Machine Learning Clusterização

Agrupamento de Clientes Por Consumo de Energia

A partir de dados de consumo de energia de clientes, vamos agrupar os consumidores por similaridade a afim de compreender o comportamento dos clientes e sua relação com o consumo de energia.

In [83]:
```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pylab
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from scipy.spatial.distance import cdist, pdist
from sklearn.metrics import silhouette_score
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

In [84]:
```
# Carregando os dados
dataset = pd.read_csv('consumo_energia.txt', delimiter = ';')
```

In [85]:
```
# Visualizando informações sobre o dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2075259 entries, 0 to 2075258
Data columns (total 9 columns):
 #   Column                 Dtype  
---  ------                 -----  
 0   Date                   object 
 1   Time                   object 
 2   Global_active_power    object 
 3   Global_reactive_power  object 
 4   Voltage                object 
 5   Global_intensity       object 
 6   Sub_metering_1         object 
 7   Sub_metering_2         object 
 8   Sub_metering_3         float64
dtypes: float64(1), object(8)
memory usage: 142.5+ MB
```

In [86]:
```
dataset.head()
```

Out[86]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub
0	16/12/2006	17:24:00	4.216	0.418	234.840	18.400	
1	16/12/2006	17:25:00	5.360	0.436	233.630	23.000	
2	16/12/2006	17:26:00	5.374	0.498	233.290	23.000	
3	16/12/2006	17:27:00	5.388	0.502	233.740	23.000	
4	16/12/2006	17:28:00	3.666	0.528	235.680	15.800	

In [87]:
```
dataset.shape
```
Out[87]:
```
(2075259, 9)
```

In [88]:
```
dataset.dtypes
```
Out[88]:
```
Date                     object
Time                     object
Global_active_power      object
Global_reactive_power    object
Voltage                  object
Global_intensity         object
Sub_metering_1           object
Sub_metering_2           object
Sub_metering_3          float64
dtype: object
```

In [89]:
```
# Checando se há valores missing
dataset.isnull().values.any()
```
Out[89]:
```
True
```

In [90]:
```
# Remove os registros com valores NA e remove as duas primeiras colunas (não são ne
dataset = dataset.iloc[0:, 2:9].dropna()
```

In [91]:
```
dataset['Voltage'] = dataset['Voltage'].astype(dtype = 'float64')
dataset['Global_active_power'] = dataset['Global_active_power'].astype(dtype = 'flo
dataset['Global_reactive_power'] = dataset['Global_reactive_power'].astype(dtype = 
dataset['Global_intensity'] = dataset['Global_intensity'].astype(dtype = 'float64')
dataset['Sub_metering_1'] = dataset['Sub_metering_1'].astype(dtype = 'float64')
dataset['Sub_metering_2'] = dataset['Sub_metering_2'].astype(dtype = 'float64')
dataset['Sub_metering_3'] = dataset['Sub_metering_3'].astype(dtype = 'float64')
```

In [92]:
```
dataset.info()
```
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2049280 entries, 0 to 2075258
Data columns (total 7 columns):
 #   Column                 Dtype  
---  ------                 -----  
 0   Global_active_power    float64
 1   Global_reactive_power  float64
 2   Voltage                float64
 3   Global_intensity       float64
 4   Sub_metering_1         float64
 5   Sub_metering_2         float64
 6   Sub_metering_3         float64
dtypes: float64(7)
memory usage: 125.1 MB
```

In [93]:
```
dataset.head()
```

Out[93]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_met
0	4.216	0.418	234.84	18.4	0.0	
1	5.360	0.436	233.63	23.0	0.0	
2	5.374	0.498	233.29	23.0	0.0	
3	5.388	0.502	233.74	23.0	0.0	
4	3.666	0.528	235.68	15.8	0.0	

In [94]:
```
# Checando se há valores missing
dataset.isnull().values.any()
```

Out[94]:
```
False
```

In [95]:
```
# Obtém os valores dos atributos. Neste caso as variaveis foram carregadas como cat
dataset_atrib = dataset.values
```

