PROBLEMA DE REGRESION CON REDES NEURONALES

In [42]:
```python
import pandas as pd
pd.options.display.max_columns = 500
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.metrics as metrics
import warnings
warnings.filterwarnings('ignore')

from sklearn.neural_network import MLPRegressor
import multiprocessing

from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import mean_squared_error
```

In [2]:
```python
from sklearn.datasets import load_boston
boston_dataset = load_boston()

boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)

boston['MEDV'] = boston_dataset.target # faltaba esta columna

boston.head()
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

- CRIM : Tasa de delincuencia per cápita por ciudad
- ZN : Proporción de terrenos residenciales divididos en zonas para lotes de más de 25,000 pies cuadrados
- INDUS : Proporción de acres comerciales no minoristas por ciudad
- CHAS : Variable ficticia de Charles River (= 1 si el tramo limita con el río; 0 en caso contrario)
- NOX : Concentración de óxido nítrico (partes por 10 millones)
- RM : Número promedio de habitaciones por vivienda
- EDAD : Proporción de unidades ocupadas por el propietario construidas antes de 1940
- DIS : Distancias ponderadas a cinco centros de empleo de Boston
- RAD : Índice de accesibilidad a carreteras radiales
- IMPUESTO : Tasa de impuesto a la propiedad de valor total por $ 10,000
- PTRATIO : Proporción alumno-maestro por ciudad
- B : 1000 (Bk - 0,63) ², donde Bk es la proporción de [personas de ascendencia afroamericana] por ciudad
- LSTAT : Porcentaje de menor estatus de la población
- MEDV : Valor medio de las viviendas ocupadas por sus propietarios en $ 1000

In [3]:
```python
boston.shape

# tenemos 506 filas y 14 columnas
```

Out[3]:
```
(506, 14)
```

In [4]:
```python
boston.info()

# parece que no hay valores nulos y el tipo de dato de las columnas es float
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

In [5]:
```python
boston.isnull().sum()

# no hay nulos
```

Out[5]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

In [6]:
```python
boston.describe()
```

Out[6]:

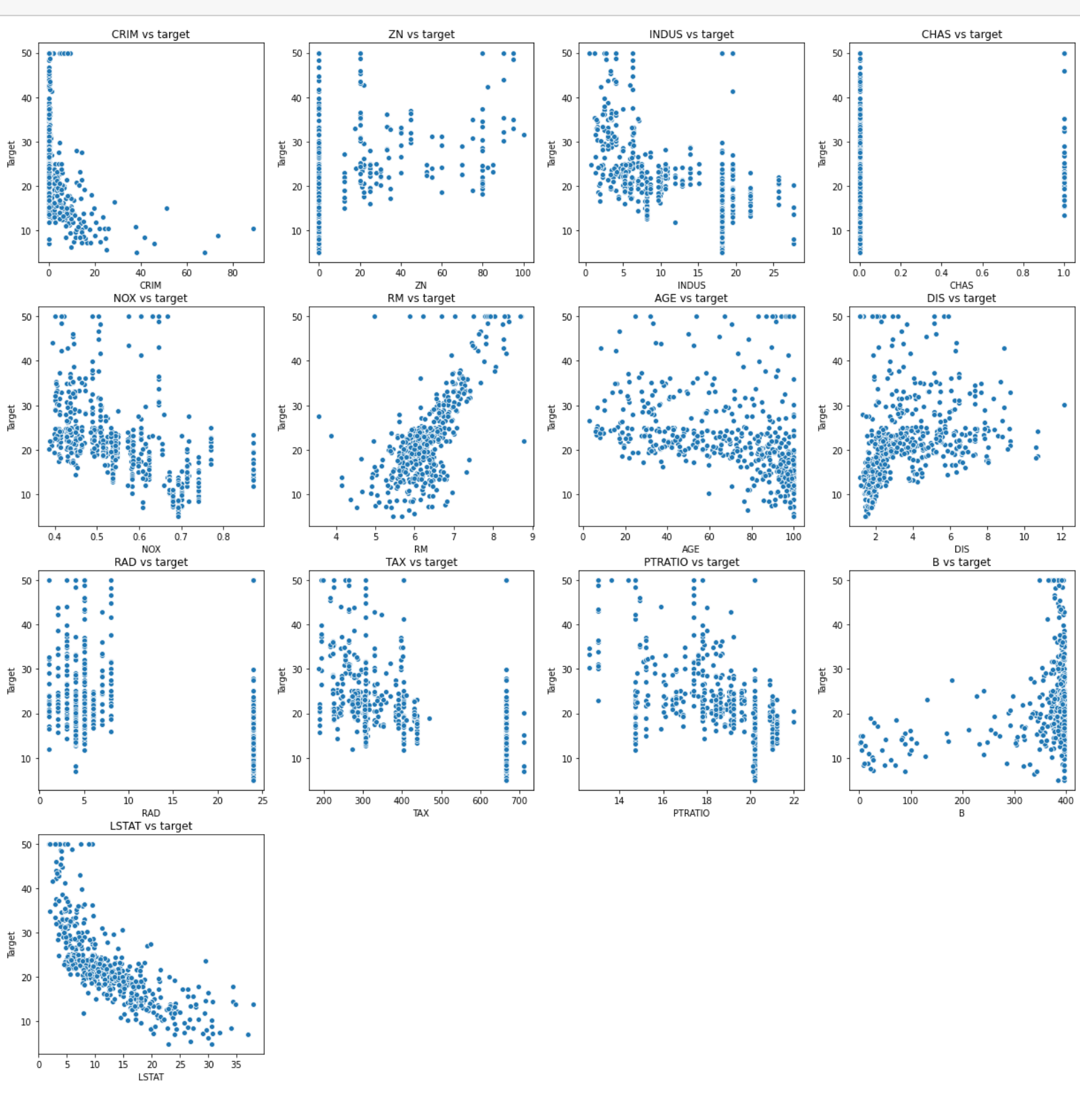
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TA
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.23715
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.53711
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.00000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.00000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.00000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.00000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.00000

In [7]:
```python
# dividimos el dataframe en dos, las variables independientes por un lado y las dependientes
por otro

X = boston.drop("MEDV", axis = 1)

Y = boston["MEDV"]
```

Vamos a graficar los datos con las funciones que tenemos guardadas

In [8]:
```python
def relaciones_vs_target(X, Y, return_type='axes'):
    '''
    Función que representa gráficos de dispersión de las variables
    en X en función a la variable Y
    '''
    fig_tot = (len(X.columns))
    fig_por_fila = 4.
    tamanio_fig = 4.
    num_filas = int( np.ceil(fig_tot/fig_por_fila) )
    plt.figure( figsize=( fig_por_fila*tamanio_fig+5, num_filas*tamanio_fig+5 ) )
    c = 0
    for i, col in enumerate(X.columns):
        plt.subplot(num_filas, fig_por_fila, i+1)
        sns.scatterplot(x=X[col], y=Y)
        plt.title( '%s vs %s' % (col, 'target') )
        plt.ylabel('Target')
        plt.xlabel(col)
    plt.show()
```

In [9]:
```python
relaciones_vs_target(X, Y, return_type='axes')

# Podemos ver que con la variable RM(Número promedio de habitaciones por vivienda) tiene una
relacion lineal positiva bastante fuerte
# y con LSTAT(Porcentaje de menor estatus de la población) tiene una relacion lienal negativ
a fuerte
```



Histogramas de las variables independientes

In [10]:
```python
plt.figure(figsize=(18,20))
n = 0
for i, column in enumerate(X.columns):
    n+=1
    plt.subplot(5, 5, n)
    sns.distplot(X[column], bins=30)
    plt.title('Distribución var {}'.format(column))
plt.show()
```

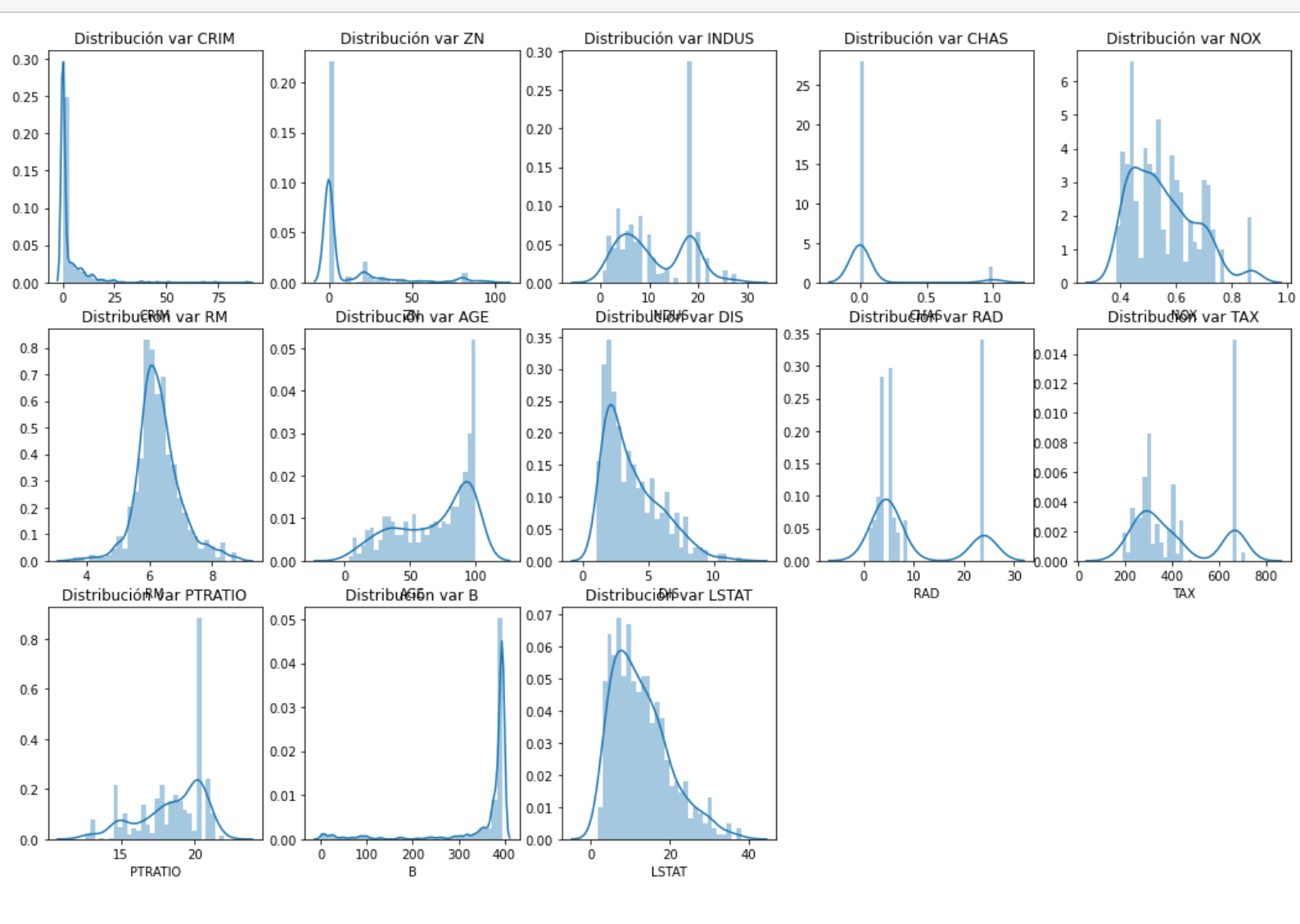
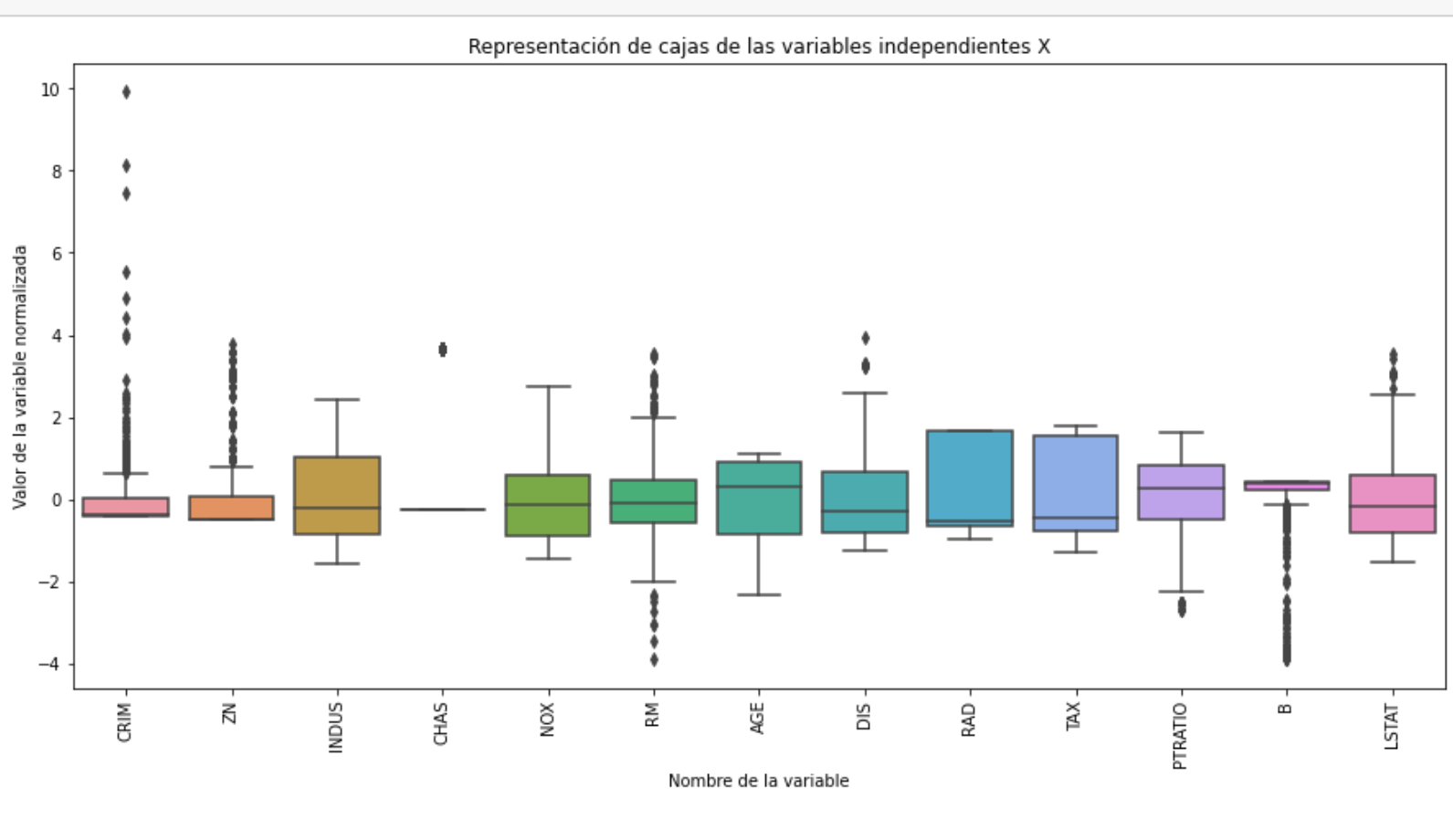


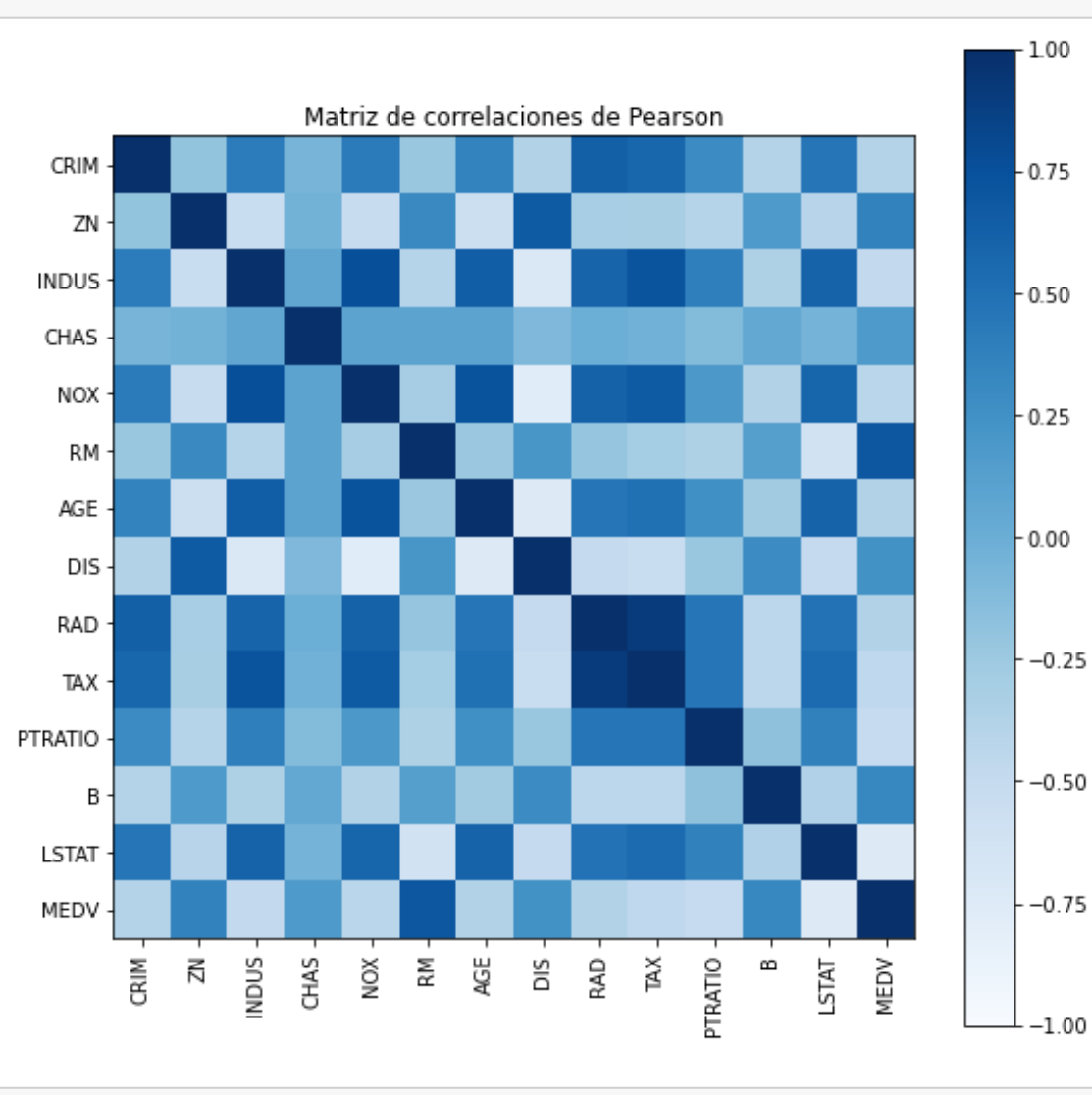
DIAGRAMA DE CAJAS DE LAS VARIABLES INDEPENDIENTES

In [11]:
```python
X_normalizado = (X-X.mean())/X.std()


plt.figure(figsize=(15,7))
ax = sns.boxplot(data=X_normalizado)
ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
plt.title(u'Representación de cajas de las variables independientes X')
plt.ylabel('Valor de la variable normalizada')
_ = plt.xlabel('Nombre de la variable')
```



Matriz de correlaciones:

In [12]:
```python
matriz_correlaciones = boston.corr(method='pearson')
n_ticks = len(boston.columns)
plt.figure( figsize=(9, 9) )
plt.xticks(range(n_ticks), boston.columns, rotation='vertical')
plt.yticks(range(n_ticks), boston.columns)
plt.colorbar(plt.imshow(matriz_correlaciones, interpolation='nearest',
                        vmin=-1., vmax=1.,
                        cmap=plt.get_cmap('Blues')))
_ = plt.title('Matriz de correlaciones de Pearson')
```



In [13]:
```python
# tabla de clasificacion de variables mas correlacionadas de mas a menos

correlaciones_target = matriz_correlaciones.values[ -1, : -1]
indices_inversos =  abs(correlaciones_target[ : ]).argsort()[ : : -1]
diccionario = {}
for nombre, correlacion in zip( boston.columns[indices_inversos], list(correlaciones_target[
indices_inversos] ) ):
    diccionario[nombre] = correlacion
pd.DataFrame.from_dict(diccionario, orient='index', columns=['Correlación con la target'])
```

Out[13]:

	Correlación con la target
LSTAT	-0.737663
RM	0.695360
PTRATIO	-0.507787
INDUS	-0.483725
TAX	-0.468536
NOX	-0.427321
CRIM	-0.388305
RAD	-0.381626
AGE	-0.376955
ZN	0.360445
B	0.333461
DIS	0.249929
CHAS	0.175260

ESTANDARIZACION DE DATOS

In [14]:
```python
obj_escalar = StandardScaler()
X_estandarizado = obj_escalar.fit_transform(X)
```

In [ ]:

Division en train y test de los datos

In [23]:
```python
X_train, X_test, Y_train, Y_test = train_test_split(X_estandarizado, Y, test_size=0.2, rando
m_state=0)
```

En vez de usar sklearn vamos a utilizar Keras y Tensorflow, ya que son mas avanzados.

In [24]:
```python
# funcion para la construccion de un modelo de regresion de redes neuronales con Kerasy Tens
orflow

def constructor_modelo():
    # Definición del modelo
    modelo = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=[X_train.shape[1]]),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)])

    # Definición del optimizador
    optimizer = tf.keras.optimizers.RMSprop(0.001)

    # Compilación del modelo
    modelo.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae', 'mse'])
    return modelo
```

In [25]:
```python
model = constructor_modelo()
```

In [26]:
```python
# Entrenamos el modelo

model.fit(X_train, Y_train)
```

```
13/13 [==============================] - 1s 2ms/step - loss: 530.2057 - mae: 20.9175 - mse: 5
30.2057
```

Out[26]:
```
<tensorflow.python.keras.callbacks.History at 0x1cfd9850670>
```

In [31]:
```python
model.predict(X_train[:11])
```

Out[31]:
```
array([[2.7540476],
       [3.2213156],
       [2.7479112],
       [2.0810788],
       [5.011512 ],
       [3.6446822],
       [2.4026396],
       [3.2118094],
       [1.4716241],
       [3.6528673],
       [1.9904481]], dtype=float32)
```

Entrenamiento del modelo¶

Entrene el modelo durante 1000 épocas y registre la precisión de entrenamiento y validación en el objeto history.

Se suele entrenar el modelo con más de una época. En este caso entrenamos el modelo con 1000 épocas. Además, hacemos uso del parámetro callbacks, que son una serie de funciones que se aplican en el entrenamiento para obtener estadísticas o parar el entrenamiento según un estado interno.

In [32]:
```python
# Muestro un punto por cada una de las épocas completadas
class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0: print('')
        print('.', end='')

EPOCHS = 1000
historico = model.fit(X_train, Y_train,
                    epochs=EPOCHS,
                    validation_split = 0.2,
                    verbose=0,
                    callbacks=[PrintDot()])
```

```

....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
```

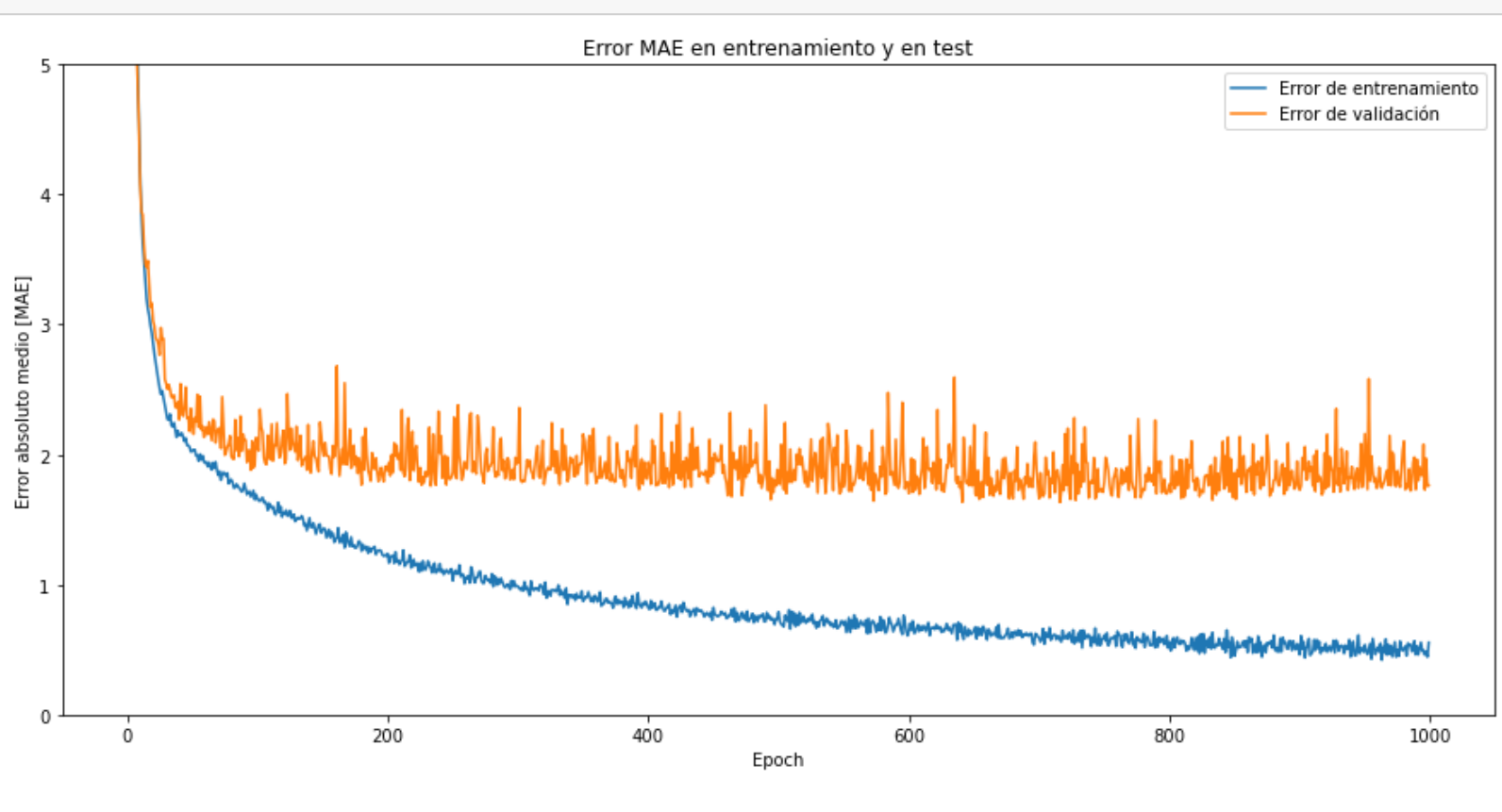
En histórico se ha guardado un objeto con el entrenamiento de la red neuronal. Se puede visualizar el progreso del entrenamiento accediendo a sus métricas y la perdida guardadas con el atributo .history.

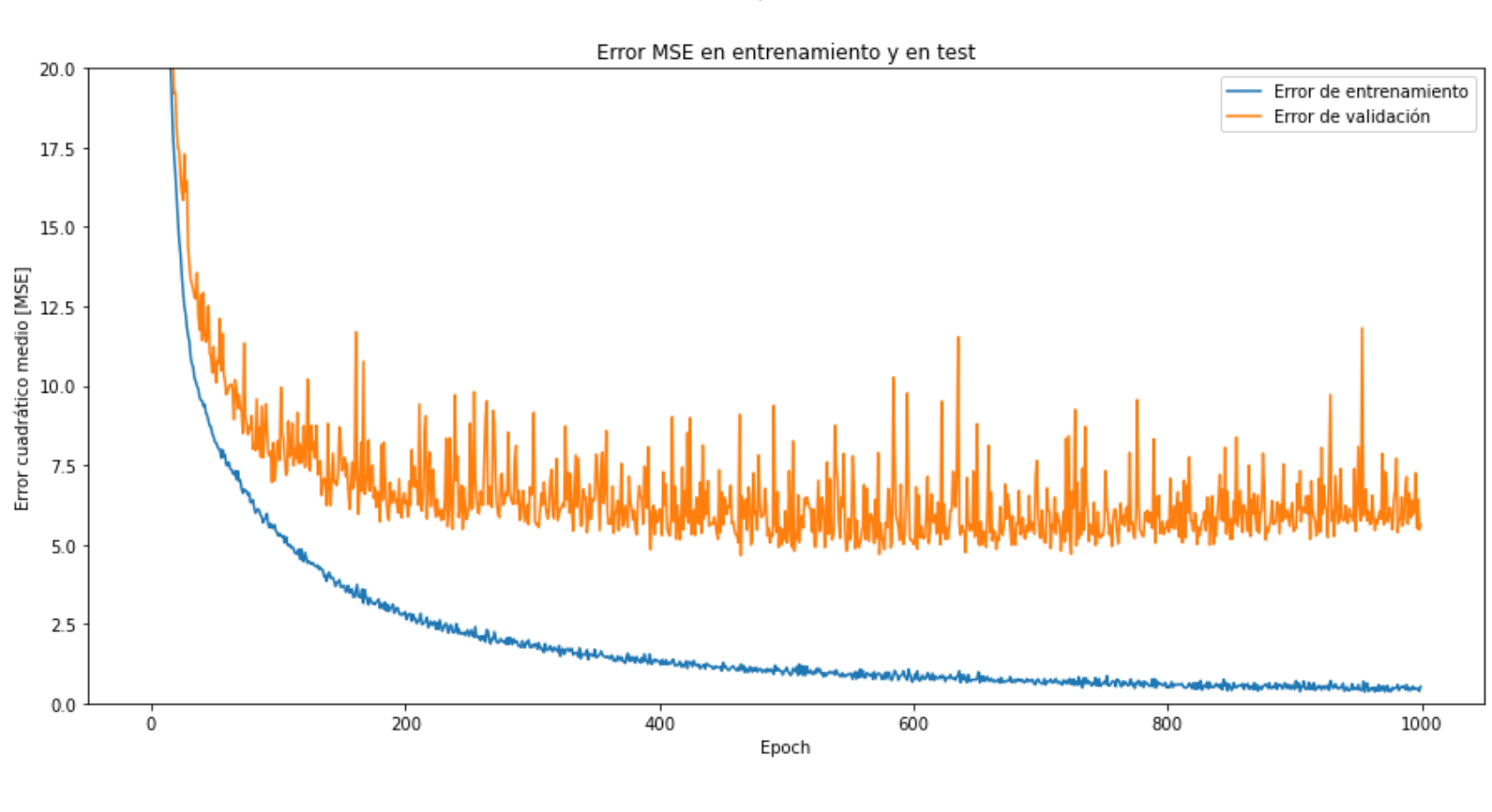
In [33]:
```python
hist = pd.DataFrame(historico.history)
hist['epoch'] = historico.epoch
hist.tail()
```

Out[33]:

	loss	mae	mse	val_loss	val_mae	val_mse	epoch
995	0.464668	0.494006	0.464668	7.249851	2.078459	7.249851	995
996	0.439597	0.459793	0.439597	5.513006	1.729518	5.513006	996
997	0.467520	0.485511	0.467520	6.426672	1.969526	6.426672	997
998	0.380176	0.443736	0.380176	5.461108	1.757322	5.461108	998
999	0.523190	0.554292	0.523190	5.625617	1.763032	5.625617	999

In [35]:
```python
def represento_historico(historico):
    hist = pd.DataFrame(historico.history)
    hist['epoch'] = historico.epoch

    plt.figure(figsize=(15,7))
    plt.xlabel('Epoch')
    plt.ylabel('Error absoluto medio [MAE]')
    plt.plot(hist['epoch'], hist['mae'],
          label='Error de entrenamiento')
    plt.plot(hist['epoch'], hist['val_mae'],
          label = 'Error de validación')
    plt.title('Error MAE en entrenamiento y en test')
    plt.ylim([0,5])
    plt.legend()

    plt.figure(figsize=(15,7))
    plt.xlabel('Epoch')
    plt.ylabel('Error cuadrático medio [MSE]')
    plt.plot(hist['epoch'], hist['mse'],
          label='Error de entrenamiento')
    plt.plot(hist['epoch'], hist['val_mse'],
          label = 'Error de validación')
    plt.title('Error MSE en entrenamiento y en test')
    plt.ylim([0,20])
    plt.legend()
    plt.show()

represento_historico(historico)
```



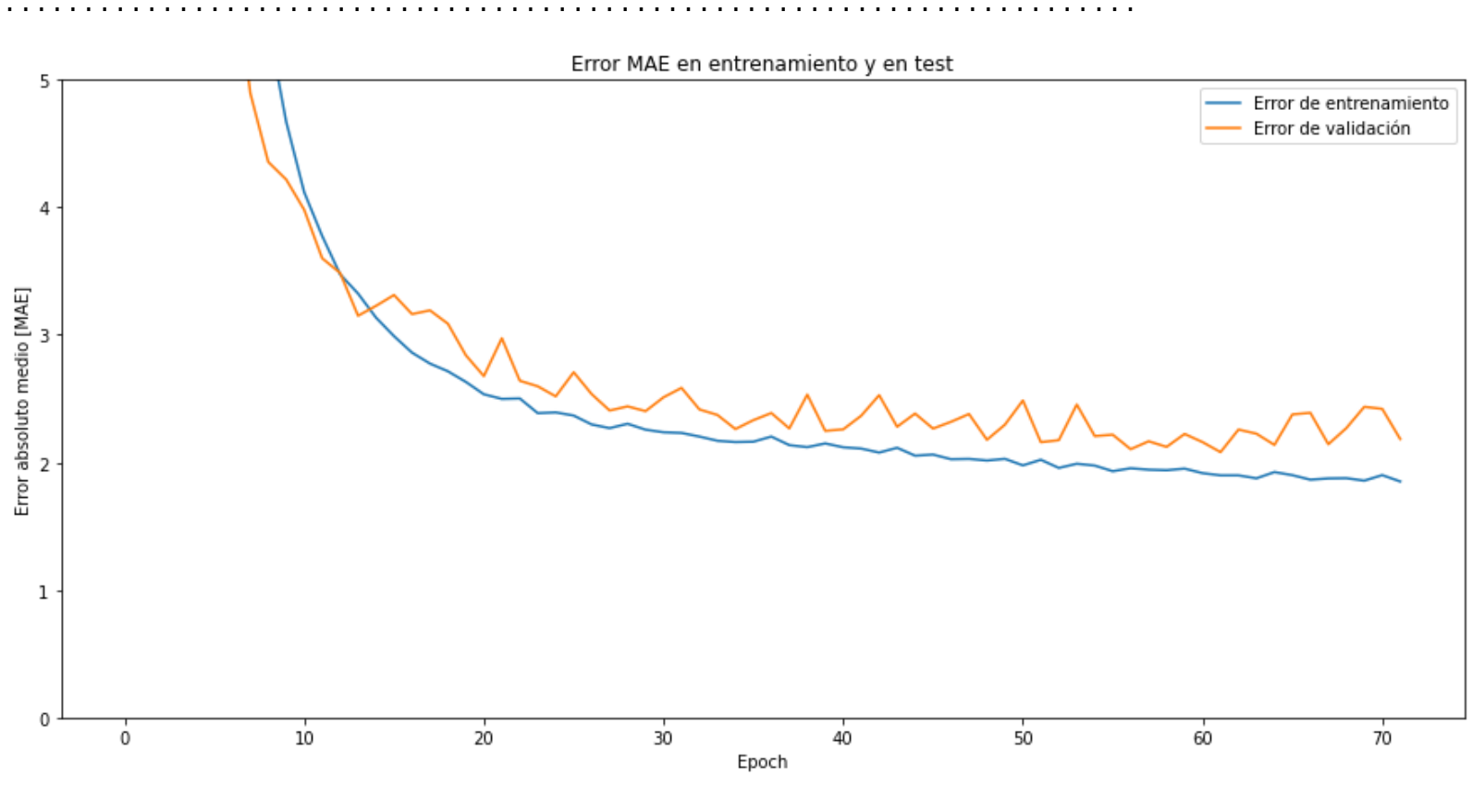


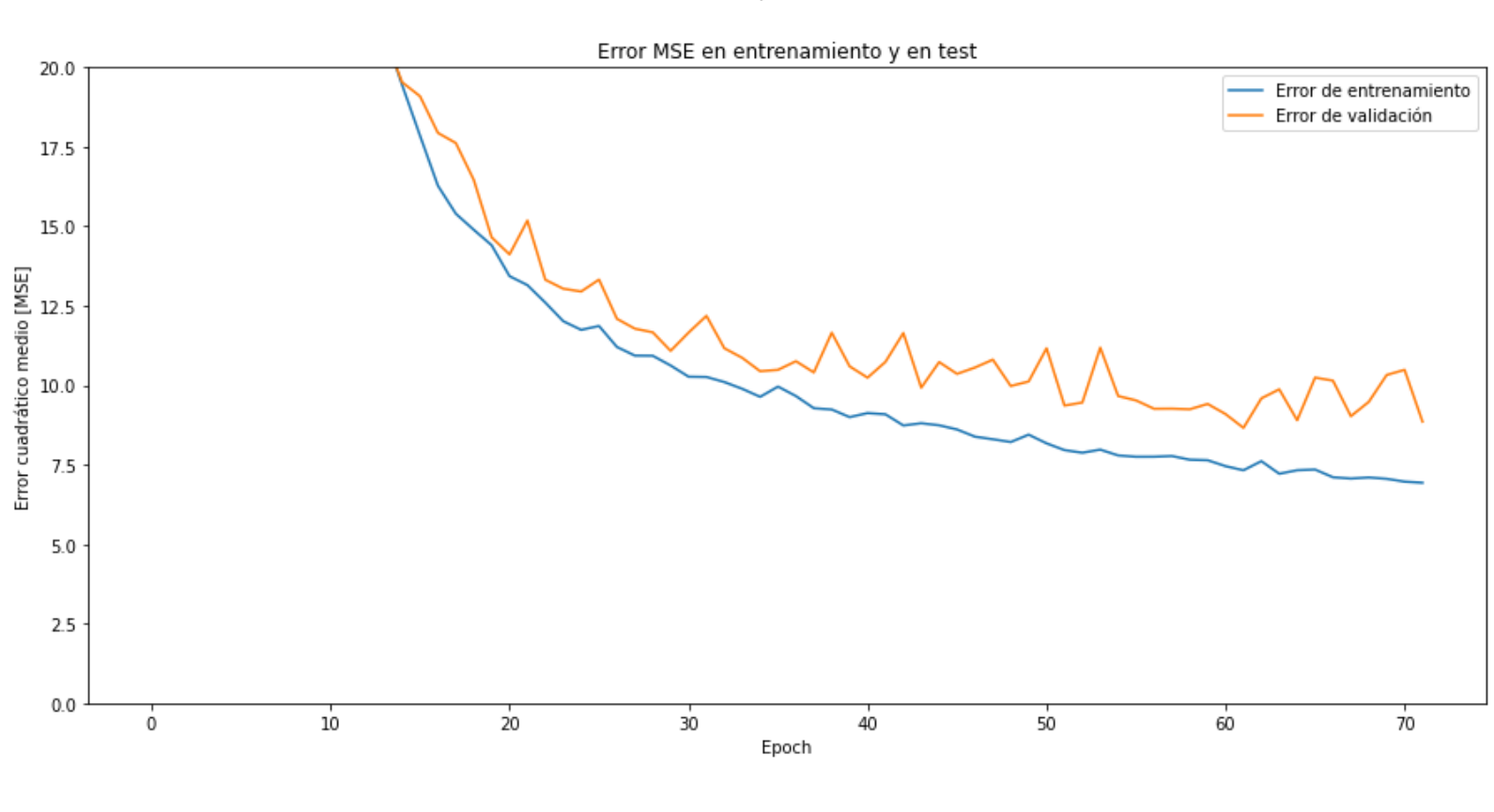
En las gráficas anteriores se puede ver poca mejora en los datos de validación a partir de un epoch, de hecho incluso se degrada. Esto se debe a que hay sobreajuste a partir de cierto punto, ya que el error en entrenamiento va disminuyendo pero en validación aumentando.

Existe un callback del método de entrenamiento denominado EarlyStopping que para el entrenamiento si después de unas épocas determinadas no hay mejoraría en validación.

In [36]:
```python
model = constructor_modelo()

# En monitor es el valor a monitorizar, que en este caso puede ser uno de los siguientes:
# [loss, mae, mse, val_loss, val_mae, val_mse]
# Patience es el número de épocas sin mejora antes de parar el entrenamiento
early_stop = keras.callbacks.EarlyStopping(monitor='val_mse', patience=10)

history = model.fit(X_train, Y_train,
                    epochs=EPOCHS,
                    validation_split = 0.2,
                    verbose=0,
                    callbacks=[early_stop, PrintDot()])

represento_historico(history)
```

```

........................................................................
```

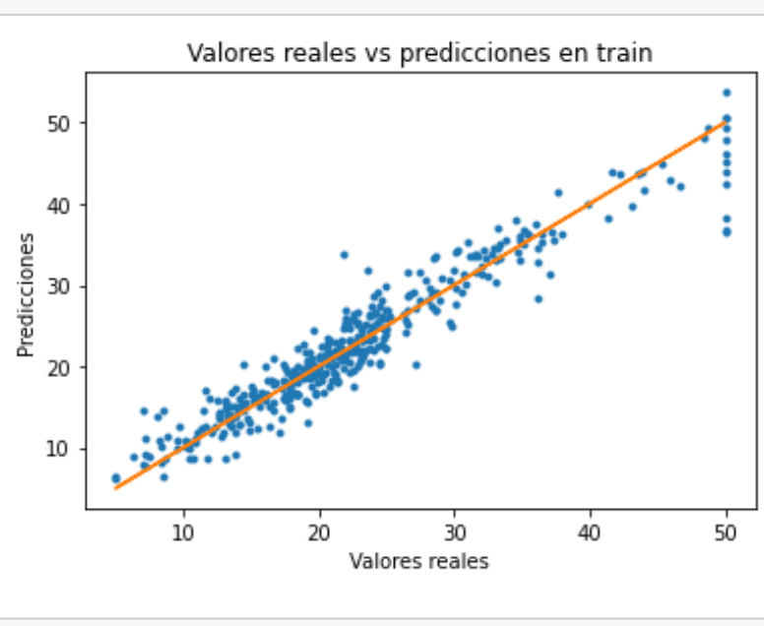




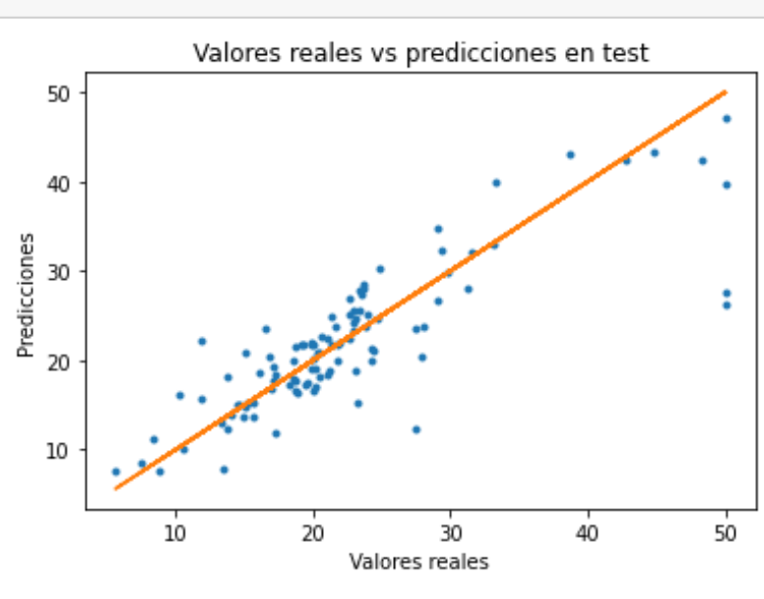
Como podemos ver, en este modelo, se ha quedado solo en 70 epocas, ya que a partir de aqui ya no mejoraba

Analizo errores:

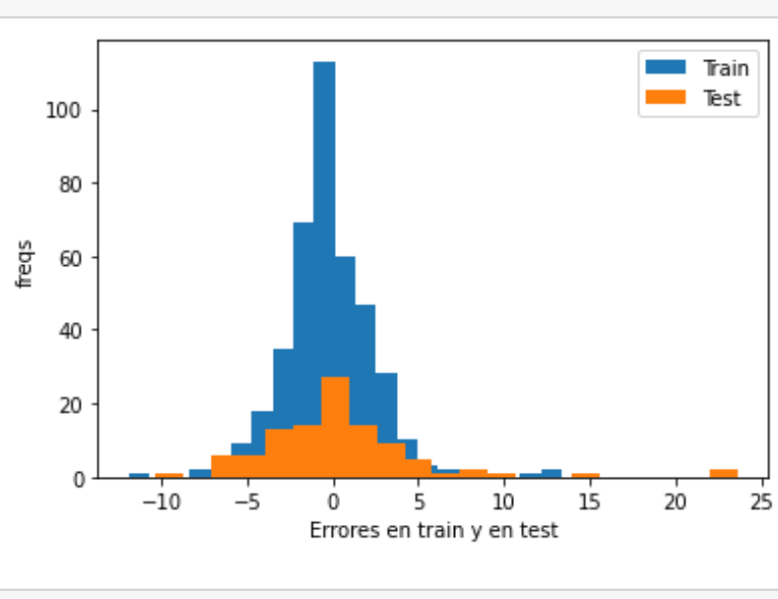
In [37]:
```python
# Valores reales versus predicciones en X_train

Y_train_pred = model.predict(X_train)
plt.title('Valores reales vs predicciones en train')
plt.xlabel('Valores reales')
plt.ylabel('Predicciones')
_ = plt.plot(Y_train, Y_train_pred, '.', Y_train, Y_train, '-')
```



In [39]:
```python
# Valores reales versus predicciones en X_test

Y_test_pred = model.predict(X_test)
plt.title('Valores reales vs predicciones en test')
plt.xlabel('Valores reales')
plt.ylabel('Predicciones')
_ = plt.plot(Y_test, Y_test_pred, '.', Y_test, Y_test, '-')
```



In [40]:
```python
# Errores en train y en test

plt.xlabel('Errores en train y en test')
plt.ylabel('freqs')
plt.hist(Y_train - Y_train_pred.flatten(), bins=21, label='Train')
plt.hist(Y_test - Y_test_pred.flatten(), bins=21, label='Test')
_ = plt.legend()
```



In [43]:
```python
# Error cuadratico medio en train y en test

error_mse_train = round(mean_squared_error(Y_train, Y_train_pred),2)
error_mse_test = round(mean_squared_error(Y_test, Y_test_pred),2)
print('El error cuadrático medio en train es: {}'.format(error_mse_train))
print('El error cuadrático medio en test es: {}'.format(error_mse_test))
```

```
El error cuadrático medio en train es: 6.96
El error cuadrático medio en test es: 23.78
```

In [ ]:

In [ ]:

In [ ]: