Tecnológico de Monterrey

Semana 3 – Actividad 1

CIENCIA Y ANALITICA DE DATOS

EQUIPO 12

Guillermo Alfonso Muñiz Hermosillo - A01793101

José Ramiro Adán Charles - A00174646

Parte 1: Fundamentos de bases de datos

Introducción

Los datos son una fuente ilimitada de conocimiento, la cual nos puede brindar perspectivas únicas sobre cualquier aspecto de nuestra existencia. Podemos recopilar datos en cualquier lugar al que deseemos observar y analizar; desde nuestra actividad en línea, nuestra actividad bancaria, nuestros patrones de compa, los lugares a los que visitamos, el lugar donde trabajamos y una infinidad de lugares comunes y extraordinarios mas, los datos siempre están presentes y es el trabajo de los buenos analistas y científicos de datos identificar y explotar el potencial oculto en todos ellos. Es por esto por lo que un analista de datos tiene que escoger las herramientas más adecuadas para poder llevar a cabo tan excepcional tarea.

Fundamentos de bases de datos y para ciencia de datos.

Una persona interesada en el análisis de datos necesita tener las mejores herramientas a su disposición, mediante la lectura de esta semana hemos podido identificar a la base de datos como una de las herramientas básicas para llevar a cabo un análisis coherente, estructurado y significativo de los datos. Nosotros en base a nuestra perspectiva, definimos una base de datos como sigue:

“Una base de datos es un sistema informático, capaz de almacenar información estructurada y relacionada sobre diversas líneas de interés; Todo esto con el fin de que la información no solo pueda ser guardada sino permitir que sea analizada, modificada y procesada de una manera eficiente para permitir obtener múltiples beneficios de ella”.

Así mismo, después de nuestro análisis hemos identificado los componentes más importantes de un sistema de base de datos:

- El más importante es su sistema de administración (DBMS), este componente es el alma del sistema de base de datos. Mediante el podemos encargarnos de la administración y control de la información almacenada en la base de datos.
- Las tablas es la manera en que nuestra base de datos será organizada. Como su nombre lo dice, es una unión de filas y columnas en las cuales se almacenará la información de la base de datos.
- Cada fila equivale a un registro en la base de datos; una pieza de información recolectada lista para ser analizada.
- Cada columna indica un atributo o campo de la información contenida en la base de datos. Con cada campo podemos llevar nuestro análisis a un nuevo nivel de procesamiento. Nosotros lo consideramos como la unidad básica de una base de datos.
- Las consultas son una parte fundamental del análisis de datos, cada sistema de base de datos viene de la mano con información ordenada que puede ser consultada. El método más común son las consultas por SQL, aunque hoy en día existen nuevas técnicas como las consultas de datos visuales (como Tableu) que amplían la gama de capacidades para la obtención de perspectivas dentro del sistema de base de datos.

Hoy en día para los analistas, científicos y aficionados del análisis de información, las bases de datos representan una herramienta fundamental para el desempeño de su trabajo. Mas adelante abordaremos como el uso de una base de datos dependerá en gran medida de la cantidad de datos con los cuales se trabajará, así como el número de fuentes con las que se lleva a cabo el análisis de datos. Sin embargo, es inequívoco que aquella persona que quiera elevar su nivel como analista de datos debe de tener en su repertorio de habilidades la capacidad y competencia para trabajar con estos sistemas tan elementales.

Fundamentos de almacenes de datos (Data Warehouse) para ciencia de datos.

La tecnología de la base de datos como la conocemos hoy en día, si bien es ampliamente funcional, es una tecnología ya algo antigua ya que data del siglo pasado; dicha tecnología ha venido adaptándose constantemente a las necesidades actuales. Sin embargo, su estructura continúa siendo prácticamente la misma. Las bases de datos nos ligan a un sistema en el cual necesitamos procesar y limpiar los datos antes de poder cargarlos a nuestra base de datos, ya que estos deben ser organizados usualmente de manera tabula. Este modelo nos trae un gran costo al procesar datos semiestructurados y no estructurados ya que no pueden ser convertidos a dicho formato fácilmente... Así mismo, en un mundo cada vez más abierto e interconectado, la cantidad de datos ha crecido exponencialmente y el tratamiento de datos a través de una sola base de datos las hace lentas y con un bajo rendimiento. Es así como, partiendo de estas necesidades, hemos identificado que han surgido los almacenes de datos (Data Warehouses).

Un almacén de datos es un contenedor centralizado de datos de todos tipos, de diferentes fuentes e incluso con distintos propósitos. Entre ellos: la ciencia de datos, el diseño de algoritmos de aprendizaje automático, la toma de decisiones etc...

A diferencia de las bases de datos, los almacenes de datos tienen un enfoque diseñado con el modelo ELT (extracción, carga y transformación) esto quiere decir que el procesamiento de la información se da después de haber cargado todos los datos disponibles. Esto nos ayuda a reducir costos de procesamiento ya que, si un conjunto de datos se vuelve muy grande, el procesamiento para convertir datos a tablas relacionales es costoso y lento. En los almacenes de datos este procesamiento se da en la última etapa, una vez que los datos han sido cargados, mediante el uso de nuevas técnicas de procesamiento las cuales nos ayudan a interpretar y clasificar los datos de una manera más eficiente y distribuida. Así mismo nos ayuda a crear una armonía entre datos de tipos muy variados y de fuentes dispares, ampliando nuestro horizonte pare crear modelos que sean capaces de tomar las mejores decisiones o proveer las mejores recomendaciones.

Parte2 Selección y limpieza de los Datos en Python

October 4, 2022

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
import ssl
import os
import requests

# Este workaround nos sirve para agregar certificados SSL ya que estamos␣
 ↪trabajando en MAC
ssl._create_default_https_context = ssl._create_unverified_context

# Declaramos la direccion del archivo csv que vamos a usar
url = 'https://raw.githubusercontent.com/PosgradoMNA/Actividades_Aprendizaje-/
 ↪main/default%20of%20credit%20card%20clients.csv'
# Creamos el path con el nombre que tendra mi archivo
path=os.path.join(os.getcwd(),'creditcardclients.csv')
# Enviamos un request para obtener el archivo
r=requests.get(url)
# Usando la request, creo un archivo nuevo y escribimos el contenido de la␣
 ↪request.
with open(path,'wb') as f:
    f.write(r.content)
# Creamos mi dataframe leyendo el archivo creado.
df = pd.read_csv("creditcardclients.csv")

# Para verificar los datos creados, imprimimos los primeros 5 registros
df.head()
```

[2]:

	ID	X1	X2	X3	X4	X5	X6	X7	X8	X9	…	X15	X16	\
0	1	20000	2.0	2.0	1.0	24.0	2.0	2.0	-1.0	-1.0	…	0.0	0.0	
1	2	120000	2.0	2.0	2.0	26.0	-1.0	2.0	0.0	0.0	…	3272.0	3455.0	
2	3	90000	2.0	2.0	2.0	34.0	0.0	0.0	0.0	0.0	…	14331.0	14948.0	
3	4	50000	2.0	2.0	1.0	37.0	0.0	0.0	0.0	0.0	…	28314.0	28959.0	
4	5	50000	1.0	2.0	1.0	57.0	-1.0	0.0	-1.0	0.0	…	20940.0	19146.0	

	X17	X18	X19	X20	X21	X22	X23	Y
0	0.0	0.0	689.0	0.0	0.0	0.0	0.0	1.0

```
1   3261.0     0.0   1000.0   1000.0  1000.0     0.0  2000.0  1.0
2  15549.0  1518.0   1500.0   1000.0  1000.0  1000.0  5000.0  0.0
3  29547.0  2000.0   2019.0   1200.0  1100.0  1069.0  1000.0  0.0
4  19131.0  2000.0  36681.0  10000.0  9000.0   689.0   679.0  0.0

[5 rows x 25 columns]
```

Verificamos si hay datos nulos

[3]:
```
df.isnull().any()
```

[3]:
```
ID     False
X1     False
X2      True
X3      True
X4      True
X5      True
X6      True
X7      True
X8      True
X9      True
X10     True
X11     True
X12     True
X13     True
X14     True
X15     True
X16     True
X17     True
X18     True
X19     True
X20     True
X21     True
X22     True
X23     True
Y       True
dtype: bool
```

Para verificar que datos son nulos. Guardamos en un Dataframe aquellos que lo son.

[4]:
```
datosNulos = df.isnull()
```

Para obtener el numero de datos nulos por columna, Creamos un dataframe con la suma de los registros que tienen valores (TRUE) y los que no tienen valores (FALSE)

[5]:
```
rows = []
for columna in datosNulos.columns.values.tolist():
    rows.append({
```

```
        'ColumnName':columna,
        'True': datosNulos[columna].value_counts().values[0],
        'False': datosNulos[columna].value_counts().values[1] if␣
 ↪len(datosNulos[columna].value_counts().values) > 1 else 0
    })
nulldf = pd.DataFrame(rows)
nulldf
```

[5]:

	ColumnName	True	False
0	ID	30000	0
1	X1	30000	0
2	X2	29999	1
3	X3	29998	2
4	X4	29998	2
5	X5	29995	5
6	X6	29997	3
7	X7	29995	5
8	X8	29993	7
9	X9	29991	9
10	X10	29984	16
11	X11	29986	14
12	X12	29989	11
13	X13	29989	11
14	X14	29987	13
15	X15	29985	15
16	X16	29983	17
17	X17	29990	10
18	X18	29992	8
19	X19	29991	9
20	X20	29992	8
21	X21	29989	11
22	X22	29989	11
23	X23	29995	5
24	Y	29997	3

1 UNIFORMIDAD Y PROCESAMIENTO DE DATOS

[6]:
```
## Para que fuera mas claro para nosotros remplazaremos el nombre de las␣
 ↪columnas
newColumnNames = ['ID','Total_Credito', 'Sexo', 'Estudios', 'Estado_Civil',␣
 ↪'Edad',␣
 ↪'PPSep2005','PPAgo2005','PPJul2005','PPJun2005','PPMay2005','PPAbr2005',␣
 ↪'TRSep2005', 'TRAgo2005','TRJul2005','TRJun2005','TRMay2005','TRAbr2005',␣
 ↪'TPPSep2005','TPPAgo2005','TPPJul2005','TPPJun2005','TPPMay2005','TPPAbr2005',␣
 ↪'Y']
# PP = Pagos Pasados
```

