Fraude_Supply_Chain

February 25, 2024

1 DataCo Global - uma empresa global do ramo de supply chain

A Empresa **DataCo Global** é uma companhia de escala global que atua no setor de supply chain envolvendo diversos produtos desde eletrônicos, brinquedos, acessórios esportivos, acessórios para acampamento e até livros.

Aqui será realizada uma análise detalhada das suas vendas e lucros diante de diversos aspectos como **categorias**, **departamentos** e **localização de clientes**, avaliando também problemas logísticos como **atrasos em entregas** e problemas com **operações fraudulentas**.

Venha descobrir insights valiosos sobre a empresa, incluindo como andam as suas vendas, os seus lucros e suas operações de logística!

1. Importando as bibliotecas e o dataset

[1]:
```
# Importando a biblioteca para manipulação de bases de dados
import pandas as pd
import locale

#Importando a biblioteca para manipulação algébrica
import numpy as np

# Bibliotecas para a EDA
import missingno
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud

# Visualização de gráficos interativos
from plotly.io import write_image

# Estatística
from scipy.stats import skew
from scipy.stats import chi2_contingency

# importando as funções Stratified K-Fold e train_test_split
```

```
from  sklearn.model_selection import train_test_split, StratifiedKFold

# Importando os modelos
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from imblearn.ensemble import BalancedRandomForestClassifier

# Feature Importance
from sklearn.inspection import permutation_importance
from sklearn.feature_selection import RFE

# importando as funções para calcular a precisão, revocação,␣
 ↪precision_recall_auc, roc_auc, medida F1 e acurácia
from sklearn.metrics import precision_score, recall_score,␣
 ↪precision_recall_curve, auc, roc_auc_score, f1_score, accuracy_score,␣
 ↪confusion_matrix

# Encoder para tratamento de variáveis categóricas
from category_encoders import CatBoostEncoder

# Pipelines
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

# Importando biblioteca para tunagem de hiperparâmetros
import optuna as opt

# Setando diretório de trabalho
import os
os.chdir(r"C:
 ↪\Users\edd-j\Downloads\Conteudos\Portfolio\Fraudes\Fraude_Supply_Chain")

# Configurar para não exibir warnings
from warnings import filterwarnings
filterwarnings('ignore')

# Exibindo todas as colunas das bases de dados
pd.set_option('display.max_columns', None)

# Ajustando a configuração de exibição do Pandas para mostrar todo o conteúdo␣
 ↪das colunas
pd.set_option('display.max_colwidth', None)
```

[2]:
```
# Importando bases de dados

df = pd.read_csv('DataCoSupplyChainDataset.csv', encoding='ISO-8859-1')
```

```
df_descricao = pd.read_csv('DescriptionDataCoSupplyChain.csv')
```

2. Dicionário dos dados

[3]:
```
# Limpeza dos dados para remover caracteres extras como ':'
df_descricao['FIELDS'] = df_descricao['FIELDS'].str.strip()
df_descricao['DESCRIPTION'] = df_descricao['DESCRIPTION'].str.strip().str.
 ↪lstrip(':').str.strip()

# Apresentando a descrição das colunas da base de dados
df_descricao
```

[3]:

	FIELDS	\
0	Type	
1	Days for shipping (real)	
2	Days for shipment (scheduled)	
3	Benefit per order	
4	Sales per customer	
5	Delivery Status	
6	Late_delivery_risk	
7	Category Id	
8	Category Name	
9	Customer City	
10	Customer Country	
11	Customer Email	
12	Customer Fname	
13	Customer Id	
14	Customer Lname	
15	Customer Password	
16	Customer Segment	
17	Customer State	
18	Customer Street	
19	Customer Zipcode	
20	Department Id	
21	Department Name	
22	Latitude	
23	Longitude	
24	Market	
25	Order City	
26	Order Country	
27	Order Customer Id	
28	order date (DateOrders)	
29	Order Id	
30	Order Item Cardprod Id	
31	Order Item Discount	
32	Order Item Discount Rate	
33	Order Item Id	

```
34       Order Item Product Price
35        Order Item Profit Ratio
36            Order Item Quantity
37                          Sales
38               Order Item Total
39         Order Profit Per Order
40                   Order Region
41                    Order State
42                   Order Status
43                Product Card Id
44            Product Category Id
45            Product Description
46                  Product Image
47                   Product Name
48                  Product Price
49                 Product Status
50     Shipping date (DateOrders)
51                  Shipping Mode

                                                    DESCRIPTION
0
Type of transaction made
1
Actual shipping days of the purchased product
2
Days of scheduled delivery of the purchased product
3
Earnings per order placed
4
Total sales per customer made per customer
5
Delivery status of orders: Advance shipping , Late delivery , Shipping canceled
, Shipping on time
6
Categorical variable that indicates if sending is late (1), it is not late (0).
7
Product category code
8
Description of the product category
9
City where the customer made the purchase
10
Country where the customer made the purchase
11
Customer's email
12
Customer name
```

13
Customer ID
14
Customer lastname
15
Masked customer key
16
Types of Customers: Consumer , Corporate , Home Office
17
State to which the store where the purchase is registered belongs
18
Street to which the store where the purchase is registered belongs
19
Customer Zipcode
20
Department code of store
21
Department name of store
22
Latitude corresponding to location of store
23
Longitude corresponding to location of store
24
Market to where the order is delivered : Africa , Europe , LATAM , Pacific Asia
, USCA
25
Destination city of the order
26
Destination country of the order
27
Customer order code
28
Date on which the order is made
29
Order code
30
Product code generated through the RFID reader
31
Order item discount value
32
Order item discount percentage
33
Order item code
34
Price of products without discount
35
Order Item Profit Ratio

```
36
Number of products per order
37
Value in sales
38
Total amount per order
39
Order Profit Per Order
40  Region of the world where the order is delivered :  Southeast Asia ,South
Asia ,Oceania ,Eastern Asia, West Asia , West of USA , US Center , West Africa,
Central Africa ,North Africa ,Western Europe ,Northern , Caribbean , South
America ,East Africa ,Southern Europe , East of USA ,Canada ,Southern Africa ,
Central Asia ,  Europe , Central America, Eastern Europe , South of  USA
41
State of the region where the order is delivered
42
Order Status : COMPLETE , PENDING , CLOSED , PENDING_PAYMENT ,CANCELED ,
PROCESSING ,SUSPECTED_FRAUD ,ON_HOLD ,PAYMENT_REVIEW
43
Product code
44
Product category code
45
Product Description
46
Link of visit and purchase of the product
47
Product Name
48
Product Price
49
Status of the product stock :If it is 1 not available , 0 the product is
available
50
Exact date and time of shipment
51
The following shipping modes are presented : Standard Class , First Class ,
Second Class , Same Day
```

3. Visão Geral do Dataset

[4]:
```
# Primeiras linhas do dataset
df.head()
```

[4]:
```
        Type  Days for shipping (real)  Days for shipment (scheduled)  \
0      DEBIT                         3                              4
1   TRANSFER                         5                              4
2       CASH                         4                              4
```

```
3     DEBIT                          3                                4
4   PAYMENT                          2                                4

   Benefit per order  Sales per customer   Delivery Status  \
0          91.250000          314.640015  Advance shipping
1        -249.089996          311.359985     Late delivery
2        -247.779999          309.720001  Shipping on time
3          22.860001          304.809998  Advance shipping
4         134.210007          298.250000  Advance shipping

   Late_delivery_risk  Category Id   Category Name Customer City  \
0                   0           73  Sporting Goods        Caguas
1                   1           73  Sporting Goods        Caguas
2                   0           73  Sporting Goods      San Jose
3                   0           73  Sporting Goods   Los Angeles
4                   0           73  Sporting Goods        Caguas

  Customer Country Customer Email Customer Fname  Customer Id Customer Lname  \
0      Puerto Rico      XXXXXXXXX          Cally        20755       Holloway
1      Puerto Rico      XXXXXXXXX          Irene        19492           Luna
2          EE. UU.      XXXXXXXXX        Gillian        19491      Maldonado
3          EE. UU.      XXXXXXXXX           Tana        19490           Tate
4      Puerto Rico      XXXXXXXXX           Orli        19489      Hendricks

  Customer Password Customer Segment Customer State           Customer Street  \
0        XXXXXXXXX         Consumer             PR  5365 Noble Nectar Island
1        XXXXXXXXX         Consumer             PR          2679 Rustic Loop
2        XXXXXXXXX         Consumer             CA      8510 Round Bear Gate
3        XXXXXXXXX      Home Office             CA           3200 Amber Bend
4        XXXXXXXXX        Corporate             PR  8671 Iron Anchor Corners

   Customer Zipcode  Department Id Department Name   Latitude   Longitude  \
0             725.0              2         Fitness  18.251453  -66.037056
1             725.0              2         Fitness  18.279451  -66.037064
2           95125.0              2         Fitness  37.292233 -121.881279
3           90027.0              2         Fitness  34.125946 -118.291016
4             725.0              2         Fitness  18.253769  -66.037048

         Market  Order City Order Country  Order Customer Id  \
0  Pacific Asia      Bekasi     Indonesia              20755
1  Pacific Asia     Bikaner         India              19492
2  Pacific Asia     Bikaner         India              19491
3  Pacific Asia  Townsville     Australia              19490
4  Pacific Asia  Townsville     Australia              19489

  order date (DateOrders)  Order Id  Order Item Cardprod Id  \
0         1/31/2018 22:56     77202                    1360
```

```
1         1/13/2018 12:27     75939                    1360
2         1/13/2018 12:06     75938                    1360
3         1/13/2018 11:45     75937                    1360
4         1/13/2018 11:24     75936                    1360

   Order Item Discount  Order Item Discount Rate  Order Item Id  \
0            13.110000                      0.04         180517
1            16.389999                      0.05         179254
2            18.030001                      0.06         179253
3            22.940001                      0.07         179252
4            29.500000                      0.09         179251

   Order Item Product Price  Order Item Profit Ratio  Order Item Quantity  \
0                    327.75                     0.29                    1
1                    327.75                    -0.80                    1
2                    327.75                    -0.80                    1
3                    327.75                     0.08                    1
4                    327.75                     0.45                    1

    Sales  Order Item Total  Order Profit Per Order    Order Region  \
0  327.75        314.640015               91.250000  Southeast Asia
1  327.75        311.359985             -249.089996      South Asia
2  327.75        309.720001             -247.779999      South Asia
3  327.75        304.809998               22.860001         Oceania
4  327.75        298.250000              134.210007         Oceania

       Order State     Order Status  Order Zipcode  Product Card Id  \
0  Java Occidental         COMPLETE            NaN             1360
1        Rajastán          PENDING            NaN             1360
2        Rajastán           CLOSED            NaN             1360
3       Queensland         COMPLETE            NaN             1360
4       Queensland  PENDING_PAYMENT            NaN             1360

   Product Category Id  Product Description  \
0                   73                  NaN
1                   73                  NaN
2                   73                  NaN
3                   73                  NaN
4                   73                  NaN

                               Product Image  Product Name  Product Price  \
0  http://images.acmesports.sports/Smart+watch   Smart watch         327.75
1  http://images.acmesports.sports/Smart+watch   Smart watch         327.75
2  http://images.acmesports.sports/Smart+watch   Smart watch         327.75
3  http://images.acmesports.sports/Smart+watch   Smart watch         327.75
4  http://images.acmesports.sports/Smart+watch   Smart watch         327.75
```

```
   Product Status shipping date (DateOrders)   Shipping Mode
0               0             2/3/2018 22:56  Standard Class
1               0            1/18/2018 12:27  Standard Class
2               0            1/17/2018 12:06  Standard Class
3               0            1/16/2018 11:45  Standard Class
4               0            1/15/2018 11:24  Standard Class
```

[5]:
```
print(f"O dataframe possui {df.shape[0]} linhas e {df.shape[1]} colunas.")
```

```
O dataframe possui 180519 linhas e 53 colunas.
```

[6]:
```
# Informações sobre todas as colunas do dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180519 entries, 0 to 180518
Data columns (total 53 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   Type                           180519 non-null  object
 1   Days for shipping (real)       180519 non-null  int64
 2   Days for shipment (scheduled)  180519 non-null  int64
 3   Benefit per order              180519 non-null  float64
 4   Sales per customer             180519 non-null  float64
 5   Delivery Status                180519 non-null  object
 6   Late_delivery_risk             180519 non-null  int64
 7   Category Id                    180519 non-null  int64
 8   Category Name                  180519 non-null  object
 9   Customer City                  180519 non-null  object
 10  Customer Country               180519 non-null  object
 11  Customer Email                 180519 non-null  object
 12  Customer Fname                 180519 non-null  object
 13  Customer Id                    180519 non-null  int64
 14  Customer Lname                 180511 non-null  object
 15  Customer Password              180519 non-null  object
 16  Customer Segment               180519 non-null  object
 17  Customer State                 180519 non-null  object
 18  Customer Street                180519 non-null  object
 19  Customer Zipcode               180516 non-null  float64
 20  Department Id                  180519 non-null  int64
 21  Department Name                180519 non-null  object
 22  Latitude                       180519 non-null  float64
 23  Longitude                      180519 non-null  float64
 24  Market                         180519 non-null  object
 25  Order City                     180519 non-null  object
 26  Order Country                  180519 non-null  object
 27  Order Customer Id              180519 non-null  int64
 28  order date (DateOrders)        180519 non-null  object
```

```
 29  Order Id                        180519 non-null  int64
 30  Order Item Cardprod Id          180519 non-null  int64
 31  Order Item Discount             180519 non-null  float64
 32  Order Item Discount Rate        180519 non-null  float64
 33  Order Item Id                   180519 non-null  int64
 34  Order Item Product Price        180519 non-null  float64
 35  Order Item Profit Ratio         180519 non-null  float64
 36  Order Item Quantity             180519 non-null  int64
 37  Sales                           180519 non-null  float64
 38  Order Item Total                180519 non-null  float64
 39  Order Profit Per Order          180519 non-null  float64
 40  Order Region                    180519 non-null  object
 41  Order State                     180519 non-null  object
 42  Order Status                    180519 non-null  object
 43  Order Zipcode                   24840 non-null   float64
 44  Product Card Id                 180519 non-null  int64
 45  Product Category Id             180519 non-null  int64
 46  Product Description             0 non-null       float64
 47  Product Image                   180519 non-null  object
 48  Product Name                    180519 non-null  object
 49  Product Price                   180519 non-null  float64
 50  Product Status                  180519 non-null  int64
 51  shipping date (DateOrders)      180519 non-null  object
 52  Shipping Mode                   180519 non-null  object
dtypes: float64(15), int64(14), object(24)
memory usage: 73.0+ MB
```

[7]:
```
numerics = ["int16", "int32", "int64", "float16", "float32", "float64"]

numericas = df.select_dtypes(include=numerics)

nao_numericas = df.select_dtypes(exclude=numerics)

print(f"Temos {numericas.shape[1]} colunas numéricas e {nao_numericas.shape[1]}␣
 ↪colunas não-numéricas.")
```

```
Temos 29 colunas numéricas e 24 colunas não-numéricas.
```

[8]:
```
# Estátísticas descritivas do dataset
df.describe()
```

[8]:

	Days for shipping (real)	Days for shipment (scheduled)	\
count	180519.000000	180519.000000	
mean	3.497654	2.931847	
std	1.623722	1.374449	
min	0.000000	0.000000	
25%	2.000000	2.000000	
50%	3.000000	4.000000	

75%	5.000000	4.000000
max	6.000000	4.000000

	Benefit per order	Sales per customer	Late_delivery_risk	\
count	180519.000000	180519.000000	180519.000000	
mean	21.974989	183.107609	0.548291	
std	104.433526	120.043670	0.497664	
min	-4274.979980	7.490000	0.000000	
25%	7.000000	104.379997	0.000000	
50%	31.520000	163.990005	1.000000	
75%	64.800003	247.399994	1.000000	
max	911.799988	1939.989990	1.000000	

	Category Id	Customer Id	Customer Zipcode	Department Id	\
count	180519.000000	180519.000000	180516.000000	180519.000000	
mean	31.851451	6691.379495	35921.126914	5.443460	
std	15.640064	4162.918106	37542.461122	1.629246	
min	2.000000	1.000000	603.000000	2.000000	
25%	18.000000	3258.500000	725.000000	4.000000	
50%	29.000000	6457.000000	19380.000000	5.000000	
75%	45.000000	9779.000000	78207.000000	7.000000	
max	76.000000	20757.000000	99205.000000	12.000000	

	Latitude	Longitude	Order Customer Id	Order Id	\
count	180519.000000	180519.000000	180519.000000	180519.000000	
mean	29.719955	-84.915675	6691.379495	36221.894903	
std	9.813646	21.433241	4162.918106	21045.379569	
min	-33.937553	-158.025986	1.000000	1.000000	
25%	18.265432	-98.446312	3258.500000	18057.000000	
50%	33.144863	-76.847908	6457.000000	36140.000000	
75%	39.279617	-66.370583	9779.000000	54144.000000	
max	48.781933	115.263077	20757.000000	77204.000000	

	Order Item Cardprod Id	Order Item Discount	Order Item Discount Rate	\
count	180519.000000	180519.000000	180519.000000	
mean	692.509764	20.664741	0.101668	
std	336.446807	21.800901	0.070415	
min	19.000000	0.000000	0.000000	
25%	403.000000	5.400000	0.040000	
50%	627.000000	14.000000	0.100000	
75%	1004.000000	29.990000	0.160000	
max	1363.000000	500.000000	0.250000	

	Order Item Id	Order Item Product Price	Order Item Profit Ratio	\
count	180519.000000	180519.000000	180519.000000	
mean	90260.000000	141.232550	0.120647	
std	52111.490959	139.732492	0.466796	

```
min         1.000000                  9.990000                -2.750000
25%     45130.500000                 50.000000                 0.080000
50%     90260.000000                 59.990002                 0.270000
75%    135389.500000                199.990005                 0.360000
max    180519.000000               1999.989990                 0.500000

       Order Item Quantity          Sales  Order Item Total  \
count        180519.000000  180519.000000     180519.000000
mean              2.127638     203.772096        183.107609
std               1.453451     132.273077        120.043670
min               1.000000       9.990000          7.490000
25%               1.000000     119.980003        104.379997
50%               1.000000     199.919998        163.990005
75%               3.000000     299.950012        247.399994
max               5.000000    1999.989990       1939.989990

       Order Profit Per Order  Order Zipcode  Product Card Id  \
count           180519.000000   24840.000000    180519.000000
mean                21.974989   55426.132327       692.509764
std                104.433526   31919.279101       336.446807
min              -4274.979980    1040.000000        19.000000
25%                 7.000000   23464.000000       403.000000
50%                31.520000   59405.000000       627.000000
75%                64.800003   90008.000000      1004.000000
max               911.799988   99301.000000      1363.000000

       Product Category Id  Product Description  Product Price  Product Status
count        180519.000000                  0.0  180519.000000        180519.0
mean             31.851451                  NaN     141.232550             0.0
std              15.640064                  NaN     139.732492             0.0
min               2.000000                  NaN       9.990000             0.0
25%              18.000000                  NaN      50.000000             0.0
50%              29.000000                  NaN      59.990002             0.0
75%              45.000000                  NaN     199.990005             0.0
max              76.000000                  NaN    1999.989990             0.0
```

3.1 Detecção de Outliers

Esta etapa consiste em detectar a quantidade de outliers presente em cada uma das variáveis da base de dados. São calculados utilizando Q1 (primeiro quartil), Q3 (terceiro quartil) e IQR (Distância Interquartil):

Q1: Este é o valor que separa os 25% menores valores da coluna. Também é conhecido como primeiro quartil.

Q3: Este é o valor que separa os 25% maiores valores da coluna. Também é conhecido como terceiro quartil.

$$IQR = Q3 - Q1$$

Será considerado Outlier quando o valor da variável (VAR):

$$VAR > Q3 + 1.5IQR$$
$$VAR < Q1 - 1.5IQR$$

[9]:
```
# Função para identificar outliers em uma variável
def detect_outliers(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = column[(column < lower_bound) | (column > upper_bound)]
    return outliers

# Dicionário para armazenar os outliers de cada variável
outliers_dict = {}

# Loop através das variáveis numéricas
for col in df.select_dtypes(include=['float64', 'int64']):
    outliers = detect_outliers(df[col])
    outliers_dict[col] = outliers

# Crie um DataFrame com os outliers
outliers_df = pd.DataFrame(outliers_dict)

# Conte quantos outliers cada variável possui
contagem_outliers = outliers_df.count()

# Exiba a contagem de outliers para cada variável
contagem_outliers_df = pd.DataFrame({'Variavel': contagem_outliers.index,␣
 ↪'Quantidade de Outliers': contagem_outliers.values})
contagem_outliers_df
```

[9]:

	Variavel	Quantidade de Outliers
0	Days for shipping (real)	0
1	Days for shipment (scheduled)	0
2	Benefit per order	18942
3	Sales per customer	1943
4	Late_delivery_risk	0
5	Category Id	0
6	Customer Id	1198
7	Customer Zipcode	0
8	Department Id	362
9	Latitude	9
10	Longitude	1414
11	Order Customer Id	1198

```
12                      Order Id                     0
13         Order Item Cardprod Id                    0
14            Order Item Discount                 7537
15       Order Item Discount Rate                    0
16                  Order Item Id                    0
17       Order Item Product Price                 2048
18        Order Item Profit Ratio                17300
19            Order Item Quantity                    0
20                          Sales                  488
21                Order Item Total                1943
22         Order Profit Per Order                18942
23                  Order Zipcode                    0
24                 Product Card Id                   0
25            Product Category Id                    0
26            Product Description                    0
27                  Product Price                 2048
28                 Product Status                    0
```

3.2 Valores Nulos

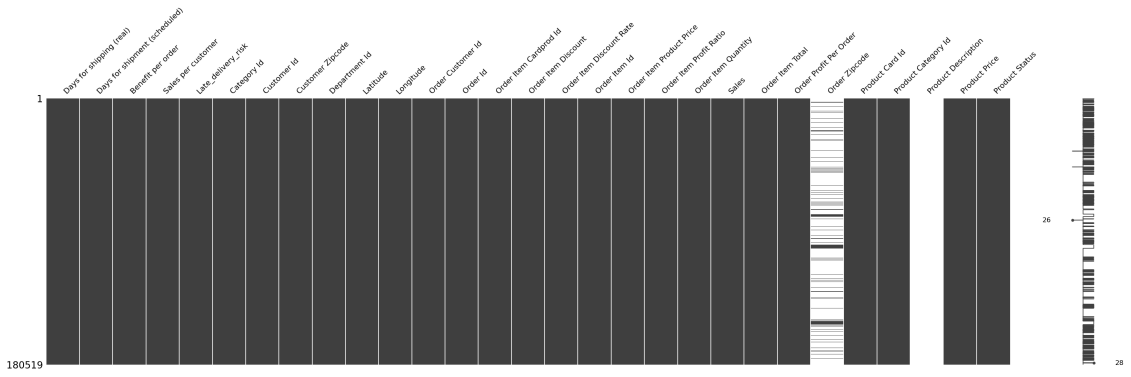
[10]:
```
# Fazendo a soma de valores nulos por coluna
df.isnull().sum()
```

[10]:
```
Type                              0
Days for shipping (real)          0
Days for shipment (scheduled)     0
Benefit per order                 0
Sales per customer                0
Delivery Status                   0
Late_delivery_risk                0
Category Id                       0
Category Name                     0
Customer City                     0
Customer Country                  0
Customer Email                    0
Customer Fname                    0
Customer Id                       0
Customer Lname                    8
Customer Password                 0
Customer Segment                  0
Customer State                    0
Customer Street                   0
Customer Zipcode                  3
Department Id                     0
Department Name                   0
Latitude                          0
Longitude                         0
Market                            0
```

```
Order City                          0
Order Country                       0
Order Customer Id                   0
order date (DateOrders)             0
Order Id                            0
Order Item Cardprod Id              0
Order Item Discount                 0
Order Item Discount Rate            0
Order Item Id                       0
Order Item Product Price            0
Order Item Profit Ratio             0
Order Item Quantity                 0
Sales                               0
Order Item Total                    0
Order Profit Per Order              0
Order Region                        0
Order State                         0
Order Status                        0
Order Zipcode                  155679
Product Card Id                     0
Product Category Id                 0
Product Description            180519
Product Image                       0
Product Name                        0
Product Price                       0
Product Status                      0
shipping date (DateOrders)          0
Shipping Mode                       0
dtype: int64
```

[11]:
```
# visualizando as colunas numéricas com dados faltantes
missingno.matrix(numericas,figsize=(40,10))
```

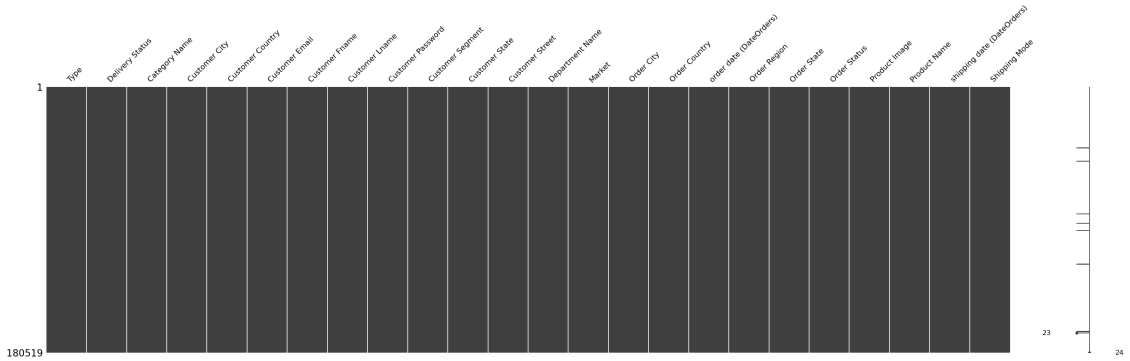
[11]: <Axes: >



Pela imagem acima, verifica-se que há duas colunas numéricas com grande quantidade de valores nulos: **Order Zipcode** e **Product Description**. pela contagem anterior, sabe-se que uma terceira coluna **(Customer Zipcode)** possui apenas 3 valores nulos, o que não foi possível verificar visualmente devido à baixa quantidade.

