Data \_Null\_;

Put “Hello, world.”;

Run;

Data ufos;

Input start time8. +3 end time8. Reports 5. +3 color $char8.;

If end >= start then length = end – start;

Else length = end + 86400 – start;

Cards;

Proc summary data =ufos;

Class color;

Var reports length;

Output out=ufostat sum=;

Data;

Set ufostat;

Avg\_rate = reports/length;

Proc print;

Run;

CARDS, LINES, and DATALINES are the same function. All produce lines of data following the statement.

CARDS4 – specifies end of lines with four semicolons

ENDSAS; cannot be combined with any other statement

%LET N = 45A;

DATA GROUP&N;

%STORY(GOLDILOCKS, BEAR, BEAR, BEAR)

%MACRO = Beginning of Macro.

%MEND = End of Macro.

Data \_null\_;

F=5;

I+1;

Named Range – If a data step has the variables X, Y, Z, and TOTAL defined in that order, then X—TOTAL represents the variables X Y Z TOTAL.

\_NUMERIC\_ - Retrieve all numeric variables.

\_CHAR\_ - Retrieve all character variables.

\_ALL\_ - Retrieve all variables.

ARRAY SEQUENCE {5} FIRST SECOND THIRD FOURTH FIFTH; /\* This array statement creates an array SEQUENCE with the 5 elements. \*/

SEQUENCE{2} – Retrieves the element SECOND.

Constants are values that appear in a program. E.g. -0.0087 or ‘Good Morning’

Macro variable constants should be enclosed in single quotes. E.g. ‘AT&T’

DATA and OUTPUT is used to output SAS datasets, while PUT and FILE are used for TEXT output.

DATA A B; or DATA \_NULL\_;

Input in a data step is controlled by the SET, MERGE, and UPDATE statements for SAS datasets, while INFILE and INPUT are used for Text input. If the file is a text file, and INFILE statement identifies it and an INPUT statement interprets it.

INFILE D;

INPUT E F G H I;

Proc datasets library=work;

Libname brady clear;

Proc Contents data=brady.loan;

SET, MERGE, and UPDATE used initial input step

Data NAMES (keep=name abbr area pop rank capital);

Options missing=’.’

Options ERRORABEND;

OPTIONS SERROR NOSERROR MERROR

SYMBOLGEN MPRINT MLOGIC OBS=10 REPLACE COMPRESS=YES \_LAST\_ INVALIDDATA=’.’

PROC OPTIONS;

DATA BRADY;

INFILE HISTORY END=NOW;

INPUT YEAR MONTH DAY PLACE & $ EVENT & $;

IF NOW THEN DO;

END;

PROC PRINT DATA=BRADY;

DATA NAMES (DROP= BRADY AIDEN BRIAN IRENE);

RENAME = (OLD=NEW)

When using the RENAME and KEEP/DROP options together in a DATA step, the KEEP= or DROP= options get applied before the RENAME= option.

BETWEEN, NOT BETWEEN, AND CONTAINS, IS NULL, IS MISSING, LIKE… \_ represents any character and % represents any sequence

\_LAST\_

PROC SORT DATA=BRADY.NAMES;

BY ID;

RUN;

PROC PRINT DATA=BRADY.NAMES;

PROC PRINTTO LOG=LOG;

RUN;

PROC CALENDAR;

PROC DATASETS PROC CATALOG PROC CONTENTS

PROC COPY LIBRARY=

PROC APPEND DATA=OLD OUT=NEW;

RUN;

PROC TRANSPOSE DATA=

PROC MEANS PROC SUMMARY PROC UNIVARIATE = Determines quantiles

PROC TABULATE – prints tables of descriptive statistics from the variables of a SAS dataset

PROC CORR PROC FREQ PROC RANK

ABS, CEIL, FLOOR, FUZZ, INT, MOD, ROUND, TRUNC, MOD

MOD – returns the remainder. A=MOD(10,3) results =1 10/3=1 remainder

PROC MEANS provides MIN, MAX, STDDEV, SUM, KURTOSIS, etc

DATA() TODAY() TIME() MDY, DATEPART, TIMEPART, MONTH, DAY

INTNX does addition; INTCK does subtraction

LEFT, COMPRESS removes blanks and charaters, UPCASE, RIGHT, SUBSTRG

MORT function can calculate one parameters of a loan if 3 are known. A=borrowed amount, N= periodic payments, P = amount of payment, R= periodic interest rate

MORT(., p, r, n) SAVING( ) COMPOUND() INTRR() NETPV()

LAG and DIF functions –

DATA \_NULL\_;

DO I = 1 TO 5;

A = LAG1(I);

PUT I = A=;

END;

I= 1 A=.

I=2 A=1

I=3 A=2

I=4 A=3

I=5 A=4

DIF Function only works with Numeric variables and subtracts the previous value from current value.

CALL Routines do not return a value.

CALL RANBIN (SEED, N, P, X);

CALL SYMPUT (macrovariable name, string); CALL SYMPUT assigns a value to a macrovariable.

IF THEN ELSE

IF BALANCE < 0 THEN DEFICIT = -&BAL;

ELSE DEFICIT = .;

END;

IF BALANCE < 0 THEN DO;

DEFICIT = -&BAL;

BALANCE = 0;

END;

ELSE DEFICIT = 0;

STOP SHORT

IF \_N\_ > 250 THEN STOP;

ABORT ends the entire program or session.

DO LOOPS

A=1;

B=4;

DO I = A TO B;

PUT I= B=;

B = 2;

END;

DATA \_NULL\_;

X=1;

DO WHILE (X < 1000);

X = X + X;

END;

PUT X=;

DATA \_NULL\_;

X=1;

DO UNTIL (X>= 1000); X = X+X; END; PUT X=;

ARRAY CHSTATE{4};

ARRAY NSTATE{4};

DO I = 1 TO 4;

CHSTATE{1} = STNAMEL(NSTATE{I});

END;

LBOUND and HBOUND – lower and higher bounds for arrays

NESTED LOOP

DO I = LBOUND1(MEASURE) TO HBOUND1(MEASURE);

DO J = LBOUND2(MEASURE) TO HBOUND2(MEASURE);

AVG(I, J) = ROUND (MEASURE{I, J, 1} + MEASURE{I, J, 2} )/2);

END;

END;

SELECT BLOCKS

SELECT (SUBGROUP);

WHEN (‘NORTH’) FILE N;

WHEN (‘WEST’) FILE W;

WHEN (‘CENTRAL’) THEN DO;

IF STATUS = ‘A’ THEN FILE CA;

ELSE FILE CI;

END;

FILENAME fileref file;

A SAS Array is a variable list with a name.

Data \_NULL\_;

ARRAY S{5} FIRST SECOND THIRD FOURTH FIFTH;

S({2} = 2.22;

I = 3;

S{3} = 33333;

FIRST = 1; S{4} = FIRST; S{5.9} = S{1};

PUT \_ALL\_;

The same variable can appear more than once in a array, e.g.

ARRAY SCHEDULE {7} WEEKEND WEEKDAY WEEKDAY WEEKDAY WEEKDAY WEEKDAY WEEKEND;

Array Variations (ranges) ARRAY GNP{1969:1994}; GNP1 represents 1969, GNP2 = 1970, etc.

Implicitly Subscripted Array

RETAIN TEN 10 TWENTY 20 THIRTY 30 FORTY 40;

ARRAY TENS (\_I\_) TEN TWENTY THIRTY FORTY;

\_I\_ = 2;

E = TENS

\_I\_ = 4;

M = TENS;

In the above example, the variable E = 20 while M = 40.

If the value of an array expression is a noninteger, it is truncated to an integer value as if the INT function were present. If an array subscript is outside the range defined for an array, it is called out of bounds, and the SAS interpreter will issue an error message and stop processing the step.

ARRAY EASY {3} $ A B C;

I = 1;

EASY{I} = ‘7’;

SUBSCRIPT for use as a pseudo-variable.

EVENT = ‘Headline’;

SUBSTR(EVENT,1, 1) = ‘D’;

Result of EVENT is now ‘Deadline’

The concatenation operator (||) is the only operator that works exclusively with character values.

TRIM function can be used to remove trailing blanks from a string before concatenating.

TRIM(‘BROKE ‘) || ‘N’ = ‘BROKEN’

NE, ^=, NOT=, NOT EQ all represent not equal to.

A function is a routine that returns a value.

Character Functions Numeric Functions

COLLATE ABS LOG 10

COMPRESS CEIL MAX

LEFT EXP MEAN

PUT FLOOR MIN

REPEAT FUZZ MOD

REVERSE INDEX N

RIGHT INDEXC ROUND

SUBSTR INT SIGN

TRANSLATE LENGTH SQRT

UPCASE LOG SUM

Conversion Type Rules

The SAS interpreter converts***character***valuesto***numeric***values when:

* A character value is used with a numeric format in the PUT function.

The SAS interpreter converts ***numeric*** values to ***character*** values when:

* A numeric expression is used with a character format in the PUT function.
* A numeric expression is used as an argument of the INPUT function.

Converting a character value to a numeric value is done with the INPUT function and a numeric informat.

Converting a numeric value to a character value is done with the PUT function and a numeric format.

%EVAL function evaluates expressions involving macro operators. It is needed when using macro operators outside of macro statements, or in the places in macro statements where a macroexpression is not expected.

The commonly used macro quoting function is %STR, which can be used in the %LET statement to assign values to macrovariables that contain semicolons or other special characters.

For example, this statement assigns the characters PROC PRINT; to the macrovariable PROCESS.

%LET PROCESS = %STR(PROC PRINT;);

CALL SYMPUT is a CALL routine that assigns a value to a macrovariable.

DATA \_NULL\_;

CALL SYMPUT (‘WHERE’, ‘HERE’);

The above data step assigns the value HERE to the macrovariable WHERE, similar to: %LET WHERE = HERE;

CALL SYMPUT (‘SIDE’, TRIM(LEFT(PUT(A, 7.))));

The SYMGET function does the opposite, returning the value of the macrovariable.

S = SYMGET(‘SIDE’);

%LET B = BETA;

%PUT THE VALUE OF B IS &B..; This generates the following statement – “THE VALUE OF B IS BETA”

The %SYSEXEC statement executes an operating system command.

The difference in macrovariable and a macro is that a macrovariable is kept in the macro processor’s table of macrovariables in memory, while a macro is stored in a file.

MACRO PARAMETERS

%MACRO *name (parameter list;*

and the macro invocation changes to

*%name (parameter list);*

If the macro processor encounters two consecutive ampersands (&&), it changes them to a single ampersand (&). For example,

%LET I = 5;

%LET FILE5 = LPT1;

This macrovariable reference

&&FILE&I

Is interpreted as LPT1. On the first scan, the && is changed to &, and 5, the value of I, is substituted for &I. resulting in

&FILE5

The value of FILE5 is then substituted to get LPT1.

The MPRINT, SYMBOLGEN, and MLOGIC options affect the notes written in the log. The MERROR and SERROR options determine how unrecognized macro objects are treated.

Suppose you want to merge several SAS datasets, but the number of SAS datasets to merge might vary. Inside a macro, you could use a macro %DO loop like this one

MERGE %DO I = 1 %TO &NDAYS;

DAY&I (KEEP=DEPT NPAGE RENAME=(NPAGE=NPAGE&I)) %END;;

To generate a statement like this:

MERGE

DAY1 (KEEP=DEPT NPAGE RENAME=(NPAGE=NPAGE1))

DAY2 (KEEP=DEPT NPAGE RENAME=(NPAGE=NPAGE2))

DAY3 (KEEP=DEPT NPAGE RENAME=(NPAGE=NPAGE3))

PROC SORT DATA=*SAS dataset* OUT=*SAS dataset* options;

BY *key variables*;

NODUPKEY eliminates duplicates that have the same key variables. NODUPS is different as it eliminates rows of data where ALL data fields are the same.

DATA work.grouptable;

SET work.people;

BY GROUP NOTSORTED;

The NOTSORTED option is needed at the end of the BY statement if the observations are not in sorted order by the BY variables.

FIRST. and LAST. Variables create binary fields (0,1) for observations in a group.

Selecting one per group. The example below selects the values of the variable COLOR using this step:

DATA PALLETTE (KEEP=COLOR);L

SET MAX;

BY COLOR;

IF FIRST.COLOR;

To find the youngest person in each group, you could sort by GROUP and AGE, and then take the first observation in each group:

BY GROUP BIRTHDAT;

IF FIRST.GROUP;

DATA TALL (KEEP=NAME AGE MONTH);

ARRAY AGEX{10} AGE1-AGE10;

SET WIDE (KEEP=NAME AGE1 AGE2 AGE3 AGE4 AGE5 AGE6 AGE7 AGE8 AGE9 AGE10);

DO MONTH = 1 TO 10;

AGE = AGEX{MONTH};

OUPUT;

END;

COPYING – PROC COPY procedure cannot copy SAS datasets within the same library or change the name of the SAS dataset as it is copied.

PROC COPY IN=WORK OUT=WORK2 MTYPE=DATA;

SELECT TEST;

PROC APPEND DATA=FROM OUT=TO;

DELETING – SAS Datasets can be deleted by using the DATASETS proc.

This procedure deletes all SAS files from the WORK directory.

PROC DATASETS LIBARARY=work NOLIST KILL;

To delete specific data files use this:

PROC DATASETS LIBRARY=work MTYPE=DATA NOWARN;

DELETE work.test work.test2;

RENAMING SAS TABLES – USE CHANGE STATEMENT IN PROC DATASETS. The syntax below changes the name of the work.place table to work.space.

PROC DATASETS LIBRARY=WORK MTYPE=(DATA VIEW);

CHANGE PLACE=SPACE;

QUIT;

DESCRIBING – The CONTENTS procedure describes the contents of SAS datasets.

PROC CONTENTS DATA= *SAS Dataset;*

The names and other attributes of variables in a SAS dataset can be changed in the DATASETS proc.

PROC DATASETS LIBRARY=work;

MODIFY work.brady;

INFORMAT *variable…. Informat specification….;*

FORMAT  *variable…. Format specification….;*

LABEL *variable=’label’….;*

RENAME *old name=new name…;*

FILE FORMATS

Example of Delimited text to be read in:

1\Meister\Ferd\N.\\609\555-3729\23 Flight St.\\W. Brunswick\NJ\08012

2\Cutter\Deanne\\\609\555-4359\20713 Cove St.\#411\Orange\NJ\08012

The INPUT process for delimited fields is easier when an array of character variables is used to hold the character values of the fields. This example reads data in the form presented above:

DATA FOLKS (KEEP=SEQUENCE NAME\_L NAME\_F NAME\_M NAME AREACODE PHONE ADDRESS1 ADDRESS2 LOCALITY STATE ZIP);

ARRAY TOKEN{12} $40.;

INFILE FOLKS MISSOVER;

COLUMN = 1;

DO I = 1 TO 12;

INPUT @ COLUMN VALUE $CHAR2000. @;

FIELDMK = INDEX(VALUE, ‘\’);

TOKEN{I} = SUBSTR(VALUE, 1, (FILEDMK – 1) MAX ));

COLUMN + FIELDMK;

END;

The INDEX function returns the index of the first occurrence of the character sequence in the string. If the character sequence is not found, INDEX returns 0.

INDEXC is the index of the first occurrence in the string of any of the characters listed.

INDEX (string, character sequence)

INDEX(‘discovery’, ‘over’) – returns the value of 5

PROC TRANSPOSE – Turns observations into variables and variables into observations. The VAR statement lists the variables to be turned into observations. The ID statement identifies the variable whose value is used as the name of the output variable.

DATA PROD;

INPUT YEAR CORN TOMATOES WHEAT SOYBEANS;

CARDS;

1985 107 38.1 90 35

1986 99 37.0 98 34

1987 108 37.8 97 46

1988 110 38.6 96 42

1989 85 42.0 98 50

1990 102 45.5 106 51

;

PROC TRANSPOSE DATA=PROD OUT=TRANPROD NAME=CROP;

VAR CORN TOMATOES WHEAT SOYBEANS;

ID YEAR;

IDLABEL YEAR;

RUN;

DATA \_NULL\_;

SET TRANPROD;

PUT \_ALL\_;

PROC PRINT DATA=TRANPROD SPLIT=’\’;

ID CROP; LABLE CROP=’CROP’;

VAR \_1985 \_1986 \_1987 \_1988 \_1989 \_1990;

RUN;

**CROP 1985 1986 1987 1988 1989 1990**

CORN 107.0 99 108.0 109.0 85 102.0

TOMATOES 38.1 37 37.8 40.6 42 45.5

WHEAT 90.0 98 97.0 97.0 98 106.5

SOYBEANS 35.0 34 46.0 46.0 50 51.0

Before transposing, the SAS dataset had 6 observations and 5 variables. After transposing, it has 4 observations and 7 variables.

Use this syntax to print a table and to format variables.

PROC PRINT DATA=BUD;

VAR R X;

FORMAT R 7.2 X 6.2;

RUN;

Use LABEL statement to control labels.

PROC PRINT DATA= BUD LABLE;

VAR R X;

FORMAT R 7.2 X 6.2;

LABEL R=’REVENUE’ X=’EXPENSES’;

RUN;

ANALYTIC PROCS

PROC SUMMARY calculates statistics on the values of variables. Can calculate SUM, MEAN, MAX, MIN. The BY and CLASS statements can be used to generates statistics for subsets of the SAS dataset defined by the combinations.

The PROC TABULATE proc has a similar syntax and calculates the same numbers, but presents them in a different kind of table, which is defined by the TABLE statement.

PROC TABULATE;

CLASS R C;

VAR AMOUNT;

TABLE R;

C\*AMOUNT\*SUM;

The PROC FREQ reports absolute frequencies. The TABLES statement names one variable for a one-way frequency table, or two or more variables separated by asterisks for a table with variables crosstabulated.

PROC FREQ DATA=ACIDRAIN;

TABLES STATE;

PROC FREQ DATA=ACIDRAIN;

TABLES STATE\*MONTH;

PROC CONTENTS can be used to find the number of observations as well.

PROC CONTENTS DATA=FOLKS NOPRINT

OUT=NFOLKS (KEEP=NOBS RENAME=(NOBS=NFOLKS));

RUN;

The above procedure counts the number of observations in the SAS dataset FOLKS, putting the results in the variable NFOLKS in the SAS dataset NFOLKS.

SCAN function returns nth token from string; Example: SCAN(‘ Now is the time ‘,)) returns ‘Now’

The steps below create default values for the macrovariables DATE and PERIOD, and then assign a value for WEEKDAY to match the value of DATE.

