Desafio-Ame-DaniloMoralesTeixeira

September 18, 2019

1 Solução do desafio Ame

Solução desenvolvida por Danilo Morales Teixeira
18/09/2019
Importando as bibliotecas fundamentais para inicio do estudo e exploração inicial dos dados

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Carregando a base de dados presente no arquivo problem1_dataset.csv utilizando a biblioteca Pandas

[2]:
```
dataset = pd.read_csv('problem1_dataset.csv')
```

Determinando número de linhas e colunas do dataset

[3]:
```
linhas = dataset.shape[0]
colunas = dataset.shape[1]
print("Este dataset possui {} linhas e {} colunas".format(linhas,colunas))
```

```
Este dataset possui 180275 linhas e 21 colunas
```

Exibindo o nome das colunas para facilitar a análise exploratória de dados (EDA)

[4]:
```
colunas = dataset.columns
print(colunas)
```

```
Index(['ITEM_ID', 'ALTURA', 'CAPACIDADE_(L)', 'COMPOSICAO', 'COR', 'FORMATO',
       'LARGURA', 'MARCA', 'PARA_LAVA_LOUCAS', 'PARA_MICRO_ONDAS', 'PESO',
       'PROFUNDIDADE', 'TEMPO_GARANTIA', 'TEM_FERRO_FUNDIDO', 'TEM_GRELHA',
       'TEM_TAMPA', 'TIPO_PRODUTO', 'TIPO_WOK', 'SESSION_ID', 'ITEM_PRICE',
       'INTERESTED'],
      dtype='object')
```

Exibindo as cinco primeiras linhas do dataset

[5]:
```
print(dataset.head(5))
```

```
     ITEM_ID  ALTURA  CAPACIDADE_(L) COMPOSICAO       COR  FORMATO  LARGURA  \
0  264220456    30.5             NaN   ALUMINIO     VINHO      NaN     14.0
1  238630912    22.0             NaN   ALUMINIO  COLORIDO      NaN     24.0
2  218228122    24.0             NaN       INOX      INOX      NaN     20.0
3  253661510    49.5             6.0   ALUMINIO  VERMELHO  REDONDO     41.5
4  253661510    49.5             6.0   ALUMINIO  VERMELHO  REDONDO     41.5

        MARCA PARA_LAVA_LOUCAS PARA_MICRO_ONDAS  ...  PROFUNDIDADE  \
0  LA CUISINE             NaN              NaN  ...          50.0
1  TRAMONTINA              No               no  ...          40.0
2  LA CUISINE             Yes               no  ...          20.0
3  TRAMONTINA             Yes              NaN  ...          47.0
4  TRAMONTINA             Yes              NaN  ...          47.0

   TEMPO_GARANTIA  TEM_FERRO_FUNDIDO TEM_GRELHA TEM_TAMPA  TIPO_PRODUTO  \
0             3.0                NAO        SIM       1.0        PANELA
1            12.0                NAO        NAO       1.0   PIPOQUEIRA
2             3.0                NAO        NAO       1.0  ESPAGUETEIRA
3             NaN                NAO        NAO       1.0   PIPOQUEIRA
4             NaN                NAO        NAO       1.0   PIPOQUEIRA

  TIPO_WOK  SESSION_ID  ITEM_PRICE  INTERESTED
0      NAO   86.709770  199.990000         0.0
1      NAO   73.156401  105.112581         0.0
2      NAO  952.331024  139.990000         0.0
3      NAO  637.759106  103.293333         1.0
4      NAO  478.531428  103.330242         0.0

[5 rows x 21 columns]
```

Analisando os dados das colunas, podemos verificar que as colunas ITEM_ID e SESSION_ID podem ser removidas uma vez que não fornecem informações úteis para o nosso estudo

Removendo as colunas ITEM_ID e SESSION_ID

[6]:
```
dataset = dataset.drop(['ITEM_ID','SESSION_ID'],axis=1)
```

Exibindo novamente as cinco primeiras linhas para verificar se as duas colunas foram removidas

[7]:
```
print(dataset.head(5))
```

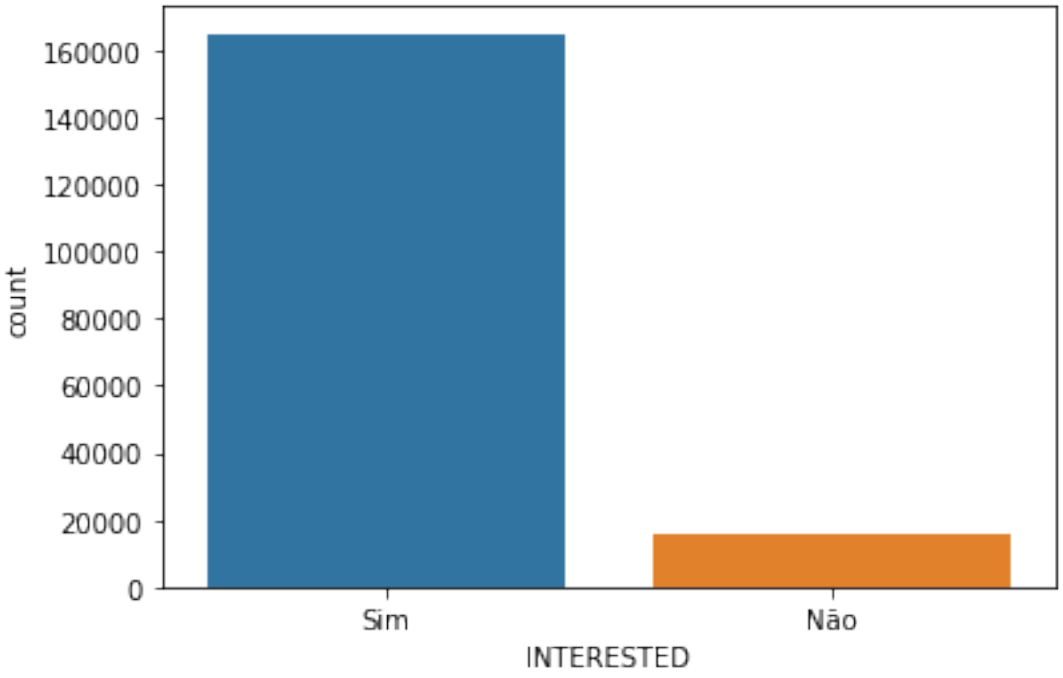
```
   ALTURA  CAPACIDADE_(L) COMPOSICAO       COR  FORMATO  LARGURA       MARCA  \
0    30.5             NaN   ALUMINIO     VINHO      NaN     14.0  LA CUISINE
1    22.0             NaN   ALUMINIO  COLORIDO      NaN     24.0  TRAMONTINA
2    24.0             NaN       INOX      INOX      NaN     20.0  LA CUISINE
3    49.5             6.0   ALUMINIO  VERMELHO  REDONDO     41.5  TRAMONTINA
4    49.5             6.0   ALUMINIO  VERMELHO  REDONDO     41.5  TRAMONTINA

  PARA_LAVA_LOUCAS PARA_MICRO_ONDAS   PESO  PROFUNDIDADE  TEMPO_GARANTIA  \
0              NaN              NaN    NaN          50.0             3.0
```

```
1              No              no  150.0          40.0           12.0
2             Yes              no  190.0          20.0            3.0
3             Yes             NaN  120.0          47.0            NaN
4             Yes             NaN  120.0          47.0            NaN

  TEM_FERRO_FUNDIDO TEM_GRELHA  TEM_TAMPA  TIPO_PRODUTO TIPO_WOK  ITEM_PRICE  \
0               NAO        SIM        1.0        PANELA      NAO  199.990000
1               NAO        NAO        1.0    PIPOQUEIRA      NAO  105.112581
2               NAO        NAO        1.0  ESPAGUETEIRA      NAO  139.990000
3               NAO        NAO        1.0    PIPOQUEIRA      NAO  103.293333
4               NAO        NAO        1.0    PIPOQUEIRA      NAO  103.330242

   INTERESTED
0         0.0
1         0.0
2         0.0
3         1.0
4         0.0
```

Analisando a distribuição de pessoas interessadas ou não nos produtos. Os valores 0 e 1 serão convertidos para os valores Sim e Não

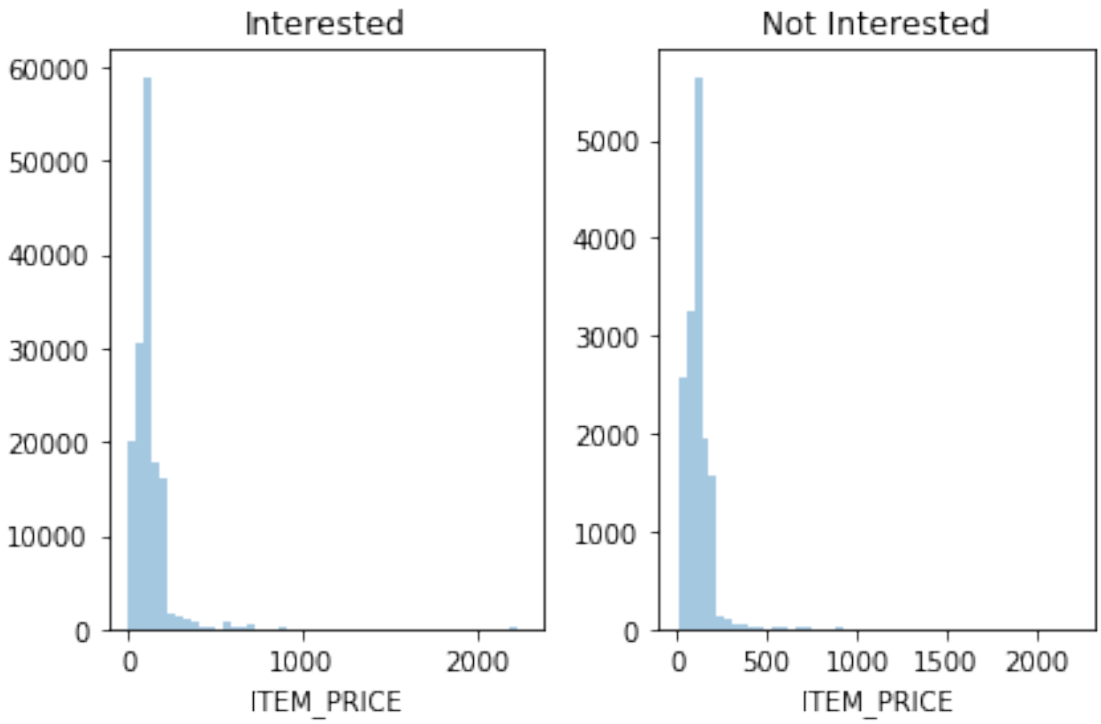
[8]:
```
dataset_tmp = dataset.copy()
dataset_tmp['INTERESTED'] = dataset['INTERESTED'].map({0 : 'Sim', 1 : 'Não'})
```

[9]:
```
sns.countplot(x='INTERESTED',data=dataset_tmp);
```



Analisando a distribuição dos preços

[10]:
```
plt.subplot(1,2,1)
sns.distplot(dataset[dataset['INTERESTED'] == 0]['ITEM_PRICE'].
 ↪dropna(),kde=False,label='Interessante');
plt.title('Interested')
plt.subplot(1,2,2)
sns.distplot(dataset[dataset['INTERESTED'] == 1]['ITEM_PRICE'].
 ↪dropna(),kde=False,label='Não intersante');
plt.title('Not Interested')
plt.tight_layout()
```

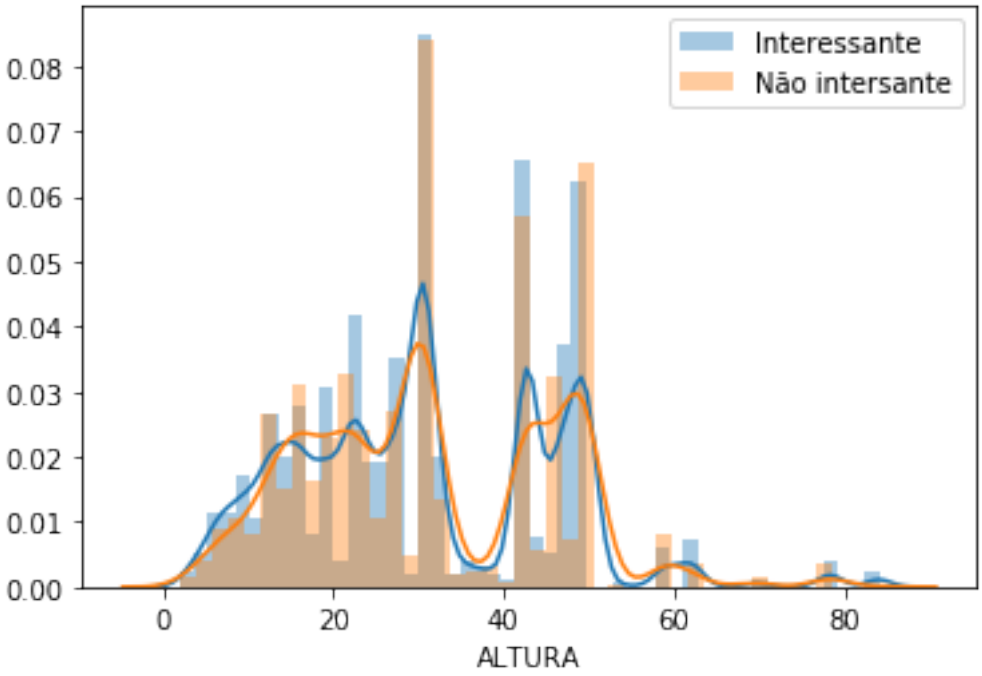


2 Observamos que os produtos que custam menos do que 500 tem níveis de interesse e desinteresse parecidos. Produtos custando acima de 100 tiveram baixa procura.

Analisando a distribuição de altura dos produtos. Iremos utilizar uma distribuição normalizada para melhor compararmos dois dois casos

[11]:
```
sns.distplot(dataset[dataset['INTERESTED'] == 0]['ALTURA'].
 ↪dropna(),kde=True,label='Interessante');
sns.distplot(dataset[dataset['INTERESTED'] == 1]['ALTURA'].
 ↪dropna(),kde=True,label='Não intersante');
```

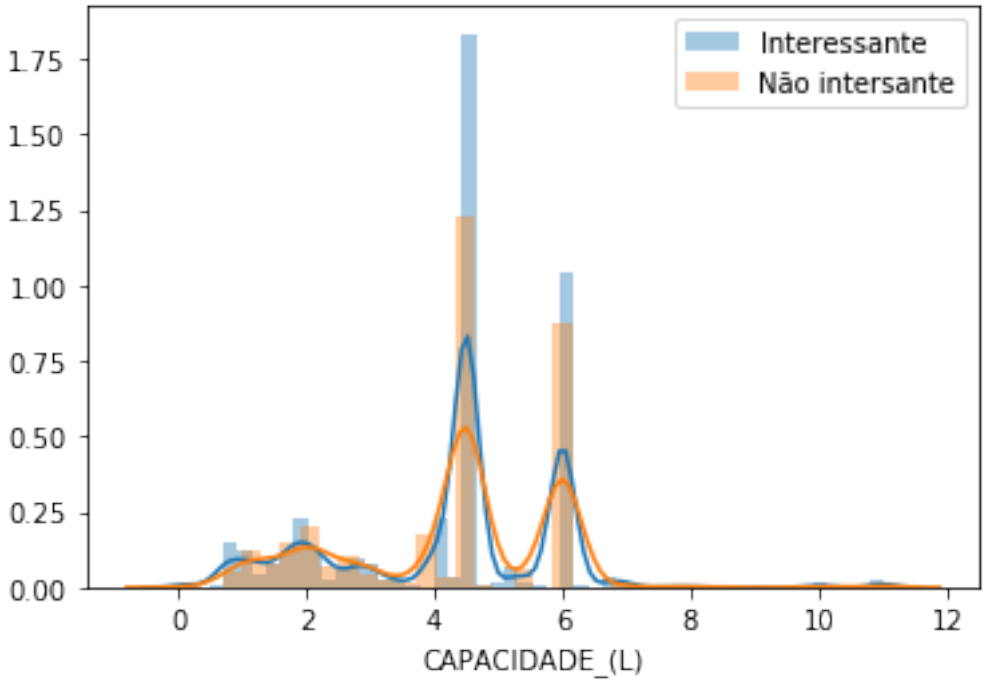
```
plt.legend(loc='best');
```



3 Observamos da distribuição normalizada que a altura do produto não alterou o interesse do cliente

Analisando a distribuição da capacidade dos produtos. Iremos utilizar uma distribuição normalizada para melhor compararmos dois dois casos

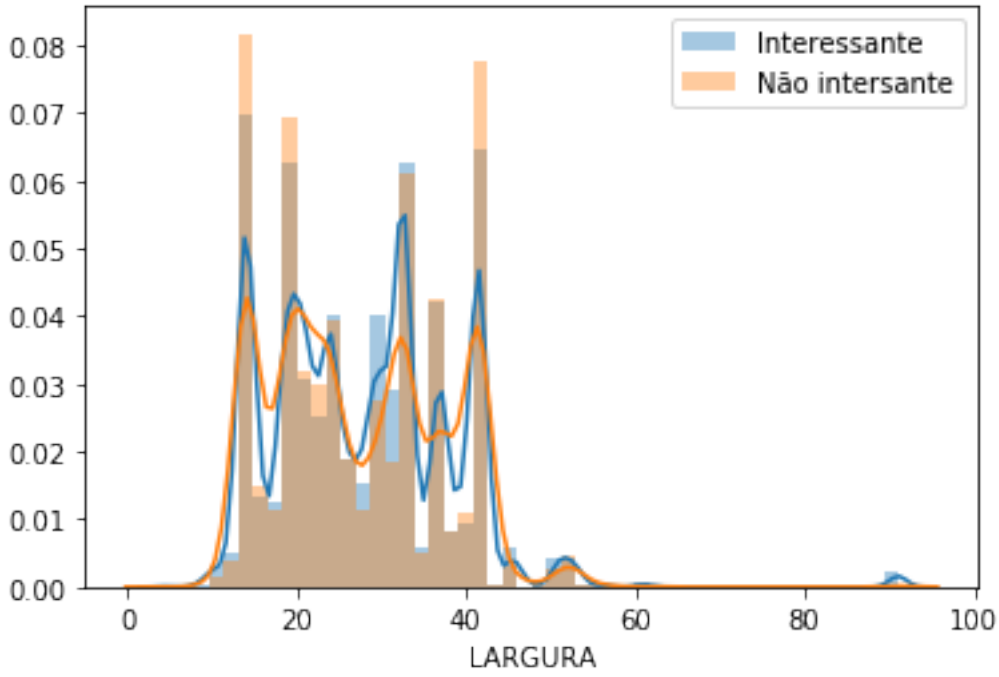
[12]:
```
sns.distplot(dataset[dataset['INTERESTED'] == 0]['CAPACIDADE_(L)'].
 ↪dropna(),kde=True,label='Interessante');
sns.distplot(dataset[dataset['INTERESTED'] == 1]['CAPACIDADE_(L)'].
 ↪dropna(),kde=True,label='Não intersante');
plt.legend(loc='best');
```



4 Observamos que a capacidade também não afetou o interesse pelo produto

Analisando a distribuição da largura dos produtos. Iremos utilizar uma distribuição normalizada para melhor compararmos dois dois casos

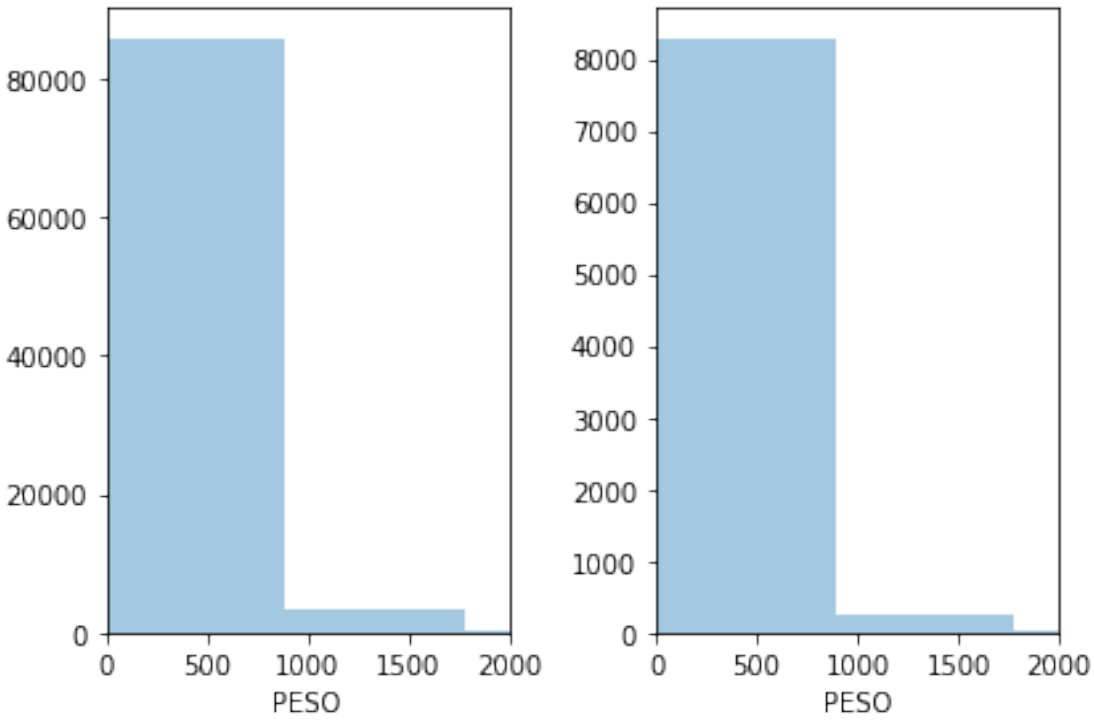
[13]:
```
sns.distplot(dataset[dataset['INTERESTED'] == 0]['LARGURA'].
 ↪dropna(),kde=True,label='Interessante');
sns.distplot(dataset[dataset['INTERESTED'] == 1]['LARGURA'].
 ↪dropna(),kde=True,label='Não intersante');
plt.legend(loc='best');
```



5 Observamos que a largura também não afetou o interesse pelo produto

Analisando a distribuição dos pesos dos produtos

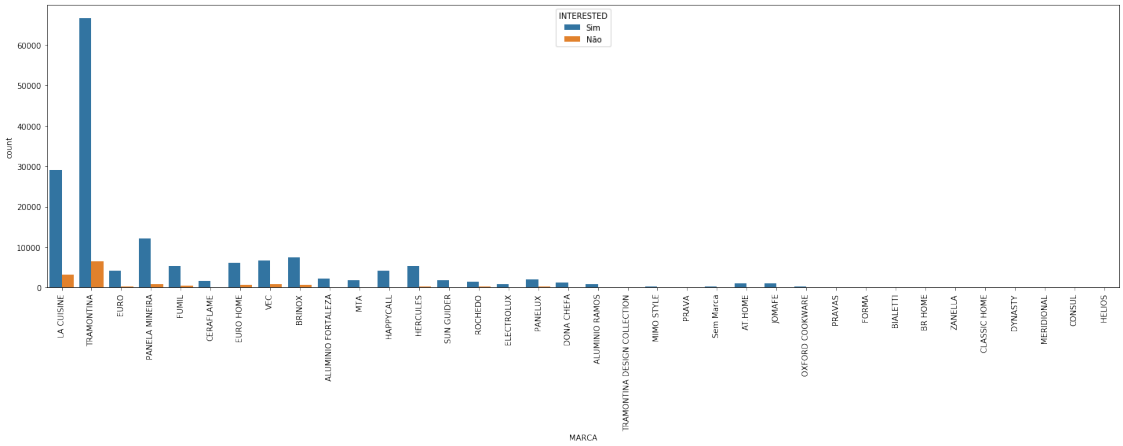
[14]:
```
plt.subplot(1,2,1)
sns.distplot(dataset[dataset['INTERESTED'] == 0]['PESO'].
 ↪dropna(),kde=False,label='Interessante');
plt.xlim([0,2000])
plt.subplot(1,2,2)
sns.distplot(dataset[dataset['INTERESTED'] == 1]['PESO'].
 ↪dropna(),kde=False,label='Não intersante');
plt.xlim([0,2000])
plt.tight_layout()
```



6 Observamos que o peso do produto não afetou o grau de interesse

Analisando o grau de interesse das diferentes marcas. Convertendo os NaNs para sem marca definida

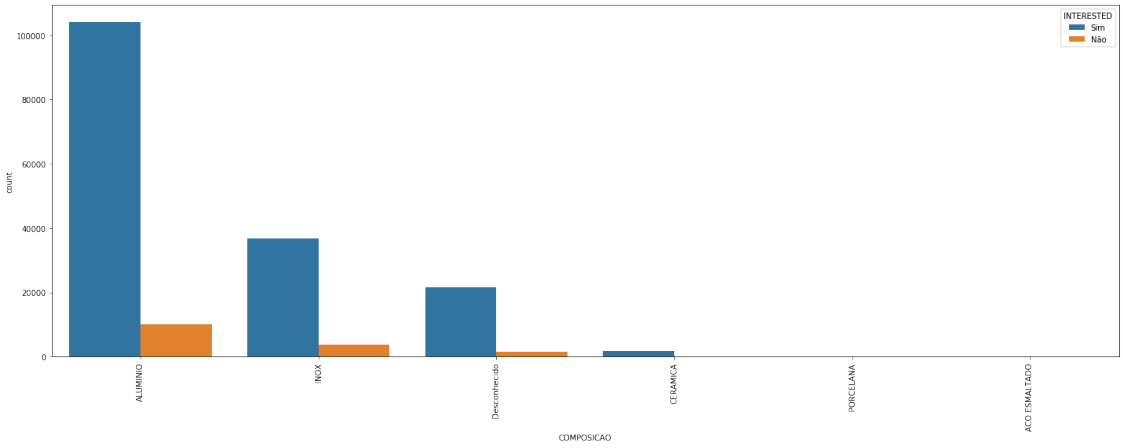
[15]:
```
dataset_tmp['MARCA'] = dataset_tmp['MARCA'].fillna('Sem Marca')
fig,ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'MARCA', data = dataset_tmp, hue='INTERESTED')
plt.xticks(rotation='vertical')
plt.tight_layout()
```



7 Este gráficos demonstra que a marca Tramontina possui maior interesse, seguida pela LA CUISINE. Este gráfico também nos mostra que ambas as marcas são as que apresentação maior taxa de desinteresse. Os produtos sem marca definida representam uma minoria

Analisando o grau de interesse das diferentes composições. Convetendo NaNs para desconhecido

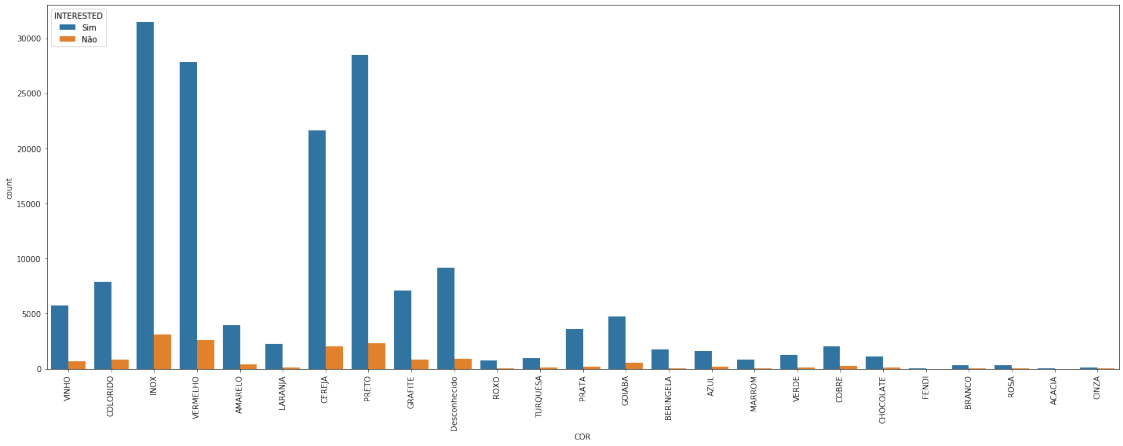
[16]:
```
dataset_tmp['COMPOSICAO'] = dataset_tmp['COMPOSICAO'].fillna('Desconhecido')
fig,ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'COMPOSICAO', data = dataset_tmp, hue='INTERESTED')
plt.xticks(rotation='vertical')
plt.tight_layout()
```



8 Observamos que produtos feitos de alumino dominam o interesse do cliente seguidos pelos de inox. Os produtos com composição desconhecida também tem atraído o interesse do cliente

Analisando o grau de interesse das em relação a cor. Convertendo NaNs para desconhecido

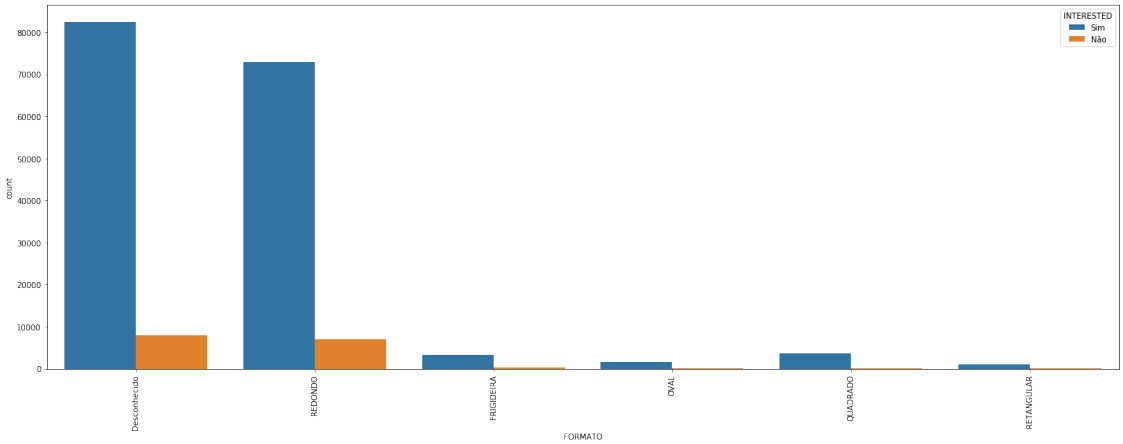
[17]:
```
dataset_tmp['COR'] = dataset_tmp['COR'].fillna('Desconhecido')
fig,ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'COR', data = dataset_tmp, hue='INTERESTED')
plt.xticks(rotation='vertical')
plt.tight_layout()
```



9 Os produtos com cores inox, vermelho, cereja e preto são os que mais cham o interesse do cliente

Analisando o grau de interesse das em relação ao formato. Convertendo NaNs para desconhecido

