

PROJETO CLUSTERING

Vitor Modesto - vml2

Daniel Reinaux - drgf2

Douglas carvalho- dcg

Italo Felipe - ifa2

INTRODUÇÃO

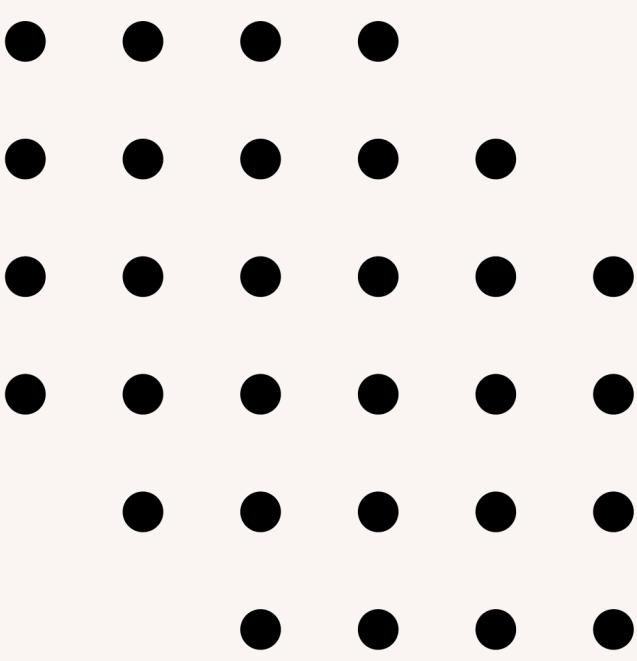
- A base de dados escolhida foi a: **ObesityDataSet_raw_and_data_synthetic**
- Essa base de dados combina dados reais e sintéticos sobre obesidade, relacionados a hábitos alimentares e condição física de um grupo de indivíduos
- Os dados são usados para determinar fatores que podem influenciar a obesidade, como frequência de atividades físicas, consumo de alimentos e etc

INTRODUÇÃO

- O clustering no dataset sobre obesidade desvenda padrões específicos de hábitos e comportamentos. Através desta segmentação, intervenções de saúde tornam-se mais focadas, visando os desafios exclusivos de cada grupo.
- Em um clustering, é possível relacionada a esse dataset identificar padrões ocultos, segmentar a população para compreender ela de uma forma melhor, descoberta de fatores de risco, melhoria da comunicação pública e etc

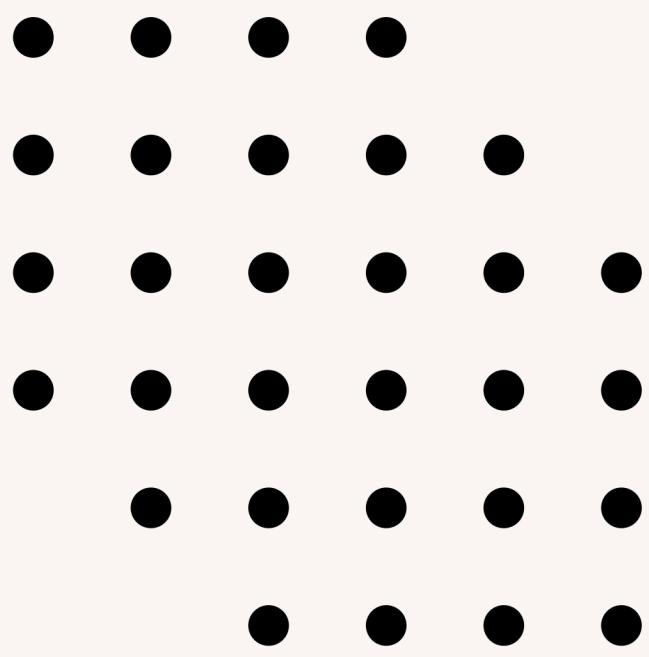
FUNDAMENTOS

- Primeiro, gostaríamos de apresentar um pouco os parâmetros utilizados para se construir a qualidade do nosso modelo:
 1. **Coeficiente de Silheuta:** Mede a similaridade de um objeto com seu próprio cluster em comparação com outros clusters. Os valores variam entre -1 e 1, onde um valor alto indica uma boa clusterização.
 2. **Pureza:** Avalia a uniformidade dos clusters originais com os clusters formados pelo modelo. Uma pureza mais alta indica que um cluster contém predominantemente pontos de uma única classe original.



FUNDAMENTOS

- Ademais, vamos apresentar os algoritmos utilizados nesse estudo:
 1. **K-means:** Um algoritmo de clustering que divide os dados em “k” clusters. Designa centróides aleatórios e regrupa os pontos com base na proximidade desses centróides.
 2. **Fuzzy Means(C-means):** Diferente do K-means, permite que um ponto pertença a vários clusters com diferentes graus. Captura nuances em dados onde as fronteiras dos clusters não são claramente definidas
 3. **DBSCAN:** Baseado na densidade dos dados, agrupa pontos próximos e considera regiões de baixa densidade como ruído. Destaca-se por identificar clusters de formas irregulares.



METODOLOGIA

- 1 Pré-processamento de Dados
- 2 Escolha do numero de clusters
- 3 Treinando o base model usando kmeans
- 4 Aplicando melhorias ao base model
- 5 Fazendo o mesmo processo para DBSCAN e C-means

• • • •

1) PRÉ PROCESSAMENTO

base de dados “Crua”:

```
[167] df = pd.read_csv('/content/drive/MyDrive/Sistemas Inteligentes/ObesityDataSet_raw_and_data_sinthetic.csv')
      df.sample(5)
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH20	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad	
360	Male	20.000000	1.870000	75.000000		no	yes	2.000000	3.0	Frequently	no	1.000000	no	1.000000	1.000000	Sometimes	Public_Transportation	Normal_Weight
458	Male	19.000000	1.690000	60.000000		no	yes	2.000000	3.0	Always	no	1.000000	no	1.000000	1.000000	Sometimes	Public_Transportation	Normal_Weight
2027	Female	18.206340	1.807406	141.799429		yes	yes	3.000000	3.0	Sometimes	no	2.472903	no	1.998047	0.840911	Sometimes	Public_Transportation	Obesity_Type_III
378	Male	18.000000	1.730000	70.000000		no	yes	3.000000	3.0	Frequently	no	1.000000	no	2.000000	1.000000	Sometimes	Public_Transportation	Normal_Weight
1324	Female	22.480889	1.605662	82.470375		yes	yes	1.557287	1.0	Sometimes	no	2.371015	no	0.288032	2.000000	Sometimes	Public_Transportation	Obesity_Type_I

```
[169] df.shape
(2111, 17)
```

• • • •

1) PRÉ PROCESSAMENTO

Verificando a existência de valores nulos :

```
0s   df.isnull().sum()

Gender          0
Age             0
Height          0
Weight           0
family_history_with_overweight  0
FAVC            0
FCVC            0
NCP             0
CAEC            0
SMOKE           0
CH20            0
SCC              0
FAF              0
TUE              0
CALC             0
MTRANS           0
NObeyesdad      0
dtype: int64
```

-> não existem valores nulos nessa base de dados

1) PRÉ PROCESSAMENTO

Convertendo as colunas categóricas para numéricas através do OrdinalEncoder:

