Modelo de Classificacao

Usando SVM (Suporte Vector Machine) Para Prever a Intenção de Compra de Usuários de E-Commerce

Definição do Problema de Negócio:

Nosso trabalho será avaliar quais atributos influenciam um usuário na compra de produtos online e construir um modelo preditivo para realizar previsões de compras futuras.

Usaremos como fonte de dados o dataset:

Online Shoppers Purchasing Intention Dataset

https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset (https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset)

O conjunto de dados consiste em valores de recursos pertencentes a 12.330 sessões online. O conjunto de dados foi formado de modo que cada sessão pertença a um usuário diferente em um período de 1 ano para evitar qualquer tendência para uma campanha específica, dia especial, usuário, perfil ou período.

O conjunto de dados consiste em 10 atributos numéricos e 8 categóricos. O atributo 'Revenue' pode ser usado como o rótulo da classe, ou seja, nossa variável ALVO

Importando os pacotes a serem utilizados

In [1]:
```
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de
# pip install -U nome_pacote

# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
```

In [3]:
```
# Importando bibliotecas que iremos utilizar...
import time
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_
from sklearn import svm
import sklearn
import matplotlib
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings('ignore', category=DeprecationWarning)
```

Carga e Dicionário de Dados

In [4]:
```
# Carregando os dados
# O arquivo online_shoppers_intention.csv estarei disponibilizando junto com o
df_original = pd.read_csv('online_shoppers_intention.csv')
df_original.head()
```

Out[4]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated
0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	2.0
2	0.0	-1.0	0.0	-1.0	1.0
3	0.0	0.0	0.0	0.0	2.0
4	0.0	0.0	0.0	0.0	10.0

Dicionário de Dados:

"Administrativo", "Duração administrativa", "Informativo", "Duração informativo", "Relacionado ao produto" e "Duração relacionada ao produto" representam o número de diferentes tipos de páginas visitadas pelo visitante nessa sessão e o tempo total gasto em cada uma dessas categorias de página. Os valores desses recursos são derivados das informações de URL das páginas visitadas pelo usuário e atualizadas em tempo real quando um usuário executa uma ação, por exemplo, passando de uma página para outra.

Os recursos "Taxa de rejeição", "Taxa de saída" e "Valor da página" representam as métricas medidas pelo "Google Analytics" para cada página no site de comércio eletrônico.

O valor do recurso "Taxa de rejeição" de uma página da web refere-se à porcentagem de visitantes que entram no site a partir dessa página e saem ("rejeição") sem acionar outras solicitações ao servidor durante essa sessão.

O valor do recurso "Taxa de saída" para uma página da web específica é calculado como a porcentagem que foi a última na sessão, para todas as exibições de página a página.

O recurso "Valor da página" representa o valor médio para uma página da web que um usuário visitou antes de concluir uma transação de comércio eletrônico.

O recurso "Dia especial" indica a proximidade do horário de visita do site a um dia especial específico (por exemplo, dia das mães, dia dos namorados) em que as sessões têm mais probabilidade de serem finalizadas com a transação. O valor desse atributo é determinado considerando a dinâmica do comércio eletrônico, como a duração entre a data do pedido e a data de entrega. Por exemplo, no dia dos namorados, esse valor assume um valor diferente de zero entre 2 e 12 de fevereiro (dia dos namorados nos EUA e Europa), zero antes e depois dessa data, a menos que esteja próximo de outro dia especial e seu valor máximo de 1 em 8 de fevereiro.

O conjunto de dados também inclui o tipo de sistema operacional, navegador, região, tipo de tráfego, tipo de visitante como visitante novo ou recorrente, um valor booleano indicando se a data da visita é final de semana e mês do ano.

A variável alvo (Revenue) é booleana, com True se a sessão gerou receita e False se não gerou.

Análise Exploratória

In [5]:
```
# Shape dos dados
df_original.shape
```

Out[5]:
```
(12330, 18)
```

In [8]:
```
# Tipos de Dados
df_original.dtypes
```

Out[8]:
```
Administrative             float64
Administrative_Duration    float64
Informational              float64
Informational_Duration     float64
ProductRelated             float64
ProductRelated_Duration    float64
BounceRates                float64
ExitRates                  float64
PageValues                 float64
SpecialDay                 float64
Month                       object
OperatingSystems             int64
Browser                      int64
Region                       int64
TrafficType                  int64
VisitorType                 object
Weekend                       bool
Revenue                       bool
dtype: object
```

In [9]:
```
# Verificando valores missing
print(df_original.isna().sum())
```

```
Administrative             14
Administrative_Duration    14
Informational              14
Informational_Duration     14
ProductRelated             14
ProductRelated_Duration    14
BounceRates                14
ExitRates                  14
PageValues                  0
SpecialDay                  0
Month                       0
OperatingSystems            0
Browser                     0
Region                      0
TrafficType                 0
VisitorType                 0
Weekend                     0
Revenue                     0
dtype: int64
```

In [10]:
```
# Removendo as linhas com valores missing
df_original.dropna(inplace = True)
```

In [11]:
```
# Verificando valores missing
print(df_original.isna().sum())
```

```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

In [12]:
```
# Shape
df_original.shape
```

Out[12]: (12316, 18)

In [13]:
```
# Verificando Valores Únicos
df_original.nunique()
```

Out[13]:
```
Administrative              27
Administrative_Duration    3336
Informational               17
Informational_Duration     1259
ProductRelated              311
ProductRelated_Duration    9552
BounceRates                1872
ExitRates                  4777
PageValues                 2704
SpecialDay                    6
Month                        10
OperatingSystems              8
Browser                      13
Region                        9
TrafficType                  20
VisitorType                   3
Weekend                       2
Revenue                       2
dtype: int64
```

Para fins de visualização, dividiremos os dados em variáveis contínuas e categóricas. Trataremos todas as variáveis com menos de 30 entradas únicas como categóricas.

In [14]:
```
# Preparando os dados para o plot

