Projeto de Mineração Estatística de Dados

Alunos:

- Dayvison
- Ednaldo
- Elayni
- Thiago
- Ítalo

Biblioetecas utilizadas

In [7]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn import preprocessing
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import classification_report, plot_roc_curve
from sklearn.naive_bayes import GaussianNB

sns.set_style('whitegrid')
```

Integração

Carregamos as bases da paraíba e de pernambuco e tiramos uma coluna desnecessária

In [8]:
```
df_pb = pd.read_csv('covid19-pb.csv')
```

In [9]:
```
df_pe = pd.read_csv('covid19-pe.csv')
```

In [10]:
```
df_pb = df_pb.drop(columns=['Unnamed: 0'])
```

In [11]:
```
df_pe = df_pe.drop(columns=['Unnamed: 0'])
```

Removendo linhas com valores faltantes para as cidades

In [12]:
```
df_pb = df_pb.dropna(subset=['city']).reset_index().drop(columns=['index'])
df_pe = df_pe.dropna(subset=['city']).reset_index().drop(columns=['index'])
```

Aqui fazemos a integração dos dados

In [13]:
```
df = pd.concat([df_pb, df_pe], axis=0).reset_index().drop(columns=['index'])
```

In [14]:
```
df
```

Out[14]:

	city	city_ibge_code	date	epidemiological_week	estimated_population	estimated_population_2019	is_l
0	João Pessoa	2507507.0	2020-03-18	202012	817511.0	809015.0	Fa
1	João Pessoa	2507507.0	2020-03-19	202012	817511.0	809015.0	Fa
2	João Pessoa	2507507.0	2020-03-20	202012	817511.0	809015.0	Fa
3	João Pessoa	2507507.0	2020-03-21	202012	817511.0	809015.0	Fa
4	João Pessoa	2507507.0	2020-03-22	202013	817511.0	809015.0	Fa
...	...	...	...	...	...	...	
284135	Vicência	2616308.0	2022-03-27	202213	32772.0	32643.0	Fa
284136	Vitória de Santo Antão	2616407.0	2022-03-27	202213	139583.0	138757.0	Fa
284137	Xexéu	2616506.0	2022-03-27	202213	14757.0	14725.0	Fa
284138	Água Preta	2600401.0	2022-03-27	202213	37082.0	36771.0	Fa
284139	Águas Belas	2600500.0	2022-03-27	202213	43686.0	43443.0	Fa

284140 rows × 18 columns

Mudamos o tipo da data para um do pandas, usando o to_datetime

In [15]:
```
df['date'] = pd.to_datetime(df['date'])
```

Ordenamos os dados pela data

In [16]:
```
df = df.sort_values('date').reset_index().drop(columns=['index'])
```

Limpeza

In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284140 entries, 0 to 284139
Data columns (total 18 columns):
 #   Column                                          Non-Null Count   Dtype
---  ------                                          --------------   -----
 0   city                                            284140 non-null  object
 1   city_ibge_code                                  282748 non-null  float64
 2   date                                            284140 non-null  datetime64[ns]
 3   epidemiological_week                            284140 non-null  int64
 4   estimated_population                            282748 non-null  float64
 5   estimated_population_2019                       282748 non-null  float64
 6   is_last                                         284140 non-null  bool
 7   is_repeated                                     284140 non-null  bool
```

```
 8   last_available_confirmed                       284140 non-null  int64
 9   last_available_confirmed_per_100k_inhabitants  282486 non-null  float64
 10  last_available_date                            284140 non-null  object
 11  last_available_death_rate                      284140 non-null  float64
 12  last_available_deaths                          284140 non-null  int64
 13  order_for_place                                284140 non-null  int64
 14  place_type                                     284140 non-null  object
 15  state                                          284140 non-null  object
 16  new_confirmed                                  284140 non-null  int64
 17  new_deaths                                     284140 non-null  int64
dtypes: bool(2), datetime64[ns](1), float64(5), int64(6), object(4)
memory usage: 35.2+ MB
```

Com base na info acima, escolhemos algumas colunas que não nos ajudariam para futuras análises

In [18]:
```
columns_to_drop = ['city_ibge_code', 'place_type', 'is_last', 'is_repeated','estimated_pop
```

In [19]:
```
df = df.drop(columns=columns_to_drop)
```

Removemos linhas onde a cidade era 'Importados/Indefinidos'

In [20]:
```
df = df.drop(df[df['city'] == 'Importados/Indefinidos'].index).reset_index().drop(columns=
```

In [21]:
```
df.isnull().sum()
```

Out[21]:
```
city                                               0
date                                               0
epidemiological_week                               0
estimated_population                               0
last_available_confirmed                           0
last_available_confirmed_per_100k_inhabitants    262
last_available_death_rate                          0
last_available_deaths                              0
order_for_place                                    0
state                                              0
new_confirmed                                      0
new_deaths                                         0
dtype: int64
```

Aqui preenchemos os valores faltantes em 'last_available_confirmed_per_100k_inhabitants' com sua mediana

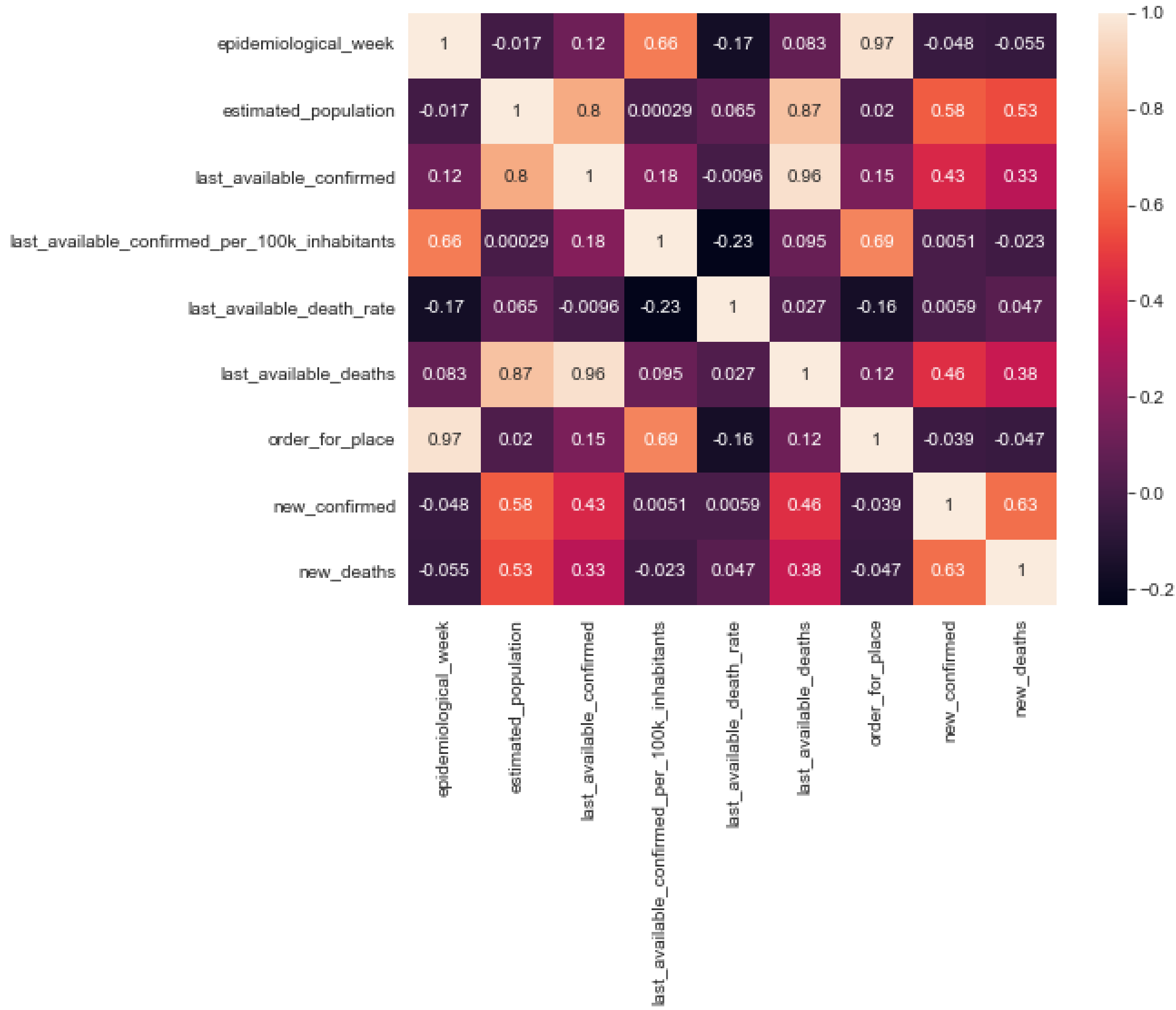
In [22]:
```
df = df.fillna(value={'last_available_confirmed_per_100k_inhabitants': df['last_available_
```