%LET PERIOD = 1;

DATA \_NULL\_;

CALL SYMPUT(‘DATE’, PUT(TODAY(), DATE9.));

RUN;

DATA \_NULL\_;

INFILE PARM MISSOVER;

LENGTH VARIABLE $8.;

INPUT VARIABLE $ VALUE $78.;

CALL SYMPUT(VARIALBE, TRIM(VALUE));

RUN;

DATA \_NULL\_;

CALL SYMPUT(‘WEEKDAY’, SCAN(PUT(WEEKDAY(“&DATE”D), WEEKDATE9.), 1));

RUN;

CONDITIONAL %INCLUDE – Example that runs a program when its Monday. The program file containing those steps could be associated with the fileref MONDAY, and then would be executed or not using these statements.

%LET ACTION = ;

DATA \_NULL\_;

IF WEEKDAY(DATE() = 2 /\* If Today is Monday \*/

THEN CALL SYMPUT(‘ACTION’, ‘%INCLUDE MONDAY;’);

RUN;

&ACTION;

Use RANUNI function to generate a random number (uniformly distributed)

T = ROUND(RANUNI(99)

YEAR = INPUT(‘DATE’,4.);

CONVERT TO CHARACTER VALUE = PUT FUNCTION

YEAR(TODAY());

OPTIONS YEARCUTOFF=1921;

Convert a month name or abbreviation to a month number using a statement like:

NUMBER = INPUT(NAME, MON3.);

The INTCK and INTNX functions do time arithmetic for time values representing whole periods of time.

INTCK does subtraction, determining the length of time between two dates.

INTCK(‘YEAR’, 1995, 2000) This returns a value of 5

INTNX does addition.

INTNX(‘YEAR’, ‘15AUG1990’D,2) returns a value of ‘15AUG1992’D because 1992 is 2 years after 1990.

You can also use the QTR, MONTH, WEEK, DAY, HOUR, MINUTE, statements to return values in addition to YEAR

A string with all blanks removed.

COMPRESS(STRING)

A phone number with letters converted to digits.

TRANSLATE(STRING, ‘22233344455566677778889990’,

‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’)

Encryption

RETAIN ALPHABET ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’

ENCODING ‘IORBPMAFNLDCEUZHKYTGSWXQJV’;

CRYPT = TRANSLATE(STRING, ENCODING, ALPHABET);

Decryption is the opposite

RETAIN ALPHABET ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’

ENCODING ‘IORBPMAFNLDCEUZHKYTGSWXQJV’;

DECRYPT = TRANSLATE(CRYPT, ALPHABET, ENCODING);

PROC CONTENTS DATA=A;

The Hexadecimal format can fit a numeric code that is 8 digits long with a variable length of 4. E.g. zip code value of 00600

DATA \_NULL\_;

LENGTH ZIP $3.;

SET MAILLIST;

FILE PRINT;

IF ZIP < ‘006000’X THEN RETURN;

PUT /NAME / ADDRESS1;

IF ADDRESS2 NE ‘ ‘ THEN PUT ADDRESS2;

IF ADDRESS3 NE ‘ ‘ THEN PUT ADDRESS3;

PUT CITY +(-1) ‘, ‘ ST ZIP $HEX5.;

RETAIN ALPHABET ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’;

LETTER = SUBSTR(ALPHABET, 3, 1);

PUT LETTER=; prints ‘LETTER = C’

DO I = 1 TO 26;

LETTER = SUBSTR(ALPHABET, I, 1);

PUT I LETTER;

END;

The above statement produce output below:

1 A

2 B

3 C

4 D etc etc

To create a Boolean character variable that is one-byte, use the PUT function with the 1. Format.

ONTIME = PUT(DATE <= DEADLINE, 1.);

SELECT BLOCKS - For example, if the key variable is KEY and the lookup variable is LOOKUP.

SELECT(KEY);

WHEN (1) LOOKUP = ‘ONE’;

WHEN (2) LOOKUP = ‘TWO’;

WHEN (3) LOOKUP = ‘THREE’;

OTHERWISE LOOKUP = ‘LOTS;

END;

Looking up prices for products is a similar breakdown.

SELECT(CATALOG);

WHEN (‘102’) PRICE = ’46.00’;

WHEN (‘201’) PRICE = ’24.00’;

OTHERWISE;

END;

Array for finding the symbol for elements, atomic number 1 – 107.

ARRAY ELEMENTS{107} E1-107;

SYMBOL = ELEMENTS{NUMBER};

MacroArray

Finding the table of element symbols for all 107 is time consuming.

%LET T\_H = Hydrogen;

%LET T\_He = Helium;

% LET T\_Li = Lithium;

Instead of writing this 107 times, you can create the macrovariables in a data step.

DATA \_NULL\_;

LENGTH SYMBOL $ 3 NAME $ 16;

INPUT NUMBER SYMBOL $ NAME $ @@;

CALL SYMPUT (‘T\_’ || TRIM(SYMBOL), TRIM(NAME));

CARDS;

1 H Hydrogen 2 HE Helium 3 Li Lithium

;

After macrovariables have been created, you can use the SYMGET function to lookup the data.

NAME = SYMGET(‘T\_’ || TRIM(LEFT(SYMBOL)));

VALUE FORMAT

PROC FORMAT;

VALUE ELEMENTS (MIN=2)

1= ‘H’ 2= ‘He’ 3= ‘Li’;

RUN;

The format can then be used in a data step for table lookup.

SYMBOL = PUT(NUMBER, ELEMENTS3.);

VALUE INFORMAT

PROC FORMAT;

INVALUE NFIBER

‘COTTON’, ‘LINEN’, ‘RAMIE’, ‘ RAYON’, ‘SILK’, ‘WOOL’ = 1

OTHER = 0;

DATA NFIBERS SFIBERS;

SET FIBERS;

IF INPUT(FIBER, NFIBER.) THEN OUPUT NFIBERS;

ELSE OUTPUT SFIBERS;

The single most important issue in the efficiency of SAS programs, is the use of extra steps, with temporary SAS datasets passing data from one step to another.

DATA D;

LENGTH CONTN $ 2.;

INFILE TEXT;

INPUT CITY $CHAR16. CONTN $CHAR16. LONG LAT POP;

IF LAT > 0;

SELECT(CONTN);

WHEN (‘NO’) CONTN = ‘NA’;

WHEN (‘SO’) CONTN = ‘SA’;

OTHERWISE;

END;

POP = ROUND(POP/1000) MAX 1;

PROC SORT DATA =D;

BY CONTN;

PROC SORT procedure cannot be combined with a data step. Every proc requires an individual step.

