Detecção de fraude em cartão de crédito

Contexto

É importante que as empresas de cartão de crédito sejam capazes de reconhecer transações fraudulentas com cartão de crédito para que os clientes não sejam cobrados por itens que não compraram.

Conteúdo

O conjunto de dados contém transações feitas por cartões de crédito em setembro de 2013 por titulares de cartões europeus.

Este conjunto de dados apresenta transações ocorridas em dois dias, onde temos 492 fraudes em 284.807 transações. O conjunto de dados é altamente desbalanceado, a classe positiva (fraudes) responde por 0,172% de todas as transações.

Ele contém apenas variáveis de entrada numéricas que são o resultado de uma transformação PCA. Infelizmente, devido a questões de confidencialidade, não podemos fornecer os recursos originais e mais informações básicas sobre os dados. As características V1, V2, … V28 são os principais componentes obtidos com PCA, as únicas características que não foram transformadas com PCA são 'Time' e 'Amount'. O recurso 'Tempo' contém os segundos decorridos entre cada transação e a primeira transação no conjunto de dados. O recurso 'Valor' é o valor da transação, esse recurso pode ser usado para aprendizado sensível ao custo dependente de exemplo. A característica 'Classe' é a variável de resposta e assume valor 1 em caso de fraude e 0 caso contrário.

Dada a taxa de desequilíbrio de classe, recomendamos medir a precisão usando a Área sob a Curva de Rechamada de Precisão (AUPRC). A precisão da matriz de confusão não é significativa para a classificação desbalanceada.

In [ ]:
```
import math
import pandas as pd
import graphviz
import pydot
import chart_studio.plotly as py
import plotly.figure_factory as ff
import seaborn as sea
import matplotlib.pyplot as plt
from   wand.image import Image as WImage
```

