1 Sistema de clasificación de estilo de manejo

1.1 1. Importando módulos

```
In [1]: import os
        import pandas as pd
        import numpy as np
        import warnings
        import matplotlib.pyplot as plt
        import matplotlib.gridspec as gridspec
        import seaborn as sns
        sns.set()
        from statsmodels.graphics.mosaicplot import mosaic
        import math
        from sklearn.manifold import LocallyLinearEmbedding
        from mpl_toolkits.mplot3d import Axes3D
```

1.2 2. Definir funciones

Se define una función para leer los 38 archivos 'csv' y otra para dividir a los archivos en ventanas de tamaño w

```
In [2]: def car_trip_filename(n):
            if (n<1) | (n>38):
                print('n out of the scope')
                return np.NAN
            route_data_kaggle='../Datos/Car trips data log/TripData/Processed Data'
            processed_file_name='fileID'+str(n)+'_ProcessedTripData.csv'
            data = pd.read_csv(route_data_kaggle + '/' + processed_file_name, delimiter = ',',
                               header=None,
                               names=['Time(s)',
                                      'Speed(m/s)',
                                      'Shift_number',
                                      'Engine_Load(%)',
                                      'Total_Acceleration(m/s^2)',
                                      'Engine RPM',
                                      'Pitch',
                                      'Lateral_Acceleration(m/s^2)',
                                      'Passenger_count(0-5)',
                                      'Load(0-10)',
                                      'Air_conditioning(0-4)',
                                      'Window_opening(0-10)',
                                      'Radio_volume(0-10)',
                                      'Rain_intensity(0-10)',
                                      'Visibility',
                                      'Driver_wellbeing(0-10)',
                                      'Driver_rush(0-10)'])
            return data

        def read_all():
            Data=[]
            for n_file in range(1,39):
                Data.append(car_trip_filename(n_file))
            return Data

        #Obtiene una lista con los valores de duración en segundos de cada
        def duracion_total_sec(Data):
            duration_sec=[]
            for m in range(len(Data)):
                duration_sec.append(Data[m].iloc[Data[m].shape[0]-1,0]-Data[m].iloc[0,0])
            return duration_sec

        # Obtiene un diccionario en el que las 'keys' son los indices de Data
        #(representando el número de archivo) y
        # el 'value' es una lista con la duración entre cada fila en segundos
        def dict_with_step(Data):
            dict_period={}
            for m in range(len(Data)):
                time_dif=[]
                for i in range(Data[m].shape[0]):
```

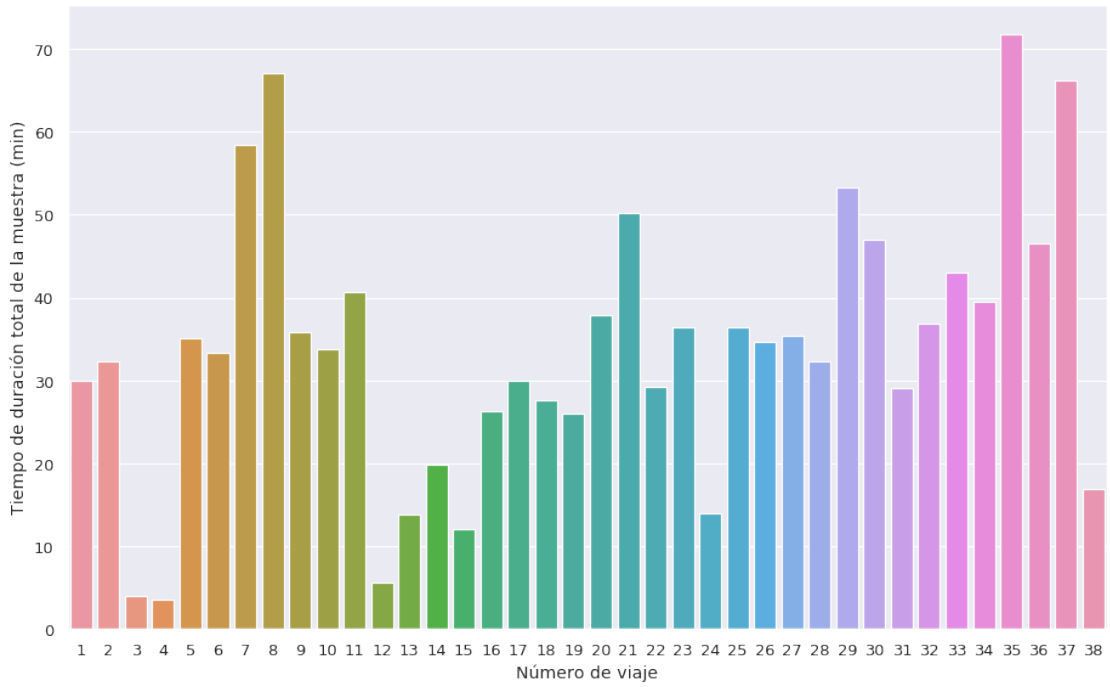
```
            if i == 0:
                time_dif.append(Data[m].iloc[i,0])
            else:
                time_dif.append(Data[m].iloc[i,0]-Data[m].iloc[i-1,0])
        dict_period[m]=time_dif
    return dict_period
```

Leemos la data de todos los archivos

```
In [3]: Data=read_all()
```

Para encontrar un tamaño de ventana adecuado, observamos las longitudes de cada archivo

```
In [4]: duracion = duracion_total_sec(Data)

        dur_min=np.asarray(duracion)/60
        fig1= plt.figure(figsize=(13,8),dpi=85)
        sns.barplot(list(range(1,len(Data)+1)),dur_min)
        plt.ylabel('Tiempo de duración total de la muestra (min)')
        plt.xlabel('Número de viaje')
        fig1.savefig("./Figuras/duracion_viajes.pdf", bbox_inches='tight')
```



Como podemos observar, cada archivo tiene una longitud distinta, por lo que la ventana w no debe poder exceder la longitud total de uno de estos archivos. En este caso el w_max que se puede escoger es el tamaño del archivo más pequeño, el archivo 4.

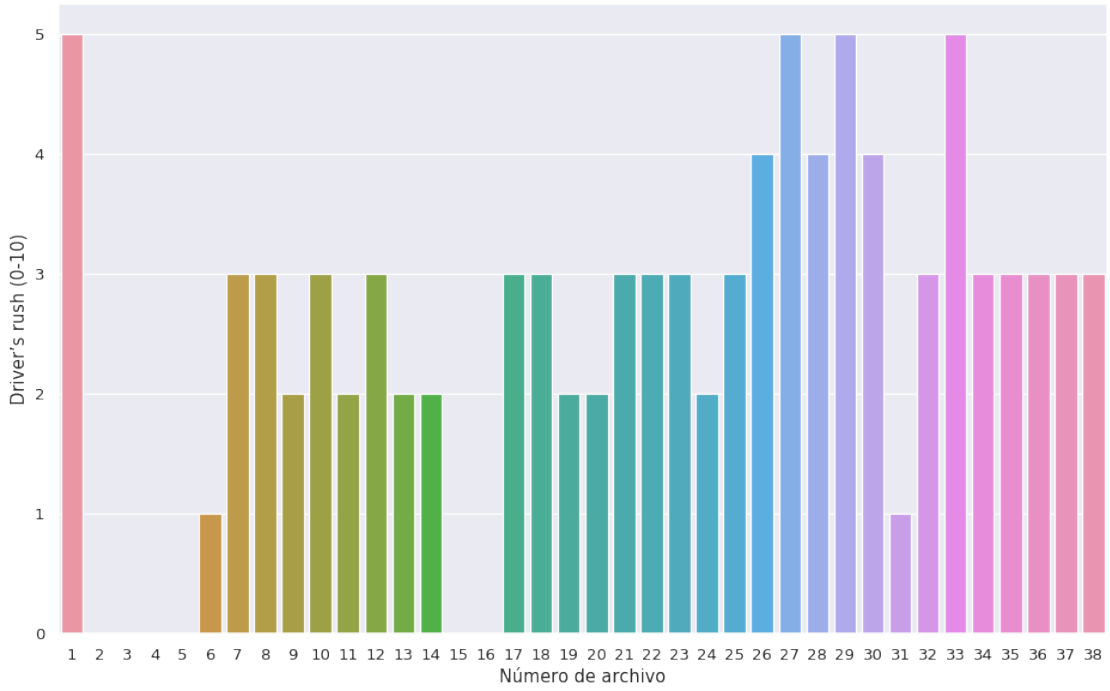
```
In [5]: w_max=Data[3].shape[0]
        print(w_max)
        w_time_aprox=w_max*0.01
        print(w_time_aprox)

21230
212.3
```

Entonces el tamaño máximo de ventana a escoger es w_max = 21230 puntos temporales. Teniendo en cuenta que entre cada punto temporal hay siempre un aproximado de 0.01 s (La data se tomo a 100 Hz). El tamaño máximo de ventana sería aquivalente a 212 segundos.

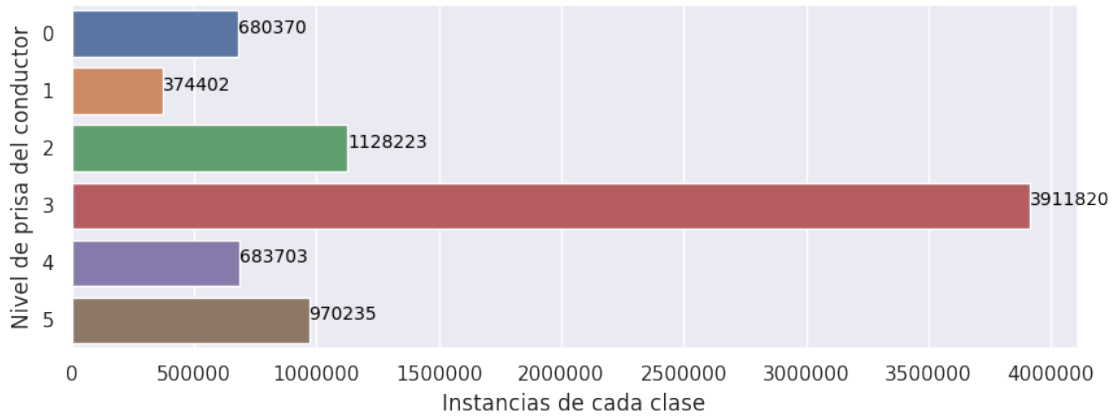
1.3 3. Clases

Se considerará como clase al feature categórico: 'Driver's rush(0-10)' ya que se verá que este feature tiene un solo valor para cada archivo. Este valor representará el estilo de conducción en este conjunto de datos.

```
In [6]: fig1= plt.figure(figsize=(13,8), dpi=87)
        sns.barplot(list(range(1,39)),
                    [int(Data[m].drop_duplicates('Driver_rush(0-10)')
                    ['Driver_rush(0-10)'].values) for m in range(len(Data))])
        plt.ylabel('Driver's rush (0-10)')
        plt.xlabel('Número de archivo')

Out[6]: Text(0.5, 0, 'Número de archivo')
```



```
In [7]: clases_total=[]
        for archivo in Data:
            for dato in archivo['Driver_rush(0-10)']:
                clases_total.append(dato )


        classes, counts = np.unique(clases_total, return_counts=True)
        classes_n_dict = dict(zip(classes, counts))

        fig1= plt.figure(figsize=(10,3.5), dpi=100)
        g=sns.barplot(list(classes_n_dict.values()),list(classes_n_dict.keys()), orient='h')
        plt.ylabel('Nivel de prisa del conductor')
        plt.xlabel('Instancias de cada clase')

        for i, v in enumerate(list(classes_n_dict.values())):
            g.text(v + 3, i, str(v), color='black')

        fig1.savefig("./Figuras/instancia_clases.pdf", bbox_inches='tight')
```

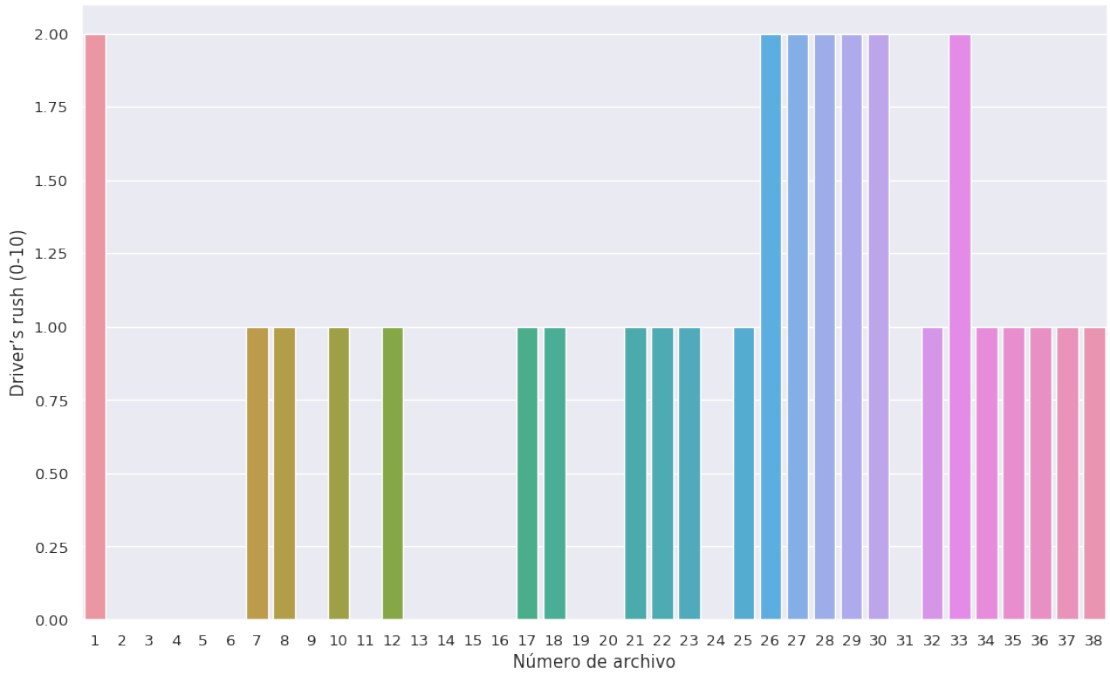


Como se observa en la gráfica, podemos encontrar valores desde 0 a 5, pero se reducirán a 3 valores para realizar la clasificación: 0 -> Tranquilo (0,1,2 en valores de Driver's Rush) 1 -> Normal (3 en valores de Driver's Rush) 2 -> Agresivo (4,5 en valores de Driver's Rush)

```
In [8]: for m in range(len(Data)):
            old = Data[m]['Driver_rush(0-10)'].values[0]
            if old<=2:
                Data[m]['Driver_rush(0-10)']='0'
            elif (old==3):
                Data[m]['Driver_rush(0-10)']='1'
            else:
                Data[m]['Driver_rush(0-10)']='2'
```

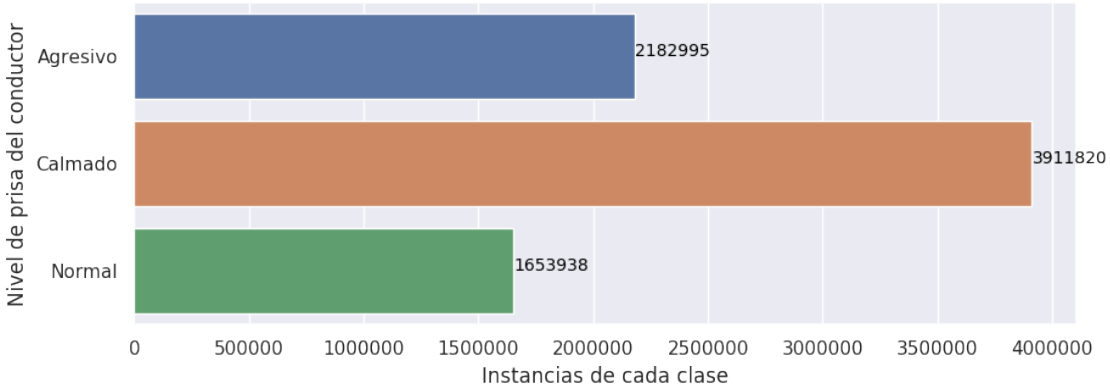
In [9]:
```
fig1= plt.figure(figsize=(13,8), dpi=87)
sns.barplot(list(range(1,39)),[int(Data[m].drop_duplicates('Driver_rush(0-10)')
            ['Driver_rush(0-10)'].values) for m in range(len(Data))])
plt.ylabel('Driver's rush (0-10)')
plt.xlabel('Número de archivo')
```

Out[9]: Text(0.5, 0, 'Número de archivo')



In [10]:
```
clases_total=[]

for archivo in Data:
    for dato in archivo['Driver_rush(0-10)']:
        clases_total.append(dato )

classes, counts = np.unique(clases_total, return_counts=True)
classes=['Agresivo','Calmado','Normal']
classes_n_dict = dict(zip(classes, counts))

fig1= plt.figure(figsize=(10,3.5), dpi=100)

g=sns.barplot(list(classes_n_dict.values()),list(classes_n_dict.keys()), orient='h')
plt.ylabel('Nivel de prisa del conductor')
plt.xlabel('Instancias de cada clase')

for i, v in enumerate(list(classes_n_dict.values())):
```

```
        g.text(v + 3, i, str(v), color='black')
    fig1.savefig("./Figuras/instancia_clases.pdf", bbox_inches='tight')
```



1.4 4. Dividimos la data en ventanas de tamaño w y extraemos los features

Ahora definimos las features que vamos a extraer. Se usarán lso siguientes parámetros de series de tiempo.

Table 1
The feature parameters.

Time-domain feature parameters	Frequency-domain feature parameters	
$x_{rms} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} x^2(n)}$	$F_C = \frac{\sum_{n=1}^{N} \bar{x}(n)x(n)}{2\pi\sum_{n=1}^{N} x(n)^2}$	$p_5 = \frac{\sum_{k=1}^{H} f_k s(k)}{\sum_{k=1}^{H} s(k)}$
$x_{p-p} = x_{\max} - x_{\min}$	$F_{MS} = \frac{\sum_{n=1}^{N} \bar{x}(n)^2}{4\pi^2\sum_{n=1}^{N} x(n)^2}$	$p_6 = \sqrt{\frac{\sum_{k=1}^{H}(f_k - p_5)^2 s(k)}{H}}$
$S_f = \frac{x_{rms}}{\left\lvert\frac{1}{N}\sum_{n=1}^{N} x(n)\right\rvert}$	$F_{RMS} = \sqrt{F_{MS}}$	$p_7 = \sqrt{\frac{\sum_{k=1}^{H} f_k^2 s(k)}{\sum_{k=1}^{H} s(k)}}$
$C_f = \frac{x_{\max}}{x_{rms}}$	$F_V = F_{MS} - (F_C)^2$	$p_8 = \sqrt{\frac{\sum_{k=1}^{H} f_k^4 s(k)}{\sum_{k=1}^{H} f_k^2 s(k)}}$
$I_f = \frac{x_{\max}}{\left\lvert\frac{1}{N}\sum_{n=1}^{N} x(n)\right\rvert}$	$p_1 = \frac{\sum_{k=1}^{H} s(k)}{H}$	$p_9 = \frac{\sum_{k=1}^{H} f_k^2 s(k)}{\sqrt{\sum_{k=1}^{H} s(k)\sum_{k=1}^{H} f_k^4 s(k)}}$
$CL_f = \frac{x_{\max}}{\left\lvert\frac{1}{N}\sum_{n=1}^{N}\sqrt{\lvert x(n)\rvert}\right\rvert^2}$	$p_2 = \frac{\sum_{k=1}^{H}(s(k) - p_1)^2}{H-1}$	$p_{10} = \frac{p_6}{p_5}$
$K_v = \frac{\frac{1}{N}\sum_{n=1}^{N} x^4(n)}{x_{rms}^4}$	$p_3 = \frac{\sum_{k=1}^{H}(s(k) - p_1)^3}{H(\sqrt{p_2})^3}$	$p_{11} = \frac{\sum_{k=1}^{H}(f_k - p_5)^3 s(k)}{H p_6^3}$
where $x(n)$ is a signal series for $n = 1,2,...,N$, N is the number of data points.	$p_4 = \frac{\sum_{k=1}^{H}(s(k) - p_1)^4}{H(p_2)^2}$	$p_{12} = \frac{\sum_{k=1}^{H}(f_k - p_5)^4 s(k)}{H F_6^4}$
	where $s(k)$ is a spectrum for $k = 1,2,...,H$, H is the number of spectrum lines: f_k is the frequency value of the kth spectrum line.	

Entonces ahora definimos los features en el dominio del tiempo que hallaremos para cada serie temporal

```
In [11]: #En todas las funciones x es un array que representa un time-serie
         def get_x_rms(x):
             N=x.shape[0]
             return np.power((np.sum(np.power(x, 2)))/N, 0.5)
```

```
def get_x_p_p(x):
    return x.max()-x.min()

def get_S_f(x):
    N=x.shape[0]
    return np.power((np.sum(np.power(x, 2)))/N, 0.5)/abs(np.sum(x)/N)

def get_C_f(x):
    N=x.shape[0]
    return x.max()/np.power((np.sum(np.power(x, 2)))/N, 0.5)

def get_I_f(x):
    N=x.shape[0]
    return x.max()/abs(np.sum(x)/N)

def get_CL_f(x):
    N=x.shape[0]
    return x.max()/np.power(np.sum(np.power(abs(x),0.5))/N,2)

def get_K_v(x):
    N=x.shape[0]
    return np.sum(np.power(x,4))/(np.power((np.sum(np.power(x, 2)))/N, 2)*N)


# df: Es el dataframe cuyas columnas son los time-series de los que se quiere
# extraer los features
def get_features(df,col_prefix=None):
    col_sufix=['x_rms','x_p_p','S_f','C_f','I_f','CL_f','K_v']

    x_rms_array=[]
    x_p_p_array=[]
    S_f_array=[]
    C_f_array=[]
    I_f_array=[]
    CL_f_array=[]
    K_v_array=[]

    for col in range(df.shape[1]):
        x_rms_array.append(get_x_rms(df.iloc[:,col].values))
        x_p_p_array.append(get_x_p_p(df.iloc[:,col].values))
        S_f_array.append(get_S_f(df.iloc[:,col].values))
        C_f_array.append(get_C_f(df.iloc[:,col].values))
        I_f_array.append(get_I_f(df.iloc[:,col].values))
        CL_f_array.append(get_CL_f(df.iloc[:,col].values))
        K_v_array.append(get_K_v(df.iloc[:,col].values))

    features=[x_rms_array,x_p_p_array,S_f_array,C_f_array,
```

```
                I_f_array,CL_f_array,K_v_array]

    if col_prefix==None:
        col_prefix=df.columns

    array_out=[]
    indexes=[]
    for col in range(df.shape[1]):
        for feat in range(7):
            indexes.append(col_prefix[col]+ '_' + col_sufix[feat])
            array_out.append(features[feat][col])

    series_out=pd.Series(array_out,indexes)
    return series_out

def concat_array_df(array_of_df):
    df_out=array_of_df[0].copy()
    for i in range(1,len(array_of_df)):
        df_out=pd.concat([df_out,array_of_df[i]],ignore_index=True)
    return df_out

# w: tamaño de ventana (número de filas)
# Data: Lista en la que cada elemento es un Dataframe que representa a un archivo
# columns: (Opcional) lista con los labels de las columnas que se quieren tener.
#La última columna se asume como clase
def get_df_features_w(w,Data,columns=None):
    X_array=[]
    Y_array=[]
    if columns == None:
        columns=Data[0].columns
    for m in range(len(Data)):
        row_index=0

        if Data[m].shape[0]>=w:
            rows_of_windows=math.floor(Data[m].shape[0]/w)

            for n in range(rows_of_windows):
                Xdf_temp=Data[m][columns].iloc[row_index:row_index+w,:-1]
                #Extraemos los features y ahora tenemos una fila por cada ventana w
                Xdf_temp=get_features(Xdf_temp)
                X_array.append(Xdf_temp)

                Ydf_temp=Data[m].iloc[row_index,-1]#Se toma solo el primer elemento
                #porque la clase se repite y es la misma para cada archivo
                Y_array.append(Ydf_temp)
                row_index+=w
        else:
            print('Tamaño de ventana w muy grande')
```

```
            X_df=pd.DataFrame(X_array)
            Y_df=pd.DataFrame({'Rush':Y_array})
            return X_df,Y_df
```

```
In [12]: X_df,Y_df=get_df_features_w(w_max,Data,['Speed(m/s)','Total_Acceleration(m/s^2)',
                                                 'Lateral_Acceleration(m/s^2)'
                                                ,'Engine RPM','Pitch','Driver_rush(0-10)'])

         Y_array=np.array([int(Y_df.values[i]) for i in range(Y_df.values.shape[0])])
         print(X_df.shape)
         print(Y_df.shape)
```

```
(349, 35)
(349, 1)
```

Cómo se puede observar al elegir el tamaño de ventana máximo con $ w=2000 $ (de aproximada mente 20 s), se obtienen 3855 ventanas. Además se extrajo los features de cada serie temporal (cada columna). Se extrajeron 7 features y existían 5 columnas, por lo que se obtuvieron 35 columnas en el *X_df*

```
In [13]: X_df,Y_df=get_df_features_w(2000,Data,['Speed(m/s)','Total_Acceleration(m/s^2)',
                                                'Lateral_Acceleration(m/s^2)'
                                                ,'Engine RPM','Pitch','Driver_rush(0-10)'])
         Y_array=np.array([int(Y_df.values[i]) for i in range(Y_df.values.shape[0])])
         print(X_df.shape)
         print(Y_df.shape)
```

```
(3855, 35)
(3855, 1)
```

1.5 5. Selección de parámetros

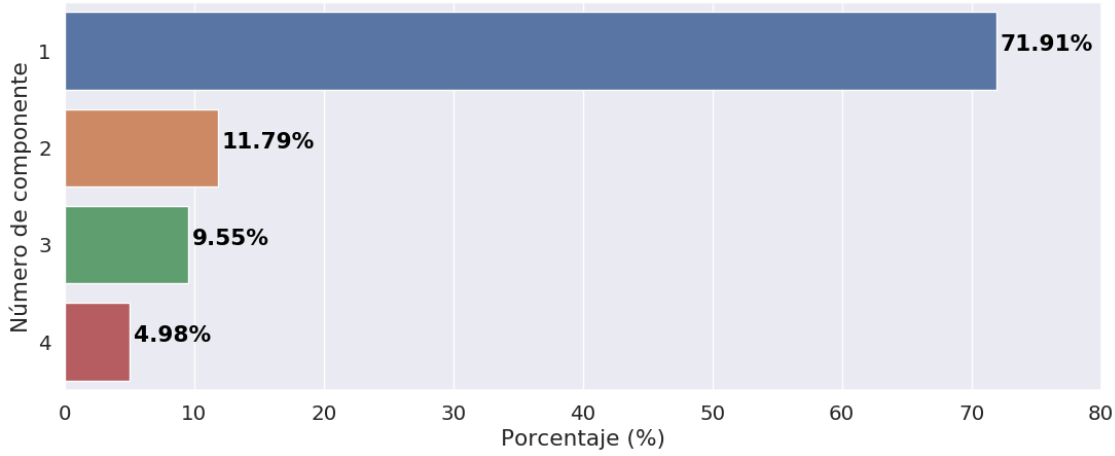
Ahora seleccionaremos sólo los parámetros que sean relevantes para la clasificación. Para eso utilizaremos el algoritmo PCA para reucir a 4 componentes.

1.6 PCA

```
In [14]: from sklearn.decomposition import PCA
         n_comp=4
         PCA_obj = PCA(n_components = n_comp)
         X_reduct = PCA_obj.fit_transform(X_df)
         print(X_reduct.shape)
         sns.set(font_scale=1.3)

         variance=PCA_obj.explained_variance_ratio_
         variance=[variance[i]*100 for i in range(variance.shape[0])]
```

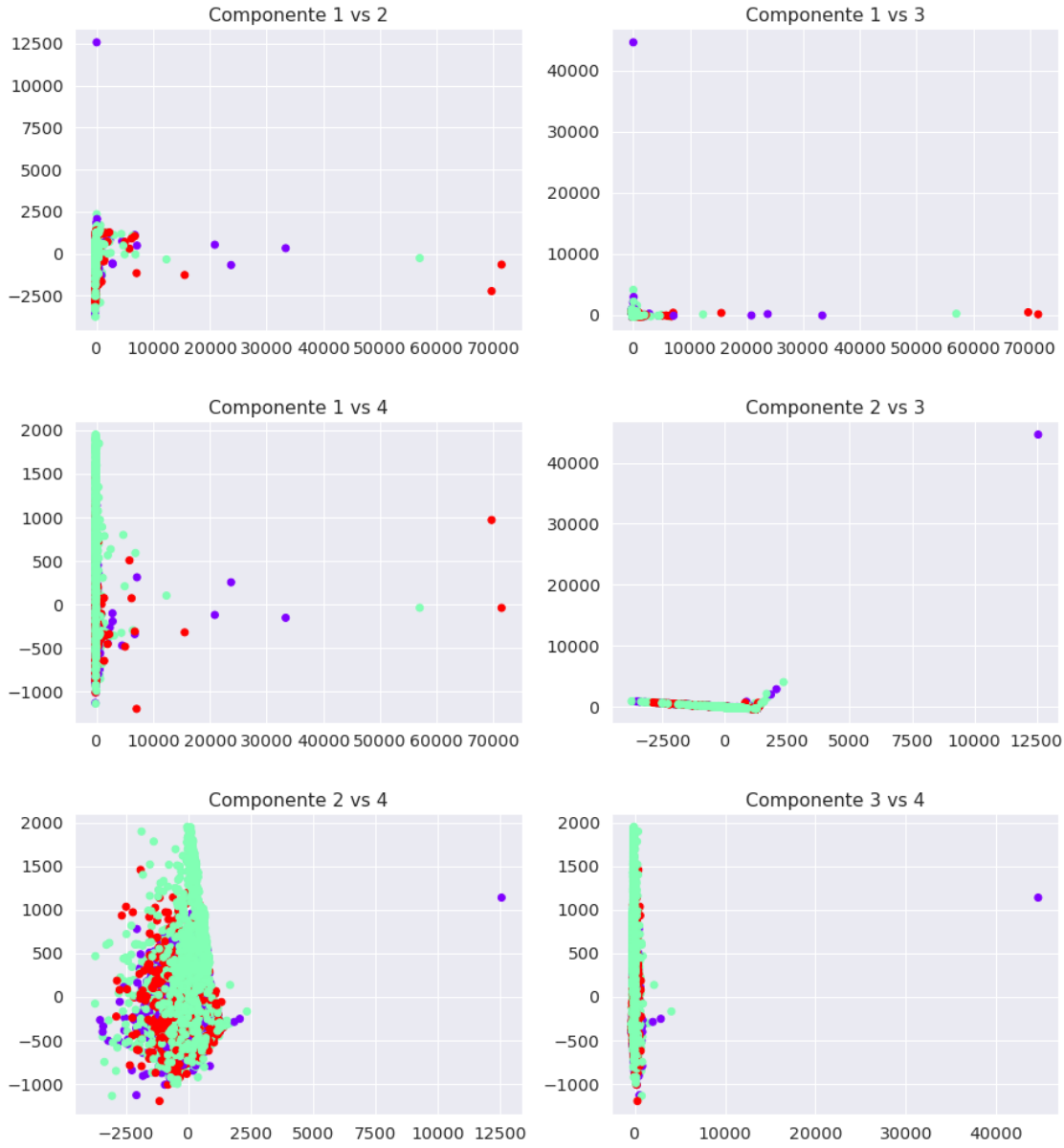
```
fig1= plt.figure(figsize=(13,5), dpi=100)
graph=sns.barplot(variance,list(range(1,n_comp+1)), orient='h')
plt.ylabel('Número de componente')
plt.xlabel('Porcentaje (%)')
plt.xlim([0, 80])

for i, v in enumerate(variance):
    graph.text(v +0.3, i , str(round(v,2))+'%', color='black', fontweight='bold')

fig1.savefig("./Figuras/PCA_dist.pdf", bbox_inches='tight')
```

```
(3855, 4)
```



In [15]:
```
fig, axes = plt.subplots(3, 2,figsize=(15,17))
axes[0, 0].scatter(X_reduct[:,0], X_reduct[:,1], c=Y_array, cmap=plt.cm.rainbow)
axes[0, 0].set_title('Componente 1 vs 2')
# plt.ylabel('Componente 2')
# plt.xlabel('Componente 1')

axes[0, 1].scatter(X_reduct[:,0], X_reduct[:,2], c=Y_array, cmap=plt.cm.rainbow)
axes[0, 1].set_title('Componente 1 vs 3')
# plt.ylabel('Componente 3')
# plt.xlabel('Componente 1')

axes[1, 0].scatter(X_reduct[:,0], X_reduct[:,3], c=Y_array, cmap=plt.cm.rainbow)
axes[1, 0].set_title('Componente 1 vs 4')
# plt.ylabel('Componente 4')
# plt.xlabel('Componente 1')

axes[1, 1].scatter(X_reduct[:,1], X_reduct[:,2], c=Y_array, cmap=plt.cm.rainbow)
```

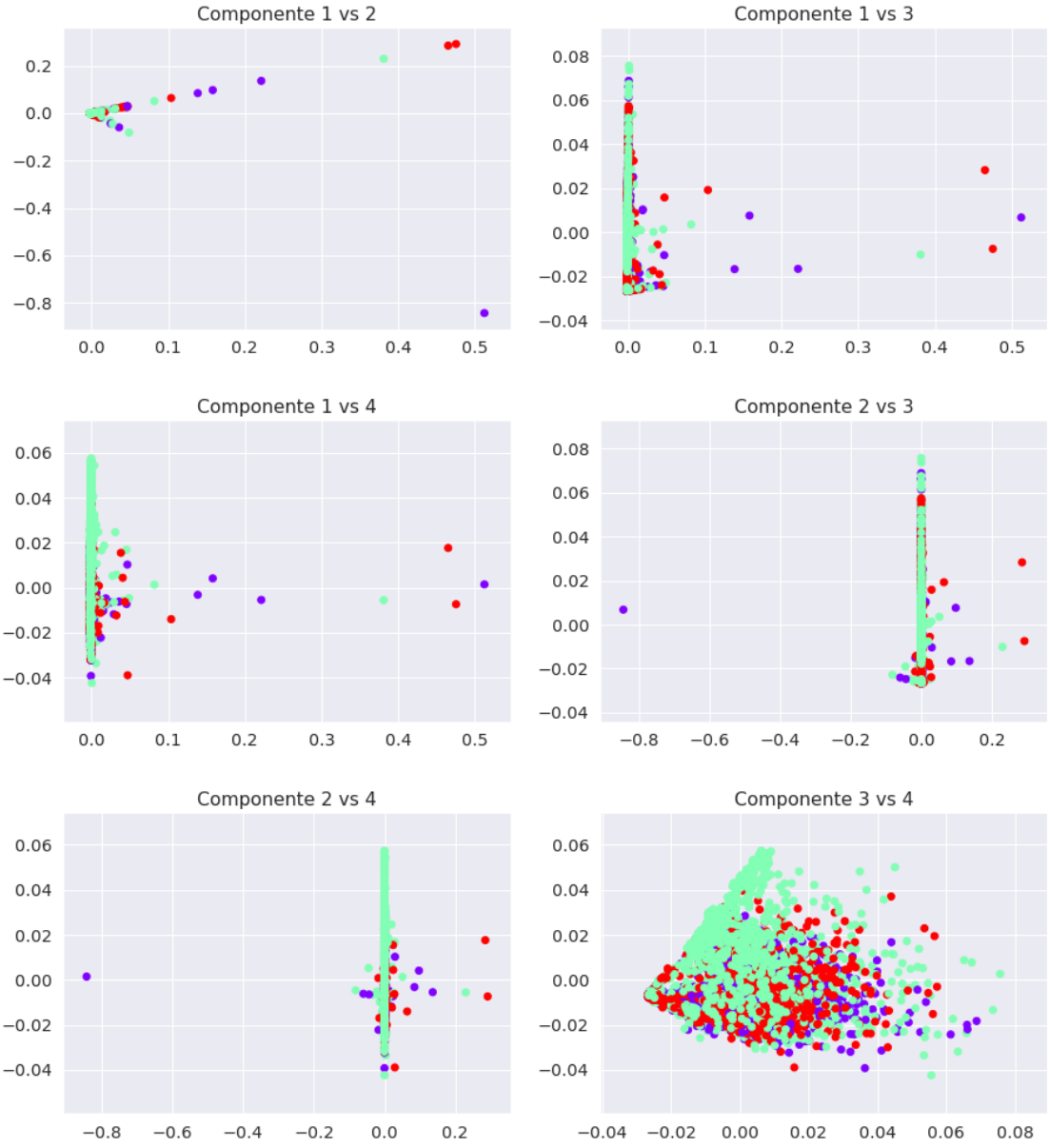
```
axes[1, 1].set_title('Componente 2 vs 3')
# plt.ylabel('Componente 3')
# plt.xlabel('Componente 2')

axes[2, 0].scatter(X_reduct[:,1], X_reduct[:,3], c=Y_array, cmap=plt.cm.rainbow)
axes[2, 0].set_title('Componente 2 vs 4')
# plt.ylabel('Componente 4')
# plt.xlabel('Componente 2')

axes[2, 1].scatter(X_reduct[:,2], X_reduct[:,3], c=Y_array, cmap=plt.cm.rainbow)
axes[2, 1].set_title('Componente 3 vs 4')
# plt.ylabel('Componente 4')
# plt.xlabel('Componente 3')

plt.subplots_adjust(hspace=0.3)

fig.savefig("./Figuras/PCA_vs.pdf", bbox_inches='tight')
```



Se usa FastICA para intentar separar loc componentes seleccionados

```
In [16]: from sklearn.decomposition import FastICA
         n_comp=4
         FastICA_tr = FastICA(n_components=n_comp, random_state=0, algorithm='deflation')
         X_reduct3 = FastICA_tr.fit_transform(X_reduct)
         X_reduct3.shape

Out[16]: (3855, 4)

In [17]: fig, axes = plt.subplots(3, 2,figsize=(15,17))
         axes[0, 0].scatter(X_reduct3[:,0], X_reduct3[:,1], c=Y_array, cmap=plt.cm.rainbow)
```

```
axes[0, 0].set_title('Componente 1 vs 2')
# plt.ylabel('Componente 2')
# plt.xlabel('Componente 1')

axes[0, 1].scatter(X_reduct3[:,0], X_reduct3[:,2], c=Y_array, cmap=plt.cm.rainbow)
axes[0, 1].set_title('Componente 1 vs 3')
# plt.ylabel('Componente 3')
# plt.xlabel('Componente 1')

axes[1, 0].scatter(X_reduct3[:,0], X_reduct3[:,3], c=Y_array, cmap=plt.cm.rainbow)
axes[1, 0].set_title('Componente 1 vs 4')
# plt.ylabel('Componente 4')
# plt.xlabel('Componente 1')

axes[1, 1].scatter(X_reduct3[:,1], X_reduct3[:,2], c=Y_array, cmap=plt.cm.rainbow)
axes[1, 1].set_title('Componente 2 vs 3')
# plt.ylabel('Componente 3')
# plt.xlabel('Componente 2')

axes[2, 0].scatter(X_reduct3[:,1], X_reduct3[:,3], c=Y_array, cmap=plt.cm.rainbow)
axes[2, 0].set_title('Componente 2 vs 4')
# plt.ylabel('Componente 4')
# plt.xlabel('Componente 2')

axes[2, 1].scatter(X_reduct3[:,2], X_reduct3[:,3], c=Y_array, cmap=plt.cm.rainbow)
axes[2, 1].set_title('Componente 3 vs 4')
# plt.ylabel('Componente 4')
# plt.xlabel('Componente 3')

plt.subplots_adjust(hspace=0.3)

fig.savefig("./Figuras/PCA_vs_separados.pdf", bbox_inches='tight')
```



1.7 Model training

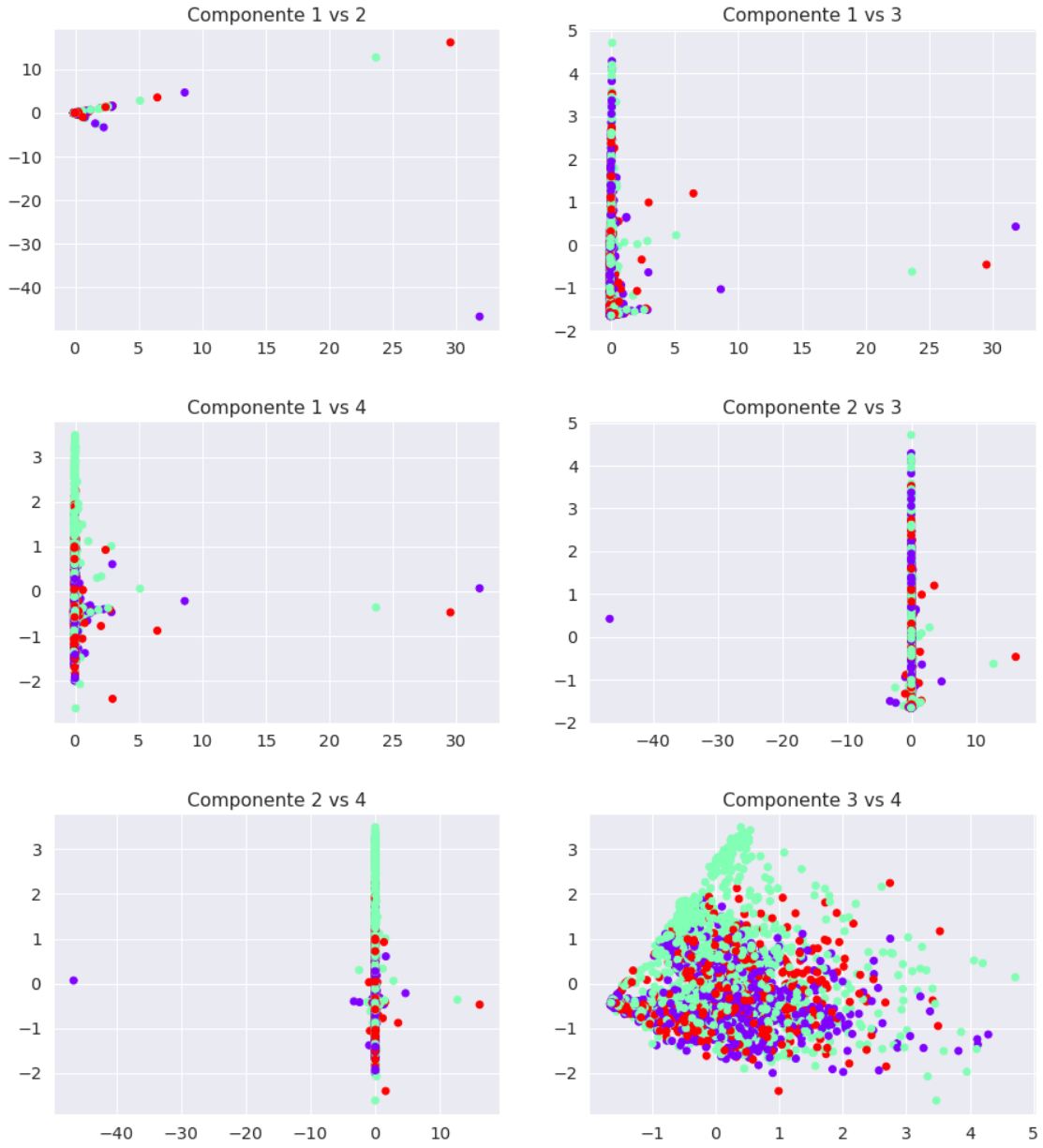
1.7.1 Splitting into train an test datasets

```
In [18]: X_data=X_reduct3
         test_size=0.3

         from sklearn.model_selection import train_test_split
         X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_array,
                         test_size=test_size, random_state=45, shuffle=True)
         X_train.shape
```

```
Out[18]: (2698, 4)
```

Se necesita también normalizar los datos

```
In [19]: from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         X_train=scaler.fit_transform(X_train)
         X_test=scaler.transform(X_test)

In [20]: X_train.shape

Out[20]: (2698, 4)

In [21]: fig, axes = plt.subplots(3, 2,figsize=(15,17))
         axes[0, 0].scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap=plt.cm.rainbow)
         axes[0, 0].set_title('Componente 1 vs 2')
         # plt.ylabel('Componente 2')
         # plt.xlabel('Componente 1')

         axes[0, 1].scatter(X_train[:,0], X_train[:,2], c=Y_train, cmap=plt.cm.rainbow)
         axes[0, 1].set_title('Componente 1 vs 3')
         # plt.ylabel('Componente 3')
         # plt.xlabel('Componente 1')

         axes[1, 0].scatter(X_train[:,0], X_train[:,3], c=Y_train, cmap=plt.cm.rainbow)
         axes[1, 0].set_title('Componente 1 vs 4')
         # plt.ylabel('Componente 4')
         # plt.xlabel('Componente 1')

         axes[1, 1].scatter(X_train[:,1], X_train[:,2], c=Y_train, cmap=plt.cm.rainbow)
         axes[1, 1].set_title('Componente 2 vs 3')
         # plt.ylabel('Componente 3')
         # plt.xlabel('Componente 2')

         axes[2, 0].scatter(X_train[:,1], X_train[:,3], c=Y_train, cmap=plt.cm.rainbow)
         axes[2, 0].set_title('Componente 2 vs 4')
         # plt.ylabel('Componente 4')
         # plt.xlabel('Componente 2')

         axes[2, 1].scatter(X_train[:,2], X_train[:,3], c=Y_train, cmap=plt.cm.rainbow)
         axes[2, 1].set_title('Componente 3 vs 4')
         # plt.ylabel('Componente 4')
         # plt.xlabel('Componente 3')

         plt.subplots_adjust(hspace=0.3)

         fig.savefig("./Figuras/PCA_vs_normalizado.pdf", bbox_inches='tight')
```



1.7.2 SVM

```
In [22]: from sklearn import svm
         from sklearn.metrics import classification_report
         SVM_clf=svm.SVC(gamma= 'auto')
         SVM_clf.fit(X_train,Y_train)
         Y_predicted=SVM_clf.predict(X_test)
         print(set(Y_predicted))
```

```
        target_names=['Tranquilo','Normal','Agresivo']
        print(classification_report(Y_test, Y_predicted,target_names=target_names))
```

```
{1}
             precision    recall  f1-score   support

  Tranquilo       0.00      0.00      0.00       313
     Normal       0.50      1.00      0.67       579
   Agresivo       0.00      0.00      0.00       265

avg / total       0.25      0.50      0.33      1157
```

1.7.3 Random Forest

```
In [23]: from sklearn.ensemble import RandomForestClassifier
         RF_clf = RandomForestClassifier(n_estimators=50, max_depth=50, random_state=0)
         RF_clf.fit(X_train,Y_train)
         Y_predicted=RF_clf.predict(X_test)
         print(set(Y_predicted))

         target_names=['Tranquilo','Normal','Agresivo']
         print(classification_report(Y_test, Y_predicted,target_names=target_names))
```

```
{0, 1, 2}
             precision    recall  f1-score   support

  Tranquilo       0.33      0.34      0.33       313
     Normal       0.54      0.64      0.58       579
   Agresivo       0.25      0.14      0.18       265

avg / total       0.41      0.44      0.42      1157
```

1.7.4 Neural Network

```
In [24]: # activation : {'identity', 'logistic', 'tanh', 'relu'}
         # solver : {'lbfgs', 'sgd', 'adam'}

         from sklearn.neural_network import MLPClassifier
         MLP_clf = MLPClassifier(solver='lbfgs', alpha=1e-5, activation='identity',
                                 hidden_layer_sizes=(15,15,15), random_state=1)
         MLP_clf.fit(X_train,Y_train)
         Y_predicted=MLP_clf.predict(X_test)
         print(set(Y_predicted))
```

```
        target_names=['Tranquilo','Normal','Agresivo']
        print(classification_report(Y_test, Y_predicted,target_names=target_names))
```

```
{0, 1, 2}
             precision    recall  f1-score   support

  Tranquilo       0.30      0.19      0.24       313
     Normal       0.52      0.85      0.65       579
   Agresivo       1.00      0.00      0.01       265

avg / total       0.57      0.48      0.39      1157
```