In [1]:
```
import time
inicio = time.time()
```

In [2]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import LeaveOneOut

from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
```

# Se importan los datos

In [3]:
```
df = pd.read_excel("..//Base_datos_Clasificacion binaria.xlsx", index_col = 0)
```

In [4]:
```
df
```

Out[4]:

| | Licitacion reparto | Importe presupuestado | Importe adjudicado | MP | Empresa sancionada | UTE |
|---|---|---|---|---|---|---|
| **0** | 0 | 1738093.21 | 1484428.72 | 6 | 1 | 0 |
| **1** | 0 | 469670.24 | 272492.00 | 4 | 0 | 0 |
| **2** | 0 | 1025088.19 | 707310.85 | 1 | 0 | 0 |
| **3** | 0 | 999890.00 | 497621.36 | 6 | 0 | 0 |
| **4** | 0 | 72598.27 | 47508.92 | 1 | 1 | 0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **2211** | 1 | 5542028.88 | 5514320.00 | 6 | 1 | 0 |
| **2212** | 1 | 6095782.00 | 5100555.00 | 6 | 1 | 0 |
| **2213** | 1 | 3752906.00 | 3744400.00 | 6 | 1 | 0 |
| **2214** | 1 | 23896564.00 | 19547338.00 | 6 | 0 | 1 |
| **2215** | 1 | 33415098.00 | 33349532.00 | 6 | 1 | 0 |

2216 rows × 6 columns

# Se estandarizan y normalizan los predictores numericos

In [5]:
```
predictores_numericos = df[["Importe presupuestado", "Importe adjudicado"]]
```

In [6]:
```
scaler = StandardScaler()
stand = scaler.fit_transform(predictores_numericos)
```

In [7]:
```
df[["Importe presupuestado", "Importe adjudicado"]] = stand
predictores_numericos = df[["Importe presupuestado", "Importe adjudicado"]]
predictores_numericos
```

Out[7]:

| | Importe presupuestado | Importe adjudicado |
|---|---|---|
| **0** | -0.205390 | -0.212546 |
| **1** | -0.237976 | -0.248383 |
| **2** | -0.223707 | -0.235525 |
| **3** | -0.224354 | -0.241726 |
| **4** | -0.248176 | -0.255036 |
| **...** | ... | ... |
| **2211** | -0.107667 | -0.093383 |
| **2212** | -0.093441 | -0.105618 |
| **2213** | -0.153630 | -0.145719 |
| **2214** | 0.363860 | 0.321573 |
| **2215** | 0.608391 | 0.729702 |

2216 rows × 2 columns

In [8]:
```
norm = (predictores_numericos - predictores_numericos.min()) / (predictores_numeri
```

In [9]:
```
df[["Importe presupuestado", "Importe adjudicado"]] = norm
predictores_numericos = df[["Importe presupuestado", "Importe adjudicado"]]
predictores_numericos
```

Out[9]:

| | Importe presupuestado | Importe adjudicado |
|---|---|---|
| **0** | 0.002679 | 0.002861 |
| **1** | 0.000699 | 0.000494 |
| **2** | 0.001566 | 0.001343 |
| **3** | 0.001526 | 0.000934 |
| **4** | 0.000079 | 0.000054 |
| **...** | ... | ... |
| **2211** | 0.008615 | 0.010734 |
| **2212** | 0.009479 | 0.009925 |
| **2213** | 0.005823 | 0.007276 |
| **2214** | 0.037259 | 0.038147 |
| **2215** | 0.052114 | 0.065109 |

2216 rows × 2 columns

In [10]:
```
df
```

Out[10]:

| | Licitacion reparto | Importe presupuestado | Importe adjudicado | MP | Empresa sancionada | UTE |
|---|---|---|---|---|---|---|
| **0** | 0 | 0.002679 | 0.002861 | 6 | 1 | 0 |
| **1** | 0 | 0.000699 | 0.000494 | 4 | 0 | 0 |
| **2** | 0 | 0.001566 | 0.001343 | 1 | 0 | 0 |
| **3** | 0 | 0.001526 | 0.000934 | 6 | 0 | 0 |
| **4** | 0 | 0.000079 | 0.000054 | 1 | 1 | 0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **2211** | 1 | 0.008615 | 0.010734 | 6 | 1 | 0 |
| **2212** | 1 | 0.009479 | 0.009925 | 6 | 1 | 0 |
| **2213** | 1 | 0.005823 | 0.007276 | 6 | 1 | 0 |
| **2214** | 1 | 0.037259 | 0.038147 | 6 | 0 | 1 |
| **2215** | 1 | 0.052114 | 0.065109 | 6 | 1 | 0 |

2216 rows × 6 columns

# Codificacion OneHotEncoder

In [11]:
```
codificador = OneHotEncoder()
```

In [12]:
```
df[["MP", "Empresa sancionada", "UTE"]] = df[["MP", "Empresa sancionada", "UTE"]].
