Module installieren/Updaten

by Woche 5

In diesem Kapitel wollen wir vor allem lernen wie man neue Module installiert und bestehende Module updatet. Das Installieren von Modulen ist in Python eigentlich eine häufige Aufgabe, da Standard-Python an sich nur wenige Module mitbringt. Die meisten Module müssen also nachinstalliert werden. Speziell für uns war es aber eben noch nicht notwendig, da wir Python über Anaconda installiert haben, sodass schon viele Module mitinstalliert wurden.

Thematisch gehört dieses Kapitel dementsprechend auch nicht wirklich in “4 Deskriptive Statistik”, sondern ist eher ein “Allgemeines” Kapitel. Aber aufgrund einer Besonderheit, über die einige von euch im vorangegangenen Kapitel gestolpert sind, gibt es nun doch Anlass an dieser Stelle darauf einzugehen.

Module installieren

Module können auf verschiedene Arten installiert werden. Die einfachste Möglichkeit ist die Verwendung von `pip`, dem Python Package Installer. `pip` ist ein Kommandozeilenprogramm, das mit Python installiert wird und das es ermöglicht, Module vom Python Package Index (PyPI) zu installieren. Letzteres ist ein Software-Verzeichnis und hilft dabei, von der Python-Community entwickelte und geteilte Software zu finden und zu installieren. Modul/Paket-Entwickler benutzen PyPI, um ihre Software zu veröffentlichen.

`pip` kann also über die Kommmandozeile aufgerufen bzw. genutzt werden. Wir können `pip` allerdings auch direkt in unseren Jupyter Notebooks verwenden. Um ein neues Modul zu installieren, müssten wir lediglich folgenden Code ausführen:

```
# Generell
!pip install <modulname>
```

```
# Beispiel für das Modul "numpy"
!pip install numpy
```

Als Beispiel wollen wir das Modul `faker` installieren, welches nicht standardmäßig mit Anaconda installiert wird. (Das Modul `faker` wird verwendet, um Testdaten zu generieren.) Führen wir also den entsprechenden Befehl aus, so vergehen einige Sekunden, bevor folgender Output erscheint:

Wichtig ist, dass am Ende der Ausgabe die Meldung `Successfully installed faker` erscheint. Das bedeutet, dass das Modul erfolgreich installiert wurde. Als Beweis kann geprüft werden ob nun eine der Funktionen des Moduls aufgerufen werden kann, z.B. die zum erzeugen einer E-Mail-Adresse:

```
import faker as fk
fk.Faker().email()
```

```
'cruzkyle@example.com'
```

> **Bestimmte Version installieren**
>
> Man kann auch eine ganz bestimmte Version eines Moduls installieren. Dazu wird der Befehl `pip install` mit dem Parameter `==` und der Versionsnummer genutzt, also `pip install <modulname>==<versionsnummer>`, z.B. `pip install numpy==1.21.2`.

Module updaten

Module können auch aktualisiert werden. Dazu wird ebenfalls `pip` verwendet und der Befehl `install` mit dem Parameter `--upgrade` genutzt. Der Befehl `--upgrade` sorgt dafür, dass das bereits installierte Modul auf die neueste Version aktualisiert wird. Der Befehl sieht also wie folgt aus:

```
# Generell
!pip install --upgrade <modulname>
```

```
# Beispiel für das Modul "numpy"
!pip install --upgrade numpy
```

Auch hier können einige Sekunden vergehen nachdem der Befehl ausgeführt wurde bis der Output erscheint. Und ebenfalls ist es wichtig, dass am Ende der Ausgabe die Meldung `Successfully installed <modulname>` steht.

Seaborn vor und nach v0.13

Einige von euch hatten vermutlich im vorangegangenen Kapitel das Problem, dass `swarmplot` und `boxplot` nicht nebeneinander, sondern übereinander gezeichnet wurden, obwohl ihr denselben Code wie im Online-Material verwendet habt. Das Problem lag vermutlich daran, dass eine ältere Version von `seaborn` bei euch installiert ist[1],

[1] Dies ist nicht eure Schuld, da es ja mit Anaconda installiert wurde und ihr nicht wusstet, dass es eine neuere Version gibt. Tatsächlich ist dies ein sehr gutes Beispiel für ein Problem zum Haare raufen, da es

wohingegen das hiesige Skript mit der aktuellsten Version erzeugt wurde. Um zu prüfen welche Version von `seaborn` bei euch installiert ist, gibt es den `pip`-Befehl `!pip show seaborn`. Alternativ bietet das Modul `seaborn` selbst auch eine Funktion `seaborn.__version__` an, die die Version des Moduls zurückgibt.

```
!pip show seaborn
```

```
Name: seaborn
Version: 0.13.2
Summary: Statistical data visualization
Home-page:
Author:
Author-email: Michael Waskom <mwaskom@gmail.com>
License:
Location: c:\users\username\...
Requires: matplotlib, numpy, pandas
Required-by:
```

```
import seaborn as sns
sns.__version__
```

```
'0.13.2'
```

In speziell diesem Fall macht es einen großen Unterschied, ob `seaborn` in einer Version vor oder ab 0.13 installiert ist. Hier der Vergleich, indem derselbe Code aber mit unterschiedlichen Versionen von `seaborn` ausgeführt wird:

seaborn 0.11.0

seaborn 0.13.2

Ein zumindest damit verwandter Hinweis findet sich übrigens auch in der `seaborn`-Dokumentation bei u.a. den beiden Funktionen `swarmplot` und `boxplot`:

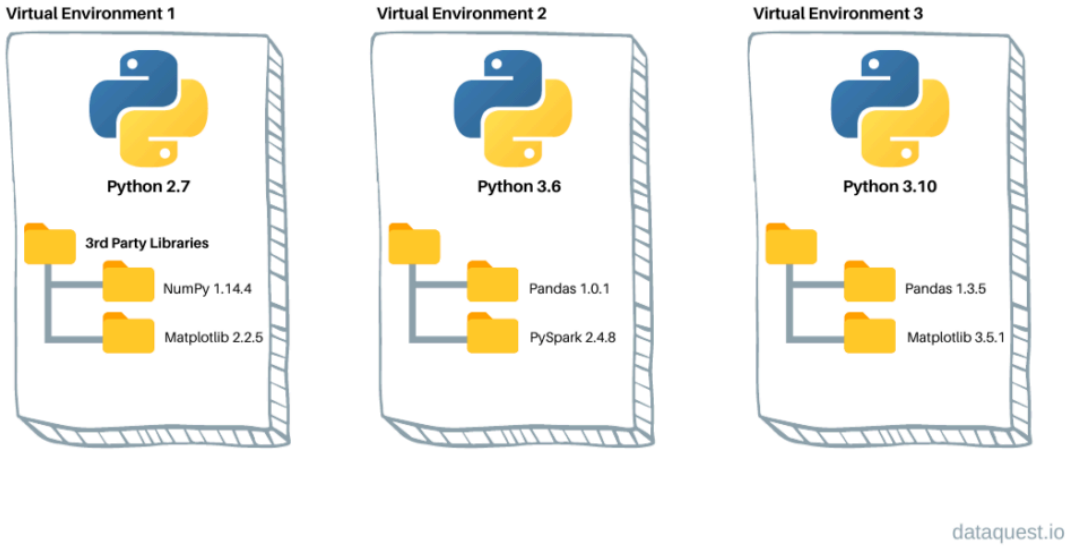
Virtual Environments

Ein weiterer wichtiger Punkt in diesem Zusammenhang sind Virtual Environments (Virtuelle Umgebungen). Dieser Teil ist für euch wohl noch nicht (und für die Prüfung dieses Kurses auch nicht) relevant, aber es ist dennoch wichtig, dass ihr davon gehört habt. Falls ihr mittel- und langfristig mit Python arbeitet, solltet ihr euch das Thema auf die TODO-Liste für nach dem Kurs schreiben.

---

nicht offensichtlich ist, dass es an der Version liegt. Auch ChatGPT und das Internet können ggf. noch nicht die Lösung parat haben, wenn die Version noch so neu ist.

Ein Environment ist eine isolierte Umgebung, in der Python und alle zugehörigen Module installiert sind. Vereinfacht ausgedrückt hat man sozusagen mehrere unterschiedliche Pythons auf einem Rechner, wenn man mehrere Environments einrichtet. Das bedeutet, dass in einem Environment nur die Module in genau den Versionen installiert sind, die für ein bestimmtes Projekt benötigt werden. Environments sind also eine Möglichkeit, verschiedene Projekte voneinander zu trennen und so zu verhindern, dass Module, die für ein Projekt benötigt werden, in einem anderen Projekt stören.



Quelle: Dataquest

Anwendungsbeispiele von Environments sind:

- Ältere Projekte, die auf älteren Versionen von Modulen basieren, können weiterhin ausgeführt werden, ohne dass die neuesten Versionen von Modulen aus anderen Projekte überschrieben werden müssen.
- Projekte, die weniger Module benötigen, müssen nicht alle Module installiert haben, die in anderen Projekten nötig sind.

Ein sehr bekanntes Tool, um Environments zu erstellen, ist `virtualenv`, welches seit Python 3.3 standardmäßig als `venv` installiert ist. Wie man auch in Anacona relativ einfach Environments erstellen kann, wird in folgendem Video gezeigt:

https://www.youtube.com/embed/oYrWkW7ENaU?si=E2-QYNnHBwzTyc5y

Weitere Ressourcen

- Virtuelle PYTHON Umgebung: Deshalb brauchst du sie - virtualenv/venv für Python-Einsteiger erklärt Nicht davon verunsichern lassen, dass im Video Python vorrangig über die Kommandozeile ausgeführt wird. Das ist nur eine andere Art Python zu nutzen, die wir hier nicht verwenden um Einsteigerfreundlich zu bleiben.

Übungen

Installiere wie oben gezeigt das Modul `faker` und prüfe ob es erfolgreich installiert wurde, indem du - ebenfalls wie oben gezeigt - eine E-Mail-Adresse generierst.

- (A) Geschafft

Stelle sicher, dass du die neueste Version von `seaborn` installiert hast. Falls nicht, führe den entsprechenden Befehl aus und prüfe ob die Version nun >=0.13 ist.

- (A) Geschafft