0101_Regression_Boston

November 6, 2022

[1]:
```
%load_ext watermark
%watermark
```

```
Last updated: 2022-11-06T20:07:01.916677-05:00

Python implementation: CPython
Python version       : 3.9.13
IPython version      : 8.6.0

Compiler    : MSC v.1929 64 bit (AMD64)
OS          : Windows
Release     : 10
Machine     : AMD64
Processor   : Intel64 Family 6 Model 158 Stepping 10, GenuineIntel
CPU cores   : 12
Architecture: 64bit
```

[2]:
```
import pandas as pd

import numpy as np

import seaborn as sns

from ipywidgets import interact

from IPython.display import display

from scipy import stats

import pickle

from datetime import datetime

import matplotlib.pyplot as plt

import os

```

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.model_selection import learning_curve, train_test_split,␣
↪GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, PolynomialFeatures,␣
↪MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, TransformedTargetRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet,␣
↪BayesianRidge
from sklearn.ensemble import BaggingRegressor, AdaBoostRegressor,␣
↪GradientBoostingRegressor, RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error

import tensorflow as tf
from tensorflow.python.client import device_lib
from tensorflow.python.platform import build_info as build
from tensorflow.keras import backend
from tensorflow.keras.optimizers import RMSprop

from keras.models import Sequential, load_model
from keras.layers import Dense, Input, Dropout
from keras.callbacks import TensorBoard
from keras.constraints import maxnorm
from keras.wrappers.scikit_learn import KerasRegressor

import folium
from folium.plugins import MarkerCluster
from folium.plugins import MousePosition

sns.set(font_scale=0.7)
```

1 Resume

https://www.kaggle.com/datasets/vikrishnan/boston-house-prices

2 Data Collection

[3]:
```python
df = pd.read_csv('_resources/boston_dataset.csv')

df.rename(columns={'TOWN': 'CIUDAD',
                   'LON': 'LON',
                   'LAT': 'LAT',
                   'CRIM': 'INDICE_CRIMEN',
                   'ZN': 'PCT_ZONA_RESIDENCIAL',
                   'INDUS': 'PCT_ZONA_INDUSTRIAL',
                   'CHAS': 'RIO_CHARLES',
                   'NOX': 'OXIDO_NITROSO_PPM',
                   'RM': 'N_HABITACIONES_MEDIO',
                   'AGE': 'PCT_CASAS_40S',
                   'DIS': 'DIS',
                   'DIS_EMPLEO': 'DISTANCIA_CENTRO_EMPLEO',
                   'RAD': 'DIS_AUTOPISTAS',
                   'TAX': 'CARGA_FISCAL',
                   'PTRATIO': 'RATIO_PROFESORES',
                   'B': 'PCT_NEGRA',
                   'MEDV': 'VALOR_MEDIANO',
                   'LSTAT': 'PCT_CLASE_BAJA'}, inplace=True)

df.head()
```

[3]:
```
       CIUDAD     LON      LAT  VALOR_MEDIANO  INDICE_CRIMEN  \
0      Nahant -70.955  42.2550           24.0        0.00632
1  Swampscott -70.950  42.2875           21.6        0.02731
2  Swampscott -70.936  42.2830           34.7        0.02729
3  Marblehead -70.928  42.2930           33.4        0.03237
4  Marblehead -70.922  42.2980           36.2        0.06905

   PCT_ZONA_RESIDENCIAL  PCT_ZONA_INDUSTRIAL  RIO_CHARLES  OXIDO_NITROSO_PPM  \
0                  18.0                 2.31            0              0.538
1                   0.0                 7.07            0              0.469
2                   0.0                 7.07            0              0.469
3                   0.0                 2.18            0              0.458
4                   0.0                 2.18            0              0.458

   N_HABITACIONES_MEDIO  PCT_CASAS_40S     DIS  DIS_AUTOPISTAS  CARGA_FISCAL  \
0                 6.575           65.2  4.0900               1           296
1                 6.421           78.9  4.9671               2           242
2                 7.185           61.1  4.9671               2           242
3                 6.998           45.8  6.0622               3           222
4                 7.147           54.2  6.0622               3           222

   RATIO_PROFESORES  PCT_NEGRA  PCT_CLASE_BAJA
```

```
0                 15.3     396.90              4.98
1                 17.8     396.90              9.14
2                 17.8     392.83              4.03
3                 18.7     394.63              2.94
4                 18.7     396.90              5.33
```

3 Data Wrangling

Verificar los tipos de datos.

[4]:
```python
df.dtypes
```

[4]:
```
CIUDAD                    object
LON                      float64
LAT                      float64
VALOR_MEDIANO            float64
INDICE_CRIMEN            float64
PCT_ZONA_RESIDENCIAL     float64
PCT_ZONA_INDUSTRIAL      float64
RIO_CHARLES                int64
OXIDO_NITROSO_PPM        float64
N_HABITACIONES_MEDIO     float64
PCT_CASAS_40S            float64
DIS                      float64
DIS_AUTOPISTAS             int64
CARGA_FISCAL               int64
RATIO_PROFESORES         float64
PCT_NEGRA                float64
PCT_CLASE_BAJA           float64
dtype: object
```

Combinar los diferentes dataset cargados.

[5]:
```python
#
```

4 Data Visualization

Para una mejor observación, se genera una columna categórica a partir de la variable objetivo.

[6]:
```python
target_column = 'VALOR_MEDIANO'

type_target_column = f'TIPO_{target_column}'
df[type_target_column] = pd.qcut(x=df[target_column], q=3,
                                 labels=['bajo', 'medio', 'alto'])
df.head()
```

[6]:

```
       CIUDAD     LON      LAT  VALOR_MEDIANO  INDICE_CRIMEN  \
0      Nahant -70.955  42.2550           24.0        0.00632
1  Swampscott -70.950  42.2875           21.6        0.02731
2  Swampscott -70.936  42.2830           34.7        0.02729
3  Marblehead -70.928  42.2930           33.4        0.03237
4  Marblehead -70.922  42.2980           36.2        0.06905

   PCT_ZONA_RESIDENCIAL  PCT_ZONA_INDUSTRIAL  RIO_CHARLES  OXIDO_NITROSO_PPM  \
0                  18.0                 2.31            0              0.538
1                   0.0                 7.07            0              0.469
2                   0.0                 7.07            0              0.469
3                   0.0                 2.18            0              0.458
4                   0.0                 2.18            0              0.458

   N_HABITACIONES_MEDIO  PCT_CASAS_40S     DIS  DIS_AUTOPISTAS  CARGA_FISCAL  \
0                 6.575           65.2  4.0900               1           296
1                 6.421           78.9  4.9671               2           242
2                 7.185           61.1  4.9671               2           242
3                 6.998           45.8  6.0622               3           222
4                 7.147           54.2  6.0622               3           222

   RATIO_PROFESORES  PCT_NEGRA  PCT_CLASE_BAJA TIPO_VALOR_MEDIANO
0              15.3     396.90            4.98               alto
1              17.8     396.90            9.14              medio
2              17.8     392.83            4.03               alto
3              18.7     394.63            2.94               alto
4              18.7     396.90            5.33               alto
```

[7]:

```python
def update_datatypes_columns():
    numeric_features = df.select_dtypes(include=np.number).columns.to_list()
    continuous_features = df.select_dtypes(
        include=np.float64).columns.to_list()
    discrete_features = df.select_dtypes(include=np.int64).columns.to_list()
    categorical_features = df.select_dtypes(
        include='category').columns.to_list()
    object_features = df.select_dtypes(include='object').columns.to_list()

    display(numeric_features, continuous_features, discrete_features,
            categorical_features, object_features)

    return numeric_features, continuous_features, discrete_features,
 ↪categorical_features, object_features


numeric_features, continuous_features, discrete_features, categorical_features,
 ↪object_features = update_datatypes_columns()
```

```
['LON',
 'LAT',
 'VALOR_MEDIANO',
 'INDICE_CRIMEN',
 'PCT_ZONA_RESIDENCIAL',
 'PCT_ZONA_INDUSTRIAL',
 'RIO_CHARLES',
 'OXIDO_NITROSO_PPM',
 'N_HABITACIONES_MEDIO',
 'PCT_CASAS_40S',
 'DIS',
 'DIS_AUTOPISTAS',
 'CARGA_FISCAL',
 'RATIO_PROFESORES',
 'PCT_NEGRA',
 'PCT_CLASE_BAJA']
```

```
['LON',
 'LAT',
 'VALOR_MEDIANO',
 'INDICE_CRIMEN',
 'PCT_ZONA_RESIDENCIAL',
 'PCT_ZONA_INDUSTRIAL',
 'OXIDO_NITROSO_PPM',
 'N_HABITACIONES_MEDIO',
 'PCT_CASAS_40S',
 'DIS',
 'RATIO_PROFESORES',
 'PCT_NEGRA',
 'PCT_CLASE_BAJA']
```

```
['RIO_CHARLES', 'DIS_AUTOPISTAS', 'CARGA_FISCAL']
```

```
['TIPO_VALOR_MEDIANO']
```

```
['CIUDAD']
```

4.1 Statistical Summary

[8]:
```
df.describe()
```

[8]:

	LON	LAT	VALOR_MEDIANO	INDICE_CRIMEN	\
count	506.000000	506.000000	506.000000	506.000000	
mean	-71.056389	42.216440	22.528854	3.613524	
std	0.075405	0.061777	9.182176	8.601545	
min	-71.289500	42.030000	5.000000	0.006320	
25%	-71.093225	42.180775	17.025000	0.082045	
50%	-71.052900	42.218100	21.200000	0.256510	
75%	-71.019625	42.252250	25.000000	3.677083	

```
max    -70.810000   42.381000      50.000000      88.976200

       PCT_ZONA_RESIDENCIAL  PCT_ZONA_INDUSTRIAL  RIO_CHARLES  \
count            506.000000           506.000000   506.000000
mean              11.363636            11.136779     0.069170
std               23.322453             6.860353     0.253994
min                0.000000             0.460000     0.000000
25%                0.000000             5.190000     0.000000
50%                0.000000             9.690000     0.000000
75%               12.500000            18.100000     0.000000
max              100.000000            27.740000     1.000000

       OXIDO_NITROSO_PPM  N_HABITACIONES_MEDIO  PCT_CASAS_40S         DIS  \
count         506.000000            506.000000     506.000000  506.000000
mean            0.554695              6.284634      68.574901    3.795043
std             0.115878              0.702617      28.148861    2.105710
min             0.385000              3.561000       2.900000    1.129600
25%             0.449000              5.885500      45.025000    2.100175
50%             0.538000              6.208500      77.500000    3.207450
75%             0.624000              6.623500      94.075000    5.188425
max             0.871000              8.780000     100.000000   12.126500

       DIS_AUTOPISTAS  CARGA_FISCAL  RATIO_PROFESORES   PCT_NEGRA  \
count      506.000000    506.000000        506.000000  506.000000
mean         9.549407    408.237154         18.455534  356.674032
std          8.707259    168.537116          2.164946   91.294864
min          1.000000    187.000000         12.600000    0.320000
25%          4.000000    279.000000         17.400000  375.377500
50%          5.000000    330.000000         19.050000  391.440000
75%         24.000000    666.000000         20.200000  396.225000
max         24.000000    711.000000         22.000000  396.900000

       PCT_CLASE_BAJA
count      506.000000
mean        12.653063
std          7.141062
min          1.730000
25%          6.950000
50%         11.360000
75%         16.955000
max         37.970000
```

4.2 Correlation

4.2.1 Dispersion Diagram

[9]:
```
@interact(x=numeric_features, y=np.roll(numeric_features, -1), hue=True,␣
↪fit_reg=True)
def _(x, y, hue, fit_reg):
    if x == y:
        sns.displot(data=df, x=x)
    else:
        if hue:
            sns.lmplot(data=df, x=x, y=y, hue=type_target_column,
                       fit_reg=fit_reg, height=6)
        else:
            sns.lmplot(data=df, x=x, y=y, fit_reg=fit_reg, height=6)

    plt.show()
```

```
interactive(children=(Dropdown(description='x', options=('LON', 'LAT',␣
↪'VALOR_MEDIANO', 'INDICE_CRIMEN', 'PCT_…
```

[10]:
```
@interact(hue=True)
def _(hue):
    if hue:
        sns.pairplot(df, hue=type_target_column, diag_kind='hist')
    else:
        sns.pairplot(df, diag_kind='hist')

    plt.show()
```

```
interactive(children=(Checkbox(value=True, description='hue'), Output()),␣
↪_dom_classes=('widget-interact',))
```

4.2.2 Pearson Correlation

[11]:
```
@interact(calc=['Pearson', 'p-valor'])
def _(calc):
    def histogram_intersection(a, b):
        pearson_corr, p_value = stats.pearsonr(a, b)

        if calc == 'Pearson':
            # Strong correlation if pearson_corr close to 1 or -1.
            return pearson_corr
        elif calc == 'p-valor':
            # Strong correlation if p-value < 0.05.
            return 1 if p_value < 0.05 else 0

    matrix = df.corr(method=histogram_intersection, numeric_only=True)
```

```
    plt.rcParams['figure.figsize'] = (8, 8)

    if calc == 'Pearson':
        chart = sns.heatmap(matrix, annot=True, square=True, center=0)
    else:
        print('Strong correlation (1) if p-value < 0.05 else 0.')
        print('1: (reject the null hypothesis that the two variables are␣
↪independent)')
        print('0: (accept the null hypothesis that the two variables are␣
↪independent)')
        chart = sns.heatmap(matrix, annot=True, square=True, center=0)

    chart.set_xticklabels(chart.get_xticklabels(),
                          rotation=45, horizontalalignment='right')
    plt.show()
```

```
interactive(children=(Dropdown(description='calc', options=('Pearson',␣
↪'p-valor'), value='Pearson'), Output())…
```

4.3 Numerical distributions

4.3.1 Histogram

[12]:
```
@interact(col=numeric_features, density=False, cumulative=False)
def _(col, density, cumulative):
    plt.rcParams['figure.figsize'] = (5, 5)

    k = int(np.ceil(1 + np.log2(df.count()[col])))

    df[col].plot.hist(bins=k, density=density, cumulative=cumulative)
    # bins: specifica cuantos grupos queremos en el histograma, los rangos se␣
↪calculan más pequeños
    plt.xlabel(col)
    plt.ylabel('FRECUENCIA ' + ('RELATIVA' if density else 'ABSOLUTA') +
               (' ACUMULADA' if cumulative else ''))
    plt.title('HISTOGRAMA')

    plt.show()

    sesgo = stats.skew(df[col])
    kurtosis = stats.kurtosis(df[col])

    return f'Sesgo={sesgo} --- Kurtosis={kurtosis}'
```

```
interactive(children=(Dropdown(description='col', options=('LON', 'LAT',␣
↪'VALOR_MEDIANO', 'INDICE_CRIMEN', 'PC…
```

4.3.2 Probability Mass Function (PMF)

[13]:
```python
@interact(column=discrete_features)
def _(column):
    if len(discrete_features) == 0:
        return

    plt.rcParams['figure.figsize'] = (5, 5)

    probabilities = df[column].value_counts(normalize=True)

    probabilities_df = pd.DataFrame(
        {'INDEX': probabilities.index, 'VALUE': probabilities.values})
    probabilities_df = probabilities_df.sort_values(by=['INDEX'])

    plt.plot(probabilities_df['INDEX'], probabilities_df['VALUE'], '--')
    plt.vlines(probabilities_df['INDEX'], 0,
               probabilities_df['VALUE'], colors='b', lw=5, alpha=0.5)
    plt.title('Función de Masa de Probabilidad')
    plt.ylabel('probabilidad')
    plt.xlabel('valores')
    plt.show()
```

```
interactive(children=(Dropdown(description='column', options=('RIO_CHARLES',␣
↪'DIS_AUTOPISTAS', 'CARGA_FISCAL')…
```

4.3.3 Probability Density Function (PDF)

[14]:
```python
@interact(column=continuous_features)
def _(column):
    plt.rcParams['figure.figsize'] = (5, 5)

    data = df[column]

    k = int(np.ceil(1 + np.log2(df.count()[column])))
    plt.hist(data, bins=k, density=True)

    loc = data.mean()
    scale = data.std()
    pdf = stats.norm.pdf(data, loc=loc, scale=scale)
    sns.lineplot(x=data, y=pdf)

    plt.show()
```

```
interactive(children=(Dropdown(description='column', options=('LON', 'LAT',␣
↪'VALOR_MEDIANO', 'INDICE_CRIMEN', …
```

4.3.4 Kernel Density Estimate (KDE)

[15]:
```
@interact(col=continuous_features)
def _(col):
    sns.displot(data=df, x=col, kde=True)
    plt.show()
```

```
interactive(children=(Dropdown(description='col', options=('LON', 'LAT',␣
↪'VALOR_MEDIANO', 'INDICE_CRIMEN', 'PC…
```

4.3.5 Cumulative Distribution Function (CDF)

[16]:
```
@interact(column=continuous_features)
def _(column):
    plt.rcParams['figure.figsize'] = (5, 5)

    data = df[column]

    k = int(np.ceil(1 + np.log2(df.count()[column])))
    plt.hist(data, bins=k, density=True, cumulative=True)

    loc = data.mean()
    scale = data.std()
    cdf = stats.norm.cdf(data, loc=loc, scale=scale)
    sns.lineplot(x=data, y=cdf)

    plt.show()
```

```
interactive(children=(Dropdown(description='column', options=('LON', 'LAT',␣
↪'VALOR_MEDIANO', 'INDICE_CRIMEN', …
```

4.4 Pie Chart

[17]:
```
@interact(x=categorical_features+object_features)
def _(x):
    if x == None:
        print('There are no discrete or categorical variables with a null value.
↪')
        return

    plt.rcParams['figure.figsize'] = (5, 5)

    labels = df[x].value_counts().index.values
    sizes = df[x].value_counts().values
    explode = np.full(df[x].value_counts().count(), 0.1)

    _, ax1 = plt.subplots()
    ax1.pie(sizes, explode=explode, labels=labels,
```

```
            autopct='%1.1f%%', shadow=True, startangle=90)
    # Equal aspect ratio ensures that pie is drawn as a circle.
    ax1.axis('equal')

    plt.show()
```

```
interactive(children=(Dropdown(description='x', options=('TIPO_VALOR_MEDIANO',␣
↪'CIUDAD'), value='TIPO_VALOR_ME…
```

4.5 Box - Violin Plot

[18]:
```
@interact(orient_h=True, violin=False)
def _(orient_h, violin):
    plt.rcParams['figure.figsize'] = (10, 6)

    if violin:
        sns.violinplot(data=df, orient=('h' if orient_h else 'v'))
    else:
        sns.boxplot(data=df, orient=('h' if orient_h else 'v'))

    plt.show()
```

```
interactive(children=(Checkbox(value=True, description='orient_h'),␣
↪Checkbox(value=False, description='violin'…
```

[19]:
```
@interact(x=numeric_features, violin=False)
def _(x, violin):
    plt.rcParams['figure.figsize'] = (8, 4)

    if violin:
        sns.violinplot(x=x, data=df)
    else:
        sns.boxplot(x=x, data=df)

    plt.show()
```

```
interactive(children=(Dropdown(description='x', options=('LON', 'LAT',␣
↪'VALOR_MEDIANO', 'INDICE_CRIMEN', 'PCT_…
```

Se agrupan por: - Quintiles si la variable es continua. - Valores originales si la variable es discreta con menos de 10 únicos valores, caso contrario se agrupa por deciles. - Valores originales si la variable es categórica.

[20]:
```
@interact(x=numeric_features+categorical_features,
          y=np.roll(numeric_features, -1),
          violin=False)
def _(x, y, violin):
    # The list is divided into 5 quintiles if the variable is continuous.
    if df[x].dtype == np.float64:
```

```
        quintiles = pd.qcut(df[x], 5, duplicates='drop')
        new_df = df[y].to_frame().join(quintiles)
    # If the variable is discrete and with less than 10 different values, it is␣
 ↪left as is, otherwise it is grouped.
    elif df[x].dtype == np.int64:
        if df[x].unique().size > 10:
            quintiles = pd.qcut(df[x], 10, duplicates='drop')
            new_df = df[y].to_frame().join(quintiles)
        else:
            new_df = df[y].to_frame().join(df[x])
    # If the variable is categorical, it is left as is.
    elif df[x].dtype == 'category':
        new_df = df[y].to_frame().join(df[x])

    plt.rcParams['figure.figsize'] = (10, 6)

    if violin:
        sns.violinplot(x=x, y=y, data=new_df)
    else:
        sns.boxplot(x=x, y=y, data=new_df)

    sns.despine(offset=10, trim=True)

    plt.show()
```

```
interactive(children=(Dropdown(description='x', options=('LON', 'LAT',␣
 ↪'VALOR_MEDIANO', 'INDICE_CRIMEN', 'PCT_…
```

4.6 Categorical Comparisons

4.6.1 Numeric to Categorical Conversion

We have 2 options to group the values. - cut: The space between the groups are equal and the frequencies of each group are different. - qcut: The space between the groups are different and the frequencies of each group are equal.

To make containers with very different data less likely we use qcut.

[21]:
```
types = ['very low', 'low', 'moderate', 'high', 'very high']

# Columns created are added to delete later.
types_column = {'TYPE_INDICE_CRIMEN': 'INDICE_CRIMEN'}

# The column INDICE_CRIMEN is divided by quintiles
df[list(types_column.keys())[0]] = pd.qcut(x=df[list(types_column.
 ↪values())[0]], q=len(types),
                                             labels=types)

# You can generate as many categorical columns from numeric columns as you like.
```

```
numeric_features, continuous_features, discrete_features, categorical_features,
    object_features = update_datatypes_columns()

df.head()
```

```
['LON',
 'LAT',
 'VALOR_MEDIANO',
 'INDICE_CRIMEN',
 'PCT_ZONA_RESIDENCIAL',
 'PCT_ZONA_INDUSTRIAL',
 'RIO_CHARLES',
 'OXIDO_NITROSO_PPM',
 'N_HABITACIONES_MEDIO',
 'PCT_CASAS_40S',
 'DIS',
 'DIS_AUTOPISTAS',
 'CARGA_FISCAL',
 'RATIO_PROFESORES',
 'PCT_NEGRA',
 'PCT_CLASE_BAJA']

['LON',
 'LAT',
 'VALOR_MEDIANO',
 'INDICE_CRIMEN',
 'PCT_ZONA_RESIDENCIAL',
 'PCT_ZONA_INDUSTRIAL',
 'OXIDO_NITROSO_PPM',
 'N_HABITACIONES_MEDIO',
 'PCT_CASAS_40S',
 'DIS',
 'RATIO_PROFESORES',
 'PCT_NEGRA',
 'PCT_CLASE_BAJA']

['RIO_CHARLES', 'DIS_AUTOPISTAS', 'CARGA_FISCAL']

['TIPO_VALOR_MEDIANO', 'TYPE_INDICE_CRIMEN']

['CIUDAD']
```

[21]:

```
        CIUDAD     LON      LAT  VALOR_MEDIANO  INDICE_CRIMEN  \
0       Nahant -70.955  42.2550           24.0        0.00632
1  Swampscott -70.950  42.2875           21.6        0.02731
2  Swampscott -70.936  42.2830           34.7        0.02729
3   Marblehead -70.928  42.2930           33.4        0.03237
4   Marblehead -70.922  42.2980           36.2        0.06905
```

```
   PCT_ZONA_RESIDENCIAL  PCT_ZONA_INDUSTRIAL  RIO_CHARLES  OXIDO_NITROSO_PPM  \
0                  18.0                 2.31            0              0.538
1                   0.0                 7.07            0              0.469
2                   0.0                 7.07            0              0.469
3                   0.0                 2.18            0              0.458
4                   0.0                 2.18            0              0.458

   N_HABITACIONES_MEDIO  PCT_CASAS_40S     DIS  DIS_AUTOPISTAS  CARGA_FISCAL  \
0                 6.575           65.2  4.0900               1           296
1                 6.421           78.9  4.9671               2           242
2                 7.185           61.1  4.9671               2           242
3                 6.998           45.8  6.0622               3           222
4                 7.147           54.2  6.0622               3           222

   RATIO_PROFESORES  PCT_NEGRA  PCT_CLASE_BAJA TIPO_VALOR_MEDIANO  \
0              15.3     396.90            4.98               alto
1              17.8     396.90            9.14              medio
2              17.8     392.83            4.03               alto
3              18.7     394.63            2.94               alto
4              18.7     396.90            5.33               alto

  TYPE_INDICE_CRIMEN
0           very low
1           very low
2           very low
3           very low
4                low
```

4.6.2 Contingency Table

[22]:
```python
@interact(col1=categorical_features, col2=categorical_features[::-1],␣
↪col3=numeric_features,
          operation=['SIZE', 'MEAN', 'STD', 'PROBABILITY'])
def _(col1, col2, col3, operation):
    if len(categorical_features) < 2:
        return

    if operation == 'MEAN':
        aggfunc = np.mean
    elif operation == 'STD':
        aggfunc = np.std
    else:
        aggfunc = np.size

    contingency_table = df.pivot_table(
```

```
        values=col3, index=col1, columns=col2, aggfunc=aggfunc, fill_value=0).
↪dropna(axis=0, how='all')

    # display(contingency_table)
    plt.rcParams['figure.figsize'] = (8, 8)

    if operation == 'PROBABILITY':
        contingency_table = contingency_table.astype('float').
↪div(contingency_table.sum(axis=1),
                                                            axis=0)
        sns.heatmap(contingency_table, annot=True, fmt='.2%', square=True)
    else:
        sns.heatmap(contingency_table, annot=True, fmt='g', square=True)

    plt.show()
```

```
interactive(children=(Dropdown(description='col1',␣
↪options=('TIPO_VALOR_MEDIANO', 'TYPE_INDICE_CRIMEN'), value…
```

4.6.3 Statistics

[23]:
```
bars = pd.DataFrame()

for num_col in numeric_features:
    bars[num_col] = df.groupby(type_target_column)[num_col].mean()

bars
```

[23]:
```
                          LON        LAT  VALOR_MEDIANO  INDICE_CRIMEN  \
TIPO_VALOR_MEDIANO
bajo               -71.027777  42.216648      14.127326       8.629393
medio              -71.053528  42.216908      21.288095       1.317865
alto               -71.088930  42.215752      32.489759       0.739675

                    PCT_ZONA_RESIDENCIAL  PCT_ZONA_INDUSTRIAL  RIO_CHARLES  \
TIPO_VALOR_MEDIANO
bajo                            2.363372            15.763023     0.034884
medio                           8.437500            10.789405     0.071429
alto                           23.650602             6.694880     0.102410

                    OXIDO_NITROSO_PPM  N_HABITACIONES_MEDIO  PCT_CASAS_40S  \
TIPO_VALOR_MEDIANO
bajo                         0.639860              5.908570      89.284302
medio                        0.531101              6.081262      62.485119
alto                         0.490331              6.880114      53.280120

                         DIS  DIS_AUTOPISTAS  CARGA_FISCAL  RATIO_PROFESORES  \
TIPO_VALOR_MEDIANO
```

```
bajo                2.750814      14.773256    526.436047          19.572093
medio               4.122695       7.821429    378.446429          18.641071
alto                4.545414       5.885542    315.915663          17.110843

                     PCT_NEGRA  PCT_CLASE_BAJA
TIPO_VALOR_MEDIANO
bajo                305.324302       19.649302
medio               382.010714       11.564405
alto                384.237831        6.505723
```

[24]:
```
@interact(var_1=numeric_features, var_2=numeric_features,␣
 ↪var_3=numeric_features,
          operation=['PROBABILITY', 'SIZE', 'MEAN', 'STD'],␣
 ↪group=categorical_features,
          bar_type=['VERTICALES', 'HORIZONTALES', 'APILADAS'])
def _(var_1, var_2, var_3, group, operation, bar_type):
    plt.rcParams['figure.figsize'] = (10, 5)

    if operation == 'PROBABILITY':
        bars = pd.DataFrame({var_1: df.groupby(group)[var_1].size(),
                             var_2: df.groupby(group)[var_2].size(),
                             var_3: df.groupby(group)[var_3].size()})

        bars = pd.DataFrame({var_1: bars[var_1].astype('float').div(bars[var_1].
 ↪sum()),
                             var_2: bars[var_2].astype('float').div(bars[var_2].
 ↪sum()),
                             var_3: bars[var_3].astype('float').div(bars[var_3].
 ↪sum())})
        ylabel = 'Cantidad de elementos (probabilidad)'
    if operation == 'SIZE':
        bars = pd.DataFrame({var_1: df.groupby(group)[var_1].size(),
                             var_2: df.groupby(group)[var_2].size(),
                             var_3: df.groupby(group)[var_3].size()})
        ylabel = 'Cantidad de elementos'
    elif operation == 'MEAN':
        bars = pd.DataFrame({var_1: df.groupby(group)[var_1].mean(),
                             var_2: df.groupby(group)[var_2].mean(),
                             var_3: df.groupby(group)[var_3].mean()})
        ylabel = 'Valores medios'
    elif operation == 'STD':
        bars = pd.DataFrame({var_1: df.groupby(group)[var_1].std(),
                             var_2: df.groupby(group)[var_2].std(),
                             var_3: df.groupby(group)[var_3].std()})
        ylabel = 'Desviación estandar'

    if bar_type == 'VERTICALES':
```

```
        bars.plot.bar(rot=0, title='Barras verticales', ylabel=ylabel)
    elif bar_type == 'HORIZONTALES':
        bars.plot.barh(title='Barras horizontales', ylabel=ylabel)
    elif bar_type == 'APILADAS':
        bars.div(bars.sum(1).astype(float), axis=0).plot(kind='bar',␣
↪stacked=True, title='Barras apiladas',
                                                         ylabel=ylabel)

    plt.show()
```

```
interactive(children=(Dropdown(description='var_1', options=('LON', 'LAT',␣
↪'VALOR_MEDIANO', 'INDICE_CRIMEN', '…
```

4.7 Temporal Trends

[25]:
```
# There are no temporary variables in the dataset.
```

4.8 Map

[26]:
```
marker_cluster = MarkerCluster()
site_map = folium.Map(location=[42.18579, -71.05133],
                      zoom_start=10)
site_map.add_child(marker_cluster)

# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the␣
↪map.
formatter = 'function(num) {return L.Util.formatNum(num, 5);};'
mouse_position = MousePosition(position='topright', separator=' Long: ',␣
↪empty_string='NaN', lng_first=False,
                               num_digits=20, prefix='Lat:',␣
↪lat_formatter=formatter, lng_formatter=formatter)

site_map.add_child(mouse_position)

# Se agregan las ubicaciones.
for index, record in df.iterrows():
    if record.TIPO_VALOR_MEDIANO == 'bajo':
        icon_color = 'green'
    elif record.TIPO_VALOR_MEDIANO == 'medio':
        icon_color = 'yellow'
    else:
        icon_color = 'red'

    marker = folium.Marker(location=[record.LAT, record.LON],
                           icon=folium.Icon(color='white',␣
↪icon_color=icon_color), popup='HOME')
```

```
    marker_cluster.add_child(marker)

site_map
```

[26]: <folium.folium.Map at 0x2170f7fcfa0>

Se eliminan las columnas categóricas creadas anteriormente.

[27]:
```
df.drop([type_target_column], axis=1, inplace=True)
df.drop(types_column.keys(), axis=1, inplace=True)

numeric_features, continuous_features, discrete_features, categorical_features,␣
 ↪object_features = update_datatypes_columns()
```

```
['LON',
 'LAT',
 'VALOR_MEDIANO',
 'INDICE_CRIMEN',
 'PCT_ZONA_RESIDENCIAL',
 'PCT_ZONA_INDUSTRIAL',
 'RIO_CHARLES',
 'OXIDO_NITROSO_PPM',
 'N_HABITACIONES_MEDIO',
 'PCT_CASAS_40S',
 'DIS',
 'DIS_AUTOPISTAS',
 'CARGA_FISCAL',
 'RATIO_PROFESORES',
 'PCT_NEGRA',
 'PCT_CLASE_BAJA']

['LON',
 'LAT',
 'VALOR_MEDIANO',
 'INDICE_CRIMEN',
 'PCT_ZONA_RESIDENCIAL',
 'PCT_ZONA_INDUSTRIAL',
 'OXIDO_NITROSO_PPM',
 'N_HABITACIONES_MEDIO',
 'PCT_CASAS_40S',
 'DIS',
 'RATIO_PROFESORES',
 'PCT_NEGRA',
 'PCT_CLASE_BAJA']

['RIO_CHARLES', 'DIS_AUTOPISTAS', 'CARGA_FISCAL']

[]

['CIUDAD']
```

5 Data Cleaning

[28]:
```
df.head()
```

[28]:
```
        CIUDAD     LON      LAT  VALOR_MEDIANO  INDICE_CRIMEN  \
0       Nahant -70.955  42.2550           24.0        0.00632
1   Swampscott -70.950  42.2875           21.6        0.02731
2   Swampscott -70.936  42.2830           34.7        0.02729
3   Marblehead -70.928  42.2930           33.4        0.03237
4   Marblehead -70.922  42.2980           36.2        0.06905

   PCT_ZONA_RESIDENCIAL  PCT_ZONA_INDUSTRIAL  RIO_CHARLES  OXIDO_NITROSO_PPM  \
0                  18.0                 2.31            0              0.538
1                   0.0                 7.07            0              0.469
2                   0.0                 7.07            0              0.469
3                   0.0                 2.18            0              0.458
4                   0.0                 2.18            0              0.458

   N_HABITACIONES_MEDIO  PCT_CASAS_40S     DIS  DIS_AUTOPISTAS  CARGA_FISCAL  \
0                 6.575           65.2  4.0900               1           296
1                 6.421           78.9  4.9671               2           242
2                 7.185           61.1  4.9671               2           242
3                 6.998           45.8  6.0622               3           222
4                 7.147           54.2  6.0622               3           222

   RATIO_PROFESORES  PCT_NEGRA  PCT_CLASE_BAJA
0              15.3     396.90            4.98
1              17.8     396.90            9.14
2              17.8     392.83            4.03
3              18.7     394.63            2.94
4              18.7     396.90            5.33
```

[29]:
```
# Completely empty rows are removed.
# In case you want to delete the row if any of its values is missing, use 'any'
 ↪in the 'how' parameter
# Use subset['col1', 'col2'] if you want to apply to some columns only.
df.dropna(axis=0, how='all', inplace=True)

df.drop_duplicates(keep='first', inplace=True)

df.shape
```

[29]:
```
(506, 17)
```

5.1 Multicollinearity

[30]:
```python
def vif_calc(df):
    vif_data = pd.DataFrame()
    vif_data['FEAUTURE'] = df.columns

    vif_data['VIF'] = [variance_inflation_factor(
        df.values, i) for i in range(len(df.columns))]

    return vif_data
```

- Si se desea usar VIF (Variance Inflation Factor), se debe utilizar el dataframe original.
- Se hace una copia para ver su funcionamiento.
- También se puede incluir las variables generadas a partir de los datos categóricos.

[31]:
```python
numeric_features_bk = numeric_features.copy()
numeric_features_bk.remove(target_column)
df_bk = df[numeric_features_bk].copy()

while True:
    vif_data = vif_calc(df_bk)

    big_vif = vif_data[vif_data.VIF >= 5].sort_values(
        by='VIF', ascending=False).head(1)

    if big_vif.shape[0] > 0:
        numeric_features_bk.remove(big_vif.iloc[0]['FEAUTURE'])
        df_bk = df[numeric_features_bk]

        print('Removed ' + big_vif.iloc[0]['FEAUTURE'] +
              ' with VIF=' + str(big_vif.iloc[0]['VIF']))
    else:
        break

del numeric_features_bk, df_bk

vif_data
```

```
Removed LAT with VIF=341127.9177264702
Removed LON with VIF=578.6409117671585
Removed RATIO_PROFESORES with VIF=85.02954731061801
Removed OXIDO_NITROSO_PPM with VIF=73.89417092973886
Removed CARGA_FISCAL with VIF=57.72034668372636
Removed N_HABITACIONES_MEDIO with VIF=39.069063497543915
Removed PCT_CASAS_40S with VIF=14.000757811090512
Removed PCT_NEGRA with VIF=10.074224239820218
Removed PCT_ZONA_INDUSTRIAL with VIF=6.900077364487575
```

[31]:
```
                FEAUTURE       VIF
0          INDICE_CRIMEN  2.040522
1   PCT_ZONA_RESIDENCIAL  2.237534
2            RIO_CHARLES  1.059249
3                    DIS  3.941629
4          DIS_AUTOPISTAS  3.738091
5          PCT_CLASE_BAJA  4.248513
```

5.2 Cardinality

If a column has the same value always (> 90%), that column can be deleted.

[32]:
```python
n_records = len(df)


def duplicate_column_values(df):
    resume = pd.DataFrame(columns=['VARIABLE', 'LESS_COMMON',
                          '% LESS_COMMON', 'MORE_COMMON', '% MORE_COMMON',␣
 ↪'DATA_TYPE'])

    for columna in df:
        n_per_value = df[columna].value_counts()
        more_common = n_per_value.iloc[0]
        less_common = n_per_value.iloc[-1]

        new_df = pd.DataFrame(data={'VARIABLE': [columna],
                                    'LESS_COMMON': [less_common],
                                    '% LESS_COMMON': [round(less_common * 100 /␣
 ↪(1.0 * n_records), 3)],
                                    'MORE_COMMON': [more_common],
                                    '% MORE_COMMON': [round(more_common * 100 /␣
 ↪(1.0 * n_records), 3)],
                                    'DATA_TYPE': [df[columna].dtype]})

        resume = pd.concat([resume, new_df], ignore_index=True)

    return resume


resume = duplicate_column_values(df)
resume
```

[32]:
```
                VARIABLE LESS_COMMON  % LESS_COMMON MORE_COMMON  \
0                 CIUDAD           1          0.198          30
1                    LON           1          0.198           5
2                    LAT           1          0.198           5
3           VALOR_MEDIANO           1          0.198          16
```

```
4          INDICE_CRIMEN            1         0.198            2
5   PCT_ZONA_RESIDENCIAL            1         0.198          372
6    PCT_ZONA_INDUSTRIAL            1         0.198          132
7            RIO_CHARLES           35         6.917          471
8      OXIDO_NITROSO_PPM            1         0.198           23
9   N_HABITACIONES_MEDIO            1         0.198            3
10         PCT_CASAS_40S            1         0.198           43
11                   DIS            1         0.198            5
12        DIS_AUTOPISTAS           17         3.360          132
13          CARGA_FISCAL            1         0.198          132
14      RATIO_PROFESORES            1         0.198          140
15             PCT_NEGRA            1         0.198          121
16        PCT_CLASE_BAJA            1         0.198            3

    % MORE_COMMON DATA_TYPE
0           5.929    object
1           0.988   float64
2           0.988   float64
3           3.162   float64
4           0.395   float64
5          73.518   float64
6          26.087   float64
7          93.083     int64
8           4.545   float64
9           0.593   float64
10          8.498   float64
11          0.988   float64
12         26.087     int64
13         26.087     int64
14         27.668   float64
15         23.913   float64
16          0.593   float64
```

[33]:

```python
resume = resume.loc[resume['% MORE_COMMON'] > 90.]
resume
```

[33]:

```
      VARIABLE LESS_COMMON  % LESS_COMMON MORE_COMMON  % MORE_COMMON DATA_TYPE
7  RIO_CHARLES          35          6.917         471         93.083     int64
```

[34]:

```python
if resume.loc[resume['% MORE_COMMON'] > 90.].size == 0:
    print('No field contains more than 90% of its data repeated.')
else:
    print('Some fields contain more than 90% of their data repeated. They must be removed.')
```

```
Some fields contain more than 90% of their data repeated. They must be removed.
```

The analysis is performed to eliminate or not the columns.

[35]:
```python
# Without normalize it returns the quantity, not the %
df.RIO_CHARLES.value_counts(normalize=True)
```

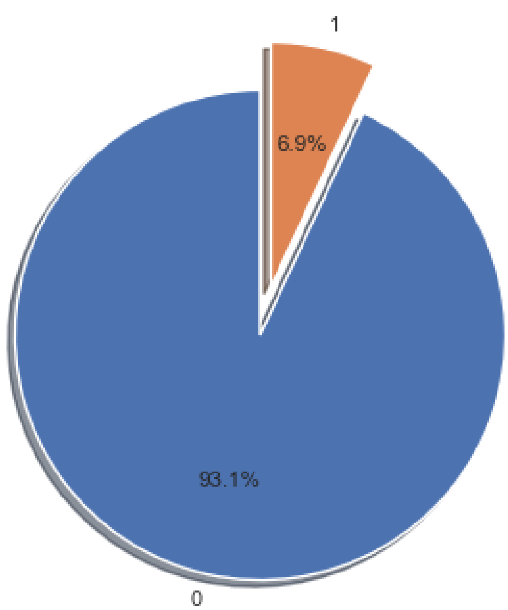
[35]:
```
0    0.93083
1    0.06917
Name: RIO_CHARLES, dtype: float64
```

[36]:
```python
df.RIO_CHARLES.value_counts()
```

[36]:
```
0    471
1     35
Name: RIO_CHARLES, dtype: int64
```

[37]:
```python
labels = df.RIO_CHARLES.value_counts().index.values
sizes = df.RIO_CHARLES.value_counts().values
explode = np.full(df.RIO_CHARLES.value_counts().count(), 0.1)

plt.rcParams['figure.figsize'] = (4, 4)
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



The `RIO_CHARLES` column will not be removed.

5.3 Outliers

It applies to both independent and dependent numerical variables.

[38]:
```python
def outliers_col(df):
    resume = pd.DataFrame(
        columns=['VARIABLE', 'FREQUENCY', 'OUTLIER', 'DATA_TYPE'])

    for col in df.select_dtypes(exclude=[object, 'category', 'datetime64[ns]']).
 ↪drop(['LON', 'LAT'], axis=1):
        # zcores absoluto de cada valor de la columna seleccionada
        zcores = np.abs(stats.zscore(df[col]))

        # TODO: Probar con 1.5 luego, así funcionan los boxplots
        n_outliers = len(df[zcores > 3])

        new_df = pd.DataFrame(data={'VARIABLE': [col],
                                    'FREQUENCY': [n_outliers],
                                    'OUTLIER': [False if n_outliers == 0 else␣
 ↪True],
                                    'DATA_TYPE': [df[col].dtype]})

        resume = pd.concat([resume, new_df], ignore_index=True)

    return resume


resume = outliers_col(df)
resume
```

```
C:\Users\ereye\AppData\Local\Temp\ipykernel_18528\3870185035.py:17:
FutureWarning: In a future version, object-dtype columns with all-bool values
will not be included in reductions with bool_only=True. Explicitly cast to bool
dtype instead.
  resume = pd.concat([resume, new_df], ignore_index=True)
```

[38]:

	VARIABLE	FREQUENCY	OUTLIER	DATA_TYPE
0	VALOR_MEDIANO	0	False	float64
1	INDICE_CRIMEN	8	True	float64
2	PCT_ZONA_RESIDENCIAL	14	True	float64
3	PCT_ZONA_INDUSTRIAL	0	False	float64
4	RIO_CHARLES	35	True	int64
5	OXIDO_NITROSO_PPM	0	False	float64
6	N_HABITACIONES_MEDIO	8	True	float64
7	PCT_CASAS_40S	0	False	float64
8	DIS	5	True	float64
9	DIS_AUTOPISTAS	0	False	int64
10	CARGA_FISCAL	0	False	int64

```
11      RATIO_PROFESORES          0    False    float64
12             PCT_NEGRA         25     True    float64
13         PCT_CLASE_BAJA         5     True    float64
```

[39]:
```
if resume.FREQUENCY.where(resume.FREQUENCY > 0).count() == 0:
    print('There are no outliers.')
else:
    print('There are some outliers. We can do a boxplot to visualize the␣
 ↪outliers better.')
```

```
There are some outliers. We can do a boxplot to visualize the outliers better.
```

Outliers are removed until none remain.

When there are many numerical variables, the elimination of outliers causes other outliers in other columns and the size of the dataset can be greatly reduced with the iterative process.

[40]:
```
# # It is commented out because a lot of data is lost during outlier removal.

# while resume.FREQUENCY.where(resume.FREQUENCY > 0).count() > 0:
#     for col in resume.VARIABLE:
#         before_len = df.shape[0]
#         df = df[np.abs(stats.zscore(df[col])) < 3]
#         after_len = df.shape[0]

#         print('0, ' if before_len == df.shape[0] else f'-{before_len - df.
 ↪shape[0]}, ',
#               end='')

#     resume = outliers_col(df)
```

6 Machine Learning

6.1 Preprocess

Se crea el `preprocess` para las variables independientes.

[41]:
```
preprocessor_resume = pd.DataFrame(data=df.dtypes, columns=['TYPE'])
preprocessor_resume['VALUES'] = preprocessor_resume.apply(lambda x: df[x.name].
 ↪unique(),
                                                          axis=1)
preprocessor_resume['VALUES_LEN'] = preprocessor_resume.apply(lambda x:␣
 ↪len(df[x.name].unique()),
                                                              axis=1)
preprocessor_resume[['IMPUTER', 'TRANSFORMER', 'STATE']] = 'UNKNOWN'

#␣
 ↪-----------------------------------------------------------------------------------------
```

```
preprocessor_resume.loc[[], ['IMPUTER']] = 'Mean'
preprocessor_resume.loc[['RIO_CHARLES'], ['IMPUTER']] = 'Mode'
preprocessor_resume.loc[['INDICE_CRIMEN', 'PCT_ZONA_RESIDENCIAL',␣
 ↪'PCT_ZONA_INDUSTRIAL', 'OXIDO_NITROSO_PPM',
                         'N_HABITACIONES_MEDIO', 'PCT_CASAS_40S', 'DIS',␣
 ↪'RATIO_PROFESORES', 'PCT_NEGRA',
                         'PCT_CLASE_BAJA', 'CARGA_FISCAL', 'DIS_AUTOPISTAS'],␣
 ↪['IMPUTER']] = 'Median'
#␣
 ↪-------------------------------------------------------------------------------------------
preprocessor_resume.loc[['INDICE_CRIMEN', 'PCT_ZONA_RESIDENCIAL',␣
 ↪'PCT_ZONA_INDUSTRIAL', 'OXIDO_NITROSO_PPM',
                         'N_HABITACIONES_MEDIO', 'PCT_CASAS_40S', 'DIS',␣
 ↪'RATIO_PROFESORES', 'PCT_NEGRA',
                         'PCT_CLASE_BAJA', 'DIS_AUTOPISTAS', 'CARGA_FISCAL'],␣
 ↪['TRANSFORMER']] = 'STANDARD_SCALER'
preprocessor_resume.loc[['RIO_CHARLES'], ['TRANSFORMER']] = 'ONE_HOT_ENCODER'
preprocessor_resume.loc[[], ['TRANSFORMER']] = 'ORDINAL_ENCODER'
#␣
 ↪-------------------------------------------------------------------------------------------
median_standard_scaler_transformer = Pipeline(steps=[('imputer',␣
 ↪SimpleImputer(strategy='median')),
                                                     ('transformer',␣
 ↪StandardScaler())])
mode_one_hot_encoder_transformer = Pipeline(steps=[('imputer',␣
 ↪SimpleImputer(strategy='most_frequent')),
                                                   ('transformer',␣
 ↪OneHotEncoder(sparse=True, drop='first'))])

median_standard_scaler_features = preprocessor_resume.query(
    'IMPUTER == "Median" and TRANSFORMER == "STANDARD_SCALER"').index.to_list()
mode_one_hot_encoder_features = preprocessor_resume.query(
    'IMPUTER == "Mode" and TRANSFORMER == "ONE_HOT_ENCODER"').index.to_list()

preprocessor_resume.loc[median_standard_scaler_features +
                        mode_one_hot_encoder_features, ['STATE']] = 'OK'

preprocessor = ColumnTransformer(transformers=[('median_standard_scaler',
                                                ␣
 ↪median_standard_scaler_transformer, median_standard_scaler_features),
                                                ('mode_one_hot_encoder',
                                                ␣
 ↪mode_one_hot_encoder_transformer, mode_one_hot_encoder_features)])
#␣
 ↪-------------------------------------------------------------------------------------------
```

```
display(preprocessor)
display(preprocessor_resume)

X = preprocessor.fit_transform(X=df)
print(f'Dimensiones de los datos: {X.shape}.')
del X
```

```
ColumnTransformer(transformers=[('median_standard_scaler',
                                 Pipeline(steps=[('imputer',
                                                  ␣
 ↪SimpleImputer(strategy='median')),
                                                 ('transformer',
                                                  StandardScaler())]),
                                 ['INDICE_CRIMEN', 'PCT_ZONA_RESIDENCIAL',
                                  'PCT_ZONA_INDUSTRIAL', 'OXIDO_NITROSO_PPM',
                                  'N_HABITACIONES_MEDIO', 'PCT_CASAS_40S',
                                  'DIS', 'DIS_AUTOPISTAS', 'CARGA_FISCAL',
                                  'RATIO_PROFESORES', 'PCT_NEGRA',
                                  'PCT_CLASE_BAJA']),
                                ('mode_one_hot_encoder',
                                 Pipeline(steps=[('imputer',
                                                  ␣
 ↪SimpleImputer(strategy='most_frequent')),
                                                 ('transformer',
                                                  OneHotEncoder(drop='first'))]),
                                 ['RIO_CHARLES'])])
```

```
                         TYPE  \
CIUDAD                 object
LON                   float64
LAT                   float64
VALOR_MEDIANO         float64
INDICE_CRIMEN         float64
PCT_ZONA_RESIDENCIAL  float64
PCT_ZONA_INDUSTRIAL   float64
RIO_CHARLES             int64
OXIDO_NITROSO_PPM     float64
N_HABITACIONES_MEDIO  float64
PCT_CASAS_40S         float64
DIS                   float64
DIS_AUTOPISTAS          int64
CARGA_FISCAL            int64
RATIO_PROFESORES      float64
PCT_NEGRA             float64
PCT_CLASE_BAJA        float64

                                                                VALUES  \
CIUDAD                [Nahant, Swampscott, Marblehead, Salem, Lynn, …
```

```
LON                   [-70.955, -70.95, -70.936, -70.928, -70.922, -…
LAT                   [42.255, 42.2875, 42.283, 42.293, 42.298, 42.3…
VALOR_MEDIANO         [24.0, 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 22…
INDICE_CRIMEN         [0.00632, 0.02731, 0.02729, 0.03237, 0.06905, …
PCT_ZONA_RESIDENCIAL  [18.0, 0.0, 12.5, 75.0, 21.0, 90.0, 85.0, 100…
PCT_ZONA_INDUSTRIAL   [2.31, 7.07, 2.18, 7.87, 8.14, 5.96, 2.95, 6.9…
RIO_CHARLES                                                     [0, 1]
OXIDO_NITROSO_PPM     [0.5379999999999999, 0.469, 0.458, 0.524, 0.49…
N_HABITACIONES_MEDIO  [6.575, 6.421, 7.185, 6.997999999999998, 7.147…
PCT_CASAS_40S         [65.2, 78.9, 61.1, 45.8, 54.2, 58.7, 66.6, 96…
DIS                   [4.09, 4.9671, 6.0622, 5.5605, 5.9505, 6.0821,…
DIS_AUTOPISTAS                             [1, 2, 3, 5, 4, 8, 6, 7, 24]
CARGA_FISCAL          [296, 242, 222, 311, 307, 279, 252, 233, 243, …
RATIO_PROFESORES      [15.3, 17.8, 18.7, 15.2, 21.0, 19.2, 18.3, 17…
PCT_NEGRA             [396.9, 392.83, 394.63, 394.12, 395.6, 386.63,…
PCT_CLASE_BAJA        [4.98, 9.14, 4.03, 2.94, 5.33, 5.21, 12.43, 19…

                      VALUES_LEN  IMPUTER     TRANSFORMER    STATE
CIUDAD                        92  UNKNOWN         UNKNOWN  UNKNOWN
LON                          375  UNKNOWN         UNKNOWN  UNKNOWN
LAT                          376  UNKNOWN         UNKNOWN  UNKNOWN
VALOR_MEDIANO                228  UNKNOWN         UNKNOWN  UNKNOWN
INDICE_CRIMEN                504   Median ＳTANDARD_SCALER       OK
PCT_ZONA_RESIDENCIAL          26   Median STANDARD_SCALER       OK
PCT_ZONA_INDUSTRIAL           76   Median STANDARD_SCALER       OK
RIO_CHARLES                    2     Mode ONE_HOT_ENCODER       OK
OXIDO_NITROSO_PPM             81   Median STANDARD_SCALER       OK
N_HABITACIONES_MEDIO         446   Median STANDARD_SCALER       OK
PCT_CASAS_40S                356   Median STANDARD_SCALER       OK
DIS                          412   Median STANDARD_SCALER       OK
DIS_AUTOPISTAS                 9   Median STANDARD_SCALER       OK
CARGA_FISCAL                  66   Median STANDARD_SCALER       OK
RATIO_PROFESORES              46   Median STANDARD_SCALER       OK
PCT_NEGRA                    357   Median STANDARD_SCALER       OK
PCT_CLASE_BAJA               455   Median STANDARD_SCALER       OK

Dimensiones de los datos: (506, 13).
```

