Predicción de pérdida de clientes en Empresas de Servicios de Telecomunicaciones (Churn)

Reducir las salidas y deserciones de clientes se ha convertido en una alta prioridad para la mayoría de los proveedores de servicios de comunicaciones a medida que los mercados maduran y la competencia se intensifica.

En este documento usaremos una base de datos de una empresa de telecomunicaciones anónima disponibilizada por IBM.

El principal objetivo es crear un model de aprendizaje automático basado en SVM y Regresión Logística (Logistic Regression, LR) para predecir la pérdida o salida de clientes en una empresa de telecomunicaciones.

Librerías

In [1]:
```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
```

Base de datos

Este conjunto de datos contiene un total de 7043 clientes y 21 características de los mismos. De las entradas, 5174 son clientes activos y 1869 son clientes que la empresa ha perdido. Observe que el conjunto de datos está desbalanceado pues por cada cliente perdido existe casi 3 clientes activos. La variable de salida para nuestro modelo de machine learning será `Churn` .

In [2]:
```
# importamos dataset
DATA_PATH = "https://raw.githubusercontent.com/carlosfab/dsnp2/master/datasets/WA_F
df = pd.read_csv(DATA_PATH)

# vemos las primeras 5 filas
df.head()
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service
1	5575-GNVDE	Male	0	No	No	34	Yes	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service
4	9237-HQITU	Female	0	No	No	2	Yes	No

5 rows × 21 columns

Detalles de la base de datos

- `customerID` - Custumer unique identifier
- `gender` - Customer gender - ['Female' 'Male']
- `SeniorCitizen` - Elderly or retired person, a senior citizen is someone who has at least attained the age of 60 of 65 years
- `Partner` - - ['No' 'Yes']
- `Dependents` - If customer has dependents - ['No' 'Yes']
- `Tenure` - Customer lifespan (in months)
- `PhoneService` - - ['No' 'Yes']
- `MultipleLines` - - ['No' 'No phone service' 'Yes']
- `InternetService` - - ['No' 'No internet service' 'Yes']
- `OnlineSecurity` - - ['No' 'No internet service' 'Yes']
- `OnlineBackup` - - ['No' 'No internet service' 'Yes']
- `DeviceProtection` - - ['No' 'No internet service' 'Yes']
- `TechSupport` - - ['No' 'No internet service' 'Yes']
- `StreamingTV` - - ['No' 'No internet service' 'Yes']
- `StreamingMovies` - - ['No' 'No internet service' 'Yes']
- `Contract` - Type of contract - ['Month-to-month' 'One year' 'Two year']
- `PaperlessBilling` - - ['No' 'Yes']
- `PaymentMethod` - payment method - ['Bank transfer (automatic)', 'Credit card (automatic)', 'Electronic check', 'Mailed check']
- `MonthlyCharges` - Monthly Recurring Charges
- `TotalCharges` - Life time value
- `Churn` - Churn value, the targer vector - ['No' 'Yes']

Limpieza del Dataset

La función get_df_size se utiliza para obtener información básica sobre el tamaño del conjunto de datos, en términos de su número de atributos y entradas.

Luego, se reemplazan los valores en blanco por NaN (valores faltantes) en el dataframe utilizando la función replace de pandas. Se reemplazan los valores faltantes en la columna TotalCharges por la mediana de los valores no faltantes de la misma columna, utilizando la función fillna de pandas. Además, se convierte esta columna de tipo objeto a tipo numérico utilizando la función apply de pandas y el método pd.to_numeric.

A continuación, se elimina la columna customerID, ya que no es una característica relevante para el modelo. Se utiliza el método drop de pandas para eliminar la columna, especificando el argumento axis=1 para indicar que se está eliminando una columna y no una fila.

Por último, se utiliza la función describe de pandas para obtener estadísticas descriptivas del conjunto de datos preprocesado y se imprimen en pantalla la cantidad de instancias con valor "Yes" y "No" en la variable objetivo Churn.

In [3]:
```
def get_df_size(df, header='Dataset dimensions'):
  print(header,
        '\n# Attributes: ', df.shape[1],
        '\n# Entries: ', df.shape[0],'\n')

get_df_size(df)

#df.info()

# reemplaza valores en blanco por NaN
df_clean = df.replace(r'^\s*$', np.nan, regex=True)

# reemplaza valores faltantes en TotalCharges por la mediana de TotalCharges.
total_charges_median = df_clean.TotalCharges.median()
df_clean['TotalCharges'].fillna(total_charges_median, inplace=True)
df_clean['TotalCharges'] = df_clean['TotalCharges'].apply(pd.to_numeric)

#CustomerID lo retiramos porque no es una característica
df_clean = df_clean.drop('customerID', axis=1)
df_clean.describe()

print("Churn No Instances: ", df_clean[df_clean['Churn'] == 'No'].shape[0])
print("Churn Yes Instances: ", df_clean[df_clean['Churn'] == 'Yes'].shape[0])
```

```
Dataset dimensions
# Attributes:  21
# Entries:  7043

Churn No Instances:  5174
Churn Yes Instances:  1869
```

Preparación de la Base de Datos

Este código realiza la preprocesamiento de los datos en el dataframe df_clean para poder usarlo en un modelo de machine learning. En primer lugar, identifica las características binarias, numéricas y categóricas en el conjunto de datos. Luego, crea una copia del dataframe original y realiza las siguientes operaciones sobre él:

Convierte las características binarias en etiquetas numéricas mediante el uso de LabelEncoder(). Crea variables dummy para las características categóricas usando pd.get_dummies(). Imprime la información del tamaño del conjunto de datos original y procesado utilizando la función get_df_size(). Retorna un nuevo dataframe df_proc que contiene todas las características preprocesadas y que puede ser utilizado en un modelo de machine learning. En resumen, el código realiza la limpieza, transformación y preparación de los datos para su posterior uso en un modelo de machine learning.

In [4]:
```
binary_feat = df_clean.nunique()[df_clean.nunique() == 2].keys().tolist()
numeric_feat = [col for col in df_clean.select_dtypes(['float','int']).columns.toli
categorical_feat = [ col for col in df_clean.select_dtypes('object').columns.to_lis
df_proc = df_clean.copy()
#Etiquetas para características binarias
le = LabelEncoder()
for i in binary_feat:
  df_proc[i] = le.fit_transform(df_proc[i])
  print(i, '\n', np.unique(df_proc[i].values))
#Dummy variables
df_proc = pd.get_dummies(df_proc, columns=categorical_feat)
get_df_size(df, header='Original dataset:')
get_df_size(df_proc, header='Processed dataset:')
df_proc.head()
```

```
gender
 [0 1]
SeniorCitizen
 [0 1]
Partner
 [0 1]
Dependents
 [0 1]
PhoneService
 [0 1]
PaperlessBilling
 [0 1]
Churn
 [0 1]
Original dataset:
# Attributes:  21
# Entries:  7043

Processed dataset:
# Attributes:  41
# Entries:  7043
```

Out[4]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	MonthlyCh
0	0	0	1	0	1	0	1	
1	1	0	0	0	34	1	0	
2	1	0	0	0	2	1	1	
3	1	0	0	0	45	0	0	
4	0	0	0	0	2	1	1	

5 rows × 41 columns

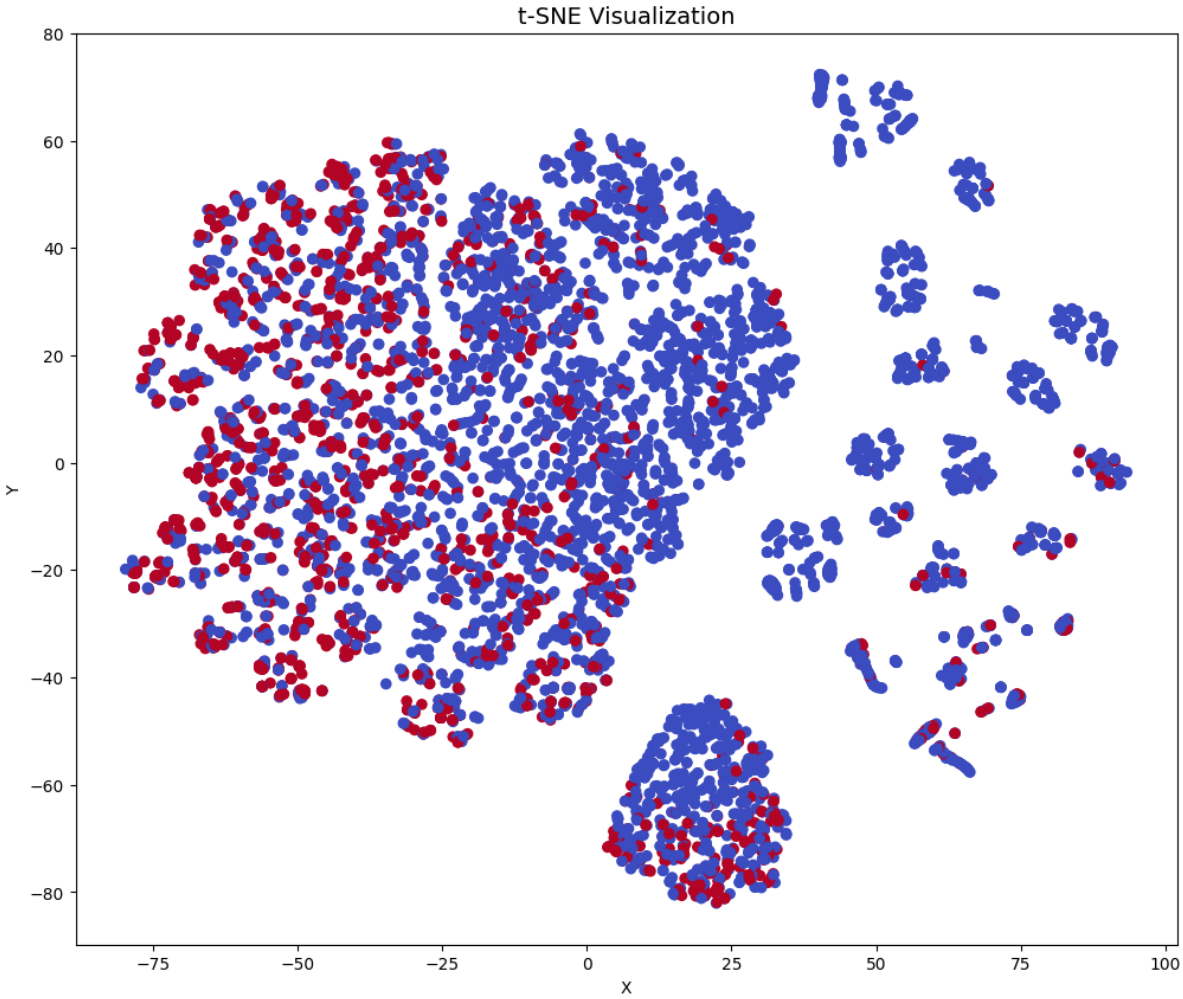
División en conjunto de entrenamiento y test

In [5]:
```
# dividimos df_proc en características y salida
X=df_proc.drop('Churn', axis=1)
y=df_proc['Churn']

# Dividimos el conjunto de entrenamiento y test
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2
```

Actividad 1: Visualización con TSNE

Utilice TSNE para visualizar su dataset de entrenamiento. Recuerde normalizar su dasaset (zscore) antes de esta y las siguientes actividades.

In [6]:
```
from scipy.stats import zscore

X_normalized = X.apply(zscore)

#Una vez normalizado el conjunto de datos, podemos visualizarlo con t-SNE

from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=26)

X_tsne = tsne.fit_transform(X_normalized)

plt.figure(figsize=(12,10))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='coolwarm', s=35)
plt.title('t-SNE Visualization', fontsize=14)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



Actividad 2: Clasificador y Optimización de Hiperparámetros SVM

Cree un clasificador con SVM (ver `SVC`) con kernel RBF. Use 5-fold cross-validation (ver `GridSearchCV` y `StratifiedKFold`) para optimizar los siguientes hiperparámetros en el conjunto de entrenamiento (`X_train`, `y_train`):

1. Gamma
2. C

Para la optimización de hiperparámetros utilice como métrica al recall.

Una vez optimizados los hiperparámetros, reentrene su clasificador con los mejores hiperparámetros encontrados. Finalmente, evalúe este último clasificador en el conjunto de test ('X_test', 'y_test'.)

In [7]:
```
# standardizing X_train and X_test
scaler = StandardScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)
```

In [8]:
```
#se importa la libreria para el clasificador SVM
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

#se crea el clasificador SVM
svm = SVC()

#importamos las librerias para realizar la validacion cruzada
from sklearn.model_selection import StratifiedKFold

#se crea el objeto para realizar la validacion cruzada
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=26)

#se realiza la busqueda de los mejores hiperparametros y usamos el kernel rbf
from sklearn.model_selection import GridSearchCV

#se crea el diccionario con los valores que se quieren probar
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel
#se crea el objeto para realizar la busqueda de los mejores hiperparametros
gs = GridSearchCV(svm, param_grid, cv=cv, scoring='recall')

#se procede a realizar el grid search en el conjunto de entramiento
gs_done = gs.fit(X_train_normalized, y_train)

#se quiere encontrar los mejores hiperparametros
best_hp = gs_done.best_params_
print("Los mejores hiperparámetros son:", best_hp)
```

```
Los mejores hiperparámetros son: {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
```

In [9]:
```
#obtenidos los mejores hiperparametros, se procede a entrenar el modelo con estos
svm = SVC(C=best_hp['C'], gamma=best_hp['gamma'], kernel=best_hp['kernel'])

#se entrena el modelo
svm.fit(X_train_normalized, y_train)

#se realiza la prediccion en el conjunto de test
y_pred = svm.predict(X_test_normalized)

#se calcula la matriz de confusion
cm = confusion_matrix(y_test, y_pred)
print("Matriz de confusión:\n", cm)

#se calcula el accuracy
acc = accuracy_score(y_test, y_pred)
print("Accuracy: ", acc)

#se calcula el AUC
auc = roc_auc_score(y_test, y_pred)
print("AUC: ", auc)

#se calcula el recall score
import sklearn.metrics as metrics
recall = metrics.recall_score(y_test, y_pred)
print("Recall: ", recall)
```

```
Matriz de confusión:
 [[856 179]
 [176 198]]
Accuracy:  0.7480482611781405
AUC:  0.6782324524012503
Recall:  0.5294117647058824
```

Actividad 3: Clasificador y Optimización de Hiperparámetros Logistic Regression

Repita la actividad 2 pero con Logistic Regression. Bajo el mismo esquema de la actividad anterior, optimice el parámetro de regularización lambda. Utilice una transformación polinomial para garantizar fronteras de decisión no lineal. Compare sus resultados con los obtenidos de la actividad anterior

In [11]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# create the pipeline with polynomial features
poly_lr_pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('lr', LogisticRegression(max_iter=5000))
])

# create the parameter grid
param_grid_lr = {
    'lr__C': [0.1, 1, 10, 100]
}

# create the GridSearchCV object
gs_lr = GridSearchCV(poly_lr_pipeline, param_grid=param_grid_lr, cv=cv, scoring='re

# fit the pipeline to the training data
gs_lr_done = gs_lr.fit(X_train_normalized, y_train)

#se quiere encontrar los mejores hiperparametros
best_lr_hp = gs_lr_done.best_params_
print("Los mejores hiperparámetros son:", best_lr_hp)
```

```
Los mejores hiperparámetros son: {'lr__C': 1}
```

In [12]:
```
#obtenidos los mejores hiperparametros, se procede a entrenar el modelo con estos
lr = LogisticRegression(C=best_lr_hp['lr__C'], max_iter=5000)

#se entrena el modelo
lr.fit(X_train_normalized, y_train)

#se realiza la prediccion en el conjunto de test
y_pred = lr.predict(X_test_normalized)

#se calcula la matriz de confusion
cm = confusion_matrix(y_test, y_pred)
print("Matriz de confusión:\n", cm)

#se calcula el accuracy
acc = accuracy_score(y_test, y_pred)
print("Accuracy: ", acc)

#se calcula el AUC
auc = roc_auc_score(y_test, y_pred)
print("AUC: ", auc)

#se calcula el recall score
recall = metrics.recall_score(y_test, y_pred)
print("Recall: ", recall)
```

```
Matriz de confusión:
 [[925 110]
 [162 212]]
Accuracy:  0.8069552874378992
AUC:  0.7302823632746905
Recall:  0.5668449197860963
```

En Comparación con el clasificador utilizado en la Actividad 2, incluso con los mejores hiperparámetros, el accuracy y el recall score aumentaron para este caso: Logistic Regression has a better performance than SVM

In [ ]: