In [ ]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.feature_selection import mutual_info_classif, SelectKBest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_curve, roc_auc_score, mean_absolut
e_error, accuracy_score, plot_roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from math import sqrt
```

O conjunto de dados apresenta em suma dados numéricos obtidos após transformação PCA (Principal Component Analysis). Não foi possível obter os dados previamente a esta transformação.

Leitura do dataset de entrada com informações referentes a transações de cartão de crédito.

In [ ]:

```
data = pd.read_csv('creditcard.csv')
print('Quantidade de linhas do dataset {}'.format(data.shape[0]))
data.head()
```

```
Quantidade de linhas do dataset 284807
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	0.6178
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.0652
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.0660
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.1782
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.5381

- **Removemos os registros que apresentam features com valores faltantes**
- **Removemos a feature "time" por achar que a mesma não é relevante para predizer se uma transação é ou não fraudulenta**

In [ ]:

```
df = data.dropna()
df = df.drop(columns="Time")
df['ID']= np.arange(1,len(df.Class)+1)
print('Quantidade de linhas do dataset sem valor Null/NaN/NaT {}'.format(df.shape[0]))
df.head()
```

```
Quantidade de linhas do dataset sem valor Null/NaN/NaT 284807
```

Out[ ]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	0.
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.

Particionamos o dataset de entrada em 80% para o conjunto de treino e 20% para o conjunto de teste.

In [ ]:

```
x_train,  x_test, y_train, y_test = train_test_split(df.drop(['ID', 'Class'], axis=1), d
f['Class'], test_size=0.20, random_state = 0)
print('Dados de treino {}\n'.format(x_train.shape))
print('Dados de teste  {}\n'.format(x_test.shape))
```

```
Dados de treino (227845, 29)

Dados de teste  (56962, 29)
```

In [ ]:

```
df_train = x_train.copy()
df_train['Class'] = y_train
df_test = x_test.copy()
df_test['Class'] = y_test
```

Descrição estatística do conjunto de treino

In [ ]:

```
df_train.describe()
```

Out[ ]:

	V1	V2	V3	V4	V5	V6	V7	
count	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.000
mean	0.002930	-0.000877	-0.001470	0.001131	-0.001714	-0.001035	-0.000411	-0.001
std	1.955265	1.649672	1.515055	1.416360	1.365962	1.326404	1.225317	1.205
min	-46.855047	-63.344698	-33.680984	-5.683171	-42.147898	-23.496714	-43.557242	-73.216
25%	-0.919898	-0.599013	-0.894424	-0.847412	-0.693585	-0.769201	-0.553573	-0.209
50%	0.021886	0.063972	0.177138	-0.017538	-0.055515	-0.274916	0.039988	0.021
75%	1.316871	0.802516	1.026049	0.744471	0.610153	0.397215	0.569938	0.325
max	2.451888	22.057729	9.382558	16.875344	34.099309	23.917837	44.054461	20.007

Descrição estatística do conjunto de teste

In [ ]:

```
df_test.describe()
```

Out[ ]:

	V1	V2	V3	V4	V5	V6	V7	V8

	V1	V2	V3	V4	V5	V6	V7	V8	
count	56962.000000	56962.000000	56962.000000	56962.000000	56962.000000	56962.000000	56962.000000	56962.000000	569
mean	-0.011720	0.003508	0.005881	-0.004524	0.006858	0.004139	0.001644	0.005440	
std	1.972334	1.657848	1.521044	1.413903	1.435957	1.355490	1.283130	1.148643	
min	-56.407510	-72.715728	-48.325589	-5.600607	-113.743307	-26.160506	-23.189397	-50.943369	
25%	-0.921972	-0.595792	-0.874649	-0.853267	-0.683487	-0.765653	-0.555542	-0.206208	
50%	-0.002761	0.072712	0.191364	-0.028170	-0.050472	-0.271310	0.040576	0.025516	
75%	1.309289	0.809015	1.031690	0.739049	0.619408	0.403661	0.572788	0.332808	
max	2.454930	14.845545	4.079168	16.491217	34.801666	73.301626	120.589494	17.573712	

Contagem dos valores de cada classe. 0 indicando uma transação onde não há fraude e 1 indicando uma fraude.

In [ ]:

```
df_train['Class'].value_counts()
```

Out[ ]:

```
0    227454
1       391
Name: Class, dtype: int64
```

In [ ]:

```
df_test['Class'].value_counts()
```

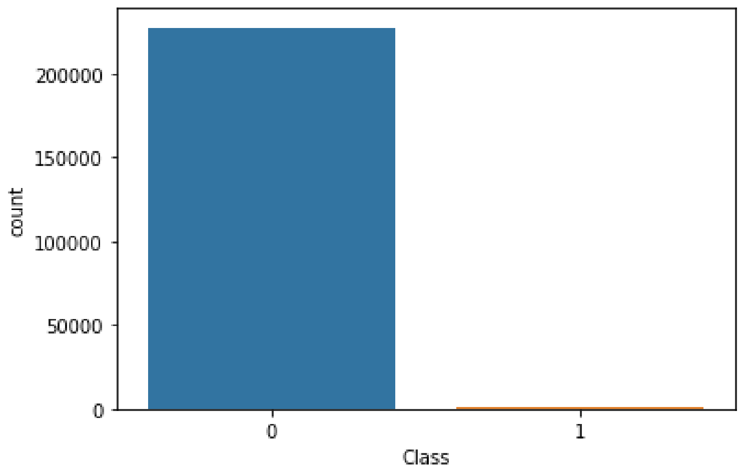
Out[ ]:

```
0    56861
1      101
Name: Class, dtype: int64
```

Gráficos com a quantidade de cada classe nos conjuntos de dados

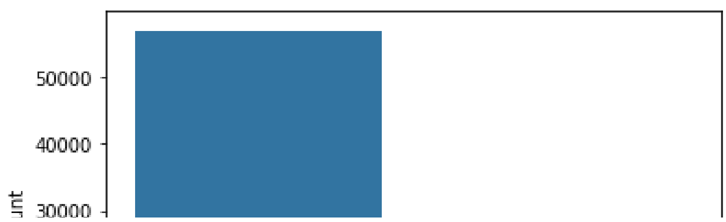
In [ ]:

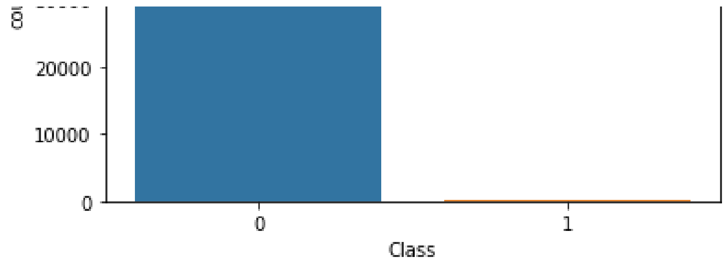
```
ax = sns.countplot(x="Class", data=df_train)
```



In [ ]:

```
bx = sns.countplot(x="Class", data=df_test)
```





In [ ]:

```
n_fraudulent_transactions = df_train['Class'].value_counts()[1]
print('Quantidade de transações fraudulentas no dataset de treino ({}) representando um total de ({})% do dataset'.format(n_fraudulent_transactions, (n_fraudulent_transactions/df_train.shape[0])*100))
n_fraudulent_transactions = df_test['Class'].value_counts()[1]
print('Quantidade de transações fraudulentas no dataset de teste ({}) representando um total de ({})% do dataset'.format(n_fraudulent_transactions, (n_fraudulent_transactions/df_test.shape[0])*100))
```

```
Quantidade de transações fraudulentas no dataset de treino (391) representando um total de (0.171607891329632)% do dataset
Quantidade de transações fraudulentas no dataset de teste (101) representando um total de (0.1773111899160844)% do dataset
```

Utilizamos o `mutual_info_classif` para estimar informações através de testes estatísticos, auxiliando na seleção de atributos que possuem forte relacionamento com a variável que estamos tentando prever.

In [ ]:

```
mic = mutual_info_classif(x_train, y_train)
mic
```

Out[ ]:

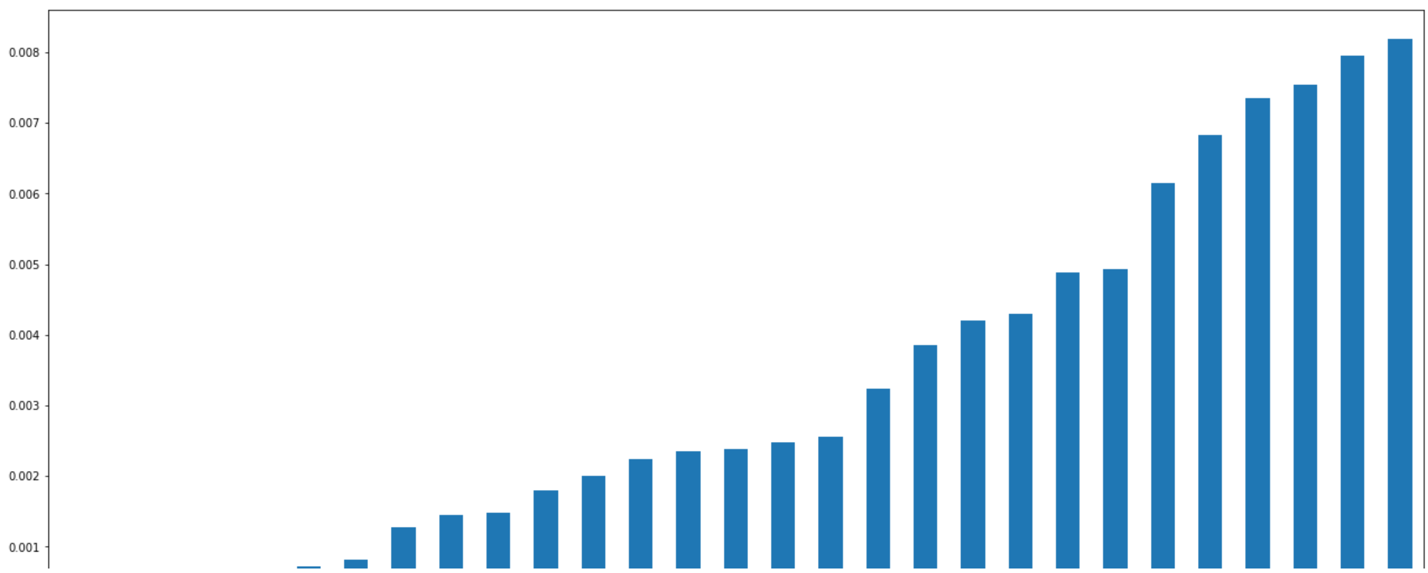
```
array([0.00223417, 0.00324333, 0.0048859 , 0.00493872, 0.0024805 ,
       0.00235812, 0.00385332, 0.00200889, 0.00420066, 0.00735391,
       0.00683457, 0.00754913, 0.00048603, 0.00796175, 0.00033733,
       0.00614541, 0.00819216, 0.00429184, 0.00147992, 0.00127442,
       0.00256113, 0.0004726 , 0.00081558, 0.00072547, 0.0006311 ,
       0.00054575, 0.00238648, 0.00180494, 0.00145779])
```

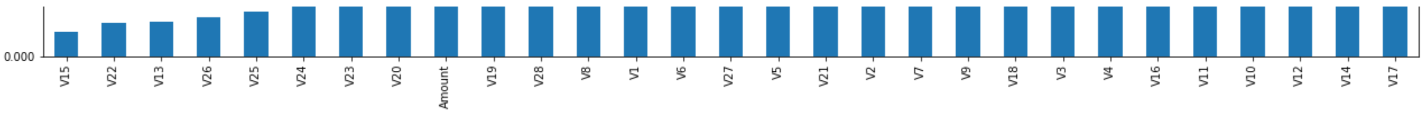
In [ ]:

```
mic = pd.Series(mic)
mic.index = x_train.columns
mic = mic.sort_values(ascending = True)
mic.plot.bar(figsize=(22,10))
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f63922f6fd0>
```





Selecionamos as K variáveis que mais se relacionam com a coluna que indica a classificação da transação. `k=22`

In [ ]:

```
selection = SelectKBest(mutual_info_classif, k= 22).fit(x_train, y_train)
X_train = x_train[x_train.columns[selection.get_support()]]
X_test = x_test[x_test.columns[selection.get_support()]]
```

Função utilizada para gerar as curvas do K fold cross validation

In [ ]:

```
def plot_Kfold_cross_validation_curves(md, x_data, y_data):
  cv = StratifiedKFold(n_splits=5)

  tprs = []
  aucs = []
  mean_fpr = np.linspace(0, 1, 100)

  fig, ax = plt.subplots()
  for i, (train, test) in enumerate(cv.split(x_data, y_data)):
      md.fit(x_data.iloc[train], y_data.iloc[train])
      viz = plot_roc_curve(md, x_data.iloc[test], y_data.iloc[test],
                          name='ROC fold {}'.format(i),
                          alpha=0.3, lw=1, ax=ax)
      interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
      interp_tpr[0] = 0.0
      tprs.append(interp_tpr)
      aucs.append(viz.roc_auc)

  ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
          label='Chance', alpha=.8)

  mean_tpr = np.mean(tprs, axis=0)
  mean_tpr[-1] = 1.0
  mean_auc = auc(mean_fpr, mean_tpr)
  std_auc = np.std(aucs)
  ax.plot(mean_fpr, mean_tpr, color='b',
          label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
          lw=2, alpha=.8)

  std_tpr = np.std(tprs, axis=0)
  tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
  tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
  ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                  label=r'$\pm$ 1 std. dev.')

  ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
        title="ROC for K fold cross-validation curves")
  ax.legend(loc="lower right")
  plt.show()
```

Random Forest

Utilizaremos a classe padrão do classificador Random Forest, não utilizamos variações na parametrização da classe devido a obtenção de um resultado satisfatório com os parâmetros padrões.

In [ ]:

```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Treino

Relatório de classificação da predição com o modelo Random forest com o sample de treino

In [ ]:

```
predictions = rf.predict(X_train)
print(classification_report(y_train, predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    227454
           1       1.00      1.00      1.00       391

    accuracy                           1.00    227845
   macro avg       1.00      1.00      1.00    227845
weighted avg       1.00      1.00      1.00    227845
```

Matriz de confusão dos valores preditos com o conjunto de treino

In [ ]:

```
pd.crosstab(y_train, predictions, rownames=['Real'],colnames=['Predito'],margins=True)
```

Out[ ]:

Predito	0	1	All
Real			
0	227454	0	227454
1	1	390	391
All	227455	390	227845

Scores das validações cruzadas

In [ ]:

```
scores = cross_val_score(rf, X_train, y_train, cv=5, scoring='accuracy')
scores
```

Out[ ]:

```
array([0.99958305, 0.99940749, 0.99962694, 0.99956111, 0.99949527])
```

Media dos scores obtidos das validações cruzadas

In [ ]:

```
scores.mean()
```

Out[ ]:

```
0.9995347714455003
```

Acurácia das predições com base no conjunto de treino

In [ ]:

```
accuracy_score(y_train, predictions)
```

Out[ ]:

```
0.9999956110513727
```

Erro absoluto com base no conjunto de treino

In [ ]:

```
e = mean_absolute_error(y_train, predictions)
e
```

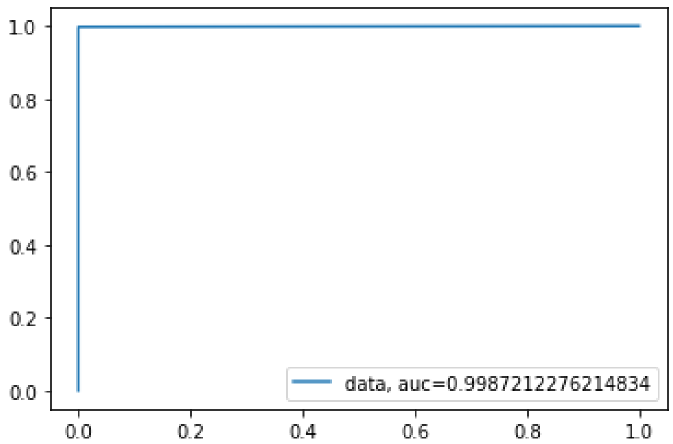
Out[ ]:

```
4.388948627356317e-06
```

Curva ROC

In [ ]:

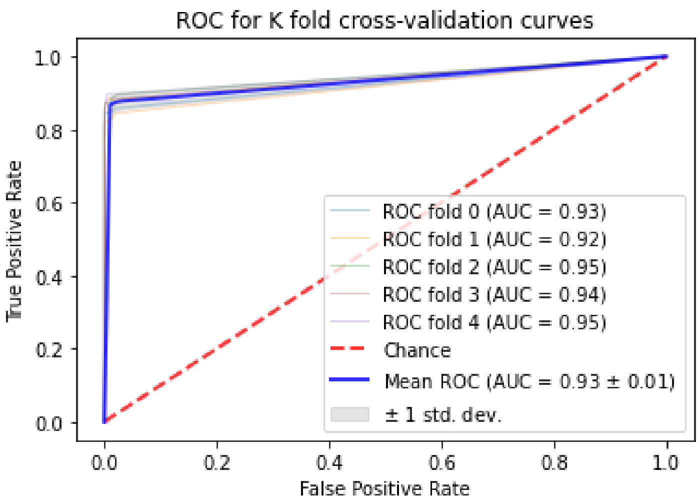
```
fpr, tpr, _ = roc_curve(y_train, predictions)
roc_auc_scr =  roc_auc_score(y_train, predictions)
plt.plot(fpr,tpr,label="data, auc="+str(roc_auc_scr))
plt.legend(loc=4)
plt.show()
```



Curvas da K fold cross-validation

In [ ]:

```
plot_Kfold_cross_validation_curves(rf, X_train, y_train)
```



Teste

Predição com o sample de teste

In [ ]:

```
predictions_test = rf.predict(X_test)
```

Relatório de classificação da predição com o modelo Random forest com o sample de teste

In [ ]:

```
print(classification_report(y_test, predictions_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.93      0.77      0.84       101

    accuracy                           1.00     56962
   macro avg       0.96      0.89      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

Matriz de confusão dos valores preditos com o conjunto de teste

In [ ]:

```
pd.crosstab(y_test, predictions_test, rownames=['Real'],colnames=['Predito'],margins=True)
```

Out[ ]:

Predito	0	1	All
Real			
0	56855	6	56861
1	23	78	101
All	56878	84	56962

Validação cruzada utilizando 5 pastas com conjunto de teste

In [ ]:

```
scores = cross_val_score(rf, X_test, predictions_test, cv=5, scoring='accuracy')
scores
```

Out[ ]:

```
array([0.99973668, 0.99964891, 0.99982444, 0.99947331, 0.99982444])
```

Media dos scores obtidos com o conjunto de teste

In [ ]:

```
scores.mean()
```

Out[ ]:

```
0.9997015557305542
```

Acurácia do modelo

In [ ]:

```
accuracy_score(y_test, predictions_test)
```

Out[ ]:

```
0.9994908886626171
```

Mean Absolute Error

In [ ]:

```
e = mean_absolute_error(y_test, predictions_test)
e
```

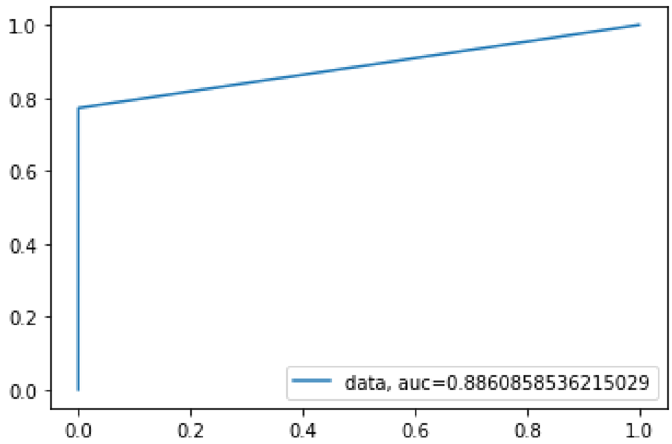
Out[ ]:

```
0.0005091113373828166
```

Curva ROC com o conjunto de teste

In [ ]:

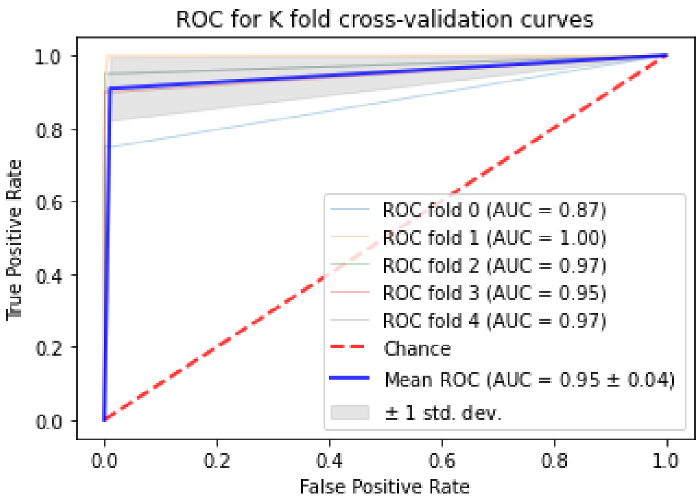
```
fpr, tpr, _ = roc_curve(y_test, predictions_test)
roc_auc_scr =  roc_auc_score(y_test, predictions_test)
plt.plot(fpr,tpr,label="data, auc="+str(roc_auc_scr))
plt.legend(loc=4)
plt.show()
```



Curvas da K fold cross-validation com os dados de teste

In [ ]:

```
plot_Kfold_cross_validation_curves(rf, X_test, y_test)
```



KNN

Utilizaremos a classe padrão do K Neighbors Classifier, utilizaremos apenas o parâmetro `n_neighbors=3` pois o mesmo demonstrou um aumento na acurácia do modelo. Para descobrir isso executamos `i` execuções com `i` variando de 1 até 25 e a execução com 3 vizinhos mostrou a melhor acurácia.

In [ ]:

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

Out[ ]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

Treino

Predição da classificação dos dados de treino com base no modelo treinado

In [ ]:

```
y_pred = knn.predict(X_train)
```

Relatório de classificação da predição com o modelo K Neighbors classifier com o sample de treino

In [ ]:

```
print(classification_report(y_train, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    227454
           1       0.96      0.76      0.85       391

    accuracy                           1.00    227845
   macro avg       0.98      0.88      0.92    227845
weighted avg       1.00      1.00      1.00    227845
```

Matriz de confusão dos valores preditos com o conjunto de treino

In [ ]:

```
pd.crosstab(y_train, y_pred, rownames=['Real'],colnames=['Predito'],margins=True)
```

Out[ ]:

Predito	0	1	All
Real			
0	227442	12	227454
1	94	297	391
All	227536	309	227845

Scores das validações cruzadas

In [ ]:

```
scores = cross_val_score(knn, X_train, y_train, cv=5, scoring='accuracy')
scores
```

Out[ ]:

```
array([0.9993636 , 0.99927582, 0.99945138, 0.99934166, 0.99912221])
```

Media dos scores obtidos das validações cruzadas

In [ ]:

```
scores.mean()
```

Out[ ]:

```
0.9993109350655051
```

Acurácia das predições com base no conjunto de treino

In [ ]:

```
accuracy_score(y_train, y_pred)
```

Out[ ]:

```
0.9995347714455002
```

Erro absoluto médio

In [ ]:

```
e = mean_absolute_error(y_train, y_pred)
e
```

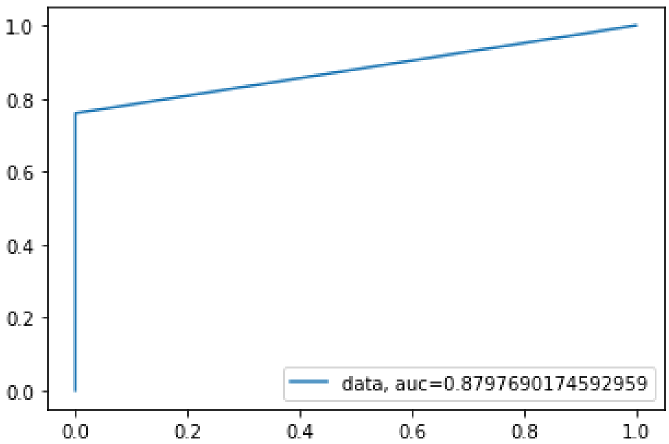
Out[ ]:

```
0.0004652285544997696
```

Curva ROC dos dados de treino

In [ ]:

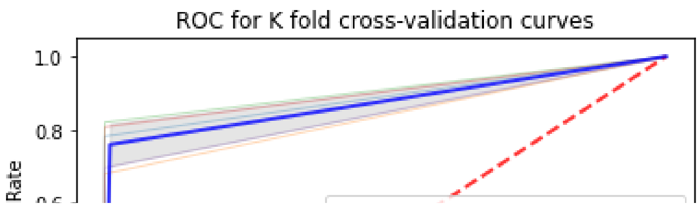
```
fpr, tpr, _ = roc_curve(y_train, y_pred)
roc_auc_scr =  roc_auc_score(y_train, y_pred)
plt.plot(fpr,tpr,label="data, auc="+str(roc_auc_scr))
plt.legend(loc=4)
plt.show()
```



Curvas da K fold cross-validation

In [ ]:

```
plot_Kfold_cross_validation_curves(knn, X_train, y_train)
```



Teste

Predição com base no modelo treinado utilizando o sample de teste

In [ ]:

```
y_pred = knn.predict(X_test)
```

Relatório de classificação da predição com o modelo KNN com o sample de teste

In [ ]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.92      0.72      0.81       101

    accuracy                           1.00     56962
   macro avg       0.96      0.86      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```

Matriz de confusão dos valores preditos com o conjunto de teste

In [ ]:

```
pd.crosstab(y_test, y_pred, rownames=['Real'],colnames=['Predito'],margins=True)
```

Out[ ]:

Predito	0	1	All
Real			
0	56855	6	56861
1	28	73	101
All	56883	79	56962

Scores das validações cruzadas

In [ ]:

```
scores = cross_val_score(knn, X_test, y_test, cv=5, scoring='accuracy')
scores
```

Out[ ]:

```
array([0.99938559, 0.99912227, 0.99894663, 0.99894663, 0.99920997])
```

Media dos scores obtidos com o conjunto de teste

In [ ]:

```
scores.mean()
```

Out[ ]:

```
0.9991222172075895
```

Acurácia do modelo com base nos dados preditos do conjunto de teste

In [ ]:

```
accuracy_score(y_test, y_pred)
```

Out[ ]:

```
0.999403110845827
```

Erro absoluto do modelo com base no conjunto de teste

In [ ]:

```
e = mean_absolute_error(y_test, y_pred)
e
```

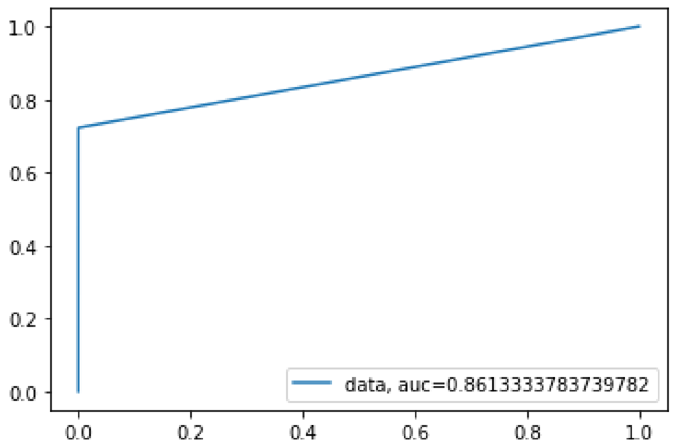
Out[ ]:

```
0.0005968891541729574
```

Curva ROC com o conjunto de teste

In [ ]:

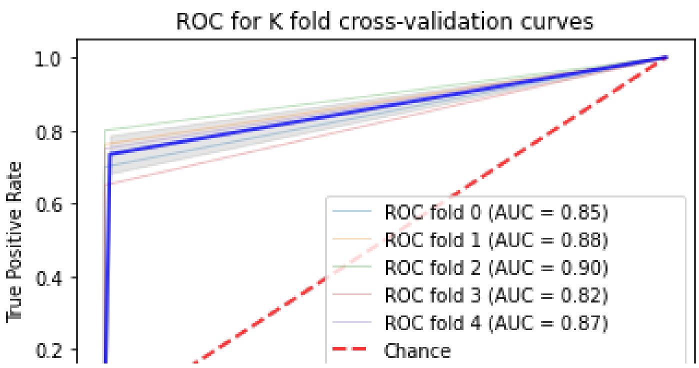
```
fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc_scr =  roc_auc_score(y_test, y_pred)
plt.plot(fpr,tpr,label="data, auc="+str(roc_auc_scr))
plt.legend(loc=4)
plt.show()
```

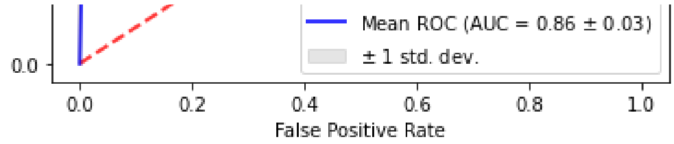


Curvas da K fold cross-validation com os dados de teste

In [ ]:

```
plot_Kfold_cross_validation_curves(knn, X_test, y_test)
```





MLPClassifier

Utilizaremos a classe do MLP Classifier com algumas alterações dos parâmetros default, pois principalmente relacionado ao número de iterações acaba fazendo com que o tempo de execução se torne algo muito custoso principalmente para executar as k validações cruzadas. Optamos por diminuir o número de layers como o número de neurônios da rede neural para 2 camadas com 50 neurônios cada e um número máximo de iterações igual a 5. Importante deixar claro que o modelo apresenta uma acurácia superor utizando a parametrização padrão da classe.

In [ ]:

```
clf = MLPClassifier(hidden_layer_sizes=(50,50), max_iter=5, alpha=0.0001,
                     solver='sgd', verbose=10,  random_state=21,tol=0.000000001)
```

Treino

Treino e predição com o modelo treinado utilizando o MLP classifier

In [ ]:

```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_train)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.02712378
Iteration 3, loss = inf
Iteration 4, loss = 0.01661020
Iteration 5, loss = 0.01060315
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Relatório de classificação da predição com o modelo MLPClassifier com o sample de treino

In [ ]:

```
print(classification_report(y_train, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    227454
           1       0.87      0.64      0.74       391

    accuracy                           1.00    227845
   macro avg       0.93      0.82      0.87    227845
weighted avg       1.00      1.00      1.00    227845
```

Matriz de confusão dos valores preditos com o conjunto de treino

In [ ]:

```
pd.crosstab(y_train, y_pred, rownames=['Real'],colnames=['Predito'],margins=True)
```

Out[ ]:

Predito	0	1	All

Predito Real	0	1	All
0	227416	38	227454
1	141	250	391
All	227557	288	227845

Scores das validações cruzadas com conjunto de treino

In [ ]:

```
scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accuracy')
scores
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.02722532
Iteration 3, loss = 0.02695025
Iteration 4, loss = 0.02307299
Iteration 5, loss = 0.01372564
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.02913047
Iteration 3, loss = 0.03468142
Iteration 4, loss = 0.01834889
Iteration 5, loss = 0.01344923
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.02834564
Iteration 3, loss = inf
Iteration 4, loss = 0.02769228
Iteration 5, loss = 0.01786017
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.02960014
Iteration 3, loss = inf
Iteration 4, loss = 0.01987128
Iteration 5, loss = 0.01891146
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.03324739
Iteration 3, loss = inf
Iteration 4, loss = 0.01918058
Iteration 5, loss = 0.01250826
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Out[ ]:

```
array([0.99918804, 0.99929777, 0.99881498, 0.99899054, 0.98870526])
```

Media dos scores obtidos das validações cruzadas

In [ ]:

```
scores.mean()
```

Out[ ]:

```
0.9989993197129629
```

Acurácia das predições com base no conjunto de treino

In [ ]:

```
accuracy_score(y_train, y_pred)
```

Out[ ]:

```
0.9992143781957032
```

Erro absoluto com base no conjunto de treino

In [ ]:

```
e = mean_absolute_error(y_train, y_pred)
e
```

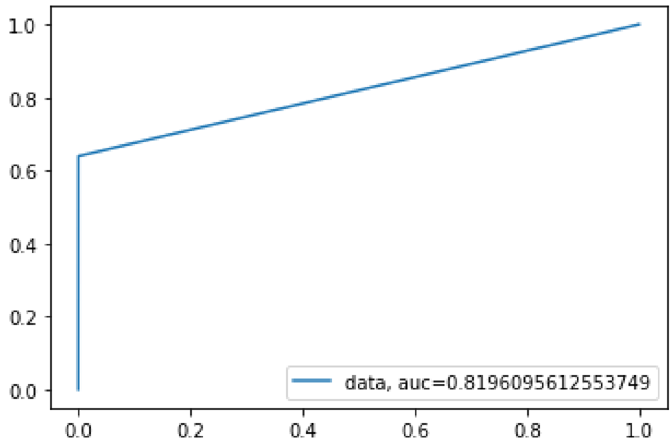
Out[ ]:

```
0.0007856218042967807
```

Curva ROC com os dados de treino

In [ ]:

```
fpr, tpr, _ = roc_curve(y_train, y_pred)
roc_auc_scr =  roc_auc_score(y_train, y_pred)
plt.plot(fpr,tpr,label="data, auc="+str(roc_auc_scr))
plt.legend(loc=4)
plt.show()
```



Curvas da K fold cross-validation com os dados de treino

In [ ]:

```
plot_Kfold_cross_validation_curves(clf, X_train, y_train)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.02722532
Iteration 3, loss = 0.02695025
Iteration 4, loss = 0.02307299
Iteration 5, loss = 0.01372564
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
```

```
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.02913047
Iteration 3, loss = 0.03468142
Iteration 4, loss = 0.01834889
Iteration 5, loss = 0.01344923
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.02834564
Iteration 3, loss = inf
Iteration 4, loss = 0.02769228
Iteration 5, loss = 0.01786017
```

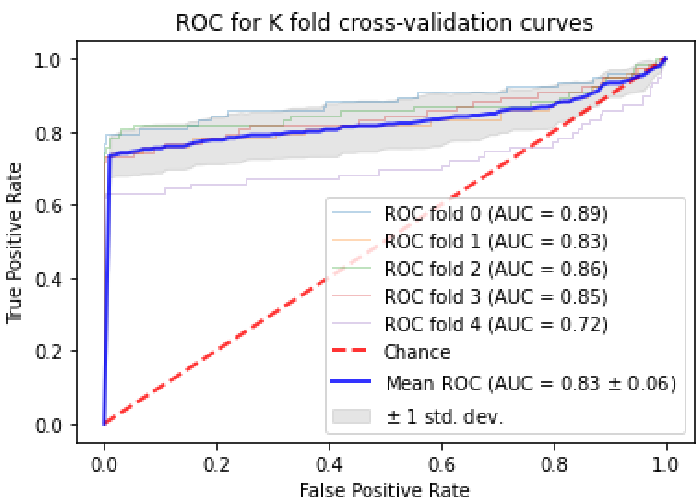
```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.02960014
Iteration 3, loss = inf
Iteration 4, loss = 0.01987128
Iteration 5, loss = 0.01891146
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.03324739
Iteration 3, loss = inf
Iteration 4, loss = 0.01918058
Iteration 5, loss = 0.01250826
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```



Teste

Predição dos dados de teste com o modelo treinado utilizando o MLP classifier

In [ ]:

```
y_pred = clf.predict(X_test)
```

Relatório de classificação da predição com o modelo MLPClassifier com o sample de teste

In [ ]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.88      0.56      0.69       101

    accuracy                           1.00     56962
   macro avg       0.94      0.78      0.84     56962
weighted avg       1.00      1.00      1.00     56962
```

Matriz de confusão dos valores preditos com o conjunto de teste

In [ ]:

```
pd.crosstab(y_test, y_pred, rownames=['Real'],colnames=['Predito'],margins=True)
```

Out[ ]:

Predito	0	1	All
Real			
0	56853	8	56861
1	44	57	101
All	56897	65	56962

Scores das validações cruzadas com conjunto de treino

In [ ]:

```
scores = cross_val_score(clf, X_test, y_test, cv=5, scoring='accuracy')
scores
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.10979617
Iteration 3, loss = 0.07809046
Iteration 4, loss = 0.05709109
Iteration 5, loss = 0.04128277
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.10426688
Iteration 3, loss = 0.07710955
Iteration 4, loss = 0.05767491
Iteration 5, loss = 0.04334860
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.10734112
Iteration 3, loss = 0.07366163
Iteration 4, loss = 0.05507909
Iteration 5, loss = 0.03982645
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.11081528
Iteration 3, loss = 0.07684522
Iteration 4, loss = 0.05440205
Iteration 5, loss = 0.03938074
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.10059844
Iteration 3, loss = 0.07369080
Iteration 4, loss = 0.05428945
Iteration 5, loss = 0.03891028
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Out[ ]:

```
array([0.99877118, 0.99850786, 0.99868329, 0.99885885, 0.99868329])
```

Média dos scores obtidos das k validações cruzadas

In [ ]:

```
scores.mean()
```

Out[ ]:

```
0.9987008904664505
```

Acurácia das predições com base no conjunto de teste

In [ ]:

```
accuracy_score(y_test, y_pred)
```

Out[ ]:

```
0.9990871107053826
```

Erro absoluto com base no conjunto de teste

In [ ]:

```
e = mean_absolute_error(y_test, y_pred)
e
```

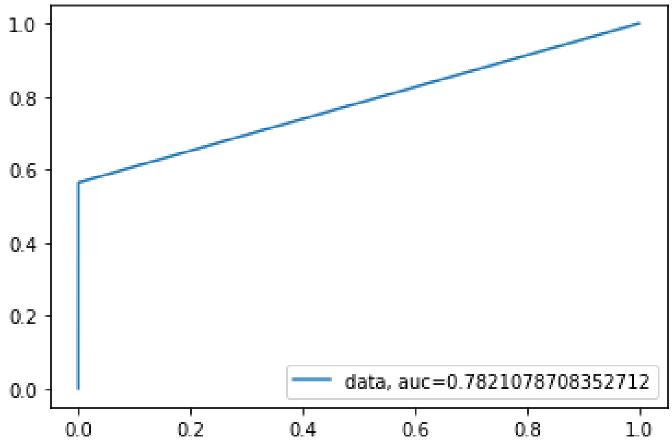
Out[ ]:

```
0.0009128892946174643
```

Curva ROC utilizando os dados de teste

In [ ]:

```
fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc_scr =  roc_auc_score(y_test, y_pred)
plt.plot(fpr,tpr,label="data, auc="+str(roc_auc_scr))
plt.legend(loc=4)
plt.show()
```



Curvas da K fold cross-validation com os dados de teste

In [ ]:

```
plot_Kfold_cross_validation_curves(clf, X_test, y_test)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.10979617
Iteration 3, loss = 0.07809046
Iteration 4, loss = 0.05709109
Iteration 5, loss = 0.04128277
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.10426688
Iteration 3, loss = 0.07710955
Iteration 4, loss = 0.05767491
Iteration 5, loss = 0.04334860
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.10734112
Iteration 3, loss = 0.07366163
Iteration 4, loss = 0.05507909
Iteration 5, loss = 0.03982645
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

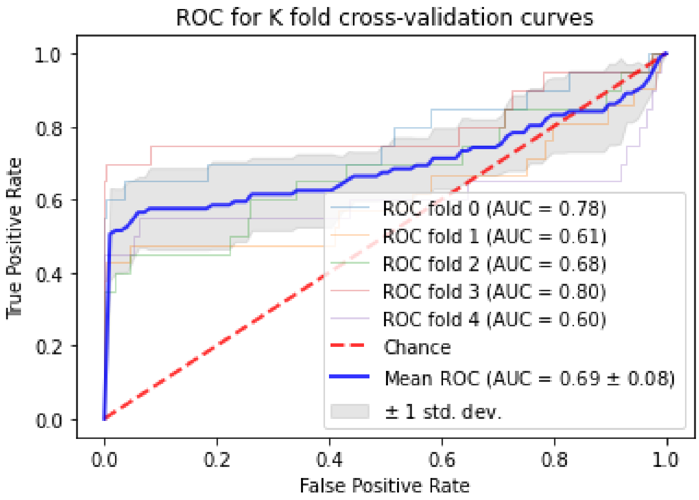
```
Iteration 1, loss = inf
Iteration 2, loss = 0.11081528
Iteration 3, loss = 0.07684522
Iteration 4, loss = 0.05440205
Iteration 5, loss = 0.03938074
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.10059844
Iteration 3, loss = 0.07369080
Iteration 4, loss = 0.05428945
Iteration 5, loss = 0.03891028
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
```

```
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the opti
mization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```



É uma pena para a execução do trabalho com base nesse tema não ter o dataset pré processamento para identificar um possível overfit nos modelos criados, haja vista a grande acurácia apresentada. Apesar da incógnita perando as features dos dados pré processamento podemos concluir que o objetivo foi alcançado com sucesso. Podemos fazer esta afirmação olhando para as taxas de falso positivo e falso negativo já que os dados em si apresentam em ampla maioria registros de transações não fraudulentas, logo o peso de marcar uma transação como fraudulenda ou não sem que a mesma tenha realmente esta classificação adquire um peso maior.