Lab #3: Probability, Simulations, Distributions:

A. Objectives:

1. Reading from an external file
2. Create contingency table
3. Simulate a probability distribution
4. The Uniform Distribution

Reading from external files:

As we learned in the last lab the **PROC IMPORT** statement is the best way to enter external data sets. The CSV file we will be using is called "Quest.csv". Download and save it in the desktop (easier to find) and mark the complete path to it. Then use the following code to import it, making sure you put the correct path on the **DATAFILE** argument.

```
PROC IMPORT OUT= Quest1
            DATAFILE= "put the complete path here…\Quest1.csv"
            DBMS=CSV REPLACE;
     GETNAMES=YES;
     DATAROW=2;
RUN;
```

The **IMPORT** statement reads the dataset and stores it as the value designated by "OUT" in this case it will be saved as "Quest1" in the library "Work".

Our dataset is now loaded in SAS. If we want to check it out a bit we need to click on the tab view and hit the "explorer" link. A folder-based SAS environment appears. Our data set is in the "WORK" folder and it is named "Quest1". Click on it and the table will appear in an excel-like format.

When you are done with manipulating the table click close from the file tab. If you don't do that the updates on the table will not be saved and SAS will stop working if you utilize that table in the next few lines of code.

As usual it makes sense to sort our data, let's say by gender. Let's go ahead and do that using the following command sequence:

```
PROC SORT data=Quest1;
BY gender;
RUN;
```

You may once again want to explore your dataset using the dropdown menu **VIEW** and navigating to **EXPLORER** and directory **WORK.** It is still saved as Quest1. Don't forget to click close from the file tab. If you don't do that the updates on the table will not be saved and SAS will stop working if you utilize that table in the next few lines of code.

Creating a "contingency table":

As we discussed in class an easy way to compare two categorical variables with a few possible outcomes is to use a contingency table. SAS is really good at creating those and the appropriate command is **PROC FREQ.** Try the following code:

```
Title 'Contingency Table 1';
PROC FREQ DATA=Quest1;
     TABLES local * gender/ norow nocol nocum nopercent;
RUN;
Title 'Contingency Table 2';
PROC FREQ DATA=Quest1;
     TABLES past_stats * past_programming/ norow nocol nocum nopercent;
RUN;
```

This code created 2 contingency tables, one comparing the variable local with the variable gender and another comparing the variable past_stats with past_programming.

The basic argument is **TABLE** which contains the variables you are comparing. The **arguments norow, nocol, nocum, nopercent;** suppress extra information, like the marginal probabilities and the relative frequencies. Remove them and re-run your code to see what the output looks like. Note also the number of missing values is given. The labels on the top left corner of the table explain what each of the numbers mean. For example Row Pct corresponds to the marginal probability with respect to rows and Col Pct the marginal probability with respect to columns.

Inputting a discrete probability distribution:

SAS has a great feature that allows the user to simulate any discrete probability distribution he or she desires. The idea is to provide SAS with an empirical probability and use that to build a dataset drawn from that probability with however many entries you want. Such a process is extremely useful when we require more data that "look like the ones we have" but we cannot collect more.

Let's start with a simple example, of a bag containing 2 Yellow, 2 Green, 2 Red and 4 Blue marbles. There are various ways to input that into SAS but we will just stick to the basic input as datalines as we learned in the very first Lab. Namely we will use the following code:

```
Title 'Random Bag';
DATA Bag;
INPUT colors $;
datalines;
Green
Green
Yellow
Yellow
Red
Red
Blue
Blue
Blue
Blue
;
RUN;
```

This should have a created a dataset named Bag, with the entries listed above. If you want to view that dataset again go to view, explorer, work, and click on Bag. If we want to see just the counts of the various outcomes we should use the PROC FREQ command as follows:

```
PROC FREQ data=Bag;
     TABLES colors/ norow nocol nocum;
RUN;
```

Once again, note the arguments **norow, nocol, nocum,** that suppress some information on the frequency table. Experiment with removing those.

By reading the relative frequency table (percentages) we get the following:

'Random Bag'
The FREQ Procedure

colors	Frequency	Percent
Blue	4	40.00
Green	2	20.00
Red	2	20.00
Yellow	2	20.00

Note that this is indeed a probability distribution since the percentages add up to 1.

What we would like to do now is simulate 200 repetitions of drawing a marble from the bag, observing its color and putting it back in. To do that we will ask SAS to create a specific probability distribution, with the numbers provided above.

The command for that is again the generation of a dataset, only this time the input is not given by the user. Instead it is decided by SAS based on the parameters we define. The following commands will do the trick:

```
Title 'Simulation of Bag';
DATA Simul(keep=Type);
CALL streaminit(1233);
ARRAY p[4] (0.2 0.2 0.2 0.4);
DO i = 1 to 200;
Type = RAND("Table", of p[*]);
output;
END;
RUN;
```

Again the argument **DATA** will give the name **Simul** to our dataset. And the variable of interest is denoted by the use of **KEEP** so for this example it is going to be "Type".

The argument **streaminit(1233)** is important. What it does is it generates the following table randomly, but saves the random process into a stream, so that if two people run this code with the same stream they will get the same answers, or if you re-run the code you will get the same answers. This allows for the TA to check your work, and for you to be able to replicate it exactly.

The argument **ARRAY** p[4] basically gives the probabilities for each outcome in a string (vector or array). The 4 is obviously to denote how many different outcomes there are. In this part of the code the outcomes don't have a name yet. They are defaulted by SAS to 1,2,3,4. We will change that in a bit.

So you insert the corresponding probabilities in any order, as long as you mark it down.

The line **DO i = 1 to 200**; is a basic loop that instructs SAS to choose from the distribution above 100 different entries numbered 1 through 200.

The next line say to define the Type variable (our simulated outcome) using a randomized selection. That is the command RAND which is a build in function is SAS that produces randomized results from the table of probabilities given (basically whatever is inside the parenthesis after RAND)

We close the code loop that started with the DO i =1 to 100 with the argument **END**; this tells SAS that it can stop repeating things and move to the next part of the code.

Let me remind you that you can always view the dataset you just created using the command **PROC PRINT.** The following code should do the trick:

```
PROC PRINT data=Simul;
TITLE "Unsorted Simulated Data";
RUN;
```

As we see the outcomes are not Blue, Green, Red, and Yellow but instead SAS uses 1,2,3,4. To fix that we need to set the format by specifying certain associations within the value CALL.

The following code does that for us:

```
PROC FORMAT;
VALUE Call 1='Green' 2='Yellow' 3='Red' 4='Blue';
RUN;
```

We append the associations **1**='Green' **2**='Yellow' **3**='Red' **4**='Blue' to a list named "Call", which we will prompt SAS to use every time it sees 1,2,3,4 as values in our dataset. For example the commands:

```
PROC PRINT data=Simul;
TITLE "Unsorted Simulated Data good labels";
FORMAT Type Call.;
RUN;
```

Will generate our 200 observations using the correct labels.

To finish off this problem, we want to compute the new probabilities coming from the simulated dataset. These are basically the proportions of each outcome vs the total number (200). This is the "frequentist" type of probability that we discussed in class (p_n). Once again, the limit of that as the number of observations (n) tends to infinity will match the probabilities we gave before. We again invoke the PROC FREQ command as follows:

```
PROC FREQ data=Simul;
TITLE "Simulated probabilities with good labels";
FORMAT Type Call.;
TABLES Type / nocum;
RUN;
```

The argument **nocum**, asks SAS not to output the cumulative probability, but keep everything else.

If you want more information about how to simulate data with SAS follow this link to an official SAS guide:

https://support.sas.com/resources/papers/proceedings15/SAS1387-2015.pdf

Analyzing the uniform distribution:

As we learned in the lecture, the simplest of the discrete probability distributions is the uniform one. For this distribution, all outcomes have the same probability, or in other words they are equally likely to occur. Let's create a uniform distribution in SAS and then simulate 30 draws from it. Let's assume we want to simulate an 8 sided die. Such a die can be created by gluing together two square bottomed pyramids, but simulating it in SAS is much easier.

All we need to do is define a table of probabilities as we did before. Obviously since all outcomes are equally likely and we have 8 of them the probability for each is 12.5% or 1/8. The following code does the trick:

```
Title 'Simulation of 8 Sided Dice';
DATA Dice(keep=Number);
CALL streaminit(1233);
ARRAY p[8] (0.125 0.125 0.125 0.125 0.125 0.125 0.125 0.125);
DO i = 1 to 30;
Number = RAND("Table", of p[*]);
output;
END;
RUN;
```

To see again the simulated probabilities we use the PROC FREQ command as follows. (Note this time we don't even need to change the labels so there is no need to invoke any formatting)

```
PROC FREQ data=Dice;
TITLE "Simulated probabilities of dice throws";
TABLES Number / nocum;
RUN;
```