

# Introdução à Inteligência Artificial

## Unidade 2

Discente: Samnatha Dantas Medeiros

Dataset utilizado: <https://github.com/tmfilho/akcdata/blob/master/data/akc-data-latest.csv>

## Ambiente, funções e *misc*

A seção a seguir contém a inicialização deste notebook (bibliotecas a serem utilizadas, funções de propósitos específicos e algumas variáveis para ficarem 'separadas' dos blocos adjacentes).

```
In [ ]: #pip install imbalanced-Learn
```

```
In [ ]: # Dependências
import random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler, LabelEncoder
from sklearn.metrics import (accuracy_score, precision_score, recall_score, confusi
```

```
In [ ]: # Funções que serão utilizadas ao longo do pré-processamento de dados
# parâmetros: dataframe, coluna numérica, coluna que armazena valor do grupo (toy,
...
Esta função substitui valores não-numéricos(caso hajam) e nulos
de colunas numéricas do dataframe pela mediana
de suas colunas de acordo com o grupo dos cães (toy, pastoreiro e etc).
...
def replace_with_median(df, n_col, g_col):
    # Calcula a mediana por grupo
    median_by_group = df.groupby(g_col)[n_col].transform('median')

    # Substitui valores nulos e strings pela mediana do grupo
    df[n_col] = df[n_col].fillna(median_by_group)

    return df

...
Esta função preenche valores categóricos com a moda
...
def replace_with_mode(df, cols, g_col):
    for c in cols:
        for group, group_data in df.groupby(g_col):
            c_mode = group_data[c].mode()[0]
            df.loc[df[g_col] == group, c] = df.loc[df[g_col] == group, c].fillna(c_mode)

    return df

...
A seguinte função irá imprimir os valores únicos de cada coluna
do dataframe a fim de ajudar a visualizar os intervalos ou os
conjuntos de valores de cada coluna para eventuais análises
...
def all_unique_in(df):
    for c in df.columns:
        valores_unicos = df[c].unique()
        if np.issubdtype(valores_unicos.dtype, np.number): # Verifica se a coluna cont
            valores_unicos = np.sort(valores_unicos) # Ordena os valores únicos numérico
        else:
            valores_unicos = np.sort(valores_unicos.astype(str)) # Converte para strings

        print(f"Valores únicos da coluna '{c}':")
        print(valores_unicos)
        print("-----")

    ...
Essa função irá separar os 3 adjetivos de temperamentos de cães em 3 novas colunas.
Isso irá permitir o uso de Label Encoding de forma mais organizada.
...
def temperaments_cols(df):
    # Separar os adjetivos em novas colunas
    df_temperament = df["temperament"].str.split(" ", expand=True)

    # Renomear as novas colunas
    num_adjectives = len(df_temperament.columns)
    new_columns = [f"temperament_{i+1}" for i in range(num_adjectives)]
    df_temperament.columns = new_columns
```

```
# Concatenar as novas colunas com o DataFrame original
df = pd.concat([df, df_temperament], axis=1)

return df

'''
Esta função permite excluir múltiplas colunas
dado um array de colunas
'''

def drop_cols(df, cols):
    return df.drop(columns=cols)
```

```
In [ ]: # colunas numéricas
numeric_cols = [
    'min_height',
    'max_height',
    'min_weight',
    'max_weight',
    'min_expectancy',
    'max_expectancy',
    'grooming_frequency_value',
    'energy_level_value',
    'shedding_value',
    'trainability_value',
    'demeanor_value'
]

# colunas categóricas (apenas as colunas group e __ serão preservadas)
text_cols = [
    'dog_breed',
    'description',
    'grooming_frequency_category',
    'shedding_category',
    'energy_level_category',
    'trainability_category',
    'demeanor_category',
]

# utilizado para agrupamento de valores
mode_cols = ['temperament']

group_col = 'group'
```

```
In [ ]: # Criando o dataframe
og = pd.read_csv('./dog_breeds.csv', encoding='utf-8') # original
df = pd.read_csv('./dog_breeds.csv', encoding='utf-8') # dataframe final
```

## Questão 1

Execute o pré-processamento necessário para que você possa executar métodos de aprendizado supervisionado e não supervisionado;

### Resolução

O pré-processamento dos dados é uma etapa essencial para a construção de um modelo, pois utiliza diversas técnicas para melhorar a qualidade dos dados.

Para esta base de dados, caso haja valores ausentes, eles serão substituídos pela:

- mediana de suas respectivas colunas (valores numéricos)
- moda de suas respectivas colunas (valores categóricos)

E algumas colunas serão removidas do dataframe por não serem revelantes para o problema a ser resolvido ou por já existir uma coluna equivalente com valores numéricos, como é o caso de `trainability_value` e `trainability_category`.

Nas células de texto código adjacentes haverá uma descrição/explicação breve sobre as decisões tomadas para o pré-processamento dos dados.

Para realizar a comparação com a base de dados original e o dataframe pré-processado, será feito uso das variáveis `og` e `df`, respectivamente.

Sobre a escolha da mediana:

A mediana é menos sensível a valores atípicos, o que a torna uma escolha mais segura quando há presença de outliers ou quando os dados têm uma distribuição assimétrica. A mediana é o valor central que divide os dados ordenados ao meio, portanto não é afetada por valores extremos.

## Exclusão de Colunas

As seguintes colunas serão excluídas do dataframe, pois elas são irrelevantes para o problema dada a correlação entre esses atributos e outras justificativas (descritas abaixo).

- **Nome da raça de cão**, não é relevante pois trata-se da indexação do dataframe, ou seja, cada objeto do dataframe é sobre uma determinada raça de cão e cada raça possui seus valores mínimos e máximos de atributos pré-estabelecidos (altura, peso, temperamento, etc). Logo, não é interessante manter essa coluna, pois o problema a ser resolvido com o modelo não é prever a qual raça de cão um determinado cão pertence.
- **Descrição** é um texto bastante extenso que pode ser de difícil processamento, além de que no próprio texto cita algumas das características já presente em outras colunas.
- **Popularidade** (coluna removida na questão 2, pois a correlação deste atributo com os demais é baixa)
- E, como já mencionado anteriormente, as colunas abaixo serão excluídas do dataframe devido a existência de colunas que possuem valores numéricos equivalentes a eles (por exemplo, em `energy_level_value` 0.2 equivale a `energy_level_category` Couch Potato)
  - `energy_level_category`
  - `demeanor_category`
  - `grooming_frequency_category`
  - `shedding_category`
  - `trainability_category`

```
In [ ]: # renomeando a primeira coluna do dataframe (originalmente sem nome)
df = df.rename(columns={df.columns[0]: 'dog_breed'})

# Removendo colunas categóricas, visto que já existem as colunas numéricas correspo
df = drop_cols(df, text_cols)
```

## Valores únicos

Apenas para registrar os valores únicos de cada coluna com o propósito de visualizar intervalos ou grupos de valores.

```
# função dedicada para obter todos os valores únicos de uma única vez  
all_unique_in()
```

### Group

- Foundation Stock Service
- Herding Group
- Hound Group
- Miscellaneous Class
- Non-Sporting Group
- Sporting Group
- Terrier Group
- Toy Group
- Working Group

### Grooming\_frequency\_value

- 0.2 - Occasional Bath/Brush
- 0.4 - Weekly Brushing
- 0.6 - 2-3 Times a Week Brushing
- 0.8 - Daily Brushing
- 1.0 - Specialty/Professional

### Shedding\_value

- 0.2 - Infrequent
- 0.4 - Occasional
- 0.6 - Seasonal
- 0.8 - Regularly
- 1.0 - Frequent

### Energy\_level\_value

- 0.2 - Couch Potato
- 0.4 - Calm
- 0.6 - Regular Exercise
- 0.8 - Energetic
- 1.0 - Needs Lots of Activity

### Trainability\_value

- 0.2 - May be Stubborn
- 0.4 - Independent

- 0.6 - Agreeable
- 0.8 - Easy Training
- 1.0 - Eager to Please

Demeanor\_value

- 0.2 - Aloof/Wary
- 0.4 - Reserved with Strangers
- 0.6 - Alert/Responsive
- 0.8 - Friendly
- 1.0 - Outgoing

```
In [ ]: # exibe os 5 primeiros elementos do dataframe
og.head()
# exibe as colunas do dataframe
og.columns
# para dar um resumo de valores não nulos encontrados
og.info()
# para retornar um resumo estatístico das variáveis numéricas
og.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 277 entries, 0 to 276
Data columns (total 21 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Unnamed: 0                            277 non-null    object
1   description                           277 non-null    object
2   temperament                           276 non-null    object
3   popularity                            198 non-null    object
4   min_height                           277 non-null    float64
5   max_height                           277 non-null    float64
6   min_weight                           275 non-null    float64
7   max_weight                           275 non-null    float64
8   min_expectancy                       274 non-null    float64
9   max_expectancy                       274 non-null    float64
10  group                                 277 non-null    object
11  grooming_frequency_value              270 non-null    float64
12  grooming_frequency_category           270 non-null    object
13  shedding_value                        257 non-null    float64
14  shedding_category                     257 non-null    object
15  energy_level_value                    271 non-null    float64
16  energy_level_category                 271 non-null    object
17  trainability_value                    253 non-null    float64
18  trainability_category                 253 non-null    object
19  demeanor_value                        252 non-null    float64
20  demeanor_category                     252 non-null    object
dtypes: float64(11), object(10)
memory usage: 45.6+ KB
```

Out[ ]:

	min_height	max_height	min_weight	max_weight	min_expectancy	max_expectancy	groon
count	277.000000	277.000000	275.000000	275.000000	274.000000	274.000000	
mean	44.225801	52.720588	17.888858	27.291416	11.306569	13.832117	
std	14.238298	15.885454	12.290600	19.061416	1.817949	2.016668	
min	12.700000	17.780000	0.000000	0.000000	0.000000	0.000000	
25%	33.020000	38.100000	8.164663	12.927383	10.000000	13.000000	
50%	45.085000	53.340000	15.875733	24.947580	12.000000	14.000000	
75%	55.880000	66.040000	22.679619	34.019428	12.000000	15.000000	
max	76.200000	88.900000	68.038855	108.862169	16.000000	19.000000	



## Preenchendo valores ausentes

Como há diferentes grupos de cães e cada um tem as suas características, os valores nulos serão substituídos de acordo com o grupo que o cão pertence. Ou seja, caso um cão do grupo Toy estiver com a altura mínima ausente, esse valor será substituído pela mediana de altura mínima de cães do grupo Toy. Isso irá auxiliar a possuir valores que condizem com a realidade.

Por exemplo: o peso mínimo e máximo de um cão do grupo toy não pode ser substituído com a mediana dos pesos de todas as raças envolvidas, pois pode fugir de um valor real.

### Valores numéricos

- popularity \*
- min\_height
- max\_height
- min\_weight
- max\_weight
- min\_expectancy
- max\_expectancy
- grooming\_frequency\_value
- energy\_level\_value
- shedding\_value
- trainability\_value
- demeanor\_value

### Valores categóricos

- group
- temperament \*\*

\* Observação quanto ao atributo de **popularidade**: o tipo de dados dessa coluna estava como object, assim fugindo de valores esperados (como inteiro ou float). Analisando o arquivo original, além dos valores ausentes encontrados, existia linhas dessa coluna preenchida com valores não numéricos. Sendo assim, ocorreu uma conversão do tipo de dado dessa coluna (object para float) e assim foi possível tratar os dados ausentes. No entanto, como a popularidade de todos os cães do grupo *Foundation Stock Service* e *Miscellaneous Class* é vazia, não foi possível tratar essa coluna da mesma forma que as demais, portanto, os valores ausentes da coluna popularidade foram substituídos com a mediana geral de popularidade de todos os cães do dataframe. Outra alternativa poderia ser a exclusão da coluna, mas no atual momento, não se viu a necessidade.

**\*\* Observação quanto ao atributo **temperament**:** foi implementada uma função que cria 3 novas colunas contendo cada um dos adjetivos da coluna original. Essa coluna vai ser preservada pois ela pode ser importante ao se classificar um cão em um determinado grupo canino junto aos demais atributos. Posteriormente, esses valores passarão pelo process de Label Encoding para atribuir um valor numérico correspondente (a partir de uma lista que possui os valores únicos de cada uma das colunas). Por fim, é notado que alguns dos atributos estão escritos como "Smart and Playful" ou então "Active but hard-working", isso dificulta a separação concisa desses adjetivos nas novas colunas. Pode-se observar isso no momento que se vai avaliar a quantidade de valores nulos totais em cada coluna, nas colunas temperament\_2 e temperament\_3 há valores 1 e 2 valores nulos, respectivamente. Mas como são poucas as ocorrências, elas poderiam ser tratadas manualmente (manipulando a planilha original).

```
In [ ]: # Preenchendo valores ausentes
df = replace_with_median(df, numeric_cols, group_col)
df = replace_with_mode(df, mode_cols, group_col)

# Verificando se ficou algum valor ausente no data frame após o pré-processamento
df.isnull().sum()
```

```
Out[ ]: temperament      0
popularity      79
min_height      0
max_height      0
min_weight      0
max_weight      0
min_expectancy  0
max_expectancy  0
group           0
grooming_frequency_value  0
shedding_value  0
energy_level_value  0
trainability_value  0
demeanor_value  0
dtype: int64
```

```
In [ ]: df = temperaments_cols(df)
```

```
In [ ]: # Soma de valores nulos encontrados no data frame (antes do pré-processamento)
df.isnull().sum()
```

```
Out[ ]: temperament      0
        popularity      79
        min_height      0
        max_height      0
        min_weight      0
        max_weight      0
        min_expectancy   0
        max_expectancy   0
        group            0
        grooming_frequency_value 0
        shedding_value   0
        energy_level_value 0
        trainability_value 0
        demeanor_value   0
        temperament_1    0
        temperament_2    1
        temperament_3    2
        dtype: int64
```

```
In [ ]: # função dedicada para obter todos os valores únicos de cada coluna de uma única vez
        all_unique_in(df)
```

Valores únicos da coluna 'temperament':

[ 'Active, Outgoing, Sweet-Natured' 'Active, Proud, Very Smart'  
 'Adaptable, Friendly, Gentle' 'Adaptable, Gentle, Smart'  
 'Adaptable, Playful, Smart' 'Affectionate, Active, Enthusiastic'  
 'Affectionate, Adaptable, Gentle' 'Affectionate, Adventurous, Athletic'  
 'Affectionate, Alert, Curious' 'Affectionate, Alert, Intelligent'  
 'Affectionate, Alert, Lively' 'Affectionate, Bright, Sensitive'  
 'Affectionate, Confident, Bold' 'Affectionate, Courageous, Spirited'  
 'Affectionate, Courageous, Strong-Willed'  
 'Affectionate, Dignified, Even-Tempered'  
 'Affectionate, Eager, Enthusiastic' 'Affectionate, Friendly, Independent'  
 'Affectionate, Gentle, Energetic' 'Affectionate, Gentle, Graceful'  
 'Affectionate, Gentle, Sociable' 'Affectionate, Happy, Plucky'  
 'Affectionate, Intelligent, Active'  
 'Affectionate, Intelligent, Enthusiastic'  
 'Affectionate, Intelligent, Kind' 'Affectionate, Intelligent, Majestic'  
 'Affectionate, Intelligent, Outgoing' 'Affectionate, Keen, Undemanding'  
 'Affectionate, Loyal, Brave' 'Affectionate, Loyal, Courageous'  
 'Affectionate, Loyal, Noble' 'Affectionate, Loyal, Playful'  
 'Affectionate, Loyal, Regal in Manner'  
 'Affectionate, Loyal, Regally Dignified' 'Affectionate, Loyal, Sensitive'  
 'Affectionate, Loyal, Smart' 'Affectionate, Outgoing, Positive'  
 'Affectionate, Playful, Calm' 'Affectionate, Playful, Outgoing'  
 'Affectionate, Smart, Alert' 'Affectionate, Smart, Energetic'  
 'Affectionate, Sprightly, Tomboyish'  
 'Affectionate, Versatile, Intelligent'  
 'Agile, Intelligent, Self-Confident' 'Agile, Spirited, Intelligent'  
 'Agile, Versatile, Lively' 'Agile, Versatile, Regal'  
 'Alert, Active, Attentive' 'Alert, Cheerful, Busy'  
 'Alert, Curious, Pleasant' 'Alert, Eager, Intelligent'  
 'Alert, Faithful, Intelligent' 'Alert, Happy, Vivacious'  
 'Alert, Inquisitive, Lively' 'Alert, Intelligent, Bold'  
 'Alert, Intelligent, Brave' 'Alert, Outgoing, Sense Of Humor'  
 'Alert, Trainable, Comical' 'Bold, Dashing, Tenderhearted'  
 'Bright, Fun-Loving, Active' 'Bright, Fun-Loving, Upbeat'  
 'Bright, Watchful, Serious-Minded' 'Charming, Bright, Happy-Go-Lucky'  
 'Charming, Graceful, Sassy' 'Charming, Mischievous, Loving'  
 'Charming, Noble, Loving' 'Charming, Patient, Low-Key'  
 'Cheerful, Optimistic, Good-Humored'  
 'Clever, Adventurous, Family-Oriented' 'Clever, Brave, Tenacious'  
 'Confident, Alert, Curious' 'Confident, Alert, Gregarious'  
 'Confident, Alert, Vigilant' 'Confident, Calm, Family-Oriented'  
 'Confident, Calm, Powerful' 'Confident, Calm, Strong-Willed'  
 'Confident, Clever, Lively' 'Confident, Courageous, Smart'  
 'Confident, Famously Funny, Fearless' 'Confident, Independent, Spirited'  
 'Confident, Intelligent, Calm' 'Confident, Proud, Wickedly Smart'  
 'Confident, Smart, Comical' 'Confident, Smart, Faithful'  
 'Confident, Smart, Good-Natured' 'Confident, Smart, Hardworking'  
 'Confident, Smart, Perceptive' 'Courageous, Alert, Determined'  
 'Courageous, Alert, Faithful' 'Courageous, Alert, Intelligent'  
 'Courageous, Dignified, Calm' 'Courageous, Dignified, Good-Natured'  
 'Courageous, Dignified, Profoundly Loyal'  
 'Courageous, Good-Natured, Determined' 'Courageous, Good-Tempered, Canny'  
 'Courageous, Intelligent, Hardworking'  
 'Courageous, Intelligent, Vivacious' 'Courageous, Spirited, Determined'  
 'Devoted, Graceful, Proud' 'Devoted, Lively, Attentive'  
 'Dignified, Alert, Devoted' 'Dignified, Bright, Serious-Minded'  
 'Dignified, Profoundly Loyal, Aristocratic' 'Dignified, Smart, Outgoing'  
 'Eager, Happy, Charming' 'Easy-Going, Bright, Brave'

'Energetic, Alert, Curious' 'Energetic, Alert, Enthusiastic'  
'Energetic, Lively, Ready to Work' 'Energetic, Merry, Responsive'  
'Energetic, Trainable, Attentive' 'Even-Tempered, Amiable, Boisterous'  
'Even-Tempered, Amiable, Eager to Please'  
'Faithful, Family-Oriented, Dependable' 'Faithful, Noble, Docile'  
'Family-Oriented, Even-Tempered, Polite' 'Fearless, Agile, Attentive'  
'Fearless, Alert, Fun-Loving' 'Fearless, Bold, Kind'  
'Fearless, Fun-Loving, Proud' 'Fearless, Patient, Vigilant'  
'Fearless, Smart, Spirited' 'Friendly, Active, Outgoing'  
'Friendly, Alert, Agile' 'Friendly, Alert, Happy'  
'Friendly, Alert, Intelligent' 'Friendly, Alert, Quick'  
'Friendly, Bright, Amusing' 'Friendly, Bright, Sweet-Natured'  
'Friendly, Cheerful, Humble' 'Friendly, Clever, Athletic'  
'Friendly, Clever, Courageous' 'Friendly, Confident, Bold'  
'Friendly, Confident, Dependable' 'Friendly, Courageous, Calm'  
'Friendly, Courageous, Intelligent' 'Friendly, Curious, Merry'  
'Friendly, Curious, Spunky' 'Friendly, Devoted, Alert and Intelligent'  
'Friendly, Devoted, Trainable' 'Friendly, Eager, Lovable'  
'Friendly, Energetic, Intelligent' 'Friendly, Energetic, Watchful'  
'Friendly, Fearless, Obedient' 'Friendly, Gentle, Playful'  
'Friendly, Good-Natured, Lively' 'Friendly, Happy, Deeply Devoted'  
'Friendly, Independent, Amusing' 'Friendly, Independent, Inquisitive'  
'Friendly, Inquisitive, Lovable' 'Friendly, Intelligent, Devoted'  
'Friendly, Intelligent, Eager to Please' 'Friendly, Lively, Outgoing'  
'Friendly, Mellow, Merry' 'Friendly, Merry, Even-Tempered'  
'Friendly, Outgoing, People-Oriented' 'Friendly, Patient, Dependable'  
'Friendly, Playful, Inquisitive' 'Friendly, Playful, Obedient'  
'Friendly, Quick, Keenly Alert' 'Friendly, Smart, Noble'  
'Friendly, Smart, Obedient' 'Friendly, Smart, Willing to Please'  
'Friendly, Spirited, Intelligent' 'Friendly, Versatile, Determined'  
'Friendly, Watchful, Lively' 'Friendly, Watchful, Loving'  
'Gentle, Brave, Noble' 'Gentle, Dignified, Independent-Minded'  
'Gentle, Dignified, Polite' 'Gentle, Faithful, Obedient'  
'Gentle, Independent, Noble' 'Gentle, Loyal, Trainable'  
'Gentle, Playful, Charming' 'Gentle, Playful, Intelligent'  
'Gentle, Smart, Happy' 'Gentle, Sociable, Intelligent'  
'Gentle, Spirited, Bold' 'Good-Natured, Calm, Strong'  
'Good-Natured, Intelligent, Devoted' 'Happy, Reserved, Upbeat'  
'Independent, Alert, Intelligent'  
'Independent, Courageous, Self-Confident'  
'Independent, Easy-Going, Sweet-Tempered'  
'Independent, Energetic, Powerful' 'Independent, Happy, Outgoing'  
'Independent, Reserved, Intelligent' 'Independent, Smart, Poised'  
'Independent, Smart, Proud' 'Independent, Sociable, Intelligent'  
'Inquisitive, Bold, Lively' 'Intelligent, Active, Sensitive'  
'Intelligent, Calm, Powerful' 'Intelligent, Devoted, Amusing'  
'Intelligent, Devoted, Lively' 'Intelligent, Devoted, Self-Confident'  
'Intelligent, Lively, Athletic' 'Intelligent, Outgoing, Funny'  
'Intelligent, Responsive, Loving'  
'Intelligent, Strong-Willed, Reserved with Strangers'  
'Loyal, Active, Alert and Intelligent' 'Loyal, Agile, Keen'  
'Loyal, Alert, Calm' 'Loyal, Alert, Curious' 'Loyal, Alert, Intelligent'  
'Loyal, Alert, Trainable' 'Loyal, Charming, Frolicking'  
'Loyal, Dignified, Brave' 'Loyal, Dignified, Watchful'  
'Loyal, Energetic, Alert' 'Loyal, Even-Tempered, Determined'  
'Loyal, Fearless, Alert' 'Loyal, Fearless, Sweet'  
'Loyal, Good-Natured, Intelligent' 'Loyal, Good-Natured, Observant'  
'Loyal, Happy, Entertaining' 'Loyal, Hardworking, Even-Tempered'  
'Loyal, Independent, Agile' 'Loyal, Independent, Calm'

