Mini-projeto Não Supervisionado

K-means é um algoritmo de aprendizado de máquina não supervisionado para a clusterizaçã
A ideia principal do K-means é agrupar dados que possuam características semelhantes em
calcular a semelhança entre os pontos e, em seguida, agrupá-
los em clusters. O algoritmo funciona iterativamente, onde inicialmente um número k de cent

K-medoids é um algoritmo de clusterização não supervisionado que é semelhante ao K-means, mas ao invés de usar centróides como representantes dos clusters, ele usa pontos reais dos dados, chamados de medóides. Um medóide é um ponto do cluster que minimiza a soma das distâncias entre ele e todos os outros pontos do cluster. O algoritmo K-medoids é mais robusto do que o K-means para dados com ruído, pois a escolha dos medóides é menos sensível a outliers. Além disso, ele também pode ser mais robusto em relação à inicialização dos centróides, uma vez que a escolha dos medóides é menos influenciada por valores extremos.

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sn
import math
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
!pip install scikit-learn-extra
from sklearn_extra.cluster import KMedoids
import sklearn.metrics as sklearn_metrics
from scipy.spatial import distance
```

```
Defaulting to user installation because normal site-packages is not write
able
Requirement already satisfied: scikit-learn-extra in /home/murilobn/.loca
l/lib/python3.10/site-packages (0.3.0)
Requirement already satisfied: numpy>=1.13.3 in /home/murilobn/.local/lib
/python3.10/site-packages (from scikit-learn-extra) (1.22.4)
Requirement already satisfied: scipy>=0.19.1 in /home/murilobn/.local/lib
/python3.10/site-packages (from scikit-learn-extra) (1.7.3)
Requirement already satisfied: scikit-learn>=0.23.0 in /home/murilobn/.lo
cal/lib/python3.10/site-packages (from scikit-learn-extra) (1.0.2)
Requirement already satisfied: joblib>=0.11 in /home/murilobn/.local/lib/
python3.10/site-packages (from scikit-learn>=0.23.0->scikit-learn-extra)
(1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/murilobn/.lo
cal/lib/python3.10/site-packages (from scikit-learn>=0.23.0->scikit-learn
-extra) (3.1.0)
```

Dataset

O conjunto de dados "US News and World Reports College Data" é um conjunto de dados de avaliações de faculdades e universidades nos Estados Unidos, disponibilizado no Kaggle, onde temos um conjunto de dados com informações de 777 instituições de ensino superior, incluindo características como tamanho do corpo estudantil, taxa de aceitação, despesas com educação, porcentagem de alunos que se formam em quatro anos, entre outras e também inclui informações sobre a classificação de cada instituição em diversas categorias, como "Melhores Faculdades de Artes Liberais" e "Melhores Faculdades Regionais do Norte". Vamos visualizar o Dataset para saber que tipos de dados estamos tratando:

In [ ]:
```
# Getting the data
csv_path = 'College.csv'
data = pd.read_csv(csv_path)
data.head()
```

Out[ ]:

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergra
0	Abilene Christian University	Yes	1660	1232	721	23	52	2885	53
1	Adelphi University	Yes	2186	1924	512	16	29	2683	122
2	Adrian College	Yes	1428	1097	336	22	50	1036	9
3	Agnes Scott College	Yes	417	349	137	60	89	510	6
4	Alaska Pacific University	Yes	193	146	55	16	44	249	86

In [ ]:
```
# Checking how many information we are dealing with and how it is distrib
data.shape
```

Out[ ]:
```
(777, 19)
```

O que é apresentado em cada coluna:

- Apps → Número de inscrições recebidas
- Accept → Número de inscrições aceitas
- Enroll → Número de novos alunos matriculados
- Top10perc → Porcentagem de novos alunos do top 10% da turma do ensino médio
- Top25perc → Porcentagem de novos alunos do top 25% da turma do ensino médio
- F.Undergrad → Número de alunos de graduação em tempo integral
- P.Undergrad → Número de alunos de graduação em tempo parcial
- Outstate → Mensalidade para estudantes fora do estado
- Room.Board → Custos de alojamento e alimentação
- Books → Custos estimados de livros
- Personal → Gastos pessoais estimados
- PhD → Porcentagem do corpo docente com doutorado
- Terminal → Porcentagem do corpo docente com grau terminal
- S.F.Ratio → Relação aluno/faculdade
- perc.alumni → Porcentagem de ex-alunos que doam
- Expend Instructional → Despesas por aluno
- Grad.Rate → Taxa de graduação

Removendo Colunas e Dados Nulos

Removeremos a coluna “unnamed” que contém o nome de cada universidade, pois não virá a ser útil em nossa análise. Logo após isso, removeremos a coluna "Private" para que possamos tratar o problema de forma não-supervisionada. Checaremos as entradas nulas, dadas por ‘NaN’ e as removemos caso existam. Neste caso, não há nenhum aparecimento, não sendo então necessário remoção.

In [ ]:
```
# Dropping the 'Unnamed' column, with the names of each university
data_att = data.drop(['Unnamed: 0'], axis=1)
data_att.head()
```

Out[ ]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate
0	Yes	1660	1232	721	23	52	2885	537	7440
1	Yes	2186	1924	512	16	29	2683	1227	12280
2	Yes	1428	1097	336	22	50	1036	99	11250
3	Yes	417	349	137	60	89	510	63	12960
4	Yes	193	146	55	16	44	249	869	7560

In [ ]:
```
# Remove Private column, but keeping a copy for future evaluation
data_att_private = data_att
data_att = data_att.drop('Private', axis=1)
```

In [ ]:
```
# Checking of NaN entries
data_att.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 777 entries, 0 to 776
Data columns (total 17 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Apps         777 non-null    int64  
 1   Accept       777 non-null    int64  
 2   Enroll       777 non-null    int64  
 3   Top10perc    777 non-null    int64  
 4   Top25perc    777 non-null    int64  
 5   F.Undergrad  777 non-null    int64  
 6   P.Undergrad  777 non-null    int64  
 7   Outstate     777 non-null    int64  
 8   Room.Board   777 non-null    int64  
 9   Books        777 non-null    int64  
 10  Personal     777 non-null    int64  
 11  PhD          777 non-null    int64  
 12  Terminal     777 non-null    int64  
 13  S.F.Ratio    777 non-null    float64
 14  perc.alumni  777 non-null    int64  
 15  Expend       777 non-null    int64  
 16  Grad.Rate    777 non-null    int64  
dtypes: float64(1), int64(16)
memory usage: 103.3 KB
```

Buscando Valores Impossíveis

Agora, percorreremos o dataset em busca de valores atípicos que podem poluir nosso dataset, como taxas de graduação acima de 100% ou abaixo de 0%, por exemplo.

In [ ]:
```
# Taking a look at the general distributions of the dataset, finding outl
data_att.describe()
```

Out[ ]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Ur
count	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	77
mean	3001.638353	2018.804376	779.972973	27.558559	55.796654	3699.907336	85
std	3870.201484	2451.113971	929.176190	17.640364	19.804778	4850.420531	152
min	81.000000	72.000000	35.000000	1.000000	9.000000	139.000000	
25%	776.000000	604.000000	242.000000	15.000000	41.000000	992.000000	9
50%	1558.000000	1110.000000	434.000000	23.000000	54.000000	1707.000000	35
75%	3624.000000	2424.000000	902.000000	35.000000	69.000000	4005.000000	96
max	48094.000000	26330.000000	6392.000000	96.000000	100.000000	31643.000000	2183

Removeremos esses valores para obter um resultado mais consistente para o algoritmo:

In [ ]:
```
# Removing rates over 100%
data_att = data_att.drop(data_att[data_att['PhD'] > 100].index)
data_att = data_att.drop(data_att[data_att['Grad.Rate'] > 100].index)
data_att_private = data_att_private.drop(data_att_private[data_att_privat
data_att_private = data_att_private.drop(data_att_private[data_att_privat
data_att.describe()
```

Out[ ]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Ur
count	775.000000	775.000000	775.000000	775.000000	775.000000	775.000000	77
mean	3003.738065	2018.963871	780.992258	27.589677	55.834839	3706.596129	85
std	3874.059459	2453.129603	930.130671	17.649382	19.813695	4854.888544	152
min	81.000000	72.000000	35.000000	1.000000	9.000000	139.000000	
25%	778.000000	605.500000	242.000000	15.000000	41.000000	990.000000	9
50%	1558.000000	1110.000000	434.000000	23.000000	54.000000	1708.000000	35
75%	3610.000000	2413.000000	902.500000	35.000000	69.000000	4055.500000	96
max	48094.000000	26330.000000	6392.000000	96.000000	100.000000	31643.000000	2183

Logo após remover esses valores, checaremos novamente os dados:

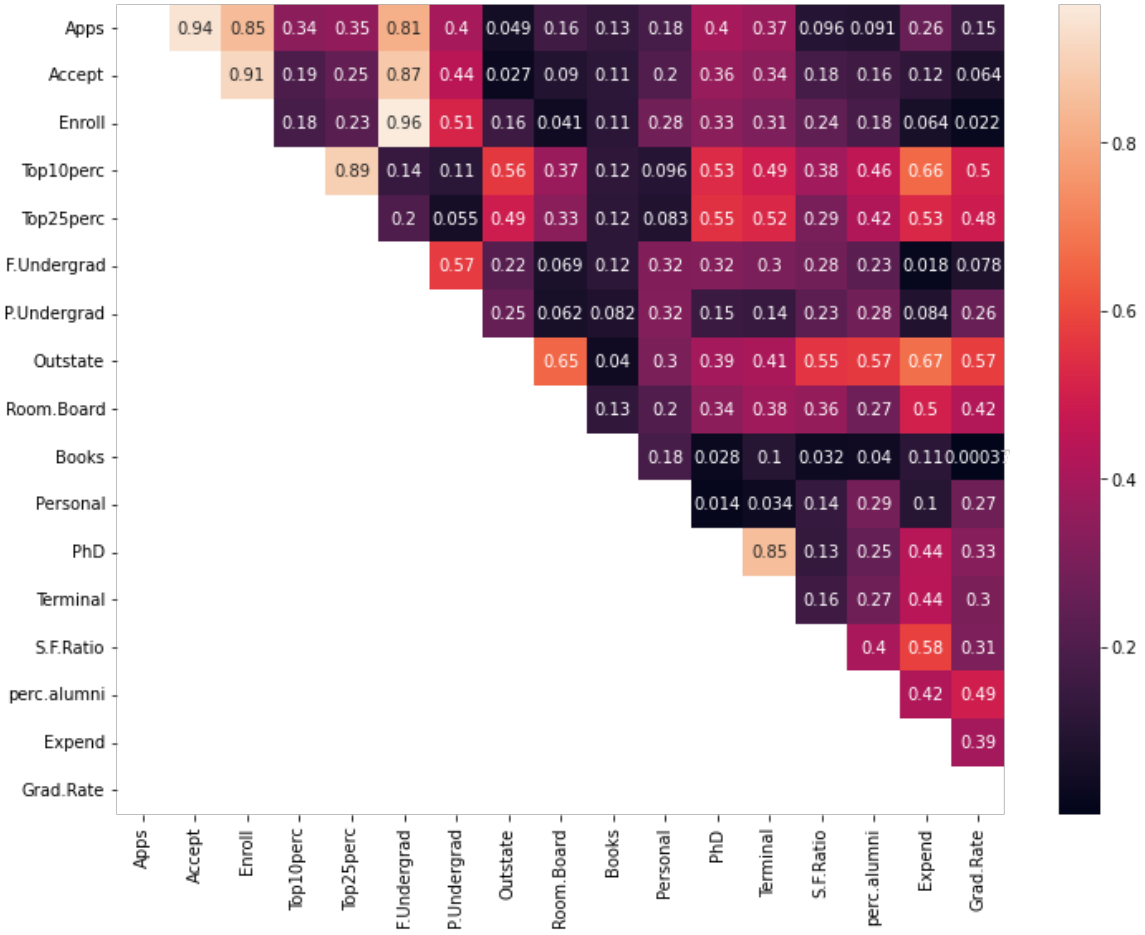
In [ ]:
```
# Rechecking our dataset
data_att.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 775 entries, 0 to 776
Data columns (total 17 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Apps         775 non-null    int64  
 1   Accept       775 non-null    int64  
 2   Enroll       775 non-null    int64  
 3   Top10perc    775 non-null    int64  
 4   Top25perc    775 non-null    int64  
 5   F.Undergrad  775 non-null    int64  
 6   P.Undergrad  775 non-null    int64  
 7   Outstate     775 non-null    int64  
 8   Room.Board   775 non-null    int64  
 9   Books        775 non-null    int64  
 10  Personal     775 non-null    int64  
 11  PhD          775 non-null    int64  
 12  Terminal     775 non-null    int64  
 13  S.F.Ratio    775 non-null    float64
 14  perc.alumni  775 non-null    int64  
 15  Expend       775 non-null    int64  
 16  Grad.Rate    775 non-null    int64  
dtypes: float64(1), int64(16)
memory usage: 109.0 KB
```

Matriz De Correlação

Após o tratamento inicial dos dados, iremos fazer uma matriz de correlação para averiguar as correlações entre os atributos.

In [ ]:
```
# Correlation matrix to check for unusable values
corr_matrix = data_att.corr()
upper_tri = abs(corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1
plt.figure(figsize=(12,9))
sns.heatmap(upper_tri, annot=True)
plt.show()
```



Pelo fato desse dataset ter um número alto de atributos, removeremos algumas das colunas que tem alta correlação entre si para evitar colinearidade. Feito isso, teremos menos atributos que por sua vez serão mais importantes para o resultado final. Os atributos selecionados foram "Apps", "Accept", "Enroll", "Terminal", "Top10perc", "Top25perc", que além de apresentarem alto grau de correlação com outros atributos, nós entendemos que eles não são necessariamente importantes para o resultado final de um ponto de vista lógico.

In [ ]:
```
# Remove variables with high multicolinearity
data_att = data_att.drop(["Apps", "Accept", "Enroll", "Terminal", "Top10p
data_att_private = data_att_private.drop(["Apps", "Accept", "Enroll", "Te
```

Mapeando Atributos de Custo

Para os atributos que representam custos do estudante, isto é, atributos que precisam ser avaliados no que se refere à variação dos valores, utilizaremos um gráfico box-plot para separá-los em quartis. Realizando esse processo, descobriremos outliers, que aparecem após a avaliação dos valores e se mostram bastante destoantes do restante e portanto, iremos removê-los também:

In [ ]:
```
# Mapping cost Columns
def attr_info(attr):
    bp = plt.boxplot(data_att[attr], showmeans=True)

    minimum = [item.get_ydata()[0] for item in bp["caps"]][::2]
    q1 = [min(item.get_ydata()) for item in bp["boxes"]]
    q3 = [max(item.get_ydata()) for item in bp["boxes"]]
    maximum = [item.get_ydata()[0] for item in bp["caps"]][1::2]
    return minimum[0], q1[0], q3[0], maximum[0]

def map_cost(value, info):
    if value < info[0]:
        return 0
    elif info[0] <= value < info[1]:
        return 1
    elif info[1] <= value < info[2]:
        return 2
    elif info[2] <= value < info[3]:
        return 3
    else:
        return 4

attrs = ["Expend", "Books", "Outstate", "Personal", "Room.Board"]

# for attr in attrs:
#     info = attr_info(attr)
#     data_att[attr] = data_att[attr].map(lambda v: map_cost(v, info))
#     data_att_private[attr] = data_att_private[attr].map(lambda v: map_c

#     data_att = data_att[data_att[attr] < 4]
#     data_att = data_att[data_att[attr] > 0]
#     data_att_private = data_att_private[data_att_private[attr] < 4]
#     data_att_private = data_att_private[data_att_private[attr] > 0]


data_att.info()
```

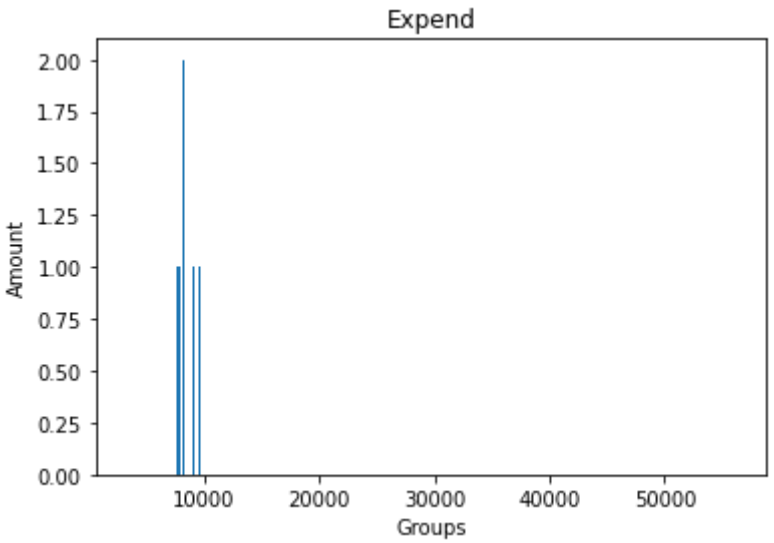
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 775 entries, 0 to 776
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   F.Undergrad  775 non-null    int64
 1   P.Undergrad  775 non-null    int64
 2   Outstate     775 non-null    int64
 3   Room.Board   775 non-null    int64
 4   Books        775 non-null    int64
 5   Personal     775 non-null    int64
 6   PhD          775 non-null    int64
 7   S.F.Ratio    775 non-null    float64
 8   perc.alumni  775 non-null    int64
 9   Expend       775 non-null    int64
 10  Grad.Rate    775 non-null    int64
dtypes: float64(1), int64(10)
memory usage: 72.7 KB
```

