

# Introducción a la Inteligencia Artificial

## Pre-procesamiento - Perspectiva Práctica

## Outline

### 1 Pre-procesamiento

- Acceso a Datos
- Análisis exploratorio de datos
- Tareas de pre-procesamiento

## Pre-procesamiento

Generalmente, los **datos** crudos **no son adecuados** para ser usados directamente y aplicarles una técnica de IA.

- Los datos pueden presentar diversos problemas:
  - Ruido
  - Registros duplicados
  - Datos incompletos o inconsistentes
  - Formato inadecuado
  - Grandes volúmenes

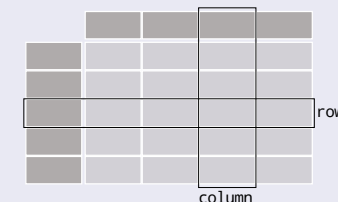
### Definición

El **pre-procesamiento** consiste en la aplicación de técnicas con el objetivo de **adequar los datos** para ser utilizados.

## Pandas

Pandas es una herramienta de análisis y manipulación de datos de código abierto rápida, potente, flexible y fácil de usar.

DataFrame



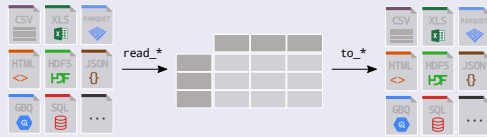
### Definición

El **DataFrame** es una estructura de datos tabular que se compone de columnas y filas ordenadas. Se pueden realizar operaciones básicas en filas / columnas como seleccionar, eliminar, agregar y renombrar.

## Lectura / Escritura de Datos

## Accediendo a datos en Google Drive

Pandas admite la integración con muchos formatos de archivo o fuentes de datos listos para usar (csv, excel, sql, json, parquet, ...)



## Ejemplo

```
import pandas as pd
iris = pd.read_csv("data/iris.csv")
iris.to_excel('iris.xlsx', sheet_name='iris', index=False)
```

## Accediendo a datos en OpenML

OpenML es un sitio donde se pueden compartir conjuntos de datos y soluciones con otras personas, ahorrando un tiempo valioso, aumentando su visibilidad y acelerando el descubrimiento.

## Instalar e importar bibliotecas de OpenML

```
!pip install openml
import openml
```

## Listar datasets de OpenML

```
datasets_df = openml.datasets.list_datasets(output_format='dataframe')
print(datasets_df.head(n=10))
```

## Accediendo a datos en OpenML

OpenML vincula datos a algoritmos y personas permitiendo desarrollar sobre el estado del arte.

## Listar datasets de OpenML

```
datasets_df = openml.datasets.list_datasets(output_format='dataframe')
print(datasets_df.head(n=10))
```

	did	name	...	NumberOfNumericFeatures	NumberOfSymbolicFeatures
2	2	anneal	...	6.0	33.0
3	3	kr-vs-kp	...	0.0	37.0
4	4	labor	...	8.0	9.0
5	5	arrhythmia	...	206.0	74.0
6	6	letter	...	16.0	1.0
7	7	audiology	...	0.0	70.0
8	8	liver-disorders	...	6.0	0.0
9	9	autos	...	15.0	11.0
10	10	lymph	...	3.0	16.0
11	11	balance-scale	...	4.0	1.0

[10 rows x 16 columns]

## Accediendo a datos en OpenML

## Descargar un dataset

```
# Iris dataset https://www.openml.org/d/61
dataset = openml.datasets.get_dataset(61)
```

## Obtener datos del dataset

```
import pandas as pd
# X - An array/dataframe where each row represents one
#       example with the corresponding feature values.
# y - the classes for each example
# categorical_indicator - an array that indicates which
#       feature is categorical
# attribute_names - the names of the features for the
#       examples (X) and target feature (y)
X, y, categorical_indicator, attribute_names = dataset.
get_data( dataset_format='dataframe',
target=dataset.default_target_attribute )
```



diplomatura universitaria en  
inteligencia artificial



## Outline

## 1 Pre-procesamiento

- Acceso a Datos
- Análisis exploratorio de datos
- Tareas de pre-procesamiento

## Análisis exploratorio de datos

Es necesaria la exploración de los datos mediante técnicas de análisis exploratorio para identificar valores inusuales, valores extremos, valores desaparecidos, discontinuidades u otras peculiaridades de los mismos.

- Usar valores estadísticos (media, desvío, valores extremos)
- Usar gráficos (histogramas)

## Estadísticas del conjunto de datos (Dataframe)

## Ejemplo

```
combined_data = pd.concat([X, y], axis=1)
combined_data.describe()
```

	sepalength	sepalwidth	petallength	petalwidth
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

## Visualizar el conjunto de datos

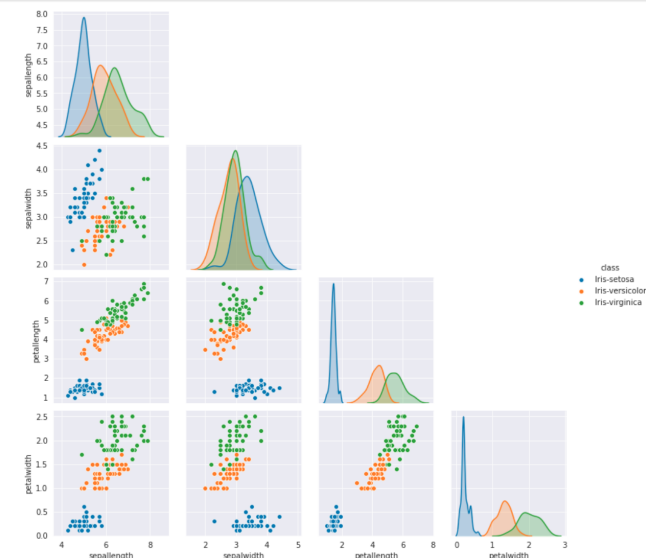
## Ejemplo

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style("darkgrid")

def hide_current_axis(*args, **kwargs):
    plt.gca().set_visible(False)

# We combine all the data so that we can map the different
# examples to different colors according to the classes.
combined_data = pd.concat([X, y], axis=1)
iris_plot = sns.pairplot(combined_data, hue="class")
iris_plot.map_upper(hide_current_axis)
plt.show()
```

## Visualizar el conjunto de datos



## Informe del conjunto de datos

**Pandas profiling** es un modulo que permite hacer facilmente un análisis exploratorio de los datos mediante estadísticas y representaciones gráficas con unas pocas líneas de código

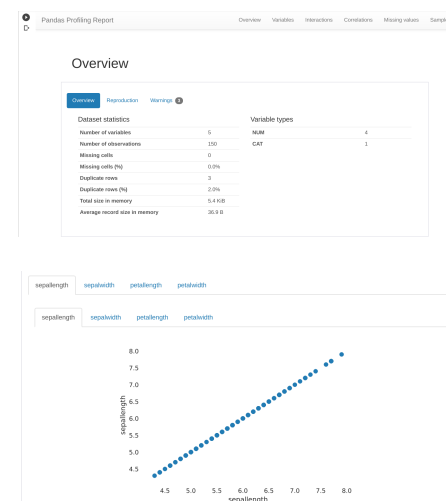
## Ejemplo

```
! pip install pandas==1.0.3
! pip install pandas_profiling==2.5.0

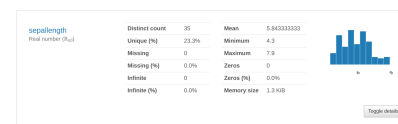
from pandas_profiling import ProfileReport
combined_data = pd.concat([X, y], axis=1)

