SAS OnDemand for Academics

In this course we will use the statistical software SAS OnDemand for Academics for our analysis. In order to use SAS OnDemand for Academics for Independant Learners you need to create a SAS acount in the following link, SAS OnDemand for Academics You can use SAS studio to write and submit your SAS programs. SAS studio is the browser based interface. Any files or programs or data that you want to use need to be uploaded to server. To create a new folder to save your data, select the Files (Home) section, right click and create a new folder. To find the file path for this folder, you can right click on the folder and go to properties. In order to upload data from your local machine to the remote server select the folder you just created and go to upload files, choose files from your computer and upload them in the folder. For more information see, online Documentation

SAS Tutorial

SAS codes generally consist of either a `data` step to create or update a dataset, or a `proc` step to analyze your data. It is crucial to note that the end of a statement line is detected by `;`. Text comments can be made using `/* ... */`.

Data import and export

There are different ways to import data into the SAS environment. You can enter data manually by specifying the name of the dataset (e.g. `COLOR`) and the names of the variables (e.g. `REGION`, `EYES`, `HAIR`, and `COUNT`) after an `input` statement. Furthermore, you should specify if your variables are numerical (default) or character valued (use `$`). The operator `@@` allows you to enter multiple observations from a single record of raw data. Start your raw data after a `datalines;` statement and make sure you end this piece with a `;`. Below we present the code to create a dataset with 6 observations.

```
data COLOR;
    input REGION EYES$ HAIR$ COUNT@@;
    datalines;
    1  blue  fair   23 1 blue  red  7
    1  blue  medium 24 1 blue  dark 11
    1  green fair   19 1 green red  7
    ;
run;
```

If you want to export your data (e.g. to import the dataset using a different software package) you can make use of `proc export`. The output file type can be passed on to SAS via the `dbms=` command, for the possible types see the online documentation.

```
proc export data=COLOR
    outfile="/home/<username>/New Folder/color.csv"
    dbms=csv
    replace;
run;
```

Please run the code presented above to save the `COLOR` as a .csv file. We proceed by importing the dataset as `COLOR2` using `proc import`.

```
proc import out=WORK.COLOR2
    datafile="/home/<username>/New Folder/color.csv"
    dbms=csv
    replace;
    getnames = yes;
run;
```

We briefly discuss the different parts of the code:

- `WORK` leads to storing data in the WORK library.
- `"/home/<username>/New Folder/"` is the pathname to access the shared folder.
- `dbms` specifies the type of file to import.
- `replace` overwrites the dataset if a dataset with the same name already exists in the WORK library.
- `getnames` allows to use the column names in a file as variable names in SAS.

Since you will be continuisly using the folder "New Folder" it is efficient to store this location as `SASDATA` using a `libname` statement.

```
libname SASDATA "/home/<username>/New Folder";
```

During this course we will mainly work with *.sas7bdat* files. Use the code below to save the `COLOR` dataset at the file location `SASDATA` as *savedata.sas7bdat* (please check that this file is created in your "New Folder" folder).

```
data SASDATA.savedata;
    set COLOR;
run;
```

Importing a *.sas7bdat* file that is saved in "New Folder" is the most convenient way to import a dataset. Using the code below you can create the dataset `COLOR3` by reading in the *savedata.sas7bdat* file.

```
data COLOR3;
    set SASDATA.savedata;
run;
```

Basic procedures

Let us continue by analyzing and processing the `COLOR` dataset using basic procedures. One of the most useful procedures is `proc means`. This procedure can be used to obtain summary statistics like the mean, median, standard deviation, skewness, kurtosis, and percentiles. Run the code below to get these summary statistics for the `COUNT` variable from the `COLOR` dataset.

```
proc means data=COLOR mean median std kurt skew p10;
    var COUNT;
run;
```

SAS can also present these statistics per subgroup (e.g. per eye-color group), using the `by` statement.

```
proc means data=COLOR mean median std kurt skew p10;
    var COUNT;
    by EYES;
run;
```

Furthermore, the `output out =` statement can be used to save the results of the analysis in a new dataset named `OUTPUT`.

```
proc means data=COLOR mean median std kurt skew p10;
    var COUNT;
    by EYES;
    output out = WORK.OUTPUT;
run;
```

Summary statistics can also be reported using `proc univariate`. Using the code below we create a new dataset name `OUTDATA` containing the 10% percentile of the `COLOR` dataset. See the online documentation for all arguments that can be saved as output.

```
proc univariate data=COLOR;
    var COUNT;
    output out=OUTDATA pctlpts=10 pctlpre=percent;
run;
```

