Resumen del capítulo: Tipos de datos

Variables categóricas y cuantitativas

Hay dos tipos básicos de variables estadísticas: categóricas y cuantitativas. Las variables categóricas toman sus valores de un conjunto limitado, mientras que las variables cuantitativas toman valores numéricos de un rango. Es importante diferenciarlos porque los métodos de análisis de datos pueden estar limitados según el tipo de datos o incluso ser aplicables a solo uno de los tipos de datos.

También hay otros tipos de variables, pero esto tiene más que ver con conceptos centrados en el software que en el análisis de datos. Estos son algunos ejemplos de otros tipos de datos:

- Lógicos (booleanos), es decir, indican si una sentencia es verdadera o falsa. Si una sentencia es verdadera, la variable toma el valor 1. Si una sentencia es falsa, es 0.
- Strings.
- Fechas y marcas temporales.

Conversión de los valores de columna a un tipo diferente

La conversión de valores de columna a un tipo diferente (int para números enteros, por ejemplo) reemplazando la columna actual.

```
df['column'] = df['column'].astype('int')
```

La conversión de valores de columna a un tipo diferente (int para números enteros, por ejemplo) sin reemplazar la columna actual.

```
df['new_column'] = df['column'].astype('int')
```

Conversión de los valores de string a números

Usamos un método estándar de pandas para convertir los valores de string a números: `to_numeric()`. Convierte los valores de columna al tipo **float64** (número de coma flotante) o **int64** (número entero), según el valor de entrada.

El método `to_numeric()` tiene un parámetro `errors`. Este parámetro determina qué hará `to_numeric` al encontrar un valor no válido:

- `errors='raise'` (predeterminado): se genera una excepción cuando se encuentra un valor incorrecto, lo que detiene la conversión a números.
- `errors='coerce'`: los valores incorrectos se reemplazan por `NaN`.
- `errors='ignore'`: los valores incorrectos no se modifican.

```
pd.to_numeric(df['column'])
pd.to_numeric(df['column'], errors='raise' )
pd.to_numeric(df['column'], errors='coerce')
pd.to_numeric(df['column'], errors='ignore')
```

Conversión de strings a horas y fechas

Python tiene un tipo de datos especial que utilizamos cuando trabajamos con fechas y horas: **datetime**.

Para convertir los strings a fechas y horas, usamos el método `to_datetime()` de pandas. Los parámetros del método incluyen el nombre de la columna que contiene strings y el formato de fecha en un string.

Establecemos el formato de fecha utilizando un sistema de designación especial:

- `%d`: día del mes (01 a 31);
- `%m`: mes (01 a 12);
- `%Y`: año en cuatro dígitos (por ejemplo, 1994);
- `%y`: año en dos dígitos (por ejemplo, 94);
- `Z` o `T`: separador estándar para la fecha y la hora;

- `%H`: hora en formato de 24 horas;
- `%I`: hora en formato de 12 horas;
- `%M`: minutos (00 a 59);
- `%S`: segundos (00 a 59).

```
df['column']= pd.to_datetime(df['column'], format='%d.%m.%Y %H:%M:%S')
```

Puedes consultar la documentación aquí para obtener más información sobre los códigos de formato de fecha y hora.

Extracción de componentes de tiempo

A menudo tenemos que estudiar las estadísticas por mes, día o año. Para hacerlo, podemos colocar la hora en la clase DatetimeIndex y aplicarle el atributo month, day o year:

```
pd.DatetimeIndex(df['column']).year
pd.DatetimeIndex(df['column']).month
pd.DatetimeIndex(df['column']).day
pd.DatetimeIndex(df['column']).hour
pd.DatetimeIndex(df['column']).minute
pd.DatetimeIndex(df['column']).second
```

Para las columnas con valores similares a fecha y hora, también puedes acceder a estas propiedades a través del accesor .dt.

```
df['time'].dt.year
df['time'].dt.month
df['time'].dt.day
df['time'].dt.hour
df['time'].dt.minute
df['time'].dt.second
```

La lista completa de componentes se puede encontrar aquí. Por ejemplo, podemos encontrar el día de la semana con el método `dt.weekday()`.