Ocean Proximity

In this exercise we will try to predict the proximity of the ocean with respect to a set of houses.

Link:

- Exercise Kaggle: https://www.kaggle.com/camnugent/california-housing-prices

1. Data Analysis

Library Import

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Activation
from keras.callbacks import EarlyStopping
import tensorflow as tf
```

Dataset Import

```
# Connect with Google Drive
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/datasets/DL1_OP/housing.csv')
df.shape
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.
(20640, 10)
```

```
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0

variables

- **longitude:** Longitude value of the coordinate.
- **latitude:** Latitude value of the coordinate.
- **housing_median_age:** Average age of the dwellings in this area.
- **total_rooms:** Total rooms.
- **total_bedrooms:** Total beds.
- **population:** Population in this zone. It is important to note that it is also a total value.
- **households:** Homes in this area. It is important to note that it is also a total value.
- **median_income:** Median salary of people in this area.
- **median_house_value:** Median house value in this area.
- **ocean_proximity:** The result! It means the proximity of the ocean with respect to the houses in this area. If you look closely, this field contains string values (labels) to determine proximity.

```
df.describe()
```

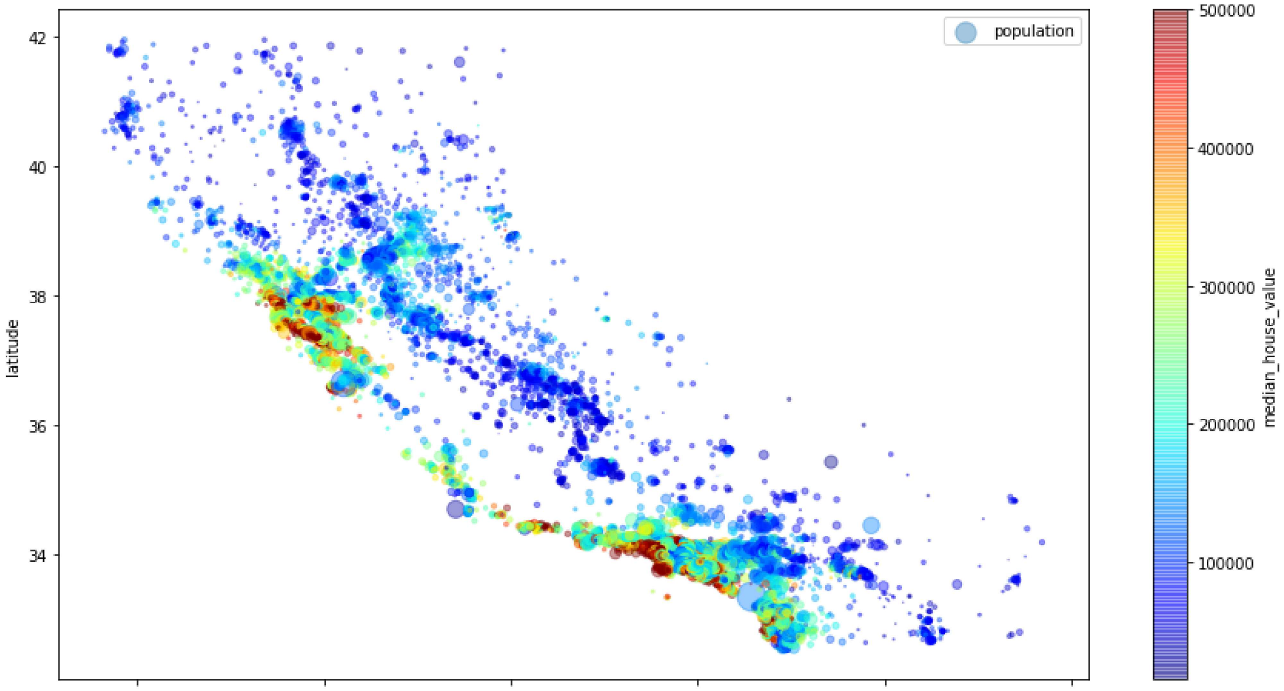
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553
std	2.003532	2.135952	12.585558	2181.615252	421.385070
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000

Let's visualize the content to get an idea of the distribution of dwellings by population and price.

