Fuente de Datos : https://www.kaggle.com/kyanyoga/sample-sales-data

IMPORTAR LIBRERÍAS Y DATASETS

```
In [ ]:
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import zipfile
import cv2
import tensorflow as tf
from tensorflow.python.keras import Sequential
from tensorflow.keras import layers, optimizers
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten,
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint, LearningRateSchedule
from IPython.display import display
from tensorflow.keras import backend as K
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.metrics.pairwise import cosine_similarity
import plotly.express as px
import plotly.graph_objects as go

from google.colab import files #Libreria para cargar ficheros directamente en Colab
%matplotlib inline
```

```
In [ ]:
# Procedemos a montar el disco usando los siguientes comandos:

from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [ ]:
sales_df = pd.read_csv('/content/drive/MyDrive/01_Oriol/01_Machine Learning/07_Masterclass-IA-Moderna/03_Market
```

```
In [ ]:
sales_df
```

Out[ ]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003
...	...	...	...	...	...	...	...	...	...	...
2818	10350	20	100.00	15	2244.40	12/2/2004 0:00	Shipped	4	12	2004
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped	1	1	2005
2820	10386	43	100.00	4	5417.57	3/1/2005 0:00	Resolved	1	3	2005
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped	1	3	2005
2822	10414	47	65.52	9	3079.44	5/6/2005 0:00	On Hold	2	5	2005

2823 rows × 25 columns

```
In [ ]:
# Veamos los tipos de datos
sales_df.dtypes
```

```
Out[ ]:
ORDERNUMBER           int64
QUANTITYORDERED       int64
PRICEEACH           float64
ORDERLINENUMBER       int64
SALES               float64
ORDERDATE            object
STATUS               object
QTR_ID                int64
MONTH_ID              int64
YEAR_ID               int64
PRODUCTLINE          object
MSRP                  int64
PRODUCTCODE          object
CUSTOMERNAME         object
PHONE                object
ADDRESSLINE1         object
ADDRESSLINE2         object
CITY                 object
STATE                object
POSTALCODE           object
COUNTRY              object
TERRITORY            object
CONTACTLASTNAME      object
CONTACTFIRSTNAME     object
DEALSIZE             object
dtype: object
```

```
In [ ]:
# Convertir la fecha del pedido en formato de fecha y hora
sales_df['ORDERDATE'] = pd.to_datetime(sales_df['ORDERDATE'])
# Comprobar el tipo de datos
sales_df.dtypes
```

```
Out[ ]:
ORDERNUMBER                    int64
QUANTITYORDERED                int64
PRICEEACH                    float64
ORDERLINENUMBER                int64
SALES                        float64
ORDERDATE             datetime64[ns]
STATUS                        object
QTR_ID                         int64
MONTH_ID                       int64
YEAR_ID                        int64
PRODUCTLINE                   object
MSRP                           int64
PRODUCTCODE                   object
CUSTOMERNAME                  object
PHONE                         object
ADDRESSLINE1                  object
ADDRESSLINE2                  object
CITY                          object
STATE                         object
POSTALCODE                    object
COUNTRY                       object
TERRITORY                     object
CONTACTLASTNAME               object
CONTACTFIRSTNAME              object
DEALSIZE                      object
dtype: object
```

```
In [ ]:
# Comprobar el número de elementos no nulos del data frame
sales_df.info()

# Comprobar el número de elementos nulos del data frame
sales_df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ORDERNUMBER       2823 non-null   int64
 1   QUANTITYORDERED   2823 non-null   int64
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   datetime64[ns]
 6   STATUS            2823 non-null   object
 7   QTR_ID            2823 non-null   int64
 8   MONTH_ID          2823 non-null   int64
 9   YEAR_ID           2823 non-null   int64
 10  PRODUCTLINE       2823 non-null   object
 11  MSRP              2823 non-null   int64
 12  PRODUCTCODE       2823 non-null   object
 13  CUSTOMERNAME      2823 non-null   object
 14  PHONE             2823 non-null   object
 15  ADDRESSLINE1      2823 non-null   object
 16  ADDRESSLINE2      302 non-null    object
 17  CITY              2823 non-null   object
 18  STATE             1337 non-null   object
 19  POSTALCODE        2747 non-null   object
 20  COUNTRY           2823 non-null   object
 21  TERRITORY         1749 non-null   object
 22  CONTACTLASTNAME   2823 non-null   object
 23  CONTACTFIRSTNAME  2823 non-null   object
 24  DEALSIZE          2823 non-null   object
dtypes: datetime64[ns](1), float64(2), int64(7), object(15)
memory usage: 551.5+ KB
```

```
Out[ ]:
ORDERNUMBER            0
QUANTITYORDERED        0
PRICEEACH              0
ORDERLINENUMBER        0
SALES                  0
ORDERDATE              0
STATUS                 0
QTR_ID                 0
MONTH_ID               0
YEAR_ID                0
PRODUCTLINE            0
MSRP                   0
PRODUCTCODE            0
CUSTOMERNAME           0
PHONE                  0
ADDRESSLINE1           0
ADDRESSLINE2        2521
CITY                   0
STATE               1486
POSTALCODE            76
COUNTRY                0
TERRITORY           1074
CONTACTLASTNAME        0
CONTACTFIRSTNAME       0
DEALSIZE               0
dtype: int64
```

```
In [ ]:
# Dado que hay muchos valores nulos en 'addressline2', 'state', 'postal code' y 'territory', podemos eliminar
# Country representa la información geográfica del pedido.
# También podemos eliminar la ciudad, la dirección1, el número de teléfono, el nombre del contacto, el apellido

df_drop  = ['ADDRESSLINE1', 'ADDRESSLINE2', 'POSTALCODE', 'CITY', 'TERRITORY', 'PHONE', 'STATE', 'CONTACTFIRSTN
sales_df = sales_df.drop(df_drop, axis = 1)
sales_df.head()
```

Out[ ]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE	MSRP
0	30	95.70	2	2871.00	2003-02-24	Shipped	1	2	2003	Motorcycles	95
1	34	81.35	5	2765.90	2003-05-07	Shipped	2	5	2003	Motorcycles	95
2	41	94.74	2	3884.34	2003-07-01	Shipped	3	7	2003	Motorcycles	95
3	45	83.26	6	3746.70	2003-08-25	Shipped	3	8	2003	Motorcycles	95
4	49	100.00	14	5205.27	2003-10-10	Shipped	4	10	2003	Motorcycles	95

```
In [ ]:
sales_df.isnull().sum()
```

```
Out[ ]:
QUANTITYORDERED    0
PRICEEACH          0
ORDERLINENUMBER    0
SALES              0
ORDERDATE          0
STATUS             0
QTR_ID             0
MONTH_ID           0
YEAR_ID            0
PRODUCTLINE        0
MSRP               0
PRODUCTCODE        0
COUNTRY            0
DEALSIZE           0
dtype: int64
```

```
In [ ]:
# Obtener las observaciones únicas de cada columna
sales_df.nunique()
```

```
Out[ ]:
QUANTITYORDERED      58
PRICEEACH          1016
ORDERLINENUMBER      18
SALES              2763
ORDERDATE           252
STATUS                6
QTR_ID                4
MONTH_ID             12
YEAR_ID               3
PRODUCTLINE           7
MSRP                 80
PRODUCTCODE         109
COUNTRY              19
DEALSIZE              3
dtype: int64
```

ANALISIS EXPLORATORIO DE LOS DATOS Y LIMPIEZA

```
In [ ]:
sales_df['COUNTRY'].value_counts().index
```

```
Out[ ]:
Index(['USA', 'Spain', 'France', 'Australia', 'UK', 'Italy', 'Finland',
       'Norway', 'Singapore', 'Canada', 'Denmark', 'Germany', 'Sweden',
       'Austria', 'Japan', 'Belgium', 'Switzerland', 'Philippines', 'Ireland'],
      dtype='object')
```

```
In [ ]:
sales_df['COUNTRY'].value_counts()
```

```
Out[ ]:
USA            1004
Spain           342
France          314
Australia       185
UK              144
Italy           113
Finland          92
Norway           85
Singapore        79
Canada           70
Denmark          63
Germany          62
Sweden           57
Austria          55
Japan            52
Belgium          33
Switzerland      31
Philippines      26
Ireland          16
Name: COUNTRY, dtype: int64
```

```
In [ ]:
# Visualizar el recuento de elementos en una columna determinada

def barplot_visualization(x):
  fig = plt.Figure(figsize = (12, 6))
  fig = px.bar(x = sales_df[x].value_counts().index, y = sales_df[x].value_counts(), color = sales_df[x].value_
  fig.show()
```

```
In [ ]:
# Llamamos a esta función para cualquier columna determinada, como 'COUNTRY'
barplot_visualization('COUNTRY')
```

```
In [ ]:
barplot_visualization('STATUS')
```

```
In [ ]:
sales_df.drop(columns= ['STATUS'], inplace = True)
sales_df
```

Out[ ]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE	MSRP	PRO
0	30	95.70	2	2871.00	2003-02-24	1	2	2003	Motorcycles	95	
1	34	81.35	5	2765.90	2003-05-07	2	5	2003	Motorcycles	95	
2	41	94.74	2	3884.34	2003-07-01	3	7	2003	Motorcycles	95	
3	45	83.26	6	3746.70	2003-08-25	3	8	2003	Motorcycles	95	
4	49	100.00	14	5205.27	2003-10-10	4	10	2003	Motorcycles	95	
...	...	...	...	...	...	...	...	...	...	...	
2818	20	100.00	15	2244.40	2004-12-02	4	12	2004	Ships	54	
2819	29	100.00	1	3978.51	2005-01-31	1	1	2005	Ships	54	
2820	43	100.00	4	5417.57	2005-03-01	1	3	2005	Ships	54	
2821	34	62.24	1	2116.16	2005-03-28	1	3	2005	Ships	54	
2822	47	65.52	9	3079.44	2005-05-06	2	5	2005	Ships	54	

2823 rows × 13 columns

```
In [ ]:
barplot_visualization('PRODUCTLINE')
```

```
In [ ]:
barplot_visualization('DEALSIZE')
```

```
In [ ]:
# Agregar variables ficticias para reemplazar variables categóricas

def dummies(x):
  dummy = pd.get_dummies(sales_df[x])
  sales_df.drop(columns = x , inplace = True)
  return pd.concat([sales_df, dummy], axis = 1)
```

```
In [ ]:
# Obtenemos variables ficticias para la columna 'PAIS'
sales_df = dummies('COUNTRY')
sales_df
```

