

ENTREGA #2
PROYECTO INTELIGENCIA ARTIFICIAL
DEFAULT PREDICTION PROJECT (AMERICAN EXPRESS)

SANTIAGO GARCIA CASTRILLON
JUAN DANIEL TABARES GOEZ

PROFESOR:
RAUL RAMOS POLLÁN



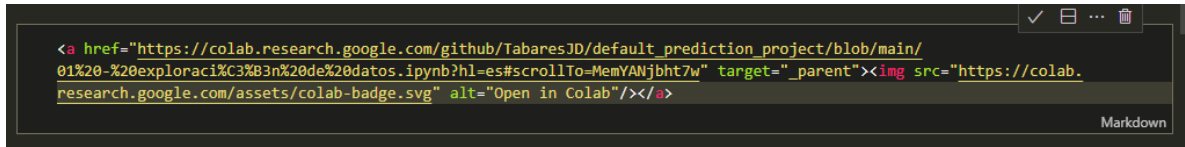
UNIVERSIDAD[®]
DE ANTIOQUIA

1 8 0 3

UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERIA
DEPARTAMENTO DE INGENIERIA MECANICA
MEDELLIN
2022

1. Avances.

Inicialmente se agregó el botón obligatorio de “open in colab”, el cual tiene que estar disponible en cada uno de los notebooks utilizados, que hasta este momento solo es uno.

A screenshot of a code cell in a Jupyter notebook. The cell contains a markdown link to a Colab notebook. The link text is "Open in Colab" and it is surrounded by a Colab logo. The code is:

```
<a href="https://colab.research.google.com/github/TabaresJD/default_prediction_project/blob/main/01%20-%20exploraci%C3%B3n%20de%20datos.ipynb?hl=es#scrollTo=MemYANjbht7w" target="_parent"></a>
```

 The cell is labeled "Markdown" in the bottom right corner.

Imagen 1. Botón open in colab.

Este es un código en markdown el cual al ejecutarlo muestra el botón que direcciona al link agregado.

En cuanto a la carga de los datos desde el dataset, se cargó una pequeña parte, debido a que eran muchos datos y los equipos con los que trabajamos no soportaban el procesamiento de tal cantidad de datos.

A screenshot of a Jupyter notebook titled "Carga de datos". The text below the title says "Se cargan 100,000 datos dados los recursos del sistema para procesar el dataset completo". There are two code cells. The first cell contains the code:

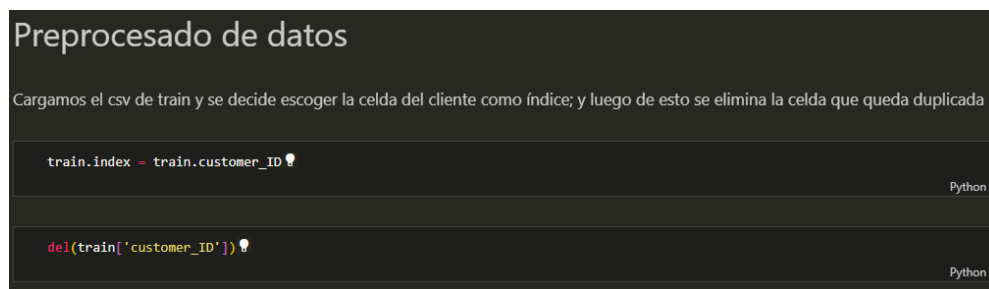
```
train = pd.read_csv('train_data.csv', nrows=100000)
```

 and is labeled "Python". The second cell contains the code:

```
labels = pd.read_csv('train_labels.csv')
```

 and is also labeled "Python".

Imagen 2. Carga de los datos .

A screenshot of a Jupyter notebook titled "Preprocesado de datos". The text below the title says "Cargamos el csv de train y se decide escoger la celda del cliente como índice; y luego de esto se elimina la celda que queda duplicada". There are two code cells. The first cell contains the code:

```
train.index = train.customer_ID
```

 and is labeled "Python". The second cell contains the code:

```
del(train['customer_ID'])
```

 and is also labeled "Python".

