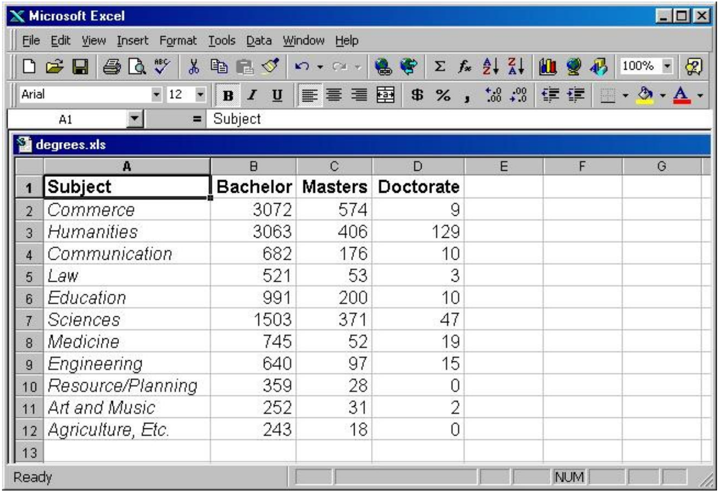
Statistics 120
Data Handling

Data Formats

- Usually raw data sets are entered using spreadsheet or produced in a form suitable for reading into a spreadsheet (see STAT 220 for more details).
- The most common spreadsheet format for data has the *variables* as columns and the *cases* as rows.
- This kind of layout is refered to as a *data matrix*.

A Data Matrix In Excel



	A	B	C	D
1	**Subject**	**Bachelor**	**Masters**	**Doctorate**
2	*Commerce*	3072	574	9
3	*Humanities*	3063	406	129
4	*Communication*	682	176	10
5	*Law*	521	53	3
6	*Education*	991	200	10
7	*Sciences*	1503	371	47
8	*Medicine*	745	52	19
9	*Engineering*	640	97	15
10	*Resource/Planning*	359	28	0
11	*Art and Music*	252	31	2
12	*Agriculture, Etc.*	243	18	0
13				

Saving in Text Format

- The way to get data from Excel into R is to save it in plain text format.
- To do this in Excel
  - Select *Save as* . . . from the *File* menu
  - Set the *save file type* to *Text (Tab delimited)*
- Files produced in this way can be read directly into R (or other software systems).
- Other spreadsheets will provide a similar save method.

Reading Data Into R

- Data can be read into R with the `read.table` function.

```
> degrees = read.table(file.choose(),
      sep = "\t", header = TRUE)
```

- This pops up a dialog box which lets you choose the file to be read.
- Once the file is chosen, R will read the data from it.
- The extra arguments mean that R will assume:
  - the columns are separated by Tab characters
  - the first row contains the variable names

Data Frames

- The data values read by read.table are stored in a special kind of object called a *data frame*.
- Data frames are simply containers which hold variables.
- In the case of the University degrees data, the data frame `degrees` holds the variables, `Subject`, `Bachelor`, `Masters` and `Doctorate`.
- Obtaining the names of the variables in a data frame.

```
> names(degrees)
[1] "X"           "Bachelor"
[3] "Masters"     "Doctorate"
```

Examining Data Frames

```
> degrees
                    X Bachelor Masters Doctorate
1            Commerce     3072     574         9
2          Humanities     3063     406       129
3       Communication      682     176        10
4                 Law      521      53         3
5           Education      991     200        10
6            Sciences     1503     371        47
7            Medicine      745      52        19
8         Engineering      640      97        15
9   Resource/Planning      359      28         0
10      Art and Music      252      31         2
11 Agriculture, Etc.      243      18         0
```

Using Case Names

- The `read.table` function has an option which allows one of the variables it reads to be used as case names (i.e. row names).

- The item is called `row.names` and its value is the column index of the variable containing the names.

```
> degrees = read.table(file.choose(),
        sep = "\t", header = TRUE,
        row.names = 1)
```

A Data Frame with Named Cases

```
> degrees
                  Bachelor Masters Doctorate
Commerce              3072     574         9
Humanities            3063     406       129
Communication          682     176        10
Law                    521      53         3
Education              991     200        10
Sciences              1503     371        47
Medicine               745      52        19
Engineering            640      97        15
Resource/Planning      359      28         0
Art and Music          252      31         2
Agriculture, Etc.      243      18         0
```

Accessing the Variables in a Data Frame

- The variables in a data frame can be extracted by using the `$` operator.

```
> degrees$Bachelor
 [1] 3072 3063  682  521  991 1503
 [7]  745  640  359  252  243


> sum(degrees$Bachelor)
[1] 12071
```

- This quite quickly becomes tedious to type.

Attaching a Data Frame

- An alternative way of getting access to the variables in a data is with an `attach` statement.

- This adds the data frame to the list of places that R searches for variables.

```
> attach(degrees)
> Bachelor
 [1] 3072 3063  682  521  991 1503
 [7]  745  640  359  252  243


> sum(Bachelor)
[1] 12071
```

Subsetting Data Frames

- One of the most important steps in any data analysis is the section of a data subset of interest.

- The R function `subset` provides a convenient way of extracting subsets of data frames.

```
> subset(degrees, Bachelor > 1000)
           Bachelor Masters Doctorate
Commerce       3072     574         9
Humanities     3063     406       129
Sciences       1503     371        47
```

- The result of subsetting is a (smaller) data frame which can be assigned to another variable.

Subsetting Cases and Variables

- The subset function can also be used to obtain a subset of the variables in a data frame, or to simultaneously subset both cases and variables.

```
> subset(degrees, Bachelor > 1000,
      select = c(Bachelor, Masters))
           Bachelor Masters
Commerce       3072     574
Humanities     3063     406
Sciences       1503     371
```

Data Summary

- The R function summary provides a quick way of getting a summary of the contents of a variable or data frame.

```
> summary(degrees)
    Bachelor       Masters        Doctorate
 Min.   : 243   Min.   : 18.0   Min.   :  0.00
 1st Qu.: 440   1st Qu.: 41.5   1st Qu.:  2.50
 Median : 682   Median : 97.0   Median : 10.00
 Mean   :1097   Mean   :182.4   Mean   : 22.18
 3rd Qu.:1247   3rd Qu.:285.5   3rd Qu.: 17.00
 Max.   :3072   Max.   :574.0   Max.   :129.00
```

- The mean, median, quartiles and extremes are printed.

Categorical Data

- A categorical variable records which of several distinct categories an observation falls into.
- Categorical values are typically described by words rather than numbers.
- For example, a person's hair colour can be classified as *black*, *blond*, *brown* or *red*, and temperatures can be classified as *cold*, *warm* or *hot*
- There is a distinction between these two cases. Temperatures can be ordered

  $$\text{cold} < \text{warm} < \text{hot}$$

  while hair-color cannot.

Terminology

- A variable which takes on categorical values is called a *factor*.
- When there is a natural ordering to the categories, such a variable is called an *ordered factor*.
- The values which a factor can take on are called the *levels* of a factor.
- The programming language term for a factor is an *enumerated type* (`enum` in C and C++). There is no equivalent of an ordered factor.

Operations on Categorical Variables

- It makes no sense to try to perform arithmetic operations on categorical data.

- For example, it it hard to see what meaning could be attached to the calculation

$$\frac{\text{Black} + \text{Blond} + \text{Brown} + \text{Red}}{4}.$$

- The only manipulations which are sensible are counting the number of observations falling in each category and dividing the observations into groups which correspond to the levels of a factor.

Creating Factors

- Factors are generaly created from character variables with the function `factor`.

```
> sex = c("Male", "Female", "Male",
      "Male", "Female")
> sexf = factor(sex)
> sexf
[1] Male   Female Male   Male   Female
Levels: Female Male
```

- Note that the levels are obtained in alphabetic order. This can be changed by specifying the levels, in the desired order, as a second argument.

Creating Ordered Factors

- Ordered factors are generally created with the function `ordered`.

- Specifying an order for factor levels is especially important when creating ordered factors. In that case, having the levels in alphabetical order is almost always *not* what you want.

```
> temp = c("hot", "warm", "cold",
      "cold", "hot", "hot")
> tempf = ordered(temp)
> tempf
[1] hot  warm cold cold hot  hot
Levels: cold < hot < warm
```

Creating Ordered Factors

- A second argument can be used to specify the order of the levels (for both `factor` and `ordered`).

```
> tempf = ordered(temp, levels = c("cold",
      "warm", "hot"))
> tempf
[1] hot  warm cold cold hot  hot
Levels: cold < warm < hot
```

- The order of the levels of a factor can be changed by reapplying `factor` or `ordered`.

```
> tempf = ordered(tempf, levels = c("hot",
      "warm", "cold"))
```

Categorical Variables and `read.table`

- When the columns of a data file contains character strings, `read.table` converts them into factors.
- This is almost always the correct thing to do with statistical data sets.
- The exception to this are row names, but these are handled specially.
- The argument `as.is` to `read.table` provides a way of overriding the conversion of character variables to factors.

Data Summaries For Categorical Data

- The function `summary` can be applied to factors and ordered factors. It prints a count of the numbers of times each level occurs in the variable.

```
> summary(sexf)
Female   Male
     2      3
```

- When factors and ordered factors occur in data frames, `summary` prints the appropriate summary for those variables.

Cross Tabulation

- The function `table` can be used to carry out cross-tabulation of several factors.

- Applying `table` to several factors produces a *contingency table*, which shows how frequently combinations of factors occur.

- As an example, we will look at a data set `students.dat` contains observations of the hair colour, eye colour and gender for 592 statistics students (the data was collected in America in the 1970s).

The Student Data

The first 11 lines of the data are as follows.

```
Hair  Eye   Sex
Brown Green Female
Brown Brown Female
Brown Brown Female
Brown Blue  Female
Brown Brown Female
Brown Blue  Male
Brown Hazel Male
Blond Blue  Male
Black Hazel Female
Blond Blue  Male
...
```

Reading and Summarising the Data

- Here is how to read the data.

```
> students = read.table("students.txt",
      header = TRUE)
```

- And to create a basic summary.

```
> summary(students)
    Hair          Eye            Sex     
 Black:108   Blue :215   Female:328  
 Blond:127   Brown:220   Male  :264  
 Brown:286   Green: 64               
 Red  : 71   Hazel: 93               
```

Cross Tabulation

- Here is how to produce a contingency table.

```
> attach(students)
> table(Hair, Eye)
       Eye
Hair    Blue Brown Green Hazel
  Black  20   68      5     15
  Blond  94    7     16     10
  Brown  84  119     29     54
  Red    17   26     14     14
```

- There is clearly an association between hair and eye colour.

Cross Tabulation

- We can make the association more apparent by reordering the levels of the factors.

```
> Hair = factor(Hair, levels = c("Black",
      "Brown", "Red", "Blond"))
> Eye = factor(Eye, levels = c("Brown",
      "Hazel", "Blue", "Green"))
> table(Hair, Eye)
       Eye
Hair    Brown Hazel Blue Green
  Black  68    15    20    5
  Brown 119    54    84   29
  Red    26    14    17   14
  Blond   7    10    94   16
```

Cross Tabulation

- It is also interesting to examine the relationship between hair colour and gender.

```
> table(Hair, Sex)
       Sex
Hair    Female Male
  Black     52   56
  Brown    158  128
  Red       37   34
  Blond     81   46
```

```
> table(Sex)
Sex
Female   Male
   328    264
```

- Note that although there are only 25% more females than males, there are nearly twice as many blond females as blond males.

Higher-Order Cross Tabulation

- A higher-order cross-tabulation can be obtained by including the effects of gender. This can be done with the command

  ```
  > table(Hair, Eye, Sex)
  ```

- The result of this is a three-way table, with rows corresponding to hair colour, columns to eye-color and sheets to gender.

Higher-Order Cross Tabulation

The first sheet produced by the analysis is for Females.

```
, , Sex = Female

       Eye
Hair    Brown Hazel Blue Green
  Black    36     5    9     2
  Brown    81    29   34    14
  Red      16     7    7     7
  Blond     4     5   64     8
```

Higher-Order Cross Tabulation

The second sheet produced by the analysis is for Males.

```
, , Sex = Male

       Eye
Hair    Brown Hazel Blue Green
  Black    32    10   11     3
  Brown    38    25   50    15
  Red      10     7   10     7
  Blond     3     5   30     8
```

The Need For Visualisation

1. Tables of counts clearly contain information, but it can be hard to extract.

2. We need to have techniques which let us “see” the patterns present in count data.

3. We will look at techniques for visualising count data next time.