'Loyal, Independent, Confident Guardian' 'Loyal, Independent, Courageous'  
 'Loyal, Independent, Deeply Affectionate' 'Loyal, Independent, Reserved'  
 'Loyal, Independent, Reserved with Strangers'  
 'Loyal, Independent, Watchful' 'Loyal, Intelligent, Active'  
 'Loyal, Intelligent, Sensitive' 'Loyal, Loving, Confident Guardian'  
 'Loyal, Mischievous, Outgoing' 'Loyal, Playful, Smart'  
 'Loyal, Self-Confident' 'Loyal, Smart, Charming'  
 'Loyal, Smart, Enthusiastic' 'Loyal, Smart, Home-Loving'  
 'Loyal, Versatile, Intelligent'  
 'Loyal, Versatile, Reserved with Strangers'  
 'Mellow, Amusing, Gentlemanly'  
 'Playful but also Work-Oriented. Very Active and Upbeat.'  
 'Playful, Alert, Sensitive' 'Playful, Bright, Self-Confident'  
 'Playful, Charming, Inquisitive' 'Playful, Charming, Lively'  
 'Playful, Charming, Mischievous' 'Playful, Curious, Peppy'  
 'Playful, Easy-Going, Devoted' 'Playful, Energetic, Bright'  
 'Playful, Hardworking, Brave' 'Playful, Intelligent, Lively'  
 'Playful, Perky, Smart' 'Playful, Versatile, Intelligent'  
 'Reserved, Graceful, Noble' 'Smart, Alert, People-Oriented'  
 'Smart, Bouncy, Charismatic' 'Smart, Brave, Courteous'  
 'Smart, Cheerful, Determined' 'Smart, Devoted, Tenacious'  
 'Smart, Patient, Calm' 'Smart, Patient, Devoted'  
 'Smart, Work-Oriented, Exuberant' 'Sociable, Energetic, Intelligent'  
 'Sociable, Patient, Docile' 'Spirited, Bright, Keenly Observant'  
 'Sweet, Fun-Loving, Sensitive' 'Sweet, Mellow, Sociable'  
 'Sweet, Patient, Devoted' 'Upbeat, Mischievous, Comical'  
 'Versatile, Calm, Family-Oriented']

-----  
 Valores únicos da coluna 'popularity':

['1' '10' '100' '101' '102' '103' '104' '105' '106' '107' '108' '109' '11'  
 '110' '111' '112' '113' '114' '115' '116' '117' '118' '119' '12' '120'  
 '121' '122' '123' '124' '125' '126' '127' '128' '129' '13' '130' '131'  
 '132' '133' '134' '135' '136' '137' '138' '14' '140' '141' '142' '143'  
 '144' '145' '146' '147' '148' '149' '150' '151' '152' '153' '154' '155'  
 '156' '157' '158' '159' '16' '160' '161' '162' '163' '164' '165' '166'  
 '167' '168' '169' '17' '170' '171' '172' '173' '174' '175' '176' '177'  
 '178' '179' '18' '180' '181' '182' '183' '184' '185' '186' '187' '188'  
 '189' '19' '190' '191' '192' '2' '20' '21' '22' '23' '24' '25' '26' '27'  
 '28' '29' '3' '30' '31' '32' '33' '34' '35' '36' '37' '38' '39' '4' '40'  
 '41' '42' '43' '44' '45' '46' '47' '48' '49' '5' '50' '51' '52' '53' '54'  
 '55' '56' '57' '58' '59' '6' '60' '61' '62' '63' '64' '65' '66' '67' '68'  
 '69' '7' '70' '71' '72' '73' '74' '75' '76' '77' '78' '79' '8' '80' '81'  
 '82' '83' '84' '85' '86' '87' '88' '89' '9' '90' '91' '92' '93' '94' '95'  
 '96' '97' '98' '99' 'nan' 'of']

-----  
 Valores únicos da coluna 'min\_height':

[12.7 15.24 17.78 20.32 21.59 22.86 24.13 24.765 25.4 26.67  
 27.94 29.21 30.48 31.75 33.02 34.29 35.56 38.1 39.37 40.005  
 40.64 41.91 43.18 44.45 45.085 45.72 46.99 48.26 49.022 49.53  
 50.8 52.07 53.34 54.61 55.88 57.15 58.42 59.69 60.96 62.23  
 63.5 64.77 66.04 68.58 69.85 71.12 76.2 ]

-----  
 Valores únicos da coluna 'max\_height':

[17.78 20.32 22.86 25.4 26.67 27.94 29.21 30.48 31.75 33.02  
 34.925 35.56 36.83 38.1 39.37 40.64 41.91 43.18 44.45 45.72  
 46.99 48.26 49.53 50.165 50.8 52.07 53.34 54.61 55.118 55.88  
 57.15 58.42 59.69 60.96 62.23 63.5 64.77 65.405 66.04 67.31  
 68.58 69.85 71.12 72.39 73.66 74.93 76.2 78.74 80.01 81.28  
 88.9 ]

```
-----
Valores únicos da coluna 'min_weight':
[ 0.          1.36077711  1.5875733   1.81436948  2.04116567  2.26796185
  2.49475804  2.72155422  2.9483504   3.17514659  3.62873896  3.85553515
  4.08233133  4.5359237   4.98951607  5.21631226  5.44310844  5.89670081
  6.35029318  6.80388555  7.71107029  8.16466266  9.0718474   9.97903214
 10.43262451 10.88621688 11.33980925 11.79340162 12.24699399 12.70058636
 13.15417873 13.6077711  14.06136347 14.51495584 14.96854821 15.87573295
 16.78291769 17.00971387 18.1436948  19.05087954 19.95806428 20.41165665
 21.77243376 22.6796185  24.94758035 25.85476509 27.2155422  28.12272694
 29.48350405 30.84428116 31.7514659  34.01942775 34.92661249 35.83379723
 36.2873896  38.10175908 38.55535145 39.91612856 40.8233133  44.90564463
 45.359237   47.62719885 49.8951607  54.4310844  63.5029318  68.0388555 ]
```

```
-----
Valores únicos da coluna 'max_weight':
[ 0.          2.72155422  2.9483504   3.17514659  3.62873896
  4.08233133  4.5359237   4.98951607  5.44310844  5.89670081
  6.35029318  6.80388555  7.03068173  7.25747792  7.71107029
  8.16466266  9.0718474   9.97903214 10.43262451 10.88621688
 11.33980925 11.79340162 12.24699399 12.70058636 13.15417873
 13.6077711  14.51495584 14.96854821 15.42214058 15.87573295
 17.23651006 18.1436948  19.95806428 20.41165665 22.22602613
 22.6796185  24.04039561 24.94758035 25.85476509 26.76194983
 27.2155422  28.12272694 29.48350405 29.93709642 30.84428116
 31.7514659  33.11224301 34.01942775 36.2873896  38.10175908
 38.55535145 40.8233133  43.09127515 43.99845989 44.90564463
 45.359237   47.62719885 49.8951607  52.16312255 54.4310844
 58.9670081  59.87419284 61.23496995 63.5029318  68.0388555
 77.1107029  79.37866475 81.6466266  90.718474  104.3262451
108.8621688 ]
```

```
-----
Valores únicos da coluna 'min_expectancy':
[ 0.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16.]
```

```
-----
Valores únicos da coluna 'max_expectancy':
[ 0.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19.]
```

```
-----
Valores únicos da coluna 'group':
['Foundation Stock Service' 'Herding Group' 'Hound Group'
 'Miscellaneous Class' 'Non-Sporting Group' 'Sporting Group'
 'Terrier Group' 'Toy Group' 'Working Group']
```

```
-----
Valores únicos da coluna 'grooming_frequency_value':
[0.2 0.4 0.6 0.8 1. ]
```

```
-----
Valores únicos da coluna 'shedding_value':
[0.2 0.4 0.6 0.8 1. ]
```

```
-----
Valores únicos da coluna 'energy_level_value':
[0.2 0.4 0.6 0.8 1. ]
```

```
-----
Valores únicos da coluna 'trainability_value':
[0.2 0.4 0.6 0.8 1. ]
```

```
-----
Valores únicos da coluna 'demeanor_value':
[0.2 0.4 0.6 0.8 1. ]
```

```
-----
Valores únicos da coluna 'temperament_1':
['Active' 'Adaptable' 'Affectionate' 'Agile' 'Alert' 'Bold' 'Bright'
```

'Charming' 'Cheerful' 'Clever' 'Confident' 'Courageous' 'Devoted'  
'Dignified' 'Eager' 'Easy-Going' 'Energetic' 'Even-Tempered' 'Faithful'  
'Family-Oriented' 'Fearless' 'Friendly' 'Gentle' 'Good-Natured' 'Happy'  
'Independent' 'Inquisitive' 'Intelligent' 'Loyal' 'Mellow' 'Playful'  
'Playful but also Work-Oriented. Very Active and Upbeat.' 'Reserved'  
'Smart' 'Sociable' 'Spirited' 'Sweet' 'Upbeat' 'Versatile']

-----  
Valores únicos da coluna 'temperament\_2':

['Active' 'Adaptable' 'Adventurous' 'Agile' 'Alert' 'Amiable' 'Amusing'  
'Bold' 'Bouncy' 'Brave' 'Bright' 'Calm' 'Charming' 'Cheerful' 'Clever'  
'Confident' 'Courageous' 'Curious' 'Dashing' 'Devoted' 'Dignified'  
'Eager' 'Easy-Going' 'Energetic' 'Even-Tempered' 'Faithful'  
'Family-Oriented' 'Famously Funny' 'Fearless' 'Friendly' 'Fun-Loving'  
'Gentle' 'Good-Natured' 'Good-Tempered' 'Graceful' 'Happy' 'Hardworking'  
'Independent' 'Inquisitive' 'Intelligent' 'Keen' 'Lively' 'Loving'  
'Loyal' 'Mellow' 'Merry' 'Mischievous' 'Noble' 'None' 'Optimistic'  
'Outgoing' 'Patient' 'Perky' 'Playful' 'Profoundly Loyal' 'Proud' 'Quick'  
'Reserved' 'Responsive' 'Self-Confident' 'Smart' 'Sociable' 'Spirited'  
'Sprightly' 'Strong-Willed' 'Trainable' 'Versatile' 'Watchful'  
'Work-Oriented']

-----  
Valores únicos da coluna 'temperament\_3':

['Active' 'Agile' 'Alert' 'Alert and Intelligent' 'Amusing' 'Aristocratic'  
'Athletic' 'Attentive' 'Boisterous' 'Bold' 'Brave' 'Bright' 'Busy' 'Calm'  
'Canny' 'Charismatic' 'Charming' 'Comical' 'Confident Guardian'  
'Courageous' 'Courteous' 'Curious' 'Deeply Affectionate' 'Deeply Devoted'  
'Dependable' 'Determined' 'Devoted' 'Docile' 'Eager to Please'  
'Energetic' 'Entertaining' 'Enthusiastic' 'Even-Tempered' 'Exuberant'  
'Faithful' 'Family-Oriented' 'Fearless' 'Frollicking' 'Fun-Loving'  
'Funny' 'Gentle' 'Gentlemanly' 'Good-Humored' 'Good-Natured' 'Graceful'  
'Gregarious' 'Happy' 'Happy-Go-Lucky' 'Hardworking' 'Home-Loving'  
'Humble' 'Independent' 'Independent-Minded' 'Inquisitive' 'Intelligent'  
'Keen' 'Keenly Alert' 'Keenly Observant' 'Kind' 'Lively' 'Lovable'  
'Loving' 'Low-Key' 'Majestic' 'Merry' 'Mischievous' 'Noble' 'None'  
'Obedient' 'Observant' 'Outgoing' 'People-Oriented' 'Peppy' 'Perceptive'  
'Playful' 'Pleasant' 'Plucky' 'Poised' 'Polite' 'Positive' 'Powerful'  
'Profoundly Loyal' 'Proud' 'Quick' 'Ready to Work' 'Regal'  
'Regal in Manner' 'Regally Dignified' 'Reserved'  
'Reserved with Strangers' 'Responsive' 'Sassy' 'Self-Confident'  
'Sense Of Humor' 'Sensitive' 'Serious-Minded' 'Smart' 'Sociable'  
'Spirited' 'Spunky' 'Strong' 'Strong-Willed' 'Sweet' 'Sweet-Natured'  
'Sweet-Tempered' 'Tenacious' 'Tenderhearted' 'Tomboyish' 'Trainable'  
'Undemanding' 'Upbeat' 'Very Smart' 'Vigilant' 'Vivacious' 'Watchful'  
'Wickedly Smart' 'Willing to Please']



```

In [ ]: df['temperament_2'].fillna('Unknown', inplace=True)
df['temperament_3'].fillna('Unknown', inplace=True)

print(df.isnull().sum()) # temperament_2 tinha 1 valor nulo antes, temperament_3 ti

temperament_values = df['temperament_1'].unique().tolist()
temperament_values += df['temperament_2'].unique().tolist()
temperament_values += df['temperament_3'].unique().tolist()
temperament_values_unique = list(set(temperament_values))
temperament_values_sorted = sorted(temperament_values_unique)

print(temperament_values_sorted)
print(len(temperament_values_sorted)) # 141 adjetivos (podendo ser por volta de 135

temperament          0
popularity            79
min_height            0
max_height            0
min_weight            0
max_weight            0
min_expectancy        0
max_expectancy        0
group                 0
grooming_frequency_value 0
shedding_value        0
energy_level_value    0
trainability_value    0
demeanor_value        0
temperament_1         0
temperament_2         0
temperament_3         0
dtype: int64
['Active', 'Adaptable', 'Adventurous', 'Affectionate', 'Agile', 'Alert', 'Alert and
Intelligent', 'Amiable', 'Amusing', 'Aristocratic', 'Athletic', 'Attentive', 'Boist
erous', 'Bold', 'Bouncy', 'Brave', 'Bright', 'Busy', 'Calm', 'Canny', 'Charismatic
', 'Charming', 'Cheerful', 'Clever', 'Comical', 'Confident', 'Confident Guardian',
'Courageous', 'Courteous', 'Curious', 'Dashing', 'Deeply Affectionate', 'Deeply Dev
oted', 'Dependable', 'Determined', 'Devoted', 'Dignified', 'Docile', 'Eager', 'Eage
r to Please', 'Easy-Going', 'Energetic', 'Entertaining', 'Enthusiastic', 'Even-Temp
ered', 'Exuberant', 'Faithful', 'Family-Oriented', 'Famously Funny', 'Fearless', 'F
riendly', 'Frollicking', 'Fun-Loving', 'Funny', 'Gentle', 'Gentlemanly', 'Good-Humo
red', 'Good-Natured', 'Good-Tempered', 'Graceful', 'Gregarious', 'Happy', 'Happy-Go
-Lucky', 'Hardworking', 'Home-Loving', 'Humble', 'Independent', 'Independent-Minded
', 'Inquisitive', 'Intelligent', 'Keen', 'Keenly Alert', 'Keenly Observant', 'Kind
', 'Lively', 'Lovable', 'Loving', 'Low-Key', 'Loyal', 'Majestic', 'Mellow', 'Merry
', 'Mischievous', 'Noble', 'Obedient', 'Observant', 'Optimistic', 'Outgoing', 'Pati
ent', 'People-Oriented', 'Peppy', 'Perceptive', 'Perky', 'Playful', 'Playful but al
so Work-Oriented. Very Active and Upbeat.', 'Pleasant', 'Plucky', 'Poised', 'Polite
', 'Positive', 'Powerful', 'Profoundly Loyal', 'Proud', 'Quick', 'Ready to Work', '
Regal', 'Regal in Manner', 'Regally Dignified', 'Reserved', 'Reserved with Stranger
s', 'Responsive', 'Sassy', 'Self-Confident', 'Sense Of Humor', 'Sensitive', 'Seriou
s-Minded', 'Smart', 'Sociable', 'Spirited', 'Sprightly', 'Spunky', 'Strong', 'Stron
g-Willed', 'Sweet', 'Sweet-Natured', 'Sweet-Tempered', 'Tenacious', 'Tenderhearted
', 'Tomboyish', 'Trainable', 'Undemanding', 'Unknown', 'Upbeat', 'Versatile', 'Very
Smart', 'Vigilant', 'Vivacious', 'Watchful', 'Wickedly Smart', 'Willing to Please',
'Work-Oriented']
141

```

```
In [ ]: '''
* Caso especial: popularidade dos grupos Miscellaneous Class e Foundation Stock Ser
Esses dois grupos não possuem valor de popularidade para nenhum dos cães que pertec
Solução: atribuir um valor de popularidade
'''

#! execute essa célula 2 vezes

print(df.isnull().sum()) # antes

# Converte popularity para float e substitui os valores nulos pela mediana (de acor
df['popularity'] = df['popularity'].replace('of', pd.NA)
df['popularity'] = pd.to_numeric(df['popularity'], errors='coerce')
df['popularity'] = df['popularity'].fillna(df['popularity'].median())

print(df.isnull().sum()) # depois

# salvando em um novo arquivo .csv (para comparações)
df.to_csv('dogs1.csv', index=False)
```

```
temperament      0
popularity        0
min_height       0
max_height       0
min_weight       0
max_weight       0
min_expectancy   0
max_expectancy   0
group            0
grooming_frequency_value  0
shedding_value   0
energy_level_value  0
trainability_value  0
demeanor_value   0
temperament_1    0
temperament_2    0
temperament_3    0
dtype: int64
temperament      0
popularity        0
min_height       0
max_height       0
min_weight       0
max_weight       0
min_expectancy   0
max_expectancy   0
group            0
grooming_frequency_value  0
shedding_value   0
energy_level_value  0
trainability_value  0
demeanor_value   0
temperament_1    0
temperament_2    0
temperament_3    0
dtype: int64
```

## Questão 2

Escolha um dos atributos de entrada para que passe a ser um atributo de saída e assim você tenha um problema supervisionado em mãos;

### Resolução

No aprendizado supervisionado, precisa-se de rótulos ou respostas conhecidas para treinar um modelo e realizar previsões ou classificações.

Dentre os atributos preservados no dataframe ( `df.columns` ), o atributo `group` será escolhido para ser o atributo de saída, pois, para enquadrar um cão em um grupo canino são levado em consideração o seu porte (altura, peso), temperamento, treinabilidade, pelagem e etc (ou seja, os atributos numéricos preservados na base de dados após o pré-processamento dos mesmos).

A *American Kennel Club* (AKC) classifica as raças de cães com base nos atributos citados, eis um exemplo de descrição a respeito dos cães do grupo *Terrier*:

"The feisty, short-legged breeds in the Terrier Group were first bred to go underground in pursuit of rodents and other vermin. Long-legged terrier breeds dig out varmints rather than burrowing in after them, while the group's "bully" breeds, created long ago for ghastly pursuits like bull-baiting, are popular companion dogs today. Breeds in the Terrier Group are excellent competitors in the sport of Earthdog."

[Fonte](#)

Algumas das colunas que possuem valores categóricos (não numéricos) irão passar por um novo tratamento utilizando técnicas de codificação de rótulos. Será utilizada a técnica **Label Encoding** que transforma cada categoria em um valor inteiro correspondente.

Nas células a seguir serão feitos mais alguns ajustes e a *plotagem* de alguns gráficos para auxiliar na resolução desta e das questões posteriores.

```
In [ ]: del df['temperament']  
del df['popularity']
```

### Matriz de Correlação

```
In [ ]: print(df.columns)
```

```
Index(['popularity', 'min_height', 'max_height', 'min_weight', 'max_weight',
      'min_expectancy', 'max_expectancy', 'group', 'grooming_frequency_value',
      'shedding_value', 'energy_level_value', 'trainability_value',
      'demeanor_value', 'temperament_1', 'temperament_2', 'temperament_3'],
      dtype='object')
```

```
In [ ]: # Criar uma cópia do dataframe original
df_encoded = df.copy()

# Aplicar a codificação Label encoding nas coluna de temperamento
label_encoder = LabelEncoder()
label_encoder.fit(temperament_values_sorted) # Ajustar o codificador com a lista de
df_encoded['temperament_1_encoded'] = label_encoder.transform(df['temperament_1'])
df_encoded['temperament_2_encoded'] = label_encoder.transform(df['temperament_2'])
df_encoded['temperament_3_encoded'] = label_encoder.transform(df['temperament_3'])

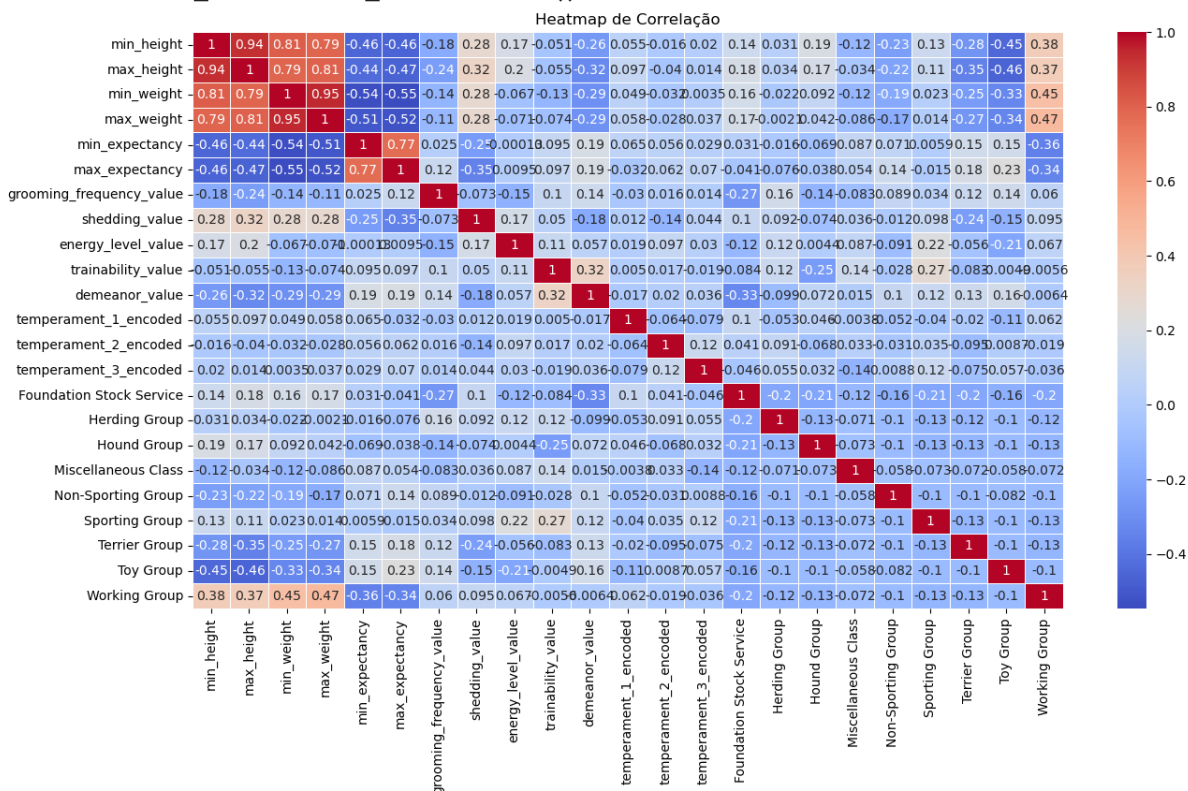
# Aplicar a codificação one-hot encoding na coluna 'group'
df_encoded = pd.get_dummies(df_encoded, columns=['group'], prefix='', prefix_sep='')

# Calcular a matriz de correlação
correlation_matrix = df_encoded.corr()

# Plotar o heatmap de correlação
plt.figure(figsize=(15, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Heatmap de Correlação')
plt.show()
```

C:\Users\saman\AppData\Local\Temp\ipykernel\_25256\3833657664.py:15: FutureWarning:  
The default value of numeric\_only in DataFrame.corr is deprecated. In a future vers  
ion, it will default to False. Select only valid columns or specify the value of nu  
meric\_only to silence this warning.

```
correlation_matrix = df_encoded.corr()
```



## Gráfico de Barras por grupo canino

O gráfico a seguir ajuda a ter uma noção da quantidade de cada grupo canino pelas suas qualidades de pelagem, treinamento, energia, comportamento e etc.

```
In [ ]: # Agrupando valores
grooming_plt = df.groupby(['group', 'grooming_frequency_value']).size().reset_index
training_plt = df.groupby(['group', 'trainability_value']).size().reset_index(name='c
energy_plt = df.groupby(['group', 'energy_level_value']).size().reset_index(name='c
shedding_plt = df.groupby(['group', 'shedding_value']).size().reset_index(name='cou
demeanor_plt = df.groupby(['group', 'demeanor_value']).size().reset_index(name='cou

# Definir a ordem dos valores únicos de grooming_frequency_value
df_values = [0.2, 0.4, 0.6, 0.8, 1.0]

# Criar o gráfico de barras
fig, axes = plt.subplots(2, 3, figsize=(32, 16))
for i, value in enumerate(df_values):
    subset = grooming_plt[grooming_plt['grooming_frequency_value'] == value]
    axes[0, 0].bar(subset['group'], subset['count'], label=f'Grooming {value}')

    subset = training_plt[training_plt['trainability_value'] == value]
    axes[0, 1].bar(subset['group'], subset['count'], label=f'Training {value}')

    subset = energy_plt[energy_plt['energy_level_value'] == value]
    axes[0, 2].bar(subset['group'], subset['count'], label=f'Energy Level {value}')

    subset = shedding_plt[shedding_plt['shedding_value'] == value]
    axes[1, 0].bar(subset['group'], subset['count'], label=f'Shedding {value}')

    subset = demeanor_plt[demeanor_plt['demeanor_value'] == value]
    axes[1, 1].bar(subset['group'], subset['count'], label=f'Demeanor {value}')

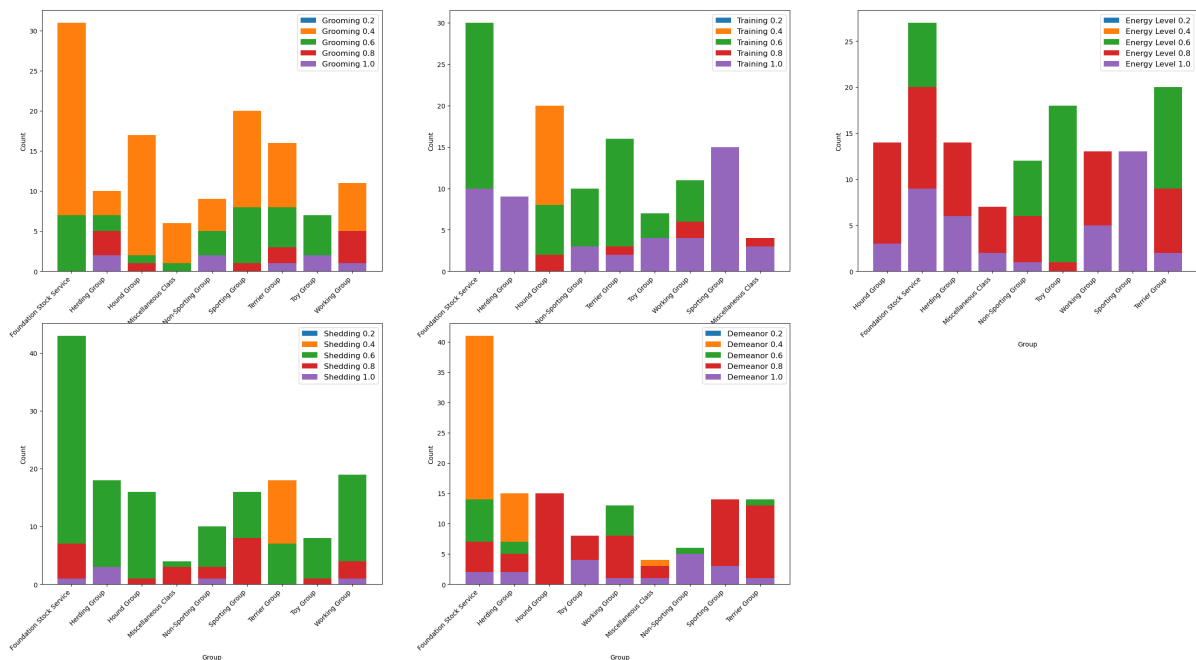
# Remover os subplots vazios
fig.delaxes(axes[1, 2])

# Configurar os eixos, títulos e Legendas
for ax in axes.flat:
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
    ax.set_xlabel('Group')
    ax.set_ylabel('Count')
    ax.legend(fontsize='large')

# Exibir a figura
plt.show()
```

C:\Users\saman\AppData\Local\Temp\ipykernel\_25256\3318937178.py:37: UserWarning: FixedFormatter should only be used together with FixedLocator

```
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
No artists with labels found to put in legend. Note that artists whose label start
with an underscore are ignored when legend() is called with no argument.
```



## Distribuição de cães por grupo e Gráfico de dispersão

Com base no gráfico de barras acima e a contagem de cães por grupo é desbalanceada, ou seja, há mais registro de cães de um determinado grupo que de outros.

Aqui podemos observar que há uma grande quantidade de cães do grupo **Foundation Stock Service** (classe majoritária) enquanto que os demais grupos tem metade ou menos da metade presente na classe majoritária, este é o caso dos cães que se enquadram na classe **Miscellaneous** (classe minoritária).

Realizar a técnica de *oversampling* pode vir a ser necessária considerando a diferença de objetos por grupo.

Também decidi plotar um gráfico de dispersão para observar de outra forma a dispersão de cães por grupo com base no seu peso mínimo e máximo. Aqui poderemos observar que cães do grupo **Toy** e **Terrier** são os de menor porte se comparado ao dos grupos **Hound** e **Working**. Outros grupos caninos tem indivíduos que podem ser tanto de pequeno-médio a grande porte (caso dos cães do grupo **Herding** e **Foundation Stock Service**).

```
In [ ]: count_by_group = df['group'].value_counts()

# Exibe a contagem por grupo
print("GRUPO CANINO\t\t\t\t\t qtd")
print("-"*32)
print(count_by_group)
print("-"*32)
print(f"Número total de cães:\t\t\t\t\t {len(df)}")
```

GRUPO CANINO	qtd
Foundation Stock Service	68
Hound Group	32
Sporting Group	32
Terrier Group	31
Working Group	31
Herding Group	30
Toy Group	21
Non-Sporting Group	21
Miscellaneous Class	11

Name: group, dtype: int64

---

Número total de cães: 277

```
In [ ]: # Cria uma lista de cores para cada grupo de cão
cores = {
    'Foundation Stock Service': '#182230',
    'Hound Group': '#d3ff41',
    'Sporting Group': '#934add',
    'Terrier Group': '#ea6c81',
    'Working Group': '#90be6d',
    'Herding Group': '#a6eaf6',
    'Toy Group': '#f7a54d',
    'Non-Sporting Group': '#f6e488',
    'Miscellaneous Class': '#81bdc3'
}

# Cria uma figura e eixos para o gráfico
fig, ax = plt.subplots(figsize=(16, 8))

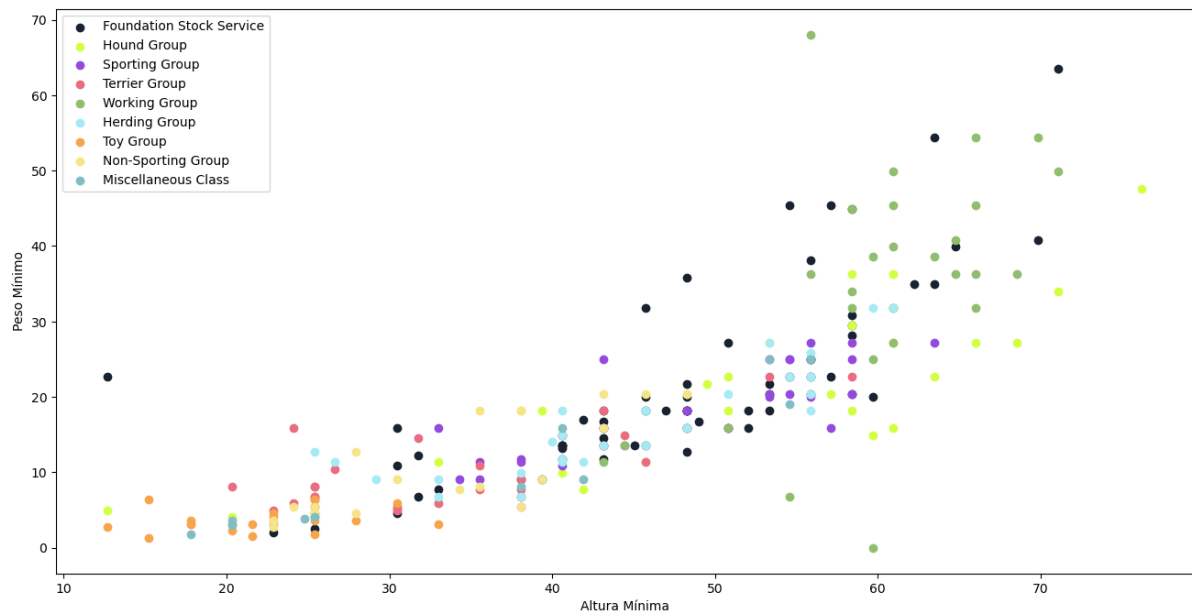
# Itera sobre cada grupo de cão
for grupo, cor in cores.items():
    # Filtra os dados apenas para o grupo de cão atual
    dados_grupo = df[df['group'] == grupo]

    # Plota o gráfico de dispersão para o grupo de cão atual
    ax.scatter(dados_grupo['min_height'], dados_grupo['min_weight'], color=cor, label=grupo)

# Define os rótulos dos eixos
ax.set_xlabel('Altura Mínima')
ax.set_ylabel('Peso Mínimo')

# Adiciona uma Legenda
ax.legend()

# Exibe o gráfico de dispersão
plt.show()
```



### Questão 3

Defina um pipeline de redução de dimensionalidade, aplique e justifique sua escolha. Lembre de comparar a base reduzida com a original (ou mais próximo dela);

### Resolução

O **PCA (Principal Component Analysis)** é uma técnica de redução utilizada para transformar um conjunto de variáveis correlacionadas (variáveis originais) em um novo conjunto de variáveis não correlacionadas (novos componentes principais) através de uma combinação linear. Assim, o PCA busca capturar a maior parte da variância dos dados em um menor número de dimensões.

Antes de realizar o redimensionamento, o dataframe possuía originalmente 14 colunas(15 se contarmos com a coluna de popularidade). A seguir foi utilizado o valor 9, mas o melhor valor de redução é aquele que melhor se aplica as métricas do modelo (acurácia, precisão e sensibilidade, vide questão 4).

A redução de dimensionalidade foi testada na questão 4 com diversos valores de `n_components` na aplicação do pipeline de redução.



```
In [ ]: # realizando uma cópia do dataframe pré-processado
doggies = df.copy()

# Criar uma instância do LabelEncoder
label_encoder = LabelEncoder()
label_encoder.fit(temperament_values_sorted) # Ajustar o codificador com a lista de
doggies['temperament_1_encoded'] = label_encoder.transform(doggies['temperament_1'])
doggies['temperament_2_encoded'] = label_encoder.transform(doggies['temperament_2'])
doggies['temperament_3_encoded'] = label_encoder.transform(doggies['temperament_3'])

# del doggies['popularity']
del doggies['temperament_1']
del doggies['temperament_2']
del doggies['temperament_3']

doggies.head()
```

```
Out[ ]: 
```

	min_height	max_height	min_weight	max_weight	min_expectancy	max_expectancy	group
0	22.86	29.21	3.175147	4.535924	12.0	15.0	Toy Group
1	63.50	68.58	22.679619	27.215542	12.0	15.0	Hound Group
2	58.42	58.42	22.679619	31.751466	11.0	14.0	Terrier Group
3	60.96	71.12	31.751466	58.967008	10.0	13.0	Working Group
4	58.42	63.50	34.019428	38.555351	10.0	14.0	Working Group

```
In [ ]: # Definindo atributo de entrada e saída
x = doggies.drop('group', axis=1) # Remover a coluna original 'group'
x = RobustScaler().fit_transform(x)

# Criar uma instância do LabelEncoder
label_encoder = LabelEncoder()

# Aplicar o Label encoding à coluna 'group'
y = label_encoder.fit_transform(doggies['group'])

# Separando dados de treinamento (20% para testes e 80% para treinamento)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.2, random_st
```

```
In [ ]: # Definir o pipeline k-NN
npca = 9
pipeline = Pipeline([
    ('scaling', StandardScaler()), # Pré-processamento: normalização dos dados
    # ('feature_selection', SelectKBest(k=10)), # Seleção das 10 melhores features
    ('dimensionality_reduction', PCA(n_components = npca)), # Redução de dimension
])

x_train_df = pd.DataFrame(x_train)

# Aplicar o pipeline aos dados de treinamento
x_train_reduced = pipeline.fit_transform(x_train)
x_train_reduced_df = pd.DataFrame(x_train_reduced)

# Verificar as dimensões dos dados originais e reduzidos
print(">>> Dataframe original")
print("Dimensões dos dados originais:", x_train.shape)
print(x_train_df.head())
print()
print("-"*100)
print("\n>>> Dataframe reduzido")
print("Dimensões dos dados reduzidos:", x_train_reduced.shape)
print(x_train_reduced_df.head())
```

```
>>> Dataframe original
Dimensões dos dados originais: (221, 14)
      0      1      2      3      4      5      6      7      8      9     10  \
0  0.694444  0.545455  2.03125  1.630435 -2.5 -2.5 -0.5  0.0  1.0  0.0  0.0
1  0.527778  0.500000  0.46875  0.217391  0.0  0.5 -0.5 -1.0  1.0 -0.5  1.0
2  0.361111  0.545455  0.31250  0.543478 -0.5 -1.0  0.0  0.0  1.0  0.0  0.5
3  0.583333  0.636364  0.93750  0.760870 -1.0  0.0 -0.5  0.0  0.0 -1.0 -0.5
4  1.138889  1.000000  1.25000  1.195652 -2.0 -1.5 -0.5  0.0  0.0 -0.5  0.0

      11      12      13
0 -0.839286  0.265625 -0.794118
1  0.000000  0.859375  1.029412
2  1.178571 -0.406250  0.838235
3 -0.803571 -0.234375  0.000000
4  0.071429 -0.390625  0.426471
```

```
-----

>>> Dataframe reduzido
Dimensões dos dados reduzidos: (221, 9)
      0      1      2      3      4      5      6  \
0  4.359974 -0.183548  1.084557 -0.259885 -0.457721  2.331020  1.102765
1  0.262496  2.257741 -1.682677  0.854942 -0.012544  0.609793  0.632791
2  1.218947  1.110797  0.023102 -0.590658  1.392439 -0.862373 -0.935517
3  2.323866 -1.199752 -1.100005  0.414727 -1.426304 -0.127812  1.076344
4  3.855716 -0.277208  0.040207  0.171298  0.052088 -0.020274  0.572959

      7      8
0 -0.008959 -0.236862
1 -1.763324  0.515962
2 -1.746800  0.730875
3 -0.228021  0.131123
4 -1.686238  0.473707
```

```
In [ ]: # salvando em um novo arquivo .csv  
doggies.to_csv('dogs2.csv', index=False)
```

## Questão 4 (Supervisionado)

Defina um pipeline de aplicação de pelo menos 2 métodos de aprendizado de máquina supervisionado e da variação de ao menos 1 parâmetro deles, execute-o e justifique suas escolhas (sobre os métodos, parâmetros e valores);

## Resolução

Para o método k-NN

**Em relação ao treinamento utilizando o método k-NN\*\* (sem oversampling)**

A tabela a seguir apresenta o valor da acurácia (com e sem pipeline\*) para cada valor de k utilizando o método k-NN. Dentre os valores obtidos, a melhor acurácia, sem a aplicação do pipeline de redução, foi para **k = 1**. Enquanto que a melhor acurácia, utilizando o pipeline de redução para 12 componentes, foi para **k = 1**.

k-nn	Acurácia (s/ pipeline)	Acurácia (PCA = 5)	Acurácia (PCA = 7)	Acurácia (PCA = 9)	Acurácia (PCA = 11)	Acurácia (PCA = 12)
1	0.44643	0.33929	0.42857	0.42857	0.51786	0.53571
3	0.39286	0.33929	0.33929	0.42857	0.35714	0.35714
5	0.35714	0.41071	0.37500	0.35714	0.39286	0.41071
7	0.41071	0.37500	0.30357	0.35714	0.46429	0.46429
9	0.35714	0.41071	0.37500	0.39286	0.48214	0.51786

**Em relação ao treinamento utilizando o método k-NN\*\* (com oversampling)**

A tabela a seguir apresenta o valor da acurácia (com e sem pipeline\*) para cada valor de k utilizando o método k-NN. Dentre os valores obtidos, a melhor acurácia, sem a aplicação do pipeline de redução, foi também para **k = 1**. Enquanto que a melhor acurácia, utilizando o pipeline de redução para 12 componentes, também foi para **k = 1**.

k-nn	Acurácia (s/ pipeline)	Acurácia (PCA = 5)	Acurácia (PCA = 7)	Acurácia (PCA = 9)	Acurácia (PCA = 11)	Acurácia (PCA = 12)
1	0.73171	0.70732	0.78049	0.85366	0.85366	0.86179
3	0.61789	0.64228	0.61789	0.67480	0.70732	0.71545
5	0.56911	0.60976	0.53659	0.60163	0.60976	0.60163
7	0.52033	0.58537	0.55285	0.60163	0.54472	0.54472
9	0.50407	0.49593	0.47154	0.53659	0.54472	0.52033

Para o método Random Forest

**Em relação ao treinamento utilizando o método de Random Forest (sem oversampling)**

A tabela a seguir apresenta o valor da acurácia (com e sem pipeline\*) para cada valor de **n\_estimators** utilizando o método Random Forest. Dentre os valores obtidos, a melhor acurácia, sem a aplicação do pipeline de redução, foi para **n\_estimators = 100**. Enquanto que a melhor acurácia, utilizando o pipeline de redução para 12 componentes, também foi para **n\_estimators = 100**.

n_estimators	Acurácia (s/ pipeline)	Acurácia (PCA = 5)	Acurácia (PCA = 7)	Acurácia (PCA = 9)	Acurácia (PCA = 11)	Acurácia (PCA = 12)
25	0.46429	0.39286	0.33929	0.42857	0.35714	0.42857
50	0.53571	0.35714	0.44643	0.44643	0.46429	0.41071
100	0.53571	0.33929	0.37500	0.41071	0.46429	0.50000
125	0.46429	0.41071	0.46429	0.41071	0.44643	0.44643
150	0.50000	0.33929	0.37500	0.46429	0.51786	0.44643

### Em relação ao treinamento utilizando o método de Random Forest (com oversampling)

A tabela a seguir apresenta o valor da acurácia (com e sem pipeline\*) para cada valor de `n_estimators` utilizando o método Random Forest. Dentre os valores obtidos, a melhor acurácia, sem a aplicação do pipeline de redução, foi para `n_estimators = 50`. Enquanto que a melhor acurácia, utilizando o pipeline de redução para 12 componentes, foi para `n_estimators = 100`.

n_estimators	Acurácia (s/ pipeline)	Acurácia (PCA = 5)	Acurácia (PCA = 7)	Acurácia (PCA = 9)	Acurácia (PCA = 11)	Acurácia (PCA = 12)
25	0.73984	0.62602	0.70732	0.73171	0.74797	0.78049
50	0.80488	0.65041	0.77236	0.74797	0.79675	0.82114
100	0.79675	0.68293	0.74797	0.78049	0.79675	0.83740
125	0.80488	0.67480	0.74797	0.78862	0.75610	0.77236
150	0.80488	0.69919	0.73171	0.79675	0.78862	0.80488

