Exercise1

Setting and removing indexes

pandas allows you to designate columns as an index. This enables cleaner code when taking subsets (as well as providing more efficient lookup under some circumstances).

In this chapter, you'll be exploring temperatures, a DataFrame of average temperatures in cities around the world. pandas is loaded as pd.

Instructions

- Look at temperatures.
- Set the index of temperatures to "city", assigning to temperatures_ind.
- Look at temperatures_ind. How is it different from temperatures?
- Reset the index of temperatures_ind, keeping its contents.
- Reset the index of temperatures_ind, dropping its contents.

In [ ]:

```
# Look at temperatures
print(____)

# Index temperatures by city
temperatures_ind = ____

# Look at temperatures_ind
print(____)

# Reset the index, keeping its contents
print(____)

# Reset the index, dropping its contents
print(____)

#_____________________________________#
#Solution

# Look at temperatures
print(temperatures)

# Index temperatures by city
temperatures_ind = temperatures.set_index('city')

# Look at temperatures_ind
print(temperatures_ind)

# Reset the index, keeping its contents
print(temperatures_ind.reset_index())

# Reset the index, dropping its contents
print(temperatures_ind.reset_index(drop=True))
```

Exercise2

Subsetting with .loc[]

The killer feature for indexes is .loc[]: a subsetting method that accepts index values. When you pass it a single argument, it will take a subset of rows.

The code for subsetting using .loc[] can be easier to read than standard square bracket subsetting, which can make your code less burdensome to maintain.

pandas is loaded as pd. temperatures and temperatures_ind are available; the latter is indexed by city.

Instructions

- Create a list called cities that contains "Moscow" and "Saint Petersburg".
- Use [] subsetting to filter temperatures for rows where the city column takes a value in the cities list.
- Use .loc[] subsetting to filter temperatures_ind for rows where the city is in the cities list.

In [ ]:

```
# Make a list of cities to subset on
cities = ["____", "____"]

# Subset temperatures using square brackets
print(temperatures[____])

# Subset temperatures_ind using .loc[]
print(temperatures_ind.loc[____])

#_____________________________________#
#Solution

# Make a list of cities to subset on
cities = ["Moscow", "Saint Petersburg"]

# Subset temperatures using square brackets
print(temperatures[temperatures["city"].isin(cities)])

# Subset temperatures_ind using .loc[]
print(temperatures_ind.loc[cities])
```

Exercise3

Setting multi-level indexes

Indexes can also be made out of multiple columns, forming a multi-level index (sometimes called a hierarchical index). There is a trade-off to using these.

The benefit is that multi-level indexes make it more natural to reason about nested categorical variables. For example, in a clinical trial, you might have control and treatment groups. Then each test subject belongs to one or another group, and we can say that a test subject is nested inside the treatment group. Similarly, in the temperature dataset, the city is located in the country, so we can say a city is nested inside the country.

The main downside is that the code for manipulating indexes is different from the code for manipulating columns, so you have to learn two syntaxes and keep track of how your data is represented.

pandas is loaded as pd. temperatures is available.

Instructions

- Set the index of temperatures to the `"country"` and `"city"` columns, and assign this to `temperatures_ind`.
- Specify two country/city pairs to keep: `"Brazil"/"Rio De Janeiro"` and `"Pakistan"/"Lahore"`, assigning to `rows_to_keep`.
- Print and subset temperatures_ind for rows_to_keep using .loc[].

In [ ]:

```
# Index temperatures by country & city
temperatures_ind = ____

# List of tuples: Brazil, Rio De Janeiro & Pakistan, Lahore
rows_to_keep = [____]

# Subset for rows to keep
print(temperatures_ind.____)

#_____________________________________#
#Solution

# Index temperatures by country & city
temperatures_ind = temperatures.set_index(['country','city'])

# List of tuples: Brazil, Rio De Janeiro & Pakistan, Lahore
rows_to_keep = [('Brazil','Rio De Janeiro'),('Pakistan','Lahore')]

# Subset for rows to keep
print(temperatures_ind.loc[rows_to_keep])
```

Exercise4

Sorting by index values

Previously, you changed the order of the rows in a DataFrame by calling `.sort_values()`. It's also useful to be able to sort by elements in the index. For this, you need to use `.sort_index()`.

pandas is loaded as pd. temperatures_ind has a multi-level index of country and city, and is available.

Instructions

- Sort temperatures_ind by the index values.
- Sort temperatures_ind by the index values at the "city" level.
- Sort temperatures_ind by ascending country then descending city.

In [ ]:

```
# Sort temperatures_ind by index values
print(____)

# Sort temperatures_ind by index values at the city level
print(____)

# Sort temperatures_ind by country then descending city
print(____)

#_____________________________________#
#Solution

# Sort temperatures_ind by index values
print(temperatures_ind.sort_index())

# Sort temperatures_ind by index values at the city level
print(temperatures_ind.sort_index(level=['city'],ascending=[True]))

# Sort temperatures_ind by country then descending city
print(temperatures_ind.sort_index(level=['country','city'],ascending=[True,False]))
```

Exercise5

Slicing index values

Slicing lets you select consecutive elements of an object using first:last syntax. DataFrames can be sliced by index values or by row/column number; we'll start with the first case. This involves slicing inside the `.loc[]` method.

Compared to slicing lists, there are a few things to remember.

- You can only slice an index if the index is sorted (using .sort_index()).
- To slice at the outer level, first and last can be strings.
- To slice at inner levels, first and last should be tuples.
- If you pass a single slice to .loc[], it will slice the rows.

pandas is loaded as pd. temperatures_ind has country and city in the index, and is available.

Instructions

- Sort the index of temperatures_ind.
- Use slicing with `.loc[]` to get these subsets:
- from Pakistan to Russia.
- from Lahore to Moscow. (This will return nonsense.)
- from Pakistan, Lahore to Russia, Moscow.

In [ ]:

```
# Sort the index of temperatures_ind
temperatures_srt = ____

# Subset rows from Pakistan to Russia
print(____)

# Try to subset rows from Lahore to Moscow
print(____)

# Subset rows from Pakistan, Lahore to Russia, Moscow
print(____)

#_____________________________________#
#Solution

# Sort the index of temperatures_ind
temperatures_srt = temperatures_ind.sort_index()

# Subset rows from Pakistan to Russia
print(temperatures_srt.loc["Pakistan":"Russia"])

# Try to subset rows from Lahore to Moscow
print(temperatures_srt.loc["Lahore":"Moscow"])

# Subset rows from Pakistan, Lahore to Russia, Moscow
print(temperatures_srt.loc[("Pakistan", "Lahore"):("Russia", "Moscow")])
```

Exercise6

Slicing in both directions

You've seen slicing DataFrames by rows and by columns, but since DataFrames are two-dimensional objects, it is often natural to slice both dimensions at once. That is, by passing two arguments to `.loc[]`, you can subset by rows and columns in one go.

pandas is loaded as pd. temperatures_srt is indexed by country and city, has a sorted index, and is available.

Instructions

- Use `.loc[]` slicing to subset rows from India, Hyderabad to Iraq, Baghdad.
- Use `.loc[]` slicing to subset columns from date to `avg_temp_c`.
- Slice in both directions at once from Hyderabad to Baghdad, and date to `avg_temp_` c.

In [ ]:

```
# Subset rows from India, Hyderabad to Iraq, Baghdad
print(____)

# Subset columns from date to avg_temp_c
print(____)

# Subset in both directions at once
print(____)

#_____________________________________#
#Solution

# Subset rows from India, Hyderabad to Iraq, Baghdad
print(temperatures_srt.loc[("India", "Hyderabad"):("Iraq", "Baghdad")])
# Subset columns from date to avg_temp_c
print(temperatures_srt.loc[:,'date':'avg_temp_c'])

# Subset in both directions at once
print(temperatures_srt.loc[("India", "Hyderabad"):("Iraq", "Baghdad"),'date':'avg_temp_c'])
```

Exercise7

Slicing time series

Slicing is particularly useful for time series since it's a common thing to want to filter for data within a date range. Add the date column to the index, then use `.loc[]` to perform the subsetting. The important thing to remember is to keep your dates in ISO 8601 format, that is, `"yyyy-mm-dd"` for year-month-day, `"yyyy-mm"` for year-month, and "yyyy" for year.

Recall from Chapter 1 that you can combine multiple Boolean conditions using logical operators, such as `&`. To do so in one line of code, you'll need to add parentheses `()` around each condition.

pandas is loaded as pd and temperatures, with no index, is available.

Instructions

- Use Boolean conditions, not `.isin()` or `.loc[]`, and the full date `"yyyy-mm-dd"`, to subset temperatures for rows in 2010 and 2011 and print the results.
- Set the index to the date column and sort it.
- Use `.loc[]` to subset `temperatures_ind` for rows in 2010 and 2011.
- Use `.loc[]` to subset `temperatures_ind` for rows from Aug 2010 to Feb 2011.

In [ ]:

```
# Use Boolean conditions to subset temperatures for rows in 2010 and 2011
temperatures_bool = ____[(____ >= ____) & (____ <= ____)]
print(temperatures_bool)

# Set date as the index and sort the index
temperatures_ind = temperatures.____.____

# Use .loc[] to subset temperatures_ind for rows in 2010 and 2011
print(____)

# Use .loc[] to subset temperatures_ind for rows from Aug 2010 to Feb 2011
print(____)

#_____________________________________#
#Solution

# Use Boolean conditions to subset temperatures for rows in 2010 and 2011
temperatures_bool = temperatures[(temperatures["date"] >= "2010-01-01") & (temperatures["da
print(temperatures_bool)

# Set date as the index and sort the index
temperatures_ind = temperatures.set_index("date").sort_index()

# Use .loc[] to subset temperatures_ind for rows in 2010 and 2011
print(temperatures_ind.loc["2010":"2011"])

# Use .loc[] to subset temperatures_ind for rows from Aug 2010 to Feb 2011
print(temperatures_ind.loc["2010-08":"2011-02"])
```

Exercise8

Subsetting by row/column number

The most common ways to subset rows are the ways we've previously discussed: using a Boolean condition or by index labels. However, it is also occasionally useful to pass row numbers.

This is done using `.iloc[]`, and like `.loc[]`, it can take two arguments to let you subset by rows and columns.

pandas is loaded as pd. temperatures (without an index) is available.

Instructions

- Use `.iloc[]` on temperatures to take subsets.
- Get the 23rd row, 2nd column (index positions 22 and 1).
- Get the first 5 rows (index positions 0 to 5).
- Get all rows, columns 3 and 4 (index positions 2 to 4).
- Get the first 5 rows, columns 3 and 4.

In [ ]:

```
# Get 23rd row, 2nd column (index 22, 1)
print(____)

# Use slicing to get the first 5 rows
print(____)

# Use slicing to get columns 3 to 4
print(____)

# Use slicing in both directions at once
print(____)

#_____________________________________#
#Solution

# Get 23rd row, 2nd column (index 22, 1)
print(temperatures.iloc[23,2])

# Use slicing to get the first 5 rows
print(temperatures.iloc[:5])

# Use slicing to get columns 3 to 4
print(temperatures.iloc[:,2:4])

# Use slicing in both directions at once
print(temperatures.iloc[:5,2:4])
```

Exercise9

Pivot temperature by city and year

It's interesting to see how temperatures for each city change over time—looking at every month results in a big table, which can be tricky to reason about. Instead, let's look at how temperatures change by year.

You can access the components of a date (year, month and day) using code of the form `dataframe["column"].dt.component` . For example, the month component is `dataframe["column"].dt.month` , and the year component is `dataframe["column"].dt.year` .

Once you have the year column, you can create a pivot table with the data aggregated by city and year, which you'll explore in the coming exercises.

pandas is loaded as pd. temperatures is available.

Instructions

- Add a year column to temperatures, from the year component of the date column.
- Make a pivot table of the `avg_temp_c` column, with country and city as rows, and year as columns.Assign to `temp_by_country_city_vs_year` , and look at the result.

In [ ]:

```
# Add a year column to temperatures
____

# Pivot avg_temp_c by country and city vs year
temp_by_country_city_vs_year = ____

# See the result
print(temp_by_country_city_vs_year)

#_____________________________________#
#Solution

# Add a year column to temperatures
temperatures["year"] = temperatures["date"].dt.year

# Pivot avg_temp_c by country and city vs year
temp_by_country_city_vs_year = temperatures.pivot_table("avg_temp_c", index = ["country", "

# See the result
print(temp_by_country_city_vs_year)
```

Exercise10

Subsetting pivot tables

A pivot table is just a DataFrame with sorted indexes, so the techniques you have learned already can be used to subset them. In particular, the `.loc[]` + slicing combination is often helpful.

pandas is loaded as pd. `temp_by_country_city_vs_year` is available.

Instructions

- Use .loc[] on temp_by_country_city_vs_year to take subsets.
- From Egypt to India.
- From Egypt, Cairo to India, Delhi.
- From Egypt, Cairo to India, Delhi, and 2005 to 2010.

In [ ]:

```
# Subset for Egypt to India
____

# Subset for Egypt, Cairo to India, Delhi
____

# Subset in both directions at once
____

#_____________________________________#
#Solution

# Subset for Egypt to India
temp_by_country_city_vs_year.loc["Egypt":"India"]

# Subset for Egypt, Cairo to India, Delhi
temp_by_country_city_vs_year.loc[("Egypt", "Cairo"):("India", "Delhi")]

# Subset in both directions at once
temp_by_country_city_vs_year.loc[("Egypt", "Cairo"):("India", "Delhi"), "2005":"2010"]
```

Exercise11

Calculating on a pivot table

Pivot tables are filled with summary statistics, but they are only a first step to finding something insightful. Often you'll need to perform further calculations on them. A common thing to do is to find the rows or columns where the highest or lowest value occurs.

Recall from Chapter 1 that you can easily subset a Series or DataFrame to find rows of interest using a logical condition inside of square brackets. For example: `series[series > value]` .

pandas is loaded as pd and the DataFrame `temp_by_country_city_vs_year is available` .

Instructions

- Calculate the mean temperature for each year, assigning to mean_temp_by_year.
- Filter mean_temp_by_year for the year that had the highest mean temperature.
- Calculate the mean temperature for each city (across columns), assigning to mean_temp_by_city.
- Filter mean_temp_by_city for the city that had the lowest mean temperature.

In [ ]:

```
# Get the worldwide mean temp by year
mean_temp_by_year = temp_by_country_city_vs_year.____

# Filter for the year that had the highest mean temp
print(mean_temp_by_year[____])

# Get the mean temp by city
mean_temp_by_city = temp_by_country_city_vs_year.____

# Filter for the city that had the lowest mean temp
print(mean_temp_by_city[____])

#_____________________________________#
#Solution

# Get the worldwide mean temp by year
mean_temp_by_year = temp_by_country_city_vs_year.mean()

# Filter for the year that had the highest mean temp
print(mean_temp_by_year[mean_temp_by_year == mean_temp_by_year.max()])

# Get the mean temp by city
mean_temp_by_city = temp_by_country_city_vs_year.mean(axis="columns")

# Filter for the city that had the lowest mean temp
print(mean_temp_by_city[mean_temp_by_city == mean_temp_by_city.min()])
```