Regression Analysis Report

I. Introduction:

1.1 Problem Statement:

“How can voice features like jitter, shimmer and frequency change act as indicators of how parkinsons affect the vocal system. “

1.2 Dataset Description:

The dataset used in this analysis is the Parkinson’s Telemonitoring Dataset, obtained from the UCI Machine Learning Repository. It contains voice measures and clinical scores to predict Parkinson’s severity.

- Features:
    - Demographics: Age, Sex (Binary: 0 - Male, 1 - Female)
    - Time-Based: Test Time (Continuous, days since recruitment)
    - Voice Measures:
        - Frequency Variation: Jitter (%), Jitter (Abs), Jitter:RAP, Jitter:PPQ5, Jitter:DDP
        - Amplitude Variation: Shimmer, Shimmer(dB), Shimmer:APQ3, Shimmer:APQ5, Shimmer:APQ11, Shimmer:DDA
        - Noise Ratios: NHR, HNR
        - Signal Complexity: RPDE, DFA, PPE
- Targets:
    - motor_UPDRS: Continuous, motor impairment score
    - total_UPDRS: Continuous, overall Parkinson’s severity score

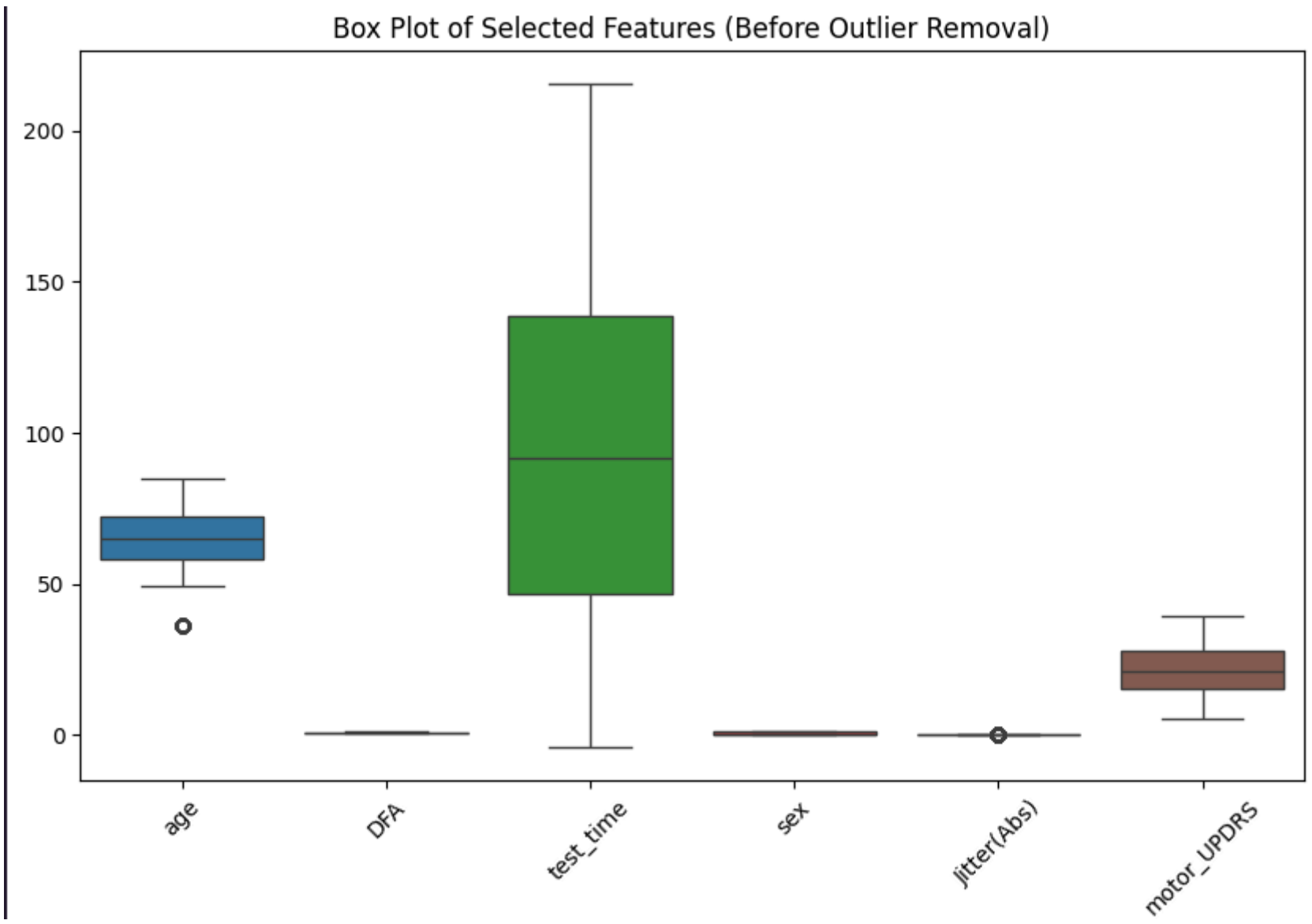
1.3 Objective

The objective of this analysis is to build a regression model that estimates the target variable (motor_UPDRS) based on the given features in the dataset.

II Methodology

2.1 Data Preprocessing

In this analysis, no significant data preprocessing was needed. Looking into the correlation of the features with the target variable, it was clear that the features and the target were not linearly correlated. That insight is the main reason why a decision tree regression model was chosen for this analysis. There were no missing values in the dataset. ṫotal_updrs̀ was dropped so as to not leak data. The outliers were noticed on the following columns and were dropped.




```
age: 101 outliers
DFA: 0 outliers
test_time: 0 outliers
sex: 0 outliers
Jitter(Abs): 366 outliers
motor_UPDRS: 0 outliers
39.511 5.0377
```

```
Shape before removing outliers: (5875, 6)
Shape after removing outliers: (5414, 6)
```

2.2 Feature Selection:

Summary of the findings from this section:

#	Train RMSE	Test RMSE	Train R^2	Test R^2	What was changed?
1	1.341	2.517	0.973	0.904,	No Change in the dataset, features were left as is.
2	1.975	2.572	0.941	0.902	Subcategories of voice measures were averaged to create a consolidated value.
3	1.640	2.076	0.960	0.935	Top 5 features deemed most important were selected.
4	1.485	2.137	0.967	0.933	Top 7 features deemed most important was selected

These findings were from naively setting the max_depth to 10 and min_sample_split to 5