DATA ABCDE;

MERGE D E A B C;

BY X;

PROC SORT DATA= BIG (KEEP= X XX XXX)

OUT= SMALL;

BY X;

DATA MAIN (KEEP=X A B C)

PART1 (KEEP=X A)

PART2 (KEEP=X B);

OUTPUT MAIN;

IF A > 0 THEN OUTPUT PART1;

IF B > 0 THEN OUTPUT PART2;

RUN;

DATA EXPMT

MAZETIME (KEEP= AGE TIME)

HUMOR (KEEP= AGE HUMORQ);

INFILE EXPMT END=LAST;

INPUT AGE PAY TIME HUMORQ;

IF TIME > 0 THEN OUTPUT MAZETIME;

IF HUMORQ > . THEN OUPUT HUMOR;

SUM + PAY;

FILE PRINT;

IF LAST THEN PUT

‘Maze Performance and Humor Experiments ‘;

RUN;

DATA \_NULL\_;

MERGE A B;

PUT \_ALL\_;

PROC SORT DATA=DUP OUT=SORTED NODUPKEY;

BY ID;

Numeric variables always occupy 8 bytes of memory, regardless of their length attribute. On the other hand, character variables should be declared with a length equal to the longest value possible to reduce memory space. Use the LENGTH or ATTRIB statement.

Reading values into code using the value informat:

PROC FORMAT;

INVALUE $SIGNAL (MIN=5)

‘Red’ = ‘01’X

‘Green’ = ‘02’X

‘Yellow’ = ‘03’X

‘Flashing red’ = ‘04’X

‘Flashing yellow’ = ‘05’X

‘Dark’ = ‘00’ X

;

And then write and display them using the corresponding value format:

PROC FORMAT;

VALUE $SIGNAL (MIN=5)

‘01’X = ‘Red’

Etc

;

For calculating descriptive statistics, the SUMMARY (or MEANS) proc is much faster than the UNIVARIATE proc. It is also more precise in its calculations. The UNIVARIATE proc is really only appropriate when you are calculating quantiles.

You should focus your efforts first on the long running steps, because that is where the potential for saving time is. Saving steps is the easiest way for speeding up SAS programs. After that, you should focus on compressing the data (declaring smaller lengths for character variables) and simplifying the program.

$CHAR format is the fastest for character variables. Use the KEEP= option to retain only variables that are to be used. In a proc summary or means, use the VAR statement to specify which variables should be processed.

PROC UNIVARS always creates a printed table unless specified by the NOPRINT option. Conversely, the PROC SUMMARY does not print a table unless specified by the PRINT option.

PROC MEANS DATA=SUMMARY;

VAR = WEIGHT;

RUN;

The DATA= option in the PROC statement identifies the data set to be summarized and the VAR statement lists one or more numeric variables to be analyzed.

**Selecting** **Statistics**

Generally we want more control over which statistics are to be selected. When you want to specifically select statistics, they are listed as options on the PROC statement.

Title1 ‘The First Two Statistical Moments’;

PROC MEANS DATA=WORK.CLASS

N MEAN VAR STD STDERR;

VAR WEIGHT;

RUN;

The list of available statistics is fairly comprehensive. A subset of which includes:

: n number of observations used to calculate the statistics

: nmiss number of observations with missing values

: min minimum value taken on by the data

: max maximum value taken on by the data

: range difference between the min and the max

: sum total of the data

: mean arithmetic mean

: std standard deviation

: stderr standard error

: var variance

: skewness symmetry of the data's distribution

: kurtosis peakedness of the data's distribution

A number of statistics having to do with percentiles and quantiles are also available, including:

: median 50th percentile

: p50 50th percentile (or second quartile)

: p25 | q1 25th percentile (or first quartile)

: p75 | q3 75th percentile (or third quartile)

: p1 p5 p10 other percentiles

: p90 p95 p99 other percentiles

PROC MEANS DATA=WORK.CLASS NOPRINT;

VAR WEIGHT;

OUTPUT OUT=SUMMARYDAT;

RUN;

SET A (FIRSTOBS=1000 OBS=1099);

\* Using DO Loop \*

DO I = 1 TO 5;

A{I} = X\*\*I;

END;

\* The same thing, but faster \*

A{1} = X;

A{2} = X\*\*2;

A{3} = X\*\*3;

A{4} = X\*\*4;

A{5} = X\*\*5;

Using the RETAIN and RENAME statements are efficient.

DATA C;

SET C (KEEP=CC RENAME=(CC=C));

RUN;

Dividing is slower than multiplication, so use this:

PERCENT = RATE\*.01; instead of PERCENT = RATE/100;

Sorting takes a long time. Use these techniques to improve efficiency.

\* Faster for small SAS datasets: fewer steps:

DATA COMBINED;

SET A B C;

PROC SORT DATA = COMBINED;

BY ID;

\*Faster for larger SAS datasets: smaller sorts:

PROC SORT DATA=A; BY ID;

PROC SORT DATA=B; BY ID;

PROC SORT DATA=C; BY ID;

DATA COMBINED;

SET A B C;

RUN;

Eliminate the use of MACRO processing by specifying the NOMACRO options. Eliminate the use of %INCLUDE statements when possible.

DATA WORK.TIME\_SERIES;

MERGE INSTRUMENT (IN=A) MACROECON (IN=B);

BY STATCOD;

IF A AND B;

RUN;

PROC SQL;

CREATE TABLE WORK.TIME\_SERIES AS SELECT

A.\*,

B.\*

FROM INSTRUMENT AS A INNER JOIN MACROECON AS B ON (A.STATCOD=B.STATCOD);

QUIT;

DATA INST\_TIME\_SERIES;

SET PRE\_TIME\_SERIES;

%DO I=1995 %TO 2016 $BY 1;

IF COHORT = &I THEN COHORT\_&I = 1; ELSE COHORT\_&I = 0;

%END;

IF COHORT\_2016 EQ . THEN DO;

COHORT\_2016=0;

END;

RUN;

DATA IN\_W\_RATES;

SET INS\_PRE\_DEF;

CON\_PREPAY\_RATE = (EXP(PRE\_PX)) / (1 + ((EXP(PRE\_PX))) + (EXP(DEF\_PX)));

CON\_DEF\_RATE = (EXP(DEF\_PX)) / (1 + (EPX(PRE\_PX))) + (EXP(DEF\_PX)));

PREPAY\_RATE = 1 – ((1 – (CON\_PREPAY\_RATE))) \*\* (1/12)));

CLAIM\_RATE = 1 – ((1-(CON\_DEF\_RATE))\*\*(1/12)));

RUN;

PROC LOGISTIC DATA=DEFAULT DES;

CLASS RES (PARAM=REF REF=’S’);

MODEL DEFAULT = LTV COHORT MTGRATE OPB;

RUN;

PROC LOGISTIC DATA = PREPAY;

CLASS COHORT (REF=’1’);

MODEL PREPAY (EVENT=’1’) = LTV HPI UNEMP OPB PROGRAM;

RUN;

VBAR Statements – Creates a vertical bar chart that summarizes the values of a category variable.

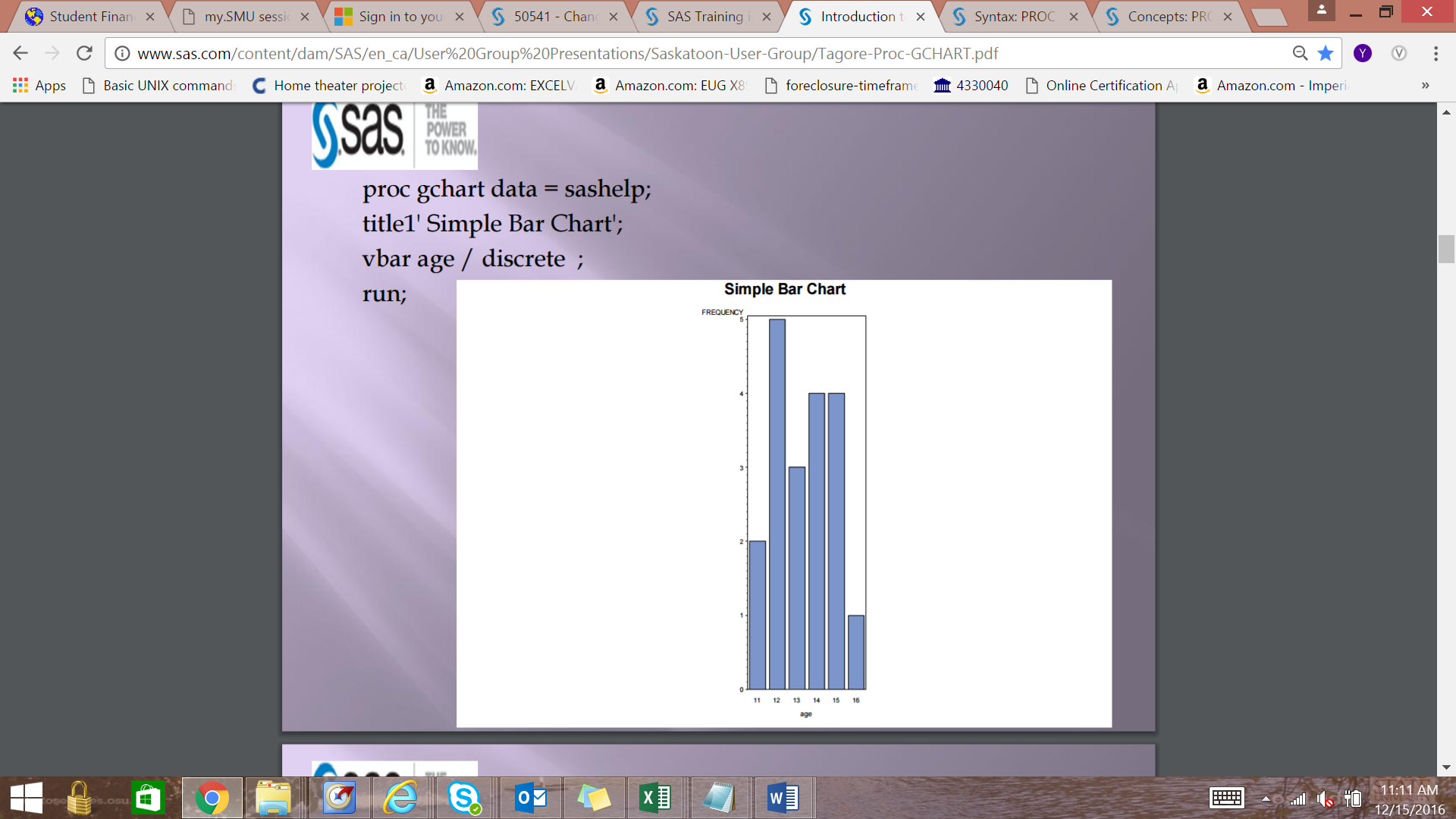
PROC GCHART DATA=SASHELP;

Title1 ‘A Simple Bar Chart’;

vbar age / discrete;

RUN;

The discrete variable creates a midpoint for each numeric value specified in the chart.



PROC GCHART DATA=SASHELP;

Title1 ‘Simple Bar Chart 2’;

VBAR Age / levels=4;

RUN;

MAINTAIN DISCRETE VALUES AND SPECIFY THE WIDTH AND THE PERCENTS ON INSIDE OF VERTICAL BAR CHART

PROC GCHART DATA=SASHELP;

Title1 ‘Simple Bar Chart 3’;

VBAR Age / DISCRETE WIDTH = 8 INSIDE = PERCENT;

RUN;

GROUP BY SEX

PROC GCHART DATA=SASHELP3;

TITLE1 ‘Simple Bar Chart by Sex’;

VBAR Age / DISCRETE WIDTH=6 INSIDE=PERCENT GROUP=SEX;

RUN;

ADD COLORS TO CHART

PATTERN1COLOR = ‘PINK’;

PATTERN2COLOR = ‘LIGHTBLUE’;

PROC GCHART DATA=SASHELP;

TITLE1 ‘ Simple Bar Chart by Sex’;

VBAR AGE / DISCRETE WIDTH = 8 INSIDE = PERCENT GROUP = SEX PATTERNID = GROUP;

RUN;

BLOCK CHARTS

Title1 ‘Total Sales’;

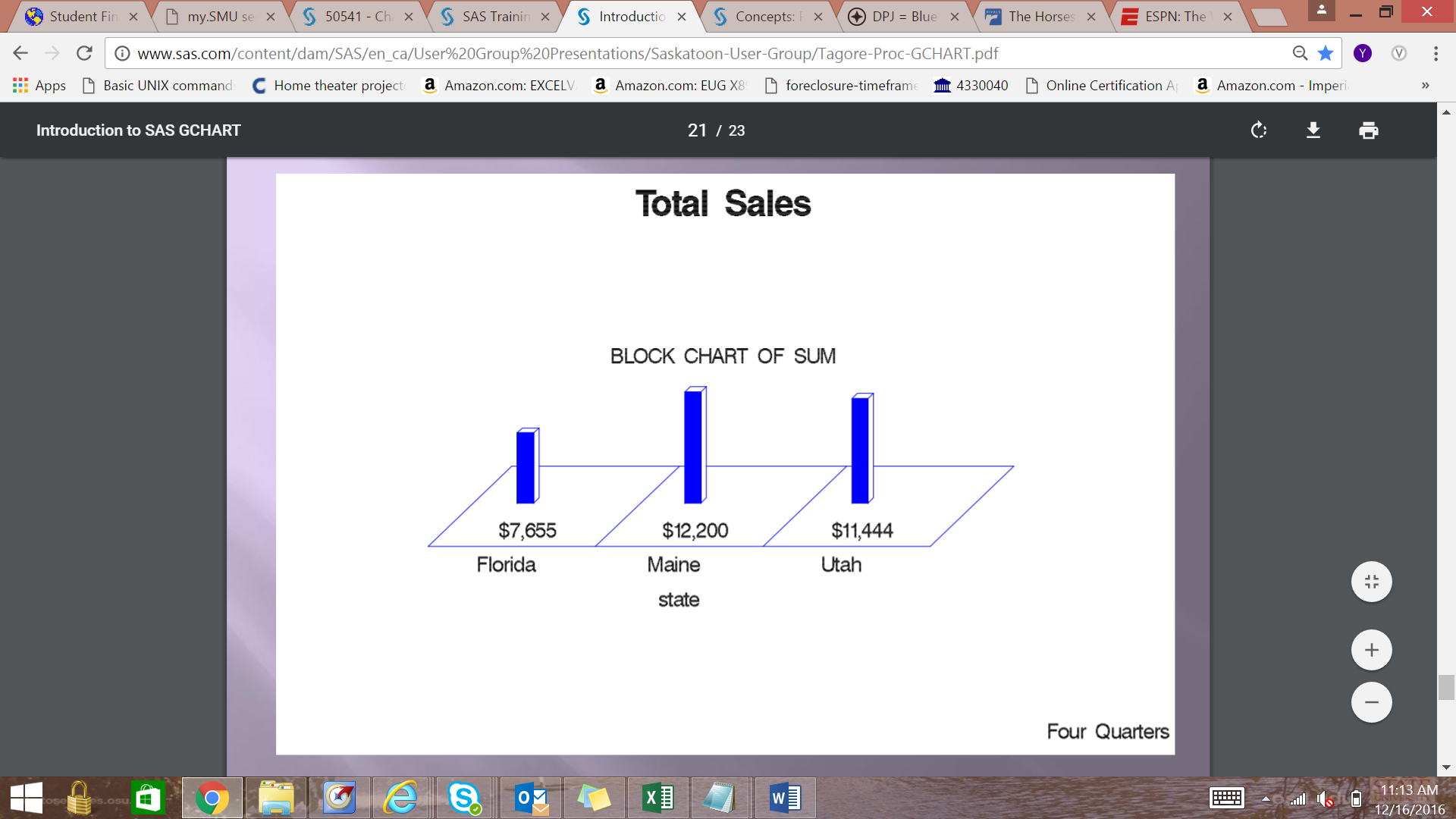
FOOTNOTE J = r ‘Four Quarters’;

PROC GCHART DATA=MYQUARTERS;

Format Sales Dollar8.;

Block State / sumvar = Sales;

Run;



PROC GCHART DATA=MYDATA;

Format Sales Dollar8.;

Pie State / sumvar=Sales Coutline=black;

RUN;

List of Different Charts Available:

1. HBAR – Horizontal Bar
2. VBAR – Vertical Bar
3. PIE – Pie Chart
4. BLOCK – Block Chart

@@ SYMBOL AT END OF INPUT STATEMENT EXPLAINED

@@

holds the input record for the execution of the next INPUT statement across iterations of the DATA step. This line-hold specifier is called **double trailing @**.

|  |  |
| --- | --- |
| **Restriction:** | The double trailing @ must be the last item in the INPUT statement. |
| **Tip:** | The double trailing @ is useful when each input line contains values for several observations, or when a record needs to be reread on the next iteration of the DATA step. |

PROC GPLOT EXAMPLE

GOPTIONS RESET=ALL BORDER;

PROC FORMAT;

VALUE MMM\_FMT

1='Jan'

2='Feb'

3='Mar'

4='Apr'

5='May'

6='Jun'

7='Jul'

8='Aug'

9='Sep'

10='Oct'

11='Nov'

12='Dec'

;

run;

DATA CITYTEMP;

INPUT MONTH FAREN CITY $ @@;

DATALINES;

1 40.5 Raleigh

1 52.1 Phoenix

1 12.2 Minn

2 16.5 Minn

2 42.2 Raleigh

2 55.1 Phoenix

;

title1 "Average Monthly Temperature";

footnote1 j=1 " Source: 1984 American Express";

footnote2 j=1 " Appointment Book";

symbol1 interpol=join value=dot;

PROC GPLOT DATA=CITYTEMP;

PLOT FAREN\*MONTH=CITY / HMINOR=0;

RUN;

SYMBOL1 INTERPOL=SPLINE WIDTH=2 VALUE=TRIANGLE C=STEELBLUE;

SYBMOL2 INTERPOL=SPLINE WIDTH=2 VALUE=CIRCLE C=INDIGO;

SYMBOL3 INTERPOL=SPLINE WIDTH=2 VALUE=SQUARE C=ORCHID;

AXIS1 LABEL=NONE

ORDER = 1 TO 12 BY 1

OFFSET=(2);

AXIS2 LABEL=("Degrees" justify=right "Fahrenheit")

ORDER=(0 TO 100 BY 10);

LEGEND1 LABEL=NONE VALUE=(TICK=1 "MINNEAPOLIS");

FORMAT MONTH MMM\_FMT.;

PLOT FAREN\*MONTH=CITY /

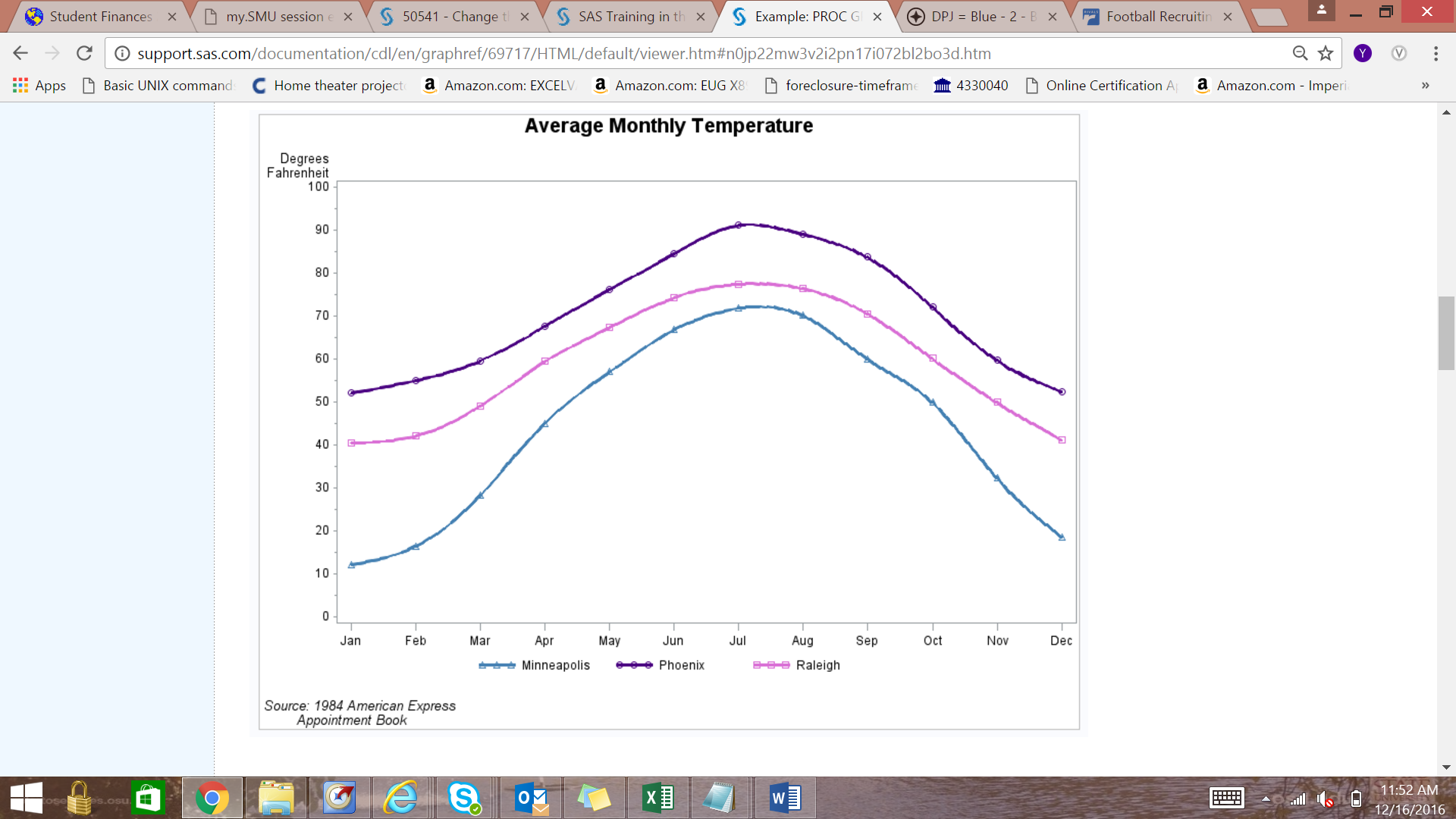
HAXIS=AXIS1 HMINOR=0

VAXIS=AXIS2 VMINOR=1

LEGEND=LEGEND1;

RUN;

QUIT;



PROC FREQ DATA=WORK.SASHELP;

TABLES SALARY\*MONTH / LIST;

RUN;

USE RANUNI TO GENERATE RANDOM SELECTED VARIABLES

DATA DISORDER;

SET ORDER;

RANDOM = RANUNI(8);

PROC SORT DATA=DISORDER OUT=CHAOS;

BY RANDOM;

DATA SAMPLE;

SET HEAVY;

IF RANUNI(2) < 0.1 THEN OUTPUT SAMPLE;

Quantiles are statistics that divide a distribution into equal parts. The median, for example, divides the distribution into two equal parts; half the values are below it, and half are above it. Other examples include quartiles (4 equal parts), deciles (10 equal parts), and percentiles (100 equal parts).

Example of how to break the data variable “SCORE” into deciles.

PROC SORT DATA=MEASURE (KEEP=SCORE) OUT=ORDER;

BY SCORE;

DATA \_NULL\_;

DO DECILE = 1 TO 9;

POINT = ROUND(NOBS + 1)\*DECILE\*.1);

SET ORDER POINT=POINT NOBS=NOBS;

PUT ‘Decile ‘ DECILE ‘is ‘ SCORE;

END;

PROC RANK DATA=MEASURE OUT=QUANTILE GROUPS=10;

VAR SCORE;

RANKS DECILE;

CONSIDER A COIN TOSS SIMULATION, FOR EXAMPLE:

IF RANUNI(5) < 0.5 THEN PUT ‘Heads’; ELSE PUT ‘Tails’;

A SERIES OF COIN TOSSES COULD BE SIMULATED USING A DO LOOP:

HEADS = 0;

DO I = 1 TO 100; HEADS + (RANUNI(5) < 0.5); END;

TAILS = 100 – HEADS;

PUT HEADS= TAILS=;

6 POSSIBLE RESULTS OF A DICE SIMULATED:

DIE = CEIL(RANUNI(5) \*6);

Finding the Appropriate Distribution of a Dataset – Proc Univariate or Proc Severity

Baseball Dataset

title 'Distribution of Plate Gaps';

ods graphics on;

proc univariate data=home.baseball;

var wins;

histogram / normal

lognormal

weibull

gamma

exponential;

run;

proc severity data=home.baseball plots=all;

loss wins;

dist weibull logn gamma exp ;

run;