**Anexo Único**

**Arquivo python desenvolvido através do framework Jupyter Notebook/Pandas para análise da base de dados**

#!/usr/bin/env python

# coding: utf-8

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

from sklearn.cluster import KMeans

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from matplotlib import pyplot as plt

import seaborn as sns

from ipywidgets import interactive

from IPython.display import SVG,display

from itertools import cycle

import numpy as np

from random import sample

df = pd.read\_csv('../Sprint 1/keys.csv',';')

df

df.info()

df["MES\_PLANTIO"]=df['DATA\_PLANTIO'].apply(lambda x: int(x[3:5]))

df["MES\_ATIVIDADE\_REALIZ"]=df['DATA\_ATIVIDADE\_REALIZ'].apply(lambda x: int(x[3:5]))

df.describe()

**### Retirando colunas**

df = df.drop(columns=["Z"

,"SIGLA\_FAZENDA"

,"SIGLA\_TALHAO"

,"DATA\_PLANTIO"

,"ANO\_PLANTIO"

,"COD\_ATIVIDADE"

,"MODO\_DE\_OPERACAO"

,"MODO\_DE\_APLICACAO"

,"MODO\_DE\_ACAO"

,"QTDE\_REALIZADA\_ATIVIDADE"

,"DATA\_ATIVIDADE\_REALIZ"

,"ANO\_ATIVIDADE\_REALIZ"

,"DIAS\_REF\_PLANTIO\_REALIZ"

,"ROTACAO"

,"CICLO"

])

df.info()

**### Funções**

**### Matriz de correlação**

**###### plota uma matriz de correlação e retorna um dataframe das correlações**

def matrizCorrelacao(df):

n=df

for col in df.keys():

n[col],\_ = converterDadosMatriz(n, col)

corrmat = n.corr()

fig = plt.figure(figsize = (50,50))

sns.heatmap(corrmat,

vmax = 1,

square = True,

annot = True,

vmin = -1

)

plt.show()

fig.savefig("matriz.png")

return corrmat

**###### Ajuda a função matrizCorrelacao a tranformar os dados em numéricos para que se possa calcular a correlação**

def converterDadosMatriz(n, col):

return pd.factorize(n[col])

**### Funções de padronização de dados**

**###### Esta função cria a base de dados com os dados que precisa, porém o target precisa estar por último na relação de colunas analisadas**

def criarBaseDados(df, colunas):

df1=df[colunas]

return df1

**###### Converte os dados escrito (qualitativo nominal) para número (quantitativo discreto) do dataframe, e a função retorna um dicionário com dois itens sendo "target" que contém valores escritos e "valores" que contém os números, se relacionam pela ordem.**

def converterDados(nome, df1):

dados={}

dados["target"]= df1[nome].unique()

df1[nome],\_=pd.factorize(df1[nome])

dados["valores"]= df1[nome].unique()

for x in range(len(dados["target"])):

print("target: %s = %s" %(dados["target"][x],dados["valores"][x]))

return dados

**#### Converter números com padrão europeu usado no Brasil para padrão americano reconhecido pelo python**

def converterDadosB(df1, coluna):

df1[coluna]=df1[coluna].apply(lambda x: str(x).replace(",","."))

df1[coluna]= df1[coluna].astype("float64")

**### Funções que criam testes e treinamentos**

**##### Função que cria teste e treinamento supervisionado**

**###### Cria os dados de teste e de treinamento com a porcentagem de 30% para teste com um target**

def criarTesteTreinamento(target, novo):

dados={}

**##Separa os dados do target**

feature\_cols = novo.keys()

x\_train = novo[feature\_cols[:-1]]

y\_train = novo[target]

**## Cria ambiente de teste**

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

x\_train, y\_train,test\_size=0.3,random\_state=1)

dados["x\_train"]=x\_train

dados["y\_train"]=y\_train

dados["x\_test"]=x\_test

dados["y\_test"]=y\_test

return dados

**#### Função que cria teste e treinamento não supervisionado**

**###### Cria os dados de teste e de treinamento com a porcentagem de 30% para teste sem um target**

def criarTestTrainUnsupervised(df):

dados={}

x\_train = df

## criar ambiente de teste

x\_train, x\_test = train\_test\_split(

x\_train ,test\_size=0.3,random\_state=1)

dados["x\_train"]=x\_train

dados["x\_test"]=x\_test

return dados

**### Função da árvore de decisão**

**##### Funções não automatizadas**

**###### Treina a árvore e plota os resultados**

def treinarPlotar(arvore, dados, target):

#treina

arvore = arvore.fit(dados["x\_train"],dados["y\_train"])

#plota

fig = plt.figure(figsize=(50,30))

\_= tree.plot\_tree(arvore, feature\_names=dados["x\_train"].columns,class\_names=target,

filled=True, rounded=True)

fig.savefig("decisiontree.svg", format='svg',dpi=1200)

**###### Verifica a árvore e mostra a acurácia (quanto maior melhor) e “amostra erroneamente” (quanto menor melhor)**

def verificarArvore(dados, arvore):

y\_pred=arvore.predict(dados["x\_test"])

count=(dados["y\_test"]!=y\_pred).sum()

print('Amostras erroneamente classificadas: {}'.format(count))

#Taxa de precisão

accuracy = metrics.accuracy\_score(dados["y\_test"], y\_pred)

print('Acuracia: {:.2f}'.format(accuracy))

**### Função automatizada da árvore**

**###### Classe que engloba as funções de árvore**

class Arvore:

def \_\_init\_\_(self, dados, target):

self.dados= dados

self.target = target

def criarArvore(self, crit, split,depth, split\_folha, leaf\_folha):

arvore = DecisionTreeClassifier(

random\_state = 0

,criterion = crit

,splitter = split

,max\_depth = depth

,min\_samples\_split=split\_folha

,min\_samples\_leaf=leaf\_folha

)

#treina

arvore = arvore.fit(self.dados["x\_train"],self.dados["y\_train"])

#plota

fig = plt.figure(figsize=(50,30))

\_= tree.plot\_tree(arvore, feature\_names=self.dados["x\_train"].columns,class\_names=self.target,

filled=True, rounded=True)

fig.savefig("decisiontree.svg", format='svg',dpi=1200)

y\_pred=arvore.predict(self.dados["x\_test"])

count=(self.dados["y\_test"]!=y\_pred).sum()

print('Amostras erroneamente classificadas: {}'.format(count))

#Taxa de precisão

accuracy = metrics.accuracy\_score(self.dados["y\_test"], y\_pred)

print('Acuracia: {:.2f}'.format(accuracy))

**###### Função que faz o usuário interagir com o algoritmo e parâmetros**

def interador(dados, target):

a= Arvore(dados, target)

inter=interactive(a.criarArvore

, crit = ["entropy","gini"]

, split = ["best", "random"]

, depth=[None,1,2,3,4,5,10,20,30]

, split\_folha=(1,50)

, leaf\_folha=(1,40))

display(inter)

**### Função do algoritmo de clusterização**

**##### Calcula a inércia dos dados ao número de clusters**

def calcularInercia(dados1):

#calcula a inercia para cada cluster

inercias=[]

maxclusters=11

for i in range(1,maxclusters):

km = KMeans(n\_clusters=i, init="k-means++")

km.fit(dados1["x\_train"])

print(i, km.inertia\_)

inercias.append(km.inertia\_)

#plota a inercia para cada cluster

plt.plot(range(1, maxclusters),inercias, marker="o")

plt.title("O metodo Elbow")

plt.xlabel("Quant de clusters")

plt.ylabel("Inercia")

plt.show()

**##### Cria os clusters, treina e retorna um dicionário com os seguintes dados: Label (refere-se ao número no cluster para cada linha), centróide (centro de referência do cluster, donde se distanciam todos os pontos ) e o Kmeans ( armazena o algoritmo já treinado).**

def criarCluster(n\_cluster, dados1):

km= {}

#configuração do cluster

kmeans = KMeans(n\_clusters=n\_cluster, init="k-means++", n\_init=30)

#treina o algoritimo

kmeans.fit(dados1["x\_train"])

# cria um uma celuda na dataframe que categoriza as linhas referentes a cada cluster

km["Label"] = kmeans.labels\_

km["centroid"] = kmeans.cluster\_centers\_

km["Kmeans"] = kmeans

return km

**##### Mostra a média do clusters com um dataFrame**

def mostrarMediaCluster(df):

return df.groupby("Clusters").mean()

**##### Plota as médias de cada cluster**

def plotagemBarraClusters(df):

clusters= df["Clusters"].unique()

print(clusters)

sd = df.groupby("Clusters").mean()

sd.plot.bar(figsize=(16, 10), ylabel="Media", title="Medias dos clusters")

**##### Plota a localização dos dados ao centróides ( o centróides são vermelhos)**

def plotaCirculo(colunas, dados1, k):

kmeans=k

def plotar(coluna\_1, coluna\_2):

plt.scatter(dados1[coluna\_1], dados1[coluna\_2], s=50, c=kmeans.labels\_)

plt.scatter(kmeans.cluster\_centers\_[:,colunas.index(coluna\_1)],

kmeans.cluster\_centers\_[:,colunas.index(coluna\_2)],

s=20, c='red')

plt.title("Cluster centroide customizado")

plt.xlabel(coluna\_1)

plt.ylabel(coluna\_2)

plt.show()

controler= interactive(plotar

, coluna\_1=colunas

, coluna\_2=colunas)

display(controler)

