Módulo 1 - Manipulação e visualização de dados

Aula 01 — A história dos computadores

A palavra computador, antigamente, era utilizada para se referir às pessoas que computavam e, um pouco antes dos computadores que conhecemos hoje ficarem populares, as pessoas que faziam cálculos eram chamadas de "computadores". A NASA, por volta do ano de 1950, tinha uma sala com vários computadores, na sua maioria mulheres, que passavam o dia calculando a trajetória de foguetes.

Em 1936, o matemático Alan Turing, propôs um modelo teórico de computador programável capaz de simular qualquer forma de computação algorítmica. Esse modelo ficou conhecido como **Máquina de Turing**. Durante a Segunda Guerra Mundial ele foi convidado junto a um grupo pequeno de mentes brilhantes, para desvendar as cartas com mensagens escondidas que os alemães usavam para se comunicar. Neste período, utilizando recursos do governo britânico, construiu uma máquina que se enquadrava ao modelo que ele propôs anos atrás. Essa máquina ficou conhecida como **Electronic Numerical Integrator and Computer (ENIAC)**, em português, Computador e Integrador Eletrônico Numérico, e o ano de 1946 ficou conhecido como o ano em que o primeiro computador surgiu.

Áreas de atuação e carreira em Machine Learning

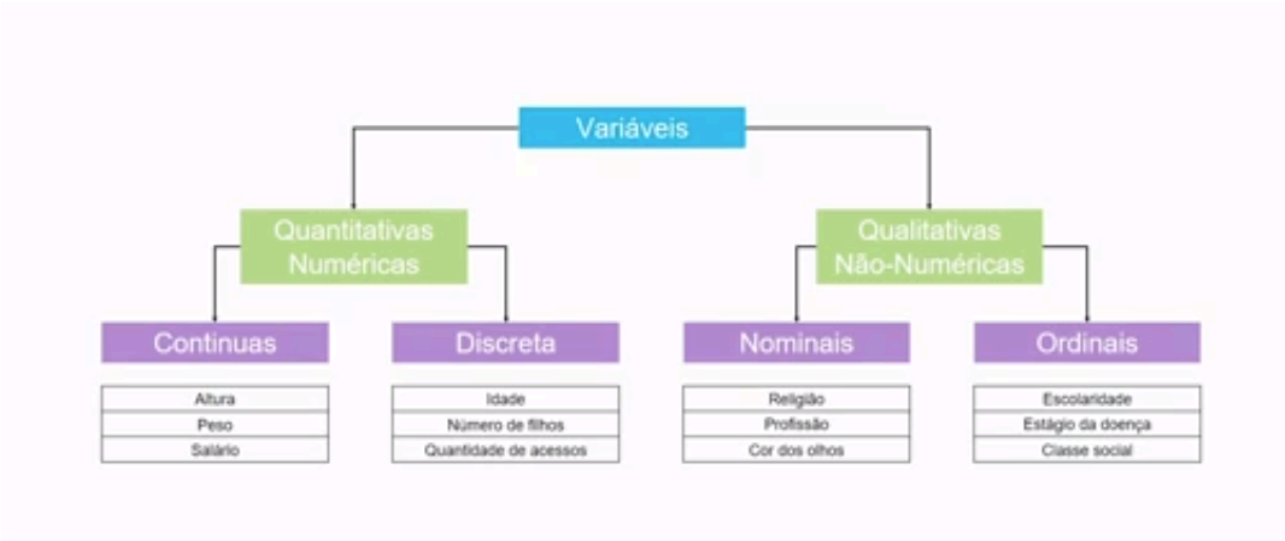
A área de Aprendizado de Máquina é uma área em constante expansão, e existem diversas carreiras relacionadas a ela. Os profissionais podem atuar em várias carreiras:

Engenharia de Dados - Coleta, transformação, carregamento
Engenheiro de Aprendizado de Máquina - Projetar, construir, implementar;
Cientista de Dados - Estatística, análise, tomada de decisões.

Aula 02 - Introdução a dados

Uma amostra de dados é um subconjunto de dados coletados que representa um universo maior de dados. O objetivo de coletar uma amostra é inferir informações sobre o universo maior de dados, sem ter que analisar todo o conjunto de dados, o que muitas vezes é inviável.