# Cria uma cópia do dataset original
df = df_original.copy()

# Listas vazias para os resultados
continuous = []
categorical = []

# Loop pelas colunas
for c in df.columns[:-1]:
    if df.nunique()[c] >= 30:
        continuous.append(c)
    else:
        categorical.append(c)
```

In [15]:
```
continuous
```

Out[15]:
```
['Administrative_Duration',
 'Informational_Duration',
 'ProductRelated',
 'ProductRelated_Duration',
 'BounceRates',
 'ExitRates',
 'PageValues']
```

In [16]:
```
# Variáveis contínuas
df[continuous].head()
```

Out[16]:

	Administrative_Duration	Informational_Duration	ProductRelated	ProductRelated_Duration	Bou
0	0.0	0.0	1.0	0.000000	
1	0.0	0.0	2.0	64.000000	
2	-1.0	-1.0	1.0	-1.000000	
3	0.0	0.0	2.0	2.666667	
4	0.0	0.0	10.0	627.500000	

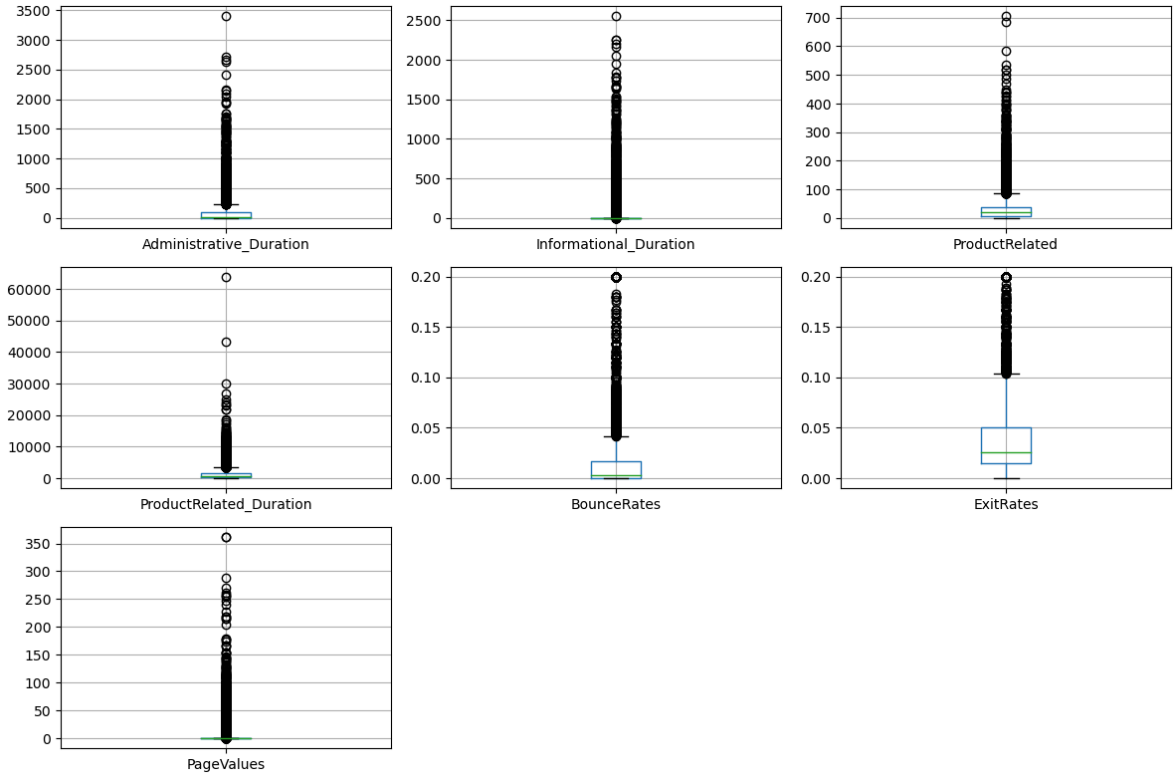
In [17]:
```
# Variáveis categóricas
df[categorical].head()
```

Out[17]:

	Administrative	Informational	SpecialDay	Month	OperatingSystems	Browser	Region	Traffic
0	0.0	0.0	0.0	Feb	1	1	1	
1	0.0	0.0	0.0	Feb	2	2	1	
2	0.0	0.0	0.0	Feb	4	1	9	
3	0.0	0.0	0.0	Feb	3	2	2	
4	0.0	0.0	0.0	Feb	3	3	1	

Gráficos para variáveis numéricas.

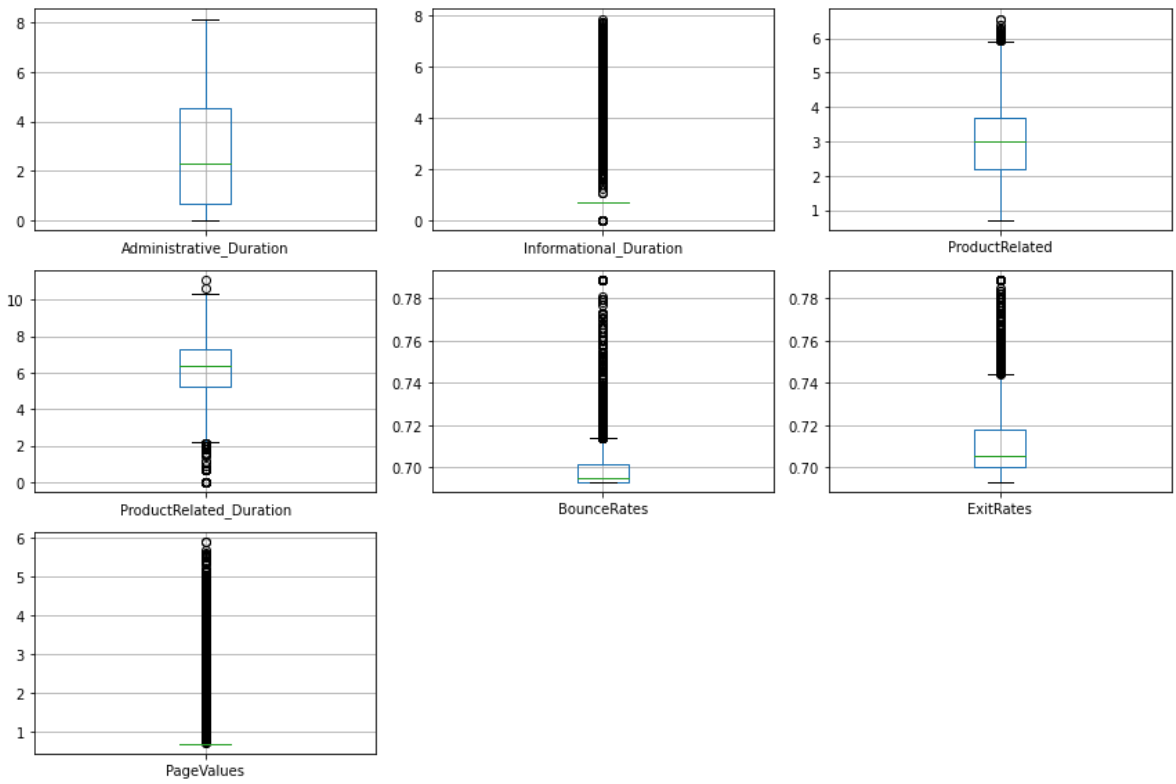
In [18]:
```
# Plot das variáveis contínuas

