Projeto de Machine Learning: Modelo de árvore de decisão para análise histórica de dados na criação de uma ferramenta preditiva e estratégica de renovação de estoque

Dataset utilizado contem os dados de vendas dos produtos da empresa X de 2016 a 2019. O produto A, B e C são importados e levam 3 meses para serem entregues pelo fornecedor na china.

Objetivo

1 - Verificar o hisorico de vendas e estoque disponivel de um produto e analisar se deve ser descontinuado ou adiquirido, tendo em vista as necesidades e estrategias comerciais da empresa.

O Dicionário de Dados

cod: Código sequencial de inserção no banco de dados

numpedido: Numero do documento de vendas

dataemissao:Data em que a venda foi efetuada

mes: Mês da venda

ano: Ano da venda

nomecliente: Nome generico do cliente

produto: Codigo do Produto

qtde: Quantidade vendida do produto

prcunit: Preço unitário do produto

total: Valor total (qtde * prcunit)

custounit: Preço unitário do produto

margem: Custo unitário do produto no dia da venda levando em consideração cambio de dolar, frete, estoque, despesas operacionais e impostos.

estoque: Estoque atual do produto no ato da venda

compra: Indica se deve ser emitido ordem de compra do produto.

Importando as bibliotecas

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Acessando os dados

In [2]:
```
# Acessando banco de dados Mysql
# Obs: Antes será necessário instalar o mysql.connector via commando no cmd (pip install mysql-connec
```

In [3]:
```
from datetime import date
import mysql.connector

db_connection = mysql.connector.connect(host="108.167.132.74", user="vetro057_dcroot", passwd="@dc202

data = pd.read_sql('SELECT * FROM fat19', con=db_connection)

db_connection.commit()
db_connection.close()
```

In [4]:
```
#Exibindo os primeios 5 registros do dataset
data.head(5)
```

Out[4]:

	cod	numpedido	dataemissao	mes	ano	nomecliente	produto	qtde	prcunit	total	custounit	margem	estoque	
0	1	1371.0	03/01/2019	1	2019	CLI3460	5140	240	23.64	5673.60	10.232	13.408	530	C
1	2	1378.0	03/01/2019	1	2019	CLI3433	5140	48	23.64	1134.72	10.232	13.408	563	C
2	3	1388.0	03/01/2019	1	2019	CLI3455	5140	16	23.64	378.24	10.232	13.408	229	C
3	4	1391.0	03/01/2019	1	2019	CLI3587	5140	8	23.64	189.12	10.232	13.408	269	C
4	5	NaN	03/01/2019	1	2019	CLI3472	5140	8	23.64	189.12	9.050	14.590	119	C

In [5]:
```
data.corr()
```

Out[5]:

	cod	numpedido	qtde	prcunit	total	custounit	margem	estoque
cod	1.000000	-0.020464	0.049127	-0.429190	-0.037243	-0.488228	-0.249581	-0.010331
numpedido	-0.020464	1.000000	0.150196	-0.188926	0.023942	0.002257	-0.240841	-0.016446
qtde	0.049127	0.150196	1.000000	-0.174324	0.835629	-0.074672	-0.179145	-0.018724
prcunit	-0.429190	-0.188926	-0.174324	1.000000	0.099779	0.660707	0.881489	0.026942
total	-0.037243	0.023942	0.835629	0.099779	1.000000	0.075882	0.081691	-0.012511
custounit	-0.488228	0.002257	-0.074672	0.660707	0.075882	1.000000	0.227947	0.008715
margem	-0.249581	-0.240841	-0.179145	0.881489	0.081691	0.227947	1.000000	0.029465
estoque	-0.010331	-0.016446	-0.018724	0.026942	-0.012511	0.008715	0.029465	1.000000

In [6]:
```
#convertendo alguns tipos de dados:

data['mes'] = data['mes'].astype(int)
data['ano'] = data['ano'].astype(int)
data['produto'] = data['produto'].astype(int)
```

Exploração dos dados

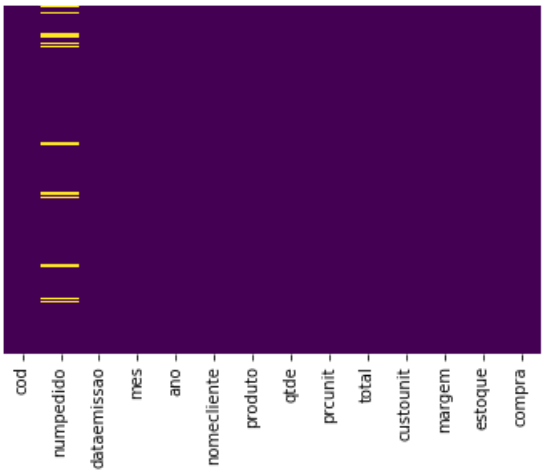
In [7]:
```
# Informação gerais.
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 14 columns):
cod             2823 non-null int64
numpedido       2609 non-null float64
dataemissao     2823 non-null object
mes             2823 non-null int32
ano             2823 non-null int32
nomecliente     2823 non-null object
produto         2823 non-null int32
qtde            2823 non-null int64
prcunit         2823 non-null float64
total           2823 non-null float64
custounit       2823 non-null float64
margem          2823 non-null float64
estoque         2823 non-null int64
compra          2823 non-null object
dtypes: float64(5), int32(3), int64(3), object(3)
memory usage: 275.8+ KB
```

In [8]:
```
# Analisando se existe alguma coluna com dados NULL utilizando grafico de temperatura.
```

In [9]:
```
sns.heatmap(data.isnull(), yticklabels=False,cbar=False,cmap='viridis')
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x261c1917f60>
```



In [10]:
```
#Removendo a coluna numpedido por conter alguns valores nulos e não ser um feature importante para a
data.drop(['numpedido'],axis=1, inplace=True)
data.head(5)
```

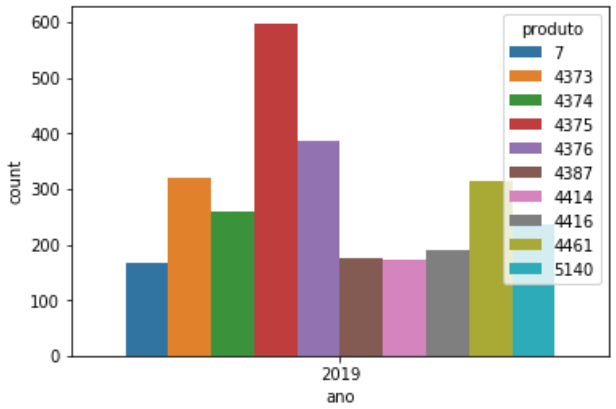
Out[10]:

	cod	dataemissao	mes	ano	nomecliente	produto	qtde	prcunit	total	custounit	margem	estoque	compra
0	1	03/01/2019	1	2019	CLI3460	5140	240	23.64	5673.60	10.232	13.408	530	COMPRA
1	2	03/01/2019	1	2019	CLI3433	5140	48	23.64	1134.72	10.232	13.408	563	COMPRA
2	3	03/01/2019	1	2019	CLI3455	5140	16	23.64	378.24	10.232	13.408	229	COMPRA
3	4	03/01/2019	1	2019	CLI3587	5140	8	23.64	189.12	10.232	13.408	269	COMPRA
4	5	03/01/2019	1	2019	CLI3472	5140	8	23.64	189.12	9.050	14.590	119	COMPRA

In [11]:
```
# Vendas por Produto
```

In [12]:
```
sns.countplot(x='ano',hue='produto',data=data)
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x261c174d8d0>
```



In [13]:
```
# Clintes que compraram no perído ordenado por quantidade comprada, idenficando os 10 princiais.
```

In [14]:
```
by_cliente_produto = data[['nomecliente','produto','qtde']].groupby(["nomecliente"]).sum()
topcliente = by_cliente_produto.sort_values(by=["qtde"],ascending=False)
topcliente.head(10)
```

Out[14]:

nomecliente	produto	qtde
CLI1889	1360491	65343
CLI3562	453558	30997
CLI2259	558528	11361
CLI3436	248647	9814
CLI3460	275835	9256
CLI3433	394846	9111
CLI761	26301	9086
CLI3454	106948	7903
CLI3226	381848	7526
CLI3060	292781	6924

In [15]:
```
# Calculando a media de venda por mes dos produtos
```

In [16]:
```
by_produto_ano = data[['produto','mes','qtde']].groupby(["produto","mes"]).mean()
by_produto_ano.sort_values(by='produto',ascending=True)
```

Out[16]:

		qtde
produto	**mes**	
7	**1**	235.066667
	11	238.600000
	10	56.272727
	8	50.000000
	7	54.074074
	9	252.000000
	5	22.368421
	4	229.900000
	3	20.800000
	2	14.916667
	6	48.750000
4373	**8**	89.565217
	12	134.000000
	11	140.466667
	10	98.700000
	9	87.588235
	7	176.371429
	4	71.222222
	5	80.516129
	3	95.562500
	2	138.973684
	1	59.468750
	6	66.611111
4374	**8**	95.764706
	12	173.111111
	11	93.888889
	10	91.333333
	9	312.833333
	7	56.750000
	3	67.100000
...	**...**	...
4416	**7**	116.125000
	3	101.588235
	5	71.176471
	4	85.750000
	2	250.812500
	1	121.923077
	6	147.714286
4461	**12**	59.125000
	11	58.800000
	9	72.722222
	8	41.333333
	7	50.129032
	10	55.875000
	4	34.785714

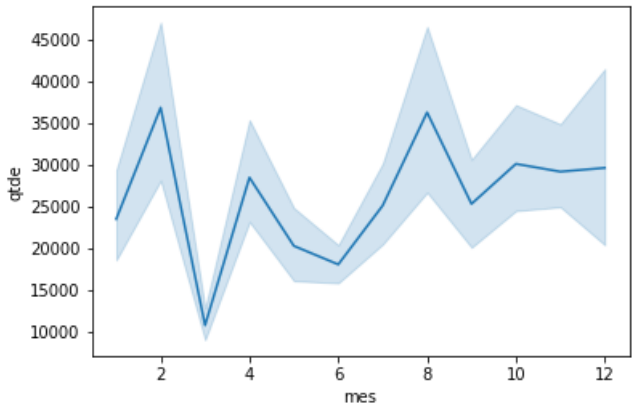
		qtde
produto	**mes**	
	3	28.230769
	2	71.159091
	1	71.238095
	5	1.000000
5140	**10**	119.666667
	9	91.071429
	8	4.000000
	7	1.000000
	6	68.509091
	4	4.285714
	3	62.909091
	2	54.925926
	1	56.195122
	11	70.800000
	5	62.000000
	12	127.351351

118 rows × 1 columns

In [17]:
```
# Grafico de vendas por mes(todos os anos)
```

In [18]:
```
sns.lineplot(x="mes", y="qtde", data=data, estimator=np.sum)
```

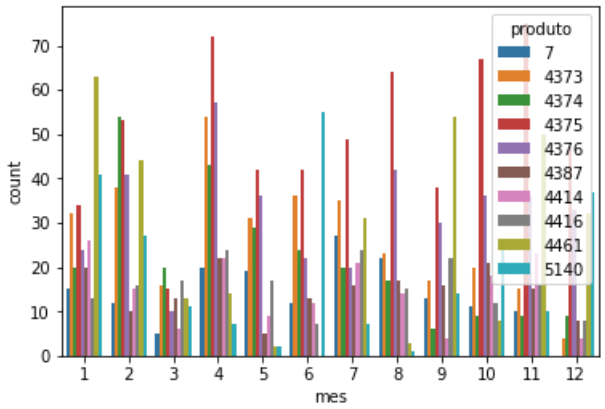
Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x261c1707e80>
```



In [19]:
```
# Vendas mensal(todos os anos) e Produto
```

In [20]:
```
sns.countplot(x='mes',hue='produto',data=data)
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x261c1825a90>
```



Tratando o dataset para treino

In [21]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt # biblioteca de visualização utilizada pelo pandas e pelo seaborn
import seaborn as sns # biblioteca de visualização com mais opções de gráficos
 #comando necessário para que as imagens sejam exibidas aqui mesmo no notebook
%matplotlib inline
```

In [22]:
```
df = data
df.head(5)
```

Out[22]:

	cod	dataemissao	mes	ano	nomecliente	produto	qtde	prcunit	total	custounit	margem	estoque	compra
0	1	03/01/2019	1	2019	CLI3460	5140	240	23.64	5673.60	10.232	13.408	530	COMPRA
1	2	03/01/2019	1	2019	CLI3433	5140	48	23.64	1134.72	10.232	13.408	563	COMPRA
2	3	03/01/2019	1	2019	CLI3455	5140	16	23.64	378.24	10.232	13.408	229	COMPRA
3	4	03/01/2019	1	2019	CLI3587	5140	8	23.64	189.12	10.232	13.408	269	COMPRA
4	5	03/01/2019	1	2019	CLI3472	5140	8	23.64	189.12	9.050	14.590	119	COMPRA

In [23]:
```
# Eliminar os atributos que nao influencia na analise
```

In [24]:
```
df.drop(['cod','dataemissao','nomecliente'],axis=1, inplace=True)
df.head(5)
```

Out[24]:

	mes	ano	produto	qtde	prcunit	total	custounit	margem	estoque	compra
0	1	2019	5140	240	23.64	5673.60	10.232	13.408	530	COMPRA
1	1	2019	5140	48	23.64	1134.72	10.232	13.408	563	COMPRA
2	1	2019	5140	16	23.64	378.24	10.232	13.408	229	COMPRA
3	1	2019	5140	8	23.64	189.12	10.232	13.408	269	COMPRA
4	1	2019	5140	8	23.64	189.12	9.050	14.590	119	COMPRA

In [25]:
```
# One Hot Encode -  Subtistuir valor da coluna Compra( Compra = 1 e NaoCompra = 0)
```

In [26]:
```
df["compra"] = df["compra"].replace("COMPRA", "1")
df["compra"] = df["compra"].replace("NAOCOMPRA", "0")
```

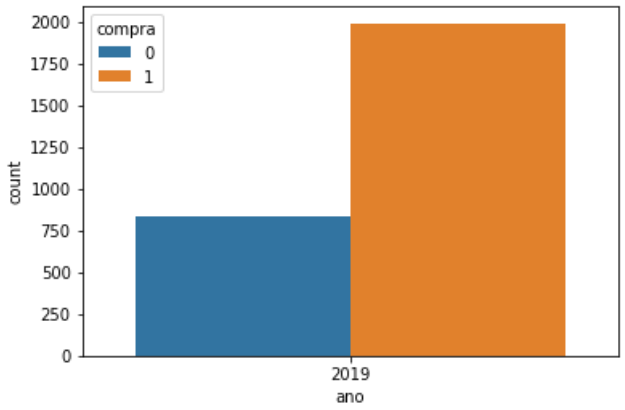
In [27]:
```
#Verificando a coluna compra que antes estava nominal (COMPRA E NÃO) e agora está binária (0 e 1)
df.head(5)
```

Out[27]:

	mes	ano	produto	qtde	prcunit	total	custounit	margem	estoque	compra
0	1	2019	5140	240	23.64	5673.60	10.232	13.408	530	1
1	1	2019	5140	48	23.64	1134.72	10.232	13.408	563	1
2	1	2019	5140	16	23.64	378.24	10.232	13.408	229	1
3	1	2019	5140	8	23.64	189.12	10.232	13.408	269	1
4	1	2019	5140	8	23.64	189.12	9.050	14.590	119	1

In [28]:
```
# Grafico mostrando a relação de decisões de compra ou não no dataset analisado.
sns.countplot(x='ano',hue='compra',data=df)
```

Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x261c1e596d8>
```



2. Treinar o Classificador

In [29]:
```
import itertools
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

In [30]:
```
train = df # Transferindo o dataset DF para train, para efetuar os treinos
```

In [31]:
```
X_train, X_test, y_train, y_test = train_test_split(train.drop(['compra'],axis=1),
                                                    train['compra'], test_size=0.30,
                                                    random_state=101)
```

In [32]:
```
# Criar um objeto do classificador DecisionTreeClassifier()
dtc = DecisionTreeClassifier(max_depth=3)
```

In [33]:
```
# Treinar o modelo dtc chamando a função fit
dtc.fit(X_train, y_train)
```

Out[33]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [34]:
```
# Fazer as predições passanto o X_TEST
predictions = dtc.predict(X_test)
```

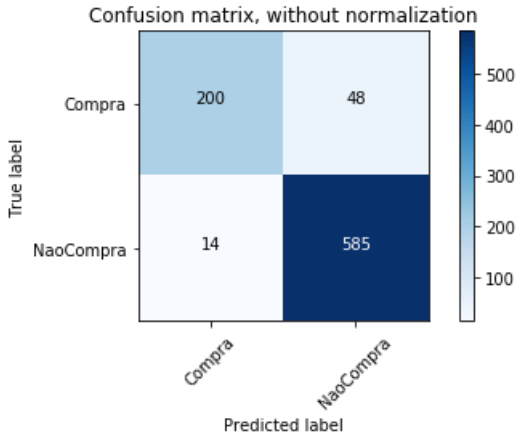
In [35]:
```
# Matrix de confusão
cnf_matrix = confusion_matrix(y_test, predictions)
cnf_matrix
```

Out[35]:
```
array([[200,  48],
       [ 14, 585]], dtype=int64)
```

In [36]:
```
#Plotar matriz de confusão
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

In [37]:
```
plot_confusion_matrix(cnf_matrix, classes=['Compra','NaoCompra'],
                      title='Confusion matrix, without normalization')
```

```
Confusion matrix, without normalization
[[200  48]
 [ 14 585]]
```



In [38]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.93      0.81      0.87       248
           1       0.92      0.98      0.95       599

   micro avg       0.93      0.93      0.93       847
   macro avg       0.93      0.89      0.91       847
weighted avg       0.93      0.93      0.93       847
```

Reavaliando o Modelo com validação cruzada

In [39]:
```
# validação cruzada para verificar se o modelo está em overfit
#ordenando os dados de proposito para dificultar o trabalho do modelo
```

In [40]:
```
train2 = df.sort_values("compra", ascending=True)  # Transferindo o dataset DF para train, para efetu
train2.head(5)
```

Out[40]:

	mes	ano	produto	qtde	prcunit	total	custounit	margem	estoque	compra
2822	11	2019	7	1000	15.58	15580.00	9.06231	6.51769	367	0
796	4	2019	4414	77	9.31	716.87	4.38227	4.92773	219	0
2265	6	2019	4374	121	10.58	1280.18	5.05669	5.52331	37	0
800	4	2019	4414	1	3.30	1.65	4.54727	-1.24727	510	0
803	4	2019	4414	33	3.30	108.90	4.54727	-1.24727	564	0

In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(train2.drop(['compra'],axis=1),
                                                    train2['compra'], test_size=0.30,
                                                    random_state=101)
```

In [42]:
```
from sklearn.model_selection import cross_validate

results = cross_validate(dtc, X_train, y_train, cv = 10, return_train_score=False)
media = results['test_score'].mean()
desvio_padrao = results['test_score'].std()
print("Accuracy com cross validation, 10 = [%.2f, %.2f]" % ((media - 2 * desvio_padrao)*100,
                                                           (media + 2 * desvio_padrao) * 100))
```

```
Accuracy com cross validation, 10 = [89.08, 96.95]
```

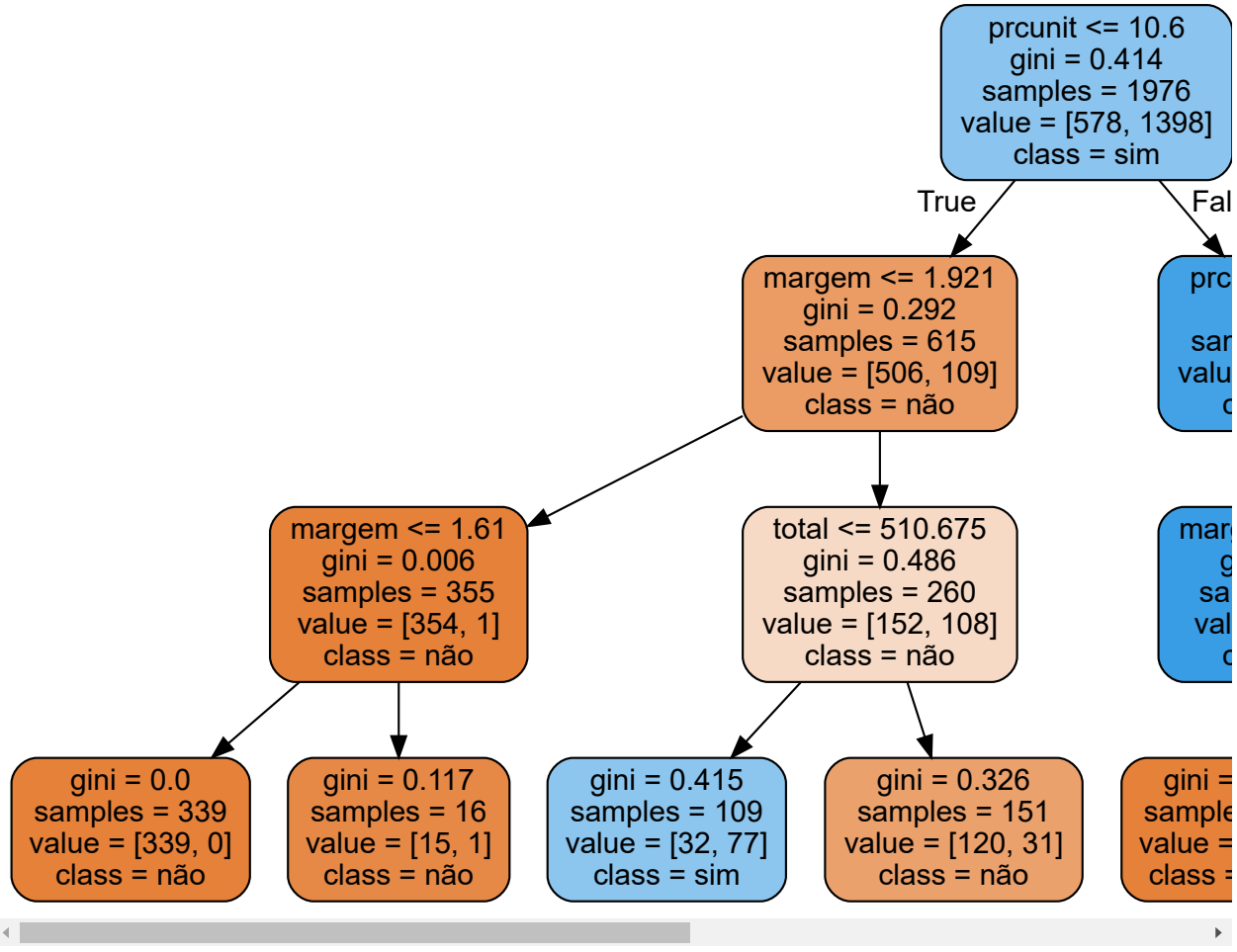
In [43]:
```
#Variando de 89% a 96%, o cross validation confirmou a eficiência do modelo.