Out[ ]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE	MSRP	PRO
0	30	95.70	2	2871.00	2003-02-24	1	2	2003	Motorcycles	95	
1	34	81.35	5	2765.90	2003-05-07	2	5	2003	Motorcycles	95	
2	41	94.74	2	3884.34	2003-07-01	3	7	2003	Motorcycles	95	
3	45	83.26	6	3746.70	2003-08-25	3	8	2003	Motorcycles	95	
4	49	100.00	14	5205.27	2003-10-10	4	10	2003	Motorcycles	95	
...	...	...	...	...	...	...	...	...	...	...	
2818	20	100.00	15	2244.40	2004-12-02	4	12	2004	Ships	54	
2819	29	100.00	1	3978.51	2005-01-31	1	1	2005	Ships	54	
2820	43	100.00	4	5417.57	2005-03-01	1	3	2005	Ships	54	
2821	34	62.24	1	2116.16	2005-03-28	1	3	2005	Ships	54	
2822	47	65.52	9	3079.44	2005-05-06	2	5	2005	Ships	54	

2823 rows × 31 columns

```
In [ ]:
sales_df = dummies('PRODUCTLINE')
```

```
In [ ]:
sales_df
```

Out[ ]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	QTR_ID	MONTH_ID	YEAR_ID	MSRP	PRODUCTCODE	DEA
0	30	95.70	2	2871.00	2003-02-24	1	2	2003	95	S10_1678	
1	34	81.35	5	2765.90	2003-05-07	2	5	2003	95	S10_1678	
2	41	94.74	2	3884.34	2003-07-01	3	7	2003	95	S10_1678	M
3	45	83.26	6	3746.70	2003-08-25	3	8	2003	95	S10_1678	M
4	49	100.00	14	5205.27	2003-10-10	4	10	2003	95	S10_1678	M
...	...	...	...	...	...	...	...	...	...	...	
2818	20	100.00	15	2244.40	2004-12-02	4	12	2004	54	S72_3212	
2819	29	100.00	1	3978.51	2005-01-31	1	1	2005	54	S72_3212	M
2820	43	100.00	4	5417.57	2005-03-01	1	3	2005	54	S72_3212	M
2821	34	62.24	1	2116.16	2005-03-28	1	3	2005	54	S72_3212	
2822	47	65.52	9	3079.44	2005-05-06	2	5	2005	54	S72_3212	M

2823 rows × 37 columns

```
In [ ]:
sales_df = dummies('DEALSIZE')
sales_df
```

Out[ ]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	QTR_ID	MONTH_ID	YEAR_ID	MSRP	PRODUCTCODE	Aus
0	30	95.70	2	2871.00	2003-02-24	1	2	2003	95	S10_1678	
1	34	81.35	5	2765.90	2003-05-07	2	5	2003	95	S10_1678	
2	41	94.74	2	3884.34	2003-07-01	3	7	2003	95	S10_1678	
3	45	83.26	6	3746.70	2003-08-25	3	8	2003	95	S10_1678	
4	49	100.00	14	5205.27	2003-10-10	4	10	2003	95	S10_1678	
...	...	...	...	...	...	...	...	...	...	...	
2818	20	100.00	15	2244.40	2004-12-02	4	12	2004	54	S72_3212	
2819	29	100.00	1	3978.51	2005-01-31	1	1	2005	54	S72_3212	
2820	43	100.00	4	5417.57	2005-03-01	1	3	2005	54	S72_3212	
2821	34	62.24	1	2116.16	2005-03-28	1	3	2005	54	S72_3212	
2822	47	65.52	9	3079.44	2005-05-06	2	5	2005	54	S72_3212	

2823 rows × 39 columns

```
In [ ]:
y = pd.Categorical(sales_df['PRODUCTCODE'])
y
```

```
Out[ ]:
['S10_1678', 'S10_1678', 'S10_1678', 'S10_1678', 'S10_1678', ..., 'S72_3212', 'S72_3212', 'S72_3212', 'S72_321
2', 'S72_3212']
Length: 2823
Categories (109, object): ['S10_1678', 'S10_1949', 'S10_2016', 'S10_4698', ..., 'S700_3962',
                           'S700_4002', 'S72_1253', 'S72_3212']
```

```
In [ ]:
y = pd.Categorical(sales_df['PRODUCTCODE']).codes
y
```

```
Out[ ]:
array([  0,   0,   0, ..., 108, 108, 108], dtype=int8)
```

```
In [ ]:
# Dado que el número de código de producto tiene 109 valores únicos, si agregamos variables one-hot,
# serían 109 columnas adicionales, podemos evitarlo usando la codificación categórica

sales_df['PRODUCTCODE'] = pd.Categorical(sales_df['PRODUCTCODE']).codes
```

```
In [ ]:
sales_df
```