### Dados para Teste

```

In [ ]: # Os objetos de entrada são os 10 primeiros objetos da base de dados original (após

# Definir os dados de teste
xd_test = np.array([
    [22.86, 29.21, 3.17514659, 4.535923700000001, 12.0, 15.0, 0.6, 0.6, 0.6, 0.8, 1.0, 25, 48, 49]
    [63.5, 68.58, 22.6796185, 27.2155422, 12.0, 15.0, 0.8, 0.2, 0.8, 0.2, 0.2, 36, 101, 9], # Houn
    [58.42, 58.42, 22.6796185, 31.751465900000003, 11.0, 14.0, 0.6, 0.4, 0.6, 1.0, 0.8, 50, 23, 27]
    [60.96, 71.12, 31.751465900000003, 58.9670081, 10.0, 13.0, 0.8, 0.6, 0.8, 1.0, 0.6, 27, 36, 10]
    [58.42, 63.5, 34.01942775, 38.55535145, 10.0, 14.0, 0.6, 0.6, 0.8, 0.4, 0.8, 3, 78, 93], # Wor
    [50.8, 63.5, 27.2155422, 45.359237, 10.0, 12.0, 0.2, 0.6, 0.8, 0.6, 0.6, 78, 112, 131], # Foun
    [58.42, 66.04, 20.41165665, 29.48350405, 11.0, 12.0, 0.2, 0.4, 0.8, 0.6, 0.6, 123, 80, 117], #
    [22.86, 48.26, 2.72155422, 15.875732950000002, 13.0, 15.0, 0.4, 0.6, 0.8, 1.0, 1.0, 93, 92, 11]
    [53.34, 63.5, 27.2155422, 31.751465900000003, 11.0, 13.0, 0.2, 0.6, 0.8, 0.4, 0.8, 66, 40, 125]
    [30.48, 40.64, 5.4431084400000005, 7.25747792, 14.0, 16.0, 0.4, 0.4, 0.6, 0.6, 0.6, 41, 5, 29]

# Novos dados de teste, um cão de cada grupo
[46.99,      58.42,   18.14,   24.95,   12,      14,   0.2,      0.6,      1,      0.6
[33.02,      45.72,    9.07,   18.14,   12,      13,      0.8,      1,      0.8,      1,
[71.12,      81.28,   34.02,   49.9,    8,       11,      0.2,      0.6,      0.6,      0.4
[25.4,       30.48,    4.08,    7.71,   12,      15,      0.2,      0.6,      0.8,      0.8
[25.4,       25.4,     4.08,    6.8,    12,      15,      0.4,      0.6,      0.4,      0.6
[34.29, 39.37,      9.07,   13.61,   10,      14,      0.8,      0.4,      0.6,      1,
[25.4,       33.02,    6.35,   10.89,   12,      15,      0.4,      0.4,      0.6,      0.6
[22.86,      25.4,     3.63,    6.35,   10,      12,      0.6,      0.6,      0.6,      0.4
[44.45,      49.53,   13.61,   22.68,   13,      16,      0.6,      0.2,      0.6,      0.4
[55.88,      66.04,   22.68,   34.02,   12,      14,      0.6,      0.6,      0.6,      0.8
])

```

## Resultados

As tabelas a seguir são um comparativo entre as previsões realizadas pelos modelos que utilizam o método k-NN e o método Random Forest, tanto com ou sem a aplicação do pipeline de redução, como também com ou sem a aplicação de *oversampling* na base de dados.

Os dados de testes utilizados são extras e pertencem a lista `xd_test`, da célula de código anterior, e ela contém 20 entradas.

Também é possível comparar as previsões pelas matrizes de confusão para cada modelo e para cada uma das situações (com ou sem pipeline, com ou sem *oversampling*).

Com base no teste realizado e registrado aqui, podemos observar que o método Random Forest classificou corretamente um cachorro a mais que o método k-NN, onde ambos estavam com *oversampling* e com a aplicação do pipeline de redução.

### Método k-NN

Após aplicar *oversampling*, obtivemos uma melhora nos acertos das previsões, pois diminuimos a tendência do modelo 'optar por prever' um grupo específico. Se formos observar os resultados ***sem oversampling*** (independente de não ter ou ter aplicado o pipeline de redução), podemos observar que o modelo tinha uma tendência de classificar a maioria dos cães como pertencente do grupo Foundation Stock Service (classe majoritária).

Caso contrário, ele *previa* que aquele objeto seria do grupo *Hound* ou *Working* (outros dois grupos mais populosos depois de Foundation Stock Service).

#	Grupo Verdadeiro	s/ pipeline s/ oversampling	c/ pipeline s/ oversampling	s/ pipeline c/ oversampling	c/ pipeline c/ oversampling
1	Toy	x	x	✓	✓
2	Hound	x	x	✓	✓
3	Terrier	x	x	x	✓
4	Working	x	x	✓	✓
5	Working	x	x	✓	✓
6	Foundation	✓	✓	✓	✓
7	Hound	x	✓	✓	✓
8	Non-sporting	x	x	✓	✓
9	Hound	x	x	✓	✓
10	Terrier	x	x	✓	✓
11	Foundation	✓	✓	x	x
12	Herding	v	v	v	✓

12	Herding	x	x	x	x
13	Hound	x	x	✓	✓
14	Misc	x	x	✓	✓
15	Non-sporting	x	x	✓	✓
16	Sporting	x	x	✓	✓
17	Terrier	x	x	✓	✓
18	Toy	x	x	✓	✓
19	Working	x	x	✓	✓
20	Herding	x	x	x	x

### Método Random Forest

#	Grupo Verdadeiro	s/ pipeline s/ oversampling	c/ pipeline s/ oversampling	s/ pipeline c/ oversampling	c/ pipeline c/ oversampling
1	Toy	x	x	✓	✓
2	Hound	x	x	✓	✓
3	Terrier	x	x	x	x
4	Working	✓	x	✓	✓
5	Working	✓	x	✓	✓
6	Foundation	x	✓	✓	✓
7	Hound	x	x	x	✓
8	Non-sporting	x	x	✓	✓
9	Hound	x	✓	✓	✓
10	Terrier	x	x	✓	✓
11	Foundation	x	✓	x	✓
12	Herding	x	x	✓	✓
13	Hound	x	x	✓	✓
14	Misc	x	x	✓	✓
15	Non-sporting	x	x	✓	✓
16	Sporting	x	x	✓	✓
17	Terrier	x	x	✓	✓
18	Toy	x	x	✓	✓
19	Working	✓	x	✓	✓
20	Herding	x	x	✓	✓

### Método k-NN

**Sobre o método k-NN** é um algoritmo é amplamente utilizado em problemas



**Sobre o método K-NN:** um algoritmo é amplamente utilizado em problemas de classificação de padrões, classificação de imagens, reconhecimento de caracteres manuscritos, processamento de fala e análise de dados biomédicos.

```
In [ ]: k_values = [1, 3, 5, 7, 9]
npca_list = [5, 7, 9, 11, 12]

print("SEM OVERSAMPLING")
print("-"*32)
for k in k_values:
    # Sem o pipeline
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(x_train, y_train)
    y_pred = knn_classifier.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    confusionm = confusion_matrix(y_test, y_pred)
    print("Sem pipeline")
    print(f"k = {k}: Acurácia = {accuracy:.5f}")
    print(f"k = {k}: Precisão = {precision:.5f}")
    print(f"k = {k}: Sensibilidade = {recall:.5f}")
    print(f'Matriz de Confusão: \n {confusionm}\n')

    # Com o pipeline
    for i in npca_list:

        pipeline = Pipeline([
            ('scaling', StandardScaler()),
            ('dimensionality_reduction', PCA(n_components=i)),
            ('classification', KNeighborsClassifier(n_neighbors=k))
        ])
        pipeline.fit(x_train, y_train)
        x_test_reduced = pipeline['dimensionality_reduction'].transform(x_test)
        y_pred = pipeline.predict(x_test)
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='macro')
        recall = recall_score(y_test, y_pred, average='macro')
        confusionm = confusion_matrix(y_test, y_pred)
        print(f"Com pipeline, n_components PCA = {i}")
        print(f"k = {k}: Acurácia = {accuracy:.5f}")
        print(f"k = {k}: Precisão = {precision:.5f}")
        print(f"k = {k}: Sensibilidade = {recall:.5f}")
        print(f'Matriz de Confusão: \n {confusionm}\n')

print("-"*100)
```

## SEM OVERSAMPLING

-----

Sem pipeline

k = 1: Acurácia = 0.68293

k = 1: Precisão = 0.67932

k = 1: Sensibilidade = 0.69425

Matriz de Confusão:

```
[[ 5  2  2  1  0  3  2  0  2]
 [ 1  5  4  0  0  0  2  0  0]
 [ 0  1  6  0  0  0  0  1  0]
 [ 1  1  0 11  0  0  2  0  0]
 [ 1  0  1  0 11  1  0  0  0]
 [ 0  2  1  1  1 11  0  0  0]
 [ 0  0  3  1  1  0 12  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  0  0  0  0  0  0  7]]
```

Com pipeline, n\_components PCA = 5

k = 1: Acurácia = 0.68293

k = 1: Precisão = 0.66978

k = 1: Sensibilidade = 0.67836

Matriz de Confusão:

```
[[ 4  1  3  1  4  2  0  0  2]
 [ 1  5  4  0  0  1  0  0  1]
 [ 0  0  7  0  0  0  1  0  0]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  0  3  0  8  0  1  0  1]
 [ 2  0  1  1  0 12  0  0  0]
 [ 1  0  1  0  0  0 14  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  1  0  0  2  0  0  4]]
```

Com pipeline, n\_components PCA = 7

k = 1: Acurácia = 0.76423

k = 1: Precisão = 0.77635

k = 1: Sensibilidade = 0.75607

Matriz de Confusão:

```
[[11  0  1  2  1  1  0  0  1]
 [ 2  6  4  0  0  0  0  0  0]
 [ 0  0  7  0  0  0  1  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  2  1  9  0  1  0  0]
 [ 2  1  1  0  0 12  0  0  0]
 [ 2  1  0  0  0  0 13  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  2  0  0  0  0  0  5]]
```

Com pipeline, n\_components PCA = 9

k = 1: Acurácia = 0.77236

k = 1: Precisão = 0.77020

k = 1: Sensibilidade = 0.78052

Matriz de Confusão:

```
[[ 7  0  1  2  2  3  0  1  1]
 [ 0  6  3  0  2  1  0  0  0]
 [ 0  0  7  0  0  0  1  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 1  0  0  0 11  0  2  0  0]
 [ 0  1  1  0  0 13  1  0  0]
 [ 1  1  0  0  1  0 13  1  0]
```

```
[ 0  0  0  0  0  0  0 16  0]
[ 0  0  0  0  0  1  0  0  7]]
```

Com pipeline, n\_components PCA = 11

k = 1: Acurácia = 0.77236

k = 1: Precisão = 0.77098

k = 1: Sensibilidade = 0.77548

Matriz de Confusão:

```
[[ 7  0  1  3  1  4  0  0  1]
 [ 0  7  3  1  0  1  0  0  0]
 [ 1  0  6  0  0  0  1  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  0  0 11  0  2  0  0]
 [ 0  1  1  1  0 12  1  0  0]
 [ 1  1  0  0  1  0 14  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  0  0  0  0  1  0  0  7]]
```

Com pipeline, n\_components PCA = 12

k = 1: Acurácia = 0.78862

k = 1: Precisão = 0.79503

k = 1: Sensibilidade = 0.79731

Matriz de Confusão:

```
[[ 7  0  1  4  1  3  0  0  1]
 [ 0  7  3  1  0  1  0  0  0]
 [ 0  0  7  0  0  0  1  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  0  0 12  0  1  0  0]
 [ 0  1  1  1  0 12  1  0  0]
 [ 1  1  1  0  0  0 14  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  0  0  0  0  1  0  0  7]]
```

-----

Sem pipeline

k = 3: Acurácia = 0.57724

k = 3: Precisão = 0.55163

k = 3: Sensibilidade = 0.56582

Matriz de Confusão:

```
[[ 5  1  1  1  2  4  1  0  2]
 [ 1  7  1  0  2  0  1  0  0]
 [ 1  1  6  0  0  0  0  0  0]
 [ 1  0  0 12  0  0  2  0  0]
 [ 1  0  1  0  9  1  1  1  0]
 [ 2  2  2  2  0  7  0  0  1]
 [ 2  0  3  0  1  0 11  0  0]
 [ 0  1  0  1  0  0  1 13  0]
 [ 3  3  0  0  0  1  0  0  1]]
```

Com pipeline, n\_components PCA = 5

k = 3: Acurácia = 0.60976

k = 3: Precisão = 0.65368

k = 3: Sensibilidade = 0.60828

Matriz de Confusão:

```
[[ 6  2  5  1  2  1  0  0  0]
 [ 1  8  2  0  0  1  0  0  0]
 [ 0  1  5  0  0  0  1  0  1]
 [ 0  2  0 13  0  0  0  0  0]
 [ 1  0  2  1  7  0  1  2  0]]
```

```
[ 2  0  3  0  2  8  1  0  0]
[ 2  0  3  0  1  0 10  1  0]
[ 0  0  1  0  1  0  0 14  0]
[ 1  0  3  0  0  0  0  0  4]]
```

Com pipeline, n\_components PCA = 7

k = 3: Acurácia = 0.64228

k = 3: Precisão = 0.65794

k = 3: Sensibilidade = 0.62922

Matriz de Confusão:

```
[[ 9  2  1  4  1  0  0  0  0]
 [ 3  5  3  0  0  1  0  0  0]
 [ 0  0  5  0  0  0  2  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  2  1  7  0  1  2  0]
 [ 1  1  3  1  0 10  0  0  0]
 [ 2  1  2  0  1  0  9  2  0]
 [ 0  0  0  0  1  0  0 15  0]
 [ 2  2  0  0  0  0  0  0  4]]
```

Com pipeline, n\_components PCA = 9

k = 3: Acurácia = 0.69106

k = 3: Precisão = 0.69780

k = 3: Sensibilidade = 0.67818

Matriz de Confusão:

```
[[11  0  2  2  0  2  0  0  0]
 [ 1  5  3  1  2  0  0  0  0]
 [ 0  0  5  0  0  0  2  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  0  2  9  0  1  1  0]
 [ 1  2  2  2  1  8  0  0  0]
 [ 1  0  3  0  1  0 11  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  1  0  0  1  0  0  5]]
```

Com pipeline, n\_components PCA = 11

k = 3: Acurácia = 0.69106

k = 3: Precisão = 0.69752

k = 3: Sensibilidade = 0.68305

Matriz de Confusão:

```
[[ 7  1  2  4  1  2  0  0  0]
 [ 0  7  3  1  1  0  0  0  0]
 [ 1  0  5  0  0  0  1  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  0  3  8  0  1  1  0]
 [ 1  2  2  1  0 10  0  0  0]
 [ 1  1  1  0  1  0 12  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  1  0  0  1  0  0  5]]
```

37 of 91

Com pipeline, n\_components PCA = 12

k = 3: Acurácia = 0.69106

k = 3: Precisão = 0.70931

k = 3: Sensibilidade = 0.69271

Matriz de Confusão:

```
[[ 8  1  2  4  0  2  0  0  0]
 [ 0  8  2  1  1  0  0  0  0]
 [ 0  0  6  0  0  0  1  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  0  3  8  0  1  1  0]
 [ 1  3  1  2  0  9  0  0  0]
 [ 1  1  3  0  2  0 10  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  1  0  0  1  0  0  5]]
```

-----

Sem pipeline

k = 5: Acurácia = 0.52846

k = 5: Precisão = 0.52095

k = 5: Sensibilidade = 0.54441

Matriz de Confusão:

```
[[ 2  2  2  1  2  5  1  0  2]
 [ 1  7  2  0  1  0  1  0  0]
 [ 1  1  5  0  0  0  0  1  0]
 [ 2  1  0  9  1  2  0  0  0]
 [ 1  0  0  1  9  1  1  1  0]
 [ 1  2  5  2  0  4  1  0  1]
 [ 2  0  2  0  1  0 12  0  0]
 [ 0  1  0  1  1  0  1 12  0]
 [ 0  2  0  0  0  1  0  0  5]]
```

Com pipeline, n\_components PCA = 5

k = 5: Acurácia = 0.51220

k = 5: Precisão = 0.54265

k = 5: Sensibilidade = 0.50354

Matriz de Confusão:

```
[[ 4  2  7  1  1  2  0  0  0]
 [ 1  5  3  1  0  2  0  0  0]
 [ 0  0  4  0  0  0  1  1  2]
 [ 1  4  0 10  0  0  0  0  0]
 [ 0  1  2  0  7  0  2  2  0]
 [ 1  1  3  0  2  8  1  0  0]
 [ 2  0  2  1  1  0 10  1  0]
 [ 0  0  0  0  4  0  0 12  0]
 [ 0  1  3  0  0  1  0  0  3]]
```

Com pipeline, n\_components PCA = 7

k = 5: Acurácia = 0.58537

k = 5: Precisão = 0.58317

k = 5: Sensibilidade = 0.56939

Matriz de Confusão:

```
[[ 6  0  4  3  2  1  0  1  0]
 [ 3  3  2  1  1  1  0  0  1]
 [ 0  0  4  0  1  0  2  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 0  1  2  2  7  0  1  1  0]
 [ 1  2  1  2  1  9  0  0  0]
 [ 2  0  2  0  1  0 10  2  0]
 [ 0  0  0  0  1  0  0 15  0]]
```

```
[ 0 2 2 0 0 0 0 0 4]]
```

Com pipeline, n\_components PCA = 9

k = 5: Acurácia = 0.64228

k = 5: Precisão = 0.67594

k = 5: Sensibilidade = 0.63833

Matriz de Confusão:

```
[[ 4 0 6 3 2 2 0 0 0]
 [ 0 8 2 1 1 0 0 0 0]
 [ 0 0 5 0 0 0 2 0 1]
 [ 0 1 0 14 0 0 0 0 0]
 [ 0 1 0 3 8 0 1 1 0]
 [ 1 0 2 2 1 10 0 0 0]
 [ 1 0 4 0 1 0 10 1 0]
 [ 0 0 0 0 0 0 0 16 0]
 [ 0 1 2 0 0 1 0 0 4]]
```

Com pipeline, n\_components PCA = 11

k = 5: Acurácia = 0.63415

k = 5: Precisão = 0.67152

k = 5: Sensibilidade = 0.61899

Matriz de Confusão:

```
[[ 6 1 5 3 1 1 0 0 0]
 [ 0 6 2 2 2 0 0 0 0]
 [ 0 1 5 0 0 0 1 0 1]
 [ 0 1 0 14 0 0 0 0 0]
 [ 0 1 0 3 8 0 1 1 0]
 [ 1 1 2 2 0 10 0 0 0]
 [ 1 0 4 0 1 0 10 1 0]
 [ 0 0 0 0 0 0 0 16 0]
 [ 0 1 3 0 0 1 0 0 3]]
```

Com pipeline, n\_components PCA = 12

k = 5: Acurácia = 0.64228

k = 5: Precisão = 0.69308

k = 5: Sensibilidade = 0.63520

Matriz de Confusão:

```
[[ 5 1 4 4 1 2 0 0 0]
 [ 0 7 2 1 2 0 0 0 0]
 [ 0 1 5 0 0 0 1 0 1]
 [ 0 1 0 14 0 0 0 0 0]
 [ 0 1 0 2 8 0 1 2 0]
 [ 0 2 1 3 0 9 1 0 0]
 [ 0 0 5 0 1 0 11 0 0]
 [ 0 0 0 0 0 0 0 16 0]
 [ 0 1 2 0 0 1 0 0 4]]
```

-----

Sem pipeline

k = 7: Acurácia = 0.50407

k = 7: Precisão = 0.50842

k = 7: Sensibilidade = 0.51184

Matriz de Confusão:

```
[[ 4 1 1 2 1 6 1 0 1]
 [ 0 7 3 0 1 0 1 0 0]
 [ 3 1 4 0 0 0 0 0 0]
 [ 1 1 0 11 1 0 1 0 0]
 [ 1 0 0 1 9 1 1 1 0]
 [ 0 4 6 1 0 3 1 0 1]]
```

```
[ 2  1  2  0  4  0  7  1  0]
[ 0  1  0  1  1  0  0 13  0]
[ 2  1  0  0  0  1  0  0  4]]
```

Com pipeline, n\_components PCA = 5

k = 7: Acurácia = 0.47154

k = 7: Precisão = 0.48318

k = 7: Sensibilidade = 0.45051

Matriz de Confusão:

```
[[ 4  2  6  1  1  3  0  0  0]
 [ 2  4  3  2  0  1  0  0  0]
 [ 0  1  3  0  0  0  2  0  2]
 [ 2  2  0 11  0  0  0  0  0]
 [ 1  1  2  0  4  0  3  3  0]
 [ 1  1  3  1  1  8  1  0  0]
 [ 2  1  2  1  1  0  9  1  0]
 [ 0  0  0  0  3  0  0 13  0]
 [ 1  2  3  0  0  0  0  0  2]]
```

Com pipeline, n\_components PCA = 7

k = 7: Acurácia = 0.53659

k = 7: Precisão = 0.56425

k = 7: Sensibilidade = 0.52453

Matriz de Confusão:

```
[[ 4  1  6  1  4  0  1  0  0]
 [ 1  8  2  0  1  0  0  0  0]
 [ 0  0  4  0  1  0  2  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 1  1  2  2  4  0  2  2  0]
 [ 2  2  1  1  1  9  0  0  0]
 [ 2  2  2  0  1  0  8  2  0]
 [ 0  1  0  0  1  0  2 12  0]
 [ 0  2  3  1  0  0  0  0  2]]
```

Com pipeline, n\_components PCA = 9

k = 7: Acurácia = 0.57724

k = 7: Precisão = 0.59613

k = 7: Sensibilidade = 0.56186

Matriz de Confusão:

```
[[ 5  2  4  1  3  2  0  0  0]
 [ 0  6  2  2  2  0  0  0  0]
 [ 0  0  5  0  0  0  2  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  0  1  1  6  0  3  2  0]
 [ 0  1  2  2  1  9  1  0  0]
 [ 1  0  4  0  1  0 10  1  0]
 [ 1  0  0  0  0  0  1 14  0]
 [ 0  1  3  1  0  1  0  0  2]]
```

Com pipeline, n\_components PCA = 11

k = 7: Acurácia = 0.59350

k = 7: Precisão = 0.61338

k = 7: Sensibilidade = 0.57956

Matriz de Confusão:

```
[[ 6  0  4  3  1  3  0  0  0]
 [ 0  5  3  2  2  0  0  0  0]
 [ 0  1  5  0  0  0  1  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  1  0  2  6  0  2  2  0]]
```



```
[ 1  2  1  2  1  9  0  0  0]
[ 1  0  3  0  0  0 11  2  0]
[ 0  0  0  0  0  0  2 14  0]
[ 0  2  3  0  0  0  0  0  3]]
```

Com pipeline, n\_components PCA = 12

k = 7: Acurácia = 0.56911

k = 7: Precisão = 0.60042

k = 7: Sensibilidade = 0.55996

Matriz de Confusão:

```
[[ 4  1  4  3  2  2  0  0  1]
 [ 0  5  3  2  2  0  0  0  0]
 [ 0  1  5  0  0  0  1  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  1  0  2  6  0  2  2  0]
 [ 0  2  1  3  2  8  0  0  0]
 [ 0  0  4  1  1  0 10  1  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  2  3  0  0  0  0  0  3]]
```

-----

Sem pipeline

k = 9: Acurácia = 0.43902

k = 9: Precisão = 0.43549

k = 9: Sensibilidade = 0.44083

Matriz de Confusão:

```
[[ 3  1  3  1  1  5  0  1  2]
 [ 0  7  1  0  2  0  2  0  0]
 [ 3  0  3  0  1  0  0  1  0]
 [ 2  2  0  8  1  1  1  0  0]
 [ 1  2  0  1  8  0  1  1  0]
 [ 0  3  5  2  0  3  1  1  1]
 [ 2  1  2  0  5  0  6  1  0]
 [ 0  0  1  1  1  0  0 13  0]
 [ 2  1  0  0  0  2  0  0  3]]
```

[illegible]

Com pipeline, n\_components PCA = 5

k = 9: Acurácia = 0.49593

k = 9: Precisão = 0.51857

k = 9: Sensibilidade = 0.47968

Matriz de Confusão:

```
[[ 3  3  7  1  1  2  0  0  0]
 [ 1  4  4  2  0  1  0  0  0]
 [ 0  3  3  0  0  0  1  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  0  3  1  4  0  2  3  0]
 [ 2  1  2  2  2  6  1  0  0]
 [ 1  1  2  0  1  0 10  2  0]
 [ 0  0  0  0  2  0  0 14  0]
 [ 1  0  3  0  0  1  0  0  3]]
```

Com pipeline, n\_components PCA = 7

k = 9: Acurácia = 0.54472

k = 9: Precisão = 0.56768

k = 9: Sensibilidade = 0.52516

Matriz de Confusão:

```
[[ 7  1  3  1  2  2  0  1  0]
 [ 2  6  4  0  0  0  0  0  0]
 [ 0  1  3  0  1  0  2  0  1]
 [ 0  0  1 14  0  0  0  0  0]
 [ 1  1  2  1  4  0  3  2  0]
 [ 1  2  1  1  1 10  0  0  0]
 [ 2  1  2  0  2  0  8  2  0]
 [ 0  1  0  0  0  0  3 12  0]
 [ 0  2  3  0  0  0  0  0  3]]
```

Com pipeline, n\_components PCA = 9

k = 9: Acurácia = 0.58537

k = 9: Precisão = 0.59536

k = 9: Sensibilidade = 0.57985

Matriz de Confusão:

```
[[ 6  0  3  3  2  3  0  0  0]
 [ 0  5  3  2  2  0  0  0  0]
 [ 0  0  5  0  0  0  2  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 1  1  0  1  5  0  3  3  0]
 [ 0  2  1  1  2 10  0  0  0]
 [ 1  1  3  0  1  0  9  2  0]
 [ 1  0  0  0  0  0  2 13  0]
 [ 0  1  2  0  0  1  0  0  4]]
```

Com pipeline, n\_components PCA = 11

k = 9: Acurácia = 0.56911

k = 9: Precisão = 0.57055

k = 9: Sensibilidade = 0.55301

Matriz de Confusão:

```
[[ 4  1  4  3  2  2  0  0  1]
 [ 0  5  4  1  2  0  0  0  0]
 [ 0  0  4  0  1  0  2  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 0  1  0  1  6  0  3  3  0]
 [ 0  2  1  2  2  9  0  0  0]
 [ 2  0  2  0  0  1 10  2  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  0  3  0  0  2  0  0  3]]
```

Com pipeline, n\_components PCA = 12

k = 9: Acurácia = 0.56098

k = 9: Precisão = 0.57756

k = 9: Sensibilidade = 0.54507

Matriz de Confusão:

```
[[ 4  1  4  4  1  2  0  1  0]
 [ 0  5  4  1  2  0  0  0  0]
 [ 0  0  4  0  1  0  2  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  1  0  1  5  0  3  3  0]
 [ 0  2  1  2  2  9  0  0  0]
 [ 1  0  3  0  1  1 10  1  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  0  3  0  0  2  0  0  3]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

Teste s/ pipeline, s/ oversampling

```
In [ ]: print("k-NN, Testando sem pipeline")

# Fazer previsões com o modelo k-NN
k = 1 # Escolha o valor de k desejado
knn_classifier = KNeighborsClassifier(n_neighbors=9)
knn_classifier.fit(x_train, y_train)
predictions = knn_classifier.predict(xd_test)
print(predictions)

predicted_groups = label_encoder.inverse_transform(predictions)
print(predicted_groups)
```

k-NN, Testando sem pipeline

```
[0 0 8 0 0 0 0 0 0 8 0 8 8 0 0 0 0 0 0 0]
['Foundation Stock Service' 'Foundation Stock Service' 'Working Group'
 'Foundation Stock Service' 'Foundation Stock Service'
 'Foundation Stock Service' 'Foundation Stock Service'
 'Foundation Stock Service' 'Foundation Stock Service' 'Working Group'
 'Foundation Stock Service' 'Working Group' 'Working Group'
 'Foundation Stock Service' 'Foundation Stock Service'
 'Foundation Stock Service' 'Foundation Stock Service'
 'Foundation Stock Service' 'Foundation Stock Service'
 'Foundation Stock Service']
```

Teste c/ pipeline, s/ oversampling

```

In [ ]: # Definir o pipeline
npca = 12
k = 1

# Ajustar o scaler nos dados de treinamento
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)

# Reduzir a dimensionalidade nos dados de treinamento
pca = PCA(n_components=npca)
x_train_reduced = pca.fit_transform(x_train_scaled)

# Criar o classificador KNN
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Treinar o classificador com os dados reduzidos
knn_classifier.fit(x_train_reduced, y_train)

# Aplicar o scaler e a redução de dimensionalidade nos dados de teste
x_test_scaled = scaler.transform(xd_test)
x_test_reduced = pca.transform(x_test_scaled)

# Fazer previsões com o classificador
y_pred = knn_classifier.predict(x_test_reduced)

# Imprimir as previsões
print("k-NN, Testando com pipeline")
print(y_pred)

# Converter as previsões de volta para os rótulos originais
predicted_groups = label_encoder.inverse_transform(y_pred)
print(predicted_groups)

k-NN, Testando com pipeline
[0 8 8 0 0 0 2 1 8 2 0 2 8 1 2 1 8 1 0 0]
['Foundation Stock Service' 'Working Group' 'Working Group'
 'Foundation Stock Service' 'Foundation Stock Service'
 'Foundation Stock Service' 'Hound Group' 'Herding Group' 'Working Group'
 'Hound Group' 'Foundation Stock Service' 'Hound Group' 'Working Group'
 'Herding Group' 'Hound Group' 'Herding Group' 'Working Group'
 'Herding Group' 'Foundation Stock Service' 'Foundation Stock Service']

```

Treinamento c/ Oversampling

```
In [ ]: from imblearn.over_sampling import SMOTE

# Realizando oversampling devido a diferença entre a quantidade de cães por grupo c
dcp = doggies.copy()

# Separa as colunas de features e o alvo
x = dcp.drop('group', axis=1)
y = dcp['group']

# Aplica o SMOTE para gerar dados sintéticos
smote = SMOTE()
x_oversampled, y_oversampled = smote.fit_resample(x, y)

# Cria um novo DataFrame com as features e o alvo oversampled
dados_oversampled = pd.DataFrame(x_oversampled, columns=x.columns)
dados_oversampled['group'] = y_oversampled

count_by_group_oversampled = dados_oversampled['group'].value_counts()

# Verifica o novo número de objetos após o oversampling
num_objetos_oversampled = len(dados_oversampled)

count_by_group = dcp['group'].value_counts()

# Exibe a contagem por grupo
print("GRUPO CANINO\t\t\t qtd")
print("-"*32)
print(count_by_group_oversampled)
print("-"*32)
print("Número total de cães após oversampling:", num_objetos_oversampled)
```

GRUPO CANINO	qtd
Toy Group	68
Hound Group	68
Terrier Group	68
Working Group	68
Foundation Stock Service	68
Non-Sporting Group	68
Sporting Group	68
Herding Group	68
Miscellaneous Class	68

Name: group, dtype: int64

Número total de cães após oversampling: 612

```
In [ ]: # Criar uma instância do LabelEncoder
label_encoder = LabelEncoder()

# Aplicar o Label encoding à coluna 'group' em dados_oversampled
dados_oversampled['group'] = label_encoder.fit_transform(dados_oversampled['group'])

# Separando dados de treinamento e testes
x_train, x_test, y_train, y_test = train_test_split(dados_oversampled.drop('group',

k_values = [1, 3, 5, 7, 9]
npca_list = [5, 7, 9, 11, 12]

print("COM OVERSAMPLING")
print("-"*32)
for k in k_values:
    # Sem o pipeline
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(x_train, y_train)
    y_pred = knn_classifier.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    confusionm = confusion_matrix(y_test, y_pred)
    print("Sem pipeline")
    print(f"k = {k}: Acurácia = {accuracy:.5f}")
    print(f"k = {k}: Precisão = {precision:.5f}")
    print(f"k = {k}: Sensibilidade = {recall:.5f}")
    print(f'Matriz de Confusão:\n{confusionm}\n')

    # Com o pipeline
    for i in npca_list:
        pipeline = Pipeline([
            ('scaling', StandardScaler()),
            ('dimensionality_reduction', PCA(n_components=i)),
            ('classification', KNeighborsClassifier(n_neighbors=k))
        ])
        pipeline.fit(x_train, y_train)
        X_test_reduced = pipeline['dimensionality_reduction'].transform(x_test)
        y_pred = pipeline.predict(X_test_reduced)
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='macro')
        recall = recall_score(y_test, y_pred, average='macro')
        confusionm = confusion_matrix(y_test, y_pred)
        print(f"Com pipeline, n_components PCA = {i}")
        print(f"k = {k}: Acurácia = {accuracy:.5f}")
        print(f"k = {k}: Precisão = {precision:.5f}")
        print(f"k = {k}: Sensibilidade = {recall:.5f}")
        print(f'Matriz de Confusão:\n{confusionm}\n')

print("-" * 100)
```

## COM OVERSAMPLING

-----

Sem pipeline

k = 1: Acurácia = 0.68293

k = 1: Precisão = 0.67932

k = 1: Sensibilidade = 0.69425

Matriz de Confusão:

```
[[ 5  2  2  1  0  3  2  0  2]
 [ 1  5  4  0  0  0  2  0  0]
 [ 0  1  6  0  0  0  0  1  0]
 [ 1  1  0 11  0  0  2  0  0]
 [ 1  0  1  0 11  1  0  0  0]
 [ 0  2  1  1  1 11  0  0  0]
 [ 0  0  3  1  1  0 12  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  0  0  0  0  0  0  7]]
```

Com pipeline, n\_components PCA = 5

k = 1: Acurácia = 0.68293

k = 1: Precisão = 0.66978

k = 1: Sensibilidade = 0.67836

Matriz de Confusão:

```
[[ 4  1  3  1  4  2  0  0  2]
 [ 1  5  4  0  0  1  0  0  1]
 [ 0  0  7  0  0  0  1  0  0]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  0  3  0  8  0  1  0  1]
 [ 2  0  1  1  0 12  0  0  0]
 [ 1  0  1  0  0  0 14  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  1  0  0  2  0  0  4]]
```

Com pipeline, n\_components PCA = 7

k = 1: Acurácia = 0.76423

k = 1: Precisão = 0.77635

k = 1: Sensibilidade = 0.75607

Matriz de Confusão:

```
[[11  0  1  2  1  1  0  0  1]
 [ 2  6  4  0  0  0  0  0  0]
 [ 0  0  7  0  0  0  1  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  2  1  9  0  1  0  0]
 [ 2  1  1  0  0 12  0  0  0]
 [ 2  1  0  0  0  0 13  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  2  0  0  0  0  0  5]]
```

Com pipeline, n\_components PCA = 9

k = 1: Acurácia = 0.77236

k = 1: Precisão = 0.77020

k = 1: Sensibilidade = 0.78052

Matriz de Confusão:

```
[[ 7  0  1  2  2  3  0  1  1]
 [ 0  6  3  0  2  1  0  0  0]
 [ 0  0  7  0  0  0  1  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 1  0  0  0 11  0  2  0  0]
 [ 0  1  1  0  0 13  1  0  0]
 [ 1  1  0  0  1  0 13  1  0]
```



```
[ 0  0  0  0  0  0  0 16  0]
[ 0  0  0  0  0  1  0  0  7]]
```

Com pipeline, n\_components PCA = 11

k = 1: Acurácia = 0.77236

k = 1: Precisão = 0.77098

k = 1: Sensibilidade = 0.77548

Matriz de Confusão:

```
[[ 7  0  1  3  1  4  0  0  1]
 [ 0  7  3  1  0  1  0  0  0]
 [ 1  0  6  0  0  0  1  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  0  0 11  0  2  0  0]
 [ 0  1  1  1  0 12  1  0  0]
 [ 1  1  0  0  1  0 14  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  0  0  0  0  1  0  0  7]]
```

Com pipeline, n\_components PCA = 12

k = 1: Acurácia = 0.78862

k = 1: Precisão = 0.79503

k = 1: Sensibilidade = 0.79731

Matriz de Confusão:

```
[[ 7  0  1  4  1  3  0  0  1]
 [ 0  7  3  1  0  1  0  0  0]
 [ 0  0  7  0  0  0  1  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  0  0 12  0  1  0  0]
 [ 0  1  1  1  0 12  1  0  0]
 [ 1  1  1  0  0  0 14  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  0  0  0  0  1  0  0  7]]
```

```
-----
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
```

Sem pipeline

k = 3: Acurácia = 0.57724

k = 3: Precisão = 0.55163

k = 3: Sensibilidade = 0.56582

Matriz de Confusão:

```
[[ 5  1  1  1  2  4  1  0  2]
 [ 1  7  1  0  2  0  1  0  0]
 [ 1  1  6  0  0  0  0  0  0]
 [ 1  0  0 12  0  0  2  0  0]
 [ 1  0  1  0  9  1  1  1  0]
 [ 2  2  2  2  0  7  0  0  1]
 [ 2  0  3  0  1  0 11  0  0]
 [ 0  1  0  1  0  0  1 13  0]
 [ 3  3  0  0  0  1  0  0  1]]
```

Com pipeline, n\_components PCA = 5

k = 3: Acurácia = 0.60976

k = 3: Precisão = 0.65368

k = 3: Sensibilidade = 0.60828

Matriz de Confusão:

```
[[ 6  2  5  1  2  1  0  0  0]
 [ 1  8  2  0  0  1  0  0  0]
 [ 0  1  5  0  0  0  1  0  1]
 [ 0  2  0 13  0  0  0  0  0]
 [ 1  0  2  1  7  0  1  2  0]
 [ 2  0  3  0  2  8  1  0  0]
 [ 2  0  3  0  1  0 10  1  0]
 [ 0  0  1  0  1  0  0 14  0]
 [ 1  0  3  0  0  0  0  0  4]]
```

Com pipeline, n\_components PCA = 7

k = 3: Acurácia = 0.64228

k = 3: Precisão = 0.65794

k = 3: Sensibilidade = 0.62922

Matriz de Confusão:

```
[[ 9  2  1  4  1  0  0  0  0]
 [ 3  5  3  0  0  1  0  0  0]
 [ 0  0  5  0  0  0  2  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  2  1  7  0  1  2  0]
 [ 1  1  3  1  0 10  0  0  0]
 [ 2  1  2  0  1  0  9  2  0]
 [ 0  0  0  0  1  0  0 15  0]
 [ 2  2  0  0  0  0  0  0  4]]
```

Com pipeline, n\_components PCA = 9

k = 3: Acurácia = 0.69106

k = 3: Precisão = 0.69780

k = 3: Sensibilidade = 0.67818

Matriz de Confusão:

```
[[11  0  2  2  0  2  0  0  0]
 [ 1  5  3  1  2  0  0  0  0]
 [ 0  0  5  0  0  0  2  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  0  2  9  0  1  1  0]
 [ 1  2  2  2  1  8  0  0  0]
 [ 1  0  3  0  1  0 11  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  1  0  0  1  0  0  5]]
```

Com pipeline, n\_components PCA = 11

k = 3: Acurácia = 0.69106

k = 3: Precisão = 0.69752

k = 3: Sensibilidade = 0.68305

Matriz de Confusão:

```
[[ 7  1  2  4  1  2  0  0  0]
 [ 0  7  3  1  1  0  0  0  0]
 [ 1  0  5  0  0  0  1  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  0  3  8  0  1  1  0]
 [ 1  2  2  1  0 10  0  0  0]
 [ 1  1  1  0  1  0 12  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  1  0  0  1  0  0  5]]
```

Com pipeline, n\_components PCA = 12

k = 3: Acurácia = 0.69106

k = 3: Precisão = 0.70931

k = 3: Sensibilidade = 0.69271

Matriz de Confusão:

```
[[ 8  1  2  4  0  2  0  0  0]
 [ 0  8  2  1  1  0  0  0  0]
 [ 0  0  6  0  0  0  1  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  0  3  8  0  1  1  0]
 [ 1  3  1  2  0  9  0  0  0]
 [ 1  1  3  0  2  0 10  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 1  0  1  0  0  1  0  0  5]]
```

-----  
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

Sem pipeline

k = 5: Acurácia = 0.52846

k = 5: Precisão = 0.52095

k = 5: Sensibilidade = 0.54441

Matriz de Confusão:

```
[[ 2  2  2  1  2  5  1  0  2]
 [ 1  7  2  0  1  0  1  0  0]
 [ 1  1  5  0  0  0  0  1  0]
 [ 2  1  0  9  1  2  0  0  0]
 [ 1  0  0  1  9  1  1  1  0]
 [ 1  2  5  2  0  4  1  0  1]
 [ 2  0  2  0  1  0 12  0  0]
 [ 0  1  0  1  1  0  1 12  0]
 [ 0  2  0  0  0  1  0  0  5]]
```

Com pipeline, n\_components PCA = 5

k = 5: Acurácia = 0.51220

k = 5: Precisão = 0.54265

k = 5: Sensibilidade = 0.50354

Matriz de Confusão:

```
[[ 4  2  7  1  1  2  0  0  0]
 [ 1  5  3  1  0  2  0  0  0]
 [ 0  0  4  0  0  0  1  1  2]
 [ 1  4  0 10  0  0  0  0  0]
 [ 0  1  2  0  7  0  2  2  0]
 [ 1  1  3  0  2  8  1  0  0]
 [ 2  0  2  1  1  0 10  1  0]
 [ 0  0  0  0  4  0  0 12  0]
 [ 0  1  3  0  0  1  0  0  3]]
```

Com pipeline, n\_components PCA = 7

k = 5: Acurácia = 0.58537

k = 5: Precisão = 0.58317

k = 5: Sensibilidade = 0.56939

Matriz de Confusão:

```
[[ 6  0  4  3  2  1  0  1  0]
 [ 3  3  2  1  1  1  0  0  1]
 [ 0  0  4  0  1  0  2  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 0  1  2  2  7  0  1  1  0]
 [ 1  2  1  2  1  9  0  0  0]
 [ 2  0  2  0  1  0 10  2  0]
 [ 0  0  0  0  1  0  0 15  0]
 [ 0  2  2  0  0  0  0  0  4]]
```

Com pipeline, n\_components PCA = 9

k = 5: Acurácia = 0.64228

k = 5: Precisão = 0.67594

k = 5: Sensibilidade = 0.63833

Matriz de Confusão:

```
[[ 4  0  6  3  2  2  0  0  0]
 [ 0  8  2  1  1  0  0  0  0]
 [ 0  0  5  0  0  0  2  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 0  1  0  3  8  0  1  1  0]
 [ 1  0  2  2  1 10  0  0  0]
 [ 1  0  4  0  1  0 10  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  1  2  0  0  1  0  0  4]]
```

Com pipeline, n\_components PCA = 11

k = 5: Acurácia = 0.63415

k = 5: Precisão = 0.67152

k = 5: Sensibilidade = 0.61899

Matriz de Confusão:

```
[[ 6  1  5  3  1  1  0  0  0]
 [ 0  6  2  2  2  0  0  0  0]
 [ 0  1  5  0  0  0  1  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 0  1  0  3  8  0  1  1  0]
 [ 1  1  2  2  0 10  0  0  0]
 [ 1  0  4  0  1  0 10  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  1  3  0  0  1  0  0  3]]
```

Com pipeline, n\_components PCA = 12

k = 5: Acurácia = 0.64228

k = 5: Precisão = 0.69308

k = 5: Sensibilidade = 0.63520

Matriz de Confusão:

```
[[ 5  1  4  4  1  2  0  0  0]
 [ 0  7  2  1  2  0  0  0  0]
 [ 0  1  5  0  0  0  1  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 0  1  0  2  8  0  1  2  0]
 [ 0  2  1  3  0  9  1  0  0]
 [ 0  0  5  0  1  0 11  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  1  2  0  0  1  0  0  4]]
```

-----

Sem pipeline

k = 7: Acurácia = 0.50407

k = 7: Precisão = 0.50842

k = 7: Sensibilidade = 0.51184

Matriz de Confusão:

```
[[ 4  1  1  2  1  6  1  0  1]
 [ 0  7  3  0  1  0  1  0  0]
 [ 3  1  4  0  0  0  0  0  0]
 [ 1  1  0 11  1  0  1  0  0]
 [ 1  0  0  1  9  1  1  1  0]
 [ 0  4  6  1  0  3  1  0  1]
 [ 2  1  2  0  4  0  7  1  0]
 [ 0  1  0  1  1  0  0 13  0]
 [ 2  1  0  0  0  1  0  0  4]]
```

[illegible]

Com pipeline, n\_components PCA = 5

k = 7: Acurácia = 0.47154

k = 7: Precisão = 0.48318

k = 7: Sensibilidade = 0.45051

Matriz de Confusão:

```
[[ 4  2  6  1  1  3  0  0  0]
 [ 2  4  3  2  0  1  0  0  0]
 [ 0  1  3  0  0  0  2  0  2]
 [ 2  2  0 11  0  0  0  0  0]
 [ 1  1  2  0  4  0  3  3  0]
 [ 1  1  3  1  1  8  1  0  0]
 [ 2  1  2  1  1  0  9  1  0]
 [ 0  0  0  0  3  0  0 13  0]
 [ 1  2  3  0  0  0  0  0  2]]
```

Com pipeline, n\_components PCA = 7

k = 7: Acurácia = 0.53659

k = 7: Precisão = 0.56425

k = 7: Sensibilidade = 0.52453

Matriz de Confusão:

```
[[ 4  1  6  1  4  0  1  0  0]
 [ 1  8  2  0  1  0  0  0  0]
 [ 0  0  4  0  1  0  2  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 1  1  2  2  4  0  2  2  0]
 [ 2  2  1  1  1  9  0  0  0]
 [ 2  2  2  0  1  0  8  2  0]
 [ 0  1  0  0  1  0  2 12  0]
 [ 0  2  3  1  0  0  0  0  2]]
```

Com pipeline, n\_components PCA = 9

k = 7: Acurácia = 0.57724

k = 7: Precisão = 0.59613

k = 7: Sensibilidade = 0.56186

Matriz de Confusão:

```
[[ 5  2  4  1  3  2  0  0  0]
 [ 0  6  2  2  2  0  0  0  0]
 [ 0  0  5  0  0  0  2  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  0  1  1  6  0  3  2  0]
 [ 0  1  2  2  1  9  1  0  0]
 [ 1  0  4  0  1  0 10  1  0]
 [ 1  0  0  0  0  0  1 14  0]
 [ 0  1  3  1  0  1  0  0  2]]
```

Com pipeline, n\_components PCA = 11

k = 7: Acurácia = 0.59350

k = 7: Precisão = 0.61338

k = 7: Sensibilidade = 0.57956

Matriz de Confusão:

```
[[ 6  0  4  3  1  3  0  0  0]
 [ 0  5  3  2  2  0  0  0  0]
 [ 0  1  5  0  0  0  1  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  1  0  2  6  0  2  2  0]
 [ 1  2  1  2  1  9  0  0  0]
 [ 1  0  3  0  0  0 11  2  0]
 [ 0  0  0  0  0  0  2 14  0]
 [ 0  2  3  0  0  0  0  0  3]]
```

Com pipeline, n\_components PCA = 12

k = 7: Acurácia = 0.56911

k = 7: Precisão = 0.60042

k = 7: Sensibilidade = 0.55996

Matriz de Confusão:

```
[[ 4  1  4  3  2  2  0  0  1]
 [ 0  5  3  2  2  0  0  0  0]
 [ 0  1  5  0  0  0  1  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  1  0  2  6  0  2  2  0]
 [ 0  2  1  3  2  8  0  0  0]
 [ 0  0  4  1  1  0 10  1  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  2  3  0  0  0  0  0  3]]
```

-----

Sem pipeline

k = 9: Acurácia = 0.43902

k = 9: Precisão = 0.43549

k = 9: Sensibilidade = 0.44083

Matriz de Confusão:

```
[[ 3  1  3  1  1  5  0  1  2]
 [ 0  7  1  0  2  0  2  0  0]
 [ 3  0  3  0  1  0  0  1  0]
 [ 2  2  0  8  1  1  1  0  0]
 [ 1  2  0  1  8  0  1  1  0]
 [ 0  3  5  2  0  3  1  1  1]
 [ 2  1  2  0  5  0  6  1  0]
 [ 0  0  1  1  1  0  0 13  0]
 [ 2  1  0  0  0  2  0  0  3]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(



Com pipeline, n\_components PCA = 5

k = 9: Acurácia = 0.49593

k = 9: Precisão = 0.51857

k = 9: Sensibilidade = 0.47968

Matriz de Confusão:

```
[[ 3  3  7  1  1  2  0  0  0]
 [ 1  4  4  2  0  1  0  0  0]
 [ 0  3  3  0  0  0  1  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  0  3  1  4  0  2  3  0]
 [ 2  1  2  2  2  6  1  0  0]
 [ 1  1  2  0  1  0 10  2  0]
 [ 0  0  0  0  2  0  0 14  0]
 [ 1  0  3  0  0  1  0  0  3]]
```

Com pipeline, n\_components PCA = 7

k = 9: Acurácia = 0.54472

k = 9: Precisão = 0.56768

k = 9: Sensibilidade = 0.52516

Matriz de Confusão:

```
[[ 7  1  3  1  2  2  0  1  0]
 [ 2  6  4  0  0  0  0  0  0]
 [ 0  1  3  0  1  0  2  0  1]
 [ 0  0  1 14  0  0  0  0  0]
 [ 1  1  2  1  4  0  3  2  0]
 [ 1  2  1  1  1 10  0  0  0]
 [ 2  1  2  0  2  0  8  2  0]
 [ 0  1  0  0  0  0  3 12  0]
 [ 0  2  3  0  0  0  0  0  3]]
```

Com pipeline, n\_components PCA = 9

k = 9: Acurácia = 0.58537

k = 9: Precisão = 0.59536

k = 9: Sensibilidade = 0.57985

Matriz de Confusão:

```
[[ 6  0  3  3  2  3  0  0  0]
 [ 0  5  3  2  2  0  0  0  0]
 [ 0  0  5  0  0  0  2  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 1  1  0  1  5  0  3  3  0]
 [ 0  2  1  1  2 10  0  0  0]
 [ 1  1  3  0  1  0  9  2  0]
 [ 1  0  0  0  0  0  2 13  0]
 [ 0  1  2  0  0  1  0  0  4]]
```

Com pipeline, n\_components PCA = 11

k = 9: Acurácia = 0.56911

k = 9: Precisão = 0.57055

k = 9: Sensibilidade = 0.55301

Matriz de Confusão:

```
[[ 4  1  4  3  2  2  0  0  1]
 [ 0  5  4  1  2  0  0  0  0]
 [ 0  0  4  0  1  0  2  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 0  1  0  1  6  0  3  3  0]
 [ 0  2  1  2  2  9  0  0  0]
 [ 2  0  2  0  0  1 10  2  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  0  3  0  0  2  0  0  3]]
```

Com pipeline, n\_components PCA = 12

k = 9: Acurácia = 0.56098

k = 9: Precisão = 0.57756

k = 9: Sensibilidade = 0.54507

Matriz de Confusão:

```
[[ 4  1  4  4  1  2  0  1  0]
 [ 0  5  4  1  2  0  0  0  0]
 [ 0  0  4  0  1  0  2  0  1]
 [ 0  1  0 14  0  0  0  0  0]
 [ 1  1  0  1  5  0  3  3  0]
 [ 0  2  1  2  2  9  0  0  0]
 [ 1  0  3  0  1  1 10  1  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  0  3  0  0  2  0  0  3]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

Teste s/ pipeline, c/ oversampling

```
In [ ]: print("Testando sem pipeline de redução")

# Fazer previsões com o modelo k-NN
k = 1 # Escolha o valor de k desejado
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(x_train, y_train)
predictions = knn_classifier.predict(xd_test)
print(predictions)

predicted_groups = label_encoder.inverse_transform(predictions)
print(predicted_groups)
```

Testando sem pipeline de redução

```
[7 2 2 8 8 0 2 4 2 6 5 6 2 3 4 5 6 7 8 2]
```

```
['Toy Group' 'Hound Group' 'Hound Group' 'Working Group' 'Working Group'
 'Foundation Stock Service' 'Hound Group' 'Non-Sporting Group'
 'Hound Group' 'Terrier Group' 'Sporting Group' 'Terrier Group'
 'Hound Group' 'Miscellaneous Class' 'Non-Sporting Group' 'Sporting Group'
 'Terrier Group' 'Toy Group' 'Working Group' 'Hound Group']
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names

s

warnings.warn(

Teste c/ pipeline, c/ oversampling

```
In [ ]: print("Testando com pipeline de redução")

# Definir o pipeline
npca = 12
k = 1

# Ajustar o scaler nos dados de treinamento
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)

# Reduzir a dimensionalidade nos dados de treinamento
pca = PCA(n_components=npca)
x_train_reduced = pca.fit_transform(x_train_scaled)

# Criar o classificador KNN
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Treinar o classificador com os dados reduzidos
knn_classifier.fit(x_train_reduced, y_train)

# Aplicar o scaler e a redução de dimensionalidade nos dados de teste
x_test_scaled = scaler.transform(xd_test)
x_test_reduced = pca.transform(x_test_scaled)

# Fazer previsões com o classificador
y_pred = knn_classifier.predict(x_test_reduced)

# Imprimir as previsões
print(y_pred)

# Converter as previsões de volta para os rótulos originais
predicted_groups = label_encoder.inverse_transform(y_pred)
print(predicted_groups)

Testando com pipeline de redução
[7 2 6 8 8 0 2 4 2 6 5 1 2 3 4 5 6 7 8 5]
['Toy Group' 'Hound Group' 'Terrier Group' 'Working Group' 'Working Group'
 'Foundation Stock Service' 'Hound Group' 'Non-Sporting Group'
 'Hound Group' 'Terrier Group' 'Sporting Group' 'Herding Group'
 'Hound Group' 'Miscellaneous Class' 'Non-Sporting Group' 'Sporting Group'
 'Terrier Group' 'Toy Group' 'Working Group' 'Sporting Group']
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does
not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

## Método Random Forest

**Sobre o método random forest:** é utilizado para classificação de imagens, reconhecimento de caracteres manuscritos, processamento de fala e análise de dados biomédicos, entre outros. Este algoritmo utiliza os exemplos de treinamento  $k$  mais próximos ao exemplo de teste para realizar previsões.

- Atributo alvo: group
- n\_estimators
- max\_depth
- criterion
- E para cada valor de , foi calculada a:
  - Acurácia,
  - Precisão,
  - e Sensibilidade do modelo

Acurácia: mede, de maneira geral, a frequência que o classificador está correto.

Precisão: mede a incidência do modelo prever o grupo de um determinado cão, assim definindo o quão correta foi a precisão

Sensibilidade: mede a capacidade do modelo identificar corretamente os casos positivos.

**n\_estimators** : o número de árvores na floresta. Quanto maior o número de estimadores, mais complexo será o modelo e maior será o tempo de treinamento. Um número maior de estimadores pode melhorar o desempenho do modelo, mas também aumentar o risco de overfitting.

**max\_depth** : a profundidade máxima de cada árvore na floresta. Controla a complexidade do modelo e a capacidade de generalização. Um valor mais alto pode levar a um modelo mais complexo, com maior tendência a overfitting.

**criterion** : a medida de qualidade utilizada para avaliar a divisão dos nós durante a construção das árvores. O valor padrão é 'gini', que utiliza o índice de Gini como critério. Outra opção é 'entropy', que utiliza a entropia como critério. Esses critérios estão relacionados à pureza das divisões nos nós da árvore.

## Treinamento s/ Oversamplig

```
In [ ]: #! Antes de executar essas células, re-execute o bloco de código da questão 3 para

# Definindo atributo de entrada e saída
x = doggies.drop('group', axis=1) # Remover a coluna original 'group'
x = RobustScaler().fit_transform(x)

# Criar uma instância do LabelEncoder
label_encoder = LabelEncoder()

# Aplicar o Label encoding à coluna 'group'
y = label_encoder.fit_transform(doggies['group'])

# Separando dados de treinamento (20% para testes e 80% para treinamento)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.2, random_st
```

```
In [ ]: # Definir os parâmetros do Random Forest
n_estimators = [25, 50, 100, 125, 150]
npca_list = [5, 7, 9, 11, 12]

# Sem o pipeline
for n in n_estimators:
    rf_classifier = RandomForestClassifier(n_estimators=n)
    rf_classifier.fit(x_train, y_train)
    y_pred = rf_classifier.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    confusionm = confusion_matrix(y_test, y_pred)

    print(f"Sem pipeline, n_estimators = {n}")
    print(f"Acurácia = {accuracy:.5f}")
    print(f'Matriz de Confusão: \n {confusionm}\n')

    for pca in npca_list:
        # Com o pipeline
        pipeline = Pipeline([
            ('scaling', StandardScaler()),
            ('dimensionality_reduction', PCA(n_components=pca)),
            ('classification', RandomForestClassifier(n_estimators=n))
        ])
        pipeline.fit(x_train, y_train)
        x_test_reduced = pipeline['dimensionality_reduction'].transform(x_test)
        y_pred = pipeline.predict(x_test)
        accuracy = accuracy_score(y_test, y_pred)
        confusionm = confusion_matrix(y_test, y_pred)

        print(f"Com pipeline, n_estimators = {n}, PCA = {pca}")
        print(f"Acurácia = {accuracy:.5f}")
        print(f'Matriz de Confusão: \n {confusionm}\n')

    print("-"*100)
```

Sem pipeline, n\_estimators = 25

Acurácia = 0.41071

Matriz de Confusão:

```
[[9 1 0 0 0 1 1 0 1]
 [4 0 0 0 0 1 0 0 1]
 [2 1 2 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [3 0 0 0 0 0 1 0 0]
 [3 2 0 0 0 1 1 0 0]
 [2 0 1 0 2 0 4 0 0]
 [0 0 0 0 0 0 2 2 0]
 [1 0 0 0 0 1 0 0 5]]
```

Com pipeline, n\_estimators = 25, PCA = 5

Acurácia = 0.32143

Matriz de Confusão:

```
[[4 0 2 1 0 2 2 0 2]
 [0 2 2 0 0 1 0 0 1]
 [3 1 0 0 0 0 1 0 0]
 [0 0 0 0 1 0 0 0 0]
 [0 2 1 0 0 0 1 0 0]
 [2 1 0 0 0 4 0 0 0]
 [2 0 2 0 1 0 4 0 0]
 [0 0 0 0 1 0 2 1 0]
 [2 2 0 0 0 0 0 0 3]]
```

Com pipeline, n\_estimators = 25, PCA = 7

Acurácia = 0.42857

Matriz de Confusão:

```
[[6 1 3 0 0 1 2 0 0]
 [1 2 1 0 1 0 0 0 1]
 [1 0 3 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0]
 [1 2 0 0 0 0 1 0 0]
 [3 0 0 0 1 3 0 0 0]
 [1 2 0 0 0 1 4 1 0]
 [0 0 0 0 0 0 2 2 0]
 [1 1 0 0 0 1 0 0 4]]
```

Com pipeline, n\_estimators = 25, PCA = 9

Acurácia = 0.37500

Matriz de Confusão:

```
[[6 0 2 2 0 2 1 0 0]
 [2 2 0 0 2 0 0 0 0]
 [4 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0]
 [2 1 0 0 0 0 1 0 0]
 [2 2 1 0 0 1 1 0 0]
 [2 0 1 0 0 0 5 1 0]
 [0 0 0 0 0 0 2 2 0]
 [1 1 0 0 0 0 0 0 5]]
```

Com pipeline, n\_estimators = 25, PCA = 11

Acurácia = 0.37500

Matriz de Confusão:

```
[[7 3 0 1 0 2 0 0 0]
 [2 2 1 0 0 1 0 0 0]
 [2 0 3 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0]]
```

```
[2 0 1 0 0 0 1 0 0]
[2 1 1 1 0 1 1 0 0]
[1 0 1 0 2 1 3 1 0]
[1 0 0 0 0 0 1 2 0]
[0 1 3 0 0 0 0 0 3]]
```

Com pipeline, n\_estimators = 25, PCA = 12

Acurácia = 0.39286

Matriz de Confusão:

```
[[8 2 0 1 0 1 1 0 0]
 [2 1 1 0 1 1 0 0 0]
 [0 0 3 0 0 0 2 0 0]
 [0 0 0 0 0 0 0 1 0]
 [1 1 0 0 0 1 1 0 0]
 [1 3 0 0 0 2 1 0 0]
 [2 0 1 0 1 0 5 0 0]
 [0 0 0 1 0 0 3 0 0]
 [2 1 0 0 0 1 0 0 3]]
```

-----  
Sem pipeline, n\_estimators = 50

Acurácia = 0.51786

Matriz de Confusão:

```
[[9 1 0 0 0 1 1 0 1]
 [2 1 2 0 0 0 1 0 0]
 [1 0 3 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0]
 [1 1 1 0 0 0 1 0 0]
 [3 1 0 0 0 3 0 0 0]
 [2 0 0 0 1 0 6 0 0]
 [0 0 0 0 0 0 2 2 0]
 [0 1 1 0 0 0 0 0 5]]
```

Com pipeline, n\_estimators = 50, PCA = 5

Acurácia = 0.39286

Matriz de Confusão:

```
[[7 0 1 0 0 1 1 0 3]
 [0 1 2 0 0 1 1 0 1]
 [3 1 0 0 0 1 0 0 0]
 [0 0 0 0 1 0 0 0 0]
 [0 1 1 0 0 1 1 0 0]
 [2 1 0 0 1 3 0 0 0]
 [2 0 0 0 0 1 6 0 0]
 [0 0 0 0 0 0 2 2 0]
 [2 1 0 0 0 1 0 0 3]]
```

Com pipeline, n\_estimators = 50, PCA = 7

Acurácia = 0.39286

Matriz de Confusão:

```
[[7 2 3 0 0 0 1 0 0]
 [2 0 1 0 0 1 1 0 1]
 [3 1 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [0 2 0 0 0 0 1 0 1]
 [3 0 0 0 0 3 1 0 0]
 [1 0 0 0 0 2 6 0 0]
 [0 0 0 0 0 0 2 2 0]
 [2 1 0 0 0 0 0 0 4]]
```

Com pipeline, n\_estimators = 50, PCA = 9

Acurácia = 0.51786

Matriz de Confusão:

```
[[9 2 1 0 0 1 0 0 0]
 [2 3 0 0 0 0 1 0 0]
 [3 0 2 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [2 1 0 0 0 0 1 0 0]
 [2 0 1 0 0 2 1 0 1]
 [1 0 1 0 1 0 5 1 0]
 [0 0 0 0 0 0 1 3 0]
 [1 1 0 0 0 0 0 0 5]]
```

Com pipeline, n\_estimators = 50, PCA = 11

Acurácia = 0.42857

Matriz de Confusão:

```
[[8 4 0 1 0 0 0 0 0]
 [2 1 2 0 0 0 0 1 0]
 [0 0 4 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0]
 [2 1 0 0 0 0 1 0 0]
 [3 2 0 0 0 1 1 0 0]
 [2 0 1 0 1 0 5 0 0]
 [0 0 0 0 0 0 2 2 0]
 [2 0 2 0 0 0 0 0 3]]
```

Com pipeline, n\_estimators = 50, PCA = 12

Acurácia = 0.41071

Matriz de Confusão:

```
[[8 2 0 0 0 3 0 0 0]
 [2 0 2 0 0 1 0 1 0]
 [1 0 3 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0]
 [2 0 0 0 0 1 1 0 0]
 [1 3 1 0 1 1 0 0 0]
 [2 0 0 0 1 0 6 0 0]
 [1 0 0 0 0 0 1 2 0]
 [1 1 1 0 0 1 0 0 3]]
```

-----  
Sem pipeline, n\_estimators = 100

Acurácia = 0.44643

Matriz de Confusão:

```
[[8 1 1 0 0 0 1 0 2]
 [2 0 1 0 0 1 1 0 1]
 [1 1 2 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0]
 [3 0 0 0 0 0 1 0 0]
 [3 1 0 0 0 3 0 0 0]
 [1 1 0 0 1 1 5 0 0]
 [0 0 0 0 0 0 2 2 0]
 [0 0 1 0 0 1 0 0 5]]
```

Com pipeline, n\_estimators = 100, PCA = 5

Acurácia = 0.33929

Matriz de Confusão:

```
[[6 0 3 0 0 2 1 0 1]
 [1 1 1 0 0 1 1 0 1]
 [3 2 0 0 0 0 0 0 0]]
```



```
[0 0 0 0 0 0 0 1 0]
[0 2 1 0 0 0 1 0 0]
[2 1 0 0 1 3 0 0 0]
[1 1 0 0 1 1 4 1 0]
[0 0 0 0 1 0 2 1 0]
[1 1 0 0 0 1 0 0 4]]
```

Com pipeline, n\_estimators = 100, PCA = 7

Acurácia = 0.46429

Matriz de Confusão:

```
[[7 1 1 0 0 2 1 0 1]
 [0 3 1 0 0 0 1 0 1]
 [4 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [1 2 0 0 0 0 1 0 0]
 [2 0 0 0 0 4 1 0 0]
 [1 0 0 0 1 2 5 0 0]
 [0 0 0 0 0 0 2 2 0]
 [1 1 0 0 0 0 0 0 5]]
```

Com pipeline, n\_estimators = 100, PCA = 9

Acurácia = 0.48214

Matriz de Confusão:

```
[[9 1 1 0 0 2 0 0 0]
 [2 2 1 0 0 0 0 1 0]
 [4 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0]
 [2 1 0 0 0 0 1 0 0]
 [1 1 1 0 0 3 1 0 0]
 [2 0 0 0 1 0 6 0 0]
 [0 0 0 0 0 0 1 3 0]
 [1 0 1 0 0 1 0 0 4]]
```

Com pipeline, n\_estimators = 100, PCA = 11

Acurácia = 0.48214

Matriz de Confusão:

```
[[10 1 0 1 0 1 0 0 0]
 [ 2 0 2 0 0 1 1 0 0]
 [ 2 0 2 0 0 0 1 0 0]
 [ 0 0 0 0 0 0 0 1 0]
 [ 2 1 0 0 0 0 1 0 0]
 [ 2 1 0 0 0 3 1 0 0]
 [ 2 0 0 0 1 0 6 0 0]
 [ 0 0 0 0 1 0 1 2 0]
 [ 0 1 1 0 0 1 0 0 4]]
```

Com pipeline, n\_estimators = 100, PCA = 12

Acurácia = 0.51786

Matriz de Confusão:

```
[[11 2 0 0 0 0 0 0 0]
 [ 2 1 2 0 0 0 1 0 0]
 [ 0 0 3 0 0 0 2 0 0]
 [ 0 0 0 0 0 0 0 1 0]
 [ 2 1 0 0 0 0 1 0 0]
 [ 3 1 0 0 0 2 1 0 0]
 [ 2 1 0 0 0 0 5 1 0]
 [ 1 0 0 0 0 0 1 2 0]
 [ 1 0 1 0 0 0 0 0 5]]
```

-----  
Sem pipeline, n\_estimators = 125

Acurácia = 0.48214

Matriz de Confusão:

```
[[9 1 1 0 0 0 1 0 1]
 [3 0 0 0 0 1 2 0 0]
 [1 1 2 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [2 1 0 0 0 0 1 0 0]
 [1 1 1 0 0 4 0 0 0]
 [2 0 0 0 1 1 5 0 0]
 [0 0 0 0 0 0 2 2 0]
 [0 0 1 0 0 1 0 0 5]]
```

Com pipeline, n\_estimators = 125, PCA = 5

Acurácia = 0.33929

Matriz de Confusão:

```
[[4 0 1 0 0 5 1 0 2]
 [0 3 2 0 0 0 0 0 1]
 [3 2 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [0 1 1 0 0 1 1 0 0]
 [2 1 0 0 0 3 1 0 0]
 [1 1 0 0 1 1 4 1 0]
 [0 0 0 0 1 0 2 1 0]
 [1 1 0 0 0 1 0 0 4]]
```

Com pipeline, n\_estimators = 125, PCA = 7

Acurácia = 0.42857

Matriz de Confusão:

```
[[8 0 3 0 0 1 1 0 0]
 [1 2 1 0 0 0 1 0 1]
 [4 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [1 2 0 0 0 0 1 0 0]
 [3 0 0 0 0 2 1 0 1]
 [2 0 1 0 0 1 5 0 0]
 [0 0 0 0 0 0 2 2 0]
 [1 1 0 0 0 0 0 0 5]]
```

Com pipeline, n\_estimators = 125, PCA = 9

Acurácia = 0.39286

Matriz de Confusão:

```
[[7 1 1 0 1 2 0 0 1]
 [2 1 1 0 0 0 2 0 0]
 [3 2 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [2 1 0 0 0 0 1 0 0]
 [3 0 1 0 0 2 1 0 0]
 [1 1 1 0 1 0 5 0 0]
 [0 0 0 0 0 0 1 3 0]
 [0 1 2 0 0 0 0 0 4]]
```

Com pipeline, n\_estimators = 125, PCA = 11

Acurácia = 0.48214

Matriz de Confusão:

```
[[10 2 0 0 0 1 0 0 0]
 [ 2 0 1 0 1 1 0 1 0]
 [ 1 0 3 0 0 0 1 0 0]
```

```
[ 0 0 0 0 0 0 0 1 0]
[ 2 0 1 0 0 0 1 0 0]
[ 2 1 0 0 0 3 1 0 0]
[ 2 1 0 0 1 0 5 0 0]
[ 0 0 0 0 0 0 1 3 0]
[ 1 1 1 0 0 1 0 0 3]]
```

Com pipeline, n\_estimators = 125, PCA = 12

Acurácia = 0.48214

Matriz de Confusão:

```
[[9 2 0 1 0 1 0 0 0]
 [2 1 1 0 0 1 1 0 0]
 [1 0 3 0 0 0 1 0 0]
 [0 0 0 0 1 0 0 0 0]
 [2 1 0 0 0 0 1 0 0]
 [2 1 1 0 0 2 1 0 0]
 [2 0 1 0 1 1 4 0 0]
 [0 0 0 0 0 0 1 3 0]
 [0 0 2 0 0 0 0 0 5]]
```

-----

Sem pipeline, n\_estimators = 150

Acurácia = 0.51786

Matriz de Confusão:

```
[[9 1 1 0 0 0 1 0 1]
 [2 2 0 0 0 1 1 0 0]
 [2 1 2 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [1 0 1 0 0 0 2 0 0]
 [3 1 0 0 0 3 0 0 0]
 [3 0 0 0 1 0 5 0 0]
 [0 0 0 0 0 0 2 2 0]
 [0 0 0 0 0 1 0 0 6]]
```

Com pipeline, n\_estimators = 150, PCA = 5

Acurácia = 0.35714

Matriz de Confusão:

```
[[6 0 2 0 0 3 1 0 1]
 [1 1 2 0 0 1 0 0 1]
 [3 2 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [0 2 1 0 0 0 1 0 0]
 [2 1 0 0 0 3 1 0 0]
 [2 1 0 0 1 0 5 0 0]
 [0 0 0 0 1 0 2 1 0]
 [2 1 0 0 0 0 0 0 4]]
```

Com pipeline, n\_estimators = 150, PCA = 7

Acurácia = 0.39286

Matriz de Confusão:

```
[[5 1 1 0 0 3 2 0 1]
 [1 0 1 0 0 2 1 0 1]
 [4 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [2 1 0 0 0 0 1 0 0]
 [2 0 0 0 1 4 0 0 0]
 [2 0 0 0 0 1 6 0 0]
 [0 0 0 0 0 0 2 2 0]
 [1 1 0 0 0 0 0 0 5]]
```

Com pipeline, n\_estimators = 150, PCA = 9

Acurácia = 0.42857

Matriz de Confusão:

```
[[9 0 1 0 1 2 0 0 0]
 [3 0 1 0 0 1 0 1 0]
 [4 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0]
 [2 1 0 0 0 0 1 0 0]
 [1 2 1 0 0 2 1 0 0]
 [1 1 0 0 0 0 6 1 0]
 [0 0 0 0 0 0 1 3 0]
 [0 1 2 0 0 0 0 0 4]]
```

Com pipeline, n\_estimators = 150, PCA = 11

Acurácia = 0.46429

Matriz de Confusão:

```
[[10 3 0 0 0 0 0 0 0]
 [ 2 0 2 0 0 1 0 1 0]
 [ 0 0 4 0 0 0 1 0 0]
 [ 0 0 0 0 0 0 0 1 0]
 [ 2 0 0 0 0 1 1 0 0]
 [ 2 1 0 0 0 3 1 0 0]
 [ 2 0 1 0 0 1 3 2 0]
 [ 1 0 0 0 0 0 1 2 0]
 [ 2 0 1 0 0 0 0 0 4]]
```

Com pipeline, n\_estimators = 150, PCA = 12

Acurácia = 0.50000

Matriz de Confusão:

```
[[11 2 0 0 0 0 0 0 0]
 [ 2 1 1 0 0 1 0 1 0]
 [ 1 0 3 0 0 0 1 0 0]
 [ 0 0 0 0 0 0 0 1 0]
 [ 2 0 0 0 0 1 1 0 0]
 [ 4 0 0 0 0 2 1 0 0]
 [ 2 0 0 0 1 0 6 0 0]
 [ 0 0 0 0 0 0 1 3 0]
 [ 2 0 2 0 0 1 0 0 2]]
```

-----

Teste s/ pipeline, s/ oversampling

```
In [ ]: # Fazer previsões com o modelo Random Forest
        predictions = rf_classifier.predict(xd_test)
        print(predictions)

        predicted_groups = label_encoder.inverse_transform(predictions)
        print(predicted_groups)

[8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8]
['Working Group' 'Working Group' 'Working Group' 'Working Group'
 'Working Group' 'Working Group' 'Working Group' 'Working Group'
 'Working Group' 'Working Group' 'Working Group' 'Working Group'
 'Working Group' 'Working Group' 'Working Group' 'Working Group'
 'Working Group' 'Working Group' 'Working Group' 'Working Group']
```

Teste c/ pipeline, s/ oversampling

```
In [ ]: print("Testando com pipeline de redução no Random Forest")

# Definir o pipeline
npca = 12
n_estimators = 100

# Ajustar o scaler nos dados de treinamento
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)

# Reduzir a dimensionalidade nos dados de treinamento
pca = PCA(n_components=npca)
x_train_reduced = pca.fit_transform(x_train_scaled)

# Criar o classificador Random Forest
rf_classifier = RandomForestClassifier(n_estimators=n_estimators)

# Treinar o classificador com os dados reduzidos
rf_classifier.fit(x_train_reduced, y_train)

# Aplicar o scaler e a redução de dimensionalidade nos dados de teste
x_test_scaled = scaler.transform(xd_test)
x_test_reduced = pca.transform(x_test_scaled)

# Fazer previsões com o classificador
y_pred = rf_classifier.predict(x_test_reduced)

# Imprimir as previsões
print(y_pred)

# Converter as previsões de volta para os rótulos originais
predicted_groups = label_encoder.inverse_transform(y_pred)
print(predicted_groups)
```

Testando com pipeline de redução no Random Forest

[0 8 2 2 0 0 0 0 2 2 0 0 2 0 2 0 2 0 0 0]

['Foundation Stock Service' 'Working Group' 'Hound Group' 'Hound Group'  
'Foundation Stock Service' 'Foundation Stock Service'  
'Foundation Stock Service' 'Foundation Stock Service' 'Hound Group'  
'Hound Group' 'Foundation Stock Service' 'Foundation Stock Service'  
'Hound Group' 'Foundation Stock Service' 'Hound Group'  
'Foundation Stock Service' 'Hound Group' 'Foundation Stock Service'  
'Foundation Stock Service' 'Foundation Stock Service']

Treinamento c/ Oversampling

```
In [ ]: # Realizando oversampling devido à diferença entre a quantidade de cães por grupo c
dcp = doggies.copy()

# Separa as colunas de features e o alvo
x = dcp.drop('group', axis=1)
y = dcp['group']

# Aplica o SMOTE para gerar dados sintéticos
smote = SMOTE()
x_oversampled, y_oversampled = smote.fit_resample(x, y)

# Cria um novo DataFrame com as features e o alvo oversampled
dados_oversampled = pd.DataFrame(x_oversampled, columns=x.columns)
dados_oversampled['group'] = y_oversampled

# Criar uma instância do LabelEncoder
label_encoder = LabelEncoder()

# Aplicar o Label encoding à coluna 'group'
dados_oversampled['group'] = label_encoder.fit_transform(dados_oversampled['group'])

# Separando dados de treinamento e testes
x_train, x_test, y_train, y_test = train_test_split(dados_oversampled.drop('group',
```

```
In [ ]: # Definir os parâmetros do Random Forest
n_estimators = [25, 50, 100, 125, 150]
npca_list = [5, 7, 9, 11, 12]

print("COM OVERSAMPLING")
print("-"*100)

for n in n_estimators:
    # Sem o pipeline
    rf_classifier = RandomForestClassifier(n_estimators=n)
    rf_classifier.fit(x_train, y_train)
    y_pred = rf_classifier.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    confusionm = confusion_matrix(y_test, y_pred)

    print(f"Sem pipeline, n_estimators = {n}")
    print(f"Acurácia = {accuracy:.5f}")
    print(f'Matriz de Confusão: \n {confusionm}\n')

    # Com o pipeline
    for pca in npca_list:
        pipeline = Pipeline([
            ('scaling', StandardScaler()),
            ('dimensionality_reduction', PCA(n_components=pca)),
            ('classification', RandomForestClassifier(n_estimators=n))
        ])
        pipeline.fit(x_train, y_train)
        x_test_reduced = pipeline['dimensionality_reduction'].transform(x_test)
        y_pred = pipeline.predict(x_test)
        accuracy = accuracy_score(y_test, y_pred)
        confusionm = confusion_matrix(y_test, y_pred)

        print(f"Com pipeline, n_estimators = {n}, PCA = {pca}")
        print(f"Acurácia = {accuracy:.5f}")
        print(f'Matriz de Confusão: \n {confusionm}\n')

print("-"*100)
```

## COM OVERSAMPLING

Sem pipeline, n\_estimators = 25

Acurácia = 0.73984

Matriz de Confusão:

```
[[ 9  3  0  1  1  1  1  1  0]
 [ 0 11  0  0  0  0  1  0  0]
 [ 0  1  6  0  0  0  1  0  0]
 [ 1  0  0 11  2  0  0  1  0]
 [ 0  1  0  0 11  1  0  1  0]
 [ 1  1  1  1  1 11  0  0  0]
 [ 0  1  1  0  1  1 12  1  0]
 [ 0  0  0  0  0  0  2 14  0]
 [ 0  0  0  0  0  1  1  0  6]]
```

Com pipeline, n\_estimators = 25, PCA = 5

Acurácia = 0.62602

Matriz de Confusão:

```
[[ 7  1  1  0  0  2  3  0  3]
 [ 1  8  0  0  3  0  0  0  0]
 [ 1  0  6  0  0  0  0  0  1]
 [ 0  0  1 11  1  0  2  0  0]
 [ 2  0  0  0 10  0  1  1  0]
 [ 3  4  0  0  0  7  1  0  1]
 [ 1  1  1  1  1  0 11  0  1]
 [ 0  0  0  0  0  0  5 11  0]
 [ 0  0  0  1  0  1  0  0  6]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(



Com pipeline, n\_estimators = 25, PCA = 7

Acurácia = 0.70732

Matriz de Confusão:

```
[[ 6  1  2  1  2  2  1  0  2]
 [ 0 10  0  0  2  0  0  0  0]
 [ 1  1  5  0  0  0  0  0  1]
 [ 0  0  1 14  0  0  0  0  0]
 [ 2  0  0  0 10  0  0  1  1]
 [ 1  2  1  0  0 10  1  0  1]
 [ 3  0  1  0  1  0 11  1  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

Com pipeline, n\_estimators = 25, PCA = 9

Acurácia = 0.73171

Matriz de Confusão:

```
[[10  1  1  1  2  2  0  0  0]
 [ 0  9  1  0  1  0  1  0  0]
 [ 1  0  7  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  1  0  0]
 [ 2  0  0  0 10  0  2  0  0]
 [ 0  5  0  1  0  9  0  0  1]
 [ 0  0  3  0  3  0  9  2  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

Com pipeline, n\_estimators = 25, PCA = 11

Acurácia = 0.74797

Matriz de Confusão:

```
[[ 9  2  2  1  0  2  0  1  0]
 [ 0  9  0  0  1  1  1  0  0]
 [ 0  1  7  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 1  0  0  0 11  0  1  1  0]
 [ 1  4  0  0  0 10  0  0  1]
 [ 2  1  1  1  2  0 10  0  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 1  0  0  0  0  1  0  0  6]]
```

Com pipeline, n\_estimators = 25, PCA = 12

Acurácia = 0.78049

Matriz de Confusão:

```
[[ 9  2  3  2  0  0  0  1  0]
 [ 0 11  0  0  0  1  0  0  0]
 [ 0  0  8  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  0  0  0 10  0  1  1  2]
 [ 2  2  0  0  0  9  1  0  2]
 [ 0  0  3  1  1  0 12  0  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  1  0  0  0  0  0  0  7]]
```

-----  
Sem pipeline, n\_estimators = 50

Acurácia = 0.80488

Matriz de Confusão:

```
[[10  2  0  0  1  2  1  1  0]
 [ 0 10  0  0  1  0  1  0  0]
 [ 1  0  7  0  0  0  0  0  0]
 [ 0  2  0 13  0  0  0  0  0]
 [ 0  0  0  0 12  1  0  1  0]
 [ 0  2  0  0  2 12  0  0  0]
 [ 0  0  1  0  0  1 14  1  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

Com pipeline, n\_estimators = 50, PCA = 5

Acurácia = 0.65041

Matriz de Confusão:

```
[[ 5  2  4  1  0  3  0  1  1]
 [ 0  9  1  1  1  0  0  0  0]
 [ 1  0  6  0  0  0  0  0  1]
 [ 0  1  1 11  1  0  1  0  0]
 [ 3  0  0  0  9  0  1  1  0]
 [ 1  3  1  0  0  9  1  0  1]
 [ 1  1  1  0  0  0 13  0  1]
 [ 0  0  0  0  0  0  4 12  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
Com pipeline, n_estimators = 50, PCA = 7
Acurácia = 0.77236
Matriz de Confusão:
[[11  1  2  2  0  0  1  0  0]
 [ 0  9  0  0  2  1  0  0  0]
 [ 1  0  6  0  0  1  0  0  0]
 [ 0  0  1 14  0  0  0  0  0]
 [ 2  0  0  0 10  0  1  0  1]
 [ 1  2  1  1  0  9  1  0  1]
 [ 0  0  3  0  0  0 14  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
Com pipeline, n_estimators = 50, PCA = 9
Acurácia = 0.74797
Matriz de Confusão:
[[ 7  4  1  2  1  1  0  1  0]
 [ 0  9  0  0  1  1  1  0  0]
 [ 0  0  8  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  2  0  0 12  0  0  0  0]
 [ 1  3  0  1  0  9  0  0  2]
 [ 1  0  1  1  1  1 11  1  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

```
Com pipeline, n_estimators = 50, PCA = 11
Acurácia = 0.79675
Matriz de Confusão:
[[12  0  3  1  0  0  0  1  0]
 [ 0 10  0  0  1  1  0  0  0]
 [ 0  1  7  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  3  0  0 10  0  1  0  0]
 [ 1  3  1  1  0  9  1  0  0]
 [ 0  0  2  0  0  0 15  0  0]
 [ 0  0  0  0  0  0  2 14  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
```

Com pipeline, n\_estimators = 50, PCA = 12

Acurácia = 0.82114

Matriz de Confusão:

```
[[ 9  2  1  2  2  0  0  1  0]
 [ 0 11  0  0  1  0  0  0  0]
 [ 0  0  8  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  0  1  0 11  0  1  1  0]
 [ 1  1  1  1  0 11  1  0  0]
 [ 0  0  1  0  0  1 15  0  0]
 [ 0  0  0  0  0  0  2 14  0]
 [ 0  1  0  0  0  0  0  0  7]]
```

-----  
Sem pipeline, n\_estimators = 100

Acurácia = 0.79675

Matriz de Confusão:

```
[[11  2  1  0  1  0  1  1  0]
 [ 0 10  0  0  2  0  0  0  0]
 [ 1  0  7  0  0  0  0  0  0]
 [ 0  1  0 13  1  0  0  0  0]
 [ 0  1  0  0 11  1  0  1  0]
 [ 1  3  0  0  0 12  0  0  0]
 [ 0  0  1  0  1  1 14  0  0]
 [ 0  0  0  0  0  0  2 14  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

Com pipeline, n\_estimators = 100, PCA = 5

Acurácia = 0.68293

Matriz de Confusão:

```
[[ 6  1  4  1  1  1  2  0  1]
 [ 0 10  0  1  1  0  0  0  0]
 [ 1  0  5  1  0  0  0  0  1]
 [ 0  1  1 12  0  0  1  0  0]
 [ 2  0  0  0 10  0  1  1  0]
 [ 1  2  0  0  0 11  1  0  1]
 [ 1  0  2  0  1  0 12  0  1]
 [ 0  0  0  0  1  0  3 12  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

```
Com pipeline, n_estimators = 100, PCA = 7
```

```
Acurácia = 0.74797
```

```
Matriz de Confusão:
```

```
[[ 8  2  2  2  0  1  1  1  0]
 [ 0  9  0  0  2  1  0  0  0]
 [ 1  0  7  0  0  0  0  0  0]
 [ 0  0  0 14  1  0  0  0  0]
 [ 2  0  0  0 11  0  1  0  0]
 [ 2  2  0  0  0 10  1  0  1]
 [ 0  0  3  0  0  0 13  1  0]
 [ 0  0  0  0  0  0  2 14  0]
 [ 0  2  0  0  0  0  0  0  6]]
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
```

```
warnings.warn(
```

```
Com pipeline, n_estimators = 100, PCA = 9
```

```
Acurácia = 0.78049
```

```
Matriz de Confusão:
```

```
[[12  2  1  1  1  0  0  0  0]
 [ 0  9  0  0  1  1  1  0  0]
 [ 0  0  8  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  2  0  9  0  1  1  0]
 [ 0  5  0  0  0 10  0  0  1]
 [ 1  0  2  1  1  0 12  0  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
```

```
warnings.warn(
```

```
Com pipeline, n_estimators = 100, PCA = 11
```

```
Acurácia = 0.79675
```

```
Matriz de Confusão:
```

```
[[ 8  5  2  1  0  0  0  1  0]
 [ 0 10  0  0  0  1  1  0  0]
 [ 0  0  8  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  0  0  0 12  0  1  1  0]
 [ 1  2  1  0  0 11  1  0  0]
 [ 1  0  2  1  0  0 13  0  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
```

```
warnings.warn(
```

Com pipeline, n\_estimators = 100, PCA = 12

Acurácia = 0.83740

Matriz de Confusão:

```
[[10  2  3  1  0  0  0  1  0]
 [ 0 10  0  0  0  1  1  0  0]
 [ 0  0  8  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  0  0  0 13  0  1  0  0]
 [ 1  2  1  1  0 10  1  0  0]
 [ 0  0  1  0  0  1 15  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

-----  
Sem pipeline, n\_estimators = 125

Acurácia = 0.80488

Matriz de Confusão:

```
[[ 9  3  1  0  1  1  1  1  0]
 [ 0 10  0  0  1  0  1  0  0]
 [ 1  0  7  0  0  0  0  0  0]
 [ 0  0  0 13  2  0  0  0  0]
 [ 1  1  0  0 10  1  0  1  0]
 [ 1  1  0  0  1 13  0  0  0]
 [ 0  0  1  0  0  1 15  0  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

Com pipeline, n\_estimators = 125, PCA = 5

Acurácia = 0.67480

Matriz de Confusão:

```
[[ 6  0  3  2  0  1  2  1  2]
 [ 0 10  0  0  2  0  0  0  0]
 [ 1  0  5  1  0  0  0  0  1]
 [ 0  0  1 12  1  0  1  0  0]
 [ 2  0  0  0 10  0  1  1  0]
 [ 1  2  1  0  0  9  1  0  2]
 [ 1  0  2  1  0  0 12  0  1]
 [ 0  0  0  0  1  0  2 13  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

```
Com pipeline, n_estimators = 125, PCA = 7
```

```
Acurácia = 0.74797
```

```
Matriz de Confusão:
```

```
[[10  1  3  2  0  0  0  1  0]
 [ 0  9  0  0  2  1  0  0  0]
 [ 1  0  6  0  0  0  0  0  1]
 [ 0  0  1 14  0  0  0  0  0]
 [ 2  0  0  0 11  0  1  0  0]
 [ 2  3  0  0  0  7  1  0  3]
 [ 0  0  3  0  0  0 14  0  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
```

```
warnings.warn(
```

```
Com pipeline, n_estimators = 125, PCA = 9
```

```
Acurácia = 0.78862
```

```
Matriz de Confusão:
```

```
[[13  1  1  1  0  0  0  1  0]
 [ 0 10  0  0  1  0  1  0  0]
 [ 0  0  7  0  0  0  0  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  1  2  0  8  0  1  1  1]
 [ 0  4  0  0  0 11  0  0  1]
 [ 1  0  1  1  1  1 12  0  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
```

```
warnings.warn(
```

```
Com pipeline, n_estimators = 125, PCA = 11
```

```
Acurácia = 0.75610
```

```
Matriz de Confusão:
```

```
[[ 9  2  3  2  0  0  0  1  0]
 [ 0  9  0  0  2  1  0  0  0]
 [ 0  0  7  0  0  1  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  0  1  0 11  0  1  1  0]
 [ 1  1  1  1  0 10  1  0  1]
 [ 2  0  2  1  1  0 11  0  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
```

```
warnings.warn(
```

Com pipeline, n\_estimators = 125, PCA = 12

Acurácia = 0.77236

Matriz de Confusão:

```
[[ 8  3  3  2  0  0  0  1  0]
 [ 0  9  0  0  1  1  1  0  0]
 [ 0  0  8  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  0  0  0 11  0  1  1  1]
 [ 1  1  1  1  0 10  1  0  1]
 [ 1  1  1  0  1  0 13  0  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

-----  
Sem pipeline, n\_estimators = 150

Acurácia = 0.80488

Matriz de Confusão:

```
[[11  0  2  0  1  1  1  1  0]
 [ 0 11  0  0  0  0  1  0  0]
 [ 0  0  6  0  0  1  1  0  0]
 [ 0  0  0 13  2  0  0  0  0]
 [ 0  0  0  0 12  1  0  1  0]
 [ 1  3  0  1  0 11  0  0  0]
 [ 0  0  1  0  1  1 13  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

Com pipeline, n\_estimators = 150, PCA = 5

Acurácia = 0.69919

Matriz de Confusão:

```
[[ 7  2  3  0  0  1  1  1  2]
 [ 0 10  0  0  2  0  0  0  0]
 [ 1  0  6  0  0  0  0  0  1]
 [ 0  0  1 12  1  0  1  0  0]
 [ 2  0  0  0 10  0  1  1  0]
 [ 1  2  1  0  0 10  1  0  1]
 [ 0  1  2  0  0  0 13  0  1]
 [ 0  0  0  0  1  0  3 12  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(



Com pipeline, n\_estimators = 150, PCA = 7

Acurácia = 0.73171

Matriz de Confusão:

```
[[ 8  3  3  2  0  0  0  1  0]
 [ 0 10  0  0  2  0  0  0  0]
 [ 1  1  5  0  0  0  0  0  1]
 [ 0  0  0 14  1  0  0  0  0]
 [ 2  0  0  0 10  0  1  1  0]
 [ 1  3  0  0  0  8  1  0  3]
 [ 0  0  3  0  0  0 13  1  0]
 [ 0  0  0  0  0  0  0 16  0]
 [ 0  0  0  0  0  2  0  0  6]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

Com pipeline, n\_estimators = 150, PCA = 9

Acurácia = 0.79675

Matriz de Confusão:

```
[[10  2  3  1  0  0  0  1  0]
 [ 0 10  0  0  1  0  1  0  0]
 [ 0  0  8  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  0  0  0 12  0  1  1  0]
 [ 0  4  0  1  0 10  0  0  1]
 [ 0  0  3  1  0  0 13  0  0]
 [ 0  0  0  0  0  0  2 14  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but PCA was fitted without feature names

warnings.warn(

Com pipeline, n\_estimators = 150, PCA = 11

Acurácia = 0.78862

Matriz de Confusão:

```
[[10  2  3  1  0  0  0  1  0]
 [ 0  9  0  0  1  1  1  0  0]
 [ 0  0  8  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  0  0  0 12  0  1  1  0]
 [ 1  3  1  0  0  9  1  0  1]
 [ 1  0  2  1  0  0 13  0  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

Com pipeline, n\_estimators = 150, PCA = 12

Acurácia = 0.80488

Matriz de Confusão:

```
[[10  1  3  2  0  0  0  1  0]
 [ 0 11  0  0  0  0  1  0  0]
 [ 0  0  8  0  0  0  0  0  0]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  0  2  0 10  0  1  1  0]
 [ 1  1  1  1  0 10  1  0  1]
 [ 1  0  2  0  0  0 14  0  0]
 [ 0  0  0  0  0  0  1 15  0]
 [ 0  1  0  0  0  1  0  0  6]]
```

```
-----
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but PCA was fitted without feature names
  warnings.warn(
```

Teste s/ pipeline, c/ oversampling

```
In [ ]: # COM OVERSAMPLING E SEM PIPELINE
# Separando dados de treinamento e testes
x_train, x_test, y_train, y_test = train_test_split(dados_oversampled.drop('group',

# Treinar o modelo Random Forest sem pipeline
rf_classifier = RandomForestClassifier(n_estimators=50)
rf_classifier.fit(x_train, y_train)

# Fazer previsões com o modelo treinado
y_pred = rf_classifier.predict(xd_test)

# Imprimir as previsões
print(y_pred)

# Converter as previsões de volta para os rótulos originais
predicted_groups = label_encoder.inverse_transform(y_pred)
print(predicted_groups)

[7 2 5 8 8 0 5 4 2 6 0 1 2 3 4 5 6 7 8 1]
['Toy Group' 'Hound Group' 'Sporting Group' 'Working Group'
 'Working Group' 'Foundation Stock Service' 'Sporting Group'
 'Non-Sporting Group' 'Hound Group' 'Terrier Group'
 'Foundation Stock Service' 'Herding Group' 'Hound Group'
 'Miscellaneous Class' 'Non-Sporting Group' 'Sporting Group'
 'Terrier Group' 'Toy Group' 'Working Group' 'Herding Group']
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does
not have valid feature names, but RandomForestClassifier was fitted with feature na
mes
```

```
warnings.warn(
```

Teste c/ pipeline, c/ oversampling

```
In [ ]: # COM OVERSAMPLING E COM PIPELINE
# Separando dados de treinamento e testes
x_train, x_test, y_train, y_test = train_test_split(dados_oversampled.drop('group',

# Definir o pipeline
npca = 12
n_estimators = 100

# Criar o pipeline
pipeline = Pipeline([
    ('scaling', StandardScaler()),
    ('dimensionality_reduction', PCA(n_components=npca)),
    ('classification', RandomForestClassifier(n_estimators=n_estimators))
])

# Treinar o modelo com o pipeline
pipeline.fit(x_train, y_train)

# Fazer previsões com o modelo treinado
y_pred = pipeline.predict(xd_test)

# Imprimir as previsões
print(y_pred)

# Converter as previsões de volta para os rótulos originais
predicted_groups = label_encoder.inverse_transform(y_pred)
print(predicted_groups)
```

```
[7 2 1 8 8 0 2 4 2 6 5 1 2 3 4 5 6 7 8 1]
['Toy Group' 'Hound Group' 'Herding Group' 'Working Group' 'Working Group'
'Foundation Stock Service' 'Hound Group' 'Non-Sporting Group'
'Hound Group' 'Terrier Group' 'Sporting Group' 'Herding Group'
'Hound Group' 'Miscellaneous Class' 'Non-Sporting Group' 'Sporting Group'
'Terrier Group' 'Toy Group' 'Working Group' 'Herding Group']
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does
not have valid feature names, but StandardScaler was fitted with feature names
warnings.warn(
```

## Questão 5 (Não-Supervisionado)

Defina um pipeline de aplicação de pelo menos 2 métodos de aprendizado de máquina não supervisionado e da variação de ao menos 1 parâmetro deles, execute-o e justifique suas escolhas (sobre os métodos, parâmetros e valores);

## Resolução

Estarei utilizando a base de dados com oversampling para os algoritmos de aprendizado não supervisionado por conta dos *melhores* resultados (se comparado com a não realização de oversampling).

### Sobre o k-Means e o Método do Cotovelo (elbow method)

O Método do Cotovelo (também conhecido por método do joelho) é uma técnica utilizada no aprendizado não supervisionado para determinar o número ideal de *clusters* em algoritmos de agrupamento, como o **k-Means**. O algoritmo de k-Means visa particionar um conjunto de dados em  $k$  clusters, onde cada ponto de dados pertence a um cluster com base em sua proximidade com o centróide desse cluster.

Com o gráfico obtido, podemos observar que a inércia diminui rapidamente à medida que o número de *clusters* aumenta em torno de 3 à 4 clusters. Após esse ponto, a taxa de diminuição torna-se mais suave.

### Sobre o Método Hierárquico

O Método Hierárquico é um algoritmo de *clustering* que agrupa objetos em uma estrutura hierárquica. Ele constrói uma árvore (dendrograma) que representa a relação de similaridade entre os objetos, onde os objetos semelhantes são agrupados em *clusters* próximos uns aos outros. A principal vantagem do método hierárquico é a sua interpretabilidade, pois ele fornece uma visualização em forma de árvore que mostra a relação de similaridade entre os objetos em diferentes níveis de agrupamento. Além disso, ele não requer um número pré-definido de *clusters*, o que pode ser uma vantagem em problemas onde o número de clusters não é conhecido antecipadamente.

## k-Means

Realizando oversampling e preparando os dados

```
In [ ]: # Realizando oversampling devido à diferença entre a quantidade de cães por grupo c
dcp = doggies.copy()

# Separa as colunas de features e o alvo
x = dcp.drop('group', axis=1)
y = dcp['group']

# Aplica o SMOTE para gerar dados sintéticos
smote = SMOTE()
x_oversampled, y_oversampled = smote.fit_resample(x, y)

# Cria um novo DataFrame com as features e o alvo oversampled
dados_oversampled = pd.DataFrame(x_oversampled, columns=x.columns)
dados_oversampled['group'] = y_oversampled

# Aplicar o Label encoding à coluna 'group' em dados_oversampled
dados_oversampled['group'] = label_encoder.fit_transform(dados_oversampled['group'])

# Definindo atributos de entrada e saída
x = dcp.drop('group', axis=1)
x = RobustScaler().fit_transform(x)

y = label_encoder.fit_transform(doggies['group'])

# Separando dados de treinamento
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_sta

# print(len(dcp)) # 277
# print(len(dados_oversampled)) # 612
```

## Gráfico Curva do Cotovelo

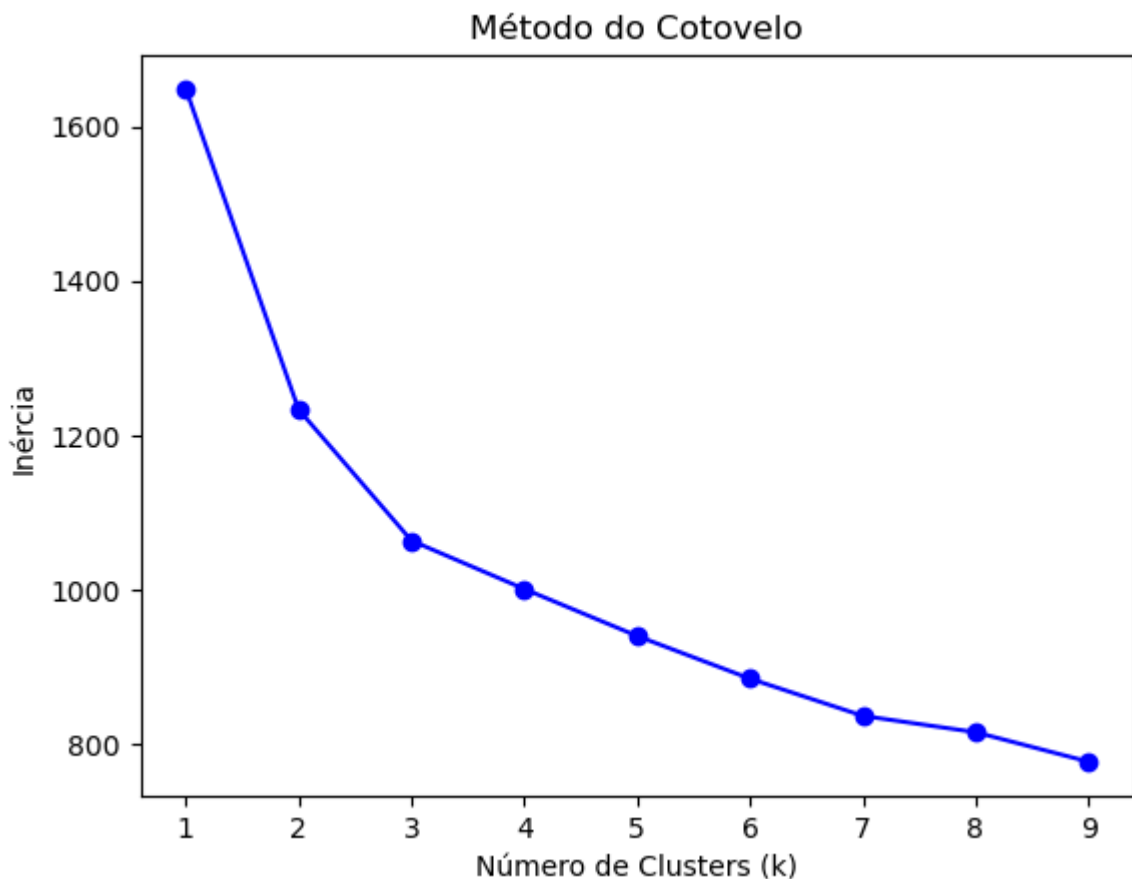
```
In [ ]: # Gráfico Curva do Cotovelo (ou Joelho)
inertia = []

# Testar diferentes valores de k
k_values = range(1, 10)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(x_train)
    inertia.append(kmeans.inertia_)

# Plotar a curva do método do cotovelo
plt.plot(k_values, inertia, 'bo-')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Inércia')
plt.title('Método do Cotovelo')
plt.show()
```

[illegible]

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



Definindo o número de cluters desejados

```
In [ ]: num_clusters = 4 # 3 ou 4, com base no gráfico da curva do cotovelo
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(x_train)

labels = kmeans.labels_ # Obtendo os rótulos dos clusters para cada ponto de dados

# Realizando previsões para novos pontos de dados
predicted_cluster = kmeans.predict(x_test)
# predicted_groups = label_encoder.inverse_transform(predicted_cluster)
# print(predicted_groups)

# Calculando a distância entre dois pontos de dados
ponto1 = dados_oversampled.loc[0]
ponto2 = dados_oversampled.loc[1]
distancia = np.linalg.norm(ponto1 - ponto2)

# Embaralhando aleatoriamente o conjunto de dados
dados_oversampled = dados_oversampled.sample(frac=1, random_state=42).reset_index(d

# Selecionando aleatoriamente K pontos de dados
K = 4
dados_selecionados = dados_oversampled.sample(n=K, random_state=42)
```

```
c:\Users\saman\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```

```
warnings.warn(
c:\Users\saman\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
warnings.warn(
```

Visualizando os clusters utilizando PCA



```
In [ ]: # Aplicar PCA para redução de dimensionalidade
pca = PCA(n_components=2)
x_pca_train = pca.fit_transform(x_train)
x_pca_test = pca.transform(x_test)

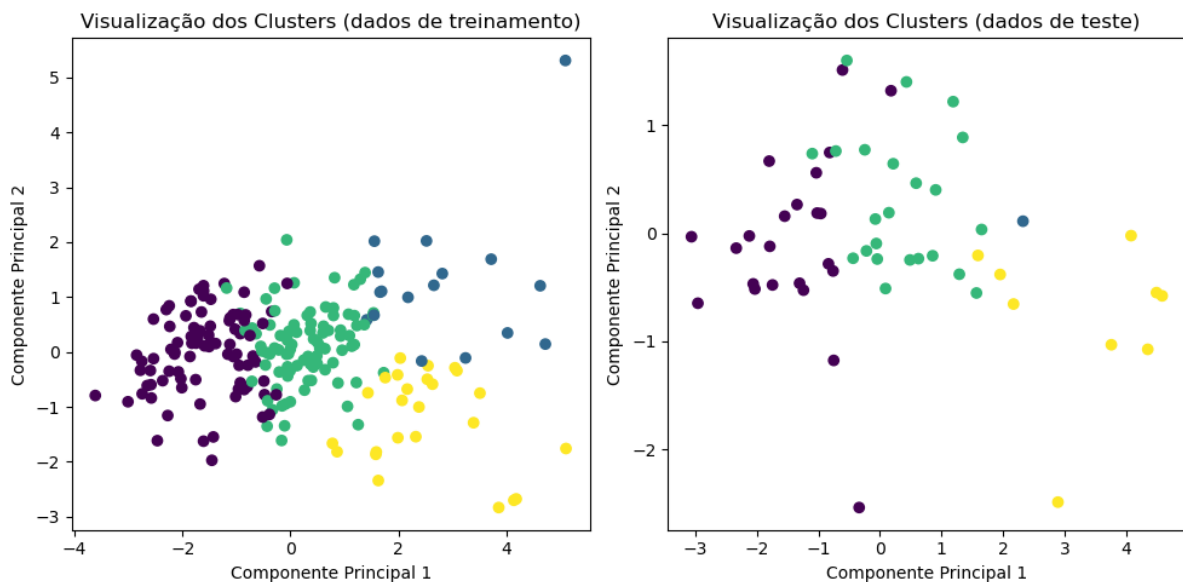
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Plotar o primeiro gráfico
axs[0].scatter(x_pca_train[:, 0], x_pca_train[:, 1], c=labels, cmap='viridis')
axs[0].set_xlabel('Componente Principal 1')
axs[0].set_ylabel('Componente Principal 2')
axs[0].set_title('Visualização dos Clusters (dados de treinamento)')

# Plotar o segundo gráfico
axs[1].scatter(x_pca_test[:, 0], x_pca_test[:, 1], c=predicted_cluster, cmap='viridis')
axs[1].set_xlabel('Componente Principal 1')
axs[1].set_ylabel('Componente Principal 2')
axs[1].set_title('Visualização dos Clusters (dados de teste)')

# Ajustar os espaçamentos entre os subplots
plt.tight_layout()

# Exibir a figura com os subplots
plt.show()
```



Contando a frequência de cada rótulo de cluster

```
In [ ]: cluster_labels, cluster_counts = np.unique(labels, return_counts=True)

predicted_groups_names = label_encoder.inverse_transform(cluster_labels)

# Imprimir o nome e a contagem de cada cluster
for name, count in zip(predicted_groups_names, cluster_counts):
    print(f"Cluster {name}: {count} pontos de dados")

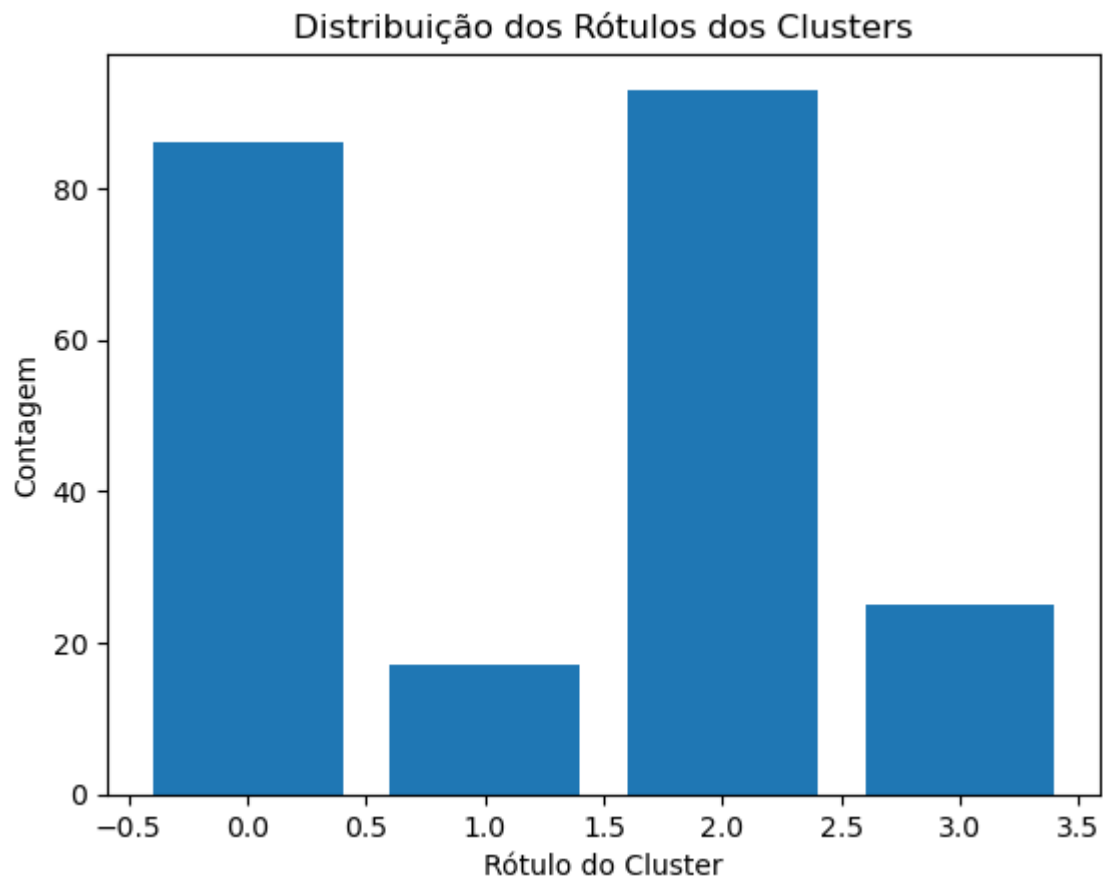
# Plotar a distribuição dos rótulos dos clusters
plt.bar(cluster_labels, cluster_counts)
plt.xlabel('Rótulo do Cluster')
plt.ylabel('Contagem')
plt.title('Distribuição dos Rótulos dos Clusters')
plt.show()
```

Cluster Foundation Stock Service: 86 pontos de dados

Cluster Herding Group: 17 pontos de dados

Cluster Hound Group: 93 pontos de dados

Cluster Miscellaneous Class: 25 pontos de dados



## Questão 6

Faça a comparação dos resultados (item 3, 4 e 5) utilizando pelo menos 2 métricas. Justifique a escolha dessas métricas e argumente sobre os resultados encontrados. Para tanto, você pode fazer um uso de gráficos e matrizes de confusão.

## Resolução

Os comparativos já foram realizados dentro das próprias questões.