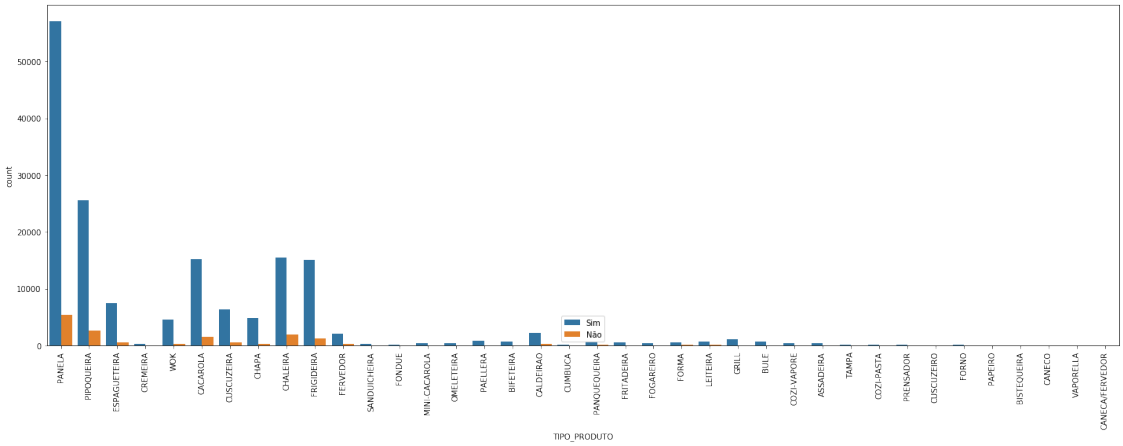
[18]:
```
dataset_tmp['FORMATO'] = dataset_tmp['FORMATO'].fillna('Desconhecido')
fig,ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'FORMATO', data = dataset_tmp, hue='INTERESTED')
plt.xticks(rotation='vertical')
plt.tight_layout()
```



10 Produtos com formato redondo e desconhecido receberam mais interesse do cliente e notamos que ambos apresentaram o mesmo grau de rejeição.

Analisando a distribuição pelo tipo de produto

[19]:
```
dataset_tmp['COR'] = dataset_tmp['TIPO_PRODUTO'].fillna('Desconhecido')
fig,ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'TIPO_PRODUTO', data = dataset_tmp, hue='INTERESTED')
plt.xticks(rotation='vertical')
plt.legend(loc='best')
plt.tight_layout()
```



11 As variáveis que mais definiram se um cliente irá ou não se interessar pelo produto são MARCA, FORMATO, COR, TIPO_PRODUTO e COMPOSISAO

Iremos agora converter as variáveis categóricas Marca, composição, cor, formato, para lavar louças, para microondas, tem_ferro_fundido, tem grelha, tipo de produto e tem wook para variáveis numéricas

Importanto bilioteca LabelEnconder para realizar esta conversão

[20]:
```
from sklearn.preprocessing import LabelEncoder
```

[21]:
```
dataset.head()
```

[21]:

	ALTURA	CAPACIDADE_(L)	COMPOSICAO	COR	FORMATO	LARGURA	MARCA	\
0	30.5	NaN	ALUMINIO	VINHO	NaN	14.0	LA CUISINE	
1	22.0	NaN	ALUMINIO	COLORIDO	NaN	24.0	TRAMONTINA	
2	24.0	NaN	INOX	INOX	NaN	20.0	LA CUISINE	
3	49.5	6.0	ALUMINIO	VERMELHO	REDONDO	41.5	TRAMONTINA	

```
4    49.5             6.0   ALUMINIO  VERMELHO  REDONDO     41.5  TRAMONTINA

  PARA_LAVA_LOUCAS PARA_MICRO_ONDAS   PESO  PROFUNDIDADE  TEMPO_GARANTIA  \
0              NaN              NaN    NaN          50.0             3.0
1               No               no  150.0          40.0            12.0
2              Yes               no  190.0          20.0             3.0
3              Yes              NaN  120.0          47.0             NaN
4              Yes              NaN  120.0          47.0             NaN

  TEM_FERRO_FUNDIDO TEM_GRELHA  TEM_TAMPA  TIPO_PRODUTO TIPO_WOK  ITEM_PRICE  \
0               NAO        SIM        1.0        PANELA      NAO  199.990000
1               NAO        NAO        1.0    PIPOQUEIRA      NAO  105.112581
2               NAO        NAO        1.0  ESPAGUETEIRA      NAO  139.990000
3               NAO        NAO        1.0    PIPOQUEIRA      NAO  103.293333
4               NAO        NAO        1.0    PIPOQUEIRA      NAO  103.330242

   INTERESTED
0         0.0
1         0.0
2         0.0
3         1.0
4         0.0
```

Convertendo variável categórica marca para numérica. Convertendo NaNs para a label sem marca

[22]:
```
dataset['MARCA'] = dataset['MARCA'].fillna('Sem Marca')
enconder_1 = LabelEncoder();
enconder_1.fit(dataset['MARCA'].values);
```

Verificando se as classes estão corretas

[23]:
```
enconder_1.classes_
```

[23]:
```
array(['ALUMINIO FORTALEZA', 'ALUMINIO RAMOS', 'AT.HOME', 'BIALETTI',
       'BR HOME', 'BRINOX', 'CERAFLAME', 'CLASSIC HOME', 'CONSUL',
       'DONA CHEFA', 'DYNASTY', 'ELECTROLUX', 'EURO', 'EURO HOME',
       'FORMA', 'FUMIL', 'HAPPYCALL', 'HELIOS', 'HERCULES', 'JOMAFE',
       'LA CUISINE', 'MERIDIONAL', 'MIMO STYLE', 'MTA', 'OXFORD COOKWARE',
       'PANELA MINEIRA', 'PANELUX', 'PRAVA', 'PRAVAS', 'ROCHEDO',
       'SUN GUIDER', 'Sem Marca', 'TRAMONTINA',
       'TRAMONTINA DESIGN COLLECTION', 'VEC', 'ZANELLA'], dtype=object)
```

[24]:
```
marca_convertida = enconder_1.transform(dataset['MARCA'].values)
```

[25]:
```
dataset['MARCA'] = marca_convertida
```

Convertendo variável categórica COMPOSICAO para numérica. Convertendo NaNs para a label desconhecido

[26]:
```
dataset['COMPOSICAO'] = dataset['COMPOSICAO'].fillna('Desconhecido')
enconder_2 = LabelEncoder();
enconder_2.fit(dataset['COMPOSICAO'].values);
```

Verificando se as classe estão corretas

[27]:
```
enconder_2.classes_
```

[27]:
```
array(['ACO ESMALTADO', 'ALUMINIO', 'CERAMICA', 'Desconhecido', 'INOX',
       'PORCELANA'], dtype=object)
```

[28]:
```
composicao_convertida = enconder_2.transform(dataset['COMPOSICAO'].values)
```

[29]:
```
dataset['COMPOSICAO'] = composicao_convertida
```

Convertendo variável categórica COR para numérica. Convertendo NaNs para a label desconhecido

[30]:
```
dataset['COR'] = dataset['COR'].fillna('Desconhecido')
enconder_3 = LabelEncoder();
enconder_3.fit(dataset['COR'].values);
```

Verificando se as classe estão corretas

[31]:
```
enconder_3.classes_
```

[31]:
```
array(['ACACIA', 'AMARELO', 'AZUL', 'BERINGELA', 'BRANCO', 'CEREJA',
       'CHOCOLATE', 'CINZA', 'COBRE', 'COLORIDO', 'Desconhecido', 'FENDI',
       'GOIABA', 'GRAFITE', 'INOX', 'LARANJA', 'MARROM', 'PRATA', 'PRETO',
       'ROSA', 'ROXO', 'TURQUESA', 'VERDE', 'VERMELHO', 'VINHO'],
      dtype=object)
```

[32]:
```
cor_convertida = enconder_3.transform(dataset['COR'].values)
```

[33]:
```
dataset['COR'] = cor_convertida
```

Convertendo variável categórica FORMATO para numérica. Convertendo NaNs para a label Sem Forma

[34]:
```
dataset['FORMATO'] = dataset['FORMATO'].fillna('Sem Forma')
enconder_4 = LabelEncoder();
enconder_4.fit(dataset['FORMATO'].values);
```

Verificando se as classe estão corretas

[35]:
```
enconder_4.classes_
```

[35]:
```
array(['FRIGIDEIRA', 'OVAL', 'QUADRADO', 'REDONDO', 'RETANGULAR',
       'Sem Forma'], dtype=object)
```

[36]:
```
forma_convertida = enconder_4.transform(dataset['FORMATO'].values)
```

[37]:
```
dataset['FORMATO'] = forma_convertida
```

Convertendo variável categórica PARA_LAVA_LOUCAS para numérica. Convertendo NaNs para a label Talvez. Corrigindo os valores NAO para No

