NHL Shooting Quality Analyzer

STAT 436 — Homework 2: Shiny Portfolio Project

App link: https://github.com/romanhauch/stat436-hw2

Interesting Findings

This application explores how NHL skaters' actual goal totals compare to their expected goals (xGoals), a model-based metric from MoneyPuck that estimates how many goals a player should score given the quality of their shot attempts. The gap between goals and xGoals reveals which players are riding hot shooting luck and which are due for a rebound.

The most striking finding was the sheer magnitude of some over- and underperformers. Morgan Geekie of Boston led the league with 32 goals despite an xGoals estimate of just 16.9 — a difference of +15.1 goals. Nathan MacKinnon (+13.9) and Sidney Crosby (+10.1) also appeared among the top overperformers, though their elite finishing talent makes sustained overperformance more plausible. On the other side, Jake DeBrusk of Vancouver scored only 13 goals against an xGoals of 25.6 (-12.6), suggesting he generated plenty of quality chances but could not convert them.

An unexpected finding emerged when comparing shot danger profiles across positions. Defensemen average just 1.9% high-danger shots compared to roughly 10% for forwards. This is intuitive on reflection — defensemen shoot mostly from the point — but the visualization makes the disparity vivid. When I filtered to individual defensemen like Zach Werenski (+9.8 goals above expected), the shot danger chart showed an unusually high proportion of medium- and high-danger shots for a defenseman, helping explain his outlier production. This kind of cross-panel discovery, where selecting a point on the scatter plot reveals the underlying shot profile, is exactly what the app is designed to surface.

Interface Design

The application uses a sidebar layout built with bslib's page_sidebar, which keeps filter controls persistently visible without consuming the main viewing area. The sidebar contains four inputs: a team selector (dropdown), position filter (checkboxes), a minimum games played slider, and a radio button to toggle how scatter plot points are colored. These serve as the first dynamic query mechanism. The minimum games slider is important for data quality — players with very few games produce noisy xGoals estimates that would clutter the plot.

The second dynamic query is a graphical input: users can drag to lasso-select points directly on the plotly scatter plot. Selected players flow into two detail panels below — a stacked bar chart showing each player's shot danger breakdown (low, medium, high) and a grouped bar chart comparing their actual goals to expected goals side by side. When no selection is made, the app defaults to displaying the eight players with the largest absolute gap between goals and xGoals, ensuring the bottom panels are never empty.

Data preparation involved filtering the MoneyPuck dataset to the "all" situation type (aggregating across 5-on-5, power play, and penalty kill) and computing derived columns including goals_above_expected, high-danger shot percentage, and a categorical luck label. Each plot panel includes a brief annotation explaining what the user is looking at, so the app is interpretable without prior familiarity with hockey analytics. Color choices were deliberate: red for overperformers and blue for underperformers in the luck status view, and a qualitative Set2 palette for positions to avoid implying ordinal ranking.

Reactive Graph Structure

The reactive graph has two main chains. The first begins with the four sidebar inputs (team, position, min_games, color_by), which feed into a single reactive expression called filtered_data(). This reactive filters the full skaters dataframe according to the current input state. The filtered_data() reactive drives the main scatter plot output. Consolidating all filtering logic into one reactive avoids duplicating filter code across outputs.

The second chain begins with the plotly selection event. When users drag-select points on the scatter plot, event_data("plotly_selected") captures the selected player names via the key aesthetic. A second reactive, selected_players(), reads this event and filters the already-filtered data to just the selected names. If no selection exists, it defaults to the top 8 players by absolute goals_above_expected. This reactive feeds both the danger breakdown chart and the goal comparison chart.

In summary, the structure flows as: sidebar inputs → filtered_data() → scatter_plot, and then plotly brush event → selected_players() → danger_chart + goal_comparison. All complex plotting logic is externalized into three helper functions (build_scatter, build_danger_chart, build_goal_comparison) to keep the server function clean and modular. Data loading and preparation happen once outside the server, avoiding redundant computation on each reactive update.