In [ ]:
```
data = pd.read_csv(r'dados/creditcard.csv',sep=',')
data.head(4)
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575

4 rows × 31 columns

In [ ]:
```
data.shape
```

Out[ ]:
```
(284807, 31)
```

Analise Exploratoria de dados

1. Usei data.info() para me informa se no dataframe possui valores nulo eo tipo do atributo
2. Usei o .describe() no qual ele me statísticas descritiva cada atributo. Seaborn para plotar
3. Correlação de dados os que me deram maior correlacao foi positiva V7,V8 em relacao ao Amount
4. Devido possui muitos dados o pairplot demorava muito para renderizar, então peguei os atributos de maior correlacao.
5. Fiz a media dos atributos em relação Class
6. statísticas descritiva em relação ao atributoo Class usando plotly

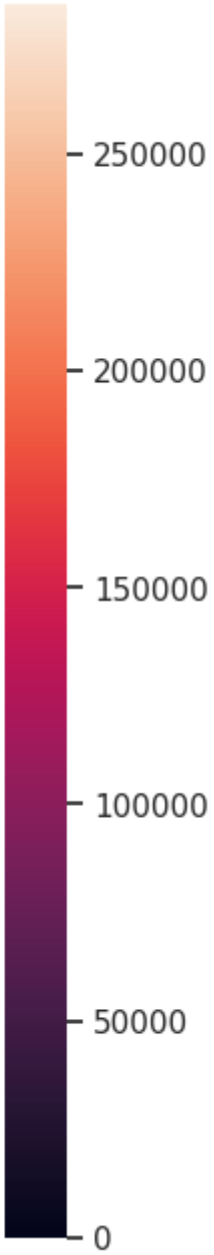
In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [ ]:
```
plt.figure(figsize=(10,8))
sea.set_theme(style="whitegrid")
sea.heatmap(data.describe().T,annot=True,robust=True)
```

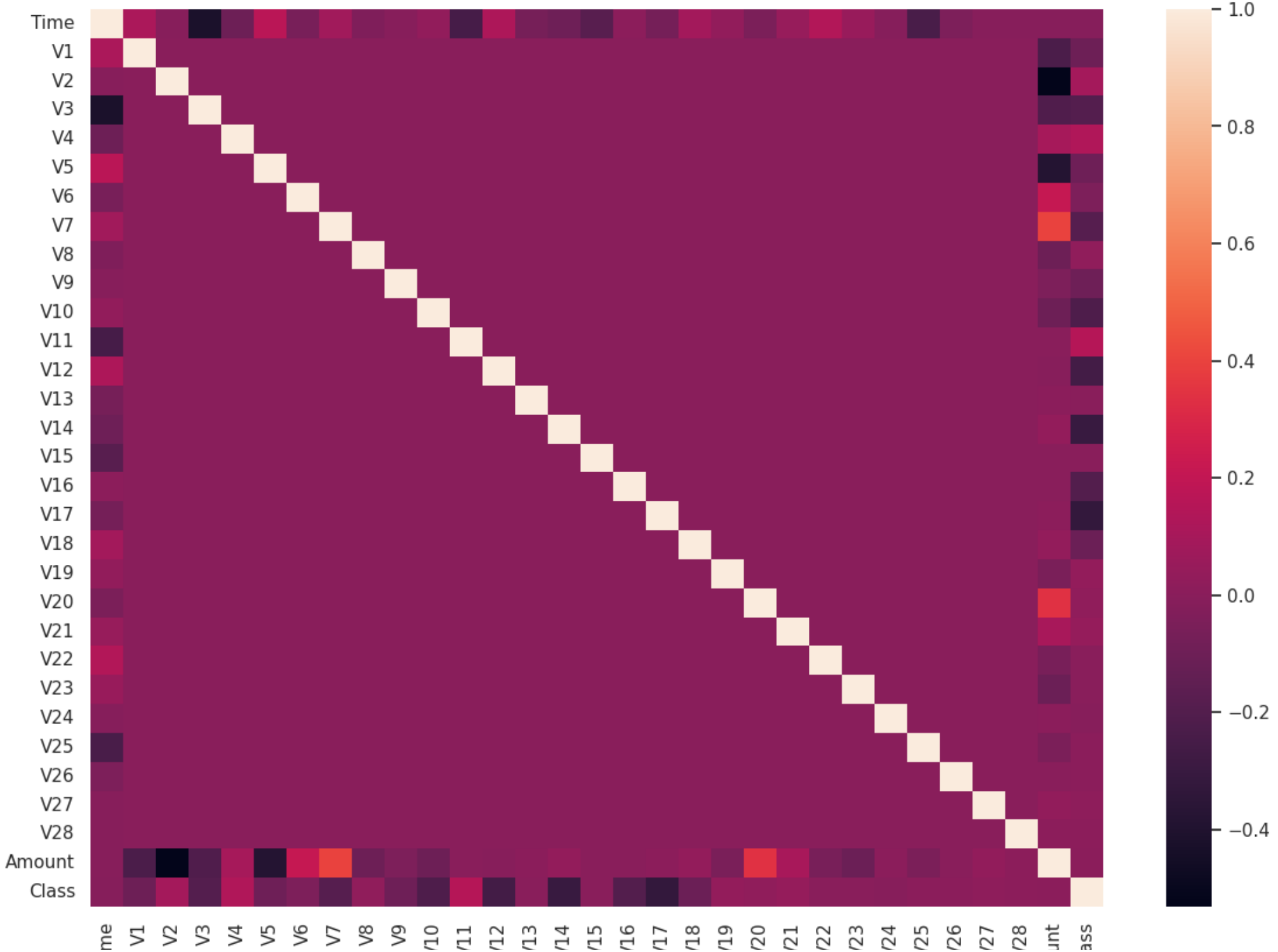
Out[ ]: <Axes: >

	count	mean	std	min	25%	50%	75%	max
Time	2.8e+05	9.5e+04	4.7e+04	0	5.4e+04	8.5e+04	1.4e+05	1.7e+05
V1	2.8e+05	1.2e-15	2	-56	-0.92	0.018	1.3	2.5
V2	2.8e+05	3.4e-16	1.7	-73	-0.6	0.065	0.8	22
V3	2.8e+05	-1.4e-15	1.5	-48	-0.89	0.18	1	9.4
V4	2.8e+05	2.1e-15	1.4	-5.7	-0.85	-0.02	0.74	17
V5	2.8e+05	9.6e-16	1.4	-1.1e+02	-0.69	-0.054	0.61	35
V6	2.8e+05	1.5e-15	1.3	-26	-0.77	-0.27	0.4	73
V7	2.8e+05	-5.6e-16	1.2	-44	-0.55	0.04	0.57	1.2e+02
V8	2.8e+05	1.2e-16	1.2	-73	-0.21	0.022	0.33	20
V9	2.8e+05	-2.4e-15	1.1	-13	-0.64	-0.051	0.6	16
V10	2.8e+05	2.2e-15	1.1	-25	-0.54	-0.093	0.45	24
V11	2.8e+05	1.7e-15	1	-4.8	-0.76	-0.033	0.74	12
V12	2.8e+05	-1.2e-15	1	-19	-0.41	0.14	0.62	7.8
V13	2.8e+05	8.2e-16	1	-5.8	-0.65	-0.014	0.66	7.1
V14	2.8e+05	1.2e-15	0.96	-19	-0.43	0.051	0.49	11
V15	2.8e+05	4.9e-15	0.92	-4.5	-0.58	0.048	0.65	8.9
V16	2.8e+05	1.4e-15	0.88	-14	-0.47	0.066	0.52	17
V17	2.8e+05	-3.8e-16	0.85	-25	-0.48	-0.066	0.4	9.3
V18	2.8e+05	9.6e-16	0.84	-9.5	-0.5	-0.0036	0.5	5
V19	2.8e+05	1e-15	0.81	-7.2	-0.46	0.0037	0.46	5.6
V20	2.8e+05	6.4e-16	0.77	-54	-0.21	-0.062	0.13	39
V21	2.8e+05	1.7e-16	0.73	-35	-0.23	-0.029	0.19	27
V22	2.8e+05	-3.6e-16	0.73	-11	-0.54	0.0068	0.53	11
V23	2.8e+05	2.6e-16	0.62	-45	-0.16	-0.011	0.15	23
V24	2.8e+05	4.5e-15	0.61	-2.8	-0.35	0.041	0.44	4.6
V25	2.8e+05	5.3e-16	0.52	-10	-0.32	0.017	0.35	7.5
V26	2.8e+05	1.7e-15	0.48	-2.6	-0.33	-0.052	0.24	3.5
V27	2.8e+05	-3.7e-16	0.4	-23	-0.071	0.0013	0.091	32
V28	2.8e+05	-1.2e-16	0.33	-15	-0.053	0.011	0.078	34
Amount	2.8e+05	88	2.5e+02	0	5.6	22	77	2.6e+04
Class	2.8e+05	0.0017	0.042	0	0	0	0	1



In [ ]:
```
plt.figure(figsize=(14,10))
sea.heatmap(data.corr())
```

Out[ ]: <Axes: >

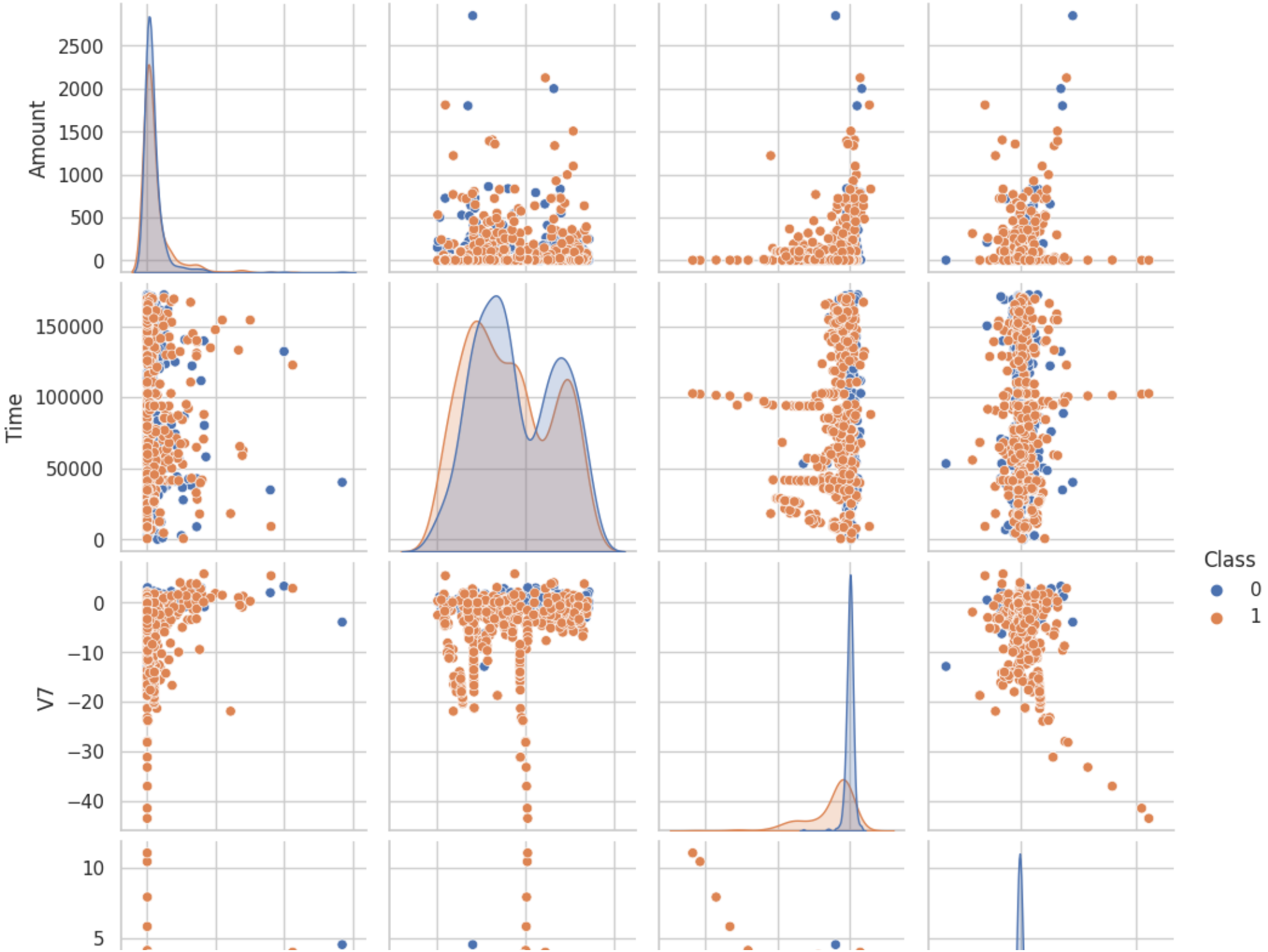

Time
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
Amount
Class
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

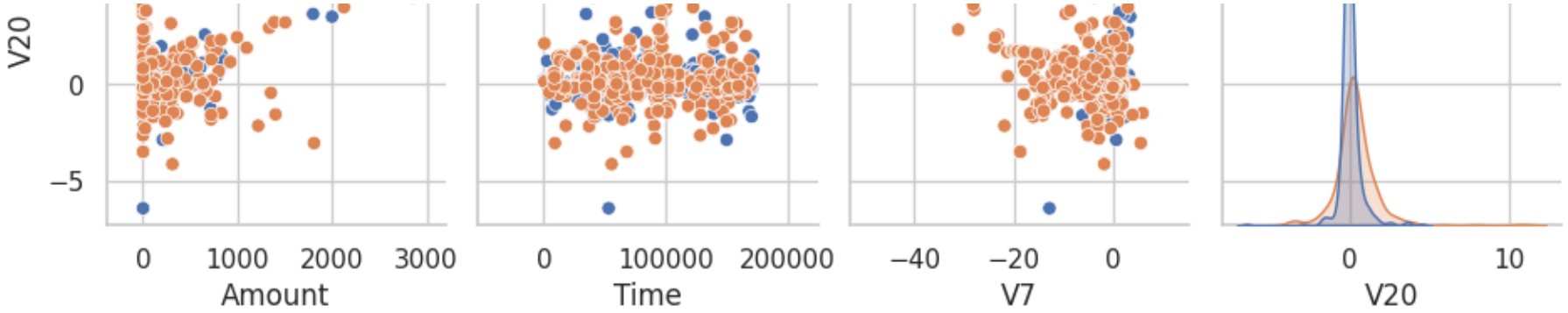
In [ ]:
```
df1 = (data[['Amount','Time','Class','V7','V20']]).loc[data['Class']==0]
df2 = (data[['Amount','Time','Class','V7','V20']]).loc[data['Class']==1]

df1 = df1.sample(480,random_state=50)
df2 = df2.sample(480)
data_2 = pd.concat([df1,df2],axis=0)


plt.figure(figsize=(12,8))
sea.set_theme(style="whitegrid")
sea.pairplot(data_2,hue='Class')
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f4b913b5410>
<Figure size 1200x800 with 0 Axes>
```


Amount
Time
V7
2500
2000
1500
1000
500
0
150000
100000
50000
0
0
−10
−20
−30
−40
10
5
Class
0
1



In [ ]:
```
data.groupby('Class').mean()
```

Out[ ]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.000644	-0.001235	-0.000
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014

2 rows × 30 columns

In [ ]:
```
columns = data.groupby('Class').mean().columns.to_list()
```

In [ ]:
```
for i in columns:
    table = ff.create_table(data.groupby('Class')[i].describe(),index_title='class/'+i,index=True)
    table.show()
```

Devido as os tipo dos atributos serem do tipo numerico e a 'Classe' ser binaria não precisei fazer tratamento dos dados

Usei a regressao logistica , Naive bayes

In [ ]:
```
from sklearn.metrics         import confusion_matrix,ConfusionMatrixDisplay,accuracy_score,classification_report,roc_auc_score,
from sklearn.model_selection import train_test_split
from sklearn.linear_model    import LogisticRegression
from sklearn.naive_bayes     import GaussianNB
from sklearn.preprocessing   import StandardScaler
```

In [ ]:
```
X = data.drop(columns='Class').values
Y = data['Class'].values
```

Regressão Logistica

In [ ]:
```
x_train , x_test , y_train , y_test = train_test_split(X,Y,test_size=0.30)
logistic = LogisticRegression()
```

In [ ]:
```
y_pred = logistic.fit(x_train,y_train).predict(x_test)
```

```
/home/mateus/MEGA/Matérias UFC CD/Introdução a Mineração de Dados/venv/lib/python3.11/site-packages/sklearn/linear_model/_logist
ic.py:458: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

In [ ]:
```
print(classification_report(y_test,y_pred))
```

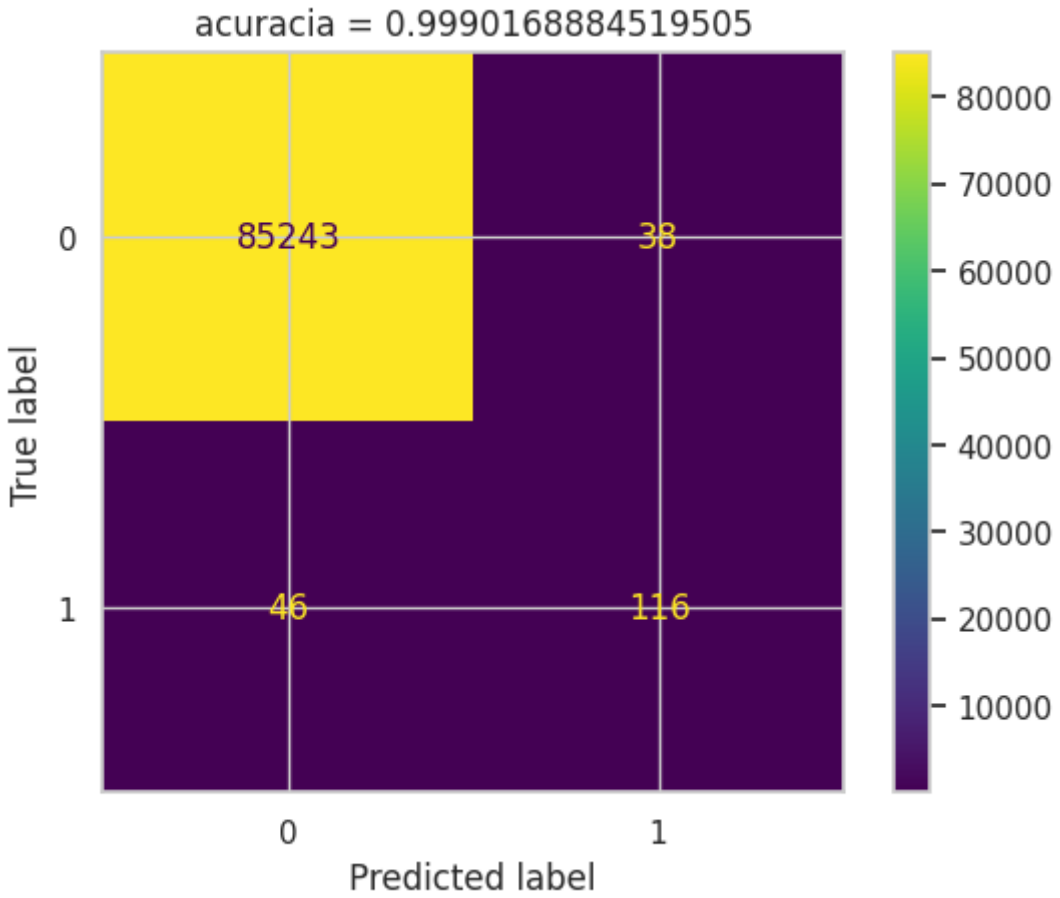
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85281
           1       0.75      0.72      0.73       162

    accuracy                           1.00     85443
   macro avg       0.88      0.86      0.87     85443
weighted avg       1.00      1.00      1.00     85443
```

In [ ]:
```
cm = confusion_matrix(y_test,y_pred)
ConfusionMatrixDisplay(cm).plot()
plt.title(f'acuracia = {accuracy_score(y_test,y_pred)}')
```

Out[ ]:
```
Text(0.5, 1.0, 'acuracia = 0.9990168884519505')
```

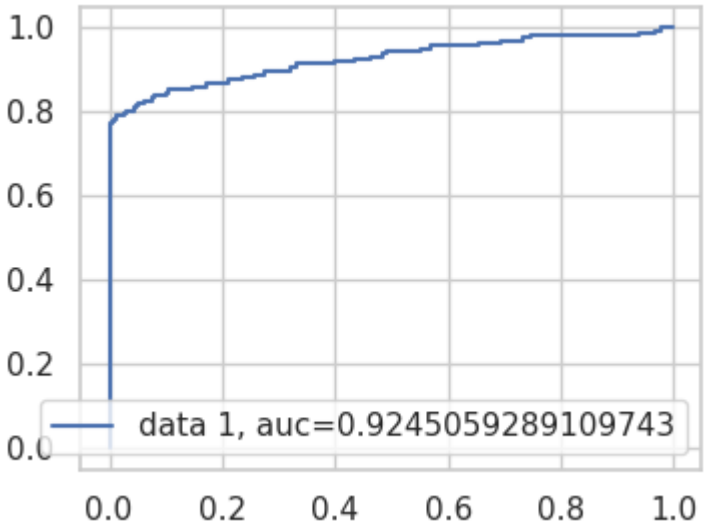


desempenho do modelos de classificação

resultado bom cima de 80%

In [ ]:
```
plt.figure(figsize=(4,3))
y_pred_prob    = logistic.predict_proba(x_test)[::,1]
fpr, tpr, null = roc_curve(y_test,   y_pred_prob)
auc            = roc_auc_score(y_test,y_pred_prob)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f4b8f60d250>
```



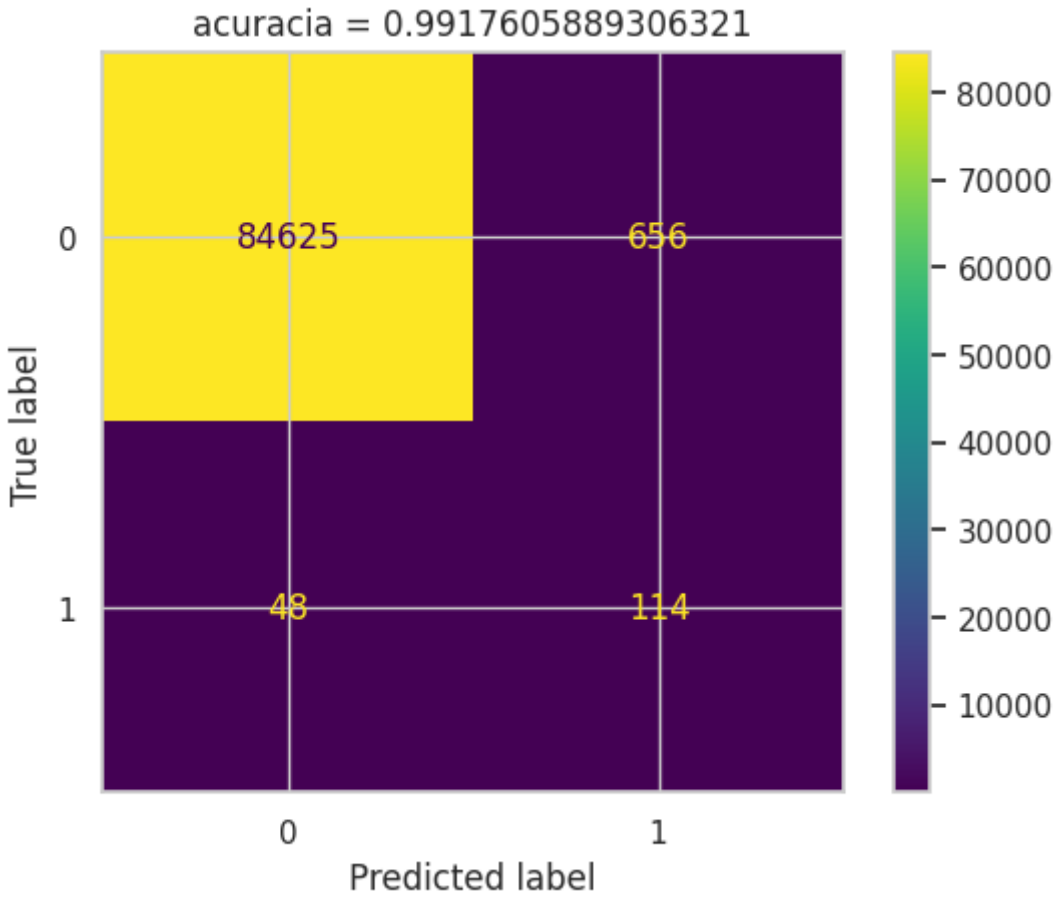
Naive Bayes

In [ ]:
```
naive = GaussianNB()
```

In [ ]:
```
y_predG = naive.fit(x_train,y_train).predict(x_test)
```

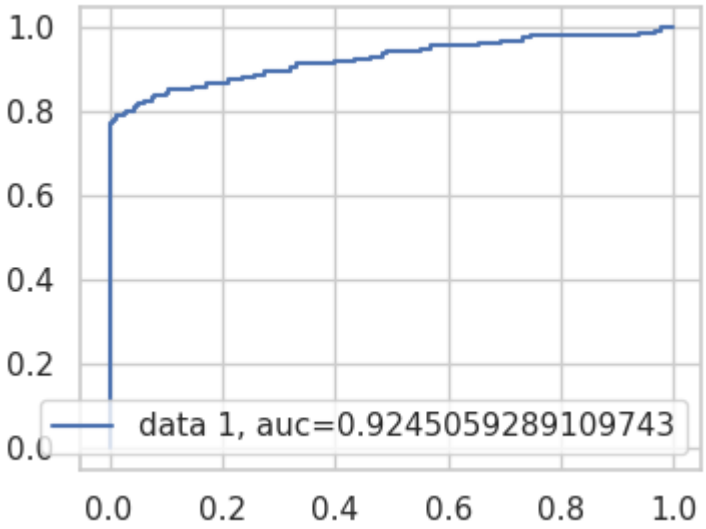
In [ ]:
```
cm = confusion_matrix(y_test,y_predG)
ConfusionMatrixDisplay(cm).plot()
plt.title(f'acuracia = {accuracy_score(y_test,y_predG)}')
```

Out[ ]:
```
Text(0.5, 1.0, 'acuracia = 0.9917605889306321')
```



In [ ]:
```
plt.figure(figsize=(4,3))
y_pred_prob    = logistic.predict_proba(x_test)[::,1]
fpr, tpr, null = roc_curve(y_test,   y_pred_prob)
auc            = roc_auc_score(y_test,y_pred_prob)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f4b8f60f9d0>
```



Rede neural

In [ ]:
```
import tensorflow as ts
```

```
2023-07-04 20:51:03.514741: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, GPU will not b
e used.
2023-07-04 20:51:03.598382: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, GPU will not b
e used.
2023-07-04 20:51:03.600282: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use avai
lable CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-07-04 20:51:05.181352: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
```

In [ ]:
```
ann = ts.keras.models.Sequential()
```

In [ ]:
```
ann.add(ts.keras.layers.Flatten())
ann.add(ts.keras.layers.Dense(units= 32,activation='relu'))
ann.add(ts.keras.layers.Dense(units= 32,activation='relu'))
ann.add(ts.keras.layers.Dense(units= 1,activation='sigmoid'))
```

Imagens do modelo estão ./plots/

- plotar a representacao grafica da rede ela estao na pasta plot no arquivo .pdf

In [ ]:
```
try:
    dot_model = ts.keras.utils.model_to_dot(ann, show_shapes=True, show_layer_names=True, dpi=96)

    graph = graphviz.Source(dot_model.to_string())

    graph.render('./"Plot da rede neural"/questao_1_plot_modeloTENSOR', format='png', cleanup=True)

    graph.view('./plots/questao_1_plot_modeloTENSOR.png')
except Exception:
    print('ERROR')
```

In [ ]:
```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

In [ ]:
```
treino = ann.fit(x_train, y_train, batch_size = 200 ,epochs = 10)
```

```
Epoch 1/10
2023-07-04 20:51:06.835647: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 23923680 exceeds 10% of free syst
em memory.
997/997 [==============================] - 4s 3ms/step - loss: 7.7466 - accuracy: 0.9952
Epoch 2/10
997/997 [==============================] - 2s 2ms/step - loss: 10.3358 - accuracy: 0.9972
Epoch 3/10
997/997 [==============================] - 3s 3ms/step - loss: 8.0735 - accuracy: 0.9963
Epoch 4/10
997/997 [==============================] - 2s 2ms/step - loss: 7.0567 - accuracy: 0.9963
Epoch 5/10
997/997 [==============================] - 3s 3ms/step - loss: 7.8792 - accuracy: 0.9962
Epoch 6/10
997/997 [==============================] - 2s 2ms/step - loss: 4.1591 - accuracy: 0.9965
Epoch 7/10
997/997 [==============================] - 3s 3ms/step - loss: 5.4611 - accuracy: 0.9961
Epoch 8/10
997/997 [==============================] - 2s 2ms/step - loss: 5.3604 - accuracy: 0.9970
Epoch 9/10
997/997 [==============================] - 3s 3ms/step - loss: 3.8222 - accuracy: 0.9966
Epoch 10/10
997/997 [==============================] - 3s 3ms/step - loss: 5.0676 - accuracy: 0.9957
```

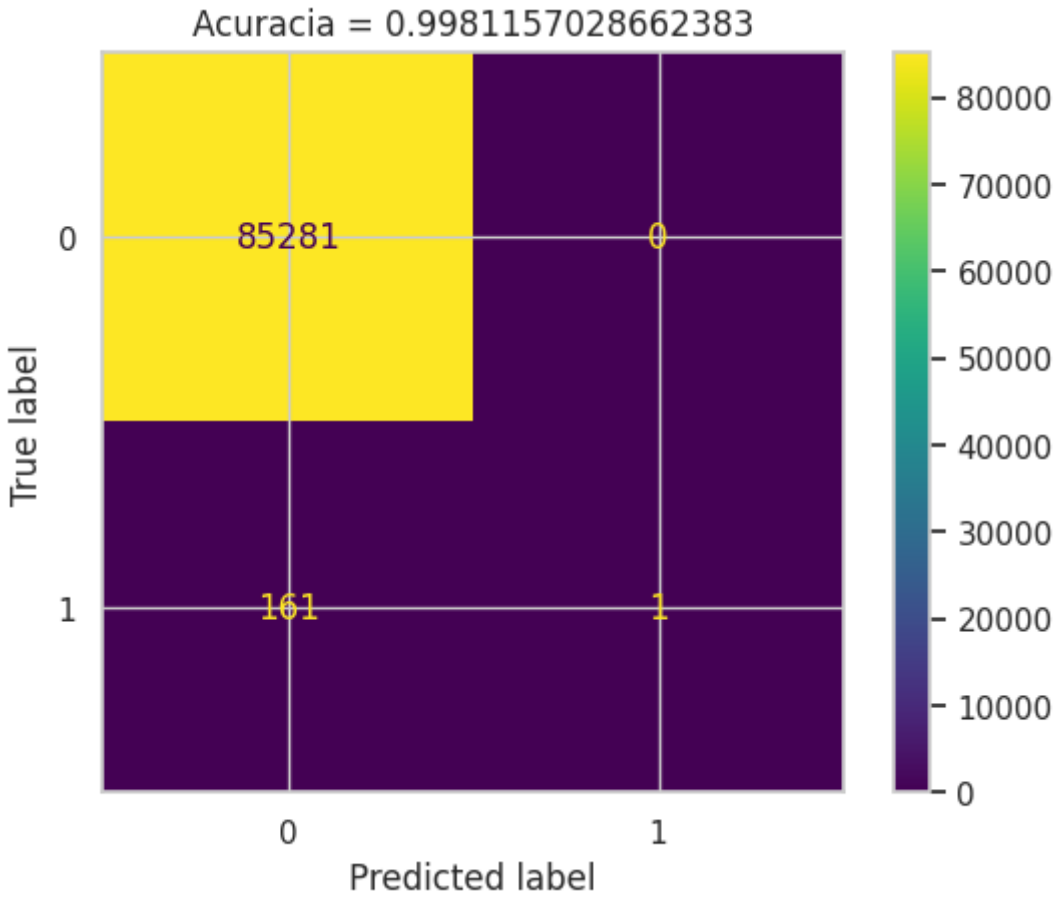
In [ ]:
```
y_pred_ANN = ((ann.predict(x_test)>0.5)*1).T[0]
```

```
2671/2671 [==============================] - 5s 2ms/step
```

In [ ]:
```
cm = confusion_matrix(y_test,y_pred_ANN)

ConfusionMatrixDisplay(cm).plot()
plt.title(f"Acuracia = {accuracy_score(y_test,y_pred_ANN)}")
```

Out[ ]:
```
Text(0.5, 1.0, 'Acuracia = 0.9981157028662383')
```



Teste | Ideia

- equalizando
  - possuem mais dados de não fraude. A ideia é pegar a mesma quantidade valores de fraude aleatoriamente

> Ocorre diminuição quantidade de dados

In [ ]:
```
data['Class'].value_counts()
```

Out[ ]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

Pegar mesma quantidade de valores da Class == 1 mas pegando aleatoriamente 492 valores de 284215 por meio da função sample do python

- com o radom state 42

> dependendo do valores de random state isso ira me da uma acuracia muito alta ou não

In [ ]:
```
data_nao_fraude = data.loc[data['Class']==0]
data_nao_fraude = data_nao_fraude.sample(492,random_state=42)
data_nao_fraude.shape
```

Out[ ]:
```
(492, 31)
```

In [ ]:
```
data_fraude = data.loc[data['Class']==1]
data_fraude.shape
```

Out[ ]:
```
(492, 31)
```

In [ ]:
```
new_data = pd.concat([data_fraude,data_nao_fraude],axis=0)
new_data.shape
```

Out[ ]:
```
(984, 31)
```

In [ ]:
```
X = (new_data.drop(columns='Class')).values
Y = new_data['Class'].values
```

In [ ]:
```
x_train , x_test , y_train , y_test = train_test_split(X,Y,test_size=0.30)
logistic = LogisticRegression()
```

In [ ]:
```
y_pred = logistic.fit(x_train,y_train).predict(x_test)
```

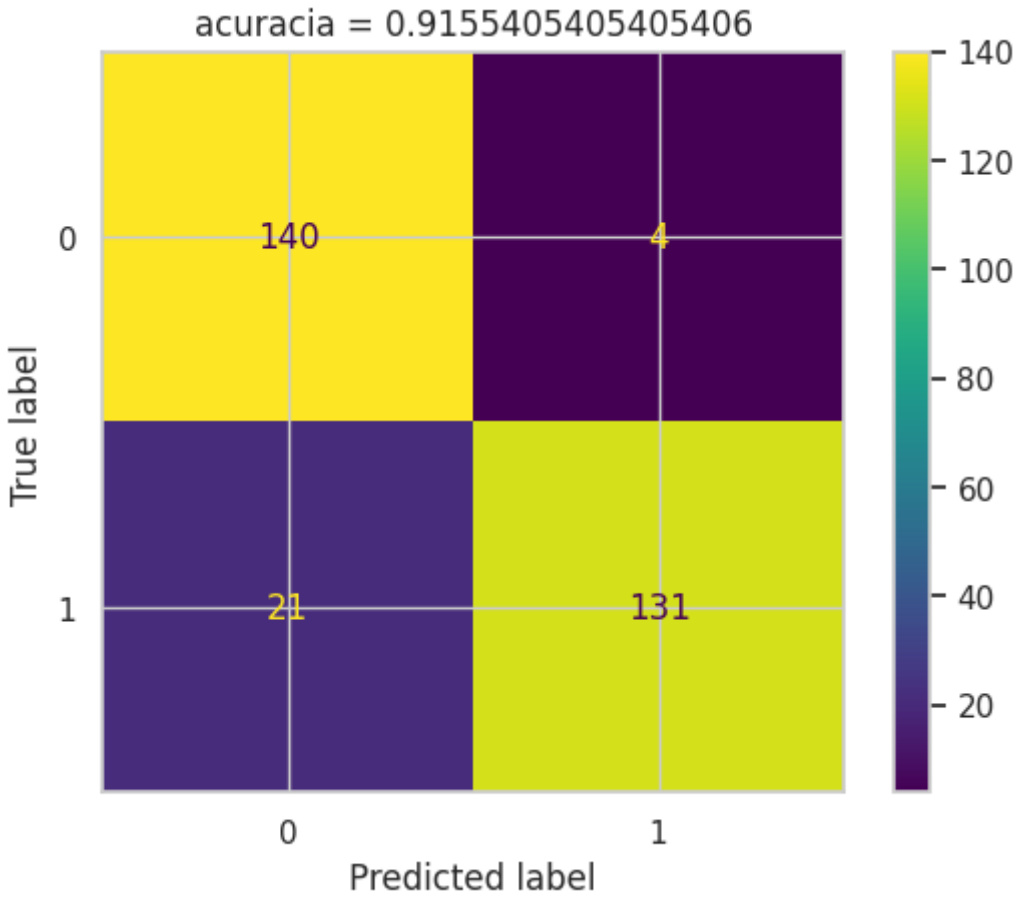
```
/home/mateus/MEGA/Matérias UFC CD/Introdução a Mineração de Dados/venv/lib/python3.11/site-packages/sklearn/linear_model/_logist
ic.py:458: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

In [ ]:
```
cm = confusion_matrix(y_test,y_pred)
ConfusionMatrixDisplay(cm).plot()
plt.title(f'acuracia = {accuracy_score(y_test,y_pred)}')
```

Out[ ]:
```
Text(0.5, 1.0, 'acuracia = 0.9155405405405406')
```


acuracia = 0.9155405405405406
140
4
21
131
0
1
True label
0
1
Predicted label
140
120
100
80
60
40
20