# *R Notebook, Python und UTF-8*

*Jörg Kantel*

*2016-10-24*

## *Python im R Notebook – ein erster Test*

Bei meinen Experimenten mit dem neuen R Notebook habe ich nicht nur dieses, sondern auch das Tufte-Paket[1] für einen schöneren HTML- und PDF-/LaTeX-Output entdeckt. Und so habe ich beides zusammen für meine ersten Experimente mit dem R Notebook und Python genutzt.

Leider verbindet sich das Notebook zur Zeit nur mit dem Python, das R unter `Sys.witch("python")` findet[2], und das ist leider das (System-) Python 2.7.5 von Apple, nicht jedoch mein Anaconda Python 3.5.

Man kann das System jedoch überlisten.

Zwar gibt untenstehender R-Code wie erwartet das System-Python von Apple aus,

[1] `http://rstudio.github.io/tufte/`

[2] Das scheint eine Eigenschaft der Knitr-Language-Engine zu sein.

```
Sys.which("python")
```

```
##            python
## "/usr/bin/python"
```

wenn man jedoch zu Beginn jeder Sitzung

```
Sys.setenv(PATH = "/Users/kantel/anaconda/bin")
Sys.which("python")
```

```
##                              python
## "/Users/kantel/anaconda/bin/python"
```

eingibt, dann findet auch *R Markdown* das Anaconda Python, der unten stehende Python-3-Code wird korrekt ausgeführt[3] und zeigt auch die ausgegebene Liste in UTF-8 mit korrekten Umlauten.

Natürlich muß der Path dem jeweiligen System angepaßt werden und es gibt auch keine Abkürzungen wie ~ oder $. Da der Pfad zum Anaconda-Python auf einem meiner Rechner `/Users/admin/anaconda/bin` und auf einem anderen `/Users/kantel/anaconda/bin` heißt, ist das nicht unproblematisch.

[3] Leider habe ich noch keine Möglichkeit gefunden, wie man dies in einer projektbezogenen Konfigurationsdatei für jedes Projekt permanent machen kann, zur Zeit führe ich den Code mit `echo=FALSE`. einfach zu Beginn eines jeden R-Markdown-Dokuments aus.

```
from collections import Counter
path = "data/aufklaerung.txt"

with open(path, encoding="utf-8", mode="r") as f:
```

```
    text = f.read()
print(text)

text= text.lower()
cntr = Counter(text.split())
liste = cntr.most_common()
print(liste[0:7])
```

```
## Beantwortung der Frage: Was ist Aufklärung?
## Aufklärung ist der Ausgang des Menschen aus seiner selbst verschuldeten Unmündigkeit.
## [('ist', 2), ('der', 2), ('ausgang', 1), ('frage:', 1), ('des', 1), ('aus', 1), ('aufklärung?', 1)]
```

Auch für diesen Output mußte ich das System überlisten: Das Tufte-CSS läßt den Code nämlich nicht breiter werden, als es die Textspalten sind und so entstehen sehr schnell die häßlichen, horizontalen Scrollbalken. Daher habe ich eine `user.css` geschrieben, die als letzte CSS-Datei eingelesen wird und bisher folgenden Code enthält:

```
body {
    background-color: #ffffff; }

pre { width: 90%;
           padding-left: 2.5%;
           overflow-x: auto; }
```

Damit habe ich erstens das in meinen Augen häßliche *beige* durch einen weißen Hintergrund ersetzt und zum anderen den Code-Blöcken eine Weite von 90 Prozent zugewiesen. Das dürfte in den meisten Fällen ausreichen, so daß die vertikalen Scrollbalken nur noch selten auftauchen sollten.

Allerdings muß man nun darauf achten, daß man neben den Code-Ausgaben keine Randnotizen oder Bilder mehr setzen sollte. Sonst entstehen sehr häßliche Lücken zwischen den Absätzen.

Wie gesagt, im Druck sieht alles schicker aus. Der *Tufte-Handout-Style* ist eben für Druckerzeugnisse entworfen und nicht für Online-Publikationen. Das merkt man deutlich.