

## ANEXO VII: ESTUDIO VARIABLES

```
In [19]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as ticker
from statsmodels.graphics.gofplots import qqplot
from scipy.stats import normaltest
```

### Cargamos el dataset con los datos

```
In [20]: df_def = pd.read_csv(r'C:\Users\User\1.PYTHON\00.TFM\00.Archivos_utilizad
```

```
In [21]: df_def
```

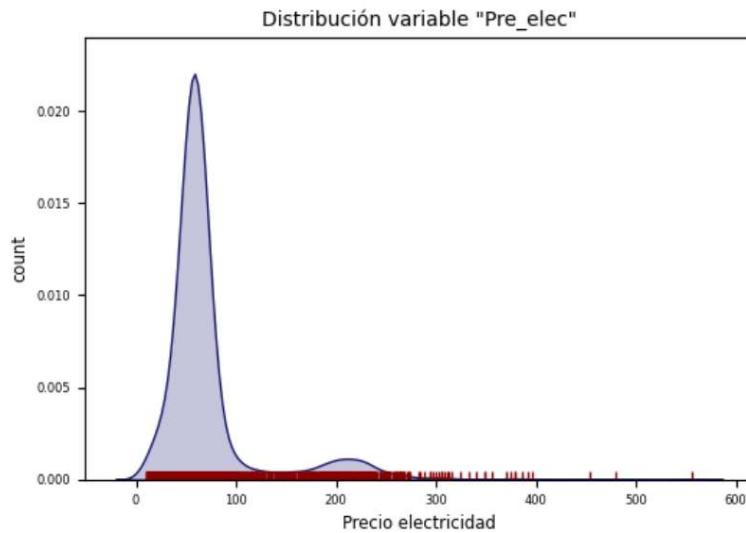
	fecha	Pre_elec	dia_sem	Dem	Prec_petr	Prec_gas	Prec_carb	Produc
0	01/01/2012	48.1992	7	20888.99	110.85	55.60	111.56	155.14
1	02/01/2012	49.7375	1	23325.20	108.35	53.75	110.20	261.03
2	03/01/2012	57.1317	2	26994.32	112.13	52.75	109.35	164.69
3	04/01/2012	54.6721	3	27934.39	113.70	53.09	109.55	164.30
4	05/01/2012	51.5471	4	28089.09	112.74	52.95	110.10	204.10
...	...	...	...	...	...	...	...	...
3647	27/12/2022	136.4629	2	23990.04	84.83	200.11	223.35	70.91
3648	28/12/2022	64.4338	3	24755.09	83.11	188.50	227.35	203.04
3649	29/12/2022	38.3779	4	25156.42	83.73	194.21	227.85	239.90
3650	30/12/2022	21.2321	5	24551.15	85.99	186.05	228.35	315.53
3651	31/12/2022	24.5088	6	21926.68	85.99	186.05	228.35	169.31

3652 rows × 28 columns

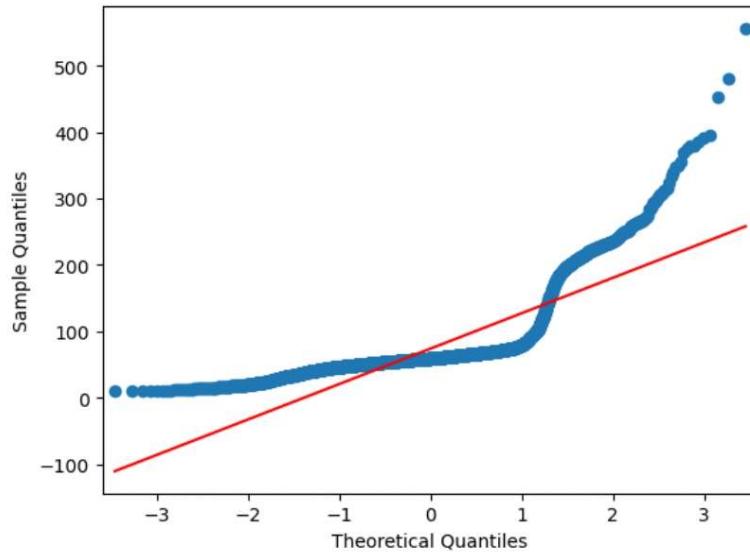
### Graficamos la distribución de la variable salida

```
In [22]: fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(6, 4))
sns.kdeplot(
    df_def['Pre_elec'],
    fill = True,
    color = "midnightblue")
sns.rugplot(
    df_def['Pre_elec'],
    color = "darkred",
    height = 0.02)
axes.set_title('Distribución variable "Pre_elec"', fontsize = 'medium')
axes.set_xlabel('Precio electricidad', fontsize='small')
axes.set_ylabel('count', fontsize='small')
axes.tick_params(labelsize = 6)
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL



```
In [27]: # Comparacion de la distribución de la variable con la distribución normal
# Representación de los cuartiles teóricos (Q-Q plot)
qqplot(df_def['Pre_elec'], line='s')
plt.show()
```



```
In [36]: # Test de DAgostino
stat, p = normaltest(df_def['Pre_elec'])
print('Estadísticos=% .3f, p=% .3f' % (stat, p))
# Interpretación
alpha = 0.05
if p > alpha:
    print('La muestra parece Gaussiana o Normal (no se rechaza la hipótesis nula)')
```

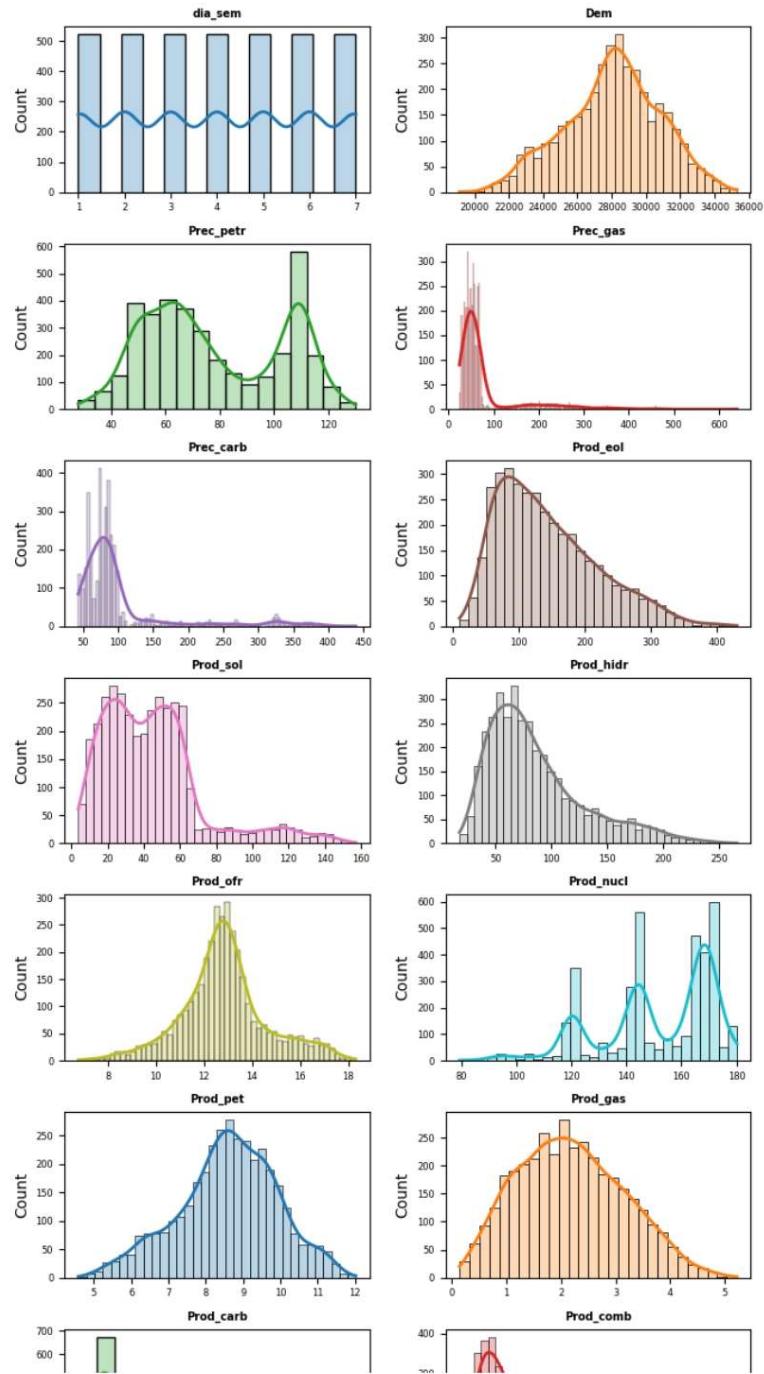
```
else:  
    print('La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)')  
  
Estadisticos=2208.295, p=0.000  
La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)
```

### Graficamos las distribuciones de las variables de entrada

```
In [5]: fig, axes = plt.subplots(nrows=7, ncols=2, figsize=(7, 15))  
axes = axes.flat  
columnas_numeric = df_def.reindex(columns = ['dia_sem', 'Dem', 'Prec_pet',  
                                         'Prod_eol', 'Prod_sol', 'Prod_pet', 'Prod_gas', 'Pr  
  
for i, colum in enumerate (columnas_numeric):  
    sns.histplot(  
        data      = df_def,  
        x         = colum,  
        stat      = "count",  
        kde       = True,  
        color     = (list(plt.rcParams['axes.prop_cycle'])*2)[i]["color"]  
        line_kws = {'linewidth': 2},  
        alpha     = 0.3,  
        ax        = axes[i]  
    )  
    axes[i].set_title(colum, fontsize = 7, fontweight = "bold")  
    axes[i].tick_params(labelsize = 6)  
    axes[i].set_xlabel("")  
  
fig.tight_layout()  
plt.subplots_adjust(top = 0.9)  
fig.suptitle('Distribución variables numéricas', fontsize = 10, fontweight = "bold")  
  
Out[5]: Text(0.5, 0.98, 'Distribución variables numéricas')
```

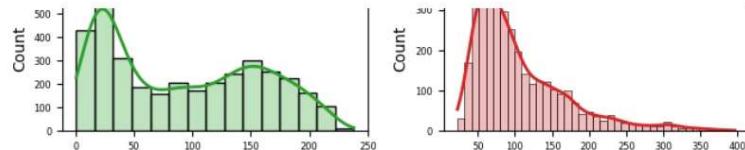
# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD EN EL MERCADO ESPAÑOL

Distribución variables numéricas



# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL



```
In [6]: fig, axes = plt.subplots(nrows=6, ncols=2, figsize=(7, 15))
axes = axes.flat
columnas_numeric = df_def.reindex(columns = ['Prod_cog', 'Prod_no_ren',
                                              'Vel_media_Alb','Vel_media_
                                              'Vel_media_Hue','Res_hidr',

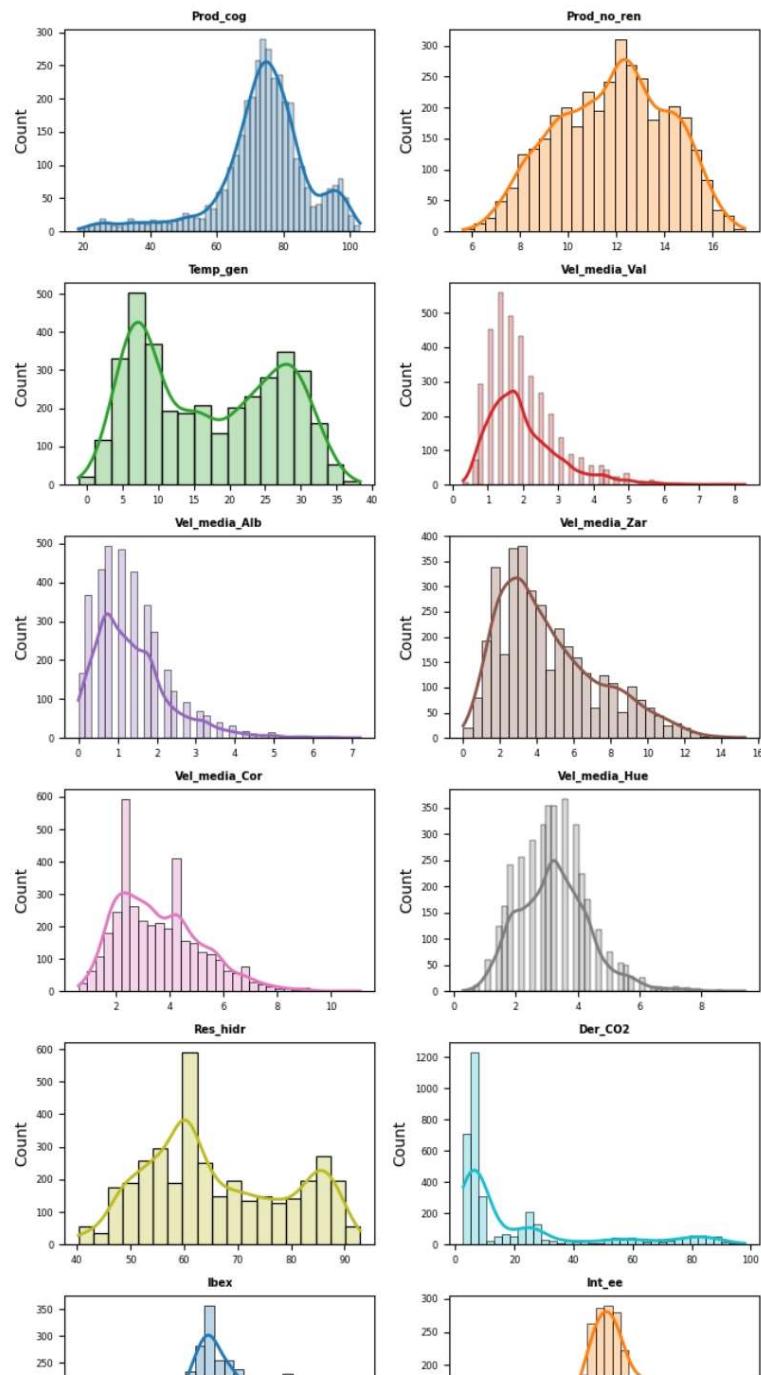
for i, colum in enumerate (columnas_numeric):
    sns.histplot(
        data      = df_def,
        x        = colum,
        stat     = "count",
        kde      = True,
        color   = (list(plt.rcParams['axes.prop_cycle'])*2)[i]["color"]
        line_kws = {'linewidth': 2},
        alpha    = 0.3,
        ax       = axes[i]
    )
    axes[i].set_title(colum, fontsize = 7, fontweight = "bold")
    axes[i].tick_params(labelsize = 6)
    axes[i].set_xlabel("")

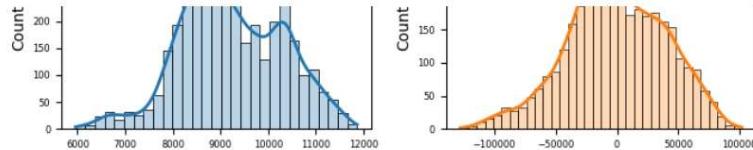
fig.tight_layout()
plt.subplots_adjust(top = 0.9)
fig.suptitle('Distribución variables numéricas', fontsize = 10, fontweig
```

Out[6]: Text(0.5, 0.98, 'Distribución variables numéricas')

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

Distribución variables numéricas





### Análisis de cada variable de entrada respecto a la variable respuesta "pre\_elec"

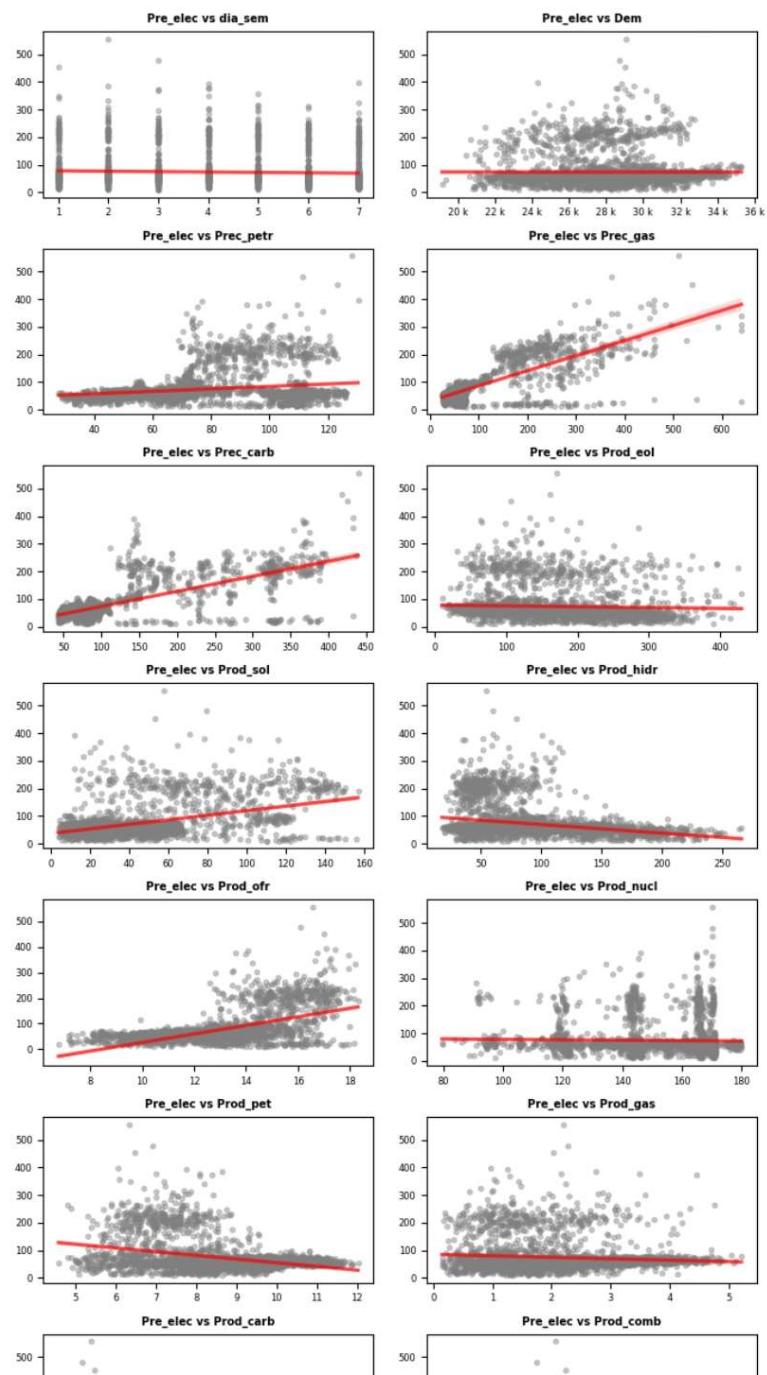
```
In [12]: fig, axes = plt.subplots(nrows=7, ncols=2, figsize=(7, 15))
axes = axes.flat
columnas_numeric = df_def.reindex(columns = ['dia_sem', 'Dem', 'Prec_pet',
                                              'Prod_eol', 'Prod_sol', 'Prod_gas',
                                              'Prod_pet', 'Prod_gas', 'Pr

for i, colum in enumerate(columnas_numeric):
    sns.regplot(
        x           = df_def[colum],
        y           = df_def['Pre_elec'],
        color       = "gray",
        marker      = '.',
        scatter_kws = {"alpha":0.4},
        line_kws    = {"color":"r","alpha":0.7},
        ax          = axes[i]
    )
    axes[i].set_title(f"Pre_elec vs {colum}", fontsize = 7, fontweight =
    axes[i].yaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].xaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].tick_params(labelsize = 6)
    axes[i].set_xlabel("")
    axes[i].set_ylabel("")

fig.tight_layout()
plt.subplots_adjust(top=0.9)
fig.suptitle('Correlación con precio', fontsize = 10, fontweight = "bold")
```

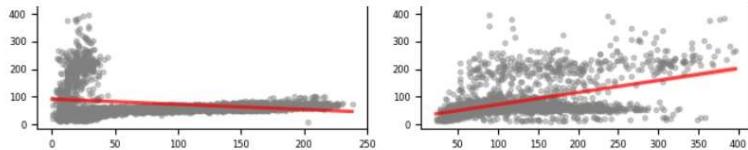
MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

Correlación con precio



# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL



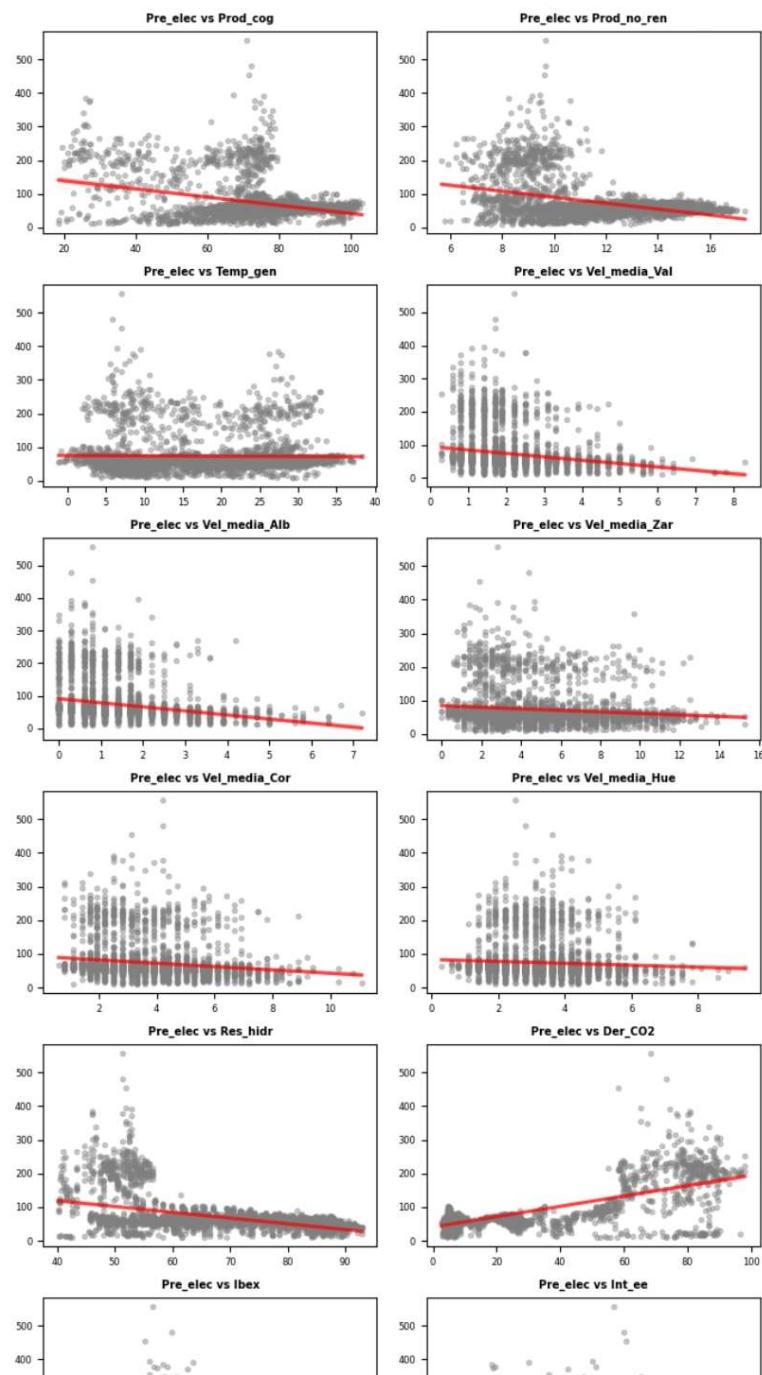
```
In [13]: fig, axes = plt.subplots(nrows=6, ncols=2, figsize=(7, 15))
axes = axes.flat
columnas_numeric = df_def.reindex(columns = ['Prod_cog', 'Prod_no_ren',
                                              'Vel_media_Alb','Vel_media_Hue','Res_hidr',
                                              'Vel_media_Hue'])

for i, colum in enumerate(columnas_numeric):
    sns.regplot(
        x           = df_def[colum],
        y           = df_def['Pre_elec'],
        color       = "gray",
        marker      = '.',
        scatter_kws = {"alpha":0.4},
        line_kws    = {"color":"r","alpha":0.7},
        ax          = axes[i])
    )
    axes[i].set_title(f"Pre_elec vs {colum}", fontsize = 7, fontweight =
#axes[i].tickLabel_format(style='sci', scilimits=(-4,4), axis='both'
    axes[i].yaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].xaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].tick_params(labelsize = 6)
    axes[i].set_xlabel("")
    axes[i].set_ylabel("")

fig.tight_layout()
plt.subplots_adjust(top=0.9)
fig.suptitle('Correlación con precio', fontsize = 10, fontweight = "bold")
```

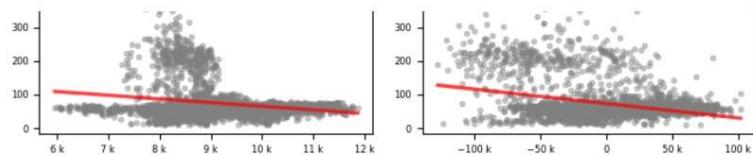
MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

Correlación con precio



## MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

### EN EL MERCADO ESPAÑOL



In [ ]:

In [ ]:

## ANEXO VIII: REGRESIÓN LINEAL (OUT-ELIM) \_\_\_\_\_

### MODELO DE REGRESIÓN LINEAL MÚLTIPLE

En este apartado estudiaremos el modelo de regresión lineal múltiple empleando el dataset creado en R con las variables seleccionadas."

#### Cargamos las librerías necesarias

```
In [1]: import pandas as pd
import numpy as np

from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.feature_selection import VarianceThreshold

from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
import statsmodels.api as sm

from scipy import stats
import time

import matplotlib.pyplot as plt
import seaborn as sns
```

#### Cargamos el dataset con los datos

Como este modelo es muy sensible a los outliers, se carga el archivo "df\_def\_outliers", cuyos datos han sido tratados para evitar la influencia negativa de estos valores

```
In [2]: df_def_outliers = pd.read_csv(r'C:\Users\User\1.PYTHON\00.TFM\00.Archivo')

In [3]: # Comprobamos Los datos cargados
print('Las dimensiones de los datos "df_def_outliers" son ' + str(df_def_outliers.shape))

Las dimensiones de los datos "df_def_outliers" son (1822, 28)

In [4]: # Comprobamos que se ha cargado bien el dataset y sus variables
df_def_outliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1822 entries, 0 to 1821
Data columns (total 28 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   fecha        1822 non-null   object  
 1   dia_sem      1822 non-null   int64  
 2   Pre_elec     1822 non-null   float64 
 3   Dem          1822 non-null   float64 
 4   Prec_petr    1822 non-null   float64 
 5   Prec_gas     1822 non-null   float64 
 6   Prec_carb    1822 non-null   float64 
 7   Prod_eol     1822 non-null   float64 
 8   Prod_sol     1822 non-null   float64 
 9   Prod_hidr    1822 non-null   float64 
 10  Prod_ofr     1822 non-null   float64 
 11  Prod_nucl    1822 non-null   float64 
 12  Prod_pet     1822 non-null   float64 
 13  Prod_gas     1822 non-null   float64 
 14  Prod_carb    1822 non-null   float64 
 15  Prod_comb    1822 non-null   float64 
 16  Prod_cog     1822 non-null   float64 
 17  Prod_no_ren  1822 non-null   float64 
 18  Temp_gen     1822 non-null   float64 
 19  Vel_media_Val 1822 non-null   float64 
 20  Vel_media_Alb 1822 non-null   float64 
 21  Vel_media_Zar 1822 non-null   float64 
 22  Vel_media_Cor 1822 non-null   float64 
 23  Vel_media_Hue 1822 non-null   float64 
 24  Res_hidr     1822 non-null   float64 
 25  Der_CO2       1822 non-null   float64 
 26  Ibex          1822 non-null   float64 
 27  Int_ee        1822 non-null   float64 
dtypes: float64(26), int64(1), object(1)
memory usage: 398.7+ KB
```

## Procesamos los datos

```
In [5]: # Definimos la variable de salida y las de entrada
X = df_def_outliers[['dia_sem', 'Dem', 'Prec_petr', 'Prec_gas', 'Prec_carb',
                     'Prod_sol', 'Prod_hidr', 'Prod_ofr', 'Prod_nucl',
                     'Prod_gas', 'Prod_carb', 'Prod_comb', 'Prod_cog',
                     'Temp_gen', 'Vel_media_Val', 'Vel_media_Alb', 'Vel_media_Zar',
                     'Vel_media_Cor', 'Vel_media_Hue', 'Res_hidr', 'Der_CO2',
                     'Int_ee']].values
y = df_def_outliers['Pre_elec'].values

In [6]: # Este modelo necesita ESTANDARIZAR los datos de las variables de entrada
X = stats.zscore(X, axis=0)

In [7]: # Dividimos los datos en "train" (80%) y "test" (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

## Desarrollo del modelo

```
In [8]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()
```

```
# Creamos el modelo
modelo_RG = linear_model.LinearRegression()

# Entrenamos el modelo
modelo_RG.fit (X_train, y_train)

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo REGRESION LINEAL MULTIPLE es: ", round(end - start,4), "segundos")
```

El tiempo empleado para desarrollar el modelo REGRESION LINEAL MULTIPLE es: 0.2862 segundos

### Comprobación del modelo

```
In [9]: # Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RG.predict(X_train), y_train), 2))
print('MSE en train:', round(mean_squared_error(modelo_RG.predict(X_train), y_train), 2))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RG.predict(X_train), y_train)), 2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RG.predict(X_test), y_test), 2))
print('MSE en test:', round(mean_squared_error(modelo_RG.predict(X_test), y_test), 2))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RG.predict(X_test), y_test)), 2))
```

MAE en train: 3.73  
MSE en train: 27.14  
RMSE en train: 5.21

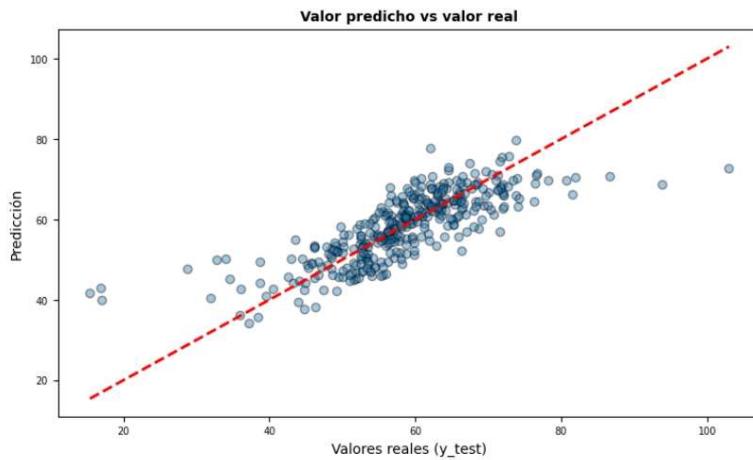
-----  
MAE en test: 4.09  
MSE en test: 32.52  
RMSE en test: 5.7

```
In [10]: # Comprobamos la precisión/rendimiento del modelo generado.
print ("La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de entrenamiento es de: ", round((modelo_RG.score (X_train, y_train))*100, 2), "%")
print ("La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de test es de: ", round((modelo_RG.score (X_test, y_test))*100, 2), "%")
```

La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de entrenamiento es de: 73.64 %  
La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de test es de: 68.05 %

```
In [11]: # Graficamos la relación entre los valores reales de "test" y los predichos
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = cross_val_predict(
    estimator = modelo_RG,
    X         = X_test,
    y         = y_test,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_test_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```



```
In [12]: MAPE_1 = (y_test - y_test_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
MAPE en test: 8.77 %
```

### Mismo modelo con diferente división (0.3 / 0.7)

```
In [13]: # Dividimos los datos en "train" y "test"
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split (X, y, test_
In [14]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RG_2 = linear_model.LinearRegression()

# Entrenamos el modelo
modelo_RG_2.fit (X_train_2, y_train_2)

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo REGRESION LINEAL MULTIPLE es: ", round(end - start,4), "segundos")
```

El tiempo empleado para desarrollar el modelo REGRESION LINEAL MULTIPLE es: 0.0449 segundos

```
In [15]: # Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RG_2.predict(X_t
print('MSE en train:', round(mean_squared_error(modelo_RG_2.predict(X_tr
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RG_2.pre
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RG_2.predict(X_te
print('MSE en test:', round(mean_squared_error(modelo_RG_2.predict(X_te
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RG_2.pre
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

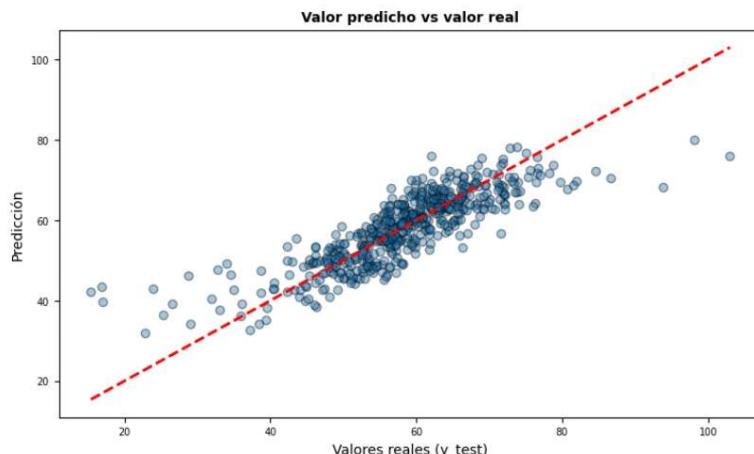
```
MAE en train: 3.74
MSE en train: 27.56
RMSE en train: 5.25
-----
MAE en test: 3.97
MSE en test: 30.07
RMSE en test: 5.48
```

```
In [16]: # Comprobamos la precisión/rendimiento del modelo generado.
print ("La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de entrenamiento es de: " + str(round((modelo_RG_2.score (X_train_2, y_train_2))*100, 2)) + "%")
print ("La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de prueba es de: " + str(round((modelo_RG_2.score (X_test_2, y_test_2))*100, 2)) + "%"))

La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de entrenamiento es de: 72.93 %
La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de prueba es de: 71.33 %
```

```
In [17]: # Graficamos la relación entre los valores reales de "test" y las predicciones
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_2_pred = cross_val_predict(
    estimator = modelo_RG_2,
    X         = X_test_2,
    y         = y_test_2,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test_2, y_test_2_pred, edgecolors = (0, 0, 0), alpha = 0.5)
axes.plot([y_test_2.min(), y_test_2.max()],
          [y_test_2.min(), y_test_2.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```



```
In [18]: MAPE_1 = (y_test_2 - y_test_2_pred)/y_test_2
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

MAPE en test: 8.03 %
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

---

In [ ]:

In [ ]:

## ANEXO IX: REGRESION LINEAL (OUT-LIM)

---

### 1 MODELO DE REGRESION LINAL MULTIPLE

*En este apartado estudiaremos el modelo de regresion lineal multiple empleando el dataset creado en R con las variables seleccionadas."*

#### 1.1 Cargamos las librerias necesarias

In [1]:

```
1 import pandas as pd
2 import numpy as np
3
4 from sklearn import linear_model
5 from sklearn.model_selection import train_test_split
6 from sklearn.metrics import mean_squared_error, mean_absolute_error
7
8 from sklearn.model_selection import cross_val_predict
9 from sklearn.model_selection import KFold
10 import statsmodels.api as sm
11
12 from scipy import stats
13 import time
14
15 import matplotlib.pyplot as plt
16 import seaborn as sns
```

#### 1.2 Cargamos el dataset con los datos

Como este modelo es muy sensible a los outliers, se carga el archivo "df\_def\_outliers", cuyos datos han sido tratados para evitar la influencia negativa de estos valores

In [2]:

```
1 df_def_outliers = pd.read_csv(r'C:\Users\User\1.PYTHON\00.TFM\00.Archivos\df_def_outliers.csv')
```

In [3]:

```
1 # Comprobamos Los datos cargados
2 print('Las dimensiones de los datos "df_def_outliers" son '+ str(df_def_outliers.shape))
```

Las dimensiones de los datos "df\_def\_outliers" son (3652, 28)



In [4]:

```
v 1 # Comprobamos que se ha cargado bien el dataset y sus variables
2 df_def_outliers.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3652 entries, 0 to 3651
Data columns (total 28 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   fecha             3652 non-null    object  
 1   Pre_elec          3652 non-null    float64
 2   dia_sem           3652 non-null    int64  
 3   Dem               3652 non-null    float64
 4   Prec_petr         3652 non-null    float64
 5   Prec_gas          3652 non-null    float64
 6   Prec_carb         3652 non-null    float64
 7   Prod_eol          3652 non-null    float64
 8   Prod_sol          3652 non-null    float64
 9   Prod_hidr         3652 non-null    float64
 10  Prod_ofr          3652 non-null    float64
 11  Prod_nucl         3652 non-null    float64
 12  Prod_pet          3652 non-null    float64
 13  Prod_gas          3652 non-null    float64
 14  Prod_carb         3652 non-null    float64
 15  Prod_comb          3652 non-null    float64
 16  Prod_cog          3652 non-null    float64
 17  Prod_no_ren       3652 non-null    float64
 18  Temp_gen          3652 non-null    float64
 19  Vel_media_Val     3652 non-null    float64
 20  Vel_media_Alb     3652 non-null    float64
 21  Vel_media_Zar     3652 non-null    float64
 22  Vel_media_Cor     3652 non-null    float64
 23  Vel_media_Hue     3652 non-null    float64
 24  Res_hidr          3652 non-null    float64
 25  Der_CO2            3652 non-null    float64
 26  Ibex              3652 non-null    float64
 27  Int_ee             3652 non-null    float64
dtypes: float64(26), int64(1), object(1)
memory usage: 799.0+ KB
```

### 1.3 Procesamos los datos

In [5]:

```
v 1 # Definimos la variable de salida y las de entrada
2 X = df_def_outliers[['dia_sem', 'Dem', 'Prec_petr', 'Prec_gas', 'Prec_ca
3   'Prod_sol', 'Prod_hidr', 'Prod_ofr', 'Prod_nucl', '
4   'Prod_gas', 'Prod_carb', 'Prod_comb', 'Prod_cog', '
5   'Temp_gen', 'Vel_media_Val', 'Vel_media_Alb', 'Vel_
6   'Vel_media_Cor', 'Vel_media_Hue', 'Res_hidr', 'Der_
7   'Int_ee']].values
8 y = df_def_outliers ['Pre_elec'].values
```

In [6]:

```
1 # Este modelo necesita ESTANDARIZAR los datos de las variables de entrada
2 X = stats.zscore(X, axis=0)
```

In [7]:

```
1 # Dividimos los datos en "train" (80%) y "test" (20%)
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
```

## 1.4 Desarrollo del modelo

In [8]:

```
1 # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
2 start = time.time()
3
4 # Creamos el modelo
5 modelo_RG = linear_model.LinearRegression()
6
7 # Entrenamos el modelo
8 modelo_RG.fit (X_train, y_train)
9
10 # Medimos el tiempo que ha tardado el modelo
11 end = time.time()
12 print("El tiempo empleado para desarrollar el modelo REGRESION LINEAL MULTIPLE es: ", round(end - start,4), "segundos")
```

El tiempo empleado para desarrollar el modelo REGRESION LINEAL MULTIPLE es: 0.0843 segundos

## 1.5 Comprobacion del modelo

In [9]:

```
1 # Obtenemos las diferentes metricas del error de "train" y "test"
2 print('MAE en train:', round(mean_absolute_error(modelo_RG.predict(X_train), y_train)))
3 print('MSE en train:', round(mean_squared_error(modelo_RG.predict(X_train), y_train)))
4 print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RG.predict(X_train), y_train))))
5 print("-----")
6 print('MAE en test:', round(mean_absolute_error(modelo_RG.predict(X_test), y_test)))
7 print('MSE en test:', round(mean_squared_error(modelo_RG.predict(X_test), y_test)))
8 print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RG.predict(X_test), y_test))))
```

MAE en train: 18.11  
MSE en train: 1069.69  
RMSE en train: 32.71  
-----  
MAE en test: 18.11  
MSE en test: 1176.19  
RMSE en test: 34.3

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

In [10]:

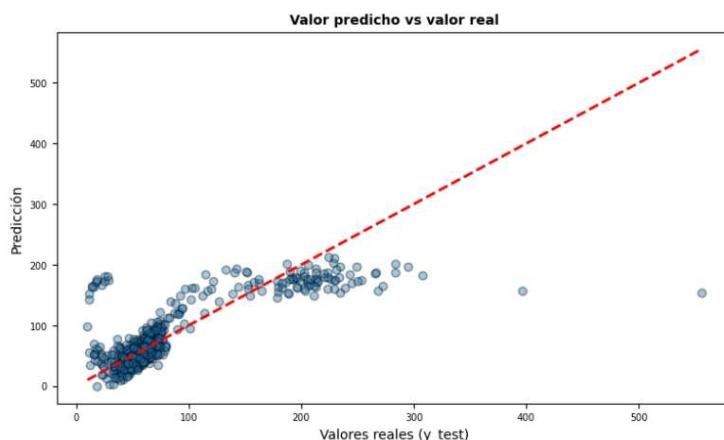
```
1 # Comprobamos la precisión/rendimiento del modelo generado.
2 print ("La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de entrenamiento es de: " + str(round((modelo_RG.score (X_train, y_train))*100, 2)) + "%")
3 print ("La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de test es de: " + str(round((modelo_RG.score (X_test, y_test))*100, 2)) + "%")
```

La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de entrenamiento es de: 61.39 %

La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de test es de: 62.52 %

In [11]:

```
1 # Graficamos la relación entre los valores reales de "test" y los predichos
2 cv = KFold(n_splits=5, random_state=123, shuffle=True)
3 y_test_pred = cross_val_predict(
4     estimator = modelo_RG,
5     X         = X_test,
6     y         = y_test,
7     cv        = cv
8 )
9 fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
10 axes.scatter(y_test, y_test_pred, edgecolors = (0, 0, 0), alpha = 0.4)
11 axes.plot([y_test.min(), y_test.max()],
12           [y_test.min(), y_test.max()],
13           '--', lw=2, color = 'red')
14 axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
15 axes.set_xlabel('Valores reales (y_test)')
16 axes.set_ylabel('Predicción')
17 axes.tick_params(labelsize = 7)
```



In [12]:

```

1 MAPE_1 = (y_test - y_test_pred)/y_test
2 MAPE_1 = np.sqrt(MAPE_1**2)
3 print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

```

MAPE en test: 39.79 %

## 1.6 Mismo modelo con diferente division (0.3 / 0.7)

In [13]:

```

1 # Dividimos las datos en "train" y "test"
2 X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split (X, y, test_s

```

In [14]:

```

1 # Calcularemos el tiempo que tarda la computadora para desarrollar el mo
2 start = time.time()
3
4 # Creamos el modelo
5 modelo_RG_2 = linear_model.LinearRegression()
6
7 # Entrenamos el modelo
8 modelo_RG_2.fit (X_train_2, y_train_2)
9
10 # Medimos el tiempo que ha tardado el modelo
11 end = time.time()
12 print("El tiempo empleado para desarrollar el modelo REGRESION LINEAL MUL
13     round(end - start,4), "segundos")

```

El tiempo empleado para desarrollar el modelo REGRESION LINEAL MULTIPLE es: 0.007 segundos

In [15]:

```

1 # Obtenemos las diferentes metricas del error de "train" y "test"
2 print('MAE en train:', round(mean_absolute_error(modelo_RG_2.predict(X_tr
3 print('MSE en train:', round(mean_squared_error(modelo_RG_2.predict(X_tr
4 print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RG_2.pred
5 print("-----")
6 print('MAE en test:', round(mean_absolute_error(modelo_RG_2.predict(X_te
7 print('MSE en test:', round(mean_squared_error(modelo_RG_2.predict(X_te
8 print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RG_2.pred

```

MAE en train: 17.54  
 MSE en train: 979.22  
 RMSE en train: 31.29  
 -----  
 MAE en test: 18.7  
 MSE en test: 1368.13  
 RMSE en test: 36.99

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

In [16]:

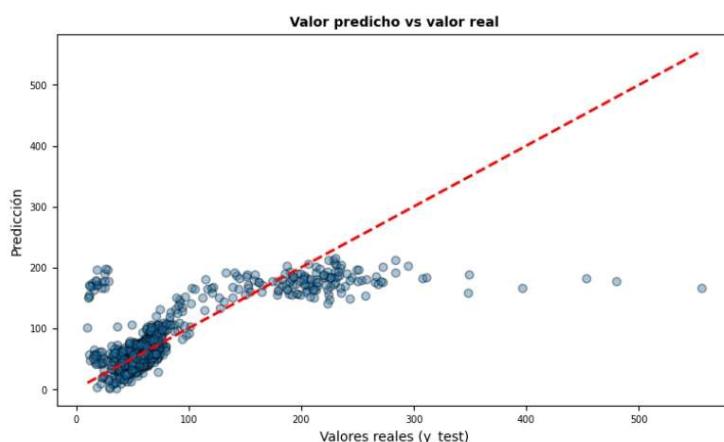
```
1 # Comprobamos la precisión/rendimiento del modelo generado.
2 print ("La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de entrenamiento es de: " + str(round((modelo_RG_2.score (X_train_2, y_train_2))*100, 2)) + "%")
3 print ("La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de test es de: " + str(round((modelo_RG_2.score (X_test_2, y_test_2))*100, 2)) + "%")
```

La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de entrenamiento es de: 62.43 %

La precisión del modelo de REGRESIÓN LINEAL MULTIPLE con los datos de test es de: 59.7 %

In [17]:

```
1 # Graficamos la relación entre los valores reales de "test" y los predichos
2 cv = KFold(n_splits=5, random_state=123, shuffle=True)
3 y_test_2_pred = cross_val_predict(
4     estimator = modelo_RG_2,
5     X         = X_test_2,
6     y         = y_test_2,
7     cv        = cv)
8
9 fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
10 axes.scatter(y_test_2, y_test_2_pred, edgecolors = (0, 0, 0), alpha = 0.4)
11 axes.plot([y_test_2.min(), y_test_2.max()],
12           [y_test_2.min(), y_test_2.max()],
13           '--', lw=2, color = 'red')
14 axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
15 axes.set_xlabel('Valores reales (y_test)')
16 axes.set_ylabel('Predicción')
17 axes.tick_params(labelsize = 7)
```



In [18]:

```
1 MAPE_1 = (y_test_2 - y_test_2_pred)/y_test_2
2 MAPE_1 = np.sqrt(MAPE_1**2)
3 print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 42.36 %

In [ ]:

```
1
```

In [ ]:

```
1
```

## ANEXO X: KNN (OUT\_ELIM)

---

### MODELO DE K-VECINOS MAS CERCANOS

En este apartado estudiaremos el modelo de k-vecinos mas cercanos empleando el dataset creado en R con las variables seleccionadas."

#### Cargamos las librerías necesarias

```
In [1]: import pandas as pd
import numpy as np

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error

from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold

from scipy import stats
import time

import matplotlib.pyplot as plt
import seaborn as sns
```

#### Cargamos el dataset con los datos

```
In [2]: df_def_outliers = pd.read_csv(r'C:\Users\User\1.PYTHON\00.TFM\00.Archivo')

In [3]: # Comprobamos Los datos cargados
print('Las dimensiones de los datos "df_def_outliers" son ' + str(df_def_
Las dimensiones de los datos "df_def_outliers" son (3652, 28)

In [4]: # Comprobamos que se ha cargado bien el dataset y sus variables
df_def_outliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3652 entries, 0 to 3651
Data columns (total 28 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   fecha        3652 non-null   object  
 1   Pre_elec     3652 non-null   float64 
 2   dia_sem      3652 non-null   int64  
 3   Dem          3652 non-null   float64 
 4   Prec_petr    3652 non-null   float64 
 5   Prec_gas     3652 non-null   float64 
 6   Prec_carb    3652 non-null   float64 
 7   Prod_eol     3652 non-null   float64 
 8   Prod_sol     3652 non-null   float64 
 9   Prod_hidr    3652 non-null   float64 
 10  Prod_ofr     3652 non-null   float64 
 11  Prod_nucl    3652 non-null   float64 
 12  Prod_pet     3652 non-null   float64 
 13  Prod_gas     3652 non-null   float64 
 14  Prod_carb    3652 non-null   float64 
 15  Prod_comb    3652 non-null   float64 
 16  Prod_cog     3652 non-null   float64 
 17  Prod_no_ren  3652 non-null   float64 
 18  Temp_gen     3652 non-null   float64 
 19  Vel_media_Val 3652 non-null   float64 
 20  Vel_media_Alb 3652 non-null   float64 
 21  Vel_media_Zar 3652 non-null   float64 
 22  Vel_media_Cor 3652 non-null   float64 
 23  Vel_media_Hue 3652 non-null   float64 
 24  Res_hidr     3652 non-null   float64 
 25  Der_CO2       3652 non-null   float64 
 26  Ibex          3652 non-null   float64 
 27  Int_ee        3652 non-null   float64 
dtypes: float64(26), int64(1), object(1)
memory usage: 799.0+ KB
```

## Procesamos los datos

```
In [5]: # Definimos la variable de salida y las de entrada
X = df_def_outliers[['dia_sem', 'Dem', 'Prec_petr', 'Prec_gas', 'Prec_carb',
                     'Prod_sol', 'Prod_hidr', 'Prod_ofr', 'Prod_nucl',
                     'Prod_gas', 'Prod_carb', 'Prod_comb', 'Prod_cog',
                     'Temp_gen', 'Vel_media_Val', 'Vel_media_Alb', 'Vel_media_Zar',
                     'Vel_media_Cor', 'Vel_media_Hue', 'Res_hidr', 'Der_CO2',
                     'Int_ee']].values
y = df_def_outliers['Pre_elec'].values

In [6]: # Este modelo necesita ESTANDARIZAR los datos de las variables de entrada
X = stats.zscore(X, axis=0)

In [7]: # Dividimos los datos en "train" (80%) y "test" (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

## Desarrollo del modelo

```
In [8]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()
```

```
# Creamos el modelo
modelo_KNN = KNeighborsRegressor()

# Obtenemos el mejor número de vecinos con Validacion Cruzada
parametros = {'n_neighbors': range(1, 20)}
modelo_KNN_optimo = GridSearchCV (modelo_KNN, parametros, cv=3)

# Entrenamos el modelo
modelo_KNN_optimo.fit (X_train, y_train)

# Obtenemos el mejor estimador
print('El mejor estimador es: {}'.format(modelo_KNN_optimo.best_estimator_))

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo KNN es: ",
      round(end - start,4), "segundos")
```

El mejor estimador es: KNeighborsRegressor(n\_neighbors=14)  
 El tiempo empleado para desarrollar el modelo KNN es: 3.7156 segundos

## Comprobacion del modelo

```
In [9]: # Obtenemos Las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_KNN_optimo.predict(X_train), y_train), 2))
print('MSE en train:', round(mean_squared_error(modelo_KNN_optimo.predict(X_train), y_train), 2))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_KNN_optimo.predict(X_train), y_train)), 2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_KNN_optimo.predict(X_test), y_test), 2))
print('MSE en test:', round(mean_squared_error(modelo_KNN_optimo.predict(X_test), y_test), 2))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_KNN_optimo.predict(X_test), y_test)), 2))

MAE en train: 9.7
MSE en train: 647.35
RMSE en train: 25.44
-----
MAE en test: 10.21
MSE en test: 773.79
RMSE en test: 27.82
```

```
In [10]: # Comprobamos La precision/rendimiento del modelo generado.
print ("La precisión del modelo K-VECINOS MAS CERCANOS con los datos de entrenamiento es de: {} %".format(round((modelo_KNN_optimo.score (X_train, y_train))*100, 2)))
print ("La precisión del modelo K-VECINOS MAS CERCANOS con los datos de test es de: {} %".format(round((modelo_KNN_optimo.score (X_test, y_test))*100, 2)))
```

La precisión del modelo K-VECINOS MAS CERCANOS con los datos de entrenamiento es de: 76.63 %  
 La precisión del modelo K-VECINOS MAS CERCANOS con los datos de test es de: 75.35 %

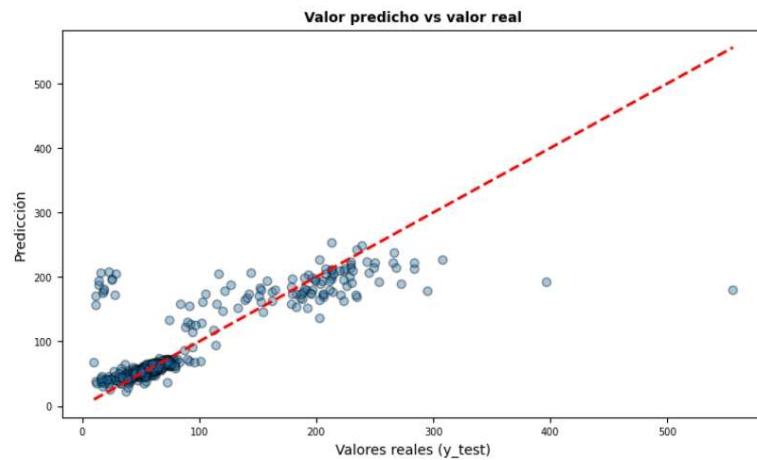
```
In [11]: # Graficamos la relacion entre los valores reales de "test" y Los predicion
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = cross_val_predict(
    estimator = modelo_KNN_optimo,
    X         = X_test,
    y         = y_test,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_test_pred, edgecolors = (0, 0, 0), alpha = 0.4)
```

```

axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

```



```

In [12]: MAPE_1 = (y_test - y_test_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

```

MAPE en test: 30.67 %

### Mismo modelo con diferente división (0.3 / 0.7)

```

In [13]: # Dividimos las datos en "train" y "test"
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split (X, y, test_

```

```

In [14]: # Calcularemos el tiempo que tarda la computadora para desarrollar el mo
start = time.time()

# Creamos el modelo
modelo_KNN_2 = KNeighborsRegressor()

# Obtenemos el mejor número de vecinos con Validación Cruzada
parametros = {'n_neighbors': range(1, 20)}
modelo_KNN_2_optimo = GridSearchCV (modelo_KNN_2, parametros, cv=3)

# Entrenamos el modelo
modelo_KNN_2_optimo.fit (X_train_2, y_train_2)

# Obtenemos el mejor estimador
print('El mejor estimador es: {}'.format(modelo_KNN_2_optimo.best_estima

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo KNN es:", round(end

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
El mejor estimador es: KNeighborsRegressor(n_neighbors=6)
El tiempo empleado para desarrollar el modelo KNN es: 2.5987 segundos
```

```
In [15]: # Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_KNN_2_optimo.predict(X_train_2), y_train_2)), 2)
print('MSE en train:', round(mean_squared_error(modelo_KNN_2_optimo.predict(X_train_2), y_train_2)), 2)
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_KNN_2_optimo.predict(X_train_2), y_train_2))), 2)
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_KNN_2_optimo.predict(X_test_2), y_test_2)), 2)
print('MSE en test:', round(mean_squared_error(modelo_KNN_2_optimo.predict(X_test_2), y_test_2)), 2)
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_KNN_2_optimo.predict(X_test_2), y_test_2))), 2)

MAE en train: 8.82
MSE en train: 534.55
RMSE en train: 23.12
-----
MAE en test: 10.88
MSE en test: 900.8
RMSE en test: 30.01
```

```
In [16]: # Comprobamos la precision/rendimiento del modelo generado.
print ("La precision del modelo de REGRESION LINEAL MULTIPLE con los datos de entrenamiento es de: " + str(round((modelo_KNN_2_optimo.score (X_train_2, y_train_2))*100, 2)))
print ("La precision del modelo de REGRESION LINEAL MULTIPLE con los datos de test es de: " + str(round((modelo_KNN_2_optimo.score (X_test_2, y_test_2))*100, 2)))
```

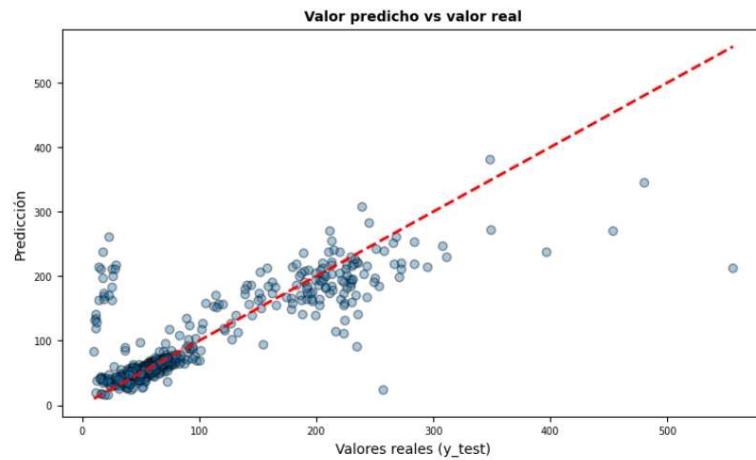
```
La precision del modelo de REGRESION LINEAL MULTIPLE con los datos de entrenamiento es de: 79.49 %
La precision del modelo de REGRESION LINEAL MULTIPLE con los datos de test es de: 73.46 %
```

```
In [17]: # Graficamos la relacion entre los valores reales de "test" y Los predicion
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_2_pred = cross_val_predict(
    estimator = modelo_KNN_2_optimo,
    X        = X_test_2,
    y        = y_test_2,
    cv      = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test_2, y_test_2_pred, edgecolors = (0, 0, 0), alpha = 0.8)
axes.plot([y_test_2.min(), y_test_2.max()], [y_test_2.min(), y_test_2.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Prediccion')
axes.tick_params(labelsize = 7)
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL



```
In [18]: MAPE_1 = (y_test_2 - y_test_2_pred)/y_test_2
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 30.0 %

In [ ]:

In [ ]:

## ANEXO XI: KNN (OUT-LIM) \_\_\_\_\_

### MODELO DE K-VECINOS MAS CERCANOS

En este apartado estudiaremos el modelo de k-vecinos mas cercanos empleando el dataset creado en R con las variables seleccionadas."

#### Cargamos las librerías necesarias

```
In [1]: import pandas as pd
import numpy as np

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error

from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold

from scipy import stats
import time

import matplotlib.pyplot as plt
import seaborn as sns
```

#### Cargamos el dataset con los datos

```
In [2]: df_def_outliers = pd.read_csv(r'C:\Users\User\1.PYTHON\00.TFM\00.Archivo')

In [3]: # Comprobamos Los datos cargados
print('Las dimensiones de los datos "df_def_outliers" son ' + str(df_def_
Las dimensiones de los datos "df_def_outliers" son (1822, 28)

In [4]: # Comprobamos que se ha cargado bien el dataset y sus variables
df_def_outliers.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1822 entries, 0 to 1821
Data columns (total 28 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   fecha        1822 non-null   object  
 1   dia_sem      1822 non-null   int64  
 2   Pre_elec     1822 non-null   float64 
 3   Dem          1822 non-null   float64 
 4   Prec_petr    1822 non-null   float64 
 5   Prec_gas     1822 non-null   float64 
 6   Prec_carb    1822 non-null   float64 
 7   Prod_eol     1822 non-null   float64 
 8   Prod_sol     1822 non-null   float64 
 9   Prod_hidr    1822 non-null   float64 
 10  Prod_ofr     1822 non-null   float64 
 11  Prod_nucl    1822 non-null   float64 
 12  Prod_pet     1822 non-null   float64 
 13  Prod_gas     1822 non-null   float64 
 14  Prod_carb    1822 non-null   float64 
 15  Prod_comb    1822 non-null   float64 
 16  Prod_cog     1822 non-null   float64 
 17  Prod_no_ren  1822 non-null   float64 
 18  Temp_gen     1822 non-null   float64 
 19  Vel_media_Val 1822 non-null   float64 
 20  Vel_media_Alb 1822 non-null   float64 
 21  Vel_media_Zar 1822 non-null   float64 
 22  Vel_media_Cor 1822 non-null   float64 
 23  Vel_media_Hue 1822 non-null   float64 
 24  Res_hidr     1822 non-null   float64 
 25  Der_CO2       1822 non-null   float64 
 26  Ibex          1822 non-null   float64 
 27  Int_ee        1822 non-null   float64 
dtypes: float64(26), int64(1), object(1)
memory usage: 398.7+ KB

```

## Procesamos los datos

```

In [5]: # Definimos la variable de salida y las de entrada
X = df_def_outliers[['dia_sem', 'Dem', 'Prec_petr', 'Prec_gas', 'Prec_carb',
                     'Prod_sol', 'Prod_hidr', 'Prod_ofr', 'Prod_nucl',
                     'Prod_gas', 'Prod_carb', 'Prod_comb', 'Prod_cog',
                     'Temp_gen', 'Vel_media_Val', 'Vel_media_Alb', 'Vel_media_Zar',
                     'Vel_media_Cor', 'Vel_media_Hue', 'Res_hidr', 'Der_CO2',
                     'Int_ee']].values
y = df_def_outliers['Pre_elec'].values

In [6]: # Este modelo necesita ESTANDARIZAR los datos de las variables de entrada
X = stats.zscore(X, axis=0)

In [7]: # Dividimos los datos en "train" (80%) y "test" (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

```

## Desarrollo del modelo

```

In [8]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

```

```
# Creamos el modelo
modelo_KNN = KNeighborsRegressor()

# Obtenemos el mejor número de vecinos con Validacion Cruzada
parametros = {'n_neighbors': range(1, 20)}
modelo_KNN_optimo = GridSearchCV (modelo_KNN, parametros, cv=3)

# Entrenamos el modelo
modelo_KNN_optimo.fit (X_train, y_train)

# Obtenemos el mejor estimador
print('El mejor estimador es: {}'.format(modelo_KNN_optimo.best_estimator_))

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo KNN es: ",
round(end - start,4), "segundos")
```

El mejor estimador es: KNeighborsRegressor()  
 El tiempo empleado para desarrollar el modelo KNN es: 1.3129 segundos

## Comprobacion del modelo

```
In [9]: # Obtenemos Las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_KNN_optimo.predict(X_train), y_train), 2))
print('MSE en train:', round(mean_squared_error(modelo_KNN_optimo.predict(X_train), y_train), 2))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_KNN_optimo.predict(X_train), y_train)), 2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_KNN_optimo.predict(X_test), y_test), 2))
print('MSE en test:', round(mean_squared_error(modelo_KNN_optimo.predict(X_test), y_test), 2))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_KNN_optimo.predict(X_test), y_test)), 2))

MAE en train: 2.43
MSE en train: 13.41
RMSE en train: 3.66
-----
MAE en test: 3.18
MSE en test: 22.57
RMSE en test: 4.75
```

```
In [10]: # Comprobamos La precision/rendimiento del modelo generado.
print ("La precisión del modelo K-VECINOS MAS CERCANOS con los datos de entrenamiento es de: {} %".format(round((modelo_KNN_optimo.score (X_train, y_train))*100, 2)))
print ("La precisión del modelo K-VECINOS MAS CERCANOS con los datos de test es de: {} %".format(round((modelo_KNN_optimo.score (X_test, y_test))*100, 2)))
```

La precisión del modelo K-VECINOS MAS CERCANOS con los datos de entrenamiento es de: 86.97 %  
 La precisión del modelo K-VECINOS MAS CERCANOS con los datos de test es de: 77.82 %

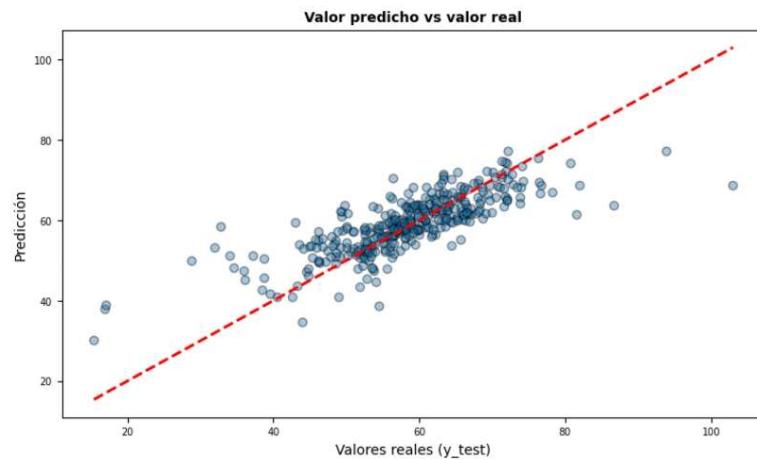
```
In [11]: # Graficamos la relacion entre los valores reales de "test" y Los predicion
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = cross_val_predict(
    estimator = modelo_KNN_optimo,
    X        = X_test,
    y        = y_test,
    cv      = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_test_pred, edgecolors = (0, 0, 0), alpha = 0.4)
```

```

axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

```



```

In [12]: MAPE_1 = (y_test - y_test_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

```

MAPE en test: 8.68 %

### Mismo modelo con diferente división (0.3 / 0.7)

```

In [13]: # Dividimos los datos en "train" y "test"
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split (X, y, test_

```

```

In [14]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_KNN_2 = KNeighborsRegressor()

# Obtenemos el mejor número de vecinos con Validación Cruzada
parametros = {'n_neighbors': range(1, 20)}
modelo_KNN_2_optimo = GridSearchCV (modelo_KNN_2, parametros, cv=3)

# Entrenamos el modelo
modelo_KNN_2_optimo.fit (X_train_2, y_train_2)

# Obtenemos el mejor estimador
print('El mejor estimador es: {}'.format(modelo_KNN_2_optimo.best_estimator_))

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo KNN es:", round(end - start, 2))

```

El mejor estimador es: KNeighborsRegressor()  
 El tiempo empleado para desarrollar el modelo KNN es: 1.2868 segundos

```
In [15]: # Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_KNN_2_optimo.predict(X_train_2), y_train_2)), 2)
print('MSE en train:', round(mean_squared_error(modelo_KNN_2_optimo.predict(X_train_2), y_train_2)), 2)
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_KNN_2_optimo.predict(X_train_2), y_train_2))), 2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_KNN_2_optimo.predict(X_test_2), y_test_2)), 2)
print('MSE en test:', round(mean_squared_error(modelo_KNN_2_optimo.predict(X_test_2), y_test_2)), 2)
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_KNN_2_optimo.predict(X_test_2), y_test_2))), 2))

MAE en train: 2.49
MSE en train: 14.6
RMSE en train: 3.82
-----
MAE en test: 3.1
MSE en test: 21.78
RMSE en test: 4.67
```

```
In [16]: # Comprobamos la precision/rendimiento del modelo generado.
print ("La precision del modelo de REGRESION LINEAL MULTIPLE con los datos de entrenamiento es de: ", round((modelo_KNN_2_optimo.score (X_train_2, y_train_2))*100, 2),
      " %")
print ("La precision del modelo de REGRESION LINEAL MULTIPLE con los datos de test es de: ", round((modelo_KNN_2_optimo.score (X_test_2, y_test_2))*100, 2), " %")
```

La precisión del modelo de REGRESION LINEAL MULTIPLE con los datos de entrenamiento es de: 85.66 %  
 La precisión del modelo de REGRESION LINEAL MULTIPLE con los datos de test es de: 79.24 %

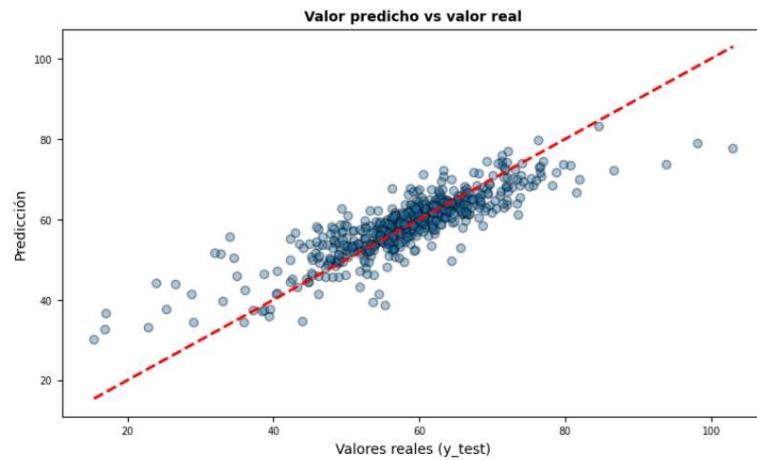
## Distribución del error de validación cruzada

```
In [17]: # Graficamos la relacion entre los valores reales de "test" y los predichos
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_2_pred = cross_val_predict(
    estimator = modelo_KNN_2_optimo,
    X         = X_test_2,
    y         = y_test_2,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test_2, y_test_2_pred, edgecolors = (0, 0, 0), alpha = 0.8)
axes.plot([y_test_2.min(), y_test_2.max()], [y_test_2.min(), y_test_2.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



```
In [18]: MAPE_1 = (y_test_2 - y_test_2_pred)/y_test_2
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 7.43 %

In [ ]:

In [ ]:

## ANEXO XII: ARBOL DE DECISION

---

### MODELO ARBOL DE DECISION

En este apartado estudiaremos el modelo de arbol de decision empleando el dataset creado en R con las variables seleccionadas."

#### Cargamos las librerias necesarias

```
In [1]: import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import r2_score

from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
import statsmodels.api as sm

from scipy import stats
import time

import matplotlib.pyplot as plt
import seaborn as sns
```

#### Cargamos el dataset con los datos

Como este modelo es muy sensible a los outliers, se carga el archivo "df\_def\_outliers", cuyos datos han sido tratados para evitar la influencia negativa de estos valores

```
In [2]: df_def = pd.read_csv(r'C:\Users\User\1.PYTHON\00.TFM\00.Archivos_utilizad
```

```
In [3]: # Comprobamos los datos cargados
print('Las dimensiones de los datos "df_def" son ' + str(df_def.shape))

Las dimensiones de los datos "df_def" son (3652, 28)
```

```
In [4]: # Comprobamos que se ha cargado bien el dataset y sus variables
df_def.info()
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD EN EL MERCADO ESPAÑOL

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3652 entries, 0 to 3651
Data columns (total 28 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   fecha             3652 non-null    object  
 1   Pre_elec          3652 non-null    float64 
 2   dia_sem           3652 non-null    int64  
 3   Dem               3652 non-null    float64 
 4   Prec_petr         3652 non-null    float64 
 5   Prec_gas          3652 non-null    float64 
 6   Prec_carb         3652 non-null    float64 
 7   Prod_eol          3652 non-null    float64 
 8   Prod_sol          3652 non-null    float64 
 9   Prod_hidr         3652 non-null    float64 
 10  Prod_ofr          3652 non-null    float64 
 11  Prod_nucl         3652 non-null    float64 
 12  Prod_pet          3652 non-null    float64 
 13  Prod_gas          3652 non-null    float64 
 14  Prod_carb         3652 non-null    float64 
 15  Prod_comb         3652 non-null    float64 
 16  Prod_cog          3652 non-null    float64 
 17  Prod_no_ren       3652 non-null    float64 
 18  Temp_gen          3652 non-null    float64 
 19  Vel_media_Val     3652 non-null    float64 
 20  Vel_media_Alb     3652 non-null    float64 
 21  Vel_media_Zar     3652 non-null    float64 
 22  Vel_media_Cor     3652 non-null    float64 
 23  Vel_media_Hue     3652 non-null    float64 
 24  Res_hidr          3652 non-null    float64 
 25  Der_CO2            3652 non-null    float64 
 26  Ibex              3652 non-null    float64 
 27  Int_ee             3652 non-null    float64 
dtypes: float64(26), int64(1), object(1)
memory usage: 799.0+ KB
```

## Procesamos los datos

```
In [5]: # Definimos la variable de salida y las de entrada
X = df_def[['dia_sem', 'Dem', 'Prec_petr', 'Prec_gas', 'Prec_carb', 'Prod_hidr', 'Prod_ofr', 'Prod_nucl', 'Prod_pet', 'Prod_gas', 'Prod_comb', 'Prod_cog', 'Prod_no_ren', 'Temp_gen', 'Vel_media_Zar', 'Vel_media_Cor', 'Vel_media_Hue', 'Res_hidr', 'Int_ee']].values
y = df_def[['Pre_elec']].values

In [6]: # Dividimos los datos en "train" (80%) y "test" (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

## Desarrollo del modelo

```
In [7]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_AD = DecisionTreeRegressor()

# Obtenemos el mejor parámetro de profundidad con Validación Cruzada
parametros = {'max_depth': range(3, 7)}
```

```

modelo_AD_optimo = GridSearchCV (modelo_AD, parametros, cv=3)

# Entrenamos el modelo
modelo_AD_optimo.fit (X_train, y_train)

# Obtenemos el mejor estimador
print('El mejor estimador es: {}'.format(modelo_AD_optimo.best_estimator_))

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo ARBOL DE DECISION es: {} segundos".format(round(end - start,4), "segundos"))

El mejor estimador es: DecisionTreeRegressor(max_depth=3)
El tiempo empleado para desarrollar el modelo ARBOL DE DECISION es: 0.52
22 segundos

```

### Comprobacion del modelo

```

In [8]: # Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE in train:', round(mean_absolute_error(modelo_AD_optimo.predict(X_train), y_train), 2))
print('MSE in train:', round(mean_squared_error(modelo_AD_optimo.predict(X_train), y_train), 2))
print('RMSE in train:', round(np.sqrt(mean_squared_error(modelo_AD_optimo.predict(X_train), y_train)), 2))
print("-----")
print('MAE in test:', round(mean_absolute_error(modelo_AD_optimo.predict(X_test), y_test), 2))
print('MSE in test:', round(mean_squared_error(modelo_AD_optimo.predict(X_test), y_test), 2))
print('RMSE in test:', round(np.sqrt(mean_squared_error(modelo_AD_optimo.predict(X_test), y_test)), 2))

MAE in train: 13.9
MSE in train: 794.2
RMSE in train: 28.19
-----
MAE in test: 14.36
MSE in test: 940.2
RMSE in test: 30.66

In [9]: # Comprobamos la precision/rendimiento del modelo generado.
print ("La precision del modelo ARBOL DE DECISION con los datos de entrenamiento es de: {} %".format(round((modelo_AD_optimo.score (X_train, y_train))*100, 2)))
print ("La precision del modelo ARBOL DE DECISION con los datos de test es de: {} %".format(round((modelo_AD_optimo.score (X_test, y_test))*100, 2)))

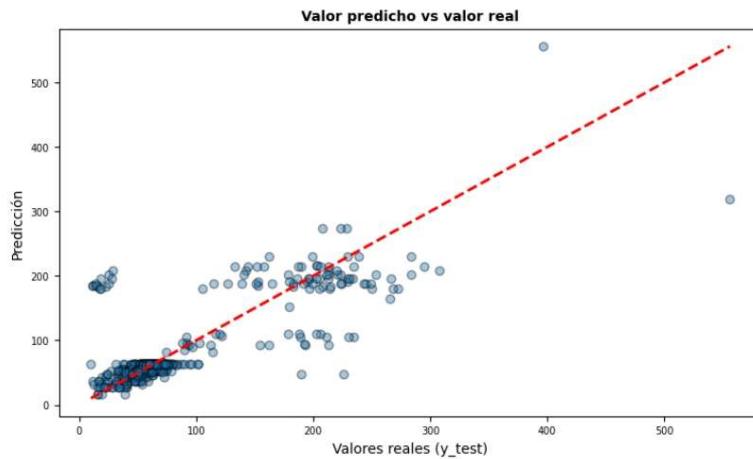
La precision del modelo ARBOL DE DECISION con los datos de entrenamiento es de: 71.32 %
La precision del modelo ARBOL DE DECISION con los datos de test es de: 70.04 %

In [10]: # Graficamos la relacion entre los valores reales de "test" y los predichos
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = cross_val_predict(
    estimator = modelo_AD_optimo,
    X         = X_test,
    y         = y_test,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_test_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')

```

```
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```



```
In [11]: MAPE_1 = (y_test - y_test_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 33.24 %

### Mismo modelo con diferente división (0.3 / 0.7)

```
In [12]: # Dividimos los datos en "train" (70%) y "test" (30%)
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split (X, y, test_
```

```
In [13]: # Calcularemos el tiempo que tarda la computadora para desarrollar el mo
start = time.time()

# Creamos el modelo
modelo_AD_2 = DecisionTreeRegressor()

# Obtenemos el mejor parametro de profundidad con Validacion Cruzada
parametros = {'max_depth': range(3, 7)}
modelo_AD_2_optimo = GridSearchCV (modelo_AD_2, parametros, cv=3)

# Entrenamos el modelo
modelo_AD_2_optimo.fit (X_train_2, y_train_2)

# Obtenemos el mejor estimador
print('El mejor estimador es: {}'.format(modelo_AD_2_optimo.best_estimad

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo ARBOL DE DECISION e
round(end - start,4), "segundos")
```

El mejor estimador es: DecisionTreeRegressor(max\_depth=4)  
 El tiempo empleado para desarrollar el modelo ARBOL DE DECISION es: 0.34  
 99 segundos

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
In [14]: # Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE in train:', round(mean_absolute_error(modelo_AD_2_optimo.predict(X_train_2), y_train_2)), 2)
print('MSE in train:', round(mean_squared_error(modelo_AD_2_optimo.predict(X_train_2), y_train_2)), 2)
print('RMSE in train:', round(np.sqrt(mean_squared_error(modelo_AD_2_optimo.predict(X_train_2), y_train_2))), 2)
print("-----")
print('MAE in test:', round(mean_absolute_error(modelo_AD_2_optimo.predict(X_test_2), y_test_2)), 2)
print('MSE in test:', round(mean_squared_error(modelo_AD_2_optimo.predict(X_test_2), y_test_2)), 2)
print('RMSE in test:', round(np.sqrt(mean_squared_error(modelo_AD_2_optimo.predict(X_test_2), y_test_2))), 2)

MAE in train: 11.98
MSE in train: 618.2
RMSE in train: 24.86
-----
MAE in test: 11.98
MSE in test: 618.2
RMSE in test: 24.86

In [15]: # Comprobamos la precisión/rendimiento del modelo generado.
print ("La precisión del modelo ARBOL DE DECISION con los datos de entrenamiento es de: ", round((modelo_AD_2_optimo.score (X_train_2, y_train_2))*100, 2))
print ("La precisión del modelo ARBOL DE DECISION con los datos de test es de: ", round((modelo_AD_2_optimo.score (X_test_2, y_test_2))*100, 2), "%")

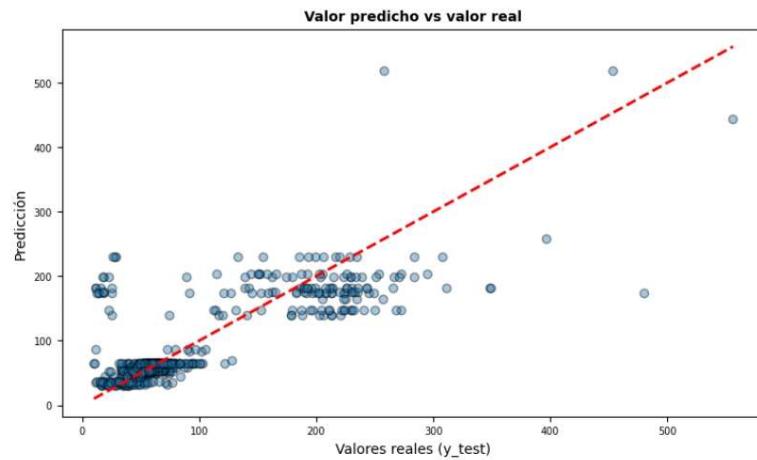
La precisión del modelo ARBOL DE DECISION con los datos de entrenamiento es de: 76.29 %
La precisión del modelo ARBOL DE DECISION con los datos de test es de: 6.35 %

In [16]: # Graficamos la relación entre los valores reales de "test" y las predicciones
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_2_pred = cross_val_predict(
    estimator = modelo_AD_2_optimo,
    X         = X_test_2,
    y         = y_test_2,
    cv       = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test_2, y_test_2_pred, edgecolors = (0, 0, 0), alpha = 0.8)
axes.plot([y_test_2.min(), y_test_2.max()],
          [y_test_2.min(), y_test_2.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



```
In [17]: MAPE_1 = (y_test_2 - y_test_2_pred)/y_test_2
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 33.34 %

In [ ]:

In [ ]:

## ANEXO XIII: RANDOM FOREST

---

### MODELO RANDOM FOREST

En este apartado estudiaremos el modelo random forest empleando el dataset creado en R con las variables seleccionadas."

#### Cargamos las librerías necesarias

```
In [1]: import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error

from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
import statsmodels.api as sm

from scipy import stats
import time

import matplotlib.pyplot as plt
import seaborn as sns
```

#### Cargamos el dataset con los datos

Como este modelo es muy sensible a los outliers, se carga el archivo "df\_def\_outliers", cuyos datos han sido tratados para evitar la influencia negativa de estos valores

```
In [2]: df_def = pd.read_csv(r'C:\Users\User\1.PYTHON\00.TFM\00.Archivos_utilizad
```

```
In [3]: # Comprobamos Los datos cargados
print('Las dimensiones de los datos "df_def" son ' + str(df_def.shape))

Las dimensiones de los datos "df_def" son (3652, 28)
```

```
In [4]: # Comprobamos que se ha cargado bien el dataset y sus variables
df_def.info()
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD EN EL MERCADO ESPAÑOL

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3652 entries, 0 to 3651
Data columns (total 28 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   fecha             3652 non-null    object  
 1   Pre_elec          3652 non-null    float64 
 2   dia_sem           3652 non-null    int64  
 3   Dem               3652 non-null    float64 
 4   Prec_petr         3652 non-null    float64 
 5   Prec_gas          3652 non-null    float64 
 6   Prec_carb         3652 non-null    float64 
 7   Prod_eol          3652 non-null    float64 
 8   Prod_sol          3652 non-null    float64 
 9   Prod_hidr         3652 non-null    float64 
 10  Prod_ofr          3652 non-null    float64 
 11  Prod_nucl         3652 non-null    float64 
 12  Prod_pet          3652 non-null    float64 
 13  Prod_gas          3652 non-null    float64 
 14  Prod_carb         3652 non-null    float64 
 15  Prod_comb         3652 non-null    float64 
 16  Prod_cog          3652 non-null    float64 
 17  Prod_no_ren       3652 non-null    float64 
 18  Temp_gen          3652 non-null    float64 
 19  Vel_media_Val     3652 non-null    float64 
 20  Vel_media_Alb     3652 non-null    float64 
 21  Vel_media_Zar     3652 non-null    float64 
 22  Vel_media_Cor     3652 non-null    float64 
 23  Vel_media_Hue     3652 non-null    float64 
 24  Res_hidr          3652 non-null    float64 
 25  Der_CO2            3652 non-null    float64 
 26  Ibex              3652 non-null    float64 
 27  Int_ee             3652 non-null    float64 
dtypes: float64(26), int64(1), object(1)
memory usage: 799.0+ KB
```

## Procesamos los datos

```
In [5]: # Definimos la variable de salida y las de entrada
X = df_def[['dia_sem', 'Dem', 'Prec_petr', 'Prec_gas', 'Prec_carb', 'Prod_hidr', 'Prod_ofr', 'Prod_nucl', 'Prod_pet', 'Prod_gas', 'Prod_comb', 'Prod_cog', 'Prod_no_ren', 'Temp_gen', 'Vel_media_Val', 'Vel_media_Alb', 'Vel_media_Zar', 'Vel_media_Cor', 'Vel_media_Hue', 'Der_CO2', 'Ibex', 'Int_ee']].values
y = df_def[['Pre_elec']].values

In [6]: # Dividimos los datos en "train" (80%) y "test" (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

## Desarrollo del modelo

```
In [7]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RF = RandomForestRegressor()

# Obtenemos el mejor hiperparámetro de profundidad con Validación Cruzada
hiperparametro = {'max_depth': range(3, 7)}
```

```

modelo_RF_optimo = GridSearchCV (modelo_RF, hiperparametro, cv=3)

# Entrenamos el modelo
modelo_RF_optimo.fit (X_train, y_train)

# Obtenemos el mejor estimador
print('El mejor estimador es: {}'.format(modelo_RF_optimo.best_estimator_))

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo RANDOM FOREST es:",
      round(end - start,4), "segundos")

```

El mejor estimador es: RandomForestRegressor(max\_depth=6)  
 El tiempo empleado para desarrollar el modelo RANDOM FOREST es: 24.5987 segundos

### Comprobacion del modelo

```

In [8]: # Obtenemos Las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RF_optimo.predict(X_train), y_train), 2))
print('MSE en train:', round(mean_squared_error(modelo_RF_optimo.predict(X_train), y_train), 2))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RF_optimo.predict(X_train), y_train)), 2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RF_optimo.predict(X_test), y_test), 2))
print('MSE en test:', round(mean_squared_error(modelo_RF_optimo.predict(X_test), y_test), 2))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RF_optimo.predict(X_test), y_test)), 2))

```

MAE en train: 8.55  
 MSE en train: 352.87  
 RMSE en train: 18.78  
 -----  
 MAE en test: 11.34  
 MSE en test: 791.17  
 RMSE en test: 28.13

```

In [9]: # Comprobamos La precision/rendimiento del modelo generado.
print ("La precisión del modelo de RANDOM FOREST con los datos de entrenamiento es de: {} %".format(round((modelo_RF_optimo.score (X_train, y_train))*100, 2)))
print ("La precisión del modelo de RANDOM FOREST con los datos de test es de: {} %".format(round((modelo_RF_optimo.score (X_test, y_test))*100, 2)))

```

La precisión del modelo de RANDOM FOREST con los datos de entrenamiento es de: 87.26 %  
 La precisión del modelo de RANDOM FOREST con los datos de test es de: 74.79 %

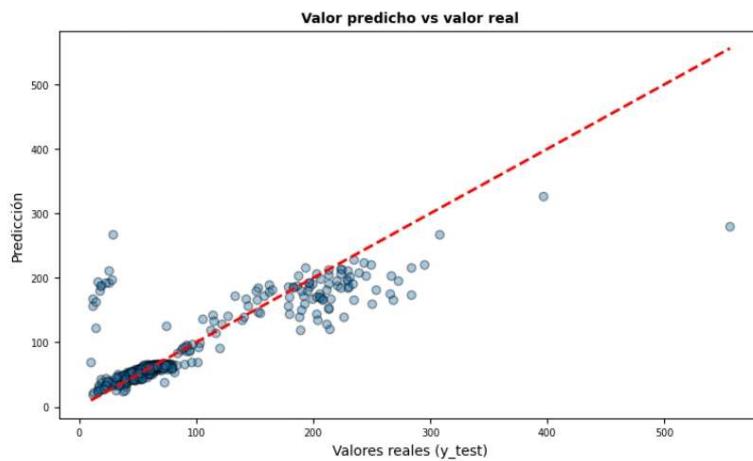
```

In [10]: # Graficamos la relacion entre los valores reales de "test" y los predichos
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = cross_val_predict(
    estimator = modelo_RF_optimo,
    X         = X_test,
    y         = y_test,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_test_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')

```

```
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```



```
In [11]: MAPE_1 = (y_test - y_test_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 28.0 %

### Optimización del modelo

```
In [12]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RF = RandomForestRegressor()

# Obtenemos el mejor parametro de profundidad con Validacion Cruzada
hiperparametros = {'n_estimators': [100, 200, 300],
                   'max_depth': [None, 3, 4, 5, 6, 7, 8, 9, 10]}
modelo_RF_optimo = GridSearchCV(modelo_RF, hiperparametros, cv=3)

# Entrenamos el modelo
modelo_RF_optimo.fit (X_train, y_train)

# Obtenemos el mejor estimador
print('El mejor estimador es: {}'.format(modelo_RF_optimo.best_params_))

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo RANDOM FOREST es:",
      round(end - start,4), "segundos")
```

El mejor estimador es: {'max\_depth': None, 'n\_estimators': 300}  
 El tiempo empleado para desarrollar el modelo RANDOM FOREST es: 488.7363 segundos

```
In [13]: # Obtenemos Las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RF_optimo.predict(X_train), y_train), 2))
print('MSE en train:', round(mean_squared_error(modelo_RF_optimo.predict(X_train), y_train), 2))
```

```

print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RF_optimo
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RF_optimo.predict(
print('MSE en test:', round(mean_squared_error(modelo_RF_optimo.predict(
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RF_optimo
MAE en train: 3.35
MSE en train: 94.96
RMSE en train: 9.74
-----
MAE en test: 9.78
MSE en test: 809.44
RMSE en test: 28.45

```

In [14]: # Comprobamos la precisión/rendimiento del modelo generado.

```

print ("La precisión del modelo de RANDOM FOREST con los datos de entrenamiento es: " + str(round((modelo_RF_optimo.score (X_train, y_train))*100, 2)) + "%")
print ("La precisión del modelo de RANDOM FOREST con los datos de test es: " + str(round((modelo_RF_optimo.score (X_test, y_test))*100, 2)) + "%")

```

La precisión del modelo de RANDOM FOREST con los datos de entrenamiento es de: 96.57 %  
 La precisión del modelo de RANDOM FOREST con los datos de test es de: 7 4.21 %

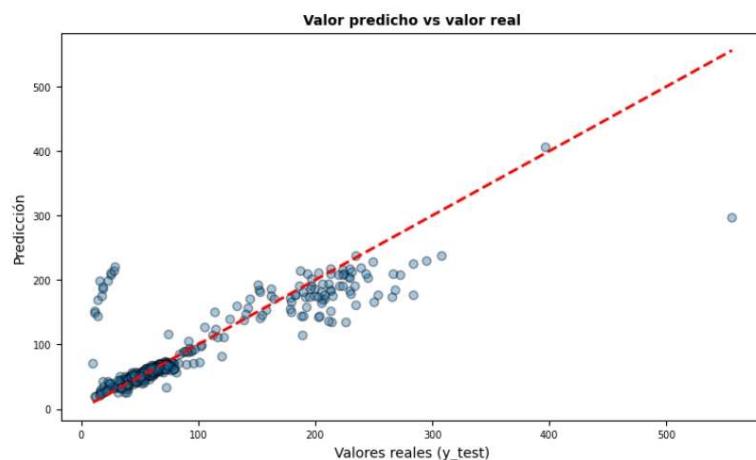
In [15]: # Graficamos la relación entre los valores reales de "test" y las predicciones.

```

cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = cross_val_predict(
    estimator = modelo_RF_optimo,
    X         = X_test,
    y         = y_test,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_test_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

```



```
In [16]: MAPE_1 = (y_test - y_test_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
MAPE en test: 26.2 %
```

**Mismo modelo con diferente división (0.3 / 0.7)**

```
In [17]: # Dividimos los datos en "train" (70%) y "test" (30%)
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split (X, y, test_
In [18]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RF_2 = RandomForestRegressor()

# Obtenemos el mejor hiperparámetro de profundidad con Validación Cruzada
hiperparametro = {'max_depth': range(3, 7)}
modelo_RF_2_optimo = GridSearchCV (modelo_RF_2, hiperparametro, cv=3)

# Entrenamos el modelo
modelo_RF_2_optimo.fit (X_train_2, y_train_2)

# Obtenemos el mejor estimador
print('El mejor estimador es: {}'.format(modelo_RF_2_optimo.best_estimator_))

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo RANDOM FOREST es:", round(end - start,4), "segundos")
```

El mejor estimador es: RandomForestRegressor(max\_depth=6)  
 El tiempo empleado para desarrollar el modelo RANDOM FOREST es: 23.6991 segundos

```
In [19]: # Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RF_2_optimo.predict(X_train_2), y_train_2), 2))
print('MSE en train:', round(mean_squared_error(modelo_RF_2_optimo.predict(X_train_2), y_train_2), 2))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RF_2_optimo.predict(X_train_2), y_train_2)), 2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RF_2_optimo.predict(X_test_2), y_test_2), 2))
print('MSE en test:', round(mean_squared_error(modelo_RF_2_optimo.predict(X_test_2), y_test_2), 2))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RF_2_optimo.predict(X_test_2), y_test_2)), 2))
```

MAE en train: 8.5  
 MSE en train: 348.76  
 RMSE en train: 18.68  
 -----  
 MAE en test: 11.74  
 MSE en test: 873.47  
 RMSE en test: 29.55

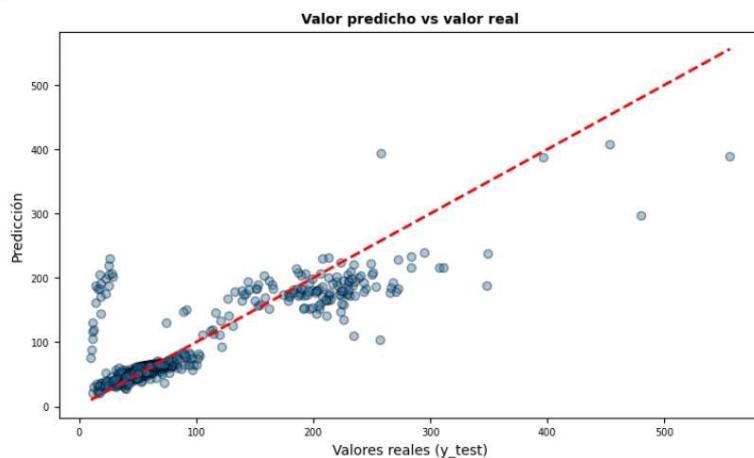
```
In [20]: # Comprobamos la precisión/rendimiento del modelo generado.
print ("La precisión del modelo de RANDOM FOREST con los datos de entrenamiento es: ", round((modelo_RF_2_optimo.score (X_train_2, y_train_2))*100, 2))
print ("La precisión del modelo de RANDOM FOREST con los datos de test es: ", round((modelo_RF_2_optimo.score (X_test_2, y_test_2))*100, 2), "%")
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
La precisión del modelo de RANDOM FOREST con los datos de entrenamiento  
es de: 86.62 %  
La precisión del modelo de RANDOM FOREST con los datos de test es de: 7  
4.27 %
```

```
In [21]: # Graficamos la relación entre los valores reales de "test" y los predi  
cv = KFold(n_splits=5, random_state=123, shuffle=True)  
y_test_2_pred = cross_val_predict(  
    estimator = modelo_RF_2_optimo,  
    X         = X_test_2,  
    y         = y_test_2,  
    cv        = cv)  
  
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))  
axes.scatter(y_test_2, y_test_2_pred, edgecolors = (0, 0, 0), alpha = 0.  
axes.plot([y_test_2.min(), y_test_2.max()],  
         [y_test_2.min(), y_test_2.max()],  
         '--', lw=2, color = 'red')  
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight  
axes.set_xlabel('Valores reales (y_test)')  
axes.set_ylabel('Predicción')  
axes.tick_params(labelsize = 7)
```



```
In [22]: MAPE_1 = (y_test_2 - y_test_2_pred)/y_test_2  
MAPE_1 = np.sqrt(MAPE_1**2)  
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

```
MAPE en test: 28.33 %
```

```
In [23]: # Optimizamos el modelo  
  
# Calcularemos el tiempo que tarda la computadora para desarrollar el mo  
start = time.time()  
  
# Creamos el modelo  
modelo_RF_2 = RandomForestRegressor()  
  
# Obtenemos el mejor parámetro de profundidad con Validación Cruzada  
hiperparametros = {'n_estimators': [100, 200, 300],  
                  'max_depth': [None, 3, 4, 5, 6, 7, 8, 9, 10]}  
modelo_RF_2_optimo = GridSearchCV (modelo_RF_2, hiperparametros, cv=3)
```

```
# Entrenamos el modelo
modelo_RF_2_optimo.fit (X_train_2, y_train_2)

# Obtenemos el mejor estimador
print('El mejor estimador es: {}'.format(modelo_RF_2_optimo.best_params_)

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo RANDOM FOREST es:",
      round(end - start,4), "segundos")
```

El mejor estimador es: {'max\_depth': None, 'n\_estimators': 300}  
 El tiempo empleado para desarrollar el modelo RANDOM FOREST es: 418.6353 segundos

```
In [24]: # Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RF_2_optimo.predict(X_train_2), y_train_2), 2),
     print('MSE en train:', round(mean_squared_error(modelo_RF_2_optimo.predict(X_train_2), y_train_2), 2),
     print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RF_2_optimo.predict(X_train_2), y_train_2)), 2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RF_2_optimo.predict(X_test_2), y_test_2), 2),
     print('MSE en test:', round(mean_squared_error(modelo_RF_2_optimo.predict(X_test_2), y_test_2), 2),
     print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RF_2_optimo.predict(X_test_2), y_test_2)), 2))
```

MAE en train: 3.36  
 MSE en train: 92.2  
 RMSE en train: 9.6  
 -----  
 MAE en test: 10.0  
 MSE en test: 885.27  
 RMSE en test: 29.75

```
In [25]: # Comprobamos la precisión/rendimiento del modelo generado.
print ("La precisión del modelo de RANDOM FOREST con los datos de entrenamiento es de: {} %".format(round((modelo_RF_2_optimo.score (X_train_2, y_train_2))*100, 2)),
print ("La precisión del modelo de RANDOM FOREST con los datos de test es de: {} %".format(round((modelo_RF_2_optimo.score (X_test_2, y_test_2))*100, 2)), "%")
```

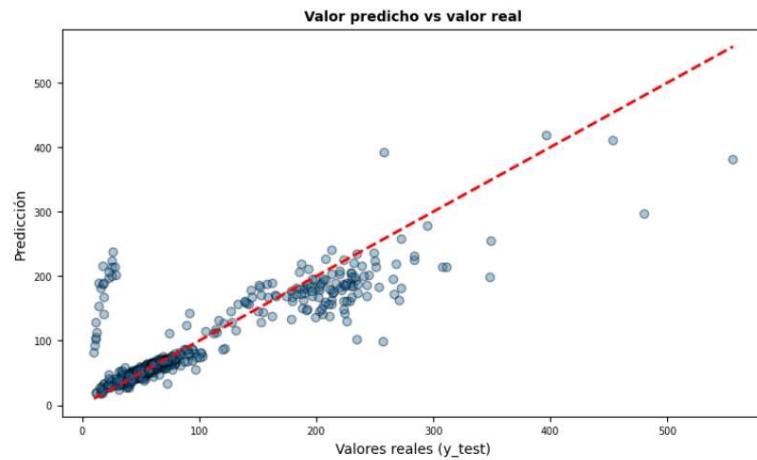
La precisión del modelo de RANDOM FOREST con los datos de entrenamiento es de: 96.46 %  
 La precisión del modelo de RANDOM FOREST con los datos de test es de: 7.3.92 %

```
In [26]: # Graficamos la relación entre los valores reales de "test" y los predichos
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_2_pred = cross_val_predict(
    estimator = modelo_RF_2_optimo,
    X         = X_test_2,
    y         = y_test_2,
    cv       = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test_2, y_test_2_pred, edgecolors = (0, 0, 0), alpha = 0.8)
axes.plot([y_test_2.min(), y_test_2.max()],
          [y_test_2.min(), y_test_2.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



```
In [27]: MAPE_1 = (y_test_2 - y_test_2_pred)/y_test_2
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 25.92 %

In [ ]:

In [ ]:

## ANEXO XIV: XG BOOST

---

### MODELO XG BOOST

En este apartado estudiaremos el modelo XG Boost empleando el dataset creado en R con las variables seleccionadas."

#### Cargamos las librerías necesarias

```
In [1]: import pandas as pd
import numpy as np

import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error

from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
import statsmodels.api as sm

from scipy import stats
import time

import matplotlib.pyplot as plt
import seaborn as sns
```

#### Cargamos el dataset con los datos

Como este modelo es muy sensible a los outliers, se carga el archivo "df\_def\_outliers", cuyos datos han sido tratados para evitar la influencia negativa de estos valores

```
In [2]: df_def = pd.read_csv(r'C:\Users\User\1.PYTHON\00.TFM\00.Archivos_utilizad
```

```
In [3]: # Comprobamos Los datos cargados
print('Las dimensiones de los datos "df_def" son ' + str(df_def.shape))

Las dimensiones de los datos "df_def" son (3652, 28)
```

```
In [4]: # Comprobamos que se ha cargado bien el dataset y sus variables
df_def.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3652 entries, 0 to 3651
Data columns (total 28 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   fecha        3652 non-null   object  
 1   Pre_elec     3652 non-null   float64 
 2   dia_sem      3652 non-null   int64  
 3   Dem          3652 non-null   float64 
 4   Prec_petr    3652 non-null   float64 
 5   Prec_gas     3652 non-null   float64 
 6   Prec_carb    3652 non-null   float64 
 7   Prod_eol     3652 non-null   float64 
 8   Prod_sol     3652 non-null   float64 
 9   Prod_hidr    3652 non-null   float64 
 10  Prod_ofr     3652 non-null   float64 
 11  Prod_nucl    3652 non-null   float64 
 12  Prod_pet     3652 non-null   float64 
 13  Prod_gas     3652 non-null   float64 
 14  Prod_carb    3652 non-null   float64 
 15  Prod_comb    3652 non-null   float64 
 16  Prod_cog     3652 non-null   float64 
 17  Prod_no_ren  3652 non-null   float64 
 18  Temp_gen     3652 non-null   float64 
 19  Vel_media_Val 3652 non-null   float64 
 20  Vel_media_Alb 3652 non-null   float64 
 21  Vel_media_Zar 3652 non-null   float64 
 22  Vel_media_Cor 3652 non-null   float64 
 23  Vel_media_Hue 3652 non-null   float64 
 24  Res_hidr     3652 non-null   float64 
 25  Der_CO2       3652 non-null   float64 
 26  Ibex          3652 non-null   float64 
 27  Int_ee        3652 non-null   float64 
dtypes: float64(26), int64(1), object(1)
memory usage: 799.0+ KB
```

## Procesamos los datos

```
In [5]: # Definimos la variable de salida y las de entrada
X = df_def[['dia_sem', 'Dem', 'Prec_petr', 'Prec_gas', 'Prec_carb', 'Pr
    'Prod_sol', 'Prod_hidr', 'Prod_ofr', 'Prod_nucl', 'Prod_pet',
    'Prod_carb', 'Prod_comb', 'Prod_cog', 'Prod_no_ren', 'Temp_
    'Vel_media_Alb', 'Vel_media_Zar', 'Vel_media_Cor', 'Vel_med
    'Der_CO2', 'Ibex', 'Int_ee']].values
y = df_def[['Pre_elec']].values

In [6]: # Dividimos los datos en "train" (80%) y "test" (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

## Desarrollo del modelo

```
In [7]: # Calcularemos el tiempo que tarda la computadora para desarrollar el mo
start = time.time()

# Creamos el modelo
modelo_XGB = xgb.XGBRegressor()

# Obtenemos los mejores hiperparámetros con Validación Cruzada
parametros = {'eta':[0.3, 0.5], 'lambda':[0.5, 1.0], 'max_depth':range(3, 10)}
```

```

modelo_XGB_optimo = GridSearchCV (modelo_XGB, parametros, cv=3)

# Entrenamos el modelo
modelo_XGB_optimo.fit (X_train, y_train)

# Obtenemos Los mejores hiperparametros para el modelo
print('Los mejores hiperparametros son: {}'.format(modelo_XGB_optimo.best_params_))

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo XG BOOST es:",
      round(end - start,2), "segundos")

Los mejores hiperparametros son: {'eta': 0.3, 'lambda': 1.0, 'max_depth': 3}
El tiempo empleado para desarrollar el modelo XG BOOST es: 104.65 segundos

```

### Comprobacion del modelo

```

In [8]: # Obtenemos Las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_XGB_optimo.predict(X_train), y_train), 2))
print('MSE en train:', round(mean_squared_error(modelo_XGB_optimo.predict(X_train), y_train), 2))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_XGB_optimo.predict(X_train), y_train)), 2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_XGB_optimo.predict(X_test), y_test), 2))
print('MSE en test:', round(mean_squared_error(modelo_XGB_optimo.predict(X_test), y_test), 2))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_XGB_optimo.predict(X_test), y_test)), 2))

MAE en train: 5.22
MSE en train: 95.84
RMSE en train: 9.79
-----
MAE en test: 11.34
MSE en test: 897.33
RMSE en test: 29.96

In [9]: # Comprobamos La precision/rendimiento del modelo generado.
print ("La precisión del modelo de XG BOOST con los datos de entrenamiento es de: {} %".format(round((modelo_XGB_optimo.score (X_train, y_train))*100, 2)))
print ("La precisión del modelo de XG BOOST con los datos de test es de: {} %".format(round((modelo_XGB_optimo.score (X_test, y_test))*100, 2)))

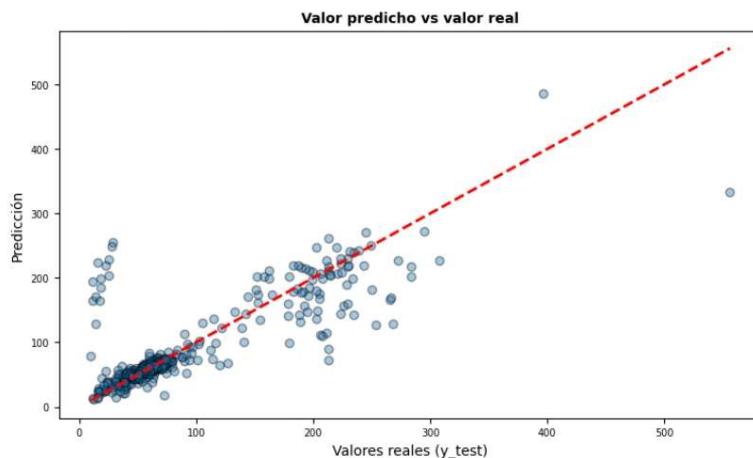
La precisión del modelo de XG BOOST con los datos de entrenamiento es de: 96.54 %
La precisión del modelo de XG BOOST con los datos de test es de: 71.41 %

In [10]: # Graficamos La relacion entre Los valores reales de "test" y Los predichos
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = cross_val_predict(
    estimator = modelo_XGB_optimo,
    X         = X_test,
    y         = y_test,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_test_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')

```

```
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```



```
In [11]: MAPE_1 = (y_test - y_test_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 28.88 %

### Optimización del modelo

```
In [12]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_XGB = xgb.XGBRegressor()

parametros = {'eta':[0.01, 2], 'lambda':[0.01, 1000], 'max_depth':range(1,10)}
modelo_XGB_optimo = GridSearchCV (modelo_XGB, parametros, cv=3)

# Entrenamos el modelo
modelo_XGB_optimo.fit (X_train, y_train)

# Obtenemos Los mejores hiperparametros para el modelo
print('Los mejores hiperparametros son: {}'.format(modelo_XGB_optimo.best_params_))

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo XG BOOST es:", round(end-start, 2))

Los mejores hiperparametros son: {'eta': 2, 'lambda': 1000, 'max_depth': 6}
El tiempo empleado para desarrollar el modelo XG BOOST es: 137.58
```

```
In [13]: # Obtenemos Las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_XGB_optimo.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_XGB_optimo.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_XGB_optimo.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_XGB_optimo.predict(X_test), y_test)))
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
print('MSE en test:', round(mean_squared_error(modelo_XGB_optimo.predict
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_XGB_optimo.
```

```
MAE en train: 3.2
MSE en train: 46.19
RMSE en train: 6.8
-----
MAE en test: 12.42
MSE en test: 900.93
RMSE en test: 30.02
```

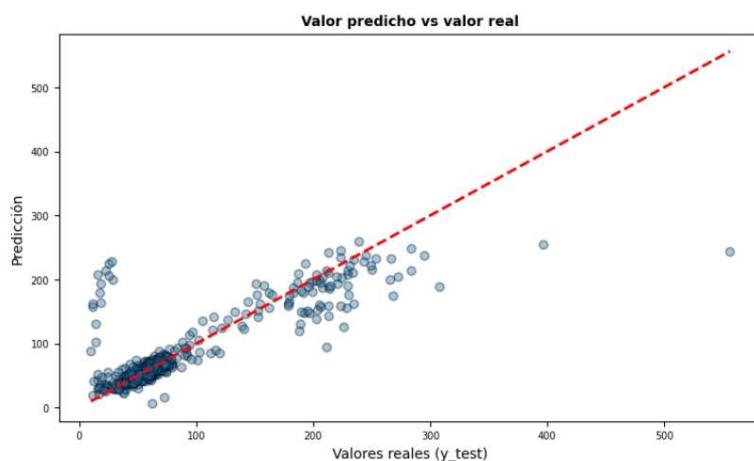
```
In [14]: # Comprobamos la precisión/rendimiento del modelo generado.
print ("La precisión del modelo de XG BOOST con los datos de entrenamiento es de: " + str(round((modelo_XGB_optimo.score (X_train, y_train))*100, 2)) + "%")
print ("La precisión del modelo de XG BOOST con los datos de test es de: " + str(round((modelo_XGB_optimo.score (X_test, y_test))*100, 2)) + "%")
```

La precisión del modelo de XG BOOST con los datos de entrenamiento es de: 98.33 %

La precisión del modelo de XG BOOST con los datos de test es de: 71.29 %

```
In [15]: # Graficamos la relación entre los valores reales de "test" y los predichos
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = cross_val_predict(
    estimator = modelo_XGB_optimo,
    X         = X_test,
    y         = y_test,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_test_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```



```
In [16]: MAPE_1 = (y_test - y_test_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

```
MAPE en test: 29.62 %
```

### Mismo modelo con diferente division (0.3 / 0.7)

```
In [17]: # Dividimos los datos en "train" (70%) y "test" (30%)
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split (X, y, test_
                                                              
In [18]: # Calcularemos el tiempo que tarda la computadora para desarrollar el mo
start = time.time()

# Creamos el modelo
modelo_XGB_2 = xgb.XGBRegressor()

# Obtenemos los mejores hiperparametros con Validacion Cruzada
parametros = {'eta':[0.3, 0.5], 'lambda':[0.5, 1.0], 'max_depth':range(3, 10)}
modelo_XGB_2_optimo = GridSearchCV (modelo_XGB_2, parametros, cv=3)

# Entrenamos el modelo
modelo_XGB_2_optimo.fit (X_train_2, y_train_2)

# Obtenemos los mejores hiperparametros para el modelo
print('Los mejores hiperparametros son: {}'.format(modelo_XGB_2_optimo.b
                                                              
# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo XG BOOST es:",
      round(end - start,2), "segundos")

Los mejores hiperparametros son: {'eta': 0.3, 'lambda': 0.5, 'max_dept
h': 4}
El tiempo empleado para desarrollar el modelo XG BOOST es: 47.96 segundo
s

In [19]: # Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_XGB_2_optimo.pre
print('MSE en train:', round(mean_squared_error(modelo_XGB_2_optimo.pre
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_XGB_2_opt
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_XGB_2_optimo.pre
print('MSE en test:', round(mean_squared_error(modelo_XGB_2_optimo.pre
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_XGB_2_opt
                                                              
MAE en train: 2.66
MSE en train: 14.73
RMSE en train: 3.84
-----
MAE en test: 11.52
MSE en test: 1159.21
RMSE en test: 34.05

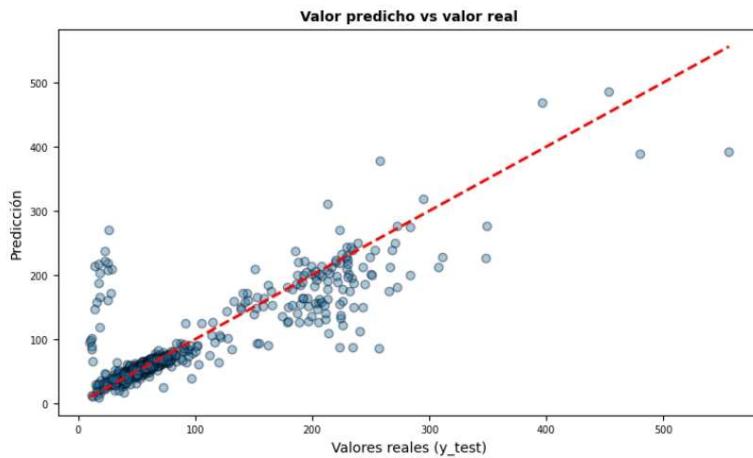
In [20]: # Comprobamos la precision/rendimiento del modelo generado.
print ("La precision del modelo de XG BOOST con los datos de entrenamiento
      round((modelo_XGB_2_optimo.score (X_train_2, y_train_2))*100, 2),
print ("La precision del modelo de XG BOOST con los datos de test es de:
      round((modelo_XGB_2_optimo.score (X_test_2, y_test_2))*100, 2), "
                                                              
La precision del modelo de XG BOOST con los datos de entrenamiento es d
e: 99.43 %
La precision del modelo de XG BOOST con los datos de test es de: 65.85 %
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
In [21]: # Graficamos la relación entre los valores reales de "test" y los predichos
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_2_pred = cross_val_predict(
    estimator = modelo_XGB_2_optimo,
    X         = X_test_2,
    y         = y_test_2,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test_2, y_test_2_pred, edgecolors = (0, 0, 0), alpha = 0.5)
axes.plot([y_test_2.min(), y_test_2.max()], [y_test_2.min(), y_test_2.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```



```
In [22]: MAPE_1 = (y_test_2 - y_test_2_pred)/y_test_2
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 25.82 %

```
In [23]: # Optimizamos el modelo

# Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_XGB_2 = xgb.XGBRegressor()

# Obtenemos los mejores hiperparámetros con Validación Cruzada
parametros = {'eta':[0.01, 2], 'lambda':[0.01, 1000], 'max_depth':range(1, 10)}
modelo_XGB_2_optimo = GridSearchCV (modelo_XGB_2, parametros, cv=3)

# Entrenamos el modelo
modelo_XGB_2_optimo.fit (X_train_2, y_train_2)

# Obtenemos los mejores hiperparámetros para el modelo
print('Los mejores hiperparámetros son: {}'.format(modelo_XGB_2_optimo.best_params_))
```

```
# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para desarrollar el modelo XG BOOST es:", round((end - start), 2))
Los mejores hiperparametros son: {'eta': 2, 'lambda': 1000, 'max_depth': 3}
El tiempo empleado para desarrollar el modelo XG BOOST es: 118.67

In [24]: # Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_XGB_2_optimo.predict(X_train_2), y_train_2)))
print('MSE en train:', round(mean_squared_error(modelo_XGB_2_optimo.predict(X_train_2), y_train_2)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_XGB_2_optimo.predict(X_train_2), y_train_2))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_XGB_2_optimo.predict(X_test_2), y_test_2)))
print('MSE en test:', round(mean_squared_error(modelo_XGB_2_optimo.predict(X_test_2), y_test_2)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_XGB_2_optimo.predict(X_test_2), y_test_2))))

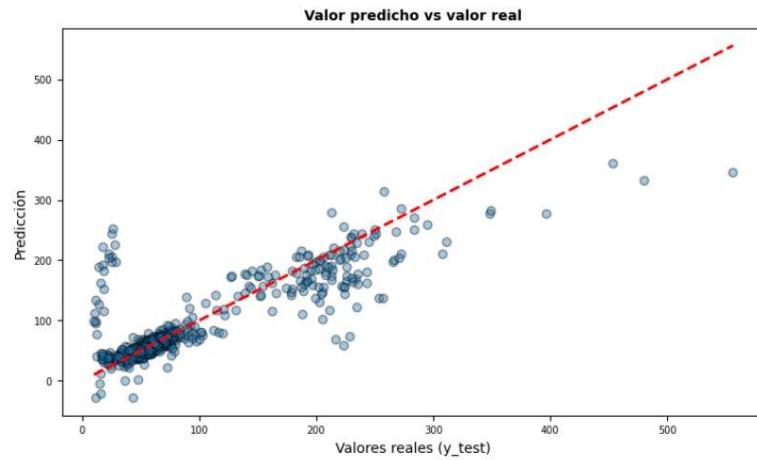
MAE en train: 7.26
MSE en train: 213.79
RMSE en train: 14.62
-----
MAE en test: 12.82
MSE en test: 924.92
RMSE en test: 30.41

In [25]: # Comprobamos la precision/rendimiento del modelo generado.
print ("La precisión del modelo de XG BOOST con los datos de entrenamiento es de: ", round((modelo_XGB_2_optimo.score (X_train_2, y_train_2))*100, 2))
print ("La precisión del modelo de XG BOOST con los datos de test es de: ", round((modelo_XGB_2_optimo.score (X_test_2, y_test_2))*100, 2))

La precisión del modelo de XG BOOST con los datos de entrenamiento es de: 91.8 %
La precisión del modelo de XG BOOST con los datos de test es de: 72.75 %

In [26]: # Graficamos la relación entre los valores reales de "test" y las predicciones
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_2_pred = cross_val_predict(
    estimator = modelo_XGB_2_optimo,
    X         = X_test_2,
    y         = y_test_2,
    cv        = cv)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test_2, y_test_2_pred, edgecolors = (0, 0, 0), alpha = 0.8)
axes.plot([y_test_2.min(), y_test_2.max()], [y_test_2.min(), y_test_2.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```



```
In [27]: MAPE_1 = (y_test_2 - y_test_2_pred)/y_test_2
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 29.76 %

In [ ]:

In [ ]:

## ANEXO XV: RED NEURONAL 80-20

---

### RED NEURONAL

En este apartado estudiaremos diferentes modelos de Redes Neuronales. Comenzaremos con la más sencilla de una sola capa oculta y pocas neuronas. Posteriormente iremos aumentando las capas y las neuronas en ellas hasta quedarnos con la que estimemos más precisa.

#### Cargamos las librerías necesarias

```
In [1]: import pandas as pd
import numpy as np

from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error, mean_absolute_error

from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
import statsmodels.api as sm

from scipy import stats
import time

import matplotlib.pyplot as plt
import seaborn as sns
```

#### Cargamos el dataset con los datos

```
In [2]: df_def = pd.read_csv(r'C:\Users\User\1.PYTHON\00.TFM\00.Archivos_utilizad
```

```
In [3]: # Comprobamos el dataframe cargado
df_def.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3652 entries, 0 to 3651
Data columns (total 28 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   fecha        3652 non-null   object  
 1   Pre_elec     3652 non-null   float64 
 2   dia_sem      3652 non-null   int64  
 3   Dem          3652 non-null   float64 
 4   Prec_petr    3652 non-null   float64 
 5   Prec_gas     3652 non-null   float64 
 6   Prec_carb    3652 non-null   float64 
 7   Prod_eol     3652 non-null   float64 
 8   Prod_sol     3652 non-null   float64 
 9   Prod_hidr    3652 non-null   float64 
 10  Prod_ofr     3652 non-null   float64 
 11  Prod_nucl    3652 non-null   float64 
 12  Prod_pet     3652 non-null   float64 
 13  Prod_gas     3652 non-null   float64 
 14  Prod_carb    3652 non-null   float64 
 15  Prod_comb    3652 non-null   float64 
 16  Prod_cog     3652 non-null   float64 
 17  Prod_no_ren  3652 non-null   float64 
 18  Temp_gen     3652 non-null   float64 
 19  Vel_media_Val 3652 non-null   float64 
 20  Vel_media_Alb 3652 non-null   float64 
 21  Vel_media_Zar 3652 non-null   float64 
 22  Vel_media_Cor 3652 non-null   float64 
 23  Vel_media_Hue 3652 non-null   float64 
 24  Res_hidr     3652 non-null   float64 
 25  Der_CO2       3652 non-null   float64 
 26  Ibex          3652 non-null   float64 
 27  Int_ee        3652 non-null   float64 
dtypes: float64(26), int64(1), object(1)
memory usage: 799.0+ KB
```

## Procesamos los datos

```
In [4]: # Definimos la variable de salida y las de entrada
X = df_def[['dia_sem', 'Dem', 'Prec_petr', 'Prec_gas', 'Prec_carb', 'Pr
    'Prod_sol', 'Prod_hidr', 'Prod_ofr', 'Prod_nucl', 'Prod_pet',
    'Prod_carb', 'Prod_comb', 'Prod_cog', 'Prod_no_ren', 'Temp_
    'Vel_media_Alb', 'Vel_media_Zar', 'Vel_media_Cor', 'Vel_med
    'Der_CO2', 'Ibex', 'Int_ee']].values
y = df_def[['Pre_elec']].values
```

```
In [5]: # Comprobamos las dos matrices de datos creadas con los datos
print(X.shape)
y = y.reshape(-1,1)
print(y.shape)

(3652, 26)
(3652, 1)
```

```
In [6]: # Realizamos la estandarización de los datos necesaria para el empleo
# de la Red Neuronal
X = stats.zscore(X, axis=0)
```

```
In [7]: # Dividimos los datos en "train_2" (70%) y "test_2" (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size = 0.2,  
random_state = 1234)
```

## Desarrollamos los modelos

### MODELO 1: 3 Capas y 5 neuronas en la capa oculta

```
In [8]: # Calcularemos el tiempo que tarda la computadora para desarrollar el mo  
start = time.time()  
  
# Creamos el modelo  
modelo_RNeu_5 = Sequential()  
  
# 1º capa oculta  
modelo_RNeu_5.add(Dense(5, input_dim=26, activation = 'relu'))  
  
# Capa de salida  
modelo_RNeu_5.add(Dense(1, activation = 'relu'))  
  
# Compilacion del modelo  
modelo_RNeu_5.compile(loss='mae', # mean_squared_error  
                      optimizer='adam',  
                      metrics=['mse', 'accuracy'])  
  
# Obtenemos el resumen de la red neuronal creada  
modelo_RNeu_5.summary()  
  
# Ajuste del modelo  
print("El modelo se está entrenando.....")  
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=50,  
                           verbose=1)  
entrenamiento = modelo_RNeu_5.fit(X_train, y_train,  
                                    validation_data = (X_test, y_test),  
                                    epochs = 2000,  
                                    batch_size = 30,  
                                    verbose = False,  
                                    callbacks=[early_stop])  
print("El entrenamiento ha finalizado!")  
  
print('-----')  
print('Datos del MODELO 1: 3 Capas y 5 neuronas en la capa oculta')  
  
# Medimos el tiempo que ha tardado el modelo  
end = time.time()  
print("El tiempo empleado para calcular el modelo ha sido: ",  
      round(end - start,4))
```

```
Model: "sequential"
-----
Layer (type)          Output Shape       Param #
dense (Dense)         (None, 5)           135
dense_1 (Dense)        (None, 1)            6
-----
Total params: 141
Trainable params: 141
Non-trainable params: 0
-----
El modelo se está entrenando.....
Epoch 989: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 1: 3 Capas y 5 neuronas en la capa oculta
El tiempo empleado para calcular el modelo ha sido: 302.278
```

```
In [9]: # Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_5.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_5.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_5.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_5.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_5.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_5.predict(X_test), y_test))))
```

MAE en train: 9.19  
MSE en train: 697.04  
RMSE en train: 26.4

-----

MAE en test: 9.62  
MSE en test: 736.51  
RMSE en test: 27.14

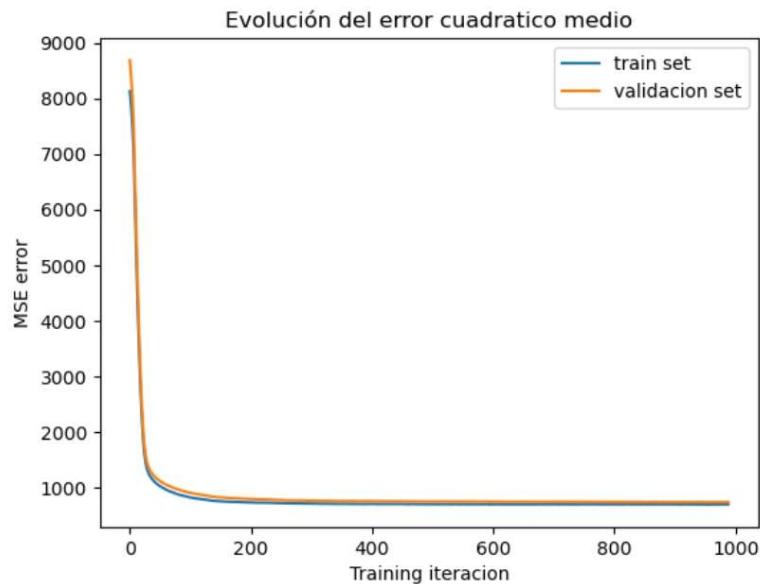
```
In [10]: # Comprobamos la precisión/rendimiento del modelo generado.
y_pred = modelo_RNeu_5.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: " + str(round(r2_score(y_train, y_pred)*100, 2)))
y_pred = modelo_RNeu_5.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de test es de: " + str(round(r2_score(y_test, y_pred)*100, 2)))
```

La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 74.84  
La precisión del modelo de RED NUERONAL con los datos de test es de: 76.53

```
In [11]: # Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validación set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteración')
plt.ylabel('MSE error')
plt.legend()
plt.show()
```

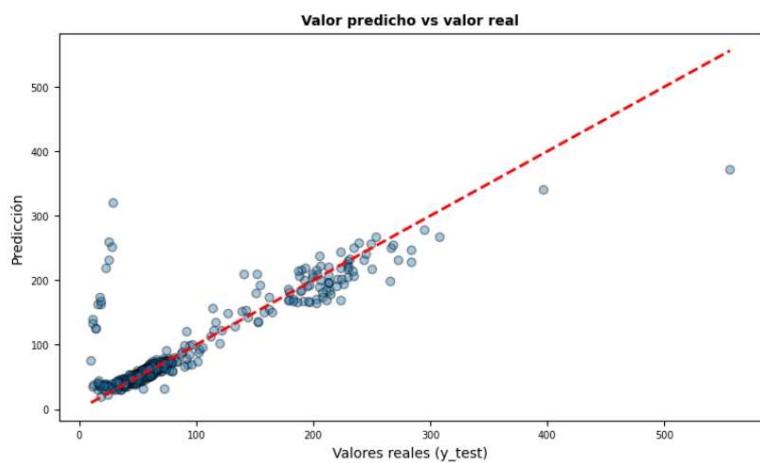
MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



```
In [12]: # Graficamos la relación entre los valores reales de "test" y los predicitos
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = modelo_RNeu_5.predict(X_test, verbose = 0).flatten()

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_test_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```



```
In [13]: MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
MAPE en test: 26.52 %
```

**MODELO 2: 3 Capas y 10 neuronas en la capa oculta**

```
In [14]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_10 = Sequential()

# 1º capa oculta
modelo_RNeu_10.add(Dense(10, input_dim=26, activation = 'relu'))

# Capa de salida
modelo_RNeu_10.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_10.compile(loss='mse',
                      optimizer='adam',
                      metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_10.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=50,
                           verbose=1)
entrenamiento = modelo_RNeu_10.fit(X_train, y_train,
                                     validation_data = (X_test, y_test),
                                     epochs = 2000,
                                     batch_size = 30,
                                     verbose = False,
                                     callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 1: 3 Capas y 10 neuronas en la capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido: ",
      round(end - start,4))
print("-----")

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_10.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_10.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_10.predict(X_train), y_train))))
print('-----')
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_10.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_10.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_10.predict(X_test), y_test))))
print('-----')

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_10.predict(X_train, verbose = 0)
```

```
print("La precisión del modelo de RED NEURONAL con los datos de entrenamiento es: ")
round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_10.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de test es: ")
round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadratico medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 10)	270
dense_3 (Dense)	(None, 1)	11

```
=====
Total params: 281
Trainable params: 281
Non-trainable params: 0
```

```
El modelo se está entrenando.....
```

```
Epoch 295: early stopping
```

```
El entrenamiento ha finalizado!
```

```
-----
Datos del MODELO 1: 3 Capas y 10 neuronas en la capa oculta
El tiempo empleado para calcular el modelo ha sido: 75.0375
```

```
-----
MAE en train: 10.16
```

```
MSE en train: 612.39
```

```
RMSE en train: 24.75
```

```
-----
MAE en test: 11.68
```

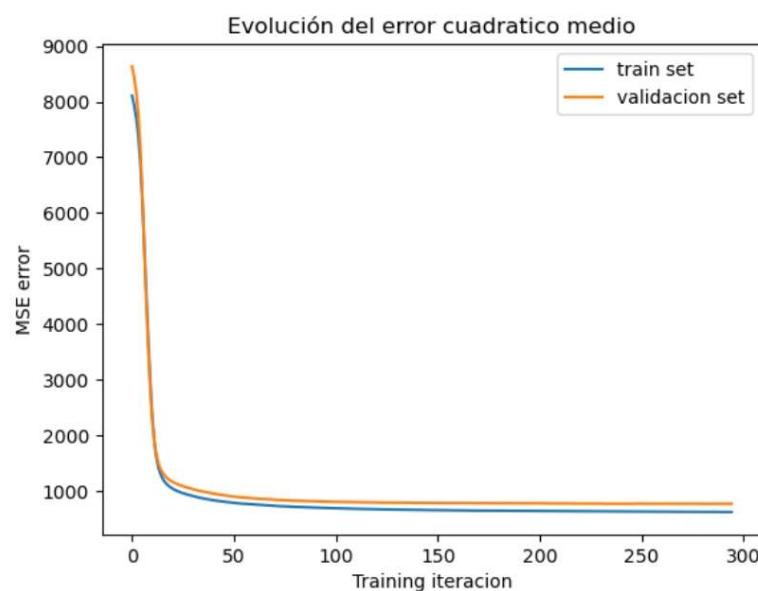
```
MSE en test: 765.27
```

```
RMSE en test: 27.66
```

```
-----
La precisión del modelo de RED NUERONAL con los datos de entrenamiento e
s de: 77.9
```

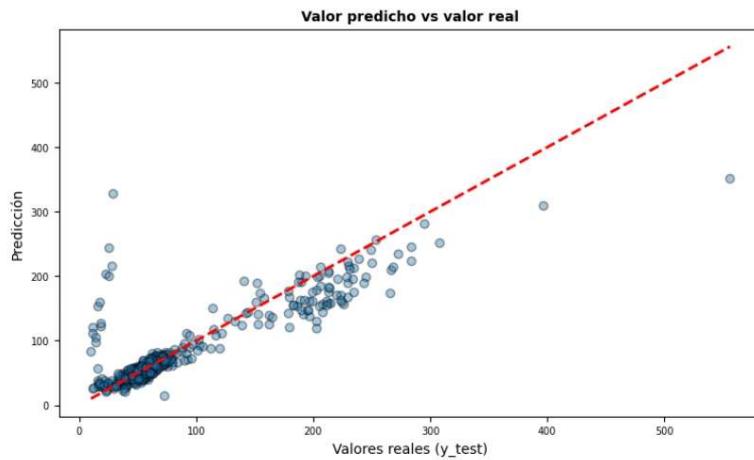
```
La precisión del modelo de RED NUERONAL con los datos de test es de: 75.
62
```

```
MAPE en test: 26.33 %
```



MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



### MODELO 3: 3 Capas y 25 neuronas en la capa oculta

```
In [15]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_25 = Sequential()

# 1º capa oculta
modelo_RNeu_25.add(Dense(25, input_dim=26, activation = 'relu'))

# Capa de salida
modelo_RNeu_25.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_25.compile(loss='mse',
                      optimizer='adam',
                      metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_25.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=50,
                           verbose=1)
entrenamiento = modelo_RNeu_25.fit(X_train, y_train,
                                    validation_data = (X_test, y_test),
                                    epochs = 2000,
                                    batch_size = 30,
                                    verbose = False,
                                    callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 1: 3 Capas y 25 neuronas en la capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido: ",
```

```
round(end - start,4))

# Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_25.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_25.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_25.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_25.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_25.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_25.predict(X_test), y_test))))
print("-----")

# Comprobamos la precisión/rendimiento del modelo generado.
y_pred = modelo_RNeu_25.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de entrenamiento es: ", round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_25.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de test es: ", round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validación set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteración')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---

```
Model: "sequential_2"
-----

| Layer (type)    | Output Shape | Param # |
|-----------------|--------------|---------|
| dense_4 (Dense) | (None, 25)   | 675     |
| dense_5 (Dense) | (None, 1)    | 26      |


-----  

Total params: 701  

Trainable params: 701  

Non-trainable params: 0
-----  

El modelo se está entrenando.....  

Epoch 272: early stopping  

El entrenamiento ha finalizado!
-----  

Datos del MODELO 1: 3 Capas y 25 neuronas en la capa oculta  

El tiempo empleado para calcular el modelo ha sido: 70.812  

MAE en train: 10.1  

MSE en train: 592.03  

RMSE en train: 24.33
-----  

MAE en test: 11.8  

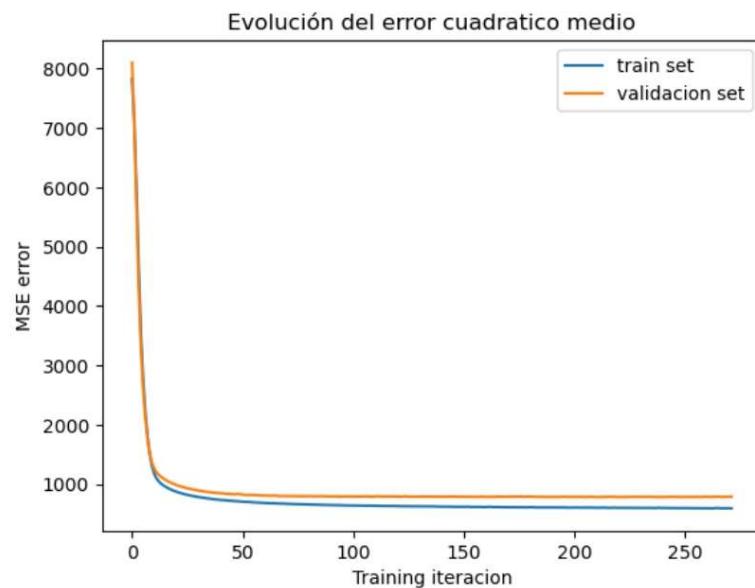
MSE en test: 791.64  

RMSE en test: 28.14
-----  

La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 78.63  

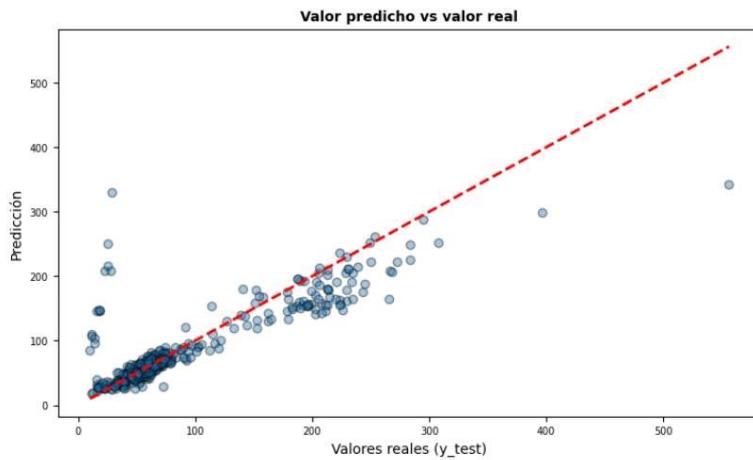
La precisión del modelo de RED NUERONAL con los datos de test es de: 74.78  

MAPE en test: 25.86 %
```



MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



#### MODELO 4: 3 Capas y 50 neuronas en la capa oculta

```
In [16]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_50 = Sequential()

# 1º capa oculta
modelo_RNeu_50.add(Dense(50, input_dim=26, activation = 'relu'))

# Capa de salida
modelo_RNeu_50.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_50.compile(loss='mse',
                      optimizer='adam',
                      metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_50.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=50,
                           verbose=1)
entrenamiento = modelo_RNeu_50.fit(X_train, y_train,
                                    validation_data = (X_test, y_test),
                                    epochs = 2000,
                                    batch_size = 30,
                                    verbose = False,
                                    callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 4: 3 Capas y 50 neuronas en la capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido: ",
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
round(end - start,4))
print("-----")

# Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_50.predict(
print('MSE en train:', round(mean_squared_error(modelo_RNeu_50.predict(X_
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_50.
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_50.predict(X_
print('MSE en test:', round(mean_squared_error(modelo_RNeu_50.predict(X_
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_50.p
print("-----")

# Comprobamos la precisión/rendimiento del modelo generado.
y_pred = modelo_RNeu_50.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de entrenamiento es: ", round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_50.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de test es: ", round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

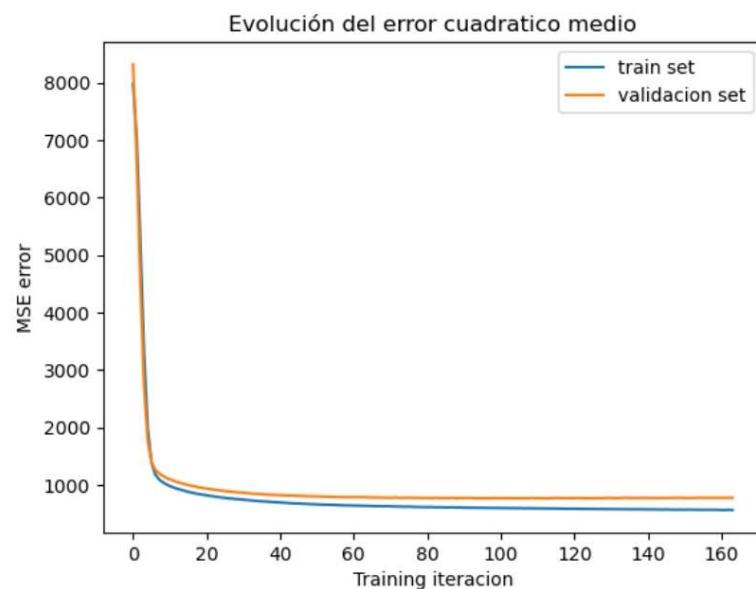
# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validación set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteración')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

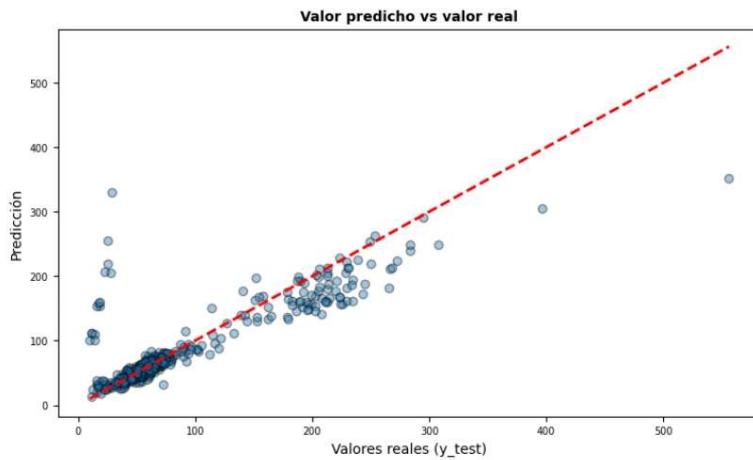
---

```
Model: "sequential_3"
-----
Layer (type)          Output Shape       Param #
dense_6 (Dense)      (None, 50)         1350
dense_7 (Dense)      (None, 1)          51
-----
Total params: 1,401
Trainable params: 1,401
Non-trainable params: 0
-----
El modelo se está entrenando......
Epoch 164: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 4: 3 Capas y 50 neuronas en la capa oculta
El tiempo empleado para calcular el modelo ha sido: 45.1214
-----
MAE en train: 9.32
MSE en train: 559.58
RMSE en train: 23.66
-----
MAE en test: 11.15
MSE en test: 778.53
RMSE en test: 27.9
-----
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 79.8
La precisión del modelo de RED NUERONAL con los datos de test es de: 75.19
MAPE en test: 25.55 %
```



MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



#### MODELO 5: 6 Capas y 5-10-20-40 neuronas en las capas ocultas

```
In [17]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_5_10_20_40 = Sequential()

# 1º capa oculta
modelo_RNeu_5_10_20_40.add(Dense(5, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_5_10_20_40.add(Dense(10, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_5_10_20_40.add(Dense(20, activation = 'relu'))
# 4º capa oculta
modelo_RNeu_5_10_20_40.add(Dense(40, activation = 'relu'))

# Capa de salida
modelo_RNeu_5_10_20_40.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_5_10_20_40.compile(loss='mse',
                                 optimizer='adam',
                                 metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_5_10_20_40.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor ='val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_5_10_20_40.fit(X_train, y_train,
                                             validation_data = (X_test, y_test),
                                             epochs = 2000,
                                             batch_size = 30,
                                             verbose = False,
                                             callbacks=[early_stop])
print("El entrenamiento ha finalizado!")
```

```

print('-----')
print('Datos del MODELO 5: 6 Capas y 5-10-20-40 neuronas en las capas ocultas')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_5_10_20_40,X_train),2))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_5_10_20_40,X_train),2))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_5_10_20_40,X_train)),2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_5_10_20_40,X_test),2))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_5_10_20_40,X_test),2))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_5_10_20_40,X_test)),2))
print("-----")

# Comprobamos la precisión/rendimiento del modelo generado.
y_pred = modelo_RNeu_5_10_20_40.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_5_10_20_40.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validación set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

Model: "sequential\_4"

Layer (type)	Output Shape	Param #
dense_8 (Dense)	(None, 5)	135
dense_9 (Dense)	(None, 10)	60
dense_10 (Dense)	(None, 20)	220
dense_11 (Dense)	(None, 40)	840
dense_12 (Dense)	(None, 1)	41
<hr/>		
Total params: 1,296		
Trainable params: 1,296		
Non-trainable params: 0		

El modelo se está entrenando.....

Epoch 83: early stopping

El entrenamiento ha finalizado!

-----  
Datos del MODELO 5: 6 Capas y 5-10-20-40 neuronas en las capas ocultas

El tiempo empleado para calcular el modelo ha sido: 26.1315

MAE en train: 10.68

MSE en train: 594.66

RMSE en train: 24.39

-----  
MAE en test: 12.48

MSE en test: 849.82

RMSE en test: 29.15

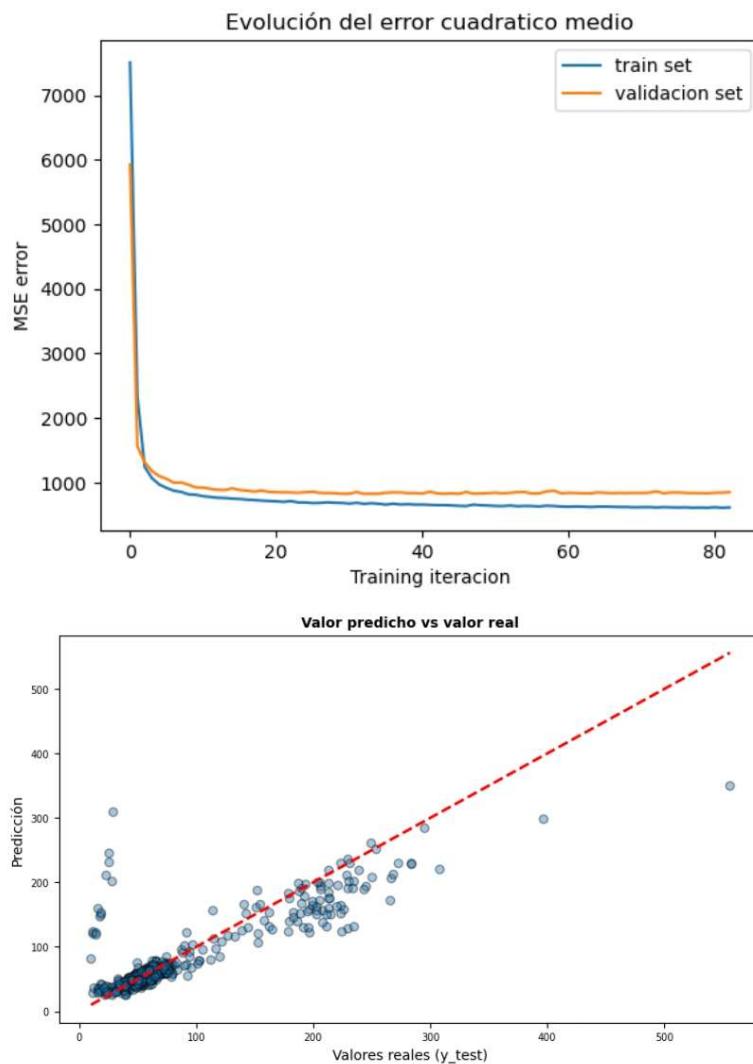
-----  
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 78.53

La precisión del modelo de RED NUERONAL con los datos de test es de: 72.92

MAPE en test: 27.87 %

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 6: 6 Capas y 40-20-10-5 neuronas en las capas ocultas**

```
In [18]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_40_20_10_5 = Sequential()

# 1º capa oculta
modelo_RNeu_40_20_10_5.add(Dense(40, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_40_20_10_5.add(Dense(20, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_40_20_10_5.add(Dense(10, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_40_20_10_5.add(Dense(5, activation = 'relu'))

# Capa de salida
modelo_RNeu_40_20_10_5.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_40_20_10_5.compile(loss='mse',
                                optimizer='adam',
                                metrics=['mse','mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_40_20_10_5.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_40_20_10_5.fit(X_train, y_train,
                                             validation_data = (X_test, y_test),
                                             epochs = 2000,
                                             batch_size = 30,
                                             verbose = False,
                                             callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 6: 6 Capas y 40-20-10-5 neuronas en las capas ocultas')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_40_20_10_5.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_40_20_10_5.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_40_20_10_5.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_40_20_10_5.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_40_20_10_5.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_40_20_10_5.predict(X_test), y_test))))
print("-----")

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_40_20_10_5.predict(X_train, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_40_20_10_5.predict(X_test, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el grafico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolucion del error cuadratico medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
plt.show()

# Graficamos la relación entre los valores reales de "test" y las predicciones
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = modelo_RNeu_40_20_10_5.predict(X_test, verbose = 0).flatten()

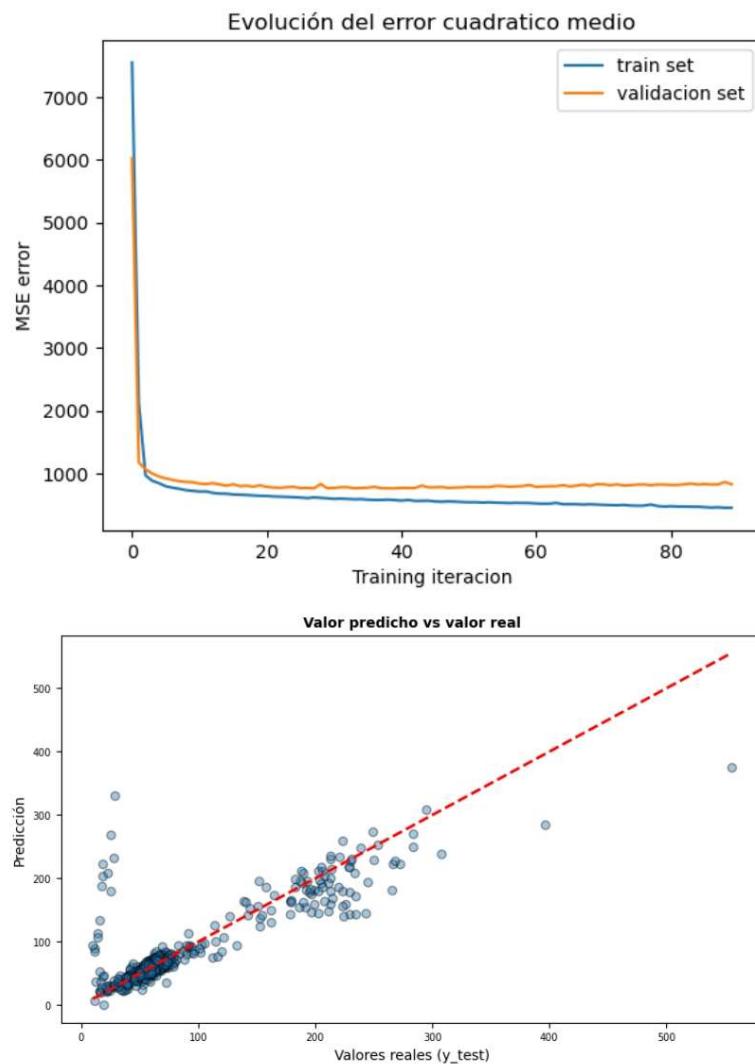
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

Model: "sequential_5"

Layer (type)          Output Shape       Param #
dense_13 (Dense)     (None, 40)           1080
dense_14 (Dense)     (None, 20)            820
dense_15 (Dense)     (None, 10)            210
dense_16 (Dense)     (None, 5)             55
dense_17 (Dense)     (None, 1)             6
=====
Total params: 2,171
Trainable params: 2,171
Non-trainable params: 0

El modelo se está entrenando.....
Epoch 90: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 6: 6 Capas y 40-20-10-5 neuronas en las capas ocultas
El tiempo empleado para calcular el modelo ha sido: 32.361
MAE en train: 8.56
MSE en train: 437.05
RMSE en train: 20.91
-----
MAE en test: 11.66
MSE en test: 828.58
RMSE en test: 28.79
-----
La precisión del modelo de RED NEURONAL con los datos de entrenamiento es de: 84.22
La precisión del modelo de RED NEURONAL con los datos de test es de: 73.6
MAPE en test: 27.31 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL



MODELO 7: 6 Capas y 10x4 neuronas en cada capa oculta

```
In [19]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_10x4 = Sequential()

# 1º capa oculta
modelo_RNeu_10x4.add(Dense(10, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_10x4.add(Dense(10, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_10x4.add(Dense(10, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_10x4.add(Dense(10, activation = 'relu'))

# Capa de salida
modelo_RNeu_10x4.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_10x4.compile(loss='mse',
                         optimizer='adam',
                         metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_10x4.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor ='val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_10x4.fit(X_train, y_train,
                                       validation_data = (X_test, y_test),
                                       epochs = 2000,
                                       batch_size = 30,
                                       verbose = False,
                                       callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 7: 6 Capas y 10x4 neuronas en cada capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_10x4.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_10x4.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_10x4.predict(X_train), y_train))))
print('-----')
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_10x4.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_10x4.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_10x4.predict(X_test), y_test))))
print('-----')

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_10x4.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_10x4.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='X_train')
plt.plot(entrenamiento.history['val_mse'], label='y_train')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteración')
plt.ylabel('MSE error')
plt.legend()

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
plt.show()

# Graficamos la relación entre los valores reales de "test" y las predicciones
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = modelo_RNeu_10x4.predict(X_test, verbose = 0).flatten()

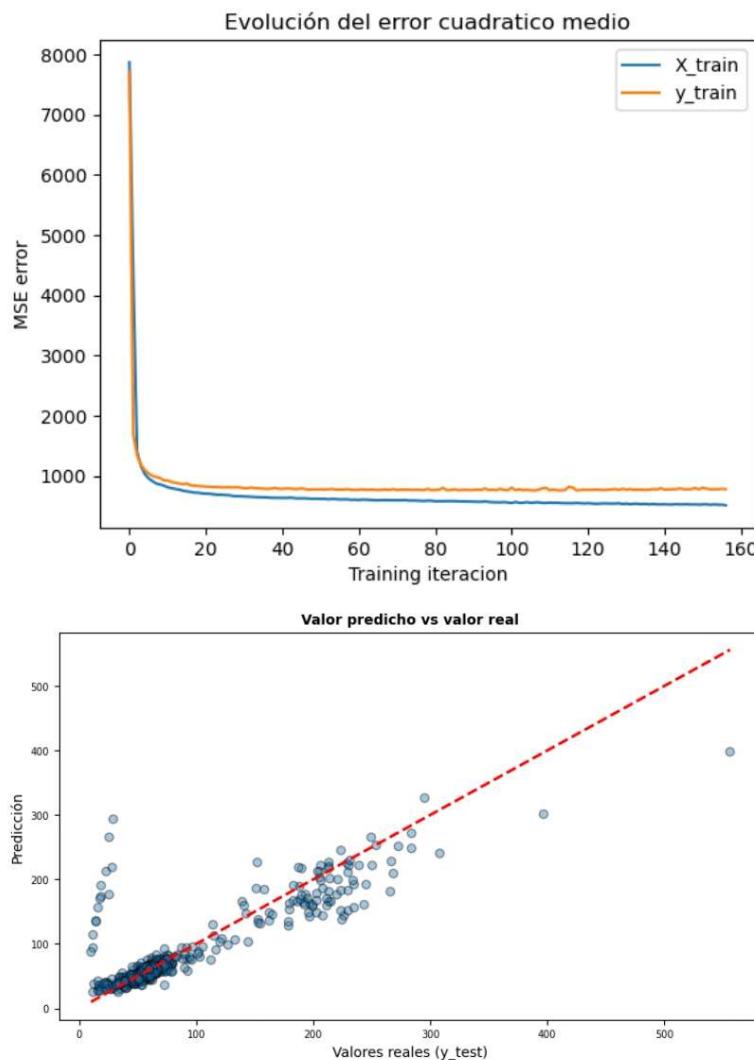
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

Model: "sequential_6"

Layer (type)          Output Shape         Param #
dense_18 (Dense)     (None, 10)           270
dense_19 (Dense)     (None, 10)           110
dense_20 (Dense)     (None, 10)           110
dense_21 (Dense)     (None, 10)           110
dense_22 (Dense)     (None, 1)            11
=====
Total params: 611
Trainable params: 611
Non-trainable params: 0

El modelo se está entrenando.....
Epoch 157: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 7: 6 Capas y 10x4 neuronas en cada capa oculta
El tiempo empleado para calcular el modelo ha sido: 47.6589
MAE en train: 9.82
MSE en train: 506.07
RMSE en train: 22.5
-----
MAE en test: 11.93
MSE en test: 781.4
RMSE en test: 27.95
-----
La precisión del modelo de RED NEURONAL con los datos de entrenamiento es de: 81.73
La precisión del modelo de RED NEURONAL con los datos de test es de: 75.1
MAPE en test: 28.11 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL



MODELO 8: 6 Capas y 20x4 neuronas en cada capa oculta

```
In [20]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_20x4 = Sequential()

# 1º capa oculta
modelo_RNeu_20x4.add(Dense(20, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_20x4.add(Dense(20, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_20x4.add(Dense(20, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_20x4.add(Dense(20, activation = 'relu'))

# Capa de salida
modelo_RNeu_20x4.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_20x4.compile(loss='mse',
                         optimizer='adam',
                         metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_20x4.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_20x4.fit(X_train, y_train,
                                       validation_data = (X_test, y_test),
                                       epochs = 2000,
                                       batch_size = 30,
                                       verbose = False,
                                       callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 8: 6 Capas y 20x4 neuronas en cada capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_20x4.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_20x4.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x4.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_20x4.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_20x4.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x4.predict(X_test), y_test))))
print("-----")

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_20x4.predict(X_train, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_20x4.predict(X_test, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el grafico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolucion del error cuadratico medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

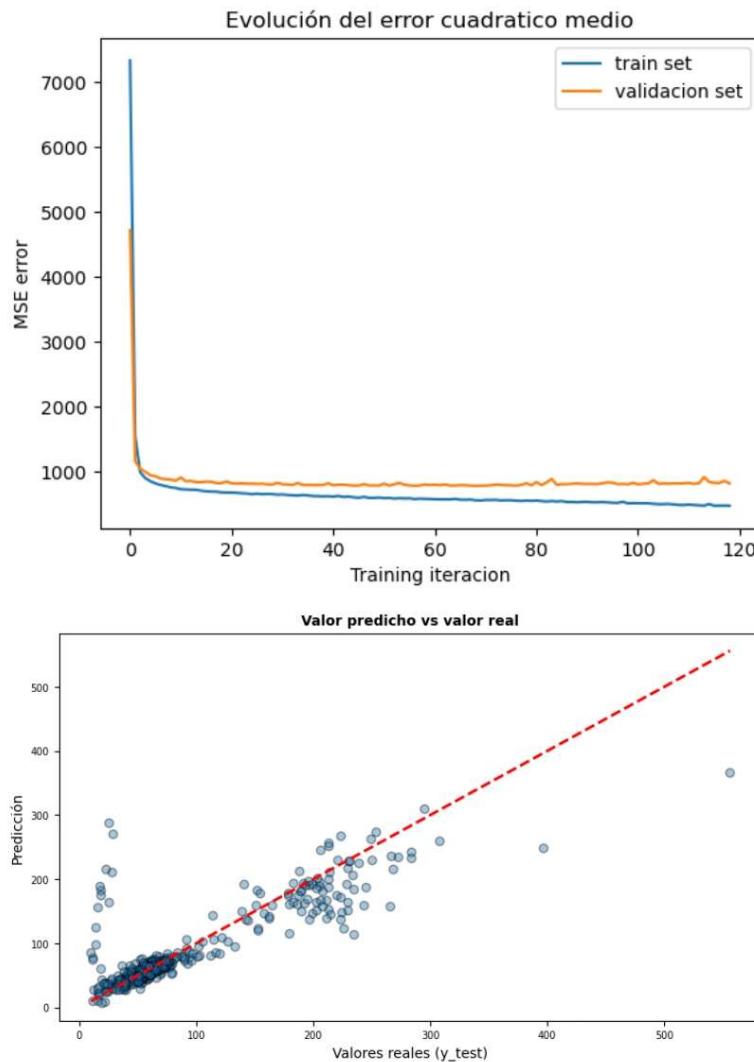
Model: "sequential_7"

Layer (type)          Output Shape       Param #
=====            =====
dense_23 (Dense)     (None, 20)           540
dense_24 (Dense)     (None, 20)           420
dense_25 (Dense)     (None, 20)           420
dense_26 (Dense)     (None, 20)           420
dense_27 (Dense)     (None, 1)            21
=====
Total params: 1,821
Trainable params: 1,821
Non-trainable params: 0

El modelo se está entrenando.....
Epoch 119: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 8: 6 Capas y 20x4 neuronas en cada capa oculta
El tiempo empleado para calcular el modelo ha sido: 34.307
MAE en train: 9.06
MSE en train: 441.05
RMSE en train: 21.0
-----
MAE en test: 11.61
MSE en test: 813.93
RMSE en test: 28.53
-----
La precisión del modelo de RED NEURONAL con los datos de entrenamiento es de: 84.08
La precisión del modelo de RED NEURONAL con los datos de test es de: 74.07
MAPE en test: 25.92 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 9: 6 Capas y 40x4 neuronas en cada capa oculta**

```
In [21]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_40x4 = Sequential()

# 1º capa oculta
modelo_RNeu_40x4.add(Dense(40, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_40x4.add(Dense(40, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_40x4.add(Dense(40, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_40x4.add(Dense(40, activation = 'relu'))

# Capa de salida
modelo_RNeu_40x4.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_40x4.compile(loss='mse',
                         optimizer='adam',
                         metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_40x4.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor ='val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_40x4.fit(X_train, y_train,
                                       validation_data = (X_test, y_test),
                                       epochs = 2000,
                                       batch_size = 30,
                                       verbose = False,
                                       callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 9: 6 Capas y 40x4 neuronas en cada capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_40x4.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_40x4.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_40x4.predict(X_train), y_train))))
print('-----')
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_40x4.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_40x4.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_40x4.predict(X_test), y_test))))
print('-----')

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_40x4.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_40x4.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadratico medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()

```

```
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

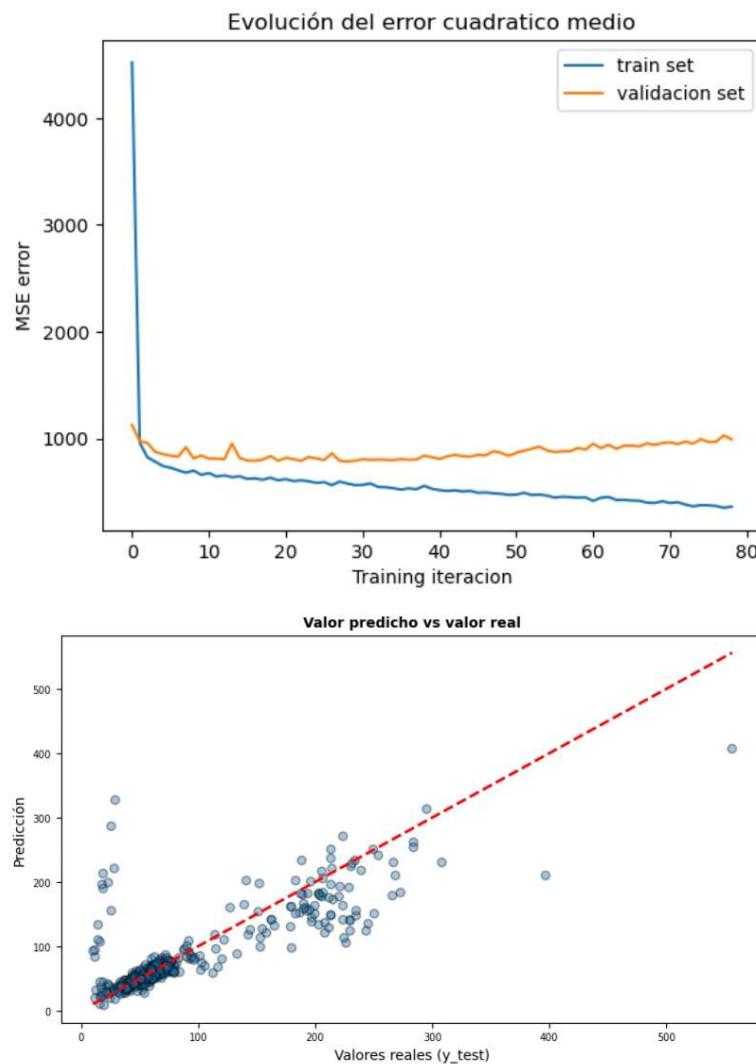
Model: "sequential_8"

Layer (type)          Output Shape       Param #
=====          =====       =====
dense_28 (Dense)     (None, 40)        1080
dense_29 (Dense)     (None, 40)        1640
dense_30 (Dense)     (None, 40)        1640
dense_31 (Dense)     (None, 40)        1640
dense_32 (Dense)     (None, 1)         41
=====
Total params: 6,041
Trainable params: 6,041
Non-trainable params: 0

El modelo se está entrenando.....
Epoch 79: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 9: 6 Capas y 40x4 neuronas en cada capa oculta
El tiempo empleado para calcular el modelo ha sido: 24.5997
MAE en train: 8.44
MSE en train: 327.74
RMSE en train: 18.1
-----
MAE en test: 13.01
MSE en test: 992.06
RMSE en test: 31.5
-----
La precisión del modelo de RED NEURONAL con los datos de entrenamiento es de: 88.17
La precisión del modelo de RED NEURONAL con los datos de test es de: 68.39
MAPE en test: 27.31 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 10: 12 Capas y 20 neuronas en cada capa oculta**

```
In [26]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_20x10 = Sequential()

# 1º capa oculta
modelo_RNeu_20x10.add(Dense(20, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 5º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 6º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 7º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 8º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 9º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 10º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))

# Capa de salida
modelo_RNeu_20x10.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_20x10.compile(loss='mse',
                           optimizer='adam',
                           metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_20x10.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_20x10.fit(X_train, y_train,
                                       validation_data = (X_test, y_test),
                                       epochs = 2000,
                                       batch_size = 30,
                                       verbose = False,
                                       callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 11: 12 Capas y 20x10 neuronas en las capas oculares')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_20x10.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_20x10.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x10.predict(X_train), y_train))))
print('-----')
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_20x10.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_20x10.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x10.predict(X_test), y_test))))
print('-----')

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_20x10.predict(X_train, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_20x10.predict(X_test, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

```

```
MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
cv = KFold(n_splits=5, random_state=123, shuffle=True)
y_test_pred = modelo_RNeu_20x10.predict(X_test, verbose = 0).flatten()

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---

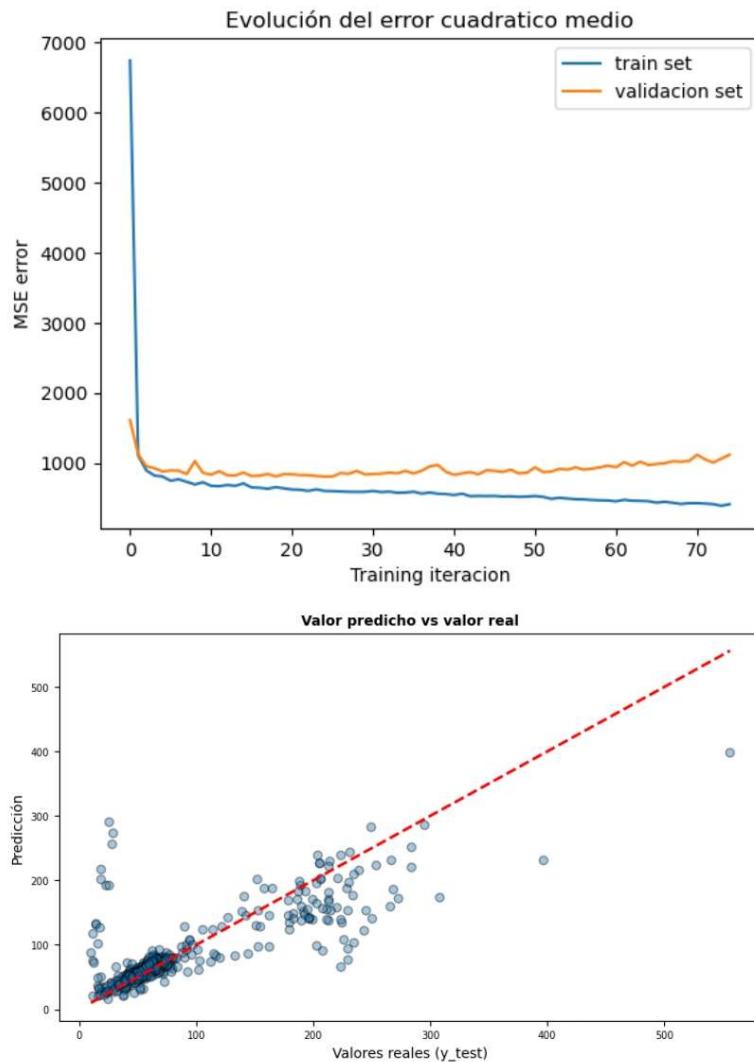
```
Model: "sequential_13"
-----

| Layer (type)      | Output Shape | Param # |
|-------------------|--------------|---------|
| dense_95 (Dense)  | (None, 20)   | 540     |
| dense_96 (Dense)  | (None, 20)   | 420     |
| dense_97 (Dense)  | (None, 20)   | 420     |
| dense_98 (Dense)  | (None, 20)   | 420     |
| dense_99 (Dense)  | (None, 20)   | 420     |
| dense_100 (Dense) | (None, 20)   | 420     |
| dense_101 (Dense) | (None, 20)   | 420     |
| dense_102 (Dense) | (None, 20)   | 420     |
| dense_103 (Dense) | (None, 20)   | 420     |
| dense_104 (Dense) | (None, 20)   | 420     |
| dense_105 (Dense) | (None, 1)    | 21      |


-----  
Total params: 4,341  
Trainable params: 4,341  
Non-trainable params: 0
-----  
El modelo se está entrenando.....  
Epoch 75: early stopping  
El entrenamiento ha finalizado!
-----  
Datos del MODELO 11: 12 Capas y 20x10 neuronas en las capas ocultas  
El tiempo empleado para calcular el modelo ha sido: 28.2739  
MAE en train: 10.3  
MSE en train: 382.69  
RMSE en train: 19.56
-----  
MAE en test: 15.2  
MSE en test: 1118.64  
RMSE en test: 33.45
-----  
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 86.19  
La precisión del modelo de RED NUERONAL con los datos de test es de: 64.36  
MAPE en test: 31.01 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 11: 12 Capas y 25x5 + 50x5 neuronas en las capas ocultas**

```
In [27]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_25x5_50x5 = Sequential()

# 1º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(25, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(50, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(25, activation = 'relu'))
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
# 4º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(50, activation = 'relu'))
# 5º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(25, activation = 'relu'))
# 6º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(50, activation = 'relu'))
# 7º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(25, activation = 'relu'))
# 8º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(50, activation = 'relu'))
# 9º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(25, activation = 'relu'))
# 10º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(50, activation = 'relu'))

# Capa de salida
modelo_RNeu_25x5_50x5.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_25x5_50x5.compile(loss='mse',
                                optimizer='adam',
                                metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_25x5_50x5.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor ='val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_25x5_50x5.fit(X_train, y_train,
                                             validation_data = (X_test, y_test),
                                             epochs = 2000,
                                             batch_size = 30,
                                             verbose = False,
                                             callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 11: 12 Capas y 25x5 + 50x5 neuronas en las capas')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_25x5_50x5.p
print('MSE en train:', round(mean_squared_error(modelo_RNeu_25x5_50x5.p
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_25x5_
print('-----')
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_25x5_50x5.p
print('MSE en test:', round(mean_squared_error(modelo_RNeu_25x5_50x5.p
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_25x5_
print('-----')

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_25x5_50x5.predict(X_train, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de entrenamiento es:", round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_25x5_50x5.predict(X_test, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de test es:", round(r2_score(y_test, y_pred)*100, 2))
```

```
round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---

```
Model: "sequential_14"
-----

| Layer (type)      | Output Shape | Param # |
|-------------------|--------------|---------|
| dense_106 (Dense) | (None, 25)   | 675     |
| dense_107 (Dense) | (None, 50)   | 1300    |
| dense_108 (Dense) | (None, 25)   | 1275    |
| dense_109 (Dense) | (None, 50)   | 1300    |
| dense_110 (Dense) | (None, 25)   | 1275    |
| dense_111 (Dense) | (None, 50)   | 1300    |
| dense_112 (Dense) | (None, 25)   | 1275    |
| dense_113 (Dense) | (None, 50)   | 1300    |
| dense_114 (Dense) | (None, 25)   | 1275    |
| dense_115 (Dense) | (None, 50)   | 1300    |
| dense_116 (Dense) | (None, 1)    | 51      |


-----
Total params: 12,326
Trainable params: 12,326
Non-trainable params: 0
-----
El modelo se está entrenando.....  

Epoch 62: early stopping  

El entrenamiento ha finalizado!
-----
Datos del MODELO 11: 12 Capas y 25x5 + 50x5 neuronas en las capas ocultas  

El tiempo empleado para calcular el modelo ha sido: 23.8389  

MAE en train: 8.06  

MSE en train: 247.95  

RMSE en train: 15.75
-----
MAE en test: 14.34  

MSE en test: 1138.28  

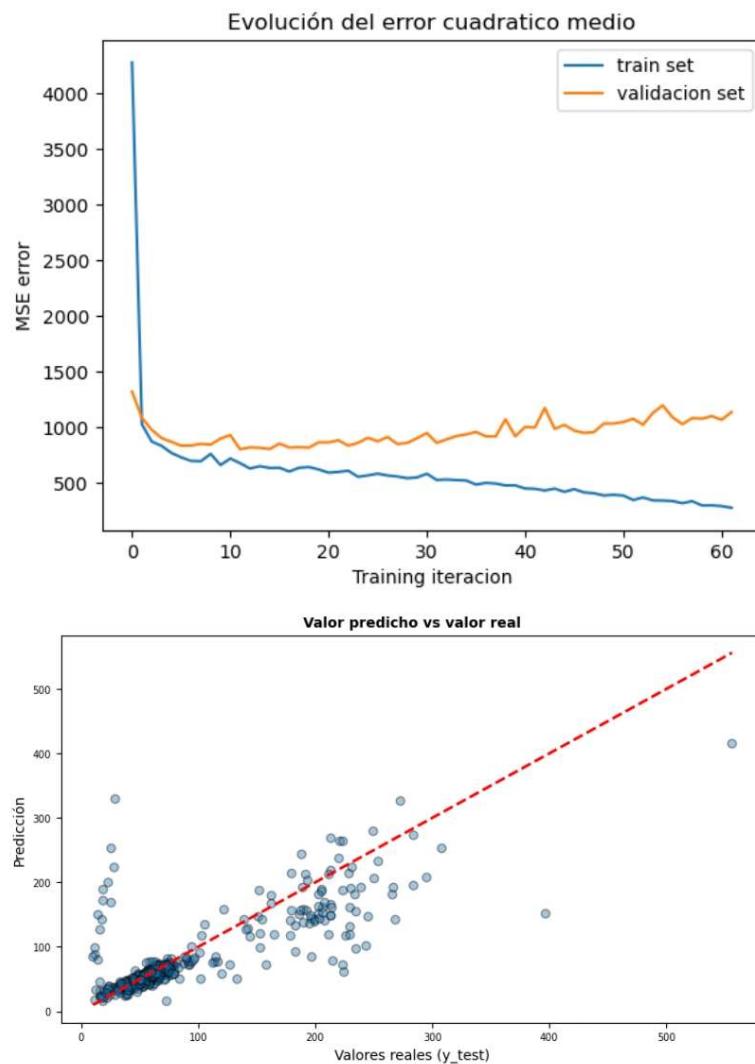
RMSE en test: 33.74
-----
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 91.05  

La precisión del modelo de RED NUERONAL con los datos de test es de: 63.73  

MAPE en test: 27.16 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 12: 12 Capas y 10x10 neuronas en cada capa oculta**

```
In [24]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_10x10 = Sequential()

# 1º capa oculta
modelo_RNeu_10x10.add(Dense(10, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 5º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 6º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 7º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 8º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 9º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 10º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))

# Capa de salida
modelo_RNeu_10x10.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_10x10.compile(loss='mse',
                           optimizer='adam',
                           metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_10x10.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_10x10.fit(X_train, y_train,
                                       validation_data = (X_test, y_test),
                                       epochs = 2000,
                                       batch_size = 30,
                                       verbose = False,
                                       callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 11: 12 Capas y 10x10 neuronas en las capas oculares')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido: ",
      round(end - start,4))
print("-----")

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_10x10.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_10x10.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_10x10.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_10x10.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_10x10.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_10x10.predict(X_test), y_test))))
print("-----")

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_10x10.predict(X_train, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de entrenamiento es: ",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_10x10.predict(X_test, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de test es: ",
      round(r2_score(y_test, y_pred)*100, 2))

```

```
round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

Model: "sequential\_11"

Layer (type)	Output Shape	Param #
dense_55 (Dense)	(None, 10)	270
dense_56 (Dense)	(None, 10)	110
dense_57 (Dense)	(None, 10)	110
dense_58 (Dense)	(None, 10)	110
dense_59 (Dense)	(None, 10)	110
dense_60 (Dense)	(None, 10)	110
dense_61 (Dense)	(None, 10)	110
dense_62 (Dense)	(None, 10)	110
dense_63 (Dense)	(None, 10)	110
dense_64 (Dense)	(None, 10)	110
dense_65 (Dense)	(None, 1)	11

Total params: 1,271  
Trainable params: 1,271  
Non-trainable params: 0

El modelo se está entrenando.....  
Epoch 83: early stopping  
El entrenamiento ha finalizado!

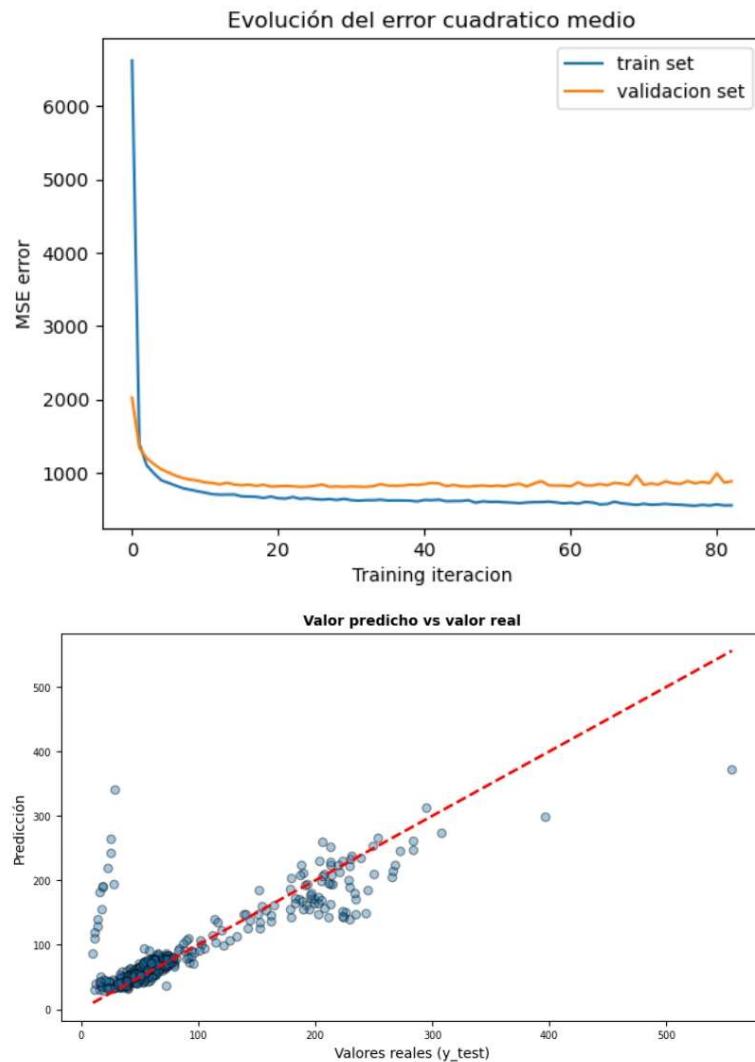
Datos del MODELO 11: 12 Capas y 10x10 neuronas en las capas ocultas  
El tiempo empleado para calcular el modelo ha sido: 35.5407

-----  
MAE en train: 11.45  
MSE en train: 556.89  
RMSE en train: 23.6

-----  
MAE en test: 13.29  
MSE en test: 884.25  
RMSE en test: 29.74

-----  
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 79.9  
La precisión del modelo de RED NUERONAL con los datos de test es de: 71.83  
MAPE en test: 32.23 %

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL



MODELO 13: 20 Capas y 20x9+30x9+10x9 neuronas en cada capa oculta

```
In [25]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_20x9_30x9_10x9 = Sequential()

# 1º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
```

```

# 4º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 5º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 6º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 7º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 8º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 9º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 10º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 11º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 12º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 13º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 14º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 15º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 16º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 17º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 18º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 19º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 20º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 21º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 22º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 23º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 24º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 25º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 26º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 27º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 28º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))

# Capa de salida
modelo_RNeu_20x9_30x9_10x9.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_20x9_30x9_10x9.compile(loss='mse',
                                      optimizer='adam',
                                      metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_20x9_30x9_10x9.summary()

```

```

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_20x9_30x9_10x9.fit(X_train, y_train,
                                                validation_data = (X_test, y_test),
                                                epochs = 2000,
                                                batch_size = 30,
                                                verbose = False,
                                                callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 13: 20 Capas y 20x9_30x9_10x9 neuronas en cada capa')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_20x9_30x9_10x9,X_train),2))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_20x9_30x9_10x9,X_train),2))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x9_30x9_10x9,X_train)),2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_20x9_30x9_10x9,X_test),2))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_20x9_30x9_10x9,X_test),2))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x9_30x9_10x9,X_test)),2))
print("-----")

# Comprobamos la precisión/rendimiento del modelo generado.
y_pred = modelo_RNeu_20x9_30x9_10x9.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_20x9_30x9_10x9.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validación set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteración')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y las predicciones
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

```

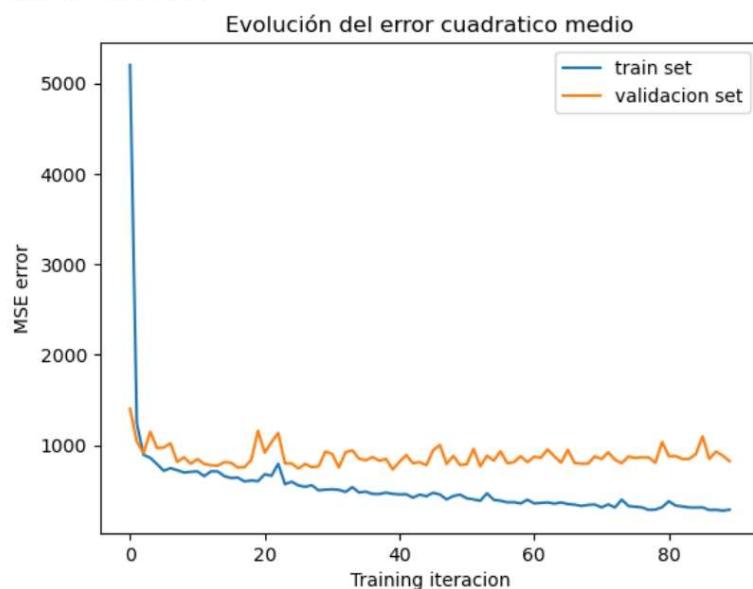
MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---

Model: "sequential\_12"

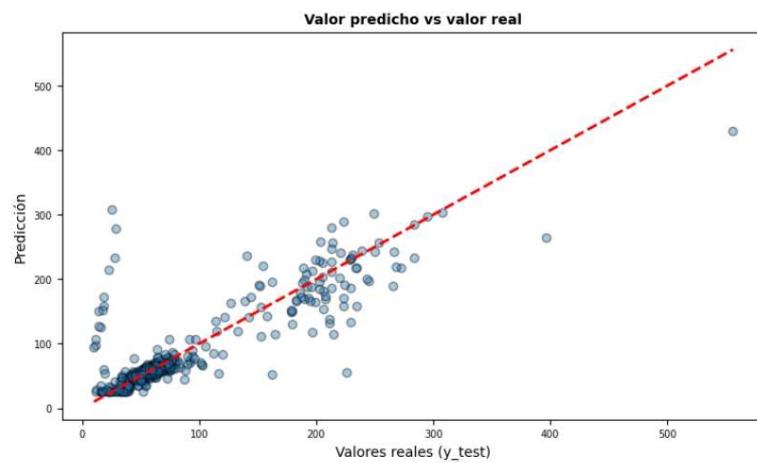
Layer (type)	Output Shape	Param #
dense_66 (Dense)	(None, 20)	540
dense_67 (Dense)	(None, 30)	630
dense_68 (Dense)	(None, 10)	310
dense_69 (Dense)	(None, 20)	220
dense_70 (Dense)	(None, 30)	630
dense_71 (Dense)	(None, 10)	310
dense_72 (Dense)	(None, 20)	220
dense_73 (Dense)	(None, 30)	630
dense_74 (Dense)	(None, 10)	310
dense_75 (Dense)	(None, 20)	220
dense_76 (Dense)	(None, 30)	630
dense_77 (Dense)	(None, 10)	310
dense_78 (Dense)	(None, 20)	220
dense_79 (Dense)	(None, 30)	630
dense_80 (Dense)	(None, 10)	310
dense_81 (Dense)	(None, 20)	220
dense_82 (Dense)	(None, 30)	630
dense_83 (Dense)	(None, 10)	310
dense_84 (Dense)	(None, 20)	220
dense_85 (Dense)	(None, 30)	630
dense_86 (Dense)	(None, 10)	310
dense_87 (Dense)	(None, 20)	220
dense_88 (Dense)	(None, 30)	630
dense_89 (Dense)	(None, 10)	310
dense_90 (Dense)	(None, 20)	220
dense_91 (Dense)	(None, 30)	630
dense_92 (Dense)	(None, 10)	310
dense_93 (Dense)	(None, 20)	220
dense_94 (Dense)	(None, 1)	21

```
=====
Total params: 11,001
Trainable params: 11,001
Non-trainable params: 0
-----
El modelo se está entrenando.....
Epoch 90: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 13: 20 Capas y 20x9_30x9_10x9 neuronas en cada capa oculata
El tiempo empleado para calcular el modelo ha sido: 52.7955
MAE en train: 6.81
MSE en train: 229.61
RMSE en train: 15.15
-----
MAE en test: 11.87
MSE en test: 824.83
RMSE en test: 28.72
-----
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 91.71
La precisión del modelo de RED NUERONAL con los datos de test es de: 73.72
MAPE en test: 26.38 %
```



# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL



In [ ]:

In [ ]:

## ANEXO XVI: RED NEURONAL 70-30

---

### RED NEURONAL

En este apartado estudiaremos diferentes modelos de Redes Neuronales. Comenzaremos con la más sencilla de una sola capa oculta y pocas neuronas. Posteriormente iremos aumentando las capas y las neuronas en ellas hasta quedarnos con la que estimemos más precisa.

#### Cargamos las librerías necesarias

```
In [1]: import pandas as pd
import numpy as np

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error, mean_absolute_error

from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
import statsmodels.api as sm

from scipy import stats
import time

import matplotlib.pyplot as plt
import seaborn as sns
```

#### Cargamos el dataset con los datos

```
In [2]: df_def = pd.read_csv(r'C:\Users\User\1.PYTHON\00.TFM\00.Archivos_utilizad
```

```
In [3]: # Comprobamos el dataframe cargado
df_def.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3652 entries, 0 to 3651
Data columns (total 28 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   fecha        3652 non-null   object  
 1   Pre_elec     3652 non-null   float64 
 2   dia_sem      3652 non-null   int64  
 3   Dem          3652 non-null   float64 
 4   Prec_petr    3652 non-null   float64 
 5   Prec_gas     3652 non-null   float64 
 6   Prec_carb    3652 non-null   float64 
 7   Prod_eol     3652 non-null   float64 
 8   Prod_sol     3652 non-null   float64 
 9   Prod_hidr    3652 non-null   float64 
 10  Prod_ofr     3652 non-null   float64 
 11  Prod_nucl    3652 non-null   float64 
 12  Prod_pet     3652 non-null   float64 
 13  Prod_gas     3652 non-null   float64 
 14  Prod_carb    3652 non-null   float64 
 15  Prod_comb    3652 non-null   float64 
 16  Prod_cog     3652 non-null   float64 
 17  Prod_no_ren  3652 non-null   float64 
 18  Temp_gen     3652 non-null   float64 
 19  Vel_media_Val 3652 non-null   float64 
 20  Vel_media_Alb 3652 non-null   float64 
 21  Vel_media_Zar 3652 non-null   float64 
 22  Vel_media_Cor 3652 non-null   float64 
 23  Vel_media_Hue 3652 non-null   float64 
 24  Res_hidr     3652 non-null   float64 
 25  Der_CO2       3652 non-null   float64 
 26  Ibex          3652 non-null   float64 
 27  Int_ee        3652 non-null   float64 
dtypes: float64(26), int64(1), object(1)
memory usage: 799.0+ KB
```

## Procesamos los datos

```
In [4]: # Definimos la variable de salida y las de entrada
X = df_def[['dia_sem', 'Dem', 'Prec_petr', 'Prec_gas', 'Prec_carb', 'Pr
    'Prod_sol', 'Prod_hidr', 'Prod_ofr', 'Prod_nucl', 'Prod_pet',
    'Prod_carb', 'Prod_comb', 'Prod_cog', 'Prod_no_ren', 'Temp_
    'Vel_media_Alb', 'Vel_media_Zar', 'Vel_media_Cor', 'Vel_med
    'Der_CO2', 'Ibex', 'Int_ee']].values
y = df_def[['Pre_elec']].values
```

```
In [5]: # Comprobamos las dos matrices de datos creadas con los datos
print(X.shape)
y = y.reshape(-1,1)
print(y.shape)
```

```
(3652, 26)
(3652, 1)
```

```
In [6]: # Realizamos la estandarización de los datos necesaria para el empleo
# de la Red Neuronal
X = stats.zscore(X, axis=0)
```

```
In [7]: # Dividimos los datos en "train_2" (70%) y "test_2" (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size = 0.3,  
random_state = 1234)
```

## Desarrollamos los modelos

### MODELO 1: 3 Capas y 5 neuronas en la capa oculta

```
In [8]: # Calcularemos el tiempo que tarda la computadora para desarrollar el mo  
start = time.time()  
  
# Creamos el modelo  
modelo_RNeu_5 = Sequential()  
  
# 1º capa oculta  
modelo_RNeu_5.add(Dense(5, input_dim=26, activation = 'relu'))  
  
# Capa de salida  
modelo_RNeu_5.add(Dense(1, activation = 'relu'))  
  
# Compilacion del modelo  
modelo_RNeu_5.compile(loss='mae', # mean_squared_error  
                      optimizer='adam',  
                      metrics=['mse', 'accuracy'])  
  
# Obtenemos el resumen de la red neuronal creada  
modelo_RNeu_5.summary()  
  
# Ajuste del modelo  
print("El modelo se está entrenando.....")  
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=50,  
                           verbose=1)  
entrenamiento = modelo_RNeu_5.fit(X_train, y_train,  
                                    validation_data = (X_test, y_test),  
                                    epochs = 2000,  
                                    batch_size = 30,  
                                    verbose = False,  
                                    callbacks=[early_stop])  
print("El entrenamiento ha finalizado!")  
  
print('-----')  
print('Datos del MODELO 1: 3 Capas y 5 neuronas en la capa oculta')  
  
# Medimos el tiempo que ha tardado el modelo  
end = time.time()  
print("El tiempo empleado para calcular el modelo ha sido: ",  
      round(end - start,4))
```

```

Model: "sequential"
-----
Layer (type)          Output Shape       Param #
dense (Dense)         (None, 5)           135
dense_1 (Dense)        (None, 1)            6
-----
Total params: 141
Trainable params: 141
Non-trainable params: 0
-----
El modelo se está entrenando.....
Epoch 843: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 1: 3 Capas y 5 neuronas en la capa oculta
El tiempo empleado para calcular el modelo ha sido: 226.1801

```

```

In [9]: # Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_5.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_5.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_5.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_5.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_5.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_5.predict(X_test), y_test))))

```

MAE en train: 9.27  
MSE en train: 667.71  
RMSE en train: 25.84  
-----  
MAE en test: 10.35  
MSE en test: 768.0  
RMSE en test: 27.71

```

In [10]: # Comprobamos la precisión/rendimiento del modelo generado.
y_pred = modelo_RNeu_5.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: " + str(round(r2_score(y_train, y_pred)*100, 2)))

```

```

y_pred = modelo_RNeu_5.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de test es de: " + str(round(r2_score(y_test, y_pred)*100, 2)))

```

La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 74.38  
La precisión del modelo de RED NUERONAL con los datos de test es de: 77.38

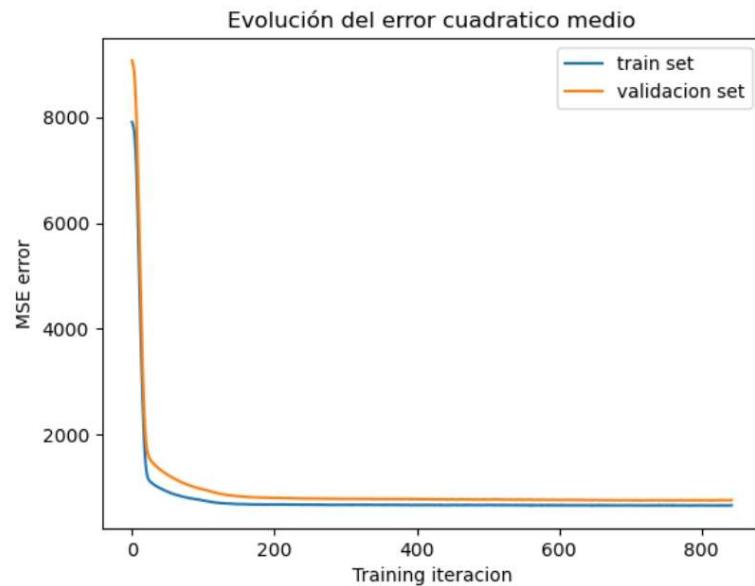
```

In [11]: # Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validación set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteración')
plt.ylabel('MSE error')
plt.legend()
plt.show()

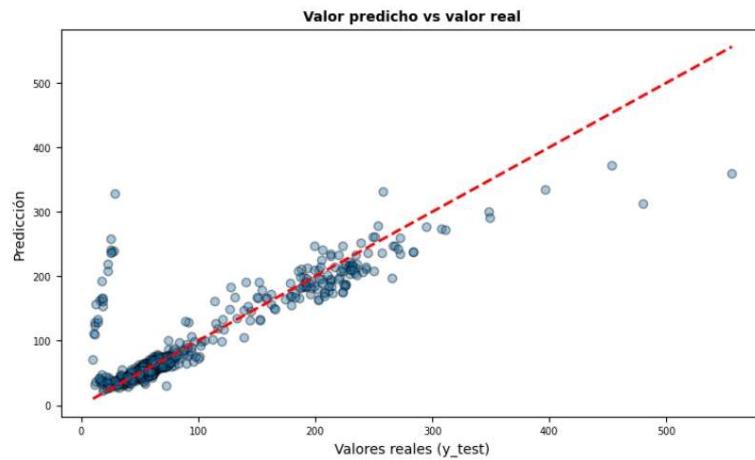
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



```
In [12]: # Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```



```
In [13]: MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")
```

MAPE en test: 27.68 %

### MODELO 2: 3 Capas y 10 neuronas en la capa oculta

```
In [14]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_10 = Sequential()

# 1º capa oculta
modelo_RNeu_10.add(Dense(10, input_dim=26, activation = 'relu'))

# Capa de salida
modelo_RNeu_10.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_10.compile(loss='mse',
                      optimizer='adam',
                      metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_10.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=50,
                           verbose=1)
entrenamiento = modelo_RNeu_10.fit(X_train, y_train,
                                     validation_data = (X_test, y_test),
                                     epochs = 2000,
                                     batch_size = 30,
                                     verbose = False,
                                     callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 1: 3 Capas y 10 neuronas en la capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido: ",
      round(end - start,4))
print("-----")

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_10.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_10.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_10.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_10.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_10.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_10.predict(X_test), y_test))))
print("-----")

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_10.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de entrenamiento es: ", round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_10.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de test es: ", round(r2_score(y_test, y_pred)*100, 2))
```

```

round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

Model: "sequential_1"
-----  

Layer (type)           Output Shape        Param #
-----  

dense_2 (Dense)        (None, 10)           270  

-----  

dense_3 (Dense)        (None, 1)            11  

-----  

Total params: 281  

Trainable params: 281  

Non-trainable params: 0  

-----  

El modelo se está entrenando.....  

Epoch 343: early stopping  

El entrenamiento ha finalizado!  

-----  

Datos del MODELO 1: 3 Capas y 10 neuronas en la capa oculta  

El tiempo empleado para calcular el modelo ha sido: 87.9459  

-----  

MAE en train: 9.92  

MSE en train: 585.8  

RMSE en train: 24.2  

-----  

MAE en test: 11.62  

MSE en test: 806.39  

RMSE en test: 28.4  

-----  

La precisión del modelo de RED NEURONAL con los datos de entrenamiento es de: 77.52  

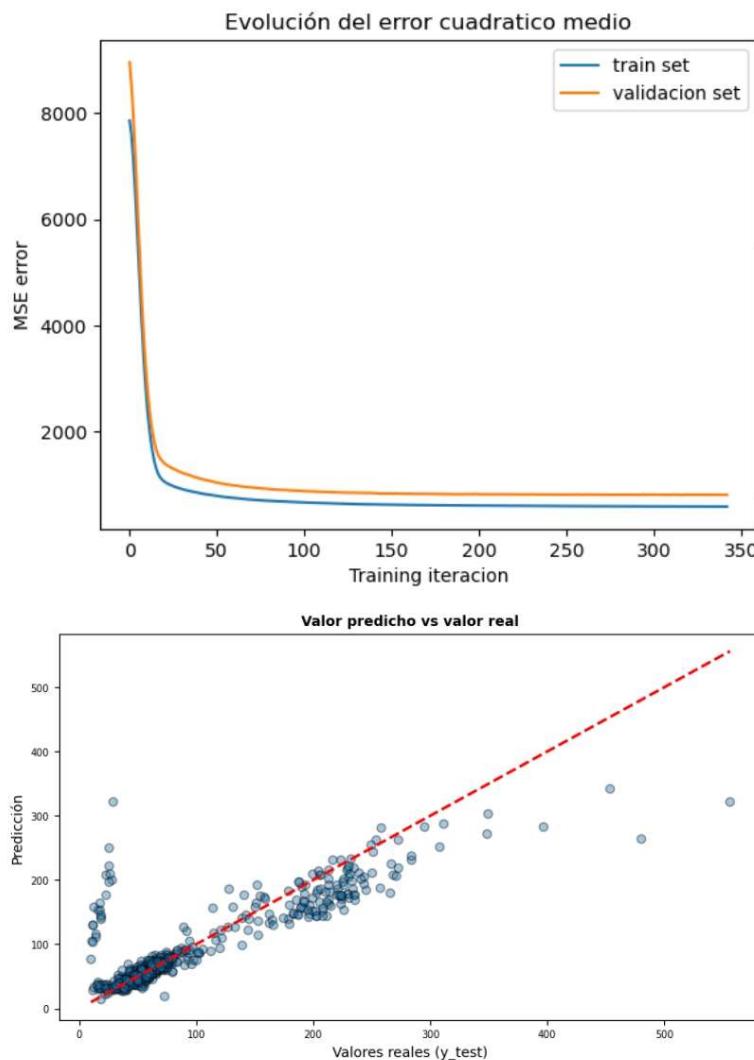
La precisión del modelo de RED NEURONAL con los datos de test es de: 76.25  

MAPE en test: 27.05 %

```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 3: 3 Capas y 25 neuronas en la capa oculta**

```
In [15]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_25 = Sequential()

# 1º capa oculta
modelo_RNeu_25.add(Dense(25, input_dim=26, activation = 'relu'))

# Capa de salida
modelo_RNeu_25.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
```

```

modelo_RNeu_25.compile(loss='mse',
                       optimizer='adam',
                       metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_25.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=50,
                           verbose=1)
entrenamiento = modelo_RNeu_25.fit(X_train, y_train,
                                     validation_data = (X_test, y_test),
                                     epochs = 2000,
                                     batch_size = 30,
                                     verbose = False,
                                     callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 1: 3 Capas y 25 neuronas en la capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido: ",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_25.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_25.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_25.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_25.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_25.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_25.predict(X_test), y_test))))
print("-----")

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_25.predict(X_train, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_25.predict(X_test, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el grafico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolucion del error cuadratico medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relacion entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()])

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

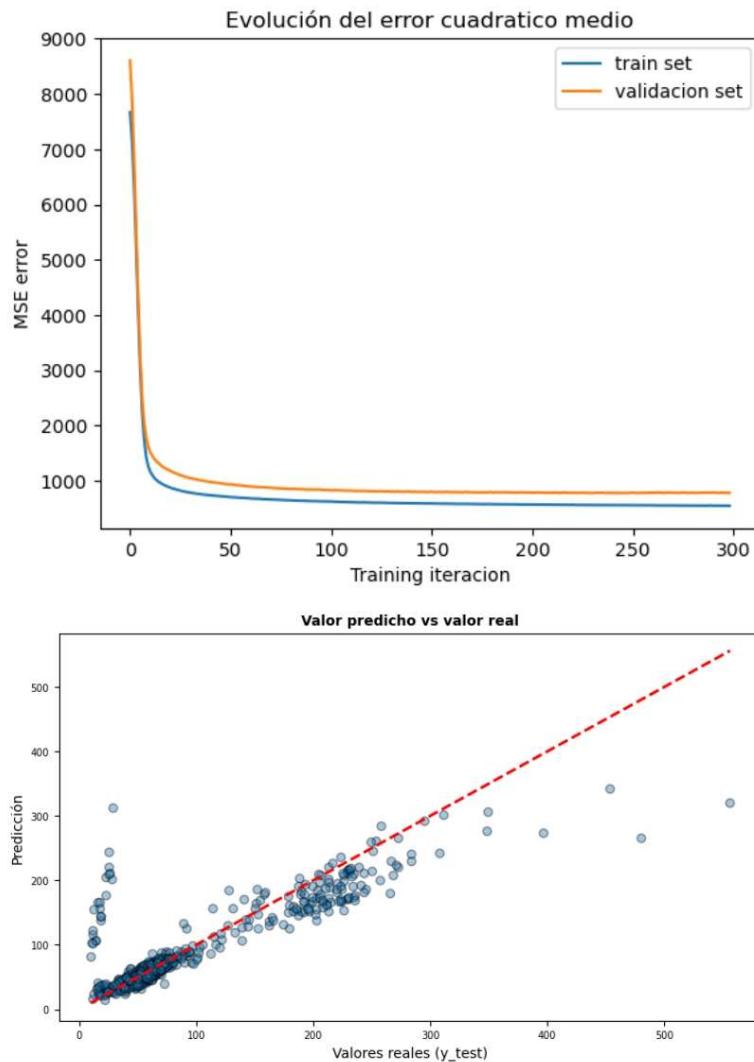
## EN EL MERCADO ESPAÑOL

```
[y_test.min(), y_test.max()],
'--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

Model: "sequential_2"
-----  
Layer (type)          Output Shape       Param #
-----  
dense_4 (Dense)      (None, 25)         675  
dense_5 (Dense)      (None, 1)          26
-----  
Total params: 701
Trainable params: 701
Non-trainable params: 0
-----  
El modelo se está entrenando.....  
Epoch 299: early stopping
El entrenamiento ha finalizado!
-----  
Datos del MODELO 1: 3 Capas y 25 neuronas en la capa oculta
El tiempo empleado para calcular el modelo ha sido: 77.4288
MAE en train: 9.55
MSE en train: 546.25
RMSE en train: 23.37
-----  
MAE en test: 11.51
MSE en test: 782.44
RMSE en test: 27.97
-----  
La precisión del modelo de RED NUERONAL con los datos de entrenamiento e
s de: 79.04
La precisión del modelo de RED NUERONAL con los datos de test es de: 76.
95
MAPE en test: 26.58 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



#### MODELO 4: 3 Capas y 50 neuronas en la capa oculta

```
In [16]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_50 = Sequential()

# 1º capa oculta
modelo_RNeu_50.add(Dense(50, input_dim=26, activation = 'relu'))

# Capa de salida
modelo_RNeu_50.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
```

```

modelo_RNeu_50.compile(loss='mse',
                       optimizer='adam',
                       metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_50.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=50,
                           verbose=1)
entrenamiento = modelo_RNeu_50.fit(X_train, y_train,
                                    validation_data = (X_test, y_test),
                                    epochs = 2000,
                                    batch_size = 30,
                                    verbose = False,
                                    callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 4: 3 Capas y 50 neuronas en la capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido: ",
      round(end - start,4))
print("-----")

# Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_50.predict(
print('MSE en train:', round(mean_squared_error(modelo_RNeu_50.predict(X_
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_50.p
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_50.predict(X_
print('MSE en test:', round(mean_squared_error(modelo_RNeu_50.predict(X_
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_50.p
print("-----")

# Comprobamos la precisión/rendimiento del modelo generado.
y_pred = modelo_RNeu_50.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de entrenamiento es: ", round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_50.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de test es: ", round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validación set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

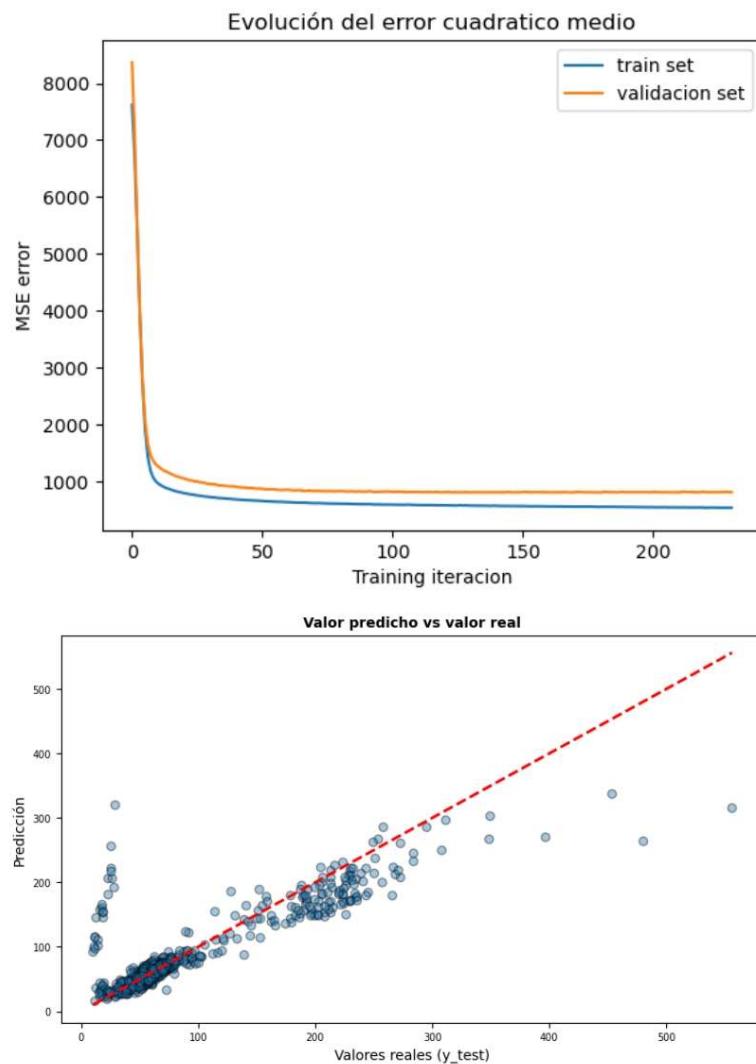
Model: "sequential_3"

Layer (type)                 Output Shape            Param #
dense_6 (Dense)              (None, 50)             1350
dense_7 (Dense)              (None, 1)              51
=====
Total params: 1,401
Trainable params: 1,401
Non-trainable params: 0

El modelo se está entrenando.....
Epoch 231: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 4: 3 Capas y 50 neuronas en la capa oculta
El tiempo empleado para calcular el modelo ha sido: 64.0895
-----
MAE en train: 9.48
MSE en train: 530.74
RMSE en train: 23.04
-----
MAE en test: 11.86
MSE en test: 808.22
RMSE en test: 28.43
-----
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 79.64
La precisión del modelo de RED NUERONAL con los datos de test es de: 76.19
MAPE en test: 27.32 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 5: 6 Capas y 5-10-20-40 neuronas en las capas ocultas**

```
In [17]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_5_10_20_40 = Sequential()

# 1º capa oculta
modelo_RNeu_5_10_20_40.add(Dense(5, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_5_10_20_40.add(Dense(10, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_5_10_20_40.add(Dense(20, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_5_10_20_40.add(Dense(40, activation = 'relu'))

# Capa de salida
modelo_RNeu_5_10_20_40.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_5_10_20_40.compile(loss='mse',
                                optimizer='adam',
                                metrics=['mse','mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_5_10_20_40.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor ='val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_5_10_20_40.fit(X_train, y_train,
                                             validation_data = (X_test, y_test),
                                             epochs = 2000,
                                             batch_size = 30,
                                             verbose = False,
                                             callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 5: 6 Capas y 5-10-20-40 neuronas en las capas ocultas')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_5_10_20_40.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_5_10_20_40.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_5_10_20_40.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_5_10_20_40.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_5_10_20_40.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_5_10_20_40.predict(X_test), y_test))))
print("-----")

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_5_10_20_40.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de entrenamiento es:",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_5_10_20_40.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de test es:",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

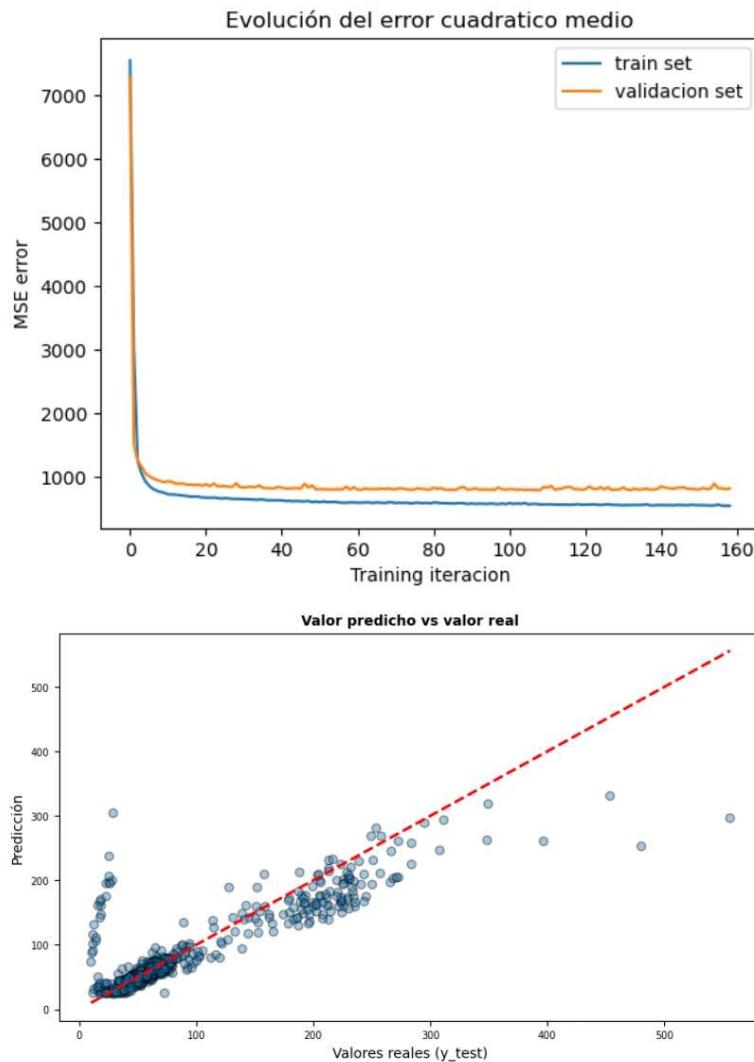
Model: "sequential_4"

Layer (type)          Output Shape       Param #
=====            =====
dense_8 (Dense)      (None, 5)           135
dense_9 (Dense)      (None, 10)          60
dense_10 (Dense)     (None, 20)          220
dense_11 (Dense)     (None, 40)          840
dense_12 (Dense)     (None, 1)           41
=====
Total params: 1,296
Trainable params: 1,296
Non-trainable params: 0

El modelo se está entrenando.....
Epoch 159: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 5: 6 Capas y 5-10-20-40 neuronas en las capas ocultas
El tiempo empleado para calcular el modelo ha sido: 57.281
MAE en train: 10.2
MSE en train: 540.43
RMSE en train: 23.25
-----
MAE en test: 12.22
MSE en test: 822.62
RMSE en test: 28.68
-----
La precisión del modelo de RED NEURONAL con los datos de entrenamiento es de: 79.26
La precisión del modelo de RED NEURONAL con los datos de test es de: 75.77
MAPE en test: 26.81 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 6: 6 Capas y 40-20-10-5 neuronas en las capas ocultas**

```
In [18]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_40_20_10_5 = Sequential()

# 1º capa oculta
modelo_RNeu_40_20_10_5.add(Dense(40, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_40_20_10_5.add(Dense(20, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_40_20_10_5.add(Dense(10, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_40_20_10_5.add(Dense(5, activation = 'relu'))

# Capa de salida
modelo_RNeu_40_20_10_5.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_40_20_10_5.compile(loss='mse',
                                optimizer='adam',
                                metrics=['mse','mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_40_20_10_5.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor ='val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_40_20_10_5.fit(X_train, y_train,
                                             validation_data = (X_test, y_test),
                                             epochs = 2000,
                                             batch_size = 30,
                                             verbose = False,
                                             callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 6: 6 Capas y 40-20-10-5 neuronas en las capas ocultas')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_40_20_10_5.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_40_20_10_5.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_40_20_10_5.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_40_20_10_5.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_40_20_10_5.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_40_20_10_5.predict(X_test), y_test))))
print("-----")

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_40_20_10_5.predict(X_train, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_40_20_10_5.predict(X_test, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el grafico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolucion del error cuadratico medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

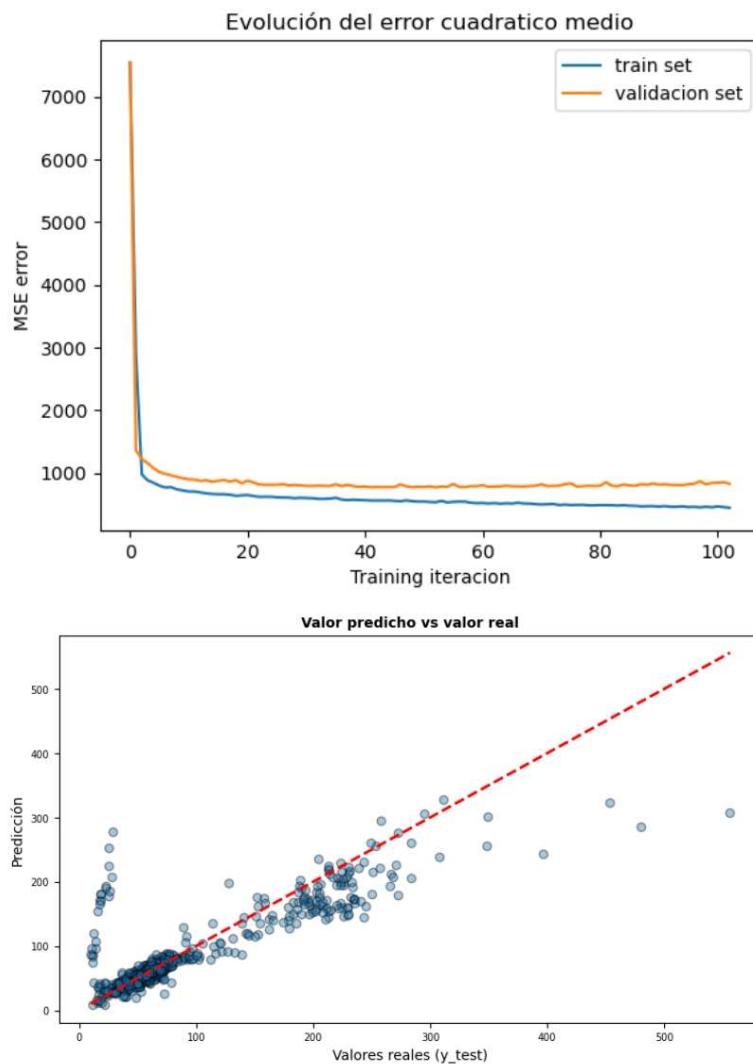
Model: "sequential_5"

Layer (type)          Output Shape       Param #
dense_13 (Dense)     (None, 40)           1080
dense_14 (Dense)     (None, 20)            820
dense_15 (Dense)     (None, 10)            210
dense_16 (Dense)     (None, 5)             55
dense_17 (Dense)     (None, 1)             6
=====
Total params: 2,171
Trainable params: 2,171
Non-trainable params: 0

El modelo se está entrenando.....
Epoch 103: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 6: 6 Capas y 40-20-10-5 neuronas en las capas ocultas
El tiempo empleado para calcular el modelo ha sido: 28.2037
MAE en train: 8.9
MSE en train: 429.97
RMSE en train: 20.74
-----
MAE en test: 12.11
MSE en test: 828.17
RMSE en test: 28.78
-----
La precisión del modelo de RED NEURONAL con los datos de entrenamiento es de: 83.5
La precisión del modelo de RED NEURONAL con los datos de test es de: 75.6
MAPE en test: 26.13 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 7: 6 Capas y 10x4 neuronas en cada capa oculta**

```
In [19]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_10x4 = Sequential()

# 1º capa oculta
modelo_RNeu_10x4.add(Dense(10, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_10x4.add(Dense(10, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_10x4.add(Dense(10, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_10x4.add(Dense(10, activation = 'relu'))

# Capa de salida
modelo_RNeu_10x4.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_10x4.compile(loss='mse',
                         optimizer='adam',
                         metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_10x4.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor ='val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_10x4.fit(X_train, y_train,
                                       validation_data = (X_test, y_test),
                                       epochs = 2000,
                                       batch_size = 30,
                                       verbose = False,
                                       callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 7: 6 Capas y 10x4 neuronas en cada capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_10x4.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_10x4.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_10x4.predict(X_train), y_train))))
print('-----')
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_10x4.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_10x4.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_10x4.predict(X_test), y_test))))
print('-----')

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_10x4.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_10x4.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='X_train')
plt.plot(entrenamiento.history['val_mse'], label='y_train')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteración')
plt.ylabel('MSE error')
plt.legend()

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

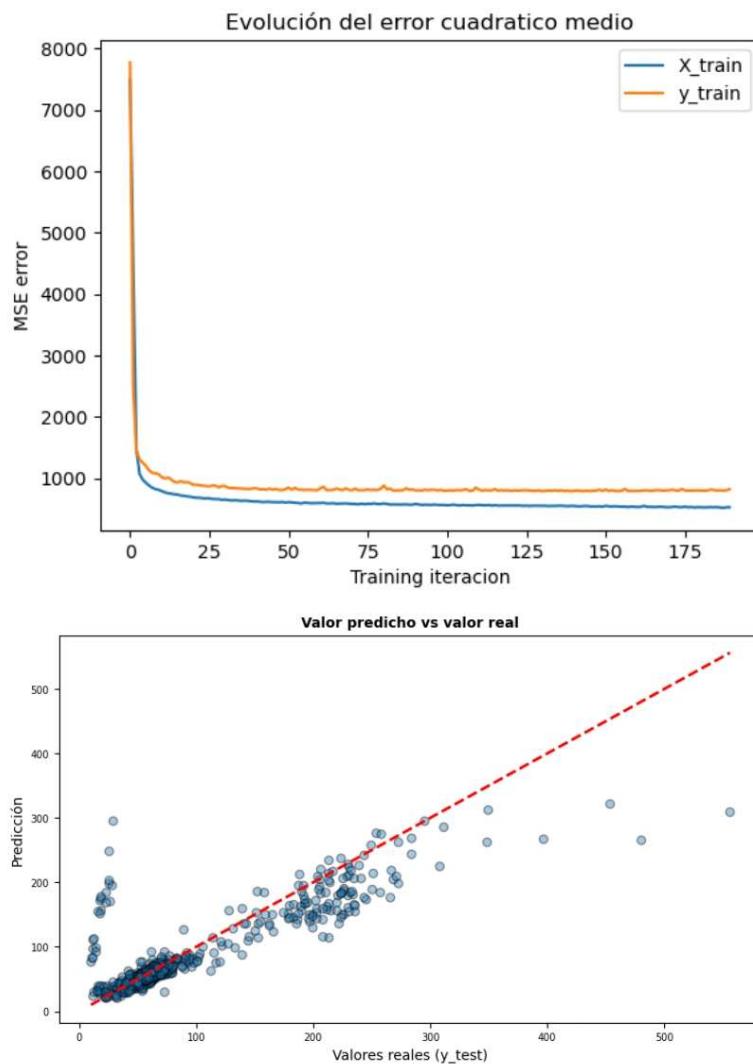
Model: "sequential_6"

Layer (type)          Output Shape       Param #
=====          =====
dense_18 (Dense)      (None, 10)        270
dense_19 (Dense)      (None, 10)        110
dense_20 (Dense)      (None, 10)        110
dense_21 (Dense)      (None, 10)        110
dense_22 (Dense)      (None, 1)         11
=====
Total params: 611
Trainable params: 611
Non-trainable params: 0

El modelo se está entrenando.....
Epoch 190: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 7: 6 Capas y 10x4 neuronas en cada capa oculta
El tiempo empleado para calcular el modelo ha sido: 64.4265
MAE en train: 10.19
MSE en train: 519.99
RMSE en train: 22.8
-----
MAE en test: 12.28
MSE en test: 821.17
RMSE en test: 28.66
-----
La precisión del modelo de RED NEURONAL con los datos de entrenamiento es de: 80.05
La precisión del modelo de RED NEURONAL con los datos de test es de: 75.81
MAPE en test: 26.37 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 8: 6 Capas y 20x4 neuronas en cada capa oculta**

```
In [20]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_20x4 = Sequential()

# 1º capa oculta
modelo_RNeu_20x4.add(Dense(20, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_20x4.add(Dense(20, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_20x4.add(Dense(20, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_20x4.add(Dense(20, activation = 'relu'))

# Capa de salida
modelo_RNeu_20x4.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_20x4.compile(loss='mse',
                         optimizer='adam',
                         metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_20x4.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_20x4.fit(X_train, y_train,
                                       validation_data = (X_test, y_test),
                                       epochs = 2000,
                                       batch_size = 30,
                                       verbose = False,
                                       callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 8: 6 Capas y 20x4 neuronas en cada capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_20x4.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_20x4.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x4.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_20x4.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_20x4.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x4.predict(X_test), y_test))))
print("-----")

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_20x4.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_20x4.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadratico medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()

```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

```
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

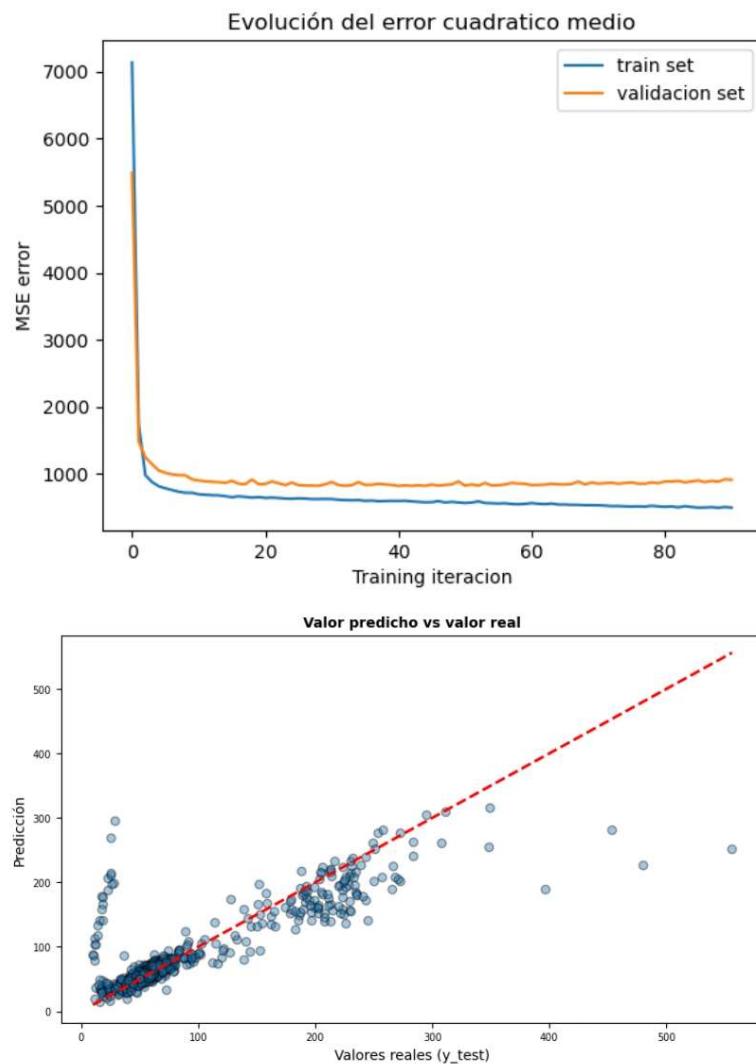
Model: "sequential_7"

Layer (type)          Output Shape       Param #
=====            =====
dense_23 (Dense)     (None, 20)           540
dense_24 (Dense)     (None, 20)           420
dense_25 (Dense)     (None, 20)           420
dense_26 (Dense)     (None, 20)           420
dense_27 (Dense)     (None, 1)            21
=====
Total params: 1,821
Trainable params: 1,821
Non-trainable params: 0

El modelo se está entrenando.....
Epoch 91: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 8: 6 Capas y 20x4 neuronas en cada capa oculta
El tiempo empleado para calcular el modelo ha sido: 29.4123
MAE en train: 9.65
MSE en train: 474.18
RMSE en train: 21.78
-----
MAE en test: 12.54
MSE en test: 913.99
RMSE en test: 30.23
-----
La precisión del modelo de RED NEURONAL con los datos de entrenamiento es de: 81.81
La precisión del modelo de RED NEURONAL con los datos de test es de: 73.08
MAPE en test: 27.98 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 9: 6 Capas y 40x4 neuronas en cada capa oculta**

```
In [21]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_40x4 = Sequential()

# 1º capa oculta
modelo_RNeu_40x4.add(Dense(40, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_40x4.add(Dense(40, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_40x4.add(Dense(40, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_40x4.add(Dense(40, activation = 'relu'))

# Capa de salida
modelo_RNeu_40x4.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_40x4.compile(loss='mse',
                         optimizer='adam',
                         metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_40x4.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor ='val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_40x4.fit(X_train, y_train,
                                       validation_data = (X_test, y_test),
                                       epochs = 2000,
                                       batch_size = 30,
                                       verbose = False,
                                       callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 9: 6 Capas y 40x4 neuronas en cada capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_40x4.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_40x4.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_40x4.predict(X_train), y_train))))
print('-----')
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_40x4.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_40x4.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_40x4.predict(X_test), y_test))))
print('-----')

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_40x4.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_40x4.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadratico medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()

```

```

plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

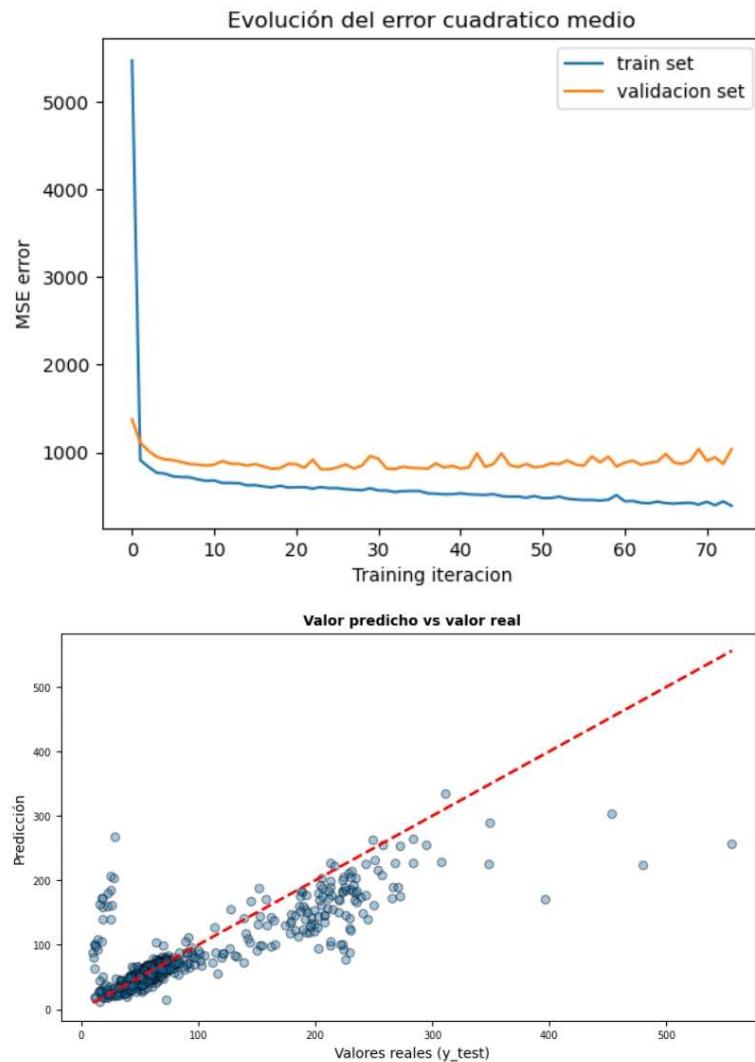
Model: "sequential_8"

Layer (type)          Output Shape       Param #
dense_28 (Dense)     (None, 40)           1080
dense_29 (Dense)     (None, 40)           1640
dense_30 (Dense)     (None, 40)           1640
dense_31 (Dense)     (None, 40)           1640
dense_32 (Dense)     (None, 1)            41
=====
Total params: 6,041
Trainable params: 6,041
Non-trainable params: 0

El modelo se está entrenando.....
Epoch 74: early stopping
El entrenamiento ha finalizado!
-----
Datos del MODELO 9: 6 Capas y 40x4 neuronas en cada capa oculta
El tiempo empleado para calcular el modelo ha sido: 26.2518
MAE en train: 9.86
MSE en train: 408.79
RMSE en train: 20.22
-----
MAE en test: 14.43
MSE en test: 1034.18
RMSE en test: 32.16
-----
La precisión del modelo de RED NEURONAL con los datos de entrenamiento es de: 84.32
La precisión del modelo de RED NEURONAL con los datos de test es de: 69.54
MAPE en test: 26.55 %

```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL



MODELO 10: 12 Capas y 25x10 neuronas en cada capa oculta

```
In [22]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_20x10 = Sequential()

# 1º capa oculta
modelo_RNeu_20x10.add(Dense(20, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 5º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 6º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 7º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 8º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 9º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))
# 10º capa oculta
modelo_RNeu_20x10.add(Dense(20, activation = 'relu'))

# Capa de salida
modelo_RNeu_20x10.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_20x10.compile(loss='mse',
                           optimizer='adam',
                           metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_20x10.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_20x10.fit(X_train, y_train,
                                       validation_data = (X_test, y_test),
                                       epochs = 2000,
                                       batch_size = 30,
                                       verbose = False,
                                       callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 10: 12 Capas y 25x10 neuronas en cada capa oculta')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_20x10.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_20x10.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x10.predict(X_train), y_train))))
print('-----')
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_20x10.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_20x10.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x10.predict(X_test), y_test))))
print('-----')

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_20x10.predict(X_train, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de entrenamiento es",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_20x10.predict(X_test, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de test es",
      round(r2_score(y_test, y_pred)*100, 2))

```

```
MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

Model: "sequential\_9"

Layer (type)	Output Shape	Param #
dense_33 (Dense)	(None, 20)	540
dense_34 (Dense)	(None, 20)	420
dense_35 (Dense)	(None, 20)	420
dense_36 (Dense)	(None, 20)	420
dense_37 (Dense)	(None, 20)	420
dense_38 (Dense)	(None, 20)	420
dense_39 (Dense)	(None, 20)	420
dense_40 (Dense)	(None, 20)	420
dense_41 (Dense)	(None, 20)	420
dense_42 (Dense)	(None, 20)	420
dense_43 (Dense)	(None, 1)	21

=====  
Total params: 4,341  
Trainable params: 4,341  
Non-trainable params: 0

-----  
El modelo se está entrenando.....  
Epoch 69: early stopping  
El entrenamiento ha finalizado!

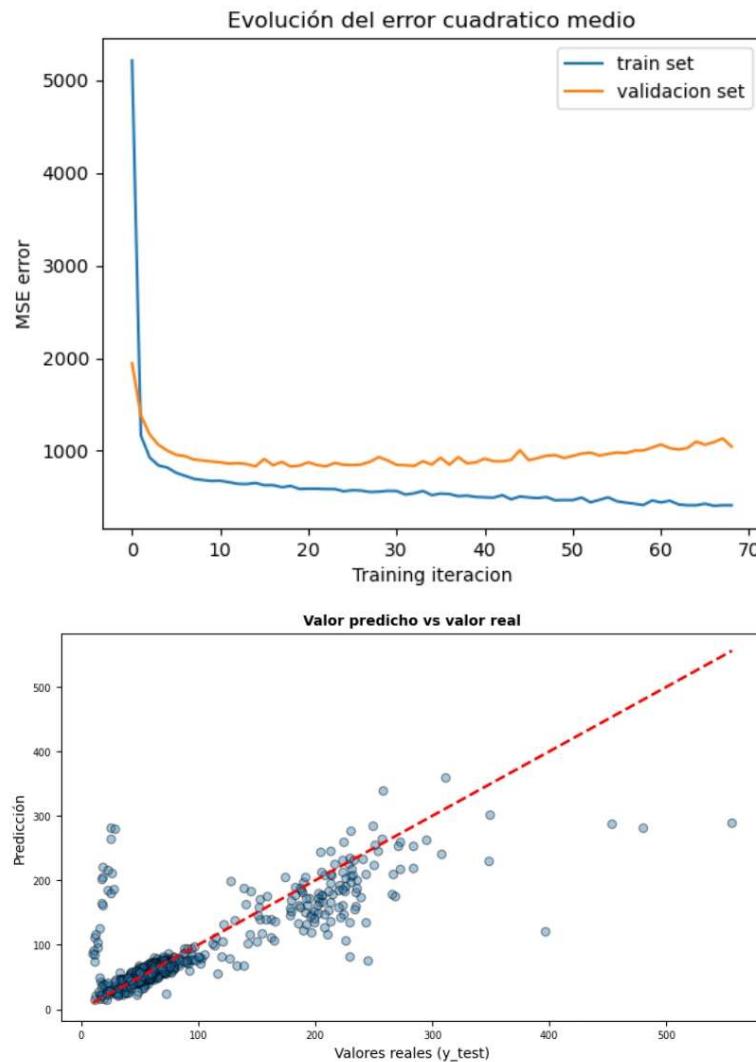
-----  
Datos del MODELO 10: 12 Capas y 25x10 neuronas en cada capa oculta  
El tiempo empleado para calcular el modelo ha sido: 25.5792  
MAE en train: 8.73  
MSE en train: 357.56  
RMSE en train: 18.91

-----  
MAE en test: 13.04  
MSE en test: 1047.02  
RMSE en test: 32.36

-----  
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 86.28  
La precisión del modelo de RED NUERONAL con los datos de test es de: 69.16  
MAPE en test: 27.31 %

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



**MODELO 11: 12 Capas y 25x5 + 50x5 neuronas en las capas ocultas**

```
In [23]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_25x5_50x5 = Sequential()

# 1º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(25, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(50, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(25, activation = 'relu'))
```

```

# 4º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(50, activation = 'relu'))
# 5º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(25, activation = 'relu'))
# 6º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(50, activation = 'relu'))
# 7º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(25, activation = 'relu'))
# 8º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(50, activation = 'relu'))
# 9º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(25, activation = 'relu'))
# 10º capa oculta
modelo_RNeu_25x5_50x5.add(Dense(50, activation = 'relu'))

# Capa de salida
modelo_RNeu_25x5_50x5.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_25x5_50x5.compile(loss='mse',
                                optimizer='adam',
                                metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_25x5_50x5.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_25x5_50x5.fit(X_train, y_train,
                                             validation_data = (X_test, y_test),
                                             epochs = 2000,
                                             batch_size = 30,
                                             verbose = False,
                                             callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 11: 12 Capas y 25x5 + 50x5 neuronas en las capas')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_25x5_50x5.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_25x5_50x5.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_25x5_50x5.predict(X_train), y_train))))
print('-----')
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_25x5_50x5.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_25x5_50x5.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_25x5_50x5.predict(X_test), y_test))))
print('-----')

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_25x5_50x5.predict(X_train, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de entrenamiento es:",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_25x5_50x5.predict(X_test, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de test es:",
      round(r2_score(y_test, y_pred)*100, 2))

```

```
round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

Model: "sequential\_10"

Layer (type)	Output Shape	Param #
dense_44 (Dense)	(None, 25)	675
dense_45 (Dense)	(None, 50)	1300
dense_46 (Dense)	(None, 25)	1275
dense_47 (Dense)	(None, 50)	1300
dense_48 (Dense)	(None, 25)	1275
dense_49 (Dense)	(None, 50)	1300
dense_50 (Dense)	(None, 25)	1275
dense_51 (Dense)	(None, 50)	1300
dense_52 (Dense)	(None, 25)	1275
dense_53 (Dense)	(None, 50)	1300
dense_54 (Dense)	(None, 1)	51

Total params: 12,326  
Trainable params: 12,326  
Non-trainable params: 0

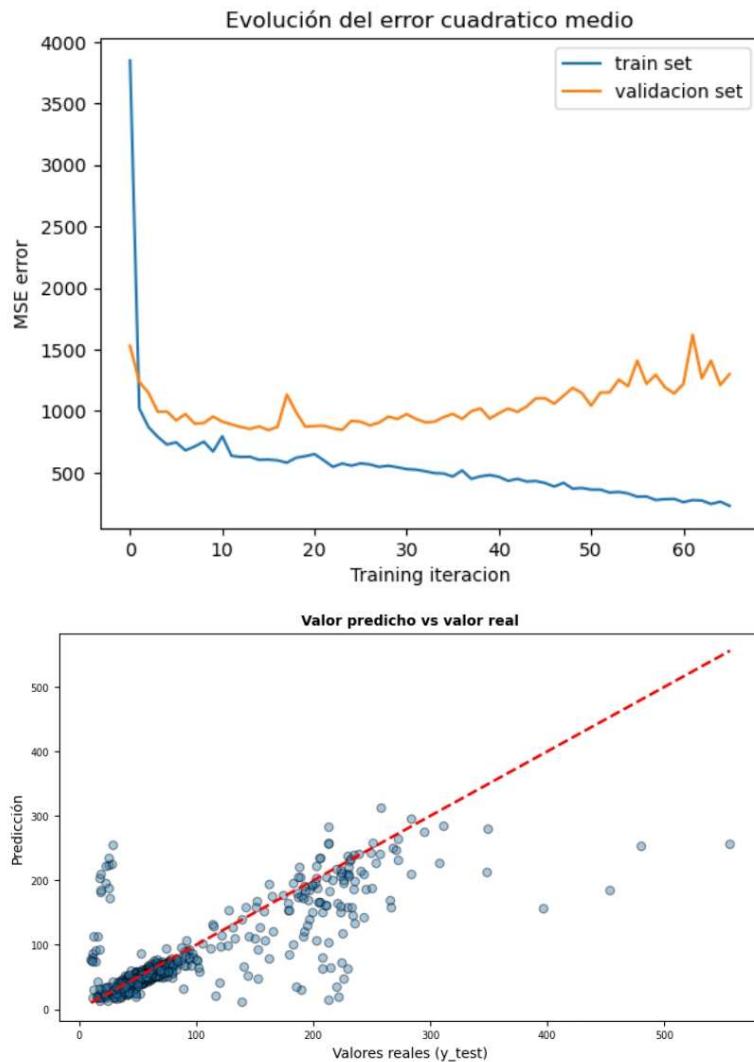
El modelo se está entrenando.....  
Epoch 66: early stopping  
El entrenamiento ha finalizado!

-----  
Datos del MODELO 11: 12 Capas y 25x5 + 50x5 neuronas en las capas ocultas  
El tiempo empleado para calcular el modelo ha sido: 26.6902  
MAE en train: 7.98  
MSE en train: 202.98  
RMSE en train: 14.25

-----  
MAE en test: 14.79  
MSE en test: 1299.94  
RMSE en test: 36.05

-----  
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 92.21  
La precisión del modelo de RED NUERONAL con los datos de test es de: 61.71  
MAPE en test: 27.2 %

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL



MODELO 12: 12 Capas y 10x10 neuronas en cada capa oculta

```
In [26]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_10x10 = Sequential()

# 1º capa oculta
modelo_RNeu_10x10.add(Dense(10, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 4º capa oculta
```

```

modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 5º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 6º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 7º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 8º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 9º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))
# 10º capa oculta
modelo_RNeu_10x10.add(Dense(10, activation = 'relu'))

# Capa de salida
modelo_RNeu_10x10.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_10x10.compile(loss='mse',
                           optimizer='adam',
                           metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_10x10.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_10x10.fit(X_train, y_train,
                                       validation_data = (X_test, y_test),
                                       epochs = 2000,
                                       batch_size = 30,
                                       verbose = False,
                                       callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 11: 12 Capas y 10x10 neuronas en las capas oculares')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido: ",
      round(end - start,4))
print("-----")

# Obtenemos las diferentes metricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_10x10.predict(X_train), y_train)))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_10x10.predict(X_train), y_train)))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_10x10.predict(X_train), y_train))))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_10x10.predict(X_test), y_test)))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_10x10.predict(X_test), y_test)))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_10x10.predict(X_test), y_test))))
print("-----")

# Comprobamos la precision/rendimiento del modelo generado.
y_pred = modelo_RNeu_10x10.predict(X_train, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de entrenamiento es: ",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_10x10.predict(X_test, verbose = 0)
print("La precision del modelo de RED NUERONAL con los datos de test es: ",
      round(r2_score(y_test, y_pred)*100, 2))

```

```
round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteracion')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight='bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---

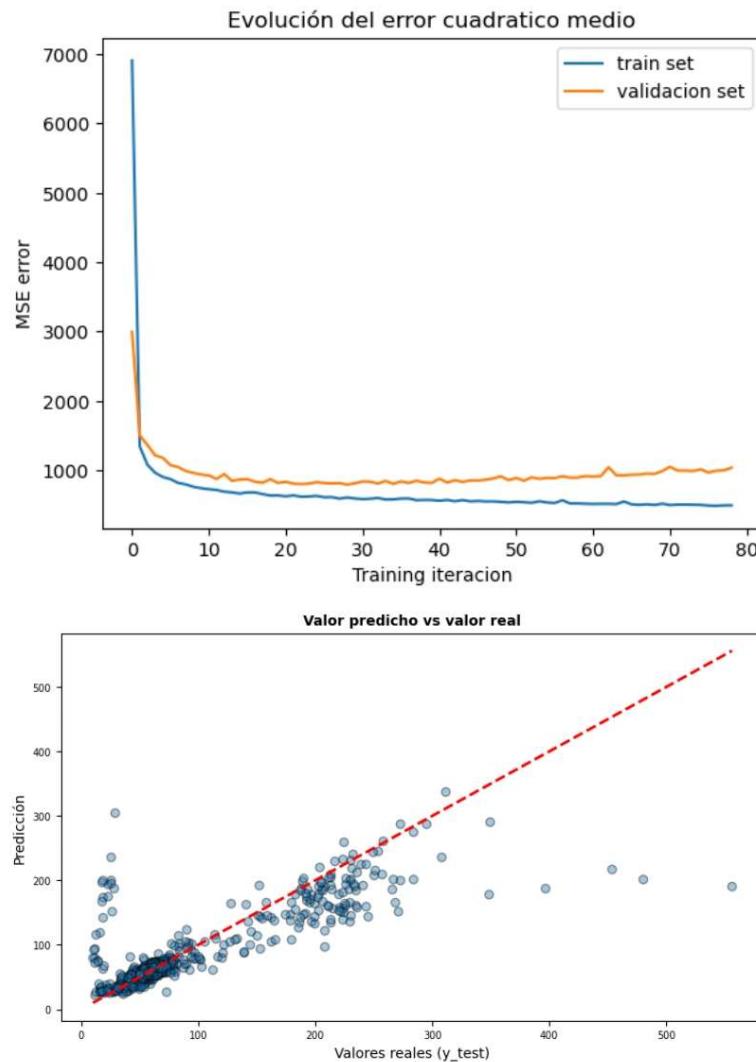
```
Model: "sequential_13"
-----

| Layer (type)      | Output Shape | Param # |
|-------------------|--------------|---------|
| dense_94 (Dense)  | (None, 10)   | 270     |
| dense_95 (Dense)  | (None, 10)   | 110     |
| dense_96 (Dense)  | (None, 10)   | 110     |
| dense_97 (Dense)  | (None, 10)   | 110     |
| dense_98 (Dense)  | (None, 10)   | 110     |
| dense_99 (Dense)  | (None, 10)   | 110     |
| dense_100 (Dense) | (None, 10)   | 110     |
| dense_101 (Dense) | (None, 10)   | 110     |
| dense_102 (Dense) | (None, 10)   | 110     |
| dense_103 (Dense) | (None, 10)   | 110     |
| dense_104 (Dense) | (None, 1)    | 11      |


-----  
Total params: 1,271  
Trainable params: 1,271  
Non-trainable params: 0
-----  
El modelo se está entrenando.....  
Epoch 79: early stopping  
El entrenamiento ha finalizado!
-----  
Datos del MODELO 11: 12 Capas y 10x10 neuronas en las capas ocultas  
El tiempo empleado para calcular el modelo ha sido: 30.1217
-----  
MAE en train: 9.87  
MSE en train: 469.89  
RMSE en train: 21.68
-----  
MAE en test: 13.14  
MSE en test: 1037.8  
RMSE en test: 32.21
-----  
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 81.97  
La precisión del modelo de RED NUERONAL con los datos de test es de: 69.43  
MAPE en test: 26.64 %
```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---



MODELO 13: 20 Capas y 20x9+30x9+10x9 neuronas en cada capa oculta

```
In [25]: # Calcularemos el tiempo que tarda la computadora para desarrollar el modelo
start = time.time()

# Creamos el modelo
modelo_RNeu_20x9_30x9_10x9 = Sequential()

# 1º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, input_dim=26, activation = 'relu'))
# 2º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 3º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
```

```

# 4º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 5º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 6º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 7º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 8º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 9º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 10º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 11º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 12º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 13º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 14º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 15º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 16º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 17º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 18º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 19º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 20º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 21º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 22º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 23º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 24º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))
# 25º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(20, activation = 'relu'))
# 26º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(30, activation = 'relu'))
# 27º capa oculta
modelo_RNeu_20x9_30x9_10x9.add(Dense(10, activation = 'relu'))

# Capa de salida
modelo_RNeu_20x9_30x9_10x9.add(Dense(1, activation = 'relu'))

# Compilacion del modelo
modelo_RNeu_20x9_30x9_10x9.compile(loss='mse',
                                      optimizer='adam',
                                      metrics=['mse', 'mae'])

# Obtenemos el resumen de la red neuronal creada
modelo_RNeu_20x9_30x9_10x9.summary()

# Ajuste del modelo
print("El modelo se está entrenando.....")

```

```

early_stop = EarlyStopping(monitor ='val_loss', mode = 'min', patience =
                           verbose = 1)
entrenamiento = modelo_RNeu_20x9_30x9_10x9.fit(X_train, y_train,
                                                validation_data = (X_test, y_test),
                                                epochs = 2000,
                                                batch_size = 30,
                                                verbose = False,
                                                callbacks=[early_stop])
print("El entrenamiento ha finalizado!")

print('-----')
print('Datos del MODELO 13: 20 Capas y 20x9_30x9_10x9 neuronas en cada capa')

# Medimos el tiempo que ha tardado el modelo
end = time.time()
print("El tiempo empleado para calcular el modelo ha sido:",
      round(end - start,4))

# Obtenemos las diferentes métricas del error de "train" y "test"
print('MAE en train:', round(mean_absolute_error(modelo_RNeu_20x9_30x9_10x9,X_train), 2))
print('MSE en train:', round(mean_squared_error(modelo_RNeu_20x9_30x9_10x9,X_train), 2))
print('RMSE en train:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x9_30x9_10x9,X_train)), 2))
print("-----")
print('MAE en test:', round(mean_absolute_error(modelo_RNeu_20x9_30x9_10x9,X_test), 2))
print('MSE en test:', round(mean_squared_error(modelo_RNeu_20x9_30x9_10x9,X_test), 2))
print('RMSE en test:', round(np.sqrt(mean_squared_error(modelo_RNeu_20x9_30x9_10x9,X_test)), 2))
print("-----")

# Comprobamos la precisión/rendimiento del modelo generado.
y_pred = modelo_RNeu_20x9_30x9_10x9.predict(X_train, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de entrenamiento es:",
      round(r2_score(y_train, y_pred)*100, 2))
y_pred = modelo_RNeu_20x9_30x9_10x9.predict(X_test, verbose = 0)
print("La precisión del modelo de RED NEURONAL con los datos de test es:",
      round(r2_score(y_test, y_pred)*100, 2))

MAPE_1 = (y_test - y_pred)/y_test
MAPE_1 = np.sqrt(MAPE_1**2)
print('MAPE en test:', round(np.mean(MAPE_1)*100, 2), "%")

# Dibujamos el gráfico de aprendizaje
plt.plot(entrenamiento.history['mse'], label='train set')
plt.plot(entrenamiento.history['val_mse'], label='validacion set')
plt.title('Evolución del error cuadrático medio')
plt.xlabel('Training iteración')
plt.ylabel('MSE error')
plt.legend()
plt.show()

# Graficamos la relación entre los valores reales de "test" y los predichos
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(9, 5))
axes.scatter(y_test, y_pred, edgecolors = (0, 0, 0), alpha = 0.4)
axes.plot([y_test.min(), y_test.max()],
          [y_test.min(), y_test.max()],
          '--', lw=2, color = 'red')
axes.set_title('Valor predicho vs valor real', fontsize = 10, fontweight = 'bold')
axes.set_xlabel('Valores reales (y_test)')
axes.set_ylabel('Predicción')
axes.tick_params(labelsize = 7)

```

MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD  
EN EL MERCADO ESPAÑOL

---

Model: "sequential\_12"

Layer (type)	Output Shape	Param #
<hr/>		
dense_66 (Dense)	(None, 20)	540
dense_67 (Dense)	(None, 30)	630
dense_68 (Dense)	(None, 10)	310
dense_69 (Dense)	(None, 20)	220
dense_70 (Dense)	(None, 30)	630
dense_71 (Dense)	(None, 10)	310
dense_72 (Dense)	(None, 20)	220
dense_73 (Dense)	(None, 30)	630
dense_74 (Dense)	(None, 10)	310
dense_75 (Dense)	(None, 20)	220
dense_76 (Dense)	(None, 30)	630
dense_77 (Dense)	(None, 10)	310
dense_78 (Dense)	(None, 20)	220
dense_79 (Dense)	(None, 30)	630
dense_80 (Dense)	(None, 10)	310
dense_81 (Dense)	(None, 20)	220
dense_82 (Dense)	(None, 30)	630
dense_83 (Dense)	(None, 10)	310
dense_84 (Dense)	(None, 20)	220
dense_85 (Dense)	(None, 30)	630
dense_86 (Dense)	(None, 10)	310
dense_87 (Dense)	(None, 20)	220
dense_88 (Dense)	(None, 30)	630
dense_89 (Dense)	(None, 10)	310
dense_90 (Dense)	(None, 20)	220
dense_91 (Dense)	(None, 30)	630
dense_92 (Dense)	(None, 10)	310
dense_93 (Dense)	(None, 1)	11
<hr/>		
Total params: 10,771		

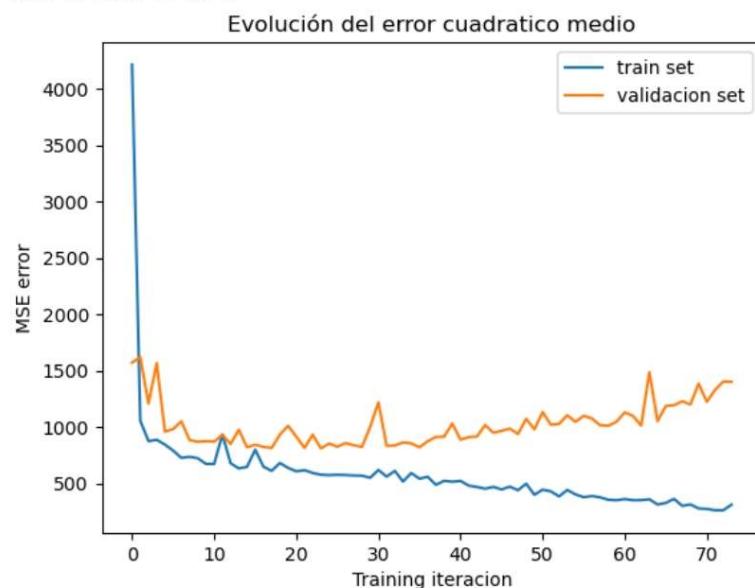
```
Trainable params: 10,771
Non-trainable params: 0

-----
El modelo se está entrenando.....
Epoch 74: early stopping
El entrenamiento ha finalizado!

-----
Datos del MODELO 13: 20 Capas y 20x9_30x9_10x9 neuronas en cada capa oculta
El tiempo empleado para calcular el modelo ha sido: 45.244
MAE en train: 8.2
MSE en train: 265.61
RMSE en train: 16.3

-----
MAE en test: 14.76
MSE en test: 1402.08
RMSE en test: 37.44

-----
La precisión del modelo de RED NUERONAL con los datos de entrenamiento es de: 89.81
La precisión del modelo de RED NUERONAL con los datos de test es de: 58.7
MAPE en test: 29.22 %
```



# MODELOS PARA LA PREDICCIÓN DEL PRECIO DE LA ELECTRICIDAD

## EN EL MERCADO ESPAÑOL

