simon-5507-02-slides

Topics to be covered

- What you will learn
  - Using variable labels
  - Simple descriptive statistics
  - Printing row with smallest/largest value
  - Missing value logic
  - Simple transformations
  - Histograms
  - Correlations
  - Scatterplots

Review definitions

- Categorical
  - Small number of possible values
  - Each value associated with a category
- Continuous
  - Large number of possible values
  - Potentially any value in an interval

Speaker notes

Before we start, let’s review a couple of definitions.

A **categorical variable** is a variable that can only take on a small number of values. Each value is usually associated with a particular category.

Examples of categorical variables are

- sex (Male or Female),
- race (White, Black, Native American, etc.),
- cancer stage (I, II, III, or IV),
- birth delivery type (Vaginal, C-section).

A **continuous variable** is a variable that can take on a large number of possible values, potentially any value in some interval.

Examples of continuous variables are

- Birth weight in grams,
- gestational age,
- fasting LDL level.

There are some variables that are on the boundary between categorical and continuous, but it is not worth quibbling about here.

The point to remember is that the types of graphs that you use and the types of statistics that you compute are dependent on many things, but first and foremost on whether the variables are categorical, continuous, or a mixture.

Today, you will see examples involving mostly continuous variables.

Semicolons are important

- Ends every SAS statement
- Easy to forget
- Use this to your advantage
  - Several short lines
  - Indent continuations

Speaker notes

Before I go too far, let me mention and important thing. Every SAS statement ends in a semicolon. This is important. You will forget a semicolon and it will lead to a cryptic error message. So here’s a quick hint. If you get an error message on a certain line of code, look to see if you forgot a semicolon on the previous line. It happens to me all the time and I’ve been using SAS for decades.

Example of stretching statement across multiple lines.

One long line

```
statement option1 option2 option3 option4;
```

versus several short lines.

```
statement
  option1
  option2
  option3
  option4;
```

Speaker notes

The use of semicolons is nice, in a way, because it allows you to stretch a complicated SAS statement across two or more rows of your program. This can often make your program more readable. It is hard to read a long line of code. Your eye has to scan left to right and you can sometimes lose track of which line you are on. Most newspapers place their articles in narrow columns because it makes them easier to read.

There is no official rule of thumb on this, but I do try to keep my lines below 50 characters. I also try to indent substatements with a data step or procedure. I use blank lines between data steps and procedures.

Don’t obsess about this now, but you’ll see a fairly consistent coding style that I use for my SAS code. You don’t have to follow my format, of course, which might be a bit too extreme for your tastes. Just experiment with things a bit until you can settle on a layout that you are comfortable with.

Rules for variable names (1/2)

- Can use mix of
  - letters (A-Z, a-z),
  - numbers (0-9)
  - underscore (_)
  - no blanks, no symbols

Speaker notes

There are important rules for variable names in SAS. You can use a mix of letters, numbers, and the underscore. You can’t use blanks or any special symbols like the dollar sign ($) or the dash/minus sign.

I’m using the variable names provided but if you create your own names, use brief (but descriptive) name for EVERY variable in your data set. There’s no precise rule, but names should be around 8 characters long. Longer variable names make your typing tedious and much shorter variable names makes your code terse and cryptic.

I’m a bit more terse with these variable names than I normally would be just to reduce the amount of typing you have to do.

You should avoid special symbols in your variable names especially symbols that are likely to cause confusion (the dash symbol, for example, which is also the symbol for subtraction). You should also avoid blanks. These rules are pretty much universal across most statistical software packages.

Rules for variable names (2/2)

- Can’t start with a number
  - “a1” but not “1a”
- Capitalization not important
  - BMI, Bmi, bmi are same
- Up to 32 characters in length

Speaker notes

You can’t start with a number. So “a1” is okay, but “1a” is not.

Capitalization is not important in SAS. So you can call your variable “BMI” with all caps, or “Bmi” with mixed capitals or “bmi” with all lower case. SAS treats all of these the same.

Your variable name has to be 32 characters or less in length.

I’m using the variable names provided but if you create your own names, use brief (but descriptive) name for EVERY variable in your data set. There’s no precise rule, but names should be around 8 characters long. Longer variable names make your typing tedious and much shorter variable names makes your code terse and cryptic.

I’m a bit more terse with these variable names than I normally would be just to reduce the amount of typing you have to do.

Recommendations for variable names (1/2)

- Avoid generic names (x1, var01, etc.)
- Keep it short
  - Use commonly known abbreviations…
  - …but nothing cryptic
- Use all lower case (age, not AGE or Age)

Speaker notes

Your variable names should by descriptive. Avoid generic names like x1, var01, and so forth.

Keep things short. You can use commonly known abbreviations, such as “wt” for “weight”. But avoid any cryptic abbreviations.

I like to keep everything in lower case. It is more readable than all upper case, and easier to remember than a mixture of upper and lower case. Some people prefer an initial upper case, and there’s nothing wrong with that. It is important, however, to be consistent.

Recommendations for variable names (2/2)

- Separate words with underscores
  - fat_brozek, not fatbrozek
- Alternative: CamelCase
  - FatBrozek
- Caution: Writer’s Exchange website
  - www.writersexchange.com

Speaker notes

You can use two or three short words in a variable name, but be sure to separate them using underscores. So the variable for percentage body fat as measured by Brozek’s equation is “fat” and “brozek” separated by an underscore. Some people prefer CamelCase, where each word starts with an initial capital letter: capital F fat, capital B brozek.

The one thing you do want to avoid is just running two or three words together and all lower case. There’s a story about a group that started up in the era before the web called Writer’s Exchange. As you can guess this was a resource for new authors. When the web came out, they decided to put their resources up on a website, www.writersexchange.com. That seemed logical enough, but then someone notices that you could read the website as “www dot writer sex change dot com”. Not exactly the image they wanted.