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2216 entries, 0 to 2215
Data columns (total 6 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Licitacion reparto     2216 non-null   int64
 1   Importe presupuestado  2216 non-null   float64
 2   Importe adjudicado     2216 non-null   float64
 3   MP                     2216 non-null   category
 4   Empresa sancionada     2216 non-null   category
 5   UTE                    2216 non-null   category
dtypes: category(3), float64(2), int64(1)
memory usage: 76.1 KB
```

In [13]:
```
codificacion = codificador.fit_transform(df[["MP"]])

mercados = pd.DataFrame(codificacion.toarray(),
                        columns = ["MP1", "MP4", "MP6"])
df = pd.concat([df, mercados], axis = 1)
df.drop("MP", axis = 1, inplace = True)
df
```

Out[13]:

| | Licitacion reparto | Importe presupuestado | Importe adjudicado | Empresa sancionada | UTE | MP1 | MP4 | MP6 |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0.002679 | 0.002861 | 1 | 0 | 0.0 | 0.0 | 1.0 |
| **1** | 0 | 0.000699 | 0.000494 | 0 | 0 | 0.0 | 1.0 | 0.0 |
| **2** | 0 | 0.001566 | 0.001343 | 0 | 0 | 1.0 | 0.0 | 0.0 |
| **3** | 0 | 0.001526 | 0.000934 | 0 | 0 | 0.0 | 0.0 | 1.0 |
| **4** | 0 | 0.000079 | 0.000054 | 1 | 0 | 1.0 | 0.0 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **2211** | 1 | 0.008615 | 0.010734 | 1 | 0 | 0.0 | 0.0 | 1.0 |
| **2212** | 1 | 0.009479 | 0.009925 | 1 | 0 | 0.0 | 0.0 | 1.0 |
| **2213** | 1 | 0.005823 | 0.007276 | 1 | 0 | 0.0 | 0.0 | 1.0 |
| **2214** | 1 | 0.037259 | 0.038147 | 0 | 1 | 0.0 | 0.0 | 1.0 |
| **2215** | 1 | 0.052114 | 0.065109 | 1 | 0 | 0.0 | 0.0 | 1.0 |

2216 rows × 8 columns

In [14]:
```
nombre_clases = ["Licitacion no reparto", "Licitacion reparto"]
nombre_predicciones = list(map(lambda x : x + " predicho", nombre_clases))
```

# Se hace la estimacion por NaiveBayes

In [15]:
```
X = df[df.columns[1:]]
y = df["Licitacion reparto"]

loo = LeaveOneOut()
gnb = GaussianNB()
predicciones = []
prob_predicciones = []

for train_index, test_index in loo.split(X):
```

```
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    gnb.fit(X_train, y_train)
    y_pred = gnb.predict(X_test) #son 0 o 1 dependiendo de si la probabilidad es m
    y_pred_prob = gnb.predict_proba(X_test)
    predicciones.append(y_pred[0])
    prob_predicciones.append(y_pred_prob[0][1])
```

In [16]:
```
gnb.get_params()
```

Out[16]:
```
{'priors': None, 'var_smoothing': 1e-09}
```

# CURVA ROC. El punto optimo es = (fpr = 0, tpr = 1)

In [17]:
```
auc_score = roc_auc_score(y, prob_predicciones)
auc_score
```

Out[17]:
```
0.7775759863235718
```

In [18]:
```
fpr, tpr, thresholds = roc_curve(y, prob_predicciones)
#fpr_gnb es 1 - especificidad = 1 - (TN / (FP + TN)) = FP / (FP + TN)
#tpr_gnb es la sensibilidad = TP / (TP + FN)
```

In [19]:
```
distances = np.linalg.norm(np.column_stack((fpr, tpr)) - np.array([0, 1]), axis=1)
optimal_threshold_index = np.argmin(distances)
optimal_threshold = thresholds[optimal_threshold_index]
optimal_threshold
```

Out[19]:
```
0.01844043297363063
```

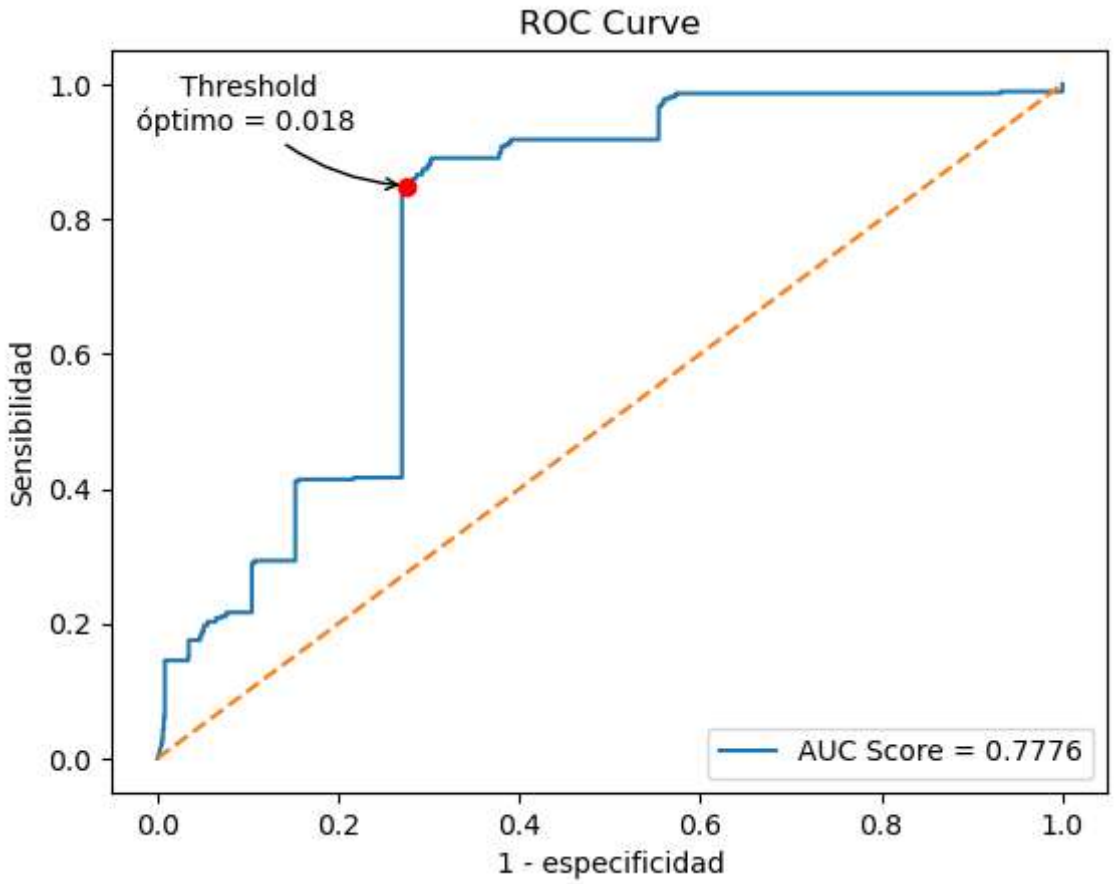
In [20]:
```
optimal_fpr = fpr[optimal_threshold_index]
optimal_fpr
```

Out[20]:
```
0.2744462452728255
```

In [21]:
```
optimal_tpr = tpr[optimal_threshold_index]
optimal_tpr
```

Out[21]:
```
0.8493150684931506
```

In [22]:
```
plt.plot(fpr, tpr, label = f'AUC Score = {auc_score:.4}')
plt.plot([0,1], linestyle="--")
plt.plot(optimal_fpr, optimal_tpr, 'ro')
plt.annotate(f'      Threshold\n óptimo = {optimal_threshold:.2}',
             xy=(optimal_fpr, optimal_tpr), xycoords='data',
             xytext=(-100, +20), textcoords='offset points', fontsize=10,
             arrowprops=dict(arrowstyle="->", connectionstyle = "arc3, rad = .2"))
plt.ylabel('Sensibilidad')
plt.xlabel('1 - especificidad')
plt.title('ROC Curve')
plt.legend(loc = 'lower right')
plt.show()
```




# Metricas de la Matriz de Confusion

In [23]:
```
predicciones = pd.Series(predicciones, name = "Prediccion licitacion reparto")
prob_predicciones = pd.Series(prob_predicciones, name = "Prob prediccion")
cp = pd.concat([y, predicciones, prob_predicciones], axis = 1)
cp
# Este dataframe no contiene las predicciones calculadas en el punto optimo de la c
# Contiene las predicciones calculadas segun el punto por defecto para determinar l
```

Out[23]:

| | Licitacion reparto | Prediccion licitacion reparto | Prob prediccion |
|---|---|---|---|
| **0** | 0 | 0 | 0.002614 |
| **1** | 0 | 1 | 0.998470 |
| **2** | 0 | 0 | 0.156212 |
| **3** | 0 | 0 | 0.001099 |
| **4** | 0 | 0 | 0.307545 |
| **...** | ... | ... | ... |
| **2211** | 1 | 0 | 0.002516 |
| **2212** | 1 | 0 | 0.002516 |
| **2213** | 1 | 0 | 0.002504 |
| **2214** | 1 | 0 | 0.006319 |
| **2215** | 1 | 0 | 0.005768 |

2216 rows × 3 columns

In [24]:
```
cp["Prediccion licitacion reparto en el punto optimo"] = 0
prediccion_final = cp["Prediccion licitacion reparto en el punto optimo"]
prediccion_final = cp.apply(lambda x: 1 if x["Prob prediccion"] >= optimal_thresho
prediccion_final
```

Out[24]:
```
0       0
1       1
2       1
3       0
4       1
       ..
2211    0
2212    0
2213    0
2214    0
2215    0
Length: 2216, dtype: int64
```

In [25]:
```
cp["Prediccion licitacion reparto en el punto optimo"] = prediccion_final
pd.concat([cp, prediccion_final], axis = 1)
cp.drop(columns = "Prediccion licitacion reparto", axis = 1, inplace = True)
cp
```

Out[25]:

| | Licitacion reparto | Prob prediccion | Prediccion licitacion reparto en el punto optimo |
|---|---|---|---|
| **0** | 0 | 0.002614 | 0 |
| **1** | 0 | 0.998470 | 1 |
| **2** | 0 | 0.156212 | 1 |
| **3** | 0 | 0.001099 | 0 |
| **4** | 0 | 0.307545 | 1 |
| **...** | ... | ... | ... |
| **2211** | 1 | 0.002516 | 0 |
| **2212** | 1 | 0.002516 | 0 |
| **2213** | 1 | 0.002504 | 0 |
| **2214** | 1 | 0.006319 | 0 |
| **2215** | 1 | 0.005768 | 0 |

2216 rows × 3 columns

In [26]:
```
cm = confusion_matrix(y, cp["Prediccion licitacion reparto en el punto optimo"]) #
cm = pd.DataFrame(cm, columns = nombre_clases, index = nombre_predicciones)
cm # esta matriz de confusion esta calculada en el punto optimo de la curva
```

Out[26]:

| | Licitacion no reparto | Licitacion reparto |
|---|---|---|
| **Licitacion no reparto predicho** | 1343 | 508 |
| **Licitacion reparto predicho** | 55 | 310 |

In [27]:
```
TP = cm.iloc[1,1]
FN = cm.iloc[1,0]
FP = cm.iloc[0,1]
TN = cm.iloc[0,0]
```

In [28]:
```
Tasa_de_aciertos = (TP + TN) / len(df)
Tasa_de_aciertos
```

Out[28]:
```
0.7459386281588448
```

In [29]:
```
Tasa_de_errores = 1 - Tasa_de_aciertos
Tasa_de_errores
```

Out[29]:
```
0.2540613718411552
```

In [30]:
```
Sensibilidad = TP / (TP + FN)
Sensibilidad # se calculo antes. Es optimal_tpr
```

Out[30]:
```
0.8493150684931506
```

In [31]:
```
Tasa_de_falsos_negativos = 1 - Sensibilidad
Tasa_de_falsos_negativos
```

Out[31]:
```
0.15068493150684936
```

In [32]:
```
Especificidad = TN / (FP + TN)
Especificidad # se calculo antes. Es optimal_fpr = 1 - Especificidad
```

Out[32]:
```
0.7255537547271745
```

In [33]:
```
Precision = TP / (TP + FP)
Precision
```

Out[33]:
```
0.37897310513447435
```

In [34]:
```
FP / (FP + TP) # 1 - Precision
```

Out[34]:
```
0.6210268948655256
```

In [35]:
```
F1 = (2 * Sensibilidad * Precision) / (Sensibilidad + Precision)
F1
```

Out[35]:
```
0.5240912933220626
```

# Métricas de clasificación binaria en el punto óptimo

In [36]:
```
nombre_fichero = "Clasificacion binaria gnb"
```

In [37]:
```
indice_resultados = ["Tasa de aciertos", "Sensibilidad", "Especificidad", "Precisi
resultados_valores = [Tasa_de_aciertos, Sensibilidad, Especificidad, Precision, F1

resultados = pd.DataFrame(data={f"{nombre_fichero}": resultados_valores},
                          index=indice_resultados)
resultados
```

Out[37]:

| | Clasificacion binaria gnb |
|---|---|
| **Tasa de aciertos** | 0.745939 |
| **Sensibilidad** | 0.849315 |
| **Especificidad** | 0.725554 |
| **Precision** | 0.378973 |
| **F1** | 0.524091 |

## Se exportan los datos

In [38]:
```
nombre_columna = list(resultados.columns)[0]
resultados.to_excel(f"Resultados de las métricas de {nombre_fichero}.xlsx")
```

# CURVA PR. El punto óptimo es = (precision = 1, recall = 1)

In [39]:
```
average_precision = average_precision_score(y, prob_predicciones)
average_precision
```

Out[39]:
```
0.3662153113591662
```

In [40]:
```
precision, recall, thresholds = precision_recall_curve(y, prob_predicciones)
#precision = TP / (TP + FP)
#recall = sensibilidad = TP / (TP + FN)
```

In [41]:
```
distances = np.linalg.norm(np.column_stack((precision, recall)) - np.array([1, 1])
optimal_threshold_index = np.argmin(distances)
optimal_threshold = thresholds[optimal_threshold_index]
optimal_threshold
```

Out[41]:
```
0.01844043297363063
```

In [42]:
```
optimal_precision = precision[optimal_threshold_index]
optimal_precision
```

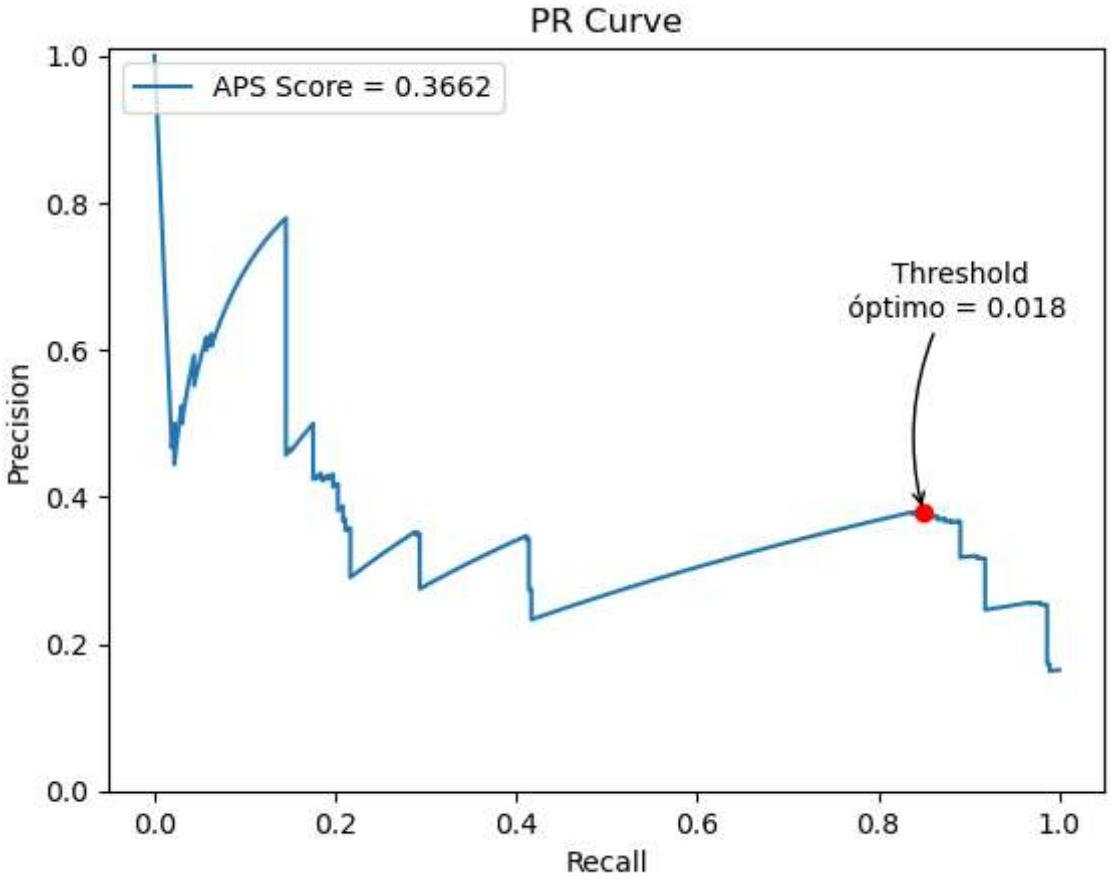
Out[42]:
```
0.37897310513447435
```

In [43]:
```
optimal_recall = recall[optimal_threshold_index]
optimal_recall
```

Out[43]:
```
0.8493150684931506
```

In [44]:
```
plt.plot(recall, precision, label = f'APS Score = {average_precision:.4}')
plt.plot(optimal_recall, optimal_precision, 'ro')
plt.annotate(f'      Threshold\n óptimo = {optimal_threshold:.2}',
             xy=(optimal_recall, optimal_precision), xycoords='data',
             xytext=(-30, +70) ,textcoords='offset points', fontsize=10,
             arrowprops=dict(arrowstyle="->", connectionstyle = "arc3, rad = .2"))
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.title('PR Curve')
plt.legend(loc = 'upper left')
```

```
plt.ylim(0, 1.01)
plt.show()
```




# Se calcula cuanto tiempo ha transcurrido en la ejecucion de todo el notebook y se guarda el tiempo en un fichero de texto para almacenarlo.

In [45]:
```
fin = time.time()
tiempo_transcurrido = fin - inicio
minutos = int((tiempo_transcurrido % 3600) // 60)
segundos = int(tiempo_transcurrido % 60)
with open(f"Tiempo de ejecucion transcurrido en el notebook {nombre_fichero}.txt",
    f.write(f"Tiempo transcurrido: {minutos} minutos, {segundos} segundos")
```

In [46]:
```
print(f"Tiempo transcurrido: {minutos} minutos, {segundos} segundos")
```

```
Tiempo transcurrido: 0 minutos, 10 segundos
```