# Tamanho da área de plotagem
fig = plt.figure(figsize = (12,8))

# Loop pelas variáveis contínuas
for i, col in enumerate(continuous):
    plt.subplot(3, 3, i + 1);
    df.boxplot(col);
    plt.tight_layout()

# Podemos salvar também nossa imagem dos BoxPlosts
plt.savefig('boxplot1.png')
```



Observe que Variáveis contínuas parecem extremamente distorcidas. Vamos aplicar transformação de log (logaritmo) para melhor visualização desses dados.

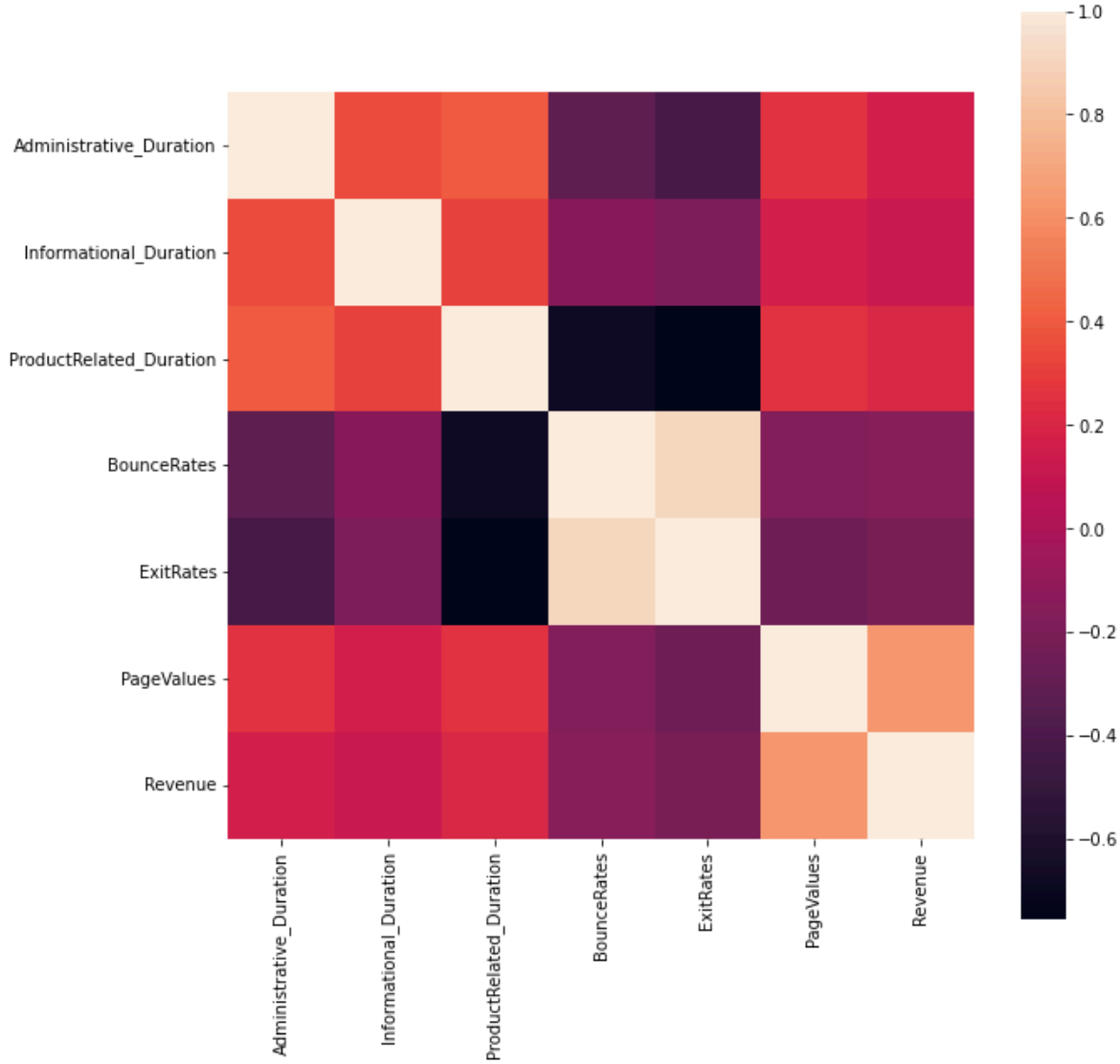
In [16]:
```
# Transformação de log nas variáveis contínuas
df[continuous] = np.log1p(1 + df[continuous])
```

In [17]:
```
# Plot das variáveis contínuas

# Tamanho da área de plotagem
fig = plt.figure(figsize = (12,8))

# Loop pelas variáveis contínuas
for i,col in enumerate(continuous):
    plt.subplot(3,3,i+1);
    df.boxplot(col);
    plt.tight_layout()
plt.savefig('boxplot2.png')
```



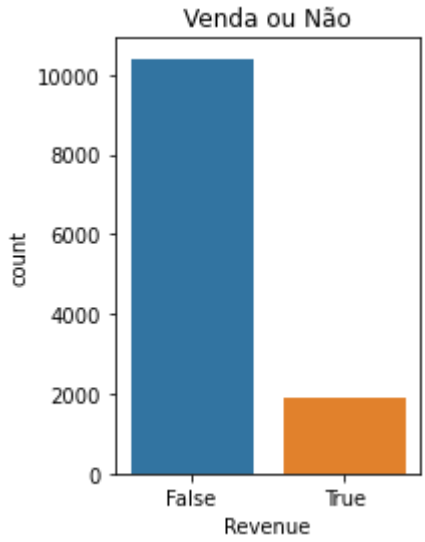
Matriz de Correlação Entre Variáveis Contínuas.

In [18]:
```
# Área de plotagem
plt.figure(figsize = (10,10))

# Matriz de Correlação
sns.heatmap(df[['Administrative_Duration',
                'Informational_Duration',
                'ProductRelated_Duration',
                'BounceRates',
                'ExitRates',
                'PageValues',
                'Revenue']].corr(), vmax = 1., square = True)
plt.show()
```

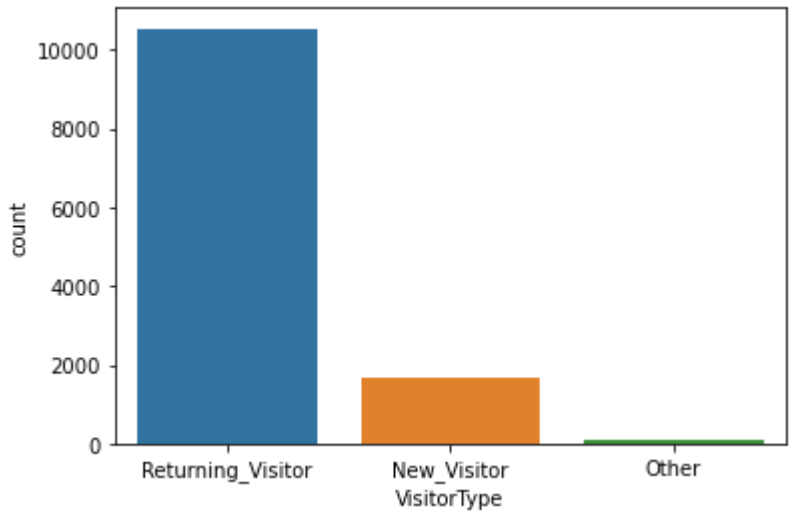


Visualização de gráficos de variáveis categóricas para analisar como a variável de destino é influenciada por elas.

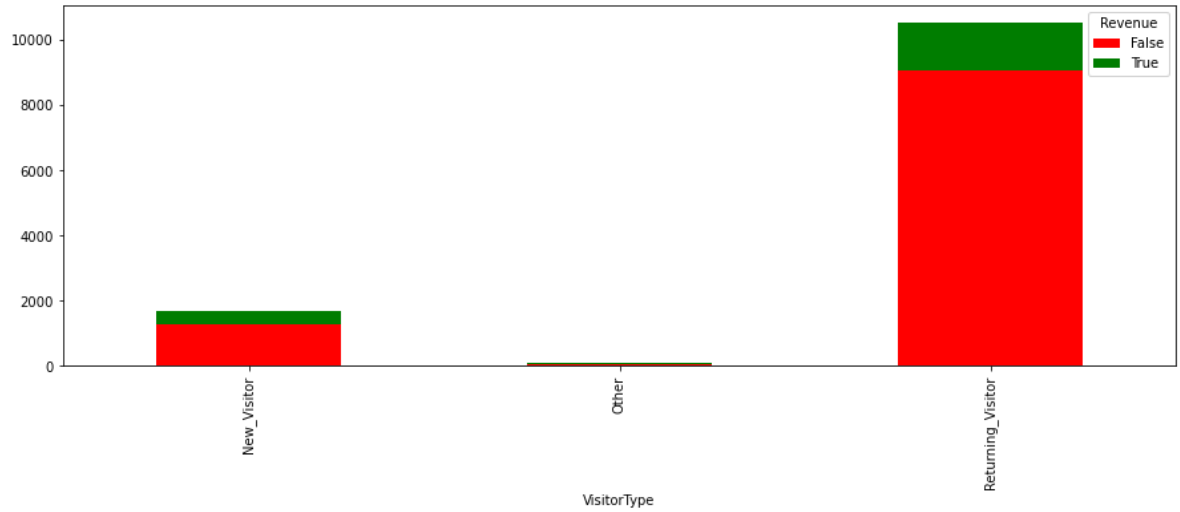
In [19]:
```
# Countplot Venda ou Não
#warnings.filterwarnings("ignore", category=FutureWarning)