Os dados podem ser classificados em:
Dados numéricos: são aqueles que podem ser quantificados em uma escala numérica, como altura, peso, temperatura, etc;
Dados categóricos: são aqueles que não possuem uma ordem ou hierarquia natural, como cor, sexo, raça, etc;
Dados ordinais: possuem uma ordem ou hierarquia natural, como nível educacional, posição em uma corrida, entre outros.



Dados Tabulares

Podemos considerar como dados tabulares informações organizadas em tabelas com colunas e linhas, onde cada coluna representa uma variável e cada linha corresponde a uma observação, registro ou valor. Os formatos mais comuns de tabela são o **CSV (comma-separated values) e o Excel.** Entre as características dos dados tabulares, incluem-se:

Tipo do dado (numérico, categórico, texto);
Presença de valores ausentes;
Distribuição dos valores.
Devido ao seu formato fácil de entender e manipular, a maioria dos dados disponíveis publicamente estão armazenados dessa forma.

Operações Comuns

As operações com tabelas são cruciais em todas as fases da análise de dados. A seguir, são resumidas algumas das operações mais comumente utilizadas durante o seu manuseamento.

"Lembre-se que o aprendizado é uma jornada longa, mas recompensadora, onde nela podemos desenvolver novos conhecimentos e abrir novas portas para o futuro".

`import pandas as pd` — Importa a biblioteca pandas, que é uma ferramenta poderosa para manipulação e análise de dados em formato tabular (dataframes).

`tabela = pd.read_csv('../../data/aulas/Matriculados.csv')` — Lê um arquivo CSV chamado "Matriculados.csv" localizado no diretório ../../data/aulas/ e armazena em um dataframe chamado tabela.

`tabela.head()`- Exibe as primeiras cinco linhas do dataframe

`Tabela['nome']- Retorna a coluna chamada 'nome' do dataframe`

`Tabela.iloc[:,:]` - Retorna todas as linhas e todas as colunas do dataframe

`n_filhos = [0, 0, 1, 2]`
`tabela['n_filhos'] = n_filhos` - Adiciona uma nova coluna chamada 'n_filhos' ao dataframe tabela, preenchida com os valores da lista n_filhos

`tabela.drop(['n_filhos'], axis=0)` - Tenta remover linhas do dataframe tabela usando a coluna 'n_filhos' como referência. No entanto, isso resultaria em erro porque axis=0 indica remoção de linhas e a coluna 'n_filhos' não é uma linha.
`tabela.drop(['n_filhos'], axis=1)` - Remove a coluna 'n_filhos' do dataframe

`tabela = pd.read_csv('../../data/aulas/temperaturas.csv', index_col=['dia'])` - Lê um arquivo CSV chamado "temperaturas.csv" localizado no diretório ../../data/aulas/ e define a coluna 'dia' como índice do dataframe tabela.
`tabela.head(7)` - Exibe as primeiras sete linhas do dataframe

`Tabela.loc['sex','temperatura']` - Retorna o valor da coluna 'temperatura' na linha cujo índice é 'sex' no dataframe

`dias = ['seg', 'qui', 'dom']`
`info = ['clima', 'vento']`
`tabela.loc[dias, info]` - Retorna as colunas 'clima' e 'vento' para as linhas cujos índices são 'seg', 'qui', 'dom' no dataframe

`tabela.loc['ter':'qui', ['temperatura', 'vento']]` - Retorna as colunas 'temperatura' e 'vento' para o intervalo de linhas entre 'ter' e 'qui' no dataframe

`tabela.loc[tabela.umidade > 50,['clima']]` - Retorna a coluna 'clima' para as linhas onde o valor da coluna 'umidade' é maior que 50 no dataframe

`tabela.loc[(tabela.umidade > 50) & (tabela.clima == 'chuva'),:]` - Retorna todas as colunas para as linhas onde 'umidade' é maior que 50 e 'clima' é 'chuva' no dataframe

`tabela.loc[(tabela.umidade > 50) | (tabela.clima == 'nublado'),:]` - Retorna todas as colunas para as linhas onde 'umidade' é maior que 50 ou 'clima' é 'nublado' no dataframe

`tabela.loc[(tabela.umidade > 50) & (tabela.clima == 'nublado'),:]` - Retorna todas as colunas para as linhas onde 'umidade' é maior que 50 e 'clima' é 'nublado'.

