SAS Tutorial for Statistical Data Analysis

Tingfang Wang

2024

Contents

Introduction

SAS (Statistical Analysis System) is a powerful software suite widely used for data management, analytics, and business intelligence. It provides a comprehensive range of tools and functionalities for data manipulation, statistical analysis, and reporting. This tutoring guide aims to help you get started with SAS and develop your skills in using this tool effectively.

In this SAS tutorial, we will cover some of the statistical procedures for data analysis that you will use in this class.

Access SAS Studio through SAS OnDemand for Academics

To access SAS Studio through SAS OnDemand for Academics, you can follow the steps outlined below:

Step 1: Register for SAS OnDemand

- Visit the SAS OnDemand website (https://www.sas.com/en_us/software/on-demand-for-academics.html).
- Click on the "Access Now" button to initiate the registration process.
- Follow the instructions to sign up for a free SAS profile by providing the required information.
- Once you complete the registration, you will receive an email with further instructions and your login credentials.

Step 2: Launching SAS Studio

- Open a web browser and navigate to the SAS OnDemand login page (https://odamid.oda.sas.com/SASODA/Home).
- Enter your SAS OnDemand account credentials (username and password) and click on the "Sign In" button.

Step 3: Accessing SAS Studio

- After successfully signing in, you will be redirected to the SAS OnDemand home page.
- Click on the "Launch SAS Studio" button. This will open SAS Studio, which is a web-based interface for SAS programming and analysis.

Step 4: Starting a SAS Session

- Once SAS Studio is launched, you will see the SAS Studio interface with various windows and tabs.
- To start a new SAS session, click on "New" in the toolbar and select "Program" from the drop-down menu. This will open a new program editor window.
- Alternatively, you can also choose to open existing SAS programs or access other SAS resources using the available options in the interface.

Step 5: Using SAS Studio

- The SAS Studio interface provides a programming environment similar to the traditional SAS software.
- Write or paste your SAS code in the program editor window.
- To execute the code, click on the "Run" button or use the keyboard shortcut (F3).
- The output and log will be displayed in separate windows within the SAS Studio interface.

Upload files to SAS Studio

To upload a data file to SAS Studio, you can follow these steps:

- Log into your account and open SAS Studio.
- In the "Explorer" pane on the left side, navigate to the folder where you want to upload the data file. You can choose an existing folder or create a new one.
- Once you're in the desired folder, click on the "Upload" button at the top of the "Explorer" pane. A file browser window will appear. Use this window to locate and select the data file you want to upload from your local machine.
- After selecting the file, click the "Open" button in the file browser window. SAS Studio will start uploading the file to the selected folder. The progress will be displayed in the "Uploads" section at the bottom of the SAS Studio interface.
- Once the upload is complete, you will see the data file appear in the folder you selected. You can now use this file in your SAS programs or analysis.

Please note that the steps may vary slightly depending on the version and setup of your SAS Studio environment. However, the general process remains the same: navigate to the desired folder, click "Upload," select the file, and wait for the upload to complete.

If you encounter any issues or need further assistance, please refer to the documentation or support resources provided by your SAS Studio environment.

Read files

SAS provides several methods to import data from files. We will explore the `DATA Step with INFILE statement` to read those files.

DATA step with the INFILE statement Syntax

In SAS, the `DATA step with the INFILE statement` is used to read data from an external file and create a SAS dataset. It allows you to control the data reading process, specify variable attributes, manipulate data values, and create new variables.

```
DATA output_dataset_name; #for example DATA my_data;
  INFILE 'path_to_the_file' DLM = ',' FIRSTOBS = 1;
  LENGTH variable $ length;
  INPUT variable1 variable2 ...; #Adjust the variable names according to your file.
RUN;
```

- `output_dataset_name`: Assign a name for the SAS dataset that will be created.
- `INFILE`: Specify the path to the file you want to import.
- `DLM`: Specify the delimiter used in the file. Typically, it is a comma (',') for CSV files. For DAT files, the delimiter can be a comma, tab, semicolon, or any other character chosen for separating the values.
- `FIRSTOBS`: Specify the line number from which SAS should start reading the data. By default, it starts from the first line (1).
- `LENGTH`: Specify the length of character variables when creating or modifying a dataset. It allows you to define the desired length for one or more character variables in the dataset.
- `INPUT`: Define the variables you want to read from the file, specifying their names in the same order as they appear in the file. SAS differentiates between variables whose values are numeric and variables whose values are character. For character variables, a dollar sign '$' must be added after the name of the variable.

Examples

Read a CSV file

1. Download the `iris` dataset from this link, and save the downloaded file as a csv file to a location on your computer.
2. Reading the iris dataset using the `DATA` step with the `INFILE` statement.

```
/* read .csv use infile */
DATA iris;
  INFILE 'path_to_file.csv' DLM=',' FIRSTOBS=1;
  LENGTH SepalLength SepalWidth PetalLength PetalWidth 8. Species $15.;
  INPUT SepalLength SepalWidth PetalLength PetalWidth Species $;
RUN;

/* Print the first 5 rows of the iris data */
PROC PRINT DATA = iris (OBS=5);
RUN;
```

Obs	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa

.

Read a DAT file

1. Download the `iris` dataset from this link, and save the downloaded file as a `dat` file to a location on your computer.
2. Reading the iris dataset using the `DATA` step with the `INFILE` statement.

```
/* read .dat use infile */
DATA iris;
  INFILE 'path_to_file.dat' DLM=',' FIRSTOBS=1;
  LENGTH SepalLength SepalWidth PetalLength PetalWidth 8. Species $15.;
  INPUT SepalLength SepalWidth PetalLength PetalWidth Species $;
RUN;

/* Print the first 5 rows of the iris data */
PROC PRINT DATA = iris (OBS=5);
RUN;
```

Obs	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa

.

Data manipulation

In this section, you will see some examples of how to manually create a dataset with the `DATA` step and how to modify a dataset that you have already loaded in your environment in SAS.

Create datasets

To manually create a dataset in SAS, you can use the DATA step. The DATA step allows you to define the structure of the dataset and manually input the data values. Here's an example of how to create datasets manually in SAS.

```
DATA sample_data1;
   LENGTH Name $20 Gender $20;
   INPUT ID Name $ Gender $;
   DATALINES;
   1 John Male
   2 Sara Female
   3 Steven Male
   ;
RUN;

DATA sample_data2;
   INPUT ID Height;
   DATALINES;
   1 180
   2 170
   3 170
   ;
RUN;

PROC PRINT DATA = sample_data1;
RUN;

PROC PRINT DATA = sample_data2;
RUN;
```

- Start with the 'DATA' statement followed by the name of your dataset. (sample_data1 and sample_data2 in this example)
- Use the `LENGTH` statement to specify the length of the character variable. In this case, the character variables' length are 20.
- Use the `INPUT` statement to define the variables and their types, separated by spaces.
- After the `DATALINES` statement, input the data values for each variable, separated by spaces or other delimiters. Each line represents a new observation.

Obs	Name	Gender	ID
1	John	Male	1
2	Sara	Female	2
3	Steven	Male	3

Obs	ID	Height
1	1	180
2	2	170
3	3	170

.

Merge datasets

SAS allows us to merge multiple datasets based on common variables using the `MERGE` statement. For example:

```
DATA merged_sample_data;
   MERGE sample_data1 sample_data2;
   BY ID;
RUN;

PROC PRINT DATA = merged_sample_data;
RUN;
```

Obs	Name	Gender	ID	Height
1	John	Male	1	180
2	Sara	Female	2	170
3	Steven	Male	3	170

.

Add a column

To add a column to a dataset in SAS, you can use the `DATA step` and the `SET` statement along with a `RETAIN` statement to define and initialize the new column.

```
/* Adding a column named 'SepalRatio' to the iris dataset, representing the ratio of sepal
length to sepal width */
data iris_with_ratio;
  set iris;
  retain SepalRatio;
  if SepalWidth ne 0 then /* make sure the denominator is not zero
```

```
    SepalRatio = SepalLength / SepalWidth;
run;

/* Print the first 5 rows of the new dataset */
proc print data = iris_with_ratio (obs=5);
run;
```

Obs	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	SepalRatio
1	5.1	3.5	1.4	0.2	Iris-setosa	1.45714
2	4.9	3.0	1.4	0.2	Iris-setosa	1.63333
3	4.7	3.2	1.3	0.2	Iris-setosa	1.46875
4	4.6	3.1	1.5	0.2	Iris-setosa	1.48387
5	5.0	3.6	1.4	0.2	Iris-setosa	1.38889

In this example, we add a new column named 'SepalRatio' to the existing 'iris' dataset. The SET statement is used to read the observations from the original dataset. The RETAIN statement initializes the new column and retains its value across iterations. We calculate the ratio of sepal length to sepal width and assign it to the 'SepalRatio' column.

Remove a column

To remove a column from a dataset in SAS, you can use the `DROP` statement within a `DATA step` to exclude the desired column.

```
/* Removing the 'PetalWidth' column from the iris dataset */
data iris_without_width;
  set iris (drop=PetalWidth);
run;

/* Print the first 5 rows of the new dataset */
proc print data = iris_without_width (obs=5);
run;
```

Obs	SepalLength	SepalWidth	PetalLength	Species
1	5.1	3.5	1.4	Iris-setosa
2	4.9	3.0	1.4	Iris-setosa
3	4.7	3.2	1.3	Iris-setosa
4	4.6	3.1	1.5	Iris-setosa
5	5.0	3.6	1.4	Iris-setosa

In this example, we create a new dataset named 'iris_without_width' by excluding the 'PetalWidth' column from the original 'iris' dataset. The DROP statement specifies the column to be dropped.

Modify a column

To modify a column in a dataset in SAS, you can use the `DATA step` along with assignment statements `variable = expression` to update the values in the desired column.

Example 1

```
/* Modifying the 'SepalLength' column in the iris dataset to convert all values to their logarithm */
data iris_modified;
  set iris;
  SepalLengthLog = log(SepalLength);
run;

/* Print the first 5 rows of the new dataset */
proc print data = iris_modified (obs=5);
run;
```

Obs	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	SepalLengthLog
1	5.1	3.5	1.4	0.2	Iris-setosa	1.62924
2	4.9	3.0	1.4	0.2	Iris-setosa	1.58924
3	4.7	3.2	1.3	0.2	Iris-setosa	1.54756
4	4.6	3.1	1.5	0.2	Iris-setosa	1.52606
5	5.0	3.6	1.4	0.2	Iris-setosa	1.60944

In this example, we create a new dataset named 'iris_modified' by modifying the 'SepalLength' column in the original 'iris' dataset. The `log()` function is used to convert all values in the 'SepalLength' column to their logarithm and save the new values in a new column named `SepalLengthLog`.

Example 2

```
/* Modifying the 'Species' column in the iris dataset to convert all values to uppercase */
data iris_modified;
  set iris;
  Species = upcase(Species);
run;

/* Print the first 5 rows of the new dataset */
proc print data = iris_modified (obs=5);
run;
```

Obs	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	5.1	3.5	1.4	0.2	IRIS-SETOSA
2	4.9	3.0	1.4	0.2	IRIS-SETOSA
3	4.7	3.2	1.3	0.2	IRIS-SETOSA
4	4.6	3.1	1.5	0.2	IRIS-SETOSA
5	5.0	3.6	1.4	0.2	IRIS-SETOSA

.

In this example, we create a new dataset named 'iris_modified' by modifying the 'Species' column in the original 'iris' dataset. The `upcase()` function is used to convert all values in the 'Species' column to uppercase.

Descriptive statistics

In SAS, the PROC MEANS procedure is used to calculate descriptive statistics for variables in a dataset. It provides a comprehensive summary of key statistical measures such as mean, median, standard deviation, minimum, maximum, quartiles, and more. Let's explore the PROC MEANS procedure in more detail:

PROC MEANS Syntax

```
PROC MEANS <options>;
  BY variable;
  VAR variable-list;
  OUTPUT <options> <statistics>=output-dataset;
RUN;
```

PROC MEANS : Additional options can be specified to control the behavior of the procedure. Some commonly used options include:

- `DATA=dataset`: Specifies the input dataset on which the descriptive statistics will be calculated.
- `BY variable`: This statement is used to calculate descriptive statistics separately for each level of a categorical variable. Multiple variables can be specified in the BY statement separated by spaces.
- `VAR variable-list`: Specifies the list of variables for which descriptive statistics are calculated. Multiple variables can be specified in the VAR statement separated by spaces.
- `MISSING`: Includes missing values in the analysis.

OUTPUT =output-dataset: This statement allows you to save the calculated statistics in an output dataset.

- use `out = stats` to name the output dataset as `stats`.
- The `<statistics>` option specifies the desired statistics to be calculated, such as `MEAN`, `MEDIAN`, `STD`, `MIN`, `MAX`, `Q1`, `Q3`, etc. Multiple statistics can be specified, and each statistic will create a corresponding variable in the output dataset.

Example

We will use the iris data in the last example to get descriptive statistics for the variable "SepalLength".

1. Descriptive statistics for "SepalLength" for all species combined.

```
/* Descriptive statistics using PROC MEANS */
PROC MEANS DATA = iris;
   VAR SepalLength;
   OUTPUT OUT=STATS MEAN=MEAN STD=STDDEV MIN=MINIMUM
      Q1 = Q1 MEDIAN=MEDIAN Q3=Q3 MAX=MAXIMUM;
RUN;

/* Output dataset from PROC MEANS */
PROC PRINT DATA=STATS;
RUN;
```

The MEANS Procedure

Analysis Variable : SepalLength				
N	Mean	Std Dev	Minimum	Maximum
150	5.8433333	0.8280661	4.3000000	7.9000000

Obs	_TYPE_	_FREQ_	MEAN	STDDEV	MINIMUM	Q1	MEDIAN	Q3	MAXIMUM
1	0	150	5.84333	0.82807	4.3	5.1	5.8	6.4	7.9

2. Descriptive statistics for "SepalLength" by different species.

```
/* Descriptive statistics using PROC MEANS */
PROC MEANS DATA = iris;
   BY Species;
   VAR SepalLength;
   OUTPUT OUT=STATS MEAN=MEAN STD=STDDEV MIN=MINIMUM
      Q1 = Q1 MEDIAN=MEDIAN Q3=Q3 MAX=MAXIMUM;
RUN;

/* Output dataset from PROC MEANS */
PROC PRINT DATA=STATS;
RUN;
```

Obs	Species	_TYPE_	_FREQ_	MEAN	STDDEV	MINIMUM	Q1	MEDIAN	Q3	MAXIMUM
1	Iris-setosa	0	50	5.006	0.35249	4.3	4.8	5.0	5.2	5.8
2	Iris-versicolor	0	50	5.936	0.51617	4.9	5.6	5.9	6.3	7.0
3	Iris-virginica	0	50	6.588	0.63588	4.9	6.2	6.5	6.9	7.9

Plots

In SAS, there are various procedures and techniques available for creating plots and visualizations to explore and present data. SAS provides a rich set of graphical procedures, such as `PROC PLOT`, `PROC SGPLOT`, `PROC GPLOT`, `PROC GCHART`, and more, that allows you to create a wide range of plots including scatter plots, bar charts, histograms, box plots, and many others. We will be focusing on the `PROC PLOT` and `PROC SGPLOT` procedures in this tutorial.

PROC PLOT Syntax

The `PROC PLOT` procedure plots the values of two variables for each observation in an input SAS data set. The coordinates of each point on the plot correspond to the two variables' values in one or more observations of the input data set.

```
PROC PLOT <options>;
   PLOT <plot-variable(s)> <options>;
   <BY variables>;
   <PLOT2 statement>;
   <PLOT3 statement>;
   ...
   <PLOTn statement>;
RUN;
```

- The `PROC PLOT` statement begins the procedure and may include options to control the overall behavior of the procedure.
- The `PLOT` statement specifies the plot variables and options for each plot. You can include multiple PLOT statements to create multiple plots within a single PROC PLOT.
- The <plot-variable(s)> specifies the two variables of the plot ($y * x$).
- The `BY` statement is optional and allows you to create separate plots for different levels of a categorical variable. It is useful when you want to compare groups or categories.
- The `PLOT2`, `PLOT3`, ..., `PLOTn` statements are optional and can be used to specify additional plots in the same PROC PLOT. This allows you to create multiple plots within a single procedure call.

PROC SGPLOT Syntax

```
PROC SGPLOT <options>;
  <plot-statement(s)>;
  <plot-options>;
RUN;
```

- PROC SGPLOT: This statement initiates the PROC SGPLOT procedure.

`<options>`: Additional options can be specified to control the behavior of the procedure. Some commonly used options include:

- `DATA=dataset`: Specifies the input dataset containing the variables to be plotted.
- `TITLE "plot title"`: Specifies the title for the plot.

- `BY variable`: Specifies a categorical variable to create separate plots for each level of the variable.

`<plot-statement(s)>`: This refers to the statements that define the specific type of plot to be created. Some commonly used plot statements include:

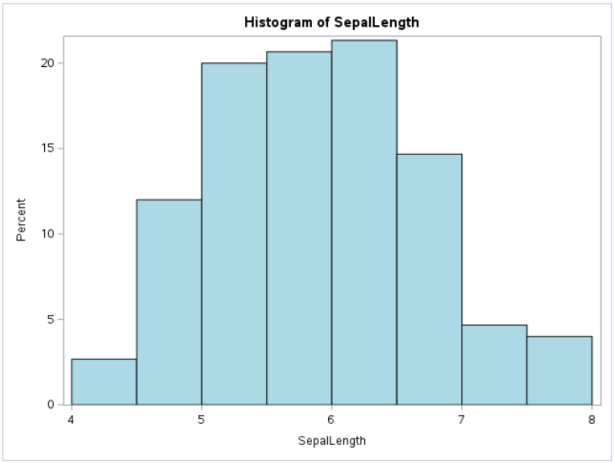
- `SCATTER` : Creates a scatter plot.
- `SERIES` : Creates a line plot.
- `HISTOGRAM` : Creates a histogram.
- `BAR` : Creates a bar chart.
- `BOXPLOT` : Creates a box plot.
- `VBOX` : Creates vertical box plot.

`<plot-options>`: These options are specific to each plot statement and control the appearance and behavior of the plot. For example, you can specify variables to be plotted, color schemes, line styles, axis labels, and many more.

Histogram

Histogram of "SepalLength" in iris dataset using the PROC SGPLOT procedure

```
/* Create a histogram using PROC SGPLOT */
PROC SGPLOT DATA = iris;
   TITLE "Histogram of Sepal Length";
   HISTOGRAM SepalLength / FILLATTRS=(COLOR=lightblue);
RUN;
```



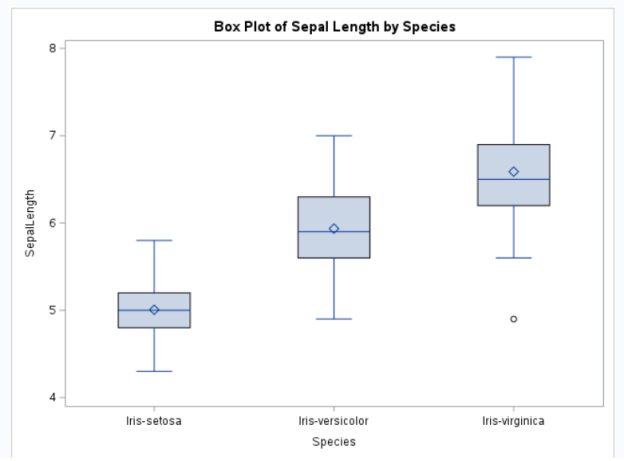
.

The `FILLATTRS=(COLOR=blue)` option sets the fill color of the histogram bars to blue. You can customize the color or other attributes based on your preference.

Boxplot

Boxplots of "SepalLength" by different iris species.

```
/* Create box plots using PROC SGPLOT */
PROC SGPLOT DATA = iris;
   TITLE 'Box Plot of Sepal Length by Species';
   VBOX SepalLength / CATEGORY=Species;
RUN;
```



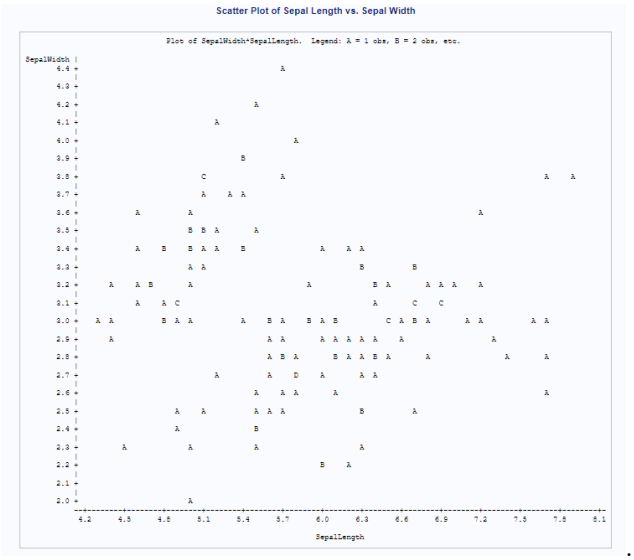
Within the `VBOX` statement, we specify the variable to be plotted on the vertical axis, which is "SepalLength" in this case. The `CATEGORY=Species` option indicates that the box plots should be created for each unique value of the "Species" variable, which represents the different iris species.

Scatterplot

1. Scatter Plot of Sepal Length vs. Sepal Width using PROC PLOT

```
PROC PLOT DATA=iris;
  PLOT SepalWidth*SepalLength;
  TITLE 'Scatter Plot of Sepal Length vs. Sepal Width';
RUN;
```

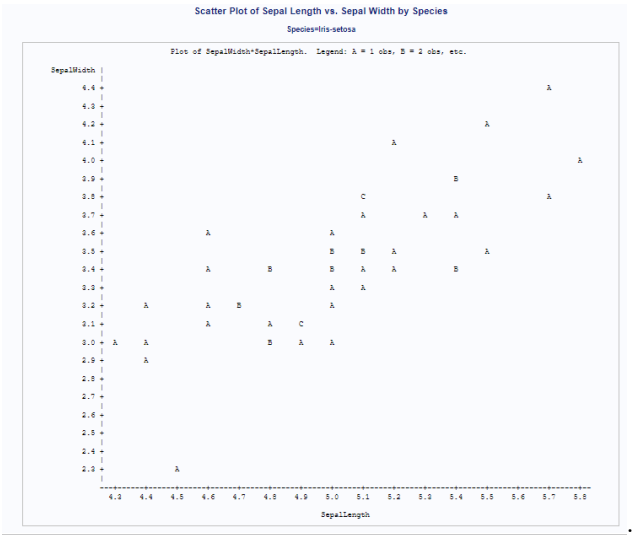
This example uses `PROC PLOT` to create a scatter plot of the "SepalWidth" variable on the y-axis and the "SepalLength" variable on the x-axis from the iris dataset.

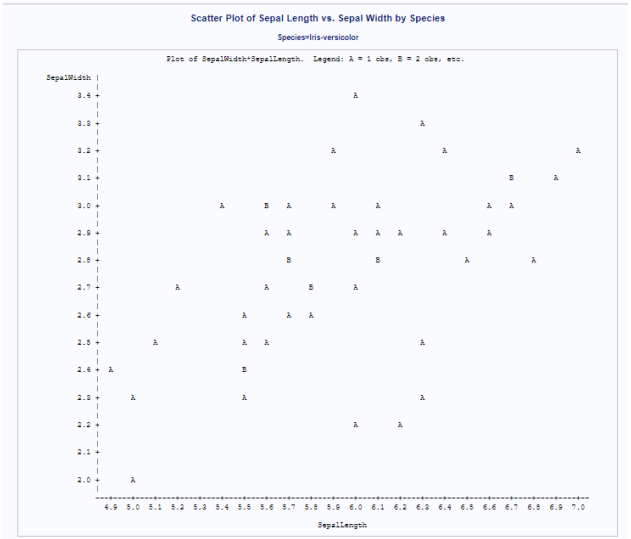


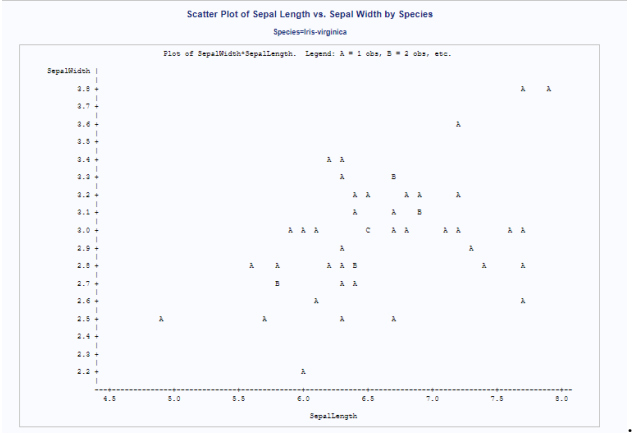
2. Scatter Plot of Sepal Length vs. Sepal Width by Species using PROC PLOT

```
PROC PLOT DATA=iris;
  BY=Species;
```

```
  PLOT SepalWidth*SepalLength;
  TITLE 'Scatter Plot of Sepal Length vs. Sepal Width by Species';
RUN;
```

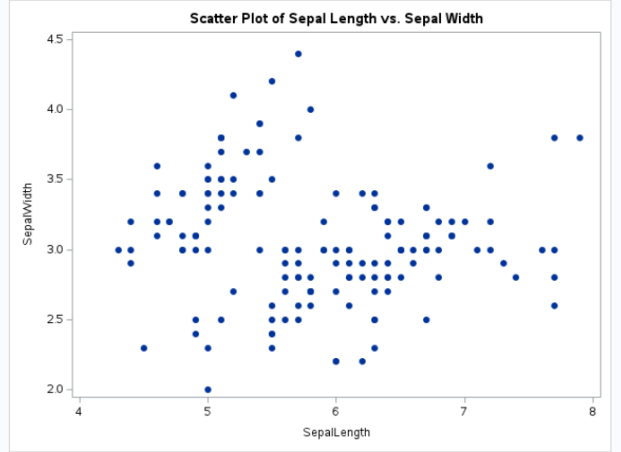






3. Scatter Plot of Sepal Length vs. Sepal Width using PROC SGPLOT

```
/* Create a scatter plot using PROC SGPLOT */
PROC SGPLOT DATA = iris;
   TITLE 'Scatter Plot of Sepal Length vs. Sepal Width';
   SCATTER x=SepalLength y=SepalWidth / MARKERSATTRS=(Symbol=CircleFilled);
RUN;
```



The `MARKERSATTRS=(Symbol=CircleFilled)` option sets the marker symbol for the scatter plot to a filled circle. You can customize the symbol or other attributes based on your preference.

QQ plot

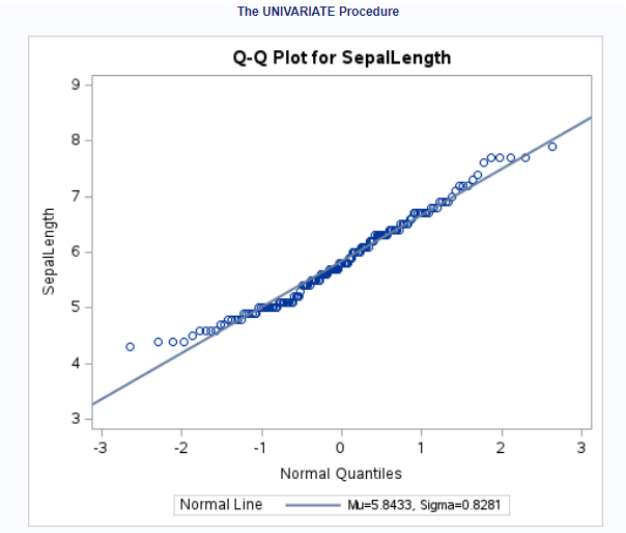
1. Syntax for QQ plot

```
PROC UNIVARIATE DATA=<dataset> NOPRINT;
   QQPLOT <variable> / NORMAL(MU=EST SIGMA=EST);
RUN;
```

   - NOPRINT prevents the procedure from printing the results in the output window. It's useful when you only want to generate graphical output.
   - QQPLOT : Replace with the name of the variable you want to analyze. This command generates a quantile-quantile plot for the specified variable.
   - The MU=EST and SIGMA=EST options use estimated mean and standard deviation values to create the theoretical normal quantiles.

2. QQ plot for `SepalLength` in the iris data

```
PROC UNIVARIATE DATA = IRIS NOPRINT;
   QQPLOT SepalLength / NORMAL(MU=EST SIGMA=EST);
RUN;
```


The UNIVARIATE Procedure
Q-Q Plot for SepalLength
SepalLength
Normal Quantiles
9
8
7
6
5
4
3
-3
-2
-1
0
1
2
3
Normal Line
Mu=5.8433, Sigma=0.8281

.

Statistical tests

In SAS, there are numerous statistical tests available for analyzing data and making inferences. Here's a summarized list of some commonly used statistical tests in SAS:

1. Parametric Tests:
    - One-sample t-Test: Compares whether the mean of a sample significantly differs from a known or hypothesized population mean. `PROC TTEST`
    - Two-sample t-Test: Compares means between two groups. `PROC TTEST`
    - Analysis of Variance (ANOVA): Compares means across multiple groups. `PROC ANOVA`
    - Analysis of Covariance (ANCOVA): Compares means while controlling for covariates. `PROC GLM`
    - Multivariate Analysis of Variance (MANOVA): Compares means of multiple dependent variables across groups. `PROC GLM or PROC MANOVA`
2. Nonparametric Tests:
    - Wilcoxon Signed Rank Test: Compares paired samples. `PROC UNIVARIATE`
    - Wilcoxon-Mann-Whitney Rank Sum Test: Compares independent samples. `PROC NPAR1WAY`
    - Kruskal-Wallis Test: Compares multiple independent samples. `PROC NPAR1WAY`
    - Friedman Test: Compares multiple related samples. `PROC FREQ or PROC NPAR1WAY`
3. Chi-square Tests:
    - Chi-square Test of Independence: Tests the independence between categorical variables. `PROC FREQ`
    - Chi-square Goodness-of-Fit Test: Tests if observed frequencies match expected frequencies. `PROC FREQ`
    - Fisher's Exact Test: Tests independence for small sample sizes. `PROC FREQ`
4. Survival Analysis:
    - Kaplan-Meier Survival Analysis: Estimates survival curves and compares survival times between groups. `PROC LIFETEST`
    - Cox Proportional Hazards Model: Assesses the relationship between survival and covariates. `PROC PHREG`

In this section, we will be focusing on the one-sample t-test, two-sample t-test, Wilcoxon-Mann-Whitney rank sum Test, and Chi-square test, regression related tests will be introduced in the `Regression` section.

One-sample t-test

1. Syntax

```
PROC TTEST DATA=<dataset>
   H0: <null hypothesis mean>
   ALPHA=<significance level>;
   VAR <variable>;
RUN;
```

- `PROC TTEST` initiates the TTEST procedure for hypothesis testing.

- `DATA=<dataset>` specifies the name of the dataset containing the variable of interest.
- `H0: <null hypothesis mean>` sets the null hypothesis mean to a specific value. Replace with the desired value you want to test against.
- `ALPHA=<significance level>` sets the significance level (alpha) for the hypothesis test. Replace with the desired value (e.g., 0.05 for a 5% significance level).
- `VAR <variable>` specifies the continuous variable for which you want to conduct the one-sample t-test.

2. To test if the mean "SepalLength" is different from 6cm.

```
/* One-sample t-test: is the mean sepal length different from 6cm? */
PROC TTEST DATA = iris
    H0 = 6
    ALPHA = 0.05;
    VAR SepalLength;
RUN;
```

The TTEST Procedure

Variable: SepalLength

N	Mean	Std Dev	Std Err	Minimum	Maximum
150	5.8433	0.8281	0.0676	4.3000	7.9000

Mean	95% CL Mean		Std Dev	95% CL Std Dev	
5.8433	5.7097	5.9769	0.8281	0.7438	0.9341

DF	t Value	Pr > |t|
149	-2.32	0.0219

.

Two-sample t-test

1. Syntax

```
PROC TTEST DATA=<dataset>;
   CLASS <grouping variable>;
   VAR <variable>;
RUN;
```

- `PROC TTEST` initiates the TTEST procedure for hypothesis testing.
- `DATA=<dataset>` specifies the name of the dataset containing the variables of interest.
- `CLASS <grouping variable>` specifies the categorical variable that defines the groups being compared.
- `VAR <variable>` specifies the continuous variable for which you want to compare the means between groups.

2. To test if the mean "SepalLength" of "Iris-setosa" and "Iris-virginica" are the same.

```
/* Take a subset of the iris data so the new data only have two iris species*/
DATA iris_subset;
   SET iris;
   IF Species = 'Iris-setosa' or Species = 'Iris-virginica';
```

```
RUN;

/* Two-sample t-test: is the mean sepal length different in the two iris species?*/
PROC TTEST DATA = iris_subset;
   CLASS Species;
   VAR SepalLength;
RUN;
```

The TTEST Procedure

Variable: SepalLength

Species	Method	N	Mean	Std Dev	Std Err	Minimum	Maximum
Iris-setosa		50	5.0060	0.3525	0.0498	4.3000	5.8000
Iris-virginica		50	6.5880	0.6359	0.0899	4.9000	7.9000
Diff (1-2)	Pooled		-1.5820	0.5141	0.1028		
Diff (1-2)	Satterthwaite		-1.5820		0.1028		

Species	Method	Mean	95% CL Mean		Std Dev	95% CL Std Dev	
Iris-setosa		5.0060	4.9058	5.1062	0.3525	0.2944	0.4392
Iris-virginica		6.5880	6.4073	6.7687	0.6359	0.5312	0.7924
Diff (1-2)	Pooled	-1.5820	-1.7860	-1.3780	0.5141	0.4511	0.5977
Diff (1-2)	Satterthwaite	-1.5820	-1.7868	-1.3772			

Method	Variances	DF	t Value	Pr > |t|
Pooled	Equal	98	-15.39	<.0001
Satterthwaite	Unequal	76.516	-15.39	<.0001

Equality of Variances				
Method	Num DF	Den DF	F Value	Pr > F
Folded F	49	49	3.25	<.0001

Wilcoxon-Mann-Whitney rank sum test

1. Syntax

```
PROC NPAR1WAY DATA=<dataset> WILCOXON;
   CLASS <grouping variable>;
   VAR <variable>;
RUN;
```

- `PROC NPAR1WAY` initiates the nonparametric one-way analysis of variance (ANOVA) procedure.
- `WILCOXON` indicates that you want to perform the Wilcoxon-Mann-Whitney rank sum test.
- `DATA=<dataset>` specifies the name of the dataset containing the variables of interest.
- `CLASS <grouping variable>` specifies the categorical variable that defines the groups being compared.
- `VAR <variable>` specifies the continuous variable for which you want to compare the distributions between groups.

2. Wilcoxon-Mann-Whitney rank sum test: Sepal Length by Species

```
/* Take a subset of the iris data so the new data only have two iris species*/
DATA iris_subset;
   SET iris;
   IF Species = 'Iris-setosa' or Species = 'Iris-virginica';
RUN;
```

```
/*  Wilcoxon-Mann-Whitney rank sum test */
PROC NPAR1WAY DATA=iris_subset WILCOXON;
   CLASS Species;
   VAR SepalLength;
RUN;
```

The NPAR1WAY Procedure

Wilcoxon Scores (Rank Sums) for Variable SepalLength Classified by Variable Species

Species	N	Sum of Scores	Expected Under H0	Std Dev Under H0	Mean Score
Iris-setosa	50	1313.50	2525.0	144.896776	26.270
Iris-virginica	50	3736.50	2525.0	144.896776	74.730

Average scores were used for ties.

Wilcoxon Two-Sample Test

Statistic	Z	Pr < Z	Pr > |Z|	t Approximation Pr < Z	t Approximation Pr > |Z|
1313.500	-8.3577	<.0001	<.0001	<.0001	<.0001

Z includes a continuity correction of 0.5.

Kruskal-Wallis Test

Chi-Square	DF	Pr > ChiSq
69.9084	1	<.0001

.

Chi-square test of independence

1. Syntax

```
PROC FREQ DATA=<dataset>;
   TABLES <variable1> * <variable2> / CHISQ;
RUN;
```

- `PROC FREQ` initiates the frequency analysis procedure.
- `DATA=<dataset>` specifies the name of the dataset containing the variables of interest.
- `TABLES <variable1> * <variable2>` specifies the two categorical variables for which you want to test the association. Separate the variable names with an asterisk (*) to indicate the cross-tabulation.
- `/ CHISQ` specifies that you want to perform a chi-square test.

2. Chi-Square test to test the association between vehicle type and drive type.

```
/* Loading the SAS build in CARS dataset from the sashelp library*/
DATA CARS;
   SET SASHELP.CARS;
RUN;

/* Chi-Square Test to Test Association between Vehicle Type and Drive Type */
PROC FREQ DATA=CARS;
   TABLES Type * DriveTrain / CHISQ;
RUN;
```

The FREQ Procedure

Frequency Percent Row Pct Col Pct	Table of Type by DriveTrain			
	DriveTrain			
Type	All	Front	Rear	Total
Hybrid	0 0.00 0.00 0.00	3 0.70 100.00 1.33	0 0.00 0.00 0.00	3 0.70
SUV	38 8.88 63.33 41.30	22 5.14 36.67 9.73	0 0.00 0.00 0.00	60 14.02
Sedan	28 6.54 10.69 30.43	179 41.82 68.32 79.20	55 12.85 20.99 50.00	262 61.21
Sports	5 1.17 10.20 5.43	8 1.87 16.33 3.54	36 8.41 73.47 32.73	49 11.45
Truck	12 2.80 50.00 13.04	0 0.00 0.00 0.00	12 2.80 50.00 10.91	24 5.61
Wagon	9 2.10 30.00 9.78	14 3.27 46.67 6.19	7 1.64 23.33 6.36	30 7.01
Total	92 21.50	226 52.80	110 25.70	428 100.00

Statistics for Table of Type by DriveTrain

Statistic	DF	Value	Prob
Chi-Square	10	185.6710	<.0001
Likelihood Ratio Chi-Square	10	187.5595	<.0001
Mantel-Haenszel Chi-Square	1	15.7157	<.0001
Phi Coefficient		0.6586	
Contingency Coefficient		0.5501	
Cramer's V		0.4657	

Sample Size = 428

Chi-square goodness of fit test

1. Syntax

```
PROC FREQ DATA=<dataset>;
   TABLES <variable> / CHISQ TESTP;
RUN;
```

- `PROC FREQ` initiates the frequency analysis procedure.
- `DATA=<dataset>` specifies the name of the dataset containing the variables of interest.
- `TABLES <variable>` specifies the categorical variable you want to test.
- `/ CHISQ` specifies that you want to perform a chi-square test.
- `TESTP` option is used to request expected frequencies.

2. Chi-Square goodness of fit test to determine if the proportions of species in the iris dataset are the same.

```
/* PERFORM CHI-SQUARE GOODNESS OF FIT TEST */
PROC FREQ DATA = iris;
   TABLES SPECIES / CHISQ;
RUN;
```

The FREQ Procedure

Species	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Iris-setosa	50	33.33	50	33.33
Iris-versicolor	50	33.33	100	66.67
Iris-virginica	50	33.33	150	100.00

Chi-Square Test for Equal Proportions	
Chi-Square	0.0000
DF	2
Pr > ChiSq	1.0000

.

3. Chi-Square goodness of fit test to determine if the proportions of species in the iris dataset are 0.2, 0.5, 0.3.

```
/* PERFORM CHI-SQUARE GOODNESS OF FIT TEST */
PROC FREQ DATA = iris;
   TABLES SPECIES / CHISQ TESTP = (20, 50, 30);
RUN;
```

The FREQ Procedure

Species	Frequency	Percent	Test Percent	Cumulative Frequency	Cumulative Percent
Iris-setosa	50	33.33	20.00	50	33.33
Iris-versicolor	50	33.33	50.00	100	66.67
Iris-virginica	50	33.33	30.00	150	100.00

Chi-Square Test for Specified Proportions	
Chi-Square	22.2222
DF	2
Pr > ChiSq	<.0001

.

Data simulation

In SAS, data simulation can be achieved using various functions and procedures. In this tutorial, we will focus on sampling from uniform, normal, and binomial distributions using different SAS functions.

Sampling from a uniform distribution

1. `RANUNI` function

   To sample from a uniform distribution in SAS, you can use the `RANUNI` function, which generates random numbers between 0 and 1. You can adjust the range by scaling and shifting the generated values.

2. Example

```
/* Generate a random sample of size 100 from a uniform distribution between 5 and 15,
and save the sample as `uniform_sample` */
data uniform_sample;
  do i = 1 to 100;
x = 5 + 10 * ranuni(156); # seed = 156
output;
  end;
run;

/* Print the first 5 rows of the sample */
proc print data = uniform_sample (OBS=5);
run;
```

Obs	i	x
1	1	5.0171
2	2	14.9440
3	3	14.5604
4	4	11.1611
5	5	5.5040

.

Sampling from a normal distribution

1. `RANNOR` function

   To sample from a normal distribution in SAS, you can use the `RANNOR` function, which generates random numbers from a standard normal distribution (mean=0, standard deviation=1). You can modify the mean and standard deviation by scaling and shifting the generated values.

2. Example

```
/* Generate a random sample of size 100 from a normal distribution with mean=50
and std deviation=10, and save the sample as 'normal_sample' */
data normal_sample;
  do i = 1 to 100;
x = 50 + 10 * rannor(156); # seed = 156
output;
  end;
run;

/* Print the first 5 rows of the sample */
proc print data = normal_sample (OBS=5);
run;
```

Obs	i	x
1	1	60.9257
2	2	67.4171
3	3	34.7176
4	4	65.8279
5	5	37.3917

.

Sampling from a binomial distribution

1. `RANBIN` Function

   To sample from a binomial distribution in SAS, you can use the `RANBIN` function, which generates random numbers from a binomial distribution with a specified number of trials and probability of success.

2. Example

```
/* Generate a random sample of size 100 from a binomial distribution with the
probability of success being 0.8 and the number of trials being 50, and
save the sample as binomial_sample*/
data binomial_sample;
  do i = 1 to 100;
x = ranbin(156, 50, 0.8); # seed = 156, n = 50, p = 0.8
output;
  end;
run;

/* Print the first 5 rows of the sample */
proc print data = binomial_sample (obs=5);
run;
```

Obs	i	x
1	1	37
2	2	41
3	3	41
4	4	40
5	5	38

Sampling using the rand() function

1. `RAND` function syntax

   The RAND function generates random numbers from various continuous and discrete distributions.

```
RAND('dist', parm, ...parm)
```

   - `dist` is a character constant, variable, or expression that identifies the distribution.
   - `parm` are shape, location, or scale parameters that are appropriate for the specific distribution.

2. Example

```
Data sample_data;
call streaminit(12345);
do i = 1 to 100;
x1 = rand("Normal", 50, 3.2);
x2 = rand("Binomial", 0.8, 63);
x3 = rand("Bernoulli", 0.2);
x4 = rand("Uniform");
x5 = rand("Table", .2,.3,.3,.2);
x6 = rand("Table", 1/6,1/6,1/6,1/6,1/6,1/6);
output;
end;
run;
```

Obs	i	x1	x2	x3	x4	x5	x6
1	1	50.8455	47	0	0.64740	2	3
2	2	53.4391	47	0	0.84267	2	6
3	3	46.3219	54	0	0.15068	4	5
4	4	48.4558	42	1	0.73841	3	5
5	5	55.3223	45	1	0.19799	3	5

Monte Carlo simulation to estimate the p-value of a test

1. Outline

- Define the Hypotheses: Clearly state the null hypothesis H_0 and the alternative hypothesis H_1.
- Data Simulation: Simulate a large number of samples under the null hypothesis.
- Calculate Test Statistics: Apply the test to each simulated dataset and calculate the test statistic.
- Count Extreme Test Statistics: Count how many of the simulated test statistics are as extreme as, or more extreme than, the observed test statistic.
- Estimate P-value: The estimated p-value is the proportion of simulated test statistics that are as extreme as, or more extreme than, the observed test statistic.

2. Example

Suppose we want to estimate the p-value of the one-sample t-test to test if the mean sepal length different from 6cm in the iris data using Monte Carlo simulation. We can do the following:

```
/* read csv use infile */

DATA iris;
  INFILE '/iris.csv' DLM=',' FIRSTOBS=1;
  LENGTH SepalLength SepalWidth PetalLength PetalWidth 8. Species $15.;
  INPUT SepalLength SepalWidth PetalLength
PetalWidth Species $;
RUN;

/* One-sample t-test on the original data: is the mean sepal length different
   from 6cm? */

PROC UNIVARIATE DATA = iris
   MU0=6;
   VAR SepalLength;
/* save standard deviation and sample size in dataset SepalLength*/
   OUTPUT OUT = SepalLength T = t_obs N = N STDDEV = Sd PROBT = pValue_obs;
RUN;

/* Define the number of MC runs*/

DATA SepalLength;
   SET SepalLength;
   Nruns = 10000;
   Mu0 = 6; /* Nruns = number of MC runs and Mu0 = null hypothesis*/
RUN;

/* Data simulation: create dataset MC_M*/

DATA MC_M;
   SET SepalLength;
   call streaminit(54321);
   DO MCrun=1 TO Nruns;
 DO j=1 TO N; /* Generate Nruns samples of size N of normal variables */
 Y = RAND("NORMAL", Mu0, Sd); /*Generate N(Mu0=6, Sd^2) variate and saves in Y.*/
 OUTPUT; /* ensures no overwriting of the saved Y */
 END; /* at this point for a given MCrun value, a sample of size N has been generated*/
   END; /* Nruns replicates generated; each replicate of size N */
RUN;

/* Check the simulated data (2 samples)*/
DATA MC_M_2Sample;
```

```
   SET MC_M;
   WHERE MCrun <= 2;
RUN;

PROC MEANS DATA = MC_M_2Sample MEAN STD MIN MAX;
   BY MCrun;
   VAR Y;
RUN;

/* Perform t-test on each of the simulated samples*/

PROC UNIVARIATE DATA=MC_M
   MU0=6 NOPRINT;
   VAR Y;
   CLASS MCrun; /* Compute test statistic T for each sample*/
   OUTPUT OUT=SampleResult T=T_MC;
RUN;

/* Extending dataset SepalLength to the size of data set SampleResult
   to compare the test statistics*/

DATA SepalLength;
   SET SepalLength;
   DO i=1 TO Nruns; OUTPUT; END;
RUN;

/* Compare the test statistics */

DATA Compare;
   MERGE SepalLength SampleResult;
   indicator = 2*( T_MC >= abs(T_obs)); /*The indicator is 2 if
   T_MC >= abs(T_obs) and 0 if otherwise */
RUN;

/* Calculate the estimated p-value*/

PROC MEANS DATA=Compare NOPRINT;
   VAR indicator;
   OUTPUT OUT=P MEAN=Pvalue;
RUN;

/* Report the p-value*/
PROC PRINT DATA=P; TITLE 'Estimated p-value';
   VAR Pvalue;
RUN;
```

Observed statistic

Obs	N	Sd	t_obs	pValue_obs	Nruns	Mu0
1	150	0.82807	-2.31717	0.021857	10000	6

Estimated p-value

Obs	Pvalue
1	0.0218

.

Regression

SAS provides several procedures for regression analysis, including `PROC REG`, `PROC GLM`, `PROC GENMOD`, and more. These procedures offer flexibility in terms of the types of regression models that can be estimated, such as linear regression, logistic regression, Poisson regression, and so on.

Here is a brief explanation of the steps involved in performing regression analysis in SAS:

- Data Preparation: First, you need to import your data into SAS. This can be done using the `Data step with INFILE statement` or by creating a SAS dataset manually. Ensure that the dataset contains the necessary variables for your regression analysis, including the dependent variable and independent variables.
- Choose the appropriate regression procedure: SAS offers several procedures for regression analysis, depending on the type of dependent variable and the specific objectives of your analysis. Some commonly used procedures include `PROC REG`, `PROC GLM`, `PROC GENMOD`, and `PROC LOGISTIC`.
- Specify the regression model: Use the appropriate procedure and specify the regression model by identifying the dependent variable and independent variables. This is typically done using the `MODEL` statement.
- Review the output: After running the regression procedure, SAS generates a comprehensive output that includes various statistics, tables, and plots. Some key elements to review and interpret include:
  - Regression coefficients; Standard errors and p-values; Model fit statistics such as R-squared, adjusted R-squared, and root mean square error (RMSE); Residual analysis to assess the assumptions of the regression model, such as normality, constant variance, and independence of errors, etc.

Data Preparation

In this tutorial, we will use the `CARS` in the `SASHELP` library to demonstrate how to do regression in SAS.

The "CARS" dataset provides information about various car models, including their make, model, origin, and specifications.

Here are some details about the variables included in the "CARS" dataset:

- Make: The manufacturer or brand of the car (e.g., Ford, Toyota, Honda).
- Model: The specific model or name of the car (e.g., Accord, Camry, Focus).
- Type: The type of car, such as compact, midsize, or SUV.
- Origin: The country of origin for the car (e.g., USA, Japan, Europe).
- DriveTrain: The type of drive-train, indicating whether the car is front-wheel drive (FWD), rear-wheel drive (RWD), or all-wheel drive (AWD).
- MSRP: The manufacturer's suggested retail price for the car.
- Invoice: The invoice price, which is the amount paid by the dealer for the car.
- EngineSize: The size of the car's engine in liters.
- Cylinders: The number of cylinders in the car's engine.
- Horsepower: The horsepower rating of the car.
- MPG_City: The estimated fuel efficiency in miles per gallon (MPG) during city driving.
- MPG_Highway: The estimated fuel efficiency in miles per gallon (MPG) during highway driving.
- Weight: The weight of the car in pounds.
- Wheelbase: The distance between the front and rear axles of the car.
- Length: The length of the car in inches.
- Width: The width of the car in inches.

```
/* Loading the SAS build in CARS dataset from the SASHELP library*/
DATA CARS;
   SET SASHELP.CARS;
RUN;
```

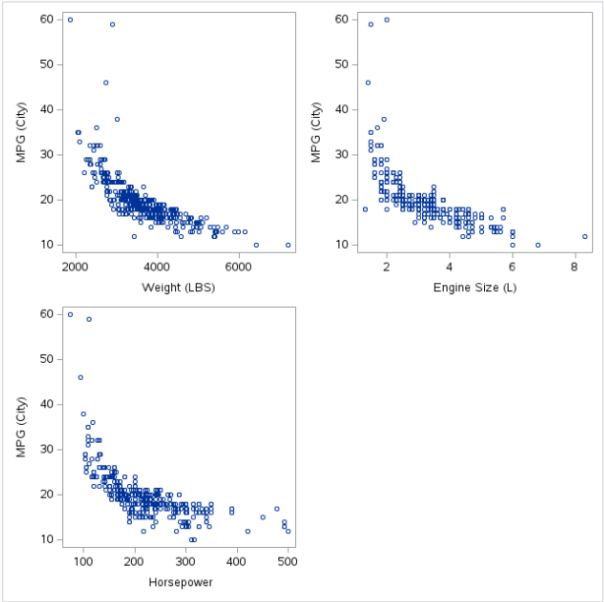
```
![Image Title](cars_data.png){width=50%}.
```

Check the data

Checking scatter plots and correlation coefficients before performing a regression analysis is important as they allow you to visually inspect the relationship between the independent and dependent variables, detect potential outliers, and check for multicollinearity.

1. Scatter plots

```
/* Make multiple scatter plots*/
PROC GPLOT Data = CARS;
PLOT MPG_City*(Weight EngineSize HorsePower);
RUN;

/* make a multi-celled scatter plot*/
PROC SGSCATTER DATA=CARS;
  PLOT (MPG_CITY)*(WEIGHT ENGINESIZE HORSEPOWER);
RUN;
```



2. Correlation coefficients

```
/* Check correlation coefficients*/
PROC CORR DATA=CARS;
VAR MPG_CITY WEIGHT ENGINESIZE HORSEPOWER;
RUN;
```

The CORR Procedure

4 Variables:	MPG_City Weight EngineSize Horsepower

Simple Statistics

Variable	N	Mean	Std Dev	Sum	Minimum	Maximum	Label
MPG_City	428	20.06075	5.23822	8586	10.00000	60.00000	MPG (City)
Weight	428	3578	758.98321	1531364	1850	7190	Weight (LBS)
EngineSize	428	3.19673	1.10859	1368	1.30000	8.30000	Engine Size (L)
Horsepower	428	215.88551	71.83603	92399	73.00000	500.00000	

Pearson Correlation Coefficients, N = 428
Prob > |r| under H0: Rho=0

	MPG_City	Weight	EngineSize	Horsepower
MPG_City MPG (City)	1.00000	-0.73797 <.0001	-0.70947 <.0001	-0.67670 <.0001
Weight Weight (LBS)	-0.73797 <.0001	1.00000	0.80787 <.0001	0.63080 <.0001
EngineSize Engine Size (L)	-0.70947 <.0001	0.80787 <.0001	1.00000	0.78743 <.0001
Horsepower	-0.67670 <.0001	0.63080 <.0001	0.78743 <.0001	1.00000

Fit a regression model

Suppose we want to predict highway MPG (Miles per Gallon) based on weight and horsepower, we should specify the model as

```
/* Fit a regression model with two predictors and the intercept */
PROC REG DATA = CARS;
MODEL MPG_City = Weight Horsepower;
RUN;

/* Fit a regression model with two predictors and without the intercept */
PROC REG DATA = CARS;
MODEL MPG_City = Weight Horsepower / noint;
RUN;

/* Fit a regression model with interactions, the interaction term is created
   in the DATA step since polynomial effects such as size*type are not allowed
   in the MODEL statement in the REG procedure*/

DATA CARS;
SET CARS;
WH_Inter = Weight*Horsepower; /*the interaction term*/
RUN;

PROC REG data=cars;
   model MPG_City = Weight Horsepower WH_Inter;
RUN;
```

Once fit the model(run the above code), we can review the output to make inferences.

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	60.36917	1.99957	30.19	<.0001
Weight	Weight (LBS)	1	-0.01014	0.00060904	-16.64	<.0001
Horsepower		1	-0.12302	0.00870	-14.13	<.0001
WH_Inter		1	0.00002791	0.00000239	11.67	<.0001

Regression diagnosis

1. Lack of fit test

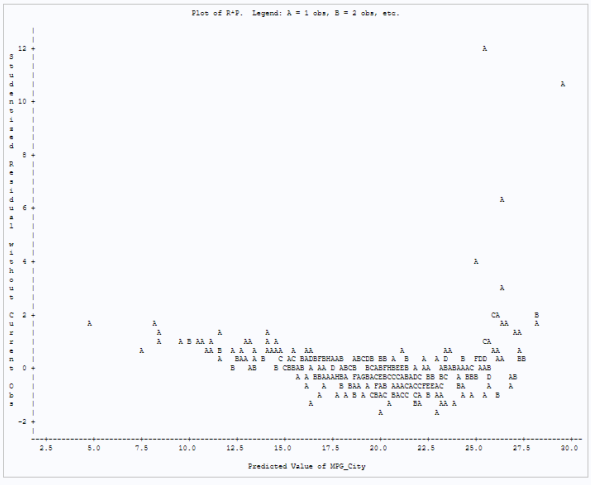
```
/* Lack of fit test to check linearity*/
PROC REG DATA = CARS;
MODEL MPG_City = Weight Horsepower EngineSize/ lackfit;
/*save the studentized residuals and the predicted values */
OUTPUT OUT=D RSTUDENT=R PREDICTED=P;
```

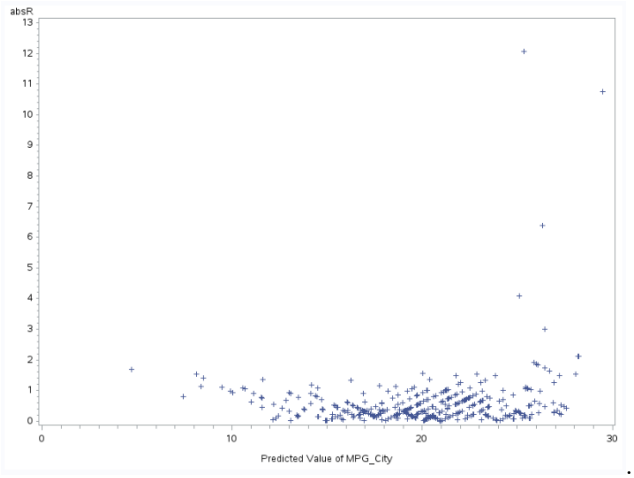
In the above `OUTPUT` statement, a dataset named "D" was created, this dataset contains all the columns in the original "CARS" data along with two additional columns which are the studentized residuals "R" and predicted values "P". Now, you can use this new dataset to do more analysis.

Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	7262.04146	2420.68049	230.42	<.0001
Error	424	4454.37910	10.50561		
Lack of Fit	400	4453.37910	11.13345	267.20	<.0001
Pure Error	24	1.00000	0.04167		
Corrected Total	427	11716			

2. Residual plot

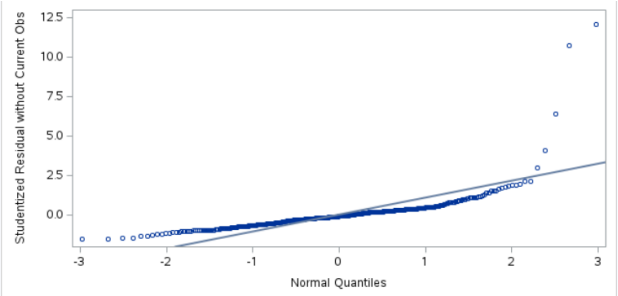
```
/* Residual plot*/
PROC PLOT Data=D;
plot R*(P Weight Horsepower EngineSize);
RUN;

/* Save the absolute value of residuals */
DATA D; SET D;
absR = abs(R);
RUN;

/* Sbsolute residuals vs fitted values to check homogeneity assumption */
PROC GPLOT DATA = D;
PLOT absR*P;
RUN;
```




absR
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0
10
20
30
Predicted Value of MPG_City

3. Residual QQ plot

```
/*Residual QQ plot*/
PROC UNIVARIATE DATA=D NORMAL PLOT; /* Check normality of the studentized residuals */
VAR R;
RUN;
```



4. Breusch Pagan Test

```
/* Breusch Pagan Test*/
PROC MODEL DATA=D;
PARMS b0 b1 b2 b3;
MPG_City = b0 + b1*Weight + b2*Horsepower + b3*EngineSize;
/*Breusch-Pagan test for heteroscedasticity (BREUSCH) wrt the specified predictors*/
fit MPG_City /WHITE BREUSCH=(Weight Horsepower EngineSize);
/*Breusch-Pagan test for heteroscedasticity individually for each of the predictors*/
fit MPG_City /BREUSCH=(Weight);
fit MPG_City /BREUSCH=(Horsepower);
fit MPG_City /BREUSCH=(EngineSize);
RUN;
```

Heteroscedasticity Test					
Equation	Test	Statistic	DF	Pr > ChiSq	Variables
MPG_City	White's Test	44.22	9	<.0001	Cross of all vars
	Breusch-Pagan	11.42	3	0.0096	Weight, Horsepower, EngineSize, 1

Heteroscedasticity Test					
Equation	Test	Statistic	DF	Pr > ChiSq	Variables
MPG_City	Breusch-Pagan	7.13	1	0.0076	Weight, 1

Heteroscedasticity Test					
Equation	Test	Statistic	DF	Pr > ChiSq	Variables
MPG_City	Breusch-Pagan	10.37	1	0.0013	Horsepower, 1

Heteroscedasticity Test					
Equation	Test	Statistic	DF	Pr > ChiSq	Variables
MPG_City	Breusch-Pagan	6.64	1	0.0100	EngineSize, 1

5. Brown Forsythe test

```
/* Brown Forsythe Test for homogeneity of variance*/

/* Get the medians of the predictors*/
PROC UNIVARIATE DATA = D NOPRINT;
VAR Weight Horsepower EngineSize;
OUTPUT OUT = Medians Median = MedW Median = MedH Median = MedE N=N;
RUN;

DATA Medians;
SET medians;
DO i = 1 TO N;
OUTPUT;
END;
RUN;

/*Test for homogeneity of residuals grouped by Weight*/
DATA WeightBF;
MERGE D Medians;
Group=(Weight > MedW);
RUN;

PROC GLM Data = WeightBF;
class Group;
model R = Group;
means Group / hovtest = BF;
run;
```

The GLM Procedure

Brown and Forsythe's Test for Homogeneity of R Variance
ANOVA of Absolute Deviations from Group Medians

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Group	1	5.2289	5.2289	6.03	0.0145
Error	426	369.7	0.8677		

.

Extra sum of squares

```
/* Prints both Type I and Type III SS along with partial F tests */
PROC GLM DATA = CARS;
MODEL MPG_City = Weight Horsepower EngineSize;
RUN;
```

The GLM Procedure

Dependent Variable: MPG_City MPG (City)

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	7262.04146	2420.68049	230.42	<.0001
Error	424	4454.37910	10.50561		
Corrected Total	427	11716.42056			

R-Square	Coeff Var	Root MSE	MPG_City Mean
0.619817	16.15710	3.241236	20.06075

Source	DF	Type I SS	Mean Square	F Value	Pr > F
Weight	1	6380.690159	6380.690159	607.36	<.0001
Horsepower	1	867.939854	867.939854	82.62	<.0001
EngineSize	1	13.411444	13.411444	1.28	0.2592

Source	DF	Type III SS	Mean Square	F Value	Pr > F
Weight	1	934.9461831	934.9461831	88.99	<.0001
Horsepower	1	448.4077827	448.4077827	42.68	<.0001
EngineSize	1	13.4114438	13.4114438	1.28	0.2592

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	37.98626828	0.81879266	46.39	<.0001
Weight	-0.00330834	0.00035069	-9.43	<.0001
Horsepower	-0.02314548	0.00354275	-6.53	<.0001
EngineSize	-0.34149600	0.30224458	-1.13	0.2592

Partial F test

```
/* Partial F test*/
PROC REG DATA = CARS;
MODEL MPG_City = Weight Horsepower EngineSize;
/* User-specified hypotheses about estimated parameters */
x2x3: test Horsepower = EngineSize = 0;
RUN;
```

The REG Procedure
Model: MODEL1

Test x2x3 Results for Dependent Variable MPG_City

Source	DF	Mean Square	F Value	Pr > F
Numerator	2	440.67565	41.95	<.0001
Denominator	424	10.50561		

.

Partial correlation coefficient

```
/*Partial correlation coefficient between MPG_City and Weight after
  adjusting for Horsepower and EngineSize*/

PROC CORR DATA=CARS;
VAR MPG_CITY WEIGHT;
PARTIAL HORSEPOWER ENGINESIZE;
RUN;
```

Pearson Partial Correlation Coefficients, N = 428
Prob > |r| under H0: Partial Rho=0

	MPG_City	Weight
MPG_City MPG (City)	1.00000	-0.41651 <.0001
Weight Weight (LBS)	-0.41651 <.0001	1.00000

.

Bootstrap

In general, the basic bootstrap method consists of four steps:

1. Compute a statistic of interest for the original data.
2. Use the DATA step or PROC SURVEYSELECT to resample (with replacement) B times from the data.
3. Use BY-group processing to compute the statistic of interest on each bootstrap sample.
4. Use the bootstrap distribution to obtain estimates for the bias and standard error of the statistic and confidence intervals for parameters.

In this section, we will again use the `CARS` in the `SASHELP` library to demonstrate how to do bootstrap in SAS. Suppose we are interested in the mean of horsepower.

Compute a statistic of interest for the original data

```
/* Loading the SAS build in CARS dataset from the SASHELP library*/
data CARS;
set SASHELP.CARS;
run;

/* Suppose we are interested in the mean of horsepower */
data Horsepower(keep=Horsepower);
set CARS;
run;

/* Compute mean on original data*/
proc means data=Horsepower noprint;
var Horsepower;
output out=OrigStats(keep = orig_mean orig_se) mean = orig_mean std = orig_se;
run;

proc print data = OrigStats;
run;
```

Obs	orig_mean	orig_se
1	215.886	71.8360

.

Generate bootstrap samples

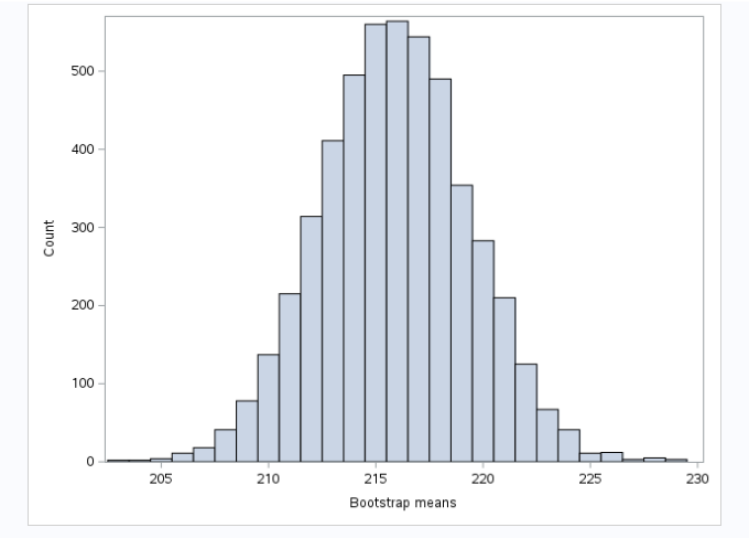
```
proc surveyselect data=Horsepower NOPRINT seed=1
     out=BootSamples
     method=urs     /* resample with replacement */
     samprate=1     /* each bootstrap sample has same number of observations as the original data*/
     OUTHITS        /* show number of times that a data point was selected */
     reps=&NumSamples; /* generate NumSamples bootstrap samples */
run;

/* Compute the statistic for each bootstrap sample */
proc means data=BootSamples noprint;
   by Replicate;
   var Horsepower;
   output out=BootMeans mean = boot_means std = boot_ses;  /* bootstrap means and standard errors */
run;

proc print data=BootMeans(obs=5);
run;
```

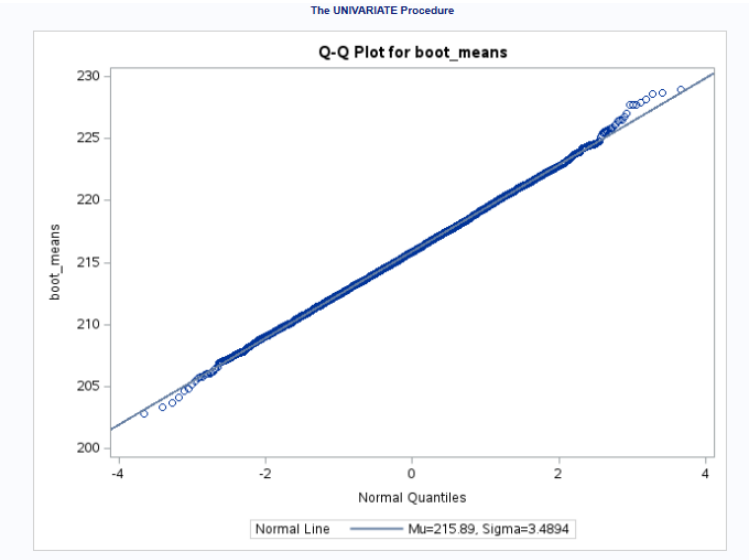
Obs	Replicate	_TYPE_	_FREQ_	boot_means	boot_ses
1	1	0	428	215.736	72.2083
2	2	0	428	212.339	70.1781
3	3	0	428	215.558	68.3142
4	4	0	428	217.745	75.9842
5	5	0	428	218.035	70.7854

.

Visualize the bootstrap distribution

```
/* Histrogram */
proc sgplot data=BootMeans;
   label boot_means = "Bootstrap means";
   histogram boot_means / scale=count;
run;

/* Q-Q Plot */
proc univariate data=BootMeans;
    var boot_means;
    qqplot / normal(mu=est sigma=est color=steelblue l=2);
run;
```





Compute bootstrap estimated bias and standard error

Bias and standard error of original sample statistic ($\hat{\theta}$):

$$\text{Bias} = \hat{B}^* = \frac{1}{B}\sum_{k=1}^{B}(\hat{\theta}_k^* - \hat{\theta})$$

$$\text{SE} = S\hat{E}^* = \sqrt{\frac{1}{B-1}\sum_{k=1}^{B}(\hat{\theta}_k^* - \bar{\hat{\theta}}^*)}$$

```
/* Compute bootstrap estimated bias and standard error */
proc means data=BootMeans noprint;
    var boot_means;
    output out = BootStats mean = boot_mean std = boot_se;
run;
```

```
/* Calculate bias */
data boot_bias_and_se(keep = orig_mean orig_se boot_bias boot_se);
    merge BootStats OrigStats;
    boot_bias = boot_mean - orig_mean;
run;

proc print data = boot_bias_and_se; run;
```

Obs	boot_se	orig_mean	orig_se	boot_bias
1	3.48942	215.886	71.8360	.005364486

.

Four bootstrap CIs

Normal approximation CI:

$$(\hat{\theta} - \hat{B}^*) \pm Z_{1-\frac{\alpha}{2}} \hat{SE}^*$$

Studentized bootstrap CI:

$$T_i^* = \frac{\hat{\theta}_i^* - \hat{\theta}}{\hat{SE}_i^*},\ \text{CI} : (\hat{\theta} - t^*_{1-\frac{\alpha}{2}} \hat{SE},\ \hat{\theta} - t^*_{\frac{\alpha}{2}} \hat{SE})$$

Basic bootstrap CI:

$$(2\hat{\theta} - \hat{\theta^*}_{1-\frac{\alpha}{2}},\ 2\hat{\theta} - \hat{\theta^*}_{\frac{\alpha}{2}})$$

Percentile bootstrap CI:

$$(\hat{\theta^*}_{\frac{\alpha}{2}},\ \hat{\theta^*}_{1-\frac{\alpha}{2}})$$

```
/* Normal Approximation CI */
data normal_ci (keep = lower_ci_norm upper_ci_norm);
    set boot_bias_and_se;
    z_value = quantile('normal', 0.975, 0, 1); /* Z-score */
    lower_ci_norm = (orig_mean - boot_bias) - z_value * boot_se;
    upper_ci_norm = (orig_mean - boot_bias) + z_value * boot_se;
run;

/* Percentile Bootstrap CI */
proc univariate data=BootMeans noprint;
    var boot_means;
    output out=boot_pctiles pctlpre =Pct95_ pctlpts=2.5 97.5 pctlname=Lower Upper;
run;
data percentile_ci(keep = lower_ci_pct upper_ci_pct);
    set boot_pctiles;
    lower_ci_pct = Pct95_Lower;
```

```
    upper_ci_pct = Pct95_Upper;
run;

/* Basic Bootstrap CI */
data basic_ci(keep = lower_ci_basic upper_ci_basic);
    merge boot_pctiles OrigStats;
    lower_ci_basic = 2 * orig_mean - Pct95_Upper;
    upper_ci_basic = 2 * orig_mean - Pct95_Lower;
run;

/* Studentized Bootstrap CI */
/* Calculate t-value percentiles */
data TValues;
   if _N_ = 1 then do;
      /* Load the single-row datasets into memory */
      set OrigStats;
      set BootMeans;
   end;
   set BootMeans;
   /* Calculate tvalue for each row */
   tvalue = (boot_means - orig_mean) / boot_ses;
run;

/* Calculate t-value percentiles */
proc univariate data=TValues noprint;
    var tvalue;
    output out=t_pctiles pctlpts=2.5 97.5 pctlpre=t_pctile_;
run;

data studentized_ci(keep = lower_ci_stu upper_ci_stu);
    merge OrigStats t_pctiles;
    lower_ci_stu = orig_mean - t_pctile_97_5 * orig_se;
    upper_ci_stu = orig_mean - t_pctile_2_5 * orig_se;
run;

/* Combine all CIs into one dataset */
data combined_ci;
    merge normal_ci studentized_ci basic_ci percentile_ci;
run;
proc print data=combined_ci noobs label;
    var lower_ci_norm upper_ci_norm lower_ci_stu upper_ci_stu
        lower_ci_basic upper_ci_basic lower_ci_pct upper_ci_pct;
    label lower_ci_norm="Normal Approximation Lower"
          upper_ci_norm="Normal Approximation Upper"
          lower_ci_stu="Studentized Lower"
          upper_ci_stu="Studentized Upper"
          lower_ci_basic="Basic Bootstrap Lower"
          upper_ci_basic="Basic Bootstrap Upper"
          lower_ci_pct="Percentile Bootstrap Lower"
          upper_ci_pct="Percentile Bootstrap Upper";
run;
```

Normal Approximation Lower	Normal Approximation Upper	Studentized Lower	Studentized Upper	Basic Bootstrap Lower	Basic Bootstrap Upper	Percentile Bootstrap Lower	Percentile Bootstrap Upper
209.041	222.719	209.320	223.008	209.112	222.609	209.162	222.659

.

Save the output in a file

Output Delivery System Syntax

To save SAS output as a PDF or Word file, you can use the Output Delivery System (ODS) in SAS. ODS allows you to direct your SAS output to different destinations, including PDF and Word files. Here's how you can save SAS output using ODS:

1. For PDF output:

```
ods pdf file="path_to_output.pdf";
/* Your SAS code producing output */
ods pdf close;
```

2. For Word output:

```
ods rtf file="path_to_output.rtf";
/* Your SAS code producing output */
ods rtf close;
```

Replace "path_to_output.pdf" with the desired file path and name for the PDF output file, and "path_to_output.rtf" with the desired file path and name for the RDF output file, although the file extension is ".rtf", it can still be opened and read by Microsoft Word. The output generated by SAS between the `ods pdf` (or `ods rtf`) and `ods pdf close` (or `ods rtf close`) statements will be saved in the specified file.

Make sure to adjust the file path and name according to your specific requirements.

Example

Suppose we want to save the SAS output of the code that reads and prints out the first 5 rows of the `iris` data. we can do:

1. Save the output as a pdf file

```
ods pdf file="'path/importiris.pdf";
/* read csv use infile */
DATA iris;
  INFILE '/home/u63468122/iris.csv' DLM=',' FIRSTOBS=1;
  LENGTH SepalLength SepalWidth PetalLength PetalWidth 8. Species $15.;
  INPUT SepalLength SepalWidth PetalLength PetalWidth Species $;
RUN;
/* Print the first 5 rows of the iris data */
PROC PRINT DATA = iris (OBS=5);
RUN;
ods pdf close;
```

2. Save the output as a rtf file(can be edited in Word)

```
ods rtf file="'path/importiris.rtf";
/* read csv use infile */
DATA iris;
  INFILE '/home/u63468122/iris.csv' DLM=',' FIRSTOBS=1;
  LENGTH SepalLength SepalWidth PetalLength PetalWidth 8. Species $15.;
  INPUT SepalLength SepalWidth PetalLength PetalWidth Species $;
RUN;
/* Print the first 5 rows of the iris data */
PROC PRINT DATA = iris (OBS=5);
RUN;
ods rtf close;
```

After running the code, you will find the specified file in the desired file path.

Conclusion

In this tutorial, we covered several important aspects of data analysis in SAS. We learned how to read data using the `INFILE` statement, perform data manipulation tasks, calculate descriptive statistics, create plots, conduct statistical tests, simulate data, perform regression analysis, bootstrap, and save the output in a file. Remember to consult SAS documentation and resources for more advanced techniques and additional options based on your specific needs.