Semana 3 - Actividad 1

Nombre: Cristian Reynaldo Miranda Jimenez

Matrícula: A01793718

Materia: Ciencia y Analítica de Datos

Profesor: Jobish Vallikavungal

Asesor: Victoria Guerrero Orozco

Fecha: 2022-10-02

Parte 1: Fundamentos de bases de datos

Revisa la página Lecturas del tema 3: Conceptos de almacenamiento y recuperación de información, y describe tus insights o entendimiento de cada uno de los subtemas: (Alrededor de 200 palabras)

- Fundamentos de bases de datos y para ciencia de datos.
Los datos se pueden categorizar de acuerdo con el tipo: • Estructurados: cuando se organiza en formato tabular como una hoja de calculo • No estructurados: imágenes, documentos de texto, imágenes, etc. Dentro de los datos estructurados, se pueden encontrar los formatos mas comunes como un Excel o un archivo txt. Adiciomamente, existen herramientas para los datos relacionales que se ocupan como Oracle, Ms Access, SQLite, etc. Para poder acceder al conjunto de datos, hay diferentes formas: • GUI: la interfaz grafica para el usuario es generada por softwares como MySql para generar consultas • ODBC: uso de drivers estandarizados para acceder a la información • Lenguajes de programación: para consultas directas se puede hacer uso de R o Python Las tablas pueden llegar a relacionarse de acuerdo a la información que mantienen, de la siguiente manera: • Relación 1 a varios: Cuando en una tabla existen registros únicos de un conjunto de individuos, grupos, etc. mientras que, en otra, los registros se duplican. Por ejemplo, una lista de pacientes y una tabla de registros de citas. • Relación varios a varios: cuando en una tabla se tienen m registros que coinciden con n registros de otra. Para evitar relaciones de varios a varios que generen registros duplicados o métricas erróneas, se debe generar un proceso de normalización de las bases. Los datawarehouse, almacenan la información de distintas fuentes de datos, consolidados en tablas. Los datawarehouse, generalmente, mantienen información histórica y se van alimentando diariamente de los nuevos registros. Uno de los modelos más utilizados en base de datos es el de estrella que consiste en tener una base de hechos en relación de varios a uno con tablas de dimensiones que contienen los principales atributos únicos (clientes, fecha, productos, etc).

- Fundamentos de almacenes de datos (Data Warehouse) para ciencia de datos.
Existe un proceso por el cual deben pasar los datos para consolidarlos en un DataWareHouse denominado: ETL (Extract, Transform and Load) que es la forma tradicional. Un ejemplo es cuando se extrae datos de una pagina web, se limpian de tal forma en que puedan ser cargados a una base

de datos. Algunas de las herramientas para realizar este proceso son Hadoop y SQL. Sin embargo, últimamente se esta optando por un proceso de ELT (Extract, Load, Transform) gracia a los avances computacionales que se han tenido con la creación de clúster para poder procesar toda la información tanto estructurada como no estructurada. ?

Parte 2: Selección y limpieza de los Datos en Python

Revisa detenidamente la página Ejercicio guiado para: Selección y limpieza de los Datos en PythonEnlaces a un sitio externo.

- En Google Colab (= Jupyter notebook, o bien alguna otra IDE de su interés), escribe tu código para realizar la selección y limpieza de los datos como se indica en el ejercicio.
- Ejecuta tu código.
- Explica con tus palabras (documentas las lineas del codigo en celdas del Text) como funciona tu programa indicando lo que realizaste en la programación, en minimum palabras.

Limpieza de datos

La limpieza de datos implica observar más de cerca los problemas en los datos que ha seleccionado incluir en el análisis.

Problema de datos

In [66]:
```
import pandas as pd
import numpy as np
```

Podemos crear un base de datos (DataFrame):

In [67]:
```
x = {'Company': ['Ford', 'Ford', 'VW', 'BMW', 'Cooper', 'Cooper'],
     'Stars' : [1, 2, np.nan, 2, 1, 1],
     'Weight' : [2, 4, 2, 2, 3, None],
     'Origin' : ['China', 'Mexico', 'Mexico', None, 'China', np.nan],
     'Length': [40, 50, 30, np.nan, 45, pd.NaT]
}

df  = pd.DataFrame(data = x)
df
```

Out[67]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30
3	BMW	2.0	2.0	None	NaN
4	Cooper	1.0	3.0	China	45
5	Cooper	1.0	NaN	NaN	NaT

In [70]:
```
# Guardamos en un archivo CSV (coma separated value)
df.to_csv('data.csv')

# Alternativamente, podemos leer un conjunto de datos ya disponible:
df = pd.read_csv('/data.csv', index_col=0)
df
```

Out[70]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30
3	BMW	2.0	2.0	None	NaN
4	Cooper	1.0	3.0	China	45
5	Cooper	1.0	NaN	NaN	NaT

Para verificar, ¿falta algún dato?:

In [71]:
```
df.isnull().values.any()
```

Out[71]:
```
True
```

In [72]:
```
df.isnull().any()
```

Out[72]:
```
Company    False
Stars       True
Weight      True
Origin      True
Length      True
dtype: bool
```

In [73]:
```
# alternativamente
df.isna().values.any()
```

Out[73]:
```
True
```

In [74]:
```
df
```

Out[74]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30
3	BMW	2.0	2.0	None	NaN
4	Cooper	1.0	3.0	China	45
5	Cooper	1.0	NaN	NaN	NaT

Solucion 1:

Descartar las observaciones con valores faltantes

In [75]:
```
df.dropna(inplace=True)
df
```

Out[75]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
4	Cooper	1.0	3.0	China	45

El problema con esta estrategia es que,

> si falta algún dato en todo el conjunto de datos, la fila correspondiente se elimina

In [76]:
```
df  = pd.DataFrame(data = x)
df
```

Out[76]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30
3	BMW	2.0	2.0	None	NaN
4	Cooper	1.0	3.0	China	45
5	Cooper	1.0	NaN	NaN	NaT

In [77]:
```
#Suelte las columnas donde falta al menos un elemento.

ndf = df.copy()
ndf.dropna(axis = 1, inplace = True) # axis 1 is columns / axis 0 is rows.
ndf
```

Out[77]:

	Company
0	Ford
1	Ford
2	VW
3	BMW
4	Cooper

	Company
5	Cooper

In [78]:
```
# Drop the rows where all elements are missing.
ndf = df.copy()
ndf.dropna(how='all', inplace = True)
ndf
```

Out[78]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30
3	BMW	2.0	2.0	None	NaN
4	Cooper	1.0	3.0	China	45
5	Cooper	1.0	NaN	NaN	NaT

Alternativamente: usamos Threshold.

Mantenga solo las filas con al menos 2 valores que **NO SEAN** nan

In [79]:
```
# Drop the rows where all elements are missing.
ndf = df.copy()
ndf.dropna(thresh=4, inplace = True) # In a row, it needs at least 4 nan values is need
ndf # in case of column  add   axis=1
```

Out[79]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30
4	Cooper	1.0	3.0	China	45

In [80]:
```
# Defina en qué columnas buscar valores faltantes.
ndf = df.copy()
ndf.dropna(thresh = 5,  #if there is not 5 nan values, the column will be eliminated
           axis = 1,
           inplace = True
           )
ndf
```

Out[80]:

	Company	Stars	Weight
0	Ford	1.0	2.0
1	Ford	2.0	4.0

	Company	Stars	Weight
2	VW	NaN	2.0
3	BMW	2.0	2.0
4	Cooper	1.0	3.0
5	Cooper	1.0	NaN

Solucion 2:

Imputar

In [81]:
```
ndf = df.copy()

wm = ndf.Weight.mean()
wm
```

Out[81]: 2.6

In [82]:
```
ndf['Weight'].fillna(value = wm,
                     inplace = True)
ndf
```

Out[82]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30
3	BMW	2.0	2.0	None	NaN
4	Cooper	1.0	3.0	China	45
5	Cooper	1.0	2.6	NaN	NaT

In [83]:
```
ndf['Length'].fillna(value = ndf.Length.median(),
                     inplace = True)
ndf
```

Out[83]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40.0
1	Ford	2.0	4.0	Mexico	50.0
2	VW	NaN	2.0	Mexico	30.0
3	BMW	2.0	2.0	None	42.5
4	Cooper	1.0	3.0	China	45.0
5	Cooper	1.0	2.6	NaN	42.5

In [84]:
```
mm  = ndf.Origin.mode()
mm
```

Out[84]:
```
0     China
1    Mexico
dtype: object
```

In [85]:
```
ndf['Origin'].fillna(value = mm[1], #'NoPais',
                     inplace = True)
ndf
```

Out[85]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40.0
1	Ford	2.0	4.0	Mexico	50.0
2	VW	NaN	2.0	Mexico	30.0
3	BMW	2.0	2.0	Mexico	42.5
4	Cooper	1.0	3.0	China	45.0
5	Cooper	1.0	2.6	Mexico	42.5

In [86]:
```
# imputar columnas especificas
ndf = df.copy()
ndf.dropna(subset=['Origin', 'Length'], inplace = True)
ndf
```

Out[86]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30
4	Cooper	1.0	3.0	China	45

In [87]:
```
favs = {'Origin': ndf.Origin.mode()[0], 'Length': ndf['Length'].mean()}
ndf.Origin.fillna(ndf.Origin.mode()[0], inplace=True)
ndf.Length.fillna(ndf.Length.mean(), inplace=True)
ndf
```

Out[87]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30
4	Cooper	1.0	3.0	China	45

¿Cuándo es una mediana mejor en comparación con la media?

In [88]:
```
data = {'Salary':  [28, 30, 30, 35, 37, 40, 400]
}
adf = pd.DataFrame(data)
adf
```

Out[88]:

	Salary
0	28
1	30
2	30
3	35
4	37
5	40
6	400

In [89]:
```
adf.describe()
```

Out[89]:

	Salary
count	7.000000
mean	85.714286
std	138.653903
min	28.000000
25%	30.000000
50%	35.000000
75%	38.500000
max	400.000000

Para seleccionar las columnas de la base de datos, puede usar la siguiente codigos:

In [90]:
```
ndf = df.copy()
ndf
```

Out[90]:

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30
3	BMW	2.0	2.0	None	NaN
4	Cooper	1.0	3.0	China	45
5	Cooper	1.0	NaN	NaN	NaT

Syntaxis de loc & iloc

- loc : If you use, `loc` , use the `names`

  > df. `loc` [row-start:row-end, column-start:column_end]

- iloc : If you use, `loc` , use the `indices`

  > df. `iloc` [row-start:row-end, column-start:columnEnd]

In [91]:
```
ndf.columns
```

Out[91]:
```
Index(['Company', 'Stars', 'Weight', 'Origin', 'Length'], dtype='object')
```

In [92]:
```
ndf.columns.sort_values()
```

Out[92]:
```
Index(['Company', 'Length', 'Origin', 'Stars', 'Weight'], dtype='object')
```

In [93]:
```
ndf.loc[2:5 , 'Company':'Origin']  # rows 2 to 5, columns  'Company' to 'Origin'
```

Out[93]:

	Company	Stars	Weight	Origin
2	VW	NaN	2.0	Mexico
3	BMW	2.0	2.0	None
4	Cooper	1.0	3.0	China
5	Cooper	1.0	NaN	NaN

In [94]:
```
favs = ['Stars', 'Weight', 'Origin']
```

In [95]:
```
ndf.loc[2:5 , favs]
```

Out[95]:

	Stars	Weight	Origin
2	NaN	2.0	Mexico
3	2.0	2.0	None
4	1.0	3.0	China
5	1.0	NaN	NaN

In [96]:
```
ndf.iloc[2:5, [1,2, 3]] # iloc  - so, indices
```

Out[96]:

	Stars	Weight	Origin
2	NaN	2.0	Mexico

	Stars	Weight	Origin
3	2.0	2.0	None
4	1.0	3.0	China

In [97]:
```
ndf.columns
```

Out[97]:
```
Index(['Company', 'Stars', 'Weight', 'Origin', 'Length'], dtype='object')
```

In [98]:
```
for i in ndf.columns:
  print(i)
```

```
Company
Stars
Weight
Origin
Length
```

In [99]:
```
ndf.Company.unique()
```

Out[99]:
```
array(['Ford', 'VW', 'BMW', 'Cooper'], dtype=object)
```

In [100…
```
df.groupby(['Company', 'Origin']).size()
```

Out[100…
```
Company  Origin
Cooper   China     1
Ford     China     1
         Mexico    1
VW       Mexico    1
dtype: int64
```

In [101…
```
df[['Company', 'Origin']].value_counts()
```

Out[101…
```
Company  Origin
Cooper   China     1
Ford     China     1
         Mexico    1
VW       Mexico    1
dtype: int64
```

Eliminar columns / Cambiar nombre de las columns

In [102…
```
ndf
```

Out[102…

	Company	Stars	Weight	Origin	Length
0	Ford	1.0	2.0	China	40
1	Ford	2.0	4.0	Mexico	50
2	VW	NaN	2.0	Mexico	30

	Company	Stars	Weight	Origin	Length
3	BMW	2.0	2.0	None	NaN
4	Cooper	1.0	3.0	China	45
5	Cooper	1.0	NaN	NaN	NaT

In [103…

```
ndf2 = ndf.drop(['Stars', 'Origin',], axis = 1)
ndf2
```

Out[103…

	Company	Weight	Length
0	Ford	2.0	40
1	Ford	4.0	50
2	VW	2.0	30
3	BMW	2.0	NaN
4	Cooper	3.0	45
5	Cooper	NaN	NaT

In [104…

```
ndf2.rename(columns = {'Company' : 'Empresa', 'Weight': 'Peso'}, inplace = True)
ndf2
```

Out[104…

	Empresa	Peso	Length
0	Ford	2.0	40
1	Ford	4.0	50
2	VW	2.0	30
3	BMW	2.0	NaN
4	Cooper	3.0	45
5	Cooper	NaN	NaT

Datos perdidos - Tener en cuenta

- Excluya las filas o características.
- Cumpliméntelas con un valor estimado.

Errores de datos Utilice recursos lógicos para descubrir errores manuales y corríjalos. O, excluya las características.

Incoherencias de codificación Decida un esquema de codificación simple y convierta y sustituya los valores.

Metadatos perdidos o erróneos Examine manualmente los campos sospechosos y compruebe el significado correcto.

Crear un informe de limpieza de datos

Registrar sus actividades de limpieza de datos es esencial para registrar las modificaciones de los datos.

Los futuros proyectos de minería de datos se beneficiarán de los detalles del trabajo disponible.

Es una excelente idea considerar las siguientes cuestiones cuando genere el informe:

- ¿Qué tipos de ruido se han producido en los datos?
- ¿Qué métodos utiliza para eliminar el ruido?

> ¿Qué técnicas han demostrado ser eficaces?

- ¿Existen casos o atributos que no se pueden recuperar?

> Asegúrese de registrar los datos que se han excluido por causas del ruido.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Parte 3: Preparación de los datos

Librerias

In [105…

```
import pandas as pd
import numpy as np
```

Carga de datos

In [106…

```
path = "https://raw.githubusercontent.com/PosgradoMNA/Actividades_Aprendizaje-/main/def

data = pd.read_csv(path, sep= ",")
data.head()
```

Out[106…

	ID	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X15	X16	X17	X18	X19
0	1	20000	2.0	2.0	1.0	24.0	2.0	2.0	-1.0	-1.0	...	0.0	0.0	0.0	0.0	689.0

	ID	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X15	X16	X17	X18	X19
1	2	120000	2.0	2.0	2.0	26.0	-1.0	2.0	0.0	0.0	...	3272.0	3455.0	3261.0	0.0	1000.0
2	3	90000	2.0	2.0	2.0	34.0	0.0	0.0	0.0	0.0	...	14331.0	14948.0	15549.0	1518.0	1500.0
3	4	50000	2.0	2.0	1.0	37.0	0.0	0.0	0.0	0.0	...	28314.0	28959.0	29547.0	2000.0	2019.0
4	5	50000	1.0	2.0	1.0	57.0	-1.0	0.0	-1.0	0.0	...	20940.0	19146.0	19131.0	2000.0	36681.0

5 rows × 25 columns

In [107…

```
# Renombre de columnas
data.rename(columns = {'X1': 'MontoCredito', 'X2': 'Genero', 'X3' : 'NivelEducacion',
                       'X4': 'EstadoCivil',  'X5': 'Edad',   'X6' : 'HistorialPagoSep',
                       'X7': 'HistorialPagoAgo', 'X8': 'HistorialPagoJul', 'X9' : 'Hist
                       'X10': 'HistorialPagoMay', 'X11': 'HistorialPagoAbr', 'X12' : 'E
                       'X13': 'EstadoCuentaAgo', 'X14': 'EstadoCuentaJul', 'X15' : 'Est
                       'X16': 'EstadoCuentaMay', 'X17': 'EstadoCuentaAbr', 'X18' : 'Mon
                       'X19': 'MontoPagoAnteriorAgo', 'X20': 'MontoPagoAnteriorJul',
                       'X21' : 'MontoPagoAnteriorJun', 'X22': 'MontoPagoAnteriorMay',
                       'X23': 'MontoPagoAnteriorAbr'}, inplace = True)

data.columns
```

Out[107…

```
Index(['ID', 'MontoCredito', 'Genero', 'NivelEducacion', 'EstadoCivil', 'Edad',
       'HistorialPagoSep', 'HistorialPagoAgo', 'HistorialPagoJul',
       'HistorialPagoJun', 'HistorialPagoMay', 'HistorialPagoAbr',
       'EstadoCuentaSep', 'EstadoCuentaAgo', 'EstadoCuentaJul',
       'EstadoCuentaJun', 'EstadoCuentaMay', 'EstadoCuentaAbr',
       'MontoPagoAnteriorSep', 'MontoPagoAnteriorAgo', 'MontoPagoAnteriorJul',
       'MontoPagoAnteriorJun', 'MontoPagoAnteriorMay', 'MontoPagoAnteriorAbr',
       'Y'],
      dtype='object')
```

In [108…

```
# Revision de variables:
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ID                    30000 non-null  int64
 1   MontoCredito          30000 non-null  int64
 2   Genero                29999 non-null  float64
 3   NivelEducacion        29998 non-null  float64
 4   EstadoCivil           29998 non-null  float64
 5   Edad                  29995 non-null  float64
 6   HistorialPagoSep      29997 non-null  float64
 7   HistorialPagoAgo      29995 non-null  float64
 8   HistorialPagoJul      29993 non-null  float64
 9   HistorialPagoJun      29991 non-null  float64
 10  HistorialPagoMay      29984 non-null  float64
 11  HistorialPagoAbr      29986 non-null  float64
 12  EstadoCuentaSep       29989 non-null  float64
 13  EstadoCuentaAgo       29989 non-null  float64
```

```
 14  EstadoCuentaJul       29987 non-null  float64
 15  EstadoCuentaJun       29985 non-null  float64
 16  EstadoCuentaMay       29983 non-null  float64
 17  EstadoCuentaAbr       29990 non-null  float64
 18  MontoPagoAnteriorSep  29992 non-null  float64
 19  MontoPagoAnteriorAgo  29991 non-null  float64
 20  MontoPagoAnteriorJul  29992 non-null  float64
 21  MontoPagoAnteriorJun  29989 non-null  float64
 22  MontoPagoAnteriorMay  29989 non-null  float64
 23  MontoPagoAnteriorAbr  29995 non-null  float64
 24  Y                     29997 non-null  float64
dtypes: float64(23), int64(2)
memory usage: 5.7 MB
```

In [109…

```
(data.describe()).transpose()
```

Out[109…

	count	mean	std	min	25%	50%	75%
ID	30000.0	15000.500000	8660.398374	1.0	7500.75	15000.5	22500.25
MontoCredito	30000.0	167484.322667	129747.661567	10000.0	50000.00	140000.0	240000.00
Genero	29999.0	1.603753	0.489125	1.0	1.00	2.0	2.00
NivelEducacion	29998.0	1.853057	0.790320	0.0	1.00	2.0	2.00
EstadoCivil	29998.0	1.551903	0.521968	0.0	1.00	2.0	2.00
Edad	29995.0	35.484214	9.218024	21.0	28.00	34.0	41.00
HistorialPagoSep	29997.0	-0.016635	1.123829	-2.0	-1.00	0.0	0.00
HistorialPagoAgo	29995.0	-0.133689	1.197254	-2.0	-1.00	0.0	0.00
HistorialPagoJul	29993.0	-0.166405	1.196048	-2.0	-1.00	0.0	0.00
HistorialPagoJun	29991.0	-0.220800	1.169153	-2.0	-1.00	0.0	0.00
HistorialPagoMay	29984.0	-0.266342	1.133296	-2.0	-1.00	0.0	0.00
HistorialPagoAbr	29986.0	-0.291136	1.150134	-2.0	-1.00	0.0	0.00
EstadoCuentaSep	29989.0	51236.862750	73645.219278	-165580.0	3565.00	22387.0	67139.00
EstadoCuentaAgo	29989.0	49190.734669	71183.385123	-69777.0	2986.00	21207.0	64027.00
EstadoCuentaJul	29987.0	47025.350152	69360.863317	-157264.0	2667.50	20089.0	60182.00
EstadoCuentaJun	29985.0	43275.652326	64345.500073	-170000.0	2329.00	19052.0	54560.00
EstadoCuentaMay	29983.0	40324.493980	60809.984983	-81334.0	1763.50	18107.0	50213.00
EstadoCuentaAbr	29990.0	38881.135745	59561.312967	-339603.0	1256.25	17081.0	49208.25
MontoPagoAnteriorSep	29992.0	5662.945886	16564.165089	0.0	1000.00	2100.0	5006.00
MontoPagoAnteriorAgo	29991.0	5922.488913	23044.177075	0.0	835.50	2009.0	5000.00
MontoPagoAnteriorJul	29992.0	5225.623400	17608.422625	0.0	390.00	1800.0	4505.50
MontoPagoAnteriorJun	29989.0	4827.252526	15668.751975	0.0	296.00	1500.0	4014.00
MontoPagoAnteriorMay	29989.0	4800.297209	15280.842069	0.0	251.00	1500.0	4033.00

	count	mean	std	min	25%	50%	75%
MontoPagoAnteriorAbr	29995.0	5216.259977	17778.848359	0.0	118.00	1500.0	4000.00
Y	29997.0	0.221189	0.415054	0.0	0.00	0.0	0.00

Con las tablas anteriores, se puede determinar que existen algunas variables con datos que no concuerdan con la descripción, tal es el caso de el nivel de educacion con 6 niveles (2 de más) y el estado civil.

In [116…

```
# Revisando datos nulos
print(data.isnull().sum())
```

```
ID                         0
MontoCredito               0
Genero                     1
NivelEducacion             2
EstadoCivil                2
Edad                       5
HistorialPagoSep           3
HistorialPagoAgo           5
HistorialPagoJul           7
HistorialPagoJun           9
HistorialPagoMay          16
HistorialPagoAbr          14
EstadoCuentaSep           11
EstadoCuentaAgo           11
EstadoCuentaJul           13
EstadoCuentaJun           15
EstadoCuentaMay           17
EstadoCuentaAbr           10
MontoPagoAnteriorSep       8
MontoPagoAnteriorAgo       9
MontoPagoAnteriorJul       8
MontoPagoAnteriorJun      11
MontoPagoAnteriorMay      11
MontoPagoAnteriorAbr       5
Y                          3
dtype: int64
```

