Random Number Generators,DO Loops and An Introduction to Proc SQL

STAT 3505

Week 11 (March 28, 2024)

Gunes Fleming Ph.D.

Random Number Generators

- **RAND** function can be used to generate random numbers from various distributions.
  - Syntax: RAND(distribution, parameter1, .., parameterk)

Random Number Generators: **RAND** Function

Distribution	Function Call	Parameter Values
Bernoulli	`RAND('BERNoulli', prob)`	where (0 <= prob <= 1)
Beta	`RAND('BETA', alpha, beta)`	where (0.01 <= alpha <= 1.5e6) and (0.20 <= beta <= 1.5e6)
Binomial	`RAND('BINOmial', prob, trials)`	where (0 <= prob <= 1) and (0 <= trials)
Cauchy	`RAND('CAUChy')`	
Chi-Square(d)	`RAND('CHISquare', df)`	where 0.03 <= df <= 4e9
Erlang	`RAND('ERLAng', alpha)`	where 1 <= alpha <= 2e9
Exponential	`RAND('EXPOnential', scale )`	where 0 <= scale <= 1e300

SAS documentation

Random Number Generators

- **RAND** function can be used to generate random numbers from various distributions.
  - Syntax: RAND(distribution, parameter1, .., parameterk)

- RAND function does not have a parameter where a seed can be specified.
- **CALL STREAMINIT** statement can be used to assign a seed/ initialize the random number generation.
- Recall, the seed value determines the starting point for the sequence of random numbers generated.

Random Sampling

- Randomly selecting records from a large data set may be helpful if your data set is so large as to prevent or slow processing, or if one is conducting a survey and needs to select a random sample from some master database.

- When you select records randomly from a larger data set (or some master database), you can achieve the sampling in a few different ways, including:
  - Sampling Without Replacement
  - Sampling With Replacement
  - Selecting a Stratified Sample

Random Sampling

- **Sampling Without Replacement**: a subset of the observations are selected randomly, and once an observation is selected it cannot be selected again.
- **Sampling With Replacement**: a subset of observations are selected randomly, and an observation may be selected more than once.
- **Selecting a Stratified Sample**: a subset of observations are selected randomly from each group of the observations defined by the value of a stratifying variable, and once an observation is selected it cannot be selected again.

Approximate-Sized vs. Exact-Sized Samples

- One can indicate a certain percentage of the population to be chosen randomly, eliminating the need for a predetermined sample size.
- In this scenario, the samples obtained are known as **approximate-sized** samples since their size is not predetermined but rather determined by the specified percentage of the population.
  - For example, select approximately 25% of a data set

- A specific sample size can be predetermined and chosen from a population. Such samples are referred to as **exact-sized** samples.
  - For example, select 15 observations from a data set

Random Number Generators: Proc Surveyselect

```
PROC SURVEYSELECT data = data5 out = sample1B
  method = SRS
  seed = 1234
  samprate = 0.30;
RUN;
```

- **DATA**: tells SAS the name of the input data set from which observations should be selected.
- **OUT**: tells SAS the name of the output data set (sample1B) in which the selected observations should be stored.
- **METHOD**: Tells SAS the sampling method that should be used. For example, SRS tells SAS to use the simple random sampling method to select observations, that is, with equal probability and **without** replacement.
- **SAMPRATE**: tells SAS what proportion (0.30) of the input data set should be sampled.

SAS Documentation

Example: Exact-Sized Random Sample Selection **with Replacement** in a **Data Step**

```
DATA sample4A;
  choose=int(ranuni(5238978)*n)+1;
  set data5 point=choose nobs=n;
  i+1;
  if i > 15 then stop;
RUN;
```

- int(ranuni(5238978)*n)+1 expression always generates a positive integer up to n (the number of obs) in your data set.
- POINT option of the SET statement selects the *choose*-th observation from data5.

- The POINT= option tells SAS to read data5 using direct access by observation number.

  In general, with the POINT= option, you name a temporary variable whose value is the number of the observation you want the SET statement to read.

- Perform the above two steps repeatedly, keeping count of the number of observations selected. The expression i + 1 takes care of the counting for us: by default, SAS sets i to 0 on the first iteration of the DATA step, and then increases i by 1 for each subsequent iteration.

Select Stratified Random Sample using Proc Surveyselect

- **Selecting a Stratified Sample** : a subset of observations are selected randomly from each group of the observations defined by the value of a stratifying variable, and once an observation is selected it cannot be selected again.

```
PROC SURVEYSELECT data = data5
        out = sample6A
        method = SRS
        seed = 12345678
        sampsize = (5 5 5);
    strata city notsorted;
RUN;
```

- The STRATA statement tells SAS to partition the input data5 into non-overlapping groups defined by the variable *city*.
- The NOTSORTED option does not tell SAS that the data set is unsorted. Instead, the NOTSORTED option tells SAS that the observations in the data set are arranged in city groups, but the groups are not necessarily in alphabetical order.
- The SAMPSIZE statement tells SAS that we are interested in sampling five observations from each of the city groups.

DO LOOPS

- Commonly used to execute a block of code repetitively.

- Some of the do loops are unconditional. That is, they can run a code repeatedly. These DO loops are called iterative DO loops.

- Whereas some DO loops are conditional in that SAS needs to be told to execute a specific task *until* a particular condition is met or to do something *while* a particular condition is met.

  We call the former a DO UNTIL loop and the latter a DO WHILE loop

DO LOOPS

- Syntax: DO *index_variable* = *start_value* TO *end_value* BY *increment;*

  SAS statements

  END;

- In an iterative DO loop, keywords DO, TO, END and values *index_variable*, *start_value*, *end_value* are required.
- Any valid SAS variable name can be used for *index_variable.* Single letters like i, j, k, etc. are commonly used.
- Iterative DO loop starts at *start_value* and stops at *end_value.*
- Default increment, which is by how much SAS changes the *index_variable* after each iteration, is 1.
- If any increment other than 1 is desired, it needs to be specified in a BY clause.

DO LOOPS: DO UNTIL

- SAS executes the DO loop **UNTIL** the expression that is specified is true (instead of a fixed number of iterations like in the case of iterative DO loops).
- Syntax: DO UNTIL *(expression);*

  SAS statements

  END;
- SAS expression/condition specified can be any valid SAS expression enclosed in parentheses.
- This expression is not evaluated until the bottom of the loop. Therefore, a DO UNTIL loop always executes at least once. As soon as the expression is determined to be true, the DO loop does not execute again.

DO UNTIL Example:

- Calculate how many years it would take to accumulate $100,000 if you deposit $3000 into a savings account with 5.4% interest rate.

DO UNTIL Example:

- Calculate how many years it would take to accumulate $100,000 if you deposit $3000 each year into a savings account with 5.4% interest rate.

```
data investmentdata;
  do until(value >= 100000);
    value + 3000;
    value + value * 0.054;
    year + 1;
    output;
  end;
run;
```

DO LOOPS: DO WHILE

- SAS executes the DO loop **WHILE** the expression that is specified is true (instead of a fixed number of iterations like in the case of iterative DO loops).
- Syntax: DO UNTIL *(expression);*

  SAS statements

  END;
- SAS expression/condition specified can be any valid SAS expression enclosed in parentheses.
- An important difference between the DO UNTIL and DO WHILE statements is that the DO WHILE expression is evaluated at the top of the DO loop. If the expression is false the first time it is evaluated, then the DO loop doesn't even execute once.

DO WHILE Example:

- Calculate how many years it would take to accumulate $100,000 if you deposit $3000 each year into a savings account with 5.4% interest rate.

```
data investmentdata2;
value = 0;
  do while(value <= 100000);
    value + 3000;
    value + value * 0.054;
    year + 1;
    output;
  end;
run;
```

An Introduction to **PROC SQL**

Fox School of Business
TEMPLE UNIVERSITY®

An Introduction to **PROC SQL**

- PROC SQL is a powerful Base SAS Procedure that combines the functionality of DATA and PROC steps into a single step.
- PROC SQL can sort, summarize, subset, join (merge), and concatenate datasets, create new variables, and print the results or create a new table or view all in one step!
- PROC SQL can not only retrieve information without having to learn SAS syntax, but it can often do this with fewer and shorter statements than traditional SAS code.
- Additionally, SQL often uses fewer resources than conventional DATA and PROC steps. Further, the knowledge learned is transferable to other SQL packages.

An Introduction to **PROC SQL**

- In general, the Structured Query Language (SQL) is a standardized language used to retrieve and update data stored in relational tables (or databases).

- When coding in SQL, the user is not required to know the physical attributes of the table such as data location and type. SQL is non- procedural. The purpose is to allow the programmer to focus on what data should be selected and not how to select the data. The method of retrieval is determined by the SQL optimizer, not by the user.

- A table is a two-dimensional representation of data consisting of columns and rows. A table in SQL is simply another term for a SAS data set.

- Tables are logically related by values such as a key column.

An Introduction to **PROC SQL**

- Terminology:

Data Processing	SAS	SQL equivalent
File	SAS dataset	Table
Record	Observation	Row
Field	Variable	Column

- The table is where the data is stored. A row represents a particular entry. An employee may be represented as a row in a table. A column represents the particular values for all rows. Salary may be a column on a table. All employees will have a value for salary.

An Introduction to **PROC SQL**

- Simple Queries: A query is a request for information from a table or tables. The query result is typically a report but can also be another table.

- For instance: I would like to select last name, department, and salary from the employee table where the employee's salary is greater than 35,000.

How would this query (request) look in SQL?

```
SELECT LASTNAME, DEPARTMENT, SALARY
FROM CLASS.EMPLOY
WHERE SALARY GT 35000
```

An Introduction to **PROC SQL**

- Basic Proc SQL Syntax:

```
PROC SQL;
  SELECT  column, column . . .
  FROM  tablename|viewname. . .
```

- In Proc SQL,
  - Statements (clauses) in the SQL procedure are not separated by semicolons, the entire query is terminated with a semicolon.
  - Items in an SQL statement are separated by a comma.
  - There is a required order of statements in a query.
  - One SQL procedure can contain many queries and a query can reference the results from previous queries.
  - The SQL procedure can be terminated with a QUIT statement, RUN statements have no effect.

An Introduction to **PROC SQL**

- Select Statement:
  - To retrieve and display data a SELECT statement is used.
  - The data will be displayed in the order you list the columns in the SELECT statement.
  - A column can be a variable, calculated value, assigned value or formatted value.
  - An asterisk (*) can be used to select all columns.
- SELECT Syntax:

```
PROC SQL options;
  SELECT column(s)
   FROM table-name | view-name
   WHERE expression
   GROUP BY column(s)
   HAVING expression
   ORDER BY column(s);
```

An Introduction to **PROC SQL**

- *From* (Part of Select Statement): Specifies the input table(s).

- For example, to select social security number, salary and bonus (columns) for all employees (rows) from the employeedata table (data set):

```
PROC SQL;
        SELECT SSN, SALARY, BONUS
        FROM CLASS.EMPLOYEEDATA;
QUIT;
```