SAS variable labels (1/2)

- Longer description of a variable
  - Can include blanks, special symbols
  - Internal documentation
  - Labels substituted on some (but not all) output
- Required in this class (see grading rubric)

Speaker notes

SAS offers an opportunity for you to add documentation to your program about individual variables. These are called variable labels. They have almost no restrictions. You can use blanks, or special symbols like a dollar sign or a dash. The documentation that variable labels provide is mostly internal, but these labels are substituted in a few places like some graphs.

Every variable in a SAS program should have a label. This label will make some (but not all) of the SAS output more readable. it is also part of the internal documentation of your program. Note that some of these labels do not fit well in this Powerpoint slide, but that’s okay.

SAS variable labels (2/2)

- Recommendations for variable labels
  - Judicious use of upper and lower case
  - Spell out abbreviations
  - Specify units of measurement
  - Any other important details

Speaker notes

What makes a good variable label? First take advantage of a mixture of upper and lower case to make your labels more readable. Spell out any abbreviations, even obvious abbreviations. If your variable has a measurement unit, specify that unit in your variable label. If there are other important details, put these in the variable label as well.

Every variable in a SAS program should have a label. This label will make some (but not all) of the SAS output more readable. it is also part of the internal documentation of your program. Note that some of these labels do not fit well in this Powerpoint slide, but that’s okay.

Documenting your program. (1)

```
* 5507-02-simon-continuous-variables.sas
  author: Steve Simon
  date: created 2021-05-30
  purpose: to work with continuous variables
  license: public domain;

* datasets created in this program
    body, original data
    body1, row with ht=29.5 removed
    body2, ht=29.5 converted to missing
    body3, ht_cm calculated;
```

Speaker notes

- Comments on the code: Documenting your program.

The demo programs in class will provide a short explanation of pretty much every new piece of SAS code. You can strip these comments out in your program.

Do not use so many comments in YOUR code. It is still a good idea, however, to include comments on the code for anything unusual or difficult in SAS to remind yourself and others what is going on.

If you borrowed and adapted any code you found on the Internet, you should also document where you found it.

If you used a large language model to generate SAS code, you should acknowledge this and include the prompt(s) that you used.;

Specifying file locations (2)

```
filename rawdata
  "../data/fat.txt";

libname module02
  "../data";

ods pdf file=
  "../results/5507-02-simon-demo.pdf";
```

Speaker notes

- Comments on the code: Specifying file locations

You should already be familiar with this.

The filename statement tells you where the raw data is stored.

The libname statement tells you where SAS will store any permanent datsets.

The ods statement tells you that SAS is going to store the results with a particular filename and use pdf format.;

Reading in the data (3)

```
data module02.body;
  infile rawdata;
  input
    case
    fat_brozek
    fat_siri
    dens
    age
    wt
    ht
    bmi
    ffw
    neck
    chest
```

Speaker notes

- Comments on the code: Reading in the data

This is the code to input all the variables in this data set. The infile statement refers to a location defined with the earlier filename statement.;

Adding labels (4)

```
label
    case="Case number"
    fat_brozek="Fat (Brozek's equation)"
    fat_siri="Fat (Siri's equation)"
    dens="Density"
    age="Age (yrs)"
    wt="Weight (lbs)"
    ht="Height (inches)"
    bmi="Body mass index (kg/m^2)"
    ffw="Fat Free Weight (lbs)"
    neck="Neck circumference (cm)"
    chest="Chest circumference (cm)"
    abdomen="Abdomen circumference (cm)"
    hip="Hip circumference (cm)"
```

Speaker notes

- Comments on the code: Adding labels

The label subcommand is part of the data step. Specify the variable name and follow the equal sign with a longer description in quotes. You can use space, punctuation, and special symbols.;

Adding extra information (5)

```
* Some additional details about this data:

  Brozek's equation is 457/Density - 414.2

  Siri's equation is 495/Density - 450

  Abdomen circumference is measured at the
  umbilicus and level with the iliac crest

  Wrist circumference is distal to the
  styloid processes;
```

Speaker notes

- Comments on the code: Adding extra information

I am including some additional details that would not fit easily into the variable labels. How much documentation you include is a judgment call. I am including this extra documentation just to remind you that such documentation is possible.;

The footnote subcommand (6)

```
proc print
    data=module02.body(obs=10);
  var case fat_brozek fat_siri dens age;
  title1 "Ten rows and five columns";
  title2 "of the body data set";
  footnote1 "Created by Steve Simon on &sysdate using SAS &sysver";
run;
```

Speaker notes

- Comments on the code: The footnote subcommand

The footnote statement is part of any SAS proc. It places information at the bottom of the page of any output produced by that proc. It is a conventient place to note who wrote the program, when it was run (using the sysdata macro variable), and what version of SAS was used (using the &sysver macro variable).;

Displaying metadata (7)

```
proc contents
    data=module02.body;
  title1 "Internal description of body dataset";
run;
```

Speaker notes

- Comments on the code: Displaying metadata

The contents procedure produces information about any dataset produced by SAS, including both temporary datasets (one part names) and permanent datasets (two part names).

For a dataset that you just created and one that is not all that complicated, using proc contents is overkill. I am showing it so you will know how to use proc contents for very complex datasets, especially ones that were created by someone other than yourself.

SAS produces a lot of information and much of it is only relevant for advanced applications. You have to wade through the details to get the important information. The important information is

- date created,
- date modified,
- observations,
- variables, and
- filename.;

Live demo, 1

- data step
  - label subcommand
- contents procedure

Break #1

- What you have learned
  - Using variable labels
- What's coming next
  - Simple descriptive statistics

Simple descriptive statistics

- Always look at first
- Is mean high, normal, or low?
- Is data spread out or tight?
  - Zero standard deviation is a red flag
- Are minimum and maximum reasonable?

