

Aplicação de Reconhecimento de Padrões em Banco de Dados de Satisfação de Passageiro em Companhias Áreas

Ricardo Piero Lippoli Batista, João Vitor Santos Barbosa, Matheus Marques Duarte

I. INTRODUÇÃO

Objetivando assimilar o conhecimento adquirido em Ciência de Dados na disciplina de Reconhecimento de Padrões, foi proposto este trabalho, que consiste em utilizar os métodos de classificação estudados durante o período para classificar um banco de dados predeterminado.

A. Base de Dados

Para essa análise, foi utilizado o banco de dados "Airline Passenger Satisfaction", que essencialmente consiste em uma pesquisa de satisfação com mais de 25000 passageiros de companhias aéreas. Com o objetivo de compreender os passageiros, entender as opiniões e correlacionar os dados, foram empregados os seguintes parâmetros:

- Gênero do Passageiro
 - HOMEM
 - MULHER
- Tipo de Cliente
 - LEAL
 - DESLEAL
- Idade (anos)
 - min: 7
 - max: 85
- Tipo de Viagem
 - PESSOAL
 - NEGOCIOS
- Classe
 - ECO
 - ECO PLUS
 - EXECUTIVA
- Distância de Voo (km)
 - min: 31
 - max: 4983
- Serviço Wi-Fi a bordo
- Hora da Partida/Hora de chegada conveniente
- Facilidade de reserva online
- Localização do portão
- Comida e Bebida
- Embarque Online
- Conforto do assento
- Entretenimento a bordo
- Serviço de bordo
 - Serviço de quarto para pernas
 - Manuseio da bagagem
 - Serviço de check-in
 - Serviço de bordo
 - Limpeza
 - Atraso de partida (min)
 - min: 0
 - max: 1128
 - Atraso de chegada (min)
 - min: 0
 - max: 1115
 - Satisfação
 - SATISFEITO
 - NEUTRO OU NÃO SATISFEITO

Observe que os seis primeiros dados referem-se apenas a informações informativas sobre o passageiro, e os dados de atraso também são adquiridos, totalizando 13 dados de satisfação sobre a viagem. Os níveis de satisfação variam de 0 a 5 para todos os atributos. Entre esses, destaca-se que 18 são numéricos e 5 são textuais. A fim de verificar a eficiência dos modelos, foram calculadas as métricas: Acurácia, F1 Score, Recall, Kappa e Precisão.

II. CLASSIFICAÇÕES

Buscando trabalhar com a base de dados mencionada anteriormente, foi necessário realizar o pré-processamento dos dados para lidar com valores ausentes. Em seguida, foi gerado um histograma para cada classe, a fim de analisar e contar a quantidade de dados em cada uma. Além disso, empregou-se um código para extrair informações sobre a assimetria de cada classe. Posteriormente, foi elaborado um boxplot das variáveis com atributos numéricos. Após a conclusão dessas etapas iniciais, foi possível realizar uma busca por variáveis que apresentassem outliers no banco de dados.

A. Classificação Linear

Uma vez que o atributo "Customer Type" foi definido como classe e "Satisfaction" como o alvo, a fim de identificar atributos redundantes para a classificação linear, utilizou-se o método `corr()` da biblioteca Pandas.

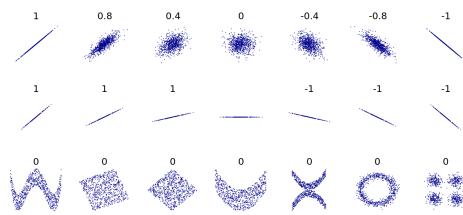


Fig. 1. Vários conjuntos de pontos (x, y), com o coeficiente de correlação de x e y para cada conjunto.

Foram utilizados três métodos diferentes para garantir a precisão da função, e após analisar o dataframe de cada método (exemplo na figura 9), foram descartados os atributos a seguir:

- Arrival Delay in Minutes
- Departure Delay in Minutes
- Cleanliness
- Inflight service
- Inflight entertainment
- Seat comfort
- Online boarding
- Ease of Online booking
- Departure/Arrival time convenient
- Inflight wifi service

Em seguida, foi realizada a Classificação Linear utilizando os atributos restantes e selecionando 500 amostras de cada classe. Dessa maneira, a classificação foi balanceada para facilitar a visualização.

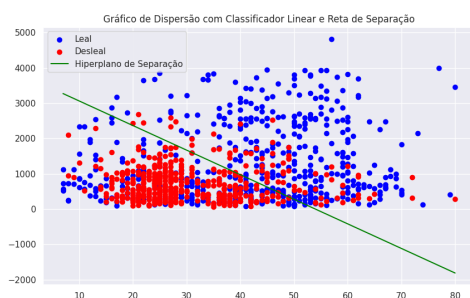


Fig. 2. Gráfico de Dispersão com Classificador Linear e Reta de Separação.

Após a classificação foi implementado um gráfico de dispersão e traçado o hiperplano de separação.

B. Naive-Bayes

A próxima Classificação realizada foi a de Naive-Bayes. Fora calculada as probabilidades de cada nível de satisfação manualmente (figura 3) para o atributo Inflight wifi service. Após isso, os atributos textuais foram convertidos em valores de -1 a 1.

```
#Frequencia:
# Não Sim
#0 = 2 = 811 = 813/25976 = 0.0313%
#1 = 2966 = 1522 = 4488/25976 = 0.1727%
#2 = 4923 = 1567 = 6490/25976 = 0.2498%
#3 = 4694 = 1623 = 6317/25976 = 0.2431%
#4 = 1953 = 3028 = 4981/25976 = 0.1917%
#5 = 35 = 2852 = 2887/25976 = 0.1111%
#Total = 14573/25976 = 0.561% | 11403/25976 = 0.439%
#TotalTudo = 25976

#Prob ser insatisfeito e dar 0:
#P0N = 2/14573 PN = 14573/25976 P0 = 813/25976
PN0 = (2/14573) * (0.561)/(0.0313)
PS0 = 1 - PN0
#Prob ser insatisfeito e dar 1:
#P0N = 2/14573 PN = 14573/25976 813/25976
PN1 = (2966/14573) * (0.561)/(0.1727)
PS1 = 1 - PN1
#Prob ser insatisfeito e dar 2:
#P0N = 2/14573 PN = 14573/25976 813/25976
PN2 = (4923/14573) * (0.561)/(0.2498)
PS2 = 1 - PN2
#Prob ser insatisfeito e dar 3:
#P0N = 2/14573 PN = 14573/25976 813/25976
PN3 = (4694/14573) * (0.561)/(0.2431)
PS3 = 1 - PN3
#Prob ser insatisfeito e dar 4:
#P0N = 2/14573 PN = 14573/25976 813/25976
PN4 = (1953/14573) * (0.561)/(0.1917)
PS4 = 1 - PN4
#Prob ser insatisfeito e dar 5:
#P0N = 2/14573 PN = 14573/25976 813/25976
PN5 = (35/14573) * (0.561)/(0.1111)
PS5 = 1/100 - PN5
```

Fig. 3. Calculo de Naive-Bayes de modo manual

Após determinar a matriz X com os valores do dataframe sem o atributo alvo e a matriz Y com os valores do atributo alvo, foi chamada a função de treino, com as matrizes produto da função anterior, foi feito os ajustes para o definir a classificação do modelo gaussiano e a classificação.

Uma vez feita a classificação, foram calculados as métricas, para validar a eficácia da classificação, junto a isso também foi feita uma matriz de confusão (figura 4) para observar o desempenho.

