

Resumen del capítulo: Estudio de data slices

Segmentos de datos y búsqueda de billetes de avión

Cuando estás trabajando en una tarea, a menudo necesitas usar una porción de datos cuidadosamente seleccionada, o un **segmento de datos**.

Una forma de obtener un segmento es aplicar un filtro. Una opción para ese filtro es una **matriz booleana** con valores verdadero(true)/falso(false). `True` marcará las líneas que queremos incluir en nuestro segmento de datos (líneas con un valor determinado, por ejemplo) mientras que `False` indicará valores que no queremos incluir. El problema es que marcar líneas manualmente lleva tiempo. Afortunadamente, pandas tiene filtros automáticos. Aquí tienes un ejemplo:

```
data['column'] == 'value'
```

Ese filtro servirá como índice de DataFrame:

```
data[data['column'] == 'value']
```

Además del signo igual, la condición para crear una matriz booleana también puede usar símbolos de comparación como `!=`, `>`, `>=`, `<` y `<=`. Podemos comparar los valores de las columnas entre sí, no solo con números:

```
data['column1'] > data['column2']
```





También podemos usar operaciones aritméticas en la condición:

```
data['column1'] / 2 > data['column2'] + 0.5
```

Para ver si hay valores particulares en la columna, podemos llamar al método `isin()`:

```
data['column'].isin(['value1', 'value2', 'value3'])
```

A veces, necesitas una selección que cumpla con múltiples condiciones simultáneamente; en esos casos, las operaciones lógicas hacen un buen trabajo. Las condiciones se colocan entre paréntesis mientras que los operadores mismos son símbolos:

 Name	 Description	 Syntax	 Operator
<u>AND</u>	El resultado de la operación lógica es True, solo cuando ambas condiciones son True.	(df['Is_Direct']) & (df['Price'] < 210)	&
<u>OR</u>	El resultado es True si al menos una de las condiciones es True.	(df['Has_luggage']) (df['Price'] < 200)	
<u>NOT</u>	El resultado es True si la condición es False"	~((df['Is_Direct']) (df['Has_luggage']))	~

Creación de segmentos de datos con el método query()

Una forma más sencilla de obtener segmentos es el método `query()`. Ponemos la condición para el segmento en la cadena dada como argumento para el método `query()`, y luego la aplicamos al DataFrame. Eso nos da el segmento que estamos buscando.

```
data.query('column == value')
```

Condiciones establecidas con el parámetro `query()`:

- Admiten varios operadores de comparación, como `!=`, `>`, `>=`, `<` y `<=`
- Pueden comprobar si los valores específicos están en la lista usando construcciones como `Date_To in ("07.07.19", "09.07.2019")` o si los valores NO están en la lista usando construcciones como `Date_To not in ("07.07.19", "09.07.2019")`
- Trabajan con los operadores lógicos AND, OR y NOT de la forma habitual, sin necesidad de poner las condiciones entre paréntesis, ya que las operaciones se ejecutan siempre en el orden *no*, *y*, *o* cuando no hay paréntesis.

También podemos extraer valores de variables externas en el método `query()`:

```
variable = 2
data.query('column > @variable')
```

Trabajar con horas y fechas

¿Recuerdas el método `to_datetime()` del curso sobre preprocesamiento que usamos para convertir cadenas en fechas? El argumento `format` de este método sigue el mismo orden que tenemos en la cadena `date_time`:

- `%d`: día del mes (01 a 31)
- `%m`: mes (01 a 12)
- `%Y`: año de cuatro dígitos (por ejemplo, 1994)
- `%y`: año de dos dígitos (por ejemplo, 94)
- `Z` o `T`: separador estándar para fecha y hora
- `%H`: hora en formato de 24 horas
- `%I`: hora en formato de 12 horas
- `%M`: para minutos (00 a 59)
- `%S`: para segundos (00 a 59)

Los expertos en datos suelen utilizar el atributo **dt** (fecha-hora) para notificar explícitamente a pandas que la operación tendrá que ver con las fechas. `dt` indica que el tipo de datos al que vamos a aplicar los métodos es fecha y hora por lo que pandas no lo tratará como una cadena o un número.

Para redondear las horas, utiliza el método `dt.round()`. Se le pasan cadenas que indican si el redondeo debe ser al día, la hora, el minuto o el segundo:

- `D`: día
- `H`: hora

- `min` o `T`: minuto
- `S`: segundo

```
data['datetime_round'] = df['datetime'].dt.round('3H')
```

Si quieres redondear hacia arriba, usa el **método** `dt.ceil()` (*ceiling*).^{*} Para redondear hacia abajo, usa `dt.floor()`.

Podemos encontrar el día de la semana con el método `dt.weekday()`.

Gráficos

`plot()` construye gráficos usando los valores en las columnas de DataFrame. Los índices están en el eje X y los valores de las columnas están en el eje Y.

```
data.plot()
```

El método `plot()` también tiene el parámetro `style` que es responsable del aspecto de los puntos:

- `'o'`: en lugar de una línea continua, cada valor se marcará como un punto
- `'x'`: en lugar de una línea continua, cada punto se marcará como una x
- `'o-'`: mostrará tanto líneas como puntos

Puedes cambiar los índices de los ejes, asignando los valores de la columna que necesitamos al eje correspondiente.

```
data.plot(x='column_x', y='column_y')
```

Podemos corregir los bordes usando los parámetros `xlim` y `ylim` que aprendiste al estudiar diagramas de caja:

```
data.plot(xlim=(x_min, x_max), ylim=(y_min, y_max))
```

Para mostrar líneas de cuadrícula, establece el parámetro `grid` en `True`:

```
data.plot(grid=True)
```

Podemos gestionar el tamaño del gráfico con el parámetro `figsize`. El ancho y alto en pulgadas se pasan entre paréntesis:

```
data.plot(figsize = (x_size, y_size))
```

Agrupación con `pivot_table()`

Cuando hay muchos datos, los puntos se fusionan, lo que dificulta sacar conclusiones. Puedes mejorar la visualización agrupando los datos. El siguiente ejemplo utiliza la agrupación para que el gráfico sea mucho más informativo.

```
(data
    .query('column_id == "value"')
    .pivot_table(index='column1', values='column2')
    .plot(grid=True, figsize=(12, 5))
)
```

Los gráficos con agrupación te ayudan a encontrar valores atípicos que antes no eran visibles. Y a veces estos valores atípicos sesgan seriamente la media. ¿Cómo te aseguras de que las anomalías no aumenten la media? Hay dos posibles soluciones:

- Deshacerte de los valores atípicos
- Utilizar la mediana en lugar de la media. La mediana es resistente a los valores atípicos, aunque aún es imperfecta: aún puede haber picos.

Guardar los resultados

Mientras realizas tu trabajo, es posible que encuentres errores en los datos. En ese caso, debes formular el problema de una manera que te permita simplificar tu búsqueda de problemas potenciales en el algoritmo para exportar datos. Para informar a las personas acerca de los errores, debes enviar un **informe de errores** aclarando qué error es y explicando cómo replicarlo.