COMANDOS PYTHON NO GOOGLE COLABORATORY

Aplicação do algoritmo K-modes Inicialmente, os dados coletados na aplicação do questionário foram importados para o Google Colaboratory, por meio dos seguintes comandos:

```
import pandas as pd
questionario = pd.read_csv
('respostas_proc.csv',sep=',')
```

A etapa de pré-processamento resultou em um conjunto de dados no formato matricial, com dimensões de 90 linhas por 36 colunas. Para verificar se nenhuma coluna apresentou dados faltantes, utilizamos o seguinte comando:

```
quest.isnull().sum()
```

Ainda na etapa de pré-processamento, realizamos a exclusão temporária da coluna contendo informações relacionadas a utilização de projetos FLOSS para não influenciar na distribuição dos professores nos *clusters*. Os seguintes comandos foram utilizados:

```
usouFLOSS = questionario['f_usou_floss']
questionario.drop(columns='f_usou_floss',
inplace= True)
```

Para a instalação, atualização e importação da biblioteca K-modes no ambiente Google Colaboratory foram utilizados os seguintes comandos:

```
pip install kmodes
pip install    upgrade kmodes
from kmodes.kmodes import KModes
```

Para aplicação do Método do Cotovelo foram utilizados os seguintes comandos:

```
import matplotlib.pyplot as plt

cost = []
K = range(1,10)
for num_clusters in list(K):
    kmode = KModes(n_clusters=num_clusters
    n_init = 5, verbose=1)
    kmode.fit_predict(questionario)
    cost.append(kmode.cost_)

plt.plot(K, cost, 'bx ')
plt.xlabel('k␣clusters')
plt.ylabel('Cost')
plt.title('Elbow␣Method␣For␣Optimal␣k')
plt.show()
```

Em seguida, o modelo k-modes foi construído. A construção e o treinamento do modelo de aprendizado no ambiente Google Colaboratory foi realizada por meio dos seguintes comandos:

```
km = KModes(n_clusters=2,
init='Huang', n_init=5)
cluster_labels = km.fit_predict(questionario)
```

Após a execução do algoritmo K-modes, foi possível identificar os centróides dos dois *clusters* gerados e os representantes de cada *cluster*, por meio dos seguintes comandos:

```
print(km.cluster_centroids_)
clusters
km.cluster_centroids_
```

Por fim, adicionamos as informações relacionadas ao uso de projetos FLOSS que tinham sido excluídas temporariamente e o *cluster* correspondente a cada professor, por meio dos seguintes comandos:

```
dataset = questionario
dataset['cluster'] = clusters
dataset['target'] = target
```

Aplicação do algoritmo Árvore de Decisão Inicialmente realizamos o tratamento e a transformação dos resultados obtidos a partir da aplicação do algoritmo K-modes, em que foi gerada uma nova coluna informando o *cluster* que cada respondente do questionário foi agrupado. Considerando a natureza dos dados categóricos, o conjunto de dados foi transformado em uma representação vetorial, de 0's e 1's. Esta operação foi automatizada com o uso da biblioteca pandas por meio dos seguintes comandos:

```
preditores = pd.get_dummies(questionario)
```

Uma vez encontrada a solução baseada no conjunto de treino, o modelo de classificação construído avaliou seu desempenho a partir do conjunto de dados de testes. Para tal procedimento, os comandos a seguir foram utilizados:

```
from sklearn.model_selection
import train_test_split

X_train, X_test, y_train,
    y_test = train_test_split(feature,
    target)
```

Em seguida, realizamos a importação da biblioteca *sklearn.tree*, construímos o modelo de aprendizado e realizamos o treino do modelo com a parte segmentada do conjunto de dados destinados para teste. Para tanto, foram utilizados os seguintes comandos:

```
from sklearn.tree
import DecisionTreeClassifier

tree = DecisionTreeClassifier
(criterion='gini',
min_samples_leaf=5,
min_samples_split=9,
max_depth=3,
random_state=20)

tree.fit(X_train, y_train)
```

O modelo gerado foi avaliado na tentativa de verificar a capacidade de generalização do modelo construído. Este processo foi automatizado pelos códigos descritos a seguir e a partir deles foi

possível verificar métricas como acurácia, cobertura (*recall*) e precisão.

```
from sklearn.model_selection
import cross_val_score

scores = cross_val_score(tree,
    features.values,
    target, scoring='accuracy', cv=30)

recall = cross_val_score(tree,
    features.values,
    target,scoring='recall', cv=30)

precision = cross_val_score(tree,
    features.values,
    target, cv=5, scoring='precision')

f1 = cross_val_score(tree,
    features.values,
    target, cv=5, scoring='f1')
```

Após a avaliação do modelo de aprendizado construído, utilizamos o recurso *Feature Importance* da Árvore de Decisão. O *Feature Importance* possibilita a visualização dos atributos que melhor determinam a classificação. Em nosso estudo, foram extraídas as respostas do questionário que mais influenciaram na classificação quanto ao uso de projetos FLOSS na EES. Para isso, foram utilizados os seguintes comandos:

```
for feature,importancia in zip(features.columns,
tree.feature_importances_):
    print( {}:{} .format(feature, importancia))
```