KAPITEL 7

Excel-Dateien mit pandas manipulieren

Nach sechs Kapiteln intensiver Einführungen in Tools, Python und pandas gönne ich Ihnen eine Pause und beginne dieses Kapitel mit einer praktischen Fallstudie, in der Sie Ihre neu erworbenen Kenntnisse in die Tat umsetzen können: Mit lediglich zehn Zeilen pandas-Code fassen Sie Dutzende von Excel-Dateien zu einem Excel-Bericht zusammen, den Sie an Ihre Vorgesetzten schicken könnten. Im Anschluss an diese Fallstudie gebe ich Ihnen eine ausführlichere Einführung in die Tools, die pandas für den Umgang mit Excel-Dateien bereitstellt: die Funktion `read_excel` und die Klasse `ExcelFile` zum Lesen sowie die Methode `to_excel` und die Klasse `ExcelWriter` zum Schreiben von Excel-Dateien. Da sich pandas nicht auf die Excel-Anwendung stützt, um Excel-Dateien zu lesen und zu schreiben, lassen sich sämtliche Codebeispiele in diesem Kapitel überall dort ausführen, wo Python läuft, einschließlich Linux.

Fallstudie: Excel-Berichte

Diese Fallstudie ist von einigen realen Berichtsprojekten inspiriert, an denen ich in den letzten Jahren beteiligt war. Auch wenn die Projekte aus völlig unterschiedlichen Branchen – einschließlich Telekommunikation, digitales Marketing und Finanzwesen – stammen, sind sie sich doch bemerkenswert ähnlich: Der Ausgangspunkt ist üblicherweise ein Verzeichnis mit Excel-Dateien, die zu einem Excel-Bericht verarbeitet werden müssen – oftmals auf monatlicher, wöchentlicher oder täglicher Basis. Im Begleit-Repository finden Sie im Verzeichnis *sales_data* Excel-Dateien mit fiktiven Verkaufstransaktionen für einen Telekommunikationsanbieter, der verschiedene Tarife (Bronze, Silber, Gold) in einigen Geschäften in den Vereinigten Staaten verkauft. Für jeden Monat gibt es zwei Dateien, eine im Unterordner *new* für neue Verträge und eine im Unterordner *existing* für Bestandskunden. Da die Berichte aus verschiedenen Systemen kommen, sind ihre Formate unterschiedlich: Die Neukunden sind in *.xlsx*-Dateien erfasst, während für die Bestandskunden das ältere *.xls*-Format verwendet wird. Die einzelnen Dateien umfassen jeweils bis zu 10.000 Transaktionen, und unser Ziel ist es, einen Excel-Bericht zu erstellen, der den Gesamtumsatz pro Geschäft und Monat zeigt. Werfen

Sie zunächst einen Blick auf die Datei *January.xlsx* im Unterordner *new* (siehe Abbildung 7-1).

	A	B	C	D	E	F	G
1	transaction_id	store	status	transaction_date	plan	contract_type	amount
2	abfbdd6d	Chicago	ACTIVE	1/1/2019	Silver	NEW	14.25
3	136a9997	San Francisco	ACTIVE	1/1/2019	Gold	NEW	19.35
4	c6688f32	San Francisco	ACTIVE	1/1/2019	Bronze	NEW	12.2
5	6ef349c1	Chicago	ACTIVE	1/1/2019	Gold	NEW	19.35
6	22066f29	San Francisco	ACTIVE	1/1/2019	Silver	NEW	14.25

Abbildung 7-1: Die ersten Zeilen der Datei January.xlsx

Die Excel-Dateien im Unterordner *existing* sehen praktisch genauso aus, außer dass ihnen die Statusspalte fehlt und sie im alten *xls*-Format gespeichert sind. In einem ersten Schritt lesen wir die neuen Transaktionen von Januar mit der Funktion `read_excel` von pandas:

```
In [1]: import pandas as pd

In [2]: df = pd.read_excel("sales_data/new/January.xlsx")
        df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9493 entries, 0 to 9492
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   transaction_id    9493 non-null   object
 1   store             9493 non-null   object
 2   status            9493 non-null   object
 3   transaction_date  9493 non-null   datetime64[ns]
 4   plan              9493 non-null   object
 5   contract_type     9493 non-null   object
 6   amount            9493 non-null   float64
dtypes: datetime64[ns](1), float64(1), object(5)
memory usage: 519.3+ KB
```

Die Funktion read_excel mit Python 3.9

Dies ist die gleiche Warnung wie in Kapitel 5: Wenn Sie `pd.read_excel` mit Python 3.9 oder höher ausführen, müssen Sie mindestens pandas 1.2 verwenden, da Sie andernfalls beim Lesen von *xlsx*-Dateien eine Fehlermeldung erhalten.

