PREPARACIÓN DE DATOS

La preparación de datos en el aprendizaje automático (machine learning) se refiere al proceso de limpiar, transformar y organizar los datos antes de utilizarlos para entrenar un modelo de aprendizaje automático. Es una etapa crítica en el flujo de trabajo de machine learning, ya que la calidad de los datos de entrada puede tener un impacto significativo en el rendimiento del modelo.

La preparación de datos implica varias tareas, que incluyen:

- Limpieza de datos: Esto implica identificar y corregir errores, valores atípicos o faltantes en los datos. Esto puede incluir el manejo de datos duplicados, la imputación de valores faltantes o la eliminación de registros inconsistentes.

- Transformación de datos: A veces, los datos en bruto pueden no ser directamente útiles para el modelo de aprendizaje automático. En este paso, se pueden realizar transformaciones en los datos, como normalización, discretización, codificación de variables categóricas, extracción de características relevantes o reducción de la dimensionalidad.

- Normalización y escalado: En muchos algoritmos de aprendizaje automático, es importante escalar las características numéricas para que tengan una escala similar. Esto evita que las características con valores más grandes dominen sobre las características con valores más pequeños.

- Selección de características: En algunos casos, es posible que los datos contengan características irrelevantes o redundantes que no aportan información útil al modelo. La selección de características implica identificar y seleccionar las características más relevantes para mejorar la precisión y la eficiencia del modelo.

La preparación de datos puede ser un proceso iterativo, donde se realizan ajustes y mejoras en función de los resultados del modelo. Un buen procesamiento de datos puede ayudar a mejorar la precisión, la eficiencia y la capacidad de generalización de los modelos de aprendizaje automático.

1. PASOS PREVIOS

Importar librerías

```
1 # pandas: manejo de dataFrame
2 import pandas as pd
3 # sklearn: scikit-learn - Librería de python para Machine Learning
```

```
4 # Normalizar: permite escalar los datos a un rango de [0...1]
5 from sklearn.preprocessing import MinMaxScaler
```

Importación de datos **normalización.csv**

```
1 datos = pd.read_csv("normalizacion.csv")
2 datos.head(10)
```

	MONTO.CREDITO	SUELDO	SEXO	ESTADO.CIVIL	DEUDA	CAPITAL.BIENES	GARANTE	DISTRIT
0	30000.0	>= 5000	F	Divorciado	<= 1000	4050.0	No	Sa Sebastia
1	100000.0	>= 5000	F	Viudo	> 1000 Y < 5000	2590.0	No	Wancha
2	30000.0	>= 5000	M	Casado	>= 5000	6507.0	No	Sa Jeronim
3	70000.0	>= 5000	M	Viudo	<= 1000	2587.0	No	Wancha
4	10000.0	>= 500 y < 2000	F	Divorciado	<= 1000	NaN	Si	Wancha

2. ELIMINACIÓN DE DATOS FALTANTES

Determinar si existen datos faltantes

```
1 # visualizar información del dataset
2 datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49 entries, 0 to 48
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   MONTO.CREDITO   48 non-null     float64
 1   SUELDO          46 non-null     object 
 2   SEXO            49 non-null     object 
 3   ESTADO.CIVIL    48 non-null     object 
 4   DEUDA           48 non-null     object 
 5   CAPITAL.BIENES  46 non-null     float64
 6   GARANTE         49 non-null     object 
 7   DISTRITO        47 non-null     object 
```

```
 8   EDAD            47 non-null     float64
dtypes: float64(3), object(6)
memory usage: 3.6+ KB
```

```
1 #número de valores nulos por cada atributo
2 datos.isnull().sum()
```

```
MONTO.CREDITO     1
SUELDO            3
SEXO              0
ESTADO.CIVIL      1
DEUDA             1
CAPITAL.BIENES    3
GARANTE           0
DISTRITO          2
EDAD              2
dtype: int64
```

```
1 #visualizar datos que tengan algun atributo nulo
2 nulos = datos[datos.isnull().any(1)]
3 nulos
```

```
<ipython-input-105-eec800afb68c>:2: FutureWarning: In a future version of pandas all
  nulos = datos[datos.isnull().any(1)]
```

	MONTO.CREDITO	SUELDO	SEXO	ESTADO.CIVIL	DEUDA	CAPITAL.BIENES	GARAN
4	10000.0	>= 500 y < 2000	F	Divorciado	<= 1000	NaN	
5	10000.0	NaN	F	Divorciado	>= 5000	14965.0	
7	70000.0	< 500	F	Casado	>= 5000	6785.0	
9	70000.0	< 500	F	NaN	>= 5000	5050.0	
11	10000.0	>= 5000	M	Casado	<= 1000	NaN	
22	70000.0	NaN	F	Viudo	>= 5000	7003.0	
26	50000.0	>= 2000 y < 5000	M	Casado	>= 5000	10877.0	
28	30000.0	< 500	M	Casado	NaN	3287.0	
36	NaN	>= 2000 y < 5000	M	Divorciado	<= 1000	2859.0	
40	90000.0	NaN	M	Casado	<= 1000	NaN	

```
1 print ('número de registros con valores nulos: ', len(nulos))
```

```
número de registros con valores nulos:  10
```

Eliminar de valores nulos

```
1 # Elimina filas (datos) con valores faltantes 'NaN'
2 # axis = 0 indica fila, axis=1 indica la columna
3 datos = datos.dropna(axis=0)
4 datos.head()
```

	MONTO.CREDITO	SUELDO	SEXO	ESTADO.CIVIL	DEUDA	CAPITAL.BIENES	GARANTE	DISTRIT
0	30000.0	>= 5000	F	Divorciado	<= 1000	4050.0	No	Sa Sebastia
1	100000.0	>= 5000	F	Viudo	> 1000 Y < 5000	2590.0	No	Wancha

```
1 # comprobar si existen nulos
2 datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 39 entries, 0 to 48
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   MONTO.CREDITO   39 non-null     float64
 1   SUELDO          39 non-null     object 
 2   SEXO            39 non-null     object 
 3   ESTADO.CIVIL    39 non-null     object 
 4   DEUDA           39 non-null     object 
 5   CAPITAL.BIENES  39 non-null     float64
 6   GARANTE         39 non-null     object 
 7   DISTRITO        39 non-null     object 
 8   EDAD            39 non-null     float64
dtypes: float64(3), object(6)
memory usage: 3.0+ KB
```

EJERCICIO: En lugar de eliminar los registros con valores nulos, experimente con la técnica de **imputación de datos faltantes** considerando la media, mediana y moda **según sea aplicable**. Vuelva a cargar los datos originales, realice la imputación pertinente y muestre el resultado en diferentes datasets.

```
1 # carga de datos originales
2 datos2 = pd.read_csv("normalizacion.csv")
3 datos2.head(10)
```

```
1 # imputación de atributos utilizando la media
2
```

```
1 # imputación de atributos utilizando la moda
2
```

```
1 # imputación de atributos utilizando la mediana
2
```

```
1 # visualización del dataset luego del proceso de imputación
2
```

2. CONVERSION DE DATOS CATEGÓRICOS A NUMÉRICOS

Revisión del atributo SUELDO

```
1 # vista del dataset
2 datos.head(10)
```

	MONTO.CREDITO	SUELDO	SEXO	ESTADO.CIVIL	DEUDA	CAPITAL.BIENES
0	30000.0	>= 5000	F	Divorciado	<= 1000	4050.0
1	100000.0	>= 5000	F	Viudo	> 1000 Y < 5000	2590.0
2	30000.0	>= 5000	M	Casado	>= 5000	6507.0
3	70000.0	>= 5000	M	Viudo	<= 1000	2587.0
6	50000.0	>= 2000 y < 5000	M	Viudo	> 1000 Y < 5000	8724.0
8	30000.0	>= 5000	M	Casado	> 1000 Y < 5000	4860.0
10	30000.0	>= 500 y < 2000	F	Casado	<= 1000	4304.0
12	50000.0	< 500	M	Soltero	<= 1000	5318.0
13	50000.0	>= 5000	M	Casado	>= 5000	8887.0
14	100000.0	>= 500 y < 2000	M	Divorciado	>= 5000	2920.0

```
1 # mostrar los valores existentes en la columna SUELDO
2 datos.SUELDO.unique()
```

```
array(['>= 5000', '>= 2000 y < 5000', '>= 500 y < 2000', '< 500'],
      dtype=object)
```

```
1 # número de valores de dominio de SUELDO
2 datos.SUELDO.value_counts()
```

```
>= 5000             13
< 500               10
>= 2000 y < 5000     8
>= 500 y < 2000      8
Name: SUELDO, dtype: int64
```

```
1 # ¿Cómo se codificaría automáticamente el atributo sueldo?
2 datos.SUELDO.astype("category").cat.codes
```

```
0     3
1     3
2     3
3     3
6     1
8     3
10    2
12    0
13    3
14    2
15    0
16    0
17    1
18    1
19    2
20    3
21    0
23    1
24    3
25    3
27    2
29    1
30    3
31    0
32    0
33    2
34    2
35    0
37    3
38    1
39    1
41    0
42    2
43    2
44    3
45    3
46    1
47    0
48    0
dtype: int8
```

Implementación de un módulo que convierte un atributo categórico a numérico utilizando las funciones **astype("category")** y **cat.codes**

```
1 def Categorico_a_numerico(atributo):
2     # Convertir el atributo al tipo categórico
3     atributo = atributo.astype("category")
4     # Convertir el atributo categórico a numérico
5     return atributo.astype("category").cat.codes
```

Convertir los atributos SUELDO, SEXO, ESTADO.CIVIL, DEUDA, GARANTE DISTRITO a numérico

```
1 # datos antes de la conversión
2 datos.head()
```

	MONTO.CREDITO	SUELDO	SEXO	ESTADO.CIVIL	DEUDA	CAPITAL.BIENES
0	30000.0	>= 5000	F	Divorciado	<= 1000	4050.0
1	100000.0	>= 5000	F	Viudo	> 1000 Y < 5000	2590.0
2	30000.0	>= 5000	M	Casado	>= 5000	6507.0
3	70000.0	>= 5000	M	Viudo	<= 1000	2587.0
6	50000.0	>= 2000 y < 5000	M	Viudo	> 1000 Y < 5000	8724.0

```
1 # convertir los categóricos a numéricos
2 datos['SUELDO'] = Categorico_a_numerico(datos['SUELDO'])
3 datos['SEXO'] = Categorico_a_numerico(datos['SEXO'])
4 datos['ESTADO.CIVIL'] = Categorico_a_numerico(datos['ESTADO.CIVIL'])
5 datos['DEUDA'] = Categorico_a_numerico(datos['DEUDA'])
6 datos['GARANTE'] = Categorico_a_numerico(datos['GARANTE'])
7 datos['DISTRITO'] = Categorico_a_numerico(datos['DISTRITO'])
8
```

```
1 # datos después de la conversión
2 datos.head()
```

	MONTO.CREDITO	SUELDO	SEXO	ESTADO.CIVIL	DEUDA	CAPITAL.BIENES	GARANTE	DISTRIT
0	30000.0	3	0	1	0	4050.0	0	
1	100000.0	3	0	3	1	2590.0	0	
2	30000.0	3	1	0	2	6507.0	0	
3	70000.0	3	1	3	0	2587.0	0	
6	50000.0	1	1	3	1	8724.0	0	

3. NORMALIZACIÓN DE DATOS

Implementar la normalización de datos por amplitud y distribución

```
# Transformación por amplitud [0..1]
def Normalizacion_Amplitud(atributo):
    return (atributo - atributo.min())/(atributo.max() - atributo.min())
```

```
# Transformación por distribución [-1..1]
def Normalizacion_Distribucion(atributo):
    return (atributo - atributo.mean())/(atributo.var())
```

Normalizar todos los atributos por amplitud

```
datos.columns
```

```
Index(['MONTO.CREDITO', 'SUELDO', 'SEXO', 'ESTADO.CIVIL', 'DEUDA',
       'CAPITAL.BIENES', 'GARANTE', 'DISTRITO', 'EDAD'],
      dtype='object')
```

```
datos['MONTO.CREDITO'] = Normalizacion_Amplitud(datos['MONTO.CREDITO'])
datos['SUELDO'] = Normalizacion_Amplitud(datos['SUELDO'])
datos['SEXO'] = Normalizacion_Amplitud(datos['SEXO'])
datos['ESTADO.CIVIL'] = Normalizacion_Amplitud(datos['ESTADO.CIVIL'])
datos['DEUDA'] = Normalizacion_Amplitud(datos['DEUDA'])
datos['CAPITAL.BIENES'] = Normalizacion_Amplitud(datos['CAPITAL.BIENES'])
datos['GARANTE'] = Normalizacion_Amplitud(datos['GARANTE'])
datos['DISTRITO'] = Normalizacion_Amplitud(datos['DISTRITO'])
datos['EDAD'] = Normalizacion_Amplitud(datos['EDAD'])
```

```
#datos después de la normalizacion por amplitud
datos.head()
```

	MONTO.CREDITO	SUELDO	SEXO	ESTADO.CIVIL	DEUDA	CAPITAL.BIENES	GARANTE	DISTR
0	0.222222	1.000000	0.0	0.333333	0.0	0.188130	0.0	0.500
1	1.000000	1.000000	0.0	1.000000	0.5	0.078306	0.0	1.000
2	0.222222	1.000000	1.0	0.000000	1.0	0.372950	0.0	0.333
3	0.666667	1.000000	1.0	1.000000	0.0	0.078080	0.0	1.000
6	0.444444	0.333333	1.0	1.000000	0.5	0.539717	0.0	0.166

4. GRABACIÓN DE DATOS PRE-PROCESADOS

```
# guardar el preprocesamiento de datos en un archivo csv
datos.to_csv("datos_norm.csv", sep=',', header=True, index=False)
```

5. TAREA

Normalizar los siguientes dataset:

> diabetes.csv

> riesgo_credito.csv