```
# gerardo Herrera... random forest (100 arboles) con 28k instacias de normal y recovering y 2


from google.colab import drive
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
import numpy as np
import pandas as pd
import os

import matplotlib.pyplot as plt
%matplotlib inline
from tqdm import tqdm_notebook
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
pd.options.display.precision = 15

import time
# Libraries
import numpy as np
import pandas as pd
pd.set_option('max_columns', None)
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

!pip install lightgbm
!pip install catboost

import datetime
import lightgbm as lgb
from scipy import stats
from sklearn.model_selection import train_test_split, StratifiedKFold, KFold, cross_val_score
from sklearn.preprocessing import StandardScaler
import os
import lightgbm as lgb
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn import metrics
from sklearn import linear_model
from tqdm import tqdm_notebook
from catboost import CatBoostClassifier
```

```
     Requirement already satisfied: lightgbm in /usr/local/lib/python3.6/dist-packages (2.2.3
     Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-packages (0
     Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from lig
     Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from lig
```

```
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (1
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/52/39/128fff65072c8327371e3c594f3c
     |████████████████████████████████| 66.2MB 66kB/s
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from cat
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from catbo
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from ca
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (1
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (1
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/li
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages
Installing collected packages: catboost
Successfully installed catboost-0.24.2
```

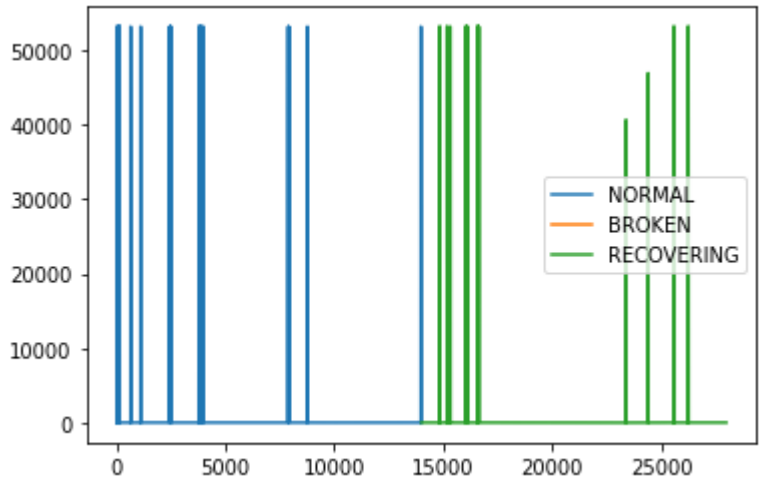
```
# sensor = pd.read_csv('../input/sensor.csv')
# sensor = pd.read_csv('../input/vombas/sensor_procesado.csv')
#sensor = pd.read_csv('dataset_sensor_procesado.csv')
#sensor = pd.read_csv('../input/bombas-sensores-conocidos/sensor2.csv')
#sensor = pd.read_csv('../input/28k-s24-balan-vombas/sensor2-ordenado_status_sin_broken_balan
sensor = pd.read_csv('/content/drive/My Drive/datasets/sensor2-ordenado_status_sin_broken_bal
#sensor.drop(['Unnamed: 0'], axis=1, inplace=True)
```

```
sensor.head()
```

| | Unnamed: 0 | timestamp | sensor_00 | sensor_01 | sensor_02 | sens |
|---|---|---|---|---|---|---|
| **0** | 0 | 2018-04-01 00:00:00 | 2.465394 | 47.092010000000002 | 53.211799999999997 | 46.3107600000 |
| **1** | 1 | 2018-04-01 00:01:00 | 2.465394 | 47.092010000000002 | 53.211799999999997 | 46.3107600000 |
| **2** | 2 | 2018-04-01 00:02:00 | 2.444734 | 47.352429999999998 | 53.211799999999997 | 46.3975700000 |
| **3** | 3 | 2018-04-01 00:03:00 | 2.460474 | 47.092010000000002 | 53.168399999999998 | 46.3975677490 |
| **4** | 4 | 2018-04-01 00:04:00 | 2.445718 | 47.135410000000000 | 53.211799999999997 | 46.3975677490 |

