BIVA
SEM 10 NOTES

Unit 1 - Introduction

What is SAS Visual Analytics?

- **Definition**: A web-based product that utilizes SAS high-performance analytics technologies.
- **Purpose**: Allows organizations to explore large datasets quickly to identify patterns and opportunities for further analysis.
- **Accessibility**: Reports and visual results can be shared via the web or mobile devices.

Visual Analytics Methodology

1. **Access**:
   - Identify or create analysis tables for use in Visual Analytics.
   - Load tables into the SAS LASR Analytic Server.
2. **Investigate**:
   - Assess data size, shape, quality issues (e.g., missing values), and necessary calculations.
3. **Prepare**:
   - Perform data manipulation based on findings from the Investigate phase (e.g., cleaning and calculating data).
4. **Analyze**:
   - Discover relationships in the data through:
     - Viewing correlations.
     - Using animations to track changes over time.
     - Filtering and comparing data across time periods.
     - Ranking values to highlight extremes.
5. **Report**:
   - Present results by:
     - Selecting and arranging analyses.
     - Creating multi-page reports with prompts and interactions.
     - Designing dashboards linking summarized and detailed information.

SAS Visual Analytics Architecture

- **Components**:
  - **Designer**: Tool for report design.
  - **Web Viewer**: Platform for viewing reports.
  - **Data Builder**: Facilitates data preparation.
  - **Explorer**: Tool for data exploration.
  - **Administrator**: Manages the environment.
  - **Mobile BI**: Allows for mobile report viewing.
  - **Graph Builder**: Creates visualizations.
  - **SAS LASR Analytic Server**: High-performance analytics engine.

Viewing SAS Visual Analytics Reports

- **Access Methods**:
  - **SAS Visual Analytics Viewer**: For report viewing.
  - **SAS Mobile BI app**: Access reports on mobile devices.
  - **SAS Add-In for Microsoft Office**: Integrates SAS capabilities with Office applications (requires additional licensing).

SAS Visual Analytics Server Components

SAS Visual Analytics is built on the SAS Business Analytics platform, consisting of **clients**, **server components**, and specialized servers for data analysis.

Client Access

- **Web Browser**: Access reports and perform analysis through a browser.
- **Mobile Device**: Use mobile apps for viewing and interacting with reports.

Server Components

- **SAS Visual Analytics Web Applications**:
  - **Home Page**: Central location for accessing all Visual Analytics features.
  - **Explorer**: Tool for data exploration.
  - **Designer**: Used for creating reports and dashboards.
  - **Viewer**: For viewing reports.
  - **Data Builder**: Tool for building and preparing data.
  - **Graph Builder**: Used for creating visualizations.
  - **Administrator**: For managing the SAS Visual Analytics environment.
- **Platform Servers**:
  - **SAS Metadata Server**: Manages metadata, providing a central repository for metadata used by all SAS applications.
  - **SAS Workspace Server**: Performs processing tasks for data analysis.
- **SAS LASR Analytic Server**: Provides a high-performance, in-memory analytics engine for concurrent data access and analysis.

Deployment Models

Non-Distributed Deployment

- **Typical Setup**: All components are installed on a single machine.
- **Example**: The classroom environment uses a non-distributed deployment on Windows.
- **Four-Tier Architecture**:
  1. **Data Sources**: Stores enterprise data (e.g., third-party databases, SAS tables).
  2. **SAS Servers**: Perform analytics and processing on the data.

3. **Middle Tier**: Provides web-based interfaces for report creation and information distribution, passing processing requests to the SAS servers.
4. **Clients**: Desktop access for most users, with SAS client software for advanced tasks, and mobile support available.

SAS Metadata Server

- **Function**: A multi-user server that stores, manages, and delivers metadata across SAS applications.
- **Types of Metadata**:
  - **Users**: User profiles and permissions.
  - **Tables**: Data tables.
  - **Cubes**: Analytical cubes.
  - **Reports**: Report templates and outputs.
  - **Dashboards**: Interactive report dashboards.
  - **SAS Programs**: Stored programs for analysis.
  - **Analytical Models**: Data models for analytics.

