++

4. FIRST STEPS IN DATA ANALYSIS

++

++++++++++++++++++++++

4.1 Objectives

++++++++++++++++++++++

Let's imagine that you have been assigned an indexing project. You are to prepare an indexed version of a large database, ready for other people to search in it for information. How shall you proceed? In this topic, we deal with first steps in hands-on analysis of machine-readable databases.

You may have to start by fending off shortcuts. Some managers and clients want work completed yesterday. The first question you are asked is: "How long is it going to take?" "The data is the same as last time; we want only a few minor changes. This won't take very long, will it?" Refuse <u>any</u> time estimate until your analysis is complete. The ultimate objective is a satisfied user. Shortcuts at the analysis and specification stage rarely (make that "never") serve that objective. The best strategy is to write time estimates once the project is fully specified in writing and the data has been analyzed.

Analysis logically comes before data preparation; we can't prepare something for indexing until we know what we have. If we are dealing with familiar data, we still have to watch for data quality problems... inconsistencies or unexpected patterns that are common within large accumulations of data.

The analysis stage provides information that is needed to prepare the data. After analyzing the data, we should have enough information so that we will be able to:

> <u>extract searchable content</u>, that is, separate out displayable text with punctuation and each search term (word, phrase, numeric value, etc.) intact, at the same time screening out all material that is not part of the

intended display;

> recognize record separations, that is, identify divisions between units in cases where data is logically divided into units which are meaningful to the searcher (a property in a real estate file, a person in a personnel file, a subsection in a manual of regulations, a heading in a book or magazine article, etc.);

> recognize field separations, that is, identify divisions between elements of data that take on different meaning according to where they occur within a record. Examples of fields... purchase order number, street address, city, postal code, quantity, item description, cost per unit, etc.

> recognize formatting aids, that is, identify bytes within the data that are intended to control display of data... indents from the margin, shifts to different fonts (e.g., italics), table spacing, etc.

This may strike you as simply finding out what is needed to reduce data to formatted printable ASCII. In a sense, it is. But there are some pitfalls along the way. The trick is to be able to recognize the pitfalls early in the indexing process while the costs are still low. It's quite discouraging to index a billion bytes and then find the index list contains large numbers of false or meaningless search terms.

++
4.2 Learn how the data was accumulated
++

Ask questions. The data came from somewhere. How was the data put together? Specifically:

> Was the same method used consistently to assemble all the data? Or were there changes along the way? (If methods or programs changed, then each subset of data must be separately analyzed.) If possible, find out why the data-gathering method changed. Was the change prompted by a quality control problem, or the introduction of new technology? In either case, check the first part of the new data carefully for errors. Starting up in unfamiliar surroundings can lead to extra errors for a while. If the

change was made on account of quality control problems, re-check the portion of data created just before the change.

> Was the data scanned from print media? Which scanning technology and software were used? What was the release date of that software? What was the quality of the print media? Was the layout regular? How thoroughly has the result been checked for reliability?

> If keypunching was used as a low cost substitute for scanning printed material, were the keypunchers and their supervisors familiar with the subject matter? Were they working in their own language? What methods of verification were used? (For example, keying by a second operator is not always reliable.) How much pressure were the keyers under? Were they being monitored and paid by the kilo-keystroke as has often been the case? People under pressure tend to make more errors than those working "at their own pace".

> Has the data been used and updated in ways that would sift out errors?

++

4.3 Learn how the data will be used

++

Put yourself in the position of a person searching within this database. Don't settle for educated guesses; find out for whom the data is being indexed and ask questions: What are the needs of prospective searchers? How do they look for information now: manually, using a mainframe computer, not at all? In what ways can the indexing setup add to the value of the data for their needs? (For example, a computerized phone book can be organized one way to create mailing lists, in quite another to optimize the speed of looking up individual names.) What kinds of search term combinations are typically used? Perhaps some forms of search were deemed impractical in the past. If searchers were invited to imagine away all restrictions, what would they like to be able to do?

Marketing questions are often outside the interests of technical people. That's unfortunate, since all sorts of technical side trips to nowhere might be avoided by interviewing even a few potential end users.

++
4.4 Access to samples and hard copy
++

After acquiring technical background re the data and marketing information on potential users, the key issues are:

> access to the data on <u>media</u> that can be handled by your computer. Non-standard media still exist. ("I just received the diskettes from the first volume on 8 inch disks"... letter in February 1992). And there are standard media that may not be useful to you... nine track 6250 BPI tape doesn't feed the average personal computer; 1.2 megabyte floppies aren't worth much on a 360K floppy drive.

> ability to extract <u>samples that are representative</u>. This depends in part on the degree of consistency of the data. Pay particular attention to the first several thousand bytes, and the very last several thousand bytes. If the data set is large, extract randomly selected portions in between. If there are known changes in how the data was accumulated, put together samples from each variation.

> access to <u>hard copy</u>. If at all possible, get paper printouts that match the samples. The paper version is helpful for identifying format and typesetting codes. The way the hard copy is laid out also tells volumes about the search philosophy used to date. Ask users how well the printed layout served their needs.

++++++++++++++++++++++++++++++++++
4.5 Access to software tools
++++++++++++++++++++++++++++++++++

A variety of programs (software tools) are supplied with each tutorial, in source code and in executable form. Source code has the extension ".C" and the DOS executable normally has the extension ".EXE". There is also the occasional batch file, a series of commands in an ASCII file that can be interpreted by DOS. If you have not already done so, include the directory that has the executable version in the series of PATH names in the AUTOEXEC.BAT file on your computer. Each program has a description, or help screen, which is displayed when you input the command (the program name without the extension) followed by /U (for Usage) or ? (for help). Normally we will include the help screen in the tutorial the first time a program is mentioned.

++

4.6 Extracting samples from larger files

++

The program CPB (copy bytes) is useful for extracting samples from larger files. Here is its help screen description:

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
Usage   cpb  byte_count  start_byte  input_file  output_file

            Copy bytes...  Copy any portion of any file to a new file. Start at a
            specified byte, copy a specified byte count. Standard output may
            not be used because DOS drops carriage returns and ctl-Z.

input:  Any file whatsoever.

output: Portion of the same file.

writeup: MIR TUTORIAL ONE, topic 4
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Suppose you wish to separate out the first 10,000 bytes of a file named BIG_DATA into a file called SAMPLE.000. The command would be:

```
CPB 10000 0 BIG_DATA SAMPLE.000
```

To isolate 7,500 bytes from the middle of a 200,000 byte file, the command takes the form:

```
CPB 7500 100000 BIG_ONE MY_SAMPL
```

CPB (copy bytes) is particularly useful for massive files. Here's how to call out a sample between 4,000 and 5,000 bytes in length from the end of a file that is 248,885,237 bytes long:

```
CPB 5000 248881000 BIGGER.YET BIGGER.END
```

The file BIGGER.END in this case would be 4,237 bytes long. CPB simply stops copying when it reaches the end of the input file.

Sampling is one of many uses of CPB. We will use it in a variety of ways further along.

If you wish to make a UNIX version of CPB, consider removing the input and output file names and using standard input and output instead. That is safe since UNIX does not quietly mess up on carriage returns and CTL-Z characters. The advantage of standard input and output is the convenience of piping from one program to another without the clutter of intervening files.

++

4.7 Byte surveys - a worked example

++

We start the analysis with a byte survey of each sample, or of one file that combines all your data samples. If you choose to combine samples, use the DOS COPY command to join them together, being very sure to use the "/b" binary flag:

```
COPY /B HEADER + BATCH1 + BATCH2 + TAIL  BIG_ONE
```

The destination file BIG_ONE has the four source files concatenated together in order: HEADER, BATCH1, BATCH2, TAIL.

If you have a series of files named SAMPLE.001, SAMPLE.002, SAMPLE.003, etc.,

```
COPY /B SAMPLE.* BIG_ONE
```

Included among the MIR support files supplied with TUTORIAL ONE is a 238,312 byte file SVP_TXT. This file contains English translation of some 17th century correspondence of the French priest and reformer, Vincent de Paul, who founded the "Congregation of the Mission". The Vincentians, generous as always, have given permission to use the correspondence for an example in demonstrating indexing and retrieval methods.

The file SVP_TXT is used here to introduce tools and first steps in data analysis. The file's quarter megabyte size is no problem. For example, the program A_BYTES classifies and counts every one of the 238,312 bytes in only four seconds on a slow machine (AT 80286 at 12 Megahertz).

The character survey is produced by the program A_BYTES ("analyze bytes")...

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
usage:  a_bytes [ /L ] file_name[s]

        Analyze the bytes (characters) used within any file, report the
        frequency of each byte present.  If the location flag /L is set, include
        offsets of the first 8 occurrences of each byte pattern present.

input:  Any file[s] whatsoever.

output:     file_name.BYT which contains up to 256 lines, one line for each
different byte present.  The byte is shown first in printable OR octal form,
then the hexadecimal equivalent. The third column is frequency.  The fourth
column shows percentage of total occurrences within the file.

        If the /L locations option is selected, the output file is name
        file_name.LOC and the offsets of the first up to 8 occurrences follow
        at the end of each line.

writeup: MIR TUTORIAL ONE, topic 4
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
        A_BYTES SVP_TXT
```

creates a report file SVP_TXT.BYT which is a byte analysis of the entire file..
Here is that report:

```
\012 [0A]   4117   1.7%
\015 [0D]   4117   1.7%
\032 [1A]      1   0.0%
\040 [20]  38883  16.3%
!    [21]    173   0.1%
'    [27]     87   0.0%
(    [28]     38   0.0%
)    [29]     38   0.0%
,    [2C]   2754   1.2%
-    [2D]    266   0.1%
.    [2E]   1922   0.8%
0    [30]    258   0.1%
1    [31]    560   0.2%
2    [32]    168   0.1%
3    [33]    117   0.0%
4    [34]    485   0.2%
5    [35]    170   0.1%
6    [36]    771   0.3%
7    [37]    174   0.1%
8    [38]    198   0.1%
9    [39]     85   0.0%
:    [3A]     87   0.0%
;    [3B]    158   0.1%
<    [3C]   1511   0.6%
=    [3D]    935   0.4%
>    [3E]   1511   0.6%
?    [3F]     62   0.0%
@    [40]    935   0.4%
A    [41]    658   0.3%
B    [42]    225   0.1%
C    [43]    403   0.2%
D    [44]    962   0.4%
E    [45]   1357   0.6%
F    [46]    124   0.1%
G    [47]    388   0.2%
H    [48]    501   0.2%
I    [49]   1715   0.7%
J    [4A]     90   0.0%
```

```
K    [4B]      5   0.0%
L    [4C]    519   0.2%
M    [4D]   1029   0.4%
N    [4E]    466   0.2%
O    [4F]    536   0.2%
P    [50]    476   0.2%
Q    [51]      9   0.0%
R    [52]    234   0.1%
S    [53]    534   0.2%
T    [54]   2364   1.0%
U    [55]    161   0.1%
V    [56]    166   0.1%
W    [57]    132   0.1%
X    [58]    841   0.4%
Y    [59]    106   0.0%
Z    [5A]      2   0.0%
[    [5B]     67   0.0%
]    [5D]     67   0.0%
^    [5E]    479   0.2%
a    [61]  11934   5.0%
b    [62]   2128   0.9%
c    [63]   3465   1.5%
d    [64]   6330   2.7%
e    [65]  21212   8.9%
f    [66]   3581   1.5%
g    [67]   2899   1.2%
h    [68]   9818   4.1%
i    [69]  11036   4.6%
```

```
j    [6A]    121   0.1%
k    [6B]    897   0.4%
l    [6C]   5929   2.5%
m    [6D]   4209   1.8%
n    [6E]  10777   4.5%
o    [6F]  15224   6.4%
p    [70]   2305   1.0%
q    [71]     96   0.0%
r    [72]  10071   4.2%
s    [73]  10261   4.3%
t    [74]  15670   6.6%
u    [75]   5655   2.4%
v    [76]   1945   0.8%
w    [77]   3214   1.3%
x    [78]    273   0.1%
y    [79]   3891   1.6%
z    [7A]     76   0.0%
|    [7C]     28   0.0%
é    [82]     43   0.0%
â    [83]      1   0.0%
à    [85]      1   0.0%
ç    [87]      7   0.0%
è    [8A]      4   0.0%
î    [8C]      4   0.0%
ô    [93]     10   0.0%
```

The report (actually a single set of four columns) can be anywhere up to 256 lines long, one line for each possible arrangement among 8 off-and-on bits within one byte. The program deliberately omits a heading line and shows only one byte pattern per line. This enables us to get a clean result when we sort the report:

```
        SORT /+10 /R < SVP_TXT.BYT > SVP_TXT.BYS
```

This is the standard DOS SORT routine. The /+10 causes the sort to start at the tenth column, that is, sort by the frequency. The /R makes it a reverse frequency sort. The top end of the output looks like this:

```
\040 [20]  38883  16.3%
e    [65]  21212   8.9%
t    [74]  15670   6.6%
o    [6F]  15224   6.4%
```

```
a    [61]  11934   5.0%
i    [69]  11036   4.6%
n    [6E]  10777   4.5%
```

If we include the "locations" flag when analyzing bytes, each line of the output contains more information. Processing is only two thirds as fast. The command would be:

```
          A_BYTES -L SVP_TXT
```

The top end of the resulting SVP_TXT.LOC (note the changed report name) looks like this:

```
\012 [0A]   4117   1.7%   31 56 120 154 178 244 309 374
\015 [0D]   4117   1.7%   30 55 119 153 177 243 308 373
\032 [1A]      1   0.0%   238311
\040 [20]  38883  16.3%   6 8 14 22 25 38 40 63
!    [21]    173   0.1%   2098 2477 2800 3527 3671 6419 8389 8395
```

The report including locations may be sorted in precisely the same manner as the BYT report:

```
          SORT /+10 /R < SVP_TXT.LOC > SVP_TXT.LOS
```

If a file already exists with the target name, A_BYTES replaces the last character with a digit. BYT becomes BY0 or BY1, etc., up to BY9. The variations on LOC are LO0 through LO9.

One final note: A_BYTES can be run on a series of files with a single command:

```
          A_BYTES -L  MYFILE HERFILE FILE.WHO WHATEVER
```

++++++++++++++++++++++

4.8 Data types

++++++++++++++++++++++

With a byte survey in hand, we can begin to answer two questions about a file:

> What is in it?

> How is it presented?

The two questions are answered simultaneously during the analysis. In the next section, we will look at how data is presented. Let's focus on the first question. A file may contain one (or possibly more) of these data types:

ASCII text / extended ASCII text
text with ASCII markup codes
text with binary markup codes
text with BCD (binary coded decimal) packed numbers
text with BASIC packed numbers
text with compression substitutions
EBCDIC
binary (compression, encryption, GIF graphic
 interchange files, sound, etc.)

ASCII text consists of the letters of the alphabet, numeric digits, punctuation characters and space (hex 20 through 76) plus tab, vertical tab and new page (hex 09, 0B and 0c respectively). The newline character (hex 0A) is normally present (except in line records and some fixed length records... see below). If the file has been processed using a PC, for each newline character, there is normally exactly one carriage return (hex 0D), immediately following. PC ASCII text files often have exactly one hex 1A or CTL-Z as the last character (EOF or end of file marker).

Extended ASCII text for our purposes is ASCII text in which accented characters appear within French, German, Spanish and other foreign language words. If we allow also for punctuation common in Spanish, the extra characters are in the ranges hex 80 to 9A and A0 to A8. These are all "high-bit-set" characters, specific to PC compatible ASCII data. ("High-bit-set" means that the leading bit is turned on; hence the value is hex 80 or higher.) The count of these characters when they are part of extended ASCII text is generally lower than the count of regular vowels in the same distribution. Greek and mathematical characters (hex AB, AC, E0 to FD) may also occur in extended ASCII text. We will look later at how to check context in order to verify whether high-bit-set characters are valid text, errors within text or indications of a different file type.

Text with ASCII markup codes uses the same characters as in ASCII or extended ASCII text. It is common practice to insert special bytes or series of bytes within data to signal how the data should be displayed. This practice is called markup. Examples of items marked:

justify text (left, center, right)
select font (courier, etc.), font subsets (italics)
select type size (pica, elite, with a count)
change to/from bold
underline
protect against "widows" and "orphans"
(ensuring paragraphs stay together)
respond to new heading level

The first indicator of ASCII markup is that some characters (often "<" and ">") are present in unusually high proportion.

<u>Text with binary markup</u> codes may contain virtually all 256 characters. Null bytes (hex 00) do not occur in ASCII text, but are common in files with binary content. If you have Microsoft Word or WordPerfect files handy, try the command A_BYTES on one of them and look at the listing that results. Notice that lower case alphabetic letters still figure prominently in the distribution.

<u>Text with packed numbers</u> is found most often in COBOL style fixed length fielded records. Distribution is like that of ASCII text except that a sprinkling of other values shows up. Binary Coded Decimal (BCD) shows up in COBOL records. BASIC language data files have their own variation of packed numbers.

<u>Text with compression substitutions</u> has binary series in the midst of normal text. This is notoriously difficult to work with unless you have access to the decompression table.

<u>EBCDIC</u> data warrants special treatment the moment it is identified. Work stations or mini computers that receive data on nine track tape often deal with non-ASCII data. (The problem is rare with personal computers.) If there are many '@' symbols and no recognizable text, it is possible that you are working with EBCDIC and not ASCII. (The reason for lots of '@' characters is that the EBCDIC space character is the same as the ASCII '@'.)   Try the EBC_ASC program on a portion of the data:

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
usage:  ebc_asc  ebcdic_input  ascii_output

      Converts an EBCDIC file to ASCII.  EBCDIC (Extended Binary
      Coded Decimal Interchange Code) data is commonly produced
      by IBM mainframe computers.  ASCII (American Standard Code
      for Information Interchange) is used on personal computers
      and computers produced by the majority of manufacturers.

input:  Any EBCDIC file

output: ASCII equivalent

writeup: MIR TUTORIAL ONE, topic 4
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

If samples start to make sense when filtered through an EBCDIC to ASCII conversion, process all samples through EBC_ASC before continuing analysis. Later you will need to process all the data through the same conversion.

<u>Binary data</u>: Suppose you find in every sample that the data looks completely meaningless. There are many reasons that data dumps appear to be a jumble. Among them are:

binary compression
encryption
binary numeric data
graphics content
sound files
non-ASCII data

Attempts to decipher binary data are time consuming and expensive. The process is called reverse engineering. It is not always legal. Go back to asking people questions (the people providing the data, and maybe a lawyer) before getting too deep!

++++++++++++++++++++++++++++
4.9 Data presentation

++++++++++++++++++++++++++++++

A byte survey is also used as part of the analysis of how the data is presented. Main options are:

byte stream
line records
fixed length records
blocked records with ASCII lengths
blocked records with binary lengths

Here we simply describe the options. The topic on deblocking data will describe the programs used. Deblocking, like EBCDIC to ASCII conversion, may have to be carried out before analysis can be completed.

One stage of the indexing process will be to create a byte stream equivalent of the file to be indexed. A byte stream is the crudest form of file... simply a series of bytes in the order in which they would be displayed if sent to a screen. The ASCII source code file for each MIR program is a byte stream.

Line records are blocks of text or other data, padded with blanks out to some fixed length, very often 80. This form of data storage goes back to punch cards (remember them?) in which as many English (or other natural language) words as possible were fitted into one 80 column card. When the next word would cause overflow, it was placed at the beginning of the next card. A telltale sign of line records is a disproportionately high number of blanks and few if any linefeeds in the byte distribution.

Fixed length records also date back to punch cards. One or more cards would be divided into groups of columns, with one group assigned to each field. Sizes of fields are fixed according to the amount of data that the file designer expects... perhaps eight columns/bytes for a purchase order number, 30 for a street address, 10 for a quantity, 55 for an item description, etc. Unused fields and unused spaces within fields are left blank. Text fields are normally left justified (extra spaces at the end), and numeric fields are right justified (leading bytes either blank or zero padded). Line feeds are rare, and would occur only within a long text field. The essence of fixed length records is that location determines meaning. Like line records, fixed length records show a very high frequency of blanks (hex 20). They are likely to have more numeric digits than line records. Alternately, if the records contain Binary Coded Decimal packed numbers, there will be a near random assortment of high bit-set-bytes.

Blocked records with ASCII lengths are series of byte streams of variable length, with a measure of length at the front of each block. This length is typically 4 digits with zero padding (for example, 0032 or 3217 or 0539). The byte survey of this data shows high frequencies of digits, especially the digit zero. Line feeds show up less often than normal, or are non-existent. Otherwise, the content is fairly typical of ASCII text.

Blocked records with binary lengths have been common in library data (so called "MARC" records) and in the publishing business. They are often more sophisticated than blocked records with ASCII lengths; there may be field sub-lengths within larger blocks. Byte surveys show a small percentage of randomly distributed binary characters. Those that have the high bit set are noticeable. High-bit-off binary bytes are hidden in the frequencies of normal text characters.

++++++++++++++++++++++++++++++

4.10 Byte distributions

++++++++++++++++++++++++++++++

Here is the distribution of alphabetical characters in a sample of English text (actually drafts of seven MIR topics). The 79,657 characters were as follows:

A	312
B	101
C	288
D	220
E	288
F	152
G	108
H	112
I	406
J	7
K	17

L	175
M	190
N	187
O	292
P	219
Q	17
R	281
S	331
T	447
U	109
V	38

```
W   117
X    15
Y    81
Z     5
a  6018
b   920
c  2764
d  2775
e  9658
f  1766
g  1482
```

```
h  2959
i  5478
j    49
k   378
l  2523
m  1996
n  5194
o  6008
p  1899
q   126
r  5433
```

s	4975
t	6966
u	2189
v	834
w	1010
x	337
y	1319
z	86

Here is the same data again, arranged from most frequent to least frequent:

e	9658
t	6966
a	6018
o	6008
i	5478
r	5433
n	5194
s	4975
h	2959
d	2775
c	2764

```
l  2523
u  2189
m  1996
p  1899
f  1766
g  1482
y  1319
w  1010
b   920
v   834
T   447
```

I	406
k	378
x	337
S	331
A	312
O	292
E	288
C	288
R	281
D	220
P	219

M	190
N	187
L	175
F	152
q	126
W	117
H	112
U	109
G	108
B	101
z	86

Y	81
j	49
V	38
Q	17
K	17
X	15
J	7
Z	5

Frequency patterns in English text vary with the subject matter, but never to a large degree. Letters j and z are little used; letters e and t together with the space character almost assuredly account for one quarter to one third of all bytes present.

European languages use the high-bit-set accented characters. In French, the frequency of unaccented letter e drops somewhat in favor of è with grave accent (infrequent) and é with acute accent (common). Spanish uses more of the letter l, the tilde form of ñ and Ñ, and of course the interrogative and exclamatory symbols ¿ and ¡ at the beginning of sentences. The PC high-bit-set accented characters apply as well in Scandinavian languages, German, and so forth. But no language based on the roman character set departs altogether from the basic patterns -- frequent use of vowels, certain consonants highly favored and others marginal across all languages.

Treat each departure from typical distributions as a cue. Be prepared to analyze each exceptional byte in its contexts.

To illustrate, let's examine what significance we can attach to the example byte survey of the file SVP_TXT above. Start by making your own complete copy with the command:

```
A_BYTES SVP_TXT
```

What further analysis should we do?

> The lower case letters have a typical English text distribution... j, z and q infrequent; e, a, t and o with very high frequency.

> Upper case letters are also normally distributed for English text... Q and Z lowest, K very low (rarely the first letter in a sentence), M appearing together with the normally high vowels and T. M is frequent at the beginning

of names, and in French greetings (Monsieur, Madame).

> There are only 70 characters out of 238,312 that have the high bit set (hex 80 or higher) and all can be verified to be valid accented French characters that might appear in names of people and places in French correspondence.

> The line feed (hex 0A) and carriage return (hex 0D) each appear 4117 times, normal in an ASCII text file with 4117 lines. The locations report shows that the first eight occurrences of 0A 0D are paired right together.

> The only other non-printing characters are hex 20 (space) and hex 1A (CTL-Z or end of file in DOS). At 16.3% of occurrences, the space is the most frequent character. The CTL-Z occurs once only, in the very last byte of the file. Byte number 238311 is the last. (Offsets count from zero upward and SVP_TXT is a 238312 byte file.) CTL-Z is the standard end of file marker for ASCII text files.

> We therefore have cumulative evidence that this is an extended ASCII file. We have not yet established whether it contains markup.

> Round and square parentheses probably are matched sets ... 38 of (and), 67 of [and]. The latter are worth checking whether they occur in normal ways within text.

> The sort by frequency reveals that < and > each appear 1511 times... a tip that they might be used as matching angle brackets. The location data shows that the first eight are indeed paired only two or three bytes apart. Together the < and > bytes comprise 1.2% of the file... abnormally high for correspondence, and therefore likely part of some sort of markup code.

> The sort by frequency also shows @ and = occurring 935 times each. Are they matched also and part of a code? The first eight appearances are within 7 bytes of each other, with the @ symbol coming first each time.

> We should investigate why ^ shows up 479 times and | is present 28 times. Neither is normal for correspondence.

The next topic explains how to examine bytes in context and how to survey patterns across an entire file. Following this are topics on analysis of detail within each of the data types.