Übungsprotokoll TOL3

Kocaj Alen

12.09.17

0. Inhaltsverzeichnis

Table of Contents

1. Vorwort

Das Ziel dieses Protokolls ist es, die Aufgabenstellung in TOL, bestehend aus einer Aufgabe im Nukleinsäuren – Bereich und einer Aufgabe aus dem Protein-Bereich sowie deren Umsetzung zu erklären.

Weiters dient dieses Protokoll als Manual für die Umsetzung der beiden Aufgaben. Für beide Aufgaben wurde eigens ein Skript entwickelt, welches relevante Daten aus dem NCBI holt und diese Daten in ein für Statistiken sinnvolles Format bringt. Weiters kann eines dieser beiden Skripte eigens ein Diagramm generieren und dem User anzeigen.

1.1 Aufgaben

Für die Aufgabe im Nukleinsäuren – Bereich wurde die Erfassung von Gewebstypen anhand eines (oder mehreren) übergebenen Genes gewählt. Die Informationen werden unter Verwendung der NCBI Datenbank Unigene abgefragt und die Gewebstypen sowie deren Anzahl aus den jeweiligen Expression Sequence Tags (ESTs) herausgeparst.

Für die Aufgabe im Protein - Bereich wurde die Darstellung von Natural Variants (Mutationen) der Aminosäuren eines (oder mehreren) Proteinen gewählt. Die Informationen werden unter Verwendung der NCBI Datenbank Uniprot abgefragt und die Mutationen herausgeparst.

1.2 Umsetzung

Beide Aufgabe wurden mithilfe eines eigens dafür geschriebenen Python – Skripts gelöst. Das Python – Skript ist in der Lage mithilfe einer vom User übergebenen Liste mit gewählten Informationen, die richtigen Daten aus der Datenbank auszuwählen. Beide Skripts verarbeiten diese Daten anschließend und bringen sie in ein CSV – Format welches mit Excel oder Libre Office Calc importiert werden kann.

Das Skript, welches die Mutationen der Aminosäuren darstellt ist sogar in der Lage, dem Benutzer ein Diagramm in Form einer kachelartigen

Heatmap zu generieren. Eine ähnliche Diagrammart kommt im Contributions – Chart auf Github vor.

1.3 Technische Realisierung

Beide Skripts wurden unter Verwendung von Python 3.6.2 entwickelt, wobei keine nicht – kompatiblen Features mit unteren Versionen verwendet wurden. Das bedeutet, dass Python – Versionen bis zu 3.4 erfolgreich das Skript ausführen können. Idealerweise sollte das Skript mit der neuesten Version ausgeführt werden, um keine unerwarteten Missverständnisse Fehler hervorzurufen.

Das Skript, welches für die Mutationen der Aminosäuren zuständig ist, verlangt für die Darstellung der Heatmap zusätzliche Python Bibliotheken, die installiert werden sollten. Zu beachten ist, dass das Feature zur Graphenerstellung standardmäßig deaktiviert ist, wodurch das Gesamtskript auch ohne Drittbibliotheken ausgeführt werden kann. Für weitere Informationen, sehen Sie bitte in der jeweiligen Skriptbeschreibung nach.

Das Skript wurde ursprünglich auf einem Linux Computer entwickelt, jedoch darauf ausgelegt, dass es auch ohne Probleme in einer Windows – basierten Umgebung laufen kann.

Technische Daten:

Software	Version
Python	3.6.2
Editor	Visual Studio Code
Betriebssystem	Linux (Arch Linux OS)
Betriebssystem	Windows 10
IDE	Keine
Für zusätzliche Features gehören installiert	
Pip (sollte mit Python 3.4 bereits installiert sein)	9.0.1
Matplotlib	2.0.2

Dill	0.2.7.1
Numpy	1.13.1
Pandas	0.20.3

1.3.1 Installation der Python Drittbibliotheken

Innerhalb der zweiten Aufgabe „Mutationen in Proteinen“ gibt es die Möglichkeit, sich Diagramm generieren zu lassen. Um dies zu nutzen müssen die oben erwähnten Python Bibliotheken installiert werden.

Die beste Art und Weise, um Python Bibliotheken zu installieren ist mit dem offiziellen Python Package Manager „pip“.

1.3.2 Installation von Pip

Folgen Sie diesen Anleitungen auf https://pip.readthedocs.io/en/stable/installing/, um den Package Manager erfolgreich zu installieren.

```
radio:Downloads alen$ python3.6 get-pip.py
Requirement already up-to-date: pip in /usr/local/lib/python3.6/site-packages
```

Um Pip auf einem Windows Computer zu installieren, sehen diesen hilfreichen StackOverflow Post:

https://stackoverflow.com/questions/4750806/how-do-i-install-pip-on-windows

1.3.3 Installation der Python Bibliotheken

Ist der Package Manager installiert, so wechseln sie in den Ordner der zweiten Aufgabe und führen sie folgendes aus:

Unter Linux:

```
$ pip install -r requirements.txt
```

Unter Windows:

```
C:\> C:\Python36\Scripts\pip.exe install -r requirements.txt
```

Beides setzt voraus, dass man sich im richten Ordner befindet.

Werden die Packages erfolgreich installiert, so sieht die Ausgabe ungefähr so aus, wie auf diesem Ausschnitt auf einem Linux Computer.

```
radio:02_mutations alen$ pip3 install -r requirements.txt
Collecting dill==0.2.7.1 (from -r requirements.txt (line 3))
  Downloading dill-0.2.7.1.tar.gz (64kB)
    100% |████████████████████████████████| 71kB 686kB/s
Collecting matplotlib==2.0.2 (from -r requirements.txt (line 4))
  Downloading matplotlib-2.0.2-cp36-cp36m-macosx_10_6_intel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.whl (
12.8MB)
    100% |████████████████████████████████| 12.8MB 71kB/s
Collecting numpy==1.13.1 (from -r requirements.txt (line 5))
  Downloading numpy-1.13.1-cp36-cp36m-macosx_10_6_intel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.whl (4.5M
B)
    100% |████████████████████████████████| 4.6MB 196kB/s
Collecting pandas==0.20.3 (from -r requirements.txt (line 6))
  Downloading pandas-0.20.3-cp36-cp36m-macosx_10_12_x86_64.whl (9.3MB)
    100% |████████████████████████████████| 9.3MB 99kB/s
Collecting python-dateutil (from matplotlib==2.0.2->-r requirements.txt (line 4))
  Downloading python_dateutil-2.6.1-py2.py3-none-any.whl (194kB)
    100% |████████████████████████████████| 194kB 2.2MB/s
Collecting six>=1.10 (from matplotlib==2.0.2->-r requirements.txt (line 4))
  Downloading six-1.10.0-py2.py3-none-any.whl
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=1.5.6 (from matplotlib==2.0.2->-r requirements.txt (line 4))
  Downloading pyparsing-2.2.0-py2.py3-none-any.whl (56kB)
    100% |████████████████████████████████| 61kB 2.4MB/s
Collecting pytz (from matplotlib==2.0.2->-r requirements.txt (line 4))
  Downloading pytz-2017.2-py2.py3-none-any.whl (484kB)
    100% |████████████████████████████████| 491kB 1.2MB/s
Collecting cycler>=0.10 (from matplotlib==2.0.2->-r requirements.txt (line 4))
  Downloading cycler-0.10.0-py2.py3-none-any.whl
Building wheels for collected packages: dill
  Running setup.py bdist_wheel for dill ... done
  Stored in directory: /Users/alen/Library/Caches/pip/wheels/e5/88/fe/7e290ce5bb39d531eb9bee5cf254ba1c3e3c7ba3339ce67bee
Successfully built dill
Installing collected packages: dill, numpy, six, python-dateutil, pyparsing, pytz, cycler, matplotlib, pandas
Successfully installed cycler-0.10.0 dill-0.2.7.1 matplotlib-2.0.2 numpy-1.13.1 pandas-0.20.3 pyparsing-2.2.0 python-dateutil-2.6.1 pytz-2017
.2 six-1.10.0
```

1.4 Terminologie

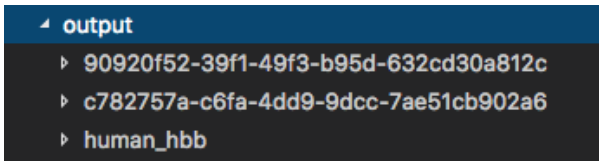
Beide Skripte arbeiten auf die gleiche Art und Weise. Sie bekommen einen Input in Form einer Liste, arbeiten alle Elemente in der Liste zeilenweise ab und speichern die gefundenene Daten akkumuliert ab.

Dieser Vorgang, in dem das Skript einmal aufgerufen und die Daten verarbeitet bzw. abspeichert, nennt sich ein Resultat speichern.

Diese Terminologie ist insofern wichtig, da jedes der beiden Skripte jeweils alle Resultate in einem eigenen Ordnung innerhalb eines Resultat Orderns

absichert. Jeder Ordner wird mit einem eindeutigen Namen versehen, um die Resultate eindeutig zu halten.

Hier ein Beispiel, wie Resultate abgesichert werden:



Dieses Beispiel bezieht sich auf das Skript, welches für das Mutations – Crawling zuständig ist. Das Skript wurde dreimal aufgerufen.

Jedes Mal, wenn das Skript aufgerufen wird, speichert es alle relevanten Daten und Files innerhalb eines für das Resultat erstellten Ordner mit einem eindeutigen Namen. In diesem Beispiel sind dies die UUIDs (90920f52-39f1-49f3-b95d-632cd30a812c) und (c782757a-c6fa-4dd9-9dcc-7ae51cb902a6), welche hierfür generiert wurden.

Neben eindeutigen Namen können ebenso eigenen Namen für das Resultat vergeben werden. In unserem Beispiel ist das das Resultat mit dem Namen "human_hbb". In diesem Ordner befinden sich vermutlich relevante Ergebnisse zu dem Protein Hemoglobin Beta (HBB). Wie die Namen vergeben werden, sehen weiter unten, bei der Beschreibung der jeweiligen Skripte.

Beachten Sie, dass ebenso der Überordner, welches alle Resultat beheimatet, eigens angepasst werden kann. Hierfür wird auf die jeweilige Parametrisierung der einzelnen Skripte verwiesen.

1.5 Ausführung

Um Komplikation beim Einrichten einer IDE zu vermeiden, sollten beide Skripte innerhalb der Command Line (DOS unter Windows, Bash unter Linux) ausgeführt werden.

1.5.1 Ausführung unter Linux:

```
$ python3.6 tissue.py
```

Beachten Sie, dass **python3.6** ebenso **python** heißen könnte, nur durch einen symbolischen Link verbunden. Ebenso setzt dies Voraus, dass man sich im richtigen Ordner befindet.

1.5.2 Ausführung unter Windows:

```
C:\> C:\Users\<ihr Name>\AppData\Local\Programs\Python\Python36\python-3.6.0.exe mutation.py
```

Bitte beachten Sie, dass der Pfad zur Python Installation eventuell zur PATH Variable hinzugefügt geworden ist. Daher würde der folgende Aufruf achten:

```
C:\> python-3.6.0.exe mutation.py
```

Ebenso hier zu beachten ist der Speicherungsort der Python Installation. Wenden Sie sich hierfür an die offizielle Dokumentation zu Python unter Windows (https://docs.python.org/3/using/windows.html).

