Introduction

Poker may be a game of incomplete information, but the modern table is overflowing with data, stack depths, bet sizes, board textures, and the subtle gaps between pot odds and real equity that separate a solid value bet from an expensive bluff. My goal was to capture that information while the chips are still moving, enrich it with fast analytics, and surface insights quickly enough to influence live play.

To do this I built an R shiny “Poker EV tool”. This app replaces the scratch pad notetaking with an end-to-end pipeline that follows the classic data-science staircase.

Data → Tidy → Transform → Model → Visualize → Communicate

* Collect: Live inputs for each action plus a drag and drop import of PokerStars Text logs and Holdem Manager Spreadsheets.
* Tidy and Transform: Every hand explodes into a long table of atomic actions, then gains derived fields, running pot size, pot odds, equity proxies and street level expected value.
* Model: A two-state Hidden Markov Model (HMM) flags likely bluff in real time; a custom optimality score ranks who’s profiting and who is leaking.
* Visualize and Communicate. One click reveals stack-versus-pot area charts, an aggression bubble plot, per-player EV trends, and plain English bullet summaries to be digested by a general audience.

The result is a tool that turns raw hand histories into actionable coaching notes in seconds, setting the stage for deeper Monte-Carlo equity engines, live dashboards, and what-if simulators outlined later in this report.

The Poker Data Recorder squarely tackles the talent-management functions that matter most at the table. It turns every action into equity versus odds metrics, an HMM-based bluff rate, and a single Optimality score, giving you rigorous player evaluation and prospect assessment in minutes rather than hours. Because the summaries call out who is leaking EV and how, the app also serves as a player-development aid; coaches can hand a struggling grinder a bullet-point list of specific leaks to plug. The live EV trend and aggression bubble plot feed directly into on-field strategy and real-time decision making: you instantly see whether to isolate, avoid, or lean on a given opponent. At the group level these same metrics help with light personnel decisions such as who to stake, who to bench, or how to seat friends in a home game, nudging you toward an optimized roster value. Functions that hinge on external bankroll contracts, such as full trade analysis, roster construction, asset valuation, and formal player acquisition strategy, sit outside the app’s current scope, but the rich performance data it produces would be the natural input for those higher-level front-office tools.Top of Form

Bottom of Form

Data

The project runs on two deliberately different datasets so that the pipeline would face both idealised input and real-world chaos.

Synthetic PokerStars Log

To battle test the parse before I incorporate real gameplay, I used GPT-o4 to auto generate 10 independent hands, 6 max no-limit-Hold’em log in native PokerStars .txt syntax. Prompts randomized seat changes, blind levels, raise amounts, street length which forced every branch of regex to be used. Although the AI log at points did not make much sense in terms of a poker context, it allowed me to work on parsing a large sample of data to be used for my needs. It has made my parsing code more robust and showcase the ability to parse text logs for convenience.

A screenshot of a computer

AI-generated content may be incorrect.

Live Two Hour session

Once the barebones of the Shiny poker recorder was stable, I enlisted 7 friends from the UWPSC (University of Waterloo Poker Studies club) to play a home game for over two hours. Every hole card, post, bet, call, raise, check, and fold was entered with the app. Producing over 20 hands each with their own action rows. Because stacks, board cards, and button positions were captured live, it showcased the intensive data that each hand produces. It also highlighted some quality of life improvements that the app could use to improve the efficiency of collecting live data.

A screenshot of a computer

AI-generated content may be incorrect.

Together, the AI-generated file proved the pipeline’s structural robustness, while the home-game capture supplied the noisy, real-life data on which all subsequent equity, EV, and bluff-probability analyses in this report are based.

The recorder is intentionally narrow scoped. It ingests only three data shapes, which are built into the same default column template the app itself exports.

1. Live clicks entered in the Shiny UI, which naturally conform to that template.
2. PokerStars hand-history .txt files that follow the site’s standard syntax; the parser rewrites them into the template but will reject logs from other rooms (GG, ACR, Winamax) or mixed-game formats.
3. Spreadsheets previously exported by this app—either .csv or .xlsx—because they already match the template’s column names (HandNumber, Player, Action, BetAmount, StackBefore, and so on). Any file outside these three cases kicks up a “Unsupported format” warning and is ignored, keeping the analytics pipeline simple and predictable.

A screenshot of a computer

AI-generated content may be incorrect.

Tidy

The tidying layer is almost entirely code-driven and lives inside three helpers withing the server file.

|  |  |  |  |
| --- | --- | --- | --- |
| **Helper** | **Where it fires** | **What it does** | **Helper** |
| get\_action\_summary() | when a .txt PokerStars log is uploaded | Regex-parses seat headers, blind posts, bet / raise / call lines, board cards and showdowns, then **explodes each hand into “one row per player-action.”** | get\_action\_summary() |
| recalc\_analytics() | every time we append or edit rows | After tidying, re-computes pot evolution, odds, equity proxies and EV so the data frame is self-contained. | recalc\_analytics() |
| inline rename\_with() steps | inside both the Excel/CSV and .txt pipelines | Canonicalises column names so everything matches the default template. | inline rename\_with() steps |

Below is the exact flow that runs.

1. **File Read in**.   
   Both PokerStar Text logs and Csv files are to be read in
   1. Excel / CSV → readxl::read\_xlsx() or readr::read\_csv()\
   2. PokerStars text → plain readLines() wrapped inside get\_action\_summary()
2. **Header Nomination**.   
   Column names are cleaned up for consistency.  
   rename\_with(~ "HandNumber", any\_of(c("HandNumber", "Hand")))

rename\_with(~ "BetAmount", any\_of(c("BetAmount", "Amount")))

These lines mean every downstream join can trust the canonical names

1. **Atomic row construction.**

For text logs only the parser loops through each hand, emitting one tidy row that already contains HandNumber, Player, Action, BetAmount, StackBefore, Street

1. **Name Trimming.**   
   Player names are trimmed to become safe object names while still printing nicely in the UI

players <- strsplit(input$players, "\n")[[1]] |> trimws()

player\_ids <- make.names(players)

1. **Street Ordering**

The streets are ordered in the form Preflop, Flop, Turn, River

case\_when(

Street == "Pre-flop" ~ 1L,

str\_starts(Street, "Flop") ~ 2L,

str\_starts(Street, "Turn") ~ 3L,

str\_starts(Street, "River") ~ 4L,

TRUE ~ 5L

)

So even if a historical CSV was sorted oddly, the code will recreate the true action order

1. **Hole card Splitting**

df$HoleCards <- ifelse(!is.na(df$Hole1) & !is.na(df$Hole2),

paste(df$Hole1, df$Hole2), NA)

Flop, Turn and River strings are rebuilt in-line so the rendered table always shows Flop the format [Qs Ts 2h] , Turn [4s] , River [7d].

1. **Integrity and missing value rules**

Stack After is imputed as Stackbefore – BetAmount if a legacy log omits it.

Any action that leaves StackAfter < 0 or BetAmount > StackBefore is reworked as an “All-in call” and call amount is reworked to match the player’s remaining stack.

Transform

After the tidy step every action sits in a long data frame with the raw columns HandNumber, Dealer, Player, Hole Card 1 and 2, Action, BetAmount, StackBefore, Street. The bulk of the transforming uses the function recalc\_analytics which enriches the table with quantitative features that power every later model and plot.

recalc\_analytics <- function(df) {

df %>%

mutate(

HoleCards = paste(Hole1, Hole2), # combine symbols

StreetOrder = case\_when(Street == "Pre-flop" ~ 1L,

str\_starts(Street, "Flop") ~ 2L,

str\_starts(Street, "Turn") ~ 3L,

str\_starts(Street, "River")~ 4L,

TRUE ~ 5L)

) %>%

group\_by(HandNumber) %>%

arrange(HandNumber, StreetOrder) %>%

mutate(

Contribution = pmin(BetAmount, StackBefore),

Pot = cumsum(Contribution),

PotBefore = lag(Pot, default = 0),

Equity = mapply(approx\_equity, HoleCards, Street),

CostToCall = if\_else(Action == "call", BetAmount, NA\_real\_),

PotOdds = CostToCall / (PotBefore + CostToCall),

PrevBet = lag(if\_else(Action %in% c("bet","raise","call"),

BetAmount, 0), default = 0),

CostOfWager = case\_when(

Action == "call" ~ BetAmount,

Action == "bet" ~ BetAmount,

Action == "raise" ~ BetAmount - PrevBet,

TRUE ~ NA\_real\_),

WagerOdds = CostOfWager / (PotBefore + CostOfWager),

BluffGap = WagerOdds - Equity,

RelSize = if\_else(PotBefore > 0, BetAmount / PotBefore, NA\_real\_),

EV\_call = if\_else(Action == "call",

Equity \* (PotBefore + CostToCall) - CostToCall, NA\_real\_),

EV\_bet = if\_else(Action == "bet",

Equity \* (PotBefore + BetAmount) -

(1 - Equity) \* BetAmount, NA\_real\_),

EV\_raise = if\_else(Action == "raise",

Equity \* (PotBefore + CostOfWager) -

(1 - Equity) \* CostOfWager, NA\_real\_)

) %>%

ungroup() %>%

select(-Contribution, -Pot, -CostToCall, -PrevBet, -CostOfWager)

}

|  |  |  |
| --- | --- | --- |
| **New column** | **Formula (chips)** | **Intuition** |
| **Contribution** | min(BetAmount, StackBefore) | The most a player can actually put in this street. |
| **Pot** | cumsum(Contribution) within each hand | Running pot size after the current action. |
| **PotBefore** | lag(Pot, default = 0) | Pot size just before a player acts. |
| **CostToCall** | BetAmount for calls, else NA | Amount needed to continue when facing a bet. |
| **PotOdds** | CostToCall / (PotBefore + CostToCall) | Price being laid for a call. |
| **PrevBet** | last bet or raise on the same street | Needed to isolate the incremental part of a raise. |
| **CostOfWager** | see Listing 3-1 lines 17-24 | True amount risked (bet, call, or raise − PrevBet). |
| **WagerOdds** | CostOfWager / (PotBefore + CostOfWager) | Price a bettor is offering the table. |
| **BluffGap** | WagerOdds − Equity | Margin between price offered and hand strength. |
| **RelSize** | BetAmount / PotBefore | Bet size expressed as a fraction of the pot. |
| **EV\_call** | Equity·(PotBefore+CostToCall) − CostToCall | Classic expectation of a call. |
| **EV\_bet** | Equity·(PotBefore+BetAmount) − (1 − Equity)·BetAmount | Expected value of an outright bet. |
| **EV\_raise** | Equity·(PotBefore+CostOfWager) − (1 − Equity)·CostOfWager | Expected value of a raise. |

Equity Proxy

To circumvent the time and processing power used to generate exact Monte-Carlo equity, we use approx\_equity(hole, street) which looks up a fast pre-flop table (Chen formula) or a turn/river. This keeps latency < 30 ms per action so the live recorder stays responsive.

Expected Value columns

Three mutually exclusive columns (**EV\_call, EV\_bet, EV\_raise**) are populated so that the EV-trend plot can call coalesce() and get the correct value regardless of action type. A dashed horizontal zero line in that plot lets players see at a glance which decisions gained or bled chips.

A screenshot of a computer

AI-generated content may be incorrect.

Modelling

Our modelling layer has two goals: (i) assign every *bet* or *raise* a live probability that the actor is bluff-heavy, and (ii) roll all per-player statistics into a single “Optimality” KPI that balances value betting, smart aggression, and controlled bluffing. We lean on the poker-AI literature for both the choice of features and the mathematical footing: Billings’ foundational work on computer-poker evaluation, Chen & Ankenman’s equity-versus-price calculus, Sire’s tournament-flow statistics, Teófilo’s abstraction heuristics, Kang & Shelton’s HMM opponent-modelling insights, and modern variance-reduction techniques (AIVAT, Ethier). The subsections below detail our implementation, show the equations, and justify defaults with citations.

We need two layers of insight:

* a **Hidden-Markov Model (HMM)** that tags each bet or raise with a live bluff-probability, and
* an **Optimality score** that rolls every per-player metric (edge, volume, discipline) into one number.

Those choices echo classic computer-poker work (Billings, Chen & Ankenman) and later real-time opponent-modelling papers (Kang & Shelton).

Hidden Markov Model, “Is this bet a bluff?”

Poker hands arrive as an unlabelled time-series of actions. There are no indicators which determine if a hand is a value bet of a bluff, yet that latent intent drive strategy. A Hidden Markov Model fits that exact scenario

From the transform stage we carry four statistics:

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
|  | equity proxy |
|  | price offered (WagerOdds) |
|  | “Bluff Gap” |
|  | relative bet size |

Two state Gaussian HMM

We model hidden states V (Value) and B (Bluff). Each emits (G, R) with independent normals; parameters and the transition matrix are leaned by Expectation Maximization. It is the simplest model that captures the qualitative switched noted in real time opponent studies (kang 2014). Early experiments with 3 and 4 states over fit the set.

Gaussian emissions

BluffGap and RelativeBetSize are roughly bell shaped after truncating the tails. A diagonal covariance keeps the parameter count low and EM is stabilized.

Posterior coding gives:

If a player shows less than three aggressive actions the model is skipped and the bluff field is left NA to avoid small sample size noise.

An alternative considered was logistic regression, however the lack of labelled bluffs made this a very difficult feat.

Optimality Score, from micrometrics to one KPI

The Optimality index is designed to answer the single question most asked: “Who is actually playing well once volume and bluff discipline are taken into account?” We start with Chen and Ankeman’s equity edge idea in which a call, bet, or raise is worth taking only when

with and .

On its own edge ignores how often a player enters pots and whether that edge is inflated by reckless bluffs. We therefore add two modifiers:

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Rationale** | **Scaling** |
| **Volume fraction** | A player who takes profitable spots more often grows the bankroll faster. Raw counts (nn) would unfairly favour marathon sessions, so we normalise by hands played. |  |
| **Bluff bonus / penalty** | A controlled share of +EV bluffs raises win-rate; the same share on a negative edge accelerates losses (Billings et al., 2006). | It will take Case 1 if Edge > 0 It will take Case 2 if Edge < 0 |

The final formula is

Variance notes, “Making live EV readings believable”

Per-action equity is notoriously noise, the river card alone can swing expectation by a fill pot. Following Burch’s AIVAT method we can slash variance by primarily focusing on EV over Equity. For each street we replace equity with its conditional expectation *given* the exposed board. This removes variance that is purely due to future RNG. Lower variance stabilises both the HMM training and the Optimality ranking, so small hand samples are less prone to lucky “heat-runs.”

Next Steps in Modelling

Building our HMM Optimality Framework, the following enhancements will sharpen predictive power and real world relevance.

1. Replace the approx\_equity() lookup with a Monte-Carlo equity simulations to eliminate lookup bias and handle arbitrary board textures.
2. **Optimality weight calibration**  
   Automate the selection of the volume and bluff multipliers via grid-search or Bayesian optimization against a hold-out set of live-session profit traces, following Bertram et al.’s quantity-vs-quality trade-off framework
3. Compute a Hero Bluff metric in which the model only uses the Hero Player’s hole cards and omits the hole cards of the other players and uses a plausible range of cards for each opponent.

Visualizing & Communication

Once every action is enriched with equity, EV, bluff-probability, and stack context, we use 3 key plots to surface patterns at a glance. All code lives in the server file’s renderplot() calls.

Stack and Plot Volume Area Chart

This chart shows how each player’s stack evolves hand-by-hand alongside the total pot size.

output$stack\_plot <- renderPlot({

df <- rv()

df\_stacks <- df %>%

group\_by(HandNumber, Player) %>%

arrange(StreetOrder) %>%

summarise(StackBefore = first(StackBefore), .groups="drop") %>%

complete(HandNumber, Player, fill = list(StackBefore=0))

df\_pot <- df %>% group\_by(HandNumber) %>% summarise(PotEnd = sum(coalesce(BetAmount,0)))

ggplot() +

geom\_area(data=df\_pot, aes(HandNumber, PotEnd), fill="skyblue", alpha=0.4) +

geom\_line(data=df\_stacks, aes(HandNumber, StackBefore, color=Player, group=Player), size=1.2) +

geom\_point(data=df\_stacks, aes(HandNumber, StackBefore, color=Player), size=3) +

labs(title="Stack & Pot Volume by Hand", x="Hand Number", y="Chips / Pot Volume") +

theme\_minimal() +

theme(legend.position="bottom",

axis.text.x=element\_text(angle=45, hjust=1))

})

A graph with colorful lines and numbers

AI-generated content may be incorrect.