plt.subplot(1,2,2)
plt.title("Venda ou Não")
sns.countplot(df['Revenue'])
plt.show()
```



In [20]:
```
# Countplot Tipo de Visitante
plt.xlabel("Tipo de Visitante")
sns.countplot(df['VisitorType'])
plt.show()
```

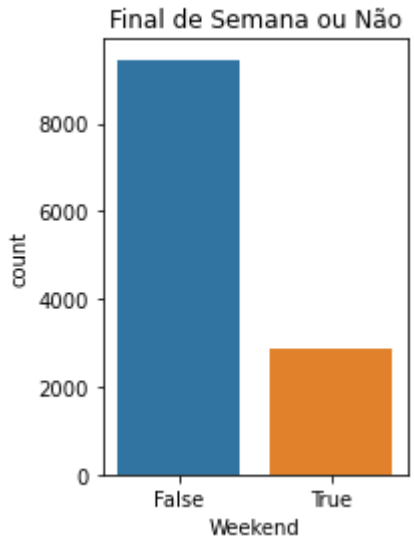


In [21]:
```
# Stacked Bar Tipo de Visitante x Revenue
pd.crosstab(df['VisitorType'], df['Revenue']).plot(kind = 'bar',
                                                   stacked = True,
                                                   figsize = (15, 5),
                                                   color = ['red', 'green'])
plt.show()
```

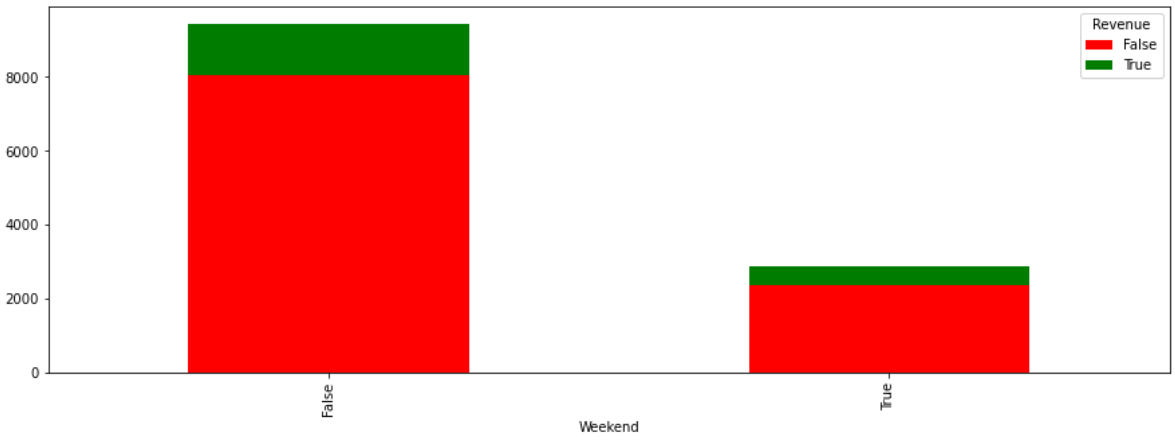


In [22]:
```
# Gráfico de Pizza de Tipos de Visitantes
labels = ['Visitante_Retornando', 'Novo_Visitante', 'Outro']
plt.title("Tipos de Visitantes")
plt.pie(df['VisitorType'].value_counts(), labels = labels, autopct = '%.2f%%')
plt.legend()
plt.show()
```

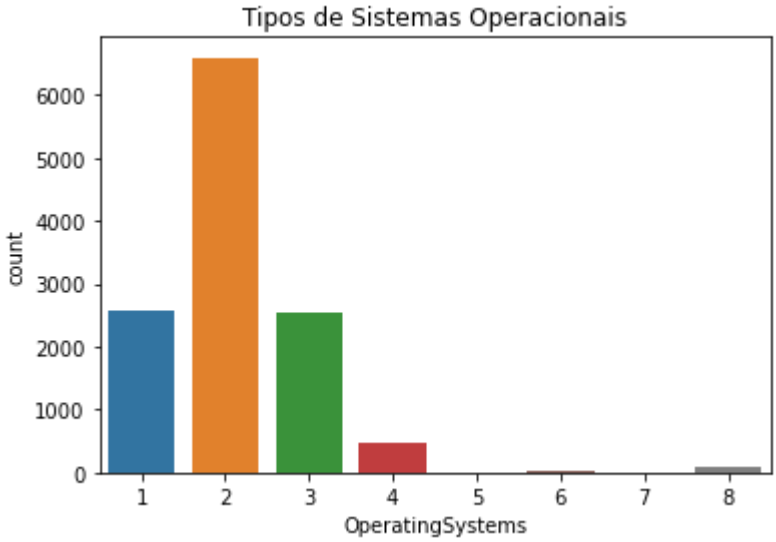
In [23]:
```
# Countplot Final de Semana ou Não
plt.subplot(1,2,1)
plt.title("Final de Semana ou Não")
sns.countplot(df['Weekend'])
plt.show()
```



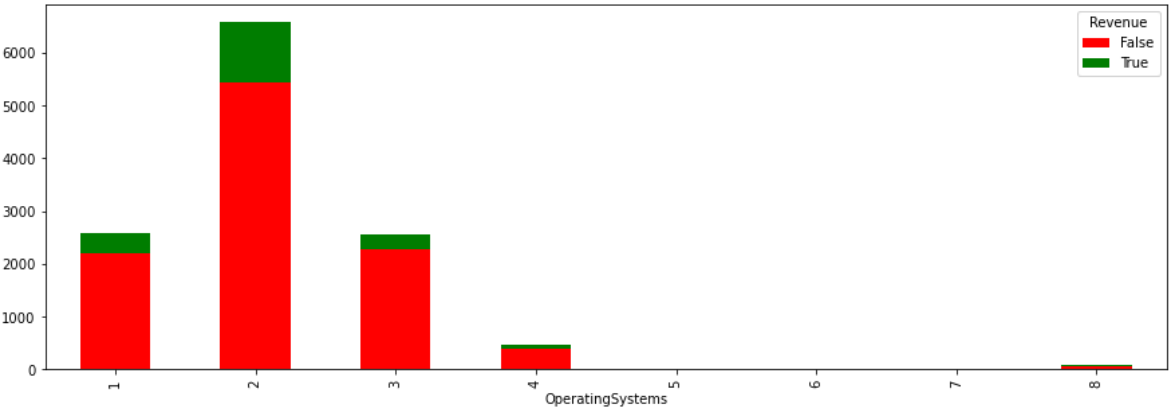
In [24]:
```
# Stacked Bar Final de Semana x Revenue
pd.crosstab(df['Weekend'], df['Revenue']).plot(kind = 'bar',
                                               stacked = True,
                                               figsize = (15, 5),
                                               color = ['red', 'green'])
plt.show()
```



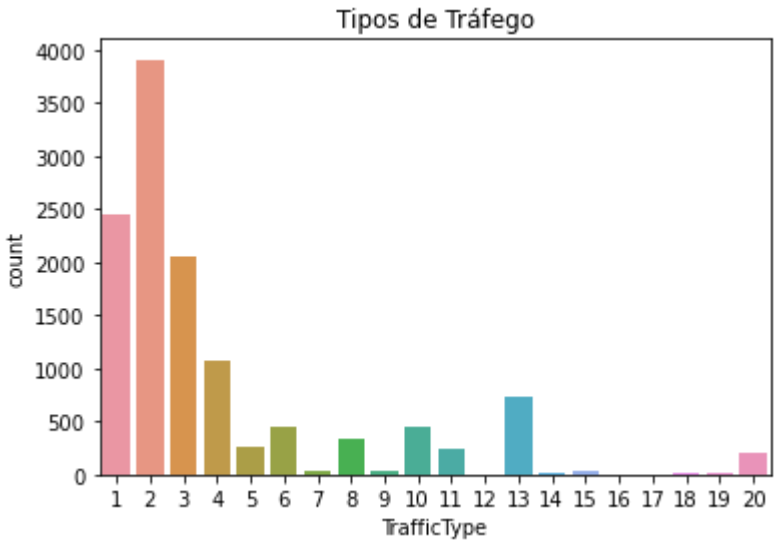
In [25]:
```
# Countplot Tipos de Sistemas Operacionais
#plt.figure(figsize = (15,6))
plt.title("Tipos de Sistemas Operacionais")
plt.xlabel("Sistema Operacional Usado")
sns.countplot(df['OperatingSystems'])
plt.show()
```



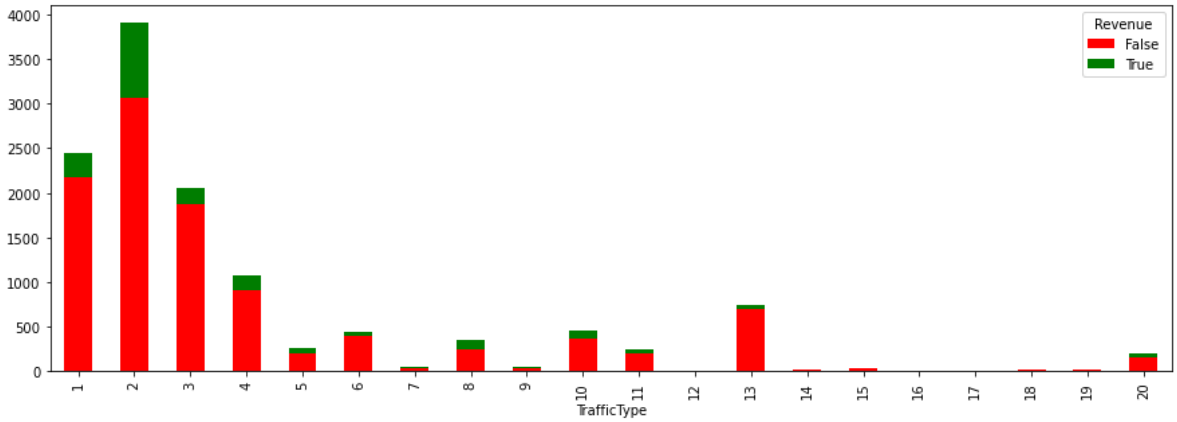
In [26]:
```
# Stacked Bar Tipo de SO x Revenue
pd.crosstab(df['OperatingSystems'], df['Revenue']).plot(kind = 'bar',
                                                        stacked = True,
                                                        figsize = (15, 5),
                                                        color = ['red', 'green
plt.show()
```



In [27]:
```
# Countplot Tipo de Tráfego
plt.title("Tipos de Tráfego")
plt.xlabel("Tipo de Tráfego")
sns.countplot(df['TrafficType'])
plt.show()
```



In [28]:
```
# Stacked Bar Tipos de Tráfego x Revenue
pd.crosstab(df['TrafficType'], df['Revenue']).plot(kind = 'bar',
                                                   stacked = True,
                                                   figsize = (15, 5),
                                                   color = ['red', 'green'])
plt.show()
```



Pré-Processamento dos Dados

In [29]:
```
df_original.head()
```

Out[29]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated
0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	2.0
2	0.0	-1.0	0.0	-1.0	1.0
3	0.0	0.0	0.0	0.0	2.0
4	0.0	0.0	0.0	0.0	10.0

In [30]:
```
# Cria o encoder
lb = LabelEncoder()

# Aplica o encoder nas variáveis que estão com string
df_original['Month'] = lb.fit_transform(df_original['Month'])
df_original['VisitorType'] = lb.fit_transform(df_original['VisitorType'])

# Remove valores missing eventualmente gerados
df_original.dropna(inplace = True)
```

In [31]:
```
df_original.head(200)
```

Out[31]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelate
0	0.0	0.000000	0.0	0.0	1.
1	0.0	0.000000	0.0	0.0	2.
2	0.0	-1.000000	0.0	-1.0	1.
3	0.0	0.000000	0.0	0.0	2.
4	0.0	0.000000	0.0	0.0	10.
...	...	...	...	...	.
195	0.0	0.000000	0.0	0.0	98.
196	2.0	56.000000	1.0	144.0	67.
197	3.0	112.960784	0.0	0.0	13.
198	0.0	0.000000	0.0	0.0	17.
199	3.0	94.000000	2.0	125.0	55.

200 rows × 18 columns

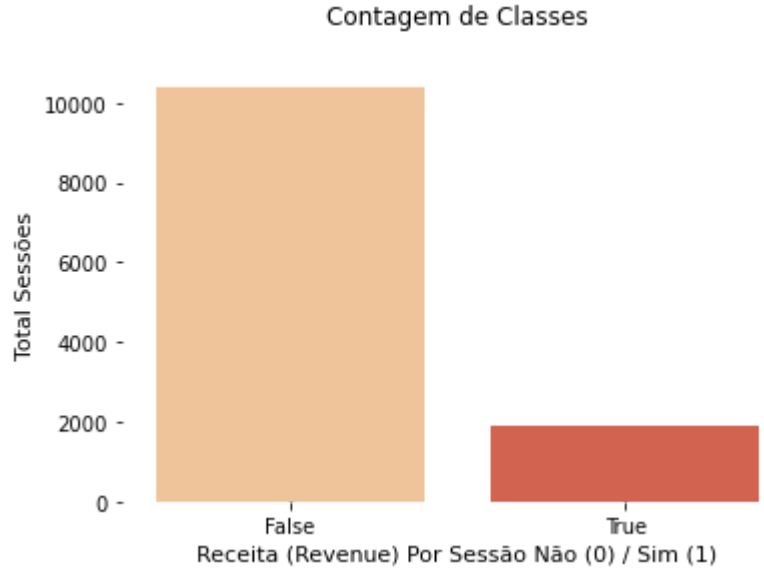
In [32]:
```
# Shape
df_original.shape
```

Out[32]:
```
(12316, 18)
```

In [33]:
```
# Verificando se a variável resposta está balanceada
target_count = df_original.Revenue.value_counts()
target_count
```

Out[33]:
```
False    10408
True      1908
Name: Revenue, dtype: int64
```

In [34]:
```
# Plot
sns.countplot(df_original.Revenue, palette = "OrRd")
plt.box(False)
plt.xlabel('Receita (Revenue) Por Sessão Não (0) / Sim (1)', fontsize = 11)
plt.ylabel('Total Sessões', fontsize = 11)
plt.title('Contagem de Classes\n')
plt.show()
```



In [35]:
```
# Instala e importa o pacote imblearn
# Se apresentar erro na instalação execute este comando !pip install scikit-le
# Após a atualização estar completa, reinicie o jupyter notebook.
#!pip install -q imblearn
#import imblearn
```

In [36]:
```
# Shape
df_original.shape
```

Out[36]:
```
(12316, 18)
```

In [37]:
```
# Variáveis explicativas
df_original.iloc[:, 0:17].head()
```

Out[37]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated
0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	2.0
2	0.0	-1.0	0.0	-1.0	1.0
3	0.0	0.0	0.0	0.0	2.0
4	0.0	0.0	0.0	0.0	10.0

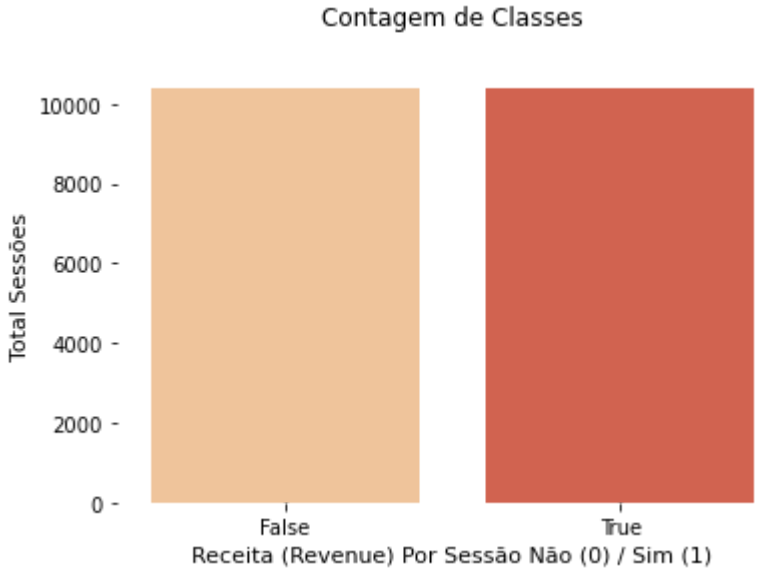
In [38]:
```
# Variável Target
df_original.iloc[:, 17].head()
```

Out[38]:
```
0    False
1    False
2    False
3    False
4    False
Name: Revenue, dtype: bool
```

Balanceamento de Classe - Oversampling

In [39]:
```
# Importa a função
#import sklearn
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE

# Seed para reproduzir o mesmo resultado
seed = 100

# Separa X e y
X = df_original.iloc[:, 0:17]
y = df_original.iloc[:, 17]

# Cria o balanceador SMOTE
smote_bal = SMOTE(random_state = seed)

# Aplica o balanceador
X_res, y_res = smote_bal.fit_resample(X, y)
```

In [40]:
```
# Plot
sns.countplot(y_res, palette = "OrRd")
plt.box(False)
plt.xlabel('Receita (Revenue) Por Sessão Não (0) / Sim (1)', fontsize = 11)
plt.ylabel('Total Sessões', fontsize = 11)
plt.title('Contagem de Classes\n')
plt.show()
```



In [41]:
```
# Shape dos dados originais
df_original.shape
```

Out[41]: (12316, 18)

In [42]:
```
# Shape dos dados reamostrados
X_res.shape
```

Out[42]: (20816, 17)

In [43]:
```
# Shape dos dados reamostrados
y_res.shape
```

Out[43]: (20816,)

In [44]:
```
# Ajustando X e y
# Para fins de demonstração deste exemplo vou utilizar somente 2.000 observaçõ
# modelo

#X = X_res
#y = y_res

X = X_res.head(1000)
y = y_res.head(1000)
```

In [45]:
```
# Divisão em Dados de Treino e Teste.
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size = 0.3,
```

Modelo SVM

Modelo Base com Kernel Linear

In [46]:
```
# Cria o modelo
modelo_v1 = svm.SVC(kernel = 'linear')
```

In [47]:
```
# Treinamento
start = time.time()
modelo_v1.fit(X_treino, y_treino)
end = time.time()
print('Tempo de Treinamento do Modelo:', end - start)
```

```
Tempo de Treinamento do Modelo: 105.63111400604248
```

In [48]:
```
# Previsões
previsoes_v1 = modelo_v1.predict(X_teste)
```

In [49]:
```
# Dicionário de métricas e metadados
SVM_dict_v1 = {'Modelo':'SVM',
               'Versão':'1',
               'Kernel':'Linear',
               'Precision':precision_score(previsoes_v1, y_teste),
               'Recall':recall_score(previsoes_v1, y_teste),
               'F1 Score':f1_score(previsoes_v1, y_teste),
               'Acurácia':accuracy_score(previsoes_v1, y_teste),
               'AUC':roc_auc_score(y_teste, previsoes_v1)}
```

In [50]:
```
# Print
print("Métricas em Teste:\n")
SVM_dict_v1
```

```
Métricas em Teste:
```

Out[50]:
```
{'Modelo': 'SVM',
 'Versão': '1',
 'Kernel': 'Linear',
 'Precision': 0.52,
 'Recall': 0.7647058823529411,
 'F1 Score': 0.6190476190476191,
 'Acurácia': 0.9466666666666667,
 'AUC': 0.7527272727272728}
```

Modelo com Kernel Linear e Dados Padronizados (Scaled)

In [51]:
```
# ***** Atenção *****
# O método nesta célula não deve ser usado, pois estaríamos aplicando o fit em
# Aplicamos o fit somente nos dados de treino e aplicamos o transform nos dado
# Padronização
# X_treino_scaled = StandardScaler().fit_transform(X_treino)
# X_teste_scaled = StandardScaler().fit_transform(X_teste)
```

In [52]:
```
# Agora sim, a forma ideal de aplicar a padronização em treino e teste
# Padronização
sc = StandardScaler()
X_treino_scaled = sc.fit_transform(X_treino)
X_teste_scaled = sc.transform(X_teste)
```

Obsevação:

Para impedir que as informações sobre a distribuição do conjunto de teste vazem em seu modelo, o ideal é aplicar a padronização em separado nos dados de treino e de teste, ajustando o redimensionador apenas aos dados de treinamento, padronizando então os conjuntos de treinamento e teste com esse redimensionador (exatamente como está na célula acima). Ao ajustar o redimensionador no conjunto de dados completo antes da divisão em treino e teste, informações sobre o conjunto de testes são usadas para transformar o conjunto de treinamento.

Conhecer a distribuição de todo o conjunto de dados pode influenciar como você detecta e processa outliers, bem como como você parametriza seu modelo. Embora os dados em si não sejam expostos, há informações sobre a distribuição dos dados. Como resultado, o desempenho do seu conjunto de testes não é uma estimativa real do desempenho em dados invisíveis.

Sempre aplique a padronização depois de fazer a divisão em treino e teste, exatamente como fizemos aqui. Usamos fit_transform() nos dados de treino e transform() nos dados de teste quando usamos o StandardScaler().

In [53]:
```
X_treino_scaled
```

Out[53]:
```
array([[-0.59034343, -0.40561073, -0.31756214, ..., -0.75751167,
         0.33597259, -0.54433105],
       [-0.59034343, -0.40561073, -0.31756214, ..., -0.41048407,
         0.33597259,  1.83711731],
       [ 0.07887179, -0.33356238, -0.31756214, ..., -0.75751167,
         0.33597259, -0.54433105],
       ...,
       [-0.59034343, -0.40561073, -0.31756214, ..., -0.41048407,
        -2.97643326, -0.54433105],
       [ 0.07887179, -0.16326629, -0.31756214, ..., -0.41048407,
        -2.97643326,  1.83711731],
       [-0.59034343, -0.40561073, -0.31756214, ..., -0.41048407,
         0.33597259, -0.54433105]])
```

In [54]:
```
X_teste_scaled
```

Out[54]:
```
array([[ 0.4134794 , -0.02571945, -0.31756214, ..., -0.06345648,
         0.33597259, -0.54433105],
       [-0.25573582, -0.26151404, -0.31756214, ..., -0.41048407,
         0.33597259, -0.54433105],
       [ 1.08269462,  1.52462998,  4.89244178, ..., -0.75751167,
         0.33597259, -0.54433105],
       ...,
       [-0.59034343, -0.40561073, -0.31756214, ..., -0.41048407,
         0.33597259, -0.54433105],
       [-0.25573582, -0.41216058, -0.31756214, ...,  0.63059872,
         0.33597259,  1.83711731],
       [-0.59034343, -0.40561073, -0.31756214, ..., -0.75751167,
         0.33597259,  1.83711731]])
```

In [55]:
```
# Cria o modelo
modelo_v2 = svm.SVC(kernel = 'linear')
```

In [56]:
```
# Treinamento
start = time.time()
modelo_v2.fit(X_treino_scaled, y_treino)
end = time.time()
print('Tempo de Treinamento do Modelo:', end - start)
```

```
Tempo de Treinamento do Modelo: 0.01396322250366211
```

In [57]:
```
# Previsões
previsoes_v2 = modelo_v2.predict(X_teste_scaled)
```

In [58]:
```
# Dicionário de métricas e metadados
SVM_dict_v2 = {'Modelo':'SVM',
               'Versão':'2',
               'Kernel':'Linear com Dados Padronizados',
               'Precision':precision_score(previsoes_v2, y_teste),
               'Recall':recall_score(previsoes_v2, y_teste),
               'F1 Score':f1_score(previsoes_v2, y_teste),
               'Acurácia':accuracy_score(previsoes_v2, y_teste),
               'AUC':roc_auc_score(y_teste, previsoes_v2)}
```

In [59]:
```
# Print
print("Métricas em Teste:\n")
SVM_dict_v2
```

```
Métricas em Teste:
```

Out[59]:
```
{'Modelo': 'SVM',
 'Versão': '2',
 'Kernel': 'Linear com Dados Padronizados',
 'Precision': 0.48,
 'Recall': 0.8571428571428571,
 'F1 Score': 0.6153846153846153,
 'Acurácia': 0.95,
 'AUC': 0.7363636363636362}
```

Otimização de Hiperparâmetros com Grid Search e Kernel RBF

In [60]:
```
# Cria o modelo
modelo_v3 = svm.SVC(kernel = 'rbf')

# Valores para o grid
C_range = np.array([50., 100., 200.])
gamma_range = np.array([0.3*0.001,0.001,3*0.001])

# Grid de hiperparâmetros
svm_param_grid = dict(gamma = gamma_range, C = C_range)

# Grid Search
start = time.time()
modelo_v3_grid_search_rbf = GridSearchCV(modelo_v3, svm_param_grid, cv = 3)

# Treinamento
modelo_v3_grid_search_rbf.fit(X_treino_scaled, y_treino)
end = time.time()
print('Tempo de Treinamento do Modelo com Grid Search:', end - start)

# Acurácia em Treino
print(f"Acurácia em Treinamento: {modelo_v3_grid_search_rbf.best_score_ :.2%}"
print("")
print(f"Hiperparâmetros Ideais: {modelo_v3_grid_search_rbf.best_params_}")
```

```
Tempo de Treinamento do Modelo com Grid Search: 0.16556692123413086
Acurácia em Treinamento: 94.86%

Hiperparâmetros Ideais: {'C': 50.0, 'gamma': 0.003}
```

In [61]:
```
# Previsões
previsoes_v3 = modelo_v3_grid_search_rbf.predict(X_teste_scaled)
```

In [62]:
```
# Dicionário de métricas e metadados
SVM_dict_v3 = {'Modelo':'SVM',
               'Versão':'3',
               'Kernel':'RBF com Dados Padronizados',
               'Precision':precision_score(previsoes_v3, y_teste),
               'Recall':recall_score(previsoes_v3, y_teste),
               'F1 Score':f1_score(previsoes_v3, y_teste),
               'Acurácia':accuracy_score(previsoes_v3, y_teste),
               'AUC':roc_auc_score(y_teste, previsoes_v3)}
```

In [63]:
```
# Print
print("Métricas em Teste:\n")
SVM_dict_v3
```

```
Métricas em Teste:
```

Out[63]:
```
{'Modelo': 'SVM',
 'Versão': '3',
 'Kernel': 'RBF com Dados Padronizados',
 'Precision': 0.52,
 'Recall': 0.8666666666666667,
 'F1 Score': 0.65,
 'Acurácia': 0.9533333333333334,
 'AUC': 0.7563636363636363}
```

Otimização de Hiperparâmetros com Grid Search e Kernel Polinomial

In [64]:
```
# Cria o modelo
modelo_v4 = svm.SVC(kernel = 'poly')

# Valores para o grid
r_range =  np.array([0.5, 1])
gamma_range =  np.array([0.001, 0.01])
d_range = np.array([2,3, 4])

# Grid de hiperparâmetros
param_grid_poly = dict(gamma = gamma_range, degree = d_range, coef0 = r_range)

# Grid Search
start = time.time()
modelo_v4_grid_search_poly = GridSearchCV(modelo_v4, param_grid_poly, cv = 3)

# Treinamento
modelo_v4_grid_search_poly.fit(X_treino_scaled, y_treino)
end = time.time()
print('Tempo de Treinamento do Modelo com Grid Search:', end - start)

# Acurácia em Treino
print(f"Acurácia em Treinamento: {modelo_v4_grid_search_poly.best_score_ :.2%}")
print("")
print(f"Hiperparâmetros Ideais: {modelo_v4_grid_search_poly.best_params_}")
```

```
Tempo de Treinamento do Modelo com Grid Search: 0.13664960861206055
Acurácia em Treinamento: 94.86%

Hiperparâmetros Ideais: {'coef0': 1.0, 'degree': 3, 'gamma': 0.01}
```

In [65]:
```
# Previsões
previsoes_v4 = modelo_v4_grid_search_poly.predict(X_teste_scaled)
```

In [66]:
```
# Dicionário de métricas e metadados
SVM_dict_v4 = {'Modelo':'SVM',
               'Versão':'4',
               'Kernel':'Polinomial com Dados Padronizados',
               'Precision':precision_score(previsoes_v4, y_teste),
               'Recall':recall_score(previsoes_v4, y_teste),
               'F1 Score':f1_score(previsoes_v4, y_teste),
               'Acurácia':accuracy_score(previsoes_v4, y_teste),
               'AUC':roc_auc_score(y_teste, previsoes_v4)}
```

In [67]:
```
# Print
print("Métricas em Teste:\n")
SVM_dict_v4
```

```
Métricas em Teste:
```

Out[67]:
```
{'Modelo': 'SVM',
 'Versão': '4',
 'Kernel': 'Polinomial com Dados Padronizados',
 'Precision': 0.4,
 'Recall': 0.7692307692307693,
 'F1 Score': 0.5263157894736842,
 'Acurácia': 0.94,
 'AUC': 0.6945454545454546}
```

In [68]:
```
# Concatena todos os dicionários em um dataframe do Pandas
resumo = pd.DataFrame({'SVM_Modelo_1':pd.Series(SVM_dict_v1),
                       'SVM_Modelo_2':pd.Series(SVM_dict_v2),
                       'SVM_Modelo_3':pd.Series(SVM_dict_v3),
                       'SVM_Modelo_4':pd.Series(SVM_dict_v4)})
```

In [69]:
```
# Print
resumo
```

Out[69]:

	SVM_Modelo_1	SVM_Modelo_2	SVM_Modelo_3	SVM_Modelo_4
Modelo	SVM	SVM	SVM	SVM
Versão	1	2	3	4
Kernel	Linear	Linear com Dados Padronizados	RBF com Dados Padronizados	Polinomial com Dados Padronizados
Precision	0.52	0.48	0.52	0.4
Recall	0.764706	0.857143	0.866667	0.769231
F1 Score	0.619048	0.615385	0.65	0.526316
Acurácia	0.946667	0.95	0.953333	0.94
AUC	0.752727	0.736364	0.756364	0.694545

In [ ]:
```
# Documentação oficial do scikit-learn
# https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.Gr
```