[12]:
```
# visualizando as colunas não numéricas com dados faltantes
missingno.matrix(nao_numericas,figsize=(40,10))
```

[12]: <Axes: >



Pela imagem acima, não é possível identificar visualmente alguma variável não numérica com valores nulos. Porém, sabe se a variável **Customer Lname** possui 8 valores nulos em sua composição, não sendo possível verificar visualmente por ser uma quantidade pequena perante o número de registros da base.

[13]:
```
# fazendo unpack de linhas e colunas
rows, columns = df.shape

# Percentual de dados faltantes por coluna
percentual_nan = ((df.isnull().sum()/rows) * 100).round(2)
percentual_nan
```

[13]:
```
Type                           0.00
Days for shipping (real)       0.00
Days for shipment (scheduled)  0.00
Benefit per order              0.00
Sales per customer             0.00
Delivery Status                0.00
Late_delivery_risk             0.00
Category Id                    0.00
Category Name                  0.00
Customer City                  0.00
Customer Country               0.00
Customer Email                 0.00
Customer Fname                 0.00
```

```
Customer Id                       0.00
Customer Lname                    0.00
Customer Password                 0.00
Customer Segment                  0.00
Customer State                    0.00
Customer Street                   0.00
Customer Zipcode                  0.00
Department Id                     0.00
Department Name                   0.00
Latitude                          0.00
Longitude                         0.00
Market                            0.00
Order City                        0.00
Order Country                     0.00
Order Customer Id                 0.00
order date (DateOrders)           0.00
Order Id                          0.00
Order Item Cardprod Id            0.00
Order Item Discount               0.00
Order Item Discount Rate          0.00
Order Item Id                     0.00
Order Item Product Price          0.00
Order Item Profit Ratio           0.00
Order Item Quantity               0.00
Sales                             0.00
Order Item Total                  0.00
Order Profit Per Order            0.00
Order Region                      0.00
Order State                       0.00
Order Status                      0.00
Order Zipcode                    86.24
Product Card Id                   0.00
Product Category Id               0.00
Product Description             100.00
Product Image                     0.00
Product Name                      0.00
Product Price                     0.00
Product Status                    0.00
shipping date (DateOrders)        0.00
Shipping Mode                     0.00
dtype: float64
```

3.3 Linhas Duplicadas

[14]:
```
# verificando se há linhas duplicadas
df[df.duplicated()]
```

[14]:
```
Empty DataFrame
Columns: [Type, Days for shipping (real), Days for shipment (scheduled), Benefit
per order, Sales per customer, Delivery Status, Late_delivery_risk, Category Id,
Category Name, Customer City, Customer Country, Customer Email, Customer Fname,
Customer Id, Customer Lname, Customer Password, Customer Segment, Customer
State, Customer Street, Customer Zipcode, Department Id, Department Name,
Latitude, Longitude, Market, Order City, Order Country, Order Customer Id, order
date (DateOrders), Order Id, Order Item Cardprod Id, Order Item Discount, Order
Item Discount Rate, Order Item Id, Order Item Product Price, Order Item Profit
Ratio, Order Item Quantity, Sales, Order Item Total, Order Profit Per Order,
Order Region, Order State, Order Status, Order Zipcode, Product Card Id, Product
Category Id, Product Description, Product Image, Product Name, Product Price,
Product Status, shipping date (DateOrders), Shipping Mode]
Index: []
```

Nota-se que o dataset contem 180.519 entradas com informações de vendas de produtos da empresa SupplyAll e que todos os registros são únicos. Desta forma, é possível concluir que não existem entradas duplicadas.

3.4 Valores Diferentes por Coluna

[15]:
```
# Definindo quantos valores diferentes existem em cada variável

contagem = pd.DataFrame(columns=['Variavel','Contagens_Distintas'])

for coluna in df.columns:
    dados = pd.DataFrame({'Variavel': [coluna], 'Contagens_Distintas':␣
 ↪[df[coluna].value_counts().shape[0]]})
    contagem = pd.concat([contagem, dados], ignore_index = True)

contagem
```

[15]:

	Variavel	Contagens_Distintas
0	Type	4
1	Days for shipping (real)	7
2	Days for shipment (scheduled)	4
3	Benefit per order	21998
4	Sales per customer	2927
5	Delivery Status	4
6	Late_delivery_risk	2
7	Category Id	51
8	Category Name	50
9	Customer City	563
10	Customer Country	2
11	Customer Email	1
12	Customer Fname	782
13	Customer Id	20652
14	Customer Lname	1109

15	Customer Password	1
16	Customer Segment	3
17	Customer State	46
18	Customer Street	7458
19	Customer Zipcode	995
20	Department Id	11
21	Department Name	11
22	Latitude	11250
23	Longitude	4487
24	Market	5
25	Order City	3597
26	Order Country	164
27	Order Customer Id	20652
28	order date (DateOrders)	65752
29	Order Id	65752
30	Order Item Cardprod Id	118
31	Order Item Discount	1017
32	Order Item Discount Rate	18
33	Order Item Id	180519
34	Order Item Product Price	75
35	Order Item Profit Ratio	162
36	Order Item Quantity	5
37	Sales	193
38	Order Item Total	2927
39	Order Profit Per Order	21998
40	Order Region	23
41	Order State	1089
42	Order Status	9
43	Order Zipcode	609
44	Product Card Id	118
45	Product Category Id	51
46	Product Description	0
47	Product Image	118
48	Product Name	118
49	Product Price	75
50	Product Status	1
51	shipping date (DateOrders)	63701
52	Shipping Mode	4

Com a informação das contagens distintas de cada variável, podemos obter as seguintes informações: * Os clientes pagaram os produtos através de 4 meios de pagamentos diferentes
* Há 9 status diferentes para os pedidos * Há 4 status diferentes para as entregas dos produtos * Os produtos são entregues para 5 grandes regiões diferentes do mundo * Os produtos foram enviados para 164 países diferentes * Os clientes são de 2 países diferentes * Há 51 categorias de produtos diferentes * Há 3 segmentos distintos de clientes * Há 20.652 clientes no total * Há 23 micro regiões do mundo onde os produtos são entregues

[16]:
```
# Desconsiderar pedidos que tiveram entrega cancelada. Está sendo levado em␣
 ↪consideração que entregas canceladas têm o dinheiro
# da compra enviado de volta ao cliente, não contribuindo para os cofres da␣
 ↪companhia
df_filtrado = df[df['Delivery Status']!='Shipping canceled']
```

4. Avaliação dos clientes

[17]:
```
# Transformando a variável 'order date (DateOrders)' em datetime para trabalhar␣
 ↪com datas
df['order date (DateOrders)'] = pd.to_datetime(df['order date (DateOrders)'])

# Extraindo o ano
df['Ano'] = df['order date (DateOrders)'].dt.year

# Agrupando por Customer_Id e Year, e contando os Order_Id únicos
compras_por_cliente_ano = df.groupby(['Customer Id', 'Ano'])['Order Id'].
 ↪nunique().reset_index()

# Renomeando a coluna para refletir a contagem
compras_por_cliente_ano.rename(columns={'Order Id': 'Quantidade de Pedidos'},␣
 ↪inplace=True)

compras_por_cliente_ano_pivot = compras_por_cliente_ano.
 ↪pivot_table(index="Customer Id",columns="Ano",values="Quantidade de␣
 ↪Pedidos",fill_value=0)

compras_por_cliente_ano_pivot
```

[17]:

Ano	2015	2016	2017	2018
Customer Id				
1	1.0	0.0	0.0	0.0
2	1.0	1.0	2.0	0.0
3	1.0	1.0	3.0	0.0
4	2.0	1.0	1.0	0.0
5	0.0	3.0	0.0	0.0
…	…	…	…	…
20753	0.0	0.0	0.0	1.0
20754	0.0	0.0	0.0	1.0
20755	0.0	0.0	0.0	1.0
20756	0.0	0.0	0.0	1.0
20757	0.0	0.0	0.0	1.0

[20652 rows x 4 columns]

[18]:
```
# Filtrando todos os clientes que compraram em 2018
```

```
compras_por_cliente_ano_pivot[compras_por_cliente_ano_pivot[2018] > 0].
 ↪value_counts()
```

[18]:
```
2015  2016  2017  2018
0.0   0.0   0.0   1.0     2123
Name: count, dtype: int64
```

Chega-se a conclusão que, todos os clientes que compraram em 2018, compraram apenas em 2018!

[19]:
```
# Filtrando para trazer somente a informação do ano de 2018
compras_2018 = compras_por_cliente_ano[compras_por_cliente_ano['Ano'] == 2018]

# Verificando se há clientes que compraram mais de uma vez em 2018
clientes_compras_multiplos_2018 = compras_2018[compras_2018['Quantidade de␣
 ↪Pedidos'] > 1]
clientes_compras_multiplos_2018
```

[19]:
```
Empty DataFrame
Columns: [Customer Id, Ano, Quantidade de Pedidos]
Index: []
```

Observa-se que, todos os clientes que compraram em 2018, compraram apenas em 2018 e apenas uma vez! Para verificar se este comportamento atípico começa antes de 2018, será analisado também o ano de 2017.

[20]:
```
# Filtrando para o ano de 2017
compras_2017 = compras_por_cliente_ano[compras_por_cliente_ano['Ano'] == 2017]

# Certificando-se de que os IDs estão em ordem crescente
compras_2017 = compras_2017.sort_values(by='Customer Id')

# Encontrar a partir de qual Customer_Id todos os seguintes compraram apenas␣
 ↪uma vez
starting_customer_id = None

for i in range(len(compras_2017)):
    current_id = compras_2017.iloc[i]['Customer Id']
    current_purchases = compras_2017.iloc[i]['Quantidade de Pedidos']

    if current_purchases != 1:
        continue  # Pula se o cliente atual fez mais de uma compra

    # Verifica se todos os IDs seguintes estão em sequência e têm apenas uma␣
 ↪compra
    if all((compras_2017.iloc[j]['Customer Id'] == current_id + j - i) and
           (compras_2017.iloc[j]['Quantidade de Pedidos'] == 1) for j in␣
 ↪range(i, len(compras_2017))):
        starting_customer_id = current_id
```

```
        break

if starting_customer_id is not None:
    print(f"Todos os Customer_Id a partir de {starting_customer_id} fizeram␣
 ↪apenas uma compra em 2017.")
else:
    print("Não há uma sequência contínua de clientes com apenas uma compra.")
```

```
Todos os Customer_Id a partir de 12440 fizeram apenas uma compra em 2017.
```

Sendo assim, entre todos os Custumer_Id a partir de 12440, foi verificado a partir de que data eles começaram a comprar.

[21]:
```
df['order date (DateOrders)'] = pd.to_datetime(df['order date (DateOrders)'])

# Filtrando 'Customer Id' a partir de 12440
df_customer_filtrado = df[df['Customer Id'] >= 12440]

# Obtendo a data mínima
data_minima = df_customer_filtrado['order date (DateOrders)'].min()

print("Data mínima para Customer Id a partir de 12440:", data_minima)
```

```
Data mínima para Customer Id a partir de 12440: 2017-10-02 13:50:00
```

[22]:
```
df[df['Customer Id'] == 12440]
```

[22]:
```
        Type  Days for shipping (real)  Days for shipment (scheduled)  \
24764  DEBIT                         2                              1

       Benefit per order  Sales per customer Delivery Status  \
24764               1.98               26.42   Late delivery

       Late_delivery_risk  Category Id Category Name Customer City  \
24764                   1           59        Books     San Marcos

      Customer Country Customer Email Customer Fname  Customer Id  \
24764          EE. UU.      XXXXXXXXX         Joanha        12440

      Customer Lname Customer Password Customer Segment Customer State  \
24764     Mirkckociv         XXXXXXXXX        Corporate             CA

           Customer Street  Customer Zipcode  Department Id Department Name  \
24764  8324 Little Common            92069.0              8       Book Shop

        Latitude   Longitude  Market Order City Order Country  \
24764  33.146751 -117.169533  Europe      Forst      Alemania

       Order Customer Id order date (DateOrders)  Order Id  \
```

```
24764              12440     2017-10-02 13:50:00     68887

       Order Item Cardprod Id  Order Item Discount  Order Item Discount Rate  \
24764                    1346                 4.66                      0.15

       Order Item Id  Order Item Product Price  Order Item Profit Ratio  \
24764         172202                    31.08                     0.08

       Order Item Quantity  Sales  Order Item Total  Order Profit Per Order  \
24764                    1  31.08             26.42                    1.98

         Order Region   Order State Order Status  Order Zipcode  \
24764  Western Europe  Brandenburgo     COMPLETE            NaN

       Product Card Id  Product Category Id  Product Description  \
24764             1346                   59                  NaN

                                    Product Image   Product Name  \
24764  http://images.acmesports.sports/Sports+Books  Sports Books

       Product Price  Product Status shipping date (DateOrders) Shipping Mode  \
24764          31.08               0            10/4/2017 13:50   First Class

        Ano
24764  2017
```

[23]:
```
df[(df['Customer Id'] < 12440) & (df['order date (DateOrders)'] >= data_minima)]
```

[23]:
```
Empty DataFrame
Columns: [Type, Days for shipping (real), Days for shipment (scheduled), Benefit
per order, Sales per customer, Delivery Status, Late_delivery_risk, Category Id,
Category Name, Customer City, Customer Country, Customer Email, Customer Fname,
Customer Id, Customer Lname, Customer Password, Customer Segment, Customer
State, Customer Street, Customer Zipcode, Department Id, Department Name,
Latitude, Longitude, Market, Order City, Order Country, Order Customer Id, order
date (DateOrders), Order Id, Order Item Cardprod Id, Order Item Discount, Order
Item Discount Rate, Order Item Id, Order Item Product Price, Order Item Profit
Ratio, Order Item Quantity, Sales, Order Item Total, Order Profit Per Order,
Order Region, Order State, Order Status, Order Zipcode, Product Card Id, Product
Category Id, Product Description, Product Image, Product Name, Product Price,
Product Status, shipping date (DateOrders), Shipping Mode, Ano]
Index: []
```

Conclusão: A partir de **2017-10-02**, **todos os clientes são novos e compraram apenas uma vez**. Sendo assim, para não trabalhar com uma base de dados com uma **possível interferência (erro operacional)**, o conjunto de dados considerará dos dados **até setembro/2017**.

[24]:
```
# Atualizando os dataframes df_filtrado e df com o intervalo de datas correto,␣
 ↪para seguir com as análises
df = pd.read_csv('DataCoSupplyChainDataset.csv', encoding='ISO-8859-1')

# Transformando a coluna de data para o formato data e hora
df['order date (DateOrders)'] = pd.to_datetime(df['order date (DateOrders)'])

# Definindo a data limite para ser considerada na análise
data_limite = pd.Timestamp('2017-09-30 23:59:59')

# Filtrando o dataframe df para conter datas menores ou iguais a que 2017-09-30
df = df[df['order date (DateOrders)'] <= data_limite]

# Desconsiderar pedidos que tiveram entrega cancelada
df_filtrado = df[df['Delivery Status']!='Shipping canceled']
```

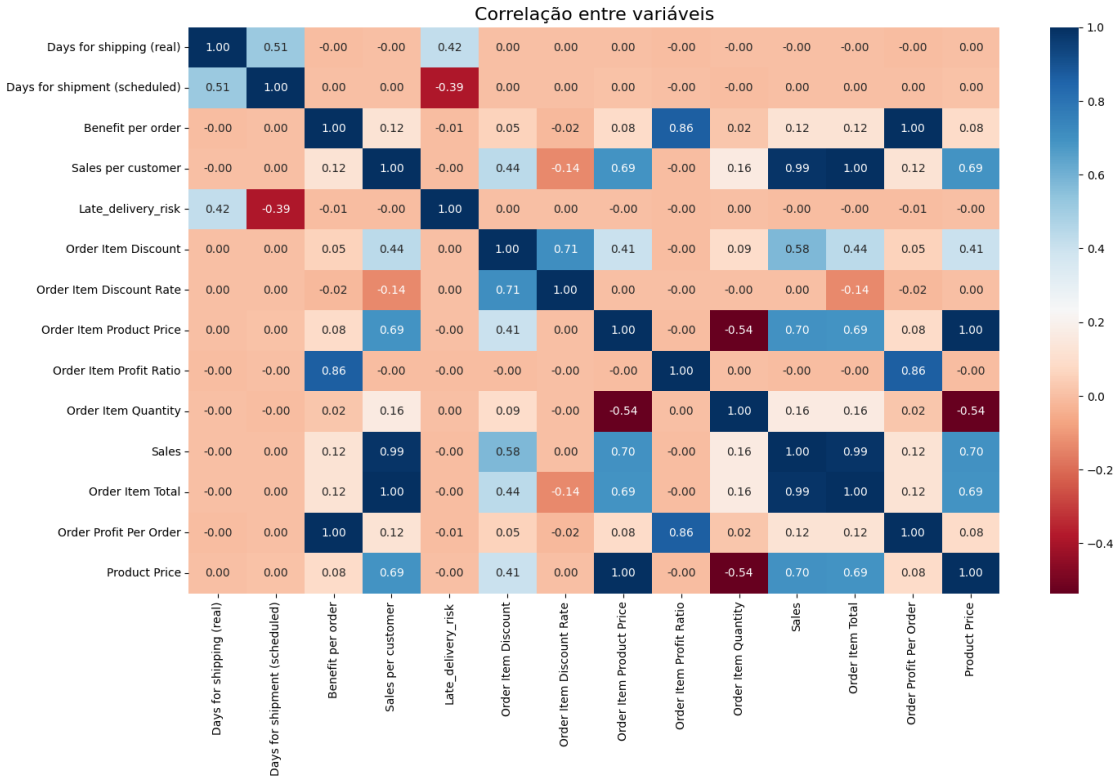
5. Relações entre variáveis

[25]:
```
numericas = df_filtrado.select_dtypes(include=numerics)

# Remover colunas específicas
numericas_filtradas = numericas.drop(['Category Id', 'Customer Id', 'Customer␣
 ↪Zipcode', 'Department Id', 'Latitude',
                              'Longitude', 'Order Customer Id', 'Order Id',␣
 ↪'Order Item Cardprod Id', 'Order Item Id',
                              'Order Zipcode', 'Product Card Id', 'Product␣
 ↪Category Id', 'Product Description',
                              'Product Status'], axis=1)

# Calcular a matriz de correlação
corr_matrix = numericas_filtradas.corr()

# Criar o heatmap
plt.figure(figsize=(16, 9))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='RdBu')
plt.title('Correlação entre variáveis', fontsize=16)
plt.show()
```



1.0.1 Variáveis altamente correlacionadas

- Benefit per order e Order Profit Per Order **(correlação 1.00)**
- Sales per customer e Sales **(correlação 0.99)**
- Sales per Customer e Order Item Total **(correlação 1.00)**
- Order Item Product Price e Product Price **(correlação 1.00)**

Obs: É importante lembrar que a correlação não implica causalidade. Mesmo quando duas variáveis estão correlacionadas, isso não significa necessariamente que uma causa a outra. Outros fatores ou variáveis não incluídas na análise podem estar contribuindo para a relação observada. Portanto, a interpretação da correlação deve ser feita com cuidado e, quando necessário, devem ser realizadas análises adicionais para entender melhor a relação entre as variáveis.

6. Avaliação de vendas e receitas

1.1 6.1 Análise temporal das vendas - quantidade

[26]:
```
# Calcula a quantidade de vendas por mes
vendas_por_mes = df_filtrado.resample('M', on='order date (DateOrders)').size()

# Criando um dataframe a partir da série
vendas_por_mes = vendas_por_mes.reset_index()
```

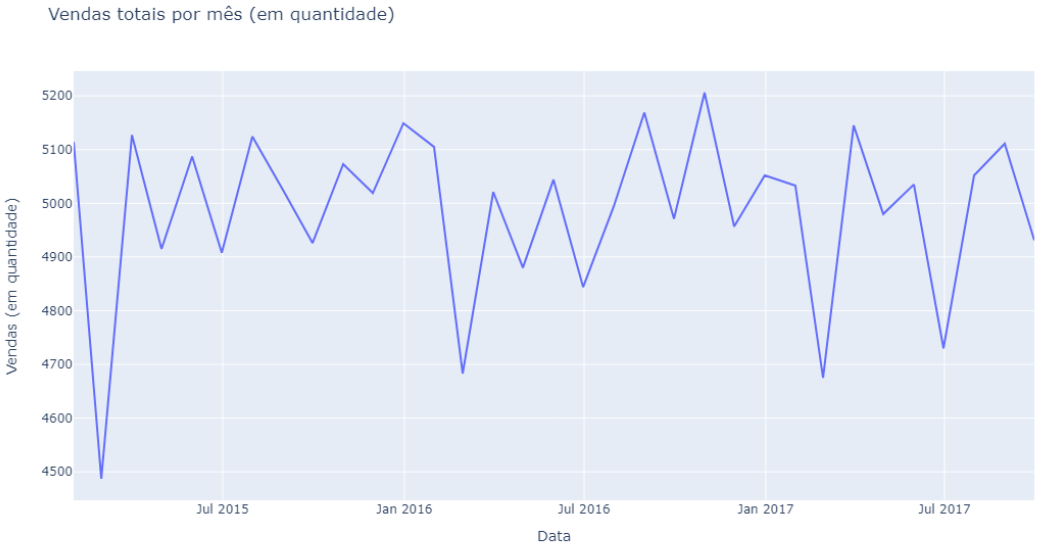
```
vendas_por_mes.rename(columns={0: 'Vendas (em quantidade)'}, inplace=True)

# Calculando a variação das vendas
vendas_por_mes['Variação de Vendas'] = vendas_por_mes['Vendas (em quantidade)'].
 ↪diff()

# Calculando a variação percentual das vendas
vendas_por_mes['Variação Percentual'] = vendas_por_mes['Vendas (em␣
 ↪quantidade)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
vendas_por_mes['Variação Percentual'] = (vendas_por_mes['Variação Percentual']␣
 ↪* 100).round(2)

# Visualização
fig = px.line(vendas_por_mes, x='order date (DateOrders)', y='Vendas (em␣
 ↪quantidade)',
              title='Vendas totais por mês (em quantidade)', labels={'order␣
 ↪date (DateOrders)': 'Data'})
fig.update_layout(width=1000, height=600)
#fig.write_image("fig1.png")
fig.show()
```



[ ]:
```
vendas_por_mes['Vendas (em quantidade)'].describe()
```

[ ]:
```
# Determinando outliers

Q1 = vendas_por_mes['Vendas (em quantidade)'].describe()[4]
Q3 = vendas_por_mes['Vendas (em quantidade)'].describe()[6]
IQR = Q3 - Q1

Limite_superior = Q3 + 1.5*IQR
Limite_inferior = Q1 - 1.5*IQR

print(f'Quantidade de vendas mensais maiores que {Limite_superior} são outliers␣
 ↪superiores')
print(f'Quantidade de vendas mensais menores que {Limite_inferior} são outliers␣
 ↪inferiores')
```

Ou seja, há apenas 1 outlier (inferior), em fevereiro/2015, quando foram vendidas 4487 itens.

[ ]:
```
# Visualizando a distribuição da quantidade de vendas ao mês
plt.figure(figsize=(16, 9))
sns.histplot(vendas_por_mes['Vendas (em quantidade)'], bins=50, kde=True)
plt.title('Distribuição da quantidade de vendas')
plt.xlabel('Quantidade')
plt.ylabel('Frequência')
plt.show()
```

Em todo o período avaliado, a média de vendas ao mês está em torno de 4987, bem próxima da mediana que está em 5025. Ou seja, A quantidade de vendas está bem concentrada em torno da média, com um baixo desvio padrão. Como a média é maior que a mediana, trata-se de uma distribuição assimétrica à esquerda.

[ ]:
```
print(f"A assimetria da distribuição de quantidades assume valor:␣
 ↪{skew(vendas_por_mes['Vendas (em quantidade)']):.2f}")
```

[ ]:
```
# Visualização
fig = px.line(vendas_por_mes, x='order date (DateOrders)', y='Variação de␣
 ↪Vendas',
              title='Variação das vendas mensais - em quantidade',␣
 ↪labels={'order date (DateOrders)': 'Data'})
fig.update_layout(width=1000, height=600)
fig.show()
```

Pelos gráficos acima, percebe-se que o mês de fevereiro é o pior mês para vendas, com as maiores variações negativas.

[ ]:
```
# Agrupando pelo dia da semana e contando as vendas
vendas_dia_da_semana = df_filtrado.groupby(df_filtrado['order date␣
 ↪(DateOrders)'].dt.day_name()).size()

# Ordenar os resultados pelos dias da semana
```

```
ordem_semana = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday']
vendas_dia_da_semana = vendas_dia_da_semana.reindex(ordem_semana)

vendas_dia_da_semana = vendas_dia_da_semana.reset_index()
vendas_dia_da_semana.rename(columns={0:'Quantidade de Vendas', 'order date␣
 ↪(DateOrders)':'Dia da Semana'}, inplace=True)

vendas_dia_da_semana
```

[ ]:
```
fig = px.bar(vendas_dia_da_semana, x='Dia da Semana', y='Quantidade de Vendas',␣
 ↪color = 'Dia da Semana',
             text = 'Quantidade de Vendas', title='Vendas por dia da semana -␣
 ↪em quantidade')

fig.update_traces(textposition='outside', texttemplate='%{text}',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

Percebe-se que as quantidade de vendas por dia da semana assumem valores gerais bem próximos uns dos outros, não tendo um dia de preferencial de compras dos clientes.

6.2 Análise temporal das vendas - financeiro

[ ]:
```
# Agrupar os dados por ano e mês e somar as vendas
sales_over_time = df_filtrado.resample('M', on='order date (DateOrders)').
 ↪sum()['Sales']

# Criando um DataFrame a partir da série
sales_over_time_df = sales_over_time.reset_index()

# Calculando a variação das vendas
sales_over_time_df['Variação de Vendas'] = sales_over_time_df['Sales'].diff()

# Calculando a variação percentual das vendas
sales_over_time_df['Variação Percentual'] = sales_over_time_df['Sales'].
 ↪pct_change()

# Convertendo a variação para porcentagem e arredondando
sales_over_time_df['Variação Percentual'] = (sales_over_time_df['Variação␣
 ↪Percentual'] * 100).round(2)


# Visualização
fig = px.line(sales_over_time_df, x='order date (DateOrders)', y='Sales',
```

```
                title='Vendas Totais por Mês (em milhões)', labels={'order date␣
 ↪(DateOrders)': 'Data', 'Sales': 'Vendas Totais (em milhões)'})
fig.update_layout(width=1000, height=600)
fig.show()
```

[ ]:
```
sales_over_time.describe()
```

As vendas variam em torno de uma média de 993 mil, terminando setembro/2017 na maior alta histórica, cerca de 1,08MM entre todos os produtos comercializados.

[ ]:
```
# Visualização
fig = px.line(sales_over_time_df, x='order date (DateOrders)', y='Variação de␣
 ↪Vendas',
                title='Variação das vendas Totais por Mês', labels={'order date␣
 ↪(DateOrders)': 'Data', 'Variação de Vendas':'Variação de Vendas (em␣
 ↪milhares)'})
fig.update_layout(width=1000, height=600)
fig.show()
```

[ ]:
```
sales_over_time_df[abs(sales_over_time_df['Variação Percentual']) >= 5]
```

As vendas oscilaram durante todo o período.

1.1.1 Destaque para os meses de variação positiva nas vendas:

- **Março/2015**: 14,30%,

- **Março/2016**: 6,41%
- **Março/2017**: 6,31%

1.1.2 Destaque para os meses de variação negativa nas vendas:

- **Fevereiro/2015**: -12,61%,

- **Fevereiro/2016**: -7,88%
- **Junho/2017**: -5,85%

1.1.3 Vendas por segmento de cliente

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
Venda_por_segmento = df_filtrado.groupby('Customer Segment')['Sales'].sum().
 ↪reset_index()

Venda_por_segmento['Sales'] = (Venda_por_segmento['Sales'] / 1000000).round(2)

Venda_por_segmento.rename(columns={'Sales': 'Venda (em milhões)', 'Customer␣
 ↪Segment': 'Segmento de Cliente'}, inplace=True)

# Ordenando os resultados
```

```
Venda_por_segmento = Venda_por_segmento.sort_values(by='Venda (em milhões)',␣
 ↪ascending=False).reset_index(drop=True)

Venda_por_segmento
```

[ ]:
```
fig = px.bar(Venda_por_segmento, x='Segmento de Cliente', y='Venda (em␣
 ↪milhões)', color = 'Segmento de Cliente',
             text = 'Venda (em milhões)', title='Venda por segmento de cliente␣
 ↪(em milhões)')

fig.update_traces(textposition='outside', texttemplate='%{text}MM',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=500)
fig.show()
```

[ ]:
```
# Agora, agrupar os dados por categoria e data, somando o lucro
Venda_por_segmento_e_data = df_filtrado.groupby(['Customer Segment', pd.
 ↪Grouper(key='order date (DateOrders)', freq='M')])['Sales'].sum().
 ↪reset_index()

Venda_por_segmento_e_data['Sales'] = (Venda_por_segmento_e_data['Sales'] /␣
 ↪1000).round(2)
Venda_por_segmento_e_data.rename(columns={'Customer Segment':'Segmento',␣
 ↪'Sales': 'Vendas (em milhares)', 'order date (DateOrders)': 'Data'},␣
 ↪inplace=True)

# Calculando a variação das vendas por segmento
Venda_por_segmento_e_data['Variação de Vendas'] = Venda_por_segmento_e_data.
 ↪groupby('Segmento')['Vendas (em milhares)'].diff()

# Calculando a variação percentual das vendas por segmento
Venda_por_segmento_e_data['Variação Percentual'] = Venda_por_segmento_e_data.
 ↪groupby('Segmento')['Vendas (em milhares)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
Venda_por_segmento_e_data['Variação Percentual'] =␣
 ↪(Venda_por_segmento_e_data['Variação Percentual'] * 100).round(2)

# Criar o gráfico usando Plotly
fig = px.line(Venda_por_segmento_e_data, x='Data', y='Vendas (em milhares)',␣
 ↪color='Segmento', title='Venda por segmento de cliente ao longo do tempo (em␣
 ↪milhares)')

fig.update_layout(width=1000, height=500)
# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Venda_por_segmento_e_data, x='Data', y='Variação de Vendas',␣
 ↪color='Segmento', title='Variação de venda por segmento de cliente ao longo␣
 ↪do tempo')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando os segmentos e os meses que tiveram uma variação maior que 10% nas␣
 ↪vendas
Venda_por_segmento_e_data[abs(Venda_por_segmento_e_data['Variação Percentual'])␣
 ↪>= 10]
```

As vendas por seguimento oscilaram durante todo o período.

1.1.4 Destaque para os meses de variação positiva nas vendas:

- **Corporate - Março/2015**: 18,49%,

- **Home Office - Março/2017**: 16,61%
- **Corporate - Março/2017**: 18,01%

1.1.5 Destaque para os meses de variação negativa nas vendas:

- **Home Office - Abril/2016**: -16,74%
- **Corporate - Fevereiro/2016**: -15,61%
- **Consumer - Fevereiro/2015**: -13,97%

1.1.6 Categorias com mais vendas

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
vendas_por_categoria = df_filtrado.groupby('Category Name')['Sales'].sum().
 ↪reset_index()

vendas_por_categoria['Sales'] = (vendas_por_categoria['Sales'] / 1000000).
 ↪round(3)

vendas_por_categoria.rename(columns={'Sales': 'Vendas (em milhões)', 'Category␣
 ↪Name': 'Nome da Categoria'}, inplace=True)

# Ordenando os resultados
vendas_por_categoria = vendas_por_categoria.sort_values(by='Vendas (em␣
 ↪milhões)', ascending=False).reset_index(drop=True)

#sales_by_category_sorted = sales_by_category_sorted.head(10)

vendas_por_categoria
```

[ ]:
```
fig = px.bar(vendas_por_categoria.head(10), x='Nome da Categoria', y='Vendas␣
 ↪(em milhões)', color = 'Nome da Categoria',
             text = 'Vendas (em milhões)', title='Top 10 categorias com mais␣
 ↪vendas (em milhões)')

fig.update_traces(textposition='outside', texttemplate='%{text}MM',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

[ ]:
```
# Agora, agrupar os dados por categoria e data, somando o lucro
Vendas_por_categoria_e_data = df_filtrado.groupby(['Category Name', pd.
 ↪Grouper(key='order date (DateOrders)', freq='M')])['Sales'].sum().
 ↪reset_index()

Vendas_por_categoria_e_data['Sales'] = (Vendas_por_categoria_e_data['Sales'] /␣
 ↪1000).round(2)
Vendas_por_categoria_e_data.rename(columns={'Category Name':'Categoria',␣
 ↪'Sales': 'Vendas (em milhares)', 'order date (DateOrders)': 'Data'},␣
 ↪inplace=True)

# Calculando a variação das vendas por categoria
Vendas_por_categoria_e_data['Variação de Vendas'] = Vendas_por_categoria_e_data.
 ↪groupby('Categoria')['Vendas (em milhares)'].diff()

# Calculando a variação percentual das vendas por categoria
Vendas_por_categoria_e_data['Variação Percentual'] =␣
 ↪Vendas_por_categoria_e_data.groupby('Categoria')['Vendas (em milhares)'].
 ↪pct_change()

# Convertendo a variação para porcentagem e arredondando
Vendas_por_categoria_e_data['Variação Percentual'] =␣
 ↪(Vendas_por_categoria_e_data['Variação Percentual'] * 100).round(2)

# Criar o gráfico usando Plotly
fig = px.line(Vendas_por_categoria_e_data, x='Data', y='Vendas (em milhares)',␣
 ↪color='Categoria', title='Vendas por categoria ao longo do tempo')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Vendas_por_categoria_e_data, x='Data', y='Variação de Vendas',␣
 ↪color='Categoria', title='Variação das vendas por categoria ao longo do␣
 ↪tempo (em milhares)')
```

```
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando as categorias e os meses que tiveram uma variação maior que 500% nas vendas
Vendas_por_categoria_e_data[abs(Vendas_por_categoria_e_data['Variação Percentual']) > 500.00]
```

As vendas por categoria oscilaram durante todo o período.

1.1.7 Destaque para os meses de variação positiva nas vendas:

- **Fitness Accessories - Setembro/2017**: 1400%,

- **Hockey - Setembro/2017**: 1189,06%
- **Strength Training - Setembro/2017**: 539,04%

1.1.8 Vendas por departamento

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
Vendas_por_depart = df_filtrado.groupby('Department Name')['Sales'].sum().reset_index()

Vendas_por_depart['Sales'] = (Vendas_por_depart['Sales'] / 1000000).round(2)

Vendas_por_depart.rename(columns={'Sales': 'Vendas (em milhões)', 'Department Name': 'Departamento'}, inplace=True)

# Ordenando os resultados
Vendas_por_depart = Vendas_por_depart.sort_values(by='Vendas (em milhões)', ascending=False).reset_index(drop=True)

Vendas_por_depart
```

[ ]:
```
fig = px.bar(Vendas_por_depart, x='Departamento', y='Vendas (em milhões)', color = 'Departamento',
             text = 'Vendas (em milhões)', title='Vendas por departamento (em milhões)')

fig.update_traces(textposition='outside', texttemplate='%{text}MM', textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

[ ]:
```
# Agora, agrupar os dados por categoria e data, somando o lucro
Vendas_por_depart_e_data = df_filtrado.groupby(['Department Name', pd.
 ↪Grouper(key='order date (DateOrders)', freq='M')])['Sales'].sum().
 ↪reset_index()

Vendas_por_depart_e_data['Sales'] = (Vendas_por_depart_e_data['Sales'] / 1000).
 ↪round(2)
Vendas_por_depart_e_data.rename(columns={'Department Name':'Departamento',␣
 ↪'Sales': 'Vendas (em milhares)', 'order date (DateOrders)': 'Data'},␣
 ↪inplace=True)

# Calculando a variação das vendas por departamento
Vendas_por_depart_e_data['Variação de Vendas'] = Vendas_por_depart_e_data.
 ↪groupby('Departamento')['Vendas (em milhares)'].diff()

# Calculando a variação percentual das vendas por departamento
Vendas_por_depart_e_data['Variação Percentual'] = Vendas_por_depart_e_data.
 ↪groupby('Departamento')['Vendas (em milhares)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
Vendas_por_depart_e_data['Variação Percentual'] =␣
 ↪(Vendas_por_depart_e_data['Variação Percentual'] * 100).round(2)

# Criar o gráfico usando Plotly
fig = px.line(Vendas_por_depart_e_data, x='Data', y='Vendas (em milhares)',␣
 ↪color='Departamento', title='Vendas por departamento ao Longo do Tempo (em␣
 ↪milhares)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Vendas_por_depart_e_data, x='Data', y='Variação de Vendas',␣
 ↪color='Departamento', title='Variação das vendas por departamento ao Longo␣
 ↪do Tempo (em milhares)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando os departamentos e os meses que tiveram uma variação maior que 50%␣
 ↪nas vendas
Vendas_por_depart_e_data[abs(Vendas_por_depart_e_data['Variação Percentual']) >␣
 ↪50.00]
```

As vendas por departamento oscilaram durante todo o período.

1.1.9 Destaque para os meses de variação positiva nas vendas:

- **Fitness - Setembro/2017**: 176,89%,
- **Outdoors - Maio/2017**: 111,84%

1.1.10 Vendas por Mercado Global

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
vendas_por_market = df_filtrado.groupby('Market')['Sales'].sum().reset_index()

vendas_por_market['Sales'] = (vendas_por_market['Sales'] / 1000000).round(2)
vendas_por_market.rename(columns={'Sales': 'Vendas (em milhões)', 'Market':␣
↪'Mercado Global'}, inplace=True)

# Ordenando os resultados
vendas_por_market = vendas_por_market.sort_values(by='Vendas (em milhões)',␣
↪ascending=False).reset_index(drop=True)

vendas_por_market
```

[ ]:
```
fig = px.bar(vendas_por_market, x='Mercado Global', y='Vendas (em milhões)',␣
↪color = 'Mercado Global',
             text = 'Vendas (em milhões)', title='Total em vendas por Mercado␣
↪Global (em milhões)')

fig.update_traces(textposition='outside', texttemplate='%{text}MM',␣
↪textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

[ ]:
```
# Agora, agrupar os dados por categoria e data, somando o lucro
Vendas_por_market_e_data = df_filtrado.groupby(['Market', pd.Grouper(key='order␣
↪date (DateOrders)', freq='M')])['Sales'].sum().reset_index()

Vendas_por_market_e_data['Sales'] = (Vendas_por_market_e_data['Sales'] /␣
↪1000000).round(2)
Vendas_por_market_e_data.rename(columns={'Market':'Mercado Global', 'Sales':␣
↪'Vendas (em milhões)', 'order date (DateOrders)': 'Data'}, inplace=True)

# Calculando a variação das vendas por mercado global
Vendas_por_market_e_data['Variação de Vendas'] = Vendas_por_market_e_data.
↪groupby('Mercado Global')['Vendas (em milhões)'].diff()

# Calculando a variação percentual das vendas por mercado global
Vendas_por_market_e_data['Variação Percentual'] = Vendas_por_market_e_data.
↪groupby('Mercado Global')['Vendas (em milhões)'].pct_change()
```

```
# Convertendo a variação para porcentagem e arredondando
Vendas_por_market_e_data['Variação Percentual'] =␣
 ↪(Vendas_por_market_e_data['Variação Percentual'] * 100).round(2)

# Criar o gráfico usando Plotly
fig = px.line(Vendas_por_market_e_data, x='Data', y='Vendas (em milhões)',␣
 ↪color='Mercado Global', title='Vendas por Mercado Global ao longo do tempo␣
 ↪(em milhões)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Vendas_por_market_e_data, x='Data', y='Variação de Vendas',␣
 ↪color='Mercado Global', title='Variação de vendas por Mercado Global ao␣
 ↪longo do tempo (em milhões)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando os mercados globais e os meses que tiveram uma variação maior que␣
 ↪200% nas vendas
Vendas_por_market_e_data[abs(Vendas_por_market_e_data['Variação Percentual']) >␣
 ↪200.00]
```

As vendas por seguimento oscilaram durante todo o período.

1.1.11 Destaque para os meses de variação positiva nas vendas:

- **Europe - Junho/2015**: 4800% - Junho e setembro foram meses bem favoráveis para o mercado Europeu.
- **Pacific Asia - Novembro/2015**: 280,77% - Para este mercado, os meses novembro e setembro foram bem positivos.

1.1.12 Vendas por região do cliente

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
Vendas_por_regiao_do_cliente = df_filtrado.groupby('Customer Country')['Sales'].
 ↪sum().reset_index()

Vendas_por_regiao_do_cliente['Sales'] = (Vendas_por_regiao_do_cliente['Sales'] /
 ↪ 1000000).round(2)
Vendas_por_regiao_do_cliente.rename(columns={'Sales': 'Vendas (em milhões)',␣
 ↪'Customer Country': 'País de origem do cliente'}, inplace=True)
```

```
# Ordenando os resultados
Vendas_por_regiao_do_cliente = Vendas_por_regiao_do_cliente.
 ↪sort_values(by='Vendas (em milhões)', ascending=False).reset_index(drop=True)

Vendas_por_regiao_do_cliente
```

[ ]:
```
fig = px.bar(Vendas_por_regiao_do_cliente, x='País de origem do cliente',␣
 ↪y='Vendas (em milhões)', color = 'País de origem do cliente',
             text = 'Vendas (em milhões)', title='Vendas por região de origem␣
 ↪do cliente (em milhões)')

fig.update_traces(textposition='outside', texttemplate='%{text}MM',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

Os clientes da empresa são basicamente de duas nacionalidades: Estados Unidos e Porto Rico.

[ ]:
```
# Agora, agrupar os dados por categoria e data, somando o lucro
Vendas_por_regiao_do_cliente_e_data = df_filtrado.groupby(['Customer Country',␣
 ↪pd.Grouper(key='order date (DateOrders)', freq='M')])['Sales'].sum().
 ↪reset_index()

Vendas_por_regiao_do_cliente_e_data['Sales'] =␣
 ↪(Vendas_por_regiao_do_cliente_e_data['Sales'] / 1000000).round(2)

Vendas_por_regiao_do_cliente_e_data.rename(columns={'Customer Country':'País de␣
 ↪origem do cliente', 'Sales': 'Vendas (em milhoes)', 'order date␣
 ↪(DateOrders)': 'Data'}, inplace=True)

# Calculando a variação das vendas por regiao do cliente
Vendas_por_regiao_do_cliente_e_data['Variação de Vendas'] =␣
 ↪Vendas_por_regiao_do_cliente_e_data.groupby('País de origem do␣
 ↪cliente')['Vendas (em milhoes)'].diff()

# Calculando a variação percentual das vendas por regiao do cliente
Vendas_por_regiao_do_cliente_e_data['Variação Percentual'] =␣
 ↪Vendas_por_regiao_do_cliente_e_data.groupby('País de origem do␣
 ↪cliente')['Vendas (em milhoes)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
Vendas_por_regiao_do_cliente_e_data['Variação Percentual'] =␣
 ↪(Vendas_por_regiao_do_cliente_e_data['Variação Percentual'] * 100).round(2)

# Criar o gráfico usando Plotly
```

```
fig = px.line(Vendas_por_regiao_do_cliente_e_data, x='Data', y='Vendas (em␣
 ↪milhoes)', color='País de origem do cliente', title='Vendas por região do␣
 ↪cliente ao Longo do Tempo (em milhoes)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Vendas_por_regiao_do_cliente_e_data, x='Data', y='Variação de␣
 ↪Vendas', color='País de origem do cliente', title='Variação de vendas por␣
 ↪região do cliente ao Longo do Tempo (em milhoes)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando os mercados globais e os meses que tiveram uma variação maior que␣
 ↪10% nas vendas
Vendas_por_regiao_do_cliente_e_data[abs(Vendas_por_regiao_do_cliente_e_data['Variação␣
 ↪Percentual']) > 10.00]
```

Durante o período avaliado, percebe-se que:

- Os Estados Unidos tiveram variações mais significativas:: **Fevereiro/2015 (-14,29%), Março/2015 (14,81%) e Maio/2017 (15,52%)**

- Puerto Rico teve variações mais significativas: **Maio/2015 (17,65%), Março/2015 (14,71%) e Março/2017 (11,43%)**

1.1.13 Vendas por região de destino

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
Vendas_por_destino = df_filtrado.groupby('Order Region')['Sales'].sum().
 ↪reset_index()

Vendas_por_destino['Sales'] = (Vendas_por_destino['Sales'] / 1000000).round(2)

Vendas_por_destino.rename(columns={'Order Region':'Região de destino', 'Sales':␣
 ↪'Vendas (em milhões)'}, inplace=True)

# Ordenando os resultados
Vendas_por_destino = Vendas_por_destino.sort_values(by='Vendas (em milhões)',␣
 ↪ascending=False).reset_index(drop=True)

Vendas_por_destino
```

[ ]:
```
fig = px.bar(Vendas_por_destino.head(10), x='Região de destino', y='Vendas (em␣
↪milhões)', color = 'Região de destino',
             text = 'Vendas (em milhões)', title='Top 10 regiões de destino com␣
↪mais vendas (em milhões)')

fig.update_traces(textposition='outside', texttemplate='%{text}MM',␣
↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

[ ]:
```
# Obtendo a lista das regiões mais lucrativas no geral
top_5_regions = Vendas_por_destino.head(5)['Região de destino'].tolist()

# Agora, agrupar os dados por categoria e data, somando o lucro
Vendas_por_destino_e_data = df_filtrado.groupby(['Order Region', pd.
↪Grouper(key='order date (DateOrders)', freq='M')])['Sales'].sum().
↪reset_index()

Vendas_por_destino_e_data['Sales'] = (Vendas_por_destino_e_data['Sales'] /␣
↪1000000).round(2)

Vendas_por_destino_e_data.rename(columns={'Order Region':'Região de destino',␣
↪'Sales': 'Vendas (em milhões)', 'order date (DateOrders)':'Data'},␣
↪inplace=True)

# Calculando a variação das vendas por regiao de destino da entrega
Vendas_por_destino_e_data['Variação de Vendas'] = Vendas_por_destino_e_data.
↪groupby('Região de destino')['Vendas (em milhões)'].diff()

# Calculando a variação percentual das vendas por regiao de destino da entrega
Vendas_por_destino_e_data['Variação Percentual'] = Vendas_por_destino_e_data.
↪groupby('Região de destino')['Vendas (em milhões)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
Vendas_por_destino_e_data['Variação Percentual'] =␣
↪(Vendas_por_destino_e_data['Variação Percentual'] * 100).round(2)

# Filtrando do dataframe 'Lucro_por_regiao_e_data' somente as regiões do␣
↪'top_5_regions'
Vendas_por_destino_e_data_filtrado =␣
↪Vendas_por_destino_e_data[Vendas_por_destino_e_data['Região de destino'].
↪isin(top_5_regions)]

# Criar o gráfico usando Plotly
```

```
fig = px.line(Vendas_por_destino_e_data_filtrado, x='Data', y='Vendas (em␣
 ↪milhões)', color='Região de destino', title='Top 5 das regiões de destino␣
 ↪com mais vendas ao longo do tempo (em milhões)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Vendas_por_destino_e_data_filtrado, x='Data', y='Variação de␣
 ↪Vendas', color='Região de destino', title='Variação das vendas das Top 5␣
 ↪regiões que mais venderam (em milhões)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando os destinos das entregas e os meses que tiveram uma variação maior␣
 ↪que 1000% nas vendas
Vendas_por_destino_e_data[~Vendas_por_destino_e_data['Variação Percentual'].
 ↪isin([np.inf, -np.inf]) &
    (abs(Vendas_por_destino_e_data['Variação Percentual']) > 500.00)]
```

As vendas de acordo com a região de destino oscilaram durante todo o período.

1.1.14 Destaque para os meses de variação positiva nas vendas:

- **Northern Europe - Junho/2015**: 1800%
- **Western Europe - Junho/2015**: 5800%
- Regiões da Europa como um todo tiveram bons desempenhos nos meses de junho e setembro

6.3 Análise de Lucratividade

[ ]:
```
# Criando uma nova coluna, margem de lucro = (Benefício por pedido / Vendas) *␣
 ↪100
df_filtrado['Profit Margin'] = (df_filtrado['Benefit per order'] /␣
 ↪df_filtrado['Sales']) * 100

# Visualizando a distribuição da margem de lucro
plt.figure(figsize=(16, 9))
sns.histplot(df_filtrado['Profit Margin'], bins=30, kde=True)
plt.title('Distribuição da Margem de Lucro')
plt.xlabel('Margem de Lucro (%)')
plt.ylabel('Frequência')
plt.show()
```

[ ]:
```
df_filtrado['Benefit per order'].describe()
```

[ ]:
```
fig = sns.boxplot(y='Benefit per order', data=df_filtrado)
plt.xticks(rotation=90)
```

O boxplot acima mostra que a mediana, Q1 (primeiro quartil) e Q3 (terceiro quartil) estão muito próximos a zero, ou seja, para grande maioria dos casos não há uma margem de lucro positiva significante na venda dos produtos, mesmo com os outliers acima do limite superior. Além disso, há outliers bem abaixo do limite inferior e bem abaixo de zero, o que indica que a distribuição é bem assimétrica à esquerda e também que grande parte das vendas gera prejuízo à empresa.

[ ]:
```
# Agrupar os dados por ano e mês e somar as vendas
benefit_over_time = df_filtrado.resample('M', on='order date (DateOrders)').
 ↪sum()['Benefit per order']

# Criar um DataFrame a partir da série
benefit_over_time_df = benefit_over_time.reset_index()

# Calculando a variação das vendas
benefit_over_time_df['Variação do Lucro'] = benefit_over_time_df['Benefit per␣
 ↪order'].diff()

# Calculando a variação percentual das vendas
benefit_over_time_df['Variação Percentual'] = benefit_over_time_df['Benefit per␣
 ↪order'].pct_change()

# Convertendo a variação para porcentagem e arredondando
benefit_over_time_df['Variação Percentual'] = (benefit_over_time_df['Variação␣
 ↪Percentual'] * 100).round(2)


# Visualização
fig = px.line(benefit_over_time_df, x='order date (DateOrders)', y='Benefit per␣
 ↪order',
              title='Lucros Totais por Mês', labels={'order date (DateOrders)':␣
 ↪'Data', 'Benefit per order': 'Lucros Totais'})
fig.show()
```

[ ]:
```
benefit_over_time.describe()
```

Os lucros variam em torno de uma média de 107 mil, atingindo a máxima histórica com 126 mil, no mês de agosto de 2017.

[ ]:
```
# Visualização
fig = px.line(benefit_over_time_df, x='order date (DateOrders)', y='Variação do␣
 ↪Lucro',
              title='Variação dos Lucros Totais por Mês', labels={'order date␣
 ↪(DateOrders)': 'Data'})
fig.update_layout(width=1000, height=500)
```

```
fig.show()
```

[ ]:
```
# Destacando os meses que tiveram uma variação maior que 10% nos lucros da␣
 ↪empresa
benefit_over_time_df[abs(benefit_over_time_df['Variação Percentual']) > 10.00]
```

A lucratividade oscilou durante todo o período avaliado, mas com destaque para as variações:

- **Fevereiro/2016**: -19,15% e **Fevereiro/2015**: -12,07%,
- **Março/2016**: 16,42% e **Março/2015**: 15,47%

1.1.15 Lucratividade por categoria

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
Lucro_por_categoria = df_filtrado.groupby('Category Name')['Benefit per order'].
 ↪sum().reset_index()

Lucro_por_categoria['Benefit per order'] = (Lucro_por_categoria['Benefit per␣
 ↪order'] / 1000).round(2)

Lucro_por_categoria.rename(columns={'Category Name':'Categoria','Benefit per␣
 ↪order': 'Lucro (em milhares)'}, inplace=True)

# Ordenando os resultados
Lucro_por_categoria = Lucro_por_categoria.sort_values(by='Lucro (em milhares)',␣
 ↪ascending=False).reset_index(drop=True)

Lucro_por_categoria
```

[ ]:
```
fig = px.bar(Lucro_por_categoria.head(10), x='Categoria', y='Lucro (em␣
 ↪milhares)', color = 'Categoria',
             text = 'Lucro (em milhares)', title='Top 10 categorias mais␣
 ↪lucrativas (em milhares)')

fig.update_traces(textposition='outside', texttemplate='%{text}k',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

[ ]:
```
# Agora, agrupar os dados por categoria e data, somando o lucro
Lucro_por_categoria_e_data = df_filtrado.groupby(['Category Name', pd.
 ↪Grouper(key='order date (DateOrders)', freq='M')])['Benefit per order'].
 ↪sum().reset_index()
```

```
Lucro_por_categoria_e_data.rename(columns={'Category Name':'Categoria',␣
 ↪'Benefit per order': 'Lucro (em milhares)', 'order date (DateOrders)':␣
 ↪'Data'}, inplace=True)

# Calculando a variação dos lucros por categoria
Lucro_por_categoria_e_data['Variação de Lucro'] = Lucro_por_categoria_e_data.
 ↪groupby('Categoria')['Lucro (em milhares)'].diff()

# Calculando a variação percentual dos lucros por categoria
Lucro_por_categoria_e_data['Variação Percentual'] = Lucro_por_categoria_e_data.
 ↪groupby('Categoria')['Lucro (em milhares)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
Lucro_por_categoria_e_data['Variação Percentual'] =␣
 ↪(Lucro_por_categoria_e_data['Variação Percentual'] * 100).round(2)

# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_categoria_e_data, x='Data', y='Lucro (em milhares)',␣
 ↪color='Categoria', title='Lucro por categoria ao longo do tempo (em␣
 ↪milhares)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_categoria_e_data, x='Data', y='Variação de Lucro',␣
 ↪color='Categoria', title='Variação no lucro por categoria ao longo do tempo␣
 ↪(em milhares)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando os meses em que as categorias tiveram uma variação maior que 2000%␣
 ↪nos lucros da empresa
Lucro_por_categoria_e_data[abs(Lucro_por_categoria_e_data['Variação␣
 ↪Percentual']) > 2000.00]
```

Os lucros oscilaram durante todo o período.

1.1.16 Destaque para os meses de variação positiva nos lucros:

- **Fitness Accessories - Setembro/2017**: 12910,55%

- **Soccer - Maio/2017**: 6,41%
- **Hockey - Setembro/2017**: 2058,97%

1.1.17 Destaque para os meses de variação negativa nos lucros:

- **Trade-In - Abril/2016**: -10646,36%,

- **Hunting & Shooting - Maio/2015**: -3182,77%
- **Golf Bags & Carts - Maio/2017**: -2013,14%

[ ]:
```
sns.set(rc = {'figure.figsize':(23,11)})
fig = sns.boxplot(x='Category Name', y='Benefit per order', data=df_filtrado)
plt.xticks(rotation=90);
```

1.1.18 Lucratividade por Mercado Global

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
Lucro_por_market = df_filtrado.groupby('Market')['Benefit per order'].sum().
 ↪reset_index()

Lucro_por_market['Benefit per order'] = (Lucro_por_market['Benefit per order'] /
 ↪ 1000000).round(2)

Lucro_por_market.rename(columns={'Benefit per order': 'Lucro (em milhões)'},␣
 ↪inplace=True)

# Ordenando os resultados
Lucro_por_market = Lucro_por_market.sort_values(by='Lucro (em milhões)',␣
 ↪ascending=False).reset_index(drop=True)

Lucro_por_market
```

[ ]:
```
fig = px.bar(Lucro_por_market, x='Market', y='Lucro (em milhões)', color =␣
 ↪'Market',
             text = 'Lucro (em milhões)', title='Lucro total por Mercado Global␣
 ↪(em milhões)')

fig.update_traces(textposition='outside', texttemplate='%{text}MM',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

[ ]:
```
# Agora, agrupar os dados por categoria e data, somando o lucro
Lucro_por_market_e_data = df_filtrado.groupby(['Market', pd.Grouper(key='order␣
 ↪date (DateOrders)', freq='M')])['Benefit per order'].sum().reset_index()

Lucro_por_market_e_data.rename(columns={'Market':'Mercado Global', 'Benefit per␣
 ↪order': 'Lucro (em milhares)', 'order date (DateOrders)': 'Data'},␣
 ↪inplace=True)
```

```
# Calculando a variação dos lucros por mercado global
Lucro_por_market_e_data['Variação de Lucro'] = Lucro_por_market_e_data.
 ↪groupby('Mercado Global')['Lucro (em milhares)'].diff()

# Calculando a variação percentual dos lucros por mercado global
Lucro_por_market_e_data['Variação Percentual'] = Lucro_por_market_e_data.
 ↪groupby('Mercado Global')['Lucro (em milhares)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
Lucro_por_market_e_data['Variação Percentual'] =␣
 ↪(Lucro_por_market_e_data['Variação Percentual'] * 100).round(2)

# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_market_e_data, x='Data', y='Lucro (em milhares)',␣
 ↪color='Mercado Global', title='Lucro por Mercado Global ao Longo do Tempo␣
 ↪(em milhares)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_market_e_data, x='Data', y='Variação de Lucro',␣
 ↪color='Mercado Global', title='Variação no lucro por Mercado Global ao Longo␣
 ↪do Tempo (em milhares)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando os meses em que mercados globais tiveram uma variação maior que␣
 ↪1000% nos lucros da empresa
Lucro_por_market_e_data[abs(Lucro_por_market_e_data['Variação Percentual']) >␣
 ↪1000.00]
```

Os lucros oscilaram durante todo o período.

1.1.19 Destaque para os meses de variação positiva nos lucros:

- **Europe - junho/2015**: 3571,26%
- **Pacific Asia - setembro/2016**: 1984,15%

[ ]:
```
sns.set(rc = {'figure.figsize':(16,9)})
fig = sns.boxplot(x='Market', y='Benefit per order', data=df_filtrado)
plt.xticks(rotation=90);
```

1.1.20 Regiões de destino mais lucrativos para a empresa

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
Lucro_por_destino = df_filtrado.groupby('Order Region')['Benefit per order'].
 ↪sum().reset_index()

Lucro_por_destino['Benefit per order'] = (Lucro_por_destino['Benefit per␣
 ↪order'] / 1000).round(2)

Lucro_por_destino.rename(columns={'Order Region':'Região de destino', 'Benefit␣
 ↪per order': 'Lucro (em milhares)'}, inplace=True)

# Ordenando os resultados
Lucro_por_destino = Lucro_por_destino.sort_values(by='Lucro (em milhares)',␣
 ↪ascending=False).reset_index(drop=True)

Lucro_por_destino
```

[ ]:
```
fig = px.bar(Lucro_por_destino.head(10), x='Região de destino', y='Lucro (em␣
 ↪milhares)', color = 'Região de destino',
             text = 'Lucro (em milhares)', title='Top 10 regiões de destino␣
 ↪mais lucrativas para a empresa (em milhares)')

fig.update_traces(textposition='outside', texttemplate='%{text}k',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

[ ]:
```
# Obtendo a lista das regiões mais lucrativas no geral
top_5_regions = Lucro_por_destino.head(5)['Região de destino'].tolist()

# Agora, agrupar os dados por categoria e data, somando o lucro
Lucro_por_destino_e_data = df_filtrado.groupby(['Order Region', pd.
 ↪Grouper(key='order date (DateOrders)', freq='M')])['Benefit per order'].
 ↪sum().reset_index()

Lucro_por_destino_e_data.rename(columns={'Order Region':'Região de destino',␣
 ↪'Benefit per order': 'Lucro (em milhares)', 'order date (DateOrders)':
 ↪'Data'}, inplace=True)

# Calculando a variação dos lucros por destino
Lucro_por_destino_e_data['Variação de Lucro'] = Lucro_por_destino_e_data.
 ↪groupby('Região de destino')['Lucro (em milhares)'].diff()

# Calculando a variação percentual dos lucros por destino
```

```
Lucro_por_destino_e_data['Variação Percentual'] = Lucro_por_destino_e_data.
 ↪groupby('Região de destino')['Lucro (em milhares)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
Lucro_por_destino_e_data['Variação Percentual'] =␣
 ↪(Lucro_por_destino_e_data['Variação Percentual'] * 100).round(2)

# Filtrando do dataframe 'Lucro_por_regiao_e_data' somente as regiões do␣
 ↪'top_5_regions'
Lucro_por_destino_e_data_filtrado =␣
 ↪Lucro_por_destino_e_data[Lucro_por_destino_e_data['Região de destino'].
 ↪isin(top_5_regions)]

# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_destino_e_data_filtrado, x='Data', y='Lucro (em␣
 ↪milhares)', color='Região de destino', title='Lucro das 5 regiões de destino␣
 ↪mais rentáveis ao longo do tempo (em milhares)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_destino_e_data_filtrado, x='Data', y='Variação de␣
 ↪Lucro', color='Região de destino', title='Variação do lucro das 5 regiões de␣
 ↪destino mais rentáveis ao longo do tempo')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando os meses em que destinos tiveram uma variação maior que 2000% nos␣
 ↪lucros da empresa
Lucro_por_destino_e_data[abs(Lucro_por_destino_e_data['Variação Percentual']) >␣
 ↪2000.00]
```

Os lucros oscilaram durante todo o período.

1.1.21 Destaque para os meses de variação positiva nos lucros:

- **Western Europe - junho/2017**: 12047,33%

- **Western Europe - junho/2015**: 5789,17%
- **Southern Europe - Setembro/2017**: 4936,73%

1.1.22 Destaque para os meses de variação negativa nos lucros:

- **Southern Europe - junho/2017**: -25257,81%,

- **South Asia - setembro/2016**: -15387,03%
- **Northern Europe - setembro/2016**: -2553,40%

[ ]:
```
sns.set(rc = {'figure.figsize':(16,9)})
fig = sns.boxplot(x='Order Region', y='Benefit per order', data=df_filtrado)
plt.xticks(rotation=90);
```

1.1.23 Segmentos de clientes mais lucrativos

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
Lucro_por_segmento = df_filtrado.groupby('Customer Segment')['Benefit per␣
 ↪order'].sum().reset_index()

Lucro_por_segmento['Benefit per order'] = (Lucro_por_segmento['Benefit per␣
 ↪order'] / 1000000).round(2)

Lucro_por_segmento.rename(columns={'Benefit per order': 'Lucro (em milhões)',␣
 ↪'Customer Segment': 'Segmento de Cliente'}, inplace=True)

# Ordenando os resultados
Lucro_por_segmento = Lucro_por_segmento.sort_values(by='Lucro (em milhões)',␣
 ↪ascending=False).reset_index(drop=True)

Lucro_por_segmento
```

[ ]:
```
fig = px.bar(Lucro_por_segmento, x='Segmento de Cliente', y='Lucro (em␣
 ↪milhões)', color = 'Segmento de Cliente',
             text = 'Lucro (em milhões)', title='Lucratividade por segmento de␣
 ↪cliente (em milhões)',
             labels={'Order Region': 'Região'})

fig.update_traces(textposition='outside', texttemplate='%{text}MM',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

[ ]:
```
# Agora, agrupar os dados por categoria e data, somando o lucro
Lucro_por_segmento_e_data = df_filtrado.groupby(['Customer Segment', pd.
 ↪Grouper(key='order date (DateOrders)', freq='M')])['Benefit per order'].
 ↪sum().reset_index()
```

```
Lucro_por_segmento_e_data.rename(columns={'Customer Segment':'Segmento',␣
 ↪'Benefit per order': 'Lucro (em milhares)', 'order date (DateOrders)':␣
 ↪'Data'}, inplace=True)

# Calculando a variação dos lucros por segmento
Lucro_por_segmento_e_data['Variação de Lucro'] = Lucro_por_segmento_e_data.
 ↪groupby('Segmento')['Lucro (em milhares)'].diff()

# Calculando a variação percentual dos lucros por segmento
Lucro_por_segmento_e_data['Variação Percentual'] = Lucro_por_segmento_e_data.
 ↪groupby('Segmento')['Lucro (em milhares)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
Lucro_por_segmento_e_data['Variação Percentual'] =␣
 ↪(Lucro_por_segmento_e_data['Variação Percentual'] * 100).round(2)



# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_segmento_e_data, x='Data', y='Lucro (em milhares)',␣
 ↪color='Segmento', title='Lucro por segmento de cliente ao Longo do Tempo')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfic'
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_segmento_e_data, x='Data', y='Variação de Lucro',␣
 ↪color='Segmento', title='Variação do lucro por segmento de cliente ao Longo␣
 ↪do Tempo')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfic'
fig.show()
```

[ ]:
```
# Destacando os meses em que segmentos tiveram uma variação maior que 40% nos␣
 ↪lucros da empresa
Lucro_por_segmento_e_data[abs(Lucro_por_segmento_e_data['Variação Percentual'])␣
 ↪> 40.00]
```

Os lucros oscilaram durante todo o período.

1.1.24 Destaque para os meses de variação positiva nos lucros:

- **Home Office - agosto/2017**: 60,62%
- **Corporate - dezembro/2015**: 47,71%

1.1.25 Destaque para os meses de variação negativa nos lucros:

- **Home Office - agosto/2016**: -43,69%

[ ]:
```
sns.set(rc = {'figure.figsize':(16,9)})
fig = sns.boxplot(x='Customer Segment', y='Benefit per order', data=df_filtrado)
plt.xticks(rotation=90);
```

1.1.26 Departamentos mais lucrativos

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
Lucro_por_depart = df_filtrado.groupby('Department Name')['Benefit per order'].
 ↪sum().reset_index()

Lucro_por_depart['Benefit per order'] = (Lucro_por_depart['Benefit per order'] /
 ↪ 1000000).round(2)

Lucro_por_depart.rename(columns={'Benefit per order': 'Lucro (em milhões)',␣
 ↪'Department Name': 'Departamento'}, inplace=True)

# Ordenando os resultados
Lucro_por_depart = Lucro_por_depart.sort_values(by='Lucro (em milhões)',␣
 ↪ascending=False).reset_index(drop=True)

Lucro_por_depart
```

[ ]:
```
fig = px.bar(Lucro_por_depart, x='Departamento', y='Lucro (em milhões)', color␣
 ↪= 'Departamento',
             text = 'Lucro (em milhões)', title='Lucratividade por departamento␣
 ↪(em milhões)')

fig.update_traces(textposition='outside', texttemplate='%{text}MM',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

[ ]:
```
# Agora, agrupar os dados por categoria e data, somando o lucro
Lucro_por_depart_e_data = df_filtrado.groupby(['Department Name', pd.
 ↪Grouper(key='order date (DateOrders)', freq='M')])['Benefit per order'].
 ↪sum().reset_index()

Lucro_por_depart_e_data.rename(columns={'Department Name':'Departamento',␣
 ↪'Benefit per order': 'Lucro (em milhares)', 'order date (DateOrders)':␣
 ↪'Data'}, inplace=True)

# Calculando a variação dos lucros por departamento
```

```
Lucro_por_depart_e_data['Variação de Lucro'] = Lucro_por_depart_e_data.
 ↪groupby('Departamento')['Lucro (em milhares)'].diff()

# Calculando a variação percentual dos lucros por departamento
Lucro_por_depart_e_data['Variação Percentual'] = Lucro_por_depart_e_data.
 ↪groupby('Departamento')['Lucro (em milhares)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
Lucro_por_depart_e_data['Variação Percentual'] =␣
 ↪(Lucro_por_depart_e_data['Variação Percentual'] * 100).round(2)

# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_depart_e_data, x='Data', y='Lucro (em milhares)',␣
 ↪color='Departamento', title='Lucro por departamento ao longo do tempo (em␣
 ↪milhares)')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_depart_e_data, x='Data', y='Variação de Lucro',␣
 ↪color='Departamento', title='Variação do lucro por departamento ao longo do␣
 ↪tempo')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando os meses em que departamentos tiveram uma variação maior que 300%␣
 ↪nos lucros da empresa
Lucro_por_depart_e_data[abs(Lucro_por_depart_e_data['Variação Percentual']) >␣
 ↪300.00]
```

Os lucros oscilaram durante todo o período.