Se defnine el `target`.

[42]:
```
targets = df[target_column]
targets.shape
```

[42]:
```
(506,)
```

6.2 Cross Validation

[43]:
```
results = pd.DataFrame(
    columns=['NAME', 'TYPE', 'POLY_DEGREE', 'SCORE', 'ESTIMATOR'])


def my_grid_search_cv(model, X, y, results, param_grid, poly_degrees):
    estimators = []

    for degree in poly_degrees:
        steps = [('preprocessor', preprocessor),
                 ('polynomial', PolynomialFeatures(degree=degree)),
                 ('model', model)]

        pipe = Pipeline(steps=steps)

        target_transformer = MinMaxScaler(feature_range=(0, 1))
        estimator = TransformedTargetRegressor(regressor=pipe,
                                               transformer=target_transformer)

        grid_search = GridSearchCV(estimator=estimator, param_grid=param_grid,
                                   scoring='neg_mean_absolute_error', cv=10,␣
↪n_jobs=-1, return_train_score=True)
        grid_search.fit(X, y)
        estimator = grid_search.best_estimator_
        estimators.append(estimator)

        score = round(grid_search.best_score_, 4)
        display(estimator)
        print(
            f'The model {str(model)} has a prediction error of +-{-score}␣
↪dollars.')

        new_df = pd.DataFrame(data={'NAME': [str(model)], 'TYPE': 'ML',␣
↪'POLY_DEGREE': degree,
                                    'SCORE': [score], 'ESTIMATOR': [estimator]})
        results = pd.concat([results, new_df], ignore_index=True)

    return (results,) + tuple(estimators)
```

6.3 Linear Regression

[44]:
```
results, ols = my_grid_search_cv(model=LinearRegression(), X=df, y=targets,␣
↪results=results,
                                 param_grid={}, poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
```

```
↪ColumnTransformer(transformers=[('median_standard_scaler',
↪       Pipeline(steps=[('imputer',
↪                        SimpleImputer(strategy='median')),
↪                       ('transformer',
↪                        StandardScaler())]),
↪       ['INDICE_CRIMEN',
↪        'PCT_ZONA_RESIDENCIAL',
↪        'PCT_ZONA_INDUSTRIAL',
↪        'OXIDO_NITROSO_PPM',
↪        'N_HABITACIONES_MEDIO',
↪        'PCT_CASAS_40S',
↪        'DIS',
↪        'DIS_AUTOPISTAS',
↪        'CARGA_FISCAL',
↪        'RATIO_PROFESORES',
↪        'PCT_NEGRA',
↪        'PCT_CLASE_BAJA']),
↪      ('mode_one_hot_encoder',
↪       Pipeline(steps=[('imputer',
↪                        SimpleImputer(strategy='most_frequent')),
↪                       ('transformer',
↪                        OneHotEncoder(drop='first'))]),
↪       ['RIO_CHARLES'])])),
```

```
                                                  ('polynomial',
                                                  ␣
↪PolynomialFeatures(degree=1)),
                                                  ('model',
                                                   LinearRegression())]),
                           transformer=MinMaxScaler())
```

```
The model LinearRegression() has a prediction error of +-3.9665 dollars.
```

[44]:

```
                 NAME TYPE POLY_DEGREE   SCORE  \
0  LinearRegression()   ML           1 -3.9665

                                           ESTIMATOR
0  TransformedTargetRegressor(regressor=Pipeline(…
```

6.4 Regularization

6.4.1 Lasso

[45]:

```
param_grid = {'regressor__model__alpha': np.linspace(0.1, 1.0, 10)}
results, lasso = my_grid_search_cv(model=Lasso(), X=df, y=targets,␣
 ↪results=results,
                                   param_grid=param_grid, poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
                                                  ␣
↪ColumnTransformer(transformers=[('median_standard_scaler',
                                                                      ␣
↪     Pipeline(steps=[('imputer',
                                                                      ␣
↪                      SimpleImputer(strategy='median')),
                                                                      ␣
↪                     ('transformer',
                                                                      ␣
↪                      StandardScaler())]),
                                                                      ␣
↪     ['INDICE_CRIMEN',
                                                                      ␣
↪      'PCT_ZONA_RESIDENCIAL',
                                                                      ␣
↪      'PCT_ZONA_INDUSTRIAL',
                                                                      ␣
↪      'OXIDO_NITROSO_PPM',
                                                                      ␣
↪      'N_HABITACIONES_MEDIO',
                                                                      ␣
↪      'PCT_CASAS_40S',
```

```
↪       'DIS',
↪       'DIS_AUTOPISTAS',
↪       'CARGA_FISCAL',
↪       'RATIO_PROFESORES',
↪       'PCT_NEGRA',
↪       'PCT_CLASE_BAJA']),
↪     ('mode_one_hot_encoder',
↪      Pipeline(steps=[('imputer',
↪                       SimpleImputer(strategy='most_frequent')),
↪                      ('transformer',
↪                       OneHotEncoder(drop='first'))]),
↪      ['RIO_CHARLES'])])),
                                                  ('polynomial',
↪PolynomialFeatures(degree=1)),
                                                  ('model',
                                                   Lasso(alpha=0.1))]),
                           transformer=MinMaxScaler())
The model Lasso() has a prediction error of +-5.8531 dollars.
```

[45]:

```
                 NAME TYPE POLY_DEGREE   SCORE  \
0  LinearRegression()   ML           1 -3.9665
1             Lasso()   ML           1 -5.8531

                                        ESTIMATOR
0  TransformedTargetRegressor(regressor=Pipeline(…
1  TransformedTargetRegressor(regressor=Pipeline(…
```

6.4.2 Ridge

[46]:

```python
param_grid = {'regressor__model__alpha': np.linspace(0.1, 1.0, 10)}
results, ridge = my_grid_search_cv(model=Ridge(), X=df, y=targets,␣
 ↪results=results,
                                   param_grid=param_grid, poly_degrees=[1])
```

```
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
 ↪ColumnTransformer(transformers=[('median_standard_scaler',
 ↪      Pipeline(steps=[('imputer',
 ↪                       SimpleImputer(strategy='median')),
 ↪                      ('transformer',
 ↪                       StandardScaler())]),
 ↪      ['INDICE_CRIMEN',
 ↪       'PCT_ZONA_RESIDENCIAL',
 ↪       'PCT_ZONA_INDUSTRIAL',
 ↪       'OXIDO_NITROSO_PPM',
 ↪       'N_HABITACIONES_MEDIO',
 ↪       'PCT_CASAS_40S',
 ↪       'DIS',
 ↪       'DIS_AUTOPISTAS',
 ↪       'CARGA_FISCAL',
 ↪       'RATIO_PROFESORES',
 ↪       'PCT_NEGRA',
 ↪       'PCT_CLASE_BAJA']),
 ↪     ('mode_one_hot_encoder',
 ↪      Pipeline(steps=[('imputer',
 ↪                       SimpleImputer(strategy='most_frequent')),
 ↪                      ('transformer',
```

```
                            OneHotEncoder(drop='first'))]),
        ['RIO_CHARLES'])])),
                                                              ('polynomial',
PolynomialFeatures(degree=1)),
                                                              ('model', Ridge())]),
                            transformer=MinMaxScaler())
The model Ridge() has a prediction error of +-3.9507 dollars.
```

[46]:
```
                  NAME TYPE POLY_DEGREE   SCORE  \
0  LinearRegression()   ML           1 -3.9665
1             Lasso()   ML           1 -5.8531
2             Ridge()   ML           1 -3.9507

                                           ESTIMATOR
0  TransformedTargetRegressor(regressor=Pipeline(…
1  TransformedTargetRegressor(regressor=Pipeline(…
2  TransformedTargetRegressor(regressor=Pipeline(…
```

6.4.3 ElasticNet

[47]:
```python
param_grid = {'regressor__model__alpha': np.linspace(0.1, 1.0, 10)}
results, elastic_net = my_grid_search_cv(model=ElasticNet(), X=df, y=targets,
 results=results,
                                          param_grid=param_grid,
 poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('median_standard_scaler',
        Pipeline(steps=[('imputer',
                            SimpleImputer(strategy='median')),
                           ('transformer',
                            StandardScaler())]),
        ['INDICE_CRIMEN',
         'PCT_ZONA_RESIDENCIAL',
         'PCT_ZONA_INDUSTRIAL',
```

```
↪       'OXIDO_NITROSO_PPM',
↪       'N_HABITACIONES_MEDIO',
↪       'PCT_CASAS_40S',
↪       'DIS',
↪       'DIS_AUTOPISTAS',
↪       'CARGA_FISCAL',
↪       'RATIO_PROFESORES',
↪       'PCT_NEGRA',
↪       'PCT_CLASE_BAJA']),
↪     ('mode_one_hot_encoder',
↪      Pipeline(steps=[('imputer',
↪                       SimpleImputer(strategy='most_frequent')),
↪                      ('transformer',
↪                       OneHotEncoder(drop='first'))]),
↪      ['RIO_CHARLES'])])),
                                                ('polynomial',
↪PolynomialFeatures(degree=1)),
                                                ('model',
                                                 ElasticNet(alpha=0.1))]),
                           transformer=MinMaxScaler())
The model ElasticNet() has a prediction error of +-4.679 dollars.
```

[47]:

```
                 NAME TYPE POLY_DEGREE   SCORE  \
0  LinearRegression()   ML           1 -3.9665
1             Lasso()   ML           1 -5.8531
2             Ridge()   ML           1 -3.9507
3        ElasticNet()   ML           1 -4.6790

                                           ESTIMATOR
0  TransformedTargetRegressor(regressor=Pipeline(…
```

```
1  TransformedTargetRegressor(regressor=Pipeline(…
2  TransformedTargetRegressor(regressor=Pipeline(…
3  TransformedTargetRegressor(regressor=Pipeline(…
```

6.5 Decision Trees

[48]:
```
param_grid = {'regressor__model__max_depth': range(1, 20),
              'regressor__model__criterion': ['squared_error', 'friedman_mse',␣
 ↪'absolute_error', 'poisson']}
results, decision_tree_regressor =␣
 ↪my_grid_search_cv(model=DecisionTreeRegressor(), X=df, y=targets,
                                                      results=results,␣
 ↪param_grid=param_grid, poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
                                                      ␣
 ↪ColumnTransformer(transformers=[('median_standard_scaler',
                                                                                ␣
 ↪      Pipeline(steps=[('imputer',
                                                                                ␣
 ↪                      SimpleImputer(strategy='median')),
                                                                                ␣
 ↪                     ('transformer',
                                                                                ␣
 ↪                      StandardScaler())]),
                                                                                ␣
 ↪      ['INDICE_CRIMEN',
                                                                                ␣
 ↪       'PCT_ZONA_RESIDENCIAL',
                                                                                ␣
 ↪       'PCT_ZONA_INDUSTRIAL',
                                                                                ␣
 ↪       'OXIDO_NITROSO_PPM',
                                                                                ␣
 ↪       'N_HABITACIONES_MEDIO',
                                                                                ␣
 ↪       'PCT_CASAS_40S',
                                                                                ␣
 ↪       'DIS',…
                                                                                ␣
 ↪       'RATIO_PROFESORES',
                                                                                ␣
 ↪       'PCT_NEGRA',
                                                                                ␣
 ↪       'PCT_CLASE_BAJA']),
```

```
↪     ('mode_one_hot_encoder',
↪      Pipeline(steps=[('imputer',
↪                       SimpleImputer(strategy='most_frequent')),
↪                      ('transformer',
↪                       OneHotEncoder(drop='first'))]),
↪      ['RIO_CHARLES'])])),
                                               ('polynomial',
↪PolynomialFeatures(degree=1)),
                                               ('model',
↪DecisionTreeRegressor(criterion='absolute_error',
↪max_depth=3))]),
                           transformer=MinMaxScaler())
The model DecisionTreeRegressor() has a prediction error of +-3.5071 dollars.
```

[48]:

```
                      NAME TYPE POLY_DEGREE   SCORE  \
0       LinearRegression()   ML           1 -3.9665
1                  Lasso()   ML           1 -5.8531
2                  Ridge()   ML           1 -3.9507
3             ElasticNet()   ML           1 -4.6790
4  DecisionTreeRegressor()   ML           1 -3.5071

                                           ESTIMATOR
0  TransformedTargetRegressor(regressor=Pipeline(…
1  TransformedTargetRegressor(regressor=Pipeline(…
2  TransformedTargetRegressor(regressor=Pipeline(…
3  TransformedTargetRegressor(regressor=Pipeline(…
4  TransformedTargetRegressor(regressor=Pipeline(…
```

6.6 Support Vector Machines (SVM)

[49]:

```
param_grid = {'regressor__model__kernel': ['linear', 'poly', 'sigmoid', 'rbf'],
↪ # Allows transformation to higher levels.
              # Border complexity: linear, curved.
              'regressor__model__gamma': [1e-3, 1e-2, 0.1],
              'regressor__model__C': [1, 10, 100]}          # Controls the
↪tradeoff between training errors and hard margins.
results, svr = my_grid_search_cv(model=SVR(), X=df, y=targets, results=results,
```

```
                                        param_grid=param_grid, poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
                                                                  ␣
↪ColumnTransformer(transformers=[('median_standard_scaler',
                                                                                          ␣
↪       Pipeline(steps=[('imputer',
                                                                                          ␣
↪                         SimpleImputer(strategy='median')),
                                                                                          ␣
↪                        ('transformer',
                                                                                          ␣
↪                         StandardScaler())]),
                                                                                          ␣
↪       ['INDICE_CRIMEN',
                                                                                          ␣
↪        'PCT_ZONA_RESIDENCIAL',
                                                                                          ␣
↪        'PCT_ZONA_INDUSTRIAL',
                                                                                          ␣
↪        'OXIDO_NITROSO_PPM',
                                                                                          ␣
↪        'N_HABITACIONES_MEDIO',
                                                                                          ␣
↪        'PCT_CASAS_40S',
                                                                                          ␣
↪        'DIS',
                                                                                          ␣
↪        'DIS_AUTOPISTAS',
                                                                                          ␣
↪        'CARGA_FISCAL',
                                                                                          ␣
↪        'RATIO_PROFESORES',
                                                                                          ␣
↪        'PCT_NEGRA',
                                                                                          ␣
↪        'PCT_CLASE_BAJA']),
                                                                                          ␣
↪      ('mode_one_hot_encoder',
                                                                                          ␣
↪       Pipeline(steps=[('imputer',
                                                                                          ␣
↪                         SimpleImputer(strategy='most_frequent')),
                                                                                          ␣
↪                        ('transformer',
```

```
↪                         OneHotEncoder(drop='first'))]),
↪       ['RIO_CHARLES'])])),
                                                   ('polynomial',
↪PolynomialFeatures(degree=1)),
                                                   ('model',
                                                    SVR(C=1, gamma=0.01))]),
                           transformer=MinMaxScaler())
The model SVR() has a prediction error of +-3.0232 dollars.
```

[49]:

```
                    NAME TYPE  POLY_DEGREE   SCORE  \
0     LinearRegression()   ML            1 -3.9665
1                Lasso()   ML            1 -5.8531
2                Ridge()   ML            1 -3.9507
3           ElasticNet()   ML            1 -4.6790
4  DecisionTreeRegressor()   ML            1 -3.5071
5                  SVR()   ML            1 -3.0232

                                           ESTIMATOR
0  TransformedTargetRegressor(regressor=Pipeline(…
1  TransformedTargetRegressor(regressor=Pipeline(…
2  TransformedTargetRegressor(regressor=Pipeline(…
3  TransformedTargetRegressor(regressor=Pipeline(…
4  TransformedTargetRegressor(regressor=Pipeline(…
5  TransformedTargetRegressor(regressor=Pipeline(…
```

6.7 K Nearest Neighbors

[50]:

```
param_grid = {'regressor__model__n_neighbors': range(3, 20, 2),
              'regressor__model__metric': ['euclidean', 'manhattan',
 ↪'chebyshev']}
results, k_neighbors_regressor = my_grid_search_cv(model=KNeighborsRegressor(),
 ↪X=df, y=targets, results=results,
                                                   param_grid=param_grid,
 ↪poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
↪ColumnTransformer(transformers=[('median_standard_scaler',
↪       Pipeline(steps=[('imputer',
↪                         SimpleImputer(strategy='median')),
```

```
↪                       ('transformer',
↪                        StandardScaler())]),
↪       ['INDICE_CRIMEN',
↪        'PCT_ZONA_RESIDENCIAL',
↪        'PCT_ZONA_INDUSTRIAL',
↪        'OXIDO_NITROSO_PPM',
↪        'N_HABITACIONES_MEDIO',
↪        'PCT_CASAS_40S',
↪        'DIS',
↪        'DIS_AUTOPISTAS',
↪        'CARGA_FISCAL',
↪        'RATIO_PROFESORES',
↪        'PCT_NEGRA',
↪        'PCT_CLASE_BAJA']),
↪      ('mode_one_hot_encoder',
↪       Pipeline(steps=[('imputer',
↪                        SimpleImputer(strategy='most_frequent')),
↪                       ('transformer',
↪                        OneHotEncoder(drop='first'))]),
↪       ['RIO_CHARLES'])])),
                                                   ('polynomial',
↪PolynomialFeatures(degree=1)),
                                                   ('model',
↪KNeighborsRegressor(metric='manhattan'))]),
```

```
                                          transformer=MinMaxScaler())
The model KNeighborsRegressor() has a prediction error of +-3.3853 dollars.
```

[50]:

```
                     NAME TYPE POLY_DEGREE   SCORE  \
0      LinearRegression()   ML           1 -3.9665
1                 Lasso()   ML           1 -5.8531
2                 Ridge()   ML           1 -3.9507
3            ElasticNet()   ML           1 -4.6790
4 DecisionTreeRegressor()   ML           1 -3.5071
5                   SVR()   ML           1 -3.0232
6   KNeighborsRegressor()   ML           1 -3.3853

                                           ESTIMATOR
0  TransformedTargetRegressor(regressor=Pipeline(…
1  TransformedTargetRegressor(regressor=Pipeline(…
2  TransformedTargetRegressor(regressor=Pipeline(…
3  TransformedTargetRegressor(regressor=Pipeline(…
4  TransformedTargetRegressor(regressor=Pipeline(…
5  TransformedTargetRegressor(regressor=Pipeline(…
6  TransformedTargetRegressor(regressor=Pipeline(…
```

6.8 Naive Bayes

[51]:

```
results, bayesian_ridge = my_grid_search_cv(model=BayesianRidge(), X=df,␣
↪y=targets, results=results,
                                             param_grid={}, poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
                                                                    ␣
↪ColumnTransformer(transformers=[('median_standard_scaler',
                                                                              ␣
↪      Pipeline(steps=[('imputer',
                                                                              ␣
↪                        SimpleImputer(strategy='median')),
                                                                              ␣
↪                       ('transformer',
                                                                              ␣
↪                        StandardScaler())]),
                                                                              ␣
↪      ['INDICE_CRIMEN',
                                                                              ␣
↪       'PCT_ZONA_RESIDENCIAL',
                                                                              ␣
↪       'PCT_ZONA_INDUSTRIAL',
                                                                              ␣
↪       'OXIDO_NITROSO_PPM',
```

```
↪       'N_HABITACIONES_MEDIO',
↪       'PCT_CASAS_40S',
↪       'DIS',
↪       'DIS_AUTOPISTAS',
↪       'CARGA_FISCAL',
↪       'RATIO_PROFESORES',
↪       'PCT_NEGRA',
↪       'PCT_CLASE_BAJA']),
↪     ('mode_one_hot_encoder',
↪      Pipeline(steps=[('imputer',
↪                       SimpleImputer(strategy='most_frequent')),
↪                      ('transformer',
↪                       OneHotEncoder(drop='first'))]),
↪      ['RIO_CHARLES'])])),
                                            ('polynomial',
↪PolynomialFeatures(degree=1)),
                                            ('model',
                                             BayesianRidge())]),
                        transformer=MinMaxScaler())

The model BayesianRidge() has a prediction error of +-3.906 dollars.
```

[51]:

	NAME	TYPE	POLY_DEGREE	SCORE	\
0	LinearRegression()	ML	1	-3.9665	
1	Lasso()	ML	1	-5.8531	
2	Ridge()	ML	1	-3.9507	
3	ElasticNet()	ML	1	-4.6790	
4	DecisionTreeRegressor()	ML	1	-3.5071	
5	SVR()	ML	1	-3.0232	
6	KNeighborsRegressor()	ML	1	-3.3853	
7	BayesianRidge()	ML	1	-3.9060	

```
                                           ESTIMATOR
0  TransformedTargetRegressor(regressor=Pipeline(…
1  TransformedTargetRegressor(regressor=Pipeline(…
2  TransformedTargetRegressor(regressor=Pipeline(…
3  TransformedTargetRegressor(regressor=Pipeline(…
4  TransformedTargetRegressor(regressor=Pipeline(…
5  TransformedTargetRegressor(regressor=Pipeline(…
6  TransformedTargetRegressor(regressor=Pipeline(…
7  TransformedTargetRegressor(regressor=Pipeline(…
```

6.9 Ensemble Methods

6.9.1 Bagging

[52]:
```python
param_grid = {'regressor__model__base_estimator': [LinearRegression(), Lasso(),␣
 ↪Ridge(), ElasticNet(), DecisionTreeRegressor(),
                                                 SVR(),␣
 ↪KNeighborsRegressor(), BayesianRidge()],
              'regressor__model__n_estimators': range(1, 21)}
results, bagging_regressor = my_grid_search_cv(model=BaggingRegressor(), X=df,␣
 ↪y=targets, results=results,
                                             param_grid=param_grid,␣
 ↪poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
                                                            ␣
 ↪ColumnTransformer(transformers=[('median_standard_scaler',
                                                                          ␣
 ↪      Pipeline(steps=[('imputer',
                                                                          ␣
 ↪                     SimpleImputer(strategy='median')),
                                                                          ␣
 ↪                    ('transformer',
                                                                          ␣
 ↪                     StandardScaler())]),
                                                                          ␣
 ↪      ['INDICE_CRIMEN',
                                                                          ␣
 ↪       'PCT_ZONA_RESIDENCIAL',
                                                                          ␣
 ↪       'PCT_ZONA_INDUSTRIAL',
                                                                          ␣
 ↪       'OXIDO_NITROSO_PPM',
                                                                          ␣
 ↪       'N_HABITACIONES_MEDIO',
```

```
                                                                                  ␣
↪        'PCT_CASAS_40S',
                                                                                  ␣
↪        'DIS',…
                                                                                  ␣
↪        'RATIO_PROFESORES',
                                                                                  ␣
↪        'PCT_NEGRA',
                                                                                  ␣
↪        'PCT_CLASE_BAJA']),
                                                                                  ␣
↪      ('mode_one_hot_encoder',
                                                                                  ␣
↪       Pipeline(steps=[('imputer',
                                                                                  ␣
↪                        SimpleImputer(strategy='most_frequent')),
                                                                                  ␣
↪                       ('transformer',
                                                                                  ␣
↪                        OneHotEncoder(drop='first'))]),
                                                                                  ␣
↪       ['RIO_CHARLES'])])),
                                                        ('polynomial',
                                                        ␣
↪PolynomialFeatures(degree=1)),
                                                        ('model',
                                                        ␣
↪BaggingRegressor(base_estimator=DecisionTreeRegressor(),
                                                                          ␣
↪n_estimators=12))]),
                          transformer=MinMaxScaler())
The model BaggingRegressor() has a prediction error of +-2.9822 dollars.
```

[52]:

```
                       NAME TYPE POLY_DEGREE   SCORE  \
0        LinearRegression()   ML           1 -3.9665
1                   Lasso()   ML           1 -5.8531
2                   Ridge()   ML           1 -3.9507
3              ElasticNet()   ML           1 -4.6790
4   DecisionTreeRegressor()   ML           1 -3.5071
5                     SVR()   ML           1 -3.0232
6     KNeighborsRegressor()   ML           1 -3.3853
7           BayesianRidge()   ML           1 -3.9060
8        BaggingRegressor()   ML           1 -2.9822

                                           ESTIMATOR
0  TransformedTargetRegressor(regressor=Pipeline(…
```

```
1    TransformedTargetRegressor(regressor=Pipeline(…
2    TransformedTargetRegressor(regressor=Pipeline(…
3    TransformedTargetRegressor(regressor=Pipeline(…
4    TransformedTargetRegressor(regressor=Pipeline(…
5    TransformedTargetRegressor(regressor=Pipeline(…
6    TransformedTargetRegressor(regressor=Pipeline(…
7    TransformedTargetRegressor(regressor=Pipeline(…
8    TransformedTargetRegressor(regressor=Pipeline(…
```

6.9.2 Boosting

[53]:
```
param_grid = {'regressor__model__base_estimator': [LinearRegression(), Lasso(),␣
 ↪Ridge(), ElasticNet(), DecisionTreeRegressor(),
                                                   SVR(),␣
 ↪KNeighborsRegressor(), BayesianRidge()],
              'regressor__model__n_estimators': range(1, 21)}
results, bagging_regressor = my_grid_search_cv(model=AdaBoostRegressor(), X=df,␣
 ↪y=targets, results=results,
                                               param_grid=param_grid,␣
 ↪poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
                                                    ␣
↪ColumnTransformer(transformers=[('median_standard_scaler',
                                                                               ␣
↪      Pipeline(steps=[('imputer',
                                                                               ␣
↪                       SimpleImputer(strategy='median')),
                                                                               ␣
↪                      ('transformer',
                                                                               ␣
↪                       StandardScaler())]),
                                                                               ␣
↪      ['INDICE_CRIMEN',
                                                                               ␣
↪       'PCT_ZONA_RESIDENCIAL',
                                                                               ␣
↪       'PCT_ZONA_INDUSTRIAL',
                                                                               ␣
↪       'OXIDO_NITROSO_PPM',
                                                                               ␣
↪       'N_HABITACIONES_MEDIO',
                                                                               ␣
↪       'PCT_CASAS_40S',
                                                                               ␣
↪       'DIS',…
```

```
↪       'RATIO_PROFESORES',
↪       'PCT_NEGRA',
↪       'PCT_CLASE_BAJA']),
↪     ('mode_one_hot_encoder',
↪      Pipeline(steps=[('imputer',
↪                       SimpleImputer(strategy='most_frequent')),
↪                      ('transformer',
↪                       OneHotEncoder(drop='first'))]),
↪      ['RIO_CHARLES'])])),
                                                        ('polynomial',
↪PolynomialFeatures(degree=1)),
                                                        ('model',
↪AdaBoostRegressor(base_estimator=DecisionTreeRegressor(),
↪n_estimators=17))]),
                           transformer=MinMaxScaler())
The model AdaBoostRegressor() has a prediction error of +-2.9533 dollars.
```

[53]:

```
                        NAME TYPE POLY_DEGREE   SCORE  \
0         LinearRegression()   ML           1 -3.9665
1                    Lasso()   ML           1 -5.8531
2                    Ridge()   ML           1 -3.9507
3               ElasticNet()   ML           1 -4.6790
4    DecisionTreeRegressor()   ML           1 -3.5071
5                      SVR()   ML           1 -3.0232
6      KNeighborsRegressor()   ML           1 -3.3853
7            BayesianRidge()   ML           1 -3.9060
8         BaggingRegressor()   ML           1 -2.9822
9        AdaBoostRegressor()   ML           1 -2.9533

                                           ESTIMATOR
0  TransformedTargetRegressor(regressor=Pipeline(…
1  TransformedTargetRegressor(regressor=Pipeline(…
2  TransformedTargetRegressor(regressor=Pipeline(…
3  TransformedTargetRegressor(regressor=Pipeline(…
```

```
4  TransformedTargetRegressor(regressor=Pipeline(…
5  TransformedTargetRegressor(regressor=Pipeline(…
6  TransformedTargetRegressor(regressor=Pipeline(…
7  TransformedTargetRegressor(regressor=Pipeline(…
8  TransformedTargetRegressor(regressor=Pipeline(…
9  TransformedTargetRegressor(regressor=Pipeline(…
```

6.9.3 Gradient Boosting (GBRT)

[54]:
```
param_grid = {'regressor__model__n_estimators': range(1, 21)}
results, gradient_boosting_regressor =␣
 ↪my_grid_search_cv(model=GradientBoostingRegressor(), X=df, y=targets,
                                                   results=results,␣
 ↪param_grid=param_grid, poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
                                                                        ␣
 ↪ColumnTransformer(transformers=[('median_standard_scaler',
                                                                                  ␣
 ↪       Pipeline(steps=[('imputer',
                                                                                  ␣
 ↪                        SimpleImputer(strategy='median')),
                                                                                  ␣
 ↪                       ('transformer',
                                                                                  ␣
 ↪                        StandardScaler())]),
                                                                                  ␣
 ↪       ['INDICE_CRIMEN',
                                                                                  ␣
 ↪        'PCT_ZONA_RESIDENCIAL',
                                                                                  ␣
 ↪        'PCT_ZONA_INDUSTRIAL',
                                                                                  ␣
 ↪        'OXIDO_NITROSO_PPM',
                                                                                  ␣
 ↪        'N_HABITACIONES_MEDIO',
                                                                                  ␣
 ↪        'PCT_CASAS_40S',
                                                                                  ␣
 ↪        'DIS',
                                                                                  ␣
 ↪        'DIS_AUTOPISTAS',
                                                                                  ␣
 ↪        'CARGA_FISCAL',
                                                                                  ␣
 ↪        'RATIO_PROFESORES',
```

```
↪      'PCT_NEGRA',
↪      'PCT_CLASE_BAJA']),
↪     ('mode_one_hot_encoder',
↪      Pipeline(steps=[('imputer',
↪                        SimpleImputer(strategy='most_frequent')),
↪                       ('transformer',
↪                        OneHotEncoder(drop='first'))]),
↪      ['RIO_CHARLES'])])),
                                                        ('polynomial',
↪PolynomialFeatures(degree=1)),
                                                        ('model',
↪GradientBoostingRegressor(n_estimators=20))]),
                           transformer=MinMaxScaler())
```

```
The model GradientBoostingRegressor() has a prediction error of +-3.3612
dollars.
```

[54]:

```
                           NAME TYPE POLY_DEGREE   SCORE  \
0            LinearRegression()   ML           1 -3.9665
1                       Lasso()   ML           1 -5.8531
2                       Ridge()   ML           1 -3.9507
3                  ElasticNet()   ML           1 -4.6790
4       DecisionTreeRegressor()   ML           1 -3.5071
5                         SVR()   ML           1 -3.0232
6         KNeighborsRegressor()   ML           1 -3.3853
7               BayesianRidge()   ML           1 -3.9060
8            BaggingRegressor()   ML           1 -2.9822
9           AdaBoostRegressor()   ML           1 -2.9533
10  GradientBoostingRegressor()   ML           1 -3.3612

                                            ESTIMATOR
0   TransformedTargetRegressor(regressor=Pipeline(…
1   TransformedTargetRegressor(regressor=Pipeline(…
2   TransformedTargetRegressor(regressor=Pipeline(…
3   TransformedTargetRegressor(regressor=Pipeline(…
4   TransformedTargetRegressor(regressor=Pipeline(…
5   TransformedTargetRegressor(regressor=Pipeline(…
```

```
6   TransformedTargetRegressor(regressor=Pipeline(…
7   TransformedTargetRegressor(regressor=Pipeline(…
8   TransformedTargetRegressor(regressor=Pipeline(…
9   TransformedTargetRegressor(regressor=Pipeline(…
10  TransformedTargetRegressor(regressor=Pipeline(…
```

6.9.4 Random Forests

[55]:
```
param_grid = {'regressor__model__n_estimators': range(1, 21)}
results, random_forest_regressor =␣
↪my_grid_search_cv(model=RandomForestRegressor(), X=df, y=targets,
                                                    results=results,␣
↪param_grid=param_grid, poly_degrees=[1])
results
```

```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
                                                                  ␣
↪ColumnTransformer(transformers=[('median_standard_scaler',
                                                                                    ␣
↪      Pipeline(steps=[('imputer',
                                                                                    ␣
↪                        SimpleImputer(strategy='median')),
                                                                                    ␣
↪                       ('transformer',
                                                                                    ␣
↪                        StandardScaler())]),
                                                                                    ␣
↪      ['INDICE_CRIMEN',
                                                                                    ␣
↪       'PCT_ZONA_RESIDENCIAL',
                                                                                    ␣
↪       'PCT_ZONA_INDUSTRIAL',
                                                                                    ␣
↪       'OXIDO_NITROSO_PPM',
                                                                                    ␣
↪       'N_HABITACIONES_MEDIO',
                                                                                    ␣
↪       'PCT_CASAS_40S',
                                                                                    ␣
↪       'DIS',
                                                                                    ␣
↪       'DIS_AUTOPISTAS',
                                                                                    ␣
↪       'CARGA_FISCAL',
                                                                                    ␣
↪       'RATIO_PROFESORES',
```

```
↪       'PCT_NEGRA',
↪       'PCT_CLASE_BAJA']),
↪     ('mode_one_hot_encoder',
↪      Pipeline(steps=[('imputer',
↪                       SimpleImputer(strategy='most_frequent')),
↪                      ('transformer',
↪                       OneHotEncoder(drop='first'))]),
↪      ['RIO_CHARLES'])])),
                                                      ('polynomial',
↪PolynomialFeatures(degree=1)),
                                                      ('model',
↪RandomForestRegressor(n_estimators=19))]),
                                   transformer=MinMaxScaler())
The model RandomForestRegressor() has a prediction error of +-2.9834 dollars.
```

[55]:

```
                         NAME TYPE POLY_DEGREE   SCORE  \
0          LinearRegression()   ML           1 -3.9665
1                     Lasso()   ML           1 -5.8531
2                     Ridge()   ML           1 -3.9507
3                ElasticNet()   ML           1 -4.6790
4     DecisionTreeRegressor()   ML           1 -3.5071
5                       SVR()   ML           1 -3.0232
6       KNeighborsRegressor()   ML           1 -3.3853
7             BayesianRidge()   ML           1 -3.9060
8          BaggingRegressor()   ML           1 -2.9822
9         AdaBoostRegressor()   ML           1 -2.9533
10  GradientBoostingRegressor()  ML           1 -3.3612
11      RandomForestRegressor()  ML           1 -2.9834

                                            ESTIMATOR
0   TransformedTargetRegressor(regressor=Pipeline(…
1   TransformedTargetRegressor(regressor=Pipeline(…
2   TransformedTargetRegressor(regressor=Pipeline(…
3   TransformedTargetRegressor(regressor=Pipeline(…
4   TransformedTargetRegressor(regressor=Pipeline(…
5   TransformedTargetRegressor(regressor=Pipeline(…
```

```
6    TransformedTargetRegressor(regressor=Pipeline(…
7    TransformedTargetRegressor(regressor=Pipeline(…
8    TransformedTargetRegressor(regressor=Pipeline(…
9    TransformedTargetRegressor(regressor=Pipeline(…
10   TransformedTargetRegressor(regressor=Pipeline(…
11   TransformedTargetRegressor(regressor=Pipeline(…
```

6.10 Learning curves

[56]:
```
train_sizes, train_scores, test_scores = learning_curve(ols, df, targets, cv=2,␣
 ↪n_jobs=-1, scoring='neg_mean_absolute_error',
                                                         train_sizes=np.
 ↪linspace(0.01, 1., 10))
train_sizes.shape, train_scores.shape, test_scores.shape
```

[56]:
```
((10,), (10, 2), (10, 2))
```

[57]:
```
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_mean
```

[57]:
```
array([-3.55271368e-15, -1.12693331e+00, -1.58588623e+00, -1.48640683e+00,
       -1.98081975e+00, -2.66137734e+00, -3.05280312e+00, -3.04847919e+00,
       -2.94260991e+00, -2.92393146e+00])
```

[58]:
```
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_mean
```

[58]:
```
array([        nan,         nan,         nan,         nan,         nan,
               nan, -9.49014422, -8.56668616, -8.05061258, -7.16271464])
```

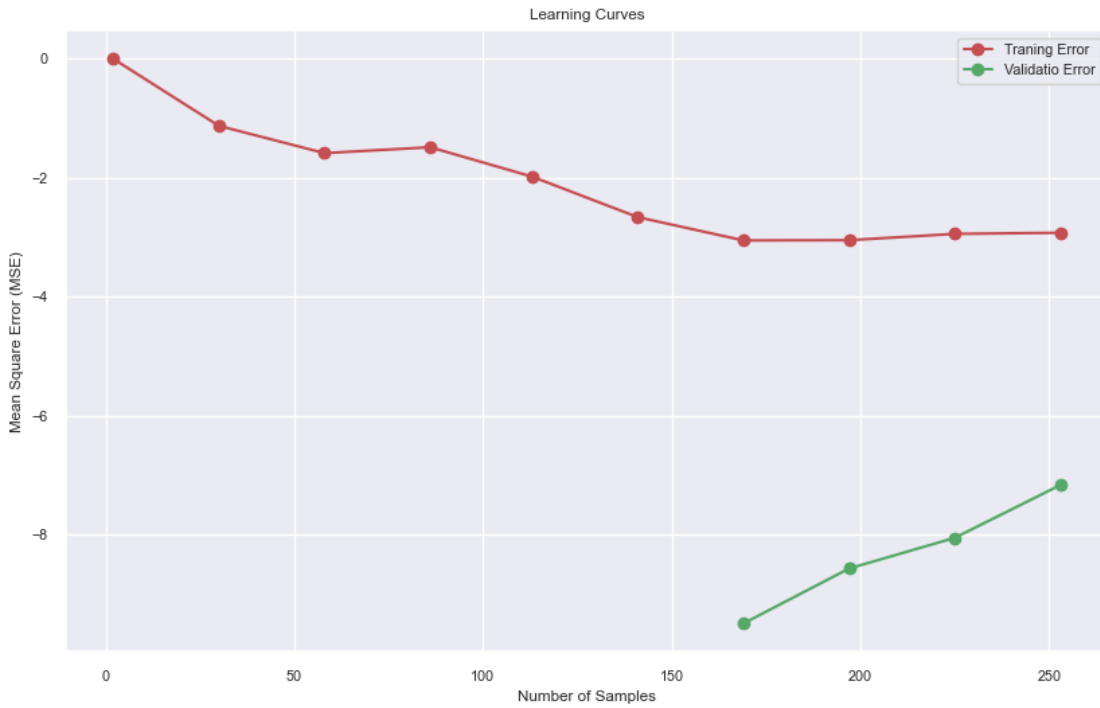
[59]:
```
train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)

plt.rc('figure', figsize=(10, 6))
plt.plot(train_sizes, train_scores_mean, 'o-',
         color='r', label='Traning Error')
plt.plot(train_sizes, test_scores_mean, 'o-',
         color='g', label='Validatio Error')

plt.title('Learning Curves')
plt.xlabel('Number of Samples')
plt.ylabel('Mean Square Error (MSE)')

plt.legend()
```

[59]:
```
<matplotlib.legend.Legend at 0x2171565d040>
```



Shortcode for all estimators.

```
[60]: results_model_key = results.copy()
results_model_key.set_index('NAME', inplace=True)


@interact(model=results_model_key.index)
def _(model):
    plt.rc('figure', figsize=(10, 6))

    train_sizes, train_scores, test_scores = learning_curve(results_model_key.
 ↪loc[model].ESTIMATOR, df, targets, cv=2,
                                                              n_jobs=-1,␣
 ↪train_sizes=np.linspace(0.01, 1., 10),
                                                             ␣
 ↪scoring='neg_mean_absolute_error')

    train_scores_mean = np.mean(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)

    plt.plot(train_sizes, train_scores_mean, 'o-',
             color='r', label='Traning error')
    plt.plot(train_sizes, test_scores_mean, 'o-',
             color='g', label='Validatio error')
```

```
    plt.title(f'Learning Curves: {model}')
    plt.xlabel('Number of Samples')
    plt.ylabel('Mean Square Error (MSE)')

    plt.legend()
    plt.show()
```

```
interactive(children=(Dropdown(description='model',␣
↪options=('LinearRegression()', 'Lasso()', 'Ridge()', 'Elas…
```

6.11 Validation curves

Los hiperparámetros ya fueron seleccionados en cada uno de los algoritmos mediante `GridSearchCV`.

7 Deep Learning

Cuda version.

[61]:
```
print(f'Tensorflow version: {tf.__version__}')
print(f"Cuda version: {build.build_info['cuda_version']}")
print(f"Cudnn version: {build.build_info['cudnn_version']}")
```

```
Tensorflow version: 2.6.0
Cuda version: 64_113
Cudnn version: 64_8
```

Enable GPU

- The environment must be created with a version of Python compatible with the operation of Tensorflow and its use of the GPU (https://www.tensorflow.org/install/pip#virtual-environment-install).
- Before installing tensorflow-gpu you must install CUDA Toolkit and cuDNN from official NVIDIA site.
- Anaconda must be restarted after installing tensorflow-gpu.

[62]:
```
# !conda install -y tensorflow-gpu keras-gpu

if len(tf.config.list_physical_devices('GPU')) == 0:
    raise SystemExit('Restart Anaconda to activate the GPU.')
else:
    print('GPU activated.')
```

```
GPU activated.
```

Available hardware.

[63]:
```
tf.config.get_visible_devices()
```

[63]:
```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
 PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Available hardware details.

[64]:
```python
device_lib.list_local_devices()
```

[64]:
```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 3114884471351915123,
 name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 2236245607
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 3384425887881826203
 physical_device_desc: "device: 0, name: NVIDIA GeForce GTX 1650, pci bus id:
0000:01:00.0, compute capability: 7.5"]
```

7.1 Tensorboard

[65]:
```python
# !rm -rf logs/
```

[86]:
```python
%load_ext tensorboard
%tensorboard - -logdir logs
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard

ERROR: Failed to launch TensorBoard (exited with 2).
Contents of stderr:
usage: tensorboard [-h] [--helpfull] [--logdir PATH] [--logdir_spec PATH_SPEC]
                   [--host ADDR] [--bind_all] [--port PORT]
                   [--reuse_port BOOL] [--load_fast {false,auto,true}]
                   [--extra_data_server_flags EXTRA_DATA_SERVER_FLAGS]
                   [--grpc_creds_type {local,ssl,ssl_dev}]
                   [--grpc_data_provider PORT] [--purge_orphaned_data BOOL]
                   [--db URI] [--db_import] [--inspect] [--version_tb]
                   [--tag TAG] [--event_file PATH] [--path_prefix PATH]
                   [--window_title TEXT] [--max_reload_threads COUNT]
                   [--reload_interval SECONDS] [--reload_task TYPE]
                   [--reload_multifile BOOL]
                   [--reload_multifile_inactive_secs SECONDS]
```

```
                    [--generic_data TYPE]
                    [--samples_per_plugin SAMPLES_PER_PLUGIN]
                    [--whatif-use-unsafe-custom-prediction␣
↪YOUR_CUSTOM_PREDICT_FUNCTION.py]
                    [--whatif-data-dir PATH]
                    {serve,dev} …
tensorboard: error: argument {serve,dev}: invalid choice: '-' (choose from␣
↪'serve', 'dev')
```

[67]:
```
tensorboard = TensorBoard(os.path.join("logs", datetime.now().
↪strftime("%Y%m%d-%H%M%S")), histogram_freq=1,
                          write_graph=True, write_images=True,␣
↪update_freq='epoch', profile_batch=2, embeddings_freq=1)
```

7.2 Preprocess

Data for neural networks is mostly normalized data, not standardized.

[68]:
```
preprocessor_resume = pd.DataFrame(data=df.dtypes, columns=['TYPE'])
preprocessor_resume['VALUES'] = preprocessor_resume.apply(lambda x: df[x.name].
↪unique(),
                                                          axis=1)
preprocessor_resume['VALUES_LEN'] = preprocessor_resume.apply(lambda x:␣
↪len(df[x.name].unique()),
                                                              axis=1)
preprocessor_resume[['IMPUTER', 'TRANSFORMER', 'STATE']] = 'UNKNOWN'

#␣
↪-----------------------------------------------------------------------------------------
preprocessor_resume.loc[[], ['IMPUTER']] = 'Mean'
preprocessor_resume.loc[['RIO_CHARLES'], ['IMPUTER']] = 'Mode'
preprocessor_resume.loc[['INDICE_CRIMEN', 'PCT_ZONA_RESIDENCIAL',␣
↪'PCT_ZONA_INDUSTRIAL', 'OXIDO_NITROSO_PPM',
                         'N_HABITACIONES_MEDIO', 'PCT_CASAS_40S', 'DIS',␣
↪'RATIO_PROFESORES', 'PCT_NEGRA',
                         'PCT_CLASE_BAJA', 'CARGA_FISCAL', 'DIS_AUTOPISTAS'],␣
↪['IMPUTER']] = 'Median'
#␣
↪-----------------------------------------------------------------------------------------
preprocessor_resume.loc[['INDICE_CRIMEN', 'PCT_ZONA_RESIDENCIAL',␣
↪'PCT_ZONA_INDUSTRIAL', 'OXIDO_NITROSO_PPM',
                         'N_HABITACIONES_MEDIO', 'PCT_CASAS_40S', 'DIS',␣
↪'RATIO_PROFESORES', 'PCT_NEGRA',
                         'PCT_CLASE_BAJA', 'DIS_AUTOPISTAS', 'CARGA_FISCAL'],␣
↪['TRANSFORMER']] = 'MIX_MAX_SCALER'
preprocessor_resume.loc[['RIO_CHARLES'], ['TRANSFORMER']] = 'ONE_HOT_ENCODER'
preprocessor_resume.loc[[], ['TRANSFORMER']] = 'ORDINAL_ENCODER'
```

```
#␣
↪------------------------------------------------------------------------------------------------
median_mix_max_scaler_transformer = Pipeline(steps=[('imputer',␣
↪SimpleImputer(strategy='median')),
                                                     ('transformer',␣
↪MinMaxScaler())])
mode_one_hot_encoder_transformer = Pipeline(steps=[('imputer',␣
↪SimpleImputer(strategy='most_frequent')),
                                                    ('transformer',␣
↪OneHotEncoder(sparse=True, drop='first'))])

median_mix_max_scaler_features = preprocessor_resume.query(
    'IMPUTER == "Median" and TRANSFORMER == "MIX_MAX_SCALER"').index.to_list()
mode_one_hot_encoder_features = preprocessor_resume.query(
    'IMPUTER == "Mode" and TRANSFORMER == "ONE_HOT_ENCODER"').index.to_list()

preprocessor_resume.loc[median_mix_max_scaler_features +
                        mode_one_hot_encoder_features, ['STATE']] = 'OK'

preprocessor = ColumnTransformer(transformers=[('median_mix_max_scaler',
                                               ␣
↪median_mix_max_scaler_transformer, median_mix_max_scaler_features),
                                               ('mode_one_hot_encoder',
                                               ␣
↪mode_one_hot_encoder_transformer, mode_one_hot_encoder_features)])
#␣
↪------------------------------------------------------------------------------------------------

display(preprocessor)
display(preprocessor_resume)

X = preprocessor.fit_transform(X=df)
print(f'Dimensiones de los datos: {X.shape}.')
del X
```

```
ColumnTransformer(transformers=[('median_mix_max_scaler',
                                 Pipeline(steps=[('imputer',
                                                 ␣
↪SimpleImputer(strategy='median')),
                                                 ('transformer',
                                                  MinMaxScaler())]),
                                 ['INDICE_CRIMEN', 'PCT_ZONA_RESIDENCIAL',
                                  'PCT_ZONA_INDUSTRIAL', 'OXIDO_NITROSO_PPM',
                                  'N_HABITACIONES_MEDIO', 'PCT_CASAS_40S',
                                  'DIS', 'DIS_AUTOPISTAS', 'CARGA_FISCAL',
                                  'RATIO_PROFESORES', 'PCT_NEGRA',
                                  'PCT_CLASE_BAJA']),
```

```
                                    ('mode_one_hot_encoder',
                                     Pipeline(steps=[('imputer',
                                                                                ␣
↪SimpleImputer(strategy='most_frequent')),
                                                      ('transformer',
                                                       OneHotEncoder(drop='first'))]),
                                     ['RIO_CHARLES'])])
```

```
                           TYPE  \
CIUDAD                   object
LON                     float64
LAT                     float64
VALOR_MEDIANO           float64
INDICE_CRIMEN           float64
PCT_ZONA_RESIDENCIAL    float64
PCT_ZONA_INDUSTRIAL     float64
RIO_CHARLES               int64
OXIDO_NITROSO_PPM       float64
N_HABITACIONES_MEDIO    float64
PCT_CASAS_40S           float64
DIS                     float64
DIS_AUTOPISTAS            int64
CARGA_FISCAL              int64
RATIO_PROFESORES        float64
PCT_NEGRA               float64
PCT_CLASE_BAJA          float64

                                                                   VALUES  \
CIUDAD                  [Nahant, Swampscott, Marblehead, Salem, Lynn, …
LON                     [-70.955, -70.95, -70.936, -70.928, -70.922, -…
LAT                     [42.255, 42.2875, 42.283, 42.293, 42.298, 42.3…
VALOR_MEDIANO           [24.0, 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 22…
INDICE_CRIMEN           [0.00632, 0.02731, 0.02729, 0.03237, 0.06905, …
PCT_ZONA_RESIDENCIAL    [18.0, 0.0, 12.5, 75.0, 21.0, 90.0, 85.0, 100…
PCT_ZONA_INDUSTRIAL     [2.31, 7.07, 2.18, 7.87, 8.14, 5.96, 2.95, 6.9…
RIO_CHARLES                                                        [0, 1]
OXIDO_NITROSO_PPM       [0.5379999999999999, 0.469, 0.458, 0.524, 0.49…
N_HABITACIONES_MEDIO    [6.575, 6.421, 7.185, 6.997999999999998, 7.147…
PCT_CASAS_40S           [65.2, 78.9, 61.1, 45.8, 54.2, 58.7, 66.6, 96…
DIS                     [4.09, 4.9671, 6.0622, 5.5605, 5.9505, 6.0821,…
DIS_AUTOPISTAS                               [1, 2, 3, 5, 4, 8, 6, 7, 24]
CARGA_FISCAL            [296, 242, 222, 311, 307, 279, 252, 233, 243, …
RATIO_PROFESORES        [15.3, 17.8, 18.7, 15.2, 21.0, 19.2, 18.3, 17…
PCT_NEGRA               [396.9, 392.83, 394.63, 394.12, 395.6, 386.63,…
PCT_CLASE_BAJA          [4.98, 9.14, 4.03, 2.94, 5.33, 5.21, 12.43, 19…

                      VALUES_LEN  IMPUTER      TRANSFORMER    STATE
CIUDAD                        92  UNKNOWN          UNKNOWN  UNKNOWN
```

```
LON                          375  UNKNOWN           UNKNOWN  UNKNOWN
LAT                          376  UNKNOWN           UNKNOWN  UNKNOWN
VALOR_MEDIANO                228  UNKNOWN           UNKNOWN  UNKNOWN
INDICE_CRIMEN                504   Median    MIX_MAX_SCALER       OK
PCT_ZONA_RESIDENCIAL          26   Median    MIX_MAX_SCALER       OK
PCT_ZONA_INDUSTRIAL           76   Median    MIX_MAX_SCALER       OK
RIO_CHARLES                    2     Mode   ONE_HOT_ENCODER       OK
OXIDO_NITROSO_PPM             81   Median    MIX_MAX_SCALER       OK
N_HABITACIONES_MEDIO         446   Median    MIX_MAX_SCALER       OK
PCT_CASAS_40S                356   Median    MIX_MAX_SCALER       OK
DIS                          412   Median    MIX_MAX_SCALER       OK
DIS_AUTOPISTAS                 9   Median    MIX_MAX_SCALER       OK
CARGA_FISCAL                  66   Median    MIX_MAX_SCALER       OK
RATIO_PROFESORES              46   Median    MIX_MAX_SCALER       OK
PCT_NEGRA                    357   Median    MIX_MAX_SCALER       OK
PCT_CLASE_BAJA               455   Median    MIX_MAX_SCALER       OK

Dimensiones de los datos: (506, 13).
```

7.3 Keras - Multilayer Perceptron (MLP)

Training and test data are generated.

[69]:
```
X_train, X_test, y_train, y_test = train_test_split(df, targets, test_size=0.2,
                                                    shuffle=True)
```

[70]:
```
def build_keras_model():
    with tf.device('/GPU:0'):
        # Clear backend
        backend.clear_session()

        keras_model = Sequential([Dense(units=20, activation='relu',␣
 ↪kernel_constraint=maxnorm(max_value=3), input_dim=13),
                                  Dropout(rate=0.2),

                                  Dense(units=20, activation='relu',
                                        kernel_constraint=maxnorm(max_value=3)),
                                  Dropout(rate=0.2),

                                  Dense(units=1, activation='relu')])

        keras_model.compile(optimizer=RMSprop(),  # optimizer
                            loss='mse',           # función de pérdida o coste
                            metrics=['mae'])      # Metrics to observe the␣
 ↪evolution of the model training

        display(keras_model.summary())
```

```
        return keras_model
```

[71]:
```
target_transformer = MinMaxScaler(feature_range=(0, 1))
y_test_re = np.reshape(y_test.values, (-1, 1))

keras_regressor = KerasRegressor(build_fn=build_keras_model, batch_size=64,␣
 ↪epochs=100, verbose=1,
                                 shuffle=True, callbacks=[tensorboard],
                                 validation_data=(preprocessor.
 ↪fit_transform(X=X_test),
                                                  target_transformer.
 ↪fit_transform(y_test_re)))

pipe = Pipeline(steps=[('preprocessor', preprocessor),
                       ('model', keras_regressor)])

keras_estimator = TransformedTargetRegressor(regressor=pipe,
                                             transformer=target_transformer)

keras_estimator.fit(X_train, y_train)
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 20)                280

_________________________________________________________________
dropout (Dropout)            (None, 20)                0

_________________________________________________________________
dense_1 (Dense)              (None, 20)                420

_________________________________________________________________
dropout_1 (Dropout)          (None, 20)                0

_________________________________________________________________
dense_2 (Dense)              (None, 1)                 21
=================================================================
Total params: 721
Trainable params: 721
Non-trainable params: 0

_________________________________________________________________

None

Epoch 1/100
7/7 [==============================] - 1s 53ms/step - loss: 0.1271 - mae: 0.2857
- val_loss: 0.1084 - val_mae: 0.2324
Epoch 2/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0952 - mae: 0.2347
- val_loss: 0.0881 - val_mae: 0.1984
```

```
Epoch 3/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0833 - mae: 0.2205
- val_loss: 0.0691 - val_mae: 0.1685
Epoch 4/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0819 - mae: 0.2199
- val_loss: 0.0679 - val_mae: 0.1650
Epoch 5/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0715 - mae: 0.1974
- val_loss: 0.0588 - val_mae: 0.1563
Epoch 6/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0692 - mae: 0.2012
- val_loss: 0.0559 - val_mae: 0.1487
Epoch 7/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0603 - mae: 0.1811
- val_loss: 0.0481 - val_mae: 0.1405
Epoch 8/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0511 - mae: 0.1708
- val_loss: 0.0414 - val_mae: 0.1348
Epoch 9/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0513 - mae: 0.1668
- val_loss: 0.0393 - val_mae: 0.1281
Epoch 10/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0451 - mae: 0.1614
- val_loss: 0.0372 - val_mae: 0.1234
Epoch 11/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0477 - mae: 0.1576
- val_loss: 0.0351 - val_mae: 0.1190
Epoch 12/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0427 - mae: 0.1498
- val_loss: 0.0326 - val_mae: 0.1150
Epoch 13/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0395 - mae: 0.1446
- val_loss: 0.0309 - val_mae: 0.1098
Epoch 14/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0392 - mae: 0.1392
- val_loss: 0.0276 - val_mae: 0.1076
Epoch 15/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0342 - mae: 0.1363
- val_loss: 0.0268 - val_mae: 0.1040
Epoch 16/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0317 - mae: 0.1301
- val_loss: 0.0273 - val_mae: 0.1028
Epoch 17/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0336 - mae: 0.1326
- val_loss: 0.0262 - val_mae: 0.1008
Epoch 18/100
7/7 [==============================] - 0s 6ms/step - loss: 0.0284 - mae: 0.1272
- val_loss: 0.0246 - val_mae: 0.0987
```

```
Epoch 19/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0304 - mae: 0.1272
- val_loss: 0.0233 - val_mae: 0.1013
Epoch 20/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0277 - mae: 0.1204
- val_loss: 0.0222 - val_mae: 0.1007
Epoch 21/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0265 - mae: 0.1191
- val_loss: 0.0217 - val_mae: 0.0968
Epoch 22/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0258 - mae: 0.1126
- val_loss: 0.0209 - val_mae: 0.0945
Epoch 23/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0246 - mae: 0.1159
- val_loss: 0.0203 - val_mae: 0.0978
Epoch 24/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0236 - mae: 0.1124
- val_loss: 0.0202 - val_mae: 0.0937
Epoch 25/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0267 - mae: 0.1174
- val_loss: 0.0195 - val_mae: 0.0928
Epoch 26/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0232 - mae: 0.1111
- val_loss: 0.0183 - val_mae: 0.0977
Epoch 27/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0230 - mae: 0.1127
- val_loss: 0.0179 - val_mae: 0.0932
Epoch 28/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0210 - mae: 0.1037
- val_loss: 0.0178 - val_mae: 0.0920
Epoch 29/100
7/7 [==============================] - 0s 6ms/step - loss: 0.0206 - mae: 0.1066
- val_loss: 0.0175 - val_mae: 0.0901
Epoch 30/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0234 - mae: 0.1124
- val_loss: 0.0169 - val_mae: 0.0888
Epoch 31/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0227 - mae: 0.1063
- val_loss: 0.0174 - val_mae: 0.0855
Epoch 32/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0181 - mae: 0.0977
- val_loss: 0.0158 - val_mae: 0.0862
Epoch 33/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0200 - mae: 0.1037
- val_loss: 0.0158 - val_mae: 0.0841
Epoch 34/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0188 - mae: 0.0950
- val_loss: 0.0151 - val_mae: 0.0878
```

```
Epoch 35/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0196 - mae: 0.1063
- val_loss: 0.0147 - val_mae: 0.0843
Epoch 36/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0193 - mae: 0.1008
- val_loss: 0.0142 - val_mae: 0.0876
Epoch 37/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0180 - mae: 0.1001
- val_loss: 0.0141 - val_mae: 0.0865
Epoch 38/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0211 - mae: 0.1018
- val_loss: 0.0146 - val_mae: 0.0801
Epoch 39/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0199 - mae: 0.1006
- val_loss: 0.0149 - val_mae: 0.0786
Epoch 40/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0188 - mae: 0.0973
- val_loss: 0.0141 - val_mae: 0.0785
Epoch 41/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0177 - mae: 0.0961
- val_loss: 0.0142 - val_mae: 0.0770
Epoch 42/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0173 - mae: 0.0950
- val_loss: 0.0139 - val_mae: 0.0772
Epoch 43/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0172 - mae: 0.0945
- val_loss: 0.0146 - val_mae: 0.0769
Epoch 44/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0158 - mae: 0.0909
- val_loss: 0.0132 - val_mae: 0.0774
Epoch 45/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0156 - mae: 0.0903
- val_loss: 0.0126 - val_mae: 0.0763
Epoch 46/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0187 - mae: 0.0968
- val_loss: 0.0132 - val_mae: 0.0749
Epoch 47/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0131 - mae: 0.0835
- val_loss: 0.0113 - val_mae: 0.0751
Epoch 48/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0169 - mae: 0.0940
- val_loss: 0.0114 - val_mae: 0.0745
Epoch 49/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0144 - mae: 0.0884
- val_loss: 0.0111 - val_mae: 0.0753
Epoch 50/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0154 - mae: 0.0908
- val_loss: 0.0132 - val_mae: 0.0745
```

```
Epoch 51/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0139 - mae: 0.0878
- val_loss: 0.0122 - val_mae: 0.0715
Epoch 52/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0154 - mae: 0.0872
- val_loss: 0.0118 - val_mae: 0.0713
Epoch 53/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0152 - mae: 0.0869
- val_loss: 0.0101 - val_mae: 0.0703
Epoch 54/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0160 - mae: 0.0892
- val_loss: 0.0107 - val_mae: 0.0702
Epoch 55/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0148 - mae: 0.0857
- val_loss: 0.0110 - val_mae: 0.0699
Epoch 56/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0151 - mae: 0.0886
- val_loss: 0.0108 - val_mae: 0.0701
Epoch 57/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0148 - mae: 0.0881
- val_loss: 0.0103 - val_mae: 0.0688
Epoch 58/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0163 - mae: 0.0905
- val_loss: 0.0108 - val_mae: 0.0685
Epoch 59/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0142 - mae: 0.0844
- val_loss: 0.0106 - val_mae: 0.0687
Epoch 60/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0126 - mae: 0.0798
- val_loss: 0.0104 - val_mae: 0.0680
Epoch 61/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0116 - mae: 0.0785
- val_loss: 0.0122 - val_mae: 0.0721
Epoch 62/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0147 - mae: 0.0852
- val_loss: 0.0115 - val_mae: 0.0684
Epoch 63/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0135 - mae: 0.0838
- val_loss: 0.0116 - val_mae: 0.0683
Epoch 64/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0135 - mae: 0.0821
- val_loss: 0.0100 - val_mae: 0.0658
Epoch 65/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0136 - mae: 0.0819
- val_loss: 0.0102 - val_mae: 0.0663
Epoch 66/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0106 - mae: 0.0771
- val_loss: 0.0109 - val_mae: 0.0672
```

```
Epoch 67/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0129 - mae: 0.0825
- val_loss: 0.0103 - val_mae: 0.0666
Epoch 68/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0125 - mae: 0.0826
- val_loss: 0.0102 - val_mae: 0.0663
Epoch 69/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0133 - mae: 0.0834
- val_loss: 0.0096 - val_mae: 0.0655
Epoch 70/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0131 - mae: 0.0834
- val_loss: 0.0107 - val_mae: 0.0692
Epoch 71/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0108 - mae: 0.0767
- val_loss: 0.0110 - val_mae: 0.0693
Epoch 72/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0132 - mae: 0.0797
- val_loss: 0.0097 - val_mae: 0.0657
Epoch 73/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0135 - mae: 0.0821
- val_loss: 0.0097 - val_mae: 0.0652
Epoch 74/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0140 - mae: 0.0844
- val_loss: 0.0098 - val_mae: 0.0656
Epoch 75/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0138 - mae: 0.0811
- val_loss: 0.0091 - val_mae: 0.0634
Epoch 76/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0119 - mae: 0.0789
- val_loss: 0.0098 - val_mae: 0.0651
Epoch 77/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0102 - mae: 0.0737
- val_loss: 0.0095 - val_mae: 0.0637
Epoch 78/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0123 - mae: 0.0773
- val_loss: 0.0094 - val_mae: 0.0631
Epoch 79/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0111 - mae: 0.0743
- val_loss: 0.0092 - val_mae: 0.0629
Epoch 80/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0113 - mae: 0.0777
- val_loss: 0.0084 - val_mae: 0.0619
Epoch 81/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0112 - mae: 0.0767
- val_loss: 0.0084 - val_mae: 0.0609
Epoch 82/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0126 - mae: 0.0795
- val_loss: 0.0105 - val_mae: 0.0677
```

```
Epoch 83/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0093 - mae: 0.0721
- val_loss: 0.0097 - val_mae: 0.0656
Epoch 84/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0111 - mae: 0.0777
- val_loss: 0.0085 - val_mae: 0.0618
Epoch 85/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0101 - mae: 0.0740
- val_loss: 0.0111 - val_mae: 0.0686
Epoch 86/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0111 - mae: 0.0748
- val_loss: 0.0098 - val_mae: 0.0636
Epoch 87/100
7/7 [==============================] - 0s 7ms/step - loss: 0.0116 - mae: 0.0773
- val_loss: 0.0101 - val_mae: 0.0650
Epoch 88/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0127 - mae: 0.0781
- val_loss: 0.0088 - val_mae: 0.0615
Epoch 89/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0114 - mae: 0.0773
- val_loss: 0.0085 - val_mae: 0.0608
Epoch 90/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0109 - mae: 0.0730
- val_loss: 0.0092 - val_mae: 0.0628
Epoch 91/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0086 - mae: 0.0696
- val_loss: 0.0096 - val_mae: 0.0630
Epoch 92/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0101 - mae: 0.0745
- val_loss: 0.0081 - val_mae: 0.0594
Epoch 93/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0116 - mae: 0.0783
- val_loss: 0.0092 - val_mae: 0.0622
Epoch 94/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0105 - mae: 0.0753
- val_loss: 0.0082 - val_mae: 0.0588
Epoch 95/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0106 - mae: 0.0743
- val_loss: 0.0084 - val_mae: 0.0600
Epoch 96/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0093 - mae: 0.0717
- val_loss: 0.0091 - val_mae: 0.0635
Epoch 97/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0097 - mae: 0.0743
- val_loss: 0.0084 - val_mae: 0.0607
Epoch 98/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0099 - mae: 0.0738
- val_loss: 0.0102 - val_mae: 0.0660
```

```
Epoch 99/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0099 - mae: 0.0723
- val_loss: 0.0083 - val_mae: 0.0600
Epoch 100/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0086 - mae: 0.0673
- val_loss: 0.0087 - val_mae: 0.0609
```

[71]:
```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('median_mix_max_scaler',
       Pipeline(steps=[('imputer',
                        SimpleImputer(strategy='median')),
                       ('transformer',
                        MinMaxScaler())]),
       ['INDICE_CRIMEN',
        'PCT_ZONA_RESIDENCIAL',
        'PCT_ZONA_INDUSTRIAL',
        'OXIDO_NITROSO_PPM',
        'N_HABITACIONES_MEDIO',
        'PCT_CASAS_40S',
        'DIS',
        'DIS_AUTOPISTAS',
        'CARGA_FISCAL',
        'RATIO_PROFESORES',
        'PCT_NEGRA',
        'PCT_CLASE_BAJA']),
      ('mode_one_hot_encoder',
       Pipeline(steps=[('imputer',
                        SimpleImputer(strategy='most_frequent')),
                       ('transformer',
                        OneHotEncoder(drop='first'))]),
       ['RIO_CHARLES'])])),
                                                     ('model',
<keras.wrappers.scikit_learn.KerasRegressor object at 0x00000217156A4310>)]),
                           transformer=MinMaxScaler())
```

Analyzing the training and evaluation data.

[72]:
```python
keras_model = keras_estimator.regressor_['model'].model

historial_train = keras_model.history
hist = pd.DataFrame(historial_train.history)
hist['epoch'] = historial_train.epoch
hist.tail()
```

[72]:

	loss	mae	val_loss	val_mae	epoch
95	0.009331	0.071720	0.009102	0.063459	95
96	0.009669	0.074332	0.008448	0.060686	96
97	0.009941	0.073811	0.010231	0.066045	97

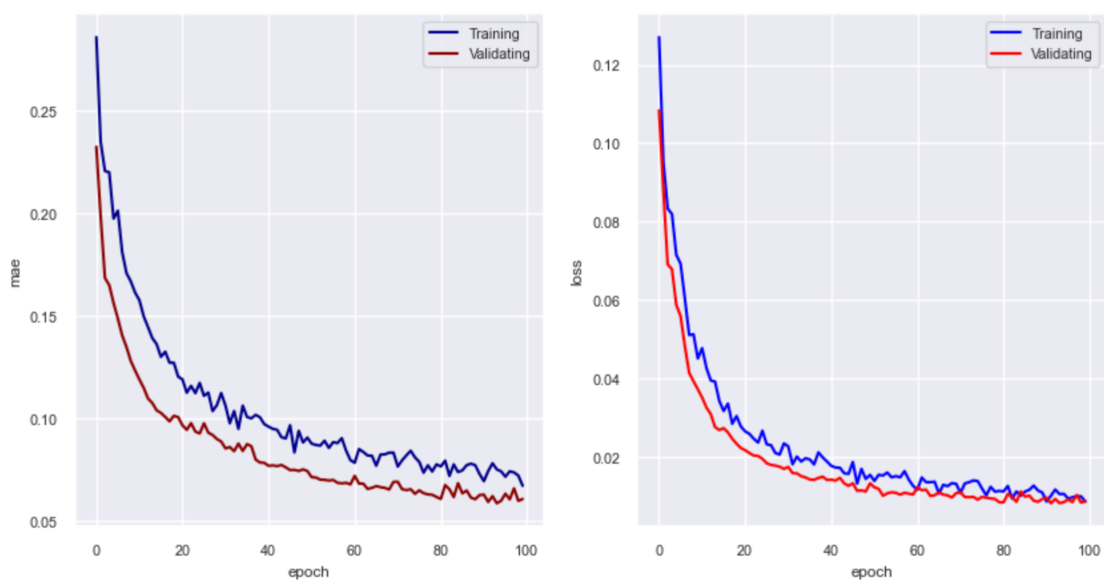
98	0.009887	0.072282	0.008312	0.059963	98
99	0.008559	0.067316	0.008669	0.060875	99

[73]:
```
def plot_metrics(train):
    plt.figure(figsize=(10, 5))

    ax1 = plt.subplot(1, 2, 1)
    ax1.set_xlabel('epoch')
    ax1.set_ylabel('mae')
    ax1.plot(train.history['mae'], color='darkblue', label='Training')
    ax1.plot(train.history['val_mae'], color='darkred', label='Validating')
    ax1.legend()

    ax1 = plt.subplot(1, 2, 2)
    ax1.set_xlabel('epoch')
    ax1.set_ylabel('loss')
    ax1.plot(train.history['loss'], color='blue', label='Training')
    ax1.plot(train.history['val_loss'], color='red', label='Validating')
    ax1.legend()


plot_metrics(historial_train)
```



Model evaluation.

[74]:
```
mae_train = mean_absolute_error(y_train, keras_estimator.predict(X=X_train))
mae_test = mean_absolute_error(y_test, keras_estimator.predict(X=X_test))
```

```python
mae_train, mae_test = round(mae_train, 4), round(mae_test, 4)

print(f'\nMAE Train: {mae_train}')
print(f'MAE Test: {mae_test}')
```

```
7/7 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step

MAE Train: 2.5253
MAE Test: 2.7228
```

[75]:
```python
keras_estimator
```

[75]:
```
TransformedTargetRegressor(regressor=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('median_mix_max_scaler',
       Pipeline(steps=[('imputer',
                        SimpleImputer(strategy='median')),
                       ('transformer',
                        MinMaxScaler())]),
       ['INDICE_CRIMEN',
        'PCT_ZONA_RESIDENCIAL',
        'PCT_ZONA_INDUSTRIAL',
        'OXIDO_NITROSO_PPM',
        'N_HABITACIONES_MEDIO',
        'PCT_CASAS_40S',
        'DIS',
        'DIS_AUTOPISTAS',
        'CARGA_FISCAL',
        'RATIO_PROFESORES',
        'PCT_NEGRA',
        'PCT_CLASE_BAJA']),
      ('mode_one_hot_encoder',
       Pipeline(steps=[('imputer',
                        SimpleImputer(strategy='most_frequent')),
                       ('transformer',
                        OneHotEncoder(drop='first'))]),
       ['RIO_CHARLES'])])),
                                                     ('model',
<keras.wrappers.scikit_learn.KerasRegressor object at 0x00000217156A4310>)]),
                           transformer=MinMaxScaler())
```

[76]:
```python
new_df = pd.DataFrame(data={'NAME': ['Keras (MLP)'], 'TYPE': 'DL',␣
 ↪'POLY_DEGREE': 0,
                            'SCORE': [-mae_test], 'ESTIMATOR':␣
 ↪[keras_estimator]})
results = pd.concat([results, new_df], ignore_index=True)
results
```

[76]:
```
                              NAME TYPE POLY_DEGREE   SCORE  \
0            LinearRegression()   ML           1 -3.9665
1                       Lasso()   ML           1 -5.8531
2                       Ridge()   ML           1 -3.9507
3                  ElasticNet()   ML           1 -4.6790
4       DecisionTreeRegressor()   ML           1 -3.5071
5                         SVR()   ML           1 -3.0232
6         KNeighborsRegressor()   ML           1 -3.3853
7              BayesianRidge()   ML           1 -3.9060
8            BaggingRegressor()   ML           1 -2.9822
9           AdaBoostRegressor()   ML           1 -2.9533
10  GradientBoostingRegressor()   ML           1 -3.3612
11      RandomForestRegressor()   ML           1 -2.9834
12                 Keras (MLP)   DL           0 -2.7228

                                            ESTIMATOR
0   TransformedTargetRegressor(regressor=Pipeline(…
1   TransformedTargetRegressor(regressor=Pipeline(…
2   TransformedTargetRegressor(regressor=Pipeline(…
3   TransformedTargetRegressor(regressor=Pipeline(…
4   TransformedTargetRegressor(regressor=Pipeline(…
5   TransformedTargetRegressor(regressor=Pipeline(…
6   TransformedTargetRegressor(regressor=Pipeline(…
7   TransformedTargetRegressor(regressor=Pipeline(…
8   TransformedTargetRegressor(regressor=Pipeline(…
9   TransformedTargetRegressor(regressor=Pipeline(…
10  TransformedTargetRegressor(regressor=Pipeline(…
11  TransformedTargetRegressor(regressor=Pipeline(…
12  TransformedTargetRegressor(regressor=Pipeline(…
```

8 Resume

[77]:
```
results_sort = results.sort_values(by=['SCORE'], ascending=[False])
results_sort
```

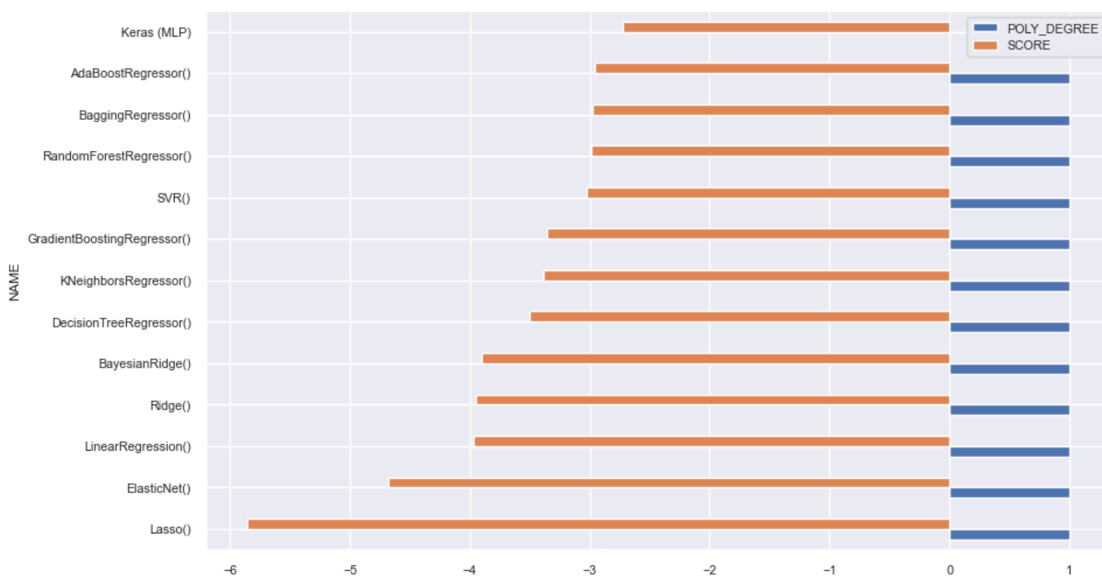
[77]:
```
                              NAME TYPE POLY_DEGREE   SCORE  \
12                 Keras (MLP)   DL           0 -2.7228
9           AdaBoostRegressor()   ML           1 -2.9533
8            BaggingRegressor()   ML           1 -2.9822
11      RandomForestRegressor()   ML           1 -2.9834
5                         SVR()   ML           1 -3.0232
10  GradientBoostingRegressor()   ML           1 -3.3612
6         KNeighborsRegressor()   ML           1 -3.3853
4       DecisionTreeRegressor()   ML           1 -3.5071
7              BayesianRidge()   ML           1 -3.9060
2                       Ridge()   ML           1 -3.9507
0            LinearRegression()   ML           1 -3.9665
```

```
3                  ElasticNet()   ML          1 -4.6790
1                       Lasso()   ML          1 -5.8531

                                        ESTIMATOR
12  TransformedTargetRegressor(regressor=Pipeline(…
9   TransformedTargetRegressor(regressor=Pipeline(…
8   TransformedTargetRegressor(regressor=Pipeline(…
11  TransformedTargetRegressor(regressor=Pipeline(…
5   TransformedTargetRegressor(regressor=Pipeline(…
10  TransformedTargetRegressor(regressor=Pipeline(…
6   TransformedTargetRegressor(regressor=Pipeline(…
4   TransformedTargetRegressor(regressor=Pipeline(…
7   TransformedTargetRegressor(regressor=Pipeline(…
2   TransformedTargetRegressor(regressor=Pipeline(…
0   TransformedTargetRegressor(regressor=Pipeline(…
3   TransformedTargetRegressor(regressor=Pipeline(…
1   TransformedTargetRegressor(regressor=Pipeline(…
```

[78]:
```python
plt.rc('figure', figsize=(10, 6))
results_sort[::-1].set_index('NAME').plot.barh(rot=0)
```

[78]: `<AxesSubplot: ylabel='NAME'>`



[79]:
```python
print(
    f'Best tentative algorithm "{results_sort.iloc[0].NAME}" with␣
↪SCORE={-results_sort.iloc[0].SCORE}.')
```

```
Best tentative algorithm "Keras (MLP)" with SCORE=2.7228.
```

8.1 Export

Save to file.

[80]:
```
best_estimator = results_sort.iloc[0].ESTIMATOR
estimator_type = results_sort.iloc[0].TYPE

if estimator_type == 'ML':
    pkl_filename = '_resources/estimator.pkl'

    with open(pkl_filename, 'wb') as file:
        pickle.dump(best_estimator, file)
else:
    # Si se intenta guardar el estimador con un modelo de Keras dentro emite un␣
 ↪error, esperar actualizaciones.

    model_folder = '_resources/model'

    if os.path.isdir(model_folder) == False:
        os.mkdir(model_folder)

    pickle.dump(preprocessor, open(f'{model_folder}/preprocessor.pkl', 'wb'))

    keras_model.save(f'{model_folder}/keras.h5')

    pickle.dump(target_transformer,
                open(f'{model_folder}/target_transformer.pkl', 'wb'))
```

Load from file.

[81]:
```
if estimator_type == 'ML':
    with open(pkl_filename, 'rb') as file:
        model_pickle = pickle.load(file)

    display(model_pickle)
else:
    preprocessor_pickle = pickle.load(
        open(f'{model_folder}/preprocessor.pkl', 'rb'))

    keras_model_h5 = load_model(f'{model_folder}/keras.h5')

    target_transformer_pickle = pickle.load(
        open(f'{model_folder}/target_transformer.pkl', 'rb'))

    # No se puede crear un TransformedTargetRegressor pasándole los parámetros
    # porque aún así se necesitaría ajustar.

    display(preprocessor_pickle, keras_model_h5, target_transformer_pickle)
```

```
ColumnTransformer(transformers=[('median_mix_max_scaler',
                                 Pipeline(steps=[('imputer',
                                                                                  ␣
 ↪SimpleImputer(strategy='median')),
                                                 ('transformer',
                                                  MinMaxScaler())]),
                                 ['INDICE_CRIMEN', 'PCT_ZONA_RESIDENCIAL',
                                  'PCT_ZONA_INDUSTRIAL', 'OXIDO_NITROSO_PPM',
                                  'N_HABITACIONES_MEDIO', 'PCT_CASAS_40S',
                                  'DIS', 'DIS_AUTOPISTAS', 'CARGA_FISCAL',
                                  'RATIO_PROFESORES', 'PCT_NEGRA',
                                  'PCT_CLASE_BAJA']),
                                ('mode_one_hot_encoder',
                                 Pipeline(steps=[('imputer',
                                                                                  ␣
 ↪SimpleImputer(strategy='most_frequent')),
                                                 ('transformer',
                                                  OneHotEncoder(drop='first'))]),
                                 ['RIO_CHARLES'])])
```

```
<keras.engine.sequential.Sequential at 0x2170dfe1fa0>
```

```
MinMaxScaler()
```

8.2 Predict

[82]:
```
_X = df.iloc[:10]

if estimator_type == 'ML':
    pred_df = pd.DataFrame(data={'REAL': _X.VALOR_MEDIANO,
                                 'PRED': model_pickle.predict(X=_X)})
else:
    # Como no se puede crear un único estimador, se realizan los pasos por␣
 ↪separados.

    d1 = preprocessor_pickle.transform(_X)
    d2 = keras_model_h5.predict(d1)
    d3 = target_transformer_pickle.inverse_transform(d2)
    d3 = np.reshape(d3, -1)

    pred_df = pd.DataFrame(data={'REAL': _X.VALOR_MEDIANO, 'PRED': d3})

pred_df
```

[82]:

	REAL	PRED
0	24.0	28.114729
1	21.6	20.725950
2	34.7	27.041367

```
3  33.4  28.570129
4  36.2  27.184189
5  28.7  24.308214
6  22.9  20.554930
7  22.1  18.404835
8  16.5  15.070657
9  18.9  18.948154
```