- It is the area of California, and to the southwest where the circles end, the ocean begins.
- The price is indicated in red for the most expensive homes.
- The homes closest to the ocean are therefore on the California coast.

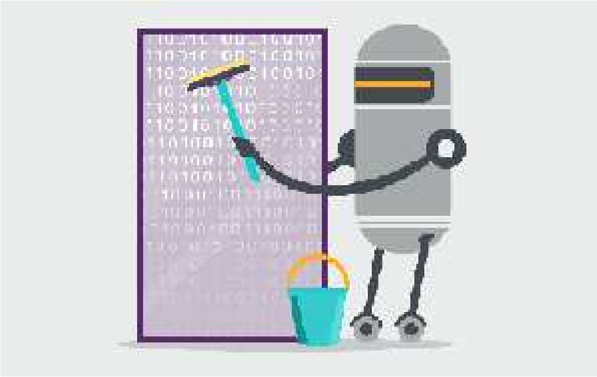
```
plt.figure(figsize=(10,7))#prepara el plot
plotter=df.copy()# copia el dataframe
plotter.plot(kind='scatter',x='longitude',y='latitude',alpha=0.4,s=plotter['population'
                  c='median_house_value',cmap=plt.get_cmap('jet'),colorbar=True)

```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3567289210>
<Figure size 720x504 with 0 Axes>
```



2. Data Pocessing

Clean and normalize dataset information



Null data is not recommended. We must give them a value (the average value or close values) or eliminate them (if there are not many).

```
df.isnull().sum()
```

```
longitude             0
latitude              0
housing_median_age    0
```

```
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

```
df.dropna(inplace=True)
df
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populat
0	-122.23	37.88	41.0	880.0	129.0	3
1	-122.22	37.86	21.0	7099.0	1106.0	24
2	-122.24	37.85	52.0	1467.0	190.0	4
3	-122.25	37.85	52.0	1274.0	235.0	5
4	-122.25	37.85	52.0	1627.0	280.0	5
...	...	...	...	...	...	
20635	-121.09	39.48	25.0	1665.0	374.0	8
20636	-121.21	39.49	18.0	697.0	150.0	3
20637	-121.22	39.43	17.0	2254.0	485.0	10
20638	-121.32	39.43	18.0	1860.0	409.0	7
20639	-121.24	39.37	16.0	2785.0	616.0	13

20433 rows × 10 columns

```
df.isnull().sum()
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

Correlation Matrix

```
corr=df.corr()
corr.style.background_gradient(cmap='coolwarm')
```

	longitude	latitude	housing_median_age	total_rooms	total_bed
longitude	1.000000	-0.924616	-0.109357	0.045480	0.0
latitude	-0.924616	1.000000	0.011899	-0.036667	-0.0
housing_median_age	-0.109357	0.011899	1.000000	-0.360628	-0.3
total_rooms	0.045480	-0.036667	-0.360628	1.000000	0.9
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.0
population	0.100270	-0.108997	-0.295787	0.857281	0.8
households	0.056513	-0.071774	-0.302768	0.918992	0.9
median_income	-0.015550	-0.079626	-0.118278	0.197882	-0.0
median_house_value	-0.045398	-0.144638	0.106432	0.133294	0.0

'total rooms', 'total rooms', 'population', are total variables of a zone (see values in the table), for which we must solve this problem, normalize them by house and not by zone

```
df['rooms_per_household']= df['total_rooms']/df['households']
df['bedrooms_per_household']=df['total_bedrooms']/df['households']
df['populatio_per_household']=df['population']/df['households']
```

```
df.drop(['total_rooms','total_bedrooms','population','households'],axis=1,inplace=True)
```

```
corr=df.corr()
corr.style.background_gradient(cmap='PiYG')
#color_map:https://matplotlib.org/2.0.2/examples/color/colormaps_reference.html
```

