Final Project Capstone 2 Report

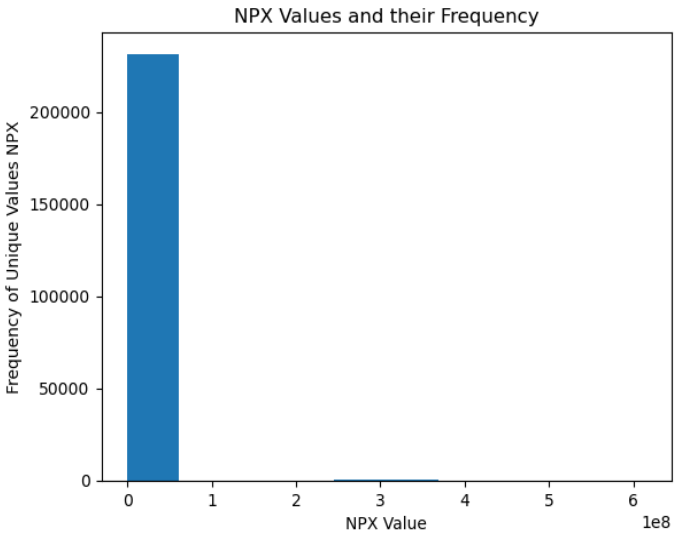
Maria Meza

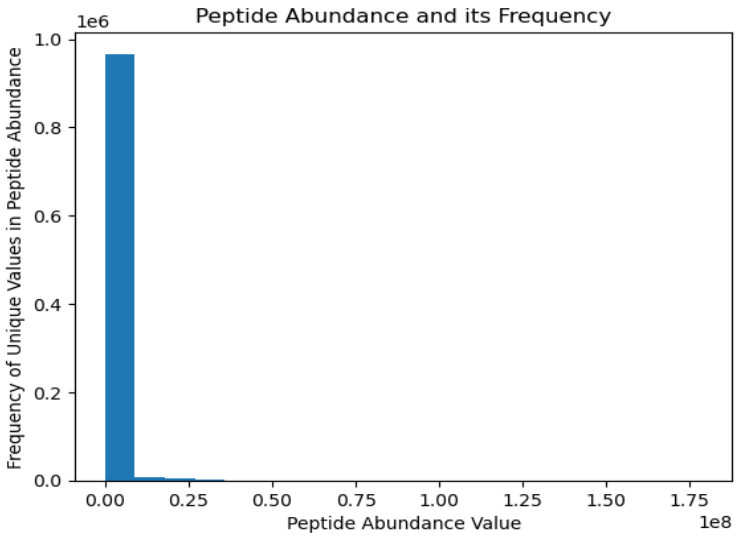
1. Introduction

Parkinson's disease affects the daily lives of people usually over sixty, but it can occur to anyone of any age. However, unlike other diseases, which might have an obvious diagnostic test, Parkinson's disease is not easy to test. Nonetheless, Parkinson's is tested by checking a patient's mood, behavior, and motor symptoms. For example, a patient with Parkinson's may exemplify issues with walking or may present tremors. The severity of Parkinson's is measured by a UPDR score that has four categories. UPDR 1 focuses on nonmotor experiences. UPDR2 focuses on daily motor experiences; UPDR3 focuses on motor issues. While UPDR4 is the most severe, focusing on motor complications. Each UPDR ranking has a scale that starts at zero. The higher the ranking, the more severe the condition. For this capstone project, I will make a model that can predict the UPDRs ranking of patients with Parkinson's based on protein abundance, unique proteins present, peptide abundance, and unique peptides present, time, and visit ID.

2. Exploratory Data Analysis

The data set in question had both quantitative and qualitative values. Yet, the variables that seem to play a major importance in this experiment seem to be the quantitative values. Thus, I explored with NPX, and protein Abundance first by creating a bar graph to see the distribution. These values did not have a normal distribution. NPX did not have a normal distribution because the range of its values was from 84 to 613851000. Furthermore, most values were below 100000000, with only a few values over200000000.

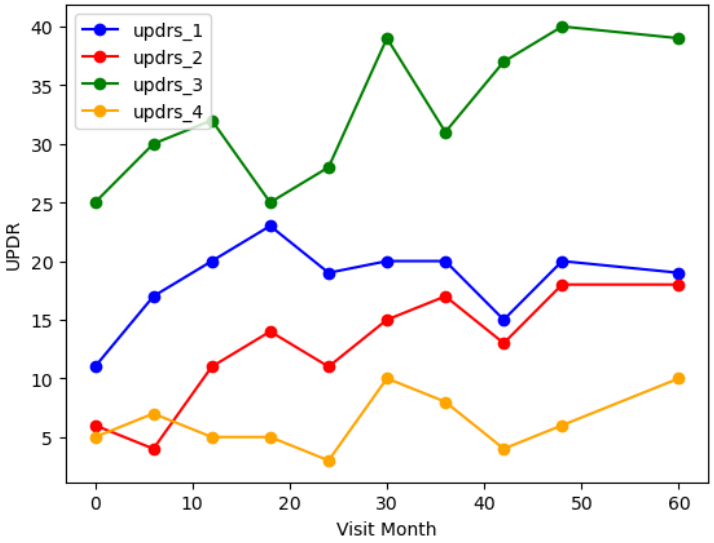


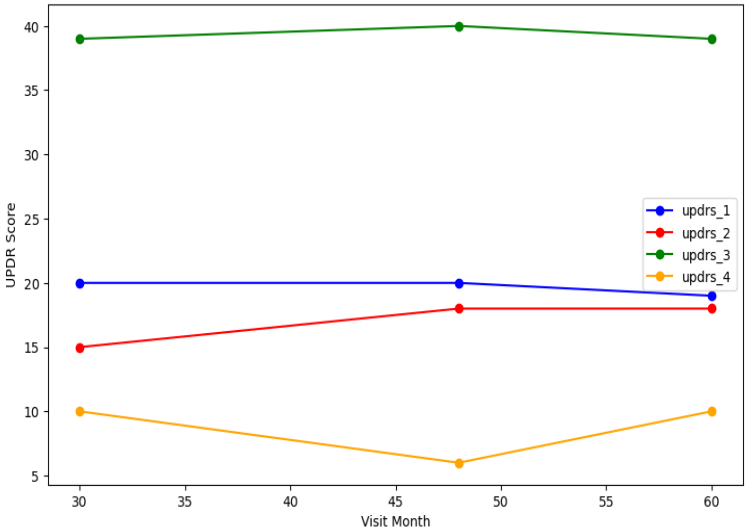


A similar pattern can be seen for the peptide abundance values, with minimum value of 10 and a maximum value of 178752000. These wide ranges tell me that it is most likely that a scaler will need to be used to avoid data leakage.

The data set in question had both quantitative and qualitative values. Yet, the variables that seem to play a major role in this experiment seem to be the quantitative values. Thus, I explored NPX and protein abundance first by creating a bar graph to see the distributions. These values did not have a normal distribution. NPX did not have a normal distribution because the range of its values was from 84 to 613851000. Furthermore, most values were below 100000000, with only a few values over 200000000. A similar pattern can be seen for peptide abundance values. Peptide abundance values had a minimum value of 10 and a maximum value of 178752000. These wide ranges indicate that it is most likely that a scaler will need to be used to avoid data leakage.

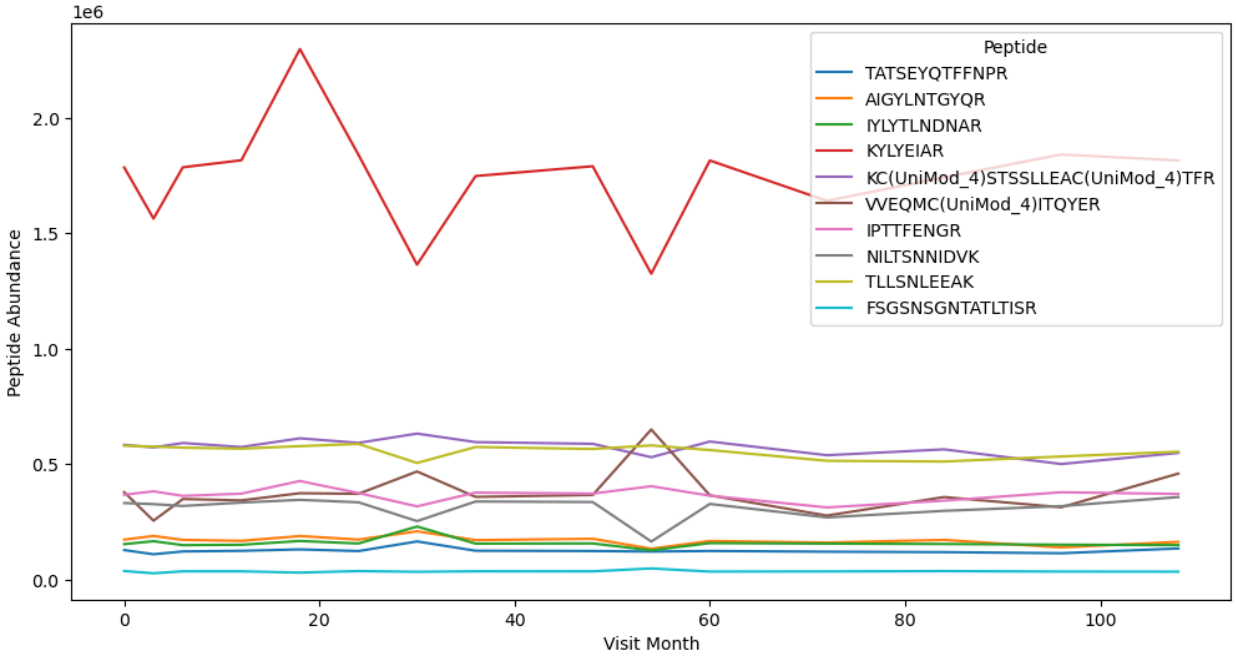
After analyzing the distribution of NPX and peptide abundance, I looked at the values for the UPDRs one through four and their progression through time by graphing them into a line graph. The values for the UPDRs did not seem to follow any trend. They randomly fluctuated up and down. Thus, I considered creating a filtered data frame that included only patients who were off medication and had no NaN values. By graphing the filtered data frame, I got relatively straight lines for the UPDRs one through four. Thus, medication did seem to have an effect on the UPDR scores.



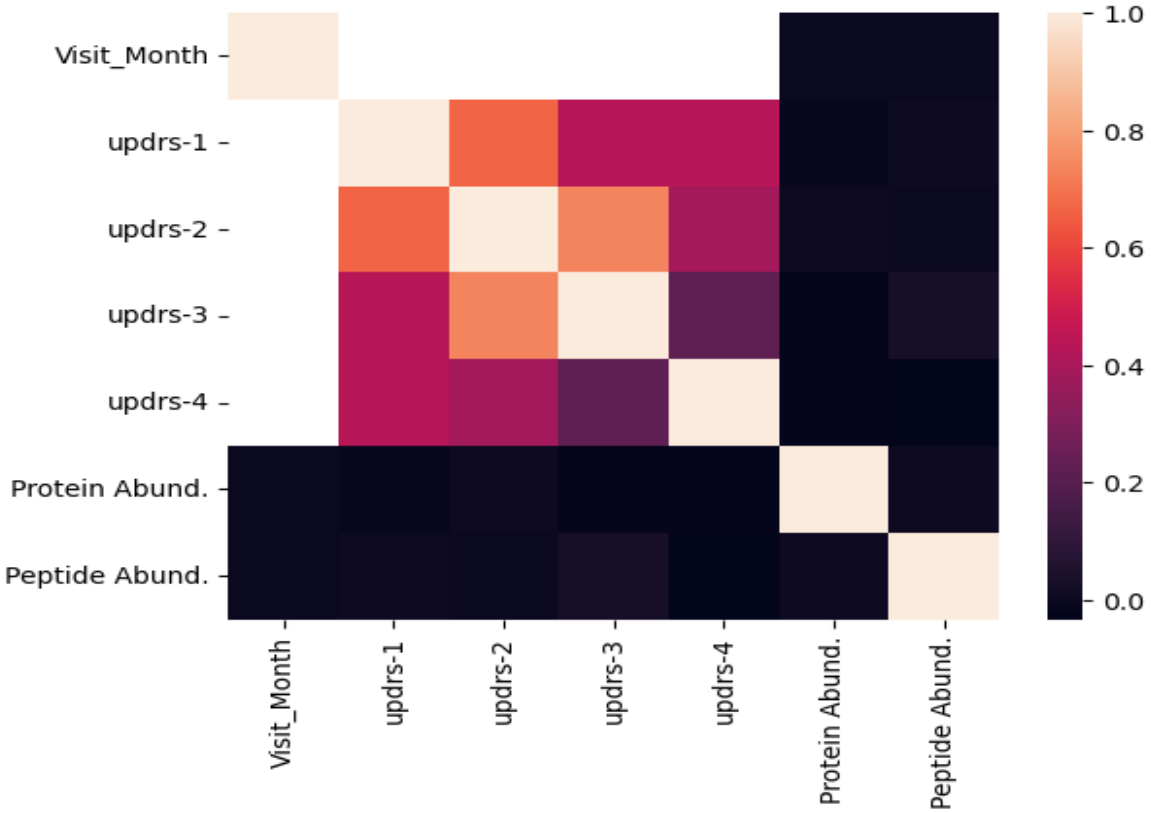


Proteins PO5090, PO2790, and PO1024 were the top three most abundant proteins in this data set. Over time, they did seem to fluctuate; however, apart from the fluctuation, there does not seem to be a trend in their change.

I also looked at the most common peptides and how they changed over time. Again, like the proteins, they had random fluctuations, but there was no trend.



Finally, looking at the correlation map, the results were unexpected. There was zero correlation among the main values of interest. The correlation map did not help this time.



3. Data Wrangling: Creating the Data Frame

Creating the data frame for this analysis consisted of two parts. Part one had three steps: grouping, pivoting, and merging data. The second part of data wrangling consisted of obtaining the features. Part one of data wrangling called for grouping features using group by. For the proteins data frame, before calling reset index, I temporarily made the visit_id represent the rows while making UniProt values into features arranged in columns. The mean NPX value for each UniProt became the value for each UniProt.

I repeated this for the peptides data frame and had the mean of the peptide abundance represent the values for the peptides organized in columns. Lastly, I merged these two data frames together and extracted the features to create a feature list.

4A. Methods Overview

To predict the UPDR score of a patient, I will design two models and modify them accordingly to try to obtain the best results. The two models of my choice include the Random Forest Regression model and the Gradient Boosting model, as these models are similar enough to perform the same supervised learning task. The only difference is that random forest regression creates slightly more complex trees, and it bags its trees instead of booting them like GBM does. For the sake of processing time, the Panda's data frame will be converted into a TensorFlow data frame and use the default settings for the Random Forest Regression. However, a scaler will be applied to the features since it was evident in the EDA that the data had an extensive range of values, especially for both NPX and protein abundance. Two different scalers will be used when building the model: the standard scaler and the robust scaler.

4B. Model Outline

For simplicity, most models will follow the same structure. Before initializing the models, a function for sMAPE will be defined. sMAPE is a metric used to measure a model's accuracy. Furthermore, a function to split the data randomly into 80 percent training data and 20 percent testing data will also be defined in place of the train_test_split function. Next, a cell containing three empty dictionaries will be made. The empty dictionaries are to store the model data, the MSE data, and the sMAPE data for easy printing when evaluating the model at the end.

Step 1: Initialize the Scaler and For Loop

Before beginning the for loop for the model, if a scaler is to be used, the scaler will always be initialized first. After initializing the scaler, a for loop is set up to run through each of the four UPDRs that are contained in a list defined as target.

Step 2: Merging Data

The first step in the for loop was to create a new data frame by merging the previously made data frame that had the protein and peptide data with the clinical data on the visit_id, while also including patient_id, visit_month, and labels. Labels represent the UPDRs one through four.

Step 3: Removing Null Values

The next step was to get rid of null values in the labels column and obtain a new features list that included the labels in it.

Step 4: Scaling Data & Returning to a Pandas Data Frame

After getting rid of the null values, the data was scaled using the scaler that was initialized at the beginning. However, when data is scaled, it is converted into an array, so the data needs to be converted back into a Pandas data frame so it can later be transformed into a TensorFlow data frame.

Step 5: Splitting Data

Using the scaled pandas data frame, split it into a training and testing data set using the previously defined function.

Step 6: Pandas Data Frame to Tensor Flow Data Frame

Successively, transform the split data from a Pandas data frame into a TensorFlow data frame, while identifying the task as a regression task.

Step7: Creating the Model

Finally, fit the new TensorFlow training and testing data frames to the TensorFlow Random Forest model while specifying the evaluation metric MSE. Save the results to the empty dictionary for the model made in the beginning.

Step8: Calling the MSE Evaluation Metric

Call the evaluation metric and store the results for each UPDR label into the empty MSE dictionary made in the beginning.

Step 9: Calling the sMAPE Function.

Call the sMAPE function that was defined in the beginning and store the results for each UPDRs in the sMAPE dictionary.

4. Results

This section will go over the differences in each model and the results for each model. The goal for both accuracy metrics is to obtain the smallest number possible.

Model 1: Random Forest Regression & Standard Scaler

Model 1 is a Random Forest Tree Regression model with a standard scaler in the beginning. It has an average MSE of 0.93. This value is higher than the one, so the model is not predictive. The average sMAPE value for model one was 144.02. The ideal sMAPE value is usually under 100; however, 144 is above 100. This model again does not seem to be predictive.

Model 2: Random Forest Regression &Robust Scaler

Model two is a Random Forest Tree Regression model with a Robust scaler in the beginning. The robust scaler helps mitigate the effects of the outliers. Model 2 performed much better with an average MSE score of 0.50. This model is on the border of being considered predictive. The sMAPE score however, was still over 100 at 140.36. Nonetheless 140 is better than model 1, which had an average sMAPE Score of 144.

Model 3: Gradient Boosting Model & No Scaler

Model 3 is a Gradient Boosting model without a scaler. Since the model was not scaled it had an MSE of 54.13. This model can be considered predictive since it is under 100. The average sMAPE score for model 3 was 97.91. It was the lowest sMAPE score Obtained throughout all four models. Yet it is still close to 100.

Model 4: Gradient Boosting Model & Robust scaler

Model 4 is a Gradient Boosting Model with a Robust scaler. This model took the longest time out of all the three models. It had an average MSE score of 0.533, and a sMAPE score of 133.07. The other models performed better than this model.

5. Conclusion

The best model is model 3 as it obtained the smallest sMAPE score, and the sMAPE score is the metric that is used to judge this Kaggle competition. Model two was a close second because it had the lowest MSE score at 0.50.

6. Future Considerations

Gradient boosting model did better than the Random Forest model when predicting UPDRS. In the future these models can be further improved by changing the parameters. Moreover, some of the models did better than other models at predicting different UPDRs, so a combination of models, also known as an ensemble method, can also be used to get better results. Finally, in the EDA portion of this analysis, medication did influence the results. Adding the medication status of a patient to the features

list of this data frame with the help of dummy regressors or one hot encoding might further improve the model results. This data can further be used to find the proteins, peptides, and genetic differences that might be involved in Parkinson's disease to help find new solutions, and medications to fight against Parkinson's.

AMP®-Parkinson‘s Disease Progression Prediction Capstone 2

Capstone 2: Maria Meza

Problem Identification

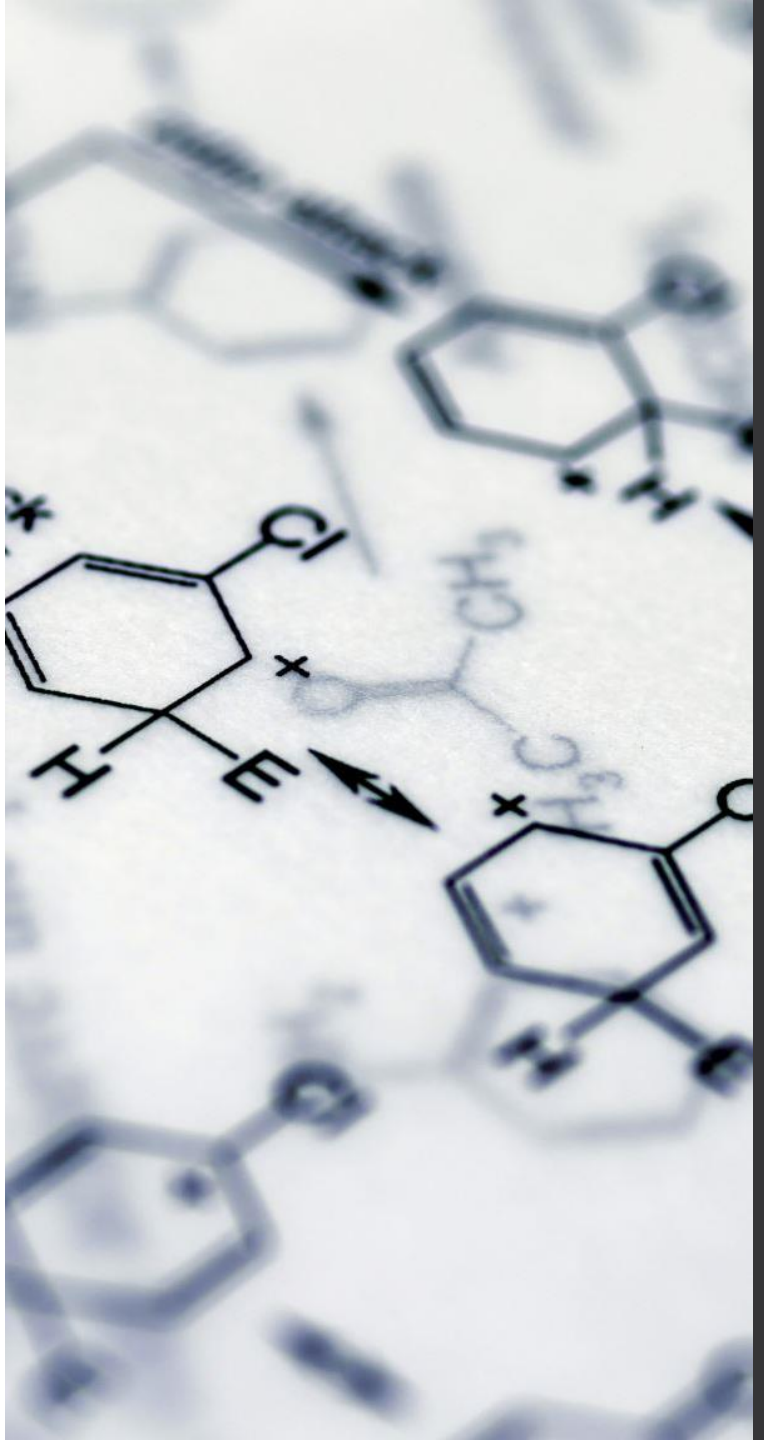
Make an appropriate model that best predicts the progression of Parkinson's disease by identifying the UPDR ranking

The best model was a **GBM without a robust scaler.**

Defining Variables

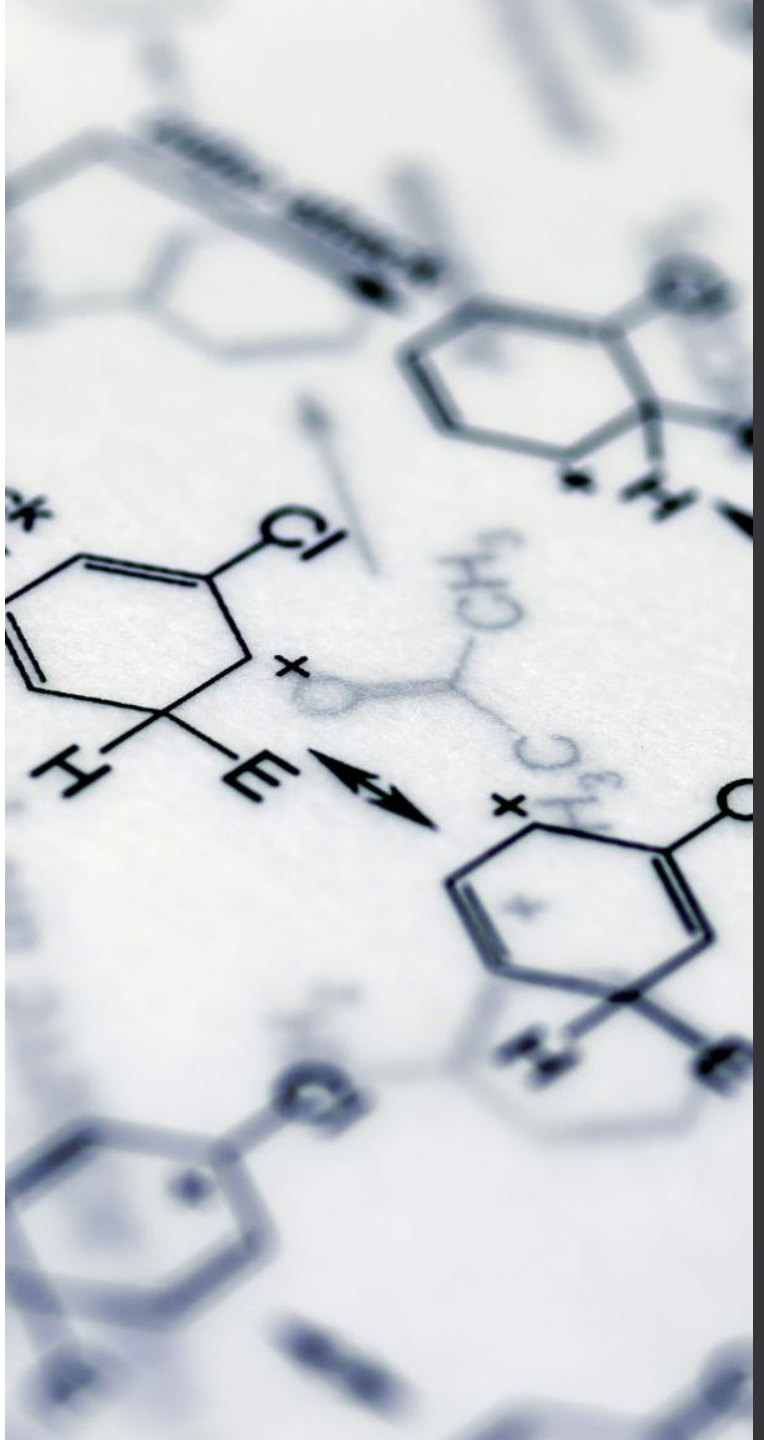
Defining Data

- NPX: Protein's occurrence in the sample. (AKA Protein Abundance).
- Peptide Abundance: How much of a peptide appeared in the sample.
  - Not to be confused with protein abundance!
- UniProt : Unique protein ID
- UPDRs1-4: The patient's score for the Unified Parkinson's Disease Rating Scale. Different levels cover different symptoms.
  - UPDR 1: Mood and behavior
  - UPDR2: Daily Motor Functions
  - UPDR3: Motor functions
  - UPDR4: Further Complications



Defining Data

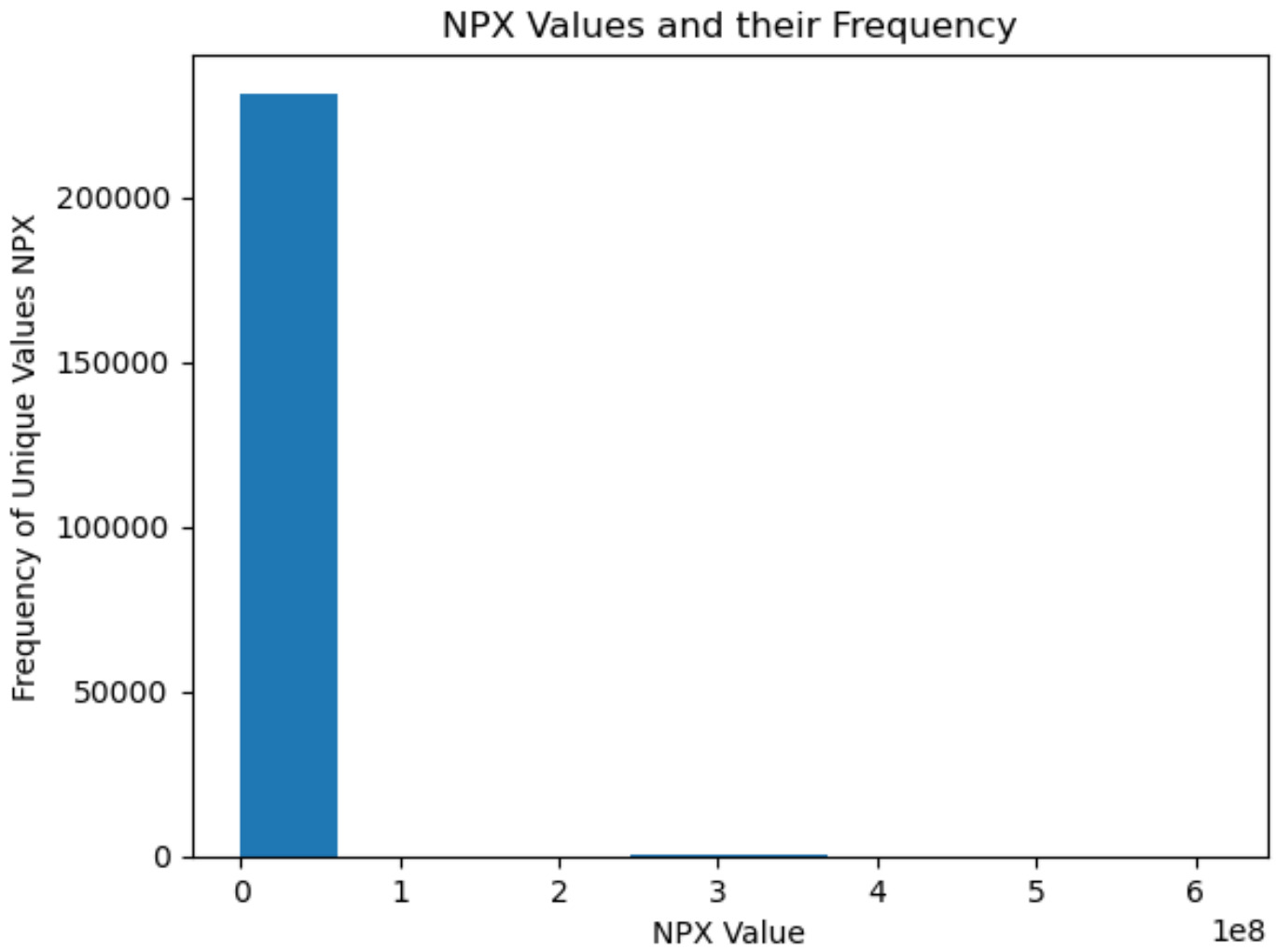
- Visit Id: The ID for the visit
- Visit Month: The months passed after each visit. Starts at zero.
- Patient Id : Unique ID given to each patient
- Clinical state on medication: Indicates whether the patient is on medication before being tested for Parkinson's



EDA

Quantitative Variables:

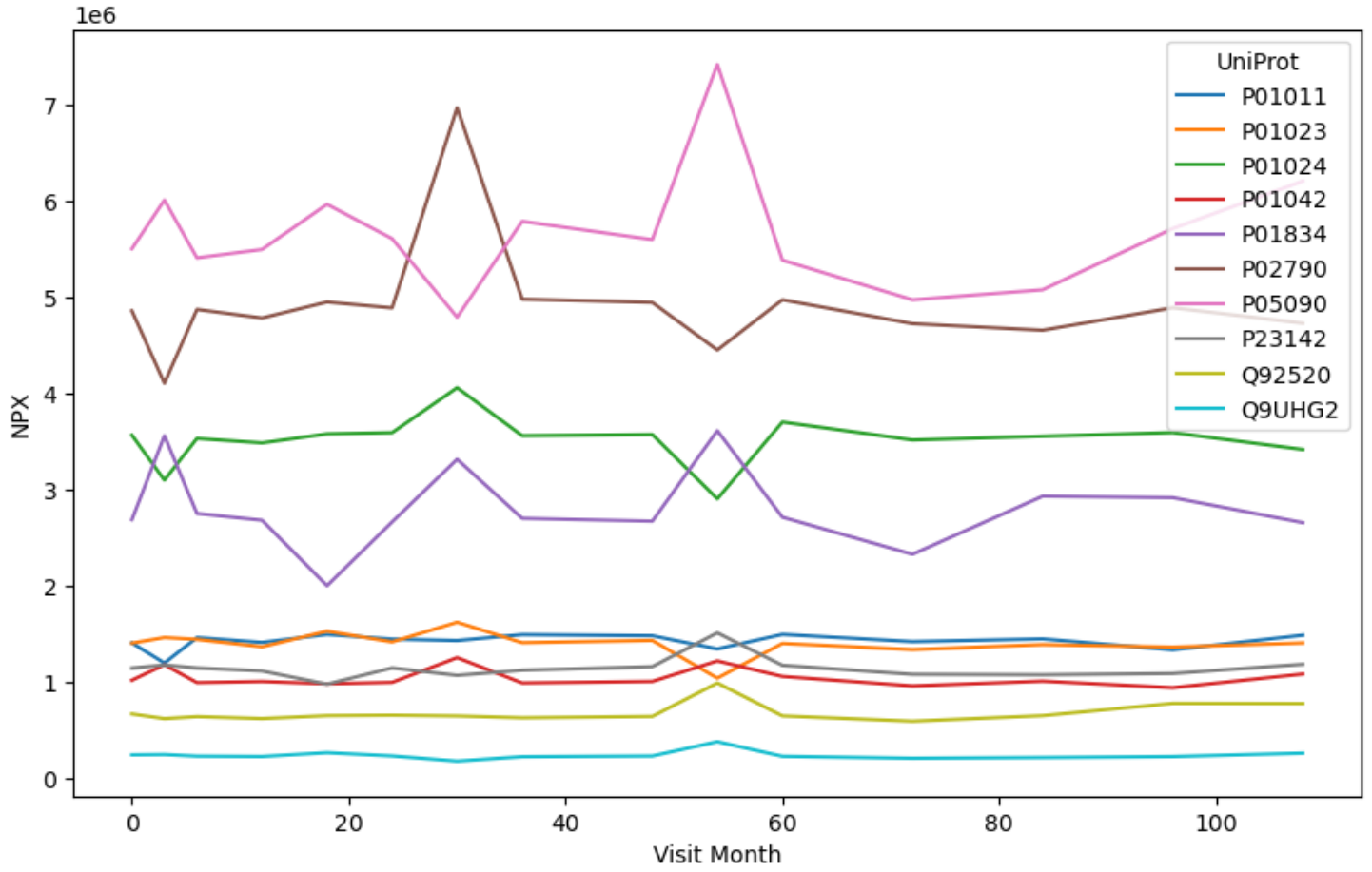
- NPX
- Protein Abundance
- UPDRs 1-4 __



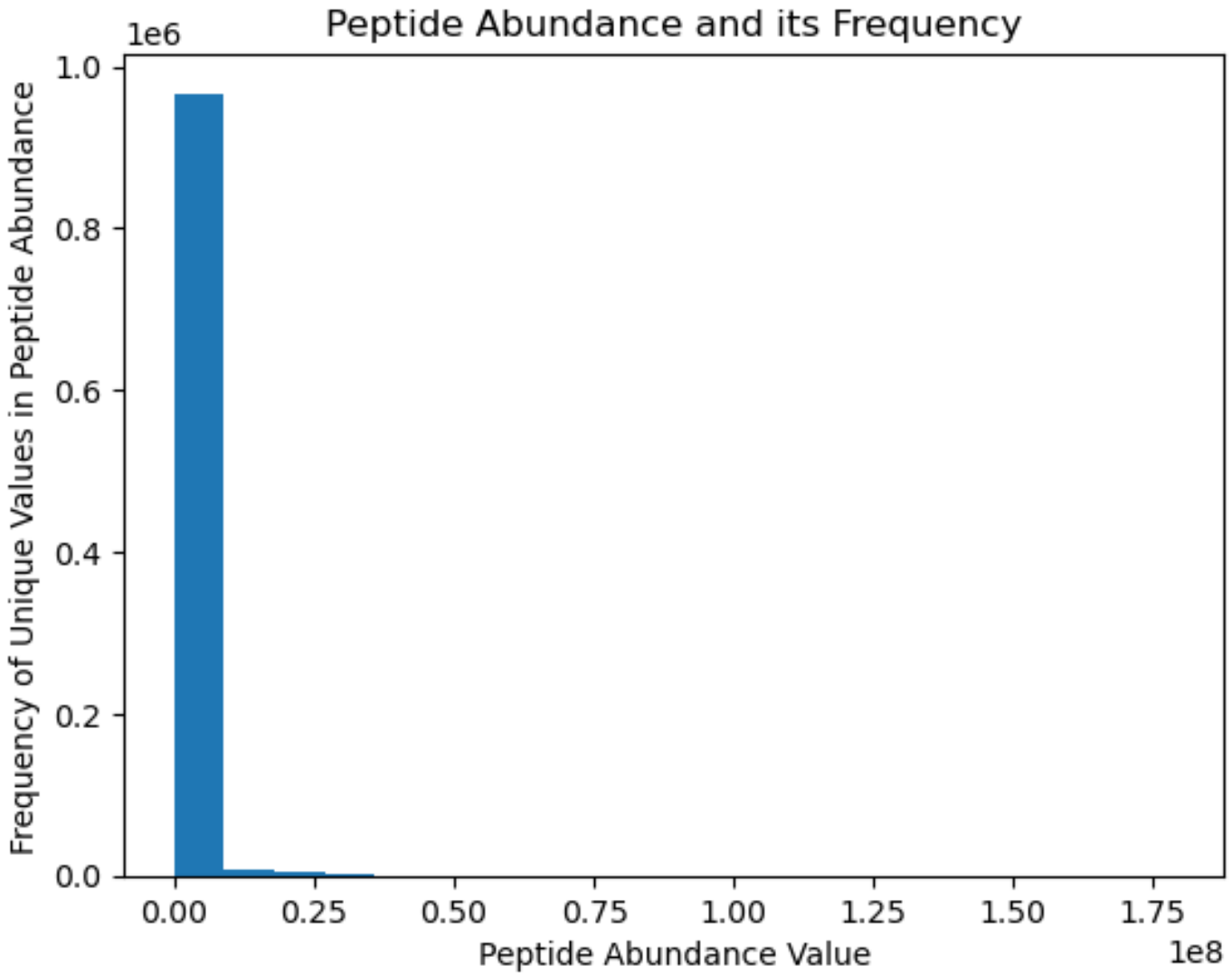
NPX

Data has a wide range of values

NPX vs Visit Month



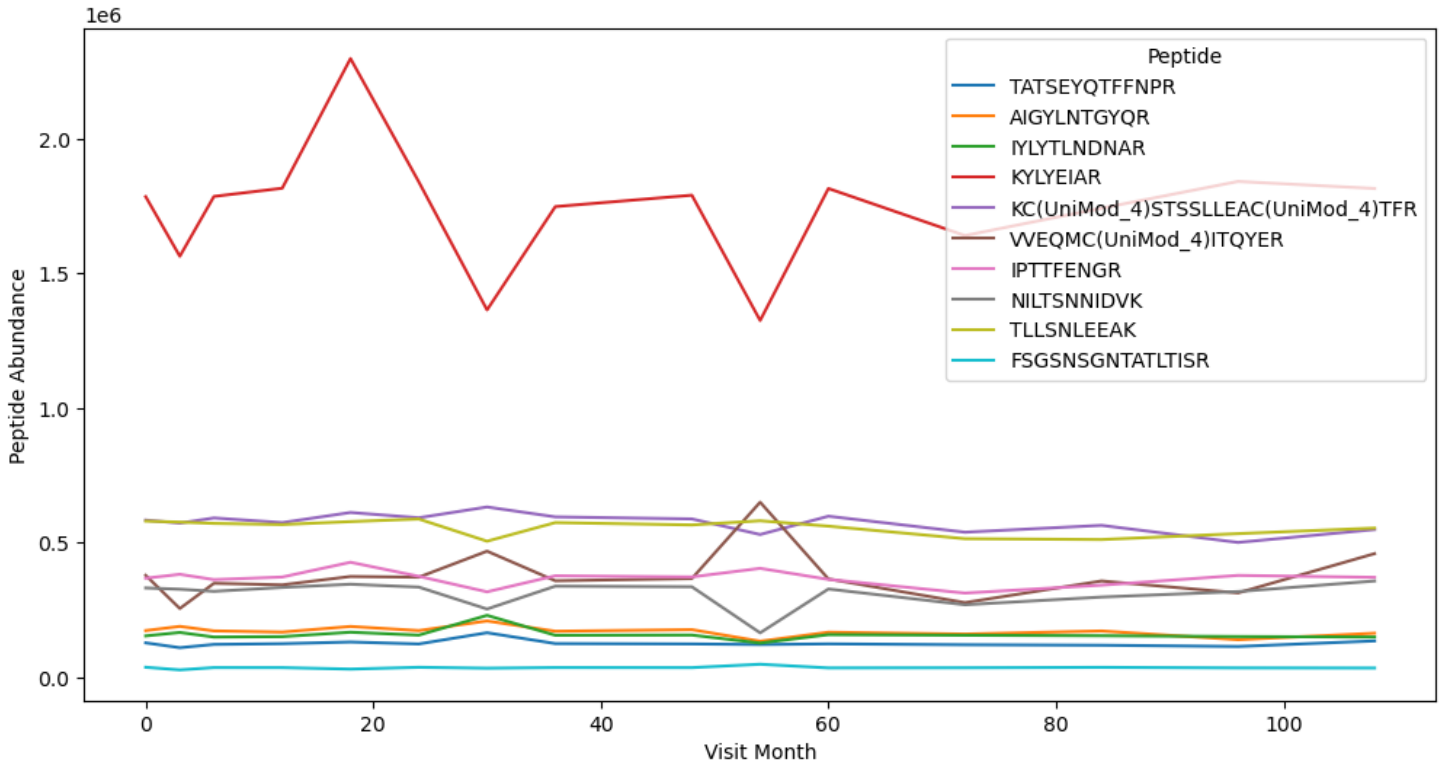
- The 3 most common proteins over the whole dataset
  - PO5090
  - PO2790
  - PO1024
- This Graph looks Similar!!!!
- (looks like UPDR Graph)



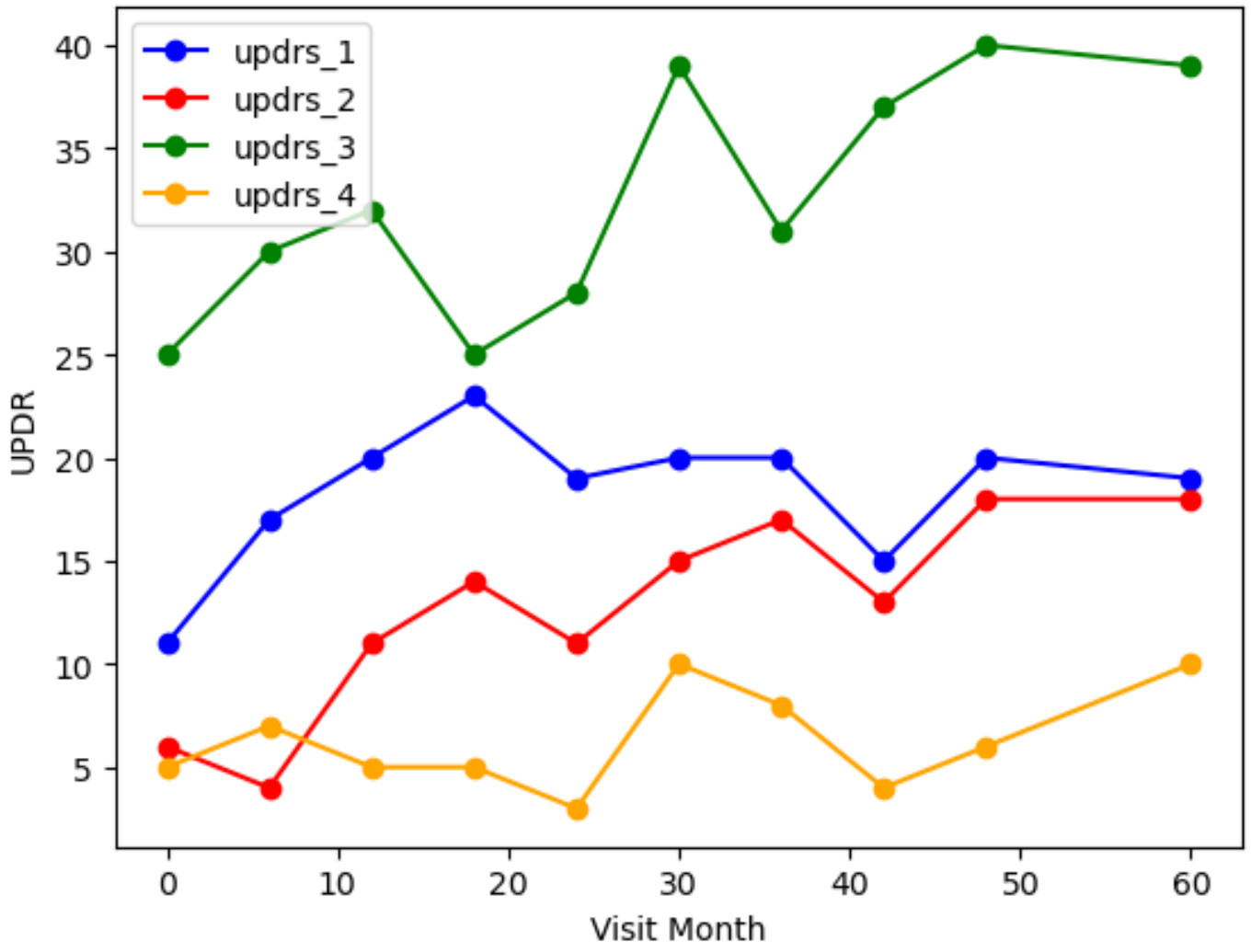
Peptide Abundance

Data has a wide range of values

Peptide vs Visit Month



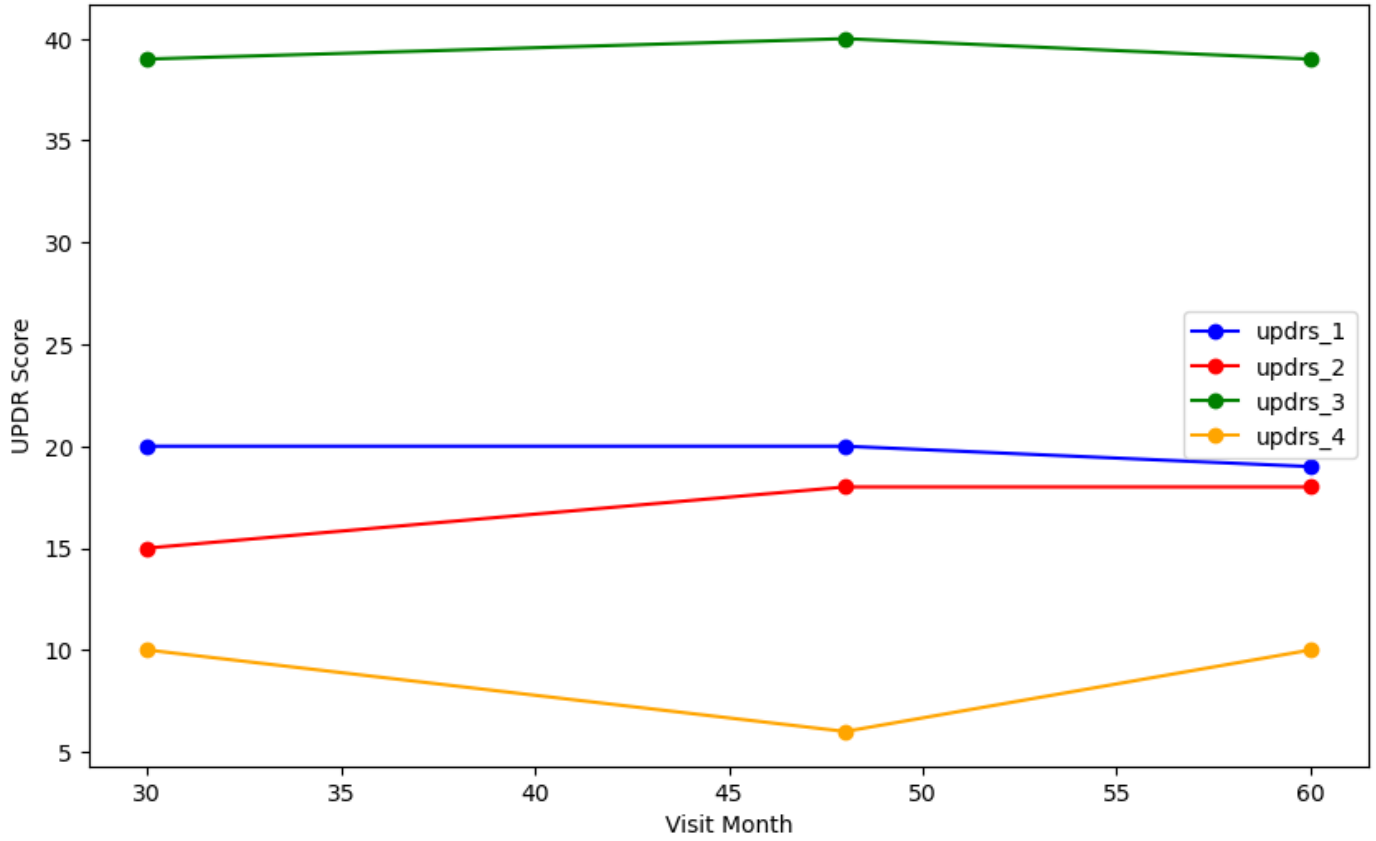
- The 3 most common peptides over the whole dataset
- KYLYEAIR
- KC
- TLLSNLEEAK



UPDRs1-4

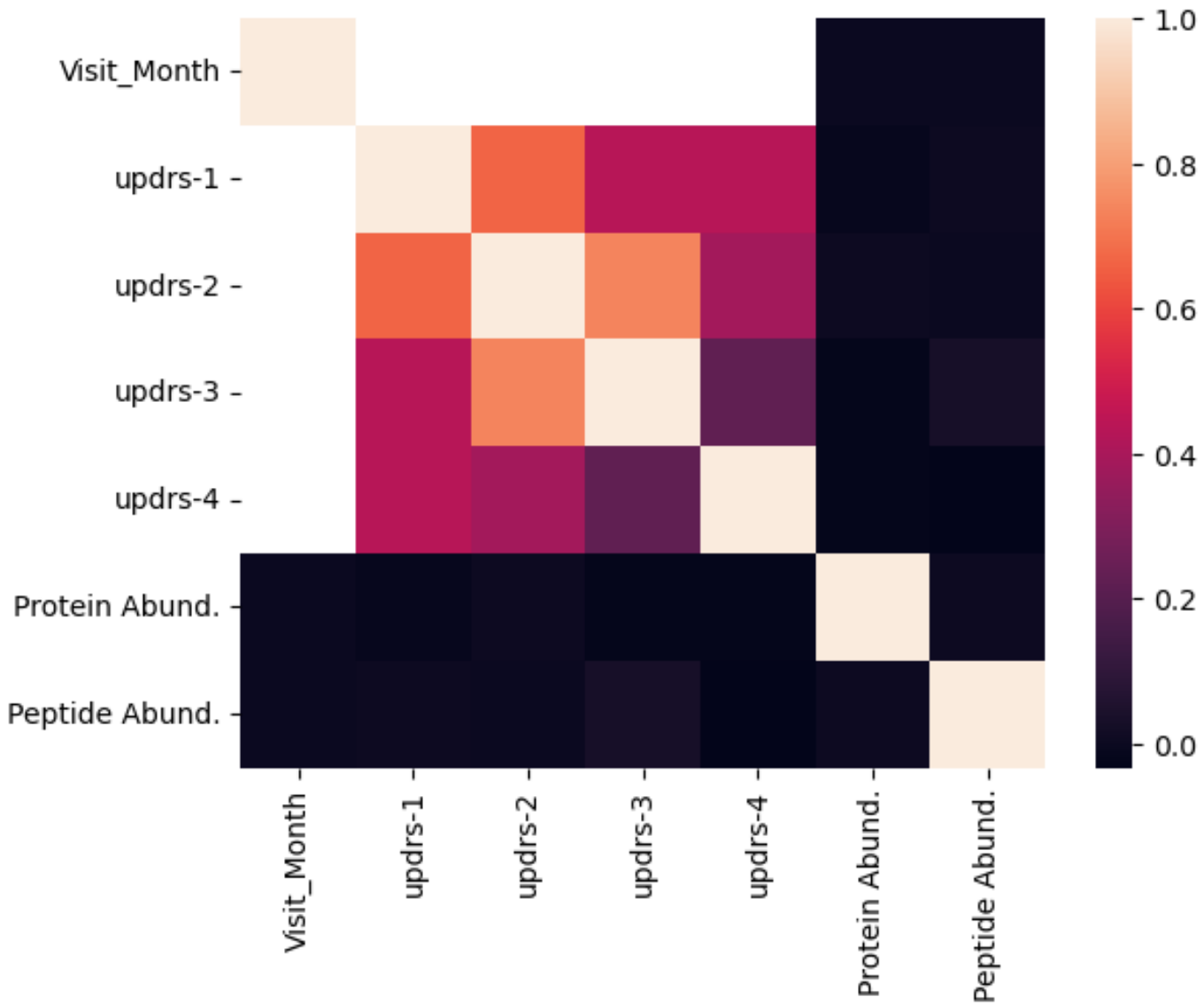
As time progresses there seems to be an increase UPDRs scores. There are some dips.

Could these be medicine related?



UPDRs1-4: Without Medication and NA values

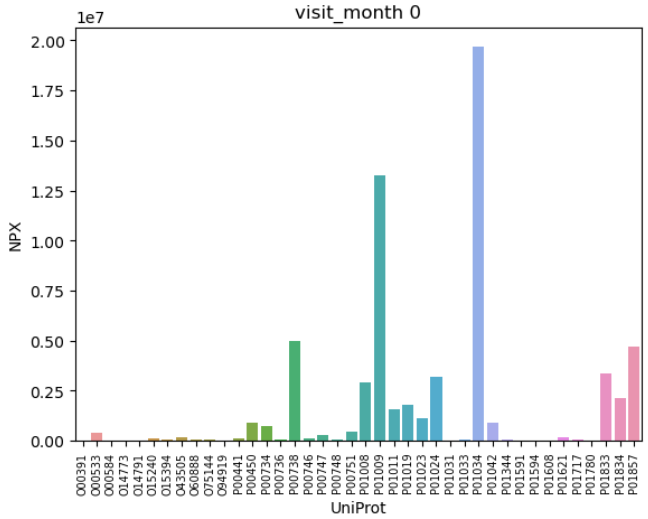
As expected it was medicine related! The UPDRs scores seem quite constant over time.

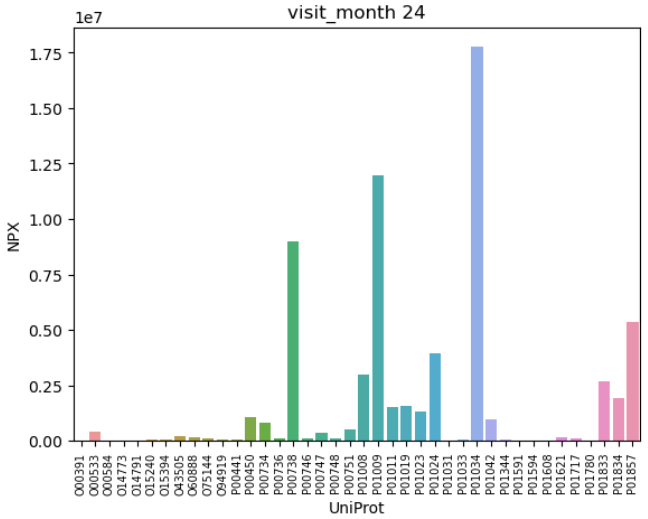


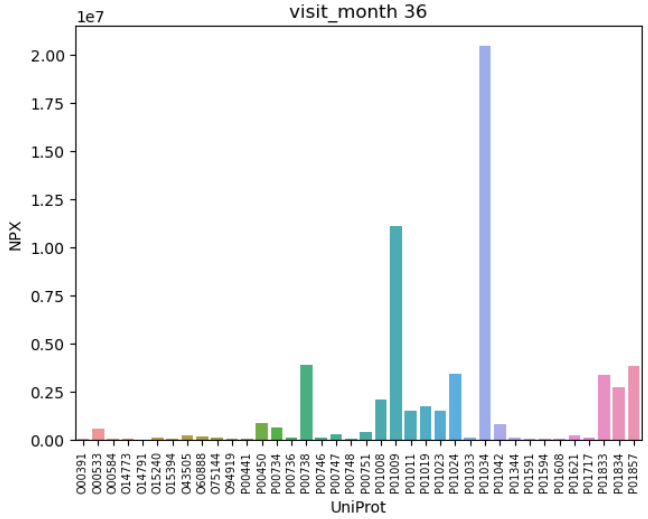
Correlation Matrix

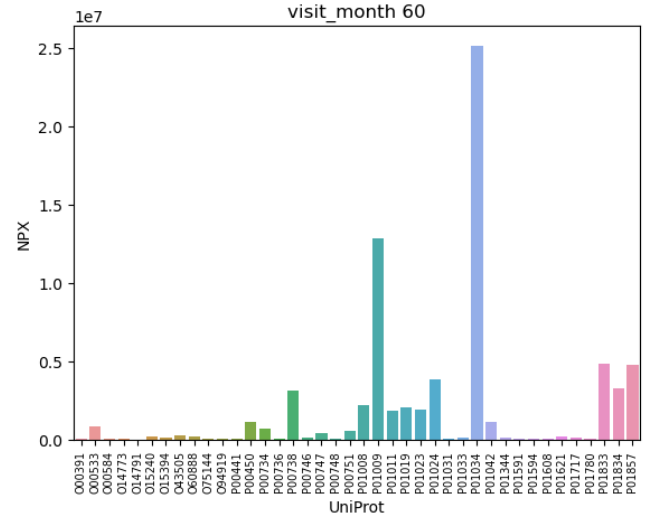
Unexpectedly there doesn't seem to be correlations among: the variables.

NPX (Protein Abundance vs Time for random patient)

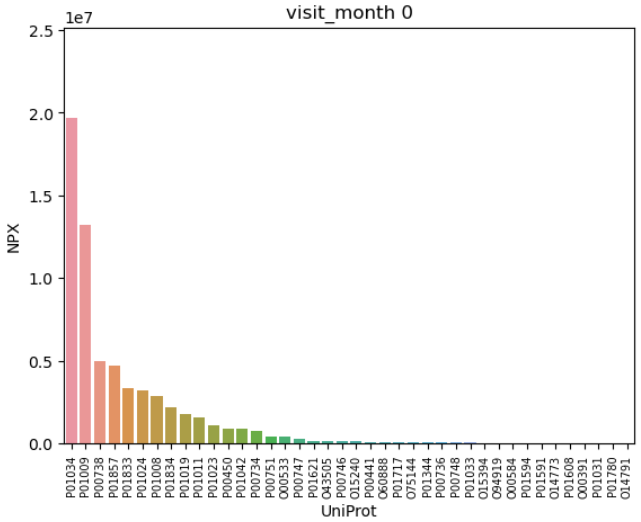


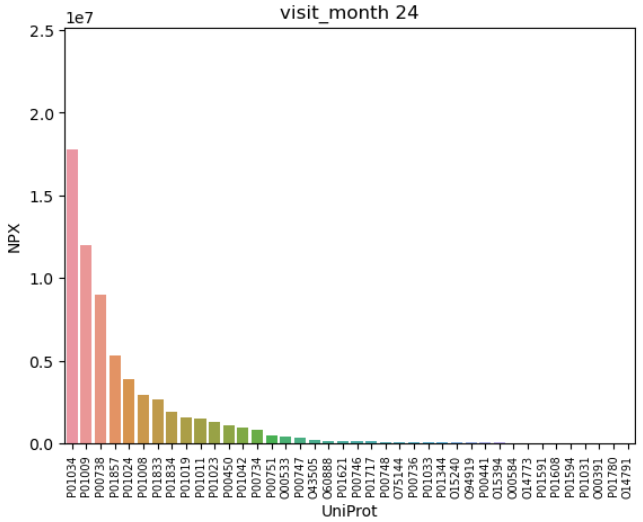


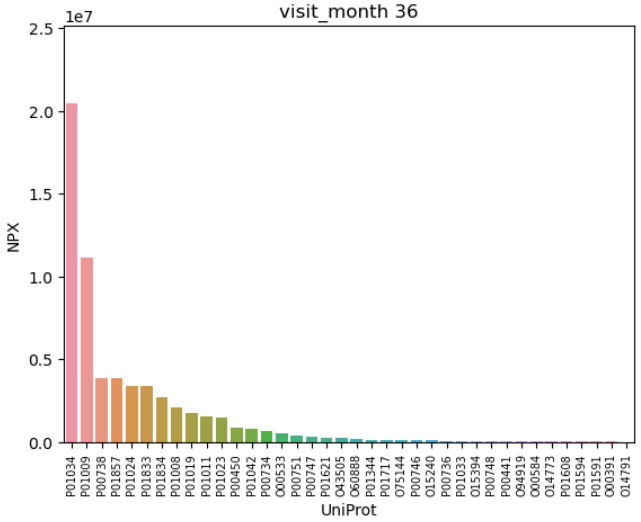


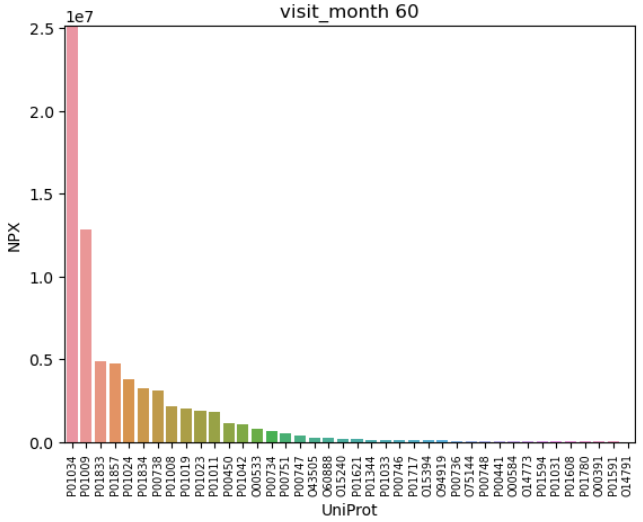


NPX (Protein Abundance vs Time for Random Patient:)

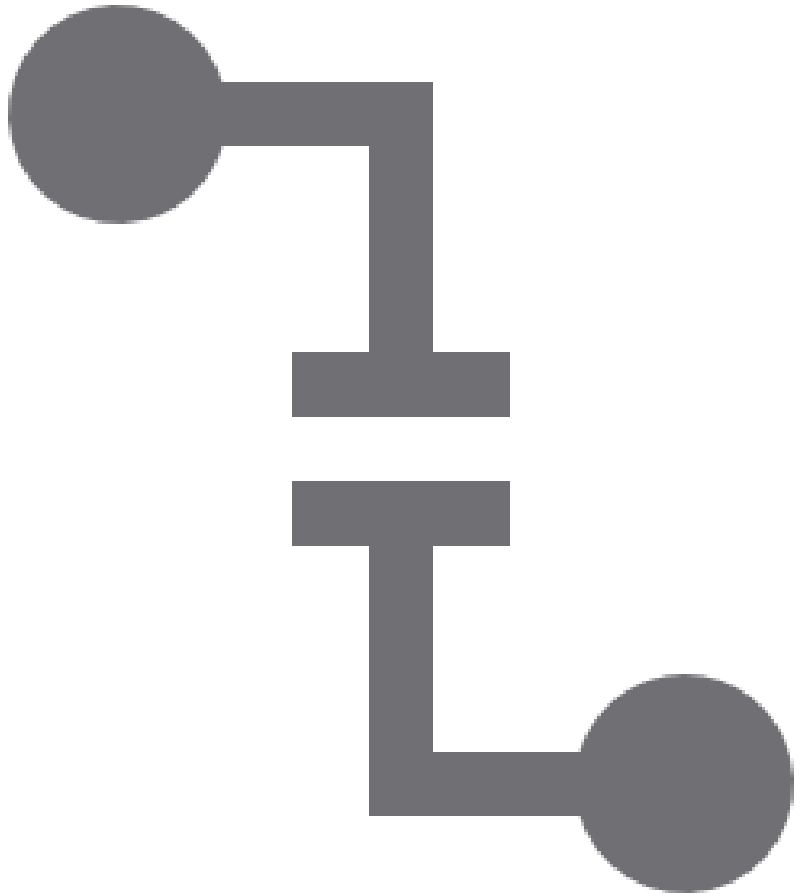








Data Wrangling



```
def prepare_dataset(train_proteins,
train_peptides):

Step 1: Grouping
    df_protein_grouped =
proteins.groupby(['visit_id','UniProt'])['NPX'].mea
n().reset_index()
    df_peptide_grouped =
peptides.groupby(['visit_id','Peptide'])['PeptideAbu
ndance'].mean().reset_index()

 Step 2: Pivoting
    df_protein =
df_protein_grouped.pivot(index='visit_id',columns
= 'UniProt', values =
'NPX').rename_axis(columns=None).reset_index()
    df_peptide =
df_peptide_grouped.pivot(index='visit_id',columns
= 'Peptide', values =
'PeptideAbundance').rename_axis(columns=None
).reset_index()


Step 3: Merging
    pro_pep_df = df_protein.merge(df_peptide, on =
['visit_id'], how = 'left')

    return pro_pep_df
```

GOAL 1 : Making a data frame with select features into columns!

Grouping

PREPARING THE DATA SET

Step 1: Grouping

```
df_protein_grouped = proteins.groupby(['visit_id','UniProt'])['NPX'].mean().reset_index()
df_peptide_grouped = peptides.groupby(['visit_id','Peptide'])['PeptideAbundance'].mean().reset_index()
```

- Obtain the desired columns from the desired data frames:
  - From proteins obtain visit_id, UniProt, and NPX
  - From peptides obtain visit_id, peptide, and PeptideAbundance
- Do not forget to reset index

Pivoting

PREPARING THE DATA SET

Step 2: Pivoting

```
df_protein = df_protein_grouped.pivot(index='visit_id',columns = 'UniProt', values = 'NPX').rename_axis(columns=None).reset_index()
df_peptide = df_peptide_grouped.pivot(index='visit_id',columns = 'Peptide', values = 'PeptideAbundance').rename_axis(columns=None).reset_index()
```

Make Uniprot and Peptide into features aligned as columns with the mean of NPX and mean of peptide abundance as the values respectively .

(Make Visit_Id the index to Merge on NEXT!)

Merging

PREPARING THE DATA SET

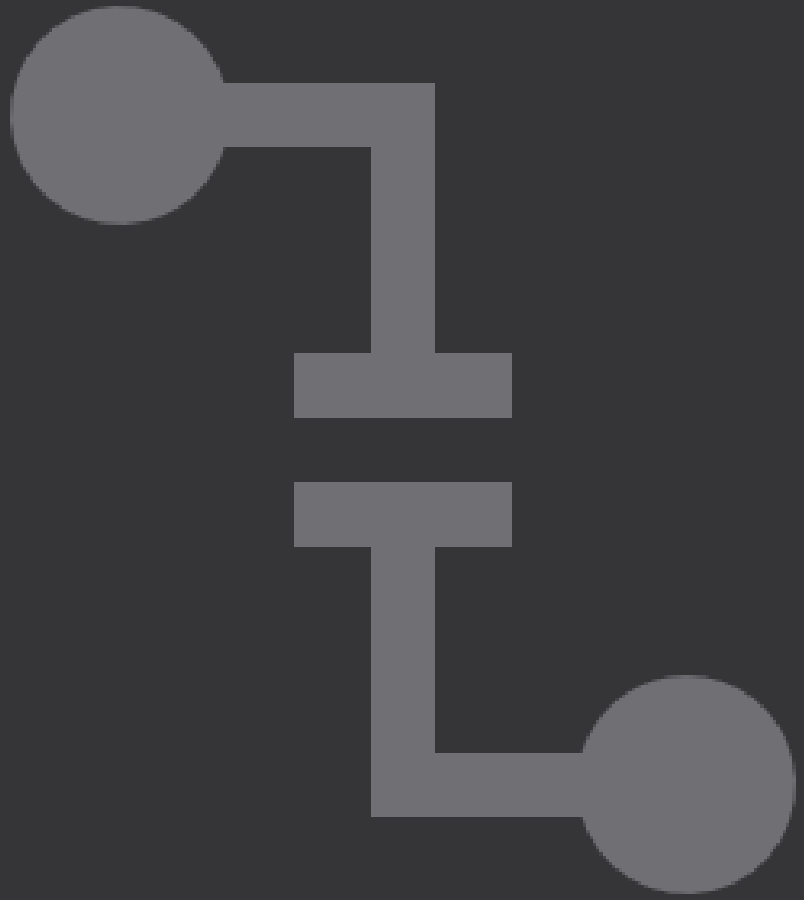
Step 3: Merging

```
pro_pep_df = df_protein.merge(df_peptide, on = ['visit_id'], how = 'left')

return pro_pep_df
```

Left merge the newly made df_protein to df_peptide on Visit_Id

Get your new **pro_pep_df**



```
def prepare_dataset(train_proteins,
train_peptides):

Step 1: Grouping
    df_protein_grouped =
proteins.groupby(['visit_id','UniProt'])['NPX'].mea
n().reset_index()
    df_peptide_grouped =
peptides.groupby(['visit_id','Peptide'])['PeptideAbu
ndance'].mean().reset_index()

 Step 2: Pivoting
    df_protein =
df_protein_grouped.pivot(index='visit_id',columns
= 'UniProt', values =
'NPX').rename_axis(columns=None).reset_index()
    df_peptide =
df_peptide_grouped.pivot(index='visit_id',columns
= 'Peptide', values =
'PeptideAbundance').rename_axis(columns=None
).reset_index()


Step 3: Merging
    pro_pep_df = df_protein.merge(df_peptide, on =
['visit_id'], how = 'left')

    return pro_pep_df
```

GOAL 2 : Obtaining Features

Obtaining Labels

```
# Creating Features List
FEATURES = [i for i in pro_pep_df.columns if i not in ["visit_id"]]
FEATURES.append("visit_month")
print(FEATURES)
```

- Make a For Loop iterating over the newly made data frame.
- Obtain all the features EXCEPT visit_id from the new data frame

Initiating Empty Dictionaries

```
#Store the trained model
model_dict = {}

# Create an empty dictionary to store the mse score of the models trained for each label.
mse_dict = {}

# Create an empty dictionary to store the sMAPE scores of the models trained for each label.
smape_dict = {}
```

An empty dictionary to:

- Store the model.
- Store MSE score
- Store sMAPE score

Preparing the Model & Modeling

Note: Modeling is done with help from TensorFlow to speed up the process

Preparing the General Model : Model 1

- Create a loop to :

Part1

- 1. Define a Function to Split Data
- 2. merge
- 3. drop null values
- 4. Scale data

Part 2

- 4. split and train the data set
- 5. Convert the dataset pandas-> tensorflow for faster processing

```
def split_dataset(dataset, test_ratio=0.20):
  test_indices = np.random.rand(len(dataset)) < test_ratio
  return dataset[~test_indices], dataset[test_indices]
```

- Instead of using train_test_split I defined a function called split_dataset to split the dataset into two subsets by using random sampling:
  - A training set and a testing.
- 80% of the data will be be in the training
- 20% of the data will be in the testing set.

Defining a Function to Split Data

Initializing the Scaler

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
```

```
scaler = StandardScaler()
```

Initializing Target Labels

```
# List of target labels to loop through and train models
target = ["updrs_1", "updrs_2", "updrs_3", "updrs_4"]
```

Initilizing Model through For Loop

```
scaler = StandardScaler()

# List of target labels to loop through and train models
target = ["updrs_1", "updrs_2", "updrs_3", "updrs_4"]

# Loop through each label
for label in target:

    # Merge the label 'visit_id', 'patient_id', 'visit_month' and label columns from `train_clinical`
    # data frame to `pro_prep_df` data frame on the `visit_id` column.
    dataset_df = pro_pep_df.merge(clinical[['visit_id', 'patient_id', 'visit_month', label]], on=['visit_id'], how='left')
```

- Initiate a for loop to go through each target UPDRs (The target variables)
- Merge the data frame you made recently to the clinical data frame on visit_id .
- Make sure you add new columns: patient_id, visit_month, label.
  - Label refers to the target variables you are trying to predict ('updrs_1', 'updrs_2', etc.).

Removing NA Values

```
# Drop null value label rows
dataset_df = dataset_df.dropna(subset=[label])
```

Missing Values will be dropped from label (UPDRs1-4)

Add Label to Features

```
# Make a new copy of the FEATURES list we created previously. Add `label` to it.
feature_list = FEATURES.copy()
feature_list.append(label)
```

- New list of Features, to include label values

Data Scaling

Scaling: Standard Scaler

```
# Scale the features in the Pandas DataFrame
train_scaled = scaler.fit_transform(dataset_df[feature_list].values)

# Convert the scaled DataFrame back to a Pandas DataFrame
train_scaled_df = pd.DataFrame(train_scaled, columns=feature_list)
```

- Scaling the data turns the data into a numpy array
- Must turn df back into pandas df

Data Spliting and Training

Splitting into Test/ Train Data Set

```
# Split the dataset into train and validation datasets.
train_df, valid_df = split_dataset(train_scaled_df)
print("{} examples in training, {} examples in testing.".format(len(train_df), len(valid_df)))
```

Using the function defined earlier to split the data into 80%training data and 20% test data

Print the number of examples (rows) in the training and validation datasets

Convert into TensorFlow Data Set

```
# Convert the Pandas DataFrame to TensorFlow datasets
train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(train_df, label=label, task=tfdf.keras.Task.REGRESSION)
valid_ds = tfdf.keras.pd_dataframe_to_tf_dataset(valid_df, label=label, task=tfdf.keras.Task.REGRESSION)
```

- tfdf.keras.Task.REGRESSION: Building a regression model
- tffdf.keras.pd_dataframe_to_tf_dataset: Converts pd data frame into a tensor flow object
  - Data is handled in batches.
  - Data is shuffled
  - Label is target variable

Creating Model 1: Random Forest With Standard Scaler

Creating Model 1

```
# Create and train the Random Forest model
rf = tfdf.keras.RandomForestModel(task=tfdf.keras.Task.REGRESSION, verbose=0)
rf.compile(metrics=["mse"])
rf.fit(x=train_ds)

# Store the trained model
model_dict[label] = rf
```

- Define the random forest model for the tensor flow data frame
- Specify MSE as the metric to be used for evaluating
- Fit the model
- Store the model

Evaluating Models

- MSE: Used to evaluate the accuracy of regression models.
- sMAPE :Used to evaluate the accuracy of regression models. It measures the percentage difference between actual and predicted values. The evaluation metric for this Kaggle competition

Evaluating Model 1

Evaluating Model 1:MSE

```
# Evaluate the trained model on the validation dataset and store the
# mse value in the `mse_dict`.
inspector = rf.make_inspector()
inspector.evaluation()
evaluation = rf.evaluate(x=valid_ds, return_dict=True)
mse_dict[label] = evaluation["mse"]
```

- Call evaluation metric data
  - Evaluate MSE & Print
- Store data in empty dictionary

Evaluating Model 1: MSE Results

MSE

In [98]:
```
for name, value in mse_dict.items():
  print(f"label {name}: mse {value:.4f}")

print("\nAverage mse", sum(mse_dict.values())/4)
```

```
label updrs_1: mse 0.8851
label updrs_2: mse 0.6803
label updrs_3: mse 0.7800
label updrs_4: mse 1.3786

Average mse 0.9309805631637573
```

- Run a for loop to print MSE values stored for Model 1
  - AVG MSE was 0.93

This is TOO close to 1.
The model is USELESS.

Evaluating Model 1:sMAPE Results

```
# Evaluate the trained model on the validation dataset and store the
# mse value in the `mse_dict`.
inspector = rf.make_inspector()
inspector.evaluation()
evaluation = rf.evaluate(x=valid_ds, return_dict=True)
mse_dict[label] = evaluation["mse"]
```

- Call evaluation metric data
  - Evaluate MSE & Print
- Store data in empty dictionary

Evaluating Model 1: sMAPE

```
for name, value in smape_dict.items():
  print(f"label {name}: sMAPE {value:.4f}")

print("\nAverage sMAPE", sum(smape_dict.values())/4)
```

```
label updrs_1: sMAPE 144.8859
label updrs_2: sMAPE 134.3554
label updrs_3: sMAPE 138.6299
label updrs_4: sMAPE 158.2205

Average sMAPE 144.02936753103 4
```

- Run a for loop to print sMAPE values stored for Model 1
- AVG sMAPE was 144.02

This is OVER 100.
The model is USELESS.

Model 2: Random Forest with RobustScaler

Model 2: Random Forest with RobustScaler

Used a robust scale instead of a standard scaler!Because my data is not originally normaly distributed.

```
30]: #empty lists 2
     model_dict2 = {}

     # Create an empty dictionary to store the mse score of the models trained for each label.
     mse_dict2 = {}

     # Create an empty dictionary to store the sMAPE scores of the models trained for each label.
     smape_dict2 = {}
```

```
31]: scaler = RobustScaler()
```

```
# Scale the features in the Pandas DataFrame using RobustScaler
train_scaled = scaler.fit_transform(dataset_df[feature_list].values)

# Convert the scaled DataFrame back to a Pandas DataFrame
train_scaled_df = pd.DataFrame(train_scaled, columns=feature_list)
```

Follow all steps for Model 1:

Except:

- Initialize new empty dictionaries for model 2
- Use Robust Scaler instead of the StandardScaler

Evaluating Model 2: MSE Results

MSE model2

```
for name, value in mse_dict2.items():
  print(f"label {name}: mse {value:.4f}")

print("\nAverage mse", sum(mse_dict2.values())/4)
```

```
label updrs_1: mse 0.5327
label updrs_2: mse 0.4218
label updrs_3: mse 0.2179
label updrs_4: mse 0.8448

Average mse 0.5042909383773804
```

- Run a for loop to print MSE values stored for Model 2
  - AVG MSE was 0.504

The MSE is 0.504.
The model may be considered predictive.

Evaluating Model 2: sMAPE

sMAPE model2

```
04]: for name, value in smape_dict2.items():
       print(f"label {name}: sMAPE {value:.4f}")

     print("\nAverage sMAPE", sum(smape_dict2.values())/4)
```

```
label updrs_1: sMAPE 147.1658
label updrs_2: sMAPE 134.7025
label updrs_3: sMAPE 134.2892
label updrs_4: sMAPE 145.2980

Average sMAPE 140.36386623828125
```

- Run a for loop to print sMAPE values stored for Model 2
- AVG sMAPE was 140.36

This is OVER 100, but better than 144

Model 3: Gradient Boosting Model

No Scaler

Model 3: Gradient Boosting Model (No Scaler)

```
05]: #Initiate Empty Lists
     #empty lists 2
     model_dict3 = {}

     # Create an empty dictionary to store the mse score of the models trained for each label.
     mse_dict3 = {}

     # Create an empty dictionary to store the sMAPE scores of the models trained for each label.
     smape_dict3 = {}
```

```
# Create and train the Gradient Boosted Trees (GBT) model
gbt = tfdf.keras.GradientBoostedTreesModel(task=tfdf.keras.Task.REGRESSION, verbose=0)
gbt.compile(metrics=["mse"])
gbt.fit(x=train_ds)

# Store the trained model
model_dict3[label] = gbt
```

Follow all steps for Model 1:

Except:

- Initialize new empty dictionaries for model 3
- Remove the scaler
- Run GBM instead of Random Forest

Evaluating Model 3: MSE Results

MSE for model 3

```
for name, value in mse_dict3.items():
  print(f"label {name}: mse {value:.4f}")

print("\nAverage mse", sum(mse_dict3.values())/4)
```

```
label updrs_1: mse 19.5055
label updrs_2: mse 23.2544
label updrs_3: mse 163.2188
label updrs_4: mse 10.5486

Average mse 54.131818532943726
```

- Run a for loop to print MSE values stored for Model 3
  - AVG MSE was 54.13

The MSE is 54.13. Since the model was NOT scaled, the model is on another scale. The model may be considered predictive.

Evaluating Model 3: sMAPE

sMAPE for model 3

```
for name, value in smape_dict3.items():
  print(f"label {name}: sMAPE {value:.4f}")

print("\nAverage sMAPE", sum(smape_dict3.values())/4)
```

```
label updrs_1: sMAPE 67.5822
label updrs_2: sMAPE 88.6727
label updrs_3: sMAPE 84.0679
label updrs_4: sMAPE 151.3394

Average sMAPE 97.91554592716952
```

- Run a for loop to print sMAPE values stored for Model 3
- AVG sMAPE was 97.91

This is UNDER 100
Best score yet.

Model 4: Gradient Boosting Model

with Robust Scaler

Model : Gradient Boosting Model (w RobustScaler)

In [111]:
```
#Initiate Empty Lists
#empty lists 2
model_dict4 = {}

# Create an empty dictionary to store the mse score of the models trained for each label.
mse_dict4 = {}

# Create an empty dictionary to store the sMAPE scores of the models trained for each label.
smape_dict4 = {}
```

In [112]:
```
scaler = RobustScaler()
```

```
# Create and train the Gradient Boosted Trees (GBT) model
gbt = tfdf.keras.GradientBoostedTreesModel(task=tfdf.keras.Task.REGRESSION, verbose=0)
gbt.compile(metrics=["mse"])
gbt.fit(x=train_ds)

# Store the trained model
model_dict4[label] = gbt
```

Follow all steps for Model 3:

Except:

- Initialize new empty dictionaries for model 3
- Add RobustScaler

Evaluating Model 4: MSE Results

MSE for Model 4

```
for name, value in mse_dict4.items():
  print(f"label {name}: mse {value:.4f}")

print("\nAverage mse", sum(mse_dict4.values())/4)
```

```
label updrs_1: mse 0.4303
label updrs_2: mse 0.3704
label updrs_3: mse 0.2597
label updrs_4: mse 1.0756

Average mse 0.5339668914675713
```

- Run a for loop to print MSE values stored for Model 4
  - AVG MSE was 0.5333

The MSE is 0.5333. The model may be considered predictive.
Second Best MSE score.
Note: This model took the longest to run!

Evaluating Model 4: sMAPE

sMAPE for Model 4

```
4]: for name, value in smape_dict4.items():
      print(f"label {name}: sMAPE {value:.4f}")

    print("\nAverage sMAPE", sum(smape_dict4.values())/4)
```

```
label updrs_1: sMAPE 132.1189
label updrs_2: sMAPE 121.8630
label updrs_3: sMAPE 124.0660
label updrs_4: sMAPE 154.2333

Average sMAPE 133.07029477951534
```

- Run a for loop to print sMAPE values stored for Model 4
- AVG sMAPEwas 133.07

This is OVER 100
Best score yet.

Comparing Models

Model 3 VS Model 2

sMAPE for model 3

```
for name, value in smape_dict3.items():
  print(f"label {name}: sMAPE {value:.4f}")

print("\nAverage sMAPE", sum(smape_dict3.values())/4)
```

```
label updrs_1: sMAPE 67.5822
label updrs_2: sMAPE 88.6727
label updrs_3: sMAPE 84.0679
label updrs_4: sMAPE 151.3394

Average sMAPE 97.91554592716952
```

MSE model2

```
for name, value in mse_dict2.items():
  print(f"label {name}: mse {value:.4f}")

print("\nAverage mse", sum(mse_dict2.values())/4)
```

```
label updrs_1: mse 0.5327
label updrs_2: mse 0.4218
label updrs_3: mse 0.2179
label updrs_4: mse 0.8448

Average mse 0.5042909383773804
```

- AVG sMAPE for model 3 was 97.91
- AVG MSE for model 2 was 0.50

Conclusion

Conclusion

The best model was a GBM without a robust scaler.

It had the lowest sMAPE Score.

(The metric for this competition)

End

Label	MSE	SMAPE
updrs_1	0.964576	144.9921
updrs_2	0.763217	138.4744
updrs_3	0.833526	140.0824
updrs_4	0.855988	156.3187

Model1

Label	MSE	SMAPE
updrs_1	0.571232	139.5744
updrs_2	0.421478	138.9512
updrs_3	0.25925	136.3823
updrs_4	1.194462	147.5631

model 2

Label	MSE	SMAPE
updrs_1	24.4994	62.35727
updrs_2	26.01915	85.25771
updrs_3	143.5099	71.68887
updrs_4	7.567027	157.0834

model 3

Label	MSE	SMAPE
updrs_1	0.462157	129.0648
updrs_2	0.432713	124.0622
updrs_3	0.244796	128.507
updrs_4	1.051201	159.1748

Model 4