**##### Função que prediz os dados correspondentes às características de um certo cluster**

def predizer(k, colunas):

kmeans = k

cs = kmeans.predict([colunas])

for c in cs:

print("O cluster correspondente é o",(c+1))

**## Matriz de correlação**

print("Matriz de correlação")

colunas =["AREA\_TALHAO",

"ESPACAMENTO",

"SIGLA\_MAT\_GEN",

"MES\_PLANTIO",

"DESC\_GRP\_ATIVIDADE",

"NOME\_ATIVIDADE",

"% AREA",

"MES\_ATIVIDADE\_REALIZ",

"IDADE\_TALHAO\_ATIVIDADE",

"IDADE\_TALHAO\_INVENTARIO",

"AREA\_INVENTARIO",

"CHOVEU?",

"QUANTO?",

"C.DIA ANT?",

"C.DIA POST?",

"C.3 DIAS A?",

"C.3 DIAS D?",

"ARM/CAD",

"IMA\_TALHAO",

"VOLUME\_HA\_TALHAO",

"FUSTES\_HA",

"DIAM\_MED\_TALHAO",

"ALTURA\_MED\_TALHAO",

"MORTES\_PERC",

"FALHA\_PERC",

"QUEBRADAS\_PERC",

"TOMBADAS\_PERC"]

matriz = criarBaseDados(df,colunas)

corr= matrizCorrelacao(matriz)

corr

**### Verificando quais dados são o melhor**

corr.query("FALHA\_PERC !=1 & FALHA\_PERC > 0.39 | FALHA\_PERC < -0.39")

**# Pergunta: A CHUVA INTERFERE NO PLANTIO?**

**#### Para responder esta pergunta vou criar três amostras que representam a população, porém vou precisar converter a base dos dados para assim criar as amostras. Isto precisou ser feito para que não ocorra nenhuma diferença das amostras e nas análises**

**### Criando base de dados com os campos que preciso**

colunas = ["CODTALHAO","QUANTO?","CHOVEU?",

"SIGLA\_MAT\_GEN","ESPACAMENTO","ALTURA\_MED\_TALHAO","FALHA\_PERC","ARM/CAD","MES\_PLANTIO"]

df1= criarBaseDados(df, colunas)

print("Transformando dados\n")

### Transformando os dados

print("\nChoveu")

valor2 = converterDados("CHOVEU?",df1 )

print("\nSIGLA\_MAT\_GEN")

valor3 = converterDados("SIGLA\_MAT\_GEN",df1)

print("\nESPACAMENTO")

valor4 = converterDados("ESPACAMENTO",df1)

print("\nALTURA\_MED\_TALHAO")

converterDadosB(df1, "ALTURA\_MED\_TALHAO")

print("\nFALHA\_PERC")

converterDadosB(df1, "FALHA\_PERC")

**#### Informações sobre os campos referentes à base de dados**

print("INFORMACÕES DA BASE DE DADOS\n")

df1.describe()

**#### Cria amostras**

amostra1 = sample(range(df1["CODTALHAO"].min(), df1["CODTALHAO"].max()), 250)

amostra1

amostra2 = sample(range(df1["CODTALHAO"].min(), df1["CODTALHAO"].max()), 250)

amostra2

amostra3 = sample(range(df1["CODTALHAO"].min(), df1["CODTALHAO"].max()), 250)

amostra3

**### Criando três amostras**

colunas = ["QUANTO?","CHOVEU?","SIGLA\_MAT\_GEN","ESPACAMENTO","ALTURA\_MED\_TALHAO","FALHA\_PERC","ARM/CAD","MES\_PLANTIO"]

#### Amostra 1

print("Criando amostra 1")

amostra1= df1.query("CODTALHAO in %s" %amostra1)

amostra1= criarBaseDados(amostra1, colunas)

amostra1 = amostra1.sort\_values(by='MES\_PLANTIO')

amostra1

**#### Infomações da 1° amostra**

amostra1.describe()

**#### Amostra 2**

print("Criando amostra 2")

amostra2= df1.query("CODTALHAO in %s" %amostra2)

amostra2= criarBaseDados(amostra2, colunas)

amostra2 = amostra2.sort\_values(by='MES\_PLANTIO')

amostra2

**#### Informações da 2° amostra**

amostra2.describe()

**#### Amostra 3**

print("Criando amostra 3")

amostra3= df1.query("CODTALHAO in %s" %amostra3)

amostra3= criarBaseDados(amostra3, colunas)

amostra3 = amostra3.sort\_values(by='MES\_PLANTIO')

amostra3

**#### Infomações da 3° amostra**

amostra3.describe()

**## Algoritmo de clusterização**

**#### Nesta etapa será criado o algoritmos para verificar a proximidade das amostras com elas mesma**

print("Amostra 1 algoritmo\n")

**### Cria algoritmo da amostra1**

dados1=criarTestTrainUnsupervised(amostra1)

dados1["x\_train"]

**#### Calcula a inércia**

calcularInercia(dados1)

**#### Criar e treina algoritimo**

clusters1 = criarCluster(4,dados1)

dados1["x\_train"]["Clusters"]= clusters1["Label"]

dados1["x\_train"]

mostrarMediaCluster(dados1["x\_train"])

plotagemBarraClusters(dados1["x\_train"])

plotaCirculo(colunas, dados1["x\_train"], clusters1["Kmeans"])

**##### Na comparação das colunas altura x mes\_plantio x falha, analisando com base no mes\_plantio, percebe-se que a falha é vista com maior incidência nos meses de maio a agosto, a altura alcança maiores medidas quando plantio acontece nos meses finais do ano**

print("Amostra 2 algoritmo\n")

**### Criando algoritmo da amostra2**

dados2=criarTestTrainUnsupervised(amostra2)

dados2["x\_train"]

calcularInercia(dados2)

clusters2 = criarCluster(4,dados2)

dados2["x\_train"]["Clusters"]= clusters2["Label"]

dados2["x\_train"]

mostrarMediaCluster(dados2["x\_train"])

plotagemBarraClusters(dados2["x\_train"])

plotaCirculo(colunas, dados2["x\_train"], clusters2["Kmeans"])

print("Amostra 3 algoritmo\n")

### Criando algoritimo da amostra3

dados3=criarTestTrainUnsupervised(amostra3)

dados3["x\_train"]

calcularInercia(dados3)

clusters3 = criarCluster(4,dados3)

dados3["x\_train"]["Clusters"]= clusters3["Label"]

dados3["x\_train"]

mostrarMediaCluster(dados3["x\_train"])

plotagemBarraClusters(dados3["x\_train"])

plotaCirculo(colunas, dados3["x\_train"], clusters3["Kmeans"])

**### Divididos em quatro clusters sendo que:**

**# - um cluster mostra que o período dos meses são no intervalo de set a dez com uma altura do fuste elevada**

**# - um dos cluster mostra uma alta quantidade de falhas e no período entre os meses de maio a agosto**

**# - outro cluster mostra a menor altura dos eucaliptos, porém com menor falha, sendo que pode apresenta também uma variação das amostras de material genético significativas**

**# - Noutro cluster de maior altura apresenta data de plantio de Janeiro a Maio**

**## Conclusão das amostras**

**# ### Ao analisar as amostra foi constatado que o período mensal afeta a produtividade e apresenta falha alta nos meses de maio a agosto**

**# ### Os períodos com a árvores mais altas e as falhas mais baixas quando plantadas no meses de janeiro a abril e setembro a dezembro, sendo que as árvores mais alta são as plantadas no meses de setembro a dezembro**

**# ### Os dados da população não são muito próximos, então as análise da cada amostra precisou ser feita com quase 50% da população para que a análise fosse feita com base numa população representativa dos dados**.

**## Análise do cluster com o periodo mensal maior**

**### Criando árvore de decisão**

print("UTILIZANDO O CLUSTER COM A MEDIA DOS MESES SENDO PROXIMO 10 E FALHA BAIXA\n")

clusterM= dados3["x\_train"].query("Clusters=='3'")

clusterM=clusterM.drop(columns=["Clusters"])

clusterM

clusterM.describe()

**#### Cria teste e treinamento**

dados = criarTesteTreinamento("MES\_PLANTIO",clusterM)

dados["x\_train"]

**#### Converte inteiro para String**

clusterM["MES\_PLANTIO"]=clusterM["MES\_PLANTIO"].astype("string")

print("Arvore automatica\n")

**#### Usando árvore automática**

interador(dados, clusterM["MES\_PLANTIO"].unique())

print("Criando arvore manual\n")

**### Cria árvore manual**

arvore = DecisionTreeClassifier(criterion='entropy',

min\_samples\_leaf=7,

min\_samples\_split=10,

max\_depth=10

)

arvore

treinarPlotar(arvore, dados, clusterM["MES\_PLANTIO"].unique())

**### Verifica árvore**

verificarArvore(dados, arvore)

print("\nTeste da arvore manual")

**## Testando**

**### Para verificar o funcionamento da árvore foram utilizados alguns dados para teste**

**#### Os algoritmos preveem qual é o mês mais vantajoso para o plantio do eucalipto para a uma produção mais eficiente**

print("\nDados não target")

print(dados["x\_test"])

print("\nDado target")

print(dados["y\_test"])

print("\nValor do teste")

W\_test = np.array([[0,0,1,0,25.05,0.69,2]])

p = arvore.predict(W\_test)

for x in p:

print("mes %s" %(x))