%matplotlib inline
profile = ProfileReport(combined_data)
profile
```

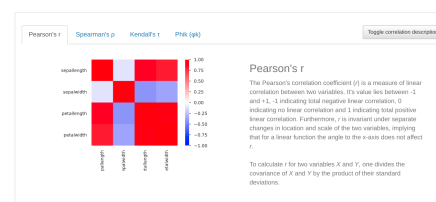
## Informe del conjunto de datos



## Variables



## Correlations



## Outline

## Tratamiento de valores faltantes

El manejo de valores faltantes es una parte esencial durante proceso de limpieza y preparación de datos porque casi todos los datos en la vida real vienen con algunos valores faltantes.

## 1 Pre-procesamiento

- Acceso a Datos
- Análisis exploratorio de datos
- Tareas de pre-procesamiento

- np.nan (not a number), None, <NA> y NaT (not a time) son los valores faltantes por defecto en Pandas.
- Pandas proporciona funciones **isnull ()**, **isna ()** para detectar valores perdidos (Ambos hacen lo mismo).
- Si el conjunto de datos usa otro tipo de simbolos ('?' o '--') se puede remplazar usando:
  - **df.replace(['?', '--'], np.nan, inplace=True)**

## Tratamiento de valores faltantes (Pandas)

## Tratamiento de valores faltantes (Pandas)

- Se puede eliminar una fila o columna con valores faltantes usando la función **dropna ()**. (fila (0) or columna (1))
  - **df.dropna(axis=0, how='all', inplace=True)** #si todos los valores faltan
  - **df.dropna(axis=0, how='any', inplace=True)** #si alguno de los valores falta
- Se pueden reemplazar valores faltantes
  - Reemplazando con un escalar: **df.fillna(25)**
  - Reemplazando con la media: **df.fillna(df.mean(), inplace=True)**
  - Reemplazando con valores previos o posteriores: **df.fillna(method='ffill')**

## Ejemplo

```
combined_data = pd.concat([X, y], axis=1)
dfhead=dfhead.fillna(dfhead.mean())
```

	sepalength	sepalwidth	petallength	petalwidth	class		sepalength	sepalwidth	petallength	petalwidth	class
0	NaN	NaN	1.4	0.2	Iris-setosa	0	4.766667	3.1	1.4	0.2	Iris-setosa
1	NaN	NaN	1.4	0.2	Iris-setosa	1	4.766667	3.1	1.4	0.2	Iris-setosa
2	4.7	NaN	1.3	0.2	Iris-setosa	2	4.700000	3.1	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa	3	4.600000	3.1	1.5	0.2	Iris-setosa
4	5.0	NaN	1.4	NaN	Iris-setosa	4	5.000000	3.1	1.4	0.2	Iris-setosa

## Tratamiento de valores faltantes (Scikit-Learn)

La biblioteca Scikit-learn ofrece no solo una gran variedad de algoritmos de aprendizaje, sino también muchas funciones convenientes para preprocesar datos y ajustar y evaluar nuestros modelos.

## Ejemplo

```
from sklearn.impute import SimpleImputer
dfdrop=combined_data.drop('class',axis=1) #se elimina columna
"class" ya que la media solo se puede calcular en
atributos numericos
imputer = SimpleImputer(missing_values=np.nan, strategy='mean
')
imputer = imputer.fit(dfdrop)
imputed_data = pd.DataFrame(imputer.transform(dfdrop))
imputed_data.columns=X.columns
combined_data = pd.concat([imputed_data, y], axis=1)
```



## Atributos categóricos

La mayoría de los algoritmos de Machine Learning prefieren trabajar con números pero muchas veces nos encontramos con atributos categóricos.

- Ordinales: se pueden entender como valores categóricos que se pueden ordenar. Por ejemplo tamaños: grande > mediano > pequeño
- Nominales: no implican ningún orden. Por ejemplo color, ya que normalmente no tiene sentido decir que, por ejemplo, el rojo es mayor que el azul.

## Atributos categóricos

## Atributos categóricos

## Agregamos atributos categóricos con valores al azar

```
combined_data['size'] =
    np.random.choice(['small','medium', 'large'],
        combined_data.shape[0])
combined_data['color'] =
    np.random.choice(['red','green', 'blue'], combined_data.
        shape[0])
```

	sepalwidth	sepalwidth	petalwidth	petalwidth	class	size	color
0	NaN	NaN	1.4	0.2	Iris-setosa	small	red
1	NaN	NaN	1.4	0.2	Iris-setosa	medium	green
2	4.7	NaN	1.3	0.2	Iris-setosa	small	green
3	4.6	3.1	1.5	0.2	Iris-setosa	medium	red
4	5.0	NaN	1.4	NaN	Iris-setosa	medium	green
...	...	...	...	...	...	...	...

Para asegurarnos de que el algoritmo de aprendizaje interprete las características ordinales correctamente, necesitamos convertir los valores categóricos en enteros.

## Atributo ordinal «size»

```
size_mapping = {
    'large': 3,
    'medium': 2,
    'small': 1}
```

```
combined_data['size'] = combined_data['size'].map(
    size_mapping)
```



## Atributos nominales: one hot encoding (Pandas)

Se crea una nueva característica ficticia para cada valor único en la columna de características nominales. (col\_red, col\_blue, col\_green / un atributo igual a 1 cuando la categoría es rojo sino 0,...)

## Codificación atributo nominal «color»

```
combined_data=pd.get_dummies(combined_data, prefix=['col'],
                             columns=['color'])
```

	sepalength	sepalwidth	petallength	petalwidth	class	size	col_blue	col_green	col_red
0	5.1	3.5	1.4	0.2	Iris-setosa	3	1	0	0
1	4.9	3.0	1.4	0.2	Iris-setosa	1	1	0	0
2	4.7	3.2	1.3	0.2	Iris-setosa	3	1	0	0
3	4.6	3.1	1.5	0.2	Iris-setosa	1	0	1	0
4	5.0	3.6	1.4	0.2	Iris-setosa	2	0	0	1

## Atributos nominales: one hot encoding (Scikit-Learn)

Se crea una nueva característica ficticia para cada valor único en la columna de características nominales. (No utilizar LabelEncoder ya que los algoritmos de ML supondrán que dos valores cercanos son más similares que dos valores distantes)

## Codificación atributo nominal «color»

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
columnTransformer = ColumnTransformer([('encoder',
    OneHotEncoder(), [6])], remainder='passthrough')
dataset = pd.DataFrame (columnTransformer.fit_transform(
    combined_data))
dataset.columns= pd.Index(['col_blue', 'col_green', 'col_red'])
    .append(combined_data.columns.drop('color'))
```

## Etiquetas de clase (Scikit-Learn)

Muchas bibliotecas de aprendizaje automático requieren que las etiquetas de clase se codifiquen como valores enteros

## Codificación de etiquetas de clase

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
class_cat = combined_data["class"]
class_cat_encoded = encoder.fit_transform(class_cat)
combined_data["class"] = class_cat_encoded
print(class_cat_encoded)
print(encoder.classes_)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
2 2]  
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

## Transformación de datos: normalización min-max (Scikit-Learn)

Muchos algoritmos de aprendizaje automático funcionan mejor cuando las características están en una escala relativamente similar o cerca de una distribución normal.

## Escalamiento Min-Max de «sepallenght»

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer
columnTransformer = ColumnTransformer([('num', MinMaxScaler(),
    [1])], remainder='passthrough')
dataset = columnTransformer.fit_transform(combined_data)
```

## Transformación de datos de multiples atributos(Scikit-Learn)

Aplicar transformaciones de datos como escalar o codificar variables categóricas es sencillo cuando todas las variables de entrada son del mismo tipo. Puede ser un desafío cuando tiene un conjunto de datos con tipos mixtos y desea aplicar transformaciones de datos selectivamente a algunas.

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
t = [('cat', OneHotEncoder(), [6]), ('num', MinMaxScaler(), [
    0,1,2,3])]
columnTransformer = ColumnTransformer(transformers=t,
    remainder='passthrough')
dataset = columnTransformer.fit_transform(combined_data)
```

## Transformación de datos de multiples atributos en pipeline(Scikit-Learn)

```
from sklearn.pipeline import Pipeline

numeric_features = ['sepalwidth', 'sepalwidth', 'petalwidth', 'petalwidth']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', MinMaxScaler())])

categorical_features = ['color']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder())])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)]
    ,remainder='passthrough')

dataset = preprocessor.fit_transform(combined_data)
```

## Reducción de datos





Reducción de la cantidad de instancias de un dataset

```
from sklearn.model_selection import train_test_split

per=0.1

_, reduced_ds = train_test_split(combined_data, test_size=int
    (per * len(combined_data)), random_state=42, stratify=
    combined_data['class'])
```

## Referencias I

-  [Pandas documentation](#)
-  [Tutorial Python](#)
-  [Tutorial Python](#)
-  [Guia del Usuario Scikit-Learn](#)