Existen datos nulos en todas las variables menos `ID` y `MontoCredito`

In [127…

```
# Eliminando valores perdidos

nData = data.copy()
nData.dropna(axis = 0, inplace = True)

print("Base sin NA's:",nData.shape[0],"\nBase completa:", data.shape[0])
print("Filas eliminadas:",data.shape[0]-nData.shape[0])
print("% de vacios: ", round((1-(nData.shape[0]/data.shape[0]))*100 ,2),"%")
```

```
Base sin NA's: 29958
Base completa: 30000
Filas eliminadas: 42
% de vacios:  0.14 %
```

Como no es adecuado eliminar todas las columnas con vacios, se procede a analizar el % de datos nulos para proceder a sacarlos de la base. Al ser un porcentaje menor a 1%, esto no tiene una afectación tan significativa.

Limpieza de datos

A continuación, se corrigen las categorias de las variables con inconsistencias

In [143…

```
print(nData.groupby('Genero').size())
print("\n",nData.groupby(['EstadoCivil']).size())
print("\n",nData.groupby(['NivelEducacion']).size() )
```

```
Genero
1.0    11863
2.0    18095
dtype: int64

 EstadoCivil
0.0       54
1.0    13643
2.0    15939
3.0      322
dtype: int64

 NivelEducacion
0.0       14
1.0    10572
2.0    14009
3.0     4909
4.0      123
5.0      280
6.0       51
dtype: int64
```

Para el caso del `EstadoCivil` , se corrige la categoria 0 con la moda.De igual forma con el `NivelEducacion`

In [144…

```
# Modas
modaEstadoCivil = nData['EstadoCivil'].mode()[0]
modaNivelEducacion = nData['NivelEducacion'].mode()[0]

# Reemplazo
nData['EstadoCivil'] = nData['EstadoCivil'].map({0:modaEstadoCivil, 1:1, 2:2, 3:3})
nData['NivelEducacion'] = nData['NivelEducacion'].map({0:4, 1:1, 2:2, 3:3, 4:4, 5:4, 6:
```

In [ ]:

```
```

Revisión final

Se procede a revisar nuevamente el dataset limpio para validar el procedimiento anterior

In [146…

```
nData.describe().transpose()
```

Out[146…

	count	mean	std	min	25%	50%	75%
ID	29958.0	15005.550504	8654.547473	1.0	7516.25	15005.5	22497.75
MontoCredito	29958.0	167555.900928	129737.299088	10000.0	50000.00	140000.0	240000.00
Genero	29958.0	1.604012	0.489070	1.0	1.00	2.0	2.00
NivelEducacion	29958.0	1.842212	0.744557	1.0	1.00	2.0	2.00
EstadoCivil	29958.0	1.555344	0.518115	1.0	1.00	2.0	2.00
Edad	29958.0	35.483443	9.214319	21.0	28.00	34.0	41.00
HistorialPagoSep	29958.0	-0.017124	1.123989	-2.0	-1.00	0.0	0.00
HistorialPagoAgo	29958.0	-0.134021	1.197171	-2.0	-1.00	0.0	0.00
HistorialPagoJul	29958.0	-0.166767	1.196026	-2.0	-1.00	0.0	0.00
HistorialPagoJun	29958.0	-0.221110	1.168419	-2.0	-1.00	0.0	0.00
HistorialPagoMay	29958.0	-0.266807	1.132307	-2.0	-1.00	0.0	0.00
HistorialPagoAbr	29958.0	-0.291575	1.149303	-2.0	-1.00	0.0	0.00
EstadoCuentaSep	29958.0	51248.119901	73674.949943	-165580.0	3559.25	22379.0	67190.00
EstadoCuentaAgo	29958.0	49200.493825	71211.232744	-69777.0	2984.00	21194.5	64027.75
EstadoCuentaJul	29958.0	47032.385273	69385.243340	-157264.0	2664.75	20085.5	60183.00
EstadoCuentaJun	29958.0	43279.335370	64364.684347	-170000.0	2327.50	19037.5	54551.25
EstadoCuentaMay	29958.0	40328.984578	60826.219326	-81334.0	1762.25	18104.5	50220.75
EstadoCuentaAbr	29958.0	38889.925763	59582.883301	-339603.0	1256.00	17067.5	49234.75
MontoPagoAnteriorSep	29958.0	5664.614460	16568.823518	0.0	1000.00	2100.0	5007.00
MontoPagoAnteriorAgo	29958.0	5925.715468	23055.983864	0.0	835.25	2009.0	5000.00
MontoPagoAnteriorJul	29958.0	5228.429969	17617.338167	0.0	390.00	1800.0	4511.50
MontoPagoAnteriorJun	29958.0	4829.873556	15676.205514	0.0	296.25	1500.0	4014.75
MontoPagoAnteriorMay	29958.0	4801.481574	15285.552652	0.0	253.25	1500.0	4040.00
MontoPagoAnteriorAbr	29958.0	5220.708025	17788.983767	0.0	118.00	1500.0	4000.00
Y	29958.0	0.221143	0.415023	0.0	0.00	0.0	0.00

Ahora, los datos categoricos son consistentes y ya no existen valores perdidos en el dataset por lo que se puede proceder con el análisis

In [ ]:

Análisis de datos

In [153…

```
nData.groupby(['Y']).size() / nData.shape[0]
```

Out[153…
```
Y
0.0    0.778857
1.0    0.221143
dtype: float64
```

El 22% del total de clientes disponen el servicio

In [ ]:

In [155…
```
nData.groupby(['Genero','Y']).size()
```

Out[155…
```
Genero  Y
1.0     0.0     8998
        1.0     2865
2.0     0.0    14335
        1.0     3760
dtype: int64
```

Al parecer más mujeres utilizan el servicio que hombres, con alrededor de mil más usuarios.

In [156…
```
nData.groupby(['Y','Genero','EstadoCivil']).size()
```

Out[156…
```
Y    Genero  EstadoCivil
0.0  1.0     1.0            3838
             2.0            5069
             3.0              91
     2.0     1.0            6603
             2.0            7584
             3.0             148
1.0  1.0     1.0            1342
             2.0            1483
             3.0              40
     2.0     1.0            1860
             2.0            1857
             3.0              43
dtype: int64
```

Repuestas de preguntas

- ¿Qué datos considero mas importantes? ¿Por qué? Considero que las variables importas de acuerdo al análisis que se lleve a cabo. Para esta caso, las variables más relevantes podrian ser las categóricas y con corte actual. Sin embargo, se pueden realizar análisis mucho más profundos mediante técnicas estadísticas descriptivas para evaluar las mejores.

- ¿Se eliminaron o reemplazaron datos nulos? ¿Qué se hizo y por qué? Se eliminaron los datos nulos. Esto debido a que la cantidad era baja, representando menos del 1% de valores perdidos por lo que excluirlos del análisis genera un ruido en la data de manera rápida.

- ¿Es necesario ordenar los datos para el análisis? Sí / No / ¿Por qué? No es necesario ordenar los datos debido a que este no altera el resultado final. Adicionalmente, se entiende que los datos son con un corte transversal de tiempo y para este caso no es relevante el orden a diferencia de contar con datos temporales como las series de tiempo.

- ¿Existen problemas de formato que deban solucionar antes del proceso de modelado? Sí / No / Por qué. Sí exiten por ejemplo en las variables categoricas. Este problema debe ser corregido ya que de no hacerlo puede generar ruido en los modelos que se implementen posteriormente.

- ¿Qué ajustes se realizaron en el proceso de limpieza de datos (agregar, integrar, eliminar, modificar registros (filas), cambiar atributos (columnas)? En el proceso de limpieza se eliminaron registros con valores perdidos y se corrigió las categorias de algunas variables que no estaban consideradas dentro de la descripción de la base de datos.

In [ ]: