

DETECÇÃO DE ANOMALIAS

Implementação

Luiz Celso Gomes-Jr/UTFPR

Detecção de Anomalias

ERBD 2019 - Chapecó/SC

Luiz Celso Gomes-Jr
gomesjr@dainf.ct.utfpr.edu.br

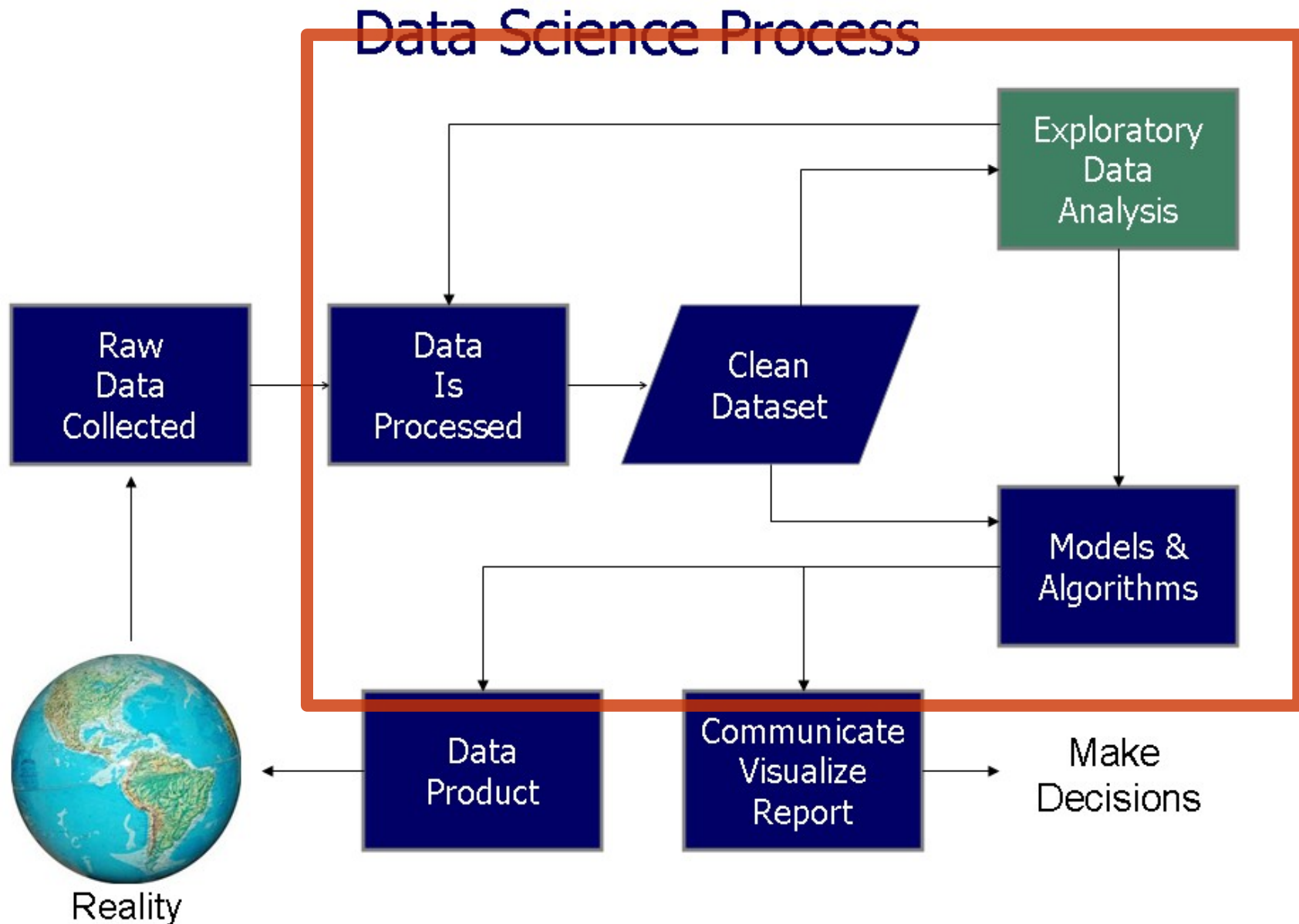
Agenda

- Outliers de ponto
- Múltiplas dimensões
- Contexto
- Redução de Dimensionalidade
- Outliers em Texto
- Outliers em Grafos

Tarefa

- Identificar anomalias no número de atendimentos de saúde (Curitiba)
 - Dias com maior procura
 - Detecção de epidemias
- Detecção de anomalias é um bom exemplo de tarefa de Ciência de Dados, portanto consideraremos o workflow típico

Ciência de Dados

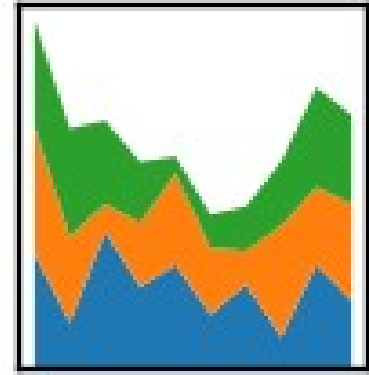
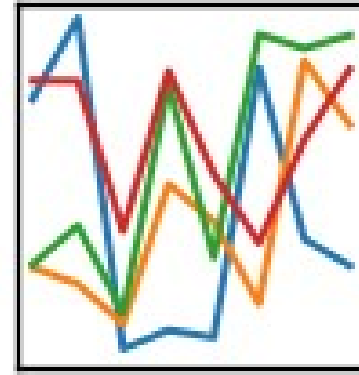


Ferramentas

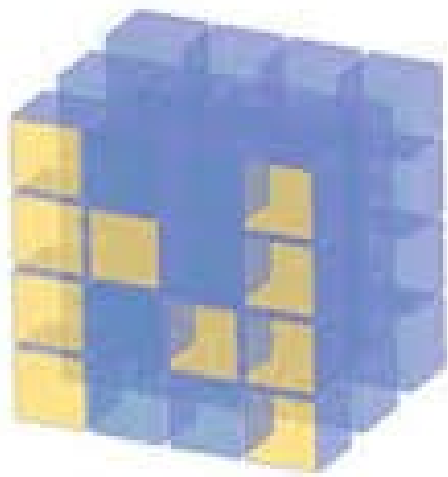
- Pandas - structured data operations and manipulations
- NumPy - Numerical Python
- Matplotlib/Seaborn - plotting graphs
- Scikit Learn - machine learning

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



- Used for **structured data** operations and **manipulations**. It is extensively used for data munging and preparation.
- Have been instrumental in boosting Python's usage in data scientist community.

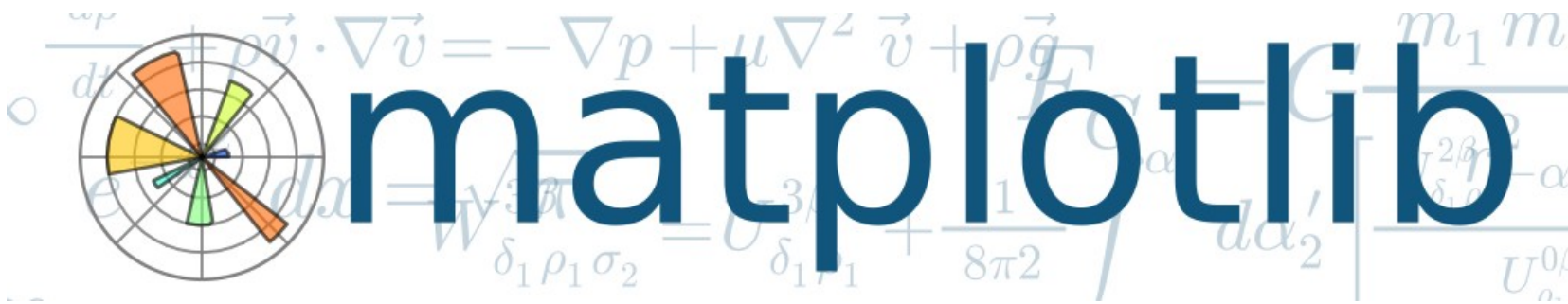


NumPy

- The most powerful feature of NumPy is n-dimensional array.
- This library also contains basic linear algebra functions, Fourier transforms, advanced random number capabilities and tools for integration with other low level languages like Fortran, C and C++
- Efficient processing



- Scikit-learn is a free software machine learning library for the Python programming language
- Features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN
- Designed to interoperate with NumPy and SciPy



- For plotting vast variety of graphs, starting from histograms to line plots to heat plots..
- You can use Pylab feature in Jupyter notebook (ipython notebook –pylab = inline) to use these plotting features inline.
- You can also use Latex commands to add math to your plot.
- Alternatives: Bokeh, Seaborn

Obtenção e Tratamento dos Dados

- Portal Dados Abertos de Curitiba
- Atendimentos de saúde entre 2016-06-01 e 2017-05-31

```
df = pd.read_csv('dadoslimpos.csv', encoding='latin-1', sep=';', low_memory=False)
```

id	Data do Atendimento	Data de Nascimento	Sexo	Código do Procedimento	Código do CID	Desencadeou Internamento
616666	2016-06-01	1920-11-12	F	301060029	Z0	Nao
200151	2016-06-01	1927-10-27	F	301010072	R4	Nao
179894	2016-06-01	1927-12-20	F	301060029	R5	Sim

Obtenção e Tratamento dos Dados

```
#Cria faixas etárias
df['Faixa'] = np.nan
df.loc[df['Idade'] < 14, 'Faixa'] = 'Crianca'
df.loc[df['Idade'] >= 14, 'Faixa'] = 'Adulto'
df.loc[df['Idade'] >= 60, 'Faixa'] = 'Idoso'

#Converte Internamento (Sim/Nao) em binário
df['Internamento'] = 0
df.loc[df['Desencadeou Internamento'] == 'Sim', 'Internamento'] = 1

#Converte data para tipo DateTime
df['Data'] = pd.to_datetime(df['Data do Atendimento'])

#Retira códigos CID errados
df = df.loc[df.loc[:, 'Código do CID'].str.contains(r'^[a-zA-Z][0-9]+$')]

#Mantém apenas o prefixo:
df['Código do CID'] = df['Código do CID'].str.slice(0, 2)
```

