Anexos

Anexos A

Algoritmos

1 *PROCESADO DE CAÍDAS*

A continuación se muestra el algoritmo realizado para el procesado de caídas

1.1 Importación librerías

[1]:
```
"""
@author: Rebeca Teresa Blanco
"""
import csv
import plotly.graph_objects as go
from plotly.offline import plot
from plotly.subplots import make_subplots
from Lector import Lector, LoadTests
from Statistics import Statistics
import numpy as np
```

1.2 Definición de funciones

A continuación, se muestran las funcones utilizadas en el programa principal:

1.2.1 Sincronización de datos

Sincroniza los datos de ambos pies. Devuelve las señales de los datos en cruodo y sus derivadas

[2]:
```
from utils import sincronizaDatos
```

1.2.2 Cálculo de ventana deslizante

Muestreando y comparando la seañal de la derivada de la aceleración con un valor umbral, detectamos los tiempos de inicio, fin y estabilización de una caída, tiempos con los qeu definiremos la ventana deslizante. Devuelve el valor de los tiempos inicio, fin y estabilización de una caída.

[3]:
```
def calculateWindow(lector, time, ini, fin):
    a = lector.acc_total_d_v[ini:fin]

    if lector.session_type == 'LATERAL_WALKING':
        # Nos saltamos el intervalo donda camina para detectar la caída
        comienzo = 4.7
    else:
        comienzo = 2

    inicio = True
    fin = False
    cont = 0
    x3 = time[len(time) - 1]
```

```
    x1 = x3
    x0 = 0
    for i in range(len(a)):
        if inicio == True and time[i] > comienzo:
            if a[i] > 0.23 or a[i] < -0.23: #Umbral comienczo caida
                x0 = time[i]
                inicio = False
        else:
            if a[i] > 0.23 or a[i] < -0.23:
                x1 = time[i]

    for i in range(len(a)):
        if time[i] >= x1 and time[i] <= x1 + 1:
            if a[i] < 0.011:
                cont = cont + 1
                if cont == 3:
                    x3 = time[i]
            else:
                cont = 0
    return x0, x1, x3
```

1.2.3 Cálculo indicadores

Calcula los indicadores de un vector de datos. Devuelve valores de los mismos (kurtosis, max,min, med, desv, var y potencia de señal)

[4]:
```
from utils import estadistica
```

1.2.4 Cálculo estadística

Devuelve la estadística de un vector (maximo, minimo, media, desviacion estandar y varianza)

[5]:
```
from utils import statWindow
```

1.2.5 Procesado caída

Procesa un ensayo de caida. Calcula y devuelve el valor de los datos en crudo e indicadores de las variables de una caída.

[6]:
```
from utils import ProcessFall
```

1.2.6 Obtención datos caídas

Abre, procesa (mediante la función anterior) y concatena los datos en crudo e indicadores de las variables para cada caída. Devuelve los datos en crudo y los indicadores de las variables de todas las caídas.

[7]:
```
from utils import getDatosFall
```

1.2.7 Obtención de gráficas caídas

Crea las figuras con las graficas de las variables de cada caída, indicando en cada una la franja en la que se ah producido la caída. Devuelve un array con todas las gráficas.

[8]:
```
from utils import graficas
```

[9]:
```
def getFigures(Falls_d,Falls_i):

    # LOADERS (OBJECTS)
    loader_d = LoadTests()
    loader_i = LoadTests()

    # LOAD FILES
    loader_d.Load(Falls_d)
    loader_i.Load(Falls_i)

    # .CSV DATA PATH
    data_d = 'data_d/'
    data_i = 'data_i/'

    # VECTORS' INICIALIZATION

    PTI_d = []
    PTI_i = []

    time_d = []
    time_i = []

    TW_d = []
    TD_d = []
    TW_i = []
    TD_i = []

    TW = 3150  # ms
    TD = 500  # ms
    T = 20  # ms
    step_size = TD // T
    window_size = TW // T

    n_d = []
    n_i = []
    n_a_v_d = []
    n_g_v_d = []
    n_p_v_d = []
```

```
n_roll_v_d = []
n_pitch_v_d = []
n_yaw_v_d = []
n_a_d = []
n_g_d = []
n_p_d = []
n_roll_d = []
n_pitch_d = []
n_yaw_d = []

k_a = []
max_a =[]
min_a = []
med_a = []
desv_a = []
var_a = []
pow_a = []

k_p = []
max_p =[]
min_p = []
med_p = []
desv_p = []
var_p = []
pow_p = []

k_w = []
max_w =[]
min_w = []
med_w = []
desv_w = []
var_w = []
pow_w = []

k_roll = []
max_roll =[]
min_roll = []
med_roll = []
desv_roll = []
var_roll = []
pow_roll = []

k_pitch = []
max_pitch =[]
min_pitch = []
med_pitch = []
desv_pitch = []
```

```
    var_pitch = []
    pow_pitch = []

    k_yaw = []
    max_yaw =[]
    min_yaw = []
    med_yaw = []
    desv_yaw = []
    var_yaw = []
    pow_yaw = []

    figures_d = []
    figures_i = []

    for i in range(len(loader_d.files)):
        t = []
        # INICIALIZATION
        lector_d = Lector()
        lector_i = Lector()

        # OPEN AND READ .CSV DATA FILES
        lector_d.lee(data_d + loader_d.files[i])
        lector_i.lee(data_i + loader_i.files[i])

        # DATA NORMALIZATION
        # lector_d.normalize()
        # lector_i.normalize()

        # CHANGE REF
        lector_d.RightRef()
        lector_i.LeftRef()

        # DATA SYNCRONIZE

        if lector_d.timestamp_v[0] < lector_i.timestamp_v[0]:
            ini = lector_d.timestamp_v[0]
        else:
            ini = lector_i.timestamp_v[0]

        for j in range(len(lector_d.timestamp_v)):
            time_d.append((lector_d.timestamp_v[j] - ini) / 1000)

        for j in range(len(lector_i.timestamp_v)):
            time_i.append((lector_i.timestamp_v[j] - ini) / 1000)

        ini_d = 0
        ini_i = 0
```

```
        fin_d = len(lector_d.timestamp_v) - 1
        fin_i = len(lector_i.timestamp_v) - 1

        if time_d[0] < time_i[0]:
            while time_d[ini_d] < time_i[0]:
                ini_d += 1
        else:
            while time_i[ini_i] < time_d[0]:
                ini_i += 1

        if time_d[fin_d] > time_i[fin_i]:
            while time_d[fin_d] > time_i[fin_i]:
                fin_d -= 1
        else:
            while time_i[fin_i] > time_d[fin_d]:
                fin_i -= 1

        # WINDOWS SIZE SYNC
        time_d = time_d[ini_d:fin_d]
        time_i = time_i[ini_i:fin_i]

        # TW y TD
        h_band_d = calculateWindow(lector_d, time_d, ini_d, fin_d)
        h_band_i = calculateWindow(lector_i, time_i, ini_i, fin_i)


        # PLOTS
        title = loader_d.Names[i] + ', Age:' + loader_d.Ages[i] + ', Weight:' +␣
↪loader_d.Weights[i] + \
                ', Height:' + loader_d.Heights[
                    i] + ', Activity:' + lector_d.session_type

        #Procesado de la caida
        datos_caida_d = ProcessFall(lector_d,time_d, ini_d, fin_d, step_size,␣
↪window_size, h_band_d[0], h_band_d[1])
        datos_caida_i = ProcessFall(lector_i,time_i, ini_i, fin_i, step_size,␣
↪window_size, h_band_i[0], h_band_i[1])

        #Vectores de indicadores para estadística de caída
        k_a.append(datos_caida_d[0][0][0])
        max_a.append(datos_caida_d[0][0][1])
        min_a.append(datos_caida_d[0][0][2])
        med_a.append(datos_caida_d[0][0][3])
        desv_a.append(datos_caida_d[0][0][4])
        var_a.append(datos_caida_d[0][0][5])
        pow_a.append(datos_caida_d[0][0][6])
```

```
        k_p.append(datos_caida_d[0][1][0])
        max_p.append(datos_caida_d[0][1][1])
        min_p.append(datos_caida_d[0][1][2])
        med_p.append(datos_caida_d[0][1][3])
        desv_p.append(datos_caida_d[0][1][4])
        var_p.append(datos_caida_d[0][1][5])
        pow_p.append(datos_caida_d[0][1][6])

        k_w.append(datos_caida_d[0][2][0])
        max_w.append(datos_caida_d[0][2][1])
        min_w.append(datos_caida_d[0][2][2])
        med_w.append(datos_caida_d[0][2][3])
        desv_w.append(datos_caida_d[0][2][4])
        var_w.append(datos_caida_d[0][2][5])
        pow_w.append(datos_caida_d[0][2][6])

        k_roll.append(datos_caida_d[0][3][0])
        max_roll.append(datos_caida_d[0][3][1])
        min_roll.append(datos_caida_d[0][3][2])
        med_roll.append(datos_caida_d[0][3][3])
        desv_roll.append(datos_caida_d[0][3][4])
        var_roll.append(datos_caida_d[0][3][5])
        pow_roll.append(datos_caida_d[0][3][6])

        k_pitch.append(datos_caida_d[0][4][0])
        max_pitch.append(datos_caida_d[0][4][1])
        min_pitch.append(datos_caida_d[0][4][2])
        med_pitch.append(datos_caida_d[0][4][3])
        desv_pitch.append(datos_caida_d[0][4][4])
        var_pitch.append(datos_caida_d[0][4][5])
        pow_pitch.append(datos_caida_d[0][4][6])

        k_yaw.append(datos_caida_d[0][5][0])
        max_yaw.append(datos_caida_d[0][5][1])
        min_yaw.append(datos_caida_d[0][5][2])
        med_yaw.append(datos_caida_d[0][5][3])
        desv_yaw.append(datos_caida_d[0][5][4])
        var_yaw.append(datos_caida_d[0][5][5])
        pow_yaw.append(datos_caida_d[0][5][6])


        #Ventanas con indicadores
        ventanas_caida_d = datos_caida_d[1]
        ventanas_caida_i = datos_caida_i[1]

        #Gráficas
```

```
        figures_d.append(graficas(lector_d, ventanas_caida_d, time_d, ini_d,␣
↪fin_d, h_band_d, window_size, step_size, title))
        figures_i.append(graficas(lector_i, ventanas_caida_i, time_i, ini_i,␣
↪fin_i, h_band_i, window_size, step_size, title))

        time_d.clear()
        time_i.clear()

    return figures_d, figures_i
```

1.3 Programa principal

1.3.1 Obtención gráficas caídas

[10]:
```
#Fall paths
Backwards_d = 'Tests/Falls/Backwards_d.csv'
Backwards_i = 'Tests/Falls/Backwards_i.csv'
Forwards_d = 'Tests/Falls/Forwards_d.csv'
Forwards_i = 'Tests/Falls/Forwards_i.csv'
LateralStand_d = 'Tests/Falls/LateralStand_d.csv'
LateralStand_i = 'Tests/Falls/LateralStand_i.csv'
LateralWalk_d = 'Tests/Falls/LateralWalk_d.csv'
LateralWalk_i = 'Tests/Falls/LateralWalk_i.csv'
Sitting_d = 'Tests/Falls/Sitting_d.csv'
Sitting_i = 'Tests/Falls/Sitting_i.csv'
Falls_d = 'Tests/Falls/Falls_d.csv'
Falls_i = 'Tests/Falls/Falls_i.csv'
```

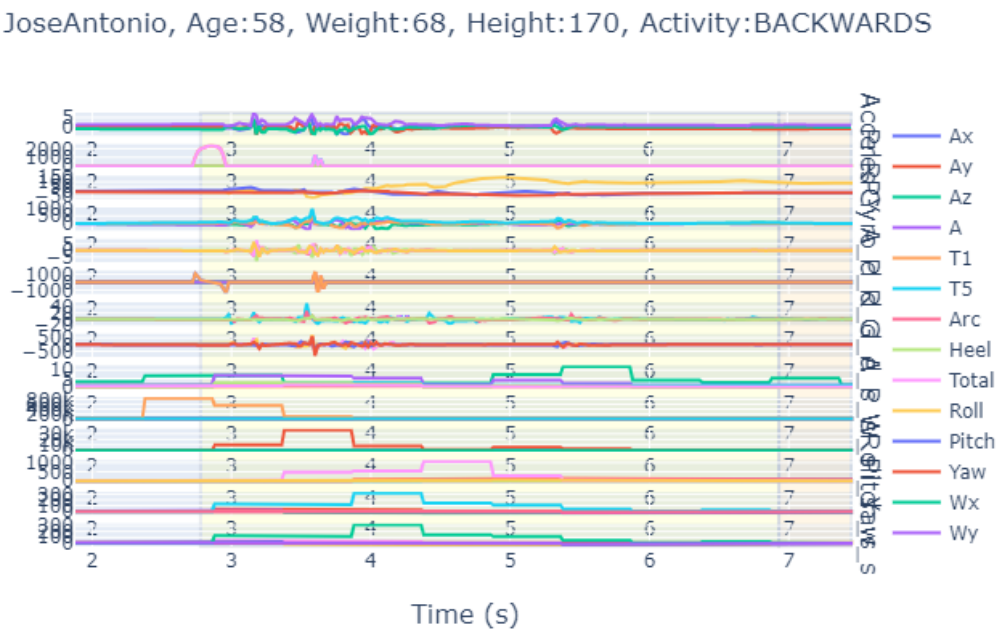
[11]:
```
fig_d_back, fig_i_back = getFigures(Backwards_d,Backwards_i)
fig_d_for, fig_i_for = getFigures(Forwards_d,Forwards_i)
fig_d_LS, fig_i_LS = getFigures(LateralStand_d,LateralStand_i)
fig_d_Lw, fig_i_LW = getFigures(LateralWalk_d,LateralWalk_i)
fig_d_sit, fig_i_sit = getFigures(Sitting_d,Sitting_i)
```

1.3.2 Visualización gráficas caídas

NOTA: Solo se muestra una gráfica por tipo de caida

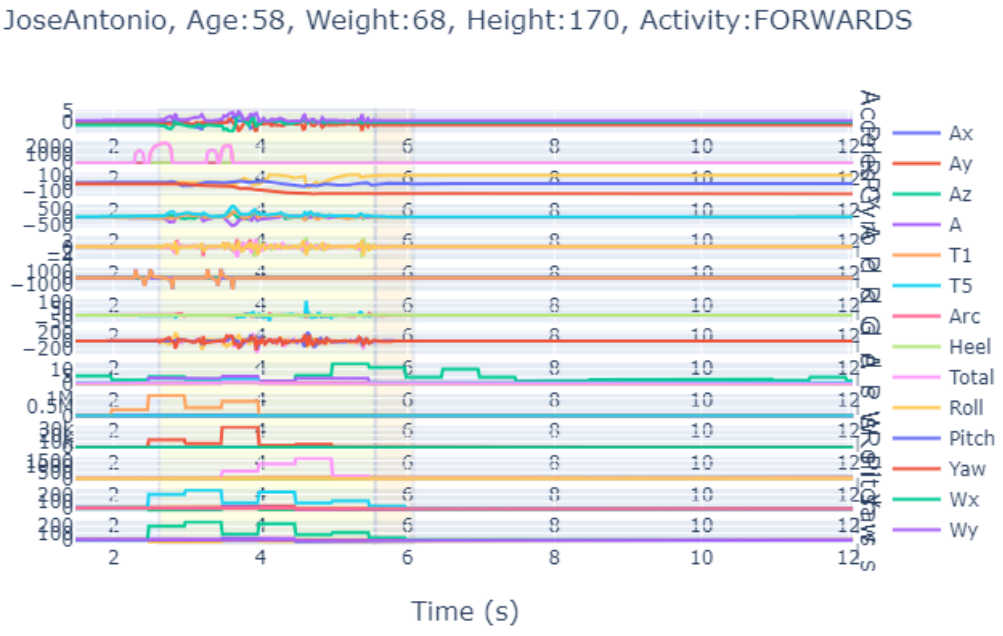
Caída hacia atrás

[12]:
```
fig_d_back[2].show()
```



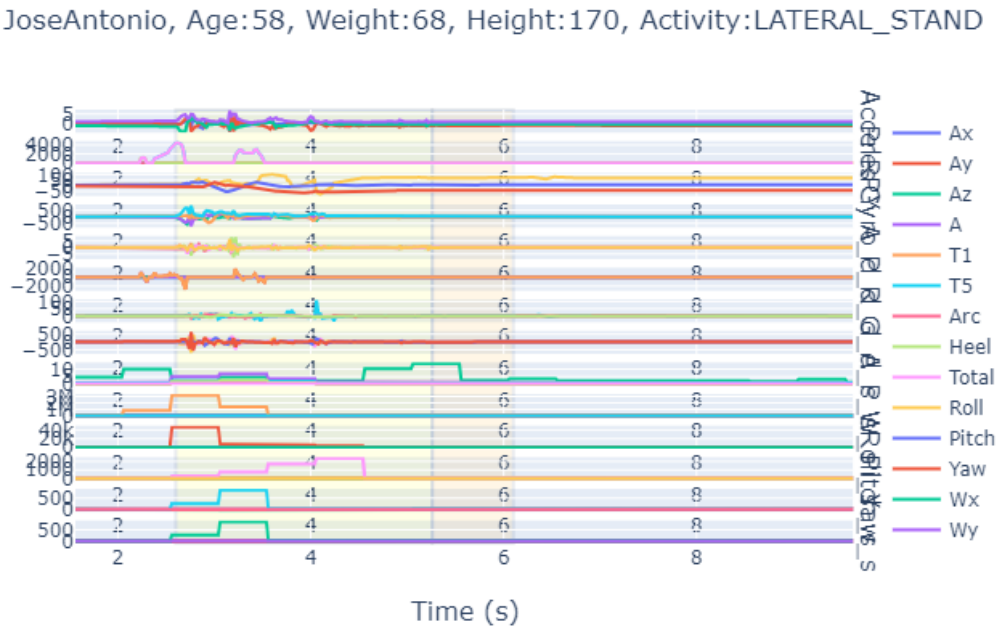
Caída hacia delante

[13]:
```
fig_d_for[10].show()
```



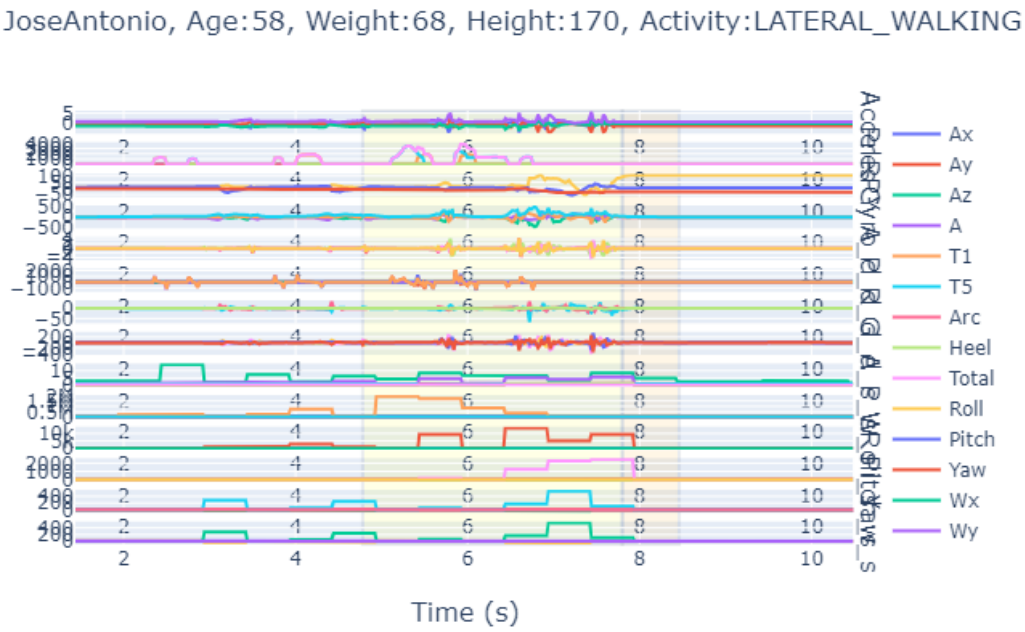
Caída hacia un lado estando de pie

[14]:
```
fig_d_LS[15].show()
```



Caída hacia un lado caminando

[15]:
```
fig_d_Lw[20].show()
```



Caída sentándose

[16]:
```
fig_d_sit[25].show()
```

1 *ALGORITMO MACHINE LEARNING*

1.1 Carga de librerias

[1]:
```
from Lector import LoadTests, Lector
from Statistics import Statistics
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.externals import joblib
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.ensemble import ExtraTreesClassifier
import csv
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
```

```
D:\Anaconda\lib\site-packages\sklearn\externals\joblib\__init__.py:15:
DeprecationWarning: sklearn.externals.joblib is deprecated in 0.21 and will be
removed in 0.23. Please import this functionality directly from joblib, which
can be installed with: pip install joblib. If this warning is raised when
loading pickled models, you may need to re-serialize those models with scikit-
learn 0.21+.
  warnings.warn(msg, category=DeprecationWarning)
```

1.2 Declaramos funciones utilizadas en el programa

1.2.1 Sincronización de los datos

Sincroniza los datos de ambos pies. Devuelve las señales de los datos en cruodo y sus derivadas def sincronizaDatos(lector, ini_d, fin_d).

[2]:
```
from utils import sincronizaDatos
```

1.2.2 Cálculo indicadores

Calcula los indicadores de un vector de datos. Devuelve valores de los mismos (kurtosis, max,min, med, desv, var y potencia de señal)

[3]:
```
from utils import estadistica
```

1.2.3 Cálculo estadística

Devuelve la estadística de un vector (maximo, minimo, media, desviacion estandar y varianza)

[4]:
```
from utils import statWindow
```

1.2.4 Cálculo de ventana deslizante

Devuelve los instantes de tiempo inicio y final de una caída.

[5]:
```
from utils import calculateWindow
```

1.2.5 Procesado caída

Procesa un ensayo de caida. Calcula y devuelve el valor de los datos en crudo e indicadores de las variables de una caída.

[1]:
```
from utils import ProcessFallClasif
```

1.2.6 Obtención datos caídas

Abre, procesa (mediante la función anterior) y concatena los datos en crudo e indicadores de las variables para cada caída. Devuelve los datos en crudo y los indicadores de las variables de todas las caídas.

[7]:
```
from utils import getDatosFall
```

1.2.7 Procesado ADL

Procesa un ensayo ADL. Calcula y devuelve el valor de los datos en crudo e indicadores de las variables de una ADL.

[8]:
```
from utils import processADL
```

1.2.8 Obtención datos ADL

Abre, procesa (mediante la función anterior) y concatena los datos en crudo e indicadores de las variables para cada ADL. Devuelve los datos en crudo y los indicadores de las variables de todas las ADL.

[9]:
```
from utils import getDatosADL
```

1.2.9 Obtención frames de los datos en crudo

A partir de los datos en crudo de las variables se obtienen los frames de los mismos.

[10]:
```
from utils import getFramesRaw
```

[11]:
```
from utils import getFramesRawData
```

1.2.10 Obtención de frames de los indicadores

A partir de los indicadores de las variables se obtienen los frames de las variables.

[12]:
```
from utils import getFrameN
```

[13]:
```
from utils import getFrameNdata
```

1.2.11 Limpiar data set

Elimina valores NaN e inf de un dataset.

[14]:
```
def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).all(1)
    return df[indices_to_keep].astype(np.float64)
```

1.2.12 Creación salidas

A partir de los datos de entrada devuelve el vector de salidas objetivo.

[15]:
```
def getTarget(frame, tipo):
    target = []
    for i in range(frame.shape[0]):
        target.append(tipo)
    return target
```

1.2.13 Carga de modelos

Carga modelos entrenados a partir del path dado.

[16]:
```
def loadModels(paths):
    modelos = []
    for i in range(len(paths)):
        modelos.append(joblib.load(paths[i]))  # Carga modelo

    return modelos
```

1.2.14 Resultados

A partir de los modelos, sus nombres y, los datos de test, devuelve un vector de resultados. Cada resultado consta del nombre del modelo, el nº de TP, TN, FN, FP y la exactitud del modelo en cuestión.

[17]:
```
def writeResults(modelos, x, y, paths):
    cont1 = 0
    cont2 = 0
    cont3 = 0
    cont4 = 0
    data = []

    for i in range(len(modelos)):
        cont1 = 0
        cont2 = 0
        cont3 = 0
        cont4 = 0
        result = modelos[i].predict(x)
        for j in range(len(result)):
            if result[j] == y[j] and y[j] == 'fall':
                # True positives
                cont1 += 1
            elif result[j] == y[j] and y[j] == 'ADL':
                # True negatives
                cont2 += 1
            elif result[j] != y[j] and y[j] == 'fall':
                # False negatives
                cont3 += 1
            else:
                # False positives
                cont4 += 1

        score = modelos[i].score(x, y.ravel())
        datos = [paths[i], cont1, cont2, cont3, cont4, score]
        data.append(datos)

    return data
```

1.2.15 Feature importance (FI)

Devuelve los gráficos de caja del resultado de la función *Feature importance* (importancia de características) incrementando el nº de las mismas, tanto para datos en crudo como para indicadores.

[47]:
```
def importancesVar(x, y, n_x, n_y, frames, n_frames):
    for i in range(2, 16, 2):
        model = ExtraTreesClassifier(random_state=0)
```

```
        model.fit(x, y.ravel())
        #print(model.feature_importances_)  # use inbuilt class␣
↪feature_importances of tree based classifiers
        # plot graph of feature importances for better visualization
        plt.figure()
        plt.title("Importancia de "+str(i)+" variables")
        feat_importances = pd.Series(model.feature_importances_, index=frames.
↪columns)
        feat_importances.nlargest(i).plot(kind='barh')
        plt.show()
    for i in range(5, 100, 20):
        n_model = ExtraTreesClassifier(random_state=1)
        n_model.fit(n_x, n_y.ravel())
        #print(n_model.feature_importances_)  # use inbuilt class␣
↪feature_importances of tree based classifiers
        plt.figure()
        plt.title("Importancia de "+str(i)+" indicadores")
        n_feat_importances = pd.Series(n_model.feature_importances_,␣
↪index=n_frames.columns)
        n_feat_importances.nlargest(i).plot(kind='barh')
        plt.show()
```

1.2.16 Feature selection

Cálculo del número óptimo de características.

Devuelven lista de modelos obtenidos usando desde 1 a X indicadores, tanto para datos en crudo como para indicadores

[19]:
```
def get_modelsRAW():
    models = dict()
    for i in range(2, 16):
        rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=i)
        model = DecisionTreeClassifier()
        models[str(i)] = Pipeline(steps=[('s', rfe), ('m', model)])
    return models
```

[20]:
```
def get_modelsN():
    models = dict()
    for i in range(2, 112):
        rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=i)
        model = DecisionTreeClassifier()
        models[str(i)] = Pipeline(steps=[('s', rfe), ('m', model)])
    return models
```

Devuelve la exactitud obtenida en función del nº de características usadas

[21]:
```
def evaluate_model(model, n_features, x, y):
    cv = RepeatedStratifiedKFold(n_splits=n_features, n_repeats=3,␣
 ↪random_state=1)
    scores = cross_val_score(model, x, y, scoring='accuracy', cv=cv, n_jobs=-1,␣
 ↪error_score='raise')
    return scores
```

Devuelve la calificación para cada nº de características escogido. Finalmente, devuelve todas en un gráfico de caja.

[22]:
```
def numberFeatures(models, n_features, x, y):
    results = []
    names = []
    for name, model in models.items():
        scores = evaluate_model(model, n_features, x, y)
        results.append(scores)
        names.append(name)
        print('>%s %.3f (%.3f)' % (name, np.mean(scores), np.std(scores)))
    # Plot model performance for comparison
    plt.figure()
    plt.boxplot(results, labels=names, showmeans=True)
    plt.show()
```

1.2.17 PCA análisis

A partir de un conjunto de datos realiza el análisis pca.

[23]:
```
def PCAanalisis(x, y, title):
    #Standardizing the features
    x = StandardScaler().fit_transform(x)
    pca = PCA(n_components=2)
    principalComponents = pca.fit_transform(x)
    principalDf = pd.DataFrame(data=principalComponents, columns=['principal␣
 ↪component 1', 'principal component 2'])
    target = pd.DataFrame(y, columns=['target'])
    finalDf = pd.concat([principalDf, target], axis=1)
    plt.figure()
    plt.title(title)
    plt.xlabel('Principal component 1')
    plt.ylabel('Principal component 2')
    targets = ['fall', 'ADL']
    colors = ['tab:blue', 'tab:orange' ] #'tab:green', 'tab:red', 'tab:purple',␣
 ↪'tab:brown', 'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan'
    for targ, color in zip(targets, colors):
        indicesToKeep = finalDf['target'] == targ
        plt.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
                    , finalDf.loc[indicesToKeep, 'principal component 2']
```

```
                    , c=color, edgecolor='none', alpha=0.5
                    , s=50)
    plt.legend(targets)
    plt.grid()
```

1.2.18 Correlaciones variables

Muestra la correlación entre las variables de las caidas y ADL. Además el heat plot.

[24]:
```
def pintaHeatmap(map, title):
    plt.figure(figsize=(5, 5))
    plt.title(title)
    sns.heatmap(map, xticklabels=map.columns.values, yticklabels=map.columns.
 ↪values, annot=True)
    plt.tight_layout()
    plt.show()
```

[25]:
```
def correlations(frame_falls_d, frame_ADL_d, n_frame_falls_d, n_frame_ADL_d):
    # CORRELATIONS
    corr_falls = frame_falls_d.corr()
    corr_ADL = frame_ADL_d.corr()
    n_corr_falls = n_frame_falls_d.corr()
    n_corr_ADL = n_frame_ADL_d.corr()

    # HEAT MAPS
    pintaHeatmap(corr_falls, "Correlaciones datos raw caídas")
    pintaHeatmap(corr_ADL, "Correlaciones datos raw ADL")
    pintaHeatmap(n_corr_ADL, "Correlaciones indicadores ADL")
    pintaHeatmap(n_corr_falls, "Correlaciones indicadores caídas ")

    # PAIR PLOTS
    sns.pairplot(frame_falls_d)
    sns.pairplot(frame_ADL_d)
```

1.2.19 Recursive Feature Elimination (RFE)

Se obtiene el nº óptimo de características y, mediante árboles de decisión se conoce la identidad de las mismas.

[26]:
```
def getRFE_raw(x, y):
    models = get_modelsRAW()
    # Visualización del nº de características óptimo
    numberFeatures(models, 16, x, y.ravel())
    # En base al resultado de las funciones anteriores se definen el nº de␣
 ↪características óptimo
    feat_raw = 12
```

```
    # Define RFE --> Árbol de decisión para escoger qué características son las
↪óptimas.
    rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=feat_raw)
    # Fit RFE
    rfe.fit(x, y.ravel())
    # Summarize all features
    for i in range(x.shape[1]):
        print('Column: %d, Selected %s, Rank: %.3f' % (i, rfe.support_[i], rfe.
↪ranking_[i]))
```

[27]:
```
def getRFE_n(n_x, n_y):
    n_models = get_modelsN()
    # Visualización del nº de características óptimo
    numberFeatures(n_models, 112 , n_x, n_y.ravel())
    # En base al resultado de las funciones anteriores se definen el nº de
↪características óptimo
    feat_n = 6
    # Define RFE --> Árbol de decisión para escoger qué características son las
↪óptimas.
    n_rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=feat_n)
    # Fit RFE
    n_rfe.fit(n_x, n_y.ravel())
    # Summarize all features
    for i in range(n_x.shape[1]):
        print('Column: %d, Selected %s, Rank: %.3f' % (i, n_rfe.support_[i],
↪n_rfe.ranking_[i]))
```

1.2.20 Feature importance

Obtención del nº de características óptimo para ambos tipos de modelos (datos en crudo e indicadores).

[43]:
```
def getFI(x,y,n_x,n_y,frames, n_frames):
    model = ExtraTreesClassifier(random_state=0)
    n_model = ExtraTreesClassifier(random_state=1)
    model.fit(x, y.ravel())
    n_model.fit(n_x,n_y.ravel())
    #print(model.feature_importances_)  # use inbuilt class feature_importances
↪of tree based classifiers
    #print(n_model.feature_importances_)  # use inbuilt class
↪feature_importances of tree based classifiers
    # plot graph of feature importances for better visualization
    plt.figure()
    plt.title("Importacia variables")
    feat_importances = pd.Series(model.feature_importances_, index=frames.
↪columns)
```

```
    feat_importances.nlargest(12).plot(kind='barh')
    plt.show()
    plt.figure()
    plt.title("Importacia indicadores")
    n_feat_importances = pd.Series(n_model.feature_importances_, index=n_frames.
↪columns)
    n_feat_importances.nlargest(6).plot(kind='barh')
    plt.show()

    # Descomentar para obtener los graficos de importancia de características
    importancesVar(x,y,n_x,n_y,frames,n_frames)
```

1.2.21 Obtención del data set

Obtención del conjunto de datos de entrenamiento y test. Devuelve los conjuntos de entrenamiento de test para datos raw, indicadores y pca de ambos.

[29]:
```
def getDataSets():
    #Fall paths
    Backwards_d = 'Tests/Falls/Backwards_d.csv'
    Backwards_i = 'Tests/Falls/Backwards_i.csv'
    Forwards_d = 'Tests/Falls/Forwards_d.csv'
    Forwards_i = 'Tests/Falls/Forwards_i.csv'
    LateralStand_d = 'Tests/Falls/LateralStand_d.csv'
    LateralStand_i = 'Tests/Falls/LateralStand_i.csv'
    LateralWalk_d = 'Tests/Falls/LateralWalk_d.csv'
    LateralWalk_i = 'Tests/Falls/LateralWalk_i.csv'
    Sitting_d = 'Tests/Falls/Sitting_d.csv'
    Sitting_i = 'Tests/Falls/Sitting_i.csv'
    Falls_d = 'Tests/Falls/Falls_d.csv'
    Falls_i = 'Tests/Falls/Falls_i.csv'

    #ADL Paths
    Lay_d = 'Tests/ADL/Lay/Lay_d.csv'
    Lay_i = 'Tests/ADL/Lay/Lay_i.csv'
    LayFP_d = 'Tests/ADL/LayFP/LayFP_d.csv'
    LayFP_i = 'Tests/ADL/LayFP/LayFP_i.csv'
    SitFP_d = 'Tests/ADL/SittingFP/SittingFP_d.csv'
    SitFP_i = 'Tests/ADL/SittingFP/SittingFP_i.csv'
    Walk_d = 'Tests/ADL/Walk/Walk_d.csv'
    Walk_i = 'Tests/ADL/Walk/Walk_i.csv'
    WalkFP_d = 'Tests/ADL/WalkFP/WalkFP_d.csv'
    WalkFP_i = 'Tests/ADL/WalkFP/WalkFP_i.csv'
    Stand_d = 'Tests/ADL/Stand/Stand_d.csv'
    Stand_i = 'Tests/ADL/Stand/Stand_i.csv'
    ADL_d = 'Tests/ADL/ADL_d.csv'
    ADL_i = 'Tests/ADL/ADL_i.csv'
```

```
    # Windows times caracteristics
    TW = 3150  # ms
    TD = 500  # ms
    T = 20  # ms
    step_size = TD // T
    window_size = TW // T

    n_frame_back_d, n_frame_for_d, n_frame_LS_d, n_frame_LW_d, n_frame_sit_d,␣
↪n_frame_falls_d, \
    n_frame_lay_d, n_frame_layFP_d, n_frame_sitFP_d, n_frame_walk_d,␣
↪n_frame_walkFP_d, n_frame_ADL_d = getFramesNdata(
        Backwards_d, Backwards_i, Forwards_d, Forwards_i, LateralStand_d,␣
↪LateralStand_i,
        LateralWalk_d, LateralWalk_i, Sitting_d, Sitting_i, Falls_d, Falls_i,␣
↪Lay_d, Lay_i,
        LayFP_d, LayFP_i, SitFP_d, SitFP_i, Walk_d, Walk_i, WalkFP_d, WalkFP_i,␣
↪ADL_d, ADL_i, window_size, step_size)
    frame_back_d, frame_for_d, frame_LS_d, frame_LW_d, frame_sit_d,␣
↪frame_falls_d, \
    frame_lay_d, frame_layFP_d, frame_sitFP_d, frame_walk_d, frame_walkFP_d,␣
↪frame_ADL_d = getFramesRawData(
        Backwards_d, Backwards_i, Forwards_d, Forwards_i, LateralStand_d,␣
↪LateralStand_i,
        LateralWalk_d, LateralWalk_i, Sitting_d, Sitting_i, Falls_d, Falls_i,␣
↪Lay_d, Lay_i,
        LayFP_d, LayFP_i, SitFP_d, SitFP_i, Walk_d, Walk_i, WalkFP_d, WalkFP_i,␣
↪ADL_d, ADL_i, window_size, step_size)

    frame_noise_d = getFramesRaw(Stand_d, Stand_i, 'adl', window_size, step_size)

    "DELETE DATASET'S SPURIOUS  VALUES"
    # frame_back_d = clean_dataset(frame_back_d)d
    # frame_for_d = clean_dataset(frame_for_d)
    # frame_LS_d = clean_dataset(frame_LS_d)
    # frame_LW_d = clean_dataset(frame_LW_d)
    # frame_sit_d = clean_dataset(frame_sit_d)
    frame_falls_d = clean_dataset(frame_falls_d)
    # frame_lay_d = clean_dataset(frame_lay_d)
    # frame_layFP_d = clean_dataset(frame_layFP_d)
    # frame_sitFP_d = clean_dataset(frame_sitFP_d)
    # frame_walk_d = clean_dataset( frame_walk_d)
    # frame_walkFP_d = clean_dataset(frame_walkFP_d)
    frame_ADL_d = clean_dataset(frame_ADL_d)
    # n_frame_back_d = clean_dataset(n_frame_back_d)
    # n_frame_for_d = clean_dataset(n_frame_for_d)
```

```
    # n_frame_LS_d = clean_dataset(n_frame_LS_d)
    # n_frame_LW_d = clean_dataset(n_frame_LW_d)
    # n_frame_sit_d = clean_dataset(n_frame_sit_d)
    n_frame_falls_d = clean_dataset(n_frame_falls_d)
    # n_frame_lay_d = clean_dataset(n_frame_lay_d)
    # n_frame_layFP_d = clean_dataset(n_frame_layFP_d)
    # n_frame_sitFP_d = clean_dataset(n_frame_sitFP_d)
    # n_frame_walk_d = clean_dataset(n_frame_walk_d)
    # n_frame_walkFP_d = clean_dataset(n_frame_walkFP_d)
    n_frame_ADL_d = clean_dataset(n_frame_ADL_d)

    "TARGETS: Build targets"
    # target_back = getTarget(frame_back_d, 'Backwards')
    # target_for =getTarget(frame_for_d, 'Forwards')
    # target_LS  =getTarget(frame_LS_d, 'Lateral Stand')
    # target_LW = getTarget(frame_LW_d, 'Lateral Walk')
    # target_sit = getTarget(frame_sit_d, 'Sitting')
    # target_lay = getTarget(frame_lay_d, 'Lay')
    # target_layFP = getTarget(frame_layFP_d, 'Lay_FP')
    # target_sitFP = getTarget(frame_sitFP_d, 'Sit_FP')
    # target_walk = getTarget(frame_walk_d, 'Walk')
    # target_walkFP = getTarget(frame_walkFP_d, 'Walk_FP')
    target_f = getTarget(frame_falls_d, 'fall')
    target_a = getTarget(frame_ADL_d, 'ADL')

    # n_target_back = getTarget(n_frame_back_d,'Backwards')
    # n_target_for =getTarget(n_frame_for_d, 'Forwards')
    # n_target_LS  =getTarget(n_frame_LS_d, 'Lateral Stand')
    # n_target_LW = getTarget(n_frame_LW_d, 'Lateral Walk')
    # n_target_sit = getTarget(n_frame_sit_d, 'Sitting')
    # n_target_lay = getTarget(n_frame_lay_d, 'Lay')
    # n_target_layFP = getTarget(n_frame_layFP_d, 'Lay_FP')
    # n_target_sitFP = getTarget(n_frame_sitFP_d, 'Sit_FP')
    # n_target_walk = getTarget(n_frame_walk_d, 'Walk')
    # n_target_walkFP = getTarget(n_frame_walkFP_d, 'Walk_FP')
    n_target_f = getTarget(n_frame_falls_d, 'fall')
    n_target_a = getTarget(n_frame_ADL_d, 'ADL')

    # t_back ={'target':target_back}
    # t_for ={'target':target_for}
    # t_LS ={'target':target_LS}
    # t_LW ={'target':target_LW}
    # t_sit ={'target':target_sit}
    # t_lay ={'target':target_lay}
    # t_layFP ={'target':target_layFP}
    # t_sitFP ={'target':target_sitFP}
    # t_walk ={'target':target_walk}
```

```
    # t_walkFP ={'target':target_walkFP}
    t_f = {'target': target_f}
    t_a = {'target': target_a}

    # n_t_back ={'target':n_target_back}
    # n_t_for ={'target':n_target_for}
    # n_t_LS ={'target':n_target_LS}
    # n_t_LW ={'target':n_target_LW}
    # n_t_sit ={'target':n_target_sit}
    # n_t_lay ={'target':n_target_lay}
    # n_t_layFP ={'target':n_target_layFP}
    # n_t_sitFP ={'target':n_target_sitFP}
    # n_t_walk ={'target':n_target_walk}
    # n_t_walkFP ={'target':n_target_walkFP}
    n_t_f = {'target': n_target_f}
    n_t_a = {'target': n_target_a}
    #
    # frame_target_back = pd.DataFrame(data=t_back)
    # frame_target_for = pd.DataFrame(data=t_for)
    # frame_target_LS = pd.DataFrame(data=t_LS)
    # frame_target_LW = pd.DataFrame(data=t_LW)
    # frame_target_sit = pd.DataFrame(data=t_sit)
    # frame_target_lay = pd.DataFrame(data=t_lay)
    # frame_target_layFP = pd.DataFrame(data=t_layFP)
    # frame_target_sitFP = pd.DataFrame(data=t_sitFP)
    # frame_target_walk = pd.DataFrame(data=t_walk)
    # frame_target_walkFP = pd.DataFrame(data=t_walkFP)
    frame_target_f = pd.DataFrame(data=t_f)
    frame_target_a = pd.DataFrame(data=t_a)

    # n_frame_target_back = pd.DataFrame(data=n_t_back)
    # n_frame_target_for = pd.DataFrame(data=n_t_for)
    # n_frame_target_LS = pd.DataFrame(data=n_t_LS)
    # n_frame_target_LW = pd.DataFrame(data=n_t_LW)
    # n_frame_target_sit = pd.DataFrame(data=n_t_sit)
    # n_frame_target_lay = pd.DataFrame(data=n_t_lay)
    # n_frame_target_layFP = pd.DataFrame(data=n_t_layFP)
    # n_frame_target_sitFP = pd.DataFrame(data=n_t_sitFP)
    # n_frame_target_walk = pd.DataFrame(data=n_t_walk)
    # n_frame_target_walkFP = pd.DataFrame(data=n_t_walkFP)
    n_frame_target_f = pd.DataFrame(data=n_t_f)
    n_frame_target_a = pd.DataFrame(data=n_t_a)

    "FEATURES and TARGETS DEFINITIONS"
    features = ['Ax', 'Ay', 'Az', 'A', 'P1', 'P5', 'Parcs', 'Pheel',␣
↪'Ptot','Roll', 'Pitch', 'Yaw', 'Wx', 'Wy', 'Wz', 'W']
```

```
   n_features = {'n_ax_k', 'n_ay_k', 'n_az_k', 'n_A_k','n_p1_k', 'n_p5_k',␣
↪'n_parc_k', 'n_heel_k', 'n_ptot_k', 'n_roll_k',
                 'n_pitch_k', 'n_yaw_k', 'n_wx_k', 'n_wy_k', 'n_wz_k', 'n_w_k',␣
↪'n_ax_max', 'n_ay_max', 'n_az_max', 'n_A_max',
                 'n_p1_max', 'n_p5_max', 'n_parc_max', 'n_heel_max',␣
↪'n_ptot_max', 'n_roll_max', 'n_pitch_max', 'n_yaw_max',
                 'n_wx_max', 'n_wy_max', 'n_wz_max', 'n_w_max', 'n_ax_min',␣
↪'n_ay_min', 'n_az_min', 'n_A_min', 'n_p1_min', 'n_p5_min',
                 'n_parc_min', 'n_heel_min', 'n_ptot_min', 'n_roll_min',␣
↪'n_pitch_min', 'n_yaw_min', 'n_wx_min','n_wy_min', 'n_wz_min',
                 'n_w_min', 'n_ax_med', 'n_ay_med', 'n_az_med', 'n_A_med',␣
↪'n_p1_med', 'n_p5_med', 'n_parc_med', 'n_heel_med',
                 'n_ptot_med', 'n_roll_med', 'n_pitch_med', 'n_yaw_med',␣
↪'n_wx_med', 'n_wy_med', 'n_wz_med', 'n_w_med', 'n_ax_dev',
                 'n_ay_dev', 'n_az_dev', 'n_A_dev', 'n_p1_dev', 'n_p5_dev',␣
↪'n_parc_dev', 'n_heel_dev', 'n_ptot_dev', 'n_roll_dev',
                 'n_pitch_dev', 'n_yaw_dev', 'n_wx_dev', 'n_wy_dev',␣
↪'n_wz_dev', 'n_w_dev', 'n_ax_var', 'n_ay_var', 'n_az_var',
                 'n_A_var', 'n_p1_var', 'n_p5_var', 'n_parc_var', 'n_heel_var',␣
↪'n_ptot_var', 'n_roll_var', 'n_pitch_var', 'n_yaw_var',
                 'n_wx_var','n_wy_var','n_wz_var', 'n_w_var', 'n_ax_pow',␣
↪'n_ay_pow', 'n_az_pow', 'n_A_pow', 'n_p1_pow', 'n_p5_pow',
                 'n_parc_pow', 'n_heel_pow', 'n_ptot_pow', 'n_roll_pow',␣
↪'n_pitch_pow', 'n_yaw_pow', 'n_wx_pow', 'n_wy_pow',
                 'n_wz_pow', 'n_w_pow'}

   # features = ['Ax', 'Ay', 'Az', 'A', 'Ptot', 'Roll', 'Pitch', 'Yaw', 'Wx',␣
↪'Wy', 'Wz', 'W']
   # n_features = { 'n_A_max', 'n_A_min',  'n_wz_min', 'n_p1_med', 'n_roll_var'}
   # n_features = {'n_wx_k', 'n_ay_dev', 'n_yaw_var', 'n_ay_pow',␣
↪'n_pitch_pow', 'n_w_pow'}

   targets = ['fall', 'ADL']
   # targets = ['Lay', 'Lay_FP', 'Sit_FP','Walk','Walk_FP']
   # targets = ['Backwards', 'Forwards', 'Lateral Stand','Lateral␣
↪Walk','Sitting']
   # targets = ['Backwards', 'Forwards', 'Lateral Stand','Lateral␣
↪Walk','Sitting','Lay', 'Lay_FP', 'Sit_FP','Walk','Walk_FP' ]

   frames = pd.concat([frame_falls_d, frame_ADL_d])
   n_frames = pd.concat([n_frame_falls_d, n_frame_ADL_d])
   target = pd.concat([frame_target_f, frame_target_a])
   n_target = pd.concat([n_frame_target_f, n_frame_target_a])
   # frames = pd.concat([frame_back_d,␣
↪frame_for_d,frame_LS_d,frame_LW_d,frame_sit_d])
```

```
    # target = pd.
↪concat([frame_target_back,frame_target_for,frame_target_LS,frame_target_LW,frame_target_sit])
    # frames = pd.concat([frame_lay_d,␣
↪frame_layFP_d,frame_sitFP_d,frame_walk_d,frame_walkFP_d])
    # target = pd.
↪concat([frame_target_lay,frame_target_layFP,frame_target_sitFP,frame_target_walk,frame_target
    # frames = pd.concat([frame_back_d,␣
↪frame_for_d,frame_LS_d,frame_LW_d,frame_sit_d,frame_lay_d,␣
↪frame_layFP_d,frame_sitFP_d,frame_walk_d,frame_walkFP_d])
    # target = pd.
↪concat([frame_target_back,frame_target_for,frame_target_LS,frame_target_LW,frame_target_sit,f
    # n_frames = pd.concat([n_frame_back_d,␣
↪n_frame_for_d,n_frame_LS_d,n_frame_LW_d,n_frame_sit_d,n_frame_lay_d,␣
↪n_frame_layFP_d,n_frame_sitFP_d,n_frame_walk_d,n_frame_walkFP_d])
    # n_target = pd.
↪concat([n_frame_target_back,n_frame_target_for,n_frame_target_LS,n_frame_target_LW,n_frame_ta
    x = frames.loc[:, features].values
    n_x = n_frames.loc[:, n_features].values
    y = target.loc[:, ['target']].values
    n_y = n_target.loc[:, ['target']].values

    # DATA SETS SPLIT: 70% train 30% test
    train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=1 / 7.0,␣
↪random_state=0)
    n_train_x, n_test_x, n_train_y, n_test_y = train_test_split(n_x, n_y,␣
↪test_size=1 / 7.0, random_state=1)

    #Scaler
    scaler = StandardScaler()
    n_scaler = StandardScaler()

    #Fit on training set only.
    scaler.fit(train_x)
    n_scaler.fit(n_train_x)
    #Apply transform to both the training set and the test set.
    train_x = scaler.transform(train_x)
    test_x = scaler.transform(test_x)
    n_train_x = n_scaler.transform(n_train_x)
    n_test_x = n_scaler.transform(n_test_x)

    # PCA
    pca = PCA(.95)
    n_pca = PCA(.95)
    pca.fit(train_x)
    n_pca.fit(n_train_x)
    train_pca_x = pca.transform(train_x)
```

```
    test_pca_x = pca.transform(test_x)
    n_train_pca_x = n_pca.transform(n_train_x)
    n_test_pca_x = n_pca.transform(n_test_x)

    return x, y, n_x, n_y,frame_falls_d, frame_ADL_d, n_frame_falls_d,␣
↪n_frame_ADL_d, train_x, test_x, train_y, test_y, n_train_x, n_test_x,␣
↪n_train_y, n_test_y, train_pca_x, test_pca_x, n_train_pca_x, n_test_pca_x
```

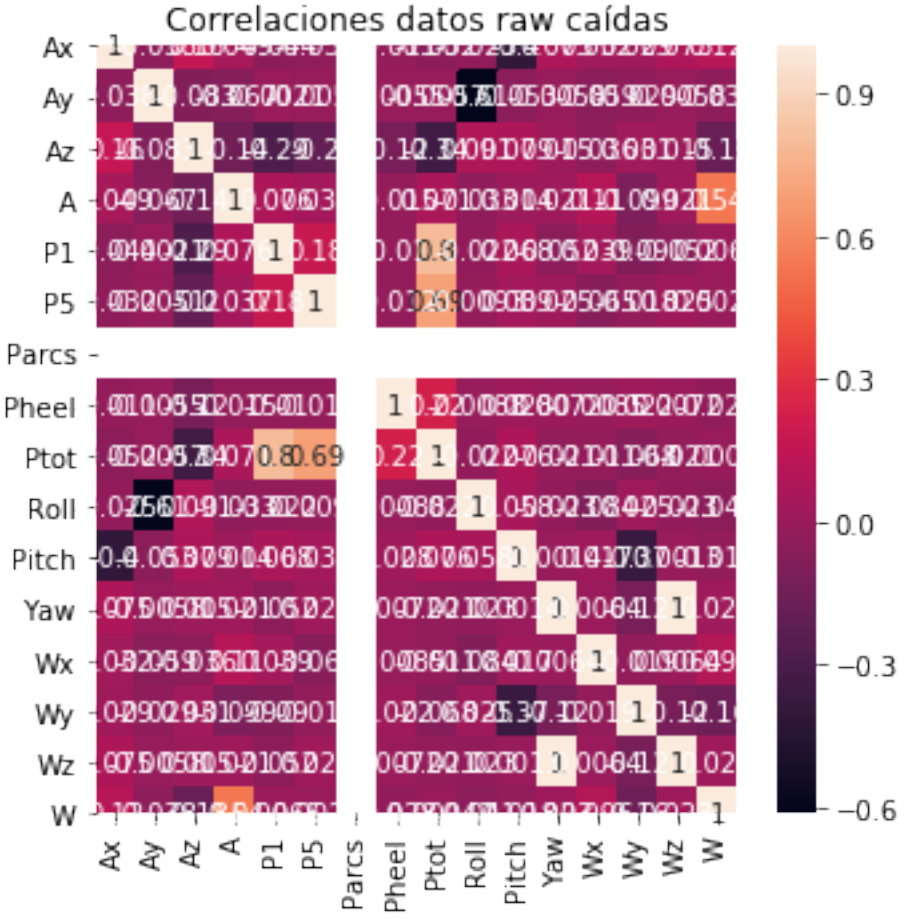
1.3 Programa principal

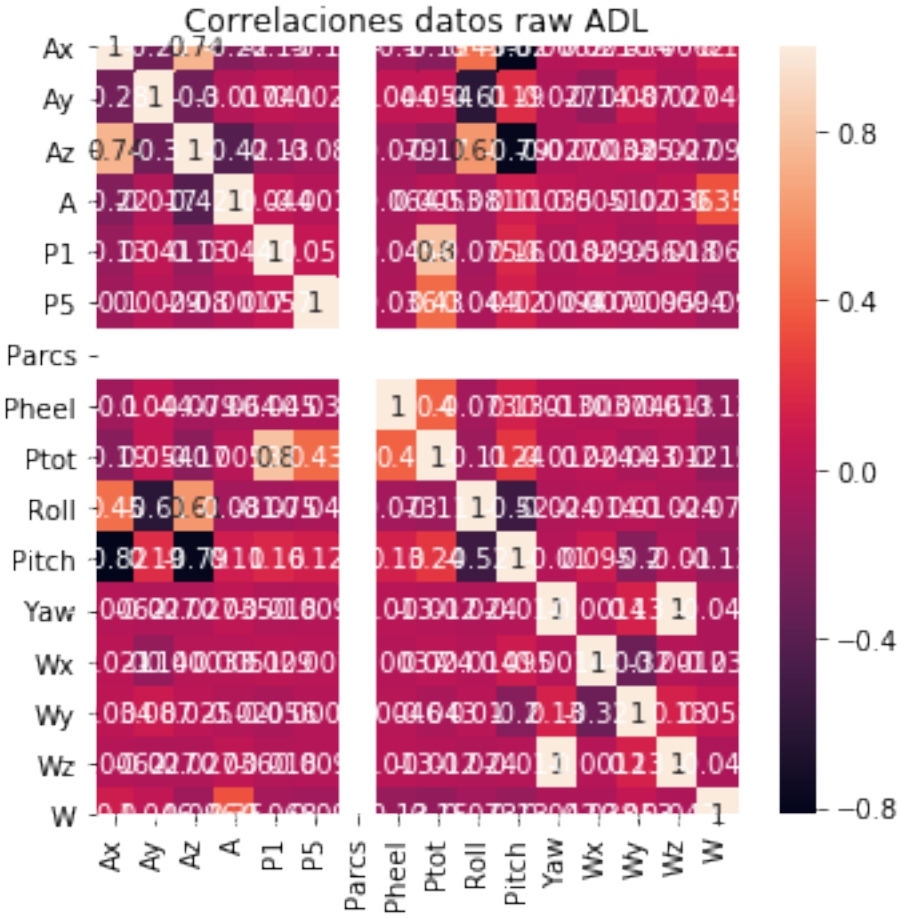
1.3.1 Definicion de vectores datos

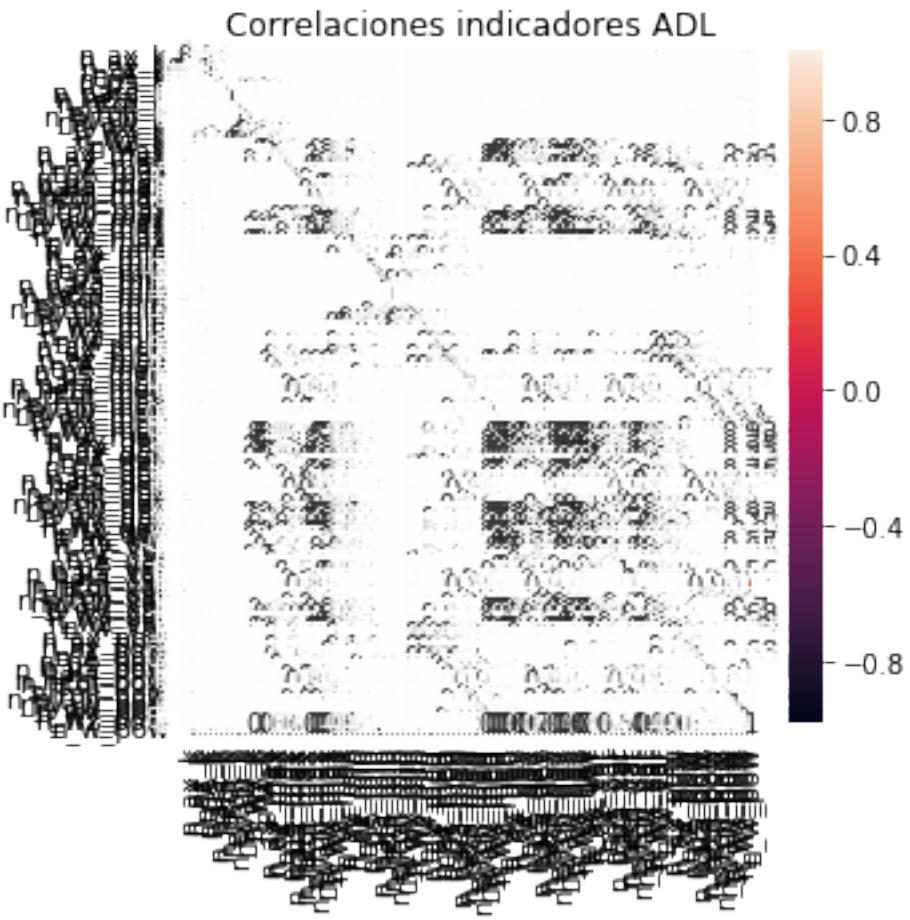
[30]:
```
x, y, n_x, n_y, frame_falls_d, frame_ADL_d, n_frame_falls_d, n_frame_ADL_d,␣
↪train_x, test_x, train_y, test_y, n_train_x, n_test_x, n_train_y, n_test_y,␣
↪train_pca_x, test_pca_x, n_train_pca_x, n_test_pca_x = getDataSets()
```

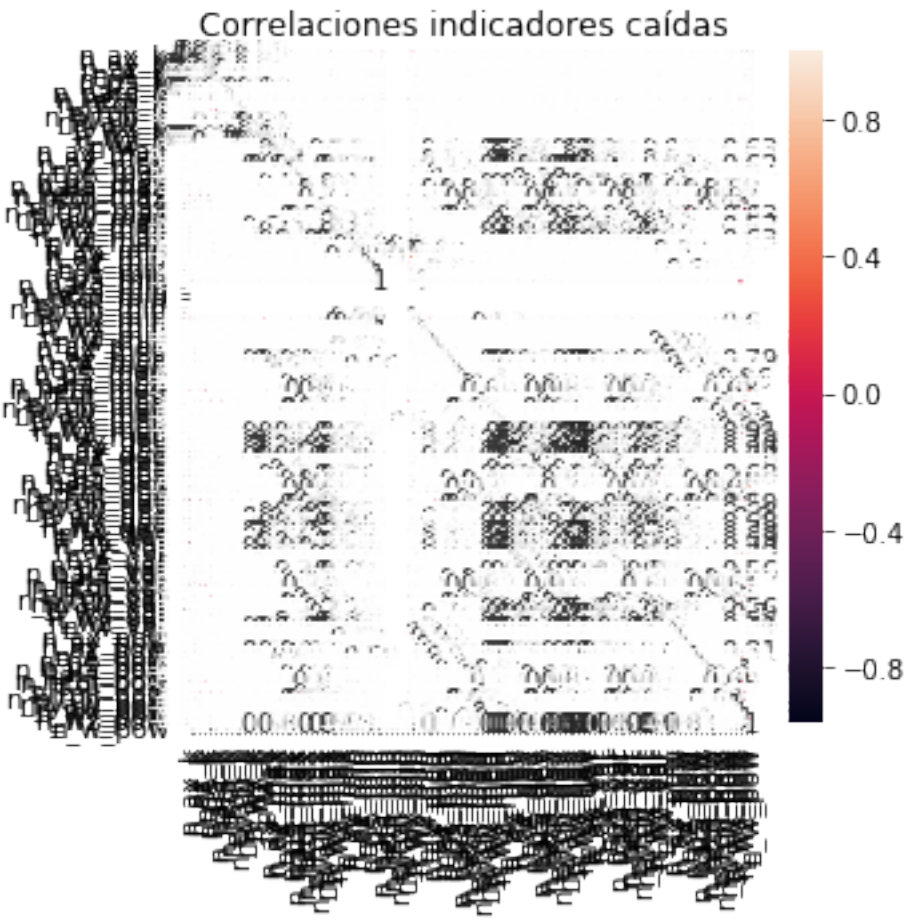
1.3.2 Correlaciones

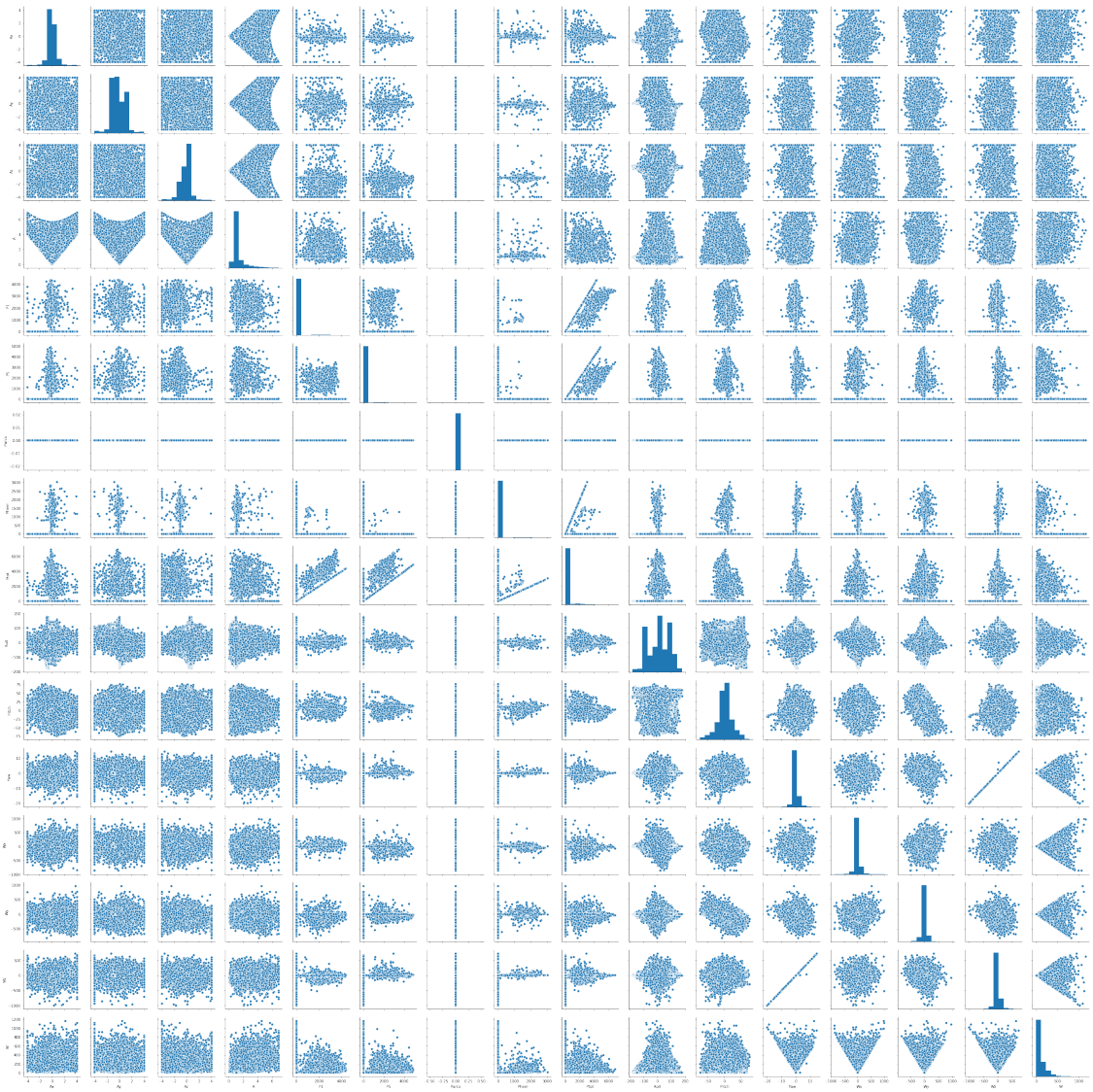
[31]:
```
correlations(frame_falls_d, frame_ADL_d, n_frame_falls_d, n_frame_ADL_d)
```

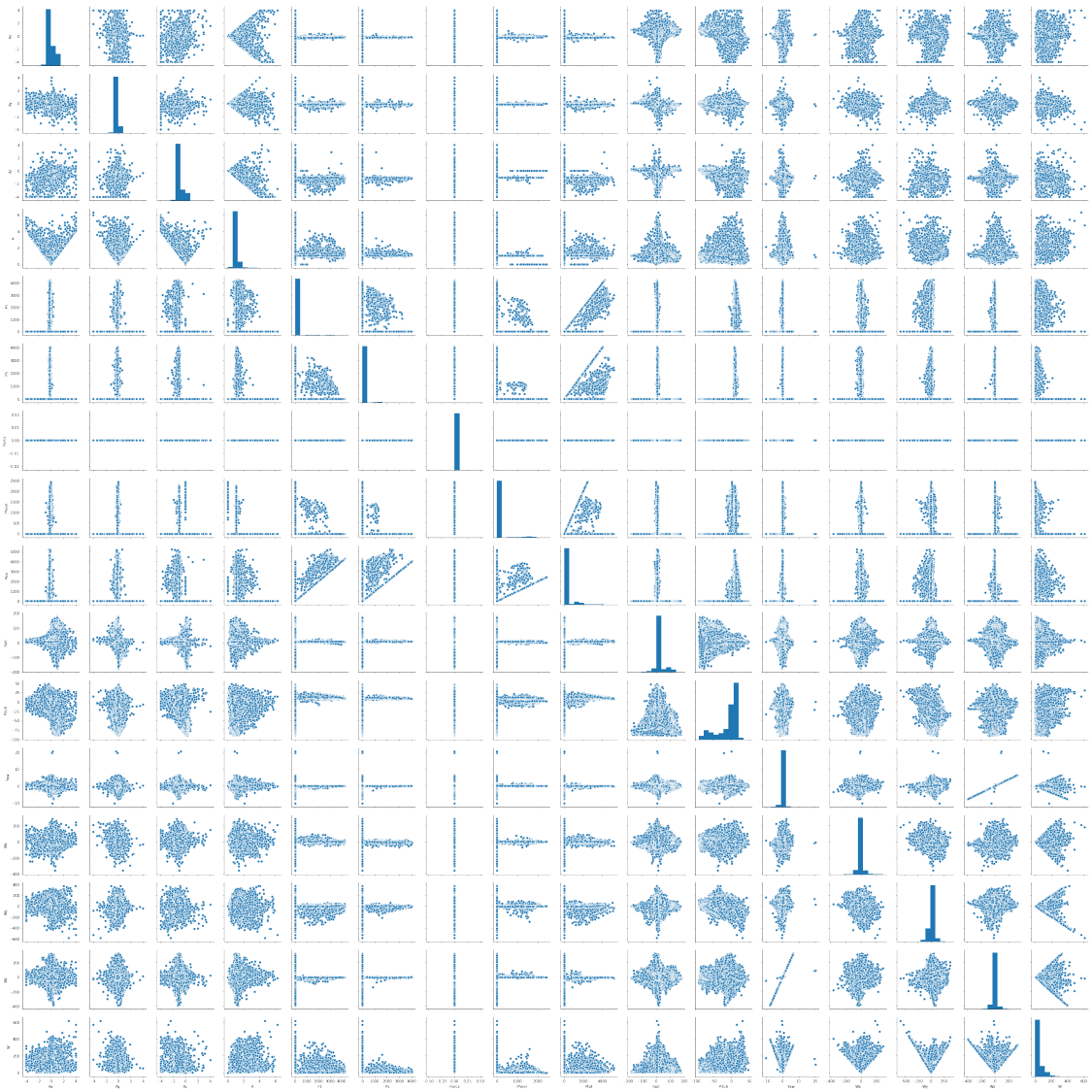



Correlaciones datos raw ADL
Ax
Ay
Az
A
P1
P5
Parcs
Pheel
Ptot
Roll
Pitch
Yaw
Wx
Wy
Wz
W
0.8
0.4
0.0
−0.4
−0.8


Correlaciones indicadores ADL
0.8
0.4
0.0
−0.4
−0.8


Correlaciones indicadores caídas
0.8
0.4
0.0
−0.4
−0.8

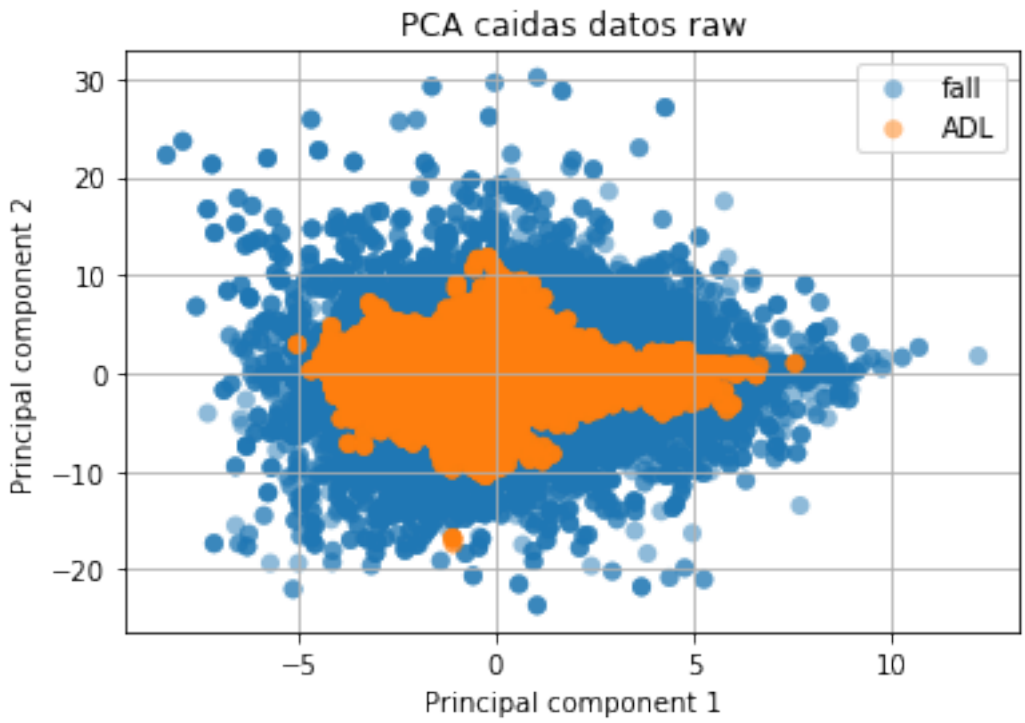




1.3.3 Análisis PCA

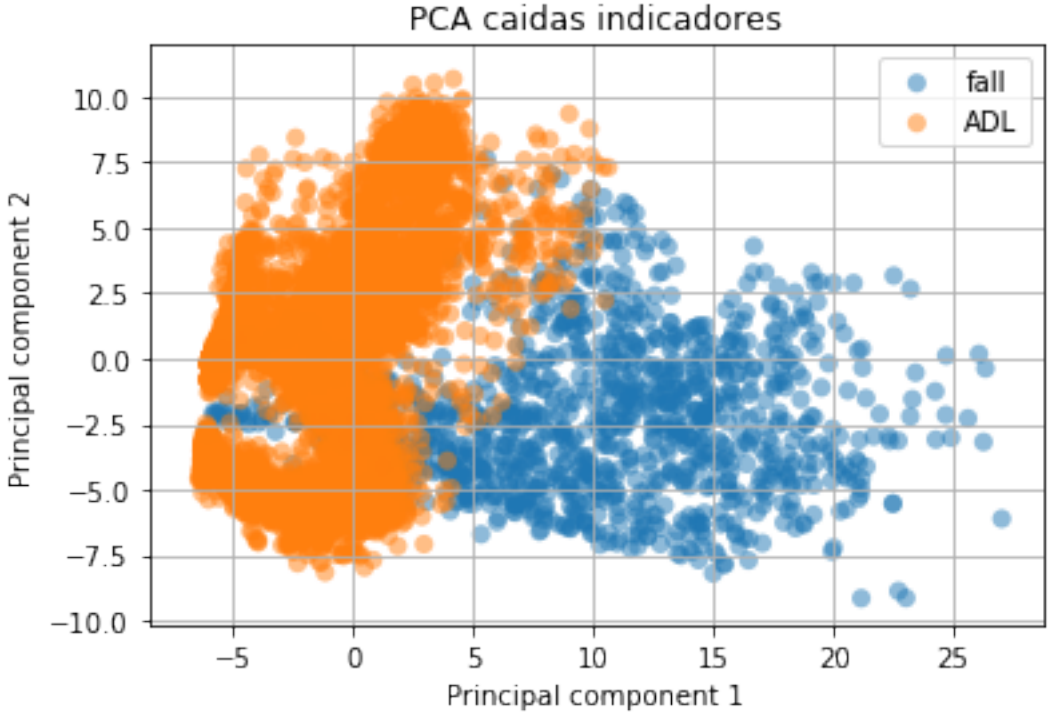
PCA datos en crudo

[32]:
```
PCAanalisis(x,y.ravel(),'PCA caidas datos raw')
```



PCA indicadores

[33]:
```
PCAanalisis(n_x,n_y.ravel(),'PCA caidas indicadores')
```

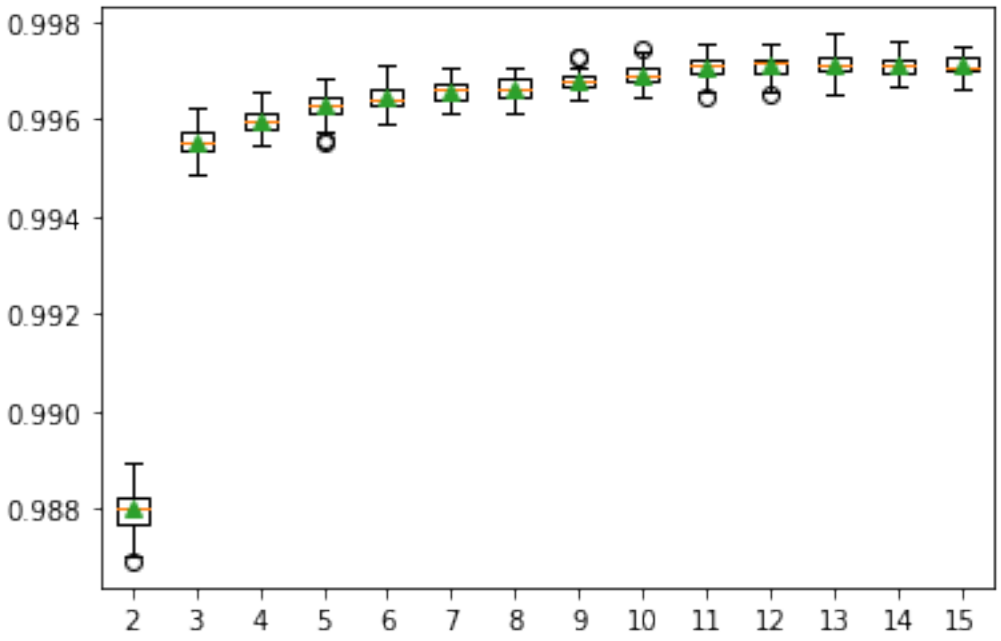


1.3.4 Reducción de características

RFE

[34]:
```
getRFE_raw(x,y)
```

```
>2 0.988 (0.000)
>3 0.996 (0.000)
>4 0.996 (0.000)
>5 0.996 (0.000)
>6 0.996 (0.000)
>7 0.997 (0.000)
>8 0.997 (0.000)
>9 0.997 (0.000)
>10 0.997 (0.000)
>11 0.997 (0.000)
>12 0.997 (0.000)
>13 0.997 (0.000)
>14 0.997 (0.000)
>15 0.997 (0.000)
```




```
Column: 0, Selected True, Rank: 1.000
Column: 1, Selected True, Rank: 1.000
Column: 2, Selected True, Rank: 1.000
Column: 3, Selected True, Rank: 1.000
```

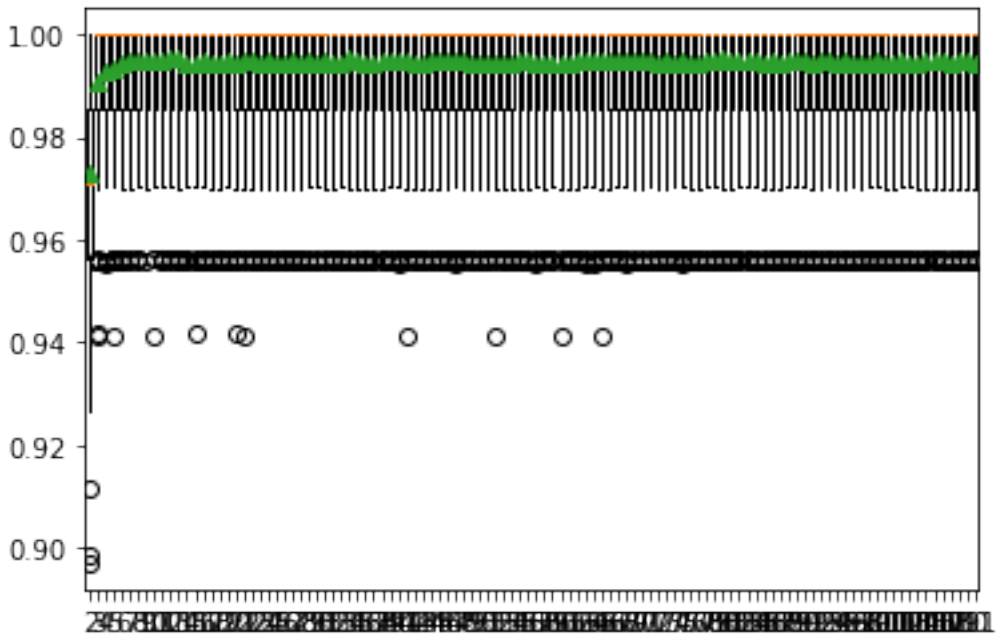
```
Column: 4, Selected False, Rank: 2.000
Column: 5, Selected False, Rank: 4.000
Column: 6, Selected False, Rank: 5.000
Column: 7, Selected False, Rank: 3.000
Column: 8, Selected True, Rank: 1.000
Column: 9, Selected True, Rank: 1.000
Column: 10, Selected True, Rank: 1.000
Column: 11, Selected True, Rank: 1.000
Column: 12, Selected True, Rank: 1.000
Column: 13, Selected True, Rank: 1.000
Column: 14, Selected True, Rank: 1.000
Column: 15, Selected True, Rank: 1.000
```

[35]:
```
getRFE_n(n_x,n_y)
```

```
>2 0.973 (0.020)
>3 0.991 (0.012)
>4 0.993 (0.011)
>5 0.993 (0.011)
>6 0.995 (0.010)
>7 0.995 (0.009)
>8 0.995 (0.009)
>9 0.995 (0.009)
>10 0.995 (0.009)
>11 0.995 (0.009)
>12 0.996 (0.008)
>13 0.995 (0.009)
>14 0.995 (0.009)
>15 0.994 (0.009)
>16 0.995 (0.009)
>17 0.994 (0.009)
>18 0.994 (0.009)
>19 0.995 (0.009)
>20 0.994 (0.010)
>21 0.995 (0.009)
>22 0.995 (0.009)
>23 0.995 (0.009)
>24 0.995 (0.009)
>25 0.995 (0.009)
>26 0.994 (0.010)
>27 0.995 (0.009)
>28 0.994 (0.009)
>29 0.995 (0.009)
>30 0.994 (0.009)
>31 0.995 (0.009)
>32 0.995 (0.009)
>33 0.994 (0.009)
>34 0.995 (0.008)
```

>35 0.995 (0.009)
>36 0.994 (0.009)
>37 0.994 (0.009)
>38 0.995 (0.009)
>39 0.995 (0.009)
>40 0.995 (0.009)
>41 0.995 (0.009)
>42 0.995 (0.009)
>43 0.995 (0.009)
>44 0.995 (0.009)
>45 0.995 (0.009)
>46 0.995 (0.009)
>47 0.995 (0.009)
>48 0.995 (0.009)
>49 0.995 (0.009)
>50 0.994 (0.009)
>51 0.994 (0.009)
>52 0.995 (0.009)
>53 0.994 (0.009)
>54 0.994 (0.010)
>55 0.995 (0.009)
>56 0.995 (0.009)
>57 0.995 (0.009)
>58 0.995 (0.009)
>59 0.994 (0.009)
>60 0.995 (0.009)
>61 0.995 (0.009)
>62 0.995 (0.009)
>63 0.995 (0.009)
>64 0.995 (0.009)
>65 0.995 (0.009)
>66 0.995 (0.009)
>67 0.995 (0.009)
>68 0.995 (0.009)
>69 0.995 (0.009)
>70 0.995 (0.009)
>71 0.995 (0.009)
>72 0.995 (0.009)
>73 0.995 (0.010)
>74 0.995 (0.009)
>75 0.995 (0.009)
>76 0.995 (0.009)
>77 0.995 (0.009)
>78 0.995 (0.009)
>79 0.995 (0.009)
>80 0.995 (0.009)
>81 0.995 (0.009)
>82 0.995 (0.009)

>83 0.995 (0.009)
>84 0.995 (0.009)
>85 0.994 (0.009)
>86 0.995 (0.009)
>87 0.995 (0.009)
>88 0.995 (0.009)
>89 0.995 (0.009)
>90 0.995 (0.009)
>91 0.995 (0.009)
>92 0.995 (0.009)
>93 0.995 (0.009)
>94 0.995 (0.009)
>95 0.995 (0.009)
>96 0.995 (0.009)
>97 0.994 (0.009)
>98 0.994 (0.010)
>99 0.995 (0.009)
>100 0.995 (0.010)
>101 0.995 (0.009)
>102 0.995 (0.009)
>103 0.995 (0.009)
>104 0.995 (0.009)
>105 0.995 (0.009)
>106 0.995 (0.009)
>107 0.995 (0.009)
>108 0.994 (0.009)
>109 0.995 (0.009)
>110 0.995 (0.009)
>111 0.994 (0.010)

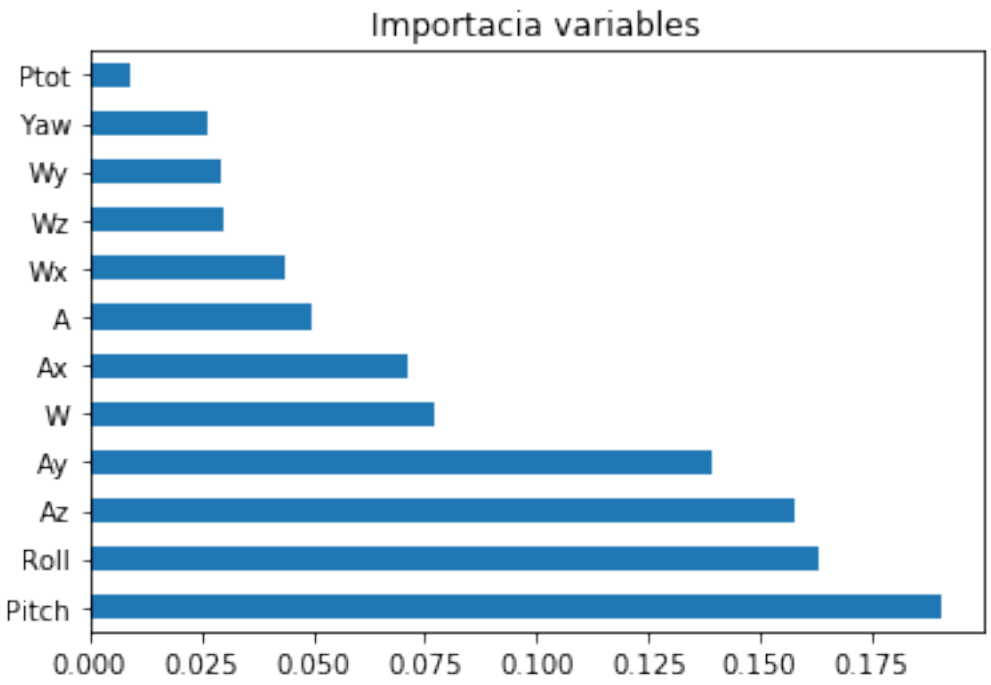


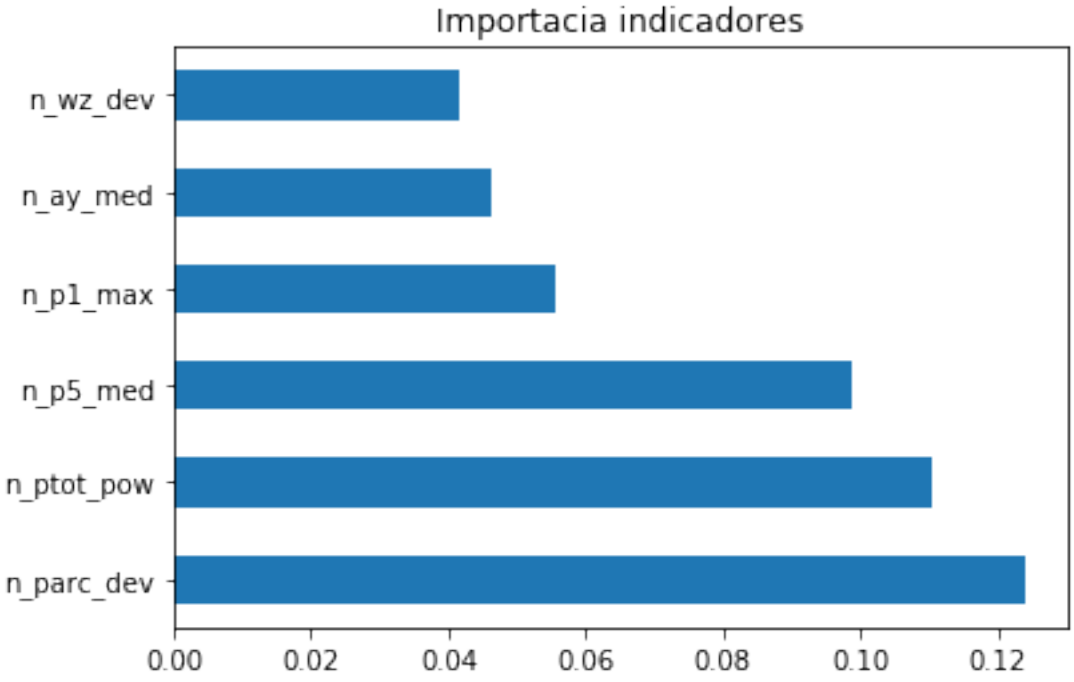

```
Column: 0, Selected False, Rank: 107.000
Column: 1, Selected False, Rank: 106.000
Column: 2, Selected False, Rank: 105.000
Column: 3, Selected False, Rank: 104.000
Column: 4, Selected False, Rank: 103.000
Column: 5, Selected False, Rank: 102.000
Column: 6, Selected False, Rank: 101.000
Column: 7, Selected False, Rank: 100.000
Column: 8, Selected False, Rank: 99.000
Column: 9, Selected False, Rank: 98.000
Column: 10, Selected False, Rank: 95.000
Column: 11, Selected False, Rank: 94.000
Column: 12, Selected False, Rank: 93.000
Column: 13, Selected False, Rank: 18.000
Column: 14, Selected False, Rank: 31.000
Column: 15, Selected False, Rank: 29.000
Column: 16, Selected False, Rank: 28.000
Column: 17, Selected False, Rank: 27.000
Column: 18, Selected False, Rank: 26.000
Column: 19, Selected False, Rank: 25.000
Column: 20, Selected False, Rank: 23.000
Column: 21, Selected False, Rank: 4.000
Column: 22, Selected True, Rank: 1.000
Column: 23, Selected False, Rank: 38.000
Column: 24, Selected False, Rank: 24.000
Column: 25, Selected False, Rank: 19.000
```

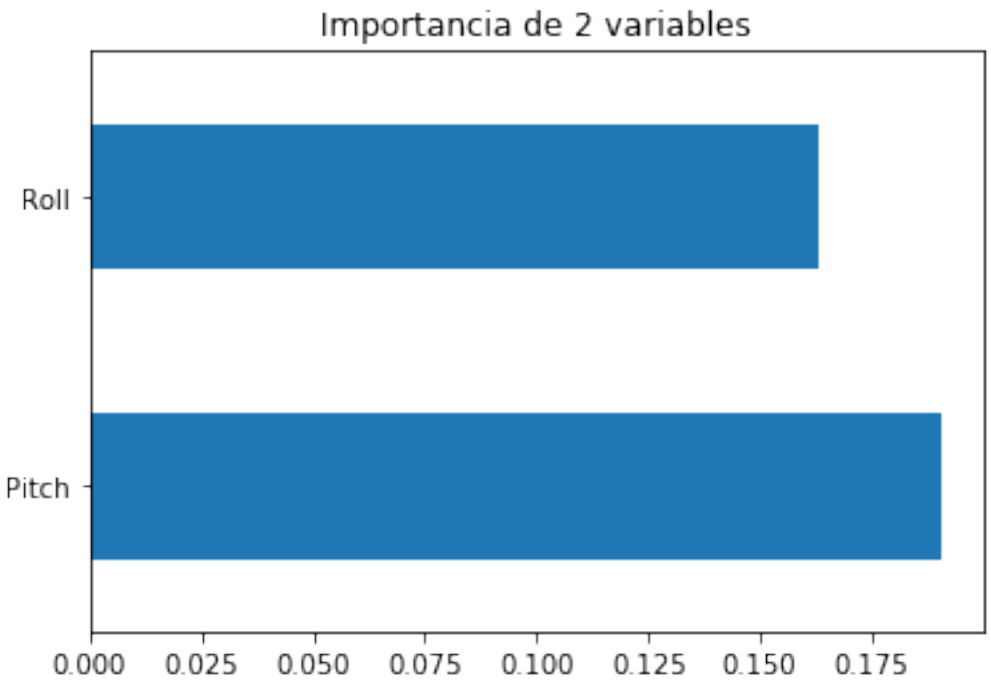
```
Column: 26, Selected False, Rank: 44.000
Column: 27, Selected False, Rank: 39.000
Column: 28, Selected False, Rank: 53.000
Column: 29, Selected False, Rank: 46.000
Column: 30, Selected True, Rank: 1.000
Column: 31, Selected False, Rank: 45.000
Column: 32, Selected False, Rank: 40.000
Column: 33, Selected False, Rank: 54.000
Column: 34, Selected False, Rank: 58.000
Column: 35, Selected False, Rank: 51.000
Column: 36, Selected False, Rank: 49.000
Column: 37, Selected False, Rank: 9.000
Column: 38, Selected False, Rank: 66.000
Column: 39, Selected False, Rank: 10.000
Column: 40, Selected False, Rank: 70.000
Column: 41, Selected False, Rank: 65.000
Column: 42, Selected False, Rank: 74.000
Column: 43, Selected False, Rank: 6.000
Column: 44, Selected False, Rank: 79.000
Column: 45, Selected False, Rank: 16.000
Column: 46, Selected False, Rank: 68.000
Column: 47, Selected False, Rank: 14.000
Column: 48, Selected False, Rank: 83.000
Column: 49, Selected False, Rank: 5.000
Column: 50, Selected False, Rank: 82.000
Column: 51, Selected False, Rank: 55.000
Column: 52, Selected False, Rank: 59.000
Column: 53, Selected False, Rank: 61.000
Column: 54, Selected False, Rank: 71.000
Column: 55, Selected False, Rank: 75.000
Column: 56, Selected False, Rank: 13.000
Column: 57, Selected False, Rank: 85.000
Column: 58, Selected False, Rank: 87.000
Column: 59, Selected False, Rank: 89.000
Column: 60, Selected False, Rank: 97.000
Column: 61, Selected False, Rank: 96.000
Column: 62, Selected False, Rank: 91.000
Column: 63, Selected False, Rank: 92.000
Column: 64, Selected False, Rank: 90.000
Column: 65, Selected False, Rank: 88.000
Column: 66, Selected False, Rank: 86.000
Column: 67, Selected False, Rank: 84.000
Column: 68, Selected False, Rank: 3.000
Column: 69, Selected False, Rank: 7.000
Column: 70, Selected False, Rank: 11.000
Column: 71, Selected True, Rank: 1.000
Column: 72, Selected False, Rank: 22.000
Column: 73, Selected False, Rank: 20.000
```

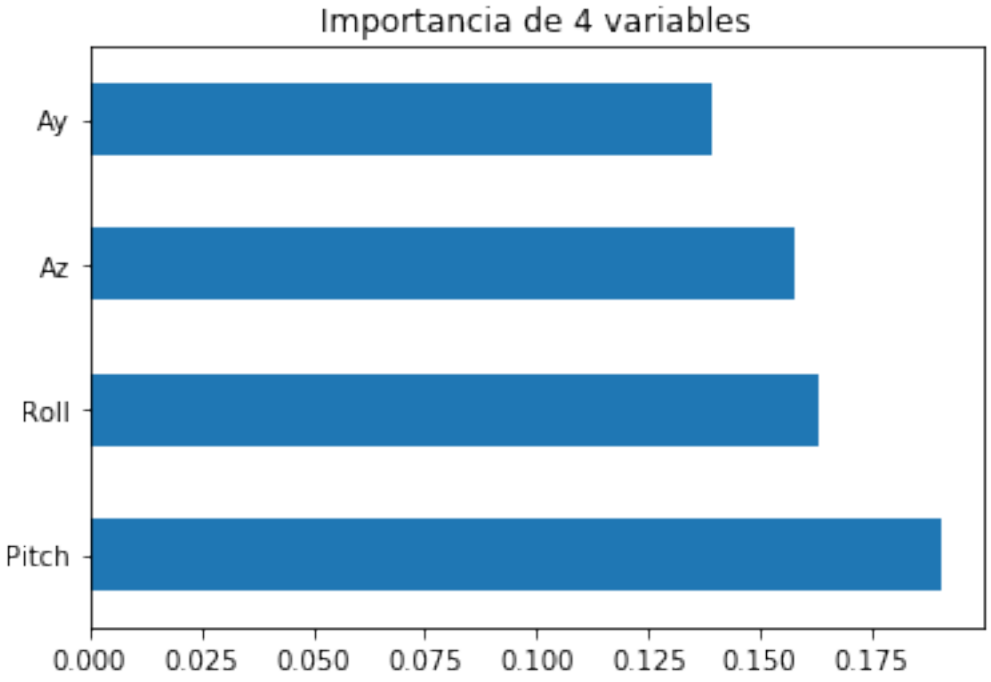
```
Column: 74, Selected False, Rank: 12.000
Column: 75, Selected False, Rank: 35.000
Column: 76, Selected False, Rank: 30.000
Column: 77, Selected False, Rank: 41.000
Column: 78, Selected True, Rank: 1.000
Column: 79, Selected False, Rank: 15.000
Column: 80, Selected False, Rank: 57.000
Column: 81, Selected False, Rank: 50.000
Column: 82, Selected False, Rank: 32.000
Column: 83, Selected False, Rank: 56.000
Column: 84, Selected False, Rank: 48.000
Column: 85, Selected False, Rank: 52.000
Column: 86, Selected False, Rank: 47.000
Column: 87, Selected False, Rank: 34.000
Column: 88, Selected False, Rank: 33.000
Column: 89, Selected False, Rank: 36.000
Column: 90, Selected False, Rank: 77.000
Column: 91, Selected False, Rank: 42.000
Column: 92, Selected False, Rank: 60.000
Column: 93, Selected False, Rank: 63.000
Column: 94, Selected False, Rank: 69.000
Column: 95, Selected False, Rank: 80.000
Column: 96, Selected False, Rank: 64.000
Column: 97, Selected False, Rank: 81.000
Column: 98, Selected False, Rank: 72.000
Column: 99, Selected False, Rank: 76.000
Column: 100, Selected False, Rank: 62.000
Column: 101, Selected False, Rank: 67.000
Column: 102, Selected False, Rank: 78.000
Column: 103, Selected False, Rank: 8.000
Column: 104, Selected True, Rank: 1.000
Column: 105, Selected False, Rank: 73.000
Column: 106, Selected False, Rank: 2.000
Column: 107, Selected False, Rank: 43.000
Column: 108, Selected False, Rank: 37.000
Column: 109, Selected False, Rank: 21.000
Column: 110, Selected False, Rank: 17.000
Column: 111, Selected True, Rank: 1.000
```

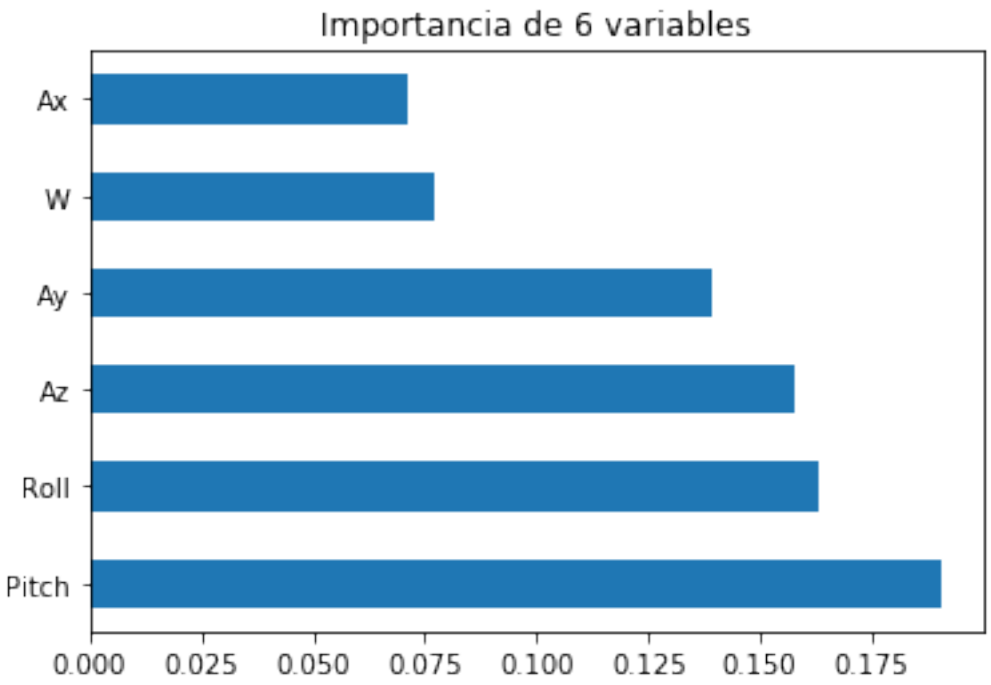
[48]:
```
getFI(x,y,n_x,n_y, frame_falls_d,n_frame_falls_d)
```

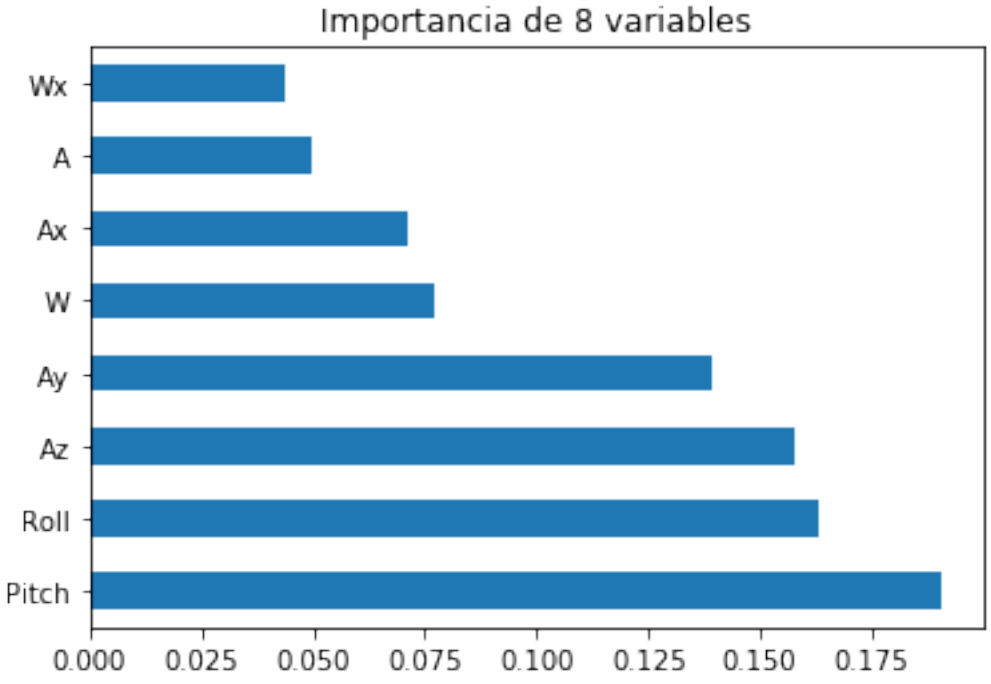

Importacia variables
Ptot
Yaw
Wy
Wz
Wx
A
Ax
W
Ay
Az
Roll
Pitch
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

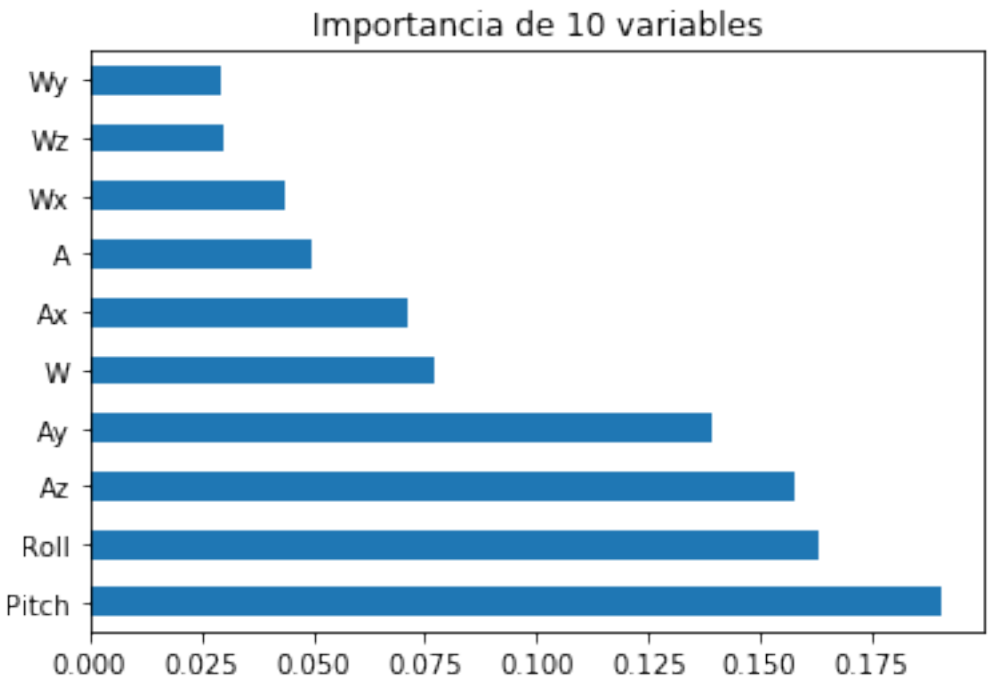

Importacia indicadores
n_wz_dev
n_ay_med
n_p1_max
n_p5_med
n_ptot_pow
n_parc_dev
0.00
0.02
0.04
0.06
0.08
0.10
0.12

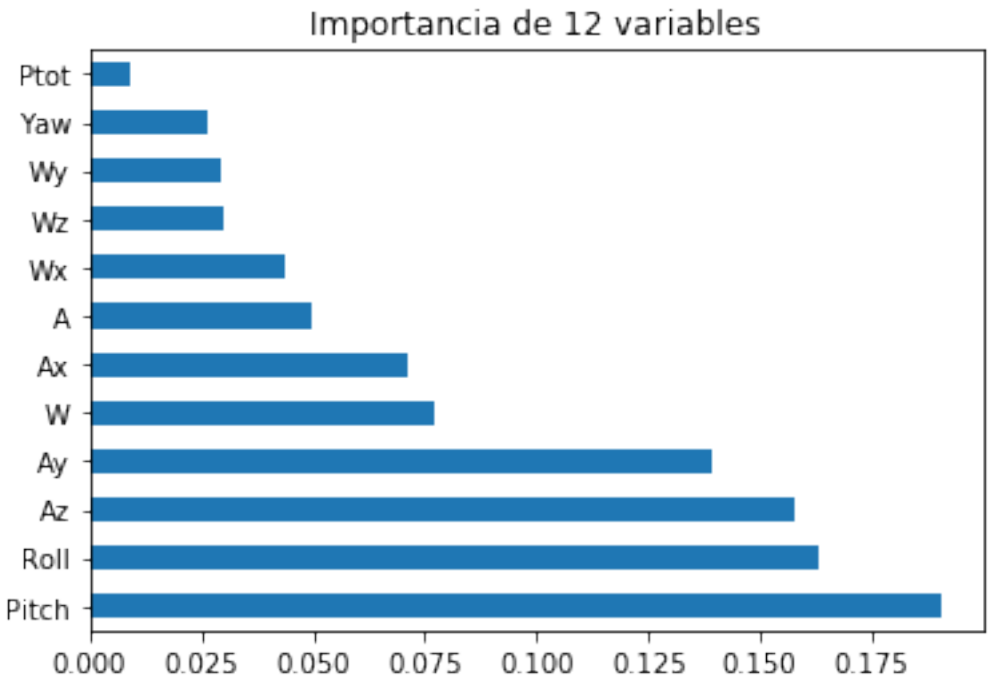

Importancia de 2 variables
Roll
Pitch
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

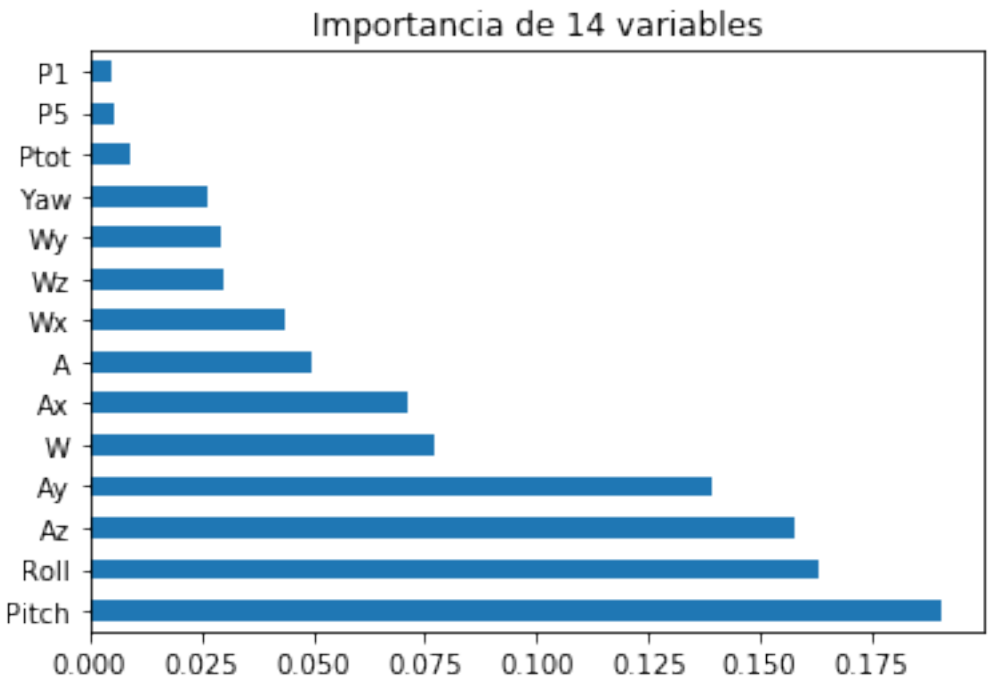

Importancia de 4 variables
Ay
Az
Roll
Pitch
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

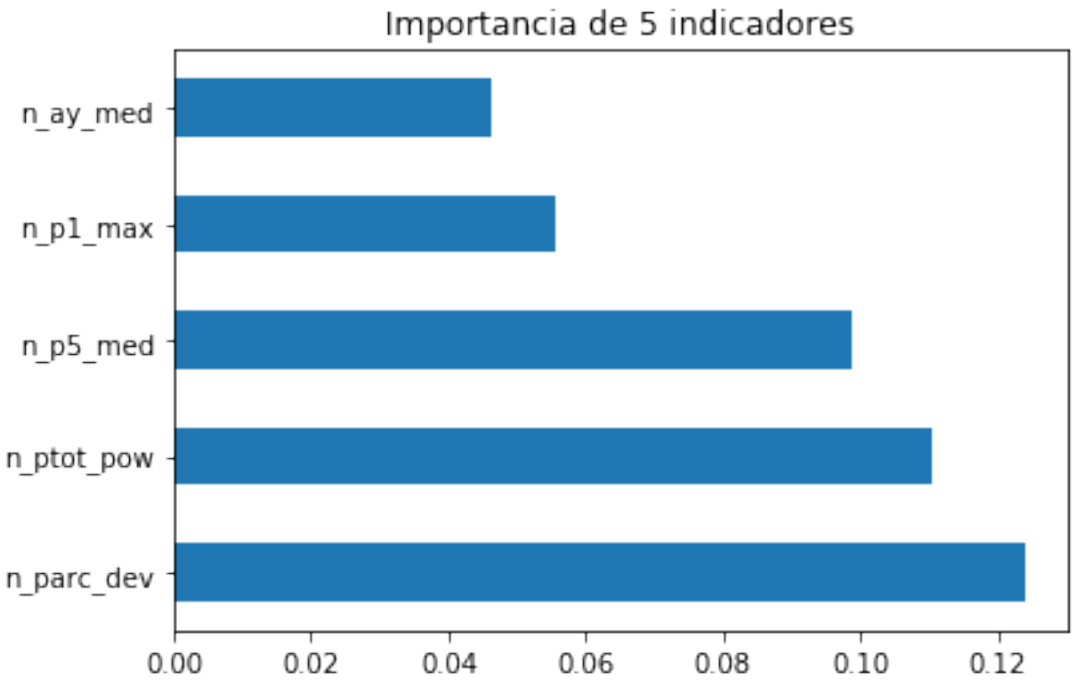

Importancia de 6 variables
Ax
W
Ay
Az
Roll
Pitch
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

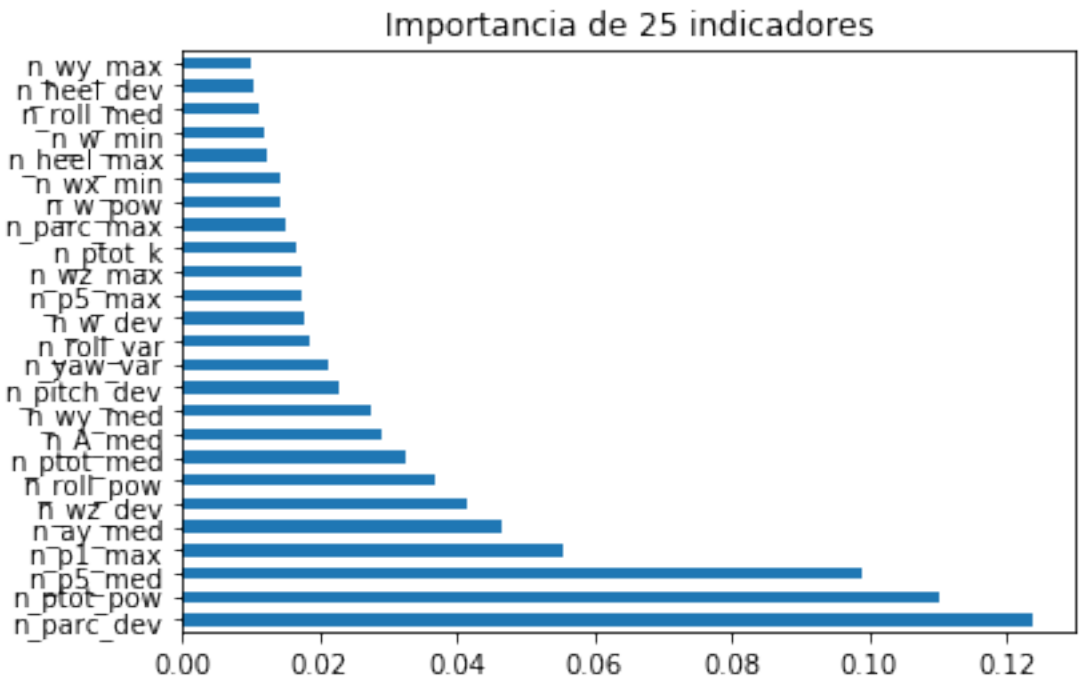

Importancia de 8 variables
Wx
A
Ax
W
Ay
Az
Roll
Pitch
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

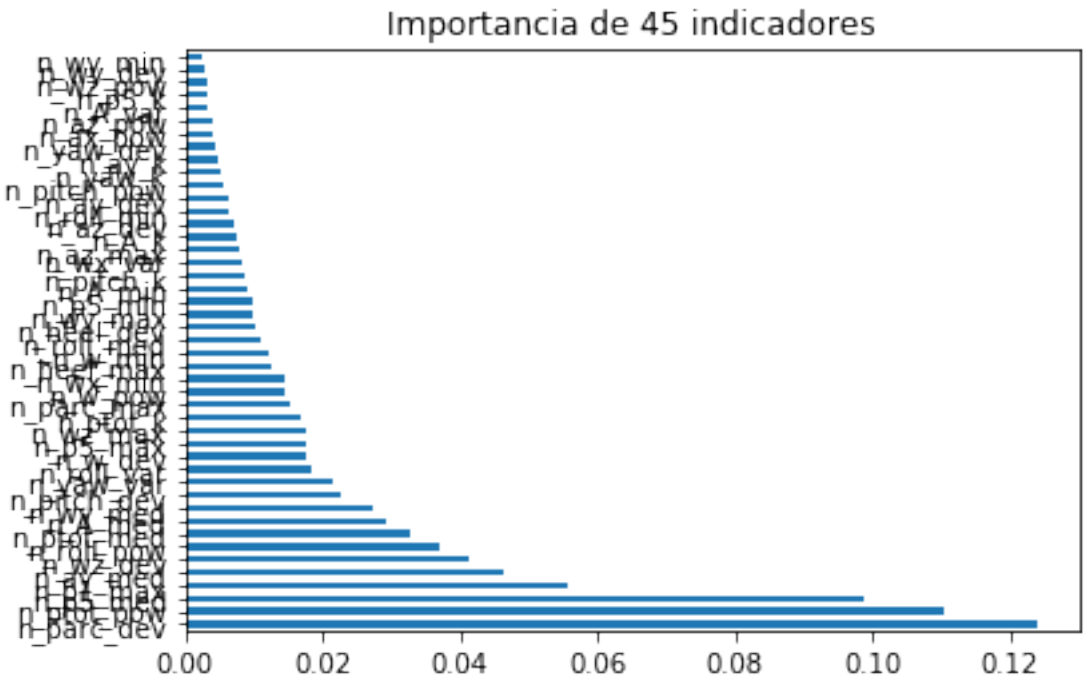

Importancia de 10 variables
Wy
Wz
Wx
A
Ax
W
Ay
Az
Roll
Pitch
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

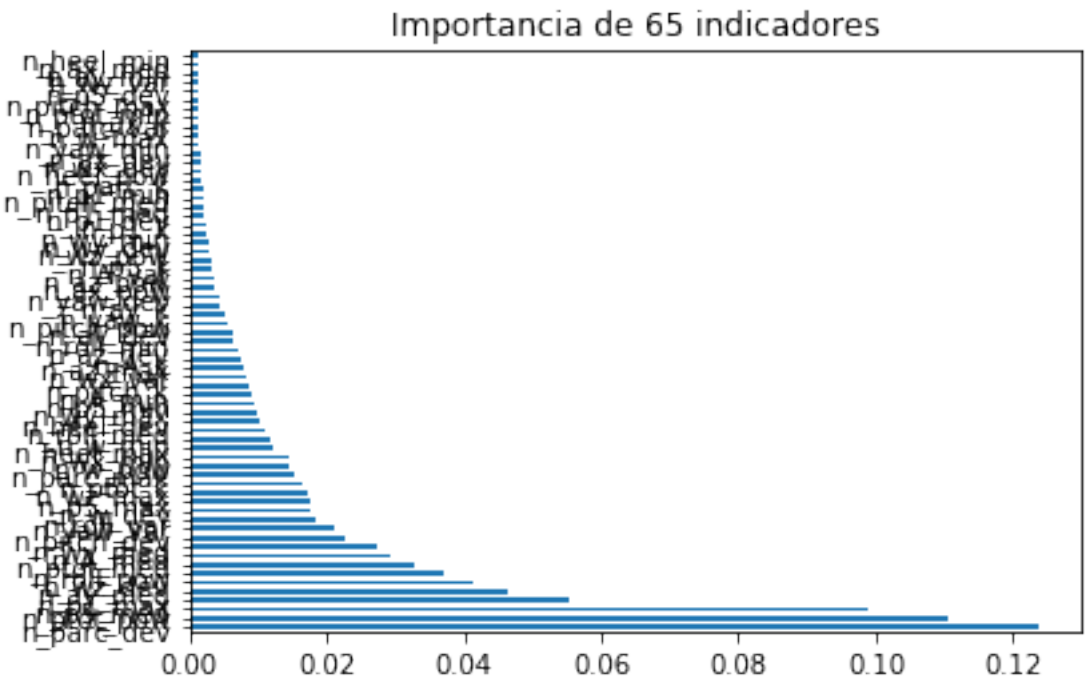

Importancia de 12 variables
Ptot
Yaw
Wy
Wz
Wx
A
Ax
W
Ay
Az
Roll
Pitch
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

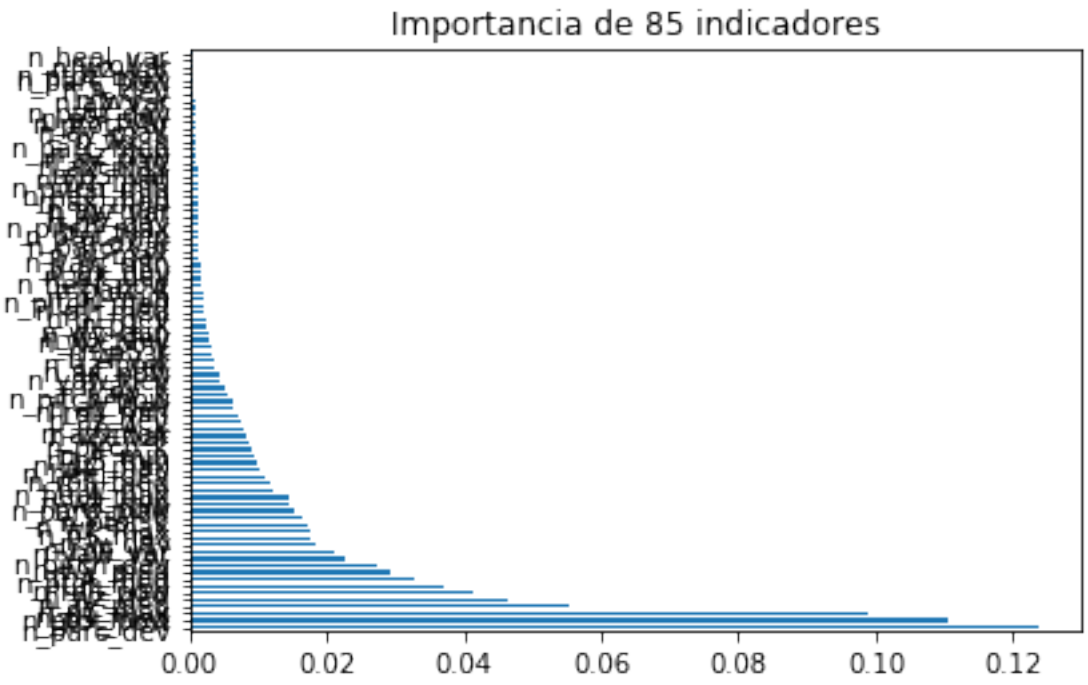

Importancia de 14 variables
P1
P5
Ptot
Yaw
Wy
Wz
Wx
A
Ax
W
Ay
Az
Roll
Pitch
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175


Importancia de 5 indicadores
n_ay_med
n_p1_max
n_p5_med
n_ptot_pow
n_parc_dev
0.00
0.02
0.04
0.06
0.08
0.10
0.12


Importancia de 25 indicadores
n_wy_max
n_heel_dev
n_roll_med
n_w_min
n_heel_max
n_wx_min
n_w_pow
n_parc_max
n_ptot_k
n_wz_max
n_p5_max
n_w_dev
n_roll_var
n_yaw_var
n_pitch_dev
n_wy_med
n_A_med
n_ptot_med
n_roll_pow
n_wz_dev
n_ay_med
n_p1_max
n_p5_med
n_ptot_pow
n_parc_dev
0.00
0.02
0.04
0.06
0.08
0.10
0.12


Importancia de 45 indicadores
0.00
0.02
0.04
0.06
0.08
0.10
0.12


Importancia de 65 indicadores
0.00
0.02
0.04
0.06
0.08
0.10
0.12


Importancia de 85 indicadores
0.00
0.02
0.04
0.06
0.08
0.10
0.12

1 *MÁQUINA DE ESTADOS*

A continuación se explican las partes del código generado para realizar el algoritmo de la máquina de estados: Importación de librerias, definición de funciones y programa principal.

1.1 Importación librerías

[1]:
```
"""
TFM_Caidas
@author: Rebeca Teresa Blanco
"""
from Lector import Lector, LoadTests
from Statistics import Statistics
import numpy as np
```

1.2 Definición de funciones

A continuación, se muestran las funcones utilizadas en el programa principal:

1.2.1 Sincronización de datos

Sincroniza los datos de ambos pies. Devuelve las señales de los datos en cruodo y sus derivadas

[2]:
```
from utils import sincronizaDatos
```

1.2.2 Estadística de indicadores

Calcula y devuelve la media y desvacioens de los vectores de indicadores.

[3]:
```
def statistics(k,max,min,med,desv,var,power):

    medias = [np.mean(k), np.mean(max),np.mean(min),np.mean(med),
              np.mean(desv),np.mean(var),np.mean(power)]
    desviaciones = [np.std(k), np.std(max), np.std(min),np.std(med),
            np.std(desv),np.std(var),np.std(power)]

    return medias
```

1.2.3 Cálculo indicadores

Calcula los indicadores de un vector de datos. Devuelve valores de los mismos (kurtosis, max,min, med, desv, var y potencia de señal)

[4]:
```
from utils import estadistica
```

1.2.4 Estado

Función en la que comparando la estadística de los indicadores de una ventana con un umbral (media del valor max. o medio de una variable). Devuelve el estado de de la ventana: reposo (0), movimiento (1), suelo (2), ambas (3).

[5]:
```
def estado(stat_a, stat_p, stat_roll, stat_pitch,stat_yaw, stat_w):

    """Valores umbral de las variables"""
    # Media y desviación de los valores máx. de la aceleración de las caídas.
    a_thres = 5.54607497298519
    a_desv = 0.7674998432650881

    # Media del valor medio del ángulo roll al final de la caída.
    roll_thres = 80

    # Media y desviación de los valores máx. de las variables
    #p_thres = 1700.3076923076924
    #p_desv = 768.1108077730397
    #roll_thres = 647.101584314192 son umbrales maximos
    #roll_desv = 135.69717828108097
    #pitch_thres = 13.520486221741178
    #pitch_desv = 54.53250942367654
    #yaw_thres = -9.135457149404836
    #yaw_desv = 16.023077403906655
    #w_thres = 0.03747536402370082
    #w_desv = 0.34590178099610824

    # Máximo de la aceleración
    max_a = stat_a[1]
    # Media de ángulo roll
    max_roll = stat_roll[2]

    # Máquina estados
    if max_a >= a_thres - a_desv and (max_roll < roll_thres or max_roll >␣
↪-roll_thres):
        salida = 2
    elif max_a < a_thres - a_desv and (max_roll > roll_thres or max_roll <␣
↪-roll_thres):
        salida = 1
    elif max_a < a_thres - a_desv and (max_roll < roll_thres or max_roll >␣
↪-roll_thres):
        salida = 0
    else:
        salida = 3

    return salida
```

1.2.5 Procesado de un ensayo

Función que procesa un ensayo mediante una ventana deslizante. Obtiene la estadística de cada ventana de tiempo a lo largo del ensayo y devuelve un vector de 1 y 0 (caida y no caida) según el valor de la estadistica de la ventana de tiempo

[6]:
```
def ProcessTest(lector, time, ini, fin, TD, TW):
    indices = []
    salida = []

    datos = sincronizaDatos(lector,ini,fin)

    A_aux = datos[0][3]
    ptot_aux = datos[0][8]
    roll_aux = datos[0][9]
    pitch_aux = datos[0][10]
    yaw_aux = datos[0][11]
    W_aux = datos[0][15]

    cont = 0

    for i in range(0, len(time), TD):
        A = A_aux[i:i + TW]
        ptot = ptot_aux[i:i + TW]
        roll = roll_aux[i:i + TW]
        pitch = pitch_aux[i:i + TW]
        yaw = yaw_aux[i:i + TW]
        W = W_aux[i:i + TW]
        stat_a = estadistica(A)
        stat_p = estadistica(ptot)
        stat_roll = estadistica(roll)
        stat_pitch = estadistica(pitch)
        stat_yaw = estadistica(yaw)
        stat_w = estadistica(W)
        if i==0:
            estadoFall = estado(stat_a, stat_p, stat_roll, stat_pitch,stat_yaw,␣
↪stat_w)
        else:
            estadoFall = estado(stat_a, stat_p, stat_roll, stat_pitch, stat_yaw,␣
↪stat_w)

        if estadoFall == 0:
            salida.append("Reposo")
        elif estadoFall == 1:
            salida.append("Suelo")
        elif estadoFall == 2:
            salida.append("Movimiento")
        else:
```

```
            salida.append("Ambas")

    caida = isFall(salida)

    return caida
```

1.2.6 isFall

Función que recorre el vector salida con todas las ventanas del ensayo y calcula el FRI (indice de riesgo de caida) en función del tipo de estado seguidos que hay en el mismo. Si es mayor a un umbral devuelve True, es decir, detecta caida

[7]:
```
def isFall(vector):
    FRI = 0
    FRI2 = 0

    for i in range(len(vector)-1):
        if vector[i] == "Movimiento" and vector[i+1] == "Movimiento":
            FRI +=1
        elif vector[i] == "Suelo" and vector[i + 1] == "Suelo":
            FRI2 +=1

    if FRI>=2 and FRI2>=2: #Mayor Exactitud, pero también mayor tasa de FP.␣
↪Menor Exactitud al aumentar FRI's --> Más restrictivo
        caida = True
    else:
        caida = False

    return caida
```

1.2.7 Obtención vectores salida

Dado el path del excel de ensayos, lee cada ensayo, lo procesa y detecta si es caida o no se obtienen dos vectores con la salida predicha y la salida real de cada ensayo. Devuelve los vectores de salida reales y predichos para cada ensayo.

[8]:
```
def getTargets(path_file_d, path_file_i, results, targets):
    # LOADERS (OBJECTS)
    loader_d = LoadTests()
    loader_i = LoadTests()

    loader_d.Load(path_file_d)
    loader_i.Load(path_file_i)

    # Path de los directorios de los ensayos para cada plantilla
    data_d = 'data_d/'
    data_i = 'data_i/'
```

```
# Definición de vectores de tiempo
time_d = []
time_i = []

#Bucle de lectura de ensayos
for i in range(len(loader_d.files)):

    # Inicializacion lectores
    lector_d = Lector()
    lector_i = Lector()

    # Carga de ficheros
    lector_d.lee(data_d + loader_d.files[i])
    lector_i.lee(data_i + loader_i.files[i])

    # Cambio de sistema de referencia plantillas
    lector_d.RightRef()
    lector_i.LeftRef()

    # DATA NORMALIZATION
    # lector_d.normalize()
    # lector_i.normalize()

    # Sincronización de vectores de datos
    if lector_d.timestamp_v[0] < lector_i.timestamp_v[0]:
        ini = lector_d.timestamp_v[0]
    else:
        ini = lector_i.timestamp_v[0]

    for j in range(len(lector_d.timestamp_v)):
        time_d.append((lector_d.timestamp_v[j] - ini) / 1000)

    for j in range(len(lector_i.timestamp_v)):
        time_i.append((lector_i.timestamp_v[j] - ini) / 1000)

    ini_d = 0
    ini_i = 0
    fin_d = len(lector_d.timestamp_v) - 1
    fin_i = len(lector_i.timestamp_v) - 1

    if time_d[0] < time_i[0]:
        while time_d[ini_d] < time_i[0]:
            ini_d += 1
    else:
        while time_i[ini_i] < time_d[0]:
            ini_i += 1
```

```
        if time_d[fin_d] > time_i[fin_i]:
            while time_d[fin_d] > time_i[fin_i]:
                fin_d -= 1
        else:
            while time_i[fin_i] > time_d[fin_d]:
                fin_i -= 1

        # Sincronización de los vectores de tiempo
        time_d = time_d[ini_d:fin_d]
        time_i = time_i[ini_i:fin_i]

        # Vector salida predicha
        valor = ProcessTest(lector_d, time_d, ini_d, fin_d,␣
↪step_size,window_size) #Procesado del ensayo-->Obtencón salida
        results.append(valor)

        # Vector salida real
        if lector_d.session_type == "BACKWARDS" or lector_d.session_type ==␣
↪"FORWARDS" or lector_d.session_type == "LATERAL_STAND" or \
                lector_d.session_type == "LATERAL_WALK" or lector_d.session_type␣
↪== "SITTING" :
            targets.append("Fall")
        else:
            targets.append("ADL")

        time_d.clear()
        time_i.clear()
```

1.2.8 Obtención de resultados

Recorre y compara los vectores salida predicha y salida real. Devuelve el nº de TP, FP, FN y TN.

[9]:
```
def conteo(results,targets):
    #Inicialización de contadores
    TP = 0 #Contador verdaderos positivos
    FP = 0 #Contador falsos positivos
    FN = 0 #Contador verdaderos negativos
    TN = 0 #Contador falsos negativos
    for i in range(len(results)):
        if results[i] == True and targets[i]=="Fall":
            TP += 1
        elif results[i] == True and targets[i]=="ADL":
            FP += 1
        elif results[i] == False and targets[i] == "Fall":
            FN +=1
```

```
        else:
            TN +=1
    return TP, FP, FN,TN
```

Muestra por pantalla los resultados

[10]:
```
def showResults(TP, FP,FN,TN, results, name):
    #Visualización de resultados por pantalla
    print("Resultados de ensayos " + name)
    exactitud = (TP+TN)*100/len(results)
    print("Verdaderos positivos " + str(TP))
    print("Falsos positivos " + str(FP))
    print("Falsos negativos " + str(FN))
    print("Verdaderos negativos " + str(TN))
    print("Exactitud " + str(exactitud) + "%")
```

1.3 Programa principal

[11]:
```
# Definición tiempos de ventana deslizante
TW = 3150  # ms
TD = 500  # ms
T = 20  # msl
# Tiempos en valores de nº de datos --> tiempo_ventana/periodo = nº de datos en␣
 ↪ventana
step_size = TD // T
window_size = TW // T
```

[12]:
```
#Paths de ensayos de caidas y ADL
#Fall paths
Backwards_d = 'Tests/Falls/Backwards_d.csv'
Backwards_i = 'Tests/Falls/Backwards_i.csv'
Forwards_d = 'Tests/Falls/Forwards_d.csv'
Forwards_i = 'Tests/Falls/Forwards_i.csv'
LateralStand_d = 'Tests/Falls/LateralStand_d.csv'
LateralStand_i = 'Tests/Falls/LateralStand_i.csv'
LateralWalk_d = 'Tests/Falls/LateralWalk_d.csv'
LateralWalk_i = 'Tests/Falls/LateralWalk_i.csv'
Sitting_d = 'Tests/Falls/Sitting_d.csv'
Sitting_i = 'Tests/Falls/Sitting_i.csv'
Falls_d = 'Tests/Falls/Falls_d.csv'
Falls_i = 'Tests/Falls/Falls_i.csv'

#ADL Paths
Lay_d = 'Tests/ADL/Lay/Lay_d.csv'
Lay_i = 'Tests/ADL/Lay/Lay_i.csv'
LayFP_d = 'Tests/ADL/LayFP/LayFP_d.csv'
```

```
LayFP_i = 'Tests/ADL/LayFP/LayFP_i.csv'
SitFP_d = 'Tests/ADL/SittingFP/SittingFP_d.csv'
SitFP_i = 'Tests/ADL/SittingFP/SittingFP_i.csv'
Walk_d = 'Tests/ADL/Walk/Walk_d.csv'
Walk_i = 'Tests/ADL/Walk/Walk_i.csv'
WalkFP_d = 'Tests/ADL/WalkFP/WalkFP_d.csv'
WalkFP_i = 'Tests/ADL/WalkFP/WalkFP_i.csv'
Stand_d = 'Tests/ADL/Stand/Stand_d.csv'
Stand_i = 'Tests/ADL/Stand/Stand_i.csv'
ADL_d = 'Tests/ADL/ADL_d.csv'
ADL_i = 'Tests/ADL/ADL_i.csv'
```

[13]:
```
#Definición de vectores de salida
results_back = []
targets_back = []
results_for = []
targets_for = []
results_LS = []
targets_LS = []
results_LW = []
targets_LW = []
results_sit = []
targets_sit = []
results_fall = []
targets_fall = []
results_adl = []
targets_adl = []
```

[14]:
```
#Procesado de ensayos --> Obtención salidas
getTargets(Backwards_d, Backwards_i, results_back, targets_back)
getTargets(Forwards_d, Forwards_i, results_for, targets_for)
getTargets(LateralStand_d, LateralStand_i, results_LS, targets_LS)
getTargets(LateralWalk_d, LateralWalk_i, results_LW, targets_LW)
getTargets(Sitting_d, Sitting_i, results_sit, targets_sit)
getTargets(Falls_d, Falls_i, results_fall, targets_fall)
getTargets(ADL_d, ADL_i, results_adl, targets_adl)
```

[15]:
```
#Vectores de salida
results_tot = np.concatenate((results_fall,results_adl))
targets_tot = np.concatenate((targets_fall,targets_adl))
```

[16]:
```
#Obtención resultados
TP, FP, FN, TN = conteo(results_tot,targets_tot)
TP_back, FP_back, FN_back, TN_back = conteo(results_back,targets_back)
TP_for, FP_for, FN_for, TN_for = conteo(results_for,targets_for)
TP_LS, FP_LS, FN_LS, TN_LS = conteo(results_LS,targets_LS)
TP_LW, FP_LW, FN_LW, TN_LW = conteo(results_LW,targets_LW)
```

```
TP_sit, FP_sit, FN_sit, TN_sit = conteo(results_sit,targets_sit)
```

[17]:
```
showResults(TP, FP, FN, TN, results_tot, "totales")
```

```
Resultados de ensayos totales
Verdaderos positivos 161
Falsos positivos 27
Falsos negativos 76
Verdaderos negativos 407
Exactitud 84.64977645305514%
```

[18]:
```
showResults(TP_back, FP_back, FN_back, TN_back, results_back, "caída hacia␣
↪atrás")
```

```
Resultados de ensayos caída hacia atrás
Verdaderos positivos 34
Falsos positivos 0
Falsos negativos 29
Verdaderos negativos 0
Exactitud 53.96825396825397%
```

[19]:
```
showResults(TP_for, FP_for, FN_for, TN_for, results_for, "caída hacia delante")
```

```
Resultados de ensayos caída hacia delante
Verdaderos positivos 98
Falsos positivos 0
Falsos negativos 33
Verdaderos negativos 0
Exactitud 74.80916030534351%
```

[20]:
```
showResults(TP_LS, FP_LS, FN_LS, TN_LS, results_LS, "caída lateral estado de␣
↪pie")
```

```
Resultados de ensayos caída lateral estado de pie
Verdaderos positivos 29
Falsos positivos 0
Falsos negativos 14
Verdaderos negativos 0
Exactitud 67.44186046511628%
```

[21]:
```
showResults(TP_LW, FP_LW, FN_LW, TN_LW, results_LW, "caída lateral caminando")
```

```
Resultados de ensayos caída lateral caminando
Verdaderos positivos 0
Falsos positivos 26
Falsos negativos 0
Verdaderos negativos 16
Exactitud 38.095238095238095%
```

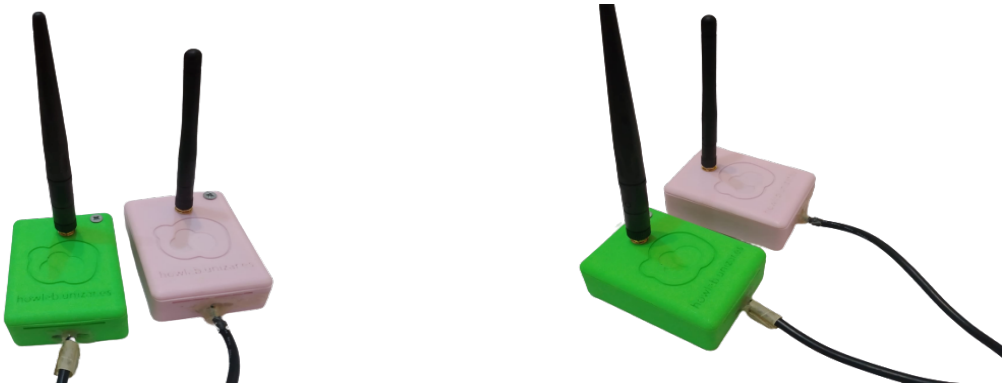
[22]:
```
showResults(TP_sit, FP_sit, FN_sit, TN_sit, results_sit, "caida sentandose")
```

```
Resultados de ensayos caida sentandose
Verdaderos positivos 13
Falsos positivos 0
Falsos negativos 45
Verdaderos negativos 6
Exactitud 29.6875%
```

Anexos B

Detalles técnicos

B.1. Receptores *bluetooth* HOWlab



(a) Vista frontal receptores *bluetooth*

(b) Vista lateral receptores *bluetooth*

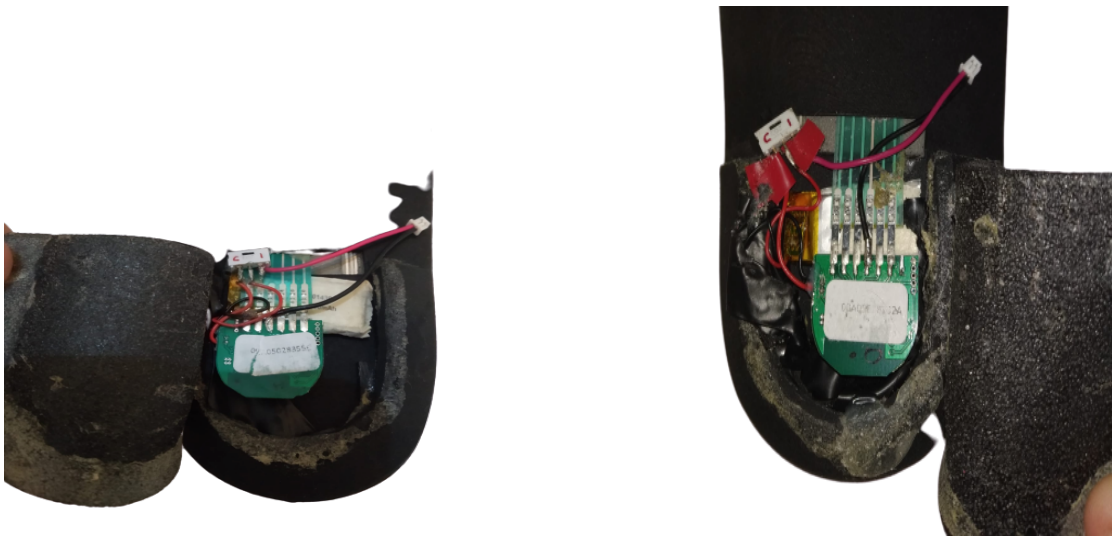
Figura B.1: Receptores *bluetooth*

B.2. Plantillas Podoactiva

(a) Vista superior plantillas

(b) Vista inferior plantillas

Figura B.2: Plantillas



(a) Electrónica plantillas izquierda

(b) Electrónica plantilla derecha

Figura B.3: Electrónica de plantillas

Anexos C

Análisis variables

Tabla C.1: SNR de aceleraciones

SNR	Ax	Ay	Az	A
Falls	-4,75332573	-18,5403909	-7,22685847	67,5864378
ADL	37,3506669	-21,4665794	-41,8208444	56,3961116

Tabla C.2: SNR de presiones

SNR	P1	P5	Parcs	Pheel	Ptot
Falls	0,41356201	inf	nan	inf	0,74093905
ADL	0,63550385	inf	nan	inf	1,07157059

Tabla C.3: SNR de ángulos

SNR	Roll	Pitch	Yaw
Falls	138,01657	25,5949774	8,1552397
ADL	209,523488	-144,414127	-3,31050501

Tabla C.4: SNR de velocidades angulares

SNR	Wx	Wy	Wz	W
Falls	2,06024682	-10,0033234	8,14443953	85,9926463
ADL	-0,9714593	-3,12920625	-3,31780799	40,768788

Figura C.1: Gráficas de variables de todas las caídas

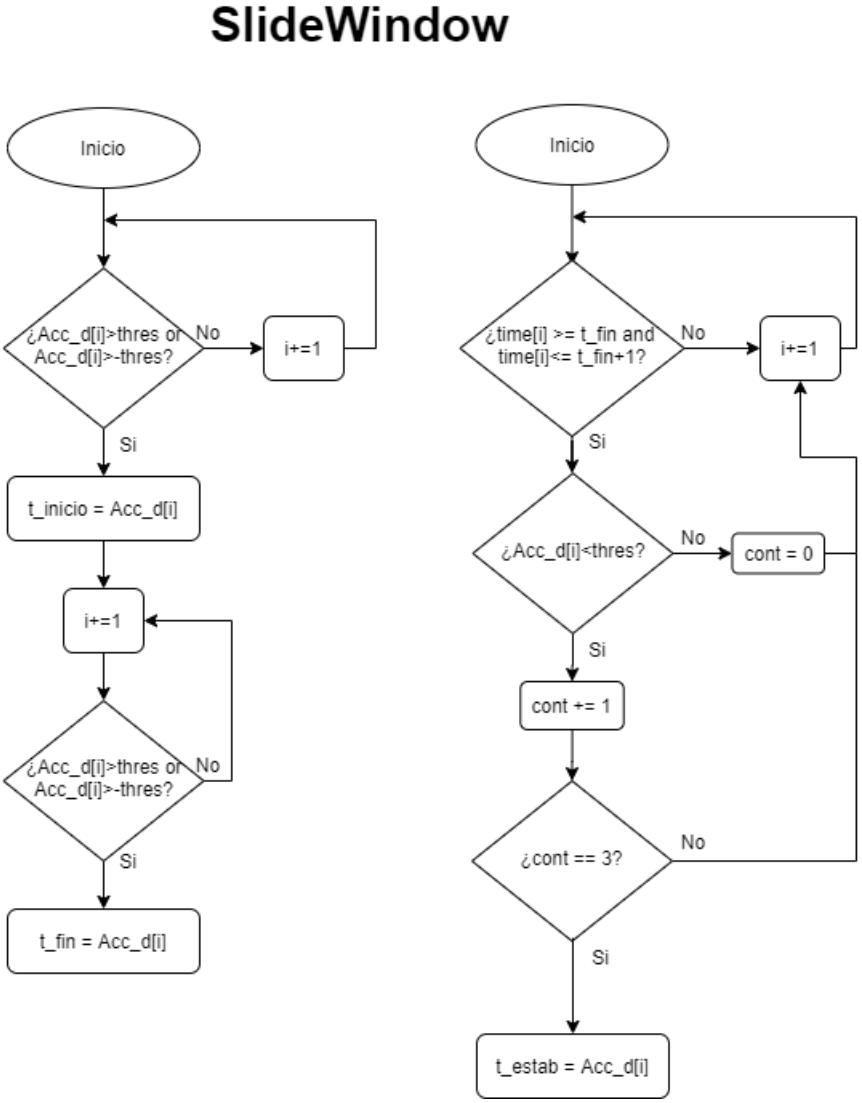


Figura C.2: Diagrama flujo para el cálculo de tiempos de ventana deslizante

En la figura C.2 aparecen los dos bucles que componen la función del cálculo del tiempo de ventana. Mediante el primero de ellos se obtienen los tiempos de inicio y fin de la caída y, mediante el segundo se obtiene el tiempo de estabilización y, por tanto el tiempo de deslizamiento. La variable *cont* se utiliza para saber cuándo se ha estabilizado la señal y, por tanto, ha terminado la caída, cuando durante 60ms (3 ciclos) el valor está por debajo de cierto umbral.

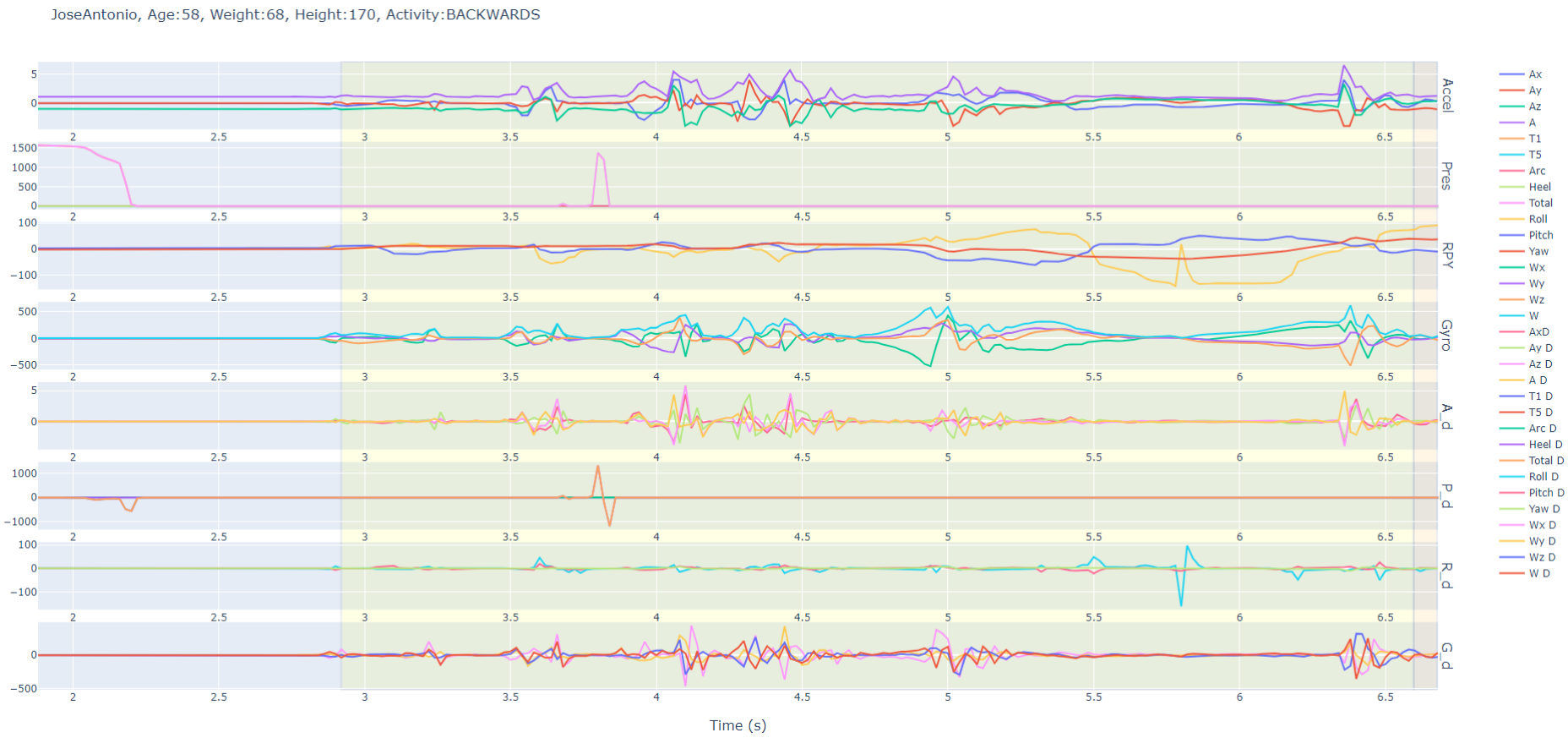


Figura C.3: Ejemplo de gráfica de datos en crudo de las variables

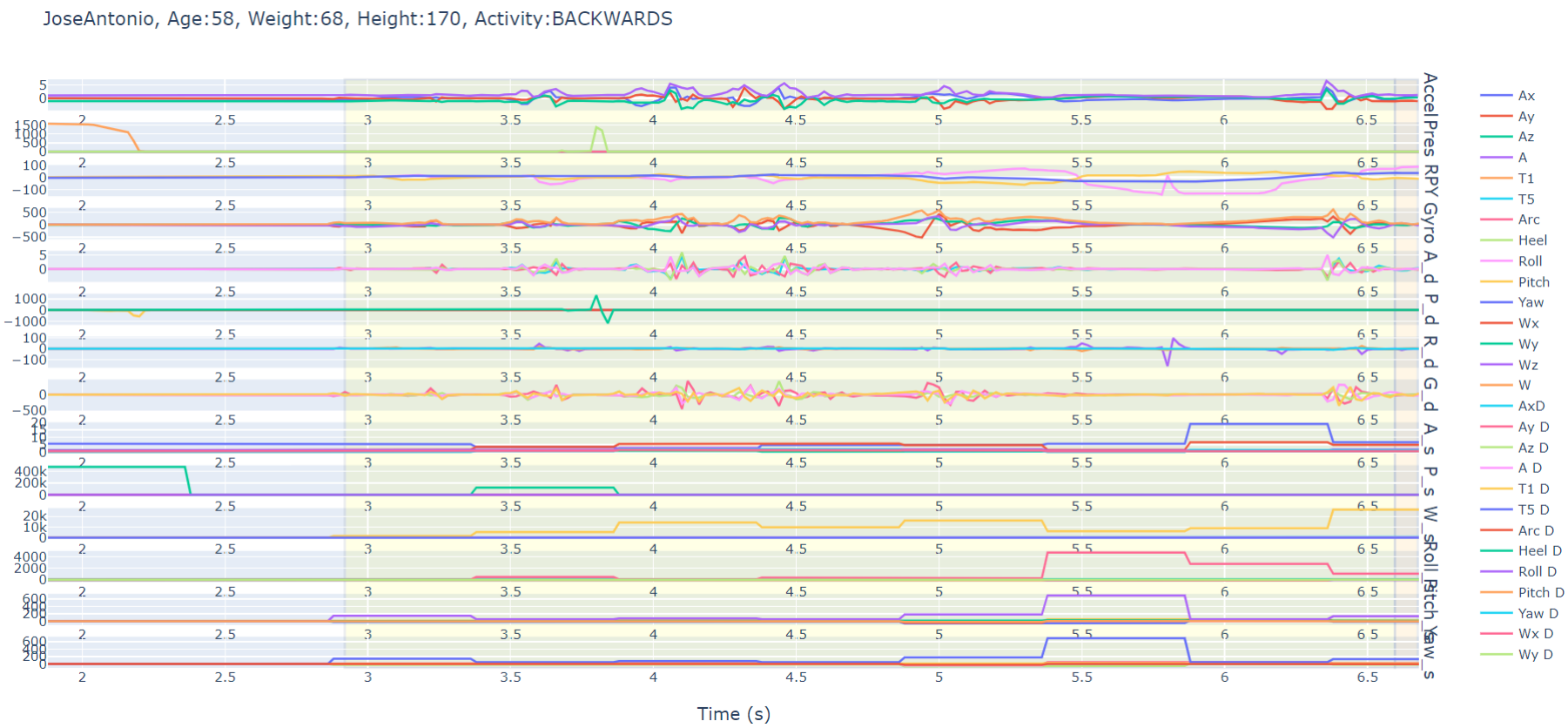


Figura C.4: Ejemplo de gráfica de indicadores de las variables

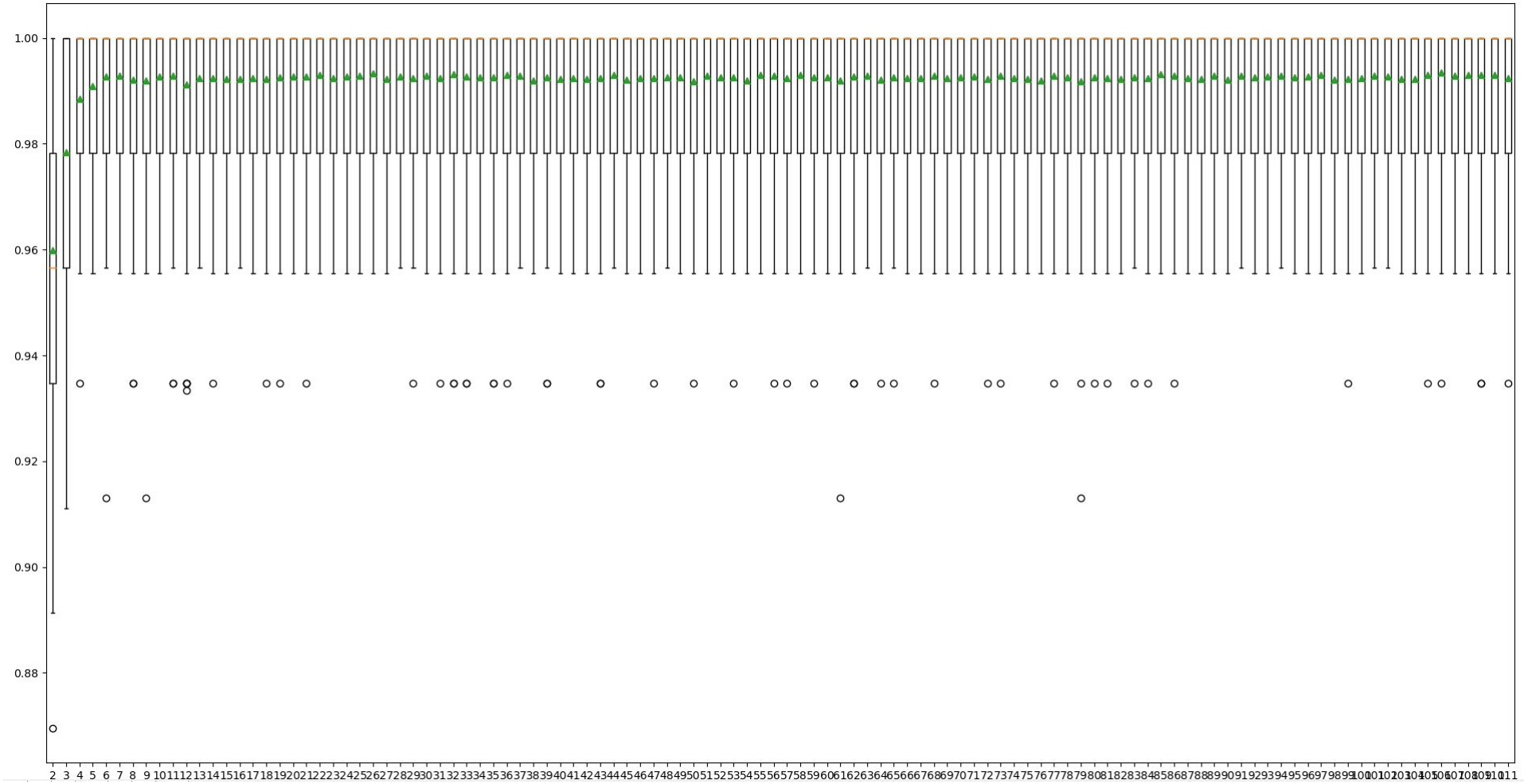


Figura C.5: Diagramas de caja para cada variable

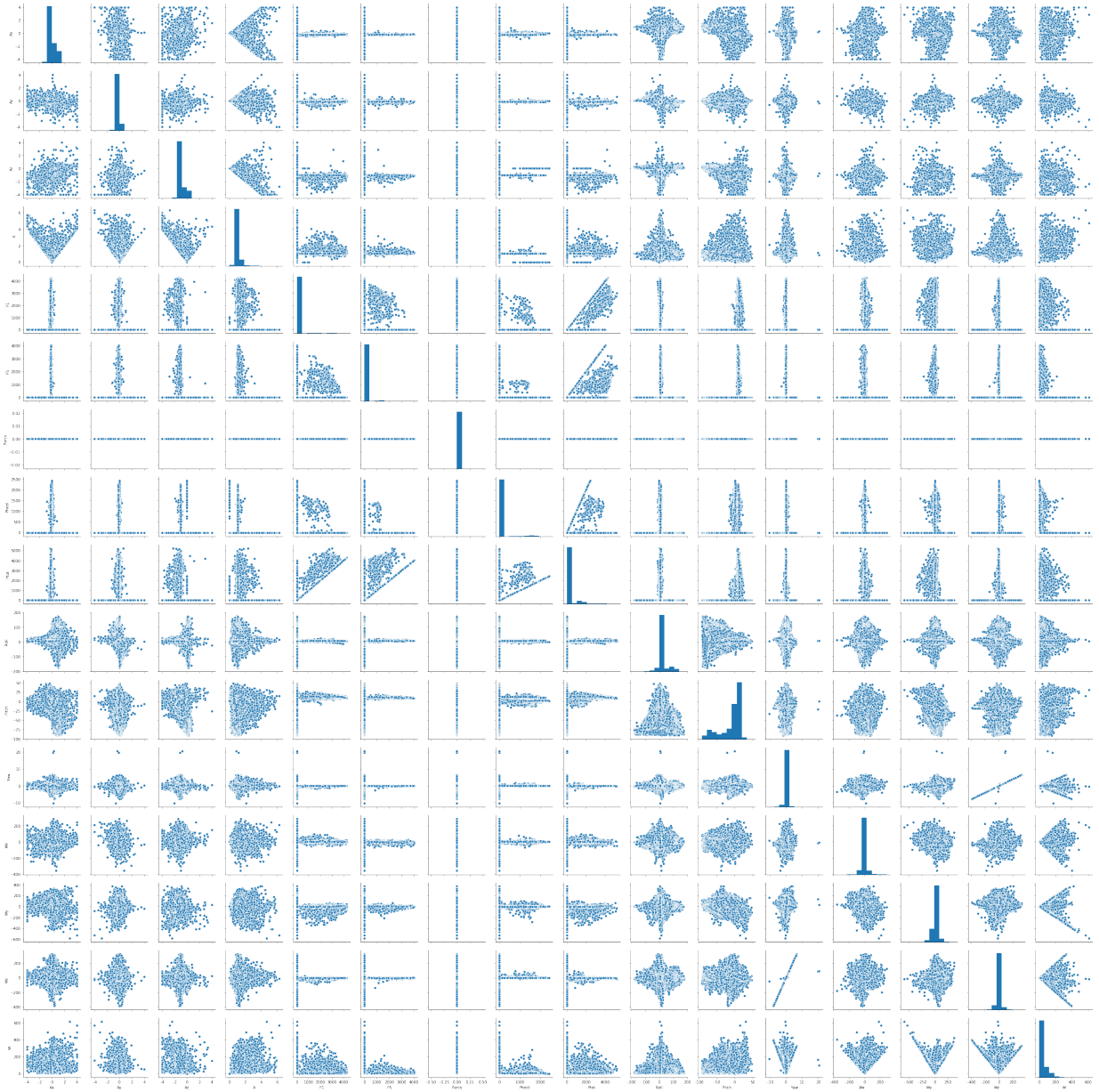


Figura C.6: Correlaciones de variables en ADL

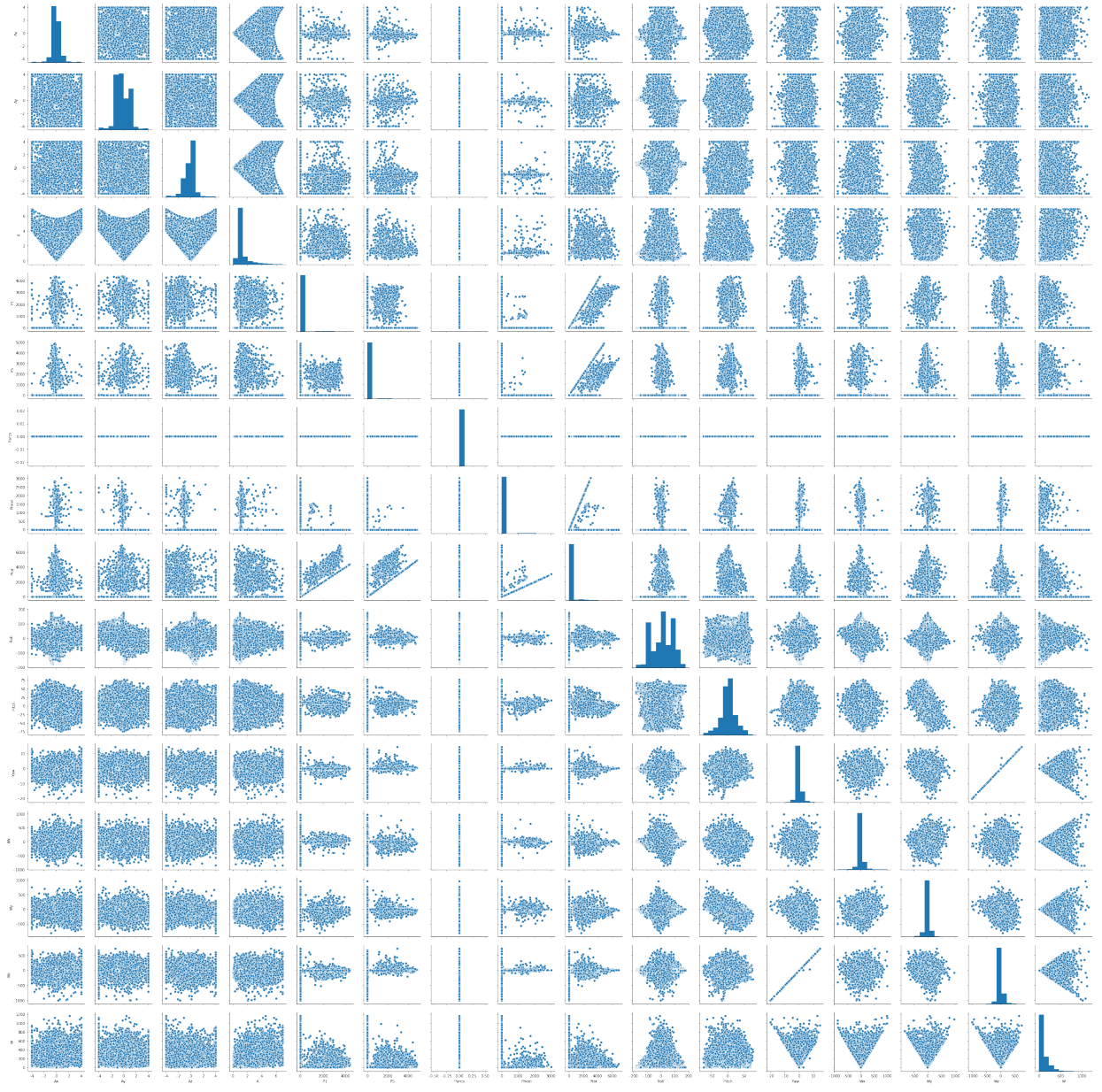


Figura C.7: Correlaciones de variables en caídas

Anexos D

Tablas de resultados

Tabla D.1: Resultados modelo sujeto 1

Resultados	Modelo	TP (*True positives*)	TN (*True negatives*)	FN (*False negatives*)	FP (*False positives*)	Exactitud total(%)
Sujeto 1 (S1)	SVC_rbf.pkl	18868	77365	3607	1528	94,934
	SVC_poly.pkl	18117	77662	4358	1231	94,486
	SVC_sig.pkl	13258	71529	9217	7364	83,642
	MLP.pkl	19195	76786	3280	2107	94,685
	NuSVC.pkl	18451	77673	4024	1220	94,826
	SVC_rbf_pca.pkl	19543	69148	2932	9745	87,494
	SVC_poly_pca.pkl	17970	70510	4505	8383	87,285
	SVC_sig_pca.pkl	11530	63913	10945	14980	74,424
	MLP_pca.pkl	19859	64715	2616	14178	83,432
	NuSVC_pca.pkl	18955	72509	3520	6384	90,229
	n_SVC_rbf.pkl	20	522	138	55	73,741
	n_SVC_poly.pkl	0	577	158	0	78,503
	n_SVC_sig.pkl	42	389	116	188	58,639
	n_MLP.pkl	78	376	80	201	61,768
	n_NuSVC.pkl	10	548	148	29	75,918

Tabla D.2: Resultados modelo sujeto 2

Resultados	Modelo	TP (*True positives*)	TN (*True negatives*)	FN (*False negatives*)	FP (*False positives*)	Exactitud total(%)
Sujeto 2 (S2)	SVC_rbf.pkl	18810	77500	3665	1393	95,010
	SVC_poly.pkl	17700	78132	4775	761	94,538
	SVC_sig.pkl	12916	69399	9559	9494	81,204
	MLP.pkl	19405	74899	3070	3994	93,031
	NuSVC.pkl	19264	76641	3211	2252	94,610
	SVC_rbf_pca.pkl	18786	72438	3689	6455	89,992
	SVC_poly_pca.pkl	15127	73386	7348	5507	87,318
	SVC_sig_pca.pkl	11820	63875	10655	15018	74,673
	MLP_pca.pkl	18949	64113	3526	14780	81,941
	NuSVC_pca.pkl	19375	65852	3100	13041	84,076
	n_SVC_rbf.pkl	3	562	155	15	76,870
	n_SVC_poly.pkl	1	576	157	1	78,503
	n_SVC_sig.pkl	120	371	38	206	66,802
	n_MLP.pkl	104	501	54	76	82,312
	n_NuSVC.pkl	2	570	156	7	77,823

Tabla D.3: Resultados de modelo sujeto 1, test sujeto 2

Resultados	Modelo	TP (*True positives*)	TN (*True negatives*)	FN (*False negatives*)	FP (*False positives*)	Exactitud total(%)
M: S1, T: S2	SVC_rbf.pkl	7909	32824	2304	1167	92,147
	SVC_poly.pkl	7482	33133	2731	858	91,880
	SVC_sig.pkl	5601	31149	4612	2842	83,137
	MLP.pkl	7902	32635	2311	1356	91,704
	NuSVC.pkl	7756	33098	2457	893	92,421
	SVC_rbf_pca.pkl	8329	29399	1884	4592	85,349
	SVC_poly_pca.pkl	6812	28889	3401	5102	80,764
	SVC_sig_pca.pkl	5655	23375	4558	10616	65,672
	MLP_pca.pkl	7657	27079	2556	6912	78,581
	NuSVC_pca.pkl	8146	30137	2067	3854	86,605
	n_SVC_rbf.pkl	41	267	27	3	91,124
	n_SVC_poly.pkl	0	270	68	0	79,881
	n_SVC_sig.pkl	51	238	17	32	85,502
	n_MLP.pkl	60	257	8	13	93,786
	n_NuSVC.pkl	32	268	36	2	88,757
	n_SVC_rbf_pca.pkl	42	240	26	30	83,431
	n_SVC_poly_pca.pkl	36	262	32	8	88,165
	n_SVC_sig_pca.pkl	54	198	14	72	74,556
	n_MLP_pca.pkl	46	225	22	45	80,177
	n_NuSVC_pca.pkl	41	250	27	20	86,094

Tabla D.4: Modelo sujeto 2, test sujeto 1

Resultados	Modelo	TP (*True positives*)	TN (*True negatives*)	FN (*False negatives*)	FP (*False positives*)	Exactitud total(%)
M: S2, T: S1	SVC_rbf.pkl	10731	43155	1643	1636	94,263
	SVC_poly.pkl	9923	44016	2451	775	94,356
	SVC_sig.pkl	7249	39900	5125	4891	82,478
	MLP.pkl	10668	41311	1706	3480	90,928
	NuSVC.pkl	10860	42368	1514	2423	93,112
	SVC_rbf_pca.pkl	10612	38797	1762	5994	86,432
	SVC_poly_pca.pkl	6402	41255	5972	3536	83,367
	SVC_sig_pca.pkl	6216	34506	6158	10285	71,235
	MLP_pca.pkl	10096	34303	2278	10488	77,668
	NuSVC_pca.pkl	10607	34986	1767	9805	79,756
	n_SVC_rbf.pkl	7	310	70	11	79,648
	n_SVC_poly.pkl	0	320	77	1	80,402
	n_SVC_sig.pkl	53	206	24	115	65,075
	n_MLP.pkl	42	275	35	46	79,648
	n_NuSVC.pkl	8	310	69	11	79,899
	n_SVC_rbf_pca.pkl	53	280	24	41	83,668
	n_SVC_poly_pca.pkl	50	303	27	18	88,693
	n_SVC_sig_pca.pkl	41	225	36	96	66,834
	n_MLP_pca.pkl	53	251	24	70	76,381
	n_NuSVC_pca.pkl	52	284	25	37	84,422

Tabla D.5: Resultados modelos RFE

Resultados	Modelo	TP (*True positives*)	TN (*True negatives*)	FN (*False negatives*)	FP (*False positives*)	Exactitud total(%)
Total RFE	SVC_rbf.pkl	18777	78599	3698	294	96,061
	SVC_poly.pkl	17805	78471	4670	422	94,976
	SVC_sig.pkl	13688	70112	8787	8781	82,669
	MLP.pkl	19193	78400	3282	493	96,275
	NuSVC.pkl	18849	78608	3626	285	96,141
	SVC_rbf_pca.pkl	18564	78558	3911	335	95,811
	SVC_poly_pca.pkl	17641	78423	4834	470	94,767
	SVC_sig_pca.pkl	13549	69940	8926	8953	82,362
	MLP_pca.pkl	19279	78011	3196	882	95,977
	NuSVC_pca.pkl	18804	78585	3671	308	96,074
	n_SVC_rbf.pkl	142	572	10	11	97,142
	n_SVC_poly.pkl	125	576	27	7	95,374
	n_SVC_sig.pkl	100	520	52	63	84,353
	n_MLP.pkl	144	571	8	12	97,278
	n_NuSVC.pkl	140	572	12	11	96,870
	n_SVC_rbf_pca.pkl	142	572	10	11	97,142
	n_SVC_poly_pca.pkl	125	576	27	7	95,374
	n_SVC_sig_pca.pkl	100	520	52	63	84,353
	n_MLP_pca.pkl	144	572	8	11	97,414
	n_NuSVC_pca.pkl	140	572	12	11	96,870

Tabla D.6: Resultados modelos FI

Resultados	Modelo	TP (*True positives*)	TN (*True negatives*)	FN (*False negatives*)	FP (*False positives*)	Exactitud total(%)
Total FI	SVC_rbf.pkl	18777	78599	3698	294	96,061
	SVC_poly.pkl	17805	78471	4670	422	94,976
	SVC_sig.pkl	13688	70112	8787	8781	82,669
	MLP.pkl	19193	78400	3282	493	96,275
	NuSVC.pkl	18849	78608	3626	285	96,141
	SVC_rbf_pca.pkl	18564	78558	3911	335	95,811
	SVC_poly_pca.pkl	17641	78423	4834	470	94,767
	SVC_sig_pca.pkl	13549	69940	8926	8953	82,362
	MLP_pca.pkl	19279	78011	3196	882	95,977
	NuSVC_pca.pkl	18804	78585	3671	308	96,074
	n_SVC_rbf.pkl	145	574	13	3	97,823
	n_SVC_poly.pkl	138	575	20	2	97,006
	n_SVC_sig.pkl	113	538	45	39	88,571
	n_MLP.pkl	147	574	11	3	98,095
	n_NuSVC.pkl	143	575	15	2	97,687
	n_SVC_rbf_pca.pkl	145	574	13	3	97,823
	n_SVC_poly_pca.pkl	140	576	18	1	97,414
	n_SVC_sig_pca.pkl	109	525	49	52	86,258
	n_MLP_pca.pkl	147	573	11	4	97,959
	n_NuSVC_pca.pkl	145	574	13	3	97,823