SAS LASR Analytic Server

- **Purpose**: An in-memory analytic server that allows secure, fast, multi-user access to data.
- **Key Features**:
  - **Persistence of Data in Memory**: Ensures quick access to large datasets.
  - **Concurrent User Access**: Multiple users can access the same in-memory data without interference.
  - **Improved Performance**: Fast analytic operations and reduced start-up times for distributed environments.

Distributed Deployment

- **Typical Setup**: Components are spread across multiple machines for better scalability and performance.

Roles and Capabilities

SAS Visual Analytics comes with **five predefined roles**:

1. **Visual Analytics: Administration**: Manages users, configurations, and settings.
2. **Visual Analytics: Analysis**: Performs data analysis and exploration.
3. **Visual Analytics: Basic**: Performs basic report viewing and exploration.
4. **Visual Analytics: Data Building**: Builds and prepares datasets for analysis.
5. **Visual Analytics: Report Viewing**: Views and interacts with reports.

- **Roles and Capabilities**:
  - Each role is associated with specific capabilities, or application actions, which define the tasks a user can perform.

BIVA
SEM 10 NOTES

Credentials and Security

- Administrators manage credentials, security controls, and role-based capabilities for users to ensure appropriate access to data and reports.

Data Used in Training

- Data accessed and created during training is stored in a metadata folder and registered in a library for easy access and management.

Unit 2 –

SAS Programs Overview

A **SAS program** is a sequence of one or more steps that are used for data processing, analysis, and report generation.

- **Data Steps**: Typically used to **create SAS datasets**.
- **PROC Steps**: Typically used to **process SAS datasets** for generating reports, graphs, and managing data.

SAS Program Structure

A **step** in SAS is a sequence of SAS statements.

- **Step Boundaries**:
  - SAS steps begin with either:
    - A **DATA** statement (for data steps).
    - A **PROC** statement (for PROC steps).
  - SAS detects the end of a step when it encounters:
    - A **RUN** statement (for most steps).
    - A **QUIT** statement (for some procedures, such as PROC SQL).
    - The beginning of another step (another **DATA** or **PROC** statement).

SAS Syntax Rules: Statements

- **SAS Statements**:
  - Typically **begin with an identifying keyword** (e.g., DATA, PROC, SET, RUN).
  - Always **end with a semicolon (;)**.
- **SAS Program Structure**:
  - **Free format**: There are no fixed column rules. Statements can be written in any column of the program.
- **Statement Guidelines**:
  - Statements **can begin and end in any column**.
  - A **single statement** can **span multiple lines**.
  - **Several statements** can appear **on the same line** (though this is not recommended for clarity).

BIVA
SEM 10 NOTES

SAS Syntax Errors

A **syntax error** occurs when there is an issue with the spelling, grammar, or structure of a SAS statement. Syntax errors are detected during **compilation** before execution.

Detecting Syntax Errors

- SAS often **detects syntax errors** through color-coding in the **Editor tab**.
- When a syntax error is detected, SAS writes an **error or warning message** to the SAS log to inform the user of the issue.

What Is a SAS Data Set?

- A **SAS data set** is a specialized file that only SAS can read.
- It is a **table** that contains **observations** (rows) and **variables** (columns).

SAS Data Set Structure

A SAS data set consists of:

1. **Descriptor Portion** - Contains metadata:
    - **General properties** (e.g., data set name, number of observations)
    - **Variable properties** (e.g., name, type, length)
    - Use **PROC CONTENTS** to display the descriptor portion.
2. **Data Portion** - Contains actual data values:
    - Values can be **character** or **numeric**.
    - Use **PROC PRINT** to display the data portion.

Data Types

- **Character Variables:**
    - Can store letters, numerals, special characters, and blanks.
    - Length can range from **1 to 32,767 characters**.
    - **1 byte per character**.
- **Numeric Variables:**
    - Store numeric values using **floating point or binary representation**.
    - **8 bytes of storage by default**.
    - Can store **16 or 17 significant digits**.

Missing Data Values

- **Missing values** are valid and represent "no data."

- Every variable in every observation must have a value (it can be missing).

SAS Libraries

- A **SAS Library** is a collection of SAS files that are stored together.
- Libraries can be **temporary** or **permanent**.

Temporary Library:

  - **Work** is a temporary library.
  - It holds SAS data sets for the duration of a session.
  - Deleted when the session ends.

Permanent Library:

  - **Sashelp** is an example of a permanent library.
  - It contains sample data sets that can be accessed throughout the session.

Accessing SAS Data Sets

- All SAS data sets have a **two-level name**: libref.dataset-name` .
  - Example: work.newsalesemps
- If the data set is in the **temporary Work library**, a **one-level name** can be used.
  - Example: newsalesemps

User-Defined Libraries

- A **user-defined library**:
  - Created and defined by the user.
  - Permanent: data remains until the user deletes it.
  - Not automatically available in a SAS session.
  - Defined within the operating system's file system.
- To create a user-defined library, use the **LIBNAME** statement to associate a **libref** with a physical path.

LIBNAME Statement

- The **LIBNAME** statement is a global SAS statement used to define a library.
  - Syntax: LIBNAME libref "path"; Example: libname orion "s:\workshop";
  - It does not require a RUN statement.
  - It remains in effect until changed or canceled, or until the session ends.

- **Changing a libref**:
  - Modify the path in the LIBNAME statement.
  - Example: libname orion "c:\myfiles";
- **Canceling a libref**:
  - Use the CLEAR option to cancel.
  - Example: libname orion clear;

Browsing a Library

- A library can be browsed in two ways:
  1. **Programmatically**, using **PROC CONTENTS**.
     - Example: PROC CONTENTS LIBRARY=orion;
  2. **Interactively** using SAS Studio, SAS Enterprise Guide, or SAS windowing environment.

Browsing a Library Programmatically

- **PROC CONTENTS** with the ***ALL*** keyword lists all files in a library.
  - Example:

PROC CONTENTS DATA=orion._ALL_ NODS;

RUN;

  - **_ALL_** lists all the files in the library.
  - **NODS** suppresses individual dataset descriptor information.

Logical Operator Priority

- Operators can be written as symbols or mnemonics, and parentheses can be used to modify the order of evaluation.

Symbol	Mnemonic	Priority
^	NOT	I
&	AND	II
`	`	OR
¬	~	NOT

CONTAINS Operator

- The **CONTAINS** operator selects observations that include a specified substring.
- **Equivalent statements**:
  - where Job_Title contains 'Rep';
  - where Job_Title ? 'Rep'; The ? can be used instead of the mnemonic.

- Key characteristics:
    - The **position** of the substring within the variable's values **does not matter**.
    - Comparisons made with the **CONTAINS** operator are **case-sensitive**.

Special WHERE Operators

- **Special WHERE operators** can be used **only in WHERE expressions** to filter data based on specific conditions.

Operator	Definition	Char	Num
CONTAINS	Includes a substring	✓	✓
BETWEEN-AND	Defines an inclusive range	✓	✓
WHERE SAME AND	Augments a WHERE expression	✓	✓
IS NULL	Identifies missing values	✓	✓
IS MISSING	Identifies missing values	✓	✓
LIKE	Matches a pattern	✓	

LABEL Statement

- The **LABEL** statement is used to assign **descriptive labels** to variables.
    - Labels can be up to **256 characters** and can include any characters (even blanks).
    - **Labels** are used automatically by many procedures.
    - Example:

```
LABEL Job_Title = 'Job Position';
```

- In **PROC PRINT**, labels are used **only** when the **LABEL** or **SPLIT=** option is specified.

SPLIT= Option

- The **SPLIT=** option in **PROC PRINT** specifies a **split character** to control **line breaks** in column headings.
    - This can be used instead of the **LABEL** option in **PROC PRINT**.

```
PROC PRINT DATA=your_data SPLIT='-';

RUN;
```

What Is a Format in SAS?

- A **format** is an instruction that modifies how data values are displayed in a report.

- The **values stored in the data set are not changed**, only how they are presented in reports is affected.
- Formats are used to control the **appearance** of a variable's value in a report.

Structure of a SAS Format

SAS formats are structured as follows:

- **$**: Indicates a **character format**.
- **format**: Specifies the **name of the SAS format**.
- **w**: Specifies the total **format width**, including decimal places and special characters.
- **.**: Required syntax. The format always contains a period as part of the name.
- **d**: Specifies the number of **decimal places** to display in numeric formats.

Raw Data Files (Flat Files)

- **Raw Data File** (also called a **flat file**) is a text file that contains one record per line.
  - Each record typically has **multiple fields**.
  - **Flat files do not have internal metadata**, so **external documentation** (a record layout) is needed.
  - The **record layout** describes the fields and their positions within each record.

Types of Fields in Raw Data Files

- Fields in raw data files can be **delimited** or **arranged in fixed columns**.
  - **Delimited** fields are separated by characters like spaces, commas, or tabs.
  - **Fixed columns** mean that fields are aligned at fixed positions within each record.

Reading Raw Data Files in SAS

- SAS provides different **input styles** to read raw data files:
  1. **Column Input**: Used for reading standard data in fixed columns.
  2. **List Input**: Used for reading standard and nonstandard data separated by blanks or delimiters.

Standard vs. Nonstandard Data

- **Standard Data**:
  - Data that SAS can read without extra instructions.
  - Examples of **standard numeric data**: 58, 67.23, 5.67E5-23, 1.2E-2.

    - **Character data** is always considered **standard**.

- **Nonstandard Data**:
    - Data that requires extra instructions or informats to be correctly read by SAS.
    - Examples of **nonstandard numeric data**: $67.23, 01/12/2010, 12May2009.

List Input

- **List input** is used for reading delimited raw data files:
    - **Space** is the default delimiter.
    - Both **standard and nonstandard data** can be read.
    - Fields must be read **sequentially**, from left to right.

Data Errors in SAS

- **Data errors** occur when a data value does not match the field specification:
    - SAS logs a **note** about the error, along with:
        - A **column ruler**.
        - The **input record**.
        - The contents of the **PDV** (Program Data Vector).
    - If there is an error, a **missing value** is assigned to the corresponding variable, and execution continues.
- **Temporary variables** created during DATA step processing:
    - ***N***: Iteration counter.
    - ***ERROR***: Indicates if a data error occurred (0 = no error, 1 = error).

Modified List Input

- In **modified list input**, **informats** can be used to define the length of character variables, instead of using a **LENGTH** statement.
    - For example, the **$12. informat** defines a length of 12 for the **First_Name** field, enabling SAS to read up to 12 characters.
    - The **colon modifier** (e.g., **:**) tells SAS to read until it encounters a delimiter.
- **Omitting the colon modifier** could cause unexpected results, as SAS may not properly read the data fields.

Reading Nonstandard Data: needs special **informats** to be read correctly.

SAS Informats

- **Selected SAS Informats**:
    - **COMMA.**: Removes commas and other non-numeric characters from numeric data.
    - **DOLLAR.**: Reads numeric data with a dollar sign and removes non-numeric characters.
    - **COMMAX.**: Reads numeric data with commas as thousand separators.
    - **DOLLARX.**: Similar to **DOLLAR.**, but treats commas and periods as non-numeric characters.
    - **EUROX.**: Reads numeric data with the **euro symbol (€)**.
    - **$CHAR.**: Reads character values and preserves leading blanks.
    - **$UPCASE.**: Reads character values and converts them to uppercase.

Date Informats

- SAS provides **date informats** to read and convert dates to **SAS date values**:
    - **MMDDYY.**: Reads dates in MMDDYY format (e.g., 01/01/60).
    - **DDMMYY.**: Reads dates in DDMMYY format (e.g., 31/12/60).
    - **DATE.**: Reads dates in SAS date format (e.g., 31DEC59).

WHERE vs. Subsetting IF Statement

- **WHERE** and **IF** statements differ in how they are used in SAS:
    - **WHERE**:
        - Used in **PROC steps** (such as **PROC SQL**).
        - Cannot be used with **SET** statements in **DATA steps**.
    - **IF**:
        - Used in **DATA steps** (after **SET** statements).
        - Can be used for conditional subsetting in **DATA steps**.

Handling Missing Values

1. **Consecutive Delimiters in List Input**:
    - SAS treats two or more consecutive delimiters as a **single delimiter**, not as a missing value.
    - If a field is missing, SAS proceeds with the next record.
2. **DSD Option**:
    - The **DSD** option:

- Sets the default delimiter to a **comma**.
- Treats consecutive delimiters as **missing values**.
- Enables reading values with embedded delimiters when enclosed in **quotation marks**.

3. **MISSOVER Option**:
    - The **MISSOVER** option prevents SAS from loading a new record when the end of the current record is reached.
    - If SAS reaches the end of a record without finding all field values, variables without values are set to **missing**.

Options for Reading Raw Data Files

- **DLM=**: Specifies an alternate delimiter (e.g., tab, comma).
- **DSD**: Makes SAS handle missing values and embedded delimiters correctly.
- **MISSOVER**: Ensures variables are set to missing if the end of a record is reached early.

DROP Statement -2.6

- The **DROP statement** specifies the **variables to exclude** from the output data set.
    - This is useful when you want to remove certain variables from the output without altering the original data set.

KEEP Statement

- The **KEEP statement** specifies the **variables to include** in the output data set.
    - This is useful when you want to retain only specific variables` from the data set, discarding the rest.

DATA Step Processing Phases

SAS processes a **DATA step** in two main phases:

1. **Compilation Phase**
2. **Execution Phase**

Compilation Phase

- **Compilation Phase** is the first phase when SAS processes a DATA step.
    - **Scans the program for syntax errors**: SAS checks the code for any syntax issues before running it.
    - **Translates the program into machine language**: The SAS code is converted into a format the computer can execute.

- **Creates the Program Data Vector (PDV)**: The PDV holds one observation at a time, which contains the data values.
- **Creates the descriptor portion of the output data set**: The descriptor holds metadata like variable names, types, and lengths.

Example: In the compilation phase, SAS prepares everything for processing, ensuring all necessary components (like the PDV) are set up before execution.

Execution Phase

- **Execution Phase** follows the compilation phase and is where the actual processing happens:
  - **Initialize PDV to missing**: The variables in the PDV are initialized to missing values.
  - **Execute SET statement**: If a SET statement is used, SAS reads data from the input dataset and loads it into the PDV.
  - **Execute other statements**: Other statements like assignments or conditional logic (e.g., IF statements) are executed.
  - **Output to SAS data set**: Finally, the processed data is written to the output data set.

Explicit Output – 2.7

- The **explicit OUTPUT statement** writes the contents of the **Program Data Vector (PDV)** to the data set(s) being created.
  - This is used when you want to force an output for the current observation at a specific point in the DATA step.
  - The **explicit OUTPUT** overrides the **implicit output** (which happens automatically at the end of the DATA step).

Example:

```
data new_data;
 set old_data;
 if salary > 50000 then do;
  bonus = salary * 0.1;
  output;  /* Explicitly writes this observation to the output dataset */
 end;
run;
```

Creating Multiple SAS Data Sets

- You can create **multiple SAS data sets** within a single DATA step by specifying more than one output data set name in the DATA statement.
    - Each observation will be written to the appropriate data set based on the conditions you define within the DATA step.

Example:

```
data usa australia other;
 set global_data;
 if country = 'USA' then output usa;
 else if country = 'Australia' then output australia;
 else output other;
run;
```

Displaying Multiple SAS Data Sets

- The **PRINT procedure (PROC PRINT)** can only print one data set at a time.
    - If you want to print multiple data sets, you need to use a separate **PROC PRINT** step for each data set.

FREQ Procedure – 2.8

- **Purpose**: The **FREQ** procedure produces a one-way frequency table for each variable listed in the **TABLES** statement.
    - If no **TABLES** statement is provided, a one-way frequency table is generated for **every variable** in the data set, which can create a large amount of output.

Example:

```
proc freq data=mydata;
 tables gender;  /* One-way frequency table for 'gender' */
run;
```

Example output:

Gender	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Male	200	60.0%	200	60.0%
Female	133	40.0%	333	100.0%

Options to Suppress Statistics

- You can use options in the **TABLES** statement to suppress certain statistics.

- **NOCUM**: Suppresses cumulative statistics.
- **NOPERCENT**: Suppresses percentage display.

Crosstabulation Table

- A **crosstabulation table** (or two-way frequency table) is generated by placing an asterisk between two variables in the **TABLES** statement.
  - This table shows the relationship between two categorical variables.

Example:

```
proc freq data=orion.sales;
 tables gender*country;  /* Two-way frequency table for gender and country */
run;
```

Output:

Gender USA Canada Mexico

Male 50 30 20

Female 60 20 15

LIST and CROSSLIST Options

- Use the **LIST** and **CROSSLIST** options in the **TABLES** statement to flatten the output:
  - **LIST**: Presents the frequency table in a simple, list format.
  - **CROSSLIST**: Shows cross-tabulated data in a list format.

NLEVELS Option

- The **NLEVELS** option provides a table that shows the number of distinct values for each analysis variable.

Example:

```
proc freq data=mydata nlevels;
 tables gender;  /* Displays the number of distinct values for 'gender' */
run;
```

Check for Uniqueness

- Use **PROC FREQ** to check for duplicates or missing values in variables (e.g., **Employee_ID**).
  - **ORDER=FREQ**: Displays results in descending frequency order, which can help identify duplicate or missing values.

Example:

```
proc freq data=mydata order=freq;
 tables employee_id;  /* Check for duplicates and missing values in 'Employee_ID' */
run;
```

MEANS Procedure

- The **MEANS** procedure produces summary reports with descriptive statistics for numeric variables.
    - **Analysis variables**: Numeric variables for which statistics are calculated.
    - **Classification variables**: Define subgroups for analysis (can be either numeric or character).

CLASS Statement

- The **CLASS** statement identifies **classification variables** that define subgroups for analysis.
    - **Classification variables** typically have a small number of distinct values (e.g., gender, country).

Example:

```
proc means data=mydata;
 class gender;  /* Gender is the classification variable */
 var salary;    /* Salary is the analysis variable */
run;
```

PROC MEANS Statement Options

- The **PROC MEANS** statement allows you to request specific statistics.
    - **MAXDEC=**: Specifies the number of decimal places for the statistics.
    - **NONOBS**: Suppresses the "N Obs" column.

Example:

```
proc means data=mydata maxdec=2 nonobs;
 var salary;
run;
```

Other PROC MEANS Statistics

- **Descriptive Statistics Keywords**:

- **MEAN**: Mean of the variable.
- **MIN**: Minimum value.
- **MAX**: Maximum value.
- **STDDEV**: Standard deviation.
- **CV**: Coefficient of variation.
- **KURTOSIS**: Kurtosis.
- **SKEWNESS**: Skewness.
- **SUM**: Sum of values.
- **N**: Number of non-missing observations.
- **NMISS**: Number of missing values.

Quantile Statistics:

- **P50**: Median (50th percentile).
- **P25**: 25th percentile (Q1).
- **P75**: 75th percentile (Q3).

Example:

```
proc means data=mydata mean min max stddev;
 var salary;
run;
```

UNIVARIATE Procedure

- **PROC UNIVARIATE** displays extreme observations, missing values, and other statistics for the variables in the **VAR** statement.
  - If the **VAR** statement is omitted, **PROC UNIVARIATE** analyzes all numeric variables in the data set.
  - **Extreme Observations**: Shows the five lowest and five highest values for the variable.

Example:

```
proc univariate data=mydata;
 var salary;  /* Analyzes 'salary' variable */
run;
```

Output includes:

- Extreme values.
- Descriptive statistics such as mean, standard deviation, and percentiles.

RENAME= Data Set Option – 2.8

- **Purpose**: The **RENAME=** data set option is used to change the name of a variable.
    - The **RENAME=** option must be specified immediately after the SAS data set name, within parentheses.
    - The name change affects:
        - **Program Data Vector (PDV)**.
        - **Output data set**.
    - It does **not affect** the input data set.

Example:

```
data newdata;
 set olddata(rename=(oldvar=newvar));
run;
```

Match-Merging

- **Purpose**: **Match-merging** is the process of combining observations from two or more data sets based on one or more common variables.

Types of Match-Merging

1. **One-to-One**:
    - A single observation in one data set matches exactly one observation in another data set.
    - The match is based on the values of one or more selected variables.

Example:

- **Data set 1**: Employee ID and Salary.
- **Data set 2**: Employee ID and Job Title.

2. **One-to-Many**:
    - A single observation in one data set is related to more than one observation in another data set.
    - For example, one employee record in the first data set matches multiple records in the second data set.

Example:

- **Data set 1**: Employee ID and Department.
- **Data set 2**: Employee ID and Phone Number (multiple phone numbers per employee).

3. **Nonmatches**:
   - At least one observation in one data set is not related to any observation in the other data set based on the selected variables.
   - Nonmatching observations are included in the output data set with missing values for the variables that do not match.

Example:

- **Data set 1**: Employee ID and Salary.
- **Data set 2**: Employee ID and Job Title.
- Some employee IDs in **Data set 1** may not have corresponding records in **Data set 2**, and vice versa.

Match-Merging: Sorting the Data Sets

- **Sorting Requirement**: For a successful match-merge, both data sets must be sorted by the variables you want to use for matching.
  - Sorting ensures that SAS can properly align matching observations from both data sets.

Example:

```
proc sort data=empsau; by EmpID; run;

proc sort data=phonec; by EmpID; run;
```

MERGE and BY Statements

- **MERGE Statement**: The **MERGE** statement is used to combine observations from two or more SAS data sets into a single observation.
  - **BY Statement**: A **BY** statement is used to indicate which variables to match on and to specify the variables used to merge the data sets.
  - **Requirements** for match-merging:
    - Two or more data sets are listed in the **MERGE** statement.
    - The variables in the **BY** statement must be common to all data sets.
    - All data sets must be sorted by the variables in the **BY** statement.

Example:

```
data merged_data;
 merge empsau(in=a) phonec(in=b);
 by EmpID;
run;
```

- This merges **empsau** and **phonec** based on the common variable **EmpID**.

One-to-One Merge Example

```
data empsauc;
 merge empsau(in=a) phonec(in=b);
 by EmpID;
run;
```

Merge with Nonmatches

- If there are **nonmatches** (i.e., some observations in one data set do not have corresponding matches in another data set), SAS will keep them in the merged data set, and the variables from the data set that does not contribute to the observation will be set to missing.

Example:

```
data empsauc;
 merge empsau(in=a) phonec(in=b);
 by EmpID;
run;
```

IN= Data Set Option

- **Purpose**: The **IN=** data set option creates a temporary variable that indicates whether a particular data set contributed to building the current observation.
  - The variable created by **IN=** is a temporary numeric variable with two possible values:
    - **1**: The data set **contributed** to the current observation.
    - **0**: The data set **did not contribute** to the current observation.

Example:

```
data merged_data;
 merge empsau(in=a) phonec(in=b);
```

```
 by EmpID;
 if a and b;  /* Only include records that appear in both data sets */
run;
```

- The **IN=** option creates temporary variables **a** and **b** to track whether records in **empsau** and **phonec** were included in the merged observation.