Simpler ETL Tasks with Pandas: Useful Functions Applied to Public Data

Tareas ETL más Simples con Pandas: Funciones Útiles Aplicadas sobre Datos Públicos

» **Jorge Kamlofsky**
GIA: Grupo de Investigación en Inteligencia Artificial. Universidad Tecnológica Nacional, Facultad Regional Haedo
CAETI: Centro de Altos Estudios en Tecnología Informática. Universidad Abierta Interamericana, Facultad de Tecnología Informática

Fernando Manzano
GIA: Grupo de Investigación en Inteligencia Artificial. Universidad Tecnológica Nacional, Facultad Regional Haedo
Instituto de Geografía, Historia y Ciencias Sociales (CONICET)

Diego López Yse
GIA: Grupo de Investigación en Inteligencia Artificial. Universidad Tecnológica Nacional, Facultad Regional Haedo

Recibido: Septiembre 2024./ Aceptado: Noviembre 2024 / Publicado; Diciembre 2024.
Cómo citar Kamlofsky J, Manzano F, Lopez Yse D. Tareas ETL más Simples con Pandas: Funciones Útiles Aplicadas sobre Datos Públicos. Revista Abierta de Informática Aplicada [Internet]. 2024 Dec. 30 [cited 2025 Jan. 26];8(1):35-58. https://doi.org/10.59471/raia2024204

Abstract

Abstract Artificial Intelligence (AI according to its initials) is undoubtedly one of the most disruptive technologies today. As well as Python, it is the most widely used programming language for the development of AI models, based on the analysis of datasets. Data analytics in Python began a dizzying growth from the development of the Pandas library, which provides functionalities of simple implementation and great utility for the processing of raw data, this being, perhaps, the most laborious task in the modeling process. In this work, the main functionalities of Pandas applied to public data are presented with applied code examples.

KEYWORDS: artificial intelligence, Python, Pandas library, public data.

Resumen

La Inteligencia Artificial (IA según sus iniciales) es sin duda, una de las tecnologías más disruptiva en la actualidad. Así como también Python, es el lenguaje de programación más usado para

desarrollos de modelos de IA, basado en el análisis de conjuntos de datos. La analítica de datos en Python inicio un crecimiento vertiginoso a partir del desarrollo de la librería Pandas, que otorga funcionalidades de simple implementación y gran utilidad para el procesamiento de datos crudos, siendo esta, quizás, la tarea más laboriosa en el proceso de elaboración de modelos. En este trabajo, se presentan las principales funcionalidades de Pandas aplicadas sobre datos públicos con ejemplos de código aplicados.

PALABRAS CLAVE: inteligencia artificial, Python, libreria Pandas, datos públicos.

1. Introducción

1.1. Trabajos relacionados

Antes de la llegada de Pandas[1] en 2008, manipular datos en Python[2] era una tarea ardua y tediosa. Los programadores solían recurrir a estructuras estándar del lenguaje como colecciones, listas y diccionarios, o a la biblioteca NumPy para trabajar con datos estructurados (por ejemplo: vectores, matrices). Si bien NumPy[3] ofrecía eficientes arrays multi-dimensionales y funciones matemáticas avanzadas, carecía de las herramientas necesarias para la manipulación de datos de alto nivel, tales como la indexación flexible, el tratamiento de datos nulos y la manipulación de series temporales. Estos procesos requerían de una codificación compleja y manual, lo que incrementaba la probabilidad de errores, haciendo además que el análisis de datos fuera lento y propenso a fallos.

Pandas fue desarrollado por Wes McKinney. Su aporte facilitó la manipulación y el análisis de datos estructurados. La introducción de Pandas revolucionó el análisis de datos en Python. Mediante la incorporación de estructuras de datos como DataFrames y Series, brindó una forma más intuitiva y eficiente de gestionar datos. Los DataFrames, en particular, ofrecieron una estructura bidimensional similar a una planilla de cálculo, permitiendo sobre estas operaciones como la limpieza, la transformación y la agregación de datos con mayor facilidad y menor código. Además, Pandas incorporó una rica funcionalidad para trabajar con datos de series temporales, realizar fusiones y uniones, y manejar datos faltantes de manera más efectiva, estableciendo un nuevo estándar en el análisis de datos en Python.

El Data Wrangling, o preprocesamiento y manipulación de datos, es un conjunto de operaciones en el proceso de Data Science que tienen que ver con tomar los datos "en bruto" provenientes de problemas del mundo real, para organizarlos, "limpiarlos" y convertirlos en un conjunto de datos apropiado para ser utilizado por parte de los algoritmos que se implementan a continuación en dicho proceso. A este conjunto de datos, resultado de la transformación del dataset original se lo denomina "vista minable".

En este trabajo, se presentan las principales funcionalidades de Pandas para el procesamiento de datos aplicadas a conjuntos de datos públicos.

1.2. Estructura de este trabajo

1 Sitio Oficial Pandas: https://pandas.pydata.org/
2 Sitio oficial Python: https://www.python.org/
3 Sitio Oficial Numpy: https://numpy.org/

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 47-70
ISSN 2591-5320

La segunda sección contiene el marco teórico, conteniendo conceptos relacionados con Data Wrangling, Pandas y sus funcionalidades, y orígenes de datos. La tercera sección presenta la aplicación de los conceptos y funciones sobre conjuntos de datos conocidos: Diabetes y Titanic. Se incluye código Python con los ejemplos de aplicación. Finaliza con las conclusiones.

1.3. Motivación y alcance

Las tareas de pre-procesamiento de datos suelen ocupar aproximadamente el 80% del tiempo de trabajo del científico de datos. Pandas presenta algunas funcionalidades acordes con estas tareas. En este trabajo se busca presentar con ejemplos y códigos, su sencilla implementación.

Este trabajo se limita a la presentación con ejemplos, de algunas funcionalidades acordes de Pandas.