```
#sensor.drop(['sensor_15'], axis=1, inplace=True)
sensor.drop(['timestamp'], axis=1, inplace=True)


# lineA DE LOS 22K INSTANCIAS
plt.plot(sensor.loc[sensor['machine_status'] == 'NORMAL', 'sensor_02'], label='NORMAL')
plt.plot(sensor.loc[sensor['machine_status'] == 'BROKEN', 'sensor_02'], label='BROKEN')
plt.plot(sensor.loc[sensor['machine_status'] == 'RECOVERING', 'sensor_02'], label='RECOVERING
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fc71c12e4e0>
```




```
cleanup_nums = {"machine_status":     {"NORMAL": 0, "RECOVERING": 1,"BROKEN": 2}}


sensor.replace(cleanup_nums, inplace=True)
sensor.head(30)
```

| | Unnamed: 0 | sensor_00 | sensor_01 | sensor_02 | sensor_03 |
|---|---|---|---|---|---|
| **0** | 0 | 2.465394 | 47.092010000000002 | 53.211799999999997 | 46.310760000000002 |
| **1** | 1 | 2.465394 | 47.092010000000002 | 53.211799999999997 | 46.310760000000002 |
| **2** | 2 | 2.444734 | 47.352429999999998 | 53.211799999999997 | 46.397570000000002 |
| **3** | 3 | 2.460474 | 47.092010000000002 | 53.168399999999998 | 46.397567749023402 |
| **4** | 4 | 2.445718 | 47.135410000000000 | 53.211799999999997 | 46.397567749023402 |
| **5** | 5 | 2.453588 | 47.092010000000002 | 53.168399999999998 | 46.397567749023402 |
| **6** | 6 | 2.455556 | 47.048609999999996 | 53.168399810790994 | 46.397567749023402 |
| **7** | 7 | 2.449653 | 47.135410000000000 | 53.168399810790994 | 46.397567749023402 |
| **8** | 8 | 2.463426 | 47.092010000000002 | 53.168399810790994 | 46.397567749023402 |
| **9** | 9 | 2.445718 | 47.178820000000002 | 53.168399999999998 | 46.397567749023402 |
| **10** | 10 | 2.464410 | 47.482640000000004 | 53125.000000000000000 | 46.397567749023402 |
| **11** | 11 | 2.444734 | 47.916660000000000 | 53.168399999999998 | 46.397567749023402 |
| **12** | 12 | 2.460474 | 48.263890000000004 | 53125.000000000000000 | 46.397567749023402 |
| **13** | 13 | 2.448669 | 48.437500000000000 | 53.168399999999998 | 46.397567749023402 |
| **14** | 14 | 2.453588 | 48.567709999999998 | 53.168399999999998 | 46.397567749023402 |
| **15** | 15 | 2.455556 | 48.394100000000002 | 53125.000000000000000 | 46.397570000000002 |
| **16** | 16 | 2.449653 | 48.394100000000002 | 53.168399999999998 | 46.310760000000002 |
| **17** | 17 | 2.463426 | 48.480899999999998 | 53.689240000000012 | 46.310760498046896 |
| **18** | 18 | 2.445718 | 48.611109999999996 | 53125.000000000000000 | 46.310760498046896 |

```
for col in sensor.columns[1:-1]:
    sensor[col] = sensor[col].fillna(sensor[col].mean())
```

```
# bosque aleatorio
```

```
sensor.fillna(sensor.mean(), inplace=True)
```

| | | | | | |
|---|---|---|---|---|---|
| **24** | 24 | 2.453588 | 49.218750000000000 | 53.038190000000000 | 46.267360687255895 |

```
sensor.head()
```

| | Unnamed: 0 | sensor_00 | sensor_01 | sensor_02 | sensor_03 | |
|---|---|---|---|---|---|---|
| **0** | 0 | 2.465394 | 47.092010000000002 | 53.211799999999997 | 46.310760000000002 | 6343 |
| **1** | 1 | 2.465394 | 47.092010000000002 | 53.211799999999997 | 46.310760000000002 | 6343 |
| **2** | 2 | 2.444734 | 47.352429999999998 | 53.211799999999997 | 46.397570000000002 | 6 |

```
print(sensor.shape)
```

```
(28002, 26)
```

```
# Encontrar características importantes en Scikit-learn

# from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
#clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
#clf.fit(X_train,y_train)
```

```
sensor.describe()
```

| | Unnamed: 0 | sensor_00 | sensor_01 | |
|---|---|---|---|---|
| **count** | 28002.000000000000000 | 28002.000000000000000 | 28002.000000000000000 | 28002.000000 |
| **mean** | 15167.099207199486045 | 2.857045751662450 | 424.341162813950007 | 451.144319 |
| **std** | 9507.526863204127949 | 33.559229314698634 | 4293.489998072976050 | 4574.023467 |
| **min** | 0.000000000000000 | 0.000000000000000 | 39.062500000000000 | 37.413190 |
| **25%** | 7000.250000000000000 | 2.337500000000000 | 46.744790000000002 | 49.739582 |
| **50%** | 14239.000000000000000 | 2.409317000000000 | 49.088539123535213 | 51.475690 |
| **75%** | 22422.750000000000000 | 2.453588000000000 | 50.651040000000002 | 52.690969 |
| **max** | 32534.000000000000000 | 2125.000000000000000 | 53125.000000000000000 | 53125.000000 |

```
# no correr
#import pandas as pd
#feature_imp = pd.Series(clf.feature_importances_,index=iris.feature_names).sort_values(ascen
#feature_imp = pd.Series(clf.feature_importances_,index=sensor.columns[19:27]).sort_values(as
#print(feature_imp)

#Visualización
#import matplotlib.pyplot as plt
#import seaborn as sns
```

```
#import seaborn as sns
#%matplotlib inline
# Creating a bar plot
#sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
#plt.xlabel('Feature Importance Score')
#plt.ylabel('Features')
#plt.title("Visualizing Important Features")
#plt.legend()
#plt.show()



X=sensor[['sensor_00', 'sensor_01', 'sensor_02', 'sensor_03','sensor_04', 'sensor_11', 'senso
#y=sensor['target']  # Labels
y=sensor['machine_status']  # Labels

# Split dataset into training set and test set
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # 80% training and 2


from sklearn.ensemble import RandomForestClassifier

#Create a Random Forest Classifier
clf=RandomForestClassifier(n_estimators=100)

start = time.time()

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

stop = time.time()
print(f"Training time: {stop - start}s")

y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

#predicciones del item 17156 q es 1
clf.predict([[0.0,53.55902,52.77777,43.402774810790994,204.72509765625,3.7302410000000004,404

     Training time: 2.754838466644287s
     Accuracy: 0.9998214604534904
     array([1])


#predicciones
```

```
clf.predict([[0.0,53.55902,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])
```

```
array([1])
```

```
#predicciones
clf.predict([[0.0,53.55902,52.77777,43.402774810790994,204.72509765625,3.7302410000000004,404
```

```
array([1])
```

```
# Extract single tree
estimator = clf.estimators_[5]
```

```
#from sklearn.tree import export_graphviz
# Export as dot file
#export_graphviz(estimator, out_file='tree.dot',
#                feature_names = ['sensor_00', 'sensor_01', 'sensor_02', 'sensor_03','sensor_
#                class_names = [ 'machine_status'],
 #               rounded = True, proportion = False,
 #               precision = 2, filled = True)
```

```
# validacion cruzada
# https://jamesrledoux.com/code/k_fold_cross_validation
```

```
from sklearn.model_selection import cross_validate
```

```
start1 = time.time()
model = RandomForestClassifier(random_state=1)
cv = cross_validate(model, X, y, cv=10)
print(cv['test_score'])
print(cv['test_score'].mean())
stop1 = time.time()
print(f"Training time: {stop1 - start1}s")
```

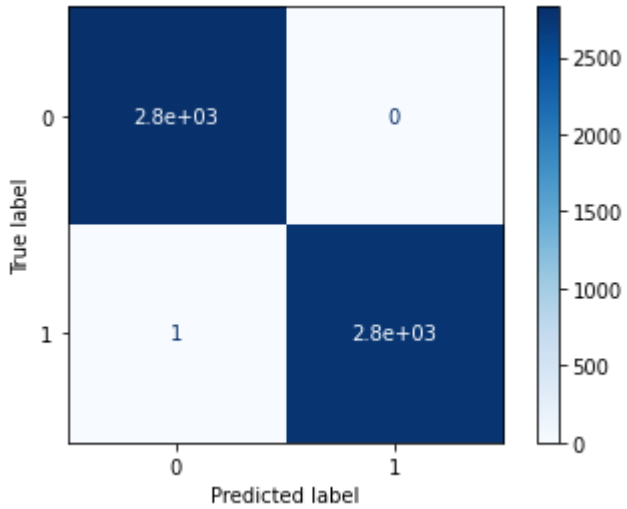
```
[0.99464477 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.85642857]
0.9850716325802009
Training time: 30.824281215667725s
```

```
#https://stackoverflow.com/questions/20662023/save-python-random-forest-model-to-file
```

```
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

start1 = time.time()
#model = RandomForestClassifier(random_state=1)
```

```
model = RandomForestClassifier(n_estimators=100)

cv = cross_validate(model, X, y, cv=10)
print(confusion_matrix(y_test,y_pred))
print(cv['test_score'])
print(cv['test_score'].mean())
stop1 = time.time()
print(f"Training time: {stop1 - start1}s")

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

#plot_confusion_matrix(clf, X_test, y_test)
# plot_confusion_matrix(clf, X_test, y_test)
plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)

plt.show()
```

```
[[2828    0]
 [   1 2772]]
[0.99678686 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.92321429]
0.9919644132197684
Training time: 30.822912454605103s
[[2828    0]
 [   1 2772]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2828
           1       1.00      1.00      1.00      2773

    accuracy                           1.00      5601
   macro avg       1.00      1.00      1.00      5601
weighted avg       1.00      1.00      1.00      5601
```




```
# version with multi scroring
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix

start1 = time.time()
#model = RandomForestClassifier(random_state=1)
model = RandomForestClassifier(n_estimators=100)

cv = cross_validate(model, X, y, cv=10)
#recall_score=cross_validation.cross_val_score(clf, X,y, cv=10, scoring ='recall')
#recall_score=cross_val_score(model, X,y, cv=10, scoring ='recall')
f1=cross_validate(model, X,y, cv=10, scoring ='f1')
recall_score=cross_validate(model, X,y, cv=10, scoring ='recall')
pre_score=cross_validate(model, X,y, cv=10, scoring ='precision_macro')
print(confusion_matrix(y_test,y_pred))
print(f"precision_macro_score:")
print(pre_score['test_score'])
print(pre_score['test_score'].mean())
print(f"test_score:")
print(cv['test_score'])
print(cv['test_score'].mean())
print(f"recall:")
print(recall_score['test_score'])
print(recall_score['test_score'].mean())
print(f"f1score:")
print(f1['test_score'])
print(f1['test_score'].mean())
stop1 = time.time()
print(f"Training time: {stop1 - start1}s")

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

#plot_confusion_matrix(clf, X_test, y_test)
# plot_confusion_matrix(clf, X_test, y_test)
plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)

plt.show()
```

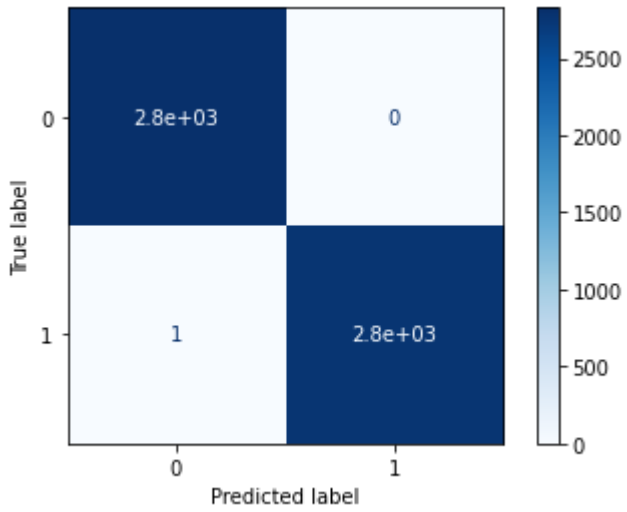
```
[[2828    0]
 [   1 2772]]
precision_macro_score:
[0.9978678  0.99964311 1.         1.         1.         1.
 1.         1.         1.         0.85199066]
0.9849501580362388
test_score:
[0.99892895 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.85785714]
0.9856429081450502
recall:
[0.99571429 0.99928622 1.         1.         1.         1.
 1.         1.         1.         0.99857143]
0.9993571938411339
f1score:
[0.99099099 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.83762732]
0.982826129738888
Training time: 127.10935044288635s
[[2828    0]
 [   1 2772]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2828
           1       1.00      1.00      1.00      2773

    accuracy                           1.00      5601
   macro avg       1.00      1.00      1.00      5601
weighted avg       1.00      1.00      1.00      5601
```

```
# version with multi scroring mejorada
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

start1 = time.time()
#model = RandomForestClassifier(random_state=1)
model = RandomForestClassifier(n_estimators=100)

#GH
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
#GH

cv = cross_validate(model, X, y, cv=10)
#recall_score=cross_validation.cross_val_score(clf, X,y, cv=10, scoring ='recall')
#recall_score=cross_val_score(model, X,y, cv=10, scoring ='recall')

#scoring = ['neg_mean_absolute_error','r2']

scores=cross_validate(model, X,y, cv=10, scoring = ['accuracy','f1','recall','precision'],ret
#recall_score=cross_validate(model, X,y, cv=10, scoring ='recall')
```

```
    #pre_score=cross_validate(model, X,y, cv=10, scoring ='precision_macro')
    print(confusion_matrix(y_test,y_pred))

    print(f"multi_metric_scores:")
    #print(scores['test_score'])
    print(scores)
    #print(scores['test_score'].mean())

    #print(scores.mean())

    #print(f"precision_macro_score:")
    #print(pre_score['test_score'])
    #print(pre_score['test_score'].mean())

    #print(f"test_score:")
    #print(cv['test_score'])
    #print(cv['test_score'].mean())

    #print(f"recall:")
    #print(recall_score['test_score'])
    #print(recall_score['test_score'].mean())

    #print(f"f1score:")
    #print(f1['test_score'])
    #print(f1['test_score'].mean())

    stop1 = time.time()
    print(f"Training time: {stop1 - start1}s")

    print(confusion_matrix(y_test,y_pred))
    print(classification_report(y_test,y_pred))

    #plot_confusion_matrix(clf, X_test, y_test)
    # plot_confusion_matrix(clf, X_test, y_test)
    plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)

    plt.show()

    gh4 = scores.get("test_accuracy")

    print(f"accuracy:")
    print(gh4)
    print(gh4.mean())

    gh3 = scores.get("test_precision")

    print(f"precision:")
    print(gh3)
    print(gh3.mean())

    gh = scores.get("test_recall")
```

```
print(f"recall:")
print(gh)
print(gh.mean())

gh2 = scores.get("test_f1")

print(f"f1:")
print(gh2)
print(gh2.mean())

CM = confusion_matrix(y_test, y_pred)
print(f"--------")
print(f"matriz de confusion:")
TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
print(f"TN={TN}, FP={FP} ")
print(f"FN={FN}, TP={TP} ")

print(f"--------")
print(f"matriz de confusion %:")
total1=(TN+TP+FN+FP)

print(f"TN={100*TN/total1}, FP={100*FP/total1} ")
print(f"FN={100*FN/total1}, TP={100*TP/total1} ")

print(f"--------")
acc1=(TN+TP)/(TN+TP+FN+FP)
print(f"accuracy1={acc1}")

print(f"--------")
re1=(TP)/(TP+FN)
print(f"reca1={re1}")

print(f"--------")
pre1=(TP)/(TP+FP)
print(f"pre1={pre1}")

print(f"--------")
f1s1=(2*pre1*re1)/(pre1+re1)
print(f"f1score={f1s1}")
```

```
[[2828    0]
 [   1 2772]]
multi_metric_scores:
{'fit_time': array([2.97870684, 3.05912113, 3.23872733, 3.08330417, 3.14520979,
       3.09104037, 3.29042935, 3.32355475, 3.08960247, 3.11018348]), 'score_time': array
       0.02597499, 0.0254004 , 0.02522922, 0.02625847, 0.02599835]), 'test_accuracy': ar
       1.        , 1.        , 1.        , 1.        , 0.86821429]), 'test_f1': array([0
       1.        , 1.        , 1.        , 1.        , 0.88341232]), 'test_recall': arra
       1.        , 1.        , 1.        , 1.        , 0.99857143]), 'test_precision': a
       1.        , 1.        , 1.        , 1.        , 0.79206799])}
Training time: 65.66045665740967s
[[2828    0]
 [   1 2772]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2828
           1       1.00      1.00      1.00      2773

    accuracy                           1.00      5601
   macro avg       1.00      1.00      1.00      5601
weighted avg       1.00      1.00      1.00      5601
```




```
accuracy:
[0.99821492 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.86821429]
0.9866072193604326
precision:
[1.         1.         1.         1.         1.         1.
 1.         1.         1.         0.79206799]
0.9792067988668555
recall:
[0.99642857 0.99928622 1.         1.         1.         1.
 1.         1.         1.         0.99857143]
0.9994286224125626
f1:
[0.99821109 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.88341232]
0.9881266398157569
--------
matriz de confusion:
TN=2828, FP=0
FN=1, TP=2772
```

```
--------
matriz de confusion %:
TN=50.490983752901265, FP=0.0
FN=0.017853954650955187, TP=49.491162292447775
--------
accuracy1=0.9998214604534904
```

```
# df[['gh4','gh3','gh','gh2']].describe()
# ['gh4','gh3','gh','gh2'].describe()

a1 = pd.Series([gh4, gh3, gh2,gh])
a1.describe()
```

```
count                                                     4
unique                                                    4
top       [0.998211091234347, 0.9996429846483399, 1.0, 1...
freq                                                      1
dtype: object
```

```
from scipy import stats

#stats.ttest_rel(df['bp_before'], df['bp_after'])
stats.ttest_rel(gh4, gh3)
#stats.ttest_rel(gh2, gh)
```

```
Ttest_relResult(statistic=0.9685818482493642, pvalue=0.3580607370167447)
```

```
# https://www.coursehero.com/file/p4hol16a/Aqu%C3%AD-prepresenta-la-proporci%C3%B3n-poblacion
```

```
import joblib
from sklearn.ensemble import RandomForestClassifier
# create RF
```

```
# save
joblib.dump(clf, "my_random_forest.joblib")
```

```
['my_random_forest.joblib']
```

```
# load
loaded_rf = joblib.load("my_random_forest.joblib")
```

```
#predicciones
#clf.predict([[0.0,53.55902,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])
#predicciones
loaded_rf.predict([[0.0,53.55902,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])
```

```
array([1])
```

```
# 1 es recovering
loaded_rf.predict([[0.0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])
```

```
array([1])
```

```
# 0 es recovering
loaded_rf.predict([[2.465394,47.092009999999995,53.2118,46.310759999999995,634375,47.52422,41
```

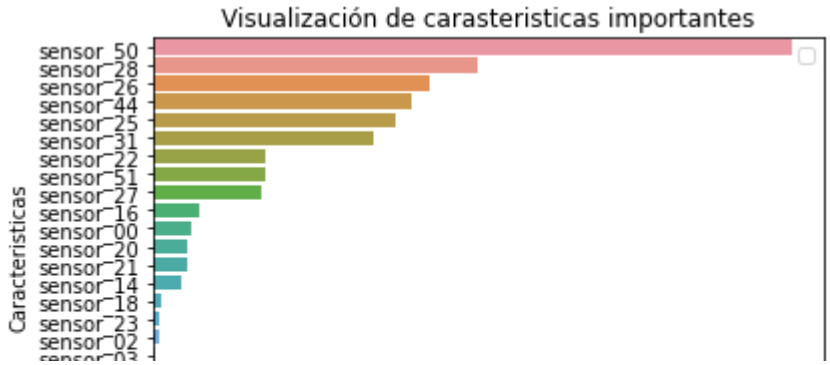
```
array([1])
```

```
# 2 es broken
loaded_rf.predict([[2.258796,47.26563,52.73437,43.4461784362793,200.11573791503898,43.62322,4
```

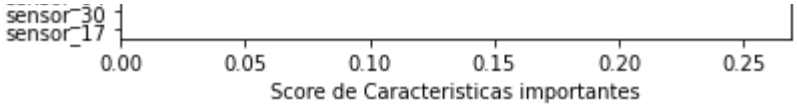
```
array([1])
```

```
import pandas as pd
#feature_imp = pd.Series(clf.feature_importances_,index=iris.feature_names).sort_values(ascen
#feature_imp = pd.Series(clf.feature_importances_,index=X.columns[1:8]).sort_values(ascending
feature_imp = pd.Series(clf.feature_importances_,index=X.columns[0:24]).sort_values(ascending
print(feature_imp)

#Visualización
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
#plt.xlabel('Feature Importance Score')
plt.xlabel('Score de Caracteristicas importantes')
#plt.ylabel('Features')
plt.ylabel('Caracteristicas')
#plt.title("Visualizing Important Features")
plt.title("Visualización de carasteristicas importantes")
plt.legend()
plt.show()

#plt.savefig('destination_path.eps', format='eps' , dpi=1000)

plt.savefig('myimage.svg', format='svg', dpi=1200)
```

```
No handles with labels found to put in legend.
sensor_50    0.256172814891090
sensor_28    0.129748334841976
sensor_26    0.110910456727885
sensor_44    0.103290519942941
sensor_25    0.096914121206027
sensor_31    0.088559052530764
sensor_22    0.044612182078668
sensor_51    0.044554635770717
sensor_27    0.043190798614105
sensor_16    0.018216469030686
sensor_00    0.015183402750871
sensor_20    0.013755244274610
sensor_21    0.013229563990397
sensor_14    0.011331161682492
sensor_18    0.003063875308692
sensor_23    0.002301928777814
sensor_02    0.001991737234993
sensor_03    0.001032824766116
sensor_19    0.000865319976897
sensor_11    0.000333454276206
sensor_01    0.000263867163295
sensor_04    0.000188620090685
sensor_30    0.000165257367289
sensor_17    0.000124356704784
dtype: float64
```




```
# https://towardsdatascience.com/how-to-visualize-a-decision-tree-from-a-random-forest-in-pyt
```




```
<Figure size 432x288 with 0 Axes>
```