jupyter notebook

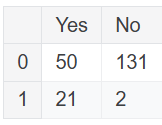
import pandas as pd

**Creating data**

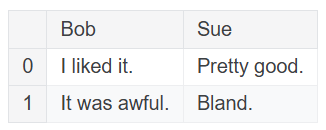
There are two core objects in pandas: the **DataFrame** and the **Series**.

A DataFrame is a table. It contains an array of individual *entries*, each of which has a certain *value*. Each entry corresponds to a row (or *record*) and a *column*.

pd.DataFrame({'Yes': [50, 21], 'No': [131, 2]})



pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], 'Sue': ['Pretty good.', 'Bland.']})



The list of row labels used in a DataFrame is known as an **Index**

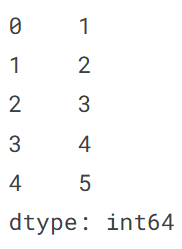
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], 'Sue': ['Pretty good.', 'Bland.']},

index=['Product A', 'Product B'])



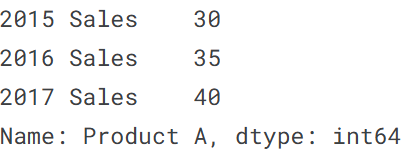
A Series, by contrast, is a sequence of data values

pd.Series([1, 2, 3, 4, 5])



A Series is, in essence, a single column of a DataFrame

pd.Series([30, 35, 40], index=['2015 Sales', '2016 Sales', '2017 Sales'], name='Product A')



**Reading data files**

wine\_reviews = pd.read\_csv("../input/wine-reviews/winemag-data-130k-v2.csv")

We can use the shape attribute to check how large the resulting DataFrame is:

wine\_reviews.shape

A number on a white background

AI-generated content may be incorrect.

We can examine the contents of the resultant DataFrame using the head() command, which grabs the first five rows:

wine\_reviews.head()

A screenshot of a computer

AI-generated content may be incorrect.

To make pandas use that column for the index (instead of creating a new one from scratch), we can specify an index\_col.

wine\_reviews = pd.read\_csv("../input/wine-reviews/winemag-data-130k-v2.csv", index\_col=0)

wine\_reviews.head()

A screenshot of a computer

AI-generated content may be incorrect.

animals = pd.DataFrame({'Cows': [12, 20], 'Goats': [22, 19]}, index=['Year 1', 'Year 2'])

animals

In the cell below, write code to save this DataFrame to disk as a csv file with the name cows\_and\_goats.csv

animals.to\_csv("cows\_and\_goats.csv")

import pandas as pd

1

A screenshot of a computer

AI-generated content may be incorrect.

fruits = pd.DataFrame({'apple':[30],'bananas':[21]})

2

A screenshot of a computer

AI-generated content may be incorrect.

Fruit\_sales = pd.DataFrame({'apple':[35,41],'bananas':[21,34]},index = ['2017 Sales','2018 Sales'])

3

A screenshot of a recipe

AI-generated content may be incorrect.

ingredients = pd.Series(['4 cups','1 cup','2 large','1 can'],index=['Flour','Milk','Eggs','Spam'],name='Dinner')

A screenshot of a computer

AI-generated content may be incorrect.

reviews = pd.read\_csv('D:/Git/Kagle\_File/winemag-data\_first150k.csv',index\_col=0)

A screenshot of a computer

AI-generated content may be incorrect.

animals = pd.DataFrame({'Cows': [12, 20], 'Goats': [22, 19]}, index=['Year 1', 'Year 2'])

animals

In the cell below, write code to save this DataFrame to disk as a csv file with the name cows\_and\_goats.csv

animals.to\_csv("cows\_and\_goats.csv")

**Indexing, Selecting & Assigning**

In Python, we can access the property of an object by accessing it as an attribute. A book object, for example, might have a title property, which we can access by calling book.title. Columns in a pandas DataFrame work in much the same way.

Hence to access the country property of reviews we can use:

reviews.country

A white background with black text

AI-generated content may be incorrect.

If we have a Python dictionary, we can access its values using the indexing ([]) operator. We can do the same with columns in a DataFrame:

reviews['country']

A white background with black text

AI-generated content may be incorrect.

to drill down to a single specific value, we need only use the indexing operator [] once more:

reviews['country'][0]

A black and white text

AI-generated content may be incorrect.

**Indexing in pandas**

**Index-based selection**

Pandas indexing works in one of two paradigms. The first is index-based selection: selecting data based on its numerical position in the data. iloc follows this paradigm.

To select the first row of data in a DataFrame, we may use the following:

df.iloc[<row\_selection>, <column\_selection>]

Both row\_selection and column\_selection can be:

* A single integer (e.g., 3)
* A list of integers (e.g., [1, 4, 5])
* A slice (e.g., 1:5)
* A boolean array (e.g., [True, False, True, ...])

iloc[:3, 0] means:

* Select **rows 0 to 2** (because :3 slices up to but not including index 3)
* Select **column at index 0**

iloc[1:3, 0] means:

* Select **rows 1 and 2** (slice from index 1 up to, but not including, index 3)
* Select the **column at index 0**

reviews.iloc[[0, 1, 2], 0] means:

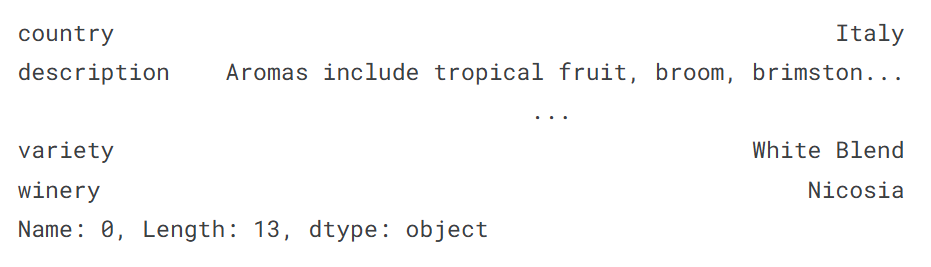
* Select **rows 0, 1, and 2** (using a list of specific row indices)
* Select the **column at index 0**

iloc[-5:] selects the **last 5 rows** of a DataFrame.

**Breakdown:**

* -5: is a slice that starts 5 rows from the end and goes to the end.
* Because no column index is specified, it includes **all columns**.

reviews.iloc[0]



Both loc and iloc are row-first, column-second. This is the opposite of what we do in native Python, which is column-first, row-second.

This means that it's marginally easier to retrieve rows, and marginally harder to get retrieve columns. To get a column with iloc, we can do the following:

reviews.iloc[:, 0]

A white background with black text

AI-generated content may be incorrect.