Resumen del capítulo: Trabajar con varias fuentes de datos

Segmentos de datos de diccionarios externos

Al trabajar con segmentos, puedes usar variables externas no solo de tipo numérico o de cadena. A veces necesitamos crear consultas usando variables que contienen estructuras más complicadas que un número o una lista independientes, por ejemplo, un diccionario, un objeto Series o un DataFrame. Puedes abordarlos como lo harías con las variables externas ordinarias. Para saber si los valores de una columna están en la lista, escribe la consulta `' in @our_list'`.

```
our_list = [1, 2, 3]
print(data.query('column in @our_list'))
```

Se realiza una verificación similar para los diccionarios. Comprobamos la presencia de un cierto valor en las claves del diccionario:

```
our_dict = {0: 10, 1: 11, 2: 12}
print(data.query('column in @our_dict'))
```

Si la variable almacena un objeto Series, entonces la declaración `in @our_series` comprobará la presencia entre los *valores* de ese Series, en lugar de entre sus *índices*.

```
our_series = pd.Series([10,11,12])
print(data.query('column in @our_series'))
```

Si necesitas comprobar si los valores `a` están presentes entre los índices del Series, usa el atributo `.index`: `a in @our_series.index`.

```
our_series = pd.Series([10,11,12])
print(data.query('column in @our_series.index'))
```

Cuando estamos trabajando con un objeto DataFrame, la inclusión en su índice se comprueba de la misma manera. Simplemente agregamos `.index` al nombre del DataFrame:

```
our_dataframe = pd.DataFrame ({
'column1': [2, 4, 6],
'column2': [3, 2, 2],
'column3': ['A', 'B', 'C'],
})
print(data.query('column in @our_dataframe.index'))
```

Para comprobar la presencia en una columna, pasa el nombre de la columna:

```
our_dataframe = pd.DataFrame ({
'column1': [2, 4, 6],
'column2': [3, 2, 2],
'column3': ['A', 'B', 'C'],
})
print(data.query('column in @our_dataframe.column2'))
```

Agregar una columna

Se pueden mostrar varios histogramas en un solo gráfico.Para hacerlo, puedes usar la siguiente construcción:

```
ax = data1.plot(kind='hist', y='column1', histtype='step', range=(0, 500), bins=25,
                                linewidth=5, alpha=0.7, label='raw')
data2.plot(kind='hist', y='column1', histtype='step', range=(0, 500), bins=25,
                       linewidth=5, alpha=0.7, label='filtered', ax=ax, grid=True, legend=True)
```

Ten en cuenta que llamamos al método `plot()` en lugar de `hist()`. Contiene el parámetro `kind` con su valor establecido en `kind = 'hist'`. Obtenemos el mismo histograma, simplemente trazado usando `plot()` en lugar de `hist()` con los siguientes parámetros:

- `histtype`: este parámetro indica el tipo de histograma; por defecto es un gráfico de barras (relleno). El valor `step` solo dibuja una línea.
- `linewidth`: ancho de línea en píxeles.
- `alpha` ("alpha channel"): establece la densidad de relleno de línea. 1 está lleno al 100% mientras que 0 es transparente. Las líneas son parcialmente transparentes con el parámetro establecido en 0,7, lo que facilita la detección de intersecciones.
- `label`: nombre de línea.
- `ax` (para "axis"): el método `plot()` devuelve los ejes sobre los que se construyó el gráfico. Para que ambos histogramas se ubiquen en un solo gráfico, guarda los ejes del primer gráfico en la variable `ax` y pásalo al parámetro `ax` del segundo `plot()`. Por lo tanto, al guardar los ejes de un histograma y construir un segundo sobre los ejes del primero, fusionamos dos gráficos.

- `legend`: muestra una leyenda (una lista de los elementos de un gráfico). Se puede encontrar en la esquina superior derecha del gráfico.

Agregar columnas a un DataFrame: para agregar una columna de `data2` a `data1`, crea una nueva columna en `data1` y asígnale los valores de la columna `data2`:

```
data1['new'] = df2['column']
```

Si la columna `new` ya hubiese estado en `df1`, todos sus elementos habrían sido reemplazados por los nuevos.

Parece relativamente sencillo: pandas copia una columna de `data2` y la coloca en `data1`.

Sin embargo, si miras más de cerca, las cosas no son tan sencillas. Para cada fila del primer DataFrame, pandas busca un "compañero", una fila con el mismo índice en el segundo DataFrame, y toma el valor de esa fila. En nuestro caso, los índices en `data1` y `data2` son los mismos, así que este es un caso trivial: todos los valores se copian en el mismo orden en que están posicionados.Sin embargo, si los índices son diferentes, obtendremos valores `NaN` donde los índices están ausentes.

Observa que nuestro DataFrame no tiene que tener el mismo número de filas. Si `data2` no tiene tantas filas como `data1`, terminaremos con algunos valores `NaN`. Si `data2` tiene más filas, simplemente no formarán parte del nuevo DataFrame.

Las columnas también pueden existir por separado, fuera de los DataFrames. Un sola columna puede guardarse en un objeto Series – un array (matriz) de valores con índices. Como Series tiene índices, la asignación de un Series a, digamos, una columna de un DataFrame, funcionará de la misma forma que hemos visto antes: los valores se copiarán en función de los índices correspondientes.

Combinar datos de dos tablas

Al trabajar en tareas, es importante elegir el método de promedio adecuado, ya que eso podría afectar los resultados. En algunos casos, la media aritmética describe los datos con mayor precisión mientras que en otros puede producir un resultado incorrecto por lo que es necesario calcular la mediana.

Recuerda que el método `pivot_table()` agrupa datos mientras que el argumento `aggfunc` define cómo procesar los datos en cada grupo. Por ejemplo, hay:

- `median`

- `count` (número de valores)
- `sum`
- `min`
- `max`
- `first` (el primer valor del grupo)
- `last` (el último valor del grupo)

Cuando llamamos a `pivot_table()`, podemos pasar al parámetro `aggfunc` varias funciones a la vez. Por ejemplo, `aggfunc=['median', 'count']` calculará tanto la mediana como el número de valores. Se mostrarán en las columnas vecinas de la tabla resultante.

Cambio de nombre de las columnas

Hemos tratado con una tabla con nombres de columnas de dos niveles. Esto es un MultiIndex (índice múltiple), un tipo de estructura jerárquica de índices que vemos cuando un índice contiene una lista de valores en lugar de un solo valor.

Pero, ¿qué ocurre si no queremos esos nombres de columnas complejos? Entonces necesitamos renombrar nuestras columnas utilizando el atributo `columns` :

```
df.columns = ['column_name_1', 'column_name_2', 'column_name_3']
```

Combinar columnas con los métodos merge() y join()

Cuando tienes que agregar varias columnas nuevas a un DataFrame existente, **unir** o **fusionar** es más eficiente que agregarlas una por una.

La fusión interna da como resultado la conjunción lógica de ambas tablas (los registros que están presentes en ambos DataFrames). `inner` es el método de fusión predeterminado del método `merge()`. **Fusión externa** (`outer`) da la disyunción lógica de ambas tablas (los registros que están presentes en cualquiera de los dos DataFrames). El método de fusión se establece en el parámetro `how`.

```
data1.merge(data2, on='column', how='inner')
```

El modo de combinación `left` indica que todas las filas del DataFrame izquierdo siempre deberían introducir el resultado de la fusión. Si hay valores ausentes en el DataFrame derecho, se reemplazarán con `NaN`. Probablemente puedas suponer que `right` te da

todas las filas correspondientes y todas las otras entradas del DataFrame derecho (siendo el DataFrame derecho el que está dentro de los paréntesis).

En la tabla que tenemos arriba, utilizando el método `merge()` se añadieron `_x` e `_y` a los nombres de columnas. Las terminaciones de los nombres de las columnas se especifican en el argumento `suffixes`.

Si se da un nombre a una columna de índice, el nombre también puede pasarse al parámetro `on`. Combinar varias columnas a la vez también es posible, solo pasa una lista de ellas al argumento `on`.

El método `join()` es similar al método `merge()`. Sin el parámetro `on`, `join()` automáticamente buscará correspondencias en función de los índices del primer y segundo DataFrames. Si una columna se pasa al parámetro `on`, `join()` lo encuentra en el primer DataFrame y comienza a compararlo con el índice del segundo DataFrame. En `join()`, a diferencia de `merge()`, el tipo de fusión `how='left'` se establece por defecto.Pero el parámetro `suffixes` se divide en dos independientes: `lsuffix` ("sufijo izquierdo") y `rsuffix` ("sufijo derecho"). También es posible combinar más de dos tablas utilizando el método `join()`: se pasan como una lista en lugar del segundo DataFrame.