Imagen 3. Comienzo del procesamiento de los datos.

Para el procesamiento de datos lo primero que realizamos es el cambio del índice por el nombre de los clientes, que son códigos por cuestiones de privacidad. Obteniendo lo siguiente.

	S_2	P_2	D_39	B_1	B_2	R_1	S_3
customer_ID							
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	2017-03-09	0.938469	0.001733	0.008724	1.006838	0.009228	0.124035
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	2017-04-07	0.936665	0.005775	0.004923	1.000653	0.006151	0.126750
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	2017-05-28	0.954180	0.091505	0.021655	1.009672	0.006815	0.123977
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	2017-06-13	0.960384	0.002455	0.013683	1.002700	0.001373	0.117169
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	2017-07-16	0.947248	0.002483	0.015193	1.000727	0.007605	0.117325

Imagen 4. Datos con los clientes como índice.

Luego de analizar estos datos se puede ver que hay varios datos por cada usuario, por lo que a continuación se agruparon por la media y se eliminó la columna que almacenaba las fechas por lo que ya no era algo representativo.

Agrupamos los datos de train por cliente con su respectiva media para facilitar el manejo de los datos

Borramos los datos de las fechas para poder tratar los datos por cada cliente; además estas fechas no tienen un orden lógico, por lo tanto no se prestan para ser tratadas como series temporales de Pandas

```
del(train['S_2'])
```

```
data = train.groupby('customer_ID').mean()
```

Imagen 5. Eliminación de las fechas

Obteniendo un dataset de la siguiente configuración, donde se tiene una fila por cada cliente con la media de sus datos.

	P_2	D_39	B_1	B_2	R_1	S_3	D_41
customer_ID							
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	0.933824	0.010704	0.012007	1.005086	0.004509	0.113215	0.005021
00000fd6641609c6ece5454664794f0340ad84ddc9a267a310b5ae68e9d8e5	0.899820	0.215205	0.025654	0.991083	0.006246	0.120578	0.004993
00001b22f846c82c51f6e3958ccd81970162bae8b007e80662ef27519fcc18c1	0.878454	0.004181	0.004386	0.815677	0.006621	NaN	0.006842
000041bdba6ecadd89a52d11886e8eaac9325906c9723355abb5ca523658edc	0.598969	0.048862	0.059876	0.955264	0.005665	0.247750	0.005490
00007889e4fcd2614b6cbe7f8f3d2e5c728eca32d9eb8ad51ca8b8c4a24cefed	0.891679	0.004644	0.005941	0.814543	0.004180	0.173102	0.005352

Imagen 6. Dataframe con una fila por usuario.

Posterior a esto, vamos a simular unas columnas como categóricas, debido a que el dataset inicial no cumple con los criterios establecidos para la entrega. Para ello buscamos columnas con una gran cantidad de datos faltantes, debido a que modificar estas no cambiara la predicción.

Valores faltantes (NaN)

Se muestran las columnas con datos faltantes mayores a 5000 instancias

```
> train.isna().sum()
k[(k!=0)&(k>5000)]
```

Python

Imagen 7. Cantidad de datos faltantes por columna

```
columnas = ['D_88','D_87','B_29','R_9','D_73','D_49','D_66','D_76','D_42','D_82']
```

Función para el cambio de variables a categóricas

```
def categorical_simulation(df,col,n):
    #borramos la columna
    del(df[col])
    #Creamos una nueva con el mismo nombre y asignamos los valores de forma aleatoria
    valores = np.array([np.random.randint(1,n) for i in range(df.shape[0])])
    d = pd.DataFrame(valores, index = df.index, columns = [col])
    df = pd.concat([df,d], axis=1)
    return(df)
```

Python

Imagen 8. Función para cambiar variables “ruido” por categóricas.

Luego de seleccionar las variables con mayor número de datos faltantes, se creó una función para rellenar esta columna con datos categóricos. Esta función se aplica para las columnas seleccionadas.

Por otra parte, debido a que la variable objetivo se encontraba en un dataset separado, se incluyó como una columna para el dataframe previamente tratado hasta este punto.

Concatenamos los DataFrames porque la variable objetivo se encuentra en un dataset diferente al cual estamos trabajando

```
labels.index = labels.customer_ID
!i(labels['customer_ID'])
labels.head()
```

customer_ID	target
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	0
00000fd6641609c6ece5454664794f0340ad84dddc9a267a310b5ae68e9d8e5	0
00001b22f846c82c51f6e3958ccd81970162bae8b007e80662ef27519fcc18c1	0
000041bd6a6ecadd89a52d11886e8eaec9325906c9723355abb5ca523658edc	0
00007889e4fcd2614b6cbe7f8f3d2e5c728eca32d9eb8ad51ca8b8c4a24cefed	0

```
datos = pd.concat([data,labels.drop(labels.index[len(data.index):-1]), axis = 1)
!tos = datos.drop(datos.index[-1])
```

Python

Imagen 9. Unión de los datos con el target

A continuación, se buscan las variables que tienen una mayor relación con el objetivo, utilizando funciones visuales como matrices de correlación y múltiples gráficas.

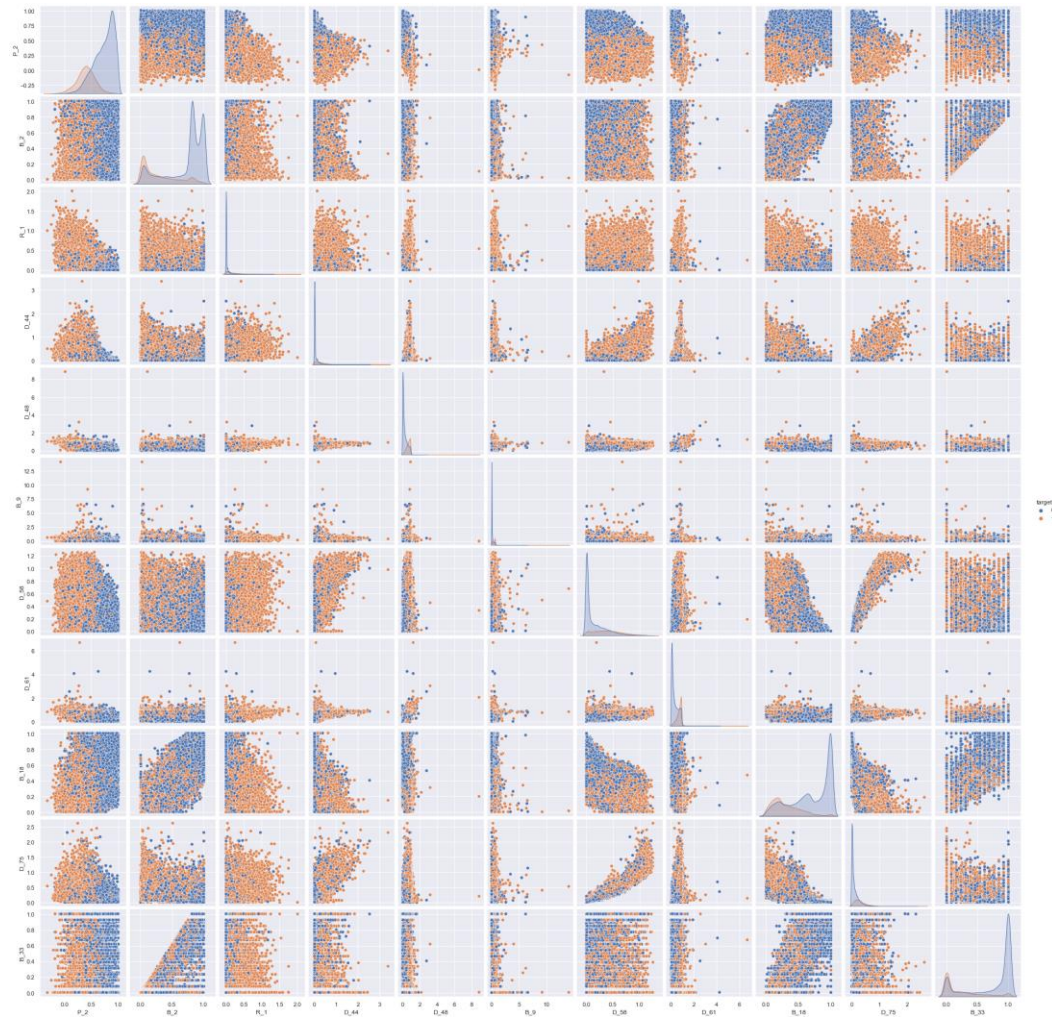


Imagen 10. Pairplot para las variables de mayor correlación.

En la imagen anterior se observa que tan mezclados están los datos donde los puntos azules son los targets 0; es decir, que el usuario tiene un buen historial de pago, mientras que los naranjados representan los usuarios que no cumplen con el pago.

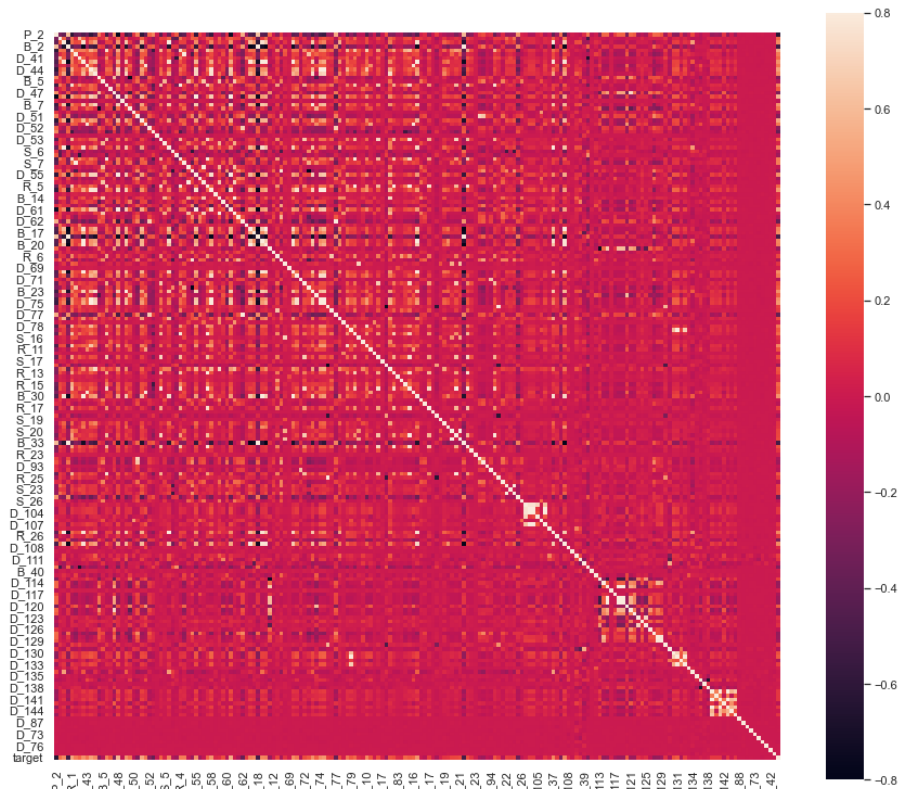


Imagen 11. Matriz de correlación



Imagen 12. Correlación del target con el resto de variables.

Además, es necesario rellenar todos los datos faltantes, tanto los categóricos como los datos numéricos. Obteniendo un dataset con menos ruido para el entrenamiento de los modelos.