Autores: Prof. João Nogueira e Prof. Dino Magri

Contato: joaonogueira@fisica.ufc.br e professor.dinomagri@gmail.com

Projeto - Parte 5 - Aprendizagem Não Supervisionada - Clusterização - Exercícios

NOTA MÁXIMA: 25 pontos

NOME COMPLETO: MARCIO FERNANDES CRUZ, EDUARDO BOLYHOS, ALISSON SALES

Definição do Problema

O departamento de Marketing quer entender o comportamento dos clientes do iFood, para isso fez uma solicitação para o time de Ciência de Dados para criar modelos de Machine Learning para segmentar os clientes.

```
1 !pip install yellowbrick==1.4 scikit-learn==1.0.2 # Lembre-se de reiniciar o ambiente
```

```
Requirement already satisfied: yellowbrick==1.4 in /usr/local/lib/python3.7/dist-packages (1.4)
Requirement already satisfied: scikit-learn==1.0.2 in /usr/local/lib/python3.7/dist-packages (1.0.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from yellowbrick==1.4) (1.4.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.7/dist-packages (from yellowbrick==1.4) (1.21.6)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from yellowbrick==1.4) (3.2.2)
Requirement already satisfied: cycler>=0.10.0 in /usr/local/lib/python3.7/dist-packages (from yellowbrick==1.4) (0.11.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn==1.0.2) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn==1.0.2) (1.1.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.2->yellowb
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.2->yell
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib!=3.0
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib!=3.0.0,>=2
```

Carregando o conjunto de dados

```
1 from google.colab import drive
2 drive.mount('/content/drive')
3
4 from google.colab import auth
5 auth.authenticate_user()
6
7 from IPython.display import clear_output
8
9 import requests
10 gcloud_token = !gcloud auth print-access-token
11 gcloud_tokeninfo = requests.get('https://www.googleapis.com/oauth2/v3/tokeninfo?access_token=' + gcloud_token[0]).json()
12
13 email_logado = gcloud_tokeninfo['email']
14
15 clear_output()
```

```
1 import os
2 import numpy as np
3 import pandas as pd
4
5 if email_logado=='marcio@marciofcruz.com':
6   pasta_raiz = '/content/drive/MyDrive/Colab Notebooks/DataSet Data Science/ifood'
7 elif email_logado=='alissondeandrade@gmail.com':
8   pasta_raiz = '/content/drive/MyDrive/MBA_FIA_DS_IA/DISCIPLINAS/3.IA/2021_2022_projeto_IA/projeto-ia-datasets/ifood'
```

```
elif email_logado=='sonekabolyhos@gmail.com':
  pasta_raiz = '/content/drive/MyDrive/Aulas Fia/Dataset'
else:
  pasta_raiz = "Não encontrado o email"

print(f'A pasta raiz do {email_logado} é {pasta_raiz}')
```

```
A pasta raiz do alissondeandrade@gmail.com é /content/drive/MyDrive/MBA_FIA_DS_IA/DISCIPLINAS/3.IA/2021_2022_projeto_IA/projeto-ia-
```

```
df_orders = pd.read_csv(f'{pasta_raiz}/df_orders.csv', parse_dates=['order_timestamp_local'])
df_orders.head(3)
```

	order_id	order_number	order_timestamp_local	order_shift	last_status_dat
0	630e2af0-b456-4b3a-b964-4d66ce5cc5df	1290139943	2019-12-11 13:22:06.497000+00:00	weekday lunch	2 11T15:22:
1	66f8163d-f081-4fbd-adb8-cfbbff9213f7	1141271215	2019-11-06 17:57:17.967000+00:00	weekday dinner	2 06T18:18:
2	e8c55557-81d4-4159-bd2b-c144e5fbabe3	694742752	2019-06-11 20:42:32.058000+00:00	weekday dinner	2 11T22:42:

3 rows × 43 columns

Segmentação por Tipo de Comida (12 pontos)

(1 ponto) `Q1` Crie a ABT de segmentação por tipo de comida utilizando as colunas `customer_id` e `merchant_dish_type` e utilizando uma frequência normalizada para cada `customer_id`, em outras palavras, utilizando o percentual de gasto que ele teve em cada categoria de comida.

