Sequential Modelling of Typing Performance Logs

Ashutosh Rai
2422308

November 10, 2025

Data Science Lab (CS 6301)

Department of Computer Science and Engineering
National Institute of Technology, Silchar

Contents

1 Abstract

Sequential log data from typing sessions, including metrics such as words per minute (WPM), accuracy, and consistency, forms the basis for analyzing and predicting personal typing performance trends. This project develops a workflow to model performance using a curated dataset of typing logs from the Monkeytype platform. The dataset, consisting of 1000 sequential sessions from a single user, is treated as a time-series problem.

The initial methodology considered a distributed preprocessing pipeline, as the original goal was to analyze thousands of records. However, due to recent platform policy changes limiting data exports, the scope was focused on a more modest dataset. Consequently, a more agile preprocessing workflow using the Pandas library was adopted. The core objective was to perform supervised sequence learning to forecast typing performance.

The analysis began with a **regression** task, to predict the user's exact WPM in the next session. This approach was unsuccessful since multiple models, including tree-based regressors, failed to outperform simple baselines, indicating that the next-session WPM target is too noisy and likely influenced by uncaptured human factors.

Based on this finding, the problem was reframed as a **classification** task, to predict the **direction** of the next session's performance (Increase, Decrease, or Stable). This pivot was successful. The final LightGBM classification model achieved **69.23%** accuracy, significantly outperforming a naive baseline and demonstrating a strong ability to predict negative performance trends. This report details the complete pipeline, from data preprocessing and exploratory analysis to the iterative modeling process that led to these findings.

2 Introduction

The ability to quantify and model human skill acquisition is a significant challenge in computational analytics. Understanding and predicting performance trends has direct real-world applications, such as powering adaptive learning systems that adjust difficulty in real-time, personalizing digital tutors, or optimizing professional training programs by identifying when a user is about to plateau or underperform. This project applies this concept to typing, a fundamental digital-age skill. It utilizes performance logs from **Monkeytype**, a popular online typing platform, to investigate whether a user's future performance can be modeled based on their past sessions.

The dataset for this analysis comprises **645** sequential typing test logs from a single user, treated as a time-series. The initial objective was to solve a **regression** problem: to predict the exact **Words Per Minute (WPM)** a user would achieve in their subsequent session. This report details the experimental process which first established this regression task to be unfeasible. We found that the target variable (next-session WPM) exhibits high variance and is "noisy," likely influenced by uncaptured confounding variables. These "human factors," such as transient focus, fatigue, or mood, are not recorded in the data and make precise WPM prediction exceptionally difficult.

Due to the poor performance of the regression approach, the problem was reframed. Instead of predicting a precise value, a simpler, more robust **classification** problem was formulated. The new objective was to predict the **directional change** in performance: will the user's WPM **Increase**, **Decrease**, or remain **Stable** relative to their current session? This pivot in methodology proved to be successful.

This report documents the complete data science pipeline, from data ingestion to final model evaluation. It is structured as follows:

- **Dataset:** A description of the raw data attributes collected from Monkeytype.
- **Data Preprocessing:** The methodology used to clean, filter, and structure the raw data for time-series analysis.
- **Exploratory Data Analysis (EDA):** Key insights and visualizations derived from the processed data.
- **Methodology & Results:** A detailed comparison of the initial regression approach and the final, successful classification model.
- **Conclusion:** A summary of the project's findings and their implications.

3 Dataset

The dataset for this project was directly downloaded from the "monkeytype" platform, provided as a CSV file. This raw file contained **1000 session records**, where each row represents a single typing test. The file included 24 attributes, describing a mix of performance metrics, error statistics, and test configurations.

The core performance metrics included **wpm** (Words Per Minute), **rawWpm** (the gross WPM before error correction), **acc** (accuracy percentage), and **consistency** (a percentage score for typing rhythm). A critical feature for error analysis was **charStats**, stored as a text string (e.g., "202;1;0;1"). This string encodes four distinct values: correct characters, incorrect characters, extra characters, and missed characters.

Contextual and behavioral attributes were also present. The **mode** (e.g., 'time') and **mode2** (e.g., 15, 30, 60) columns defined the test type and duration. The **timestamp** was a 64-bit integer representing the exact completion time in milliseconds since the Unix epoch, which is essential for placing the sessions in a correct chronological order. Other features like **restartCount** and **testDuration** offered insights into user behavior during a session.

The raw data also contained columns that were not useful for this analysis. For instance, **funbox** and **tags** were empty (all NULL values), while **_id** was a unique database identifier irrelevant for modeling. Table 1 provides a summary of the key features that were selected for the preprocessing stage.

Table 1: Description of Key Raw Data Features

Feature Name	Description
timestamp	64-bit integer: The Unix timestamp (milliseconds) of session completion.
wpm	Float: Final calculated Words Per Minute.
acc	Float: Final accuracy percentage.
rawWpm	Float: Raw WPM before error correction.
consistency	Float: Typing rhythm stability percentage.
charStats	String: Semicolon-delimited (e.g., "150;2;0;1") string of error counts.
mode	String: The primary test mode (e.g., 'time', 'words').
mode2	Integer: The test duration or word count (e.g., 30, 60, 100).
restartCount	Integer: The number of times the test was restarted.
testDuration	Float: The actual duration of the test in seconds.

4 Data Preprocessing

The raw data file was not suitable for direct analysis. The data had to be filtered, cleaned, and features had to be engineered to create a dataset ready for time-series modeling. This process was conducted using the Pandas library.