Furthermore, `proc univariate` can be used to draw histograms with fitted probability density curves (e.g. a normal density), or to draw quantile-quantile plots.

```
proc univariate data=COLOR;
    var COUNT;
    histogram COUNT / normal;
    qqplot    COUNT / normal;
RUN;
```

Subsets and merging

We will finish this tutorial by discussing subsets of your data as well as merging datasets. To create the subset `SUBDATA` we use a `data` step and start by setting the old dataset `COLOR`. The `where` statement can be used to put constrains on the old dataset. You can use the online documentation as a reference for all comparison operators.

```
data SUBDATA;
    set COLOR;
    where HAIR ^= 'red';
run;
```

Using the code above we created a subset with all observations that did not have red hair. Within this `data` step it is possible to process your data. Next to selecting the records that did not have red hair, we delete participants with a `COUNT` value above 30 as well as those with green eyes. Finally, we create a new variable `COUNTSQ` which equals the initial `COUNT` variable squared.

```
data SUBDATA;
    set COLOR;
    where HAIR ^= 'red';
    if COUNT > 30 or EYES = 'green' then delete;
    else COUNTSQ = COUNT**2;
run;
```

To illustrate how to combine two datasets, we start with creating two subsets `SUBSET1` and `SUBSET2` using the code below. The `keep` statement can be used to select only a subset of the variables, and thus deleting the other variables. The `drop` statement can be used similarly, but defines which columns of the dataset should be left out.

```
data SUBSET1;
    set COLOR;
    keep REGION COUNT;
run;

data SUBSET2;
    set COLOR;
    drop HAIR;
run;
```

In the case you would be working with two (comparable) datasets from different sources it is easy to paste one under the other by using the `set` statement.

```
data COMBINED;
    set SUBSET1 SUBSET2;
run;
```

Sometimes you are dealing with two (or more) datasets that all contain information about the same subjects. Let us create such an example using the code below. Try to understand why the `@@` is only used while creating the second dataset.

```
data CLASS;
    input NAME $ YEAR $  MAJOR $ ;
    datalines;
    Jennifer    first       .
    Tom         third       Theater
    Elissa      fourth      Math
    Rachel      first       Math
    ;
run;

data TIME;
    input Name$ DATE$  TIME ROOM@@;
    datalines;
    Jennifer    14sep2000   10  103
    Rachel      14sep2000   10  103
    Tom         14sep2000   11  207
    Elissa      15sep2000   10  105
    ;
run;
```

You can create a new merged (by the `NAME` variable) dataset `CL_TIME` using the code below.

```
data CL_TIME;
    merge CLASS TIME;
    by NAME;
run;
```

However, if you run this code SAS will return an error. While programming in SAS it can be crucial to sort you dataset in the order of the variables you are using. To do so one can rely on `proc sort`, which is also useful if you want to quickly see the maximum value of a variable. We will now sort the dataset `CLASS` and `TIME` by `NAME` since we want to merge based on the value of this later.

```
proc sort data=CLASS;
    by NAME;
run;

proc sort data=TIME;
    by NAME;
run;
```

Rerunning the merging code that previously failed now results in creating the combined dataset `CL_TIME`. While merging it is possible to process the sub datasets using `keep` and `drop` within brackets behind the names of the subsets. This way it is also possible to use the `rename` statement to change variable names.

```
data CL_TIME;
    merge CLASS (keep=NAME YEAR)
          TIME  (drop=DATE rename=(ROOM=CLASS_NUMBER));
    by NAME;
run;
```

Question 0

On Canvas you can find the datafile *iq.sas7bdat* dataset file, including the following variables:

- `LANG`: language test score.
- `IQ`: IQ score.
- `CLASS`: class id.
- `GS`: number of students in the class.
- `SES`: social-economic status of family.
- `COMB`: a binary variable indicating whether the students taught in a multi-grade class (`1`=yes and `0`=no).

Practice with the basic SAS commands by answering the questions below.

(a) Save the datafile in your "New Folder" and import the `IQ` dataset into SAS.

(b) Compute all the summary statistics for the variable `IQ` using `proc means`.

(c) Compute these summary statistics for the variable `IQ` in each `CLASS` variable adding

```
    by CLASS;
```

(d) Draw a quantile-quantile plot and a histogram for the variable `IQ` using `proc univariate`.