Salve o resultado na variável `abt_seg_tipo_comida`

```
abt_seg_tipo_comida2 = pd.crosstab(df_orders.customer_id,df_orders.merchant_dish_type , normalize='index').reset_index()
abt_seg_tipo_comida = abt_seg_tipo_comida2.drop(columns='customer_id').copy()
```

```
abt_seg_tipo_comida.shape
```

```
(30079, 57)
```

CHECKPOINT - O DataFrame `abt_seg_tipo_comida` deverá conter exatamente `(30079, 57)`

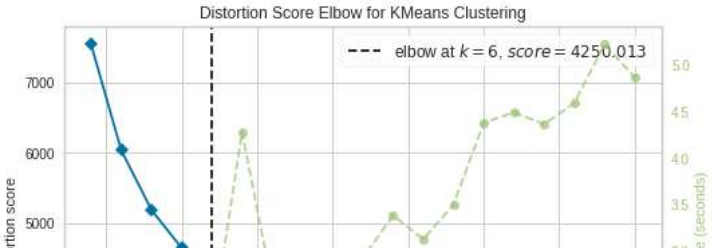
(2 pontos) `Q2` Utilizando a biblioteca `yellowbrick` crie o gráfico do cotovelo para visualizar quantos clusters devemos utilizar no KMeans.

Os seguintes parâmetros devem ser considerados:

- `random_state=15`
- `k=(2, 21)`

Utilize a variável `abt_seg_tipo_comida` para testar com k-clusters.

```
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer

kmeans = KMeans(random_state=15)
visualizer = KElbowVisualizer(kmeans, k=(2, 21))
visualizer.fit(abt_seg_tipo_comida)
visualizer.show();
```



(1 ponto) `Q3` Qual é a quantidade de cluster sugerida pela biblioteca `yellowbriks` ? Você concorda com esse valor para k?

[RESPOSTA]: Foram sugeridos 6 clusters. Concordamos com a indicação mas, a partir da observação do decaimento do gráfico, é possível também sugerir a utilização de 4 ou 5 clusters. Fato que poderia ser testado.

(2 pontos) `Q4` Ajuste o KMeans com os seguintes parâmetros:

- `n_clusters=9`
- `random_state=15`

Após crei uma nova coluna no DF `abt_seg_tipo_comida` chamada `cluster_categorias` que deverá conter o número do cluster associado as características de cada linha.

```
1 kmeans = KMeans(n_clusters=9, random_state=15)
2 kmeans.fit(abt_seg_tipo_comida)
3 abt_seg_tipo_comida['cluster_categorias'] = kmeans.labels_
4 abt_seg_tipo_comida.head()
```

merchant_dish_type	Africana	Argentina	Açaí	Baiana	Bebidas	Cafeteria	Carnes
0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
1	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
2	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
3	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
4	0.0	0.0	0.0	0.0	0.0	0.0	0.333333

(2 pontos) `Q5` Faça a avaliação do ajuste do KMeans utilizando o Mapa de Calor.

O que podemos concluir?

```
1 import seaborn as sns
2 import matplotlib.pyplot as plt
```

```
1 fig, ax = plt.subplots(figsize=(25, 6))
2 sns.heatmap(abt_seg_tipo_comida.groupby('cluster_categorias').mean().round(1), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f345a9cb290>
```


[RESPOSTA]: Existem alguns clusters que claramente agrupam a preferência, como o custer 2 em que 90% dos pedidos se enquadram na categoria "Lanches" e o cluster 8 que tem preferência pela categoria "Comida Brasileira". Também é possível observar clusters que reunem várias categorias, como o cluster 6, onde não é possível estabelecer claramente uma preferência.

(4 pontos) `Q6` Utilize a técnica Surrougate Tree para selecionar as 9 variáveis mais importantes.