In [96]:
```
dataset_atrib
```

Out[96]:
```
array([[  4.216,   0.418, 234.84 , ...,   0.   ,   1.   ,  17.   ],
       [  5.36 ,   0.436, 233.63 , ...,   0.   ,   1.   ,  16.   ],
       [  5.374,   0.498, 233.29 , ...,   0.   ,   2.   ,  17.   ],
       ...,
       [  0.938,   0.   , 239.82 , ...,   0.   ,   0.   ,   0.   ],
       [  0.934,   0.   , 239.7  , ...,   0.   ,   0.   ,   0.   ],
       [  0.932,   0.   , 239.55 , ...,   0.   ,   0.   ,   0.   ]])
```

In [97]:
```
# Coleta uma amostra de 1% dos dados para não comprometer a memória do computador
amostra1, amostra2 = train_test_split(dataset_atrib, train_size = .01)
```

In [98]:
```
amostra1.shape
```

Out[98]:
```
(20492, 7)
```

In [99]:
```
# Aplica redução de dimensionalidade
# Transforma as 7 variáveis em 2 variaveis principais. Esse método utiliza Algebra
# entre os dados e assim "juntar" as variaveis, medindo a semelhança pela variância
pca = PCA(n_components = 2).fit_transform(amostra1)
```

In [100…
```
# Determinando um range de K
k_range = range(1,12)
```

In [103…
```
# Aplicando o modelo K-Means para cada valor de K (esta célula pode levar bastante
k_means_var = [KMeans(n_clusters = k).fit(pca) for k in k_range]
```

In [104…
```
# Ajustando o centróide do cluster para cada modelo
centroids = [X.cluster_centers_ for X in k_means_var]
```

In [105…
```
# Calculando a distância euclidiana de cada ponto de dado para o centróide
k_euclid = [cdist(pca, cent, 'euclidean') for cent in centroids]
dist = [np.min(ke, axis = 1) for ke in k_euclid]
```

In [106…
```
# Soma dos quadrados das distâncias dentro do cluster
soma_quadrados_intra_cluster = [sum(d**2) for d in dist]
```

In [107…

```
# Soma total dos quadrados
soma_total = sum(pdist(pca)**2)/pca.shape[0]
```

In [108…

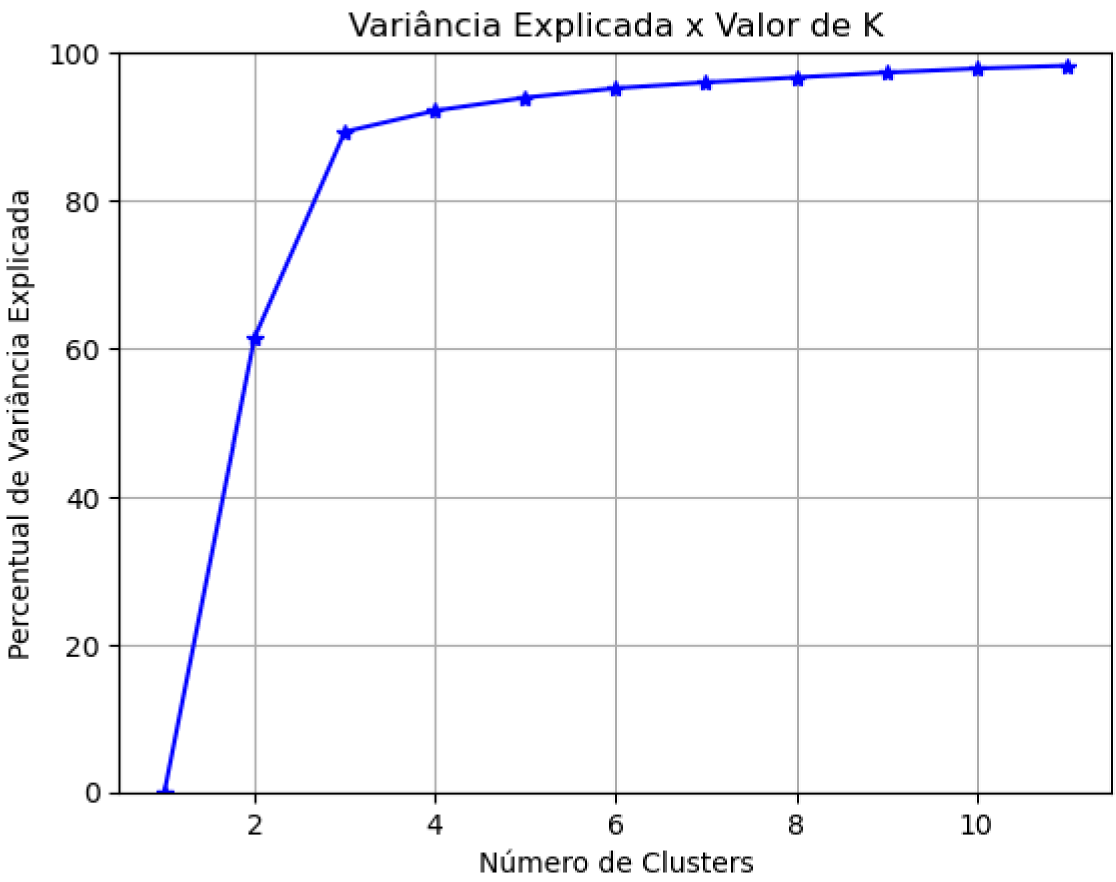
```
# Soma dos quadrados entre clusters
soma_quadrados_inter_cluster = soma_total - soma_quadrados_intra_cluster
```

In [109…

```
# Curva de Elbow
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(k_range, soma_quadrados_inter_cluster/soma_total * 100, 'b*-')
ax.set_ylim((0,100))
plt.grid(True)
plt.xlabel('Número de Clusters')
plt.ylabel('Percentual de Variância Explicada')
plt.title('Variância Explicada x Valor de K')
```

Out[109]:

```
Text(0.5, 1.0, 'Variância Explicada x Valor de K')
```



In [112…

```
# Criando um modelo com K = 8
modelo_v1 = KMeans(n_clusters = 8)
modelo_v1.fit(pca)
```

Out[112]:

```
KMeans()
```

In [ ]:

In [ ]:

In [113…

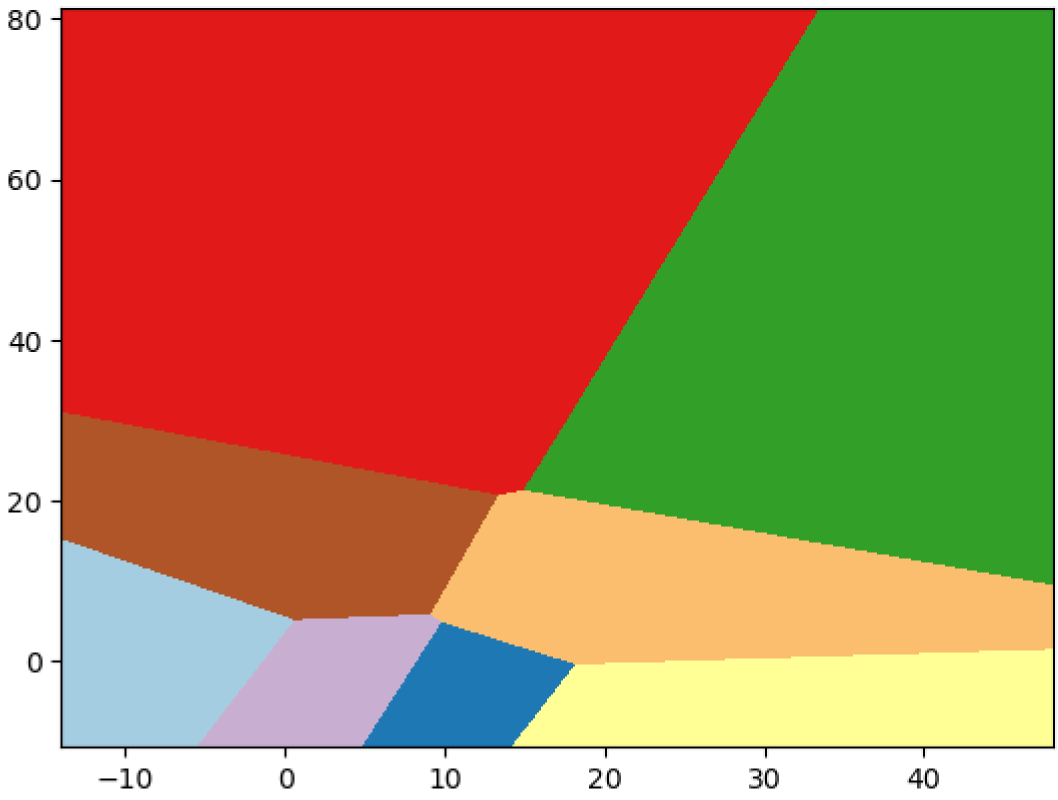
```
# Obtém os valores mínimos e máximos e organiza o shape
x_min, x_max = pca[:, 0].min() - 5, pca[:, 0].max() - 1
y_min, y_max = pca[:, 1].min() + 1, pca[:, 1].max() + 5
