Projeto de Ciência de Dados

Professor: Luciano Barbosa

2024.1

Grupo:

- Camila Barbosa Vieira (cbv2)
- Luis Felipe Rodrigues de Oliveira (lfro2)

Objetivo

O objetivo deste trabalho é analisar como as condições climáticas afetam a partida ou não dos voos nos dez maiores aeroportos do Brasil. Para isso, coletaremos dados sobre o status dos voos e as condições climáticas correspondentes durante um período de 30 dias, a partir de 7 de abril de 2024.

Coleta de Dados

Dados dos Voos

Utilizaremos o site Avionio (www.avionio.com) para coletar informações sobre os voos, incluindo:

- Horário (Time)
- Data (Date)
- Código IATA do aeroporto (IATA)
- Destino (Destination)
- Número do voo (Flight)
- Companhia aérea (Airline)
- Status do voo (Status)
- Origem (Origin)

A coleta será feita por meio de um web crawler utilizando a biblioteca BeautifulSoup, que permitirá extrair essas informações das classes HTML correspondentes.

Dados Climáticos

Os dados climáticos serão obtidos por meio da API da Open-Meteo (https://api.open-meteo.com/v1/forecast). As variáveis climáticas a serem coletadas incluem:

- Temperatura a 2 metros (temperature_2m)
- Umidade relativa a 2 metros (relative_humidity_2m)
- Ponto de orvalho a 2 metros (dew_point_2m)
- Temperatura aparente (apparent_temperature)

- Probabilidade de precipitação (precipitation_probability)
- Precipitação (precipitation)
- Chuva (rain)
- Pancadas de chuva (showers)
- Queda de neve (snowfall)
- Pressão ao nível do mar (pressure_msl)
- Cobertura de nuvens (cloud_cover)
- Visibilidade (visibility)
- Velocidade do vento a 10 metros (wind_speed_10m)
- Direção do vento a 10 metros (wind_direction_10m)
- Rajadas de vento a 10 metros (wind_gusts_10m)

Análise

A partir dos dados coletados, realizaremos uma análise para identificar possíveis correlações entre as condições climáticas e o status dos voos (atrasos, cancelamentos, etc.). Essa análise permitirá entender melhor como diferentes variáveis climáticas podem impactar as operações de voo nos principais aeroportos do Brasil.

Imports e Downloads

```
import re
import requests
import numpy as np
import pandas as pd
import seaborn as sns
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
from sklearn.impute import KNNImputer
from datetime import datetime, timedelta
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import ttest_ind, mannwhitneyu, shapiro
```

Coleta de Dados

Web Crawler

- Integrar dados ou extrair dados da Web

Voos

Descrição

O link "https://www.avionio.com/en/airport/{cidade}/departures?ts={initial+(day*i)}&page={page}" direciona para a página correta do Avionio, permitindo a coleta dos dados dos voos. Para isso, é necessário substituir:

- **{cidade}**: pela sigla IATA do aeroporto de partida.
- **{initial}**: pelo timestamp correspondente ao dia inicial do período desejado.
- **{day}**: pelo valor representativo de um dia em milissegundos (86400000 ms).
- **{i}**: pelo número de dias a partir da data inicial.
- **{page}**: pelo número da página, começando do zero.

Detalhamento

1. **Sigla IATA do Aeroporto**:
   - Substitua **{cidade}** pela sigla IATA do aeroporto de partida. Por exemplo, "GRU" para o Aeroporto Internacional de São Paulo/Guarulhos.
2. **Timestamp do Dia Inicial**:
   - **{initial}** representa o timestamp do dia inicial do período. Por exemplo, para 7 de abril de 2024, você precisa converter essa data para timestamp em milissegundos.
3. **Incremento Diário**:
   - **{day}** é o incremento diário de 86400000 milissegundos, equivalente a um dia. Para avançar para o próximo dia, você adiciona esse valor ao timestamp inicial.
4. **Número do Dia**:
   - **{i}** é o número do dia a partir da data inicial. Para o dia inicial, **{i}** é 0; para o segundo dia, **{i}** é 1, e assim por diante.
5. **Número da Página**:
   - **{page}** representa o número da página de resultados, começando do zero. Para obter todos os voos de um dia, é necessário iterar pelas páginas até que todos os dados sejam coletados.

Exemplo

Para coletar dados de partidas do Aeroporto Internacional de São Paulo/Guarulhos (GRU) no dia 7 de abril de 2024:

1. **Sigla do aeroporto**: "GRU"
2. **Timestamp do dia inicial**: suponha que seja "1712484000000".
3. **Valor diário em milissegundos**: 86400000
4. **Número do dia (i)**: 0 (para o dia inicial)
5. **Página**: 0 (primeira página de resultados)

O link ficaria:

```
https://www.avionio.com/en/airport/GRU/departures?
ts=1712484000000&page=0
```

Para coletar dados do dia seguinte (8 de abril de 2024), o timestamp seria incrementado por 86400000:

```
https://www.avionio.com/en/airport/GRU/departures?
ts=1712570400000&page=0
```

Continuando dessa maneira, é possível coletar os dados para cada dia do período de 30 dias, alterando o valor de **{i}** e o timestamp correspondente. Para cada dia, itere pelas páginas até que todos os voos sejam coletados.

```
initial = 1712484000000 #timestamp para o dia 07 Apr
day = 86400000          #timestamp para duração de um dia
total_days = 30         #total de dias coletados

cidades = ['GRU', 'CGH', 'BSB', 'GIG', 'CNF', 'VCP', 'SDU', 'REC',
'POA', 'SSA']           #Dez maiores aeroportos do Brasil
categorias = ["Time", "Date", "IATA code", "Destination", "Flight",
"Airline", "Status"]   #Categorias do df
classes = ['tt-t', 'tt-d', 'tt-i', 'tt-ap', 'tt-f', 'tt-al', 'tt-s']
#Classes HTML para coletar os valores necessários
regex = r"([a-zA-Z0-9: ])"
#Regex para apenas pegar letras e números dos valores necessários,
eliminando espaços e tabulações...

df = pd.DataFrame()
#Criação do dataframe vazio

for cidade in cidades:
  print(cidade)
  for i in range(total_days):
    data = {}
    date = datetime.strptime('07 Apr 2024', '%d %b %Y') +
timedelta(days=i)
    previous_date = datetime.strptime('07 Apr 2024', '%d %b %Y') +
timedelta(days=i)
    next_date = datetime.strptime('07 Apr 2024', '%d %b %Y') +
timedelta(days=i)
    page = 0

    while(previous_date == date):
      url = f"https://www.avionio.com/en/airport/{cidade}/departures?
ts={initial + (day*i)}&page={page}"
      response = requests.get(url)
      if response.status_code == 200:
        html_content = response.content
        soup = BeautifulSoup(html_content, 'lxml')

        for classe, categoria in zip(classes, categorias):
          data_elements = soup.find_all('td', class_= classe)
          data[categoria] = []
          for element in data_elements:
            elements = []
            description = element.text
            elements.append(re.findall(regex, description))
            data[categoria].append(("".join(elements[0]).strip()))
```

```
        day_df = (pd.DataFrame(data))

        day_df["Origin"] = cidade
        df = pd.concat([df, day_df], ignore_index=True)

        date_string = data['Date'][-1] + ' 2024'
        previous_date = datetime.strptime(date_string, '%d %b %Y')
        page -= 1
      else:
        print(f"Erro ao acessar o site: {response.status_code}")

    page = 1
    while(next_date == date):
      url = f"https://www.avionio.com/en/airport/{cidade}/departures?
ts={initial + (day*i)}&page={page}"
      response = requests.get(url)
      if response.status_code == 200:
        html_content = response.content
        soup = BeautifulSoup(html_content, 'lxml')

        for classe, categoria in zip(classes, categorias):
          data_elements = soup.find_all('td', class_= classe)
          data[categoria] = []
          for element in data_elements:
            elements = []
            description = element.text
            elements.append(re.findall(regex, description))
            data[categoria].append(("".join(elements[0]).strip()))

        day_df = (pd.DataFrame(data))
        day_df["Origin"] = cidade
        df = pd.concat([df, day_df], ignore_index=True)

        date_string = data['Date'][-1] + ' 2024'
        next_date = datetime.strptime(date_string, '%d %b %Y')
        page += 1
      else:
        print(f"Erro ao acessar o site: {response.status_code}")
df
```

```
GRU
CGH
BSB
GIG
CNF
VCP
SDU
REC
POA
SSA
```

```
{"type":"dataframe","variable_name":"df"}
```

```
df.to_csv('/content/avionio.csv', index=False)
```

```
len(df['IATA code'].unique())
```

```
162
```

```
df['IATA code'].unique()
```

```
array(['CWB', 'MAO', 'MDZ', 'SCL', 'CXJ', 'CNF', 'JPA', 'POA', 'AEP',
       'MCZ', 'AJU', 'MAD', 'REC', 'SSA', 'VVI', 'GIG', 'SLZ', 'NAT',
       'BPS', 'VIX', 'FOR', 'CGB', 'IGU', 'GYN', 'BSB', 'UIO', 'JJD',
       'RAO', 'NVT', 'FLN', 'BEL', 'PDP', 'LDB', 'ASU', 'ATL', 'SDU',
       'MIA', 'PTY', 'MCO', 'IOS', 'MVD', 'CGR', 'CAC', 'MEX', 'THE',
       'IZA', 'RVD', 'OPS', 'MGF', 'UDI', 'MOC', 'JJG', 'PNZ', 'LIM',
       'XAP', 'PMW', 'SJP', 'JOI', 'BOG', 'ADD', 'EZE', 'IST', 'SDQ',
       'FRA', 'VDC', 'LHR', 'LIS', 'DXB', 'DOH', 'JDO', 'JFK', 'LAX',
       'IMP', 'FCO', 'PFB', 'CDG', 'PPB', 'PET', 'MXP', 'BCN', 'LAD',
       'ZRH', 'YYZ', 'EWR', 'IAD', 'AMS', 'IAH', 'DFW', 'ORD', 'BOS',
       'AAX', 'JNB', 'VCP', 'YUL', 'UNA', 'DSS', 'GEL', 'OPO', 'PVH',
       'CPT', 'PUJ', 'CGH', 'BEY', 'COR', 'RRJ', 'UBA', 'CLV', 'IPN',
       'ARU', 'BYO', 'GRU', 'BVB', 'AUX', 'RBR', 'MCP', 'MAB', 'STM',
       'BRA', 'FLL', 'ROS', 'CPV', 'SJK', 'LUX', 'TFL', 'GVR', 'CKS',
       'LEC', 'LHN', 'VAG', 'JMA', 'CFB', 'GNM', 'POJ', 'CUR', 'ATM',
       'CCP', 'PGZ', 'PMG', 'MII', 'TJL', 'JTC', 'CMG', 'ORY', 'CAW',
       'ROO', 'GPB', 'MDE', 'MEM', 'SJU', 'MVF', 'FEN', 'FEC', 'PAV',
       'CAU', 'SET', 'LPA', 'RIA', 'URG', 'BGX', 'CSU', 'SRA', 'ALQ'],
      dtype=object)
```

Clima

A partir dos aeroportos listados no DataFrame de voos, foi criada uma lista de dicionários contendo a sigla do aeroporto, bem como suas coordenadas de latitude e longitude. Posteriormente, utilizaremos essas informações para recuperar os dados climáticos de todas essas localidades durante o período especificado. O objetivo final é unir esses dados climáticos com a tabela de voos, criando assim um conjunto completo de informações que relacionam voos e condições climáticas.

```
latitudes_longitudes = [
  {"sigla": "CWB", "latitude": -25.5285, "longitude": -49.1758},
  {"sigla": "MAO", "latitude": -3.0386, "longitude": -60.0497},
  {"sigla": "MDZ", "latitude": -32.832, "longitude": -68.8272},
  {"sigla": "SCL", "latitude": -33.393, "longitude": -70.7858},
  {"sigla": "CXJ", "latitude": -29.1971, "longitude": -51.1875},
  {"sigla": "CNF", "latitude": -19.6244, "longitude": -43.9719},
  {"sigla": "JPA", "latitude": -7.1458, "longitude": -34.9489},
  {"sigla": "POA", "latitude": -29.9939, "longitude": -51.1711},
  {"sigla": "AEP", "latitude": -34.558, "longitude": -58.4155},
  {"sigla": "MCZ", "latitude": -9.5108, "longitude": -35.7917},
```

```
{"sigla": "AJU", "latitude": -10.9853, "longitude": -37.0703},
{"sigla": "MAD", "latitude": 40.4983, "longitude": -3.5676},
{"sigla": "REC", "latitude": -8.1264, "longitude": -34.9236},
{"sigla": "SSA", "latitude": -12.9086, "longitude": -38.3225},
{"sigla": "VVI", "latitude": -17.6447, "longitude": -63.1354},
{"sigla": "GIG", "latitude": -22.8089, "longitude": -43.2436},
{"sigla": "SLZ", "latitude": -2.5863, "longitude": -44.2369},
{"sigla": "NAT", "latitude": -5.7681, "longitude": -35.3761},
{"sigla": "VIX", "latitude": -20.2581, "longitude": -40.2865},
{"sigla": "BPS", "latitude": -16.4397, "longitude": -39.0808},
{"sigla": "CGB", "latitude": -15.6529, "longitude": -56.1172},
{"sigla": "IGU", "latitude": -25.5951, "longitude": -54.4872},
{"sigla": "FOR", "latitude": -3.7763, "longitude": -38.5326},
{"sigla": "GYN", "latitude": -16.6294, "longitude": -49.2263},
{"sigla": "BSB", "latitude": -15.8711, "longitude": -47.9186},
{"sigla": "UIO", "latitude": -0.1292, "longitude": -78.3579},
{"sigla": "JJD", "latitude": -2.898, "longitude": -40.3516},
{"sigla": "NVT", "latitude": -26.8835, "longitude": -48.656},
{"sigla": "RAO", "latitude": -21.1342, "longitude": -47.7743},
{"sigla": "FLN", "latitude": -27.6705, "longitude": -48.5528},
{"sigla": "BEL", "latitude": -1.3813, "longitude": -48.4781},
{"sigla": "PDP", "latitude": -34.8551, "longitude": -55.0943},
{"sigla": "LDB", "latitude": -23.3331, "longitude": -51.1342},
{"sigla": "ASU", "latitude": -25.2398, "longitude": -57.5191},
{"sigla": "ATL", "latitude": 33.6367, "longitude": -84.4281},
{"sigla": "SDU", "latitude": -22.9105, "longitude": -43.1631},
{"sigla": "MIA", "latitude": 25.7959, "longitude": -80.2871},
{"sigla": "PTY", "latitude": 9.0714, "longitude": -79.3835},
{"sigla": "MCO", "latitude": 28.4294, "longitude": -81.3089},
{"sigla": "IOS", "latitude": -14.815, "longitude": -39.0331},
{"sigla": "MVD", "latitude": -34.8384, "longitude": -56.0308},
{"sigla": "CAC", "latitude": -25.0014, "longitude": -53.5017},
{"sigla": "CGR", "latitude": -20.4687, "longitude": -54.6728},
{"sigla": "MEX", "latitude": 19.4361, "longitude": -99.0719},
{"sigla": "THE", "latitude": -5.0594, "longitude": -42.8235},
{"sigla": "IZA", "latitude": -21.5131, "longitude": -43.1733},
{"sigla": "RVD", "latitude": -17.7241, "longitude": -50.9141},
{"sigla": "OPS", "latitude": -11.8858, "longitude": -55.5861},
{"sigla": "MGF", "latitude": -23.478, "longitude": -52.0122},
{"sigla": "UDI", "latitude": -18.8836, "longitude": -48.2251},
{"sigla": "MOC", "latitude": -16.7069, "longitude": -43.8189},
{"sigla": "JJG", "latitude": -29.3033, "longitude": -49.0533},
{"sigla": "LIM", "latitude": -12.0219, "longitude": -77.1143},
{"sigla": "PNZ", "latitude": -9.3624, "longitude": -40.5691},
{"sigla": "XAP", "latitude": -27.1342, "longitude": -52.6566},
{"sigla": "PMW", "latitude": -10.2915, "longitude": -48.3566},
{"sigla": "SJP", "latitude": -20.8166, "longitude": -49.4065},
{"sigla": "JOI", "latitude": -26.2231, "longitude": -48.7973},
{"sigla": "BOG", "latitude": 4.7016, "longitude": -74.1469},
```

```
{"sigla": "ADD", "latitude": 8.9779, "longitude": 38.7993},
{"sigla": "EZE", "latitude": -34.8222, "longitude": -58.5358},
{"sigla": "IST", "latitude": 41.2753, "longitude": 28.7519},
{"sigla": "SDQ", "latitude": 18.4294, "longitude": -69.6689},
{"sigla": "FRA", "latitude": 50.0379, "longitude": 8.5622},
{"sigla": "VDC", "latitude": -14.8613, "longitude": -40.8631},
{"sigla": "LHR", "latitude": 51.4716, "longitude": -0.4643},
{"sigla": "LIS", "latitude": 38.7742, "longitude": -9.1342},
{"sigla": "DXB", "latitude": 25.2532, "longitude": 55.3657},
{"sigla": "DOH", "latitude": 25.2736, "longitude": 51.6086},
{"sigla": "JDO", "latitude": -7.2189, "longitude": -39.2701},
{"sigla": "JFK", "latitude": 40.6413, "longitude": -73.7781},
{"sigla": "LAX", "latitude": 33.9425, "longitude": -118.407},
{"sigla": "IMP", "latitude": -5.5313, "longitude": -47.4594},
{"sigla": "FCO", "latitude": 41.8003, "longitude": 12.2389},
{"sigla": "PFB", "latitude": -28.2433, "longitude": -52.3264},
{"sigla": "CDG", "latitude": 49.0097, "longitude": 2.5479},
{"sigla": "PPB", "latitude": -22.1751, "longitude": -51.4246},
{"sigla": "PET", "latitude": -31.7183, "longitude": -52.3274},
{"sigla": "MXP", "latitude": 45.6301, "longitude": 8.7231},
{"sigla": "BCN", "latitude": 41.2974, "longitude": 2.0833},
{"sigla": "LAD", "latitude": -8.8583, "longitude": 13.2312},
{"sigla": "ZRH", "latitude": 47.4582, "longitude": 8.5481},
{"sigla": "YYZ", "latitude": 43.6777, "longitude": -79.6248},
{"sigla": "EWR", "latitude": 40.6895, "longitude": -74.1745},
{"sigla": "IAD", "latitude": 38.9531, "longitude": -77.4565},
{"sigla": "AMS", "latitude": 52.3105, "longitude": 4.7683},
{"sigla": "IAH", "latitude": 29.9902, "longitude": -95.3368},
{"sigla": "DFW", "latitude": 32.8998, "longitude": -97.0403},
{"sigla": "ORD", "latitude": 41.9742, "longitude": -87.9073},
{"sigla": "BOS", "latitude": 42.3656, "longitude": -71.0096},
{"sigla": "AAX", "latitude": -19.5639, "longitude": -46.9643},
{"sigla": "JNB", "latitude": -26.1337, "longitude": 28.2426},
{"sigla": "VCP", "latitude": -23.0067, "longitude": -47.1344},
{"sigla": "YUL", "latitude": 45.4577, "longitude": -73.7499},
{"sigla": "UNA", "latitude": -15.3552, "longitude": -38.9997},
{"sigla": "DSS", "latitude": 14.6711, "longitude": -17.0711},
{"sigla": "GEL", "latitude": -28.2825, "longitude": -54.1691},
{"sigla": "OPO", "latitude": 41.2481, "longitude": -8.6814},
{"sigla": "PVH", "latitude": -8.709, "longitude": -63.9023},
{"sigla": "CPT", "latitude": -33.9648, "longitude": 18.6017},
{"sigla": "PUJ", "latitude": 18.5674, "longitude": -68.3634},
{"sigla": "CGH", "latitude": -23.6261, "longitude": -46.6564},
{"sigla": "BEY", "latitude": 33.8209, "longitude": 35.4883},
{"sigla": "COR", "latitude": -31.3236, "longitude": -64.2144},
{"sigla": "RRJ", "latitude": -27.4874, "longitude": -49.1746},
{"sigla": "UBA", "latitude": -19.7647, "longitude": -47.9644},
{"sigla": "CLV", "latitude": -17.7255, "longitude": -48.6073},
{"sigla": "IPN", "latitude": -19.4705, "longitude": -42.4876},
```

```
{"sigla": "ARU", "latitude": -21.1413, "longitude": -50.4247},
{"sigla": "BYO", "latitude": -21.2473, "longitude": -56.4525},
{"sigla": "GRU", "latitude": -23.4356, "longitude": -46.4731},
{"sigla": "BVB", "latitude": 2.8489, "longitude": -60.6901},
{"sigla": "AUX", "latitude": -7.2279, "longitude": -48.2405},
{"sigla": "RBR", "latitude": -9.8689, "longitude": -67.898},
{"sigla": "MCP", "latitude": 0.0506, "longitude": -51.0722},
{"sigla": "MAB", "latitude": -5.3686, "longitude": -49.1381},
{"sigla": "STM", "latitude": -2.4247, "longitude": -54.7856},
{"sigla": "BRA", "latitude": -12.0827, "longitude": -45.0084},
{"sigla": "FLL", "latitude": 26.0726, "longitude": -80.1527},
{"sigla": "ROS", "latitude": -32.9037, "longitude": -60.785},
{"sigla": "CPV", "latitude": -7.2699, "longitude": -35.8964},
{"sigla": "SJK", "latitude": -23.2292, "longitude": -45.8615},
{"sigla": "LUX", "latitude": 49.6233, "longitude": 6.2042},
{"sigla": "TFL", "latitude": -17.891, "longitude": -41.5136},
{"sigla": "GVR", "latitude": -18.8953, "longitude": -41.9822},
{"sigla": "CKS", "latitude": -6.1153, "longitude": -50.0017},
{"sigla": "LEC", "latitude": -12.4823, "longitude": -41.2773},
{"sigla": "LHN", "latitude": -13.2681, "longitude": -44.9488},
{"sigla": "VAG", "latitude": -21.5563, "longitude": -45.4756},
{"sigla": "JMA", "latitude": -21.6699, "longitude": -45.4444},
{"sigla": "CFB", "latitude": -21.5903, "longitude": -45.4731},
{"sigla": "GNM", "latitude": -14.2175, "longitude": -42.7808},
{"sigla": "POJ", "latitude": -18.6696, "longitude": -46.4905},
{"sigla": "CUR", "latitude": 12.1889, "longitude": -68.9598},
{"sigla": "ATM", "latitude": -3.2539, "longitude": -52.2548},
{"sigla": "CCP", "latitude": -36.7727, "longitude": -73.0631},
{"sigla": "PGZ", "latitude": -27.1591, "longitude": -49.2235},
{"sigla": "PMG", "latitude": -22.5487, "longitude": -55.7036},
{"sigla": "MII", "latitude": -22.1969, "longitude": -49.9264},
{"sigla": "TJL", "latitude": -22.7371, "longitude": -46.9864},
{"sigla": "JTC", "latitude": -22.1572, "longitude": -49.0681},
{"sigla": "CMG", "latitude": -19.0119, "longitude": -57.6724},
{"sigla": "ORY", "latitude": 48.7253, "longitude": 2.359},
{"sigla": "CAW", "latitude": -21.6983, "longitude": -41.3017},
{"sigla": "ROO", "latitude": -16.5863, "longitude": -54.7243},
{"sigla": "GPB", "latitude": -25.3875, "longitude": -51.5202},
{"sigla": "MDE", "latitude": 6.1645, "longitude": -75.4231},
{"sigla": "MEM", "latitude": 35.0425, "longitude": -89.9767},
{"sigla": "SJU", "latitude": 18.4394, "longitude": -66.0018},
{"sigla": "MVF", "latitude": -5.2025, "longitude": -37.3641},
{"sigla": "FEN", "latitude": -3.8549, "longitude": -32.4233},
{"sigla": "FEC", "latitude": -12.2035, "longitude": -38.9067},
{"sigla": "PAV", "latitude": -9.401, "longitude": -40.4897},
{"sigla": "CAU", "latitude": -8.2824, "longitude": -36.0132},
{"sigla": "SET", "latitude": -23.948, "longitude": -46.3335},
{"sigla": "LPA", "latitude": 27.9319, "longitude": -15.3866},
{"sigla": "RIA", "latitude": -29.378, "longitude": -53.7188},
```

```
  {"sigla": "URG", "latitude": -29.7822, "longitude": -57.0382},
  {"sigla": "BGX", "latitude": -31.3905, "longitude": -54.1122},
  {"sigla": "CSU", "latitude": -29.7114, "longitude": -53.6882},
  {"sigla": "SRA", "latitude": -27.9067, "longitude": -54.5203},
  {"sigla": "ALQ", "latitude": -29.7865, "longitude": -57.0368}
]

def apiOpenMeteo(latitude, longitude, sigla):
    hourly_data = {}
    url = f"https://archive-api.open-meteo.com/v1/era5?
latitude={latitude}&longitude={longitude}&start_date=2024-04-
07&end_date=2024-05-
07&hourly=temperature_2m,relative_humidity_2m,dew_point_2m,apparent_te
mperature,precipitation_probability,precipitation,rain,showers,snowfal
l,pressure_msl,cloud_cover,visibility,wind_speed_10m,wind_direction_10
m,wind_gusts_10m"
    hourly_categories =
['temperature_2m','relative_humidity_2m','dew_point_2m','apparent_temp
erature','precipitation_probability','precipitation','rain','showers',
'snowfall','pressure_msl','cloud_cover','visibility','wind_speed_10m',
'wind_direction_10m','wind_gusts_10m']

    response = requests.get(url)
    hourly = response.json()['hourly']
    timestamps = hourly['time']
    hourly_data['timestamp'] = timestamps
    hourly_data['sigla'] = [sigla] * len(timestamps)

    for hourly_category in hourly_categories:
        hourly_data[hourly_category] = hourly[hourly_category]


    hourly_dataframe = pd.DataFrame(data=hourly_data)

    return hourly_dataframe

dfs_clima = []

for loc in latitudes_longitudes:
    df_clima = apiOpenMeteo(loc['latitude'], loc['longitude'],
loc['sigla'])
    dfs_clima.append(df_clima)

df_clima = pd.concat(dfs_clima, ignore_index=True)
df_clima

{"type":"dataframe","variable_name":"df_clima"}

df_clima.to_csv('/content/clima.csv', index=False)
```

Junção

```
df_voo = pd.read_csv('/content/avionio.csv')
df_clima = pd.read_csv('/content/clima.csv')

df_voo['datetime'] = pd.to_datetime(df_voo['Date'] + ' 2024 ' +
df_voo['Time'].apply(lambda x: x[:2] + ':00'))
df_clima['datetime'] = pd.to_datetime(df_clima['timestamp'])

pd.set_option('display.max_columns', None)

dfComplete = pd.merge(df_voo, df_clima, left_on=['datetime',
'Origin'], right_on=['datetime', 'sigla'])
dfComplete = dfComplete.drop(columns=['sigla', 'timestamp'])
dfComplete = pd.merge(dfComplete, df_clima, left_on=['datetime', 'IATA
code'], right_on=['datetime', 'sigla'], suffixes=('', '_dst'))
dfComplete = dfComplete.drop(columns=['sigla', 'Time', 'Date',
'timestamp'])
dfComplete
```

```
{"type":"dataframe","variable_name":"dfComplete"}
```

```
dfComplete.to_csv('/content/dfComplete.csv', index=False)
```

Estatísticas Descritivas

- Visualizações

```
dfComplete = pd.read_csv('/content/dfComplete.csv')
dfComplete
```

```
{"type":"dataframe","variable_name":"dfComplete"}
```

```
dfComplete.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129905 entries, 0 to 129904
Data columns (total 37 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   IATA code                     129905 non-null  object
 1   Destination                   129905 non-null  object
 2   Flight                        129905 non-null  object
 3   Airline                       129755 non-null  object
 4   Status                        129905 non-null  object
 5   Origin                        129905 non-null  object
 6   datetime                      129905 non-null  object
 7   temperature_2m                129905 non-null  float64
 8   relative_humidity_2m          129905 non-null  int64
 9   dew_point_2m                  129905 non-null  float64
 10  apparent_temperature          129905 non-null  float64
 11  precipitation_probability     0 non-null       float64
```

```
 12  precipitation                  129905 non-null  float64
 13  rain                           129905 non-null  float64
 14  showers                        0 non-null       float64
 15  snowfall                       129905 non-null  float64
 16  pressure_msl                   129905 non-null  float64
 17  cloud_cover                    129905 non-null  int64
 18  visibility                     0 non-null       float64
 19  wind_speed_10m                 129905 non-null  float64
 20  wind_direction_10m             129905 non-null  int64
 21  wind_gusts_10m                 129905 non-null  float64
 22  temperature_2m_dst             129905 non-null  float64
 23  relative_humidity_2m_dst       129905 non-null  int64
 24  dew_point_2m_dst               129905 non-null  float64
 25  apparent_temperature_dst       129905 non-null  float64
 26  precipitation_probability_dst  0 non-null       float64
 27  precipitation_dst              129905 non-null  float64
 28  rain_dst                       129905 non-null  float64
 29  showers_dst                    0 non-null       float64
 30  snowfall_dst                   129905 non-null  float64
 31  pressure_msl_dst               129905 non-null  float64
 32  cloud_cover_dst                129905 non-null  int64
 33  visibility_dst                 0 non-null       float64
 34  wind_speed_10m_dst             129905 non-null  float64
 35  wind_direction_10m_dst         129905 non-null  int64
 36  wind_gusts_10m_dst             129905 non-null  float64
dtypes: float64(24), int64(6), object(7)
memory usage: 36.7+ MB
```

Percebe-se que, exceto pelas colunas vazias, não há muitos dados ausentes e há poucas variáveis categóricas. Isso é relevante porque a ausência de dados incompletos facilita a análise estatística e a modelagem preditiva, reduzindo a necessidade de técnicas de imputação ou de exclusão de amostras. Além disso, a predominância de variáveis numéricas simplifica a aplicação de algoritmos de aprendizado de máquina, que geralmente requerem menos pré-processamento comparado às variáveis categóricas.

```
dfComplete.describe()
```

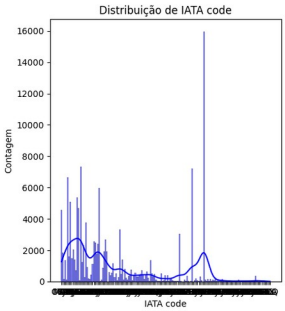
{"type":"dataframe"}

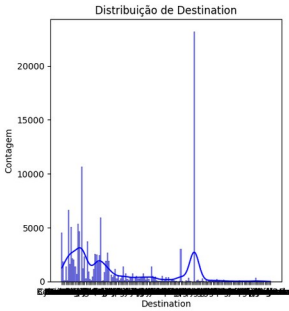
Na análise inicial, é possível identificar colunas vazias que podem ser eliminadas posteriormente. Observamos que, na maioria dos casos, os valores da mediana são próximos aos valores da média, indicando uma baixa presença de outliers.

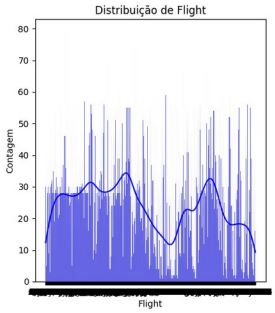
Distribuição das Variáveis

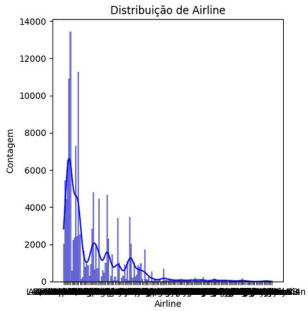
```
fig, axs = plt.subplots(9, 4, figsize=(20, 5 * 9))
for i, coluna in enumerate(dfComplete.loc[:, dfComplete.columns !=
'datetime'].columns):
  ax = axs[i // 4, i % 4]
```

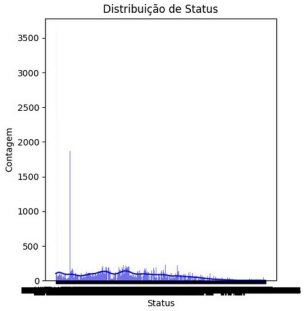
```
  sns.histplot(data=dfComplete.loc[:, dfComplete.columns !=
'datetime'], x=coluna, ax=ax, color='blue', alpha=0.5, kde=True)

  ax.set_title(f'Distribuição de {coluna}')
  ax.set_xlabel(f'{coluna}')
  ax.set_ylabel('Contagem')
plt.tight_layout()
plt.show()
```

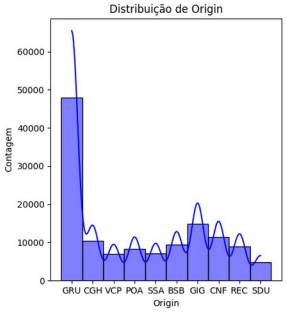

Distribuição de IATA code
Contagem
IATA code

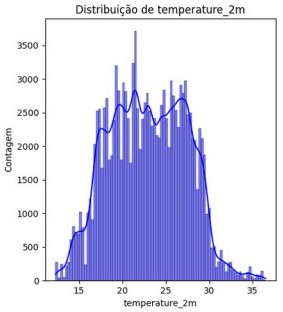

Distribuição de Destination
Contagem
Destination

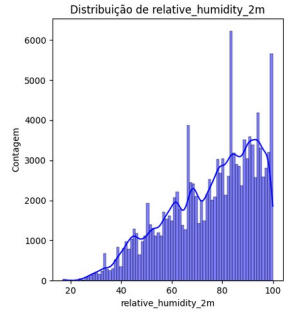

Distribuição de Flight
Contagem
Flight

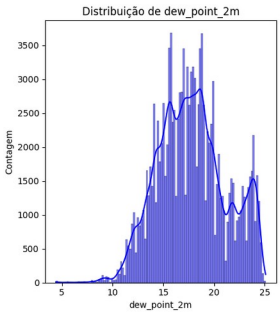

Distribuição de Airline
Contagem
Airline

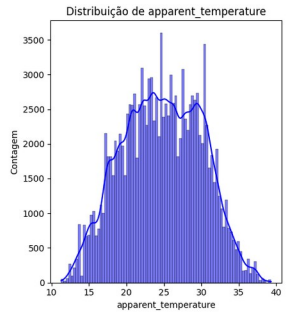

Distribuição de Status
Contagem
Status

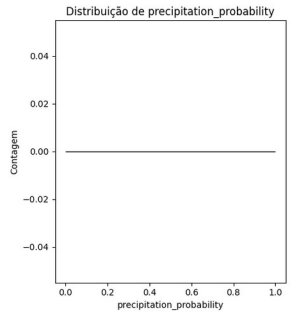

Distribuição de Origin
Contagem
GRU CGH VCP POA SSA BSB GIG CNF REC SDU
Origin

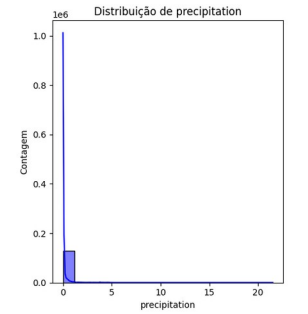

Distribuição de temperature_2m
Contagem
temperature_2m

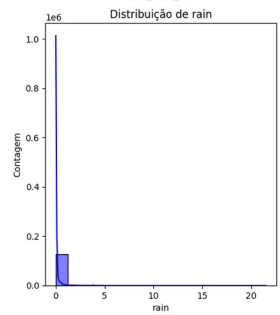

Distribuição de relative_humidity_2m
Contagem
relative_humidity_2m

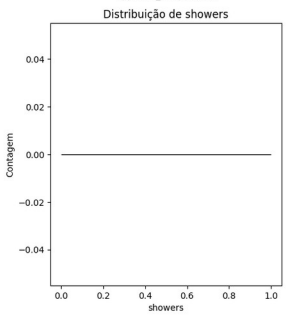

Distribuição de dew_point_2m
Contagem
dew_point_2m

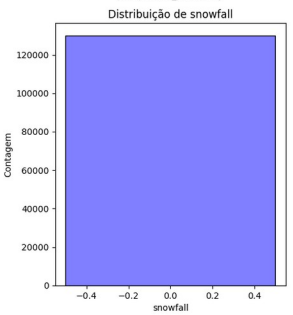

Distribuição de apparent_temperature
Contagem
apparent_temperature

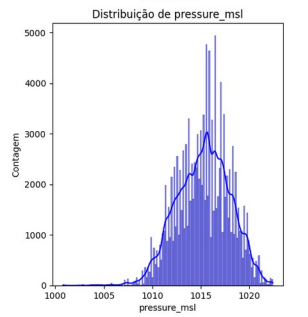

Distribuição de precipitation_probability
Contagem
precipitation_probability

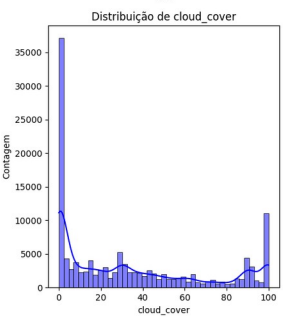

Distribuição de precipitation
1e6
Contagem
precipitation

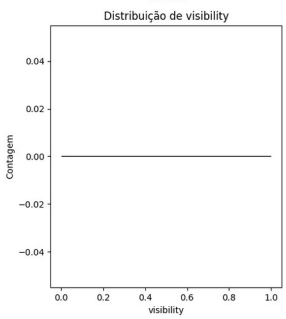

Distribuição de rain
1e6
Contagem
rain

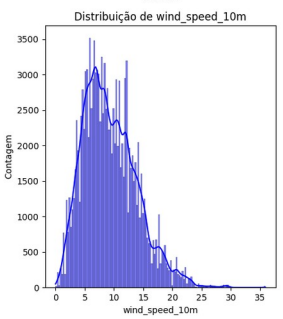

Distribuição de showers
Contagem
showers

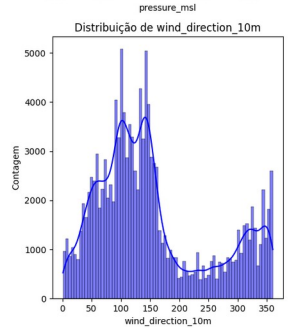

Distribuição de snowfall
Contagem
snowfall

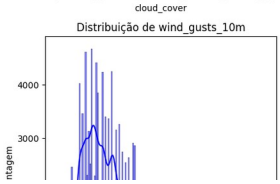

Distribuição de pressure_msl
Contagem
pressure_msl

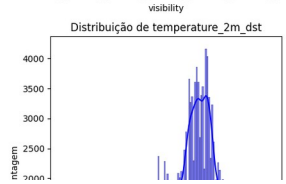

Distribuição de cloud_cover
Contagem
cloud_cover

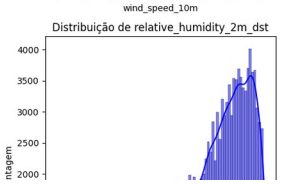

Distribuição de visibility
Contagem
visibility

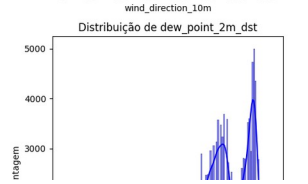

Distribuição de wind_speed_10m
Contagem
wind_speed_10m


Distribuição de wind_direction_10m
Contagem
wind_direction_10m


Distribuição de wind_gusts_10m


Distribuição de temperature_2m_dst

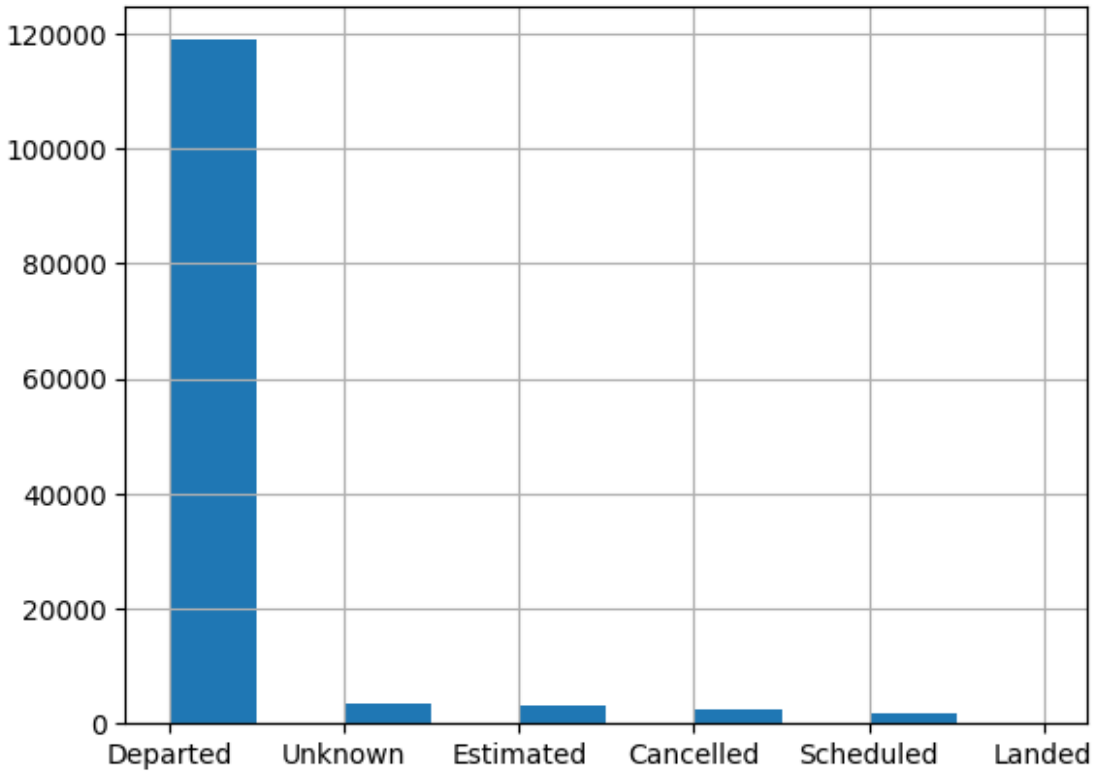

Distribuição de relative_humidity_2m_dst


Distribuição de dew_point_2m_dst

Através da análise da distribuição das variáveis, foi possível identificar algumas variáveis com valores únicos. Essas variáveis não são relevantes para a análise, pois não agregam informação adicional. Além disso, observa-se que a maioria das variáveis apresenta uma distribuição próxima à normal. Essa característica é vantajosa, pois facilita a aplicação de métodos estatísticos, que frequentemente assumem a normalidade dos dados para gerar resultados mais precisos e interpretáveis. A combinação de variáveis com distribuições normais e a ausência de dados incompletos ou categóricos excessivos contribui para a robustez e a eficácia das análises subsequentes.

```
dfComplete['Status'] = dfComplete['Status'].apply(lambda x: re.sub(r'\
s*\d{2}:\d{2}', '', str(x)))
dfComplete['Status'].hist()
```

```
<Axes: >
```



Covariância

A covariância mede a tendência conjunta de duas variáveis em se desviarem de suas respectivas médias. Se a covariância é positiva, indica que as variáveis tendem a aumentar ou diminuir juntas. Se é negativa, uma variável tende a aumentar enquanto a outra diminui. A magnitude da covariância indica a força da relação linear entre as variáveis.

```
dfComplete[dfComplete.Status ==
'Departed'].select_dtypes(include=['float64', 'int64']).cov()
```

```
{"type":"dataframe"}
```

```
dfComplete[dfComplete.Status ==
'Cancelled'].select_dtypes(include=['float64', 'int64']).cov()
```

```
{"type":"dataframe"}
```

Correlação de Pearson

A correlação de Pearson mede a força e a direção da associação linear entre duas variáveis. Baseia-se nas médias e desvios padrão dos dados. Um coeficiente próximo de 1 ou -1 indica uma forte associação linear. Um coeficiente próximo de 0 indica pouca ou nenhuma associação linear. Um coeficiente positivo indica que as variáveis tendem a aumentar juntas, enquanto um coeficiente negativo indica que uma variável tende a aumentar enquanto a outra diminui. A correlação de Pearson é sensível a outliers, que podem distorcer o coeficiente de correlação.

```
dfComplete[dfComplete.Status ==
'Departed'].select_dtypes(include=['float64',
'int64']).corr(method='pearson')
```

```
{"type":"dataframe"}
```

```
dfComplete[dfComplete.Status ==
'Cancelled'].select_dtypes(include=['float64',
'int64']).corr(method='pearson')
```

```
{"type":"dataframe"}
```

Correlação de Spearman

A correlação de Spearman mede a força e a direção da associação monotônica (não necessariamente linear) entre duas variáveis. Baseia-se nas posições (ranks) dos dados. Um coeficiente próximo de 1 ou -1 indica uma forte associação monotônica. Um coeficiente próximo de 0 indica pouca ou nenhuma associação monotônica. Como se baseia em ranks, a correlação de Spearman é menos sensível a outliers do que a correlação de Pearson. É útil para identificar relações monotônicas não lineares que a correlação de Pearson poderia não capturar.

```
dfComplete[dfComplete.Status ==
'Departed'].select_dtypes(include=['float64',
'int64']).corr(method='spearman')
```

```
{"type":"dataframe"}
```

```
dfComplete[dfComplete.Status ==
'Cancelled'].select_dtypes(include=['float64',
'int64']).corr(method='spearman')
```

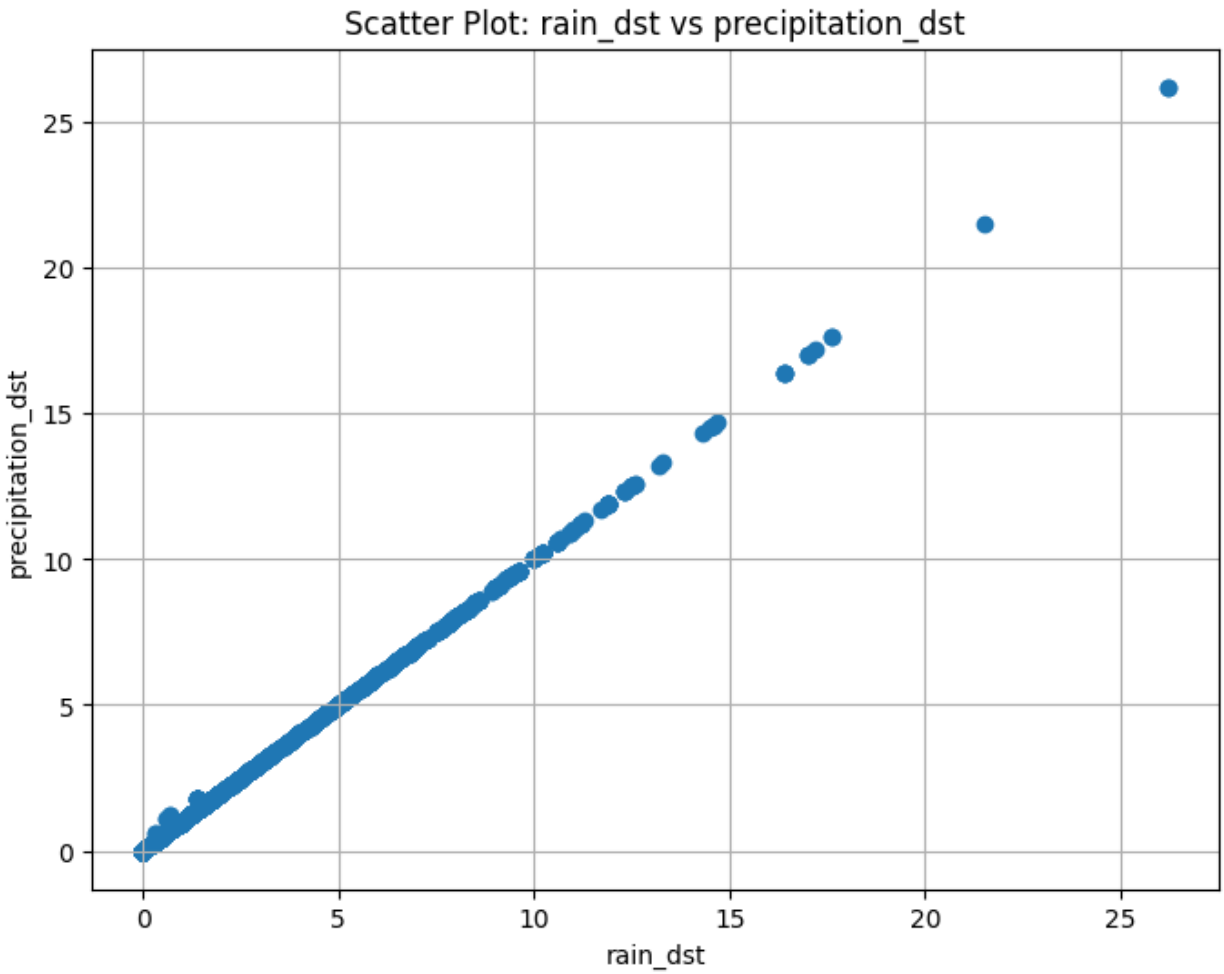
```
{"type":"dataframe"}
```

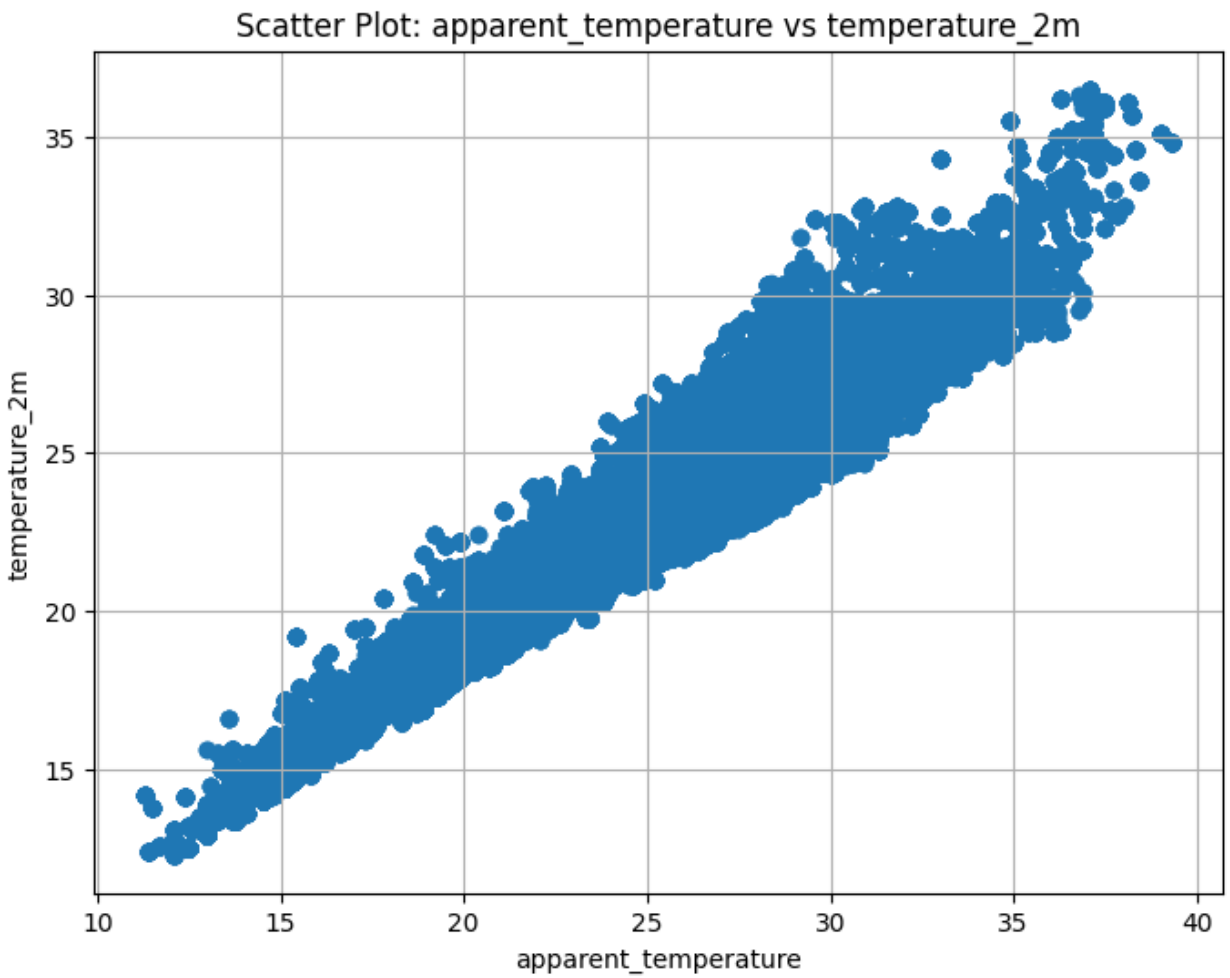
Comportamento de Pares de Variáveis Altamente Relacionadas

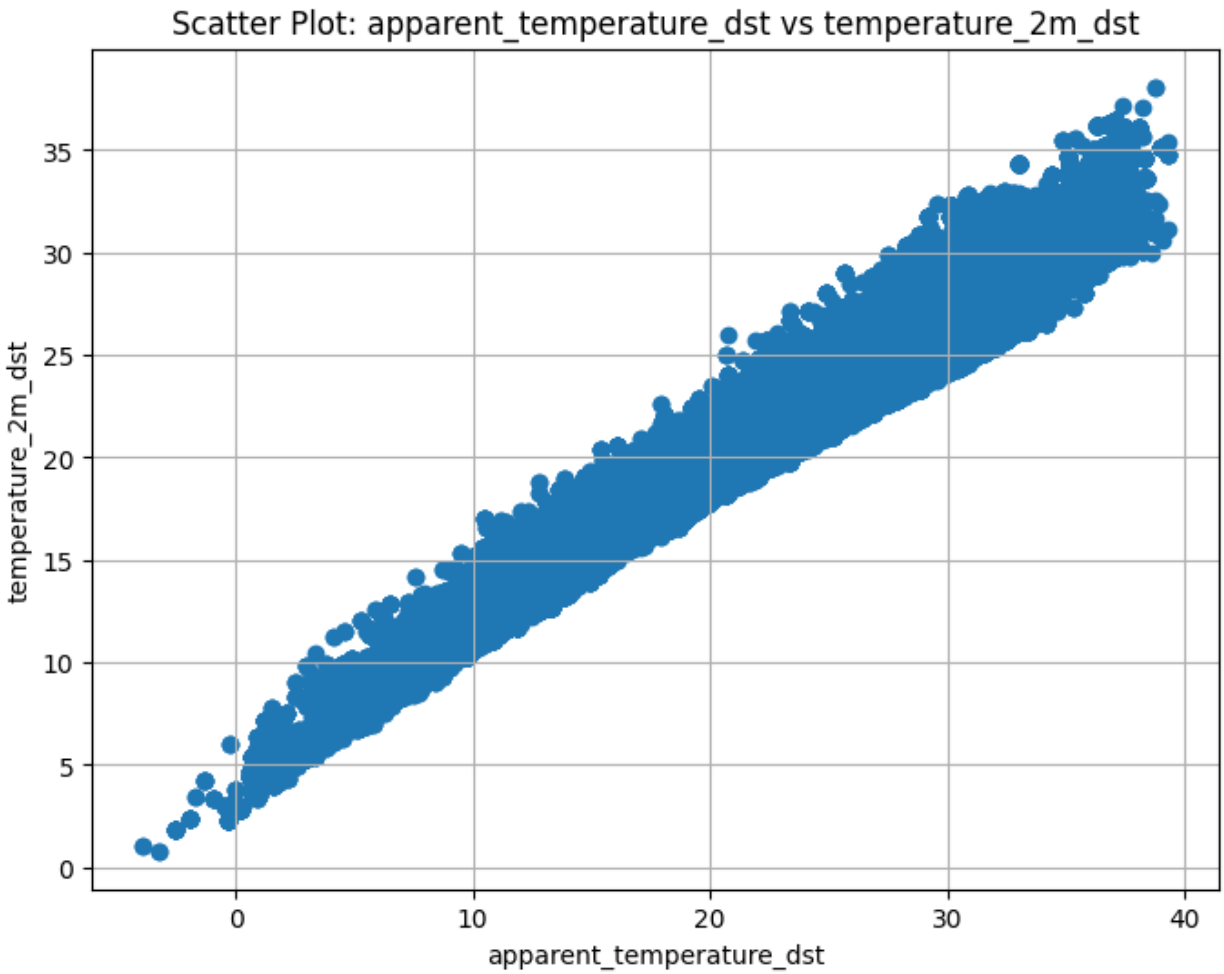
```
corr_spearman = dfComplete.select_dtypes(include=['float64',
'int64']).corr(method='spearman')
high_corr_pairs =
corr_spearman.abs().unstack().sort_values(ascending=False)
high_corr_pairs = high_corr_pairs[(high_corr_pairs > 0.85) &
(high_corr_pairs < 1)].reset_index()
high_corr_pairs = high_corr_pairs[high_corr_pairs.index % 2 ==
0].reset_index().drop(columns=['index'])
high_corr_pairs
```

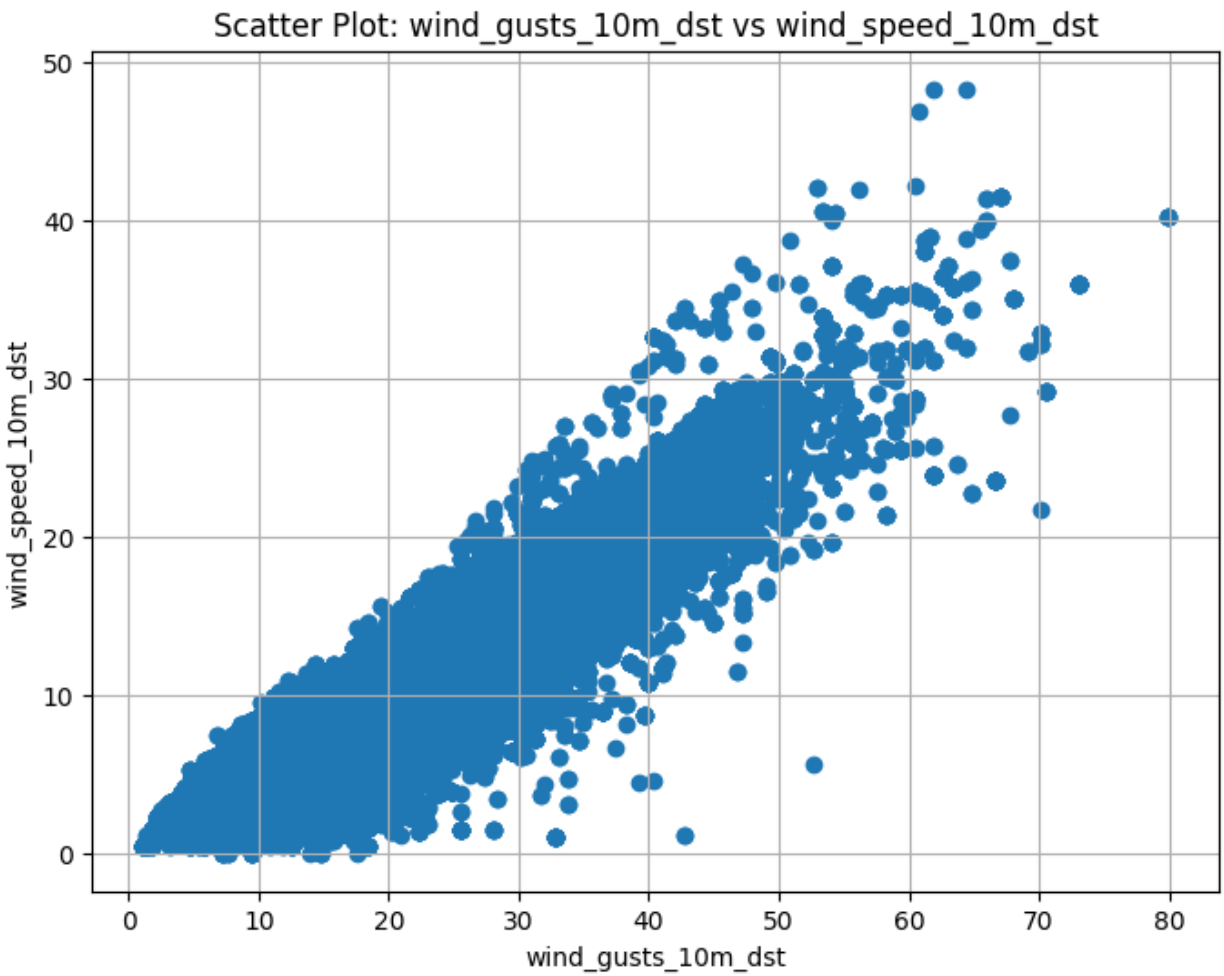

```
{"summary":"{\n  \"name\": \"high_corr_pairs\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"level_0\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"apparent_temperature\",\n          \"wind_gusts_10m\",\n
\"apparent_temperature_dst\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"level_1\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"temperature_2m\",\n          \"wind_speed_10m\",\n
\"temperature_2m_dst\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": 0,\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.04653991397335136,\n        \"min\": 0.8892980695990584,\n
\"max\": 0.999999617962444,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.9611247874582416,\n
0.8892980695990584,\n          0.9379980069452561\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"high_corr_pairs"}
```

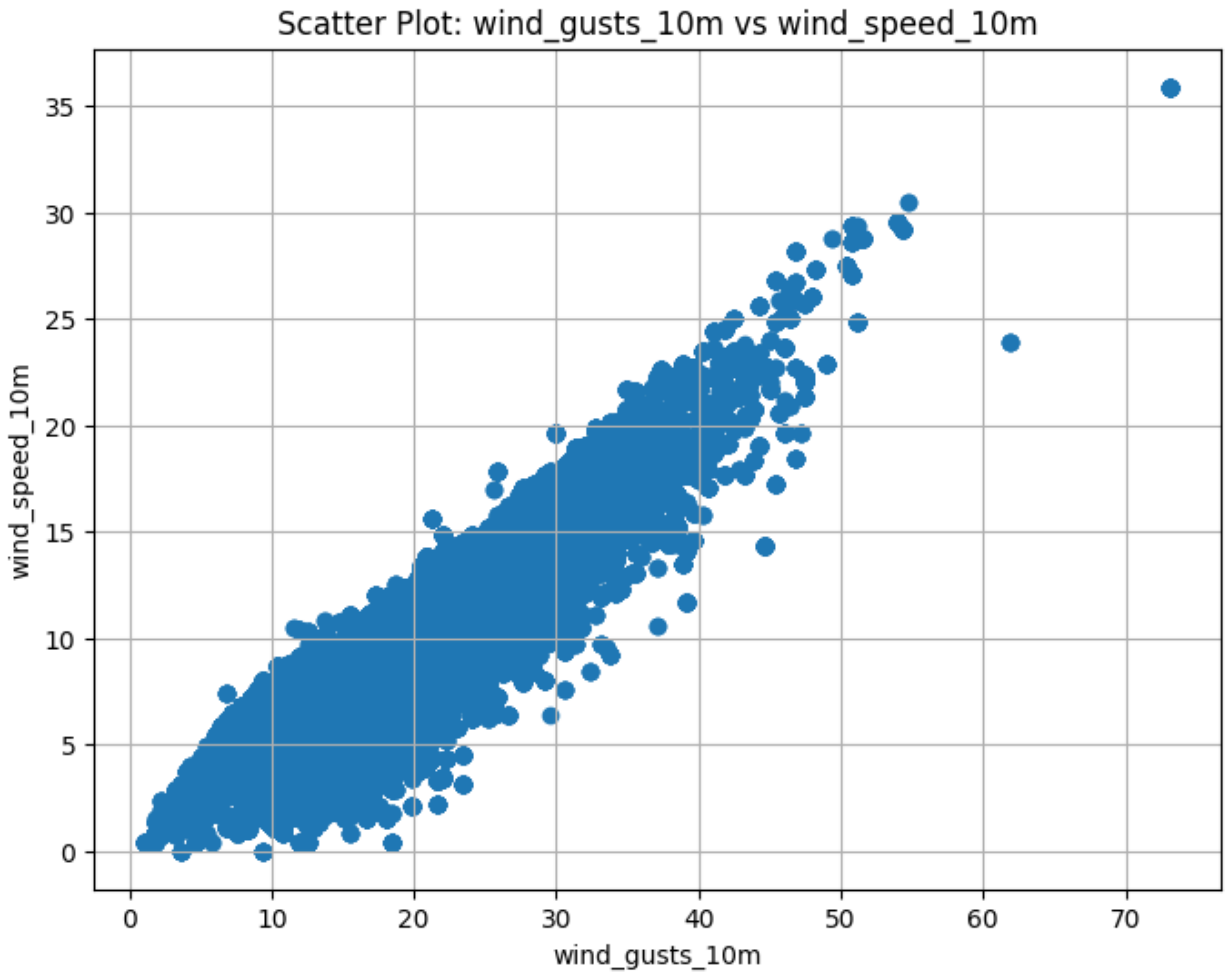

```
def scatter_plot(pair):
  var1, var2, _ = pair
  plt.figure(figsize=(8, 6))
  plt.scatter(dfComplete[var1], dfComplete[var2])
  plt.xlabel(var1)
  plt.ylabel(var2)
  plt.title(f"Scatter Plot: {var1} vs {var2}")
  plt.grid(True)
  plt.show()

high_corr_pairs.apply(scatter_plot, axis=1)
```


Scatter Plot: rain_dst vs precipitation_dst
precipitation_dst
rain_dst
0
5
10
15
20
25


Scatter Plot: apparent_temperature vs temperature_2m
temperature_2m
apparent_temperature
35
30
25
20
15
10
15
20
25
30
35
40


Scatter Plot: apparent_temperature_dst vs temperature_2m_dst
temperature_2m_dst
apparent_temperature_dst
0
5
10
15
20
25
30
35
0
10
20
30
40


Scatter Plot: wind_gusts_10m_dst vs wind_speed_10m_dst
wind_speed_10m_dst
wind_gusts_10m_dst
0
10
20
30
40
50
60
70
80




```
0    None
1    None
2    None
3    None
4    None
dtype: object
```

Pré-Processamento dos Dados

- Definição de tipos
- Tratamento de dados ausentes
- Normalização e discretização
- Limpeza de dados (univariado, bivariado e multivariado)

```
dfComplete = pd.read_csv('/content/dfComplete.csv')
dfComplete
```

```
{"type":"dataframe","variable_name":"dfComplete"}
```

Informação Redundante

Linhas do conjunto de dados que possui apenas valores repetidos não adicionam nenhuma variabilidade ou informação útil à análise. A remoção dessas linhas diminui a complexidade do modelo, facilita a visualização e análise, além de reduzir o tempo de processamento.

```
print(dfComplete.shape)
dfComplete = dfComplete.drop_duplicates()
dfComplete.shape
```

```
(129905, 37)
```

```
(105038, 37)
```

Identificando colunas vazias (todos os valores são nulos) ou com apenas um valor (não acrescentam informação à análise).

```
colunas_vazias = dfComplete.columns[dfComplete.isna().all()].tolist()
colunas_valores_iguais = dfComplete.columns[dfComplete.apply(lambda x:
x.nunique()) == 1].tolist()
colunas_vazias += colunas_valores_iguais
print(colunas_vazias)

dfComplete = dfComplete.drop(columns=colunas_vazias)
```

```
['precipitation_probability', 'showers', 'visibility',
'precipitation_probability_dst', 'showers_dst', 'visibility_dst',
'snowfall']
```

Definição de Tipos

```
dfComplete.dtypes
```

```
IATA code                  object
Destination                object
Flight                     object
Airline                    object
Status                     object
Origin                     object
datetime                   object
temperature_2m            float64
relative_humidity_2m        int64
dew_point_2m              float64
apparent_temperature      float64
precipitation             float64
rain                      float64
pressure_msl              float64
cloud_cover                 int64
wind_speed_10m            float64
wind_direction_10m          int64
```

```
wind_gusts_10m               float64
temperature_2m_dst           float64
relative_humidity_2m_dst       int64
dew_point_2m_dst             float64
apparent_temperature_dst     float64
precipitation_dst            float64
rain_dst                     float64
snowfall_dst                 float64
pressure_msl_dst             float64
cloud_cover_dst                int64
wind_speed_10m_dst           float64
wind_direction_10m_dst         int64
wind_gusts_10m_dst           float64
dtype: object
```

Todas as informações contidas na tabela de clima são de natureza numérica, representadas como inteiros ou float. Portanto, apenas as colunas presentes no conjunto de dados de voos requerem avaliação individual por serem dados categóricos.

```
dfComplete['IATA code'] = dfComplete['IATA code'].astype('category')
dfComplete['Destination'] =
dfComplete['Destination'].astype('category')
dfComplete['Flight'] = dfComplete['Flight'].astype('category')
dfComplete['Airline'] = dfComplete['Airline'].astype('category')
dfComplete['Status'] = dfComplete['Status'].astype('category')
dfComplete['Origin'] = dfComplete['Origin'].astype('category')

print('IATA code:', dfComplete['IATA code'].cat.categories)
print('Destination:', dfComplete['Destination'].cat.categories)
print('Flight:', dfComplete['Flight'].cat.categories)
print('Airline:', dfComplete['Airline'].cat.categories)
print('Status:', dfComplete['Status'].cat.categories)
print('Origin:', dfComplete['Origin'].cat.categories)
```

```
IATA code: Index(['AAX', 'ADD', 'AEP', 'AJU', 'ALQ', 'AMS', 'ARU',
'ASU', 'ATL', 'ATM',
       ...
       'URG', 'VAG', 'VCP', 'VDC', 'VIX', 'VVI', 'XAP', 'YUL', 'YYZ',
'ZRH'],
      dtype='object', length=162)
Destination: Index(['Addis Ababa', 'Alegrete', 'Amsterdam', 'Aracaju',
'Aracatuba',
       'Araguaina', 'Araxa', 'Asuncion', 'Atlanta', 'Bage',
       ...
       'Toronto', 'Tres Lagoas', 'Uberaba', 'Uberlandia', 'Una',
'Uruguaiana',
       'Varginha', 'Vitoria', 'Vitoria Da Conquista', 'Zurich'],
      dtype='object', length=157)
Flight: Index(['2Z2201', '2Z2203', '2Z2205', '2Z2207', '2Z2209',
```

```
'2Z2210', '2Z2213',
       '2Z2215', '2Z2216', '2Z2217',
       ...
       'VS7819', 'VS7822', 'VS7823', 'VS7831', 'W8926', 'WB1225',
'WB1342',
       'WD5800', 'WD5801', 'ZP1837'],
      dtype='object', length=5322)
Airline: Index(['ANA', 'ASKY', 'Aero FlightOps UK', 'Aerolineas
Argentinas',
       'Aerolineas Argentinas  1', 'Aerolineas Argentinas  2',
       'Aerolineas Argentinas  3', 'Aerolineas Argentinas  4',
       'Aerolineas Argentinas  5', 'Aeromexico',
       ...
       'Turkish Airlines', 'Turkish Airlines  2', 'Turkish Airlines
3',
       'Turkish Airlines  4', 'United Airlines', 'United Airlines  1',
       'Virgin Atlantic', 'VoePass', 'VoePass  1', 'VoePass  2'],
      dtype='object', length=155)
Status: Index(['Cancelled', 'Departed 00:00', 'Departed 00:01',
'Departed 00:02',
       'Departed 00:03', 'Departed 00:04', 'Departed 00:05', 'Departed
00:06',
       'Departed 00:07', 'Departed 00:08',
       ...
       'Estimated 23:43', 'Estimated 23:45', 'Estimated 23:47',
       'Estimated 23:48', 'Estimated 23:49', 'Estimated 23:50',
       'Estimated 23:58', 'Landed', 'Scheduled', 'Unknown'],
      dtype='object', length=1895)
Origin: Index(['BSB', 'CGH', 'CNF', 'GIG', 'GRU', 'POA', 'REC', 'SDU',
'SSA', 'VCP'], dtype='object')
```

Limpeza da Coluna Status

```
dfComplete['Status'] = dfComplete['Status'].apply(lambda x: re.sub(r'\
s*\d{2}:\d{2}', '', str(x)))

dfComplete['Status'] = dfComplete['Status'].astype('category')
print('Status:', dfComplete['Status'].cat.categories)

Status: Index(['Cancelled', 'Departed', 'Estimated', 'Landed',
'Scheduled', 'Unknown'], dtype='object')
```

Com base na análise do conjunto de dados, identificamos que há 105.038 registros. A categoria com a maior variedade contém 5.322 tipos distintos, indicando sua potencial relevância para os resultados da análise. Uma variedade muito alta, aproximando-se do número de registros indicaria uma coluna com comportamento identificadora, não sendo útil a análise.

O método cat.codes é mais apropriado quando existe uma ordem intrínseca nos dados categóricos. Entretanto, para simplificação e para garantir a uniformidade dos tipos de dados como numéricos, será empregado esse método para as variáveis categóricas.

```
dfComplete['IATA code'] = dfComplete['IATA code'].cat.codes
dfComplete['Destination'] = dfComplete['Destination'].cat.codes
dfComplete['Flight'] = dfComplete['Flight'].cat.codes
dfComplete['Airline'] = dfComplete['Airline'].cat.codes
dfComplete['Origin'] = dfComplete['Origin'].cat.codes

dfComplete.dtypes
```

```
IATA code                    int16
Destination                  int16
Flight                       int16
Airline                      int16
Status                    category
Origin                        int8
datetime                    object
temperature_2m             float64
relative_humidity_2m         int64
dew_point_2m               float64
apparent_temperature       float64
precipitation              float64
rain                       float64
pressure_msl               float64
cloud_cover                  int64
wind_speed_10m             float64
wind_direction_10m           int64
wind_gusts_10m             float64
temperature_2m_dst         float64
relative_humidity_2m_dst     int64
dew_point_2m_dst           float64
apparent_temperature_dst   float64
precipitation_dst          float64
rain_dst                   float64
snowfall_dst               float64
pressure_msl_dst           float64
cloud_cover_dst              int64
wind_speed_10m_dst         float64
wind_direction_10m_dst       int64
wind_gusts_10m_dst         float64
dtype: object
```

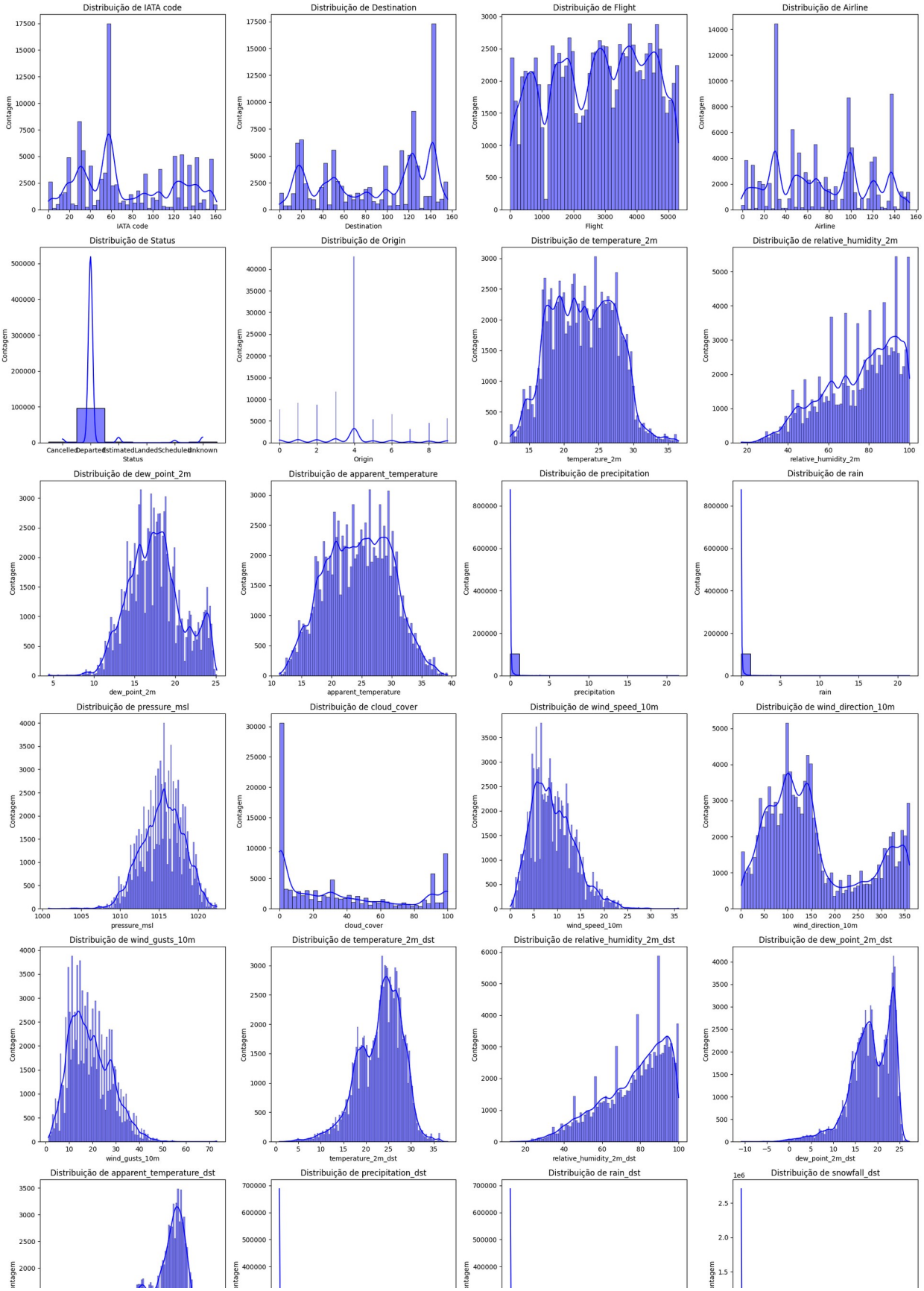
Visualização inicial

```
dfComplete.describe()
```

```
{"type":"dataframe"}
```

```
fig, axs = plt.subplots(8, 4, figsize=(20, 5 * 8))
for i, coluna in enumerate(dfComplete.loc[:, dfComplete.columns !=
'datetime'].columns):
  ax = axs[i // 4, i % 4]
```

```
  sns.histplot(data=dfComplete, x=coluna, ax=ax, color='blue',
alpha=0.5, kde=True)

  ax.set_title(f'Distribuição de {coluna}')
  ax.set_xlabel(f'{coluna}')
  ax.set_ylabel('Contagem')
plt.tight_layout()
plt.show()
```


Distribuição de IATA code
Distribuição de Destination
Distribuição de Flight
Distribuição de Airline
Distribuição de Status
Distribuição de Origin
Distribuição de temperature_2m
Distribuição de relative_humidity_2m
Distribuição de dew_point_2m
Distribuição de apparent_temperature
Distribuição de precipitation
Distribuição de rain
Distribuição de pressure_msl
Distribuição de cloud_cover
Distribuição de wind_speed_10m
Distribuição de wind_direction_10m
Distribuição de wind_gusts_10m
Distribuição de temperature_2m_dst
Distribuição de relative_humidity_2m_dst
Distribuição de dew_point_2m_dst
Distribuição de apparent_temperature_dst
Distribuição de precipitation_dst
Distribuição de rain_dst
Distribuição de snowfall_dst
Contagem
IATA code
Destination
Flight
Airline
Status
Origin
temperature_2m
relative_humidity_2m
dew_point_2m
apparent_temperature
precipitation
rain
pressure_msl
cloud_cover
wind_speed_10m
wind_direction_10m
wind_gusts_10m
temperature_2m_dst
relative_humidity_2m_dst
dew_point_2m_dst
1e6

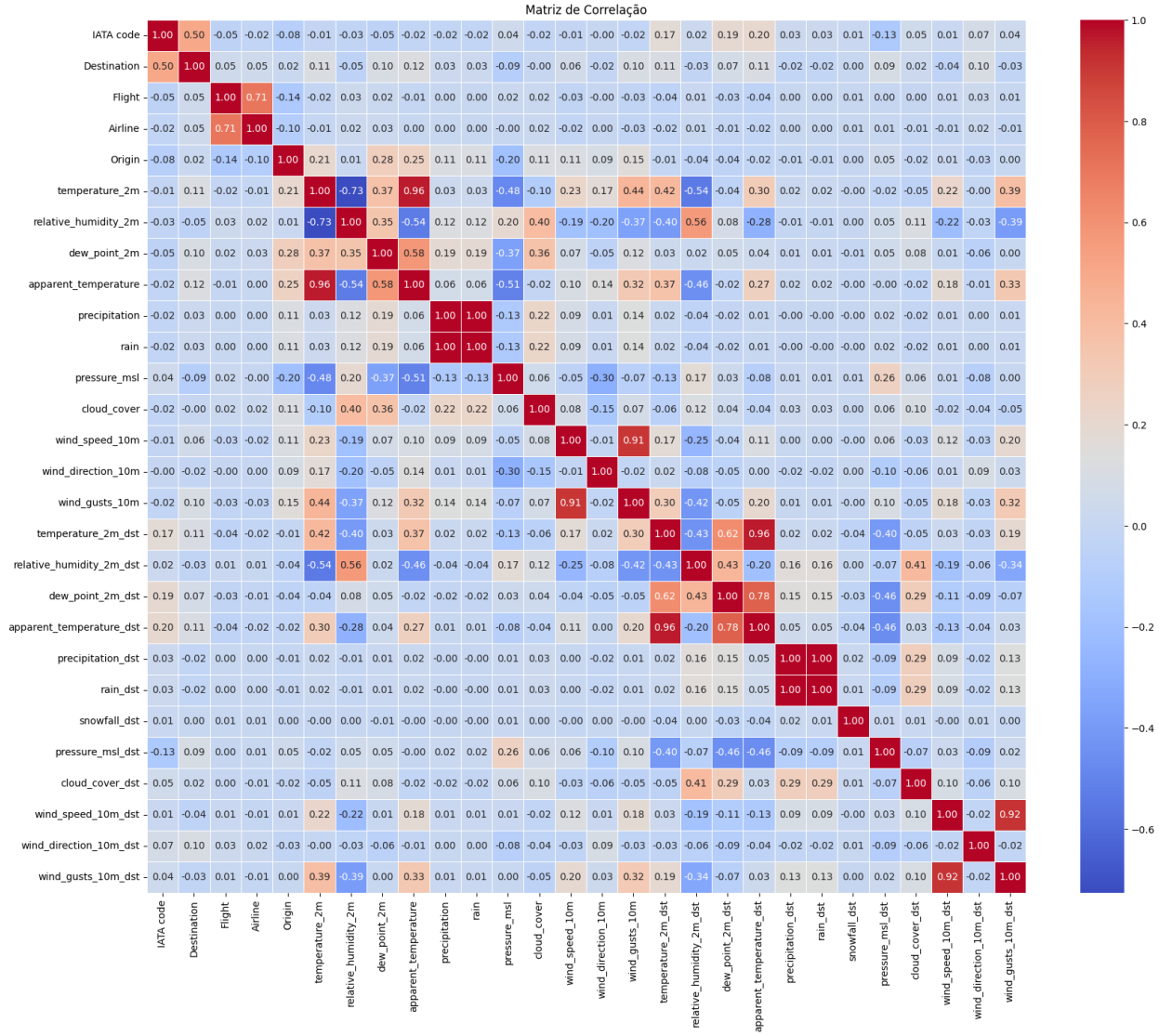
Na primeira análise, observa-se que a maioria das variáveis segue uma distribuição normal, o que permite a aplicação de diversos métodos estatísticos e analíticos. No entanto, há um evidente desbalanceamento nos dados, particularmente na variável 'Status de voo', onde uma classe é significativamente mais predominante em relação às demais. Este desbalanceamento pode influenciar os resultados das análises e previsões, indicando a necessidade de abordar esse problema em etapas subsequentes para garantir a precisão e a robustez das conclusões.

Correlação entre Variáveis

A análise de correlação entre variáveis é uma etapa essencial no entendimento da estrutura de uma base de dados. Além de fornecer insights sobre o comportamento dos dados, essa análise permite identificar variáveis fortemente relacionadas, o que pode indicar redundância no conjunto de dados. A presença de variáveis altamente correlacionadas aumenta a dimensionalidade do modelo sem necessariamente agregar novas informações relevantes, o que pode prejudicar a eficiência computacional e a interpretabilidade do modelo.

Dado o número substancial de colunas em nosso conjunto de dados, é prudente conduzir a análise de correlação por tipo de variável. Isso permite uma visualização mais organizada e compreensível, facilitando a identificação de padrões e relações entre as variáveis. Ao segmentar a análise por tipo de variável, como numérico ou categórico, podemos explorar de forma mais eficaz as relações entre os diferentes tipos de dados presentes na base de dados.

```
correlation_matrix = dfComplete.loc[:,
~dfComplete.columns.isin(['datetime', 'Status'])].corr()

plt.figure(figsize=(20, 16))
sns.heatmap(correlation_matrix , annot=True, fmt=".2f",
cmap='coolwarm', linewidths=0.5)
plt.title('Matriz de Correlação')
plt.show()
```




```
high_correlation_pairs = []
cols = correlation_matrix.columns

for i in range(len(cols)):
  for j in range(i + 1, len(cols)):
    if abs(correlation_matrix.iloc[i, j]) > 0.85:
      high_correlation_pairs.append((cols[i], cols[j],
correlation_matrix.iloc[i, j]))

print("Pares de colunas com correlação maior que 0.85:")
for col1, col2, corr in high_correlation_pairs:
  print(f"{col1} e {col2}: {corr:.2f}")
```

```
Pares de colunas com correlação maior que 0.85:
temperature_2m e apparent_temperature: 0.96
precipitation e rain: 1.00
```

```
wind_speed_10m e wind_gusts_10m: 0.91
temperature_2m_dst e apparent_temperature_dst: 0.96
precipitation_dst e rain_dst: 1.00
wind_speed_10m_dst e wind_gusts_10m_dst: 0.92
```

Essas variáveis que ultrapassaram o limiar de correlação, definido como 85%, são consideradas redundantes para o conjunto de dados, pois apresentam uma alta correlação entre si, o que pode introduzir multicolinearidade e aumentar a complexidade sem fornecer informações adicionais significativas. A identificação e remoção dessas variáveis podem ajudar a simplificar o conjunto de dados e melhorar a eficiência e interpretabilidade da análise.

Esta abordagem visa eliminar a redundância nas características, preservando a qualidade e a relevância das informações contidas no conjunto de dados. Ao reduzir o número de características altamente correlacionadas, podemos mitigar o risco de overfitting e melhorar a capacidade do modelo de generalizar para novos dados.

```
columns_to_remove = set()

for col1, col2, _ in high_correlation_pairs:
  columns_to_remove.add(col2)

dfComplete = dfComplete.drop(columns=columns_to_remove)
```

Limpeza de dados (Detecção de Outliers)

- Univariado
  - Z-Score robusto
  - Tukey
- Bivariado
  - Transformar em Univariado (Razão de uma variável pela outra)
- Multivariado
  - Elliptic Envelope
  - Distância Mahalanobis
  - Isolation Forests

Os outliers podem distorcer a distribuição dos dados e influenciar significativamente as medidas de tendência central, comprometendo a análise estatística. Portanto, é essencial identificar e remover os outliers a fim de realizar uma análise mais precisa e robusta dos dados. Para isso, são empregadas técnicas estatísticas específicas.

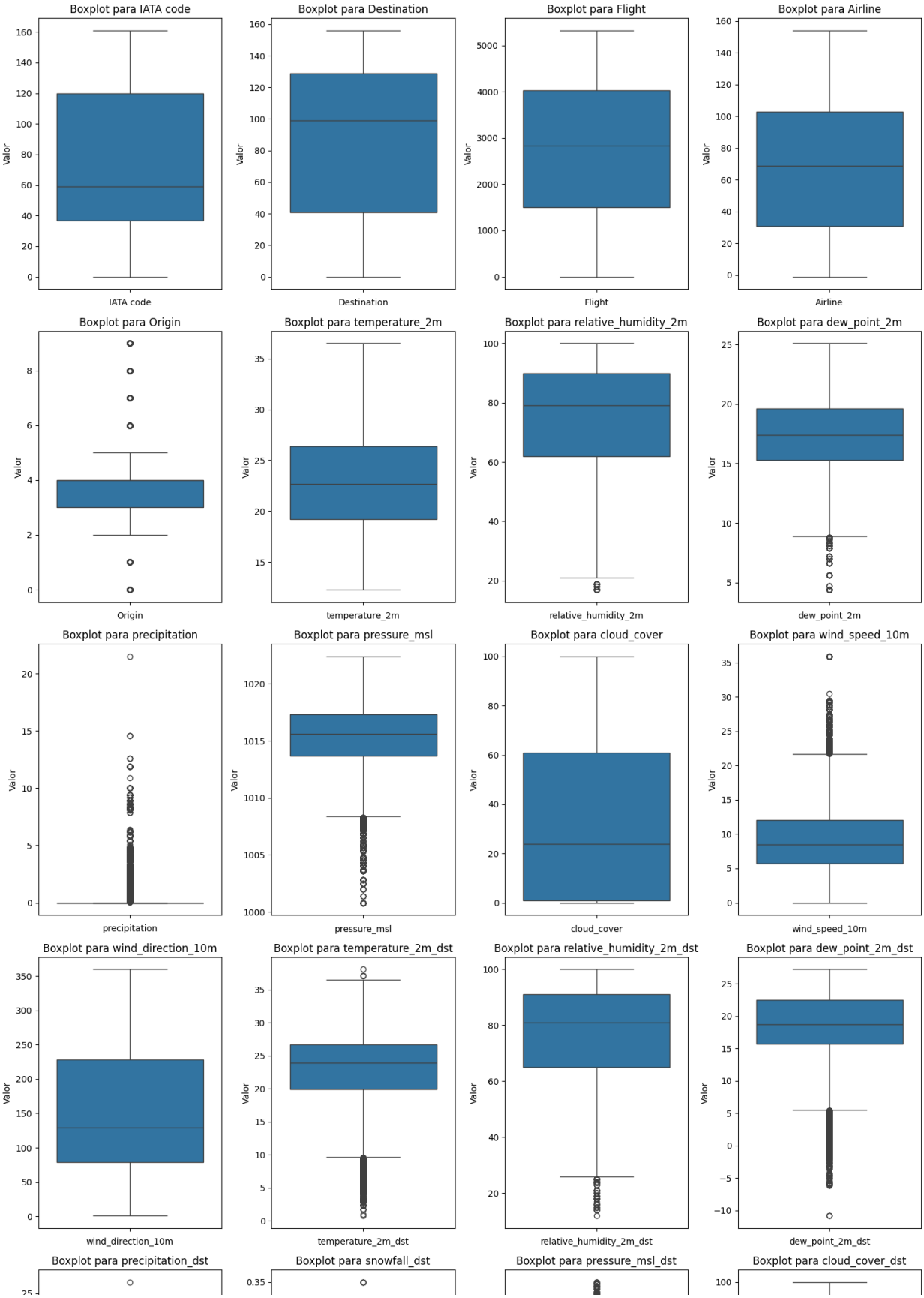
A média é uma medida de tendência central que pode ser significativamente influenciada por outliers. Portanto, ao comparar a média com a mediana, podemos estimar a presença de outliers.

```
dfComplete.describe()
```

{"type":"dataframe"}

Uma das técnicas mais comuns para identificar outliers é o uso de boxplots. Nessa representação gráfica, a linha dentro da caixa representa a mediana dos dados, enquanto a caixa em si representa o intervalo interquartil, que abrange os valores entre o primeiro e o terceiro quartil. Os whiskers (ou "bigodes") estendem-se a partir da caixa e representam os limites do intervalo dos dados. Tipicamente, esses limites são calculados como o valor do intervalo interquartil multiplicado por 1.5. Qualquer ponto fora desse intervalo é considerado um outlier.

```
fig, axs = plt.subplots(6, 4, figsize=(15, 5*6))
for i, column in enumerate(dfComplete.loc[:,
~dfComplete.columns.isin(['datetime', 'Status'])].columns):
  ax = axs[i // 4, i % 4]
  sns.boxplot(data=dfComplete[column], ax=ax)
  ax.set_title(f'Boxplot para {column}')
  ax.set_xlabel(column)
  ax.set_ylabel('Valor')
plt.tight_layout()
plt.show()
```


Boxplot para IATA code
Valor
IATA code
Boxplot para Destination
Valor
Destination
Boxplot para Flight
Valor
Flight
Boxplot para Airline
Valor
Airline
Boxplot para Origin
Valor
Origin
Boxplot para temperature_2m
Valor
temperature_2m
Boxplot para relative_humidity_2m
Valor
relative_humidity_2m
Boxplot para dew_point_2m
Valor
dew_point_2m
Boxplot para precipitation
Valor
precipitation
Boxplot para pressure_msl
Valor
pressure_msl
Boxplot para cloud_cover
Valor
cloud_cover
Boxplot para wind_speed_10m
Valor
wind_speed_10m
Boxplot para wind_direction_10m
Valor
wind_direction_10m
Boxplot para temperature_2m_dst
Valor
temperature_2m_dst
Boxplot para relative_humidity_2m_dst
Valor
relative_humidity_2m_dst
Boxplot para dew_point_2m_dst
Valor
dew_point_2m_dst
Boxplot para precipitation_dst
Boxplot para snowfall_dst
Boxplot para pressure_msl_dst
Boxplot para cloud_cover_dst

Os outliers serão substituídos por NaN, de modo que sejam tratados como valores ausentes. Esta abordagem baseia-se na premissa de que outliers não contribuem positivamente para o treinamento do modelo, sendo equivalentes à ausência de dados.

Ao substituir outliers por NaN, garantimos que esses valores extremos não distorçam as análises estatísticas e a modelagem preditiva. Além disso, essa prática facilita o tratamento uniforme de dados problemáticos, permitindo a aplicação consistente de técnicas de imputação ou exclusão de valores ausentes.

Z-Score Robusto

```
outliers_values = 0
for column in dfComplete.loc[:, ~dfComplete.columns.isin(['datetime',
'Status'])].columns:
  data = dfComplete[column]
  mediana = np.median(data)
  mad = np.median(np.abs(data - mediana))
  zscore_robusto = 0.6745 * (data - mediana) / mad
  outliers_values_column = np.sum(np.abs(zscore_robusto) > 3.5)
  outliers_values += outliers_values_column
  print(f'{column}: N° de outliers {outliers_values_column} e razão
{outliers_values_column/len(data)}')
print(f'\nValor total de outliers: {outliers_values}')
```

```
IATA code: N° de outliers 0 e razão 0.0
Destination: N° de outliers 0 e razão 0.0
Flight: N° de outliers 0 e razão 0.0
Airline: N° de outliers 0 e razão 0.0
Origin: N° de outliers 0 e razão 0.0
temperature_2m: N° de outliers 0 e razão 0.0
relative_humidity_2m: N° de outliers 0 e razão 0.0
dew_point_2m: N° de outliers 46 e razão 0.00043793674670119387
precipitation: N° de outliers 14817 e razão 0.14106323425807804
pressure_msl: N° de outliers 206 e razão 0.0019611949960966508
cloud_cover: N° de outliers 0 e razão 0.0
wind_speed_10m: N° de outliers 261 e razão 0.002484815019326339
wind_direction_10m: N° de outliers 0 e razão 0.0
temperature_2m_dst: N° de outliers 381 e razão 0.003627258706372932
relative_humidity_2m_dst: N° de outliers 30 e razão
0.0002856109217616482
dew_point_2m_dst: N° de outliers 540 e razão 0.005140996591709667
precipitation_dst: N° de outliers 23123 e razão 0.22013937812981968
snowfall_dst: N° de outliers 11 e razão 0.00010472400464593766
pressure_msl_dst: N° de outliers 572 e razão 0.0054456482415887585
cloud_cover_dst: N° de outliers 0 e razão 0.0
wind_speed_10m_dst: N° de outliers 885 e razão 0.008425522191968621
wind_direction_10m_dst: N° de outliers 0 e razão 0.0

Valor total de outliers: 40872
```

Tukey

```
def count_outliers(df, column):
  Q1 = df[column].quantile(0.25)
  Q3 = df[column].quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR
  return np.sum((df[column] < lower_bound) | (df[column] >
upper_bound))

outliers_values = 0
for column in dfComplete.loc[:, ~dfComplete.columns.isin(['datetime',
'Status'])].columns:
  outliers_values_column = count_outliers(dfComplete, column)
  outliers_values += outliers_values_column
  print(f'{column}: N° de outliers {outliers_values_column} e razão
{outliers_values_column/len(data)}')

print(f'\nValor total de outliers: {outliers_values}')
```

```
IATA code: N° de outliers 0 e razão 0.0
Destination: N° de outliers 0 e razão 0.0
Flight: N° de outliers 0 e razão 0.0
Airline: N° de outliers 0 e razão 0.0
Origin: N° de outliers 36399 e razão 0.3465317313734077
temperature_2m: N° de outliers 0 e razão 0.0
relative_humidity_2m: N° de outliers 45 e razão 0.00042841638264247227
dew_point_2m: N° de outliers 191 e razão 0.0018183895352158266
precipitation: N° de outliers 14817 e razão 0.14106323425807804
pressure_msl: N° de outliers 571 e razão 0.005436127877530037
cloud_cover: N° de outliers 0 e razão 0.0
wind_speed_10m: N° de outliers 1115 e razão 0.01061520592547459
wind_direction_10m: N° de outliers 0 e razão 0.0
temperature_2m_dst: N° de outliers 1153 e razão 0.010976979759706011
relative_humidity_2m_dst: N° de outliers 190 e razão
0.001808869171157105
dew_point_2m_dst: N° de outliers 2028 e razão 0.019307298311087417
precipitation_dst: N° de outliers 23123 e razão 0.22013937812981968
snowfall_dst: N° de outliers 11 e razão 0.00010472400464593766
pressure_msl_dst: N° de outliers 1334 e razão 0.012700165654334622
cloud_cover_dst: N° de outliers 0 e razão 0.0
wind_speed_10m_dst: N° de outliers 1967 e razão 0.018726556103505397
wind_direction_10m_dst: N° de outliers 0 e razão 0.0

Valor total de outliers: 82944
```

Apesar do Z-Score Robusto ser mais adequado para dados que não seguem uma distribuição normal, ele é mais sensível a valores extremos e requer definição de um limite para identificar outliers, o que pode ser subjetivo. Já o Método de Tukey (IQR) Assume uma distribuição normal dos dados, é uma abordagem mais simples e amplamente utilizada.

```
def remove_outliers(df, column):
  Q1 = df[column].quantile(0.25)
  Q3 = df[column].quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR
  return np.where((df[column] < lower_bound) | (df[column] >
upper_bound), np.nan, df[column])

df_clean = pd.DataFrame()
for column in dfComplete.loc[:, ~dfComplete.columns.isin(['datetime',
'Status', 'IATA code', 'Origin'])].columns:
  df_clean[column] = remove_outliers(dfComplete, column)

df_clean['Status'] = dfComplete['Status']
df_clean['datetime'] = dfComplete['datetime']
df_clean['IATA code'] = dfComplete['IATA code']
df_clean['Origin'] = dfComplete['Origin']
```

Isolation Florest

```
rng = np.random.RandomState(42)
clf = IsolationForest(max_samples=100, random_state=rng)
clf.fit(dfComplete.loc[:, ~dfComplete.columns.isin(['datetime',
'Status'])])
```

```
IsolationForest(max_samples=100,
                random_state=RandomState(MT19937) at 0x7913CAFB2940)
```

```
scores = clf.predict(dfComplete.loc[:,
~dfComplete.columns.isin(['datetime', 'Status'])])

dfComplete_clf = dfComplete.copy()
dfComplete_clf['outlier'] = scores
dfComplete_clf[dfComplete_clf['outlier'] == -1]
```

```
{"type":"dataframe"}
```

De acordo com a análise global do Isolation Florest, a qual considera múltiplas variáveis, quase mais vinte mil linhas devem ser descartadas por não representarem a distribuição dos dados, serem outliers. No entanto, a remoção não será realizada, dada a preocupação com os desbalanceamento dos dados. Buscamos evitar remoção de qualquer linha das classes minoritárias para ter o máximo de informações disponíveis.

Tratamento de Dados Ausentes

A identificação e tratamento dos valores ausentes no dataset são etapas essenciais no processo de pré-processamento de dados. É crucial determinar a porcentagem de dados faltantes em cada coluna. Essa análise permite avaliar a extensão do problema e decidir sobre a melhor abordagem para lidar com os dados ausentes.

Até um determinado limiar de porcentagem de dados faltantes, é viável aplicar estratégias de imputação ou preenchimento dos valores ausentes. Essas estratégias podem incluir a substituição dos valores ausentes por estatísticas descritivas, como a média, mediana ou moda da coluna, ou por valores previamente estabelecidos com base no conhecimento do domínio.

No entanto, acima de um certo limiar de porcentagem de dados faltantes, a coluna em questão pode perder seu poder informativo e relevância para a análise. Nesses casos, a remoção da coluna é frequentemente a abordagem mais apropriada, uma vez que manter colunas com uma quantidade significativa de valores ausentes pode distorcer os resultados da análise e prejudicar a eficácia do modelo.

Colunas

```
null_values = df_clean.isnull().sum() / len(df_clean)
null_values = null_values[null_values != 0]
null_values
```

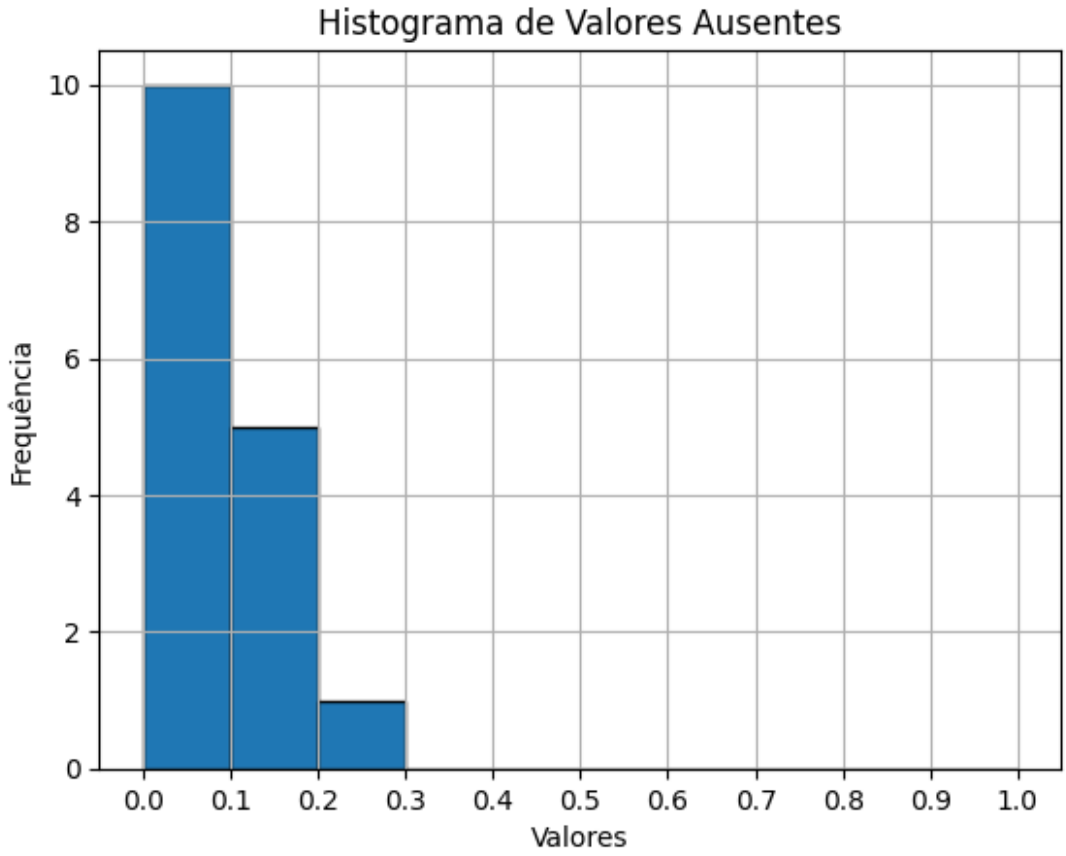
```
relative_humidity_2m        0.000428
dew_point_2m                0.001818
precipitation               0.141063
pressure_msl                0.005436
wind_speed_10m              0.010615
temperature_2m_dst          0.010977
relative_humidity_2m_dst    0.001809
dew_point_2m_dst            0.019307
precipitation_dst           0.220139
snowfall_dst                0.000105
pressure_msl_dst            0.012700
wind_speed_10m_dst          0.018727
Status                      0.173889
datetime                    0.173889
IATA code                   0.173889
Origin                      0.173889
dtype: float64
```

```
null_values = df_clean.isnull().sum() / len(df_clean)
null_values = null_values[null_values != 0]
null_values
```

```
relative_humidity_2m        0.000428
dew_point_2m                0.001818
precipitation               0.141063
pressure_msl                0.005436
wind_speed_10m              0.010615
temperature_2m_dst          0.010977
relative_humidity_2m_dst    0.001809
dew_point_2m_dst            0.019307
precipitation_dst           0.220139
snowfall_dst                0.000105
pressure_msl_dst            0.012700
wind_speed_10m_dst          0.018727
```

```
Status                          0.173889
datetime                        0.173889
IATA code                       0.173889
Origin                          0.173889
dtype: float64
```

```
plt.hist(null_values, bins=10, range=(0, 1), edgecolor='black')
plt.title('Histograma de Valores Ausentes')
plt.xlabel('Valores')
plt.ylabel('Frequência')
plt.xticks(np.arange(0, 1.1, 0.1))
plt.grid(True)
plt.show()
```



Após a análise do histograma e a definição do limiar de 30% para valores ausentes, optamos por remover todas as colunas em que mais há mais de 30% de dados faltantes. Essa decisão visa simplificar o treinamento do modelo e evitar a distorção da distribuição natural dos dados devido à necessidade de imputação de uma grande quantidade de valores ausentes.

Essa abordagem de pré-processamento permite reduzir a complexidade do modelo e preservar a integridade dos dados, concentrando-se em variáveis mais informativas e relevantes para a análise. Dessa forma, garantimos uma representação mais precisa e eficiente dos dados, contribuindo para a qualidade e a robustez do modelo resultante.

```
df_clean2 = df_clean.dropna(thresh=0.3*len(df_clean), axis=1)
df_clean.shape, df_clean2.shape
```

```
((105038, 24), (105038, 24))
```

```
colunas_valores_iguais = df_clean.columns[df_clean.apply(lambda x:
x.nunique()) == 1].tolist()
print(colunas_valores_iguais)

df_clean = df_clean.drop(columns=colunas_valores_iguais)
```

```
['precipitation', 'precipitation_dst', 'snowfall_dst']
```

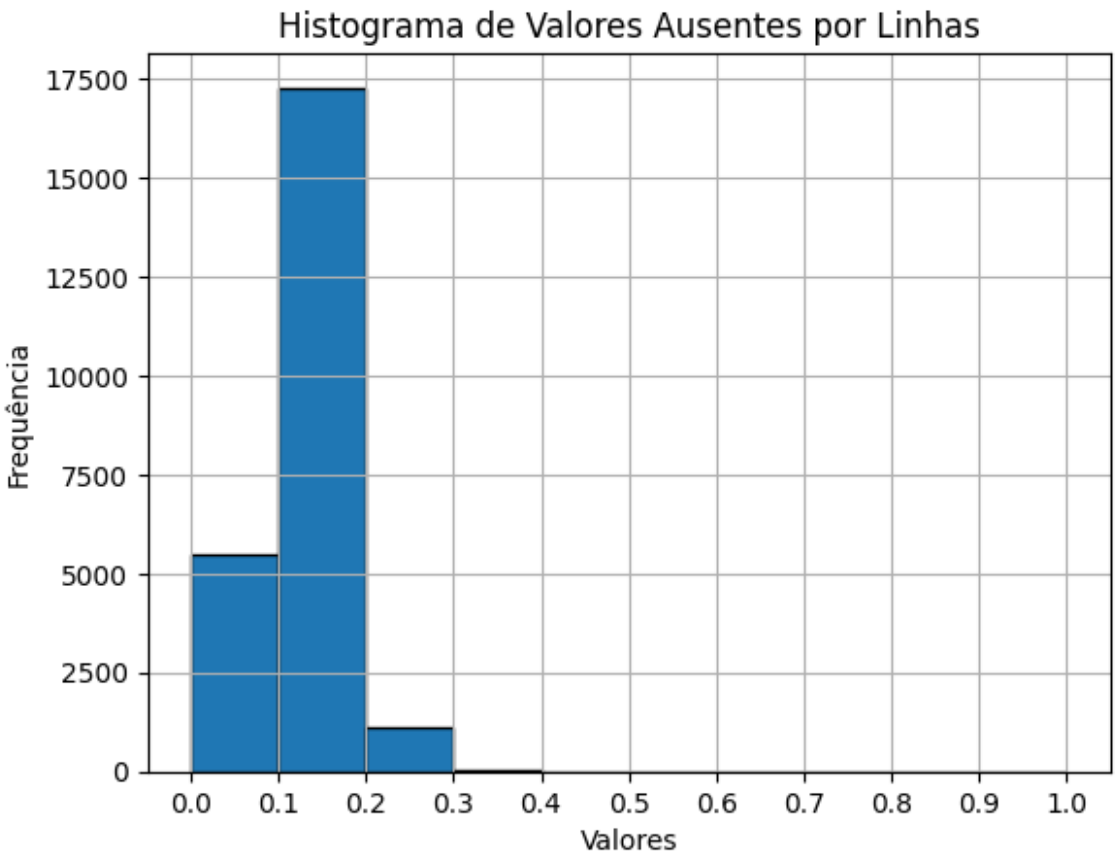
Linhas

Uma segunda análise é conduzida para verificar a quantidade de dados ausentes por instância. Caso essa quantidade supere o limiar estabelecido, considera-se que aquela amostra contribui pouco para a compreensão do problema real em questão.

```
null_values_lines = df_clean.isnull().sum(axis=1) /
len(df_clean.columns)
null_values_lines = null_values_lines[null_values_lines != 0]
null_values_lines
```

```
23        0.047619
24        0.047619
25        0.047619
195       0.190476
196       0.190476
            ...
105023    0.238095
105024    0.190476
105025    0.190476
105026    0.190476
105027    0.047619
Length: 23907, dtype: float64
```

```
plt.hist(null_values_lines, bins=10, range=(0, 1), edgecolor='black')
plt.title('Histograma de Valores Ausentes por Linhas')
plt.xlabel('Valores')
plt.ylabel('Frequência')
plt.xticks(np.arange(0, 1.1, 0.1))
plt.grid(True)
plt.show()
```




```
df_clean2 = df_clean.dropna(thresh=0.3*len(df_clean.columns), axis=0)
df_clean.shape, df_clean2.shape
```

```
((105038, 21), (105038, 21))
```

Remoção dos Valores Ausentes

O KNNImputer é uma classe da biblioteca scikit-learn usada para imputar valores ausentes em conjuntos de dados. Ele preenche os valores ausentes considerando os valores dos K vizinhos mais próximos que são mais semelhantes em termos das características observadas. Cada valor ausente é substituído pela média dos valores correspondentes dos vizinhos mais próximos.

As principais vantagens do KNNImputer incluem sua capacidade de aproveitar as relações e similaridades entre as amostras, resultando em estimativas mais precisas em comparação com técnicas simples, como substituição por média ou mediana. No entanto, o KNNImputer pode ser computacionalmente intensivo para conjuntos de dados muito grandes, o que pode ser uma limitação em termos de tempo e recursos computacionais. Por exemplo, foram necessários treze minutos para a realização dessa etapa pelo Colab.

```
df_imputed =
pd.DataFrame(KNNImputer(n_neighbors=3).fit_transform(df_clean.loc[:,
~df_clean.columns.isin(['datetime', 'Status'])]),
columns=df_clean.loc[:, ~df_clean.columns.isin(['datetime',
```

```
'Status'])].columns)
df_imputed
```

```
{"type":"dataframe","variable_name":"df_imputed"}
```

```
df_imputed['Status'] = df_clean['Status']
df_imputed['datetime'] = df_clean['datetime']

df_imputed.to_csv('/content/df_imputed.csv', index=False)
```

Normalização e Discretização

Para garantir que nenhuma variável tenha uma influência desproporcional no treinamento do modelo, é essencial normalizar todo o conjunto de dados.

Utilizaremos a técnica de Min-Max Scaling, que ajusta os valores para um intervalo entre 0 e 1, preservando suas distribuições relativas. A normalização pelo método Min-Max Scaling transforma os dados de modo que o valor mínimo de cada variável se torne 0 e o valor máximo se torne 1.

Essa abordagem assegura que todas as variáveis contribuam de maneira equilibrada durante o treinamento, evitando que variáveis com magnitudes maiores dominem o processo de aprendizado. Além disso, a normalização mantém a distribuição original dos dados, permitindo que o modelo capture as relações intrínsecas de maneira eficaz.

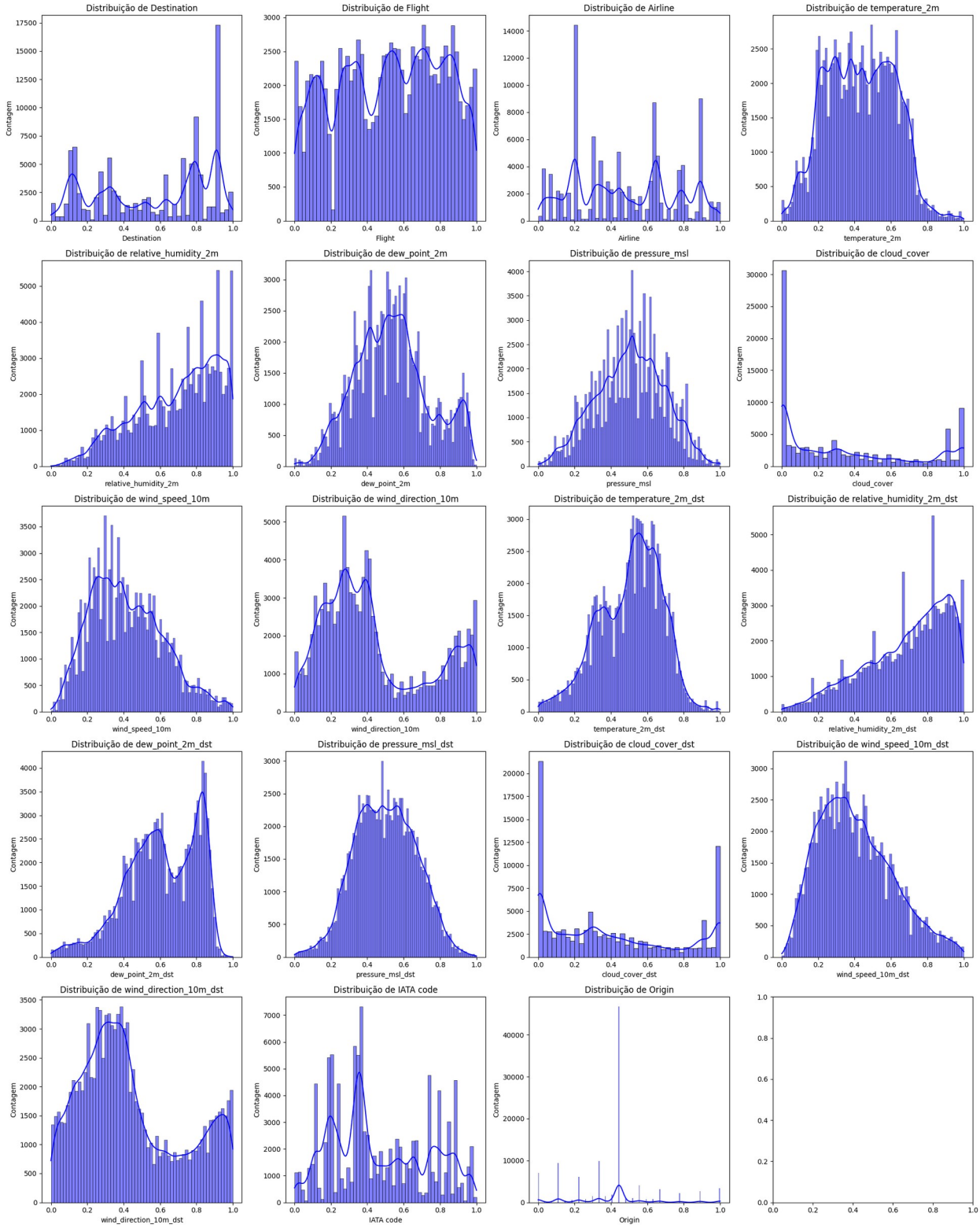
```
df_imputed = pd.read_csv('/content/df_imputed.csv')
df_imputed
```

```
{"type":"dataframe","variable_name":"df_imputed"}
```

```
scaler = MinMaxScaler()
df_normalized = scaler.fit_transform(df_imputed.loc[:,
~df_imputed.columns.isin(['datetime', 'Status'])])
df_normalized = pd.DataFrame(df_normalized, columns=df_imputed.loc[:,
~df_imputed.columns.isin(['datetime', 'Status'])].columns)
```

Exemplo de como a Discretização pode ser utilizada

```
fig, axs = plt.subplots(5, 4, figsize=(20, 5 * 5))
for i, coluna in enumerate(df_normalized.columns):
  ax = axs[i // 4, i % 4]

  sns.histplot(data=df_normalized, x=coluna, ax=ax, color='blue',
alpha=0.5, kde=True)

  ax.set_title(f'Distribuição de {coluna}')
  ax.set_xlabel(f'{coluna}')
  ax.set_ylabel('Contagem')
plt.tight_layout()
plt.show()
```




```
pd.cut(df_normalized['cloud_cover'], 10).value_counts().describe()
```

```
count       10.000000
mean     10503.800000
std      10798.006378
min       2778.000000
25%       5188.750000
50%       7628.000000
75%       9765.500000
max      39882.000000
Name: count, dtype: float64
```

```
pd.qcut(df_normalized['cloud_cover'], 4).value_counts().describe()
```

```
count        4.000000
mean     26259.500000
std       1284.370793
min      24903.000000
25%      25673.250000
50%      26072.000000
75%      26658.250000
max      27991.000000
Name: count, dtype: float64
```

```
pd.cut(df_normalized['cloud_cover_dst'], 10).value_counts().describe()
```

```
count       10.000000
mean     10503.800000
std       7770.464805
min       3271.000000
25%       5794.500000
50%       8850.000000
75%      11196.750000
max      30018.000000
Name: count, dtype: float64
```

```
pd.qcut(df_normalized['cloud_cover_dst'], 4).value_counts().describe()
```

```
count        4.000000
mean     26259.500000
std        507.921582
min      25669.000000
25%      26080.000000
50%      26229.500000
75%      26409.000000
max      26910.000000
Name: count, dtype: float64
```

```
df_normalized['Status'] = df_imputed['Status']
df_normalized['datetime'] = df_imputed['datetime']

df_normalized.to_csv('/content/df_normalized.csv', index=False)
```

Testes de Hipóteses

- Comparação de valores de categorias e visualizar diferenças

```
dfComplete = pd.read_csv('/content/df_normalized.csv')
dfComplete
```


```
{"type":"dataframe","variable_name":"dfComplete"}
```


O teste de hipótese é um procedimento estatístico usado para tomar decisões sobre a validade de uma afirmação (hipótese) com base em dados amostrais. Envolve formular duas hipóteses: a hipótese nula (H0), que representa a situação atual ou uma posição de não-efeito, e a hipótese alternativa (H1), que representa uma mudança ou um efeito. O teste então calcula a probabilidade (valor p) de observar os dados amostrais se a hipótese nula fosse verdadeira. Com base no valor p e em um nível de significância predefinido (geralmente 0,05), decidimos se rejeitamos ou não a hipótese nula, ajudando a inferir se há evidência estatística suficiente para apoiar a hipótese alternativa.

- Para cada variável, dividimos os dados em duas amostras: voos cancelados e voos não cancelados.
- Teste de Normalidade (teste de Shapiro-Wilk).
- Escolha do Teste Estatístico: Dependendo do resultado do teste de normalidade, escolhemos entre o teste t de Student (para distribuições normais) e o teste de Mann-Whitney U (para distribuições não normais).
- Comparar o valor p para decidir se rejeitamos ou não a hipótese nula.

```
dfComplete.columns
```

```
Index(['Destination', 'Flight', 'Airline', 'temperature_2m',
       'relative_humidity_2m', 'dew_point_2m', 'pressure_msl',
'cloud_cover',
       'wind_speed_10m', 'wind_direction_10m', 'temperature_2m_dst',
       'relative_humidity_2m_dst', 'dew_point_2m_dst',
'pressure_msl_dst',
       'cloud_cover_dst', 'wind_speed_10m_dst',
'wind_direction_10m_dst',
       'IATA code', 'Origin', 'Status', 'datetime'],
      dtype='object')
```

```
alpha = 0.05

colunas_clima = [
    'temperature_2m', 'relative_humidity_2m', 'dew_point_2m',
'pressure_msl', 'cloud_cover',
    'wind_speed_10m', 'wind_direction_10m', 'temperature_2m_dst',
'relative_humidity_2m_dst',
    'dew_point_2m_dst', 'pressure_msl_dst', 'cloud_cover_dst',
'wind_speed_10m_dst', 'wind_direction_10m_dst'
]

def realizar_teste(coluna):
```

```
    cancelados = dfComplete[dfComplete["Status"] == "Cancelled"]
[coluna]
    nao_cancelados = dfComplete[dfComplete["Status"] != "Cancelled"]
[coluna]

    shapiro_cancelados = shapiro(cancelados)
    shapiro_nao_cancelados = shapiro(nao_cancelados)

    if shapiro_cancelados.pvalue > alpha and
shapiro_nao_cancelados.pvalue > alpha:
        t_stat, p_value = ttest_ind(cancelados, nao_cancelados)
        teste = "t de Student"
    else:
        u_stat, p_value = mannwhitneyu(cancelados, nao_cancelados)
        teste = "Mann-Whitney U"

    if p_value < alpha:
        resultado = f"Rejeitar a hipótese nula: Há diferença
significativa na taxa de cancelamento com relação a {coluna}."
    else:
        resultado = f"Não rejeitar a hipótese nula: Não há diferença
significativa na taxa de cancelamento com relação a {coluna}."

    return coluna, teste, p_value, resultado

resultados = [realizar_teste(coluna) for coluna in colunas_clima]

# Exibir os resultados
for coluna, teste, p_value, resultado in resultados:
    print(f"Variável: {coluna}")
    print(f"Hipótese Nula (H0): Não há diferença na taxa de
cancelamento com relação à variável climática {coluna}.")
    print(f"Hipótese Alternativa (H1): Há diferença na taxa de
cancelamento com relação à variável climática {coluna}.")
    print(f"Teste escolhido: {teste}")
    print(f"p-value: {p_value}")
    print(resultado)
    print("-" * 80)
```

```
/usr/local/lib/python3.10/dist-packages/scipy/stats/
_morestats.py:1882: UserWarning: p-value may not be accurate for N >
5000.
  warnings.warn("p-value may not be accurate for N > 5000.")

Variável: temperature_2m
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática temperature_2m.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática temperature_2m.
Teste escolhido: Mann-Whitney U
```

```
p-value: 0.12490515651932953
Não rejeitar a hipótese nula: Não há diferença significativa na taxa
de cancelamento com relação a temperature_2m.
--------------------------------------------------------------------
----------
Variável: relative_humidity_2m
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática relative_humidity_2m.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática relative_humidity_2m.
Teste escolhido: Mann-Whitney U
p-value: 4.4654174247014924e-08
Rejeitar a hipótese nula: Há diferença significativa na taxa de
cancelamento com relação a relative_humidity_2m.
--------------------------------------------------------------------
----------
Variável: dew_point_2m
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática dew_point_2m.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática dew_point_2m.
Teste escolhido: Mann-Whitney U
p-value: 6.055909338043278e-06
Rejeitar a hipótese nula: Há diferença significativa na taxa de
cancelamento com relação a dew_point_2m.
--------------------------------------------------------------------
----------
Variável: pressure_msl
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática pressure_msl.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática pressure_msl.
Teste escolhido: Mann-Whitney U
p-value: 9.381289408631887e-06
Rejeitar a hipótese nula: Há diferença significativa na taxa de
cancelamento com relação a pressure_msl.
--------------------------------------------------------------------
----------
Variável: cloud_cover
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática cloud_cover.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática cloud_cover.
Teste escolhido: Mann-Whitney U
p-value: 0.022594735173525986
Rejeitar a hipótese nula: Há diferença significativa na taxa de
cancelamento com relação a cloud_cover.
--------------------------------------------------------------------
----------
```

```
Variável: wind_speed_10m
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática wind_speed_10m.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática wind_speed_10m.
Teste escolhido: Mann-Whitney U
p-value: 1.0040451762848916e-13
Rejeitar a hipótese nula: Há diferença significativa na taxa de
cancelamento com relação a wind_speed_10m.
----------------------------------------------------------------------
----------
Variável: wind_direction_10m
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática wind_direction_10m.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática wind_direction_10m.
Teste escolhido: Mann-Whitney U
p-value: 0.16520777535472897
Não rejeitar a hipótese nula: Não há diferença significativa na taxa
de cancelamento com relação a wind_direction_10m.
----------------------------------------------------------------------
----------
Variável: temperature_2m_dst
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática temperature_2m_dst.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática temperature_2m_dst.
Teste escolhido: Mann-Whitney U
p-value: 0.00014007622244675713
Rejeitar a hipótese nula: Há diferença significativa na taxa de
cancelamento com relação a temperature_2m_dst.
----------------------------------------------------------------------
----------
Variável: relative_humidity_2m_dst
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática relative_humidity_2m_dst.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática relative_humidity_2m_dst.
Teste escolhido: Mann-Whitney U
p-value: 1.6779202356498529e-06
Rejeitar a hipótese nula: Há diferença significativa na taxa de
cancelamento com relação a relative_humidity_2m_dst.
----------------------------------------------------------------------
----------
Variável: dew_point_2m_dst
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática dew_point_2m_dst.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática dew_point_2m_dst.
```

```
Teste escolhido: Mann-Whitney U
p-value: 0.18071720901354582
Não rejeitar a hipótese nula: Não há diferença significativa na taxa
de cancelamento com relação a dew_point_2m_dst.
----------------------------------------------------------------------
----------
Variável: pressure_msl_dst
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática pressure_msl_dst.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática pressure_msl_dst.
Teste escolhido: Mann-Whitney U
p-value: 1.0246802728975749e-08
Rejeitar a hipótese nula: Há diferença significativa na taxa de
cancelamento com relação a pressure_msl_dst.
----------------------------------------------------------------------
----------
Variável: cloud_cover_dst
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática cloud_cover_dst.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática cloud_cover_dst.
Teste escolhido: Mann-Whitney U
p-value: 0.021132137782631046
Rejeitar a hipótese nula: Há diferença significativa na taxa de
cancelamento com relação a cloud_cover_dst.
----------------------------------------------------------------------
----------
Variável: wind_speed_10m_dst
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática wind_speed_10m_dst.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática wind_speed_10m_dst.
Teste escolhido: Mann-Whitney U
p-value: 0.28231480224049377
Não rejeitar a hipótese nula: Não há diferença significativa na taxa
de cancelamento com relação a wind_speed_10m_dst.
----------------------------------------------------------------------
----------
Variável: wind_direction_10m_dst
Hipótese Nula (H0): Não há diferença na taxa de cancelamento com
relação à variável climática wind_direction_10m_dst.
Hipótese Alternativa (H1): Há diferença na taxa de cancelamento com
relação à variável climática wind_direction_10m_dst.
Teste escolhido: Mann-Whitney U
p-value: 0.01750429101314592
Rejeitar a hipótese nula: Há diferença significativa na taxa de
cancelamento com relação a wind_direction_10m_dst.
```

```
----------------------------------------------------------------------
----------
```

O número de voos cancelados é muito menor do que o número de voos não cancelados, isso pode afetar os testes de normalidade. Pequenas amostras são menos prováveis de passar nos testes de normalidade, o que pode resultar em uma escolha mais frequente do teste Mann-Whitney U.

O teste Mann-Whitney U é mais robusto em relação a outliers e distribuições não normais, tornando-o uma escolha segura quando há dúvidas sobre a normalidade dos dados ou quando os dados são muito desbalanceados.