dmo_anl_vw_tot_mov_periodo

October 25, 2024

1 Análise da tabela dmo_anl_vw_tot_mov_periodo

Para a análise será necessário, primeiramente, importar a tabela utilizando pandas.

O caminho padrão é

[1]:
```
path = '../copper-etl/dmo_anl_vw_tot_mov_periodo.csv'
```

[2]:
```
import pandas as pd

mov_periodo = pd.read_csv(path, sep=',')
```

1.1 Análise observativa das caracteristicas da tabela

Inicialmente, será observado a caracteristica dos dados da tabela

1.1.1 Primeiros 5 dados

[ ]:
```
mov_periodo.head(5)
```

[ ]:
```
   id_dt_hora_minuto  cod_bilh  cd_estac_bu       dt_validacao  \
0                 46      3001          619  10/24/23 00:00:00
1                 46      3001          619  10/25/23 00:00:00
2                 46      3001          619  10/26/23 00:00:00
3                 46      3001          619  10/27/23 00:00:00
4                 46      3001          619  10/28/23 00:00:00

   total_validacoes tipo_dia
0                59        U
1               107        U
2                67        U
3                66        U
4                78        S
```

1.1.2 Estatisticas básicas das colunas

[4]:
```
mov_periodo.describe()
```

[4]:

	id_dt_hora_minuto	cod_bilh	cd_estac_bu	total_validacoes
count	3.028393e+06	3.028393e+06	3.028393e+06	3.028393e+06
mean	5.379721e+01	2.281298e+04	6.300346e+02	1.247573e+01
std	2.206573e+01	3.160558e+04	8.872497e+01	3.239844e+01
min	1.000000e+00	2.530000e+03	0.000000e+00	1.000000e+00
25%	3.400000e+01	3.000000e+03	5.590000e+02	1.000000e+00
50%	5.400000e+01	5.001000e+03	6.160000e+02	4.000000e+00
75%	7.200000e+01	2.500000e+04	7.070000e+02	1.100000e+01
max	9.600000e+01	9.800000e+04	9.120000e+02	1.894000e+03

1.1.3 Tipo dos dados brutos

[5]:
```
mov_periodo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3028393 entries, 0 to 3028392
Data columns (total 6 columns):
 #   Column             Dtype 
---  ------             ----- 
 0   id_dt_hora_minuto  int64 
 1   cod_bilh           int64 
 2   cd_estac_bu        int64 
 3   dt_validacao       object
 4   total_validacoes   int64 
 5   tipo_dia           object
dtypes: int64(4), object(2)
memory usage: 138.6+ MB
```

1.1.4 Contagem dos valores

[6]:
```
#tamanho do dataset
mov_periodo.shape
```

[6]:
```
(3028393, 6)
```

[7]:
```
def get_value_counts(df, column):
    value_counts = df[column].value_counts()
    print("len", len(value_counts))
    return value_counts

get_value_counts(mov_periodo,'id_dt_hora_minuto')
```

```
len 96
```

[7]:
```
id_dt_hora_minuto
23    47286
30    45838
29    45623
```

```
26    45544
27    45208
      …
11       52
9        46
12       45
7        43
8        35
Name: count, Length: 96, dtype: int64
```

[8]:
```
get_value_counts(mov_periodo,'cod_bilh')
```

```
len 45
```

[8]:
```
cod_bilh
11100    226259
24000    209455
3000     208922
5000     202585
25000    181857
28000    171333
4000     169820
90484    163949
98000    150983
2530     149047
2540     117377
90267    103402
2780      99410
3001      94062
2650      93687
5001      88068
14000     83114
4740      65435
2870      64635
5750      64322
90266     58221
2730      56122
2790      37450
2910      31144
2732      27426
2720      21500
5100      20939
3100      18992
14001     13716
21120     11502
5101       7909
3101       7783
90261      4556
```

```
90260      1382
15000       875
18000       354
11000       232
10000       213
14200       150
18020        64
15001        64
11001        45
90158        19
14201         9
10001         4
Name: count, dtype: int64
```

[9]:
```
get_value_counts(mov_periodo,'cd_estac_bu')
```

```
len 96
```

[9]:
```
cd_estac_bu
517    55335
763    52415
710    52351
764    51372
713    50503
       …
514    14784
512    13038
559    12635
0       1403
912      395
Name: count, Length: 96, dtype: int64
```

[ ]:
```
get_value_counts(mov_periodo,'total_validacoes')
```

```
len 1008
```

[ ]:
```
total_validacoes
1       810847
2       429690
3       269547
4       192086
5       145922
         …
917          1
617          1
955          1
1029         1
942          1
```

```
Name: count, Length: 1008, dtype: int64
```

[11]:
```
get_value_counts(mov_periodo,'dt_validacao')
```

```
len 31
```

[11]:
```
dt_validacao
10/11/23 00:00:00    111225
10/27/23 00:00:00    110118
10/20/23 00:00:00    110064
10/18/23 00:00:00    109680
10/25/23 00:00:00    109661
10/19/23 00:00:00    109517
10/26/23 00:00:00    109439
10/10/23 00:00:00    109439
10/24/23 00:00:00    108850
10/31/23 00:00:00    108677
10/30/23 00:00:00    108413
10/23/23 00:00:00    108161
10/16/23 00:00:00    108144
10/17/23 00:00:00    107602
10/09/23 00:00:00    107334
10/06/23 00:00:00    106196
10/05/23 00:00:00    103636
10/02/23 00:00:00    102483
10/04/23 00:00:00    100798
10/13/23 00:00:00     98731
10/21/23 00:00:00     97003
10/28/23 00:00:00     95931
10/07/23 00:00:00     92193
10/14/23 00:00:00     91545
10/12/23 00:00:00     81896
10/22/23 00:00:00     77833
10/15/23 00:00:00     76508
10/29/23 00:00:00     76245
10/08/23 00:00:00     74755
10/01/23 00:00:00     67937
10/03/23 00:00:00     48379
Name: count, dtype: int64
```

[12]:
```
get_value_counts(mov_periodo,'tipo_dia')
```

```
len 3
```

[12]:
```
tipo_dia
U    2148168
D     503553
S     376672
```

```
Name: count, dtype: int64
```

A análise dos dados demonstra que os dados mencionados são referentes ao período do mês de outubro de 2023. Além disso, é possível deduzir que a coluna tipo_dia é referente a dias úteis, domingos e sábados. Além disso, não há nenhum dado que seja do mesmo tamanho do dataset. Logo, isso demonstra que não há identificadores nas colunas. Por fim, a coluna dt_validacao não contém variação no horário, o que indica que os dados extraidos refletem a todas as validações realizadas no dia.