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02), np.arange(y_min, y_max, .02))
```

```
Z = modelo_v1.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

In [114…

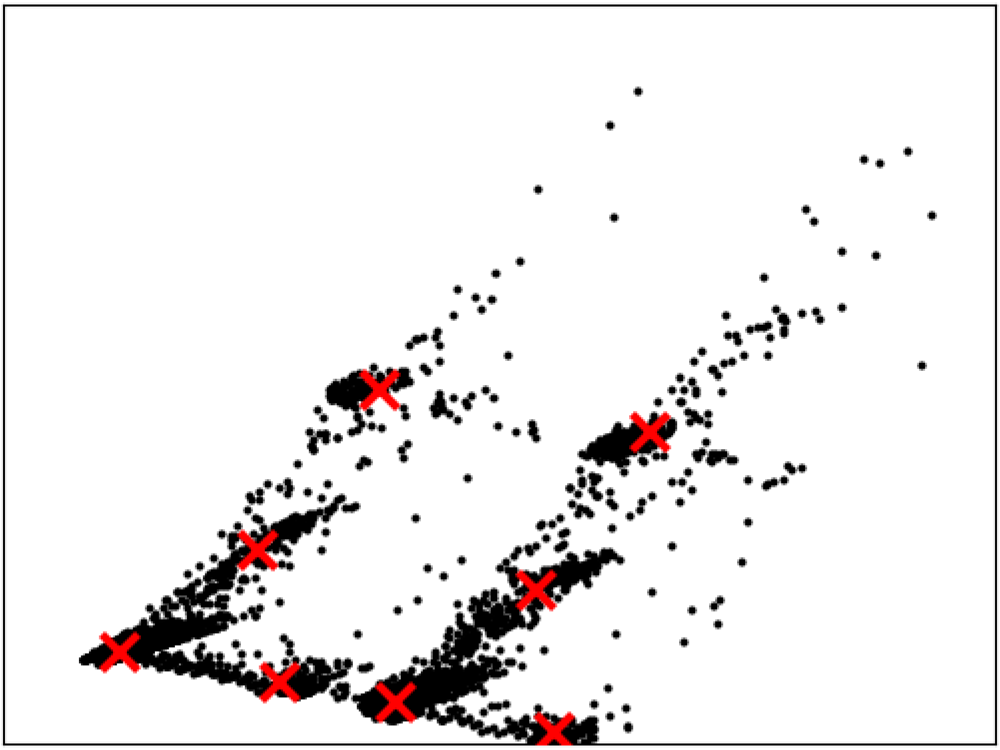
```
# Plot das áreas dos clusters
plt.figure(1)
plt.clf()
plt.imshow(Z,
           interpolation = 'nearest',
           extent = (xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Paired,
           aspect = 'auto',
           origin = 'lower')
```

Out[114]: <matplotlib.image.AxesImage at 0x23c8588cd30>



In [115…

```
# Plot dos centróides
plt.plot(pca[:, 0], pca[:, 1], 'k.', markersize = 4)
centroids = modelo_v1.cluster_centers_
inert = modelo_v1.inertia_
plt.scatter(centroids[:, 0], centroids[:, 1], marker = 'x', s = 169, linewidths = 3
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```



In [117…
```
#?silhouette_score
```

In [118…
```
# Silhouette Score
labels = modelo_v1.labels_
silhouette_score(pca, labels, metric = 'euclidean')
```

Out[118]:
```
0.8088812305122378
```

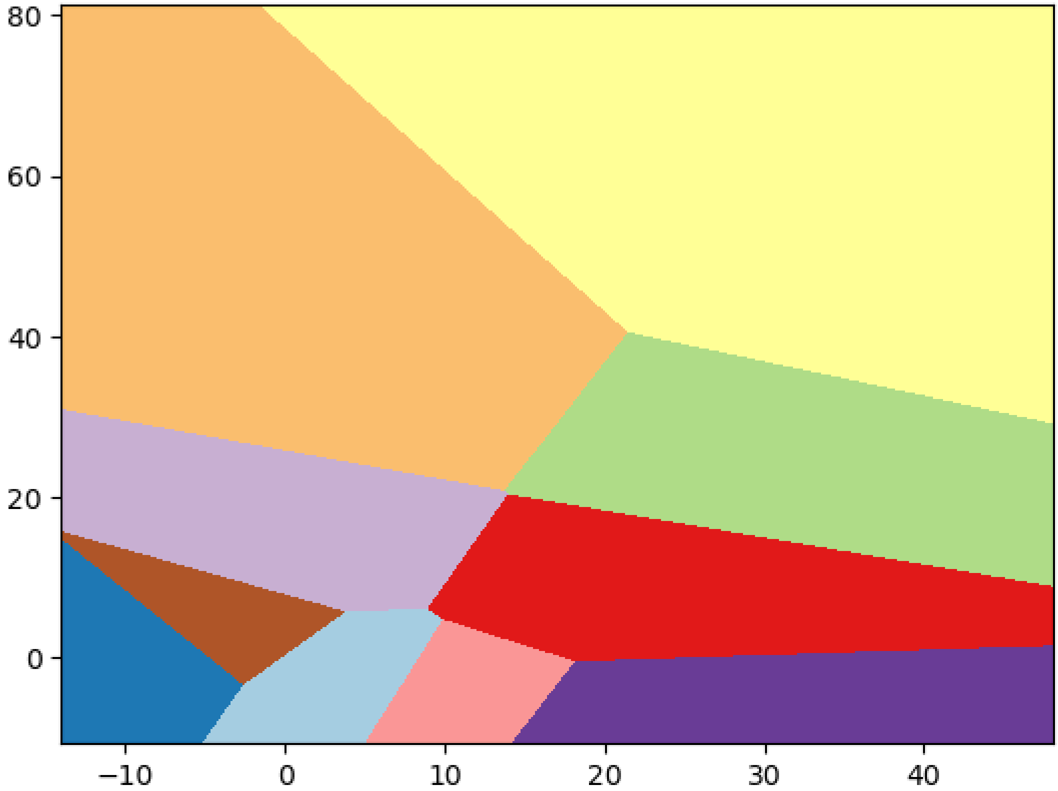
In [119…
```
# Criando um modelo com K = 10
modelo_v2 = KMeans(n_clusters = 10)
modelo_v2.fit(pca)
```