In [23]:
```
df.isnull().sum()
```

Out[23]:
```
city                                             0
date                                             0
epidemiological_week                             0
estimated_population                             0
last_available_confirmed                         0
last_available_confirmed_per_100k_inhabitants    0
last_available_death_rate                        0
last_available_deaths                            0
order_for_place                                  0
state                                            0
new_confirmed                                    0
new_deaths                                       0
dtype: int64
```

In [24]:
```python
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot=True)
```

Out[24]: <AxesSubplot:>



Olhamos aqui a correlação e decidimos remover 'order_for_place'

In [25]:
```python
df = df.drop(columns=['order_for_place'])
```

Suavização dos dados

Fizemos uma espécie de binning com a coluna 'epidemiological_week_binning'

In [26]:
```python
df['epidemiological_week_binning'] = df['epidemiological_week'].apply(lambda x: x - 20201(
```

In [27]:
```python
df
```

Out[27]:

	city	date	epidemiological_week	estimated_population	last_available_confirmed	last_available_confirn
0	Recife	2020-03-12	202011	1653461.0	2	
1	Recife	2020-03-13	202011	1653461.0	2	

	city	date	epidemiological_week	estimated_population	last_available_confirmed	last_available_confirn
2	Recife	2020-03-14	202011	1653461.0	6	
3	Recife	2020-03-15	202012	1653461.0	7	
4	Recife	2020-03-16	202012	1653461.0	7	
...	...	...	...	...	...	
282743	Gado Bravo	2022-03-27	202213	8303.0	777	
282744	Guarabira	2022-03-27	202213	59115.0	10003	
282745	Gurinhém	2022-03-27	202213	14127.0	948	
282746	Marcação	2022-03-27	202213	8653.0	876	
282747	Águas Belas	2022-03-27	202213	43686.0	2047	

282748 rows × 12 columns

Tentamos usar a regressão mas não conseguimos ter resultados bons para uma suavização decente, então usamos a função log para suavizar

In [28]:
```
df['last_available_confirmed'] = df['last_available_confirmed'].apply(lambda x: 0 if x==0
df['last_available_confirmed_per_100k_inhabitants'] = df['last_available_confirmed_per_10(
df['last_available_deaths'] = df['last_available_deaths'].apply(lambda x: 0 if x==0 else r
df['state'] = df['state'].apply(lambda x: 1 if x=='PB' else 0)
```

In [29]:
```
df
```

Out[29]:

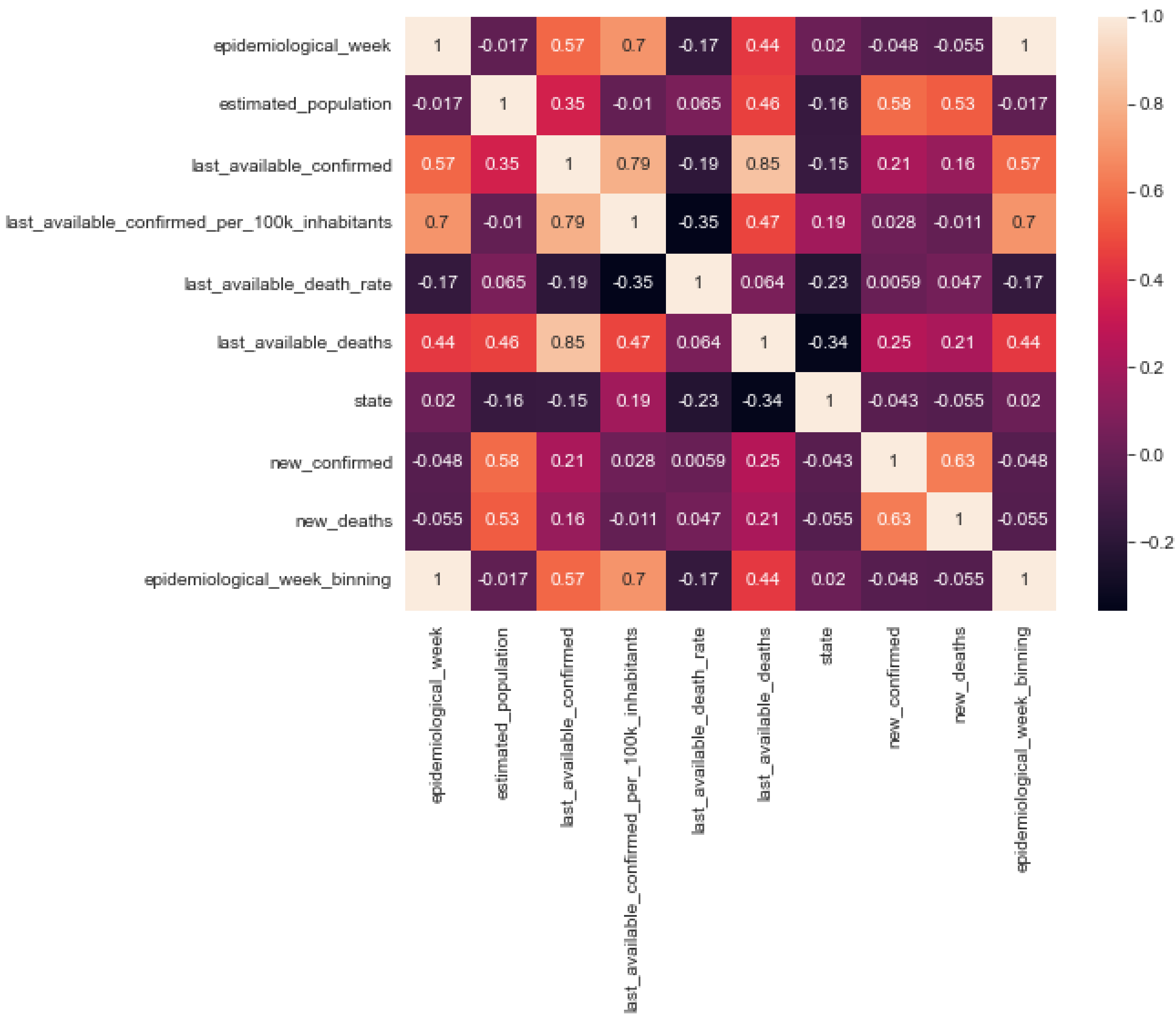
	city	date	epidemiological_week	estimated_population	last_available_confirmed	last_available_confirn
0	Recife	2020-03-12	202011	1653461.0	0.693147	
1	Recife	2020-03-13	202011	1653461.0	0.693147	
2	Recife	2020-03-14	202011	1653461.0	1.791759	
3	Recife	2020-03-15	202012	1653461.0	1.945910	
4	Recife	2020-03-16	202012	1653461.0	1.945910	
...	...	...	...	...	...	
282743	Gado Bravo	2022-03-27	202213	8303.0	6.655440	
282744	Guarabira	2022-03-27	202213	59115.0	9.210640	

	city	date	epidemiological_week	estimated_population	last_available_confirmed	last_available_confirn
282745	Gurinhém	2022-03-27	202213	14127.0	6.854355	
282746	Marcação	2022-03-27	202213	8653.0	6.775366	
282747	Águas Belas	2022-03-27	202213	43686.0	7.624131	

282748 rows × 12 columns

In [30]:
```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot=True)
```

Out[30]: <AxesSubplot:>



Retiramos a coluna original após o binning

In [31]:
```
df = df.drop(columns=['epidemiological_week'])
```

In [32]:
```
df
```

Out[32]:

	city	date	estimated_population	last_available_confirmed	last_available_confirmed_per_100k_inhabitant

	city	date	estimated_population	last_available_confirmed	last_available_confirmed_per_100k_inhabitant
0	Recife	2020-03-12	1653461.0	0.693147	-2.11229
1	Recife	2020-03-13	1653461.0	0.693147	-2.11229
2	Recife	2020-03-14	1653461.0	1.791759	-1.01368
3	Recife	2020-03-15	1653461.0	1.945910	-0.85955
4	Recife	2020-03-16	1653461.0	1.945910	-0.85955
...	...	...	...	...	.
282743	Gado Bravo	2022-03-27	8303.0	6.655440	9.14399
282744	Guarabira	2022-03-27	59115.0	9.210640	9.73632
282745	Gurinhém	2022-03-27	14127.0	6.854355	8.81143
282746	Marcação	2022-03-27	8653.0	6.775366	9.22263
282747	Águas Belas	2022-03-27	43686.0	7.624131	8.45227

282748 rows × 11 columns

Redução de Dados

Usamos o PCA para reduzir a quantidade de colunas e agregamos no dataframe que tínhamos

In [33]:
```
pca = PCA(n_components=6)
X = pca.fit_transform(df[df.columns[2:]])
```

In [34]:
```
pca_df = pd.DataFrame(X, columns=[f'PCA_{i}' for i in range(X.shape[1])])
```

In [35]:
```
df_reduzido = pd.concat([df[['city','date']], pca_df], axis=1)
```

In [36]:
```
df_reduzido
```

Out[36]:

	city	date	PCA_0	PCA_1	PCA_2	PCA_3	PCA_4	PCA_5
0	Recife	2020-03-12	1.618795e+06	-76.302072	-203.880589	18.446737	1.966962	-3.598755
1	Recife	2020-03-13	1.618795e+06	-76.267156	-205.879929	18.436611	1.977415	-3.563989
2	Recife	2020-03-14	1.618795e+06	-76.297444	-201.869802	17.198119	2.816320	-3.639600
3	Recife	2020-03-15	1.618795e+06	-75.240072	-204.849922	17.033631	2.945959	-3.588165
4	Recife	2020-03-16	1.618795e+06	-75.222614	-205.849592	17.028567	2.951186	-3.570782
...	...	...	...	...	...	...	...	...

	city	date	PCA_0	PCA_1	PCA_2	PCA_3	PCA_4	PCA_5
282743	Gado Bravo	2022-03-27	-2.636256e+04	108.323291	1.407786	1.690374	0.361222	0.031445
282744	Guarabira	2022-03-27	2.444944e+04	108.957956	-4.868768	-1.548229	-0.552153	-0.026910
282745	Gurinhém	2022-03-27	-2.053856e+04	108.390912	0.685542	1.385639	-0.233543	0.031481
282746	Marcação	2022-03-27	-2.601256e+04	108.331881	1.365459	1.515506	0.360885	0.032814
282747	Águas Belas	2022-03-27	9.020442e+03	108.727133	-2.974793	0.606933	-1.243732	-0.012953

282748 rows × 8 columns

FP-Tree

No FP Tree deu problema, pois nossos dados não são bons para utilizar isso, eu peguei um código que deixa os dados do jeito que é necessário para utilizar a FP-Tree, com as saídas dos dataframes equivalentes, mas mesmo assim não funcionou.

In [37]:
```
df_teste = df[['city']]
df_teste["incident_count"] = 1
df_table = df_teste.groupby("city").sum().sort_values("incident_count", ascending=False).r
```

```
C:\Users\Dayvison\AppData\Local\Temp/ipykernel_11180/2937121498.py:2: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  df_teste["incident_count"] = 1
```

In [38]:
```
df_table
```

Out[38]:

	city	incident_count
0	Paulista	1403
1	Alagoinha	1401
2	Condado	1395
3	Quixaba	1389
4	Triunfo	1365
...	...	...
396	Mirandiba	639
397	São Domingos	636
398	Monte Horebe	622
399	Poço de José de Moura	621
400	Pedra Branca	621

401 rows × 2 columns

In [39]:
```
transaction = []
for i in range(df_table.shape[0]):
```

```python
    transaction.append([str(df_table.values[i,j]) for j in range(df_table.shape[1])])

# creating the numpy array of the transactions
transaction = np.array(transaction)

# importing the required module
from mlxtend.preprocessing import TransactionEncoder

# initializing the transactionEncoder
te = TransactionEncoder()
te_ary = te.fit(transaction).transform(transaction)
dataset = pd.DataFrame(te_ary, columns=te.columns_)

# dataset after encoded
dataset.head()
```

Out[39]:

	1350	1355	1365	1389	1395	1401	1403	621	622	636	...	Vicência	Vieirópolis	Vista Serrana	Vitória de Santo Antão	Várze
0	False	False	False	False	False	False	True	False	False	False	...	False	False	False	False	Fals
1	False	False	False	False	False	True	False	False	False	False	...	False	False	False	False	Fals
2	False	False	False	False	True	False	False	False	False	False	...	False	False	False	False	Fals
3	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	Fals
4	False	False	True	False	False	False	False	False	False	False	...	False	False	False	False	Fals

5 rows × 493 columns

In [40]:

```python
dataset
```

Out[40]:

	1350	1355	1365	1389	1395	1401	1403	621	622	636	...	Vicência	Vieirópolis	Vista Serrana	Vitória de Santo Antão	Vá
0	False	False	False	False	False	False	True	False	False	False	...	False	False	False	False	F
1	False	False	False	False	False	True	False	False	False	False	...	False	False	False	False	F
2	False	False	False	False	True	False	False	False	False	False	...	False	False	False	False	F
3	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	F
4	False	False	True	False	False	False	False	False	False	False	...	False	False	False	False	F
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
396	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	F
397	False	False	False	False	False	False	False	False	False	True	...	False	False	False	False	F
398	False	False	False	False	False	False	False	False	True	False	...	False	False	False	False	F
399	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	F
400	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	F

401 rows × 493 columns

In [41]:

```python
first30 = df_table["city"].head(30).values
```

```
dataset = dataset.loc[:,first30]

dataset.shape
```

Out[41]:
```
(401, 30)
```

In [42]:
```
from mlxtend.frequent_patterns import fpgrowth

#running the fpgrowth algorithm
res = fpgrowth(dataset ,min_support=0.05, use_colnames=True)

# printing top 10
res
```

Out[42]:

support	itemsets

Clusterização

K-Means

Para o K-Means, focamos em fazer 2 grupos, que seriam os dois estados, para ver se ele conseguiria dizer que entrada de dados é de qual estado.

In [43]:
```
le = preprocessing.LabelEncoder()
city = le.fit_transform(df['city'])
```

In [44]:
```
df['city'] = city
```

In [45]:
```
df.head()
```

Out[45]:

	city	date	estimated_population	last_available_confirmed	last_available_confirmed_per_100k_inhabitants	last_avai
0	281	2020-03-12	1653461.0	0.693147	-2.112295	
1	281	2020-03-13	1653461.0	0.693147	-2.112295	
2	281	2020-03-14	1653461.0	1.791759	-1.013683	
3	281	2020-03-15	1653461.0	1.945910	-0.859556	
4	281	2020-03-16	1653461.0	1.945910	-0.859556	

In [46]:
```
df.corr()['state']
```

Out[46]:
```
city                                              0.008217
estimated_population                             -0.157372
last_available_confirmed                         -0.148714
last_available_confirmed_per_100k_inhabitants     0.190137
last_available_death_rate                        -0.227802
last_available_deaths                            -0.344318
state                                             1.000000
```

```
new_confirmed                                  -0.042711
new_deaths                                     -0.054578
epidemiological_week_binning                    0.019618
Name: state, dtype: float64
```

In [47]:
```
kmeans = KMeans(n_clusters=2)
```

In [48]:
```
pred = kmeans.fit_predict(df.drop(columns=['city','date','state']).values)
```

In [49]:
```
pred = kmeans.fit_predict(df['city'].values.reshape(-1,1))
```

Ele não se saiu muito bem como vemos abaixo

In [50]:
```
np.mean( pred == df['state'] )*100
```

Out[50]:
```
45.90094359641801
```

Árvore de decisão

In [51]:
```
clf = DecisionTreeClassifier(random_state=0, max_depth=5)
```

In [52]:
```
df.head()
```

Out[52]:

	city	date	estimated_population	last_available_confirmed	last_available_confirmed_per_100k_inhabitants	last_avai
0	281	2020-03-12	1653461.0	0.693147	-2.112295	
1	281	2020-03-13	1653461.0	0.693147	-2.112295	
2	281	2020-03-14	1653461.0	1.791759	-1.013683	
3	281	2020-03-15	1653461.0	1.945910	-0.859556	
4	281	2020-03-16	1653461.0	1.945910	-0.859556	

In [53]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop(columns=['date','state','city'
```

In [54]:
```
clf.fit(X_train, y_train)
```

Out[54]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, random_state=0)
```

In [55]:
```
y_pred = clf.predict(X_test)
```

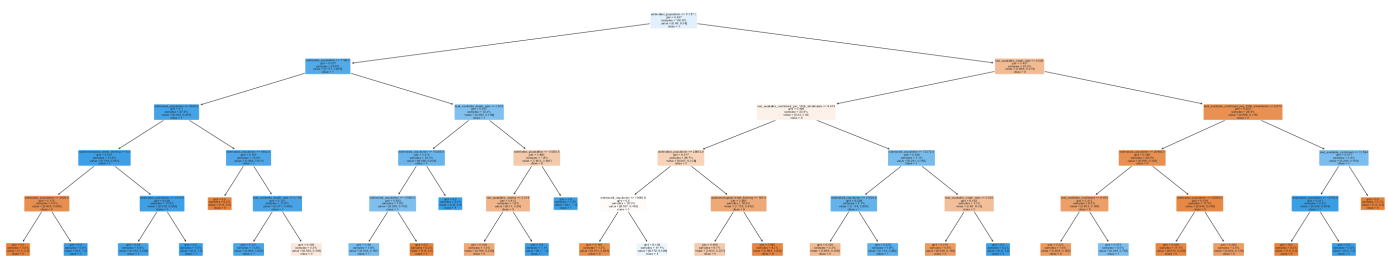
Temos um bom resultado predizendo o estado

In [56]:
```
print(classification_report(y_test, y_pred, target_names=['PE','PB']))
```

```
              precision    recall  f1-score   support

          PE       0.86      0.76      0.81     32575
          PB       0.82      0.89      0.85     38112

    accuracy                           0.83     70687
   macro avg       0.84      0.83      0.83     70687
weighted avg       0.84      0.83      0.83     70687
```

In [57]:

```python
class_label = ['0','1']

plt.figure(figsize=(50,10))
plot_tree(
    clf,
    feature_names = X_train.columns,
    class_names = class_label,
    filled=True,
    proportion = True,
    fontsize=6,
    rounded = True)

plt.savefig('arvore.png')
plt.show()
```



In [58]:

```python
features = df.drop(columns=['date','state','city']).columns.to_list()
```

Aqui vemos quais colunas foram mais importantes

In [59]:

```python
for feature, value in zip(features, clf.feature_importances_):
    print(f'{feature}: {value}')
```

```
estimated_population: 0.7184680636860525
last_available_confirmed: 0.018336985317452127
last_available_confirmed_per_100k_inhabitants: 0.06825249581441824
last_available_death_rate: 0.16923980982708442
last_available_deaths: 0.004724183972653684
new_confirmed: 0.0
new_deaths: 0.0
epidemiological_week_binning: 0.020978461382339 11
```

In [60]:

```python
scores_cross = cross_val_score(clf, df.drop(columns=['date','state','city']), df['state'],
```

Validação cruzada e a média

In [61]:

```python
scores_cross
```

Out[61]:

```
array([0.64364279, 0.81877984, 0.8225641 , 0.80222812, 0.79837312,
       0.81913351, 0.83908046, 0.8441733 , 0.84420315, 0.8438141 ])
```

In [62]:
```
scores_cross.mean()
```
Out[62]:
```
0.8075992502820855
```

Naive de Bayes

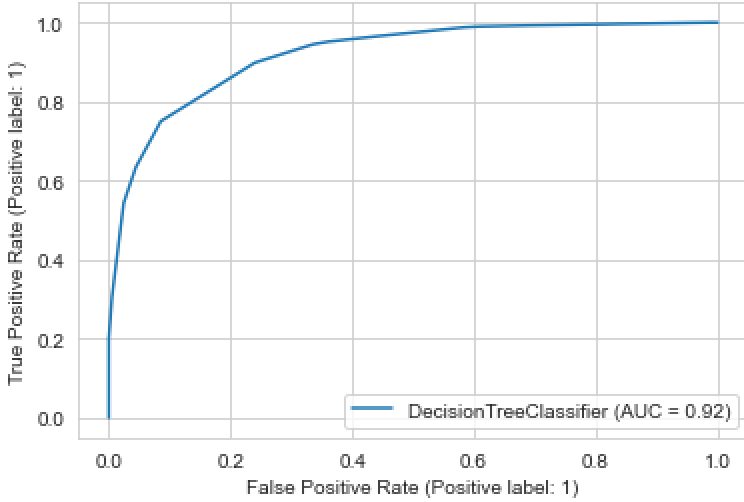
In [63]:
```
naive = GaussianNB()
```
In [64]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop(columns=['date','state','city'
```
In [65]:
```
naive.fit(X_train, y_train)
```
Out[65]:
```
▾ GaussianNB
GaussianNB()
```
In [66]:
```
y_pred = naive.predict(X_test)
```

Também temos um resultado bom com o naive bayes

In [67]:
```
print(classification_report(y_test, y_pred, target_names=['PE','PB']))
```
```
              precision    recall  f1-score   support

          PE       0.73      0.07      0.12     32587
          PB       0.55      0.98      0.71     38100

    accuracy                           0.56     70687
   macro avg       0.64      0.52      0.41     70687
weighted avg       0.63      0.56      0.44     70687
```
In [68]:
```
scores_cross = cross_val_score(naive, df.drop(columns=['date','state','city']), df['state'
```
In [69]:
```
scores_cross
```
Out[69]:
```
array([0.55982317, 0.55865606, 0.55720601, 0.55943413, 0.5603183 ,
       0.56657825, 0.5596817 , 0.55833775, 0.5574733 , 0.55761477])
```
In [70]:
```
scores_cross.mean()
```
Out[70]:
```
0.5595123433707048
```

Aqui é a curva roc da árvore de decisão

In [71]:
```
plot_roc_curve(clf, X_test, y_test, pos_label=1)
plt.show()
```
```
C:\Users\Dayvison\AppData\Roaming\Python\Python39\site-packages\sklearn\utils\deprecation.
py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curv
e` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`s
klearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDispla
y.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



Curva roc do naive bayes

In [72]:
```
plot_roc_curve(naive, X_test, y_test, pos_label=1)
plt.show()
```

```
C:\Users\Dayvison\AppData\Roaming\Python\Python39\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```