1.1.27 Destaque para os meses de variação positiva nos lucros:

- **Footwear - março/2016**: 401,98%

- **Fitness - março/2016**: 338,62%
- **Fitness - agosto/2017**: 331,72%

1.1.28 Lucro por região do cliente

[ ]:
```
# Agrupando por 'Category Name' e somando as vendas
Lucro_por_regiao_do_cliente = df_filtrado.groupby('Customer Country')['Benefit␣
 ↪per order'].sum().reset_index()

Lucro_por_regiao_do_cliente['Benefit per order'] =␣
 ↪(Lucro_por_regiao_do_cliente['Benefit per order'] / 1000000).round(2)
Lucro_por_regiao_do_cliente.rename(columns={'Benefit per order': 'Lucro (em␣
 ↪milhões)', 'Customer Country': 'País de origem do cliente'}, inplace=True)

# Ordenando os resultados
Lucro_por_regiao_do_cliente = Lucro_por_regiao_do_cliente.sort_values(by='Lucro␣
 ↪(em milhões)', ascending=False).reset_index(drop=True)

Lucro_por_regiao_do_cliente
```

[ ]:
```
fig = px.bar(Lucro_por_regiao_do_cliente, x='País de origem do cliente',␣
 ↪y='Lucro (em milhões)', color = 'País de origem do cliente',
             text = 'Lucro (em milhões)', title='Lucro por região de origem do␣
 ↪cliente')

fig.update_traces(textposition='outside', texttemplate='%{text}MM',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

[ ]:
```
# Agora, agrupar os dados por categoria e data, somando o lucro
Lucro_por_regiao_do_cliente_e_data = df_filtrado.groupby(['Customer Country',␣
 ↪pd.Grouper(key='order date (DateOrders)', freq='M')])['Benefit per order'].
 ↪sum().reset_index()

Lucro_por_regiao_do_cliente_e_data['Benefit per order'] =␣
 ↪(Lucro_por_regiao_do_cliente_e_data['Benefit per order'] / 1000).round(2)

Lucro_por_regiao_do_cliente_e_data.rename(columns={'Customer Country':'País de␣
 ↪origem do cliente', 'Benefit per order': 'Lucro (em milhares)', 'order date␣
 ↪(DateOrders)': 'Data'}, inplace=True)

# Calculando a variação dos lucros por região do cliente
Lucro_por_regiao_do_cliente_e_data['Variação de Lucro'] =␣
 ↪Lucro_por_regiao_do_cliente_e_data.groupby('País de origem do␣
 ↪cliente')['Lucro (em milhares)'].diff()

# Calculando a variação percentual dos lucros por região do cliente
```

```
Lucro_por_regiao_do_cliente_e_data['Variação Percentual'] =␣
 ↪Lucro_por_regiao_do_cliente_e_data.groupby('País de origem do␣
 ↪cliente')['Lucro (em milhares)'].pct_change()

# Convertendo a variação para porcentagem e arredondando
Lucro_por_regiao_do_cliente_e_data['Variação Percentual'] =␣
 ↪(Lucro_por_regiao_do_cliente_e_data['Variação Percentual'] * 100).round(2)

# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_regiao_do_cliente_e_data, x='Data', y='Lucro (em␣
 ↪milhares)', color='País de origem do cliente', title='Lucro por região do␣
 ↪cliente - Ao Longo do Tempo')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Criar o gráfico usando Plotly
fig = px.line(Lucro_por_regiao_do_cliente_e_data, x='Data', y='Variação de␣
 ↪Lucro', color='País de origem do cliente', title='Variação do lucro por␣
 ↪região do cliente - Ao Longo do Tempo')
fig.update_layout(width=1000, height=500)

# Mostrar o gráfico
fig.show()
```

[ ]:
```
# Destacando os meses em que departamentos tiveram uma variação maior que 30%␣
 ↪nos lucros da empresa
Lucro_por_regiao_do_cliente_e_data[abs(Lucro_por_regiao_do_cliente_e_data['Variação␣
 ↪Percentual']) > 30.00]
```

Os lucros oscilaram durante todo o período.

1.1.29 Destaque para os meses de variação positiva nos lucros:

- **Puerto Rico - março/2016**: 38,72%

1.1.30 Destaque para os meses de variação negativa nos lucros:

- **Puerto Rico - janeiro/2016**: -33,25%

7. Análise de Operações de Envio

7.1 Risco de atraso nas entregas

[ ]:
```
risco_entrega_atrasada = df_filtrado['Late_delivery_risk'].apply(lambda lat:␣
 ↪'Risco Alto' if lat == 1 else 'Risco Baixo')
risco_entrega_atrasada_count = risco_entrega_atrasada.value_counts()
```

```
# Tamanho do gráfico
plt.figure(figsize=(8,6))

# Cria um gráfico de barras com índice e contagem
barra = plt.bar(
    risco_entrega_atrasada_count.index, # valor no eixo x
    risco_entrega_atrasada_count.values, # valor no eixo y
    color = ['steelblue', 'lightcoral'] # cores das barras
)

# Rotulo do eixo y, letra tamanho 8
plt.ylabel('Número de entregas', fontsize = 12)

# Titulo, letra tamanho 14
plt.title('Risco de atraso nas entregas da empresa', fontsize = 16)

# Adicionando a contagem em cima das barras
for bar in barra:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval, int(yval), va='bottom',␣
 ↪ha='center', fontsize=13)

plt.show()
```

[ ]:
```
risco_entrega_atrasada.value_counts(1).round(4)
```

Pela análise foi possível verificar que, no período entre janeiro/2015 à setembro/2017, **57.28% das entregas** realizadas pela empresa tiveram um **alto risco de atraso na entrega**.

1.1.31 Risco de atraso por departamento

[ ]:
```
risco_atraso_depart = df_filtrado.groupby(['Department Name',␣
 ↪'Late_delivery_risk']).size().unstack(fill_value=0)

# Renomeando colunas
risco_atraso_depart.rename(columns={'Department Name':'Departamento', 0: 'Risco␣
 ↪Baixo', 1: 'Risco Alto'}, inplace=True)
risco_atraso_depart = risco_atraso_depart.reset_index()

# Removendo a coluna Late_delivery_risk
risco_atraso_depart.columns = ['Departamento', 'Risco Baixo', 'Risco Alto']
```

[ ]:
```
risco_atraso_depart
```

[ ]:
```
fig = go.Figure()

# Risco de entrega baixo (Late_delivery_risk = 0)
```

```
fig.add_trace(go.Bar(x=risco_atraso_depart['Departamento'],␣
 ↪y=risco_atraso_depart['Risco Baixo'], name='Risco Baixo',␣
 ↪marker_color='blue'))

# Risco de entrega alto (Late_delivery_risk = 1)
fig.add_trace(go.Bar(x=risco_atraso_depart['Departamento'],␣
 ↪y=risco_atraso_depart['Risco Alto'], name='Risco Alto', marker_color='red'))

# Atualizando o layout do gráfico
fig.update_layout(title='Risco de atraso em entrega por departamento',␣
 ↪xaxis=dict(title='Nome do Departamento'),
                  yaxis=dict(title='Contagem'), barmode='group')

fig.update_layout(width=1000, height=500)

# Mostrando o gráfico
fig.show()
```

Pelo gráfico acima pode-se concluir que para **todos os departamentos** o risco da entregas atrasar **é majoritariamente alto**.

1.1.32 Risco de atraso por mercado global

[ ]:
```
risco_atraso_mercado_global = df_filtrado.groupby(['Market',␣
 ↪'Late_delivery_risk']).size().unstack(fill_value=0)

# Renomeando colunas
risco_atraso_mercado_global.rename(columns={0: 'Risco Baixo', 1: 'Risco Alto'},␣
 ↪inplace=True)
risco_atraso_mercado_global = risco_atraso_mercado_global.reset_index()

# Removendo a coluna Late_delivery_risk
risco_atraso_mercado_global.columns = ['Mercado Global', 'Risco Baixo', 'Risco␣
 ↪Alto']
```

[ ]:
```
risco_atraso_mercado_global
```

[ ]:
```
fig = go.Figure()

# Risco de entrega baixo (Late_delivery_risk = 0)
fig.add_trace(go.Bar(x=risco_atraso_mercado_global['Mercado Global'],␣
 ↪y=risco_atraso_mercado_global['Risco Baixo'], name='Risco Baixo',␣
 ↪marker_color='blue'))

# Risco de entrega alto (Late_delivery_risk = 1)
```

```
fig.add_trace(go.Bar(x=risco_atraso_mercado_global['Mercado Global'],␣
 ↪y=risco_atraso_mercado_global['Risco Alto'], name='Risco Alto',␣
 ↪marker_color='red'))

# Atualizando o layout do gráfico
fig.update_layout(title='Risco de atraso em entrega por Mercado Global',␣
 ↪xaxis=dict(title='Mercado Global'),
                  yaxis=dict(title='Contagem'), barmode='group')

fig.update_layout(width=1000, height=500)

# Mostrando o gráfico
fig.show()
```

Ou seja, em cada um dos mercados globais **a maioria dos pedidos tem um alto risco da entrega atrasar**.

1.1.33 Proporção de risco de atraso alto por categoria

[ ]:
```
risco_atraso_cat = df_filtrado.groupby(['Category Name', 'Late_delivery_risk']).
 ↪size().unstack(fill_value=0)

# Renomeando colunas
risco_atraso_cat.rename(columns={0: 'Risco Baixo', 1: 'Risco Alto'},␣
 ↪inplace=True)
risco_atraso_cat = risco_atraso_cat.reset_index()

# Removendo a coluna Late_delivery_risk
risco_atraso_cat.columns = ['Category Name', 'Risco Baixo', 'Risco Alto']

# Criando uma coluna chamada 'Proporção de Alto Risco de Atraso'
risco_atraso_cat['Proporção de Alto Risco de Atraso'] =␣
 ↪(risco_atraso_cat['Risco Alto']/(risco_atraso_cat['Risco␣
 ↪Alto']+risco_atraso_cat['Risco Baixo'])).round(3)

# Ordenando o DataFrame pelo 'Proporção de Alto Risco de Atraso'
risco_atraso_cat.sort_values(by='Proporção de Alto Risco de Atraso',␣
 ↪ascending=False, inplace=True)

risco_atraso_cat = risco_atraso_cat.reset_index(drop=True)

risco_atraso_cat
```

Pelo dataframe acima pode-se concluir que, com exceção de uma categoria (**Men's Golf Clubs**), em **todos as outras** os pedidos realizados têm alto risco de atraso na entrega em sua maioria.

7.2 Entrega real VS Entrega agendada

[ ]:
```
df_filtrado['Days for shipping (real)'].describe()
```

[ ]:
```
df_filtrado['Days for shipping (real)'].value_counts()
```

[ ]:
```
df_filtrado['Days for shipment (scheduled)'].describe()
```

[ ]:
```
# Define a localidade para adicionar o separador de milhares
locale.setlocale(locale.LC_ALL, '')

# Cria uma figura com 2 subplots
fig, axs = plt.subplots(1, 2, figsize=(16, 9))

# Cria um boxplot para a variável 'entrega real' no primeiro subplot
bp1 = axs[0].boxplot(df_filtrado['Days for shipping (real)'], patch_artist=True)
axs[0].set_title('Boxplot dos dias reais de entrega')

# Calcula a média e mediana dias de entrega real
mean_entrega_real = np.mean(df_filtrado['Days for shipping (real)'])
median_entrega_real = np.median(df_filtrado['Days for shipping (real)'])

# Define a cor do boxplot
bp1['boxes'][0].set_facecolor('lightblue')

# Adiciona a legenda da média e mediana com separador de milhares
max_rent = np.max(df_filtrado['Days for shipping (real)'])
axs[0].annotate(f'Média = {locale.format_string("%.2f", mean_entrega_real,␣
 ↪grouping=True)}\nMediana = {locale.format_string("%.2f",␣
 ↪median_entrega_real, grouping=True)}',
                xy=(1, max_rent*0.8),
                xytext=(1.15, max_rent*0.8),
                bbox=dict(facecolor='lightblue', edgecolor='blue'),
                fontsize=10)

# Cria um boxplot para a variável 'entrega agendada' no segundo subplot
bp2 = axs[1].boxplot(df_filtrado['Days for shipment (scheduled)'],␣
 ↪patch_artist=True)
axs[1].set_title('Boxplot dos dias previstos de entrega')

# Calcula a média e mediana dos dias previstos para entrega
mean_entrega_agendada = np.mean(df_filtrado['Days for shipment (scheduled)'])
median_entrega_agendada = np.median(df_filtrado['Days for shipment␣
 ↪(scheduled)'])

# Define a cor do boxplot
bp2['boxes'][0].set_facecolor('lightgreen')

# Adiciona a legenda da média e mediana com separador de milhares
```

```
max_total = np.max(df_filtrado['Days for shipment (scheduled)'])
axs[1].annotate(f'Média = {locale.format_string("%.2f", mean_entrega_agendada,␣
 ↪grouping=True)}\nMediana = {locale.format_string("%.2f",␣
 ↪median_entrega_agendada, grouping=True)}',
                xy=(1, max_total*0.8),
                xytext=(1.15, max_total*0.8),
                bbox=dict(facecolor='lightgreen', edgecolor='green'),
                fontsize=10)

# Mostra os gráficos
plt.show()
```

Pela análise dos boxplots:

Distribuição dos dias reais de entrega

- O conjunto de dados dos dias reais de entrega não possui outliers superiores ou inferiores
- O boxplot mostra que a mediana (Q2) está mais próxima de Q1 do que de Q3, mostrando que existe uma cauda mais longa com números maiores, possuindo mais registros com valores baixos do que altos, trata-se de uma distribuição assimétrica à direita. Isso indica que a maioria dos produtos demoram poucos dias para serem entregues (não significando que não estão em atraso).

Distribuição dos dias previstos de entrega

- O conjunto de dados dos dias previstos de entrega não possui outliers superiores ou inferiores
- O boxplot mostra que a mediana (Q2) é exatamente igual ao Q3 e também ao valor máximo, indicando que na grande maioria dos casos a empresa acaba prevendo uma mesma quantidade de dias para a entrega de produtos. Isso ainda contribui para a formação de uma cauda mais longa com números menores, possuindo mais registros com valores altos do que baixos, caracterizando uma distribuição assimétrica à esquerda

[ ]:
```
mediana_entrega_real = df_filtrado['Days for shipping (real)'].median()
mediana_entrega_agendada = df_filtrado['Days for shipment (scheduled)'].median()

mediana_entrega_real_format = (
    "{:,.0f}".format(mediana_entrega_real)
    .replace(",", ".")
)

mediana_entrega_agendada_format = (
    "{:,.0f}".format(mediana_entrega_agendada)
    .replace(",", ".")
)

# Crie dois subplots (um para o histograma de 'entrega real' e outro para o␣
 ↪histograma de 'entrega agendada')
fig = make_subplots(rows=1, cols=2, subplot_titles=('Histograma dos dias para␣
 ↪entrega - real', 'Histograma dos dias para entrega - previsto'))
```

```
# Adicione o histograma de 'entrega  real' ao primeiro subplot
histogram_entrega_real = go.Histogram(x=df_filtrado['Days for shipping␣
 ↪(real)'], nbinsx=8, marker=dict(color='blue'))
fig.add_trace(histogram_entrega_real, row=1, col=1)

# Adicione o histograma de 'entrega agendada' ao segundo subplot
histogram_entrega_agendada = go.Histogram(x=df_filtrado['Days for shipment␣
 ↪(scheduled)'], nbinsx=8, marker=dict(color='green'))
fig.add_trace(histogram_entrega_agendada, row=1, col=2)

# Adicione a linha da mediana a ambos os subplots
line_entrega_real = go.Scatter(x=[mediana_entrega_real, mediana_entrega_real],␣
 ↪y=[0, 60000], mode='lines',
                               line=dict(color='lightblue', dash='dash'),
                               showlegend=True,
                               name=f"Mediana dos dias para entrega - real =␣
 ↪{mediana_entrega_real_format}")
fig.add_trace(line_entrega_real, row=1, col=1)

line_entrega_agendada = go.Scatter(x=[mediana_entrega_agendada,␣
 ↪mediana_entrega_agendada], y=[0, 120000], mode='lines',
                               line=dict(color='lightgreen', dash='dash'),
                               showlegend=True,
                               name=f"Mediana dos dias para entrega - agendada =␣
 ↪{mediana_entrega_agendada_format}")
fig.add_trace(line_entrega_agendada, row=1, col=2)

# Atualize o layout e as configurações dos subplots
fig.update_layout(title_text='Histograma da quantidade de dias para entrega␣
 ↪real e agendada',
                  autosize=False,
                  width=1200,  # Largura total dos subplots
                  height=500)

fig.update_yaxes(range=[0, 60000], row=1, col=1)
fig.update_yaxes(range=[0, 120000], row=1, col=2)

# Mostre o gráfico
fig.show()
```

Os histogramas indicados acima acabam comprovando o comportamento das distribuições verificado pelos boxplots anteriormente.

[ ]:
```
fig, axes = plt.subplots(1, 2, figsize = (18,9))
plt.tight_layout()
```

```
sns.boxplot(y=df_filtrado['Days for shipping (real)'] - df_filtrado['Days for␣
 ↪shipment (scheduled)'],ax= axes[0], palette = 'YlGnBu_r', orient='v')
axes[0].set_title('Boxplot - Delay entre dias reais e previstos para entrega',␣
 ↪fontsize=18)


sns.histplot(df_filtrado['Days for shipping (real)'] - df_filtrado['Days for␣
 ↪shipment (scheduled)'], ax=axes[1], color='lightblue', bins=8)
axes[1].set_title('Histograma - Delay entre dias reais e previstos para␣
 ↪entrega', fontsize=18)

plt.show()
```

[ ]:
```
delay = df_filtrado['Days for shipping (real)'] - df_filtrado['Days for␣
 ↪shipment (scheduled)']
delay.describe()
```

O boxplot e o histograma indicam que 75% dos pedidos possuem até 1 dia de atraso, sendo 4 dias o máximo de atraso que um pedido teve e 2 dias o máximo de adiantamento na chegada.

[ ]:
```
print(f"A assimetria da distribuição do delay assume valor: {skew(delay):.2f},␣
 ↪ou seja, é levemente assimétrica à esquerda.")
```

[ ]:
```
delay.value_counts(1)
```

[ ]:
```
# Contando o número de pedidos com atraso de pelo menos 1 dia
delay_1_dia = (delay >= 1).sum()

# Calculando o número total de pedidos
total_de_pedidos = len(delay)

# Calculando a porcentagem de pedidos com atraso
porcentagem_com_atraso = (delay_1_dia / total_de_pedidos) * 100

# Exibindo o resultado
print(f"Porcentagem de pedidos com atraso de pelo menos 1 dia:␣
 ↪{porcentagem_com_atraso:.2f}%")
print()
print(f"Porcentagem de pedidos com apenas 1 dia de atraso: {delay.
 ↪value_counts(1)[1]:.2f}%")
print()
print(f"Porcentagem de pedidos com 2 dias de atraso: {delay.value_counts(1)[2]:.
 ↪2f}%")
print()
print(f"Porcentagem de pedidos com 3 dias de atraso: {delay.value_counts(1)[3]:.
 ↪2f}%")
print()
```

```
print(f"Porcentagem de pedidos com 4 dias de atraso: {delay.value_counts(1)[4]:.
↪2f}%")
```

[ ]:
```
print(f"Porcentagem de pedidos com um dia adiantado: {delay.value_counts(1)[-1]:
↪.2f}%")
print()
print(f"Porcentagem de pedidos com dois dias adiantado: {delay.
↪value_counts(1)[-2]:.2f}%")
```

[ ]:
```
print(f"Porcentagem de pedidos que chegou exatamente no dia previsto: {delay.
↪value_counts(1)[0]:.2f}%")
```

[ ]:
```
# Obter a contagem para cada status de entrega

delay = (df_filtrado['Days for shipping (real)'] - df_filtrado['Days for␣
↪shipment (scheduled)']).value_counts()
delay_df = delay.reset_index()
delay_df.columns = ['Dias de Atraso', 'Quantidade de Pedidos']

# Convertendo 'Dias de Atraso' para string para obter cores distintas
delay_df['Dias de Atraso'] = delay_df['Dias de Atraso'].astype(str)


fig = px.bar(delay_df, y='Dias de Atraso', x='Quantidade de Pedidos',␣
↪color='Dias de Atraso',
                 title="Quantidade de Pedidos de acordo com os dias de atraso",␣
↪text = 'Quantidade de Pedidos')

fig.update_traces(textposition='outside', texttemplate='%{text}',␣
↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

7.3 Análise dos status de entrega

[ ]:
```
# Obter a contagem para cada status de entrega

status_entrega = df_filtrado['Delivery Status'].value_counts()
status_entrega_df = status_entrega.reset_index()
status_entrega_df.columns = ['Status de Entrega', 'Quantidade de Pedidos']


fig = px.bar(status_entrega_df, x='Status de Entrega', y='Quantidade de␣
↪Pedidos', color='Status de Entrega',
                 title="Quantidade de Pedidos por Status de Entrega", text =␣
↪'Quantidade de Pedidos')
```

```
fig.update_traces(textposition='outside', texttemplate='%{text}',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

[ ]:
```
status_entrega_tempo = df_filtrado.groupby([df['order date (DateOrders)'].dt.
 ↪to_period('M'), 'Delivery Status']).size()

status_entrega_tempo = status_entrega_tempo.unstack(fill_value=0)
status_entrega_tempo.index = status_entrega_tempo.index.to_timestamp()

fig = px.line(status_entrega_tempo, title='Status de Entrega ao Longo do Tempo',
                  labels={'value': 'Quantidade de Pedidos', 'order date␣
 ↪(DateOrders)': 'Data', 'Delivery Status': 'Status de Entrega'})

fig.update_layout(width=1000, height=500)
fig.show()
```

[ ]:
```
atraso_cat = df_filtrado.groupby(['Category Name', 'Delivery Status']).size().
 ↪unstack(fill_value=0)
atraso_cat = atraso_cat.reset_index()

# Removendo a coluna Delivery Status
#atraso_cat.columns = ['Category Name', 'Advance shipping', 'Late delivery',␣
 ↪'Shipping canceled', 'Shipping on time']
atraso_cat.columns = ['Category Name', 'Advance shipping', 'Late delivery',␣
 ↪'Shipping on time']


# Criando uma coluna chamada 'Proporção de entregas em atraso'
atraso_cat['Proporção de entregas em atraso'] = (atraso_cat['Late delivery']/
 ↪(atraso_cat['Late delivery']+atraso_cat['Advance␣
 ↪shipping']+atraso_cat['Shipping on time'])).round(3)

# Ordenando o DataFrame pelo 'Proporção de entregas em atraso'
atraso_cat.sort_values(by='Proporção de entregas em atraso', ascending=False,␣
 ↪inplace=True)

atraso_cat = atraso_cat.reset_index(drop=True)

atraso_cat
```

Pelo dataframe acima pode-se concluir que, com exceção de uma categoria (**Men's Golf Clubs**), em **todos as outras** a entregas em atraso **superam** a soma das entregas no prazo, canceladas e antes do prazo.

7.4 Análise dos status de pedido

[ ]:
```
# Obter a contagem para cada status de entrega

status_pedido = df_filtrado['Order Status'].value_counts()
status_pedido_df = status_pedido.reset_index()
status_pedido_df.columns = ['Status de Pedido', 'Quantidade de Pedidos']


fig = px.bar(status_pedido_df, x='Status de Pedido', y='Quantidade de Pedidos',␣
 ↪color='Status de Pedido',
                 title="Quantidade de pedidos por status de pedido", text =␣
 ↪'Quantidade de Pedidos')

fig.update_traces(textposition='outside', texttemplate='%{text}',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

[ ]:
```
status_pedido_tempo = df_filtrado.groupby([df['order date (DateOrders)'].dt.
 ↪to_period('M'), 'Order Status']).size()

status_pedido_tempo = status_pedido_tempo.unstack(fill_value=0)
status_pedido_tempo.index = status_pedido_tempo.index.to_timestamp()

fig = px.line(status_pedido_tempo, title='Status de Pedido ao Longo do Tempo',
                  labels={'value': 'Quantidade de Pedidos', 'order date␣
 ↪(DateOrders)': 'Data', 'Order Status': 'Status de Pedido'})

fig.update_layout(width=1000, height=500)
fig.show()
```

7.5 Análise de pedidos com entrega cancelada

[ ]:
```
entrega_cancelada_df = df[df['Delivery Status']=='Shipping canceled']
entrega_cancelada_df = entrega_cancelada_df['Order Status'].value_counts()

entrega_cancelada_df = entrega_cancelada_df.reset_index()
entrega_cancelada_df.columns = ['Motivo do Cancelamento', 'Quantidade de␣
 ↪Pedidos']

entrega_cancelada_df['Motivo do Cancelamento'] = entrega_cancelada_df['Motivo␣
 ↪do Cancelamento'].replace({'SUSPECTED_FRAUD': 'POSSIVEL FRAUDE', 'CANCELED':␣
 ↪'CANCELADO PELO USUARIO'})
entrega_cancelada_df
```

[ ]:
```
fig = px.bar(entrega_cancelada_df, x='Motivo do Cancelamento', y='Quantidade de␣
↪Pedidos', color='Motivo do Cancelamento',
                title="Pedidos Com Entrega Cancelada", text = 'Quantidade de␣
↪Pedidos')

fig.update_traces(textposition='outside', texttemplate='%{text}',␣
↪textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

Percebe-se que, dos pedidos que tiveram entrega cancelada, ou foram cancelados pelo próprio cliente ou por suspeita de fraude a própria empresa cancelou.

[ ]:
```
operacoes_fraude = df[df['Order Status']=='SUSPECTED_FRAUD']
operacoes_fraude.head()
```

7.6 Análise de fraudes em operações

Fraudes em produtos

[ ]:
```
fraude_prod = operacoes_fraude.groupby('Product Name').size().
↪reset_index(name='Compras Fraudulentas')
fraude_prod.sort_values(by='Compras Fraudulentas', ascending=False,␣
↪inplace=True)

# Renomeando colunas
fraude_prod.rename(columns={'Product Name': 'Nome do Produto'}, inplace=True)

fraude_prod
```

[ ]:
```
fig = px.bar(fraude_prod.head(10), x='Nome do Produto', y='Compras␣
↪Fraudulentas', color='Nome do Produto',
                title="Top 10 produtos com mais compras fraudulentas", text =␣
↪'Compras Fraudulentas')

fig.update_traces(textposition='outside', texttemplate='%{text}',␣
↪textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

Fraudes em categorias

[ ]:
```
fraude_cat = operacoes_fraude.groupby('Category Name').size().
↪reset_index(name='Compras Fraudulentas')
fraude_cat.sort_values(by='Compras Fraudulentas', ascending=False, inplace=True)

# Renomeando colunas
fraude_cat.rename(columns={'Category Name': 'Nome da Categoria'}, inplace=True)
```

```
fraude_cat
```

[ ]:
```
fig = px.bar(fraude_cat.head(10), x='Nome da Categoria', y='Compras␣
↪Fraudulentas', color='Nome da Categoria',
                title="Top 10 categorias com mais compras fraudulentas", text␣
↪= 'Compras Fraudulentas')

fig.update_traces(textposition='outside', texttemplate='%{text}',␣
↪textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

Fraudes em departamentos

[ ]:
```
fraude_dep = operacoes_fraude.groupby('Department Name').size().
↪reset_index(name='Compras Fraudulentas')
fraude_dep.sort_values(by='Compras Fraudulentas', ascending=False, inplace=True)

# Renomeando colunas
fraude_dep.rename(columns={'Department Name': 'Departamento'}, inplace=True)

fraude_dep
```

[ ]:
```
fig = px.bar(fraude_dep.head(10), x='Departamento', y='Compras Fraudulentas',␣
↪color='Departamento',
                title="Compras fraudulentas por departamento", text = 'Compras␣
↪Fraudulentas')

fig.update_traces(textposition='outside', texttemplate='%{text}',␣
↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

Fraudes por hora

[ ]:
```
# Criando a coluna target
df['target'] = df['Order Status'].apply(lambda x: 1 if x == 'SUSPECTED_FRAUD'␣
↪else 0)

# Agrupando por hora e status de fraude e contando as ocorrências
fraude_por_hora = df.groupby([df['order date (DateOrders)'].dt.hour.
↪rename('Hora'), 'target']).size().reset_index(name='Contagem')

# Agora, usando o Seaborn para criar um gráfico de barras
plt.figure(figsize=(14, 8))
barplot = sns.barplot(data=fraude_por_hora, x='Hora', y='Contagem',␣
↪hue='target')
```

```
# Adicionando títulos e rótulos
plt.title('Contagem de transações fraudulentas e não fraudulentas por hora do␣
 ↪dia')
plt.xlabel('Hora do Dia')
plt.ylabel('Número de Transações')

# Ajustando os rótulos da legenda
handles, labels = barplot.get_legend_handles_labels()
barplot.legend(handles=handles, title='Status da Transação', labels=['Não␣
 ↪Fraude', 'Fraude'])


plt.show()
```

Para verificar se há uma relação significativa entre a hora do dia e a ocorrência de fraudes, será realizado um **teste de hipótese**. Para tal, é escolhido o **teste do Qui-Quadrado de Independência** pois pretende-se verificar a **independência entre duas variáveis categóricas em uma tabela de contingência**. Esse teste compara as contagens observadas de cada combinação de categorias com as contagens que seriam esperadas se as variáveis fossem independentes. Se houver uma diferença significativa entre as contagens observadas e as esperadas, o teste indicará uma possível associação entre as variáveis.