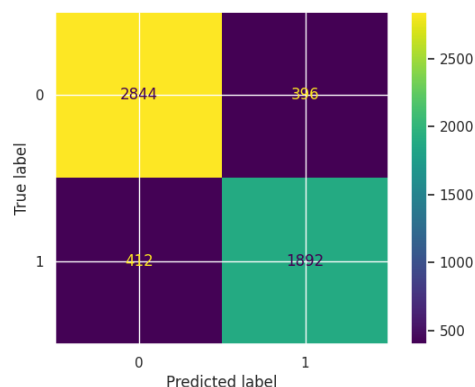


Fig. 4. Matriz de confusão referente a Naive-Bayes

C. Árvore de Decisão (simples)

Após isso, utilizando a função tree. DecisionTreeClassifier() da biblioteca sklearn, foi realizada a Classificação de Árvore de Decisão e plotado o resultado (figura 5 e em anexo). Logo em seguida também foi calculado as métricas e feita a matriz de confusão para o método de classificação (figura 6).

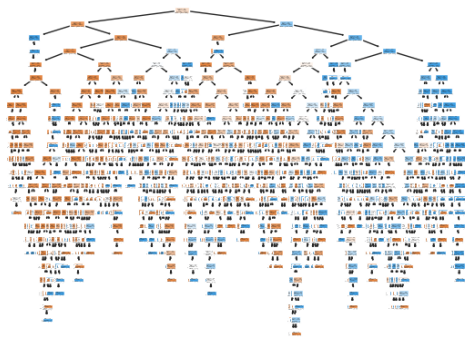


Fig. 5. Plote da classificação por árvore de decisão

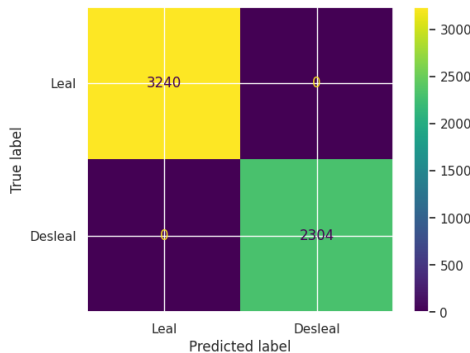


Fig. 6. Matriz de confusão referente a árvore de decisão

D. Rede Neural MLP

Utilizando a função `MLPClassifier()` da biblioteca `sklearn`, foi realizada a Classificação de rede neural MLP e após treinar o modelo, foi obtido o valor do score. Logo em seguida, foi feita a matriz de confusão para o método de classificação (figura 7).

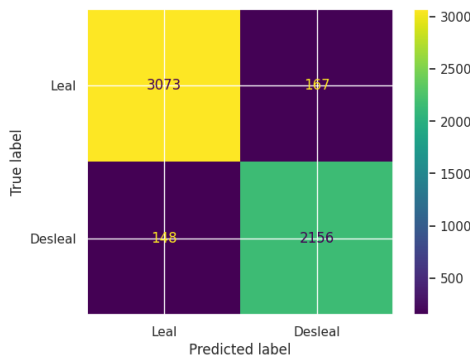


Fig. 7. Matriz de confusão referente a Rede Neural MLP

E. Random Florest

Semelhante a classificação anterior, usando a função `RandomForestClassifier()` da biblioteca `sklearn`, foi realizada a Classificação de random florest e após treinar o modelo. Logo em seguida, foi feita a matriz de confusão para o método de classificação (figura 8).

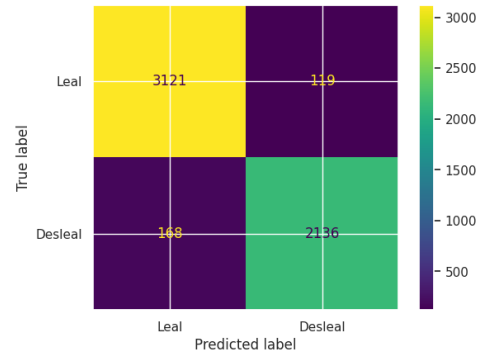


Fig. 8. Matriz de confusão referente a random florest

III. Resultados e Discussão

A. Pré-processamento

- Quantos registros a base de dados possui? 25976 Registros.
- Quantos atributos? 25 atributos.
- O que quer dizer cada atributo? 6 são dados sobre o passageiro enquanto outros 13 são notas dadas por eles para os serviços das companhias aereas.
- A base de dados possui valores ausentes? Sim, 83 dados faltantes.
- A base de dados possui outliers? Sim, 4 atributos com outliers.
- Análise monovariada e bivariada dos atributos: Exemplo da analise bivariada feita ao longo do trabalho:



Fig. 9. tabela de correlação

- Existem atributos com elevada assimetria? Sim, 2 atributos com assimetria maior que 6.
- Qual a melhor forma de normalizar os dados? No trabalho como os valores eram muito proximos, não foram usados formas de normalização. Contudo ao longo dos testes a aplicação do \log_{1p} e $\sqrt{\text{rt}}$ para os valores se mostrou eficiente.
- Existe desbalanceamento entre as classes? Sim, porem é pequeno.
- É viável fazer seleção de atributos? É viavel uma vez que são menos atributos para tratar e alguns possuem alta correlação.

B. Processamento Treinamento dos modelos

- Ajuste dos hiperparâmetros: Qual método usar?
- Qual métrica utilizada para o ajuste? Foi utilizada a metrica accuracy, F1 score, kappa, precision e recall
- Naive-Bayes: Accuracy: 0.83 | F1 Score: 0.83
- Árvore de Decisão (simples): Accuracy: 1.0 | F1 Score: 1.0
- Rede Neural MLP: Accuracy: 0.59 | F1 Score: 0.63
- Random Florest Accuracy: 0.89 | F1 Score: 0.90

C. Pós-processamento: Interpretação dos resultados

- Qual modelo desempenhou melhor? Levando em consideração tempo de processamento (medido com cronometro) e valor de metricas, o melhor modelo foi o random florest.
- tabela comparativa com os resultados da validação cruzada

	Naive-Bayes	Rede Neural MLP	Random Florest	Árvore de Decisão (simples)
Accuracy	0,854256854257	0,943181818182	0,943181818182	1,000000000000
F1 Score	0,854331275544	0,943148704229	0,948865483540	1,000000000000
Kappa	0,699660152931	0,883168679283	0,894182659273	1,000000000000
Precision	0,821180555556	0,935763888889	0,926649305556	1,000000000000
Recall	0,826923076923	0,928110202325	0,948888888889	1,000000000000
Processamento	4,500000000000	20,000000000000	4,000000000000	60,000000000000
Score/Tempo	0,96	-13,60	2,45	-53,00
Score	5,46	6,46	6,45	7,00
Final	-4,50	-20,00	-4,00	-60,00

Fig. 10. tabela para comparação das metricas

- Curva ROC Final

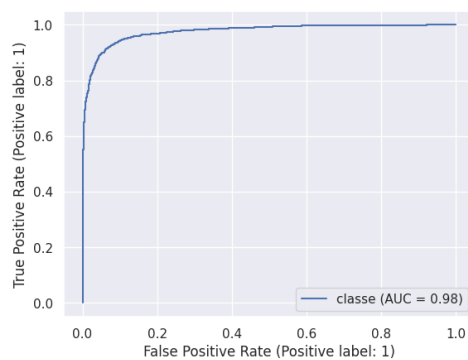


Fig. 11. Curva ROC Final

- Matriz de confusão do melhor modelo: figura 8

IV. CONCLUSÕES

Em conclusão, o estudo sobre reconhecimento de padrões revelou pontos significativos sobre a eficácia de diversos classificadores. O classificador de árvore de decisões simples demonstrou uma performance excepcional, liderando em termos de acurácia e F1 Score, refletindo sua capacidade de aprender padrões complexos. Por outro lado, o Random Forest apresentou um desempenho sólido, figurando como uma alternativa eficiente com um tempo de execução aceitável. Destaca-se que, embora a Árvore de Decisão tenha revelado competência em classificação, sua demora na execução sugere a necessidade de considerações adicionais em relação à eficiência temporal, fator crucial em muitos contextos práticos. Além disso, a exploração de arquiteturas de redes neurais alternativas e ajustes nos parâmetros pode proporcionar melhorias adicionais em todos os modelos propostos.