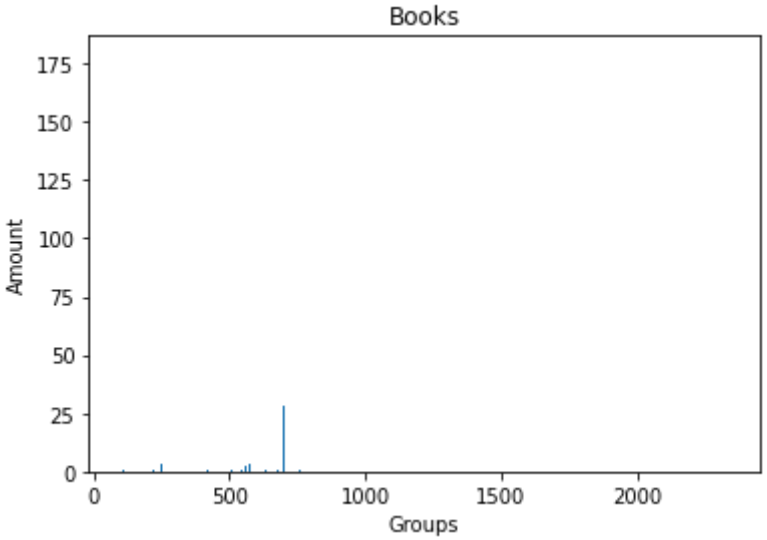
Visualizando a Distribuição dos Dados

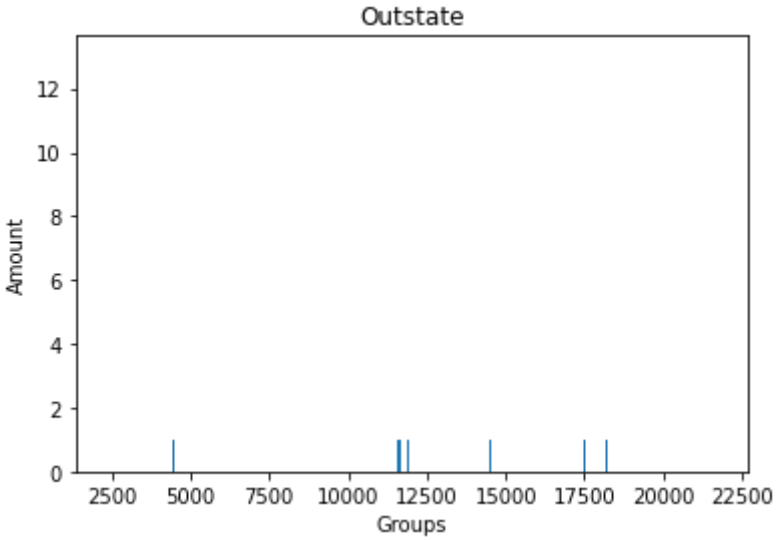
Após todos os processos de tratamento e normalização dos dados, temos o dataset final a ser utilizado pelo algoritmo. Agora, iremos visualizar brevemente a nova distribuição dos dados de custo que foram mapeados:

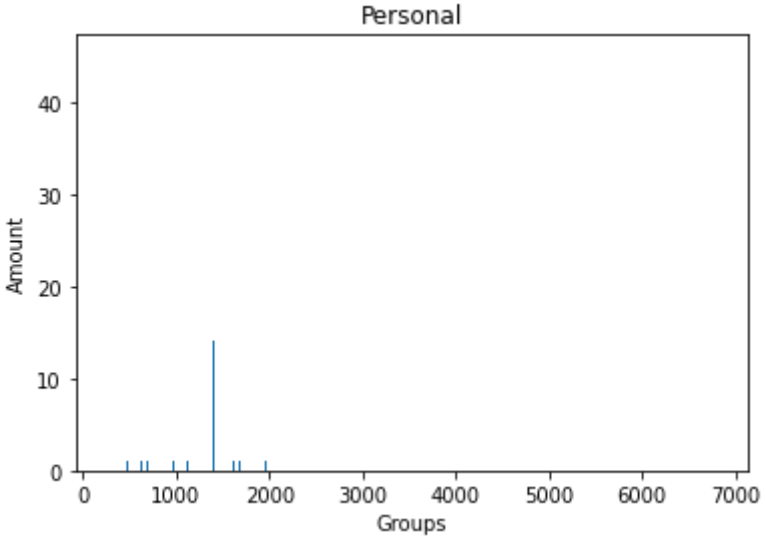
In [ ]:
```
def plot_bar(attr):
    labels = data_att[attr].unique()
    labels.sort()
    values = data_att[attr].value_counts().sort_index()

    plt.bar(labels, values)
    plt.title(attr)
    plt.xlabel("Groups")
    plt.ylabel("Amount")
    plt.show()
```

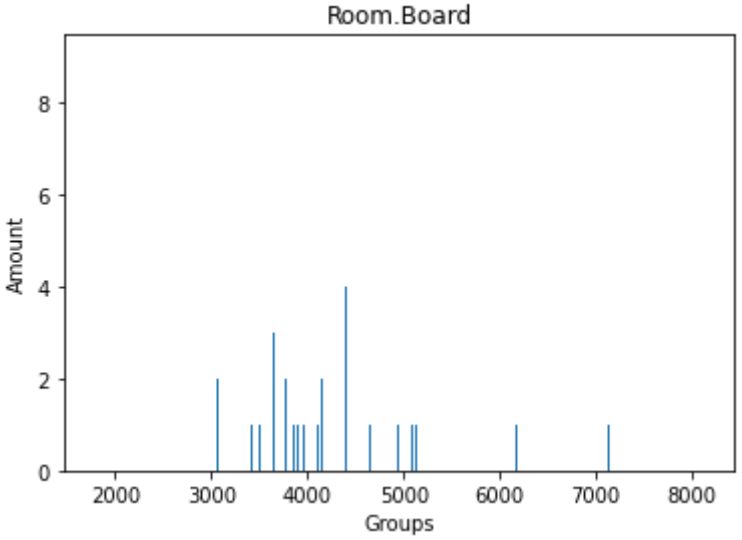
In [ ]:
```
for attr in attrs:
    plot_bar(attr)
```






Outstate
Amount
Groups
0
2
4
6
8
10
12
2500
5000
7500
10000
12500
15000
17500
20000
22500


Personal
Amount
Groups
0
10
20
30
40
0
1000
2000
3000
4000
5000
6000
7000


Room.Board
Amount
Groups
0
2
4
6
8
2000
3000
4000
5000
6000
7000
8000

Normalizando os dados

Normalizamos os dados do Dataset, visto que esses algoritmos de clusterização dependem fortemente da distância entre os pontos de dadods para determinar os clusters e, por consequência, são extremamente sensíveis a dados com valores extremamente distintos. Ter as características dos dados com escalas muito diferentes acaba fazendo com que a distância calculada possa ser dominada pelas características com maiores valores absolutos.

Ao normalizar os dados, as características são transformadas em escala comum, como pode ser visto abaixo, garantindo que cada um contribua igualmente para a distância entre os pontos dados:

In [ ]:
```
# Normalizing the data
data_att_no_norm = data_att
scaled_data_att = pd.DataFrame(normalize(data_att, axis=0), columns=data_
data_att = scaled_data_att
data_att.head()
```

Out[ ]:

	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	PhD	S.F.Ratio
0	0.016973	0.011038	0.023870	0.026376	0.028183	0.052566	0.033765	0.044438
1	0.015785	0.025220	0.039399	0.051552	0.046972	0.035841	0.013988	0.029953
2	0.006095	0.002035	0.036094	0.029972	0.025052	0.027836	0.025565	0.031671
3	0.003000	0.001295	0.041581	0.043560	0.028183	0.020907	0.044377	0.018905
4	0.001465	0.017862	0.024255	0.032930	0.050103	0.035841	0.036659	0.029216