#imprimir arvore de decisao
```

Treino final e arvovre de de decisão

In [44]:
```
from sklearn.tree import export_graphviz
import graphviz

#treina o modelo final
dtc.fit(X_train, y_train)
features = X_train.columns
dot_data = export_graphviz(dtc, out_file=None, filled=True, rounded=True,
                class_names=["não", "sim"],
                feature_names =  features)

graph = graphviz.Source(dot_data)
```

In [45]:
```
graph
```

Out[45]:

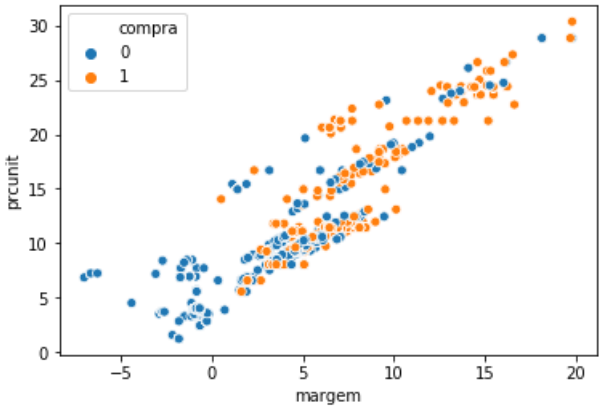


In [46]:
```
# Rank dos tributos mais relevantes
```

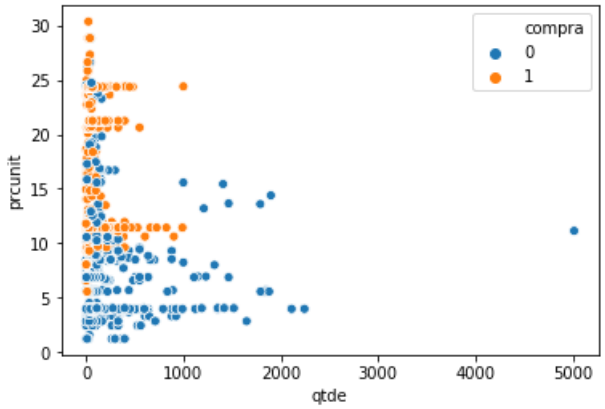
In [48]:
```
# Plotar atributos mais relevantes
```

In [49]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

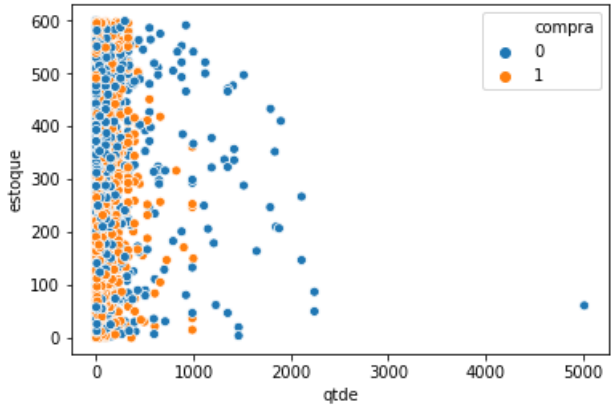
In [50]:
```
ax = sns.scatterplot(x=train["margem"], y=train["prcunit"], hue="compra",data=train)
```



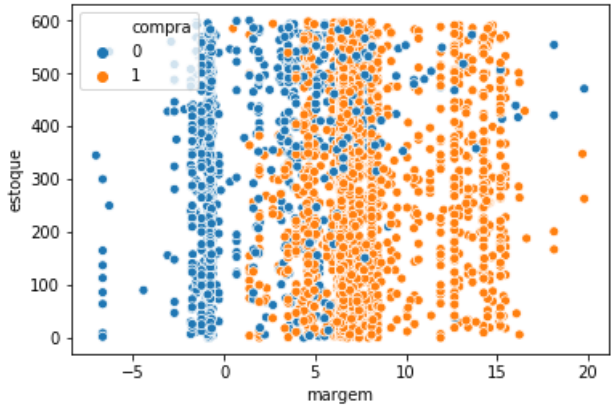
In [51]:
```
ax = sns.scatterplot(x=train["qtde"], y=train["prcunit"], hue="compra",data=train)
```



In [52]:
```
ax = sns.scatterplot(x=train["qtde"], y=train["estoque"], hue="compra",data=train)
```



In [53]:
```
ax = sns.scatterplot(x=train["margem"], y=train["estoque"], hue="compra",data=train)
```



Teste unitário do modelo.

In [54]:
```
train.head(0)
```

Out[54]:

mes	ano	produto	qtde	prcunit	total	custounit	margem	estoque	compra

In [55]:
```
# Criando produtos ficticios contendo informações dos produtos a ser analisado pelo modelo.
produto1 = [1,2019,4391,11,4.11,45.21,6.4655,-2.3555,260]
produto2 = [8,2019,7,11,4.11,45.21,8.55,4.52,160,]
```

In [56]:
```
#Submetendo o Produto1 para o modelo analisar se comprar ou não. [0] = Nao compra  [1] = Compra
dtc.predict([produto1])
```

Out[56]:
```
array(['0'], dtype=object)
```

In [57]:
```
#Submetendo o Produto2 para o modelo analisar se comprar ou não. [0] = Nao compra  [1] = Compra
dtc.predict([produto2])
```

Out[57]:
```
array(['1'], dtype=object)
```

Teste com dados reais

In [58]:
```
#carregando dados para teste:
dt_teste = pd.read_csv('Teste-01.csv')
print("Qtde Registros: ",len(dt_teste))
```

```
Qtde Registros:  135
```

In [59]:
```
dt_teste.head(3)
```

Out[59]:

	cod	numpedido	dataemissao	mes	ano	nomecliente	produto	qtde	prcunit	total	custounit	margem	estoque
0	2456	12268.0	4/11/2019	4	2019	CLI3562	4373	1	3.88	1.94	318.616	0.693845	522
1	2457	2214.0	4/11/2019	4	2019	CLI3060	4373	1	8.05	4.03	299.216	505.785000	432
2	2458	8220.0	4/13/2019	4	2019	CLI2259	4373	132	11.97	1580.04	299.216	897.785000	361

In [60]:
```
#tratando a base de teste:
#convertendo tipos
dt_teste['mes'] = dt_teste['mes'].astype(int)
dt_teste['ano'] = dt_teste['ano'].astype(int)
dt_teste['produto'] = dt_teste['produto'].astype(int)

#excluindo colunas
dt_teste.drop(['numpedido','cod','dataemissao','nomecliente'],axis=1, inplace=True)

#one hot enconding
dt_teste["compra"] = dt_teste["compra"].replace("COMPRA", "1")
dt_teste["compra"] = dt_teste["compra"].replace("NAOCOMPRA", "0")

dt_teste.head(3)
```

Out[60]:

	mes	ano	produto	qtde	prcunit	total	custounit	margem	estoque	compra
0	4	2019	4373	1	3.88	1.94	318.616	0.693845	522	0
1	4	2019	4373	1	8.05	4.03	299.216	505.785000	432	0
2	4	2019	4373	132	11.97	1580.04	299.216	897.785000	361	1

In [61]:
```
#array contendo o resultado já conhecido do dataset de teste
testes_resultados = dt_teste['compra'].values
testes_resultados
```

Out[61]:
```
array(['0', '0', '1', '1', '1', '1', '1', '1', '1', '1', '0', '0', '1',
       '0', '0', '1', '1', '1', '1', '1', '0', '1', '0', '1', '0', '0',
       '0', '0', '1', '0', '0', '1', '0', '0', '0', '1', '1', '1', '1',
       '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0',
       '1', '0', '0', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
       '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '0', '1', '0',
       '0', '1', '0', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
       '1', '1', '1', '1', '0', '0', '1', '0', '1', '1', '0', '0', '1',
       '0', '1', '1', '1', '0', '1', '1', '1', '1', '1', '0', '1', '0',
       '0', '1', '1', '1', '1', '1', '1', '1', '0', '0', '0', '1', '1',
       '1', '1', '1', '1', '1'], dtype=object)
```

In [62]:
```
#dropando a coluna COMPRA para enviar o dt_teste_pred para o modelo
dt_teste_pred = dt_teste
dt_teste_pred.drop(['compra'],axis=1, inplace=True)
dt_teste_pred.head(3)
```

Out[62]:

	mes	ano	produto	qtde	prcunit	total	custounit	margem	estoque
0	4	2019	4373	1	3.88	1.94	318.616	0.693845	522
1	4	2019	4373	1	8.05	4.03	299.216	505.785000	432
2	4	2019	4373	132	11.97	1580.04	299.216	897.785000	361

In [63]:
```
# efetuando predições
predicoes = dtc.predict(dt_teste_pred)
predicoes
```

Out[63]:
```
array(['0', '1', '1', '0', '1', '1', '1', '1', '1', '1', '0', '0', '1',
       '0', '0', '1', '1', '1', '1', '1', '0', '1', '0', '1', '1', '1',
       '0', '0', '1', '0', '0', '1', '0', '0', '0', '1', '1', '1', '1',
       '1', '1', '1', '1', '1', '0', '1', '0', '1', '1', '0', '0', '0',
       '1', '0', '0', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
       '1', '1', '1', '1', '1', '1', '1', '1', '1', '0', '1', '0', '1',
       '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
       '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0',
       '1', '1', '1', '1', '0', '1', '1', '1', '1', '1', '0', '1', '0',
       '0', '1', '1', '1', '1', '1', '1', '1', '0', '0', '0', '1', '1',
       '1', '1', '1', '1', '1'], dtype=object)
```

In [64]:
```
# Calculando taxa de acertos
acertos = (predicoes == testes_resultados).sum()
total_registros = len(testes_resultados)
taxa_acertos = acertos/total_registros
print("taxa de acerto = ",taxa_acertos *100, "%")
```

```
taxa de acerto =  91.85185185185185 %
```

Produto Final

In [65]:
```
#Exportando o modelo e colocando em produção
from sklearn.externals import joblib
joblib.dump(dtc, 'decision_tree.pk1')
```

Out[65]:
```
['decision_tree.pk1']
```

In [68]:
```
#Criando API para prover o serviço de classificação
from flask import Flask, jsonify, request

# [1] importo o deserializador
from sklearn.externals import joblib

# [2] Carrego a classe de predição do diretório local
dtc = joblib.load('decision_tree.pk1')

app = Flask(__name__)

@app.route('/compra_predictor')
def compra_predictor():

    # [3] Recupero as informações de uma Flor
    mes = int(request.args.get('mes'))
    ano = int(request.args.get('ano'))
    produto = int(request.args.get('produto'))
    qtde = int(request.args.get('qtde'))
    prcunit = float(request.args.get('prcunit'))
    total = float(request.args.get('total'))
    custounit = float(request.args.get('custounit'))
    margem = float(request.args.get('margem'))
    estoque = int(request.args.get('estoque'))

    event = [mes, ano, produto, qtde, prcunit, total, custounit, margem, estoque ]
    target_names = ['NaoCompra', 'Compra']

    # [4] Realiza predição com base no evento
    prediction = dtc.predict([event])[0]

    res = int(prediction[0])
    result = target_names[res]

    return jsonify(result), 200

app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on http://127.0.0.1:5000/ (http://127.0.0.1:5000/) (Press CTRL+C to quit)
127.0.0.1 - - [20/Jun/2020 15:15:01] "GET /compra_predictor?mes=8&ano=2019&produto=7&qtde=11&prcunit
=4.11&total=45.21&custounit=8.55&margem=4.52&estoque=160 HTTP/1.1" 200 -
```

In [67]:
```
#Testando a chamada a API
```

In [ ]:
```
http://127.0.0.1:5000/compra_predictor?mes=8&ano=2019&produto=7&qtde=11&prcunit=4.11&total=45.21&cust
```

In [ ]:
```
http://127.0.0.1:5000/compra_predictor?mes=1&ano=2019&produto=4391&qtde=11&prcunit=4.11&total=45.21&c
```