

Trabalho avaliativo do processo de mineração de dados - parte final

Alunos: Iago Batista Antunes Leobas, Francisco Raphael F. de Araujo, Gabriel Teixeira.

Disciplina: Mineração de Dados

Data: 26/09/2022

```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
```


```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/card_transdata.csv")
```

Verificação do Dataset e iniciando ações de limpeza caso haja necessidade...

```
data.head()
```



	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_pric
0	57.877857	0.311140	1.94594
1	10.829943	0.175592	1.29421
2	5.091079	0.805153	0.42771
3	2.247564	5.600044	0.36266
4	44.190936	0.566486	2.22276

```
print(data.shape)
```

(701072, 8)

```
onlycredit = data[(data['used_chip'] == 1.0)]
print(onlycredit.shape)
```

```
(245436, 8)
```

```
onlyonline = onlycredit[(onlycredit['online_order'] == 1.0)]
print(onlyonline.shape)
```

```
(159743, 8)
```

```
onlyonline.isnull().any().any()
```

```
False
```

```
X = onlyonline.drop(['fraud'], axis=1)
Y = onlyonline['fraud']
```

```
print(X, Y)
```

	distance_from_home	distance_from_last_transaction	\
3	2.247564	5.600044	
4	44.190936	0.566486	
10	14.263530	0.158758	
11	13.592368	0.240540	
15	179.665148	0.120920	
...	
701052	5.901801	0.110441	
701057	0.350711	0.562992	
701063	3.000823	0.148435	
701065	2.602851	1.275756	
701070	3.353519	0.089108	

	ratio_to_median_purchase_price	repeat_retailer	used_chip	\
3	0.362663	1.0	1.0	
4	2.222767	1.0	1.0	
10	1.136102	1.0	1.0	
11	1.370330	1.0	1.0	
15	0.535640	1.0	1.0	
...	
701052	3.303179	1.0	1.0	
701057	0.727901	0.0	1.0	
701063	0.467753	1.0	1.0	
701065	7.307051	1.0	1.0	
701070	0.343640	1.0	1.0	

	used_pin_number	online_order
3	0.0	1.0
4	0.0	1.0
10	0.0	1.0
11	0.0	1.0
15	1.0	1.0
...
701052	0.0	1.0
701057	0.0	1.0
701063	1.0	1.0
701065	0.0	1.0
701070	0.0	1.0

```
[159743 rows x 7 columns] 3      0.0
4      0.0
10     0.0
11     0.0
15     0.0
...
701052 0.0
701057 0.0
701063 0.0
701065 1.0
701070 0.0
Name: fraud, Length: 159743, dtype: float64
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_state = 42,
```

▼ Imprimindo a acurácia dos testes

```
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, Y_train)
pred = lr.predict(X_test)
acc = accuracy_score(Y_test, pred)

f'Acurácia:{acc * 100:.2f}'
```

```
'Acurácia:99.29'
```

```
only_real = onlyonline.fraud
only_total = onlyonline.drop(['fraud'], axis=1)
only_total
```

distance_from_home distance_from_last_transaction ratio_to_median_purchase

Realizando previsões e imprimindo os primeiros 30 resultados

```
11          13.592500          0.240540          1.

pred = lr.predict(only_total)

only_val = pd.DataFrame({'real':only_real, 'previsao':pred})
only_val.head(n=30)
```

	real	previsao
3	0.0	0.0
4	0.0	0.0
10	0.0	0.0
11	0.0	0.0
15	0.0	0.0
28	0.0	0.0

▼ Comparando os valores da previsão com os valores reais

```
only_val.previsao.value_counts()
```

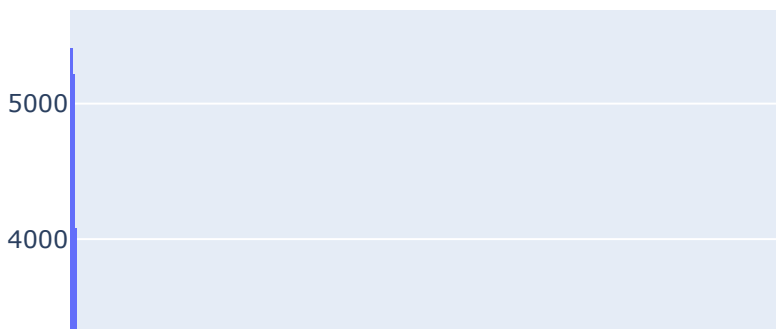
```
0.0    145650
1.0     14093
Name: previsao, dtype: int64
```

```
only_val.real.value_counts()
```

```
0.0    144670
1.0     15073
Name: real, dtype: int64
```

▼ Histograma da Razão da transação do preço de compra para o preço de compra mediano

```
import plotly.express as px
px.histogram(onlyonline, x = 'ratio_to_median_purchase_price')
```



```
data['credit_and_online'] = np.where (
    (data['used_chip'] == 1.0) & (data['used_pin_number'] == 1.0) & (data['online_order']
    'yes',
    'no'
)
```

- Criando a coluna `credit_and_online` para facilitar a
- visualização, dessa forma unindo informações de 3 colunas: `"used_chip"`, `"used_pin_number"` e `"online_order"`

data

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase
0	57.877857	0.311140	1.
1	10.829943	0.175592	1.
2	5.091079	0.805153	0.
3	2.247564	5.600044	0.
4	44.190936	0.566486	2.
...	
701067	14.760684	0.256367	0.
701068	15.453180	0.251033	1.
701069	2.131753	9.002525	1.
701070	3.353519	0.089108	0.
701071	3.682917	0.652274	0.

701072 rows × 9 columns

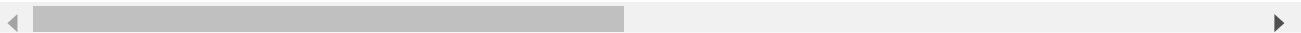
```
data2 = data[data.credit_and_online != 'no']
```

B

data2

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase
15	179.665148	0.120920	0.
51	43.281314	3.367793	0.
55	24.268906	0.136521	1.
98	6.136181	2.579574	1.
138	5.169928	0.534060	1.
...	
700792	54.018855	0.215318	0.
700843	11.077239	3.175977	2.
700848	3.687145	9.964012	1.
700962	5.914416	0.008577	0.
701063	3.000823	0.148435	0.

15909 rows × 9 columns



```
data2 = data2.drop(['used_chip'], axis=1)
data2
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase
15	179.665148	0.120920	0.
51	43.281314	3.367793	0.
55	24.268906	0.136521	1.
98	6.136181	2.579574	1.
138	5.169928	0.534060	1.
...	
700792	54.018855	0.215318	0.
700843	11.077239	3.175977	2.
700848	3.687145	9.964012	1.
700962	5.914416	0.008577	0.
701063	3.000823	0.148435	0.

15909 rows × 8 columns



```
data2 = data2.drop(['used_pin_number'], axis=1)
data2
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase
15	179.665148	0.120920	0.
51	43.281314	3.367793	0.
55	24.268906	0.136521	1.
98	6.136181	2.579574	1.
138	5.169928	0.534060	1.
...	
700792	54.018855	0.215318	0.
700843	11.077239	3.175977	2.
700848	3.687145	9.964012	1.
700962	5.914416	0.008577	0.
701063	3.000823	0.148435	0.

15000 rows x 7 columns

```
data2 = data2.drop(['online_order'], axis=1)
data2
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase
15	179.665148	0.120920	0.
51	43.281314	3.367793	0.
55	24.268906	0.136521	1.
98	6.136181	2.579574	1.
138	5.169928	0.534060	1.
...	
700792	54.018855	0.215318	0.
700843	11.077239	3.175977	2.
700848	3.687145	9.964012	1.
700962	5.914416	0.008577	0.
701063	3.000823	0.148435	0.

15909 rows x 6 columns



```
print(data2.shape)
```

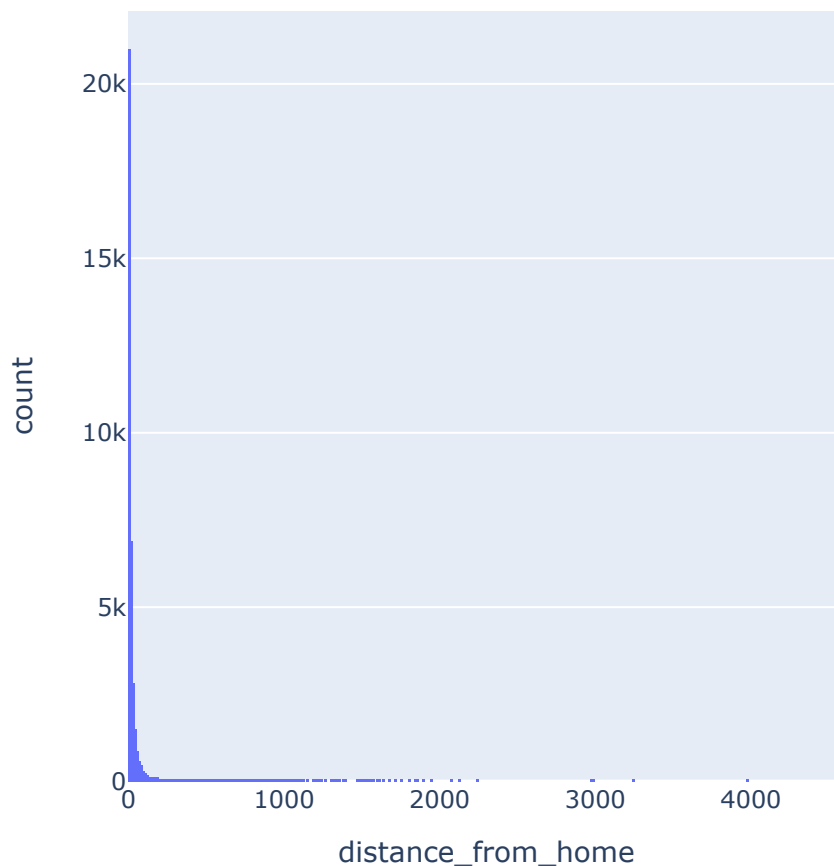
(15909, 6)

Resposta para Hipótese 2

"Os usuários geralmente fazem compras online em lugares familiares, ou em casa ou no trabalho, por isso compras realizadas longe de casa ou muito distante da última compra são suspeitas."

histograma da distância da compra em relação a casa do usuário

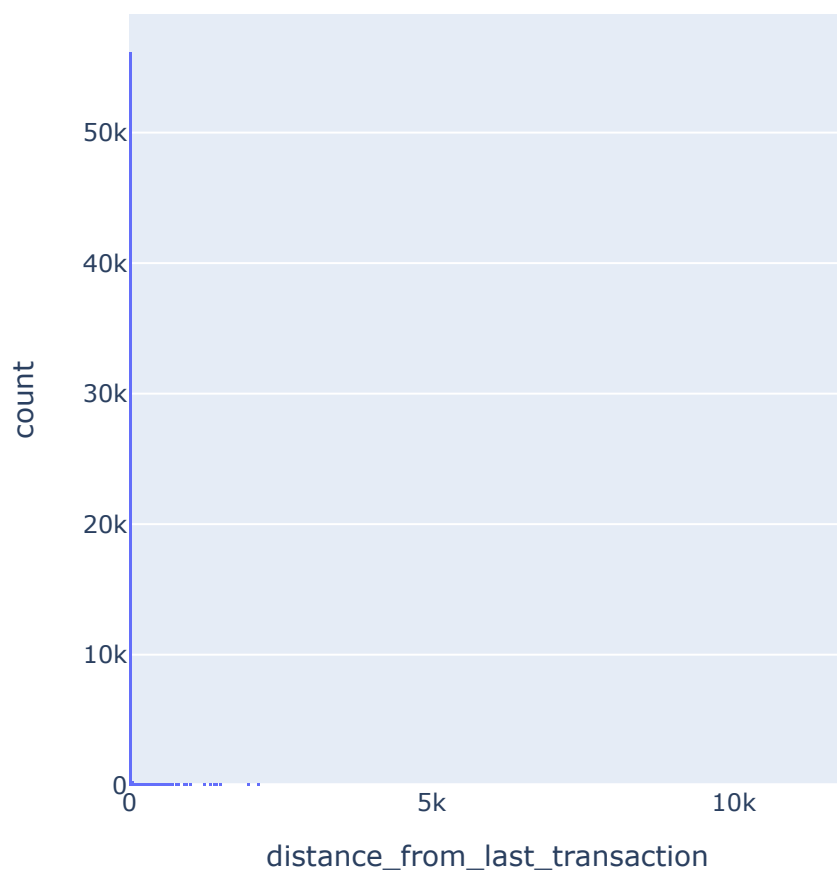
```
import plotly.express as px
px.histogram(onlyonline, x = 'distance_from_home')
```



Double-click (or enter) to edit

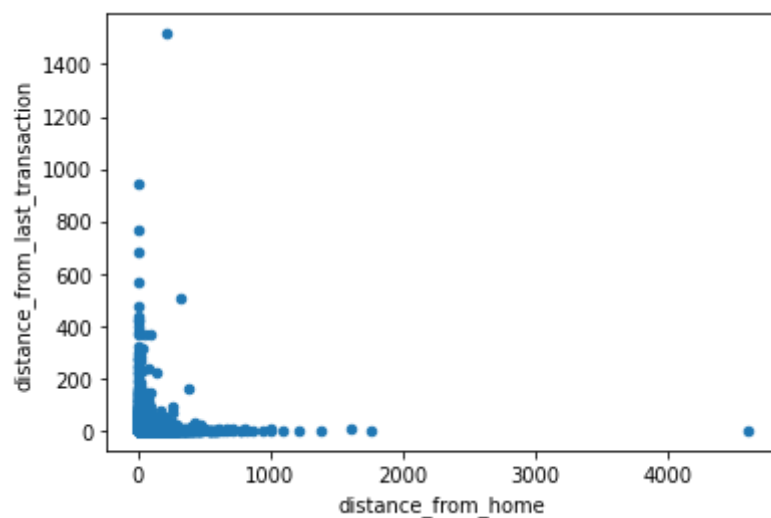
histograma da distância da compra em relação a última compra realizada

```
import plotly.express as px
px.histogram(onlyonline, x = 'distance_from_last_transaction')
```



```
import matplotlib.pyplot as plt
data2.plot.scatter('distance_from_home', 'distance_from_last_transaction')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fad0ce33fd0>



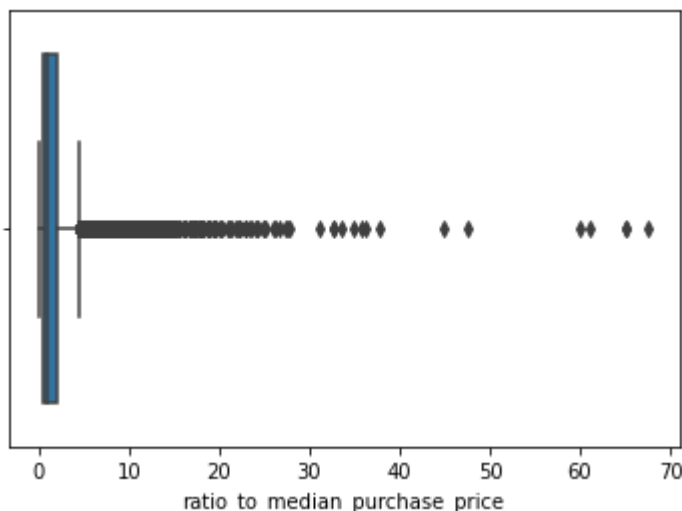
Representação com outliers para melhor visualizar os registros que destoam da média de valor médio gasto.

Dessa forma é notório que os registros se concentram em até 5 vezes mais o valor médio de compra.

Respondendo nossa hipótese nº 3 "Usamos o Ratio_to_median_purchase_price que nos revela a mediana do preço médio de compra. Com isso, consideramos que a grande maioria das compras devem ser até 10 vezes o valor de compra médio de cada usuário. Pois assim, conseguimos separar compras que estão distantes do normal das transações."

```
sns.boxplot(data2['ratio_to_median_purchase_price'])
```

/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only valid po
<matplotlib.axes._subplots.AxesSubplot at 0x7fad0bf704d0>



Arvore de decisão

```
from sklearn.tree import DecisionTreeClassifier
```

```
arvore = DecisionTreeClassifier(criterion='entropy')  
arvore.fit(X_train, Y_train)
```

```
DecisionTreeClassifier(criterion='entropy')
```

```
arvore.feature_importances_
```

```
array([1.48997107e-02, 2.58654101e-02, 8.49040540e-01, 1.49223520e-04,
```

0.00000000e+00, 1.10045116e-01, 0.00000000e+00])

```
from sklearn import tree
figura, eixos = plt.subplots(nrows = 1, ncols = 1, figsize = (10,10))
tree.plot_tree(arvore)
```



```

singleItemFrequency = {}

for item in listOfItems:
    frequency = 0
    for i in range(len(registros)):
        if(item in registros[i]):
            frequency += 1
    singleItemFrequency[item] = frequency

print(singleItemFrequency)

```

```
{'0.5356404825310114': 1, '1.0': 90, '0.6760582745829428': 1, '1.5017878561262346': 1}
```

```

supportThreshold = sThreshold / len(registros)
print("Suporte mínimo: ", supportThreshold)

```

```
Suporte mínimo: 0.001
```

```

afterCleaning = {}

for key, value in singleItemFrequency.items():
    if(value > supportThreshold):
        afterCleaning[key]= value

print (afterCleaning)

```

```
{'0.5356404825310114': 1, '1.0': 90, '0.6760582745829428': 1, '1.5017878561262346': 1}
```

```

def is_in_array(item1, item2, tocheck):
    for i in range(len(tocheck)):
        if item1 in tocheck[i] and item2 in tocheck[i]:
            return True
    return False

itemFrequency = []

for item1 in listOfItems:
    for item2 in listOfItems:
        if(item1 == item2):
            continue
        frequency = 0
        isIn = is_in_array(item1, item2, itemFrequency)
        if (isIn):
            continue
        for i in range(len(registros)):
            if(item1 and item2 in registros[i]):
                frequency += 1
        itemFrequency.append([item1,item2,frequency])

```

```
print(itemFrequency)
```

```
[[ '0.5356404825310114', '1.0', 90], [ '0.5356404825310114', '0.6760582745829428', 1],
```

```
twoItemsAfterCleaning = []
```

```
for i in range(len(itemFrequency)):
    if(itemFrequency[i][2] > supportThreshold):
        twoItemsAfterCleaning.append(itemFrequency[i])
print (twoItemsAfterCleaning)
```

```
[[ '0.5356404825310114', '1.0', 90], [ '0.5356404825310114', '0.6760582745829428', 1],
```

```
for item in twoItemsAfterCleaning:
    value = singleItemFrequency[item[0]]
    if value <= 0:
        value = 1
    confiance = (item[2]/numberOfTransactions)/value
print ("confiança da comparação da Media Das Compras daquele usuario e se essa transação j
```

```
confiança da comparação da Media Das Compras daquele usuario e se essa transação já c
```

▼ Algoritmo de Clusterização - KMeans

Utilizaremos nesse processo de agrupamento e análise, as colunas (distance_from_home, distance_from_last_transaction).

Essas colunas foram selecionadas, baseado na hipótese que compras realizada muito distante da casa do cliente e que também foi distante da última compra efetuada foge do comum de uma transação não fraudulenta.

```
V = data2.drop(['ratio_to_median_purchase_price'],('repeat_retailer'),('fraud'),('credit_
```

```
V = V.iloc[:,[0,1]].values #Deixando somente os dados brutos
```

```
V = pd.DataFrame(V) #conversao em data frame para realizar estracao utilizando (sample)
```

```
V = V.sample(200) #Extração randomica para retirada de amostra de 200 casos.
```

```
print(V.shape) #Confirmacao de tamanho de novo data com amostra de 200
```

(200, 2)

```
print(V.iloc[0:6,:]) #verificando dados existente separados
```

		0	1
14466	123.940520	0.253659	
2658	0.952618	0.782716	
9619	5.651673	0.387412	
5593	6.786892	0.412772	
1463	7.665412	1.325145	
14521	32.853787	0.213621	

Após preparação de dados

▼ Realizacao de Verificação

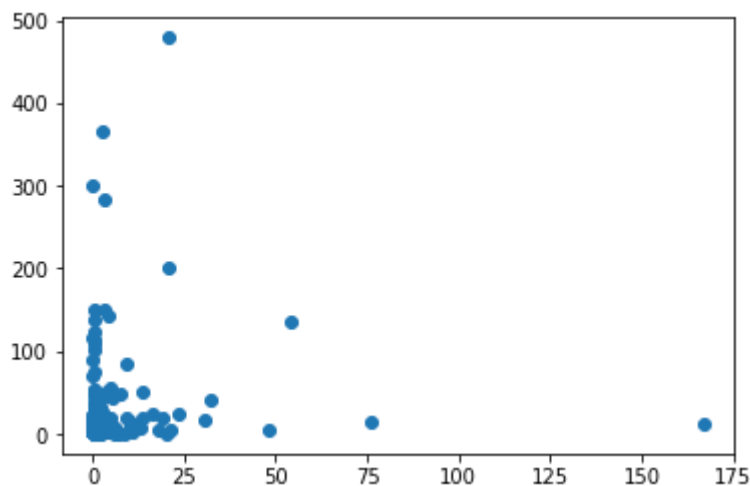
```
from sklearn.cluster import KMeans
```

```
import matplotlib.pyplot as plt
```

```
kmeans = KMeans(n_clusters=3, init = 'random', random_state = 1)
y_kmeans = kmeans.fit_predict(V)
```

```
plt.scatter(V.iloc[:,1], V.iloc[:,0])
```

<matplotlib.collections.PathCollection at 0x7fad0c5f5850>

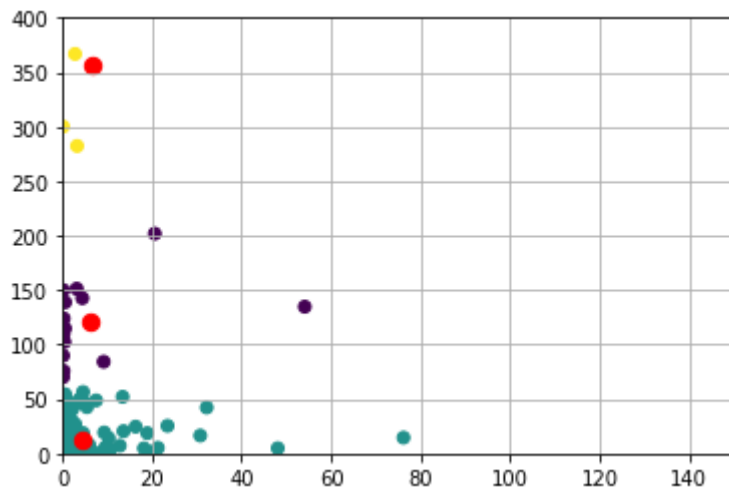


```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', n_init = 10, max_iter = 300)
pred_y = kmeans.fit_predict(V)
```

```
plt.scatter(V.iloc[:,1], V.iloc[:,0], c = pred_y) #posicionamento dos eixos x e y
plt.grid()
plt.xlim(0, 150) #range do eixo x
plt.ylim(0, 400) #range do eixo y
```



```
plt.scatter(kmeans.cluster_centers_[ :,1],kmeans.cluster_centers_[ :,0], s = 70, c = 'red')  
plt.show()
```



[Colab paid products](#) - [Cancel contracts here](#)

