

Preparación del dataframe

```
In [ ]: # Librerías
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

import sys
import sklearn.neighbors._base
sys.modules['sklearn.neighbors.base'] = sklearn.neighbors._base
from missingpy import MissForest
```

```
In [ ]: df_ventas = pd.read_csv("df_ventas_3.csv", sep=",")

# Paso previo para crear var_objetivo
df_ventas["num_compras"] = 1
```

```
In [ ]: df_ventas.head()
```

```
Out[ ]:
```

	Unnamed: 0		item_id		num_order	created_at
0	0	000010d95384a6ba3d57dd870e7b337c	65717498f0771a49497d80f11160093c		2017-09-2	15:46:3
1	1	00001a8fb0bd42b1e16ba731e30cc490	09b538e85ce396ecbb70695f91007830		2018-09-1	21:27:0
2	2	0000302bc9b9a670dfcb14381555ff45	bc150db52b5a565d31b1c70969638ca9		2018-11-1	16:36:1
3	3	000039147df4aacf0aa8b3a552e8ecdb	434cf1eaf255b367ce2d3343bb96b1fe		2017-09-0	12:08:4
4	4	000091029a220c2fdf12700f07f70b1d	f268c24275ad1d887925fca2909e2c2d		2018-09-2	09:45:1

5 rows × 27 columns

```
In [ ]: # Se elimina la columna derivada del índice
df_ventas.drop('Unnamed: 0', inplace=True, axis=1)
```

```
In [ ]: #Añadir columnas a la lista que se quiera hacer dummies
```

```
columns_to_encoding = ['day', 'analytic_category']

for column in columns_to_encoding:
    dummies = pd.get_dummies(df_ventas[column])
    for dummy in dummies.columns:
        df_ventas[dummy] = dummies[dummy]
    df_ventas.drop(column, axis = 1, inplace=True)
df_ventas.columns
```

```
Out[ ]: Index(['item_id', 'num_order', 'created_at', 'product_id', 'qty_ordered',
              'base_cost', 'price', 'discount_percent', 'customer_id', 'zipcode',
              'longitud_zip', 'country', 'region', 'city', 'date', 'year', 'hour',
              'week', 'margin_total', 'price_total', 'name', 'marca_value',
              'nombre_corto', 'num_compras', 'Friday', 'Monday', 'Saturday', 'Sunday',
              'Thursday', 'Tuesday', 'Wednesday', 'cosmética y belleza', 'herbolario',
              'higiene', 'infantil', 'nutrición', 'ortopedia', 'perfumeria',
              'veterinaria', 'vida íntima', 'óptica'],
              dtype='object')
```

```
In [ ]: # Diccionario con las funciones de agregación
```

```
dic_agg = {
    "price_total" : "sum",
    "qty_ordered": "sum",
    "discount_percent": "mean",
    "customer_id": "first",
    "city": "first",
    "num_compras": "first",
    "country": "first",
    "region": "first",
    "zipcode": "first",
    "hour": "first",
}
```

```
dic_dummies = dict(zip(df_ventas.columns[list(range(24, len(df_ventas.columns)))],
```

```
dic_agg_2 = {
    "price_total" : "mean",
    "qty_ordered": "mean",
    "discount_percent": "mean",
    #"num_order": "size",
    "city": pd.Series.mode,
    "country": pd.Series.mode,
    "region": pd.Series.mode,
    "zipcode": pd.Series.mode,
    "num_compras": "sum",
    "hour": "mean"
}
```

```
In [ ]: dic_agg.update(dic_dummies)
        dic_agg_2.update(dic_dummies)
```

GroupBy por pedidos

Paso previo requerido para poder hacer una correcta agrupación por cliente

```
In [ ]: # Groupby por pedidos

df_pedidos = df_ventas.groupby("num_order", as_index = False).agg(dic_agg)
```

Agrupación por clientes

Obteniendo el dataframe final deseado

```
In [ ]: # Agrupación clientes
df_clientes = df_pedidos.groupby("customer_id").agg(dic_agg_2)
```

```
In [ ]: df_clientes.head()
```

```
Out [ ]:
```

	price_total	qty_ordered	discount_percent	city	cour
customer_id					
000053b1e684c9e7ea73727b2238ce18	17.770	1.0	25.0	Torrelavega	Sp
0001c82eb924a3dca30593bf7d8f2227	33.570	1.0	5.0	Leres De Jaca	Sp
0003883910709aa39bf38b05c51c03a3	82.560	6.0	7.0	Madrid	Sp
0003a36a46798bafcc69637f52f75e95	79.550	3.0	5.0	Barcelona	Sp
0004a12374b272a1c591fd5122cde6a1	33.318	3.8	7.0	Aviles	Sp

5 rows × 6 columns

Obtención de variable objetivo y modelo a predecir

```
In [ ]: # Var_obj

df_clientes["num_compras"].replace(1, 0, inplace=True)
mascara_var_obj = df_clientes["num_compras"] > 1
df_clientes["num_compras"][mascara_var_obj] = 1
```

/tmp/ipykernel_288071/4007882791.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df_clientes["num_compras"][mascara_var_obj] = 1

```
In [ ]: #Convertir categóricas restantes

list_col_cat = ["city", "country", "region", "zipcode"]

numeric_model = LabelEncoder()
for column in list_col_cat:
    list_values = list(df_clientes[column].unique())
    numeric_model.fit_transform(list_values)
    df_clientes[column] = numeric_model.transform(df_clientes[column])
```

Como uno de los objetivos es que porcentaje de probabilidad de repetir tienen los clientes, se utilizará el propio dataframe como si fuese un dataframe a predecir.

```
In [ ]: #df de predicción
df_clientes_no_target = df_clientes.drop("num_compras", axis = 1)

target = df_clientes["num_compras"]
```

```
In [ ]: # Imputación de valores perdidos
```

```
In [ ]: df_imputed = df_clientes_no_target
imputer = MissForest(criterion=('squared_error', 'gini'))

#Perform the imputation
df_imputed = imputer.fit_transform(df_imputed)
```

```
/workspaces/DatathonProject/venv/lib/python3.10/site-packages/missingpy/mis
sforest.py:528: UserWarning: No missing value located; returning original d
ataset.
  warnings.warn("No missing value located; returning original ")
```

```
In [ ]: #List of columns with still NaN
columns_na = df_clientes_no_target.columns[df_clientes_no_target.isna().any(
#Loop for replacing values of columns with NaN from the df imputed
for column in columns_na:
    index = df_clientes_no_target.columns.tolist().index(column)
    df_clientes_no_target[column] = df_imputed[:,index]
#Check there's no columns with NaN
df_clientes_no_target.isnull().sum()
```

```
Out[ ]: price_total      0
        qty_ordered     0
        discount_percent 0
        city             0
        country          0
        region           0
        zipcode          0
        hour             0
        Friday           0
        Monday           0
        Saturday         0
        Sunday           0
        Thursday         0
        Tuesday          0
        Wednesday        0
        cosmética y belleza 0
        herbolario       0
        higiene          0
        infantil         0
        nutrición        0
        ortopedia        0
        perfumeria       0
        veterinaria      0
        vida íntima      0
        óptica           0
        dtype: int64
```

Modelo

```
In [ ]: # Librerías para el modelo

        from xgboost import XGBClassifier
        from sklearn.metrics import accuracy_score, auc, confusion_matrix, f1_score,
        from imblearn.over_sampling import SMOTE
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import classification_report
```

Training-Test

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(df_clientes_no_target, t
```

XGBoost

```
In [ ]: ## Función para métricas

def saca_metricas(y1, y2):
    print('matriz de confusión')
    # print(confusion_matrix(y1, y2))
    print('accuracy')
    print(accuracy_score(y1, y2))
    print('precision')
    print(precision_score(y1, y2))
    print('recall')
    print(recall_score(y1, y2))
    print('f1')
    print(f1_score(y1, y2))
    false_positive_rate, recall, thresholds = roc_curve(y1, y2)
    roc_auc = auc(false_positive_rate, recall)
    print('AUC')
    print(roc_auc)
    plt.plot(false_positive_rate, recall, 'b')
    plt.plot([0, 1], [0, 1], 'r--')
    plt.title('AUC = %0.2f' % roc_auc)
```

```
In [ ]: xgboost = XGBClassifier()

modelXGB = xgboost.fit(X_train, y_train, eval_metric='rmse')

y_pred_XGB = modelXGB.predict(X_test)

print(classification_report(y_test, y_pred_XGB))
```

/workspaces/DatathonProject/venv/lib/python3.10/site-packages/xgboost/sklearn.py:793: UserWarning: `eval_metric` in `fit` method is deprecated for better compatibility with scikit-learn, use `eval_metric` in constructor or `set_params` instead.

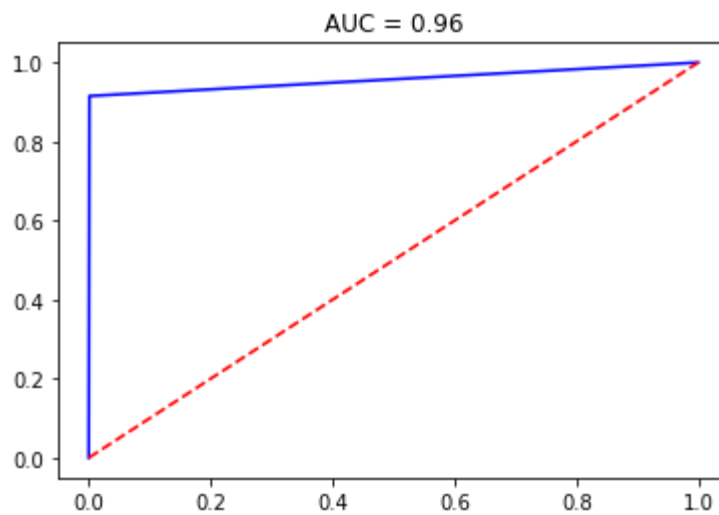
```
warnings.warn(
              precision    recall  f1-score   support

         0       0.93      1.00      0.96      12700
         1       1.00      0.92      0.96      10860

 accuracy          0.96      0.96      0.96      23560
 macro avg          0.97      0.96      0.96      23560
weighted avg          0.96      0.96      0.96      23560
```

```
In [ ]: saca_metricas(y_test, y_pred_XGB)
```

```
matriz de confusión
accuracy
0.9602716468590832
precision
0.9984930681133213
recall
0.9151933701657459
f1
0.9550302680887864
AUC
0.9570061339017707
```



Variables con más influencia

```
In [ ]: # Variables

best_xgb_features = modelXGB.feature_importances_

best_xgb_features = pd.DataFrame(best_xgb_features,
                                index = X_train.columns,
                                columns=['importance']).sort_values('importance',
```

```
In [ ]: best_xgb_features
```

Out[]:

	importance
Sunday	0.123242
Monday	0.109119
Thursday	0.104810
Tuesday	0.104153
higiene	0.082865
Wednesday	0.082481
Friday	0.081803
Saturday	0.075944
cosmética y belleza	0.056750
qty_ordered	0.042281
discount_percent	0.035694
hour	0.028115
herbolario	0.020871
infantil	0.016945
nutrición	0.012191
vida íntima	0.004873
ortopedia	0.002868
óptica	0.002585
perfumeria	0.002380
price_total	0.002255
city	0.001948
zipcode	0.001909
region	0.001848
country	0.001556
veterinaria	0.000512

```
In [ ]: mascara = best_xgb_features["importance"] > 0.01
best_features = best_xgb_features[mascara]
num_otras = 1 - best_features["importance"].sum()
nueva_fila = {"importance": num_otras}
best_features_otras = best_features
best_features_otras = best_features_otras.sort_values('importance', ascending=True)
best_features_otras = best_features_otras.append(nueva_fila, ignore_index=True)
best_index = best_features.index.tolist()
best_index.append("otras")
best_features_otras.index = best_index
```



```
/tmp/ipykernel_288071/2109732315.py:7: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
```

```
best_features_otras = best_features_otras.append(nueva_fila, ignore_index=True)
```

```
In [ ]: best_features_otras * 100
best_features * 100
```

```
Out[ ]:
```

	importance
Sunday	12.324236
Monday	10.911889
Thursday	10.481029
Tuesday	10.415273
higiene	8.286453
Wednesday	8.248102
Friday	8.180315
Saturday	7.594436
cosmética y belleza	5.675022
qty_ordered	4.228088
discount_percent	3.569385
hour	2.811548
herbolario	2.087096
infantil	1.694452
nutrición	1.219057

Probabilidad de volver a comprar

```
In [ ]: clients_predict = modelXGB.predict_proba(df_clientes_no_target)
list_predict = []
for value in clients_predict:
    list_predict.append(value[1])
```

```
In [ ]: df_clientes_no_target["predict"] = list_predict
df_clientes_no_target["predict"] = df_clientes_no_target["predict"] * 100

df_pctge = pd.DataFrame(df_clientes_no_target["predict"])

df_pctge = df_pctge.sort_values('predict', ascending=False)

df_pctge.head()
```

Out[]:

	predict
customer_id	
e7a7135b3f6fd679d7b0f717f442e478	100.0
70c2f118940eb9d99c3bcc15f3a9f01c	100.0
94c32f9125a78db76c6002fb47973a70	100.0
ed8709d793531889b3912326d311d70e	100.0
94c71292ad0a8f8b2a18128cacda6650	100.0

```
In [ ]: # Intervalos

list_var = ["0-20", "20-40", "40-60", "60-80", "80-100"]

df_pctge["predict_interval"] = df_pctge["predict"]

df_pctge["predict_interval"][df_pctge["predict"] <= 100] = "80-100"
df_pctge["predict_interval"][df_pctge["predict"] < 80] = "60-80"
df_pctge["predict_interval"][df_pctge["predict"] < 60] = "40-60"
df_pctge["predict_interval"][df_pctge["predict"] < 40] = "20-40"
df_pctge["predict_interval"][df_pctge["predict"] < 20] = "0-20"
```

Exportación de resultados final

```
In [ ]: # Porcentaje de clientes
df_pctge.to_excel("pctge_repetir_v2.xlsx")

# Porcentaje de importancia variables
best_features.to_excel("beast_features_v2.xlsx")

# Porcentaje de importancia variables agrupadas en otras las que menos impor
best_features_otras.to_excel("beast_features_otras_v2.xlsx")
```