H0: Não há associação entre as variáveis categóricas hora e fraude

HA: Há associação entre as variáveis categóricas hora e fraude

É definido um nível de significância (alpha) de 0.05 para o teste.

[ ]:
```
# Criando a tabela de contingência
tabela_contingencia = pd.crosstab(df['order date (DateOrders)'].dt.hour.
 ↪rename('Hora'), df['target'])

# Realizar o teste do qui-quadrado
chi2, p, dof, expected = chi2_contingency(tabela_contingencia)

print(f'O p-valor obtido foi de: {p:.6f}')

# Interpretação dos resultados
if p < 0.05:
    print("Rejeitamos a hipótese nula. Há uma relação significativa entre␣
 ↪'Hora' e 'target'.")
else:
    print("Não rejeitamos a hipótese nula. 'Hora' e 'target' parecem ser␣
 ↪independentes.")
```

Fraudes por dia

[ ]:
```
# Agrupando por dia e status de fraude e contando as ocorrências
fraude_por_dia = df.groupby([df['order date (DateOrders)'].dt.day.
 ↪rename('Dia'), 'target']).size().reset_index(name='Contagem')

# Agora, usando o Seaborn para criar um gráfico de barras
plt.figure(figsize=(14, 8))
barplot = sns.barplot(data=fraude_por_dia, x='Dia', y='Contagem', hue='target')

# Adicionando títulos e rótulos
plt.title('Contagem de transações fraudulentas e não fraudulentas por dia do␣
 ↪mês')
plt.xlabel('Dia do Mês')
plt.ylabel('Número de Transações')

# Ajustando os rótulos da legenda
handles, labels = barplot.get_legend_handles_labels()
barplot.legend(handles=handles, title='Status da Transação', labels=['Não␣
 ↪Fraude', 'Fraude'])


plt.show()
```

Para verificar se há uma relação significativa entre o dia do mês e a ocorrência de fraudes, será realizado um teste de hipótese:

H0: Não há associação entre as variáveis categóricas dia e fraude

HA: Há associação entre as variáveis categóricas dia e fraude

É definido um nível de significância (alpha) de 0.05 para o teste.

[ ]:
```
# Criando a tabela de contingência
tabela_contingencia = pd.crosstab(df['order date (DateOrders)'].dt.day.
 ↪rename('Hora'), df['target'])

# Realizar o teste do qui-quadrado
chi2, p, dof, expected = chi2_contingency(tabela_contingencia)

print(f'O p-valor obtido foi de: {p:.10f}')

# Interpretação dos resultados
if p < 0.05:
    print("Rejeitamos a hipótese nula. Há uma relação significativa entre 'dia'␣
 ↪e 'target'.")
else:
    print("Não rejeitamos a hipótese nula. 'dia' e 'target' parecem ser␣
 ↪independentes.")
```

Fraudes por mês

[ ]:
```
# Agrupando por mes e status de fraude e contando as ocorrências
fraude_por_mes = df.groupby([df['order date (DateOrders)'].dt.month.
 ↪rename('Mes'), 'target']).size().reset_index(name='Contagem')

# Agora, usando o Seaborn para criar um gráfico de barras
plt.figure(figsize=(14, 8))
barplot = sns.barplot(data=fraude_por_mes, x='Mes', y='Contagem', hue='target')

# Adicionando títulos e rótulos
plt.title('Contagem de transações fraudulentas e não fraudulentas por meses do␣
 ↪ano')
plt.xlabel('Mês')
plt.ylabel('Número de Transações')

# Ajustando os rótulos da legenda
handles, labels = barplot.get_legend_handles_labels()
barplot.legend(handles=handles, title='Status da Transação', labels=['Não␣
 ↪Fraude', 'Fraude'])


plt.show()
```

Para verificar se há uma relação significativa entre o mês do ano e a ocorrência de fraudes, será realizado um teste de hipótese:

H0: Não há associação entre as variáveis categóricas mês e fraude

HA: Há associação entre as variáveis categóricas mês e fraude

É definido um nível de significância (alpha) de 0.05 para o teste.

[ ]:
```
# Criando a tabela de contingência
tabela_contingencia = pd.crosstab(df['order date (DateOrders)'].dt.month.
 ↪rename('Mes'), df['target'])

# Realizar o teste do qui-quadrado
chi2, p, dof, expected = chi2_contingency(tabela_contingencia)

print(f'O p-valor obtido foi de: {p:.10f}')

# Interpretação dos resultados
if p < 0.05:
    print("Rejeitamos a hipótese nula. Há uma relação significativa entre 'mes'␣
 ↪e 'target'.")
else:
    print("Não rejeitamos a hipótese nula. 'mes' e 'target' parecem ser␣
 ↪independentes.")
```

7.7 Análise dos modos de entrega

[ ]:
```
# Obter a contagem para cada status de entrega

modos_entrega = df_filtrado['Shipping Mode'].value_counts()
modos_entrega_df = modos_entrega.reset_index()
modos_entrega_df.columns = ['Modo de Entrega', 'Quantidade de Pedidos']


fig = px.bar(modos_entrega_df, x='Modo de Entrega', y='Quantidade de Pedidos',␣
 ↪color='Modo de Entrega',
                title="Quantidade de pedidos por modo de envio", text =␣
 ↪'Quantidade de Pedidos')

fig.update_traces(textposition='outside', texttemplate='%{text}',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

[ ]:
```
modos_entrega_tempo = df_filtrado.groupby([df['order date (DateOrders)'].dt.
 ↪to_period('M'), 'Shipping Mode']).size()

modos_entrega_tempo = modos_entrega_tempo.unstack(fill_value=0)
modos_entrega_tempo.index = modos_entrega_tempo.index.to_timestamp()

fig = px.line(modos_entrega_tempo, title='Modos de Entrega ao Longo do Tempo',
                  labels={'value': 'Quantidade de Pedidos', 'order date␣
 ↪(DateOrders)': 'Data', 'Shipping Mode': 'Modo de Entrega'})
fig.update_layout(width=1000, height=500)

fig.show()
```

[ ]:
```
# Criando a coluna que mostra se a entrega foi atrasada (1) ou não (0)
df_filtrado['Atraso'] = (df_filtrado['Days for shipping (real)'] >␣
 ↪df_filtrado['Days for shipment (scheduled)']).astype(int)

# Agrupando por 'shipping mode' e contando atrasos
atraso_por_modo_de_entrega = df_filtrado.groupby('Shipping Mode')['Atraso'].
 ↪value_counts().unstack(fill_value=0)

# Calculando a porcentagem de atrasos
atraso_por_modo_de_entrega['Entregas Totais'] = atraso_por_modo_de_entrega.
 ↪sum(axis=1)
atraso_por_modo_de_entrega['Porcentagem de Atraso'] =␣
 ↪((atraso_por_modo_de_entrega[1] / atraso_por_modo_de_entrega['Entregas␣
 ↪Totais']) * 100).round(2)
```

```
# Criando o dataframe final
final_df = atraso_por_modo_de_entrega.rename(columns={1: 'Entregas Atrasadas',␣
 ↪0: 'Entregas à Tempo'})
final_df = final_df[['Entregas Totais', 'Entregas Atrasadas', 'Porcentagem de␣
 ↪Atraso']].reset_index()

#final_df = final_df.set_index('Atraso')
#final_df.reset_index(drop=True, inplace=True)
```

[ ]:
```
final_df
```

[ ]:
```
fig = px.bar(final_df, x='Shipping Mode', y='Porcentagem de Atraso',␣
 ↪color='Shipping Mode',
                title="Porcentagem de atraso por modo de envio", text =␣
 ↪'Porcentagem de Atraso',
            labels={'Shipping Mode': 'Modo de Envio'})

fig.update_traces(textposition='outside', texttemplate='%{text}%',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=500)

fig.show()
```

Pelo gráfico acima, percebe-se que ocorreram com atraso:

- Todas **(100%)** as entregas do modo **First Class**.
- Quase metade **(47,71%)** das entregas do modo **Same Day**.
- Praticamente 80% **(79,81%)** das entregas do modo **Second Class**.
- Quase 40% **(39,78%)** das entregas do modo **Standard Class**.

8. Análise dos tipos de pagamento

[ ]:
```
tipo_pagamento = df_filtrado['Type'].value_counts()

# Tamanho do gráfico
plt.figure(figsize=(8,6))

# Cria um gráfico de barras com índice e contagem
barra = plt.bar(
    tipo_pagamento.index, # valor no eixo x
    tipo_pagamento.values, # valor no eixo y
    color = ['steelblue', 'lightcoral', 'olive','#ff7f50'] # cores das barras
)

# Rotulo do eixo y, letra tamanho 8
plt.ylabel('Número de pedidos', fontsize = 12)

# Titulo, letra tamanho 14
```

```
plt.title('Quantidade de pedidos por método de pagamento', fontsize = 16)

# Adicionando a contagem em cima das barras
for bar in barra:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval, int(yval), va='bottom',␣
 ↪ha='center', fontsize=13)

plt.show()
```

[ ]:
```
# Porcentagem de pedidos pagos para cada modo de pagamento
df_filtrado['Type'].value_counts(1)
```

[ ]:
```
tipo_pagamento_mensal = df_filtrado.groupby([df['order date (DateOrders)'].dt.
 ↪to_period('M'), 'Type']).size()

tipo_pagamento_mensal = tipo_pagamento_mensal.unstack(fill_value=0)
tipo_pagamento_mensal.index = tipo_pagamento_mensal.index.to_timestamp()

fig = px.line(tipo_pagamento_mensal, title='Tipos de pagamento ao longo do␣
 ↪tempo',
                  labels={'value': 'Quantidade de Pedidos', 'order date␣
 ↪(DateOrders)': 'Data', 'Type': 'Tipo de Pagamento'})

fig.update_layout(width=1000, height=500)

fig.show()
```

9. Análise dos preços e descontos

[ ]:
```
mediana_precos = df_filtrado['Order Item Product Price'].median()
mediana_descontos = df_filtrado['Order Item Discount'].median()

mediana_precos_format = (
    "{:,.0f}".format(mediana_precos)
    .replace(",", ".")
)

mediana_descontos_format = (
    "{:,.0f}".format(mediana_descontos)
    .replace(",", ".")
)

# Crie dois subplots (um para o histograma de 'preços' e outro para o␣
 ↪histograma de 'descontos')
fig = make_subplots(rows=1, cols=2, subplot_titles=('Histograma dos preços',␣
 ↪'Histograma dos descontos'))
```

```
# Adicione o histograma de 'preços' ao primeiro subplot
histogram_preços = go.Histogram(x=df_filtrado['Order Item Product Price'],␣
 ↪nbinsx=10, marker=dict(color='blue'))
fig.add_trace(histogram_preços, row=1, col=1)

# Adicione o histograma de 'descontos' ao segundo subplot
histogram_descontos = go.Histogram(x=df_filtrado['Order Item Discount'],␣
 ↪nbinsx=12, marker=dict(color='green'))
fig.add_trace(histogram_descontos, row=1, col=2)

# Adicione a linha da mediana a ambos os subplots
line_precos = go.Scatter(x=[mediana_precos, mediana_precos], y=[0, 144844],␣
 ↪mode='lines',
                              line=dict(color='lightblue', dash='dash'),
                              showlegend=True,
                              name=f"Mediana dos preços =␣
 ↪{mediana_precos_format}")
fig.add_trace(line_precos, row=1, col=1)

line_descontos = go.Scatter(x=[mediana_descontos, mediana_descontos], y=[0,␣
 ↪125593], mode='lines',
                              line=dict(color='lightgreen', dash='dash'),
                              showlegend=True,
                              name=f"Mediana dos descontos =␣
 ↪{mediana_descontos_format}")
fig.add_trace(line_descontos, row=1, col=2)

# Atualize o layout e as configurações dos subplots
fig.update_layout(title_text='Histograma dos preços e descontos dos produtos',
                  autosize=False,
                  width=1200,  # Largura total dos subplots
                  height=500)

fig.update_yaxes(range=[0, 150000], row=1, col=1)
fig.update_yaxes(range=[0, 150000], row=1, col=2)

# Mostre o gráfico
fig.show()
```

[ ]:
```
# Define a localidade para adicionar o separador de milhares
locale.setlocale(locale.LC_ALL, '')

# Cria uma figura com 2 subplots
fig, axs = plt.subplots(1, 2, figsize=(16, 9))

# Cria um boxplot para a variável 'preço' no primeiro subplot
```

```
bp1 = axs[0].boxplot(df_filtrado['Order Item Product Price'], patch_artist=True)
axs[0].set_title('Boxplot dos preços')

# Calcula a média e mediana do preço
mean_preco = np.mean(df_filtrado['Order Item Product Price'])
median_preco = np.median(df_filtrado['Order Item Product Price'])

# Define a cor do boxplot
bp1['boxes'][0].set_facecolor('lightblue')

# Adiciona a legenda da média e mediana com separador de milhares
max_rent = np.max(df_filtrado['Order Item Product Price'])
axs[0].annotate(f'Média = {locale.format_string("%.2f", mean_preco,␣
 ↪grouping=True)}\nMediana = {locale.format_string("%.2f", median_preco,␣
 ↪grouping=True)}',
                xy=(1, max_rent*0.8),
                xytext=(1.15, max_rent*0.8),
                bbox=dict(facecolor='lightblue', edgecolor='blue'),
                fontsize=10)

# Cria um boxplot para a variável 'desconto' no segundo subplot
bp2 = axs[1].boxplot(df_filtrado['Order Item Discount'], patch_artist=True)
axs[1].set_title('Boxplot dos descontos')

# Calcula a média e mediana do desconto
mean_desconto = np.mean(df_filtrado['Order Item Discount'])
median_desconto = np.median(df_filtrado['Order Item Discount'])

# Define a cor do boxplot
bp2['boxes'][0].set_facecolor('lightgreen')

# Adiciona a legenda da média e mediana com separador de milhares
max_total = np.max(df_filtrado['Order Item Discount'])
axs[1].annotate(f'Média = {locale.format_string("%.2f", mean_desconto,␣
 ↪grouping=True)}\nMediana = {locale.format_string("%.2f", median_desconto,␣
 ↪grouping=True)}',
                xy=(1, max_total*0.8),
                xytext=(1.15, max_total*0.8),
                bbox=dict(facecolor='lightgreen', edgecolor='green'),
                fontsize=10)

# Mostra os gráficos
plt.show()
```

[ ]:
```
df_filtrado['Order Item Product Price'].describe()
```

[ ]:
```
df_filtrado['Order Item Discount'].describe()
```

[ ]:
```
# Assimetria

print(f"A assimetria da distribuição dos preços assume valor:␣
 ↪{skew(df_filtrado['Order Item Product Price']):.2f}")
print(f"A assimetria da distribuição dos descontos assume valor:␣
 ↪{skew(df_filtrado['Order Item Discount']):.2f}")
```

Distribuição de preços

- A mediana Q2 está muito mais próxima de Q1 do que de Q3, indicando que a maioria dos preços assume um valor pequeno e que a distribuição possui assimetria à direita (calculada em 1.37), o que é acentuado pelos outliers superiores.

Distribuição de descontos

- A mediana Q2 está mais próxima de Q1 do que de Q3, indicando que a maioria dos descontos assume um valor pequeno e que a distribuição também possui assimetria à direita (calculada em 1.74), o que é acentuado pelos outliers superiores. Neste caso, os dados estão mais centralizados (média mais próxima da mediana) e menos dispersos (desvio padrão menor), em comparação à distribuição dos preços.

10. Análise de produtos mais vendidos e lucrativos

10.1 Análise dos produtos mais vendidos

[ ]:
```
# Calcular o valor total de vendas por item

nome_produto = df_filtrado.groupby('Product Name', as_index=False)['Sales'].
 ↪sum()

nome_produto = nome_produto.rename(columns={'Sales': 'Vendas Totais', 'Product␣
 ↪Name':'Nome do Produto'})



# Calcular o valor total de vendas de todos os itens
valor_total_vendas = df_filtrado['Sales'].sum()

# Calcular a porcentagem do valor total de vendas para cada item
nome_produto['Percentual_do_valor_de_venda'] = ((nome_produto['Vendas Totais'] /
 ↪ valor_total_vendas) * 100).round(2)

# Ordenar os itens por contribuição de valor descendente
nome_produto_ordenado = nome_produto.
 ↪sort_values('Percentual_do_valor_de_venda', ascending=False)

# Calcular a porcentagem acumulada do valor total
nome_produto_ordenado['Percentual_acumulado'] =␣
 ↪nome_produto_ordenado['Percentual_do_valor_de_venda'].cumsum()
```

```
nome_produto_ordenado.head(20)
```

[ ]:
```
# Categorizando os produtos

nome_produto_ordenado['Categoria'] = 'C'
nome_produto_ordenado.loc[nome_produto_ordenado['Percentual_acumulado'] <= 80,␣
 ↪'Categoria'] = 'A'
nome_produto_ordenado.loc[(nome_produto_ordenado['Percentual_acumulado'] > 80)␣
 ↪& (nome_produto_ordenado['Percentual_acumulado'] <= 95), 'Categoria'] = 'B'

nome_produto_ordenado['Vendas Totais (em milhoes)'] =␣
 ↪(nome_produto_ordenado['Vendas Totais']/1000000).round(3)
nome_produto_ordenado = nome_produto_ordenado.drop('Vendas Totais', axis=1)

nome_produto_ordenado.head(20)
```

[ ]:
```
fig = px.bar(nome_produto_ordenado[nome_produto_ordenado['Categoria'] == 'A'],␣
 ↪x='Nome do Produto', y='Vendas Totais (em milhoes)', color='Nome do Produto',
                 title="Produtos que correspondem a 80% do valor total de␣
 ↪vendas", text = 'Vendas Totais (em milhoes)')

fig.update_traces(textposition='outside', texttemplate='%{text}MM',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

Os produtos do gráfico acima correspondem a 80% das vendas (em valores financeiros) da DataCo Global:

- Field & Stream Sportsman 16 Gun Fire Safe
- Perfect Fitness Perfect Rip Deck

- Diamondback Women's Serene Classic Comfort Bi
- Nike Men's Free 5.0+ Running Shoe
- Nike Men's Dri-FIT Victory Golf Polo
- Pelican Sunstream 100 Kayak

- Nike Men's CJ Elite 2 TD Football Cleat

[ ]:
```
# Juntando todos os nomes de produtos em uma única string
produtos_venda = " ".join(nome_produto for nome_produto in df['Product Name'])

# gerando a nuvem de palavras
wordcloud = WordCloud(background_color="white").generate(produtos_venda)

# plotando a nuvem de palavras
plt.figure(figsize=(10, 5))
```

```
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Com base nas informações encontradas, a DataCo Global poderia adotar as seguintes ações:

Campanhas Direcionadas: Concentrar esforços de marketing e vendas nos produtos mais lucrativos, utilizando campanhas direcionadas para impulsionar ainda mais as vendas desses produtos.

Pacotes e Promoções: Criar pacotes de produtos ou promoções que incluam produtos populares junto com itens menos vendidos para aumentar a exposição destes últimos.

10.2 Análise dos produtos mais lucrativos

[ ]:
```
# Calcular o valor total de lucro por item

nome_produto_lucro = df_filtrado.groupby('Product Name', as_index=False)['Order␣
↪Profit Per Order'].sum()

nome_produto_lucro = nome_produto_lucro.rename(columns={'Order Profit Per␣
↪Order': 'Lucros Totais', 'Product Name':'Nome do Produto'})


# Calcular o valor total de lucros de todos os itens
valor_total_lucros = df_filtrado['Order Profit Per Order'].sum()

# Calcular a porcentagem do valor total de lucros para cada item
nome_produto_lucro['Percentual_do_valor_de_lucro'] =␣
↪((nome_produto_lucro['Lucros Totais'] / valor_total_lucros) * 100).round(2)

# Ordenar os itens por contribuição de valor descendente
nome_produto_lucro_ordenado = nome_produto_lucro.
↪sort_values('Percentual_do_valor_de_lucro', ascending=False)

# Calcular a porcentagem acumulada do valor total
nome_produto_lucro_ordenado['Percentual_acumulado'] =␣
↪nome_produto_lucro_ordenado['Percentual_do_valor_de_lucro'].cumsum()

nome_produto_lucro_ordenado.head(20)
```

[ ]:
```
# Categorizando os produtos

nome_produto_lucro_ordenado['Categoria'] = 'C'
nome_produto_lucro_ordenado.
↪loc[nome_produto_lucro_ordenado['Percentual_acumulado'] <= 80, 'Categoria']␣
↪= 'A'
```

```
nome_produto_lucro_ordenado.
 ↪loc[(nome_produto_lucro_ordenado['Percentual_acumulado'] > 80) &␣
 ↪(nome_produto_lucro_ordenado['Percentual_acumulado'] <= 95), 'Categoria'] =␣
 ↪'B'

nome_produto_lucro_ordenado['Lucros Totais (em milhares)'] =␣
 ↪(nome_produto_lucro_ordenado['Lucros Totais']/1000).round(3)
nome_produto_lucro_ordenado = nome_produto_lucro_ordenado.drop('Lucros Totais',␣
 ↪axis=1)

nome_produto_lucro_ordenado.head(20)
```

[ ]:
```
fig = px.
 ↪bar(nome_produto_lucro_ordenado[nome_produto_lucro_ordenado['Categoria'] ==␣
 ↪'A'], x='Nome do Produto', y='Lucros Totais (em milhares)', color='Nome do␣
 ↪Produto',
                 title="Produtos que correspondem a 80% do valor total do␣
 ↪lucro", text = 'Lucros Totais (em milhares)')

fig.update_traces(textposition='outside', texttemplate='%{text}k',␣
 ↪textfont_size=12)
fig.update_layout(width=1000, height=600)
fig.show()
```

Os produtos do gráfico acima correspondem a 80% das vendas (em valores financeiros) da DataCo Global:

- Field & Stream Sportsman 16 Gun Fire Safe
- Perfect Fitness Perfect Rip Deck

- Diamondback Women's Serene Classic Comfort Bi
- Nike Men's Free 5.0+ Running Shoe
- Nike Men's Dri-FIT Victory Golf Polo
- Pelican Sunstream 100 Kayak

- O'Brien Men's Neoprene Life Vest

11. Modelagem - Previsão de Entregas em Atraso

Objetivo: Prever se um pedido será entregue com atraso, o que é importante para a gestão de expectativas dos clientes e planejamento logístico.

[ ]:
```
df_filtrado.info()
```

[ ]:
```
# Criando a coluna target
df_filtrado['target'] = (df_filtrado['Days for shipping (real)'] >␣
 ↪df_filtrado['Days for shipment (scheduled)']).astype(int)
```

```
[ ]: # Removendo as colunas desnecessárias

colunas_para_remover = ['Days for shipping (real)', 'Days for shipment␣
 ↪(scheduled)', 'Category Name', 'Customer Email', 'Customer Fname',
                        'Customer Id', 'Customer Lname', 'Customer Password',␣
 ↪'Customer Street', 'Customer Zipcode',
                        'Department Name', 'Latitude', 'Longitude', 'Order␣
 ↪Customer Id', 'Order Id',
                        'Order Item Cardprod Id', 'Order Item Id', 'Order Profit␣
 ↪Per Order', 'Order Zipcode', 'Product Card Id',
                        'Product Description', 'Product Image', 'Product Price',␣
 ↪'Product Status', 'shipping date (DateOrders)',
                        'Late_delivery_risk', 'Delivery Status', 'Profit␣
 ↪Margin', 'Atraso', 'order date (DateOrders)']

df_filtrado.drop(columns=colunas_para_remover, inplace=True)

df_filtrado.head()
```

```
[ ]: # Porcentagem pedidos atrasados (1) e não atrasados (0) em todo o dataset
df_filtrado['target'].value_counts(1)
```

```
[ ]: # class weight
weights = df_filtrado.target.value_counts(1)[0]/df_filtrado.target.
 ↪value_counts(1)[1]
```

```
[ ]: # Divisão em X e y
X = df_filtrado.drop(columns=['target'], axis = 1)
y = df_filtrado.target
```

```
[ ]: # Divisão em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3,␣
 ↪random_state = 42, stratify=y)
```

```
[ ]: # Porcentagem pedidos atrasados (1) e não atrasados (0) no conjunto de treino
y_train.value_counts(1)
```

```
[ ]: # Porcentagem pedidos atrasados (1) e não atrasados (0) no conjunto de teste
y_test.value_counts(1)
```

```
[ ]: # Instanciando modelo XGBoost
modelo_XGBoost = XGBClassifier(n_estimators = 1000, max_depth = 8,␣
 ↪learning_rate = 1e-3, n_jobs =-1, random_state = 0,␣
 ↪scale_pos_weight=weights, eval_metric='error')

# Instanciando modelo LightGBM
```

```
modelo_LightGBM = LGBMClassifier(n_estimators = 1000, max_depth = 8, num_leaves␣
 ↪= 2^8, learning_rate = 1e-3, n_jobs =-1, random_state = 0,␣
 ↪is_unbalance=True, verbose=-1)

# Instanciando modelo catboost
modelo_CatBoost = CatBoostClassifier(n_estimators = 1000, max_depth = 8,␣
 ↪learning_rate = 1e-3, random_state = 0, scale_pos_weight = weights, verbose␣
 ↪= 0)

# Instanciando o modelo Balanced Random Forest
modelo_BRandom_Forest = BalancedRandomForestClassifier(n_estimators = 1000,␣
 ↪max_depth = 8, random_state = 0, verbose = 0)
```

2 Construção de uma função de validação cruzada - Stratified K-Fold

[ ]:
```
# Função para aplicação da validação cruzada para obtenção das métricas dos␣
 ↪modelos

def validacao_cruzada(X, y, modelo, k, threshold):

    # Inicializando a função StratifiedKFold
    folds = StratifiedKFold(n_splits=k, shuffle=True, random_state=40)

    # Criando listas para armazenar os valores de precisão, revocação,␣
 ↪acurácia, medida-F1, precision_recall_auc e roc_auc
    # em cada fold

    precisoes = list()
    revocacoes=list()
    acuracias=list()
    Medida_F1=list()
    precision_recall_auc=list()
    rocs_auc=list()
    cm_total = np.zeros((2, 2))

    # Será aplicado o método "split" no objeto folds, que retornará uma lista
    # com os índices das instâncias que pertencem ao conjunto de treino e
    # outra com os índices das instâncias que pertencem ao conjunto de teste

    for k, (train_index, test_index) in enumerate(folds.split(X,y)):
        print("=-"*6 + f"Fold: {k+1}" + "-="*6)

        # Dividindo os dados em treino e teste para cada um dos folds
        X_train_intern, y_train_intern = X.iloc[train_index, :], y.
 ↪iloc[train_index]
```

```
        X_test_intern, y_test_intern = X.iloc[test_index, :], y.iloc[test_index]

        # train_index e test_index: São os índices das instâncias do conjunto
        # de treino e teste, respectivamente, selecionados em cada um dos folds

        ##########################################
        ############## Preprocessing #############
        ##########################################

        # Instanciando o CatBoost Encoder
        encoder = CatBoostEncoder()

        # Criando um imputer para preencher com a moda os valores faltantes de␣
↪variáveis categóricas
        cat_imputer = SimpleImputer(strategy='most_frequent')

        # Criando um imputer para preencher com a mediana os valores faltantes␣
↪de variáveis numéricas
        num_imputer = SimpleImputer(strategy='median')

        # Criando pipelines para variáveis categóricas e numéricas que preenche␣
↪os valores faltantes
        cat_pipeline = Pipeline([('encoder', encoder), ('imputer',␣
↪cat_imputer)])
        num_pipeline = Pipeline([('imputer', num_imputer)])

        # Identifica as variáveis categóricas e numéricas
        cat_cols = X_train_intern.select_dtypes(include=['object']).columns
        num_cols = X_train_intern.select_dtypes(exclude=['object']).columns

        # Aplicando os pipelines no conjunto de treinamento para preencher␣
↪valores faltantes em colunas categóricas e numéricas
        X_train_intern[cat_cols] = cat_pipeline.
↪fit_transform(X_train_intern[cat_cols], y_train_intern)
        X_train_intern[num_cols] = num_pipeline.
↪fit_transform(X_train_intern[num_cols])

        # Aplicando os pipelines ao conjunto de teste para preencher valores␣
↪faltantes em colunas categóricas e numéricas
        X_test_intern[cat_cols] = cat_pipeline.
↪transform(X_test_intern[cat_cols])
        X_test_intern[num_cols] = num_pipeline.
↪transform(X_test_intern[num_cols])

        # Treinando o modelo
        modelo.fit(X_train_intern, y_train_intern)
```

```
        # Obtendo as probabilidades de cada registro pertencer a classe 1
        y_pred_proba = modelo.predict_proba(X_test_intern)[:, 1]

        # Obtendo as previsões do modelo
        y_pred = np.where(y_pred_proba > threshold, 1, 0)

        # Calculando a precisão e revocação para determinar a␣
↪precision_recall_auc
        precisao, revocacao, limiares = precision_recall_curve(y_test_intern,␣
↪y_pred)

        # Calculando a matriz de confusão do fold
        cm_total += confusion_matrix(y_test_intern, y_pred)

        # Determinando as métricas para cada fold
        precisao_revocacao_auc = auc(revocacao, precisao)
        roc_auc = roc_auc_score(y_test_intern, y_pred)
        acuracia_score = accuracy_score(y_test_intern, y_pred)
        precisao_score = precision_score(y_test_intern, y_pred)
        revocacao_score = recall_score(y_test_intern, y_pred)
        f1score = f1_score(y_test_intern, y_pred)

        # Armazenando as métricas nas listas criadas
        precisoes.append(precisao_score)
        revocacoes.append(revocacao_score)
        precision_recall_auc.append(precisao_revocacao_auc)
        rocs_auc.append(roc_auc)
        acuracias.append(acuracia_score)
        Medida_F1.append(f1score)

        # Exibindo as métricas para cada um dos folds
        print(f"Precisão: {precisao_score:.4f}")
        print(f"Revocação: {revocacao_score:.4f}")
        print(f"Acurácia: {acuracia_score:.4f}")
        print(f"Medida F1: {f1score:.4f}")
        print(f"Precision-Recall AUC: {precisao_revocacao_auc:.4f}")
        print(f"ROC AUC: {roc_auc:.4f}")

    # Transformando as listas em arrays para fazer operações matemáticas
    precisoes = np.array(precisoes)
    revocacoes = np.array(revocacoes)
    precision_recall_auc = np.array(precision_recall_auc)
    rocs_auc = np.array(rocs_auc)
    acuracias = np.array(acuracias)
    Medida_F1 = np.array(Medida_F1)
```

```
    # Calculando as médias das métricas
    media_revocacao = np.mean(revocacoes)
    media_precisao = np.mean(precisoes)
    media_acuracia = np.mean(acuracias)
    media_F1 = np.mean(Medida_F1)
    media_pr_AUC = np.mean(precision_recall_auc)
    media_roc_AUC = np.mean(rocs_auc)

    # Calculando os desvios padrão para cada métrica
    std_revocacao = np.std(revocacoes)
    std_precisao = np.std(precisoes)
    std_acuracia = np.std(acuracias)
    std_F1 = np.std(Medida_F1)
    std_pr_AUC = np.std(precision_recall_auc)
    std_roc_AUC = np.std(rocs_auc)

    # Exibindo as médias das métricas obtidas
    print()
    print("=-"*6 + "Exibindo a média das métricas obtidas" + "-="*6)
    print(f"Média da acurácia: {media_acuracia:.4f} +/- {std_acuracia:.4f}")
    print(f"Média da revocação: {media_revocacao:.4f} +/- {std_revocacao:.4f}")
    print(f"Média da precisão: {media_precisao:.4f} +/- {std_precisao:.4f}")
    print(f"Média da Medida F1: {media_F1:.4f} +/- {std_F1:.4f}")
    print(f"Média da ROC AUC: {media_roc_AUC:.4f} +/- {std_roc_AUC:.4f}")
    print(f"Média da PR AUC: {media_pr_AUC:.4f} +/- {std_pr_AUC:.4f}")

    # Plotando a matriz de confusão agregada com heatmap
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm_total, annot=True, fmt=".0f", cmap="Blues")
    plt.title("Matriz de Confusão Agregada de Todos os Folds")
    plt.ylabel('Verdadeiro')
    plt.xlabel('Previsto')
    plt.show()
```

3 Modelo LightGBM

[ ]:
```
validacao_cruzada(X, y, modelo_LightGBM, k = 5, threshold = 0.5)
```

4 Modelo XGBoost

[ ]:
```
validacao_cruzada(X, y, modelo_XGBoost, k = 5, threshold = 0.5)
```

5 Modelo CatBoost

```
[ ]: validacao_cruzada(X, y, modelo_CatBoost, k = 5, threshold = 0.5)
```

6 Modelo Balanced Random Forest

```
[ ]: validacao_cruzada(X, y, modelo_BRandom_Forest, k = 5, threshold = 0.5)
```

Dentre todas as métricas de avaliação, será dada prioridade para a **ROC AUC**, que mede a capacidade do modelo distinguir entre entregas atrasadas e não atrasadas. Caso o custo de uma entrega em atraso não identificada previamente (falso negativo) seja alto, esta métrica se torna uma grande arma para **reduzir custos operacionais e aumentar o lucro da empresa**.

6.1 Possíveis impactos de constantes atrasos em entrega

1) Satisfação do Cliente

- **Expectativa dos clientes**: Atrasos não identificados e não comunicados podem levar à insatisfação do cliente, quebra de confiança e potencial perda de clientes.
- **Reputação online**: Uma entrega atrasada pode levar a avaliações negativas em redes sociais, podendo causar danos à reputação da empresa perante potenciais clientes.

2) Custos financeiros diretos

- **Custos de Remessa e Logística**: Atrasos podem aumentar os custos de logística, especialmente se forem necessárias medidas corretivas, como reexpedição ou entrega expressa.
- **Penalidades Contratuais**: Em alguns casos, atrasos nas entregas podem levar a multas ou penalidades contratuais, especialmente em negócios B2B, onde os contratos são mais rigorosos.

3) Implicações de Longo Prazo

- **Relações com Clientes B2B**: No caso de clientes empresariais, os atrasos nas entregas podem interromper suas operações, prejudicando a relação comercial de longo prazo, consequentemente as fontes de receita ao longo prazo.
- **Efeito Cascata na Cadeia de Suprimentos**: Atrasos em um nó da cadeia podem afetar outros, especialmente em um modelo just-in-time, onde os produtos são produzidos ou entregues exatamente quando necessários.

4) Estratégias Competitivas

- **Perda de Vantagem Competitiva**: Em um mercado global, a capacidade de entregar no prazo pode ser um diferencial competitivo. Atrasos frequentes podem abrir margem para o crescimento de concorrentes.
- **Perda de Mercado**: Clientes insatisfeitos podem se voltar para concorrentes com histórico de entregas mais confiáveis.

Sendo assim, considerando o impacto financeiro e operacional dos atrasos em entregas, **é fundamental** para uma empresa de supply chain como a DataCo Global, com presença global, **ter uma logística capaz de cumprir com prazos e metas**.

Na modelagem de previsão, a priorização da ROC AUC é uma estratégia-chave para isso. Com uma ROC AUC elevada, a empresa consegue melhor classificar entre pedidos com alta probabilidade de

atraso e aqueles que provavelmente serão entregues no prazo, permitindo-lhes tomar medidas para **mitigar os impactos negativos**. Isso pode incluir desde a **comunicação antecipada com os clientes** sobre potenciais atrasos, **reajustes na logística** para acelerar entregas subsequentes, ou alterações na gestão do estoque para lidar com possíveis interrupções na cadeia de suprimentos.

Dentre os modelos escolhidos para avaliar a métrica **ROC AUC**, o XGBoost foi o que melhor performou, com uma ROC AUC média de **0.7344**, ou seja, significa que existe 73,44% de chance de que o modelo classifique corretamente um pedido aleatório atrasado como **atrasado** e um pedido aleatório não atrasado como **não atrasado**.

Como o LightGBM atingiu uma ROC AUC praticamente idêntica a do XGBoost e, possui um processamento mais rápido que o XGBoost, será escolhido o LightGBM para passar por um processo de **tunagem de hiperparâmetros**.

7 Feature Selection

[ ]:
```
# Inicializando o RFE
rfe = RFE(estimator = modelo_LightGBM, n_features_to_select = 20, step = 1)

encoder = CatBoostEncoder()
# Ajustar e transformar os dados de treinamento
X_train_encoded = encoder.fit_transform(X_train, y_train)

# Transformar os dados de teste
X_test_encoded = encoder.transform(X_test)

# Treinando o RFE
rfe.fit(X_train_encoded, y_train)

# Obtendo as features selecionadas
features_importantes = np.array(list(X_train_encoded.columns))[rfe.support_]

features_importantes
```

[ ]:
```
validacao_cruzada(X[features_importantes], y, modelo_LightGBM, k = 5, threshold␣
↪= 0.5)
```

Neste caso, realizando uma feature selection e excluindo as 5 colunas menos importantes para o modelos, atingimos o mesmo resultado que com o dataset original. Assim, por questões de processamento, é preferível manter o dataset com as 5 colunas a menos.

8 Tunagem de Hiperparâmetros

[ ]:
```
def tunagem_hiperparametros(trial, k = 5, threshold = 0.5):

    # Parâmetros para serem tunados
    learning_rate = trial.suggest_float('learning_rate', 1e-3, 1e-1, log=True)
    max_depth = trial.suggest_int('max_depth', 1, 20)
```

```
    subsample = trial.suggest_float('subsample', 0.5, 1, step = 0.1)
    colsample_bytree = trial.suggest_float('colsample_bytree', 0.5, 1, step = 0.
↪1)
    min_child_samples = trial.suggest_int('min_child_samples', 1, 20)
    min_child_weight = trial.suggest_float('min_child_weight', 1e-3, 1e-1)

    # Inicializando a função StratifiedKFold
    folds = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)

    # Criando listas para armazenar os valores de precisão, revocação,␣
↪acurácia, medida-F1, precision_recall_auc e roc_auc
    # em cada fold
    precisoes = list()
    revocacoes=list()
    acuracias=list()
    Medida_F1=list()
    precision_recall_auc=list()
    rocs_auc=list()

    # Será aplicado o método "split" no objeto folds, que retornará uma lista
    # com os índices das instâncias que pertencem ao conjunto de treino e
    # outra com os índices das instâncias que pertencem ao conjunto de teste

    for k, (train_index, test_index) in enumerate(folds.
↪split(X[features_importantes],y)):
        print("=-"*6 + f"Fold: {k+1}" + "-="*6)

        # Dividindo os dados em treino e teste para cada um dos folds
        X_train_intern, y_train_intern = X[features_importantes].
↪iloc[train_index, :], y.iloc[train_index]
        X_test_intern, y_test_intern = X[features_importantes].iloc[test_index,␣
↪:], y.iloc[test_index]

        # train_index e test_index: São os índices das instâncias do conjunto
        # de treino e teste, respectivamente, selecionados em cada um dos folds

        ##############################################
        ############### Preprocessing ################
        ##############################################

        # Instanciando o CatBoost Encoder
        encoder = CatBoostEncoder()

        # Criando um imputer para preencher com a moda os valores faltantes de␣
↪variáveis categóricas
        cat_imputer = SimpleImputer(strategy='most_frequent')
```

```
      # Criando um imputer para preencher com a mediana os valores faltantes␣
↪de variáveis numéricas
      num_imputer = SimpleImputer(strategy='median')

      # Criando pipelines para variáveis categóricas e numéricas que preenche␣
↪os valores faltantes
      cat_pipeline = Pipeline([('encoder', encoder), ('imputer',␣
↪cat_imputer)])
      num_pipeline = Pipeline([('imputer', num_imputer)])

      # Identifica as variáveis categóricas e numéricas
      cat_cols = X_train_intern.select_dtypes(include=['object']).columns
      num_cols = X_train_intern.select_dtypes(exclude=['object']).columns

      # Aplicando os pipelines no conjunto de treinamento para preencher␣
↪valores faltantes em colunas categóricas e numéricas
      X_train_intern[cat_cols] = cat_pipeline.
↪fit_transform(X_train_intern[cat_cols], y_train_intern)
      X_train_intern[num_cols] = num_pipeline.
↪fit_transform(X_train_intern[num_cols])

      # Aplicando os pipelines ao conjunto de teste para preencher valores␣
↪faltantes em colunas categóricas e numéricas
      X_test_intern[cat_cols] = cat_pipeline.
↪transform(X_test_intern[cat_cols])
      X_test_intern[num_cols] = num_pipeline.
↪transform(X_test_intern[num_cols])

      # Instanciando o Modelo LightGBM
      modelo_LightGBM = LGBMClassifier(n_estimators = 1000, max_depth =␣
↪max_depth, num_leaves = 2^8, subsample = subsample,
                                       min_child_weight = min_child_weight,␣
↪min_child_samples = min_child_samples,
                                       colsample_bytree = colsample_bytree,␣
↪learning_rate = learning_rate, n_jobs =-1,
                                       random_state = 0, is_unbalance=True,␣
↪verbose=-1)

      # Treinando o modelo LightGBM
      modelo_LightGBM.fit(X_train_intern, y_train_intern)

      # Obtendo as probabilidades de cada registro pertencer a classe 1
      y_pred_proba = modelo_LightGBM.predict_proba(X_test_intern)[:, 1]

      # Obtendo as previsões do modelo
```

```
        y_pred = np.where(y_pred_proba > threshold, 1, 0)

        # Calculando a precisão e revocação para determinar a␣
↪precision_recall_auc
        precisao, revocacao, limiares = precision_recall_curve(y_test_intern,␣
↪y_pred)

        # Determinando as métricas para cada fold
        precisao_revocacao_auc = auc(revocacao, precisao)
        roc_auc = roc_auc_score(y_test_intern, y_pred)
        acuracia_score = accuracy_score(y_test_intern, y_pred)
        precisao_score = precision_score(y_test_intern, y_pred)
        revocacao_score = recall_score(y_test_intern, y_pred)
        f1score = f1_score(y_test_intern, y_pred)

        # Armazenando as métricas nas listas criadas
        precisoes.append(precisao_score)
        revocacoes.append(revocacao_score)
        precision_recall_auc.append(precisao_revocacao_auc)
        rocs_auc.append(roc_auc)
        acuracias.append(acuracia_score)
        Medida_F1.append(f1score)

    # Transformando as listas em arrays para fazer operações matemáticas
    precisoes = np.array(precisoes)
    revocacoes = np.array(revocacoes)
    precision_recall_auc = np.array(precision_recall_auc)
    rocs_auc = np.array(rocs_auc)
    acuracias = np.array(acuracias)
    Medida_F1 = np.array(Medida_F1)

    # Calculando as médias das métricas
    media_revocacao = np.mean(revocacoes)
    media_precisao = np.mean(precisoes)
    media_acuracia = np.mean(acuracias)
    media_F1 = np.mean(Medida_F1)
    media_pr_AUC = np.mean(precision_recall_auc)
    media_roc_AUC = np.mean(rocs_auc)

    # Calculando os desvios padrão para cada métrica
    std_revocacao = np.std(revocacoes)
    std_precisao = np.std(precisoes)
    std_acuracia = np.std(acuracias)
    std_F1 = np.std(Medida_F1)
    std_pr_AUC = np.std(precision_recall_auc)
    std_roc_AUC = np.std(rocs_auc)
```

```
    return media_roc_AUC


study = opt.create_study(direction='maximize')
study.optimize(tunagem_hiperparametros, n_trials = 20)
```

[ ]:
```
# Melhores parâmetros obtidos do último Trial
params = {'learning_rate': 0.06672510713241127, 'max_depth': 16, 'subsample': 1.
 ↪0, 'colsample_bytree': 0.6, 'min_child_samples': 12, 'min_child_weight': 0.
 ↪034638755930084086}
```

[ ]:
```
# LightGBM executado para os melhores parâmetros
modelo_LightGBM = LGBMClassifier(n_estimators = 1000, num_leaves = 2^8, n_jobs␣
 ↪=-1, random_state = 0, is_unbalance=True, **params, verbose=-1)
```

[ ]:
```
# Métricas do LightGBM utilizando os melhores parâmetros
validacao_cruzada(X, y, modelo_LightGBM, k = 5, threshold = 0.5)
```

Após 20 iterações de tunagem de hiperparâmetros utilizando uma **Bayesian Search**, a ROC AUC média do modelo LightGBM melhorou pouco, de **73,32%** para **73,41%**. Sendo assim, tornou-se ainda melhor na identificação de pedidos com potencial entrega atrasada, ajudando a DataCo Global na identiicação precoce destes casos, contribuindo para **mitigar os riscos decorrentes de problemas de logística**.

Próximos passos: para melhorar ainda mais o desempenho do modelo, pode-se implementar técnicas de *feature engineering*, ou seja, criando outras variáveis à partir das já existentes, para buscar encontrar fatores que influenciem positivamente o modelo.

12. Modelagem - Previsão de Fraudes

Objetivo: Prever se um pedido poderá ser identificado como fraude, que é importante para evitar perdas financeiras à empresa.

[ ]:
```
df.info()
```

[ ]:
```
# Criando a coluna target
df['target'] = df['Order Status'].apply(lambda x: 1 if x == 'SUSPECTED_FRAUD'␣
 ↪else 0)
df.head()
```

[ ]:
```
# Removendo as colunas desnecessárias

colunas_para_remover = ['Days for shipping (real)', 'Days for shipment␣
 ↪(scheduled)', 'Category Name', 'Customer Email', 'Customer Fname',
                        'Customer Id', 'Customer Lname', 'Customer Password',␣
 ↪'Customer Street', 'Customer Zipcode',
                        'Department Name', 'Latitude', 'Longitude', 'Order␣
 ↪Customer Id', 'Order Id',
```

```
                      'Order Item Cardprod Id', 'Order Item Id', 'Order Profit␣
 ↪Per Order', 'Order Zipcode', 'Product Card Id',
                      'Product Description', 'Product Image', 'Product␣
 ↪Status', 'shipping date (DateOrders)', 'order date (DateOrders)',
                      'Late_delivery_risk', 'Delivery Status', 'Order Status']

df.drop(columns=colunas_para_remover, inplace=True)

df.head()
```

[ ]:
```
# Porcentagem de fraude (1) e não fraude (0) em todo o dataset
df['target'].value_counts(1)
```

[ ]:
```
fraude_ou_nao = df['target'].value_counts()

# Tamanho do gráfico
plt.figure(figsize=(8,6))

# Cria um gráfico de barras com índice e contagem
barra = plt.bar(
    [0, 1], # valor no eixo x
    fraude_ou_nao.values, # valor no eixo y
    color = ['steelblue', 'lightcoral'] # cores das barras
)

# Define os rótulos do eixo x para 'Não Fraude' e 'Fraude'
plt.xticks([0, 1], ['Não Fraude', 'Fraude'])

# Rotulo do eixo y, letra tamanho 8
plt.ylabel('Número de pedidos', fontsize = 12)

# Titulo, letra tamanho 14
plt.title('Quantidade de operações fraudulentas', fontsize = 16)

#plt.xticks(fraude_ou_nao.index.astype(str), fraude_ou_nao.index.astype(str))

# Adicionando a contagem em cima das barras
for bar in barra:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval, int(yval), va='bottom',␣
 ↪ha='center', fontsize=13)

plt.grid(False)

plt.show()
```

Percebe-se que o conjunto de dados é extremamente desbalanceado, contendo 97.7% de operações não fraudulentas e apenas 2.3% de operações fraudulentas.

[ ]:
```
# class weight
weights = df.target.value_counts(1)[0]/df.target.value_counts(1)[1]
```

[ ]:
```
# Divisão em X e y
X = df.drop(columns=['target'], axis = 1)
y = df.target
```

[ ]:
```
# Divisão em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3,␣
 ↪random_state = 42, stratify=y)
```

[ ]:
```
# Porcentagem de fraude (1) e não fraude (0) no conjunto de treino
y_train.value_counts(1)
```

[ ]:
```
# Porcentagem de fraude (1) e não fraude (0) no conjunto de teste
y_test.value_counts(1)
```

[ ]:
```
# Instanciando modelo XGBoost
modelo_XGBoost = XGBClassifier(n_estimators = 1000, max_depth = 8,␣
 ↪learning_rate = 1e-3, n_jobs =-1, random_state = 0,␣
 ↪scale_pos_weight=weights, eval_metric='error')

# Instanciando modelo LightGBM
modelo_LightGBM = LGBMClassifier(n_estimators = 1000, max_depth = 8, num_leaves␣
 ↪= 2^8, learning_rate = 1e-3, n_jobs =-1, random_state = 0,␣
 ↪is_unbalance=True, verbose=-1)

# Instanciando modelo catboost
modelo_CatBoost = CatBoostClassifier(n_estimators = 1000, max_depth = 8,␣
 ↪learning_rate = 1e-3, random_state = 0, scale_pos_weight = weights, verbose␣
 ↪= 0)

# Instanciando o modelo Balanced Random Forest
modelo_BRandom_Forest = BalancedRandomForestClassifier(n_estimators = 1000,␣
 ↪max_depth = 8, random_state = 0, verbose = 0)

# 1) Executar os modelos com a validação cruzada
# 2) fazer uma feature engineering, no próprio código da validação cruzada
# 3) Para o melhor modelo, fazer uma feature selection (RFE que usa feature␣
 ↪importance), após o código da validação cruzada
# 4) Por fim, para este melhor modelo, fazer uma tunagem de hiperparâmetros
```

9 Construção de uma função de validação cruzada - Stratified K-Fold

[ ]:
```
# Função para aplicação da validação cruzada para obtenção das métricas dos modelos

def validacao_cruzada(X, y, modelo, k, threshold):

    # Inicializando a função StratifiedKFold
    folds = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)

    # Criando listas para armazenar os valores de precisão, revocação, acurácia, medida-F1, precision_recall_auc e roc_auc
    # em cada fold

    precisoes = list()
    revocacoes=list()
    acuracias=list()
    Medida_F1=list()
    precision_recall_auc=list()
    rocs_auc=list()
    cm_total = np.zeros((2, 2))

    # Será aplicado o método "split" no objeto folds, que retornará uma lista
    # com os índices das instâncias que pertencem ao conjunto de treino e
    # outra com os índices das instâncias que pertencem ao conjunto de teste

    for k, (train_index, test_index) in enumerate(folds.split(X,y)):
        print("=-"*6 + f"Fold: {k+1}" + "-="*6)

        # Dividindo os dados em treino e teste para cada um dos folds
        X_train_intern, y_train_intern = X.iloc[train_index, :], y.iloc[train_index]
        X_test_intern, y_test_intern = X.iloc[test_index, :], y.iloc[test_index]

        # train_index e test_index: São os índices das instâncias do conjunto
        # de treino e teste, respectivamente, selecionados em cada um dos folds

        ##############################################
        ############## Preprocessing ##############
        ##############################################

        # Instanciando o CatBoost Encoder
        encoder = CatBoostEncoder()

        # Criando um imputer para preencher com a moda os valores faltantes de variáveis categóricas
```

```
        cat_imputer = SimpleImputer(strategy='most_frequent')

        # Criando um imputer para preencher com a mediana os valores faltantes␣
↪de variáveis numéricas
        num_imputer = SimpleImputer(strategy='median')

        # Criando pipelines para variáveis categóricas e numéricas que preenche␣
↪os valores faltantes
        cat_pipeline = Pipeline([('encoder', encoder), ('imputer',␣
↪cat_imputer)])
        num_pipeline = Pipeline([('imputer', num_imputer)])

        # feature engineering

        # Identifica as variáveis categóricas e numéricas
        cat_cols = X_train_intern.select_dtypes(include=['object']).columns
        num_cols = X_train_intern.select_dtypes(exclude=['object']).columns

        # Aplicando os pipelines no conjunto de treinamento para preencher␣
↪valores faltantes em colunas categóricas e numéricas
        X_train_intern[cat_cols] = cat_pipeline.
↪fit_transform(X_train_intern[cat_cols], y_train_intern)
        X_train_intern[num_cols] = num_pipeline.
↪fit_transform(X_train_intern[num_cols])

        # Aplicando os pipelines ao conjunto de teste para preencher valores␣
↪faltantes em colunas categóricas e numéricas
        X_test_intern[cat_cols] = cat_pipeline.
↪transform(X_test_intern[cat_cols])
        X_test_intern[num_cols] = num_pipeline.
↪transform(X_test_intern[num_cols])

        # Treinando o modelo
        modelo.fit(X_train_intern, y_train_intern)

        # Obtendo as probabilidades de cada registro pertencer a classe 1
        y_pred_proba = modelo.predict_proba(X_test_intern)[:, 1]

        # Obtendo as previsões do modelo
        y_pred = np.where(y_pred_proba > threshold, 1, 0)

        # Calculando a precisão e revocação para determinar a␣
↪precision_recall_auc
        precisao, revocacao, limiares = precision_recall_curve(y_test_intern,␣
↪y_pred)
```

```
        # Calculando a matriz de confusão do fold
        cm_total += confusion_matrix(y_test_intern, y_pred)

        # Determinando as métricas para cada fold
        precisao_revocacao_auc = auc(revocacao, precisao)
        roc_auc = roc_auc_score(y_test_intern, y_pred)
        acuracia_score = accuracy_score(y_test_intern, y_pred)
        precisao_score = precision_score(y_test_intern, y_pred)
        revocacao_score = recall_score(y_test_intern, y_pred)
        f1score = f1_score(y_test_intern, y_pred)

        # Armazenando as métricas nas listas criadas
        precisoes.append(precisao_score)
        revocacoes.append(revocacao_score)
        precision_recall_auc.append(precisao_revocacao_auc)
        rocs_auc.append(roc_auc)
        acuracias.append(acuracia_score)
        Medida_F1.append(f1score)

        # Exibindo as métricas para cada um dos folds
        print(f"Precisão: {precisao_score:.4f}")
        print(f"Revocação: {revocacao_score:.4f}")
        print(f"Acurácia: {acuracia_score:.4f}")
        print(f"Medida F1: {f1score:.4f}")
        print(f"Precision-Recall AUC: {precisao_revocacao_auc:.4f}")
        print(f"ROC AUC: {roc_auc:.4f}")

    # Transformando as listas em arrays para fazer operações matemáticas
    precisoes = np.array(precisoes)
    revocacoes = np.array(revocacoes)
    precision_recall_auc = np.array(precision_recall_auc)
    rocs_auc = np.array(rocs_auc)
    acuracias = np.array(acuracias)
    Medida_F1 = np.array(Medida_F1)

    # Calculando as médias das métricas
    media_revocacao = np.mean(revocacoes)
    media_precisao = np.mean(precisoes)
    media_acuracia = np.mean(acuracias)
    media_F1 = np.mean(Medida_F1)
    media_pr_AUC = np.mean(precision_recall_auc)
    media_roc_AUC = np.mean(rocs_auc)

    # Calculando os desvios padrão para cada métrica
    std_revocacao = np.std(revocacoes)
    std_precisao = np.std(precisoes)
    std_acuracia = np.std(acuracias)
```

```
    std_F1 = np.std(Medida_F1)
    std_pr_AUC = np.std(precision_recall_auc)
    std_roc_AUC = np.std(rocs_auc)

    # Exibindo as médias das métricas obtidas
    print()
    print("=-"*6 + "Exibindo a média das métricas obtidas" + "-="*6)
    print(f"Média da acurácia: {media_acuracia:.4f} +/- {std_acuracia:.4f}")
    print(f"Média da revocação: {media_revocacao:.4f} +/- {std_revocacao:.4f}")
    print(f"Média da precisão: {media_precisao:.4f} +/- {std_precisao:.4f}")
    print(f"Média da Medida F1: {media_F1:.4f} +/- {std_F1:.4f}")
    print(f"Média da ROC AUC: {media_roc_AUC:.4f} +/- {std_roc_AUC:.4f}")
    print(f"Média da PR AUC: {media_pr_AUC:.4f} +/- {std_pr_AUC:.4f}")

    # Plotando a matriz de confusão agregada com heatmap
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm_total, annot=True, fmt=".0f", cmap="Blues")
    plt.title("Matriz de Confusão Agregada de Todos os Folds")
    plt.ylabel('Verdadeiro')
    plt.xlabel('Previsto')
    plt.show()
```

10 Modelo LightGBM

[ ]:
```
validacao_cruzada(X, y, modelo_LightGBM, k = 5, threshold = 0.5)
```

11 Modelo XGBoost

[ ]:
```
validacao_cruzada(X, y, modelo_XGBoost, k = 5, threshold = 0.5)
```

12 Modelo CatBoost

[ ]:
```
validacao_cruzada(X, y, modelo_CatBoost, k = 5, threshold = 0.5)
```

13 Modelo Balanced Random Forest

[ ]:
```
validacao_cruzada(X, y, modelo_BRandom_Forest, k = 5, threshold = 0.5)
```

Entre os modelos escolhidos para avaliar a métrica **revocação**, o XGBoost foi o que melhor performou, com:

- uma revocação média de **0.9587**, ou seja, de todos os casos que realmente eram fraude, ele identificou **95,87%**.
- uma ROC AUC média de **0.8933**, ou seja, há **89,33%** de que o modelo classifique corretamente um pedido aleatório de fraude como fraude e um pedido aleatório de não fraude como não fraude.

- uma Medida F1 média de **0.2058**, o que indica que o modelo ainda possui muitos falsos positivos (precisão baixa).

14 Feature Selection

[ ]:
```
# Inicializando o RFE
rfe = RFE(estimator = modelo_XGBoost, n_features_to_select = 21, step = 1)

encoder = CatBoostEncoder()
# Ajustar e transformar os dados de treinamento
X_train_encoded = encoder.fit_transform(X_train, y_train)

# Transformar os dados de teste
X_test_encoded = encoder.transform(X_test)

# Treinando o RFE
rfe.fit(X_train_encoded, y_train)

# Obtendo as features selecionadas
features_selecionadas = np.array(list(X_train_encoded.columns))[rfe.support_]

features_selecionadas
```

[ ]:
```
importances = modelo_XGBoost.feature_importances_

# A ordem das features selecionadas pelo RFE pode ser acessada por `ranking_`
rfe_ranking = rfe.ranking_

# Criando um array com a importância das features e o ranking do RFE
features = np.array(list(X_train_encoded.columns))
features_importance_rank = list(zip(features, importances, rfe_ranking))

# Ordenar as features com base no ranking do RFE
# Features com ranking 1 são as selecionadas. Outras têm um ranking maior e␣
 ↪foram eliminadas.
features_importance_rank.sort(key=lambda x: x[2])

# Separar as informações para plotagem
sorted_features = [x[0] for x in features_importance_rank]
sorted_importances = [x[1] for x in features_importance_rank]
sorted_rank = [x[2] for x in features_importance_rank]

# Criar um gráfico de barras com a importância das features
plt.figure(figsize=(12, 8))
bars = plt.bar(sorted_features, sorted_importances, color='blue')

# Marcar features eliminadas com uma cor diferente
```

```
for bar, rank in zip(bars, sorted_rank):
    if rank != 1:
        bar.set_color('red')

# Adicionar legendas e títulos
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importances with RFE Selection')
plt.show()
```

[ ]:
```
validacao_cruzada(X[features_selecionadas], y, modelo_XGBoost, k = 5, threshold␣
↪= 0.5)
```

Neste caso, realizando uma feature selection e excluindo as 3 colunas menos importantes para o modelos, atingimos o mesmo resultado que com o dataset original. Assim, por questões de processamento, é preferível manter o dataset com as 3 colunas a menos.

15 Tunagem de Hiperparâmetros para o XGBoost

[ ]:
```
def tunagem_hiperparametros(trial, k = 5, threshold = 0.5):

    # Parâmetros para serem tunados
    learning_rate = trial.suggest_float('learning_rate', 1e-3, 1e-1, log=True)
    max_depth = trial.suggest_int('max_depth', 1, 20)
    subsample = trial.suggest_float('subsample', 0.5, 1, step = 0.1)
    colsample_bytree = trial.suggest_float('colsample_bytree', 0.5, 1, step = 0.
↪1)
    min_child_weight = trial.suggest_int('min_child_weight', 1, 20)

    # Inicializando a função StratifiedKFold
    folds = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)

    # Criando listas para armazenar os valores de precisão, revocação,␣
↪acurácia, medida-F1, precision_recall_auc e roc_auc
    # em cada fold
    precisoes = list()
    revocacoes=list()
    acuracias=list()
    Medida_F1=list()
    precision_recall_auc=list()
    rocs_auc=list()

    # Será aplicado o método "split" no objeto folds, que retornará uma lista
    # com os índices das instâncias que pertencem ao conjunto de treino e
    # outra com os índices das instâncias que pertencem ao conjunto de teste
```

```
   for k, (train_index, test_index) in enumerate(folds.
↪split(X[features_selecionadas],y)):
        print("=-"*6 + f"Fold: {k+1}" + "-="*6)

        # Dividindo os dados em treino e teste para cada um dos folds
        X_train_intern, y_train_intern = X[features_selecionadas].
↪iloc[train_index, :], y.iloc[train_index]
        X_test_intern, y_test_intern = X[features_selecionadas].
↪iloc[test_index, :], y.iloc[test_index]

        # train_index e test_index: São os índices das instâncias do conjunto
        # de treino e teste, respectivamente, selecionados em cada um dos folds

        ##########################################
        ############## Preprocessing #############
        ##########################################

        # Instanciando o CatBoost Encoder
        encoder = CatBoostEncoder()

        # Criando um imputer para preencher com a moda os valores faltantes de␣
↪variáveis categóricas
        cat_imputer = SimpleImputer(strategy='most_frequent')

        # Criando um imputer para preencher com a mediana os valores faltantes␣
↪de variáveis numéricas
        num_imputer = SimpleImputer(strategy='median')

        # Criando pipelines para variáveis categóricas e numéricas que preenche␣
↪os valores faltantes
        cat_pipeline = Pipeline([('encoder', encoder), ('imputer',␣
↪cat_imputer)])
        num_pipeline = Pipeline([('imputer', num_imputer)])

        # Identifica as variáveis categóricas e numéricas
        cat_cols = X_train_intern.select_dtypes(include=['object']).columns
        num_cols = X_train_intern.select_dtypes(exclude=['object']).columns

        # Aplicando os pipelines no conjunto de treinamento para preencher␣
↪valores faltantes em colunas categóricas e numéricas
        X_train_intern[cat_cols] = cat_pipeline.
↪fit_transform(X_train_intern[cat_cols], y_train_intern)
        X_train_intern[num_cols] = num_pipeline.
↪fit_transform(X_train_intern[num_cols])
```

```
        # Aplicando os pipelines ao conjunto de teste para preencher valores␣
↪faltantes em colunas categóricas e numéricas
        X_test_intern[cat_cols] = cat_pipeline.
↪transform(X_test_intern[cat_cols])
        X_test_intern[num_cols] = num_pipeline.
↪transform(X_test_intern[num_cols])

        # Instanciando o Modelo XGBoost
        modelo_XGBoost = XGBClassifier(n_estimators = 1000, max_depth =␣
↪max_depth, learning_rate = learning_rate,
                                       subsample = subsample, colsample_bytree␣
↪= colsample_bytree, min_child_weight = min_child_weight,
                                       n_jobs =-1, random_state = 0,␣
↪scale_pos_weight=weights, eval_metric='error')

        # Treinando o modelo XGBoost
        modelo_XGBoost.fit(X_train_intern, y_train_intern)

        # Obtendo as probabilidades de cada registro pertencer a classe 1
        y_pred_proba = modelo_XGBoost.predict_proba(X_test_intern)[:, 1]

        # Obtendo as previsões do modelo
        y_pred = np.where(y_pred_proba > threshold, 1, 0)

        # Calculando a precisão e revocação para determinar a␣
↪precision_recall_auc
        precisao, revocacao, limiares = precision_recall_curve(y_test_intern,␣
↪y_pred)

        # Determinando as métricas para cada fold
        precisao_revocacao_auc = auc(revocacao, precisao)
        roc_auc = roc_auc_score(y_test_intern, y_pred)
        acuracia_score = accuracy_score(y_test_intern, y_pred)
        precisao_score = precision_score(y_test_intern, y_pred)
        revocacao_score = recall_score(y_test_intern, y_pred)
        f1score = f1_score(y_test_intern, y_pred)

        # Armazenando as métricas nas listas criadas
        precisoes.append(precisao_score)
        revocacoes.append(revocacao_score)
        precision_recall_auc.append(precisao_revocacao_auc)
        rocs_auc.append(roc_auc)
        acuracias.append(acuracia_score)
        Medida_F1.append(f1score)

    # Transformando as listas em arrays para fazer operações matemáticas
```

```
    precisoes = np.array(precisoes)
    revocacoes = np.array(revocacoes)
    precision_recall_auc = np.array(precision_recall_auc)
    rocs_auc = np.array(rocs_auc)
    acuracias = np.array(acuracias)
    Medida_F1 = np.array(Medida_F1)

    # Calculando as médias das métricas
    media_revocacao = np.mean(revocacoes)
    media_precisao = np.mean(precisoes)
    media_acuracia = np.mean(acuracias)
    media_F1 = np.mean(Medida_F1)
    media_pr_AUC = np.mean(precision_recall_auc)
    media_roc_AUC = np.mean(rocs_auc)

    # Calculando os desvios padrão para cada métrica
    std_revocacao = np.std(revocacoes)
    std_precisao = np.std(precisoes)
    std_acuracia = np.std(acuracias)
    std_F1 = np.std(Medida_F1)
    std_pr_AUC = np.std(precision_recall_auc)
    std_roc_AUC = np.std(rocs_auc)

    return media_roc_AUC


study = opt.create_study(direction='maximize')
study.optimize(tunagem_hiperparametros, n_trials = 20)
```

[ ]:
```
params = {'learning_rate': 0.006273985142858805, 'max_depth': 7, 'subsample': 0.
↪7, 'colsample_bytree': 0.8, 'min_child_weight': 12}
```

[ ]:
```
# XGBoost executado para os melhores parâmetros
modelo_XGBoost = XGBClassifier(n_estimators = 1000, n_jobs =-1, random_state =␣
↪0, scale_pos_weight=weights, eval_metric='error', **params)
```

[ ]:
```
validacao_cruzada(X[features_selecionadas], y, modelo_XGBoost, k = 5, threshold␣
↪= 0.5)
```

Com a realização da tunagem de hiperparâmetros:

- A ROC AUC saltou de 0.8933 para **0.9042**
- A revocação praticamente **se manteve a mesma**
- A precisão saltou de 0.1155 para **0.1338**
- A Medida F1 saltou de 0.2058 para **0.2344**
- A PR AUC saltou de 0.5376 para **0.5428**
- A acurácia saltou de 0.8307 para **0.8597**

No geral houve uma pequena melhora no modelo.

16 O quanto a DataCo Global ganhou ao identificar corretamente as fraudes?

[ ]:
```
encoder = CatBoostEncoder()

X_train_encoded = encoder.fit_transform(X_train[features_selecionadas], y_train)
X_test_encoded = encoder.transform(X_test[features_selecionadas])

# Fazendo previsões de probabilidade de fraude para o conjunto de teste
xgb_probs = modelo_XGBoost.predict_proba(X_test_encoded)[:, 1]

# Aqui são trazidas as probabilidades encontradas pelo XGBoost

df_test = X_test[features_selecionadas].copy()
df_test['fraude'] = y_test
df_test['XGB_Prob'] = xgb_probs
```

[ ]:
```
# Função para calcular o impacto financeiro das decisões de bloqueio de␣
↪transações

def calculo_impacto_financeiro(df, blocked_col, fraud_col, profit_col):

    # Calculando perdas por fraude (transações que são fraudes e não foram␣
↪bloqueadas)
    df['fraud_loss'] = ((df[fraud_col]) & (~df[blocked_col])) * df[profit_col]

    # lucro obtido de transações legítimas que o sistema corretamente␣
↪identificou como não fraudulentas e, portanto, não bloqueou.
    df['saved_profit'] = ((~df[fraud_col]) & (~df[blocked_col])) *␣
↪df[profit_col]

    # Representa o lucro líquido após considerar tanto as perdas por fraude␣
↪quanto o lucro preservado.
    df['total_profit'] = df['saved_profit'] - df['fraud_loss']

    return df[['fraud_loss', 'saved_profit', 'total_profit']].sum()
```

[ ]:
```
# Definindo uma gama de limiares possíveis
possiveis_thresholds = np.linspace(0.01, 0.99, 99)

# Inicializando uma lista para armazenar os resultados
impactos_financeiros = []

# Testando cada limiar
for threshold in possiveis_thresholds:
    # Aplicando o limiar atual
    df_test['blocked'] = df_test['XGB_Prob'] >= threshold
```

```
    # Calculando o impacto financeiro para o limiar atual
    impacto = calculo_impacto_financeiro(df_test, 'blocked', 'fraude', 'Benefit␣
 ↪per order')

    # Armazenando os resultados, incluindo o limiar
    impactos_financeiros.append({
        'threshold': threshold,
        'Perda por fraude': impacto['fraud_loss'],
        'Lucro Salvo': impacto['saved_profit'],
        'Lucro Total': impacto['total_profit']
    })

# Convertendo os resultados em um DataFrame
results_df = pd.DataFrame(impactos_financeiros)

# Encontrando o limiar com o maior lucro
best_result = results_df.loc[results_df['Lucro Total'].idxmax()]
best_result_df = pd.DataFrame([best_result])

# Exibindo o melhor limiar e o lucro associado
best_result_df
```

[ ]:
```
print(f"O modelo conseguiu lucrar {best_result_df['Lucro Salvo'].iloc[0]:.2f}␣
 ↪ao identificar corretamente operações fraudulentas!")
print(f"Porém, modelo perdeu {best_result_df['Perda por fraude'].iloc[0]:.2f}␣
 ↪por não identificar corretamente outras operações que realmente eram fraudes!
 ↪")
print(f"Considerando as perdas por fraude, o modelo conseguiu lucrar␣
 ↪{best_result_df['Lucro Total'].iloc[0]:.2f} ao total!")
```

[ ]:
```

```