`tabela.loc[(tabela.umidade > 50) | (tabela.clima == 'nublado'),['temperatura', 'umidade']]` - Retorna as colunas 'temperatura' e 'umidade' para as linhas onde 'umidade' é maior que 50 ou 'clima' é 'nublado'.

```
voos = pd.read_csv('../../data/aulas/juncao_tabelas/voos.csv')
voos.head()
```

```
qnt_voos =
pd.read_csv('../../data/aulas/juncao_tabelas/quatidade_de_voos.csv')
qnt_voos.tail()
```

`pd.merge(voos, qnt_voos, on='month')` - Junta os dataframes voos e qnt_voos com base na coluna 'month'. Os dados das duas tabelas serão combinados de acordo com os valores dessa coluna.

```
produtos1 =
pd.read_csv('../../data/aulas/juncao_tabelas/produtos1.csv',
index_col=['id'])
produtos1.head()
```

```
produtos2 =
pd.read_csv('../../data/aulas/juncao_tabelas/produtos2.csv',
index_col=['id'])
produtos2.head()
```

`produtos = pd.concat([produtos1, produtos2])` - Concatena (junta) os dataframes produtos1 e produtos2, criando um novo dataframe chamado produtos.

```
produtos.head()
```

```
vendas = pd.read_csv('../../data/aulas/juncao_tabelas/vendas.csv')
vendas.head()
```

`vendas = vendas.join(produtos, on='produto_id')` - Junta o dataframe vendas com produtos usando a coluna 'produto_id' como referência. A operação adiciona ao dataframe vendas informações do dataframe produtos para registros onde o 'produto_id' coincide.

```
vendas.head()
```

`vendas['total'] = vendas.quantidade * vendas.preco` - Cria uma nova coluna chamada 'total' que calcula o valor total para cada venda multiplicando a quantidade pelo preço.

```
vendas.head()
```

```
voos = pd.read_csv('../../data/aulas/voos.csv')
voos.head()
```

`pd.pivot_table(voos, index=['year', 'month'])` - Cria uma tabela dinâmica a partir do dataframe voos, com índices em duas colunas, 'year' e 'month'. Uma tabela dinâmica permite reorganizar e resumir dados de várias maneiras.

`pd.pivot_table(voos, index=['month', 'year'])` - Cria uma tabela dinâmica a partir do dataframe voos, mas agora com 'month' como primeiro índice e 'year' como segundo índice.

`voos_por_ano = voos.groupby('year')`- Agrupa os dados do dataframe voos pela coluna 'year'. Este agrupamento permite operações como calcular médias, somar, contar, etc., dentro de cada grupo.

`voos_por_ano.describe()` - Retorna estatísticas descritivas (como média, mínimo, máximo, etc.) para cada grupo do dataframe

`voos_por_ano['passengers'].mean()` - Calcula a média do número de passageiros para cada grupo por ano.

`voos_por_mes = voos.groupby('month')` - Retorna o dataframe agrupado por mês.
`voos_por_mes`

`voos_por_mes.describe()` - Retorna estatísticas descritivas para cada grupo do dataframe

`voos_por_mes['passengers'].mean()` - Calcula a média do número de passageiros para cada grupo por mês.

`pd.pivot_table(voos, index=['month', 'year']).transpose()` - Cria uma tabela dinâmica como antes, mas agora transposta, o que significa que as linhas e colunas são invertidas.

`voos.transpose()`- Transpõe o dataframe voos, trocando as linhas pelas colunas.

Pipeline

O Pipeline de dados é composto por etapas que vão desde a coleta até a análise de dados. O processo começa com a definição do problema e dos objetivos do estudo, seguido pela coleta de dados de diversas fontes, como bancos de dados, dispositivos IoT, arquivos CSV, entre outros. É importante garantir que os dados coletados sejam confiáveis, completos e diversificados, para poderem representar adequadamente o fenômeno estudado.

O objetivo final é extrair insights e conhecimento a partir dos dados, que possam ser aplicados em tomadas de decisão, otimização de processos ou desenvolvimento de novos produtos e serviços.

Coleta de Dados

A coleta de dados é uma etapa crucial para ML, por ser a partir dos dados que os algoritmos são treinados para fazer previsões e tomar decisões. Esses dados podem ser obtidos de diversas fontes, como:

Bancos de dados internos da empresa;

Dados públicos disponíveis na internet;

Sensores ou dispositivos IoT (Internet of Things).

É importante que os dados coletados sejam relevantes para o problema em questão e que sejam de qualidade, ou seja, que não contenham erros ou informações duplicadas.

Informação dos Dados

Podemos obter várias informações, muitas vezes chamadas de insights, analisando conjuntos de dados. É através destes insights que encontramos tendências, padrões e correlações no nosso conjunto. Além disso, podemos usar técnicas de análise de dados para extrair diversas informações mais complexas.

`import seaborn as sns` - Importa a biblioteca Seaborn, uma biblioteca de visualização de dados que funciona bem com pandas e é ótima para criar gráficos estatísticos.
`import pandas as pd`

`df = sns.load_dataset('titanic')`- Carrega um conjunto de dados do Titanic (um dos conjuntos de dados integrados no Seaborn) e armazena em um dataframe chamado df.
`df.head()`

`df['sex'].value_counts()`- Conta a quantidade de valores distintos na coluna 'sex', mostrando o número de passageiros masculinos e femininos no dataset do Titanic.

`df['embark_town'].value_counts()` – Conta a quantidade de valores distintos na coluna 'embark_town', que representa a cidade de embarque dos passageiros no Titanic.

`df.info()` - Fornece uma visão geral do dataframe df, incluindo o número de linhas e colunas, tipos de dados de cada coluna e número de valores nulos.

`df.describe()` - Retorna estatísticas descritivas para colunas numéricas, como contagem, média, desvio padrão, valores mínimo e máximo, e percentis.

`df.isnull()` - Cria um dataframe booleano onde cada célula indica se o valor correspondente no dataframe original é nulo (True) ou não (False).

`df.isnull().any()` - Retorna um resultado booleano para cada coluna, indicando se a coluna possui algum valor nulo.

`sns.heatmap(df.isnull())`- Cria um mapa de calor para visualizar os valores nulos no dataframe df. As áreas mais claras indicam presença de valores nulos.

`df = sns.load_dataset('iris')` - Carrega o conjunto de dados da íris (também integrado no Seaborn) e armazena em um dataframe chamado df.

`numeric_cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']` - Define uma lista chamada numeric_cols com as colunas numéricas do dataframe df.
`df[numeric_cols].corr()` - Calcula a matriz de correlação para as colunas numéricas no dataframe df. A correlação mostra a relação entre duas variáveis.

```
def quadrado(x):
    return x * x
```
- Define uma função chamada quadrado que retorna o quadrado de um número x.
`df['sepal_length'].apply(quadrado)` - Aplica a função quadrado a cada elemento da coluna 'sepal_length', retornando uma nova série com os valores ao quadrado.

`df['sepal_length'].apply(lambda x : x * x)` - Aplica uma função lambda (anônima) para retornar o quadrado de cada valor na coluna 'sepal_length'.

`df['sepal_length'] = df['sepal_length'].apply(lambda x : x * x)` - Atualiza a coluna 'sepal_length' para conter os valores ao quadrado, usando a função lambda.
`df.head()`

`df.sort_values(by='petal_width')` - Classifica o dataframe df pela coluna 'petal_width' em ordem crescente.

`df.sort_values(by=['petal_width', 'petal_length'])` - Classifica o dataframe df por duas colunas, primeiro por 'petal_width' e depois por 'petal_length', ambas em ordem crescente.

Pré-processamento dos Dados

O pré-processamento de dados consiste em uma série de técnicas aplicadas aos dados antes que eles possam ser utilizados para a análise estatística e desenvolvimento de soluções

de problemas. O objetivo do pré-processamento também é melhorar a qualidade dos dados e torná-los mais adequados para serem utilizados pelos modelos.

