**فصل 2- بارگیری داده‌ها**

**2.0 مقدمه**

گام نخست در هر اقدام یادگیری ماشین، وارد کردن داده‌‌های خام به سیستم است. داده‌‌های خام ممکن است فایلرخداد[[1]](#footnote-1)، فایل مجموعه داده، یا پایگاه داده باشند. افزون بر این، اغلب می‌‌خواهیم داده‌‌ها را از چندین منبع بازیابی کنیم. دستورالعمل‌‌های فصل حاضر به روش‌‌های بارگیری داده‌‌ از منابع مختلف، از جمله فایل‌‌های *مقادیر جداشده با ویرگول* (CSV) و پایگاه دادۀ SQL می‌‌پردازند. روش‌‌های تولید داده‌‌های شبیه‌سازی‌شده با ویژگی‌‌های مطلوب برای آزمایش را نیز پوشش می‌‌دهیم. در نهایت، در حالی که راه‌های بسیاری برای بارگیری داده‌ها در اکوسیستم پایتون وجود دارند، بر استفاده از مجموعۀ گستردۀ روش‌های کتاب‌خانۀ Pandas برای بارگیری داده‌های خارجی و نیز scikit-learn - نوعی کتاب‌خانۀ یادگیری ماشینی متن‌‌باز در پایتون - برای تولید داده‌‌های شبیه‌سازی‌شده تمرکز خواهیم کرد.

**2.1 بارگیری یک نمونه مجموعه داده‌‌**

**مسئله**

می‌خواهید یک مجموعه دادۀ نمونۀ موجود را بارگیری کنید.

**راه‌حل**

scikit-learn چند مجموعه دادۀ محبوب و رایج را ارائه می‌‌دهد که می‌‌توانید از آنها استفاده کنید:





**بحث**

اغلب، نمی‌خواهیم قبل از بررسی چند نوع الگوریتم‌ یا روش‌ یادگیری ماشین، به بارگیری، تبدیل و پاک‌سازی مجموعه داده‌های واقعی بپردازیم. خوش‌بختانه، scikit-learn برخی از مجموعه داده‌های رایج را ارائه می‌‌دهد که می‌توانیم به سرعت بارگیری کنیم. این مجموعه داده‌‌ها اغلب مجموعه داده‌‌های «اسباب‌بازی»[[2]](#footnote-2) نامیده می‌‌شوند زیرا بسیار کوچک‌تر و پاک‌تر از مجموعه داده‌هایی‌اند که در دنیای واقعی می‌‌بینیم. برخی از مجموعه داده‌‌های نمونۀ محبوب در scikit-learn عبارتند از:

load\_boston

حاوی 503 نمونه مشاهده در مورد قیمت مسکن بوستون است. مجموعه داده خوبی برای بررسی الگوریتم‌‌های رگرسیون است.

load\_iris

حاوی 150 نمونه مشاهده در مورد ویژگی‌هایی از گل زنبق است. مجموعه داده خوبی برای بررسی الگوریتم‌‌های دسته‌بندی است.

load\_digits

حاوی 1797 نمونه مشاهده از تصاویر ارقام دست‌نویس است. مجموعه داده خوبی برای آموزش دسته‌بندی تصاویر است.

**مطالعۀ بیشتر**

* [مجموعه داده‌‌های اسباب‌بازی scikit-learn](https://scikit-learn.org/stable/datasets/toy_dataset.html)
* [مجموعه داده Digit](https://scikit-learn.org/stable/auto_examples/datasets/plot_digits_last_image.html)

**2.2 ایجاد مجموعه‌دادۀ شبیه‌سازی‌شده**

**مسئله**

باید مجموعه داده‌ای را از داده‌‌های شبیه‌سازی‌شده تولید کنید.

**راه‌حل**

scikit-learn روش‌‌های زیادی برای ایجاد داده‌‌های شبیه‌سازی دارد. از این میان، سه روش به‌ طور ویژه مفید هستند.

وقتی می‌‌خواهیم مجموعه داده‌‌ای طراحی کنیم که با رگرسیون خطی استفاده شود، make\_regression روش خوبی است:



اگر می‌‌خواهیم مجموعه داده شبیه‌سازی‌شده‌ای را برای دسته‌بندی ایجاد کنیم، می‌‌توانیم از make\_classification استفاده کنیم:





در نهایت، اگر مجموعه داده‌‌ای را می‌‌خواهیم که با تکنیک‌‌های خوشه‌بندی به‌خوبی کار کند، کتابخانه scikit-learn تابع make\_blobs را ارائه می‌‌دهد:





**بحث**

همان‌گونه که از راه‌حل‌‌ها مشخص است، make\_regression یک ماتریس ویژگی از مقادیر اعشاری و یک بردار هدف از مقادیر اعشاری را برمی‌گرداند، در حالی که make\_classification و make\_blobs یک ماتریس ویژگی از مقادیر اعشاری و یک بردار هدف از اعداد صحیح را برمی‌گرداند که تعلق به یک دسته را نشان می‌‌دهد.

مجموعه داده‌‌های شبیه‌سازی‌شدۀ scikit-learn روش‌‌های گسترده‌‌ای را برای کنترل نوع داده‌‌های تولیدشده ارائه می‌‌دهند. مستندات scikit-learn حاوی توضیحات کاملی از تمام پارامترهاست، اما تعداد معدودی قابل‌توجه هستند.

در make\_regression و make\_classification، n\_informative تعداد ویژگی‌‌هایی را تعیین می‌‌کند که برای تولید بردار هدف استفاده می‌‌شود. اگر n\_informative کمتر از تعداد کل ویژگی‌‌ها (n\_features) باشد، مجموعه داده حاصل حاوی ویژگی‌‌های اضافی خواهد بود که ممکن است از طریق تکنیک‌‌های انتخاب ویژگی شناسایی شوند.

افزون بر این، make\_classification حاوی یک پارامتر weights است که با آن می‌‌توانیم مجموعه داده‌‌ها را با دسته‌های نامتعادل شبیه‌سازی کنیم. مثلاً، weights = [.25, .75] یک مجموعه داده را با 25% از مشاهدات متعلق به یک دسته و 75% از مشاهدات متعلق به دسته دوم برمی‌گرداند.

برای make\_blobs، پارامتر centers تعداد خوشه‌‌های تولیدشده را تعیین می‌‌کند. با استفاده از کتاب‌خانه مصورسازی matplotlib، می‌‌توانیم خوشه‌‌های حاصل از make\_blobs را نشان دهیم:





**مطالعۀ بیشتر**

* [مستندات make\_regression](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_regression.html#sklearn.datasets.make_regression)
* [مستندات make\_classification](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html#sklearn.datasets.make_classification)
* [مستندات make\_blobs](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html#sklearn.datasets.make_blobs)

**2.3 بارگیری فایل CSV**

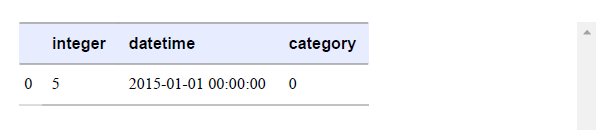
**مسئله**

باید یک فایل CSV را وارد کنید.

**راه‌حل**

از read\_csv کتاب‌خانه pandas برای بارگیری یک فایل CSV محلی یا میزبانی‌شده استفاده کنید:







**بحث**

دو نکته در مورد بارگیری فایل‌‌های CSV وجود دارد. نخست این‌که، قبل از بارگیری، بهتر است نگاهی گذرا به محتوای فایل بیندازید. بهتر است ابتدا ببینیم که ساختار مجموعه داده‌ از قبل چگونه است و چه پارامترهایی را برای بارگیری در فایل باید تنظیم کنیم. دوم، read\_csv بیش از 30 پارامتر دارد و لذا مستندات آن ممکن است دلهره‌آور باشد. خوش‌بختانه، این پارامترها عمدتاً برای مدیریت انواع فرمت‌‌های CSV وجود دارند. مثلاً، فایل‌های CSV طوری نام‌گذاری می‌‌شوند که مقادیر در واقع با ویرگول از هم جدا شوند (مثلاً، یک سطر ممکن است 2,”2015-01-01 00:00:00”, 0 باشد). با این حال، فایل‌های CSV ممکن است از کاراکترهای دیگر به‌عنوان جداکننده استفاده کنند (مثلا تب[[3]](#footnote-3)). پارامتر sep در pandas امکان تعریف جداکننده مورد استفاده در فایل را فراهم می‌‌کند. یکی از مشکلات رایج قالب‌بندی فایل‌های CSV این است که از خط اول فایل برای تعریف سرآیند ستون (مثلاً، integer، datetime، category در راه‌حل ما) استفاده می‌شود، اگرچه همواره این‌گونه نیست. با پارامتر header، می‌‌توانیم مشخص کنیم سطر سرآیند وجود دارد یا کجا وجود دارد. اگر سطر سرآیند وجود نداشته باشد، header=None را تنظیم می‌‌کنیم.

**2.4 بارگیری فایل اکسل**

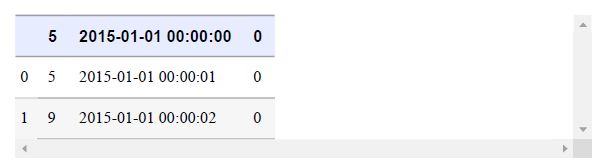
**مسئله**

باید یک صفحه‌گسترده اکسل را وارد کنید.

**راه‌حل**

برای بارگیری صفحه‌گسترده اکسل از read\_excel کتاب‌خانه pandas استفاده کنید:





**بحث**

این راه‌حل مشابه راه‌حل ما برای خواندن فایل‌‌های CSV است. تفاوت اصلی آن پارامتر اضافی، sheetname، است که مشخص می‌‌کند در فایل اکسل کدام صفحه را می‌‌خواهیم بارگذاری کنیم. Sheetname می‌تواند هم یک رشته حاوی نام صفحه و هم اعداد صحیح را بپذیرد که به موقعیت‌های صفحه اشاره می‌کنند (با شروع از نمایه صفر). اگر باید چندین صفحه را بارگیری کنیم، آنها را در یک لیست قرار می‌دهیم. برای مثال، sheetname=[0,1,2, "Monthly Sales"] یک دیکشنری از دیتا فریم‌‌های pandas را برمی‌گرداند که حاوی صفحه‌‌های اول، دوم و سوم و صفحه‌ای با نامMonthly Sales است.

**2.5 بارگیری فایل JSON**

**مسئله**

برای پیش‌پردازش داده‌‌ها باید یک فایل JSON را بارگیری کنید.

**راه‌حل**

کتاب‌خانه pandas تابع read\_json را برای تبدیل یک فایل JSON به یک شیء pandas ارائه می‌‌دهد:





**بحث**

وارد کردن فایل‌های JSON به pandas مشابه دستورالعمل‌های اخیری است که دیده‌ایم. تفاوت اصلی آن در پارامتر orient است، که ساختار فایل JSON را به pandas نشان می‌‌دهد. با این حال، ممکن است کمی آزمایش لازم باشد تا بفهمید کدام آرگومان (split، records، index، columns و values) مناسب است. یکی دیگر از ابزارهای مفیدی که pandas ارائه می‌‌دهد json\_normalize است، که می‌‌تواند به تبدیل داده‌‌های نیمه‌ساختار‌یافته JSON به دیتافریم pandas کمک کند.

**مطالعۀ بیشتر**

* [اسناد json\_normalize](http://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.json_normalize.html)

**بارگذاری‌یک فایل پارکت[[4]](#footnote-4)**

**مسئله**

شما باید‌یک فایل پارکت را بارگذاری کنید.

**راه‌حل**

تابع read\_parquet از کتابخانه‌یpandas به ما این امکان می‌دهد که فایل‌های Parquet را بخوانیم:



|  |  |  |  |
| --- | --- | --- | --- |
|  | دسته بندی | تاریخ و زمان | عدد |
| 0 | 0 | 00:00:00 2015-01-01 | 5 |
| 1 | 0 | 00:00:01 2015-01-01 | 5 |

**بحث**

Parquet‌یک فرمت محبوب ذخیره سازی داده در فضای بزرگ داده است. اغلب با ابزارهای داده‌های بزرگ مانند Hadoop و Spark مورد استفاده قرار می‌گیرد. در حالی که PySpark خارج از تمرکز این کتاب است، به احتمال زیاد شرکت‌هایی که در مقیاس بزرگ فعالیت می‌کنند از‌یک قالب ذخیره‌سازی داده کارآمد مانند پارکت استفاده می‌کنند، و دانستن نحوه خواندن آن در‌یک دیتافریم و دستکاری آن بسیار ارزشمند است.

**همچنین ببینید:**

* [مستندات پارکت آپاچی[[5]](#footnote-5)](https://parquet.apache.org/)

**2.7 بارگیری‌یک فایل Avro**

**مسئله**

شما باید‌یک فایل Avro را در دیتافرِیم pandas بارگذاری کنید.

**راه‌حل**

از تابع read\_avro در کتابخانه‌ی pandavro استفاده کنید:



|  |  |  |  |
| --- | --- | --- | --- |
|  | دسته بندی | تاریخ و زمان | عدد |
| 0 | 0 | 00:00:00 2015-01-01 | 5 |
| 1 | 0 | 00:00:01 2015-01-01 | 5 |

**بحث**

Apache Avro‌یک فرمت داده باینری و منبع باز است که برای ساختار داده بر طرح‌واره‌ها متکی است. در زمان نوشتن، به اندازه پارکت رایج نیست. با این حال، فرمت‌های داده‌های باینری بزرگ مانند Avro، Thrift و Protocol Buffer به دلیل ماهیت کارآمدشان در حال افزایش محبوبیت هستند. اگر با سیستم‌های داده بزرگ کار می‌کنید، احتمالاً در آینده نزدیک با‌یکی از این فرمت‌ها مواجه خواهید شد.

**همچنین ببینید:**

* [مستندات پارکت آپاچی[[6]](#footnote-6)](https://avro.apache.org/docs/1.2.0/)

**2.8 جستجو در پایگاه داده SQLite[[7]](#footnote-7)**

**مسئله**

شما باید داده‌ها را از‌یک پایگاه داده با استفاده از زبان SQL بارگیری کنید.

**راه‌حل**

read\_sql\_query از کتابخانه‌ی Pandas به ما این امکان را می‌دهد که‌یک دستور کوئری[[8]](#footnote-8) SQL در پایگاه داده ایجاد کرده و آن را بارگذاری کنیم:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | نام | نام خانوادگی | سن | امتیاز پیش‌آزمون |
| 0 | Json | Miller | 42 | 4 |
| 1 | Molly | Jaboson | 52 | 24 |

**بحث**

Apache SQL زبانی برای استخراج داده‌ها از پایگاه‌های داده است. در این دستور، ابتدا از create\_engine برای تعریف اتصال به موتور پایگاه داده SQL به نام SQLite استفاده می‌کنیم. در مرحله بعد ما از read\_sql\_query در کتابخانه‌ی Pandas برای کوئری زدن روی پایگاه داده با استفاده از زبان SQL و قرار دادن نتایج در‌یک DataFrame استفاده می‌کنیم.

SQL به خودی خود‌یک زبان است و اگرچه فراتر از محدوده این کتاب است، اما مطمئناً برای هر کسی که می‌خواهد در مورد یادگیری ماشینی بیاموزد، ارزش دارد. کوئری SQL ما، SELECT \* FROM data، از پایگاه داده می‌خواهد که تمام ستون‌های (\*) جدول را به نام عنوان داده به ما بدهد.

توجه داشته باشید که این‌یکی از معدود دستور العمل‌های این کتاب است که بدون کد اضافی اجرا نمی‌شود. به طور خاص، create\_engine('sqlite:///sample.db') فرض می‌کند که‌یک پایگاه داده SQLite از قبل وجود دارد.

* [SQLite](https://www.sqlite.org/index.html)
* [آموزش SQL در وبسایت W3Schools](https://www.w3schools.com/sql/)

**2.9 کوئری زدن روی‌یک پایگاه داده‌ی SQL از راه دور**

**مسئله**

شما باید به‌یک پایگاه داده SQL از راه دور متصل شوید و داده‌ها را از آن بخوانید.

**راه‌حل**

یک اتصال با pymysql ایجاد کنید و آن را با Pandas در‌یک DataFrame بخوانید:



|  |  |  |  |
| --- | --- | --- | --- |
|  | عدد | تاریخ و زمان | دسته بندی |
| 0 | 5 | 00:00:00 2015-01-01 | 0 |
| 1 | 5 | 00:00:01 2015-01-01 | 0 |

**بحث**

از بین تمام دستور العمل‌های ارائه شده در این فصل، احتمالاً این دستور العملی است که بیشتر در دنیای واقعی استفاده خواهیم کرد. در حالی که اتصال و خواندن از‌یک پایگاه داده نمونه‌ی sqlite مفید است، احتمالاً نماینده‌ی جداولی نیست که باید در‌یک محیط سازمانی به آنها متصل شوید. اکثر نمونه‌های SQL که به آن‌ها متصل می‌شوید، از شما می‌خواهند که به میزبان و پورت‌یک دستگاه از راه دور متصل شوید، و‌یک نام کاربری و رمز عبور برای احراز هویت مشخص کنید. این مثال از شما می‌خواهد که‌یک نمونه SQL در حال اجرا را [به صورت محلی راه‌اندازی کنید](https://github.com/kylegallatin/mysql-db-example) که از‌یک سرور راه دور در لوکال‌هاست تقلید می‌کند تا بتوانید حسی از گردش کار دریافت کنید.

**همچنین ببینید:**

* [مستندات PyMySQL](https://pymysql.readthedocs.io/en/latest/user/examples.html)
* [مستندات Read SQL در کتابخانه‌ی Pandas](https://pandas.pydata.org/docs/reference/api/pandas.read_sql.html)

**2.10 بارگیری داده‌ها از Google Sheet**

**مسئله**

شما باید داده‌ها را مستقیماً از Google Sheet بخوانید.

**راه‌حل**

از read\_CSV در کتابخانه‌ی pandas استفاده کنید و نشانی اینترنتی ارسال کنید که Google Sheet را به‌عنوان CSV صادر می‌کند:



|  |  |  |  |
| --- | --- | --- | --- |
|  | عدد | تاریخ و زمان | دسته بندی |
| 0 | 5 | 00:00:00 2015-01-01 | 0 |
| 1 | 5 | 00:00:01 2015-01-01 | 0 |

**بحث**

در حالی که Google Sheets را می‌توان به راحتی دانلود کرد، گاهی اوقات مفید است که بتوانید آنها را مستقیماً در پایتون بدون هیچ مرحله میانی بخوانید. پارامتر کوئری /export?format=csv در انتهای URL بالا‌یک نقطه پایانی ایجاد می‌کند که می‌توانیم فایل را دانلود کنیم یا در Pandas بخوانیم.

**همچنین ببینید:**

* [Google Sheets API](https://developers.google.com/sheets/api)

**2.11 بارگیری داده‌ها از سطل[[9]](#footnote-9) S3**

**مسئله**

شما باید‌یک فایل CSV را از‌یک سطل S3 که به آن دسترسی دارید بخوانید.

**راه‌حل**

گزینه‌های ذخیره سازی را به Pandas اضافه کنید تا به شیء[[10]](#footnote-10) S3 دسترسی داشته باشد:



|  |  |  |  |
| --- | --- | --- | --- |
|  | عدد | تاریخ و زمان | دسته بندی |
| 0 | 5 | 00:00:00 2015-01-01 | 0 |
| 1 | 5 | 00:00:01 2015-01-01 | 0 |

**بحث**

اکنون بسیاری از شرکت‌ها داده‌ها را در فروشگاه‌های blob ارائه‌دهنده ابری مانندAmazon S3 یاGoogle Cloud Storage(GCS) نگهداری می‌کنند. معمولاً متخصصان یادگیری ماشینی برای بازیابی داده‌ها به این منابع متصل می‌شوند. اگرچه S3 URI (s3://machine-learning-python-cookbook/data.csv) عمومی‌است، اما همچنان از شما می‌خواهد که اعتبار دسترسی AWS خود را برای دسترسی به آن ارائه دهید. شایان ذکر است که اشیاء عمومی‌همچنین دارای URL‌های HTTP هستند که می‌توانند فایل‌ها را از آن دانلود کنند، مانند [این مورد برای فایل CSV](https://oreil.ly/byelc).

**همچنین ببینید:**

* [Amazon S3](https://aws.amazon.com/s3/)
* [اعتبارنامه امنیتی AWS](https://docs.aws.amazon.com/IAM/latest/UserGuide/security-creds.html)

**2.12 بارگذاری داده‌های بدون ساختار**

**مسئله**

شما باید داده‌های بدون ساختار مانند متن یا تصاویر را بارگیری کنید.

**راه‌حل**

از تابع باز پایتون برای بارگذاری اطلاعات استفاده کنید:



|  |  |  |  |
| --- | --- | --- | --- |
|  | دسته بندی | تاریخ و زمان | عدد |
| 0 | 0 | 00:00:00 2015-01-01 | 5 |
| 1 | 0 | 00:00:01 2015-01-01 | 5 |

**بحث**

در حالی که داده‌های ساختاریافته را می‌توان به راحتی از CSV، JSON یا پایگاه‌های داده مختلف خواند، داده‌های بدون ساختار می‌توانند چالش برانگیزتر باشند و ممکن است نیاز به پردازش سفارشی داشته باشند. گاهی اوقات باز کردن و خواندن فایل‌ها با استفاده از تابع باز اصلی پایتون مفید است. این مورد به ما امکان می‌دهد فایل‌ها را باز کنیم و سپس محتوای آن فایل را بخوانیم.

**همچنین ببینید:**

* [تابع باز پایتون](https://docs.python.org/3/library/functions.html#open)
* [مدیران context در پایتون](https://docs.python.org/3/library/contextlib.html)

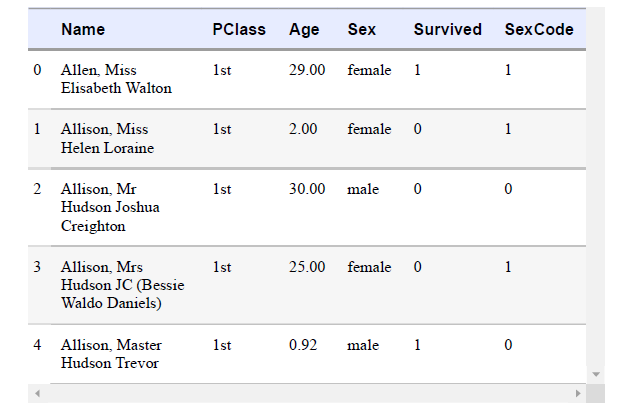
**فصل 3- آماده‌سازی داده[[11]](#footnote-11)**

**3.0 مقدمه**

آماده‌سازی اصطلاح گسترده‌ای است که اغلب به‌طور غیررسمی برای توصیف فرآیند تبدیل داده‌های خام به قالبی تمیز، سازمان‌یافته و آماده استفاده می‌شود. به نظر ما، آماده‌سازی فقط یک مرحله در پیش‌پردازش داده‌هایمان است، اما گامی مهم محسوب می‌‌شود.

متداول‌ترین ساختار داده‌ای که برای «آماده‌سازی» داده‌ها استفاده می‌شود دیتافریم است، که می‌تواند هم شهودی و هم فوق‌العاده همه‌کاره باشد. فریم‌‌های داده جدولی‌اند، یعنی بر اساس سطر‌‌ها و ستون‌‌هایی هستند که در صفحه‌گسترده مشاهده می‌‌کنید. در اینجا، یک دیتافریم حاصل از داده‌‌های مربوط به مسافران کشتی *تایتانیک* ایجاد می‌‌شود:





سه نکته مهم در این دیتافریم وجود دارد. اول، در یک دیتافریم، هر سطر مربوط به یک مشاهده (مثلاً، یک مسافر) و هر ستون مربوط به یک ویژگی (جنس، سن، و غیره) است. مثلاً، با نگاه‌کردن به اولین مشاهده، می‌‌توان دریافت که خانم الیزابت والتون آلن از مسافران کلاس اول[[12]](#footnote-12) کشتی، 29 ساله و زن بود، و از این فاجعه جان سالم به‌در برد.

دوم، هر ستون حاوی یک نام (مثلاً، Name، PClass، Age) و هر سطر حاوی یک عدد شاخص است (مثلاً، 0 برای خانم الیزابت والتون آلن خوش‌شانس). از اینها برای انتخاب و دستکاری مشاهدات و ویژگی‌‌ها استفاده خواهیم کرد.

سوم، دو ستون، Sex و SexCode، حاوی اطلاعات یکسان در قالب‌‌های مختلف است. در Sex، یک زن با رشته female نشان داده می‌‌شود، در حالی که در SexCode، یک زن با استفاده از عدد صحیح 1 نشان داده می‌‌شود. می‌‌خواهیم تمام ویژگی‌‌های ما یکتا باشند، و بنابراین باید یکی از این ستون‌‌ها را حذف کنیم.

در این فصل، با هدف ایجاد مجموعه‌ای از مشاهدات تمیز و ساختاریافته برای پیش‌پردازش بیشتر، تکنیک‌های مختلفی را برای دستکاری فریم‌های داده با استفاده از کتاب‌خانه pandas پوشش خواهیم داد.

**3.1 ایجاد دیتافریم**

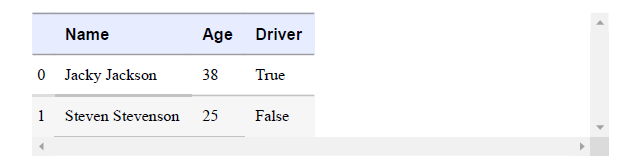
**مسئله**

می‌خواهید دیتافریم جدیدی را ایجاد کنید.

**راه‌حل**

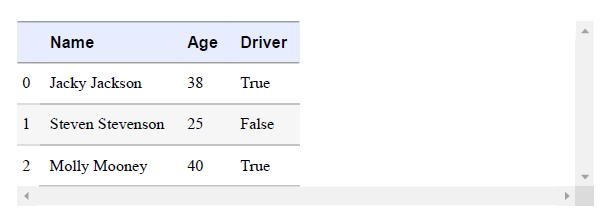
pandas روش‌‌های زیادی برای ایجاد یک شیء دیتافریم جدید دارد. یکی از روش‌‌های آسان ایجاد دیتافریم خالی با استفاده از DataFrame، و سپس، تعریف جداگانۀ هر ستون است:





پس از ایجاد یک شیء دیتافریم، می‌توانیم سطر‌های جدیدی را نیز به انتهای آن اضافه کنیم:





**بحث**

pandas روش‌های ظاهراً بی‌شماری را برای ایجاد یک دیتافریم ارائه می‌‌دهد. در دنیای واقعی، ایجاد دیتافریم خالی و سپس پرکردن آن تقریباً هرگز اتفاق نخواهد افتاد. بلکه، دیتافریم ما از داده‌های واقعی ایجاد می‌شود که از منابع دیگر بارگیری می‌کنیم (مثلاً، پایگاه داده یا فایل CSV).

**3.2 دریافت اطلاعات در مورد داده‌ها**

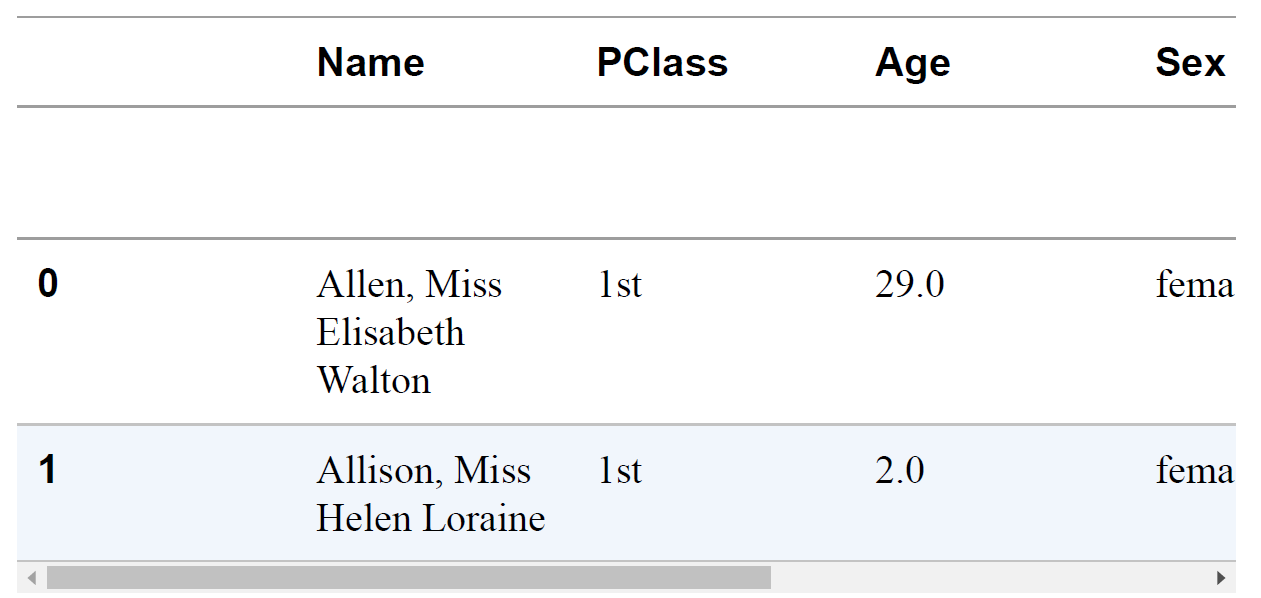
**مسئله**

شما می‌خواهید برخی از ویژگی‌های یک DataFrame را مشاهده کنید.

**راه‌حل**

یکی از ساده‌ترین کارهایی که می‌توانیم پس از بارگذاری داده‌ها انجام دهیم، مشاهده چند ردیف اول با استفاده از head است:



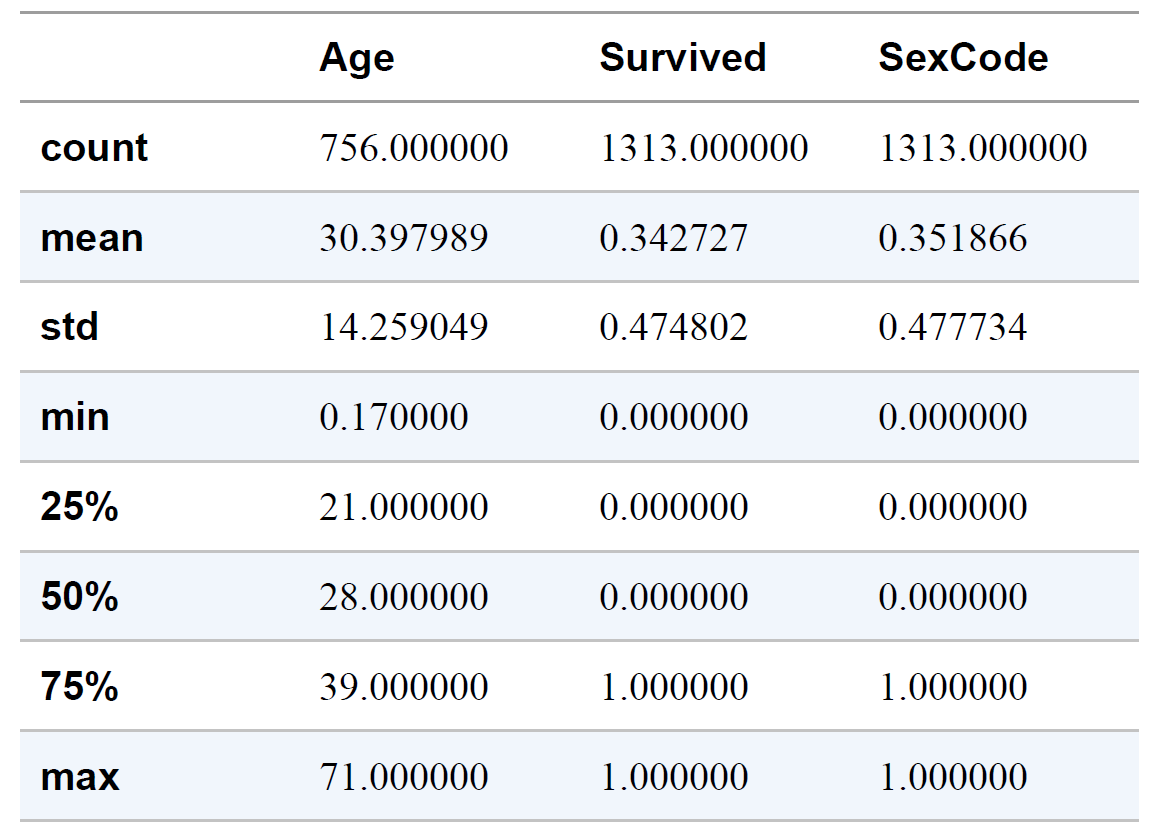


همچنین می‌توانیم به تعداد سطرها و ستون‌ها نگاهی بیندازیم:



ما می‌توانیم آمار توصیفی برای هر ستون عددی را با استفاده از describe بدست آوریم:



****

علاوه بر این، روش info می‌تواند اطلاعات مفیدی را نشان دهد:



**بحث**

پس از بارگیری برخی از داده ها، ایده خوبی است که بفهمیم ساختار آن چگونه است و چه نوع اطلاعاتی در آن وجود دارد. در حالت ایده آل، ما داده‌های کامل را مستقیماً مشاهده می‌کنیم. اما در بیشتر موارد دنیای واقعی، داده‌ها می‌توانند هزاران تا صدها هزار تا میلیون‌ها سطر و ستون داشته باشند. در عوض، برای مشاهده برش‌های کوچک و محاسبه آمار خلاصه داده‌ها باید به کشیدن نمونه تکیه کنیم.

در راه حل خود، ما از مجموعه داده اسباب بازی مسافران تایتانیک استفاده می‌کنیم. با استفاده از head، می‌توانیم به چند ردیف اول (پنج به طور پیش فرض) داده‌ها نگاه کنیم. از طرف دیگر، می‌توانیم از tail برای مشاهده چند ردیف آخر استفاده کنیم. با شکل می‌توانیم ببینیم که DataFrame ما شامل چند ردیف و ستون است. با توصیف می‌توانیم برخی از آمار توصیفی اولیه برای هر ستون عددی را ببینیم. و در نهایت، اطلاعات تعدادی از نقاط داده مفید را در مورد DataFrame نشان می‌دهد، از جمله انواع داده‌های شاخص و ستون، مقادیر غیر تهی و میزان استفاده از حافظه.

شایان ذکر است که آمار خلاصه، همیشه داستان کامل را بیان نمی‌کند. به عنوان مثال، Pandas با ستون‌های زنده‌مانده و کد جنسیتی به عنوان ستون‌های عددی رفتار می‌کنند زیرا دارای 1 و 0 هستند. با این حال، در این مورد مقادیر عددی نشان دهنده دسته‌ها هستند. به عنوان مثال، اگر Survived برابر با 1 باشد، نشان می‌دهد که مسافر از فاجعه جان سالم به در برده است. به همین دلیل، برخی از آمار خلاصه‌ی ارائه شده منطقی نیستند، مانند انحراف استاندارد ستون کد جنسیتی (نشانگر جنسیت مسافر).

**3.3 برش DataFrame ها**

**مسئله**

شما باید یک زیر مجموعه داده یا برش‌هایی از یک DataFrame را انتخاب کنید.

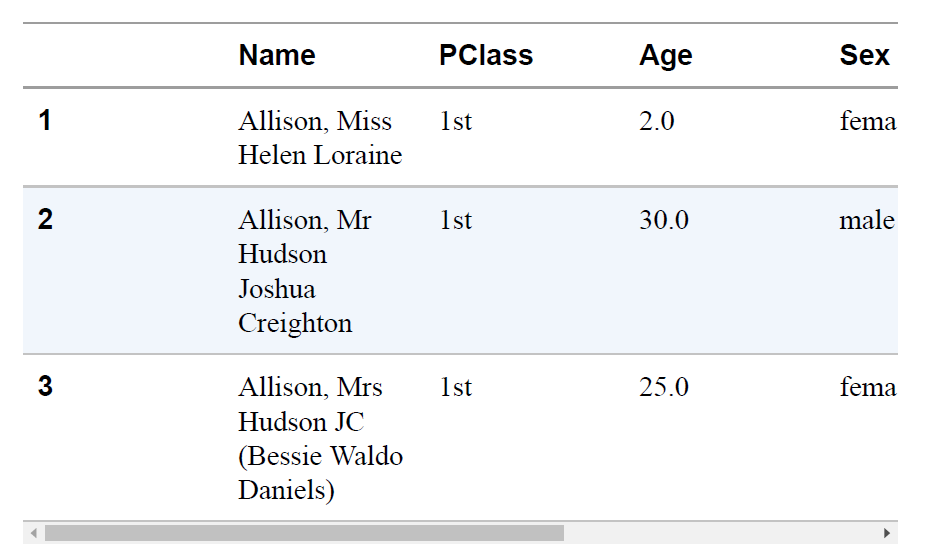
**راه‌حل**

از loc یا iloc برای انتخاب یک یا چند ردیف یا مقدار استفاده کنید:



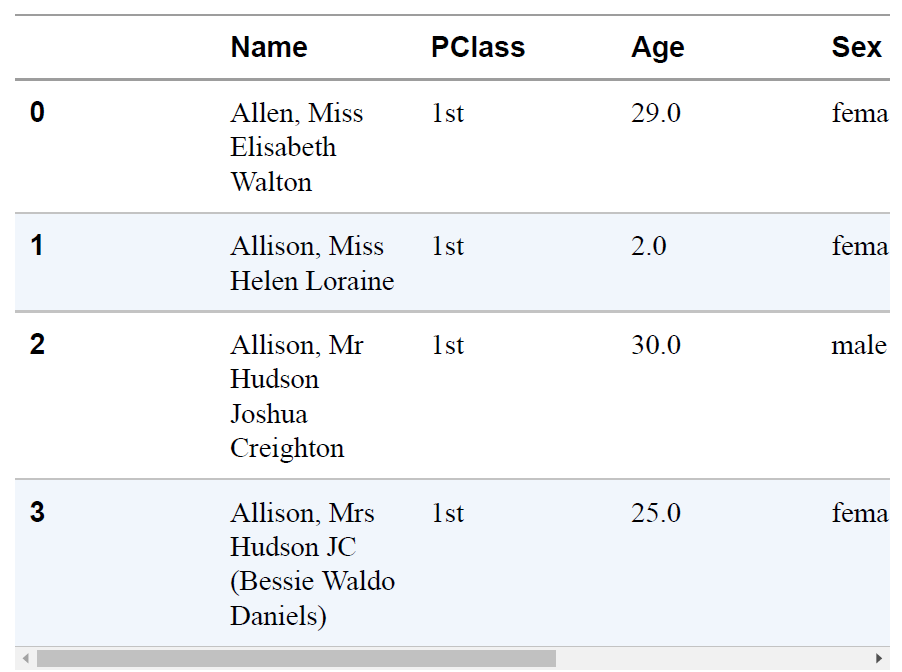
می‌توانیم از : برای تعریف برش ردیف‌هایی که می‌خواهیم، استفاده کنیم. مانند انتخاب ردیف‌های دوم، سوم و چهارم:





حتی می‌توانیم از آن برای به دست آوردن تمام ردیف‌ها تا یک نقطه استفاده کنیم. مانند همه ردیف‌ها تا ردیف چهارم:



****

DataFrame‌ها نیازی به ایندکس شدن عددی ندارند. ما می‌توانیم ایندکس یک DataFrame را روی هر مقداری که مقدار آن برای هر ردیف منحصر به فرد باشد، تنظیم کنیم. برای مثال، می‌توانیم فهرست را به‌عنوان نام مسافران تنظیم کنیم و سپس ردیف‌ها را با استفاده از یک نام انتخاب کنیم:



**بحث**

همه سطرها در DataFrame در کتابخانه‌ی Pandas دارای یک مقدار شاخص منحصر به فرد است. به طور پیش فرض، این شاخص یک عدد صحیح است که موقعیت ردیف را در DataFrame نشان می‌دهد. با این حال، لازم نیست که حضور داشته باشد. شاخص‌های DataFrame را می‌توان به صورت رشته‌های الفبایی منحصر به فرد یا شماره مشتری تنظیم کرد. Pandas برای انتخاب ردیف‌ها و تکه‌های ردیف‌ها دو روش ارائه می‌دهد:

* loc زمانی مفید است که نمایه DataFrame یک برچسب (به عنوان مثال، یک رشته) باشد.
* iloc با جستجوی موقعیت در DataFrame کار می‌کند. برای مثال، iloc[0] بدون در نظر گرفتن اینکه شاخص یک عدد صحیح است یا یک برچسب، ردیف اول را برمی‌گرداند.

استفاده زیاد و راحت بودن با loc و iloc مفید است زیرا هنگام پاکسازی داده‌ها زیاد ظاهر می‌شوند.

**3.4 انتخاب سطر‌‌ها بر اساس شرط**

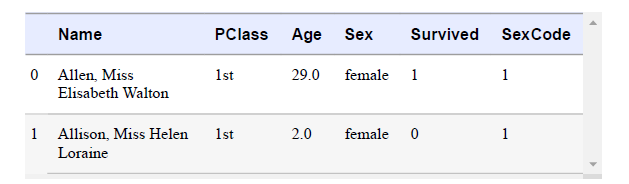
**مسئله**

می‌خواهید سطر‌‌های دیتافریم را بر اساس شرط یا شروط خاص انتخاب کنید.

**راه‌حل**

این کار را می‌‌توان به‌راحتی در pandas انجام داد. مثلاً، اگر بخواهیم تمام زنان *تایتانیک* را انتخاب کنیم:

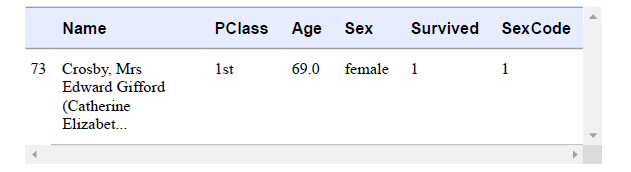




لحظه‌ای وقت بگذارید و به فرمت این راه‌حل نگاه کنید. dataframe['Sex'] == 'female' عبارت شرطی ما است؛ با قرار دادن آن در dataframe[]، به pandas می‌گوییم که «همه سطر‌هایی را در دیتافریم انتخاب کند که مقدار dataframe['Sex'] آن 'female’ است.

شرایط چندگانه نیز آسان است. مثلاً، در اینجا همه سطر‌هایی را انتخاب می‌کنیم که مسافر در آنها یک زن 65 ساله یا بیشتر است:





**بحث**

انتخاب مشروط و فیلترکردن داده‌‌ها یکی از رایج‌‌ترین کارها در داده‌ورزی است. به‌ندرت پیش می‌آید که تمام داده‌‌های خام اولیه را بخواهید؛ بلکه، فقط به بخشی از آن علاقه یا نیاز دارید. مثلاً، ممکن است فقط به فروشگاه‌‌های ایالت‌‌های خاص یا سوابق بیماران بالاتر از سن خاصی علاقه‌مند باشید.

**3.5 مرتب سازی مقادیر**

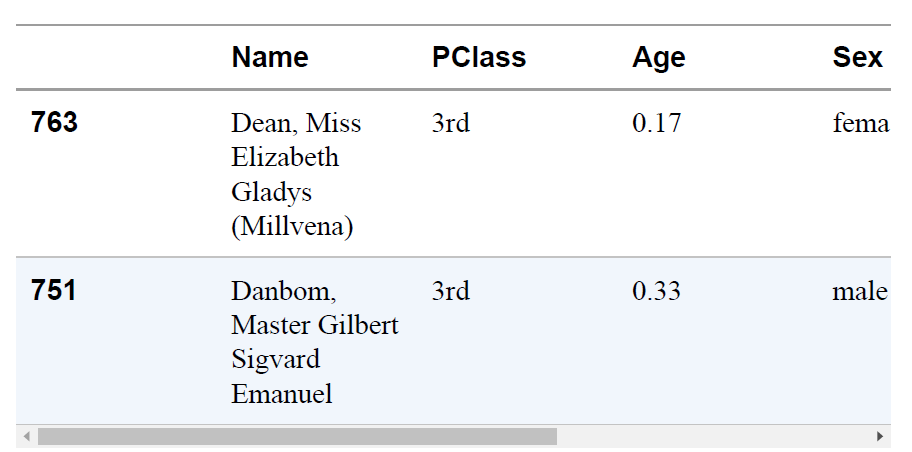
**مسئله**

شما باید یک دیتافریم را بر اساس مقادیر یک ستون مرتب کنید.

**راه‌حل**

از تابع sort\_values در کتابخانه‌ی pandas استفاده کنید:





**بحث**

در طول تجزیه و تحلیل و کاوش داده ها، مرتب کردن یک DataFrame بر اساس یک ستون یا مجموعه ای از ستون‌های خاص اغلب مفید است. آرگومان by در sort\_values فهرستی از ستون‌ها را می‌گیرد که براساس آن DataFrame مرتب می‌شود و این مرتب سازی بر اساس ترتیب نام ستون‌ها در لیست مرتب است.

به‌طور پیش‌فرض، آرگومان accending روی True تنظیم شده است، بنابراین مقادیر پایین‌ترین به بالاترین را مرتب می‌کند. اگر ما مسن‌ترین مسافران را به جای جوان‌ترین مسافران می‌خواستیم، می‌توانیم آن را روی False تنظیم کنیم.

**3.6 جایگزینی مقادیر**

**مسئله**

باید مقادیر را در یک دیتافریم جایگزین کنید.

**راه‌حل**

تابع replace در pandas راهی آسان برای یافتن و جایگزینی مقادیر است. برای مثال، می‌توانیم هر نمونه از female را در ستون Sex با Woman جایگزین کنیم:





می توانیم چندین مقدار را همزمان نیز جایگزین کنیم:

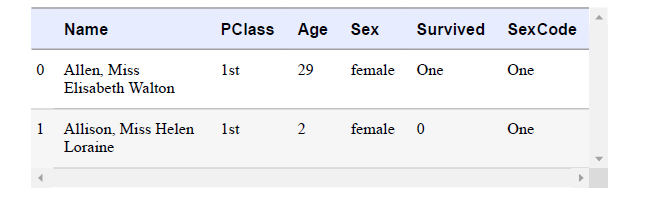




همچنین می‌توانیم کل شیء دیتافریم را تعیین کرده (به جای یک ستون خاص) و مقداری را در آن یافته و جایگزین کنیم:

# Replace values, show two rows

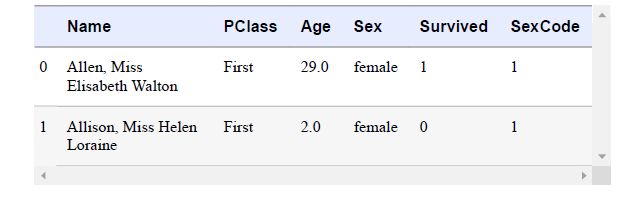
dataframe.replace(1, "One").head(2)



replace عبارات منظم[[13]](#footnote-13) را نیز می‌‌پذیرد:

# Replace values, show two rows

dataframe.replace(r"1st", "First", regex=True).head(2)



**بحث**

replace ابزاری است که برای جایگزینی مقادیر استفاده می‌‌کنیم که ساده، و در عین حال، برای پذیرش عبارات منظم بسیار قدرتمند است.

**3.7 تغییر نام ستون‌ها**

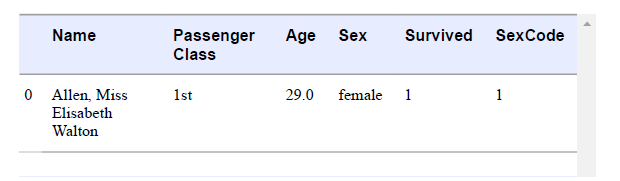
**مسئله**

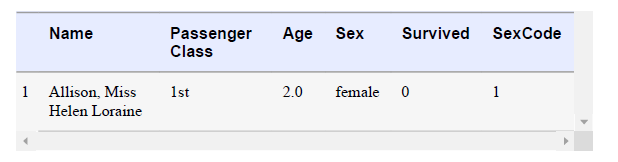
می‌خواهید نام یک ستون را در یک دیتافریم pandas تغییر دهید.

**راه‌حل**

نام ستون‌‌ها را با استفاده از روش rename تغییر دهید:

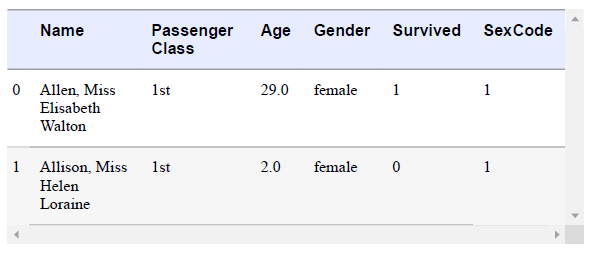






توجه داشته باشید که روش rename می‌‌تواند یک دیکشنری را به‌عنوان پارامتر بپذیرد. می‌‌توانیم از دیکشنری برای تغییر همزمان نام چندین ستون استفاده کنیم:





**بحث**

استفاده از rename با دیکشنری به‌عنوان آرگومان برای پارامتر columns روش ترجیحی‌ برای تغییر نام ستون‌هاست زیرا برای هر تعداد ستون کار می‌‌کند. اگر بخواهیم همه ستون‌ها را یک‌جا تغییر نام دهیم، این بخش مفید کد در واقع یک دیکشنری ایجاد می‌کند که نام ستون‌های قدیمی به‌عنوان کلید و رشته‌های خالی به‌عنوان مقادیر خواهد بود:





**3.8 یافتن حداقل، حداکثر، جمع، میانگین و تعداد**

**مسئله**

می‌خواهید حداقل، حداکثر، مجموع، میانگین یا تعداد یک ستون عددی را پیدا کنید.

**راه‌حل**

pandas چند روش‌ برای آماره‌‌های توصیفی رایج ارائه می‌‌دهد:



Maximum: 71.0

Minimum: 0.17

Mean: 30.397989417989415

Sum: 22980.879999999997

Count: 756

**بحث**

علاوه بر آماره‌‌های مورد استفاده در راه‌حل، pandas واریانس (var)، انحراف معیار (std)، کشیدگی (kurt)، چولگی (skew)، خطای استاندارد میانگین (sem)، نما (mode)، میانه (medi)، و چند آماره دیگر را ارائه می‌‌دهد.

افزون بر این، می‌‌توانیم این روش‌‌ها را برای کل دیتافریم نیز به کار ببریم:

# Show counts

dataframe.count()

Name 1313

PClass 1313

Age 756

Sex 1313

Survived 1313

SexCode 1313

dtype: int64

**3.9 یافتن مقادیر یکتا**

**مسئله**

می‌خواهید تمام مقادیر یکتا را در یک ستون انتخاب کنید.

**راه‌حل**

از unique برای مشاهده آرایه‌‌ای از تمام مقادیر یکتا در یک ستون استفاده کنید:



همچنین، value\_counts تمام مقادیر یکتا را با تعداد دفعات تکرارشان نمایش می‌‌دهد:

# Show counts

dataframe['Sex'].value\_counts()

male 851

female 462

Name: Sex, dtype: int64

**بحث**

هم unique و هم value\_counts برای دستکاری و نمایش ستون‌‌های رسته‌ای مفید هستند. اغلب در ستون‌های رسته‌ای، دسته‌‌هایی وجود دارند که باید در مرحله آماده‌سازی داده‌ها مدیریت شوند. مثلاً، در مجموعه داده تایتانیک، PClass ستونی است که دسته بلیط یک مسافر را نشان می‌‌دهد. در کشتی تایتانیک، سه دسته وجود داشت؛ با این حال، اگر از value\_counts استفاده کنیم، می‌توانیم مشکلی را ببینیم:





در حالی که تقریباً همۀ مسافران مطابق انتظار به یکی از سه دسته تعلق دارند، یک مسافر دسته \* را دارد. چند راهبرد برای رفع این نوع مسائل وجود دارد، که در فصل 5 به آنها خواهیم پرداخت، اما در حال حاضر، فقط بدانید که دسته‌های «اضافی» در داده‌های رسته‌ای رایج هستند و نباید نادیده گرفته شوند.

در نهایت، اگر صرفاً بخواهیم تعداد مقادیر یکتا را بشماریم، می‌‌توانیم از nunique استفاده کنیم:

# Show number of unique values

dataframe['PClass'].nunique()

4

**3.10 مدیریت مقادیر مفقود[[14]](#footnote-14)**

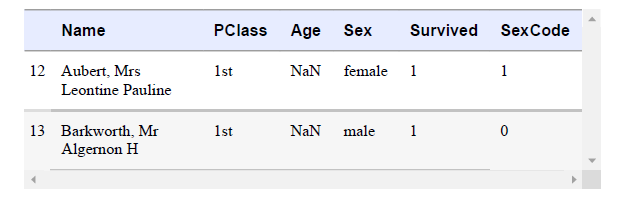
**مسئله**

می‌خواهید مقادیر مفقود را در یک دیتافریم انتخاب کنید.

**راه‌حل**

isnull و notnull مقدار boolean را برمی‌گرداند که نشان می‌‌دهد آیا مقداری مفقود است یا خیر:





**بحث**

مقادیر مفقود در واقع مسئله‌ای رایج در داده‌ورزی هستند، با این حال، بسیاری دشواری کار با داده‌‌های مفقود را دست‌کم می‌‌گیرند. pandas از NumPy’s NaN (نه یک عدد) برای نشان‌دادن مقادیر مفقود استفاده می‌‌کند، اما باید توجه داشته باشید که NaN کاملاً در pandas پیاده‌سازی نشده است. مثلاً، اگر بخواهیم تمام رشته‌‌های حاوی male را با مقادیر مفقود جایگزین کنیم، دچار خطا می‌‌شویم:



------------------------------------------------------------

---------------

NameError Traceback (most

recent call last)

<ipython-input-7-5682d714f87d> in <module>()

1 # Attempt to replace values with NaN

----> 2 dataframe['Sex'] = dataframe['Sex'].replace('male',

NaN)

NameError: name 'NaN' is not defined

------------------------------------------------------------

---------------

برای عملکرد کامل با NaN، ابتدا باید کتاب‌خانه NumPy را وارد کنیم:



یک مجموعه داده اغلب از مقدار خاصی برای نشان‌دادن مشاهدۀ مفقودی استفاده می‌‌کند، مانند NONE، و -999. read\_csv pandas پارامتری دارد که با آن می‌‌توانیم مقادیر به‌کاررفته برای نشان‌دادن مقادیر مفقود را مشخص کنیم:



**3.11 حذف یک ستون**

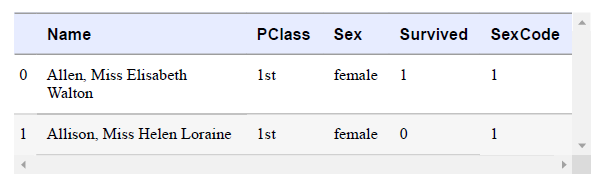
**مسئله**

می‌خواهید یک ستون را از دیتافریم خود حذف کنید.

**راه‌حل**

بهترین راه برای حذف یک ستون استفاده از drop با پارامتر axis=1 (یعنی محور ستون) است:

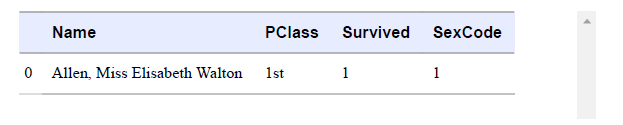


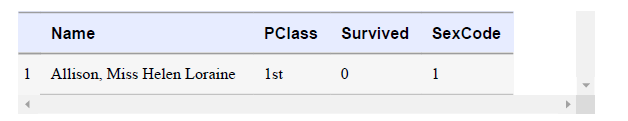


می‌توانید از فهرستی از نام ستون‌ها به‌عنوان آرگومان اصلی برای حذف‌کردن همزمان چندین ستون استفاده کنید:

# Drop columns

dataframe.drop(['Age', 'Sex'], axis=1).head(2)

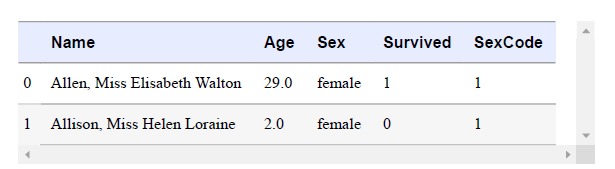




اگر ستونی نام ندارد (که گاهی ممکن است اتفاق بیفتد)، می‌توانید با استفاده از dataframe.columns، آن را با نمایۀ ستون آن حذف کنید:

# Drop column

dataframe.drop(dataframe.columns[1], axis=1).head(2)



**بحث**

drop روشی مصطلح برای حذف یک ستون است. روش دیگر del dataframe['Age'] است که بیشتر اوقات کارساز است، اما به دلیل نوع فراخوانی آن در pandas توصیه نمی‌شود (جزئیات آن خارج از حوصلۀ کتاب حاضر است).

یکی از عادت‌‌هایی که توصیه می‌‌کنم یاد بگیرید این است که هرگز از آرگومان inplace=True pandas استفاده نکنید. بسیاری از توابع pandas یک پارامتر inplace دارند که اگر مقدار آن True باشد مسقیم دیتافریم را ویرایش می‌کند. این امر ممکن است سبب ایجاد مشکلاتی در مسیرپردازش داده‌‌های پیچیده‌تر شود، زیرا دیتافریم‌ها را اشیای تغییرپذیری[[15]](#footnote-15) در نظر می‌گیریم (که به‌لحاظ فنی واقعا همین‌گونه‌اند). توصیه می‌‌کنم دیتافریم‌ها را اشیای تغییرناپذیری[[16]](#footnote-16) در نظر بگیرید. مثلاً:



در این مثال، دیتافریم dataframe را تغییر نمی‌دهیم، بلکه دیتافریم جدیدی را می‌سازیم که نسخه تغییر‌یافته‌ای از dataframe به‌نام dataframe\_name\_dropped است. اگر دیتافریم‌های خود را اشیای تغییرناپذیری در نظر بگیرید، از مشکلات زیادی در ادامۀ راه نجات خواهید یافت.

**3.12 حذف یک سطر**

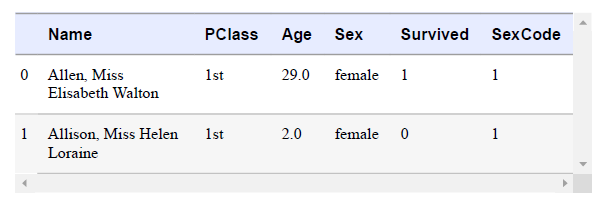
**مسئله**

می‌خواهید یک یا چند سطر را از یک دیتافریم حذف کنید.

**راه‌حل**

از یک شرط منطقی برای ایجاد دیتافریم جدیدی، شامل همه سطرها به‌جز سطر‌‌هایی که می‌‌خواهید حذف کنید، استفاده کنید:



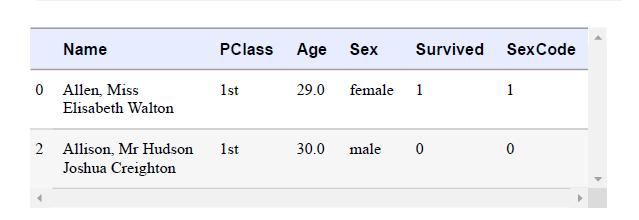


**بحث**

در حالی که به‌لحاظ تخصصی، می‌‌توانید از روش drop استفاده کنید (مثلاً df.drop([0, 1], axis=0) برای حذف‌کردن دو سطر اول)، روش کاربردی‌‌تر این است که صرفاً یک شرط منطقی را در داخل [] df قرار دهید زیرا می‌توانیم از توان شرطی‌ها برای حذف همزمان یک سطر یا (به احتمال زیاد) سطر‌های بسیاری استفاده کنیم.

می‌توانیم از شروط منطقی برای حذف آسان سطر‌های منفرد با تطبیق مقداری یکتا استفاده کنیم:

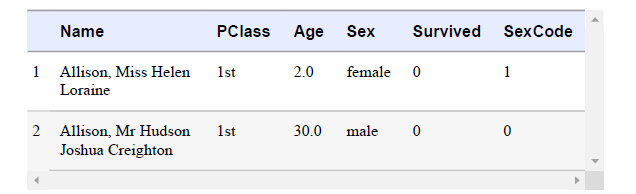




و حتی می‌‌توانیم از آن برای حذف یک سطر از طریق نمایۀ سطر استفاده کنیم:



Name



**3.13 حذف سطر‌‌های تکراری**

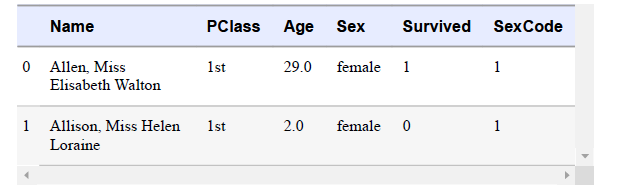
**مسئله**

می‌خواهید سطر‌‌های تکراری را از دیتافریم خود حذف کنید.

**راه‌حل**

از drop\_duplicates استفاده کنید، اما مراقب پارامترها باشید:





**بحث**

خوانندگان مشتاق متوجه خواهند شد که این راه‌حل در واقع هیچ سطری را حذف نکرده است:



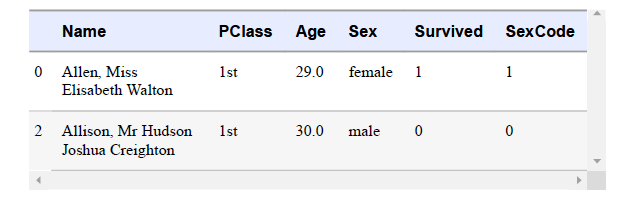
Number Of Rows In The Original DataFrame: 1313

Number Of Rows After Deduping: 1313

دلیل آن این است که drop\_duplicates مطابق پیش‌فرض تنها سطر‌هایی را حذف می‌کند که کاملاً با تمام ستون‌ها مطابقت دارند. در این شرایط، هر سطر در دیتافریم ما، dataframe، در واقع یکتاست. با این حال، اغلب می‌‌خواهیم تنها زیرمجموعه‌‌ای از ستون‌‌ها را برای بررسی سطر‌‌های تکراری در نظر بگیریم. می‌‌توانیم این کار را با استفاده از پارامتر subset انجام دهیم:

# Drop duplicates

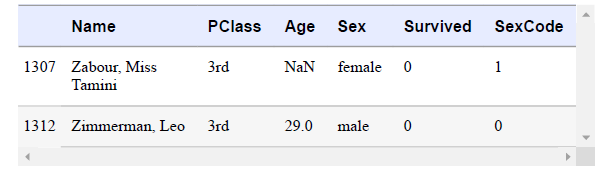
dataframe.drop\_duplicates(subset=['Sex'])



به خروجی قبلی نگاه دقیقی بیاندازید: به drop\_duplicates گفتیم که فقط دو سطر با مقدار مشابه برای Sex را تکراری در نظر بگیرد و آنها را حذف کند. اکنون، یک دیتافریم با تنها دو سطر باقی مانده است: یک مرد و یک زن. ممکن است بپرسید چرا drop\_duplicates تصمیم گرفت این دو سطر را به جای دو سطر مختلف نگه دارد. پاسخ این است که drop\_duplicates به‌طور پیش‌فرض اولین مرتبۀ یک سطر تکراری را نگه می‌دارد و بقیه را حذف می‌کند. می‌‌توانیم این رفتار را با استفاده از پارامتر keep کنترل کنیم:

# Drop duplicates

dataframe.drop\_duplicates(subset=['Sex'], keep='last')



یکی از روش‌‌های مرتبط duplicated است، که یک سری مقدار boolean را باز می‌‌گرداند که نشان می‌‌دهد آیا سطری تکراری است یا خیر. اگر نمی‌‌خواهید صرفاً موارد تکراری را حذف کنید، این روش راه خوبی است.

**3.14 گروه‌بندی سطر‌‌ها بر اساس مقادیر**

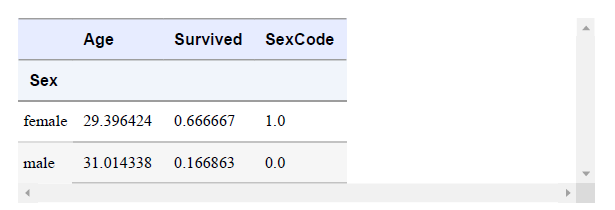
**مسئله**

می‌خواهید سطر‌‌ها را بر اساس مقداری مشترک گروه‌بندی کنید.

**راه‌حل**

groupby یکی از قدرتمندترین ویژگی‌‌های pandas است:





**بحث**

groupby جایی است که شکل‌گیری داده‌ورزی واقعاً آغاز می‌‌شود. معمولاً، دیتافریم‌‌هاییداریم که در آن هر سطر یک شخص یا یک رویداد است، و می‌‌خواهیم آنها را بر اساس معیاری گروه‌بندی، و سپس، آماره‌ای را محاسبه کنیم. مثلاً، می‌‌توانید نوعی دیتافریم را تصور کنید که در آن هر سطر یک مورد فروش در یک رستوران زنجیره‌‌ای ملی است، و کل فروش هر رستوران را می‌‌خواهیم. می‌توانیم این کار را با گروه‌بندی سطر‌ها بر اساس تک‌‌تک رستوران‌‌‌ها و سپس محاسبه مجموع هر گروه انجام دهیم.

کاربرانی که تازه با groupby آشنا شده‌اند اغلب خطی مانند خط زیر را می‌‌نویسند و از نتیجه گیج می‌‌شوند:



چرا نتیجۀ مفیدتری نداشت؟ زیرا groupby باید با عملیاتی که می‌خواهیم برای هر گروه اعمال کنیم جفت شود، مانند محاسبه آماره‌ کلی (مثلاً میانگین، میانه، جمع). وقتی در مورد گروه‌بندی صحبت می‌‌کنیم، اغلب از کوتاه‌نویسی استفاده می‌‌کنیم و می‌‌گوییم «بر اساس جنسیت گروه‌بندی کن»، اما ناقص است. برای این‌که گروه‌بندی مفید باشد، باید بر اساس چیزی گروه‌بندی کنیم، و سپس، برای هر یک از آن گروه‌‌ها یک تابع را به کار ببریم:

# Group rows, count rows

dataframe.groupby('Survived')['Name'].count()

Survived

0 863

1 450

Name: Name, dtype: int64

متوجه شدید Name بعد از groupby اضافه شده است؟ به این دلیل ‌است که آماره‌‌های خلاصۀ معینی فقط برای انواع خاصی از داده‌‌ها معنادارند. برای مثال، در حالی که محاسبه میانگین سن بر اساس جنسیت منطقی است، محاسبه کل سن بر اساس جنسیت منطقی نیست. در این مورد، داده‌ها را به بازمانده/نجات‌یافته یا غیربازمانده گروه‌بندی می‌کنیم، سپس تعداد اسامی (یعنی مسافران) را در هر گروه می‌شماریم.

می‌توانیم براساس ستون اول نیز گروه‌بندی کنیم، سپس آن گروه‌بندی را با ستون دوم گروه‌بندی کنیم:

# Group rows, calculate mean

dataframe.groupby(['Sex','Survived'])['Age'].mean()

Sex Survived

female 0 24.901408

1 30.867143

male 0 32.320780

1 25.951875

Name: Age, dtype: float64

**3.15 گروه‌بندی سطر‌‌ها بر اساس زمان**

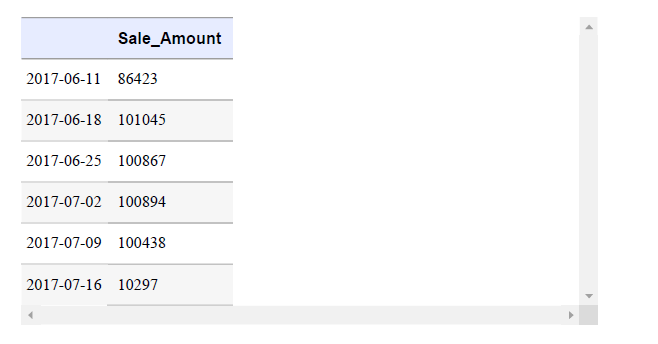
**مسئله**

باید سطر‌‌ها را بر اساس دوره‌‌های زمانی گروه‌‌بندی کنید.

**راه‌حل**

از resample برای گروه‌‌بندی سطر‌‌ها بر اساس دوره‌‌های زمان استفاده کنید:





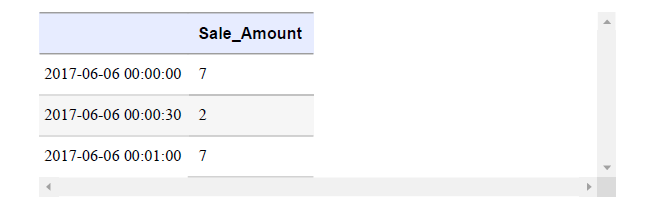
**بحث**

مجموعه دادۀ استاندارد *تایتانیک* ما یک ستون زمانی ندارد، بنابراین برای این دستورالعمل، دیتافریم ساده‌ای را ایجاد کرده‌ایم که در آن هر سطر یک مورد فروش جداگانه است. برای هر مورد فروش، تاریخ و زمان آن و مبلغ دلاری آن را می‌دانیم (این داده‌ها واقع‌بینانه نیستند، زیرا هر مورد فروش دقیقاً با فاصله 30 ثانیه از فروش بعدی اتفاق افتاده و مبلغ دلاری آن دقیق است، اما برای سادگی کار، این‌گونه فرض می‌‌کنیم).

داده‌‌های خام به صورت زیر است:

# Show three rows

dataframe.head(3)

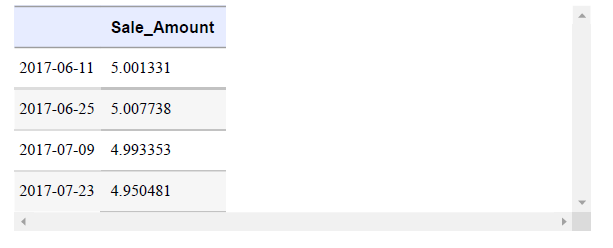


توجه داشته باشید که تاریخ و زمان هر مورد فروش در واقع شاخص دیتافریم است زیرا برای datetime، شاخص باید مقادیری شبیه به دورۀ زمانی باشد.

با استفاده از datetime، می‌توانیم سطر‌ها را با دوره‌های زمانی مختلفی گروه‌بندی کنیم (آفست) و سپس می‌توانیم آماره‌ای را برای هر گروه زمانی محاسبه کنیم:

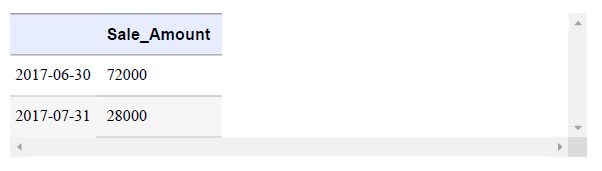
# Group by two weeks, calculate mean

dataframe.resample('2W').mean()



# Group by month, count rows

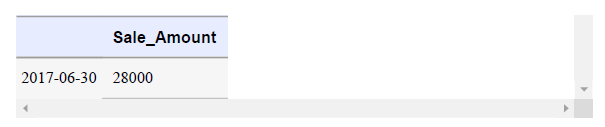
dataframe.resample('M').count()



ممکن است متوجه شوید که در دو خروجی، شاخص دورۀ زمانی یک تاریخ است، با این‌که به‌ترتیب بر حسب هفته‌ و ماه‌ گروه‌بندی می‌کنیم زیرا به‌طور پیش‌فرض، resample برچسب «لبۀ» سمت راست (آخرین برچسب) گروه زمانی را برمی‌گرداند. می‌‌توانیم این رفتار را با استفاده از پارامتر label کنترل کنیم:





****

**مطالعۀ بیشتر**

* [فهرست اصطلاحات و اسامی مستعار آفست زمانی pandas](http://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html)

**3.16 تجمیع عملیات و آمار**

**مسئله**

شما باید یک عملیات را روی هر ستون (یا مجموعه ای از ستون ها) در یک دیتافریم جمع کنید.

**راه‌حل**

از روش agg در کتابخانه‌ی pandas استفاده کنید. در اینجا، ما به راحتی می‌توانیم حداقل مقدار هر ستون را بدست آوریم:



گاهی اوقات، ما می‌خواهیم توابع خاصی را به مجموعه‌های خاصی از ستون‌ها اعمال کنیم:



|  |  |  |
| --- | --- | --- |
|  | سن | جنسیت |
| میانگین | 30.397989 | Nan |
| مینیمم | Nan | 0.0 |
| ماکزیمم | Nan | 1.0 |

همچنین می‌توانیم توابع انبوه را برای گروه‌ها اعمال کنیم تا آمار توصیفی و خاص‌تری به دست آوریم:



|  |  |  |  |
| --- | --- | --- | --- |
| کلاس P | زنده مانده |  | تعداد |
| 0 | \* | 0 | 1 |
| 1 | اول | 0 | 129 |
| 2 | اول | 1 | 193 |
| 3 | دوم | 0 | 160 |
| 4 | دوم | 1 | 119 |
| 5 | سوم | 0 | 573 |
| 6 | سوم | 1 | 138 |

**بحث**

توابع انبوه به ویژه در طول تجزیه و تحلیل داده‌های اکتشافی برای یادگیری اطلاعات در مورد زیرجمعیت‌های[[17]](#footnote-17) مختلف داده‌ها و رابطه بین متغیرها مفید هستند. با گروه‌بندی داده‌ها و اعمال آمار انبوه، می‌توانید الگوهایی را در داده‌ها مشاهده کنید که ممکن است در طی فرآیند یادگیری ماشین یا مهندسی ویژگی مفید باشند. در حالی که نمودارهای بصری نیز مفید هستند، اغلب مفید است که چنین آمار توصیفی خاصی نیز به عنوان مرجع برای درک بهتر داده‌ها وجود داشته باشد.

**همچنین ببینید:**

* [مستندات agg در کتابخانه‌ی Pandas](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.agg.html)

**3.17 حلقه زدن[[18]](#footnote-18) روی یک ستون**

**مسئله**

می‌خواهید روی هر عنصر یک ستون حلقه بزنید و اقدامی را انجام دهید.

**راه‌حل**

می توانید ستون pandas را مانند هر دنباله دیگری در پایتون در نظر بگیرید:



**بحث**

علاوه بر حلقه‌ها[[19]](#footnote-19) (که اغلب حلقه‌های for نامیده می‌شوند)، می‌توانیم از list comprehension نیز استفاده کنیم:



علی‌رغم وسوسۀ بازگشت به حلقه‌های for، راه‌حل پایتونی‌تر استفاده از روش apply pandas است که در دستور بعدی توضیح داده شده است.

**3.18 اِعمال یک تابع روی همه اجزا در یک ستون**

**مسئله**

می‌خواهید تابعی را روی تمام اجزای یک ستون اعمال کنید.

**راه‌حل**

از apply برای اعمال تابع داخلی یا ویژه‌ای روی هر عنصر در یک ستون استفاده کنید:



0 ALLEN, MISS ELISABETH WALTON

1 ALLISON, MISS HELEN LORAINE

Name: Name, dtype: object

**بحث**

Apply راهی عالی برای پاک‌سازی و آماده‌سازی داده‌‌ها است. می‌‌توان تابعی را برای انجام عملیات مفیدی (جدا کردن نام و نام خانوادگی، تبدیل رشته‌‌ها به مقادیر شناور و غیره) و سپس نگاشت آن تابع به هر عنصر در یک ستون نوشت.

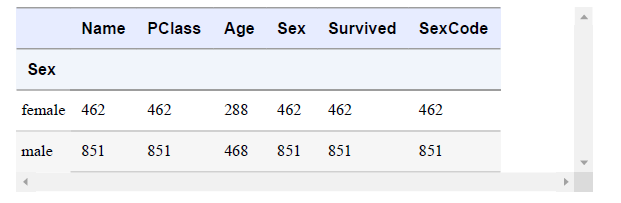
**3.19 اعمال یک تابع در گروه‌ها**

**مسئله**

سطرها را با استفاده از groupby گروه‌بندی کرده‌اید و می‌‌خواهید برای هر گروه، تابعی را اعمال کنید.

**راه‌حل**

groupby و apply را تلفیق کنید:



**بحث**

در دستورالعمل 3.16، به apply اشاره کردم. apply به‌ویژه زمانی مفید است که می‌‌خواهید تابعی را در گروه‌‌ها اعمال کنید. با ترکیب groupby و apply، می‌‌توانیم آماره‌ای خاص را محاسبه یا هر تابعی را جداگانه برای هر گروه اعمال کنیم.

**3.20 الحاق[[20]](#footnote-20) دیتافریم‌ها**

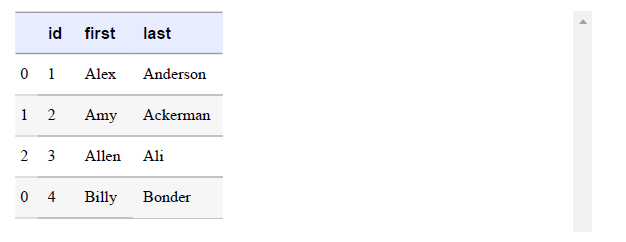
**مسئله**

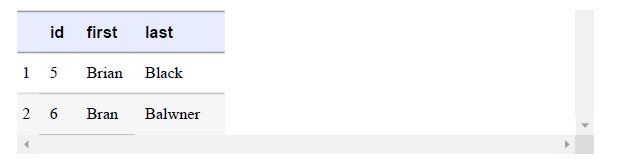
می‌خواهید دو دیتافریم را به هم الحاق کنید.

**راه‌حل**

از concat با axis=0 برای الحاق در امتداد محور سطر استفاده کنید:



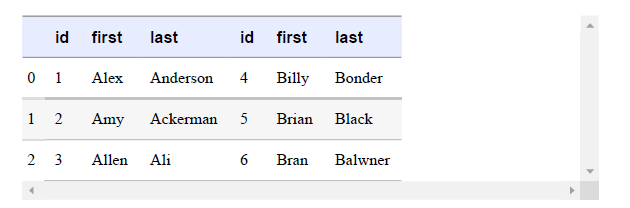




می توانید از axis=1 برای الحاق در امتداد محور ستون استفاده کنید:

# Concatenate DataFrames by columns

pd.concat([dataframe\_a, dataframe\_b], axis=1)



**بحث**

الحاق کلمه‌‌ای نیست که خارج از علوم رایانه و برنامه‌نویسی زیاد بشنوید، بنابراین اگر قبلاً آن را نشنیده‌اید، نگران نباشید. تعریف غیررسمی *الحاق* چسباندن دو شیء به هم است. در راه‌حل، دو دیتافریم کوچک را با استفاده از پارامتر axis به هم چسباندیم تا ببینیم که آیا می‌‌خواهیم دو دیتافریم را در امتداد هم قرار دهیم یا کنار هم.

به عنوان روش دیگر، می‌‌توانیم از append برای افزودن سطر جدیدی به دیتافریم استفاده کنیم:





**3.21 ادغام[[21]](#footnote-21) دیتافریم‌ها**

**مسئله**

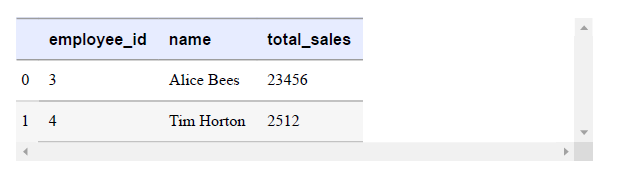
می‌خواهید دو دیتافریم را ادغام کنید.

**راه‌حل**

برای پیوستن داخلی[[22]](#footnote-22)، از merge با پارامتر on استفاده کنید تا ستون مورد نظر برای ادغام را مشخص کنید:

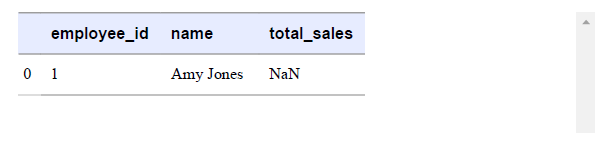


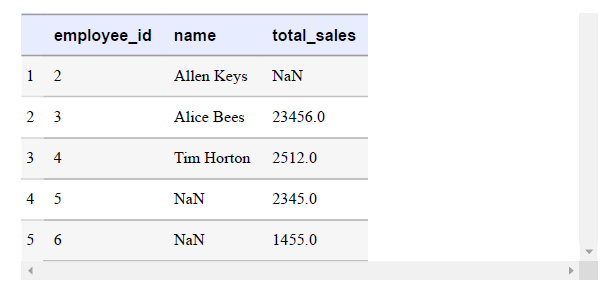




merge به‌ طور پیش‌فرض سبب پیوستن داخلی می‌‌شود. اگر بخواهیم پیوست خارجی[[23]](#footnote-23) را انجام دهیم، می‌‌توانیم آن را با پارامتر how مشخص کنیم:

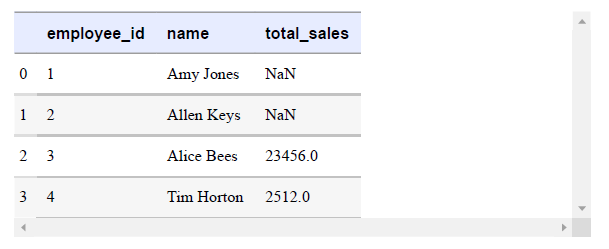






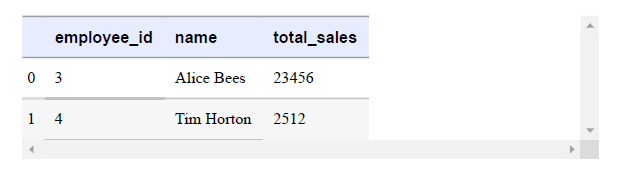
از همین پارامتر می‌‌توان برای تعیین پیوست‌‌های چپ و راست استفاده کرد:





می‌توانیم نام ستون را در هر دیتافریم برای ادغام در آن نیز مشخص کنیم:





اگر به جای ادغام در دو ستون، بخواهیم در شاخص‌های هر دیتافریم چیزی را ادغام کنیم، می‌‌توانیم پارامترهای left\_on و right\_on را با right\_index=True و left\_index=True جایگزین کنیم.

**بحث**

اغلب، داده‌هایی که باید استفاده کنیم پیچیده‌اند و هموارهیک بخش نمی‌‌شود. بلکه در دنیای واقعی، معمولاً با مجموعه داده‌های متفاوت، از کوئری‌ها یا فایل‌های پایگاه داده متعدد، مواجه هستیم. برای جمع‌آوری همه این داده‌ها در یک مکان، می‌توانیم هر کوئری داده یا فایل داده را به‌عنوان دیتافریم جداگانه در pandas بارگذاری، و سپس، آنها را با هم در یک دیتافریم ادغام کنیم.

این فرآیند ممکن است برای هر کسی آشنا باشد که از SQL، زبانی محبوب برای انجام عملیات ادغام (به نام join)، استفاده کرده باشد. در حالی که پارامترهای دقیق pandas متفاوت خواهند بود، از همان الگوهای عمومی سایر زبان‌‌ها و ابزارهای نرم‌افزاری پیروی می‌‌کنند.

برای هر عملیات merge، سه جنبه وجود دارد. ابتدا، باید دو دیتافریمی را مشخص کنیم که می‌‌خواهیم با هم ادغام کنیم. در راه‌حل، آنها را dataframe\_employees و dataframe\_sales نا‌میدیم. دوم، باید نام(های) ستون‌‌هایی را برای ادغام مشخص کنیم، یعنی ستون‌‌هایی که مقادیرشان بین دو دیتافریم مشترک است. مثلاً، در راه‌حل ما، هر دو دیتافریم ستونی به نام working\_id دارند. برای ادغام دو دیتافریم، مقادیر موجود در ستون employee\_id هر دیتافریم را با یکدیگر تطبیق می‌‌دهیم. اگر این دو ستون از یک نام استفاده می‌‌کنند، می‌‌توانیم از پارامتر on استفاده کنیم. با این حال، اگر نام‌های متفاوتی داشته باشند، می‌توانیم از left\_on و right\_on استفاده کنیم.

دیتافریم چپ و راست چیست؟ پاسخ ساده این است که دیتافریم سمت چپ اولین موردی است که در merge مشخص کردیم و دیتافریم سمت راست دومین مورد است. این زبان در مجموعه پارامترهای بعدی که به آن نیاز خواهیم داشت دوباره مطرح می‌‌شود.

آخرین جنبه، که درک آن برای برخی افراد دشوارتر است، نوع عملیات ادغامی است که می‌‌خواهیم انجام دهیم. این امر با پارامتر how مشخص می‌‌شود. merge از چهار نوع اصلی پیوست پشتیبانی می‌‌کند:

درونی:

فقط سطر‌هایی را برمی‌گرداند که در هر دو دیتافریم مطابقت دارند (مثلاً، هر سطری را با مقدار employee\_id برگردانید که هم در dataframe\_employees و هم dataframe\_sales ظاهر می‌شود).

بیرونی:

همه سطرها را در هر دو دیتافریم برگردانید. اگر سطری در یک دیتافریم وجود دارد اما در دیتافریم دیگر وجود ندارد، مقادیر NaN را برای مقادیر مفقود پر کنید (مثلاً، همه سطر‌ها را هم در working\_id و هم dataframe\_sales برگردانید).

چپ:

همه سطر‌ها را از دیتافریم سمت چپ برگردانید، اما فقط سطر‌هایی از دیتافریم سمت راست که با دیتافریم سمت چپ مطابقت دارند. مقادیر NaN را برای مقادیر مفقود پر کنید (مثلاً، برگرداندن تمام سطر‌‌ها از dataframe\_employees اما فقط سطر‌‌هایی از dataframe\_sales که مقداری برای staff\_id دارند که در dataframe\_employees ظاهر می‌‌شود).

راست:

همه سطر‌ها را از دیتافریم سمت راست برگردانید، اما فقط سطر‌هایی از دیتافریم سمت چپ که با دیتافریم سمت راست مطابقت دارند. مقادیر NaN را برای مقادیر مفقود پر کنید (مثلاً، برگرداندن تمام سطر‌‌ها از dataframe\_sales اما فقط سطر‌هایی از dataframe\_employees که مقداری برای staff\_id دارند که در dataframe\_sales ظاهر می‌شود).

اگر در حال حاضر همه اینها را متوجه نشدید، تشویق‌تان می‌کنم که پارامتر how را در کد خود تمرین کنید و ببینید که چگونه بر نتیجۀ merge تأثیر می‌گذارد.

**مطالعۀ بیشتر**

* [توضیح تصویری پیوست‌‌های SQL](http://bit.ly/2Fxgcpe)
* [اسناد pandas در مورد ادغام](http://bit.ly/2Fuo4rH)

**فصل 4. مدیریت داده‌‌های عددی**

**4.0 مقدمه**

داده‌‌های کمّی برای اندازه‌گیری چیزی‌اند - خواه اندازه دسته، فروش ماهانه، یا نمرات دانش‌آموزان. راهی طبیعی برای نمایش این مقادیر به‌صورت عددی است (مثلاً، 29 دانش‌آموز، 529،392 دلار فروش). در این فصل، راهبردهای فراوانی را برای تبدیل داده‌های عددی خام به ویژگی‌های هدفمند برای الگوریتم‌های یادگیری ماشین پوشش خواهیم داد.

**4.1 تغییر مقیاس یک ویژگی**

**مسئله**

باید مقادیر یک ویژگی عددی را دوباره مقیاس‌‌بندی کنید تا بین دو مقدار معین باشد.

**راه‌حل**

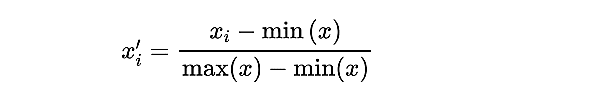
از MinMaxScaler کتابخانه scikit-learn برای تغییر مقیاس نوعی آرایه ویژگی استفاده کنید:





**بحث**

تغییر مقیاس کار پیش‌پردازش رایجی در یادگیری ماشین است. بسیاری از الگوریتم‌هایی که بعداً در کتاب حاضر توضیح داده می‌شوند فرض می‌کنند که همه ویژگی‌ها در مقیاس یکسانی‌، معمولاً 0 تا 1 یا 1- تا 1، قرار دارند. چند تکنیک‌ تغییر مقیاس وجود دارد، اما یکی از ساده‌ترین آنها *مقیاس‌بندی حداقل-حداکثر* نام دارد. مقیاس‌بندی حداقل-حداکثر از حداقل و حداکثر مقادیر یک ویژگی برای تغییر مقیاس مقادیر در محدوده‌ای استفاده می‌‌کند. مشخصاً، *مقیاس‌بندی حداقل-حداکثر* محاسبه می‌‌کند:



که در آن x بردار ویژگی است، xi یک عنصر منفرد از ویژگی x، و x'i عنصر تغییر‌مقیاس‌یافته است. در مثال ما، می‌توانیم از آرایه خروجی ببینیم که این ویژگی با موفقیت بین 0 و 1 تغییر مقیاس داده است:



MinMaxScaler کتابخانه scikit-learn دو راه را برای تغییر مقیاس یک ویژگی ارائه می‌‌دهد. اول این‌که از fit برای محاسبه حداقل و حداکثر مقادیر ویژگی استفاده می‌‌کند، سپس از transform برای تغییر مقیاس ویژگی استفاده می‌‌کند. راه دوم استفاده از fit\_transform برای انجام همزمان هر دو عملیات است. هیچ تفاوت ریاضیاتی بین این دو راه وجود ندارد، اما گاهی اوقات جدانگه‌داشتن عملیات‌‌ها عملاً سودمند است، زیرا می‌‌توانیم تبدیل یکسانی را در مجموعه داده‌‌های مختلف اعمال کنیم.

**مطالعۀ بیشتر**

* [مقیاس‌بندی ویژگی، ویکی‌پدیا](http://bit.ly/2Fuug2Z)
* [درباره مقیاس‌بندی و نرمال‌سازی ویژگی، سباستین راشکا](http://bit.ly/2FwwRcM)

**4.2 استانداردسازی ویژگی**

**مسئله**

می‌خواهید یک ویژگی را طوری تبدیل کنید که میانگین 0 و انحراف معیار 1 داشته باشد.

**راه‌حل**

StandardScaler کتابخانه scikit-learn هر دو تبدیل را انجام می‌‌دهد:



array([[-0.76058269],

[-0.54177196],

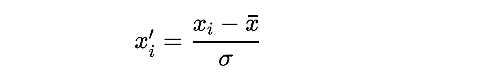
[-0.35009716],

[-0.32271504],

[ 1.97516685]])

**بحث**

روش جایگزین رایجی برای مقیاس‌بندی حداقل-حداکثر که در دستورالعمل 4.1 بررسی شد تغییر مقیاس ویژگی‌‌ها به صورت توزیع تقریباً استاندارد و نرمال است. بدین منظور، از استانداردسازی برای تبدیل داده‌‌ها به‌گونه‌‌ای استفاده می‌‌کنیم که میانگین x̄ صفر و انحراف معیار σ یک باشد. به‌ویژه، هر جزء در ویژگی به‌گونه‌‌ای تبدیل می‌‌شود که:



که در آن x'i فرم استانداردشدۀ ما از xi است. ویژگی تبدیل‌شده تعداد انحراف معیار را نشان می‌‌دهد که مقدار اصلی از مقدار میانگین ویژگی فاصله دارد (که در آمار به آن z-score نیز می‌‌گویند).

استانداردسازی روش رایج مقیاس‌پذیری برای پیش‌پردازش یادگیری ماشین است، و بنا به تجربه‌ام، بیشتر از مقیاس‌گذاری حداقل-حداکثر استفاده می‌شود. با این حال، این روش به الگوریتم یادگیری بستگی دارد. مثلاً، تحلیل مؤلفه‌‌های اصلی[[24]](#footnote-24) اغلب با استفاده از استانداردسازی بهتر عمل می‌‌کند، در حالی که مقیاس‌بندی حداقل-حداکثر برای شبکه‌‌های عصبی توصیه می‌‌شود (هر دو الگوریتم بعداً در این کتاب بررسی خواهند شد). عموماً، به صورت پیش‌فرض استانداردسازی را توصیه می‌کنم، مگر این‌که دلیل خاصی برای استفاده از روش جایگزینی داشته باشید.

می توانیم با مشاهده میانگین و انحراف معیار خروجی راه‌حل خود، تأثیر استانداردسازی را ببینیم:

# Print mean and standard deviation

print("Mean:", round(standardized.mean()))

print("Standard deviation:", standardized.std())

i i

Mean: 0.0

Standard deviation: 1.0

اگر داده‌های ما مقادیر پرت قابل‌توجهی داشته باشند، ممکن است با تأثیر بر میانگین و واریانس ویژگی، بر استانداردسازی ما تأثیر منفی بگذارد. در این سناریو، اغلب بهتر است ویژگی را با استفاده از محدوده میانه و چارک تغییر مقیاس دهید. در کتابخانه scikit-learn، این کار را با استفاده از روش RobustScaler انجام می‌‌دهیم:

# Create scaler

robust\_scaler = preprocessing.RobustScaler()

# Transform feature

robust\_scaler.fit\_transform(x)

array([[ -1.87387612],

[ -0.875 ],

[ 0. ],

[ 0.125 ],

[ 10.61488511]])

**4.3 نرمال‌سازی مشاهدات**

**مسئله**

می‌خواهید مقادیر ویژگی مشاهدات را تغییر مقیاس دهید تا نُرم واحدی داشته باشید (طول کل 1).

**راه‌حل**

از Normalizer با آرگومان norm استفاده کنید:

# Load libraries

import numpy as np

from sklearn.preprocessing import Normalizer

# Create feature matrix

features = np.array([[0.5, 0.5],

[1.1, 3.4],

[1.5, 20.2],

[1.63, 34.4],

[10.9, 3.3]])

# Create normalizer

normalizer = Normalizer(norm="l2")

# Transform feature matrix

normalizer.transform(features)

array([[ 0.70710678, 0.70710678],

[ 0.30782029, 0.95144452],

[ 0.07405353, 0.99725427],

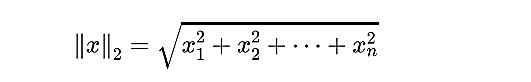
[ 0.04733062, 0.99887928],

[ 0.95709822, 0.28976368]])

**بحث**

بسیاری از روش‌های تغییر مقیاس (مثلاً، مقیاس‌بندی حداقل-حداکثر و استانداردسازی) بر اساس ویژگی‌ها عمل می‌کنند؛ با این حال، می‌‌توانیم در خود مشاهدات و نمونه‌ها نیز تغییر مقیاس دهیم. Normalizer مقادیر مشاهدات را دوباره تغییر مقیاس می‌‌دهد تا نُرم واحدی داشته باشد (مجموع طول آنها 1 می‌شود). این نوع تغییر مقیاس اغلب زمانی استفاده می‌شود که ویژگی‌های مشابه زیادی داشته باشیم (مثلاً، دسته‌بندی متن زمانی که هر کلمه یا گروه‌های n کلمه‌ای به عنوان ویژگی در نظر گرفته شود).

Normalizer سه گزینه norm را ارائه می‌‌دهد که نرم اقلیدسی[[25]](#footnote-25) (اغلب L2 نامیده می‌‌شود) آرگومان پیش‌فرض است:



که در آن x نمونه مشاهده و xn مقدار آن مشاهده برای ویژگی nام است.

# Transform feature matrix

features\_l2\_norm = Normalizer(norm="l2").transform(features)

# Show feature matrix

features\_l2\_norm

array([[ 0.70710678, 0.70710678],

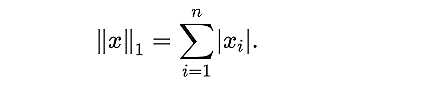
[ 0.30782029, 0.95144452],

[ 0.07405353, 0.99725427],

[ 0.04733062, 0.99887928],

[ 0.95709822, 0.28976368]])

به‌علاوه، می‌توانیم نرم منهتن[[26]](#footnote-26) (L1) را به صورت زیر مشخص کنیم:



# Transform feature matrix

features\_l1\_norm = Normalizer(norm="l1").transform(features)

# Show feature matrix

features\_l1\_norm

n

array([[ 0.5 , 0.5 ],

[ 0.24444444, 0.75555556],

[ 0.06912442, 0.93087558],

[ 0.04524008, 0.95475992],

[ 0.76760563, 0.23239437]])

به‌صورت تجربی، نرم L2 را می‌‌توان فاصله بین دو نقطه در نیویورک برای یک پرنده (یعنی خط مستقیم) در نظر گرفت، در حالی که L1 را می‌‌توان فاصله یک انسان در حال راه رفتن در خیابان در نظر گرفت (یک بلوک به سمت شمال، یک بلوک به سمت شرق، یک بلوک به سمت شمال، یک بلوک به سمت شرق، و غیره راه می‌‌رود)، به همین دلیل است که به آن «نرم منهتن» یا «نرم تاکسی‌[[27]](#footnote-27)» می‌گویند.

در عمل، توجه داشته باشید که norm=’l1’ مقادیر مشاهده را تغییر مقیاس می‌دهد تا مجموع آنها به 1 برسد که گاهی اوقات ممکن است ویژگی مطلوبی باشد:

# Print sum

print("Sum of the first observation\'s values:",

features\_l1\_norm[0, 0] + features\_l1\_norm[0, 1])

Sum of the first observation's values: 1.0

**4.4 ایجاد ویژگی‌‌های چند جمله‌‌ای و تعاملی**

**مسئله**

می‌خواهید ویژگی‌‌های چند جمله‌‌ای و تعاملی را ایجاد کنید.

**راه‌حل**

هرچند برخی تصمیم می‌‌گیرند که ویژگی‌‌های چند جمله‌‌ای و تعاملی را دستی ایجاد کنند، scikit-learn یک تابع توکار[[28]](#footnote-28) را برای انجام این کار ارائه می‌‌دهد:

# Load libraries

import numpy as np

from sklearn.preprocessing import PolynomialFeatures

# Create feature matrix

features = np.array([[2, 3],

[2, 3],

[2, 3]])

# Create PolynomialFeatures object

polynomial\_interaction = PolynomialFeatures(degree=2,

include\_bias=False)

# Create polynomial features

polynomial\_interaction.fit\_transform(features)

array([[ 2., 3., 4., 6., 9.],

[ 2., 3., 4., 6., 9.],

[ 2., 3., 4., 6., 9.]])

پارامتر degree حداکثر درجه چندجمله‌‌ای را تعیین می‌‌کند. مثلاً، degree=2 ویژگی‌‌های جدیدی را ایجاد می‌‌کند که نهایتا تا توان دوم افزایش یافته‌اند:



در حالی که degree=3 ویژگی‌‌های جدیدی را ایجاد می‌‌کند که نهایتا تا توان سوم افزایش یافته‌اند (توان سوم و همه توان‌های کمتر از آن):



افزون بر این، به‌طور پیش‌فرض، PolynomialFeatures دربرگیرندۀ ویژگی‌های تعاملی[[29]](#footnote-29) است:



با تنظیم interaction\_only روی True، می‌توانیم ویژگی‌های ایجادشده را فقط به ویژگی‌های تعاملی محدود کنیم:

interaction = PolynomialFeatures(degree=2,

interaction\_only=True, include\_bias=False)

interaction.fit\_transform(features)

array([[ 2., 3., 6.],

[ 2., 3., 6.],

[ 2., 3., 6.]])

**بحث**

ویژگی‌های چندجمله‌ای اغلب زمانی ایجاد می‌شوند که بخواهیم این مفهوم را لحاظ کنیم که رابطه‌ای غیرخطی بین ویژگی‌ها و هدف وجود دارد. مثلاً، ممکن است فکر کنیم که تأثیر سن بر احتمال ابتلا به بیماری خاصی در طول زمان ثابت نیست، اما با افزایش سن افزایش می‌یابد. می‌توانیم این اثر غیرثابت را در یک ویژگی، x، با ایجاد فرم‌های مرتبه‌بالاتر آن ویژگی (x2، x3، و غیره) رمزگذاری کنیم.

افزون بر این، اغلب با موقعیت‌‌هایی مواجه می‌‌شویم که تأثیر یک ویژگی به ویژگی دیگر وابسته است. مثلاً، اگر بخواهیم پیش‌بینی کنیم که قهوه ما شیرین است یا نه، و دو ویژگی داشتیم: 1) آیا قهوه هم زده شده است یا نه، و 2) آیا شکر اضافه کرده‌ایم. به‌طور جداگانه، هر ویژگی شیرینی قهوه را پیش‌بینی نمی‌کند، اما ترکیبی از آن دو شیرین بودن قهوه را پیش‌بینی می‌کند. یعنی، قهوه فقط در صورتی شیرین می‌‌شود که قهوه شکر داشته باشد و هم‌زده شود. تأثیرات هر ویژگی بر هدف (شیرینی قهوه) به یکدیگر وابسته است. می‌‌توانیم آن رابطه را با گنجاندن یک ویژگی تعاملی رمزگذاری کنیم که حاصل ضرب دو ویژگی مستقل است.

**4.5 تبدیل ویژگی‌ها**

**مسئله**

می‌خواهید نوعی تبدیل سفارشی به یک یا چند ویژگی را انجام دهید.

**راه‌حل**

در scikit-learn، از FunctionTransformer برای اعمال یک تابع به مجموعه‌‌ای از ویژگی‌‌ها استفاده کنید:

# Load libraries

import numpy as np

from sklearn.preprocessing import FunctionTransformer

# Create feature matrix

features = np.array([[2, 3],

[2, 3],

[2, 3]])

# Define a simple function

def add\_ten(x):

return x + 10

# Create transformer

ten\_transformer = FunctionTransformer(add\_ten)

# Transform feature matrix

ten\_transformer.transform(features)



می توانیم همان تبدیل را در pandas با استفاده از apply انجام دهیم:

# Load library

import pandas as pd

# Create DataFrame

df = pd.DataFrame(features, columns=["feature\_1",

"feature\_2"])

# Apply function

df.apply(add\_ten)



**بحث**

معمولاً، ممکن است بخواهید نوعی تبدیل سفارشی به یک یا چند ویژگی را انجام دهید. مثلاً، ممکن است بخواهیم یک ویژگی را ایجاد کنیم که لگاریتم طبیعی مقادیر ویژگی‌‌های مختلف باشد. می‌توانیم این کار را با ایجاد یک تابع و سپس نگاشت آن به ویژگی‌ها با استفاده از FunctionTransformer در کتابخانه scikit-learn یا apply pandas انجام دهیم. در راه‌حل ما، تابع بسیار ساده‌ای به ‌نام add\_ten را ایجاد کردیم که 10 را به هر ورودی اضافه کرد، اما دلیلی ندارد که نتوانیم تابع بسیار پیچیده‌‌تری را تعریف کنیم.

**4.6 تشخیص مقادیر پرت[[30]](#footnote-30)**

**مسئله**

می‌خواهید مشاهدات و نمونه‌های افراطی[[31]](#footnote-31) و دور از توزیع را شناسایی کنید.

**راه‌حل**

متأسفانه، تشخیص مقادیر پرت بیشتر نوعی هنر است تا علم. با این حال، معمولاً فرض می‌کنیم داده‌‌ها توزیع نرمال دارند و بر اساس این فرض، یک بیضی حول داده‌‌ها «ترسیم» می‌‌کنیم، و هر مشاهده‌‌ای را در داخل بیضی به‌عنوان مقدار غیر پرت (با برچسب 1) و هر مشاهده خارج از بیضی را به‌عنوان مقدار پرت (با برچسب 1-) طبقه‌بندی می‌‌کنیم.:



array([-1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

یکی از محدودیت‌های اصلی این رویکرد لزوم تعیین پارامتر contamination است، که درصد مشاهدات و نمونه‌هایی است که داده پرت هستند و این در حالی است که ما این درصد را نمی‌دانیم. Contamination را برآورد ما از پاکی داده‌هایمان در نظر بگیرید. اگر انتظار داشته باشیم داده‌های ما مقادیر پرت کمی داشته باشند، می‌توانیم contamination را به مقداری کوچک تنظیم کنیم. با این حال، اگر معتقد باشیم که داده‌ها به احتمال زیاد مقادیر پرت داشته باشند، می‌توانیم آن را روی مقدار بیشتری تنظیم کنیم.

به جای نگاه‌کردن به کل مشاهدات و نمونه‌ها، می‌توانیم در عوض به ویژگی‌ها به صورت تکی نگاه کنیم و مقادیر افراطی و پرت را در آن ویژگی‌ها با استفاده از دامنۀ بین‌چارکی (IQR) شناسایی کنیم:

(array([0]),)

دامنۀ بین‌ چارکی تفاوت بین چارک اول و سوم مجموعه‌‌ای از داده‌‌ها است. می‌‌توانید دامنۀ بین‌ چارکی را به عنوان انتشار و توزیع بخش عمده‌‌ای از داده‌‌ها در نظر بگیرید، در حالی که مقادیر پرت مشاهداتی هستند که از مرکز اصلی داده‌‌ها فاصله دارند. مقادیر پرت معمولاً به مقادیری گفته می‌شود که مقدار آن‌ها از 1.5 برابر IRQ چارک اول کمتر یا از 1.5 برابر IRQ چارک سوم بیشتر باشد.

**بحث**

تکنیک برتر واحدی برای تشخیص مقادیر پرت وجود ندارد. بلکه، مجموعه‌ای از تکنیک‌ها را داریم که همگی مزایا و معایب خاص خود را دارند. بهترین راهبرد ما اغلب امتحان‌ کردن چندین تکنیک (مثلاً، هم EllipticEnvelope و هم تشخیص مبتنی بر دامنۀ بین‌ چارکی) و نگاه‌کردن به کل نتایج است.

در صورت امکان، باید به مشاهدات و نمونه‌هایی که در زمرۀ مقادیر پرت تشخیص داده‌ایم نگاه بیندازیم و بکوشیم آنها را درک کنیم. مثلاً، اگر مجموعه داده‌‌ای از خانه‌‌ها داشته باشیم و یکی از ویژگی‌‌ها تعداد اتاق‌‌ها باشد، آیا یک مقدار پرت با 100 اتاق واقعاً یک خانه است یا در واقع هتلی است که به‌اشتباه طبقه‌بندی شده است؟

**مطالعۀ بیشتر**

* [سه روش برای تشخیص مقادیر پرت (و منبع تابع دامنۀ بین‌چارکی استفاده‌شده در این دستورالعمل)](http://bit.ly/2FzMC2k)

**4.7 مدیریت مقادیر پرت**

**مسئله**

مقادیر پرتی دارید که باید درباره آن‌ها تصمیم‌گیری کنید.

**راه‌حل**

معمولاً، سه راهبرد داریم که می‌توانیم از آنها برای مدیریت مقادیر پرت استفاده کنیم. ابتدا، می‌‌توانیم آنها را حذف کنیم:

# Load library

import pandas as pd

# Create DataFrame

houses = pd.DataFrame()

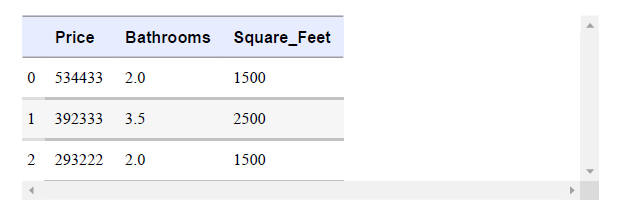
houses['Price'] = [534433, 392333, 293222, 4322032]

houses['Bathrooms'] = [2, 3.5, 2, 116]

houses['Square\_Feet'] = [1500, 2500, 1500, 48000]

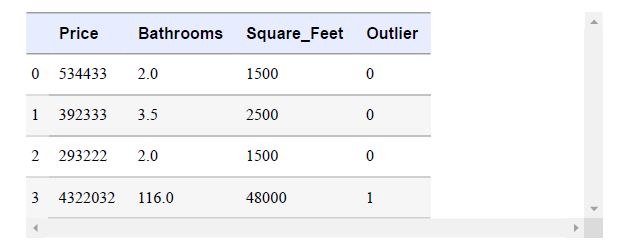
# Filter observations

houses[houses['Bathrooms'] < 20]



دوم، می‌توانیم آنها را در زمرۀ مقادیر پرت برچسب بزنیم، ولی در زمرۀ ویژگی‌‌ها لحاظ کنیم:





در نهایت، می‌توانیم این ویژگی را تبدیل کنیم تا اثر مقدار پرت را کاهش دهیم:

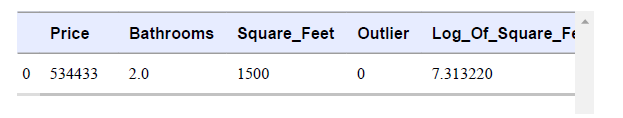
# Log feature

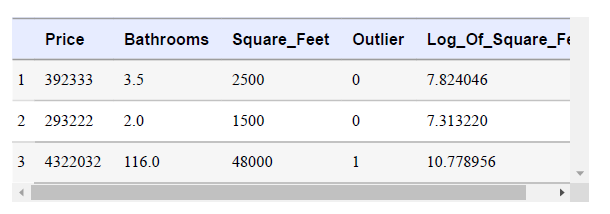
houses["Log\_Of\_Square\_Feet"] = [np.log(x) for x in

houses["Square\_Feet"]]

# Show data

houses





**بحث**

مشابه تشخیص مقادیر پرت، هیچ قانون سفت‌و‌سختی برای برخورد با آنها وجود ندارد. نحوه برخورد با آنها باید بر اساس دو جنبه باشد. اول، باید در نظر بگیریم که چه چیزی آنها را به مقدار پرت تبدیل می‌‌کند. اگر فکر کنیم که آنها خطاهایی در داده‌ها هستند، مانند یک حسگر شکسته یا مقداری که به ‌اشتباه کدگذاری شده، ممکن است مشاهده را حذف یا مقادیر پرت را با NaN جایگزین کنیم، زیرا نمی‌توانیم به آن مقادیر باور داشته باشیم تا از آن‌ها استفاده کنیم. با این حال، اگر بدانیم که مقادیر پرت مقادیر واقعی هستند و مقدار افراطی دارند (مثلاً، یک خانه [عمارت] با 200 حمام)، طبقه‌بندی آنها در زمرۀ مقادیر پرت یا تبدیل مقادیر آنها روش مناسب‌‌تری است.

دوم، نحوه برخورد ما با مقادیر پرت باید بر اساس هدف ما برای یادگیری ماشین باشد. مثلاً، اگر بخواهیم قیمت خانه را بر اساس ویژگی‌های خانه پیش‌بینی کنیم، ممکن است به‌طور منطقی فرض کنیم که قیمت عمارت‌هایی با بیش از 100 حمام بر اساس روش متفاوتی نسبت به قیمت خانه‌های خانوادگی معمولی تعیین شود. افزون بر این، اگر در حال آموزش مدلی هستیم که از آن در بخشی از یک اپلیکیشن آنلاین وام مسکن استفاده کنیم، ممکن است فرض کنیم که کاربران بالقوه ما میلیاردرهایی نیستند که به دنبال خرید عمارتی‌اند.

پس اگر مقادیر پرت داشته باشیم، چه باید بکنیم؟ فکر کنید که چرا آنها مقادیر پرت هستند، هدفی نهایی برای داده‌‌ها در ذهن داشته باشید، و مهم‌تر از همه، به یاد داشته باشید که عدم پرداختن به مقادیر پرت عواقبی دارد.

نکته‌ای دیگر: اگر مقادیر پرتی دارید، ممکن است استانداردسازی روش مناسبی نباشد، زیرا میانگین و واریانس ممکن است بسیار تحت تأثیر مقادیر پرت باشد. در این صورت، از روش تغییر مقیاس‌ قوی‌تری در برابر مقادیر پرت مانند RobustScaler استفاده کنید.

**مطالعۀ بیشتر**

* [مستندات RobustScaler](http://bit.ly/2DcgyNT)

**4.8 گسسته‌سازی ویژگی‌ها**

**مسئله**

یک ویژگی عددی دارید و می‌‌خواهید آن را به مقادیر گسسته بشکنید.

**راه‌حل**

بسته به این‌که چگونه می‌‌خواهیم داده‌‌ها را بشکنیم، می‌‌توانیم از دو تکنیک استفاده کنیم. ابتدا، می‌توانیم ویژگی را با توجه به یک مقدار آستانه‌ باینریزه کنیم[[32]](#footnote-32):



array([[0],

[0],

[1],

[1],

[1]])

دوم، می‌‌توانیم ویژگی‌‌های عددی را با توجه به آستانه‌‌های متعدد تقسیم کنیم:

# Bin feature

np.digitize(age, bins=[20,30,64])

array([[0],

[0],

[1],

[2],

[3]])

توجه داشته باشید که آرگومان‌‌های پارامتر bins لبه سمت چپ هر ستون را نشان می‌‌دهند. مثلاً، آرگومان 20 جزئی با مقدار 20 را دربر نمی‌‌گیرد، فقط دو مقدار کوچک‌تر از 20 را شامل می‌‌شود. می‌‌توانیم این رفتار را با تنظیم پارامتر right روی True تغییر دهیم:

# Bin feature

np.digitize(age, bins=[20,30,64], right=True)

array([[0],

[0],

[0],

[2],

[3]])

**بحث**

گسسته‌سازی زمانی ممکن است راهبرد مفیدی باشد که دلیلی برای این باور داشته باشیم که یک ویژگی عددی باید بیشتر شبیه ویژگی رسته‌ای باشد. برای مثال، ممکن است باور داشته باشیم که تفاوت بسیار اندکی در عادات خرج‌کرد افراد 19 و 20 ساله وجود دارد، اما تفاوت قابل‌توجهی بین افراد 20 و 21 ساله وجود دارد (سنی در ایالات متحده که بزرگسالان جوان می‌‌توانند الکل مصرف کنند). در این مثال، تقسیم افراد در داده‌های ما به افرادی که می‌توانند الکل بنوشند و افرادی که نمی‌توانند ممکن است مفید باشد. به همین ترتیب، در موارد دیگر، گسسته‌سازی داده‌های خود در سه یا چند مقدار گسسته ممکن است مفید باشد.

در راه‌حل ارائه شده، شاهد دو روش گسسته‌سازی بودیم -Binarizer کتابخانه scikit-learn برای گسسته‌سازی دومقداری و digitize کتابخانه numpy برای گسسته‌سازی سه یا چندمقداری؛ با این حال، می‌توانیم از digitize مانند Binarizer برای گسسته‌سازی دومقداری که تنها با تعیین یک مقدار آستانه انجام می‌شود نیز استفاده کنیم:

# Bin feature

np.digitize(age, bins=[18])

array([[0],

[0],

[1],

[1],

[1]])

**مطالعۀ بیشتر**

* [مستندات digitize](http://bit.ly/2HSciFP)

**4.9 گروه‌بندی مشاهدات با استفاده از خوشه‌بندی**

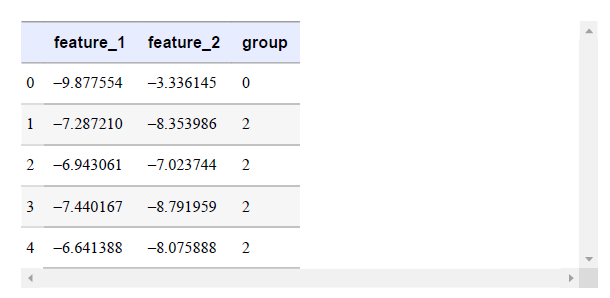
**مسئله**

می‌خواهید مشاهدات را خوشه‌بندی کنید تا مشاهدات مشابه با هم گروه‌بندی شوند.

**راه‌حل**

اگر می‌دانید که k گروه دارید، می‌توانید از خوشه‌بندی k-means برای گروه‌بندی مشاهدات مشابه استفاده کرده و یک ویژگی جدید ایجاد کنید که مشحص کند هر مشاهده متعلق به کدام گروه است:





**بحث**

کمی از خودمان جلوتر می‌رویم و در ادامه کتاب، به الگوریتم‌های خوشه‌بندی بیشتر خواهیم پرداخت. با این حال، می‌‌خواستم به این نکته اشاره کنم که می‌‌توانیم از خوشه‌بندی به‌عنوان مرحله پیش‌پردازش استفاده کنیم. به‌ویژه، از الگوریتم‌های یادگیری بدون نظارت مانند k-means برای خوشه‌بندی مشاهدات در گروه‌ها استفاده می‌کنیم. نتیجه نهایی آن یک ویژگی رسته‌ای است که مقدار آن برای مشاهدات مشابه یکسان است (مشاهدات مشابه همه متعلق به یک گروه هستند).

اگر در حال حاضر همه اینها را متوجه نشدید، نگران نباشید: فقط این ایده را کنار بگذارید که می‌‌توان از خوشه‌بندی در پیش‌پردازش استفاده کرد. اگر واقعاً نمی‌توانید صبر کنید، هم‌اکنون به فصل 19 بروید.

**4.10 حذف مشاهدات با مقادیر مفقود**

**مسئله**

باید مشاهدات حاوی مقادیر مفقود را حذف کنید.

**راه‌حل**

حذف مشاهدات با مقادیر مفقود با یک خط کد هوشمندانه‌ از numpy کار آسانی است:



array([[ 1.1, 11.1],

[ 2.2, 22.2],

[ 3.3, 33.3],

[ 4.4, 44.4]])

به‌علاوه، می‌توانیم مشاهدات مفقود را با استفاده از pandas حذف کنیم:

# Load library

import pandas as pd

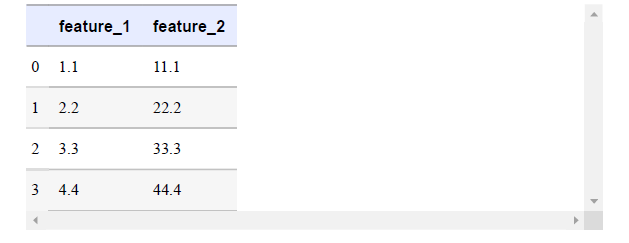
# Load data

dataframe = pd.DataFrame(features, columns=["feature\_1",

"feature\_2"])

# Remove observations with missing values

dataframe.dropna()



**بحث**

بیشتر الگوریتم‌های یادگیری ماشین نمی‌توانند مقادیر مفقود را در آرایه‌های هدف و ویژگی مدیریت کنند. به همین دلیل، نمی‌توانیم مقادیر مفقود را در داده‌های خود نادیده بگیریم و باید در طول پیش‌پردازش به این مشکل رسیدگی کنیم.

ساده‌‌ترین راه‌حل حذف کامل هر مشاهده‌ای است که حاوی یک یا چند مقدار مفقود است، کاری که با استفاده از numpy یا pandas به ‌سرعت و به ‌راحتی انجام می‌‌شود.

البته ما زیاد مشتاق نیستیم که مشاهدات با مقادیر مفقود را حذف کنیم. حذف‌کردن آنها مانند انفجار هسته‌ای است، زیرا الگوریتم ما دسترسی به اطلاعات موجود در مقادیر غیرمفقود مشاهده را نیز از دست می‌‌دهد.

به همان اندازه مهم است که، بسته به علت مقادیر مفقود، حذف‌کردن مشاهدات ممکن است سوگیری[[33]](#footnote-33) را در داده‌‌های ما ایجاد کند. سه نوع داده مفقود وجود دارند:

مقدار مفقودی که کاملا به صورت تصادفی مفقود شده است[[34]](#footnote-34)(MCAR):

احتمال مفقود شدن یک مقدار مستقل از همه چیز است. برای مثال، یک پاسخ‌دهنده به نظرسنجی قبل از پاسخ دادن به یک سؤال، تاس می‌اندازد: اگر شش بیاورد، از آن سؤال صرف‌نظر می‌کند.

مقدار مفقودی که تصادفی مفقود شده است[[35]](#footnote-35)(MAR):

احتمال مفقود شدن یک مقدار کاملاً تصادفی نیست، بلکه به اطلاعات جمع‌آوری‌شده در سایر ویژگی‌ها بستگی دارد. مثلاً، در یک نظرسنجی، در مورد هویت جنسیتی و حقوق سالانه سؤال می‌‌شود، و زنان بیشتر از سؤال حقوق صرف‌نظر می‌‌کنند؛ با این حال، پاسخ‌ ندادن آنها فقط به اطلاعاتی بستگی دارد که در ویژگی هویت جنسی خود جمع‌آوری‌ کرده‌ایم.

مفقود غیرتصادفی[[36]](#footnote-36)(MNAR):

احتمال مفقود شدن یک مقدار تصادفی نیست و به اطلاعاتی بستگی دارد که در ویژگی‌‌های ما جمع‌آوری‌ نشده است. مثلاً، در یک نظرسنجی، در مورد هویت جنسیتی سؤال می‌‌شود، و زنان به ‌احتمال بیشتری از سوال حقوق صرف‌نظر می‌‌کنند، و ویژگی هویت جنسیتی در داده‌‌های خود نداریم.

گاهی اوقات، حذف‌کردن مشاهدات در صورتی که MCAR یا MAR باشند قابل‌قبول است. با این حال، اگر مقدار MNAR باشد، این‌که مقداری مفقود است خودش نوعی اطلاعات است. حذف‌کردن مشاهدات MNAR ممکن است سوگیری را به داده‌های ما وارد کند، زیرا مشاهدات حاصل از نوعی اثر سیستماتیک مشاهده‌نشده را حذف می‌کنیم.

**مطالعۀ بیشتر**

* [شناسایی سه نوع داده مفقود](http://bit.ly/2Fto4bx)
* [مقداردهی/جانهی داده‌‌های مفقود](http://bit.ly/2FAkKLI)

**4.11 جانهی[[37]](#footnote-37) مقادیر مفقود**

**مسئله**

مقادیر مفقودی در داده‌‌های خود دارید و می‌‌خواهید مقادیر آنها را پر یا پیش‌‌بینی کنید.

**راه‌حل**

اگر مقدار کمی داده دارید، مقادیر مفقود را با استفاده از الگوریتم (KNN) پیش‌بینی کنید:



True Value: 0.8730186114

Imputed Value: 1.09553327131

به‌علاوه، می‌توانیم از ماژول Imputer کتابخانه scikit-learn برای پر کردن مقادیر مفقود با میانگین، میانه یا پرتکرارترین مقدار ویژگی استفاده کنیم. با این حال، معمولاً به نتایج بدتری نسبت به الگوریتم KNN می‌‌رسیم:



True Value: 0.8730186114

Imputed Value: -3.05837272461

**بحث**

دو راهبرد اصلی برای جایگزینی داده‌‌های مفقود با مقادیر جایگزین وجود دارند، که هر کدام نقاط قوت و ضعفی دارند. اول، می‌‌توانیم از یادگیری ماشین برای پیش‌بینی مقادیر داده‌‌های مفقود استفاده کنیم. بدین منظور، ویژگی با مقادیر مفقود را برداری هدف در نظر می‌‌گیریم و از زیرمجموعۀ باقی‌ماندۀ ویژگی‌‌ها برای پیش‌بینی مقادیر مفقود استفاده می‌‌کنیم. در حالی که می‌توانیم از الگوریتم‌های یادگیری ماشین مختلفی برای جانهی مقادیر استفاده کنیم، روش محبوب برای این کار استفاده از الگوریتم KNN است. الگوریتم KNN بعداً در فصل 14 بررسی می‌‌شود، اما توضیح کوتاه این است که الگوریتم از مشاهدات k-نزدیک‌ترین (طبق نوعی معیار فاصله) برای پیش‌بینی مقدار مفقود استفاده می‌کند. در راه‌حل خود، مقدار مفقود را با استفاده از پنج مشاهدۀ نزدیک‌تر پیش‌بینی کردیم.

نقطه ضعف الگوریتم KNN این است که برای این‌که بدانیم کدام مشاهدات به مقدار مفقود نزدیک‌ترند، باید فاصله بین مقدار مفقود و هر مشاهده را محاسبه کنیم. این مورد در مجموعه داده‌‌های کوچک‌تر معقول است، اما اگر یک مجموعه داده میلیون‌‌ها مشاهده داشته باشد، به‌سرعت مشکل‌ساز می‌‌شود.

راهبرد جایگزین و مقیاس‌پذیرتر آن است که تمام مقادیر مفقود را با مقداری متوسط پر کنیم. مثلاً، در راه‌حل خود، از scikit-learn برای پر کردن مقادیر مفقود با مقدار میانگین یک ویژگی استفاده کردیم. مقدار جانهی‌شده اغلب به اندازه زمانی که از الگوریتم KNN استفاده کردیم به مقدار واقعی نزدیک نیست، اما می‌توانیم روش «پر کردن مقادیر مفقود با مقدار میانگین» را برای داده‌های حاوی میلیون‌ها مشاهده به‌راحتی مقیاس‌گذار کنیم.

اگر از جانهی استفاده می‌‌کنیم، بهتر است یک ویژگی باینری را ایجاد کنیم که نشان دهد آیا مشاهده حاوی مقدار جانهی‌شده‌ای است یا خیر.

**مطالعۀ بیشتر**

* [مطالعۀ الگوریتم KNN به‌عنوان نوعی روش جانهی](http://bit.ly/2HS9sAT)

1. logfile [↑](#footnote-ref-1)
2. toy [↑](#footnote-ref-2)
3. Tab [↑](#footnote-ref-3)
4. - Parquet [↑](#footnote-ref-4)
5. - Apache Parquet documentation [↑](#footnote-ref-5)
6. - Apache Avro documentation [↑](#footnote-ref-6)
7. - Querying a SQLite Database [↑](#footnote-ref-7)
8. - query [↑](#footnote-ref-8)
9. - S3 Bucket [↑](#footnote-ref-9)
10. - object [↑](#footnote-ref-10)
11. Data wrangling [↑](#footnote-ref-11)
12. First-class [↑](#footnote-ref-12)
13. Regular expressions [↑](#footnote-ref-13)
14. Missing values [↑](#footnote-ref-14)
15. mutable [↑](#footnote-ref-15)
16. immutable [↑](#footnote-ref-16)
17. - subpopulations [↑](#footnote-ref-17)
18. Looping [↑](#footnote-ref-18)
19. loop [↑](#footnote-ref-19)
20. Concatenating [↑](#footnote-ref-20)
21. Merge [↑](#footnote-ref-21)
22. Inner join [↑](#footnote-ref-22)
23. Outer join [↑](#footnote-ref-23)
24. Principal Component Analysis [↑](#footnote-ref-24)
25. Euclidean norm [↑](#footnote-ref-25)
26. Manhattan norm [↑](#footnote-ref-26)
27. Taxicab norm [↑](#footnote-ref-27)
28. Built-in method [↑](#footnote-ref-28)
29. توضیح مترجم: ویژگی‌های تعاملی ویژگی‌هایی هستند که از ترکیب ویژگی‌های دیگر محاسبه می‌شوند. به عنوان مثال ویژگی x1x2 از ترکیب و تعامل دو ویژگی x1 و x2 ایجاد شده است. [↑](#footnote-ref-29)
30. outliers [↑](#footnote-ref-30)
31. extreme [↑](#footnote-ref-31)
32. توضیح مترجم: یعنی اگر مقدار ویژگی از مقدار آستانه کمتر بود صفر و در غیر این صورت مقدار آن ویژگی 1 در نظر گرفته می‌شود. [↑](#footnote-ref-32)
33. bias [↑](#footnote-ref-33)
34. Missing Completely At Random (MCAR) [↑](#footnote-ref-34)
35. Missing At Random (MAR) [↑](#footnote-ref-35)
36. Missing Not At Random (MNAR) [↑](#footnote-ref-36)
37. imputing [↑](#footnote-ref-37)