```
[170] categoricas = ['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE', 'SCC', 'CALC', 'MTRANS']
encoder = OrdinalEncoder()
for cat in categoricas:
    df[cat] = encoder.fit_transform(df[[cat]])
df.sample(6)
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH20	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad	
1492	1.0	18.000000	1.782722	108.044313		1.0	1.0	2.000000	2.655265	2.0	0.0	2.297896	0.0	1.000000	0.552665	3.0	3.0	Obesity_Type_I
1394	1.0	22.815416	1.732694	98.441130		1.0	1.0	2.000000	2.993623	2.0	0.0	2.326635	0.0	2.236586	1.529423	3.0	3.0	Obesity_Type_I
339	0.0	19.000000	1.530000	42.000000		0.0	0.0	2.000000	3.000000	2.0	0.0	1.000000	1.0	2.000000	0.000000	1.0	3.0	Insufficient_Weight
712	0.0	19.054938	1.585886	42.541794		0.0	0.0	2.910345	3.000000	1.0	0.0	1.000000	1.0	1.461005	0.000000	2.0	3.0	Insufficient_Weight
1328	1.0	30.967417	1.688436	90.000000		1.0	1.0	2.180047	2.733077	2.0	0.0	2.779620	0.0	1.454129	0.000000	2.0	0.0	Obesity_Type_I
901	1.0	19.241058	1.856811	88.633616		1.0	1.0	2.047069	3.829101	2.0	0.0	1.641022	0.0	1.554817	0.248218	2.0	3.0	Overweight_Level_I

Foi escolhido o Ordinal Encoder para essa base de dados pois as suas colunas categóricas não tinham muitos valores diferentes, logo, poderíamos facilmente substituir as atribuições por um valor numérico

• • • •

1) PRÉ PROCESSAMENTO

Separando as features do target(label) e dividindo a base de dados entre treino e teste:

```
Separando as features do target
[171] X = df.iloc[:, 0:16]
      y = df.iloc[:, -1]

Dividindo a base de dados entre treino e teste
[172] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

Usamos um test_size baixo pois o dataset tem poucas instâncias, então o treinamento do modelo poderia ser prejudicado com um test_size maior

```
[260] X_train.shape, X_test.shape
((1899, 16), (212, 16))
```

1) PRÉ PROCESSAMENTO

No caso dessa base de dados, testamos treinar o base model com e sem escalonamento e percebemos que tivemos resultados(tanto para pureza quanto pra silhueta) bem melhores sem o uso do escalonamento

Escalonando os dados X sem Escalonar os dados

```
[251] silhouette_base_model = metrics.silhouette_score(X_test_scaled, cluster_labels)
      print ('silhouette coefficient for the above clustering = ', silhouette_base_model)

      silhouette coefficient for the above clustering =  0.1487559525572936

[252] # Medindo a pureza
      def purity_score(y_true, y_pred):
          # compute contingency matrix (also called confusion matrix)
          contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
          return np.sum(npamax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)

      purity_base = purity_score(y_test, cluster_labels)
      print ('Purity for the above clustering = ', purity_base)

      Purity for the above clustering =  0.3018867924528302
```

```
[265] silhouette_base_model = metrics.silhouette_score(X_test, cluster_labels)
      print ('silhouette coefficient for the above clustering = ', silhouette_base_model)

      silhouette coefficient for the above clustering =  0.5005730477798578

[267] # Medindo a pureza
      def purity_score(y_true, y_pred):
          # compute contingency matrix (also called confusion matrix)
          contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
          return np.sum(npamax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)

      purity_base = purity_score(y_test, cluster_labels)
      print ('Purity for the above clustering = ', purity_base)

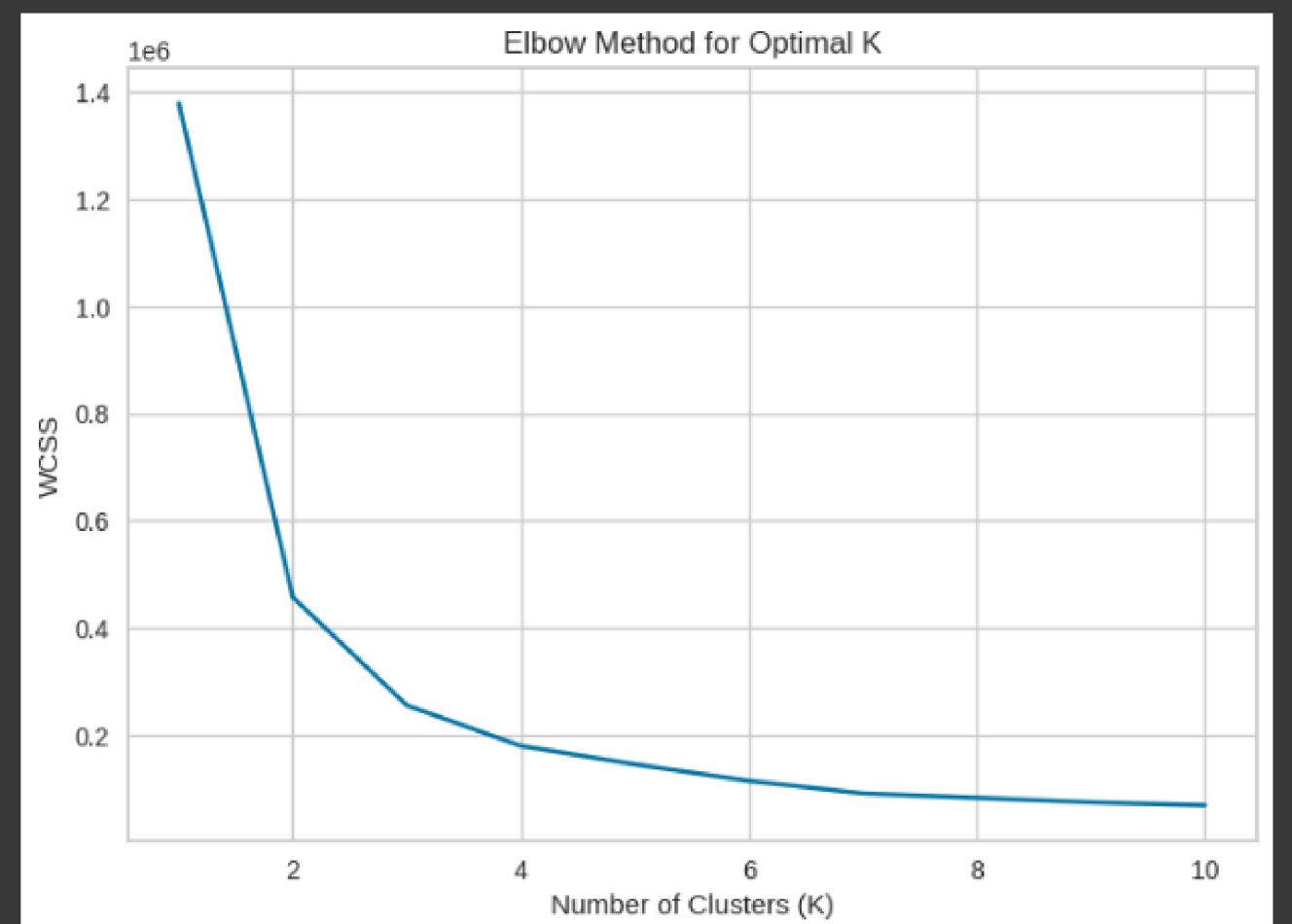
      Purity for the above clustering =  0.42924528301886794
```

2) ESCOLHA DO NUMERO DE CLUSTERS

```
[261] wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X_train)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.show()
```



Com base no método do cotovelo, percebemos que o numero de clusters teria que ser 2 ou 3, então, fizemos os testes para aferir qual se saía melhor

2) ESCOLHA DO NUMERO DE CLUSTERS

2 Clusters X 3 Clusters

```
[269] silhouette_base_model = metrics.silhouette_score(X_test, cluster_labels)
      print ('silhouette coefficient for the above clustering = ', silhouette_base_model)

silhouette coefficient for the above clustering =  0.5969633913465453

[270] # Medindo a pureza
      def purity_score(y_true, y_pred):
          # compute contingency matrix (also called confusion matrix)
          contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
          return np.sum(npamax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)

      purity_base = purity_score(y_test, cluster_labels)
      print ('Purity for the above clustering = ', purity_base)

Purity for the above clustering =  0.330188679245283
```

```
[271] silhouette_base_model = metrics.silhouette_score(X_test, cluster_labels)
      print ('silhouette coefficient for the above clustering = ', silhouette_base_model)

silhouette coefficient for the above clustering =  0.5005730477798578

[272] # Medindo a pureza
      def purity_score(y_true, y_pred):
          # compute contingency matrix (also called confusion matrix)
          contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
          return np.sum(npamax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)

      purity_base = purity_score(y_test, cluster_labels)
      print ('Purity for the above clustering = ', purity_base)

Purity for the above clustering =  0.42924528301886794
```

Durante a análise do número de clusters, percebemos que, apesar do coeficiente de silhouette ser maior para 2 clusters, o nível de pureza cai consideravelmente usando esse número de clusters, logo, percebemos que isso afetaria a coêrecia do modelo, então optamos por usar 3 Clusters, que tem um silhouette menor, mas uma pureza consideravelmente maior

• • • 3) TREINANDO O BASE MODEL USANDO KMEANS

```
[271] kmeans = KMeans(n_clusters=3)
      kmeans.fit(X_train)
      cluster_labels = kmeans.predict(X_test)

      kmeans.cluster_centers_
```

Treinamos o modelo utilizando o X_train e pegamos os cluster labels para o X_test

```
[272] silhouette_base_model = metrics.silhouette_score(X_test, cluster_labels)
      print ('silhouette coefficient for the above clustering = ', silhouette_base_model)

      silhouette coefficient for the above clustering =  0.5005730477798578

[273] # Medindo a pureza
      def purity_score(y_true, y_pred):
          # compute contingency matrix (also called confusion matrix)
          contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
          return np.sum(np.amax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)

      purity_base = purity_score(y_test, cluster_labels)
      print ('Purity for the above clustering = ', purity_base)

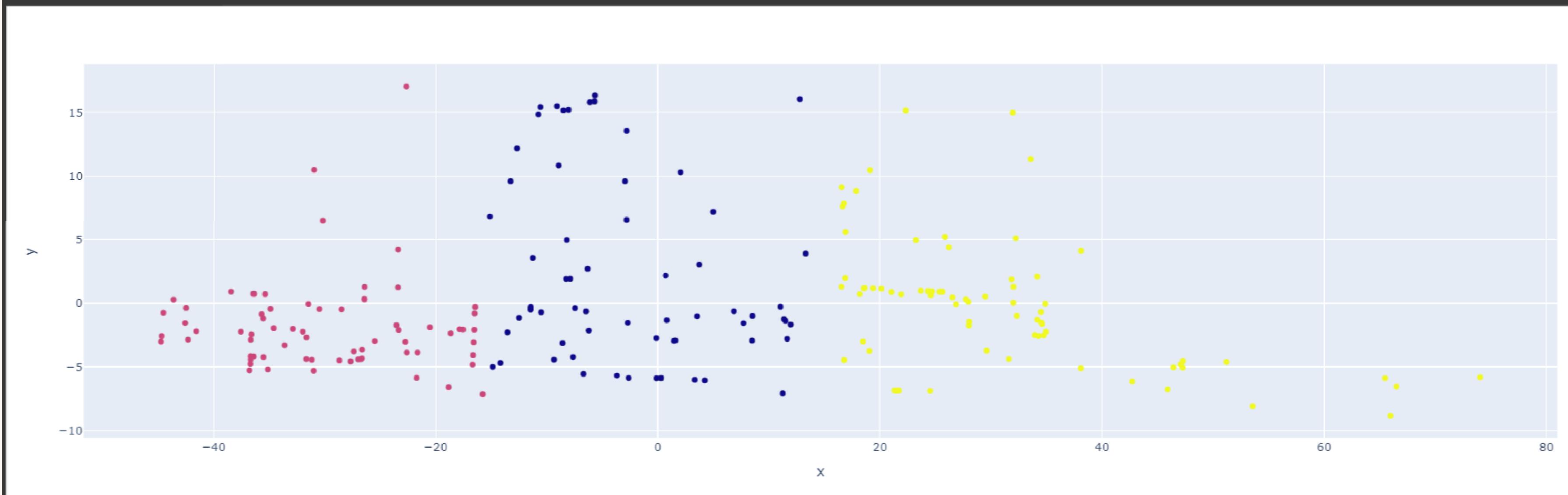
      Purity for the above clustering =  0.42924528301886794
```

Resultados iniciais para o o base model

3) TREINANDO O BASE MODEL USANDO KMEANS

Reduzimos a dimensionalidade da base de dados para plotar o gráfico do resultado do base model:

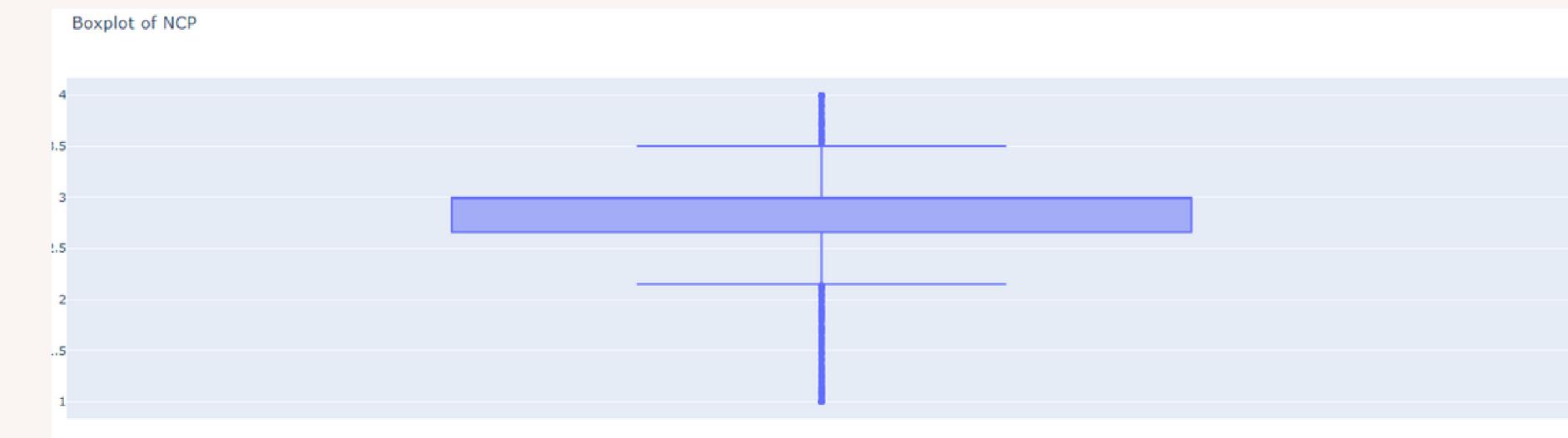
```
pca = PCA(n_components=2) # Queremos reduzir a dimensionalidade para 2  
df_2d = pca.fit_transform(X_test)  
grafico = px.scatter(x = df_2d[:, 0], y = df_2d[:, 1], color = cluster_labels)  
grafico.show()
```



• • • • 4) APLICANDO MELHORIAS AO BASE MODEL

3.1) Removendo outliers

A partir de gráficos de BOXPLOT, 2 deles chamaram a atenção pelos seus outliers:



No primeiro gráfico, percebemos que tinham outliers com uma distância considerável da media dos valores, enquanto no segundo foi notado que essa coluna tinha uma grande composição de valores outliers

• • • •

4) APLICANDO MELHORIAS AO BASE MODEL

3.1) Removendo outliers

Com base nos outliers mostrados no slide passado, decidimos por filtrar a coluna de idade, escolhendo apenas valores acima de 55, tirando os outliers mais distantes da média. Enquanto para a coluna NCP, optamos por tira-la da base de dados, uma vez que boa parte das suas instâncias eram outliers

```
[274] X_train_NoNCP = X_train.drop(columns=['NCP'])
X_test_NoNCP = X_test.drop(columns=['NCP'])

kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train_NoNCP)

cluster_labels = kmeans.predict(X_test_NoNCP)
silhouette_avg = metrics.silhouette_score(X_test_NoNCP, cluster_labels)
print('Melhoria comparativamente da silhouette com o base model: ', silhouette_avg - silhouette_base_model)
print('Melhoria comparativamente da purity com o base model:', purity_score(y_test, cluster_labels) - purity_base)
```

```
[275] X_train_FilterAge = X_train[X_train['Age'] < 55]
X_test_FilterAge = X_test[X_test['Age'] < 55]

# Aplicando o mesmo filtro aos DataFrames Y_train e Y_test
y_train_FilterAge = y_train.loc[X_train_FilterAge.index]
y_test_FilterAge = y_test.loc[X_test_FilterAge.index]

kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train_FilterAge)

cluster_labels = kmeans.predict(X_test_FilterAge)
silhouette_avg = metrics.silhouette_score(X_test_FilterAge, cluster_labels)
print ('Melhoria comparativamente com o base model: ', silhouette_avg - silhouette_base_model)
print('Melhoria comparativamente da purity com o base model:', purity_score(y_test_FilterAge, cluster_labels) - purity_base)
```

• • • •

4) APLICANDO MELHORIAS AO BASE MODEL

4 .1) Removendo outliers

É importante deixar claro que para testar as melhorias no modelo com a retirada dos outliers, criamos um X_teste para cada uma delas, logo, para confirmar a atualização, temos que alterar o dataframe principal, tanto o de treino: X_train, y_train, quanto de teste: X_test, y_test

```
Aplicando as mudanças calcular no DatataFrame Principal

[185] X_train = X_train.drop(columns=['NCP'])
      X_test = X_test.drop(columns=['NCP'])

      X_train = X_train[X_train['Age'] < 55]
      X_test = X_test[X_test['Age'] < 55]

      # Aplicando o mesmo filtro aos DataFrames Y_train e Y_test
      y_train = y_train.loc[X_train.index]
      y_test = y_test.loc[X_test.index]

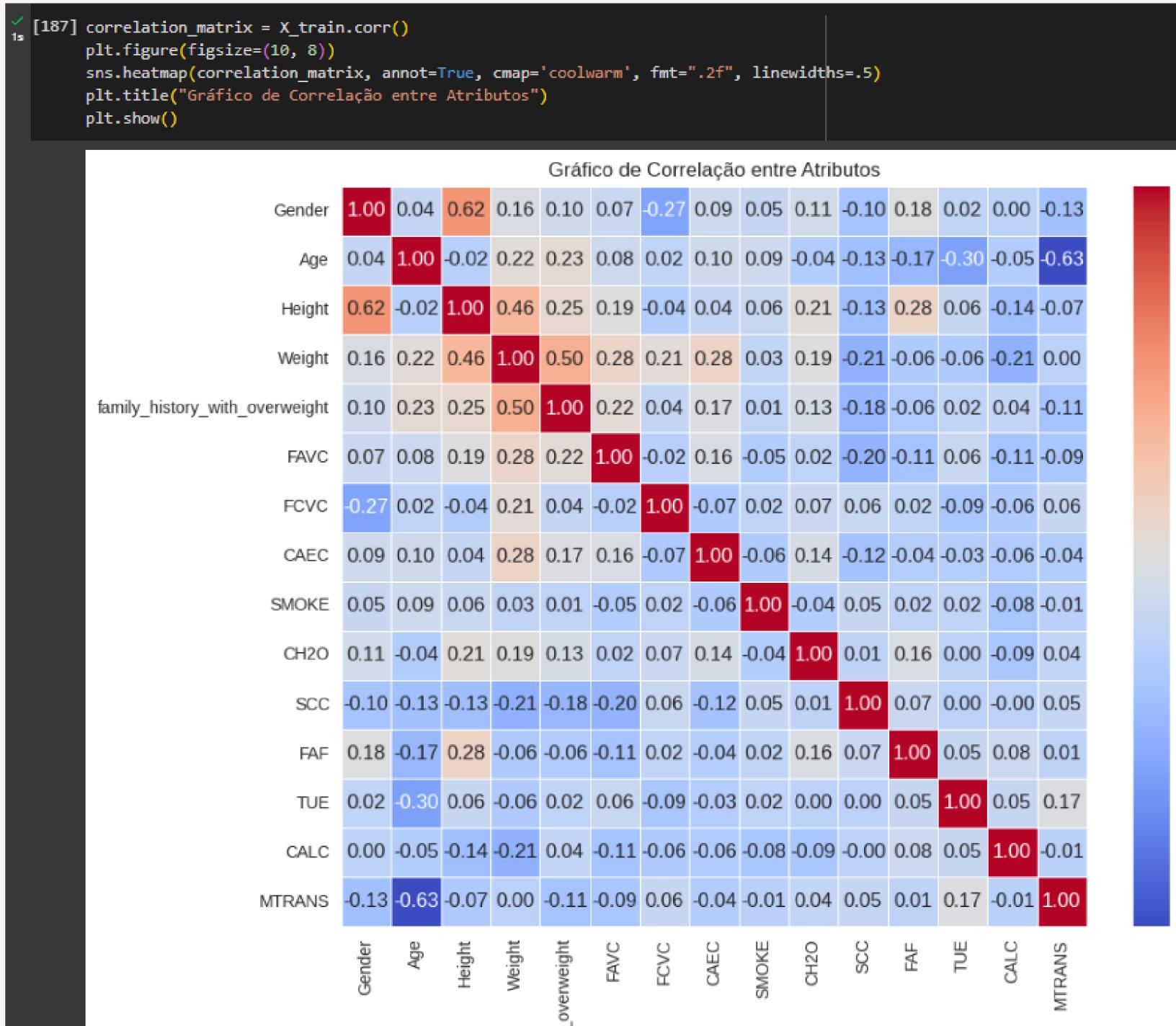
      kmeans = KMeans(n_clusters=3)
      kmeans.fit(X_train)

      cluster_labels = kmeans.predict(X_test)
      silhouette_base_model = metrics.silhouette_score(X_test, cluster_labels)
      print ('silhouette coefficient for the above clutering = ', silhouette_base_model)
      purity_base = purity_score(y_test, cluster_labels)
      print ('Purity for the above clutering = ', purity_base)
```

Como houve uma remoção de algumas linhas na retirada dos outliers de “Age”, temos que fazer mudanças no y_train e y_test, usando o .index para saber exatamente o que sobrou em X após a exclusão de algumas instancias

4) APLICANDO MELHORIAS AO BASE MODEL

4.2) Vendo correlações entre as features



Valor de ρ (+ ou -)	Interpretação
0.00 a 0.19	Uma correlação bem fraca
0.20 a 0.39	Uma correlação fraca
0.40 a 0.69	Uma correlação moderada
0.70 a 0.89	Uma correlação forte
0.90 a 1.00	Uma correlação muito forte

Fizemos esse gráfico para verificar se existem features com uma correlação muito forte entre elas, uma vez que ter duas variáveis com forte correlação pode prejudicar a eficiência do modelo. Ao analisar o gráfico das correções, duas correlações chamaram a atenção do grupo, sendo elas: Gender X Height e Age X MTRANS, vamos analisar as mudanças nas métricas usadas nesse projeto ao excluir cada uma das features citadas acima

• • • 4) APLICANDO MELHORIAS AO BASE MODEL

4.2) Vendo correlações entre as features:Gender X Height

Excluindo a coluna “Gender” X Excluindo a coluna “Height”

```
[188] # Vamos excluir o genero, escalar os dados e testar o coeficiente de silhouette
X_train_NoGender = X_train.drop(columns=['Gender'])
X_test_NoGender = X_test.drop(columns=['Gender'])

# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X_NoGender)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train_NoGender)

cluster_labels = kmeans.predict(X_test_NoGender)
silhouette_avg = metrics.silhouette_score(X_test_NoGender, cluster_labels)
print ('Melhoria comparativamente com o base model: ', silhouette_avg - silhouette_base_model)
print('Melhoria comparativamente da purity com o base model:', purity_score(y_test, cluster_labels) - purity_base)

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
Melhoria comparativamente com o base model:  0.0006531059209590184
Melhoria comparativamente da purity com o base model: 0.0
```

```
# Vamos excluir a altura, escalar os dados e testar o coeficiente de silhouette
X_train_NoHeight = X_train.drop(columns=['Height'])
X_test_NoHeight = X_test.drop(columns=['Height'])

# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X_NoHeight)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train_NoHeight)

cluster_labels = kmeans.predict(X_test_NoHeight)
silhouette_avg = metrics.silhouette_score(X_test_NoHeight, cluster_labels)
print ('Melhoria comparativamente com o base model: ', silhouette_avg - silhouette_base_model)
print('Melhoria comparativamente da purity com o base model:', purity_score(y_test, cluster_labels) - purity_base)

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
Melhoria comparativamente com o base model:  2.14439525101362e-05
Melhoria comparativamente da purity com o base model: 0.0
```

Com isso, percebemos que houve uma melhora muito pequena em ambas as abordagens, então chegamos a conclusão que não valia a pena tirar uma coluna que facilitava a interpretabilidade do modelo(como gender e height) a troco de uma melhora pouco significativa

• • • • 4) APLICANDO MELHORIAS AO BASE MODEL

4.2) Vendo correlações entre as features: Age X MTRANS

Excluindo a coluna “Age” X Excluindo a coluna “MTRANS”

```
# Vamos excluir o atributo Age, escalar os dados e testar o coeficiente de silhouette
X_train_NoAge = X_train.drop(columns=['Age'])
X_test_NoAge = X_test.drop(columns=['Age'])
# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X_NoAge)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train_NoAge)

cluster_labels = kmeans.predict(X_test_NoAge)
silhouette_avg = metrics.silhouette_score(X_test_NoAge, cluster_labels)
print ('Melhoria comparativamente com o base model: ', silhouette_avg - silhouette_base_model)
print('Melhoria comparativamente da purity com o base model:', purity_score(y_test, cluster_labels) - purity_base)

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
Melhoria comparativamente com o base model:  0.06315151718015721
Melhoria comparativamente da purity com o base model: 0.0
```

```
# Vamos excluir o atributo MTRANS, escalar os dados e testar o coeficiente de silhouette
X_train_NoMtrans = X_train.drop(columns=['MTRANS'])
X_test_NoMtrans = X_test.drop(columns=['MTRANS'])

# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X_NoMtrans)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train_NoMtrans)

cluster_labels = kmeans.predict(X_test_NoMtrans)
silhouette_avg = metrics.silhouette_score(X_test_NoMtrans, cluster_labels)
print ('Melhoria comparativamente com o base model: ', silhouette_avg - silhouette_base_model)
print('Melhoria comparativamente da purity com o base model:', purity_score(y_test, cluster_labels) - purity_base)

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
Melhoria comparativamente com o base model:  0.003174286069079102
Melhoria comparativamente da purity com o base model: 0.0
```

Com isso, percebemos que houve uma melhoria considerável com a exclusão da coluna “Age”, enquanto a exclusão da coluna “MTRANS” teve uma melhoria minúscula, por conta disso, optamos por excluir a coluna “Age” da base de dados

• • • •

4) APLICANDO MELHORIAS AO BASE MODEL

4.2) Vendo correlações entre as features: Age X MTRANS

É importante deixar claro que para testar as melhorias no modelo com a retirada das colunas mostradas, criamos um X_teste para cada uma delas, logo, para confirmar a atualização, temos que alterar o dataframe principal, tanto o de treino: X_train, quanto de teste: X_test

Subindo as alterações pra base de dados principal

```
[194] X_train = X_train.drop(columns=['Age'])
      X_test = X_test.drop(columns=['Age'])

      kmeans = KMeans(n_clusters=3)
      kmeans.fit(X_train)

      cluster_labels = kmeans.predict(X_test)
      silhouette_base_model = metrics.silhouette_score(X_test, cluster_labels)
      print ('silhouette coefficient for the above clustering = ', silhouette_base_model)
      print('Melhoria comparativamente da purity com o base model:', purity_score(y_test, cluster_labels) - purity_base)
```

• • • •

4) APLICANDO MELHORIAS AO BASE MODEL

4.2) Otimizando os parâmetros do KMEANS

Criamos a função optimize que utiliza GridSearchCV para fazer a validação cruzada, e a partir dela, vamos conferir quais são os melhores parâmetros para a nossa base de dados

