal to B, which would include both "sides" of the Venn diagram, without the middle. Knowing the above operators, this could be achieved in several ways:

```
>> A ~= B
ans =
      0      1       0       1       1       0       1       0
>>  AorB - AandB
ans =
      0      1       0       1       1       0       1       0
>> (A & ~B) | (~ A & B)
ans =
      0      1       0       1       1       0       1       0
```

Okay, admittedly, that was a bit abstract. Let's try using our `iqbrain` data set to make this a bit more relevant.

First, let's calculate the mean IQ for all the participants in our sample again.

```
>> mean(iqbrain(:,2))
ans =
  113.4500
```

If you print out the contents of `iqbrain`, you can see that there are many participants with IQs lower than 100. Can you check how many? What proportion of the sample had IQs less than 100?

```
>> nLowIQ = sum(iqbrain(:,2)' < 100)
nLowIQ =
    16
>> pLowIQ = sum(iqbrain(:,2)' < 100) ./ length(iqbrain)
pLowIQ =
    0.4000
```

See, now that we've learned the basics, it's fairly easy to start working with data! Let's try a few more:

What were the highest and lowest IQs in the sample?

```
>> lowIQ = min(iqbrain(:,2))
lowIQ =
    77
>> highIQ = max(iqbrain(:,2))
highIQ =
   144
```

3.5 Isolating Specific Portions of Data

Now, we have descriptive statistics and the ability to combine different types of data with Boolean operators. However, one important thing we cannot do yet is look up specific values across conditions.

In addition to having MATLAB report 1s and 0s, it would be more useful to obtain the index values when the statement is true. This can be done with the `find` function.

```
>> find(A)
ans =
     1     2     3     4
>> find(B)
ans =
     1     3     5     7
```

This can also be applied to the use of the Boolean statements from the previous section.

```
>> find(A & B)
ans =
      1     3
>> find(AandB)
ans =
      1      3
```

Additionally, MATLAB also has the function **intersect**, which is specifically made to output values that are part of two sets of values.

```
>> intersect(find(A),find(B))
ans =
      1     3
```

We can also apply this logic to *not* statements and use the MATLAB function **setdiff** to obtain a list of values that are part of one set but not another (unidirectionally, as in Figure 3.1c).

```
>> find(AnotB)
ans =
      2     4
>> setdiff(find(A),find(B))
ans =
      2     4
>> setdiff(find(B),find(A))
ans =
      5     7
```

While not terribly useful on its own, with the use of these indices and multiple data sets that are organized through similar ways, we can find when both things are true. For example, what if I asked you *which* participants were male? (Reminder: `Gender=1` corresponds to males.)

```
>> find(iqbrain(:,1)==1)
ans =
      2
      3
      4
      9
     10
     ...
```

What about which participants had IQs above 100?

```
>> find(iqbrain(:,2)>100)
ans =
      1
      2
      3
      4
      5
      ...
```

Let's try and put those both together: Which participants were males with IQs above 100?

```
>> find(iqbrain(:,1)==1 & iqbrain(:,2)>100)
ans =
      2
      3
      4
     10
     12
     13
     24
     26
     28
     32
     33
     37
```

How many participants is that? What about how many participants were females with IQs above 100?

```
>> nMaleHighIQ = length(find(iqbrain(:,1)==1 & iqbrain(:,2)>100))
nMaleHighIQ =
    12
>> nFemaleHighIQ = length(find(iqbrain(:,1)==2 & iqbrain(:,2)>100))
nFemaleHighIQ =
    11
```

3.6 Simple Analyses Involving Text Data

Hopefully, you are starting to see why this type of code can be so important to our analyses. Let's try this type of analysis out with the word database example data set. First, let's load the data in again.

```
>> clear,clc
>> fid = fopen('JanschewitzB386appB.txt','r');
>> formatstring = [ '%s %s' repmat(' %f' ,1,19)  ];
>> worddata=textscan(fid,formatstring,'headerlines',5, ...
'delimiter','\t');
>> fclose(fid);
```

★ TIP #21

Variable names in MATLAB *cannot* start with a number. For example, naming a variable as 460 words would produce an error. Give it a try and see for yourself! ■

As a reminder, the headers for the columns are listed below and also in the `data_legend.txt` file in the same folder as the data.

```
 1 Word
 2 Type
 3 Letters
 4 Syllables
 5 K&F - Freq.
 6 ANEW - Valence
 7 ANEW - Arousal
 8 Personal Use - Mean
 9 Personal Use - SD
10 Familiarity - Mean
11 Familiarity - SD
12 Offensiveness - Mean
13 Offensiveness - SD
14 Tabooness - Mean
15 Tabooness - SD
16 Valence - Mean
17 Valence - SD
18 Arousal - Mean
19 Arousal - SD
20 Imageability - Mean
21 Imageability - SD
```

Let's try and get the mean arousal ratings for the taboo words. First, we need to use `find` to get their row numbers within the database. Remember that we ended up with a cell array when we first learned to load this data using `textscan` (p. 32).

```
>> find(worddata{2}=='taboo')
??? Undefined function or method 'eq' for input arguments
of type 'cell'.
```

Unfortunately, you can't use the simple comparison operators with text. That being said, it kind of makes sense that strings need to work a bit differently since "greater than" and "less than" don't really make sense with strings. To do comparisons with strings, we get to use another MATLAB function: `strcmp`.

```
>> strcmp(worddata{2},{'taboo'})
ans =
     1
     1
     1
     1
     1
     ...
>> find(strcmp(worddata{2},{'taboo'}))
ans =
     1
     2
     3
     4
     5
     ...
```

Okay, now we can look up the arousal ratings for only these words, which is simply this selection of rows.

```
>> worddata{18}(find(strcmp(worddata{2},{'taboo'})))
ans =
    4.0100
    4.0800
    4.7400
    3.1800
    4.6200
    ...
```

And now we just get the mean of these values! Let's also try the mean imageability and tabooness.

```
>> mAroTaboo = mean(worddata{18}(find(strcmp(worddata{2}, ...
{'taboo'}))))
mAroTaboo =
    4.3357
>> mImagTaboo = mean(worddata{20}(find(strcmp(worddata{2}, ...
{'taboo'}))))
mImagTaboo =
    4.5410
>> mTabTaboo = mean(worddata{14}(find(strcmp(worddata{2}, ...
{'taboo'}))))
mTabTaboo =
    4.8284
```

If you are having a bit of trouble following the code above, don't worry; we'll discuss the combining of multiple functions more in the next section.

Given this data set, one more particularly useful analysis would be to make a list of our word types. This can easily be done by passing the column of word types into a new function: `unique`. This function is great for isolating only the *unique* values within a column of values. It automatically sorts the values, rather than listing them in their order of occurrence.

```
>> unique(worddata{2})
ans =
    ''
    'neg hi ar'
    'neg lo ar'
    'pos hi ar'
    'pos lo ar'
    'rel neu'
    'taboo'
    'unrel neu'
```

This almost works, but it seems we also have an empty entry. This is a remnant of the extra comment lines we noted at the bottom of the file earlier. We can solve this simply by constraining our data that goes into the **`unique`** function to only the rows with the actual words.

```
>> types=unique(worddata{2}(1:460))
types =
    'neg hi ar'
    'neg lo ar'
    'pos hi ar'
    'pos lo ar'
    'rel neu'
    'taboo'
    'unrel neu'
```

Now, we can also get the means for other word types relatively easily.

```
>> mAroNegHi=mean(worddata{18}(find(strcmp(worddata{2},types{1}))))
mAroNegHi =
    3.1987
>> mAroNegLo=mean(worddata{18}(find(strcmp(worddata{2},types{2}))))
mAroNegLo =
    2.5715
```

★ TIP #22

If `strcmp` is useful for your work, you may also want to look into `strncmp`. ■

3.7 Combining Functions

Now, you can compare data in lots of ways! However, to conduct the more "interesting" analyses, you often have to string together several MATLAB functions into one coherent line of code. To do this, you need to start from the basics and *work outward* to get to the end result.

To some, this may seem like common sense (if you understand what I mean by working outward), but for some, this is a real leap of logic, so try and bear with me. When you want to do a particular analysis, you need to decompose it into manageable steps. Then, you need to begin from the most basic, initial step and build up. However, to do this you need to work outward. Let's try an example:

In the previous section, I came up with the following line as the code to calculate the mean arousal for only the taboo words.

```
>> mAroTaboo = mean(worddata{18}(find(strcmp(worddata{2},{'taboo'}))))
mAroTaboo =
    4.3357
```

Let's start from the beginning and work out how I came to this. First, we begin with our columns for the word types and the mean arousal ratings.

```
>> worddata{2}
ans =
    'taboo'
    'taboo'
    'taboo'
    'taboo'
    'taboo'
    ...
 >> worddata{18}
ans =
    4.0100
    4.0800
    4.7400
    3.1800
    4.6200
    ...
```

Using the operators we learned for comparing values, let's ask MATLAB to show us when the word is of the 'taboo' type.

```
>> strcmp(worddata{2},{'taboo'})
ans =
     1
     1
     1
     1
     1
     ...
```

Now, we will ask MATLAB to find the indices where this comparison was reported to be "true" (i.e., 1).

```
>> find(strcmp(worddata{2},{'taboo'}))
ans =
     1
     2
     3
     4
     5
     ...
```

Next, we need to ask MATLAB to show us the arousal ratings, but only for these selected rows.

```
>> worddata{18}(find(strcmp(worddata{2},{'taboo'})))
ans =
    4.0100
    4.0800
    4.7400
    3.1800
    4.6200
    ...
```

We're almost there! Let's ask MATLAB to calculate the mean of these values.

```
>> mean(worddata{18}(find(strcmp(worddata{2},{'taboo'}))))
ans =
    4.3357
```

And now we arrive at the same line of code and output value as we began. Hopefully that made sense. Before we move on, let's try replicating this logic with another analysis—this time using the `iqbrain` data set.

```
>> clear,clc
>> iqbrain = load('data.txt');
```

In this sample, what was the mean IQ for (a) males and (b) females? Which was higher?

The first answer, the mean IQ for males, can be calculated using the line below.

```
>> mMaleIQ = mean(iqbrain(find(iqbrain(:,1)==1),2))
mMaleIQ =
   115
```

Give it a try and see if you can replicate it yourself or at least understand parts of it.

How'd you do? Hopefully, you understood most of it. Regardless, let's try and work through the steps incrementally that led us to that "final" line of code. First, let's take a look at the data stored in the `iqbrain` variable.

```
>> iqbrain
iqbrain =
   1.0e+06 *
    0.0000    0.0001    0.0001    0.0001    0.8169
    0.0000    0.0001       NaN    0.0001    1.0011
    0.0000    0.0001    0.0001    0.0001    1.0384
    0.0000    0.0001    0.0002    0.0001    0.9654
    0.0000    0.0001    0.0001    0.0001    0.9515
    ...
```

Notice that it's all in scientific notation, making it a bit harder to read. Let's look at only the first four columns, without showing the brain-size column.

```
>> iqbrain(:,1:4)
ans =
    2.0000    133.0000    118.0000     64.5000
    1.0000    140.0000         NaN     72.5000
    1.0000    139.0000    143.0000     73.3000
    1.0000    133.0000    172.0000     68.8000
    2.0000    137.0000    147.0000     65.0000
    ...
```

Okay, so let's focus on just the column that denotes the gender.

```
>> iqbrain(:,1)
ans =
    2
    1
    1
    1
    2
    ...
```

Okay, now we check when the gender column matches the value for males.

```
>> iqbrain(:,1)==1
ans =
     0
     1
     1
     1
     0
     ...
```

Next, we get the corresponding participant/row numbers.

```
>> find(iqbrain(:,1)==1)
ans =
      2
      3
      4
      9
     10
     ...
```

Now, get the IQs again, and then the IQs for these specific row numbers.

```
>> iqbrain(:,2)
ans =
    133
    140
    139
    133
    137
    ...
>> iqbrain(find(iqbrain(:,1)==1),2)
ans =
    140
    139
    133
     89
    133
    ...
```

We're almost there! Now, we just need to calculate the mean for this set of IQs.

```
>> mMaleIQ = mean(iqbrain(find(iqbrain(:,1)==1),2))
mMaleIQ =
   115
```

At this point, it should be almost trivial to get the mean IQ for the females.

```
>> mFemaleIQ = mean(iqbrain(find(iqbrain(:,1)==2),2))
mFemaleIQ =
  111.9000
```

It looks like the IQs are slightly higher for the males in this sample. Right now, we can't say anything about this difference being statistically significant; that will come later.

Let's try one more analysis and learn something along the way:

In this sample, were IQs relatively higher for taller participants? What was the median height of the participants (in inches)? What was the mean IQ for participants taller or shorter than the median?

```
>> medHeight = median(iqbrain(:,4))
medHeight =
    NaN
```

Hmm . . . I don't think that's what you were expecting. The problem is that in this column, as well as the height column, the researchers did not provide the height or weight of the participants for confidentiality reasons. But how do we do the descriptive statistics without these? MATLAB clearly doesn't know what to tell us.

3.8 What Is NaN?

Sometimes when we try analyzing our data, MATLAB will respond with 'NaN'. NaN stands for "Not a Number." Basically, MATLAB doesn't think it can answer your command with a number. You just encountered one such instance of this occurring. We can easily find these cases using **`isnan`**.

```
>> find(isnan(iqbrain(:,4)))
ans =
     21
```

In this particular case, the researchers who conducted the `iqbrain` study did not provide the height and weight of a few participants when sharing the data set.

To calculate the median, ignoring the NaN values, we can use the **`nanmedian`** function rather than `median`.

```
>> medHeight = nanmedian(iqbrain(:,4))
medHeight =
     68
```

For other descriptive statistics with NaNs, there also exist other comparable functions such as **nanmean** and **nanstd**. Unfortunately, these NaN-accomodating functions are not part of the core MATLAB distribution and are only included in the Statistics Toolbox. Simply, these functions skip over the NaN values and calculate the appropriate statistic.

If you need such a function but do not have the Statistics Toolbox, you can easily develop your own implementation using `isnan`. (Later, we'll make our own function equivalent to `nanmedian`, `nanmean`, etc.)

```
>> medHeight = median(iqbrain(~isnan(iqbrain(:,4)),4))
medHeight =
    68
```

Now, to answer the actual question regarding IQ and height:

```
>> mean(iqbrain(find(iqbrain(:,4)>medHeight & ...
~isnan(iqbrain(:,4))),2))
ans =
  116.6316
>> mean(iqbrain(find(iqbrain(:,4)<medHeight & ...
~ isnan(iqbrain(:,4))),2))
ans =
  113.6471
```

It looks like the mean IQ is a bit higher for the taller participants.

3.9 Unbalanced or Unexpected Parenthesis or Bracket

One common error that is likely to arise now that we're making these more complicated lines of code is to accidentally not close all of our brackets or to open too many of them. When this occurs, MATLAB will report an error as below.

```
>> mean(iqbrain(find(iqbrain(:,4)<medHeight & ~isnan(iqbrain(:,4)),2))
??? mean(iqbrain(find(iqbrain(:,4)<medHeight & ~isnan(iqbrain(:,4)),2))
                                                                       |
Error: Expression or statement is incorrect-possibly unbalanced (,  {,
or [.
```

When you do come across this error, reread your code carefully and hopefully you'll spot the missing bracket. This would also be a good instance to break the line down and work outward again to rewrite the line of code in question.

EXERCISES

Let's take the training wheels off and try out some of the new functions and operations that we learned in this chapter!

1. How many females were in the `iqbrain` data set?
2. What was the average height of the participants? What were the heights of the tallest and shortest participants? (in inches)
3. What was the weight of the tallest participant? (in pounds)
4. In the `worddb` data set, what were the most positive (valence) words?
5. Which word rated highest on the personal-use scale? Is it the same as the most familiar word?
6. What is the average valence for each word type?
7. Are taboo words more imageable than high-arousal positive and negative words? Are high-arousal words more imageable than low-arousal words?
8. From this database, what are the 10 most imageable words? Which words rank 100 to 110 in terms of word length?
9. What is the median number of letters and syllables across all word types?

See page 194 for the solutions. Now we're making some good progress! Hopefully now you are starting to feel much more capable within MATLAB and can start conducting some basic analyses of your own data. In the next chapter, we will take this one step further and learn to plot our data.

FUNCTION REVIEW

Help: `help lookfor doc`

Basic analyses: `mean median std var min max sum length size sort`

Comparisons: `= ~ == ~= < > =< => strcmp intersect setdiff`

General: `unique`

NaN related: `isnan nanmean nanmedian nanstd`