Analytics.Bet

Applied Analytics and Sports Betting
Lecture 6: Model Evaluation

Typesetting

These convenience functions simply help with uploading graphs and regression results.

In[15]:=
```
slate = RGBColor["#37474f"];
ClearAll@save;
save[plot_, fn_ : Automatic, recolor_ : (Green → White)] :=
 With[{f = Export[$WorkDirectory <> "Analytics.Bet/slides/img/" <> Replace[fn, Automatic →
         DateString[{"Year", "Month", "Day", "-", "Hour", "Minute", "Second"}]] <> ".png",
     plot /. recolor, Background → None]}, Column[{plot, f}, Center]]
showlm = ColorNegate@Rasterize@Framed[Column[{#["ParameterTable"],
         Row[{"Estimated Standard Deviation: ", Sqrt@#["EstimatedVariance"]}]}],
       Background → ColorNegate@slate] &;
```

Data Collection

We need to download all the boxscores and player summary information for 2021.

Boxscores

Collect a list of all boxscore hyperlinks from the complete list of games for each month Basketball-Reference.com:

In[18]:=
```
bsurls = Flatten[
    With[{url = "https://www.basketball-reference.com/leagues/NBA_2021_games.html"},
     Select[Import[#, "Hyperlinks"], StringContainsQ@"/boxscores/20"] & /@
      Flatten@StringCases[Import[url, "Hyperlinks"],
        ___ ~~ FileBaseName[url] ~~ "-" ~~ __ ~~ ".html"]], 1]; // AbsoluteTiming
```

It is almost always better to download the raw HTML's to a local directory and then process it, rather than scrape and parse in one step, because you can iterate on your parsing code more quickly.

Download all of the boxscores in bulk to a local directory:

```
URLDownload[bsurls, $WorkDirectory <> "Analytics.Bet/Data/BR"];
```

Player Summary

To download the player totals, we need to scrape each month's list of games again. Even better than this approach would be to generate the player totals from the boxscores directly. That way, you could more easily move into rolling averages as needed. The more direct approach we are doing here can sometimes be a convenient first step. Note here we are also violating the suggestion to download first. So, this is a great example of a first step approach, before the more rigorous approach that we did with box scores above.

As a scraping suggestion, use Inspect in your browser to find the names of the tables you wish to extract data from.

Scrape and import player total stats for all months from Basketball-Reference.com:

In[20]:=
```
ClearAll@brScrapeOne;
brScrapeOne[url_String, id_] := brScrapeOne[Import[url, "XMLObject"], id]
brScrapeOne[xml_, id_] :=
 Cases[FirstCase[xml, XMLElement[_, r_ /; StringMatchQ["id" /. r, id], tbl_] :>
    FirstCase[tbl, XMLElement["tbody", _, __], {}, ∞], {}, ∞], XMLElement["tr", _, rows_] :>
   Association[Cases[rows, XMLElement["td" | "th", inf_, content_] :> (("data-stat" /. inf) ->
        First[content /. XMLElement["a", _, {t_}] :> t, None]), ∞]], ∞]
```

In[22]:=
```
ClearAll@brScrapePages;
brScrapePages[url_ : "https://www.basketball-reference.com/leagues/NBA_2021_games.html",
  id_ : "div_schedule"] := With[{urls = Flatten@StringCases[
      Import[url, "Hyperlinks"], ___ ~~ FileBaseName[url] ~~ "-" ~~ __ ~~ ".html"]},
  Flatten[brScrapeOne[Import[#, "XMLObject"], id] & /@ If[Length@urls == 0, {url}, urls], 1]]
```

In[•]:=
```
playertotals =
   brScrapePages["https://www.basketball-reference.com/leagues/NBA_2021_totals.html",
    "totals_stats"]; // AbsoluteTiming
```

Out[•]=
```
{14.6287, Null}
```

Store the parsed and downloaded information locally:

In[•]:=
```
DumpSave[$WorkDirectory <> "Analytics.Bet/Data/BR/playertotals.mx", playertotals];
```

Data Manipulation

With the now-local boxscore data, we need to extract the actual rosters for each home and away team for each game.

Define a function to extract the starting rosters from a given game by looking for the first five players for each team:

```
In[23]:= ClearAll@Rosters;
Rosters[fn_] := With[{home = StringTake[FileBaseName[fn], {10, 12}]},
  <|"date" → ToExpression /@ StringTake[FileBaseName[fn], {4, {5, 6}, {7, 8}}],
   "home_abbr" → home, brScrapeOne[If[! FileExistsQ[fn <> ".mx"],
        With[{g = Import[fn, "XMLObject"]}, Export[fn <> ".mx", g];
         g], Import[fn <> ".mx"]], "box-" ~~ t : Repeated[_, {3}] ~~ "-game-basic" /;
        #[t, home]][[ ;; 5, "player"]] & /@
    <|"away_starters" → Unequal, "home_starters" → Equal|>|>]
```

Example of Serialization

Serializing even local data can pay huge dividends. The function above automatically serializes the HTML file if it hasn't been serialized before, so the first time you run it, it is importing from the slower HTML.

```
In[ ]:= rosters = ResourceFunction["DynamicMap"][Rosters,
     FileNames[$WorkDirectory <> "Analytics.Bet/Data/BR/*.html"][[ ;; 5]]]; // AbsoluteTiming
Out[ ]= {11.1799, Null}
```

Re-running it now imports from the much faster serialized files: 10x faster.

```
In[ ]:= rosters = ResourceFunction["DynamicMap"][Rosters,
     FileNames[$WorkDirectory <> "Analytics.Bet/Data/BR/*.html"][[ ;; 5]]]; // AbsoluteTiming
Out[ ]= {1.11944, Null}
```

Extract the rosters for all games. The Wolfram Resource Function DynamicMap automatically shows a progress indicator as it processes the files.

```
In[24]:= rosters = ResourceFunction["DynamicMap"][Rosters,
     FileNames[$WorkDirectory <> "Analytics.Bet/Data/BR/*.html"]]; // AbsoluteTiming
Out[24]= $Aborted
```

Save the results locally in a serialized format for very efficient reading later:

```
In[ ]:= DumpSave[$WorkDirectory <> "Analytics.Bet/Data/BR/rosters.mx", rosters];
```

Data Management

Import the historical market closing lines data and incorporate with all the other data above.

Import from a downloaded CSV:

```
In[19]:= odds = Import[$WorkDirectory <> "Analytics.Bet/Code/nba odds 2021.csv"];
```

Define a function to convert American odds to breakeven probabilities:

```
In[28]:= AmericanToBreakeven[amer_] := If[amer < 0, -amer / (-amer + 100), 100 / (100 + amer)]
```

Convert the two-row input information into single-row information, clean pk's, and calculate the away probability from the moneyline:

In[29]:=
```
totes = <|"date" → With[{month = Quotient[#[[1, 1]], 100]},
        {If[month == 12, 2020, 2021], month, Mod[#[[1, 1]], 100]}],
     "away" → #[[1, 4]], "home" → #[[2, 4]], "market" → Max[#[[All, 11]] /. "pk" → 0],
     "spread" → Min[#[[All, 11]] /. "pk" → 0],
     "away_prob" → N@AmericanToBreakeven[ToExpression@#[[1, 12]]],
     "market_h1" → Max[#[[All, 13]] /. "pk" → 0], "spread_h1" → Min[#[[All, 13]] /. "pk" → 0],
     "actual" → Total[#[[All, 9]]], "actual_home" → #[[1, 9]],
     "actual_away" → #[[2, 9]]|> & /@ Partition[Rest@odds, 2];
totes[[1]]
```

Out[29]=
```
⟨|date → {2020, 12, 22}, away → GoldenState, home → Brooklyn,
 market → 234.5, spread → 7.5, away_prob → 0.273973, market_h1 → 115,
 spread_h1 → 1, actual → 224, actual_home → 99, actual_away → 125|⟩
```

Now recall the information we computed and stored locally earlier.

In[31]:=
```
Get[$WorkDirectory <> "Analytics.Bet/Data/BR/" <> # <> ".mx"] & /@
  {"playertotals", "rosters"};
```

One of the most common data management tasks is mapping identifiers from different sources. Here we try to sort both sources alphabetically and see how close we get.

In[26]:=
```
Rule @@@
 Transpose@{Union@rosters[[All, "home_abbr"]], DeleteCases[Union@odds[[All, 4]], "Team"]}
```

Out[26]=
```
{ATL → Atlanta, BOS → Boston, BRK → Brooklyn, CHI → Charlotte, CHO → Chicago,
 CLE → Cleveland, DAL → Dallas, DEN → Denver, DET → Detroit, GSW → GoldenState,
 HOU → Houston, IND → Indiana, LAC → LAClippers, LAL → LALakers, MEM → Memphis,
 MIA → Miami, MIL → Milwaukee, MIN → Minnesota, NOP → NewOrleans, NYK → NewYork,
 OKC → OklahomaCity, ORL → Orlando, PHI → Philadelphia, PHO → Phoenix, POR → Portland,
 SAC → Sacramento, SAS → SanAntonio, TOR → Toronto, UTA → Utah, WAS → Washington}
```

They are almost all correct, but Chicago and Charlotte got swapped so we manually fix that and keep a mapping function.