- Ajuste a Árvore de Decisão. Lembre-se de no fit remover a coluna `cluster_categorias` das features (colunas), uma vez que é isso que queremos identificar, sendo o nosso alvo. (1 ponto)
- Recupere as 9 features mais importantes. (1 ponto)
- Faça o mapa de calor para facilitar as conclusões sobre os grupos e quais variáveis impactam mais cada grupo gerado. (1 ponto)
- Faça uma análise do resultado obtido no mapa de calor. (1 ponto)

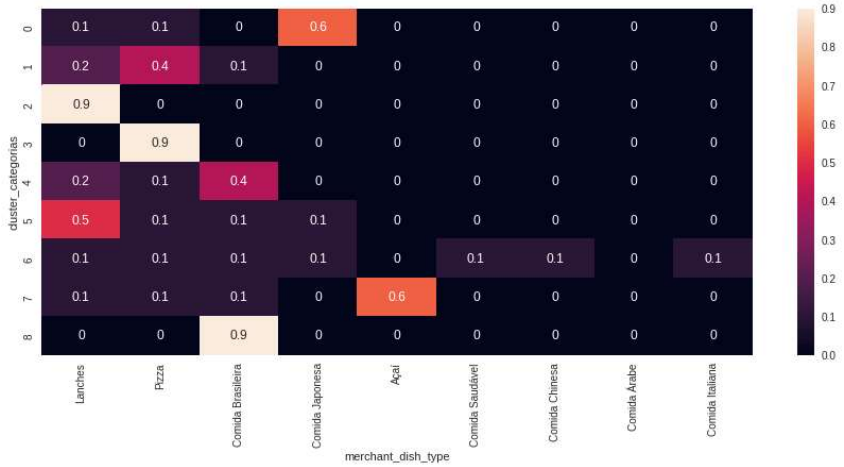
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
X = abt_seg_tipo_comida.iloc[:, 1:-1]
y = abt_seg_tipo_comida.cluster_categorias
clf.fit(X, y)
```

```
DecisionTreeClassifier()
```

```
features_mais_importantes = pd.Series(clf.feature_importances_, index=X.columns.to_list()).sort_values(ascending=False).head(9).index
```

```
fig, ax = plt.subplots(figsize=(15, 6))
sns.heatmap(abt_seg_tipo_comida.groupby('cluster_categorias')[features_mais_importantes].mean().round(1), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3457a0b110>
```



[RESPOSTA]: Utilizando a técnica Surrougate Tree foi possível destacar os agrupamentos mais importantes e ratificar as observações realizadas no exercício anterior.

Segmentação por RFV (13 pontos)

(4 pontos) `Q7` Crei a ABT para a segmentação por RFV utilizando a base `df_orders` com as seguintes características:

- Faça o agrupamento pelo `customer_id`. (1 ponto)
- Aplicar a função de agregação `count` para `order_id` e o resultado deve ser salvo em uma nova coluna com o nome de `total_pedidos`. (1 ponto)
- Aplicar a função de agregação `sum` para `order_total` e o resultado deve ser salvo em uma nova coluna com o nome de `total_receita`. (1 ponto)
- Reinicie o index da estrutura criada e salve na variável `abt_rfv`. (1 ponto)

```
abt_rfv = df_orders.groupby('customer_id').agg(
        total_pedidos = ('order_id', 'count'),
```

```
3        total_receita = ('order_total', 'sum')
4    ).reset_index()
```

```
1 abt_rfv
```

	customer_id	total_pedidos	total_receita
0	0001a8e61d8b08ad436e8e6f4adeb399b88df962c72d9d...	6	293.70
1	0001a9f97d01d2696cf70c7657ee2d039388d691720ff9...	6	345.18
2	0004720dc16aed1f98fd79f59736170e0d686199cd9ae5...	7	385.56
3	0006a32816a3af172048de7db87c97c4c8c7ad7e6385fa...	2	103.20
4	00081913eb21cd12aecc831bda704f8c6482723b55e664...	3	153.96
...	...	...	...
30074	fffc6d5755829710151aec9a2f9b7e38fc6157d6854260...	5	300.80
30075	fffca8befe314aa1b1bc4629875f0141689ec77b9e1a16...	26	978.49
30076	fffd4c0add1e365532304b76bad96c2a585bd15be92902...	9	410.87
30077	fffdc611548c91b0cbf436b5a7f9535e2ea6b221ccee50...	11	391.99
30078	ffffd03e8288072c13bd62cb34976a851b4fcd51c890b3...	10	815.61

30079 rows × 3 columns

+ Código + Texto

(1 ponto) `Q8` Faça o ranqueamento (rank) das colunas `total_pedidos` e `total_receita` com a opção `pct=True` e adicione o resultado no mesmo DF `abt_rfv`, sendo o nome da duas novas colunas como `rank_pct_pedidos` e `rank_pct_receita` respectivamente.

```
1 abt_rfv['rank_pct_pedidos'] = abt_rfv['total_pedidos'].rank(pct=True)
2
3 abt_rfv['rank_pct_receita'] = abt_rfv['total_receita'].rank(pct=True)
```

```
1 abt_rfv
```

	customer_id	total_pedidos	total_receita
0	0001a8e61d8b08ad436e8e6f4adeb399b88df962c72d9d...	6	293.70
1	0001a9f97d01d2696cf70c7657ee2d039388d691720ff9...	6	345.18
2	0004720dc16aed1f98fd79f59736170e0d686199cd9ae5...	7	385.56
3	0006a32816a3af172048de7db87c97c4c8c7ad7e6385fa...	2	103.20
4	00081913eb21cd12aecc831bda704f8c6482723b55e664...	3	153.96
...	...	...	...
30074	fffc6d5755829710151aec9a2f9b7e38fc6157d6854260...	5	300.80
30075	fffca8befe314aa1b1bc4629875f0141689ec77b9e1a16...	26	978.49
30076	fffd4c0add1e365532304b76bad96c2a585bd15be92902...	9	410.87
30077	fffdc611548c91b0cbf436b5a7f9535e2ea6b221ccee50...	11	391.99
30078	ffffd03e8288072c13bd62cb34976a851b4fcd51c890b3...	10	815.61

30079 rows × 5 columns

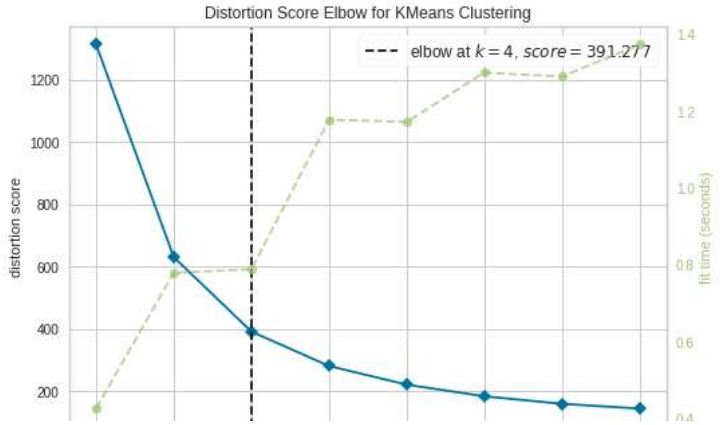
(1 pontos) `Q9` Utilizando a biblioteca `yellowbrick` crie o gráfica do cotovelo para visualizar quantos clusters devemos utilizar no KMeans na base `abt_rfv`.

Os seguintes parâmetros devem ser considerados:

- `random_state=15`
- `k=(2, 10)`

Utilize as features `rank_pct_pedidos` e `rank_pct_receita` do DF `abt_rfv` para testar com k-clusters.

```
1 X_abt_rfv = abt_rfv[['rank_pct_pedidos', 'rank_pct_receita']].copy()
2
3 kmeans = KMeans(random_state = 15)
4 visualizer = KElbowVisualizer(kmeans, k=(2, 10))
5 visualizer.fit(X_abt_rfv)
6 visualizer.show();
```



(1 ponto) `Q10` Ajuste o KMeans com os seguintes parâmetros:

- `n_clusters=4`
- `random_state=15`

Após crei uma nova coluna no DF `abt_rfv` chamada `cluster_fv` que deverá conter o número do cluster associado as características de cada linha.

```
1 kmeans = KMeans(n_clusters=4, random_state=15)
2 kmeans.fit(X_abt_rfv)
3 abt_rfv['cluster_fv'] = kmeans.labels_
```

```
1 abt_rfv
```

	customer_id	total_pedidos	total_rece
0	0001a8e61d8b08ad436e8e6f4adeb399b88df962c72d9d...	6	293
1	0001a9f97d01d2696cf70c7657ee2d039388d691720ff9...	6	345
2	0004720dc16aed1f98fd79f59736170e0d686199cd9ae5...	7	385
3	0006a32816a3af172048de7db87c97c4c8c7ad7e6385fa...	2	103
4	00081913eb21cd12aecc831bda704f8c6482723b55e664...	3	153
...	...	...	
30074	fffc6d5755829710151aec9a2f9b7e38fc6157d6854260...	5	300
30075	fffca8befe314aa1b1bc4629875f0141689ec77b9e1a16...	26	978
30076	fffd4c0add1e365532304b76bad96c2a585bd15be92902...	9	410
30077	fffdc611548c91b0cbf436b5a7f9535e2ea6b221ccee50...	11	391
30078	ffffd03e8288072c13bd62cb34976a851b4fcd51c890b3...	10	815

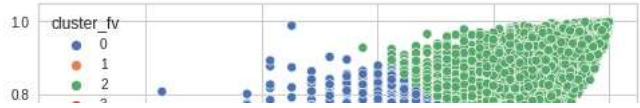
30079 rows × 6 columns

(1 ponto) `Q11` Faça o plot do tipo `scatter` considerando os seguintes parâmetros:

`x='rank_pct_pedidos', y='rank_pct_receita', hue='cluster_fv', s=50, pallete='deep'`

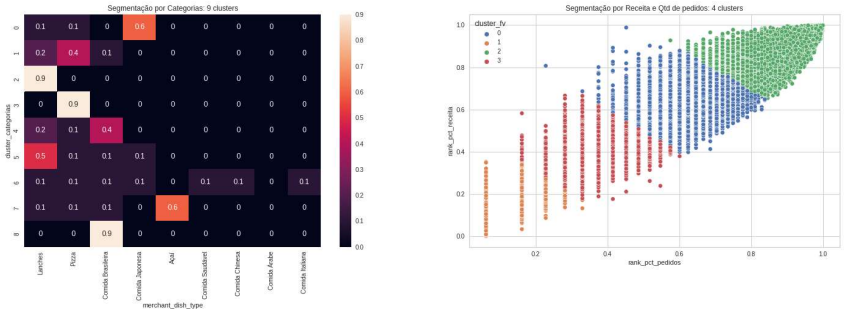
Utilize a biblioteca `seaborn`.

```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3
4 sns.scatterplot(data=abt_rfv, x='rank_pct_pedidos', y='rank_pct_receita', hue='cluster_fv', s=50, palette='deep');
```



(2 pontos) `Q12` Faça um único plot dos dois gráficos criados anteriormente (mapa de calor e o plot do tipo `scatter`). Utilize as variáveis `abt_seg_tipo_comida` e `abt_rfv`.

```
1 fig, ax = plt.subplots(figsize=(24, 7), nrows=1, ncols=2)
2
3 ax[0].set_title('Segmentação por Categorias: 9 clusters');
4 sns.heatmap(abt_seg_tipo_comida.groupby('cluster_categorias')[features_mais_importantes].mean().round(1), annot=True, ax=ax[0]);
5 ax[1].set_title('Segmentação por Receita e Qtd de pedidos: 4 clusters');
6 sns.scatterplot(data=abt_rfv, x='rank_pct_pedidos', y='rank_pct_receita', hue='cluster_fv', s=50, palette='deep',ax=ax[1]);
```



(1 ponto) `Q13` Faça a junção da `abt_seg_tipo_comida` com a `abt_rfv`. Lembre-se de reiniciar o indices da `abt_seg_tipo_comida` e utilize o seguintes parâmetros na junção (`merge`):

- `on='customer_id'`
- `how='left'`

Por fim, filtre apenas as colunas: `'customer_id', 'cluster_categorias', 'cluster_fv'`.

Salve o resultado na variável `resultado_segmentacao`.

