User data analysis

Main script: `user_data_crunch.py`

Webapp folder location: `PhosQuest\service_scripts`

Technologies and tools: `Python, Pandas, NumPy, statsmodels`

Strategy:

The script primarily utilises the `Pandas` library for data handling operations such as reading, defining and transforming data structures, filtering, subsetting, indexing, categorising, column insertion, merging of data-frames and various other tasks. `Pandas` is a software library written for the `Python` programming language and is optimised for data wrangling and analysis.

For specific operations such as float handling and statistics operations, `NumPy` and `statsmodels` packages are also utilised.

The overall aim of the script is to perform a sequential series of operations that transform, analyse, summarise and extract user data, such that visualisation is possible at a later stage.

These steps are handled by 6 inter-linked functions that have been divided into different data handling categories.

User data:

Data takes the form of a table comprising one identifier column (mixed text & numerical fields) and six integer/float columns for a total of seven categories, with the number of records varying from 1000's to 10000's of rows. The script is written to handle a table of these dimensions, however some flexibility is afforded to the user. See function: `user_data_check()` .

Functions:

1. `user_data_check()` : *Data table structure check and basic filtering.*

- **Input**: user data table.
- Implements an error check in case of issues such as missorting of the original table. The function calculates fold changes based on intensity column inputs and then checks if the values match the original uploaded values. Boolean values returned, are used to determine if the table should be passed for later processing or return an error message.

- Determines the format of the input table by checking how many columns are present. If CV columns missing, these columns are appended with entry values of 1. Summing these "mock" CV columns gives a value equal to the length of the dataframe. This is utilised at a later stage for filtering the data. See function: `table_sort_parse()` .
- Parse data entries that have at least 1 quantitation value.
- **Output**: filtered dataframe or error message.

2. `create_filtered_dfs()` : *Dataframe analysis and filtering.*

- **Input**: dataframe output of `user_data_check()` function.
- Splits ID entry into separate gene name and site columns.
- Performs a series of calculations, transformations and logical checks. This analysis is appended as extra columns in the table.
- Filters the dataframe to include only phospho-site entries.
- **Output**:
  1: analysed dataframe of phospho-hits only.
  2: dataframe of all entries.

3. `correct_pvalue()` : *Further analysis of p-values in filtered dataframe.* * **Input**: phospho-hits dataframe output of `create_filtered_dfs()` function. * `fdrcorrection()` function of **statsmodels** is used to correct p-values for multiple-testing errors. * Benjamini-Hochberg method for multiple testing correction utilised. * Permissible error rate = 0.05. * Analysis appended to dataframe as 2 extra columns: rejected hypotheses (boolean) and corrected p-values. * **Output**: dataframe with expanded analysis.

4. `table_sort_parse()` : *Table sorting and filtering.* * **Input**: phospho-hits dataframe output of `correct_pvalue()` function. * Dataframe columns re-organised into new order for legibility. * Sorting of dataframe using various categories calculated in preceding functions. * Call `link_ud_to_db()` function which cross-references user data with PhosQuest database. This analysis returns 2 dictionaries:
1: Dictionary 1 entries are appended as 3 extra columns into the main dataframe.
2: Dictionary 2 passed for further analysis. See function: `kinase_analysis()` . * Differential dataframe filtering according to various categories in previous analyses. If CV columns are not present in the original uploaded data, then only the corrected p-value threshold (<=0.05) is applied. If CVs present, then both CV and corrected p-value thresholds are utilised. * Dataframe of phospho-hits only is fed into the `user_data_volcano_plot()` function of the `plotting.py` script. * Dataframe of significant hits is fed into the `style_df()` function of the `plotting.py` script. * **Output**:
1: analysed dataframe of all phospho-hits only.
2: dataframe of significant phospho-hits.
3: kinase dictionary.

5. `data_extract()` : *Further analysis to generate metrics of the user data.* * **Input**:
1: phospho-hits only dataframe output of `table_sort_parse()` function.
2: all hits dataframe output of `user_data_check()` function. * 4 sets of analyses carried out to produce

a "Metrics" of the data for later visualisation. * Data is fed into the `pie_chart()` function of the `plotting.py` script. * **Output**: 3 dataframes and 1 integer value.
1: dataframe 1 - % enrichment.
2: dataframe 2 - Phosphorylated AA residue frequency distribution.
3: dataframe 3 - Multiple phosphorylation frequency distribution.
4: integer - number proteins represented in the data.

6. `kinase_analysis()` : *Kinase and corresponding substrate/sites analysis and kinase relative actvity calculations.* * **Input**:
1: kinase dictionary from `table_sort_parse()` function.
2: significant hits dataframe output of `table_sort_parse()` function. * **Analysis 1** - Kinase and corresponding substrate/sites analysis: * Kinase dictionary converted to a dataframe with each row a unique kinase (also index), with the first column a set of substrate/site(s) per kinase. * Dataframe transformed such that each kinase has 1 unique substrate/site per row. * Further calculation determines frequency distributions for each kinase and substrate/site. These are fed into the `wordcloud_freq_charts()` function of the `plotting.py` script. * Lists of kinases and substrate/sites are each converted to lists of strings. These are fed into the `wordcloud_freq_charts()` function of the `plotting.py` script. * **Analysis 2** - kinase relative activity calculation: * Subset of significant hits dataframe generated and hits with both intensities reported are parsed. * Dataframe generated in **Analysis 1** (converted kinase dictionary) is merged with significant hits subset, based on matching substrate/sites. * Calculate mean of absolute log2 fold changes. * Calculate sum of log2 fold changes and then pass sign (positive/negative) to mean of absolute log2 fold changes. * These relative activity values are mapped to the corresponding list of substrate/site(s) as a new dataframe. This data is fed into the `style_df()` function of the `plotting.py` script. * **Output**: 2 dataframes and 2 word lists.
1: dataframe 1 - Kinase frequency distribution analysis.
2: dataframe 2 - Substrate/site frequency distribution analysis.
3: word list 1 - kinases.
4: word list 2 - substrate/site.

***Limitations and further work*:**

1. Generally the script is somewhat flexible with regards to user data input i.e. CV columns may or not be present and non-phosphorylated entries in the table can also be gene-name only entries . However, users will likely use a number of different proteomics software packages to generate their data. The raw format of these output tables will necessitate some data wrangling, pre-upload to PhosQuest. This increases the chances of table format issues i.e. such as missorting of columns (which the crunch script partially takes into account). Improvements in `user_data_check()` could expand on the number of checks available i.e. are p-values not assigned to hits with intensities reported and are CVs assigned for hits with intensity of 0 reported in both replicates.
2. The analysis could be made more interactive, by taking in user cut offs for variables such as %CV and p-value.

3. Code a template that takes a raw output table from the user i.e. closer to the native analysis of proteomics software packages. Intensity columns, for example, maybe reported for the separate replicates of each experiment i.e. control and treatment/condition. The advantage of this approach, is that less needs to be done by the user and it should decrease the likely hood of issues cropping-up such as those mentioned earlier. The disadvantage is that the burden of averaging intensities, calculating CVs and p-values to name a few examples, will fall on the script. This will necessitate a substantial expansion of the code base.
4. `correct_pvalue()` function could be expanded to include other methods such as the more stringent Bonferroni method.
5. Currently there are a number of "metrics" calculated by the `kinase_analysis()` function, that could be used to create a small summary table to compliment the current visuals. These include measures such as the number of unique kinases that map to user data substrate/sites (and vice-versa) and the sum of kinase and substrate/site frequencies.
6. The full analysis table can be downloaded from PhosQuest, however, it could be improved as follows. The `Pandas` style function can be adjusted to export the data also. However, it was found that not all the styling is exported to the csv i.e. the barplots overlaying the log2 fold change column.