2. Genexpression nach Gewebe

2.1 Aufgabe

Ziel dieser Aufgabe ist das Erstellen einer Statistik, welches zu einem oder mehreren Genen die verschiedenen Gewebstypen sowie deren Anzahl anzeigt. Die abzufragende NCBI Datenbank hierfür ist Unigene. Dort findet man innerhalb eines Clusters die benötigten Expression Sequence Tags – also kurze Subsequenzen einer cDNA oder eines Genstranskripts – welche mit dem jeweiligen Gewebe verknüpft ist.

2.2 Umsetzung

Das Projekt zur dieser Aufgabe befindet sich im Ordner „01_tissues“.

Zur Abfragung und Specherung der Gewebstypen wurde das Python Skript “tissue.py” erstellt, welches die Unigene Datenbank abfragt. An das Skript übergeben wird eine Textdatei, welches diverse Input Parameter beinhaltet. Ausgebeben werden die diversen Gewebstypen akkumuliert auf der Konsole. Zusätzlich speichert das Skript die gefundenen Gewebstypen innerhalb einer CSV Datei. Zu leichteren Identifizierung wird für jeden Parameter innerhalb des Input Files eines CSV Datei erstellt, welches Meta Daten enthält. Für weitere Informationen, sehen sie weiter unten nach.

2.3 Beispielaufruf

Ein verkürzter Abschnitt eines Beispielaufrufs des Skripts:

```
radio:01 alen$ python3.6 tissue.py unigene-links
[i] starting unigene tissue crawler
[i] reading unigene query from unigene-links
[i] crawling unigene with {'org': 'Hs', 'cid': '523443', 'maxest': 1000000000}

Tissues in human_hbb
====================
adipose tissue: 15
adrenal gland: 4
blood: 641
bone: 23
bone marrow: 177
brain: 96
connective tissue: 67
ear: 1
embryonic tissue: 7
eye: 26
ganglia: 1
heart: 20
intestine: 11
```

```
pituitary gland: 6
placenta: 230
prostate: 19
skin: 11
spleen: 96
stomach: 2
testis: 4
thymus: 5
thyroid: 2
umbilical cord: 1
uncharacterized tissue: 14
uterus: 23
vascular: 2

[i] meta data saved to output/human_hbb/meta.csv
[i] csv saved to output/human_hbb/tissues.csv
```

Hier sieht man wie der Unigene – Crawler die Parameter aus dem Input File "unigene-links" liest und anschließend das Crawling beginnt. Innerhalb des Input Files wurden diverse Parameter übergeben, die der Crawler dafür nimmt, um Unigene abzufragen. Was Parameter darstellen, finden Sie weiter unten.

Nachdem der Content der Unigene Webseite angekommen ist, verarbeitet der Crawler diese und sucht sich die relevanten Informationen; in diesem Fall die Gewebstypen.

Diese gibt er dann auf die Konsole heraus und speichert sie anschließend innerhalb einer CSV Datei – in diesem Fall tissues.csv – ab.

Was das andere File meta.csv darstellt, wird weiter unten erläutert.

Zu beachten ist noch, dass das Resultat innerhalb des Ordners "output" mit dem Namen "human_hbb" abgespeichert wird. Das Benennen eines Resultates, wie zuvor erwähnt, scheint über das Input File möglich zu sein.

2.4 Parametrisierung

2.4.1 Parametrisierung des Input Files

Das Input File, welches von tissue.py eingelesen wird, sieht folgendermaßen aus:

```
> human hbb
[ugid]914190
[org]Hs [cid]523443
[link]https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?ORG=Hs&CID=523443&MAXEST=2363
[org]Mm [cid]1293
```

Als Erstes zu beachten ist die erste Zeile. Ich habe mir erlaubt zur Benennung der Resultate ein ähnliches Format zu wählen, wie es in Fasta Files üblich ist. Diese erste Zeile markiert die Header Line, welche den Namen für das Resultat vergibt. Der Name des Resultates sollte keine Sonderzeichen enthalten, darf aber ruhig aus mehreren Wörter / Leerzeichen bestehen, die zu Unterstrichen aufgelöst werden. Eine komplette Leerzeile wird ignoriert.

In diesem Beispiel wird die Header Line „human hbb“ zu dem Namen „human_hbb“ aufgelöst.

Ich hab lange überlegt, welche Parameter das Skript erwarten sollte und habe mich zu einer flexiblen Parametrisierung entschieden. Das Input File darf folgende Parameter enthalten:

- Ugid: eine eindeutige Unigene ID, welche zu einem Unigene Eintrag zeigt
- Org: die Abkürzung des Organismus (Hs → Homo Sapiens)
- Cid: eine nicht immer eindeutige Cluster ID, welche bereits alle Gewebstypen innerhalb der ESTs anzeigt
- Link: ein eindeutiger Link zu einem Cluster Eintrag im Unigene

Ich geben dem Benutzer also die Möglichkeit, je nach bevorzugtem

Parameter das Skript aufzurufen. Mit dem Link ist es zwar expliziter, jedoch ist man mit einer ugid bzw. einer cid / org um Einiges schneller.

Was die Notation angeht, orientiert sich das Input File an NCBI / Entrez Term Parameter. Boolsche Operationen werden jedoch nicht unterstützt (kommen womöglich in Version 2.0).

Einige Voraussetzungen werden an die Parameter gestellt:

- wird eine Ugid übergeben, dann für das Skript zunächst eine Anfrage an den Unigene Eintrag durch. Anschließend parst das Skript die Cluster ID und den Organismus heraus und führt dann eine zweite Anfrage durch, nur dieses Mal an einen Cluster Eintrag im Unigene.
- wird eine Ugid übergeben, werden alle anderen Parameter in derselben Zeile ignoriert.
- die Übergabe einer Cluster ID setzt einen Organismus voraus und vice versa
- wird ein Link übergeben, werden alle anderen Parameter in derselben Zeile ignoriert.
- an jeden Nicht – Link Request an Unigene wird der Parameter Maxest hinzugefügt, welcher beschreibt wie viel Expression Sequence Tags maximal angezeigt werden dürfen. Standardmäßig wird der Wert auf 1.000.000.000 gesetzt.

Der Benutzer sollte sich an den oben beschriebenen Parametern beziehungsweise Restriktionen halten, um die 100% Funktionsfreiheit des Programs zu gewährleisten.

Um Zusammenzufassen, in der obigen Datei werden für das Resultat mit dem Namen „human_hbb“ vier Anfragen an Unigene geschickt und die jeweils geparsten Gewebstypen mitsamt ihrer Anzahl akkumuliert.

Möchte man z.b das Gene eines Menschen und das einer Maus vergleichen kann das Skript einfach zwei Mal mit einem jeweils anderen Input File aufgerufen werden.

2.4.2 Parametrisierung des Programms

Wie zuvor erwähnt ist es auch möglich das Programm zu parametrisieren. Übliche Anwendungsweisen sind u.a der Name des Input Files, der Name des Überordners für Resultate, etc.

Für das Skript gibts es zwei Art und Weisen, wie man die Parameter festlegen kann.

2.4.3 Parametrisierung im Quellcode

Das Programm kann in der Main Funktion des Skript manuell mit Parameters ausgestattet werden.

```
def main():
    # define application, default program parameter
    app = {
        "input_file": "example",
        "output_dir": "output/",
        "timeout": 15,
        "debug": False,
    }
```

Hier wird ein Dictionary, also ein Key – Value Verzeichnis festgelegt, welches für das gesamte Skript globale Parameter enthält.

- input_file: der Defaultname für das Input File ist „example“, dies kann hier überschrieben werden
- output_dir: der Name des Überordners aller Resultates
- timeout: der Timeout ist die Anzahl der Sekunden, die das Skript maximal wartet für jeden HTTP Request, den es an Unigene schickt. Das NCBI ja bekannt ist, dass es manchmal recht langsam ist, ist es wichtig eine passende Timout Dauer festzusetzen. Wird nach Ablauf des Timeouts immer noch keine Antwort von NCBI erwartet, überspringt das Programm den jeweiligen Unigene Link bzw. Parameter.

- debug: schaltet den Debug Modus ein, worin bei jeder Anfrage an Unigene nicht nur das Directory mit allen Query Parametern angezeigt wird, sondern zu Kontrolle auch der Zusammengesetzte Link. Ein Beispiel ohne Debug Modus:

```
radio:01 alen$ python3.6 tissue.py
[i] starting unigene tissue crawler
[i] reading unigene query from example
[i] crawling unigene with {'ugid': '914190'}
[i] crawling unigene with {'org': 'Hs', 'cid': '523443', 'maxest': 1000000000}
[i] crawling unigene with {'org': 'Hs', 'cid': '523443', 'maxest': 1000000000}
[i] crawling unigene with {'org': 'Hs', 'cid': '523443', 'maxest': 2363}
[i] crawling unigene with {'org': 'Mm', 'cid': '1293', 'maxest': 1000000000}
[i] crawling unigene with {'org': 'Hs', 'cid': '241570', 'maxest': 1000000000}

Tissues in human_hbb
====================
adipose tissue: 45
adrenal gland: 12
```

Selbiges Beispiel mit Debug Modus:

```
radio:01 alen$ python3.6 tissue.py
[i] starting unigene tissue crawler
[i] reading unigene query from example
[i] crawling unigene with {'ugid': '914190'}
[d] URL: https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?ugid=914190
[i] crawling unigene with {'org': 'Hs', 'cid': '523443', 'maxest': 1000000000}
[d] URL: https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?org=Hs&cid=523443&maxest=1000000000
[i] crawling unigene with {'org': 'Hs', 'cid': '523443', 'maxest': 1000000000}
[d] URL: https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?org=Hs&cid=523443&maxest=1000000000
[i] crawling unigene with {'org': 'Hs', 'cid': '523443', 'maxest': 2363}
[d] URL: https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?org=Hs&cid=523443&maxest=2363
[i] crawling unigene with {'org': 'Mm', 'cid': '1293', 'maxest': 1000000000}
[d] URL: https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?org=Mm&cid=1293&maxest=1000000000
[i] crawling unigene with {'org': 'Hs', 'cid': '241570', 'maxest': 1000000000}
[d] URL: https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?org=Hs&cid=241570&maxest=1000000000

Tissues in human_hbb
====================
adipose tissue: 45
adrenal gland: 12
```

2.4.4 Parametrisierung als Kommandozeilenparameter

Entschließt man sich jedoch die Paramatrisierung des Programms aus Sicherheitsgründen vorzunehmen, ermöglicht das Skript ebenso Kommandozeilenparameter an das Skript zu übergeben.

Ein Beispielaufruf:

```
radio:01 alen$ python3.6 tissue.py example resultate 40 -debug
```

Hier wird an das Skript die folgenden Parameter übergeben:

- input_file: example
- output_dir: resultate
- timeout: 40
- debug: True (durch das Übergeben von -debug wird das Flag aktiviert; ansonsten bleibt der Modus ausgeschaltet.)

Beachten Sie, dass die Reihenfolge der Parameter fest vorgegeben ist. Wird eine andere Reihenfolge der Parameter gesetzt, kann das Skript zum Absturz kommen.

2.5 Ausgabe

```
pituitary gland: 6
placenta: 230
prostate: 19
skin: 11
spleen: 96
stomach: 2
testis: 4
thymus: 5
thyroid: 2
umbilical cord: 1
uncharacterized tissue: 14
uterus: 23
vascular: 2

[i] meta data saved to output/human_hbb/meta.csv
[i] csv saved to output/human_hbb/tissues.csv
```

Das Skript ist in der Lage die Gewebstypen sowie deren Anzahl aller übergebenen Gene im Input File auf die Konsole auszugeben. Gleichzeitig speichert das Skript die Daten im CSV File “tissues.csv” ab und erstellt eine

“meta.csv” Datei. Diese beiden Dateien können leicht via Microsoft Office Excel oder Libre Office Calc importiert werden.

2.5.1 tissues.csv

Die Datei tissues.csv wird pro Resultat für alle im Input File übergeben Gene akkumuliert erstellt.

```
> homo sapiens esr1 and egfr
[org]hs [cid]208124
[org]hs [cid]488293
```

In diesem Beispiel wird Unigene für das Gen Estrogen receptor1 (ESR1) und Epidermal growth factor receptor (EGFR) abgefragt.

Die Gewebstypen der beiden Gene werden im Resultatsordner “homo_sapiens_esr1_and_egfr” abgespeichert.

Hier ein verkürzter Abschnitt des erstellten CSV Files.

```
Tissues in homo_sapiens_esr1_and_egfr:
tissue;number
adipose tissue;2
adrenal gland;1
ascites;2
bladder;4
bone;2
brain;33
cervix;3
connective tissue;4
embryonic tissue;4
esophagus;1
eye;17
heart;5
intestine;7
kidney;12
larynx;3
```

2.5.2 meta.csv

Die meta.csv Datei beinhaltet Daten die nicht per-se für Statistiken verwendet werden können, aber dennoch Aufschluss über das Resultat geben.
Damit ist gemeint, dass in der meta.csv zu jedem Gen kontextaussgenden Daten gespeichert sind wie:

- Die UGID
- Cluster_ID
- Organismus
- Name des Genes
- Anzahl der Gewebstypen ingesamt
- Die Parameter im Input File

Zu dem vorherigen Aufruf das dazugehörende abschnittene meta.csv:

```
Meta data for homo_sapiens_esr1_and_egfr:
ugid;cluster_id;organism;tissue_cnt;name;query
163517;208124;Hs;122;Estrogen receptor 1 (ESR1);{'org': 'Hs', 'cid': '208124',
703452;488293;Hs;336;Epidermal growth factor receptor (EGFR);{'org': 'Hs', 'ci
```

Auf wenn diese Daten keine Aussage zu den Gewebstypen bzw. die Anzahl der jeweiligen Gewebsarten macht, so denke ich mir würde es den Nutzer dennoch interessiert, was genau abgefragt worden ist.

Bei z.b einem Aufruf von 100 Genes des Homo Sapiens hat es nicht unbedingt einen Sinn, die Daten mit dem Output der Konsole zu vergleichen.

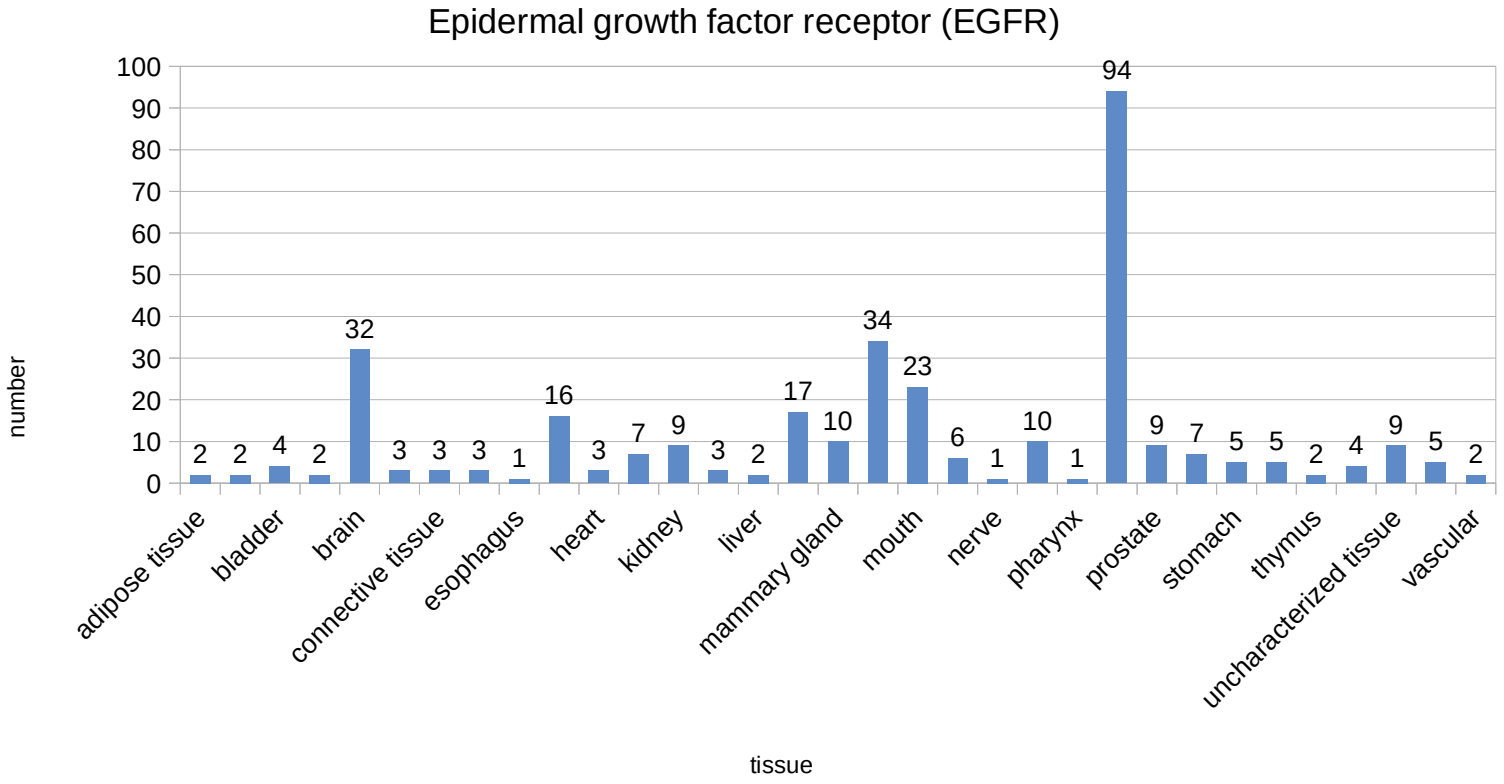
2.6 Auswertung / Statistik

2.6.1 Auswertung mit einem Gen

Name: Epidermal growth factor receptor (EGFR)

URL: https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?ORG=Hs&CID=488293&MAXEST=377

Query line for tissue.py: "[org]hs [cid]488293"



Wie man hier deutlich erkennen kann, kommt "placenta" mit Abstand am Häufigsten in den EST sequences dieses Genes vor. Die einzig Weiteren Gewebstypen, die etwas stark hervorstechen sind "brain" und "mixed". Alle anderen halten sich eher im mittleren Bereich zwischen 1 und 23 auf.

2.6.1 Auswertung mit mehreren Genen

Name:

- Epidermal growth factor receptor (EGFR)
- Epidermal growth factor (EGF)

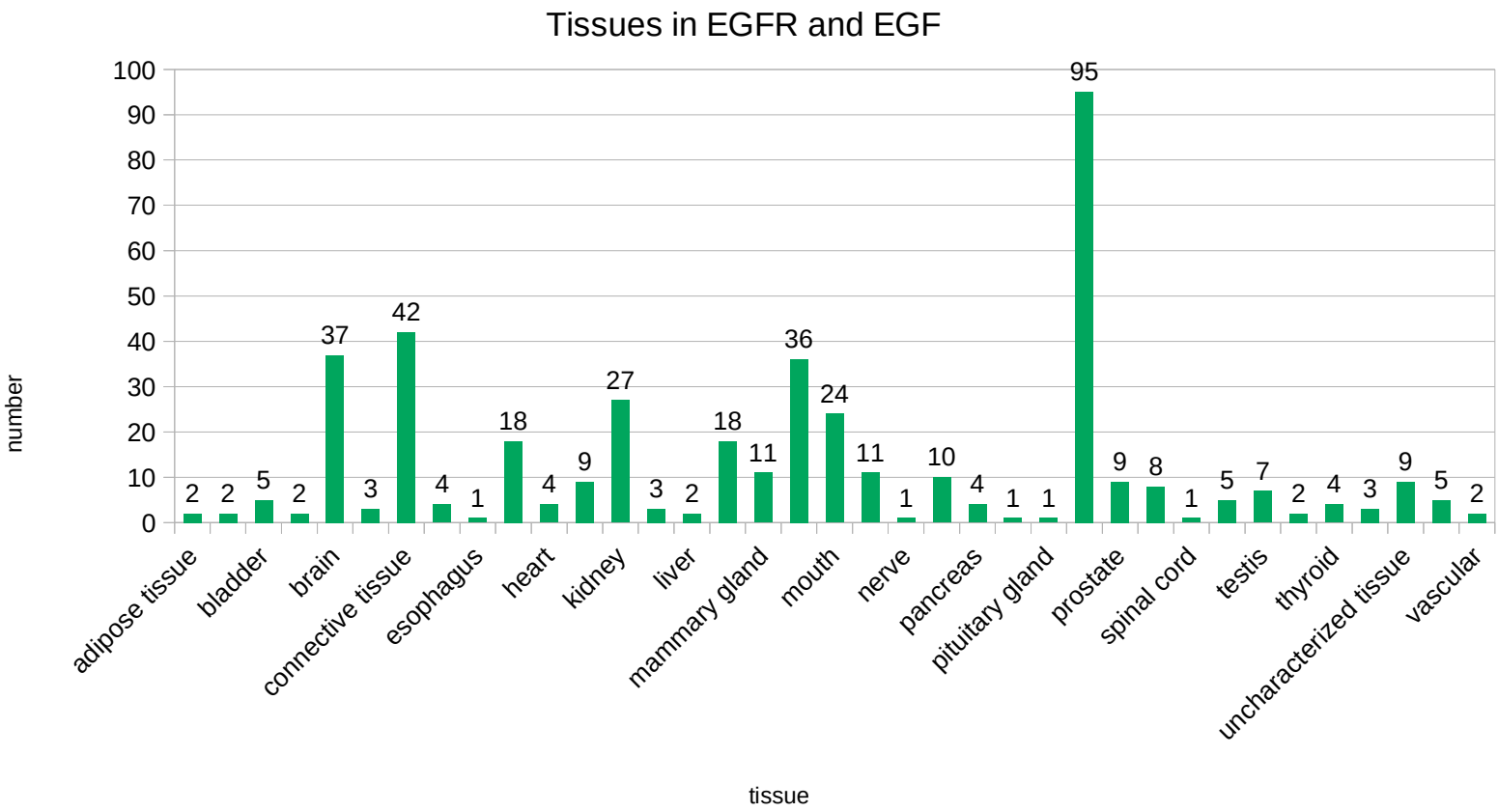
URLs:

- https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?ORG=Hs&CID=488293&MAXEST=377
- https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?ORG=Hs&CID=419815&MAXEST=377

Query lines for tissue.py:

[org]hs [cid]488293

[org]hs [cid]419815



Durch Hinzugabe des Epidermal Growth Factors (EGF) wurde das Bindegewebe “connective tissue” in ihrer Anzahl deutlich erhöht (von 3 auf 42). Auf wenn “placenta” nach wie vor noch am Häufigsten vorkommt, so wurde die Anzahl dieses Gewebes durch das EGF Gen kaum erhöht.

So sieht man den Fokus und die Unterschiede beider Gene (EGFR und EGF) ziemlich deutlich.

Leider wurde beim Einfügen die Gewebsbeschreibungen etwas weggeschnitten.

Die Aussagekraft dieses Diagramm kann durch das Hinzufügen weiterer Gene noch erhöht werden. So könnten bei n>=1000 Gene des Homo Sapiens dies sicher schon etwas mehr über die Gewebsverteilung der Gene beim Menschen aussagen.

Interessant wären nun Vergleiche zwischen verschiedenen Organismen.

2.6.1 Auswertung eines Genes von mehreren Organismen

In dieser Auswertung wird das Gen "Epidermal growth factor receptor" (EGFR) beim Homo sapiens (Mensch), Sus scrofa (Wildschwein) und Mus musculus (Maus) verglichen.

Name:

- Epidermal growth factor receptor (EGFR) (Mensch)
- Epidermal growth factor receptor (EGFR) (Wildschwein)
- Epidermal growth factor receptor (EGFR) (Maus)

URLs:

- https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?ORG=Hs&CID=488293&MAXEST=377
- https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?ORG=Ssc&CID=54725&MAXEST=104
- https://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?ORG=Mn&CID=8534&MAXEST=124

Query lines for tissue.py:

> egfr homo sapiens

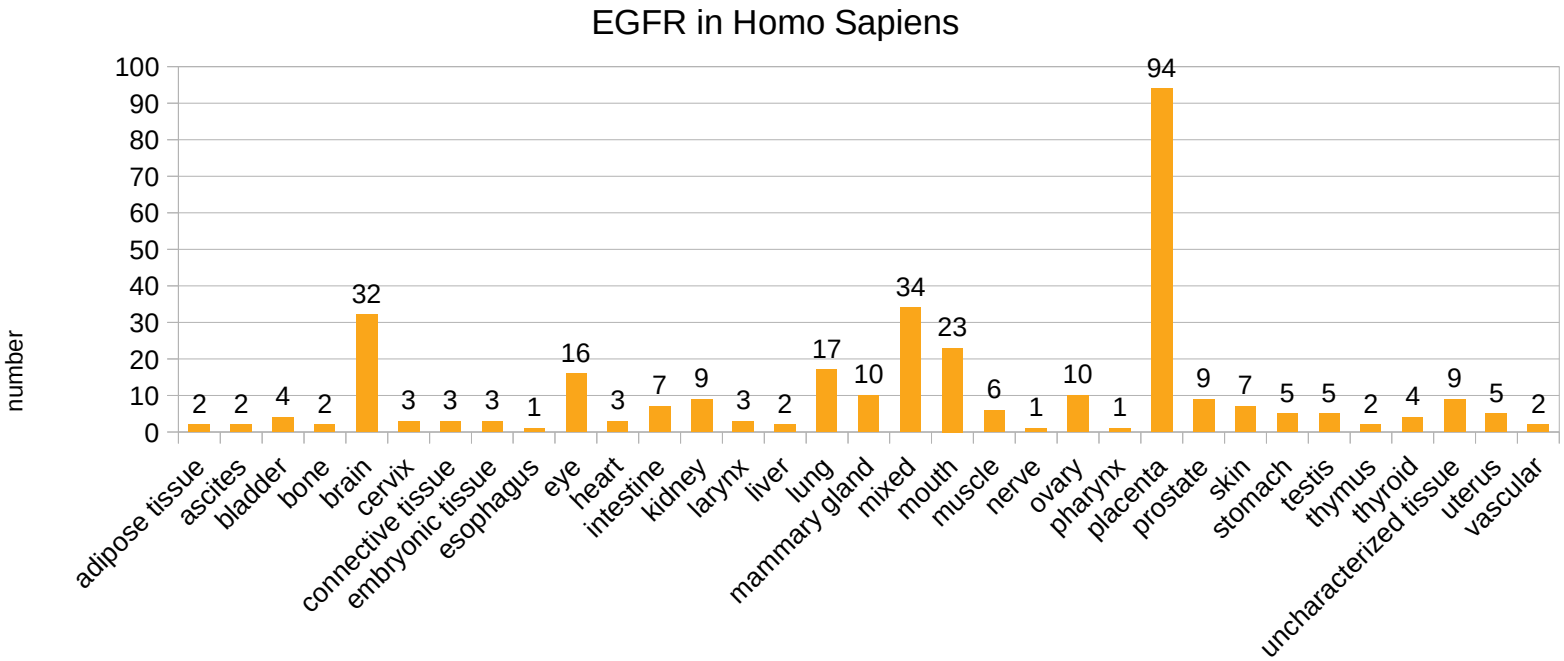
[org]hs [cid]488293

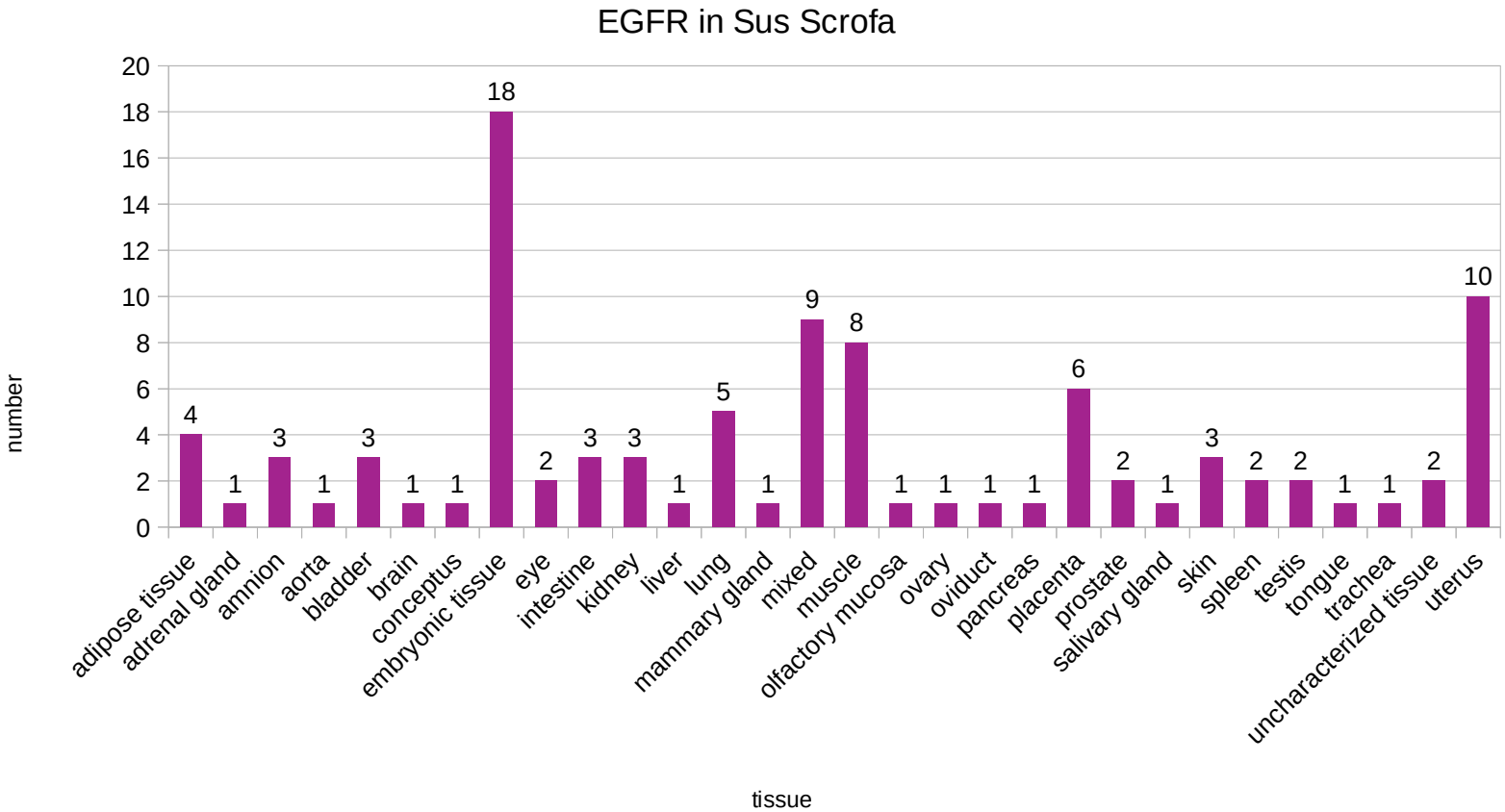
> egfr sus scrofa

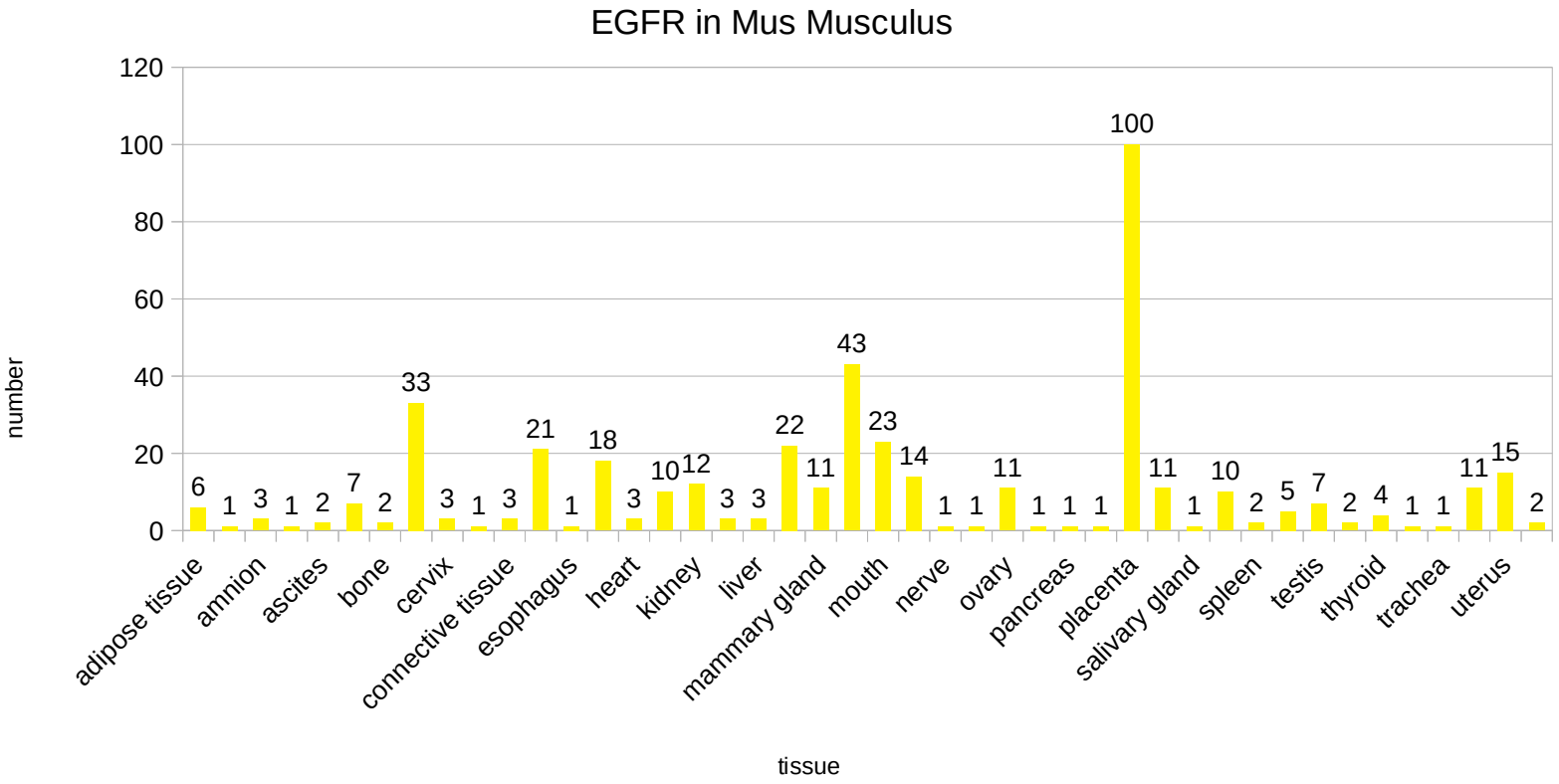
[org]ssc [cid]54725

> egfr mus musculus

[org]mn [cid]8534


EGFR in Homo Sapiens
number
adipose tissue 2
ascites 2
bladder 4
bone 2
brain 32
cervix 3
connective tissue 3
embryonic tissue 3
esophagus 1
eye 16
heart 3
intestine 7
kidney 9
larynx 3
liver 2
lung 17
mammary gland 10
mixed 34
mouth 23
muscle 6
nerve 1
ovary 10
pharynx 1
placenta 94
prostate 9
skin 7
stomach 5
testis 5
thymus 2
thyroid 4
uncharacterized tissue 9
uterus 5
vascular 2


EGFR in Sus Scrofa
number
tissue
adipose tissue 4
adrenal gland 1
amnion 3
aorta 1
bladder 3
brain 1
conceptus 1
embryonic tissue 18
eye 2
intestine 3
kidney 3
liver 1
lung 5
mammary gland 1
mixed 9
muscle 8
olfactory mucosa 1
ovary 1
oviduct 1
pancreas 1
placenta 6
prostate 2
salivary gland 1
skin 3
spleen 2
testis 2
tongue 1
trachea 1
uncharacterized tissue 2
uterus 10



Auch wenn der Mensch für dieses Gen am Meisten Gewebe aufweist (336) im Vergleich zum Wildschwein (98) und der Maus (124), so ist es erstaunlich, dass bei der Maus "placenta" dennoch von der Anzahl her über der des Menschen liegt. (100, 94).

Man erkennt sofort, dass das Wildschwein vom Organismus her komplett verschieden ist, wenn man sich die Gewebstypen und deren Auftretungsanzahl an.

Sehr spannend ist jedoch auch die Ähnlichkeit der Maus zum Menschen. Das Balkendiagramm und die Gewebstypen haben bis auf einige Ausnahmen, ziemlich große Gemeinsamkeiten. Besondern bei der Plazenta, beim Hirn, beim Mund und bei den Augen.

3. Mutationen in Proteinen

3.1 Aufgabe

Ziel dieser Aufgabe ist das Erstellen einer Statistik, welches zu einem oder mehreren Proteinen die verschiedenen Mutationen (Natural Variants) sowie deren Anzahl anzeigt. Das Skript soll u.a Fragen beantworten wie: „Wie oft mutiert Arginin(R) zu Histidin(H)" im Protein TP53 (Cellular tumor antigen p53). Die abzufragende Datenbank hierfür ist Unipro. Dort findet man innerhalb eines Uniprot Eintrag alle Natural Variants vorliegen.

3.2 Umsetzung

Das Projekt zur dieser Aufgabe befindet sich im Ordner „02_mutations".

Zur Abfragung und Speicherung der Mutationen wurde das Python Skript "mutations.py" erstellt, welches die Uniprot Datenbank abfragt.

An das Skript übergeben wird eine Textdatei, welche eine Liste an Uniprot IDs (z.b.: P04637) beinhaltet. Ausgegeben werden die Arten und Anzahl der Mutationen zu den Proteinen (bei mehreren Proteinen werden die Mutationen akkumuliert). Zu leichteren Identifizierung das gesamte Resultat eines CSV Datei erstellt, welches Meta Daten zu den ganzen Proteinen enthält. Für weitere Informationen, was eine Meta Datei darstellt verweise ich auf Abschnitt 2.5.1 der vorherigen Aufgabe.

Das Skript ist auch zusätzlich in der Lage die gefundenen Mutationen innerhalb einer kachelartigen Heatmap (ähnlich der Contributions – Table auf Github) darzustellen.

Dieses Diagramm (Heatmap) wird vom Skript auch noch serialisiert und abgespeichert, damit es später mit Python wieder angezeigt werden kann.

Damit keine Missverständnisse auftreten, das erstellte Diagramm wird sowohl als Bild, als auch als binäres File abgesichert. Für weitere Informationen sehen die Abschnitt Ausgabe weiter unten.

3.3 Beispielaufruf

3.3.1 Beispielaufruf ohne Graph

Ein verkürzter Abschnitt eines Beispielaufrufs des Skripts ohne Erstellung eines Diagramms:

```
radio:02_mutations alen$ python3.6 mutation.py uniprot_ids
[i] starting uniprot crawler
[i] reading uniprot ids from uniprot_ids
[i] crawling http://www.uniprot.org/uniprot/P04637
[i] crawling http://www.uniprot.org/uniprot/P68871
[i] crawling http://www.uniprot.org/uniprot/Q14524

Mutations in human
==================
[Key]: A
       > D (26)
       > E (5)
       > F (2)
       > G (17)
       > I (3)
       > P (18)
       > S (19)
       > T (32)
       > V (41)

[Key]: C
       > A (1)
       > F (10)
       > G (12)


[Key]: Y
       > A (1)
       > C (21)
       > D (8)
       > F (7)
       > G (1)
       > H (15)
       > K (1)
       > N (10)
       > Q (1)
       > S (12)
       > Y (3)

[i] meta data saved to output/human/meta.csv
[i] csv saved to output/human/variants.csv
[i] stopping uniprot crawler
```

Hier sieht man wie der Uniprot – Crawler die Protein IDs aus dem Input File "uniprot_ids" liest und anschließend das Crawling beginnt.

Nachdem der Content der Uniprot Webseite angekommen ist, verarbeitet der Crawler diese und sucht sich die relevanten Informationen; in diesem Fall die Mutationen.

Diese gibt er dann auf die Konsole heraus und speichert sie anschließend innerhalb einer CSV Datei – in diesem Fall tissues.csv – ab.

Die CSV Datei „meta.csv“ enthält zu jeden abgefragten Protein den dazugehörenden Namen.

3.3.1 Beispielaufruf mit Graph

Ein verkürzter Abschnitt eines Beispielaufrufs des Skripts mit Erstellung eines Diagramms:

```
[Key]: Y
       > A (1)
       > C (21)
       > D (8)
       > F (7)
       > G (1)
       > H (15)
       > K (1)
       > N (10)
       > Q (1)
       > S (12)
       > Y (3)

[i] meta data saved to output/human/meta.csv
[i] csv saved to output/human/variants.csv
[i] figure picture saved to output/human/variants_mesh.png
[i] figure binary saved to output/human/variants_mesh.matpltlib
[i] stopping uniprot crawler
```

Wie hier zu sehen ist, wird neben den beiden CSV Dateien noch ein Bild „variants_mesh.png“ sowie eine Binärdatei „variants_mesh.matpltlib“ erstellt. Beide Dateien werden im Abschnitt Ausgabe erklärt.

3.4 Parametrisierung

3.4.1 Parametrisierung des Input Files

Das Input File, welches von mutations.py eingelesen wird, sieht folgendermaßen aus:

```
> human
P04637
P68871
Q14524
```

Als Erstes zu beachten ist die erste Zeile. Ich habe mir erlaubt zur Benennung der Resultate ein ähnliches Format zu wählen, wie es in Fasta Files üblich ist. Diese erste Zeile markiert die Header Line, welche den Namen für das Resultat vergibt. Der Name des Resultates sollte keine Sonderzeichen enthalten, darf aber ruhig aus mehreren Wörter / Leerzeichen bestehen, die zu Unterstrichen aufgelöst werden. Eine komplette Leerzeile wird ignoriert.

Das Skript erwartet gültige Uniprot IDs, mit welchen er nach den Mutationen sucht.

Leere Zeilen werden ignoriert, Sonderzeichen sollten im ganzen File nicht vorkommen.

3.4.2 Parametrisierung des Programms

Wie zuvor erwähnt ist es auch möglich das Programm zu parametrisieren. Übliche Anwendungsweisen sind u.a der Name des Input Files, der Name des Überordners für Resultate, etc.

Für das Skript gibts es zwei Art und Weisen, wie man die Parameter festlegen kann.

3.4.3 Parametrisierung im Quellcode

Das Programm kann in der Main Funktion des Skript manuell mit Parameters ausgestattet werden.

```
def main():
    # define application, default program parameter
    app = {
        "input_file": "uniprot_ids",
        "output_dir": "output/",
        "graph": False,
        "timeout": 15,
    }
```

Hier wird ein Dictionary, also ein Key – Value Verzeichnis festgelegt, welches für das gesamte Skript globale Parameter enthält.

- input_file: der Defaultname für das Input File ist „example“, dies kann hier überschrieben werden
- output_dir: der Name des Überordners aller Resultates
- graph: wird Graph auf True gesetzt, wird das Programm für jedes Resultat ein Diagramm mit der Python Bibliothek matplotlib erstellen. Das Diagramm stellt eine kachelartige Heatmap dar. Gleichzeitig serialisiert das Skript den Graphen in einer Binär Datei

  Vorsicht: Zur Erstellung eines Graphen müssen die Python Bibliotheken installiert sein, welches sich in der Datei requirements.txt befinden. Für weitere Informationen, gehen Sie zum Abschnitt 3.5.3 variants_mesh.png oder 3.5.4 variants_mesh.matpltlib
- timeout: der Timeout ist die Anzahl der Sekunden, die das Skript maximal wartet für jeden HTTP Request, den es an Unigene schickt. Das NCBI ja bekannt ist, dass es manchmal recht langsam ist, ist es wichtig eine passende Timoutdauer festzusetzen. Wird nach Ablauf des Timeouts immer noch keine Antwort von NCBI erwartet,

überspringt das Programm den jeweiligen Unigene Link bzw. Parameter.

3.4.4 Parametrisierung als Kommandozeilenparameter

Entschließt man sich jedoch die Paramatrisierung des Programms aus Sicherheitsgründen vorzunehmen, ermöglicht das Skript ebenso Kommandozeilenparameter an das Skript zu übergeben.

Ein Beispielaufruf:

```
[radio:02_mutations alen$ python3.6 mutation.py uniprot_ids resultat -graph 50
```

Hier wird an das Skript die folgenden Parameter übergeben:

- input_file: unitprot_ids
- output_dir: resultat
- graph: True (durch das Übergeben von -graph wird das Flag aktiviert; ansonsten wird kein Diagramm erstellt.)
- timeout: 50

Beachten Sie, dass die Reihenfolge der Parameter fest vorgegeben ist. Wird eine andere Reihenfolge der Parameter gesetzt, kann das Skript zum Absturz kommen.

3.5 Ausgabe

```
[Key]: Y
       > A (1)
       > C (21)
       > D (8)
       > F (7)
       > G (1)
       > H (15)
       > K (1)
       > N (10)
       > Q (1)
       > S (12)
       > Y (3)

[i] meta data saved to output/human/meta.csv
[i] csv saved to output/human/variants.csv
[i] stopping uniprot crawler
```

Das Skript ist in der Lage die Mutationen sowie deren Anzahl aller übergebenen Proteine im Input File auf die Konsole auszugeben. Gleichzeitig speichert das Skript die Daten im CSV File "variants.csv" ab und erstellt eine "meta.csv" Datei. Diese beiden Dateien können leicht via Microsoft Office Excel oder Libre Office Calc importiert werden.

Ist das Graph Flag gesetzt (auf True gesetzt) so erstellt das Programm zwei weitere Dateien. Zum einen das Bild der Heatmap "variants_mesh.png" und zum anderen die binäre Serialisierung des Diagramms "variants_mesh.matpltlib".

3.5.1 variants.csv

Die Datei variants.csv wird pro Resultat für alle im Input File übergeben Proteine akkumuliert erstellt.

```
> human
P04637
P68871
Q14524
```

In diesem Beispiel wird Uniprot für die Proteine P04637 (Cellular tumor antigen p53), P68871 (Hemoglobin subunit beta) und Q14524 (Sodium channel protein type 5 subunit beta) abgefragt.

Die Mutationen der Proteine werden im Resultatsordner "human" abgespeichert.

Hier ein verkürzter Abschnitt des erstellten CSV Files.

```
Mutations in human:
from AA;to AA;number of variants
A;D;26
A;E;5
A;F;2
A;G;17
A;I;3
A;P;18
A;S;19
A;T;32
A;V;41
C;A;1
C;F;10
C;G;12
C;H;1
```

3.5.2 meta.csv

Die meta.csv Datei beinhaltet Daten die nicht per-se für Statistiken verwendet werden können, aber dennoch Aufschluss über das Resultat geben.
Damit ist gemeint, dass in der meta.csv zu jeden Gen kontextaussagenden Daten gespeichert sind wie:

- Die Uniprot IDs
- Der Protein Name

Zu dem vorherigen Aufruf das dazugehörende abgeschnittene meta.csv:

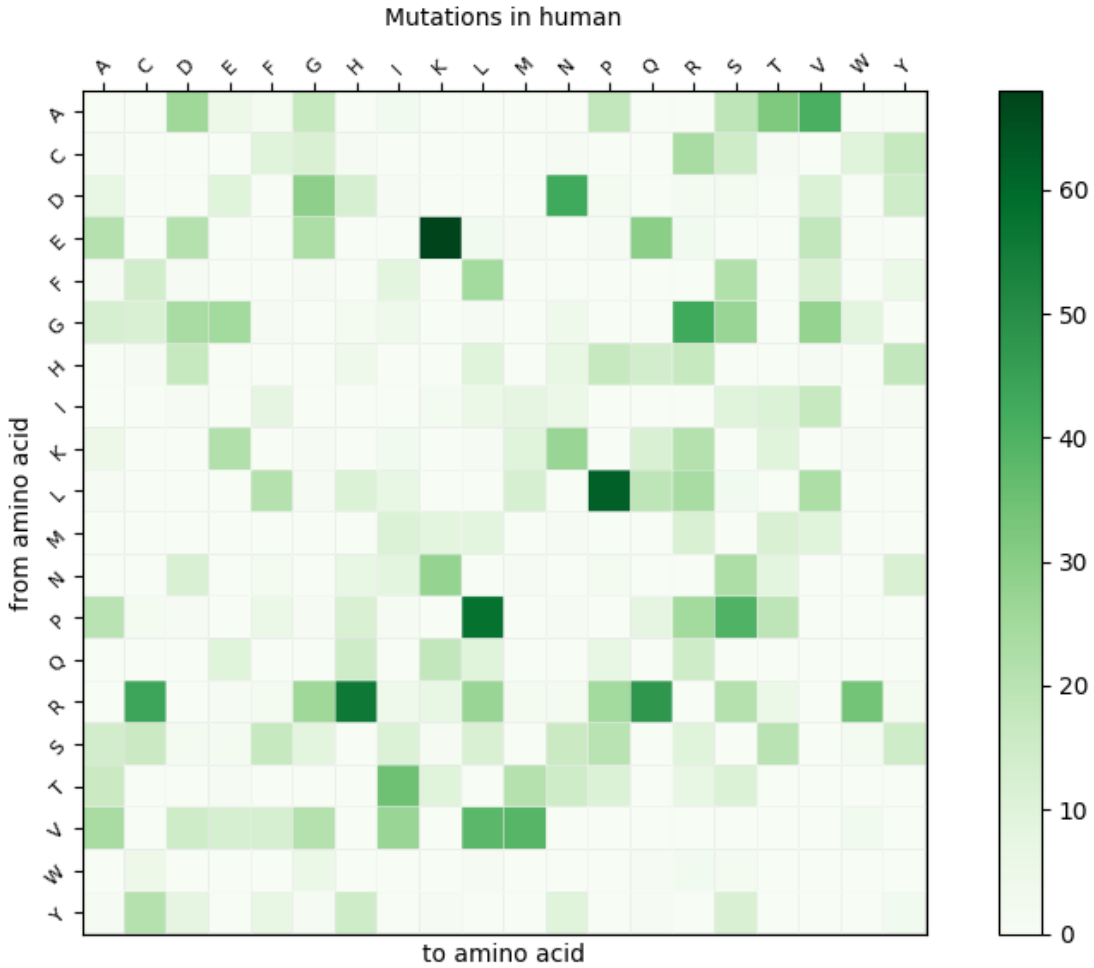
```
Crawled proteins in human:
id;name
P04637;Cellular tumor antigen p53
P68871;Hemoglobin subunit beta
Q14524;Sodium channel protein type 5 subunit alpha
```

Auf wenn diese Daten keine Aussage zu den Mutationen bzw. die Anzahl der jeweiligen Mutationen macht, so denke ich mir würde es den Nutzer dennoch interessiert, was genau abgefragt worden ist.

Bei z.b einem Aufruf von 100 Proteinen des Homo Sapiens hat es nicht unbedingt einen Sinn, die Daten mit dem Output der Konsole zu vergleichen.

3.5.2 variants_mesh.png

Dieses Bild zeigt die gefundenen Mutationen aller übergeben Proteine innerhalb einer mit matplotlib generierten Heatmap. Auf unseres Beispiel bezogen, zu den übergebenen drei.



Meiner Meinung nach ist das eine bessere Art und Weise im Vergleich zu einem Balkendiagramm, wie man die Stärke eines Wertes ausdrücken. So sieht man beim ersten Blick schon dass innerhalb der drei Proteine Glutaminsäure(E) ziemlich häufig zu Lysin(K) wechselt, gefolgt von Leucin (L) zu Prolin(P).

Zu beachte ist, dass um dieses Bild generieren zu können, zuvor die im File „requirements.txt“ angegeben Python Bibliotheken installiert werden müssen. Für mehr Informationen zur Installation, sehen sie im Abschnitt 1.3.1 Installation der Python Drittbibliotheken nach.

3.5.2 variants_mesh.matpltlib

Der Nachteil bei dem zuvor erwähnten Bild ist, dass es keinerlei Interaktion zum Benutzer ermöglicht, dass es letztendlich nur ein statisches Bild ist. Das dahinterliegende Figure des Bildes ist jedoch in der Lage dem User weitere Informationen zu geben.

Als Beispiel, zeigt das Bild zu einer Mutationen den Stärkegrad in Form der Farbe an, aber keine genaue Nummer.

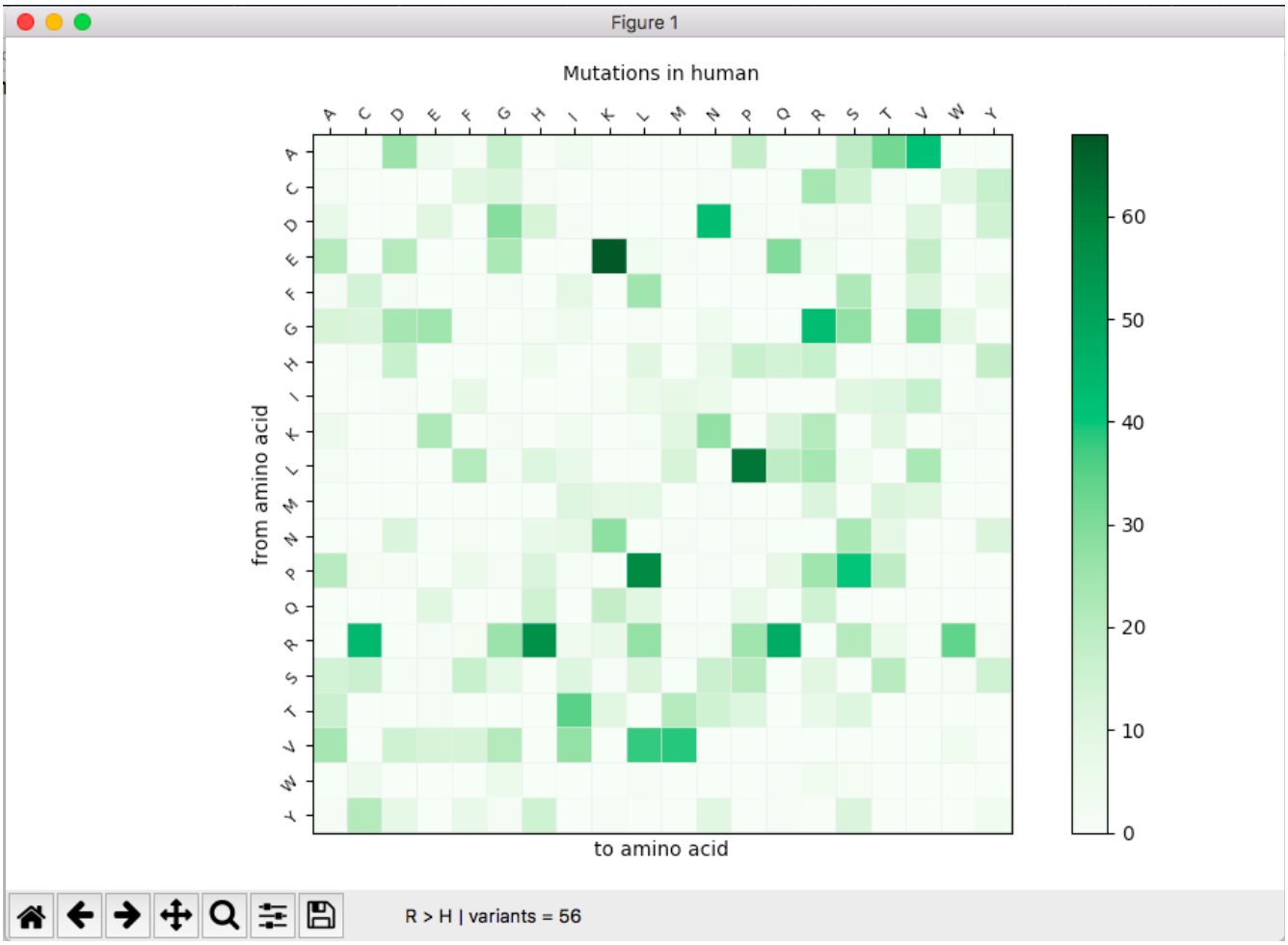
Hat man jedoch das Figure offen, so sieht man auch die Anzahl der Mutationen sobald man mit der Maus über das Feld fährt.

Um nun dem Benutzer die Möglichkeit zu geben, das Figure erneut aufzurufen und sich im Detail anzusehen, habe ich mich entschlossen, das Matplotlib Figure mit der Python Bibliothek Dill binär zu serialisieren. Das heißt, die gesamte Datenstruktur zu erhalten und einfach in einem File zu verewigen.

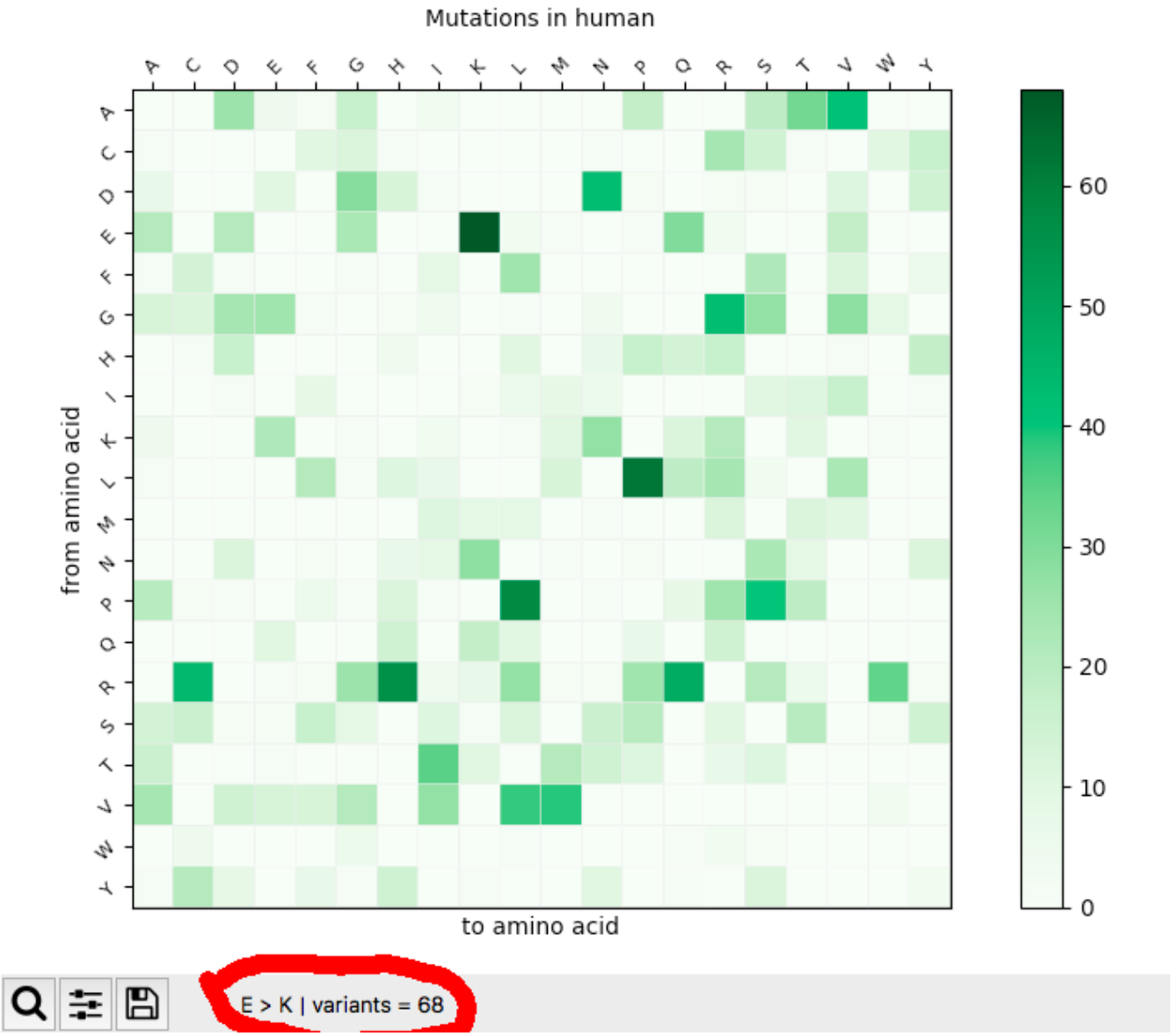
Möchte der Benutzer nun das Figure erneut aufscheinen lassen, habe ich mir erlaubt im Rahmen dieser zweiten Aufgabe ein weiteres Skript zu verfassen, welches genau diese Aufgabe erfüllt. Das Skript heißt „showgraph.py“ und liegt im Ordner der zweiten Aufgabe im „utils/“ Ordner. Um dieses Skript ebenso nutzen zu können, müssen zuvor die benötigten Drittbibliotheken installiert sein.

Ein Aufruf mit dem “showgraph.py” Skript zu unserem vorherigen Beispiel mit den drei Proteinen zeigt die Heatmap in ihrer ganzen Pracht.

```
[radio:02_mutations alen$ python3.6 utils/showgraph.py output/human/variants_mesh.matpltlib
```



Beachten Sie in der Statusleiste im Figure die durch den Kontext angezeigten Infomationen.



Hier kann man nun deutlich erkennen, dass es 68 Mutationen zwischen Glutaminsäure und Lysin gibt. Auch wenn es auf dem Bild nicht sichtbar ist, meine Maus ist in diesem Moment darüber gefahren.

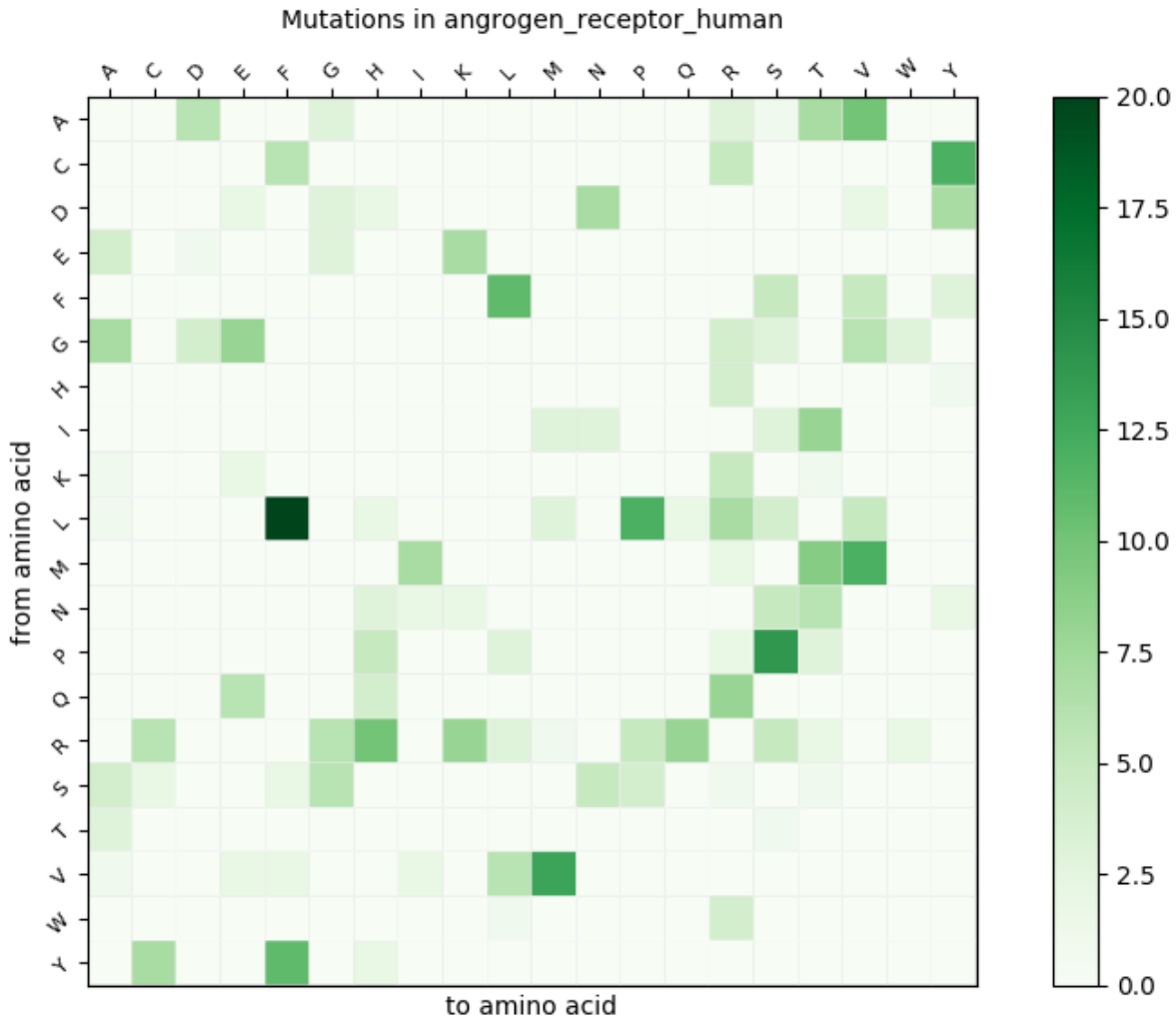
3.6 Auswertung / Statistik

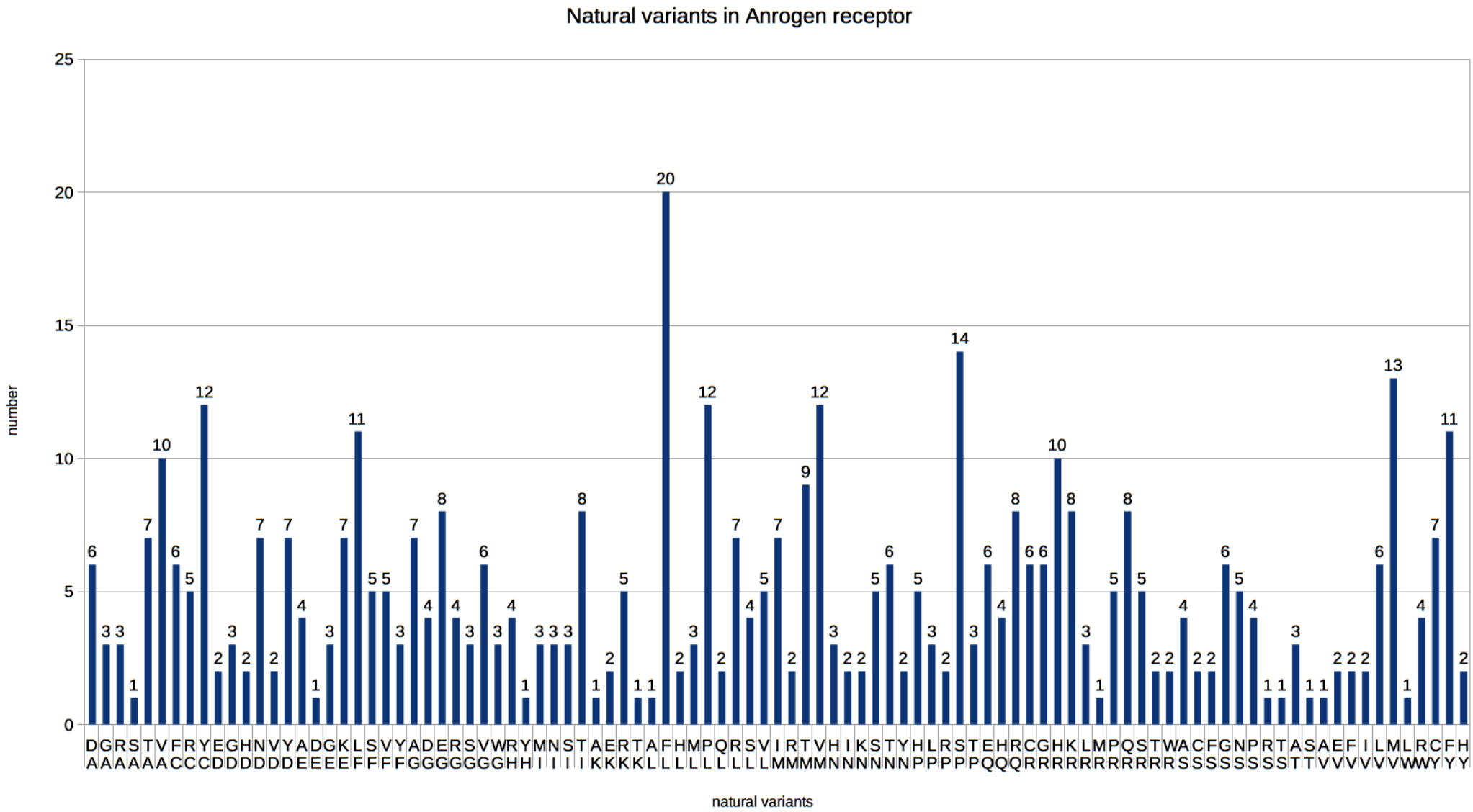
3.6.1 Auswertung mit einem Protein

Name: Androgen receptor

Uniprot ID: P10275

Das Diagramm befindet sich auf der nächsten Seite. Zuerst die Heatmap.




Natural variants in Anrogen receptor
number
natural variants
0
5
10
15
20
25
6 3 3 1 7 10 6 5 12 2 3 2 7 2 7 4 1 3 7 11 5 5 3 7 4 8 4 3 6 3 4 1 3 3 3 8 1 2 5 1 1 20 2 3 12 2 7 4 5 7 2 9 12 3 2 2 5 6 2 5 3 2 14 3 6 4 8 6 6 10 8 3 1 5 8 5 2 2 4 2 2 6 5 4 1 1 3 1 1 2 2 2 6 13 1 4 7 11 2
DAGRSTVFRYEGHNVYADGKLSVYADERSVWRYMNSTAERTAFHMPQRSVIRTVHIKSTYHLRSTEHRCGHKLMPQSTWACFGNPRTASAEFILMLRCFH
AAAAACCCDDDDDDDEEEEFFFFGGGGGGGHHIIIIIKKKLLLLLLLLLMMMMNNNNNNNPPPPQQQRRRRRRRRRRSSSSSSSSSTTVVVVVVWWYYY

Da das Diagramm ansonsten einen Großteil der Daten wegschnitten hätte, musste ich es im Querformat darstellen.

In der untersten Zeile auf der X – Achse sieht man die Von – Aminosäuren, eins darüber sind die Zu – Aminosäuren.

Sowohl die Heatmap als auch das Diagramm zeigen beide eine relative häufige Umwandlung in diesem Protein von Leucin (L) zu Phenylalanin (F), 20 Mal.

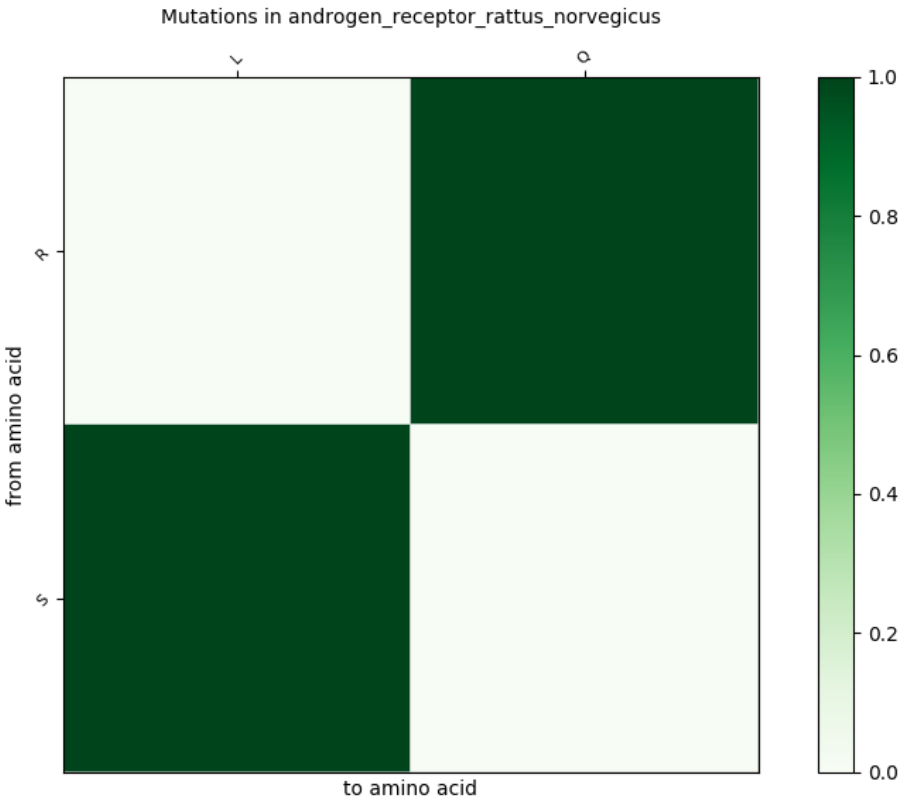
Weiters stark umgewandelt wurde Prolin (P) zu Serin (S), ganze 14 Mal und Valin (V) zu Leucin (L), 13 Mal.

3.6.1 Auswertung eines Proteins mit mehreren Organismen

In dieser Auswertung wird das Protein "Androgen Receptor" für die beiden Organismen Homo Sapiens (Mensch) und Rattus norvegicus (Ratte) vergleichen. Da die Auswertung für Menschen bereits im letzten Abschnitt stattgefunden hat, wird nurmer die Auswertung von Maus ergänzt und diese beiden zusammen ausgewertet.

Name: Androgen receptor

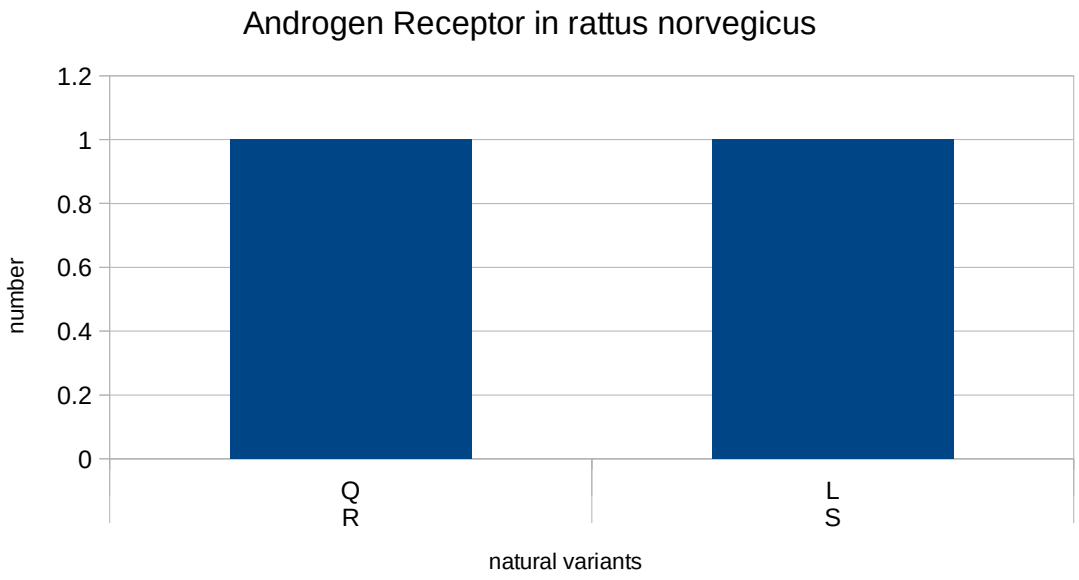
Uniprot ID: P15207



Auch wenn diese Heatmap etwas merkwürdig wirken mag, so sieht man dennoch, dass für die Ratte im Vergleich zum Menschen das Protein kaum bis gar keine Mutationen durchführt.

Lediglich eine von Serin (S) zu Leucin (L) und von Arginin (R) zu Glutamin (Q).

Man sollte mehrere Organismen bezüglich dieses Proteins vergleichen, um mehr Aufschluss zwischen den verschiedenen Organismen zu bekommen.


Androgen Receptor in rattus norvegicus
1.2
1
0.8
0.6
0.4
0.2
0
number
Q
R
L
S
natural variants