Out[ ]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	QTR_ID	MONTH_ID	YEAR_ID	MSRP	PRODUCTCODE	Aus
0	30	95.70	2	2871.00	2003-02-24	1	2	2003	95	0	
1	34	81.35	5	2765.90	2003-05-07	2	5	2003	95	0	
2	41	94.74	2	3884.34	2003-07-01	3	7	2003	95	0	
3	45	83.26	6	3746.70	2003-08-25	3	8	2003	95	0	
4	49	100.00	14	5205.27	2003-10-10	4	10	2003	95	0	
...	...	...	...	...	...	...	...	...	...	...	
2818	20	100.00	15	2244.40	2004-12-02	4	12	2004	54	108	
2819	29	100.00	1	3978.51	2005-01-31	1	1	2005	54	108	
2820	43	100.00	4	5417.57	2005-03-01	1	3	2005	54	108	
2821	34	62.24	1	2116.16	2005-03-28	1	3	2005	54	108	
2822	47	65.52	9	3079.44	2005-05-06	2	5	2005	54	108	

2823 rows × 39 columns

```
In [ ]:
# Agrupamos los datos según la fecha del pedido
sales_df_group = sales_df.groupby(by = "ORDERDATE").sum()
sales_df_group
```

Out[ ]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	QTR_ID	MONTH_ID	YEAR_ID	MSRP	PRODUCTCODE	Australia
ORDERDATE										
2003-01-06	151	288.78	10	12133.25	4	4	8012	363	174.0	0
2003-01-09	142	284.96	10	11432.34	4	4	8012	372	181.0	0
2003-01-10	80	150.14	3	6864.05	2	2	4006	155	37.0	0
2003-01-29	541	1417.54	136	54702.00	16	16	32048	1695	723.0	0
2003-01-31	443	1061.89	91	44621.96	13	13	26039	1365	720.0	0
...	...	...	...	...	...	...	...	...	...	...
2005-05-13	259	561.18	21	31821.90	12	30	12030	728	101.0	0
2005-05-17	509	1269.43	105	59475.10	28	70	28070	1669	462.0	0
2005-05-29	607	1148.40	94	51233.18	30	75	30075	1328	797.0	13
2005-05-30	187	542.16	18	14578.75	14	35	14035	618	265.0	0
2005-05-31	696	1561.40	112	78918.03	38	95	38095	2065	899.0	0

