I have now implemented getting the game data and determined the structure of my data in Python and have the following fields in my Dataframe:

Weekly Aggregated Game Features:

Total Features (the net/gross/magnitude values of our features, from which our incremental features are derived using the daily change in these features)

'revenue_total': total game revenue at the end of the week
'sales_total': total game sales at the end of the week
'reviews_total': total game reviews at the end of the week
'followers_total': total game followers at the end of the week
'wishlists_total': total game wishlists at the end of the week
'avgPlaytime_total': the average total time a player has spent in game since the game's launch, measured at the end of the week
'score_total': the game's cumulative review score at the end of the week
'days_since_release_total': the number of days since the game's release
'price_total': the price of the game at the end of the week

Incremental Features (approximating the distribution of daily changes throughout the week)

'incremental_reviews_mean': average of the daily change in review count
'players_mean': average of the daily player count (interpretation: how many players were active per day on average throughout the week)
'incremental_sales_mean': average of the daily change in game sales
'incremental_revenue_mean': average of the daily change in game revenue
'incremental_followers_mean': average of the daily change in follower count
'incremental_wishlists_mean': average of the daily change in wishlist count
'incremental_avgPlaytime_mean': average of the daily change in players' average playtime (interpretation: approximates the average time a player spent in game each day during the week)

Weekly Aggregated Cross-Genre Features:

Total Features (the net/gross/magnitude values of our features across an entire genre)

'genre_revenue_mean': the genre's average total game revenue at the end of the week
'genre_revenue_std': the standard deviation of games within the genre's total game revenue
'genre_sales_mean': the genre's average total game sales at the end of the week
'genre_sales_std': the standard deviation of games within the genre's total game sales
'genre_reviews_mean': the genre's average total game reviews at the end of the week
'genre_reviews_std': the standard deviation of games within the genre's total game reviews
'genre_followers_mean': the genre's average total game followers at the end of the week
'genre_followers_std': the standard deviation of games within the genre's total game followers
'genre_wishlists_mean': the genre's average total game wishlists at the end of the week
'genre_wishlists_std': the standard deviation of games within the genre's total game wishlists

'genre_avgPlaytime_mean': the genre's average of the average total time a player has spent in game since the game's launch, measured at the end of the week
'genre_avgPlaytime_std': the standard deviation of games within the genre's average total time a player has spent in game since the game's launch
'genre_score_mean': the genre's average cumulative review score at the end of the week
'genre_score_std': the standard deviation of games within the genre's cumulative review score
'genre_days_since_release_mean': the genre's average number of days since the game's release
'genre_days_since_release_std': the standard deviation of games within the genre's number of days since the game's release
'genre_price_mean': the genre's average game purchase price at the end of the week
'genre_price_std': the standard deviation of games within the genre's purchase price
Incremental Features (approximating the distribution of daily changes throughout the week - note these features are exactly the same as for an individual game, but averaged over the entire genre)
'genre_incremental_reviews_mean': genre average of the daily change in review count
'genre_incremental_reviews_std': genre standard deviation of daily change in review count
'genre_players_mean': genre average of the daily player count (interpretation: across the genre, how many players were active per day on average throughout the week)
'genre_players_std': genre standard deviation of the daily player count
'genre_incremental_sales_mean': genre average of the daily change in game sales
'genre_incremental_sales_std': genre standard deviation of daily change in review count
'genre_incremental_revenue_mean': genre average of the daily change in game revenue
'genre_incremental_revenue_std': genre standard deviation of daily change in game revenue
'genre_incremental_followers_mean': genre average of the daily change in follower count
'genre_incremental_followers_std': genre standard deviation of daily change in follower count
'genre_incremental_wishlists_mean': genre average of the daily change in wishlist count
'genre_incremental_wishlists_std': genre standard deviation of daily change in wishlist count
'genre_incremental_avgPlaytime_mean': genre average of the daily change in players' average playtime (interpretation: approximates the genre-wide average amount of time a player spent in game each day during the week)
'genre_incremental_avgPlaytime_std': genre standard deviation of daily change in players' average playtime.

Game Engagement Metrics

Player Growth Rate

- Calculation: Weekly percentage change in average daily players
- Rationale: Direct indicator of game popularity trends; growing player base suggests potential revenue growth
- Code: player_growth = game_history['players_mean'].pct_change()

Quality-Adjusted Engagement

- Calculation: Average daily players weighted by average daily playtime this week and review score
- Rationale: Measures not just quantity but quality of engagement; a better predictor of sustainable revenue
- Code: quality_engagement = (game_history['players_mean'] * game_history['incremental_avgPlaytime_mean'] * (game_history['score_total'] / 100))

Sentiment Metrics

Weighted Sentiment Momentum

- Calculation: Change in review score weighted by log of review volume
- Rationale: Captures changing customer satisfaction grounded by the number of reviews, since the fewer reviews there are, the larger the impact of individual reviews
- Code: weighted_sentiment = (game_history['score_total'] * np.log(game_history['reviews_total']))
  sentiment_momentum = weighted_sentiment.pct_change()

Sentiment Divergence

- Calculation: Difference between short-term and long-term player sentiment trends
- Rationale: Identifies emerging positive or negative shifts in customer satisfaction away from the expected historical trend
- Code: sentiment_divergence = (game_history['score_total'].rolling(4).mean() - game_history['score_total'].rolling(8).mean())

Monetization Metrics

Revenue Per Active Player Hour

- Calculation: Weekly revenue growth divided by this week's average player count weighted by average playtime this week
- Rationale: Measures monetization efficiency; higher values suggest better revenue extraction from player activity
- Code: revenue_extraction_efficiency = game_history['revenue_total'].diff() / (game_history['players_mean']*game_history['avgPlaytime_total']))

Wishlist Conversion Rate

- Calculation: New sales divided by the previous week's wishlist count
- Rationale: Indicates marketing effectiveness and purchase intent conversion as wishlisted games get removed from players' wishlists once purchased
- Code: (game_history['sales_total'].diff() / game_history['wishlists_total'].shift(1))

Lifecycle-Adjusted Metrics

Retention Strength

- Calculation: Actual average daily players divided by expected daily players (previous week's total sales divided by square root of days since release)
- Rationale: Measures how well a game maintains its player base relative to its age and sales. Controls for natural player drop-off over time by using the square root of a game's age to decay total sales, serving as a model for our expected active player base
- Code: retention_strength = game_history['players_mean'] / (game_history['sales_total'] / (1 + np.sqrt(game_history['days_since_release'])))

Revenue Sustainability

- Calculation: Actual average daily revenue change divided by expected average daily revenue (revenue total/days since release squared)
- Rationale: Evaluates a game's revenue generation ability relative to its age and cumulative revenue. Controls for natural revenue drop-off over time by using the game's cumulative revenue divided by the game's age squared, serving as a model for our expected revenue for the game age and historical revenue
- Code: revenue_sustainability = game_history['incremental_revenue_mean'] / (game_history['revenue_total']/game_history['days_since_release_total']**2)

Stability Metrics

Player Base Stability

- Calculation: Coefficient of variation of daily player count (standard deviation divided by mean)
- Rationale: Measures consistency of player engagement; lower values suggest more predictable player behaviour and potentially more reliable revenue
- Code: player_stability = game_history['players_std'] / game_history['players_mean']

Revenue Predictability

- Calculation: 4-week rolling coefficient of variation of daily revenue changes
- Rationale: Indicates reliability of revenue streams; stable revenue might suggest better future performance predictability
- Code: revenue_predictability = game_history['incremental_revenue_std'] / game_history['incremental_revenue_mean'].rolling(4).mean()

Leading Indicator Metrics

Social Momentum

- Calculation: Difference between short-term and long-term follower trends normalized by total followers

- Rationale: Measures acceleration or deceleration in social interest; could predict future player base changes
- Code:

```
short_term_growth = (
      game_history['followers_total'].pct_change(short_window)
      .rolling(window=short_window).mean()
  )
  long_term_growth = (
      game_history['followers_total'].pct_change(long_window)
      .rolling(window=long_window).mean()
  )
  # Compare short-term vs long-term growth
  social_momentum = (short_term_growth - long_term_growth)
  # Normalize by total followers for cross-sectional comparison
  normalized_momentum = social_momentum / np.log1p(game_history['followers_total'])
```

Purchase Intent Momentum

- Calculation: Difference between short-term and long-term follower trends normalized by total followers
- Rationale: Indicates acceleration or deceleration in purchase interest; might predict future sales trends
- Code:

```
short_term_growth = (
      game_history['wishlists_total'].pct_change(short_window)
      .rolling(window=short_window).mean()
  )
  long_term_growth = (
      game_history['wishlists_total'].pct_change(long_window)
      .rolling(window=long_window).mean()
  )
  # Compare short-term vs long-term growth
  purchase_momentum = (short_term_growth - long_term_growth)
  # Normalize by total wishlists for cross-sectional comparison
  normalized_momentum = purchase_momentum / np.log1p(game_history['wishlists_total'])
```

Value Metrics

Quality-Adjusted Value

- Calculation: (Review score × Average total playtime) / Price
- Rationale: Measures value proposition to players; higher values might indicate stronger competitive position
- Code: value_ratio = (game_history['score_total'] * game_history['avgPlaytime_total']) / game_history['price_total']

Engagement Efficiency

- Calculation: Geometric mean of daily playtime change divided by price and normalized average daily sales growth
- Rationale: Measures market-validated engagement value by combining how much new playtime a game generates per dollar with corresponding growth in sales. Higher values indicate strong content value which is being validated by the market through new purchases. This helps identify games that retain and engage existing players and attract new ones, suggesting sustainable value creation and growth potential. High engagement alone could be from a small loyal playerbase and high sales alone could be from marketing/discounts. High scores in both suggest genuine value creation
- Code: engagement_efficiency = game_history['incremental_avgPlaytime_mean'] / game_history['price_total']

Market Competition Metrics

FOR ALL THE ABOVE FEATURES, WE CALCULATE THE AVERAGE AND STANDARD DEVIATION OF THAT FEATURE ACROSS THE GAME'S AUDIENCEOVERLAP AND GENRE, THEN CALCULATE A Z-SCORE AS FOLLOWS:

Genre Relative {feature name}

- Calculation: Game's **{feature name}** subtracted by the average **{feature name}** of genre peers and normalized (divided) by the standard deviation of **{feature name}** in the genre. (Essentially calculating the game's player growth z-score relative to the genre)
- Rationale: Measures relative performance vs competition; controls for genre-wide trends
- Code: (game_history['**{feature name}**'] - genre_history['**{feature name}**'])/genre_history['**{feature name}**']

Portfolio Level Metrics

Weighted Portfolio Playerbase Momentum

- Calculation: For each game at each date, we multiply the game's revenue (as a weight) by the game's player growth rate. We then sum these products across all games and divide by the total revenue
- Rationale: Player growth is often a leading indicator of revenue and by weighting by revenue we focus on player changes in games that matter most financially. Revenue-weighted growth rate may provide early signals of revenue momentum shifts, and capture these changes before they fully impact the publisher's financials.

Weighted Portfolio Engagement Quality

- Calculation: For each game at each date, we multiply the game's revenue (as a weight) by the game's quality-adjusted engagement, normalized by price. We then sum these products across all games and divide by the total revenue
- Rationale: This aims to measure how effectively the publisher creates engaging content, and how well that engagement aligns with their pricing. Price normalization helps compare engagement across different price tiers to provide insight into the publisher's value creation efficiency relative to their pricing power.

Weighted Portfolio Lifecycle Efficiency

- Calculation: For each game at each date, we multiply the game's revenue (as a weight) by the game's retention strength and revenue sustainability. We then sum these products across all games and divide by the total revenue
- Rationale: We combine two key underlying metrics, both normalized for game age which seek to measure the publisher's ability to maintain player bases against natural time decay, convert retained players into sustained revenue and manage games across different lifecycle stages. Revenue weighting ensures focus on economically important games regardless of lifecycle stage.

Genre Herfindahl Index

- Calculation: Sum of squared genre revenue shares
- Rationale: Measures genre diversification; lower values suggest better diversified portfolio
- Code:

```
    def genre_concentration(publisher_games):
        game_revenues = publisher_games.groupby('genre')['revenue_total'].sum()
        total_revenue = genre_revenues.sum()
        return ((genre_revenues / total_revenue) ** 2).sum()
```

Game Revenue Concentration

- Calculation: Sum of squared game revenue shares (Herfindahl Index) within publisher's portfolio
- Rationale: Measures game diversification; lower values suggest better diversified revenue streams
- Code:

```
        def genre_concentration(publisher_games):
game_revenues = publisher_games.groupby('game')['revenue_total']
total_revenue = game_revenues.sum()
return ((genre_revenues / total_revenue) ** 2).sum()
```

FOR AGGREGATING ALL OTHER FEATURES ACROSS GENRES:

I've developed a framework for aggregating game-level metrics into publisher-level features for stock price prediction. The system transforms individual game performance metrics into portfolio-wide indicators while maintaining economic relevance and managing feature space dimensionality.

1. Data Structure:

1. Input:
- List of Game objects, each containing:
  - Basic metadata (name, genres, tags, release_date)
  - Historical feature DataFrame with 55 metrics per game including:
    - Base metrics (14 features like player_growth_rate, retention_strength)
    - Genre-relative metrics (19 features measuring performance vs genre peers)
    - Audience-overlap-relative metrics (19 features measuring performance vs similar games)

2. Output:
- Publisher-level DataFrame containing:
  - Revenue-weighted means and standard deviations for all features
  - Age-adjusted means for base features
  - Outperformance ratios for relative features

Key Feature Categories:

1. Revenue-Weighted Means/Deviations:
- Purpose: Capture central tendency and dispersion of metrics weighted by economic importance
- Application: Applied to all features (base and relative)
- Properties:
  - More weight to revenue-significant games
  - Handles missing data appropriately
  - Maintains feature interpretability

2. Age-Adjusted Means:
- Purpose: Balance current performance with historical context
- Application: Applied to base features only
- Properties:
  - Uses sqrt(age) decay for smooth transition
  - Combines revenue and age weighting
  - Emphasizes recent performance while maintaining mature game relevance

3. Outperformance Ratios:
- Purpose: Measure competitive success across portfolio
- Application: Applied to relative features only
- Properties:
  - Proportion of revenue from games beating peers
  - Binary classification (>0 = outperforming)
  - Indicates competitive positioning

Implementation Details:

1. Feature Generation Flow:

```python
generate_publisher_aggregate_features(games, dates)
├── For all features:
│   ├── calculate_revenue_weighted_mean()
│   └── calculate_revenue_weighted_std()
├── For base features:
│   └── calculate_age_weighted_mean()
└── For relative features:
    └── calculate_outperformance_ratio()
```

2. Key Design Decisions:
- Revenue weighting prioritizes economically significant signals
- Age adjustment uses sqrt decay to maintain mature game relevance
- Outperformance uses binary threshold at zero for relative metrics
- Consistent handling of missing data across all aggregations

3. Feature Naming Convention:
- Revenue-weighted means: portfolio_{feature}_mean
- Revenue-weighted std: portfolio_{feature}_deviation
- Age-adjusted means: portfolio_{feature}_age_adjusted
- Outperformance ratios: portfolio_{feature}_strength

Current Status:

- Implemented and tested all core aggregation functions
- Created comprehensive feature generation pipeline
- Validated economic rationale for each feature type
- Established proper handling of missing data and edge cases