[38]:
```
dataset['PARA_LAVA_LOUCAS'] = dataset['PARA_LAVA_LOUCAS'].fillna('Talvez')
dataset[dataset['PARA_LAVA_LOUCAS'] == 'NAO']['PARA_LAVA_LOUCAS'] = 'No'
enconder_5 = LabelEncoder();
enconder_5.fit(dataset['PARA_LAVA_LOUCAS'].values);
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Verificando as classes

[39]:
```
enconder_5.classes_
```

[39]:
```
array(['NAO', 'No', 'Talvez', 'Yes'], dtype=object)
```

[40]:
```
lavar_louca_convertida = enconder_5.transform(dataset['PARA_LAVA_LOUCAS'].
↪values)
```

[41]:
```
dataset['PARA_LAVA_LOUCAS'] = lavar_louca_convertida
```

Convertendo variável categórica PARA_MICRO_ONDAS para numérica. Convertendo NaNs para a label Talvez.

[42]:
```
dataset['PARA_MICRO_ONDAS'] = dataset['PARA_MICRO_ONDAS'].fillna('Talvez')
enconder_6 = LabelEncoder();
enconder_6.fit(dataset['PARA_MICRO_ONDAS'].values);
```

Verificando as classes

[43]:
```
enconder_6.classes_
```

[43]:
```
array(['Talvez', 'no', 'yes'], dtype=object)
```

[44]:
```
para_micro_ondas_convertida = enconder_6.transform(dataset['PARA_MICRO_ONDAS'].
↪values)
```

[45]:
```
dataset['PARA_MICRO_ONDAS'] = para_micro_ondas_convertida
```

Convertendo variável categórica TEM_FERRO_FUNDIDO para numérica. Convertendo NaNs para a label Talvez.

[46]:
```
dataset['TEM_FERRO_FUNDIDO'] = dataset['TEM_FERRO_FUNDIDO'].fillna('Talvez')
enconder_7 = LabelEncoder();
enconder_7.fit(dataset['TEM_FERRO_FUNDIDO'].values);
```

Verificando as classes

[47]:
```
enconder_7.classes_
```

[47]:
```
array(['NAO', 'SIM'], dtype=object)
```

[48]:
```
ferro_fundido_convertida = enconder_7.transform(dataset['TEM_FERRO_FUNDIDO'].
↪values)
```

[49]:
```
dataset['TEM_FERRO_FUNDIDO'] = ferro_fundido_convertida
```

Convertendo variável categórica TEM_GRELHA para numérica. Convertendo NaNs para a label Talvez.

[50]:
```
dataset['TEM_GRELHA'] = dataset['TEM_GRELHA'].fillna('Talvez')
enconder_8 = LabelEncoder();
enconder_8.fit(dataset['TEM_GRELHA'].values);
```

Verificando as classes

[51]:
```
enconder_8.classes_
```

[51]:
```
array(['NAO', 'SIM'], dtype=object)
```

[52]:
```
grelha_convertida = enconder_8.transform(dataset['TEM_GRELHA'].values)
```

[53]:
```
dataset['TEM_GRELHA'] = grelha_convertida
```

Convertendo variável categórica TIPO_PRODUTO para numérica. Convertendo NaNs para a label Desconhecido.

[54]:
```
dataset['TIPO_PRODUTO'] = dataset['TIPO_PRODUTO'].fillna('Desconhecido')
enconder_9 = LabelEncoder();
enconder_9.fit(dataset['TIPO_PRODUTO'].values);
```

Verificando as classes

[55]:
```
enconder_9.classes_
```

[55]:
```
array(['NAO', 'SIM'], dtype=object)
```

[56]:
```
tipo_produto_convertida = enconder_9.transform(dataset['TIPO_PRODUTO'].values)
```

[57]:
```
dataset['TIPO_PRODUTO'] = tipo_produto_convertida
```

Convertendo variável categórica TIPO_WOK para numérica. Convertendo NaNs para a label Desconhecido.

[58]:
```
dataset['TIPO_WOK'] = dataset['TIPO_WOK'].fillna('Desconhecido')
enconder_10 = LabelEncoder();
enconder_10.fit(dataset['TIPO_WOK'].values);
```

Verificando as classes

[59]:
```
enconder_10.classes_
```

[59]:
```
array(['NAO', 'SIM'], dtype=object)
```

[60]:
```
tipo_wok_convertida = enconder_10.transform(dataset['TIPO_WOK'].values)
```

[61]:
```
dataset['TIPO_WOK'] = tipo_wok_convertida
```

Verificando a quantidade de NaNs presentes nas variáveis ALTURA, CAPACIDADE_(L), LARGURA, PESO, PROFUNDIDADE, TEMPO_GARANTIA e ITEM_PRICE

[62]:
```
n_nans_altura = dataset['ALTURA'].isna().sum()
n_nans_altura_normalizado = n_nans_altura * 100.0 / linhas
print("Esta coluna possui {} NaNs, o que corresponde a {:0,.2f} % do total".
 ↪format(n_nans_altura,n_nans_altura_normalizado))
```

```
Esta coluna possui 9268 NaNs, o que corresponde a 5.14 % do total
```

[63]:
```
n_nans_capacidade = dataset['CAPACIDADE_(L)'].isna().sum()
n_nans_capacidade_normalizado = n_nans_capacidade * 100.0 / linhas
print("Esta coluna possui {} NaNs, o que corresponde a {:0,.2f} % do total".
 ↪format(n_nans_capacidade,n_nans_capacidade_normalizado))
```

```
Esta coluna possui 103604 NaNs, o que corresponde a 57.47 % do total
```

[64]:
```
n_nans_largura = dataset['LARGURA'].isna().sum()
n_nans_largura_normalizado = n_nans_largura * 100.0 / linhas
print("Esta coluna possui {} NaNs, o que corresponde a {:0,.2f} % do total".
 ↪format(n_nans_largura,n_nans_largura_normalizado))
```

```
Esta coluna possui 9268 NaNs, o que corresponde a 5.14 % do total
```

[65]:
```
n_nans_peso = dataset['PESO'].isna().sum()
n_nans_peso_normalizado = n_nans_peso * 100.0 / linhas
print("Esta coluna possui {} NaNs, o que corresponde a {:0,.2f} % do total".
 ↪format(n_nans_peso,n_nans_peso_normalizado))
```

```
Esta coluna possui 81751 NaNs, o que corresponde a 45.35 % do total
```

[66]:
```
n_nans_profundidade = dataset['PROFUNDIDADE'].isna().sum()
n_nans_profundidade_normalizado = n_nans_profundidade * 100.0 / linhas
print("Esta coluna possui {} NaNs, o que corresponde a {:0,.2f} % do total".
 ↪format(n_nans_profundidade,n_nans_profundidade_normalizado))
```

```
Esta coluna possui 9268 NaNs, o que corresponde a 5.14 % do total
```

[67]:
```
n_nans_garantia = dataset['TEMPO_GARANTIA'].isna().sum()
n_nans_garantia_normalizado = n_nans_garantia * 100.0 / linhas
print("Esta coluna possui {} NaNs, o que corresponde a {:0,.2f} % do total".
 ↪format(n_nans_garantia,n_nans_garantia_normalizado))
```

```
Esta coluna possui 57505 NaNs, o que corresponde a 31.90 % do total
```

[68]:
```
n_nans_preco = dataset['ITEM_PRICE'].isna().sum()
n_nans_preco_normalizado = n_nans_preco * 100.0 / linhas
print("Esta coluna possui {} NaNs, o que corresponde a {:0,.2f} % do total".
 ↪format(n_nans_preco,n_nans_preco_normalizado))
```

```
Esta coluna possui 13097 NaNs, o que corresponde a 7.27 % do total
```

Substituindo NaNs utilizando o método backward propagation

[82]:
```
dataset['ITEM_PRICE'] = dataset['ITEM_PRICE'].fillna(method='bfill')
dataset['TEMPO_GARANTIA'] = dataset['TEMPO_GARANTIA'].fillna(method='bfill')
dataset['PROFUNDIDADE'] = dataset['PROFUNDIDADE'].fillna(method='bfill')
dataset['PESO'] = dataset['PESO'].fillna(method='bfill')
dataset['LARGURA'] = dataset['LARGURA'].fillna(method='bfill')
dataset['CAPACIDADE_(L)'] = dataset['CAPACIDADE_(L)'].fillna(method='bfill')
dataset['ALTURA'] = dataset['ALTURA'].fillna(method='bfill')
```

Verificando se todos os NaNs foram remodidos

[83]:
```
dataset.isna().sum()
```

[83]:
```
ALTURA                0
CAPACIDADE_(L)        0
COMPOSICAO            0
COR                   0
FORMATO               0
LARGURA               0
MARCA                 0
PARA_LAVA_LOUCAS      0
PARA_MICRO_ONDAS      0
PESO                  0
PROFUNDIDADE          0
TEMPO_GARANTIA        0
TEM_FERRO_FUNDIDO     0
TEM_GRELHA            0
TEM_TAMPA             0
TIPO_PRODUTO          0
TIPO_WOK              0
ITEM_PRICE            0
INTERESTED            0
dtype: int64
```

Uma vez que todas as variáveis foram exploras e corrigidas, podemos passar a explorar técnicas de machine learning. Como o problema consiste em determinar se o produto será interessante ou não para o cliente, o método de regressão logística parece ser um bom método para se implementar inicialmente

Primeiramente devemos determinar as variaveis X e Y. Para o nosso problema a variável Y é INTERESTED e a variável X são as demais colunas

[84]:
```
X = dataset.drop('INTERESTED',axis=1).values
Y = dataset['INTERESTED'].values
```

Vamos gerar uma amostra de treino e teste utilizando a função train_test_split. A amostra de treino terá 70% dos dados e a amostra de teste terá 30% dos dados

[85]:
```
from sklearn.model_selection import train_test_split
```

[86]:
```
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size=0.30,␣
↪random_state=42)
```

Importanto biblioteca da Regressão Logística e realizando o procedimento da regressão utilizando os valores padrões da função. Iremos mais adiante obter os melhores parâmetros desta função

[87]:
```
from sklearn.linear_model import LogisticRegression
```

[88]:
```
logreg = LogisticRegression(penalty='l2',solver='newton-cg',max_iter=10000)
```

[89]:
```
logreg.fit(X_treino,Y_treino)
```

[89]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10000,
                   multi_class='warn', n_jobs=None, penalty='l2',
```

```
                   random_state=None, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
```

[90]:
```
previsao_logref = logreg.predict(X_teste)
```

Utilizando a matriz de confusão para comparar os resultados

[91]:
```
from sklearn.metrics import confusion_matrix
```

[92]:
```
matriz_confusao = confusion_matrix(Y_teste, previsao_logref)
print(matriz_confusao)
```

```
[[49381     0]
 [ 4702     0]]
```

12 Observamos que quando utilizamos todas as colunas, nenhum verdadeiro negativo é encontrado. Iremos agora testar incluindo apenas as colunas que se mostraram interessantes para o estudo

[93]:
```
X_cols = ['MARCA','FORMATO','COR','TIPO_PRODUTO','COMPOSICAO']
```

[94]:
```
X_new = dataset[X_cols].values
Y_new = dataset['INTERESTED'].values
```

[95]:
```
X_treino2, X_teste2, Y_treino2, Y_teste2 = train_test_split(X_new, Y_new,␣
 ↪test_size=0.30, random_state=42)
```

[96]:
```
logreg2 = LogisticRegression(penalty='l2',solver='newton-cg',max_iter=10000)
```

[97]:
```
logreg2.fit(X_treino2,Y_treino2)
```

[97]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10000,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
```

[98]:
```
previsao_logref_2 = logreg2.predict(X_teste2)
```

[99]:
```
matriz_confusao = confusion_matrix(Y_teste2, previsao_logref_2)
print(matriz_confusao)
```

```
[[49381     0]
 [ 4702     0]]
```

13 Observamos que mesmo neste caso a regressão logistica falhou em prever os verdadeiros negativos

Vamos agora utilizar o Decision Tree Classifier para tentar obter uma melhor performance no modelo

[100]:
```
from sklearn.tree import DecisionTreeClassifier
```

[101]:
```
dtc =␣
↪DecisionTreeClassifier(criterion='entropy',max_depth=20000,max_features='log2')
```

[102]:
```
dtc.fit(X_treino2,Y_treino2)
```

[102]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=20000,
                       max_features='log2', 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')
```