In[27]:=
```
HomeAbbrToName = <|"ATL" → "Atlanta", "BOS" → "Boston",
    "BRK" → "Brooklyn", "CHI" → "Chicago", "CHO" → "Charlotte", "CLE" → "Cleveland",
    "DAL" → "Dallas", "DEN" → "Denver", "DET" → "Detroit", "GSW" → "GoldenState",
    "HOU" → "Houston", "IND" → "Indiana", "LAC" → "LAClippers", "LAL" → "LALakers",
    "MEM" → "Memphis", "MIA" → "Miami", "MIL" → "Milwaukee", "MIN" → "Minnesota",
    "NOP" → "NewOrleans", "NYK" → "NewYork", "OKC" → "OklahomaCity", "ORL" → "Orlando",
    "PHI" → "Philadelphia", "PHO" → "Phoenix", "POR" → "Portland", "SAC" → "Sacramento",
    "SAS" → "SanAntonio", "TOR" → "Toronto", "UTA" → "Utah", "WAS" → "Washington"|>;
```

Joining is a database-like process where rows with the same “keys” are matched together. In this case, a game is defined by the date and the home team.

Use the identifier mapping above to join the rosters information with that derived from the historical odds:

In[30]:=
```
totes2 = JoinAcross[totes,
    <|"home" → HomeAbbrToName[#@"home_abbr"], #|> & /@ rosters, {"date", "home"}];
```

Let's define two summary pieces of information: each player's points per game, and each player's points per minute, and calculate the totals based on the roster for each team in each game.

In[32]:=
```
playerppg =
  <|#player → Quiet@Check[N[ToExpression[#pts] / ToExpression[#g]], 0] & /@ playertotals|>;
playerppm = <|#player → Quiet@Check[N[ToExpression[#pts] / ToExpression[#mp]], 0] & /@
    playertotals|>;
```

In[34]:=
```
totes3 = <|#, Flatten@
      Table[ah <> "_starters_" <> First@pp → Total[KeyTake[Last@pp, #[ah <> "_starters"]]],
       {pp, {"ppg" → playerppg, "ppm" → playerppm}}, {ah, {"away", "home"}}]|> & /@ totes2;
```

Now, for each game, we have the date, team names, market lines, actual results, starters, and starters total PPG and PPM.

Display an example game:

In[35]:=
```
totes3[[1]]
```

Out[35]=
```
⟨|date → {2020, 12, 22}, away → GoldenState, home → Brooklyn, market → 234.5,
 spread → 7.5, away_prob → 0.273973, market_h1 → 115, spread_h1 → 1, actual → 224,
 actual_home → 99, actual_away → 125, home_abbr → BRK, away_starters →
  {Andrew Wiggins, Stephen Curry, Kelly Oubre, James Wiseman, Eric Paschall},
 home_starters → {Kyrie Irving, Kevin Durant, Joe Harris, Spencer Dinwiddie,
   DeAndre Jordan}, away_starters_ppg → 71.5879, home_starters_ppg → 82.0695,
 away_starters_ppm → 2.5788, home_starters_ppm → 2.69371|⟩
```

Model Building

There are two different ways we can begin:

- By Total: Take as inputs the average *total* points scored in games where the away team now was also the away team then, and the average total points scored in games where the home team now was also the home team then.
- By Team: Take as inputs the average points scored *by the away team* in games where the away team now was also the away team then, and the average points scored *by the home team* in games where the home team now was also the home team then.

Let's define these overall season averages:

In[36]:=
```
homepts = GroupBy[totes3, #home & → (#"actual_home" &), Mean /* N];
hometotal = GroupBy[totes3, #home & → (#"actual" &), Mean /* N];
awaypts = GroupBy[totes3, #away & → (#"actual_away" &), Mean /* N];
awaytotal = GroupBy[totes3, #away & → (#"actual" &), Mean /* N];
```

Merge all of that information into a new comprehensive dataset and see what a typical game looks like.

In[40]:=
```
totes4 = <|#, "away_pts_avg" → awaypts[#away], "home_pts_avg" → homepts[#home],
    "away_total_avg" → awaytotal[#away], "home_total_avg" → hometotal[#home]|> & /@ totes3;
totes4[[1]]
```

Out[40]=
```
⟨|date → {2020, 12, 22}, away → GoldenState, home → Brooklyn, market → 234.5,
 spread → 7.5, away_prob → 0.273973, market_h1 → 115, spread_h1 → 1, actual → 224,
 actual_home → 99, actual_away → 125, home_abbr → BRK, away_starters →
  {Andrew Wiggins, Stephen Curry, Kelly Oubre, James Wiseman, Eric Paschall},
 home_starters → {Kyrie Irving, Kevin Durant, Joe Harris, Spencer Dinwiddie,
   DeAndre Jordan}, away_starters_ppg → 71.5879, home_starters_ppg → 82.0695,
 away_starters_ppm → 2.5788, home_starters_ppm → 2.69371, away_pts_avg → 113.838,
 home_pts_avg → 110.116, away_total_avg → 224.568, home_total_avg → 227.86|⟩
```

Run two regressions, one using the AvgAwayTotal / AvgHomeTotal and one using the AvgAwayPts / AvgHomePts.

In[96]:=
```
lm1 =
 LinearModelFit[Values@KeyTake[totes4, {"away_total_avg", "home_total_avg", "actual"}],
  {AvgAwayTotal, AvgHomeTotal}, {AvgAwayTotal, AvgHomeTotal}];
lm2 = LinearModelFit[Values@KeyTake[totes4, {"away_pts_avg", "home_pts_avg", "actual"}],
  {AvgAwayPts, AvgHomePts}, {AvgAwayPts, AvgHomePts}];
```

Print the results of the regressions:

In[97]:=
```
showlm@lm1
```

Out[97]=

	Estimate	Standard Error	t-Statistic	P-Value
1	−223.102	27.0611	−8.24441	4.48537×10^{-16}
AvgAwayTotal	0.998296	0.0761451	13.1104	1.05948×10^{-36}
AvgHomeTotal	0.997408	0.0938759	10.6247	3.22237×10^{-25}

Estimated Standard Deviation: 17.895

In[98]:=
```
showlm@lm2
```

Out[98]=

	Estimate	Standard Error	t-Statistic	P-Value
1	−62.4916	23.9157	−2.61299	0.00909226
AvgAwayPts	1.60653	0.154395	10.4053	2.69526×10^{-24}
AvgHomePts	0.947927	0.14668	6.46257	1.51671×10^{-10}

Estimated Standard Deviation: 18.8063

Model Evaluation: Naive Backtesting

Define a general-purpose naive backtesting tool for this domain. The backtest function takes the historical data on which to backtest, the linear regression model, and a replacement rule to transform the information in a given game from the historical data into the parameters needed for the linear regression model.

In[95]:=
```
ClearAll@Backtest;
Backtest[data_, lm_, rep_, α_ : 1] := DeleteCases[
  With[{modelmu = If[NumericQ@#, #, Null] &[α (lm["BestFit"] /. rep[#]) + (1 - α) #market],
      modelsgm = Sqrt@lm["EstimatedVariance"]},
    With[{po = Probability[x > #market, x ≈ NormalDistribution[modelmu, modelsgm]],
      pu = Probability[x < #market, x ≈ NormalDistribution[modelmu, modelsgm]]}, If[
      ! NumericQ@po || ! NumericQ@pu, Null, <|KeyTake[#, {"date", "away", "home", "market",
         "actual"}], "model" → modelmu, "po" → po, "pu" → pu, AssociationThread[{"dir",
        "size", "result"} → If[po > 110 / 210., {1, (po / (110 / 210) - 1) / (210 / 110 - 1.),
          If[#actual > #market, 1 / 1.1, If[#actual < #market, -1, 0]]}, If[pu > 110 / 210.,
          {-1, (pu / (110 / 210) - 1) / (210 / 110 - 1.), If[#actual < #market, 1 / 1.1,
            If[#actual > #market, -1, 0]]}, {0, 0, Null}]]]|>]]] & /@ data, Null];
```

Run the backtests on our two models, lm1 and lm2, defined above.

In[99]:=
```
backtest1 = Backtest[totes4, lm1, {AvgAwayTotal → #@"away_total_avg",
      AvgHomeTotal → #@"home_total_avg"} &]; // AbsoluteTiming
```

Out[99]= {5.1063, Null}

In[100]:=
```
backtest2 = Backtest[totes4, lm2,
    {AvgAwayPts → #@"away_pts_avg", AvgHomePts → #@"home_pts_avg"} &]; // AbsoluteTiming
```

Out[100]= {5.15965, Null}

Summarizing backtest results

Some of the helper functions we need for evaluating naive backtests include determining the vig from a particular pair of lines, given both decimal odds, and the standard deviation of profit, assuming a particular decimal odds, probability, and vig. In most cases, we will assume the decimal odds are 210/110, the probability is 50-50, and the vig is from a -110 / -110 line.

Define and show an example of the vig function:

In[101]:=
```
BetVig[do1_, do2_] := With[{or = 1 / do1 + 1 / do2 - 1}, or / (1. + or)]
BetVig[210 / 110, 210 / 110]
```

Out[102]= 0.0454545

Define and show an example of the standard deviation function:

In[103]:=
```
BetSigma[do_, p_, vig_] := Sqrt[p (do - 1) ^ 2 + (1 - p) - vig^2]
BetSigma[210 / 110, .5, BetVig[210 / 110, 210 / 110]]
```

Out[104]= 0.954545

Define a function summarize a particular backtest, with an optional export parameter to generate a graph rather than return the raw results:

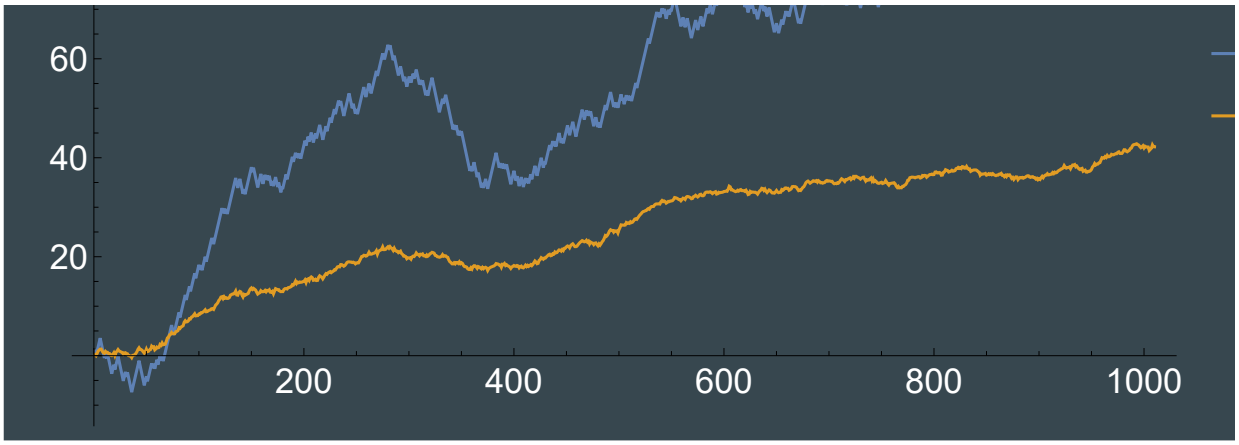
In[116]:=
```
ClearAll@Summarize;
Summarize[backtest_, export_ : True, plotpct_ : False] := With[
  {s = With[{d = Select[Values@KeyTake[backtest, {"size", "result"}], AllTrue@NumericQ]},
     With[{nd = d[[All, 2]], sgm = BetSigma[210 / 110, .5, BetVig[210 / 110, 210 / 110]]},
      <|"Flat Results" → StringRiffle[Values@<|1 → 0, -1 → 0, 0 → 0,
           Counts[Sign[DeleteCases[d[[All, 2]], Null]]]|>, "-"], "Flat Units" → Length@nd,
       "Flat P&L" → Total@nd, "Flat P&L SD" → Sqrt[Length[nd]] sgm,
       "Flat ROI" → If[export, ToString[PercentForm[#, {5, 2}]] &, Identity]@Mean@nd,
       "Kelly Units" → Total[d[[All, 1]]], "Kelly P&L" → d[[All, 1]].d[[All, 2]],
       "Kelly P&L SD" → Sqrt[Total[d[[All, 1]]]] sgm, "Kelly ROI" → If[export, ToString[
            PercentForm[#, {5, 2}]] &, Identity][d[[All, 1]].d[[All, 2]] / Total[d[[All, 1]]]],
       "plot" → ListLinePlot[{Accumulate[d[[All, 2]]] / If[plotpct, Range[Length@d], 1],
          Accumulate[Times @@@ d] / If[plotpct, Accumulate@
             Abs[d[[All, 1]] /. 0 → .0000000001], 1]}, ImageSize → 600, LabelStyle →
          Directive[18, If[export, White, Black]], PlotLegends → {"Flat", "Kelly"}]|>]]},
  If[export, Framed[Row[{Framed[Style[Grid[Flatten[Riffle[SplitBy[List @@@
              Normal@KeyDrop[s, "plot"], StringTake[First@#, 1] &], {{{"", ""}}}}] /.
           r_Real :> NumberForm[r, {5, 2}], 1], Background → slate], 18, White],
       Background → slate, FrameStyle → LightGray, FrameMargins → 20], s@"plot"},
     Spacer@2], Background → slate, FrameStyle → slate], s]]
```

Let's summarize the two backtests we just saw.

In[119]:=
```
Summarize /@ {backtest1, backtest2}
```

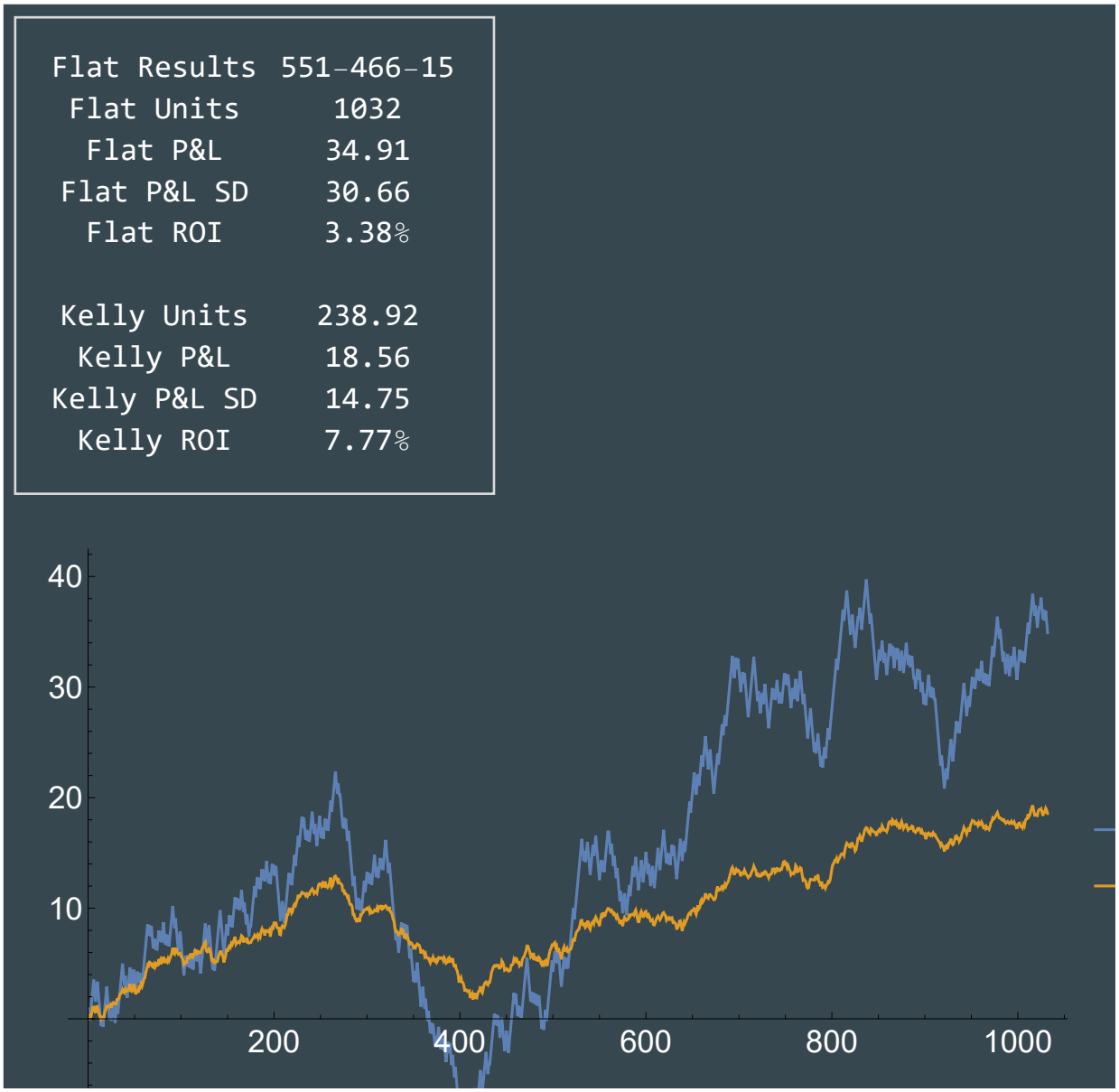
Out[119]=

Flat Results	580–415–15
Flat Units	1010
Flat P&L	112.27
Flat P&L SD	30.34
Flat ROI	11.12%
Kelly Units	222.73
Kelly P&L	42.21
Kelly P&L SD	14.25
Kelly ROI	18.95%



,

Flat Results	551–466–15
Flat Units	1032
Flat P&L	34.91
Flat P&L SD	30.66
Flat ROI	3.38%
Kelly Units	238.92
Kelly P&L	18.56
Kelly P&L SD	14.75
Kelly ROI	7.77%



}

Moving from In-Sample to Out-of-Sample with Rolling Averages

In-sample averages like the full-season away and home totals use information from the future. A more realistic model would be built only using data available at the time of the game. Thus, we need to use some kind of rolling or moving averages rather than full in-sample values.

There are a variety of different ways to calculate moving averages.

Define functions for calculating simple moving averages (SMA), recently-weighted moving averages (RWMA), and exponential moving averages (EMA). For cases with insufficient data to calculate, return a Null rather than an error.

In[120]:=
```
sma[n_][l_] := N@Mean@Take[Reverse@l, UpTo@n]
rwma[h_][l_] := Quiet@
  Check[With[{w = (1 / 2) ^ (# / h) & /@ Range@Length@l}, w.Reverse@l / N@Total[w]], Null]
ema[n_][l_] := Quiet@Check[N@Last@ExponentialMovingAverage[l, 2 / (n + 1)], Null]
```

Calculate four different moving averages for a specific history of games:

In[123]:=
```
Grid[Prepend[With[{hist = {90, 120, 100, 110}},
    Through[{Last, sma[∞], sma[2], rwma[2], ema[.5]}[hist[[ ;; #]]] & /@
      Range[Length@hist] /. r_Real :> NumberForm[r, {5, 2}]],
  Style[#, Bold] & /@ {"history", "soa", "sma(2)", "rwma(2)", "ema(.5)"}]]
```

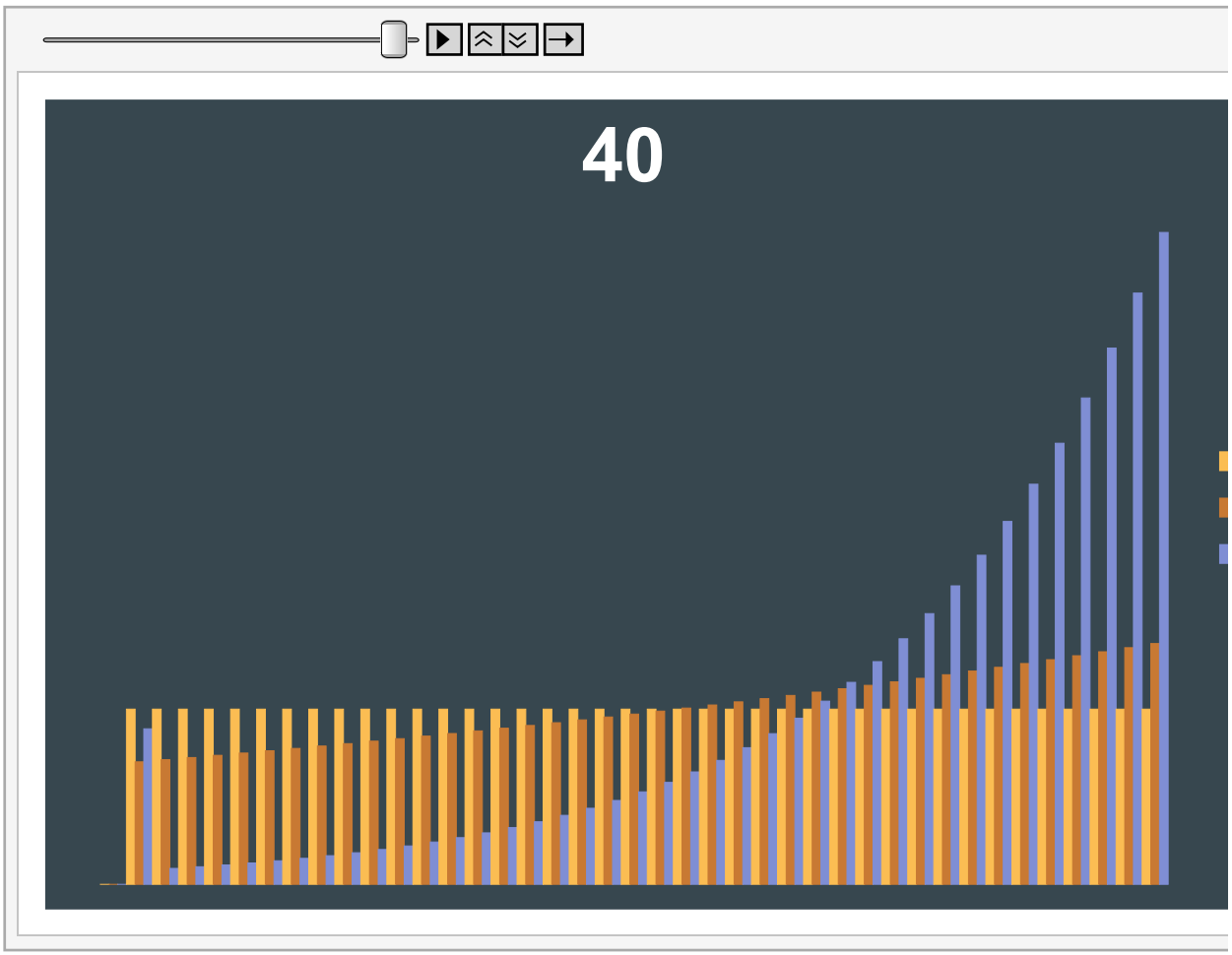
Out[123]=

history	soa	sma(2)	rwma(2)	ema(.5)
90	90.00	90.00	90.00	90.00
120	105.00	105.00	107.57	130.00
100	103.33	110.00	104.14	90.00
110	105.00	105.00	106.43	116.67

It can be useful to look at the animation of the weights as the parameter increases. Note how quickly the exponential drops off and how the simple moving average is likely to generate jumpier results because it incorporates new information rather than simply updating the weights.

In[199]:=
```
weights = Table[
    Framed[BarChart[Transpose@Table[CoefficientList[Expand@f[n][Table[x^i, {i, 40}]], x],
        {f, {sma, rwma, ema[Rescale[#, {1, 40}, {1, 20.5}]] &}}], ImageSize → 600,
      Axes → False, ChartLegends → (Style[#, 20, Bold, White] & /@ {"SMA", "RWMA", "EMA"}),
      PlotLabel → Style[n, 40, Bold, White], Background → slate],
     Background → slate, FrameStyle → slate], {n, 1, 40}];
Export[$WorkDirectory <> "Analytics.Bet/Code/weights.gif",
 weights, "DisplayDurations" → 1 / 2];
ListAnimate[weights]
```

Out[200]=



We also need to update our dataset. But rather than updating our dataset for a particular kind of moving average, let's be more general and just include for each game a list of the past totals for both the home and away team. That way, when we decide to change the kind of rolling average we use, or the parameter, we don't need to rebuild the dataset.

Generate and append to the dataset a list of historical total points scored by each team:

In[124]:=
```
historical = FoldList[ReplacePart[#,
      Table[{#2@ah, ah} → Append[#[#2@ah, ah], #2["actual"]], {ah, {"away", "home"}}]] &,
   AssociationMap[<|"home" → {}, "away" → {}|> &, Union@totes4[[All, "home"]]], totes4];
totes5 = MapThread[<|#, "historical" → #2|> &, {totes4, Most@historical}];
```

Display an example of an early game:

In[126]:=
```
totes5[[19]]
```

Out[126]=
```
⟨|date → {2020, 12, 25}, away → GoldenState, home → Milwaukee, market → 230.5,
 spread → 10, away_prob → 0.181818, market_h1 → 113, spread_h1 → 2.5, actual → 237,
 actual_home → 99, actual_away → 138, home_abbr → MIL, away_starters →
  {Stephen Curry, James Wiseman, Eric Paschall, Andrew Wiggins, Kelly Oubre},
 home_starters → {Giannis Antetokounmpo, Khris Middleton, Jrue Holiday,
   Donte DiVincenzo, Brook Lopez}, away_starters_ppg → 71.5879,
 home_starters_ppg → 88.9397, away_starters_ppm → 2.5788, home_starters_ppm → 2.84276,
 away_pts_avg → 113.838, home_pts_avg → 110.829, away_total_avg → 224.568,
 home_total_avg → 230., historical → ⟨|Atlanta → ⟨|home → {}, away → {228}|⟩,
   Boston → ⟨|home → {243, 218}, away → {}|⟩, Brooklyn → ⟨|home → {224}, away → {218}|⟩,
   Charlotte → ⟨|home → {}, away → {235}|⟩, Chicago → ⟨|home → {228}, away → {}|⟩,
   Cleveland → ⟨|home → {235}, away → {}|⟩, Dallas → ⟨|home → {}, away → {208, 253}|⟩,
   Denver → ⟨|home → {246, 229}, away → {}|⟩, Detroit → ⟨|home → {}, away → {212}|⟩,
   GoldenState → ⟨|home → {}, away → {224}|⟩, Houston → ⟨|home → {}, away → {}|⟩,
   Indiana → ⟨|home → {228}, away → {}|⟩, LAClippers → ⟨|home → {}, away → {225, 229}|⟩,
   LALakers → ⟨|home → {225, 253}, away → {}|⟩,
   Memphis → ⟨|home → {250}, away → {}|⟩, Miami → ⟨|home → {209}, away → {220}|⟩,
   Milwaukee → ⟨|home → {}, away → {243}|⟩, Minnesota → ⟨|home → {212}, away → {}|⟩,
   NewOrleans → ⟨|home → {}, away → {212, 209}|⟩, NewYork → ⟨|home → {}, away → {228}|⟩,
   OklahomaCity → ⟨|home → {}, away → {}|⟩, Orlando → ⟨|home → {220}, away → {}|⟩,
   Philadelphia → ⟨|home → {220}, away → {}|⟩, Phoenix → ⟨|home → {208}, away → {}|⟩,
   Portland → ⟨|home → {220}, away → {}|⟩, Sacramento → ⟨|home → {}, away → {246}|⟩,
   SanAntonio → ⟨|home → {}, away → {250}|⟩, Toronto → ⟨|home → {212}, away → {}|⟩,
   Utah → ⟨|home → {}, away → {220}|⟩, Washington → ⟨|home → {}, away → {220}|⟩|⟩|⟩
```

Notice some teams such as Houston have not played a single game yet while some such as Dallas and Denver have played multiple. Milwaukee and Golden State both played one away game before this one, but neither has played a home game. Thus, this game would not be in the sample because there is no way to generate an average total for Milwaukee as a home team yet.

Out-of-Sample Regressions and Backtests

We need a convenience function to apply a particular parameterized moving average to the historical data.

Define a function to generate the three columns of data needed for regressions based on a particular moving average function:

In[127]:=
```
RollingData[data_, f_, others_ : {}] :=
 Select[Join[f /@ {#historical[#away, "away"], #historical[#home, "home"]},
     Values@KeyTake[#, others], {#actual}] & /@ data, AllTrue[NumericQ]]
```

Run three linear regressions for three different moving averages:

In[128]:=
```
lms = Table[f → LinearModelFit[RollingData[totes5, f], {AvgAwayTotal, AvgHomeTotal},
      {AvgAwayTotal, AvgHomeTotal}], {f, {sma[1], sma[5], sma[∞]}}];
```

To evaluate and compare the three different models, we can use techniques such as residual sum of squares (RSS), calibration, shrinkage, and more.

Here, we will use RSS and its variants and derivatives such as Adjusted R-Squared. Note that the decision of which is better will basically always be the same between all of these metrics, whether it is because of a lower RSS or SD, or a higher Adjusted R Squared.

In[129]:=
```
⟨|"RSS" → #["ANOVATableSumsOfSquares"][[3]], "SD" → Sqrt[#["EstimatedVariance"]],
    "R²" → #["RSquared"], "AdjR²" → #["AdjustedRSquared"]|⟩ & /@ Association[lms] // Dataset
```

Out[129]=

	RSS	SD	R^2	AdjR^2
sma[1]	431914.	19.7082	0.0424438	0.0407216
sma[5]	417077.	19.3667	0.0753378	0.0736748
sma[∞]	418857.	19.408	0.0713902	0.06972

Another way to evaluate models, of course, is to compare their backtesting performance.

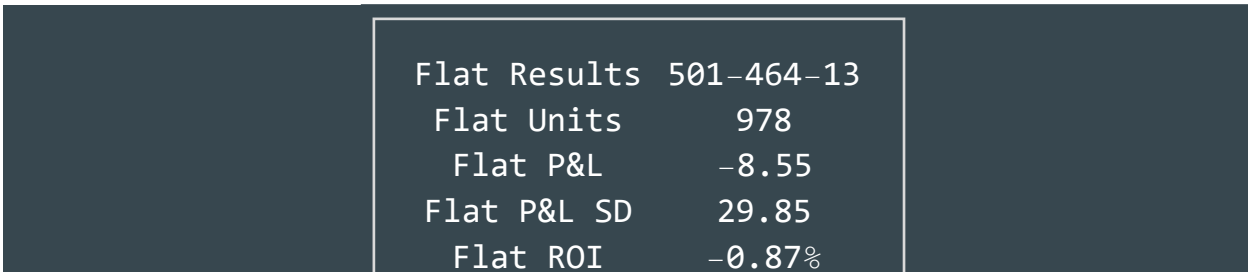
Run backtests using each of the three different regression models (based on the three different moving average functions):

In[130]:=
```
backtests = Table[
    Select[Backtest[totes5, Last@v, {AvgAwayTotal → First[v][#historical[#away, "away"]],
        AvgHomeTotal → First[v][#historical[#home, "home"]]} &],
     NumericQ[#model] &], {v, lms}]; // AbsoluteTiming
```

Out[130]= {15.5195, Null}

Summarize all three of the backtests:

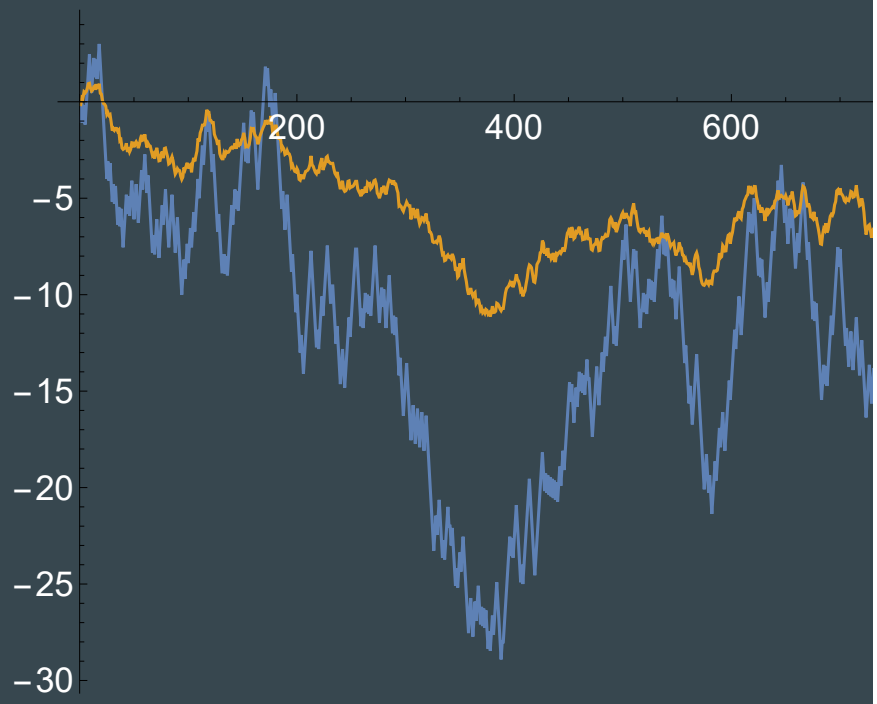
In[132]:=
```
save@
 Grid[MapThread[{Framed[Style[#1, 24, Bold, White], Background → slate, FrameStyle → slate],
     Summarize@#2} &,
   {{"SMA(1)", "SMA(5)", "SOA = SMA(∞)"}, backtests}], Background → slate]
```



Out[132]=

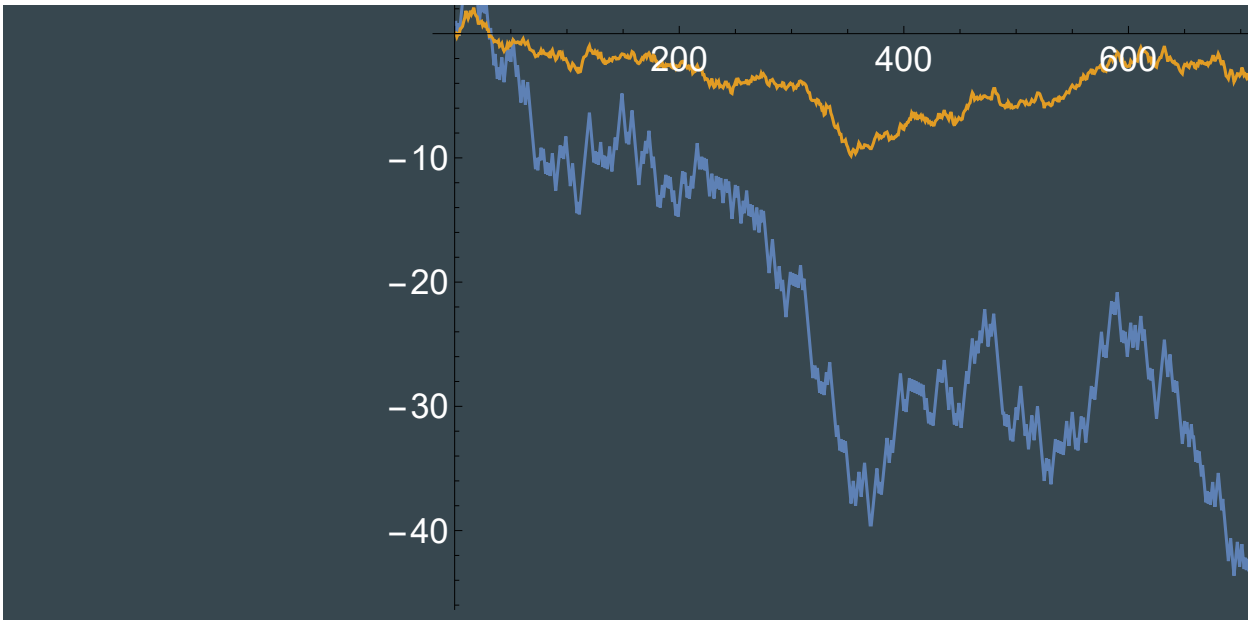
Kelly Units	221.63
Kelly P&L	-6.64
Kelly P&L SD	14.21
Kelly ROI	-3.00%

SMA (1)



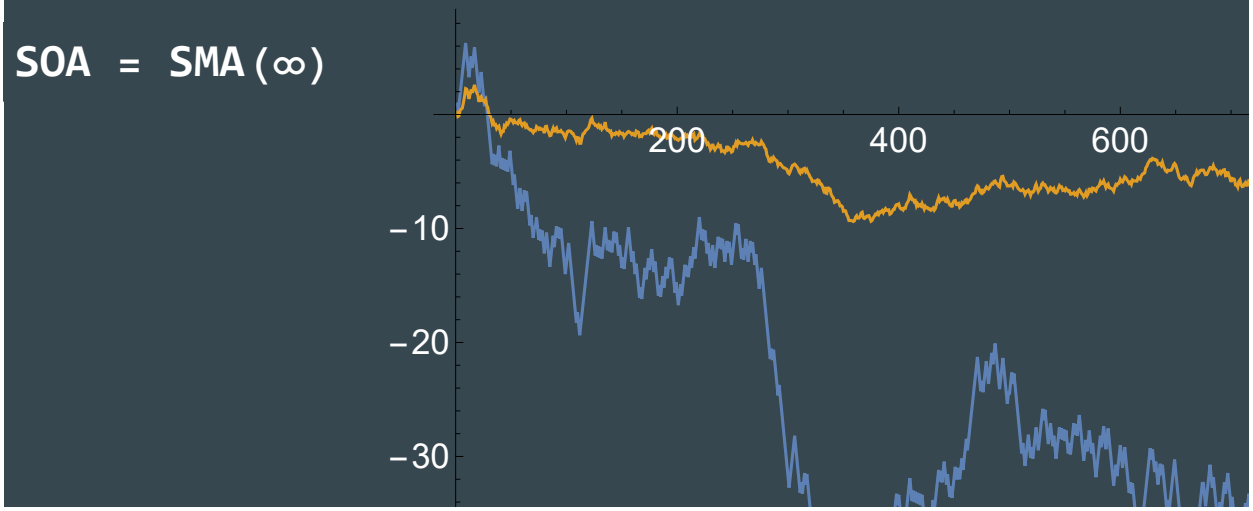
Flat Results	473-459-14
Flat Units	946
Flat P&L	-29.00
Flat P&L SD	29.36
Flat ROI	-3.07%
Kelly Units	189.63
Kelly P&L	-1.01
Kelly P&L SD	13.14
Kelly ROI	-0.53%

SMA (5)



Flat Results	480-468-11
Flat Units	959
Flat P&L	-31.64
Flat P&L SD	29.56
Flat ROI	-3.30%
Kelly Units	177.81
Kelly P&L	-6.68
Kelly P&L SD	12.73
Kelly ROI	-3.76%

SOA = SMA(∞)



C:/Users/pmaymin/Documents/My Work/Analytics.Bet/slides/img/2021071

Parameter tuning or fitting

Rather than focus on just three possible moving averages, all variants of the simple moving average, we can calculate all three types of moving averages across a variety of parameters.

Calculate the linear regression model for SMA[n] for n = 1, ..., 40.

In[143]:=
```
smas = Table[{n, LinearModelFit[RollingData[totes5, sma[n]], {AvgAwayTotal, AvgHomeTotal},
      {AvgAwayTotal, AvgHomeTotal}]}, {n, 40}]; // AbsoluteTiming
```

Out[143]=
```
{0.878321, Null}
```

Calculate the linear regression model for RWMA[n] for n = 1, ..., 40.

In[144]:=
```
rwmas =
   Table[{h, LinearModelFit[RollingData[totes5, rwma[h]], {AvgAwayTotal, AvgHomeTotal},
      {AvgAwayTotal, AvgHomeTotal}]}, {h, 40}]; // AbsoluteTiming
```

Out[144]=
```
{40.5839, Null}
```

Calculate the linear regression model for EMA[n] for n = 1, ..., 20.5 in steps of 0.5.

In[149]:=
```
emas = Table[{n, LinearModelFit[RollingData[totes5, ema[n]], {AvgAwayTotal, AvgHomeTotal},
      {AvgAwayTotal, AvgHomeTotal}]}, {n, 1, 20.5, .5}]; // AbsoluteTiming
```

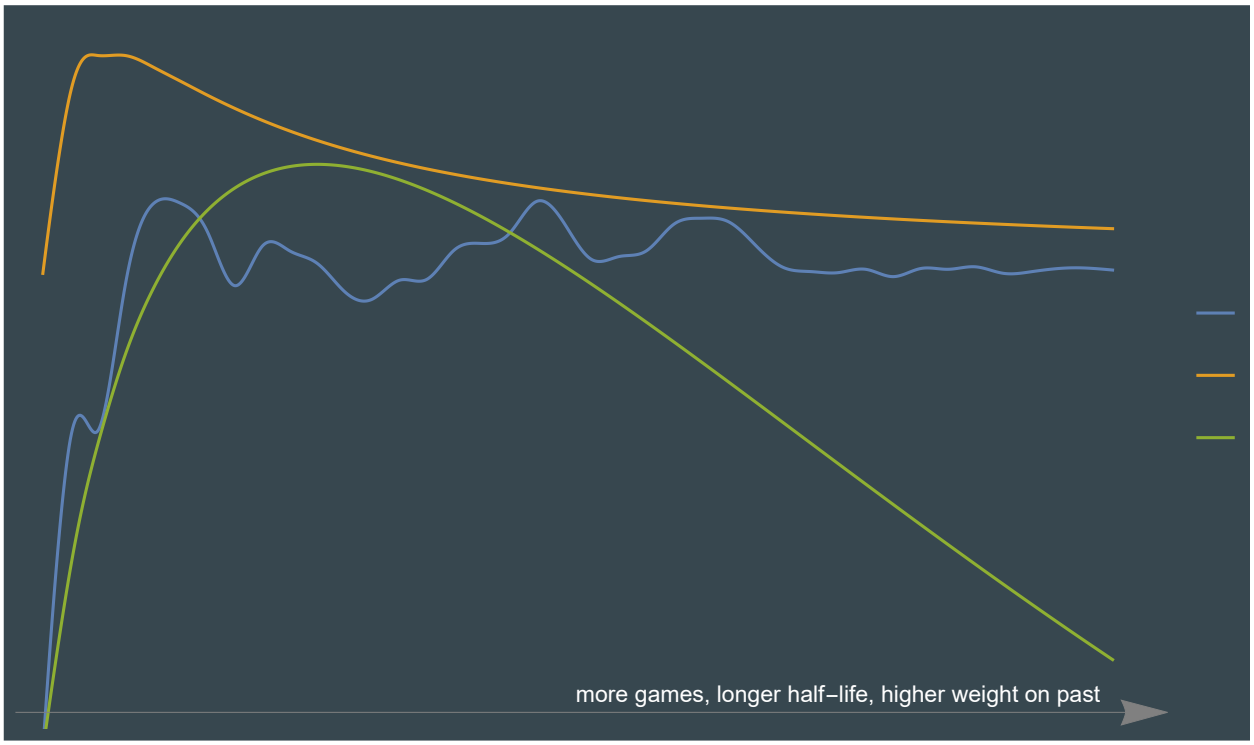
Out[149]=
```
{42.18, Null}
```

How does the adjusted R squared vary across parameters and families of moving averages?

Plot the AdjustedRSquared for each parameter and family, stretching the EMA from 2-21 to 2-40 for a better visual comparison:

In[249]:=
```
Framed[ListLinePlot[MapAt[#["AdjustedRSquared"] &,
   {smas, rwmas, MapAt[Rescale[#, {1, 20.5}, {1, 40}] &, emas, {All, 1}]}, {All, All, 2}],
  PlotRange → {{0, 42}, {.042, .085}}, PlotLegends → {"SMA", "RWMA", "EMA"},
  Epilog → {Gray, Arrow[{{0, .043}, {42, 0.043}}], Inset[Style[
      "more games, longer half-life, higher weight on past", 12, White], {30, .044}]},
  Axes → False, InterpolationOrder → 3, ImageSize → 600, LabelStyle → Directive[18, White],
  Background → slate], Background → slate, FrameStyle → slate]
```

Out[249]=



Backtesting

Often when doing time-intensive computations, it is useful to save the results to a file as you go so if things crash you can restart. A common trick is to see if the file for a particular calcluation already exists and to skip it if so.

Export CSV's of each backtest:

In[154]:=
```
Table[With[{fn = $WorkDirectory <> "Analytics.Bet/Data/BR/backtests/" <>
      ToString@First@v <> ".csv"}, If[! FileExistsQ@fn,
    Export[fn, Dataset[MapAt[#.{10000, 100, 1} &, Select[Backtest[totes5, Last@v,
        {AvgAwayTotal → First[v][#historical[#away, "away"]], AvgHomeTotal → First[v][
          #historical[#home, "home"]]} &], NumericQ[#model] &], {All, "date"}]]]],
  {v, Flatten[{MapAt[sma, smas, {All, 1}], MapAt[rwma, rwmas, {All, 1}],
    MapAt[ema, emas, {All, 1}]}, 1]}]; // AbsoluteTiming
```

Out[154]= {25.5641, Null}

Now read the CSV's back in:

In[155]:=
```
bigbacktests = KeySort[
    <|MapAt[ToExpression, StringSplit[StringReplace[FileBaseName@#, "]" → ""], "["], 2] →
       Normal@Import[#, "Dataset", "HeaderLines" → 1] & /@ FileNames[
      $WorkDirectory <> "Analytics.Bet/Data/BR/backtests/*.csv"]|>]; // AbsoluteTiming
```

Out[155]=
```
{6.41042, Null}
```

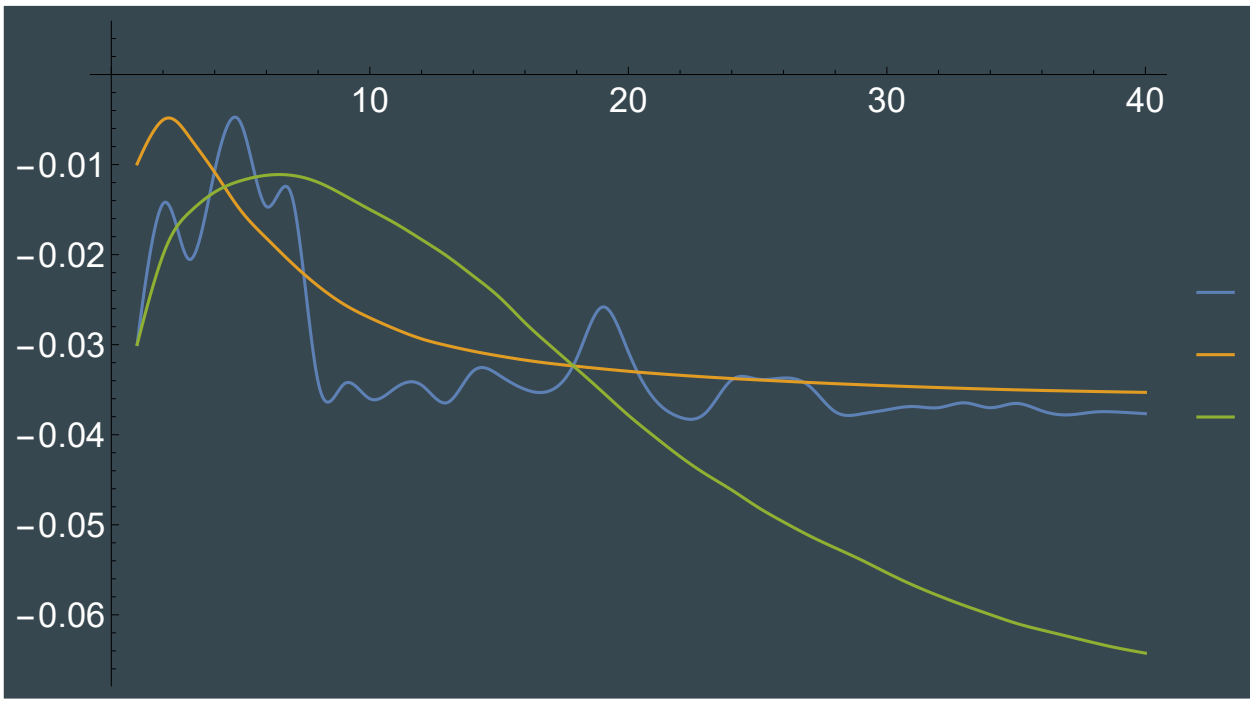
Extract just the Kelly ROI's from each backtest:

In[158]:=
```
rois = Summarize[#, False]@"Kelly ROI" & /@ bigbacktests;
```

Plot the Kelly ROI's, rescaling the EMA 1-20.5 to 1-40:

In[248]:=
```
Framed[ListLinePlot[
  <|Table[ToUpperCase@t → MapAt[If[t == "ema", Rescale[#, {1, 20.5}, {1, 40}], #] &,
      List @@@ Normal@KeyMap[Last, KeySelect[rois, First@# == t &]], {All, 1}],
    {t, {"sma", "rwma", "ema"}}]|>, ImageSize → 600, LabelStyle → Directive[18, White],
  InterpolationOrder → 3], Background → slate, FrameStyle → slate]
```

Out[248]=



Ad hoc calibration and extremes

We will discuss calibration in detail later but for now we can ask some reasonable motivating questions: how good is our model overall and in the tails?

We will need to pick a specific model. Based on the above graphs, something in the neighborhood of SMA(5) seems reasonable: a relatively myopic and equally weighted average of recent performance.

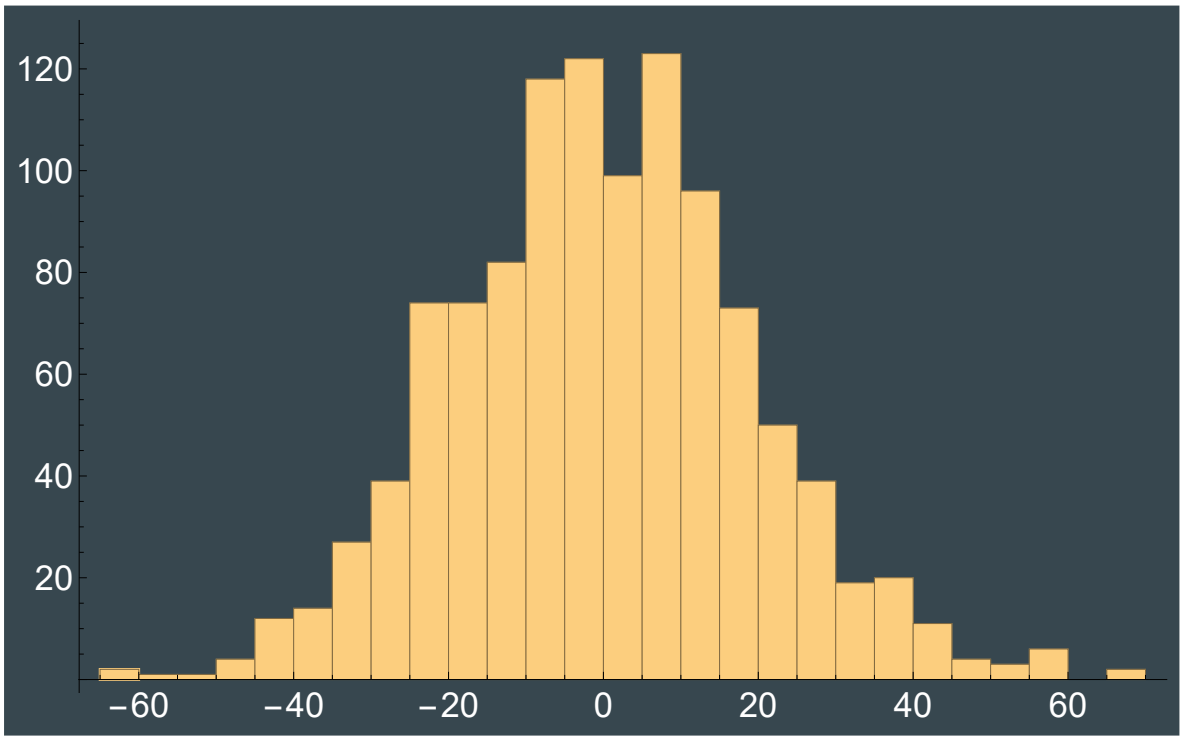
Calibration overall

How centered are our model estimates compared to the actual results?

Plot a histogram of the excess total over the model estimate for the SMA(5) backtest:

```
In[•]:= save@Framed[Histogram[#actual - #model & /@ bigbacktests[{"sma", 5}],
    LabelStyle → Directive[18, White], ImageSize → 600],
  Background → slate, FrameStyle → slate]
```

Out[•]=



C:/Users/pmaymin/Documents/My Work/Analytics.Bet/slides/img/20210711-133659.png

Calibration in the tails

When the model predicts very high or very low likelihood of success, is it accurate?

Count the number of times the actual total exceeded the model when the model probability of the over was above 65%:

```
In[•]:= #actual - #model & /@ Select[bigbacktests[{"sma", 5}], #po > .65 &] // Sign // Counts
```

Out[•]= ⟨| -1 → 112, 1 → 39 |⟩

Count the number of times the actual total exceeded the model when the model probability of the over was below 35%:

```
In[•]:= #actual - #model & /@ Select[bigbacktests[{"sma", 5}], #po < .35 &] // Sign // Counts
```

Out[•]= ⟨| 1 → 98, -1 → 48 |⟩

In[ ]:=
```
112 / 39.
```

Out[ ]=
```
2.87179
```

In both cases, the model is severely miscalibrated. If it were well-calibrated, we should see about half the games score over and half score under the model estimate.

Exploring the extremes

One useful exercise is looking at the extreme model predictions. One metric is to look at the games that had the highest model over probabilities.

In[ ]:=
```
Dataset@Take[ReverseSortBy[bigbacktests[{"sma", 5}], #po &][[1]], 6]
```

Out[ ]=

date	20201228
away	Houston
home	Denver
market	222
actual	235
model	239.616

The other useful metric is to look at the games that had the largest residual, meaning the difference between the actual and the model.

In[ ]:=
```
Dataset@Take[ReverseSortBy[bigbacktests[{"sma", 5}], Abs[#actual - #model] &][[1]], 6]
```

Out[ ]=

date	20210426
away	SanAntonio
home	Washington
market	231.5
actual	289
model	219.986

The two different metrics address different issues. Sorting by extreme probability highlights possible overconfidence by the model. Are there factors or modeling choices we can make to reduce that overconfidence? Sorting by extreme residuals on the other hand highlights possible missing information. The model was so wildly off, are we missing something crucial entirely?

Player-based team totals

What if we augment our SMA(5) model with player-specific points per game?

Regress SMA(5) home and away totals, and home and away starters PPG, on the actual totals:

In[•]:=
```
lm =
  LinearModelFit[RollingData[totes5, sma[5], {"away_starters_ppg", "home_starters_ppg"}],
   {AvgAwayTotal, AvgHomeTotal, AwayStarters, HomeStarters},
   {AvgAwayTotal, AvgHomeTotal, AwayStarters, HomeStarters}];
```

In[•]:=
```
showlm@lm
```

Out[•]=

	Estimate	Standard Error	t-Statistic	P-Value
1	72.2506	16.4551	4.39077	0.0000123739
AvgAwayTotal	0.400103	0.0505176	7.92008	5.73076×10^{-15}
AvgHomeTotal	0.201011	0.0567849	3.53986	0.000416918
AwayStarters	0.140393	0.04569	3.07272	0.00217285
HomeStarters	0.10248	0.0463604	2.2105	0.0272739
Estimated Standard Deviation: 19.2576				

Compare it to the regression results without the starters:

In[•]:=
```
showlm@smas[[4, 2]]
```

Out[•]=

	Estimate	Standard Error	t-Statistic	P-Value
1	80.5317	16.4044	4.90915	1.05141×10^{-6}
AvgAwayTotal	0.424676	0.0500971	8.47706	7.25269×10^{-17}
AvgHomeTotal	0.21565	0.0563355	3.82795	0.000136417
Estimated Standard Deviation: 19.3667				

The coefficients on the total's do not change much.

What if we try points-per-minute instead of points-per-game?

Regress SMA(5) home and away totals, and home and away starters PPM, on the actual totals:

In[•]:=
```
lm =
 LinearModelFit[RollingData[totes5, sma[5], {"away_starters_ppm", "home_starters_ppm"}],
  {AvgAwayTotal, AvgHomeTotal, AwayStarters, HomeStarters},
  {AvgAwayTotal, AvgHomeTotal, AwayStarters, HomeStarters}];
showlm@lm
```

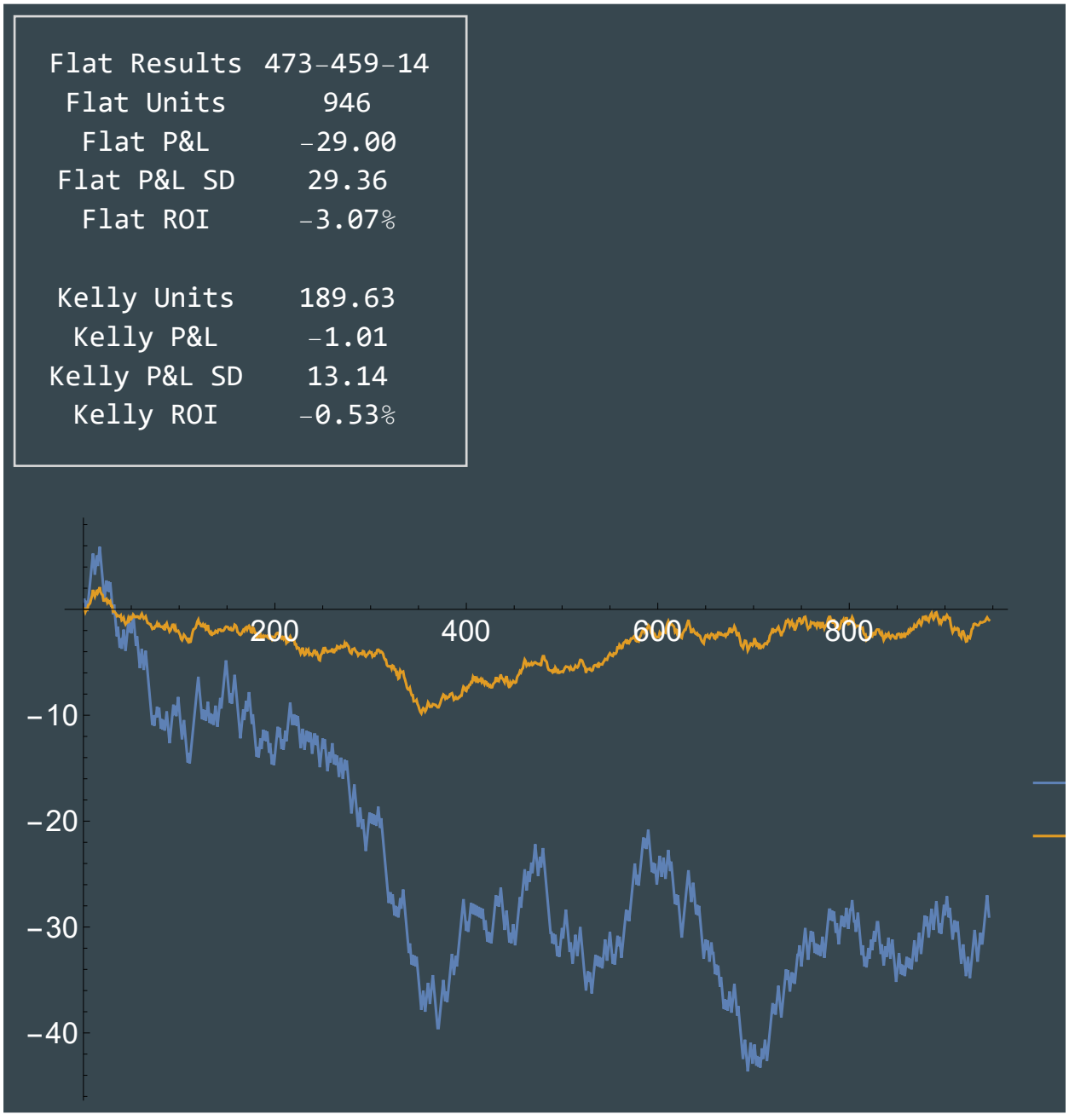
Out[•]=

	Estimate	Standard Error	t-Statistic	P-Value
1	66.6579	16.7869	3.97081	0.0000762372
AvgAwayTotal	0.396479	0.0507578	7.81121	1.30619×10^{-14}
AvgHomeTotal	0.208168	0.056754	3.6679	0.000256145
AwayStarters	5.84719	1.85284	3.15579	0.00164371
HomeStarters	3.41209	1.84028	1.85412	0.0639874
Estimated Standard Deviation: 19.2697				

Shrinkage

In[170]:=
```
save@Summarize[bigbacktests[{"sma", 5}]]
```

Out[170]=



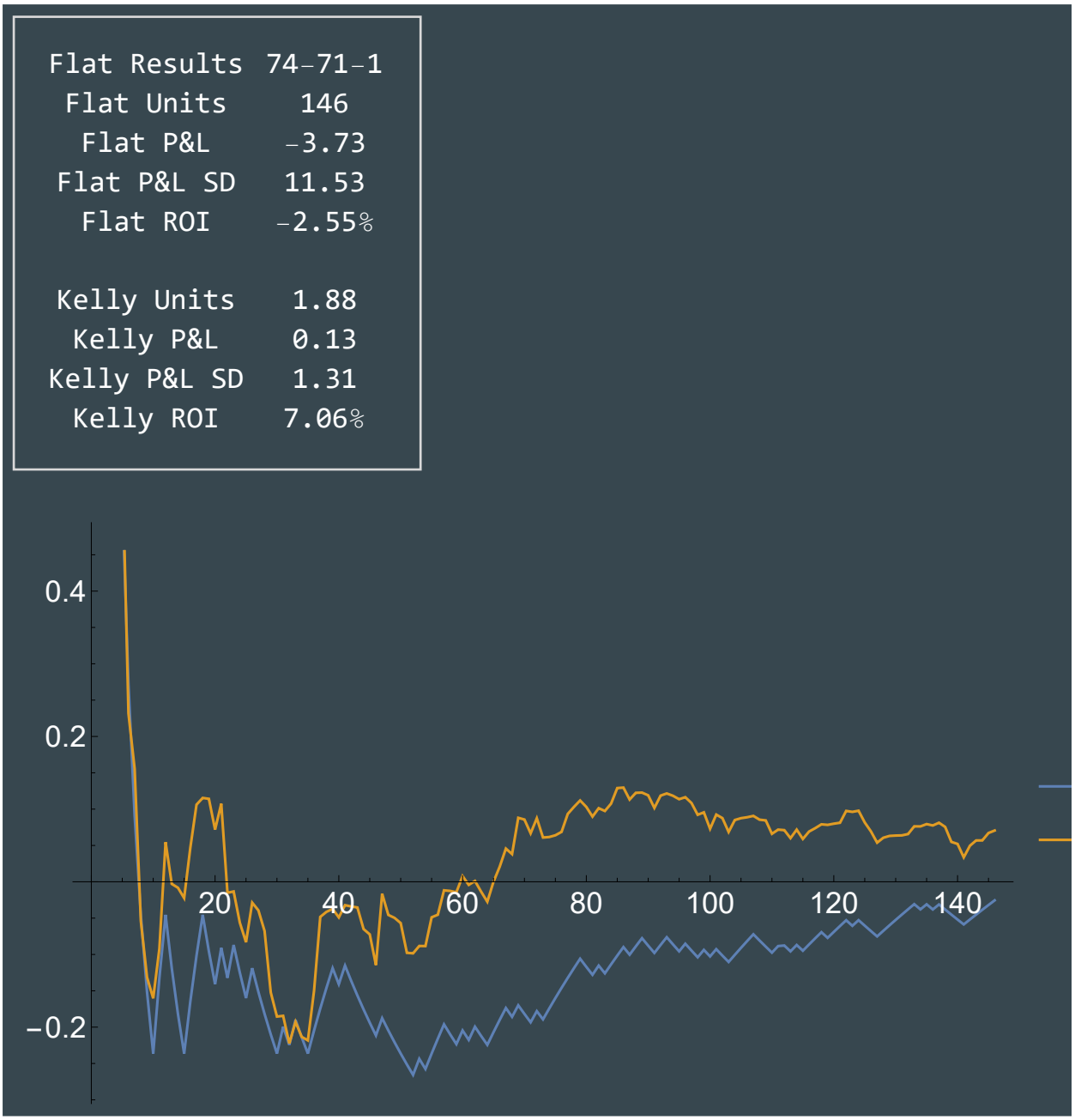
C:/Users/pmaymin/Documents/My Work/Analytics.Bet/slides/img/20210712-090947.png

In[167]:=
```
shrunk = Backtest[totes5, smas[[5, 2]], {AvgAwayTotal → sma[5][#historical[#away, "away"]],
    AvgHomeTotal → sma[5][#historical[#home, "home"]]} &, .114];
```

In[172]:=
```
save@Summarize[shrunk, True, True]
```

Flat Results	74–71–1
Flat Units	146
Flat P&L	–3.73
Flat P&L SD	11.53
Flat ROI	–2.55%
Kelly Units	1.88
Kelly P&L	0.13
Kelly P&L SD	1.31
Kelly ROI	7.06%

Out[172]=



C:/Users/pmaymin/Documents/My Work/Analytics.Bet/slides/img/20210712-091050.png

In[215]:=
```
data = Values@bigbacktests[{"sma", 5}][[All, {"model", "market", "actual"}]];
```

In[225]:=
```
shrink = Values@
  FindFit[data, b model + c market, {b, c}, {model, market}, NormFunction → (Norm[#, 1] &)]
```

Out[225]= {0.113559, 0.8871}

In[226]:=
```
regress = Values@
  FindFit[data, b model + c market, {b, c}, {model, market}, NormFunction → (Norm[#, 2] &)]
```

Out[226]=
```
{0.0909361, 0.909454}
```