Casino-player-similarity-report

Abstract

I synthesized a comprehensive casino session dataset and embedded latent archetypes reflecting distinct player behaviors. I assessed data quality by examining descriptive statistics, distribution shapes, skewness, and kurtosis to ensure realistic patterns. To identify similar players, I trained an **LSTM-based sequence model** to generate 64-dimensional embeddings and measured likeness using cosine similarity. I demonstrated this approach by detecting likely churners and flagging players exhibiting high-risk gambling behaviors.

Data synthesis

The generative rules and snippets are shown in the table 1.

Column	***Approach***	***Python Code Snippet***
playerId	100 unique player IDs	player_ids = [f"{str(i).zfill(3)}" for i in range(num_players)]
day	Random date from 2025-04-19 to 2025-05-19	day = start_date + timedelta(days=random.randint(0, date_range))
timeOfDay	Random time in 24h format with uniform distribution	time_of_day = f"{random.randint(0, 23):02d}:{random.randint(0, 59):02d}:{random.randint(0, 59):02d}"

gameTitle	Random pick from 10 themed game titles	game_titles = [f"Mystic Quest {i}" for i in range(1, 11)] game_title = random.choice(game_titles)
timeOnDevice	Uniform within range; varies by archetype (e.g., high rollers get longer times)	time_on_device = np.random.uniform(900, 3600) *(for high rollers)*
netWin	90% chance positive (house wins), 10% negative (player wins), scaled to coinIn	net_win = -np.random.uniform(1, coin_in * 0.5) *(if win)* net_win = np.random.uniform(0, coin_in * 0.3) *(otherwise)*
coinIn	Uniform within range; varies by archetype	coin_in = np.random.uniform(500, 1500) *(for high rollers)* coin_in = np.random.uniform(1,500) (otherwise)
ageGroup	Categorical with weighted probabilities (e.g., 25–34 = 40%)	age_group = np.random.choice(age_groups, p=age_group_probs) age_group_probs =
distance	Concatenated from 3 Gaussian clusters and clipped to 0–100km	distance_clusters = [np.random.normal(...)] distances = np.clip(np.concatenate(...), 0, 100)
handlePulls	Poisson-distributed,	handle_pulls = np.random.poisson(100)

clipped to [1, 150], λ depends on archetype	handle_pulls = int(np.clip(handle_pulls, 1, 150))

Table 1. The synthesis approach.

Latent features

I defined three hidden player archetypes, each with distinct feature patterns:

Weekend High Rollers – I give them higher coinIn ranges, longer timeOnDevice, and bias their session dates toward Fridays through Sundays as well as evening hours.

Local Regulars – I assign them small coinIn, shorter timeOnDevice, low distance values, and concentrate their visits on weekday hours.

Lucky Streakers – I let them play at random times with fewer total sessions, but give them a higher 10% chance of negative netWin (i.e., player wins).

Archetype	**Encoded Feature Behavior**
High Roller	High coinIn, long timeOnDevice, prefers evening/weekend
Local Regular	Small coinIn, short timeOnDevice, visits often on weekdays
Lucky Streaker	Random time, rare sessions, higher chance of negative netWin (player wins)

Table 2. Latent player archetypes

To add **non-linearity**, I inject a burst-bet pattern into about 15% of all sessions: these sessions combine very high coinIn with very low timeOnDevice, simulating impulsive, high-stakes bets. I also build session-sequence dependencies: each player alternates consistently between two favorite games, and I implement loss-chasing behavior by increasing coinIn after a recent average net loss.

Data quality evaluation

The synthetic dataset comprises 100 players and 10 000 session records. Each record contains:

1. **Demographics & context**: playerId, ageGroup, distance
2. **Temporal signals**: day, timeOfDay (extracted as hour)
3. **In-game behavior**: gameTitle (encoded as gameCode), handlePulls, coinIn, netWin, timeOnDevice

Data Quality Assessment

I computed standard descriptive statistics (count, mean, standard deviation, min, 25th/50th/75th percentiles, max) and measured each feature's skewness and excess kurtosis:

coinIn (10 000 obs):

– Mean ≈ $511, SD ≈ $506, IQR [55.6, 954.4], range [14.35, 2209.33]; skewness ≈ 0.66; kurtosis ≈ –0.64

netWin (10 000 obs):

– Mean ≈ $66, SD ≈ $137, IQR [5.27, 133.48], range [–716.94, 447.85]; skewness ≈ –0.63; kurtosis ≈ 4.54

timeOnDevice (10 000 obs):

– Mean ≈ 1 197 s, SD ≈ 1 034 s, IQR [402, 2056.5], range [60, 3598]; skewness ≈ 0.92; kurtosis ≈ –0.54

handlePulls (10 000 obs):

– Mean ≈ 52, SD ≈ 37, IQR [20, 94], range [7, 141]; skewness ≈ 0.51; kurtosis ≈ –1.43

distance (10 000 obs):

– Mean ≈ 34 km, SD ≈ 30.7 km, IQR [6.8, 42.68], range [0, 100]; skewness ≈ 0.93; kurtosis ≈ –0.47

hour (10 000 obs):

– Mean ≈ 11.5, SD ≈ 6.9, IQR [6, 18], range [0, 23]; skewness ≈ –0.01; kurtosis ≈ –1.20

gameCode (10 000 obs):

– Mean ≈ 0.50, SD ≈ 0.50, IQR [0, 1], range [0, 1]; skewness ≈ 0.01; kurtosis ≈ –2.00

ageCode (10 000 obs):

– Mean ≈ 2.15, SD ≈ 1.43, IQR [1, 3], range [0, 5]; skewness ≈ 0.67; kurtosis ≈ –0.66

Plausibility Checks

1. **coinIn**'s moderate right skew and long tail match a mix of low-stake locals and occasional high rollers.
2. **netWin**'s negative skew and high positive kurtosis reflect frequent small casino profits and rare large player wins.
3. **timeOnDevice** is right-skewed—most sessions are short, but a tail of long stays appears.
4. **handlePulls** clusters at low counts with a heavy tail, consistent with Poisson sampling.
5. **distance** shows local and regional clusters plus outliers, matching the Gaussian-mixture design.
6. **hour** is nearly uniform overall, since only high rollers are evening-biased.
7. **gameCode** distribution confirms no single game dominates.
8. **ageCode** skew toward younger brackets reflects the weighted age-group sampling.

Additional plausibility check

- **Bet-size vs. pulls**: A scatter of handlePulls vs. coinIn reveals three clusters—two linear archetype trends plus a non-linear burst-bet group.
- **Behavioral clusters**: A t-SNE embedding of session features separates players into three clusters, validating the hidden archetypes.
- **Loss-chasing signal**: Comparing average coinIn after wins vs. losses shows a mild increase (due to the mild multiplier) of coinIn after loss.

Together, these statistics and distributional checks confirm that the synthetic data faithfully captures both the intended ranges and the embedded latent behaviors.

Solution Formulation

Because I need to capture both intra-session timing (weekend vs. weekday, hour-of-day) and inter-session ordering, **sequence models like LSTM, Transformer or RNN are ideal,** as they all produce fixed-length embeddings per player. Here I'll illustrate with an LSTM (since RNN and transformer are similar).

To answer the questions, I chose a **sequence model (e.g. LSTM) over generic clustering** because casino session data is inherently temporal and ordered—static methods like k-means or DBSCAN would collapse each player's entire history into one flat point and lose the rich timing and sequence information.

It handles key challenges as follows:

• Temporal patterns (day/timeOfDay)

I encode day-of-week and hour-of-day as cyclical features (sine/cosine) and feed them into the LSTM along with wagers and wins. The model learns, for example, that high rollers play more on weekends and evenings, while locals stick to weekday hours.

• Session sequences (ordering matters)

The LSTM processes each player's sessions in chronological order, carrying a hidden state that captures how past outcomes influence future behavior. Alternating games, streaks of wins or losses, and escalating bets all become learnable patterns rather than noise.

Technical details:

Sessions are first sorted by timestamp (dt) for each player and then converted into an ordered list of session feature vectors. Because different players have different session counts, I use pad_sequences (with maxlen=112, corresponding to the 90th percentile) to either truncate older sessions or zero-pad shorter histories at the front. A Masking layer in the LSTM ensures those leading zeros are ignored, so only real session data influences the learned embeddings.

• Sparse negative netWin events

Because only ~10% of sessions are player wins, I add a binary "win" flag as an input feature and—if needed—apply sample weights or oversampling so the LSTM pays extra attention to those rare but important jackpot events. This ensures the embedding reflects both the common loss patterns and the occasional big wins.

In the codes attached, I feed each player's chronologically ordered session features into an LSTM encoder, producing a 64-dimensional embedding, and then compute cosine similarity between embeddings to find behavioral peers. For example, for a random player 010, the top five most similar players by cosine similarity are:

Top 5 players similar to 010:

063 0.996333

098 0.994458

094 0.976160

086 0.973580

081 0.971546

Name: 010, dtype: float32

The embeddings can easily be applied in real-world problem solving.

For churn detection, I flag players who haven't returned in the last time window as churners, then use their embeddings to find active players with similar profiles. **These are at highest risk of dropping off and are prime targets for retention offers**.

- Top 10 high-churn-risk players: ['060', '094', '013', '040', '069', '071', '006', '024', '050', '009']

For high-risk gambling, I define a "risky" prototype—for example, players in the top 10 percent by average wager who play late at night—compute its centroid in embedding space and **flag any players whose embeddings exceed a chosen similarity threshold.**

High - risk players (sim > 0.8):

014 0.821580

031 0.907487

033 1.000000

035 0.818944

041 0.818087

056 0.820895

062 0.851510

068 0.918609

084 0.945347

089 0.832534

The player embeddings are very informative to use. To support behavioral segmentation and outreach, I can apply clustering on embeddings only for segmentation (rather than as the primary similarity measure) to define segments such as VIP high-rollers versus casual locals, and tailor communications accordingly. I can use nearest-neighbor lookups to recommend promotions: for a given VIP, I find peers whose offers worked well and mirror those. For regulatory alerts, I maintain a sliding window of recent sessions, recompute the player's embedding, and trigger a flag if it

drifts into a concern region, such as heavy 3 AM bets combined with loss-chasing patterns.

Pros of Player Embeddings from Sequence Models

1. Rich behavioral representation: Maintains the full sequence of sessions, so it captures dynamics like loss-chasing, game alternation, and weekend/weekday patterns rather than relying on static aggregates.
2. Joint nonlinear modeling: Learns complex, time-dependent relationships (e.g. how wagering and session length evolve together) without hand-crafting features.
3. Unified, compact output: Produces fixed-length vectors that can be reused for any downstream task—churn prediction, risk detection, segmentation—using the same representation.

Cons

1. Data alignment required: Sessions must be chronologically ordered and padded or truncated to a uniform length. In this task I use pad_sequences (with maxlen=112, corresponding to the 90th percentile) to either truncate older sessions or zero-pad shorter histories at the front. Players with very sparse or extremely long histories need special handling.
2. Training complexity: Sequence models demand more compute and careful tuning (sequence length, masking strategy, hyperparameters) compared to simpler static methods.
3. Opaque embeddings: The learned vectors are less interpretable—insights require additional probing or surrogate models, whereas traditional clustering yields clear centroids or cluster labels.