Problem 2 Introduction

1a. Sneha Kelkar, sgk18001

1b. In this notebook, we are comparing two AI models to predict the most likely value of the motor UPDRS for Parkinson's patients. Primarily, we are using AdaBoost and Random forest. In the end, we will compare the results displayed by each model and pick the best

1c. Methods

1. As described, no missing value is recorded, so handling null values is not an issue
2. Even though it is unclear whether we should scale the values or not, through trial and error, scaling the variables produced better results
3. We will use GridSearch to optimize the models, comparing the validation values. We will optimize the models using GridSearchCV
4. We will use cross validation to obtain reliable estimates of the test mean squared error and make sure no patient is simultaneously represented in both split data sets

In [41]:
```
# ------------------------------------------------------------------------------------------
# load the libraries that are required for this project:
#-------------------------------------------------------------------------------------------
import pandas as pd             # Pandas is for data analysis and structure manipulation
import numpy as np              # NumPy is for numerical operations
```

Data Cleaning

In [42]:
```
# load the dataset and drop 'total_UPDRS'
df = pd.read_csv('dataset_parkinson.csv')
df = df.drop('total_UPDRS', axis = 1)
```

In [43]:
```
#inspecting the data types and checking for missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5875 entries, 0 to 5874
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   name            5875 non-null   int64  
 1   motor_UPDRS     5875 non-null   float64
 2   Jitter(%)       5875 non-null   float64
 3   Jitter(Abs)     5875 non-null   float64
 4   Jitter:RAP      5875 non-null   float64
 5   Jitter:PPQ5     5875 non-null   float64
 6   Jitter:DDP      5875 non-null   float64
 7   Shimmer         5875 non-null   float64
 8   Shimmer(dB)     5875 non-null   float64
 9   Shimmer:APQ3    5875 non-null   float64
 10  Shimmer:APQ5    5875 non-null   float64
 11  Shimmer:APQ11   5875 non-null   float64
 12  Shimmer:DDA     5875 non-null   float64
 13  NHR             5875 non-null   float64
 14  HNR             5875 non-null   float64
 15  RPDE            5875 non-null   float64
 16  DFA             5875 non-null   float64
 17  PPE             5875 non-null   float64
dtypes: float64(17), int64(1)
memory usage: 826.3 KB
```

In [44]:
```
df.describe()
```

Out[44]:

	name	motor_UPDRS	Jitter(%)	Jitter(Abs)	Jitter:RAP	Jitter:PPQ5	Jitter:DDP	Shimmer	Shimmer(dB)	Shimmer:APQ3	Shimmer:APQ5	…
count	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	
mean	21.494128	21.296229	0.006154	0.000044	0.002987	0.003277	0.008962	0.034035	0.310960	0.017156	0.020144	
std	12.372279	8.129282	0.005624	0.000036	0.003124	0.003732	0.009371	0.025835	0.230254	0.013237	0.016664	
min	1.000000	5.037700	0.000830	0.000002	0.000330	0.000430	0.000980	0.003060	0.026000	0.001610	0.001940	
25%	10.000000	15.000000	0.003580	0.000022	0.001580	0.001820	0.004730	0.019120	0.175000	0.009280	0.010790	
50%	22.000000	20.871000	0.004900	0.000034	0.002250	0.002490	0.006750	0.027510	0.253000	0.013700	0.015940	
75%	33.000000	27.596500	0.006800	0.000053	0.003290	0.003460	0.009870	0.039750	0.365000	0.020575	0.023755	
max	42.000000	39.511000	0.099990	0.000446	0.057540	0.069560	0.172630	0.268630	2.107000	0.162670	0.167020	

Scaling and splitting the data

In [45]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
unique_patients = df['name'].unique()
np.random.shuffle(unique_patients)
print(unique_patients)

# Split unique patients into train and test groups
train_patients, test_patients = train_test_split(unique_patients, test_size=0.3, random_state=42)

train_df = df[df['name'].isin(train_patients)]
test_df = df[df['name'].isin(test_patients)]