Análise Exploratória

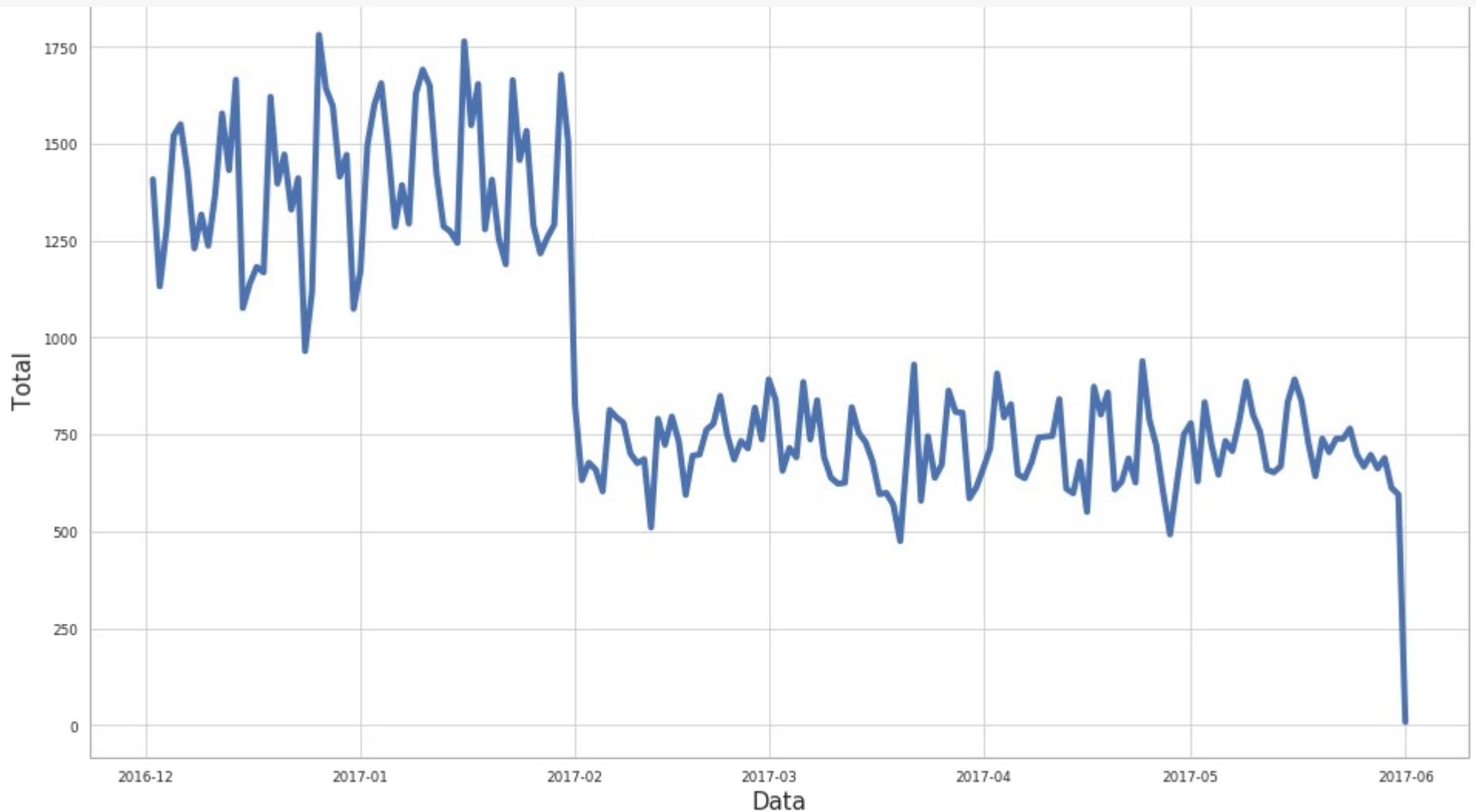
- Contagem de atendimentos por dia
- Exibição da Série Temporal

```
#Contagem de atendimentos por dia  
df_count = df.groupby(by="Data").size().reset_index(name='Total')  
  
df_count.head()
```

	Data	Total
0	2016-06-01	743
1	2016-06-02	659
2	2016-06-03	688
3	2016-06-04	600
4	2016-06-05	563

Análise Exploratória

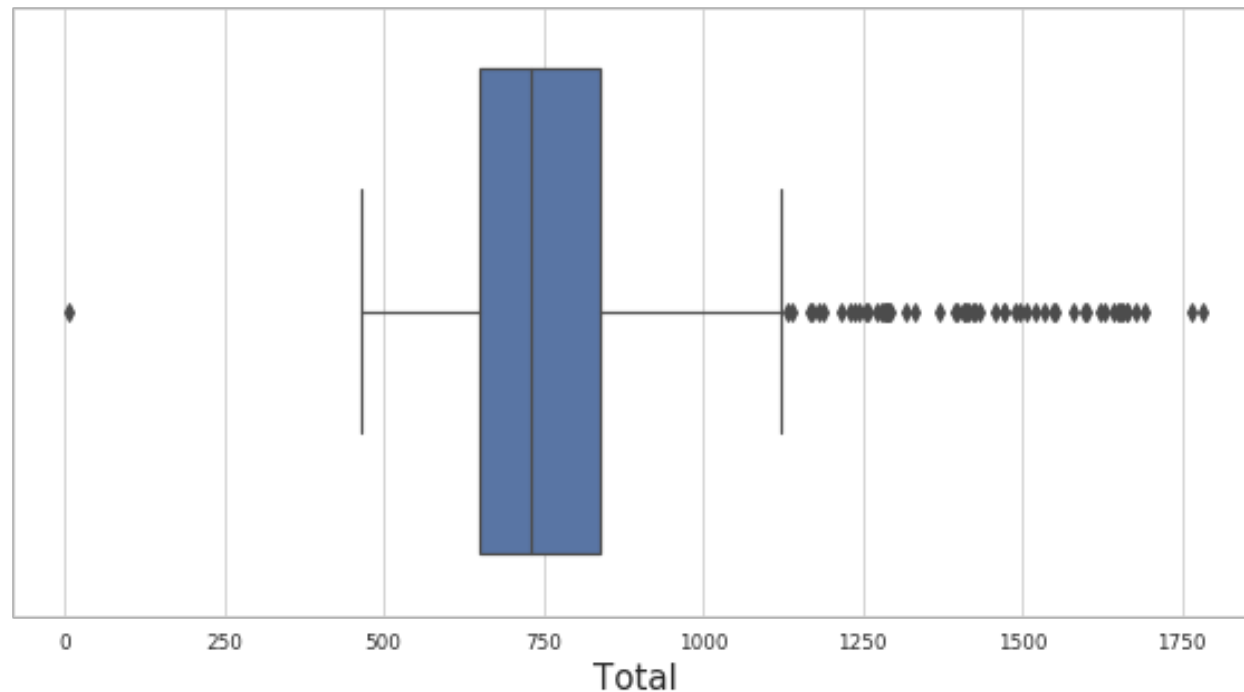
```
sns.lineplot(x = 'Data', y = 'Total', ax=ax, data=df_count.query)
```



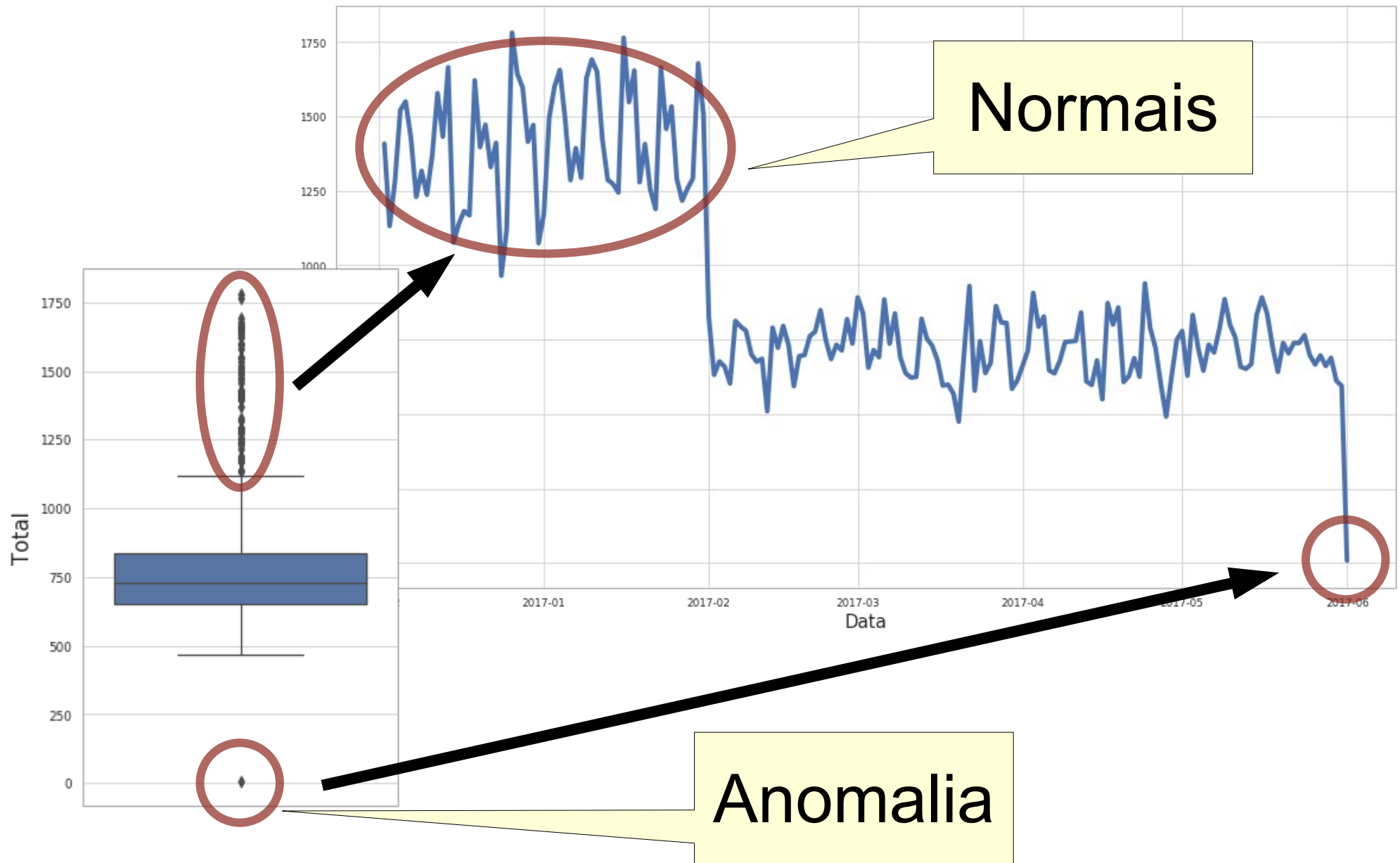
Análise Exploratória

- Detecção de anomalias inicial - BoxPlot

```
sns.boxplot(x=df_count["Total"])
```



Análise Exploratória



Detecção de Anomalias

- Detecção de anomalias inicial – Z-Scores
- Pontos com z-score maior que 2 sigmas são considerados anomalias

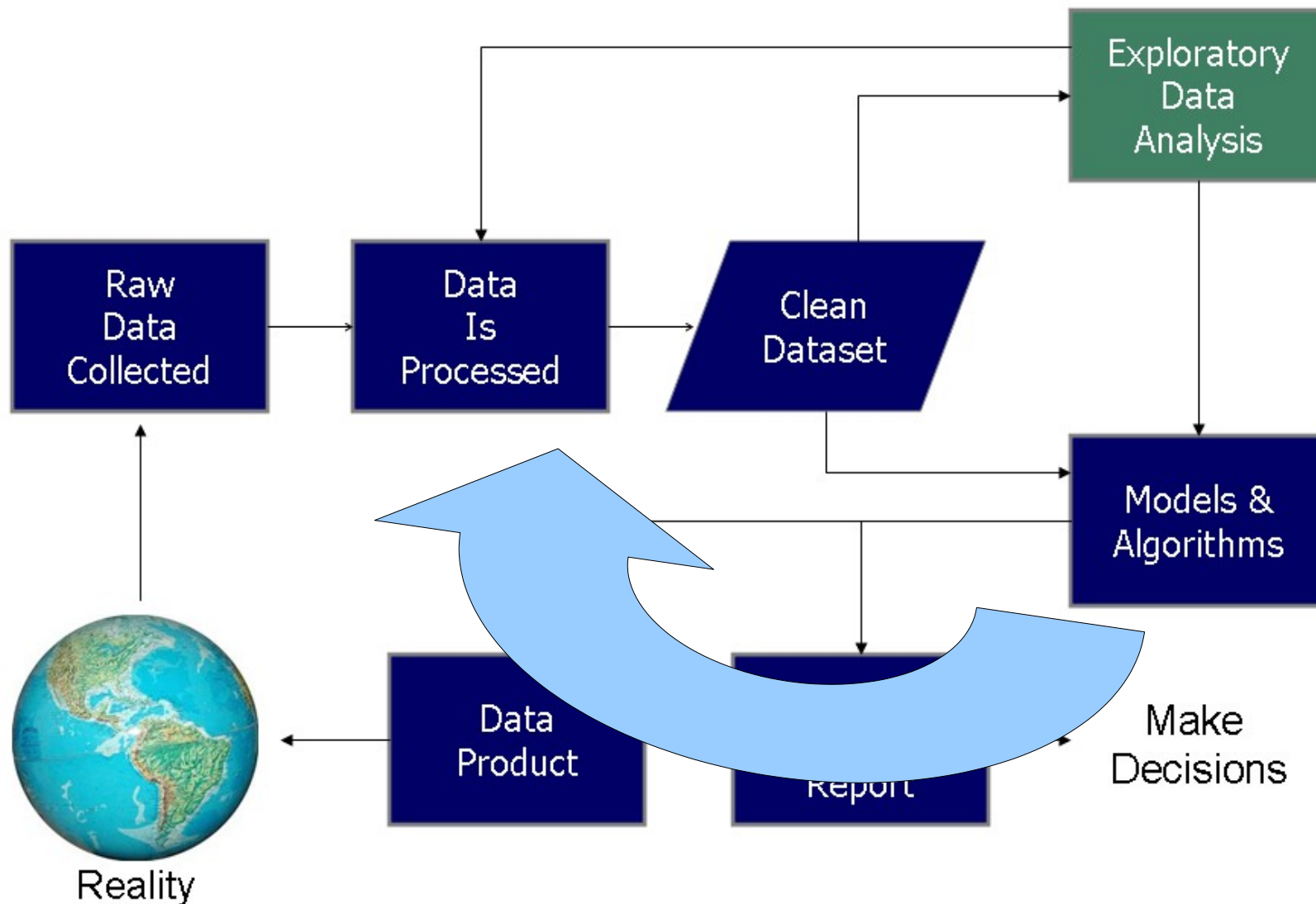
```
df['z_score'] = np.abs(stats.zscore(df['Total']))  
df['z_predic'] = df['z_score'].apply(lambda x: -1 if x >= 2 else 1)
```