Wie Sie sehen können, hat pandas die Datentypen aller Spalten richtig erkannt, einschließlich des Datumsformats von `transaction_date`. Damit können Sie ohne weitere Vorbereitungen mit den Daten arbeiten. Da dieses Beispiel bewusst einfach gehalten ist, können wir gleich ein kurzes Skript namens *sales_report_pandas.py* erstellen, das Beispiel 7-1 zeigt. Dieses Skript liest alle Excel-Dateien aus beiden Verzeichnissen ein, aggregiert die Daten und schreibt die Zusammenfassungstabelle in

eine neue Excel-Datei. Schreiben Sie das Skript mit VS Code selbst oder öffnen Sie das aus dem Begleit-Repository. Zur Erinnerung: Wie Sie Dateien in VS Code erstellen oder öffnen, hat Kapitel 2 erläutert. Wenn Sie es selbst erstellen, sollten Sie es neben dem Ordner *sales_data* speichern – so können Sie das Skript ausführen, ohne die Dateipfade anpassen zu müssen.

Beispiel 7-1: Das Skript sales_report_pandas.py

```
from pathlib import Path

import pandas as pd

# Verzeichnis dieser Datei
this_dir = Path(__file__).resolve().parent  ❶

# Alle Excel-Dateien aus allen Unterordnern von sales_data einlesen.
parts = []
for path in (this_dir / "sales_data").rglob("*.xls*"):  ❷
    print(f'Reading {path.name}')
    part = pd.read_excel(path, index_col="transaction_id")
    parts.append(part)

# Den DataFrame von jeder Datei zu einem einzigen DataFrame kombinieren.
# pandas übernimmt es, die Spalten ordnungsgemäß auszurichten.
df = pd.concat(parts)

# Jeden store in eine Spalte pivotieren und alle Transaktionen pro Datum summieren.
pivot = pd.pivot_table(df,
                       index="transaction_date", columns="store",
                       values="amount", aggfunc="sum")

# Resampling zum Ende des Monats und einen Indexnamen zuweisen.
summary = pivot.resample("M").sum()
summary.index.name = "Month"

# Zusammenfassungsbericht in Excel-Datei schreiben.
summary.to_excel(this_dir / "sales_report_pandas.xlsx")
```

❶ Bis zu diesem Kapitel habe ich Dateipfade in Strings angegeben. Wenn Sie stattdessen die Klasse `Path` aus dem Modul `pathlib` der Standardbibliothek verwenden, stehen Ihnen leistungsfähige Tools zur Verfügung: Mit Pfadobjekten können Sie die Pfade leicht konstruieren, indem Sie einzelne Bestandteile über Schrägstriche miteinander verketten, wie es vier Zeilen darunter bei `this_dir / "sales_data"` geschieht. Diese Pfade funktionieren plattformübergreifend und erlauben Ihnen, Filter wie `rglob` anzuwenden, wie unter dem nächsten Punkt erläutert. Der Ausdruck `__file__` wird zum Pfad der Quellcodedatei aufgelöst, wenn Sie das Skript ausführen – mit der Methode `parent` erhalten Sie demnach den Verzeichnisnamen dieser Datei. Die vor dem Aufruf von `parent` eingeschobene Methode `resolve` wandelt den Pfad in einen *absoluten Pfad* um. Möchten Sie dieses Skript stattdessen von einem Jupyter Notebook aus ausführen, müssen Sie diese Zeile durch `this_dir = Path(".")`.

`resolve()` ersetzen, wobei der Punkt das aktuelle Verzeichnis darstellt. In den meisten Fällen akzeptieren Funktionen und Klassen, die einen Pfad in Form eines Strings entgegennehmen, auch ein Pfadobjekt.

❷ Mit der Methode `rglob` des Pfadobjekts ist es am einfachsten, alle Excel-Dateien aus einem bestimmten Verzeichnis rekursiv einzulesen. Der Name `glob` stammt aus der Unix-Welt und bezeichnet eine (»globale«) Erweiterung von Pfadnamen, die Platzhalterzeichen enthalten. Das Fragezeichen (`?`) vertritt als Platzhalterzeichen genau ein Zeichen, während das Sternchen (`*`) für beliebig viele (auch null) Zeichen steht. Das `r` in `rglob` bezieht sich auf die *rekursive* Ausführung, d. h., `rglob` sucht nach übereinstimmenden Dateien in allen Unterverzeichnissen – dementsprechend ignoriert `glob` Unterverzeichnisse. Wenn man `*.xls*` als Suchmuster für `rglob` angibt, werden Excel-Dateien sowohl im alten als auch im neuen Format gefunden, da das Suchmuster sowohl `.xls` als auch `.xlsx` einschließt. In der Regel empfiehlt es sich, mit dem leicht erweiterten Suchausdruck `[!~$]*.xls*` temporäre Excel-Dateien (deren Namen mit `~$` beginnt) zu ignorieren. Mehr Hintergrundinformationen zum Globbing in Python finden Sie in der Python-Dokumentation (*https://oreil.ly/fY0qG*).

Führen Sie das Skript aus, indem Sie zum Beispiel in VS Code rechts oben auf die Schaltfläche *Datei ausführen* klicken. Es dauert einen Moment, bis das Skript fertig abgearbeitet ist. Anschließend ist die Excel-Arbeitsmappe *sales_report_pandas.xlsx* im selben Verzeichnis wie das Skript zu sehen. Abbildung 7-2 zeigt den Inhalt des Tabellenblatts *Sheet1*. Für lediglich zehn Zeilen Code ist das ein beeindruckendes Ergebnis – selbst wenn Sie noch die Breite der ersten Spalte anpassen müssen, um die Datumswerte zu sehen!

	A	B	C	D	E	F	G
1	**Month**	**Boston**	**Chicago**	**Las Vegas**	**New York**	**an Francisc**	**ashington DC**
2	#########	21784.1	51187.7	23012.75	49872.85	58629.85	14057.6
3	#########	21454.9	52330.85	25493.1	46669.85	55218.65	15235.4
4	#########	20043	48897.25	23451.1	41572.25	52712.95	14177.05
5	#########	18791.05	47396.35	22710.15	41714.3	49324.65	13339.15
6	#########	18036.75	45117.05	21526.55	40610.4	47759.6	13147.1
7	#########	21556.25	49460.45	21985.05	47265.65	53462.4	14284.3
8	#########	19853	47993.8	23444.3	40408.3	50181.6	14161.5
9	#########	22332.9	50838.9	24927.65	45396.85	55336.35	16127.05
10	#########	19924.5	49096.25	24410.7	42830.6	49931.45	14994.4
11	#########	16550.95	42543.8	22827.5	34090.05	44311.65	12846.7
12	#########	21312.9	52011.6	24860.25	46959.85	55056.45	14057.6
13	#########	19722.6	49355.1	24535.75	42364.35	50933.45	14702.15

Abbildung 7-2: Die Excel-Datei sales_report_pandas.xlsx (in der erzeugten Form, ohne die Spaltenbreiten anzupassen)

Für simple Fälle wie diesen bietet pandas eine wirklich einfache Lösung, um mit Excel-Dateien zu arbeiten. Aber es geht noch besser – immerhin würden ein Titel, ein paar Formatierungen (einschließlich der Spaltenbreiten und einer einheitlichen Anzahl von Dezimalstellen) sowie ein Diagramm nicht schaden. Genau darum kümmern wir uns im nächsten Kapitel. Dann greifen wir direkt auf die Writer-Bibliotheken zu, die pandas hinter den Kulissen verwendet. Vorher aber sehen wir uns genauer an, wie man Excel-Dateien mit pandas lesen und schreiben kann.

Excel-Dateien mit pandas lesen und schreiben

Der Code der Fallstudie verwendet die Funktionen `read_excel` und `to_excel` mit ihren Standardargumenten, um die Dinge einfach zu halten. In diesem Abschnitt stelle ich Ihnen die gängigsten Argumente und Optionen für das Lesen und Schreiben von Excel-Dateien mit pandas vor. Los geht es mit der Funktion `read_excel` und der Klasse `ExcelFile`, bevor wir uns die Methode `to_excel` und die Klasse `ExcelWriter` ansehen. Nebenbei führe ich auch die `with`-Anweisung von Python ein.

Die Funktion read_excel und die Klasse ExcelFile

Die Fallstudie verwendet Excel-Arbeitsmappen, bei denen die Daten komfortabel in Zelle A1 des ersten Tabellenblatts lagen. In der Praxis sind Ihre Excel-Dateien wahrscheinlich nicht so gut organisiert. Hierfür bietet pandas Parameter, um den Lesevorgang zu optimieren. Für die nächsten Beispiele verwenden wir die Datei *stores.xlsx*, die Sie im Ordner *xl* des Begleit-Repositorys finden. Abbildung 7-3 zeigt das erste Tabellenblatt.

	A	B	C	D	E	F
1						
2		Store	Employees	Manager	Since	Flagship
3		New York	10	Sarah	7/20/2018	FALSE
4		San Francisco	12	Neriah	11/2/2019	MISSING
5		Chicago	4	Katelin	1/31/2020	
6		Boston	5	Georgiana	4/1/2017	TRUE
7		Washington DC	3	Evan		FALSE
8		Las Vegas	11	Paul	1/6/2020	FALSE
9						

2019 | 2020 | 2019-2020

Abbildung 7-3: Das erste Tabellenblatt der Arbeitsmappe stores.xlsx

Mit den Parametern `sheet_name`, `skiprows` und `usecols` können Sie pandas den Bereich der Zellen mitteilen, den Sie einlesen wollen. Wie üblich empfiehlt es sich, mit der Methode `info` die Datentypen des zurückgegebenen DataFrame zu inspizieren.

```
In [3]: df = pd.read_excel("xl/stores.xlsx",
                           sheet_name="2019", skiprows=1, usecols="B:F")
        df
```

```
Out[3]:            Store  Employees    Manager      Since Flagship
        0       New York         10      Sarah 2018-07-20    False
        1  San Francisco         12     Neriah 2019-11-02  MISSING
        2        Chicago          4    Katelin 2020-01-31      NaN
        3         Boston          5  Georgiana 2017-04-01     True
        4  Washington DC          3       Evan        NaT    False
        5      Las Vegas         11       Paul 2020-01-06    False

In [4]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Store      6 non-null      object
 1   Employees  6 non-null      int64
 2   Manager    6 non-null      object
 3   Since      5 non-null      datetime64[ns]
 4   Flagship   5 non-null      object
dtypes: datetime64[ns](1), int64(1), object(3)
memory usage: 368.0+ bytes
```

Alles sieht gut aus – bis auf die Spalte `Flagship`, denn ihr Datentyp sollte `bool` und nicht `object` sein. Dies lässt sich mit einer Konverterfunktion korrigieren, die die verantwortlichen Zellen in dieser Spalte umwandelt (anstatt die Funktion `fix_missing` zu schreiben, hätten wir auch einen Lambda-Ausdruck bereitstellen können):

```
In [5]: def fix_missing(x):
            return False if x in ["", "MISSING"] else x

In [6]: df = pd.read_excel("xl/stores.xlsx",
                           sheet_name="2019", skiprows=1, usecols="B:F",
                           converters={"Flagship": fix_missing})
        df

Out[6]:            Store  Employees    Manager      Since  Flagship
        0       New York         10      Sarah 2018-07-20     False
        1  San Francisco         12     Neriah 2019-11-02     False
        2        Chicago          4    Katelin 2020-01-31     False
        3         Boston          5  Georgiana 2017-04-01      True
        4  Washington DC          3       Evan        NaT     False
        5      Las Vegas         11       Paul 2020-01-06     False

In [7]: # Die Spalte Flagship hat jetzt den Datentyp "bool".
        df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Store      6 non-null      object
 1   Employees  6 non-null      int64
 2   Manager    6 non-null      object
 3   Since      5 non-null      datetime64[ns]
```

```
 4   Flagship   6 non-null      bool
dtypes: bool(1), datetime64[ns](1), int64(1), object(2)
memory usage: 326.0+ bytes
```

Die Funktion `read_excel` akzeptiert auch eine Liste von Tabellenblattnamen. In diesem Fall gibt sie ein Wörterbuch mit dem DataFrame als Wert und dem Namen des Tabellenblatts als Schlüssel zurück. Um alle Tabellenblätter einzulesen, müssen Sie `sheet_name=None` setzen. Beachten Sie auch, dass ich hier in `usecols` die Spaltennamen der Tabelle aufzähle:

```
In [8]: sheets = pd.read_excel("xl/stores.xlsx", sheet_name=["2019", "2020"],
                               skiprows=1, usecols=["Store", "Employees"])
        sheets["2019"].head(2)

Out[8]:            Store  Employees
        0       New York         10
        1  San Francisco         12
```

Wenn die Quelldatei keine Spaltenüberschriften enthält, setzen Sie `header=None` und geben die Überschriften in `names` an. Denken Sie daran, dass `sheet_name` auch Blattindizes akzeptiert:

```
In [9]: df = pd.read_excel("xl/stores.xlsx", sheet_name=0,
                           skiprows=2, skipfooter=3,
                           usecols="B:C,F", header=None,
                           names=["Branch", "Employee_Count", "Is_Flagship"])
        df

Out[9]:           Branch  Employee_Count Is_Flagship
        0       New York              10        False
        1  San Francisco              12      MISSING
        2        Chicago               4          NaN
```

Um mit `NaN`-Werten umzugehen, verwenden Sie eine Kombination aus `na_values` und `keep_default_na`. Das nächste Beispiel weist pandas an, nur Zellen mit dem Wort `MISSING` als `NaN` zu interpretieren und sonst nichts:

```
In [10]: df = pd.read_excel("xl/stores.xlsx", sheet_name="2019",
                            skiprows=1, usecols="B,C,F", skipfooter=2,
                            na_values="MISSING", keep_default_na=False)
         df

Out[10]:           Store  Employees Flagship
         0       New York         10    False
         1  San Francisco         12      NaN
         2        Chicago          4
         3         Boston          5     True
```

Mit der Klasse `ExcelFile` bietet pandas eine alternative Möglichkeit, Excel-Dateien zu lesen. Dies macht vor allem dann einen Unterschied, wenn Sie mehrere Tabellenblätter aus einer Datei im veralteten *xls*-Format einlesen möchten: In diesem Fall ist die Verwendung von `ExcelFile` schneller, da sie pandas daran hindert, die gesamte Datei mehrmals einzulesen. Die Klasse `ExcelFile` lässt sich auch als Kontextmanager (siehe Kasten) verwenden, sodass die Datei wieder richtig geschlossen wird.

Kontextmanager und die with-Anweisung

Zunächst einmal hat die `with`-Anweisung in Python nichts mit der `With`-Anweisung in VBA zu tun: In VBA dient sie dazu, eine Reihe von Anweisungen auf demselben Objekt auszuführen, während sie in Python verwendet wird, um Ressourcen wie Dateien oder Datenbankverbindungen zu verwalten. Wenn Sie die neuesten Verkaufsdaten laden möchten, um sie analysieren zu können, müssen Sie möglicherweise eine Datei öffnen oder eine Verbindung zu einer Datenbank herstellen. Nachdem Sie die Daten gelesen haben, empfiehlt es sich, die Datei oder die Verbindung sobald wie möglich wieder zu schließen. Andernfalls kann es passieren, dass Sie keine weitere Datei öffnen oder keine weitere Verbindung zur Datenbank einrichten können – Dateihandler und Datenbankverbindungen sind begrenzte Ressourcen. Eine Textdatei öffnen und schließen Sie manuell wie folgt (wobei `w` für das Öffnen der Datei im Modus Schreiben – `write` – steht und die Datei ersetzt wird, falls sie bereits existiert):

```
In [11]: f = open("output.txt", "w")
         f.write("etwas Text")
         f.close()
```

Dieser Code erzeugt eine Datei *output.txt* im selben Verzeichnis wie das Notebook, aus dem Sie den Code ausführen, und schreibt »etwas Text« in die Datei. Um eine Datei zu *lesen*, geben Sie den Modus `r` (für `read`) anstelle von `w` an, und um etwas an das Ende der Datei *anzufügen*, verwenden Sie `a` (für `append`). Da sich Dateien auch von außerhalb Ihres Programms manipulieren lassen, könnte eine derartige Operation fehlschlagen. Für solche Fälle ist der `try/except`-Mechanismus vorgesehen, den ich in Kapitel 11 vorstellen werde. Da diese Operation jedoch so häufig vorkommt, können Sie mit der `with`-Anweisung in Python die Dinge vereinfachen:

```
In [12]: with open("output.txt", "w") as f:
             f.write("Some text")
```

Wenn die Codeausführung den Rumpf der `with`-Anweisung verlässt, wird die Datei automatisch geschlossen, egal ob eine Ausnahmesituation aufgetreten ist oder nicht. Damit ist garantiert, dass die Ressourcen ordnungsgemäß aufgeräumt werden. Objekte, die die `with`-Anweisung unterstützen, bezeichnet man als *Kontextmanager*; hierzu gehören die `ExcelFile`- und `ExcelWriter`-Objekte dieses Kapitels sowie die Datenbankverbindungsobjekte, die Sie in Kapitel 11 kennenlernen werden.

Sehen Sie sich jetzt die Klasse `ExcelFile` in Aktion an:

```
In [13]: with pd.ExcelFile("xl/stores.xls") as f:
             df1 = pd.read_excel(f, "2019", skiprows=1, usecols="B:F", nrows=2)
             df2 = pd.read_excel(f, "2020", skiprows=1, usecols="B:F", nrows=2)

             df1

Out[13]:            Store  Employees Manager      Since Flagship
```

```
0       New York         10   Sarah 2018-07-20    False
1  San Francisco         12  Neriah 2019-11-02 MISSING
```

Über `ExcelFile` können Sie auch auf die Namen aller Tabellenblätter zugreifen:

```
In [14]: stores = pd.ExcelFile("xl/stores.xlsx")
         stores.sheet_names

Out[14]: ['2019', '2020', '2019-2020']
```

Schließlich ist es mit pandas möglich, Excel-Dateien von einer URL einzulesen, ähnlich wie wir es mit CSV-Dateien in Kapitel 5 getan haben. So liest der folgende Code eine Datei direkt aus dem Begleit-Repository ein:

```
In [15]: url = ("https://raw.githubusercontent.com/fzumstein/"
                "python-for-excel/1st-edition/xl/stores.xlsx")
         pd.read_excel(url, skiprows=1, usecols="B:E", nrows=2)

Out[15]:            Store  Employees Manager      Since
         0       New York         10   Sarah 2018-07-20
         1  San Francisco         12  Neriah 2019-11-02
```

xlsb-Dateien über pandas lesen

Wenn Sie pandas in einer Version unterhalb von 1.3 verwenden, müssen Sie beim Lesen von *xlsb*-Dateien in der Funktion `read_excel` oder der Klasse `ExcelFile` die Engine explizit angeben:

```
pd.read_excel("xl/stores.xlsb", engine="pyxlsb")
```

Dazu ist es erforderlich, das Paket `pyxlsb` zu installieren, da es nicht Teil von Anaconda ist – darauf und auf die anderen Engines geht das nächste Kapitel ein.

Als Zusammenfassung zeigt Tabelle 7-1 die gebräuchlichsten Parameter der Funktion `read_excel`. Die vollständige Liste finden Sie in der offiziellen Dokumentation (*https://oreil.ly/v8Yes*).

Tabelle 7-1: Ausgewählte Parameter für read_excel

Parameter	Beschreibung
`sheet_name`	Anstelle eines Blattnamens können Sie auch den Index des Blatts (nullbasiert) angeben, z. B. `sheet_name=0`. Wenn Sie `sheet_name=None` setzen, liest pandas die gesamte Arbeitsmappe ein und gibt ein Wörterbuch in Form von `{"sheetname": df}` zurück. Um eine Auswahl von Tabellenblättern zu lesen, übergeben Sie eine Liste mit Tabellenblattnamen oder -indizes.
`skiprows`	Damit können Sie die angegebene Anzahl von Zeilen überspringen.
`usecols`	Wenn die Excel-Datei die Namen der Spaltenüberschriften enthält, geben Sie diese in einer Liste an, um die Spalten auszuwählen, z. B. `["Store", "Employees"]`. Alternativ können Sie auch eine Liste von Spaltenindizes übergeben, z. B. `[1, 2]`, oder einen String (keine Liste!) von Excel-Spaltennamen, einschließlich Bereichen, z. B. `"B:D,G"`. Es lässt sich ebenfalls eine Funktion übergeben: Um zum Beispiel nur die Spalten einzuschließen, die mit `Manager` beginnen, verwenden Sie `usecols=lambda x: x.startswith("Manager")`.
`nrows`	Anzahl der Zeilen, die Sie lesen möchten.

Tabelle 7-1: Ausgewählte Parameter für read_excel (Fortsetzung)

Parameter	Beschreibung
`index_col`	Gibt als Spaltenname oder Index an, welche Spalte der Index sein soll, z. B. `index_col=0`. Wenn Sie eine Liste mit mehreren Spalten übergeben, wird ein hierarchischer Index erzeugt.
`header`	Wenn Sie `header=None` setzen, werden die standardmäßigen Ganzzahlen als Überschriften zugewiesen, es sei denn, Sie geben die gewünschten Namen über den Parameter `names` an. Wenn Sie eine Liste von Indizes übergeben, werden hierarchische Spaltenüberschriften erstellt.
`names`	Die gewünschten Namen Ihrer Spalten als Liste bereitstellen.
`na_values`	pandas interpretiert die folgenden Zellenwerte standardmäßig als `NaN` (in Kapitel 5 eingeführt): leere Zellen, `#NA`, `NA`, `null`, `#N/A`, `N/A`, `NaN`, `n/a`, `-NaN`, `1.#IND`, `nan`, `#N/A N/A`, `-1.#QNAN`, `- nan`, `NULL`, `-1.#IND`, `<NA>`, `1.#QNAN`. Wenn Sie dieser Liste einen oder mehrere Werte hinzufügen möchten, geben Sie sie über `na_values` an.
`keep_default_na`	Möchten Sie die Standardwerte ignorieren, die pandas als `NaN` interpretiert, setzen Sie `keep_default_na=False`.
`convert_float`	Excel speichert alle Zahlen intern als Gleitkommazahlen, und standardmäßig wandelt pandas alle Zahlen ohne signifikante Dezimalstellen in Ganzzahlen um. Wenn Sie dieses Verhalten ändern möchten, setzen Sie `convert_float=False` (was Berechnungen etwas schneller machen kann).
`converters`	Ermöglicht Ihnen, für jede Spalte eine Funktion bereitzustellen, um die Werte der Spalte zu konvertieren. Zum Beispiel lässt sich mit `converters={"column_name": lambda x: x.upper()}` der Text in einer bestimmten Spalte in Großbuchstaben umwandeln.

So viel zum Lesen von Excel-Dateien mit pandas. Nun wechseln Sie die Seiten und lernen im nächsten Abschnitt, wie Sie Excel-Dateien schreiben.

Die Methode to_excel und die Klasse ExcelWriter

Um mit pandas eine Excel-Datei zu schreiben, geht dies am einfachsten mit der Methode `to_excel` eines DataFrame. Dabei lässt sich festlegen, in welche Zelle welches Tabellenblatts Sie den DataFrame schreiben möchten. Außerdem können Sie entscheiden, ob Sie die Spaltenüberschriften und den Index des DataFrame einschließen möchten und wie Sie mit Datentypen wie `np.nan` und `np.inf` umgehen wollen, die keine äquivalente Darstellung in Excel haben. Erstellen Sie zunächst einen DataFrame mit verschiedenen Datentypen und rufen Sie dann seine Methode `to_excel` auf:

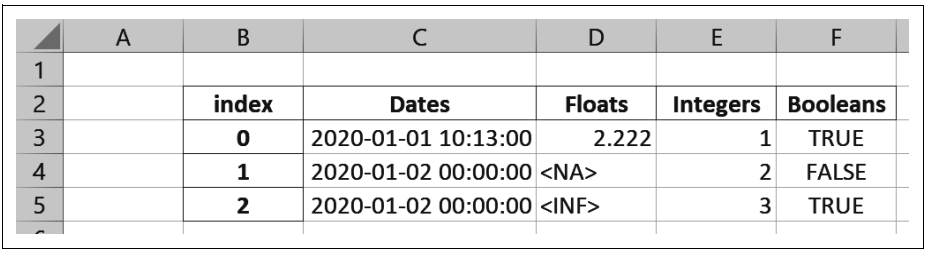
```
In [16]: import numpy as np
         import datetime as dt

In [17]: data = [[dt.datetime(2020,1,1, 10, 13), 2.222, 1, True],
                 [dt.datetime(2020,1,2), np.nan, 2, False],
                 [dt.datetime(2020,1,2), np.inf, 3, True]]
         df = pd.DataFrame(data=data,
                           columns=["Dates", "Floats", "Integers", "Booleans"])
         df.index.name="index"
         df
```

```
Out[17]:                    Dates  Floats  Integers  Booleans
         index
         0      2020-01-01 10:13:00   2.222         1      True
         1      2020-01-02 00:00:00     NaN         2     False
         2      2020-01-02 00:00:00     inf         3      True

In [18]: df.to_excel("written_with_pandas.xlsx", sheet_name="Output",
                     startrow=1, startcol=1, index=True, header=True,
                     na_rep="<NA>", inf_rep="<INF>")
```

Der Befehl `to_excel` erzeugt eine Excel-Datei, wie sie Abbildung 7-4 zeigt (die Spalte C müssen Sie verbreitern, um die Datumswerte richtig zu sehen):



	A	B	C	D	E	F
1						
2		**index**	**Dates**	**Floats**	**Integers**	**Booleans**
3		**0**	2020-01-01 10:13:00	2.222	1	TRUE
4		**1**	2020-01-02 00:00:00	<NA>	2	FALSE
5		**2**	2020-01-02 00:00:00	<INF>	3	TRUE

Abbildung 7-4: Ausschnitt aus der Datei written_with_pandas.xlsx

Wenn Sie mehrere DataFrames in dasselbe Tabellenblatt oder in verschiedene Tabellenblätter schreiben möchten, müssen Sie die Klasse `ExcelWriter` bemühen. Der folgende Beispielcode schreibt denselben DataFrame an zwei verschiedene Stellen auf `Sheet1` und noch einmal auf `Sheet2`:

```
In [19]: with pd.ExcelWriter("written_with_pandas2.xlsx") as writer:
             df.to_excel(writer, sheet_name="Sheet1", startrow=1, startcol=1)
             df.to_excel(writer, sheet_name="Sheet1", startrow=10, startcol=1)
             df.to_excel(writer, sheet_name="Sheet2")
```

Da wir die Klasse `ExcelWriter` als Kontextmanager verwenden, wird die Datei automatisch auf den Datenträger geschrieben, wenn sie den Kontextmanager verlässt, d. h., wenn die Einrückung endet. Andernfalls müssten Sie `writer.save()` explizit aufrufen. Tabelle 7-2 gibt eine Zusammenfassung der gebräuchlichsten Parameter an, die `to_excel` übernimmt. Die vollständige Liste der Parameter finden Sie in der offiziellen Dokumentation (*https://oreil.ly/ESKAG*).

Tabelle 7-2: Ausgewählte Parameter für to_excel

Parameter	Beschreibung
`sheet_name`	Name des Blatts, in das geschrieben werden soll.
`startrow` und `startcol`	`startrow` ist die erste Zeile, in die der DataFrame geschrieben wird, und `startcol` die erste Spalte. Da die Indizierung bei null beginnt, verwenden Sie `startrow=2` und `startcol=1`, wenn Sie Ihren DataFrame in Zelle B3 schreiben wollen.
`index` und `header`	Möchten Sie den Index und/oder die Überschrift verbergen, setzen Sie `index=False` und `header=False`.

Tabelle 7-2: Ausgewählte Parameter für to_excel (Fortsetzung)

Parameter	Beschreibung
`na_rep` und `inf_rep`	Standardmäßig wird `np.nan` in eine leere Zelle konvertiert, während `np.inf`, die NumPy-Darstellung für unendlich, in den String `inf` umgewandelt wird. Dieses Verhalten können Sie ändern, wenn Sie Werte bereitstellen.
`freeze_panes`	Friert die ersten Zeilen und Spalten ein, indem ein Tupel bereitgestellt wird: Zum Beispiel friert `(2, 1)` die beiden ersten Zeilen der ersten Spalte ein.

Wie Sie sehen, funktioniert das Lesen und Schreiben einfacher Excel-Dateien mit pandas sehr gut. Es gibt allerdings auch Grenzen – sehen Sie sich an, welche das sind.

Beschränkungen beim Einsatz von pandas mit Excel-Dateien

Excel-Dateien mithilfe der pandas-Schnittstelle zu lesen und zu schreiben, funktioniert gut in einfachen Fällen, doch es gibt auch Grenzen:

- Wenn Sie DataFrames in Dateien schreiben, können Sie weder eine Titelleiste noch ein Diagramm einbinden.
- Es gibt keine Möglichkeit, das Standardformat von Header und Index in Excel zu ändern.
- Beim Lesen von Dateien transformiert pandas automatisch Zellen mit Fehlern wie `#REF!` oder `#NUM!` in `NaN`, sodass es nicht mehr möglich ist, in Ihren Tabellenblättern nach bestimmten Fehlern zu suchen.
- Um mit großen Excel-Dateien zu arbeiten, können zusätzliche Einstellungen erforderlich sein, die leichter zu kontrollieren sind, indem Sie die Reader- und Writer-Pakete direkt einsetzen, wie das nächste Kapitel zeigt.

Zum Schluss

Das Schöne an pandas ist, dass es eine konsistente Schnittstelle bietet, um mit allen unterstützten Excel-Formaten zu arbeiten – ob nun *xls*, *xlsx*, *xlsm* oder *xlsb*. Dadurch ist es für Sie ein Leichtes, ein Verzeichnis mit Excel-Dateien zu lesen, die Daten zu aggregieren und die Zusammenfassung in einen Excel-Bericht auszugeben – mit lediglich zehn Codezeilen.

Allerdings erledigt pandas die eigentliche Arbeit nicht selbst: Hinter den Kulissen wählt es ein Reader- oder Writer-Paket, das den Job übernimmt. Im nächsten Kapitel zeige ich, welche Reader- und Writer-Pakete pandas verwendet und wie Sie sie direkt oder in Kombination mit pandas nutzen können. So können wir die Einschränkungen umgehen, die der vorherige Abschnitt beschrieben hat.