1.1.5 Alteração dos dados

dt_validacao Como alguns dados não foram importados com o tipo correto, será necessário alterar seus tipos para possibilitar uma melhor análise. Portanto, a coluna dt_validacao será transformada em datetime

[13]:
```
mov_periodo['dt_validacao'] = pd.to_datetime(mov_periodo['dt_validacao'],␣
↪format='%m/%d/%y %H:%M:%S')
mov_periodo.head()
```

[13]:

	id_dt_hora_minuto	cod_bilh	cd_estac_bu	dt_validacao	total_validacoes	\
0	46	3001	619	2023-10-24	59	
1	46	3001	619	2023-10-25	107	
2	46	3001	619	2023-10-26	67	
3	46	3001	619	2023-10-27	66	
4	46	3001	619	2023-10-28	78	

	tipo_dia
0	U
1	U
2	U
3	U
4	S

tipo_dia Dado que será necessário realizar uma tabela de correlação. Será necessário codificar a coluna tipo_dia. Por conta da variedade de dados dessa coluna ser pouca (3), foi utilizado o one hot enconding. O one hot enconding também irá auxiliar a criação de um modelo que utiliza de distâncias, como KNN.

[14]:
```
df_encoded = pd.get_dummies(mov_periodo, columns=['tipo_dia'])
df_encoded.head()
```

[14]:

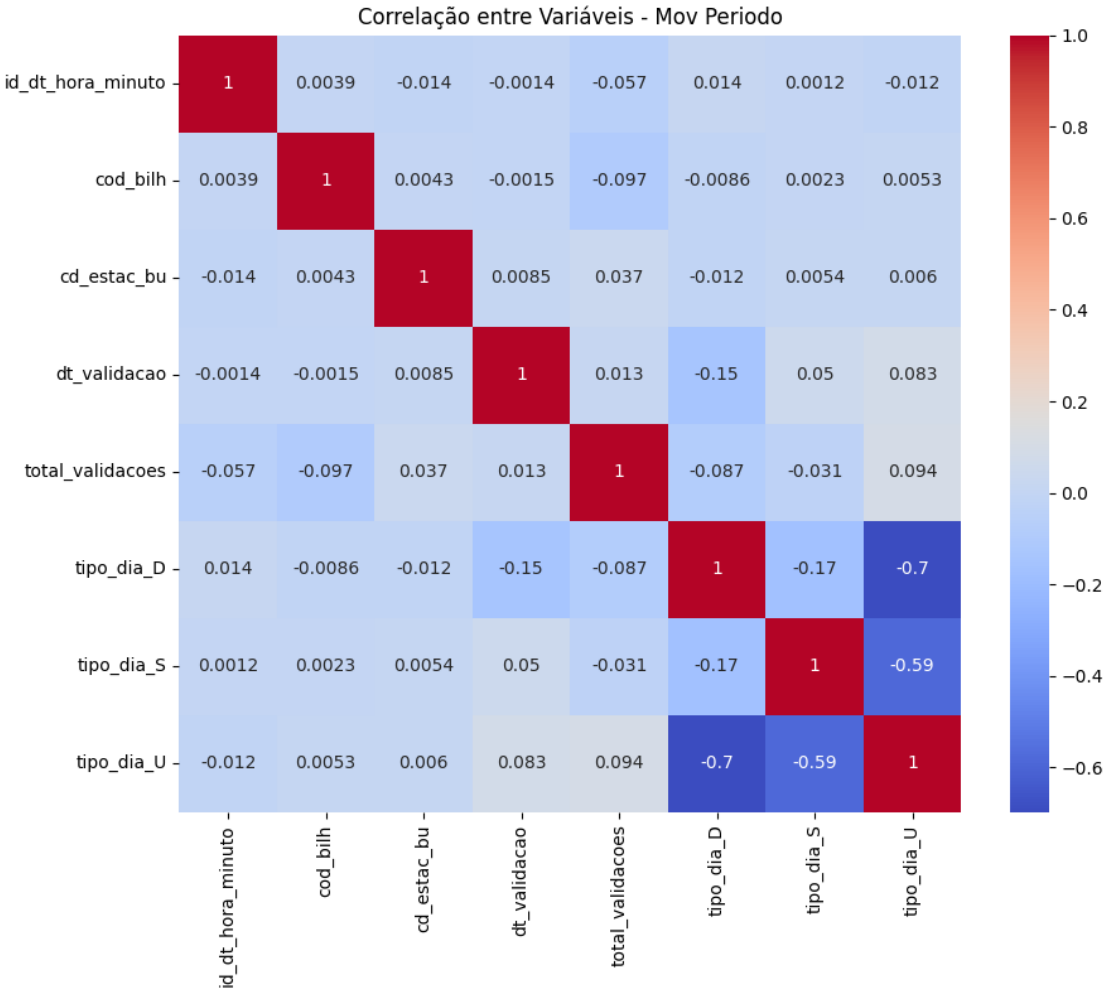
	id_dt_hora_minuto	cod_bilh	cd_estac_bu	dt_validacao	total_validacoes	\
0	46	3001	619	2023-10-24	59	
1	46	3001	619	2023-10-25	107	
2	46	3001	619	2023-10-26	67	
3	46	3001	619	2023-10-27	66	
4	46	3001	619	2023-10-28	78	

```
   tipo_dia_D  tipo_dia_S  tipo_dia_U
0       False       False        True
1       False       False        True
2       False       False        True
3       False       False        True
4       False        True       False
```

1.2 Análise de correlações

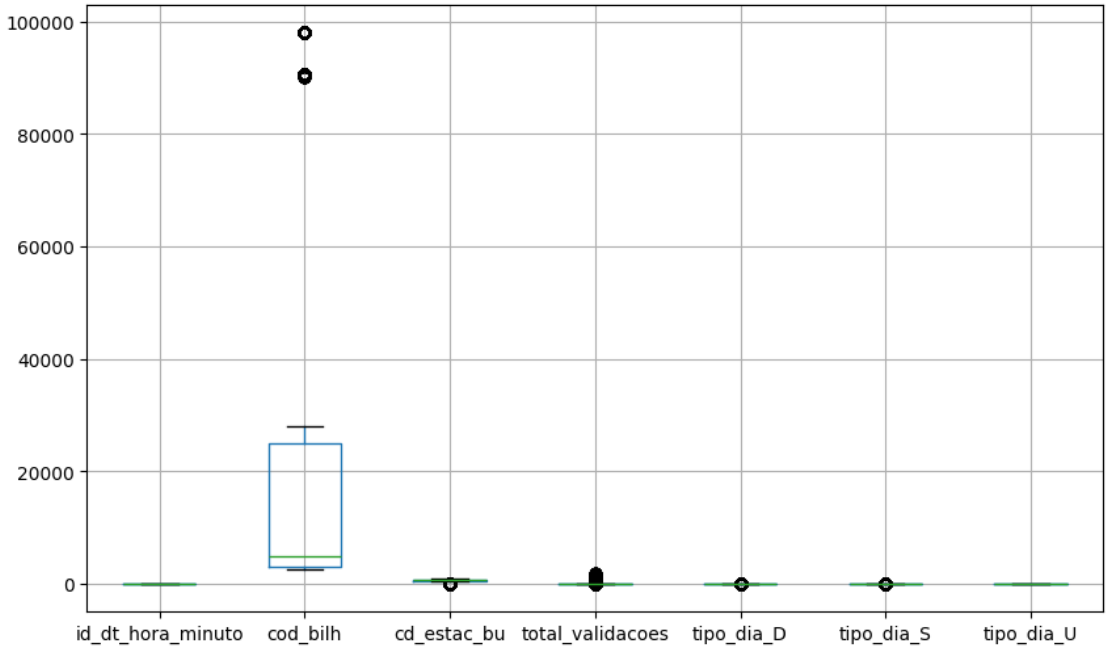
Aqui será feito a análise da correlação entre as colunas do dataset

[16]:
```
import seaborn as sns
import matplotlib.pyplot as plt

corr_mov = df_encoded.corr()

plt.figure(figsize=(10,8))
sns.heatmap(corr_mov, annot=True, cmap='coolwarm')
plt.title('Correlação entre Variáveis - Mov Periodo')
plt.show()
```



1.3 Remoção de outliers

[17]:
```
plt.figure(figsize=(10, 6))
df_encoded.boxplot()
plt.show()
```



Diante desse gráfico, é possível observar que há alguns cod_bilhete que são pouco utilizados. Além disso, há picos no total_validacoes que ocorrem em alguns dias específicos. Por conta que o cod_bilhete é uma variável categórica, ela não será tirada como outlier

[18]:
```
import numpy as np

def detect_outliers(df, columns, threshold=3):
    z_scores = np.abs((df[columns] - df[columns].mean()) / df[columns].std())
    return df[(z_scores < threshold).all(axis=1)]

colunas_especificas = ['id_dt_hora_minuto',␣
 ↪'cd_estac_bu',        'dt_validacao',        'total_validacoes']
mov_no_outliers = detect_outliers(df_encoded, colunas_especificas)

print(f'Tamanho de Mov sem Outliers: {mov_no_outliers.shape}')
mov_no_outliers.head()
```

```
C:\Users\Inteli\AppData\Local\Temp\ipykernel_13292\2023904907.py:4:
PerformanceWarning: Adding/subtracting object-dtype array to DatetimeArray not
vectorized.
  z_scores = np.abs((df[columns] - df[columns].mean()) / df[columns].std())

Tamanho de Mov sem Outliers: (2980433, 8)
```

[18]:

	id_dt_hora_minuto	cod_bilh	cd_estac_bu	dt_validacao	total_validacoes	\
0	46	3001	619	2023-10-24	59	
1	46	3001	619	2023-10-25	107	

```
2                46      3001          619   2023-10-26                67
3                46      3001          619   2023-10-27                66
4                46      3001          619   2023-10-28                78

   tipo_dia_D  tipo_dia_S  tipo_dia_U
0       False       False        True
1       False       False        True
2       False       False        True
3       False       False        True
4       False        True       False
```

[19]:
```
mov_no_outliers.describe()
```

[19]:
```
       id_dt_hora_minuto      cod_bilh   cd_estac_bu  \
count       2.980433e+06  2.980433e+06  2.980433e+06
mean        5.392676e+01  2.302649e+04  6.300588e+02
min         1.000000e+00  2.530000e+03  5.010000e+02
25%         3.500000e+01  3.000000e+03  5.590000e+02
50%         5.400000e+01  5.001000e+03  6.160000e+02
75%         7.200000e+01  2.500000e+04  7.070000e+02
max         9.600000e+01  9.800000e+04  8.030000e+02
std         2.204786e+01  3.177998e+04  8.767596e+01

                      dt_validacao  total_validacoes
count                      2980433      2.980433e+06
mean   2023-10-16 11:36:14.940487168      9.502028e+00
min              2023-10-01 00:00:00      1.000000e+00
25%              2023-10-09 00:00:00      1.000000e+00
50%              2023-10-17 00:00:00      3.000000e+00
75%              2023-10-24 00:00:00      1.000000e+01
max              2023-10-31 00:00:00      1.090000e+02
std                              NaN      1.529596e+01
```

[20]:
```
mov_no_outliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2980433 entries, 0 to 3028392
Data columns (total 8 columns):
 #   Column             Dtype
---  ------             -----
 0   id_dt_hora_minuto  int64
 1   cod_bilh           int64
 2   cd_estac_bu        int64
 3   dt_validacao       datetime64[ns]
 4   total_validacoes   int64
 5   tipo_dia_D         bool
 6   tipo_dia_S         bool
 7   tipo_dia_U         bool
```

```
dtypes: bool(3), datetime64[ns](1), int64(4)
memory usage: 145.0 MB
```

1.4 Tentativa de identificação de grupos

1.4.1 Cálculo do PCA

[22]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
mov_scaled = scaler.fit_transform(mov_no_outliers.
 ↪drop(columns=['dt_validacao']))

pca = PCA(n_components=2)
mov_pca = pca.fit_transform(mov_scaled)

print(f'Variância explicada - Mov: {pca.explained_variance_ratio_}')
```

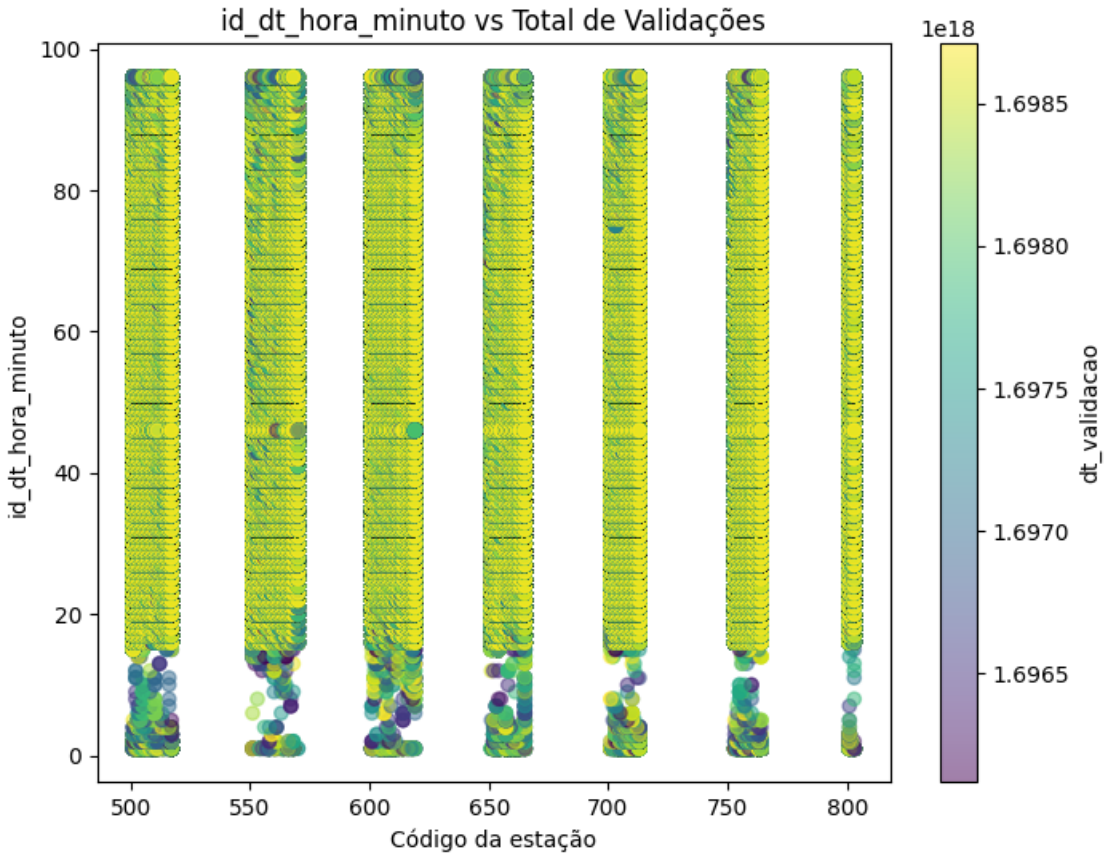
```
Variância explicada - Mov: [0.26510303 0.16949465]
```

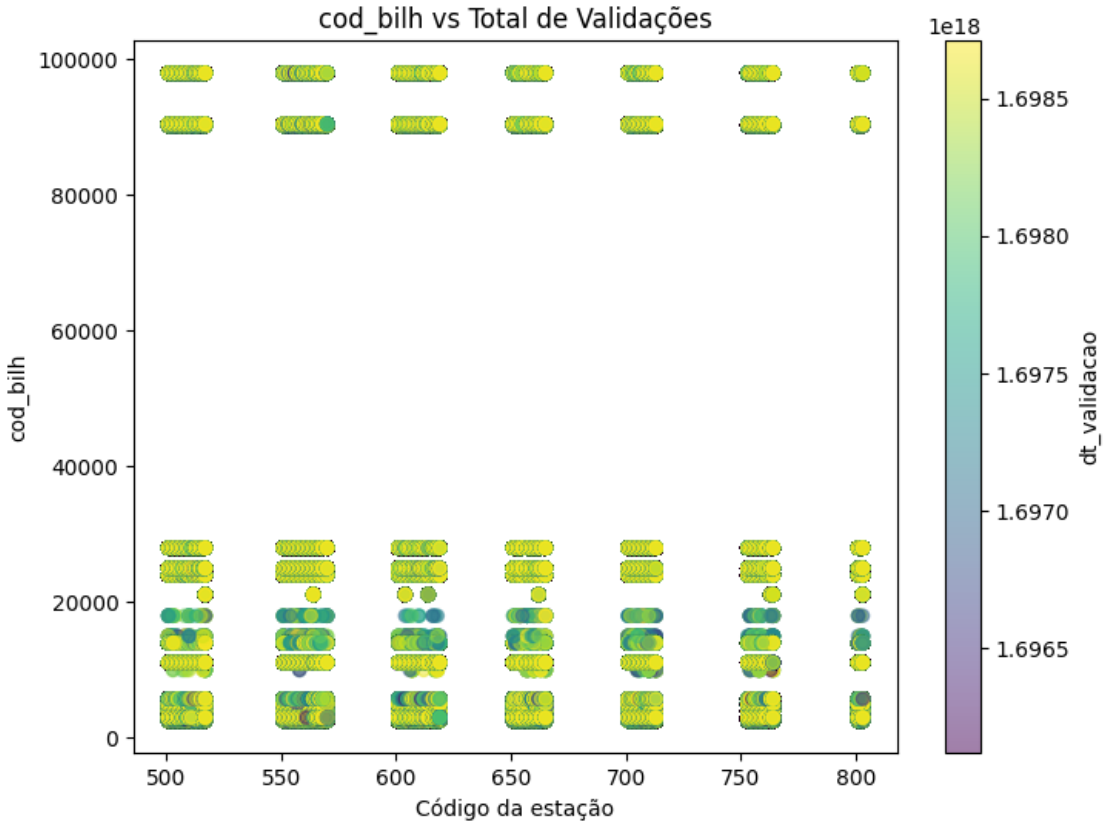
1.4.2 Visualização do PCA

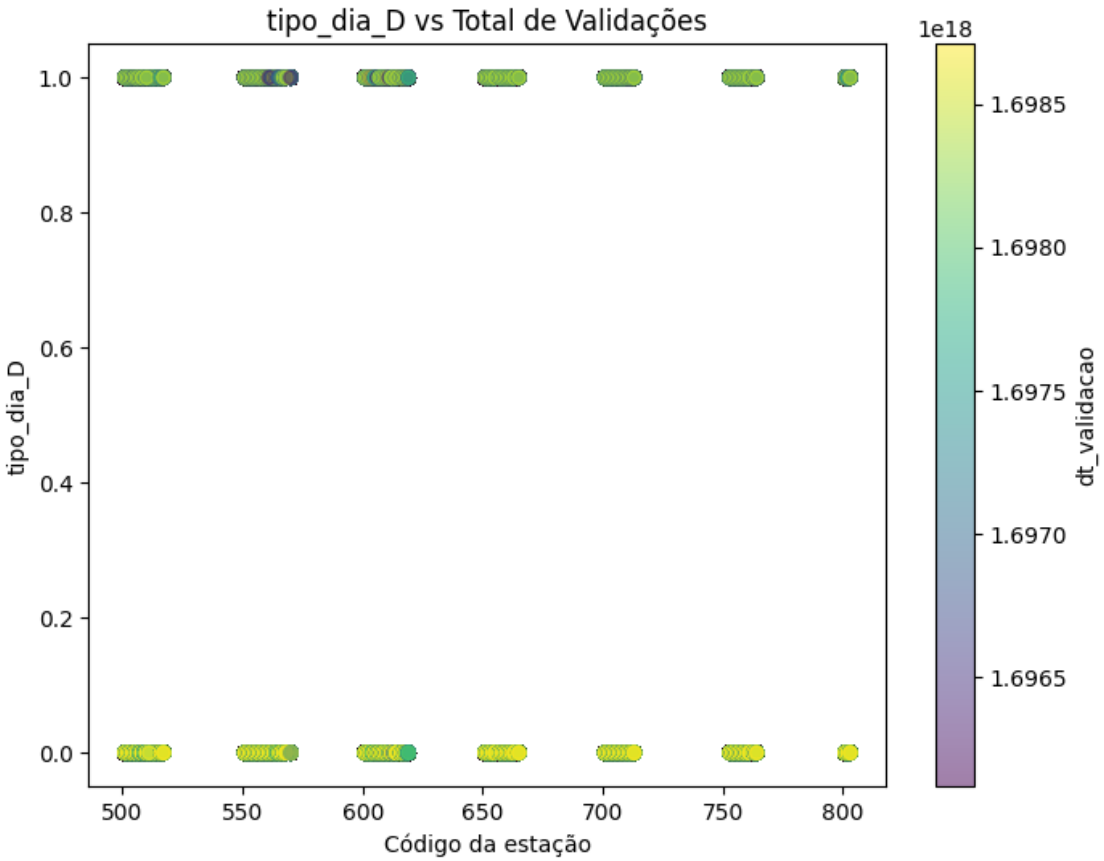
Os gráficos de PCA levam grande tempo para serem feitos. Por conta disso, há uma variável de controle para evitar que eles sempre sejam criados. Mude show_pca para criar ou não os gráficos.

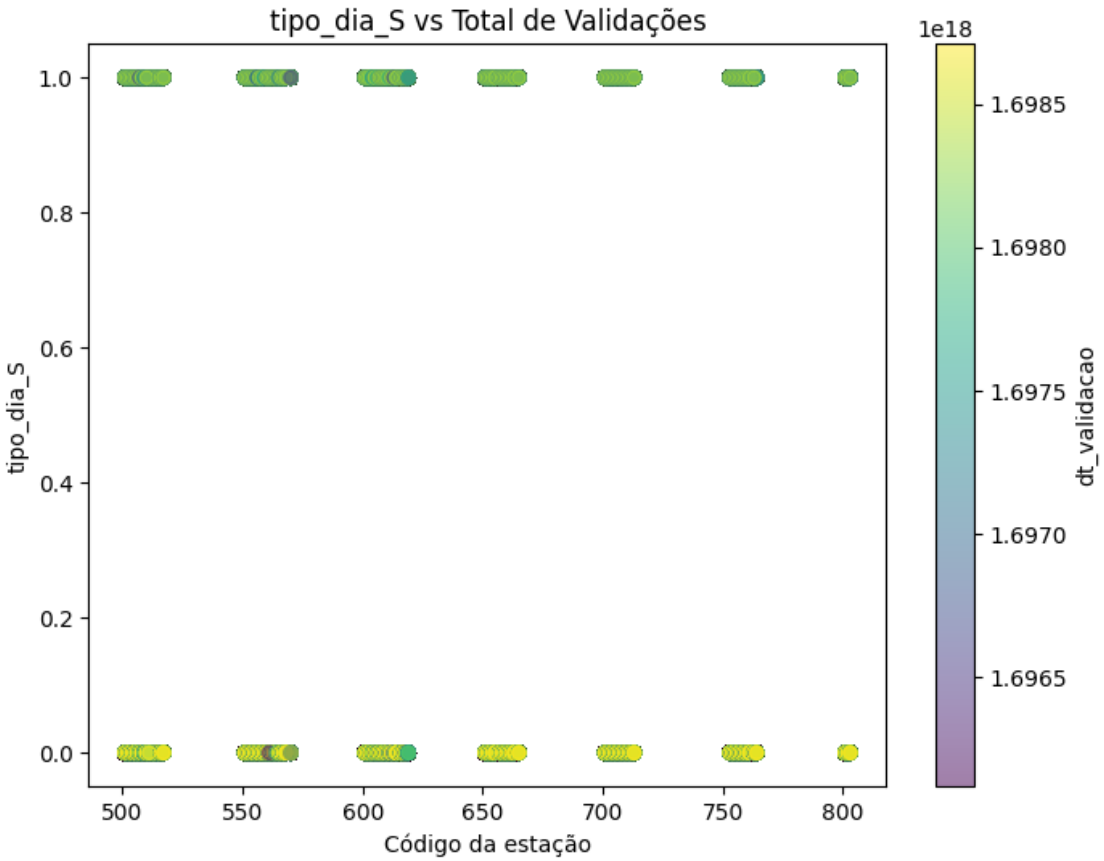
[23]:
```
show_pca = True
```

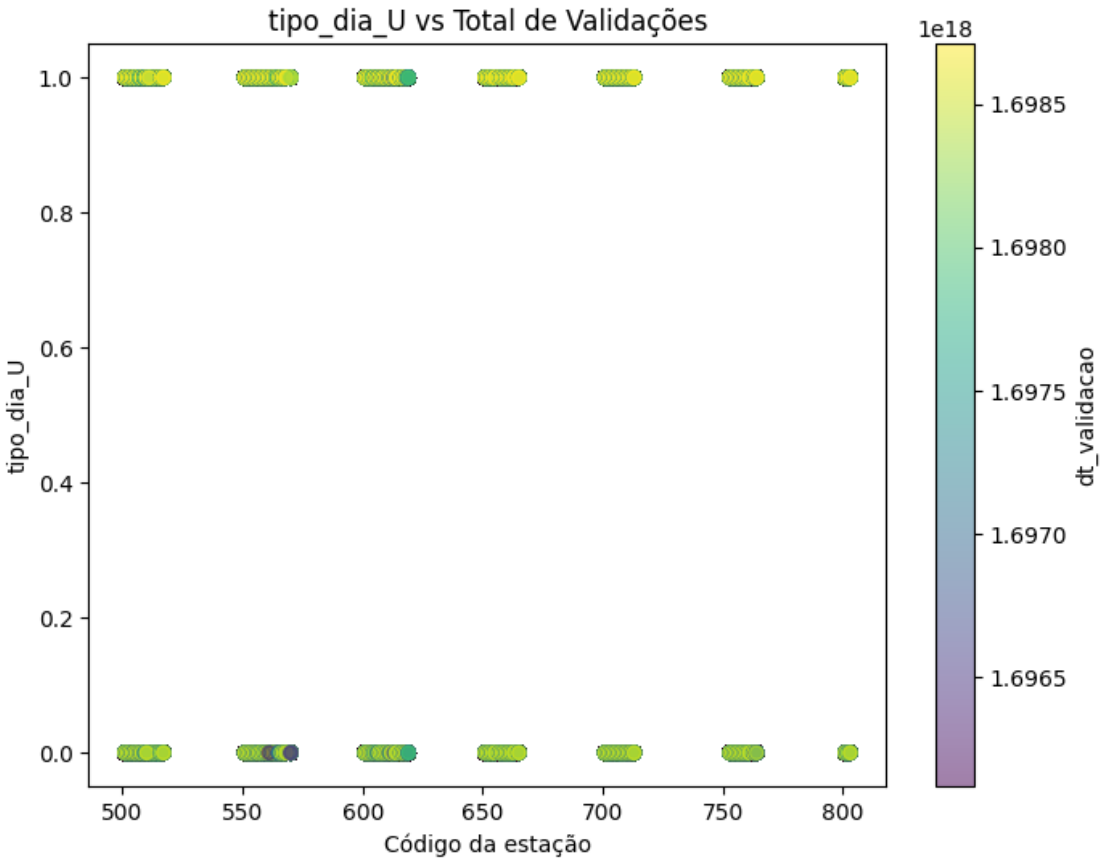
[24]:
```
import matplotlib.pyplot as plt

# Lista das variáveis que você quer comparar com total_validacoes
variaveis = ['id_dt_hora_minuto', 'cod_bilh', 'tipo_dia_D', 'tipo_dia_S',␣
 ↪'tipo_dia_U', 'total_validacoes']

# Loop para criar um gráfico de dispersão para cada variável em relação a␣
 ↪total_validacoes
if show_pca:
    for var in variaveis:
        plt.figure(figsize=(8, 6))  # Tamanho do gráfico
        plt.scatter(mov_no_outliers['cd_estac_bu'], mov_no_outliers[var],␣
 ↪c=mov_no_outliers['dt_validacao'], alpha=0.5, cmap='viridis')
        plt.title(f'{var} vs Total de Validações')
        plt.xlabel('Código da estação')
        plt.ylabel(var)
        plt.colorbar(label='dt_validacao')
        plt.show()
```

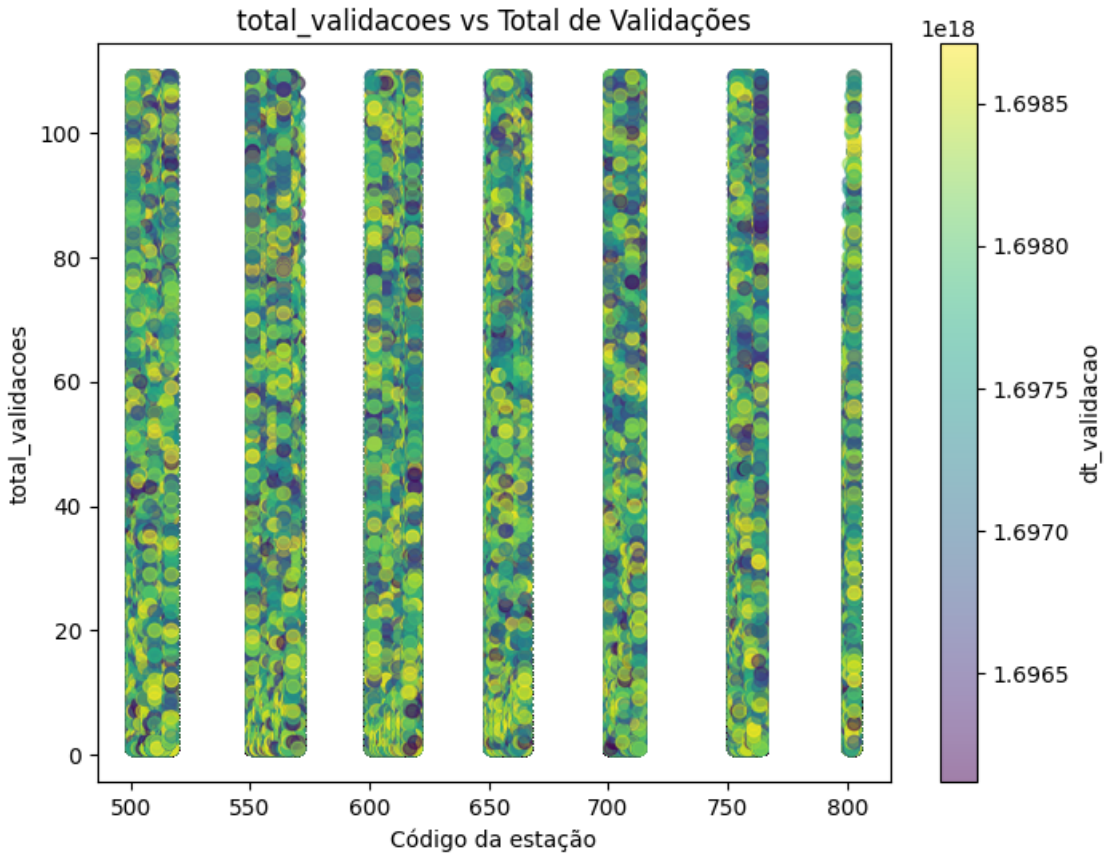

id_dt_hora_minuto vs Total de Validações
id_dt_hora_minuto
Código da estação
dt_validacao
1e18
100
80
60
40
20
0
500
550
600
650
700
750
800
1.6985
1.6980
1.6975
1.6970
1.6965


cod_bilh vs Total de Validações
1e18
1.6985
1.6980
1.6975
1.6970
1.6965
dt_validacao
100000
80000
60000
40000
20000
0
cod_bilh
500
550
600
650
700
750
800
Código da estação


tipo_dia_D vs Total de Validações
tipo_dia_D
Código da estação
dt_validacao
1e18
1.6985
1.6980
1.6975
1.6970
1.6965
1.0
0.8
0.6
0.4
0.2
0.0
500
550
600
650
700
750
800


tipo_dia_S vs Total de Validações
tipo_dia_S
Código da estação
dt_validacao
1e18
1.6985
1.6980
1.6975
1.6970
1.6965
1.0
0.8
0.6
0.4
0.2
0.0
500
550
600
650
700
750
800


tipo_dia_U vs Total de Validações
tipo_dia_U
Código da estação
dt_validacao
1e18
1.6985
1.6980
1.6975
1.6970
1.6965
1.0
0.8
0.6
0.4
0.2
0.0
500
550
600
650
700
750
800



Não é possível organizar os dados em grupos por meio do PCA. Devido a isso será utilizado o KMeans para tentar clusterizar

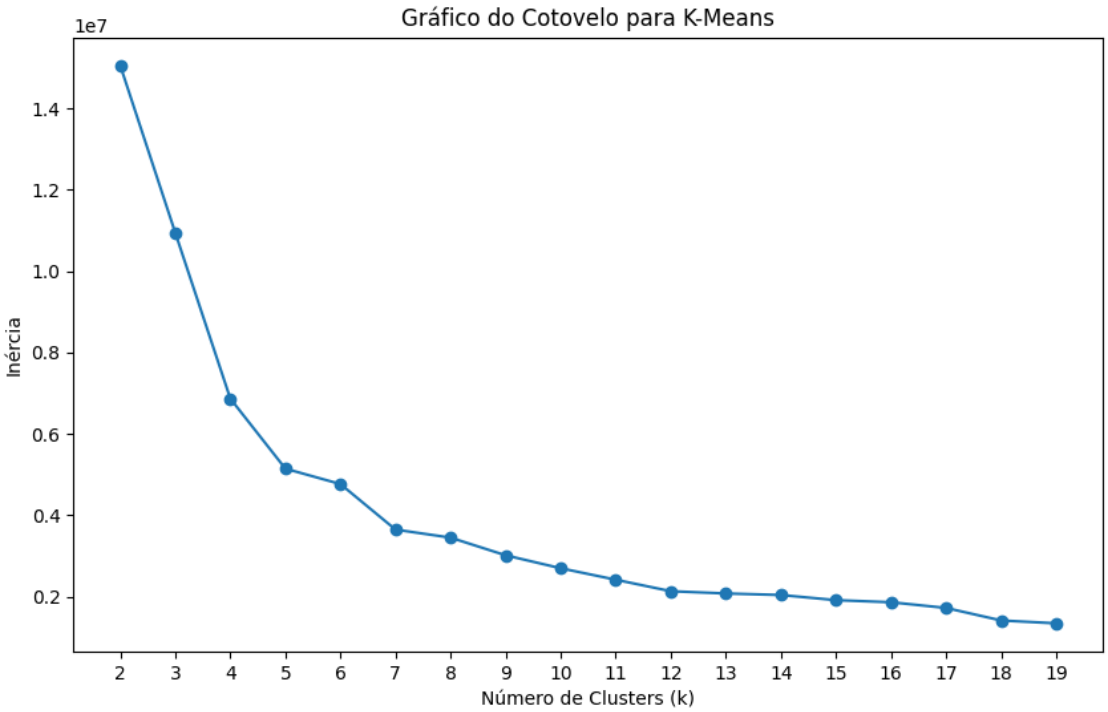
1.4.3 KMeans

Antes de definir os clusters e centroids, iremos verificar o número de clusters

[25]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Seu DataFrame e normalização
df_filtered = mov_no_outliers.drop(columns=['id_dt_hora_minuto',␣
 ↪'dt_validacao'])

df_filtered['tipo_dia_D'] = df_filtered['tipo_dia_D'].astype(int)
df_filtered['tipo_dia_S'] = df_filtered['tipo_dia_S'].astype(int)
df_filtered['tipo_dia_U'] = df_filtered['tipo_dia_U'].astype(int)
```

```
scaler = StandardScaler()
normalized_data = scaler.fit_transform(df_filtered)

K = range(2, 20)
inertia = []

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(normalized_data)
    inertia.append(kmeans.inertia_)

# Plotar o gráfico do cotovelo
plt.figure(figsize=(10, 6))
plt.plot(K, inertia, marker='o')
plt.title('Gráfico do Cotovelo para K-Means')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Inércia')
plt.xticks(K)
plt.show()
```

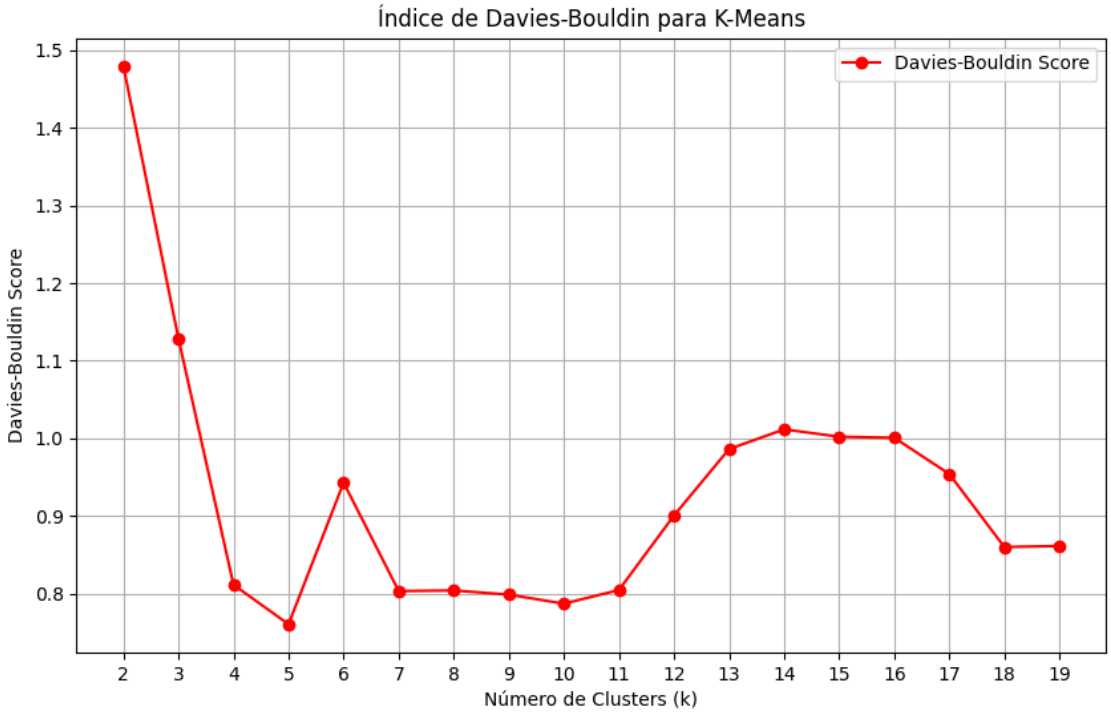
```
c:\Users\Inteli\AppData\Local\pypoetry\Cache\virtualenvs\src-EEZguaxu-
py3.12\Lib\site-packages\joblib\externals\loky\backend\context.py:136:
UserWarning: Could not find the number of physical cores for the following
reason:
found 0 physical cores < 1
Returning the number of logical cores instead. You can silence this warning by
setting LOKY_MAX_CPU_COUNT to the number of cores you want to use.
  warnings.warn(
  File "c:\Users\Inteli\AppData\Local\pypoetry\Cache\virtualenvs\src-EEZguaxu-
py3.12\Lib\site-packages\joblib\externals\loky\backend\context.py", line 282, in
_count_physical_cores
    raise ValueError(f"found {cpu_count_physical} physical cores < 1")
```



Além do elbow graph, iremos utilizar o davies_bouldin_score para identificar a concentração interna dos clusters e a dispersão entre os clusters. Quanto menor o valor do índice, melhor.

[26]:
```
from sklearn.metrics import davies_bouldin_score

K = range(2, 20)
davies_bouldin_scores = []

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(normalized_data)
    db_score = davies_bouldin_score(normalized_data, kmeans.labels_)
    davies_bouldin_scores.append(db_score)

# Plotar o Índice de Davies-Bouldin
plt.figure(figsize=(10, 6))
plt.plot(K, davies_bouldin_scores, marker='o', color='red',␣
 ↪label='Davies-Bouldin Score')
plt.title('Índice de Davies-Bouldin para K-Means')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Davies-Bouldin Score')
plt.xticks(K)
plt.legend()
plt.grid()
```

```
plt.show()
```



Dado o resultado anterior, o melhor número de clusters é 5. Portanto, esse modelo será salvo.

[27]:
```
import joblib
kmeans = KMeans(n_clusters=5, random_state=42)
df_filtered['cluster'] = kmeans.fit_predict(normalized_data)
joblib.dump(kmeans, 'kmeans_model.pkl')
```

[27]:
```
['kmeans_model.pkl']
```

Após isso, será possível importar esse modelo sempre que necessário

[28]:
```
import joblib
kmeans_loaded = joblib.load('kmeans_model.pkl')
```

Visualização dos clusters em Gráfico Para ser possível visualizar a distribuição dos gráficos será realizado um PCA para identificar as principais variavéis que impactaram o knn. Além disso, o gráfico será disponibilizado em plotly para ser interativo e conter 3 variáveis ao invés de 3

PCA

[29]:
```
features = df_filtered[['cod_bilh', 'cd_estac_bu', 'total_validacoes',␣
↪'tipo_dia_D', 'tipo_dia_S', 'tipo_dia_U']]
pca = PCA(n_components=3)
```

```
pca_result = pca.fit_transform(features)

# Adicionar resultados da PCA ao DataFrame
df_filtered['PC1'] = pca_result[:, 0]
df_filtered['PC2'] = pca_result[:, 1]
df_filtered['PC3'] = pca_result[:, 2]

# Preparar centróides para PCA
centroids = kmeans_loaded.cluster_centers_
centroids_pca = pca.transform(centroids)

# Adicionar os centróides ao DataFrame para uso no gráfico
centroids_df = pd.DataFrame(centroids_pca, columns=['PC1', 'PC2', 'PC3'])
centroids_df['cluster'] = range(len(centroids_df))  # Atribuir um número de␣
 ↪cluster para cada centróide
```

```
c:\Users\Inteli\AppData\Local\pypoetry\Cache\virtualenvs\src-EEZguaxu-
py3.12\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid
feature names, but PCA was fitted with feature names
  warnings.warn(
```

Gráfico

[30]:
```
import plotly.express as px

amostra = df_filtered.sample(frac=0.01)


fig = px.scatter_3d(amostra,
                    x='PC1',
                    y='PC2',
                    z='PC3',
                    color='cluster',
                    title='KMeans Clustering 3D (5 Clusters)',
                    labels={'cluster': 'Cluster'},
                    color_continuous_scale=px.colors.qualitative.Set1)

# Adicionar centróides ao gráfico 3D
fig.add_scatter3d(x=centroids_df['PC1'],
                  y=centroids_df['PC2'],
                  z=centroids_df['PC3'],
                  mode='markers',
                  marker=dict(color='black', size=10, symbol='x'),
                  name='Centroids')

# Mostrar gráfico 3D
fig.show()
```

1.5 Exportação para parquet

[31]:
```
mov_no_outliers.to_parquet('./datalake/dmo_anl_vw_tot_mov_periodo.parquet')
```

1.6 Conclusão

Por falta do dicionário de dados, não é possível ter certeza sobre o que exatamente os dados significam. Entretanto, é possível deduzir que esses dados tem relação com o output dos sistemas que entram em contato com o bilhete. Dessa forma, essa tabela se demonstra útil para o grupo Quartzo, uma vez que se relaciona com o fluxo de pessoas nos serviços da CPTM.