Parkinson's Disease Progression Prediction

Presented by:
Jamilya Seisembekova
Rim Boughanmi
Rayan Karout
Juan Useche

What is Parkinson disease?

- **Definition:**

Parkinson's disease (PD), is a chronic degenerative disorder of the central nervous system that affects both the motor system and non-motor systems.

- **Cause:**

Parkinson's is primarily caused by the loss of dopamine-producing cells in the brain.
Note that the exact cause is not fully understood, and it may involve a combination of genetic and environmental factors.

- **Treatment:**

Highlight that while there is no cure, treatments can help manage symptoms and improve quality of life.

Goal of the Competition

AMP PD is a joint initiative by the government, industry, and nonprofits, managed by the Foundation of the National Institutes of Health. They have created the AMP PD Knowledge Platform to study Parkinson's disease extensively, seeking biomarkers for diagnosis, prognosis, and disease progression.

Specifications

Predict UPDRS score for patients which measure progression in patients with Parkinson's disease

sMAPE

Dispersion measure required by the competition

File exploration

Train Clinical data

Shape of the dataset:

```
: train_clinical_data.shape
(2615, 9)
```

Head of the dataset

```
: train_clinical_data.head()
```

	visit_id	patient_id	visit_month	updrs_1	updrs_2	updrs_3	updrs_4	updrs_4_missing	clinical_state_on_medication
0	55_0	55	0	10.0	6.0	15.0	0.0	1	On
1	55_3	55	3	10.0	7.0	25.0	0.0	1	On
2	55_6	55	6	8.0	10.0	34.0	0.0	1	On
3	55_9	55	9	8.0	9.0	30.0	0.0	0	On
4	55_12	55	12	10.0	10.0	41.0	0.0	0	On

Unified Parkinson's Disease Rating Scale

- **UPDRS 1:**

  Cognitive and emotional aspects: mood and behavior

- **UPDRS 2:**

  Activities of Daily Living

- **UPDRS 3:**

  Tremors, rigidity, slowness of movement and postural instability

- **UPDRS 4:**
  Disease medications or other therapeutic interventions

File exploration

Peptides data

Shape of the dataset:

```
Peptides_data.shape
```

```
(981834, 6)
```

Head of the dataset:

```
: Peptides_data.head()
```

	visit_id	visit_month	patient_id	UniProt	Peptide	PeptideAbundance
0	55_0	0	55	O00391	NEQEQPLGQWHLS	11254.3
1	55_0	0	55	O00533	GNPEPTFSWTK	102060.0
2	55_0	0	55	O00533	IEIPSSVQQVPTIIK	174185.0
3	55_0	0	55	O00533	KPQSAVYSTGSNGILLC(UniMod_4)EAEGEPQPTIK	27278.9
4	55_0	0	55	O00533	SMEQNGPGLEYR	30838.7

Peptides

- **UniProt:**

  Resource for protein sequence

- **Peptide:**

  Peptides associated with the disease can help in the early detection, diagnosis, and monitoring of the condition

- **Peptide Abundance:**

  Quantity of specific peptides present in biological sample

File exploration

Protein data

Shape of the dataset:

```
: Proteins_data.shape
(232741, 5)
```

Head of the dataset:

```
: Proteins.head()
```

	visit_id	NPX
0	10053_0	4.713356e+08
1	10053_12	4.666370e+08
2	10053_18	5.220732e+08
3	10138_12	6.515605e+08
4	10138_24	6.720872e+08

Protein

- **NPX:**

  Normalized Protein Expression (NPX) quantifies the relative amount of a specific protein.
  Measure used in proteomics to quantify the abundance of proteins.

Data cleaning strategy

```python
def fill_missing_updrs_123(data, updrs_column):
    # Create a temporary filled column within each patient group
    filled_column = f'{updrs_column}_filled'
    data[filled_column] = data.groupby('patient_id')[updrs_column].ffill()
    data[filled_column] = data.groupby('patient_id')[filled_column].bfill()
    data[updrs_column] = data[updrs_column].fillna(data[filled_column])

    # Dropping the temporary filled column
    data.drop(columns=filled_column, inplace=True)
    return data

# Filling missing values for UPDRS_1
train_clinical_data = fill_missing_updrs_123(train_clinical_data, 'updrs_1')

# Filling missing values for UPDRS_2
train_clinical_data = fill_missing_updrs_123(train_clinical_data, 'updrs_2')

# Filling missing values for UPDRS_3
train_clinical_data = fill_missing_updrs_123(train_clinical_data, 'updrs_3')
```

Train clinical data and supplemental clinical data:

UPDRS 1, UPDRS 2, UPDRS 3:

Initially, we created temporary columns to temporarily hold the filled data within each patient's record. These temporary columns were used to propagate non-missing values forward and backward within each patient group. Finally, we filled the missing values with the temporary filled data, ensuring we accounted for each patient's unique data sequence. This process helped us maintain data integrity without losing individual patient distinctions.

