ANÁLISE DOS PRINCIPAIS FATORES CONTRIBUINTES E COMPARAÇÃO DAS OCORRÊNCIAS AERONÁUTICA EM ACIDENTES E INCIDENTES NACIONAIS

Coleta dos Dados

In [1]:
```
#Importando as Bibliotecas

import sys
import IPython
import pandas as pd
import seaborn as sns
import matplotlib as plt
import codecs
import matplotlib.pyplot as plt
import scipy.stats as stats
```

In [2]:
```
# Configura o Pandas para mostrar todas as linhas e colunas
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

In [3]:
```
sys.stdout.encoding = 'utf-8'
```

In [4]:
```
#Verificando as versões do Jupyter e Python
print("Versão do Jupyter Notebook:", IPython.__version__)
print("Versão do Python:", sys.version)
```

```
Versão do Jupyter Notebook: 7.22.0
Versão do Python: 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
```

In [5]:
```
# Consultando tabela de ocorrencia e seus tipos
df_ocorrencia = pd.read_csv('ocorrencia.csv', sep = ';' , encoding='latin-1')
df_ocorrencia.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6769 entries, 0 to 6768
Data columns (total 22 columns):
 #   Column                          Non-Null Count  Dtype 
---  ------                          --------------  ----- 
 0   codigo_ocorrencia               6769 non-null   int64 
 1   codigo_ocorrencia1              6769 non-null   int64 
 2   codigo_ocorrencia2              6769 non-null   int64 
 3   codigo_ocorrencia3              6769 non-null   int64 
 4   codigo_ocorrencia4              6769 non-null   int64 
 5   ocorrencia_classificacao        6769 non-null   object
 6   ocorrencia_latitude             5135 non-null   object
 7   ocorrencia_longitude            5135 non-null   object
 8   ocorrencia_cidade               6769 non-null   object
 9   ocorrencia_uf                   6769 non-null   object
 10  ocorrencia_pais                 6769 non-null   object
 11  ocorrencia_aerodromo            6769 non-null   object
 12  ocorrencia_dia                  6769 non-null   object
 13  ocorrencia_hora                 6767 non-null   object
 14  investigacao_aeronave_liberada  6531 non-null   object
 15  investigacao_status             6428 non-null   object
 16  divulgacao_relatorio_numero     5987 non-null   object
 17  divulgacao_relatorio_publicado  6769 non-null   object
 18  divulgacao_dia_publicacao       1781 non-null   object
 19  total_recomendacoes             6769 non-null   int64 
 20  total_aeronaves_envolvidas      6769 non-null   int64 
 21  ocorrencia_saida_pista          6769 non-null   object
dtypes: int64(7), object(15)
memory usage: 1.1+ MB
```

In [6]:
```
# Consultando tabela de ocorrencia_tipo e seus tipos
df_ocorrencia_tipo = pd.read_csv('ocorrencia_tipo.csv', sep = ';', encoding='utf-8')
df_ocorrencia_tipo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7100 entries, 0 to 7099
Data columns (total 4 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   codigo_ocorrencia1        7100 non-null   int64 
 1   ocorrencia_tipo           7099 non-null   object
 2   ocorrencia_tipo_categoria  7099 non-null   object
 3   taxonomia_tipo_icao       7099 non-null   object
dtypes: int64(1), object(3)
memory usage: 222.0+ KB
```

In [7]:
```
# Consultando tabela de recomendacao e seus tipos
df_recomendacao = pd.read_csv('recomendacao.csv', sep = ';', encoding='utf-8')
df_recomendacao.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2120 entries, 0 to 2119
Data columns (total 9 columns):
 #   Column                          Non-Null Count  Dtype 
---  ------                          --------------  ----- 
 0   codigo_ocorrencia4              2120 non-null   int64 
 1   recomendacao_numero             2120 non-null   object
 2   recomendacao_dia_assinatura     2120 non-null   object
 3   recomendacao_dia_encaminhamento  2120 non-null   object
 4   recomendacao_dia_feedback       1689 non-null   object
 5   recomendacao_conteudo           2116 non-null   object
 6   recomendacao_status             2120 non-null   object
 7   recomendacao_destinatario_sigla  2120 non-null   object
 8   recomendacao_destinatario       2120 non-null   object
dtypes: int64(1), object(8)
memory usage: 149.2+ KB
```

In [8]:
```
# Consultando tabela de fator_contribuinte e seus tipos
df_fator_contribuinte = pd.read_csv('fator_contribuinte.csv', sep = ';', encoding='utf-8')
df_fator_contribuinte.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4776 entries, 0 to 4775
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   codigo_ocorrencia3   4776 non-null   int64 
 1   fator_nome           4776 non-null   object
 2   fator_aspecto        4776 non-null   object
 3   fator_condicionante  4776 non-null   object
 4   fator_area           4776 non-null   object
dtypes: int64(1), object(4)
memory usage: 186.7+ KB
```

In [9]:
```
# Consultando tabela de aeronave e seus tipos
df_aeronave = pd.read_csv('aeronave.csv', sep = ';', encoding='utf-8')
df_aeronave.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6339 entries, 0 to 6338
Data columns (total 23 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   codigo_ocorrencia2           6339 non-null   int64  
 1   aeronave_matricula           6306 non-null   object 
 2   aeronave_operador_categoria  6306 non-null   object 
 3   aeronave_tipo_veiculo        6306 non-null   object 
 4   aeronave_fabricante          6306 non-null   object 
 5   aeronave_modelo              6306 non-null   object 
 6   aeronave_tipo_icao           6306 non-null   object 
 7   aeronave_motor_tipo          6289 non-null   object 
 8   aeronave_motor_quantidade    6306 non-null   object 
 9   aeronave_pmd                 6306 non-null   float64
 10  aeronave_pmd_categoria       6306 non-null   float64
 11  aeronave_assentos            6064 non-null   float64
 12  aeronave_ano_fabricacao      6078 non-null   float64
 13  aeronave_pais_fabricante     6306 non-null   object 
 14  aeronave_pais_registro       6306 non-null   object 
 15  aeronave_registro_categoria  6306 non-null   object 
 16  aeronave_registro_segmento   6306 non-null   object 
 17  aeronave_voo_origem          6305 non-null   object 
 18  aeronave_voo_destino         6305 non-null   object 
 19  aeronave_fase_operacao       6306 non-null   object 
 20  aeronave_tipo_operacao       6306 non-null   object 
 21  aeronave_nivel_dano          6306 non-null   object 
 22  aeronave_fatalidades_total   6306 non-null   float64
dtypes: float64(5), int64(1), object(17)
memory usage: 1.1+ MB
```

Processamento/Tratamento de Dados

Uma etapa muito importante um projeto de ciência de Dados é o tratamento de dados. Através desse processo/etapa, os dados são limpos, padronizados e validados, garantindo a sua qualidade e confiabilidade. Dados bem tratados proporcionam resultados mais precisos e insights relevantes. Além disso, o tratamento adequado dos dados contribui para a identificação de padrões, tendências e outliers, facilitando a detecção de problemas e oportunidade. O tratamento de dados é essencial para a construção de modelos preditivos e de aprendizado de máquina.

1 - Alteração de Tipo de Dados de Colunas:

Nessa etapa de do processamento será feito a alteração dos tipos de dados necessários!

Ocorrencia:

In [10]:

```
#Ocorrencia
#Transformando colunas object para string

# Lista das colunas para conversão para string
colunas_para_str_ocorrencia = ['ocorrencia_classificacao', 'ocorrencia_cidade', 'ocorrencia_uf',
                               'ocorrencia_pais', 'ocorrencia_aerodromo']

# Convertendo as colunas para o tipo 'str' usando o método astype
df_ocorrencia[colunas_para_str_ocorrencia] = df_ocorrencia[colunas_para_str_ocorrencia].astype(str)

print(df_ocorrencia.dtypes)
```

```
codigo_ocorrencia                int64
codigo_ocorrencia1               int64
codigo_ocorrencia2               int64
codigo_ocorrencia3               int64
codigo_ocorrencia4               int64
ocorrencia_classificacao        object
ocorrencia_latitude             object
ocorrencia_longitude            object
ocorrencia_cidade               object
ocorrencia_uf                   object
ocorrencia_pais                 object
ocorrencia_aerodromo            object
ocorrencia_dia                  object
ocorrencia_hora                 object
investigacao_aeronave_liberada  object
investigacao_status             object
divulgacao_relatorio_numero     object
divulgacao_relatorio_publicado  object
divulgacao_dia_publicacao       object
total_recomendacoes              int64
total_aeronaves_envolvidas       int64
ocorrencia_saida_pista          object
dtype: object
```

Nessa etapa do tratamento foi criada uma lista com o nome ‘colunas_para_str_ocorrencia’, que contém os nomes das colunas que serão convertidas para o tipo "str" (string). Em seguida é usado o método ‘astype(str)’ para converter as colunas da lista ‘colunas_para_str_ocorrencia’ para o tipo "str". Isso significa que os valores em cada coluna da lista serão tratados como strings. Mesmo que os valores originais das colunas fossem de outros tipos (números inteiros ou floats), após essa operação, eles serão interpretados como strings.

In [11]:
```
#Ocorrencia
#Transformando colunas object para data

# convertendo objetc
df_ocorrencia['ocorrencia_dia'] = pd.to_datetime(df_ocorrencia['ocorrencia_dia'])
df_ocorrencia['divulgacao_dia_publicacao'] = pd.to_datetime(df_ocorrencia['divulgacao_dia_publicacao'])

print(df_ocorrencia.dtypes)
```

```
codigo_ocorrencia                    int64
codigo_ocorrencia1                   int64
codigo_ocorrencia2                   int64
codigo_ocorrencia3                   int64
codigo_ocorrencia4                   int64
ocorrencia_classificacao            object
ocorrencia_latitude                 object
ocorrencia_longitude                object
ocorrencia_cidade                   object
ocorrencia_uf                       object
ocorrencia_pais                     object
ocorrencia_aerodromo                object
ocorrencia_dia              datetime64[ns]
ocorrencia_hora                     object
investigacao_aeronave_liberada      object
investigacao_status                 object
divulgacao_relatorio_numero         object
divulgacao_relatorio_publicado      object
divulgacao_dia_publicacao   datetime64[ns]
total_recomendacoes                  int64
total_aeronaves_envolvidas           int64
ocorrencia_saida_pista              object
dtype: object
```

Ainda no processamento para a tabela de ‘ocorrencia’, foi utilizado o método de ‘pd.to_datetime()’ para converter as colunas 'ocorrencia_dia' e 'divulgacao_dia_publicacao' para o tipo de dado "datetime". O resultado da conversão é atribuído de volta às mesmas colunas, substituindo os valores antigos, mas agora, eles serão tratados como objetos de data.

Ocorrencia_tipo

In [12]:
```
#Ocorrencia_tipo

# Lista das colunas para conversão para string
colunas_para_str_ocorrencia_tipo = ['ocorrencia_tipo','ocorrencia_tipo_categoria','taxonomia_tipo_icao']

# Convertendo as colunas para o tipo 'str' usando o método astype
df_ocorrencia_tipo[colunas_para_str_ocorrencia_tipo] = df_ocorrencia_tipo[colunas_para_str_ocorrencia_tipo].astype

print(df_ocorrencia_tipo.dtypes)
```

```
codigo_ocorrencia1           int64
ocorrencia_tipo             object
ocorrencia_tipo_categoria   object
taxonomia_tipo_icao         object
dtype: object
```

No processo da tabela ocorrencia_tipo é criada uma lista ‘colunas_para_str_ocorrencia_tipo’, que contém os nomes das colunas que serão convertidas para o tipo "str" (string). o método usado é astype(str) para converter as colunas da lista ‘colunas_para_str_ocorrencia_tipo’ para o tipo "str". Isso significa que os valores em cada coluna da lista serão tratados como strings, mesmo que os valores originais das colunas fossem de outros tipos (números inteiros ou floats), após essa operação, eles serão interpretados como strings.

Recomendacao

In [13]:

```
#recomendacao

# Lista das colunas para conversão para string
colunas_para_str_recomendacao = ['recomendacao_numero','recomendacao_conteudo','recomendacao_status',
                                 'recomendacao_destinatario_sigla','recomendacao_destinatario']

# Convertendo as colunas para o tipo 'str' usando o método astype
df_recomendacao[colunas_para_str_recomendacao] = df_recomendacao[colunas_para_str_recomendacao].astype(str)

print(df_recomendacao.dtypes)
```

```
codigo_ocorrencia4                int64
recomendacao_numero              object
recomendacao_dia_assinatura      object
recomendacao_dia_encaminhamento  object
recomendacao_dia_feedback        object
recomendacao_conteudo            object
recomendacao_status              object
recomendacao_destinatario_sigla  object
recomendacao_destinatario        object
dtype: object
```

É definida uma lista 'colunas_para_str_recomendacao', que contém os nomes das colunas que serão convertidas para o tipo 'str', o método usado é astype(str) para converter as colunas da lista ‘colunas_para_str_recomendacao’ para o tipo "str". Isso significa que os valores em cada coluna da lista serão tratados como strings, mesmo que os valores originais das colunas fossem de outros tipos (números inteiros ou floats), após essa operação, eles serão interpretados como strings.

In [14]:

```
# Recomendacao
#Transformando colunas object para data

# convertendo objetc
df_recomendacao['recomendacao_dia_assinatura'] = pd.to_datetime(df_recomendacao['recomendacao_dia_assinatura'])
df_recomendacao['recomendacao_dia_encaminhamento'] = pd.to_datetime(df_recomendacao['recomendacao_dia_encaminhamen
df_recomendacao['recomendacao_dia_feedback'] = pd.to_datetime(df_recomendacao['recomendacao_dia_feedback'])

print(df_recomendacao.dtypes)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\arrays\datetimes.py in objects_to_datetime64ns(data, dayfirst, yearfirst, u
tc, errors, require_iso8601, allow_object)
   2084         try:
-> 2085             values, tz_parsed = conversion.datetime_to_datetime64(data)
   2086             # If tzaware, these values represent unix timestamps, so we

pandas\_libs\tslibs\conversion.pyx in pandas._libs.tslibs.conversion.datetime_to_datetime64()

TypeError: Unrecognized value type: <class 'str'>

During handling of the above exception, another exception occurred:

OutOfBoundsDatetime                       Traceback (most recent call last)
<ipython-input-14-457767f383ee> in <module>
      5 df_recomendacao['recomendacao_dia_assinatura'] = pd.to_datetime(df_recomendacao['recomendacao_dia_assinatur
a'])
      6 df_recomendacao['recomendacao_dia_encaminhamento'] = pd.to_datetime(df_recomendacao['recomendacao_dia_encamin
hamento'])
----> 7 df_recomendacao['recomendacao_dia_feedback'] = pd.to_datetime(df_recomendacao['recomendacao_dia_feedback'])
      8
      9 print(df_recomendacao.dtypes)

~\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py in to_datetime(arg, errors, dayfirst, yearfirst, utc, fo
rmat, exact, unit, infer_datetime_format, origin, cache)
    799                 result = result.tz_localize(tz)
    800     elif isinstance(arg, ABCSeries):
--> 801         cache_array = _maybe_cache(arg, format, cache, convert_listlike)
    802         if not cache_array.empty:
    803             result = arg.map(cache_array)

~\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py in _maybe_cache(arg, format, cache, convert_listlike)
    176         unique_dates = unique(arg)
    177         if len(unique_dates) < len(arg):
--> 178             cache_dates = convert_listlike(unique_dates, format)
    179             cache_array = Series(cache_dates, index=unique_dates)
    180     return cache_array

~\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py in _convert_listlike_datetimes(arg, format, name, tz, un
it, errors, infer_datetime_format, dayfirst, yearfirst, exact)
```

```
    463         assert format is None or infer_datetime_format
    464         utc = tz == "utc"
--> 465         result, tz_parsed = objects_to_datetime64ns(
    466             arg,
    467             dayfirst=dayfirst,

~\anaconda3\lib\site-packages\pandas\core\arrays\datetimes.py in objects_to_datetime64ns(data, dayfirst, yearfirst, u
tc, errors, require_iso8601, allow_object)
   2088             return values.view("i8"), tz_parsed
   2089         except (ValueError, TypeError):
-> 2090             raise e
   2091
   2092     if tz_parsed is not None:

~\anaconda3\lib\site-packages\pandas\core\arrays\datetimes.py in objects_to_datetime64ns(data, dayfirst, yearfirst, u
tc, errors, require_iso8601, allow_object)
   2073
   2074     try:
-> 2075         result, tz_parsed = tslib.array_to_datetime(
   2076             data,
   2077             errors=errors,

pandas\_libs\tslib.pyx in pandas._libs.tslib.array_to_datetime()

pandas\_libs\tslib.pyx in pandas._libs.tslib.array_to_datetime()

pandas\_libs\tslib.pyx in pandas._libs.tslib.array_to_datetime()

pandas\_libs\tslib.pyx in pandas._libs.tslib.array_to_datetime()

pandas\_libs\tslibs\np_datetime.pyx in pandas._libs.tslibs.np_datetime.check_dts_bounds()

OutOfBoundsDatetime: Out of bounds nanosecond timestamp: 2-11-28 00:00:00
```

Esse erro acontece porque existem valores nas colunas que estão fora do alcance de identificação de timestamps da biblioteca pandas. Uma forma de identificar os erros é definir o argumento errors='coerce' no método pd.to_datetime(), que converterá os valores inválidos para NaT (Not-a-Time) e permitirá que o processo continue sem interrupções

In [15]:
```
# convertendo para datetime com o argumento errors='coerce'
df_recomendacao['recomendacao_dia_assinatura'] = pd.to_datetime(df_recomendacao['recomendacao_dia_assinatura'],
                                                                errors='coerce')
df_recomendacao['recomendacao_dia_encaminhamento'] = pd.to_datetime(df_recomendacao['recomendacao_dia_encaminhamen
                                                                    errors='coerce')
df_recomendacao['recomendacao_dia_feedback'] = pd.to_datetime(df_recomendacao['recomendacao_dia_feedback'],
                                                              errors='coerce')
print(df_recomendacao.dtypes)
```

```
codigo_ocorrencia4                      int64
recomendacao_numero                    object
recomendacao_dia_assinatura    datetime64[ns]
recomendacao_dia_encaminhamento datetime64[ns]
recomendacao_dia_feedback      datetime64[ns]
recomendacao_conteudo                  object
recomendacao_status                    object
recomendacao_destinatario_sigla        object
recomendacao_destinatario              object
dtype: object
```

Fator_contribuinte

In [16]:
```
# Fator_contribuinte
# Lista das colunas para conversão para string
colunas_para_str_fator_contri = ['fator_nome','fator_aspecto','fator_condicionante','fator_area']

# convertendo objetc
df_fator_contribuinte[colunas_para_str_fator_contri]=df_fator_contribuinte[colunas_para_str_fator_contri].astype(s

print(df_fator_contribuinte.dtypes)
```

```
codigo_ocorrencia3      int64
fator_nome             object
fator_aspecto          object
fator_condicionante    object
fator_area             object
dtype: object
```

É definida uma lista 'colunas_para_str_fator_contri', que contém os nomes das colunas que serão convertidas para o tipo 'str', o método usado é astype(str) para converter as colunas da lista ‘colunas_para_str_fator_contri’ para o tipo "str". Isso significa que os valores em cada coluna da lista serão tratados como strings, mesmo que os valores originais das colunas fossem de outros tipos (números inteiros ou floats), após essa operação, eles serão interpretados como strings.

Aeronave

In [17]:

```
#aeronave

# Lista das colunas para conversão de string
colunas_para_str_aeronave = ['aeronave_matricula','aeronave_operador_categoria','aeronave_tipo_veiculo',
                             'aeronave_fabricante','aeronave_modelo','aeronave_tipo_icao','aeronave_motor_tipo',
                             'aeronave_motor_quantidade','aeronave_pais_fabricante','aeronave_pais_registro',
                            'aeronave_registro_categoria','aeronave_registro_segmento','aeronave_voo_origem',
                            'aeronave_voo_destino','aeronave_fase_operacao','aeronave_tipo_operacao','aeronave_niv

# Convertendo
df_aeronave[colunas_para_str_aeronave] = df_aeronave[colunas_para_str_aeronave].astype(str)

print(df_aeronave.dtypes)
```

```
codigo_ocorrencia2              int64
aeronave_matricula             object
aeronave_operador_categoria    object
aeronave_tipo_veiculo          object
aeronave_fabricante            object
aeronave_modelo                object
aeronave_tipo_icao             object
aeronave_motor_tipo            object
aeronave_motor_quantidade      object
aeronave_pmd                  float64
aeronave_pmd_categoria        float64
aeronave_assentos             float64
aeronave_ano_fabricacao       float64
aeronave_pais_fabricante       object
aeronave_pais_registro         object
aeronave_registro_categoria    object
aeronave_registro_segmento     object
aeronave_voo_origem            object
aeronave_voo_destino           object
aeronave_fase_operacao         object
aeronave_tipo_operacao         object
aeronave_nivel_dano            object
aeronave_fatalidades_total    float64
dtype: object
```

É definida uma lista 'colunas_para_str_aeronave', que contém os nomes das colunas que serão convertidas para o tipo 'str', o método usado é astype(str) para converter as colunas da lista ‘colunas_para_str_aeronave’ para o tipo "str". Isso significa que os valores em cada coluna da lista serão tratados como strings, mesmo que os valores originais das colunas fossem de outros tipos (números inteiros ou floats), após essa operação, eles serão interpretados como strings.

In [18]:

```
#aeronave

# Listas das Colunas para conversão para inteiro
colunas_para_int_aeronave = ['aeronave_pmd','aeronave_pmd_categoria','aeronave_assentos',
                             'aeronave_ano_fabricacao','aeronave_fatalidades_total']


# Convertendo
df_aeronave[colunas_para_int_aeronave] = df_aeronave[colunas_para_int_aeronave].astype(int)

print(df_aeronave.dtypes)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-b180c056fecb> in <module>
      7
      8 # Convertendo
----> 9 df_aeronave[colunas_para_int_aeronave] = df_aeronave[colunas_para_int_aeronave].astype(int)
     10
     11 print(df_aeronave.dtypes)

~\anaconda3\lib\site-packages\pandas\core\generic.py in astype(self, dtype, copy, errors)
   5875         else:
   5876             # else, only a single dtype is given
-> 5877             new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   5878             return self._constructor(new_data).__finalize__(self, method="astype")
   5879

~\anaconda3\lib\site-packages\pandas\core\internals\managers.py in astype(self, dtype, copy, errors)
    629         self, dtype, copy: bool = False, errors: str = "raise"
    630     ) -> "BlockManager":
--> 631         return self.apply("astype", dtype=dtype, copy=copy, errors=errors)
    632
    633     def convert(

~\anaconda3\lib\site-packages\pandas\core\internals\managers.py in apply(self, f, align_keys, ignore_failures, **kwar
gs)
    425                     applied = b.apply(f, **kwargs)
    426                 else:
--> 427                     applied = getattr(b, f)(**kwargs)
    428             except (TypeError, NotImplementedError):
    429                 if not ignore_failures:

~\anaconda3\lib\site-packages\pandas\core\internals\blocks.py in astype(self, dtype, copy, errors)
    671             vals1d = values.ravel()
    672             try:
--> 673                 values = astype_nansafe(vals1d, dtype, copy=True)
    674             except (ValueError, TypeError):
    675                 # e.g. astype_nansafe can fail on object-dtype of strings

~\anaconda3\lib\site-packages\pandas\core\dtypes\cast.py in astype_nansafe(arr, dtype, copy, skipna)
   1066
   1067         if not np.isfinite(arr).all():
```

```
-> 1068             raise ValueError("Cannot convert non-finite values (NA or inf) to integer")
   1069
   1070     elif is_object_dtype(arr):

ValueError: Cannot convert non-finite values (NA or inf) to integer
```

Esse erro ocorre porque a conversão de valores como NaN ou infinito para inteiros não é permitida no Pandas. Para resolver esse problema, será usado o método pd.to_numeric() com o argumento errors='coerce' para converter as colunas para valores numéricos, tratando os valores não finitos como NaN (Not-a-Number), e, em seguida, utilizar o método fillna() para substituir os NaNs por um valor adequado, como 0.

In [19]:

```
#aeronave

# Listas das Colunas para conversão para inteiro
colunas_para_int_aeronave = ['aeronave_pmd','aeronave_pmd_categoria','aeronave_assentos',
                             'aeronave_ano_fabricacao','aeronave_fatalidades_total']


# Convertendo as colunas para valores numéricos e tratando os valores não finitos como NaN
df_aeronave[colunas_para_int_aeronave] = df_aeronave[colunas_para_int_aeronave].apply(pd.to_numeric, errors='coerc

# Substituindo os NaNs por 0
df_aeronave[colunas_para_int_aeronave] = df_aeronave[colunas_para_int_aeronave].fillna(0).astype(int)


print(df_aeronave.dtypes)
```

```
codigo_ocorrencia2             int64
aeronave_matricula            object
aeronave_operador_categoria   object
aeronave_tipo_veiculo         object
aeronave_fabricante           object
aeronave_modelo               object
aeronave_tipo_icao            object
aeronave_motor_tipo           object
aeronave_motor_quantidade     object
aeronave_pmd                   int32
aeronave_pmd_categoria         int32
aeronave_assentos              int32
aeronave_ano_fabricacao        int32
aeronave_pais_fabricante      object
aeronave_pais_registro        object
aeronave_registro_categoria   object
aeronave_registro_segmento    object
aeronave_voo_origem           object
aeronave_voo_destino          object
aeronave_fase_operacao        object
aeronave_tipo_operacao        object
aeronave_nivel_dano           object
aeronave_fatalidades_total     int32
dtype: object
```

2- Verificação de Duplicadas

Ocorrencia

In [20]:
```
# Consulta para verificar linhas duplicadas da tabela ocorrencia

duplicados = df_ocorrencia.duplicated()
print("Lista da colunas duplicadas da tabela ocorrencia")
print(duplicados)

duplicados1 = duplicados.sum()
print(f"A Quantidade de Linhas duplicadas da tabela ocorrencia é {duplicados1}")
```

```
Lista da colunas duplicadas da tabela ocorrencia
0       False
1       False
2       False
3       False
4       False
5       False
6       False
7       False
8       False
9       False
10      False
11      False
12      False
13      False
14      False
15      False
16      False
17      False
```

'duplicados = df_ocorrencia.duplicated()': Nesta linha, a variável duplicados recebe uma série booleana que indica quais linhas do DataFrame df_ocorrencia são duplicadas. Cada elemento da série é True se a linha correspondente no DataFrame for uma duplicata e False caso contrário.

'print("Lista da colunas duplicadas da tabela ocorrencia")': Essa linha simplesmente imprime uma mensagem informando que a lista de colunas duplicadas será exibida.

'print(duplicados)': Aqui, a série booleana duplicados é impressa na saída. Ela mostrará uma lista de True e False correspondendo às linhas duplicadas e não duplicadas, respectivamente.

'duplicados1 = duplicados.sum()': Nesta linha, a variável duplicados1 recebe o resultado da soma dos valores booleanos da série duplicados. Como True é considerado como 1 e False como 0, essa soma nos dará a quantidade total de linhas duplicadas no DataFrame df_ocorrencia.

'print(f"A Quantidade de Linhas duplicadas da tabela ocorrencia é {duplicados1}")': Finalmente, a quantidade de linhas duplicadas é impressa usando uma f-string. O valor de duplicados1 é substituído na posição {duplicados1} da string durante a impressão.

Ocorrencia_tipo

In [21]:
```
# Consulta para verificar linhas duplicadas da tabela ocorrencia_tipo

duplicados_ocorrencia_tipo = df_ocorrencia_tipo.duplicated()
print("Lista da colunas duplicadas da tabela ocorrencia_tipo")
print(duplicados_ocorrencia_tipo)

duplicados_ocorrencia_tipo1 = duplicados_ocorrencia_tipo.sum()
print(f"A Quantidade de Linhas duplicadas da tabela ocorrencia_tipo é {duplicados_ocorrencia_tipo1}")
```

```
Lista da colunas duplicadas da tabela ocorrencia_tipo
0       False
1       False
2       False
3       False
4       False
5       False
6       False
7       False
8       False
9       False
10      False
11      False
12      False
13      False
14      False
15      False
16      False
17      False
```

'duplicados_ocorrencia_tipo = df_ocorrencia_tipo.duplicated()': Nesta linha, a variável duplicados_ocorrencia_tipo recebe uma série booleana que indica quais linhas do DataFrame df_ocorrencia_tipo são duplicadas. Cada elemento da série é True se a linha correspondente no DataFrame for uma duplicata e False caso contrário.

'print("Lista da colunas duplicadas da tabela ocorrencia_tipo")': Essa linha simplesmente imprime uma mensagem informando que a lista de colunas duplicadas da tabela ocorrencia_tipo será exibida.

'print(duplicados_ocorrencia_tipo)': Aqui, a série booleana duplicados_ocorrencia_tipo é impressa na saída. Ela mostrará uma lista de True e False correspondendo às linhas duplicadas e não duplicadas, respectivamente.

'duplicados_ocorrencia_tipo1 = duplicados_ocorrencia_tipo.sum()': Nesta linha, a variável duplicados_ocorrencia_tipo1 recebe o resultado da soma dos valores booleanos da série duplicados_ocorrencia_tipo. Isso irá resultar na quantidade total de linhas duplicadas no DataFrame df_ocorrencia_tipo. A soma conta quantos elementos da série são True (ou seja, são duplicados) e retorna o total.

'print(f"A Quantidade de Linhas duplicadas da tabela ocorrencia_tipo é {duplicados_ocorrencia_tipo1}")': Finalmente, a quantidade de linhas duplicadas é impressa usando uma f-string. O valor de duplicados_ocorrencia_tipo1 é substituído na posição {duplicados_ocorrencia_tipo1} da string durante a impressão.

Recomendacao

In [22]:
```
# Consulta para verificar linhas duplicadas da tabela recomendacao

duplicados_recomendacao = df_recomendacao.duplicated()
print("Lista da colunas duplicadas da tabela recomendacao")
print(duplicados_recomendacao)

duplicados_recomendacao1 = duplicados_recomendacao.sum()
print(f"A Quantidade de Linhas duplicadas da tabela recomendacao é {duplicados_recomendacao1}")
```

```
Lista da colunas duplicadas da tabela recomendacao
0       False
1       False
2       False
3       False
4       False
5       False
6       False
7       False
8       False
9       False
10      False
11      False
12      False
13      False
14      False
15      False
16      False
17      False
```

'duplicados_recomendacao = df_recomendacao.duplicated()': Nesta linha, a variável duplicados_recomendacao recebe uma série booleana que indica quais linhas do DataFrame df_recomendacao são duplicadas. Cada elemento da série é True se a linha correspondente no DataFrame for uma duplicata e False caso contrário.

'print("Lista da colunas duplicadas da tabela recomendacao")': Essa linha simplesmente imprime uma mensagem informando que a lista de colunas duplicadas da tabela recomendacao será exibida.

'print(duplicados_recomendacao)': Aqui, a série booleana duplicados_recomendacao é impressa na saída. Ela mostrará uma lista de True e False correspondendo às linhas duplicadas e não duplicadas, respectivamente.

'duplicados_recomendacao1 = duplicados_recomendacao.sum()': Nesta linha, a variável duplicados_recomendacao1 recebe o resultado da soma dos valores booleanos da série duplicados_recomendacao. Isso irá resultar na quantidade total de linhas duplicadas no DataFrame df_recomendacao. A soma conta quantos elementos da série são True (ou seja, são duplicados) e retorna o total.

'print(f"A Quantidade de Linhas duplicadas da tabela recomendacao é {duplicados_recomendacao1}")': Finalmente, a quantidade de linhas duplicadas é impressa usando uma f-string. O valor de duplicados_recomendacao1 é substituído na posição {duplicados_recomendacao1} da

Fator_contribuinte

In [23]:
```
# Consulta para verificar linhas duplicadas da tabela fator_contribuinte

duplicados_fator_contribuinte = df_fator_contribuinte.duplicated()
print("Lista da colunas duplicadas da tabela fator_contribuinte")
print(duplicados_fator_contribuinte)

duplicados_fator_contribuinte1 = duplicados_fator_contribuinte.sum()
print(f"A Quantidade de Linhas duplicadas da tabela fator_contribuinte é {duplicados_fator_contribuinte1}")
```

```
Lista da colunas duplicadas da tabela fator_contribuinte
0       False
1       False
2       False
3       False
4       False
5       False
6       False
7       False
8       False
9       False
10      False
11      False
12      False
13      False
14      False
15      False
16      False
17      False
```

'duplicados_fator_contribuinte = df_fator_contribuinte.duplicated()': Nesta linha, a variável duplicados_fator_contribuinte recebe uma série booleana que indica quais linhas do DataFrame df_fator_contribuinte são duplicadas. Cada elemento da série é True se a linha correspondente no DataFrame for uma duplicata e False caso contrário.

'print("Lista da colunas duplicadas da tabela fator_contribuinte")': Essa linha simplesmente imprime uma mensagem informando que a lista de colunas duplicadas da tabela fator_contribuinte será exibida.

'print(duplicados_fator_contribuinte)': Aqui, a série booleana duplicados_fator_contribuinte é impressa na saída. Ela mostrará uma lista de True e False correspondendo às linhas duplicadas e não duplicadas, respectivamente.

'duplicados_fator_contribuinte1 = duplicados_fator_contribuinte.sum()': Nesta linha, a variável duplicados_fator_contribuinte1 recebe o resultado da soma dos valores booleanos da série duplicados_fator_contribuinte. Isso irá resultar na quantidade total de linhas duplicadas no DataFrame df_fator_contribuinte. A soma conta quantos elementos da série são True (ou seja, são duplicados) e retorna o total.

'print(f"A Quantidade de Linhas duplicadas da tabela fator_contribuinte é {duplicados_fator_contribuinte1}")': Finalmente, a quantidade de linhas duplicadas é impressa usando uma f-string. O valor de duplicados_fator_contribuinte1 é substituído na posição {duplicados_fator_contribuinte1} da string durante a impressão.

Aeronave

In [24]:

```
# Consulta para verificar linhas duplicadas da tabela aeronave

duplicados_aeronave = df_aeronave.duplicated()
print("Lista da colunas duplicadas da tabela aeronave")
print(duplicados_aeronave)

duplicados_aeronave1 = duplicados_aeronave.sum()
print(f"A Quantidade de Linhas duplicadas da tabela aeronave é {duplicados_aeronave1}")
```

```
Lista da colunas duplicadas da tabela aeronave
0       False
1       False
2       False
3       False
4       False
5       False
6       False
7       False
8       False
9       False
10      False
11      False
12      False
13      False
14      False
15      False
16      False
17      False
```

duplicados_aeronave = df_aeronave.duplicated(): Nesta linha, a variável duplicados_aeronave recebe uma série booleana que indica quais linhas do DataFrame df_aeronave são duplicadas. Cada elemento da série é True se a linha correspondente no DataFrame for uma duplicata e False caso contrário.

print("Lista da colunas duplicadas da tabela aeronave"): Essa linha simplesmente imprime uma mensagem informando que a lista de colunas duplicadas da tabela aeronave será exibida.

print(duplicados_aeronave): Aqui, a série booleana duplicados_aeronave é impressa na saída. Ela mostrará uma lista de True e False correspondendo às linhas duplicadas e não duplicadas, respectivamente.

duplicados_aeronave1 = duplicados_aeronave.sum(): Nesta linha, a variável duplicados_aeronave1 recebe o resultado da soma dos valores booleanos da série duplicados_aeronave. Isso irá resultar na quantidade total de linhas duplicadas no DataFrame df_aeronave. A soma conta quantos elementos da série são True (ou seja, são duplicados) e retorna o total.

print(f"A Quantidade de Linhas duplicadas da tabela aeronave é {duplicados_aeronave1}"): Finalmente, a quantidade de linhas duplicadas é impressa usando uma f-string. O valor de duplicados_aeronave1 é substituído na posição {duplicados_aeronave1} da string durante a

3 – Verificação de Linhas em Branco:

Nessa etapa, verificamos a quantidade de linhas em branco

Ocorrencia

In [25]:

```
def linha_em_branco(linha):
    colunas = linha.split(";")  # Divide a linha em colunas usando ";" como separador
    for coluna in colunas:
        if coluna.strip():  # Verifica se a coluna não está em branco após remover espaços em branco
            return False
    return True



for linha in df_ocorrencia:
    if linha_em_branco(linha):
        print("Linha em branco:", linha)
    else:
        print("Linha não em branco:", linha)

```

```
Linha não em branco: codigo_ocorrencia
Linha não em branco: codigo_ocorrencia1
Linha não em branco: codigo_ocorrencia2
Linha não em branco: codigo_ocorrencia3
Linha não em branco: codigo_ocorrencia4
Linha não em branco: ocorrencia_classificacao
Linha não em branco: ocorrencia_latitude
Linha não em branco: ocorrencia_longitude
Linha não em branco: ocorrencia_cidade
Linha não em branco: ocorrencia_uf
Linha não em branco: ocorrencia_pais
Linha não em branco: ocorrencia_aerodromo
Linha não em branco: ocorrencia_dia
Linha não em branco: ocorrencia_hora
Linha não em branco: investigacao_aeronave_liberada
Linha não em branco: investigacao_status
Linha não em branco: divulgacao_relatorio_numero
Linha não em branco: divulgacao_relatorio_publicado
Linha não em branco: divulgacao_dia_publicacao
Linha não em branco: total_recomendacoes
Linha não em branco: total_aeronaves_envolvidas
Linha não em branco: ocorrencia_saida_pista
```

Ocorrencia_tipo

In [26]:

```
def linha_em_branco(linha):
    colunas = linha.split(";")  # Divide a linha em colunas usando ";" como separador
    for coluna in colunas:
        if coluna.strip():  # Verifica se a coluna não está em branco após remover espaços em branco
            return False
    return True



for linha in df_ocorrencia_tipo:
    if linha_em_branco(linha):
        print("Linha em branco:", linha)
    else:
        print("Linha não em branco:", linha)

```

```
Linha não em branco: codigo_ocorrencia1
Linha não em branco: ocorrencia_tipo
Linha não em branco: ocorrencia_tipo_categoria
Linha não em branco: taxonomia_tipo_icao
```

Recomendacao

In [27]:

```python
def linha_em_branco(linha):
    colunas = linha.split(";")  # Divide a linha em colunas usando ";" como separador
    for coluna in colunas:
        if coluna.strip():  # Verifica se a coluna não está em branco após remover espaços em branco
            return False
    return True



for linha in df_recomendacao:
    if linha_em_branco(linha):
        print("Linha em branco:", linha)
    else:
        print("Linha não em branco:", linha)

```

```
Linha não em branco: codigo_ocorrencia4
Linha não em branco: recomendacao_numero
Linha não em branco: recomendacao_dia_assinatura
Linha não em branco: recomendacao_dia_encaminhamento
Linha não em branco: recomendacao_dia_feedback
Linha não em branco: recomendacao_conteudo
Linha não em branco: recomendacao_status
Linha não em branco: recomendacao_destinatario_sigla
Linha não em branco: recomendacao_destinatario
```

Fator_contribuinte

In [28]:

```python
def linha_em_branco(linha):
    colunas = linha.split(";")  # Divide a linha em colunas usando ";" como separador
    for coluna in colunas:
        if coluna.strip():  # Verifica se a coluna não está em branco após remover espaços em branco
            return False
    return True



for linha in df_fator_contribuinte:
    if linha_em_branco(linha):
        print("Linha em branco:", linha)
    else:
        print("Linha não em branco:", linha)
```

```
Linha não em branco: codigo_ocorrencia3
Linha não em branco: fator_nome
Linha não em branco: fator_aspecto
Linha não em branco: fator_condicionante
Linha não em branco: fator_area
```

Aeronave

In [29]:

```python
def linha_em_branco(linha):
    colunas = linha.split(";")  # Divide a linha em colunas usando ";" como separador
    for coluna in colunas:
        if coluna.strip():  # Verifica se a coluna não está em branco após remover espaços em branco
            return False
    return True



for linha in df_aeronave:
    if linha_em_branco(linha):
        print("Linha em branco:", linha)
    else:
        print("Linha não em branco:", linha)
```

```
Linha não em branco: codigo_ocorrencia2
Linha não em branco: aeronave_matricula
Linha não em branco: aeronave_operador_categoria
Linha não em branco: aeronave_tipo_veiculo
Linha não em branco: aeronave_fabricante
Linha não em branco: aeronave_modelo
Linha não em branco: aeronave_tipo_icao
Linha não em branco: aeronave_motor_tipo
Linha não em branco: aeronave_motor_quantidade
Linha não em branco: aeronave_pmd
Linha não em branco: aeronave_pmd_categoria
Linha não em branco: aeronave_assentos
Linha não em branco: aeronave_ano_fabricacao
Linha não em branco: aeronave_pais_fabricante
Linha não em branco: aeronave_pais_registro
Linha não em branco: aeronave_registro_categoria
Linha não em branco: aeronave_registro_segmento
Linha não em branco: aeronave_voo_origem
Linha não em branco: aeronave_voo_destino
Linha não em branco: aeronave_fase_operacao
Linha não em branco: aeronave_tipo_operacao
Linha não em branco: aeronave_nivel_dano
Linha não em branco: aeronave_fatalidades_total
```

4 – Validação de Dados

O código faz a verificação de linhas em branco em uma base de dados representada pela lista. Aqui estão os detalhes do que o código faz:

1 - A função linha_em_branco é definida com um parâmetro linha. Essa função verifica se todas as colunas de uma linha estão em branco.

2 - A linha é dividida em colunas usando o ponto e vírgula (;) como separador.

3 - Um loop for percorre cada coluna nas colunas divididas.

4 - Para cada coluna, o método .strip() é usado para remover os espaços em branco do início e do final.

5 - A instrução if coluna.strip(): verifica se a coluna não está em branco após a remoção dos espaços. Se a coluna tiver algum conteúdo, o fluxo entra na instrução if e a função retorna False, indicando que a linha não está em branco. Se todas as colunas estiverem em branco, a função retorna True, indicando que a linha está em branco.

No bloco de código principal:

1 - Um loop for percorre cada linha na lista.

2 - Para cada linha, a função linha_em_branco é chamada para verificar se a linha está em branco.

3 - Se a função linha_em_branco retornar True, a linha é considerada em branco, e a mensagem "Linha em branco" junto com a linha é impressa.

4 - Caso contrário, a mensagem "Linha não em branco" junto com a linha é impressa.

O código verifica cada linha na base de dados e informa se ela está em branco ou não, com base nas colunas vazias ou não vazias.

A validação de dados é um processo que envolve a verificação e a confirmação de que os dados coletados, armazenados ou processados estão corretos, consistentes e se encontram dentro de limites aceitáveis. Para alcançar esse objetivo, são estabelecidos critérios pré-definidos, que funcionam como regras ou padrões a serem seguidos. Esses critérios podem envolver verificações de formato, faixas de valores, integridade referencial e outros aspectos relevantes, dependendo do contexto em que os dados são utilizados.

Ocorrencia

In [30]:

```python
# Lista de classificações aceitáveis
classificacoes_aceitaveis = ["INCIDENTE GRAVE", "INCIDENTE", "ACIDENTE"]

# Verificação dos valores da coluna
valores_invalidos = df_ocorrencia[~df_ocorrencia["ocorrencia_classificacao"].isin(classificacoes_aceitaveis)]

if not valores_invalidos.empty:
    mensagens = []
    for index, row in valores_invalidos.iterrows():
        mensagem = f"Linha {index + 1}: Classificação inválida '{row['ocorrencia_classificacao']}'"
        mensagens.append(mensagem)

    print("Valores inválidos encontrados na coluna 'ocorrencia_classificacao':")
    for mensagem in mensagens:
        print(mensagem)
else:
    print("Todos os valores são válidos na coluna 'ocorrencia_classificacao'.")

```

```
Todos os valores são válidos na coluna 'ocorrencia_classificacao'.
```

O código executa uma verificação nos valores da coluna "ocorrencia_classificacao" e identifica se algum valor não está presente na lista de classificações aceitáveis.

Passo a Passo:

1 - 'classificacoes_aceitaveis:' Uma lista contendo as classificações aceitáveis ("INCIDENTE GRAVE", "INCIDENTE", "ACIDENTE").

2 - 'valores_invalidos:' Filtra o DataFrame df_ocorrencia para identificar as linhas onde o valor na coluna "ocorrencia_classificacao" não está na lista de classificações aceitáveis. Ele utiliza o método .isin() para verificar isso.

3 - 'if not valores_invalidos.empty:' Verifica se há valores inválidos identificados na coluna. Se o DataFrame 'valores_invalidos' não estiver vazio (ou seja, se valores inválidos foram encontrados), o bloco de código dentro deste if será executado.

4 - Loop sobre os valores inválidos: Para cada linha no DataFrame valores_invalidos, ele cria uma mensagem indicando a linha correspondente e o valor inválido encontrado na coluna "ocorrencia_classificacao". Essas mensagens são armazenadas na lista mensagens.

5 - Exibe mensagens de valores inválidos: Após o loop, o código exibe as mensagens identificando as linhas onde foram encontrados valores inválidos na coluna "ocorrencia_classificacao".

6 - Caso contrário: Se nenhum valor inválido for encontrado na coluna, o código exibe uma mensagem indicando que todos os valores são válidos na coluna "ocorrencia_classificacao".

Recomendacao

In [31]:
```
# Verificação da correspondência entre os campos
indices_sem_sigla = df_recomendacao[df_recomendacao["recomendacao_destinatario_sigla"] == ""].index
indices_com_sigla = df_recomendacao[df_recomendacao["recomendacao_destinatario_sigla"] != ""].index

numeros_sem_sigla = df_recomendacao.loc[indices_sem_sigla, "recomendacao_numero"]
numeros_com_sigla = df_recomendacao.loc[indices_com_sigla, "recomendacao_numero"]

numeros_faltantes = set(numeros_sem_sigla) - set(numeros_com_sigla)

if numeros_faltantes:
    print("Números de recomendação sem correspondente em Recomendacao_destinatario_sigla:")
    for numero in numeros_faltantes:
        print(f"Número: {numero}")
else:
    print("Todos os números de recomendação têm correspondente em Recomendacao_destinatario_sigla.")

```

```
Todos os números de recomendação têm correspondente em Recomendacao_destinatario_sigla.
```

O código executa uma verificação nos dados da tabela "recomendacao" para verificar se cada valor na coluna "recomendacao_numero" tem um valor correspondente (ou seja, não vazio) na coluna "recomendacao_destinatario_sigla".

Passo a Passo:

1 - 'indices_sem_sigla:' Filtra os índices das linhas onde a coluna "recomendacao_destinatario_sigla" está vazia.

2 - 'indices_com_sigla:' Filtra os índices das linhas onde a coluna "recomendacao_destinatario_sigla" não está vazia.

3 - 'numeros_sem_sigla:' Obtém os valores da coluna "recomendacao_numero" correspondentes aos índices encontrados em indices_sem_sigla.

4 - 'numeros_com_sigla:' Obtém os valores da coluna "recomendacao_numero" correspondentes aos índices encontrados em indices_com_sigla.

5 - 'numeros_faltantes:' Calcula a diferença de conjuntos entre numeros_sem_sigla e numeros_com_sigla, ou seja, encontra os números de recomendação que não têm um correspondente em "recomendacao_destinatario_sigla".

6 - Verificação e saída: O código verifica se existem números de recomendação sem correspondentes em "recomendacao_destinatario_sigla".

Fator_contribuinte

In [32]:
```
# Lista de classificações aceitáveis
fator_area_aceitavel = ["FATOR HUMANO", "FATOR OPERACIONAL", "OUTRO", "FATOR MATERIAL"]

# Verificação dos valores da coluna
valores_invalidos1 = df_fator_contribuinte[~df_fator_contribuinte["fator_area"].isin(fator_area_aceitavel)]

if not valores_invalidos1.empty:
    mensagens = []
    for index, row in valores_invalidos.iterrows():
        mensagem = f"Linha {index + 1}: Classificação inválida '{row['fator_area_aceitavel']}'"
        mensagens.append(mensagem)

    print("Valores inválidos encontrados na coluna 'fator_area_aceitavel':")
    for mensagem in mensagens:
        print(mensagem)
else:
    print("Todos os valores são válidos na coluna 'fator_area_aceitavel'.")
```

```
Todos os valores são válidos na coluna 'fator_area_aceitavel'.
```

O código verifica se os valores na coluna "fator_area" da tabela "df_fator_contribuinte" estão dentro da lista de classificações aceitáveis definida em fator_area_aceitavel.

Passo a Passo:

1 - 'fator_area_aceitavel': É uma lista contendo as classificações aceitáveis ("FATOR HUMANO", "FATOR OPERACIONAL", "OUTRO", "FATOR MATERIAL").

2 - 'valores_invalidos': Filtra o DataFrame df_fator_contribuinte para identificar as linhas onde o valor na coluna "fator_area" não está na lista de classificações aceitáveis. Ele utiliza o método .isin() para verificar isso.

3 - 'if not valores_invalidos.empty:' Verifica se há valores inválidos identificados na coluna. Se o DataFrame valores_invalidos não estiver vazio (ou seja, se valores inválidos foram encontrados), o bloco de código dentro deste if será executado.

4 - Loop sobre os valores inválidos1: Para cada linha no DataFrame valores_invalidos, ele cria uma mensagem indicando a linha correspondente e a classificação inválida encontrada na coluna "fator_area". Essas mensagens são armazenadas na lista mensagens.

5 - Exibe mensagens de valores inválidos: Após o loop, o código exibe as mensagens identificando as linhas onde foram encontrados valores inválidos na coluna "fator_area".

6 - Caso contrário: Se nenhum valor inválido for encontrado na coluna, o código informa que todos os valores são válidos na coluna "fator_area".

5 - Substituição de Valores:

A substituição possibilita a adaptação dos registros às novas regras de negócios, a correção de imprecisões, a incorporação de informações complementares e a garantia de que todos os componentes de um sistema que utilizam esses dados possam operar com informações consistentes e atualizadas.

ocorrencia

In [33]:

```python
def substituir_acentos(texto):
    substituicoes = {
        "Á": "A",
        "À": "A",
        "Â": "A",
        "Ã": "A",
        "É": "E",
        "Ê": "E",
        "Í": "I",
        "Ó": "O",
        "Ô": "O",
        "Õ": "O",
        "Ú": "U",
        "Ç": "C"
    }
    for acento, original in substituicoes.items():
        texto = texto.replace(acento, original)
    return texto


# Aplicar a função substituir_acentos à coluna ocorrencia_cidade
df_ocorrencia["ocorrencia_cidade"] = df_ocorrencia["ocorrencia_cidade"].apply(substituir_acentos)

print(df_ocorrencia['ocorrencia_cidade'])
```

```
0                          SAO PAULO
1                            VITORIA
2                           AMERICANA
3                      BELO HORIZONTE
4                             MANAUS
5                        PORTO ALEGRE
6                             LOBATO
7                   NOVA BANDEIRANTES
8                            NAVIRAI
9                           LINHARES
10                           GOIANIA
11                         UBERLANDIA
12                          BOA VISTA
13                          ARACATUBA
14                  BRAGANCA PAULISTA
15                           LONTRAS
16                          TERESINA
17                          QUERENCIA
18                          GUARULHOS
19                  ENGENHEIRO CALDAS
```

Foi criada uma função chamada substituir_acentos que realiza a substituição de caracteres acentuados por suas versões não acentuadas em uma string. Essa função é definida com um dicionário de substituições contendo pares de acentos e suas correspondentes letras não acentuadas.

1 -Foi definido a função 'substituir_acentos', que toma um parâmetro texto.

2 - Dentro da função, possui um dicionário chamado 'substituicoes' que mapeia caracteres acentuados para suas versões não acentuadas.

3 - Usando um loop for, percorre cada par chave-valor no dicionário de 'substituicoes'.

4 - Dentro do loop, é usado o método '.replace()' para substituir cada ocorrência do caractere acentuado pelo caractere não acentuado correspondente na string 'texto'.

5 - Finalmente, a função retorna a string 'texto' após todas as substituições terem sido feitas.

6 - Fora da função, é aplicado a função 'substituir_acentos' à coluna "ocorrencia_cidade" do DataFrame 'df_ocorrencia' usando o método '.apply()'. Isso significa que cada valor nessa coluna será processado pela função 'substituir_acentos'.

7 - Por fim, é impresso a coluna "ocorrencia_cidade" do DataFrame 'df_ocorrencia' para ver os resultados da substituição de acentos.

No geral, o código realiza a substituição de caracteres acentuados por suas versões não acentuadas na coluna "ocorrencia_cidade" do DataFrame df_ocorrencia, tornando os dados mais padronizados e facilitando a busca e comparação de informações.

In [34]:
```
# Substituir "não" por "nao" na coluna divulgacao_relatorio_publicado
df_ocorrencia["divulgacao_relatorio_publicado"] = df_ocorrencia["divulgacao_relatorio_publicado"].str.replace("NÃO
df_ocorrencia["ocorrencia_saida_pista"] = df_ocorrencia["ocorrencia_saida_pista"].str.replace("NÃO", "NAO")
df_ocorrencia["investigacao_aeronave_liberada"] = df_ocorrencia["investigacao_aeronave_liberada"].str.replace("NÃO


# Exibir a tabela atualizada
print(df_ocorrencia[["divulgacao_relatorio_publicado",'ocorrencia_saida_pista',"investigacao_aeronave_liberada"]])

```

```
    divulgacao_relatorio_publicado ocorrencia_saida_pista  \
0                              NAO                    SIM
1                              NAO                    NAO
2                              NAO                    SIM
3                              NAO                    NAO
4                              NAO                    NAO
5                              NAO                    NAO
6                              NAO                    NAO
7                              NAO                    SIM
8                              NAO                    SIM
9                              NAO                    NAO
10                             NAO                    SIM
11                             NAO                    NAO
12                             NAO                    NAO
13                             NAO                    NAO
14                             NAO                    NAO
15                             NAO                    NAO
16                             NAO                    NAO
17                             NAO                    NAO
```

No código está sendo feito uma substituição específica de strings em duas colunas do DataFrame df_ocorrencia.

Passo a Passo:

1 - 'df_ocorrencia["divulgacao_relatorio_publicado"] = df_ocorrencia["divulgacao_relatorio_publicado"].str.replace("NÃO", "NAO"):' Está substituindo todas as ocorrências da string "NÃO" pela string "NAO" na coluna "divulgacao_relatorio_publicado". O método .str.replace() é usado para realizar essa substituição de forma eficiente em todas as células da coluna.

2 - 'df_ocorrencia["ocorrencia_saida_pista"] = df_ocorrencia["ocorrencia_saida_pista"].str.replace("NÃO", "NAO"):' Da mesma forma, está substituindo todas as ocorrências da string "NÃO" pela string "NAO" na coluna "ocorrencia_saida_pista".

3 - 'df_ocorrencia["investigacao_aeronave_liberada"] = df_ocorrencia["investigacao_aeronave_liberada"].str.replace("NÃO", "NAO")' Da mesma forma, está substituindo todas as ocorrências da string "NÃO" pela string "NAO" na coluna "investigacao_aeronave_liberada".

4 - 'print(df_ocorrencia[["divulgacao_relatorio_publicado",'ocorrencia_saida_pista',"investigacao_aeronave_liberada"]])': Está imprimindo as colunas "divulgacao_relatorio_publicado", "ocorrencia_saida_pista" e "investigacao_aeronave_liberada" do DataFrame df_ocorrencia após a realização das substituições. Isso exibirá a tabela atualizada com as substituições feitas.

Em resumo, o código está buscando por ocorrências da string "NÃO" nas colunas "divulgacao_relatorio_publicado", "ocorrencia_saida_pista" e "investigacao_aeronave_liberada" do DataFrame df_ocorrencia e substituindo essas ocorrências pela string "NAO". Isso é útil para padronizar os dados e facilitar futuras análises ou processamentos que envolvam essas colunas.

In [35]:

```
# Substituir "***" por "" na coluna divulgacao_relatorio_publicado
df_ocorrencia["ocorrencia_uf"] = df_ocorrencia["ocorrencia_uf"].str.replace("***", " ")
df_ocorrencia["ocorrencia_aerodromo"] = df_ocorrencia["ocorrencia_aerodromo"].str.replace("***", " ")
df_ocorrencia["investigacao_aeronave_liberada"] = df_ocorrencia["investigacao_aeronave_liberada"].str.replace("***
df_ocorrencia["divulgacao_relatorio_numero"] = df_ocorrencia["divulgacao_relatorio_numero"].str.replace("***", ' '


# Exibir a tabela atualizada
print(df_ocorrencia[["ocorrencia_uf","ocorrencia_aerodromo","investigacao_aeronave_liberada","divulgacao_relatorio
```

```
<ipython-input-35-4043b07920f8>:2: FutureWarning: The default value of regex will change from True to False in a futu
re version.
  df_ocorrencia["ocorrencia_uf"] = df_ocorrencia["ocorrencia_uf"].str.replace("***", " ")
```

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-35-4043b07920f8> in <module>
      1 # Substituir "***" por "" na coluna divulgacao_relatorio_publicado
----> 2 df_ocorrencia["ocorrencia_uf"] = df_ocorrencia["ocorrencia_uf"].str.replace("***", " ")
      3 df_ocorrencia["ocorrencia_aerodromo"] = df_ocorrencia["ocorrencia_aerodromo"].str.replace("***", " ")
      4 df_ocorrencia["investigacao_aeronave_liberada"] = df_ocorrencia["investigacao_aeronave_liberada"].str.replace
("***", " ")
      5 df_ocorrencia["divulgacao_relatorio_numero"] = df_ocorrencia["divulgacao_relatorio_numero"].str.replace("**
*", ' ')

~\anaconda3\lib\site-packages\pandas\core\strings\accessor.py in wrapper(self, *args, **kwargs)
     99                 )
    100                 raise TypeError(msg)
--> 101             return func(self, *args, **kwargs)
    102
    103         wrapper.__name__ = func_name

~\anaconda3\lib\site-packages\pandas\core\strings\accessor.py in replace(self, pat, repl, n, case, flags, regex)
   1309                 warnings.warn(msg, FutureWarning, stacklevel=3)
   1310             regex = True
-> 1311         result = self._data.array._str_replace(
   1312             pat, repl, n=n, case=case, flags=flags, regex=regex
   1313         )

~\anaconda3\lib\site-packages\pandas\core\strings\object_array.py in _str_replace(self, pat, repl, n, case, flags, re
gex)
    158             if is_compiled_re or len(pat) > 1 or flags or callable(repl):
    159                 n = n if n >= 0 else 0
--> 160                 compiled = re.compile(pat, flags=flags)
    161                 f = lambda x: compiled.sub(repl=repl, string=x, count=n)
    162             else:

~\anaconda3\lib\re.py in compile(pattern, flags)
    250 def compile(pattern, flags=0):
    251     "Compile a regular expression pattern, returning a Pattern object."
--> 252     return _compile(pattern, flags)
    253
    254 def purge():

~\anaconda3\lib\re.py in _compile(pattern, flags)
```

```
    302     if not sre_compile.isstring(pattern):
    303         raise TypeError("first argument must be string or compiled pattern")
--> 304     p = sre_compile.compile(pattern, flags)
    305     if not (flags & DEBUG):
    306         if len(_cache) >= _MAXCACHE:

~\anaconda3\lib\sre_compile.py in compile(p, flags)
    762     if isstring(p):
    763         pattern = p
--> 764         p = sre_parse.parse(p, flags)
    765     else:
    766         pattern = None

~\anaconda3\lib\sre_parse.py in parse(str, flags, state)
    946
    947     try:
--> 948         p = _parse_sub(source, state, flags & SRE_FLAG_VERBOSE, 0)
    949     except Verbose:
    950         # the VERBOSE flag was switched on inside the pattern.  to be

~\anaconda3\lib\sre_parse.py in _parse_sub(source, state, verbose, nested)
    441     start = source.tell()
    442     while True:
--> 443         itemsappend(_parse(source, state, verbose, nested + 1,
    444                            not nested and not items))
    445         if not sourcematch("|"):

~\anaconda3\lib\sre_parse.py in _parse(source, state, verbose, nested, first)
    666                 item = None
    667             if not item or item[0][0] is AT:
--> 668                 raise source.error("nothing to repeat",
    669                                    source.tell() - here + len(this))
    670             if item[0][0] in _REPEATCODES:

error: nothing to repeat at position 0
```

Esse erro deu por conta do '***' que é considerado um metacaractere. Para fazer uma substituição do padrão, é preciso escapar os asteriscos para que eles não sejam interpretados como metacaracteres. Vai ser usado \ para escapar os asteriscos e torná-los caracteres literais na expressão regular.

In [36]:

```
df_ocorrencia["ocorrencia_uf"] = df_ocorrencia["ocorrencia_uf"].str.replace("\*\*\*", " ")
df_ocorrencia["ocorrencia_aerodromo"] = df_ocorrencia["ocorrencia_aerodromo"].str.replace("\*\*\*", " ")
df_ocorrencia["investigacao_aeronave_liberada"] = df_ocorrencia["investigacao_aeronave_liberada"].str.replace("\*\
df_ocorrencia["divulgacao_relatorio_numero"] = df_ocorrencia["divulgacao_relatorio_numero"].str.replace("\*\*\*",

# Exibir a tabela atualizada
print(df_ocorrencia[["ocorrencia_uf", "ocorrencia_aerodromo", "investigacao_aeronave_liberada", "divulgacao_relato
```

```
<ipython-input-36-f66f2e18820a>:1: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_ocorrencia["ocorrencia_uf"] = df_ocorrencia["ocorrencia_uf"].str.replace("\*\*\*", " ")
<ipython-input-36-f66f2e18820a>:2: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_ocorrencia["ocorrencia_aerodromo"] = df_ocorrencia["ocorrencia_aerodromo"].str.replace("\*\*\*", " ")
<ipython-input-36-f66f2e18820a>:3: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_ocorrencia["investigacao_aeronave_liberada"] = df_ocorrencia["investigacao_aeronave_liberada"].str.replace("\*
\*\*", " ")
<ipython-input-36-f66f2e18820a>:4: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_ocorrencia["divulgacao_relatorio_numero"] = df_ocorrencia["divulgacao_relatorio_numero"].str.replace("\*\*\*",
' ')

     ocorrencia_uf ocorrencia_aerodromo investigacao_aeronave_liberada  \
0               SP                 SBSP                           SIM
1               ES                 SBVT                           SIM
2               SP                 SDAI                           SIM
3               MG                 SBBH                           SIM
```

Ocorrencia_tipo

In [37]:

```
df_ocorrencia_tipo["ocorrencia_tipo"] = df_ocorrencia_tipo["ocorrencia_tipo"].str.replace("\*\*\*", " ")
df_ocorrencia_tipo["ocorrencia_tipo_categoria"] = df_ocorrencia_tipo["ocorrencia_tipo_categoria"].str.replace("\*\
df_ocorrencia_tipo["taxonomia_tipo_icao"] = df_ocorrencia_tipo["taxonomia_tipo_icao"].str.replace("\*\*\*", " ")


# Exibir a tabela atualizada
print(df_ocorrencia_tipo[["ocorrencia_tipo","ocorrencia_tipo_categoria","taxonomia_tipo_icao"]])

```

```
<ipython-input-37-8409884585c8>:1: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_ocorrencia_tipo["ocorrencia_tipo"] = df_ocorrencia_tipo["ocorrencia_tipo"].str.replace("\*\*\*", " ")
<ipython-input-37-8409884585c8>:2: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_ocorrencia_tipo["ocorrencia_tipo_categoria"] = df_ocorrencia_tipo["ocorrencia_tipo_categoria"].str.replace("\*
\*\*", " ")
<ipython-input-37-8409884585c8>:3: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_ocorrencia_tipo["taxonomia_tipo_icao"] = df_ocorrencia_tipo["taxonomia_tipo_icao"].str.replace("\*\*\*", " ")

                                      ocorrencia_tipo  \
0            DESCOMPRESSÃO NÃO INTENCIONAL / EXPLOSIVA
1                                      ESTOURO DE PNEU
2                                      ESTOURO DE PNEU
3                                    EXCURSÃO DE PISTA
4    FALHA OU MAU FUNCIONAMENTO DE SISTEMA / COMPON...
5                                      ESTOURO DE PNEU
6                                    EXCURSÃO DE PISTA
7                            PERDA DE CONTROLE NO SOLO
```

Recomendacao

In [38]:
```
df_recomendacao["recomendacao_status"] = df_recomendacao["recomendacao_status"].str.replace("\*\*\*", " ")

# Exibir a tabela atualizada
print(df_recomendacao["recomendacao_status"])
```

```
0                        IMPLEMENTADA
1                  AGUARDANDO RESPOSTA
2                        IMPLEMENTADA
3                        IMPLEMENTADA
4                  AGUARDANDO RESPOSTA
5                  AGUARDANDO RESPOSTA
6                  AGUARDANDO RESPOSTA
7                        IMPLEMENTADA
8                  AGUARDANDO RESPOSTA
9                        IMPLEMENTADA
10                       IMPLEMENTADA
11                 AGUARDANDO RESPOSTA
12                 AGUARDANDO RESPOSTA
13                       IMPLEMENTADA
14                       IMPLEMENTADA
15                       IMPLEMENTADA
16                       IMPLEMENTADA
17                 AGUARDANDO RESPOSTA
18                       IMPLEMENTADA
```

Fator_contribuinte

In [39]:
```
df_fator_contribuinte["fator_condicionante"] = df_fator_contribuinte["fator_condicionante"].str.replace("\*\*\*",


print(df_fator_contribuinte['fator_condicionante'])
```

```
0                                INDIVIDUAL
1                     OPERAÇÃO DA AERONAVE
2                     OPERAÇÃO DA AERONAVE
3                     OPERAÇÃO DA AERONAVE
4                                INDIVIDUAL
5                     OPERAÇÃO DA AERONAVE
6                     OPERAÇÃO DA AERONAVE
7                                INDIVIDUAL
8
9
10                    OPERAÇÃO DA AERONAVE
11                    OPERAÇÃO DA AERONAVE
12                    OPERAÇÃO DA AERONAVE
13                  MANUTENÇÃO DA AERONAVE
14
15                    OPERAÇÃO DA AERONAVE
16                    OPERAÇÃO DA AERONAVE
17                    OPERAÇÃO DA AERONAVE
18                               INDIVIDUAL
```

Aeronave

In [40]:

```
df_aeronave["aeronave_matricula"] = df_aeronave["aeronave_matricula"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_operador_categoria"] = df_aeronave["aeronave_operador_categoria"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_tipo_veiculo"] = df_aeronave["aeronave_tipo_veiculo"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_fabricante"] = df_aeronave["aeronave_fabricante"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_tipo_icao"] = df_aeronave["aeronave_tipo_icao"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_motor_tipo"] = df_aeronave["aeronave_motor_tipo"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_motor_quantidade"] = df_aeronave["aeronave_motor_quantidade"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_registro_categoria"] = df_aeronave["aeronave_registro_categoria"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_registro_segmento"] = df_aeronave["aeronave_registro_segmento"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_fase_operacao"] = df_aeronave["aeronave_fase_operacao"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_tipo_operacao"] = df_aeronave["aeronave_tipo_operacao"].str.replace("\*\*\*", " ")
df_aeronave["aeronave_nivel_dano"] = df_aeronave["aeronave_nivel_dano"].str.replace("\*\*\*", " ")


print(df_aeronave[["aeronave_matricula","aeronave_operador_categoria","aeronave_tipo_veiculo","aeronave_fabricante
```

```
<ipython-input-40-55ffaa66bf2a>:1: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_aeronave["aeronave_matricula"] = df_aeronave["aeronave_matricula"].str.replace("\*\*\*", " ")
<ipython-input-40-55ffaa66bf2a>:2: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_aeronave["aeronave_operador_categoria"] = df_aeronave["aeronave_operador_categoria"].str.replace("\*\*\*", "
")
<ipython-input-40-55ffaa66bf2a>:3: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_aeronave["aeronave_tipo_veiculo"] = df_aeronave["aeronave_tipo_veiculo"].str.replace("\*\*\*", " ")
<ipython-input-40-55ffaa66bf2a>:4: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_aeronave["aeronave_fabricante"] = df_aeronave["aeronave_fabricante"].str.replace("\*\*\*", " ")
<ipython-input-40-55ffaa66bf2a>:5: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_aeronave["aeronave_tipo_icao"] = df_aeronave["aeronave_tipo_icao"].str.replace("\*\*\*", " ")
<ipython-input-40-55ffaa66bf2a>:6: FutureWarning: The default value of regex will change from True to False in a fu
ture version.
  df_aeronave["aeronave_motor_tipo"] = df_aeronave["aeronave_motor_tipo"].str.replace("\*\*\*", " ")
```

Como parte do processo de substituição de valores, nas tabelas foi (df_ocorrencia, df_ocorrencia_tipo, df_recomendacao, df_fator_contribuinte, df_aeronave) em algumas colunas tinhamos o valor "***" e foi substituido para "".

Passo a Passo:

1 - 'nome_da_tabela["nome_da_coluna"]' é definido qual tabela será alterada

2 - 'str.replace("***", " ")': Em 'str.replace' informo que é um texto e que será substituido, em ("***", " ") primeiro vem o valor antigo, em seguida o novo valor

6 – Renomeando os Nomes das Colunas:

Nomes de colunas descritivos tornam o conjunto de dados mais fácil de entender. Isso ajuda a compreender rapidamente o conteúdo de cada coluna.

Ocorrencia

In [41]:

```python
# Dicionário com o mapeamento dos nomes das colunas
novo_nome_colunas = {
    'ocorrencia_classificacao': 'classificacao',
    'ocorrencia_latitude': 'latitude',
    'ocorrencia_longitude': 'longitude',
    'ocorrencia_cidade': 'cidade',
    'ocorrencia_uf': 'uf',
    'ocorrencia_pais': 'pais',
    'ocorrencia_aerodromo': 'aerodromo',
    'ocorrencia_dia': 'dia_da_ocorrencia',
    'ocorrencia_hora': 'hora_da_ocorrencia',
    'ocorrencia_saida_pista': 'saida_pista'
}

# Renomear as colunas usando o dicionário
df_ocorrencia = df_ocorrencia.rename(columns=novo_nome_colunas)

# Exibir o DataFrame com as colunas renomeadas
print(df_ocorrencia)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

recomendacao

In [42]:

```
# Dicionário com o mapeamento dos nomes das colunas
novo_nome_colunas = {
    'recomendacao_numero': 'numero_da_recomendacao',
    'recomendacao_dia_assinatura': 'dia_da_assinatura',
    'recomendacao_dia_encaminhamento': 'dia_do_encaminhamento',
    'recomendacao_dia_feedback': 'dia_do_feedback',
    'recomendacao_conteudo': 'conteudo_da_recomendacao',
    'recomendacao_status': 'status_da_recomendacao'}

# Renomear as colunas usando o dicionário
df_recomendacao = df_recomendacao.rename(columns=novo_nome_colunas)

# Exibir o DataFrame com as colunas renomeadas
print(df_recomendacao)
```

```
     codigo_ocorrencia4         numero_da_recomendacao dia_da_assinatura  \
0                 80258         A-137/CENIPA/2019 - 01        2021-11-16
1                 80252         A-094/CENIPA/2021 - 01        2022-09-21
2                 80073         A-063/CENIPA/2021 - 01        2021-11-12
3                 80073         A-063/CENIPA/2021 - 02        2021-11-12
4                 79997         A-050/CENIPA/2021 - 01        2022-08-05
5                 79983         A-046/CENIPA/2021 - 01        2022-08-05
6                 79864         A-019/CENIPA/2021 - 01        2022-09-21
7                 79748         A-156/CENIPA/2020 - 01        2022-04-12
8                 79717         A-147/CENIPA/2021 - 01        2022-09-21
9                 79713         A-146/CENIPA/2020 - 01        2021-07-08
10                79713         A-146/CENIPA/2020 - 02        2021-07-08
11                79713         A-146/CENIPA/2020 - 03        2021-07-08
12                79692         A-140/CENIPA/2020 - 01        2021-07-21
13                79619         A-130/CENIPA/2020 - 01        2021-07-08
14                79565         A-117/CENIPA/2020 - 01        2021-07-08
15                79536         A-107/CENIPA/2020 - 01        2021-03-29
16                79536         A-107/CENIPA/2020 - 02        2021-03-29
17                79505         A-101/CENIPA/2020 - 01        2022-09-21
```

Fator_contribuinte

In [43]:

```
# Dicionário com o mapeamento dos nomes das colunas
novo_nome_colunas = {
    'fator_nome': 'nome_do_fator',
    'fator_aspecto': 'aspecto_do_fator',
    'fator_area': 'area_do_fator'}

# Renomear as colunas usando o dicionário
df_fator_contribuinte = df_fator_contribuinte.rename(columns=novo_nome_colunas)

# Exibir o DataFrame com as colunas renomeadas
print(df_fator_contribuinte)
```

```
    codigo_ocorrencia3                          nome_do_fator  \
0                80383                                ATITUDE
1                80383             JULGAMENTO DE PILOTAGEM
2                80383                 PLANEJAMENTO DE VOO
3                80383                SUPERVISÃO GERENCIAL
4                80365                                ATITUDE
5                80365             JULGAMENTO DE PILOTAGEM
6                80365           POUCA EXPERIÊNCIA DO PILOTO
7                80365                    PROCESSO DECISÓRIO
8                80258          INFRAESTRUTURA AEROPORTUÁRIA
9                80258           PRESENÇA DE FAUNA (NÃO AVE)
10               80252             JULGAMENTO DE PILOTAGEM
11               80252                 PLANEJAMENTO DE VOO
12               80252           POUCA EXPERIÊNCIA DO PILOTO
13               80073            MANUTENÇÃO DA AERONAVE
14               80002                          OUTRO FATOR
15               79999             JULGAMENTO DE PILOTAGEM
16               79999                 PLANEJAMENTO DE VOO
17               79999           POUCA EXPERIÊNCIA DO PILOTO
```

Aeronave

In [44]:

```python
# Dicionário com o mapeamento dos nomes das colunas
novo_nome_colunas = {
    'aeronave_matricula': 'matricula_da_aeronave',
    'aeronave_tipo_veiculo': 'tipo_da_aeronave',
    'aeronave_fabricante': 'fabricante_da_aeronave',
    'aeronave_modelo': 'modelo_da_aeronave',
    'aeronave_motor_tipo': 'tipo_do_motor',
    'aeronave_motor_quantidade': 'quantidade_de_motor',
    'aeronave_pmd': 'peso_maximo_decolagem_pmd',
    'aeronave_pmd_categoria': 'peso_maximo_decolagem_categoria',
    'aeronave_assentos': 'quantidade_de_assentos',
    'aeronave_ano_fabricacao': 'ano_de_fabricacao',
    'aeronave_pais_fabricante': 'pais_fabricante',
    'aeronave_pais_registro': 'pais_registro',
    'aeronave_registro_categoria': 'registro_da_categoria',
    'aeronave_registro_segmento': 'registro_de_segmento',
    'aeronave_voo_origem': 'origem_do_voo',
    'aeronave_voo_destino': 'destino_do_voo',
    'aeronave_fase_operacao': 'fase_da_operacao',
    'aeronave_tipo_operacao': 'tipo_da_operacao',
    'aeronave_nivel_dano': 'nivel_do_dano',
    'aeronave_fatalidades_total': 'fatalidade'}

# Renomear as colunas usando o dicionário
df_aeronave = df_aeronave.rename(columns=novo_nome_colunas)

# Exibir o DataFrame com as colunas renomeadas
print(df_aeronave)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

Como parte do processo para renomear as colunas das tabelas (ocorrencia, recomendacao, fator_contribuinte, aeronave) é criado um dicionário onde contém pares chave-valor, onde a chave representa o nome atual da coluna no DataFrame e o valor representa o novo nome que você deseja atribuir a essa coluna.

A linha a seguir utiliza o método ‘rename’ para renomear as colunas de acordo com o dicionário de mapeamento novo_nome_colunas. Isso significa que cada coluna no DataFrame cujo nome corresponde a uma chave no dicionário será renomeada para o valor correspondente a essa chave.

Análise e Exploração dos Dados

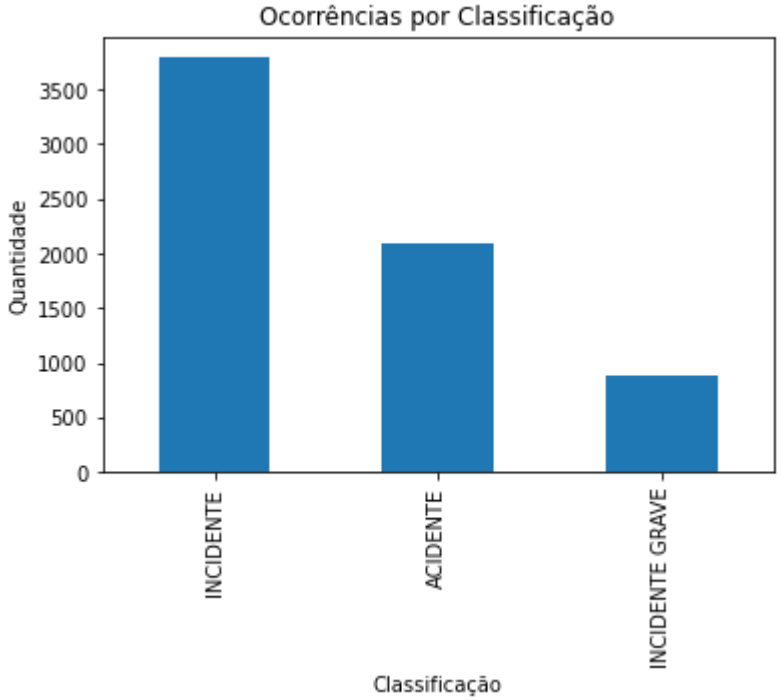
A análise e exploração de dados desempenham um papel fundamental em todos os setores da sociedade moderna. Essa prática é essencial para a tomada de decisões informadas, o desenvolvimento de estratégias eficazes e a compreensão profunda de fenômenos complexos. A importância da análise e exploração de dados pode ser resumida em várias dimensões cruciais:

Análises relacionadas à Tabela "ocorrencia":

In [52]:

```
# 1. Ocorrências por classificação
classificacao_counts = df_ocorrencia['classificacao'].value_counts()

# Crie um gráfico de barras
classificacao_counts.plot(kind='bar')

# Defina os rótulos dos eixos
plt.xlabel('Classificação')
plt.ylabel('Quantidade')

# Defina o título do gráfico
plt.title('Ocorrências por Classificação')


print(f'Quantidade de Classificação: \n{classificacao_counts}')
plt.show()
```

```
Quantidade de Classificação:
INCIDENTE          3795
ACIDENTE           2090
INCIDENTE GRAVE     884
Name: classificacao, dtype: int64
```



'classificacao_counts = df_ocorrencia['classificacao'].value_counts():' Aqui, foi criado uma variável chamada classificacao_counts que armazena a contagem de valores únicos na coluna 'classificacao' do DataFrame df_ocorrencia. O método value_counts() retorna uma série que conta quantas vezes cada valor único aparece na coluna 'classificacao'.

'classificacao_counts.plot(kind='bar'):' Esta linha cria um gráfico de barras usando a série classificacao_counts. O argumento kind='bar' especifica que retorne um gráfico de barras.

'plt.xlabel('Classificação'):' Esta linha define um rótulo para o eixo x do gráfico, que é "Classificação".

'plt.ylabel('Quantidade'):' Esta linha define um rótulo para o eixo y do gráfico, que é "Quantidade".

'plt.title('Ocorrências por Classificação'):' Esta linha define o título do gráfico como "Ocorrências por Classificação".

'print(f'Quantidade de Classificação: \n{classificacao_counts}'):' Isso imprime a contagem de classificações no formato "Quantidade de Classificação:" seguido das contagens reais, com uma quebra de linha entre cada uma.

'plt.show():' Essa linha exibe o gráfico de barras na janela gráfica ou no ambiente de desenvolvimento. O gráfico mostrará a contagem de ocorrências para cada classificação presente na coluna 'classificacao' do DataFrame.

O código é útil para visualizar como as ocorrências estão distribuídas por classificação usando um gráfico de barras. Ele fornece uma representação visual das contagens de cada classificação.

In [53]:
```
# 2. Relação entre saída de pista e recomendações
saida_pista_recomendacoes = df_ocorrencia.groupby('saida_pista')['total_recomendacoes'].mean()

print(f'Relação entre Saída de pista e recomendações:\n{saida_pista_recomendacoes}')
```

```
Relação entre Saída de pista e recomendações:
saida_pista
NAO    0.283016
SIM    0.617834
Name: total_recomendacoes, dtype: float64
```

Esse código, está calculando a relação entre a variável 'saida_pista' (que indica se a ocorrência envolveu saída de pista ou não) e a média das 'total_recomendacoes'.

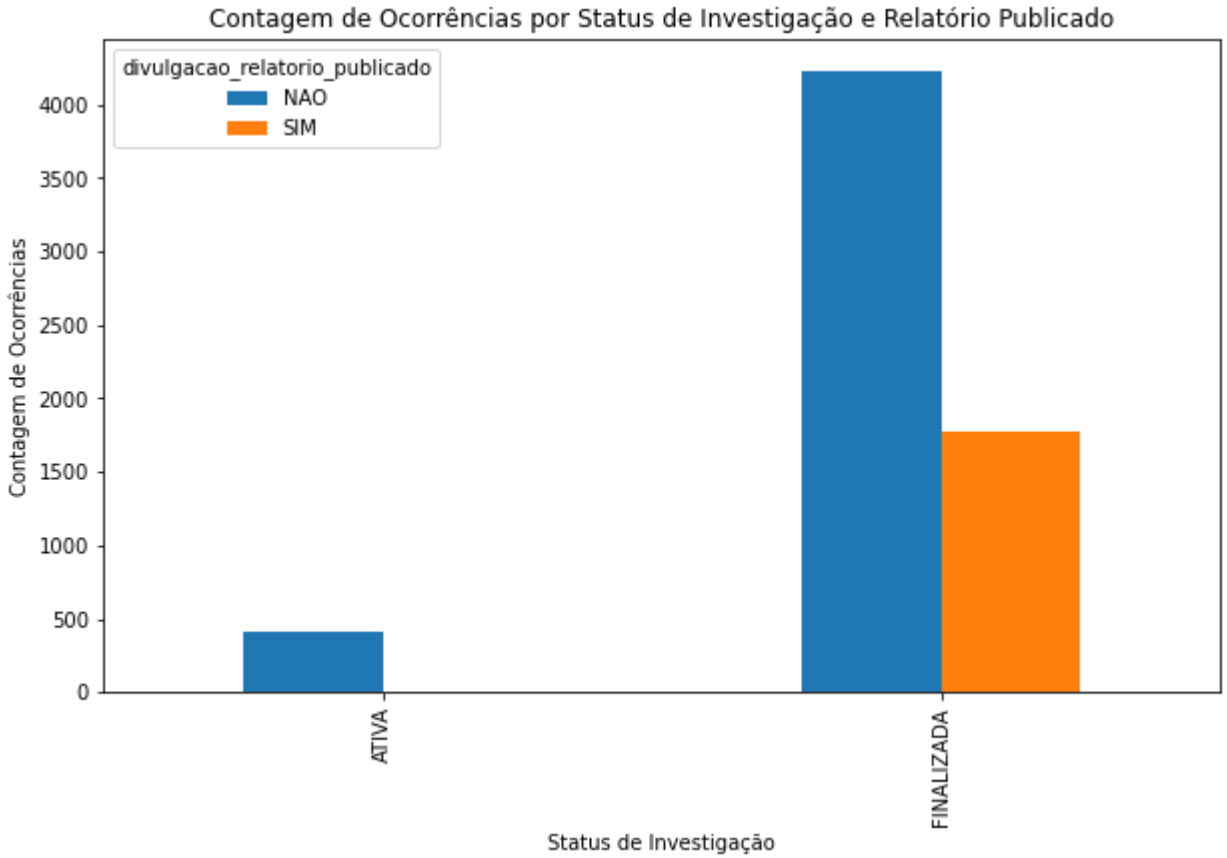
'saida_pista_recomendacoes = df_ocorrencia.groupby('saida_pista')['total_recomendacoes'].mean():' Nesta linha, está usando o método groupby() do DataFrame df_ocorrencia para agrupar os dados com base na coluna 'saida_pista'. Em seguida, está calculando a média da coluna 'total_recomendacoes' para cada grupo. Isso cria uma série que relaciona os valores únicos da coluna 'saida_pista' com as médias das 'total_recomendacoes' para cada grupo.

'print(f'Relação entre Saída de pista e recomendações:\n{saida_pista_recomendacoes}'):' Esta linha imprime o resultado da relação entre a saída de pista e as recomendações. A mensagem de impressão inclui o título "Relação entre Saída de pista e recomendações" e, em seguida, exibe os valores calculados na série saida_pista_recomendacoes.

O resultado final deste código mostrará a média das recomendações para cada categoria de "saida_pista" presente no DataFrame. Isso pode ajudar a entender a média de recomendações associadas a ocorrências de saída de pista em comparação com aquelas que não envolvem saída de pista. A média das recomendações é um indicador da gravidade ou importância das ocorrências.

In [54]:

```
# 3. Contagem de ocorrências por status de investigação e relatório publicado
contagem_por_status = df_ocorrencia.groupby(['investigacao_status', 'divulgacao_relatorio_publicado']).size().unst

contagem_por_status.plot(kind='bar', figsize=(10, 6))

#Rótulos dos eixos
plt.xlabel('Status de Investigação')
plt.ylabel('Contagem de Ocorrências')
plt.title('Contagem de Ocorrências por Status de Investigação e Relatório Publicado')


plt.show()
print(contagem_por_status)

```



divulgacao_relatorio_publicado	NAO	SIM
investigacao_status		
ATIVA	415	0
FINALIZADA	4235	1778

Esse código, está realizando uma análise da contagem de ocorrências com base em duas variáveis: "investigacao_status" (status de investigação) e "divulgacao_relatorio_publicado" (se o relatório foi publicado ou não).

'contagem_por_status = df_ocorrencia.groupby(['investigacao_status', 'divulgacao_relatorio_publicado']).size().unstack(fill_value=0):' Nesta linha, está usando o método groupby() do DataFrame df_ocorrencia para agrupar os dados com base em duas colunas, "investigacao_status" e "divulgacao_relatorio_publicado". Em seguida, está usando size() para contar o número de ocorrências em cada combinação de valores dessas

duas colunas. O método unstack() é usado para transformar os resultados em um DataFrame, onde as combinações de valores de "investigacao_status" se tornam as colunas e as combinações de valores de "divulgacao_relatorio_publicado" se tornam os índices das linhas. O argumento fill_value=0 é usado para preencher as células vazias com 0, caso não haja ocorrências para alguma combinação.

'contagem_por_status.plot(kind='bar', figsize=(10, 6)):' Esta linha cria um gráfico de barras com base no DataFrame contagem_por_status. O argumento kind='bar' especifica que deseja um gráfico de barras. O argumento figsize=(10, 6) define o tamanho da figura do gráfico.

'plt.xlabel('Status de Investigação'):' Esta linha define um rótulo para o eixo x do gráfico, que é "Status de Investigação".

'plt.ylabel('Contagem de Ocorrências'):' Esta linha define um rótulo para o eixo y do gráfico, que é "Contagem de Ocorrências".

'plt.title('Contagem de Ocorrências por Status de Investigação e Relatório Publicado'):' Esta linha define o título do gráfico como "Contagem de Ocorrências por Status de Investigação e Relatório Publicado".

'plt.show():' Esta linha exibe o gráfico de barras na janela gráfica.

'print(contagem_por_status):' Por fim, é imprisso o DataFrame contagem_por_status, que exibe a contagem de ocorrências para cada combinação de status de investigação e relatório publicado.

In [55]:
```python
# 4. Relação entre localização (UF) e total de fatalidades
uf_fatalidades = df_ocorrencia.groupby('uf')['total_aeronaves_envolvidas'].sum()

# Ordene do maior para o menor
uf_fatalidades = uf_fatalidades.sort_values(ascending=False)

print(uf_fatalidades)
```

```
uf
SP    1653
MG     640
RJ     588
PR     556
RS     400
GO     368
MT     360
PA     345
AM     279
BA     258
SC     228
MS     194
DF     159
PE     130
CE      96
ES      92
MA      86
RR      74
AC      66
TO      60
RO      55
PI      46
AL      34
PB      28
SE      23
RN      22
AP      15
         4
Name: total_aeronaves_envolvidas, dtype: int64
```

Nesse código, está calculando a relação entre a localização (Unidade da Federação - UF) e o total de fatalidades em ocorrências de aviação, com base nas informações contidas no DataFrame df_ocorrencia.

'uf_fatalidades = df_ocorrencia.groupby('uf')['total_aeronaves_envolvidas'].sum():' Nesta linha, está agrupando os dados do DataFrame df_ocorrencia com base na coluna 'uf' (Unidade da Federação) e calculando a soma dos valores da coluna 'total_aeronaves_envolvidas' para cada UF. Isso cria uma série que relaciona cada UF com o total de aeronaves envolvidas em ocorrências naquela UF.

'uf_fatalidades = uf_fatalidades.sort_values(ascending=False):' Aqui, está classificando a série uf_fatalidades em ordem decrescente (do maior valor para o menor valor) usando o método sort_values(). O argumento ascending=False é usado para indicar que deseja uma classificação decrescente.

'print(uf_fatalidades):' Esta linha imprime a série uf_fatalidades após a classificação decrescente. O resultado será uma lista das UFs com o maior total de aeronaves envolvidas em ocorrências de aviação.

Em resumo, esse código fornece uma lista das Unidades da Federação (UFs) com o maior número de aeronaves envolvidas em ocorrências, ajudando a identificar as regiões com um maior histórico de fatalidades ou incidentes na aviação. A classificação decrescente permite que você veja as UFs mais impactadas no topo da lista.

In [56]:

```
# 5. Relação entre UF x classificacao
# Agrupar por UF e classificação, contar ocorrências e reformatar o DataFrame
uf_classificacao_counts = df_ocorrencia.groupby(['uf', 'classificacao']).size().unstack(fill_value=0)

# Ordenar do maior para o menor com base na classificação "Acidente"
uf_classificacao_counts_sorted = uf_classificacao_counts.sort_values(by='ACIDENTE', ascending=False)

print(uf_classificacao_counts_sorted)

```

```
classificacao  ACIDENTE  INCIDENTE  INCIDENTE GRAVE
uf
SP                435       1027              163
MT                198        106               55
RS                180        157               58
MG                172        380               86
PR                161        316               71
PA                148        150               46
GO                145        128               86
MS                 95         64               32
RJ                 78        445               55
AM                 75        176               28
BA                 75        137               41
SC                 71        132               20
RR                 41         23                9
MA                 39         30               17
TO                 24         23               13
CE                 23         57               15
PE                 21         91               14
RO                 20         25                9
PI                 16         21                8
AC                 15         39               12
ES                 14         64               13
DF                 10        131               15
SE                  9         11                2
PB                  8         15                5
RN                  6         12                4
AP                  5          9                1
AL                  3         25                6
                    3          1                0
```

Neste código, está explorando a relação entre a Unidade da Federação (UF) e as classificações de ocorrências em aviação.

'uf_classificacao_counts = df_ocorrencia.groupby(['uf', 'classificacao']).size().unstack(fill_value=0):' Nesta linha, está agrupando os dados do DataFrame df_ocorrencia com base em duas colunas, 'uf' e 'classificacao'. Em seguida, está contando o número de ocorrências para cada combinação de UF e classificação. O método unstack() é usado para reformatar os dados, transformando as classificações em colunas e as UFs em índices das linhas. O argumento fill_value=0 é usado para preencher as células vazias com 0, caso não haja ocorrências para alguma combinação.

'uf_classificacao_counts_sorted = uf_classificacao_counts.sort_values(by='ACIDENTE', ascending=False):' Aqui, está classificando o DataFrame uf_classificacao_counts com base na coluna 'ACIDENTE' em ordem decrescente. Isso significa que você classificando as UFs com base no número de ocorrências classificadas como "ACIDENTE" em cada UF.

'print(uf_classificacao_counts_sorted):' Esta linha imprime o DataFrame uf_classificacao_counts_sorted após a classificação. O resultado será uma tabela que mostra o número de ocorrências classificadas como "ACIDENTE" em cada UF, com as UFs classificadas do maior para o menor número de ocorrências de acidentes.

Em resumo, esse código ajuda a identificar as UFs com o maior número de ocorrências classificadas como "ACIDENTE" na aviação, permitindo que analise a distribuição dessas ocorrências em todo o país.

In [57]:
```
# 6. Relação entre Fatalidade x classificacao

# Junte as tabelas df_ocorrencia e df_aeronave usando a coluna 'codigo_ocorrencia2' como chave
relacao_ocorrencia_aeronave = pd.merge(df_ocorrencia, df_aeronave, on='codigo_ocorrencia2')

# Agrupe por classificação de ocorrência e calcule o total de fatalidades
total_fatalidades_por_classificacao = relacao_ocorrencia_aeronave.groupby('classificacao')['fatalidade'].sum()

print(total_fatalidades_por_classificacao)

```

```
classificacao
ACIDENTE           923
INCIDENTE            0
INCIDENTE GRAVE      0
Name: fatalidade, dtype: int32
```

Neste código, está explorando a relação entre a classificação de ocorrências de aviação e o total de fatalidades associadas a essas ocorrências.

'relacao_ocorrencia_aeronave = pd.merge(df_ocorrencia, df_aeronave, on='codigo_ocorrencia2'):' Nesta linha, está mesclando dois DataFrames, df_ocorrencia e df_aeronave, com base na coluna 'codigo_ocorrencia2' como chave. Isso cria um novo DataFrame chamado relacao_ocorrencia_aeronave que combina informações de ocorrências e aeronaves com base no código de ocorrência.

'total_fatalidades_por_classificacao = relacao_ocorrencia_aeronave.groupby('classificacao')['fatalidade'].sum():' Aqui, está agrupando os dados do DataFrame relacao_ocorrencia_aeronave com base na coluna 'classificacao' (classificação de ocorrência) e, em seguida, está calculando a soma das fatalidades (coluna 'fatalidade') para cada classe de ocorrência. Isso cria uma série que relaciona as diferentes classificações de ocorrências com o total de fatalidades associadas a cada uma.

'print(total_fatalidades_por_classificacao):' Esta linha imprime a série total_fatalidades_por_classificacao após o cálculo. O resultado será uma lista das classificações de ocorrências e o total de fatalidades associadas a cada uma.

Em resumo, esse código fornece informações sobre o total de fatalidades em ocorrências de aviação com base em sua classificação. Ele ajuda a entender a gravidade das ocorrências em diferentes categorias de classificação, permitindo uma análise mais detalhada da segurança e das medidas de prevenção.

In [58]:

```
# 7. Relação entre uf x fatalidade

# Junte as tabelas df_ocorrencia e df_aeronave usando a coluna 'codigo_ocorrencia2' como chave
relacao_ocorrencia_aeronave = pd.merge(df_ocorrencia, df_aeronave, on='codigo_ocorrencia2')

# Agrupe por classificação de ocorrência e calcule o total de fatalidades
total_fatalidades_por_classificacao = relacao_ocorrencia_aeronave.groupby('uf')['fatalidade'].sum()

# Ordene do maior para o menor
total_fatalidades_por_classificacao = total_fatalidades_por_classificacao.sort_values(ascending=False)

print(total_fatalidades_por_classificacao)
```

```
uf
SP    172
PA     96
MT     90
MG     88
GO     70
PR     70
AM     52
RJ     43
MS     34
RS     32
BA     28
PE     22
MA     21
SC     19
RR     18
TO     15
PI     13
RN      7
ES      6
RO      5
CE      5
AC      4
SE      4
AL      4
AP      3
DF      2
PB      0
        0
Name: fatalidade, dtype: int32
```

relacao_ocorrencia_aeronave = pd.merge(df_ocorrencia, df_aeronave, on='codigo_ocorrencia2'): Nesta etapa, ocorre a combinação de dois conjuntos de dados, representados pelos DataFrames df_ocorrencia e df_aeronave. A junção é realizada com base na coluna 'codigo_ocorrencia2', resultando em um novo DataFrame chamado relacao_ocorrencia_aeronave. Esse DataFrame combina informações sobre ocorrências e aeronaves usando o código de ocorrência como chave.

'total_fatalidades_por_classificacao = relacao_ocorrencia_aeronave.groupby('classificacao')['fatalidade'].sum(): Nessa fase, o DataFrame relacao_ocorrencia_aeronave é agrupado de acordo com a coluna 'classificacao' (classificação de ocorrência). Em seguida, é calculada a soma das fatalidades (presentes na coluna 'fatalidade') para cada classe de ocorrência. O resultado é uma série que relaciona as diferentes classificações de ocorrências com o total de fatalidades associadas a cada uma.

'print(total_fatalidades_por_classificacao): Esta linha tem como objetivo imprimir na tela a série total_fatalidades_por_classificacao após a realização dos cálculos. O resultado final será uma lista das classificações de ocorrências e o total de fatalidades associadas a cada uma.

Em resumo, o propósito desse código é fornecer informações detalhadas sobre o total de fatalidades em ocorrências de aviação, categorizando esses dados de acordo com suas classificações. Essa análise facilita a compreensão da gravidade das ocorrências em diferentes categorias, possibilitando uma avaliação mais aprofundada da segurança e das medidas de prevenção na aviação.

ocorrencia_tipo

In [59]:

```
# 8. Contagem de Tipos de Ocorrências
contagem_tipos_ocorrencias = df_ocorrencia_tipo['ocorrencia_tipo'].value_counts()

# Top 10 com mais casos
top_10_mais_casos = contagem_tipos_ocorrencias.head(10)

# Top 10 com menos casos
top_10_menos_casos = contagem_tipos_ocorrencias.tail(10)

print("Top 10 com mais casos:")
print(top_10_mais_casos)

print("\nTop 10 com menos casos:")
print(top_10_menos_casos)
```

```
Top 10 com mais casos:
FALHA DO MOTOR EM VOO                                   855
FALHA OU MAU FUNCIONAMENTO DE SISTEMA / COMPONENTE      778
ESTOURO DE PNEU                                         706
PERDA DE CONTROLE NO SOLO                               518
PERDA DE CONTROLE EM VOO                                425
COM TREM DE POUSO                                       419
COLISÃO COM AVE                                         340
OUTROS                                                  332
EXCURSÃO DE PISTA                                       294
COLISÃO COM OBSTÁCULO DURANTE A DECOLAGEM E POUSO       205
Name: ocorrencia_tipo, dtype: int64

Top 10 com menos casos:
PERDA DE SEPARAÇÃO / COLISÃO EM VOO    2
REBOQUE DE PLANADOR                    2
COLISÃO EM VOO COM OBSTÁCULO           2
EVACUAÇÃO                              1
SAÍDA DE PISTA                         1
EXPLOSÃO                               1
RELACIONADO COM SECURITY               1
HIPÓXIA                                1
CAUSADO POR RICOCHETE                  1
nan                                    1
Name: ocorrencia_tipo, dtype: int64
```

"contagem_tipos_ocorrencias = df_ocorrencia_tipo['ocorrencia_tipo'].value_counts():" Calcula a contagem de ocorrências para cada tipo presente na coluna 'ocorrencia_tipo' do DataFrame df_ocorrencia_tipo. O resultado é uma série que mostra a quantidade de ocorrências para cada tipo.

"top_10_mais_casos = contagem_tipos_ocorrencias.head(10)": Seleciona os 10 tipos de ocorrências com o maior número de casos, com base na contagem feita anteriormente. Cria um novo objeto contendo esses dados.

"top_10_menos_casos = contagem_tipos_ocorrencias.tail(10)": Seleciona os 10 tipos de ocorrências com o menor número de casos, com base na contagem feita anteriormente. Cria um novo objeto contendo esses dados.

"print("Top 10 com mais casos:")": Imprime uma mensagem indicando que a lista a seguir é referente aos 10 tipos de ocorrências com o maior número de casos.

"print(top_10_mais_casos)": Imprime a lista dos 10 tipos de ocorrências com o maior número de casos.

"print("\nTop 10 com menos casos:")": Imprime uma mensagem indicando que a lista a seguir é referente aos 10 tipos de ocorrências com o menor número de casos.

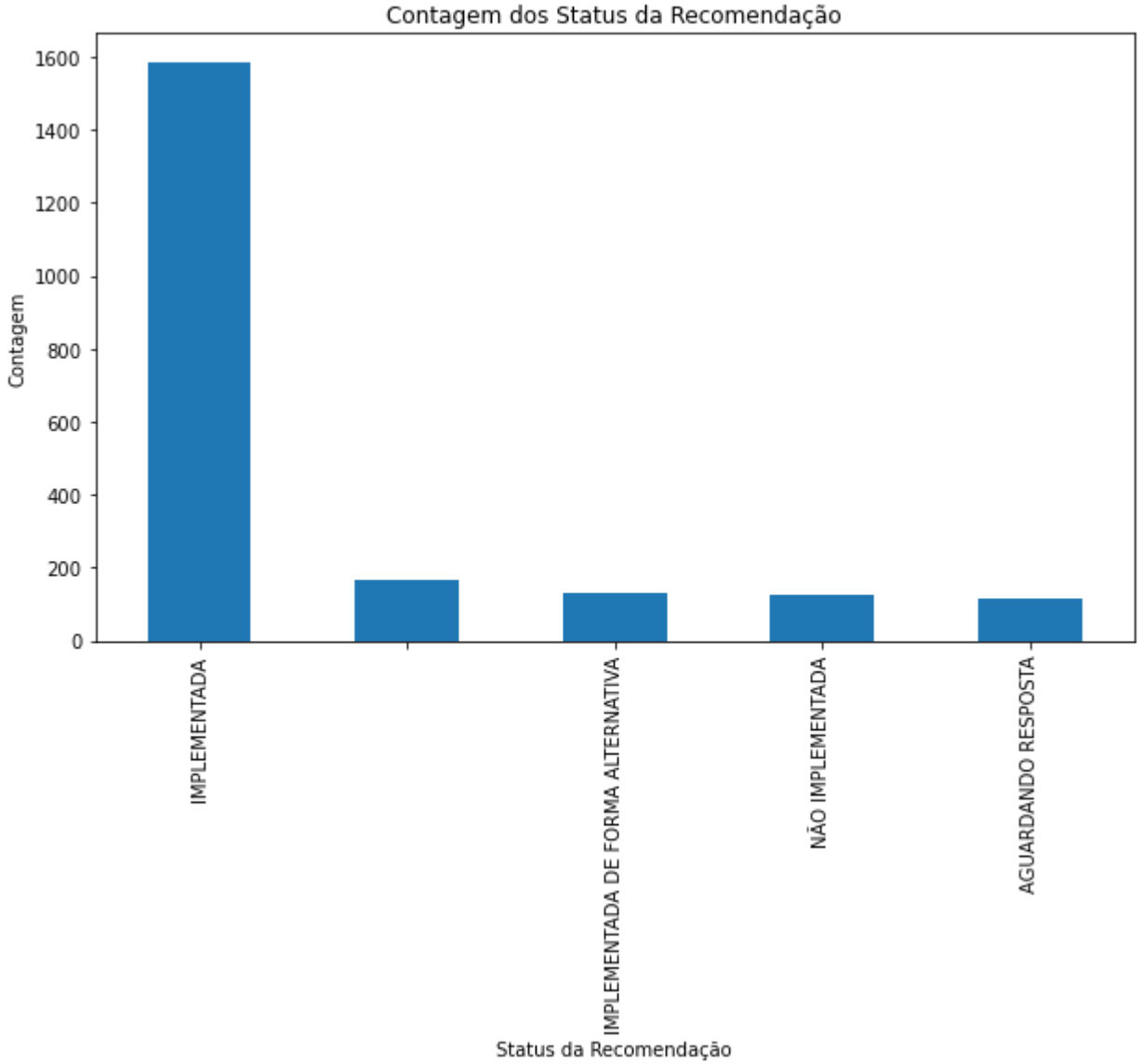
"print(top_10_menos_casos)": Imprime a lista dos 10 tipos de ocorrências com o menor número de casos.

Em resumo, esse código gera e exibe as contagens dos tipos de ocorrências em aviação, destacando os 10 tipos com mais e menos casos. Essa análise é útil para identificar padrões e focar em áreas específicas de preocupação ou prevenção.

Recomendacao

In [72]:

```
# 9 Contagem dos Status da Recomendacao

# Calcula a contagem dos status da recomendação
status_counts = df_recomendacao['status_da_recomendacao'].value_counts()

# Calcula a porcentagem
status_percentages = (status_counts / status_counts.sum()) * 100

# Crie um novo DataFrame com contagem e porcentagem
status_df = pd.DataFrame({'Contagem': status_counts, '%': status_percentages})

#Gráfico de Barras
plt.figure(figsize=(10, 6))  # Defina o tamanho do gráfico
status_counts.plot(kind='bar')

# Defina os rótulos dos eixos
plt.xlabel('Status da Recomendação')
plt.ylabel('Contagem')
plt.title('Contagem dos Status da Recomendação')

# Exiba o gráfico
plt.show()

# Exiba o DataFrame resultante
print(status_df)
```



	Contagem	%
IMPLEMENTADA	1585	74.764151
	168	7.924528
IMPLEMENTADA DE FORMA ALTERNATIVA	128	6.037736
NÃO IMPLEMENTADA	126	5.943396
AGUARDANDO RESPOSTA	113	5.330189

status_counts = df_recomendacao['status_da_recomendacao'].value_counts(): Calcula a contagem dos diferentes valores presentes na coluna 'status_da_recomendacao' do DataFrame df_recomendacao. O resultado é uma série que mostra a quantidade de ocorrências para cada status de recomendação.

status_percentages = (status_counts / status_counts.sum()) * 100: Calcula as porcentagens correspondentes a cada status de recomendação, dividindo a contagem de cada status pelo total de ocorrências e multiplicando por 100.

status_df = pd.DataFrame({'Contagem': status_counts, '%': status_percentages}): Cria um novo DataFrame chamado status_df, que combina as contagens e as porcentagens calculadas anteriormente. Isso fornece uma visão tabular dos dados.

plt.figure(figsize=(10, 6)): Define o tamanho da figura do gráfico como 10 unidades de largura por 6 unidades de altura.

status_counts.plot(kind='bar'): Gera um gráfico de barras usando a contagem de cada status de recomendação. Esse gráfico mostra visualmente a distribuição das contagens.

plt.xlabel('Status da Recomendação'): Define o rótulo do eixo x como 'Status da Recomendação'.

plt.ylabel('Contagem'): Define o rótulo do eixo y como 'Contagem'.

plt.title('Contagem dos Status da Recomendação'): Define o título do gráfico como 'Contagem dos Status da Recomendação'.

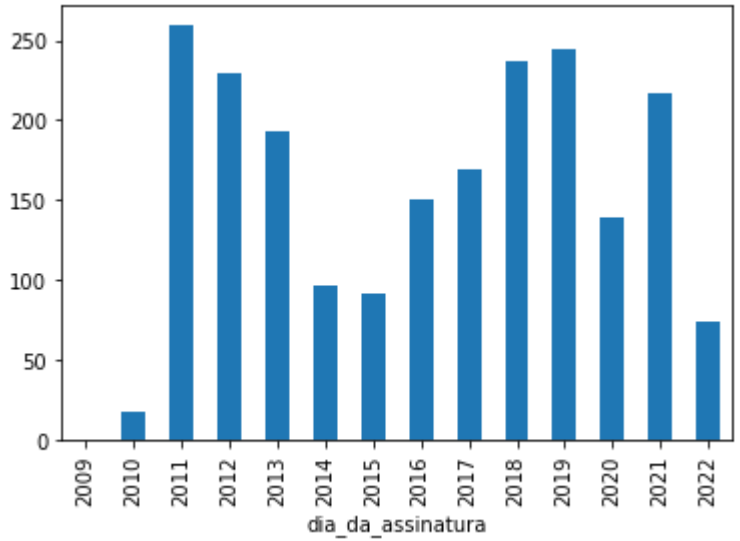
plt.show(): Exibe o gráfico de barras.

print(status_df): Imprime o DataFrame resultante, que contém tanto a contagem quanto as porcentagens dos diferentes status de recomendação.

Resumidamente, esse código realiza a análise estatística dos diferentes status de recomendação, calcula as contagens e porcentagens correspondentes, gera um gráfico de barras para visualização e exibe um DataFrame tabular com essas informações. Isso é útil para entender a distribuição e proporções dos diferentes estados de recomendação em um conjunto de dados.

In [73]:
```
# 10. Quantidade de Recomendações com assinatura por ano
df_recomendacao.groupby(df_recomendacao['dia_da_assinatura'].dt.year)['numero_da_recomendacao'].count().plot(kind=
```

Out[73]: <AxesSubplot:xlabel='dia_da_assinatura'>



df_recomendacao.groupby(df_recomendacao['dia_da_assinatura'].dt.year): Agrupa o DataFrame df_recomendacao pelo ano da coluna 'dia_da_assinatura'. Isso utiliza o método groupby para agrupar os dados de acordo com os anos em que as recomendações foram assinadas.

['numero_da_recomendacao'].count(): Conta o número de ocorrências (recomendações) para cada grupo de anos. Utiliza a coluna 'numero_da_recomendacao' para contar a quantidade de recomendações em cada ano.

.plot(kind='bar'): Gera um gráfico de barras com base nas contagens obtidas no passo anterior. O argumento kind='bar' especifica que o gráfico a ser gerado é do tipo barras.

Em resumo, esse código agrupa as recomendações por ano de assinatura, conta o número de recomendações para cada ano e cria um gráfico de barras que mostra a distribuição da quantidade de recomendações ao longo dos anos. Isso é útil para visualizar padrões ou tendências na quantidade de recomendações ao longo do tempo.

Fator Contribuinte

In [62]:
```
# 11 Relação aspecto_do_fator x area_do_fator
pd.crosstab(df_fator_contribuinte['aspecto_do_fator'], df_fator_contribuinte['area_do_fator'])
```

Out[62]:

area_do_fator	FATOR HUMANO	FATOR MATERIAL	FATOR OPERACIONAL	OUTRO
aspecto_do_fator				
ASPECTO DE FABRICAÇÃO	0	12	0	0
ASPECTO DE MANUSEIO DO MATERIAL	0	5	0	0
ASPECTO DE PROJETO	0	16	0	0
ASPECTO MÉDICO	88	0	0	0
ASPECTO PSICOLÓGICO	1530	0	0	0
DESEMPENHO DO SER HUMANO	0	0	2757	0
ELEMENTOS RELACIONADOS AO AMBIENTE OPERACIONAL	0	0	137	0
ERGONOMIA	14	0	0	0
INFRAESTRUTURA AEROPORTUÁRIA	0	0	91	0
INFRAESTRUTURA DE TRÁFEGO AÉREO	0	0	10	0
OUTRO	0	0	0	116

pd.crosstab: Este é um método do pandas que cria uma tabela de frequência cruzada, ou seja, uma tabela que mostra a distribuição conjunta de duas variáveis categóricas. Ele é usado para analisar a relação entre as categorias das variáveis.

df_fator_contribuinte['aspecto_do_fator']: Este é o primeiro argumento para o método crosstab. Ele especifica a variável categórica cujas categorias serão exibidas nas linhas da tabela cruzada. No caso, é a coluna 'aspecto_do_fator' do DataFrame df_fator_contribuinte.

df_fator_contribuinte['area_do_fator']: Este é o segundo argumento para o método crosstab. Ele especifica a variável categórica cujas categorias serão exibidas nas colunas da tabela cruzada. Neste caso, é a coluna 'area_do_fator' do DataFrame df_fator_contribuinte.

A tabela resultante mostrará a contagem de ocorrências para cada combinação de categorias das variáveis 'aspecto_do_fator' e 'area_do_fator'. Cada célula da tabela representa a contagem de casos onde uma determinada categoria de 'aspecto_do_fator' se encontra com uma categoria específica de 'area_do_fator'.

Por exemplo, se 'aspecto_do_fator' tem categorias A, B e C, e 'area_do_fator' tem categorias X, Y e Z, a tabela mostrará quantos casos têm A e X, A e Y, A e Z, B e X, B e Y, B e Z, C e X, C e Y, C e Z.

Essa análise é útil para entender a distribuição conjunta das categorias dessas variáveis e identificar possíveis padrões ou relações entre elas.

In [63]:
```
# 12 Relação aspecto_do_fator x fator_condicionante
pd.crosstab(df_fator_contribuinte['aspecto_do_fator'], df_fator_contribuinte['fator_condicionante'])
```

Out[63]:

fator_condicionante		INDIVIDUAL	MANUTENÇÃO DA AERONAVE	OPERAÇÃO DA AERONAVE	ORGANIZACIONAL	PRESTAÇÃO DE SERVIÇOS DE TRÁFEGO AÉREO	PSICOSSOCIAL
aspecto_do_fator							
ASPECTO DE FABRICAÇÃO	12	0	0	0	0	0	0
ASPECTO DE MANUSEIO DO MATERIAL	5	0	0	0	0	0	0
ASPECTO DE PROJETO	16	0	0	0	0	0	0
ASPECTO MÉDICO	88	0	0	0	0	0	0
ASPECTO PSICOLÓGICO	0	973	0	0	410	0	147
DESEMPENHO DO SER HUMANO	0	0	274	2454	0	29	0
ELEMENTOS RELACIONADOS AO AMBIENTE OPERACIONAL	137	0	0	0	0	0	0
ERGONOMIA	14	0	0	0	0	0	0
INFRAESTRUTURA AEROPORTUÁRIA	91	0	0	0	0	0	0
INFRAESTRUTURA DE TRÁFEGO AÉREO	10	0	0	0	0	0	0
OUTRO	116	0	0	0	0	0	0

pd.crosstab: Como mencionado anteriormente, este é um método do pandas que cria uma tabela de frequência cruzada.

df_fator_contribuinte['aspecto_do_fator']: Este é o primeiro argumento do método crosstab e especifica a variável categórica cujas categorias serão exibidas nas linhas da tabela cruzada. No caso, é a coluna 'aspecto_do_fator' do DataFrame df_fator_contribuinte.

df_fator_contribuinte['fator_condicionante']: Este é o segundo argumento do método crosstab e especifica a variável categórica cujas categorias serão exibidas nas colunas da tabela cruzada. Aqui, é a coluna 'fator_condicionante' do DataFrame df_fator_contribuinte.

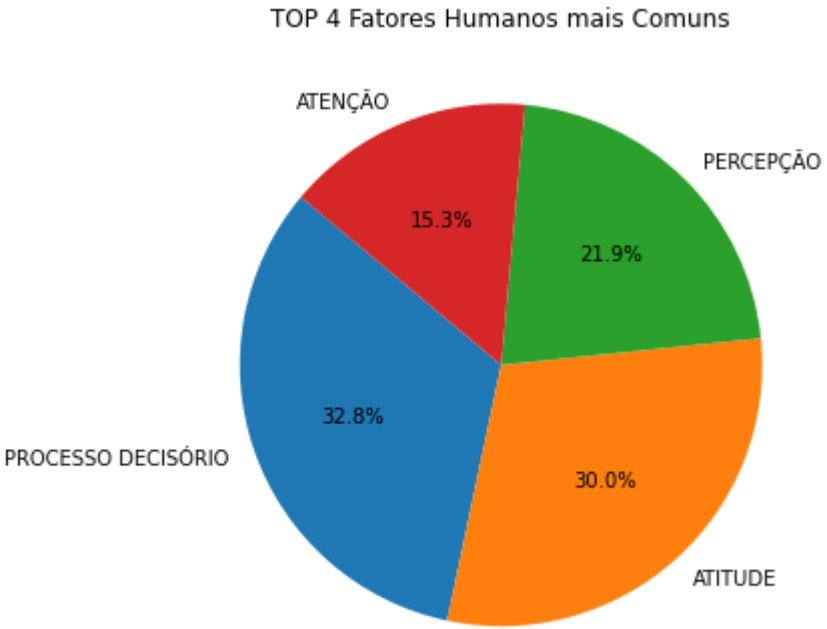
A tabela resultante mostrará a contagem de ocorrências para cada combinação de categorias das variáveis 'aspecto_do_fator' e 'fator_condicionante'. Cada célula da tabela representa a contagem de casos onde uma determinada categoria de 'aspecto_do_fator' se encontra com uma categoria específica de 'fator_condicionante'.

Por exemplo, se 'aspecto_do_fator' tem categorias A, B e C, e 'fator_condicionante' tem categorias X, Y e Z, a tabela mostrará quantos casos têm A e X, A e Y, A e Z, B e X, B e Y, B e Z, C e X, C e Y, C e Z.

Essa análise é útil para entender a distribuição conjunta das categorias dessas variáveis e identificar possíveis padrões ou relações entre

In [64]:

```
#13 Contabilização de Fatores Humanos
# Filtra as linhas onde 'area_do_fator' é igual a 'Fator_humano'
filtro_fator_humano = df_fator_contribuinte['area_do_fator'] == 'FATOR HUMANO'

# Contagem de 'nome_do_fator' para 'Fator_humano'
contagem_fator_humano = df_fator_contribuinte[filtro_fator_humano]['nome_do_fator'].value_counts()

# Porcentagem de 'nome_do_fator' para 'Fator_humano'
porcentagem_fator_humano = (contagem_fator_humano / contagem_fator_humano.sum()) * 100

# Crie um DataFrame com as contagens e porcentagens
df_resultado = pd.DataFrame({'Contagem': contagem_fator_humano, 'Porcentagem (%)': porcentagem_fator_humano})

# Seleciona apenas o TOP 4
top_4_fatores = contagem_fator_humano.head(4)

# Gráfico de pizza
plt.figure(figsize=(6, 6))  # Defina o tamanho do gráfico
plt.pie(top_4_fatores, labels=top_4_fatores.index, autopct='%1.1f%%', startangle=140)

# Título do gráfico
plt.title('TOP 4 Fatores Humanos mais Comuns')

# Exiba o gráfico
plt.show()

# Exibe o DataFrame resultante
print(df_resultado)
```


TOP 4 Fatores Humanos mais Comuns
ATENÇÃO
PERCEPÇÃO
15.3%
21.9%
32.8%
PROCESSO DECISÓRIO
30.0%
ATITUDE

	Contagem	Porcentagem (%)
PROCESSO DECISÓRIO	265	16.237745
ATITUDE	242	14.828431
PERCEPÇÃO	177	10.845588
ATENÇÃO	124	7.598039
PROCESSOS ORGANIZACIONAIS	99	6.066176
CAPACITAÇÃO E TREINAMENTO	95	5.821078
SISTEMAS DE APOIO	87	5.330882
CULTURA ORGANIZACIONAL	64	3.921569
MEMÓRIA	59	3.615196
MOTIVAÇÃO	55	3.370098
CULTURA DO GRUPO DE TRABALHO	45	2.757353
COMUNICAÇÃO	40	2.450980
ORGANIZAÇÃO DO TRABALHO	40	2.450980
ESTADO EMOCIONAL	37	2.267157
DESORIENTAÇÃO	31	1.899510
DINÂMICA DE EQUIPE	28	1.715686
RELAÇÕES INTERPESSOAIS	21	1.286765
CARACTERÍSTICAS DA TAREFA	18	1.102941
INDÍCIOS DE ESTRESSE	14	0.857843
FADIGA	12	0.735294
ILUSÕES VISUAIS	10	0.612745
CONDIÇÕES FÍSICAS DO TRABALHO	9	0.551471
INFLUÊNCIAS EXTERNAS	9	0.551471
CLIMA ORGANIZACIONAL	7	0.428922
INCONSCIÊNCIA	7	0.428922
LIDERANÇA	4	0.245098
EQUIPAMENTO - CARACTERÍSTICAS ERGONÔMICAS	4	0.245098
USO DE MEDICAMENTO	4	0.245098
ÁLCOOL	4	0.245098
ANSIEDADE	4	0.245098
SOBRECARGA DE TAREFAS	3	0.183824
DIETA INADEQUADA	2	0.122549
USO ILÍCITO DE DROGAS	2	0.122549
OBESIDADE	2	0.122549
ENFERMIDADE	2	0.122549
INSÔNIA	2	0.122549
DOR	1	0.061275
HIPÓXIA	1	0.061275
VESTIMENTA INADEQUADA	1	0.061275
INTOXICAÇÃO ALIMENTAR	1	0.061275

filtro_fator_humano = df_fator_contribuinte['area_do_fator'] == 'FATOR HUMANO': Está sendo criado um filtro para selecionar apenas as linhas em que a coluna 'area_do_fator' é igual a 'FATOR HUMANO' no DataFrame df_fator_contribuinte.

contagem_fator_humano = df_fator_contribuinte[filtro_fator_humano]['nome_do_fator'].value_counts(): Essa linha conta a quantidade de ocorrências de cada valor na coluna 'nome_do_fator' do subconjunto de dados filtrados anteriormente.

porcentagem_fator_humano = (contagem_fator_humano / contagem_fator_humano.sum()) * 100: Calcula a porcentagem de cada fator humano em relação ao total de fatores humanos.

df_resultado = pd.DataFrame({'Contagem': contagem_fator_humano, 'Porcentagem (%)': porcentagem_fator_humano}): Combina as contagens e porcentagens em um novo DataFrame chamado df_resultado.

top_4_fatores = contagem_fator_humano.head(4): Seleciona os quatro fatores humanos mais comuns.

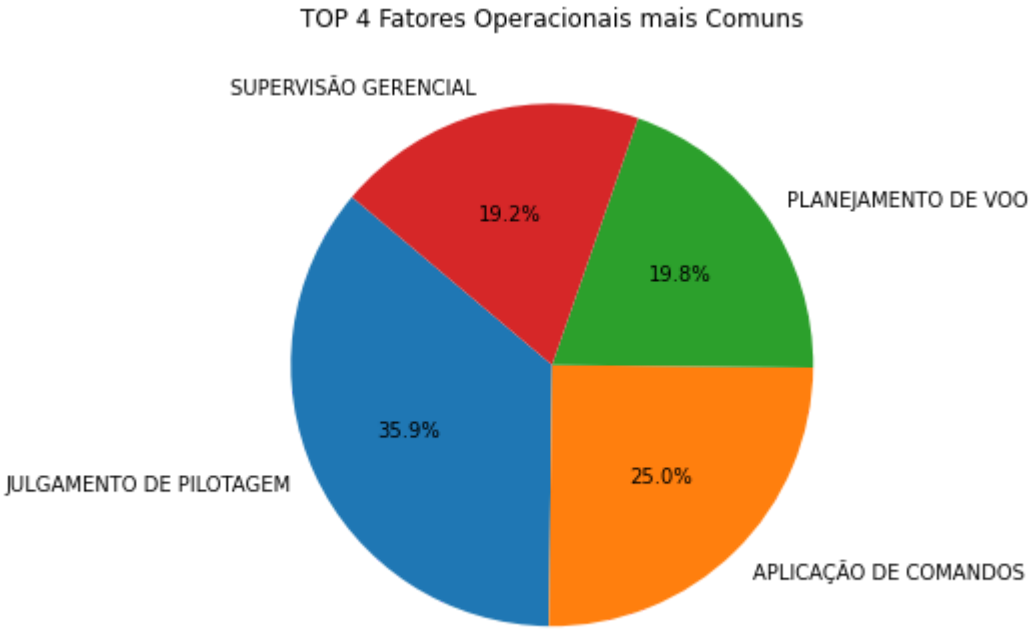
plt.figure(figsize=(6, 6)) plt.pie(top_4_fatores, labels=top_4_fatores.index, autopct='%1.1f%%', startangle=140) plt.title('TOP 4 Fatores Humanos mais Comuns') plt.show(): Cria um gráfico de pizza com as informações dos quatro fatores mais comuns. O parâmetro autopct exibe a porcentagem em cada fatia.

print(df_resultado): Exibe o DataFrame df_resultado que contém as contagens e porcentagens de cada fator humano.

O código fornece uma análise visual e tabular dos fatores humanos mais comuns, destacando os quatro principais por meio de um gráfico de pizza.

In [65]:

```
#14 Contabilização de Fatores Operacionais

# Filtra as linhas onde 'area_do_fator' é igual a 'Fator Operacional'
filtro_fator_operacional = df_fator_contribuinte['area_do_fator'] == 'FATOR OPERACIONAL'

# Contagem de 'nome_do_fator' para 'Fator Operacional'
contagem_fator_operacional = df_fator_contribuinte[filtro_fator_operacional]['nome_do_fator'].value_counts()

# Porcentagem de 'nome_do_fator' para 'Fator Operacional'
porcentagem_fator_operacional = (contagem_fator_operacional / contagem_fator_operacional.sum()) * 100

# Crie um DataFrame com as contagens e porcentagens
df_resultado = pd.DataFrame({'Contagem': contagem_fator_operacional, 'Porcentagem (%)': porcentagem_fator_operacio

# Seleciona apenas o TOP 4
top_4_fatores_operacionais = contagem_fator_operacional.head(4)

# Gráfico de pizza
plt.figure(figsize=(6, 6))  # Defina o tamanho do gráfico
plt.pie(top_4_fatores_operacionais, labels=top_4_fatores_operacionais.index, autopct='%1.1f%%', startangle=140)

# Defina o título do gráfico
plt.title('TOP 4 Fatores Operacionais mais Comuns')

# Exiba o gráfico
plt.show()

# Exibe o DataFrame resultante
print(df_resultado)
```


TOP 4 Fatores Operacionais mais Comuns
SUPERVISÃO GERENCIAL
19.2%
PLANEJAMENTO DE VOO
19.8%
35.9%
JULGAMENTO DE PILOTAGEM
25.0%
APLICAÇÃO DE COMANDOS

	Contagem	Porcentagem (%)
JULGAMENTO DE PILOTAGEM	633	21.135225
APLICAÇÃO DE COMANDOS	441	14.724541
PLANEJAMENTO DE VOO	349	11.652755
SUPERVISÃO GERENCIAL	339	11.318865
MANUTENÇÃO DA AERONAVE	274	9.148581
POUCA EXPERIÊNCIA DO PILOTO	200	6.677796
INDISCIPLINA DE VOO	136	4.540902
CONDIÇÕES METEOROLÓGICAS ADVERSAS	128	4.273790
INSTRUÇÃO	112	3.739566
COORDENAÇÃO DE CABINE	97	3.238731
INFRAESTRUTURA AEROPORTUÁRIA	91	3.038397
ESQUECIMENTO DO PILOTO	60	2.003339
PLANEJAMENTO GERENCIAL	58	1.936561
PESSOAL DE APOIO	17	0.567613
FRASEOLOGIA DO ÓRGÃO ATS	10	0.333890
PUBLICAÇÕES (ATS)	8	0.267112
PRESENÇA DE FAUNA (NÃO AVE)	5	0.166945
DESVIO DE NAVEGAÇÃO	5	0.166945
COORDENAÇÃO DE TRÁFEGO (ATS)	5	0.166945
FRASEOLOGIA DA TRIPULAÇÃO	5	0.166945
CONHECIMENTO DE NORMAS (ATS)	4	0.133556
PRESENÇA DE AVE	4	0.133556
PLANEJAMENTO DE TRÁFEGO (ATS)	4	0.133556
SUPERVISÃO (ATS)	3	0.100167
EQUIPAMENTO DE APOIO (ATS)	2	0.066778
LIMITE DE AUTORIZAÇÃO	2	0.066778
EMPREGO DE MEIOS (ATS)	1	0.033389
HABILIDADE DE CONTROLE (ATS)	1	0.033389
SUBSTITUIÇÃO NA POSIÇÃO (ATS)	1	0.033389

filtro_fator_operacional = df_fator_contribuinte['area_do_fator'] == 'FATOR OPERACIONAL': Cria um filtro para selecionar apenas as linhas em que a coluna 'area_do_fator' é igual a 'FATOR OPERACIONAL' no DataFrame df_fator_contribuinte.

contagem_fator_operacional = df_fator_contribuinte[filtro_fator_operacional]['nome_do_fator'].value_counts(): Conta a quantidade de ocorrências de cada valor na coluna 'nome_do_fator' do subconjunto de dados filtrados.

porcentagem_fator_operacional = (contagem_fator_operacional / contagem_fator_operacional.sum()) * 100: Calcula a porcentagem de cada fator operacional em relação ao total de fatores operacionais.

df_resultado = pd.DataFrame({'Contagem': contagem_fator_operacional, 'Porcentagem (%)': porcentagem_fator_operacional}): Combina as contagens e porcentagens em um novo DataFrame chamado df_resultado.

top_4_fatores_operacionais = contagem_fator_operacional.head(4): Seleciona os quatro fatores operacionais mais comuns.

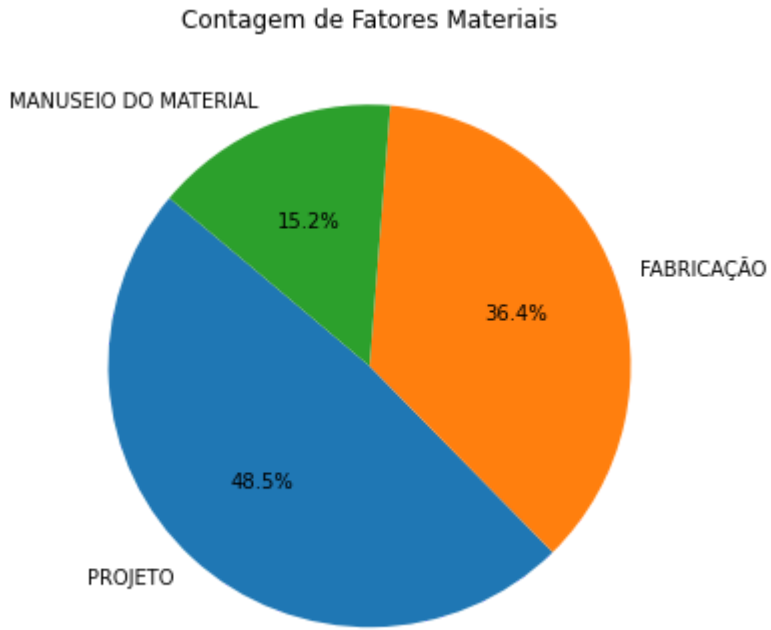
plt.figure(figsize=(6, 6)) plt.pie(top_4_fatores_operacionais, labels=top_4_fatores_operacionais.index, autopct='%1.1f%%', startangle=140) plt.title('TOP 4 Fatores Operacionais mais Comuns') plt.show(): Cria um gráfico de pizza com as informações dos quatro fatores operacionais mais comuns. O parâmetro autopct exibe a porcentagem em cada fatia.

print(df_resultado): Exibe o DataFrame df_resultado que contém as contagens e porcentagens de cada fator operacional.

Assim como o código anterior, este proporciona uma análise visual e tabular dos fatores operacionais mais comuns, destacando os quatro principais por meio de um gráfico de pizza.

In [66]:

```
#15  Contabilização de Fatores Materiais

# Filtra as linhas onde 'area_do_fator' é igual a 'Fator_Material'
filtro_fator_material = df_fator_contribuinte['area_do_fator'] == 'FATOR MATERIAL'

# Contagem de 'nome_do_fator' para 'Fator_Material'
contagem_fator_material = df_fator_contribuinte[filtro_fator_material]['nome_do_fator'].value_counts()

# Porcentagem de 'nome_do_fator' para 'Fator_Material'
porcentagem_fator_material = (contagem_fator_material / contagem_fator_material.sum()) * 100

# Crie um DataFrame com as contagens e porcentagens
df_resultado = pd.DataFrame({'Contagem': contagem_fator_material, 'Porcentagem (%)': porcentagem_fator_material})

# Gráfico de pizza
plt.figure(figsize=(6, 6))  # Defina o tamanho do gráfico
plt.pie(contagem_fator_material, labels=contagem_fator_material.index, autopct='%1.1f%%', startangle=140)

# Título do gráfico
plt.title('Contagem de Fatores Materiais')


plt.show()
# Exibe o DataFrame resultante
print(df_resultado)
```



	Contagem	Porcentagem (%)
PROJETO	16	48.484848
FABRICAÇÃO	12	36.363636
MANUSEIO DO MATERIAL	5	15.151515

filtro_fator_material = df_fator_contribuinte['area_do_fator'] == 'FATOR MATERIAL': Cria um filtro para selecionar apenas as linhas em que a coluna 'area_do_fator' é igual a 'FATOR MATERIAL' no DataFrame df_fator_contribuinte.

contagem_fator_material = df_fator_contribuinte[filtro_fator_material]['nome_do_fator'].value_counts(): Conta a quantidade de ocorrências de cada valor na coluna 'nome_do_fator' do subconjunto de dados filtrados.

porcentagem_fator_material = (contagem_fator_material / contagem_fator_material.sum()) * 100: Calcula a porcentagem de cada fator material em relação ao total de fatores materiais.

df_resultado = pd.DataFrame({'Contagem': contagem_fator_material, 'Porcentagem (%)': porcentagem_fator_material}): Combina as contagens e porcentagens em um novo DataFrame chamado df_resultado.

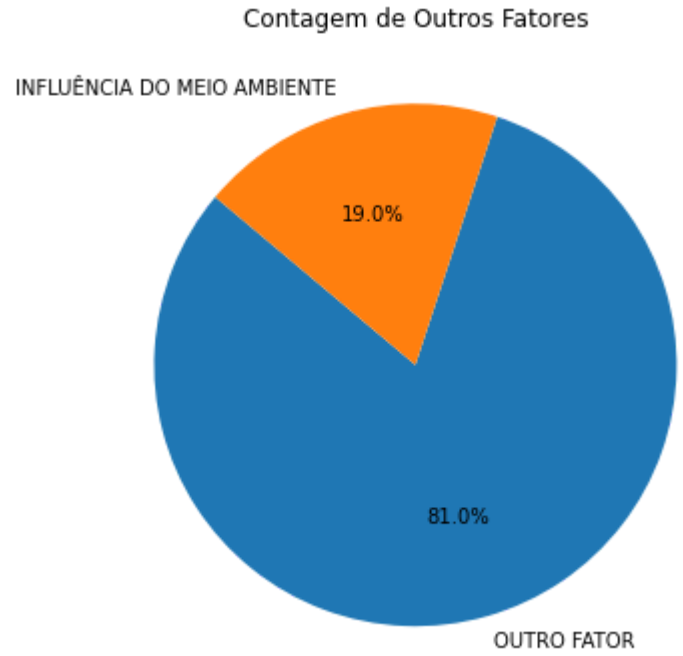
plt.figure(figsize=(6, 6)) plt.pie(contagem_fator_material, labels=contagem_fator_material.index, autopct='%1.1f%%', startangle=140) plt.title('Contagem de Fatores Materiais') plt.show(): Cria um gráfico de pizza com as informações da contagem de fatores materiais. O parâmetro autopct exibe a porcentagem em cada fatia.

print(df_resultado): Exibe o DataFrame df_resultado que contém as contagens e porcentagens de cada fator material.

Este código fornece uma análise visual e tabular dos fatores materiais, destacando a contagem de cada fator em um gráfico de pizza e

In [67]:

```
#16 Contabilização de Outro Fator

# Filtra as linhas onde 'area_do_fator' é igual a 'Fator_outro'
filtro_fator_outro = df_fator_contribuinte['area_do_fator'] == 'OUTRO'

# Contagem de 'nome_do_fator' para 'Fator_outro'
contagem_fator_outro = df_fator_contribuinte[filtro_fator_outro]['nome_do_fator'].value_counts()

# Porcentagem de 'nome_do_fator' para 'Fator_outro'
porcentagem_fator_outro = (contagem_fator_outro / contagem_fator_outro.sum()) * 100

# Crie um DataFrame com as contagens e porcentagens
df_resultado = pd.DataFrame({'Contagem': contagem_fator_outro, 'Porcentagem (%)': porcentagem_fator_outro})

# Gráfico de pizza
plt.figure(figsize=(6, 6))  # Defina o tamanho do gráfico
plt.pie(contagem_fator_outro, labels=contagem_fator_outro.index, autopct='%1.1f%%', startangle=140)

# Defina o título do gráfico
plt.title('Contagem de Outros Fatores')

# Exiba o gráfico
plt.show()

# Exibe o DataFrame resultante
print(df_resultado)
```



	Contagem	Porcentagem (%)
OUTRO FATOR	94	81.034483
INFLUÊNCIA DO MEIO AMBIENTE	22	18.965517

filtro_fator_outro = df_fator_contribuinte['area_do_fator'] == 'OUTRO': Cria um filtro para selecionar apenas as linhas em que a coluna 'area_do_fator' é igual a 'OUTRO' no DataFrame df_fator_contribuinte.

contagem_fator_outro = df_fator_contribuinte[filtro_fator_outro]['nome_do_fator'].value_counts(): Conta a quantidade de ocorrências de cada valor na coluna 'nome_do_fator' do subconjunto de dados filtrados.

porcentagem_fator_outro = (contagem_fator_outro / contagem_fator_outro.sum()) * 100: Calcula a porcentagem de cada fator 'OUTRO' em relação ao total de fatores 'OUTRO'.

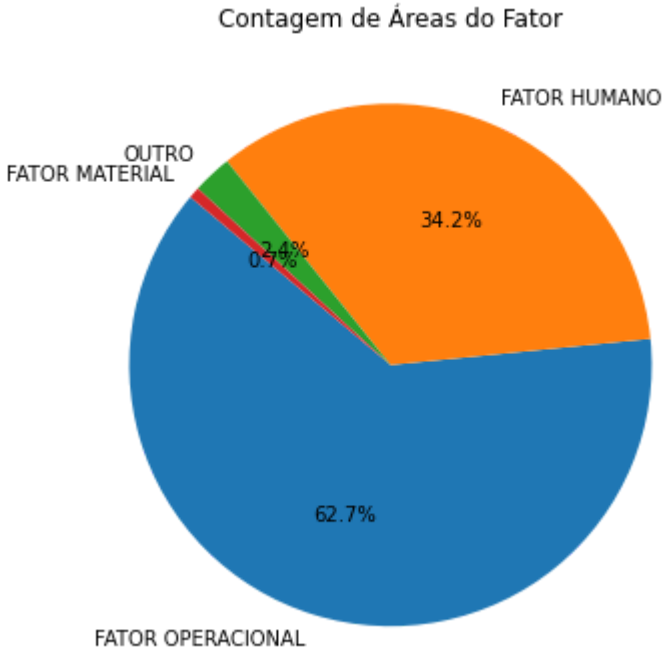
df_resultado = pd.DataFrame({'Contagem': contagem_fator_outro, 'Porcentagem (%)': porcentagem_fator_outro}): Combina as contagens e porcentagens em um novo DataFrame chamado df_resultado.

plt.figure(figsize=(6, 6)) plt.pie(contagem_fator_outro, labels=contagem_fator_outro.index, autopct='%1.1f%%', startangle=140) plt.title('Contagem de Outros Fatores') plt.show(): Cria um gráfico de pizza com as informações da contagem de fatores 'OUTRO'. O parâmetro autopct exibe a porcentagem em cada fatia.

print(df_resultado): Exibe o DataFrame df_resultado que contém as contagens e porcentagens de cada fator 'OUTRO'.

In [68]:

```
#17 Contagem de Area dos Fatores

# Contagem de 'area_do_fator' e cálculo da porcentagem
contagem_area_do_fator = df_fator_contribuinte['area_do_fator'].value_counts()
porcentagem_area_do_fator = (contagem_area_do_fator / contagem_area_do_fator.sum()) * 100

# Crie um DataFrame com as contagens e porcentagens
df_resultado = pd.DataFrame({'Contagem': contagem_area_do_fator, 'Porcentagem (%)': porcentagem_area_do_fator})

# Gráfico de pizza
plt.figure(figsize=(6, 6))  # Defina o tamanho do gráfico
plt.pie(porcentagem_area_do_fator, labels=contagem_area_do_fator.index, autopct='%1.1f%%', startangle=140)

# Defina o título do gráfico
plt.title('Contagem de Áreas do Fator')

# Exiba o gráfico
plt.show()

# Exibe o DataFrame resultante
print(df_resultado)
```



	Contagem	Porcentagem (%)
FATOR OPERACIONAL	2995	62.709380
FATOR HUMANO	1632	34.170854
OUTRO	116	2.428811
FATOR MATERIAL	33	0.690955

contagem_area_do_fator = df_fator_contribuinte['area_do_fator'].value_counts(): Conta a quantidade de ocorrências de cada valor na coluna 'area_do_fator' do DataFrame df_fator_contribuinte.

porcentagem_area_do_fator = (contagem_area_do_fator / contagem_area_do_fator.sum()) * 100: Calcula a porcentagem de cada área em relação ao total de áreas de fatores.

df_resultado = pd.DataFrame({'Contagem': contagem_area_do_fator, 'Porcentagem (%)': porcentagem_area_do_fator}): Combina as contagens e porcentagens em um novo DataFrame chamado df_resultado.

plt.figure(figsize=(6, 6)) plt.pie(porcentagem_area_do_fator, labels=contagem_area_do_fator.index, autopct='%1.1f%%', startangle=140) plt.title('Contagem de Áreas do Fator') plt.show(): Cria um gráfico de pizza com as informações da contagem de áreas de fatores. O parâmetro autopct exibe a porcentagem em cada fatia.

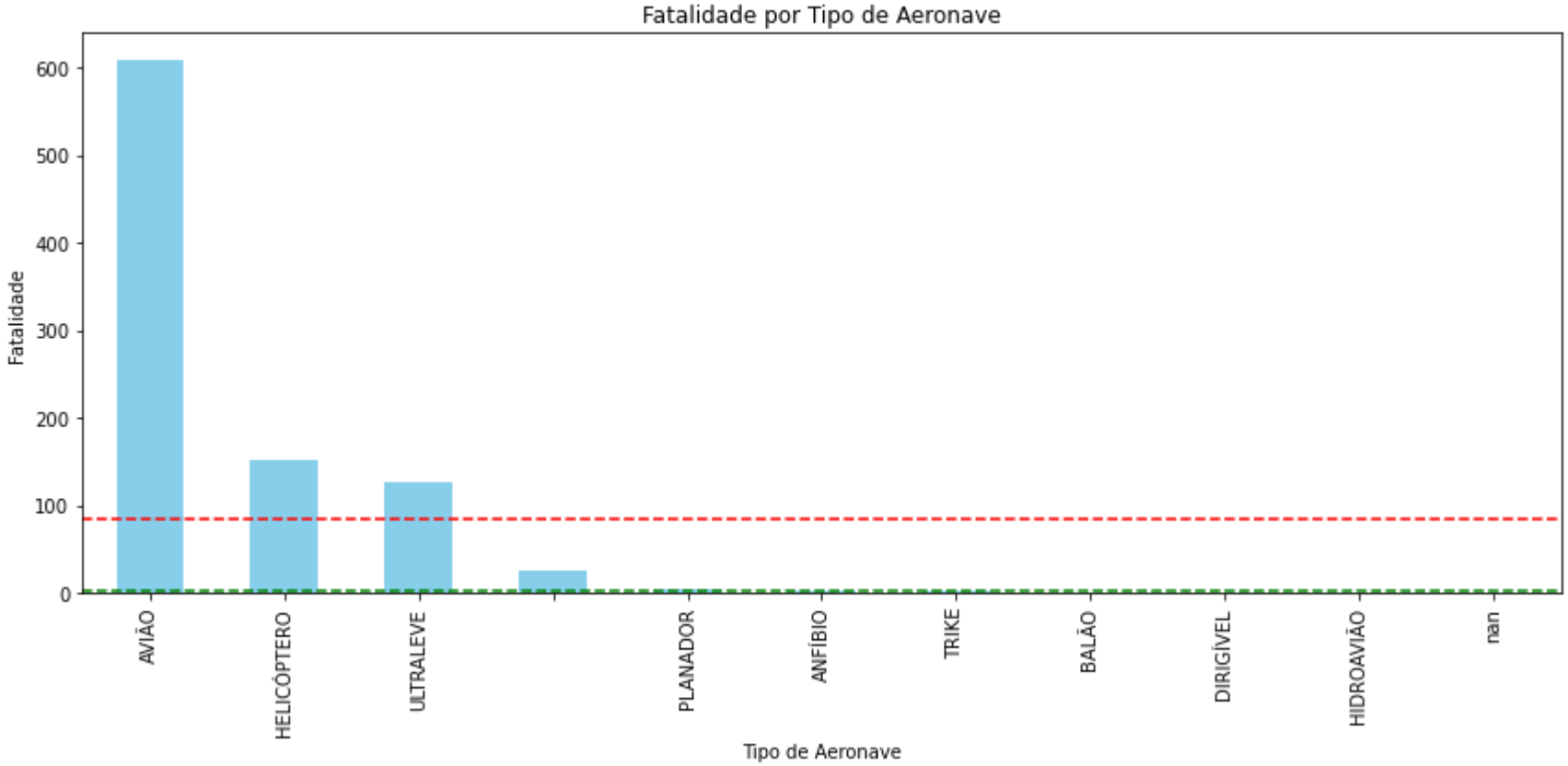
print(df_resultado): Exibe o DataFrame df_resultado que contém as contagens e porcentagens de cada área de fator.

Este código fornece uma análise visual e tabular da distribuição das diferentes áreas de fatores, destacando a contagem e a porcentagem de cada área em um gráfico de pizza e apresentando as informações detalhadas em um DataFrame.

Aeronave

In [69]:

```
# 18 Fatalidade por Tipo de Aeronave

#Agrupando e somando os Valores
fatalidades_por_tipo = df_aeronave.groupby('tipo_da_aeronave')['fatalidade'].sum()

#Ordena os resultados em ordem decrescente
fatalidades_por_tipo = fatalidades_por_tipo.sort_values(ascending=False)

# Calcula a média e mediana das fatalidades
media_fatalidades = fatalidades_por_tipo.mean()
mediana_fatalidades = fatalidades_por_tipo.median()

#Gráfico de Barras
plt.figure(figsize=(12,6))
fatalidades_por_tipo.plot(kind='bar', color='skyblue')
plt.axhline(y=media_fatalidades, color='red', linestyle='--', label=f'Média ({media_fatalidades:.2f})')
plt.axhline(y=mediana_fatalidades, color='green', linestyle='--', label=f'Mediana ({mediana_fatalidades:.2f})')
plt.title('Fatalidade por Tipo de Aeronave')
plt.xlabel('Tipo de Aeronave')
plt.ylabel('Fatalidade')
plt.xticks(rotation=90)
plt.tight_layout()

#Gráfico
plt.show()

print(f" A Mediana de Fatalidades por Tipo de Aeronave é {mediana_fatalidades} e a Média é {media_fatalidades}")

print(fatalidades_por_tipo)
```




```
 A Mediana de Fatalidades por Tipo de Aeronave é 3.0 e a Média é 83.9090909090909
tipo_da_aeronave
AVIÃO          609
HELICÓPTERO    152
ULTRALEVE      127
                25
PLANADOR         4
ANFÍBIO          3
TRIKE            3
BALÃO            0
DIRIGÍVEL        0
HIDROAVIÃO       0
nan              0
Name: fatalidade, dtype: int32
```

fatalidades_por_tipo = df_aeronave.groupby('tipo_da_aeronave')['fatalidade'].sum(): Usa o método groupby para agrupar os dados pelo tipo de aeronave e, em seguida, soma as fatalidades para cada grupo.

fatalidades_por_tipo = fatalidades_por_tipo.sort_values(ascending=False): Ordena os resultados em ordem decrescente, para que os tipos de aeronave com o maior número de fatalidades apareçam no topo.

media_fatalidades = fatalidades_por_tipo.mean() mediana_fatalidades = fatalidades_por_tipo.median(): Calcula a média e a mediana das fatalidades para os diferentes tipos de aeronave.

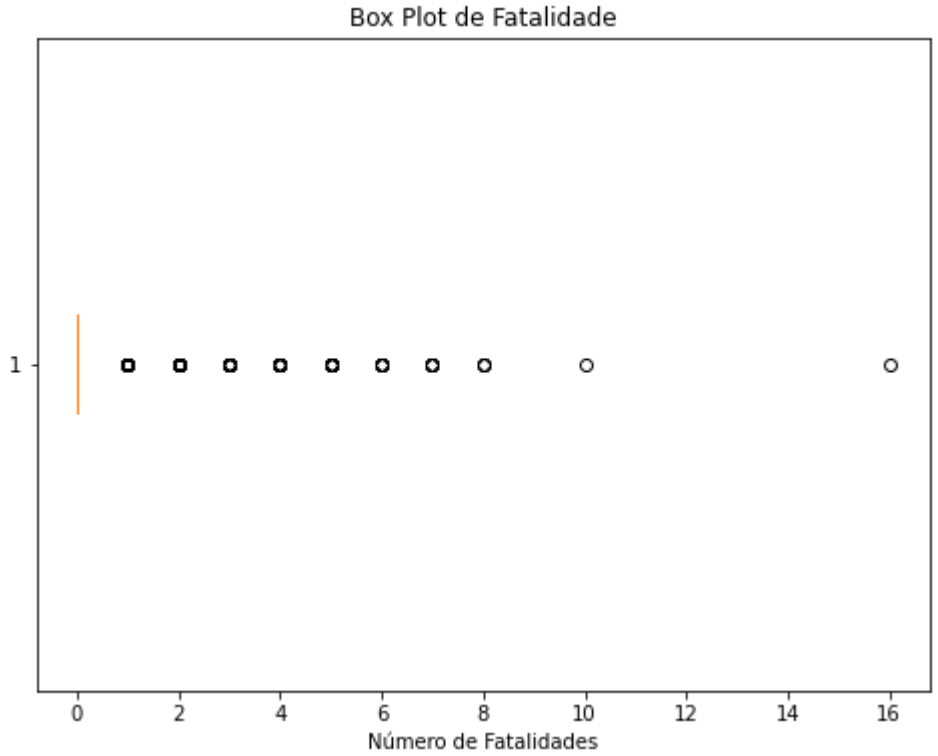
plt.figure(figsize=(12,6)) fatalidades_por_tipo.plot(kind='bar', color='skyblue') plt.axhline(y=media_fatalidades, color='red', linestyle='--', label=f'Média ({media_fatalidades:.2f})') plt.axhline(y=mediana_fatalidades, color='green', linestyle='--', label=f'Mediana ({mediana_fatalidades:.2f})') plt.title('Fatalidade por Tipo de Aeronave') plt.xlabel('Tipo de Aeronave') plt.ylabel('Fatalidade') plt.xticks(rotation=90) plt.tight_layout(): Cria um gráfico de barras mostrando a quantidade de fatalidades para cada tipo de aeronave. Adiciona linhas horizontais para representar a média e a mediana.

plt.show(): Exibe o gráfico

print(f" A Mediana de Fatalidades por Tipo de Aeronave é {mediana_fatalidades} e a Média é {media_fatalidades}") print(fatalidades_por_tipo): Exibe informações adicionais, incluindo a mediana e a média das fatalidades por tipo de aeronave, bem como a contagem de fatalidades para cada tipo.

Este código fornece uma visualização clara da distribuição das fatalidades por tipo de aeronave, destacando a média e a mediana por meio de linhas no gráfico de barras e apresentando as informações detalhadas na saída do console.

In [70]:
```
#19 Verficação de Outliers para Fatalidade

# Box Plot
plt.figure(figsize=(8, 6))  # Ajusta o tamanho do gráfico
plt.boxplot(df_aeronave['fatalidade'], vert=False)  # vert=False para ter o box plot horizontal
plt.title('Box Plot de Fatalidade')
plt.xlabel('Número de Fatalidades')

# Exibe o gráfico
plt.show()
```



plt.figure(figsize=(8, 6)): Ajusta o tamanho do gráfico para 8 unidades de largura por 6 unidades de altura.

plt.boxplot(df_aeronave['fatalidade'], vert=False): Gera um box plot horizontal para a variável 'fatalidade' do DataFrame df_aeronave. O parâmetro vert=False é utilizado para que o box plot seja desenhado horizontalmente.

plt.title('Box Plot de Fatalidade') plt.xlabel('Número de Fatalidades'): Adiciona um título ao gráfico e rótulo ao eixo x para indicar que o box plot está representando a distribuição da variável 'fatalidade'.

plt.show(): Exibe o gráfico de box plot.

O box plot é uma ferramenta útil para identificar a presença de outliers e para entender a distribuição estatística de uma variável. Ele mostra a mediana (linha no meio do retângulo), os quartis (caixa), e potenciais outliers (pontos fora das "whiskers" - linhas que se estendem dos quartis).

In [71]:
```
#20 Quantidade de Classificações das Ocorrencias por Tipo da Aeronave

# Mescla dos dois DataFrames com base na coluna 'codigo_ocorrencia2'
merged_df = df_ocorrencia.merge(df_aeronave, on='codigo_ocorrencia2')

# Agrupamento dos dados por 'tipo_da_aeronave' e 'classificacao' e contagem da quantidade de ocorrências
resultado = merged_df.groupby(['tipo_da_aeronave', 'classificacao']).size().unstack(fill_value=0)

# Ordene as colunas de acordo com o número de acidentes (do maior para o menor)
resultado = resultado.sort_values(by='ACIDENTE', ascending=False)

# Exibe o resultado
print(resultado)

```

```
classificacao     ACIDENTE  INCIDENTE  INCIDENTE GRAVE
tipo_da_aeronave
AVIÃO                 1422       2935              703
ULTRALEVE              245         52               54
HELICÓPTERO            233        379               70
                        49        102               17
PLANADOR                13          5                3
nan                     12         20                1
ANFÍBIO                  7          8                0
TRIKE                    5          0                0
BALÃO                    1          0                0
DIRIGÍVEL                0          2                0
HIDROAVIÃO               0          1                0
```

merged_df = df_ocorrencia.merge(df_aeronave, on='codigo_ocorrencia2'): Combina os DataFrames df_ocorrencia e df_aeronave com base na coluna 'codigo_ocorrencia2'. Isso cria um novo DataFrame chamado merged_df que contém as informações de ambos os DataFrames.

resultado = merged_df.groupby(['tipo_da_aeronave', 'classificacao']).size().unstack(fill_value=0): Agrupa os dados por 'tipo_da_aeronave' e 'classificacao' e conta a quantidade de ocorrências para cada combinação. O método unstack transforma os resultados em um formato de tabela, e fill_value=0 preenche valores ausentes com zero.

resultado = resultado.sort_values(by='ACIDENTE', ascending=False): Ordena as colunas do DataFrame com base no número de acidentes ('ACIDENTE') em ordem decrescente.

print(resultado): Exibe o resultado final, que é a tabela com a contagem de ocorrências para cada tipo de aeronave e classificação, ordenadas pelo número de acidentes.

Este código fornece uma visão tabular da quantidade de ocorrências classificadas para cada tipo de aeronave. O resultado mostra a distribuição das classificações de ocorrências (como INCIDENTE, INCIDENTE GRAVE, ACIDENTE, etc.) para diferentes tipos de aeronaves, destacando as

Criação de Modelos de Machine Learning

In [49]:

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler

# Mescla os dois DataFrames com base na coluna 'codigo_ocorrencia2'
merged_df = df_ocorrencia.merge(df_aeronave, on='codigo_ocorrencia2')

# Escolhe as características (features) relevantes
features = ['fatalidade']

# Loop sobre as diferentes classes de 'classificacao'
for classificacao in ['ACIDENTE', 'INCIDENTE', 'INCIDENTE GRAVE']:
    print(f"\n\n{'='*20} Modelo para '{classificacao}' {'='*20}")

    # Cria uma coluna 'target' para a classe atual
    y = (merged_df['classificacao'] == classificacao).astype(int)

    # Seleciona as features relevantes
    X = merged_df[features]

    # Divide os dados em conjunto de treinamento e teste
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Padroniza os dados (opcional, mas geralmente recomendado para regressão logística)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Define os parâmetros a serem testados
    param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

    # Cria o modelo de Regressão Logística
    model = LogisticRegression()

    # Realiza a busca em grade para encontrar os melhores parâmetros
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train, y_train)

    # Obtém o melhor modelo com os melhores parâmetros
    best_model = grid_search.best_estimator_
```

```
42
43      # Realiza previsões no conjunto de teste
44      y_pred = best_model.predict(X_test)
45
46      # Avalia o desempenho do modelo
47      accuracy = accuracy_score(y_test, y_pred)
48      conf_matrix = confusion_matrix(y_test, y_pred)
49      classification_rep = classification_report(y_test, y_pred)
50
51      # Exibe os resultados
52      print(f"Melhores Parâmetros: {grid_search.best_params_}")
53      print(f"Acurácia para '{classificacao}': {accuracy:.2f}")
54      print("Matriz de Confusão:")
55      print(conf_matrix)
56      print("Relatório de Classificação:")
57      print(classification_rep)
```

```
==================== Modelo para 'ACIDENTE' ====================
Melhores Parâmetros: {'C': 0.01}
Acurácia para 'ACIDENTE': 0.75
Matriz de Confusão:
[[867   0]
 [317  84]]
Relatório de Classificação:
              precision    recall  f1-score   support

           0       0.73      1.00      0.85       867
           1       1.00      0.21      0.35       401

    accuracy                           0.75      1268
   macro avg       0.87      0.60      0.60      1268
weighted avg       0.82      0.75      0.69      1268



==================== Modelo para 'INCIDENTE' ====================
Melhores Parâmetros: {'C': 0.001}
Acurácia para 'INCIDENTE': 0.62
Matriz de Confusão:
[[ 84 481]
 [  0 703]]
Relatório de Classificação:
              precision    recall  f1-score   support

           0       1.00      0.15      0.26       565
           1       0.59      1.00      0.75       703

    accuracy                           0.62      1268
   macro avg       0.80      0.57      0.50      1268
weighted avg       0.77      0.62      0.53      1268



==================== Modelo para 'INCIDENTE GRAVE' ====================
Melhores Parâmetros: {'C': 0.001}
Acurácia para 'INCIDENTE GRAVE': 0.87
```

```
Matriz de Confusão:
[[1104    0]
 [ 164    0]]
Relatório de Classificação:
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      1104
           1       0.00      0.00      0.00       164

    accuracy                           0.87      1268
   macro avg       0.44      0.50      0.47      1268
weighted avg       0.76      0.87      0.81      1268


C:\Users\home\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\home\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\home\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

from sklearn.model_selection import train_test_split, GridSearchCV from sklearn.linear_model import LogisticRegression from sklearn.metrics import accuracy_score, classification_report, confusion_matrix from sklearn.preprocessing import StandardScaler:

São importadas as bibliotecas necessárias, incluindo funções para dividir o conjunto de dados, criar um modelo de Regressão Logística, realizar busca em grade para otimização de parâmetros e avaliar o desempenho do modelo.

merged_df = df_ocorrencia.merge(df_aeronave, on='codigo_ocorrencia2'): Os DataFrames df_ocorrencia e df_aeronave são mesclados com base na coluna 'codigo_ocorrencia2', combinando informações relevantes sobre ocorrências e aeronaves.

features = ['fatalidade']: É escolhida a característica 'fatalidade' como a única feature para treinar o modelo

for classificacao in ['ACIDENTE', 'INCIDENTE', 'INCIDENTE GRAVE']: O código realiza um loop sobre as diferentes classes de 'classificacao': 'ACIDENTE', 'INCIDENTE' e 'INCIDENTE GRAVE'.

y = (merged_df['classificacao'] == classificacao).astype(int) X = merged_df[features] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42):

Os dados são preparados, criando um vetor de destino y, selecionando as features X e dividindo o conjunto de dados em conjuntos de treinamento e teste.

scaler = StandardScaler() X_train = scaler.fit_transform(X_train) X_test = scaler.transform(X_test): Os dados são padronizados usando StandardScaler, o que é recomendado para modelos de Regressão Logística.

param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}: Define-se uma grade de valores para o parâmetro de regularização 'C' a ser testado.

model = LogisticRegression(): Um modelo de Regressão Logística é criado.

grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy') grid_search.fit(X_train, y_train): Realiza-se uma busca em grade para encontrar os melhores parâmetros para o modelo, utilizando validação cruzada (cv=5) e a métrica de acurácia como critério de avaliação.

best_model = grid_search.best_estimator_ O melhor modelo com os melhores parâmetros é selecionado.

y_pred = best_model.predict(X_test) accuracy = accuracy_score(y_test, y_pred) conf_matrix = confusion_matrix(y_test, y_pred) classification_rep = classification_report(y_test, y_pred): Realiza-se previsões no conjunto de teste e avalia o desempenho do modelo calculando acurácia, matriz de confusão e relatório de classificação.

print(f"Melhores Parâmetros: {grid_search.best_params_}") print(f"Acurácia para '{classificacao}': {accuracy:.2f}") print("Matriz de Confusão:") print(conf_matrix) print("Relatório de Classificação:") print(classification_rep):

Os resultados são exibidos, incluindo os melhores parâmetros encontrados, a acurácia e outras métricas de avaliação para cada classe de 'classificacao' no loop.

Esse código é útil para ajustar os parâmetros de um modelo de Regressão Logística para diferentes classes e avaliar como o modelo performa em cada uma delas usando validação cruzada.

In [73]:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression


# Mescla os dois DataFrames com base na coluna 'codigo_ocorrencia2'
merged_df = df_ocorrencia.merge(df_aeronave, on='codigo_ocorrencia2')

# Escolhe as características (features) relevantes
features = ['fatalidade']

X = merged_df[features]

# Cria um modelo de Regressão Logística
model = LogisticRegression()

# Treina o modelo com todos os dados disponíveis
y = (merged_df['classificacao'] == 'ACIDENTE').astype(int)  # Converte para 0 ou 1
model.fit(X, y)

# Agora, você pode fazer previsões de probabilidade para a classe "ACIDENTE"
# com a feature de "fatalidade" maior que zero
proba_acidente = model.predict_proba(X)[:, 1]

# Filtra as probabilidades onde a classificação é "ACIDENTE"
filtro_acidente = (merged_df['classificacao'] == 'ACIDENTE')

# Calcula a probabilidade média de fatalidade maior que zero
probabilidade_media = proba_acidente[filtro_acidente].mean()

print(f"Probabilidade média de fatalidade maior que zero quando classificação é 'ACIDENTE': {probabilidade_media:.
```

```
Probabilidade média de fatalidade maior que zero quando classificação é 'ACIDENTE': 0.43
```

merged_df = df_ocorrencia.merge(df_aeronave, on='codigo_ocorrencia2'): Os DataFrames df_ocorrencia e df_aeronave são mesclados com base na coluna 'codigo_ocorrencia2', combinando informações relevantes sobre ocorrências e aeronaves.

features = ['fatalidade'] X = merged_df[features]: É escolhida a característica 'fatalidade' como a única feature para treinar o modelo de Regressão Logística.

model = LogisticRegression(): Um modelo de Regressão Logística é criado.

y = (merged_df['classificacao'] == 'ACIDENTE').astype(int) model.fit(X, y): O modelo de Regressão Logística é treinado utilizando todas as instâncias disponíveis nos conjuntos de dados de X (features) e y (rótulos). A variável y é criada considerando se a 'classificacao' é 'ACIDENTE' e convertendo os valores para 0 ou 1.

proba_acidente = model.predict_proba(X)[:, 1]: O modelo é usado para fazer previsões de probabilidade. predict_proba retorna as probabilidades para cada classe, e [:, 1] seleciona as probabilidades associadas à classe positiva, neste caso, 'ACIDENTE'.

filtro_acidente = (merged_df['classificacao'] == 'ACIDENTE'): Um filtro é criado para selecionar as instâncias onde a 'classificacao' é 'ACIDENTE'.

probabilidade_media = proba_acidente[filtro_acidente].mean(): Calcula-se a média das probabilidades de 'ACIDENTE' apenas para as instâncias onde a classificação é 'ACIDENTE'.

print(f"Probabilidade média de fatalidade maior que zero quando classificação é 'ACIDENTE': {probabilidade_media:.2f}"): Exibe a probabilidade média calculada.

Em resumo, o código treina um modelo de Regressão Logística para prever a probabilidade de um acidente (considerando a classificação 'ACIDENTE') com base na feature 'fatalidade' e, em seguida, calcula a probabilidade média de fatalidade maior que zero para as instâncias onde a classificação é 'ACIDENTE'.

In [74]:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

# Mescla os dois DataFrames com base na coluna 'codigo_ocorrencia2'
merged_df = df_ocorrencia.merge(df_aeronave, on='codigo_ocorrencia2')

# Escolhe as características (features) relevantes
features = ['fatalidade']

X = merged_df[features]

# Cria um modelo de Regressão Logística
model = LogisticRegression()

# Treina o modelo com todos os dados disponíveis
y = (merged_df['classificacao'] == 'INCIDENTE GRAVE').astype(int)  # Converte para 0 ou 1
model.fit(X, y)

# Agora, você pode fazer previsões de probabilidade para a classe "ACIDENTE"
# com a feature de "fatalidade" maior que zero
proba_acidente = model.predict_proba(X)[:, 1]

# Filtra as probabilidades onde a classificação é "ACIDENTE"
filtro_acidente = (merged_df['classificacao'] == 'INCIDENTE GRAVE')

# Calcula a probabilidade média de fatalidade maior que zero
probabilidade_media = proba_acidente[filtro_acidente].mean()

print(f"Probabilidade média de fatalidade maior que zero quando classificação é 'INCIDENTE GRAVE': {probabilidade_
```

```
Probabilidade média de fatalidade maior que zero quando classificação é 'INCIDENTE GRAVE': 0.14
```

merged_df = df_ocorrencia.merge(df_aeronave, on='codigo_ocorrencia2'): Mescla os DataFrames df_ocorrencia e df_aeronave com base na coluna 'codigo_ocorrencia2', combinando informações relevantes sobre ocorrências e aeronaves.

features = ['fatalidade'] X = merged_df[features]: Escolhe a característica 'fatalidade' como a única feature para treinar o modelo de Regressão Logística.

model = LogisticRegression(): Cria um modelo de Regressão Logística.

y = (merged_df['classificacao'] == 'INCIDENTE GRAVE').astype(int) model.fit(X, y): Treina o modelo de Regressão Logística usando todas as instâncias disponíveis nos conjuntos de dados X (features) e y (rótulos). A variável y é criada considerando se a 'classificacao' é 'INCIDENTE GRAVE' e convertendo os valores para 0 ou 1.

proba_acidente = model.predict_proba(X)[:, 1]: Utiliza o modelo para fazer previsões de probabilidade. predict_proba retorna as probabilidades para cada classe, e [:, 1] seleciona as probabilidades associadas à classe positiva, neste caso, 'INCIDENTE GRAVE'.

filtro_acidente = (merged_df['classificacao'] == 'INCIDENTE GRAVE'): Cria um filtro para selecionar as instâncias onde a 'classificacao' é 'INCIDENTE GRAVE'.

probabilidade_media = proba_acidente[filtro_acidente].mean(): Calcula a média das probabilidades de 'INCIDENTE GRAVE' apenas para as instâncias onde a classificação é 'INCIDENTE GRAVE'.

print(f"Probabilidade média de fatalidade maior que zero quando classificação é 'INCIDENTE GRAVE' {probabilidade_media:.2f}"): Exibe a probabilidade média calculada.

Portanto, o código treina um modelo de Regressão Logística para prever a probabilidade de um incidente grave (considerando a classificação 'INCIDENTE GRAVE') com base na feature 'fatalidade' e, em seguida, calcula a probabilidade média de fatalidade maior que zero para as instâncias onde a classificação é 'INCIDENTE GRAVE'

In [75]:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

# Mescla os dois DataFrames com base na coluna 'codigo_ocorrencia2'
merged_df = df_ocorrencia.merge(df_aeronave, on='codigo_ocorrencia2')

# Escolhe as características (features) relevantes
features = ['fatalidade']

X = merged_df[features]

# Cria um modelo de Regressão Logística
model = LogisticRegression()

# Treina o modelo com todos os dados disponíveis
y = (merged_df['classificacao'] == 'INCIDENTE').astype(int)  # Converte para 0 ou 1
model.fit(X, y)

# Agora, você pode fazer previsões de probabilidade para a classe "ACIDENTE"
# com a feature de "fatalidade" maior que zero
proba_acidente = model.predict_proba(X)[:, 1]

# Filtra as probabilidades onde a classificação é "ACIDENTE"
filtro_acidente = (merged_df['classificacao'] == 'INCIDENTE')

# Calcula a probabilidade média de fatalidade maior que zero
probabilidade_media = proba_acidente[filtro_acidente].mean()

print(f"Probabilidade média de fatalidade maior que zero quando classificação é 'INCIDENTE': {probabilidade_media:

```

```
Probabilidade média de fatalidade maior que zero quando classificação é 'INCIDENTE': 0.60
```

merged_df = df_ocorrencia.merge(df_aeronave, on='codigo_ocorrencia2'): Mescla os DataFrames df_ocorrencia e df_aeronave com base na coluna 'codigo_ocorrencia2', combinando informações relevantes sobre ocorrências e aeronaves.

features = ['fatalidade'] X = merged_df[features]: Escolhe a característica 'fatalidade' como a única feature para treinar o modelo de Regressão Logística.

model = LogisticRegression(): Cria um modelo de Regressão Logística.

y = (merged_df['classificacao'] == 'INCIDENTE').astype(int) model.fit(X, y): Treina o modelo de Regressão Logística usando todas as instâncias disponíveis nos conjuntos de dados X (features) e y (rótulos). A variável y é criada considerando se a 'classificacao' é 'INCIDENTE' e convertendo os valores para 0 ou 1.

proba_acidente = model.predict_proba(X)[:, 1]: Utiliza o modelo para fazer previsões de probabilidade. predict_proba retorna as probabilidades para cada classe, e [:, 1] seleciona as probabilidades associadas à classe positiva, neste caso, 'INCIDENTE'.

filtro_acidente = (merged_df['classificacao'] == 'INCIDENTE'): Cria um filtro para selecionar as instâncias onde a 'classificacao' é 'INCIDENTE'.

probabilidade_media = proba_acidente[filtro_acidente].mean(): Calcula a média das probabilidades de 'INCIDENTE' apenas para as instâncias onde a classificação é 'INCIDENTE'.

print(f"Probabilidade média de fatalidade maior que zero quando classificação é 'INCIDENTE': {probabilidade_media:.2f}"): Exibe a probabilidade média calculada.

Portanto, o código treina um modelo de Regressão Logística para prever a probabilidade de um incidente (considerando a classificação 'INCIDENTE') com base na feature 'fatalidade' e, em seguida, calcula a probabilidade média de fatalidade maior que zero para as instâncias onde a classificação é 'INCIDENTE'.

In [ ]:
```
1
```