While collaborating on a research project with colleagues from the University of Antwerp, we became interested in understanding how frequently specific amino-acid motifs appear in the human proteome. This question arose because we were studying a neuropathy-causing mutation in the protein HSP27, which is a small heat-shock protein that functions as a molecular chaperone. The particular mutation in HSP27 changes the native proline residue at position 182 to leucine (P182L), which disrupts a conserved motif known as the "[I/V]x[I/V]" motif wherein the first and third residues are either isoleucine (I) or valine (V) while the central residue can be anything (x). In mammalian small heat-shock proteins, the central residue is typically proline.

The [I/V]x[I/V] motif is an example of a short linear motif (SLiM), which are motifs of typically 3-8 residues that can play important biological roles, such as mediating protein-protein interactions or promoting degradation. Notably, SLiMs generally appear in intrinsically disordered regions (IDRs) of proteins, thus ensuring accessibility and promoting promiscuous interactions with many other proteins. If you are interested in searching for SLiMs in your protein of interest, here is a useful website called SlimSearch, which is based on this paper from the Davey Lab.

Anyway, during our hunt for [I/V]x[I/V] SLiMs in the human proteome, I wrote some Python scripts that I used to perform proteome-wide calculations on UniProt proteomes or FASTA files. I've written a short Python tutorial that outlines the following steps, with example screenshots and code showing how to:

1. **Download and format UniProt proteome files**
2. **Compute the mean and median protein length**
3. **Plot histograms of protein length distributions**

I wrote this in Python3.7.4 and the code requires a few packages (Numpy, Pandas, and BioPython) that you might have to install ahead of time.

First, open up a new text file and import these packages

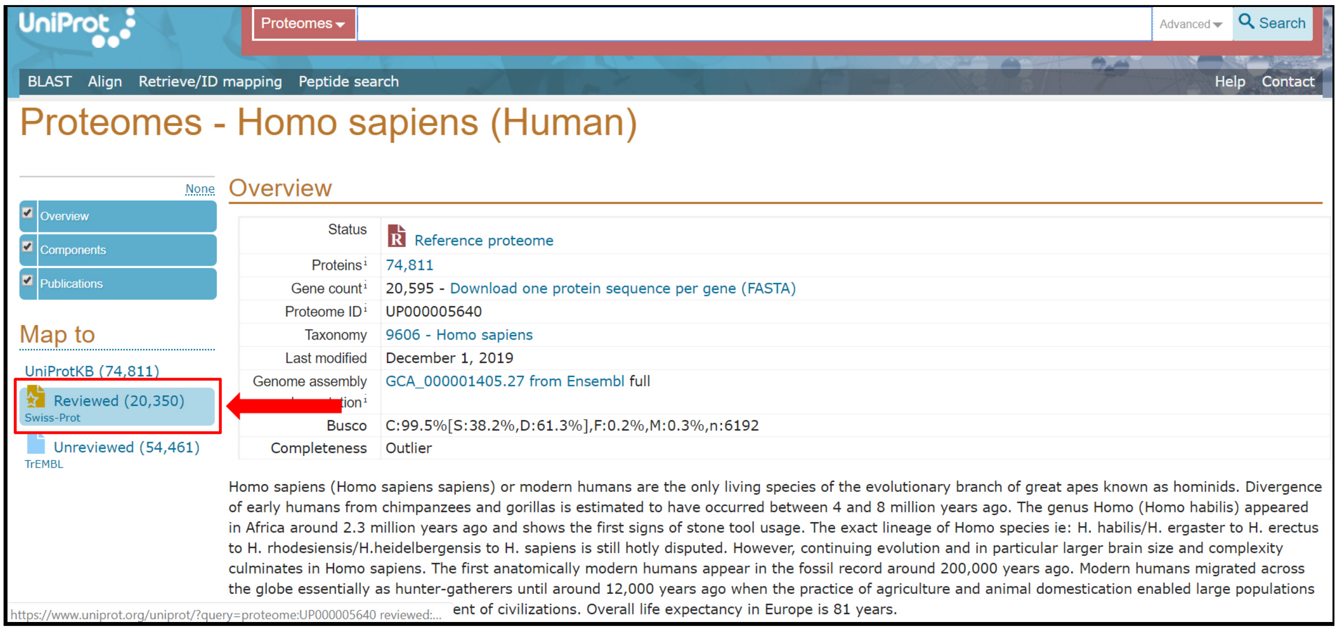
```python
import os, sys
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from Bio import SeqIO
import itertools
```

Now, let's create a class called ***Proteome*** where we will perform our large-scale counting exercises. By default, we must begin the class with the init function. In case you are unfamiliar with classes, take a look at this helpful explanation.

```python
class Proteome(object):
    """ Some useful calculations on proteomes """

    def __init__(self):
        self.type = None
```

Save this file as "run.py" in the directory of your choice.

In order to start counting motifs within a proteome, we first need a proteome! The UniProt website maintains a large repository of proteomes, so download the **Reviewed** human proteome from this link. Make sure to download the files in **FASTA** format! UniProt maintains a very large database of alternatively spliced isoforms of each gene; for the purposes of this example, we are only interested in the "canonical" isoform of each gene, i.e. one gene, one amino acid sequence. So, make sure to click the "Reviewed" version of the proteome.



Proteomes - Homo sapiens (Human)

Overview

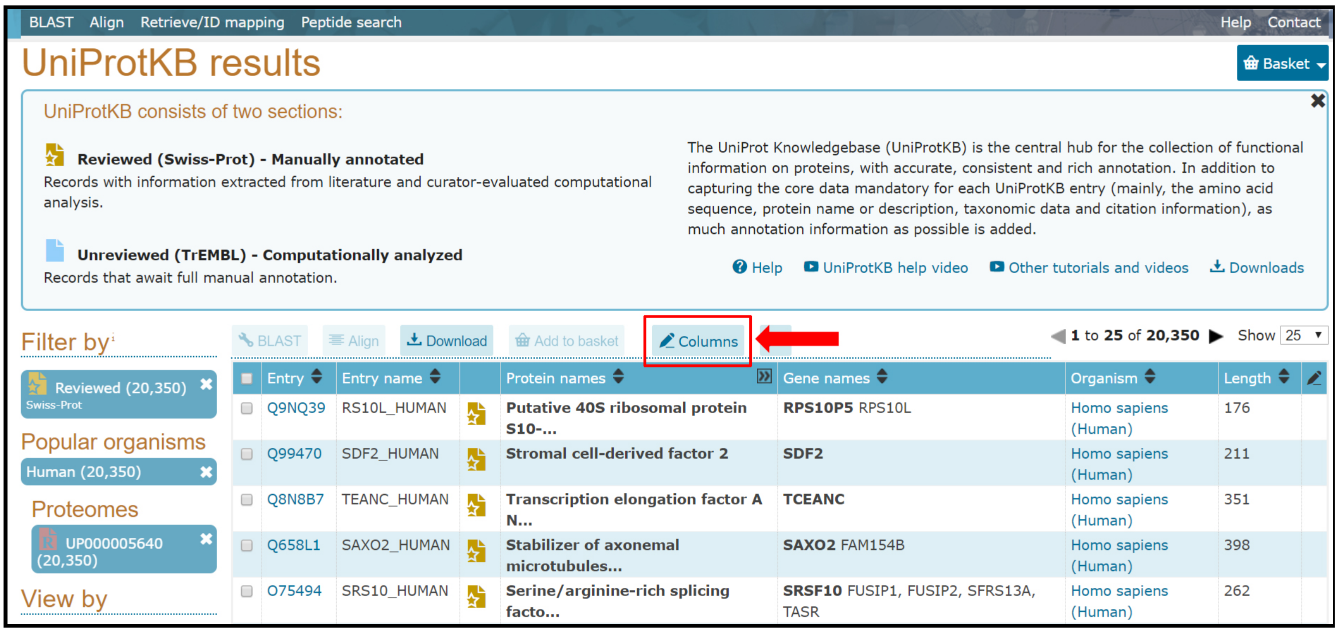
Status	Reference proteome
Proteins	74,811
Gene count	20,595 - Download one protein sequence per gene (FASTA)
Proteome ID	UP000005640
Taxonomy	9606 - Homo sapiens
Last modified	December 1, 2019
Genome assembly ...tion	GCA_000001405.27 from Ensembl full
Busco	C:99.5%[S:38.2%,D:61.3%],F:0.2%,M:0.3%,n:6192
Completeness	Outlier

Homo sapiens (Homo sapiens sapiens) or modern humans are the only living species of the evolutionary branch of great apes known as hominids. Divergence of early humans from chimpanzees and gorillas is estimated to have occurred between 4 and 8 million years ago. The genus Homo (Homo habilis) appeared in Africa around 2.3 million years ago and shows the first signs of stone tool usage. The exact lineage of Homo species ie: H. habilis/H. ergaster to H. erectus to H. rhodesiensis/H.heidelbergensis to H. sapiens is still hotly disputed. However, continuing evolution and in particular larger brain size and complexity culminates in Homo sapiens. The first anatomically modern humans appear in the fossil record around 200,000 years ago. Modern humans migrated across the globe essentially as hunter-gatherers until around 12,000 years ago when the practice of agriculture and animal domestication enabled large populations ...ent of civilizations. Overall life expectancy in Europe is 81 years.

https://www.uniprot.org/uniprot/?query=proteome:UP000005640 reviewed:...

Figure 1: Navigate to the human proteome on UniProt and click ***Reviewed***.

After clicking on **Reviewed**, the website will take you to the page shown below. However, before we download the proteome, we first must specify a few things about the file format. Click on the **Columns** button to specify the data that we'll include in the file.


BLAST Align Retrieve/ID mapping Peptide search Help Contact

UniProtKB results

Basket

UniProtKB consists of two sections:

Reviewed (Swiss-Prot) - Manually annotated
Records with information extracted from literature and curator-evaluated computational analysis.

Unreviewed (TrEMBL) - Computationally analyzed
Records that await full manual annotation.

The UniProt Knowledgebase (UniProtKB) is the central hub for the collection of functional information on proteins, with accurate, consistent and rich annotation. In addition to capturing the core data mandatory for each UniProtKB entry (mainly, the amino acid sequence, protein name or description, taxonomic data and citation information), as much annotation information as possible is added.

Help · UniProtKB help video · Other tutorials and videos · Downloads

Filter by: Reviewed (20,350) Swiss-Prot

Popular organisms: Human (20,350)

Proteomes: UP000005640 (20,350)

View by

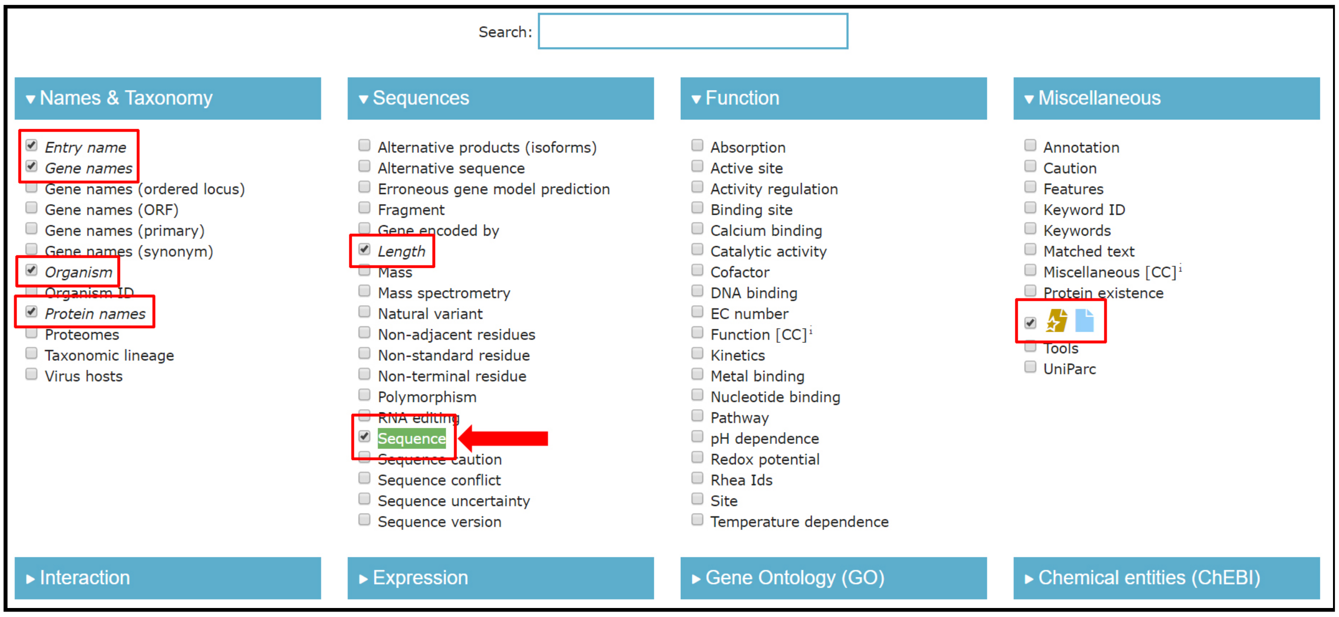
BLAST · Align · Download · Add to basket · Columns

1 to 25 of 20,350 · Show 25

Entry	Entry name		Protein names	Gene names	Organism	Length
Q9NQ39	RS10L_HUMAN		**Putative 40S ribosomal protein S10-...**	**RPS10P5** RPS10L	Homo sapiens (Human)	176
Q99470	SDF2_HUMAN		**Stromal cell-derived factor 2**	**SDF2**	Homo sapiens (Human)	211
Q8N8B7	TEANC_HUMAN		**Transcription elongation factor A N...**	**TCEANC**	Homo sapiens (Human)	351
Q658L1	SAXO2_HUMAN		**Stabilizer of axonemal microtubules...**	**SAXO2** FAM154B	Homo sapiens (Human)	398
O75494	SRS10_HUMAN		**Serine/arginine-rich splicing facto...**	**SRSF10** FUSIP1, FUSIP2, SFRS13A, TASR	Homo sapiens (Human)	262

Figure 2: Specify which parameters to include in the download file by clicking on ***Columns***.

On the **Column** page, you must click the box next to **Sequence** in order to indicate that you want each protein's sequence included in the downloaded file.


Search:

Names & Taxonomy
- [✓] *Entry name*
- [✓] *Gene names*
- Gene names (ordered locus)
- Gene names (ORF)
- Gene names (primary)
- Gene names (synonym)
- [✓] *Organism*
- Organism ID
- [✓] *Protein names*
- Proteomes
- Taxonomic lineage
- Virus hosts

Sequences
- Alternative products (isoforms)
- Alternative sequence
- Erroneous gene model prediction
- Fragment
- Gene encoded by
- [✓] *Length*
- Mass
- Mass spectrometry
- Natural variant
- Non-adjacent residues
- Non-standard residue
- Non-terminal residue
- Polymorphism
- RNA editing
- [✓] Sequence
- Sequence caution
- Sequence conflict
- Sequence uncertainty
- Sequence version

Function
- Absorption
- Active site
- Activity regulation
- Binding site
- Calcium binding
- Catalytic activity
- Cofactor
- DNA binding
- EC number
- Function [CC]
- Kinetics
- Metal binding
- Nucleotide binding
- Pathway
- pH dependence
- Redox potential
- Rhea Ids
- Site
- Temperature dependence

Miscellaneous
- Annotation
- Caution
- Features
- Keyword ID
- Keywords
- Matched text
- Miscellaneous [CC]
- Protein existence
- [✓]
- Tools
- UniParc

Interaction · Expression · Gene Ontology (GO) · Chemical entities (ChEBI)

Figure 3: Click the box next to ***Sequence***

Then scroll up and click **Save** in the upper-right corner.

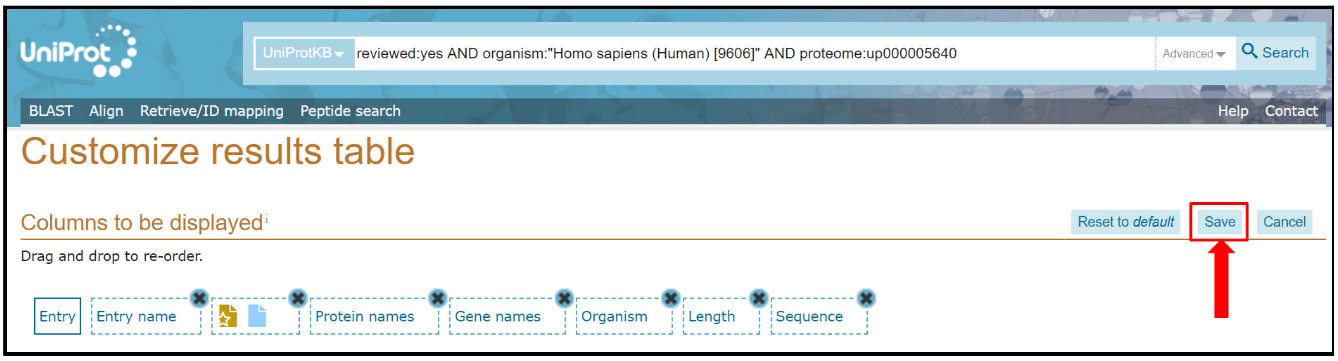
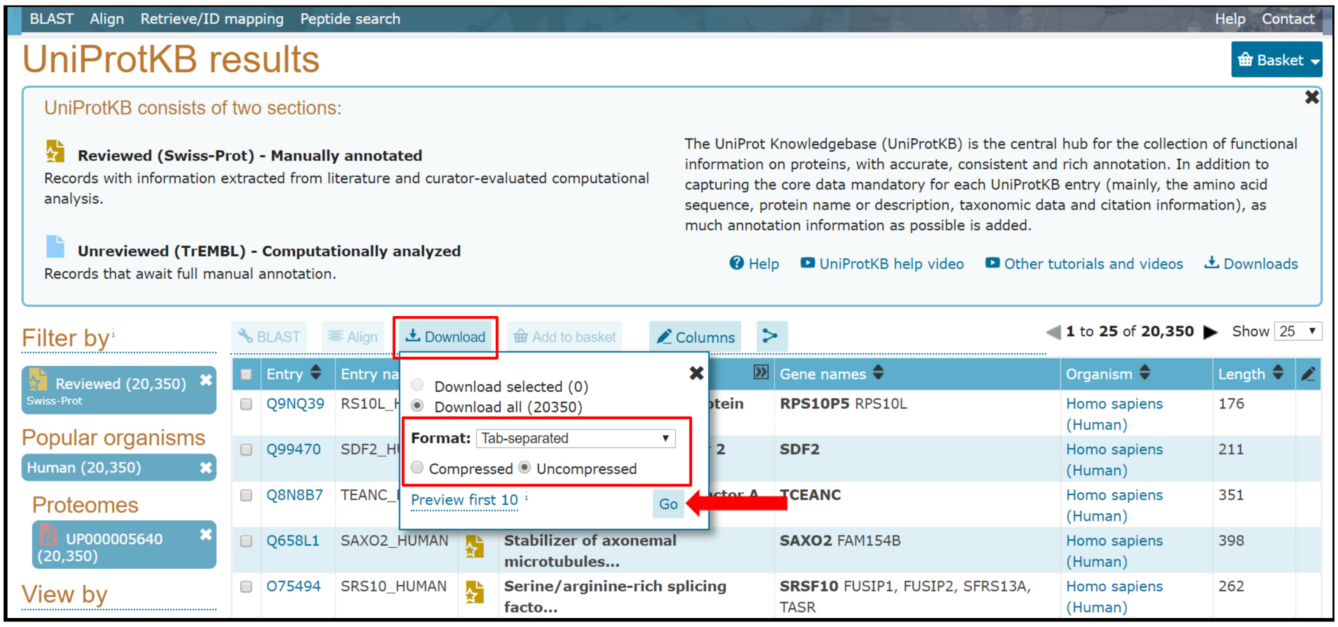


Figure 4: Click ***Save***

This will take you back to the UniProtKB Results screen from above. Click on **Download** and then, under **Format**, choose **Tab-separated** from the drop-down menu. Also, make sure to click the button **Uncompressed**. Now, you can click **Go** to download the file containing 20,350 FASTA files. I saved this file as "uniprot_human.tab" and it's roughly 14.5 MB in size.



UniProtKB results

UniProtKB consists of two sections:

Reviewed (Swiss-Prot) - Manually annotated
Records with information extracted from literature and curator-evaluated computational analysis.

Unreviewed (TrEMBL) - Computationally analyzed
Records that await full manual annotation.

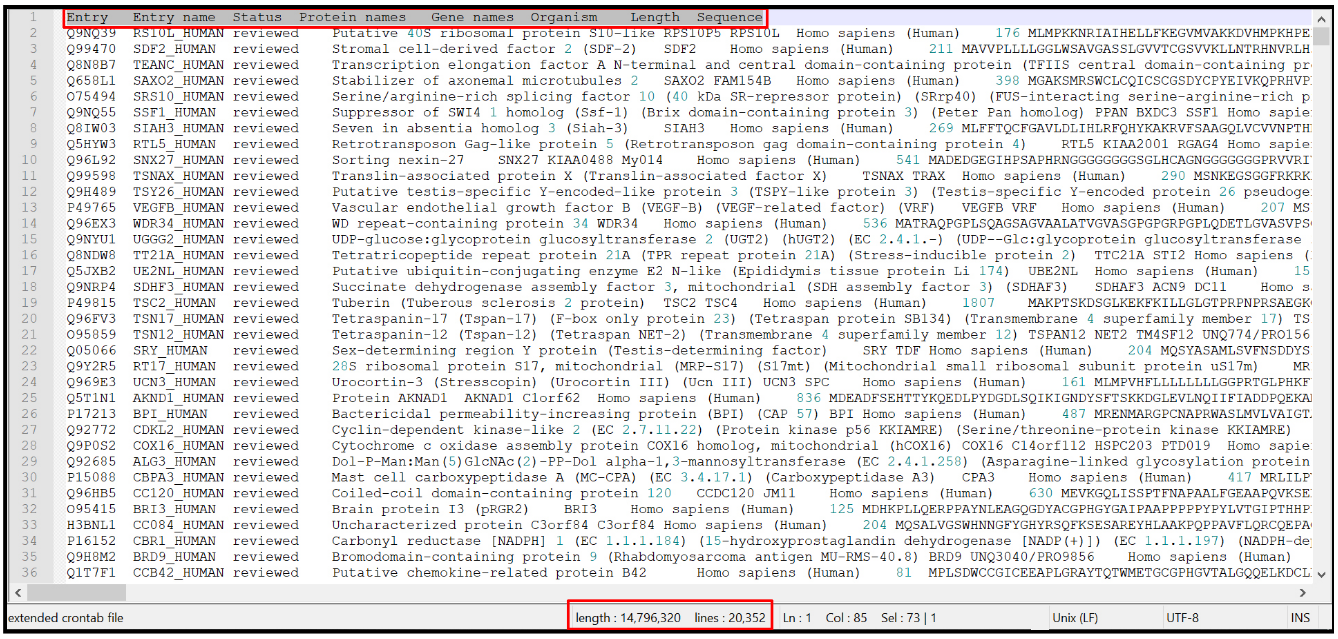
The UniProt Knowledgebase (UniProtKB) is the central hub for the collection of functional information on proteins, with accurate, consistent and rich annotation. In addition to capturing the core data mandatory for each UniProtKB entry (mainly, the amino acid sequence, protein name or description, taxonomic data and citation information), as much annotation information as possible is added.

Format: Tab-separated
Compressed [✓ Uncompressed]

Entry	Entry na			Gene names	Organism	Length
Q9NQ39	RS10L_H		tein	**RPS10P5** RPS10L	Homo sapiens (Human)	176
Q99470	SDF2_H		2	**SDF2**	Homo sapiens (Human)	211
Q8N8B7	TEANC_		ctor A	**TCEANC**	Homo sapiens (Human)	351
Q658L1	SAXO2_HUMAN		**Stabilizer of axonemal microtubules...**	**SAXO2** FAM154B	Homo sapiens (Human)	398
O75494	SRS10_HUMAN		**Serine/arginine-rich splicing facto...**	**SRSF10** FUSIP1, FUSIP2, SFRS13A, TASR	Homo sapiens (Human)	262

Figure 5: Download the tab-separated file

Before we move ahead, let's open up the file and see what it contains. It should come as a [long-name-here].tab file, so rename it something convenient and store it in the same directory as your new .py file. Then, open up the .tab file in your favorite text editor. Below is a screenshot from Notepad++, and you can see that I've highlighted the first row that contains the column headings: ***Entry, Entry name, Status, Protein names, Gene names, Organism, Length, Sequence***. At the bottom of Notepad++, it shows us that we have 20,352 lines in this file. The first line contains the column headers and the very last line is empty, so there are **20,350 proteins** here. This is a good sanity check: the number of proteins in the file matches the value listed on the UniProt website.



```
Entry   Entry name  Status  Protein names   Gene names  Organism    Length  Sequence
Q9NQ39  RS10L_HUMAN reviewed    Putative 40S ribosomal protein S10-like RPS10P5 RPS10L  Homo sapiens (Human)    176 MLMPKKNRIAIHELLFKEGVMVAKKDVHMPKHPE
Q99470  SDF2_HUMAN  reviewed    Stromal cell-derived factor 2 (SDF-2)   SDF2    Homo sapiens (Human)    211 MAVVPLLLLGGLWSAVGASSLGVVTCGSVVKLLNTRHNVRLH
Q8N8B7  TEANC_HUMAN reviewed    Transcription elongation factor A N-terminal and central domain-containing protein (TFIIS central domain-containing pr
Q658L1  SAXO2_HUMAN reviewed    Stabilizer of axonemal microtubules 2   SAXO2 FAM154B   Homo sapiens (Human)    398 MGAKSMRSWCLCQICSCGSDYCPYEIVKQPRHVP
O75494  SRS10_HUMAN reviewed    Serine/arginine-rich splicing factor 10 (40 kDa SR-repressor protein) (SRrp40) (FUS-interacting serine-arginine-rich p
Q9NQ55  SSF1_HUMAN  reviewed    Suppressor of SWI4 1 homolog (Ssf-1) (Brix domain-containing protein 3) (Peter Pan homolog) PPAN BXDC3 SSF1 Homo sapie
Q8IW03  SIAH3_HUMAN reviewed    Seven in absentia homolog 3 (Siah-3)    SIAH3   Homo sapiens (Human)    269 MLFFTQCFGAVLDLIHLRFQHYKAKRVFSAAGQLVCVVNPTH
Q5HYW3  RTL5_HUMAN  reviewed    Retrotransposon Gag-like protein 5 (Retrotransposon gag domain-containing protein 4)    RTL5 KIAA2001 RGAG4 Homo sapie
Q96L92  SNX27_HUMAN reviewed    Sorting nexin-27    SNX27 KIAA0488 My014    Homo sapiens (Human)    541 MADEDGEGIHPSAPHRNGGGGGGGSGLHCAGNGGGGGGPRVVRI
Q99598  TSNAX_HUMAN reviewed    Translin-associated protein X (Translin-associated factor X)    TSNAX TRAX  Homo sapiens (Human)    290 MSNKEGSGGFRKRK
Q9H489  TSY26_HUMAN reviewed    Putative testis-specific Y-encoded-like protein 3 (TSPY-like protein 3) (Testis-specific Y-encoded protein 26 pseudoge
P49765  VEGFB_HUMAN reviewed    Vascular endothelial growth factor B (VEGF-B) (VEGF-related factor) (VRF)   VEGFB VRF   Homo sapiens (Human)    207 MS
Q96EX3  WDR34_HUMAN reviewed    WD repeat-containing protein 34 WDR34   Homo sapiens (Human)    536 MATRAQPGPLSQAGSAGVAALATVGVASGPGPGRPGPLQDETLGVASVPS
Q9NYU1  UGGG2_HUMAN reviewed    UDP-glucose:glycoprotein glucosyltransferase 2 (UGT2) (hUGT2) (EC 2.4.1.-) (UDP--Glc:glycoprotein glucosyltransferase
Q8NDW8  TT21A_HUMAN reviewed    Tetratricopeptide repeat protein 21A (TPR repeat protein 21A) (Stress-inducible protein 2)  TTC21A STI2 Homo sapiens (
Q5JXB2  UE2NL_HUMAN reviewed    Putative ubiquitin-conjugating enzyme E2 N-like (Epididymis tissue protein Li 174)  UBE2NL  Homo sapiens (Human)    15
Q9NRP4  SDHF3_HUMAN reviewed    Succinate dehydrogenase assembly factor 3, mitochondrial (SDH assembly factor 3) (SDHAF3)   SDHAF3 ACN9 DC11    Homo s
P49815  TSC2_HUMAN  reviewed    Tuberin (Tuberous sclerosis 2 protein)  TSC2 TSC4   Homo sapiens (Human)    1807    MAKPTSKDSGLKEKFKILLGLGTPRPNPRSAEGK
Q96FV3  TSN17_HUMAN reviewed    Tetraspanin-17 (Tspan-17) (F-box only protein 23) (Tetraspan protein SB134) (Transmembrane 4 superfamily member 17) TS
O95859  TSN12_HUMAN reviewed    Tetraspanin-12 (Tspan-12) (Tetraspan NET-2) (Transmembrane 4 superfamily member 12) TSPAN12 NET2 TM4SF12 UNQ774/PRO156
Q05066  SRY_HUMAN   reviewed    Sex-determining region Y protein (Testis-determining factor)    SRY TDF Homo sapiens (Human)    204 MQSYASAMLSVFNSDDYS
Q9Y2R5  RT17_HUMAN  reviewed    28S ribosomal protein S17, mitochondrial (MRP-S17) (S17mt) (Mitochondrial small ribosomal subunit protein uS17m)    MR
Q969E3  UCN3_HUMAN  reviewed    Urocortin-3 (Stresscopin) (Urocortin III) (Ucn III) UCN3 SPC    Homo sapiens (Human)    161 MLMPVHFLLLLLLLGGPRTGLPHKF
Q5T1N1  AKND1_HUMAN reviewed    Protein AKNAD1  AKNAD1 C1orf62  Homo sapiens (Human)    836 MDEADFSEHTTYKQEDLPYDGDLSQIKIGNDYSFTSKKDGLEVLNQIIFIADDPQEKA
P17213  BPI_HUMAN   reviewed    Bactericidal permeability-increasing protein (BPI) (CAP 57) BPI Homo sapiens (Human)    487 MRENMARGPCNAPRWASLMVLVAIGT
Q92772  CDKL2_HUMAN reviewed    Cyclin-dependent kinase-like 2 (EC 2.7.11.22) (Protein kinase p56 KKIAMRE) (Serine/threonine-protein kinase KKIAMRE)
Q9P0S2  COX16_HUMAN reviewed    Cytochrome c oxidase assembly protein COX16 homolog, mitochondrial (hCOX16) COX16 C14orf112 HSPC203 PTD019  Homo sapie
Q92685  ALG3_HUMAN  reviewed    Dol-P-Man:Man(5)GlcNAc(2)-PP-Dol alpha-1,3-mannosyltransferase (EC 2.4.1.258) (Asparagine-linked glycosylation protein
P15088  CBPA3_HUMAN reviewed    Mast cell carboxypeptidase A (MC-CPA) (EC 3.4.17.1) (Carboxypeptidase A3)   CPA3    Homo sapiens (Human)    417 MRLILP
Q96HB5  CC120_HUMAN reviewed    Coiled-coil domain-containing protein 120   CCDC120 JM11    Homo sapiens (Human)    630 MEVKGQLISSPTFNAPAALFGEAAPQVKSE
O95415  BRI3_HUMAN  reviewed    Brain protein I3 (pRGR2)    BRI3    Homo sapiens (Human)    125 MDHKPLLQERPPAYNLEAGQGDYACGPHGYGAIPAAPPPPPYPYLVTGIPTHHP
H3BNL1  CC084_HUMAN reviewed    Uncharacterized protein C3orf84 C3orf84 Homo sapiens (Human)    204 MQSALVGSWHNNGFYGHYRSQFKSESAREYHLAAKPQPPAVFLQRCQEPA
P16152  CBR1_HUMAN  reviewed    Carbonyl reductase [NADPH] 1 (EC 1.1.1.184) (15-hydroxyprostaglandin dehydrogenase [NADP(+)]) (EC 1.1.1.197) (NADPH-de
Q9H8M2  BRD9_HUMAN  reviewed    Bromodomain-containing protein 9 (Rhabdomyosarcoma antigen MU-RMS-40.8) BRD9 UNQ3040/PRO9856   Homo sapiens (Human)
Q1T7F1  CCB42_HUMAN reviewed    Putative chemokine-related protein B42      Homo sapiens (Human)    81  MPLSDWCCGICEEAPLGRAYTQTWMETGCGPHGVTALGQQELKDCL
```

```
extended crontab file    length : 14,796,320    lines : 20,352    Ln : 1    Col : 85    Sel : 73 | 1    Unix (LF)    UTF-8    INS
```

Figure 6: Open up the file and peruse it

Now that we know what our proteome file looks like, we can write a Python function to open it up and store it in a Pandas dataframe.

```python
    def load_uniprot(self, uniprot):
        """ Reads a UniProt file; returns a pandas dataframe

        Parameters
        ----------
        uniprot : file downloaded from UniProt containing a proteome of interest

        Returns
        -------
        proteome_dataframe, a Pandas dataframe that contains the FASTA IDs, sequences, and length of sequences """

        self.uniprot_proteome = pd.read_csv(uniprot, sep='\t', names=['Entry', 'Entry name', 'Status', 'Protein names', 'Gene names', 'Organism', 'Length', 'Sequence'])
        self.uniprot_proteome['Length'][1:] =  self.uniprot_proteome['Length'][1:].astype(int)  # convert the length values to integers

        return self.uniprot_proteome
```

We've now loaded the proteome into a Pandas dataframe that contains 3 columns:

1. **ID** = the UniProt ID
2. **Sequence** = the protein's sequence
3. **Length** = the protein's length.

To read the fasta file, we've made use of the SeqIO.index function in BioPython, which converts a FASTA file into a dictionary with keys and values respectively corresponding to IDs and sequences. With **SeqIO.index()**, one has a fast way to load very large FASTA files into Python without having to do a loop or loading of all lines into memory at once (e.g. with the Seq.list() or Seq.to_dict() functions).

With our proteome in an accessible data structure, we can begin to calculate some basic properties. For instance, we might want to know the total proteome length, in case we'd like to compute percentages.

```
    def proteome_length(self, query_proteome):
        """ Reads a Pandas dataframe containing a proteome of interest; returns the total number of residues

        Parameters
        ----------
        query_proteome : Pandas dataframe, contains the proteome of interest

        Returns
        -------
        total_length :  int, total number of residues in proteome_dataframe """

        total_length = query_proteome['Length'][1:].sum(axis=0, skipna=True)

        return total_length
```

This function takes a proteome in a Pandas dataframe as input and **returns the total number of residues in that proteome**. To quickly count up the number of residues in the proteome, we can sum the **Length** column that we've already appended to our dataframe in the **load_fasta** function above. The argument "axis=0" is required to tell Pandas to sum down the column and not across the row.

So, now that we have the total number of residues in the proteome, let's write a function to search the proteome for a specific motif. We want this function to output two values:

1. the total number of times that we counted the queried motif. Let's call this value ***sum***.
2. the observed number of motifs divided by the total number of peptides of size *n* in the proteome, or the ***fraction***.

Because we are counting up motifs in a "sliding" manner, *i.e.* by starting at residue 1 and working our way toward the C-terminus, then we should also calculate how many peptides of size n exist in a similar manner. For example, if our queried motif is "QVERY" then our denominator, x, becomes:

$$x = (N-n)+1$$

where N is the length of the proteome and n is the length of the queried motif. (I encourage you to calculate this on your own using, e.g. N = 10 and n = 2. You'll see that there exist 9 di-peptides when sliding from residue 1 to residue 10, i.e. 1-2, 2-3, 3-4, ... , 9-10).

```python
    def find_motifs(self, data_frame, query):
        """ Searches a Pandas dataframe containing a proteome for a specific amino acid/motif; returns the number of instances and fraction

        Parameters
        ----------
        data_frame : Pandas dataframe, contains the proteome of interest
        query :  str, the amino acid or motif to be searched

        Returns
        -------
        sum :  int, total number of hits for the queried residue/motif
        fraction:  float, sum divided by the total number of residues or motifs of the same size  """

        # Extract sequences from the pandas dataframe
        seqs = data_frame['Sequence']

        # Calculate the total number of residues in the queried proteome
        total_residues = self.proteome_length(data_frame)

        # Initialize an array
        hits_array = []

        # Loop over each sequence and count the number of times the queried motif shows up
        for protein in seqs:
            hits_array.append(protein.count(query))

        # total number of hits for the queried residue/motif, check if any hits are found
        sum = np.sum(np.asarray(hits_array))
        if sum < 1:
            fraction = 0
        else:
            fraction = sum/(total_residues - len(query)+1)  # total number of hits divided by the total number of residues or motifs of the same size

        return sum, fraction
```

Let's test out the code. We can quickly instantiate the class, load the FASTA file, and calculate the total length of the proteome.

```python
# Instantiate the class
human = Proteome()

# Load the FASTA file containing the human proteome
uniprot = human.load_uniprot('uniprot_human.tab') # replace with the name of your .tab file

# What is the length of the proteome?
print('Proteome length = %.0f AA' % human.proteome_length(uniprot))
```

> Proteome length = 11354232 AA

For the human proteome, we see that there are N = 11,354,232 residues. So, for our example above with the motif "QVERY", there are x = (11,354,232 − 5)+1 = 11,354,228 penta-peptides in the proteome.

With our new function **find_motifs**, we can also count how many particular motifs exist in the proteome.

```python
# Find how many Q, QV, QVE, QVER, QVERI, QVERIE, QVERIES
motif = 'QVERIES'
for i in range(0, len(motif)+1):
    print('%s = %.0f' % (motif[0:i], human.find_motifs(uniprot, motif[0:i])[0]))
```

```
> Q = 541533
> QV = 31622
> QVE = 2108
> QVER = 116
> QVERI = 6
> QVERIE = 2
> QVERIES = 0
```

By iterating over the queried motif "QVERIES", we see that the number of found motifs dramatically decreases with increasing length of the peptide. In fact, in the entire proteome comprising 11,354,226 hepta-peptides, **there is not a single QVERIES motif to be found**. We'll return to this point a little later.

We now know that there are about **11.4 million amino acids** in the proteome. Indeed, it would take some time to count motifs by hand...! We can do some basic calculations with these numbers, and we find that **the average length of a human protein is 558 residues**, or *ca* 61 kDa (using an estimate of *ca* 110 Da per amino acid).

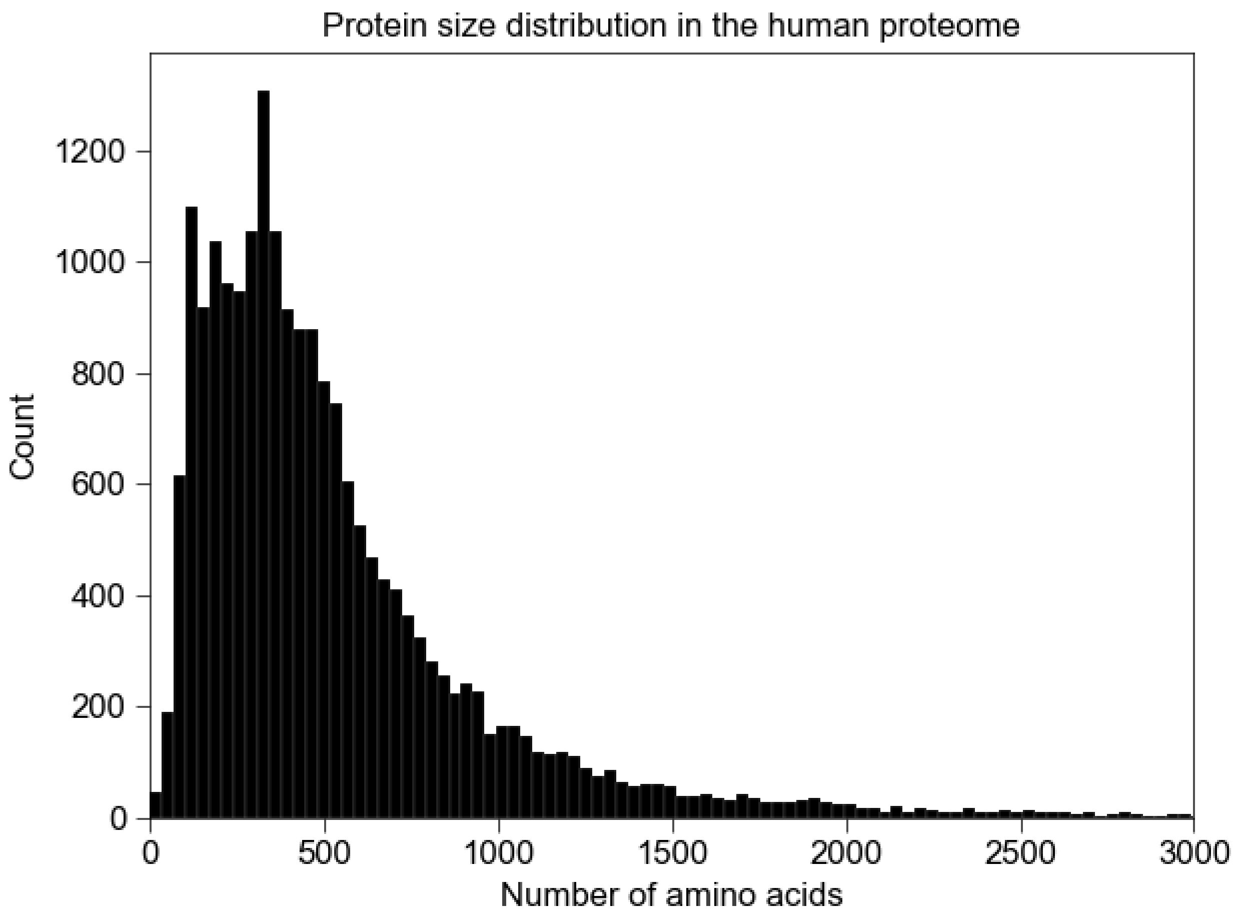
We can also compute this with numpy, where we obtain the same average length of 558 residues. The median length, however, is only 415 residues, and this reflects a skewed distribution.

```python
# What are the mean & median protein lengths in the proteome?
mean = np.mean(uniprot['Length'][1:])
median = np.median(uniprot['Length'][1:])
print('Mean protein length = %.0f AA' % mean)
print('Median protein length = %.0f AA' % median)
```

> > Mean protein length = 558 AA
> > Median protein length = 415 AA

We can plot a histogram of protein lengths to visualize the distribution over the proteome. I limited the x axis because the vast majority of proteins are fewer than 3000 residues. Indeed, we can see that our distribution of protein sizes (below) are skewed to the right with a tail.

```python
# Plot a histogram of sequence length
plt.hist(uniprot['Length'][1:], bins=1000, color='k', edgecolor='w', linewidth=0.1)
plt.title('Protein size distribution in the human proteome')
plt.ylabel('Count')
plt.xlabel('Number of amino acids')
plt.xlim(0,3000)
plt.tight_layout()
plt.show()
```

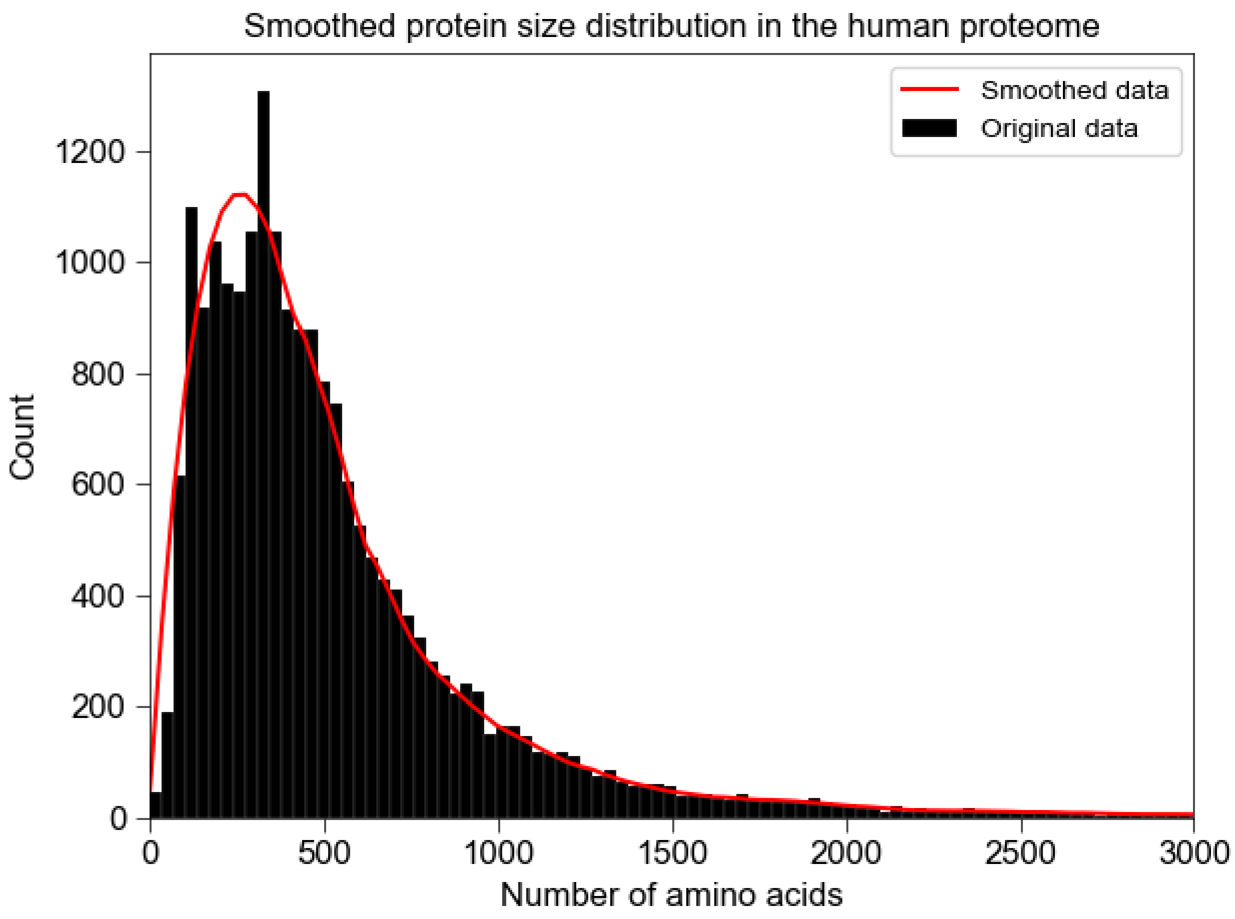


Histogram of protein lengths in the human proteome

We can smooth out our distribution by applying a filter known as the Savitzky-Golay filter.

```
from scipy.signal import savgol_filter

def filter(data_to_filter, window=19, poly=3):
        """ Applys the Savitzky-Golay filter to a 1D array

        Parameters
        ----------
        data_to_filter : dataframe or array, contains the data to be smoothed
        window :  int, the length of the filter window
        poly : int, the order of the polynomial (must be < window)

        Returns
        -------
        filtered :  array, the smoothed data """
    data = np.asarray(data_to_filter).astype(np.float64)
    filtered = savgol_filter(data, window, poly)

    return filtered

# Bin the data
```

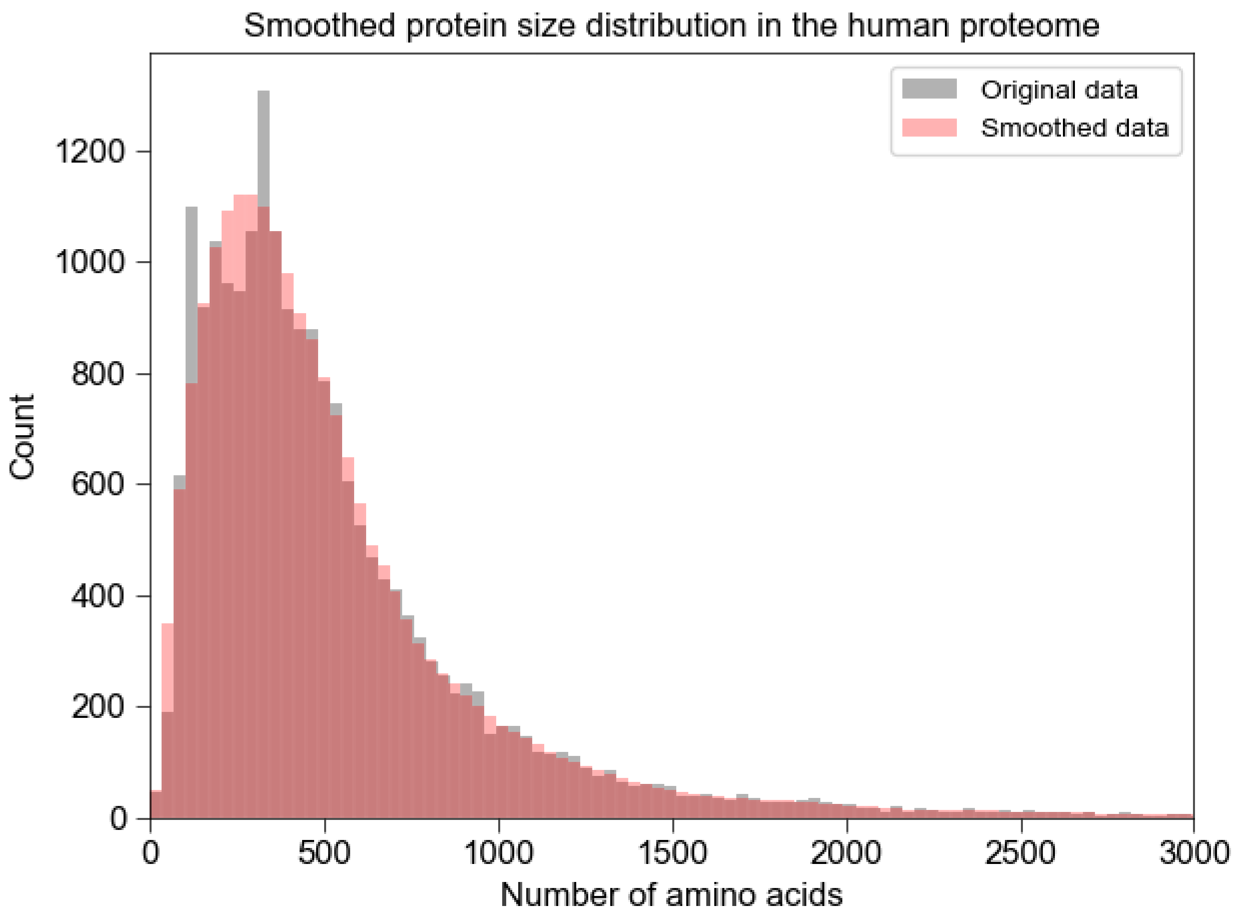
```
data, bins = np.histogram(np.asarray(uniprot['Length'][1:]), bins=1000, density=False)

# Smooth the binned data
filtered_data = filter(data, window=19, poly=3)

# Plot the histogram & the smoothed data
plt.hist(uniprot['Length'][1:], bins=1000, color='k', edgecolor='w', linewidth=0.1,
label='Original data')
plt.plot(bins[:-1], filtered_data, 'r', label='Smoothed data')
plt.title('Smoothed protein size distribution in the human proteome')
plt.ylabel('Count')
plt.xlabel('Number of amino acids')
plt.xlim(0,3000)
plt.ylim([0,None])
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```



Smoothing the histogram with a Savitzky-Golay filter

We can also recover the histogram-style plot of our smoothed data by making use of the **weights** variable in the plt.hist function:

```python
# Plot our histogram as above
plt.hist(uniprot['Length'][1:], bins=1000, density=False, color='k', alpha=0.5,
edgecolor='w', linewidth=0.1, label='Original data')

# Plot the smoothed data as a histogram by setting weights = filtered_data
plt.hist(bins[:-1], bins=1000, weights = filtered_data, density=False, color='r',
alpha=0.5, edgecolor='w', linewidth=0.1, label='Smoothed data')

# These are the exact same plotting parameters as above
plt.title('Smoothed protein size distribution in the human proteome')
plt.ylabel('Count')
plt.xlabel('Number of amino acids')
plt.xlim(0,3000)
plt.ylim([0,None])
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```



Smoothing the histogram with a Savitzky-Golay filter and recovering the binned values

Now that we've plotted our smoothed distribution, we can compare this to other organisms. Do humans, on average, have shorter or longer proteins that some model organisms like *E. coli* or *S. cerevisiae*? To test this, we can simply download the respective proteomes from UniProt using the steps outlined above. You can click these links to reach the *E. coli* proteome or the *S. cerevisiae* proteome that I used for the calculations below.

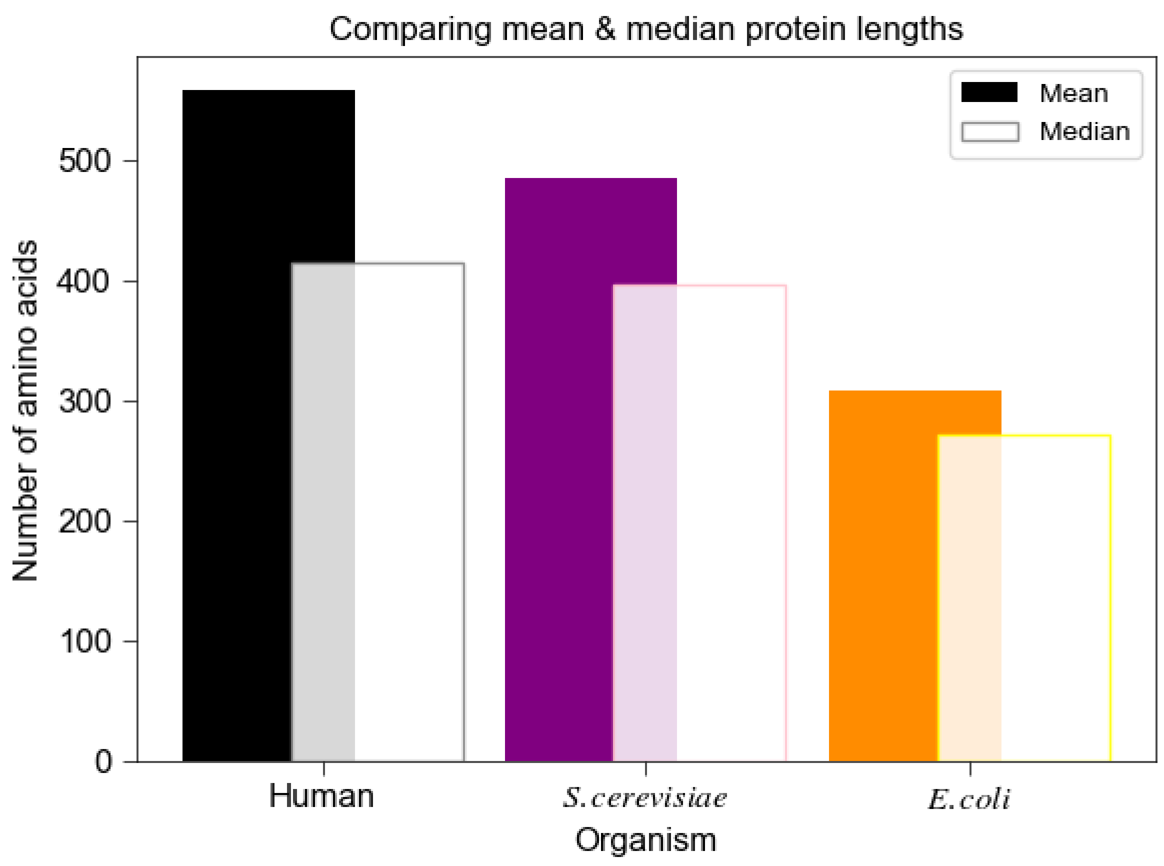
First, we see from the UniProt download pages that these two proteomes are much smaller than the human proteome: *E. coli* and *S. cerevisiae* respectively contain ~4,400 and ~6,000 proteins, which pale in number compared to the ~20,400 found in humans. Thus, to plot these distributions on the same y-axis, we must set the **density** option to **True** in our histograms, which ensures that the area under the histogram sums to 1. This enables us to compare histograms on the same y axis.

We can set up our script to compare these distributions:

```python
# Instantiate the class
human = Proteome()

# Load the FASTA file containing the human proteome
uniprot = human.load_uniprot('uniprot_human.tab') # replace with the name of your .tab file
ecoli = human.load_uniprot('uniprot_ecoli_k12.tab')
yeast = human.load_uniprot('uniprot_yeast.tab')

# How many amino acids are in each proteome?
print('\n')
print('Human has %.0f AA' % human.proteome_length(uniprot))
print('Yeast has %.0f AA' % human.proteome_length(yeast))
print('E coli has %.0f AA' % human.proteome_length(ecoli))
print('\n')

# What are the mean & median lengths of a protein in each proteome?
print('Mean (median) human protein = %.0f AA (%.0f AA)' % (human.proteome_length(uniprot)/(len(uniprot['Length'][1:])), np.median(uniprot['Length'][1:])))
print('Mean (median) yeast protein = %.0f AA (%.0f AA)' % (human.proteome_length(yeast)/(len(yeast['Length'][1:])), np.median(yeast['Length'][1:])))
print('Mean (median) E coli protein = %.0f AA (%.0f AA)' % (human.proteome_length(ecoli)/(len(ecoli['Length'][1:])), np.median(ecoli['Length'][1:])))

```

```
> Human has 11354232 AA
> Yeast has 2936363 AA
> E coli has 1354187 AA
> Mean (median) human protein = 558 AA (415 AA)
> Mean (median) yeast protein = 485 AA (396 AA)
> Mean (median) E coli protein = 309 AA (271 AA)
```

From this analysis, we can see a couple of interesting features. First, the average human protein (558 AA) is longer than the average yeast protein (485 AA) and nearly twice as long the *E. coli* average (309 AA). Interestingly, however, the median protein sizes from human and yeast are roughly equivalent, which suggests that the human proteome has a long(er) tail that includes some very large proteins.

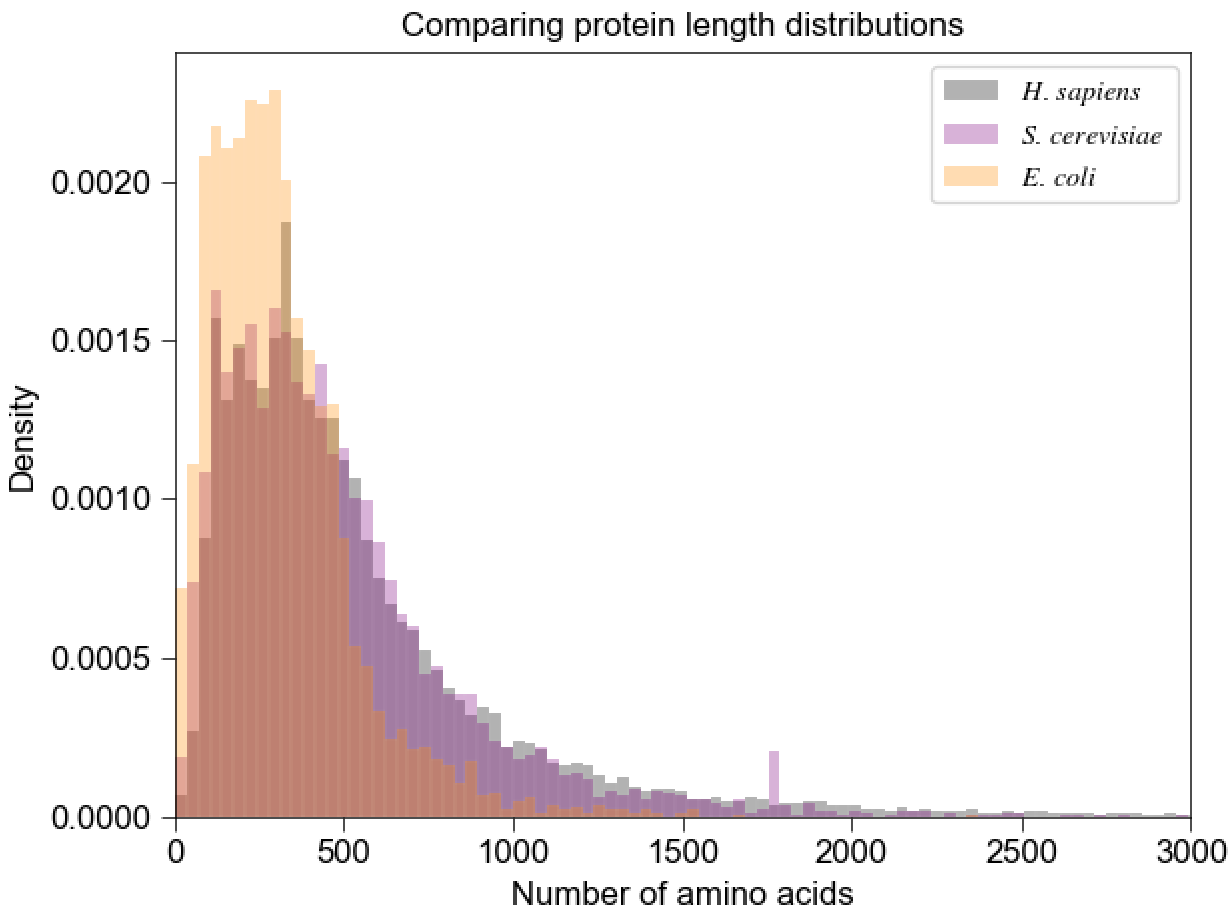


Comparing the mean and median protein lengths across three different organisms

```python
# Plot the mean & median protein lengths in a bar graph
fig = plt.figure(figsize=(6,4.5))
ax = fig.add_subplot(111)
ax.bar(1, human.proteome_length(uniprot)/(len(uniprot['Length'][1:])), color='k',
label='Mean')
ax.bar(1.5, np.median(uniprot['Length'][1:]), color='w', edgecolor='grey', alpha=0.85,
label='Median')
ax.bar(2.5, human.proteome_length(yeast)/(len(yeast['Length'][1:])), color='purple')
ax.bar(3, np.median(yeast['Length'][1:]), color='w', edgecolor='pink', alpha=0.85)
ax.bar(4, human.proteome_length(ecoli)/(len(ecoli['Length'][1:])), color='darkorange')
ax.bar(4.5, np.median(ecoli['Length'][1:]), color='w', edgecolor='yellow', alpha=0.85)

# Axis labels, legend, and title
ax.set_ylabel('Number of amino acids')
ax.set_xlabel('Organism')
ax.legend(loc='upper right')
ax.set_title('Comparing mean & median protein lengths')
```

```

# Tick positions & labels
ax.set_xticks([1.25, 2.75, 4.25])
ax.set_xticklabels(['Human', r'$S$. $cerevisiae$', r'$E$. $coli$'])

plt.tight_layout()
plt.show()
```

We can also plot these data in histogram format as follows:

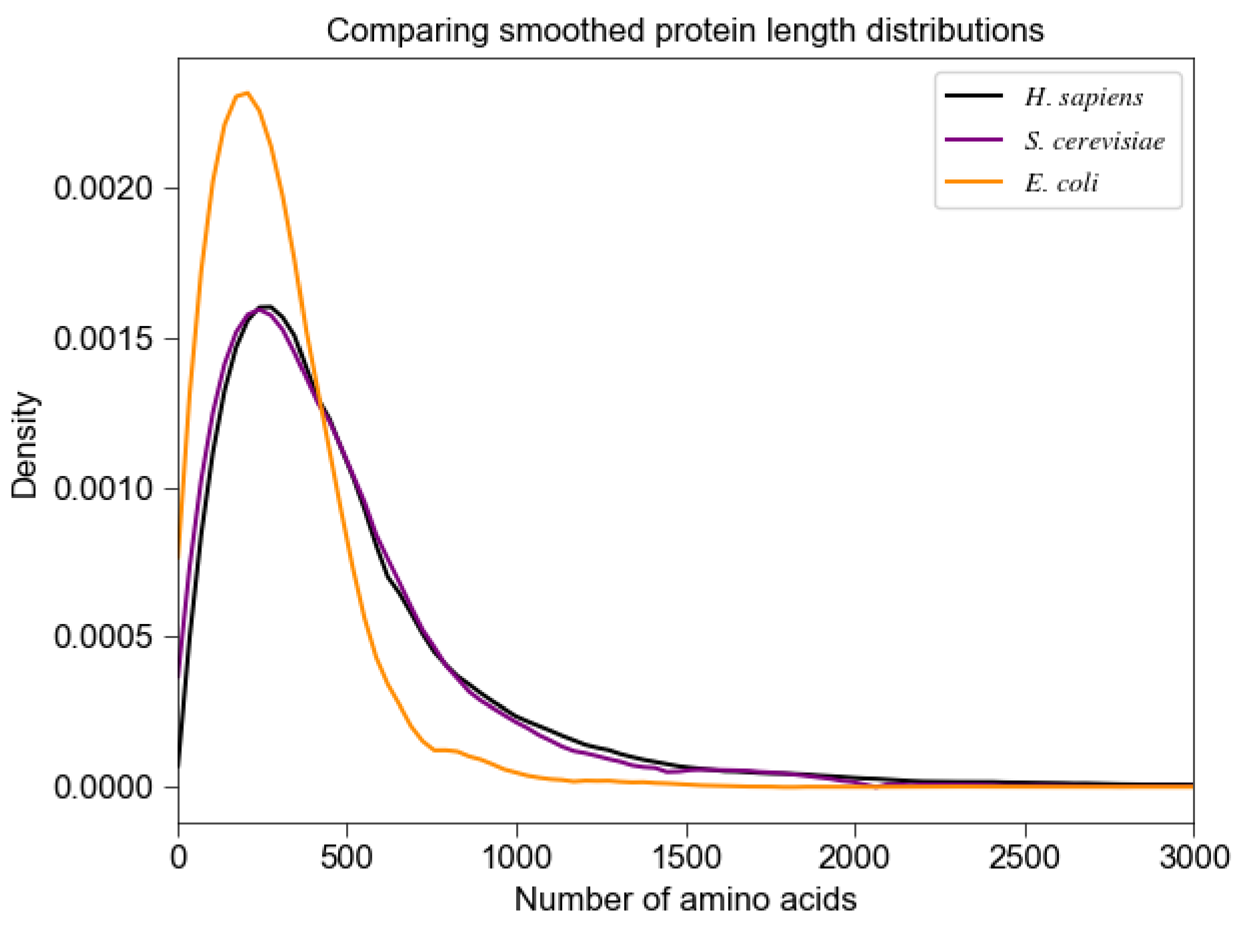


Comparing the distributions of protein lengths for three different organisms

```
# Get the bins from the human distribution
human_dat, human_bins = np.histogram(np.asarray(uniprot['Length'][1:]), bins=1000,
density=True)

# Plot the histograms using the same bins
plt.hist(uniprot['Length'][1:], bins=human_bins, color='k', alpha=0.3, edgecolor='w',
linewidth=0.1, label=r'$H$. $sapiens$', density=True)
```

```
plt.hist(yeast['Length'][1:], bins=human_bins, color='purple', alpha=0.3,
edgecolor='w', linewidth=0.1, label=r'$S$. $cerevisiae$', density=True)
plt.hist(ecoli['Length'][1:], bins=human_bins, color='darkorange', alpha=0.3,
edgecolor='w', linewidth=0.1, label=r'$E$. $coli$', density=True)

# Set up the axis limits, title, and legend
plt.xlim(0,3000)
plt.ylabel('Density')
plt.xlabel('Number of amino acids')
plt.title('Comparing protein length distributions')
plt.legend(loc='upper right')

# Plot
plt.tight_layout()
plt.show()
```

or with our smoothing function:



Comparing the smoothed distributions of protein lengths for three different organisms

```python
# Histogram the data using the same bins
human_dat, human_bins = np.histogram(np.asarray(uniprot['Length'][1:]), bins=1000, density=True)
ecoli_dat, ecoli_bins = np.histogram(np.asarray(ecoli['Length'][1:]), bins=human_bins, density=True)
yeast_dat, yeast_bins = np.histogram(np.asarray(yeast['Length'][1:]), bins=human_bins, density=True)

# Plot the smoothed data as lines
plt.plot(human_bins[:-1], filter(human_dat), 'k-', label=r'$H$. $sapiens$')
plt.plot(yeast_bins[:-1], filter(yeast_dat), '-', color='purple', label=r'$S$. $cerevisiae$')
plt.plot(ecoli_bins[:-1], filter(ecoli_dat), '-', color='darkorange', label=r'$E$. $coli$')

# Set up the axis limits, title, and legend
plt.xlim(0,3000)
plt.ylabel('Density')
plt.xlabel('Number of amino acids')
plt.title('Comparing smoothed protein length distributions')
plt.legend(loc='upper right')

# Plot
plt.tight_layout()
plt.show()
```

For me, it was surprising to see how much the yeast proteome resembles the human proteome! Maybe to an evolutionary biologist this would not be surprising; however, to a structural biologist like myself, I never would have guessed that the proteome of the tiny, single-celled *S. cerevisiae* could look so similar to humans. Alongside protein length, perhaps the hubristic tendencies of humans have also been evolutionarily conserved...

The histograms above tell us **three things**:

1. there are many more small proteins in *E. coli*
2. proteins became significantly longer between the appearance of *E. coli* and that of *S. cerevisiae*
3. the distribution of protein lengths has not changed much between *S. cerevisiae* and humans.

Above we compared the protein length distribution of a prokaryote, *E. coli*, to two eukaryotes, the single-celled organism *S. cerevisiae* and far more complex *H. sapiens*. **What do protein length distributions from other prokaryotes or archaea look like**? **How do proteins in human mitochondria compare**?

I downloaded the proteomes (using the above outline) from the following organisms:

1. *Methanocaldococcus jannaschii* (an archaeon)
2. *Thermus thermophilus* (a bacterium)
3. *Bacillus subtilis* (a bacterium)

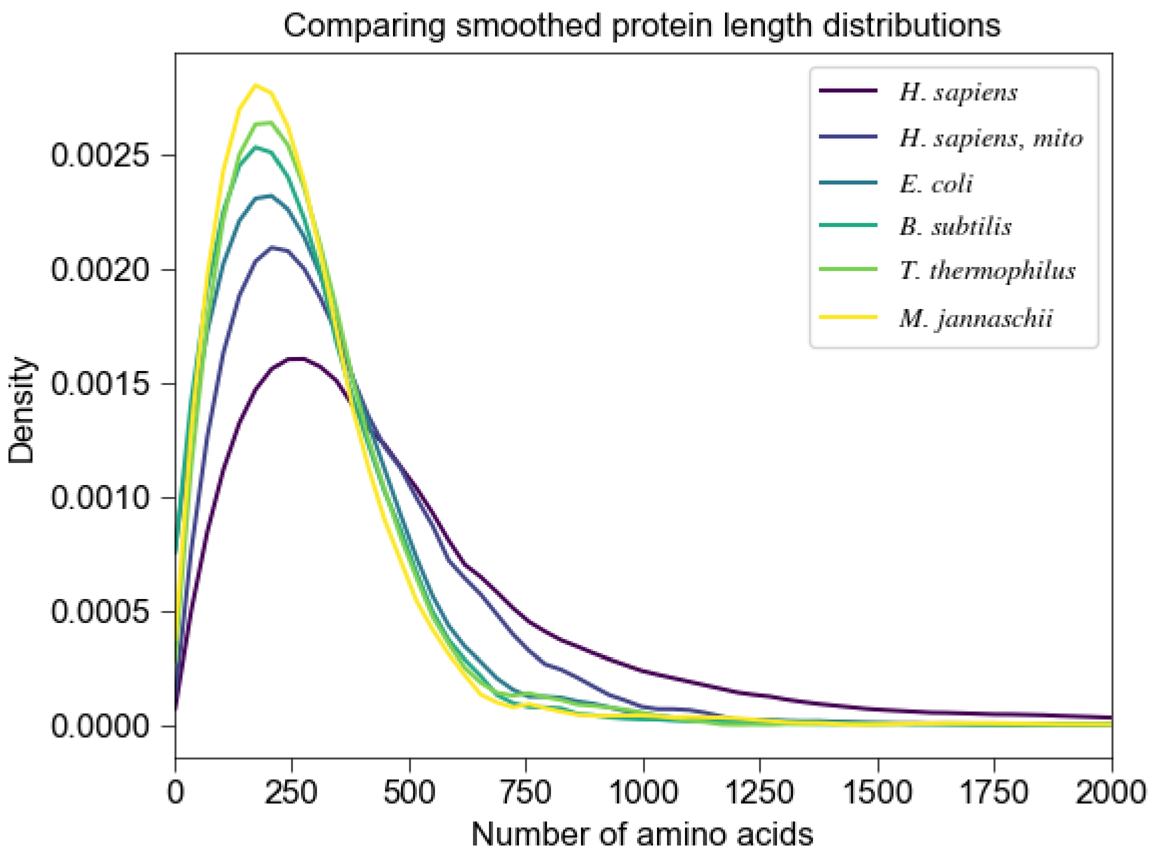
4. and the mitochondrial proteome from *Homo sapiens*

The mitochondrial proteome was made available by The Broad Institute in their MitoCarta2.0 release, which can be found in the following publication.

```python
# Load the new proteomes
mj = human.load_uniprot('m_jannaschii.tab')
bs = human.load_uniprot('b_subtilis.tab')
tt = human.load_uniprot('t_thermophilus.tab')

# in order to load the mitochondrial proteome, we need a new function
# that can read generic FASTA files and return a pandas dataframe

def load_fasta(self, fasta):
    """ Reads a fasta file with identifiers (IDs) and sequences; returns a pandas dataframe

        Parameters
        ----------
        fasta : file in FASTA format with identifiers (e.g. >) and sequences

        Returns
        -------
        fasta_proteome, a Pandas dataframe that contains the FASTA IDs, sequences, and length of sequences """

    # Load the fasta file
    self.seq_list = SeqIO.index(fasta, "fasta")  # makes dictionary, but contains Seq objects

    # Create arrays to store the relevant information
    ids = []  ; seqs = []
    for key, value in self.seq_list.items():
        ids.append(key)
        seqs.append(str(value.seq)) # string the Seq object go get only the sequence

        # Create a dictionary for subsequent conversion into a dataframe
        self.dictionary = {'ID':ids, 'Sequence':seqs}

        # Load the dictionary into a dataframe
        self.fasta_proteome = pd.DataFrame(self.dictionary)
        self.fasta_proteome['Length'] = self.fasta_proteome['Sequence'].str.len()  # create column in the dataframe with sequence lengths
        self.fasta_proteome['Length'][1:] =  self.fasta_proteome['Length'][1:].astype(int) # convert the length values to integers

        return self.fasta_proteome

# now load the mitochondrial proteome using "load_fasta"
mito = human.load_fasta('Human.MitoCarta2.0.fasta')
```

Now that the data have been loaded into pandas dataframes, we can plot the smoothed protein length distributions. To speed things up, we can write a function:

```python
def plot_list_hist(list_organisms, names):

    # get reference data
    ref_dat, ref_bins = np.histogram(np.asarray(list_organisms[0]['Length'][1:]), bins=1000, density=True)  # get reference bins for 1st item in list

    # make array
    data = np.zeros(shape=(len(list_organisms), len(ref_bins)-1))
    colors = plt.cm.viridis(np.linspace(0.0, 1, len(list_organisms)))

    # loop and plot
    fig = plt.figure(figsize=(6,4.5))
    for i, item in enumerate(list_organisms):
        dat, bins = np.histogram(np.asarray(item['Length'][1:]), bins=ref_bins, density=True)
        data[i, :] = dat
        plt.plot(ref_bins[:-1], filter(dat), '-', label=names[i], color=colors[i])

    plt.xlim(0,2000)
    plt.ylabel('Density')
    plt.xlabel('Number of amino acids')
    plt.title('Comparing smoothed protein length distributions')
    plt.legend(loc='upper right')
    plt.tight_layout()

    #plt.show()
    return fig

# create a list of data and data names
data = [uniprot, mito, ecoli, bs, tt,  mj]
data_names = [r'$H$. $sapiens$', r'$H$. $sapiens$, $mito$', r'$E$. $coli$', r'$B$. $subtilis$', r'$T$. $thermophilus$', r'$M$. $jannaschii$']

# use the "plot_list" function
plot_list_hist(data, data_names)
plt.show()
```



Comparing the smoothed distributions of protein lengths across organisms

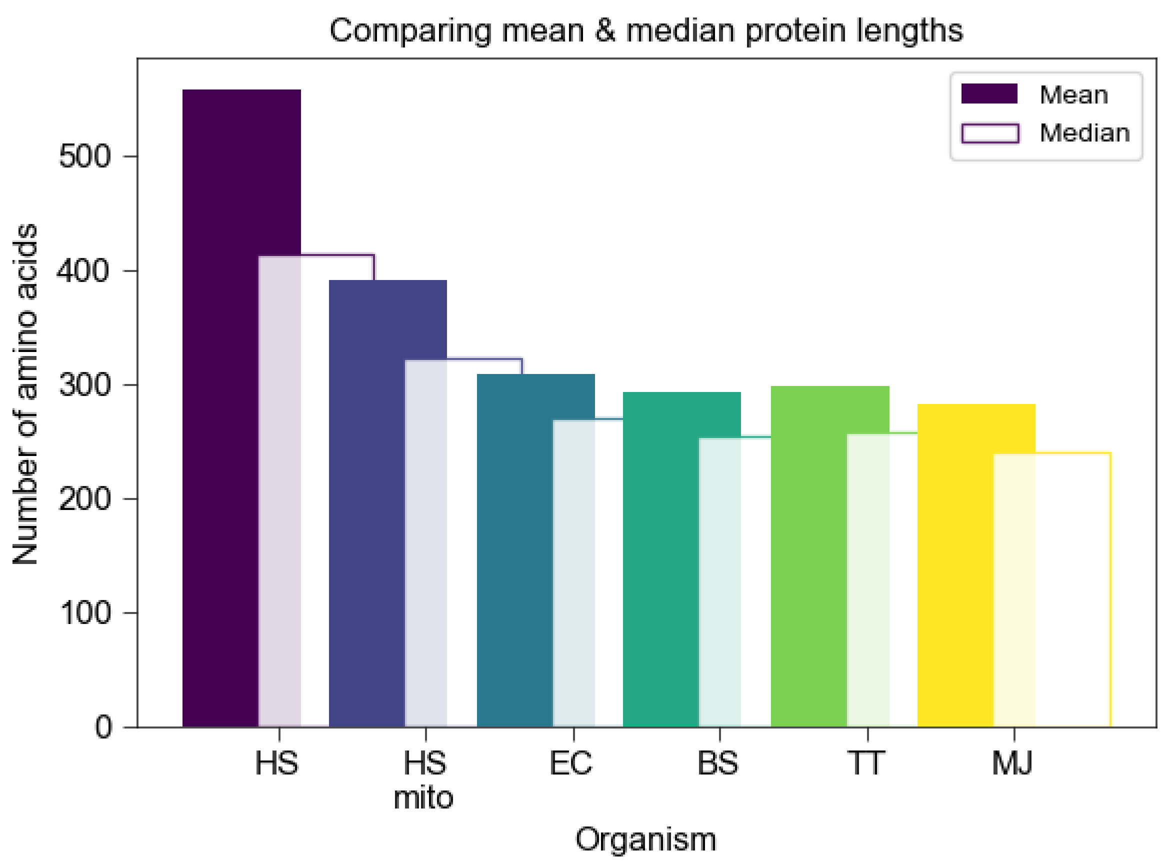
---

From this comparison, we can see that the **mitochondrial** proteome lies somewhere between the ***E. coli*** and ***human*** proteomes. Of course, our human proteome dataset also contains all of the mitochondrial proteins; so, had we selectively removed the ~1000 mitochondrial proteins from the human proteome, we would see a slightly more pronounced difference. Nevertheless, for proteins with fewer than 500 amino acids, the mitochondrial proteome looks more similar in terms of its length distribution to *E. coli*. For proteins longer than 500 residues, however, the long tail of the mitochondrial proteome bears more resemblance to human. Overall, these apparent differences may reflect the more ancient evolutionary origin of mitochondria: more smaller proteins than expected for the human proteome, but more larger proteins than expected for prokaryotes or archaea.

Finally, we can also plot the mean and median protein lengths, as we did earlier. To make this a little faster, however, we can create a small loop:

```
# create the figure
fig = plt.figure(figsize=(6,4.5))
ax = fig.add_subplot(111)

# create a list of colors using the colormaps package
abbrv_names = ['HS', 'HS\nmito', 'EC', 'BS', 'TT', 'MJ']
colors = plt.cm.viridis(np.linspace(0.0, 1, len(abbrv_names)))

# initiate a counter and empty array (for xtick positions)
```

```python
counter = 1
xtick_list = []

# now loop over the organisns
for i in range(len(abbrv_names)):
    if i == 0:
        ax.bar(counter, human.proteome_length(orgs[i])/(len(orgs[i]['Length'][1:])), color=colors[i], label='Mean')
        ax.bar(counter+0.5, np.median(orgs[i]['Length'][1:]), color='w', edgecolor=colors[i], alpha=0.85, label='Median')
    else:
        ax.bar(counter, human.proteome_length(orgs[i])/(len(orgs[i]['Length'][1:])), color=colors[i])
        ax.bar(counter+0.5, np.median(orgs[i]['Length'][1:]), color='w', edgecolor=colors[i], alpha=0.85)
    xtick_list.append(counter+0.25)
    counter += 1


# Axis labels & the legend
ax.set_ylabel('Number of amino acids')
ax.set_xlabel('Organism')
ax.legend(loc='upper right')

# Tick positions & labels
ax.set_xticks(xtick_list)
ax.set_xticklabels(abbrv_names)
ax.set_title('Comparing mean & median protein lengths')

#plt.xticks(rotation=0, fontsize=8)
plt.tight_layout()
plt.show()
```



Comparing the mean and median protein lengths across organisms. The abbrevations are as follows: Homo sapiens (HS), mitochondrial (mito), E. coli (EC), Bacillus subtilis (BS), Thermus thermophilus (TT), Methanocaldococcus jannaschii (MJ).

We can clearly see that the median protein length in the mitochondria is ~100 residues smaller than the typical human protein. The difference in the means is even more pronounced, which reflects the very long tail in the human proteome (as discussed above).

Next time, I'll discuss how we can use Python to calculate biophysical properties from protein sequences, and how we can perform such calculations on the proteome scale.