Data cleaning strategy

Train clinical data and supplemental clinical data:

UPDRS 4:

We began by generating an additional column to flag missing UPDRS_4 values, creating a binary indicator where 1 signifies a missing value and 0 indicates existing data. Afterward, we tackled the missing UPDRS_4 values directly by substituting them with zeros. This approach helps differentiate between missing and actual zero values, enabling clarity in the dataset.

```
:
# Creating a binary indicator column for missing UPDRS_4 values
train_clinical_data['updrs_4_missing'] = train_clinical_data['updrs_4'].isnull().astype(int)

# Filling missing values in UPDRS_4 with zeros
train_clinical_data['updrs_4'] = train_clinical_data['updrs_4'].fillna(0)
```

Merging data sets

8- Merging all three final Datasets

```
#MergingData

Merged_data = pd.merge(Peptides, Proteins, on=['visit_id'])

Merged_data2 = pd.merge(Final_clinical_data, Merged_data, on='visit_id')

Final_merge = Merged_data2[['visit_id', 'patient_id', 'visit_month', 'updrs_1'

Final_merge
```

✓ 0.1s

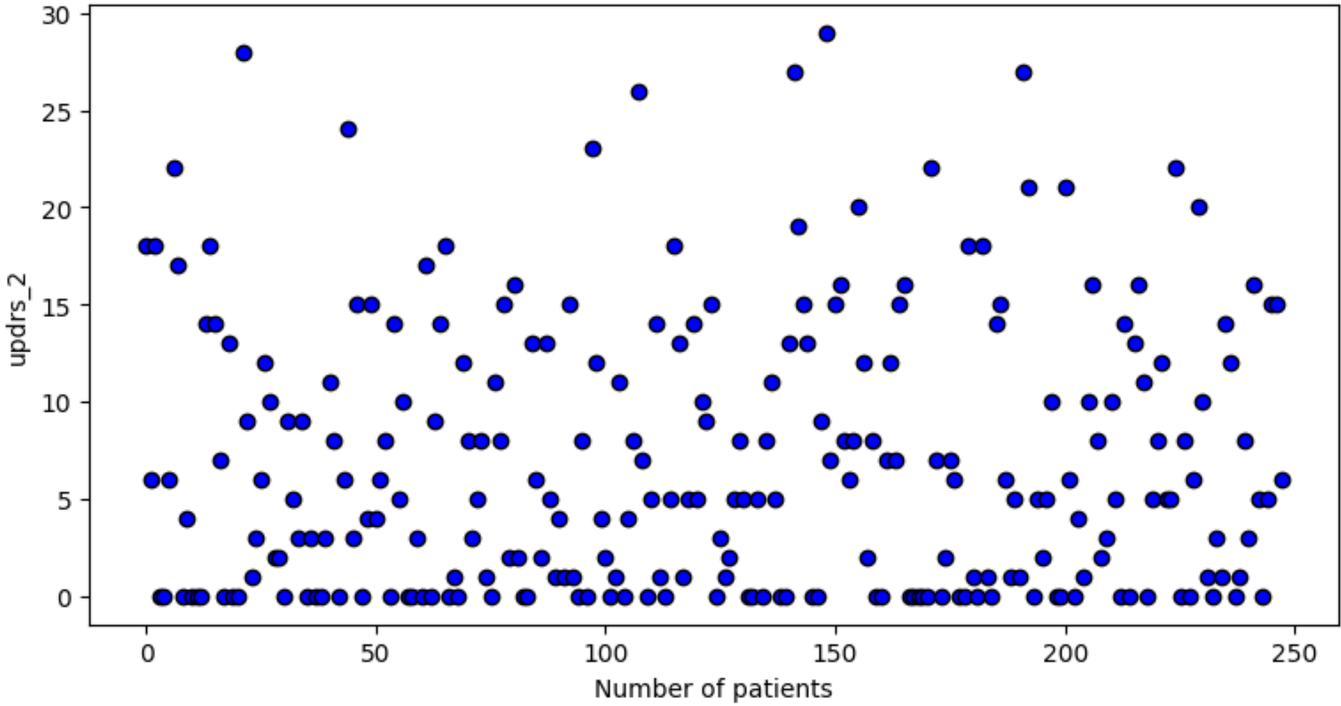
Data Merging

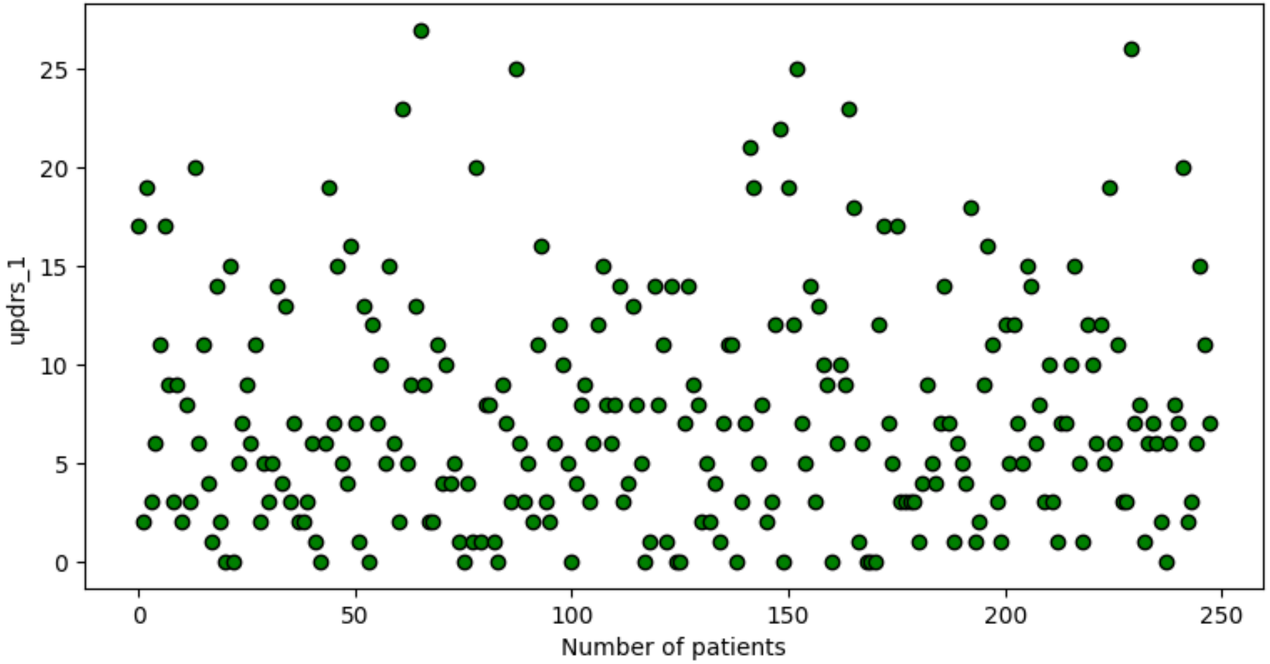
The merging of both datasets of train clinical data and supplemental clinical data that have similar columns with our peptide and protein data

Data Engineering

Creation of a new feature “Total UPDRS”, which is is the most widely applied rating instrument for Parkinson disease

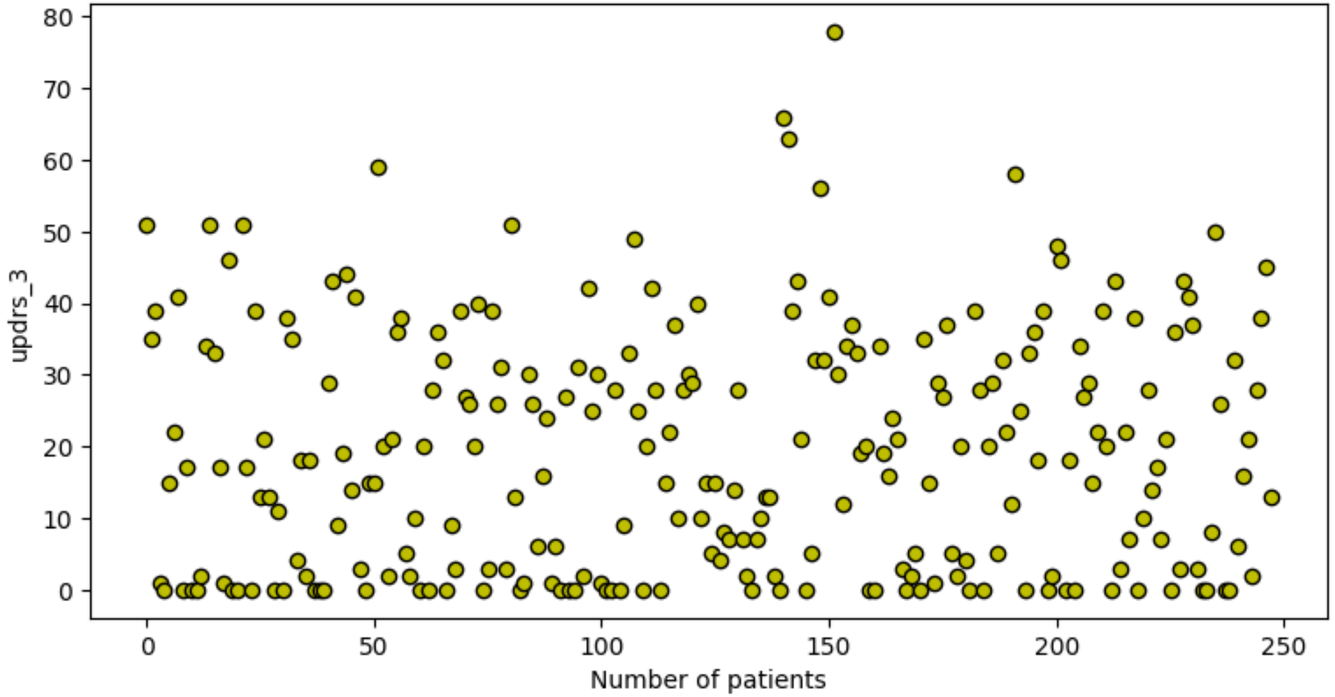
UPDRS 1

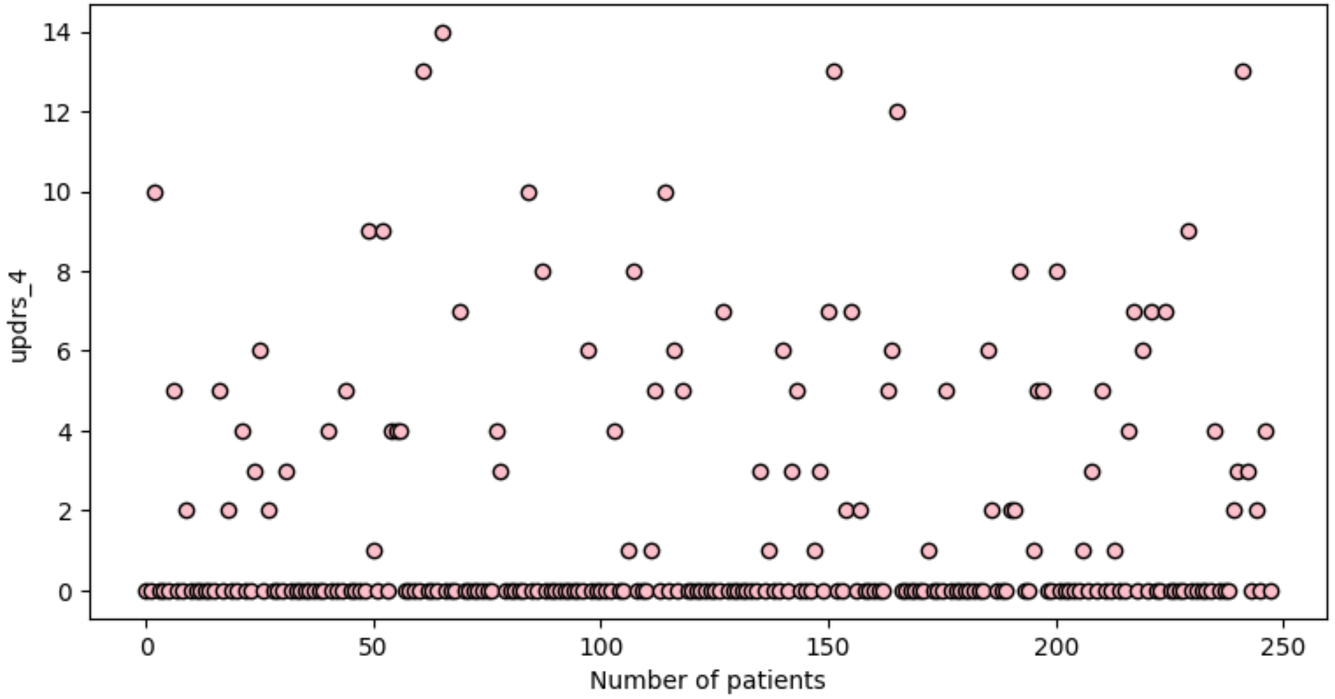




UPDRS 2

UPDRS 3





UPDRS 4

Data manipulation

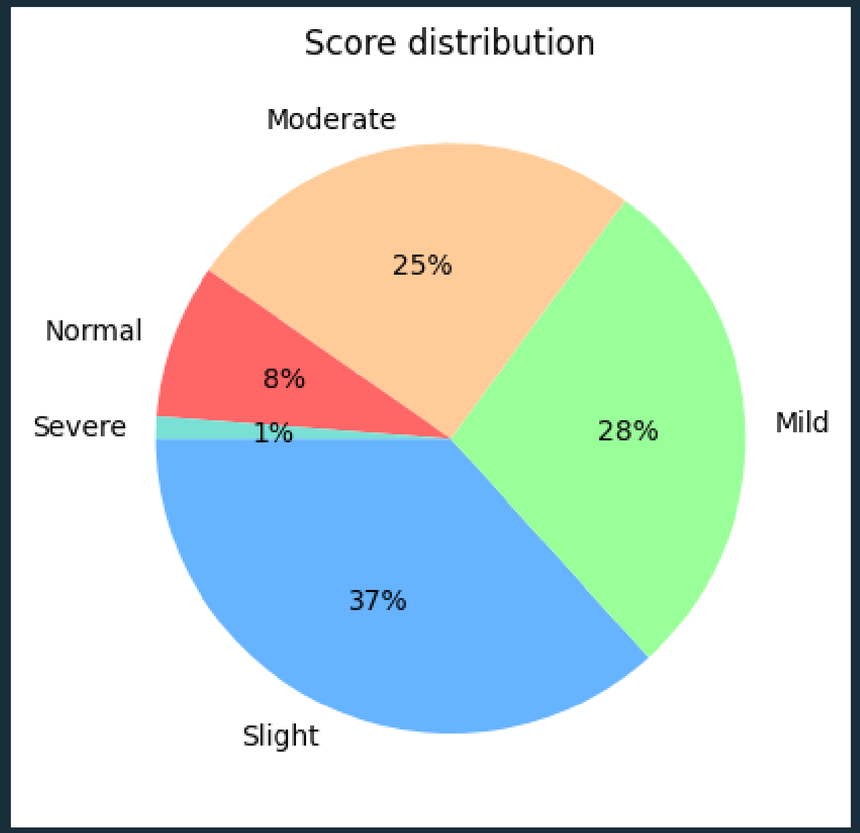
- **Rank Total_UPDRS**

```
Conditions = [
    Final_clinical_data_graphical["Total_UPDRS"] <= 15,
    (Final_clinical_data_graphical["Total_UPDRS"] > 16) & (Final_clinical_data_graphical["Total_UPDRS"] <= 40),
    (Final_clinical_data_graphical["Total_UPDRS"] > 41) & (Final_clinical_data_graphical["Total_UPDRS"] <= 70),
    (Final_clinical_data_graphical["Total_UPDRS"] > 71) & (Final_clinical_data_graphical["Total_UPDRS"] <= 100),
    Final_clinical_data_graphical["Total_UPDRS"] > 100
]

values = [0, 1, 2, 3, 4]

Final_clinical_data_graphical["UPDRS_Score"] = np.select(Conditions, values, default=0)
Final_clinical_data_graphical
```

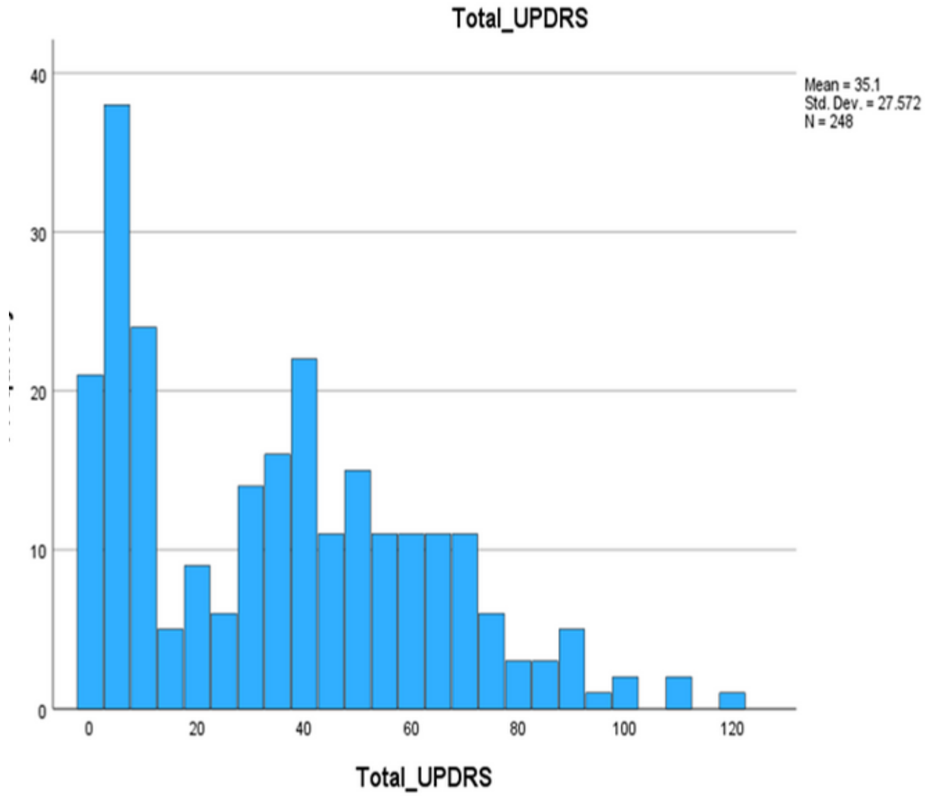
- **Print new entry**

pdrs_3	updrs_4	updrs_4_missing	clinical_state_on_medication	Total_UPDRS	NPX	PeptideAbundance	Standardized_NPX	Peptide Abundance Standardized	UPDRS_Score
51.0	0.0	0	On	86.0	7.649231e+08	7.649234e+08	2.137472	2.137476	3
35.0	0.0	0	Off	43.0	5.979835e+08	5.979831e+08	0.333456	0.333452	2
39.0	10.0	0	Off	86.0	5.840639e+08	5.840636e+08	0.183036	0.183032	3
1.0	0.0	1	Unknown	4.0	7.711963e+08	7.711966e+08	2.205263	2.205267	0
0.0	0.0	1	Unknown	6.0	6.153522e+08	6.153520e+08	0.521149	0.521147	0
...	...	...	...	...	...	...	...	...	...
2.0	0.0	0	Unknown	5.0	5.435102e+08	5.435107e+08	-0.255204	-0.255199	0
28.0	2.0	0	Off	41.0	5.450832e+08	5.450833e+08	-0.238205	-0.238204	0
38.0	0.0	0	Off	68.0	5.587043e+08	5.587045e+08	-0.091011	-0.091008	2

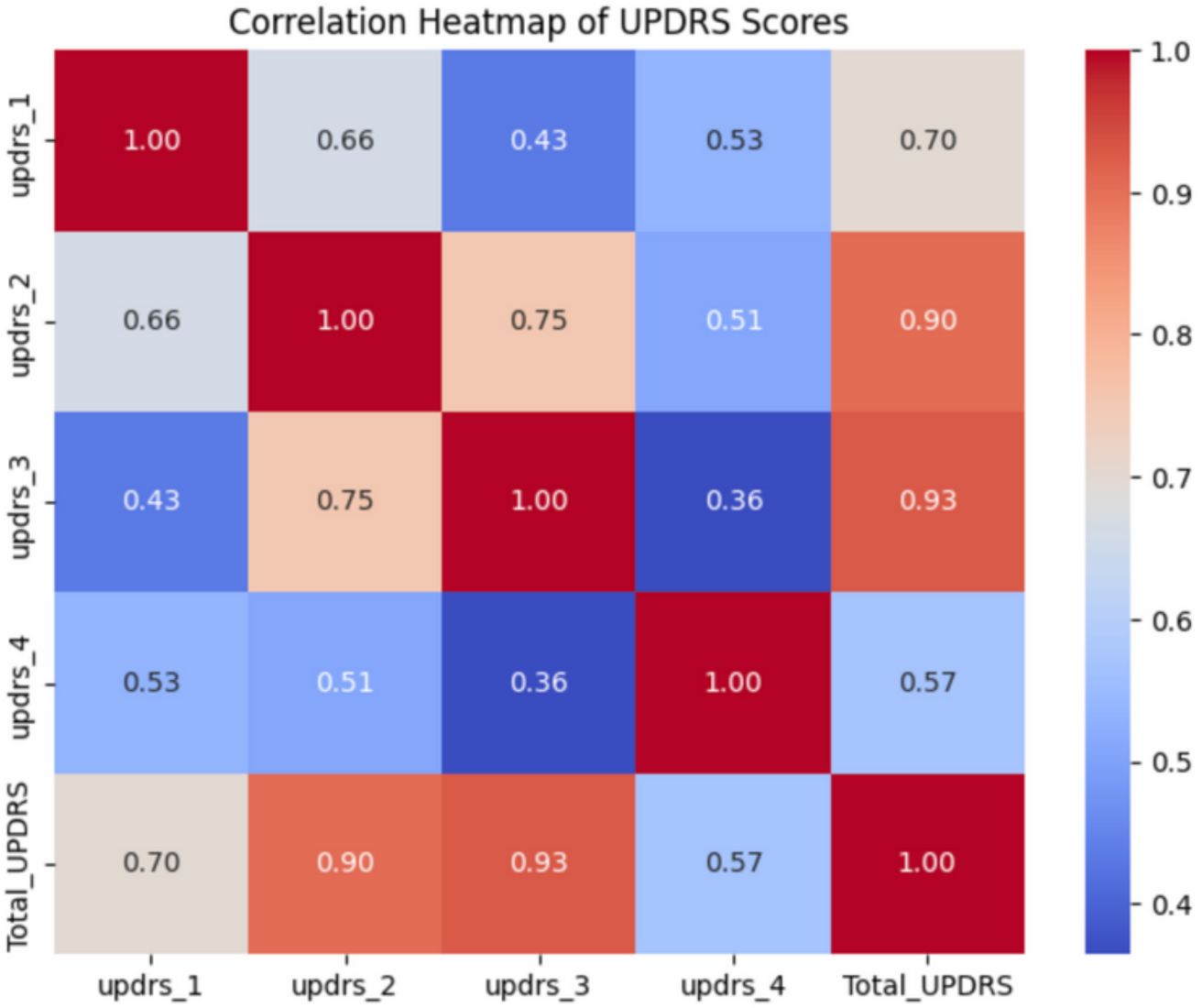
Score Distribution



UPDRS Frequency



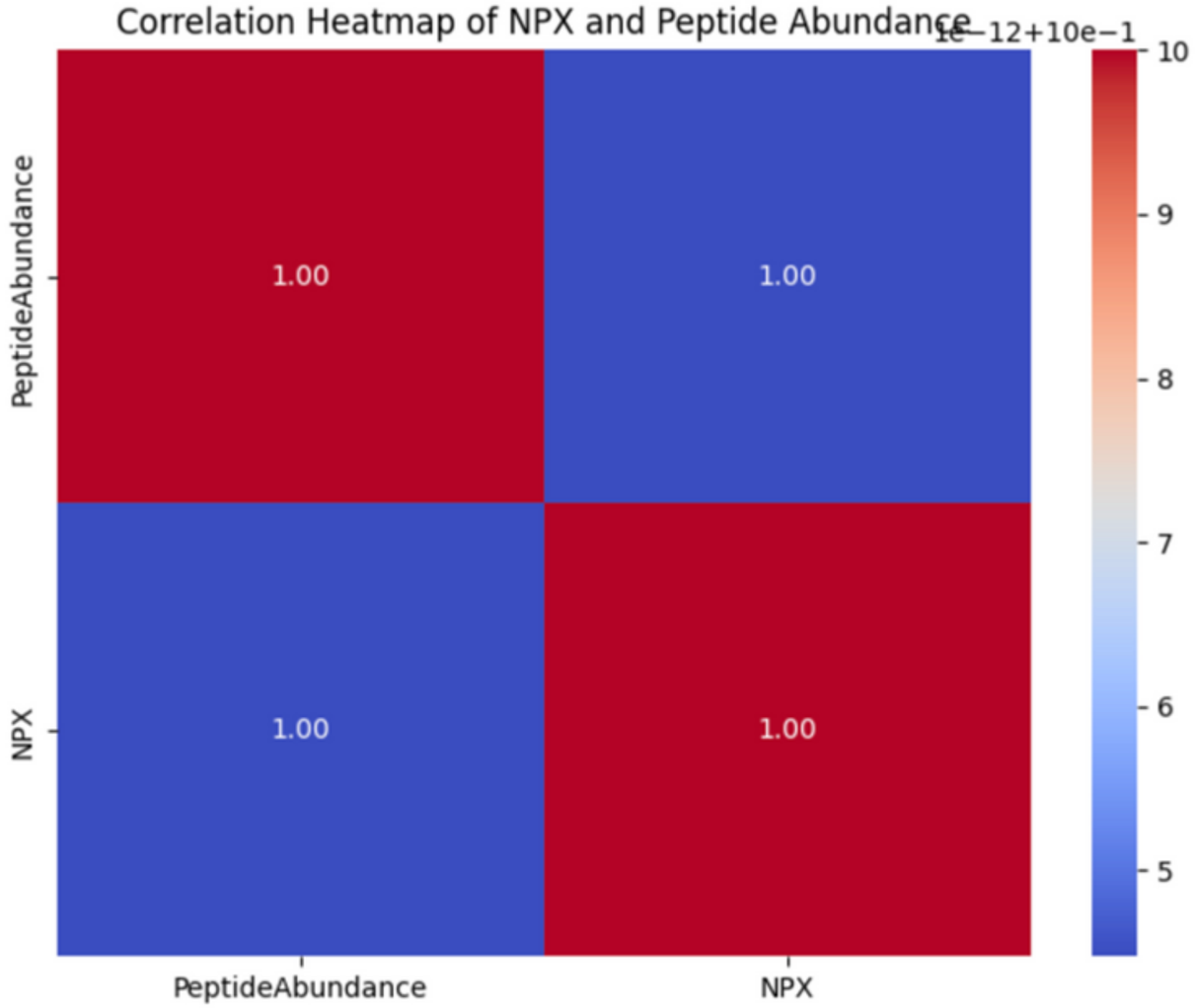
Data manipulation



Heatmap Conclusion

UPDRS 2 (Non-Motor**)** and **UPDRS 3** (Motor) seem to be highly correlated with the **Total UPDRS**. Healthcare professionals may need to consider both motor and non-motor symptoms when assessing the overall condition of a Parkinson's disease patient.

Data manipulation



Heatmap Conclusion

There is a very strong relationship between NPX and the summed Peptide abundance for each patient.Given the strong correlation we might consider using either NPX or Peptide Abundance for our machine learning model as they seem to capture similar infformation

Logistic regression

We did a logistic regression modeling on scaled data . Then we run the model again utilizing both L1 (Lasso) and L2 (Ridge) regularization techniques. where we employed a cross-validation to assess and compare the models' performance in predicting outcomes on the dataset.

Model	Regular Logistic regression	Logistic regression with l1 l2, 1st solver	Logistic regression with l1 l2, 2nd solver
sMAPE	147.89	46.55	46.55

Decision Tree

The code divides data for training and testing, it explores various tree parameters using GridSearchCV for optimisation, then evaluates the best model's performance on the test set and displays the chosen parameters alongside the accuracy achieved.

RMSE	sMAPE	R2
50.145	17.025	0.9238

Random Forest

We divided our dataset for training and testing, scaled features using StandardScaler, and optimized a Random Forest Regressor through GridSearchCV for the best parameters. Evaluating on the test set using SMAPE, we achieved strong predictive performance. This approach ensures accuracy in our model's predictions.

RMSE	sMAPE	R2
3.82	11.939	0.9778

Random Forest

```python
# Split the data into training and testing sets
X_train, X_test, train, test = train_test_split(Xless, y, test_size=0.25, random_state=3)

# Feature scaling using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Random Forest Regressor model
rf_regressor = RandomForestRegressor()

# Define the parameter grid for hyperparameter optimization
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Use GridSearchCV to find the best hyperparameters with SMAPE metric
grid_search = GridSearchCV(rf_regressor, param_grid, cv=5)
grid_search.fit(X_train_scaled, train)

# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Make predictions on the test set
pred = best_estimator.predict(X_test_scaled)

# Evaluate the model's performance using SMAPE
smape = calculate_smape(test, pred)
print("Best Parameters:", best_params)
print("SMAPE:", smape)
```

In a parallel exploration, we implemented a Random Forest model with a reduced set of features, further refining our approach for enhanced efficiency without compromising predictive accuracy.

RMSE	sMAPE	R2
3.689	11.54	0.979

K-Means CLustering

We employed KMeans clustering on a refined subset of features to unveil inherent patterns within our dataset. Using a distortion-based method, the algorithm suggested an optimal cluster count of three, balancing efficiency and meaningful segmentation. Visualizing the clusters in a three-dimensional space revealed distinct groupings, highlighting potential insights within our data


THANK YOU FOR YOUR
ATTENTION!

CODE and QA