252 rows × 38 columns

```
In [ ]:
fig = px.line(x = sales_df_group.index, y = sales_df_group.SALES, title = 'Sales')
fig.show()
```

```
In [ ]:
# Podemos eliminar 'ORDERDATE' y quedarnos con el resto de datos relacionados con la fecha como 'MONTH'
sales_df.drop("ORDERDATE", axis = 1, inplace = True)
sales_df.shape
```

```
Out[ ]:
(2823, 38)
```

```
In [ ]:
plt.figure(figsize = (15, 15))
corr_matrix = sales_df.iloc[:, :9].corr()
sns.heatmap(corr_matrix, annot = True, cbar = False)
```

```
Out[ ]:
<matplotlib.axes._subplots.AxesSubplot at 0x7f91b57184e0>
```


```
In [ ]:
# Parece que el ID del trimestre y los ID del mes están altamente correlacionados
# Dejemos 'QTR_ID' (o 'MONTH_ID')
sales_df.drop("QTR_ID", axis = 1, inplace = True)
sales_df.shape
```

```
Out[ ]:
(2823, 37)
```

```
In [ ]:
# Representar las distribuciones
# Distplot muestra el (1) histograma, (2) kde plot y (3) rug plot.
# (1) Histograma: es una visualización gráfica de datos que usa barras con varias alturas. Cada barra agrupa lo
# (2) Densidad: la estimación de la función de densidad se utiliza para visualizar la densidad de probabilidad
# (3) Gráfico de alfombra: gráfico de datos para una única variable cuantitativa, que se muestra como marcas a

import plotly.figure_factory as ff

plt.figure(figsize = (10, 10))

for i in range(8):
  if sales_df.columns[i] != 'ORDERLINENUMBER':
    col_name = sales_df.columns[i]
    fig = ff.create_distplot([sales_df[col_name].apply(lambda x: float(x))], ['distplot'])
    fig.update_layout(title_text = col_name)
    fig.show()
```

```
Epoch 1/500
Epoch 2/500
Epoch 3/500
Epoch 4/500
Epoch 5/500
Epoch 6/500
Epoch 7/500
Epoch 8/500
Epoch 9/500
Epoch 10/500
Epoch 11/500
Epoch 12/500
Epoch 13/500
Epoch 14/500
Epoch 15/500
Epoch 16/500
Epoch 17/500
Epoch 18/500
Epoch 19/500
Epoch 20/500
Epoch 21/500
Epoch 22/500
Epoch 23/500
Epoch 24/500
Epoch 25/500
Epoch 26/500
Epoch 27/500
Epoch 28/500
Epoch 29/500
Epoch 30/500
Epoch 31/500
Epoch 32/500
Epoch 33/500
Epoch 34/500
Epoch 35/500
Epoch 36/500
Epoch 37/500
Epoch 38/500
Epoch 39/500
Epoch 40/500
Epoch 41/500
Epoch 42/500
Epoch 43/500
Epoch 44/500
Epoch 45/500
Epoch 46/500
Epoch 47/500
Epoch 48/500
Epoch 49/500
Epoch 50/500
Epoch 51/500
Epoch 52/500
Epoch 53/500
Epoch 54/500
Epoch 55/500
Epoch 56/500
Epoch 57/500
Epoch 58/500
Epoch 59/500
Epoch 60/500
Epoch 61/500
Epoch 62/500
Epoch 63/500
Epoch 64/500
Epoch 65/500
Epoch 66/500
Epoch 67/500
Epoch 68/500
Epoch 69/500
Epoch 70/500
Epoch 71/500
Epoch 72/500
Epoch 73/500
Epoch 74/500
Epoch 75/500
Epoch 76/500
Epoch 77/500
Epoch 78/500
Epoch 79/500
Epoch 80/500
Epoch 81/500
Epoch 82/500
Epoch 83/500
Epoch 84/500
Epoch 85/500
Epoch 86/500
Epoch 87/500
Epoch 88/500
Epoch 89/500
Epoch 90/500
Epoch 91/500
Epoch 92/500
Epoch 93/500
Epoch 94/500
Epoch 95/500
Epoch 96/500
Epoch 97/500
Epoch 98/500
Epoch 99/500
Epoch 100/500
Epoch 101/500
Epoch 102/500
Epoch 103/500
Epoch 104/500
Epoch 105/500
Epoch 106/500
Epoch 107/500
Epoch 108/500
Epoch 109/500
Epoch 110/500
Epoch 111/500
Epoch 112/500
Epoch 113/500
Epoch 114/500
Epoch 115/500
Epoch 116/500
Epoch 117/500
Epoch 118/500
Epoch 119/500
Epoch 120/500
Epoch 121/500
Epoch 122/500
Epoch 123/500
Epoch 124/500
Epoch 125/500
Epoch 126/500
Epoch 127/500
Epoch 128/500
Epoch 129/500
Epoch 130/500
Epoch 131/500
Epoch 132/500
Epoch 133/500
Epoch 134/500
Epoch 135/500
Epoch 136/500
Epoch 137/500
Epoch 138/500
Epoch 139/500
Epoch 140/500
Epoch 141/500
Epoch 142/500
Epoch 143/500
Epoch 144/500
Epoch 145/500
Epoch 146/500
Epoch 147/500
Epoch 148/500
Epoch 149/500
Epoch 150/500
Epoch 151/500
Epoch 152/500
Epoch 153/500
Epoch 154/500
Epoch 155/500
Epoch 156/500
Epoch 157/500
Epoch 158/500
Epoch 159/500
Epoch 160/500
Epoch 161/500
Epoch 162/500
Epoch 163/500
Epoch 164/500
Epoch 165/500
Epoch 166/500
Epoch 167/500
Epoch 168/500
Epoch 169/500
Epoch 170/500
Epoch 171/500
Epoch 172/500
Epoch 173/500
Epoch 174/500
Epoch 175/500
Epoch 176/500
Epoch 177/500
Epoch 178/500
Epoch 179/500
Epoch 180/500
Epoch 181/500
Epoch 182/500
Epoch 183/500
Epoch 184/500
Epoch 185/500
Epoch 186/500
Epoch 187/500
Epoch 188/500
Epoch 189/500
Epoch 190/500
Epoch 191/500
Epoch 192/500
Epoch 193/500
Epoch 194/500
Epoch 195/500
Epoch 196/500
Epoch 197/500
Epoch 198/500
Epoch 199/500
Epoch 200/500
Epoch 201/500
Epoch 202/500
Epoch 203/500
Epoch 204/500
Epoch 205/500
Epoch 206/500
Epoch 207/500
Epoch 208/500
Epoch 209/500
Epoch 210/500
Epoch 211/500
Epoch 212/500
Epoch 213/500
Epoch 214/500
Epoch 215/500
Epoch 216/500
Epoch 217/500
Epoch 218/500
Epoch 219/500
Epoch 220/500
Epoch 221/500
Epoch 222/500
Epoch 223/500
Epoch 224/500
Epoch 225/500
Epoch 226/500
Epoch 227/500
Epoch 228/500
Epoch 229/500
Epoch 230/500
Epoch 231/500
Epoch 232/500
Epoch 233/500
Epoch 234/500
Epoch 235/500
Epoch 236/500
Epoch 237/500
Epoch 238/500
Epoch 239/500
Epoch 240/500
Epoch 241/500
Epoch 242/500
Epoch 243/500
Epoch 244/500
Epoch 245/500
Epoch 246/500
Epoch 247/500
Epoch 248/500
Epoch 249/500
Epoch 250/500
Epoch 251/500
Epoch 252/500
Epoch 253/500
Epoch 254/500
Epoch 255/500
Epoch 256/500
Epoch 257/500
Epoch 258/500
Epoch 259/500
Epoch 260/500
Epoch 261/500
Epoch 262/500
Epoch 263/500
Epoch 264/500
Epoch 265/500
Epoch 266/500
Epoch 267/500
Epoch 268/500
Epoch 269/500
Epoch 270/500
Epoch 271/500
Epoch 272/500
Epoch 273/500
Epoch 274/500
Epoch 275/500
Epoch 276/500
Epoch 277/500
Epoch 278/500
Epoch 279/500
Epoch 280/500
Epoch 281/500
Epoch 282/500
Epoch 283/500
Epoch 284/500
Epoch 285/500
Epoch 286/500
Epoch 287/500
Epoch 288/500
Epoch 289/500
Epoch 290/500
Epoch 291/500
Epoch 292/500
Epoch 293/500
Epoch 294/500
Epoch 295/500
Epoch 296/500
Epoch 297/500
Epoch 298/500
Epoch 299/500
Epoch 300/500
Epoch 301/500
Epoch 302/500
Epoch 303/500
Epoch 304/500
Epoch 305/500
Epoch 306/500
Epoch 307/500
Epoch 308/500
Epoch 309/500
Epoch 310/500
Epoch 311/500
Epoch 312/500
Epoch 313/500
Epoch 314/500
Epoch 315/500
Epoch 316/500
Epoch 317/500
Epoch 318/500
Epoch 319/500
Epoch 320/500
Epoch 321/500
Epoch 322/500
Epoch 323/500
Epoch 324/500
Epoch 325/500
Epoch 326/500
Epoch 327/500
Epoch 328/500
Epoch 329/500
Epoch 330/500
Epoch 331/500
Epoch 332/500
Epoch 333/500
Epoch 334/500
Epoch 335/500
Epoch 336/500
Epoch 337/500
Epoch 338/500
Epoch 339/500
Epoch 340/500
Epoch 341/500
Epoch 342/500
Epoch 343/500
Epoch 344/500
Epoch 345/500
Epoch 346/500
Epoch 347/500
Epoch 348/500
Epoch 349/500
Epoch 350/500
Epoch 351/500
Epoch 352/500
Epoch 353/500
Epoch 354/500
Epoch 355/500
Epoch 356/500
Epoch 357/500
Epoch 358/500
Epoch 359/500
Epoch 360/500
Epoch 361/500
Epoch 362/500
Epoch 363/500
Epoch 364/500
Epoch 365/500
Epoch 366/500
Epoch 367/500
Epoch 368/500
Epoch 369/500
Epoch 370/500
Epoch 371/500
Epoch 372/500
Epoch 373/500
Epoch 374/500
Epoch 375/500
Epoch 376/500
Epoch 377/500
Epoch 378/500
Epoch 379/500
Epoch 380/500
Epoch 381/500
Epoch 382/500
Epoch 383/500
Epoch 384/500
Epoch 385/500
Epoch 386/500
Epoch 387/500
Epoch 388/500
Epoch 389/500
Epoch 390/500
Epoch 391/500
Epoch 392/500
Epoch 393/500
Epoch 394/500
Epoch 395/500
Epoch 396/500
Epoch 397/500
Epoch 398/500
Epoch 399/500
Epoch 400/500
Epoch 401/500
Epoch 402/500
Epoch 403/500
Epoch 404/500
Epoch 405/500
Epoch 406/500
Epoch 407/500
Epoch 408/500
Epoch 409/500
Epoch 410/500
Epoch 411/500
Epoch 412/500
Epoch 413/500
Epoch 414/500
Epoch 415/500
Epoch 416/500
Epoch 417/500
Epoch 418/500
Epoch 419/500
Epoch 420/500
Epoch 421/500
Epoch 422/500
Epoch 423/500
Epoch 424/500
Epoch 425/500
Epoch 426/500
Epoch 427/500
Epoch 428/500
Epoch 429/500
Epoch 430/500
Epoch 431/500
Epoch 432/500
Epoch 433/500
Epoch 434/500
Epoch 435/500
Epoch 436/500
Epoch 437/500
Epoch 438/500
Epoch 439/500
Epoch 440/500
Epoch 441/500
Epoch 442/500
Epoch 443/500
Epoch 444/500
Epoch 445/500
Epoch 446/500
Epoch 447/500
Epoch 448/500
Epoch 449/500
Epoch 450/500
Epoch 451/500
Epoch 452/500
Epoch 453/500
Epoch 454/500
Epoch 455/500
Epoch 456/500
Epoch 457/500
Epoch 458/500
Epoch 459/500
Epoch 460/500
Epoch 461/500
Epoch 462/500
Epoch 463/500
Epoch 464/500
Epoch 465/500
Epoch 466/500
Epoch 467/500
Epoch 468/500
Epoch 469/500
Epoch 470/500
Epoch 471/500
Epoch 472/500
Epoch 473/500
Epoch 474/500
Epoch 475/500
Epoch 476/500
Epoch 477/500
Epoch 478/500
Epoch 479/500
Epoch 480/500
Epoch 481/500
Epoch 482/500
Epoch 483/500
Epoch 484/500
Epoch 485/500
Epoch 486/500
Epoch 487/500
Epoch 488/500
Epoch 489/500
Epoch 490/500
Epoch 491/500
Epoch 492/500
Epoch 493/500
Epoch 494/500
Epoch 495/500
Epoch 496/500
Epoch 497/500
Epoch 498/500
Epoch 499/500
Epoch 500/500
```

Out[ ]: <tensorflow.python.keras.callbacks.History at 0x7f91b88ae668>

In [ ]:
```
autoencoder.save_weights('autoencoder_1.h5')
```

In [ ]:
```
pred = encoder.predict(sales_df_scaled)
```

In [ ]:
```
pred
```

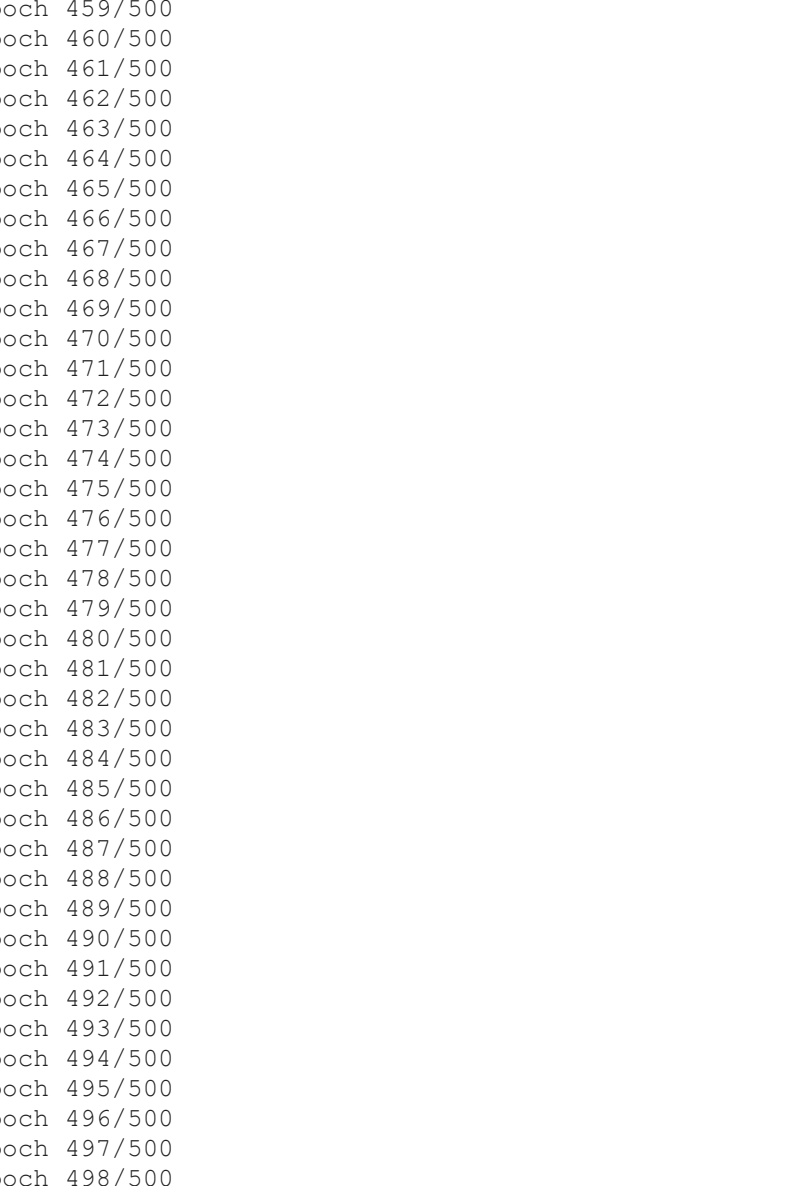
Out[ ]:
```
array([[ 0.        ,  0.9572773 ,  2.052799  , ...,  2.8538415 ,
         5.177384  ,  3.7766817 ],
       [ 0.        ,  1.474308  ,  3.2650466 , ...,  1.1344082 ,
         4.54118   ,  6.102257  ],
       [ 0.        ,  1.8946625 ,  6.611377  , ...,  1.6915857 ,
         7.6524625 ,  6.197023  ],
       ...,
       [ 0.        ,  5.2380247 ,  1.7845033 , ...,  8.191807  ,
         2.2916932 ,  4.8318996 ],
       [ 0.        ,  8.066888  ,  2.925734  , ...,  3.8033655 ,
         7.006839  , 14.542607  ],
       [ 0.        ,  5.8996243 ,  0.51634616, ...,  7.410111  ,
         4.386255  ,  3.2716434 ]], dtype=float32)
```

In [ ]:
```
scores = []

range_values = range(1, 15)

for i in range_values:
  kmeans = KMeans(n_clusters = i)
  kmeans.fit(pred)
  scores.append(kmeans.inertia_)

plt.plot(range_values, scores, 'bx-')
plt.title('Encontrar el número correcto de clústers')
plt.xlabel('Nº de Clústers')
plt.ylabel('WCSS')
plt.show()
```



In [ ]:
```
kmeans = KMeans(3)
kmeans.fit(pred)
labels = kmeans.labels_
y_kmeans = kmeans.fit_predict(sales_df_scaled)
```

In [ ]:
```
df_cluster_dr = pd.concat([sales_df, pd.DataFrame({'cluster':labels})], axis = 1)
df_cluster_dr.head()
```

Out[ ]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	MONTH_ID	YEAR_ID	MSRP	PRODUCTCODE	Australia	Austria	Belgium
0	30	95.70	2.0	2871.00	2	2003	95	0	0	0	0
1	34	81.35	5.0	2765.90	5	2003	95	0	0	0	0
2	41	94.74	2.0	3884.34	7	2003	95	0	0	0	0
3	45	83.26	6.0	3746.70	8	2003	95	0	0	0	0
4	49	100.00	14.0	5205.27	10	2003	95	0	0	0	0

In [ ]:
```
cluster_centers = pd.DataFrame(data = kmeans.cluster_centers_, columns = [sales_df.columns])
cluster_centers
```

Out[ ]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	MONTH_ID	YEAR_ID	MSRP	PRODUCTCODE	Australia	Austria	B
0	1.245428	0.800220	-0.259579	2.573861	-0.088008	0.136857	1.430265	-0.871802	-0.084647	0.043374	-0.
1	0.293073	0.580208	-0.010805	0.458608	0.003368	-0.012482	0.376634	-0.165542	-0.013717	0.010642	-0.
2	-0.468912	-0.724370	0.043454	-0.810304	0.007142	-0.003285	-0.581757	0.285478	0.025174	-0.016801	0.

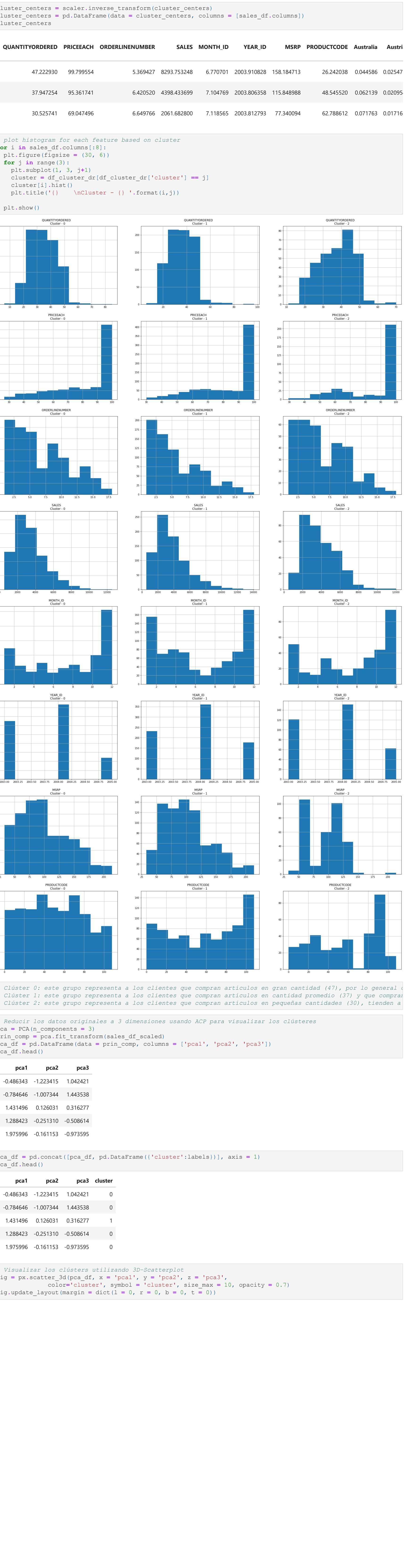
In [ ]:
```
cluster_centers = scaler.inverse_transform(cluster_centers)
cluster_centers = pd.DataFrame(data = cluster_centers, columns = [sales_df.columns])
cluster_centers
```

Out[ ]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	MONTH_ID	YEAR_ID	MSRP	PRODUCTCODE	Australia	Austri
0	47.222930	99.799554	5.369427	8293.753248	6.770701	2003.910828	158.184713	26.242038	0.044586	0.02547
1	37.947254	95.361741	6.420520	4398.433699	7.104769	2003.806358	115.848988	48.545520	0.062139	0.02095
2	30.525741	69.047496	6.649766	2061.682800	7.118565	2003.812793	77.340094	62.788612	0.071763	0.01716

In [ ]:
```
# plot histogram for each feature based on cluster
for i in sales_df.columns[:8]:
  plt.figure(figsize = (30, 6))
  for j in range(3):
    plt.subplot(1, 3, j+1)
    cluster = df_cluster_dr[df_cluster_dr['cluster'] == j]
    cluster[i].hist()
    plt.title('{}    \nCluster - {} '.format(i,j))

  plt.show()
```



In [ ]:
```
# Cluster 0: este grupo representa a los clientes que compran artículos en gran cantidad (47), por lo general a
# Cluster 1: este grupo representa a los clientes que compran artículos en cantidad promedio (37) y que compran
# Cluster 2: este grupo representa a los clientes que compran artículos en pequeñas cantidades (30), tienden a
```

In [ ]:
```
# Reducir los datos originales a 3 dimensiones usando ACP para visualizar los clústeres
pca = PCA(n_components = 3)
prin_comp = pca.fit_transform(sales_df_scaled)
pca_df = pd.DataFrame(data = prin_comp, columns = ['pca1', 'pca2', 'pca3'])
pca_df.head()
```

Out[ ]:

	pca1	pca2	pca3
0	-0.486343	-1.223415	1.042421
1	-0.784646	-1.007344	1.443538
2	1.431496	0.126031	0.316277
3	1.288423	-0.251310	-0.508614
4	1.975996	-0.161553	-0.973595

In [ ]:
```
pca_df = pd.concat([pca_df, pd.DataFrame({'cluster':labels})], axis = 1)
pca_df.head()
```

Out[ ]:

	pca1	pca2	pca3	cluster
0	-0.486343	-1.223415	1.042421	0
1	-0.784646	-1.007344	1.443538	0
2	1.431496	0.126031	0.316277	1
3	1.288423	-0.251310	-0.508614	0
4	1.975996	-0.161553	-0.973595	0

In [ ]:
```
# Visualizar los clusters utilizando 3D-Scatterplot
fig = px.scatter_3d(pca_df, x = 'pca1', y = 'pca2', z = 'pca3',
              color='cluster', symbol = 'cluster', size_max = 10, opacity = 0.7)
fig.update_layout(margin = dict(l = 0, r = 0, b = 0, t = 0))
```