Criando uma Cópia do Dataset

Criaremos uma cópia do nosso dataset tratado e normalizado para que haja uma cópia para utilizarmos no algoritmo K-means e outra para o K-medoids:

In [ ]:
```
# Creating separate copies for each method we will be using
data_att_means = data_att
data_att_medoids = data_att
```

Métricas para descobrir o melhor valor de K

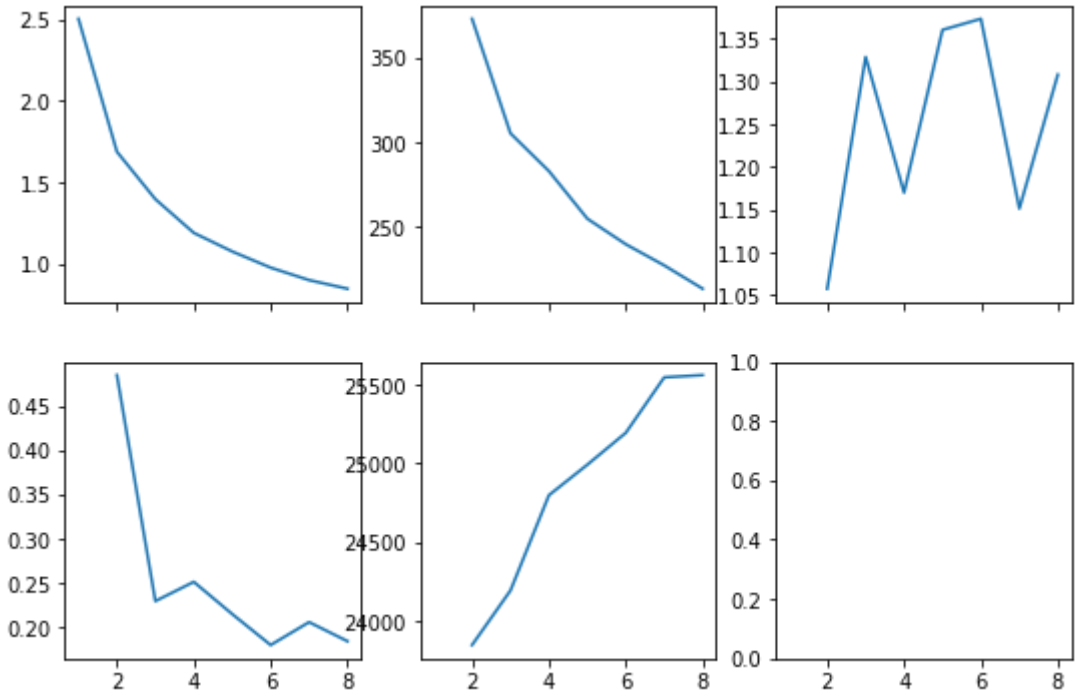
Para descobrir qual é o melhor valor de K, faremos um cálculo de métricas para avaliar a qualidade do modelo. A função bic_score vai calcular o valor da métrica Bayesian Information Criterion (BIC) para o modelo e vai avaliar a qualidade do modelo considerando a capacidade de ajuste do modelo e seu número de parâmetros. Quanto menor o valor do BIC, melhor é o modelo.

Além do BIC, outras métricas de avaliação são feitas, como o "Elbow Method", o índice de Calinski-Harabasz, o índice de Davies-Bouldin e o coeficiente de silhueta. Para isso, o código assume uma lista de K e testa o desempenho do K-Médias em cada métrica para que seja possível definir qual o melhor K. Observando os gráficos resultantes, é feita a escolha de K=2, sendo aquele que em mais métricas foi responsável por obter o melhor desempenho.

In [ ]:
```
def bic_score(kmeans, X):
    """
    Computes the BIC metric for a given clusters

    Parameters:
    -----------------------------------------
    kmeans:  List of clustering object from scikit learn

    X     :  multidimension np array of data points

    Returns:
    -----------------------------------------
    BIC value
    """
    # assign centers and labels
    centers = [kmeans.cluster_centers_]
    labels = kmeans.labels_
    # number of clusters
    m = kmeans.n_clusters
    # size of the clusters
    n = np.bincount(labels)
    # size of data set
    N, d = X.shape

    # compute variance for all clusters beforehand
    cl_var = (1.0 / (N - m) / d) * sum([sum(distance.cdist(X.iloc[labels
                                                                  'euclidean')**

    const_term = 0.5 * m * np.log(N) * (d+1)

    BIC = np.sum([n[i] * np.log(n[i]) -
                  n[i] * np.log(N) -
                  ((n[i] * d) / 2) * np.log(2*np.pi*cl_var) -
                  ((n[i] - 1) * d / 2) for i in range(m)]) - const_term

    return (BIC)
```

In [ ]:
```
# Calculate scores using various methods
scores = {"elbow": [], "calinski-harabasz": [], "davies-bouldin": [], "si

k_range = [i for i in range(2, 9)]

kmeans = KMeans(n_clusters=1, n_init=10).fit(data_att_means)
scores["elbow"].append(kmeans.inertia_)

for k in k_range:
    kmeans = KMeans(n_clusters=k, n_init=10).fit(data_att_means)
    labels = kmeans.labels_
    scores["elbow"].append(kmeans.inertia_) # Elbow is better
    scores["calinski-harabasz"].append(sklearn_metrics.calinski_harabasz_
    scores["davies-bouldin"].append(sklearn_metrics.davies_bouldin_score(
    scores["silhouette"].append(sklearn_metrics.silhouette_score(data_att
    scores["bic"].append(bic_score(kmeans, data_att_means)) # Lower is be
```

In [ ]:
```
fig, ax = plt.subplots(2, 3, figsize=(9,6), sharex=True, sharey=False)

ax[0, 0].plot([1] + k_range, scores["elbow"], label="elbow")
ax[0, 1].plot(k_range, scores["calinski-harabasz"], label="calinski-harab
ax[0, 2].plot(k_range, scores["davies-bouldin"], label="davies-bouldin")
ax[1, 0].plot(k_range, scores["silhouette"], label="silhouette")
ax[1, 1].plot(k_range, scores["bic"], label="bic")

plt.show()
```