Out[119]:
```
KMeans(n_clusters=10)
```

In [120…
```
# Obtém os valores mínimos e máximos e organiza o shape
x_min, x_max = pca[:, 0].min() - 5, pca[:, 0].max() - 1
y_min, y_max = pca[:, 1].min() + 1, pca[:, 1].max() + 5
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02), np.arange(y_min, y_max, .02))
Z = modelo_v2.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

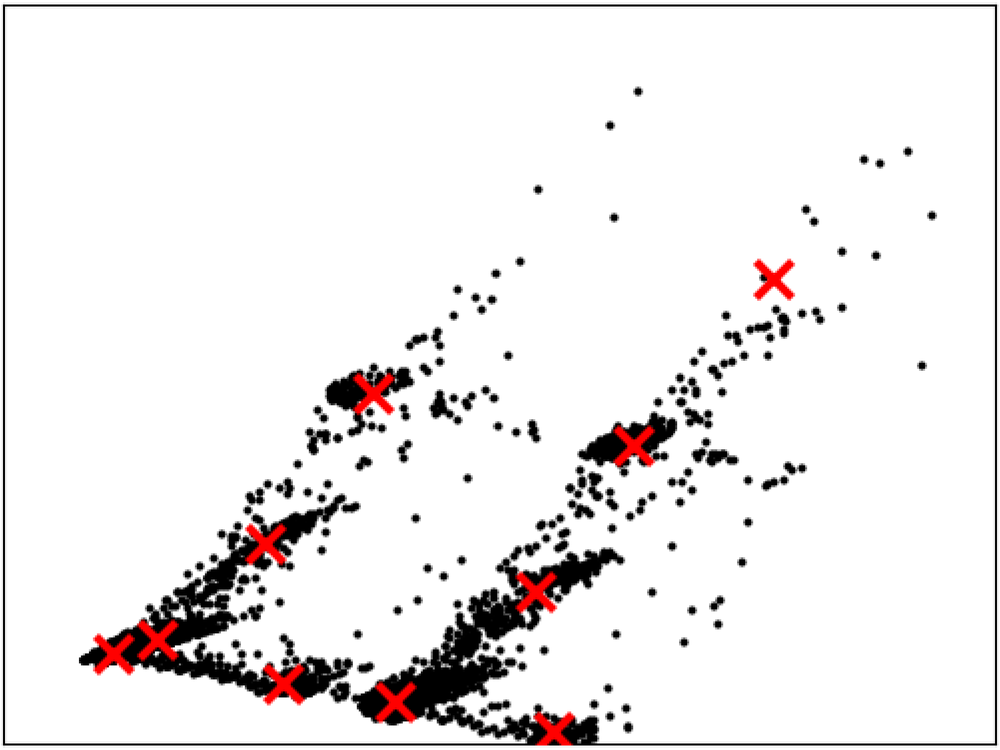
In [121…
```
# Plot das áreas dos clusters
plt.figure(1)
plt.clf()
plt.imshow(Z,
           interpolation = 'nearest',
           extent = (xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Paired,
           aspect = 'auto',
           origin = 'lower')
```

Out[121]:
```
<matplotlib.image.AxesImage at 0x23c998d0040>
```



In [122…

```
# Plot dos centróides
plt.plot(pca[:, 0], pca[:, 1], 'k.', markersize = 4)
centroids = modelo_v2.cluster_centers_
inert = modelo_v2.inertia_
plt.scatter(centroids[:, 0], centroids[:, 1], marker = 'x', s = 169, linewidths = 3
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```



In [123…

```
# Silhouette Score -- utilizamos para avaliar se o numero de cluster é o ideal... m
labels = modelo_v2.labels_
silhouette_score(pca, labels, metric = 'euclidean')
```

Out[123]:

```
0.682286610628249
```

Criando o Cluster Map com os clusters do Modelo V1 que apresentou melhor Silhouette Score.

In [124…

```
# Lista com nomes das colunas
names = ['Global_active_power', 'Global_reactive_power', 'Voltage', 'Global_intensi
```

In [125…

```
# Cria o cluster map
cluster_map = pd.DataFrame(amostra1, columns = names)
cluster_map['Global_active_power'] = pd.to_numeric(cluster_map['Global_active_power
cluster_map['cluster'] = modelo_v1.labels_
```

In [128…

```
cluster_map
```

Out[128]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub
0	0.318	0.114	242.34	1.4	0.0	
1	2.184	0.134	242.47	9.0	0.0	
2	0.392	0.366	243.91	2.2	0.0	
3	3.426	0.000	232.05	14.6	35.0	
4	1.514	0.194	239.18	6.4	0.0	
...	...	...	...	...	...	
20487	0.394	0.000	241.34	1.6	2.0	
20488	1.450	0.108	240.64	6.0	0.0	
20489	0.320	0.084	242.66	1.4	0.0	
20490	0.376	0.000	243.32	1.6	0.0	
20491	0.202	0.000	239.46	0.8	0.0	

20492 rows × 8 columns

In [129…

```
# Calcula a média de consumo de energia por cluster
cluster_map.groupby('cluster')['Global_active_power'].mean()
```

Out[129]:

```
cluster
0    0.515455
1    1.810525
2    4.635189
3    3.511884
4    3.776012
5    1.106406
6    2.352873
7    2.556937
Name: Global_active_power, dtype: float64
```

In [130…

```
# Calcula a quantidade de observacoes por cluster
cluster_map.groupby('cluster')['Global_active_power'].count()
```

Out[130]:
```
cluster
0    12853
1     5894
2      397
3      207
4      324
5      399
6      181
7      237
Name: Global_active_power, dtype: int64
```

In [ ]:

Gerando Cluster com dados normalizados

In [131…
```python
# Obtém os valores dos atributos. Neste caso as variaveis foram carregadas como cat
dataset_atrib = dataset.values

# Importa biblioteca para fazer a normalizacao
from sklearn.preprocessing import MinMaxScaler

# Cria o objeto para normalizar e faz a normalizacao dos dados
Padronizador = MinMaxScaler()
dataset_atrib = Padronizador.fit_transform(dataset_atrib)


amostra1, amostra2 = train_test_split(dataset_atrib, train_size = .01)
pca = PCA(n_components = 2).fit_transform(amostra1)
k_range = range(1,12)
k_means_var = [KMeans(n_clusters = k).fit(pca) for k in k_range]
centroids = [X.cluster_centers_ for X in k_means_var]
k_euclid = [cdist(pca, cent, 'euclidean') for cent in centroids]
dist = [np.min(ke, axis = 1) for ke in k_euclid]
soma_quadrados_intra_cluster = [sum(d**2) for d in dist]
soma_total = sum(pdist(pca)**2)/pca.shape[0]
# Soma dos quadrados entre clusters
soma_quadrados_inter_cluster = soma_total - soma_quadrados_intra_cluster
# Criando um modelo com K = 8
modelo_v1 = KMeans(n_clusters = 8)
modelo_v1.fit(pca)
# Obtém os valores mínimos e máximos e organiza o shape
x_min, x_max = pca[:, 0].min() - 5, pca[:, 0].max() - 1
y_min, y_max = pca[:, 1].min() + 1, pca[:, 1].max() + 5
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02), np.arange(y_min, y_max, .02))
Z = modelo_v1.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
# Silhouette Score
labels = modelo_v1.labels_
silhouette_score(pca, labels, metric = 'euclidean')
# Lista com nomes das colunas
names = ['Global_active_power', 'Global_reactive_power', 'Voltage', 'Global_intensi

# Cria o cluster map
cluster_map = pd.DataFrame(amostra1, columns = names)
cluster_map['Global_active_power'] = pd.to_numeric(cluster_map['Global_active_power
cluster_map['cluster'] = modelo_v1.labels_
cluster_map
```

Out[131]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Su
0	0.113706	0.000000	0.619063	0.107884	0.011364	
1	0.019736	0.000000	0.491438	0.029046	0.000000	
2	0.016658	0.000000	0.745719	0.016598	0.000000	
3	0.015571	0.113669	0.665267	0.020747	0.000000	
4	0.167663	0.034532	0.470436	0.161826	0.000000	
...	...	...	...	...	...	
20487	0.017563	0.054676	0.589338	0.020747	0.000000	
20488	0.055948	0.166906	0.680775	0.058091	0.000000	
20489	0.081115	0.195683	0.602585	0.082988	0.000000	
20490	0.033134	0.143885	0.586430	0.037344	0.000000	
20491	0.138874	0.000000	0.487561	0.132780	0.011364	

20492 rows × 8 columns

In [132…

```
# Calcula da quantidade de observações por cluster
cluster_map.groupby('cluster')['Global_active_power'].count()
```

Out[132]:

```
cluster
0    4276
1    3639
2    9073
3     627
4    1750
5     536
6     417
7     174
Name: Global_active_power, dtype: int64
```

In [ ]:

In [ ]: