Integrantes: Kennedy Edmilson, Gabriel Oliveira, Gabriel Schneider e Vinícius Novaes.

```
!pip install plotly --upgrade
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (5.8.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-packages (from plotly) (8.0.
```

```
import pandas
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
import plotly.express as px
import plotly.graph_objects as go
import numpy as np
```

```
datapath = './college-data.csv'
df_orig = pandas.read_csv(datapath)
df_orig = df_orig.drop('Unnamed: 0',axis=1)
df_orig
```

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Book
0	Yes	1660	1232	721	23	52	2885	537	7440	3300	45
1	Yes	2186	1924	512	16	29	2683	1227	12280	6450	75
2	Yes	1428	1097	336	22	50	1036	99	11250	3750	40
3	Yes	417	349	137	60	89	510	63	12960	5450	45
4	Yes	193	146	55	16	44	249	869	7560	4120	80
...	...	...	...	...	...	...	...	...	...	...	.
772	No	2197	1515	543	4	26	3089	2029	6797	3900	50

```
df_orig.columns
```

```
Index(['Private', 'Apps', 'Accept', 'Enroll', 'Top10perc', 'Top25perc',
       'F.Undergrad', 'P.Undergrad', 'Outstate', 'Room.Board', 'Books',
       'Personal', 'PhD', 'Terminal', 'S.F.Ratio', 'perc.alumni', 'Expend',
       'Grad.Rate'],
      dtype='object')
```

777 rows × 18 columns

```
df_orig.describe()
```

Percebemos que há uma grande diferença entre os mínimos e máximos dos valores de alguns atributos, sendo necessário a normalização.

```
df_orig.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 777 entries, 0 to 776
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Private      777 non-null    object 
 1   Apps         777 non-null    int64  
 2   Accept       777 non-null    int64  
 3   Enroll       777 non-null    int64  
 4   Top10perc    777 non-null    int64  
 5   Top25perc    777 non-null    int64  
 6   F.Undergrad  777 non-null    int64  
 7   P.Undergrad  777 non-null    int64  
 8   Outstate     777 non-null    int64  
 9   Room.Board   777 non-null    int64  
 10  Books        777 non-null    int64  
 11  Personal     777 non-null    int64  
 12  PhD          777 non-null    int64  
 13  Terminal     777 non-null    int64  
 14  S.F.Ratio    777 non-null    float64
 15  perc.alumni  777 non-null    int64  
 16  Expend       777 non-null    int64  
 17  Grad.Rate    777 non-null    int64  
dtypes: float64(1), int64(16), object(1)
memory usage: 109.4+ KB
```

Percebe-se que temos atributos categóricos(não são numéricos).

Iremos categorizar a classe "private"

```
from sklearn.preprocessing import LabelEncoder
df_orig["Private"] = LabelEncoder().fit_transform(df_orig["Private"])
df_orig["Private"]
```

```
0      1
1      1
2      1
3      1
4      1
      ..
772    0
773    1
774    1
775    1
776    1
Name: Private, Length: 777, dtype: int64
```

Plotamos as correlações entre os atributos e classe apenas para obter um panorama geral.

```
plt.subplots(figsize=(16,12))
sns.heatmap(
    df_orig.corr(),
    annot=True,
    square=True,
    cbar=True
)
```

```
# Here we removed redundant data and the labels
df_orig = df_orig.drop(['Terminal', 'Top25perc'],axis=1)
df_orig
```

```
sns.set_style('whitegrid')
sns.lmplot('Expend', 'PhD', data=df_orig, fit_reg=False, aspect=1, palette='coolwarm')
```

```
sns.set_style('whitegrid')
sns.lmplot('Outstate', 'Grad.Rate', data=df_orig, fit_reg=False, aspect=1, palette='coolwarm')
```

```
sns.lmplot('S.F.Ratio', 'perc.alumni', data=df_orig, fit_reg=False, aspect=1, palette='coolwarm')
```

Dropamos a classe que será predita.

```
df = df_orig.drop(['Private'], axis=1)
df
```

```
cnt=0
tradutor = {}
for col in df.columns:
  tradutor[col] =cnt
  cnt+=1
```

Iremos normalizar os dados, a fim da diferença de ordem de magnitude dos atributos não afetar o algorítmo.

```
from sklearn.preprocessing import MinMaxScaler
df =  MinMaxScaler().fit_transform(df)
df
```

```
    array([[0.03288693, 0.04417701, 0.10791254, ..., 0.1875    , 0.0726714 ,
            0.46296296],
           [0.04384229, 0.07053089, 0.07503539, ..., 0.25      , 0.13838671,
            0.42592593],
           [0.0280549 , 0.03903572, 0.04734938, ..., 0.46875   , 0.10460535,
```

```
 0.40740741],
...,
[0.04198863, 0.07018813, 0.10382256, ..., 0.3125    , 0.09683865,
 0.36111111],
[0.22127341, 0.09067713, 0.20166745, ..., 0.765625  , 0.70126492,
 0.82407407],
[0.06056693, 0.06790312, 0.10319333, ..., 0.4375    , 0.02494015,
 0.82407407]])
```

Nós estamos fazendo um agrupamento com 5 centróides(escolha aleatória).

```
km = KMeans(n_clusters=5)
km.fit(df)
rotulos = km.fit_predict(df)
centroides = km.cluster_centers_
#labels = km.labels_
#df['label'] = labels
```

Vendo em gráfico a distribuição de acordo com os atributos: Apps e Accept

```
grafico1 = px.scatter(x = df[:,tradutor['Apps']], y = df[:,tradutor['Accept']], color = rotulos)
grafico3 = go.Figure(data = grafico1.data )
grafico3.show()
```

```
grafico1 = px.scatter(x = df[:,tradutor['Outstate']], y = df[:,tradutor['Grad.Rate']], color = rotulos)
grafico1.show()
```

```
grafico1 = px.scatter(x = df[:,tradutor['Expend']], y = df[:,tradutor['PhD']], color = rotulos)
grafico1.show()
```

Podemos combinar todos atributos em 2, para visualizamos de maneira 2d com PCA.

```
from sklearn.decomposition import PCA


pca = PCA(n_components=2)
df_orig_mais_pca = pca.fit_transform(df)
df_orig_mais_pca.shape
```

```
    (777, 2)
```

```
grafico = px.scatter(x= df_orig_mais_pca[:,0], y = df_orig_mais_pca[:,1], color=rotulos)
grafico.show()
```

```
sse = {}
for k in range(1, 9):

    km = KMeans(n_clusters=k, random_state = 0)
    test = km.fit(df)

    sse[k] = km.inertia_

plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.show()
```

Aqui, podemos ver 3 centróides são bons. Depois da 4º iteração, a taxa devariação diminui abruptamente.

Iremos fazer a análise por Silhueta

```
from sklearn.metrics import silhouette_samples, silhouette_score
for i, k in enumerate([2, 3, 4]):
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # Run the Kmeans algorithm
    km = KMeans(n_clusters=k)
    labels = km.fit_predict(df)
    centroids = km.cluster_centers_

    # Get silhouette samples
    silhouette_vals = silhouette_samples(df, labels)

    # Silhouette plot
    y_ticks = []
    y_lower, y_upper = 0, 0
    for i, cluster in enumerate(np.unique(labels)):
        cluster_silhouette_vals = silhouette_vals[labels == cluster]
        cluster_silhouette_vals.sort()
        y_upper += len(cluster_silhouette_vals)
        ax1.barh(range(y_lower, y_upper), cluster_silhouette_vals, edgecolor='none', height=1)
        ax1.text(-0.03, (y_lower + y_upper) / 2, str(i + 1))
        y_lower += len(cluster_silhouette_vals)

    # Get the average silhouette score and plot it
    avg_score = np.mean(silhouette_vals)
    ax1.axvline(avg_score, linestyle='--', linewidth=2, color='green')
    ax1.set_yticks([])
    ax1.set_xlim([-0.1, 1])
    ax1.set_xlabel('Silhouette coefficient values')
    ax1.set_ylabel('Cluster labels')
    ax1.set_title('Silhouette plot for the various clusters', y=1.02);
```

```
# Scatter plot of data colored with labels
ax2.scatter(df[:, 0], df[:, 1], c=labels)
ax2.scatter(centroids[:, 0], centroids[:, 1], marker='*', c='r', s=250)
ax2.set_xlim([-2, 2])
ax2.set_xlim([-2, 2])
ax2.set_xlabel('Eruption time in mins')
ax2.set_ylabel('Waiting time to next eruption')
ax2.set_title('Visualization of clustered data', y=1.02)
ax2.set_aspect('equal')
plt.tight_layout()
plt.suptitle(f'Silhouette analysis using k = {k}',
             fontsize=16, fontweight='semibold', y=1.05);
```

Analisando os scores(linha vertical tracejada verde), percebemos que é maior para 3 clusters, corroborando com o método Elbow.

Iremos rodar o algorítmo para 3 centróides.

```
km = KMeans(n_clusters=3)
km.fit(df)
rotulos = km.fit_predict(df)
centroides = km.cluster_centers_
```

Iremos plotar os mesmos 3 gráficos anteriores para compararmos.

```
grafico1 = px.scatter(x = df[:,tradutor['Apps']], y = df[:,tradutor['Accept']], color = rotulos)
grafico3 = go.Figure(data = grafico1.data )
grafico3.show()
```

```
grafico1 = px.scatter(x = df[:,tradutor['Outstate']], y = df[:,tradutor['Grad.Rate']], color = rotulos)
grafico1.show()
```

```
grafico1 = px.scatter(x = df[:,tradutor['Expend']], y = df[:,tradutor['PhD']], color = rotulos)
grafico1.show()
```

Plotando com PCA, com apenas 3, conseguimos definir bem os clusters.

```
grafico = px.scatter(x= df_orig_mais_pca[:,0], y = df_orig_mais_pca[:,1], color=rotulos)
grafico.show()
```