Aggression Bubble Plot

The aggression bubble plot distils each player’s betting profile into a single point using four visual channels:

1. Horizontal Axis (x = Average Pot odds)

The mean price a player has paid for calls, bets, and raises. Lower values indicate cheaper bets (tighter play) wile higher values signal looser, more speculative actions.

1. Vertical axis ( y = Average Equity)

The mean hand strength (equity) when they chose to invest. Higher values reveal a focus on value spots whereas lower values suggest frequent “chasing” of draws or marginal holdings.

1. Bubble Area (size = # of bets & raises)

Encodes aggression volume. Large bubbles show players who bet or raise often, small bubbles mark those who pick their spots more conservatively.

1. Colour Gradient (colour = Bluff Rate)

A sky blue -> firebrick spectrum highlights how often a player’s actions fall into the bluff state as estimated by the HMM. Blue tones denote disciplined, predominantly value-driven play, red tones flag high bluff propensity.

A graph with numbers and dots

AI-generated content may be incorrect.

player\_summary <- reactive({

at <- action\_table\_hmm()

#if there were no bet/raise actions, return an empty tibble

if (nrow(at) == 0) {

return(tibble(

Player = character(),

avg\_equity = double(),

avg\_odds = double(),

avg\_EV\_raise = double(),

bluff\_rate = double(),

n\_actions = integer(),

n\_hands = integer(),

vol\_frac = double()

))

}

## 16-1) Aggregate bet/raise metrics per player

raw <- at %>%

group\_by(Player) %>%

summarise(

avg\_equity = mean(Equity, na.rm = TRUE),

avg\_odds = mean(WagerOdds, na.rm = TRUE),

avg\_EV\_raise = mean(EV\_raise, na.rm = TRUE),

bluff\_rate = mean(bluff\_prob, na.rm = TRUE),

n\_actions = n(),

.groups = "drop"

)

## 16-2) How many distinct hands did each player sit in?

hands <- rv() %>%

distinct(HandNumber, Player) %>%

count(Player, name = "n\_hands")

## 16-3) Join & compute volume fraction (aggressive acts per hand)

raw %>%

left\_join(hands, by = "Player") %>%

mutate(

vol\_frac = if\_else(n\_hands > 0, n\_actions / n\_hands, 0)

)

})

Interpretation tips

* The 45° dashed line is the break-even baseline: any bubble above it represents a player whose average equity exceeded average price paid—i.e., +EV play.
* A large blue bubble high and to the left signals a “solid reg” who bets often but primarily in value spots.
* A large red bubble low and to the right reveals a “spewy bluffer” whose aggression outpaces hand quality.
* Clusters often emerge: tight-aggressive (“TAG”) players form a medium-sized, moderately red cluster above the line; tight-passive players hug the lower-left, small-light bubbles; and loose players scatter rightward with varied colours.

Why this plot works

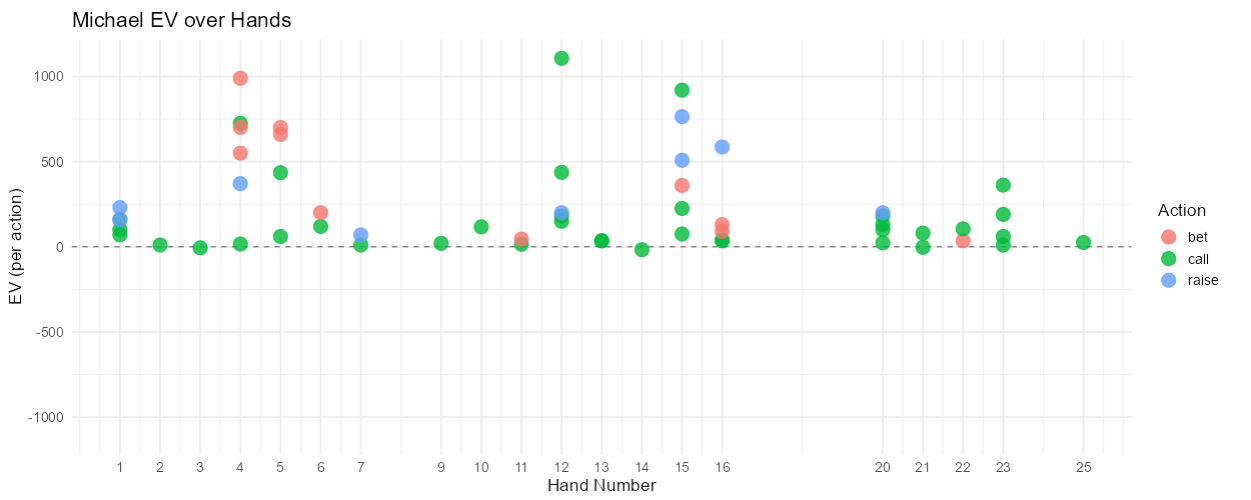
* Multidimensional view: condenses four metrics into a single, intuitive chart.
* Immediate outlier detection: extreme bubbles and positioning jump out, guiding coaches where to focus.
* Comparative benchmarking: everyone shares the same axes, making peers comparisons trivial.