```
# TR = Total del Recibo
# TPP = Total de Pagos Pasados
df.columns = newColumnNames # Remplazamos los nombres de las columnas
nulldf['ColumnName'] = df.columns # Remplazamos los valores de las columnas en
 ↪el dataframe de Nulos para mantener consistencia
df.head()
```

[6]:

	ID	Total_Credito	Sexo	Estudios	Estado_Civil	Edad	PPSep2005	\
0	1	20000	2.0	2.0	1.0	24.0	2.0	
1	2	120000	2.0	2.0	2.0	26.0	-1.0	
2	3	90000	2.0	2.0	2.0	34.0	0.0	
3	4	50000	2.0	2.0	1.0	37.0	0.0	
4	5	50000	1.0	2.0	1.0	57.0	-1.0	

	PPAgo2005	PPJul2005	PPJun2005	…	TRJun2005	TRMay2005	TRAbr2005	\
0	2.0	-1.0	-1.0	…	0.0	0.0	0.0	
1	2.0	0.0	0.0	…	3272.0	3455.0	3261.0	
2	0.0	0.0	0.0	…	14331.0	14948.0	15549.0	
3	0.0	0.0	0.0	…	28314.0	28959.0	29547.0	
4	0.0	-1.0	0.0	…	20940.0	19146.0	19131.0	

	TPPSep2005	TPPAgo2005	TPPJul2005	TPPJun2005	TPPMay2005	TPPAbr2005	Y
0	0.0	689.0	0.0	0.0	0.0	0.0	1.0
1	0.0	1000.0	1000.0	1000.0	0.0	2000.0	1.0
2	1518.0	1500.0	1000.0	1000.0	1000.0	5000.0	0.0
3	2000.0	2019.0	1200.0	1100.0	1069.0	1000.0	0.0
4	2000.0	36681.0	10000.0	9000.0	689.0	679.0	0.0

[5 rows x 25 columns]

[7]:
```
nulldf.head()
```

[7]:

	ColumnName	True	False
0	ID	30000	0
1	Total_Credito	30000	0
2	Sexo	29999	1
3	Estudios	29998	2
4	Estado_Civil	29998	2

Lo primero que decidimos analizar fue si era conveniente descartar los valores faltantes.

[8]:
```
dropdf = df.copy()
dropdf.dropna(inplace=True) #Eliminamos los valores nulos, asignando esto a un
 ↪nuevo data frame para mantener nuestros datos originales
dropdf.head()
```

[8]:
```
   ID  Total_Credito  Sexo  Estudios  Estado_Civil  Edad  PPSep2005  \
0   1          20000   2.0       2.0           1.0  24.0        2.0
1   2         120000   2.0       2.0           2.0  26.0       -1.0
2   3          90000   2.0       2.0           2.0  34.0        0.0
3   4          50000   2.0       2.0           1.0  37.0        0.0
4   5          50000   1.0       2.0           1.0  57.0       -1.0

   PPAgo2005  PPJul2005  PPJun2005  …  TRJun2005  TRMay2005  TRAbr2005  \
0        2.0       -1.0       -1.0  …        0.0        0.0        0.0
1        2.0        0.0        0.0  …     3272.0     3455.0     3261.0
2        0.0        0.0        0.0  …    14331.0    14948.0    15549.0
3        0.0        0.0        0.0  …    28314.0    28959.0    29547.0
4        0.0       -1.0        0.0  …    20940.0    19146.0    19131.0

   TPPSep2005  TPPAgo2005  TPPJul2005  TPPJun2005  TPPMay2005  TPPAbr2005    Y
0         0.0       689.0         0.0         0.0         0.0         0.0  1.0
1         0.0      1000.0      1000.0      1000.0         0.0      2000.0  1.0
2      1518.0      1500.0      1000.0      1000.0      1000.0      5000.0  0.0
3      2000.0      2019.0      1200.0      1100.0      1069.0      1000.0  0.0
4      2000.0     36681.0     10000.0      9000.0       689.0       679.0  0.0

[5 rows x 25 columns]
```

[9]:
```
print("ORIGINAL SHAPE: ", df.shape)
print("DROP NA SHAPE: ", dropdf.shape)
```

```
ORIGINAL SHAPE:  (30000, 25)
DROP NA SHAPE:  (29958, 25)
```

[10]:
```
dropdf.isna().values.any() # Verificamos que ya no existan valores nulos.
```

[10]:
```
False
```

Se han eliminado un total de 42 filas de datos del conjunto de datos. Considerando el tamaño original del conjunto, podria ser no tan relevante.

Definitivamente podriamos continuar usando este nuevo conjunto de dato y agregar las anotaciones pertinentes y teniendo precaucion en los resultados obtenidos al modelar.

Sin embargo, si queremos realizar un analisis de datos sin eliminar informacion necesitamos una manera no tan drastica para el tratamiento de datos nulos.

1.1 ANALIZANDO DATOS NULOS Y TECNICAS DE LIMPIEZA

[11]:
```
# Copiamos nuestros datos a un dataframe nuevo, donde realizaremos las␣
 ↪transformaciones y acciones necesarias de limpieza
dfLimpio = df.copy()
print("ORIGINAL DATAFRAME SHAPE: ", df.shape)
print("DATAFRAME LIMPIO SHAPE ORIGINAL: ", dfLimpio.shape)
```

```
ORIGINAL DATAFRAME SHAPE:  (30000, 25)
DATAFRAME LIMPIO SHAPE ORIGINAL:  (30000, 25)
```

Analizaremos las acciones correspondientes a los datos faltantes por columna y llevaremos a cabo la que consideremos la mejor tecnica de limpieza. Para esto, nos conviene primero saber el tipo de datos almacenados en cada variable:

[12]:
```
dfLimpio.dtypes
```

[12]:
```
ID                int64
Total_Credito     int64
Sexo            float64
Estudios        float64
Estado_Civil    float64
Edad            float64
PPSep2005       float64
PPAgo2005       float64
PPJul2005       float64
PPJun2005       float64
PPMay2005       float64
PPAbr2005       float64
TRSep2005       float64
TRAgo2005       float64
TRJul2005       float64
TRJun2005       float64
TRMay2005       float64
TRAbr2005       float64
TPPSep2005      float64
TPPAgo2005      float64
TPPJul2005      float64
TPPJun2005      float64
TPPMay2005      float64
TPPAbr2005      float64
Y               float64
dtype: object
```

1.2 Procedemos entonces al analisis por columna:

1.2.1 En la Columna SEXO (2) Solo existe un dato faltante. El tipo de dato es flotantes de 0 a 1. Consideramos por lo tanto que estos valores han sido ya codificados con un encoder para poder ser utilizados al ser esta una variable categorica.

Solo nos ocuparemos pues de remplazar valores faltantes.

[13]:
```
# Primero como sabemos tenemos un valor Nulo existente en esta columna, por lo␣
↪que debe ser limpiado.
dfLimpio['Sexo'].isnull().values.any()
```

[13]:
```
True
```

[14]:
```
# Verificamos el numero de Valores en la columna Sexo
dfLimpio['Sexo'].value_counts()
```

[14]:
```
2.0    18112
1.0    11887
Name: Sexo, dtype: int64
```

Para este caso consideramos pertinente remplazar el valor no existente con aquel que se repite mas: Moda. Debido a que es un valor categorico, la importancia de un solo valor faltante hace conveniente remplazarlo por aquel que sea mas comun en el conjunto de datos, sin un gran costo al modelo futuro.

[15]:
```
# La moda es 2.0 o lo que es equivalente a Mujer (Female) por lo que␣
↪remplazaremos con ese valor la celda que contiene null
moda = dfLimpio.Sexo.mode()[0]
# Remplazamos el registro faltante
dfLimpio.Sexo.fillna(moda, inplace=True)
print("HAY VALORES NULOS?: ", dfLimpio['Sexo'].isnull().values.any())
```

```
HAY VALORES NULOS?:  False
```

[16]:
```
# Despues verificamos que solo existan 2 valores. Ahora se muestra el registro␣
↪añadido a los valores de femenino
dfLimpio['Sexo'].value_counts()
```

[16]:
```
2.0    18113
1.0    11887
Name: Sexo, dtype: int64
```

1.2.2 Para la columnas con valores categoricos, consideramos el mismo tratamiento descrito anteriormente por lo que se llevara a cabo a continuacion

Estas columnas son - Estudios - Estado_Civil

[17]:
```
# Primero verificamos los valores nulos
print('Hay Nulos en Estudios: ',dfLimpio['Estudios'].isnull().values.any())
print('Hay Nulos en Estado Civil: ',dfLimpio['Estado_Civil'].isnull().values.
  ↪any())
```

```
Hay Nulos en Estudios:  True
Hay Nulos en Estado Civil:  True
```

[18]:
```
# Verificamos el numero de valores antes de la limpieza en cada uno
print('Numero de Registros en Estudios:\n',dfLimpio['Estudios'].value_counts())
print('\nNumero de Registros en Estado Civil:\n',dfLimpio['Estado_Civil'].
  ↪value_counts())
print('\nNumero de Nulos en Estudios:',nulldf.loc[3,'False'])
print('Numero de Nulos en Estado Civil:',nulldf.loc[4,'False'])
```

```
Numero de Registros en Estudios:
 2.0    14030
1.0    10585
3.0     4915
5.0      280
4.0      123
6.0       51
0.0       14
Name: Estudios, dtype: int64

Numero de Registros en Estado Civil:
 2.0    15964
1.0    13657
3.0      323
0.0       54
Name: Estado_Civil, dtype: int64

Numero de Nulos en Estudios: 2
Numero de Nulos en Estado Civil: 2
```

[19]:
```
# Calculamos la Moda en ambos casos:
modaEstudios = dfLimpio.Estudios.mode()[0]
modaEstadoCivil = dfLimpio.Estado_Civil.mode()[0]
print('La moda de Estudios es: ', modaEstudios)
print('La moda de Estado Civil es: ', modaEstadoCivil)
```

```
La moda de Estudios es:  2.0
La moda de Estado Civil es:  2.0
```

[20]:
```
# Remplazamos por la moda los registros faltantes
dfLimpio.Estudios.fillna(modaEstudios, inplace=True)
dfLimpio.Estado_Civil.fillna(modaEstadoCivil, inplace=True)
```

```
# Verificamos que los cambios se hayan efectuado.
print("HAY VALORES NULOS EN ESTUDIOS?: ", dfLimpio['Estudios'].isnull().values.
  ↪any())
print("HAY VALORES NULOS EN ESTADO CIVIL?: ", dfLimpio['Estado_Civil'].isnull().
  ↪values.any())

# Verificamos que el numero de Registros haya incrementado en la moda
print('\nNumero de Registros en Estudios:\n',dfLimpio['Estudios'].
  ↪value_counts())
print('\nNumero de Registros en Estado Civil:\n',dfLimpio['Estado_Civil'].
  ↪value_counts())
```

```
HAY VALORES NULOS EN ESTUDIOS?:  False
HAY VALORES NULOS EN ESTADO CIVIL?:  False

Numero de Registros en Estudios:
 2.0    14032
1.0    10585
3.0     4915
5.0      280
4.0      123
6.0       51
0.0       14
Name: Estudios, dtype: int64

Numero de Registros en Estado Civil:
 2.0    15966
1.0    13657
3.0      323
0.0       54
Name: Estado_Civil, dtype: int64
```

Ahora bien como vemos, ya no existen datos de registros faltantes. Sin embargo de acuerdo a la informacion brindade del conjunto de datos hay errores en los valores de los Registros.

- Estudios: Los valores de acuerdo a la guia son: (1 = graduate school; 2 = university; 3 = high school; 4 = others. Se puede observar que hay hasta 7 valores diferentes (0-6) por lo que conviene asumir que para este estudio asumiremos que los valores 0, 5, 6 se sumaran a 4 (Otros) ya que consideramos que de esta manera es como se diseño el modelo mencionado en el estudio.
- Estado Civil: Mismo caso que el anterior, los valores esperados en la guia son: (1 = married; 2 = single; 3 = others) Por lo que 1 y 2 permaneceran intactos, y los valores 0 los añadiremos a el valor 3 (Otros)

Por lo que Aplicaremos la transformación map() de Python para llevar a cabo la agrupación de los registros considerados como 'otros' y que tambien son los de menor frecuencia. Lo haremos

mediante un diccionario. En la notación a:b, el entero "a" es sustituido por "b".

[21]:
```
# Sustituimos los valores 0, 5 y 6 por un 4 (Otros). Tenemos que mapear los␣
 ↪valores que no modificaran su valor 1,2,3,4
dfLimpio['Estudios'] = dfLimpio['Estudios'].map({0:4, 1:1, 2:2, 3:3, 4:4, 5:4,␣
 ↪6:4})
# Sustitoimos el valor 0 por un 3(Otros). Tenemos que mapear los valores que no␣
 ↪modificaran su valor 1,2,3
dfLimpio['Estado_Civil'] = dfLimpio['Estado_Civil'].map({0:3, 1:1, 2:2, 3:3})

# Verificamos de nuevo que los cambios se hayan efectuado.
print('\nNumero de Registros en Estudios:\n',dfLimpio['Estudios'].
 ↪value_counts())
print('\nNumero de Registros en Estado Civil:\n',dfLimpio['Estado_Civil'].
 ↪value_counts())
```

```
Numero de Registros en Estudios:
 2    14032
1    10585
3     4915
4      468
Name: Estudios, dtype: int64

Numero de Registros en Estado Civil:
 2    15966
1    13657
3      377
Name: Estado_Civil, dtype: int64
```

1.3 Ahora bien, para la variable de edad, este es una variable no categorica, por lo que usar la moda no es algo recomendado.

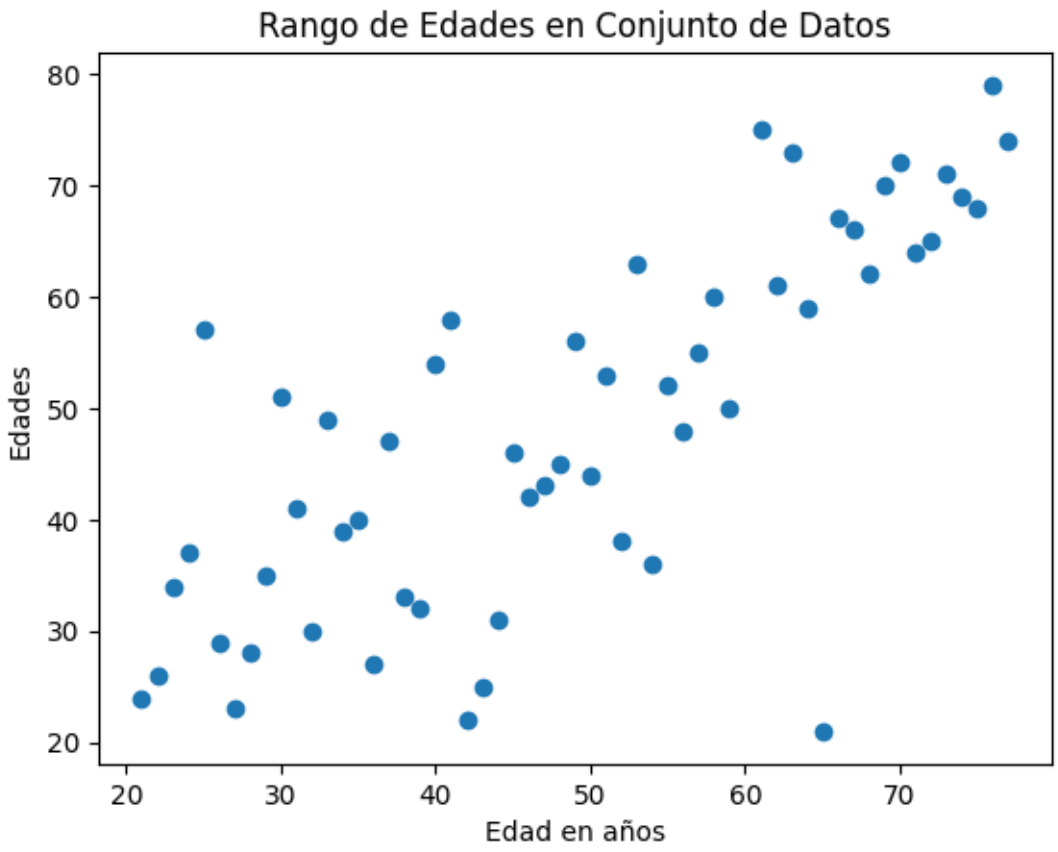
Primero que nada necesitamos analizar los valores de las edades, e intentar encontrar un error de entrada de datos. Ejemplo: Una edad menor a 0 o un outlier de edad alto.

[22]:
```
# Haremos este estudio de manera visual
import matplotlib.pyplot as plt

edad = dfLimpio.Edad.unique()
X = np.arange(min(edad), max(edad-1), 1)

plt.plot(X, edad, 'o')
plt.title('Rango de Edades en Conjunto de Datos')

plt.xlabel('Edad en años')
plt.ylabel('Edades')
plt.show()
```



Observando el grafico nos dimos cuenta que no hay un outlier en edad. Pero si sabemos que hay valores faltantes. Por lo cual debemos de encontrar la mejor manera de usar estos datos para rellenar los valores nulos con la mejor aproximacion.

[23]:
```
# Primero obtendremos las medidas estadisticas de la columna edad
dfLimpio.Edad.describe()
```

[23]:
```
count    29995.000000
mean        35.484214
std          9.218024
min         21.000000
25%         28.000000
50%         34.000000
75%         41.000000
max         79.000000
Name: Edad, dtype: float64
```

Como podemos observar la media es de 35.4 años lo cual nos parece es un buen valor de remplazo, solo sera convertido a el entero mas cercano, en este caso 35

[24]:
```
# Calculamos la Media,
media = int(dfLimpio.Edad.mean())
print('La Media es: ', media)
# Verificamos el numero de valores antes de la limpieza en cada uno
print('Numero de Registros en Edad Media:\n',dfLimpio['Edad'].value_counts()[35.
  ↪0])
print('Numero de Nulos en Edad:',nulldf.loc[5,'False'])
```

```
La Media es:  35
Numero de Registros en Edad Media:
 1113
Numero de Nulos en Edad: 5
```

[25]:
```
# Remplazamos la Edad en los 5 registros por la media de las edades
dfLimpio.Edad.fillna(media, inplace=True)


# Verificamos que los cambios se hayan efectuado.
print("HAY VALORES NULOS EN EDAD?: ", dfLimpio['Edad'].isnull().values.any())

# Verificamos que el numero de Registros haya incrementado en la moda
print('\nNumero de Registros en Edad Media:\n',dfLimpio['Edad'].
  ↪value_counts()[35.0])
```

```
HAY VALORES NULOS EN EDAD?:  False

Numero de Registros en Edad Media:
 1118
```

Observamos tambien que este dato indica un numero entero, y su tipo actual es flotante. Por lo que consideramos necesario convertir a entero.

[26]:
```
print('Tipo de dato en Edad Original: ', dfLimpio.Edad.dtype)
dfLimpio.Edad = dfLimpio.Edad.astype(int)
print('Nuevo tipo de dato en Edad: ', dfLimpio.Edad.dtype)
```

```
Tipo de dato en Edad Original:  float64
Nuevo tipo de dato en Edad:  int64
```

1.4 Para las columnas 6-11 que indican el historial de pagos, hablos de una variable categorica con los siguientes valores esperados:

-1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.

Lo primero a realizar es identificar la estrategia para los datos que no se cuente con informacion.

[27]:
```
# Obtenemos del data frame de datos faltantes cuantos hay en estas columnas.
nulldf[6:12]
```

[27]:

	ColumnName	True	False
6	PPSep2005	29997	3
7	PPAgo2005	29995	5
8	PPJul2005	29993	7
9	PPJun2005	29991	9
10	PPMay2005	29984	16
11	PPAbr2005	29986	14

Como podemos observar son de las columnas con mas datos faltantes, a su vez son columnas relacionadas ya que indican pagos en diferentes meses del añ 2005. Por lo que la estrategia a elegir debe ser igual para todas para no causar sesgos.

[28]:
```
# PRimero Obtenemos la descripcion estadistica de todas los datos
dfLimpio.iloc[:, 6:12].describe()
```

[28]:

	PPSep2005	PPAgo2005	PPJul2005	PPJun2005	PPMay2005	PPAbr2005
count	29997.000000	29995.000000	29993.000000	29991.000000	29984.000000	29986.000000
mean	-0.016635	-0.133689	-0.166405	-0.220800	-0.266342	-0.291136
std	1.123829	1.197254	1.196048	1.169153	1.133296	1.150134
min	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000
25%	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000

[29]:
```
# Tambien obtenemos la describcion de sus valores. Utilizando la funcion apply␣
  ↪ejecutamos el metodo value counts a cada una de las columnas del dataframe
print('\nNumero de Registros:\n',dfLimpio.iloc[:, 6:12].apply(pd.value_counts))
```

```
Numero de Registros:
       PPSep2005  PPAgo2005  PPJul2005  PPJun2005  PPMay2005  PPAbr2005
-2.0        2759       3782       4085       4348     4546.0     4895.0
-1.0        5684       6047       5935       5685     5535.0     5735.0
 0.0       14736      15728      15761      16450    16937.0    16278.0
```

```
 1.0       3688         28          4          2        NaN        NaN
 2.0       2667       3927       3819       3157     2624.0     2765.0
 3.0        322        326        240        180      178.0      184.0
 4.0         76         99         76         69       84.0       49.0
 5.0         26         25         21         35       17.0       13.0
 6.0         11         12         23          5        4.0       19.0
 7.0          9         20         27         58       58.0       46.0
 8.0         19          1          2          2        1.0        2.0
```

Como podemos observar, los valores no corresponden a la describcion del problema, ya que el valor de -1 es el mas pequeño pero en los registros es el -2. Por lo que necesitariamos preguntar la descripcion de esta data. Sin embargo creemos que la falta de esta informacion es vital ya que involucra el historial de pagos y en estas se basarian las predicciones de incumplimiento de pago.

Para este caso estaremos usando la moda de nuevo. Ya que esta es una variable categorica. Por lo que podemos observar tambien en la tabla anterior, la moda de este conjunto de datos es 0 para todos los casos.

[30]:
```
dfLimpio.iloc[:, 6:12].mode()
```

[30]:

	PPSep2005	PPAgo2005	PPJul2005	PPJun2005	PPMay2005	PPAbr2005
0	0.0	0.0	0.0	0.0	0.0	0.0

[31]:
```
print('\nNumero de Registros con valor 0:\n',dfLimpio.iloc[:, 6:12].apply(pd.
  ↪value_counts).iloc[2])
# Verificamos el numero de valores antes de la limpieza en cada uno
print('\nNumero de Nulos en Pagos Pasados: \n',nulldf.loc[6:12,'False'])
```

```
Numero de Registros con valor 0:
 PPSep2005    14736.0
PPAgo2005    15728.0
PPJul2005    15761.0
PPJun2005    16450.0
PPMay2005    16937.0
PPAbr2005    16278.0
Name: 0.0, dtype: float64

Numero de Nulos en Pagos Pasados:
 6      3
7       5
8       7
9       9
10     16
11     14
12     11
Name: False, dtype: int64
```

[32]:
```
# Remplazamos con la moda en todos los registros de todas las columnas
dfLimpio.PPSep2005.fillna(0.0, inplace=True)
dfLimpio.PPAgo2005.fillna(0.0, inplace=True)
dfLimpio.PPJul2005.fillna(0.0, inplace=True)
dfLimpio.PPJun2005.fillna(0.0, inplace=True)
dfLimpio.PPMay2005.fillna(0.0, inplace=True)
dfLimpio.PPAbr2005.fillna(0.0, inplace=True)
# Verificamos que los cambios se hayan efectuado.
print("HAY VALORES NULOS EN Pagos Pasados?: ", dfLimpio.iloc[:, 6:12].isnull().
 ↪values.any())

# Verificamos que el numero de Registros haya incrementado en la moda
print('\nNumero de Registros con valor 0:\n',dfLimpio.iloc[:, 6:12].apply(pd.
 ↪value_counts).iloc[2])
```

```
HAY VALORES NULOS EN Pagos Pasados?:  False

Numero de Registros con valor 0:
 PPSep2005    14739.0
PPAgo2005     15733.0
PPJul2005     15768.0
PPJun2005     16459.0
PPMay2005     16953.0
PPAbr2005     16292.0
Name: 0.0, dtype: float64
```

2 Para finalizar, las Columnas 12-23.

Todas estas son variables con datos numericos que indican cantidades en USD. Asi que lo primero que hay que hacer es analizar las medidas estadisticas.

[33]:
```
# Obtenemos del data frame de datos faltantes cuantos hay en estas columnas.
nulldf[12:24]
```

[33]:

	ColumnName	True	False
12	TRSep2005	29989	11
13	TRAgo2005	29989	11
14	TRJul2005	29987	13
15	TRJun2005	29985	15
16	TRMay2005	29983	17
17	TRAbr2005	29990	10
18	TPPSep2005	29992	8
19	TPPAgo2005	29991	9
20	TPPJul2005	29992	8
21	TPPJun2005	29989	11
22	TPPMay2005	29989	11
23	TPPAbr2005	29995	5

[34]:
```
# PRimero Obtenemos la descripcion estadistica de todas los datos
dfLimpio.iloc[:, 12:24].describe()
```

[34]:

	TRSep2005	TRAgo2005	TRJul2005	TRJun2005	\
count	29989.000000	29989.000000	2.998700e+04	29985.000000	
mean	51236.862750	49190.734669	4.702535e+04	43275.652326	
std	73645.219278	71183.385123	6.936086e+04	64345.500073	
min	-165580.000000	-69777.000000	-1.572640e+05	-170000.000000	
25%	3565.000000	2986.000000	2.667500e+03	2329.000000	
50%	22387.000000	21207.000000	2.008900e+04	19052.000000	
75%	67139.000000	64027.000000	6.018200e+04	54560.000000	
max	964511.000000	983931.000000	1.664089e+06	891586.000000	

	TRMay2005	TRAbr2005	TPPSep2005	TPPAgo2005	\
count	29983.000000	29990.000000	29992.000000	2.999100e+04	
mean	40324.493980	38881.135745	5662.945886	5.922489e+03	
std	60809.984983	59561.312967	16564.165089	2.304418e+04	
min	-81334.000000	-339603.000000	0.000000	0.000000e+00	
25%	1763.500000	1256.250000	1000.000000	8.355000e+02	
50%	18107.000000	17081.000000	2100.000000	2.009000e+03	
75%	50213.000000	49208.250000	5006.000000	5.000000e+03	
max	927171.000000	961664.000000	873552.000000	1.684259e+06	

	TPPJul2005	TPPJun2005	TPPMay2005	TPPAbr2005
count	29992.000000	29989.000000	29989.000000	29995.000000
mean	5225.623400	4827.252526	4800.297209	5216.259977
std	17608.422625	15668.751975	15280.842069	17778.848359
min	0.000000	0.000000	0.000000	0.000000
25%	390.000000	296.000000	251.000000	118.000000
50%	1800.000000	1500.000000	1500.000000	1500.000000
75%	4505.500000	4014.000000	4033.000000	4000.000000
max	896040.000000	621000.000000	426529.000000	528666.000000

Para estos casos particulares, el rango de valores sera demasiado grande. Por lo que, no es conveniente imprimir todos los valores de todos los elementos. Nuestro siguiente paso fue decidir por la mediana, esto debido a ser datos de dinero. Ya que una media podria resultar muy alta si el sesgo es positivo o muy baja si es negativo.

[35]:
```
# Construimos un DataFrame para mostrar las diferencias entre cada uno de los␣
 ↪valores
aux = pd.DataFrame({
    'ColumnName':dfLimpio.iloc[:, 12:24].columns.values,
    'Media': dfLimpio.iloc[:, 12:24].mean().values,
    'Mediana': dfLimpio.iloc[:, 12:24].median().values,
    'Moda': dfLimpio.iloc[:, 12:24].mode().values[0]
})
aux
```

[35]:

	ColumnName	Media	Mediana	Moda
0	TRSep2005	51236.862750	22387.0	0.0
1	TRAgo2005	49190.734669	21207.0	0.0
2	TRJul2005	47025.350152	20089.0	0.0
3	TRJun2005	43275.652326	19052.0	0.0
4	TRMay2005	40324.493980	18107.0	0.0
5	TRAbr2005	38881.135745	17081.0	0.0
6	TPPSep2005	5662.945886	2100.0	0.0
7	TPPAgo2005	5922.488913	2009.0	0.0
8	TPPJul2005	5225.623400	1800.0	0.0
9	TPPJun2005	4827.252526	1500.0	0.0
10	TPPMay2005	4800.297209	1500.0	0.0
11	TPPAbr2005	5216.259977	1500.0	0.0

[36]:
```
# Remplazamos con la mediana en todos los registros de todas las columnas
columnas = aux.ColumnName.to_list()
mediana = aux.Mediana

for idx, columna in enumerate(columnas):
    dfLimpio[columna].fillna(mediana[idx], inplace=True)

# Verificamos que los cambios se hayan efectuado.
print("HAY VALORES NULOS EN Total del Recibo y Total de Pagos Pasados: ",␣
 ↪dfLimpio.iloc[:, 6:12].isnull().values.any())
```

```
HAY VALORES NULOS EN Total del Recibo y Total de Pagos Pasados:  False
```

3 Finalmente validaremos si queda algun valor Nulo en todo el conjunto de Datos

[37]:
```
print("HAY VALORES NULOS EN EL CONJUNTO DE DATOS: ", dfLimpio.isnull().values.
 ↪any())
```

```
HAY VALORES NULOS EN EL CONJUNTO DE DATOS:  True
```

[38]:
```
dfLimpio.isnull().any()
```

[38]:
```
ID              False
Total_Credito   False
Sexo            False
Estudios        False
Estado_Civil    False
Edad            False
PPSep2005       False
PPAgo2005       False
PPJul2005       False
```

```
PPJun2005        False
PPMay2005        False
PPAbr2005        False
TRSep2005        False
TRAgo2005        False
TRJul2005        False
TRJun2005        False
TRMay2005        False
TRAbr2005        False
TPPSep2005       False
TPPAgo2005       False
TPPJul2005       False
TPPJun2005       False
TPPMay2005       False
TPPAbr2005       False
Y                 True
dtype: bool
```

Como podemos observar, el conjunto de datos esta practicamente limpio. Solo quedan registros vacios en la variable calculada Y. Al nosotros no saber con exactitud el modelo usado para calcularlo. Hemos decidido eliminar las filas que contengan Nulo en ese valor.

[39]:
```
# Verificamos el numero de valores antes eliminarlos
print('Numero de Registros en Y\n',dfLimpio['Y'].value_counts())
print('Numero de Nulos en Y:',nulldf.loc[24, 'False'])
print('Forma de Conjunto de Datos: ', dfLimpio.shape)
```

```
Numero de Registros en Y
 0.0    23362
1.0     6635
Name: Y, dtype: int64
Numero de Nulos en Y: 3
Forma de Conjunto de Datos:  (30000, 25)
```

[40]:
```
dfLimpio.dropna(inplace=True) #Eliminamos los valores nulos restantes, aquellos␣
 ↪que no nos conviene calcular

# Validamos las modificaciones hechas.
print("HAY VALORES NULOS EN EL CONJUNTO DE DATOS: ", dfLimpio.isnull().values.
 ↪any())
print('Forma de Conjunto de Datos: ', dfLimpio.shape)
```

```
HAY VALORES NULOS EN EL CONJUNTO DE DATOS:  False
Forma de Conjunto de Datos:  (29997, 25)
```

Parte 3: Preparación de los datos

Con base en los resultados de tu libreta de Google Colab de la Parte 2 responde detalladamente las siguientes preguntas:

¿Qué datos consideraron más importantes? ¿Por qué?
Para nosotros, todos los datos del conjunto de datos son sumamente importantes para un modelo como el descrito en la definición del conjunto de datos. Los campos de identificación (edad, sexo, estado civil, escolaridad y edad) nos pueden brindar la perspectiva socioeconómica de los datos, nos sirven para analizar las situaciones por las que los sujetos de estudio atraviesan y las relaciones que tienen con los datos de su historial crediticio que vienen en las siguientes columnas. Es por esto que creemos que los primeros 5 campos se relacionan íntimamente con el resto de las columnas con información más detallada del historial crediticio y un analista de datos podría encontrar las relaciones necesarias para realizar buenas predicciones.

¿Se eliminaron o reemplazaron datos nulos? ¿Qué se hizo y por qué?
Si, decidimos remplazar datos nulos en la mayoría de los campos. En primer lugar, porque para algunos el número de campos faltantes era demasiado pequeño 1-5 y en segundo lugar creímos que contábamos con la suficiente información en las columnas para poder calcular alguna de las variables de remplazo. Ya fuera la media, la mediana o la moda, las cuales nos sirvieron para remplazar los valores faltantes.

¿Es necesario limpiar los datos para el análisis? Sí / No / ¿Por qué?
Creemos que para este caso SI es necesario limpiar un poco los datos para poder procesarlos y generar un mejor análisis. Mediante nuestro análisis logramos identificar campos que no se adaptaban a el modelo descrito por lo que decidimos transformar dichos campos para que pudiera ser utilizado, en especial en un par de campos que contaban con categorías como Genero y Grado de estudios. Creemos que, sin estas transformaciones y llenado de datos nulos, el análisis podría carecer de precisión.

¿Existen problemas de formato que deban solucionar antes del proceso de modelado? Sí / No / Por qué.
En nuestro análisis, SI encontramos problemas de formato de datos. En especial porque la definición proporcionada mencionaba que los campos debían de tener ciertos valores categóricos que no eran respetados. Así mismo creímos conveniente transformar el campo de edad a entero ya que
1.- No parecía haber indicios de edades flotantes.
2.- Porque no parece ser relevante para los modelos un cambio en el tipo de dato.

¿Qué ajustes se realizaron en el proceso de limpieza de datos (agregar, integrar, eliminar, modificar registros (filas), cambiar atributos (columnas)?
En el ejercicio realizamos las siguientes acciones:

- Agregar datos faltantes o valores nulos en todas las columnas excepto la última (Y) ya que consideramos que era un variable calculada por el modelo usado en el ejercicio.
- Integrar valores de variables categóricas. Unificando los valores diferentes a los establecidos en la definición del problema.
- Eliminamos 3 registros (filas) que no contaban con valores en la columna Y. Debido a que no nos parecía una buena práctica usar algún método de sustitución como la media, moda o mediana como en otras columnas.
- Cambiamos atributos debido a que consideramos que el tipo de dato no era el adecuado para realizar un análisis y podría causar confusión o entradas de datos de tipo erróneos

Bibliografía

- Cavell-Clarke, S. (2018). What is a database? BookLife Publishing.
- De los datos, E. M. (2021, enero 16). Técnicas para codificar las variables categóricas (I): codificación ordinal y one-hot. El mundo de los datos. https://elmundodelosdatos.com/tecnicas-para-codificar-variables-categoricas-ordinal-one-hot/
- CSB, & SJU Libraries. (2020). Research guides: General library research tutorials: Module 4: Searching a database. https://guides.csbsju.edu/general-research/searching
- Kim, K. (s/f). What is a database? Definition, types and examples. Fivetran.com. Recuperado el 4 de octubre de 2022, de https://www.fivetran.com/blog/what-is-a-database
- ¿Qué es un almacén de datos? (s/f). Oracle.com. Recuperado el 4 de octubre de 2022, de https://www.oracle.com/mx/database/what-is-a-data-warehouse/
- Reed, N. (2021, enero 19). The history of databases. ThinkAutomation. https://www.thinkautomation.com/histories/the-history-of-databases/