Treinando os Modelos utilizando o K-means e o K-medoids

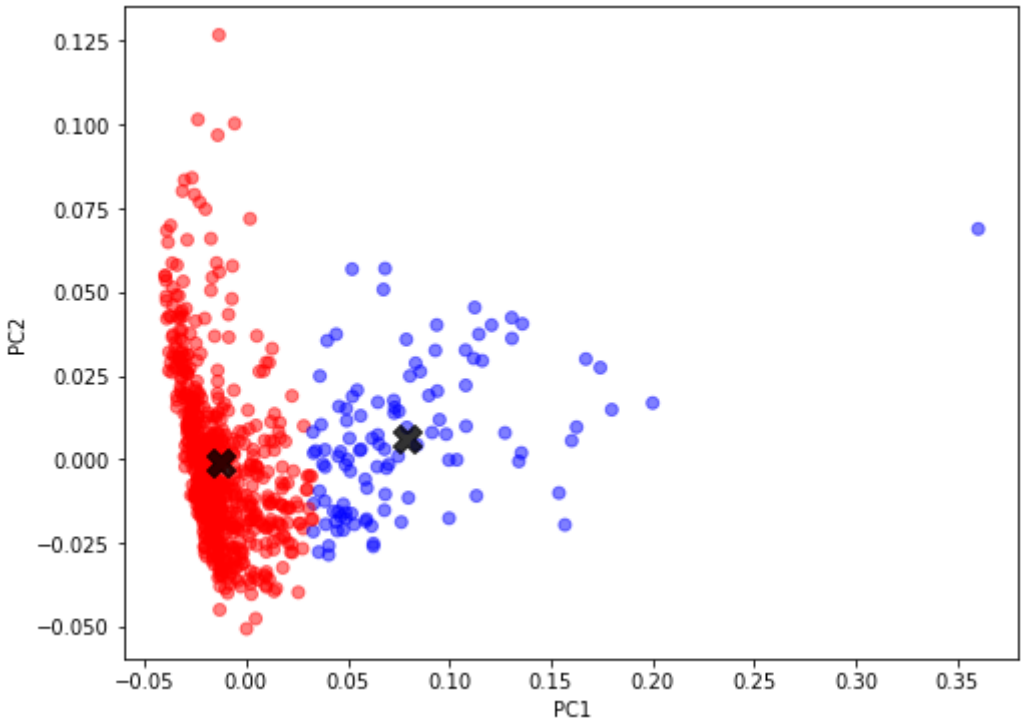
Agora, entraremos na fase de treinamento dos modelos, para isso, é necessário escolher a quantidade de clusters a ser utilizado em cada um dos modelos. O número de clusters selecionado para ambos foi 2, porém é importante frisar que esse número não foi escolhido aleatoriamente, pois mais adiante no nosso projeto, utilizaremos métricas para definir esse valor.

K-Média

In [ ]:
```
kmeans = KMeans(n_clusters=2)
kmeans.fit(data_att_means)
```

Out[ ]:
```
KMeans(n_clusters=2)
```

In [ ]:
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(data_att_means.values)

fig, ax = plt.subplots(figsize=(8, 6))

colors = np.array(['red', 'blue'])
ax.scatter(X_pca[:, 0], X_pca[:, 1], c=colors[kmeans.labels_], alpha=0.5)
ax.scatter(pca.transform(kmeans.cluster_centers_)[:, 0], pca.transform(km
           c='black', s=200, alpha=0.8, marker='X')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
plt.show()
```

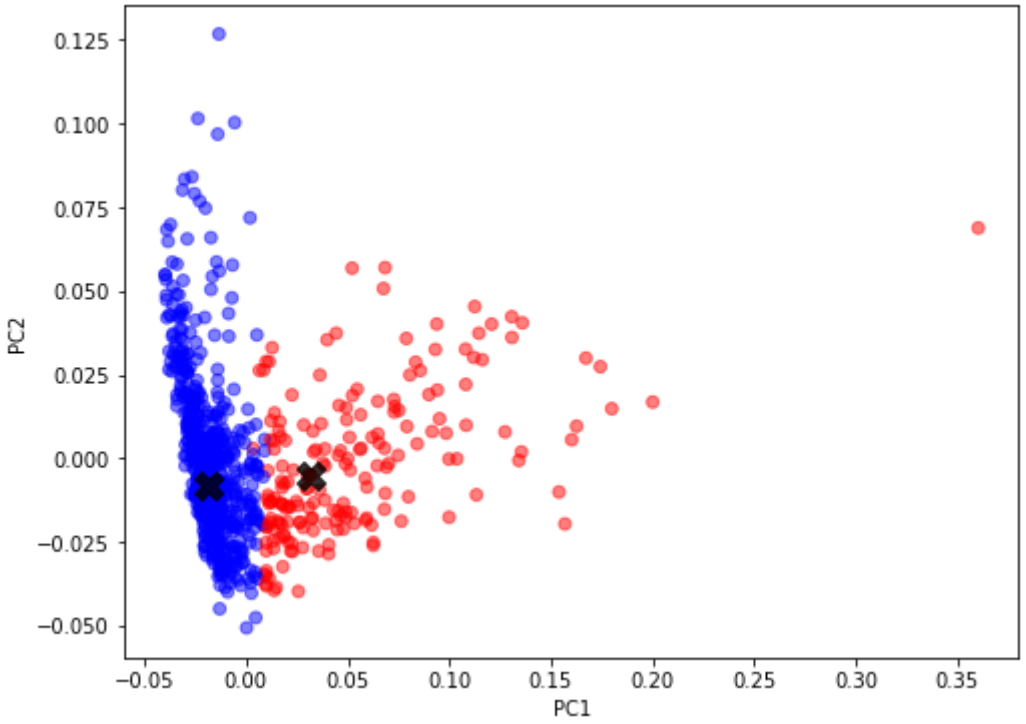


K-Medóide

In [ ]:
```
kmedoids = KMedoids(n_clusters=2)
kmedoids.fit(data_att_medoids)
```

Out[ ]:
```
KMedoids(n_clusters=2)
```

In [ ]:
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(data_att_medoids.values)

fig, ax = plt.subplots(figsize=(8, 6))

colors = np.array(['red', 'blue'])
ax.scatter(X_pca[:, 0], X_pca[:, 1], c=colors[kmedoids.labels_], alpha=0.
ax.scatter(pca.transform(kmedoids.cluster_centers_)[:, 0], pca.transform(
           c='black', s=200, alpha=0.8, marker='X')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
plt.show()
```



Comparação de Métricas

Para uma boa métrica, precisamos ter noção dos conceitos de distância intracluster e intercluster. A distância intracluster define o quão distante do centróide os elementos estão, já a intercluster define o quão distante os clusters estão uns dos outros. Para uma boa métrica, é necessário que ela tenha uma baixa distância intracluster e uma boa distância intercluster. Abaixo, calcularemos as distâncias intra e intercluster por meio dos métodos SSE e Centroid Linkage Distance:

In [ ]:
```
sse = kmeans.inertia_
centroids_dist = math.dist(kmeans.cluster_centers_[0], kmeans.cluster_cen
print(f"Intracluster distance (SSE): {sse}")
print(f"Intercluster distance (Centroid Linkage Distance): {centroids_dis
```

```
Intracluster distance (SSE): 1.6889725361573902
Intercluster distance (Centroid Linkage Distance): 0.09290897195836594
```

In [ ]:
```
sse = kmedoids.inertia_
centroids_dist = math.dist(kmedoids.cluster_centers_[0], kmedoids.cluster_
print(f"Intracluster distance (SSE): {sse}")
print(f"Intercluster distance (Centroid Linkage Distance): {centroids_dis
```

```
Intracluster distance (SSE): 32.209068668281915
Intercluster distance (Centroid Linkage Distance): 0.052329624071337906
```

Perguntas para o Cluster

Com os clusters formados, podemos agora nos dedicar a observar cada cluster e suas características. Vamos, através de informações gerais, determinar qual cluster tem indícios de pertencer a uma determinada classificação de "Private" e, a partir dessa escolha, determinar e visualizar o desempenho dos modelos na tarefa.

Inicialmente, retomamos os valores originais dos atributos utilizados para realizar os agrupamentos, de forma a lidarmos com valores mais facilmente relacionáveis com o que vemos na vida real e, assim, melhorando a análise.

K-Médias

No caso do K-Médias, vemos que a distribuição dos clusters ficou em quantidade bem desbalanceada, com muito mais universidades agrupadas no cluster "0" do que no "1". Dentre as informações presentes na tabela abaixo, alguns atributos que se destacam são "F.Undergrad" e "P.Undergrad", índices que estão relacionados tanto com a distribuição dos alunos dentro da universidade quanto com o tamanho da universidade em si, e que mostra o agrupamento "1" com universidades com quantidades muito maior de alunos do que o agrupamento "0", em ambos os casos há uma quantidade consideravelmente maior de alunos que estudam em tempo integral do que apenas em um turno. Além disso, os aspectos de custo como "Books" e "Personal" mostram um maior gasto por parte dos alunos presentes no agrupamento "1", enquanto o "Outstate" mostra uma mensalidade menor nesse mesmo grupo.

Podemos justificar o caso do "Outstate" devido a tendência de bolsas para reduzir a mensalidade para alunos vindos de outros estados, visto os custos extras já relacionados com tal mudança. Existe também uma tendência maior das universidades privadas no exterior de fornecerem mais bolsa para alunos de fora do estado, enquanto universidades públicas focam suas bolsas mais para sua comunidade, ou seja, seu próprio estado. Dessa forma, acreditamos que o cluster "0" adquiriu mais características de universidades públicas, sendo essa a classificação escolhida.

In [ ]:
```
data_att_means = data_att_no_norm
data_att_means['Cluster'] = kmeans.labels_
data_att_means.head()
```

Out[ ]:

	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	PhD	S.F.Ratio	perc.a
0	2885	537	7440	3300	450	2200	70	18.1	
1	2683	1227	12280	6450	750	1500	29	12.2	
2	1036	99	11250	3750	400	1165	53	12.9	
3	510	63	12960	5450	450	875	92	7.7	
4	249	869	7560	4120	800	1500	76	11.9	

In [ ]:
```
data_att_means_0 = data_att_means[data_att_means['Cluster'] == 0]
data_att_means_1 = data_att_means[data_att_means['Cluster'] == 1]
```

In [ ]:
```
data_att_means_0.describe()
```

Out[ ]:

	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	
count	665.000000	665.000000	665.000000	665.000000	665.000000	665.000000	665
mean	2117.183459	443.598496	10894.508271	4411.306767	544.255639	1246.249624	71
std	1875.817720	494.976885	4018.860847	1105.929734	170.594401	623.690153	16
min	139.000000	1.000000	2580.000000	1780.000000	110.000000	250.000000	8
25%	925.000000	78.000000	7850.000000	3620.000000	450.000000	800.000000	61
50%	1419.000000	266.000000	10520.000000	4270.000000	500.000000	1125.000000	73
75%	2675.000000	639.000000	13380.000000	5150.000000	600.000000	1500.000000	84
max	11278.000000	3144.000000	21700.000000	8124.000000	2340.000000	6800.000000	100

In [ ]:
```
data_att_means_1.describe()
```

Out[ ]:

	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	
count	110.000000	110.000000	110.000000	110.000000	110.000000	110.000000	110
mean	13315.318182	3358.436364	7757.345455	4039.245455	579.445455	1925.209091	81
std	6111.164468	2764.049577	2831.048535	987.894984	125.162030	695.672074	8
min	690.000000	604.000000	2340.000000	2325.000000	96.000000	600.000000	48
25%	9413.250000	1663.250000	5934.250000	3346.000000	500.000000	1434.500000	77
50%	12753.500000	2661.000000	7314.000000	3962.000000	597.500000	1917.000000	83
75%	16133.750000	4041.250000	8894.250000	4531.000000	650.000000	2296.500000	88
max	31643.000000	21836.000000	18420.000000	7425.000000	860.000000	4288.000000	96

K-Medóides

Desta vez os agrupamentos ficaram melhor distribuídos, embora não tanto. Olhando para os mesmos atributos que vimos anteriormente, vemos as mesmas tendências observadas no K-Médias porém com menos extremos, principalmente devido a distribuição mais similar de elementos entre os clusters. Entretanto, a classificação foi a inversa do K-Médias.

In [ ]:
```
data_att_medoids = data_att_no_norm
data_att_medoids['Cluster'] = kmedoids.labels_
data_att_medoids.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 775 entries, 0 to 776
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   F.Undergrad  775 non-null    int64  
 1   P.Undergrad  775 non-null    int64  
 2   Outstate     775 non-null    int64  
 3   Room.Board   775 non-null    int64  
 4   Books        775 non-null    int64  
 5   Personal     775 non-null    int64  
 6   PhD          775 non-null    int64  
 7   S.F.Ratio    775 non-null    float64
 8   perc.alumni  775 non-null    int64  
 9   Expend       775 non-null    int64  
 10  Grad.Rate    775 non-null    int64  
 11  Cluster      775 non-null    int64  
dtypes: float64(1), int64(11)
memory usage: 78.7 KB
```

In [ ]:
```
data_att_medoids_0 = data_att_medoids[data_att_medoids['Cluster'] == 0]
data_att_medoids_1 = data_att_medoids[data_att_medoids['Cluster'] == 1]
```

In [ ]:
```
data_att_medoids_0.describe()
```

Out[ ]:

	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	
count	191.000000	191.000000	191.000000	191.000000	191.000000	191.000000	19
mean	9956.947644	2459.099476	7746.481675	4051.235602	575.235602	1763.240838	7
std	6263.177240	2373.296878	3008.931777	1081.258439	173.707473	781.403524	1
min	690.000000	71.000000	2340.000000	1780.000000	96.000000	300.000000	3
25%	5273.500000	1206.500000	5782.000000	3298.000000	500.000000	1210.000000	7
50%	8528.000000	1674.000000	7090.000000	3933.000000	570.000000	1700.000000	8
75%	13465.000000	3107.000000	8881.500000	4704.500000	645.000000	2158.000000	8
max	31643.000000	21836.000000	18420.000000	7425.000000	2000.000000	6800.000000	9

In [ ]:
```
data_att_medoids_1.describe()
```

Out[ ]:

	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	
count	584.000000	584.000000	584.000000	584.000000	584.000000	584.000000	584.
mean	1662.388699	333.446918	11333.181507	4458.989726	540.751712	1205.051370	70.
std	1233.984981	363.120071	3919.261596	1084.354562	161.702918	576.401206	17.
min	139.000000	1.000000	2580.000000	1880.000000	225.000000	250.000000	8.
25%	860.000000	65.250000	8594.500000	3670.000000	450.000000	800.000000	60.
50%	1287.500000	208.000000	10924.000000	4324.000000	500.000000	1100.000000	73.
75%	1995.500000	474.250000	13762.500000	5222.500000	600.000000	1500.000000	84.
max	9205.000000	1983.000000	21700.000000	8124.000000	2340.000000	4913.000000	100.

Avaliando previsão

In [ ]:
```
data_att_private_means = data_att_private
data_att_private_medoids = data_att_private
```

In [ ]:
```
# Create converter from "Private" to "Cluster"
def converter_means(cluster):
    if cluster=='Yes':
        return 0
    else:
        return 1

def converter_medoids(cluster):
    if cluster == 'Yes':
        return 0
    else:
        return 1
```

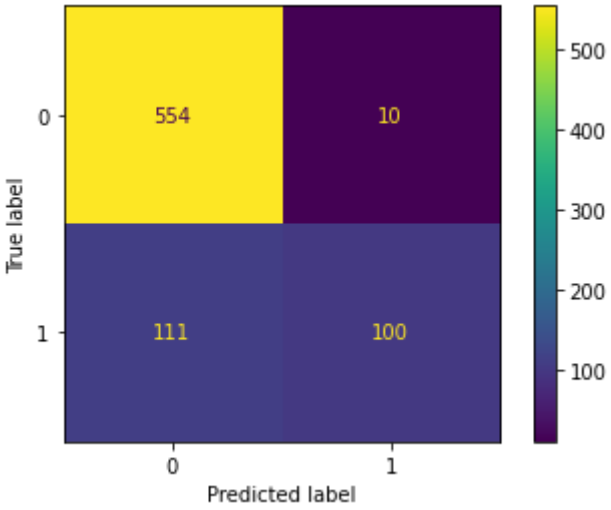
K-Médias

In [ ]:
```
data_att_private_means['Cluster'] = data_att_private_means['Private'].app
data_att_private_means.head()
```

Out[ ]:

	Private	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	PhD	S.F.Rati
0	Yes	2885	537	7440	3300	450	2200	70	18.
1	Yes	2683	1227	12280	6450	750	1500	29	12.
2	Yes	1036	99	11250	3750	400	1165	53	12.
3	Yes	510	63	12960	5450	450	875	92	7.
4	Yes	249	869	7560	4120	800	1500	76	11.

In [ ]:
```
cm = sklearn_metrics.confusion_matrix(data_att_private_means['Cluster'],k
disp = sklearn_metrics.ConfusionMatrixDisplay(cm)
disp.plot()
plt.show()
```



In [ ]:
```
print(sklearn_metrics.classification_report(data_att_private_means['Clust
```

```
              precision    recall  f1-score   support

           0       0.83      0.98      0.90       564
           1       0.91      0.47      0.62       211

    accuracy                           0.84       775
   macro avg       0.87      0.73      0.76       775
weighted avg       0.85      0.84      0.83       775
```

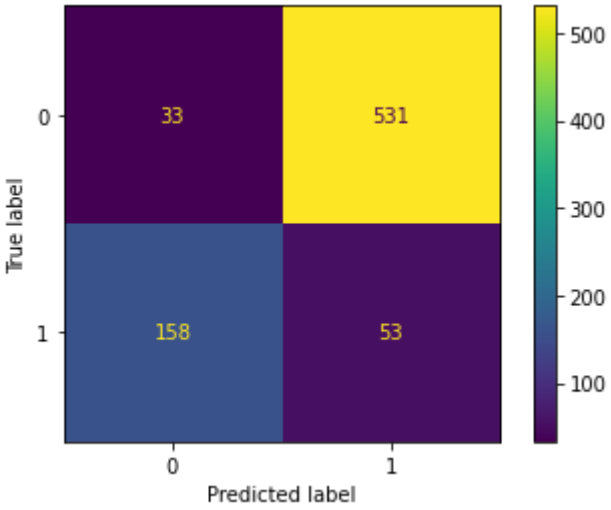
K-Medóides

In [ ]:
```
data_att_private_medoids['Cluster'] = data_att_private_medoids['Private']
data_att_private_medoids.head()
```

Out[ ]:

	Private	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	PhD	S.F.Rati
0	Yes	2885	537	7440	3300	450	2200	70	18.
1	Yes	2683	1227	12280	6450	750	1500	29	12.
2	Yes	1036	99	11250	3750	400	1165	53	12.
3	Yes	510	63	12960	5450	450	875	92	7.
4	Yes	249	869	7560	4120	800	1500	76	11.

In [ ]:
```
cm = sklearn_metrics.confusion_matrix(data_att_private_medoids['Cluster']
disp = sklearn_metrics.ConfusionMatrixDisplay(cm)
disp.plot()
plt.show()
```



In [ ]:
```
print(sklearn_metrics.classification_report(data_att_private_medoids['Clu
```

```
              precision    recall  f1-score   support

           0       0.17      0.06      0.09       564
           1       0.09      0.25      0.13       211

    accuracy                           0.11       775
   macro avg       0.13      0.15      0.11       775
weighted avg       0.15      0.11      0.10       775
```