

Ctrl+ Pro 01

Aula 11

# Limpeza de Dados de Rede Social



tecnologia e inovação

Uma escola do grupo **CNA**<sup>®</sup>



**Para o Professor**

**<https://www.loom.com/share/0a42a4dad8a4456a8ceae59a0cedba93>**

# Objetivos:

- Ensinar técnicas fundamentais de limpeza de dados usando um dataset de rede social, abordando tipos de dados, tratamento de valores nulos e codificação de categorias.

# Todos com Logados no Portal do Aluno?



# Mineração de Dados



# Para o Professor

O conteúdo dos slides que se seguem (até a parte do primeiro dataframe) está presente na leitura obrigatória pré-aula. Por isso não gaste mais do que 5 minutos em todos estes tópicos, apenas para refrescar a memória da turma.

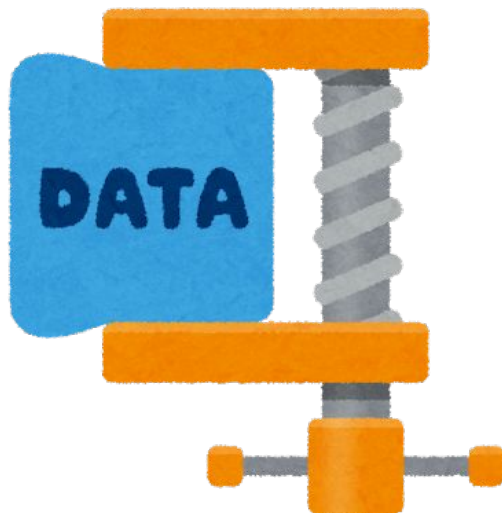




# Data Frame de Rede Social

Nessa primeira parte do projeto, você irá aprender a fazer a limpeza de dados de um arquivo csv.

[CHECKLIST DA AULA](#) - Acompanhe seu progresso [imprimir]



# Dados de Rede Social

Relembrando a leitura pré-aula... Análise de Dados de Redes Sociais

- O que são redes sociais?
- Qual a importância da análise de dados de redes sociais?
- O que são Dados quantitativos e Qualitativos? Exemplos.

Atividade

Revisão



# Professor

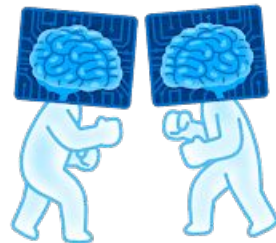
Incentive seus alunos a irem marcando o Checklist conforme você avança na aula. Faça isso junto com a turma



# Introdução à Limpeza de Dados

Dados brutos geralmente possuem inconsistências, valores nulos e formatos variados.

Limpeza de dados é um dos processos mais importantes da Ciência de Dados e Inteligência Artificial.



## Curiosidade:

- Em projetos de IA, cerca de **80% do tempo** é gasto na limpeza e preparação dos dados, e apenas 20% na modelagem!



# Data Frame de Rede Social

Nesse projeto os dados foram coletados através de **Web Scraping** e **APIs**:

Primeiro, você precisará importar o arquivo "Rede Social Ctrlplay.csv" em Data Frame do Pandas.

```
import pandas as pd
df = pd.read_csv('Rede_Social_Ctrlplay.csv')
#pd.read_csv(): Carrega o dataset em um Data Frame do Pandas.
df
```

# Data Frame de Rede Social

CÉLULA 01 DE CÓDIGO COLAB

```
import pandas as pd
df = pd.read_csv('Rede_Social_Ctrlplay.csv')
#pd.read_csv(): Carrega o dataset em um DataFrame do Pandas.
df
```

- Carregamos o dataset `Rede_Social_Ctrlplay.csv` em um DataFrame Pandas.
- Exibimos os primeiros registros para entender a estrutura.



## Contexto:

- Antes de qualquer análise, é essencial visualizar os dados para identificar padrões e possíveis problemas.

# Inspecionando os Dados



CÉLULA 02 DE CÓDIGO COLAB

```
df.info()  
# Usamos a função ".info" para visualizarmos as colunas disponíveis  
no dataframe e o tipo delas
```

## Explicação:

- Exibe informações como tipos de dados e quantidade de valores nulos.
- Ajuda a entender a estrutura da base.