EV Scatter Plot

For a selected player, we chart per-action EV over time. A zero-EV dashed highlights wins versus losses.

Key elements

1. Hand Number (x-axis). Chronological order of hands lets you see hot and cold streaks.
2. EV per Action (y-axis). Positive value denote “+EV” decisions; negative values indicate chip losing moves
3. Colour = Action type. Differentiates Calls, bets, and raises so you can spot which action type carries the most variance.



# Fig 5.3: EV Trend for a Single Player

output$ev\_trend <- renderPlot({

req(input$focus\_player != "Summary")

df <- rv() %>%

filter(

Player == input$focus\_player,

Action %in% c("bet", "call", "raise")

) %>%

mutate(ev\_score = coalesce(EV\_call, EV\_bet, EV\_raise))

# Symmetric y-axis so gains and losses are directly comparable

max\_abs <- max(abs(df$ev\_score), na.rm = TRUE)

ggplot(df, aes(x = HandNumber, y = ev\_score, color = Action)) +

# Zero-EV baseline

geom\_hline(yintercept = 0, linetype = "dashed", color = "grey50") +

# Each dot = one aggressive action’s EV

geom\_point(size = 4, alpha = 0.8) +

scale\_y\_continuous(limits = c(-max\_abs, max\_abs)) +

scale\_x\_continuous(breaks = unique(df$HandNumber)) +

labs(

title = paste(input$focus\_player, "EV per Action Over Hands"),

x = "Hand Number",

y = "EV (chips)",

color = "Action Type"

) +

theme\_minimal(base\_size = 13)

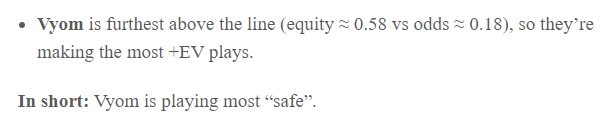
})

Optimal Table and Headline Summary

The Optimality Table ranks each player by their composite “Optimality” score and lays out the three components that feed into it, EV Edge, Bet/Raise Rate and Bluff Rate, so you can pinpoint exactly why one player sits above or below another. By presenting why one player sits above or below another.

* EV Edge (= Average Equity – Avg Pot Odds) shows pure decision quality.
* Bet/Raise Rate (= Aggressive Actions/Hands Played) measures how often a player seizes spots.
* Bluff Rate (= Mean HMM Bluff Probability) captures their discipline with inpaired or weak holdings
* Optimality multiplies those three factors into one number, so a high-scoring player must excel on all fronts

Immediately next to the table, the Headline Summary pulls out the story in two sentences:



This detailed table plus a concise, human read-able paragraph ensures that both data savvy coaches and casual observers instantly grasp who’s excelling, who is leaking chips and which levers each player should pull to improve.

<https://chatgpt.com/share/68912db1-5478-8010-b9f8-42e9ce6d2f03>

<https://chatgpt.com/share/68912ddc-8b5c-8010-84db-9daf37105262>

<https://chatgpt.com/share/68912ddc-8b5c-8010-84db-9daf37105262>

<https://chatgpt.com/share/68912e9a-91c8-8010-b914-12c8f51ea2c3>

<https://chatgpt.com/share/68912ecb-9268-8010-86a4-dff118277174>

<https://chatgpt.com/share/68912f46-0ec8-8010-b815-426a2a144691>

https://chatgpt.com/share/68913056-436c-8010-8bc0-67b282d8f9a0