Em suma, este trabalho não apenas aprofundou a compreensão dos métodos de reconhecimento de padrões, mas também forneceu orientações valiosas para a escolha prática de classificadores, considerando a dualidade entre desempenho e eficiência computacional.

▼ Bibliotecas

```
## Importando as Bibliotecas Necessárias ##
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, confusion_matrix, ConfusionMatrixDisplay, f1_score, cohen_kappa_score, precision_recall_fscore_support
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import make_classification
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from numpy.linalg import inv
from copy import copy
from sklearn.metrics import RocCurveDisplay

sns.set()

# Lendo a base de dados
df = pd.read_excel("m.xlsx")

xp = copy(df)

xraw = copy(xp)

xp.shape

(25976, 25)
```

▼ Código processamento

```
df = copy(xp)
df.isnull().sum()

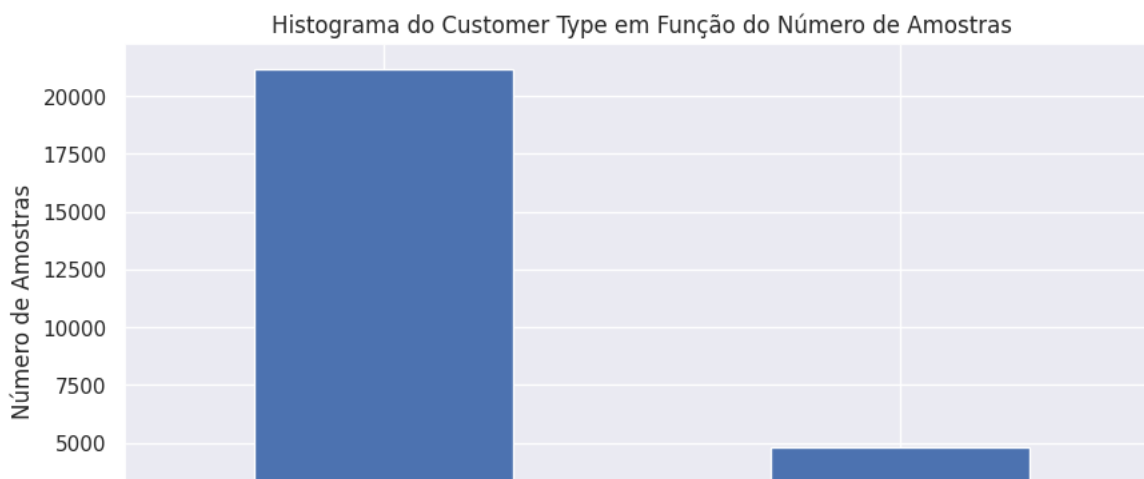
Column1      0
id            0
Gender        0
Customer Type 0
Age           0
Type of Travel 0
Class         0
Flight Distance 0
Inflight wifi service 0
Departure/Arrival time convenient 0
Ease of Online booking 0
Gate location 0
Food and drink 0
Online boarding 0
Seat comfort 0
Inflight entertainment 0
On-board service 0
Leg room service 0
Baggage handling 0
Checkin service 0
Inflight service 0
Cleanliness 0
Departure Delay in Minutes 0
Arrival Delay in Minutes 0
satisfaction 0
dtype: int64

xp['Arrival Delay in Minutes'] = xp['Arrival Delay in Minutes'].replace(np.nan, 0)

# Contando a quantidade de dados de cada classe
df=xp
contagem_classes = df['Customer Type'].value_counts()

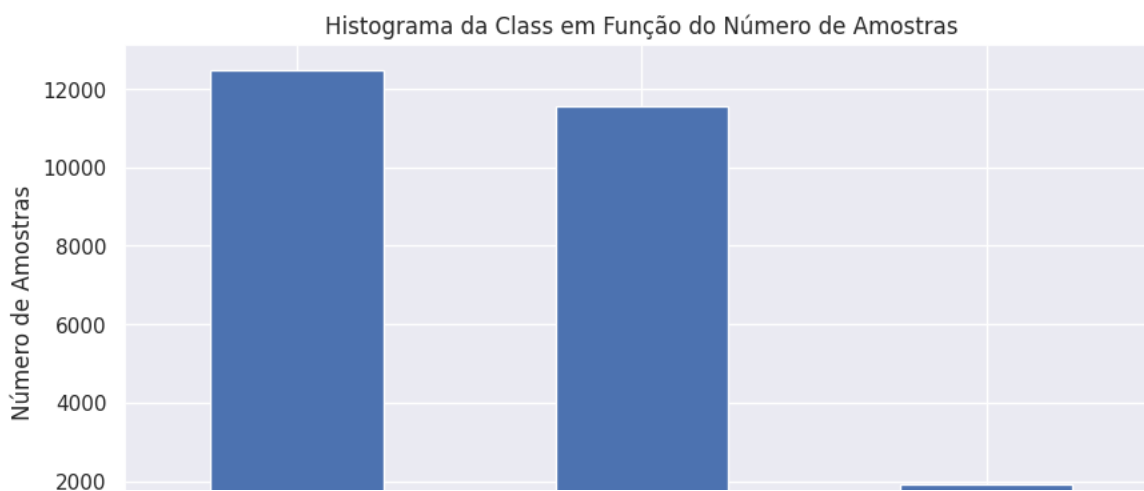
# Plotando o histograma das classes
plt.figure(figsize=(10, 5))
contagem_classes.plot(kind='bar')
```

```
plt.xlabel('Customer Type')
plt.ylabel('Número de Amostras')
plt.title('Histograma do Customer Type em Função do Número de Amostras')
plt.xticks(rotation=360)
plt.show()
```



```
# Contando a quantidade de dados de cada classe
df=xp
contagem_classes = df['Class'].value_counts()

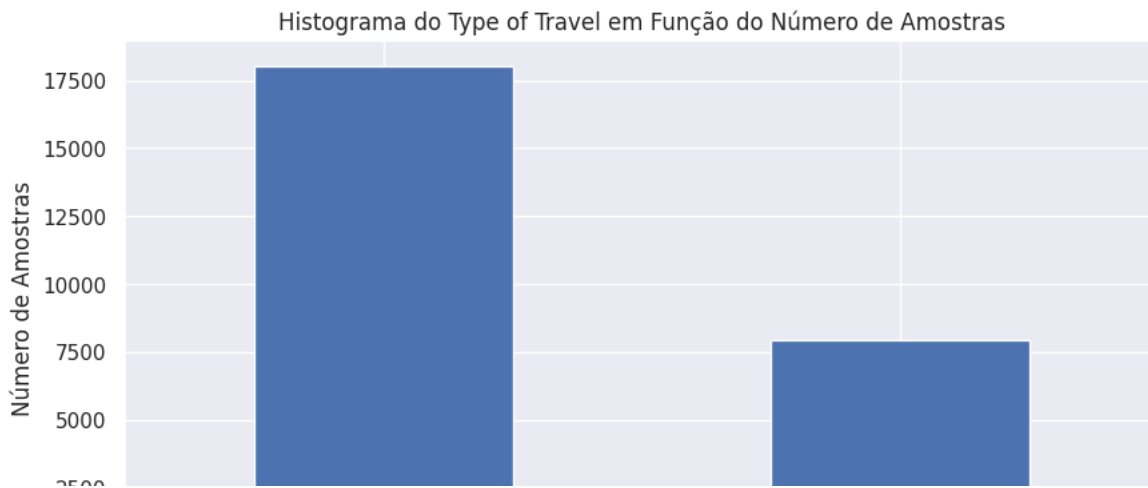
# Plotando o histograma das classes
plt.figure(figsize=(10, 5))
contagem_classes.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Número de Amostras')
plt.title('Histograma da Class em Função do Número de Amostras')
plt.xticks(rotation=360)
plt.show()
```



```
# Contando a quantidade de dados de cada classe
df=xp
contagem_classes = df['Type of Travel'].value_counts()

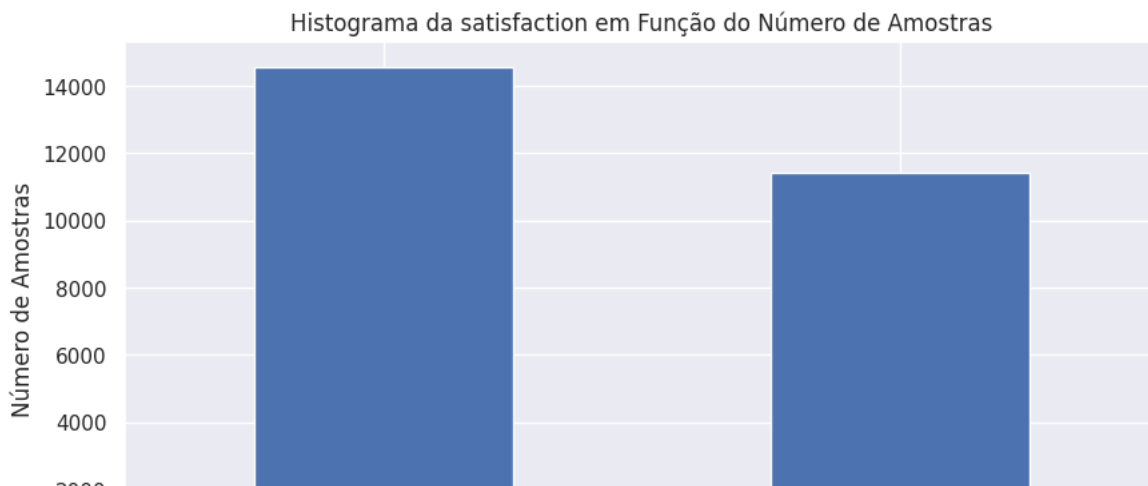
# Plotando o histograma das classes
plt.figure(figsize=(10, 5))
contagem_classes.plot(kind='bar')
plt.xlabel('Type of Travel')
plt.ylabel('Número de Amostras')
plt.title('Histograma do Type of Travel em Função do Número de Amostras')
```

```
plt.xticks(rotation=360)
plt.show()
```



```
# Contando a quantidade de dados de cada classe
df=xp
contagem_classes = df['satisfaction'].value_counts()

# Plotando o histograma das classes
plt.figure(figsize=(10, 5))
contagem_classes.plot(kind='bar')
plt.xlabel('satisfaction')
plt.ylabel('Número de Amostras')
plt.title('Histograma da satisfaction em Função do Número de Amostras')
plt.xticks(rotation=360)
plt.show()
```



```
atributos_numericos = df.select_dtypes(include=['float64', 'int64'])
```

```
# Calcula a assimetria para cada atributo numérico
assimetria = atributos_numericos.skew()
```

```
# Exibe a assimetria de cada atributo
print(assimetria)
```

```
Column1          0.000000
id              -0.011364
Age             -0.000088
Flight Distance  1.102927
Inflight wifi service  0.040792
Departure/Arrival time convenient -0.324685
```

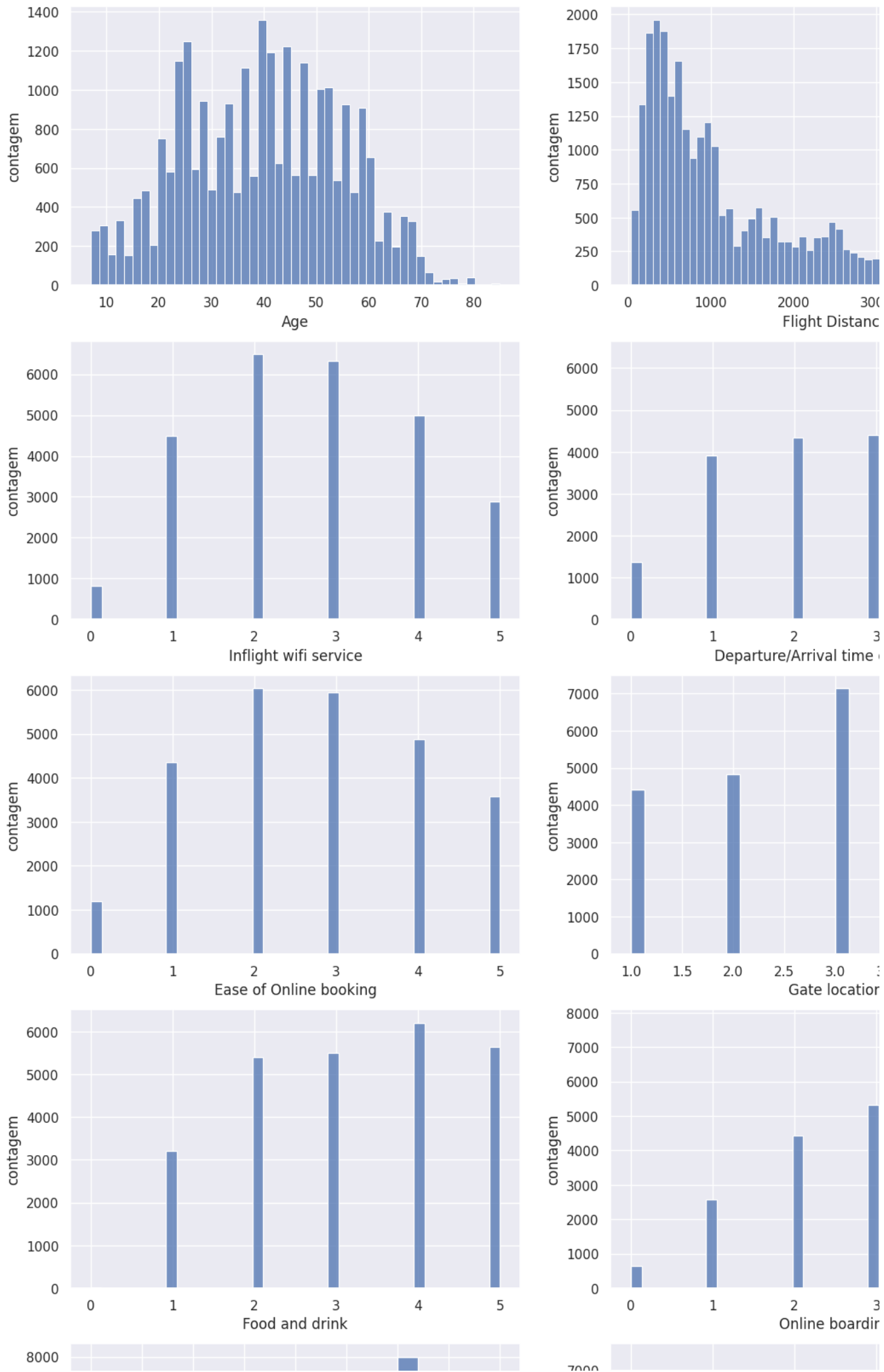
Ease of Online booking	-0.020655
Gate location	-0.055798
Food and drink	-0.170229
Online boarding	-0.469218
Seat comfort	-0.498047
Inflight entertainment	-0.371353
On-board service	-0.426507
Leg room service	-0.341210
Baggage handling	-0.678863
Checkin service	-0.373016
Inflight service	-0.696803
Cleanliness	-0.304285
Departure Delay in Minutes	7.193970
Arrival Delay in Minutes	6.994086

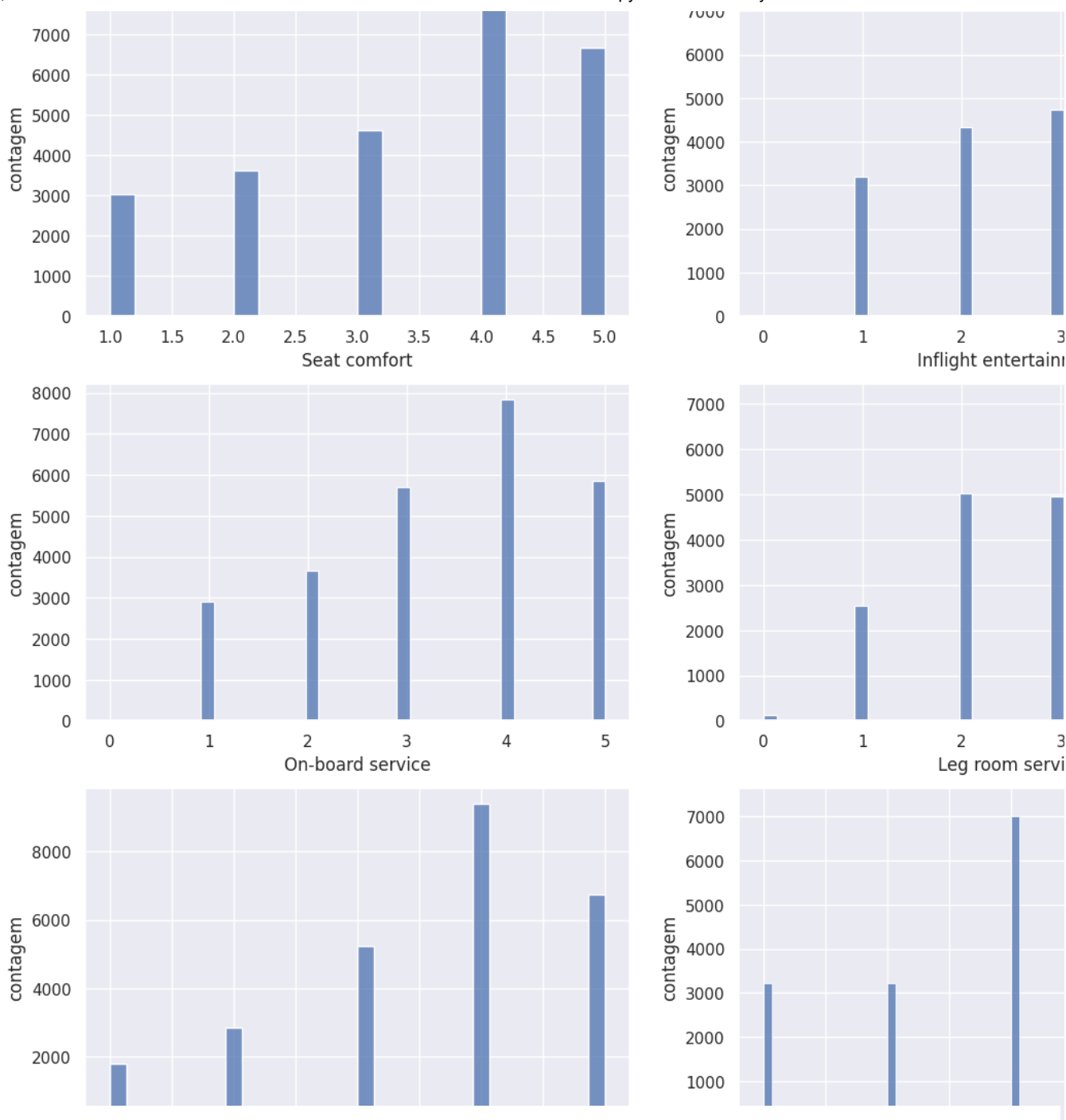
dtype: float64

#aplicando histogramas para os atributos contínuos:

```
def multigraficos_histograma(data, nrows, ncols, nomes):
    fig, axs = plt.subplots(nrows = nrows, ncols = ncols, figsize=(15, 5*nrows))
    k = 0
    for i in range(nrows):
        for j in range(ncols):
            sns.histplot(ax = axs[i,j], data = data, x = nomes[k])
            axs[i, j].set_xlabel(nomes[k])
            axs[i, j].set_ylabel("contagem")
            k+=1
        #
    #
#

nrows = 9
ncols = 2
multigraficos_histograma(df, nrows, ncols, var_num)
```



```
# boxplot
```

```
var_num = ['Age', 'Flight Distance', 'Inflight wifi service', 'Departure/Arrival time convenient', 'Ease of Online booking',
            'Gate location', 'Food and drink', 'Online boarding', 'Seat comfort', 'Inflight entertainment', 'On-board service',
            'Leg room service', 'Baggage handling', 'Checkin service', 'Inflight service', 'Cleanliness',
            'Departure Delay in Minutes', 'Arrival Delay in Minutes']
```

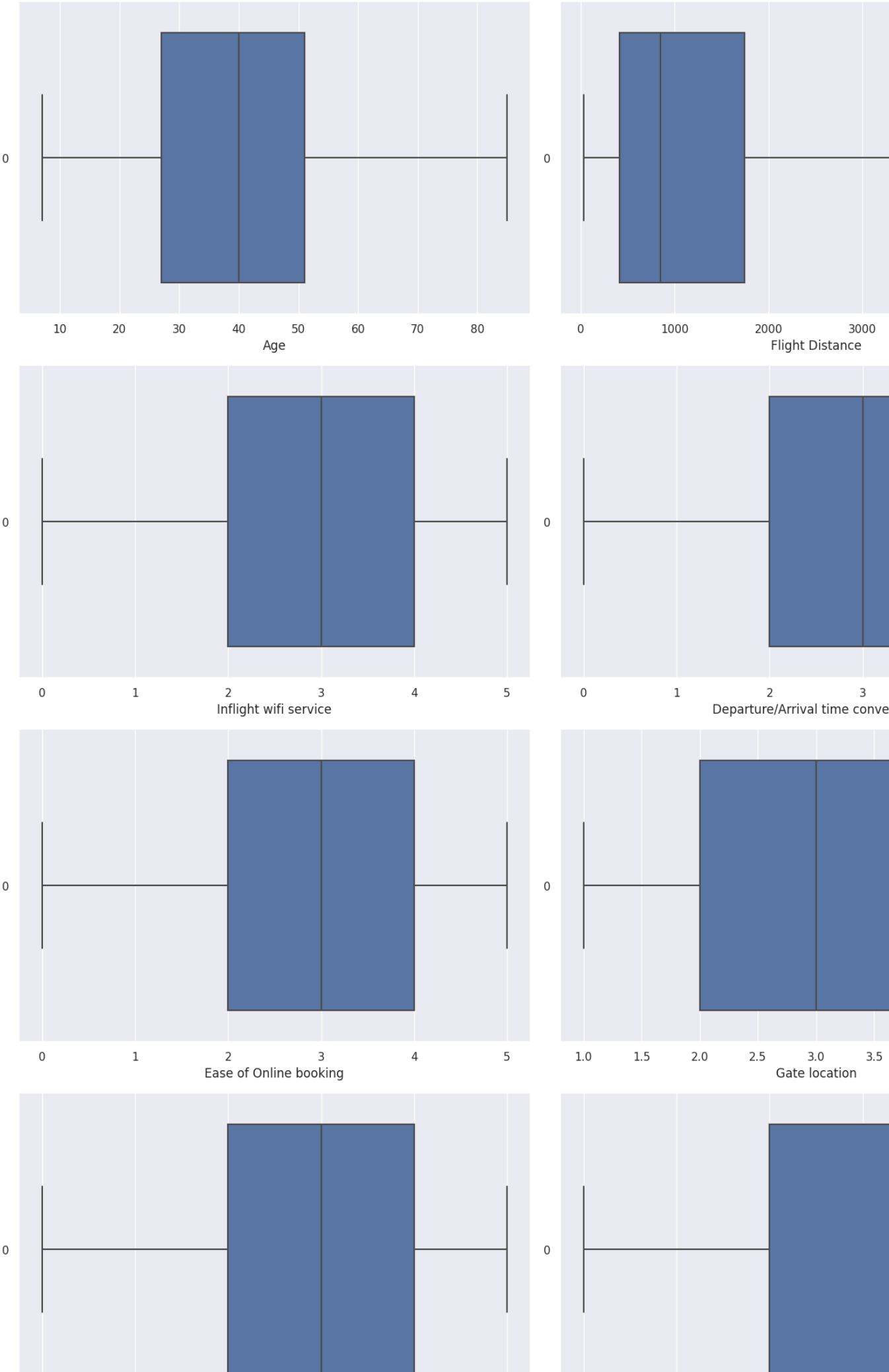
```
nrows = 9
```

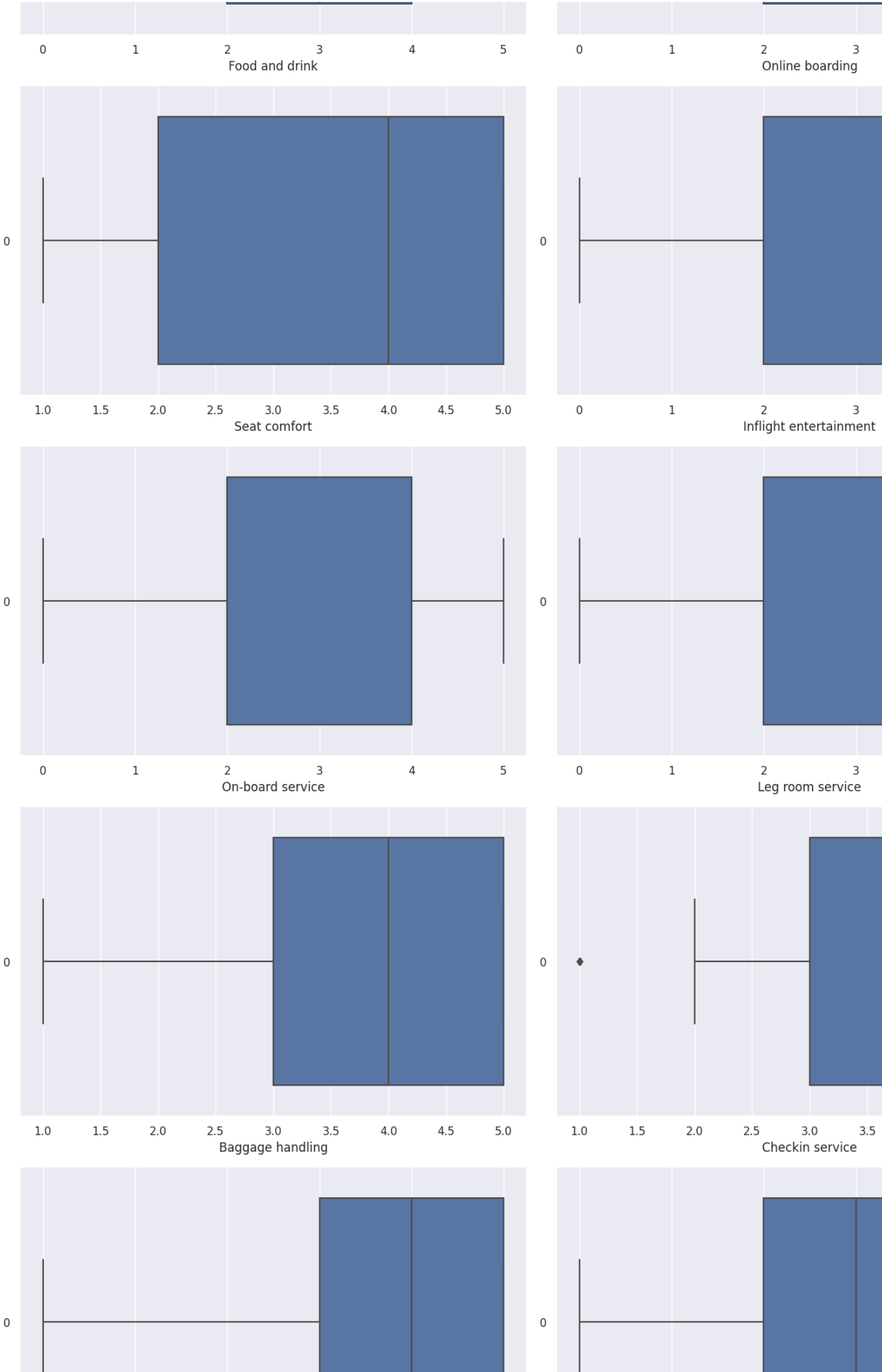
```
ncols = 2
```

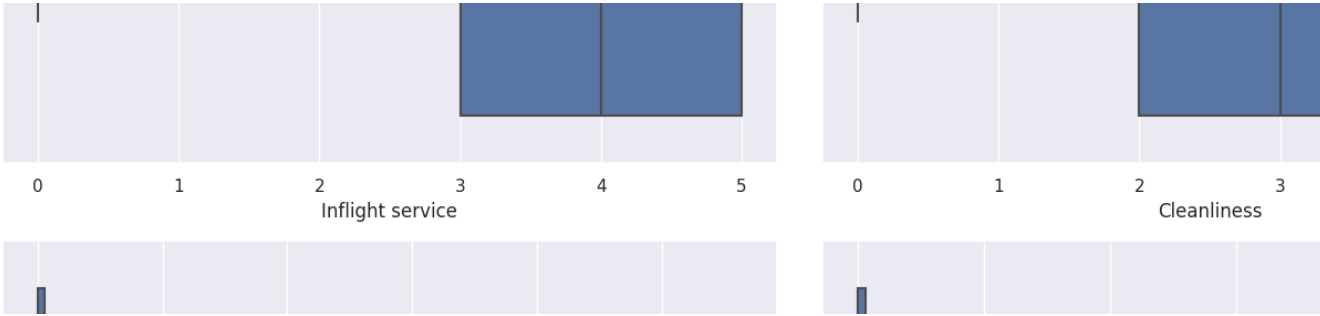
```
fig, axs = plt.subplots(nrows=nrows, ncols=ncols, figsize=(15, 5*nrows))
```

```
for i, var in enumerate(var_num):
    row = i // ncols
    col = i % ncols
    sns.boxplot(ax=axs[row, col], data=df[var].dropna(), orient='h')
    axs[row, col].set_xlabel(var)
```

```
plt.tight_layout()
```







```
print("Variaveis com outliers: \n")

dsc = df[var_num].describe() #obtendo estatísticas descritivas dos atributos numericos

for name in dsc.columns:
    q1 = dsc[name]["25%"]
    q3 = dsc[name]["75%"]
    iqr = q3-q1

    min_ = q1 - 1.5*iqr
    max_ = q3 + 1.5*iqr

    out_inf = np.where(df[name] < min_)[0]
    out_sup = np.where(df[name] > max_)[0]

    if ((out_inf.shape[0] > 0) or (out_sup.shape[0] > 0)):
        print(name)
#

Variaveis com outliers:

Flight Distance
Checkin service
Departure Delay in Minutes
Arrival Delay in Minutes
```

↘ Classificação Linear

```
df.corr(method='spearman').style.background_gradient(cmap='coolwarm')
```

	Column1	id	Age	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink	Online boarding	Seat comfort
Column1	1.000000	0.006889	-0.008206	-0.002322	-0.003532	-0.003789	-0.002222	-0.001078	-0.010043	0.001093	-0.004468
id	0.006889	1.000000	0.014094	0.134367	-0.031660	-0.003406	0.009216	0.002237	-0.007904	0.056276	0.051489
Age	-0.008206	0.014094	1.000000	0.074394	0.007254	0.029867	0.010127	0.001820	0.021836	0.206976	0.153589
Flight Distance	-0.002322	0.134367	0.074394	1.000000	0.002596	-0.008876	0.057619	0.006139	0.044451	0.192243	0.138861
Inflight wifi service	-0.003532	-0.031660	0.007254	0.002596	1.000000	0.343469	0.707514	0.345670	0.119902	0.435962	0.111958
Departure/Arrival time convenient	-0.003789	-0.003406	0.029867	-0.008876	0.343469	1.000000	0.442511	0.463909	-0.017700	0.070337	-0.001597
Ease of Online booking	-0.002222	0.009216	0.010127	0.057619	0.707514	0.442511	1.000000	0.469197	0.021621	0.369473	0.017355
Gate location	-0.001078	0.002237	0.001820	0.006139	0.345670	0.463909	0.469197	1.000000	-0.010186	0.002201	-0.002925
Food and drink	-0.010043	-0.007904	0.021836	0.044451	0.119902	-0.017700	0.021621	-0.010186	1.000000	0.236103	0.560461
Online boarding	0.001093	0.056276	0.206976	0.192243	0.435962	0.070337	0.369473	0.002201	0.236103	1.000000	0.437239
Seat comfort	-0.004468	0.051489	0.153589	0.138861	0.111958	-0.001597	0.017355	-0.002925	0.560461	0.437239	1.000000
Inflight entertainment	-0.011029	-0.000835	0.073733	0.112542	0.192678	-0.025824	0.039318	-0.002203	0.613431	0.296994	0.607135
On-board service	0.000951	0.054226	0.067881	0.109271	0.109330	0.061351	0.038749	-0.032194	0.050640	0.171435	0.141125
Leg room service	-0.003310	0.038373	0.044322	0.119719	0.149755	-0.001714	0.103574	-0.002347	0.033438	0.132556	0.112542
Baggage handling	0.001747	0.066637	-0.029406	0.069701	0.109526	0.078020	0.035720	-0.015915	0.050351	0.119922	0.108020

```

df = copy(xp)
leal = df[df['Customer Type'] == 'Loyal Customer'].iloc[:500]
desleal = df[df['Customer Type'] == 'disloyal Customer'].iloc[:500]

df = pd.concat([leal, desleal])

df['Customer Type'] = df['Customer Type'].map({'Loyal Customer': 1, 'disloyal Customer': -1})

df['Age'] = (df['Age'])
df['Flight Distance'] = (df['Flight Distance'])
df['Gate location'] = (df['Gate location'])
df['Food and drink'] = (df['Food and drink'])
df['On-board service'] = (df['On-board service'])
df['Leg room service'] = (df['Leg room service'])
df['Baggage handling'] = (df['Baggage handling'])
df['Checkin service'] = (df['Checkin service'])

X = df[['Age', 'Flight Distance', 'Checkin service', 'Baggage handling', 'Leg room service', 'On-board service', 'Food and drink', 'Gate location']]
y = df['Customer Type'].values

X = np.concatenate((X, np.ones((X.shape[0], 1)))), axis=1)

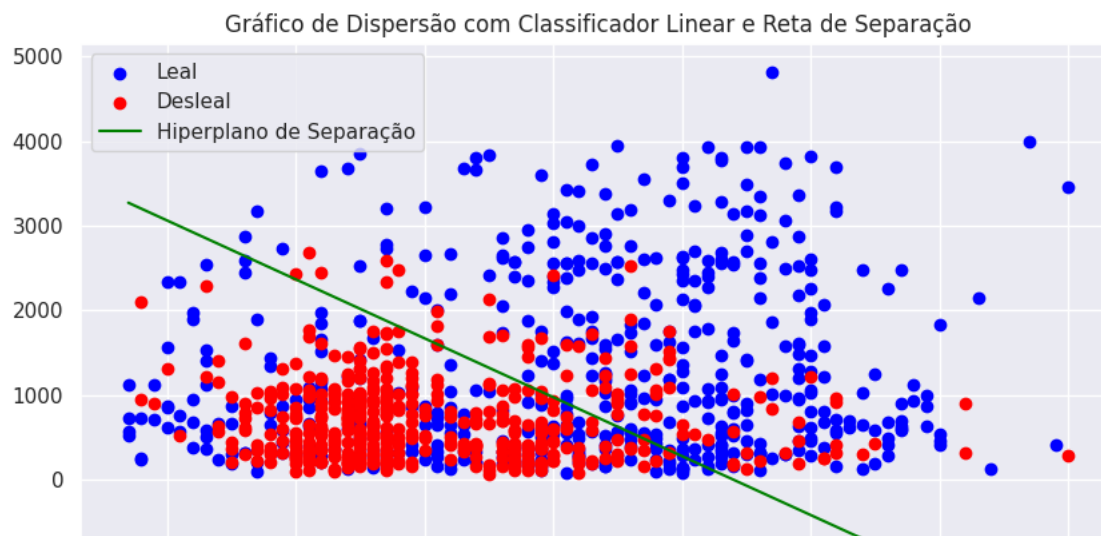
w = np.dot(inv(np.dot(X.T, X)), np.dot(X.T, y))

w1, w2, w0 = w[0], w[1], w[8]

x_hyperplane = np.linspace(X[:, 0].min(), X[:, 0].max(), 1000)
y_hyperplane = (-w0 - w1 * x_hyperplane) / w2

plt.figure(figsize=(10, 6))
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], label='Leal', c='blue')
plt.scatter(X[y == -1][:, 0], X[y == -1][:, 1], label='Desleal', c='red')
plt.plot(x_hyperplane, y_hyperplane, color='green', label='Hiperplano de Separação')
plt.legend()
plt.title('Gráfico de Dispersão com Classificador Linear e Reta de Separação')
plt.grid(True)
plt.show()

```



✓ Criação dos dados iniciais

```

df = copy(xp)

leal = df[df['Customer Type'] == 'Loyal Customer'].iloc[:12000]
desleal = df[df['Customer Type'] == 'disloyal Customer'].iloc[:12000]

df = pd.concat([leal, desleal])

df['Customer Type'] = df['Customer Type'].map({'Loyal Customer': 1, 'disloyal Customer': -1})
df['Gender'] = df['Gender'].map({'Male': 1, 'Female': 0})
df['Type of Travel'] = df['Type of Travel'].map({'Business travel': 1, 'Personal Travel': 0})
df['Class'] = df['Class'].map({'Business': 1, 'Eco': -1, 'Eco Plus': 0})
df['satisfaction'] = df['satisfaction'].map({'satisfied': 1, 'neutral or dissatisfied': 0})

```

```
df['Flight Distance'] = np.log(df['Flight Distance'])
df['Arrival Delay in Minutes'] = np.log1p(df['Arrival Delay in Minutes'])
df['Age'] = np.sqrt(df['Age'])
df['Departure Delay in Minutes'] = np.log1p(df['Arrival Delay in Minutes'])

df.drop(['id'], axis = 1, inplace = True)
df.drop(['Column1'], axis = 1, inplace = True)
df.drop(['Cleanliness'], axis = 1, inplace = True)
df.drop(['Ease of Online booking'], axis = 1, inplace = True)
df.drop(['Inflight entertainment'], axis = 1, inplace = True)
df.drop(['Arrival Delay in Minutes'], axis = 1, inplace = True)

X = df.drop(['satisfaction'], axis = 1).values
y = df['satisfaction'].values
```

✓ Classificação Naive-Bayes

```
#Frequencia:
#      Não      Sim
#0 = 2      = 811      = 813/25976      = 0.0313%
#1 = 2966    = 1522    = 4488/25976    = 0.1727%
#2 = 4923    = 1567    = 6490/25976    = 0.2498%
#3 = 4694    = 1623    = 6317/25976    = 0.2431%
#4 = 1953    = 3028    = 4981/25976    = 0.1917%
#5 = 35      = 2852    = 2887/25976    = 0.1111%
#Total = 14573/25976 = 0.561% | 11403/25976 = 0.439%
#TotalTudo = 25976

#Prob ser insatisfeito e dar 0:
#P0N = 2/14573 PN = 14573/25976 P0 = 813/25976
PN0 = (2/14573) * (0.561)/(0.0313)
PS0 = 1 - PN0
#Prob ser insatisfeito e dar 1:
#P0N = 2/14573 PN = 14573/25976 813/25976
PN1 = (2966/14573) * (0.561)/(0.1727)
PS1 = 1 - PN1
#Prob ser insatisfeito e dar 2:
#P0N = 2/14573 PN = 14573/25976 813/25976
PN2 = (4923/14573) * (0.561)/(0.2498)
PS2 = 1 - PN2
#Prob ser insatisfeito e dar 3:
#P0N = 2/14573 PN = 14573/25976 813/25976
PN3 = (4694/14573) * (0.561)/(0.2431)
PS3 = 1 - PN3
#Prob ser insatisfeito e dar 4:
#P0N = 2/14573 PN = 14573/25976 813/25976
PN4 = (1953/14573) * (0.561)/(0.1917)
PS4 = 1 - PN4
#Prob ser insatisfeito e dar 5:
#P0N = 2/14573 PN = 14573/25976 813/25976
PN5 = (35/14573) * (0.561)/(0.1111)
PS5 = 1/100 - PN5

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=125, stratify = y
)

# Build a Gaussian Classifier
clf = GaussianNB()

# Model training
clf.fit(X_train, y_train)

# Predict Output
predicted = clf.predict([X_test[6]])

print("Actual Value:", y_test[6])
print("Predicted Value:", predicted[0])

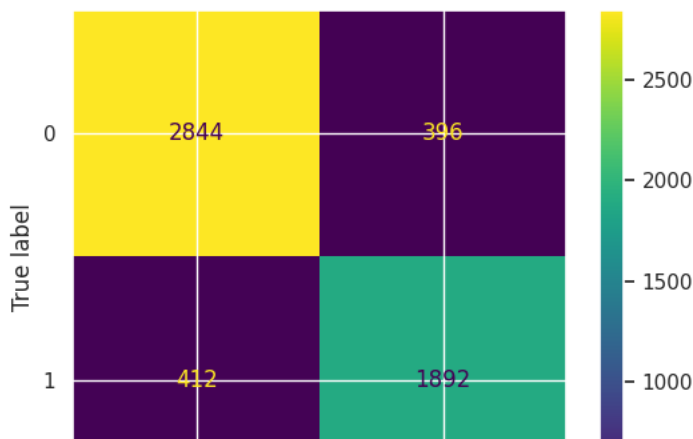
Actual Value: 0
Predicted Value: 0

y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test, average="weighted")
kappa = cohen_kappa_score(y_pred, y_test)
prec = precision_score(y_pred, y_test)
recal = recall_score(y_pred, y_test)

print("Accuracy:", accuracy)
```

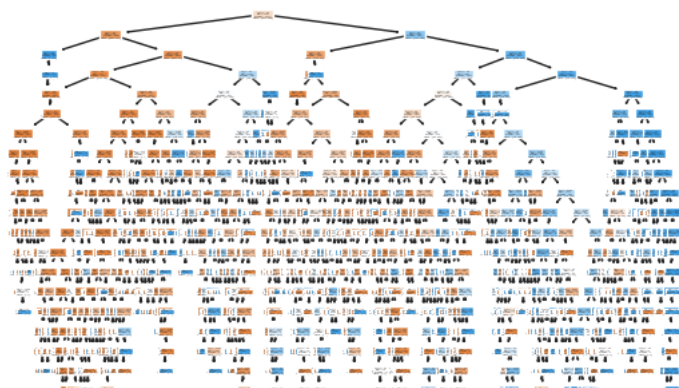


```
labels = [0,1]
cm = confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot();
```



- ✓ Classificação Árvore de Decisão (simples)

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y.astype(float))
tree.plot_tree(clf, filled=True)
y_pred = clf.predict(X_test)
```



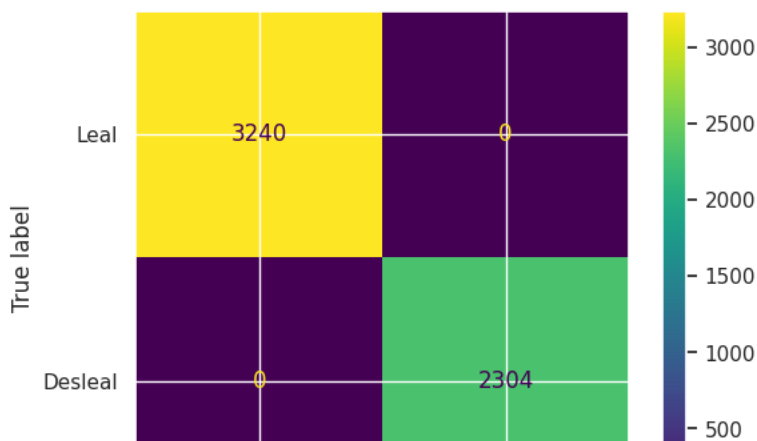
```
accuracy = accuracy_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test, average="weighted")
kappa = cohen_kappa_score(y_pred, y_test)
prec = precision_score(y_pred, y_test)
recal = recall_score(y_pred, y_test)

print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Kappa:", kappa)
```

```
print("Precision:", prec)
print("Recall:", recal)
```

```
Accuracy: 1.0
F1 Score: 1.0
Kappa: 1.0
Precision: 1.0
Recall: 1.0
```

```
labels = ["Leal", "Desleal"]
cm = confusion_matrix(y_test, y_pred)
labels = ["Leal", "Desleal"]
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot();
```



✓ Classificação Rede Neural MLP

```
clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
clf.predict_proba(X_test[:1])
y_pred = clf.predict(X_test)
```

```
accuracy = accuracy_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test, average="weighted")
kappa = cohen_kappa_score(y_pred, y_test)
prec = precision_score(y_pred, y_test)
recal = recall_score(y_pred, y_test)
```

```
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Kappa:", kappa)
print("Precision:", prec)
print("Recall:", recal)
```

```
Accuracy: 0.9431818181818182
F1 Score: 0.9431487042288005
Kappa: 0.8831686792834539
Precision: 0.9357638888888888
Recall: 0.9281102023245803
```

```
labels = ["Leal", "Desleal"]
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot();
```



Random Florest

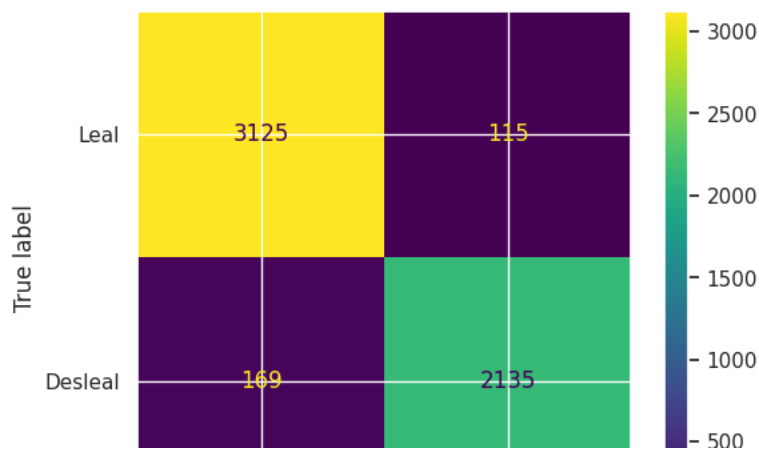
```
clf = RandomForestClassifier(n_estimators = 800)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
accuracy = metrics.accuracy_score(y_test, y_pred)
f1 = f1_score(y_pred, y_test, average="weighted")
kappa = cohen_kappa_score(y_pred, y_test)
prec = precision_score(y_pred, y_test)
recal = recall_score(y_pred, y_test)
```

```
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Kappa:", kappa)
print("Precision:", prec)
print("Recall:", recal)
```

```
Accuracy: 0.9431818181818182
F1 Score: 0.9488654835402949
Kappa: 0.8941826592728082
Precision: 0.9266493055555556
Recall: 0.9488888888888889
```

```
labels = ["Leal", "Desleal"]
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot();
```



ROC

```
clf = SVC(random_state=0).fit(X_train, y_train)
y_pred = clf.decision_function(X_test)
RocCurveDisplay.from_predictions(y_test, y_pred, name='classe')
plt.show()
```