Valores Faltantes

```
import piplite
await piplite.install('seaborn')

import pandas as pd
import seaborn as sns
import numpy as np

titanic = pd.read_csv('../../data/aulas/titanic.csv')
titanic.head()
titanic.info()

sns.heatmap(titanic.isna())

titanic.deck.isna().value_counts()

titanic.age.isna().value_counts()

titanic.embarked.isna().value_counts()

titanic.dropna( subset=['embarked', 'embark_town'], inplace=True)

titanic.deck.value_counts()

titanic.deck.fillna('C')

Titanic.deck

titanic.deck.fillna('C').value_counts()

titanic.drop(['deck'], axis=1, inplace=True)
titanic.info()

titanic.age.mean()

titanic.age.replace(np.NaN, titanic.age.mean())

Titanic.age

titanic['age'] = titanic.age.replace(np.NaN, titanic.age.mean())

sns.heatmap(titanic.isna())
```

Transformando Informações

```
titanic = pd.read_csv('../../data/aulas/titanic.csv')
titanic.head()

Titanic.sex
pd.get_dummies(titanic.sex)

sex = pd.get_dummies(titanic.sex)

titanic[['female', 'male']] = sex

titanic.drop(['sex'], axis=1, inplace=True)

titanic.drop(['male'], axis=1, inplace=True)

titanic.loc[(titanic.female == 0) & (titanic.who == 'woman') |
(titanic.female == 1) & (titanic.who == 'man')]

titanic.loc[(titanic.survived == 0) & (titanic.alive == 'yes') |
(titanic.survived == 1) & (titanic.alive == 'no')]

titanic.drop(['who', 'alive'], axis=1, inplace=True)

pd.get_dummies(titanic.embark_town)

titanic[['Queenstown', 'Southampton']] =
pd.get_dummies(titanic.embark_town).drop(['Cherbourg'], axis=1)

titanic.drop(['embark_town'], axis=1, inplace=True)
```

Normalização

```
iris = pd.read_csv('../../data/aulas/iris.csv')
iris.head()

iris.species.value_counts()

iris.describe()

from sklearn.preprocessing import MinMaxScaler
Iris.columns

numerical_columns = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width']
```

```
iris[numerical_columns].head()

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(iris[numerical_columns])

Scaled_data

scaled_data = pd.DataFrame(scaled_data, columns = numerical_columns)
scaled_data.head()

scaled_data.describe()

iris.describe()
```

Ética de Dados

A ética de dados envolve o tratamento justo e transparente das informações, incluindo a **coleta, armazenamento, processamento, análise e compartilhamento de dados**. Isso inclui a necessidade de obter consentimento informado, proteger a privacidade e a segurança dos dados, e evitar discriminação ou viés em relação às informações coletadas.

Sensibilidade de Dados

A sensibilidade de dados é a consideração das implicações sociais e políticas que os dados podem ter. Ela aborda questões como o uso adequado dos dados para evitar danos e preservar os direitos e a privacidade das pessoas e comunidades envolvidas. Além disso, a sensibilidade de dados também envolve garantir que os dados coletados representem de forma justa e precisa as populações e comunidades, levando em conta questões de inclusão e diversidade.

Visualizações fundamentais

A visualização de dados é uma técnica utilizada para representar informações em gráficos, tabelas e outros formatos visuais que facilitam entender e analisar grandes quantidades de dados.

```
import seaborn as sns
import pandas as pd

data_dic = {'Data': [15, 16, 17, 18, 19, 20, 21],
```

```
            'Dolar': [5.8567, 5.8567, 5.5851, 5.7201, 5.7557, 5.638,
5.5917]}

df = pd.DataFrame(data_dic, index=None)

sns.lineplot(df, x='Dolar', y='Data')

sns.lineplot(df, x='Data', y='Dolar')

sns.barplot(df, x='Dolar', y='Data')

sns.barplot(df, x='Data', y='Dolar')

df = sns.load_dataset('iris')
df.head()

sns.pairplot(df, hue='species')

sns.scatterplot(df,  x='petal_length',  y='petal_width',  hue='species',
style='species')

import matplotlib.pyplot as plt
import seaborn as sns
# Set style
sns.set(style="whitegrid", color_codes=True)
# Load some sample data
titanic = sns.load_dataset("titanic")
# Make the barplot
bar = sns.barplot(x="sex", y="survived", hue="class", data=titanic);
# Define some hatches
hatches = ['-', '+', 'x']
# Loop over the bars
for i,thisbar in enumerate(bar.patches):
    # Set a different hatch for each bar
    thisbar.set_hatch(hatches[i%3])
plt.show()
```

Filmes:

- **Jogo da imitação:** Conta a historia de Alan Turing precursor dos estudos sobre inteligência artificial e sua contribuição para a invenção do computador.
- **Estrelas além do tempo:** Mostra a história de três matemáticas que se tornam pioneiras na programação.