Detecção de Anomalias

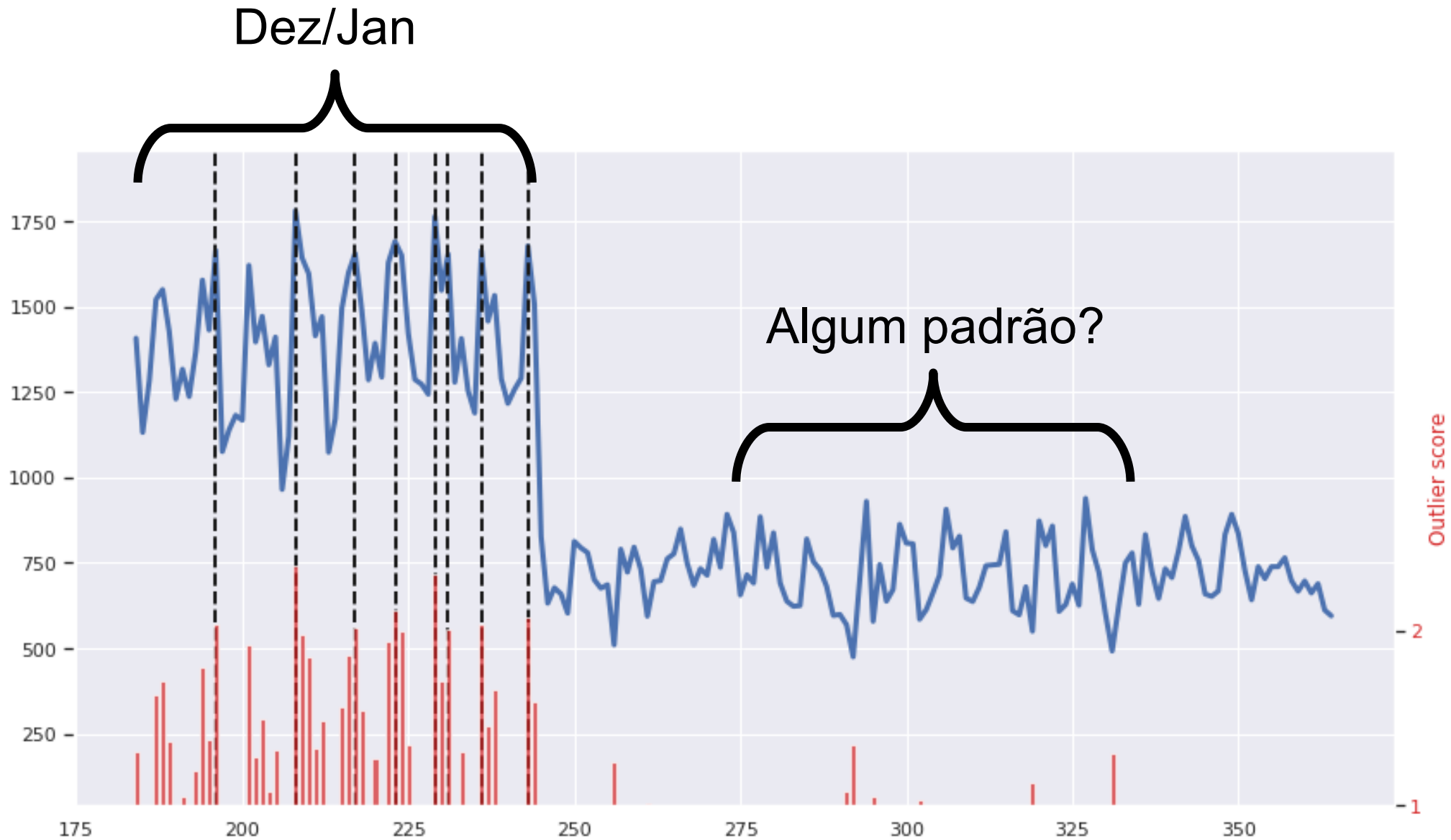


Ciência de Dados

Data Science Process



Análise Exploratória



Análise Exploratória

- Verificar se existe padrão de volume de atendimento por dia da semana
- Extrair também mês e timestamp

```
#Extrai dia da semana
df['Dia_Semana'] = df['Data'].dt.weekday

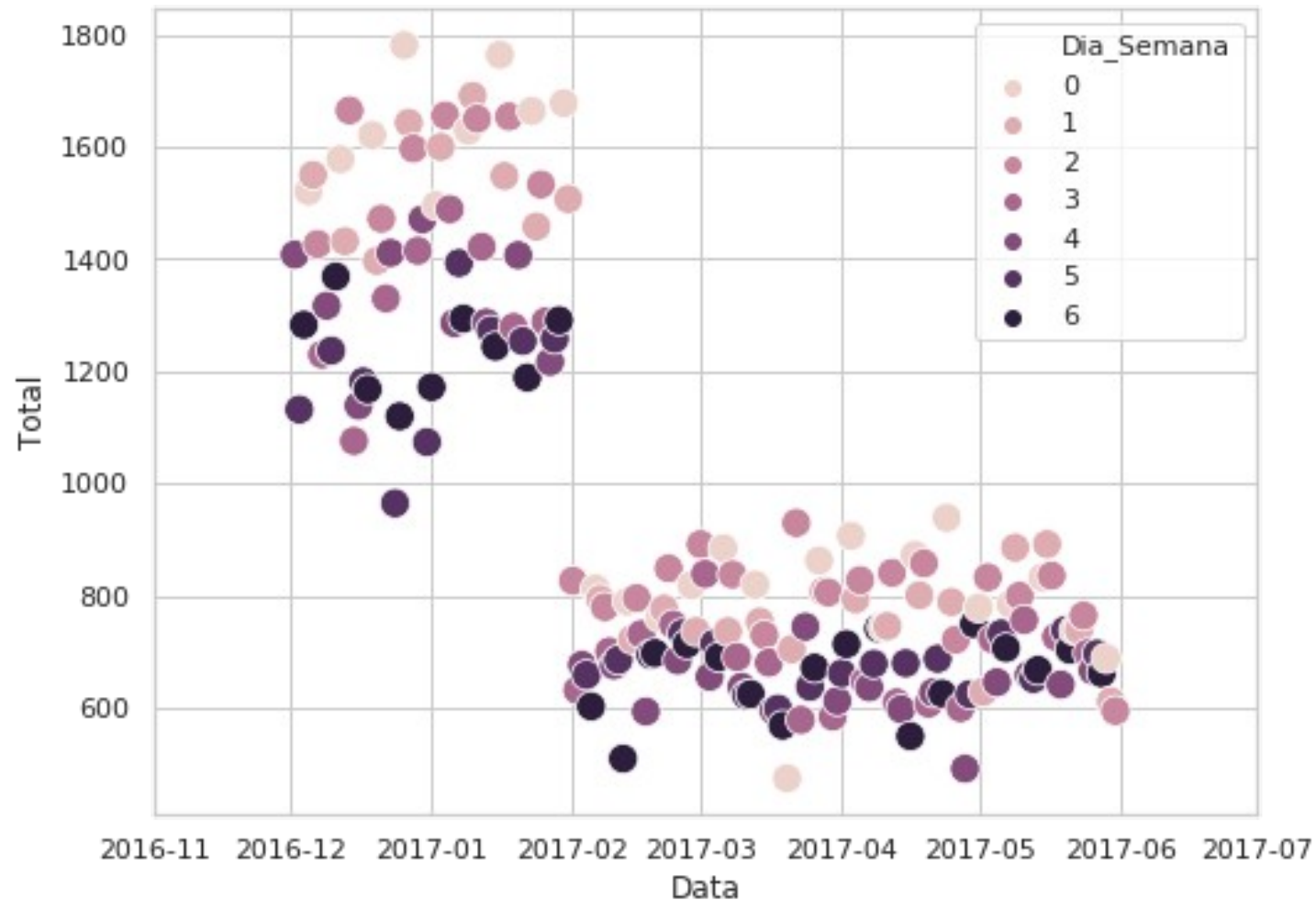
#Extrai mês
df['Mes'] = df['Data'].dt.month_name()

#Extrai TimeStamp
df['ts'] = df[['Data']].apply(lambda x: x['Data'].timestamp(), axis=1).astype(int)
```

	Data	Total	ts	Dia_Semana	Mes
184	2016-12-02	1408	1480636800	4	December
185	2016-12-03	1132	1480723200	5	December
186	2016-12-04	1283	1480809600	6	December

Análise Exploratória

```
sns.scatterplot(x="Data", y="Total",  
                hue="Dia_Semana", s=150, data=df_analysis, legend='full')
```



Contexto

- Claramente a detecção do que é ou não anomalia neste caso depende do **contexto**
- Um pico de atendimento numa segunda-feira é normal, enquanto num domingo seria anômalo
- Estratégia: transformar em detecção multivariada usando LOF (Local Outlier Factor)

Modelo LOF

- Dimensões: Atendimentos X Dia Semana

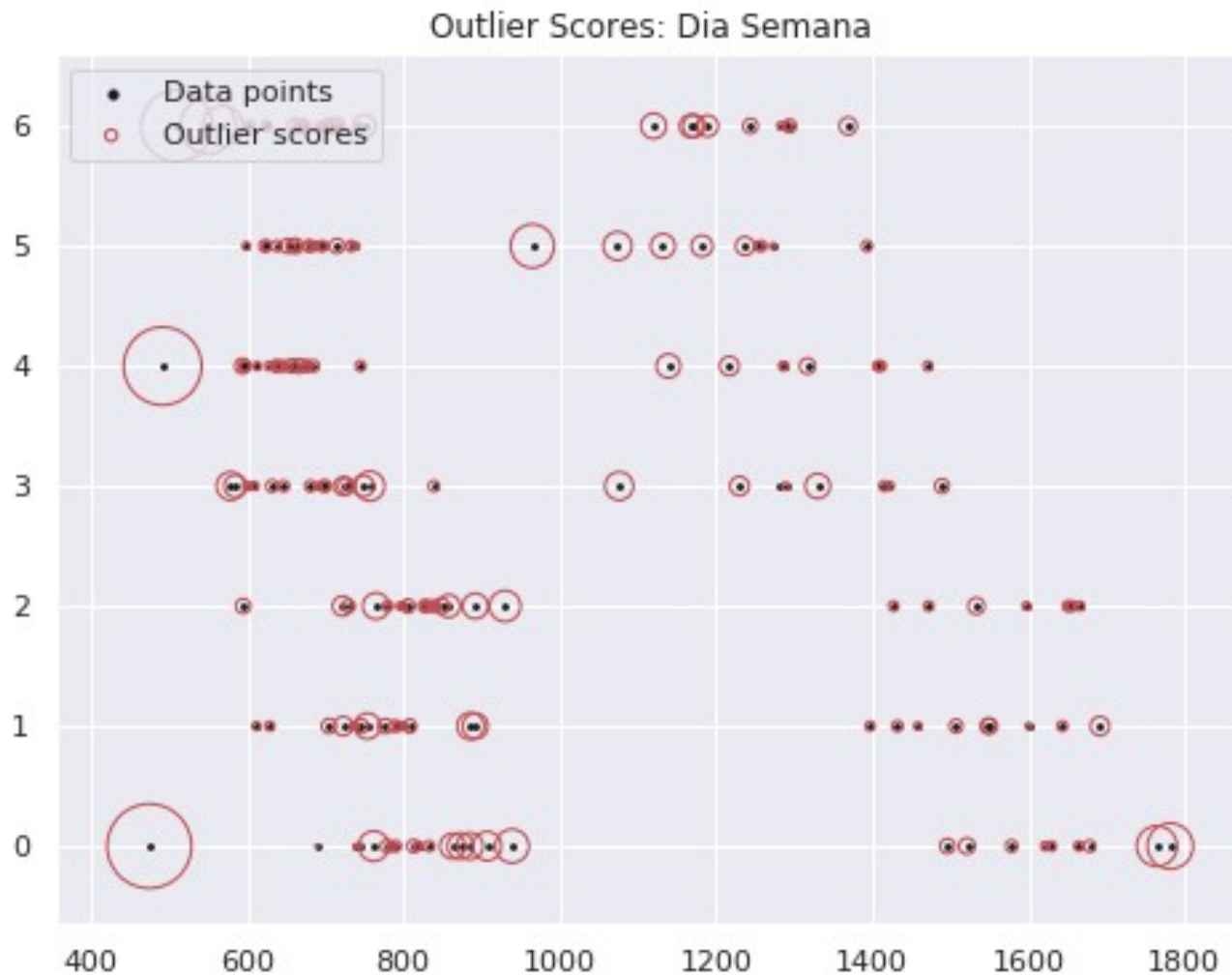
```
df_analysis_a = df_analysis[['Total', 'Dia_Semana']]  
df_analysis_a = get_LOF_scores(df_analysis_a, n_neighbors=10, contamination=0.05)
```

```
def get_LOF_scores(df, n_neighbors=10, contamination=0.05):  
    np.random.seed(42)  
  
    # fit the model for outlier detection (default)  
    clf = LocalOutlierFactor(n_neighbors=n_neighbors, contamination=contamination)  
    # use fit_predict to compute the predicted labels of the training samples  
    # (when LOF is used for outlier detection, the estimator has no predict,  
    # decision_function and score_samples methods).  
    y_pred = clf.fit_predict(df)  
  
    X_scores = clf.negative_outlier_factor_  
  
    df['LOF_score'] = X_scores  
    df['LOF_predictions'] = y_pred  
  
    return df
```



Modelo LOF

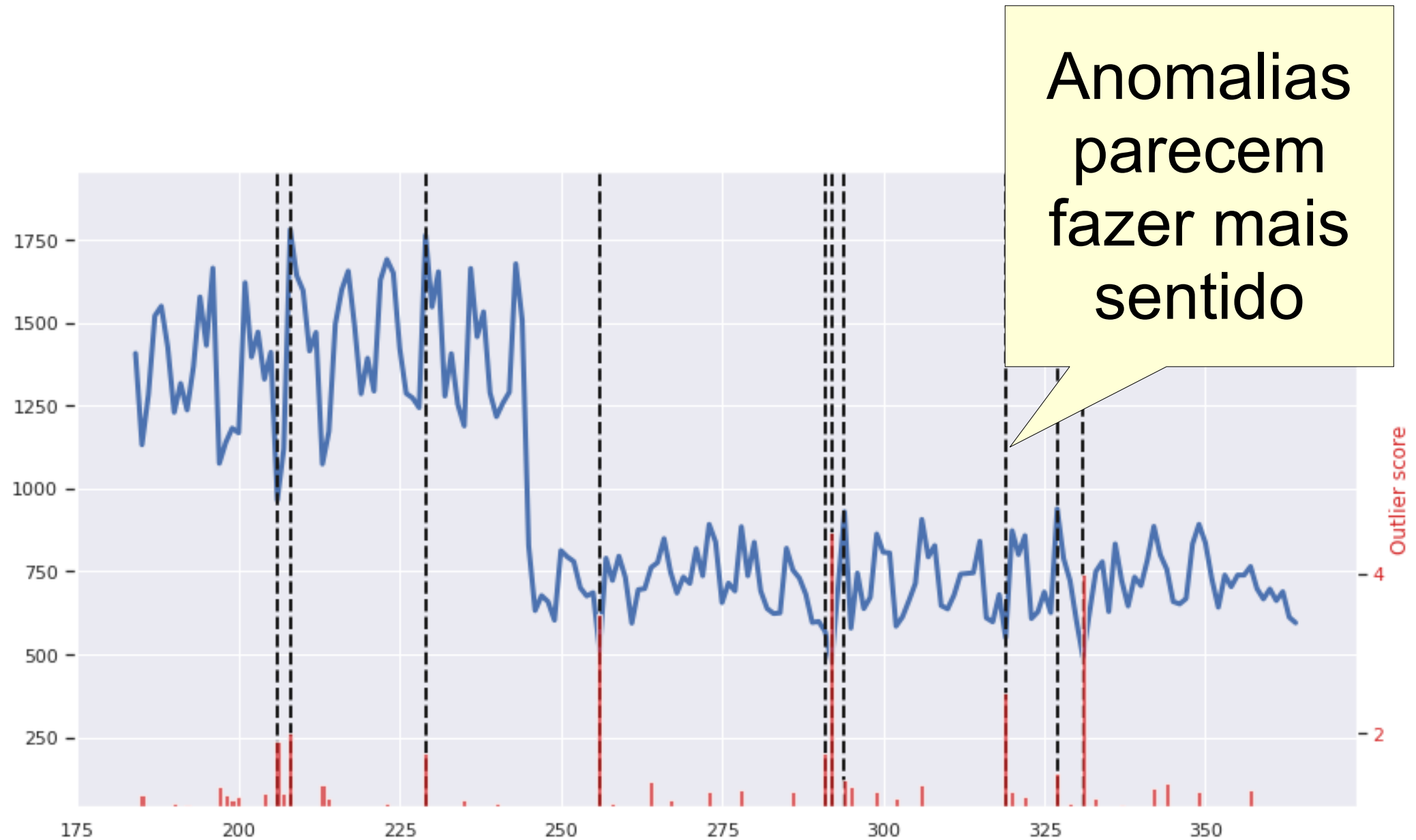
- Dimensões: Atendimentos X Dia Semana



Modelo LOF

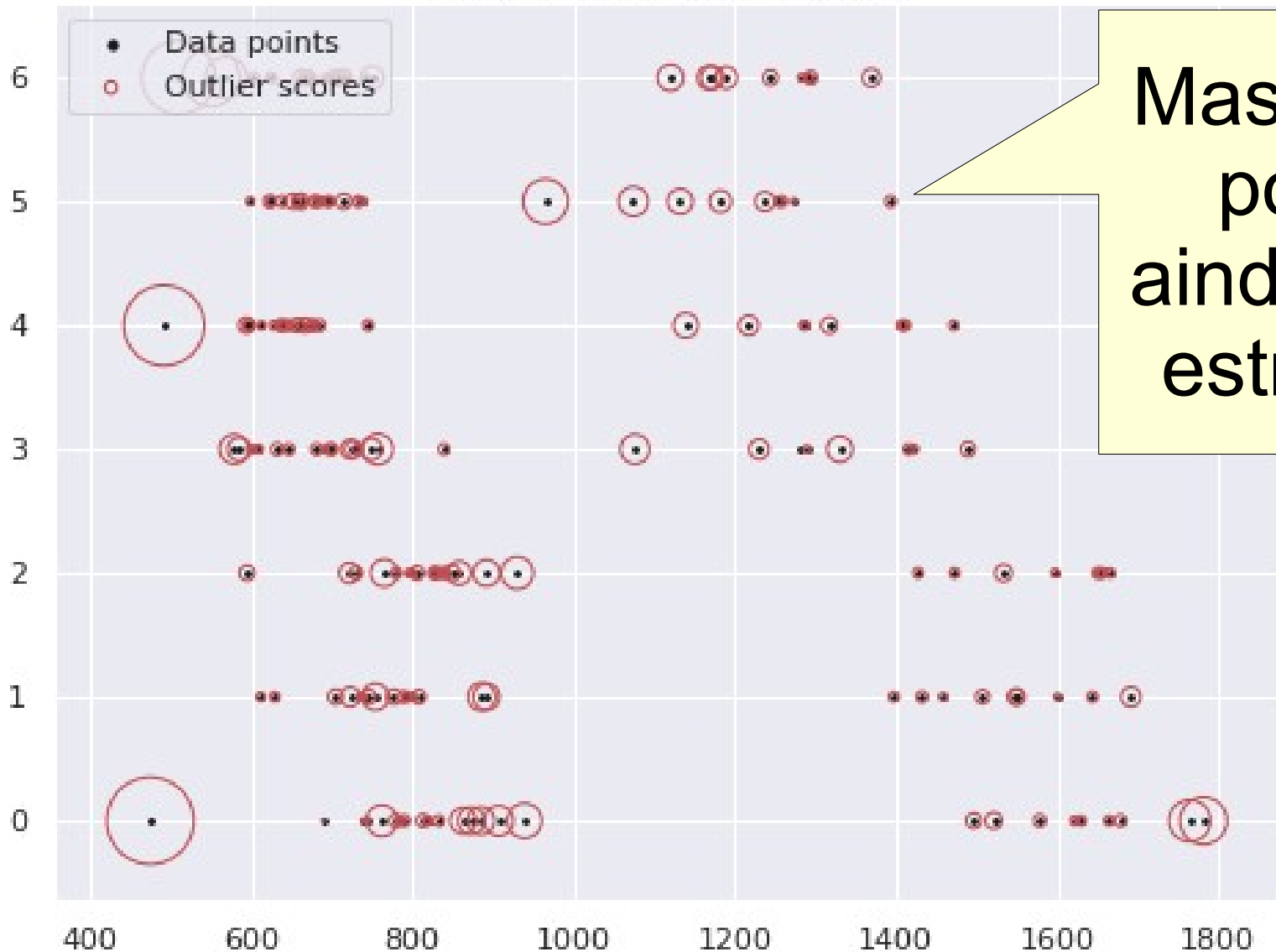


Modelo LOF



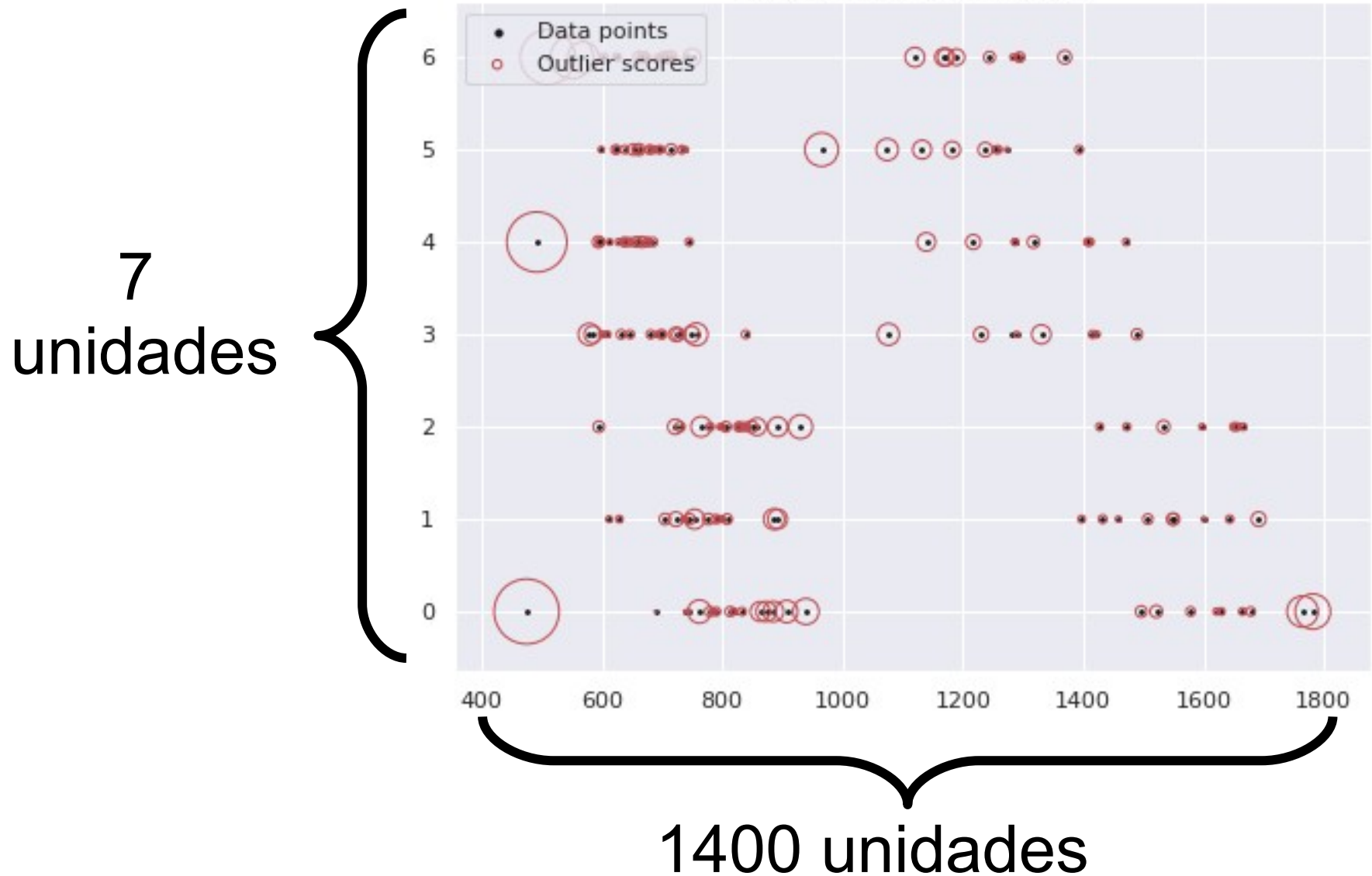
Modelo LOF

Outlier Scores: Dia Semana

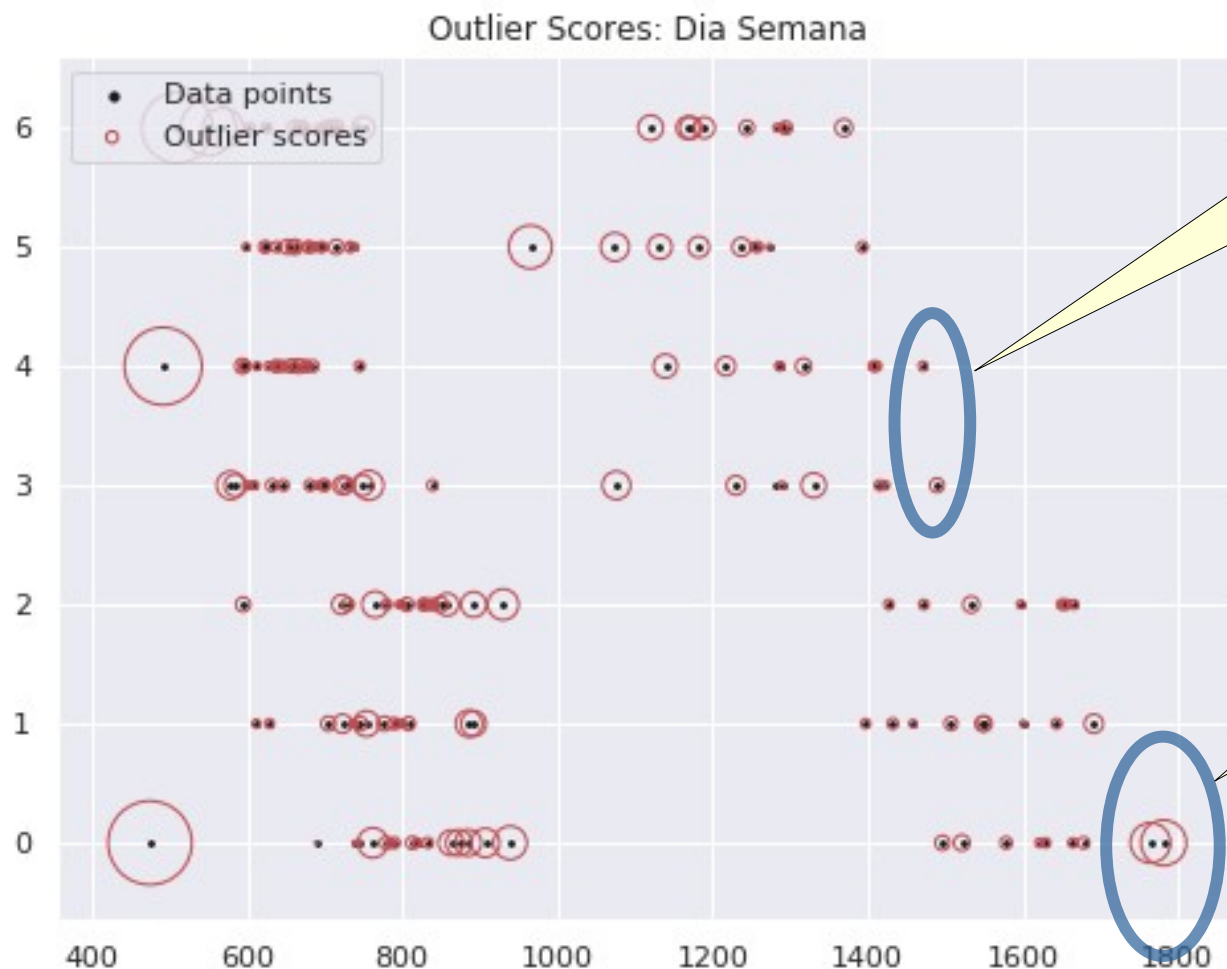


Mas muitos
pontos
ainda estão
estranhos

Modelo LOF - Escala



Modelo LOF - Escala



Vizinhos!

Não vizinhos!

Modelo LOF - Normalização

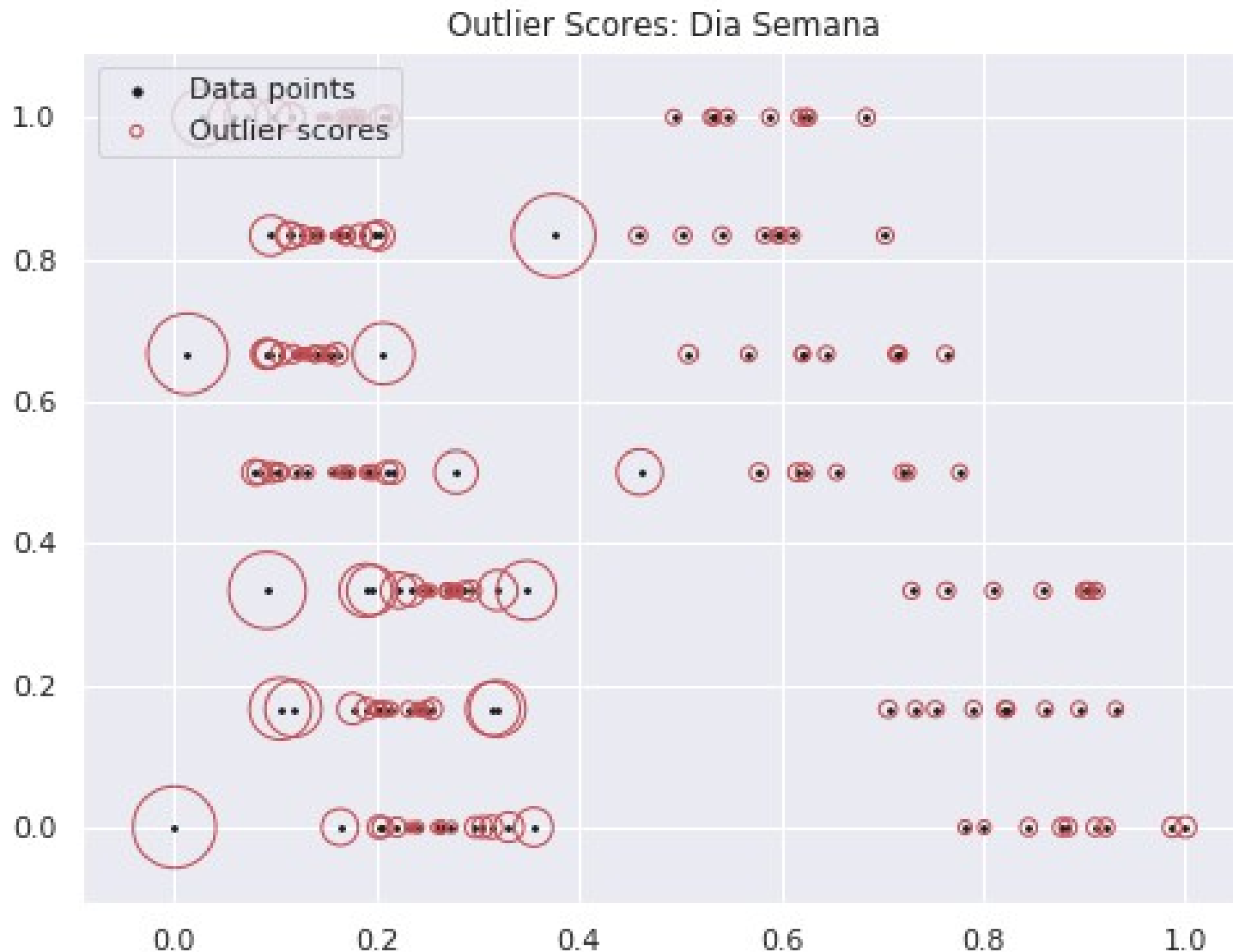
```
df_analysis_a = df_analysis[['Total', 'Dia_Semana']]

#Normaliza os dados
df_analysis_a = pd.DataFrame(
    MinMaxScaler().fit_transform(df_analysis_a),
    index = df_analysis_a.index,
    columns=df_analysis_a.columns)

scores = get_LOF_scores(df_analysis_a, n_neighbors=10, contamination=0.3)
```



Modelo LOF - Normalização



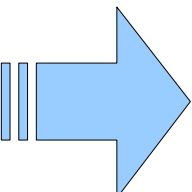
Variáveis Categóricas

- Variáveis categóricas precisam ser tratadas a parte
- Casos claros são as variáveis código da doença e faixa etária, que precisam ser transformadas em um números reais
- Outros casos a se considerar são dia da semana e mês
- Estratégia: obter variáveis *dummies*

Variáveis Categóricas

```
df_mes = df_analysis[['Mes', 'Data']].set_index('Data').sample(8)
df_dummies = pd.get_dummies(df_mes)
```

Mes							
Data							
		Mes_April	Mes_December	Mes_February	Mes_March	Mes_May	
2016-12-29	December						
2016-12-15	December						
2017-04-26	April						
2016-12-31	December						
2017-03-18	March						
2017-05-27	May						
2017-02-23	February						
2017-05-05	May						



Data		Mes_April	Mes_December	Mes_February	Mes_March	Mes_May	
2016-12-29		0	1	0	0	0	
2016-12-15		0	1	0	0	0	
2017-04-26		1	0	0	0	0	
2016-12-31		0	1	0	0	0	
2017-03-18		0	0	0	1	0	
2017-05-27		0	0	0	0	1	
2017-02-23		0	0	1	0	0	
2017-05-05		0	0	0	0	1	

Modelo Final

- Variáveis: mês, dia da semana, código CID, faixa etária, internamento
- Totais de atendimentos por dia calculados agrupando por código CID, faixa etária, internamento
- Normalizados e com respectivas dummies
- Modelo final com 96 dimensões!
- Algoritmo escolhido: SOM

Modelo Final

Processamento dos Dados

```
#Seleciona apenas "Certain infectious and parasitic diseases (A00-B99)"
df_final = df[df['Código do CID'] <= 'B99']

#Contagem de atendimentos por dia e por código
df_final = df_final.groupby(['Data', 'Código do CID', 'Faixa', 'Internamento'])
                    .size().reset_index(name='Total')

df_final['Categoria'] =
    df_final["Código do CID"] + '-' +
    df_final["Faixa"] + '-' +
    df_final["Internamento"].map(str)

df_final = df_final[['Data', 'Categoria', 'Total']]
```

	Data	Categoria	Total
0	2016-06-01	A0-Adulto-0	18
1	2016-06-01	A0-Crianca-0	10
2	2016-06-01	A0-Idoso-0	4
3	2016-06-01	A2-Adulto-0	1
4	2016-06-01	A5-Adulto-0	1

Modelo Final

Processamento dos Dados

```
# Reorganiza o DataFrame para ter categorias como colunas  
df_final = df_final.pivot(index='Data', columns='Categoria', values='Total')  
df_final = df_final.fillna(0)
```

Categoria	Data	A0- Adulto- 0	A0- Adulto- 1	A0- Crianca- 0	A0- Crianca- 1	A0- Idoso- 0	A0- Idoso- 1	A1- Adulto- 0	A1- Adulto- 1	A1- Idoso- 0
0	2016-06-01	18.0	0.0	10.0	0.0	4.0	0.0	0.0	0.0	0.0
1	2016-06-02	17.0	0.0	8.0	0.0	2.0	0.0	0.0	0.0	0.0
2	2016-06-03	18.0	2.0	11.0	0.0	3.0	0.0	0.0	0.0	0.0
3	2016-06-04	7.0	0.0	9.0	0.0	5.0	0.0	0.0	0.0	0.0

Modelo Final

Processamento dos Dados

#Obtenção das dummies

```
df_s = pd.get_dummies(df_final['Data'].dt.day_name().astype(str))
```

```
df_m = pd.get_dummies(df_final['Data'].dt.month_name())
```

```
df_dummies = pd.concat([df_s, df_m, df_final], axis=1)
```

```
df_f = df_dummies.drop('Data', axis=1)
```

#Normalização

```
df_f = pd.DataFrame(MinMaxScaler().fit_transform(df_f), index = df_f.index, columns=df_f.columns)
```

Modelo Final

Processamento dos Dados

e	March	May	November	October	September	A0- Adulto-0	A0- Adulto-1	A0- Crianca-0	A0- Crianca-1	A0- Idoso-0
0	0.0	0.0	0.0	0.0	0.0	0.133333	0.0	0.200000	0.0	0.4
0	0.0	0.0	0.0	0.0	0.0	0.122222	0.0	0.155556	0.0	0.2
0	0.0	0.0	0.0	0.0	0.0	0.133333	1.0	0.222222	0.0	0.3
0	0.0	0.0	0.0	0.0	0.0	0.011111	0.0	0.177778	0.0	0.5
0	0.0	0.0	0.0	0.0	0.0	0.044444	0.0	0.311111	0.0	0.1
0	0.0	0.0	0.0	0.0	0.0	0.244444	0.0	0.133333	0.0	0.1

Modelo Final

Mapa SOM

```
from minisom import MiniSom

map_size=20
contamination=0.02

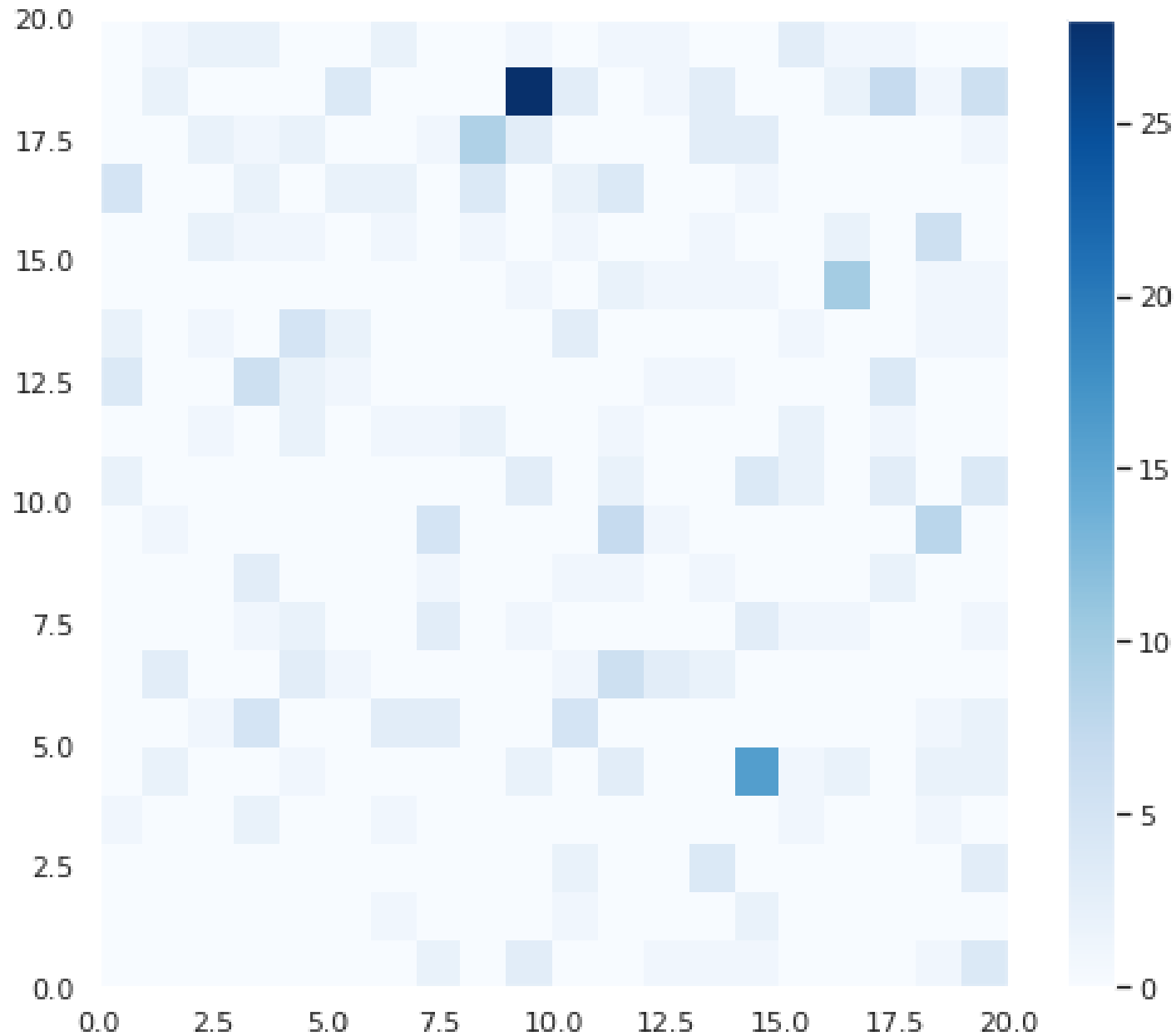
som = MiniSom(map_size, map_size, df_f.shape[1], sigma=1, learning_rate=0.5,
               neighborhood_function='triangle', random_seed=10)

# Outlier scores
quantization_errors = np.linalg.norm(som.quantization(df_f.values) - df_f.values, axis=1)

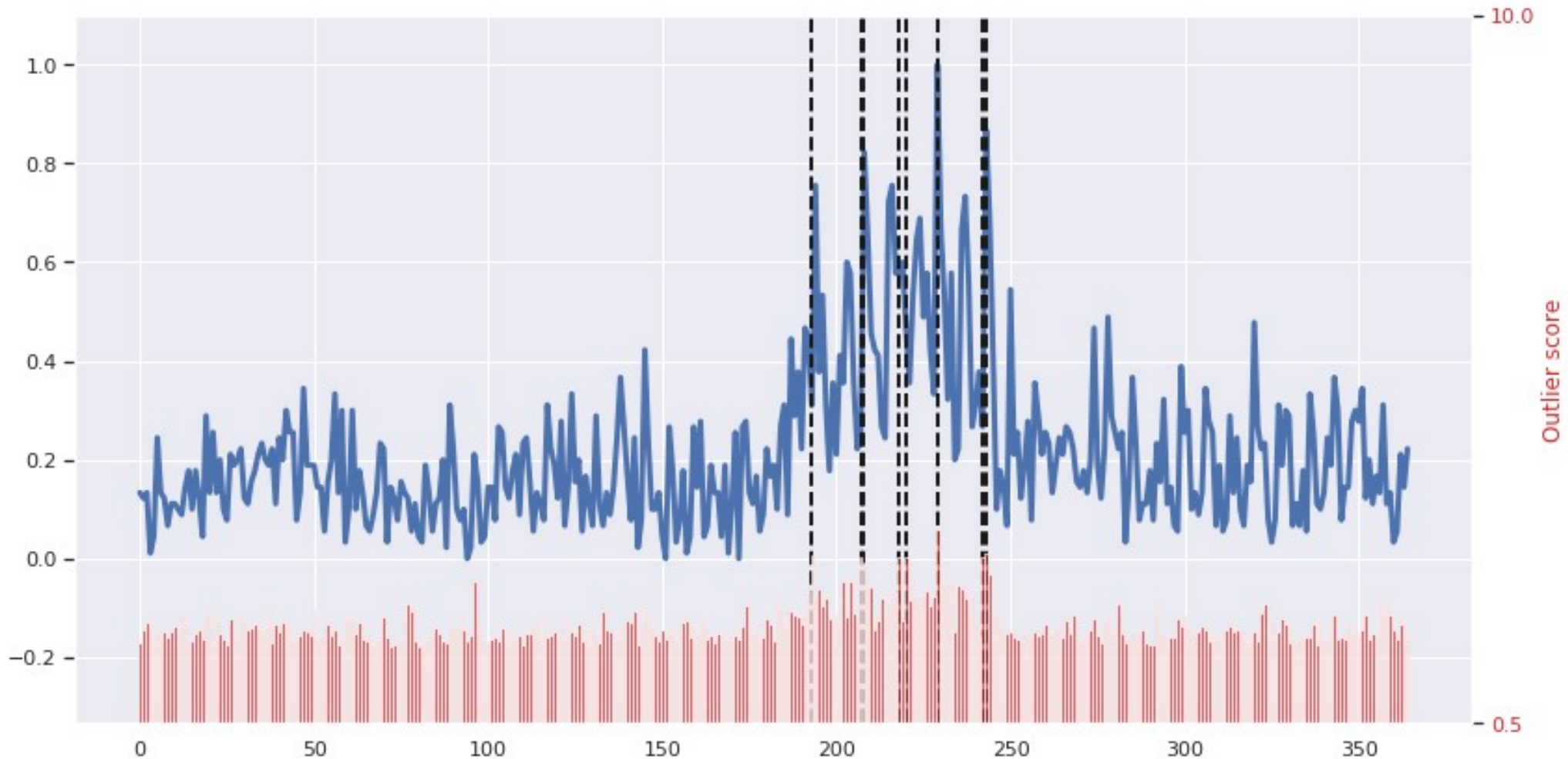
# Definição de um limite para determinar um outlier
error_treshold = np.percentile(quantization_errors,
                               100*(1-contamination))

# Armazena scores e predições no DataFrame
df_f['SOM_qe_score'] = quantization_errors
df_f['SOM_qe_predictions'] = df_f['SOM_qe_score'].apply(lambda x: -1 if x >= error_treshold else 1)
```


Modelo Final



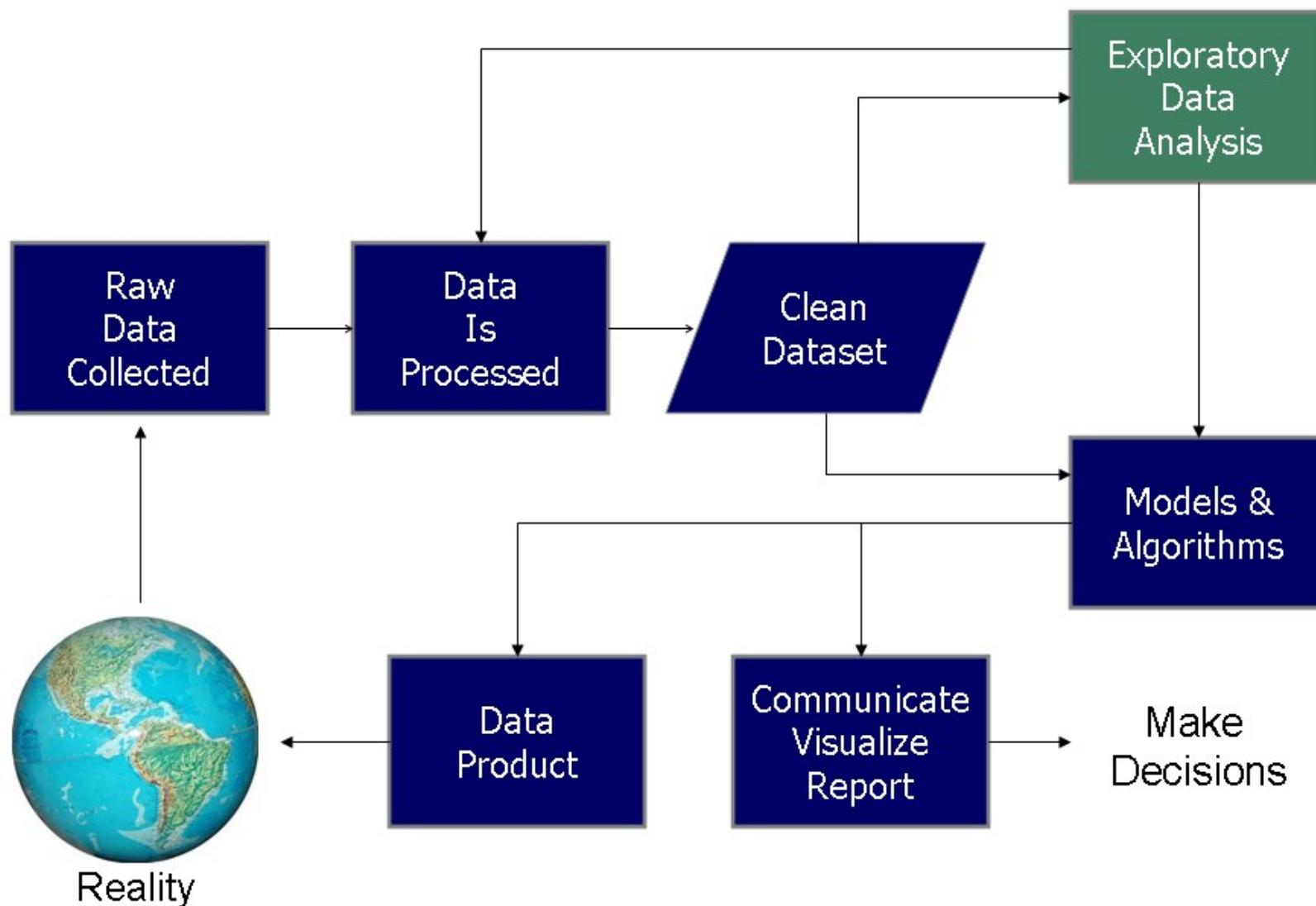
Modelo Final



Obs: valores da variável A3-Adulto-0 (total de atendimentos de adultos com diagnósticos de CID A2 sem internamento)

Ciência de Dados

Data Science Process



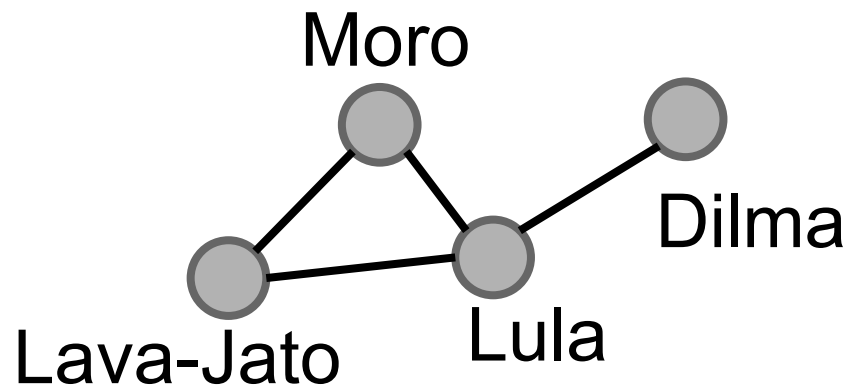
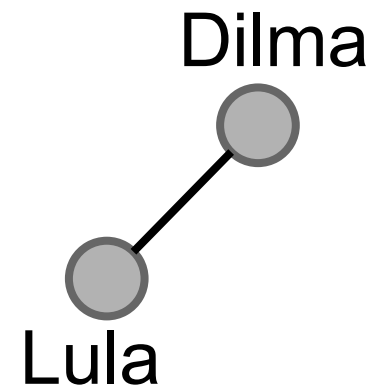
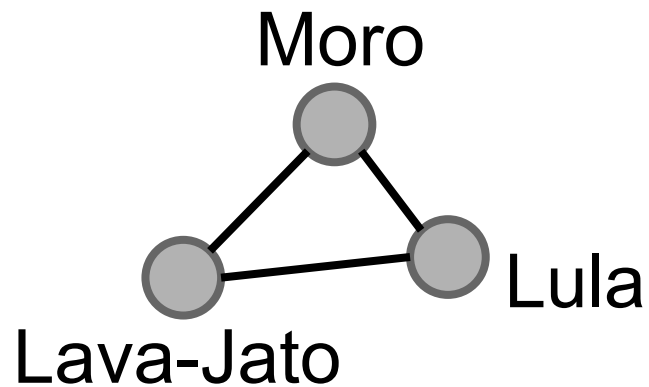
Detecção de Anomalias em Texto usando Grafos

- Cenário: Corpus de notícias falsas
- Objetivo: Identificar mudanças no discurso das notícias ao longo do tempo e comparar com evento reais
- Estratégia: Representar notícias como grafos

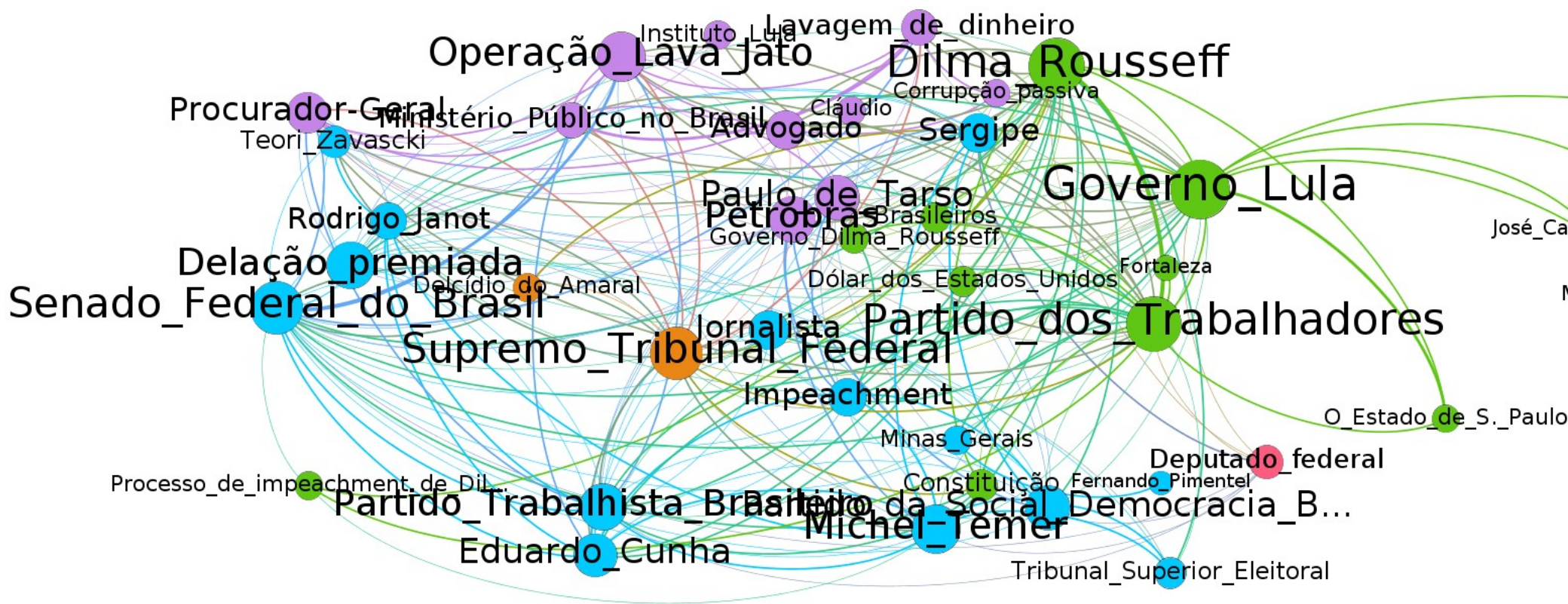
Texto → Grafo

...**Moro** investiga **Lula** na
operação **Lava-Jato**...

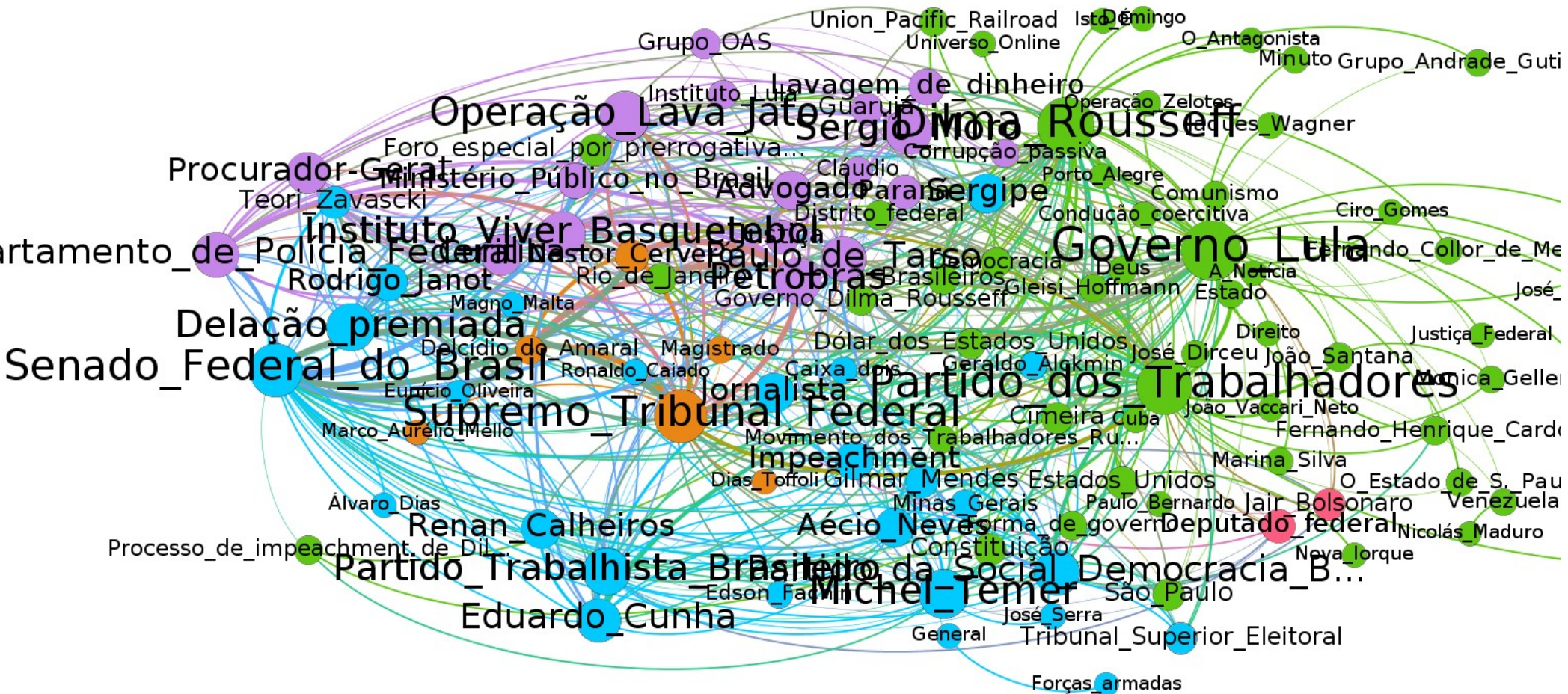
...**Lula** se reúne com
Dilma para tratar...



Grafos de janelas de tempo



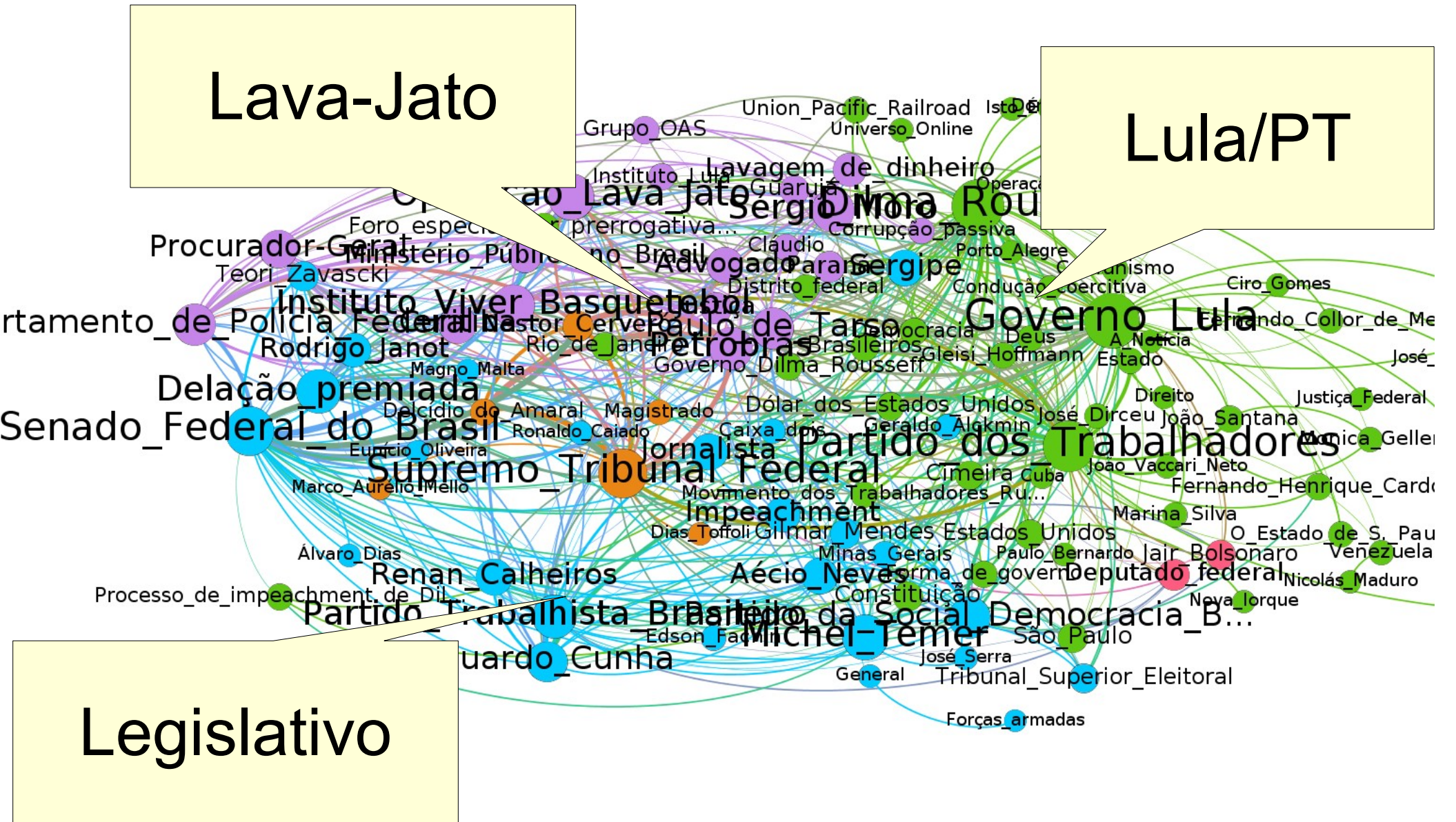
Grafos de janelas de tempo



Identificação de tópicos

- Algoritmo de agrupamento em grafos (Modularidade)
- Agrupamentos representam entidades frequentemente co-citadas
- Agrupamentos usados como representantes de tópicos

Agrupamentos/Tópicos

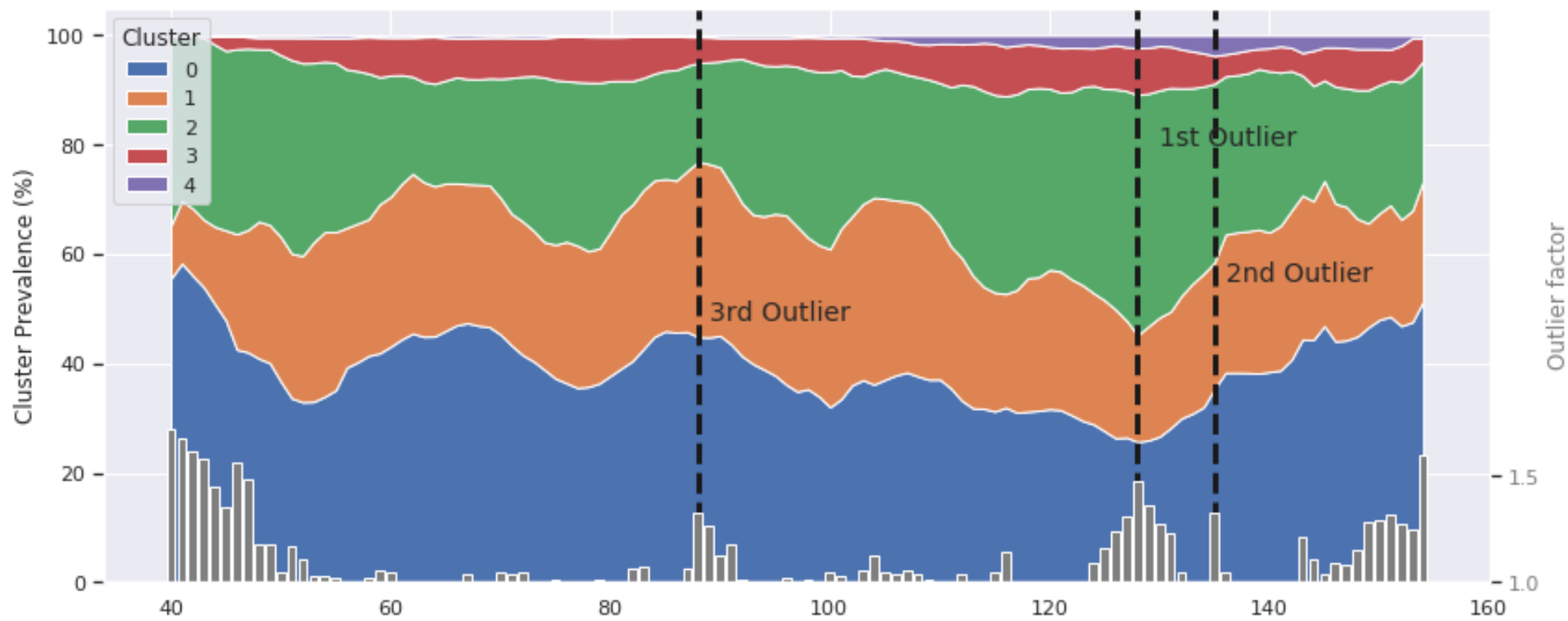
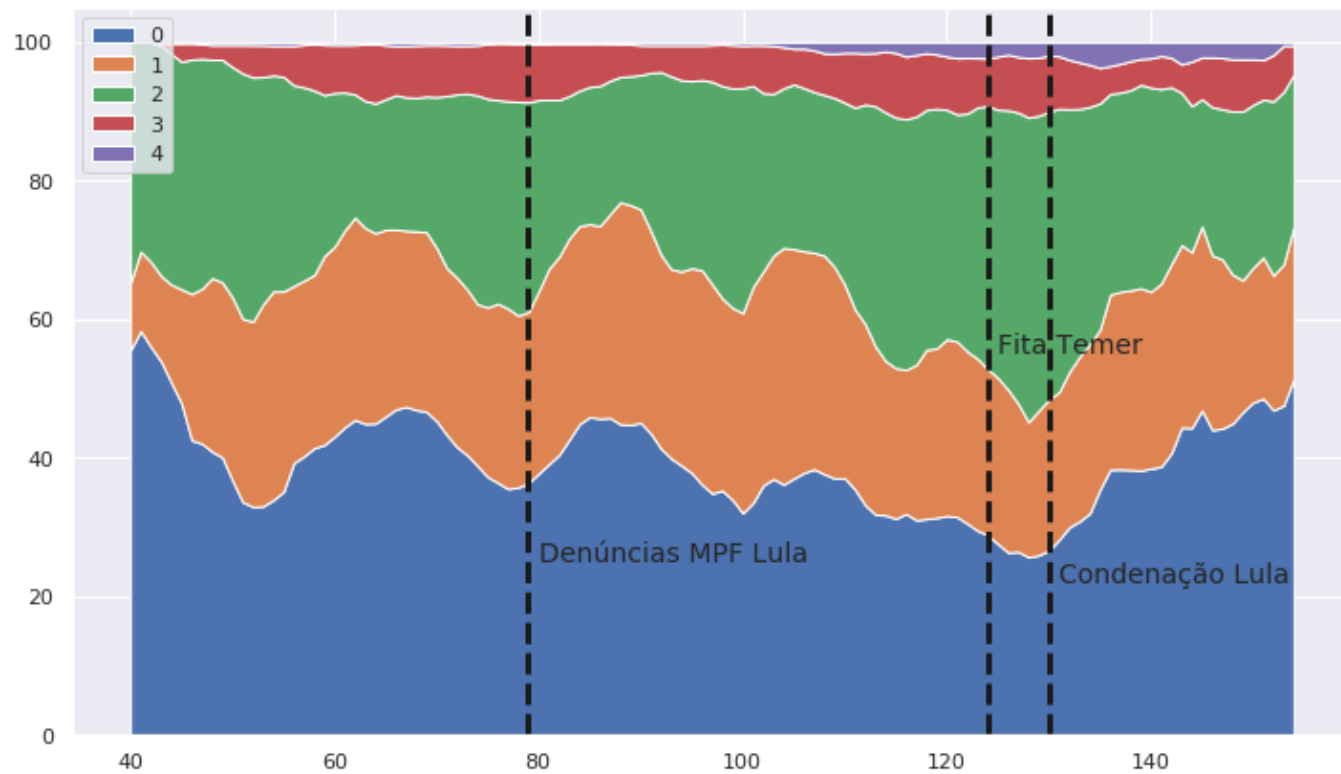


Estratégias

- Detecção de anomalias na distribuição dos tópicos (agrupamentos)
- Detecção de anomalias sobre a evolução dos grafos

Anomalias nas Distribuições dos Tópicos

- Cálculo de LOF para os percentuais dos 5 tópicos ao longo do tempo



Anomalias na evolução dos grafos

- Conversão dos grafos em matrizes de adjacência
- Cálculo de SVD para prever crescimento
- Grau de anomalia para o janelas é a norma da diferença das matrizes reais e previstas

Anomalias na evolução dos grafos

```
from scipy.linalg import svd
import networkx as nx

M1 = nx.to_pandas_adjacency(G1)

M2 = nx.to_pandas_adjacency(G2)

# Singular-value decomposition
U, s, VT = svd(M1)

# create n x n Sigma matrix
Sigma = diag(s)

## Reduzir dimensionalidade (varia de caso a caso)

# reconstruct matrix
M1f = U.dot(Sigma.dot(VT))

#Calcula distância/grau de anormalidade usando norma Frobenius
distancia = np.linalg.norm(M2 - M1f)
```


Obrigado

- Email: luizcelso@gmail.com,
gomesjr@dainf.ct.utfpr.edu.br
- Programa de Pós (PPGCA):
ppgca.dainf.ct.utfpr.edu.br