1. Exploramos los datos

Leemos el archivo excel

```
data = pd.read_excel("../Data/E-Commerce_train.xlsx")
data.head()
```

Python

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_importance	Gender	Discount_offered	Weight_in_gms	Reached.on.Time_Y.N
0	1	D	Flight	4	2	177	3	low	F	44	1233	1
1	2	F	Flight	4	5	216	2	low	M	59	3088	1
2	3	A	Flight	2	2	183	4	low	M	48	3374	1
3	4	B	Flight	3	3	176	4	medium	M	10	1177	1
4	5	C	Flight	2	2	184	3	medium	F	46	2484	1

Valores duplicados

No existen valores duplicados

```
data[data.duplicated()]
```

Python

ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_importance	Gender	Discount_offered	Weight_in_gms

Observamos la informacion

Veamos la cantidad de datos non-null y tambien los tiopos de datos

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8999 entries, 0 to 8998
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   ID                   8999 non-null   int64 
 1   Warehouse_block      8999 non-null   object
 2   Mode_of_Shipment     8999 non-null   object
 3   Customer_care_calls  8999 non-null   int64 
 4   Customer_rating      8999 non-null   int64 
 5   Cost_of_the_Product  8999 non-null   int64 
 6   Prior_purchases      8999 non-null   int64 
 7   Product_importance   8999 non-null   object
 8   Gender               8999 non-null   object
 9   Discount_offered     8999 non-null   int64 
 10  Weight_in_gms        8999 non-null   int64 
 11  Reached.on.Time_Y.N  8999 non-null   int64 
dtypes: int64(8), object(4)
memory usage: 843.8+ KB
```

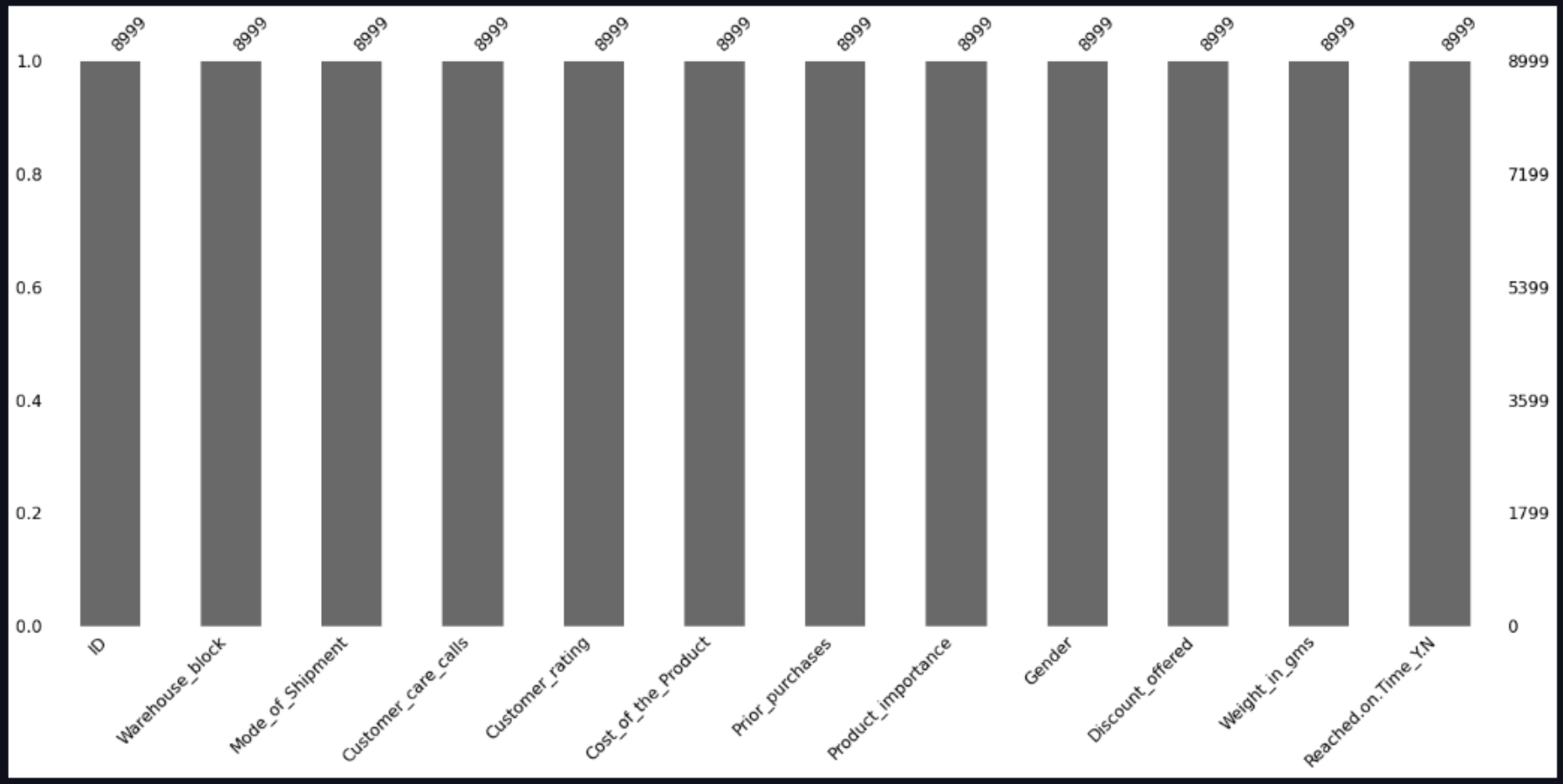
Al parecer todo esta completo y tambien vemos que hay 4 columnas que posiblemente sean categoricas y nos sirvan para nuestro modelo

Observamos valores faltantes

Comprobamos valores faltantes

```
[25]
msn.bar(data)
```

```
<AxesSubplot:>
```



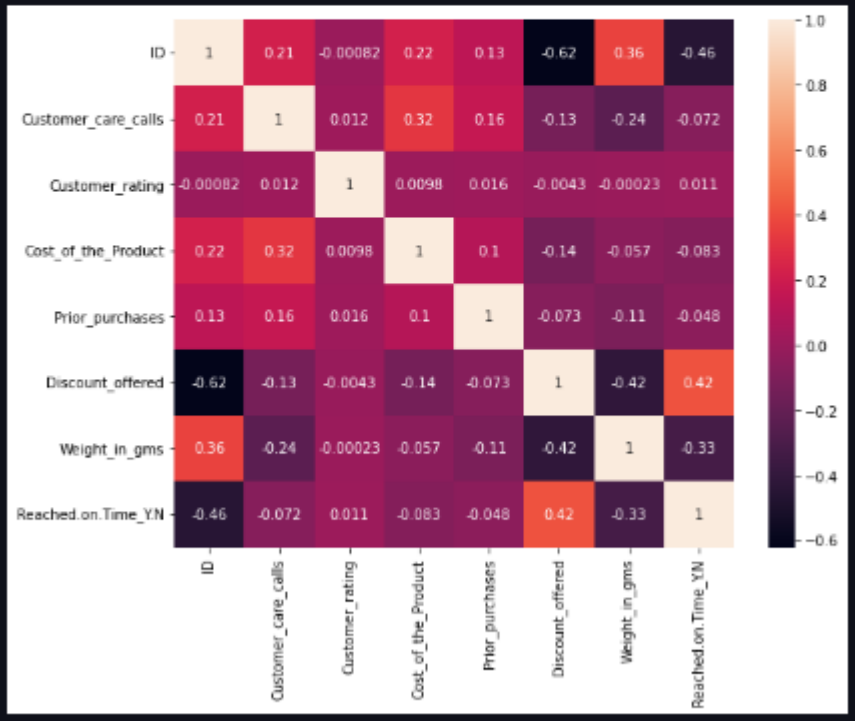
El 100% de los valores de las columnas esta completas

Vemos la correlacion de la data

```
corr = data.corr()
```

```
plt.figure(figsize=(9,7))
sns.heatmap(corr,annot=True)
```

```
<AxesSubplot:>
```

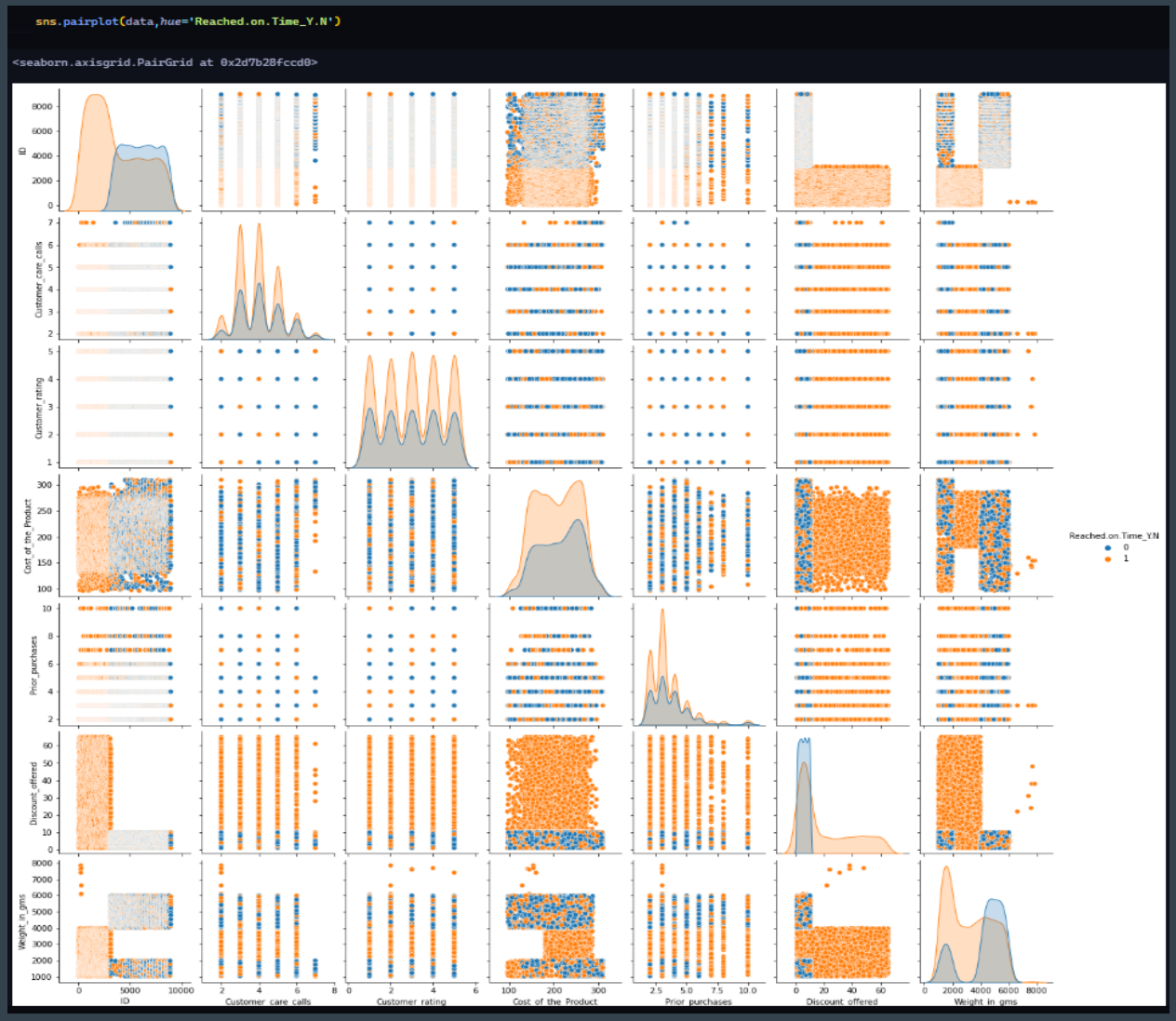


No existe una buena correlacion entre las caracteristicas y la variable objetivo

Vemos la dispersión de los datos

```
sns.pairplot(data,hue='Reached.on.Time_Y.N')
```

```
<seaborn.axisgrid.PairGrid at 0x2d7b28fccd0>
```

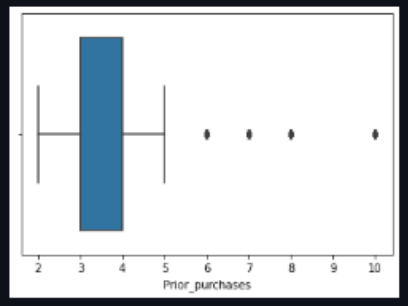


Identificacion de Outlier

Los outlier identificados en los boxplot no son outlier como tal, mas bien son datos que estan dentro de lo permitido y que sucede muy pocas veces

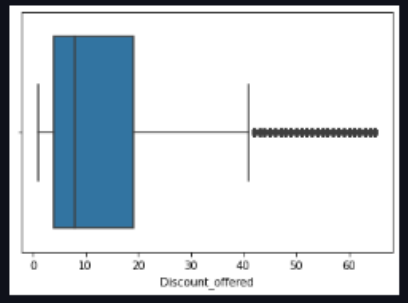
```
#sns.boxplot(x="Customer_care_calls",data= data)
#sns.boxplot(x="Customer_rating",data= data)
#sns.boxplot(x="Cost_of_the_Product",data= data)
sns.boxplot(x="Prior_purchases",data= data)
```

```
<AxesSubplot:xlabel='Prior_purchases'>
```



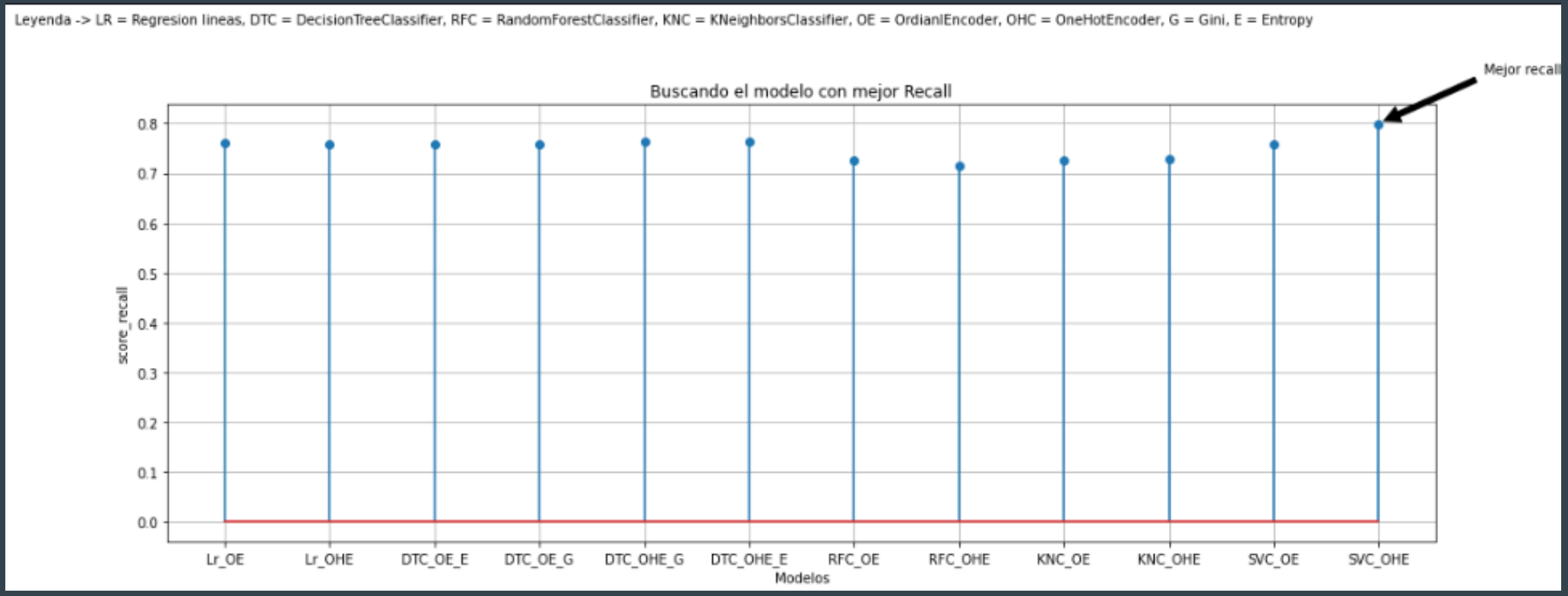

```
sns.boxplot(x="Discount_offered",data= data)
```

```
<AxesSubplot:xlabel='Discount_offered'>
```



Hasta el momento no encontramos datos duplicados ni faltantes, lo que sí deberíamos de manejar son los datos categóricos y la estandarización.

De acuerdo a la siguiente gráfica podemos ver cual es el mejor modelo y a partir de eso empezar a completar el informe.



[El mejor modelo es Support Vector Classifier]

2. Modelamiento

Separando datos categoricos y numericos en listas

[7]
```
categoricos = [cat for cat in data.columns if data[cat].dtype == "object" and data[cat].nunique() ≤ 10]
```

[8]
```
categoricos
```

```
['Warehouse_block', 'Mode_of_Shipment', 'Product_importance', 'Gender']
```

[9]
```
numericos = [num for num in data.columns if data[num].dtype ≠ "object"]
```

El ID no ayudara en el modelo

[10]
```
numericos.remove('ID')
```

[11]
```
numericos
```

```
['Customer_care_calls',
 'Customer_rating',
 'Cost_of_the_Product',
 'Prior_purchases',
 'Discount_offered',
 'Weight_in_gms',
 'Reached.on.Time_Y.N']
```

Ordenamos las columnas

[12]
```
data = data[numericos + categoricos]
```

Separando en caracteristicas y variable objetivos

Como valores de X no consideramos **Reached.on.Time_Y.N** ya que es la variable objetivo y tambien descartamos **Gender** ya que no nos dara mejora en el modelo

```
X = data.loc[:,[dx for dx in data.columns if dx not in ['Reached.on.Time_Y.N','Gender']]]
y = data['Reached.on.Time_Y.N']
```

```
X
```

	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Discount_offered	Weight_in_gms	Warehouse_block	Mode_of_Shipment	Product_importance
0	4	2	177	3	44	1233	D	Flight	low
1	4	5	216	2	59	3088	F	Flight	low
2	2	2	183	4	48	3374	A	Flight	low
3	3	3	176	4	10	1177	B	Flight	medium
4	2	2	184	3	46	2484	C	Flight	medium
...	...	...	...	...	...	...	...	...	...
8994	3	1	217	3	1	4177	D	Ship	low
8995	5	3	232	3	3	4526	F	Ship	medium
8996	4	5	260	3	6	4221	A	Ship	medium
8997	4	2	184	3	5	5931	B	Ship	medium

Separando valores en train test

```
from sklearn.model_selection import train_test_split
```

```
train_x, test_x, train_y, test_y = train_test_split(X,y,test_size=0.3,random_state=1)
```

Pipelin y ColumnsTransform

Usamos **tuberias** para que se nos haga mas facil llegar a los resultados

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
```

Existen diversas formas de manejar los datos categorios, en mi caso usare **OrdinalEncoder** y **OneHotEncoder** y asi evaluar los resultados

OrdinalEncoder

Primero transformo las columnas

```
encoder = OrdinalEncoder()
standard = StandardScaler()
```

```
encoder_standard = ColumnTransformer(transformers=[
    ("encoder",encoder,categoricos[:3]),
    ("standar",standard,[num for num in numericos if num not in ['Reached.on.Time_Y.N']])
])
```

OneHotEncoder

Primero transformo las columnas

```
encoder2 = OneHotEncoder()
standard2 = StandardScaler()
```

```
encoder_standard2 = ColumnTransformer(transformers=[
    ("encoder",encoder2,categoricos[:3]),
    ("standar",standard2,[num for num in numericos if num not in ['Reached.on.Time_Y.N']])
])
```

Usando SVC y OneHotEncoder

OneHot

Agregamos a la tuberia los pasos

```
pipeline_svc2 = Pipeline(steps=[
        ("previo",encoder_standard2),
        ("gridknn",gridSVC)
    ])
```

Ajustamos

```
pipeline_svc2.fit(train_x,train_y)
```

```
Pipeline(steps=[('previo',
                 ColumnTransformer(transformers=[('encoder', OneHotEncoder(),
                                                  ['Warehouse_block',
                                                   'Mode_of_Shipment',
                                                   'Product_importance']),
                                                 ('standar', StandardScaler(),
                                                  ['Customer_care_calls',
                                                   'Customer_rating',
                                                   'Cost_of_the_Product',
                                                   'Prior_purchases',
                                                   'Discount_offered',
                                                   'Weight_in_gms'])])),
                ('gridknn',
                 GridSearchCV(estimator=SVC(random_state=1),
                              param_grid={'gamma': ('scale', 'auto'),
                                          'kernel': ('linear', 'poly', 'rbf',
                                                     'sigmoid')},
                              scoring='recall'))])
```

Predecimos y evaluamos

```
pred_svc2= pipeline_svc2.predict(test_x)
recall_score(test_y,pred_svc2)
```

```
0.7996443390634262
```

Aplicación a E-commerce-test

Como ya tenemos las tuberias listas solo es necesario colocar la data. Las tuberias automaticamente manejaran los datos categoricos y estandarizar los datos

Recordemos que el mejor modelo fue SVC-OneHotEnoder

```
pred_final= pipeline_svc2.predict(ecommerce_test)
```

Creamos el DataFrame

```
df = pd.DataFrame({'pred':pred_final})
```

Creamos el Archivo CSV

```
df.to_csv("../Edward-Htc.csv",index=False)
```

pred
0
1
0
0
0
1
0
1
0
0
0
0
1
0
0
0
0
0
0
1
0
0
1
1
1
1
1
1
0
1
1
1
1
1
1
1
0
1
1
1
1
1
0
0
0
1
1
1
1