```
[206] def optimize(n_clusters, init, n_init, max_iter, tol, algorithm, params, cv = 5):
    np.random.seed(0)
    Kms = KMeans(n_clusters=n_clusters, init=init, n_init=n_init, max_iter=max_iter, tol=tol, algorithm=algorithm, random_state = 0)
    # Usaremos GridSearchCV: Técnica que usa validação cruzada para fazer a combinação e teste com vários parâmetros e vai retornar o melhor deles
    grid_search = GridSearchCV(estimator = Kms, param_grid = params, n_jobs = -1, cv = cv) # Param_grid é a lista de parâmetros que ele irá testar
    grid_search.fit(X) # GridSearch já faz a diferenciação de treino e teste por si só

    melhores_parametros = grid_search.best_params_
    best_kmeans_model = grid_search.best_estimator_

    melhor_resultado = metrics.silhouette_score(X, best_kmeans_model.labels_) # pegando o silhouette_score para o melhor modelo a partir da validação cruzada
    print(f'melhores parâmetros pelo GridSearch: {melhores_parametros}')
    print('silhouette coefficient for the above clustering = ', melhor_resultado)

    return melhores_parametros, melhor_resultado
```

• • • •

4) APLICANDO MELHORIAS AO BASE MODEL

4.2) Otimizando os parâmetros do KMEANS

Exemplo de utilização da função:

```
▼ Otimizando init

✓ [208] init = None
2s     params = {'init': ['k-means++', 'random']} # Possíveis valores pra init
          opt_param, score = optimize(n_clusters=n_clusters,
                                         init=init,
                                         n_init=n_init,
                                         max_iter=max_iter,
                                         tol=tol,
                                         algorithm=algorithm,
                                         params=params)

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1366: FutureWarning:
algorithm='auto' is deprecated, it will be removed in 1.3. Using 'lloyd' instead.

melhores parametros pelo GridSearch: {'init': 'k-means+'}
silhouette coefficient for the above clustering =  0.5005126385469671
```

Aplicamos essa função para cada um dos parâmetros do KMEANS e todos retornaram como melhor parâmetro o default, logo, não foi possível obter uma melhora a partir dessa estratégia, mas foi possível constatar quais são os melhores parâmetros para futuros testes e aplicações

• • • • 5) MESMO PROCESSO PARA DBSCAN E C-MEANS

5.1) DBSCAN

Fazendo o treinamento do base model de DBSCAN

```
[226] dbSCAN = DBSCAN(eps=0.5, min_samples=5)
      # Em DBSCAN, nao temos o .predict(), logo, nao precisarwemos fazer a diferenciação entre X_train e X_test, entao vamos concatenar-los
      X_combined = pd.concat([X_train, X_test], axis=0)
      y_combined = pd.concat([y_train, y_test], axis=0)
      scaler = StandardScaler()
      X_scaled = scaler.fit_transform(X_combined)

      cluster_labels_dbSCAN = dbSCAN.fit_predict(X_scaled)

      # Calcule o silhouette score
      silhouette_avg_dbSCAN = metrics.silhouette_score(X_scaled, cluster_labels_dbSCAN)

      print('Silhouette coefficient for DBSCAN clustering =', silhouette_avg_dbSCAN)
      purity_base = purity_score(y_combined, cluster_labels_dbSCAN)
      print ('Purity for the above clustering = ', purity_base)

Silhouette coefficient for DBSCAN clustering = -0.2015722097448407
Purity for the above clustering =  0.404467680608365
```

• • • • • 5) MESMO PROCESSO PARA DBSCAN E C-MEANS

5.1) DBSCAN

Para otimizar os parâmetros do DBSCAN, fizemos um código que iterando tanto sobre o eps_values quanto tava o min_samples_values, para testar todas as combinações dos valores para esses atributos que queremos testar, armazenando o melhor resultado com base no coeficiente de silhueta

```
[228] # Colocando os parametros que queremos testar
      eps_values = np.arange(0.1, 0.9, 0.1)
      min_samples_values = range(1, 10)

      # Inicializando os melhores parametros que serão retornados como None
      best_eps = None
      best_min_samples = None
      best_score = -1

      for eps in eps_values:
          for min_samples in min_samples_values:
              dbscan = DBSCAN(eps=eps, min_samples=min_samples)
              labels = dbscan.fit_predict(X_scaled)

              # Apenas calculando o Coeficiente de Silhueta para clusterizações com mais de 1 cluster e menos que o número total de pontos
              if len(set(labels)) > 1 and -1 in labels:
                  score = metrics.silhouette_score(X_scaled, labels) # silhouette
                  purity_base = purity_score(y_combined, labels)
                  if score > best_score:
                      best_eps = eps
                      best_min_samples = min_samples
                      best_score = score
                      best_purity = purity_base

      print(f"Melhor eps: {best_eps}")
      print(f"Melhor min_samples: {best_min_samples}")
      print(f"Melhor Coeficiente de Silhueta: {best_score}")
      print(f"Melhor purity: {best_purity}")

      Melhor eps: 0.8
      Melhor min_samples: 2
      Melhor Coeficiente de Silhueta: -0.056276635021174436
      Melhor purity: 0.7105513307984791
```

Com isso, chegamos a conclusão de que os melhores valores para eps e min_samples são 0.8 e 2, respectivamente. Tendo esses dois valores em mente, conseguimos aumentar em certa de 0.15 o coeficiente de silhueta do DBSCAN, o que é uma melhora considerável em comparação ao base model, tendo também uma melhora drástica na pureza, chegando a conclusão que a melhoria foi um sucesso.

• • • • 5) MESMO PROCESSO PARA DBSCAN E C-MEANS

5.1) cMeans

Treinando o base model e verificando o resultado para o cMeans

```
# parametros que serão futuramente otimizado
error = 0.005
max_iter = 1000
init = None

# aplicando o cMeans
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    X_combined.T, ncenters, 2, error=error, maxiter=max_iter, init=init)

# Visualizando os clusters resultantes
cluster_membership = np.argmax(u, axis=0)

# Reduzindo a dimensionalidade para plotar o grafico
pca = PCA(n_components=2)
df_2d = pca.fit_transform(X_combined)

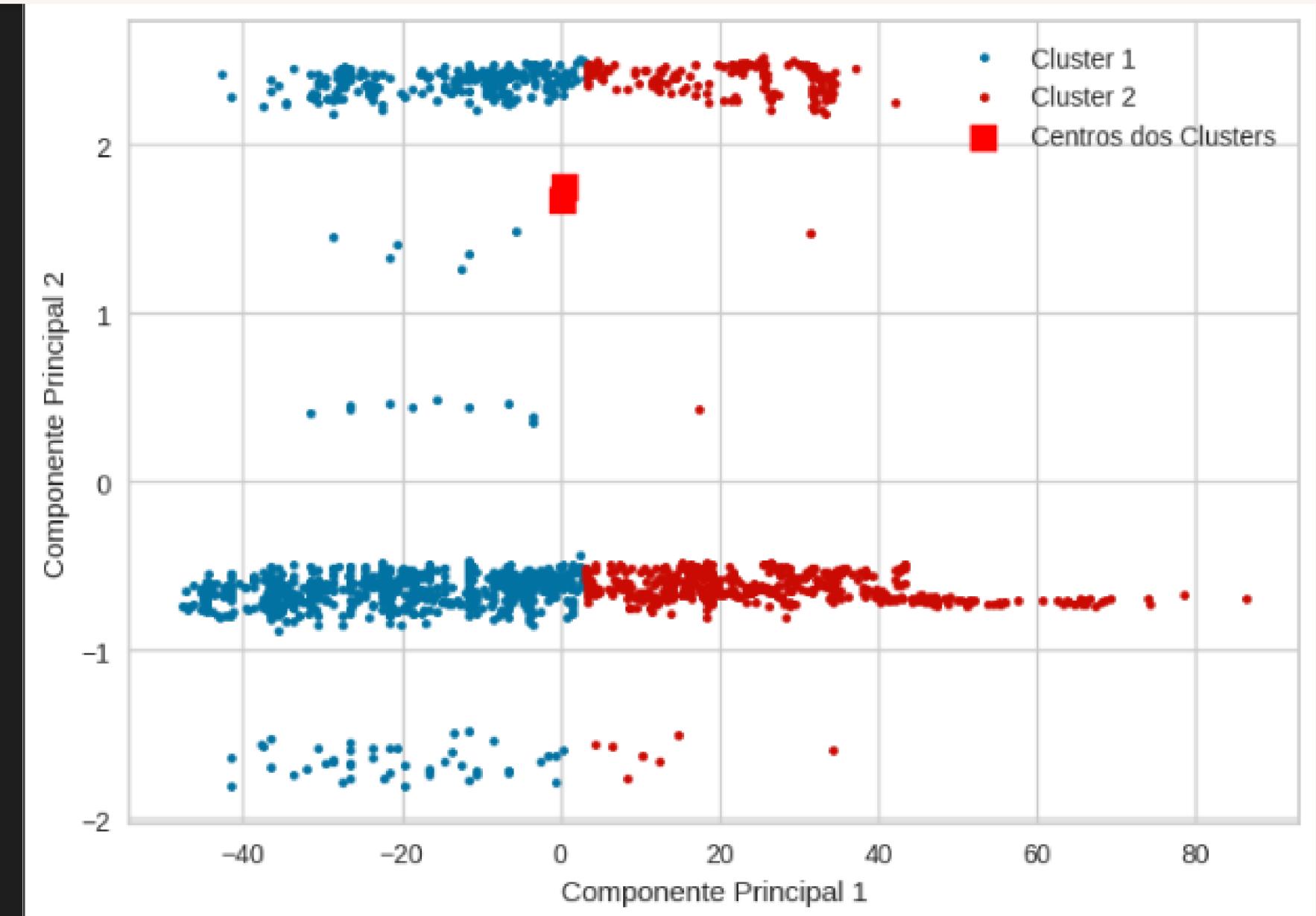
# Plotando os dados
colors = ['b', 'r', 'g', 'c', 'm', 'y', 'k']
for j in range(ncenters):
    plt.plot(df_2d[cluster_membership == j, 0],
              df_2d[cluster_membership == j, 1], '.', color=colors[j], label=f'Cluster {j + 1}')

plt.scatter(cntr[:, 0], cntr[:, 1], marker='s', s=100, c='red', label='Centros dos Clusters')

plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.legend()

plt.show()

print(f'Valor FPC: {fpc:.2f}')
```



5) MESMO PROCESSO PARA DBSCAN E C-MEANS

5.1) cMeans

Resultado do base model para as métricas utilizadas no projeto:

```
# Verificando os resultados
silhouette_avg = metrics.silhouette_score(X_combined, cluster_membership)
print(f'Coeficiente de Silhueta: {silhouette_avg}')
purity_base = purity_score(y_combined, cluster_labels_dbSCAN)
print ('Purity for the above clustering = ', purity_base)
```

• • • • 5) MESMO PROCESSO PARA DBSCAN E C-MEANS

5.1) cMeans

De forma semelhante ao que foi feito no DBSCAN, vamos fazer uma codificação para iterar sobre os valores que queremos testar para os parâmetros do c-means para que assim possamos descobrir quais são os melhores valores para eles na nossa base de dados

```
# parametros que serão futuramente otimizado
ncenters_values = range(2, 11)
error_values = [0.001, 0.005, 0.01]
max_iter_values = [100, 500, 1000]

best_ncenters = None
best_error = None
best_max_iter = None
best_score = -1

# Aplicando a redução de dimensionalidade usando PCA
pca = PCA(n_components=2)
df_2d = pca.fit_transform(X_scaled)

for ncenters in ncenters_values:
    for error in error_values:
        for max_iter in max_iter_values:
            # Aplicando o algoritmo c-Means
            cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
                X_combined.T, ncenters, 2, error=error, maxiter=max_iter, init=None)

            # Calculando a associação de cluster para cada ponto de dados
            cluster_membership = np.argmax(u, axis=0)

            # Calcula o Coeficiente de Silhueta
            silhouette_avg = metrics.silhouette_score(X_combined, cluster_membership)
            purity_base = purity_score(y_combined, labels)
            if silhouette_avg > best_score:
                best_ncenters = ncenters
                best_error = error
                best_max_iter = max_iter
                best_score = silhouette_avg
                best_purity = purity_base

print(f"Melhor número de centros de clusters: {best_ncenters}")
print(f"Melhor valor de erro: {best_error}")
print(f"Melhor número máximo de iterações: {best_max_iter}")
print(f"Melhor Coeficiente de Silhueta: {best_score}")
print(f"Melhor purity: {best_purity}")

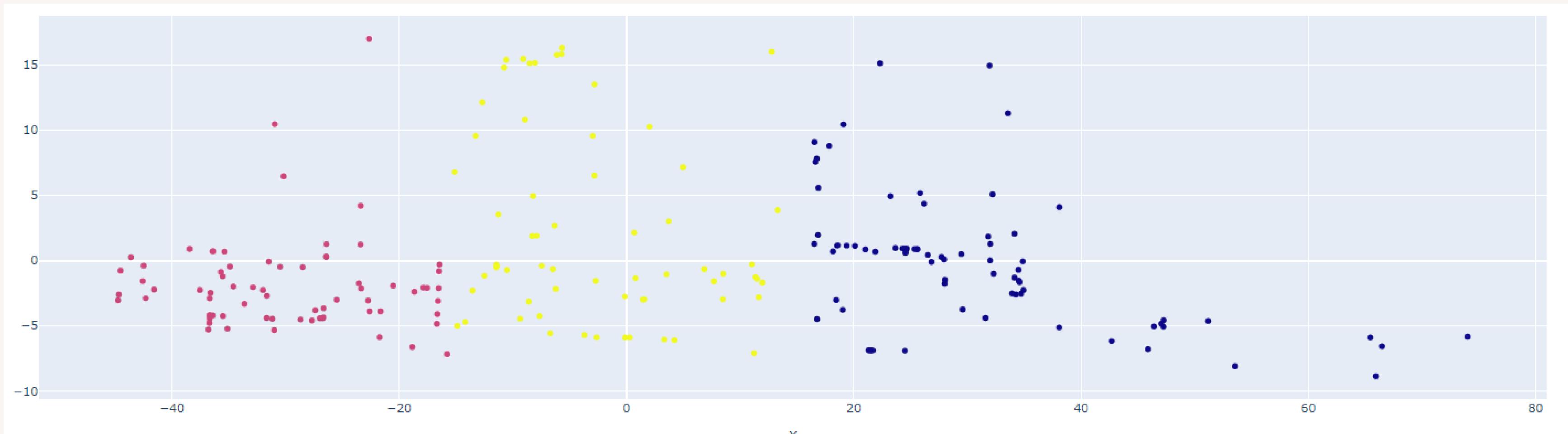
Melhor número de centros de clusters: 2
Melhor valor de erro: 0.001
Melhor número máximo de iterações: 100
Melhor Coeficiente de Silhueta: 0.6046457246230491
Melhor purity: 0.43155893536121676
```

Apesar de não conseguirmos uma melhora significativa no coeficiente de silhueta, conseguimos aumentar a pureza do modelo para essa base dados.

RESULTADOS

1

KMEANS



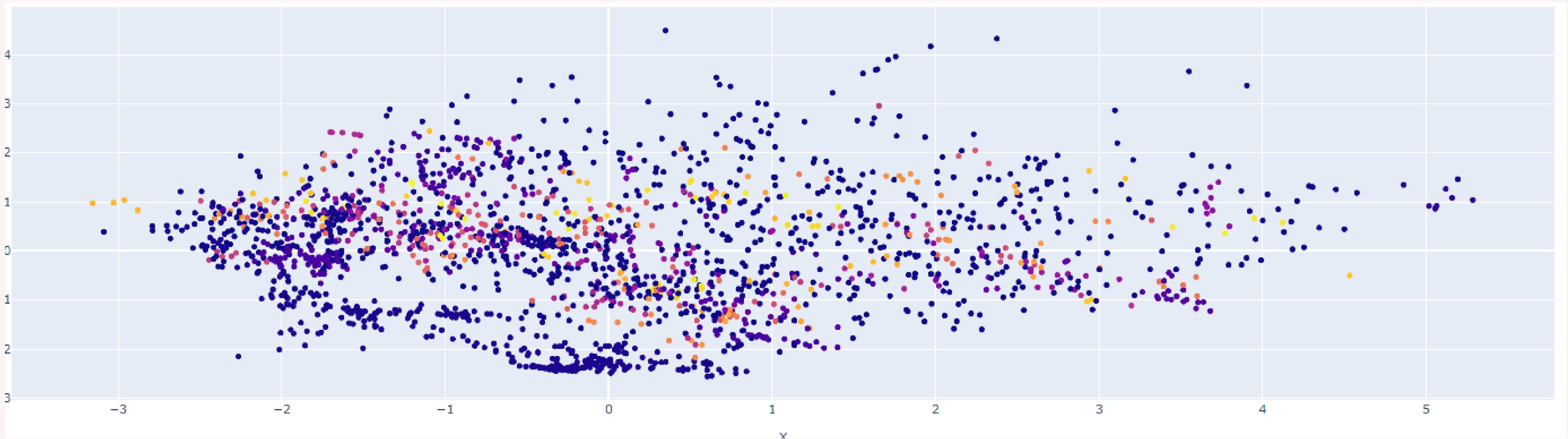
Silhueta: 0.566

Pureza: 0.43

RESULTADOS

1

DBSCAN

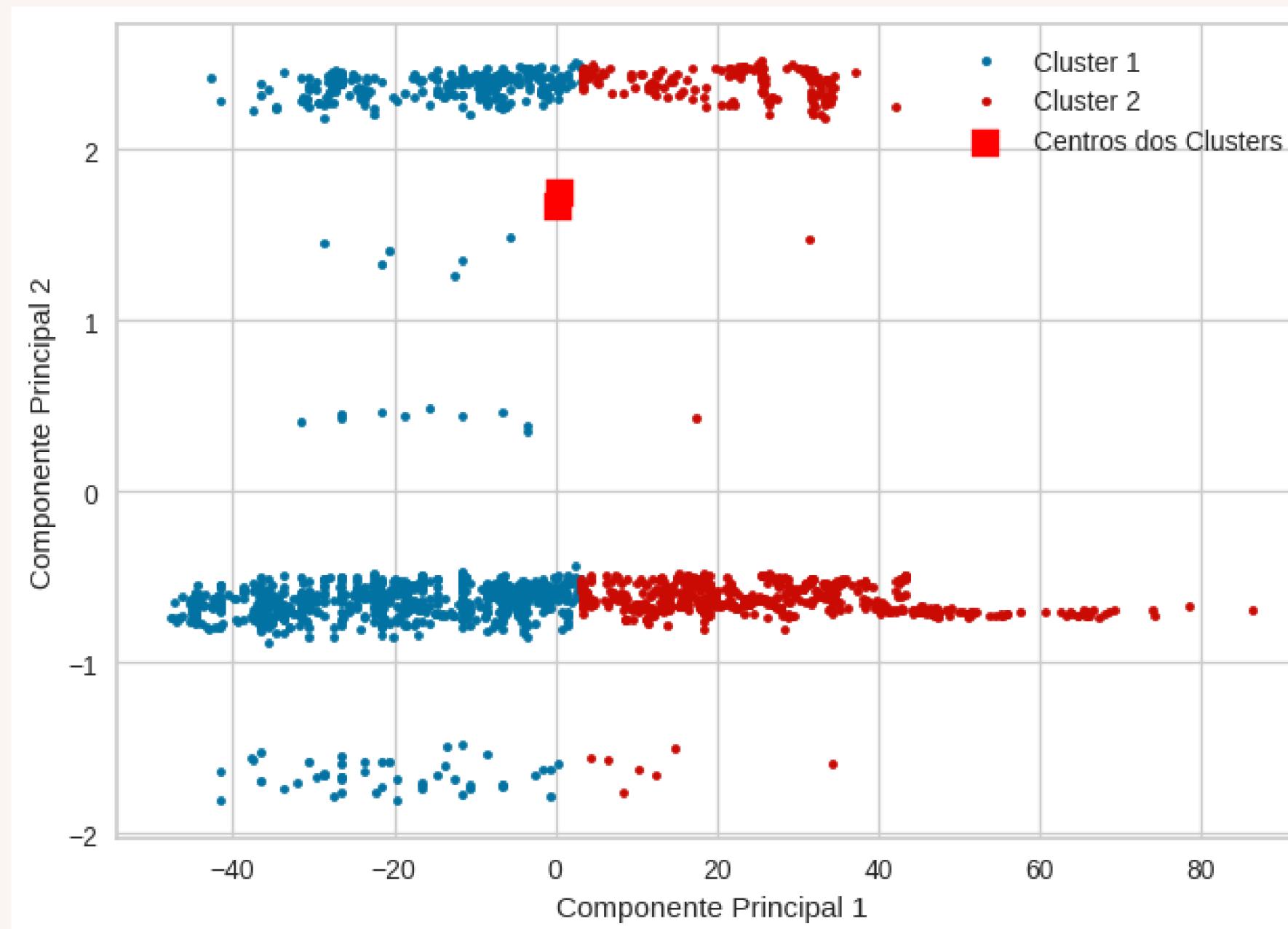


Silhueta: -0.05

Pureza: 0.71

RESULTADOS

1 CMEANS



Silhueta: 0.604

Pureza: 0.431

CONCLUSÃO

A partir de todos os testes, chegamos a conclusão de que o C-Means foi o modelo que teve melhor desempenho em questão de coeficiente de silhueta, enquanto o DBSCAN teve o melhor índice de pureza, apesar de seu baixo coeficiente de silhueta. Além disso, chegamos a conclusão de que a melhoria mais eficiente feita no kmeans foi a retirada de colunas com alto grau de correlação, porém não apresentou uma melhora significativa na alteração dos valores de seus parâmetros, uma vez que foi constatado que os melhores valores eram os próprios valores default. Enquanto para os outros dois modelos, a mudança de valores para seus parâmetros acarretou em uma melhora significativa no desempenho de ambos.