Loading Data into a pandas DataFrame

The GOOG.csv and fake_company.csv are available to download at the bottom of this page. If it doesn't get downloaded upon clicking, try right-click and choose the "Save As..." option.

In machine learning you will most likely use databases from many sources to train your learning algorithms. Pandas allows us to load databases of different formats into DataFrames. One of the most popular data formats used to store databases is csv. CSV stands for *Comma Separated Values* and offers a simple format to store data. We can load CSV files into Pandas DataFrames using the pd.read_csv() function. Let's load Google stock data into a Pandas DataFrame. The GOOG.csv file contains Google stock data from 8/19/2004 till 10/13/2017 taken from Yahoo Finance.

Example 1. Load the data from a .csv file.

```
# We load Google stock data in a DataFrame

Google_stock = pd.read_csv('./GOOG.csv')


# We print some information about Google_stock

print('Google_stock is of type:', type(Google_stock))

print('Google_stock has shape:', Google_stock.shape)
```

```
Google_stock is of type: class 'pandas.core.frame.DataFrame'
Google_stock has shape: (3313, 7)
```

We see that we have loaded the GOOG.csv file into a Pandas DataFrame and it consists of 3,313 rows and 7 columns. Now let's look at the stock data

Example 2. Look at the first few rows of the DataFrame

```
Google_stock
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	49.676899	51.693783	47.669952	49.845802	49.845802	44994500
1	2004-08-20	50.178635	54.187561	49.925285	53.805050	53.805050	23005800
2	2004-08-23	55.017166	56.373344	54.172661	54.346527	54.346527	18393200
... ...							
3311	2017-10-12	987.450012	994.119995	985.000000	987.830017	987.830017	1262400

3312	2017-10-13	992.000000	997.210022	989.000000	989.679993	989.679993	1157700

3313 rows × 7 columns

We see that it is quite a large dataset and that Pandas has automatically assigned numerical row indices to the DataFrame. Pandas also used the labels that appear in the data in the CSV file to assign the column labels.

When dealing with large datasets like this one, it is often useful just to take a look at the first few rows of data instead of the whole dataset. We can take a look at the first 5 rows of data using the .head() method, as shown below

Example 3. Look at the first 5 rows of the DataFrame

```
Google_stock.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	49.676899	51.693783	47.669952	49.845802	49.845802	44994500
1	2004-08-20	50.178635	54.187561	49.925285	53.805050	53.805050	23005800
2	2004-08-23	55.017166	56.373344	54.172661	54.346527	54.346527	18393200
3	2004-08-24	55.260582	55.439419	51.450363	52.096165	52.096165	15361800
4	2004-08-25	52.140873	53.651051	51.604362	52.657513	52.657513	9257400

We can also take a look at the last 5 rows of data by using the .tail() method:

Example 4. Look at the last 5 rows of the DataFrame

```
Google_stock.tail()
```

	Date	Open	High	Low	Close	Adj Close	Volume
3308	2017-10-09	980.000000	985.424988	976.109985	977.000000	977.000000	891400
3309	2017-10-10	980.000000	981.570007	966.080017	972.599976	972.599976	968400
3310	2017-10-11	973.719971	990.710022	972.250000	989.250000	989.250000	1693300
3311	2017-10-12	987.450012	994.119995	985.000000	987.830017	987.830017	1262400
3312	2017-10-13	992.000000	997.210022	989.000000	989.679993	989.679993	1157700

We can also optionally use .head(N) or .tail(N) to display the first and last N rows of data, respectively.

Let's do a quick check to see whether we have any NaN values in our dataset. To do this, we will use the .isnull() method followed by the .any() method to check whether any of the columns contain NaN values.

Example 5. Check if any column contains a NaN. Returns a boolean for each column label.

```
Google_stock.isnull().any()
```

```
Date         False
Open         False
High         False
Low          False
Close        False
Adj Close    False
Volume       False
dtype: bool
```

We see that we have no NaN values.

When dealing with large datasets, it is often useful to get statistical information from them. Pandas provides the .describe() method to get descriptive statistics on each column of the DataFrame. Let's see how this works:

Example 6. See the descriptive statistics of the DataFrame

```
# We get descriptive statistics on our stock data

Google_stock.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	3313.000000	3313.000000	3313.000000	3313.000000	3313.000000	3.313000e+03
mean	380.186092	383.493740	376.519309	380.072458	380.072458	8.038476e+06
std	223.818650	224.974534	222.473232	223.853780	223.853780	8.399521e+06
min	49.274517	50.541279	47.669952	49.681866	49.681866	7.900000e+03
25%	226.556473	228.394516	224.003082	226.407440	226.407440	2.584900e+06
50%	293.312286	295.433502	289.929291	293.029114	293.029114	5.281300e+06
75%	536.650024	540.000000	532.409973	536.690002	536.690002	1.065370e+07
max	992.000000	997.210022	989.000000	989.679993	989.679993	8.276810e+07

If desired, we can apply the .describe() method on a single column as shown below:

Example 7. See the descriptive statistics of one of the columns of the DataFrame

```
# We get descriptive statistics on a single column of our DataFrame

Google_stock['Adj Close'].describe()
```

```
count    3313.000000
mean      380.072458
std       223.853780
min        49.681866
25%       226.407440
50%       293.029114
75%       536.690002
max       989.679993
Name: Adj Close, dtype: float64
```

Similarly, you can also look at one statistic by using one of the many statistical functions Pandas provides. Let's look at some examples:

Example 8. Statistical operations - Min, Max, and Mean

```
# We print information about our DataFrame

print()

print('Maximum values of each column:\n', Google_stock.max())

print()

print('Minimum Close value:', Google_stock['Close'].min())

print()

print('Average value of each column:\n', Google_stock.mean())
```

```
Maximum values of each column:
Date         2017-10-13
Open                992
High             997.21
Low                 989
Close            989.68
Adj Close        989.68
Volume         82768100
dtype: object

Minimum Close value: 49.681866

Average value of each column:
Open         3.801861e+02
High         3.834937e+02
Low          3.765193e+02
```

Close 3.800725e+02
Adj Close 3.800725e+02
Volume 8.038476e+06
dtype: float64

Another important statistical measure is data correlation. Data correlation can tell us, for example, if the data in different columns are correlated. We can use the .corr() method to get the correlation between different columns, as shown below:

Example 9. Statistical operation - Correlation

```
# We display the correlation between columns

Google_stock.corr()
```

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999904	0.999845	0.999745	0.999745	-0.564258
High	0.999904	1.000000	0.999834	0.999868	0.999868	-0.562749
Low	0.999845	0.999834	1.000000	0.999899	0.999899	-0.567007
Close	0.999745	0.999868	0.999899	1.000000	1.000000	-0.564967
Adj Close	0.999745	0.999868	0.999899	1.000000	1.000000	-0.564967
Volume	-0.564258	-0.562749	-0.567007	-0.564967	-0.564967	1.000000

A correlation value of 1 tells us there is a high correlation and a correlation of 0 tells us that the data is not correlated at all.

groupby() method

We will end this Introduction to Pandas by taking a look at the .groupby() method. The .groupby() method allows us to group data in different ways. Let's see how we can group data to get different types of information. For the next examples, we are going to load fake data about a fictitious company.

```
# We load fake Company data in a DataFrame

data = pd.read_csv('./fake_company.csv')


data
```

	Year	Name	Department	Age	Salary
0	1990	Alice	HR	25	50000
1	1990	Bob	RD	30	48000
2	1990	Charlie	Admin	45	55000

3	1991	Dakota	HR	26	52000
4	1991	Elsa	RD	31	50000
5	1991	Frank	Admin	46	60000
6	1992	Grace	Admin	27	60000
7	1992	Hoffman	RD	32	52000
8	1992	Inaar	Admin	28	62000

We see that the data contains information for the year 1990 through 1992. For each year we see name of the employees, the department they work for, their age, and their annual salary. Now, let's use the .groupby() method to get information.

Example 10. Demonstrate groupby() and sum() method

Let's calculate how much money the company spent on salaries each year. To do this, we will group the data by *Year* using the .groupby() method and then we will add up the salaries of all the employees by using the .sum() method.

```
# We display the total amount of money spent in salaries each year

data.groupby(['Year'])['Salary'].sum()
```

```
Year
1990    153000
1991    162000
1992    174000
Name: Salary, dtype: int64
```

We see that the company spent a total of 153,000 dollars in 1990, 162,000 in 1991, and 174,000 in 1992.

Example 11. Demonstrate groupby() and mean() method

Now, let's suppose I want to know what was the average salary for each year. In this case, we will group the data by *Year* using the .groupby() method, just as we did before, and then we use the .mean() method to get the average salary. Let's see how this works

```
# We display the average salary per year

data.groupby(['Year'])['Salary'].mean()
```

```
Year
1990    51000
1991    54000
1992    58000
Name: Salary, dtype: int64
```

We see that the average salary in 1990 was 51,000 dollars, 54,000 in 1991, and 58,000 in 1992.

Example 12. Demonstrate groupby() on single column

Now let's see how much did each employee gets paid in those three years. In this case, we will group the data by *Name* using the .groupby() method and then we will add up the salaries for each year. Let's see the result

```
# We display the total salary each employee received in all the years they worked for the company

data.groupby(['Name'])['Salary'].sum()
```

```
Name
Alice       162000
Bob         150000
Charlie     177000
Name: Salary, dtype: int64
```

We see that Alice received a total of 162,000 dollars in the three years she worked for the company, Bob received 150,000, and Charlie received 177,000.

Example 13. Demonstrate groupby() on two columns

Now let's see what was the salary distribution per department per year. In this case, we will group the data by *Year* and by *Department* using the .groupby() method and then we will add up the salaries for each department. Let's see the result

```
# We display the salary distribution per department per year.

data.groupby(['Year', 'Department'])['Salary'].sum()
```

```
Year  Department
1990  Admin        55000
      HR           50000
      RD           48000
1991  Admin        60000
      HR           52000
      RD           50000
1992  Admin       122000
      RD           52000
Name: Salary, dtype: int64
```

We see that in 1990 the Admin department paid 55,000 dollars in salaries,the HR department paid 50,000, and the RD department 48,0000. While in 1992 the Admin department paid 122,000 dollars in salaries and the RD department paid 52,000.

Recommended Practice

We recommend you practice the examples available at the 10 minutes to pandas as a conclusive tutorial.

Supporting Materials

- GOOG.csv
- fake_company.csv