Although decision trees implicitly handle feature selection implicitly, in this case manual intervention was done. Initially, all the 19 meaningful
features were kept as is. As decision trees check each feature at each split, the tree took very long to train. After testing the performance can be seen on #1 in the above table.

After looking at the heatmap, it was not a surprise to see different sub-variations of voice measures be correlated, so to reduce redundancy and help the training time, related features were combined together and averaged. After this change, the time to train the decision tree was significantly reduced. The results of testing this iteration can be in in #2 in the above table.

Although the training time was significantly reduced, a slight decline in model performance was observed. Given that performance is a critical factor in this case, optimizing hyperparameters using cross-validation and grid search proved to be a major computational bottleneck, often requiring approximately 30 minutes per iteration. This made even minor adjustments highly time-consuming.

To enhance both training efficiency and model accuracy, reverse feature selection was initially considered. However, due to the computational complexity of implementing this method with a custom-built model, it was deemed impractical. Instead, a more efficient approach was adopted by utilizing a pre-trained Decision Tree Regressor from Scikit-learn. Grid search was employed to identify the optimal hyperparameters for this model. Subsequently, the feature importances were extracted and ranked according to their relevance.

This approach effectively approximated the process of reverse feature selection while significantly reducing computational overhead. Since both models (the custom implementation and the Scikit-learn decision tree) operate under the same fundamental principles, it was assumed that their feature importance rankings would be relatively similar. This method provided a practical trade-off between computational efficiency and model interpretability without compromising performance. The model concluded that the following features were the most important:

```
Decision Tree Regressor Performance:
R-squared: 0.942
Mean Squared Error: 3.673
Root Mean Squared Error: 1.917

Sorted Feature Importances:
age: 0.6893
DFA: 0.0886
test_time: 0.0762
sex: 0.0441
Jitter(Abs): 0.0307
RPDE: 0.0143
NHR: 0.0134
PPE: 0.0130
Shimmer:APQ3: 0.0123
Shimmer: 0.0110
Jitter:PPQ5: 0.0026
HNR: 0.0014
Jitter:RAP: 0.0007
Shimmer:APQ5: 0.0007
Jitter:DDP: 0.0005
Jitter(%): 0.0005
Shimmer(dB): 0.0003
Shimmer:DDA: 0.0003
Shimmer:APQ11: 0.0003
```

Features deemed most important by sklearn's pre built Decision Tree Regressor

With that insight, the Top N features were selected:
- Top 5 Features: Observation can be seen in #3 of the above table.
- Top 7 features: Observation can be found in #4 of the above table

In conclusion,

- Model 1 performs exceptionally well on the training data (lowest Train RMSE and highest Train R^2) but has a relatively high Test RMSE (2.517) and lower Test R^2 (0.904). This suggests it might be overfitting the training data.

- Model 3 has a good balance between Train and Test performance, with the lowest Test RMSE (2.076) and highest Test R^2 (0.935). It generalizes well to unseen data.

- Model 4 also performs well, with a low Train RMSE (1.485) and high Train R^2 (0.967), and decent Test RMSE (2.137) and Test R^2 (0.933).

Thus, going further, model 3 is chosen.

2.3 Performance Of Model From Scratch:

Due to our model being from scratch, running a GridSearchCV or even a RandomSearchCV takes very long as the model is not thoroughly optimised unlike prebuilt ones from , thus the result of the gridsearch on sklearn's prebuilt decision tree regressor. So a GridSearchCV was performed on the sklearn's optimised model. Upon which, the following parameters were found to be the best:

```
{'max_depth':None, 'min_samples_leaf': 4, 'min_samples_split': 10}
```

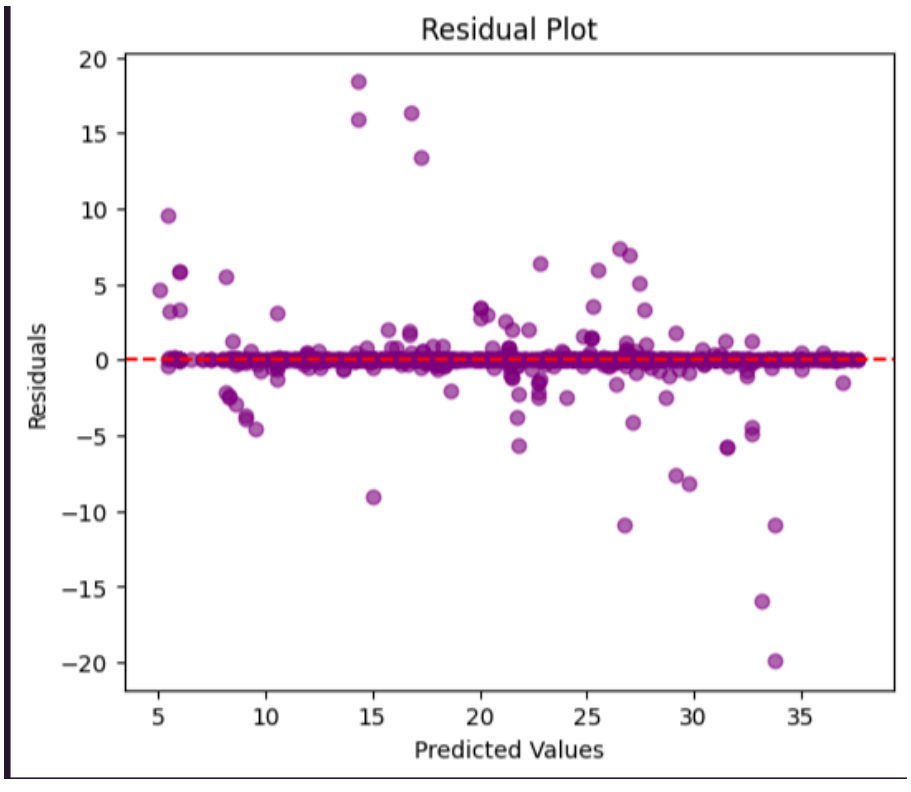
With these parameters, our custom model was run, and thus achieved a final performance of:

Note: The actual value for max_depth was set to 100 due to computational complexities and the time it took to train the model from scratch

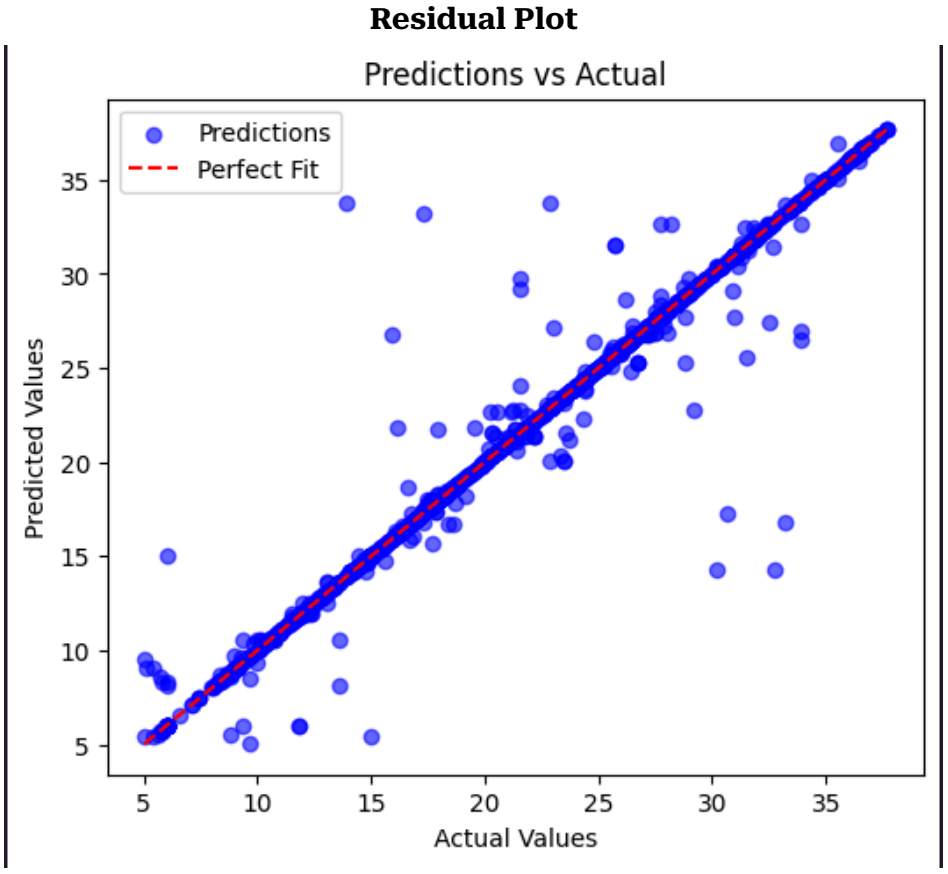
```
Train R²: 0.990, Train RMSE: 0.833
Test R²: 0.975, Test RMSE: 1.275
```

```
Cross-validation R2 scores: [0.98680593 0.95673693 0.9664107  0.97016346 0.97977016 0.98354258
 0.95255693 0.91988994 0.95461224 0.95076184 0.9463263  0.96033323
 0.98283046 0.97467011 0.97861429]
Mean R2: 0.964268339211189
Standard Deviation: 0.017500537384788354
```

Result Cross Validation with 15 folds for R^2



Residual Plot



Predicted vs Actual plot

IV Comparison Against Other Models

```
Without Hyperparameter Optimization:
Linear Regression - RMSE: 7.916042639931879, R-squared: 0.06409826364056115
Ridge Regression - RMSE: 7.9160281350303645, R-squared: 0.06410169342250982
Lasso Regression - RMSE: 8.040807425940931, R-squared: 0.034364273461730765
Decision Tree - RMSE: 1.548368976278438, R-squared: 0.9641934131408831
KNN Regressor - RMSE: 5.020846244565821, R-squared: 0.6234975147623767

With Hyperparameter Optimization:
Decision Tree - Best RMSE: 1.422436278945712, Best R-squared: 0.9697810309424479
KNN Regressor - Best RMSE: 3.9314555661951816, Best R-squared: 0.7691548543286174
Ridge Regression - Best RMSE: 7.914860647904227, Best R-squared: 0.06437773303013938
Lasso Regression - Best RMSE: 7.914052140915509, Best R-squared: 0.06456887183574989
```

Results before and after hyper parameter optimization for prebuilt models

For comparison, KNN regressor, Decision Tree Regressor and variations of Linear Regression models were imported from sk-learn. They were run both before and after hyperparameter optimization.

Without Hyperparameter Optimization:

- Decision Tree performs the best, with the lowest RMSE (1.363) and highest R-squared (0.972).
- KNN Regressor is the second-best, with an RMSE of 3.231 and R-squared of 0.842.

With Hyperparameter Optimization:

- Decision Tree is still the best, with an RMSE of 1.42 and R-squared of 0.969

Linear, Ridge, and Lasso Regression perform poorly, with high RMSE and low R-squared values both before and after optimization, showing that this dataset is not well suited for these models.

IV Conclusion

4.2 Final Model

After evaluating the models and applying hyperparameter optimization, the Decision Tree Regressor proved to be the most effective model for this dataset. Initially, without hyperparameter optimization, the Decision Tree performed the best, with the lowest RMSE of 1.548 and the highest R-squared of 0.964. KNN Regressor followed, but the results were less impressive.

After applying hyperparameter optimization, Decision Tree maintained its position as the best model, with the best RMSE improving to 1.42 and the R-squared increasing to 0.97. On the other hand, while KNN's performance improved with optimization, it still didn’t surpass the Decision Tree. KNN achieved an RMSE of 3.93 and an R-squared of 0.769, which was a significant improvement but still not enough to outperform the Decision Tree.

Linear models, such as Ridge and Lasso regression, continued to underperform both before and after optimization, demonstrating that they were not well suited for this dataset. Their RMSE remained high, and R-squared values were low, reinforcing that linear models did not capture the underlying patterns effectively.

Therefore, Decision Tree remained the final choice, particularly due to its balance between accuracy and ability to handle complex, non-linear relationships in the data.

4.3 Challenges

The biggest challenge encountered in this analysis was feature selection. The dataset’s lack of correlation between features and the target variable made linear regression unsuitable, as evidenced by the poor performance metrics. Identifying the most relevant features was key to improving model performance and reducing noise. It became clear that unnecessary features increased training time and complicated the model, ultimately hindering performance.

Given that the target variable motor_UPDRS ranges from ~5 to ~36, the challenge was to minimize RMSE without overfitting the model.
Another key insight was the addition of the min_samples_leaf parameter in the Decision Tree. Initially overlooked, this hyperparameter played a critical role in preventing overfitting and boosting model performance, ultimately making a significant difference in the results.

V Discussion:

5.1 Impact of Hyperparameter Tuning:

V. Discussion

5.1 Impact of Hyperparameter Tuning
The hyperparameter tuning had a major impact on model performance. Without it, the model produced the following results:

Train RMSE: 1.640
Test RMSE: 2.076
Train R^2: 0.960
Test R^2: 0.935

However, after optimizing the hyperparameters, performance significantly improved:

Train RMSE: 0.833
Test RMSE: 1.275
Train R^2: 0.990
Test R^2: 0.975

This shows the essential role hyperparameter tuning plays in boosting the model's performance. By fine-tuning key parameters, we were able to enhance the model's ability to generalize to unseen data, achieving lower RMSE and higher R-squared values. Hyperparameter tuning reduced overfitting and improved both the training and testing results, making the model more reliable and accurate.

VI. Summary Of Results:

Model	R2 Test	RMSE	Approach
From Scratch	0.904	2.517	As Is.
From Scratch	0.935	2.076	After Feature Selection
From Scratch	0.975	1.275	After Hyper Parameter Optimisation
From Scratch	0.964	<- 0.0175 standard deviation ->	After Cross Validation with 15 folds.
KNN Regression	0.769	3.931	After Hyper Parameter Optimisation
Decision Tree (prebuilt)	0.969	1.422	After Hyper Parameter Optimisation
Linear Regression	0.064	7.916	“”
Lasso Regression	0.0645	7.9140	“”
Ridge Regression	0.0643	7.9148	“”