

# Proposta de Solução\_Final\_v1

February 14, 2022

## 1 Proposta de Solução - João Carlos Casoto Júnior

Olá!

Fiz minha proposta de solução e criei um modelo de classificação para prever fraudes de acordo com os dados das subestações.

### 1.0.1 Etapas desenvolvidas

#### 1. Data Understanding

- Checar estrutura do dataset e distribuição de valores quantitativos
- Procurar por valores faltantes/NA
- Checar a distribuição dos dados na variável target ('classe\_cliente')

#### 2. Data Visualization

- Quant (boxplots, histogramas e Matriz de Correlação)
- Quali (Countplots, Crosstab Chart)

#### 3. Data Wrangling

- Remap das variáveis qualitativas [resultado não foi bom, código comentado]
- One Hot Encoding nas variáveis qualitativas

#### 4. Data Modelling & Evaluation

- Padronizar dados, separar dataset em treino/teste (80/20) e aplicar regra de Cross Validation
- Checar distribuição dos dados com PCA e t-SNE
- Tratar base desbalanceada, com:
  1. UnderSampling [desempenho ruim, código comentado]
  2. OverSampling ROSE
  3. OverSampling SMOTE
- Aplicação de escalas nos dados (Normalização e Padronização)
- Treinamento dos algoritmos:
  - KNN
  - Regressão Linear
  - Random Forest c/ GridSearchCV
  - Gradient Boosting c/ GridSearchCV [resultado não foi bom, código comentado]
  - SVM [resultado não foi bom, código comentado]

#### 5. Model Deployment

- Geração de arquivo .sav (pickle)
- API Deploy (script complementar **API\_Deploy.py**)

### 1.0.2 Outros possíveis testes a serem explorados

1. Inclusão no pipeline de um algoritmo de redes neurais/redes neurais bayesianas (acho que a ponderação nos tipos de erros do modelo ajudariam no desempenho final);
2. Realizar mais testes com ajuste de hiperparâmetros no random forest com GradientBoosting;
3. Ajuste de hiperparâmetros para novas tentativas com SVM;
4. Melhor entendimento do processo afim de tentar incluir novas variáveis (mais explicativas) no dataset.
5. Melhor ajuste do código para desenvolvimento da API (estruturação em classes, padronização de inputs em formato JSON)

## 2 Data Understanding

```
[1]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import os

pd.set_option('display.float_format', lambda x: '%.2f' % x) # Remover notação
    científica do DataFrame
pd.options.display.max_columns = None # Mostrar todas as colunas
warnings.filterwarnings("ignore") # Ignorar warnings
```

```
[2]: # Leitura do dataset
dataset = pd.read_csv('dados.csv')
display(dataset.head())
```

	id_subestacao	local_medidor	tipo_medidor	consumo_medio_mensal	\
0	14	comercial	digital	293	
1	30	residencial	digital	190	
2	38	residencial	digital	399	
3	40	residencial	analogico	307	
4	15	residencial	digital	407	

	temperatura_maxima	temperatura_minima	numero_fases	classe_cliente
0	42	10	monofasica	normal
1	36	13	trifasica	normal
2	31	14	bifasica	normal
3	45	11	monofasica	normal
4	39	19	trifasica	normal

```
[3]: # Checando estrutura do dataset e distribuição de valores quantitativos
display(dataset.info())
display(dataset.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 150000 entries, 0 to 149999

Data columns (total 8 columns):

#	Column	Non-Null Count	Dtype
0	id_subestacao	150000 non-null	int64
1	local_medidor	150000 non-null	object
2	tipo_medidor	150000 non-null	object
3	consumo_medio_mensal	150000 non-null	int64
4	temperatura_maxima	150000 non-null	int64
5	temperatura_minima	150000 non-null	int64
6	numero_fases	150000 non-null	object
7	classe_cliente	150000 non-null	object

dtypes: int64(4), object(4)

memory usage: 9.2+ MB

None

	id_subestacao	consumo_medio_mensal	temperatura_maxima \
count	150000.00	150000.00	150000.00
mean	25.03	277.82	39.49
std	14.17	102.49	5.77
min	1.00	100.00	30.00
25%	13.00	190.00	34.00
50%	25.00	278.00	39.00
75%	37.00	365.00	44.00
max	49.00	549.00	49.00

	temperatura_minima
count	150000.00
mean	16.99
std	4.32
min	10.00
25%	13.00
50%	17.00
75%	21.00
max	24.00

```
[4]: # Procurando por valores faltantes/NA
dataset.isna().sum()
```

```
[4]: id_subestacao      0
     local_medidor      0
     tipo_medidor       0
     consumo_medio_mensal 0
     temperatura_maxima  0
     temperatura_minima  0
```

```
numero_fases          0
classe_cliente         0
dtype: int64
```

```
[5]: # Verificando as colunas qualitativas se não há preenchimentos indevidos
      ↪(blanks, descrições fora do padrão, etc.)
for col in dataset.loc[:, dataset.dtypes == np.object]:
    display(dataset[col].value_counts())
```

```
industrial      62718
comercial       49695
residencial     37587
Name: local_medidor, dtype: int64
```

```
digital         75669
analogico       74331
Name: tipo_medidor, dtype: int64
```

```
monofasica      50582
trifasica       50146
bifasica        49272
Name: numero_fases, dtype: int64
```

```
normal          145483
fraudador        4517
Name: classe_cliente, dtype: int64
```

```
[6]: # Checando a distribuição dos dados na variável 'classe_cliente'
dataset['classe_cliente'].value_counts(normalize = True)
```

```
[6]: normal          0.97
      fraudador      0.03
      Name: classe_cliente, dtype: float64
```

### 2.0.1 Notas

1. Dados distribuídos em 150 mil registros e oito variáveis (incluindo variável target 'classe\_cliente');
2. Contém variáveis numéricas e categóricas; separarei o dataset para gerar visualizações pontuais;
3. Nas variáveis quantitativas, o consumo médio mensal e as temperaturas representam informações diferentes, portanto, escalas diferentes (valores Min/Max). Dependendo do modelo a ser treinado, será necessário normalizar a base e remover a coluna 'id\_subestacao';
4. Procurei por valores missing ou indevidos tanto nas variáveis quantitativas (contagem simples) quanto qualitativas (via tabelas de contingência) mas nada foi encontrado, logo nenhuma ação será necessária;

5. Avaliando a distribuição da variável 'classe\_cliente', vemos que os dados estão altamente desbalanceados (razão 97/3), por isso será necessário aplicarmos alguma técnica de normalização na base (ex.: ROSE, SMOTE, etc.).

## 3 Data Visualization

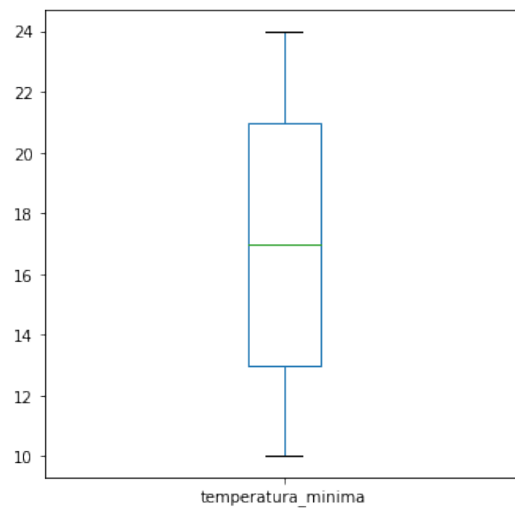
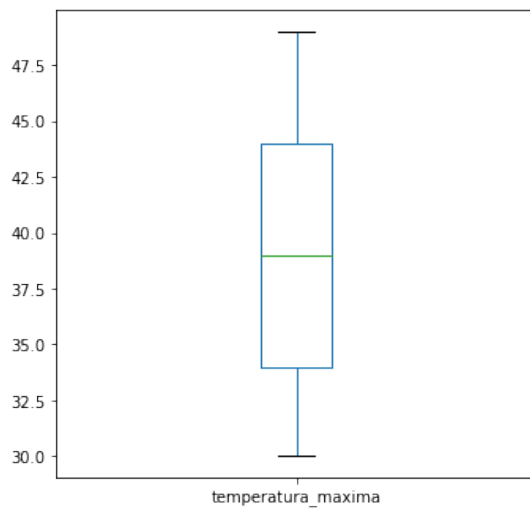
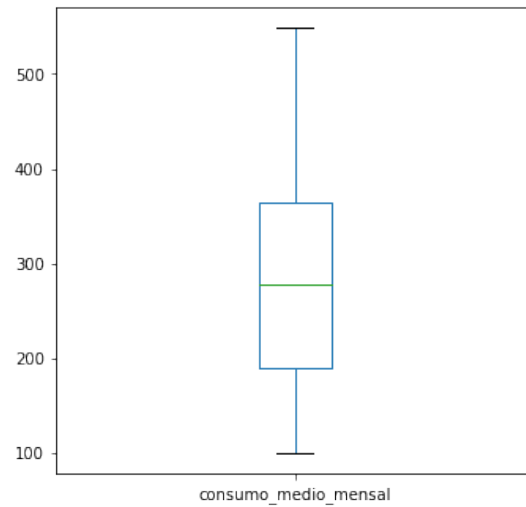
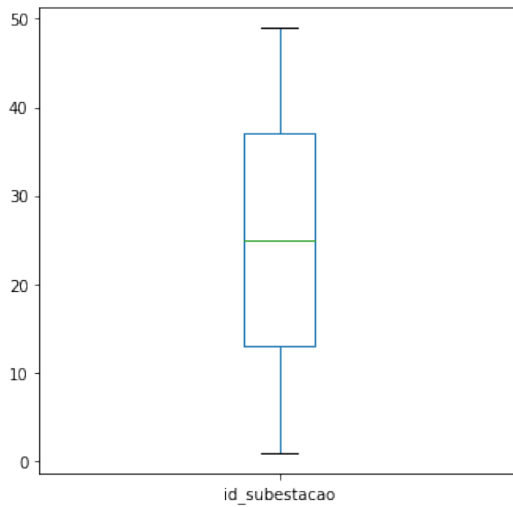
### 3.1 Quantitativas

```
[7]: # Separar colunas categóricas das numéricas para visualizar estrutura dos dados
categorical = dataset.loc[:, dataset.dtypes == np.object]
numerical = dataset.loc[:, dataset.dtypes == np.int64]
print('Categóricas: ', categorical.columns.tolist(), '\nNuméricas: ', numerical.
      ↪columns.tolist())
```

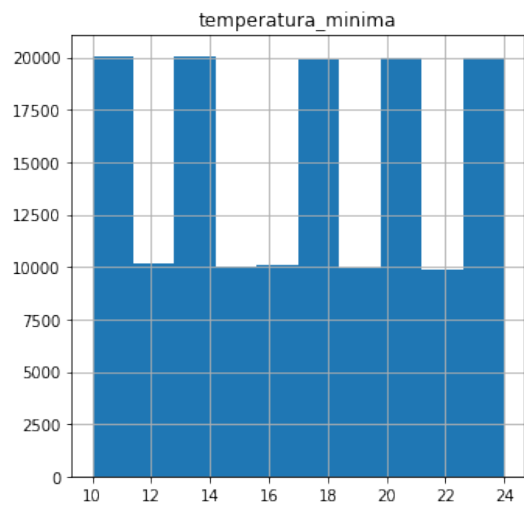
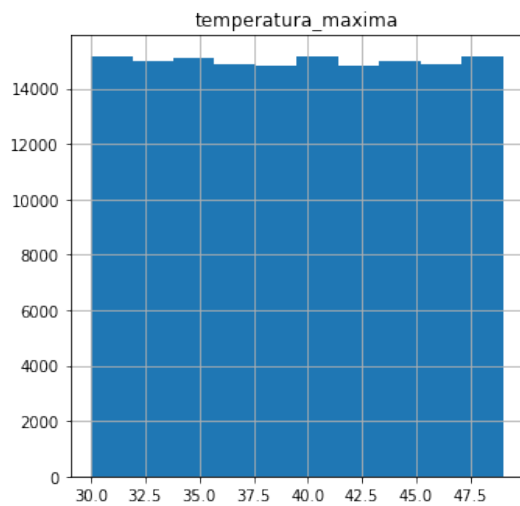
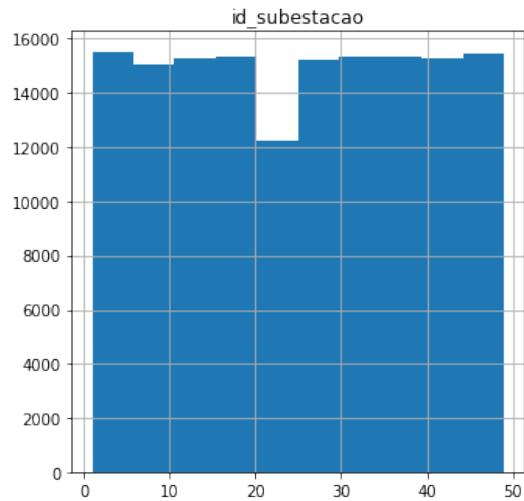
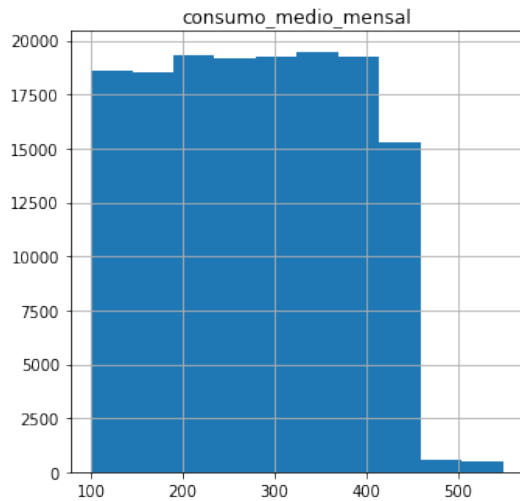
```
Categóricas: ['local_medidor', 'tipo_medidor', 'numero_fases',
              'classe_cliente']
Numéricas: ['id_subestacao', 'consumo_medio_mensal', 'temperatura_maxima',
            'temperatura_minima']
```

```
[8]: # Boxplot com distribuições das variáveis numéricas
numerical.plot(kind = 'box', subplots = True, layout = (2,2), sharex = False,
               ↪sharey = False, figsize=(12, 12))
```

```
[8]: id_subestacao      AxesSubplot(0.125,0.536818;0.352273x0.343182)
consumo_medio_mensal  AxesSubplot(0.547727,0.536818;0.352273x0.343182)
temperatura_maxima    AxesSubplot(0.125,0.125;0.352273x0.343182)
temperatura_minima    AxesSubplot(0.547727,0.125;0.352273x0.343182)
dtype: object
```



```
[9]: # Histogramas de distribuição
numerical.hist(figsize=(12, 12))
plt.show()
```



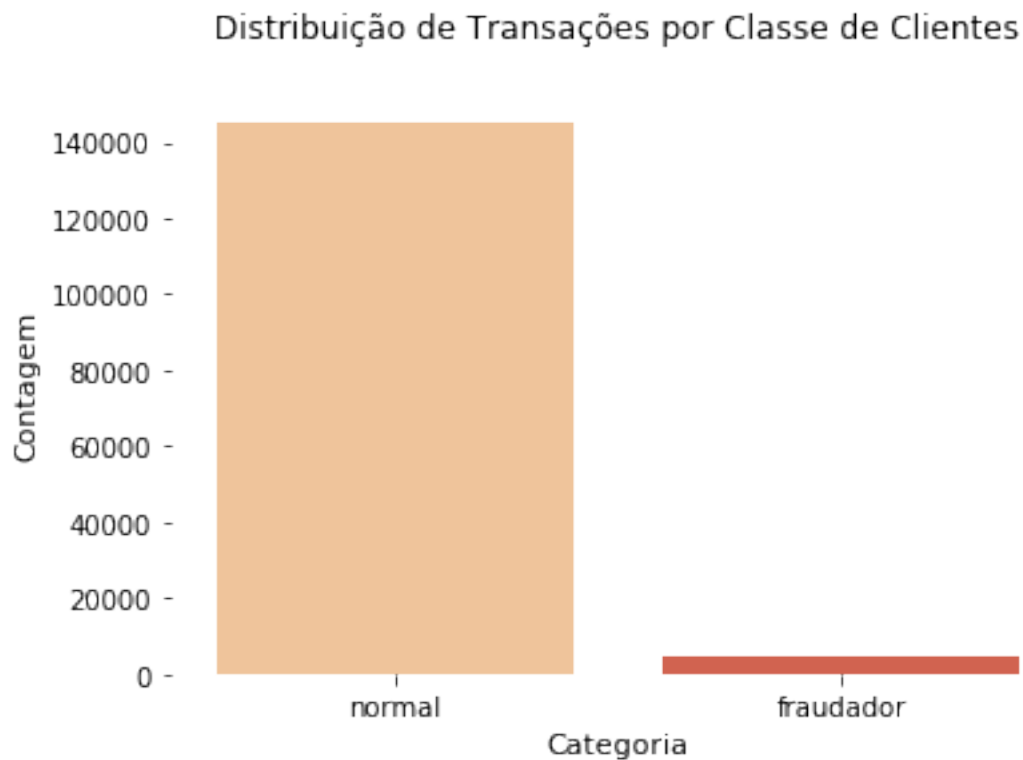
### 3.1.1 Notas

1. As variáveis categóricas estão bem distribuídas (histograma) e sem outliers na distribuição (boxplots), por mais que tenhamos alguns registros de consumo médio mensal mais altos do que a média;
2. Todos os dados serão mantidos a priori.

## 3.2 Qualitativas

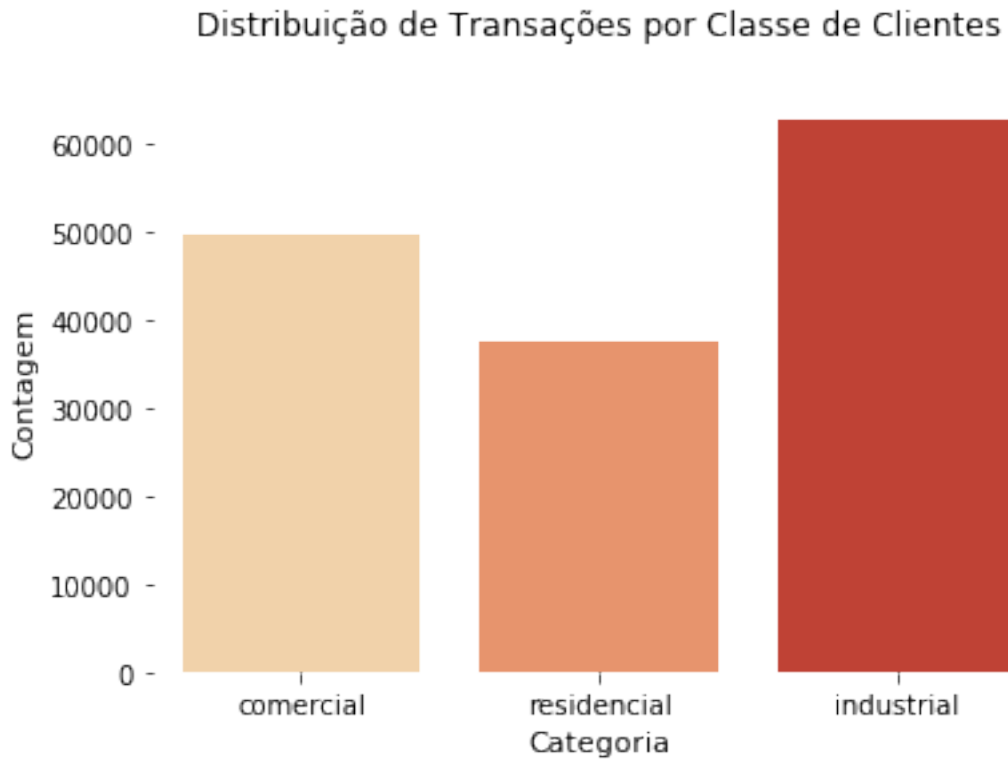
```
[10]: # Plot de contagem por tipo de transação (normal/fraudulenta)
sns.countplot(categorical['classe_cliente'], palette = "OrRd")
plt.box(False)
plt.xlabel('Categoria', fontsize = 11)
plt.ylabel('Contagem', fontsize = 11)
```

```
plt.title('Distribuição de Transações por Classe de Clientes\n')
plt.show()
```



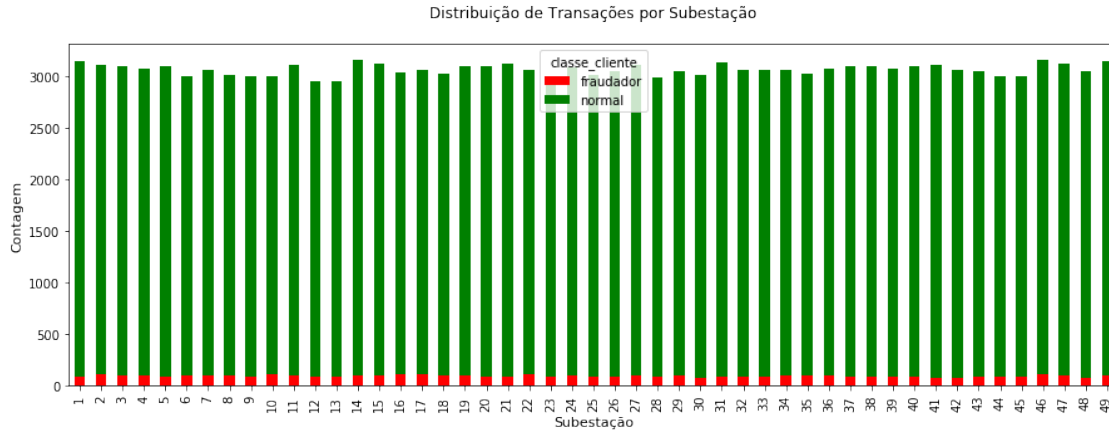
```
[11]: # Plot de contagem por tipo de transação (norma/fraudulenta)
sns.countplot(categorical['local_medidor'], palette = "OrRd")
plt.box(False)
plt.xlabel('Categoria', fontsize = 11)
plt.ylabel('Contagem', fontsize = 11)
plt.title('Distribuição de Transações por Classe de Clientes\n')
plt.show()
```





```
[12]: # Stacked Bar Subestação x Classe do cliente
pd.crosstab(dataset['id_subestacao'], dataset['classe_cliente']).plot(kind = 'bar',
    stacked = True,
    figsize = (15, 5),
    color = ['red', 'green'])

plt.xlabel('Subestação', fontsize = 11)
plt.ylabel('Contagem', fontsize = 11)
plt.title('Distribuição de Transações por Subestação\n')
plt.show()
```



### 3.2.1 Nota

Há um forte desbalanceamento na classe dos clientes, por isso será utilizado técnicas para balancear os dados no dataset (ROSE, UnderSampling e replicação das transações fraudulentas nas subestações afim de balancear o dataset).

## 3.3 Data Wrangling

[13]: `# Criar cópias do dataset original`

```
# dataset_modified = dataset.copy()
dataset_modified2 = dataset.copy()
```

[14]: `# Converte colunas categóricas para 'Categorical' e mapeia em formato de números - [APENAS PARA REGISTRO]`

```
# for col in categorical.columns.tolist():
#     dataset_modified[col] = pd.Categorical(dataset_modified[col]) # Converte para categórico
#     cats = dict(enumerate(dataset_modified[col].cat.categories)) # Mapeia categorias em dict
#     cats = {v: k for k, v in cats.items()} # Inverte par chave/valor do dict
#     dataset_modified[col] = dataset_modified[col].map(cats) # Remapeia base

## Converte colunas para int64
# for col in categorical.columns.tolist():
#     dataset_modified[col] = pd.to_numeric(dataset_modified[col]) # Converte para categórico
# dataset_modified.head()
```

```
[15]: # One Hot Encoding nas variáveis categóricas (conversão do DataFrame para
      ↳ formato wide)

from sklearn.preprocessing import OneHotEncoder

cols = categorical.columns.tolist()
cols.remove('classe_cliente')
display(cols)

# Converte colunas para int64
for col in categorical.columns.tolist():
    dataset[col] = dataset[col].astype(str) # Converte para categórico

dataset_modified2 = dataset.copy()

for col in cols:
    ohe = OneHotEncoder().fit(dataset_modified2[col].values.reshape(-1,1))
    ohe_cols = list(map(lambda val: col + '_' + str(val), ohe.categories_[0]))
    print(ohe_cols)
    ohe_frame = pd.DataFrame(ohe.transform(dataset_modified2[col].values.
↳ reshape(-1,1)).toarray(), columns = ohe_cols)
    dataset_modified2 = pd.concat([dataset_modified2, ohe_frame], join =
↳ 'inner', axis = 1)
    dataset_modified2.drop(columns = {col}, inplace = True)
# Mapear classe target para binário
dataset_modified2['classe_cliente'] = dataset_modified2['classe_cliente'].
↳ map(lambda x: 1 if x == 'fraudador' else 0)
dataset_modified2.head()
```

```
['local_medidor', 'tipo_medidor', 'numero_fases']
```

```
['local_medidor_comercial', 'local_medidor_industrial',
'local_medidor_residencial']
```

```
['tipo_medidor_analogico', 'tipo_medidor_digital']
```

```
['numero_fases_bifasica', 'numero_fases_monofasica', 'numero_fases_trifasica']
```

```
[15]:   id_subestacao  consumo_medio_mensal  temperatura_maxima  \
0           14           293           42
1           30           190           36
2           38           399           31
3           40           307           45
4           15           407           39

      temperatura_minima  classe_cliente  local_medidor_comercial  \
0           10           0           1.00
1           13           0           0.00
2           14           0           0.00
```

3	11	0	0.00
4	19	0	0.00

	local_medidor_industrial	local_medidor_residencial \
0	0.00	0.00
1	0.00	1.00
2	0.00	1.00
3	0.00	1.00
4	0.00	1.00

	tipo_medidor_analogico	tipo_medidor_digital	numero_fases_bifasica \
0	0.00	1.00	0.00
1	0.00	1.00	0.00
2	0.00	1.00	1.00
3	1.00	0.00	0.00
4	0.00	1.00	0.00

	numero_fases_monofasica	numero_fases_trifasica
0	1.00	0.00
1	0.00	1.00
2	0.00	0.00
3	1.00	0.00
4	0.00	1.00

[16]: *# Matriz de correlação de Pearson*

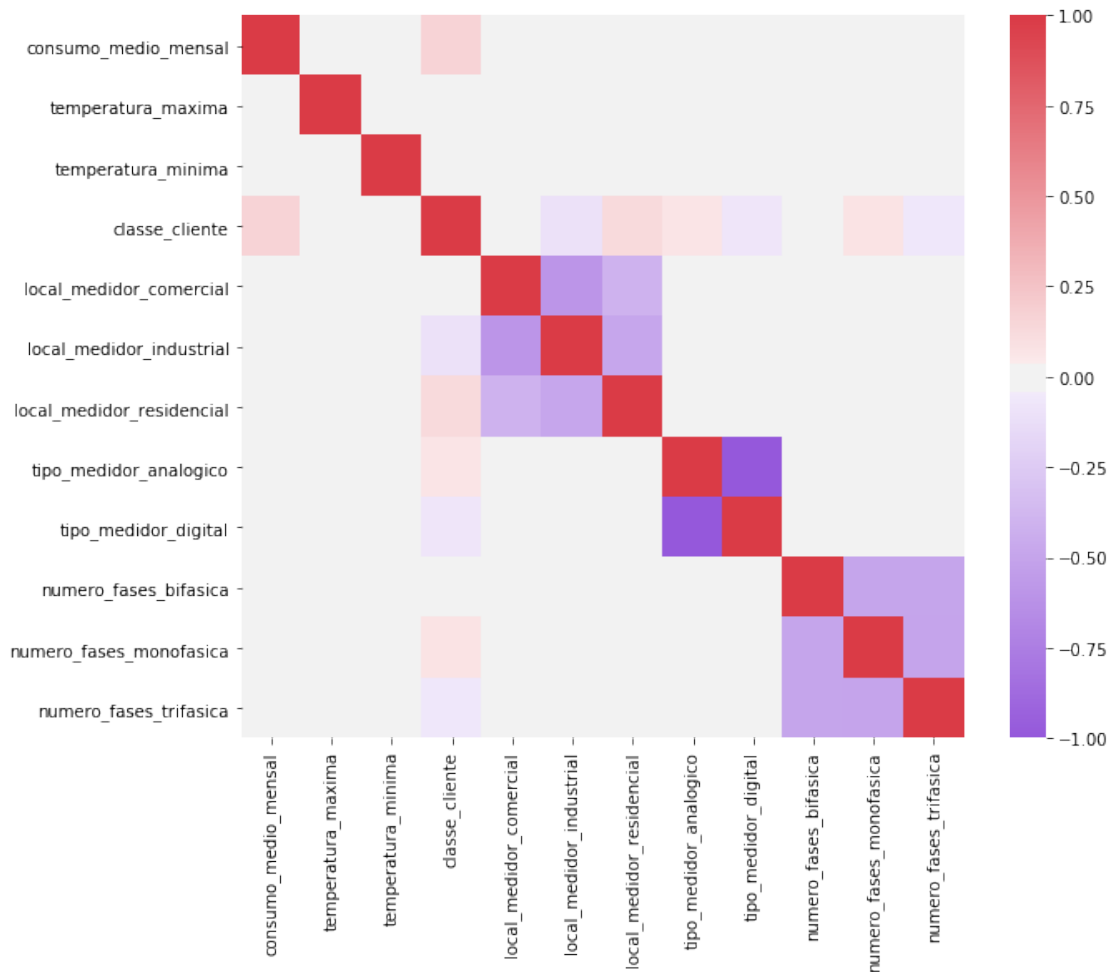
```
corr = dataset_modified2.iloc[:, 1:].corr()
display(corr.style.background_gradient(cmap = 'coolwarm').set_precision(2))
```

*# Visão alternativa com heatmap*

```
f, ax = plt.subplots(figsize = (10, 8))
display(sns.heatmap(corr, mask = np.zeros_like(corr, dtype = np.bool),
                    cmap = sns.diverging_palette(1000, 10, as_cmap = True),
                    square = True, ax = ax))
```

<pandas.io.formats.style.Styler at 0x1c0b48980c8>

<matplotlib.axes.\_subplots.AxesSubplot at 0x1c0b57a66c8>



### 3.4 Data Modelling & Evaluation

#### 3.4.1 Padronizando dados, separando dataset em treino/teste e Cross Validation

```
[17]: # Reordenando coluna target para o final do dataset
dataset_modified2 = pd.concat([dataset_modified2.drop(columns = 'classe_cliente'), \
                                dataset_modified2['classe_cliente'], axis = 1])
```

```
[18]: # Separando variáveis X e Y

array = dataset_modified2.values

X = array[:, 1:-1]
Y = array[:, -1]

# Criando cross-validation score para coletar acurácia ajustada dos modelos
```

```

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# Definindo hiperparâmetros para testes de CV (10 treinamentos para cada modelo)
num_folds = 10
kfold = KFold(num_folds, shuffle = True)

```

```

[19]: # Separando dados entre treino e teste (ratio 80/20, normalizadas e
      ↪ padronizadas)
from sklearn.model_selection import train_test_split

test_size = 0.3
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = test_size)
# X_trainS, X_testS, Y_trainS, Y_testS = train_test_split(X, Y, test_size =
      ↪ test_size)

print(f"NOTA: Base de treino com {len(X_train)} linhas e teste {len(X_test)}
      ↪ linhas.")

```

NOTA: Base de treino com 105000 linhas e teste 45000 linhas.

### 3.4.2 Checando distribuição dos dados com t-SNE e PCA

```

[20]: # # Checando distribuição dos dados com t-SNE scatter plot
      # import matplotlib.patches as mpatches
      # from sklearn.manifold import TSNE
      # X_reduced_tsne = TSNE(n_components = 2).fit_transform(X)

      # f, ax = plt.subplots(figsize=(24,16))

      # blue_patch = mpatches.Patch(color='#0A0AFF', label='Normal')
      # red_patch = mpatches.Patch(color='#AF0000', label='Fraud')

      # ax.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(Y == 0),
      ↪ cmap='coolwarm', label='No Fraud', linewidths=2)
      # ax.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(Y == 1),
      ↪ cmap='coolwarm', label='Fraud', linewidths=2)
      # ax.set_title('t-SNE', fontsize=14)

      # ax.grid(True)

      # ax.legend(handles=[blue_patch, red_patch])

```

```

[21]: # # Checando distribuição dos dados com PCA scatter plot
      # import matplotlib.patches as mpatches

```

```
# from sklearn.decomposition import PCA

# pca = PCA(n_components = 2).fit_transform(X)

# f, ax = plt.subplots(figsize=(24,16))

# blue_patch = mpatches.Patch(color='#0A0AFF', label='No Fraud')
# red_patch = mpatches.Patch(color='#AF0000', label='Fraud')

# ax.scatter(pca[:,0], pca[:,1], c=(Y == 0), cmap='coolwarm', label='No Fraud',
→ linewidths=2)
# ax.scatter(pca[:,0], pca[:,1], c=(Y == 1), cmap='coolwarm', label='Fraud',
→ linewidths=2)
# ax.set_title('PCA', fontsize=14)

# ax.grid(True)

# ax.legend(handles=[blue_patch, red_patch])
```

### 3.4.3 Tratar desbalanceamento da base

```
[22]: # !pip install imbalanced-learn
```

```
[23]: # UnderSampling

# from sklearn.datasets import make_classification
# from imblearn.under_sampling import RandomUnderSampler

# # 50/50
# ros = RandomUnderSampler(sampling_strategy = 1)
# X_over, Y_over = ros.fit_resample(pd.DataFrame(X_train), pd.DataFrame(Y_train.
→ astype('int')))

# ax = Y_over.iloc[:, 0].value_counts().plot.pie(autopct = '%.2f')
# _ = ax.set_title('UnderSampling Shareof - Variável "Target"')
# display(Y_over.iloc[:, 0].value_counts())
```

### 3.4.4 Nota

UnderSampling não gerou um bom resultado, por isso foi comentado (baixa quantidade de casos fraudulentos).

```
[24]: # # OverSampling (ROSE)

# from sklearn.datasets import make_classification
# from imblearn.over_sampling import RandomOverSampler
```

```

# # Oversampling com ratio 50/50
# ros = RandomOverSampler(sampling_strategy = 'not majority')
# X_over, Y_over = ros.fit_resample(pd.DataFrame(X_train), pd.DataFrame(Y_train.
→ astype('int')))

# ax = Y_over.iloc[:, 0].value_counts().plot.pie(autopct = '%.2f')
# _ = ax.set_title('OverSampling Shareof - Variável "Target"')
# display(Y_over.iloc[:, 0].value_counts())

```

```

[25]: # OverSampling (SMOTE)

from imblearn.over_sampling import SMOTE

# Classificação
sm = SMOTE()
X_over, Y_over = sm.fit_resample(X_train, Y_train)

# Plot
ax = pd.DataFrame(Y_over)[0].value_counts().plot.pie(autopct = '%.2f')
_ = ax.set_title('OverSampling Shareof - Variável "Target"')
display(pd.DataFrame(Y_over)[0].value_counts())

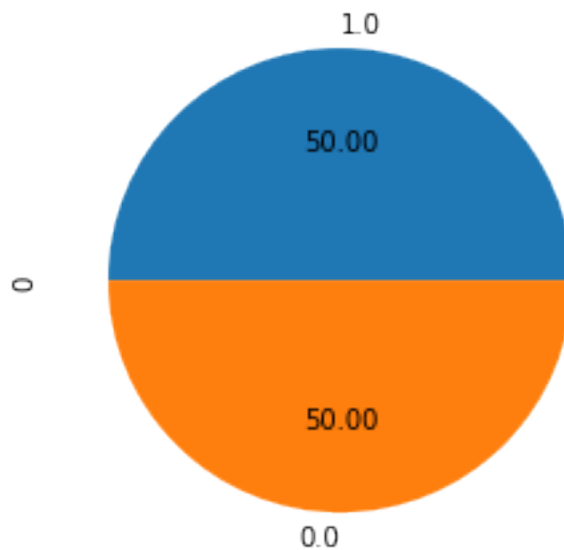
```

```

1.00    101872
0.00    101872
Name: 0, dtype: int64

```

OverSampling Shareof - Variável "Target"





### 3.4.5 Aplicação de Escalas (Normalização e Padronização)

```
[26]: # Normalizando dados com escala MinMax (dados estarão no range de 0 a 1) e ↵
      ↪ padronizacao

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

print("Dados originais: \n\n", X)

scaler = MinMaxScaler(feature_range = (0, 1))

X_train_norm = scaler.fit_transform(X_over)
X_test_norm = scaler.transform(X_test)

print("\nDados normalizados (escala MinMax): \n\n", X_train_norm)

# Padronizando dados (StandardScaler)

scaler = StandardScaler().fit(X_over)

X_train_scaled = scaler.fit_transform(X_over)
X_test_scaled = scaler.transform(X_test)

print("\nDados padronizados (StandardScaler): \n\n", X_train_scaled)
```

Dados originais:

```
[[293.  42.  10. ...  0.   1.   0.]
 [190.  36.  13. ...  0.   0.   1.]
 [399.  31.  14. ...  1.   0.   0.]
 ...
 [381.  34.  17. ...  0.   0.   1.]
 [427.  38.  21. ...  0.   0.   1.]
 [277.  46.  13. ...  0.   1.   0.]]
```

Dados normalizados (escala MinMax):

```
[[0.74832962 0.94736842 0.          ... 0.          0.          1.          ]
 [0.09576837 0.10526316 0.5         ... 1.          0.          0.          ]
 [0.14476615 0.21052632 0.          ... 0.          1.          0.          ]
 ...
 [0.70948626 0.63157895 0.26150041 ... 1.          0.          0.          ]
 [0.57237615 0.10542667 0.21439667 ... 0.          1.          0.          ]
 [0.30859554 0.11685791 0.11138567 ... 0.          1.          0.          ]]
```

Dados padronizados (StandardScaler):

```
[[ 0.99321007  1.47640785 -1.6465928 ... -0.70225033 -0.98539269
  1.84172728]
 [-1.61838534 -1.32017186 -0.00487196 ...  1.65377485 -0.98539269
 -0.60868568]
 [-1.42229285 -0.9705994  -1.6465928 ... -0.70225033  1.18731062
 -0.60868568]
 ...
 [ 0.83775623  0.42769046 -0.78797145 ...  1.65377485 -0.98539269
 -0.60868568]
 [ 0.28903208 -1.31962883 -0.94263383 ... -0.70225033  1.18731062
 -0.60868568]
 [-0.76663623 -1.2816664  -1.28086444 ... -0.70225033  1.18731062
 -0.60868568]]
```

## 3.5 Treinamento dos Algoritmos

### 3.5.1 Regressão Logística

```
[27]: # Regressão logística sem padronização

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Fit e avaliação do modelo
regl = LogisticRegression()
regl.fit(X_over, Y_over)
regl.score(X_over, Y_over)

# Predições
results = cross_val_score(regl, X_over, Y_over, cv = kfold)
predictions = regl.predict(X_test)

# Score
print("Acurácia (Regressão Logística): %.3f" % (results.mean() * 100))
display(print(classification_report(Y_test, predictions)))
```

Acurácia (Regressão Logística): 77.885

	precision	recall	f1-score	support
0.0	0.99	0.77	0.87	43611
1.0	0.09	0.76	0.17	1389
accuracy			0.77	45000
macro avg	0.54	0.76	0.52	45000
weighted avg	0.96	0.77	0.84	45000

None

```
[28]: # Regressão logística com padronização

# Fit e avaliação do modelo
regrl = LogisticRegression()
regrl.fit(X_train_scaled, Y_over)
regrl.score(X_train_scaled, Y_over)

# Predições
results = cross_val_score(regrl, X_train_scaled, Y_over, cv = kfold)
predictions = regrl.predict(X_test_scaled)

# Score
print("Acurácia (Regressão Logística): %.3f" % (results.mean() * 100))
display(print(classification_report(Y_test, predictions)))
```

Acurácia (Regressão Logística): 77.871

	precision	recall	f1-score	support
0.0	0.99	0.77	0.86	43611
1.0	0.09	0.76	0.17	1389
accuracy			0.77	45000
macro avg	0.54	0.76	0.52	45000
weighted avg	0.96	0.77	0.84	45000

None

### 3.5.2 Nota

Acurácia não é uma boa métrica para este problema, visto que ele prevê praticamente todos os casos de não fraude enquanto que a precisão dos casos fraudadores é inferior à 10%, por isso este modelo não será utilizado.

### 3.5.3 KNN

```
[29]: from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error
from sklearn.metrics import precision_score, recall_score
import math

alpha = [n for n in range(3, 50, 2)]
argmax = 0
```

```

best_param = int

for param in alpha:
    model = KNeighborsClassifier(n_neighbors = param)
    knn = model.fit(X_train_norm, Y_over)
    preds = knn.predict(X_test_norm)
    score = precision_score(Y_test, preds, average = None)[1]
    if score > argmax:
        argmax = score
        best_param = param

# Apresentar melhor parâmetro
print(best_param, argmax)

# Fit no modelo com melhor parâmetro e avaliação (score)

model = KNeighborsClassifier(n_neighbors = best_param)
knn = model.fit(X_train_norm, Y_over)
preds = knn.predict(X_test_norm)

print(precision_score(Y_test, preds, average = None)[0])
print(confusion_matrix(Y_test, preds))
print(classification_report(Y_test, preds))

```

3 0.19792117799913383

0.9781687006629032

[[41759 1852]

[ 932 457]]

	precision	recall	f1-score	support
0.0	0.98	0.96	0.97	43611
1.0	0.20	0.33	0.25	1389
accuracy			0.94	45000
macro avg	0.59	0.64	0.61	45000
weighted avg	0.95	0.94	0.95	45000

### 3.5.4 Notas

1. O modelo não desempenhou bem também, mesmo que melhor em comparação à Regressão Logística. A taxa de precisão para fraudes foi baixa (aprox. 18%);
2. Foi processado um range de K-vizinhos ímpares entre 3 e 49 (limitei ao número de subestações).

### 3.5.5 Random Forest com GridSearchCV

```
[30]: # Grid Search

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import average_precision_score, roc_auc_score, f1_score

num_trees = 1000
random_forest = RandomForestClassifier(n_estimators = num_trees)

# Usando um grid completo de todos os parâmetros
param_grid = {"max_depth": [5, None],
              "max_features": [5, 7],
              "criterion": ["gini", "entropy"]}

# Executando o Grid Search com média de precisão como métrica a ser avaliada
grid_search = GridSearchCV(random_forest, param_grid = param_grid,
    ↪return_train_score = True, \
                          scoring = 'average_precision', n_jobs = 4)

# Previsões
grid_search.fit(X_over, Y_over)
predictions = grid_search.predict(X_test)
display(print(classification_report(Y_test, predictions)))

# Coletando melhores parâmetros do GridSearchCV e treinando um novo modelo
best_params = grid_search.best_params_
best_params['n_estimators'] = num_trees

display(best_params)

random_forest = RandomForestClassifier(**best_params)
random_forest.fit(X_over, Y_over)

# Previsões
predictions = random_forest.predict(X_test)
print("Precisão em Teste:", precision_score(Y_test, predictions, average =
    ↪None))
display(print(classification_report(Y_test, predictions)))
```

	precision	recall	f1-score	support
0.0	0.98	0.99	0.99	43611
1.0	0.63	0.32	0.42	1389
accuracy			0.97	45000
macro avg	0.81	0.66	0.70	45000

weighted avg	0.97	0.97	0.97	45000
--------------	------	------	------	-------

None

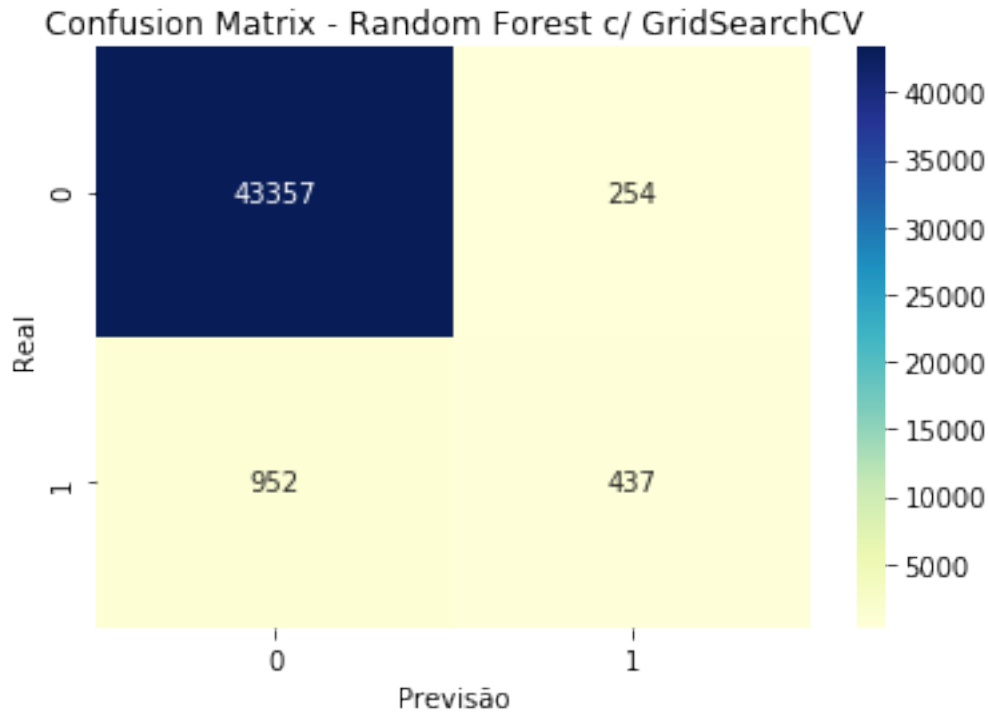
```
{'criterion': 'gini',  
 'max_depth': None,  
 'max_features': 5,  
 'n_estimators': 1000}
```

Precisão em Teste: [0.97851452 0.63241679]

	precision	recall	f1-score	support
0.0	0.98	0.99	0.99	43611
1.0	0.63	0.31	0.42	1389
accuracy			0.97	45000
macro avg	0.81	0.65	0.70	45000
weighted avg	0.97	0.97	0.97	45000

None

```
[31]: # Gerando Confusion Matrix  
confusion_matrix = pd.DataFrame(confusion_matrix(Y_test, predictions))  
sns.heatmap(confusion_matrix, annot = True, cmap = "YlGnBu", fmt = '2g')  
  
plt.title('Confusion Matrix - Random Forest c/ GridSearchCV')  
plt.xlabel('Previsão')  
plt.ylabel('Real')  
plt.show()
```



```
[32]: # Curva ROC e F1 Score
print('ROCAUC: ', roc_auc_score(Y_test, predictions))
print('F1: ', f1_score(Y_test, predictions))
```

ROCAUC: 0.6543953060765526  
F1: 0.42019230769230764

### 3.5.6 Notas

O Grid Search foi criado com quatro parâmetros: - `n_estimators/trees` - A tendência é ter um modelo melhor com mais estimadores, mas com tempo de processamento também mais alto; - `max depth` - Determina a “profundidade” da árvore. Um valor-teto pode ser colocado afim de eliminar overfitting; - `max_features` - Determina a quantidade de atributos a serem desmembrados nas random forests; - `criterion` - Determina como o algoritmo fará a divisão dos atributos e a classificação dos nós em cada árvore;

Finalmente temos um modelo com melhor desempenho em relação aos demais, mantendo a precisão mais alta nas duas classes.

### 3.5.7 Gradient Boosting Classifier com GridSearchCV

```
[33]: # Grid de parâmetros (usando hiperparâmetros do último treinamento)
from sklearn.ensemble import GradientBoostingClassifier

param_grid = {'learning_rate': [0.1, 0.01],
```

```

        'max_depth': [None],
        'max_features': [5],
        'min_samples_leaf': [3, 4],
        'subsample': [0.5, 0.7],
        'n_estimators': [1000]
    }

    # Regressor
    random_forest_gb = GradientBoostingClassifier()

    # Modelo criado com GridSearchCV
    gs_cv = GridSearchCV(random_forest_gb, param_grid, scoring =_
        ↪ 'average_precision', n_jobs = 4).fit(X_over, Y_over)

    # Imprime os melhores parâmetros
    print('Melhores Hiperparâmetros: %r' % gs_cv.best_params_)

```

Melhores Hiperparâmetros: {'learning\_rate': 0.1, 'max\_depth': None, 'max\_features': 5, 'min\_samples\_leaf': 3, 'n\_estimators': 1000, 'subsample': 0.7}

```

[36]: # Treinando o modelo com melhores parâmetros
random_forest_gb = GradientBoostingClassifier(**gs_cv.best_params_)
random_forest_gb.fit(X_over, Y_over)

# Previsões
predictions = random_forest_gb.predict(X_test)
print("Precisão em Teste:", precision_score(Y_test, predictions, average =_
    ↪ None))
display(print(classification_report(Y_test, predictions)))

```

Precisão em Teste: [0.97830611 0.54156171]

	precision	recall	f1-score	support
0.0	0.98	0.99	0.98	43611
1.0	0.54	0.31	0.39	1389
accuracy			0.97	45000
macro avg	0.76	0.65	0.69	45000
weighted avg	0.96	0.97	0.97	45000

None

### 3.5.8 Nota

O GradientBoosting não performou tão bem quanto a primeira versão do modelo, provavelmente precisaria aumentar e testar com novos hiperparâmetros para o modelo (o que é computacional-



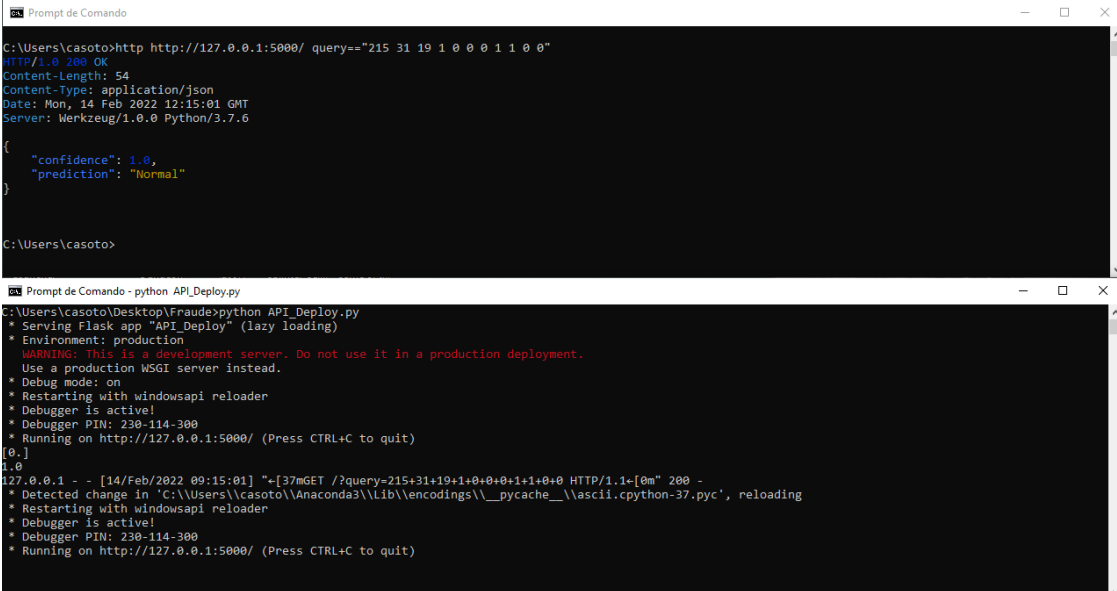
mente mais pesado e com tempo de processamento maior).

### 3.6 Model Deployment

```
[42]: # Coletando modelo de random forest com GridSearchCV e gerando arquivo .sav
      ↪(picke)
import pickle
f = 'classificador.sav'
pickle.dump(random_forest, open(f, 'wb'))
```

```
[46]: # Teste de integração com input de linha (script avulso API_Deploy.py)
from IPython.display import Image
Image('API_test.png')
```

```
[46]:
```



```
C:\Users\casoto>http http://127.0.0.1:5000/ query="215 31 19 1 0 0 0 1 1 0 0"
HTTP/1.0 200 OK
Content-Length: 54
Content-Type: application/json
Date: Mon, 14 Feb 2022 12:15:01 GMT
Server: Werkzeug/1.0.0 Python/3.7.6

{
  "confidence": 1.0,
  "prediction": "Normal"
}

C:\Users\casoto>
```

```
C:\Users\casoto\Desktop\Fraude>python API_Deploy.py
* Serving Flask app "API_Deploy" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 230-114-300
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
1.0
127.0.0.1 - - [14/Feb/2022 09:15:01] "[37mGET /?query=215+31+19+1+0+0+0+1+1+0+0 HTTP/1.1-[0m" 200 -
* Detected change in 'C:\Users\casoto\Anaconda3\Lib\encodings\__pycache__\ascii.cpython-37.pyc', reloading
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 230-114-300
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

## 4 Fim

### 4.0.1 SVM [comentado]

```
[ ]: # # SVM com Kernel Linear + SMOTE

# from sklearn import svm
# from sklearn.metrics import precision_score, recall_score, f1_score,
  ↪accuracy_score, roc_auc_score
# from sklearn.decomposition import PCA

# # Treina o Modelo
```

```

# sum_v1 = svm.SVC(kernel = 'linear', class_weight = 'balanced', probability = True)
# sum_v1.fit(X_over, Y_over)
# pred_sum_v1 = sum_v1.predict(X_test)

# # Avaliação
# sum_dict_v1 = {'Modelo': 'SVM',
#               'Versão': '1',
#               'Kernel': 'Linear sem Padronização',
#               'Precision': precision_score(pred_sum_v1, Y_test),
#               'Recall': recall_score(pred_sum_v1, Y_test),
#               'F1 Score': f1_score(pred_sum_v1, Y_test),
#               'Acurácia': accuracy_score(pred_sum_v1, Y_test),
#               'AUC': roc_auc_score(Y_test, pred_sum_v1)}

# display(sum_dict_v1)
# display(print(classification_report(Y_test, pred_sum_v1)))

```

```

[ ]: # SVM com Kernel linear + sem dados padronizados (originais)

# from sklearn import svm
# from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, roc_auc_score
# from sklearn.decomposition import PCA

# # Treina o Modelo
# sum_v1 = svm.SVC(kernel = 'linear', class_weight = 'balanced', probability = True)
# sum_v1.fit(X_over, Y_over)
# pred_sum_v1 = sum_v1.predict(X_test)

# # Avaliação
# sum_dict_v1 = {'Modelo': 'SVM',
#               'Versão': '1',
#               'Kernel': 'Linear sem Padronização',
#               'Precision': precision_score(pred_sum_v1, Y_test),
#               'Recall': recall_score(pred_sum_v1, Y_test),
#               'F1 Score': f1_score(pred_sum_v1, Y_test),
#               'Acurácia': accuracy_score(pred_sum_v1, Y_test),
#               'AUC': roc_auc_score(Y_test, pred_sum_v1)}

# display(sum_dict_v1)
# display(print(classification_report(Y_test, pred_sum_v1)))

```

```

[ ]: # # SVM com Kernel Linear + Scaling dos dados

```

```

# sum_v2 = svm.SVC(kernel = 'linear')

```

```

# sum_v2.fit(X_train_scaled, Y_over)
# pred_sum_v2 = sum_v1.predict(X_test_scaled)

# # Avaliação
# sum_dict_v2 = {'Modelo': 'SVM',
#               'Versão': '2',
#               'Kernel': 'Linear com Padronização',
#               'Precision': precision_score(pred_sum_v2, Y_test),
#               'Recall': recall_score(pred_sum_v2, Y_test),
#               'F1 Score': f1_score(pred_sum_v2, Y_test),
#               'Acurácia': accuracy_score(pred_sum_v2, Y_test),
#               'AUC': roc_auc_score(Y_test, pred_sum_v2)}

# display(sum_dict_v2)
# display(print(classification_report(Y_test, pred_sum_v2)))

```

```

[ ]: # SVM com Kernel RBF e GridSearchCV

# from sklearn.model_selection import GridSearchCV

# sum_v3 = sum.SVC(kernel = 'rbf')

# # Valores para o grid
# C_range = np.array([50., 100., 200.])
# gamma_range = np.array([0.3*0.001, 0.001, 3*0.001])

# # Grid de hiperparâmetros
# sum_param_grid = dict(gamma = gamma_range, C = C_range)

# # Grid Search
# # start = time.time()
# sum_v3_grid_search_rbf = GridSearchCV(sum_v3, sum_param_grid, cv = 3)

# # Treinamento
# sum_v3_grid_search_rbf.fit(X_train_scaled, Y_over)
# # end = time.time()
# # print('Tempo de Treinamento do Modelo com Grid Search:', end - start)

# # Acurácia em Treino
# print(f"Acurácia em Treinamento: {sum_v3_grid_search_rbf.best_score_ :.2%}",
#       end = '\n')
# print(f"Hiperparâmetros Ideais: {sum_v3_grid_search_rbf.best_params_}")

# # Previsões
# pred_sum_v3 = sum_v3_grid_search_rbf.predict(X_test_scaled)

# # Dicionário de métricas e metadados

```

```

# sum_dict_v3 = {'Modelo': 'SVM',
#               'Versão': '3',
#               'Kernel': 'RBF com GridSearchCV',
#               'Precision': precision_score(pred_sum_v3, Y_test),
#               'Recall': recall_score(pred_sum_v3, Y_test),
#               'F1 Score': f1_score(pred_sum_v3, Y_test),
#               'Acurácia': accuracy_score(pred_sum_v3, Y_test),
#               'AUC': roc_auc_score(Y_test, pred_sum_v3)}

# display(sum_dict_v3)
# display(print(classification_report(Y_test, pred_sum_v3)))

```

```

[ ]: # # SVM com Kernel Polinomial + GridSearchCV

# # Cria o modelo
# sum_v4 = svm.SVC(kernel = 'poly')

# # Valores para o grid
# r_range = np.array([0.5, 1])
# gamma_range = np.array([0.0001, 0.001, 0.01, .05, 0.1, 0.01, 1, 2])
# d_range = np.array([2, 3, 4])

# # Grid de hiperparâmetros
# param_grid_poly = dict(gamma = gamma_range, degree = d_range, coef0 = r_range)

# # Grid Search
# # start = time.time()
# sum_v4_grid_search_poly = GridSearchCV(sum_v4, param_grid_poly, cv = 3)

# # Treinamento
# sum_v4_grid_search_poly.fit(X_train_scaled, Y_over)
# # end = time.time()
# # print('Tempo de Treinamento do Modelo com Grid Search:', end - start)

# # Acurácia em Treino
# print(f"Acurácia em Treinamento: {sum_v4_grid_search_poly.best_score_ :.2%}",
#       end = '\n')
# print(f"Hiperparâmetros Ideais: {sum_v4_grid_search_poly.best_params_}")

# # Previsões
# pred_sum_v4 = sum_v4_grid_search_poly.predict(X_test_scaled)

# # Dicionário de métricas e metadados
# sum_dict_v4 = {'Modelo': 'SVM',
#               'Versão': '4',
#               'Kernel': 'Polinomial com Dados Padronizados',
#               'Precision': precision_score(pred_sum_v4, Y_test),

```

```
#          'Recall':recall_score(pred_sum_v4, Y_test),
#          'F1 Score':f1_score(pred_sum_v4, Y_test),
#          'Acurácia':accuracy_score(pred_sum_v4, Y_test),
#          'AUC':roc_auc_score(Y_test, pred_sum_v4)}

# display(sum_dict_v4)
# display(print(classification_report(Y_test, pred_sum_v4)))
```

```
[ ]: # # Concatena todos os dicionários em um dataframe do Pandas
# resumo = pd.DataFrame({'SVM_dict_v1':pd.Series(SVM_dict_v1),
#                        'SVM_dict_v2':pd.Series(SVM_dict_v2),
#                        'SVM_dict_v3':pd.Series(SVM_dict_v3),
#                        'SVM_dict_v4':pd.Series(SVM_dict_v4)})
# resumo
```

```
[43]: X_test[0]
```

```
[43]: array([215.,  31.,  19.,   1.,   0.,   0.,   0.,   1.,   1.,   0.,   0.]
```

## 5 Fim