Speaker notes

You should always look at simple descriptive statistics first before embarking on any data analysis. This is especially true for very complex data analyses. Often the complex analyses are largely opaque and difficult to follow. You will have a better time understanding how the mixture of variables in these models behave if you first have an understand how each variable behaves individually.

First examine the mean relative to your expectations. Is it about what you'd expect or is it low or high relative to your expectations?

Look at the standard deviation. A rough rule of thumb is that about 68% of the data will lie within one standard deviation of the mean and that about 95% of the data will lie within two standard deviations of the mean. That won't be true for highly skewed distributions, but it is still worthwhile to think about these calculations. Do they indicate that the data is widely spread or tightly packed?

Watch out for any variables that have zero for a standard deviation. This tells you that the data is constant (no variation), which would preclude that variable's use in most data analyses.

Look at the minimum and maximum values. Are they extreme to cause concern? If they are outside the range of possible values, investigate. Maybe there is a data entry error.

Computing simple statistics (8)

```
proc means
    n mean std min max
    data=module02.body;
  var ht;
  title1 "Descriptive statistics for ht";
  title2 "The mean is normal for adults";
  title3 "The standard deviation shows tightly packed data";
  title4 "The maximum value is reasonable";
  title2 "The minimum is very low";
run;
```

Speaker notes

- Comments on the code: Computing simple statistics

The means procedure will produce descriptive statistics for your data. By default, it will produce the count of non-missing values, the mean, the standard deviation, and the minimum and maximum values, but I am listing them explicitly here, just for show.

The data option tells SAS which data set you want descriptive statistics on, and the var statement tells SAS which variable(s) you want descriptive statistics on.;

Live demo, 2

- means procedure
  - n option
  - mean option
  - std option
  - min option
  - max option

Break #2

- What you have learned
  - Simple descriptive statistics
- What’s coming next
  - Printing row with smallest/largest value

Sorting your data

- Uses the sort procedure
- Specify the dataset with data=
- Specify the sorting variable with the by subcommand
    - Use descending keyword to sort in reverse order

Speaker notes

If you have an outlier, you might want to look at that value in the context of the other values in its row of data.

You do this by sorting the data so that the shortest subject becomes the first row of the data and the tallest subject becomes the last. Then print just the very first row of your data.

Warning! Be careful about sorting your data if you can’t get the data easily back to the original order. It might be okay, but there are times when you’d like your data all the way back and that means data in the original order. This data set has a case variable that you can resort by in order to get back ot the original order.

If you don’t have a case variable, store the sorted data in a separate location: something along the lines of proc sort data=x out=y.

Printing row with smallest or largest value

- Investigate other variables associated with outlier
- Is the data shifted left or right?
- Are other values consistent with the outlier?

Speaker notes

If you spot an outlier, an unusually large or small value for a variable, look at the rest of the data for that individual. That may offer a clue about why that outlier is so extreme.

Sometimes data gets misaligned read during the process of importing it into SAS. It can occur because of a missing or duplicate delimiter. We'll talk about delimiters in the next module. Look to see if there is an unexpected missing value or if the data for one variable looks like it fits better for the variable to the left or to the right.

If you don't see any shifting, look to see if there is coherence among all the variables. A patient with a very small value for height, for example, should also have a very small value for weight.

Printing row with smallest value (9)

```
proc sort
    data=module02.body;
  by ht;
run;

proc print
    data=module02.body(obs=1);
  title1 "The row with the smallest ht";
  title2 "Note the inconsistency with wt";
run;
```

Speaker notes

- Comments on the code: Printing row with smallest value

The sort procedure will take a dataset specified by data= and arrange the data in order from smallest to largest using the variable specified in the by subcommand.

If you want to store the sorted data in a different location, use the out= option;

Printing row with largest value (10)

```
proc sort
    data=module02.body;
  by descending ht;
run;

proc print
    data=module02.body(obs=1);
  title1 "The row with the largest ht";
  title2 "This seems quite normal to me";
run;
```

Speaker notes

- Comments on the code: Printing row with largest value

The descending option sorts the data in reverse order.;

Live demo, 3

- sort procedure
    - by subcommand
        - descending option

Break #3

- What you have learned
  - Printing row with smallest/largest value
- What’s coming next
  - Missing value logic

What to do with outliers

- Depends on the context, ask for help!
  - Live with it
  - Delete the entire observation
  - Convert the value to missing

Speaker notes

With this outlier on the low end, you might consider doing nothing other than noting the unusual value.

Alternately, you could delete the entire row associated with this value. Finally, you might consider converting the small ht value to a missing value code.

There is no one method that is preferred in this setting. If you encounter an unusual value, you should discuss it with your research team, investigate the original data sources, if possible, and review any procedures for handling unusual data values that might be specified in your research protocol.

Notice that I store the revised data sets with the row removed and with the 29.5 replaced by a missing value in different data frames. This is good programming practice. If you ever have to make a destructive change to your data set (a change that wipes out one or more values or a change that is difficult to undo), it is good form to store the new results in a fresh spot. That way, if you get cold feet, you can easily backtrack.

We’ll use the data set with the 29.5 changed to a missing value for all of the remaining analyses of this data set.

How to handle outliers

- No option is best in all cases
  - Live with them
  - Remove the entire row
  - Convert outlier to missing
- Always report clearly

Speaker notes

When you encounter outliers, you have several options. Often the best approach is to live with them. You might report both a mean and a median, for example.

You could remove a patient entirely from the analysis. This is a pretty radical step and should be done if you can argue that all of the patient’s data is hopelessly compromised. You might also argue that the patient clearly is ineligible for the study and should never have been let in.

You could also convert the outlier to a missing value. This is also a big step, but at least it allows you to use the rest of the data for that patient.

Generally, an extreme value by itself does not support removal of a patient or converting an outlier to missing. You need to provide external evidence. A notation in a lab notebook, for example, of sample handling problems, might justify this step. You could also remove an outlier if you could argue that the value is incompatible with life, such as a patient with a body mass index of 5.1.

Importing missing values

- Different coding schemes
  - dot (.) or blank ( ), the SAS standard
  - other symbols (*, ?)
  - NA, the R standard
  - NULL, the SQL standard
  - Extreme numbers (-1, 9, 99, 999)
  - Blank ( ) or empty ()

Speaker notes

Your data set may arrive with missing values in it already. Data might be designated as missing for a variety of reasons (lab result lost, value below the limit of detection, patient refused to answer this question) and how you handle missing values is way beyond the scope of this class. Just remember to tread cautiously around missing values as they are a minefield.

Missing data could be coded as a single dot (.), which is the SAS standard for missing numeric data. You might also see some other symbol such as an asterisk or a question mark. R uses NA for missing and SQL uses NULL.

You may see extreme numeric values, extreme meaning outside a reasonable range, as codes for missing values. A birthweight of -1, for example, probably indicates missingness, as a baby weighing -1 kilograms would float up to the ceiling after being born. Other times, of course, -1 would not work, such as when recording your checking account balance the day before payday.

Sometimes, one or more nines would indicate a missing value. A Likert scale item coded 1 for strongly disagree through 5 for strongly agree will often code a non-response as 9. Make sure you have enough nines. An IQ score would not use 99 as the missing code because that is a legitimate value. A code of 999 makes more sense here because even Albert Einstein's IQ falls well below 999.

One or more blanks might represent missing or an empty field (a zero length string) might represent missing. I discourage the use of blanks or empty fields for several reasons. First, when your computer diplays a single blank, it looks just like an empty field. Second, an empty or blank field might represent not missingness, but rather that the data entry person has not gotten to enter that value yet. Third, some systems (not SAS) might convert an empty field to zero, which is often not what you want.

Advice on importing missing values

- Read the data dictionary
- Always ask **WHY** a value is missing
- Convert any non-standard missing codes
  - if iq=999 then iq=.

Speaker notes

If there is a data dictionary, read it for information about how missing values are coded. If there is no data dictionary, you'll have to use a bit of guesswork, but you can usually figure things out.

It's important to ask early why a missing value is missing.

If the data you are importing uses anything other than a dot to designate missing, make sure you convert to the standard code for missing right away. You don't want to include a few 999's in your estimate of average IQ because it will make the group look a lot smarter than they really are.

Missing value logic in SAS

- Stored internally as most extremely negative number
  - Approximately -1.8×10^{308} (on most computers)
- Can identify with = . or missing()
  - Differs from R
- Use caution with less than/greater than comparisons
  - age < 18 will include children **AND** missing ages
  - use age ^= . & age < 18 instead

Speaker notes

SAS stores missing values internally as the most extremely negative number. On most systems that would be -1.8 time ten to a ridiculously large number. It is more extreme than a negative googol (-1 times 10 to the 100 power) and far larger in absolute value than the number of atoms in the universe (approximately 10 to the 78 power or 10 to the 82 power). So you are unlikely to use such an extremely negative number in any real world calculations.

You can identify a missing value using the missing function or the = . comparison. This is handy. In R trying to use an equality comparison involving missing values will evaluate not to true or false but to missing. This produces a complex three valued logic system. In SAS, any equality comparison will produce either true or false and no third value. This has its advantages.

The disadvantage occurs when using less than and greater than comparisons. You might want to identify children in your dataset using age < 18, but this would include children and anyone in your dataset with a missing age. You should exclude missing values explicitly first before doing any less than or greater than calculations. That’s a good rule to follow in R as well.

Removing a row of data (11)

```
data module02.body1;
  set module02.body;
  if ht = 29.5 then delete;
run;
```

Speaker notes

- Comments on the code: Removing a row of data

The if … then subcommand inside a data step will evaluate a logical comparison betweeh if and then and execute the subcommand following then only if the logical comparison evaluates to true.

The delete subcommand remove a row from the dataset;

Converting outlier to missing (12)

```
data module02.body2;
  set module02.body;
  if ht=29.5 then ht=.;
run;
```

Speaker notes

- Comments on the code: Converting outlier to missing

This code converts the height to a missing value, but keeps the rest of the variables for that particular subject.;

Printing negative values (wrong way) (13)

```
proc print
    data=module02.body2;
  where ht < 0;
  title1 "Printing negative values for ht (wrong way)";
  title2 "Use where ht ^= . & ht < 0 instead";

run;
```

Speaker notes

- Comments on the code: Printing negative values (wrong way)

The where subcommand processes the data in a SAS procedure and only includs that data in the procedure if the logical comparison evaluates to true.

This code is actually an example of what you should NOT do, because missing values will produce a true result for less than comparisons and false results for a greater than comparison. You should always include a check for missingness as part of any logical comparison.;

Counting missing values (14)

```
proc means
    n nmiss mean std min max
    data=module02.body2;
  var ht;
  title1 "There is one missing value";
run;
```

Speaker notes

- Comments on the code: Counting missing values

If you are concerned at all about missing values (and you should be), ask for the number of missing values in proc means using nmiss.;

Live demo, 4

- data step
  - set subcommand
  - if … then subcommand
- print procedure
  - where subcommand
- means procedure
  - nmiss option

Break #4

- What you have learned
  - Missing value logic
- What’s coming next
  - Simple transformations

Transforming values

- Use data step to create a new variable
  - Unit conversion: temperature = 5/9 * (temperature - 32)
  - New variable: bmi = wt_kg / ht_m^2

Speaker notes

Sometimes you want to alter some of your data. This could be a units conversion, a body mass index calculation, or something else. This is done in the data step. You could do it along with the input statement, but it is often valuable to read in the data as is, and make changes which are stored in a different dataset.

Non destructive transformations, Different variable name

```
data name1;
  set name1;
  wt_kg = wt / 2.2;
run;
```

Speaker notes

Sometimes you want to alter some of your data. This could be a units conversion, a body mass index calculation, or something else. This is done in the data step. You could do it along with the input statement, but it is often valuable to read in the data as is, and make changes which are stored in a different dataset.

If you make a transformation, you should do it in a non-destructive way, meaning a way that allows you to revert back, if needed. Do this by transforming into a new variable, as shown here or by storing the transformed data in a new dataset, as shown on the next slide.

Non destructive transformations, Different dataset name

```
data name2;
  set name1;
  wt = wt / 2.2;
run;
```

Transforming values (15)

```
data module02.body3;
  set module02.body;
  check_bmi = (wt / 2.2) / (ht / 39.37)**2;
  check_ht = sqrt((wt / 2.2) / bmi) * 39.37;
  check_wt = (bmi * (ht / 39.37)**2) * 2.2;
run;

proc print
    data=module02.body3;
  var ht check_ht wt check_wt bmi check_bmi;
  where ht=29.5;
  title1 "Recalculating ht, wt, and bmi";
  title2 "Assuming two out of three are correct.";
```

Speaker notes

- Comments on the code: Transforming values

Transform variables inside of a data step using standard computer formulas. SAS uses a double astersk (**) instead of a caret (^) to denote raising to a power.;

Live demo, 5

- No new keywords

Break #5

- What you have learned
  - Simple transformations
- What's coming next
  - Histograms

Historial sidenote, IBM mainframe printer

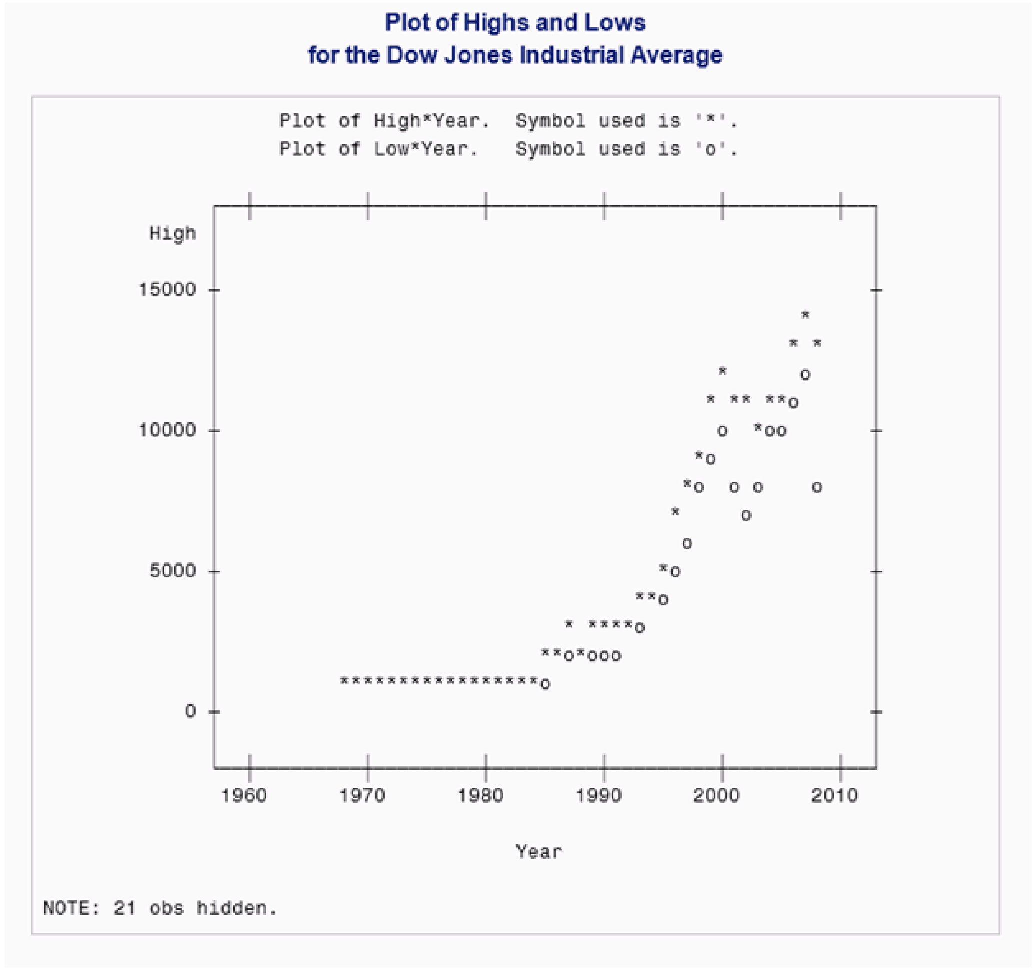
Speaker notes

When I was in graduate school, SAS would only print to a large printer attached to the IBM mainframe computer. I don’t know the exact model, but it did look something like this.

This printer was fast, probably faster than many of today’s printers. But it has a limited character set, mostly just the things you would find on a manual typewriter. It could only print in one color, black. And it could only use a fixed width font, not too much different that what you see today with the Courier New font.

This led to some serious compromises that you had to make with

Historical sidenote, Sample output from proc plot



```
Plot of High*Year.  Symbol used is '*'.
Plot of Low*Year.   Symbol used is 'o'.
```

```
NOTE: 21 obs hidden.
```

Speaker notes

Here’s an example of output from proc plot. It’s actually a bit better than what I had in graduate school because this graph is using some of the box drawing characters that cam with the IBM personal computers.

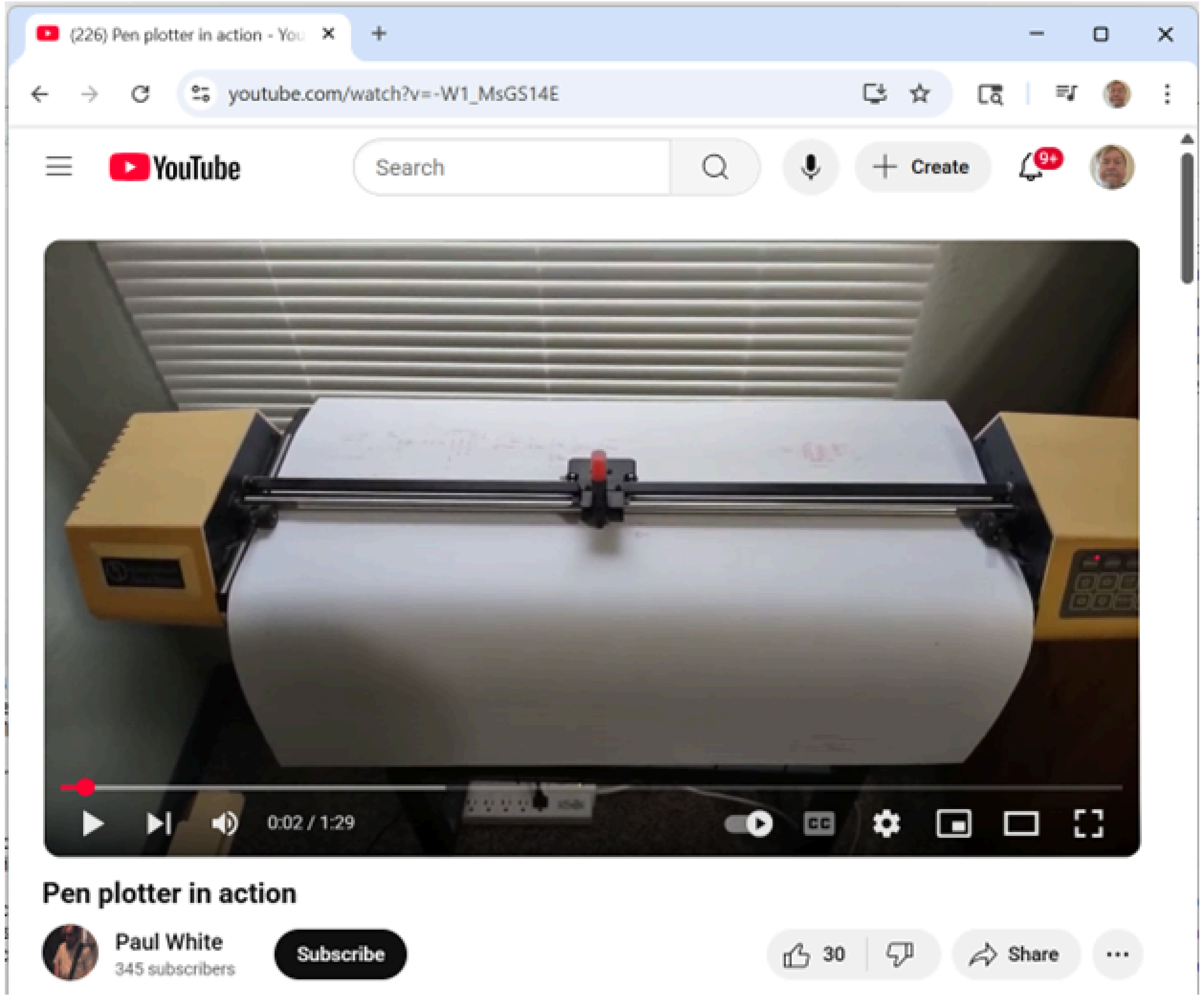
Historical side note, box drawing characters

```
Box drawing characters

░   ▒   ▓   │   ┤   ╡   ╢   ╖   ╕   ╣   ║   ╗   ╝   ╜   ╛   ┐
└   ┴   ┬   ├   ─   ┼   ╞   ╟   ╚   ╔   ╩   ╦   ╠   ═   ╬   ╧
╨   ╤   ╥   ╙   ╘   ╒   ╓   ╫   ╪   ┘   ┌   █   ▄   ▌   ▐   ▀

Before box drawing characters

| - +
```

Speaker notes

The box drawing characters allowed you to place a single line frame, a double line frame, or a thick frame around other text, and it looked pretty cool. Before box drawing characters, you had just three characters for drawing frames. The vertical bar was used for the Y axis, the minus sign for the X axis, and the plus sign for corners and tick marks.

Historical side note, Pen plotter



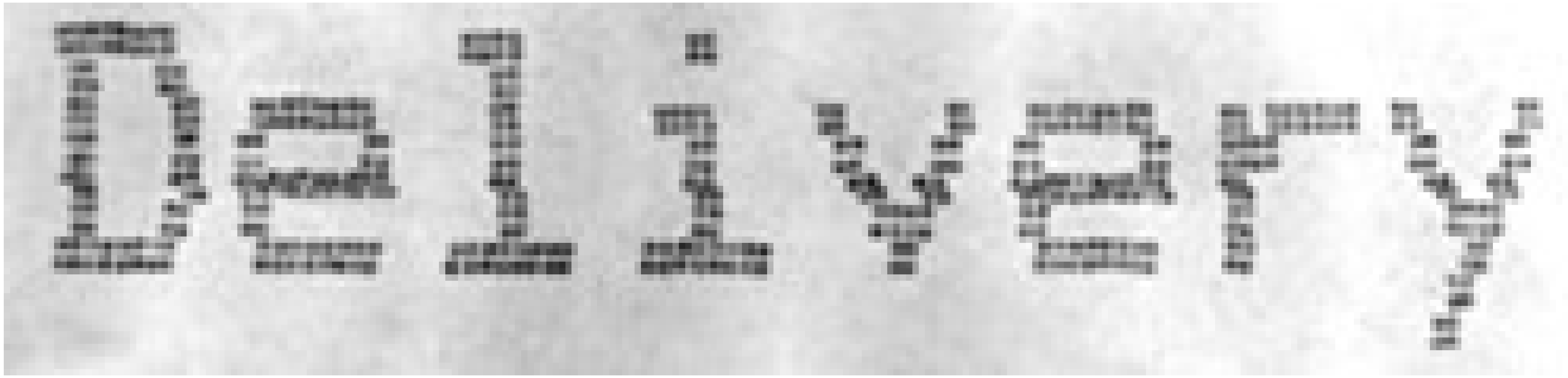
Speaker notes

There were pen plotters back then. It worked kind of like an etch-a-sketch. One part of the pen plotter moved the paper in a vertical direction, and the pen slid back and forth in a horizontal direction. It was a lot of fun to watch.

The pen plotter allowed you to use multiple colors, but limited typically to four, six, or eight different pens. It also allowed you to draw diagonal lines and curves.

The interesting thing about the pen plotter was when you have a lot of data points overlapping. The pen plotter would put so much ink down that parts of the paper would get soaked and start to tear.

Historical note, Dot matrix printer



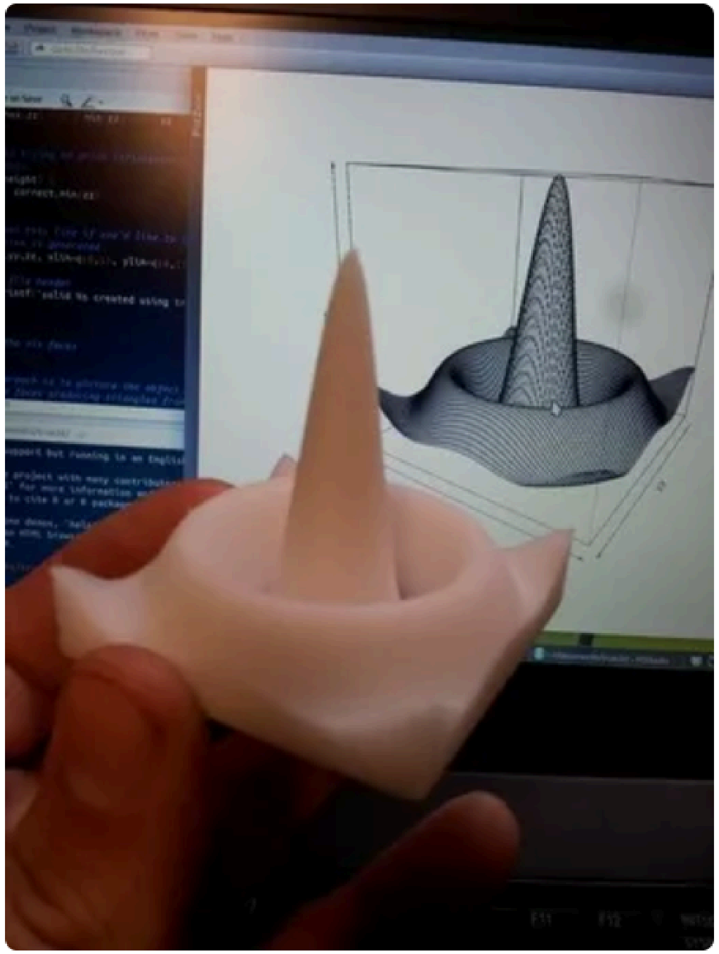
Today’s technology, Ink jet or laser printer

- 300 dots per inch or better
- Up to 16.7 million colors
  - CMYK system versus RGB system

Speaker notes

Today, most statistical graphs are printed using ink jet or laser technology. Starting with the Apple LaserWriter in 1985, laser printer technology allowed for very precise graphs with fine details.

Today’s technology, 3D printers (SAS does not support these)



Speaker notes

As far as I know, you cannot use SAS to create solid objects using a 3D printer. It is possible to do this in R, but it is not easy. I expect this to become possible and easy for both R and SAS in the near future.

Drawing histograms

- Histograms can assess normality/non-normality
  - Skewness
  - Bimodal distributions
  - Outliers
- How many bars? Multiple recommendations
  - Five to twenty bars
  - Square root of n bars
  - Trial and error

Speaker notes

A histogram can help you assess the distributional pattern of your data. Is it normally distributed (the classical bell shaped curve)? If so, you can make fairly strong assumptions about the data. The mean plus or minus one standard deviation will contain roughly 68% of the data. The mean plus or minus two standard deviations will contain roughly 95% of the data.

Normality is also an assumption needed for a variety of confidence intervals and hypothesis tests. It is not a life-or-death assumption, especially when your sample size is large. Also, the normality assumption is often better assessed using residuals from your statistical model.

The other nice thing about the histogram is that it can identify what type of deviation from normality you might have. Identifying bimodality is the trickiest, because if you have too few bars you might miss the gap between modes. If you have too many bars, you might see multiple modes because of the spikiness that is artificially produced when you have too many bars.

There is no simple rule for deciding how many bars you should have in your histogram. Once sources suggests that 5 to 20 is a reasonable number of bars. Another suggests that the number of bars should be roughly equal to the square root of the sample size. To be honest, the best approach might be to try several different choices and see which one gives a reasonable picture of the distribution of your variable.

Drawing a histogram (default) (16)

```
proc sgplot
    data=module02.body2;
  histogram ht;
  title1 "Histogram with default bins";
run;
```

Speaker notes

- Comments on the code: Drawing a histogram (default)

The sgplot procedure produces a wide range of visualizations. The histogram subcommand will produce a histogram.;

Drawing a histogram (more bars) (17)

```
proc sgplot
    data=module02.body2;
  histogram ht / binstart=60 binwidth=1;
  title "Histogram with narrow bins";
run;
```

Speaker notes

- Comments on the code: Drawing a histogram (more bars)

The binstart option tells SAS that one of the bins (not necessarily the first one) starts at 60. The bindwidth option tells SAS that each bin has a width of 1 unit (or plus or minus 0.5 units);

Drawing a histogram (fewer bars) (18)

```
proc sgplot
    data=module02.body2;
  histogram ht / binstart=60 binwidth=5;
  title "Histogram with wide bins";
run;
```

Speaker notes

- Comments on the code: Drawing a histogram (fewer bars)

This code produce wider bins, 5 units (or plus or minus 2.5 inches).;

Live demo, 6

- sgplot procedure
    - histogram subcommand
        - binstart option
        - binwidth option

Break #6

- What you have learned
  - Histograms
- What’s coming next
  - Correlations

Correlations

- Informal interpretation
  - between +0.7 and +1.0: strong positive association
  - between +0.3 and +0.7: weak positive association
  - between -0.3 and +0.3: little or no association
  - between -0.3 and -0.7: weak positive association
  - between -0.7 and -1.0: strong negative association

Speaker notes

The correlation coefficient is a single number between -1 and +1 that quantifies the strength and direction of a relationship between two continuous variables. As a rough rule of thumb, a correlation larger than +0.7 indicates a strong positive association and a correlation smaller than -0.7 indicates a strong negative association. A correlation between +0.3 and +0.7 (-0.3 and -0.7) indicates a weak positive (negative) association. A correlation between -0.3 and +0.3 indicates little or no association.

Don't take these rules too literally. You're not trying to make definitive statements about your data set. You are just trying to get comfortable with some general patterns that occur in your data set. A complex and definitive statistical analysis will almost certainly not agree with at least some of the preliminary correlations noted here.

The corr procedure produces, by default, a square correlation matrix of all the numeric variables. The noprob and nosimple options cut down on the amount of information printed. The with statement produces a rectangular correlation matrix.

Computing correlations (default) (19)

```
proc corr
    data=module02.body2
    noprint
    outp=correlations;
  var fat_brozek fat_siri;
  with neck -- wrist;
run;
```

Speaker notes

- Comments on the code: Computing correlations (default)

The corr procedure produces correlations. The noprint keytword tells SAS to not print anything. The noprint option is commonly used when a SAS proc is used to produce a new dataset rather than output.

The outp option tells the procedure which dataset to create that will contain the correlations.

The var subcommand tells SAS what variables to correlate.

If there is no with subcommand, the variables identified in var are correlated with each other in a square correlation matrix.

If there is a with subcommand, the corr procedure will produce a rectangular matrix correlating each varialbe in var with each variable in with.

The double dash (–) tells SAS to use neck, wrist and every variable in between them.;

Processing correlations (20)

```
data correlations;
  set correlations;
  if _type_ NE "CORR" then delete;
  drop _type_;
  fat_brozek=round(fat_brozek, 0.01);
  fat_siri=round(fat_siri, 0.01);
run;
```

Speaker notes

- Comments on the code: Processing correlations

The output dataset in SAS includes a lot of descriptive statistics other than just the correlations. The if … then statement will delete anything other than a correlation.

The drop subcommand will remove the *type* variable, as it is no longer needed.

The round function will round to the nearest multiple of 0.01.;

Sorting the correlations (21)

```
proc sort
    data=correlations;
  by descending fat_brozek;
run;

proc print
    data=correlations;
  title1 "Abdomen, hip, and chest show the strongest correlations";
run;
```

Speaker notes

- Comments on the code: Sorting the correlations

While not required, sorting the correlations can sometimes help interpretability.;

Live demo, 7

- corr procedure - noprint option - outp option with subcommand
- drop subcommand (data step)
- round function

Break #7

- What you have learned
  - Correlations
- What’s coming next
  - Scatterplots

Drawing a scatterplot (22)

```
proc sgplot
    data=module02.body2;
  scatter x=abdomen y=fat_brozek;
  pbspline x=abdomen y=fat_brozek;
  title1 "Simple scatterplot shows a strong positive trend";
  title2 "It levels off for high values.";
  title3 "This may be due solely to a single outlier on the high end";
run;
```

Speaker notes

- Comments on the code: Drawing a scatterplot

The scatter subcommand in the sgplot procedure produces a scatterplot.

The pbspline subcommand adds a smoothing to help you visualize whether the trend is linear or not.;

Closing the pdf file (23)

```
ods pdf close;
```

Speaker notes

- Comments on the code: Closing the pdf file

I always seem to forget this last statement and then I get upset with SAS for not providing the PDF output. But SAS can’t produce the PDF output until you tell it you are done. So don’t yell at your computer when it’s your own darn fault (just like Jimmy Buffet in the Margaritaville song).

If you don’t get any pdf file when you are done, or your pdf file is the one left over from a previous analysis, it’s probably because you forgot this last very important statement.;

Summary

- What you have learned
  - Using variable labels
  - Simple descriptive statistics
  - Printing row with smallest/largest value
  - Missing value logic
  - Simple transformations
  - Histograms
  - Correlations
  - Scatterplots