2. Marco teórico

2.1. El pre-procesamiento de datos

Las tareas de pre-procesamiento de datos se las resume en las siglas ETL (siglas en inglés de: Extraer, Transformar y Cargar). Se las conoce también como Data Wrangling, y son de importancia fundamental para la obtención de modelos destacados.

La calidad de los datos es un determinante clave del éxito de los modelos predictivos. Datos inconsistentes o incorrectos pueden llevar a interpretaciones erróneas y decisiones subóptimas. El data wrangling permite a los científicos de datos manejar grandes volúmenes de datos heterogéneos y complejos, facilitando la extracción de características relevantes y la creación de conjuntos de datos coherentes y precisos.

2.2. ETL con Pandas

Pandas es una librería de gran importancia para el análisis y manipulación de datos en Python, y su importancia en procesos ETL es ampliamente reconocida. Pandas ofrece estructuras de datos rápidas, flexibles y expresivas, diseñadas para facilitar la manipulación de datos en tiempo real: dataframes, data series, etc.

Una de las principales ventajas de Pandas es su capacidad para manejar grandes volúmenes de datos y realizar operaciones complejas de transformación con una sintaxis sencilla e intuitiva.

2.3. Principales funcionalidades de Pandas para Data Wrangling

Se presenta a continuación las funcionalidades de Pandas más utilizadas en el procesamiento y limpieza de conjuntos de datos.

2.31. Lectura y escritura de datos

Pandas ofrece funciones integradas para leer y escribir datos en varios formatos como CSV, Excel, JSON, y SQL, lo que facilita la importación y exportación de datos desde y hacia distintas fuentes.

Código Python:
```
import pandas as pd

# Leer un archivo CSV
df = pd.read_csv('archivo.csv')

# Escribir el DataFrame en un archivo CSV
df.to_csv('salida.csv', index=False)

# Leer un archivo Excel
df_excel = pd.read_excel('archivo.xlsx')

# Escribir el DataFrame en un archivo Excel
df.to_excel('salida.xlsx', index=False)
```

2.3.2. Tratamiento de Anomalías y Manejo de datos faltantes y nulos

Pandas proporciona diversas funciones para manejar datos faltantes, como *isnull*(), *dropna*(), y *fillna*(), permitiendo identificar, eliminar o rellenar valores nulos en los DataFrames.

Código Python:
```
# Identificar valores faltantes
print(df.isnull().sum())

# Eliminar filas con valores faltantes
df_sin_na = df.dropna()

# Rellenar valores faltantes con un valor específico
df_rellenado = df.fillna(0)
```

2.3.3. Indexación y selección de datos

Pandas facilita la indexación y selección de datos mediante operadores como loc y iloc, permitiendo seleccionar datos por etiquetas o por posición.

La función *iloc* en Pandas se utiliza para realizar la indexación basada en posiciones enteras (en números). Siendo ventajoso cuando es necesario seleccionar filas y columnas de un DataFrame mediante índices numéricos. La sintaxis de *iloc* aporta una gran flexibilidad, permitiendo la selección de datos por un único índice, rango de índices o listas de índices. Por ejemplo, *df.iloc*[0] selecciona la primera fila del DataFrame, mientras que *df.iloc*[:, 1:3] selecciona todas las filas, pero solo las columnas de la segunda a la tercera. Esta capacidad de seleccionar datos basados en su posición numérica resulta de gran utilidad en el preprocesamiento de datos y en la limpieza de datasets cuando los nombres de las columnas no son conocidos de antemano o cuando se quiere trabajar con subconjuntos específicos de datos.

Código Python
```
# Selección de filas 0 a 5 y columnas 0 a 3 (primeras 5 filas, 3 columnas)
filas_posiciones = df.iloc[0:5, 0:3]
```

La función *loc* en Pandas se utiliza para realizar la indexación basada en etiquetas (en letras). Esto significa que se puede acceder a filas y columnas de un DataFrame utilizando los nombres de las etiquetas, en lugar de sus posiciones numéricas. *loc* es particularmente útil cuando se trabaja con DataFrames que poseen índices significativos o cuando se es necesario acceder a datos basados en condiciones específicas. Por ejemplo, *df.loc*[0] selecciona la fila con índice 0 (observar la similitud con el ejemplo con *iloc*), y *df.loc*[:, ['A', 'B']] selecciona todas las filas pero solo las columnas con etiquetas 'A' y 'B' ('A' y 'B' son los nombres de las columnas). Además, *loc* puede ser utilizado para seleccionar subconjuntos de datos basados en condiciones, como *df.loc[df*['A'] > 1], que selecciona todas las filas donde el valor en la columna 'A' es mayor que uno.

Código Python
 # Selección de las filas 0 a 5, columnas con nombre 'columna1'
filas_etiquetadas = df.loc[0:5, 'columna1':'columna3']

 # Selección por posición con iloc
filas_posiciones = df.iloc[0:5, 0:3]

2.3.4. Agrupación y agregación

Las funciones *groupby()* y métodos de agregación permiten agrupar datos por una o más columnas y aplicar funciones de agregación como sum, mean, count, etc.

Código Python
 # Agrupar por una columna y calcular la media
agrupado = df.groupby('columna1').mean()

 # Agrupar por múltiples columnas y contar los registros
agrupado_multiples = df.groupby(['columna1', 'columna2']).count()

2.3.5. Fusiones y uniones

Pandas permite combinar DataFrames utilizando las funciones *merge(), join()*, y *concat(),* facilitando la unión de datos de distintas fuentes.

Código Python:
 # Combinar dos DataFrames con merge
df_merged = pd.merge(df1, df2, on='columna_comun')

 # Unir DataFrames por índice con join
df_joined = df1.join(df2, how='inner')

 # Concatenar DataFrames
df_concatenado = pd.concat([df1, df2], axis=0)

2.3.6. Transformación de datos: Normalización y Discretización

Normalización de Datos

Es el proceso de ajustar los valores de una característica para que estén en un rango específico, normalmente entre 0 y 1. Esto se hace para asegurar que todas las características contribuyan

de manera equitativa al análisis, especialmente en algoritmos sensibles a la escala de los datos, como las redes neuronales.

