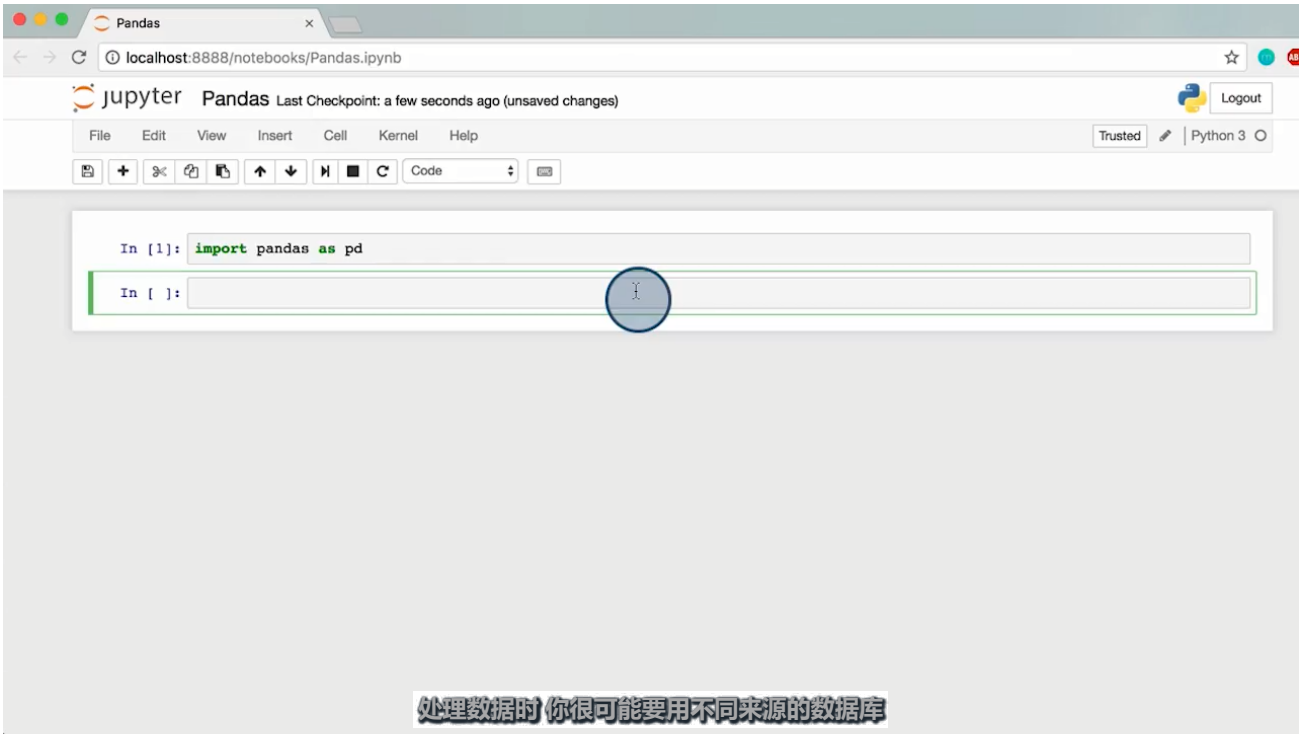
# 将数据加载到 Pandas DataFrame 中




00:00 / 04:06 1x CC

在机器学习中，你很有可能会使用来自很多来源的数据库训练学习算法。Pandas 使我们能够将不同格式的数据库加载到 DataFrame 中。用于存储数据库的最热门数据格式是 csv。CSV 是指*逗号分隔值*，是一种简单的数据存储格式。我们可以使用 `pd.read_csv()` 函数将 CSV 文件加载到 Pandas DataFrame 中。我们将 Google 股票数据加载到一个 Pandas DataFrame 中。GOOG.csv 文件包含从雅虎金融那获取的 2004 年 8 月 19 日至 2017 年 10 月 13 日 Google 股票数据。

```
# 我们将 Google 股票数据加载到 DataFrame 中
Google_stock = pd.read_csv('./GOOG.csv')

# 我们输出关于 Google_stock 的一些信息
print('Google_stock is of type:', type(Google_stock))
print('Google_stock has shape:', Google_stock.shape)
```

可以看出，我们将 GOOG.csv 文件加载到了 Pandas DataFrame 中，其中包含 3,313 行和 7 列数据。现在我们来看看股票数据

```
Google_stock
```

| | Date | Open | High | Low | Clos |
|---|---|---|---|---|---|
| **0** | 2004-08-19 | 49.676899 | 51.693783 | 47.669952 | 49.845 |
| **1** | 2004-08-20 | 50.178635 | 54.187561 | 49.925285 | 53.805 |
| **2** | 2004-08-23 | 55.017166 | 56.373344 | 54.172661 | 54.346 |
| ... | ... | | | | |
| **3311** | 2017-10-12 | 987.450012 | 994.119995 | 985.000000 | 987.83( |
| **3312** | 2017-10-13 | 992.000000 | 997.210022 | 989.000000 | 989.67! |

*3313 rows × 7 columns*

可以看出，这是一个非常庞大的数据集，Pandas 自动为该 DataFrame 分配了数字行索引。Pandas 还使用出现在 CSV 文件中的标签为列分配标签。

在处理这样的大型数据集时，通常有必要直接查看前几行数据，而不是整个数据集。我们可以使用 `.head()` 方法查看前 5 行数据，如下所示

```
Google_stock.head()
```

| | | | | | | |
|---|---|---|---|---|---|---|
| *0* | *2004-08-19* | *49.676899* | *51.693783* | *47.669952* | *49.845802* | |
| *1* | *2004-08-20* | *50.178635* | *54.187561* | *49.925285* | *53.805050* | |
| *2* | *2004-08-23* | *55.017166* | *56.373344* | *54.172661* | *54.346527* | |
| *3* | *2004-08-24* | *55.260582* | *55.439419* | *51.450363* | *52.096165* | |
| *4* | *2004-08-25* | *52.140873* | *53.651051* | *51.604362* | *52.657513* | |

我们还可以使用 `.tail()` 方法查看最后 5 行数据：

```
Google_stock.tail()
```

| | Date | Open | High | Low | Clos |
|---|---|---|---|---|---|
| ***3308*** | *2017-10-09* | *980.000000* | *985.424988* | *976.109985* | *977.00* |
| ***3309*** | *2017-10-10* | *980.000000* | *981.570007* | *966.080017* | *972.59* |
| ***3310*** | *2017-10-11* | *973.719971* | *990.710022* | *972.250000* | *989.25* |
| ***3311*** | *2017-10-12* | *987.450012* | *994.119995* | *985.000000* | *987.83* |
| ***3312*** | *2017-10-13* | *992.000000* | *997.210022* | *989.000000* | *989.67* |

我们还可以选择使用 `.head(N)` 或 `.tail(N)` 分别显示前 `N` 行和后 `N` 行数据。

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

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

可以看出没有任何 `NaN` 值。

在处理大型数据集时，通常有必要获取关于数据集的统计信息。通过使用 Pandas 的 `.describe()` 方法，可以获取关于 DataFrame 每列的描述性统计信息。我们来看看代码编写方式：

```
# We get descriptive statistics on our stock data
Google_stock.describe()
```

| | Open | High | Low | Close |
|---|---|---|---|---|
| **count** | 3313.000000 | 3313.000000 | 3313.000000 | 3313.000000 |
| **mean** | 380.186092 | 383.493740 | 376.519309 | 380.072458 |
| **std** | 223.818650 | 224.974534 | 222.473232 | 223.853780 |
| **min** | 49.274517 | 50.541279 | 47.669952 | 49.681866 |
| **25%** | 226.556473 | 228.394516 | 224.003082 | 226.407440 |
| **50%** | 293.312286 | 295.433502 | 289.929291 | 293.029114 |
| **75%** | 536.650024 | 540.000000 | 532.409973 | 536.690002 |

```
max        992.000000    997.210022    989.000000    989.679993
```

如果有必要，我们可以对单列应用 `.describe()` 方法，如下所示：

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

同样，你可以使用 Pandas 提供的很多统计学函数查看某个统计信息。我们来看一些示例：

```
# 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
```

```
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
```

另一个重要统计学衡量指标是数据相关性。数据相关性可以告诉我们不同列的数据是否有关联。我们可以使用 `.corr()` 方法获取不同列之间的关联性，如下所示：

```
# We display the correlation between columns
Google_stock.corr()
```

| | Open | High | Low | Close | Adj C |
|---|---|---|---|---|---|
| **Open** | 1.000000 | 0.999904 | 0.999845 | 0.999745 | 0.999 |
| **High** | 0.999904 | 1.000000 | 0.999834 | 0.999868 | 0.999 |
| **Low** | 0.999845 | 0.999834 | 1.000000 | 0.999899 | 0.999 |
| **Close** | 0.999745 | 0.999868 | 0.999899 | 1.000000 | 1.000 |
| **Adj Close** | 0.999745 | 0.999868 | 0.999899 | 1.000000 | 1.000 |
| **Volume** | -0.564258 | -0.562749 | -0.567007 | -0.564967 | -0.564 |

在这个 Pandas 入门课程的最后，我们将讲解 `.groupby()` 方法。`.groupby()` 方法使我们能够以不同的方式对数据分组。我们来看看如何分组数据，以获得不同类型的信息。在下面的示例中，我们将加载关于虚拟公司的虚拟数据。

```
# 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 | Alice | HR | 26 | 52000 |
| 4 | 1991 | Bob | RD | 31 | 50000 |
| 5 | 1991 | Charlie | Admin | 46 | 60000 |
| 6 | 1992 | Alice | Admin | 27 | 60000 |
| 7 | 1992 | Bob | RD | 32 | 52000 |
| 8 | 1992 | Charlie | Admin | 28 | 62000 |

可以看出，上述数据包含从 1990 年到 1992 年的信息。对于每一年，我们都能看到员工姓名、所在的部门、年龄和年薪。现在，我们使用 `.groupby()` 方法获取信息。

我们来计算公司每年在员工薪资上花费的数额。为此，我们将使用 `.groupby()` 方法按*年份*对数据分组，然后使用 `.sum()` 方法将所有员工的薪资相加。

```
# We display the total amount of money spent in salaries each yea
data.groupby(['Year'])['Salary'].sum()
```

> *1991 162000*
> *1992 174000*
> *Name: Salary, dtype: int64*

可以看出，该公司在 1990 年的薪资花费总额为 153,000 美元，在 1991 年为 162,000 美元，在 1992 年为 174,000 美元。

现在假设我们想知道每年的平均薪资是多少。为此，我们将使用 `.groupby()` 方法按*年份*对数据分组，就像之前一样，然后使用 `.mean()` 方法获取平均薪资。我们来看看代码编写方式

```
# We display the average salary per year
data.groupby(['Year'])['Salary'].mean()
```

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

可以看出，1990 年的平均薪资为 51,000 美元，1991 年为 54,000 美元，1992 年为 58,000 美元。

现在我们来看看在这三年的时间内每位员工都收到多少薪资。在这种情况下，我们将使用 `.groupby()` 方法按照*Name*来对数据分组。之后，我们会把每年的薪资加起来。让我们来看看结果。

```
# We display the total salary each employee received in all the y
data.groupby(['Name'])['Salary'].sum()
```

> *Name*
> *Alice 162000*
> *Bob 150000*

我们看到，Alice在公司工作的三年时间里共收到了162,000美元的薪资，Bob收到了150,000，Charlie收到了177,000。

现在让我们看看每年每个部门的薪资分配状况。在这种情况下，我们将使用`.groupby()`方法按照*Year*和*Department*对数据分组，之后我们会把每个部门的薪资加起来。让我们来看看结果。

```
# 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
```

我们看到，1990年，管理部门支付了55,000美元的薪资，HR部门支付了50,000，研发部门支付了48,000。1992年，管理部门支付了122,000美元的薪资，研发部门支付了52,000。