	longitude	latitude	housing_median_age	median_income	med
longitude	1.000000	-0.924616	-0.109357	-0.015550	
latitude	-0.924616	1.000000	0.011899	-0.079626	
housing_median_age	-0.109357	0.011899	1.000000	-0.118278	
median_income	-0.015550	-0.079626	-0.118278	1.000000	
median_house_value	-0.045398	-0.144638	0.106432	0.688355	
rooms_per_household	-0.027307	0.106423	-0.153031	0.325307	
bedrooms_per_household	0.013402	0.070025	-0.077918	-0.062299	
populatio_per_household	0.002304	0.002522	0.013258	0.018894	

```
ocean_proximity=df.ocean_proximity.astype('category')
ocean_proximity.unique()
```

```
['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND']
Categories (5, object): ['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN']
```

```
(df['ocean_proximity']=='ISLAND').sum()
```

```
5
```

There are no houses on islands

```
df=df[df['ocean_proximity']!='ISLAND']
```

Data preparation

- To normalize, the already created StandardScaler object from the sklearn.preprocessing library is used: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- On the other hand, it is necessary to create an array of dimension nx1, since it is adequate to work with neural networks.

```
df['longitude']
```

```
0       -122.23
1       -122.22
2       -122.24
3       -122.25
4       -122.25
          ...
20635   -121.09
20636   -121.21
20637   -121.22
20638   -121.32
20639   -121.24
Name: longitude, Length: 20428, dtype: float64
```

```
df['longitude'].values
```

```
array([-122.23, -122.22, -122.24, ..., -121.22, -121.32, -121.24])
```

We will apply reshape(-1,1), where the -1 does not indicate a specific dimension, but rather the array will have N rows by one column. This is the dimension that the network expects in its input data.

We use StandardScaler to Normally scale a variable

```
scaler=preprocessing.StandardScaler()
df['longitude']= scaler.fit_transform(df['longitude'].values.reshape(-1,1))
df['latitude']= scaler.fit_transform(df['latitude'].values.reshape(-1,1))
df['housing_median_age']= scaler.fit_transform(df['housing_median_age'].values.reshape(
df['median_income']= scaler.fit_transform(df['median_income'].values.reshape(-1,1))
df['median_house_value']= scaler.fit_transform(df['median_house_value'].values.reshape(
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
  """
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
```

```
df
```

	longitude	latitude	housing_median_age	median_income	median_house_value
0	-1.327063	1.051474	0.982560	2.344844	2.129617
1	-1.322072	1.042112	-0.606022	2.332314	1.314260
2	-1.332054	1.037431	1.856279	1.782662	1.258805
3	-1.337044	1.037431	1.856279	0.932756	1.165225
4	-1.337044	1.037431	1.856279	-0.013287	1.173024
...	...	...	...	...	...

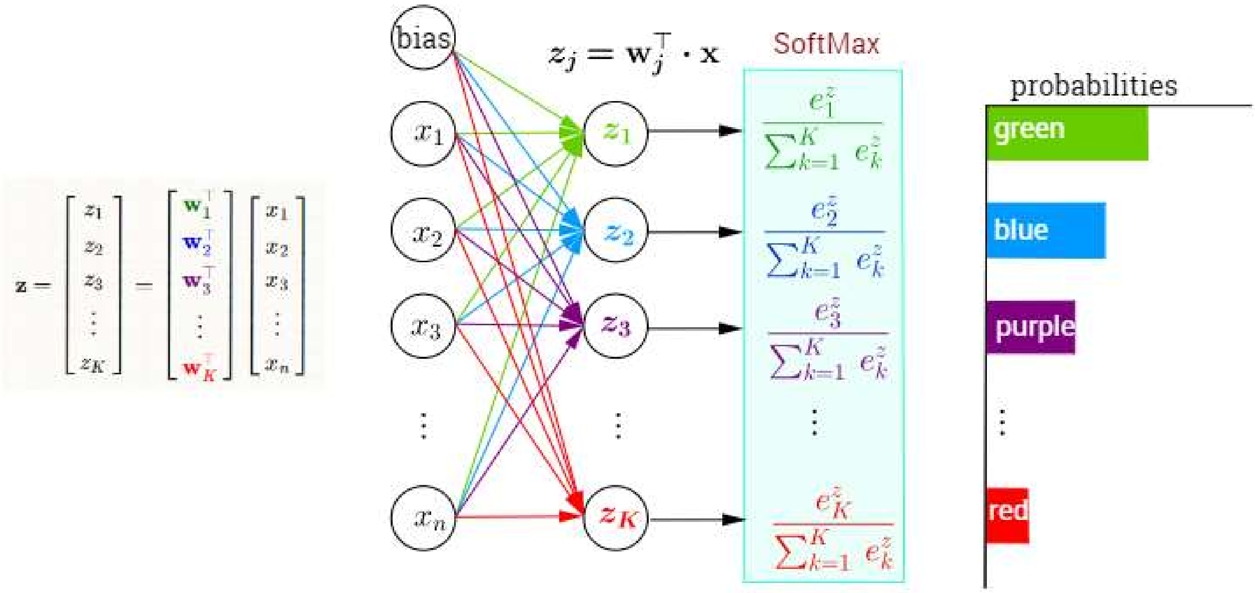
```
df.describe(include='all')
```

	longitude	latitude	housing_median_age	median_income	median_house
count	2.042800e+04	2.042800e+04	2.042800e+04	2.042800e+04	2.0428
unique	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	
mean	8.026535e-15	6.663849e-15	3.990727e-15	-2.247674e-16	9.7734
std	1.000024e+00	1.000024e+00	1.000024e+00	1.000024e+00	1.0000
min	-2.385114e+00	-1.448222e+00	-2.194603e+00	-1.775068e+00	-1.6621
25%	-1.112458e+00	-7.975518e-01	-8.443092e-01	-6.886636e-01	-7.5684
50%	5.345088e-01	-6.430762e-01	2.941069e-02	-1.758134e-01	-2.3500
75%	7.790584e-01	9.765767e-01	6.648433e-01	4.594316e-01	5.0150
max	2.625658e+00	2.956673e+00	1.856279e+00	5.859087e+00	2.5403

3. Modelo de Red Neuronal

Ejemplo de Red Neuronal Clasificador con función de activación SoftMax

Multi-Class Classification with NN and SoftMax Function



$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$
bias
x_1
x_2
x_3
x_n
$z_j = \mathbf{w}_j^\top \cdot \mathbf{x}$
z_1
z_2
z_3
z_K
SoftMax
$\frac{e_1^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_2^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_3^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_K^z}{\sum_{k=1}^K e_k^z}$
probabilities
green
blue
purple
red

- The first thing we have to do is remove the output variable. If you don't do this, the model will be trained with the same variable you want to predict. A new DataFrame is created that we will call X and it will be our input variable.
- We have to make a LeavelEncoder to be able to assign a numeric value to the output variable (ocean proximity), and that until now has character values. A DataFrame Y is created for the output variable.
- Create a set of training and validation data. For this we use train_test_split, with the input data and the output variable. Which serves to divide arrays or matrices into random subsets of train and test. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
# Crea X , que sea igual al dataframe menos la feature que queremos averiguar.
X=df.drop(['ocean_proximity'],axis=1)
X
```

	longitude	latitude	housing_median_age	median_income	median_house_value
0	-1.327063	1.051474	0.982560	2.344844	2.129617
1	-1.322072	1.042112	-0.606022	2.332314	1.314260
2	-1.332054	1.037431	1.856279	1.782662	1.258805
3	-1.337044	1.037431	1.856279	0.932756	1.165225

```
# This creates and , with a LabelEncoder to pass the values of the feature we want to f
encoder=preprocessing.LabelEncoder()
Y=encoder.fit_transform(df.ocean_proximity)
```

```
# This reates the training and validation dataset.
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
```

Proper form is needed for the output variable. So we do a OneHotencoding to convert a class vector (integers) to a binary class array.
https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical

```
# This converts y_train and y_test to the format needed to train our model with: np_uti
num_classes=len(np.unique(Y))
Y_train=np_utils.to_categorical(y_train,num_classes)
Y_test=np_utils.to_categorical(y_test,num_classes)
Y_test
```

```
array([[0., 1., 0., 0.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       ...,
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [1., 0., 0., 0.]], dtype=float32)
```

Model

- Adam optimizer is used and a loss function that is categorical since it is a categorization problem.
- There are 8 input dimensions and 4 output dimensions that must be respected due to the nature of the data. The activation function of the last layer must be sotmax because it will find the probabilities of the last layer that has four possible outputs.

```
es=EarlyStopping(monitor='val_loss',mode='min',verbose=2,restore_best_weights=True,pati
model=Sequential()
model.add(Dense(500,input_dim=8,activation='relu'))
model.add(Dense(300,activation='relu'))
model.add(Dense(300,activation='relu'))
model.add(Dense(300,activation='relu'))
model.add(Dense(100,activation='relu'))
model.add(Dense(100,activation='relu'))
model.add(Dense(100,activation='relu'))
model.add(Dense(4,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_8 (Dense)             (None, 500)               4500      
                                                                 
 dense_9 (Dense)             (None, 300)               150300    
                                                                 
 dense_10 (Dense)            (None, 300)               90300     
                                                                 
 dense_11 (Dense)            (None, 300)               90300     
                                                                 
 dense_12 (Dense)            (None, 100)               30100     
                                                                 
 dense_13 (Dense)            (None, 100)               10100     
                                                                 
 dense_14 (Dense)            (None, 100)               10100     
                                                                 
 dense_15 (Dense)            (None, 4)                 404       
                                                                 
=================================================================
Total params: 386,104
Trainable params: 386,104
Non-trainable params: 0
_________________________________________________________________
```

Training

- We must train the model by passing the training data X_train and Y_train.
- We also define a validation split, since we are using EarlyStoping. In this way, a validation set is defined that will be used throughout the training and not at the end, as is the case with the Test data. It is to be expected that the accuracy of the train and the validation will grow together as long as we do not have overfitting. On the contrary, these values differ when the network begins to memorize the data.

EarlyStopping will stop the model earlier when it starts to overfit. You may not complete all 150 epochs of training. This is done from the callbacks parameter, every time the model trains an epoch, it will verify if it is convenient to stop learning or not.

```
es=EarlyStopping(monitor='val_loss',mode='min',verbose=2,restore_best_weights=True,pati
history=model.fit(X_train,Y_train,validation_split=0.25,batch_size=200,epochs=150,callb
```

```
62/62 [==============================] - 1s 21ms/step - loss: 0.1655 - accuracy: 0
Epoch 33/150
```

```
62/62 [==============================] - 1s 21ms/step - loss: 0.1558 - accuracy: 0
Epoch 34/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1480 - accuracy: 0
Epoch 35/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1732 - accuracy: 0
Epoch 36/150
62/62 [==============================] - 1s 22ms/step - loss: 0.1591 - accuracy: 0
Epoch 37/150
62/62 [==============================] - 1s 22ms/step - loss: 0.1647 - accuracy: 0
Epoch 38/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1597 - accuracy: 0
Epoch 39/150
62/62 [==============================] - 1s 22ms/step - loss: 0.1484 - accuracy: 0
Epoch 40/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1633 - accuracy: 0
Epoch 41/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1549 - accuracy: 0
Epoch 42/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1491 - accuracy: 0
Epoch 43/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1379 - accuracy: 0
Epoch 44/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1520 - accuracy: 0
Epoch 45/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1336 - accuracy: 0
Epoch 46/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1489 - accuracy: 0
Epoch 47/150
62/62 [==============================] - 1s 22ms/step - loss: 0.1403 - accuracy: 0
Epoch 48/150
62/62 [==============================] - 1s 20ms/step - loss: 0.1247 - accuracy: 0
Epoch 49/150
62/62 [==============================] - 1s 20ms/step - loss: 0.1294 - accuracy: 0
Epoch 50/150
62/62 [==============================] - 1s 20ms/step - loss: 0.1210 - accuracy: 0
Epoch 51/150
62/62 [==============================] - 1s 20ms/step - loss: 0.1306 - accuracy: 0
Epoch 52/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1309 - accuracy: 0
Epoch 53/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1341 - accuracy: 0
Epoch 54/150
62/62 [==============================] - 1s 20ms/step - loss: 0.1228 - accuracy: 0
Epoch 55/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1300 - accuracy: 0
Epoch 56/150
62/62 [==============================] - 1s 20ms/step - loss: 0.1257 - accuracy: 0
Epoch 57/150
62/62 [==============================] - 1s 20ms/step - loss: 0.1177 - accuracy: 0
Epoch 58/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1211 - accuracy: 0
Epoch 59/150
62/62 [==============================] - 1s 21ms/step - loss: 0.1382 - accuracy: 0
Epoch 60/150
60/62 [============================>.] - ETA: 0s - loss: 0.1293 - accuracy: 0.9454
```

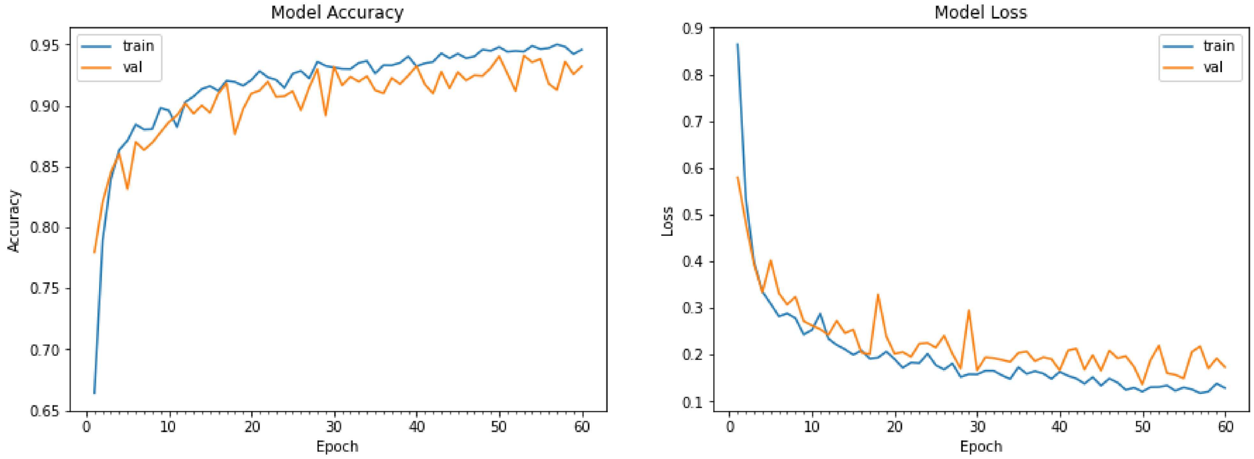
```
results=model.evaluate(X_test,Y_test)
results
```

```
128/128 [==============================] - 0s 3ms/step - loss: 0.1628 - accuracy: 0.
[0.16281427443027496, 0.9434654712677002]
```

```
# FUNCIÓN AUXILIAR, EJECUTAR, NO MODIFICAR.
def plot_model_history(model_history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # Summarize history for accuracy
    axs[0].plot(range(1,len(model_history.history['accuracy'])+1),model_history.history
    axs[0].plot(range(1,len(model_history.history['val_accuracy'])+1),model_history.his
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1,len(model_history.history['accuracy'])+1),len(model_h
    axs[0].legend(['train', 'val'], loc='best')
    # summarize history for loss
    axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history['lo
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.history
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_histo
    axs[1].legend(['train', 'val'], loc='best')
    plt.show()
```

```
plot_model_history(history)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: MatplotlibDepreca
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: MatplotlibDepreca
```



✓ 0 s completado a las 15:14 ● ✕