4.1 Data Filtering

The raw data contained sessions of multiple types (e.g., 15, 30, and 60-second tests). To ensure all sessions were directly comparable, the dataset was first filtered to include **only the 30-second tests**, identified by `mode2 == 30`. Furthermore, as a deliberate design choice, the dataset was capped to the first **645** records of this type.

```
df_user1 = pd.read_csv('user1.csv')
filtered_user1 = df_user1[df_user1['mode2'] == 30].head(645).copy()
```

Listing 1: Loading and filtering the dataset in Pandas

4.2 Feature Removal

Next, non-informative or redundant columns were removed. These included:

- **All-Null Columns:** `funbox` and `tags`, which contained no data.
- **Irrelevant Metadata:** `_id` (a database key) and `isPb` (a personal-best flag).
- **Constant Difficulty Settings:** Columns like `punctuation`, `numbers`, `language`, and `difficulty` were constant ('False', 'False', 'english', 'normal') for all filtered records and thus provided no learning signal.

```
cols_to_drop = [
    '_id', 'mode', 'mode2', 'quoteLength', 'punctuation', 'numbers',
    'language', 'funbox', 'difficulty', 'lazyMode', 'tags',
    'blindMode', 'bailedOut', 'isPb'
]

filtered_user1 = filtered_user1.drop(columns=cols_to_drop)
```

Listing 2: Dropping non-informative columns

4.3 Feature Engineering

Two key transformations were performed. First, the `timestamp` (a millisecond-based integer) was converted into a proper `datetime` object. This is essential for all time-series analysis. From this new `datetime` column, a categorical feature, **`time_of_day`**, was engineered to label each session as 'morning', 'afternoon', 'evening', or 'night'.

```
filtered_user1['datetime'] = pd.to_datetime(
    filtered_user1['timestamp'], unit='ms'
)

```

```python
def categorize_time_of_day(hour):
    if 5 <= hour < 12:
        return 'morning'
    elif 12 <= hour < 17:
        return 'afternoon'
    elif 17 <= hour < 21:
        return 'evening'
    else:
        return 'night'

filtered_user1['time_of_day'] = filtered_user1['datetime'].dt.hour.apply(
    categorize_time_of_day
)

# Drop the original redundant timestamp
filtered_user1 = filtered_user1.drop(columns=["timestamp"])
```

Listing 3: Converting timestamp and engineering `time_of_day`

Second, the **`charStats`** column, which was a string (e.g., "202;1;0;1"), was parsed. This string was split by its semicolon delimiter into four new integer columns: **`correct_characters`**, **`incorrect_characters`**, **`extra_characters`**, and **`missed_characters`**. This step makes error counts explicit and usable by the models.

```python
def split_char_stats(df):
    char_stats_split = df['charStats'].str.split(';', expand=True)
    char_stats_split.columns = [
        'correct_characters', 'incorrect_characters',
        'extra_characters', 'missed_characters'
    ]

    for col in char_stats_split.columns:
        char_stats_split[col] = pd.to_numeric(
            char_stats_split[col], errors='coerce'
        ).fillna(0).astype(int)

    df = pd.concat([df.drop(columns=['charStats']), char_stats_split], axis
    =1)
    return df

filtered_user1 = split_char_stats(filtered_user1)
```

Listing 4: Parsing the `charStats` string column

4.4 Preprocessed Data

After these steps, the DataFrame was complete and contained 645 rows and 14 columns, all with correct data types and no missing values. This final, clean dataset was saved as a pickle file to be used in the subsequent analysis. The resulting dataset schema is shown below.

```
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   datetime              645 non-null    datetime64[ns]
 1   time_of_day           645 non-null    object
```

```
2   wpm                    645 non-null    float64
3   rawWpm                 645 non-null    float64
4   acc                    645 non-null    float64
5   consistency            645 non-null    float64
6   restartCount           645 non-null    int64
7   testDuration           645 non-null    float64
8   afkDuration            645 non-null    int64
9   incompleteTestSeconds  645 non-null    float64
10  correct_characters     645 non-null    int64
11  incorrect_characters   645 non-null    int64
12  extra_characters       645 non-null    int64
13  missed_characters      645 non-null    int64
```

Listing 5: Preprocessed Dataset Schema

5 Exploratory Data Analysis

After preprocessing, the clean dataset of 645 sessions was explored to uncover patterns, trends, and relationships. This analysis was essential for understanding the data's characteristics before attempting to build predictive models.

5.1 Descriptive Statistics

The first step was to compute the descriptive statistics for the key numeric features. This provides a high-level summary of the data's central tendency and spread. A selection of these statistics is presented in Table 2.

Table 2: Descriptive statistics for key attributes

Attribute	Mean	Std. Dev.	Median (50%)	Max
wpm	70.45	7.83	71.19	91.19
acc	96.12	2.41	96.48	100.00
consistency	71.58	6.85	72.41	86.69
restartCount	0.74	1.41	0.00	18.00

The data shows an average **WPM** of 70.45, with a standard deviation of 7.83, indicating a moderate amount of performance variance. The user's **accuracy** is high and relatively consistent (mean 96.12, std 2.41). It is also noteworthy that the **restartCount** is low on average (0.74) but has a high maximum (18), suggesting rare, outlier sessions where the user struggled.

5.2 Performance Distributions

To visualize the information in the table, a histogram in Figure 1 shows the distribution of the primary performance metric, **WPM**.

```
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.histplot(filtered_user1['wpm'], kde=True, bins=30)
plt.title("User 1 - WPM Distribution")
plt.xlabel("Words Per Minute (WPM)")
plt.ylabel("Frequency")
plt.show()
```

Listing 6: Plotting the WPM distribution

The user's performance is roughly normally distributed, centered around 70-75 WPM, but with a noticeable tail of lower-performing sessions, which pulls the mean slightly below the median.

5.3 Correlation Analysis

To understand the relationships between different metrics, a correlation heatmap was generated. Figure 2 visualizes the Pearson correlation coefficient between all major numeric features.

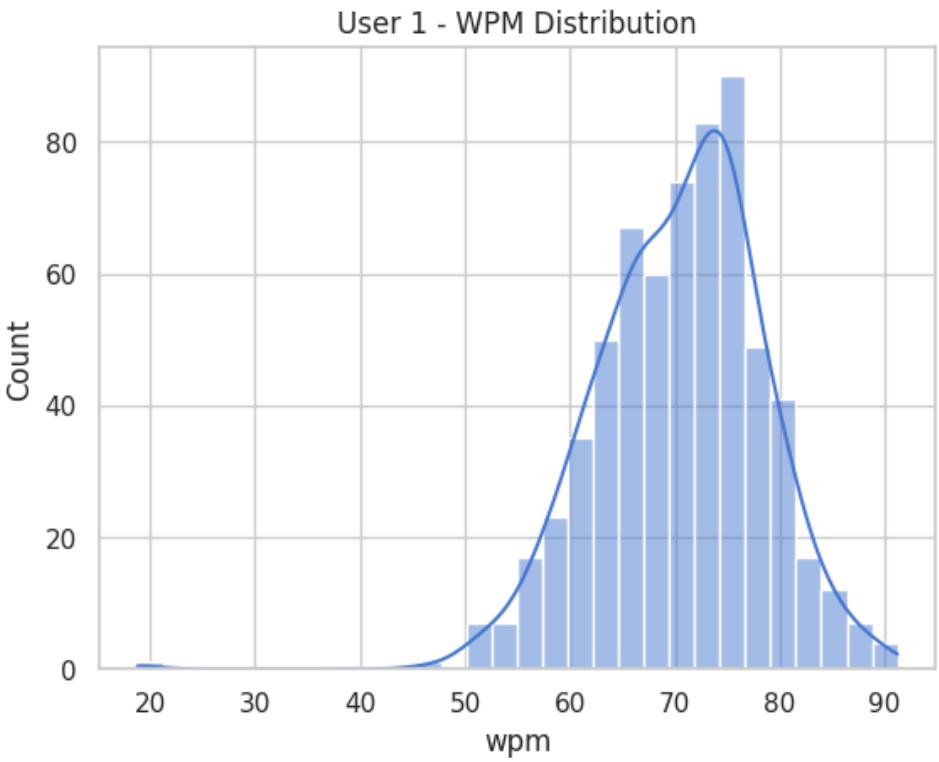

Figure 1: Distribution of WPM

```
numeric_cols = ['wpm', 'rawWpm', 'acc', 'consistency',
                'restartCount', 'testDuration', 'afkDuration',
                'correct_characters', 'incorrect_characters']

plt.figure(figsize=(10, 8))
sns.heatmap(filtered_user1[numeric_cols].corr(),
            annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap of Performance Metrics")
plt.show()
```

Listing 7: Plotting the correlation heatmap

The heatmap reveals strong positive correlations, such as **wpm** with **rawWpm** and **correct_characters**. A strong negative correlation is seen between **accuracy** and **incorrect_characters**, as expected.

5.4 Performance Over Time

The most critical analysis is tracking performance chronologically. Figure 3 shows the WPM for every session, sorted by `datetime`. This raw plot appears very ”noisy” or volatile, with performance jumping significantly between consecutive sessions. This high variance is what makes a regression task (predicting the exact next WPM) so difficult. To visualize the underlying trend, a 10-session rolling average was applied, as shown in Figure 4. The smoothed plot clearly reveals a gradual decline in the user’s performance over this specific 645-session period, a trend that was hidden in the noisy raw data.

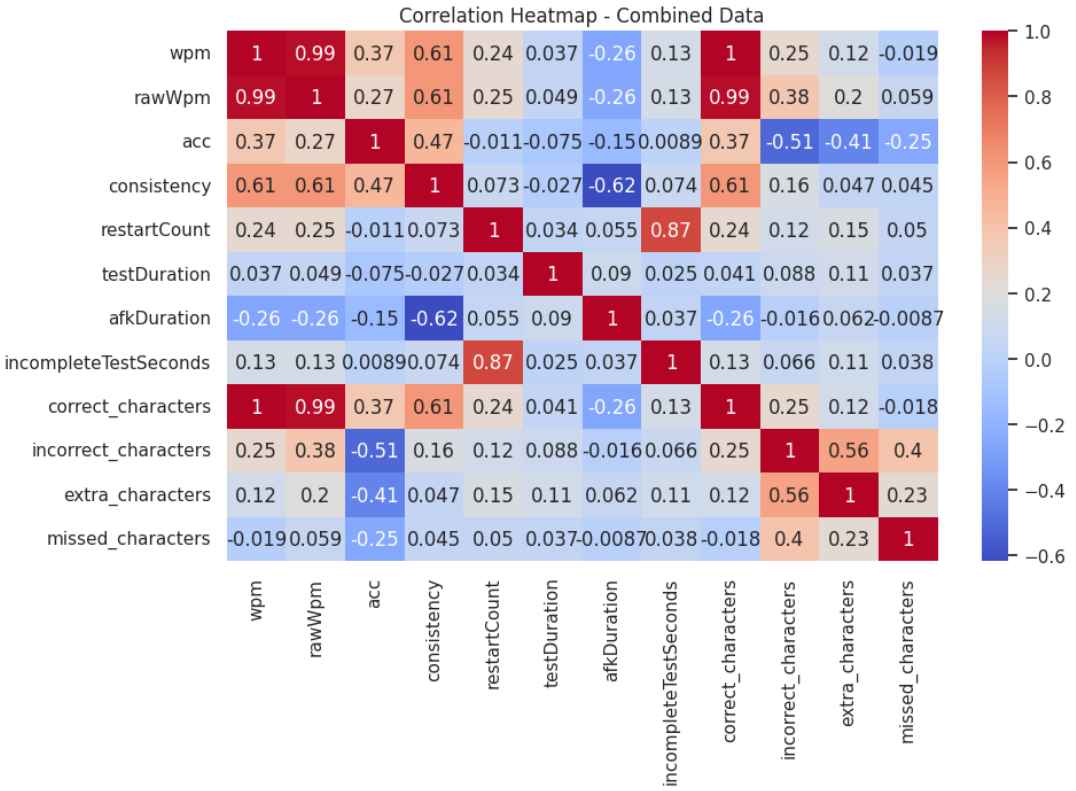

Figure 2: Correlation heatmap

5.5 Contextual and Behavioral Factors

Next, the impact of user behavior and context was investigated. Figure 5 and Figure 6 show the effect of restarting the test and the `time_of_day` on WPM. The analysis showed two interesting points. First, sessions that were restarted (1 or more times) have a slightly higher median WPM, suggesting the user restarts to correct a poor start. Second, there is no clear pattern indicating that one time of day is significantly better or worse for performance.

5.6 Session Clustering

To discover deeper, hidden patterns, an unsupervised clustering analysis was performed. The goal was to see if sessions could be automatically grouped into distinct ”session types”. The features were first scaled using `StandardScaler` so that all features were on a comparable scale.

```
FEATURES = [
    'wpm', 'rawWpm', 'acc', 'consistency',
    'correct_characters', 'incorrect_characters', 'extra_characters',
    'missed_characters', 'restartCount', 'testDuration'
]

X = filtered_user1[FEATURES].astype(float)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Listing 8: Scaling features for clustering

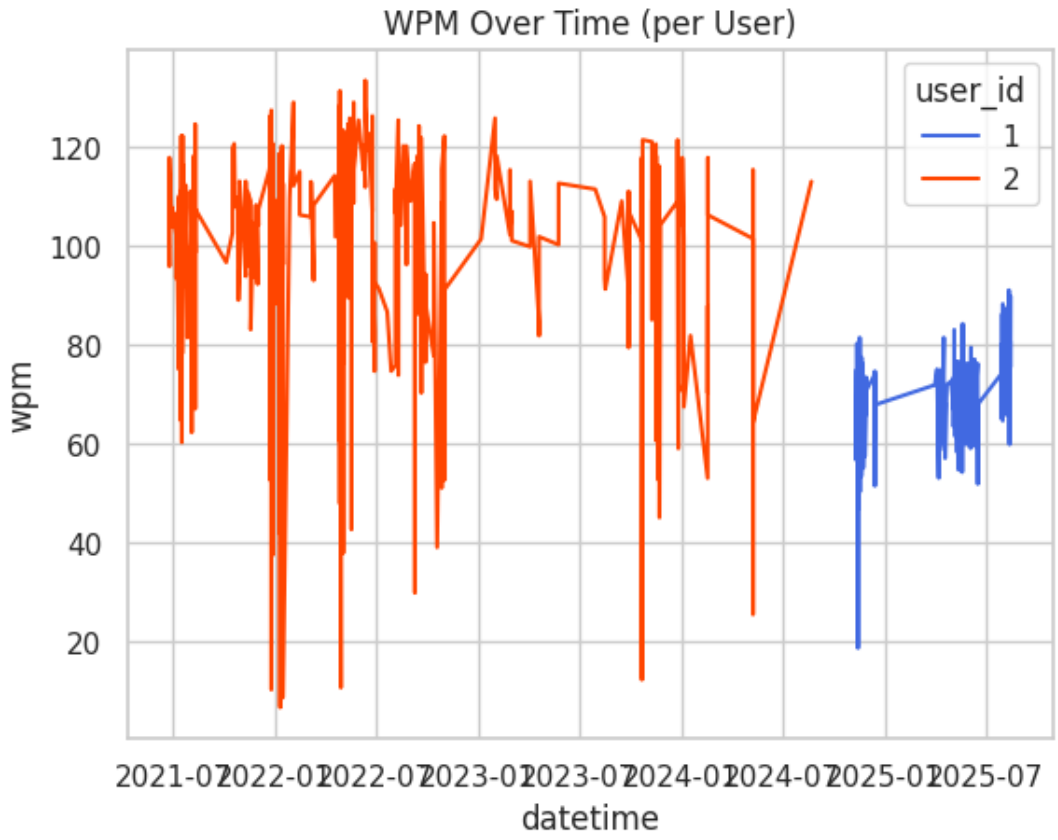

Figure 3: Raw WPM over time

KMeans clustering was performed. The optimal number of clusters, `k`, was determined by analyzing the Silhouette Score. A value of **k=8** produced the highest score (0.271) and was chosen as the optimal number of clusters. To visualize these 8 clusters, Principal Component Analysis (PCA) was used to reduce the 10-dimensional feature space down to 2 dimensions. Figure 7 plots the clusters in this new space. To understand what each cluster represents, a heatmap of the cluster profiles was generated. This heatmap in Figure 8 shows the average value of each feature for each cluster, standardized (z-scored) for easy comparison.

Based on the feature means for each cluster, we can define these 8 distinct session types:

- **Cluster 0:** High WPM and high accuracy, occurring almost exclusively at night.
- **Cluster 1:** High WPM but with low accuracy and high errors. This is a **fast but sloppy** profile.
- **Cluster 2:** Average WPM and high accuracy, occurring almost exclusively in the evening.
- **Cluster 3:** The largest cluster, representing the **baseline performance**.
- **Cluster 4:** A small outlier cluster with high WPM but an **extremely high restart count**.
- **Cluster 5:** The **peak performance** cluster, with the highest WPM and high accuracy.
- **Cluster 6:** A small outlier cluster with **extremely low WPM** and high AFK duration, likely representing interrupted tests.

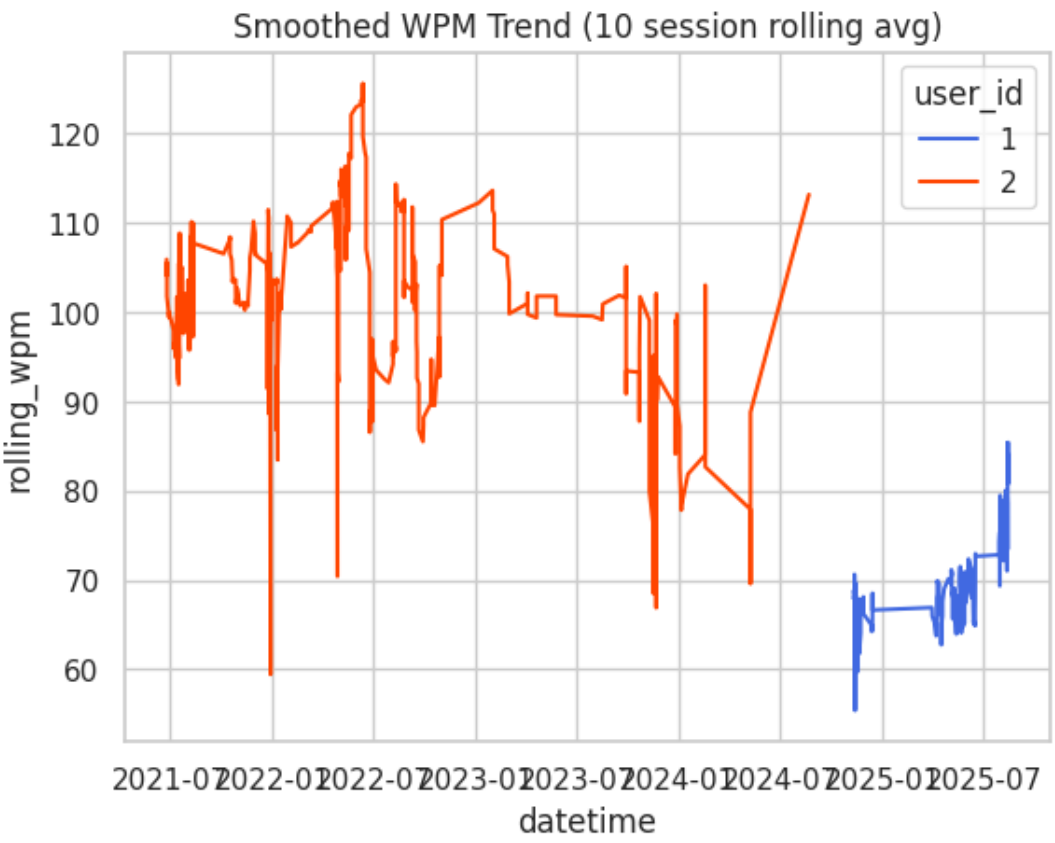

Figure 4: Smoothed WPM trend (10-session rolling average)

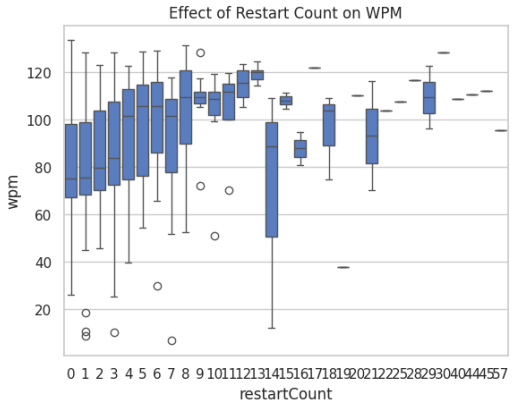

Figure 5: Effect of restart count on WPM

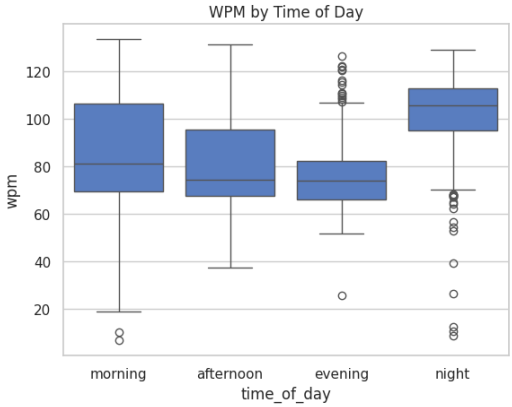

Figure 6: WPM by time of day

- **Cluster 7:** A tiny outlier cluster with high WPM but low accuracy.

This analysis confirms that distinct, identifiable performance patterns exist in the data.

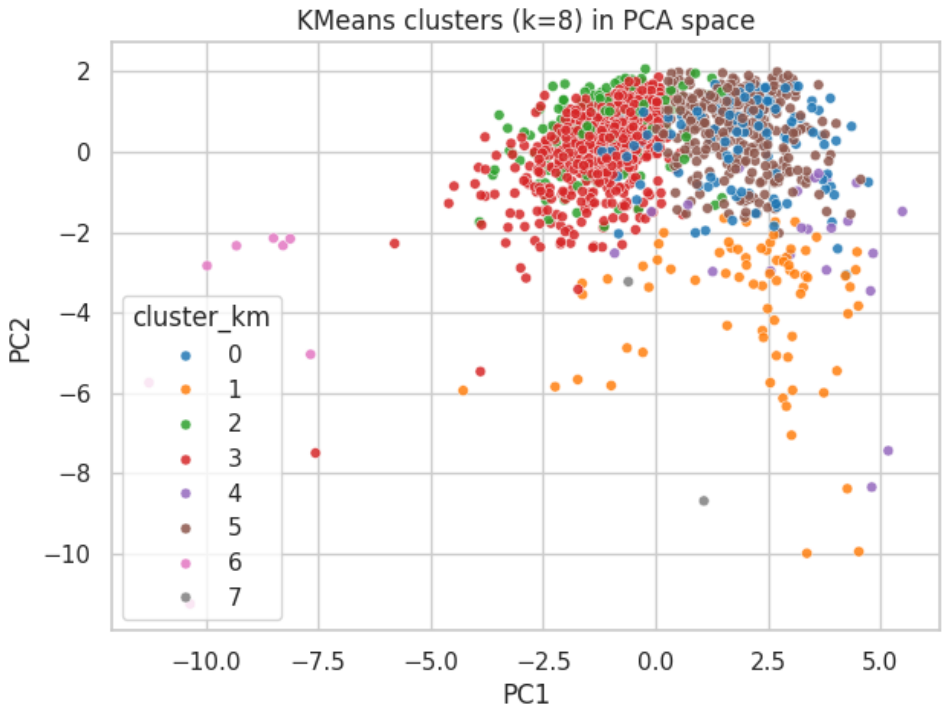

Figure 7: KMeans clusters (k=8) in PCA space

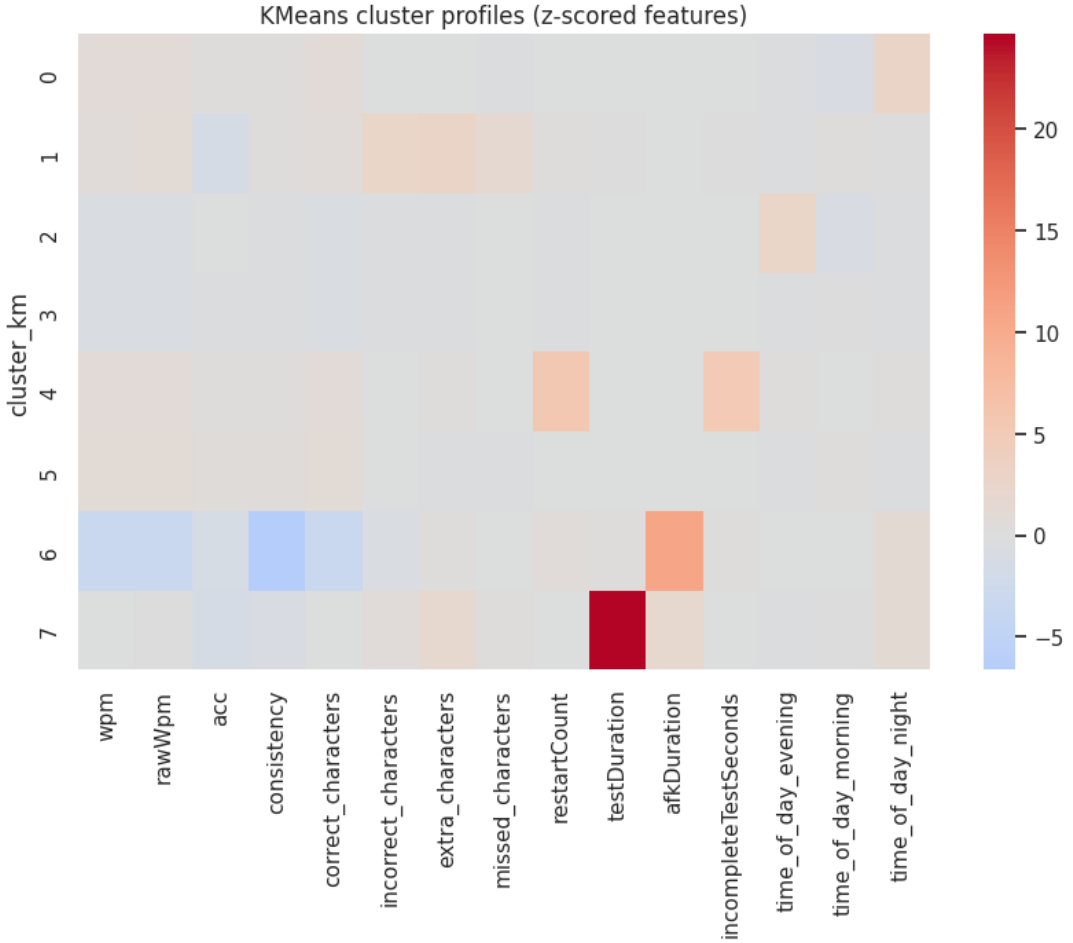

Figure 8: KMeans cluster profiles (z-scored)

6 Methodology and Results

This section details the core of the experimental process, which followed two main paths. First, a **regression** approach was attempted to predict the exact WPM of the next session. When this proved unsuccessful, the problem was reframed as a **classification** task. This section covers the methodology and results for both approaches.

6.1 Feature Engineering

The first step was to convert the processed DataFrame into a dataset suitable for supervised learning. Since the goal is to predict the *next* session's performance based on the *current* session, two things must be created:

1. A target variable, **`y_next`**, which is the WPM of the following session.
2. A set of features based *only* on past and current data.

The target **`y_next`** was created by shifting the `wpm` column up by one row. Features were created by calculating lags (`wpm_lag1`), rolling averages (`wpm_roll3_mean`), and exponential moving averages (`wpm_ewm3`). Cyclical features for `time_of_day` were also added.

```
def engineer_final_features(df):
    df = df.copy()
    df["y_next"] = df["wpm"].shift(-1)
    df["wpm_lag1"] = df["wpm"].shift(1)
    df["wpm_lag2"] = df["wpm"].shift(2)
    df[f"wpm_roll3_mean"] = df["wpm"].rolling(3, min_periods=1).mean()
    df[f"wpm_ewm3"] = df["wpm"].ewm(span=3, min_periods=1, adjust=False).
   mean()
    df[f"wpm_cummean"] = df["wpm"].expanding(min_periods=1).mean()
    df = df.dropna(subset=["y_next", "wpm_lag1", "wpm_lag2"])

    return df
```

Listing 9: Key logic of the feature engineering function

After dropping rows that had missing values from the shifting process (e.g., the first two rows and the last row), the final dataset had **643** usable samples, each with **12 features**.

```
Feature sample: ['wpm', 'acc', 'consistency', 'wpm_lag1',
'wpm_roll3_mean', 'wpm_ewm3', 'wpm_ewm5', 'wpm_minus_roll3',
'wpm_cummean', 'wpm_roll20_mean', 'tod_sin', 'tod_cos']
```

Listing 10: Final feature list

The resulting DataFrame structure is shown in the table below, which demonstrates how the target (`y_next` for a given row) aligns with the features of that same row. For example, in the first row, the model will learn to predict 62.80 (the target) using 56.80, 69.56, 63.18, etc. (the features).

	datetime	wpm	y_next	wpm_lag1	wpm_lag2	wpm_roll3_mean
1	2024-11-10 08:37:10	56.80	62.80	69.56	NaN	63.180000
2	2024-11-10 08:37:48	62.80	66.79	56.80	69.56	63.053333
3	2024-11-10 08:38:22	66.79	75.16	62.80	56.80	62.130000
4	2024-11-11 09:00:17	75.16	65.99	66.79	62.80	68.250000

```
5  2024-11-11 11:44:06  65.99   73.19     75.16     66.79       69.313333
```

Listing 11: Audit of the final time-series data structure

6.2 Evaluation Strategy

It is critical to respect the chronological order of the data. Using a standard k-fold or a random train-test split would cause data leakage, as the model would be "predicting the past" using data from the future. Therefore, a strict chronological split was used. The dataset (of 643 samples) was split into a **training set (the first 90%)** and a **test set (the final 10%)**. The model is trained on the past and evaluated only on the most recent, unseen "future" data.

```python
def time_split_single(X, y, meta, train=0.9, val=0.0, test=0.1):
    n = len(X)
    i1 = int(n * train)
    i2 = int(n * (train + val))

    X_tr, y_tr = X.iloc[:i1], y.iloc[:i1]
    X_va, y_va = X.iloc[i1:i2], y.iloc[i1:i2]
    X_te, y_te = X.iloc[i2:],  y.iloc[i2:]

    # ... (return splits) ...
```

Listing 12: Chronological data split

This split resulted in 578 samples for training and 65 samples for testing as evident from the console output.

```
Train size: (578, 12) (578,)
Test size: (65, 12) (65,)
```

Listing 13: Final data split

6.3 Regression Approach

The initial objective was to solve a **regression** problem: to predict the exact `y_next` (the WPM of the next session). Before training complex models, two simple baselines were established on the test set (65 samples) to measure a baseline performance.

- **Baseline 1 (Previous WPM):** Assumes the next WPM will be identical to the current WPM. This yielded a high Mean Absolute Error (MAE) of 6.730.
- **Baseline 2 (Rolling-3 WPM):** Assumes the next WPM will be the average of the last 3 sessions. This was a stronger baseline, with an MAE of 5.534.

Four models were then trained on the 578-sample training set to predict the log-transformed WPM, which helps stabilize variance. The models were: Linear Regression, Random Forest Regressor (RFR), LightGBM Regressor (LGBM), and XGBoost Regressor. Their performance on the test set is summarized in Table 3.

Table 3: Regression model performance on the test set

Model	MAE	RMSE	Accuracy (±3.0 WPM)
Baseline (Previous WPM)	6.730	8.905	32.31%
Baseline (Rolling-3 WPM)	5.534	7.167	N/A
Linear Regression	5.078	6.655	43.08%
LightGBM Regressor	5.053	6.681	**46.15%**
Random Forest Regressor	5.210	6.840	41.54%
XGBoost Regressor	5.649	7.431	43.08%

6.4 Regression Analysis and Conclusion

The results in Table 3 are conclusive. **None of the complex, tuned models could significantly outperform the simple 'Rolling 3-window average WPM' baseline.** The best-performing model, LightGBM, had an MAE of 5.053, only slightly better than the baseline's 5.534. Furthermore, its accuracy was very low, at only 46.15%.

Figure 9 (which plots the LightGBM model's predictions) shows this failure visually. The model's predictions (the dashed line) are overly-smoothed and closely follow a simple average. It completely fails to capture the high-variance, "noisy" spikes of the user's actual performance.

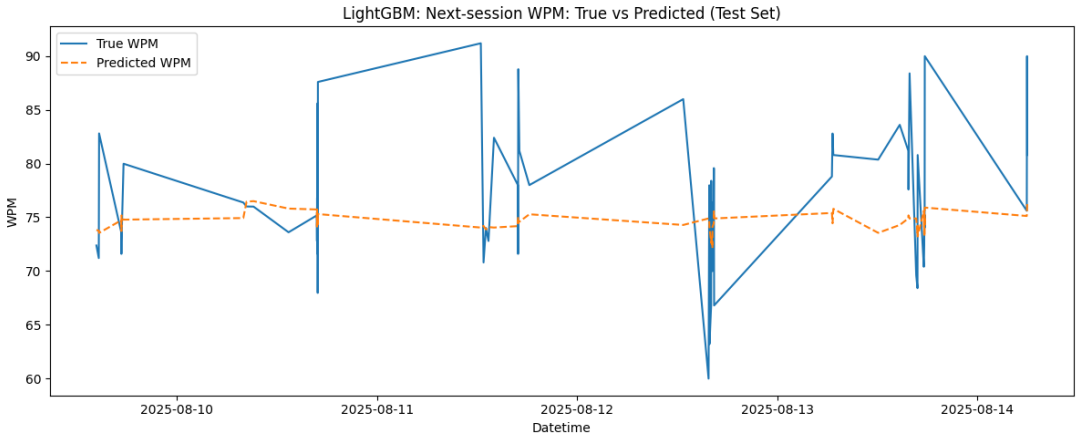

Figure 9: LGBM Regressor: True vs Predicted WPM

The reason for this failure becomes clear when analyzing the feature importances, as shown in Figure 11. Both the RFR and LGBM models are overwhelmingly reliant on features that are just simple variations of the recent WPM (e.g., `wpm_roll20_mean`, `wpm_roll3_mean`, `wpm_cummean`). This confirms that the models were not finding any deep, predictive signal. They simply learned the same rule as our baseline: "The next WPM will be very similar to the recent average." The conclusion is that the target variable (exact next-session WPM) is too noisy and highly influenced by uncaptured human factors (focus, fatigue, etc.) to be predicted accurately with the available features. This led to the decision to pivot the problem.

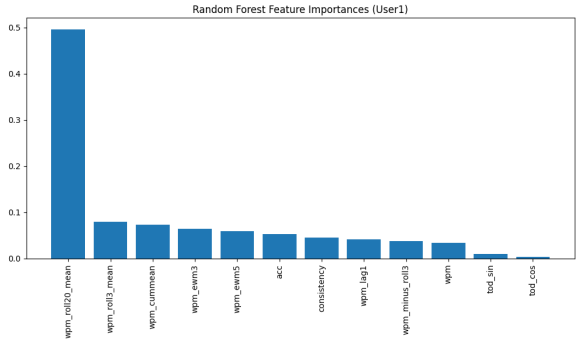

Figure 10: RFR Feature Importances

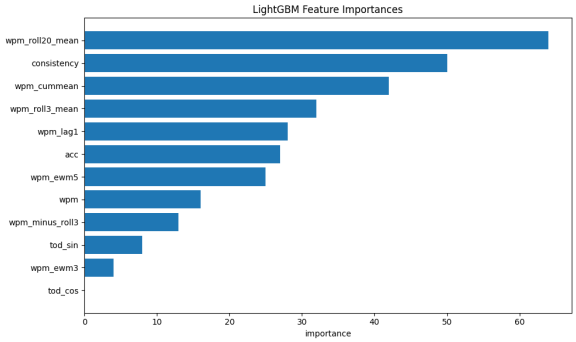

Figure 11: LightGBM Feature Importances

6.5 Problem Reformulation

Based on the failure of the regression approach, the problem was subsequently reformulated. Rather than predicting a continuous value (WPM), a more robust **classification** approach was adopted. The objective was redefined to predict the categorical **direction** of performance change: 'Increase', 'Decrease', or 'Stable'. A new target variable, `y_class`, was engineered by calculating the change in WPM from the current session to the next (`wpm_delta`) and binning it. A session was labeled 'Increase' or 'Decrease' if the WPM changed by more than 3.0, otherwise, it was labeled 'Stable'.

```
df['y_next'] = df['wpm'].shift(-1)
df['wpm_delta'] = df['y_next'] - df['wpm']

bin_width = 3.0
def classify_performance(delta):
    if delta > bin_width:
        return 'Increase'
    elif delta < -bin_width:
        return 'Decrease'
    else:
        return 'Stable'
df['y_class'] = df['wpm_delta'].apply(classify_performance)
```

Listing 14: Engineering the new classification target

A check of the training data (579 samples) revealed a well-balanced dataset, as shown in the console output. This confirmed that no oversampling techniques were necessary.

```
Training Set (y_tr) Class Counts:
y_class
Increase    204
Decrease    196
Stable      179
Name: count, dtype: int64
```

Listing 15: Training set class distribution

6.6 Model Comparison

The same 6-feature set and 90/10 time-series split (579 train, 65 test) were used to test multiple classification models. All models were tuned using **GridSearchCV** with **TimeSeriesSplit** to find the best hyperparameters. The performance of the models on the held-out test set is summarized in Table 4.

Table 4: Model comparison on the test set

Model	Best CV Accuracy	Test Accuracy
Logistic Regression (Polynomial)	50.23%	60.00%
Support Vector Classifier (SVC)	48.84%	55.38%
LightGBM Classifier	46.53%	**69.23%**

The **LightGBM Classifier** performed the best, achieving the highest test accuracy of **69.23%**. This is a strong positive result and a significant improvement over the other models. A naive baseline of predicting the majority class ('Increase', 25 instances) on the test set would have an accuracy of only 38.5% (25/65). We futher performed an in-depth analysis on the LightGBM classification model.

6.7 Model Results Analysis

The LightGBM Classifier, which achieved a 69.23% accuracy on the test set, was selected as the final model. This subsection details its performance, best parameters, and key predictive features. The model was trained using **GridSearchCV** with **TimeSeriesSplit** to find the optimal hyperparameters. The best parameters found were for a relatively simple and constrained model, which is appropriate for a dataset with a weak signal, preventing overfitting.

```
Best parameters found: {'learning_rate': 0.01, 'max_depth': 4,
'n_estimators': 200, 'num_leaves': 8}
```

Listing 16: Best hyperparameter set for the LGBM Classifier

6.7.1 Performance Metrics

The model's full performance on the test set is detailed in Table 5. The overall weighted F1-Score of 0.69 is a robust result. The model's strongest capability is identifying "Decrease" sessions, achieving a high Recall of 0.89 and an F1-Score of 0.81 for that class. This means it correctly identified 89% of all sessions where the user's performance was about to drop. Its performance on "Increase" (0.64 F1) and "Stable" (0.63 F1) sessions are also solid.

6.7.2 Confusion Matrix

The confusion matrix in Table 6 details the model's specific successes and failures. The values on the diagonal represent correct predictions. This matrix confirms the findings from the classification report. The model is highly effective on the "Decrease" class, correctly predicting 17 out of 19 instances. Its main area of confusion lies in distinguishing between "Stable" and "Increase" sessions. For example, it misclassified 6 "Stable" sessions as "Increase" and 6 "Increase" sessions as "Stable". This is logical, as the boundary between these two classes is the most subtle.

Table 5: Classification report for the LGBM model

Class	Precision	Recall	F1-Score	Support
Decrease	0.74	0.89	0.81	19
Increase	0.68	0.60	0.64	25
Stable	0.65	0.62	0.63	21
Weighted Avg	**0.69**	**0.69**	**0.69**	**65**

Table 6: Confusion matrix on the test set

	Predicted: Decrease	Predicted: Stable	Predicted: Increase
Actual: Decrease	**17**	1	1
Actual: Stable	2	**13**	6
Actual: Increase	4	6	**15**

6.7.3 Feature Importances

Finally, the model's feature importances were analyzed to see which features it found most predictive. Figure 12 shows the importance scores.

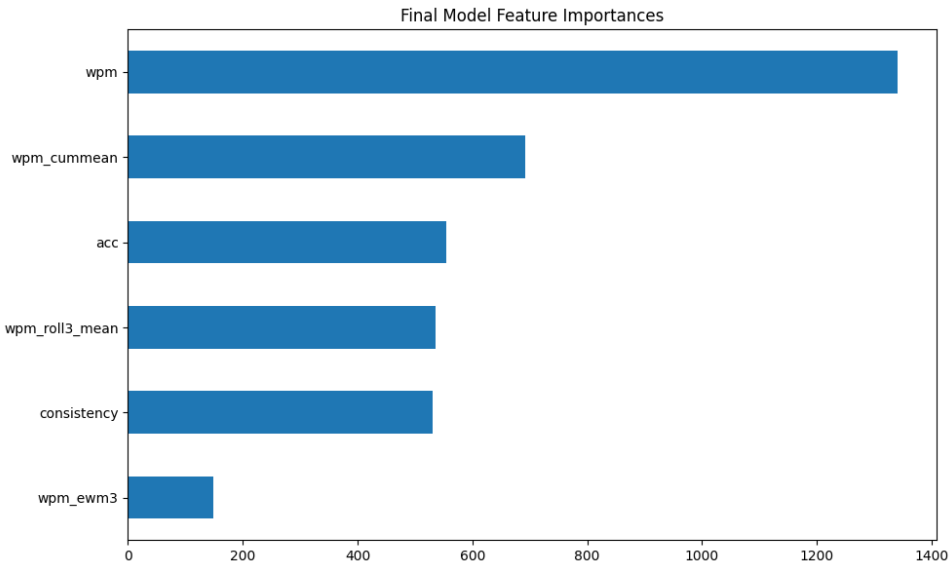

Figure 12: LGBM Classifier feature importances

The plot clearly shows that the model relies most heavily on recent performance metrics. The **`wpm`** (current session's WPM) is the single most important feature. This is followed by long-term (**`wpm_cummean`**) and short-term (**`wpm_roll3_mean`**) trends, as well as the current **`accuracy`**. This confirms that recent performance and trends are the most valuable predictors for the next session's outcome.

7 Conclusion

This project successfully developed a complete workflow for analyzing and modeling sequential typing performance logs. The primary challenge was not model selection, but problem definition. The analysis demonstrated that while predicting the exact WPM of a future session is unfeasible, predicting the **direction** of performance change is a solvable problem.

The initial **regression** approach failed. The results conclusively showed that complex models could not outperform a simple rolling-average baseline. This failure was attributed to the high-variance, ”noisy” nature of the target variable, which is likely driven by uncaptured human factors such as daily focus, fatigue, or mood.

The critical pivot to a **classification** task, predicting whether performance would ’Increase’, ’Decrease’, or ’Stable’ was highly successful. The final LightGBM model achieved **69.23%** accuracy on the test set, significantly outperforming a naive baseline. The model proved particularly effective at identifying negative performance trends, correctly predicting 89% of all ”Decrease” sessions.

The model’s limitations, such as its confusion between ”Stable” and ”Increase” sessions, are a reflection of the data itself. The signal is weak, and the iterative experiments show that we have reached the performance ceiling for this feature set. Ultimately, this project demonstrates a robust, data-driven process. Starting with a clear goal, using exploratory analysis to understand the data’s nature, and iteratively reframing the problem to achieve a meaningful and predictive result.