## Curiosidade:

- O tipo de dado influencia nas análises! Exemplo: Modelos de Machine Learning não lidam bem com strings e preferem números.



Professor, os slides que se seguem é apenas uma revisão da leitura pré-aula. Não gaste mais do que 5 minutos apenas repassando os tópicos.





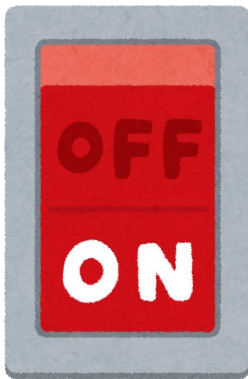
# Tipos de Colunas dataframe

Assim como variáveis em Python, um dataframe pode ter vários tipos de colunas, desde valores numéricos e textos a listas e dicionários. Revisando... [Introdução ao conceito de Bits.](#)



# Introdução ao Conceito de Bits

Um bit é a menor unidade de informação no mundo dos computadores, e ele pode ter apenas dois valores: 0 ou 1. Pense em um interruptor de luz que pode estar apenas em duas posições, ligado (1) ou desligado (0). Esse é o conceito básico de um bit.



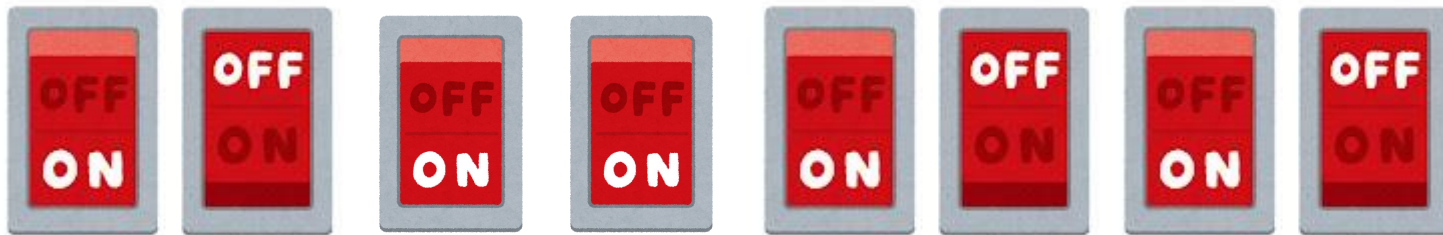
# Introdução ao Conceito de Bits

Agora, imagine que você tem quatro interruptores juntos, ou seja, 4 bits. Com esses quatro interruptores, você pode criar diferentes combinações de "ligado" e "desligado", dando a você 16 possibilidades ( $2^4 = 16$ ). É como ter 16 diferentes sinais de luz que você pode usar para transmitir mensagens.



# Introdução ao Conceito de Bits

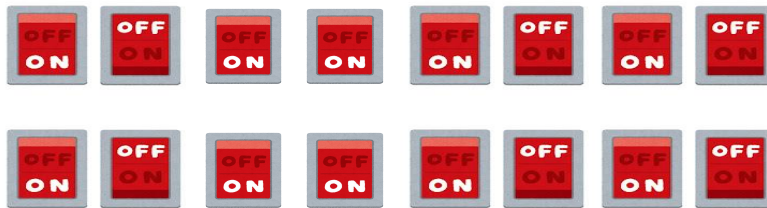
Se aumentarmos para 8 bits (ou um byte), já temos 256 combinações possíveis ( $2^8 = 256$ ). É como se você tivesse 256 diferentes cores de luz para escolher. Nos videogames antigos, por exemplo, gráficos de 8 bits eram usados, permitindo a criação de personagens e cenários com uma quantidade limitada de detalhes.



# Introdução ao Conceito de Bits

Agora, com 16 bits, as combinações saltam para 65.536 ( $2^{16} = 65.536$ ). Aqui, as possibilidades de informação e detalhes crescem muito mais.

Um exemplo de **16 bits** no dia a dia pode ser encontrado em **impressoras domésticas**, especialmente ao lidar com a **impressão de cores**. Impressoras que trabalham com 16 bits por cor (RGB) podem representar até 65.536 tonalidades diferentes para cada cor (vermelho, verde e azul), resultando em impressões com graduações de cor muito mais suaves e precisas.



# Introdução ao Conceito de Bits

Cada vez que dobramos o número de bits, dobramos também a quantidade de informações que podemos representar e processar, permitindo que nossos dispositivos digitais façam coisas cada vez mais complexas!





# Tipos de Colunas em um Dataframe

**Numérico:** como o próprio nome sugere, contém números que seguem a mesma lógica de valores inteiros (int) e flutuantes(float).

**Texto:** String (**object** ou str), tipos de dados que armazenam sequências de caracteres. Utilizados para representar texto ou dados alfanuméricos.

**Booleano:** tipo de dado que armazena apenas dois valores possíveis: True (verdadeiro) ou False (falso). Usado para representar estados binários ou condições lógicas.

**Categorical:** Tipo de dado que representa categorias ou grupos fixos e limitados. Economiza memória ao armazenar categorias como índices em vez de strings completas. Funciona semelhante a um dicionário em Python.

# Tipos de Colunas em um Dataframe

Na base de dados que será usada no projeto, você consegue visualizar três tipos de colunas:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300 entries, 0 to 299
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   usuario_id            300 non-null   int64
1   publicacao_id         300 non-null   int64
2   data_hora              300 non-null   object
3   curtidas               268 non-null   float64
4   comentarios           263 non-null   float64
5   compartilhamentos     277 non-null   float64
6   categoria              300 non-null   object
7   estado                 300 non-null   object
dtypes: float64(3), int64(2), object(3)
memory usage: 18.9+ KB
```

Lembra dos Bits? Olha ali no int64 e float64.

Será que esses números são aleatórios ou tem um motivo?

# Identificando valores nulos

CÉLULA 03 DE CÓDIGO COLAB

```
df.isnull().sum():
```

```
#Retorna a soma dos valores ausentes em cada coluna do DataFrame. Em dados de redes sociais, isso pode incluir campos como curtidas, comentários, compartilhamentos, etc.
```

	0
usuario_id	0
publicacao_id	0
data_hora	0
curtidas	32
comentarios	37
compartilhamentos	23
categoria	0
estado	0
dtype:	int64

## Explicação:

- Conta quantos valores nulos existem em cada coluna.
- Passo essencial para evitar erros em análises futuras.

## Contexto:

- Valores nulos podem indicar erros no sistema, problemas de coleta ou dados ausentes propositalmente.

# Preenchendo Valores Nulos

CÉLULA 04 DE CÓDIGO COLAB

```
for column in df.columns:
    if df[column].isnull().any():
        mean_value = df[column].mean()
        df[column].fillna(mean_value, inplace=True)
```

## Explicação:

- Preenche valores nulos com a média da coluna.
- Essa estratégia é útil para algumas numéricas.

## Contexto:

- Em IA, escolher a melhor estratégia para valores ausentes pode afetar drasticamente o resultado do modelo!

# Tratamento de Dados

```
CÉLULA 04 DE CÓDIGO COLAB  
  
for column in df.columns:  
    if df[column].isnull().any():  
        mean_value = df[column].mean()  
        df[column].fillna(mean_value, inplace=True)
```

Esse laço for percorre todas as colunas do DataFrame df (df.columns) e retorna uma lista dos nomes das colunas no DataFrame.

Para cada coluna, o método df[column].isnull() retorna uma Série booleana onde cada valor indica se o valor correspondente na coluna é nulo (True) ou não (False). O método any() verifica se há pelo menos um valor True na Série, ou seja, se existe pelo menos um valor nulo na coluna.

Se existirem valores nulos na coluna, a média dos valores não nulos é calculada e armazenada em mean\_value. O método mean() calcula a média dos valores na coluna.

O método fillna(mean\_value) substitui todos os valores nulos na coluna pelo valor da média calculada. O parâmetro inplace=True faz com que a modificação seja feita diretamente no DataFrame original df, sem a necessidade de criar uma cópia.

# Separando Dados

```
# Separando dados  
df_object = df.select_dtypes(include=['object'])
```

## Explicação:

- Separamos colunas do tipo **texto** (categóricas) e **numéricas**.
- Necessário porque o tratamento é diferente para cada tipo.



**Curiosidade:** A conversão de texto para números é essencial para IA! Modelos matemáticos não entendem palavras.



# Atividade 01

Crie um df chamado `df_sem_object` e nela armazene os dados do seu df excluindo os tipos 'object' (ou seja, mantendo apenas colunas numéricas). Depois imprima o df com object e o sem object, qual a diferença entre eles?



# Mapeando Estados e Categorias

CÉLULA 07 DE CÓDIGO COLAB