[103]:
```
previsao_dtc = dtc.predict(X_teste2)
```

[104]:
```
matriz_confusao_dtc = confusion_matrix(Y_teste2, previsao_dtc)
print(matriz_confusao_dtc)
```

```
[[49381     0]
 [ 4702     0]]
```

14 Observamos que o mesmo problema ocorre com o Decision Tree Classifier

Testaremos agora o Random Forest Classifier antes de implementarmos a rede neural

[105]:
```
from sklearn.ensemble import RandomForestClassifier
```

[106]:
```
rfc = RandomForestClassifier(n_estimators=1000,␣
↪max_depth=500,criterion='entropy')
```

[107]:
```
rfc.fit(X_teste2,Y_teste2)
```

[107]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                       max_depth=500, max_features='auto', 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, n_estimators=1000,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[108]:
```
previsao_rfc = rfc.predict(X_teste2)
```

[109]:
```
matriz_confusao_dtc = confusion_matrix(Y_teste2, previsao_rfc)
print(matriz_confusao_dtc)
```

```
[[49381     0]
 [ 4699     3]]
```

15 O random forest classifier apresentou um resultado melhor comparado aos demais, porém, ainda assim falha em prever os verdadeiros negativos

Vamos agora implementar uma rede neural utilizando o Keras

[176]:
```
from keras.models import Sequential
from keras.layers import Dense,LeakyReLU,ReLU
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from keras.layers.advanced_activations import PReLU
from keras.optimizers import SGD
from keras.callbacks import ReduceLROnPlateau
```

[219]:
```
def modelo_keras():
    model = Sequential()
    model.add(Dense(64, input_dim=18, activation='tanh'))
    model.add(Dense(128,activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy','mse','mae','mape'])
    return model
```

[222]:
```
modelo = modelo_keras()
modelo.fit(X_treino, Y_treino, epochs=10, batch_size=100,␣
 ↪validation_data=(X_teste, Y_teste))
```

```
Train on 126192 samples, validate on 54083 samples
Epoch 1/10
126192/126192 [==============================] - 4s 35us/step - loss: 0.2959 -
acc: 0.9128 - mean_squared_error: 0.0793 - mean_absolute_error: 0.1572 -
mean_absolute_percentage_error: 79013939.2644 - val_loss: 0.2972 - val_acc:
0.9131 - val_mean_squared_error: 0.0796 - val_mean_absolute_error: 0.1435 -
val_mean_absolute_percentage_error: 62705775.0173
Epoch 2/10
126192/126192 [==============================] - 3s 21us/step - loss: 0.2930 -
acc: 0.9142 - mean_squared_error: 0.0785 - mean_absolute_error: 0.1564 -
mean_absolute_percentage_error: 78187448.4619 - val_loss: 0.2954 - val_acc:
0.9131 - val_mean_squared_error: 0.0794 - val_mean_absolute_error: 0.1683 -
val_mean_absolute_percentage_error: 90258480.9156
Epoch 3/10
126192/126192 [==============================] - 3s 22us/step - loss: 0.2923 -
acc: 0.9142 - mean_squared_error: 0.0784 - mean_absolute_error: 0.1564 -
mean_absolute_percentage_error: 78200697.8671 - val_loss: 0.2951 - val_acc:
0.9131 - val_mean_squared_error: 0.0794 - val_mean_absolute_error: 0.1679 -
val_mean_absolute_percentage_error: 89756913.2144
Epoch 4/10
126192/126192 [==============================] - 3s 23us/step - loss: 0.2920 -
```

```
acc: 0.9142 - mean_squared_error: 0.0783 - mean_absolute_error: 0.1563 -
mean_absolute_percentage_error: 78161708.2593 - val_loss: 0.2972 - val_acc:
0.9131 - val_mean_squared_error: 0.0798 - val_mean_absolute_error: 0.1783 -
val_mean_absolute_percentage_error: 101296476.8478
Epoch 5/10
126192/126192 [==============================] - 3s 22us/step - loss: 0.2919 -
acc: 0.9142 - mean_squared_error: 0.0783 - mean_absolute_error: 0.1564 -
mean_absolute_percentage_error: 78171914.4422 - val_loss: 0.2948 - val_acc:
0.9131 - val_mean_squared_error: 0.0793 - val_mean_absolute_error: 0.1666 -
val_mean_absolute_percentage_error: 88292076.5955
Epoch 6/10
126192/126192 [==============================] - 3s 22us/step - loss: 0.2922 -
acc: 0.9142 - mean_squared_error: 0.0784 - mean_absolute_error: 0.1564 -
mean_absolute_percentage_error: 78209877.4005 - val_loss: 0.2949 - val_acc:
0.9131 - val_mean_squared_error: 0.0793 - val_mean_absolute_error: 0.1690 -
val_mean_absolute_percentage_error: 90963836.6061
Epoch 7/10
126192/126192 [==============================] - 3s 23us/step - loss: 0.2920 -
acc: 0.9142 - mean_squared_error: 0.0783 - mean_absolute_error: 0.1564 -
mean_absolute_percentage_error: 78199161.8922 - val_loss: 0.2945 - val_acc:
0.9131 - val_mean_squared_error: 0.0793 - val_mean_absolute_error: 0.1626 -
val_mean_absolute_percentage_error: 83868730.5683
Epoch 8/10
126192/126192 [==============================] - 3s 22us/step - loss: 0.2918 -
acc: 0.9142 - mean_squared_error: 0.0783 - mean_absolute_error: 0.1565 -
mean_absolute_percentage_error: 78303852.0011 - val_loss: 0.2963 - val_acc:
0.9131 - val_mean_squared_error: 0.0795 - val_mean_absolute_error: 0.1431 -
val_mean_absolute_percentage_error: 62295351.4781
Epoch 9/10
126192/126192 [==============================] - 3s 22us/step - loss: 0.2920 -
acc: 0.9142 - mean_squared_error: 0.0783 - mean_absolute_error: 0.1564 -
mean_absolute_percentage_error: 78234695.5656 - val_loss: 0.2950 - val_acc:
0.9131 - val_mean_squared_error: 0.0793 - val_mean_absolute_error: 0.1493 -
val_mean_absolute_percentage_error: 69170196.3021
Epoch 10/10
126192/126192 [==============================] - 3s 23us/step - loss: 0.2920 -
acc: 0.9142 - mean_squared_error: 0.0783 - mean_absolute_error: 0.1564 -
mean_absolute_percentage_error: 78143418.0870 - val_loss: 0.2938 - val_acc:
0.9131 - val_mean_squared_error: 0.0792 - val_mean_absolute_error: 0.1600 -
val_mean_absolute_percentage_error: 80992779.6801
```

[222]:
```
<keras.callbacks.History at 0x1a4e5227d0>
```

[223]:
```
loss_keras, precisao_keras, mse_keras, mae_keras,mape_keras = modelo.
↪evaluate(X_treino, Y_treino)
```

```
126192/126192 [==============================] - 4s 29us/step
```

[224]:
```
print('Precisao: %.2f' % (precisao_keras*100))
print('Loss: %.2f' % (loss_keras*100))
print('MSE: %.2f' % (mse_keras*100))
print('MAE: %.2f' % (mae_keras*100))
print('MAPE: %.2f' % (mape_keras*100))
```

```
Precisao: 91.42
Loss: 29.09
MSE: 7.81
MAE: 15.90
MAPE: 8114898107.47
```

[225]:
```
previsao_keras = modelo.predict_classes(X_teste)
```

[226]:
```
matriz_confusao_keras = confusion_matrix(Y_teste, previsao_keras)
print(matriz_confusao_keras)
```

```
[[49381     0]
 [ 4702     0]]
```

[230]:
```
X_cols =␣
 ↪['TIPO_PRODUTO','ITEM_PRICE','COMPOSICAO']#'ALTURA','PESO','ITEM_PRICE','FORMATO','TIPO_PRO
X_new2 = dataset[X_cols].values
Y_new2 = dataset['INTERESTED'].values
X_treino3, X_teste3, Y_treino3, Y_teste3 = train_test_split(X_new2, Y_new2,␣
 ↪test_size=0.30, random_state=42)
```

[231]:
```
def modelo2_keras():
    model = Sequential()
    model.add(Dense(32, input_dim=3))
    model.add(ReLU())
    model.add(Dense(64))
    model.add(ReLU())
    model.add(Dense(1, activation='sigmoid'))
    sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss='binary_crossentropy', optimizer=sgd,␣
 ↪metrics=['accuracy','mse','mae','mape'])
    return model
```

[234]:
```
modelo2 = modelo2_keras()
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc', patience=3,␣
 ↪verbose=1, factor=0.2, mode='auto')
modelo2.fit(X_treino3, Y_treino3, epochs=10, batch_size=100,␣
 ↪callbacks=[learning_rate_reduction], validation_data=(X_teste3, Y_teste3))
```

```
Train on 126192 samples, validate on 54083 samples
Epoch 1/10
```

```
126192/126192 [==============================] - 5s 39us/step - loss: 1.3848 -
acc: 0.9135 - mean_squared_error: 0.0864 - mean_absolute_error: 0.0864 -
mean_absolute_percentage_error: 707981.3380 - val_loss: 1.4013 - val_acc: 0.9131
- val_mean_squared_error: 0.0869 - val_mean_absolute_error: 0.0869 -
val_mean_absolute_percentage_error: 8.6940
Epoch 2/10
126192/126192 [==============================] - 3s 28us/step - loss: 1.3826 -
acc: 0.9142 - mean_squared_error: 0.0858 - mean_absolute_error: 0.0858 -
mean_absolute_percentage_error: 8.5782 - val_loss: 1.4013 - val_acc: 0.9131 -
val_mean_squared_error: 0.0869 - val_mean_absolute_error: 0.0869 -
val_mean_absolute_percentage_error: 8.6940
Epoch 3/10
126192/126192 [==============================] - 3s 27us/step - loss: 1.3826 -
acc: 0.9142 - mean_squared_error: 0.0858 - mean_absolute_error: 0.0858 -
mean_absolute_percentage_error: 8.5782 - val_loss: 1.4013 - val_acc: 0.9131 -
val_mean_squared_error: 0.0869 - val_mean_absolute_error: 0.0869 -
val_mean_absolute_percentage_error: 8.6940
Epoch 4/10
126192/126192 [==============================] - 5s 40us/step - loss: 1.3826 -
acc: 0.9142 - mean_squared_error: 0.0858 - mean_absolute_error: 0.0858 -
mean_absolute_percentage_error: 8.5782 - val_loss: 1.4013 - val_acc: 0.9131 -
val_mean_squared_error: 0.0869 - val_mean_absolute_error: 0.0869 -
val_mean_absolute_percentage_error: 8.6940

Epoch 00004: ReduceLROnPlateau reducing learning rate to 0.020000000298023225.
Epoch 5/10
126192/126192 [==============================] - 5s 37us/step - loss: 1.3826 -
acc: 0.9142 - mean_squared_error: 0.0858 - mean_absolute_error: 0.0858 -
mean_absolute_percentage_error: 8.5782 - val_loss: 1.4013 - val_acc: 0.9131 -
val_mean_squared_error: 0.0869 - val_mean_absolute_error: 0.0869 -
val_mean_absolute_percentage_error: 8.6940
Epoch 6/10
126192/126192 [==============================] - 5s 39us/step - loss: 1.3826 -
acc: 0.9142 - mean_squared_error: 0.0858 - mean_absolute_error: 0.0858 -
mean_absolute_percentage_error: 8.5782 - val_loss: 1.4013 - val_acc: 0.9131 -
val_mean_squared_error: 0.0869 - val_mean_absolute_error: 0.0869 -
val_mean_absolute_percentage_error: 8.6940
Epoch 7/10
126192/126192 [==============================] - 4s 32us/step - loss: 1.3826 -
acc: 0.9142 - mean_squared_error: 0.0858 - mean_absolute_error: 0.0858 -
mean_absolute_percentage_error: 8.5782 - val_loss: 1.4013 - val_acc: 0.9131 -
val_mean_squared_error: 0.0869 - val_mean_absolute_error: 0.0869 -
val_mean_absolute_percentage_error: 8.6940

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.003999999910593033.
Epoch 8/10
126192/126192 [==============================] - 5s 38us/step - loss: 1.3826 -
acc: 0.9142 - mean_squared_error: 0.0858 - mean_absolute_error: 0.0858 -
```

```
mean_absolute_percentage_error: 8.5782 - val_loss: 1.4013 - val_acc: 0.9131 -
val_mean_squared_error: 0.0869 - val_mean_absolute_error: 0.0869 -
val_mean_absolute_percentage_error: 8.6940
Epoch 9/10
126192/126192 [==============================] - 4s 36us/step - loss: 1.3826 -
acc: 0.9142 - mean_squared_error: 0.0858 - mean_absolute_error: 0.0858 -
mean_absolute_percentage_error: 8.5782 - val_loss: 1.4013 - val_acc: 0.9131 -
val_mean_squared_error: 0.0869 - val_mean_absolute_error: 0.0869 -
val_mean_absolute_percentage_error: 8.6940
Epoch 10/10
126192/126192 [==============================] - 5s 38us/step - loss: 1.3826 -
acc: 0.9142 - mean_squared_error: 0.0858 - mean_absolute_error: 0.0858 -
mean_absolute_percentage_error: 8.5782 - val_loss: 1.4013 - val_acc: 0.9131 -
val_mean_squared_error: 0.0869 - val_mean_absolute_error: 0.0869 -
val_mean_absolute_percentage_error: 8.6940

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.0007999999448657036.
```

[234]:
```
<keras.callbacks.History at 0x1a5755ef10>
```

[235]:
```
loss_keras2, precisao_keras2, mse_keras2, mae_keras2,mape_keras2 = modelo2.
↪evaluate(X_treino3, Y_treino3)
```

```
126192/126192 [==============================] - 4s 34us/step
```

[236]:
```
print('Precisao: %.2f' % (precisao_keras2*100))
print('Loss: %.2f' % (loss_keras2*100))
print('MSE: %.2f' % (mse_keras2*100))
print('MAE: %.2f' % (mae_keras2*100))
print('MAPE: %.2f' % (mape_keras2*100))
```

```
Precisao: 91.42
Loss: 138.26
MSE: 8.58
MAE: 8.58
MAPE: 857.82
```

[238]:
```
previsao_keras2 = modelo2.predict_classes(X_teste3)
```

[239]:
```
matriz_confusao_keras2 = confusion_matrix(Y_teste3, previsao_keras2)
print(matriz_confusao_keras2)
```

```
[[49381     0]
 [ 4702     0]]
```

16 Diferentes implementações de redes neurais utilizando o keras levaram a um mesmo resultado comparando com o Random Forest Classifier por exemplo. Iremos fazer uma análise identica porém removendo todas as linhas que contenham qualquer valor do tipo NaN

[241]:
```
dataset2 = pd.read_csv('problem1_dataset.csv')
```

[243]:
```
dataset2 = dataset2.dropna(axis=0)
```

[244]:
```
dataset2 = dataset2.drop(['ITEM_ID','SESSION_ID'],axis=1)
```

[247]:
```
enconder_1a = LabelEncoder();
enconder_1a.fit(dataset2['MARCA'].values);
marca_convertida = enconder_1a.transform(dataset2['MARCA'].values)
dataset2['MARCA'] = marca_convertida
```

[248]:
```
enconder_2a = LabelEncoder();
enconder_2a.fit(dataset2['COMPOSICAO'].values);
composicao_convertida = enconder_2a.transform(dataset2['COMPOSICAO'].values)
dataset2['COMPOSICAO'] = composicao_convertida
```

[250]:
```
enconder_3a = LabelEncoder();
enconder_3a.fit(dataset2['COR'].values);
cor_convertida = enconder_3.transform(dataset2['COR'].values)
dataset2['COR'] = cor_convertida
```

[251]:
```
enconder_4a = LabelEncoder();
enconder_4a.fit(dataset2['FORMATO'].values);
forma_convertida = enconder_4a.transform(dataset2['FORMATO'].values)
dataset2['FORMATO'] = forma_convertida
```

[253]:
```
dataset2[dataset2['PARA_LAVA_LOUCAS'] == 'NAO']['PARA_LAVA_LOUCAS'] = 'No'
enconder_5a = LabelEncoder();
enconder_5a.fit(dataset2['PARA_LAVA_LOUCAS'].values);
lavar_louca_convertida = enconder_5.transform(dataset2['PARA_LAVA_LOUCAS'].
 ↪values)
dataset2['PARA_LAVA_LOUCAS'] = lavar_louca_convertida
```

[254]:
```
enconder_6a = LabelEncoder();
enconder_6a.fit(dataset2['PARA_MICRO_ONDAS'].values);
para_micro_ondas_convertida = enconder_6a.
 ↪transform(dataset2['PARA_MICRO_ONDAS'].values)
dataset2['PARA_MICRO_ONDAS'] = para_micro_ondas_convertida
```

[255]:
```
enconder_7a = LabelEncoder();
enconder_7a.fit(dataset2['TEM_FERRO_FUNDIDO'].values);
ferro_fundido_convertida = enconder_7a.transform(dataset2['TEM_FERRO_FUNDIDO'].
 ↪values)
dataset2['TEM_FERRO_FUNDIDO'] = ferro_fundido_convertida
```

[256]:
```
enconder_8a = LabelEncoder();
enconder_8a.fit(dataset2['TEM_GRELHA'].values);
grelha_convertida = enconder_8a.transform(dataset2['TEM_GRELHA'].values)
dataset2['TEM_GRELHA'] = grelha_convertida
```

[257]:
```
enconder_9a = LabelEncoder();
enconder_9a.fit(dataset2['TIPO_PRODUTO'].values);
tipo_produto_convertida = enconder_9a.transform(dataset2['TIPO_PRODUTO'].values)
dataset2['TIPO_PRODUTO'] = tipo_produto_convertida
```

[258]:
```
enconder_10a = LabelEncoder();
enconder_10a.fit(dataset2['TIPO_WOK'].values);
tipo_wok_convertida = enconder_10a.transform(dataset2['TIPO_WOK'].values)
dataset2['TIPO_WOK'] = tipo_wok_convertida
```

[263]:
```
X = dataset2.drop('INTERESTED',axis=1).values
Y = dataset2['INTERESTED'].values
```

[264]:
```
X_treino2, X_teste2, Y_treino2, Y_teste2 = train_test_split(X, Y, test_size=0.
↪30, random_state=42)
```

[265]:
```
logreg = LogisticRegression(penalty='l2',solver='newton-cg',max_iter=10000)
```

[266]:
```
logreg.fit(X_treino2,Y_treino2)
```

[266]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10000,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
```

[267]:
```
previsao_logreg = logreg.predict(X_teste2)
```

[268]:
```
matriz_confusao_logreg = confusion_matrix(Y_teste2, previsao_logreg)
print(matriz_confusao_logreg)
```

```
[[7484    0]
 [ 717    0]]
```

[287]:
```
def modelo3_keras():
    model = Sequential()
    model.add(Dense(128, input_dim=18,activation='tanh'))
    model.add(Dense(128,activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))
    sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy','mse','mae','mape'])
    return model
```

[289]:
```
modelo2 = modelo3_keras()
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc', patience=3,␣
↪verbose=1, factor=0.2, mode='auto')
```

```
modelo2.fit(X_treino2, Y_treino2, epochs=50, batch_size=50,␣
 ↪callbacks=[learning_rate_reduction], validation_data=(X_teste2, Y_teste2))
```

```
Train on 19135 samples, validate on 8201 samples
Epoch 1/50
19135/19135 [==============================] - 3s 178us/step - loss: 0.2917 -
acc: 0.9154 - mean_squared_error: 0.0777 - mean_absolute_error: 0.1532 -
mean_absolute_percentage_error: 76490513.5981 - val_loss: 0.2986 - val_acc:
0.9126 - val_mean_squared_error: 0.0801 - val_mean_absolute_error: 0.1857 -
val_mean_absolute_percentage_error: 109509071.9854
Epoch 2/50
19135/19135 [==============================] - 1s 56us/step - loss: 0.2889 -
acc: 0.9158 - mean_squared_error: 0.0770 - mean_absolute_error: 0.1531 -
mean_absolute_percentage_error: 76475534.1751 - val_loss: 0.2962 - val_acc:
0.9126 - val_mean_squared_error: 0.0796 - val_mean_absolute_error: 0.1782 -
val_mean_absolute_percentage_error: 101161835.5371
Epoch 3/50
19135/19135 [==============================] - 1s 77us/step - loss: 0.2861 -
acc: 0.9158 - mean_squared_error: 0.0766 - mean_absolute_error: 0.1532 -
mean_absolute_percentage_error: 76733006.8356 - val_loss: 0.2985 - val_acc:
0.9126 - val_mean_squared_error: 0.0795 - val_mean_absolute_error: 0.1466 -
val_mean_absolute_percentage_error: 66588088.1405
Epoch 4/50
19135/19135 [==============================] - 2s 102us/step - loss: 0.2870 -
acc: 0.9158 - mean_squared_error: 0.0768 - mean_absolute_error: 0.1529 -
mean_absolute_percentage_error: 76375825.6054 - val_loss: 0.2951 - val_acc:
0.9126 - val_mean_squared_error: 0.0796 - val_mean_absolute_error: 0.1580 -
val_mean_absolute_percentage_error: 78434620.8843

Epoch 00004: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 5/50
19135/19135 [==============================] - 1s 76us/step - loss: 0.2839 -
acc: 0.9158 - mean_squared_error: 0.0762 - mean_absolute_error: 0.1527 -
mean_absolute_percentage_error: 76428122.2556 - val_loss: 0.2918 - val_acc:
0.9126 - val_mean_squared_error: 0.0788 - val_mean_absolute_error: 0.1680 -
val_mean_absolute_percentage_error: 90492496.3414
Epoch 6/50
19135/19135 [==============================] - 2s 89us/step - loss: 0.2834 -
acc: 0.9158 - mean_squared_error: 0.0761 - mean_absolute_error: 0.1530 -
mean_absolute_percentage_error: 76894767.2642 - val_loss: 0.2915 - val_acc:
0.9126 - val_mean_squared_error: 0.0788 - val_mean_absolute_error: 0.1466 -
val_mean_absolute_percentage_error: 66515056.8370
Epoch 7/50
19135/19135 [==============================] - 2s 79us/step - loss: 0.2827 -
acc: 0.9158 - mean_squared_error: 0.0760 - mean_absolute_error: 0.1518 -
mean_absolute_percentage_error: 75727907.8484 - val_loss: 0.2912 - val_acc:
0.9126 - val_mean_squared_error: 0.0786 - val_mean_absolute_error: 0.1648 -
```

```
val_mean_absolute_percentage_error: 87233528.8126

Epoch 00007: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
Epoch 8/50
19135/19135 [==============================] - 1s 58us/step - loss: 0.2814 -
acc: 0.9158 - mean_squared_error: 0.0758 - mean_absolute_error: 0.1527 -
mean_absolute_percentage_error: 76901525.0065 - val_loss: 0.2893 - val_acc:
0.9126 - val_mean_squared_error: 0.0784 - val_mean_absolute_error: 0.1540 -
val_mean_absolute_percentage_error: 75038331.3137
Epoch 9/50
19135/19135 [==============================] - 1s 58us/step - loss: 0.2811 -
acc: 0.9158 - mean_squared_error: 0.0757 - mean_absolute_error: 0.1524 -
mean_absolute_percentage_error: 76621893.7089 - val_loss: 0.2902 - val_acc:
0.9126 - val_mean_squared_error: 0.0785 - val_mean_absolute_error: 0.1465 -
val_mean_absolute_percentage_error: 66764119.9025
Epoch 10/50
19135/19135 [==============================] - 1s 58us/step - loss: 0.2806 -
acc: 0.9158 - mean_squared_error: 0.0757 - mean_absolute_error: 0.1507 -
mean_absolute_percentage_error: 74818570.9914 - val_loss: 0.2888 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1607 -
val_mean_absolute_percentage_error: 82757297.7715

Epoch 00010: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
Epoch 11/50
19135/19135 [==============================] - 1s 54us/step - loss: 0.2804 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1543 -
mean_absolute_percentage_error: 78848933.7277 - val_loss: 0.2889 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1521 -
val_mean_absolute_percentage_error: 73143688.8126
Epoch 12/50
19135/19135 [==============================] - 1s 60us/step - loss: 0.2805 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1508 -
mean_absolute_percentage_error: 74907224.9281 - val_loss: 0.2887 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1546 -
val_mean_absolute_percentage_error: 75893346.9177
Epoch 13/50
19135/19135 [==============================] - 2s 113us/step - loss: 0.2804 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1512 -
mean_absolute_percentage_error: 75340413.4100 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0782 - val_mean_absolute_error: 0.1563 -
val_mean_absolute_percentage_error: 77922945.2506

Epoch 00013: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
Epoch 14/50
19135/19135 [==============================] - 2s 97us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1534 -
mean_absolute_percentage_error: 77851119.7094 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0782 - val_mean_absolute_error: 0.1553 -
```

```
val_mean_absolute_percentage_error: 76792011.2913
Epoch 15/50
19135/19135 [==============================] - 2s 86us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1524 -
mean_absolute_percentage_error: 76705009.7978 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0782 - val_mean_absolute_error: 0.1549 -
val_mean_absolute_percentage_error: 76266828.9721
Epoch 16/50
19135/19135 [==============================] - 1s 62us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1522 -
mean_absolute_percentage_error: 76512964.5989 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75702656.9735

Epoch 00016: ReduceLROnPlateau reducing learning rate to 3.200000264769187e-07.
Epoch 17/50
19135/19135 [==============================] - 1s 60us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75774418.2576 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75684017.4428
Epoch 18/50
19135/19135 [==============================] - 1s 60us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1517 -
mean_absolute_percentage_error: 75904398.0057 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75721752.0498
Epoch 19/50
19135/19135 [==============================] - 1s 60us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75860962.6527 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75747792.6487

Epoch 00019: ReduceLROnPlateau reducing learning rate to 6.400000529538374e-08.
Epoch 20/50
19135/19135 [==============================] - 1s 59us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75822019.6708 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75742432.6409
Epoch 21/50
19135/19135 [==============================] - 1s 59us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75823701.3703 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75745368.6585
Epoch 22/50
```

```
19135/19135 [==============================] - 1s 55us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75827373.6462 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75746054.9709

Epoch 00022: ReduceLROnPlateau reducing learning rate to 1.2800001059076749e-08.
Epoch 23/50
19135/19135 [==============================] - 1s 55us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840454.4176 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75746904.7970
Epoch 24/50
19135/19135 [==============================] - 1s 56us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75842460.7849 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748424.8019
Epoch 25/50
19135/19135 [==============================] - 1s 72us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75844244.8644 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75747972.5604

Epoch 00025: ReduceLROnPlateau reducing learning rate to 2.5600002118153498e-09.
Epoch 26/50
19135/19135 [==============================] - 1s 63us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840527.9185 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748013.5340
Epoch 27/50
19135/19135 [==============================] - 2s 85us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840599.1743 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748027.8493
Epoch 28/50
19135/19135 [==============================] - 1s 69us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840723.7565 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748126.1183

Epoch 00028: ReduceLROnPlateau reducing learning rate to 5.1200004236307e-10.
Epoch 29/50
```

```
19135/19135 [==============================] - 1s 62us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840677.0985 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748128.4331
Epoch 30/50
19135/19135 [==============================] - 1s 67us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840679.2036 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748130.5548
Epoch 31/50
19135/19135 [==============================] - 2s 79us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840686.3507 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.0932

Epoch 00031: ReduceLROnPlateau reducing learning rate to 1.0240001069306004e-10.
Epoch 32/50
19135/19135 [==============================] - 2s 97us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.2934 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748131.9956
Epoch 33/50
19135/19135 [==============================] - 2s 103us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.6342 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.2395
Epoch 34/50
19135/19135 [==============================] - 2s 80us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.2036 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907

Epoch 00034: ReduceLROnPlateau reducing learning rate to 2.0480002416167767e-11.
Epoch 35/50
19135/19135 [==============================] - 1s 76us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.5840 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907
Epoch 36/50
19135/19135 [==============================] - 1s 77us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
```

```
mean_absolute_percentage_error: 75840687.4398 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907
Epoch 37/50
19135/19135 [==============================] - 1s 77us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.5798 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907

Epoch 00037: ReduceLROnPlateau reducing learning rate to 4.096000622011431e-12.
Epoch 38/50
19135/19135 [==============================] - 1s 77us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.3164 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907
Epoch 39/50
19135/19135 [==============================] - 1s 77us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.4272 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907
Epoch 40/50
19135/19135 [==============================] - 1s 75us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.0781 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907

Epoch 00040: ReduceLROnPlateau reducing learning rate to 8.192000897078167e-13.
Epoch 41/50
19135/19135 [==============================] - 1s 75us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.4858 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907
Epoch 42/50
19135/19135 [==============================] - 1s 76us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.4335 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907
Epoch 43/50
19135/19135 [==============================] - 1s 78us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.7042 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
```

```
val_mean_absolute_percentage_error: 75748132.1907

Epoch 00043: ReduceLROnPlateau reducing learning rate to 1.6384001360475466e-13.
Epoch 44/50
19135/19135 [==============================] - 1s 77us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.7032 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907
Epoch 45/50
19135/19135 [==============================] - 2s 79us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.6154 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907
Epoch 46/50
19135/19135 [==============================] - 1s 78us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.1847 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907

Epoch 00046: ReduceLROnPlateau reducing learning rate to 3.2768002178849846e-14.
Epoch 47/50
19135/19135 [==============================] - 1s 73us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.7094 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907
Epoch 48/50
19135/19135 [==============================] - 1s 73us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.6175 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907
Epoch 49/50
19135/19135 [==============================] - 1s 74us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840686.8921 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
val_mean_absolute_percentage_error: 75748132.1907

Epoch 00049: ReduceLROnPlateau reducing learning rate to 6.553600300244697e-15.
Epoch 50/50
19135/19135 [==============================] - 1s 75us/step - loss: 0.2803 -
acc: 0.9158 - mean_squared_error: 0.0756 - mean_absolute_error: 0.1516 -
mean_absolute_percentage_error: 75840687.3541 - val_loss: 0.2886 - val_acc:
0.9126 - val_mean_squared_error: 0.0783 - val_mean_absolute_error: 0.1544 -
```

```
val_mean_absolute_percentage_error: 75748132.1907
```

[289]:
```
<keras.callbacks.History at 0x1a29615ad0>
```

17 Observou-se que mesmo removendo os NaNs de todo os dataset sem substituição, a precisão do modelo continuou muito parecida. Analise das variáveis nos mostrou que remover uma ou mais variável não afetou o resultado final

[ ]: