STAT 8678 - SAS Programming & Data Analysis

Chi-Kuang Yeh

2026-01-23

Table of contents

Preface

Description

This course covers programming using the SAS statistical software package, and it provides an introduction to data analysis stressing the implementation using SAS.

Topics include two main parts:

1) **SAS Programming**: data management and manipulation, basic procedures, macro programming;
2) **Data Analysis**: descriptive statistical analysis, one- and two-sample inference, basic categorical data analysis, regression analysis, and other selected topics.

Prerequisites

MATH 4544/6544 – Biostatistics, or equivalent.

Instructor

Chi-Kuang Yeh, Assistant Professor in the Department of Mathematics and Statistics, Georgia State University.

- Office: Suite 1407, 25 Park Place.
- Email: cyeh@gsu.edu.

Office Hour

10:00–13:00 on Monday, or by appointment.

Grade Distribution

- Assignments: 60%
- Exam: 20%
- Project: 20%

Assignment

☐ A1, due on Jan 28, 2026
☐ A2 TBA

Midterm

☐ March 4, 2026

Topics and Corresponding Lectures

Those chapters are based on the lecture notes. This part will be updated frequently.

Status	Topic	Lecture
	Welcome and Overview	1
	Basic SAS Operation	2
	SAS Syntax	3
	Import and Export Data	4
	Random Variable	5

Recommended Textbooks

- Statistics 480: Introduction to SAS, The Pennsylvania State University.
- SAS Training, SAS Institute.
- SAS Resources, University of California, Los Angeles.

Acknowledgments

Special thanks to Li-Hsiang Lin for providing the base materials given on this website.

Part I

Introduction

1 Introduction to Basic SAS Operation

Learning objective:

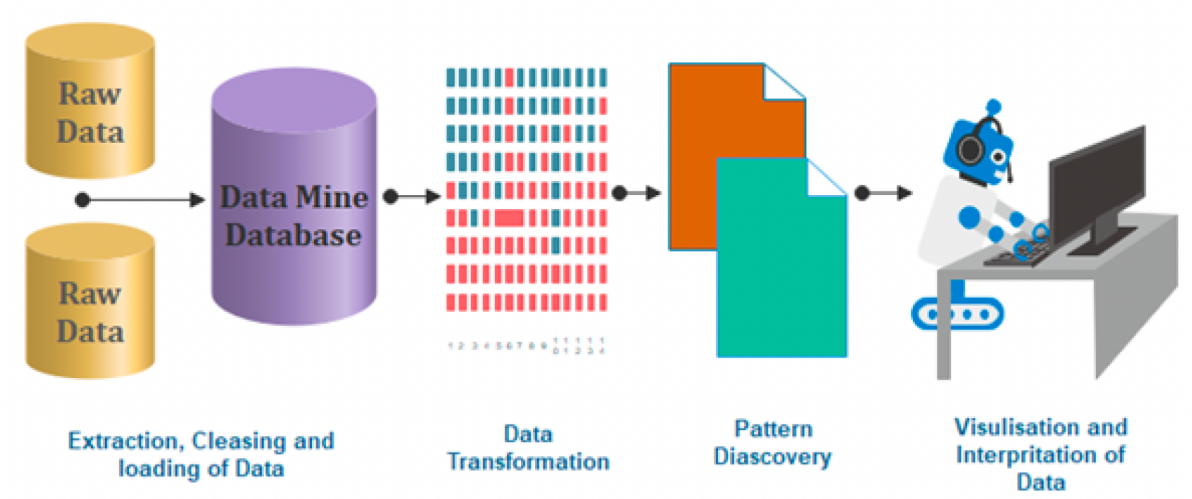
1. Familiarize ourselves with SAS windows (editor, log, output)
2. Create a dataset
3. Sorting Data (by 1 or more variables)
4. Obtain summary statistics of variables

1.1 Introduction to SAS

Q: What is SAS?

SAS (Statistical Analysis Software) is a prominent tool in the field of Data Analytics, offering a comprehensive suite for data manipulation, mining, management, and retrieval across various sources, coupled with robust statistical analysis capabilities. It excels in a range of functions including *data management, statistical analysis, report generation, business modelling, application development, and data warehousing.* SAS is user-friendly, featuring a point-and-click interface for those without technical expertise, while also providing deeper functionality through the SAS programming language. This software is instrumental in employing qualitative methods and processes that enhance employee productivity and business profitability.

Within SAS, data extraction and categorization into tables are pivotal for identifying and understanding data trends. This versatile suite supports advanced analytics, business intelligence, predictive analysis, and data management, facilitating effective operation in dynamic and compet- itive business environments. Additionally, SAS's platform-independent nature allows it to operate seamlessly across various operating systems, including Linux, Windows, Mac, and Ubuntu. SAS provides extensive support to programmatically transform and analyze data in the comparison of drag and drop interface of other Business Intelligence tools. It provides very fine control over data manipulation and analysis.



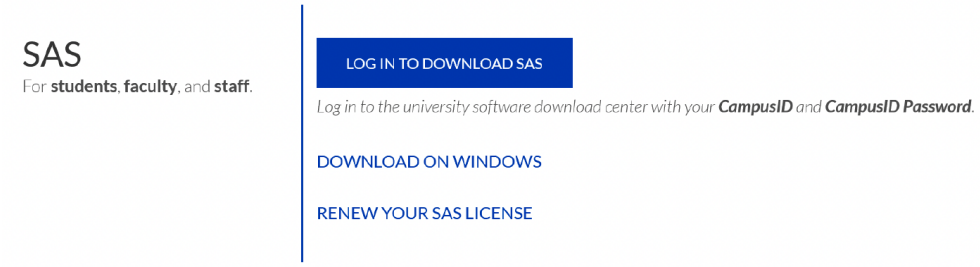
1.1.1 SAS Installzation

Georgia State University (GSU) has purchased license, so we can access SAS University Edition for free!

To install SAS University Edition, choose from the following options:

- **Option 1**:

  Download on your personal PC: Free SAS license available to GSU students, faculty, and staff via Technology Services (download required; check system requirements): Download from https://technology.gsu.edu/technology-services/software-equipment/university-licensed-software/ (Need to log-in from your GSU Account)



`Get Help for the Installation from <https://gsutech.service-now.com/sp>`

- **Option 2**:

- On Campus Access: SAS can be found on all GSU Library PCs: Floors 1-4 (not available on Library Macs, because there is no Mac version of SAS)
- Graduate Biostatistics Computer Lab (SPH): 6th floor of the Urban Life building (swipe card access required)
- Common MILE Lab whose opening time is
  - Monday & Wednesday: 9 – 18
  - Tuesday & Thursday: 9 – 17
  - Friday: 9 – 15

- **Option 3**:

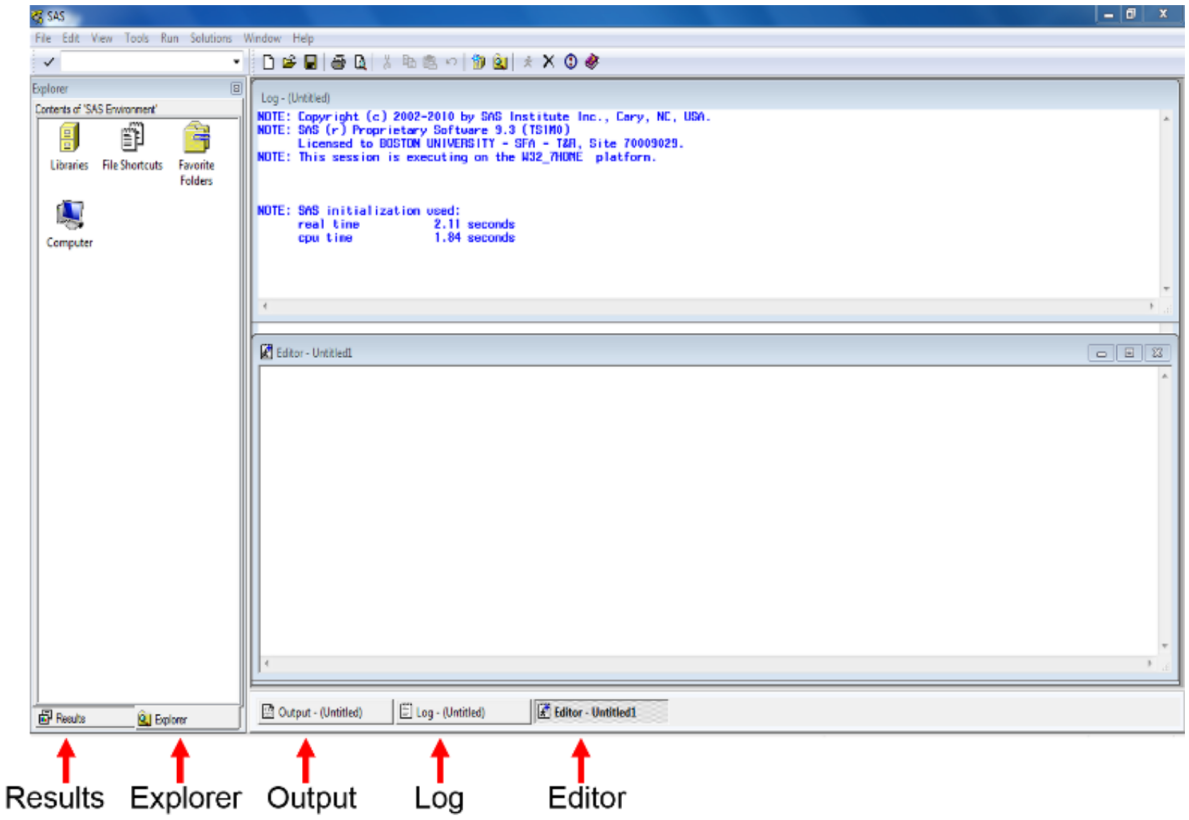
  Access via VLab, GSU's Remote Desktop Environment. Download and Connect to Cisco AnyConnect Client to connect to GSU's VPN (secureaccess.gsu.edu). Once connected to the VPN, login to VLab at: https://vlab.gsu.edu/ to access SAS.

- **Option 4**:

  Access via SAS OnDemand for Academics/SAS Studio. If you do not already have one, create a SAS profile at https://welcome.oda.sas.com/ Then, sign in with credentials and click SAS®Studio to access the web-based SAS environment.

1.1.2 SAS Windows

Once SAS has started, the screen will look similar to the following: The main SAS window is divided into several sub-windows:



- The menu and toolbar along the top of the window
- The **explorer/results browser** along the left hand side, where you can a listing of the results of successful SAS program.
- The **log** to the top right. This gives you information about possible errors after you have run your SAS program.
- The **program editor** below the log on the bottom right, where you create your SAS program.
- The windows bar along the bottom for you to switch all windows.

The **Editor (Program Editor)** window is a text editor that facilitates writing SAS programs (code). The Log window displays system messages, errors, and resource usage and is thus used to review program statements. The Output window displays output from statistical procedures run within the SAS program; however this is no longer the default. In SAS 9.3 output is sent to the Results Viewer which opens automatically when you run a procedure that generates output. The Results window displays a map of the Output window, and is useful for navigating the results of complicated analyses. Finally, the Explorer window contains all of the data sets in the current SAS session.

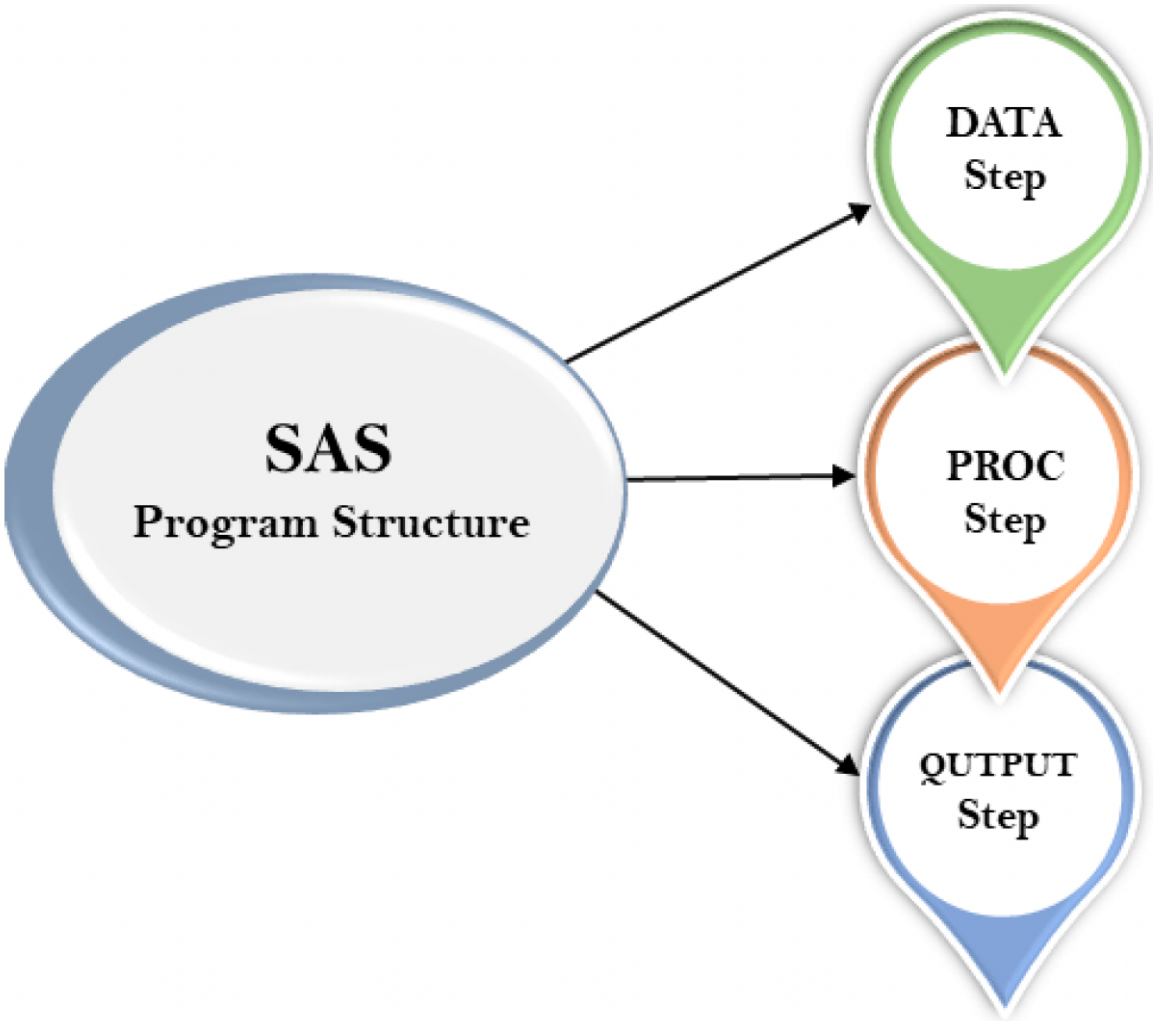
These windows can be moved or resized as desired. Only one SAS window is active at a time. The active window will have a shaded title bar at the top of the window, and a highlighted

windows bar at the bottom of the screen. In the above example, the Program Editor is the active window, with an "*Untitled*" program name. Note that the menu options for the SAS toolbar along the top of the screen depend on which window is currently active. (The active window can be changed by clicking on that window with the mouse, or by selecting the desired window from the Window menu.)

1.2 SAS Program

The programming structure of SAS consists of **three** significant steps:

- **DATA** step: create and modify a SAS data set for follow-up analysis
- **PROC** step: conduct data analysis
- **OUTPUT** step: show the analysis results




```
*Syntax of the SAS program:;
DATA dataset name; /* Name of the data set. */
INPUT var1,var2; /* Defines the variables in this data set. */
NEW_VAR; /* Creates a new variable. */
LABEL; /* Assign labels to variables. */
DATALINES; /* Enters the data. */
RUN;
```

1.3 SAS Dataset

SAS dataset is used to organize data values in a tabular form, i.e., in the form of rows for observations and columns for variables.

A SAS data set is a matrix whose each column is for each **variable** and whose each row for each **observation** (e.g., subject).

Data sets can be entered in the SAS programming code or can be read in from a variety of external sources, such as text files, csv files, and Microsoft Excel. In subsequent classes we will discuss reading in data sets from external files. Once a data set has been created, commands or procedures can operate on these data sets.

Other than these steps programming structure also includes data set, label, variables, values, and run.

1.4 SAS Examples

1.4.1 Creating a Dataset

Our first task in using SAS will be to create a small dataset and "print" that dataset to the output window. As we mentioned in previous paragraph SAS programs usually start with a DATA step where the dataset is created. Once the dataset is available, various procedures can be run on the dataset. The example below is written in the SAS Program window. The program creates a dataset called "People" with 3 variables (columns) which are 'gender', 'height', and 'weight' and 14 observations (rows). Note that the values of the variables on each line are separated by one or more blanks. A few other things that you should note:

- All SAS statements end with a semicolon (**;**)
- More than one SAS statement can be put on a line, or a SAS statement can continue across several lines, if every statement *ends with a semicolon.*
- Data listed as part of the program is also terminated with a semicolon. Data does not have to be entered in the program; it can also be read from files that are external to the SAS program (more on that next week)
- *gender* is a character variables as indicated by the $, and height and weight are numeric variables.
- Once the dataset is created, various SAS procedures (called **PROC**s) can be used to analyze the data and present results. We will start with a listing of the data created with a procedure called **PROC PRINT.**

```
title1 'STAT 8678 Example 1';
title2 'Your name';
```

```
DATA people;
INPUT gender $ height weight;

DATALINES;
m 63 125
m 76 195
f 62 109
m 75 186
f 67 115
f 60 120
m 75 205
m 71 185
m 63 140
f 59 135
f 65 125
m 68 167
m 72 220
f 66 155
;

PROC PRINT DATA=people;
RUN;
```

The LOG window gives information on the execution of the program. If your program did not execute properly you should examine the log for error messages that may explain the failure. The program would then be modified if necessary and rerun.

SAS creates a new window called the **Results Viewer** when the program is executed and produces output. This window is in HTML format and a new tab for the window is created below the left-hand windows.

1.4.2 Sorting Data

The data can be sorted (in our case by *gender*) using **PROC SORT** by adding the following lines to the program. We can then go ahead and print our new dataset sorted by gender with the proc print step.

> **Reminder**
>
> Any procedural step we do must begin with PROC and every line must end with a semicolon and the command run;

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

PROC PRINT data=people;
TITLE3 "Raw data sorted only by gender";
RUN;
```

> **Note**
>
> Any time we use PROC SORT our original dataset is sorted. SAS does not create a copy then sort!

1.5 Generating Summary Statistics

The last procedure to be executed in this exercise is **PROC UNIVARIATE**. This procedure will allow us to see summary statistics for any *quantitative variable*. The output from PROC UNIVARIATE will be important for the early part of this statistical methods class. It will provide measures of central tendency (mean, median, mode) and measures of dispersion (variance, standard deviation, range) as well as other basic statistics.

```
PROC UNIVARIATE DATA=people PLOT;
  BY gender;
  TITLE3 "Univariate procedure output done separately by gender";
  TITLE4 "The analysis was done for two quantitative variables";
  VAR HEIGHT WEIGHT;
RUN;
```

The code below applies the procedure only to the variable gender. It can be run on any quantitative variable and it could be run on several variables at the same time by listing several variables in the **VAR** statement, which would provide a separate analysis for each variable.

The results for PROC UNIVARIATE will be listed in the results viewer.

1.6 Other notes

Other Note

- SAS does not distinguish between upper-case letters or lower-case letters in the program, either can be used. However, it does distinguish between upper and lower case in datasets, so the character strings "Carol", "carol" and "CAROL" would be considered different values of the variable "name" in the program above.
- Comments: Additionally, you may add comments anywhere in your program either by beginning the statement with an asterisk (*) and ending it with a semicolon (;) or by beginning with /* and ending with */. These comments may be thought of as marginal notes, and will show in the program editor and log, but not in the output window.

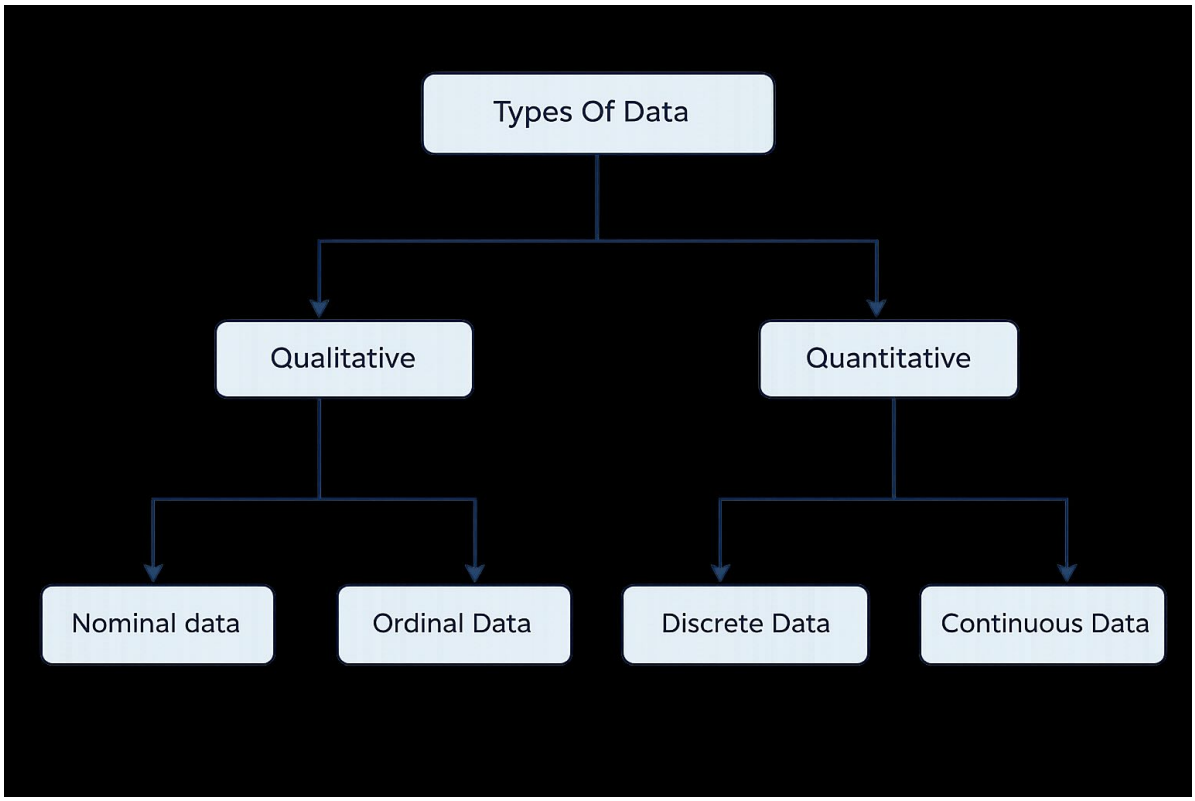
Shortcuts:

- F3 or the "running man": submits/run your program;
- F4: recalls text once the program editor;
- F5: directs user to the program editor;
- F6: directs user to the log;
- F7: directs user to the output;
- Ctrl+E clears content in the current window.
- Also, text can be copied with Ctrl+C, cut with Ctrl+X, or pasted with Ctrl+V.

1.7 Data Type

We can classify variables into *quantitative* variables and *qualitative* variables:

- Qualitative variables yield non-numerical information. Qualitative variables are often referred to as categorical variables, such as blood type. Qualitative variables can be further classified as
  - A nominal variable is a qualitative variable where no ordering is possible or implied in the levels, such as gender.
  - A ordinal variable is a qualitative variable with an order implied in the levels, such as health (poor, reasonable, good, or excellent)
- Quantitative variables yield numerical measurements. Quantitative variables can be further classified as discrete or continuous.
  - A discrete variable can assume only a countable number of values, such as headache severity scores.
  - A continuous variable is one that can take any one of an uncountable number of values in an interval, such as weight.



Question:

What is the type of each variable in the following dataset?

- AGE: The respondent's age in years
- GENDER: The respondent's sex coded 1 for male and 2 for female
- HAPPY: The respondent's general happiness, coded:1 for "Not too happy"2 for "Pretty happy"3 for "Very happy"
- TVHOURS: The average number of hours the respondent watched TV during a day

Table 1.1: Survey Respondent Summary

Respondent	AGE	Gen	HAPPY	TVHOURS
1	41	1	2	0
2	25	2	1	0
3	43	1	2	4
4	38	1	2	2
5	53	2	3	2
6	43	2	2	6

7	56	2	2	2

Answer:

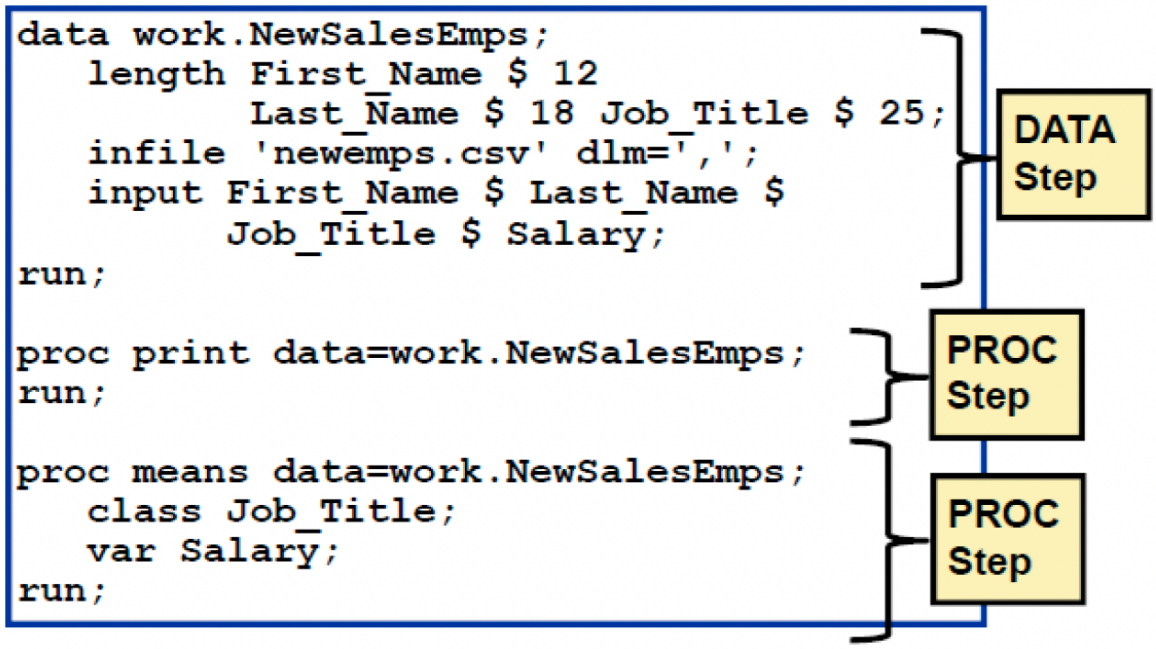
- Age: Continuous variable;
- SEX: qualitative;
- HAPPY: Discrete variable;
- TVHOURS: Continuous variable

2 Working with SAS Syntax

Learning objective:

1. Identify the characteristics of SAS statements.
2. Explain SAS syntax rules.
3. Insert SAS comments using two methods.
4. Identify SAS syntax errors.
5. Diagnose and correct a program with errors.
6. Save the corrected program

In general, writing a SAS program is to write a sequence of steps, and a step is a sequence of SAS statements:



How many statements are in the following step?

```
data work.NewSalesEmps;
   length First_Name $ 12
          Last_Name $ 18 Job_Title $ 25;
   infile 'newemps.csv' dlm=',';
   input First_Name $ Last_Name $
         Job_Title $ Salary;
run;
```

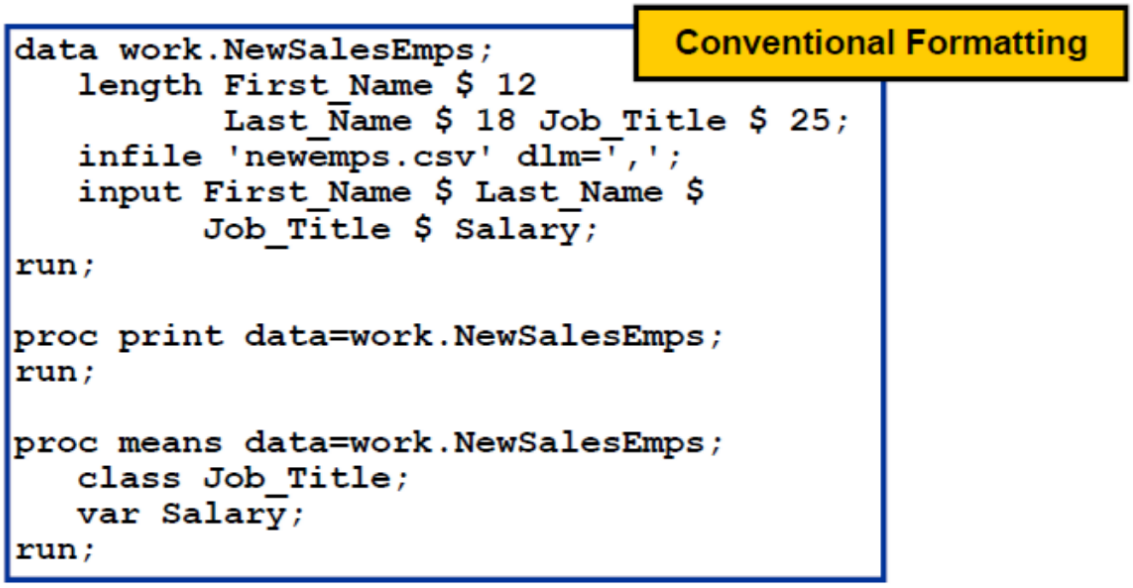
2.1 SAS Statments

SAS statements have these characteristics (Check the highlight context in the following examples):

- Usually begin with an identifying keyword.
- Always end with a semicolon.

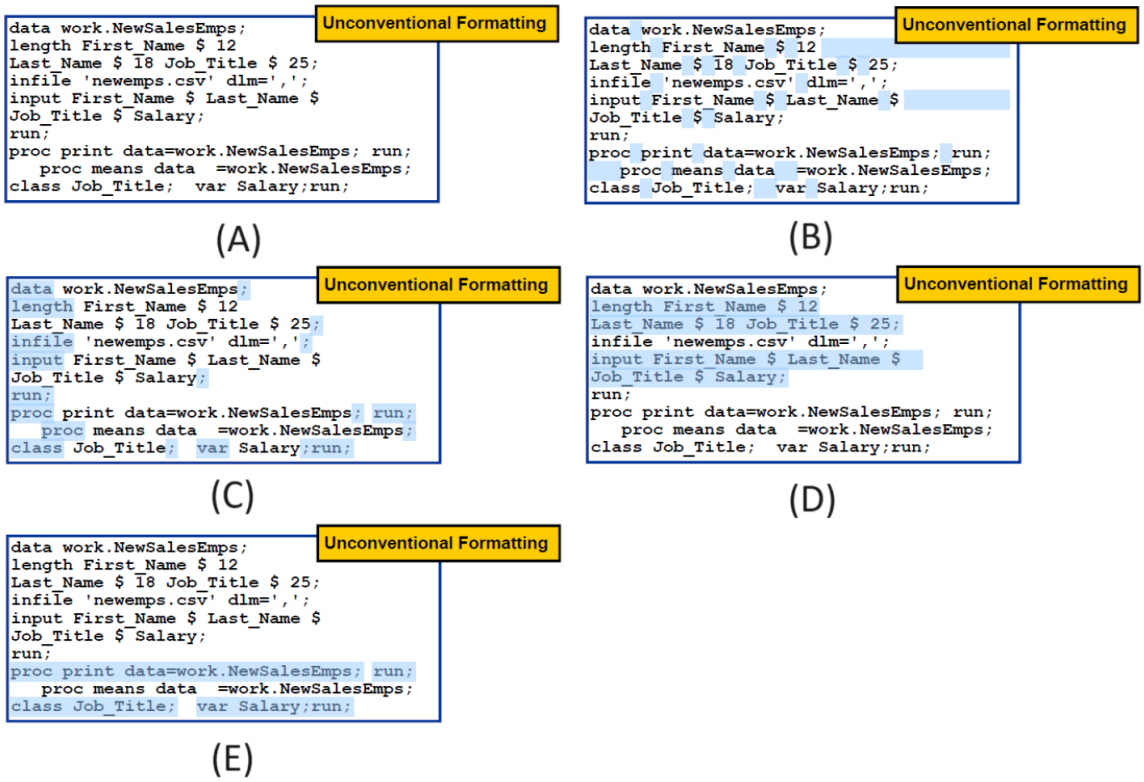
```
data work.NewSalesEmps;
   length First_Name $ 12
          Last_Name $ 18 Job_Title $ 25;
   infile 'newemps.csv' dlm=',';
   input First_Name $ Last_Name $
         Job_Title $ Salary;
run;

proc print data=work.NewSalesEmps;
run;

proc means data=work.NewSalesEmps;
   class Job_Title;
   var Salary;
run;
```

SAS programming statements are easier to read if you begin DATA, PROC, and RUN statements in column one and indent the other statements. This makes it more structured. Also, consistent spacing also makes a SAS program easier to read.



Overall, we can type SAS statements in the following ways:

- One or more blanks can be used to separate words.
- They can begin and end in any column.
- A single statement can span multiple lines.
- Several statements can be on the same line.



Sometimes we may want to make comments/notes to easy remember our SAS statements or to let other people easily understand what our code want to say. These comments are text that SAS ignores during processing. You can use comments anywhere in a SAS program to document the purpose of the program, explain segments of the program, or mark SAS code as non-executing text.

There are two ways to insert comments in a SAS program:

1. Using an asterisk (*) to begin the comment and a semicolon (;) to end the comment.

```
* This is a comment in SAS;
```

2. Using slash-asterisk (/*) to begin the comment and asterisk-slash (*/) to end the comment.

```
/* This is a comment in SAS */
```

Avoid placing the /* comment symbols in columns 1 and 2. On some operating environments, SAS might interpret these symbols as a request to end the SAS job or session. An example is given below:

```
*----------------------------------------*
|   This program creates and uses the    |
|   data set called work.NewSalesEmps.   |
*----------------------------------------*;
data work.NewSalesEmps;
   length First_Name $ 12 Last_Name $ 18
          Job_Title $ 25;
   infile 'newemps.csv' dlm=',';
   input First_Name $ Last_Name $
         Job_Title $ Salary /*numeric*/;
run;
/*
proc print data=work.NewSalesEmps;
run;
*/
proc means data=work.NewSalesEmps;
   *class Job_Title;
   var Salary;
run;
```

2.2 Diagnosing and Correcting Syntax Errors

Syntax errors occur when program statements do not conform to the rules of the SAS language.

Examples of syntax errors:

- misspelled keywords
- unmatched quotation marks
- missing semicolons
- invalid options

When SAS encounters a syntax error, SAS prints a warning or an error message to the log; for example,

```
ERROR 22-322: Syntax error, expecting one of the following:
              a name, a quoted string, (, /, ;, _DATA_, _LAST_,
              _NULL_.
```

When SAS encounters a syntax error, SAS underlines the error and the following information is written to the SAS log:

- the word **ERROR** or **WARNING**
- the location of the error + an explanation of the error

This program has three syntax errors. What are the errors?

```
daat work.NewSalesEmps;
   length First_Name $ 12
          Last_Name $ 18 Job_Title $ 25;
   infile 'newemps.csv' dlm=',';
   input First_Name $ Last_Name $
         Job_Title $ Salary;
run;

proc print data=work.NewSalesEmps
run;

proc means data=work.NewSalesEmps average max;
   class Job_Title;
   var Salary;
run;
```

2.3 Solution to the example

1. Exercise 1: 5
2. Exercise 2:

```
daat work.NewSalesEmps;
   length First_Name $ 12
          Last_Name $ 18 Job_Title $ 25;
   infile 'newemps.csv' dlm=',';
   input First_Name $ Last_Name $
         Job_Title $ Salary;
run;

proc print data=work.NewSalesEmps
run;

proc means data=work.NewSalesEmps average max;
   class Job_Title;
   var Salary;
run;
```

3 Import and Export Dataset

Learning objective:

1. Import a csv file into SAS
2. Export a csv file from SAS after data process

One of the strengths of SAS as a data analysis tool is its ability to read data from many sources, subset or combine data sets, and modify the datasets to accomplish various tasks. The most common types of external data sets used in SAS are EXCEL files (XLS extent), comma separated value files (CSV extent) and various space separate text files (PRN or TXT extent). A CSV file is actually a text file and can be read in any text reader (NOTEPAD or WORDPAD in Windows). In fact, the SAS files themselves, as well as the LOG and the LST files produced by a SAS by a batch submit, are also simple text files.

Format	Description	File Extension
WK1	Lotus 1 spreadsheet	.WK1
WK3	Lotus 3 spreadsheet	.WK3
WK4	Lotus 4 spreadsheet	.WK4
EXCEL	Excel Version 4 or 5 spreadsheet	.XLS
EXCEL4	Excel Version 4 spreadsheet	.XLS
EXCEL5	Excel Version 5 spreadsheet	.XLS
EXCEL97	Excel 97 spreadsheet	.XLS
DLM	delimited file (default delimiter is a blank)	.*
TAB	delimited file (tab-delimited values)	.TXT
CSV	delimited file (comma-separated values)	.CSV

3.1 Reading from External Files

The **PROC IMPORT** statement is the best way to enter external data sets. The CSV file we will be using is called "grades.csv". Download and save it in your favourite folder 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= GRADES_temp
DATAFILE= "Put Your Path Here/grades_temp.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 "Grades" in the library "Work".

The **DBMS** statement defines the type of input SAS should be reading. The following table gives you all the possible choices. The REPLACE argument forces SAS to overwrite any older datasets with the same name.

The **GETNAMES** = YES or NO statement for spreadsheets and delimited external files, determines whether to generate SAS variable names from the column names in the input file's first row of data. If you specify **GETNAMES** = NO or if the column names are not valid SAS names, PROC IMPORT uses the variable names VAR0, VAR1, VAR2, and so on. You may replace the equals sign with a blank.

The **DATAROW** argument tells SAS where to start reading for input data. In our case it is row 2 since row 1 is used for variable names.

We use the print procedure to see the dataset,

```
PROC PRINT DATA=GRADES_temp;
RUN
```

The first few rows are shown as follows

The SAS System

Obs	Student	Quiz1	Quiz2	Quiz3	Quiz4	Quiz5	Quiz6	Midterm	EC	Gender
1	.	7	7	7	7	7	7	25	2	M
2	1	4.55	6.8	6	5.1	5.75	6.7	13	1	M
3	2	5	6.1	6.7	5.4	7	3.8	21.5	1	M
4	3	3.75	6.5	6.9	2.1	6.6	5.5	21	2	M
5	4	5.05	6.2	7	5.9	6.75	6	24.5	1	M
6	5	5.35	6.8	6.6	5.1	6.6	6.5	24	1	M
7	6	7	7	6.7	6.1	7	7	25	1	F

3.2 Export a File from SAS

After reading a dataset into SAS, we may need to conduct some initial step/analysis to re-organize dataset for doing further data analysis. In the *grade* example, the first row shows the maximum points available for each quiz. We need to remove this row so that our analysis is correct. This can be done by the following SAS code

```
DATA GRADES;
  SET GRADES_temp NOBS=COUNT
    IF _n_         <= 1 THEN DELETE;
RUN;
```

Note: GRADES_temp is the name of the old dataset and *GRADES* is the name of the new dataset after the first row is deleted.

After re-organize the dataset, we may want to export the updated dataset for use in the future:

This can be done by

```
PROC EXPORT DATA= GRADES
  OUTFILE= "Put Your Path Here/grade_v2.csv"
  DBMS=CSV REPLACE;
  REPLACE;
RUN;
```

We will talk about more reorganizing skills in SAS in the next topic.

Which statement is true concerning the DATALINES statement based on reading the comment?

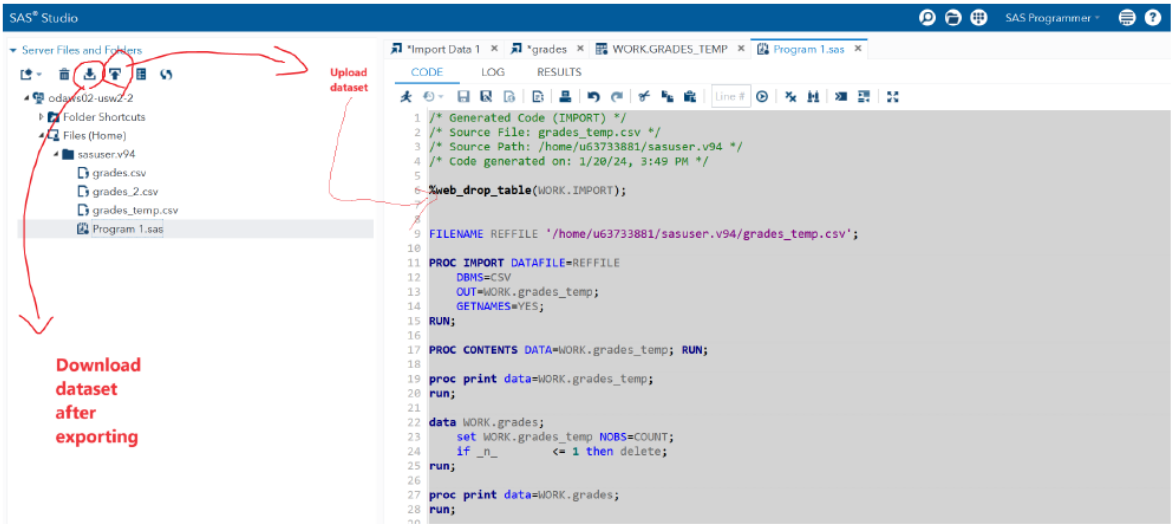
a. The DATALINES statement is used when reading data located in a raw data file.

b. The DATALINES statement is used when reading data located directly in the program.

3.3 Import & Export in SAS Virtual Studio

The key is

- Upload data before conducting "Import"
- Download data after conducting "Export"



A example given below will be demonstrated during the class.

```
/* Generated Code (IMPORT) */
/* Source File: grades_temp.csv */
/* Source Path: /home/u63733881/sasuser.v94 */
/* Code generated on: 1/20/24, 3:49 PM */

%web_drop_table(WORK.IMPORT);

FILENAME REFFILE '/home/u63733881/sasuser.v94/grades_temp.csv';

PROC IMPORT DATAFILE=REFFILE
    DBMS=CSV
    OUT=WORK.grades_temp;
    GETNAMES=YES;
RUN;

PROC CONTENTS DATA=WORK.grades_temp;
RUN;

PROC print data=WORK.grades_temp;
RUN;

DATA WORK.grades;
SET WORK.grades_temp NOBS=COUNT;
IF _n_ <= 1 THEN DELTE;
RUN;
```

```
PROC PRINT DATA=WORK.grades;
RUN;

PROC EXPORT data=WORK.grades
    OUTFILE = "/home/u63733881/sasuser.v94/grades_2.csv"
    DBMS = csv
    REPLACE;
RUN;
```

3.4 Solution to the example

Answer of the example: b

4 Random Variables

Learning Objectives

1. Distinguish between **discrete** and **continuous** random variables
2. Define random variables for real-world data problems
3. Identify appropriate probability distributions for common data types
4. Connect random variables to **data columns** used in SAS programs to represent a real world question

4.1 What Is a Random Variable?

A **random variable (RV)** is a numerical quantity whose value depends on the outcome of a random experiment.

We typically denote a random variable by an uppercase letter, such as X and its realized value by a lowercase letter, such as $X = x$. The random variable can be *continuous* or *discrete*.

In practice, we often observe multiple realizations, or running the random experiment multiple times, say n times or n realizations. We denote these realizations as:

$$X_1 = x_1,\ X_2 = x_2,\ \ldots,\ X_n = x_n.$$

The number n is called the **sample size**.

Exercise: Define whether the following random variables are discrete or continuous, and the possible values that X takes.

- Number of emails received by a server in one hour ($X = 0, 1, 2, \ldots$: discrete)
- Time (in minutes) until a machine fails ($X = x \in [0, \infty)$: continuous)
- Total number of defects on a manufactured item ($X = 0, 1, 2, \ldots$: discrete)
- Daily maximum temperature in Atlanta (in degrees Fahrenheit) ($X = x \in (-\infty, \infty)$: continuous)
- Whether a randomly selected loan defaults within one year ($X = 1$ if default, 0 otherwise: discrete)

Exercise: Define whether the following random variables are discrete or continuous, and the possible values that X takes.

1. Number of transactions made by a customer in a day
2. Response time (in seconds) of a web service request
3. Count of hospital admissions in a city per week discrete)
4. Proportion of time a system is idle during a day

4.2 Discrete vs Continuous Random Variables

4.2.1 Discrete Random Variables

A **discrete random variable** takes values in a **finite or countable set**.

Examples:

- Number of heads in three coin flips
- Number of students passing an exam
- Number of events occurring in a fixed time period

A discrete random variable is described by a **probability mass function (PMF)**:

Value of X	x_1	x_2	x_3	...	x_m
Probability	p_1	p_2	p_3	...	p_m

These probabilities satisfy:

- $0 \leq p_i \leq 1$,
- $\sum_{i=1}^{m} p_i = 1$.

We calculate the probability of events modelled by discrete random variables by summing up the probability p_i for the values x_i that make up the event.

Suppose the length X (in minutes) of an international phone call has distribution:

X	1	2	3	4
P(X)	0.2	0.5	0.2	0.1

Then, calculate the following probabilities

a. $P(X \leq 2)$
b. $P(X < 2)$
c. $P(X > 1)$

4.3 Common Discrete Distributions

4.3.1 Bernoulli Distribution

Used for **binary outcomes**:

- success / failure
- yes / no
- 1 / 0

Notation: $X \sim \text{Ber}(p)$, where p is the probability of success.

Example:
Whether a patient has diabetes (1 = yes, 0 = no).

How to specify the probability p will be discussed in the following lecture. This is related to the procedure of statistical inference.

4.3.2 Poisson Distribution

Used to model **counts of events over time or space**.

Notation: $X \sim \text{Poi}(\lambda)$

where λ is the **mean rate**.

Examples:

- Number of trades per day
- Number of system failures per week
- Number of arrivals to a service queue

Similar to the probability p from the Bernoulli distribution, how to specify the rate λ will be discussed in the following lecture.

4.4 Continuous Random Variables

A **continuous random variable** can take **any value** x **in an interval**.

Examples:

- Height of individuals
- Time until failure of a component
- Test scores treated as continuous

Probabilities are defined using **density functions**, not point probabilities. Some common continuous distributions are as follows.

4.4.1 Normal Distribution

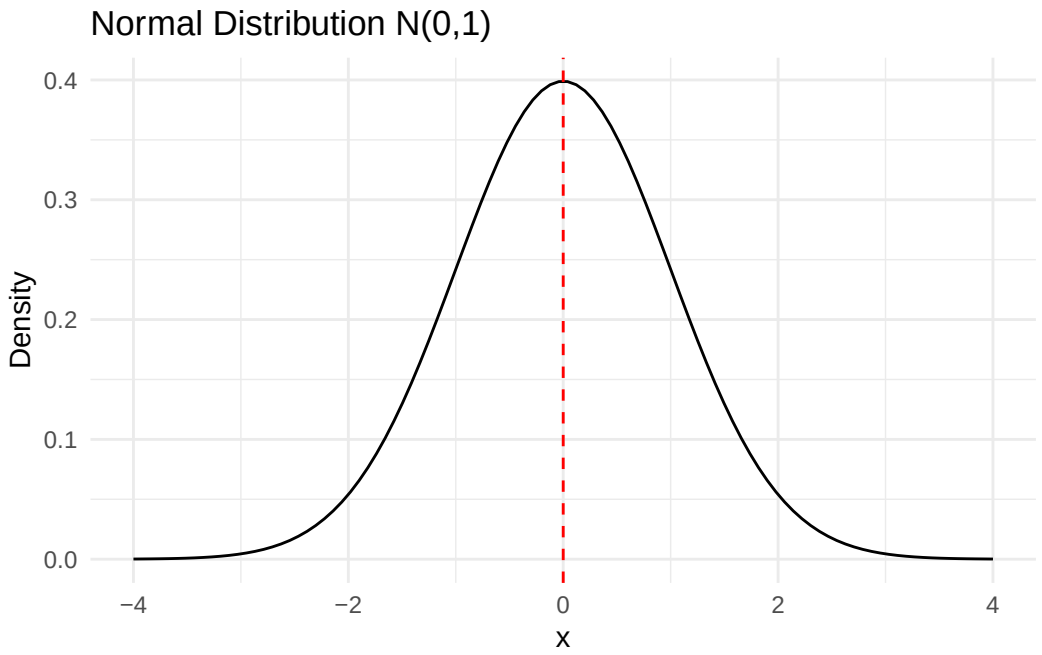
The normal distribution is also called the **Gaussian distribution**. It is characterized by two parameters: the mean μ and the standard deviation σ. It is unimodal and symmetric around the mean which is the centre of the mass. A continuous random variable X that has a normal distribution is said to be *normal* or *normally distributed*.

Notation: $X \sim N(\mu, \sigma^2)$. Sometimes the standard deviation may be used instead of the variance, which is the square of the variance.

Institution of the normal distribution:

- mean: μ
- variance: σ^2

```
# Example: plot normal distribution with mean = 0, sd = 1
x <- seq(-4, 4, length=100)
y <- dnorm(x, mean=0, sd=1)
library(ggplot2)
# use ggplot2
data.frame(x, y) |>
  ggplot(aes(x=x, y=y)) +
  geom_line() +
  ggtitle("Normal Distribution N(0,1)") +
  xlab("x") + ylab("Density") + theme_minimal() +
  geom_vline(xintercept=0, linetype="dashed", color="red")
```

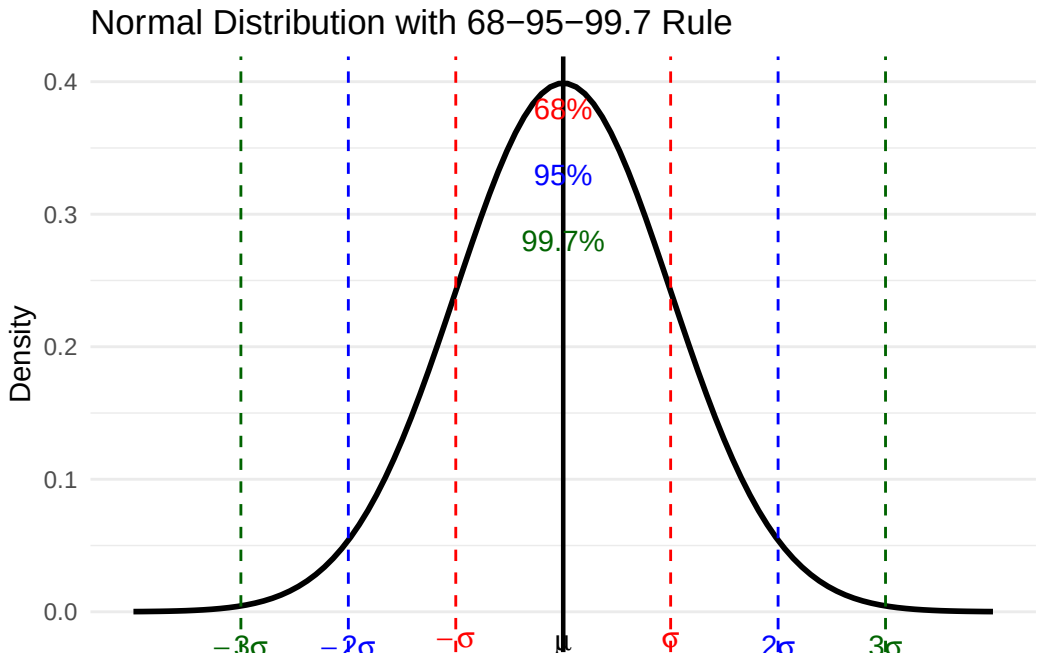


A distribution plot of the normal distribution with mean 0 and standard deviation 1 is shown above. It is often referred as the *standard normal distribution*. In practice, we would like to "standardized" the data to have mean 0 and standard deviation 1 for the subsequent analysis.

Normal distribution play an important role in statistical inference. One of the important property is the 68-95-99.7 rule: - About 68% of the data falls within one standard deviation of the mean - About 95% of the data falls within two standard deviations of the mean - About 99.7% of the data falls within three standard deviations of the mean

```
x <- seq(-4, 4, length = 100)
y <- dnorm(x)

library(ggplot2)

df <- data.frame(x, y)

ggplot(df, aes(x = x, y = y)) +
  geom_line(linewidth = 1) +
  theme_minimal() +
  labs(
    title = "Normal Distribution with 68-95-99.7 Rule",
    x = NULL,
    y = "Density"
  ) +
```

```r
# remove numeric x-axis ticks and labels
scale_x_continuous(breaks = NULL) +

# vertical reference lines
geom_vline(xintercept = 0, linewidth = 0.8) +
geom_vline(xintercept = c(-1, 1), linetype = "dashed", color = "red") +
geom_vline(xintercept = c(-2, 2), linetype = "dashed", color = "blue") +
geom_vline(xintercept = c(-3, 3), linetype = "dashed", color = "darkgreen") +

# percentage labels
annotate("text", x = 0, y = 0.38, label = "68%", color = "red") +
annotate("text", x = 0, y = 0.33, label = "95%", color = "blue") +
annotate("text", x = 0, y = 0.28, label = "99.7%", color = "darkgreen") +

# mu and sigma labels only
annotate("text", x =  0, y = -0.01, label = expression(mu), vjust = 1.5) +
annotate("text", x =  1, y = -0.01, label = expression(sigma),  vjust = 1.5, color = "red")
annotate("text", x = -1, y = -0.01, label = expression(-sigma), vjust = 1.5, color = "red")
annotate("text", x =  2, y = -0.01, label = expression(2*sigma),  vjust = 1.5, color = "blu
annotate("text", x = -2, y = -0.01, label = expression(-2*sigma), vjust = 1.5, color = "blu
annotate("text", x =  3, y = -0.01, label = expression(3*sigma),  vjust = 1.5, color = "dar
annotate("text", x = -3, y = -0.01, label = expression(-3*sigma), vjust = 1.5, color = "dar
```



4.4.2 Exponential Distribution

The exponential distribution is another common distribution where a few outcomes are most likely with a rapidly decreasing probability for larger values. It is often used to model waiting times or lifetimes of objects. It is similar to the *geometric distribution* in the discrete case.

Two ways to specify the parameter.

1. $X \sim \text{Exp}(\lambda)$, where λ is the *rate* parameter.
2. $X \sim \text{Exp}(\beta)$, where β is the *scale* parameter, and $\beta = 1/\lambda$.

The notation is either $X \sim \text{Exp}(\lambda)$ or $X \sim \text{Exp}(\beta)$. But to be sure which parameterization is used, we need to check the definition of the distribution.

Please define adequate random variables for the following data example, and thaink about what distribution can be used to model the data.

- Suppose we want to know whether the rate of diabetes of a certain area is too high. The researchers randomly discuss with 10 individuals in a certain area to know whether they have diabetes. The data are collected as "yes" or "no" answers as follows

Individual	1	2	3	4	5	6	7	8	9	10
Outcome	1	0	0	0	0	1	0	0	0	0

- A comapny that manufactures light blulb claims that a particular type of light bulb will last 850 hours on average. To justify the claim more scientifically, a researcher randomly selects 10 light bulbs of that type and measures the lifetimes (in hours) of the light bulbs as follows:

Light Bulb	1	2	3	4	5	6	7	8
	831	832	840	819	822	836	829	817

In the following classes we will assume the dataset we have can be well-modelled represented for the underline studies. The data collection, however, is beyond the scope of the class. For those interested in data collection, please refer to the *survey sampling* or *experimental design* area.

4.5 Solution to the exercise

Answer of the exercise

Exercise 1

1. Number of transactions made by a customer in a day ($X = 0, 1, 2, ...$: discrete)
2. Response time (in seconds) of a web service request ($X = x > 0$: continuous)
3. Count of hospital admissions in a city per week ($X = 0, 1, 2, ...$: discrete)
4. Proportion of time a system is idle during a day ($X = x \in [0, 1]$: continuous)

Exercise 2:

- $P(X \leq 2) = 0.7$
- $P(X < 2) = 0.2$
- $P(X > 1) = 0.8$

Part II

Statistical Analysis

5 Introduction to Statistical Inference – part I

6 Introduction to Statistical Inference – part II

7 One Sample Nonparametric Test

8 One Sample Proportion Test

9 Writing SAS Macro Program: Using One Sample Variance Test/CI as Example

10 χ^2 Goodness of Fit Test

References