Analyzing Player Popularity Trends in Limbus Company via Sequences, Summations, and Linear Regression-Based Recurrence Models with Time Lags

Reynard Anderson Wijaya - 13524111
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13524111@std.stei.itb.ac.id

***Abstract*—Active player count has long been a key indicator of a game's popularity. Prior studies have shown a strong correlation between sustained player activity and overall game success. One approach to analyzing this relationship is through mathematical modeling using sequences, summation, and recurrence relations based on active player data. This paper analyzes the popularity trends of the online game Limbus Company by examining its active player data over time through the lens of these discrete mathematical tools.**

Keywords*—Sequence, Summation, Recurrence Relations, Linear Regression, Time Lag, Jupyter Notebook, Python, Limbus Company

I. Introduction

Nine out of ten individuals in the world own smartphones, which means that most individuals have access to services and experiences from the internet. Knowing this, the gaming industry has taken advantage from the increasing amount of mobile phone users and usage by adjusting to the development of mobile games, including gacha.

Gacha games are a type of game genre that uses a randomizer or gacha mechanic, similar to gachapon, a capsule toy vending machine. This mechanic is the main focus of the game as it is needed for the players progression and the developers monetization system. Some players will spend not only in-game money, but also real money in order to get virtual in-game items or characters. And, around 77% of consumers consider the convenience of the game before making transactions in-game. Given these reasons, it drives most gaming industries into making a gacha game on mobile platforms, which is accessible and can fit in easily between people's daily lives compared to other types of games that require more time and focus, and PC or console games which have limited device options.

There are other reasons for the rise of the gacha genre, such as the new era of gaming where games are not an isolated experience anymore. Players from each game share their experience in social platforms like Discord, Reddit, or X (aka. Twitter), it can be about their pulls (gacha results), character and team builds, or event discussion.

On the gaming industry side, they needed to appeal to players with unique experience or high quality visuals for every gacha they introduced, this will cause the players and fans to stay loyal to the game.

Fig 1.1 Limbus Company Steam icon

(Source: Limbus Company's Steam page)

Limbus Company is a turn-based strategy gacha game developed and published by Project Moon, a small independent studio from South Korea. Released in February 2023, the game has received very positive reviews for its unique art style, complex combat system, and deep narrative, standing out within the gacha genre despite the studio's relatively small size. However, while it enjoys a dedicated community, Limbus Company doesn't have the same massive player base as more mainstream gacha titles like Genshin Impact or Wuthering Waves.

This contrast gives an opportunity for analysis by examining its active player trends through mathematical modeling using sequences, summation, and recurrence relations. This paper aims to understand how popularity evolves in smaller-scale gacha games.

II. Base Theory

A. Sequence

A sequence is an ordered list of elements defined as a function from a set of natural numbers to a given set. In discrete mathematics, sequences are commonly used to represent values that vary with its index.

For example:

$$a_n = 2n \Rightarrow \{a_1, a_2, a_3, ...\} = \{2, 4, 6, ...\}$$

Additionally, strings can also be viewed as a sequence of characters, for example:

$$gacha \Rightarrow a_1 a_2 a_3 a_4 a_5$$

Therefore, sequences are useful for representing an ordered list of elements, which is required in various analyses.

B. Summation

Summation is the operation of adding every element from a sequence in a certain range. It is denoted using sigma (∑) notation:

$$\sum_{k=m}^{n} a_k = a_m + a_{m+1} + a_{m+2} + ... + a_n$$

Summation is important for analyses which require calculating cumulative numerical data from a certain range.

C. Recurrence Relation

A recurrence relation defines each element in a sequence based on one or more of its previous elements. An example of recurrence relation is as follows:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + c_3 a_{n-3} + ... + c_k a_{n-k}$$

where $c_1, c_2, c_3, ..., c_k$ are real numbers and $c_k \neq 0$.

Recurrence relations are useful in modeling functions from a sequential data (or sequence).

D. Linear Regression and Time Lag

The recurrence relations obtained through the analysis are represented as linear regression models with time-lagged variables. Linear regression is a mathematical model used to examine the relationships between a dependent variable and one or more independent variables. Here's a simple example of linear regression:

$$y = \alpha \cdot x + \beta$$

α is slope, representing how much x affects y. While β is intercept, showing y's starting point when $x = 0$.

In this paper's context, time lag refers to the delay between past or prior values and present or current values, where the prior ones influences the current.

This paper explores four forms of linear regression recurrence models:

1. Single-Lag

   $$a_n \approx \alpha \cdot a_{n-k} + \beta$$

   (This model shows how variable a_n is affected by a single prior variable a_{n-k} where k is the time lag)

2. 2-Lag

   $$a_n \approx \alpha \cdot a_{n-k_1} + \beta \cdot a_{n-k_2} + \gamma$$

   (This model shows how variable a_n is affected by two prior variables a_{n-k_1} and a_{n-k_2} where k_1 and k_2 are the time lags)

3. 3-Lag

   $$a_n \approx \alpha \cdot a_{n-k_1} + \beta \cdot a_{n-k_2} + \gamma \cdot a_{n-k_3} + \delta$$

   (This model shows how variable a_n is affected by two prior variables a_{n-k_1}, a_{n-k_2}, and a_{n-k_3} where k_1, k_2, and k_1 are the time lags)

4. Summation-Lag

   $$a_n \approx \alpha \cdot (\sum_{i=1}^{k} a_{n-i}) + \beta$$

   (This model shows how variable a_n is affected by the sum of *k* prior variables where *k* is the time lag)

The models shown above are impossible to reach 100% accuracy, so R-squared (R^2) is used to measure how close the real data points are to the regression line model. The formula to find R^2 is as follows:

$$R^2 = \frac{SSR}{SST} = \frac{\sum (\hat{y}_i - \bar{y})^2}{\sum (y_i - \bar{y})^2}$$

Fig 2.D.1 R^2 formula

(Source: R-squared in Linear Regression Models: Concepts, Examples[6])

$\hat{y}$ represents the prediction or a point on the regression line, y_i represents the actual values or the points, and $\bar{y}$ represents the mean of all the values. The closer the R^2 value is to 1, the greater the model's accuracy is.

E. Jupyter Notebook

Jupyter Notebook is a document that allows users to write and run code, display output, and add markdown text and visualization all in one place. It is an essential tool for data scientists to explore and analyze data in order to get insights efficiently.

While Jupyter Notebook supports multiple programming languages, Python is the most commonly used language in data science.

F. Python

Python is a versatile high-level programming language known for its simplicity and readability. Its continuous development in libraries and frameworks have broadened its applicability across numerous fields, including data science and analysis. In this paper, four open-source Python libraries and modules are used to process data and modeling:

1) Pandas

Pandas is a Python library used for manipulating and analyzing data structures or functions to perform operations efficiently in the form of DataFrame (structured table).

2) *re module*
re is a Python module used for regular expression matching operations for strings.

3) *itertools module*
itertool is a Python module used for providing iterator functions for data

4) *LinearRegression from sklearn.linear_model*
LinearRegression is a class from Python module sklearn.linear_model used for modeling a linear regression based on the relationship between one or more independent variables and a dependent variable.

III. Method

The method used in this analysis involves modeling recurrence relations based on one or more preceding elements in the sequence. This includes testing single-lag, multi-lag, and summation-lag regression models, each following a recurrence relation structure to find underlying patterns in the data through the use of Python. The goal is to identify the most accurate mathematical representation of the active player trend over time.

IV. Active Player Dataset Analysis

The dataset used in this analysis, Limbus_Company_Active_Player.csv, was taken from SteamDB on June 18, 2025. The analysis will be conducted using Jupyter Notebook with Python programming language, which provides the tools necessary for data pre-processing, model construction, and statistical evaluation. The following steps outline the procedure used to obtain the recurrence-based model from the active player dataset, mainly on the "DateTime" and "Average Players" columns. In this paper, a_{n-k} represents the average active player count from k days prior. **Finding out the correlation between a_n & each a_{n-k} and a_n & each $\sum_{i=1}^{k} a_{n-i}$ are not a required step.** It is done to give quick estimations on which time lags are more dependent. The time lag range is limited to 30 days to ensure manageable computation while still capturing a diverse set of results.

A. *Library, Module, and Class Import and Environment Setup*

1) *Library, Module, and Class Import*

```
import pandas as pd
import re
import itertools
from sklearn.linear_model import LinearRegression
```

Fig. 4.A.1 Importing required libraries, modules, and classes
(Source: Author's archive)

2) *Environment Setup*

```
# Load Limbus_Company_Active_Player.csv file
df = pd.read_csv("Limbus_Company_Active_Player.csv")

# Ensure 'Average Players' and 'DateTime' column exist
df['DateTime'] = pd.to_datetime(df['DateTime'])
df = df.sort_values('DateTime')  # sorts and ensure correct time order
players = df['Average Players']

# The lag range is limited to 30 days to ensure manageable computation
# while still capturing a diverse set of results
max_lag = 30
```

Fig. 4.A.2 Loading dataset, ensuring required columns exists in a correct format and order, and limiting time lag range
(Source: Author's archive)

B. *Lag Correlation Analysis*

1) *Calculation of Lag Correlation*

```
correlations = {} # dictionary to keep lag:correlation

# Compute correlation for each lag
for lag in range(1, max_lag + 1):
    lagged = players.shift(lag) # shifts index by lag
    corr = players.corr(lagged) # get the correlation value between current a_n with a_n-lag
    correlations[f'a_n-{lag}'] = corr # put values into dictionary
```

Fig. 4.B.1 Calculating correlation values for each lag and storing the results
(Source: Author's archive)

2) *Display Sorted Lag Correlation*

```
# Sort the dictionary of lag correlations in descending order based on the absolute value of the correlation
correlations_sorted = dict(sorted(correlations.items(), key=lambda item: abs(item[1]), reverse=True))

# Show results
for lag, corr in correlations_sorted.items():
    print(f"{lag}: correlation = {corr:.4f}")
```

Fig. 4.B.2a Sorting and displaying stored lag correlation results in descending order based on absolute correlation values
(Source: Author's archive)

```
a_n-1: correlation = 0.9657
a_n-7: correlation = 0.9518
a_n-2: correlation = 0.9493
a_n-6: correlation = 0.9388
```

Fig. 4.B.2b A portion of the top 4 highest sorted lag correlation results
(Source: Author's archive)

```
a_n-26: correlation = 0.8262
a_n-29: correlation = 0.8226
a_n-25: correlation = 0.8195
a_n-30: correlation = 0.8125
```

Fig. 4.B.2c A portion of the bottom 4 lowest sorted lag correlation results
(Source: Author's archive)

Based on the sorted lag correlation results, the strongest correlation is observed at lag a_{n-1}, followed by a_{n-7} and a_{n-2}. Conversely, the weakest correlations are at lags a_{n-29}, a_{n-25}, and a_{n-30}, with a_{n-30} being the lowest overall correlation.

C. *Single-Lag Regression Modeling*

1) *Construct Single-Lag DataFrame*

```
results = [] # holds tuples of (lag, alpha, beta, r2)

for label in correlations:
    match = re.search(r'a_n-(\d+)', label) # extract lag values from string-formatted key
    if not match:
        continue # skip if key format is invalid

    k = int(match.group(1)) # convert obtained lag values from string into number

    df_lag = pd.DataFrame({ # create dataframe
        'y': players, # y represents a_n
        f'x_{k}': players.shift(k) # x represents a_n-k
    }).dropna() # removes NaN values
```

Fig. 4.C.1 Constructing a DataFrame for each lag consisting of the current value a_n and its lagged value a_{n-k}
(Source: Author's archive)

2) *Creating Single-Lag Regression Models and Storing Its Statistical Values*

```
    X = df_lag[[f'x_{k}']].values
    y = df_lag['y'].values

    model = LinearRegression() # create linear regression model
    model.fit(X, y)

    alpha = model.coef_[0] # get model slope
    beta = model.intercept_ # get model intercept
    r2 = model.score(X, y) # get model R-squared

    # Stores tuple (lag, slope, intercept, r2)
    results.append((k, alpha, beta, r2))
```

Fig. 4.C.2 Creating regression model in the form of $a_n \approx \alpha \cdot a_{n-k} + \beta$ and storing the model's slope (α), intercept (β), and accuracy (R^2 score) for each lag
(Source: Author's archive)

3) *Display Sorted Single-Lag Regression Models*

```
# Sort the results by R-squared score in descending order
results.sort(key=lambda x: x[3], reverse=True)

# Print sorted results (top 5)
for k, alpha, beta, r2 in results[:5]:
    print(f"Lag {k:2d}: a_n ≈ {alpha:.4f} * a_(n-{k}) + {beta:.2f} | R² = {r2:.4f}")
```

Fig. 4.C.3a Sorting and displaying the top 5 stored single-lag regression model results in descending order based on R^2 score
(Source: Author's archive)

```
Lag  1: a_n ≈ 0.9651 * a_(n-1) + 581.92 | R² = 0.9326
Lag  7: a_n ≈ 0.9510 * a_(n-7) + 832.15 | R² = 0.9059
Lag  2: a_n ≈ 0.9487 * a_(n-2) + 852.65 | R² = 0.9011
Lag  6: a_n ≈ 0.9380 * a_(n-6) + 1065.79 | R² = 0.8814
Lag  5: a_n ≈ 0.9318 * a_(n-5) + 1137.65 | R² = 0.8694
```

Fig. 4.C.3b Top 5 sorted single-lag regression model results
(Source: Author's archive)

Based on the sorted single-lag regression models, the strongest single lag regression model is:

$$a_n \approx 0.9651 \cdot a_{n-1} + 581.92$$

with an R^2 score of 0.9326, indicating a strong level of accuracy.

D. *Multi-Lag Regression Modeling*

Multi-lag regression modeling is limited to 2-lag and 3-lag to ensure manageable computation while still capturing a diverse set of results.

1) *Limiting Time Lag Range*

```
lags = list(range(1, max_lag + 1)) # range of lag values from 1 to 30
results = [] # holds tuples of (combination, slope, intercept, r2)
```

Fig. 4.D.1 Limiting time lag range to 30 days
(Source: Author's archive)

2) *Construct 2-Lag DataFrame*

```
# Test all 2-lag combinations
for r in [2]:
    for combination in itertools.combinations(lags, r):
        lagged_df = pd.DataFrame({'a_n': players}) # create dataframe and a_n as dependent variable
        for k in combination:
            lagged_df[f'a_n-{k}'] = players.shift(k) # a_n-k1 and a_n-k2 as independent variable
        lagged_df = lagged_df.dropna() # removes NaN values
```

Fig. 4.D.2 Constructing a DataFrame for each lag combination consisting of the current value a_n and its lagged values a_{n-k_1} and a_{n-k_2}
(Source: Author's archive)

3) *Construct 3-Lag DataFrame*

```
# Test all 3-lag combinations
for r in [3]:
    for combination in itertools.combinations(lags, r):
        lagged_df = pd.DataFrame({'a_n': players}) # create dataframe and a_n as dependent variable
        for k in combination:
            lagged_df[f'a_n-{k}'] = players.shift(k) # a_n-k1, a_n-k2, and a_n-k3 as independent variable
        lagged_df = lagged_df.dropna() # removes NaN values
```

Fig. 4.D.3 Constructing a DataFrame for each lag combination consisting of the current value a_n and its lagged values a_{n-k_1}, a_{n-k_2}, and a_{n-k_3}
(Source: Author's archive)

4) *Creating 2-Lag & 3-Lag Regression Models and Storing Its Statistical Values*

```
        X = lagged_df.drop(columns='a_n')
        y = lagged_df['a_n']

        model = LinearRegression()
        model.fit(X, y)

        alpha = model.coef_ # get model slope
        beta = model.intercept_ # get model intercept
        r2 = model.score(X, y) # get model R-squared

        # Store tuple (combination, slope, intercept, r2)
        results.append((combination, alpha, beta, r2))
```

Fig. 4.D.4 Creating regression model in the form of $a_n \approx \alpha \cdot a_{n-k_1} + \beta \cdot a_{n-k_2} + \gamma$ for 2-lag and

$a_n \approx \alpha \cdot a_{n-k_1} + \beta \cdot a_{n-k_2} + \gamma \cdot a_{n-k_3} + \delta$ for 3-lag, and storing the model's slope (α), intercept (β), and accuracy (R^2 score) for each lag combination
(Source: Author's archive)

5) *Display Sorted 2-Lag and 3-Lag Regression Models*

```
# Sort the results by r2 score in descending order
results.sort(key=lambda x: x[3], reverse=True)

# Print sorted results (top 5)
for combination, coefs, intercept, r2 in results[:5]:
    terms = " + ".join([f"{coef:.4f}·a_(n-{lag})" for coef, lag in zip(coefs, combination)])
    print(f"Combination {combination}: a_n ≈ {terms} + {intercept:.2f} | R² = {r2:.4f}")
```

Fig. 4.D.5a Sorting and displaying the top 5 stored 2-lag and 3-lag regression model results in descending order based on R^2 score
(Source: Author's archive)

```
Combination (1, 7): a_n ≈ 0.6105·a_(n-1) + 0.3792·a_(n-7) + 183.06 | R² = 0.9512
Combination (1, 14): a_n ≈ 0.7108·a_(n-1) + 0.2802·a_(n-14) + 174.49 | R² = 0.9474
Combination (1, 5): a_n ≈ 0.7176·a_(n-1) + 0.2687·a_(n-5) + 231.90 | R² = 0.9442
Combination (1, 6): a_n ≈ 0.7067·a_(n-1) + 0.2781·a_(n-6) + 259.47 | R² = 0.9438
Combination (1, 13): a_n ≈ 0.7880·a_(n-1) + 0.1968·a_(n-13) + 269.39 | R² = 0.9409
```

Fig. 4.D.5b Top 5 sorted 2-lag regression model results
(Source: Author's archive)

```
Combination (1, 14, 15): a_n ≈ 0.8746·a_(n-1) + 0.6886·a_(n-14) + -0.5764·a_(n-15) + 225.30 | R² = 0.9652
Combination (1, 7, 8): a_n ≈ 0.7450·a_(n-1) + 0.5944·a_(n-7) + -0.3529·a_(n-8) + 228.67 | R² = 0.9574
Combination (1, 28, 29): a_n ≈ 0.9245·a_(n-1) + 0.5861·a_(n-28) + -0.5262·a_(n-29) + 267.77 | R² = 0.9570
Combination (1, 7, 15): a_n ≈ 0.6859·a_(n-1) + 0.4733·a_(n-7) + -0.1770·a_(n-15) + 307.59 | R² = 0.9549
Combination (1, 14, 22): a_n ≈ 0.7515·a_(n-1) + 0.4549·a_(n-14) + -0.2274·a_(n-22) + 367.66 | R² = 0.9542
```

Fig. 4.D.5c Top 5 sorted 3-lag regression model results
(Source: Author's archive)

Based on the sorted multi-lag (2-lag and 3-lag) regression models, the strongest 2-lag regression model is:

$$a_n \approx 0.6105 \cdot a_{n-1} + 0.3792 \cdot a_{n-7} + 183.06$$

with an R^2 score of 0.9512.

While the strongest 3-lag regression model is:

$$a_n \approx 0.8746 \cdot a_{n-1} + 0.6886 \cdot a_{n-14} - 0.5764 \cdot a_{n-15} + 225.3$$

with an R^2 score of 0.9652.

Both models indicate a strong level of accuracy.

E. *Summation-Lag Correlation Analysis*

1) *Calculation of Lag Summation*

```
correlation_sums = {} # dictionary to keep sum_lag:correlation

# Compute correlation for each lag summation
for lag in range(2, 31):
    sum_lag = players.copy()
    for i in range(1, lag+1):
        sum_lag += players.shift(i) # calculate summation by lag amount

    # Create dataframe
    sum_players = pd.DataFrame({
        'a_n': players, # a_n as dependent variable
        'sum_lag': sum_lag # the sum from a_n-1 to a_n-lag as independent variable
    }).dropna() # removes NaN values
```

Fig. 4.E.1 Calculating each lag summations and storing the results in a DataFrame
(Source: Author's archive)

2) *Calculation of Summation-Lag Correlation*

```
    # get the correlation value between current a_n with sum_lag
    corr = sum_players['a_n'].corr(sum_players['sum_lag'])
    # put values into dictionary
    correlation_sums[f'sum_lag_{lag}'] = corr
```

Fig. 4.E.2 Calculating correlation values for each summation and storing the results
(Source: Author's archive)

3) *Display Sorted Summation-Lag Correlation*

```
# Sort the dictionary of summation-lag correlations in descending order based on the absolute value of the correlation
sorted_corr = dict(sorted(correlation_sums.items(), key=lambda x: abs(x[1]), reverse=True))

# Show results
for label, corr in sorted_corr.items():
    print(f"{label}: correlation = {corr:.4f}")
```

Fig. 4.E.3a Sorting and displaying stored summation-lag correlation results in descending order based on absolute correlation values
(Source: Author's archive)

```
sum_lag_2: correlation = 0.9849
sum_lag_3: correlation = 0.9785
sum_lag_4: correlation = 0.9740
sum_lag_7: correlation = 0.9733
```

Fig. 4.E.3b A portion of the top 4 highest sorted summation-lag correlation results
(Source: Author's archive)

```
sum_lag_27: correlation = 0.9356
sum_lag_28: correlation = 0.9347
sum_lag_29: correlation = 0.9330
sum_lag_30: correlation = 0.9310
```

Fig. 4.E.3c A portion of the bottom 4 lowest sorted summation-lag correlation results
(Source: Author's archive)

Based on the sorted summation-lag correlation results, the strongest correlation is observed at $\sum_{i=1}^{2} a_{n-i}$, followed by $\sum_{i=1}^{3} a_{n-i}$ and $\sum_{i=1}^{4} a_{n-i}$ Conversely, the weakest correlations are at $\sum_{i=1}^{28} a_{n-i}$, $\sum_{i=1}^{29} a_{n-i}$, and $\sum_{i=1}^{30} a_{n-i}$, with $\sum_{i=1}^{30} a_{n-i}$ being the lowest overall correlation.

F. *Summation-Lag Regression Modeling*

1) *Construct Summation-Lag DataFrame*

```
summation_models = [] # holds tuples of (lag, alpha, beta, r2)

for k in range(2, 31):
    sum_lag = players.copy()
    for i in range(1, k+1):
        sum_lag += players.shift(i) # calculate summation by lag amount

    # Create dataframe
    df_sum = pd.DataFrame({
        'a_n': players, # a_n as dependent variable
        'sum_k': sum_lag # the sum from a_n-1 to a_n-lag as independent variable
    }).dropna() # removes NaN values
```

Fig. 4.F.1 Constructing a DataFrame for each summation-lag consisting of the current value a_n and its summation-lagged value $\sum_{i=1}^{k} a_{n-i}$

(Source: Author's archive)

This whole section (except creating a tuple) follows the exact same method as IV.E.1.

2) *Creating Summation-Lag Regression Models and Storing Its Statistical Values*

```
    X = df_sum[['sum_k']].values
    y = df_sum['a_n'].values

    model = LinearRegression() # create linear regression model
    model.fit(X, y)

    alpha = model.coef_[0] # get model slope
    beta = model.intercept_ # get model intercept
    r2 = model.score(X, y) # get model accuracy

    # Store tuple (k, slope, intercept, r2)
    summation_models.append((k, alpha, beta, r2))
```

Fig. 4.F.2 Creating regression model in the form of $a_n \approx \alpha \cdot (\sum_{i=1}^{k} a_{n-i}) + \beta$ and storing the model's slope (α), intercept (β), and accuracy (R^2 score) for each summation-lag

(Source: Author's archive)

3) *Display Sorted Summation-Lag Regression Models*

```
# Sort the results by R-squared score in descending order
summation_models.sort(key=lambda x: x[3], reverse=True)

# Print sorted results (top 5)
for k, alpha, beta, r2 in summation_models[:5]:
    print(f"k = {k:2d}: a_n ≈ {alpha:.4f} * SUM(a_(n-1) to a_(n-{k})) + {beta:.2f} | R² = {r2:.4f}")
```

Fig. 4.F.3a Sorting and displaying the top 5 stored summation-lag regression model results in descending order based on R^2 score

(Source: Author's archive)

```
k =  2: a_n ≈ 0.3327 * SUM(a_(n-1) to a_(n-2)) + 38.06 | R² = 0.9701
k =  3: a_n ≈ 0.2488 * SUM(a_(n-1) to a_(n-3)) + 80.74 | R² = 0.9574
k =  4: a_n ≈ 0.1988 * SUM(a_(n-1) to a_(n-4)) + 104.90 | R² = 0.9487
k =  7: a_n ≈ 0.1247 * SUM(a_(n-1) to a_(n-7)) + 52.02 | R² = 0.9473
k =  5: a_n ≈ 0.1658 * SUM(a_(n-1) to a_(n-5)) + 97.45 | R² = 0.9457
```

Fig. 4.F.3b Top 5 sorted summation-lag regression model results

(Source: Author's archive)

Based on the sorted summation-lag regression models, the strongest summation-lag regression model is:

$$a_n \approx 0.3327(\sum_{i=1}^{2} a_{n-i}) + 38.06$$

with an R^2 score of 0.9701, indicating a strong level of accuracy.

V. Conclusion

Linear regression-based recurrence models cand be used to find a fitting mathematical model representation of Limbus Company's average active player trend. From the analyses done, three types of regression models were formed. Among them, the best and most accurate model is:

$$a_n \approx 0.3327(\sum_{i=1}^{2} a_{n-i}) + 38.06$$

with an R^2 score of 0.9701. From this model, it can be concluded that the trend of Limbus Company's active player count is rising, as the model adds positive values from the previous 2 days. However, it has a slow growth rate, indicated by the low coefficient of 0.3327.

Further optimization will be necessary for future model tuning, as the Limbus_Company_Active_Player.csv dataset will continue to grow over time while the game remains in active service.

VI. Attachment

Source code used to analyze the data: https://github.com/RND815/Analysis-Limbus-Company-Active-Player-Trend

VII. Acknowledgement

The author would like to first and foremost, thank God for all the guidance He has provided throughout the process of researching, learning, and writing this paper. The author also wishes to thank the lecturer of ITB Discrete Mathematics IF1220, Mr. Rinaldi Munir, for sharing his knowledge and guiding the students throughout the class. Lastly, the author expresses his gratitude to his family and friends for their continuous support during the semester and the completion of this paper.

References

[1] L. Gu and A. L. Jia. "Player activity and popularity in online social games and their Implications for player retention," 2018 16th Annual Workshop on Network and Systems Support for Games (NetGames), Amsterdam, Netherlands, pp. 1-6, June 2018.

[2] Jones, M. "Why are gacha games becoming so popular?" GameSpace, 16 June 2025. https://gamespace.com/all-articles/news/why-are-gacha-games-becoming-so-popular/. [Accessed on 17 June 2025]

[3] Munir, Rinaldi. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025-2/matdis24-25-2.htm. [Accessed on 18 June 2025]

[4] https://steamdb.info/app/1973530/charts/#1y. [Accessed on 18 June 2025]

[5] S. Vikram. “Difference between linear and multiple regression,” Shiksha, 16 September 2024. https://www.shiksha.com/online-courses/articles/linear-and-multiple-regression/. [Accessed on 18 June 2025]

[6] K. Ajitesh. “R-squared in linear regression models: concepts, examples,” Vitalflux, 13 November 2023. https://vitalflux.com/r-squared-explained-machine-learning/. [Accessed on 19 June 2025]

[7] Macro Pulse. “Understanding time lags and their effects in regression analysis: A deep dive into supervised learning,” Medium, 28 October 2024. https://medium.com/@jangdaehan1/understanding-time-lags-and-their-effects-in-regression-analysis-a-deep-dive-into-supervised-b9394df25755. [Accessed on 19 June 2025]

[8] P. Benjamin. “How to use jupyter notebook: a beginner’s tutorial,” Dataquest, 28 January 2025. https://www.dataquest.io/blog/jupyter-notebook-tutorial/#:~:text=Jupyter%20Notebook%20is%20an%20incredibly,a%20cohesive%20and%20expressive%20workflow. [Accessed on 19 June 2025]

[9] W. Summer. “What is python? Everything you need to know to get started,” DataCamp, 30 July 2024. https://www.datacamp.com/blog/all-about-python-the-most-versatile-programming-language?utm_source=google&utm_medium=paid_search&utm_campaignid=19589720824&utm_adgroupid=152984014214&utm_device=c&utm_keyword=&utm_matchtype=&utm_network=g&utm_adpostion=&utm_creative=733936254692&utm_targetid=aud-1700705940199:dsa-2222697811398&utm_loc_interest_ms=&utm_loc_physical_ms=1007700&utm_content=ps-other~apac-en~dsa~tofu~blog-python&accountid=9624585688&utm_campaign=230119_1-ps-other~dsa~tofu_2-b2c_3-apac_4-prc_5-na_6-na_7-le_8-pdsh-go_9-nb-e_10-na_11-na&gad_source=1&gad_campaignid=19589720824&gbraid=0AAAAADQ9WsG4lEdji1zf3XxT7hSgi6vza&gclid=CjwKCAjw6s7CBhACEiwAuHQckkUzjTnIfBjhOiDFr8maxTkuGH7M8gC9ec2CqUGIiLMYGCZ41xoeExoC-HYQAvD_BwE. [Accessed on 19 June 2025]

[10] “Pandas introduction,” GeeksforGeeks, 12 May 2025. https://www.geeksforgeeks.org/pandas/introduction-to-pandas-in-python/. [Accessed on 19 June 2025]

[11] Python Software Foundation. “re — regular expression operations,” Python 3.13.5 documentation, 2024. https://docs.python.org/3/library/re.html. [Accessed on 19 June 2025]

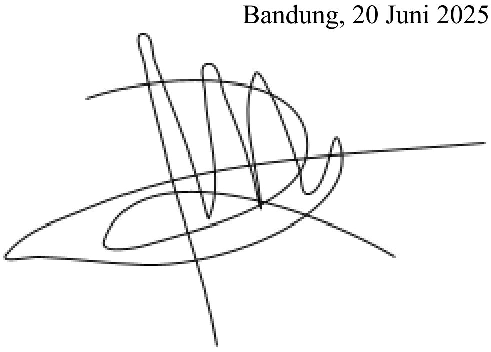
[12] Python Software Foundation. “itertools — functions creating iterators for efficient looping,” Python 3.13.5 documentation, 2024. https://docs.python.org/3/library/itertools.html. [Accessed on 19 June 2025]

[13] scikit-learn developers. “LinearRegression,” scikit-learn 1.7.0, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html. [Accessed on 19 June 2025]

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Juni 2025



Reynard Anderson Wijaya
13524111