```
1 resultado_segmentacaoA = (
2     abt_seg_tipo_comida2
3     .merge(abt_seg_tipo_comida, left_index=True, right_index=True))
```

```
1 resultado_segmentacaoB = (
2     abt_rfv
3     .merge(X_abt_rfv, left_index=True, right_index=True))
```

```
1 resultado_segmentacao = (
2     resultado_segmentacaoA
3     .merge(resultado_segmentacaoB, on='customer_id', how='left')
4     .filter(['customer_id', 'cluster_categorias', 'cluster_fv'])
5     .reset_index()
6 )
7 resultado_segmentacao.head()
```

```
1 resultado_segmentacao = resultado_segmentacao.drop(columns='index').copy()
```

```
1 resultado_segmentacao.head()
```

	customer_id	cluster_categorias	cluster_fv
0	0001a8e61d8b08ad436e8e6f4adeb399b88df962c72d9d...	3	
1	0001a9f97d01d2696cf70c7657ee2d039388d691720ff9...	4	
2	0004720dc16aed1f98fd79f59736170e0d686199cd9ae5...	5	
3	0006a32816a3af172048de7db87c97c4c8c7ad7e6385fa...	1	
4	00081913eb21cd12aecc831bda704f8c6482723b55e664...	5	

(1 ponto) `Q14` Faça o agrupamento pelo `cluster_categorias` e `cluster_fv` para visualizar a quantidade de clientes que existem em cada categoria gerada em relação aos grupos de frequência e venda.

- Utilize a função de agregação `count`
- Utilize a função `unstack` para facilitar a visualização
- Salve na variável `resultado_analise`.

Por fim, crie uma nova coluna, chamada `total`, nesse DF com a soma total de clientes dentro de cada cada grupo do `cluster_categorias`.

```
1 from numpy.ma.core import default_fill_value
2 resultado_analise =  (
3     resultado_segmentacao
4     .groupby(by=["cluster_categorias", "cluster_fv"])
5     .count()
6     .unstack(level=0)
7 )
8
9 resultado_analise["total"] = resultado_analise.sum(axis=1)
10
11 resultado_analise
12
```

	customer_id									total
cluster_categorias	0	1	2	3	4	5	6	7	8	
cluster_fv										
0	388	1517	575	338	1295	2011	996	145	246	7511
1	556	561	1931	1247	589	614	984	360	653	7495
2	326	1203	292	123	1824	1981	1498	109	228	7584
3	445	1274	967	579	1062	1840	821	186	315	7489

(1 ponto) `Q15` Faça uma análise sobre os resultados obtidos nos exercícios Q12, Q13 e Q14.

Para facilitar, utilize a variável `resultado_segmentacao` para verificar a frequencia dos grupos gerados no `cluster_fv` dentro dos grupos do `cluster_categorias`.

[RESPOSTA]:

```
1 soma_total = resultado_analise["total"].sum()
2
3 for i in range(0,9):
4   nome = '{}_'.format(i)
5   resultado_analise[nome] =  resultado_analise["customer_id"][i] / soma_total
6
7
8 resultado_analise.head(10)
```

	customer_id								total	0_
cluster_categorias	0	1	2	3	4	5	6	7	8	

Verificamos que existe uma maior distribuição no cluster de categorias 5, representando 21% total sendo pratos do tipo "lanche". Com base na distribuição do Cluster_FV, identificamos que o cluster com maior receita é o cluster 2, que possui como o principal tipo de prato o lanche. Mesmo sendo um valor agregado não expressivo, a quantidade de vendas acaba sendo mais alta que as demais categorias de clientes.

2	326	1203	292	123	1824	1981	1498	109	228	7584	0.010
3	445	1274	967	579	1062	1840	821	186	315	7489	0.014