**ANEXO 2**

**PRONOSTICO**

|  |
| --- |
| +\*In[ ]:\*+[source, ipython3]  ----  !conda env list  ----  +\*In[ ]:\*+  [source, ipython3]  ----  import warnings  import numpy as np  import pandas as pd  from math import sqrt  import matplotlib.pyplot as plt  from numpy import vectorize  from numpy import concatenate  from sklearn.metrics import mean\_squared\_error, explained\_variance\_score, max\_error, mean\_absolute\_error, mean\_squared\_log\_error, median\_absolute\_error, mean\_absolute\_percentage\_error, r2\_score, mean\_poisson\_deviance, mean\_gamma\_deviance, mean\_tweedie\_deviance  from tensorflow.keras.models import Sequential  from tensorflow.keras import optimizers  from tensorflow.keras.layers import Conv1D, MaxPooling1D  from tensorflow.keras.layers import Dense, LSTM, RepeatVector, TimeDistributed, Flatten  from numpy.random import seed  import tensorflow as tf  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Función para escalar los datos, recibe dato a escalar (vector), minimo y maximo  def escala(x,mi,ma):  return (x-mi)/(ma-mi)  escalador = vectorize(escala)  #Ejemplo de uso  # values[:,0]=escalador(values[:,0],0,300)  def Reescala(x,mi,ma):  return x\*(ma-mi)+mi  Reescalador = vectorize(Reescala)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  def evalua\_error(y\_re, y\_pr):  y\_re1 = y\_re.reshape((len(y\_re)))  y\_pr1 = y\_pr.reshape((len(y\_re)))  y\_re\_mod = y\_re1  y\_pr\_mod = y\_pr1    y\_re\_mod[y\_re\_mod==0]=1  y\_pr\_mod[y\_pr\_mod==0]=1  MAD = 0  CFE = 0  MSE = 0  MAPE = 0  for i in range (0,len(x\_pron)):  CFE = CFE + (y\_re1[i] - y\_pr1[i])  MAD = MAD + abs(y\_re1[i] - y\_pr1[i])  MSE = MSE + (y\_re1[i] - y\_pr1[i])\*(y\_re1[i] - y\_pr1[i])  MAPE = MAPE + 100\*(y\_re\_mod[i] - y\_pr\_mod[i])/y\_re\_mod[i]  MAD = MAD/len(y\_re1)  MSE = MSE/len(y\_re1)  RMSE = np.sqrt(MSE)  MAPE = MAPE/len(y\_re1)  VARIANZA\_EXPLICADA = explained\_variance\_score(y\_re, y\_pr)  ERROR\_RESIDUAL\_MAXIMO = max\_error(y\_re, y\_pr)  MEDIA\_ERROR\_ABSOLUTO = mean\_absolute\_error(y\_re, y\_pr)    ERROR\_CUADRATICO\_MEDIO = mean\_squared\_error(y\_re, y\_pr)  ERROR\_CUADRATICO\_LOGARITMICO = 1 #mean\_squared\_log\_error(y\_re\_mod, y\_pr\_mod)  MEDIA\_ERROR\_ABSOLUTO = median\_absolute\_error(y\_re, y\_pr)  ERROR\_PORCENTUAL\_ABSOLUTO = mean\_absolute\_percentage\_error(y\_re, y\_pr)  COEFICIENTE\_DETERMINACION\_R2 = r2\_score(y\_re, y\_pr)  DESVIACION\_POISSON = 1 #mean\_poisson\_deviance(y\_re, y\_pr)  DESVIACION\_GAMMA = 1 #mean\_gamma\_deviance(y\_re\_mod, y\_pr\_mod)  DESVIACION\_TWEEDIE = 1 #mean\_tweedie\_deviance(y\_re, y\_pr)  return [CFE, MAD, MSE, RMSE, MAPE, VARIANZA\_EXPLICADA,  ERROR\_RESIDUAL\_MAXIMO, MEDIA\_ERROR\_ABSOLUTO, ERROR\_CUADRATICO\_MEDIO,  ERROR\_CUADRATICO\_LOGARITMICO, MEDIA\_ERROR\_ABSOLUTO, ERROR\_PORCENTUAL\_ABSOLUTO,  COEFICIENTE\_DETERMINACION\_R2, DESVIACION\_POISSON, DESVIACION\_GAMMA, DESVIACION\_TWEEDIE]  ----  +\*In[ ]:\*+  [source, ipython3]  ----  Ciudad = 'CALI'  PATH = 'D:/DENGUE\_CODIGO/' + Ciudad + '/RESULTADOS/'  csv\_path\_x = PATH + 'Set\_datos\_X10.csv'  csv\_path\_y = PATH + 'Set\_datos\_Y10.csv'  dx = pd.read\_csv(csv\_path\_x, index\_col=False)  dx = dx.drop(dx.columns[[0]], axis='columns')  dy = pd.read\_csv(csv\_path\_y, index\_col=False)  dy = dy.drop(dy.columns[[0]], axis='columns')  dx.head()  valx = dx.values  valy = dy.values  # print(valx)  # print(valy)  semanas\_pronostico = 8  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Crear una matriz con los limites para el esclalado de los datos  limites=np.array(([np.amin(valx[:,0])],[np.amax(valx[:,0])]))  for i in range(1,len(valx[1,:])):  limites=np.insert(limites, limites.shape[1], [np.amin(valx[:,i]), np.amax(valx[:,i])], axis=1)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # ventana\_pronostico = 2  x = np.zeros((len(valx), len(valx[0])))  for i in range(0,len(valx[1,:])):  x[:,i]=escalador(valx[:,i],limites[0][i],limites[1][i])  y = escalador(valy,np.amin(valy),np.amax(valy))  y= np.array(y).reshape(-1)  valy= np.array(valy).reshape(-1)  print(np.amin(valy), np.amax(valy))  #y = y[1:]  #x= x[:-1,:]  print(x.shape)  # print(x)  print(y.shape)  # print(y)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Numero de datos para el pronostico, se tomaran los ultimos  ventana = 8  total = len(valy) - ventana  n\_pronostico = total-int(0.95\*total)  #Numero de datos para el entrenamiento  n\_train = int(0.7\*total)  #Numero de datos para test  n\_test = total - n\_train - n\_pronostico  #print("Cantidad total de datos = ",len(endog\_y),"\nNumero de datos para pronostico = ", n\_pronostico,"\nNumero de datos para entrenamiento = ",n\_train,"\nNumero de datos para test = ",n\_test)  print(n\_pronostico,n\_train,n\_test)  shift=0  x\_train = x[0:n\_train,:]  x\_test = x[n\_train:n\_train+n\_test,:]  x\_pron = x[n\_train+n\_test:n\_train+n\_test+n\_pronostico,:]  y\_train = y[:n\_train+shift]  y\_test = y[n\_train+shift:n\_train+n\_test+shift]  y\_test\_inv = valy[n\_train+shift:n\_train+n\_test+shift]  y\_pron = y[n\_train+n\_test+shift:n\_train+n\_test+n\_pronostico+shift]  y\_pron\_inv = valy[n\_train+n\_test+shift:n\_train+n\_test+n\_pronostico+shift]  print(x\_train.shape,x\_test.shape,x\_pron.shape,y\_train.shape,y\_test.shape,y\_pron.shape)  # print(x\_train)  # # reshape input to be 3D [samples, timesteps, features]  x\_trainMLP = x\_train  y\_trainMLP = y\_train  x\_testMLP = x\_test  y\_testMLP = y\_test  x\_pronMLP = x\_pron  y\_pronMLP = y\_pron  x\_train = x\_train.reshape((x\_train.shape[0], 1, x\_train.shape[1]))  # test\_X = test\_X.reshape((test\_X.shape[0], 1, test\_X.shape[1]))  x\_test = x\_test.reshape((x\_test.shape[0], 1, x\_test.shape[1]))  x\_pron = x\_pron.reshape((x\_pron.shape[0], 1, x\_pron.shape[1]))  Er = np.zeros([9,16])  Er\_t = ['CFE', 'MAD', 'MSE', 'RMSE', 'MAPE', 'Varianza explicada', 'Error residual máximo', 'Error absoluto medio', 'Error cuadrático medio',  'Error cuadrático logarítmico medio', 'Mediana del error absoluto', 'Error porcentual absoluto medio', 'R^2 (Coeficiente de determinación)',  'Desviación de Poisson media', 'Desviación Gamma media', 'Desviación Tweedie media']  ----  +\*In[ ]:\*+  [source, ipython3]  ----  from itertools import islice  def window(seq, n=2):  "Returns a sliding window (of width n) over data from the iterable"  " s -> (s0,s1,...s[n-1]), (s1,s2,...,sn), ... "  it = iter(seq)  result = tuple(islice(it, n))  if len(result) == n:  yield result  for elem in it:  result = result[1:] + (elem,)  yield result  #[print(list(i)) for i in list(window(y\_pron\_inv,2))];  ----  +\*In[ ]:\*+  [source, ipython3]  ----  def data\_gen(shift=2, cnn=False, mix=False):  total = len(valy) - shift  n\_pronostico = shift  #Numero de datos para el entrenamiento  n\_train = int(0.7\*total)  #Numero de datos para test  n\_test = total - n\_train - n\_pronostico  #print("Cantidad total de datos = ",len(endog\_y),"\nNumero de datos para pronostico = ", n\_pronostico,"\nNumero de datos para entrenamiento = ",n\_train,"\nNumero de datos para test = ",n\_test)  #print(n\_pronostico,n\_train,n\_test)  y\_win= np.array(list(window(y,shift)))  valy\_win =np.array(list(window(valy,shift)))  x\_train = x[0:n\_train,:]  x\_test = x[n\_train:n\_train+n\_test,:]  x\_pron = x[n\_train+n\_test:n\_train+n\_test+n\_pronostico,:]  y\_train = y\_win[:n\_train,:]  y\_test = y\_win[n\_train:n\_train+n\_test,:]  y\_test\_inv = valy\_win[n\_train:n\_train+n\_test]  y\_pron = y\_win[n\_train+n\_test:n\_train+n\_test+n\_pronostico]  y\_pron\_inv = valy\_win[n\_train+n\_test:n\_train+n\_test+n\_pronostico]  #print(x\_train.shape,x\_test.shape,x\_pron.shape,y\_train.shape,y\_test.shape,y\_pron.shape)  # print(x\_train)  # print('\n\n','y','\n\n')  # print(y\_train)        x\_train = x\_train.reshape((x\_train.shape[0], 1, x\_train.shape[1]))  # test\_X = test\_X.reshape((test\_X.shape[0], 1, test\_X.shape[1]))  x\_test = x\_test.reshape((x\_test.shape[0], 1, x\_test.shape[1]))  x\_pron = x\_pron.reshape((x\_pron.shape[0], 1, x\_pron.shape[1]))      if cnn:  x\_train = x\_train.reshape((x\_train.shape[0], x\_train.shape[-1], 1))  x\_test = x\_test.reshape((x\_test.shape[0], x\_test.shape[-1], 1))  x\_pron = x\_pron.reshape((x\_pron.shape[0], x\_pron.shape[-1], 1))    if mix:    x\_train = x\_train.reshape((x\_train.shape[0], x\_train.shape[-1], 1))  x\_test = x\_test.reshape((x\_test.shape[0], x\_test.shape[-1], 1))  x\_pron = x\_pron.reshape((x\_pron.shape[0], x\_pron.shape[-1], 1))  subsequences = 2  timesteps = x\_train.shape[1]//subsequences  x\_train = x\_train.reshape((x\_train.shape[0], subsequences, timesteps, 1))  x\_test = x\_test.reshape((x\_test.shape[0], subsequences, timesteps, 1))  x\_pron = x\_pron.reshape((x\_pron.shape[0], subsequences, timesteps, 1))    return x\_train,x\_test,x\_pron,y\_train,y\_test,y\_test\_inv, y\_pron, y\_pron\_inv  # # # reshape input to be 3D [samples, timesteps, features]  # x\_trainMLP = x\_train  # y\_trainMLP = y\_train  # x\_testMLP = x\_test  # y\_testMLP = y\_test  # x\_pronMLP = x\_pron  # y\_pronMLP = y\_pron  # x\_train = x\_train.reshape((x\_train.shape[0], 1, x\_train.shape[1]))  # # test\_X = test\_X.reshape((test\_X.shape[0], 1, test\_X.shape[1]))  # x\_test = x\_test.reshape((x\_test.shape[0], 1, x\_test.shape[1]))  # x\_pron = x\_pron.reshape((x\_pron.shape[0], 1, x\_pron.shape[1]))  # Er = np.zeros([9,16])  # Er\_t = ['CFE', 'MAD', 'MSE', 'RMSE', 'MAPE', 'Varianza explicada', 'Error residual máximo', 'Error absoluto medio', 'Error cuadrático medio',  # 'Error cuadrático logarítmico medio', 'Mediana del error absoluto', 'Error porcentual absoluto medio', 'R^2 (Coeficiente de determinación)',  # 'Desviación de Poisson media', 'Desviación Gamma media', 'Desviación Tweedie media']  ----  +\*In[ ]:\*+  [source, ipython3]  ----  def cnn\_lstm\_win(shift=2):    x\_train,x\_test,x\_pron,y\_train,y\_test,y\_test\_inv, y\_pron, y\_pron\_inv = data\_gen(shift, cnn=False, mix=True)  seed1 = 3  np.random.seed(seed1)  epochs1 = 200  batch = 1  lr = 0.00003  adam = optimizers.Adam(lr)  tf.random.set\_seed(seed1)    model\_CNN\_LSTM = Sequential()  model\_CNN\_LSTM.add(TimeDistributed(Conv1D(filters=64, kernel\_size=1, activation='relu'),  input\_shape=(None, x\_train.shape[2], x\_train.shape[3])))  model\_CNN\_LSTM.add(TimeDistributed(MaxPooling1D(pool\_size=2)))  model\_CNN\_LSTM.add(TimeDistributed(Flatten()))  model\_CNN\_LSTM.add(LSTM(150, activation='relu'))  model\_CNN\_LSTM.add(Dense(shift))  model\_CNN\_LSTM.compile(loss='mse', optimizer=adam)          # fit network  CNN\_LSTM\_history = model\_CNN\_LSTM.fit(x\_train, y\_train,  validation\_data=(x\_test, y\_test), epochs=epochs1, verbose=0)  scores = model\_CNN\_LSTM.evaluate(x\_train, y\_train)  print("\n%s: %.2f%%" % (model\_CNN\_LSTM.metrics\_names, scores\*100))  # Prediccion con datos de Pronostico - Modelo CNN  yhatP\_CNN\_LSTM = model\_CNN\_LSTM.predict(x\_pron)  yhatP\_CNN\_LSTM = np.reshape(yhatP\_CNN\_LSTM, (yhatP\_CNN\_LSTM.shape[0],shift))  inv\_yhatP\_CNN\_LSTM = Reescala(yhatP\_CNN\_LSTM, np.amin(valy), np.amax(valy))    print(np.amin(valy), np.amax(valy))  rmseCNN\_LSTM = sqrt(mean\_squared\_error(y\_pron\_inv, inv\_yhatP\_CNN\_LSTM))  print('Test RMSE: %.3f' % rmseCNN\_LSTM)  inv\_yhatP\_CNN\_LSTM = np.reshape(inv\_yhatP\_CNN\_LSTM, (inv\_yhatP\_CNN\_LSTM.shape[0],shift))  plt.clf()  plt.title('CNN-LSTM - '+CIUDAD+' mean\_squared\_error: '+rmseCNN\_LSTM)  plt.plot(inv\_yhatP\_CNN\_LSTM[:,0], label='Pronostico '+str(shift)+' semanas')  plt.plot(y\_pron\_inv[:,0], label='Real')  plt.legend()  plt.grid()  plt.show()  # print(inv\_yhatP\_CNN\_LSTM[:,0],y\_pron\_inv[:,0])    # Error = mean\_squared\_error(inv\_yhatP\_CNN\_LSTM, y\_pron\_inv, squared=False)  # print('Error cuadratico medio: ', Error)  # print('\n\n\n+-------------------------------------------------------------------------------------------+\n')  return  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # Err\_cnn\_lstm = []  # for i in range(semanas\_pronostico):  Er1 = cnn\_lstm\_win(6)  Er1 = cnn\_lstm\_win(8)  Er1 = cnn\_lstm\_win(10)  Er1 = cnn\_lstm\_win(12)  #Err\_cnn\_lstm = np.append(Err\_cnn\_lstm, Er1)  # print(Err\_cnn\_lstm)  # plt.plot(Err\_cnn\_lstm,label="Error")  # plt.title('Error CNN-LSTM')  # plt.xlabel('Número de semanas pronosticadas')  # plt.ylabel('Error cuadrado medio')  # plt.legend()  # plt.grid()  # plt.show()  ----  +\*In[ ]:\*+  [source, ipython3]  ----  def perceptron\_win(shift=2):    x\_train,x\_test,x\_pron,y\_train,y\_test,y\_test\_inv, y\_pron, y\_pron\_inv = data\_gen(shift, cnn=False, mix=False)  seed = 3  np.random.seed(seed)  tf.random.set\_seed(seed)  epochs = 200  batch = 1  lr = 0.0003  adam = optimizers.Adam(lr)  model\_MLP1 = Sequential()  model\_MLP1.add(Dense(70, activation='relu'))  model\_MLP1.add(Dense(20, activation='relu'))  # model\_MLP1.add(Dense(1, activation='sigmoid'))  # model\_MLP1.add(Dense(150, activation='sigmoid', use\_bias=True, kernel\_initializer='glorot\_uniform', bias\_initializer='zeros'))  # model\_MLP1.add(Dense(70, activation='sigmoid', use\_bias=True, kernel\_initializer='glorot\_uniform', bias\_initializer='zeros'))  # model\_MLP1.add(Dense(40, activation='sigmoid', use\_bias=True, kernel\_initializer='glorot\_uniform', bias\_initializer='zeros'))  # model\_MLP1.add(Dense(15, activation='sigmoid', use\_bias=True, kernel\_initializer='glorot\_uniform', bias\_initializer='zeros'))  model\_MLP1.add(Dense(shift, activation='sigmoid'))  model\_MLP1.compile(loss='mean\_squared\_error', optimizer='adam', metrics=['mean\_squared\_error'])  MLP1\_history = model\_MLP1.fit(x\_train, y\_train, epochs=epochs, batch\_size=batch,validation\_data=(x\_test, y\_test),verbose=0, shuffle=False)  scores = model\_MLP1.evaluate(x\_train, y\_train)  print("\n%s: %.2f%%" % (model\_MLP1.metrics\_names[0], scores[0]\*100))  print("\n%s: %.2f%%" % (model\_MLP1.metrics\_names[1], scores[1]\*100))  yhatP = model\_MLP1.predict(x\_pron)      inv\_yhatP = Reescala(yhatP, np.amin(valy), np.amax(valy))  inv\_yhatP\_mlp1 = np.reshape(inv\_yhatP, (inv\_yhatP.shape[0],shift))    inv\_yhatP = np.reshape(inv\_yhatP, (inv\_yhatP.shape[0],shift))  plt.clf()  plt.plot(inv\_yhatP[:,0], label='Pronostico '+str(shift)+' semanas')  plt.plot(y\_pron\_inv[:,0], label='Real')  plt.legend()  plt.grid()  plt.show()    # Error = mean\_squared\_error(inv\_yhatP, y\_pron\_inv, squared=False)  # print('Error cuadratico medio: ', Error)  # print('\n\n\n+-------------------------------------------------------------------------------------------+\n')  return  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # Err\_mlp = []  # for i in range(semanas\_pronostico):  Er1 = perceptron\_win(6)  Er1 = perceptron\_win(8)  Er1 = perceptron\_win(10)  Er1 = perceptron\_win(12)  # Err\_mlp = np.append(Err\_mlp, Er1)  # print(Err\_mlp)  # plt.plot(Err\_mlp,label="Error")  # plt.title('Error MLP')  # plt.xlabel('Número de semanas pronosticadas')  # plt.ylabel('Error cuadrado medio')  # plt.legend()  # plt.grid()  # plt.show()  ----  +\*In[ ]:\*+  [source, ipython3]  ----  def lstm\_win(shift=2):    x\_train,x\_test,x\_pron,y\_train,y\_test,y\_test\_inv, y\_pron, y\_pron\_inv = data\_gen(shift, cnn=False,mix=False)  seed = 3  np.random.seed(seed)  epochs = 200  batch = 1  lr = 0.00003  adam = optimizers.Adam(lr)  tf.random.set\_seed(seed)  model\_LSTM1 = Sequential()  model\_LSTM1.add(LSTM(100, input\_shape=(x\_train.shape[1], x\_train.shape[2])))  model\_LSTM1.add(Dense(shift))  model\_LSTM1.compile(loss='mae', optimizer='adam')  # fit network  LSTM1\_history = model\_LSTM1.fit(x\_train, y\_train, epochs=epochs, batch\_size=batch, validation\_data=(x\_test, y\_test), verbose=0, shuffle=False)  scores = model\_LSTM1.evaluate(x\_train, y\_train)  print("\n%s: %.4f%%" % (model\_LSTM1.metrics\_names[0], scores\*100))  yhatP = model\_LSTM1.predict(x\_pron)  inv\_yhatLSTM = Reescala(yhatP, np.amin(valy), np.amax(valy))  inv\_yhat\_LSTM1 = np.reshape(inv\_yhatLSTM, (inv\_yhatLSTM.shape[0],shift))  rmseP = sqrt(mean\_squared\_error(y\_pron\_inv, inv\_yhat\_LSTM1))  print('Test RMSE: %.3f' % rmseP)    inv\_yhatLSTM = np.reshape(inv\_yhatLSTM, (inv\_yhatLSTM.shape[0],shift))  plt.clf()  plt.plot(inv\_yhat\_LSTM1[:,0], label='Pronostico '+str(shift)+' semanas')  plt.plot(y\_pron\_inv[:,0], label='Real')  plt.legend()  plt.grid()  plt.show()    # Error = mean\_squared\_error(inv\_yhat\_LSTM1, y\_pron\_inv, squared=False)  # print('Error cuadratico medio: ', Error)  # print('\n\n\n+-------------------------------------------------------------------------------------------+\n')  return  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # Err\_lstm = []  # for i in range(semanas\_pronostico):  Er1 = lstm\_win(6)  Er1 = lstm\_win(8)  Er1 = lstm\_win(10)  Er1 = lstm\_win(12)  # Err\_lstm = np.append(Err\_lstm, Er1)  # print(Err\_lstm)  # plt.plot(Err\_lstm,label="Error")  # plt.title('Error LSTM')  # plt.xlabel('Número de semanas pronosticadas')  # plt.ylabel('Error cuadrado medio')  # plt.legend()  # plt.grid()  # plt.show()  ----  +\*In[ ]:\*+  [source, ipython3]  ----  def cnn\_win(shift=2):    x\_train,x\_test,x\_pron,y\_train,y\_test,y\_test\_inv, y\_pron, y\_pron\_inv = data\_gen(shift, cnn=True, mix=False)  seed = 3  np.random.seed(seed)  epochs = 200  batch = 1  lr = 0.00003  adam = optimizers.Adam(lr)  tf.random.set\_seed(seed)    model\_CNN = Sequential()  model\_CNN.add(Conv1D(filters=64, kernel\_size=2, activation='relu', input\_shape=(x\_train.shape[1], x\_train.shape[2])))  model\_CNN.add(MaxPooling1D(pool\_size=2))  model\_CNN.add(Flatten())  model\_CNN.add(Dense(75, activation='relu'))  model\_CNN.add(Dense(shift))  model\_CNN.compile(loss='mse', optimizer = 'adam')  # fit network  CNN\_history = model\_CNN.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=epochs, verbose=0)  scores = model\_CNN.evaluate(x\_train, y\_train)  print("\n%s: %.2f%%" % (model\_CNN.metrics\_names, scores\*100))  # Prediccion con datos de Pronostico - Modelo CNN  yhatP\_CNN = model\_CNN.predict(x\_pron)  yhatP\_CNN = np.reshape(yhatP\_CNN, (yhatP\_CNN.shape[0],shift))  inv\_yhatP\_CNN = Reescala(yhatP\_CNN, np.amin(valy), np.amax(valy))  rmseCNN = sqrt(mean\_squared\_error(y\_pron\_inv, inv\_yhatP\_CNN))  print('Test RMSE: %.3f' % rmseCNN)    inv\_yhatP\_CNN = np.reshape(inv\_yhatP\_CNN, (inv\_yhatP\_CNN.shape[0],shift))  plt.clf()  plt.plot(inv\_yhatP\_CNN[:,0], label='Pronostico '+str(shift)+' semanas')  plt.plot(y\_pron\_inv[:,0], label='Real')  plt.legend()  plt.grid()  plt.show()    # Error = mean\_squared\_error(inv\_yhatP\_CNN, y\_pron\_inv, squared=False)  # print('Error cuadratico medio: ', Error)  # print('\n\n\n+-------------------------------------------------------------------------------------------+\n')  return  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # Err\_cnn = []  # for i in range(semanas\_pronostico):  # Er1 = cnn\_win(i+1)  # Err\_cnn = np.append(Err\_cnn, Er1)  # print(Err\_cnn)  # plt.plot(Err\_cnn,label="Error")  # plt.title('Error CNN')  # plt.xlabel('Número de semanas pronosticadas')  # plt.ylabel('Error cuadrado medio')  # plt.legend()  # plt.grid()  # plt.show()  Er1 = cnn\_win(6)  Er1 = cnn\_win(8)  Er1 = cnn\_win(10)  Er1 = cnn\_win(12)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ---- |

SERIES DE TIEMPO

|  |
| --- |
| +\*In[ ]:\*+[source, ipython3]  ----  import warnings  warnings.filterwarnings("ignore")  ----  +\*In[ ]:\*+  [source, ipython3]  ----  import pandas as pd  import numpy as np  import statsmodels.api as sm  from statsmodels.tsa.ar\_model import AutoReg;  import matplotlib.pyplot as plt  from sklearn.metrics import mean\_squared\_error  from sklearn.metrics import mean\_absolute\_error  from pandas import DataFrame  from pandas import concat  #plt.style.use('fivethirtyeight')  #plt.style.use('classic')  plt.style.use('ggplot')  ----  == Función para generar desfases  +\*In[ ]:\*+  [source, ipython3]  ----  # Función para transformar datos crudos en datos para el aprendizaje de una serie de tiempo  def desfasar(data, n\_atras=1, n\_adelante=1, dropnan=True):  n\_vars = 1 if type(data) is list else data.shape[1]  dfd = DataFrame(data)  cols, names = list(), list()  aux=dfd.columns.values  # Secuencia de entrada (t-n, ... t-1)  for i in range(n\_atras, 0, -1):  cols.append(dfd.shift(i))  names += [f'{aux[j]}(t-{i})' for j in range(n\_vars)]  # Secuencia de pronostico (t, t+1, ... t+n)  for i in range(0, n\_adelante+1):  cols.append(dfd.shift(-i))  if i == 0:  names += [f'{aux[j]}(t)' for j in range(n\_vars)]  else:  names += [f'{aux[j]}(t+{i})' for j in range(n\_vars)]  # Juntar toda la informacion  agg = concat(cols, axis=1)  agg.columns = names  # Eliminar los registros con valores NaN  if dropnan:  agg.dropna(inplace=True)  return agg  ----  == Función para la evaluacion del modelo Autorregresivo  +\*In[ ]:\*+  [source, ipython3]  ----  def analisis\_modelo(FILE\_D, n\_desfase, Limite, C, F):  #Lectura de datos  df = pd.read\_csv(FILE\_D)  df = df.dropna()    if C == 'Todos':  Columna = 'ALL\_CASES'  Mensaje = 'Todos los casos'  else:  Columna = 'CONFIRMED\_CASES'  Mensaje = 'Casos confirmados'  exog\_x1=df[['PTPM\_CON', 'TA2\_AUT\_60\_MEAN', 'TA2\_AUT\_60\_AMAX', 'HRA2\_AUT\_60\_MEAN', 'HRA2\_AUT\_60\_AMAX', 'NINO3', 'NINO4', 'NINO3.4','CONFIRMED\_CASES', 'ALL\_CASES']]    V\_mse = np.zeros((n\_desfase))  V\_mae = np.zeros((n\_desfase))  for desfase in range (0, n\_desfase):  exog\_x = desfasar(exog\_x1, 0, desfase);  endog\_y=df[Columna].shift(-desfase-1).dropna();  exog\_x\_Train = exog\_x[0:Limite];  exog\_x\_Test = exog\_x[Limite:];  endog\_y\_Train = endog\_y[0:Limite];  endog\_y\_Test = endog\_y[Limite:];  model = AutoReg(endog\_y\_Train, lags=0, trend='c',exog=exog\_x\_Train);  model\_fit = model.fit();  yhat = model\_fit.predict(len(endog\_y\_Train),len(endog\_y\_Train)+len(endog\_y\_Test)-1, exog\_oos=exog\_x\_Test);  mse = mean\_squared\_error(endog\_y\_Test.values, yhat.values);  mae = mean\_absolute\_error(endog\_y\_Test.values, yhat.values);  V\_mse[desfase] = mse;  V\_mae[desfase] = mae;  (min\_mse,i\_mse) = min((v,i) for i,v in enumerate(V\_mse));  (min\_mae,i\_mae) = min((v,i) for i,v in enumerate(V\_mae));  desfase = np.minimum(i\_mse, i\_mae);  exog\_x = desfasar(exog\_x1, 0, desfase);  endog\_y = df[Columna].shift(-desfase-1).dropna();  exog\_x\_Train = exog\_x[0:Limite];  exog\_x\_Test = exog\_x[Limite:];  endog\_y\_Train = endog\_y[0:Limite];  endog\_y\_Test = endog\_y[Limite:];  model = AutoReg(endog\_y\_Train, lags=0, trend='c',exog=exog\_x\_Train);  model\_fit = model.fit();  yhat = model\_fit.predict(len(endog\_y\_Train),len(endog\_y\_Train)+len(endog\_y\_Test)-1, exog\_oos=exog\_x\_Test);  mse = mean\_squared\_error(endog\_y\_Test.values, yhat.values);  mae = mean\_absolute\_error(endog\_y\_Test.values, yhat.values);  plt.figure(figsize=(10, 8));  plt.title('Error cuadratico medio\n' + Mensaje + ' - Desfase de 0 a 120')  plt.xlabel('Dias de desfase')  plt.ylabel('Error cuadratico medio')  plt.plot(V\_mse);  plt.show();  plt.figure(figsize=(10, 8));  plt.title('Error absoluto medio\n' + Mensaje + ' - Desfase de 0 a 120')  plt.xlabel('Dias de desfase')  plt.ylabel('Error absoluto medio')  plt.plot(V\_mae);  plt.show();  print ('Valor minimo del error cuadratico medio para', Mensaje, min\_mse,'Indice del valor minimo: ',i\_mse)  print ('Valor minimo del error absoluto medio para', Mensaje, min\_mae,'Indice del valor minimo: ',i\_mae)  # Graficar prediccion vs Valores de test  plt.figure(figsize=(10, 8));  plt.title('Modelo AutoRegresivo - Datos '+F+'\nPronostico - ' + Mensaje)  plt.xlabel('Dias 2018 - ...')  plt.ylabel('Número de casos')  plt.plot(yhat.values, c='g', label='Pronostico')  plt.plot(endog\_y\_Test.values, c='b', label=Mensaje)  plt.grid=True  plt.legend()  plt.show()  # # Calculo del error cuadratico medio "Datos reales vs pronostico"  # rmse = mean\_squared\_error(endog\_y\_Test.values, yhat.values);  print(Mensaje,'El error cuadratico medio del pronostico es {}'.format(round(mse, 10)))  ----  == Evaluación de datos diarios eliminando los NAN  +\*In[ ]:\*+  [source, ipython3]  ----  FILE\_D = 'DATOS\_DIARIOS/DATOS\_DIARIOS\_ESTACIONES\_NINO.csv'  # 3611 Para seleccionar hasta 2016-12-31  # 3970 Para seleccionar hasta 2017-12-31  hasta\_2016 = 3611  hasta\_2017 = 3970  Limite = hasta\_2017  n\_desfase = 120  analisis\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Confirmado','Diarios')  analisis\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Todos','Diarios')  ----  == Evaluación de datos diarios interpolando los NAN  +\*In[ ]:\*+  [source, ipython3]  ----  FILE\_D = 'DATOS\_DIARIOS/DATOS\_DIARIOS\_ESTACIONES\_NINO\_INTERPOLADOS.csv'  # 3611 Para seleccionar hasta 2016-12-31  # 3970 Para seleccionar hasta 2017-12-31  hasta\_2016 = 3611  hasta\_2017 = 3970  Limite = hasta\_2017  n\_desfase = 120  analisis\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Confirmado','Diarios')  analisis\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Todos','Diarios')  ----  == Evaluación de datos semanales  +\*In[ ]:\*+  [source, ipython3]  ----  FILE\_D = 'DATOS\_SEMANALES/DATOS\_SEMANALES\_ESTACIONES\_NINO\_INTERPOLADOS.csv'  # 528 Para seleccionar hasta 2016-12-31  # 580 Para seleccionar hasta 2017-12-31  hasta\_2016 = 528  hasta\_2017 = 580  Limite = hasta\_2017  n\_desfase = 52  analisis\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Confirmado', 'Semanales')  analisis\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Todos', 'Semanales')  ----  == Evaluación de datos Mensuales No estandarizados  +\*In[ ]:\*+  [source, ipython3]  ----  #Lectura de datos  FILE\_D = 'DATOS\_MENSUALES/DATOS\_MENSUALES\_ESTACIONES\_NINO.csv'  # 122 Para seleccionar hasta 2016-12-31  # 134 Para seleccionar hasta 2017-12-31  hasta\_2016 = 122  hasta\_2017 = 134  Limite = hasta\_2017  n\_desfase = 4  analisis\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Confirmado','Mensuales No Estandarizados')  analisis\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Todos','Mensuales No Estandarizados')  ----  == Evaluación de datos Mensuales No estandarizados  +\*In[ ]:\*+  [source, ipython3]  ----  #Lectura de datos  FILE\_D = 'DATOS\_MENSUALES/DATOS\_MENSUALES\_NORMALIZADOS\_CALI.csv'  # 122 Para seleccionar hasta 2016-12-31  # 134 Para seleccionar hasta 2017-12-31  hasta\_2016 = 122  hasta\_2017 = 134  Limite = hasta\_2017  n\_desfase = 4  analisis\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Confirmado','Mensuales Estandarizados')  analisis\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Todos','Mensuales Estandarizados')  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ----  +\*In[ ]:\*+  [source, ipython3]  ----  +\*In[ ]:\*+[source, ipython3]  ----  import warnings  warnings.filterwarnings("ignore")  ----  +\*In[ ]:\*+  [source, ipython3]  ----  import pandas as pd  import numpy as np  import statsmodels.api as sm  from statsmodels.tsa.arima\_model import ARIMA  import matplotlib.pyplot as plt  from sklearn.metrics import mean\_squared\_error  from sklearn.metrics import mean\_absolute\_error  from pandas import DataFrame  from pandas import concat  from math import sqrt  #plt.style.use('fivethirtyeight')  #plt.style.use('classic')  #plt.style.use('ggplot')  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # Función para transformar datos crudos en datos para el aprendizaje de una serie de tiempo  def desfasar(data, n\_atras=1, n\_adelante=1, dropnan=True):  n\_vars = 1 if type(data) is list else data.shape[1]  dfd = DataFrame(data)  cols, names = list(), list()  aux=dfd.columns.values  # Secuencia de entrada (t-n, ... t-1)  for i in range(n\_atras, 0, -1):  cols.append(dfd.shift(i))  names += [f'{aux[j]}(t-{i})' for j in range(n\_vars)]  # Secuencia de pronostico (t, t+1, ... t+n)  for i in range(0, n\_adelante+1):  cols.append(dfd.shift(-i))  if i == 0:  names += [f'{aux[j]}(t)' for j in range(n\_vars)]  else:  names += [f'{aux[j]}(t+{i})' for j in range(n\_vars)]  # Juntar toda la informacion  agg = concat(cols, axis=1)  agg.columns = names  # Eliminar los registros con valores NaN  if dropnan:  agg.dropna(inplace=True)  return agg  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # #Lectura de datos  # FILE = 'DATOS\_SEMANALES/DATOS\_SEMANALES\_ESTACIONES\_NINO\_INTERPOLADOS.csv'  # df = pd.read\_csv(FILE, index\_col=0, parse\_dates=True)  # df = df.dropna()  # df = df.replace(np.inf, np.nan).replace(-np.inf, np.nan).dropna()  # exog\_x=df[['PTPM\_CON', 'TA2\_AUT\_60\_MEAN', 'TA2\_AUT\_60\_AMIN', 'TA2\_AUT\_60\_AMAX', 'HRA2\_AUT\_60\_MEAN', 'HRA2\_AUT\_60\_AMIN', 'NINO3', 'ANOM.1', 'NINO4', 'ANOM.2', 'NINO3.4', 'ANOM.3', 'ONI\_ASCII', 'MEI']]  # endog\_y\_c=df['CONFIRMED\_CASES']  # endog\_y\_a=df['ALL\_CASES']  # exog\_x\_Train = exog\_x[0:445]  # exog\_x\_Test = exog\_x[445:607]  # exog\_x\_pron = exog\_x[607:]  # endog\_y\_Train\_c = endog\_y\_c[0:445]  # endog\_y\_Test\_c = endog\_y\_c[445:607]  # endog\_y\_pron\_c = endog\_y\_c[607:]  # endog\_y\_Train\_a = endog\_y\_a[0:445]  # endog\_y\_Test\_a = endog\_y\_a[445:607]  # endog\_y\_pron\_a = endog\_y\_a[607:]  # # print(endog\_y\_Train\_c)  # # print(endog\_y\_Test\_c)  # # print(endog\_y\_c)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  def evalua\_modelo(FILE, n\_desfase, Limite\_I, Limite\_S, C, N):  import pandas as pd  df = pd.read\_csv(FILE)  df = df.dropna()    if C == 'Todos':  Columna = 'ALL\_CASES'  Mensaje = 'Todos los casos'  else:  Columna = 'CONFIRMED\_CASES'  Mensaje = 'Casos confirmados'  exog\_x1 = df[['PTPM\_CON', 'TA2\_AUT\_60\_MEAN', 'TA2\_AUT\_60\_AMAX', 'HRA2\_AUT\_60\_MEAN', 'HRA2\_AUT\_60\_AMAX', 'NINO3', 'NINO4', 'NINO3.4','CONFIRMED\_CASES', 'ALL\_CASES']]      aic = []  MSE = []  MAE = []  pp = []  pd = []  pq = []  dsf = []  arma\_params = []  for desfase in range (0, n\_desfase):  exog\_x = desfasar(exog\_x1, 0, desfase);  print(exog\_x)  endog\_y = df[Columna].shift(-desfase).dropna();  exog\_x\_Train = exog\_x[0:Limite\_I];  exog\_x\_Test = exog\_x[Limite\_I:Limite\_S];  exog\_x\_Pron = exog\_x[Limite\_S:];  endog\_y\_Train = endog\_y[0:Limite\_I];  endog\_y\_Test = endog\_y[Limite\_I:Limite\_S];  endog\_y\_Pron = endog\_y[Limite\_S:];  for p in range(N):  for d in range(N):  for q in range(N):  try:  # Ajuste del modelo - ARMA Casos Confirmados  model = ARIMA(endog\_y\_Train, order=(p, d, q), exog=exog\_x\_Train);  model\_fit = model.fit(disp=False);  #Predicción para el conjunto de test  yhat = model\_fit.predict(len(endog\_y\_Train), len(endog\_y\_Train)+len(endog\_y\_Test)-1, exog=exog\_x\_Test);  mse = mean\_squared\_error(endog\_y\_Test.values, yhat.values);  mae = mean\_absolute\_error(endog\_y\_Test.values, yhat.values);  pp.append(p)  pd.append(d)  pq.append(q)  dsf.append(desfase)  aic.append(model\_fit.aic);  arma\_params.append('ARMA P = {} x D = {} x Q = {}, Desfase = {}, AIC = {}, MSE = {}, MAE = {}'.format(p, d, q, desfase, model\_fit.aic, mse, mae));  MSE.append(mse);  MAE.append(mae);  except:  continue  # Extracción de valores minimos del modelo  #Grafico de ajuste del modelo  plt.title(r'Criterio de información de Akaike (AIC)')  plt.xlabel('Numero de prueba')  plt.ylabel('Valor (AIC)')  plt.plot(aic)  plt.show()  print('\n\n\n')  plt.title(r'Error cuadratico medio (MSE)')  plt.xlabel('Numero de prueba')  plt.ylabel('Valor (MSE)')  plt.plot(MSE)  plt.show()  print('\n\n\n')  plt.title(r'Error absoluto medio (MAE)')  plt.xlabel('Numero de prueba')  plt.ylabel('Valor (MAE)')  plt.plot(MAE)  plt.show()      #Minimo por AIC  indice = aic.index(np.array(aic).min())  p = pp[indice]  d = pd[indice]  q = pq[indice]  desfase = dsf[indice]  print('Minimo del Criterio Akaike: ',np.array(aic).min())  print('Parametros: P= ', p , 'D = ' , d, 'Q = ' , q , 'Desfase = ' , desfase)  exog\_x = desfasar(exog\_x1, 0, desfase);  endog\_y=df[Columna].shift(-desfase).dropna();  exog\_x\_Train = exog\_x[0:Limite\_I];  exog\_x\_Test = exog\_x[Limite\_I:Limite\_S];  exog\_x\_Pron = exog\_x[Limite\_S:];  endog\_y\_Train = endog\_y[0:Limite\_I];  endog\_y\_Test = endog\_y[Limite\_I:Limite\_S];  endog\_y\_Pron = endog\_y[Limite\_S:];    model = ARIMA(endog\_y\_Train, order=(p, d, q), exog=exog\_x\_Train);  model\_fit = model.fit(disp=False);  #Predicción para el conjunto de test  # print(len(endog\_y\_Train))  # print(len(endog\_y\_Train)+len(endog\_y\_Test)-1)  # print(len(exog\_x\_Test))  # print(exog\_x\_Test)  # print('\n\n\n')  yhat = model\_fit.predict(len(endog\_y\_Train), len(endog\_y\_Train)+len(endog\_y\_Test)-1, exog=exog\_x\_Test);  #Prediccion para el conjunto de pronostico  # print(len(endog\_y\_Train)+len(endog\_y\_Test))  # print(len(endog\_y\_Train)+len(endog\_y\_Test)+len(endog\_y\_Pron)-1)  # print(len(exog\_x\_Pron))  # print(exog\_x\_Pron)  # print('\n\n\n')        # +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    yhat\_p = model\_fit.predict(len(endog\_y\_Train), len(endog\_y\_Train)+len(endog\_y\_Pron)-1, exog=exog\_x\_Pron);      y\_l = []  y\_l.append(endog\_y\_Pron)  y\_l.append(yhat\_p)  print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* y\_l \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  print(y\_l)  # Trazar el grafico de TEST  plt.figure(figsize=(10, 8));  plt.title(r'TEST - Conjunto de evaluación - '+Mensaje)  plt.plot(yhat.values,color="#0000ff", ls="--", lw="4",label='Pronostico');  plt.plot(endog\_y\_Test.values,color="green", ls="-.", lw="4",label='Casos reales');  plt.xlabel('Fecha')  plt.ylabel('Número de casos')  plt.grid=True  plt.legend()  plt.show()    # Trazar el grafico de PRONOSTICO  plt.figure(figsize=(10, 8));  plt.title(r'PRONOSTICO - Conjunto de evaluación - '+Mensaje)  plt.plot(yhat\_p.values,color="#0000ff", ls="--", lw="4",label='Pronostico');  plt.plot(endog\_y\_Pron.values,color="green", ls="-.", lw="4",label='Casos reales');  plt.xlabel('Fecha')  plt.ylabel('Número de casos')  plt.grid=True  plt.legend()  plt.show()      #Minimo por MSE  indice = MSE.index(np.array(MSE).min())  p = pp[indice]  d = pd[indice]  q = pq[indice]  desfase = dsf[indice]  print('Minimo Error Cuadratico Medio: ',np.array(MSE).min())  print('Parametros: P= ',p, 'D = ' , d,'Q = ',q,'Desfase = ',desfase)  exog\_x = desfasar(exog\_x1, 0, desfase);  endog\_y=df[Columna].shift(-desfase).dropna();  exog\_x\_Train = exog\_x[0:Limite\_I];  exog\_x\_Test = exog\_x[Limite\_I:Limite\_S];  exog\_x\_Pron = exog\_x[Limite\_S:];  endog\_y\_Train = endog\_y[0:Limite\_I];  endog\_y\_Test = endog\_y[Limite\_I:Limite\_S];  endog\_y\_Pron = endog\_y[Limite\_S:];          model = ARIMA(endog\_y\_Train, order=(p, d, q), exog=exog\_x\_Train);  model\_fit = model.fit(disp=False);  #Predicción para el conjunto de test  yhat = model\_fit.predict(len(endog\_y\_Train), len(endog\_y\_Train)+len(endog\_y\_Test)-1, exog=exog\_x\_Test);  #Prediccion para el conjunto de pronostico  yhat\_p = model\_fit.predict(len(endog\_y\_Train), len(endog\_y\_Train)+len(endog\_y\_Pron)-1, exog=exog\_x\_Pron);  # Trazar el grafico de TEST  plt.figure(figsize=(10, 8));  plt.title(r'TEST - Conjunto de evaluación - '+Mensaje)  plt.plot(yhat.values,color="#0000ff", ls="--", lw="4",label='Pronostico');  plt.plot(endog\_y\_Test.values,color="green", ls="-.", lw="4",label='Casos reales');  plt.xlabel('Fecha')  plt.ylabel('Número de casos')  plt.grid=True  plt.legend()  plt.show()    # Trazar el grafico de PRONOSTICO  plt.figure(figsize=(10, 8));  plt.title(r'PRONOSTICO - Conjunto de evaluación - '+Mensaje)  plt.plot(yhat\_p.values,color="#0000ff", ls="--", lw="4",label='Pronostico');  plt.plot(endog\_y\_Pron.values,color="green", ls="-.", lw="4",label='Casos reales');  plt.xlabel('Fecha')  plt.ylabel('Número de casos')  plt.grid=True  plt.legend()  plt.show()      #Minimo por MAE  indice = MAE.index(np.array(MAE).min())  p = pp[indice]  d = pd[indice]  q = pq[indice]  desfase = dsf[indice]  print('Minimo Error Absoluto: ',np.array(MAE).min())  print('Parametros: P= ',p, 'D = ' , d,'Q = ',q,'Desfase = ',desfase)  exog\_x = desfasar(exog\_x1, 0, desfase);  endog\_y=df[Columna].shift(-desfase).dropna();  exog\_x\_Train = exog\_x[0:Limite\_I];  exog\_x\_Test = exog\_x[Limite\_I:Limite\_S];  exog\_x\_Pron = exog\_x[Limite\_S:];  endog\_y\_Train = endog\_y[0:Limite\_I];  endog\_y\_Test = endog\_y[Limite\_I:Limite\_S];  endog\_y\_Pron = endog\_y[Limite\_S:];          model = ARIMA(endog\_y\_Train, order=(p, d, q), exog=exog\_x\_Train);  model\_fit = model.fit(disp=False);  #Predicción para el conjunto de test  yhat = model\_fit.predict(len(endog\_y\_Train), len(endog\_y\_Train)+len(endog\_y\_Test)-1, exog=exog\_x\_Test);  #Prediccion para el conjunto de pronostico  yhat\_p = model\_fit.predict(len(endog\_y\_Train), len(endog\_y\_Train)+len(endog\_y\_Pron)-1, exog=exog\_x\_Pron);  # Trazar el grafico de TEST  plt.figure(figsize=(10, 8));  plt.title(r'TEST - Conjunto de evaluación - '+Mensaje)  plt.plot(yhat.values,color="#0000ff", ls="--", lw="4",label='Pronostico');  plt.plot(endog\_y\_Test.values,color="green", ls="-.", lw="4",label='Casos reales');  plt.xlabel('Fecha')  plt.ylabel('Número de casos')  plt.grid=True  plt.legend()  plt.show()    # Trazar el grafico de PRONOSTICO  plt.figure(figsize=(10, 8));  plt.title(r'PRONOSTICO - Conjunto de evaluación - '+Mensaje)  plt.plot(yhat\_p.values,color="#0000ff", ls="--", lw="4",label='Pronostico');  plt.plot(endog\_y\_Pron.values,color="green", ls="-.", lw="4",label='Casos reales');  plt.xlabel('Fecha')  plt.ylabel('Número de casos')  plt.grid=True  plt.legend()  plt.show()      print('\n\nParametros de Criterio Akaike minimo: ',aic.index(np.array(aic).min()),arma\_params[aic.index(np.array(aic).min())])  print('\nParametros de RMSE minimo: ',MSE.index(np.array(MSE).min()),arma\_params[MSE.index(np.array(MSE).min())])  print('\nParametros de RMSE minimo: ',MAE.index(np.array(MAE).min()),arma\_params[MAE.index(np.array(MAE).min())])  print('\n\n\n')  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Datos mensuales no estandarizados  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Lectura de datos  FILE\_D = 'D:/EN REVISION/PRUEBA TODOS/DATOS\_SEMANALES/DATOS\_SEMANALES\_ESTACIONES\_NINO\_INTERPOLADOS.csv'  # # 122 Para seleccionar hasta 2016-12-31  # # 134 Para seleccionar hasta 2017-12-31  # hasta\_2016 = 122  # hasta\_2017 = 445  # Limite = hasta\_2017  n\_desfase = 16  # evalua\_modelo(FILE\_D, n\_desfase, 445, 607, 'Confirmado', 2)  evalua\_modelo(FILE\_D, n\_desfase, 445, 607, 'Todos', 1)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Datos mensuales estandarizados  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # #Lectura de datos  # FILE\_D = 'DATOS\_MENSUALES/DATOS\_MENSUALES\_NORMALIZADOS\_CALI.csv'  # # 122 Para seleccionar hasta 2016-12-31  # # 134 Para seleccionar hasta 2017-12-31  # hasta\_2016 = 122  # hasta\_2017 = 134  # Limite = hasta\_2017  # n\_desfase = 4  # evalua\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Confirmado', 2)  # evalua\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Todos', 2)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Datos semanales  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # #Lectura de datos  # FILE\_D = 'DATOS\_SEMANALES/DATOS\_SEMANALES\_ESTACIONES\_NINO\_INTERPOLADOS.csv'  # # 528 Para seleccionar hasta 2016-12-31  # # 580 Para seleccionar hasta 2017-12-31  # hasta\_2016 = 528  # hasta\_2017 = 580  # Limite = hasta\_2017  # n\_desfase = 4  # evalua\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Confirmado', 2)  # evalua\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Todos', 2)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Datos diarios interpolando NAN  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # FILE\_D = 'DATOS\_DIARIOS/DATOS\_DIARIOS\_ESTACIONES\_NINO\_INTERPOLADOS.csv'  # # 3611 Para seleccionar hasta 2016-12-31  # # 3970 Para seleccionar hasta 2017-12-31  # hasta\_2016 = 3611  # hasta\_2017 = 3970  # Limite = hasta\_2017  # n\_desfase = 4  # evalua\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Confirmado', 2)  # evalua\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Todos', 2)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Datos diarios eliminando NAN  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # FILE\_D = 'DATOS\_DIARIOS/DATOS\_DIARIOS\_ESTACIONES\_NINO.csv'  # # 3611 Para seleccionar hasta 2016-12-31  # # 3970 Para seleccionar hasta 2017-12-31  # hasta\_2016 = 3611  # hasta\_2017 = 3970  # Limite = hasta\_2017  # n\_desfase = 4  # evalua\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Confirmado', 2)  # evalua\_modelo(FILE\_D, n\_desfase, hasta\_2017, 'Todos', 2)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  FILE = 'D:/EN REVISION/PRUEBA TODOS/DATOS\_SEMANALES/DATOS\_SEMANALES\_ESTACIONES\_NINO\_INTERPOLADOS.csv'  import pandas as pd  df = pd.read\_csv(FILE)  df = df.dropna()  df  ----  +\*In[ ]:\*+  [source, ipython3]  ----  exog\_x1 = df[['PTPM\_CON', 'TA2\_AUT\_60\_MEAN', 'TA2\_AUT\_60\_AMAX', 'HRA2\_AUT\_60\_MEAN', 'HRA2\_AUT\_60\_AMAX', 'NINO3', 'NINO4', 'NINO3.4','CONFIRMED\_CASES', 'ALL\_CASES']]  exog\_x1  ----  +\*In[ ]:\*+  [source, ipython3]  ----  FILE = 'D:/EN REVISION/PRUEBA TODOS/DATOS\_SEMANALES/DATOS\_SEMANALES\_ESTACIONES\_NINO\_INTERPOLADOS.csv'  import pandas as pd  df = pd.read\_csv(FILE)  df = df.dropna()  df  exog\_x1 = df[['PTPM\_CON', 'TA2\_AUT\_60\_MEAN', 'TA2\_AUT\_60\_AMAX', 'HRA2\_AUT\_60\_MEAN', 'HRA2\_AUT\_60\_AMAX', 'NINO3', 'NINO4', 'NINO3.4','CONFIRMED\_CASES', 'ALL\_CASES']]  exog\_x1 = pd.DataFrame(exog\_x1);  exog\_x1  exog\_x = desfasar(exog\_x1, 0, 3);  # # endog\_y = df[Columna].shift(-3).dropna();  # # exog\_x  ---- |

SELECCIÓN DE CARACTERISTICAS

|  |
| --- |
| == FINAL  +\*In[ ]:\*+  [source, ipython3]  ----  ## CARGAR LIBRERIAS  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Ignorar los warning  import warnings  warnings.filterwarnings("ignore")  import pandas as pd  import numpy as np  from numpy import vectorize  # Import the necessary libraries first  from sklearn.feature\_selection import SelectKBest, f\_classif, mutual\_info\_classif, f\_regression, chi2, mutual\_info\_regression, SelectPercentile, SelectFpr, SelectFdr, SelectFwe, GenericUnivariateSelect  from sklearn.feature\_selection import VarianceThreshold  import numpy as np  import matplotlib.pyplot as plt  from sklearn.datasets import load\_iris  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import MinMaxScaler  from sklearn.svm import SVR  from sklearn.pipeline import make\_pipeline  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ## FUNCIÓN PARA DESFASE  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # Función para transformar datos crudos en datos para el aprendizaje de una serie de tiempo  def desfasar(data, n\_atras=1, n\_adelante=1, dropnan=True):  n\_vars = 1 if type(data) is list else data.shape[1]  dfd = pd.DataFrame(data)  cols, names = list(), list()  aux=dfd.columns.values  # Secuencia de entrada (t-n, ... t-1)  for i in range(n\_atras, 0, -1):  cols.append(dfd.shift(i))  names += [f'{aux[j]}(t-{i})' for j in range(n\_vars)]  # Secuencia de pronostico (t, t+1, ... t+n)  for i in range(0, n\_adelante+1):  cols.append(dfd.shift(-i))  if i == 0:  names += [f'{aux[j]}(t)' for j in range(n\_vars)]  else:  names += [f'{aux[j]}(t+{i})' for j in range(n\_vars)]  # Juntar toda la informacion  agg = pd.concat(cols, axis=1)  agg.columns = names  # Eliminar los registros con valores NaN  if dropnan:  agg.dropna(inplace=True)  return agg  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ## FUNCION PARA ESCALADO  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Función para escalar los datos, recibe dato a escalar (vector), minimo y maximo  def escala(x,mi,ma):  return (x-mi)/(ma-mi)  escalador = vectorize(escala)  #Ejemplo de uso  # values[:,0]=escalador(values[:,0],0,300)  def Reescala(x,mi,ma):  return x\*(ma-mi)+mi  Reescalador = vectorize(Reescala)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ## FUNCION PARA SELECCION DE CARACTERISTICAS  ----  +\*In[ ]:\*+  [source, ipython3]  ----  def seleccion(param,n):  plt.figure(1,figsize=(10,5))  plt.clf()  X\_indices = np.arange(X.shape[-1])  # #############################################################################  # Univariate feature selection with F-test for feature scoring  # We use the default selection function to select the four  # most significant features  selector = SelectKBest(param, k=n)  selector.fit(x\_train, y\_train)  #print("p\_values: ",selector.pvalues\_)  scores = selector.scores\_  mask = selector.get\_support() #list of booleans  new\_features = [] # The list of your K best features  for bool, feature,score in zip(mask, feature\_names,scores):  if bool:  new\_features.append([feature,score])  #print('caracteristicas: ',sorted(new\_features, key = lambda x: x[1], reverse=True))  # #############################################################################  # Compare to the weights of an SVM  clf = make\_pipeline(MinMaxScaler(), SVR(kernel='linear'))  clf.fit(x\_train, y\_train)  print('Classification accuracy without selecting features: {}'  .format(clf.score(x\_test, y\_test)))  svm\_weights = np.abs(clf[-1].coef\_).sum(axis=0)  svm\_weights /= svm\_weights.sum()  plt.bar(X\_indices , svm\_weights, width=.5, label='SVM weight')#- .25  clf\_selected = make\_pipeline(  SelectKBest(param, k=n), MinMaxScaler(), SVR( kernel='linear')  )  clf\_selected.fit(x\_train, y\_train)    print('Classification accuracy after univariate feature selection: {}'  .format(clf\_selected.score(x\_test, y\_test)))  aux = clf\_selected.score(x\_test, y\_test) # para retornar valor  svm\_weights\_selected = np.abs(clf\_selected[-1].coef\_).sum(axis=0)  svm\_weights\_selected /= svm\_weights\_selected.sum()  plt.bar(X\_indices[selector.get\_support()], svm\_weights\_selected,  width=.5, label='SVM weights after selection') # - .05  plt.title("Comparing feature selection")  plt.xlabel('Feature number')  #plt.yticks(())  plt.grid()  plt.axis('tight')  plt.legend(loc='upper right')  plt.show()  return aux    ----  +\*In[ ]:\*+  [source, ipython3]  ----  ## LECTURA DE DATOS Y DEFINICIÓN DE PARAMETROS  ----  +\*In[ ]:\*+  [source, ipython3]  ----  Ciudad = 'CALI/'  DATA = 'barranquilla\_consolidado\_2013\_2020.csv'  PATH\_I = 'D:/DENGUE\_CODIGO/'  PATH\_O = PATH\_I + Ciudad + 'RESULTADOS/CLIMA/'  FILE = PATH\_I + 'DATOS\_BASE/' + DATA  #Lectura de datos diarios  df1 = pd.read\_csv(FILE, index\_col=None, parse\_dates=True)  df1 = df1.interpolate()  # Variables globales  desfase = 23  n\_semanas = 53  title\_string = 'Variables climaticas'  nombre\_a = 'caracteristicas\_24.csv'  nombre\_g = 'rendimiento\_24.jpg'  df1  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ## EXTRACCION DE LOS DATOS A SER USADOS  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Extraer los datos climaticos y del niño  #XS=df1[['PRECIPITATION', 'DRY\_DAY', 'TEMPERATURE\_AMAX', 'REL\_HUMIDITY\_AMAX', 'TEMPERATURE\_AMIN', 'REL\_HUMIDITY\_AMIN', 'TEMPERATURE\_MEAN', 'TEMPERATURE\_RANGE', 'REL\_HUMIDITY\_MEAN', 'MEI', 'ONI', 'SST1+2', 'SST3', 'SST4', 'SST3.4', 'TNI', 'DENGUE']]  XS=df1[['PRECIPITATION', 'DRY\_DAYS', 'TEMPERATURE\_AMAX', 'REL\_HUMIDITY\_AMAX', 'TEMPERATURE\_AMIN', 'REL\_HUMIDITY\_AMIN', 'TEMPERATURE\_MEAN', 'TEMPERATURE\_RANGE', 'REL\_HUMIDITY\_MEAN', 'DENGUE']]  # #Extraer la columna con el numero de las semanas  Semanas = df1['SEMANA']  #Extraer el numero de casos de Dengue y generar las salidas para los datos desfasados  out = XS['DENGUE']  out = out[desfase+1:].values  # #Extraer los numeros de semana con el desfase y calacualr las señales seno y coseno  Semanas = Semanas.iloc[desfase:].values  S\_s = np.sin(Semanas\*2\*np.pi/n\_semanas)  S\_c = np.cos(Semanas\*2\*np.pi/n\_semanas)  #Hallar los datos desfasados  XS = desfasar(XS, 0, desfase)  # #Insertar las columnas con las señales seno y cos y con el numero de la semana  XS.insert(0,"Semana\_cos",S\_c)  XS.insert(0,"Semana\_sin",S\_s)  XS.insert(0,"Semana",Semanas)  XS = XS.drop(XS.index[len(XS)-1])  #Insertar la salida, para formar el conjunto de datos  XS.insert(XS.shape[1],'DENGUE\_ESPERADO',out)  print(XS.shape)  #Guardar el archivo de datos desfasados con toda la informacion generada  XS.to\_csv(PATH\_O + 'datos\_semana\_desfase\_23\_todos.csv', index=True)  #Extraer solo los datos requeridos para el entrenamiento  # , 'DENGUE(t+15)', 'DENGUE(t+14)', 'DENGUE(t+13)', 'DENGUE(t+12)', 'DENGUE(t+11)', 'DENGUE(t+10)', 'DENGUE(t+9)', 'DENGUE(t+8)', 'DENGUE(t+7)', 'DENGUE(t+6)', 'DENGUE(t+5)', 'DENGUE(t+4)', 'DENGUE(t+3)', 'DENGUE(t+2)', 'DENGUE(t+1)', 'DENGUE(t)',  XS\_REAL = XS[['Semana', 'Semana\_sin', 'Semana\_cos', 'PRECIPITATION(t)', 'DRY\_DAYS(t)', 'TEMPERATURE\_AMAX(t)', 'REL\_HUMIDITY\_AMAX(t)', 'TEMPERATURE\_AMIN(t)', 'REL\_HUMIDITY\_AMIN(t)', 'TEMPERATURE\_MEAN(t)', 'TEMPERATURE\_RANGE(t)', 'REL\_HUMIDITY\_MEAN(t)', 'PRECIPITATION(t+1)', 'DRY\_DAYS(t+1)', 'TEMPERATURE\_AMAX(t+1)', 'REL\_HUMIDITY\_AMAX(t+1)', 'TEMPERATURE\_AMIN(t+1)', 'REL\_HUMIDITY\_AMIN(t+1)', 'TEMPERATURE\_MEAN(t+1)', 'TEMPERATURE\_RANGE(t+1)', 'REL\_HUMIDITY\_MEAN(t+1)', 'PRECIPITATION(t+2)', 'DRY\_DAYS(t+2)', 'TEMPERATURE\_AMAX(t+2)', 'REL\_HUMIDITY\_AMAX(t+2)', 'TEMPERATURE\_AMIN(t+2)', 'REL\_HUMIDITY\_AMIN(t+2)', 'TEMPERATURE\_MEAN(t+2)', 'TEMPERATURE\_RANGE(t+2)', 'REL\_HUMIDITY\_MEAN(t+2)', 'PRECIPITATION(t+3)', 'DRY\_DAYS(t+3)', 'TEMPERATURE\_AMAX(t+3)', 'REL\_HUMIDITY\_AMAX(t+3)', 'TEMPERATURE\_AMIN(t+3)', 'REL\_HUMIDITY\_AMIN(t+3)', 'TEMPERATURE\_MEAN(t+3)', 'TEMPERATURE\_RANGE(t+3)', 'REL\_HUMIDITY\_MEAN(t+3)', 'PRECIPITATION(t+4)', 'DRY\_DAYS(t+4)', 'TEMPERATURE\_AMAX(t+4)', 'REL\_HUMIDITY\_AMAX(t+4)', 'TEMPERATURE\_AMIN(t+4)', 'REL\_HUMIDITY\_AMIN(t+4)', 'TEMPERATURE\_MEAN(t+4)', 'TEMPERATURE\_RANGE(t+4)', 'REL\_HUMIDITY\_MEAN(t+4)', 'PRECIPITATION(t+5)', 'DRY\_DAYS(t+5)', 'TEMPERATURE\_AMAX(t+5)', 'REL\_HUMIDITY\_AMAX(t+5)', 'TEMPERATURE\_AMIN(t+5)', 'REL\_HUMIDITY\_AMIN(t+5)', 'TEMPERATURE\_MEAN(t+5)', 'TEMPERATURE\_RANGE(t+5)', 'REL\_HUMIDITY\_MEAN(t+5)', 'PRECIPITATION(t+6)', 'DRY\_DAYS(t+6)', 'TEMPERATURE\_AMAX(t+6)', 'REL\_HUMIDITY\_AMAX(t+6)', 'TEMPERATURE\_AMIN(t+6)', 'REL\_HUMIDITY\_AMIN(t+6)', 'TEMPERATURE\_MEAN(t+6)', 'TEMPERATURE\_RANGE(t+6)', 'REL\_HUMIDITY\_MEAN(t+6)', 'PRECIPITATION(t+7)', 'DRY\_DAYS(t+7)', 'TEMPERATURE\_AMAX(t+7)', 'REL\_HUMIDITY\_AMAX(t+7)', 'TEMPERATURE\_AMIN(t+7)', 'REL\_HUMIDITY\_AMIN(t+7)', 'TEMPERATURE\_MEAN(t+7)', 'TEMPERATURE\_RANGE(t+7)', 'REL\_HUMIDITY\_MEAN(t+7)', 'PRECIPITATION(t+8)', 'DRY\_DAYS(t+8)', 'TEMPERATURE\_AMAX(t+8)', 'REL\_HUMIDITY\_AMAX(t+8)', 'TEMPERATURE\_AMIN(t+8)', 'REL\_HUMIDITY\_AMIN(t+8)', 'TEMPERATURE\_MEAN(t+8)', 'TEMPERATURE\_RANGE(t+8)', 'REL\_HUMIDITY\_MEAN(t+8)', 'PRECIPITATION(t+9)', 'DRY\_DAYS(t+9)', 'TEMPERATURE\_AMAX(t+9)', 'REL\_HUMIDITY\_AMAX(t+9)', 'TEMPERATURE\_AMIN(t+9)', 'REL\_HUMIDITY\_AMIN(t+9)', 'TEMPERATURE\_MEAN(t+9)', 'TEMPERATURE\_RANGE(t+9)', 'REL\_HUMIDITY\_MEAN(t+9)', 'PRECIPITATION(t+10)', 'DRY\_DAYS(t+10)', 'TEMPERATURE\_AMAX(t+10)', 'REL\_HUMIDITY\_AMAX(t+10)', 'TEMPERATURE\_AMIN(t+10)', 'REL\_HUMIDITY\_AMIN(t+10)', 'TEMPERATURE\_MEAN(t+10)', 'TEMPERATURE\_RANGE(t+10)', 'REL\_HUMIDITY\_MEAN(t+10)', 'PRECIPITATION(t+11)', 'DRY\_DAYS(t+11)', 'TEMPERATURE\_AMAX(t+11)', 'REL\_HUMIDITY\_AMAX(t+11)', 'TEMPERATURE\_AMIN(t+11)', 'REL\_HUMIDITY\_AMIN(t+11)', 'TEMPERATURE\_MEAN(t+11)', 'TEMPERATURE\_RANGE(t+11)', 'REL\_HUMIDITY\_MEAN(t+11)', 'PRECIPITATION(t+12)', 'DRY\_DAYS(t+12)', 'TEMPERATURE\_AMAX(t+12)', 'REL\_HUMIDITY\_AMAX(t+12)', 'TEMPERATURE\_AMIN(t+12)', 'REL\_HUMIDITY\_AMIN(t+12)', 'TEMPERATURE\_MEAN(t+12)', 'TEMPERATURE\_RANGE(t+12)', 'REL\_HUMIDITY\_MEAN(t+12)', 'PRECIPITATION(t+13)', 'DRY\_DAYS(t+13)', 'TEMPERATURE\_AMAX(t+13)', 'REL\_HUMIDITY\_AMAX(t+13)', 'TEMPERATURE\_AMIN(t+13)', 'REL\_HUMIDITY\_AMIN(t+13)', 'TEMPERATURE\_MEAN(t+13)', 'TEMPERATURE\_RANGE(t+13)', 'REL\_HUMIDITY\_MEAN(t+13)', 'PRECIPITATION(t+14)', 'DRY\_DAYS(t+14)', 'TEMPERATURE\_AMAX(t+14)', 'REL\_HUMIDITY\_AMAX(t+14)', 'TEMPERATURE\_AMIN(t+14)', 'REL\_HUMIDITY\_AMIN(t+14)', 'TEMPERATURE\_MEAN(t+14)', 'TEMPERATURE\_RANGE(t+14)', 'REL\_HUMIDITY\_MEAN(t+14)', 'PRECIPITATION(t+15)', 'DRY\_DAYS(t+15)', 'TEMPERATURE\_AMAX(t+15)', 'REL\_HUMIDITY\_AMAX(t+15)', 'TEMPERATURE\_AMIN(t+15)', 'REL\_HUMIDITY\_AMIN(t+15)', 'TEMPERATURE\_MEAN(t+15)', 'TEMPERATURE\_RANGE(t+15)', 'REL\_HUMIDITY\_MEAN(t+15)', 'PRECIPITATION(t+16)', 'DRY\_DAYS(t+16)', 'TEMPERATURE\_AMAX(t+16)', 'REL\_HUMIDITY\_AMAX(t+16)', 'TEMPERATURE\_AMIN(t+16)', 'REL\_HUMIDITY\_AMIN(t+16)', 'TEMPERATURE\_MEAN(t+16)', 'TEMPERATURE\_RANGE(t+16)', 'REL\_HUMIDITY\_MEAN(t+16)', 'PRECIPITATION(t+17)', 'DRY\_DAYS(t+17)', 'TEMPERATURE\_AMAX(t+17)', 'REL\_HUMIDITY\_AMAX(t+17)', 'TEMPERATURE\_AMIN(t+17)', 'REL\_HUMIDITY\_AMIN(t+17)', 'TEMPERATURE\_MEAN(t+17)', 'TEMPERATURE\_RANGE(t+17)', 'REL\_HUMIDITY\_MEAN(t+17)', 'PRECIPITATION(t+18)', 'DRY\_DAYS(t+18)', 'TEMPERATURE\_AMAX(t+18)', 'REL\_HUMIDITY\_AMAX(t+18)', 'TEMPERATURE\_AMIN(t+18)', 'REL\_HUMIDITY\_AMIN(t+18)', 'TEMPERATURE\_MEAN(t+18)', 'TEMPERATURE\_RANGE(t+18)', 'REL\_HUMIDITY\_MEAN(t+18)', 'PRECIPITATION(t+19)', 'DRY\_DAYS(t+19)', 'TEMPERATURE\_AMAX(t+19)', 'REL\_HUMIDITY\_AMAX(t+19)', 'TEMPERATURE\_AMIN(t+19)', 'REL\_HUMIDITY\_AMIN(t+19)', 'TEMPERATURE\_MEAN(t+19)', 'TEMPERATURE\_RANGE(t+19)', 'REL\_HUMIDITY\_MEAN(t+19)', 'PRECIPITATION(t+20)', 'DRY\_DAYS(t+20)', 'TEMPERATURE\_AMAX(t+20)', 'REL\_HUMIDITY\_AMAX(t+20)', 'TEMPERATURE\_AMIN(t+20)', 'REL\_HUMIDITY\_AMIN(t+20)', 'TEMPERATURE\_MEAN(t+20)', 'TEMPERATURE\_RANGE(t+20)', 'REL\_HUMIDITY\_MEAN(t+20)', 'PRECIPITATION(t+21)', 'DRY\_DAYS(t+21)', 'TEMPERATURE\_AMAX(t+21)', 'REL\_HUMIDITY\_AMAX(t+21)', 'TEMPERATURE\_AMIN(t+21)', 'REL\_HUMIDITY\_AMIN(t+21)', 'TEMPERATURE\_MEAN(t+21)', 'TEMPERATURE\_RANGE(t+21)', 'REL\_HUMIDITY\_MEAN(t+21)', 'PRECIPITATION(t+22)', 'DRY\_DAYS(t+22)', 'TEMPERATURE\_AMAX(t+22)', 'REL\_HUMIDITY\_AMAX(t+22)', 'TEMPERATURE\_AMIN(t+22)', 'REL\_HUMIDITY\_AMIN(t+22)', 'TEMPERATURE\_MEAN(t+22)', 'TEMPERATURE\_RANGE(t+22)', 'REL\_HUMIDITY\_MEAN(t+22)', 'PRECIPITATION(t+23)', 'DRY\_DAYS(t+23)', 'TEMPERATURE\_AMAX(t+23)', 'REL\_HUMIDITY\_AMAX(t+23)', 'TEMPERATURE\_AMIN(t+23)', 'REL\_HUMIDITY\_AMIN(t+23)', 'TEMPERATURE\_MEAN(t+23)', 'TEMPERATURE\_RANGE(t+23)', 'REL\_HUMIDITY\_MEAN(t+23)', 'DENGUE\_ESPERADO']]  # XS\_REAL = XS[['MEI(t)', 'ONI(t)', 'SST1+2(t)', 'SST3(t)', 'SST4(t)', 'SST3.4(t)', 'TNI(t)', 'MEI(t+4)', 'ONI(t+4)', 'SST1+2(t+4)', 'SST3(t+4)', 'SST4(t+4)', 'SST3.4(t+4)', 'TNI(t+4)', 'MEI(t+8)', 'ONI(t+8)', 'SST1+2(t+8)', 'SST3(t+8)', 'SST4(t+8)', 'SST3.4(t+8)', 'TNI(t+8)', 'MEI(t+12)', 'ONI(t+12)', 'SST1+2(t+12)', 'SST3(t+12)', 'SST4(t+12)', 'SST3.4(t+12)', 'TNI(t+12)', 'MEI(t+16)', 'ONI(t+16)', 'SST1+2(t+16)', 'SST3(t+16)', 'SST4(t+16)', 'SST3.4(t+16)', 'TNI(t+16)', 'MEI(t+20)', 'ONI(t+20)', 'SST1+2(t+20)', 'SST3(t+20)', 'SST4(t+20)', 'SST3.4(t+20)', 'TNI(t+20)', 'DENGUE\_ESPERADO']]  print(XS\_REAL.shape)  df = XS\_REAL  df  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ## GENERAR DATOS DE ENTRENAMIENTO, EVALUACIÓN Y PRONOSTICO  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Separacion de entradas y salidas  XS\_REAL = XS\_REAL.values  X = XS\_REAL[:,:-1]  Y = XS\_REAL[:,-1]  #Numero de datos para el entrenamiento  n\_train = int(0.75\*len(Y))  #Numero de datos para test  n\_test = len(Y) - n\_train  #print("Cantidad total de datos = ",len(endog\_y),"\nNumero de datos para pronostico = ", n\_pronostico,"\nNumero de datos para entrenamiento = ",n\_train,"\nNumero de datos para test = ",n\_test)  print(n\_train,n\_test)  x\_train = X[0:n\_train,:]  x\_test = X[n\_train:n\_train+n\_test,:]  y\_train = Y[0:n\_train]  y\_test = Y[n\_train:n\_train+n\_test]  print(x\_train.shape,x\_test.shape,y\_train.shape,y\_test.shape)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Crear una matriz con los limites para el esclalado de los datos  limites=np.array(([np.amin(XS\_REAL[:,0])],[np.amax(XS\_REAL[:,0])]))  for i in range(1,len(XS\_REAL[1,:])):  limites=np.insert(limites, limites.shape[1], [np.amin(XS\_REAL[:,i]), np.amax(XS\_REAL[:,i])], axis=1)  print(limites.shape)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  scaled = np.zeros((len(XS\_REAL), len(XS\_REAL[0])))  for i in range(0,len(XS\_REAL[1,:])):  scaled[:,i]=escalador(XS\_REAL[:,i],limites[0][i],limites[1][i])    x, y = scaled[:, :-1], scaled[:, -1]  print(scaled.shape)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # nombres de las columnas  feature\_names = list(df.columns.values)  feature\_names  ----  +\*In[ ]:\*+  [source, ipython3]  ----  x\_train\_s = x[0:n\_train,:]  x\_test\_s = x[n\_train:n\_train+n\_test,:]  y\_train\_s = y[0:n\_train]  y\_test\_s = y[n\_train:n\_train+n\_test]  print(x\_train\_s.shape,x\_test\_s.shape,y\_train\_s.shape,y\_test\_s.shape)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # ¿ESTO NO SE ESTA USANDO? ------------------------------------------------  V=np.var(x\_train\_s, axis=0)  print(V.shape)  sel = VarianceThreshold(threshold=0.05)  X2 = sel.fit\_transform(x\_train\_s)  print(X2.shape)  print(X2)  # sel\_chi2 = SelectKBest (chi2, k = 4) # seleccionar 4 características  # X\_train\_chi2 = sel\_chi2.fit\_transform (x\_train\_s, y\_train\_s)  # print (sel\_chi2.get\_support ())  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #Definir el numero de variables que seran usadas en la seleccion de cacteristicas  n\_var = XS\_REAL.shape[1] # NUMERO DE VARIABLES  print(n\_var)  score\_maquinas=[]  for i in range(1,n\_var-1):  print("caracteristicas: ",i)  val=seleccion(f\_regression,i) #valor F anova  score\_maquinas.append([feature\_names[i], abs(val)])    # #print(len(score\_maquinas))  # plt.figure(1,figsize=(20,10))  # plt.clf()  # X\_indices = np.arange(len(score\_maquinas))  # print(score\_maquinas[i][1])  # plt.bar(X\_indices , score\_maquinas[i][1], width=.5, label='score') #- .45  # plt.title("'rendimiento maquinas'")  # plt.xlabel('Feature number')  # #plt.yticks(())  # plt.grid()  # plt.axis('tight')  # plt.legend(loc='upper right')  # plt.savefig(nombre\_g, dpi=300)  # plt.show()  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ----  +\*In[ ]:\*+  [source, ipython3]  ----  score\_maquinas\_orden = sorted(score\_maquinas, key = lambda x: x[1], reverse=True)  df\_score\_maquinas\_orden = pd.DataFrame(score\_maquinas\_orden)  df\_score\_maquinas\_orden  df\_score\_maquinas\_orden.to\_csv(PATH\_O + 'caracteristicas\_Wrapper\_ordenado.csv', index=True)  x\_g = df\_score\_maquinas\_orden[0]  y\_g = df\_score\_maquinas\_orden[1]  plt.figure(1,figsize=(25,9))  plt.y\_pos = np.arange(len(df\_score\_maquinas\_orden))  plt.bar(x\_g, y\_g)  X\_indices = np.arange(len(df\_score\_maquinas\_orden))  plt.ylabel('Peso', fontsize = 16)  plt.xlabel('Caracteristica', fontsize = 16)  plt.suptitle('Selección de caracteristicas - Wrapper', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  plt.xticks(X\_indices, x\_g, rotation='vertical', fontsize = 5)  # ax.set\_xscale('log')  #ax.invert\_yaxis()  plt.grid()  plt.savefig(PATH\_O + 'Wrapper\_ordenado.jpg', dpi=100, bbox\_inches='tight')  plt.show()  caracteristicas\_select=[]  print(len(score\_maquinas\_orden))  for i in range(0, len(score\_maquinas\_orden)):  if score\_maquinas\_orden[i][1]>6.5:  caracteristicas\_select.append(score\_maquinas\_orden[i][0])  caracteristicas\_select  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #print(len(score\_maquinas))  df\_score\_maquinas = pd.DataFrame(score\_maquinas)  plt.figure(1,figsize=(25,9))  plt.clf()  X\_indices = np.arange(len(score\_maquinas))  features = df\_score\_maquinas[0]  plt.bar(X\_indices , df\_score\_maquinas[1], width=.5, label='score') #- .45  plt.xticks(X\_indices, features, rotation='vertical', fontsize = 5)  plt.suptitle('rendimiento maquinas', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  plt.xlabel('Feature number', fontsize = 16)  plt.ylabel('Peso', fontsize = 16)  #plt.yticks(())  plt.grid()  plt.axis('tight')  plt.legend(loc='upper right')  plt.savefig(PATH\_O + 'Wrapper\_No\_ordenado.jpg', dpi=100, bbox\_inches='tight')  plt.show()  # GRAFICAR LOS DATOS OBTENIDOS EN FORMA ORDENADA  plt.figure(1,figsize=(25,9))  plt.clf()  X\_indices = np.arange(len(df\_score\_maquinas\_orden[0]))  features = df\_score\_maquinas\_orden[0]  plt.bar(X\_indices , df\_score\_maquinas\_orden[1], width=.5, label='score') #- .45  plt.xticks(X\_indices, features, rotation='vertical', fontsize = 5)  plt.suptitle('rendimiento maquinas - Valores ordenados', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  plt.xlabel('Feature number', fontsize = 16)  plt.ylabel('Peso', fontsize = 16)  #plt.yticks(())  plt.grid()  plt.axis('tight')  plt.legend(loc='upper right')  plt.savefig(PATH\_O + 'Wrapper\_ordenado\_01.jpg', dpi=100, bbox\_inches='tight')  plt.show()  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ## mutual\_info\_regression  ----  +\*In[ ]:\*+  [source, ipython3]  ----  n=n\_var-1 #numero caracteristicas  # Feature extraction  test = SelectKBest(score\_func = mutual\_info\_regression, k=n) #cambiar score\_func y k <- mejor segun grafica arriba  fit = test.fit(x, y)  # Summarize scores  np.set\_printoptions(precision=3)  scores = test.scores\_  mask = test.get\_support() #list of booleans  new\_features = [] # The list of your K best features  for bool, feature,score in zip(mask, feature\_names,scores):  if bool:  new\_features.append([feature,score])  CR1 = sorted(new\_features, key = lambda x: x[1], reverse=True)  df\_CR1 = pd.DataFrame(CR1)  #print(f'{n} caracteristicas: ',CR)  df\_CR1  #features = fit.transform(x)  # Summarize selected features  #print(features[0:5,:])  plt.figure(1,figsize=(25,9))  #ax = plt.gca()  plt.y\_pos = np.arange(len(df\_CR1))  X\_indices = np.arange(len(df\_CR1[0]))  features = df\_CR1[0]  score\_feature = df\_CR1[1]  plt.bar(features, score\_feature)  plt.xticks(X\_indices, features, rotation='vertical', fontsize = 5)  plt.xlabel('Peso', fontsize = 16)  plt.ylabel('Caracteristica', fontsize = 16)  plt.suptitle('Selección de caracteristicas - mutual\_info\_regression', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  # plt.xscale('log')  # plt.invert\_yaxis()  plt.grid()  plt.savefig(PATH\_O + 'Mutual\_info\_regression\_ordenado.jpg', dpi=100, bbox\_inches='tight')  plt.show()  df\_CR1  AAA = df\_CR1  for i in range (0, len(CR1)):  if CR1[i][1]>0.35:  caracteristicas\_select.append(CR1[i][0])  caracteristicas\_select  ----  +\*In[ ]:\*+  [source, ipython3]  ----  df\_CR1.to\_csv(PATH\_O + 'caracteristicas\_mutual\_regresion.csv', index=True)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ## f\_classif  ----  +\*In[ ]:\*+  [source, ipython3]  ----  n=n\_var-1 #numero caracteristicas  # Feature extraction  test = SelectKBest(score\_func = f\_classif, k=n) #cambiar score\_func y k <- mejor segun grafica arriba  fit = test.fit(x, y)  # Summarize scores  np.set\_printoptions(precision=3)  scores = test.scores\_  mask = test.get\_support() #list of booleans  new\_features = [] # The list of your K best features  for bool, feature,score in zip(mask, feature\_names,scores):  if bool:  new\_features.append([feature,score])    CR2 = sorted(new\_features, key = lambda x: x[1], reverse=True)  df\_CR2 = pd.DataFrame(CR2)  #print(f'{n} caracteristicas: ',CR)  df\_CR2  #print(f'{n} caracteristicas: ',sorted(new\_features, key = lambda x: x[1], reverse=True))  plt.figure(1,figsize=(25,9))  #ax = plt.gca()  plt.y\_pos = np.arange(len(df\_CR2))  X\_indices = np.arange(len(df\_CR2[0]))  features = df\_CR2[0]  score\_feature = df\_CR2[1]  plt.bar(features, score\_feature)  plt.xticks(X\_indices, features, rotation='vertical', fontsize = 5)  plt.xlabel('Peso', fontsize = 16)  plt.ylabel('Caracteristica', fontsize = 16)  plt.suptitle('Selección de caracteristicas - f\_classif', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  #plt.xscale('log')  #plt.invert\_yaxis()  plt.grid()  plt.savefig(PATH\_O + 'f\_classif\_ordenado.jpg', dpi=100, bbox\_inches='tight')  plt.show()  df\_CR2  BBB = df\_CR2  for i in range (0, len(CR2)):  if CR2[i][1]>5.0:  caracteristicas\_select.append(CR2[i][0])  caracteristicas\_select  ----  +\*In[ ]:\*+  [source, ipython3]  ----  df\_CR2.to\_csv(PATH\_O + 'caracteristicas\_peso\_f\_clasif.csv', index=True)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  ## f\_regression  ----  +\*In[ ]:\*+  [source, ipython3]  ----  n=n\_var-1 #numero caracteristicas  # Feature extraction  test = SelectKBest(score\_func = f\_regression, k=n) #cambiar score\_func y k <- mejor segun grafica arriba  fit = test.fit(x, y)  # Summarize scores  np.set\_printoptions(precision=3)  scores = test.scores\_  mask = test.get\_support() #list of booleans  new\_features = [] # The list of your K best features  for bool, feature,score in zip(mask, feature\_names,scores):  if bool:  new\_features.append([feature,score])  CR3 = sorted(new\_features, key = lambda x: x[1], reverse=True)  df\_CR3 = pd.DataFrame(CR3)  #print(f'{n} caracteristicas: ',CR)    #print(f'{n} caracteristicas: ',sorted(new\_features, key = lambda x: x[1], reverse=True))  plt.figure(1,figsize=(25,9))  X\_indices = np.arange(len(df\_CR3[0]))  #ax = plt.gca()  plt.y\_pos = np.arange(len(df\_CR3))  features = df\_CR3[0]  score\_feature = df\_CR3[1]  plt.bar(features, score\_feature)  plt.xlabel('Peso', fontsize = 16)  plt.ylabel('Caracteristica', fontsize = 16)  plt.suptitle('Selección de caracteristicas - f\_regression', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  plt.xticks(X\_indices, features, rotation='vertical', fontsize = 5)  plt.grid()  #ax.set\_xscale('log')  #ax.invert\_yaxis()  plt.savefig(PATH\_O + 'f\_regression\_ordenado.jpg', dpi=100, bbox\_inches='tight')  plt.show()  df\_CR3  CCC = df\_CR3  for i in range (0, len(CR3)):  if CR3[i][1]>15.0:  caracteristicas\_select.append(CR3[i][0])  print(len(caracteristicas\_select))  caracteristicas\_select1 = set(caracteristicas\_select)  print(len(caracteristicas\_select1))  caracteristicas\_select2 = sorted(list(caracteristicas\_select1))  caracteristicas\_select2  ----  +\*In[ ]:\*+  [source, ipython3]  ----  df\_CR2.to\_csv(PATH\_O + 'caracteristicas\_f\_regression\_ordenado.csv', index=True)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  df\_CR1.sort\_values(0, inplace=True)  AAA1 = df\_CR1  df\_CR2.sort\_values(0, inplace=True)  BBB1 = df\_CR2  df\_CR3.sort\_values(0, inplace=True)  CCC1 = df\_CR3  ----  +\*In[ ]:\*+  [source, ipython3]  ----  LLL1 = np.reshape(AAA1.values[:,0],(len(AAA1),1))  AAA1\_V = np.reshape(AAA1.values[:,1],(len(AAA1),1))  AAA1\_S = escalador(AAA1\_V, np.amin(AAA1\_V), np.amax(AAA1\_V))  BBB1\_V = np.reshape(BBB1.values[:,1],(len(AAA1),1))  BBB1\_S = escalador(BBB1\_V, np.amin(BBB1\_V), np.amax(BBB1\_V))  CCC1\_V = np.reshape(CCC1.values[:,1],(len(AAA1),1))  CCC1\_S = escalador(CCC1\_V, np.amin(CCC1\_V), np.amax(CCC1\_V))  carac = np.append(LLL1,AAA1\_V,axis=1)  carac = np.append(carac,BBB1\_V,axis=1)  carac = np.append(carac,CCC1\_V,axis=1)  Escal = AAA1\_S + BBB1\_S + CCC1\_S  carac  Escalados = np.append(LLL1,Escal,axis=1)  Escalados  C0 = list(carac[:,0])  C1 = list(carac[:,1])  C2 = list(carac[:,2])  C3 = list(carac[:,3])  Caracteristicas\_sin\_escala = pd.DataFrame({'Caracteristica': C0, 'mutual\_info\_regression' : C1, 'f\_classif' : C2, 'f\_regression': C3})  Caracteristicas\_sin\_escala.plot.bar(figsize = (25,9))  X\_indices = np.arange(len(Caracteristicas\_sin\_escala))  plt.xticks(X\_indices, Caracteristicas\_sin\_escala['Caracteristica'], rotation='vertical', fontsize = 5)  plt.ylabel('Peso', fontsize = 16)  plt.xlabel('Caracteristica', fontsize = 16)  plt.suptitle('Selección de caracteristicas - Metodos de filtro - No normalizados', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  plt.grid()  plt.savefig(PATH\_O + 'Metodos\_de\_filtro-NO-Normalizados.jpg', dpi=100, bbox\_inches='tight')  plt.show()  carac\_s = np.append(LLL1,AAA1\_S,axis=1)  carac\_s = np.append(carac\_s,BBB1\_S,axis=1)  carac\_s = np.append(carac\_s,CCC1\_S,axis=1)  C0\_S = list(carac\_s[:,0])  C1\_S = list(carac\_s[:,1])  C2\_S = list(carac\_s[:,2])  C3\_S = list(carac\_s[:,3])  Caracteristicas\_escalado = pd.DataFrame({'Caracteristica': C0\_S, 'mutual\_info\_regression' : C1\_S, 'f\_classif' : C2\_S, 'f\_regression': C3\_S})  Caracteristicas\_escalado.plot.bar(figsize = (25,9))  X\_indices = np.arange(len(Caracteristicas\_escalado))  plt.xticks(X\_indices, Caracteristicas\_escalado['Caracteristica'], rotation='vertical', fontsize = 5)  plt.ylabel('Peso', fontsize = 16)  plt.xlabel('Caracteristica', fontsize = 16)  plt.suptitle('Selección de caracteristicas - Metodos de filtro - Normalizados', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  plt.grid()  plt.savefig(PATH\_O + 'Metodos\_de\_filtro-Normalizados.jpg', dpi=100, bbox\_inches='tight')  plt.show()  #pd.concat([df\_CR1,df\_CR2,df\_CR3], ignore\_index=True, axis=1)  # limite\_caracteristica = np.array(([np.amin(caracteristicas\_v[:,0])],[np.amax(caracteristicas\_v[:,0])]))  # scaled\_feature[:,i]=escalador(caracteristicas\_v[:,i],limite\_caracteristica[0][i],limite\_caracteristica[1][i])  ----  +\*In[ ]:\*+  [source, ipython3]  ----  Escalados1 = list(Escalados[:,0])  Escalados2 = list(Escalados[:,1])  Caract\_Escal = pd.DataFrame({'Caracteristica': Escalados1, 'Acumulado': Escalados2})  Caract\_Escal  Caract\_Escal.plot.bar(figsize = (25,9))  X\_indices = np.arange(len(Caract\_Escal))  plt.xticks(X\_indices, Caract\_Escal['Caracteristica'], rotation='vertical', fontsize = 5)  plt.ylabel('Peso', fontsize = 16)  plt.xlabel('Caracteristica', fontsize = 16)  plt.suptitle('Selección de caracteristicas - Acumulado metodos de filtro', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  plt.grid()  plt.savefig(PATH\_O + 'Metodos\_de\_filtro-Acumulaos.jpg', dpi=100, bbox\_inches='tight')  plt.show()  Caract\_Escal\_Filtado = Caract\_Escal[Caract\_Escal['Acumulado']>0.5]  print(Caract\_Escal\_Filtado.shape)  Caract\_Escal\_Filtado.plot.bar(figsize = (25,9))  X\_indices = np.arange(len(Caract\_Escal\_Filtado))  plt.xticks(X\_indices, Caract\_Escal\_Filtado['Caracteristica'], rotation='vertical', fontsize = 5)  plt.ylabel('Peso', fontsize = 16)  plt.xlabel('Caracteristica', fontsize = 16)  plt.suptitle('Selección de caracteristicas - Acumulado metodos de filtro > 0.5', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  plt.grid()  plt.savefig(PATH\_O + 'Metodos\_de\_filtro-Acumulados\_mas\_05.jpg', dpi=100, bbox\_inches='tight')  plt.show()  #Caract\_Escal\_Filtado  ----  +\*In[ ]:\*+  [source, ipython3]  ----  Caract\_Escal\_ORDEN = Caract\_Escal.sort\_values('Acumulado',ascending=False)  Caract\_Escal\_ORDEN.plot.bar(figsize = (25,9))  X\_indices = np.arange(len(Caract\_Escal\_ORDEN))  plt.xticks(X\_indices, Caract\_Escal\_ORDEN['Caracteristica'], rotation='vertical', fontsize = 5)  plt.ylabel('Peso', fontsize = 16)  plt.xlabel('Caracteristica', fontsize = 16)  plt.suptitle('Selección de caracteristicas - Acumulado metodos de filtro - Ordenado', y=1.05, fontsize = 24)  plt.title(title\_string, fontsize=18)  plt.grid()  plt.savefig(PATH\_O + 'Metodos\_de\_filtro-Acumulados\_Ordenados.jpg', dpi=100, bbox\_inches='tight')  plt.show()  Caract\_Escal\_ORDEN.to\_csv(PATH\_O + 'caracteristicas\_filtro\_acumulado\_ordenado.csv', index=True)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # for i in range (0,len(Escalados)):  # if (Escalados[[i],[1]]>=0.5):  # print(Escalados[[i],[0]], "\t\t\t\t", Escalados[[i],[1]])  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # Escalados = Escalados[Escalados[:,1].argsort(),]  # #print(Escalados)  # Escalados = Escalados[::-1]  # #print(Escalados)  # Caracteristicas\_orden = list(Escalados[:,0])  # #print(Caracteristicas\_orden)  # Data\_orden = df[Caracteristicas\_orden]  # Data\_orden  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # caracteristicas\_s = pd.concat([df\_CR1,df\_CR2,df\_CR3], ignore\_index=True, axis=1)  # caracteristicas\_s  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # feature\_name = caracteristicas\_s[2]  # del caracteristicas\_s[2]  # del caracteristicas\_s[4]  # caracteristicas\_s.set\_index([0],inplace=True)  # caracteristicas\_s=caracteristicas\_s.rename(columns={1: "mutual\_info\_regression", 3: "f\_classif", 5:"f\_regression"})  # caracteristicas\_s.to\_csv(PATH\_O + "caracterisitcas\_ordenadas.csv", index=True)  # caracteristicas\_s  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # caracteristicas\_v = caracteristicas\_s.values  # #Crear una matriz con los limites para el esclalado de los datos  # limite\_caracteristica = np.array(([np.amin(caracteristicas\_v[:,0])],[np.amax(caracteristicas\_v[:,0])]))  # for i in range(1,len(caracteristicas\_v[1,:])):  # limite\_caracteristica = np.insert(limite\_caracteristica, limite\_caracteristica.shape[1], [np.amin(caracteristicas\_v[:,i]), np.amax(caracteristicas\_v[:,i])], axis=1)  # scaled\_feature = np.zeros((len(caracteristicas\_v), len(caracteristicas\_v[0])))  # for i in range(0,len(caracteristicas\_v[1,:])):  # scaled\_feature[:,i]=escalador(caracteristicas\_v[:,i],limite\_caracteristica[0][i],limite\_caracteristica[1][i])  # caracteristicas\_v = pd.DataFrame(scaled\_feature)  # caracteristicas\_v = pd.concat([feature\_name,caracteristicas\_v], ignore\_index=True, axis=1)  # caracteristicas\_v.set\_index([0],inplace=True)  # caracteristicas\_v=caracteristicas\_v.rename(columns={1: "mutual\_info\_regression", 2: "f\_classif", 3:"f\_regression"})  # caracteristicas\_v.to\_csv(PATH\_O + "caracterisitcas\_ord\_normal.csv", index=True)  # caracteristicas\_v  # acumulado = caracteristicas\_v['mutual\_info\_regression']+caracteristicas\_v['f\_classif']+caracteristicas\_v['f\_regression']  # acumulado  # # acumulado.plot.bar(figsize = (15,5))  # # # X\_indices = np.arange(len(acumulado))  # # # plt.xticks(X\_indices, acumulado[0], rotation='vertical', fontsize = 8)  # # plt.ylabel('Peso', fontsize = 16)  # # plt.xlabel('Caracteristica', fontsize = 16)  # # plt.title('Selección de caracteristicas - Acumulado metodos de filtro Alejo', fontsize = 24)  # # plt.grid()  # # # plt.savefig(PATH\_O + 'acumulados\_16.jpg', dpi=300)  # # plt.show()  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # caracteristicas\_s.plot.barh(figsize = (10,55))  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # caracteristicas\_v.plot.barh(figsize = (10,55))  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #para capturar las caracteristicas deseadas  #caracteristicas\_v.loc[['ONI(t+15)', 'ONI(t+11)', 'ONI(t+7)', 'ONI(t+3)', 'MEI(t+15)', 'MEI(t+11)', 'MEI(t+7)', 'MEI(t+3)', 'SST4(t+15)', 'SST4(t+11)', 'SST4(t+7)', 'SST4(t+3)', 'TNI(t+15)', 'TNI(t+11)', 'TNI(t+7)', 'TNI(t+3)', 'SST3(t+15)', 'SST3(t+11)', 'SST3(t+7)', 'SST3(t+3)', 'SST3.4(t+15)', 'SST3.4(t+11)', 'SST3.4(t+7)', 'SST3.4(t+3)', 'SST1+2(t+15)', 'SST1+2(t+11)', 'SST1+2(t+7)', 'SST1+2(t+3)', 'TEMPERATURE\_AMAX(t+15)', 'TEMPERATURE\_AMAX(t+14)', 'TEMPERATURE\_AMAX(t+13)', 'TEMPERATURE\_AMAX(t+12)', 'TEMPERATURE\_AMAX(t+11)', 'TEMPERATURE\_AMAX(t+10)', 'TEMPERATURE\_AMAX(t+9)', 'TEMPERATURE\_AMAX(t+8)', 'TEMPERATURE\_AMAX(t+7)', 'TEMPERATURE\_AMAX(t+6)', 'TEMPERATURE\_AMAX(t+5)', 'TEMPERATURE\_AMAX(t+4)', 'TEMPERATURE\_AMAX(t+3)', 'TEMPERATURE\_AMAX(t+2)', 'TEMPERATURE\_AMAX(t+1)', 'TEMPERATURE\_AMAX(t)', 'TEMPERATURE\_MEAN(t+15)', 'TEMPERATURE\_MEAN(t+14)', 'TEMPERATURE\_MEAN(t+13)', 'TEMPERATURE\_MEAN(t+12)', 'TEMPERATURE\_MEAN(t+11)', 'TEMPERATURE\_MEAN(t+10)', 'TEMPERATURE\_MEAN(t+9)', 'TEMPERATURE\_MEAN(t+8)', 'TEMPERATURE\_MEAN(t+7)', 'TEMPERATURE\_MEAN(t+6)', 'TEMPERATURE\_MEAN(t+5)', 'TEMPERATURE\_MEAN(t+4)', 'TEMPERATURE\_MEAN(t+3)', 'TEMPERATURE\_MEAN(t+2)', 'TEMPERATURE\_MEAN(t+1)', 'TEMPERATURE\_MEAN(t)', 'TEMPERATURE\_RANGE(t+15)', 'TEMPERATURE\_RANGE(t+14)', 'TEMPERATURE\_RANGE(t+13)', 'TEMPERATURE\_RANGE(t+12)', 'TEMPERATURE\_RANGE(t+11)', 'TEMPERATURE\_RANGE(t+10)', 'TEMPERATURE\_RANGE(t+9)', 'TEMPERATURE\_RANGE(t+8)', 'TEMPERATURE\_RANGE(t+7)', 'TEMPERATURE\_RANGE(t+6)', 'TEMPERATURE\_RANGE(t+5)', 'TEMPERATURE\_RANGE(t+4)', 'TEMPERATURE\_RANGE(t+3)', 'TEMPERATURE\_RANGE(t+2)', 'TEMPERATURE\_RANGE(t+1)', 'TEMPERATURE\_RANGE(t)', 'REL\_HUMIDITY\_AMAX(t+15)', 'REL\_HUMIDITY\_AMAX(t+14)', 'REL\_HUMIDITY\_AMAX(t+13)', 'REL\_HUMIDITY\_AMAX(t+12)', 'REL\_HUMIDITY\_AMAX(t+11)', 'REL\_HUMIDITY\_AMAX(t+10)', 'REL\_HUMIDITY\_AMAX(t+9)', 'REL\_HUMIDITY\_AMAX(t+8)', 'REL\_HUMIDITY\_AMAX(t+7)', 'REL\_HUMIDITY\_AMAX(t+6)', 'REL\_HUMIDITY\_AMAX(t+5)', 'REL\_HUMIDITY\_AMAX(t+4)', 'REL\_HUMIDITY\_AMAX(t+3)', 'REL\_HUMIDITY\_AMAX(t+2)', 'REL\_HUMIDITY\_AMAX(t+1)', 'REL\_HUMIDITY\_AMAX(t)', 'REL\_HUMIDITY\_MEAN(t+15)', 'REL\_HUMIDITY\_MEAN(t+14)', 'REL\_HUMIDITY\_MEAN(t+13)', 'REL\_HUMIDITY\_MEAN(t+12)', 'REL\_HUMIDITY\_MEAN(t+11)', 'REL\_HUMIDITY\_MEAN(t+10)', 'REL\_HUMIDITY\_MEAN(t+9)', 'REL\_HUMIDITY\_MEAN(t+8)', 'REL\_HUMIDITY\_MEAN(t+7)', 'REL\_HUMIDITY\_MEAN(t+6)', 'REL\_HUMIDITY\_MEAN(t+5)', 'REL\_HUMIDITY\_MEAN(t+4)', 'REL\_HUMIDITY\_MEAN(t+3)', 'REL\_HUMIDITY\_MEAN(t+2)', 'REL\_HUMIDITY\_MEAN(t+1)', 'REL\_HUMIDITY\_MEAN(t)', 'DENGUE(t+15)', 'DENGUE(t+14)', 'DENGUE(t+13)', 'DENGUE(t+12)', 'DENGUE(t+11)', 'DENGUE(t+10)', 'DENGUE(t+9)', 'DENGUE(t+8)', 'DENGUE(t+7)', 'DENGUE(t+6)', 'DENGUE(t+5)', 'DENGUE(t+4)', 'DENGUE(t+3)', 'DENGUE(t+2)', 'DENGUE(t+1)', 'DENGUE(t)']]  ----  +\*In[ ]:\*+  [source, ipython3]  ----  #df\_CR3.to\_csv(PATH\_O + 'caracteristicas\_f\_regresion.csv', index=True)  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # caracteristicas = pd.concat([df\_CR1,df\_CR2,df\_CR3], ignore\_index=True, axis=1)  # caracteristicas.to\_csv(nombre\_a, index=True)  # caracteristicas  ----  +\*In[ ]:\*+  [source, ipython3]  ----  # caracteristicas.to\_csv(PATH\_O + 'caracteristicas\_TODAS\_CARACTERISTICAS.csv', index=True)  ---- |