Description

Your company wishes to better understand the evolution of our sales globally.

More specifically, it is unclear whether we should create a VIP program for our clients, focus on particular customer segments or which markets, if any, we should pay more attention to. We also do not know if higher category shipments are being profitable or we are losing money instead and should drop them.

Try to help your company improve its situation.

Submit a pdf file applying machine learning techniques to try and extract insights from the sales data. For each technique, you should include

(i) a screenshot of the implementation in jupyter
(ii) a visualization of the model or its results (if it is possible)
(iii) an interpretation of the results obtained by the model (accuracy, coefficients)

Remember that the file submitted must include the full name(s) of the student(s) involved.

To do:

Using the data from the previous session OR from the ORACLE database, add a new column 'VIP' with two values: 0 and 1. The value for VIP will be 1 for those in the list of top 50 customers according to the profit made thanks to them, 0 for the rest.

Train and interpret the output of the following models:

- Decision Tree with target to predict: VIP
- Decision Tree Regressor with target to predict: Profit
- Linear Regression with target to predict: Profit
- Kmeans with scaler

Preparación de datos

```
#imports
import oracledb
import pandas as pd
import io
import pandas as pd
import numpy as np
```

```
#BD CONECTION
connection = oracledb.connect(user="UBD1463", password='              , host="diana.lcc.uma.es", sid="ATENEA")
cursor = connection.cursor()
```

```
#Obtengo los 50 clientes que mas profit generan:
cursor.execute("""
    SELECT C.IDCUSTOMER  , SUM(O.PROFIT) AS "TOTAL PROFIT"
    FROM CUSTOMER C  JOIN FACT_ORDER O ON C.IDCUSTOMER = O.CUSTOMER_IDCUSTOMER
    GROUP BY C.IDCUSTOMER
    ORDER BY 2 DESC
    FETCH FIRST 50 ROWS ONLY
    """)
top_customers = {row[0] for row in cursor.fetchall()}
```

Ilustración 1

1. Hacemos la conexión en la base de datos
2. Obtenemos los 50 mejores clientes con la consulta de SQL.

```
#Obtengo los datos de la base de datos
cursor.execute("""
    SELECT
        O.*,
        C.MARKET,
        P.CATEGORY,
        D.YEAR AS YEAR_ORDER, D.MONTH AS MONTH_ORDER,
        D2.YEAR AS YEAR_SHIP, D2.MONTH AS MONTH_SHIP
    FROM FACT_ORDER O
        JOIN  CUSTOMER C ON O.CUSTOMER_IDCUSTOMER = C.IDCUSTOMER
        JOIN PRODUCT P ON O.PRODUCT_IDPRODUCT = P.IDPRODUCT
        JOIN  DATES D ON O.DATE_IDORDER = D.IDDATE
        JOIN DATES D2 ON O.DATE_IDSHIP = D2.IDDATE

""")

# Obtener los nombres de las columnas
columns = [col[0] for col in cursor.description]

# Recuperar los datos
rows = cursor.fetchall()

df = pd.DataFrame(rows, columns=columns)
df['VIP'] = df['CUSTOMER_IDCUSTOMER'].isin(top_customers).astype(int)
```

Ilustración 2

Obtenemos los registros de los pedidos. Obtenemos el mercado, la categoría del producto, el mes y año de pedido y de envío y las medidas del fact (*profit, shipment cost, sales, quantity*). Además, tendremos los *ids* de prioridad, *shipment y returned* que no los obtenemos el *SQL* porque nos interesa tener valores no numéricos para los entrenamientos.

Además, insertamos el atributo VIP.

```
# Cargamos las librerias generales de sklearn
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn import metrics
```

```
# Preparamos los datos para el SVM
from sklearn import svm

classifierSVM = svm.SVC(kernel="rbf",C=1)

clfSVM = make_pipeline(StandardScaler(), classifierSVM)



#Reemplazamos los valores de market
X = df.replace(to_replace="Africa",value=0)
X = X.replace(to_replace="LATAM", value=1)
X = X.replace(to_replace="Asia Pacific",value=2)
X = X.replace(to_replace="USCA",value=3)
X = X.replace(to_replace="Europe",value=4)

#Reemplazamos los valores de Category product
X = X.replace(to_replace="Furniture", value=0)
X = X.replace(to_replace="Technology", value=1)
X = X.replace(to_replace="Office Supplies", value=2)


# Quitamos los Ids
X = X.drop(["PRODUCT_IDPRODUCT"],axis=1)
X = X.drop(["IDFACT_ORDER"],axis=1)

X = X.drop(["DATE_IDSHIP"],axis=1)
X = X.drop(["DATE_IDORDER"],axis=1)
X = X.drop(["CUSTOMER_IDCUSTOMER"],axis=1)

XReg = X
```

Ilustración 3

Le damos un valor numérico cada mercado y un valor numérico a cada producto de categoría. Luego eliminamos los ids que no nos sirven.

ıS	SALES	QUANTITY	DISCOUNT	PROFIT	SHIPPING_COST	PREPARATION_TIME	MARKET	CATEGORY	YEAR_ORDER	MONTH_ORDER	YEAR_SHIP	MONTH_SHIP	VIP
2	105.72	4.0	0.0	43.32	6.73	6.0	4	2	2013	4	2013	4	0
2	6.45	1.0	0.0	1.65	1.37	6.0	4	2	2013	4	2013	4	0
2	182.07	1.0	0.0	40.05	59.24	0.0	4	1	2014	4	2014	4	0
2	85.26	1.0	0.0	0.84	4.06	5.0	4	1	2014	12	2014	12	0
2	553.92	4.0	0.0	22.08	206.56	2.0	4	1	2014	8	2014	8	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2	332.28	2.0	0.0	43.14	60.06	5.0	2	0	2015	1	2015	1	0
2	362.40	1.0	0.0	39.84	42.22	5.0	2	0	2015	1	2015	1	0
2	132.48	1.0	0.0	14.55	18.63	5.0	2	0	2015	1	2015	1	0
2	103.56	2.0	0.0	38.28	8.12	5.0	2	2	2015	1	2015	1	0
2	19.29	1.0	0.0	1.14	3.95	5.0	2	2	2015	1	2015	1	0

Ilustración 4 Dataframe con los resultados

Decision Tree with target to predict: VIP

```
# We can also generate an overall data profiling using ydata_profiling library
# Be mindful about the number of columns as it can cause the library to take a long time to generate the report
# If the dataset is too large, we can use minimal=True to speed up the report generation at the cost of missing some information
# Other alternatives, such as interaction targets to be analyzed can be configured as indicated in the documentation (https://docs.profiling.ydata.ai/lat

from ydata_profiling import ProfileReport
report = ProfileReport(X, title="Order profiling", correlations={"auto": {"calculate": True}}, minimal=True)
report
```

Ilustración 5 Report con ydata_profilng

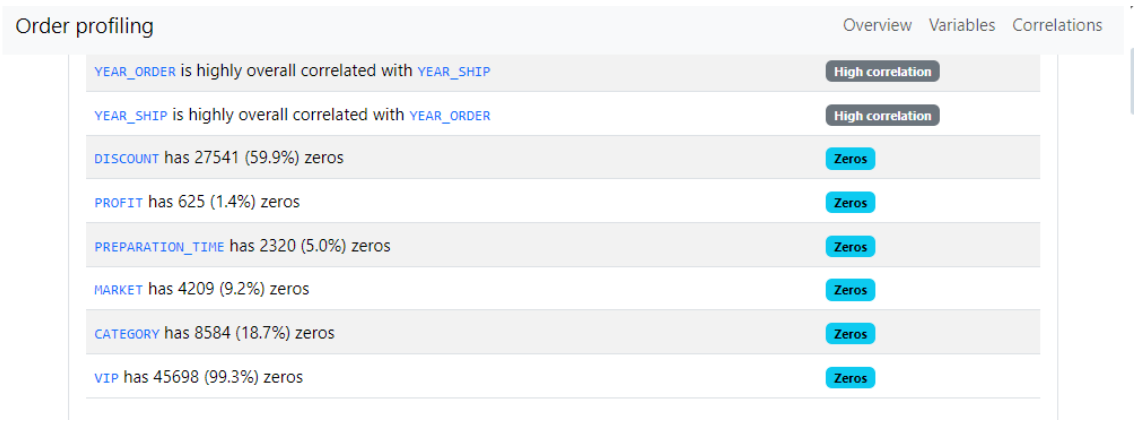
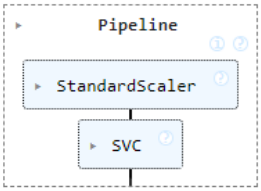


Ilustración 6

Como podemos ver solo el 0.7% de los pedidos provienen de clientes vips lo que va a causar muchas dificultades para entrenar un modelo que nos sea útil.

```
Y = X["VIP"]
X = X.drop(["VIP"],axis=1)
```

```
# Una vez tenemos los datos entrenamos el clasificador. Utilizamos la opción .values para pasar únicamente los valores
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
clfSVM.fit(X_train.values,Y_train.values)
```




```
# Comprobamos el accuracy del clasificador
clfSVM.score(X_test.values, Y_test.values)
```

```
0.9936758893280633
```

Ilustración 7

De hecho, si entrenamos un clasificador veremos que la precisión es casi 1 por lo que puede ser que no nos sea de mucha utilidad. El clasificador podría tener esa gran precisión ya que como hemos visto la mayoría de los pedidos no provienen de clientes VIP. Si nos pusiéramos en la situación de que el

clasificador responde siempre a que el pedido no es VIP, teniendo en cuenta que solo un 0.7% de predidos son VIP aun así obtendría una precisión de 0.993 (como vemos coincide con la precisión del clasificador).

Lo que podríamos hacer es obtener todos los pedidos VIPS y obtener al azar el mismo número de pedidos no VIPS y probar a entrenar el clasificador.

```
from sklearn.utils import resample


# Separar los pedidos VIP y no VIP
vip_orders = X[X['VIP'] == 1]
non_vip_orders = X[X['VIP'] == 0]

# Obtener el número de pedidos VIP
n_vip = len(vip_orders)

# Submuestrear aleatoriamente los pedidos no VIP para que coincidan con el número de pedidos VIP
non_vip_undersampled = resample(non_vip_orders,
                                replace=False,
                                n_samples=n_vip,
                                random_state=42)

# Combinar los pedidos VIP con los no VIP submuestreados
df_balanced = pd.concat([vip_orders, non_vip_undersampled])

# Mezclar el DataFrame resultante
X = df_balanced.sample(frac=1, random_state=42).reset_index(drop=True)

# Recuperamos la etiqueta de clase a predecir
Y = df_balanced["VIP"]
X = X.drop(["VIP"],axis=1)
```

```
# Una vez tenemos los datos entrenamos el clasificador. Utilizamos la opción .values para pasar únicamente los valores
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
clfSVM.fit(X_train.values,Y_train.values)
```

Ilustración 8

```
# Comprobamos el accuracy del clasificador
clfSVM.score(X_test.values, Y_test.values)
```

```
0.485
```

Ilustración 9

Volvemos a entrenar el clasificador y obtenemos 0.485. Es un valor de precisión muy bajo para sacar alguna conclusión interesante mediante un árbol de decisión. Es posible que el valor sea bajo la precisión porque el total de pedidos en el *dataset* es tan solo de 604 pedidos, es demasiado pequeño.

Si usáramos el clasificado entrenado con todos los datos, aunque probablemente esté muy lejos de ser ideal. Podríamos crear el árbol de decisión de la siguiente manera:

```
# Ahora vamos a entrenar un árbol de decisión
# Para poder pintarlo después deberemos cambiar los valores numéricos de la etiqueta de clase por cadenas
from sklearn import tree
YCat = Y.replace(to_replace=0, value="No Vip")
YCat = YCat.replace(to_replace=1, value="Vip")
XTree_train, XTree_test, YTree_train, YTree_test = train_test_split(X,YCat, test_size=0.33, random_state=42)
classifierTree = tree.DecisionTreeClassifier(max_depth=3, criterion="entropy")
classifierTree.fit(XTree_train.values, YTree_train.values)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

```
# Analizamos la bondad del clasificador con el conjunto de test
classifierTree.score(XTree_test.values,YTree_test.values)
```

```
0.99433465085639
```

Ilustración 10

De esta forma podríamos obtener el árbol de decisión. En mi caso he puesto 3 niveles de profundidad.

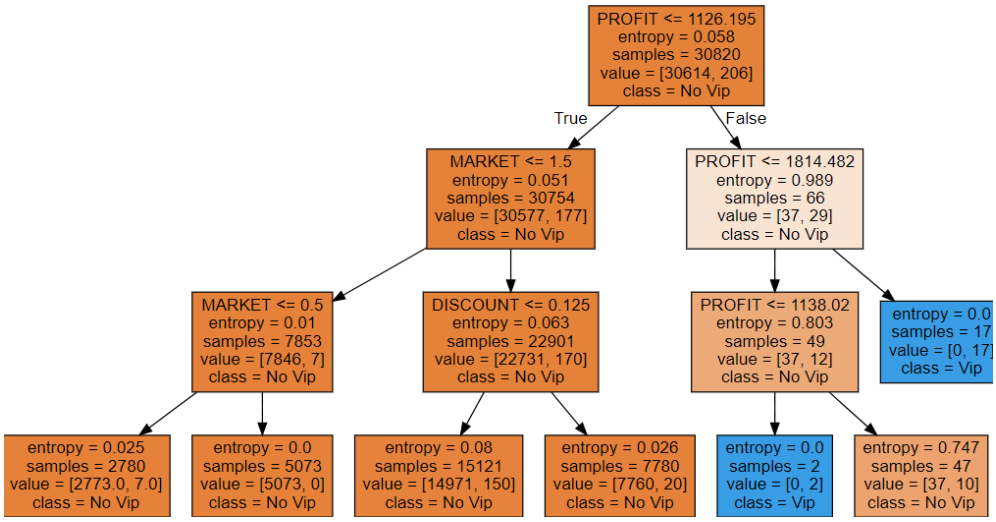


Ilustración 11

Según este calificador lo más probable es que un pedido sea de un cliente vip si tiene un profit entre **1138.02 y 1814.482**.

Si obtuviéramos el árbol de decisión para el otro calificador con los datos balanceados de los pedidos con un 50 % vip y 50% no vip(aunque es poco preciso) obtendríamos esta bondad del clasificador y este diagrama:

```
# Analizamos la bondad del clasificador con el conjunto de test
classifierTree.score(XTree_test.values,YTree_test.values)
```

```
0.515
```

Ilustración 12

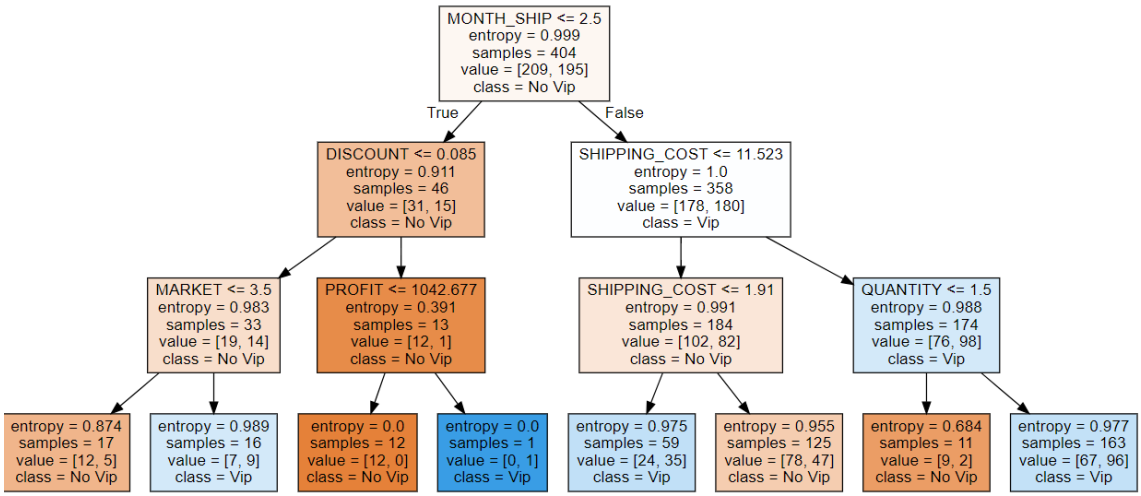


Ilustración 13

Según este árbol de decisión

- Clientes con MONTH_SHIP > 2.5 (es decir de Marzo en adelante) y SHIPPING_COST <= 11.523 tienen alta probabilidad de ser VIP.
- Clientes con QUANTITY <= 1.5 también tienden a ser VIP.
- Clientes con PROFIT > 1042.677 pueden ser VIP, pero son casos raros.
- Los clientes con DISCOUNT <= 0.085 y PROFIT bajo tienen menos probabilidades de ser VIP.

Decision Tree Regressor with target to predict: Profit

Esta vez vamos a hacer un árbol de regresión para intentar predecir el *profit*. Se entrena el modelo de regresión y especificamos una profundidad de 2 en este caso.

```
# Vamos ahora a entrenar un regresor. Podemos crear uno mediante árboles de decisión
regressionTree = tree.DecisionTreeRegressor(max_depth=2)
YReg = X["PROFIT"]
XReg = XReg.drop("PROFIT",axis=1)
regressionTree.fit(XReg.values,YReg.values)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=2)
```

Ilustración 14

Podemos gráfica el resultado de la siguiente manera:

```
dot_data_reg = tree.export_graphviz(regressionTree, feature_names = XReg.columns, out_file=None)
graphReg = graphviz.Source(dot_data_reg)
graphReg
```

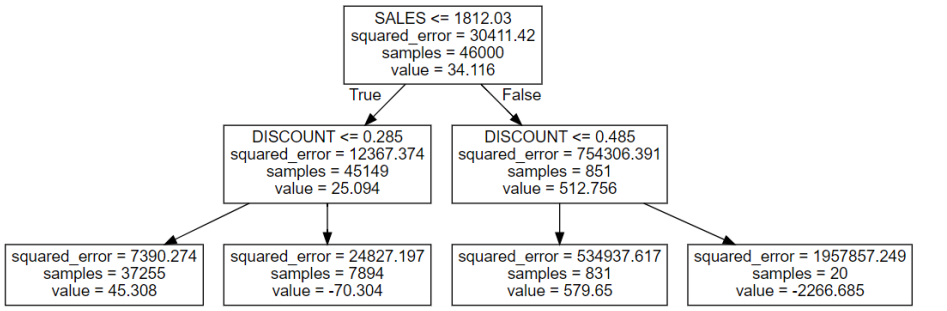


Ilustración 15

El modelo encuentra que los beneficios son generalmente más altos cuando *sales* es bajo y *discount* es bajo. Sin embargo, en algunos casos con altos descuentos (DISCOUNT > 0.485), los beneficios pueden volverse negativos. Las muestras con SALES > 1812.03 son pocas (851 de 46000), pero presentan una gran variabilidad en *profit*, con algunas pérdidas significativas. Un descuento mayor a 0.285 parece estar asociado a pérdidas cuando *sales* es bajo.

```
# Comprobamos el ajuste del regresor
regressionTree.score(XReg.values,YReg.values)
```

```
0.31733250442703353
```

Ilustración 16

Como vemos el coeficiente muy pequeño, Un valor de 0.317 indica que el modelo captura cierta relación entre las variables, pero queda un 68.27% de la variabilidad de los datos sin explicar.

El modelo no es muy preciso, aunque tampoco inútil, probablemente por la profundidad del árbol. Si probamos con un árbol de profundidad 3 obtendríamos lo siguiente:

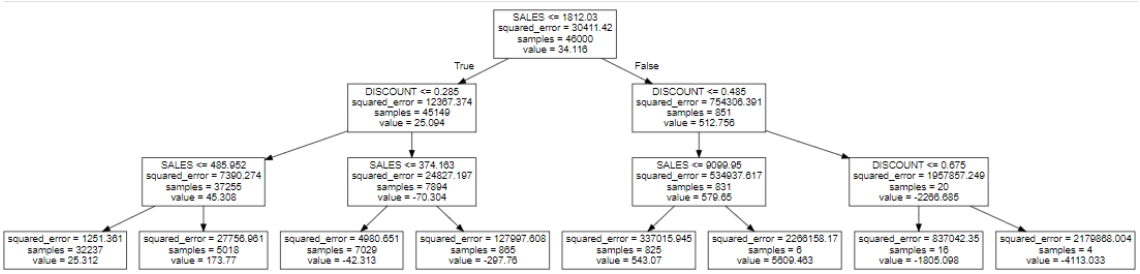


Ilustración 17 Árbol de regresión de 3 profundidades

```
# Comprobamos el ajuste del regresor
regressionTree.score(XReg.values,YReg.values)
```

```
0.5431502185798935
```

Ilustración 18 Nuevo coeficiente calculado

También podríamos ver los pesos de la importancia para cada variable de la siguiente forma:

```
# Normalmente, salvo casos simples o fórmulas puramente numéricas, el regresor no cubrirá la variabilidad de la muestra de forma muy completa
# No obstante, podemos analizar a qué variables le ha dado más peso para extraer posible conocimiento
# Ojo, son pesos de importancia de la variable, no coeficientes (lo veremos más abajo)
regTreeCoef = pd.DataFrame(regressionTree.feature_importances_,XReg.columns, columns=['Importance'])
regTreeCoef
```

Ilustración 19 Importancia para árbol de regresión de profundidad 2

Width = 2

	Importance
SHIPMENT_IDSHIPMENT	0.000000
PRIORITY_IDPRIORITY	0.000000
RETURNS_IDRETURNS	0.000000
SALES	0.447455
QUANTITY	0.000000
DISCOUNT	0.552545
SHIPPING_COST	0.000000
PREPARATION_TIME	0.000000
MARKET	0.000000
CATEGORY	0.000000
YEAR_ORDER	0.000000
MONTH_ORDER	0.000000
YEAR_SHIP	0.000000
MONTH_SHIP	0.000000
VIP	0.000000

Width = 3

	Importance
SHIPMENT_IDSHIPMENT	0.000000
PRIORITY_IDPRIORITY	0.000000
RETURNS_IDRETURNS	0.000000
SALES	0.654746
QUANTITY	0.000000
DISCOUNT	0.345254
SHIPPING_COST	0.000000
PREPARATION_TIME	0.000000
MARKET	0.000000
CATEGORY	0.000000
YEAR_ORDER	0.000000
MONTH_ORDER	0.000000
YEAR_SHIP	0.000000
MONTH_SHIP	0.000000
VIP	0.000000

Ilustración 20 Importancia de cada variable en los árboles de regresión profundidad 2 y 3

Las únicas variables importantes que son importantes parecen ser *sales* y *discount*.

Linear Regression with target to predict: Profit

```
# Vamos ahora a entrenar un regresor lineal de manera muy similar
from sklearn.linear_model import LinearRegression

linearReg = LinearRegression().fit(X.values,Y.values)
```

```
# Y comparamos resultados
linearReg.score(X.values, Y.values)
```

```
0.0469097117436067
```

```
# Obtenemos los coeficientes para comparar
linearRegCoef = pd.DataFrame(linearReg.coef_, X.columns, columns=['Coefficients'])
linearRegCoef
```

Ilustración 21

El modelo de regresión lineal entrenado muestra un coeficiente de determinación 0.0469, lo que indica que solo explica el 4.69% de la variabilidad en los datos, sugiriendo que nuevo es un ajuste pobre.

	Coefficients
SHIPMENT_IDSHIPMENT	-0.001026
PRIORITY_IDPRIORITY	-0.000367
RETURNS_IDRETURNS	0.003288
SALES	0.000021
QUANTITY	-0.000454
DISCOUNT	0.009443
PROFIT	0.000074
SHIPPING_COST	-0.000072
PREPARATION_TIME	-0.000061
MARKET	0.001645
CATEGORY	0.003022
YEAR_ORDER	-0.025846
MONTH_ORDER	-0.001868
YEAR_SHIP	0.025340
MONTH_SHIP	0.002165

Ilustración 22

Los coeficientes muestran el impacto de cada variable en la predicción, donde DISCOUNT (0.009443) tiene un efecto positivo y YEAR_ORDER (-0.025846) uno negativo, entre otros. Lo más probable que debido al bajo desempeño del modelo la relación entre variables no sea lineal.

Kmeans with scaler

Antes de empezar a agrupar por clusters, tenemos que elegir el nº de clusters. Voy a basarme en dos conceptos:

1. El coeficiente de silhouette que mide la separación entre clusters (valores más altos indican mejor agrupación).
2. El método del codo analiza la “inercia” (suma de distancias cuadradas de los puntos al centroide de su cluster). La idea es elegir un número de clusters donde la reducción de la inercia deje de ser significativa (el "codo" de la curva).

Lo hacemos de la siguiente manera:

```python
# Para terminar vamos a intentar hacer clustering de los datos
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

XRegClusters = XReg
```

```python
# Rango de valores de k a probar
k_values = range(2, 11)  # De 2 a 10 clusters

# Listas para almacenar los resultados
inertia = []
silhouette_scores = []

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=5).fit(XRegClusters)
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(XRegClusters, kmeans.labels_))

# Crear DataFrame con los resultados
df_results = pd.DataFrame({
    "Número de Clusters (k)": k_values,
    "Inercia": inertia,
    "Silhouette Score": silhouette_scores
})
```

Ilustración 23

Vamos calculando la inercia y coeficiente de silhouette para cada nº de clusters.

```python
import seaborn as sns
import matplotlib.pyplot as plt
# Graficar con Seaborn
sns.set(style="whitegrid")

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Gráfico del Método del Codo
sns.lineplot(ax=axes[0], data=df_results, x="Número de Clusters (k)", y="Inercia", marker="o", color="b")
axes[0].set_title("Método del Codo")
axes[0].set_xlabel("Número de Clusters (k)")
axes[0].set_ylabel("Inercia")

# Gráfico del Coeficiente de Silhouette
sns.lineplot(ax=axes[1], data=df_results, x="Número de Clusters (k)", y="Silhouette Score", marker="o", color="r")
axes[1].set_title("Coeficiente de Silhouette")
axes[1].set_xlabel("Número de Clusters (k)")
axes[1].set_ylabel("Silhouette Score")
plt.show()
```

Ilustración 24 Visualización utilizando seaborn y matplotlib

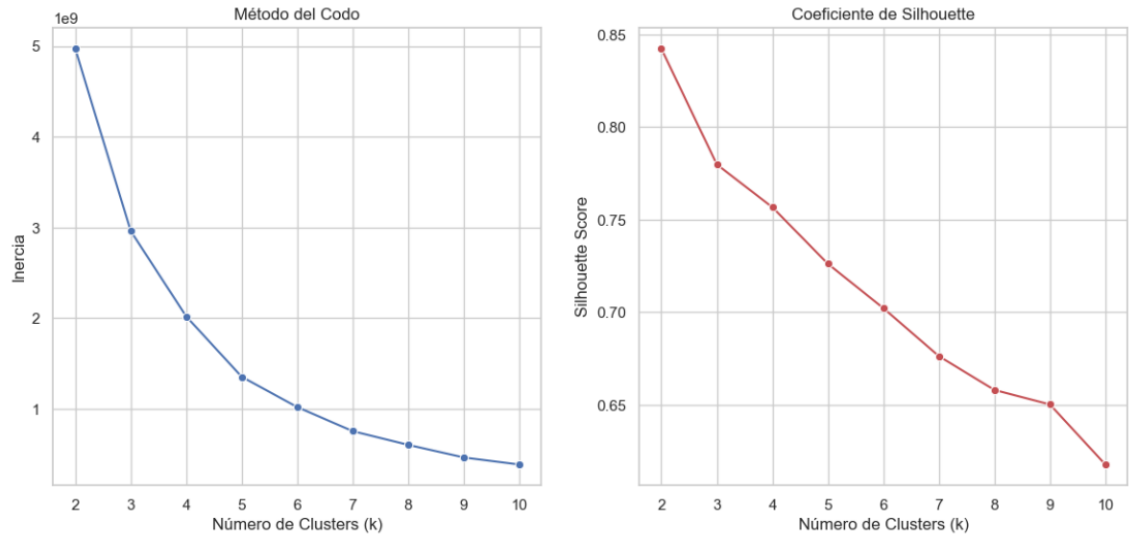


Ilustración 25 Resultado

Basándonos en ambos métodos parece que 4 o 5 son las mejores opciones para separar los clusters, en mi caso voy a usar 4.

```
numClusters = 4
kMeansClusters = KMeans(n_clusters=numClusters, random_state=42, n_init=5).fit(XRegClusters.values)
kMeansClusters.labels_
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int32)
```

```
# Para poder interpretar los datos unimos los clusters con sus respectivos datos
DFClusters=pd.DataFrame(kMeansClusters.labels_,columns=['Cluster'])
DFClusters.index = XRegClusters.index
kMeansResult = XRegClusters.join(DFClusters)
kMeansResult
```

NS	SALES	QUANTITY	DISCOUNT	SHIPPING_COST	PREPARATION_TIME	MARKET	CATEGORY	YEAR_ORDER	MONTH_ORDER	YEAR_SHIP	MONTH_SHIP	VIP	Cluster
2	105.72	4.0	0.0	6.73	6.0	4	2	2013	4	2013	4	0	0
2	6.45	1.0	0.0	1.37	6.0	4	2	2013	4	2013	4	0	0
2	182.07	1.0	0.0	59.24	0.0	4	1	2014	4	2014	4	0	0
2	85.26	1.0	0.0	4.06	5.0	4	1	2014	12	2014	12	0	0
2	553.92	4.0	0.0	206.56	2.0	4	1	2014	8	2014	8	0	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2	332.28	2.0	0.0	60.06	5.0	2	0	2015	1	2015	1	0	0
2	362.40	1.0	0.0	42.22	5.0	2	0	2015	1	2015	1	0	0
2	132.48	1.0	0.0	18.63	5.0	2	0	2015	1	2015	1	0	0
2	103.56	2.0	0.0	8.12	5.0	2	2	2015	1	2015	1	0	0
2	19.29	1.0	0.0	3.95	5.0	2	2	2015	1	2015	1	0	0

Ilustración 26

Otengo los clusters y se insertan en un *dataframe*.

```
# Para comprobar la asignación de clusters de forma visual podemos utilizar las visualizaciones que vimos anteriormente
# y analizar como se correlacionan las distintas variables e instancias con los clusters


sns.set(rc={"figure.figsize":(16, 8)})
bubbleplot = sns.scatterplot(data=kMeansResult,x='SALES',y='QUANTITY', \
                             size='SHIPPING_COST',hue='Cluster', sizes=(100,800))
```

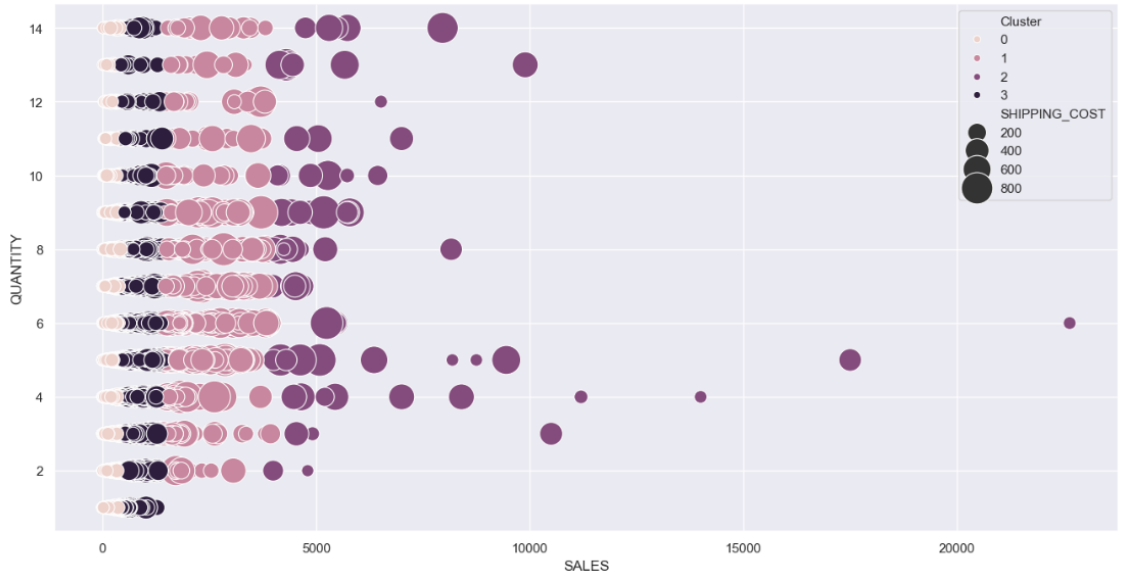


Ilustración 27

Lo visualizamos con un *bubble plot*. A continuación, lo comparamos con los clusters obtenidos tras realizar un **escalado**.

```
# Vamos a probar ahora escalando los datos de entrada

scaledKMeans = make_pipeline(StandardScaler(), KMeans(n_clusters=numClusters, random_state=42, n_init=5))
scaledKMeans.fit(XRegClusters.values)

DFScaledClusters=pd.DataFrame(scaledKMeans[1].labels_,columns=['Cluster'])
DFScaledClusters.index = XRegClusters.index
scaledKMeansResult = XRegClusters.join(DFScaledClusters)
scaledKMeansResult
```

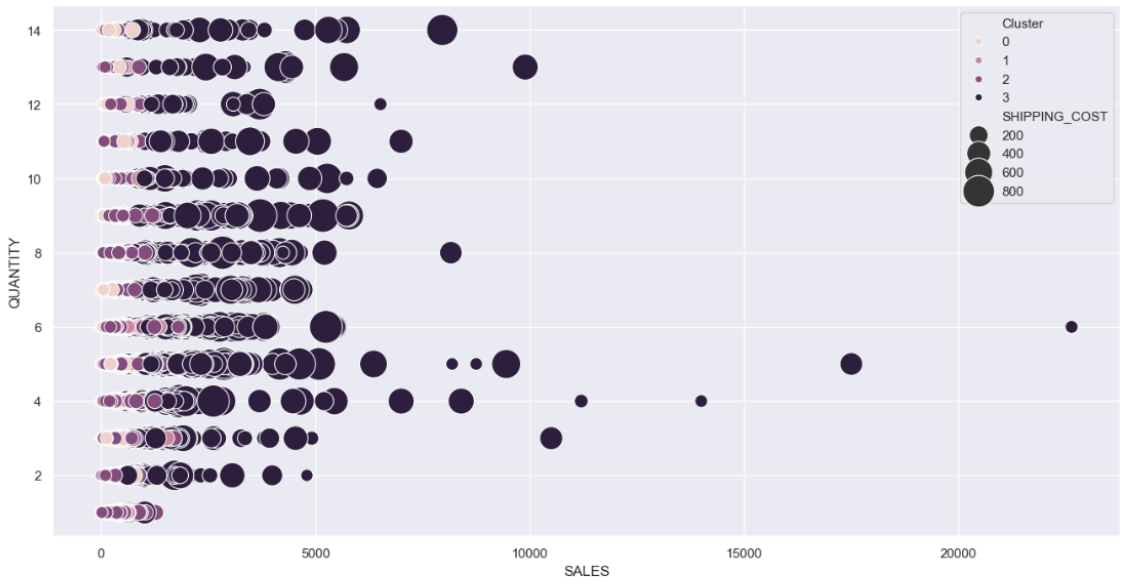
	SHIPMENT_IDSHIPMENT	PRIORITY_IDPRIORITY	RETURNS_IDRETURNS	SALES	QUANTITY	DISCOUNT	SHIPPING_COST	PREPARATION_TIME	MARKET	CATEGOI
0	4	4	2	105.72	4.0	0.0	6.73	6.0	4	
1	4	4	2	6.45	1.0	0.0	1.37	6.0	4	
2	2	1	2	182.07	1.0	0.0	59.24	0.0	4	
3	4	4	2	85.26	1.0	0.0	4.06	5.0	4	
4	1	1	2	553.92	4.0	0.0	206.56	2.0	4	
...	...	...	...	...	...	...	...	...	...	
45995	3	2	2	332.28	2.0	0.0	60.06	5.0	2	
45996	3	2	2	362.40	1.0	0.0	42.22	5.0	2	
45997	3	2	2	132.48	1.0	0.0	18.63	5.0	2	
45998	3	2	2	103.56	2.0	0.0	8.12	5.0	2	
45999	3	2	2	19.29	1.0	0.0	3.95	5.0	2	

46000 rows × 16 columns

Ilustración 28

Kmeans es sensible a la magnitud de los datos, por lo que se normalizan todas las variables para que tengan media 0 y desviación estándar 1. Seguidamente volvemos a calcular los clusters y los volvemos visualizar con un *bubble plot*.

```
sns.set(rc={"figure.figsize":(16, 8)})
bubbleplot = sns.scatterplot(data=scaledKMeansResult,x='SALES',y='QUANTITY', \
                             size='SHIPPING_COST',hue='Cluster', sizes=(100,800))
```




```
from sklearn.metrics import silhouette_samples, silhouette_score
silhouette_avg = silhouette_score(XReg, scaledKMeans[1].labels_)
print("For ",numClusters," clusters, the average Silhouette score is :",silhouette_avg)
```

```
For  4  clusters, the average Silhouette score is : -0.03694690596104023
```

Ilustración 29 Visualización final del Kmeans escalado