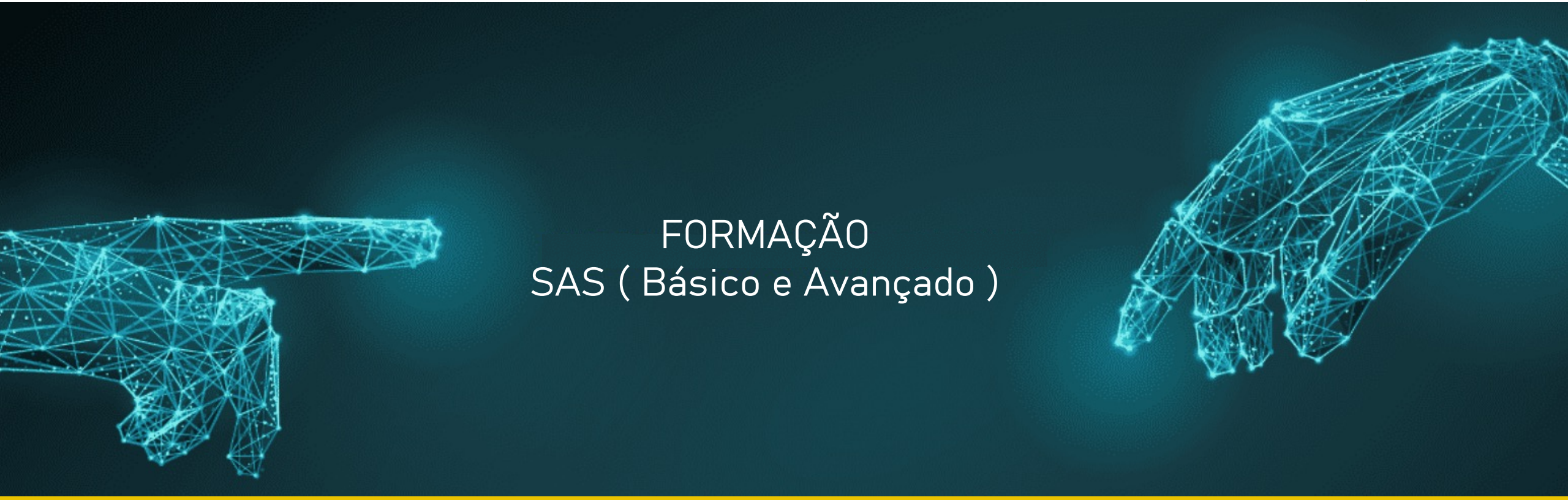
HabberTec
spirit of innovation

Cofidis



FORMAÇÃO
SAS (Básico e Avançado)

Nosso sonho é mudar o mundo através da tecnologia.

Confidencial

Este documento é propriedade intelectual de Habber Tec e não pode ser alterado ou usado para qualquer outro propósito que não seja acordado anteriormente e sem a permissão de Habber Tec.

1. Apresentação Instrutor
2. Apresentação Alunos
3. Orientações da formação
4. Conteúdo Programático
5. Casos de Estudo
6. Agradecimento

William Rosario – Consultor

Profissional com mais de 22 anos de experiência no setor de Tecnologia da Informação, atuando em projetos de grande porte e alta complexidade nos setores financeiro, varejo, telecomunicações, saúde, educação, setor público e indústria. Minha trajetória inclui passagens por empresas de destaque em Portugal, Moçambique e Brasil, sempre com foco em soluções de Business Intelligence, Data Warehousing, integração de dados e visualização analítica.

Tecnologias:

- SAS: DIS, Guide, Visual Analytics, Administrator, Viya, Decision Manager, Visual Investigator, Visual Forecasting, Visual Text Analytics, Visual Statistics, Model Manager, Intelligent Decisioning, Compliance Solutions, Fraud Framework, Risk Modeling, entre outras.
- MicroStrategy: Desenvolvimento de relatórios, design de projetos, administração de plataforma e engenharia de projetos.
- Microsoft: SQL Server, SSIS, Power BI.
- Oracle: Banco de dados Oracle.
- Big Data e Cloud: Hive, Parquet, Airflow, Dremio, Git, Amazon EMR, Amazon S3 Buckets, AWS.

William Rosario - Consultor

Formação Acadêmica

- Bacharelado em Sistemas de Informação – UNIP
- Especialização em Data Warehousing – PUC-RJ

Certificações

- SAS:
  - SAS Visual Analytics Exploration and Design
- MicroStrategy:
  - Certified Engineer (MCE) Project
  - Certified Platform Administrator
  - Certified Project Designer
  - Certified Report Developer

Apresentação Alunos
(Nome, Função e Expectativa)

Orientações

- Cada formando deverá ter um portátil com acesso a internet e cadastrado no portal da SAS® OnDemand for Academics (https://odamid.oda.sas.com).
- Os formandos deverão ter acesso e instalado o software TEAMS e permanecer com a webcam ligada durante as sessões .
- O horário da formação será de 09:30 às 13:00 (Matutino) e 14:30 às 18:00 (Vespertino).
- Intervalo será de 15 minutos às 11:15 (Matutino) e às 16:15 (Vespertino).
- Caso tenham alguma pergunta favor sinalizar, levantando a mão.

Conteúdo Programático

BÁSICO

1. DATA Step vs. PROC SQL
2. Funções do SAS
3. Manipulação Avançada de Dados
4. Importação de Dados
5. Conexão com Bancos de Dados

AVANÇADO

1. Conexão com Bancos de Dados
2. Visualização e Relatórios
3. Performance e Eficiência
4. Aprender SAS Macros
5. Manipulação de Grandes Volumes de Dados
6. Integração com Ferramentas Analíticas

Quem conhece o SAS?
Quais ferramentas utiliza?

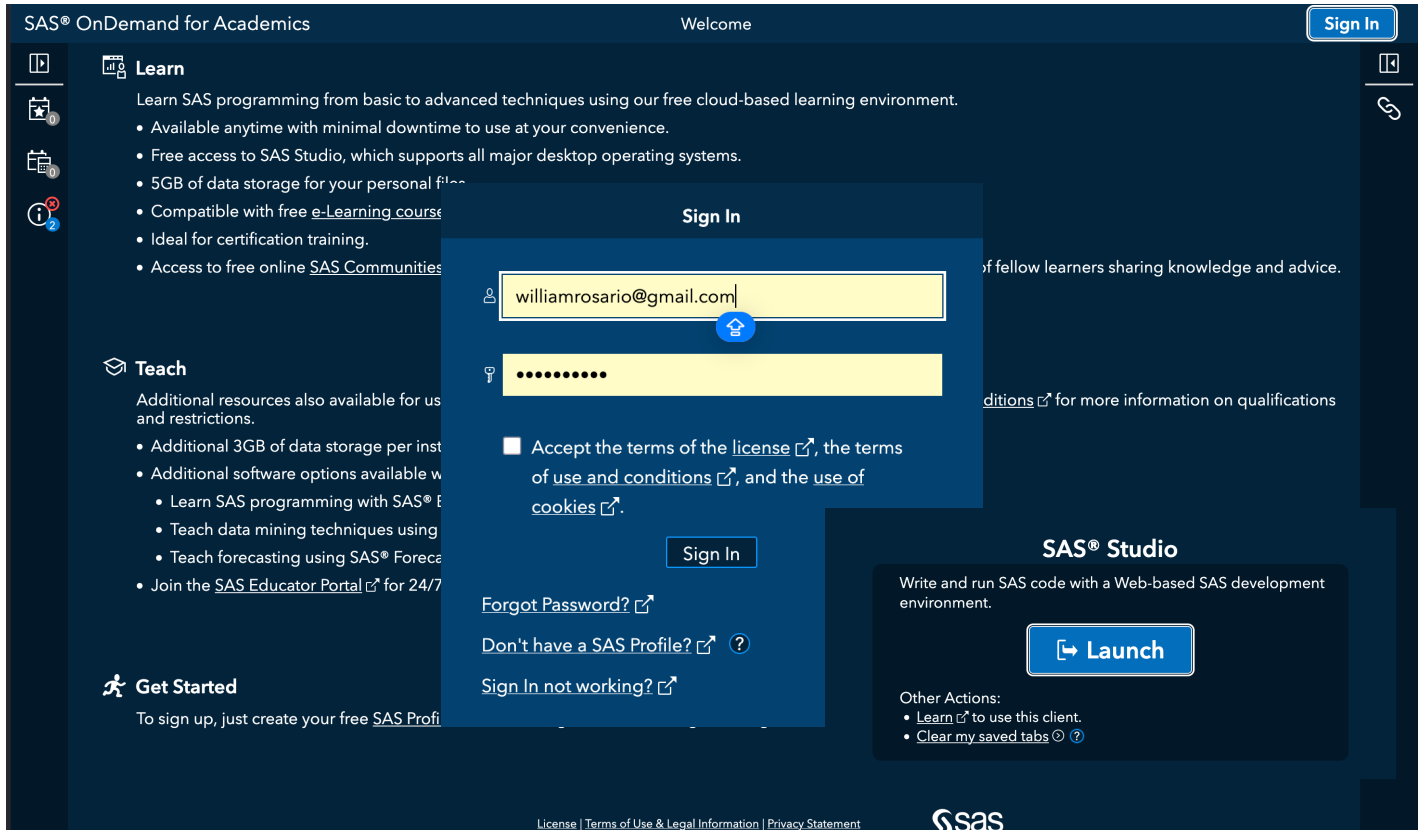
O que é SAS?

O SAS (Statistical Analysis System) é uma suíte de softwares desenvolvida para análise estatística, mineração de dados, gerenciamento de bancos de dados, geração de relatórios e gráficos, entre outras funções relacionadas ao processamento e análise de grandes volumes de dados. Criado inicialmente para pesquisas agrícolas, o SAS tornou-se uma das ferramentas mais utilizadas em ambientes corporativos, acadêmicos e governamentais para transformar dados brutos em informações valiosas para a tomada de decisão.

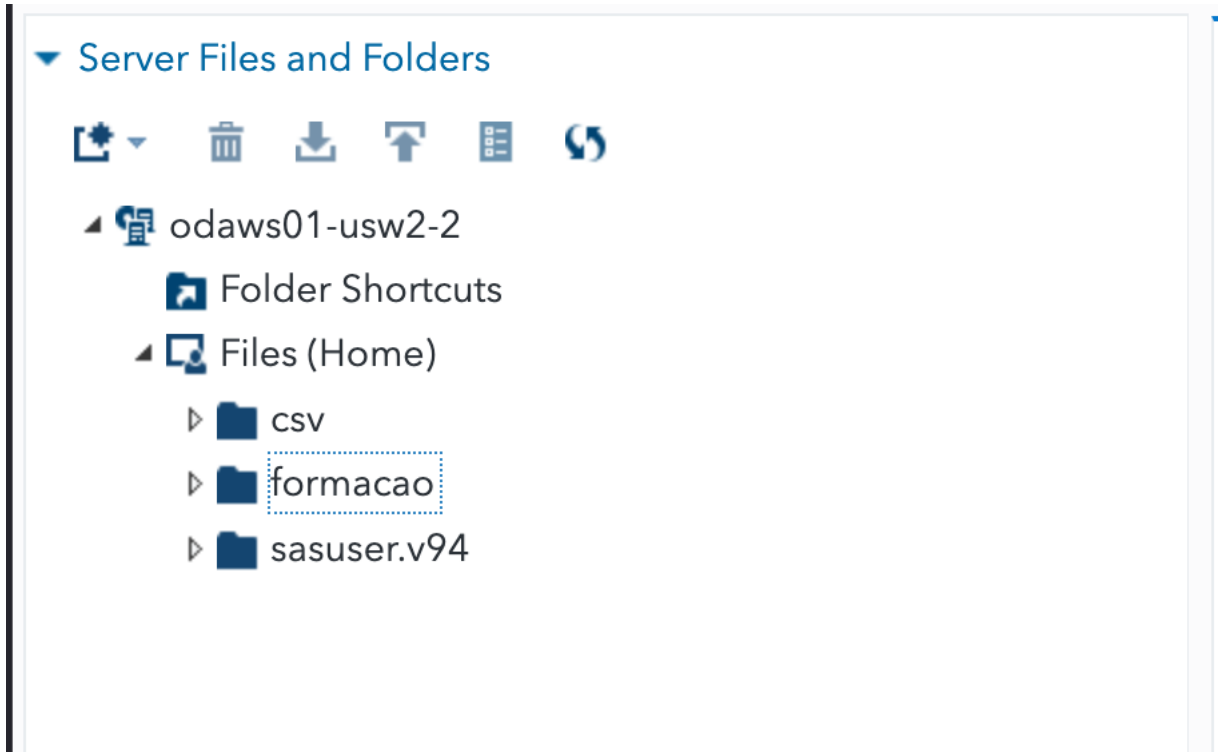
O SAS permite acessar, manipular, analisar e apresentar dados de praticamente qualquer formato, sendo reconhecido por sua portabilidade e integração com diferentes sistemas operacionais e bancos de dados.

Ele é amplamente utilizado por cientistas de dados, estatísticos e profissionais de diversas áreas que precisam realizar análises complexas, cálculos estatísticos, modelagens e visualizações de dados de maneira eficiente e confiável.

Login na plataforma SAS® OnDemand for Academics
https://odamid.oda.sas.com



Criar diretorias “tabelas”,“formacao” e “csv”



Libname

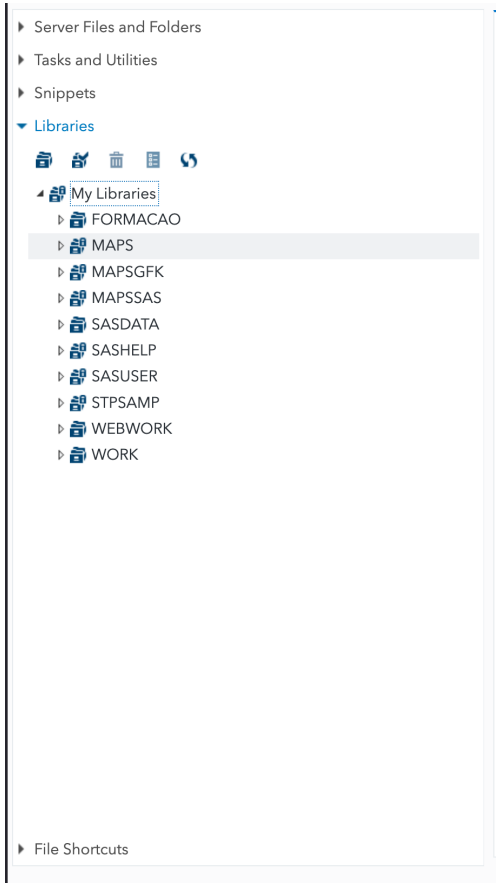
LIBNAME no SAS serve para criar uma referência a um local de armazenamento de arquivos de dados, chamado de biblioteca (library). Essa biblioteca pode ser uma pasta no seu computador, um diretório em rede, ou até uma conexão com banco de dados. Ao definir uma biblioteca com LIBNAME, você facilita o acesso e a organização dos arquivos SAS, pois pode referenciá-los pelo nome da biblioteca ao invés do caminho completo

```
LIBNAME nome_biblioteca 'caminho_do_diretorio';     LIBNAME tabelas '/home/u40916726/tabelas';
```

O que é a biblioteca WORK?

A WORK é uma biblioteca temporária padrão do SAS. Sempre que você não especifica uma biblioteca ao criar um dataset, o SAS armazena esse arquivo automaticamente na WORK. Os dados da WORK existem apenas durante a sessão atual do SAS: ao fechar o SAS, tudo que estava na WORK é apagado.

Criar Libname “FORMACAO”



DATA STEP

O DATA step é uma das principais estruturas de programação no SAS, usada para criar, importar, modificar, combinar ou calcular dados. Ele permite transformar dados brutos em datasets SAS, manipular variáveis, filtrar registros, criar novas variáveis.

```
data copia;
   set original;
run;
```

DATA
A palavra-chave `DATA` inicia um bloco de código chamado data step. Ela define o nome do dataset que será criado ou modificado.

SET
O comando `SET` é utilizado dentro do data step para ler observações de um ou mais datasets SAS já existentes. Ele carrega os dados linha a linha para dentro do data step, permitindo processamento sequencial de cada registro

RUN
Executa as instruções submetidas até aquele ponto e inicia a execução do passo (step) atual.

DATA STEP

O data step é extremamente versátil. Entre suas principais aplicações estão:

- Copiar datasets: criar uma cópia de um dataset existente.
- Concatenar datasets: ler vários datasets em sequência, combinando todos os registros em um novo dataset.
- Modificar datasets: aplicar transformações, criar novas variáveis ou filtrar registros durante a leitura dos dados.
- Aplicar opções: como `keep=`, `drop=`, `rename=` para manipular variáveis durante a leitura.
- Criar novos datasets a partir de dados brutos ou outros datasets SAS.
- Importar dados de arquivos texto ou externos.
- Criar, modificar ou excluir variáveis.
- Recodificar valores de variáveis.
- Filtrar (subsetting) registros.
- Concatenar ou mesclar datasets.
- Calcular variáveis derivadas.
- Gerar relatórios simples ou exportar dados para arquivos externos.

PROC SQL

O PROC SQL é uma das principais procedures do SAS, permitindo ao usuário utilizar a linguagem SQL (Structured Query Language) para manipulação, consulta e transformação de dados dentro do ambiente SAS. Ele oferece uma alternativa poderosa ao tradicional DATA step, sendo especialmente útil para quem já tem familiaridade com SQL em bancos de dados relacionais.

```
proc sql;
   select * from tabela;
quit;
```

Entre suas principais aplicações estão:

- SELECT: Seleciona colunas e registros de uma tabela.
- CREATE TABLE ... AS SELECT: Cria uma nova tabela SAS a partir de uma seleção.
- WHERE: Filtra registros de acordo com condições específicas.
- GROUP BY: Agrupa registros para cálculos agregados (SUM, AVG, COUNT, etc.).
- HAVING: Filtra grupos após o agrupamento.
- ORDER BY: Ordena os resultados.
- JOINS: Realiza junções entre tabelas (INNER, LEFT, RIGHT, FULL).

DATA Step vs. PROC SQL

Diferenças e Casos de Uso

- DATA STEP: Melhor para manipulação sequencial de dados, criação de variáveis e leitura linha por linha.
- PROC SQL: Mais eficiente para operações relacionais, como joins e agregações.

SAS DATA STEP

```
proc sort data=formacao.empregados
out=empregados_ordenado;
  by idade;
run;
data contagem_idade;
  set empregados_ordenado;
  by idade;
  if first.idade then quantidade = 0;
  quantidade + 1;
  if last.idade then output;
  keep idade quantidade;
run;
```

SQL

```
proc sql;
  select idade, count(*) as quantidade
  from formacao.empregados
  group by idade;
quit;
```

DATA Step vs. PROC SQL

Critério	PROC SQL	DATA Step
Paradigma	Declarativo (SQL Blocos)	Procedural (passo a passo, linha a linha)
Foco principal	Operações relacionais: joins, subconsultas, agregações	Manipulação de dados linha a linha, criação de variáveis, controle de fluxo
Performance em joins	Melhor para múltiplos joins complexos	Pode ser mais lento em joins com grandes volumes
Performance em loops/lógica	Não tão eficiente para lógica sequencial ou loops	Muito eficiente em lógica de linha e loops (IF, DO, RETAIN, etc.)
Leitura linha a linha	Não possui	Sim – processa linha a linha, ideal para regras condicionais complexas
Criação de variáveis	Apenas no SELECT, geralmente com funções agregadas	Flexível – pode criar variáveis em qualquer ponto do DATA Step
Retenção de valores	Não retém valor entre linhas (não tem RETAIN)	Sim – com RETAIN, pode acumular valores entre iterações
Joins	INNER, LEFT, RIGHT, FULL OUTER – padrão SQL	Com MERGE, requer datasets ordenados por BY
Agregações	Simples com GROUP BY, HAVING	Manual – requer BY + FIRST. e LAST.
Manipulação avançada	Limitada (sem loop DO, sem RETAIN)	Muito poderosa – com lógica de controle (IF, DO, RETAIN, ARRAY, etc.)
Criação de tabelas	CREATE TABLE	Implicitamente cria dataset ao final do step
Leitura de várias fontes	Suporte direto via CONNECT TO e Pass-Through SQL	Precisa de LIBNAMEs ou INFILE/INPUT para ler fontes externas
Macros	Pode usar, mas mais limitado	Integra muito bem com CALL SYMPUT, CALL EXECUTE etc.
Uso típico	Relatórios rápidos, extrações complexas de várias tabelas	Transformações linha a linha, limpeza e preparação de dados
Sintaxe	SQL	Estilo SAS procedural

DATA Step vs. PROC SQL

Quando Usar?	Melhor opção
Precisa de joins complexos entre várias tabelas?	PROC SQL
Precisa criar colunas com lógica condicional entre linhas?	DATA Step
Quer performance em transformações linha a linha (como flags, contadores)?	DATA Step
Precisa de agrupamentos com média, soma, contagem?	Ambos (mas PROC SQL é mais direto)
Vai importar dados externos de um banco de dados?	PROC SQL com Pass-Through

Funções de Manipulação de Strings (Texto)

Função	Descrição	Exemplo
SUBSTR()	Extrai parte de uma string	SUBSTR(nome, 1, 4) → “Mari” de “Mariana”
SCAN()	Retorna uma palavra específica de uma string	SCAN("João da Silva", 2) → “da”
INDEX()	Retorna a posição de uma substring	INDEX("teclado", "cla") → 3
TRIM()	Remove espaços à direita	TRIM("abc ") → "abc"
LEFT()	Alinha texto à esquerda	LEFT(" abc") → "abc "
UPCASE()	Converte para maiúsculas	UPCASE("abc") → "ABC"
LOWCASE()	Converte para minúsculas	LOWCASE("ABC") → "abc"
CATX()	Concatena com delimitador e ignora valores em branco	CATX('-', 'SP', '', '123') → "SP-123"
REVERSE()	Inverte a ordem dos caracteres de uma string.	REVERSE("Mariana") → “anairaM”
TRANWRD()	Substitui todas as ocorrências de uma substring por outra.	TRANWRD("Rua das Flores", "Rua", "Avenida") → “Avenida das Flores”
TRANSLATE()	Substitui caracteres específicos em uma string por outros caracteres, posição a posição.	TRANSLATE("João da Silva", "x", "a")

Funções de Manipulação de Strings (Texto)

Exercício 1 - SUBSTR

- Extrair o primeiro nome.
  SUBSTR(variavel, inicio, fim)

```
DATA exerc_substr;
  nome = ”William Rosario";
RUN;
```

```
DATA exerc_substr;
  nome = ”William Rosario";
  primeiro_nome = SUBSTR(nome, 1, 8);
  /* Esperado: ”William" */
RUN;
```

Funções de Manipulação de Strings (Texto)

Exercício 2 - SCAN

- Obter uma palavra específica da string.
  SCAN(variavel, posicao,”busca”)

```
DATA exerc_scan;
  endereco = "Rua das Palmeiras, 123 Centro";
RUN;
```

```
DATA exerc_scan;
  endereco = "Rua das Palmeiras, 123 Centro";
  rua = SCAN(endereco, 1, ',');
  /* Esperado: "Rua das Palmeiras" */
RUN;
```

Funções de Manipulação de Strings (Texto)

Exercício 3 – INDEX

- Encontrar a posição de uma substring.
  INDEX(variavel,”busca”)

```
DATA exerc_index;
 frase = "Entrega em São Paulo";
RUN;
```

```
DATA exerc_index;
 frase = "Entrega em São Paulo";
 posicao = INDEX(frase, "São");
 /* Esperado: posição inicial da palavra "São"  12*/
RUN;
```

Funções de Manipulação de Strings (Texto)

Exercício 4 - TRIM

- Remover espaços à direita.
  TRIM(variavel)

```
DATA exerc_trim;
  texto = ” SAS Studio  ";
RUN;
```

```
DATA exerc_trim;
   texto =  "SAS Studio ";
   ajustado = TRIM(texto);
  /* Esperado: "SAS Studio" (sem espaço extra) */
RUN;
```

Funções de Manipulação de Strings (Texto)

Exercício 5 - LEFT

- Alinhar string à esquerda.
  LEFT(variavel);

```
DATA exerc_left;
  texto = "   Formação SAS";
RUN;
```

```
DATA exerc_left;
  texto = "   Formação SAS";
  ajustado = LEFT(texto);
RUN;
```

Funções de Manipulação de Strings (Texto)

Exercício 6 - UPCASE

- Converter string para maiúsculas.
  UPCASE(variavel);

```
DATA exerc_upcase;
  nome = ”william rosario";
RUN;
```

```
DATA exerc_upcase;
  nome = ”william rosario";
  nome_maiusculo = UPCASE(nome);
RUN;
```

Funções de Manipulação de Strings (Texto)

Exercício 7 - LOWCASE

- Converter string para minúsculas.
  LOWCASE(variavel);

```
DATA exerc_lowcase;
  cargo = "GERENTE DE PROJETOS";
RUN;
```

```
DATA exerc_lowcase;
  cargo = "GERENTE DE PROJETOS";
  cargo_padronizado = LOWCASE(cargo);
RUN;
```

Funções de Manipulação de Strings (Texto)

Exercício 8 - CATX

- Concatenar strings com delimitador, ignorando valores em branco.
  CATX(“delimitador”,variavel1,variavel2);

```
DATA exerc_catx;
 estado = "SP";
 cidade = "";
 cep = "12345-678";
 /* Esperado: "SP - 12345-678" */
RUN;
```

```
DATA exerc_catx;
 estado = "SP";
 cidade = "";
 cep = "12345-678";
 endereco = CATX(" - ", estado, cidade, cep);
 /* Esperado: "SP - 12345-678" */
RUN;
```

Funções de Manipulação de Strings (Texto)

Exercício 9 - REVERSE

- Inverte a ordem dos caracteres de uma string.
  REVERSE(variavel);

```
data exerc_reverse;
   nome = "João da Silva";
run;
```

```
data exerc_reverse;
   nome = "João da Silva";
   nome_reverse= REVERSE(nome);
/* avliS ad oãoJ */
run;
```

Funções de Manipulação de Strings (Texto)

Exercício 10 - TRANWRD

- Substitui todas as ocorrências de uma substring por outra.
  TRANWRD(variavel,"parametro1", "parametro2");

```
DATA exerc_tranwrd;
  endereco = "Rua das Flores";
/* Esperado: "Avenida das Flores" */
RUN;
```

```
DATA exerc_tranwrd;
  endereco = "Rua das Flores";
  resultado = TRANWRD(endereco,"Rua","Avenida");
/* Esperado: "Avenida das Flores" */
RUN;
```

Funções de Manipulação de Strings (Texto)

Exercício 11 - TRANSLATE

- Substituir caracteres específicos em uma string por outros caracteres correspondentes
  TRANSLATE(texto,de,para);

```
data exerc_translate;
   nome = "João da Silva";
run;
```

```
data exerc_translate;
   nome = "João da Silva";
   nome_traduzido = translate(nome, "x", "a");
run;
 /* Esperado: "João dx Silvx" */
```

Funções Estatísticas e Matemáticas

Função	Descrição	Exemplo
MEAN()	Média aritmética	MEAN(10, 20, 30) → 20
STD()	Desvio padrão	STD(4, 6, 9)
SUM()	Soma dos valores	SUM(x, y, z)
ROUND()	Arredonda para casas decimais	ROUND(4.567, 0.1) → 4.6
INT()	Trunca para inteiro	INT(3.89) → 3
MOD()	Resto da divisão	MOD(7, 3) → 1
MAX()	Valor máximo	MAX(2, 9, 5) → 9
MIN()	Valor mínimo	MIN(2, 9, 5) → 2

Funções Estatísticas e Matemáticas

Exercício 12 - MEAN

- Calcular a média dos valores, ignorando valores faltantes.
  MEAN(variaveis);
  A=10 , B=. e C=30

```
DATA exerc_mean;
  a = 10;
  b = .;
  c = 30;
  media = MEAN(a, b, c);
  /* Esperado: 20 */
RUN;
```

Funções Estatísticas e Matemáticas

Exercício 13 - SUM

- Somar os valores, ignorando valores faltantes.
  SUM(variaveis);
  A=10 , B=. e C=30

```
DATA exerc_sum;
  a = 10;
  b = .;
  c = 30;
  total = SUM(a, b, c);
  /* Esperado: 40 */
RUN;
```

Funções Estatísticas e Matemáticas

Exercício 14 - STD

- Calcular o desvio padrão dos valores.
  STD(variaveis)
  A=10 , B=20 e C=30

```
DATA exerc_std;
  a = 10;
  b = 20;
  c = 30;
  total = STD(a, b, c);
  /* Esperado: 10 */
RUN;
```

Funções Estatísticas e Matemáticas

Exercício 15 - MIN

- Retornar o menor valor entre os argumentos.
  MIN(VARIAVEIS);
  A=10 , B=2 e C=30

```
DATA exerc_min;
  a = 10;
  b = 2;
  c = 30;
  minimo = MIN(a,b,c);
  /* Esperado: 2 */
RUN;
```

Funções Estatísticas e Matemáticas

Exercício 16 - MAX

- Retornar o maior valor entre os argumentos.
  MAX(variaveis);
  A=10 , B=2 e C=30

```
DATA exerc_max;
  a = 10;
  b = 2;
  c = 20;
  maximo = max(a,b,c);
  /* Esperado: 20 */
RUN;
```

Funções Estatísticas e Matemáticas

Exercício 17 - ROUND

- Arredondar o valor para o múltiplo especificado.
  ROUND(variavel, casas decimais);
  A=10.23 | Round = 0,1

```
DATA exerc_round;
  a = 10.23;
  arred = ROUND(a, 0.1);
  /* Esperado: 10.2 */
RUN;
```

Funções Estatísticas e Matemáticas

Exercício 18 - INT

- Retornar a parte inteira de um número.
  INT(variavel);
  A=10.23

```
DATA exerc_int;
  a = 10.23;
  inteiro = int(a);
  /* Esperado: 10.*/
RUN;
```

Funções de Data e Hora

Função	Descrição	Exemplo
TODAY()	Retorna a data atual	TODAY() → 09JUN2025
DATEPART()	Extrai a parte de data de um datetime	DATEPART('01JAN2024:13:20'dt) → 01JAN2024
TIMEPART()	Extrai a parte de hora de um datetime	TIMEPART('01JAN2024:13:20’dt) -> 13:20:00
INTCK()	Conta intervalos entre datas	INTCK('month', '01JAN2021'd, '01JAN2022'd) → 12
INTNX()	Avança ou recua datas por intervalo	INTNX('month', '01JAN2022'd, 1) → 01FEB2022
YEAR()	Extrai o ano	YEAR('15DEC2023'd) → 2023
MONTH()	Extrai o mês	MONTH('15DEC2023'd) → 12
DAY()	Extrai o dia	DAY('15DEC2023'd) → 15

Funções de Data e Hora

Exercício 19 - TODAY

- Retorna a data atual.
  TODAY();

```
DATA exerc_today;
  data_atual = TODAY();
  FORMAT data_atual date9.;
RUN;
```

Funções de Data e Hora

Exercício 20 - DATEPART

- Extrai a parte de data de um datetime.
  DATEPART();

```
DATA exerc_datepart;
  dt = '09JUN2025:15:30:00'dt;
  data_somente = DATEPART(dt);
  FORMAT data_somente date9.;
  /* Esperado: 09JUN2025 */
RUN;
```

Funções de Data e Hora

Exercício 21 - TIMEPART

- Extrai a parte de data de um datetime.
  TIMEPART();

```
DATA exerc_timepart;
  dt = '09JUN2025:15:30:00'dt;
  hora_somente = TIMEPART(dt);
  FORMAT hora_somente time8.;
  /* Esperado: 15:30:00 */
RUN;
```

Funções de Data e Hora

Exercício 22 - INTCK

- Conta intervalos entre datas.
  INTCK();

```
DATA exerc_intck;
  dias = INTCK('day', '01JAN2023'd, '10JAN2023'd);
  /* Esperado: 9 */
RUN;
```

Funções de Data e Hora

Exercício 23 - INTNX

- Avança ou recua datas por interval.
  INTNX();

```
DATA exerc_intnx;
  nova_data = INTNX('month', '01JAN2023'd, 2);
  FORMAT nova_data date9.;
  /* Esperado: 01MAR2023 */
RUN;
```

Funções de Data e Hora

Exercício 24 - YEAR

- Extrai o ano.
  YEAR ();

```
DATA exerc_year;
  D = '09JUN2025'd;
  ANO = YEAR(D);
  /* Esperado: 2025 */
RUN;
```

Funções de Data e Hora

Exercício 25 - MONTH

- Extrai o mês.
  MONTH();

```
DATA exerc_month;
  D = '09JUN2025'd;
  MES = MONTH(D);
  /* Esperado: 6 */
RUN;
```

Funções de Data e Hora

Exercício 26 - DAY

- Extrai o dia.
  DAY();

```
DATA exerc_day;
  D = '09JUN2025'd;
  DIA = day(D);
  /* Esperado: 9 */
RUN;
```

Conversão de Tipos (Numérico/Texto/Data)

Função	Descrição	Exemplo
PUT()	Converte valor numérico para texto (com formato)	PUT(1000, 8.) → " 1000"
INPUT()	Converte texto para valor numérico ou data	INPUT("20250101", yymmdd8.) → 01JAN2025

Function Call	Raw Type	Raw Value	Returned Type	Returned Value
A PUT(name, $10.);	**char, char format**	'Richard'	**char always**	'Richard '
B PUT(age, 4.);	num, num format	30	**char always**	' 30'
C PUT(name, $nickname.);	**char, char format**	'Richard'	**char always**	'Rick'
D INPUT(agechar, 4.);	**char always**	'30'	num, num informat	30
E INPUT(agechar, $4.);	**char always**	'30'	**char, char informat**	' 30'
F INPUT(cost,comma7.);	**char always**	'100,541'	num, num informat	100541

Conversão de Tipos (Numérico/Texto/Data)

Exercício 27 - PUT

- Converte valor numérico para texto (com formato).
  PUT(variavel,formato);

```
DATA exerc_put;
  num = 12345;
  txt = PUT(num, 5.);
  /* Esperado: "12345" */
RUN;
```

Conversão de Tipos (Numérico/Texto/Data)

Exercício 28 - INPUT

- Converte texto para valor numérico ou data.
  INPUT(variavel,formato);

```
DATA exerc_input;
  texto = "98765";
  numero = INPUT(texto, 5.);
  /* Esperado: 98765 */
RUN;
```

Extras

Função	Descrição	Exemplo
IFN()	Retorna valor baseado em condição (numérico)	IFN(x>10, "Alta", "Baixa")
IFC()	Igual ao IFN, mas para strings	IFC(x>0, "Positivo", "Negativo")
COALESCE()	Retorna o primeiro valor não-nulo	COALESCE(var1, var2)
COMPRESS()	Remove caracteres de uma string	COMPRESS("123-456", "-") → "123456"
MDY()	Cria uma data a partir de mês, dia e ano.	MDY(6, 15, 1990) -> 15JUN1990

Extras

Exercício 29 - IFN

- Retorna valor baseado em condição (numérico).
  IFN(condição,resultado1, resultado2);

```
DATA exerc_ifn;
  nota = 8;
  status = IFN(nota >= 7, 1, 0);
  /* Esperado: 1 */
RUN;
```

Extras

Exercício 30 - IFC

- Retorna valor baseado em condição (texto).
  IFC(condição,resultado1, resultado2);

```
DATA exerc_ifc;
  nota = 6.5;
  resultado = IFC(nota >= 7, "Aprovado", "Reprovado");
  /* Esperado: "Reprovado" */
RUN;
```

Extras

Exercício 31 - COALESCE

- Retorna o primeiro valor não-nulo
  COALESCE(variaveis);

```
DATA exerc_coalesce;
  a = .;
  b = 5;
  c = 10;
  resultado = COALESCE(a, b, c);
  /* Esperado: 5 */
RUN;
```

Extras

Exercício 32 - COMPRESS

- Remove caracteres de uma string
  COMPRESS(variaveis);

```
DATA exerc_compress;
  nome_bruto = "João da Silva";
  nome_limpo = COMPRESS(nome_bruto, " ");
  /* Esperado: "JoãodaSilva" — removeu os espaços */
RUN;
```

Extras

Exercício 33 - MDY

- Cria uma data a partir de mês, dia e ano.
  MDY(mes,dia,ano);

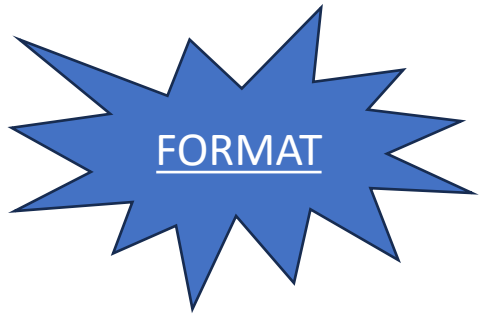
```
DATA exerc_mdy;
  mes = 6;
  dia = 9;
  ano = 2025;
  data_nascimento = MDY(mes, dia, ano);
  FORMAT data_nascimento DATE9.;
  /* Esperado: 09JUN2025 */
RUN;
```

FORMAT

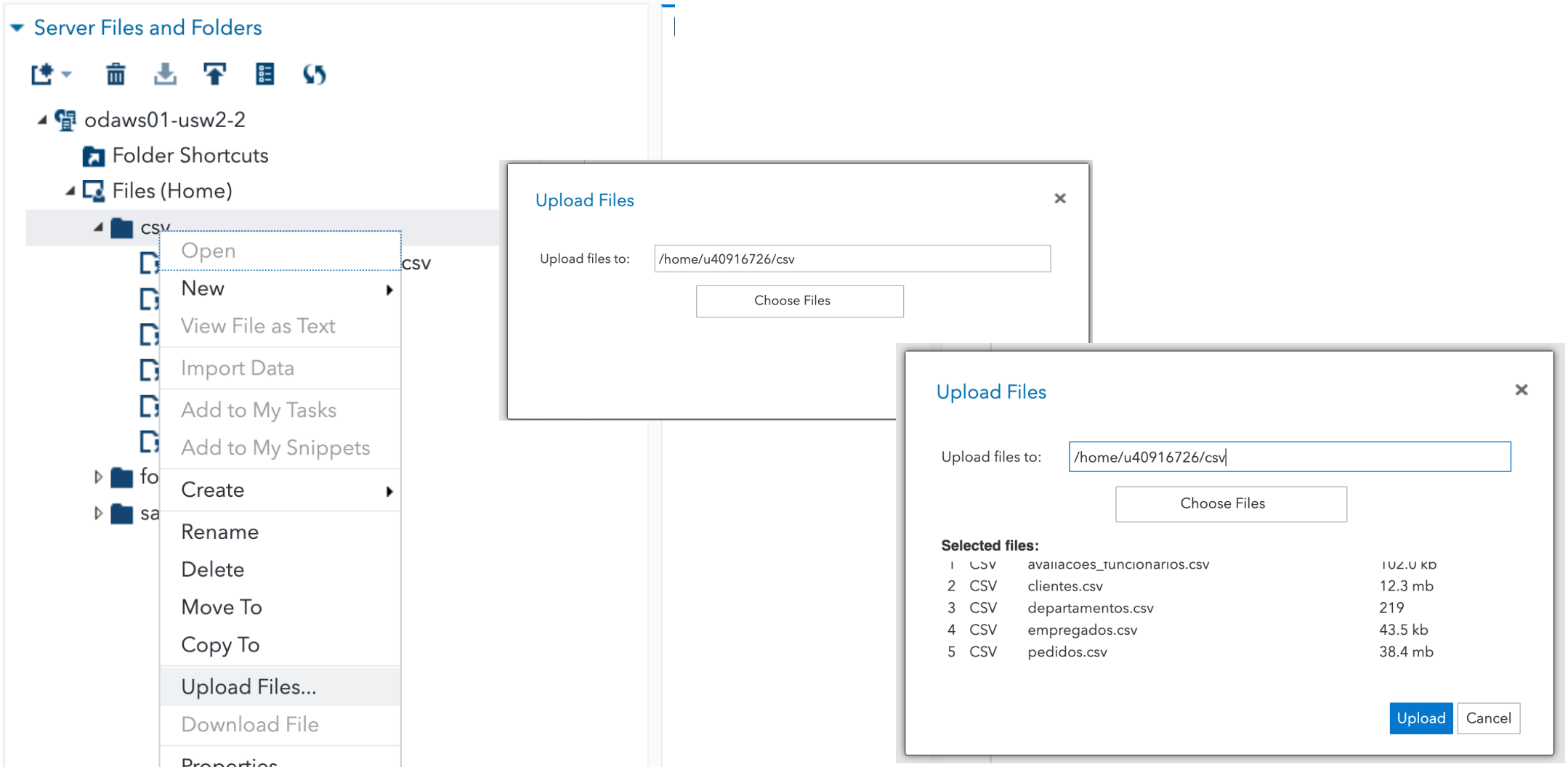
FORMAT é usado para controlar como os valores são exibidos, sem alterar o valor real armazenado na variável, ajustar a aparência dos dados para facilitar a leitura e a interpretação.
Ex. Exibir datas, moedas, porcentagens e outros formatos específicos.

Type of Language Element	Language Element	Input	Result
Date formats	DATE.	19069	17MAR12
	DATE9.	19069	17MAR2012
	DAY.	19069	17
	DDMMYY.	19069	17/03/12
	DDMMYY10.	19069	17/03/2012
	DDMMYYB.	19069	17 03 12
	DDMMYYB10.	19069	17 03 2012
	DDMMYYC.	19069	17:03:12
	DDMMYYC10.	19069	17:03:2012
	DDMMYYD.	19069	17-03-12
	DDMMYYD10.	19069	17-03-2012
	DDMMYYN6.	19069	170312
	DDMMYYN8.	19069	17032012
	DDMMYYP.	19069	17.03.12
	DDMMYYP10.	19069	17.03.2012
	DDMMYYS.	19069	17/03/12

```
data exemplo;
   data_nascimento = '12jun1990'd;
   format data_nascimento date9.;
run;
```



UPLOAD CSV


Server Files and Folders
odaws01-usw2-2
Folder Shortcuts
Files (Home)
csv
Open
New
View File as Text
Import Data
Add to My Tasks
Add to My Snippets
Create
Rename
Delete
Move To
Copy To
Upload Files...
Download File
Properties
Upload Files
Upload files to:
/home/u40916726/csv
Choose Files
Upload Files
Upload files to:
/home/u40916726/csv
Choose Files
Selected files:
2 CSV clientes.csv 12.3 mb
3 CSV departamentos.csv 219
4 CSV empregados.csv 43.5 kb
5 CSV pedidos.csv 38.4 mb
Upload
Cancel

PROC IMPORT

PROC IMPORT é utilizado para importar dados de arquivos externos (CSV, Excel, TXT) para dentro de datasets (tabelas) SAS de forma automatizada e flexível

```
PROC IMPORT
DATAFILE="/home/u40916726/csv/empregados.csv"
    OUT=formacao.empregados
    DBMS=CSV
    REPLACE;
    GETNAMES=YES;
RUN;
```

PROC IMPORT

Comando	Descrição
PROC IMPORT	Inicia o procedimento de importação de dados externos para o SAS.
DATAFILE	Especifica o caminho completo do arquivo externo a ser importado (neste caso, um arquivo CSV).
OUT	Define o nome e a biblioteca do novo dataset SAS que será criado com os dados importados.
DBMS	Indica o tipo do arquivo de origem; neste exemplo, um arquivo CSV (Comma-Separated Values).
REPLACE	Se já existir um dataset com o mesmo nome, ele será substituído pelo novo.
GETNAMES	Indica que a primeira linha do arquivo CSV contém os nomes das variáveis (colunas). Se fosse NO, os nomes seriam gerados automaticamente pelo SAS.

PROC IMPORT

Comando	Descrição
DELIMITER	Permite especificar outro delimitador, caso não seja vírgula (por exemplo, ponto e vírgula).
DATAROW	Define a partir de qual linha os dados devem ser lidos, útil quando o arquivo tem cabeçalhos extras.
GUESSINGROWS	Controla quantas linhas iniciais o SAS usa para adivinhar o tipo de cada coluna.
SHEET	Usado para importar uma planilha específica de um arquivo Excel.

PROC IMPORT

Exercício 34

- PROC IMPORT

  Importar os 5 ficheiros CSV que estão na diretoria “CSV” para a libname formacao, considerando o DBMS = CSV, fazendo a substituição dos dados (REPLACE) e cabeçalho = SIM (GETNAMES).

```
PROC IMPORT
DATAFILE="/home/u40916726/csv/empregados.csv"
    OUT=formacao.empregados
    DBMS=CSV
    REPLACE;
    GETNAMES=YES;
RUN;
```

MANIPULAÇÃO AVANÇADA DE DADOS

O que é manipulação avançada de dados no SAS?

Manipulação avançada de dados no SAS refere-se ao uso de técnicas e ferramentas para tratar, transformar, combinar, limpar e preparar grandes volumes de dados, muitas vezes de diferentes fontes e formatos, para análise e geração de relatórios. Isso inclui tarefas como importação/exportação de arquivos, uso de funções para processamento de variáveis, tratamento de dados faltantes, criação de variáveis derivadas, agregações, transposições, junções complexas, uso de arrays e processamento iterativo.

O objetivo é garantir que os dados estejam prontos e adequados para análises estatísticas, modelagens ou visualizações.

MANIPULAÇÃO AVANÇADA DE DADOS

Função (DATA STEP)	Equivalente no PROC SQL?	Explicação / Exemplo em SQL
RETAIN	Não tem equivalente direto	SQL não processa linha a linha sequencialmente. Use SUM() com GROUP BY ou subqueries.
IF / THEN	CASE WHEN	sql SELECT CASE WHEN idade < 18 THEN 'Menor' ELSE 'Adulto' END AS status
DO UNTIL / DO WHILE	Não tem em SQL padrão	SQL não suporta loops. Use macros, iterative joins, ou código fora do SQL para loops.
DELETE	DELETE FROM	sql DELETE FROM tabela WHERE salario IS NULL;
KEEP / DROP	SELECT (escolha colunas)	sql SELECT nome, idade FROM tabela; — Colunas não selecionadas são descartadas.
LENGTH (TABLE)	Não aplica em SQL puro	No PROC SQL, os comprimentos de variáveis character são definidos implicitamente. Para controle, use LENGTH no DATA STEP.
ARRAY	Não há array em SQL SAS	Use funções como MEAN(of ...) no DATA STEP, ou reestruture a tabela (verticalize com UNION ALL).
FIRST. / LAST.	GROUP BY + MIN()/MAX() ou MONOTONIC() com subquery	sql SELECT cliente, MIN(data) AS primeira_compra FROM vendas GROUP BY cliente;
BY	GROUP BY, ORDER BY	sql SELECT cliente, SUM(valor) FROM vendas GROUP BY cliente ORDER BY cliente;
INFILE + INPUT	Não em PROC SQL puro	Leitura de arquivos deve ser feita com DATA STEP. PROC IMPORT também pode ser usado.

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 35

- FIRST. / LAST.
  Escreva uma consulta para selecionar a primeira e a última data de pedido de cada cliente.

```
proc sort data=formacao.pedidos out=pedidos_ordenado;
  by cliente_id data_pedido;
run;
data work.final(keep=cliente_id primeira_compra ultima_compra);
  set pedidos_ordenado;
  by cliente_id;
  format primeira_compra ultima_compra ddmmyy10.;
  retain primeira_compra ultima_compra;
  if first.cliente_id then do;
    primeira_compra = data_pedido;
  end;
  ultima_compra = data_pedido;
  if last.cliente_id then output;
run;
```

```
proc sql;
  create table exerc_first_last as
  select cliente_id,
  min(data_pedido) format=ddmmyy10. as primeira_compra,
  max(data_pedido) format=ddmmyy10. as utlima_compra
  from formacao.pedidos
  group by cliente_id;
quit;
```

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 36

- Escreva uma consulta para selecionar apenas empregados que trabalham no departamento "TI" e ganham mais de $ 5500.

```
data funcionarios_ti_5500;
   set formacao.empregados;
   if departamento = 'TI' and salario = 5500;
run;
```

```
proc sql;
   select *
   from formacao.empregados
   where departamento = 'TI'
     and salario = 5500;
quit;
proc sql;
   create table funcionarios_ti_5500 as
   select *
   from formacao.empregados
   where departamento = 'TI'
     and salario = 5500;
quit;
```

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 37

- Combine as tabelas clientes e pedidos para exibir todos os pedidos feitos por clientes de uma cidade específica.

```
proc sort data=formacao.clientes; by cliente_id; run;
proc sort data=formacao.pedidos;  by cliente_id; run;

data pedidos_por_produto;
   merge formacao.clientes(in=inc) formacao.pedidos(in=inp);
   by cliente_id;
   if inc and inp and produto = 'Notebook';
run;
```

```
proc sql;
   create table pedidos_por_produto as
   select *
   from formacao.clientes as c
   inner join formacao.pedidos as p
     on c.cliente_id = p.cliente_id
   where p.produto = 'Notebook';
quit;
```

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 38

- Use o DATA Step para criar uma nova variável que classifica os empregados com base na experiência em "Júnior" até 4 anos e "Sênior" maior ou igual a 4 anos, do departamento de TI, criando um nova tabela somente com o id e a classificação.

```
data empregados_classificados;
  set formacao.empregados;
  where departamento = 'TI';

  length classificacao $10;

  if experiencia < 4 then classificacao = 'Júnior';
  else classificacao = 'Sênior';

  keep id classificacao;
run;
```

```
proc sql;
  create table empregados_classificados as
  select
    id,
    case
      when experiencia < 4 then 'Júnior'
      else 'Sênior'
    end as classificacao
  from formacao.empregados
  where departamento = 'TI';
quit;
```

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 38.1

- Duplique o exercício anterior e adicione as seguintes variáveis: o tempo de experiência e quanto falta para a mudança de classificação.

```
data empregados_classificados;
  set formacao.empregados;
  where departamento = 'TI';
  length classificacao $10;
  if experiencia < 4 then do;
    classificacao = 'Júnior';
    falta_mudar_class = 4 - experiencia;
  end;
  else do;
    classificacao = 'Sênior';
    falta_mudar_class = 0;
  end;
  keep id experiencia classificacao falta_mudar_class;
run;
```

```
proc sql;
  create table empregados_classificados as
  select
    id, experiencia,
    case
      when experiencia < 4 then 'Júnior'
      else 'Sênior’ end as classificacao,
    case
      when experiencia < 4 then 4 - experiencia
      else 0 end as falta_mudar_class
  from formacao.empregados
  where departamento = 'TI';
quit;
```

MANIPULAÇÃO AVANÇADA DE DADOS

Critério	PROC APPEND	UNION (em PROC SQL)	UNION ALL (em PROC SQL)
Objetivo	Anexar dados de uma tabela a outra	Combinar conjuntos de dados, eliminando duplicatas	Combinar conjuntos de dados, mantendo duplicatas
Modo de operação	Procedimento separado	Comando dentro do PROC SQL	Comando dentro do PROC SQL
Elimina duplicatas?	Não	Sim	Não
Requer estrutura idêntica?	Sim (ou FORCE para forçar)	Sim (mesmo número e tipo de colunas)	Sim
Eficiência (muito dados)	Muito eficiente	Pode ser mais lento com UNION (verifica duplicatas)	Mais rápido que UNION em grandes volumes
Permite adicionar tabela vazia?	Sim	Sim	Sim
Mantém ordem?	Mantém ordem inserida	Não (precisa de ORDER BY)	Não (precisa de ORDER BY)

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 39

- Utilizando o DATA Step, divida a tabela formacao.empregados em várias tabelas separadas por departamento (ex: TI, RH, Financeiro). Em seguida, una essas tabelas novamente utilizando PROC APPEND. Por fim, reproduza o mesmo resultado utilizando PROC SQL.

```
data empregados_ti empregados_rh empregados_financeiro;
  set formacao.empregados;

  if departamento = 'TI' then output empregados_ti;
  else if departamento = 'RH' then output empregados_rh;
  else if departamento = 'Financeiro' then output
empregados_financeiro;
run;
```

```
data empregados_unificados;
  set empregados_ti;
run;

proc append base=empregados_unificados
data=empregados_rh force; run;
proc append base=empregados_unificados
data=empregados_financeiro force; run;
```

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 39 continuação em Proc SQL;

- Utilizando o DATA Step, divida a tabela formacao.empregados em várias tabelas separadas por departamento (ex: TI, RH, Financeiro). Em seguida, una essas tabelas novamente utilizando PROC APPEND. Por fim, reproduza o mesmo resultado utilizando PROC SQL com UNION ALL.

```
proc sql;
  create table empregados_unificados_sql as
  (select * from formacao.empregados where departamento in ('TI'))
  union all
  (select * from formacao.empregados where departamento in ('RH',))
  union all
  (select * from formacao.empregados where departamento in ('Financeiro'));
quit;
```

MANIPULAÇÃO AVANÇADA DE DADOS

PROC SORT	Descritivo	Comando Exemplo
Básico	Ordena dataset por uma ou mais variáveis	proc sort data=entrada out=saida; by var1 var2; run;
Ordenação Descendente	Ordena variáveis em ordem decrescente	proc sort data=entrada out=saida; by descending var1 var2; run;
Remover Duplicatas por Chave (NODUPKEY)	Remove registros duplicados baseados nas variáveis BY	proc sort data=entrada out=saida nodupkey; by var1 var2; run;
Remover Duplicatas Exatas (NODUP)	Remove registros duplicados exatos (todas variáveis)	proc sort data=entrada out=saida nodup; by _all_; run;
Método Alternativo (TAGSORT)	Método alternativo eficiente para dados com muitos duplicados	proc sort data=entrada out=saida tagsort; by var1; run;
Sequência de Ordenação (SORTSEQ)	Define a regra regional/unicode para ordenação	proc sort data=entrada out=saida sortseq=unicode; by var1; run;

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 40

- Ordenar clientes por estado, depois cidade, depois nome (todas ascendente):
- Ordenar clientes por estado (descendente), depois cidade (ascendente):
- Remover registros duplicados com base nas variáveis nome e cidade (mantém a primeira ocorrência):
- Remover registros completamente duplicados (em todas as colunas):
- Ordenar usando o método tagsort (útil em muitos dados duplicados), por estado:
- Ordenar por nome usando regras Unicode (ex: acentos corretamente ordenados):

MANIPULAÇÃO AVANÇADA DE DADOS

Proc Format

- Datas no SAS são armazenadas como números (dias desde 01jan1960). Para exibir como DDMMYYYY sem separadores, precisamos de um formato customizado.

```
proc format;
  picture ddmmyyyy (default=8)
  low-high = '%0d%0m%Y' (datatype=date);
run;
```

MANIPULAÇÃO AVANÇADA DE DADOS

```
proc format;
  picture ddmmyyyy (default=8)
  low-high = '%0d%0m%Y' (datatype=date);
run;
```

Explicação dos Componentes

proc format;

- Inicia o procedimento que permite criar formatos personalizados em SAS.

picture ddmmyyyy (default=8)

- Cria um formato do tipo picture chamado `ddmmyyyy`.
- O parâmetro `(default=8)` define o comprimento padrão do formato como 8 caracteres, adequado para datas no formato `ddmmaaaa` (por exemplo, `15062025` para 15/06/2025).

low-high = '%0d%0m%Y' (datatype=date);

- `low-high` indica que o formato se aplica a todo o intervalo possível de datas SAS.
- A string `'%0d%0m%Y'` define o padrão de formatação:
- `%0d`: dia com dois dígitos, com zero à esquerda se necessário.
- `%0m`: mês com dois dígitos, com zero à esquerda se necessário.
- `%Y`: ano com quatro dígitos.
- `(datatype=date)` informa ao SAS que os valores a serem formatados são datas SAS (números inteiros que representam dias desde 01/01/1960).

run;

- Finaliza o procedimento que permite criar formatos personalizados em SAS.

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 41

- Criar para todas as datas das tabelas importadas “empregados” e “pedidos” com o formato YYYYMMDD.

```
proc format;
  picture yyyymmdd (default=8)
    low-high = '%0Y%0m%d' (datatype=date);
run;

proc print data=formacao.pedidos (obs=100);
  format data_pedido yyyymmdd.;
run;
```

MANIPULAÇÃO AVANÇADA DE DADOS

Macro

- Uma macro no SAS é um conjunto de instruções que automatiza tarefas repetitivas, tornando o código mais flexível e dinâmico. Com macros, você pode criar pequenos "programas" parametrizáveis dentro do SAS, permitindo rodar procedimentos múltiplas vezes com diferentes entradas, sem precisar reescrever o código. Elas são muito úteis para padronizar processos, evitar redundância e facilitar a manutenção de programas SAS

```
%macro existe_setor(setor);
   %global resultado;
   proc sql noprint;
      select case when count(*) > 0 then 'VERDADEIRO' else 'FALSO' end
      into :resultado trimmed
      from formacao.departamentos
      where setor = "&setor";
   quit;
   %put &resultado;
%mend existe_setor;

%existe_setor(Administrativo);
```

MANIPULAÇÃO AVANÇADA DE DADOS

Explicação dos Componentes

%macro existe_setor(setor);

- Início do macro chamado `existe_setor`, que recebe um parâmetro chamado `setor`.

%global resultado;

- Declara a variável macro global chamada `resultado`. Isso faz com que o valor de `resultado` esteja disponível fora do macro, após sua execução.

proc sql noprint;

- Inicia um bloco PROC SQL no SAS, que permite executar comandos SQL. O parâmetro `noprint` indica que não será exibida nenhuma saída na tela.

```
select case when count(*) > 0 then 'VERDADEIRO' else 'FALSO' end
    into :resultado trimmed
    from formacao.departamentos
    where setor = "&setor";
```

- Utiliza um `CASE` para verificar se existe pelo menos um registro na tabela `formacao.departamentos` onde a coluna `setor` é igual ao valor passado como parâmetro.

MANIPULAÇÃO AVANÇADA DE DADOS

Explicação dos Componentes

quit;

- Finaliza o bloco PROC SQL.

%put &resultado;

- Exibe o valor de `resultado` no log do SAS. Isso serve para mostrar se o setor existe ou não.

%mend existe_setor;

- Finaliza o macro.

%existe_setor(Administrativo);

- Chama a macro passando o valor `Administrativo` como parâmetro.

MANIPULAÇÃO AVANÇADA DE DADOS

Principais Comandos e Funções de Macro no SAS

Comando	Exemplo	Descrição
%let	%let nome=João;	Cria e atribui valor a uma variável macro.
%put	%put O nome é &nome;	Exibe mensagens ou valores de variáveis macro no log do SAS.
%if ... %then ... %else	%if &idade >= 18 %then %put Maior de idade; %else %put Menor de idade;	Realiza uma decisão condicional dentro de um macro.
%do ... %end	%do i=1 %to 5; %put Iteração &i; %end;	Executa um loop dentro de um macro.
%sysfunc	%let data=%sysfunc(today(), date9.); %put Hoje é &data;	Usa funções do SAS dentro de macros (como today(), mean(), etc.).
%eval	%let soma=%eval(3+4);	Avalia expressões **inteiras** dentro de macros.
%sysevalf	%let media=%sysevalf(10/3);	Avalia expressões **com ponto flutuante**.
%substr	%let texto=Exemplo; %put %substr(&texto,2,3);	Retorna parte de uma string macro: xem.
%upcase	%put %upcase(&texto);	Converte string para maiúsculas.
%lowcase	%put %lowcase(&texto);	Converte string para minúsculas.
%length	%put %length(&texto);	Retorna o número de caracteres da string.
%scan	%put %scan(Olá mundo,2);	Retorna a 2ª palavra da string (separada por espaço por padrão).
%global	%global resultado;	Cria variável macro **global** (acessível fora do macro).
%local	%local temp;	Cria variável macro **local** (válida apenas dentro do macro).
%include	%include '/caminho/meuarquivo.sas';	Inclui e executa um script .sas externo.
%macro / %mend	%macro saudacao; %put Olá!; %mend;	Define um bloco de macro reutilizável.

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 42

- Criar uma macro que faça o UPCASE no parametro enviado.

```
%macro to_upcase(texto);
  %upcase(&texto)
%mend;

data exerc42;
  nome_macro = "%to_upcase(william rosario)";
run;
```

```
%macro to_upcase_macro(texto, var_saida);
  %global &var_saida;
  %let &var_saida = %upcase(&texto);
%mend;

%to_upcase_macro(william rosario, nome_maisculo);

/* Usar no data step */
data exerc42;
  nome_macro = "&nome_maisculo";
run;
```

MANIPULAÇÃO AVANÇADA DE DADOS

Exercício 42.1

- Criar um macro que faça o update na coluna setor filtrando por um departamento específico na tabela formacao.departamento.

```
%macro inserir_departamento(departamento, setor);
   data departamentos;
      set departamentos end=last;
      output;
      if last then do;
         departamento = "&departamento";
         setor = "&setor";
         output;
      end;
   run;
%mend inserir_departamento;

%inserir_departamento(Financeiro, Administrativo);
```

```
%macro inserir_departamento(departamento, setor);
   proc sql;
      insert into departamentos (departamento, setor)
      values ("&departamento", "&setor");
   quit;
%mend inserir_departamento;

%inserir_departamento(Financeiro, Administrativo);
```

CONEXÃO COM O BANCO DE DADOS

O SAS permite acessar bancos de dados (como Oracle, SQL Server, PostgreSQL, etc.) diretamente.

O que é LIBNAME?
A instrução LIBNAME permite conectar-se a uma base de dados e tratar suas tabelas como bibliotecas SAS.

Exemplos:
SAS
libname minha_conexao base 'caminho';

ODBC
libname minha_conexao odbc dsn=meu_dsn user=meu_usuario password=minha_senha schema=public;

minha_conexao é o apelido (libref) da conexão.
odbc é o tipo de conexão (pode variar: oracle, mysql, postgres, etc.).
dsn refere-se à configuração ODBC do banco.

CONEXÃO COM O BANCO DE DADOS

O que é Pass-Through ?
É uma técnica que permite enviar SQL nativo diretamente para o banco de dados, aproveitando otimizações específicas daquele banco (índices, funções, joins complexos).

```
proc sql;
  connect to odbc (dsn=SQLServerDSN user=william password=123456);
  create table resultado as
  select * from connection to odbc (
    select nome, idade from clientes where idade > 30
  );
  disconnect from odbc;
quit;
```

connect to odbc (...): abre a conexão com o banco.
connection to odbc (...): envia um SQL nativo do banco.
create table resultado as: armazena o resultado como uma tabela SAS.
disconnect from odbc: encerra a conexão.

CONEXÃO COM O BANCO DE DADOS

Quando Usar?

Situação	Melhor opção
Acesso simples a tabelas	LIBNAME
Consultas SQL nativas (JOINs, funções SQL)	PROC SQL com pass-through
Performance é crítica	Pass-through (executa no banco)
Exportação de dados	LIBNAME + PROC EXPORT

LIBNAME é simples e permite usar o banco como uma pasta de datasets SAS.
PROC SQL com pass-through é mais poderoso e eficiente para queries complexas.

PERFORMANCE E EFICIÊNCIA

Índices

Objetivo: Acelerar a leitura e busca de dados, especialmente em tabelas grandes com filtros (where) frequentes.
Como funciona: Assim como no SQL, o índice cria uma estrutura que permite acesso mais rápido aos dados.

```
proc datasets lib=formacao;
  modify pedidos;
  index create pedido_id / unique;
quit;
```

Explicação:
- Cria um índice sobre a variável pedido_id da tabela pedidos.
- unique garante que os valores sejam únicos (opcional).
- Ideal para colunas usadas em where, by, merge.

Nem sempre índice melhora performance. Use quando:
- A tabela for grande.
- A coluna for muito usada em filtros (where) ou ordenações (by).
- A tabela for usada repetidamente no programa.

PERFORMANCE E EFICIÊNCIA

Tipo de Índice	Descrição	Sintaxe Exemplo	Quando Usar
Simples	Índice criado sobre uma única variável.	index create id_cliente;	Quando você faz buscas ou filtros frequentes por uma única variável.
Composto	Índice criado sobre duas ou mais variáveis juntas.	index create idx_completo = (id_cliente data_venda);	Quando o filtro envolve múltiplas variáveis, na ordem em que foram indexadas.
Único (unique)	Garante que os valores do índice são únicos (sem repetições).	index create id_cliente / unique;	Quando a variável é uma chave primária (ex: CPF, código único).
Nominal (nomeado)	Você pode dar um nome ao índice, útil em índices compostos.	index create idx_nome = (sexo idade);	Para facilitar identificação e referência, especialmente em compostos.
Automático	Criado automaticamente pelo SAS (ex: ao usar BY com PROC SORT, FIRST. etc).	-	Interno ao SAS; não é visível diretamente, mas pode acelerar certas operações.

PERFORMANCE E EFICIÊNCIA

```
proc datasets lib=formacao;
  modify pedidos;

  /* Índice simples */
  index create cliente_id;

  /* Índice composto */
  index create idx_data_cliente = (cliente_id data_pedido);

  /* Índice único */
  index create pedido_id / unique;
quit;
```

```
proc contents data= formacao.pedidos;
run;
```

```
data clientes_filtrados;
  set formacao.pedidos(idxname=cliente_id);
  where cliente_id = 12345;
run;
```

PERFORMANCE E EFICIÊNCIA

Exercício 43

- Criar um indice unico (unique) para a tabela formacao.clientes , coluna cliente_id.

```
proc datasets lib=formacao;
  modify clientes;
  index create cliente_id / unique;
quit;
```

PERFORMANCE E EFICIÊNCIA

Compressão de Datasets

Objetivo: Reduzir o uso de disco e melhorar leitura/escrita (I/O), principalmente em datasets com muitas colunas de texto ou valores repetidos.

```
data formacao.clients_c(compress=yes);
  set formacao.clientes;
run;
```

Explicação:

- compress=yes: aplica compressão automática de variáveis de caractere e/ou numéricas.
- Reduz o tamanho do arquivo SAS (.sas7bdat) no disco.
- Pode acelerar leitura/gravação, especialmente em servidores com I/O lento.

- compress=binary (melhor para dados numéricos).
- compress=char (melhor para dados de texto).

PERFORMANCE E EFICIÊNCIA

Exercício 44

- Criar uma tabela chamada pedidos_compress, utilizando a compressão = yes , e verificar a diferença de tamanho com a original.

```
data formacao.pedidos_compress(compress=yes);
  set formacao.pedidos;
run;
```

PERFORMANCE E EFICIÊNCIA

Query Planning e Otimização

Objetivo: Compreender como o SAS executa passos de código para escrever scripts mais eficientes e com menos uso de recursos.

```
data resultado;
  set pedidos(keep=cliente_id valor data_pedido);
  where valor > 1000;
run;
```

*Filtro e seleção de variáveis diretamente no set evita processamento extra.

- /* Mais eficiente */

```
set pedidos(where=(valor>1000));
```

- /* Menos eficiente */

```
set pedidos;
if valor > 1000;
```

•Evite variáveis temporárias desnecessárias.
•Remova colunas que não são usadas no resultado final com drop= ou keep=.
•Use PROC SQL apenas quando necessário.
•Em muitos casos, DATA STEP é mais rápido que PROC SQL para operações simples.
•Evite sort desnecessário.
•Ordenações são pesadas. Só use proc sort se for realmente necessário (como para merge).

PERFORMANCE E EFICIÊNCIA

Prática	Vantagem
Dividir o processo em partes	Permite reuso, evita reprocessamento desnecessário.
Utilizar options fullstimer;	Medir tempo de CPU/real gasto em cada etapa do código.

```
options compress=yes fullstimer;
data pedidos_filtro(keep=cliente_id valor data_pedido);
  set pedidos(where=(valor>1000));
run;
proc sort data=pedidos_filtro out=pedidos_ordenada;
  by cliente_id;
run;
```

- compress=yes: reduz espaço.
- where aplicado na leitura: mais rápido.
- keep= usado direto: menos memória.
- fullstimer: mostra tempo de execução e I/O no log.

MANIPULAÇÃO DE GRANDES VOLUMES DE DADOS

Particionamento de Dados

Objetivo: Dividir grandes datasets em partes menores para facilitar processamento, manutenção e paralelização.

Como particionar?
Por variável de agrupamento (ano, região, tipo etc.)
Em arquivos separados ou tabelas distintas

```
data pedidos_2023 pedidos_2024;
  set pedidos;

  if year(data_pedido) = 2023 then output pedidos_2023;
  else if year(data_pedido) = 2024 then output pedidos_2024;
run;
```

Por que usar?
- Facilita o processamento seletivo (apenas dados de 2024, por exemplo)
- Evita ler dados irrelevantes em análises futuras

PERFORMANCE E EFICIÊNCIA

Paralelismo

Objetivo: Acelerar o processamento de dados dividindo o trabalho em várias threads ou subprocessos.

Método	**Descrição**
SAS Grid / Viya	Permite execução distribuída em clusters (via SAS Grid Manager)
MP CONNECT	Executa tarefas em paralelo em sessões independentes
signon, rsubmit	Comandos para controle remoto/paralelo (exige licença SAS/CONNECT)
Paralelismo manual	Dividir dados e processar em paralelo com agendadores externos (Ex: cron)

PERFORMANCE E EFICIÊNCIA

Paralelismo com MP CONNECT

```
/* Ativar sessões paralelas */
signon tarefa1;
signon tarefa2;
rsubmit tarefa1;
  data parte1;
    set bigdata(firstobs=1 obs=1000000);
    /* processamento */
  run;
endrsubmit;
rsubmit tarefa2;
  data parte2;
    set bigdata(firstobs=1000001 obs=2000000);
    /* processamento */
  run;
endrsubmit;
waitfor _all_ tarefa1 tarefa2;
```

Importante: Requer configuração do ambiente e licenças específicas (Grid, Viya ou SAS/CONNECT).

PERFORMANCE E EFICIÊNCIA

FIRST. e LAST. no DATA Step

Objetivo: Trabalhar com agrupamentos de dados sem proc sql, com excelente performance.

Como funciona:
Após um BY, SAS cria variáveis automáticas FIRST. e LAST. para cada variável do BY.
Útil para calcular totais por grupo, pegar primeira/última linha, controlar transições de grupo.

Pré-requisito:

```
proc sort data=formacao.pedidos;
  by cliente_id data_pedido;
run;
```

```
data primeira_compra;
  set formacao.pedidos;
  by cliente_id;
  if first.cliente_id then output;
run;
```

```
data total_cliente;
  set formacao.pedidos;
  by cliente_id;
  retain total 0;
  total + valor;
  if last.cliente_id then do;
    output;
    total = 0;
  end;
run;
```

Vantagens:
- Muito mais rápido do que PROC SQL ou PROC MEANS em grandes volumes
- Executa linha a linha (streaming), ideal para ambientes com pouca RAM

INTEGRAÇÃO DO SAS COM FERRAMENTAS ANALÍTICAS

O SAS combina poderosas procedures estatísticas (PROCs) com a capacidade de manipulação e consulta de dados via DATA Step e PROC SQL.

PROC REG – Regressão Linear
PROC GLM – Modelos Lineares Gerais
PROC LOGISTIC – Regressão Logística

INTEGRAÇÃO DO SAS COM FERRAMENTAS ANALÍTICAS

PROC REG – Regressão Linear

```
proc reg data=meus_dados;
  model y = x1 x2 x3;
run;
quit;
```

Uso: Avaliar relação linear entre variáveis (ex: prever vendas com base em preço e propaganda).

INTEGRAÇÃO DO SAS COM FERRAMENTAS ANALÍTICAS

PROC GLM – Modelos Lineares Gerais

```
proc glm data=meus_dados;
  class grupo;
  model y = grupo x1;
run;
quit;
```

Uso: ANOVA, ANCOVA, e modelos com variáveis categóricas e contínuas.

INTEGRAÇÃO DO SAS COM FERRAMENTAS ANALÍTICAS

PROC LOGISTIC – Regressão Logística

```
data meus_dados;
  input id_cliente idade sexo $ renda evento_binario;
  datalines;
1 25 M 3500 0
2 45 F 7000 1
3 33 F 5000 1
4 28 M 4200 0
5 60 F 10000 1
6 40 M 8000 0
;
run;
```

```
proc logistic data=meus_dados outmodel=modelo_log;
  class sexo (param=ref ref='M'); /* necessário para variáveis categóricas */
  model evento_binario(event='1') = idade sexo renda;
run;
```

Uso: Modelar eventos binários (sim/não), como conversão de clientes ou risco de inadimplência.

INTEGRAÇÃO DO SAS COM FERRAMENTAS ANALÍTICAS

PROC LOGISTIC – Regressão Logística

Explicações:

- outmodel=modelo_log: salva o modelo em um objeto reutilizável.
- sexo (param=ref ref='M'): usa codificação dummy (ref. masculina).
- A variável dependente evento_binario deve conter 0 e 1.

```
/*Aplicar o Modelo a Novos Dados (Score)*/
proc logistic inmodel=modelo_log;
  score data=meus_dados out=meus_dados_com_score
        outroc=roc_data;
run;
```

O que é gerado:

•Dataset meus_dados_com_score com a coluna P_1 → probabilidade do evento = 1.
•Opcional: outroc=roc_data → dados da curva ROC para avaliação.

INTEGRAÇÃO DO SAS COM FERRAMENTAS ANALÍTICAS

PROC LOGISTIC – Regressão Logística

```
/*Juntar Resultados com os Dados Originais (opcional)*/
proc sql;
  create table analise_final as
  select a.*, b.P_1 as prob_evento
  from meus_dados a
  left join meus_dados_com_score b
  on a.id_cliente = b.id_cliente;
quit;
```

INTEGRAÇÃO DO SAS COM FERRAMENTAS ANALÍTICAS

PROC FORECAST – Previsões

```
data vendas;
  input data :date9. vendas;
  format data date9.;
  datalines;
01JAN2023 100
01FEB2023 120
01MAR2023 130
01APR2023 140
01MAY2023 160
01JUN2023 170
01JUL2023 175
01AUG2023 180
01SEP2023 185
01OCT2023 190
01NOV2023 200
01DEC2023 210;
run;
```

```
proc forecast data=vendas out=previsto lead=6 interval=month method=expo;
  id data;
  var vendas;
run;
```

Código	O que faz
data=vendas	Usa a série histórica criada
out=previsto	Cria uma nova tabela com os valores previstos
lead=6	Gera previsão para os 6 meses seguintes
interval=month	Indica que os dados estão em intervalo mensal
method=expo	Método de previsão: suavização exponencial (outros: STEPM, WINTERS)
id data;	Define a variável de tempo
var vendas;	Variável que será prevista

INTEGRAÇÃO DO SAS COM FERRAMENTAS ANALÍTICAS

Exercício 45 - PROC FORECAST

- Criar uma tabela de pedidos particionada e ordenada pelo ultimo ano de informação.
- Criar um index na tabela particionada criada acima pelo campo cliente_id.
- Criar uma previsão de pedidos para os próximos 6 meses por cliente_id
- Salvar o resultado na libname formacao

INTEGRAÇÃO DO SAS COM FERRAMENTAS ANALÍTICAS

PROCs	Uso
PROC GENMOD	Modelos lineares generalizados (GLM)
PROC PHREG	Modelos de sobrevivência (Cox)
PROC SVM	Support Vector Machines (em Viya)
PROC ARIMA/ESM	Modelos ARIMA e exponenciais para séries

EXTRA

PROC EXPORT: usado para exportar dados de tabelas SAS para arquivos externos.

```
proc sql outobs=500;
  create table clientes_500 as
  select *
  from formcao.clientes;
quit;

proc export data=clientes_500
  outfile="/home/u40916726/csv/clientes_500.csv"
  dbms=csv
  replace;
  delimiter=';';
run;
```

Comando	Explicação
proc sql outobs=500;	Seleciona no máximo 500 registros
create table clientes_500 as	Cria tabela temporária com esses dados
outfile=...	Caminho completo do arquivo CSV
dbms=csv	Formato de saída é CSV
delimiter=';'	Define separador como ;
replace	Sobrescreve o arquivo, se já existir

EXTRA

Comando	Explicação
PROC EXPORT	Inicia o procedimento para exportar dados de um dataset SAS para um arquivo externo (como CSV, Excel, banco de dados, etc.).
DATA=<libref.>SAS-data-set	Especifica o dataset SAS que será exportado. Pode incluir a biblioteca (libref), ex: work.vendas.
<(SAS-data-set-options)>	(Opcional) Filtros ou opções para selecionar parte dos dados, como where=.
OUTFILE="filename"	Define o caminho e nome do arquivo de saída, como "C:\dados\saida.csv". Usado para arquivos externos como CSV, TXT, XLSX.
OUTTABLE="tablename"	Usado ao exportar para bancos de dados — define o nome da tabela de destino.
DBMS=identifier	Informa o tipo de destino (ex: CSV, XLSX, ACCESS, MYSQL, etc.).
REPLACE	(Opcional) Substitui o arquivo existente se ele já existir.
LABEL	(Opcional) Exporta os rótulos (label) das variáveis, em vez dos nomes.
<data-source-statement(s)>	(Opcional) Instruções específicas do tipo de saída (como encoding, delimitadores).

EXTRA

Exercício 46 - PROC EXPORT

- Exportar todas as tabelas (avaliacoes_funcionarios, clientes, departamentos, empregados, pedidos) para diferentes tipos de saída.

DBMS=	Tipo de arquivo	Observações
CSV	Arquivo CSV (valores separados por vírgula ou ;)	Permite usar delimiter=';' para ponto e vírgula
TAB	Arquivo de texto com tabulação (.txt)	Usa tab como separador
DLM	Texto delimitado por outro caractere	Combinado com delimiter=
XLS	Arquivo Excel 97-2003 (.xls)	Requer Excel de 32 bits no Windows
XLSX	Arquivo Excel moderno (.xlsx)	Compatível com Excel 2007+

VISUALIZAÇÃO E RELATÓRIOS

PROC REPORT

Função:

- Criação de relatórios tabulares altamente customizáveis, permitindo sumarizar, analisar e apresentar dados de forma estruturada.

Diferenciais:

- Permite criar tanto tabelas de detalhe quanto de resumo.
- Suporta a criação de colunas calculadas diretamente no relatório, sem necessidade de criar variáveis extras no dataset.
- Grande flexibilidade para customização do layout, ordenação, formatação e inclusão de totais e subtotais.
- Possibilita lógica condicional e cálculos personalizados usando blocos COMPUTE.

```
proc report data=dataset;
  column var1 var2 var3;
run;
```

VISUALIZAÇÃO E RELATÓRIOS

PROC REPORT

Comando / Bloco	Função / Descrição
PROC REPORT DATA=...	Inicia o relatório e indica a tabela de entrada.
COLUMN col1 col2 ...	Define as colunas que aparecerão no relatório e a ordem.
DEFINE	Define o uso e as propriedades da coluna (ex: agrupamento, rótulo, alinhamento).
BREAK location var </options>	Insere quebras (ex: subtotal) ao mudar o valor de uma variável.
RBREAK location </options>	Adiciona uma quebra de linha geral (ex: total geral no final da tabela).
COMPUTE location <var>	Bloco para adicionar texto, lógica condicional ou formatação.
CALL DEFINE(col, 'attr', val)	Muda dinamicamente atributos como cor ou estilo dentro de COMPUTE.
BY var	Executa o relatório em grupos, requer PROC SORT antes.
FREQ var	Conta a frequência de valores da variável, como em PROC FREQ.
WEIGHT var	Aplica pesos aos cálculos (como soma ponderada).

VISUALIZAÇÃO E RELATÓRIOS

Exercício 46

- PROC REPORT – Criar um relatório tabular simples mostrando a quantidade de clientes por estado:

```
proc report data=formacao.clientes nowd;
  column estado n;
  define estado / group 'Estado';
  define n / 'Quantidade de Clientes';
run;
```

Comando	Explicação
proc report data=formacao.clientes nowd;	Inicia o PROC REPORT com a tabela formacao.clientes. O nowd ativa o modo linha de comando, ou seja, sem a janela interativa (WINDOWS).
column estado n;	Define as colunas que irão aparecer no relatório. estado será agrupado e n representa a contagem automática de linhas (frequência) por grupo.
define estado / group 'Estado';	Diz ao SAS para agrupar por estado (similar a GROUP BY no SQL). O rótulo da coluna será "Estado".
define n / 'Quantidade de Clientes';	Define que a coluna n será usada apenas como exibição e dá o rótulo "Quantidade de Clientes". Como n não é uma variável real, o SAS entende que é o número de observações por grupo.
run;	Executa o relatório.

VISUALIZAÇÃO E RELATÓRIOS

PROC TABULATE

Função:

- Geração de tabelas sumarizadas (tabulações cruzadas), exibindo estatísticas descritivas com possibilidade de múltiplas dimensões (páginas, linhas, colunas).

Diferenciais:

- Ideal para criar tabelas cruzadas (cross-table) e sumarizações hierárquicas.
- Permite múltiplas dimensões e hierarquias complexas, exibindo estatísticas como soma, média, contagem, etc.
- Oferece recursos avançados de formatação, rotulagem e organização dos dados.

```
proc tabulate data=dataset;
  class var1 var2;
  var var3;
  table var1, var2*var3*mean;
run;
```

VISUALIZAÇÃO E RELATÓRIOS

Comando	Função / Descrição
PROC TABULATE <options>;	Inicia o procedimento. Pode incluir opções como DATA=, OUT=, FORMAT=, etc.
BY variable(s)	Executa o relatório separadamente por grupo de dados. Requer que os dados estejam previamente ordenados.
CLASS variable(s) </options>;	Define variáveis categóricas (ex: sexo, ano, região) que serão usadas como linhas ou colunas no relatório.
VAR analysis-variable(s);	Define variáveis numéricas, sobre as quais serão feitas análises estatísticas (ex: soma, média, min, max).
TABLE <page, row, column>;	Define o layout do relatório em páginas (opcional), linhas e colunas. Pode incluir estatísticas como mean, sum, etc.
CLASSLEV variable / STYLE=...;	Permite aplicar estilo a níveis específicos de variáveis de classe (como cor de fundo, fonte etc.).
FREQ variable;	Conta as observações com base em uma variável de frequência (ex: já agregada).
KEYLABEL keyword='label';	Substitui os rótulos padrão das palavras-chave como MEAN, SUM, etc., por nomes mais legíveis no relatório.
KEYWORD keyword / STYLE=...;	Permite formatar visualmente palavras-chave (como MEAN, N, etc.) nas células da tabela.
WEIGHT variable;	Aplica pesos aos cálculos de estatísticas. Útil para médias ponderadas, por exemplo.

VISUALIZAÇÃO E RELATÓRIOS

Exercício 47

- PROC TABULATE – Criar um relatório com o resumo estatístico das notas finais da tabela avaliações de funcionários por ano:

```
proc tabulate data=formacao.avaliacoes_funcionarios;
 class ano ;
 var nota_final;
 table ano, nota_final*(mean min max);
run;
```

Comando	Explicação
proc tabulate data=formacao.avaliacoes_funcionarios;	Inicia o procedimento TABULATE utilizando os dados da tabela avaliacoes_funcionarios (localizada na biblioteca formacao).
class ano;	Declara que ano é uma variável categórica (classe), ou seja, será usada para quebras de linha ou coluna no relatório.
var nota_final;	Indica que nota_final é uma variável numérica de análise, ou seja, sobre a qual serão aplicadas estatísticas.
table ano, nota_final*(mean min max);	Monta a tabela cruzada: As linhas (ano) correspondem aos anos. As colunas mostrarão as estatísticas (média, mínimo e máximo) da variável nota_final para cada ano.

VISUALIZAÇÃO E RELATÓRIOS

Exercício 48

- PROC TABULATE – Criar um relatório com o resumo estatístico das notas finais de avaliações de funcionários por ano e sexo da tabela de empregados:

```
proc tabulate data=avaliacoes_com_sexo;
  class ano sexo;
  var nota_final;
  table
    ano*sexo
    nota_final*(mean min max std n)
    / misstext='-';
run;
```

Comando	Explicação
proc tabulate data=avaliacoes_com_sexo;	Inicia o procedimento TABULATE usando a tabela avaliacoes_com_sexo.
class ano sexo;	Define ano e sexo como variáveis categóricas (agrupamento).
var nota_final;	Define nota_final como variável numérica para calcular estatísticas.
table ano*sexo,	Linha da tabela: combinação de ano e sexo (ex: 2023M, 2023F...).
nota_final*(mean min max std n)	Coluna da tabela: estatísticas aplicadas sobre nota_final.
n	Número de observações (quantidade de notas avaliadas).
/ misstext='-';	Mostra - se não houver dados disponíveis para alguma célula.
run;	Finaliza o PROC TABULATE.

VISUALIZAÇÃO E RELATÓRIOS

PROC SGPLOT

Função:

- Criação de gráficos (plots) de alta qualidade, como histogramas, boxplots, gráficos de linhas, barras, dispersão, entre outros.

Diferenciais:

- Permite sobrepor múltiplos tipos de gráficos em um mesmo eixo.
- Fácil de usar para visualizações rápidas e customizáveis.
- Suporta agrupamentos e comparações entre grupos diretamente nos gráficos.

```
proc sgplot data=dataset;
  scatter x=var1 y=var2;
  series x=var1 y=var3;
run;
```

VISUALIZAÇÃO E RELATÓRIOS

Comando	Descrição / Objetivo
PROC SGPLOT	Inicia o procedimento de criação de gráficos.
BAND	Faixa sombreada entre valores (ex: intervalos de confiança).
BUBBLE	Gráfico de bolhas com tamanho proporcional à variável.
DOT	Gráfico de pontos para categorias.
ELLIPSE	Desenha elipse de agrupamento/confiança.
GRADLEGEND	Legenda para variáveis com cores graduais.
HBARBASIC	Versão simplificada de HBAR.
HBOX	Boxplot horizontal.
HEATMAPPARM	Mapa de calor com dados pré-agrupados.
HISTOGRAM	Distribuição de uma variável numérica.

Comando	Descrição / Objetivo
STYLEATTRS	Define atributos de estilo como cores, marcadores, linhas.
BLOCK	Destaca áreas com base em categorias.
DENSITY	Plota curva de densidade para variáveis contínuas.
DROPLINE	Linhas verticais/horizontais de um ponto até o eixo.
FRINGE	Marcas no eixo indicando observações.
HBAR	Gráfico de barras horizontais.
HBARPARM	Barras horizontais com dados resumidos.
HEATMAP	Mapa de calor com base em densidade.
HIGHLOW	Linhas verticais ou horizontais entre mínimo e máximo.
HLINE	Linha horizontal para categorias.

VISUALIZAÇÃO E RELATÓRIOS

Comando	Descrição / Objetivo	Comando	Descrição / Objetivo
INSET	Insere textos explicativos dentro do gráfico.	KEYLEGEND	Controla a legenda dos elementos.
LINEPARM	Linha com ponto e inclinação definida.	LOESS	Ajuste de curva local (LOESS).
NEEDLE	Gráfico de agulhas – linhas verticais finas.	PBSPLINE	Ajuste suave com splines penalizados.
POLYGON	Desenha polígonos com X, Y e ID.	REFLINE	Linhas de referência (ex: metas).
REG	Regressão linear sobre os dados.	SCATTER	Gráfico de dispersão (X vs Y).
SERIES	Linha conectando pontos (ex: série temporal).	STEP	Gráfico de passos (stepwise).
SYMBOLCHAR	Define símbolos com caracteres personalizados.	SYMBOLIMAGE	Usa imagem como símbolo gráfico.
TEXT	Adiciona texto nas coordenadas X e Y.	VBAR	Gráfico de barras verticais.
VBARBASIC	Versão simples do VBAR.	VBARPARM	Barras verticais com dados agregados.
VBOX	Boxplot vertical.	VECTOR	Vetores com direção e magnitude.
VLINE	Linha vertical para categorias.	WATERFALL	Gráfico de cascata (acúmulo de valores).
XAXIS	Configurações do eixo X.	YAXIS	Configurações do eixo Y.

VISUALIZAÇÃO E RELATÓRIOS

Exercício 49

- PROC SGPLOT – Criar um gráfico de barra (vbar) que mostra a quantidade de clientes por estado.

```
proc sgplot data=formacao.clientes;
  vbar estado / stat=freq datalabel;
  title "Quantidade de Clientes por Estado";
run;
```

Comando	Explicação
proc sgplot data=clientes;	Usa a tabela clientes para criar o gráfico.
vbar estado	Cria barras verticais para cada valor da variável estado.
stat=freq	Conta quantas vezes cada estado aparece (frequência).
datalabel	Mostra o número de clientes acima de cada barra.
title	Define o título do gráfico.

VISUALIZAÇÃO E RELATÓRIOS

PROC SGSCATTER

Função:

- Criação de painéis de gráficos de dispersão (scatter plots), incluindo matrizes de dispersão (scatterplot matrix) para explorar relações entre múltiplas variáveis ao mesmo tempo.

Diferenciais:

- Especializado em gráficos de dispersão múltiplos, podendo usar diferentes eixos em cada painel.
- Permite criar matrizes de dispersão, facilitando a análise visual de correlação entre várias variáveis.
- Menos opções de customização que o SGPLOT, mas mais eficiente para múltiplos scatter plots simultâneos.

```
proc sgscatter data=dataset;
  matrix var1 var2 var3 / diagonal=(histogram normal);
run;
```

VISUALIZAÇÃO E RELATÓRIOS

Comando	Explicação
PROC SGSCATTER <options>;	Inicia o procedimento para criar gráficos de dispersão múltiplos e matrizes de gráficos.
`COMPARE X= variable	(variable-1 … variable-n) Y= variable
MATRIX variable-1 < … variable-n>	Cria uma matriz de gráficos de dispersão entre todas as combinações das variáveis especificadas. Ajuda a explorar relacionamentos entre múltiplas variáveis numéricas.
PLOT plot-request(s)	Permite especificar gráficos individuais ou múltiplos para serem criados no procedimento. Pode incluir gráficos de dispersão com variáveis específicas.

VISUALIZAÇÃO E RELATÓRIOS

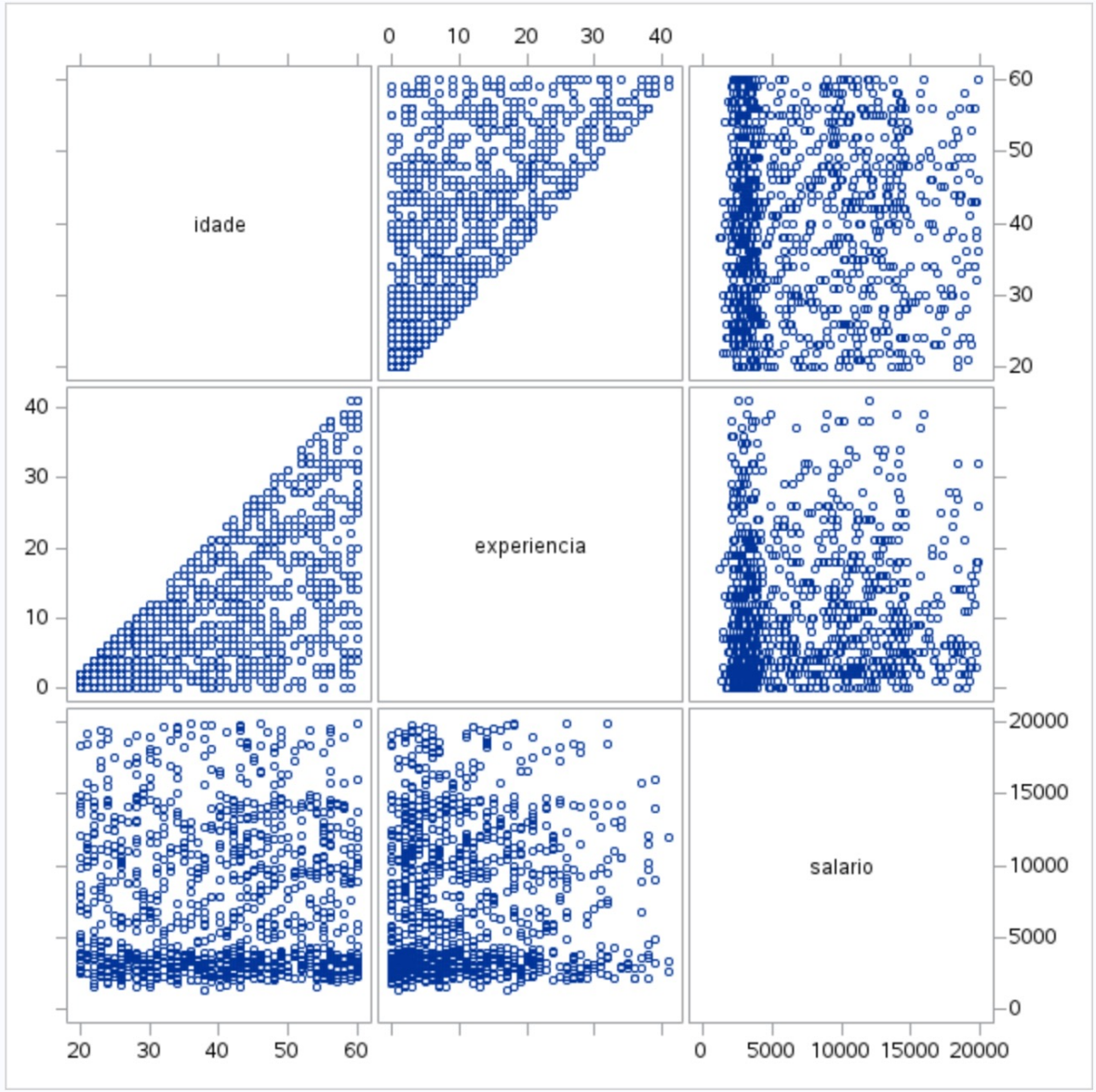
Exercício 50

- PROC SGSCATTER – Criar uma matriz de dispersão para analisar relação entre idade, experiência e salário dos empregados

```
proc sgscatter data=empregados;
  matrix idade experiencia salario;
run;
```

Comando	Explicação
proc sgscatter data=empregados;	Inicia o procedimento SGSCATTER usando o conjunto de dados chamado empregados.
matrix idade experiencia salario;	Cria uma matriz de gráficos de dispersão entre as variáveis numéricas idade, experiencia e salario. Isso gera gráficos de dispersão para todas as combinações possíveis entre essas variáveis, facilitando a análise visual das relações entre elas.
run;	Executa o procedimento, exibindo os gráficos gerados.

VISUALIZAÇÃO E RELATÓRIOS



VISUALIZAÇÃO E RELATÓRIOS

Estrutura da Matriz
- Diagonais principais: Cada célula da diagonal mostra o nome da variável (idade, experiencia, salario). Não há gráfico nesses quadrantes, pois não faz sentido plotar uma variável contra ela mesma.
- Quadrantes fora da diagonal: Cada quadrante mostra um gráfico de dispersão entre duas variáveis diferentes.

Interpretação de Cada Quadrante
1. Quadrantes acima e abaixo da diagonal
Esses quadrantes mostram a mesma relação, mas invertendo os eixos X e Y. Por exemplo:
- Quadrante (linha: idade, coluna: experiencia): idade no eixo Y, experiencia no eixo X.
- Quadrante (linha: experiencia, coluna: idade): experiencia no eixo Y, idade no eixo X.

2. Relações específicas
- Quadrante idade × experiencia = Mostra como a experiência dos empregados varia em função da idade. Normalmente, espera-se uma correlação positiva: quanto maior a idade, maior a experiência, embora o padrão exato dependa dos dados da empresa.
- Quadrante idade × salario = Mostra a relação entre idade e salário. É possível identificar se empregados mais velhos tendem a ter salários maiores, ou se não há relação clara.
- Quadrante experiencia × salario = Mostra a relação entre experiência e salário. Geralmente, espera-se que mais experiência resulte em salários mais altos, mas o gráfico pode revelar se isso ocorre de fato.

Como Ler Cada Quadrante
- Pontos alinhados em diagonal ascendente: Indica correlação positiva entre as variáveis (quando uma aumenta, a outra também tende a aumentar).
- Pontos alinhados em diagonal descendente: Indica correlação negativa (quando uma aumenta, a outra tende a diminuir).
- Pontos dispersos sem padrão: Indica pouca ou nenhuma correlação linear entre as variáveis.

Exemplo Visual
- Quadrante entre idade e experiencia: Os pontos formam uma faixa diagonal, indicando que, quanto maior a idade, maior a experiência, o que é esperado.
- Quadrante entre idade e salario: Os pontos estão mais dispersos, sugerindo que não há uma relação linear forte entre idade e salário.
- Quadrante entre experiencia e salario: Também apresenta dispersão, indicando baixa correlação linear entre experiência e salário.

Resumo

Cada quadrante fora da diagonal mostra a relação entre um par de variáveis. A análise visual permite identificar padrões, tendências e possíveis correlações entre idade, experiência e salário dos empregados, auxiliando na compreensão do perfil dos funcionários e em decisões de gestão de pessoas.

PROJETO FINAL

Exercício FINAL

- Criar uma libname SAS via commando com o seu ultimo nome, apontando para diretoria com o seu ultimo nome;
- (PROC IMPORT) Importar as tabelas de vendas, vendedor, cliente, região e produto (ficheiros disponiveis no GITHUB);
- (DATA STEP e/ou PROC SQL) Criar uma consulta utilizando no minimo 3 tabelas, por exemplo nome do cliente, com quais produtos comprou e o valor total;
- Utilizar pelo menos 5 funções distintas dentro da consulta (Data, Númerica ou Texto);
- (PROC REPORT) Gerar um relatório com todas as colunas com as top 100 vendas;
- (FORECAST) Gerar a previsão de vendas por região para os próximos 6 meses;
- (PROC EXPORT) Exportar o resultado da previsão para a diretoria com seu último nome.

OBRIGADO

WILLIAM ROSARIO

Senior Consultant

william.rosario@habber.com

https://www.linkedin.com/in/warosario/

MADRID
Calle Caleruega 81
28083 Madrid
T. +34 911 255 303
info.es@habber.com

LISBOA
Av Miguel Bombarda 36
1050-165 Lisboa
T. +34 911 255 303
info.es@habber.com

REINO UNIDO
Suite 86, 105 London Street
Reading RG1 4QD.
T. +351 91 294 5904
info.uk@habber.com

MAPUTO
Rua de França 303
Maputo - Moçambique
T. +258 84 957 6534
info.mz@habber.com