About the project

The end goal of this project is to classify patients with high protein concentration in urine and the healthy group based on SERS (Surface Enchanced Raman Spectroscopy) spectral data and biomedical data.
This project is to be released as a research paper later in 2022 or 2023. Some information is not fully shown here as a result.

The project is divided into several Jupyter notebooks with the following names: 1) Import raw urine spectra (part 1) 2) Spectra processing (part 2) 3) Classification of patients (part 3)

Author of all codes: Sultan Aitekenov, sultanaitekenov@gmail.com

Part of the upcoming abstract: Excessive protein excretion in human urine is an early and sensitive marker of diabetic nephropathy, primary and secondary renal disease. Kidney problems, particularly chronic kidney disease, remain among the few growing causes of mortality in the world. Therefore, it is important to develop efficient, expressive, and low-cost method for protein determination. Surface enhanced Raman spectroscopy (SERS) methods are potential candidates to achieve those criteria. In this paper, the SERS methods was developed to distinguish patients with proteinuria and the healthy group. Commercial gold nanoparticles with the diameter of 60 nm and 100 nm, and silver nanoparticles with the diameter of 100 nm were employed. Silver, gold, silicon and test slides covered with aluminium tape were utilized as substrates. Obtained spectra were analysed with several machine learning algorithms coupled with the PCA, ROC curve, and cross-validation methods.

Import raw urine spectra (part 1)

The end goal of this notebook is to create a dictionary file that contains information about each patient (keys) and their respective spectra (values). This script runs for 181 sec.

Imports important modules

In [1]:
```
# imports modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import glob # imports to find nested files
import os
import re
```

Searches for file names and detects corrupted files

The spectral data is saved in hundreds of txt files. The txt files were generated by the software called LabSpecthe that operates a Raman microscope.

In [2]:
```
# search for file names
path = "Data raw urine spectra/"
substrate_folders = glob.glob(path + "/**NPs")
patients_folders = glob.glob(path + "/**NPs/**[0-9]")
txt_naming = glob.glob(path + "/**NPs/**[0-9]/*.txt")
```

In [3]:
```
# visualize txt_naming, first 5 entries
txt_naming[:5]
```

Out[3]:
```
['Data raw urine spectra\\Ag_100nm_AgNPs\\1\\1-1.txt',
 'Data raw urine spectra\\Ag_100nm_AgNPs\\1\\1-2.txt',
 'Data raw urine spectra\\Ag_100nm_AgNPs\\1\\1-3.txt',
 'Data raw urine spectra\\Ag_100nm_AgNPs\\1\\2-1.txt',
 'Data raw urine spectra\\Ag_100nm_AgNPs\\1\\2-2.txt']
```

From the output below, a reader can understand how the data is organized. The cell below finds "corrupted" files.

In [4]:
```
# Finds corrupted files and save them as strings in a list. Some files are corrupted.
# This block finds that files by bytes size. Corrupted files take more than 1 mb.
corrupted_files = []
for i in range(0, len(txt_naming)):
    check_size=os.stat(txt_naming[i])
    size=check_size.st_size
    if size > 1000000:
        corrupted_files.append(txt_naming[i])
corrupted_files
```

Out[4]:
```
['Data raw urine spectra\\Ag_100nm_AgNPs\\1\\3-1.txt',
 'Data raw urine spectra\\Ag_100nm_AgNPs\\107\\3-1.txt',
 'Data raw urine spectra\\Ag_100nm_AgNPs\\108\\1-1.txt',
 'Data raw urine spectra\\Ag_100nm_AgNPs\\116\\2-2.txt',
 'Data raw urine spectra\\Ag_100nm_AgNPs\\62\\3-1.txt',
 'Data raw urine spectra\\Ag_100nm_AuNPs\\108\\2-1.txt',
 'Data raw urine spectra\\Ag_100nm_AuNPs\\110\\3-1.txt',
 'Data raw urine spectra\\Ag_100nm_AuNPs\\115\\2-1.txt',
 'Data raw urine spectra\\Al_tape_100nm_AuNPs\\113\\1-1.txt',
 'Data raw urine spectra\\Al_tape_60nm_AuNPs\\104\\1-1.txt',
 'Data raw urine spectra\\Au_60nm_AuNPs\\36\\3-1.txt']
```

In [5]:
```
# Remove corrupted files by finding their names (filepath).
for i in range(0, len(corrupted_files)):
    txt_naming.remove(corrupted_files[i])
```

Create dictionaries: Keys - patients, Value - spectra

Firstly, a function that organizes data into a dictionary is created.

In [6]:
```
def raman_create_dict(files_path, substrate):
    """
    example: files_path = 'Data raw urine spectra\\Au_100nm_AuNPs\\99\\3-3.txt',
    "Au_100nm_AuNPs" refers to experimental set
    99 - ID of a patient
```

```
    3-3.txt - contains spectral data

    Input:
    1) files_path
    2) substrate - a string with only these arguments:
    Ag_100nm_AgNPs,
    Ag_100nm_AuNPs,
    Al_tape_60nm_AuNPs,
    Al_tape_100nm_AuNPs,
    Au_60nm_AuNPs,
    Au_100nm_AuNPs,
    Si_60nm_AuNPs

    Output:
    dictionary with Keys - patients, and Values - spectra
    """

     #find a relevant list containing relevant paths
    rel_path = []
    for file_path in files_path:
        a = re.search(substrate, file_path) #finds patients within a single set of a s
        if a != None:
            rel_path.append(file_path)
    rel_files_path = rel_path

    # create empty dictionary with numeric key values to delete later
    # our research group does not have ID higher than 200, so 300 would be more than e
    dict_raw_spectra = {}
    for i in range(0,300):
        dict_raw_spectra[i] = []

    # create matrix from relevant path
    for file_path in rel_files_path:
        # keys in dict
        y = re.findall(r"\\([0-9]*)\\", file_path)
        key = int(y[0])

        # values in dict
        value = pd.read_table(file_path)
        value_sliced = np.array(value.iloc[:,2:])
        # converts to dt
        # if statement prevents files with incorrect numbers of row to pass
        if value_sliced.shape[0] <= 100:
            dict_raw_spectra[key].append(value_sliced)

    # delete keys without values, if keys have values, this script concatenates all va
    for key in list(dict_raw_spectra.keys()):
        if len(dict_raw_spectra[key]) == 0:
            del dict_raw_spectra[key]
        else:
            value=np.concatenate(dict_raw_spectra[key])
            dict_raw_spectra[key]=value

    return dict_raw_spectra
```

The cell below creates dictionary for each substrate. Since 7 experimental sets are available, each of them are saved into the single dictionary called 'raw_urine_spectra'

In [7]:
```
# create a dictionaries for each substrate (experimental set)
```

```
Ag_100nm_AgNPs = raman_create_dict(txt_naming, "Ag_100nm_AgNPs")
Ag_100nm_AuNPs = raman_create_dict(txt_naming, "Ag_100nm_AuNPs")
Al_tape_60nm_AuNPs = raman_create_dict(txt_naming, "Al_tape_60nm_AuNPs")
Al_tape_100nm_AuNPs = raman_create_dict(txt_naming, "Al_tape_100nm_AuNPs")
Au_60nm_AuNPs = raman_create_dict(txt_naming, "Au_60nm_AuNPs")
Au_100nm_AuNPs = raman_create_dict(txt_naming, "Au_100nm_AuNPs")
Si_60nm_AuNPs = raman_create_dict(txt_naming, "Si_60nm_AuNPs")

# create dictionary of dictionaires
raw_urine_spectra = {"Ag_100nm_AgNPs": Ag_100nm_AgNPs,
                     "Ag_100nm_AuNPs": Ag_100nm_AuNPs,
                     "Al_tape_60nm_AuNPs": Al_tape_60nm_AuNPs,
                     "Al_tape_100nm_AuNPs": Al_tape_100nm_AuNPs,
                     "Au_60nm_AuNPs": Au_60nm_AuNPs,
                     "Au_100nm_AuNPs": Au_100nm_AuNPs,
                     "Si_60nm_AuNPs": Si_60nm_AuNPs}
```

Saves output dictionaries into pickle

In [8]:

```
# save the output dictionary into a pickle file
import pickle

with open("raw_urine_spectra.pkl","wb") as file:

    pickle.dump(raw_urine_spectra,file)
```