```
# Cria um dicionário para mapear estados para números
estado_map = {estado: i+1 for i, estado in
              enumerate(df_object['estado'].unique())}

# Cria um dicionário para mapear categorias para números
categoria_map = {categoria: i+1 for i, categoria in
                 enumerate(df_object['categoria'].unique())}

# Aplica o mapeamento para criar novas colunas
df_object['estado_num'] = df_object['estado'].map(estado_map)
df_object['categoria_num'] = df_object['categoria'].map(categoria_map)

# Exclui a coluna 'data_hora'
df_object = df_object.drop('data_hora', axis=1)

# Reordena as colunas para colocar as colunas numéricas
# no início
df_object = df_object[['estado_num', 'categoria_num', 'categoria',
                       'estado']]

df_object
```



## Explicação:

- Converter categorias em números
- Criar um dicionário de mapeamento para transformar texto em valores numéricos.



## Contexto:

- Em Machine Learning, essa conversão é chamada de **Label Encoding**.

# Maapeando Estados e Categorias

## Explicação:

- Juntamos os dados categóricos (agora numéricos) com os dados numéricos.
- Esse passo prepara os dados para serem utilizados em modelos de IA!



CÉLULA 08 DE CÓDIGO COLAB

```
# Concatena os DataFrames df_object e df_sem_object
df_concatenado = pd.concat([df_object, df_sem_object], axis=1)

# Exibe o DataFrame concatenado
df_concatenado
```



# Para o professor

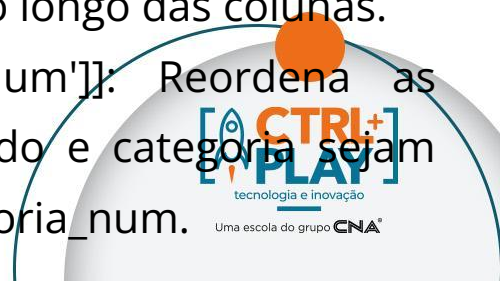
- `df_object['estado'].unique()`: Obtém todos os valores únicos na coluna estado do DataFrame `df_object`.
- `enumerate(...)`: Cria uma sequência de pares (índice, valor) para cada valor único.
- `{estado: i+1 for i, estado in ...}`: Cria um dicionário onde a chave(key) é o valor do estado e o valor(value) é um número inteiro (começando de 1) que corresponde à posição do estado na lista de valores únicos. Isso resulta em um mapeamento de estados para números.



# Para o professor

Similar ao passo anterior, mas para a coluna categoria. Cria um dicionário onde cada categoria é mapeada para um número inteiro.

- `df_object['estado'].map(estado_map)`: Substitui cada valor na coluna estado pelo número correspondente no dicionário estado\_map, criando uma nova coluna estado\_num com esses valores numéricos.
- `df_object['categoria'].map(categoria_map)`: Faz o mesmo para a coluna categoria, criando a coluna categoria\_num.
- `df_object.drop('data_hora', axis=1)`: Remove a coluna data\_hora do DataFrame. O parâmetro axis=1 indica que a operação deve ser realizada ao longo das colunas.
- `df_object[['estado', 'estado_num', 'categoria', 'categoria_num']]`: Reordena as colunas do DataFrame para que as colunas originais estado e categoria sejam seguidas pelas novas colunas numéricas estado\_num e categoria\_num.



# Mapeando Estados e Categorias

CÉLULA 09 DE CÓDIGO COLAB

```
df_concatenado.info()  
# visualizamos a informação do novo dataframe
```

```
estado          0  
estado_num      0  
categoria       0  
categoria_num   0  
usuario_id      0  
publicacao_id   0  
curtidas        0  
comentarios     0  
compartilhamentos 0  
dtype: int64
```

CÉLULA 10 DE CÓDIGO COLAB

```
df_concatenado.isnull().sum()  
#Verificamos se não há nenhum valor nan ainda
```

## Explicação:


- Inspecionamos o DataFrame final para garantir que todos os dados foram limpos corretamente.
- Um dataset bem estruturado reduz o tempo de treino de um modelo de IA em até **50%**!



# Removendo colunas não necessárias

## Explicação:

- Excluimos as colunas que não são mais necessárias, pois já temos as versões numéricas.
- Em projetos de Ciência de Dados, o excesso de colunas pode gerar **"ruído"**, atrapalhando análises e modelos.



```
# Exclui as colunas não necessárias
df_concatenado = df_concatenado.drop(['estado',
    'categoria', 'usuario_id'], axis=1)
# Exibe o DataFrame resultante
df_concatenado
```

# Para o professor

- `df_concatenado.drop(['estado', 'categoria', 'usuario_id'], axis=1)`: Este comando remove as colunas especificadas (estado, categoria, e usuario\_id) do DataFrame `df_concatenado`. O parâmetro `axis=1` indica que a operação deve ser aplicada às colunas.



# Motivo para Excluir essas Colunas

- estado e categoria: Essas colunas foram substituídas por colunas numéricas (estado\_num e categoria\_num) durante o processo de transformação. Agora que você obteve as representações numéricas, as colunas originais (estado e categoria) são redundantes e podem ser removidas para evitar duplicação, mantendo o DataFrame mais limpo e fácil de usar.
- usuario\_id: Embora não tenha sido mencionado explicitamente, a coluna usuario\_id pode ser um identificador único para os usuários. Geralmente, identificadores únicos são removidos durante a análise ou modelagem de dados, pois não fornecem informações úteis para o modelo e podem até introduzir viés ou overfitting.



# Motivo para Excluir essas Colunas

**Obs:** O termo *overfitting*, usado num contexto de *machine learning*, refere-se a um modelo que está viciado em seus dados, como por exemplo uma IA de reconhecimento facial que foi alimentada apenas com rostos de uma etnia e só reconhece faces com um tom de pele específico.



# Conclusão do Tratamento de Dados

A organização dos dados dessa maneira visa:

- **Simplificação:** Remover colunas desnecessárias ou redundantes simplifica o DataFrame e facilita a análise.
- **Preparação para Modelagem:** Em muitos casos, para modelos de machine learning, apenas os dados numéricos ou de interesse são usados. Colunas categóricas são transformadas em numéricas e as colunas não informativas, como IDs ou categorias originais já transformadas, são removidas.
- **Redução de Ruído:** A remoção de identificadores e dados originais não necessários ajuda a evitar a introdução de ruído no processo analítico, garantindo que apenas informações relevantes estejam presentes.



# Para o Professor

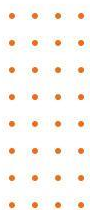
Professor, evite leitura de texto em aula. Discuta com os alunos oralmente os passos a seguir



# Conclusão

## Transformação de Dados Categóricos:

- Criação de Mapeamentos: Convertimos categorias textuais em números usando dicionários de mapeamento, facilitando a inclusão desses dados em modelos de machine learning que exigem entradas numéricas.
- Aplicação de Mapeamentos: Utilizamos o método map para transformar valores categóricos em suas representações numéricas correspondentes, preparando o DataFrame para análises quantitativas.



# Conclusão

## Concatenação e Organização dos Dados:

- Concatenação de DataFrames: Combinamos diferentes partes do DataFrame para integrar colunas numéricas e categóricas transformadas com as demais informações do DataFrame original.
- Reordenação e Limpeza: Após a concatenação, reorganizamos e limpamos o DataFrame para manter apenas as colunas necessárias e eliminar redundâncias.





# Conclusão

O processo que vimos é crucial para a preparação adequada de dados para análise e modelagem. Ao:

- Transformar Dados Categóricos: Convertendo categorias para números, facilitamos a integração desses dados em análises quantitativas e modelos preditivos.
- Excluir Colunas Redundantes: Garantimos que o DataFrame esteja livre de informações desnecessárias que possam atrapalhar a análise.
- Concatenar e Organizar Dados: Preparamos um conjunto de dados coeso e limpo, pronto para análises mais avançadas ou construção de modelos de machine learning.

Este trabalho de preparação é um passo fundamental na análise de dados, pois assegura que os dados estejam em um formato que maximize a eficácia dos modelos e análises subsequentes.



# Tarefas

Tarefas do Astro

Avaliação da Aula



# Materiais

Colab Professor

Colab Aluno