X_train = train_df.drop(['motor_UPDRS', 'name'], axis = 1)
X_test = test_df.drop(['motor_UPDRS', 'name'], axis = 1)
y_train = train_df['motor_UPDRS']
y_test = test_df['motor_UPDRS']

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
[37 19 41 29  2 11 12 28 26 16 10 15 25 31 27 38 17 13  6 30 34 33  9 36
  5 32 20 40 42  3 21  4 39 22 35  1  8 23 18 24  7 14]
```

after getting inconsistent results for sample values using 'StratifiedShuffleSplit' we can use 'test_train_split' from the same library which randomly splits the data into trianing and testing sets automatically for a balanced data set

Model Training

In [46]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
#based off of Practice 04
```

Random Forest

In [47]:
```
# Hyper-parameter grid
param_grid_rf = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [10, 20, 30, 40],
    'max_features': [None, 'sqrt', 'log2'],
    'bootstrap': [True],
}

rf = RandomForestRegressor(random_state =42)
grid_rf = GridSearchCV(rf, param_grid=param_grid_rf, scoring='neg_mean_squared_error', cv=3, n_jobs=-1)
grid_rf.fit(X_train_scaled, y_train)


best_params_rf = grid_rf.best_params_
best_score_rf = grid_rf.best_score_
# Print the best parameters and their score on the validation set
print("Best parameters for Random Forest:", best_params_rf)
print("Best MSE on the validation set for Random Forest:", best_score_rf)
# Now you can retrain your model on the full training set with the best parameters
rf_best = RandomForestRegressor(**best_params_rf, random_state=42)
rf_best.fit(X_train_scaled, y_train)

y_pred = rf_best.predict(X_test_scaled)
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}')
```

```
Best parameters for Random Forest: {'bootstrap': True, 'max_depth': 10, 'max_features': None, 'n_estimators': 200}
Best MSE on the validation set for Random Forest: -100.86027431055403
Mean squared error: 73.51313658392789
```

Gradient Boosting (AdaBoost)

In [48]:
```
param_grid_adaboost = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1],
    'estimator__max_depth': [1, 2, 3],
}
base_estimator = DecisionTreeRegressor()
estimator = AdaBoostRegressor(estimator=base_estimator)
grid_search_ada = GridSearchCV(estimator=estimator, param_grid=param_grid_adaboost, scoring='neg_mean_squared_error', cv=3)

grid_search_ada.fit(X_train_scaled, y_train)

best_params_adaboost = grid_search_ada.best_params_
best_score_adaboost = grid_search_ada.best_score_
# Print the best parameters and their score on the validation set
print("Best parameters for AdaBoost:", best_params_adaboost)
print("Best score on the validation set for AdaBoost:", best_score_adaboost)

best_model = DecisionTreeRegressor(max_depth = best_params_adaboost['estimator__max_depth'])
best_model = AdaBoostRegressor(estimator = best_model, n_estimators = best_params_adaboost['n_estimators'],
                               learning_rate = best_params_adaboost['learning_rate'], random_state=42)
best_model.fit(X_train_scaled, y_train)

y_pred = best_model.predict(X_test_scaled)
test_MSE = mean_squared_error(y_test, y_pred)
print(f" Mean Squared Error on the test set: {test_MSE:.4f}")
```

```
Best parameters for AdaBoost: {'estimator__max_depth': 2, 'learning_rate': 1, 'n_estimators': 200}
Best score on the validation set for AdaBoost: -90.20736780385869
 Mean Squared Error on the test set: 54.6584
```

1d. Results

According to the results displayed, Forest Tree shows better results. Therefore, we will pick it model for our application. When allowing patients to enter both the training and the test dataset, the results become much more reasonable. As it currently stands, the model cannot be used for prediction due to the lack of consistency in the data, this was. EDIT: I noticed that the more I ran the code, the worse the results became..

In [ ]:
```
```