Código Python:
```
from sklearn.preprocessing import MinMaxScaler

# Crear el escalador
scaler = MinMaxScaler()

# Ajustar y transformar el df
df_normalized = scaler.fit_transform(df)
```

Discretización de Datos

La discretización convierte valores continuos en categorías discretas. Una técnica común es el *one hot encoding,* que convierte las categorías en columnas binarias.

Código Python:
```
# Ajustar y transformar la columna 'glucose'
df_encoded = pd.get_dummies(df,columns=['columna1','columna2'])
```

2.4. Orígenes de datos

2.4.1. Conjuntos de Datos o Datasets

Este trabajo implementa las funcionalidades sobre los datasets públicos disponibles: Diabetes y Titanic.

El dataset denominado Diabetes contiene datos de pacientes utilizados para predecir un diagnóstico diabetes. Incluye múltiples columnas (variables) con atributos relevantes de los pacientes, como el número de embarazos, la concentración de glucosa en sangre, la presión arterial diastólica, el grosor del pliegue cutáneo del tríceps, la cantidad de insulina, el índice de masa corporal (BMI), la función del pedigrí de la diabetes y la edad. También incluye una variable o columna objetivo (Diabetes) que indica si el paciente tiene diabetes (1) o no (0). Este conjunto de datos es de uso común en la construcción de modelos predictivos y para la investigación médica y en la construcción de modelos predictivos de salud.

Por otro lado, el dataset denominado Titanic, posee información sobre los pasajeros del famoso barco RMS Titanic, que se hundió en el Atlántico Norte tras embestir un témpano durante su viaje inaugural en 1912. Cada fila del dataset contiene datos de un pasajero e incluye diversas características tales como el nombre, el género, la edad, la clase del billete (1ª, 2ª o 3ª clase), el número de hermanos/cónyuges a bordo (SibSp), el número de padres/hijos a bordo (Parch), el número del ticket, la tarifa del ticket, la cabina asignada, el puerto de embarque (C: Cherburgo; Q: Queenstown; S: Southampton) y una columna que indica si el pasajero sobrevivió o no (variable objetivo). Este dataset es ampliamente utilizado para análisis de supervivencia y demostraciones de técnicas de machine learning.

El dataset de Diabetes no presenta datos faltantes ni datos categóricos. Sin embargo, algunas columnas como *Glucose, BloodPressure, SkinThickness, Insulin, y BMI* tienen valores de cero que son físicamente improbables y deberían ser tratados como valores faltantes. Por otro lado, el dataset de Titanic presenta datos faltantes en los campos Age, Cabin y Embarked, y además presenta datos numéricos y categóricos.

Además de permitir la creación de modelos predictivos, estos datasets presentan desafíos de interés para las tareas ETL, de gran interés en este trabajo.

2.4.2. Descarga de los Datasets

Para descargar y convertir estos conjuntos de datos a DataFrames de Pandas en Python, se utilizó la biblioteca *pandas*, que ofrece funciones integradas para leer datos de diversas fuentes, incluyendo archivos CSV y Excel. A continuación, se brinda una explicación de cómo realizar esta tarea para los dos conjuntos de datos: Diabetes y Titanic.

3. Desarrollo Técnico: Aplicación de Funcionalidades de Pandas sobre Datasets públicos

Se procederá a aplicar las funcionalidades más importantes de Pandas mencionadas anteriormente a los datasets de Diabetes y Titanic.

3.1. Carga, Lectura y escritura de datos, descripción

3.1.1. Carga del Conjunto de Datos de Diabetes

Primero, se descarga el conjunto de datos Diabetes (disponible en la carpeta "datasets") y se carga en un DataFrame de Pandas.

Código Python:

```
# Diabetes
#
import pandas as pd
import os
separador=os.sep

# Path del archivo
carpeta=str(os.path.join(os.getcwd(),"datasets"))
archivo=str(os.path.join(carpeta,"diabetes.csv"))

# Convertir archivo a DataFrame de Pandas
df_diabetes = pd.read_csv(archivo)

# Tamaño del datast
print("Tamaño: ",df_diabetes.shape)
```

```
# Columnas
columnas_diabetes=df_diabetes.columns.tolist()
print("Columnas =",columnas_diabetes)
```

Salida de Consola:
Tamaño: (768, 9)

Columnas = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Diabetes']

3.1.2. Descripción del Dataset Diabetes:

Cuestiones como tamaño del dataset, tipos de variables, y estadísticos destacados pueden presentarse inmediatamente luego de cargar el dataset.

Código Python:

```
# Dataset Diabetes:

# Tipos de variables
print("\nTipos de variable:")
print(df_diabetes.info())

# Estadisticos:
print("\nEstadísticos:")
print(df_diabetes.describe())
print(df_diabetes.head())
```

Salida de Consola:

```
Tipos de variable:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Diabetes                  768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

Estadísticos:

	Pregnancies	*Glucose*	*BloodPressure*	*SkinThickness*	*Insulin *
count	*768.000000*	*768.000000*	*768.000000*	*768.000000*	*768.000000*
mean	*3.845052*	*120.894531*	*69.105469*	*20.536458*	*79.799479*
std	*3.369578*	*31.972618*	*19.355807*	*15.952218*	*115.244002*
min	*0.000000*	*0.000000*	*0.000000*	*0.000000*	*0.000000*
25%	*1.000000*	*99.000000*	*62.000000*	*0.000000*	*0.000000*
50%	*3.000000*	*117.000000*	*72.000000*	*23.000000*	*30.500000*
75%	*6.000000*	*140.250000*	*80.000000*	*32.000000*	*127.250000*
max	*17.000000*	*199.000000*	*122.000000*	*99.000000*	*846.000000*

	BMI	*Diabetes Pedigree Function*	*Age*	*Diabetes*
count	*768.000000*	*768.000000*	*768.000000*	*768.000000*
mean	*31.992578*	*428.235091*	*33.240885*	*0.348958*
std	*7.884160*	*340.485655*	*11.760232*	*0.476951*
min	*0.000000*	*0.100000*	*21.000000*	*0.000000*
25%	*27.300000*	*205.000000*	*24.000000*	*0.000000*
50%	*2.000000*	*337.000000*	*29.000000*	*0.000000*
75%	*36.600000*	*591.500000*	*41.000000*	*1.000000*
max	*67.100000*	*2329.000000*	*81.000000*	*1.000000*

3.1.3. Carga del Conjunto de Datos de Titanic

Se procedió a descargar el conjunto de datos Titanic desde un repositorio de Github y a cargarlo en un DataFrame de Pandas.

Código Python:

```
# Titanic
#
# URL del dataset en GitHub
url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv'

# Cargar el dataset desde la URL
df_titanic = pd.read_csv(url)
```

Estos fragmentos de código descargan los archivos CSV directamente y los convierten en DataFrames de Pandas, lo que facilita su manipulación y análisis posterior. Los DataFrames resultantes contendrán los datos en bruto, y a partir de aquí se pueden identificar y abordar los problemas de calidad de datos mencionados anteriormente.

3.1.4. Descripción del Dataset Titanic

Puede obtenerse información que permita describir al dataset.

Código Python:

```
# Dataset Titanic
#
# Tamaño del datast
print("\nTamaño: ",df_titanic.shape)
```

```
# Columnas
columnas_titanic=df_titanic.columns.tolist()
print("\nColumnas=",columnas_titanic)

# Tipos de variables
print("\nTipos de variable:")
print(df_titanic.info())

# Estadisticos:
print("\nEstadísticos:")
print(df_titanic.describe())

# Guardar CSV
nombre_archivo="datasets"+str(separador)+"titanic.csv"
df_titanic.to_csv(nombre_archivo)
```

Nota:
"separador" representa a los separadores de carpeta "/" ó "\". Coloca automáticamente uno u otro tras detectar el OS.

Código Python Previo:

```
import os
separador = os.sep
```

Salida de Consola:

Tamaño: (891, 12)

Columnas= ['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']

Tipos de variable:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):

#Column	Non-Null Count	Dtype
0 PassengerId	891 non-null	int64
1 Survived	891 non-null	int64
2 Pclass	891 non-null	int64
3 Name	891 non-null	object
4 Sex	891 non-null	object
5 Age	714 non-null	float64
6 SibSp	891 non-null	int64
7 Parch	891 non-null	int64
8 Ticket	891 non-null	object
9 Fare	891 non-null	float64
10 Cabin	204 non-null	object

```
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None

Estadísticos:
       PassengerId    Survived      Pclass         Age       SibSp
\
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

             Parch        Fare
count   891.000000  891.000000
mean      0.381594   32.204208
std       0.806057   49.693429
min       0.000000    0.000000
25%       0.000000    7.910400
50%       0.000000   14.454200
75%       0.000000   31.000000
max       6.000000  512.329200
```

3.2. Tratamiento de Anomalías

Luego de descargar el conjunto de datos, se identificaron algunas anomalías comunes como valores duplicados, datos faltantes y tipos de datos inconsistentes.

3.2.1. Anomalías en Dataset Diabetes

Con el siguiente código pueden visualizarse anomalías en Diabetes.

Código Python:

```
# Dataset de Diabetes

# Identificar valores duplicados
duplicados = df_diabetes.duplicated().sum()
print(f"\nValores duplicados: {duplicados}")

# Identificar valores faltantes
valores_faltantes = df_diabetes.isnull().sum()
print("\nValores faltantes por columna:")
print(valores_faltantes)
```

```
# Mostrar Variables con ceros
print("\nCantidad de Ceros:")
for columna in columnas_diabetes[1:]:

    num_zeros = (df_diabetes[columna] == 0).sum()
    print(f"{columna}: {num_zeros}")
```

Salida de consola:

```
Valores duplicados: 0

Valores faltantes por columna:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Diabetes                    0
dtype: int64

Cantidad de Ceros:
Glucose:                    5
BloodPressure:              35
SkinThickness:              227
Insulin:                    374
BMI:                        11
DiabetesPedigreeFunction:   0
Age:                        0
Diabetes:                   500
```

Se puede observar la inexistencia de datos duplicados ni faltantes y la presencia de varios ceros que deben tratarse como datos faltantes.

Cualquier valor nulo en las variables: 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', en este dataset es una anomalía. Estos valores nulos pueden indicar errores en la recopilación de datos, fallos en la entrada de datos o problemas de calidad de los datos que deberan ser corregidos antes de cualquier análisis subsiguiente.

Para tratar estas anomalías se puede abordar alguno de los siguientes caminos:

1. Imputación de Valores Faltantes: Usar la media, mediana o moda de la columna correspondiente para sustituir los valores nulos.
2. Modelos Predictivos: Utilizar otras variables del dataset para predecir y sustituir los valores nulos mediante modelos de regresión.

3. Eliminar Registros: Si el número de valores nulos es pequeño en comparación con el tamaño total del dataset, eliminar esos registros puede ser una opción viable.

Normalmente, si la cantidad de faltantes es significativa, lo más simple y conveniente es reemplazar al valor faltante por algún indicador de tendencia central como ser: media (promedio), mediana o moda. En caso contrario, la eliminación de los registros es recomendado. Una mejor opción consiste en entrenar un modelo con los datos completos, y predecir los valores faltantes, aunque no resulta ser simple.

Manejo de Datos Faltantes

Se considera a los ceros en las variables indicadas como datos faltantes. Reemplazaremos, entonces, a los ceros por el promedio de la columna, obtenido tras extraer los ceros.

Código Python:

```
# Dataset Diabetes:

medias=[]
variables=[]
print("Promedios de columnas excluyendo ceros (reemplazarán a los datos faltantes o nulos)")
for i in range (1,len(columnas_diabetes)-1):
   variable=columnas_diabetes[i]
   mean_excluding_zeros = df_diabetes[df_diabetes[variable] != 0][variable].mean()
   variables.append(variable)
   medias.append(mean_excluding_zeros)
dfMedias=pd.DataFrame()
dfMedias["Variable"]=variables
dfMedias["Promedio"]=medias
print(dfMedias)

print("\nReemplazo de los datos faltantes o nulos)")
# Reemplazar los ceros por el valor medio calculado

columns_to_impute = df_diabetes.columns.difference(['Pregnancies', 'Diabetes'])
# Calcular los promedios excluyendo los ceros para cada columna relevante
mean_values = {}
for column in columns_to_impute:
   mean_values[column] = df_diabetes[df_diabetes[column] != 0][column].mean()

# Reemplazar los ceros por los promedios calculados
for column in columns_to_impute:
   df_diabetes[column] = df_diabetes[column].replace(0, mean_values[column])

# Mostrar los resultados
print(df_diabetes)
```

Salida de consola:
Promedios de columnas excluyendo ceros (reemplazarán a los datos faltantes o nulos)

	Variable	*Promedio*
0	*Glucose*	*121.686763*
1	*BloodPressure*	*72.405184*
2	*SkinThickness*	*29.153420*
3	*Insulin*	*155.548223*
4	*BMI*	*32.457464*
5	*DiabetesPedigreeFunction*	*428.235091*
6	*Age*	*33.240885*

Observese en la Tabla 1 cómo se presenta el dataset tras reemplazar los ceros por promedios.

Tabla 1: Vista del Dataset Diabetes tras tratamiento de anomalías.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Diabetes
0	6	148.0	72.0	35.00000	155.548223	33.6	627.00	50	1
1	1	85.0	66.0	29.00000	155.548223	26.6	351.00	31	0
2	8	183.0	64.0	29.15342	155.548223	23.3	672.00	32	1
3	1	89.0	66.0	23.00000	94.000000	28.1	167.00	21	0
4	0	137.0	40.0	35.00000	168.000000	43.1	2288.00	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.00000	180.000000	32.9	171.00	63	0
764	2	122.0	70.0	27.00000	155.548223	36.8	0.34	27	0
765	5	121.0	72.0	23.00000	112.000000	26.2	245.00	30	0
766	1	126.0	60.0	29.15342	155.548223	30.1	349.00	47	1
767	1	93.0	70.0	31.00000	155.548223	30.4	315.00	23	0

3.2.2. Anomalías en Dataset Titanic

El siguiente código permite visualizar anomalías en Titanic.

Código Python:

```
# Dataset de Titanic

# Identificar valores duplicados
duplicados = df_titanic.duplicated().sum()
print(f"\nValores duplicados: {duplicados}")

# Identificar valores faltantes
valores_faltantes = df_titanic.isnull().sum()
print("\nValores faltantes por columna:")
print(valores_faltantes)
```


Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 47-70
ISSN 2591-5320

```
# Mostrar Variables con ceros
print("\nCantidad de Ceros:")
for columna in columnas_titanic[:]:
    num_zeros = (df_titanic[columna] == 0).sum()
    print(f"{columna}: {num_zeros}")
```

Salida de consola:

```
Valores duplicados: 0
Valores faltantes por columna:
PassengerId     0
Survived        0
Pclass          0
Name            0
Sex             0
Age           177
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin         687
Embarked        2
dtype: int64

Cantidad de Ceros:
PassengerId:    0
Survived:     549
Pclass:         0
Name:           0
Sex:            0
Age:            0
SibSp:        608
Parch:        678
Ticket:         0
Fare:          15
Cabin:          0
Embarked:       0
```

Se puede apreciar la existencia de datos faltantes en Age, Embarked y Cabin. Así como también la presencia de varios ceros, siendo esto razonable, no es necesario un tratamiento especial.

Por otro lado, hay tipos de datos no numéricos (object), los cuales deberían ser discretizados. Esto se hace con una técnica llamada "One-hot-encoding". Es una etapa del pre-procesamiento para convertir datos categóricos en representaciones numéricas compatibles. Esto se tratará más adelante.

Para tratar valores faltantes:

1. Imputación de Valores Faltantes: Usar la media, mediana o moda de la columna correspondiente para sustituir los valores nulos.
2. Modelos Predictivos: Utilizar otras variables del dataset para predecir y sustituir los valores nulos mediante modelos de regresión.
3. Eliminar Registros: Si el número de valores nulos es pequeño en comparación con el tamaño total del dataset, eliminar esos registros puede ser una opción viable.

Manejo de Datos Faltantes:

Tratamiento de los datos faltantes en "Age" y en "Cabin". En el caso de "Age" se procede a imputar la mediana (valor no superado por la mitad de los datos), mientras que a "Cabin", se le imputará el valor "NC".

Código Python:

```
# Dataset Titanic:

# Imputar datos faltantes
# A faltantes de edad se imputa la mediana
df_titanic['Age'].fillna(df_titanic['Age'].median(), inplace=True)

# A faltantes de cabina, se le imputa "NC"
df_titanic['Cabin'].fillna('NC', inplace=True)
```

A continuación, en la Tabla 2 se presenta una vista del dataframe Titanic tras estos cambios.

Tabla 2: Vista del Dataset Titanic tras tratamiento de anomalías

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NC	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NC	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NC	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NC	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	28.0	1	2	W./C. 6607	23.4500	NC	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NC	Q

891 rows × 12 columns

3.3. Indexación y Selección de Datos

Pandas facilita la indexación y selección de datos mediante operadores como *loc* y *iloc*, permitiendo seleccionar datos por etiquetas o por posición.

3.3.1. Uso de iloc en Dataset Titanic

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 47-70
ISSN 2591-5320

Del dataset Titanic, se seleccionan primero a filas 0 a 5, columnas 1 a 4 (recordemos que en python la columna 1 corresponde con la segunda, ya que el índice inicia en 0). Luego, se calcula el promedio de la columna 5 (columna 5: columna con índice 5).

Código Python:

```
# Seleccionar las columnas de la segunda a la cuarta
sub_df=df_titanic.iloc[0:5, 1:4]
print("\nColumnas 1 a 4:\n", sub_df)

# Valor promedio de la columna 5
valor=df_titanic.iloc[:,5].mean()
print("\nValor promedio de la columna 5 (", str(df_titanic.columns.to_list()[5]), "):", str(valor))
```

Salida de consola:

Columnas 1 a 4:

	Survived	Pclass	Name
0	0	3	Braund, Mr. Owen Harris
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...
2	1	3	Heikkinen, Miss. Laina
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)
4	0	3	Allen, Mr. William Henry

Valor promedio de la columna 5 (Age): 29.36158249158249

3.3.2. Uso de loc en Dataset Diabetes

Se utiliza el método *loc* en Pandas para obtener las primeras cinco filas del dataset de diabetes, seleccionando solo las columnas 'Age', 'Glucose', y 'Diabetes' (que representa el valor de diabetes). Se agrega complejidad a la selección: se seleccionan a continuación todas las filas del dataset donde la glucosa supere el valor de 195, presentando solo las columnas 'Age', 'Glucose', y 'Diabetes'.

Código Python:

```
# Seleccionar las primeras 5 filas y las columnas 'Age', 'Glucose' y 'Diabetes'
selected_data = df_diabetes.loc[:4, ['Age', 'Glucose', 'Diabetes']]
print(selected_data)

print("")
# Seleccionar todas las donde glucose > 195 y las columnas 'Age', 'Glucose' y 'Diabetes'
filtrado_Glucose = df_diabetes.loc[df_diabetes['Glucose'] > 195, ['Age', 'Glucose', 'Diabetes']]
print(filtrado_Glucose)
```

Salida de consola:

	Age	Glucose	Diabetes
0	50	148.0	1
1	31	85.0	0
2	32	183.0	1
3	21	89.0	0
4	33	137.0	1

	Age	*Glucose*	*Diabetes*
8	*53*	*197.0*	*1*
22	*41*	*196.0*	*1*
206	*57*	*196.0*	*1*
228	*31*	*197.0*	*0*
359	*29*	*196.0*	*1*
408	*39*	*197.0*	*1*
561	*28*	*198.0*	*1*
579	*62*	*197.0*	*1*

3.4. Agrupación y Agregación

La agrupación y agregación en Pandas son herramientas poderosas para resumir y obtener información significativa de los datos. En los siguientes ejemplos, se presentan estas operaciones en los datasets Titanic y Diabetes, obteniendo información resumida útil. Estas técnicas son esenciales para el análisis exploratorio de datos y la preparación de datos para el modelado.

3.4.1. Agrupación y Agregación sobre Diabetes

Se agrupan los datos por Diagnóstico de Diabetes (positivo=1, negativo=0), y se calcula el menor valor de la edad y el máximo del nivel de Glucosa.

Código Python:

```
# Agrupar por Diagnóstico de diabetes y calcular la menor edad y mayor glucosa
agrupado_diabetes = df_diabetes.groupby('Diabetes').agg({'Age': 'min', 'Glucose': 'max'})
print(agrupado_diabetes)
```

Salida de Consola:

	Age	*Glucose*
Diabetes		
0	*21*	*197.0*
1	*21*	*199.0*

3.4.2. Agrupación y Agregación sobre Titanic

Se agruparán los datos por clase, y se calcula el promedio de la edad y tarifa.

Código Python:

```
# Agrupar por clase de pasajero y calcular la edad y tarifa promedio
agrupado_titanic = df_titanic.groupby('class').agg({'age': 'mean', 'fare': 'mean'})
print(agrupado_titanic)
```

Salida de Consola:

	age	*fare*
class		
First	*38.233441*	*84.154687*
Second	*29.877630*	*20.662183*
Third	*25.140620*	*13.675550*

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 47-70
ISSN 2591-5320

3.5. Fusiones y Uniones

Las fusiones y uniones en Pandas son técnicas poderosas para combinar información de múltiples DataFrames. Los ejemplos siguientes, dan cuenta de cómo realizar fusiones en el dataset Titanic y uniones en el dataset de Diabetes, permitiendo la integración de información diversa para un análisis más completo. Estas técnicas son esenciales en el análisis de datos para obtener una visión holística de los datos disponibles.

3.5.1. Uniones (joins) sobre Diabetes

Se presenta a continuación una unión entre dos DataFrames que contienen información diferente sobre pacientes con diabetes. Un DataFrame contiene medidas médicas y el otro contiene información demográfica.

Código Python:

```
# Crear DataFrames adicionales para simular la unión
df_medidas = df_diabetes[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',  'Age']]
print(df_medidas)
print("")
df_demografico = df_diabetes[['Age', 'Diabetes']]
print(df_demografico)
print("")
# Realizar la unión
unido_diabetes = df_medidas.join(df_demografico.set_index('Age'), on='Age', how='inner', lsuffix='_
med', rsuffix='_demo')
print(unido_diabetes.head(10))
```

Explicación del Código:

1. Carga del Dataset: Utilización de Pandas para cargar el dataset de Diabetes desde un archivo CSV.
2. Crear DataFrames Adicionales: Creación de dos DataFrames adicionales, df_medidas y df_demografico, para simular la necesidad de una unión.
3. Unión: Se utiliza join() para unir los DataFrames sobre la columna 'Age', mediante una unión interna ('inner').

Salida de Consola:

Datos médicos:

	Glucose	*BloodPressure*	*SkinThickness*	*Insulin*	*BMI*	*Age*
0	*148.0*	*72.0*	*35.00000*	*155.548223*	*33.6*	*50*
1	*85.0*	*66.0*	*29.00000*	*155.548223*	*26.6*	*31*
2	*83.0*	*64.0*	*29.15342*	*155.548223*	*23.3*	*32*
3	*89.0*	*66.0*	*23.00000*	*94.000000*	*28.1*	*21*
4	*137.0*	*40.0*	*35.00000*	*168.000000*	*43.1*	*33*
..	*..*	*...*	*...*	*...*	*...*	*...*
763	*101.0*	*76.0*	*48.00000*	*180.000000*	*32.9*	*63*
764	*122.0*	*70.0*	*27.00000*	*155.548223*	*36.8*	*27*
765	*121.0*	*72.0*	*23.00000*	*112.000000*	*26.2*	*30*
766	*126.0*	*60.0*	*29.15342*	*155.548223*	*30.1*	*47*
767	*93.0*	*70.0*	*31.00000*	*155.548223*	*30.4*	*23*

[768 rows x 6 columns]

Datos demográficos:

	Age	*Diabetes*
0	*50*	*1*
1	*31*	*0*
2	*32*	*1*
3	*21*	*0*
4	*33*	*1*
..	*...*	*...*
763	*63*	*0*
764	*27*	*0*
765	*30*	*0*
766	*47*	*1*
767	*23*	*0*

[768 rows x 2 columns]

Union de Dataframes

	Glucose	*BloodPressure*	*SkinThickness*	*Insulin*	*BMI*	*Age*	*Diabetes*
0	*148.0*	*72.0*	*35.00000*	*155.548223*	*33.6*	*50*	*1*
0	*148.0*	*72.0*	*35.00000*	*155.548223*	*33.6*	*50*	*0*
0	*148.0*	*72.0*	*35.00000*	*155.548223*	*33.6*	*50*	*0*
0	*148.0*	*72.0*	*35.00000*	*155.548223*	*33.6*	*50*	*0*
0	*148.0*	*72.0*	*35.00000*	*155.548223*	*33.6*	*50*	*1*
0	*148.0*	*72.0*	*35.00000*	*155.548223*	*33.6*	*50*	*1*
0	*148.0*	*72.0*	*35.00000*	*155.548223*	*33.6*	*50*	*1*
0	*148.0*	*72.0*	*35.00000*	*155.548223*	*33.6*	*50*	*1*
21	*99.0*	*84.0*	*29.15342*	*155.548223*	*35.4*	*50*	*1*
21	99.0	84.0	29.15342	155.548223	35.4	50	0

3.5.2. Fusiones (merge) sobre Titanic

Se lleva adelante una fusión entre dos DataFrames simulando que uno contiene información básica de pasajeros y el otro contiene detalles adicionales, utilizando una columna común, como ser: 'Pclass'.

Código Python:

```
# Crear un DataFrame adicional para simular la fusión
data_adicional = {'Pclass': [1, 2, 3], 'Average_Fare': [84.154687, 20.662183, 13.675550]}
df_adicional = pd.DataFrame(data_adicional)
# Realizar la fusión
fusionado_titanic = pd.merge(df_titanic, df_adicional, on='Pclass', how='left')
print(fusionado_titanic[['Pclass', 'Fare', 'Average_Fare']].head(10))
```

Explicación del Código

1. Carga del Dataset: Utilizamos seaborn para cargar el dataset Titanic.
2. Crear un DataFrame Adicional: Creamos un DataFrame adicional df_adicional que contiene las columnas 'Pclass' y 'Average_Fare'.
3. Fusión: Utilizamos pd.merge() para fusionar los DataFrames sobre la columna 'Pclass', utilizando una unión izquierda ('left').

Salida de Consola:

	Pclass	Fare	Average_Fare
0	3	7.2500	13.675550
1	1	71.2833	84.154687
2	3	7.9250	13.675550
3	1	53.1000	84.154687
4	3	8.0500	13.675550
5	3	8.4583	13.675550
6	1	51.8625	84.154687
7	3	21.0750	13.675550
8	3	11.1333	13.675550
9	2	30.0708	20.662183

3.6. Transformación de Datos

3.6.1. Normalización sobre Diabetes

Se lleva adelante la normalización (explicado en 2.1.6) del conjunto de datos de Diabetes.

Código Python:

```
from sklearn.preprocessing import StandardScaler

# Crear el escalador estándar
scaler = StandardScaler()
target = df_diabetes['Diabetes']

# Ajustar el escalador y transformar los datos
data_normalized = scaler.fit_transform(df_diabetes)

# Convertir el resultado a un DataFrame y agregar de nuevo la columna 'Outcome'
data_normalized = pd.DataFrame(data_normalized, columns=df_diabetes.columns)
data_normalized['Diabetes'] = target

# Mostrar los primeros registros del dataset normalizado
print("Primeros registros del dataset normalizado:")
print(data_normalized.head())
```

Salida de Consola:

Primeros registros del dataset normalizado:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
0	0.639947	0.865108	-0.033518	6.655021e-01	-3.345079e-16	
1	-0.844885	-1.206162	-0.529859	-1.746338e-02	-3.345079e-16	
2	1.233880	2.015813	-0.695306	8.087936e-16	-3.345079e-16	
3	-0.844885	-1.074652	-0.529859	-7.004289e-01	-7.243887e-01	
4	-1.141852	0.503458	-2.680669	6.655021e-01	1.465506e-01	

	BMI	*DiabetesPedigreeFunction*	*Age*	*Diabetes*
0	*0.166292*	*0.584149*	*1.425995*	*1*
1	*-0.852531*	*-0.226986*	*-0.190672*	*0*
2	*-1.332833*	*0.716400*	*-0.105584*	*1*
3	*-0.634212*	*-0.767743*	*-1.041549*	*0*
4	*1.548980*	*5.465654*	*-0.020496*	*1*

3.6.2. Discretización sobre Titanic

Antes de discretizar, se procede a analizar las variables categóricas, para visualizar cuáles de ellas son susceptibles al proceso de discretización.

Código Python:

```
# Tratamiento de las variables categoricas:
variables_categoricas=['Name','Sex','Cabin','Ticket','Embarked']

# Cantidad de valores posibles (no numéricos) por variable
df_cat=pd.DataFrame()
val_posibles=[]
for i in range(len(variables_categoricas)):
    variable=variables_categoricas[i]
    valor=len(list(set(list(df_titanic[variable]))))
    val_posibles.append(valor)
df_cat["Variable"]=variables_categoricas
df_cat["valores posibles"]=val_posibles
print("Valores posibles por variable categorica:")
print(df_cat)
```

Salida de consola:

Valores posibles por variable categorica:

	Variable	*valores posibles*
0	*Name*	*891*
1	*Sex*	*2*
2	*Cabin*	*148*
3	*Ticket*	*681*
4	*Embarked*	*4*

Al observar la salida, se evidencia la conveniencia de realizar el proceso de discretización (one-hot-encoding) sobre las variables "Embarked" y "Sex". Además, si se observa que la variable numérica "Pclass" actúa como categórica, más que numérica, se incorporará a esta variable en el proceso. Se sugiere la eliminación de columnas PassangerId, Name, Cabin y Ticket

Código Python:

```
# Eliminacion de columnas
df_titanic_cleaned = df_titanic.drop(columns=['Name', 'Cabin', 'Ticket', PassengerId])
# Discretizacion de variables:
df_titanic_encoded=pd.get_dummies(df_titanic_cleaned,columns=['Pclass','Sex','Embarked'])
df_titanic_encoded
```

Explicación del Código:

1. Crear DataFrame de Ejemplo: Creamos un DataFrame df con las columnas Pclass, Sex, Embarked, Name, Cabin, y Ticket.
2. Eliminar Columnas: Usamos df.drop(columns=['Name', 'Cabin', 'Ticket']) para eliminar las columnas no deseadas.
3. One Hot Encoding: Aplicamos one hot encoding a las columnas restantes (Pclass, Sex, Embarked).

En la Tabla 3 puede observarse al nuevo dataset tras haberse aplicado el proceso de discretización.

Tabla 3: Vista del Dataset Diabetes tras tratamiento de anomalías.

	Survived	Age	SibSp	Parch	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	0	22.0	1	0	7.2500	0	0	1	0	1	0	0	1
1	1	38.0	1	0	71.2833	1	0	0	1	0	1	0	0
2	1	26.0	0	0	7.9250	0	0	1	1	0	0	0	1
3	1	35.0	1	0	53.1000	1	0	0	1	0	0	0	1
4	0	35.0	0	0	8.0500	0	0	1	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	27.0	0	0	13.0000	0	1	0	0	1	0	0	1
887	1	19.0	0	0	30.0000	1	0	0	1	0	0	0	1
888	0	28.0	1	2	23.4500	0	0	1	1	0	0	0	1
889	1	26.0	0	0	30.0000	1	0	0	0	1	1	0	0
890	0	32.0	0	0	7.7500	0	0	1	0	1	0	1	0

891 rows × 13 columns

Conclusiones

El uso de Pandas en Python ha transformado la manipulación y análisis de datos, haciéndolos más accesibles y eficientes. Trabajar con los datasets Diabetes y Titanic, permitió dar cuenta de las principales funcionalidades de Pandas en el manejo de datos faltantes, seleccionar y indexar datos, agrupar y agregar información, fusionar y unir datasets, y transformar datos.

1. Lectura y Escritura de Datos: La facilidad para leer y escribir archivos en diferentes formatos (CSV, Excel), disponibles localmente o descargados desde una url nos permitió integrar los datasets rápidamente en nuestros proyectos y guardar los resultados para su posterior análisis.
2. Manejo de Datos Faltantes: Se identificaron y manejaron valores nulos y faltantes de manera eficiente, lo que resulta crucial para mantener la integridad del análisis de datos.
3. Indexación y Selección de Datos: Las funciones loc y iloc permitieron una selección precisa de datos, ya sea por etiquetas o por posiciones, facilitando el acceso a subconjuntos específicos de los datasets.
4. Agrupación y Agregación: Se agruparon datos y se calcularon estadísticas agregadas, lo que nos permitió obtener detalles valiosos sobre la distribución y características de los datos.
5. Fusiones y Uniones: La capacidad de fusionar y unir datasets nos permitió combinar información de múltiples fuentes, enriqueciendo nuestro análisis.
6. Transformaciones de datos: como normalización y discretización presentan vistas equivalentes del dataset original, pero más aceptable por parte de muchos algoritmos.

En resumen, Pandas proporciona una serie de herramientas poderosas y flexibles que facilitan enormemente el proceso de data wrangling, haciendo que tareas complejas sean más manejables y accesibles incluso para aquellos que no son expertos en programación. La integración de estas herramientas en el flujo de trabajo de análisis de datos permite tomar decisiones informadas basadas en datos limpios y bien estructurados.

Referencias

» **McKinney, W. "Data Structures for Statistical Computing in Python"**. En Proceedings of the 9th Python in Science Conference (SciPy 2010).

» **McKinney, W. "Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython"**. O'Reilly Media. ISBN: 978-1449319793 (2012).

» Muhammad Arham, "**Pandas. How to One-Hot-Encode Data**", KD-Nuggets (2023). En linea: https://www.kdnuggets.com/2023/07/pandas-onehot-encode-data.html

» McKinney, Wes. **"Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython."** O'Reilly Media, 2017. ISBN: 978-1491957660.

» Chen, D., Ma, C., & Zheng, L. (2021). "**Pandas for Everyone: Python Data Analysis**". Addison-Wesley Professional.

» Wes McKinney. **"Data Structures for Statistical Computing in Python."** Proceedings of the 9th Python in Science Conference, 2010. Enlace al documento: https://conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf

» **Pandas Documentation.** The official documentation for Pandas. Enlace a la documentación: https://pandas.pydata.org/pandas-docs/stable/

» Kahn,Michael. **Diabetes.** UCI Machine Learning Repository. https://doi.org/10.24432/C5T59G.

» Kaggle, **Titanic - Machine Learning from Disaster**. Enlace: https://www.kaggle.com/c/titanic/data