ASSESSING THE IMPACT OF RELIEF PITCHING IN MAJOR LEAGUE BASEBALL

ASSESSING THE IMPACT OF RELIEF PITCHING IN MAJOR LEAGUE BASEBALL

By

BENJAMIN ANDREW MEYERSON

A Thesis Submitted to The W.A. Franke Honors College

In Partial Fulfillment of the Bachelors degree
With Honors in

Statistics and Data Science

THE UNIVERSITY OF ARIZONA

M A Y 2 0 2 3

Approved by:

Dr. Joe Watkins
Department of Mathematics

Assessing the Impact of Relief Pitching In Major League Baseball

Benjamin Meyerson
Advisor: Dr. Joe Watkins

Abstract:

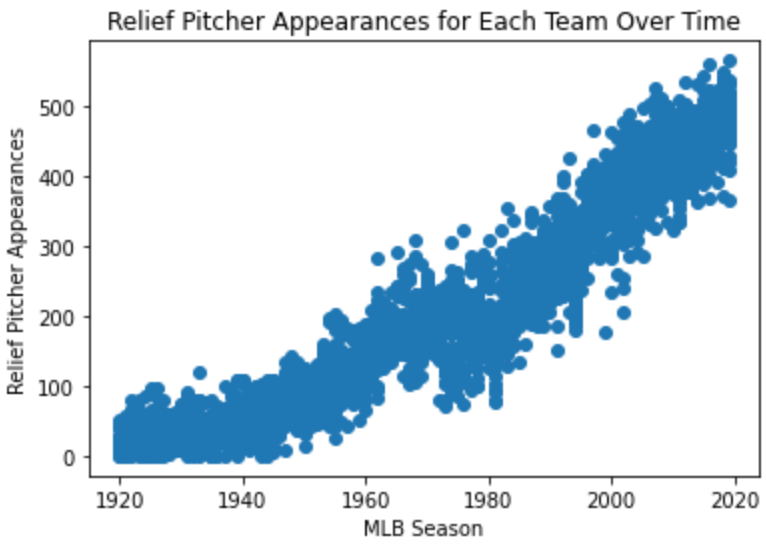
The usage of relief pitchers in Major League Baseball has been growing rapidly since the late 1970's, but relief pitchers' salaries have continued to be the lowest of any position group. This study investigates the importance of relief pitchers for MLB teams and how it has changed over time. This was done by creating linear regression models to assess the relationships between relief pitching performance metrics and teams' performance, allowing us to determine if each metric is significant, as well as how it impacts team performance. First, the relationships between the relief pitching metrics and teams' performance relative to their expected winning percentage were investigated, then the relationships between the relief pitching metrics and teams' raw winning percentage were analyzed as well. There were no meaningful relationships found between any relief pitching metrics and teams' performance relative to the expected winning percentage, but relief pitching depth was found to be the most significant predictor of teams' raw winning percentage of the metrics that were tested. This indicates that when MLB general managers are building their rosters, quality of depth in relief pitching should take a higher priority than bringing in a star. Additionally, it was found that increasing the quality of relief pitching depth tends to increase team wins at a rate of 6 additional wins for every 0.5 decrease in average runs allowed per nine innings, further highlighting the importance of quality relief pitching depth in the modern MLB.

Introduction:

Baseball is a massive industry in the United States with the top level of the professional leagues, Major League Baseball (MLB), bringing in nearly $11 billion in revenue in 2022 (Brown). With payrolls for MLB teams often eclipsing $150 million yearly, building a competitive roster of players is a game of resource allocation at a very large scale. Any money spent on one player can't be spent on another. This means that each team must come up with their own valuations of players and positions in order to build the best roster possible with the amount of money that they have.

One position that has often been seen as less valuable over the years is that of the relief pitcher. Relief pitchers, or relievers, are the pitchers that enter the game after the starting pitcher, or starter, leaves. This lower valuation could be for any number of reasons including injury, fatigue, poor performance, or that the job of a typical reliever is to just pitch one or potentially two innings. The low number of innings pitched is drastically different from starting pitchers, who are generally expected to pitch between five and seven innings in today's game. However, relief pitchers will pitch much more often than their starting counterparts who only pitch once every five or six games. Because teams carry many relief pitchers on their roster and they throw fewer innings than starting pitchers, they generally pay their relievers much lower salaries than their starting pitchers (MLB Positional Payrolls).

Though traditional thinking about relief pitchers may have been reasonable in previous seasons, the number of appearances by relief pitchers has been steadily climbing in every MLB season since the 1970's as shown in the graph below, indicating an increased importance in the game.



This leads to the question of whether or not relief pitchers should have more value placed on them. If MLB teams are increasing relief pitchers' usage, should they also be increasing their value relative to their teammates?

Currently, much of relievers' value comes in their specialization. Because they only face a few batters at a time, relievers can afford to have more gimmicks than a starting pitcher, such as much higher pitch velocities or more unusual windups. If utilized for too many pitches at a time, these could be detrimental to a pitcher's health and their performance, but for only a small amount of pitches, they can be highly effective. Additionally, because relievers only enter the game with the intention of pitching to a few batters, the team's manager can afford to create a more specific plan for how to pitch to those batters. This, combined with the decreasing effectiveness of starting pitchers as the game goes on (Whiteside), has played a large part in MLB teams choosing to give more appearances to their relief pitchers in order to put themselves in the best position to win games.

An inevitable consequence of relief pitchers' specialization is that they are much harder to evaluate using traditional statistical methods. Most positions on a baseball team can be evaluated using a statistics called Wins Above Replacement (WAR), which measures the number of additional wins that a player added to their team over what would be expected from a replacement level player (Wins Above Replacement). There are no perfect statistics, and WAR shouldn't be looked at on its own to evaluate any players, but it helps to illustrate the issue of evaluating relief pitchers because they do not pitch enough innings to compare to starting pitchers or other position players. Consequently, relief pitchers are undervalued by WAR, and many other cumulative statistics, because they do not play the same volume of innings that other players do. On the other hand, the innings that the best relievers do pitch in tend to be later in the game and when the score is close, making them some of the most consequential innings. This is the central issue with evaluating elite relief pitchers. Though they do not pitch as many innings as a starting pitcher, their innings tend to be in higher leverage situations (Slowinski).

In an attempt to remedy this issue, this study will seek to evaluate relief pitchers purely based on their relationship with wins. This will first be done by completing a regression analysis using different features of relief pitching to predict teams' winning percentage over what would be expected based on their run differential. In this analysis, the study is looking to quantify the amount of wins that quality relief pitchers add to a team with a static run differential, and thus, quantify those relief pitchers' "added wins." After that, another regression analysis will be completed using teams' relief pitching statistics to predict their winning percentage directly. This is a more simplistic but more

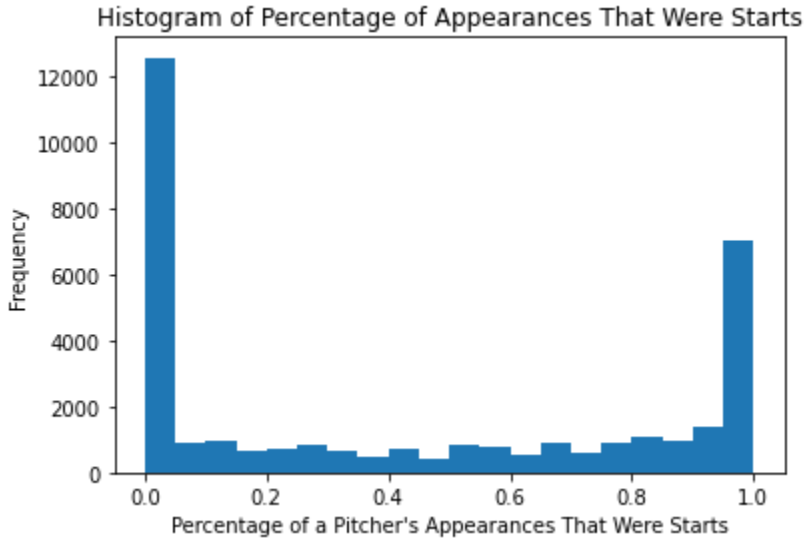
powerful approach than the first one, yielding more easily interpretable results, making it a worthwhile step.

Methods:

The first step in this analysis is aggregating historical data on MLB teams' records and their relief pitchers from each year. The information on historical team performance and run differential was taken from the Lahman database, which has a record of many professional baseball statistics going back to 1871 (Lahman). The Lahman database does not delve into advanced or era-adjusted statistics for pitchers, so that data was sourced from fangraphs, another major source for historical baseball statistics (Major League Leaderboards). From here, season level, era-adjusted performance statistics for every player to throw a pitch in the MLB was collected and added to the dataset. While data from as far back as 1871 was available in both data sources, it was decided that only data from 1920 - 2019 would be included in the final dataset. This is due to a large set of rule changes that took place in professional baseball in 1920, changing the game drastically and beginning what many people now call the "live ball era" (Vincent). For the other side of the bounds, 2019 was chosen as the most recent year to include in the dataset in order to avoid including the shortened 2020 season and the rule changes that were instituted to combat the effects of the COVID-19 pandemic.

After the two general datasets were collected, we began the process of merging them into a single dataset with one row per team season. The first step in this process was to eliminate the starting pitchers from the dataset, leaving only relief pitchers. There is no definitive way to differentiate between relief and starting pitchers because players

will often fulfill both roles at different points in the season, so we had to define our own metric for classification. We did this by examining the percentage of games started by a pitcher, with the goal of finding a point where pitchers who started a higher percentage of games were considered to be starters and those who started a lower percentage of games would be considered relievers. By inspecting the histogram displaying the distribution of percentage of appearances that were starts, shown below, we were able to see a dip in frequency after the 15% bar, making that our cutoff point for determining which players are relief pitchers and which are starters.



After eliminating the starting pitchers from the dataset, we still had to combine the individual relief pitchers' statistics into some summary statistics for the team to get a final dataset with only one row per team season. In order to get a good view of each team's relief pitchers, we decided to record 11 different measures of relief pitching depth, quality, and usage. The specific metrics recorded have been listed and defined in Appendix A. It was important to include data regarding the quality of a team's top end relief pitching due to the fact that a teams best relievers will most often be the ones

inserted into high leverage situations, giving them the greatest opportunity to impact wins and losses. On the other hand, the depth of a relief pitching corps is also incredibly important to teams' long term efforts due to the impact of injuries and runs of bad games within the top relief pitchers over the course of the season. For these reasons, we decided to include measures of both top end quality and relief pitching depth as potential predictor variables to investigate.

The consolidation of relief pitching and team data into a single dataset quickly revealed that although seasonal boundaries had been set to avoid including statistics from seasons with vastly different rules, we had failed to consider the change in strategy and gameplay within the mostly static set of rules. Relief pitchers were not a large part of the game of baseball until well after 1920, with starting pitchers expected to pitch the entire game. This meant our dataset had many null values in the early years because teams would go entire seasons without a single player who fit our definition of a relief pitcher throwing a pitch for them. To combat this, we decided to further restrict the seasons included in the dataset to include only data from the seasons 1950-2019. The removal of all seasons prior to 1950 eliminated the majority of the seasons in which teams did not use a single relief pitcher, but it also eliminated the seasons played during World War 2, a period where many of the best players from the MLB were drafted into military service.

The analysis of this data will be accomplished in two parts. First, we will use linear regression to determine the relationship between the compiled relief pitching metrics and teams' winning percentage above what would be expected based on their run differential. After that, we will use the same regression methods to determine the

relationship between the relief pitching metrics and teams' winning percentage for the season. Within each of these methods, the dataset will be split into an "early" dataset containing data from the seasons between 1950 and 1999, and a "modern" dataset containing data from the seasons between 2000 and 2019. The methods will be run on both datasets to see if the general trends or strength of trends has changed between the two time periods.

The first method is meant to test the hypothesis that a team with good relief pitchers will perform better in close games, leading them to have a better winning percentage than would be expected for their run differential. This hypothesis is based on the idea that the high leverage innings that relief pitchers typically pitch have more of an impact on a team's winning percentage than low leverage innings. To test this, we start by calculating each team's expected winning percentage by running a linear regression using run differential to predict winning percentage, then plugging the team's run differential into the resultant equation. We then subtract this number from the team's actual winning percentage to get their winning percentage over expected (the team's residual from the previous regression). We will then use the relief pitching metrics for each team to attempt to predict their winning percentage over expected through a regression model. If we find significant results in this model, our hypothesis would be proven correct and we could use the regression equation to predict the amount of additional wins adding a different relief pitcher would provide a team provided that their run differential stays constant.

The second method of analysis is more direct in its approach in simply using regression to find the relationship between the relief pitching metrics and teams' winning

percentages. While it is more straightforward, by removing the dependency on run differential from the response variable, this method takes into account that relief pitchers will impact a team's run differential over the course of the season. Significant results in this regression model would show how changing the performance of a team's relief pitchers impacts their winning percentage directly. While more directly interpretable than the first method of analysis, this method is more broad and doesn't attempt to explain any specific part of a relief pitcher's impact, just their overall impact on teams' winning percentage.

Results:

When looking at the relationships between teams' winning percentage over what would be expected based on their run differential and their relief pitching performance metrics, the first step was to visualize each individual relationship. This was done by creating a scatterplot for each individual relief pitching metric with winning percentage over expected and the relief pitching metric as the axes. We began by doing this for the early dataset, and the resulting scatter plots can be seen in appendix B. These charts did not show a strong relationship between any of the metrics and winning percentage over expected, with none of them having any obvious discernible pattern in their shape. This was further evidenced by the low correlation values for these relationships, with the largest absolute correlation value being only about 0.11 (RP1_Apps, see appendix A for feature definitions), with multiple other relief pitching metrics having much weaker correlations.

This shows that there is a barely discernible relationship between the relief pitching metrics and teams' winning percentage over expected in the early dataset, but

in order to get a better understanding of the modern usage of relief pitchers, the same procedure was run for the modern dataset. After creating the same scatter plots as before, shown in appendix C, it can be seen that the plots show slightly stronger relationships than were seen in the full dataset, particularly in the ERA- and FIP- metrics. Though not immediately striking, these stronger visual relationships were backed up by the correlation values between the relief pitching metrics and winning percentage over expected. Four of the relief pitching metrics showed absolute correlation values above 0.14, larger than the highest correlation value of 0.11 from the early dataset.

In order to further measure these relationships, linear regressions were run with each individual statistic used to predict winning percentage over expected. To make the comparison of effect sizes more clear, the predictors were all standardized prior to the regression being run. Similar to the visualizations, this analysis was first run using the early dataset, the results of which are shown in the table below.

	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-statistic	F-test P-value
num_RP_used	-0.0003	0.6697	-0.0007	0.1821	0.6697
total_RP_Apps	0.0009	0.2310	0.0004	1.4361	0.2310
RP1_Apps	0.0028	0.0001	0.0120	14.8831	0.0001
RP1_ERA-	-0.0020	0.0051	0.0060	7.8645	0.0051
RP1_FIP-	-0.0018	0.0145	0.0044	5.9899	0.0145
best_RP_ERA-	-0.0005	0.4674	-0.0004	0.5286	0.4674
best_RP_FIP-	-0.0013	0.0794	0.0018	3.0829	0.0794
top3Apps_ERA-	-0.0014	0.0590	0.0022	3.5729	0.0590
top3Apps_FIP-	-0.0010	0.1632	0.0008	1.9465	0.1632
top5Apps_ERA-	-0.0001	0.8695	-0.0009	0.0270	0.8695
top5Apps_FIP-	-0.0006	0.3767	-0.0002	0.7820	0.3767

Similar to what was seen in the visualizations and correlations, the results from the early dataset showed many weak relationships. Once again, in order to compare the results between different time periods, the analysis was run a second time using the

modern dataset. The results of the second set of linear regressions are shown in the following table.

	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-statistic	F-test P-value
num_RP_used	-0.0010	0.3288	-0.0001	0.9551	0.3288
total_RP_Apps	0.0004	0.6931	-0.0014	0.1559	0.6931
RP1_Apps	0.0027	0.0071	0.0104	7.2906	0.0071
RP1_ERA-	-0.0021	0.0398	0.0054	4.2453	0.0398
RP1_FIP-	-0.0022	0.0271	0.0065	4.9082	0.0271
best_RP_ERA-	-0.0035	0.0004	0.0193	12.7667	0.0004
best_RP_FIP-	-0.0032	0.0015	0.0151	10.1912	0.0015
top3Apps_ERA-	-0.0036	0.0003	0.0200	13.2147	0.0003
top3Apps_FIP-	-0.0034	0.0007	0.0174	11.6177	0.0007
top5Apps_ERA-	-0.0035	0.0004	0.0189	12.5362	0.0004
top5Apps_FIP-	-0.0035	0.0004	0.0192	12.7278	0.0004

After completing the regression analysis with winning percentage over expected as the response variable, the same procedure was completed with teams' unadjusted winning percentage as the response variable. As before, the first step of this analysis was visualizing the data using the early dataset, so scatter plots with one of the relief pitching statistics on the vertical axis and teams' winning percentage on the horizontal axis were created to look for any strong patterns. The resulting charts (appendix D) show much stronger visual relationships between the relief pitching metrics and winning percentage than was seen when looking at winning percentage over expected, especially when looking at the ERA- related metrics. This is further shown when looking at the correlation values, where the lowest absolute correlation between winning percentage and one of the ERA- related metrics is greater than 0.26, which is significantly higher than even the best values for winning percentage over expected.

Similar to what was seen in the analysis of winning percentage over expected, when looking at the modern dataset, the visual relationships between individual relief pitching metrics and winning percentage were much more apparent. In this set of

scatterplots, seen in appendix E, similar to the ones showing the early dataset, the ERA- related metrics stand out as having the strongest visual relationships, though in the plots with only modern data, RP1_ERA- does not stand out as much as the others. The absolute correlation values for top3Apps_ERA- and top5Apps_ERA- were a step above the rest at 0.49 and 0.51 respectively, indicating that these two metrics had the strongest relationships with winning percentage and highlighting them as potentially important predictors.

Finally, linear regressions were run with each individual relief pitching metric used as a predictor of winning percentage in single predictor models utilizing the early dataset and the modern dataset. As in the previous regression analysis, the predictors were standardized beforehand to make comparisons between coefficients more clear. The results of the regression models that were given the early dataset are shown in the table below.

	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-statistic	F-test P-value
num_RP_used	-0.0069	0.0015	0.0080	10.1662	0.0015
total_RP_Apps	-0.0009	0.6819	-0.0007	0.1680	0.6819
RP1_Apps	0.0045	0.0393	0.0028	4.2562	0.0393
RP1_ERA-	-0.0199	0.0000	0.0719	89.4181	0.0000
RP1_FIP-	-0.0155	0.0000	0.0433	52.7015	0.0000
best_RP_ERA-	-0.0201	0.0000	0.0733	91.2700	0.0000
best_RP_FIP-	-0.0141	0.0000	0.0355	42.9397	0.0000
top3Apps_ERA-	-0.0251	0.0000	0.1149	149.1810	0.0000
top3Apps_FIP-	-0.0184	0.0000	0.0611	75.1982	0.0000
top5Apps_ERA-	-0.0262	0.0000	0.1254	164.5882	0.0000
top5Apps_FIP-	-0.0196	0.0000	0.0694	86.0550	0.0000

Though there are some non-negligible relationships identified in the early dataset, the same regression analysis was also run on the modern dataset in order to compare the two sets of results. The resulting regression models and some of their relevant attributes can be seen in the table below.

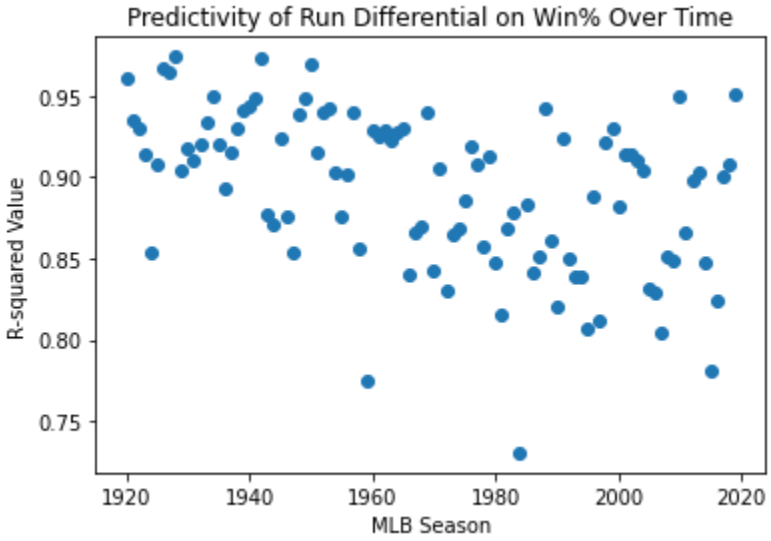
	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-statistic	F-test P-value
num_RP_used	-0.0152	0.0000	0.0419	27.1887	0.0000
total_RP_Apps	-0.0001	0.9742	-0.0017	0.0010	0.9742
RP1_Apps	0.0131	0.0000	0.0311	20.2152	0.0000
RP1_ERA-	-0.0233	0.0000	0.1015	68.6826	0.0000
RP1_FIP-	-0.0171	0.0000	0.0538	35.0374	0.0000
best_RP_ERA-	-0.0301	0.0000	0.1701	123.7873	0.0000
best_RP_FIP-	-0.0253	0.0000	0.1199	82.6098	0.0000
top3Apps_ERA-	-0.0358	0.0000	0.2417	191.8939	0.0000
top3Apps_FIP-	-0.0292	0.0000	0.1603	115.3435	0.0000
top5Apps_ERA-	-0.0369	0.0000	0.2571	208.2579	0.0000
top5Apps_FIP-	-0.0311	0.0000	0.1813	133.6338	0.0000

Discussion:

The results of the regression models showed very weak relationships between the individual relief pitching metrics and winning percentage over expected within the early dataset. There were only three metrics that were shown to be significant predictors at the 95% confidence level, RP1_Apps, RP1_ERA-, and RP1_FIP-. Even so, the adjusted R^2 values of the three statistically significant models are too low to be meaningful with the highest of them (RP1_Apps) being only 0.012, showing that the best of the models is only able to explain 1.2% of the variation in winning percentage over expected. Though the regression models trained using the modern dataset displayed more significant results than those that used the early dataset, with all of the metrics besides num_RP_used and total_RP_Apps being significant predictors at the 95% confidence level, they had the same issues with low adjusted R^2 values. In these modern regression models, the highest adjusted R^2 value was seen in the model with top3Apps_ERA- being used to predict winning percentage over expected, and was only 0.02. Once again, while this is higher than what was seen in the models trained using

the early dataset, explaining 2% of the variation in winning percentage over expected is not enough to justify the use of the model.

Because neither the models based on the early dataset nor those based on the modern dataset were able to explain a non-negligible percentage of the variation in winning percentage over expected, the initial hypothesis could not be proven. The performance and usage of relief pitchers were not able to explain a meaningful amount of teams' performance relative to their run differential. Part of this can likely be explained by the fact that run differential alone is a very good predictor of winning percentage, with the plot below showing that in every season since the start of the live ball era (1920) run differential has been able to explain greater than 70% of the variation in teams' winning percentage.



This means that once run differential is accounted for by the winning percentage over expected formula, there is very little explainable variation left to predict. Additionally, a large portion of the remaining unexplained variance can likely be attributed to the randomness of the sport of baseball, rather than any quantifiable

statistic. As with all other sports, the impact of randomness introduced by the human aspect of baseball cannot be ignored.

When looking at the results of the regressions that attempted to predict winning percentage using the relief pitching metrics, a stark difference can be seen between the adjusted R^2 values of those that used the early dataset and those that used the modern dataset. In both sets of models, every metric except for total_RP_Apps was a significant predictor of winning percentage at the 95% confidence level, but the models that used the modern dataset often showed adjusted R^2 values over double those of the models that used the early dataset. This indicates that in the modern MLB, relief pitching performance and usage has much more of an impact on winning percentage than it previously had, a logical conclusion based on the drastic increase in the usage of relief pitchers.

In both sets of models, it can be seen that relief pitching depth as measured by ERA- is the best predictor of winning percentage. The highest adjusted R^2 value in both sets of models being from the model using top5Apps_ERA- as a predictor, followed by the model using top3Apps_ERA- as a predictor lend credence to this fact. The best model trained using the modern dataset (top5Apps_ERA-) has an adjusted R^2 value of 0.2571, meaning that 25.7% of the variation in MLB winning percentage can be explained by the average ERA- of the top five relief pitchers weighted by the number of appearances that they made in the season, making this a very meaningful model. After asserting that the model is meaningful, the coefficients can be examined by looking at the equation below.

$$Win\% = 0.5 - 0.0369(top5AppsERA -)$$

The intercept coefficient for this model is 0.5, a number that should be expected based on the fact that the average winning percentage in the MLB will always be 0.5 and the predictors have been standardized, and thus, they have a mean of 0. Because ERA- is structured in a way where a lower value indicates better performance, the metric specific coefficient for the model shows that for every 13.4% (one standard deviation of ERA-) better that a team's top five relief pitchers perform relative to league average (around 0.5 fewer runs allowed per nine innings, depending on the year), their winning percentage would be expected to increase by 0.0369 points. This increase in winning percentage translates to about six additional games won over the course of the 162 game MLB season, which for many teams, could be the difference between making the playoffs and not.

ERA- being the relief pitching metric that is most predictive of winning percentage is not surprising because it is based on the number of runs given up by the pitchers and less runs conceded will generally lead to more wins. However, when applying the results to the actual building of an MLB roster, it should be noted that a pitcher's past FIP- is more predictive of their future ERA- than their past ERA- is (Richards). This means that even though this analysis suggests that a team should look to optimize the average ERA- of their top five relief pitchers in order to most effectively improve their winning percentage through relief pitching, it is more important to look at potential additions' FIP- than their ERA-. This is because FIP- attempts to eliminate the impact of team defense, something that is not very consistent across seasons and has a large impact on ERA- even though it is out of the pitcher's control.

Though this study was able to produce some meaningful results regarding the importance of relief pitching in the MLB and how it has changed over time, there were some limitations in the data used. First, data was aggregated at a season level for each pitcher, so it was often impossible to discern which pitchers threw the most high leverage innings for each team. This made it difficult to truly test the hypothesis that teams with better relief pitchers will win more close games because they will have the advantage in those high leverage innings. While data aggregated at the season level can give an overview of the relief pitchers' performances, data that recorded the situation and outcomes for each pitcher appearance could potentially be used to better assess the hypothesis. Additionally, nothing but pitching outcomes were accessible in the dataset. The outcomes and performance of pitchers are certainly among the most important factors to consider when looking at the quality of a team's relief pitching, but other factors may have also been important to consider. For example, whether a pitcher is right or left handed as well as the types of pitches that they are able to throw are commonly used as reasons that MLB general managers will sign someone, so adding those factors to the models may have benefitted them. Future studies could consider assessing the importance of those factors on teams' relief pitching success as well as their winning percentage to determine how impactful they are compared to their perceived importance, or if they are even important at all.

Acknowledgements:

This work was completed as one of the requirements for a Bachelors of Science in Statistics and Data Science with Honors from the College of Science at the University of Arizona. Special thanks goes to Dr. Joe Watkins for advising me through the process

of my work as well as Sean Lahman and the FanGraphs Baseball team for the use of their historical baseball databases.

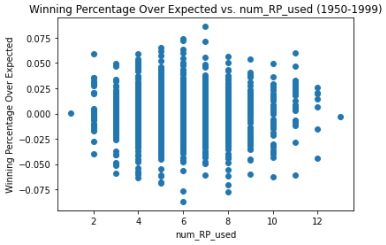
Appendices:

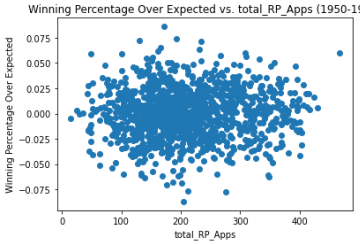
Appendix A: Relief Pitching Metrics Recorded With Definitions

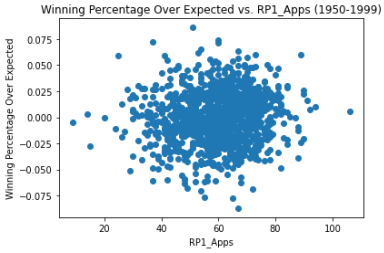
Metric Name in Code	Metric Definition
num_RP_used	Total number of relief pitchers used by a team in a season
total_RP_Apps	Total number of appearances made by relief pitchers for a team in a season
RP1_Apps	Number of appearances made by the most frequently used relief pitcher on a team during the season
RP1_ERA-	ERA- statistic of the most frequently used relief pitcher on a team
RP1_FIP-	FIP- statistic of the most frequently used relief pitcher on a team
best_RP_ERA-	Best ERA- statistic from the five most frequently used relief pitchers on a team
best_RP_FIP-	Best FIP- statistic from the five most frequently used relief pitchers on a team
top3Apps_ERA-	Weighted average of ERA- statistics from the three most frequently used relief pitchers where the weights are given by the number of appearances made by each pitcher
top3Apps_FIP-	Weighted average of FIP- statistics from the three most frequently used relief pitchers where the weights are given by the number of appearances made by each pitcher
top5Apps_ERA-	Weighted average of ERA- statistics from the five most frequently used relief pitchers where the weights are given by the number of appearances made by each pitcher
top5Apps_FIP-	Weighted average of FIP- statistics from the five most frequently used relief pitchers where the weights are given by the number of appearances made by each pitcher

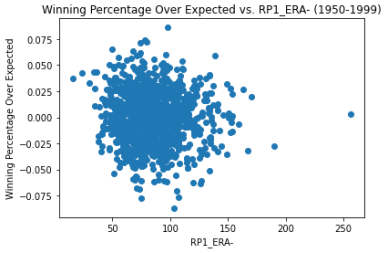
Appendix B: Relief Pitching Metrics vs. Winning Percentage Over Expected

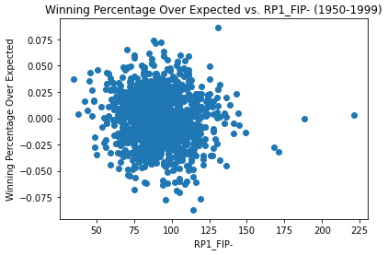
Scatterplots (Early Dataset)

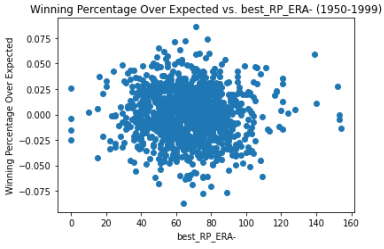


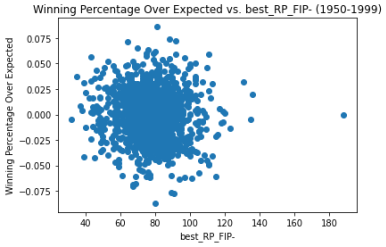


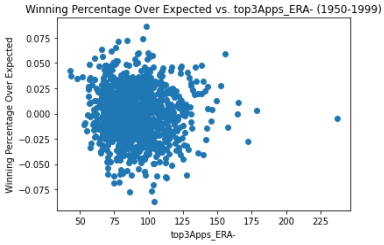


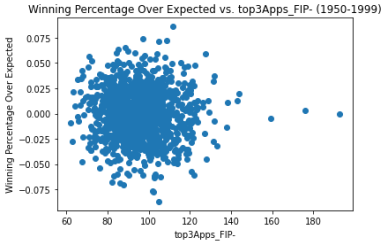


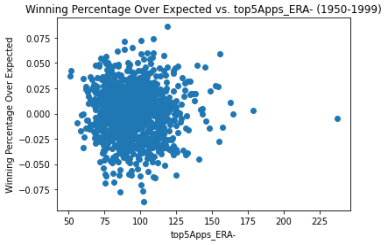


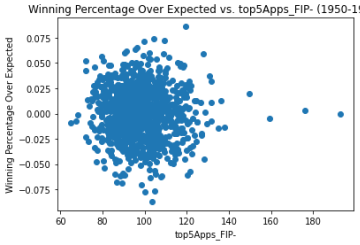






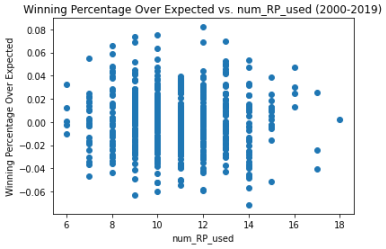


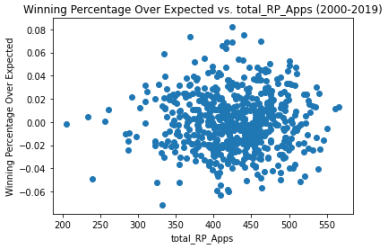


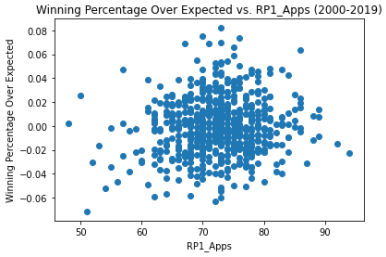


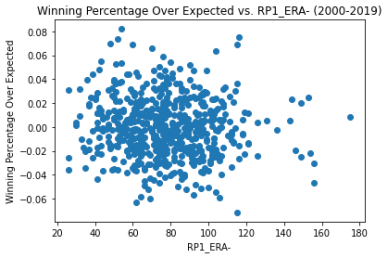
Appendix C: Relief Pitching Metrics vs. Winning Percentage Over Expected

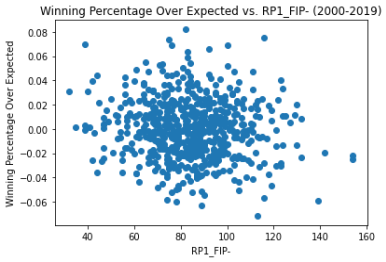
Scatterplots (Modern Dataset)

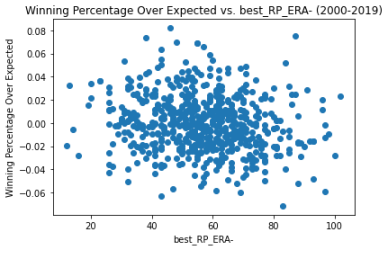


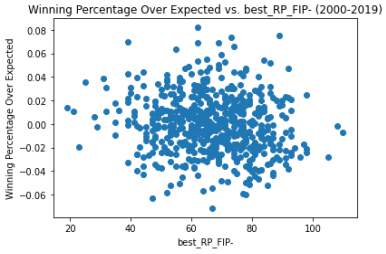


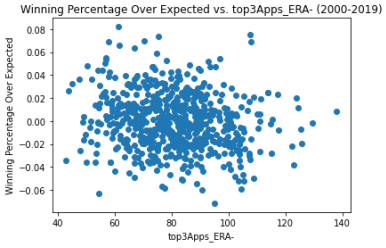


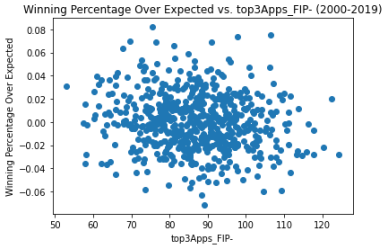


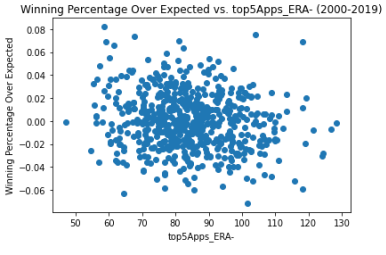


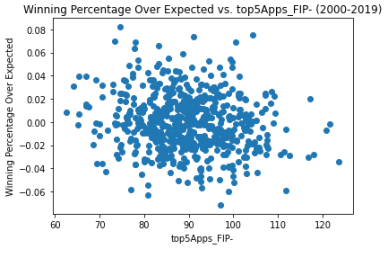




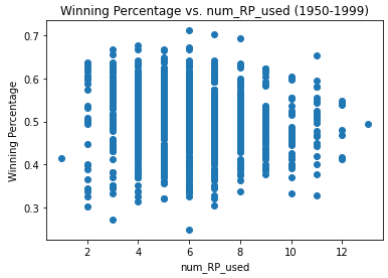


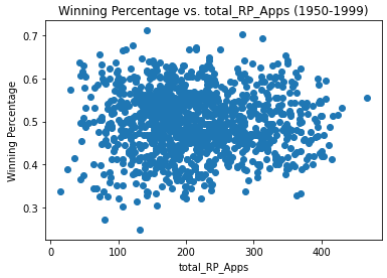


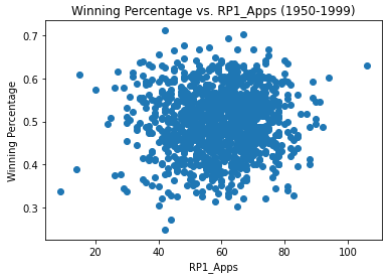


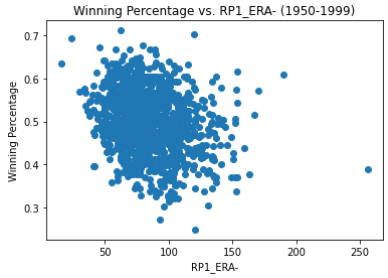


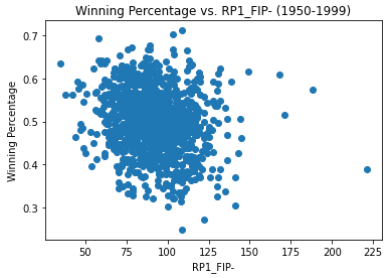
Appendix D: Relief Pitching Metrics vs. Winning Percentage Scatterplots (Early Dataset)

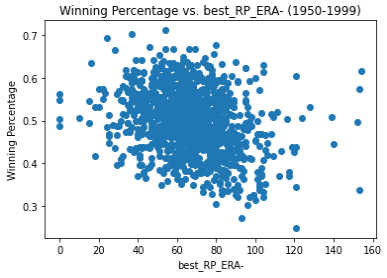


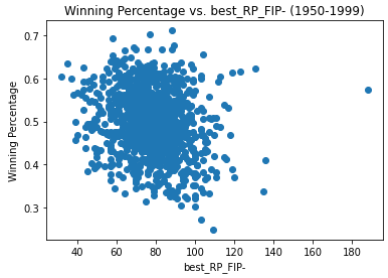


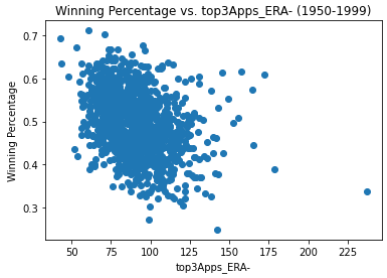


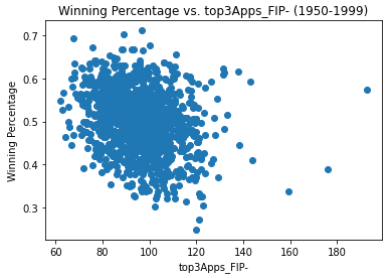


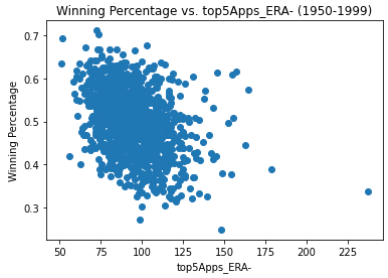


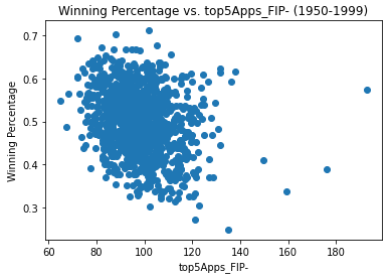




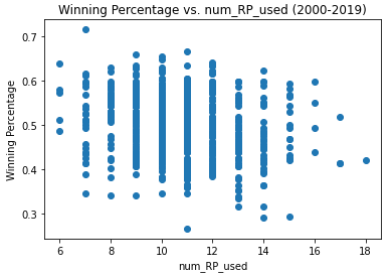


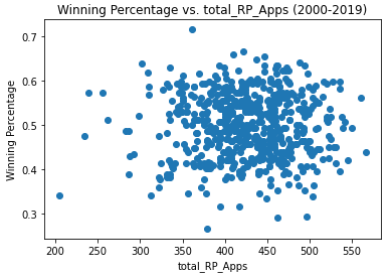


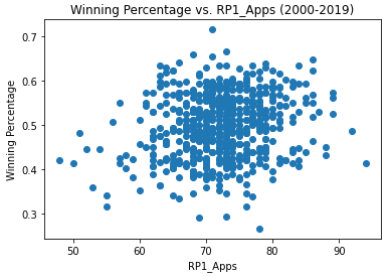


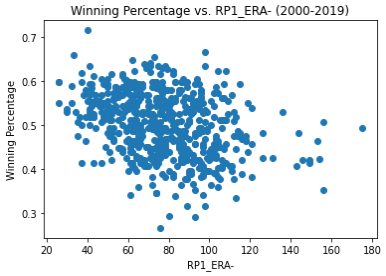


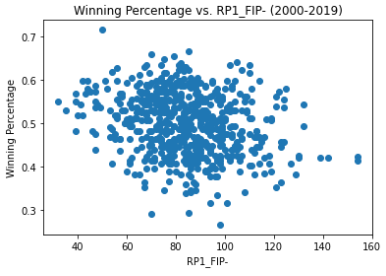
Appendix E: Relief Pitching Metrics vs. Winning Percentage Scatterplots (Modern Dataset)

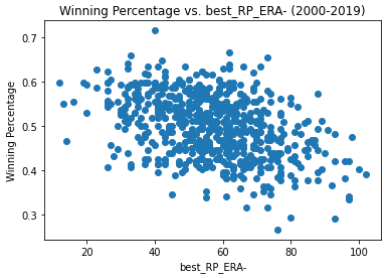


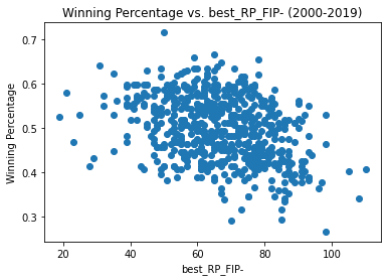


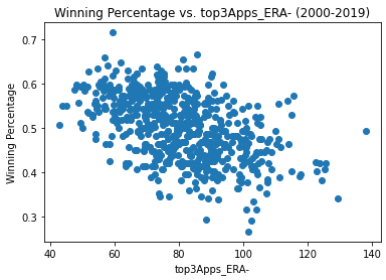


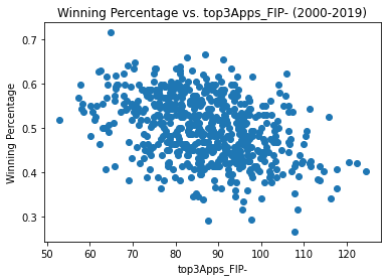


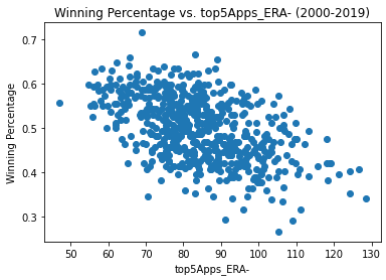


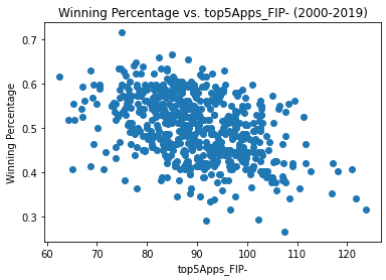












Glossary:

ERA-. An era and park adjusted statistic displaying the number of earned runs a pitcher gives up per nine innings as compared to their peers. A value of 100 is league average, while every point above 100 is 1% worse than league average and every point below 100 is 1% better than league average.

FIP-. A very similar statistic to ERA- in terms of scaling and interpretation, but with an attempt to eliminate the impact of the defense behind the pitcher, hence the name Fielding Independent Pitching (FIP). Instead of using the number of runs allowed by a pitcher like ERA-, FIP- only looks at the events that pitchers are able to control fully (strikeouts, unintentional walks, hit batters, and home runs).

Works Cited:

Brown, Maury. “MLB Sets New Revenue Record, Exceeding $10.8 Billion for 2022.” *Forbes*, Forbes Magazine, 13 Jan. 2023, https://www.forbes.com/sites/maurybrown/2023/01/10/mlb-sets-new-revenue-record-exceeding-108-billion-for-2022/?sh=2125a76a77ee.

Lahman, Sean. “Download Lahman's Baseball Database.” *SeanLahman.com*, 20 Feb. 2023, https://www.seanlahman.com/baseball-archive/statistics/.

“Major League Leaderboards Dashboard: Fangraphs Baseball.” *Major League Leaderboards Dashboard | FanGraphs Baseball*, 24 Mar. 2023, https://www.fangraphs.com/leaders.aspx.

“MLB Positional Payrolls.” *Spotrac.com*, 2023, https://www.spotrac.com/mlb/positional/.

Richards, Dan. “The Relative Value of FIP, XFIP, Siera, and Xera Pt. II.” *Pitcher List*, 9 Apr. 2020, https://www.pitcherlist.com/the-relative-value-of-fip-xfip-siera-and-xera-pt-ii/.

Slowinski, Piper. “FIP.” *Sabermetrics Library*, FanGraphs, 15 Feb. 2010, https://library.fangraphs.com/pitching/fip/.

Slowinski, Piper. “LI.” *Sabermetrics Library*, 17 Feb. 2010, https://library.fangraphs.com/misc/li/.

Van Rossum, Guido et al. *Python 3 Reference Manual*.. CreateSpace, 2009.

Vincent, David. “How Rules Changes in 1920 Affected Home Runs.” *Society for American Baseball Research*, SABR, 19 Jan. 2022, https://sabr.org/journal/article/how-rules-changes-in-1920-affected-home-runs/.

“Wins above Replacement (WAR): Glossary.” *MLB.com*, https://www.mlb.com/glossary/advanced-stats/wins-above-replacement.

Whiteside, David et al. “Changes in a Starting Pitcher's Performance Characteristics Across the Duration of a Major League Baseball Game.” *International journal of sports physiology and performance* vol. 11,2 (2016): 247-54. doi:10.1123/ijspp.2015-0121

Assessing the Impact of Relief Pitching in Major League Baseball Honors Thesis Code

Setup

In [ ]:
```
# General Imports
import pandas as pd
import numpy as np
import datetime as dt
import scipy.stats as ss
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

Get and clean relief pitcher data

Upload Data From FanGraphs

In [ ]:
```
basic_stats = pd.read_csv('data/fg_basic_stats.csv')
adv_stats = pd.read_csv('data/fg_adv_stats.csv')

full_df = pd.merge(basic_stats, adv_stats, on=['playerid', 'Season', 'Name', 'Team'])
full_df.head()
```

Out[ ]:

	playerid	Season	Name	Team	G	GS	IP	WAR	WHIP	ERA-	FIP-	ERA
0	200	1999	Pedro Martinez	BOS	31	29	213.1	11.6	0.92	42	31	2.07
1	1001964	1972	Steve Carlton	PHI	41	41	346.1	11.1	0.99	56	60	1.97
2	1001098	1973	Bert Blyleven	MIN	40	40	325.0	10.8	1.12	63	59	2.52
3	815	1997	Roger Clemens	TOR	34	34	264.0	10.7	1.03	45	50	2.05
4	60	2001	Randy Johnson	ARI	35	34	249.2	10.4	1.01	55	47	2.49

Isolate only relief pitchers

In [ ]:
```
full_df['StarterP'] = full_df['GS'] / full_df['G']
relief_df = full_df[full_df['StarterP'] <= 0.15]
print(f"Total number of relief pitcher seasons: {len(relief_df)}")
```

Total number of relief pitcher seasons: 14447

Remove all data columns that aren't available for the entire data set

In [ ]:
```
common_cols = []

for col in relief_df.columns:
    if len(relief_df[relief_df[col].isnull()]) == 0: common_cols.append(col)
```

```
relief_df = relief_df[common_cols]
relief_df.head()
```

Out[ ]:

	playerid	Season	Name	Team	G	GS	IP	WAR	WHIP	ERA-	FIP-	ERA	StarterP
634	1012743	1977	Bruce Sutter	CHC	62	0	107.1	5.2	0.86	31	39	1.34	0.0
748	1003712	1986	Mark Eichhorn	TOR	69	0	157.0	4.9	0.96	41	56	1.72	0.0
861	1012743	1979	Bruce Sutter	CHC	62	0	101.1	4.8	0.98	54	47	2.22	0.0
935	1006868	1979	Jim Kern	TEX	71	0	143.0	4.7	1.13	38	65	1.57	0.0
943	650	2003	Eric Gagne	LAD	77	0	82.1	4.7	0.69	30	19	1.20	0.0

Create final, clean relief pitcher dataframe

In [ ]:
```
intro_cols = ['Season', 'Name', 'Team', 'G', 'WAR', 'ERA', 'WHIP', 'IP']
stat_cols = ['ERA-', 'FIP-']
clean_relief_df = relief_df[intro_cols+stat_cols]
clean_relief_df = clean_relief_df.sort_values('Season').rename(columns={'G': 'Apps'})\
    .reset_index(drop=True)
clean_relief_df.head()
```

Out[ ]:

	Season	Name	Team	Apps	WAR	ERA	WHIP	IP	ERA-	FIP-
0	1920	Benn Karr	BOS	26	0.2	4.81	1.45	91.2	133	102
1	1920	Bob McGraw	NYY	15	-0.3	4.67	1.63	27.0	124	130
2	1920	Mule Watson	PIT	5	-0.4	8.74	1.94	11.1	275	207
3	1920	George Payne	CHW	12	-0.1	5.46	1.62	29.2	145	112
4	1920	Fritz Coumbe	CIN	3	0.1	4.91	1.43	14.2	163	82

Get team performance data

Pull team data from the Lahman database and clean team IDs

In [ ]:
```
team_df = pd.read_csv('data/baseballdatabank-2022.2/core/Teams.csv')
team_df = team_df[team_df['yearID'].between(1920, 2019)].reset_index(drop=True)
team_df['RD'] = team_df['R'] - team_df['RA']
team_df = team_df[['yearID', 'lgID', 'teamIDBR', 'franchID', 'G', 'W', 'L',
                   'R', 'RA', 'RD']]
for i in range(len(team_df)):
    if team_df.loc[i, 'teamIDBR'] in ['WSH', 'WSA']:
        team_df.loc[i, 'teamIDBR'] = 'WAS'
    elif team_df.loc[i, 'teamIDBR'] == 'MLN':
        team_df.loc[i, 'teamIDBR'] = 'MIL'
```

```
    elif team_df.loc[i, 'teamIDBR'] == 'SEP':
        team_df.loc[i, 'teamIDBR'] = 'SEA'
```

Add winning percentage and expected winning percentage columns

In [ ]:
```
# Add winning percentage column to normalize for different season lengths
team_df['WinP'] = team_df['W'] / team_df['G']

# Adding expected winning percentage column
## Create a dict with regression equations for expected winning percentage
## based on run differential by year
xwinp_lm_dict = {}
for year in list(range(1920, 2020)):
    temp_df = team_df[team_df['yearID'] == year]
    x = temp_df['RD']
    x = sm.add_constant(x)
    y = temp_df['WinP']
    lm = sm.OLS(y, x).fit()
    xwinp_lm_dict[year] = [lm.params[0], lm.params[1], lm.rsquared]

## Plug team values in to yearly regression equations to get expected winning percentage
for i in range(len(team_df)):
    lm_vals = xwinp_lm_dict[team_df.loc[i, 'yearID']]
    team_df.loc[i, 'XWinP'] = lm_vals[0] + (lm_vals[1] * team_df.loc[i, 'RD'])

# Add a column for the difference between actual and expected winning percentage
# to measure overperformance
team_df['ExcessWinP'] = team_df['WinP'] - team_df['XWinP']

team_df.head()
```

Out[ ]:

	yearID	lgID	teamIDBR	franchID	G	W	L	R	RA	RD	WinP	XWinP	ExcessWinP
0	1920	AL	BOS	BOS	154	72	81	650	698	-48	0.467532	0.466913	0.000620
1	1920	NL	BRO	LAD	155	93	61	660	528	132	0.600000	0.582008	0.017992
2	1920	NL	BSN	ATL	153	62	90	523	670	-147	0.405229	0.403610	0.001619
3	1920	AL	CHW	CHW	154	96	58	794	665	129	0.623377	0.580089	0.043287
4	1920	NL	CHC	CHC	154	75	79	619	635	-16	0.487013	0.487374	-0.000361

Stitch together final dataframe

Add teams' top 5 relief pitchers by Apps to their team performance records

In [ ]:
```
for i, row in team_df.iterrows():
    temp_RPs = clean_relief_df[((clean_relief_df['Season'] == row['yearID']) & \
                                (clean_relief_df['Team'] == row['teamIDBR']))]
    temp_RPs = temp_RPs.sort_values('Apps', ascending=False).reset_index(drop=True)
    team_df.loc[i, 'num_RP_used'] = len(temp_RPs)
    team_df.loc[i, 'total_RP_Apps'] = temp_RPs['Apps'].sum()
```

```
        if len(temp_RPs) == 0: continue
        if len(temp_RPs) > 5: temp_RPs = temp_RPs.head(5)
        for j in range(len(temp_RPs)):
            team_df.loc[i, f'RP{j+1}_Apps'] = temp_RPs.loc[j, 'Apps']
            team_df.loc[i, f'RP{j+1}_ERA-'] = temp_RPs.loc[j, 'ERA-']
            team_df.loc[i, f'RP{j+1}_FIP-'] = temp_RPs.loc[j, 'FIP-']
    print(f'Last year where a team had 0 relief pitchers throw 10 innings or more: \
    {team_df[team_df["RP1_Apps"].isna()]["yearID"].max()}')
    team_df
```

Last year where a team had 0 relief pitchers throw 10 innings or more: 1944

Out[ ]:

	yearID	lgID	teamIDBR	franchID	G	W	L	R	RA	RD	...	RP2_FIP-	RP3_Apps	RP3
0	1920	AL	BOS	BOS	154	72	81	650	698	-48	...	78.0	NaN	
1	1920	NL	BRO	LAD	155	93	61	660	528	132	...	110.0	NaN	
2	1920	NL	BSN	ATL	153	62	90	523	670	-147	...	NaN	NaN	
3	1920	AL	CHW	CHW	154	96	58	794	665	129	...	NaN	NaN	
4	1920	NL	CHC	CHC	154	75	79	619	635	-16	...	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2217	2019	NL	STL	STL	162	91	71	764	662	102	...	71.0	66.0	
2218	2019	AL	TBR	TBD	162	96	66	769	656	113	...	76.0	65.0	
2219	2019	AL	TEX	TEX	162	78	84	810	878	-68	...	77.0	50.0	
2220	2019	AL	TOR	TOR	162	67	95	726	828	-102	...	105.0	53.0	
2221	2019	NL	WSN	WSN	162	93	69	873	724	149	...	95.0	52.0	

2222 rows × 30 columns

Add summary data columns

In [ ]:
```
team_df['best_RP_ERA-'] = np.nanmin(team_df[[x for x in team_df.columns \
                                             if 'ERA-' in x]], axis=1)
team_df['best_RP_FIP-'] = np.nanmin(team_df[[x for x in team_df.columns \
                                             if 'FIP-' in x]], axis=1)

def add_avgs(x, num, stat):
    total_ip = 0
    weighted_total = 0
    for rp in [f'RP{x+1}' for x in range(num)]:
        if not x[f'{rp}_Apps'] == x[f'{rp}_Apps']: continue
        total_ip += x[f'{rp}_Apps']
        weighted_total += x[f'{rp}_Apps']*x[f'{rp}_{stat}']
    if total_ip == 0: weighted_avg = np.nan
    else: weighted_avg = weighted_total / total_ip
    return(weighted_avg)
```

```
team_df['top3Apps_ERA-'] = team_df.apply(lambda row: add_avgs(row, 3, 'ERA-'), axis=1)
team_df['top3Apps_FIP-'] = team_df.apply(lambda row: add_avgs(row, 3, 'FIP-'), axis=1)
team_df['top5Apps_ERA-'] = team_df.apply(lambda row: add_avgs(row, 5, 'ERA-'), axis=1)
team_df['top5Apps_FIP-'] = team_df.apply(lambda row: add_avgs(row, 5, 'FIP-'), axis=1)
final_df = team_df.copy()
```

```
/var/folders/78/2bdp3ycx1bgccctn999st8d80000gn/T/ipykernel_42687/2468811858.py:1: RuntimeWarning: All-NaN axis encountered
  team_df['best_RP_ERA-'] = np.nanmin(team_df[[x for x in team_df.columns \
/var/folders/78/2bdp3ycx1bgccctn999st8d80000gn/T/ipykernel_42687/2468811858.py:3: RuntimeWarning: All-NaN axis encountered
  team_df['best_RP_FIP-'] = np.nanmin(team_df[[x for x in team_df.columns \
```

Save and View Final Dataframe

In [ ]:
```
full_df.to_csv('data/full_df.csv', index=False)
final_df.to_csv('data/final_df.csv', index=False)
final_df.head()
```

Out[ ]:

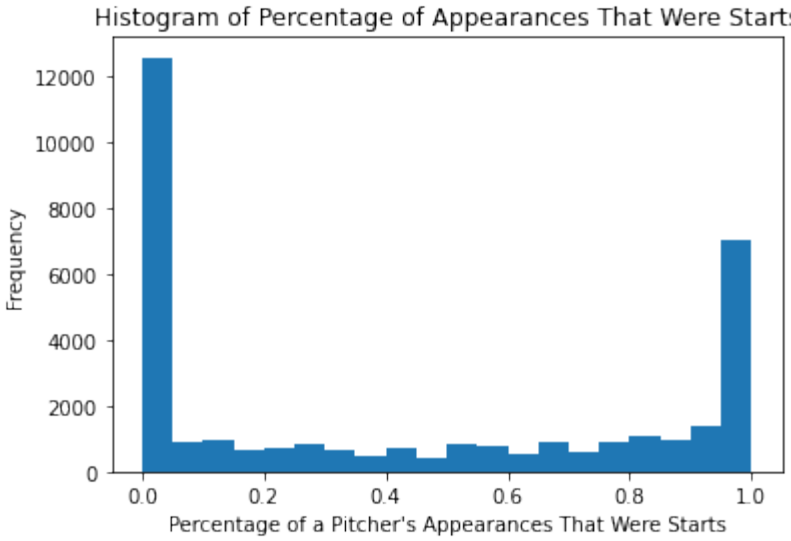
	yearID	lgID	teamIDBR	franchID	G	W	L	R	RA	RD	...	RP4_FIP-	RP5_Apps	RP5_ER
0	1920	AL	BOS	BOS	154	72	81	650	698	-48	...	NaN	NaN	N
1	1920	NL	BRO	LAD	155	93	61	660	528	132	...	NaN	NaN	N
2	1920	NL	BSN	ATL	153	62	90	523	670	-147	...	NaN	NaN	N
3	1920	AL	CHW	CHW	154	96	58	794	665	129	...	NaN	NaN	N
4	1920	NL	CHC	CHC	154	75	79	619	635	-16	...	NaN	NaN	N

5 rows × 36 columns

Visualizing the Data

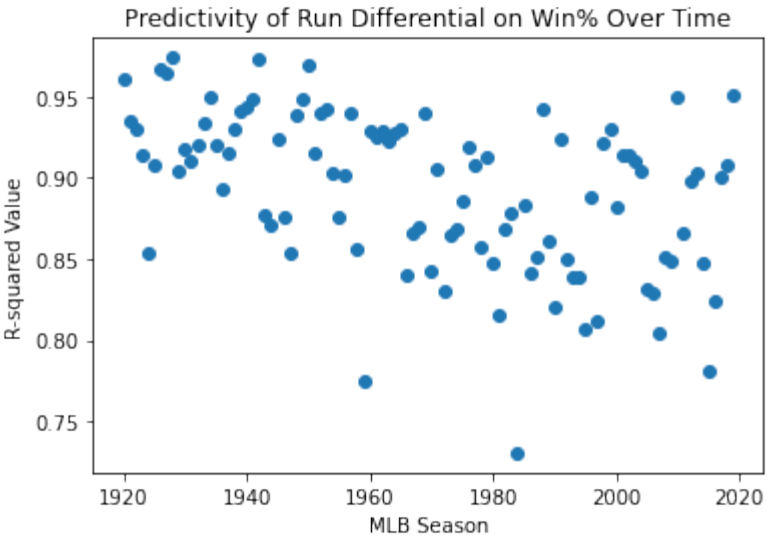
1. Histogram of percentage of appearances that were starts

In [ ]:
```
hist_df = full_df[full_df['Season'] >= 1920]

plt.hist(hist_df['StarterP'], bins=20)
plt.title("Histogram of Percentage of Appearances That Were Starts")
plt.xlabel("Percentage of a Pitcher's Appearances That Were Starts")
plt.ylabel("Frequency")
plt.show()
```



2. Run differential as a predictor of winning percentage R-squared over time

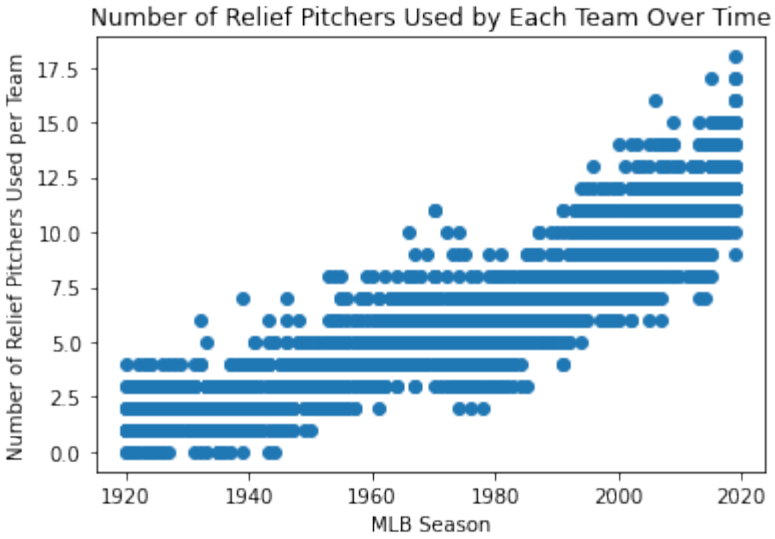
In [ ]:
```
years = []
rsq_vals = []
for year in xwinp_lm_dict.keys():
    years.append(year)
    rsq_vals.append(xwinp_lm_dict[year][2])

plt.scatter(years, rsq_vals)
# z = np.polyfit(years, rsq_vals, 1)
# p = np.poly1d(z)
# plt.plot(years, p(years), 'r--')
plt.title("Predictivity of Run Differential on Win% Over Time")
plt.xlabel("MLB Season")
plt.ylabel("R-squared Value")
plt.show()
```



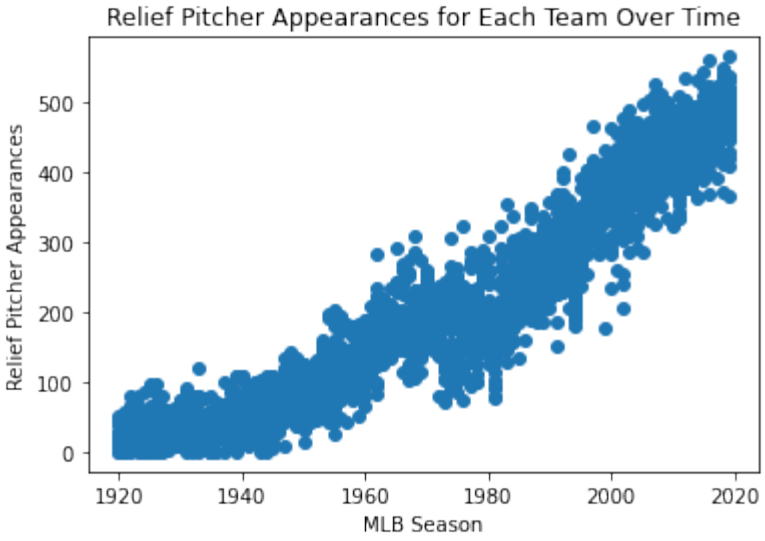
3. Identifying rise of relief pitcher usage

In [ ]:
```
plt.scatter(final_df['yearID'], final_df['num_RP_used'])
plt.title("Number of Relief Pitchers Used by Each Team Over Time")
plt.xlabel("MLB Season")
```

```
    plt.ylabel("Number of Relief Pitchers Used per Team")
    plt.show()
```



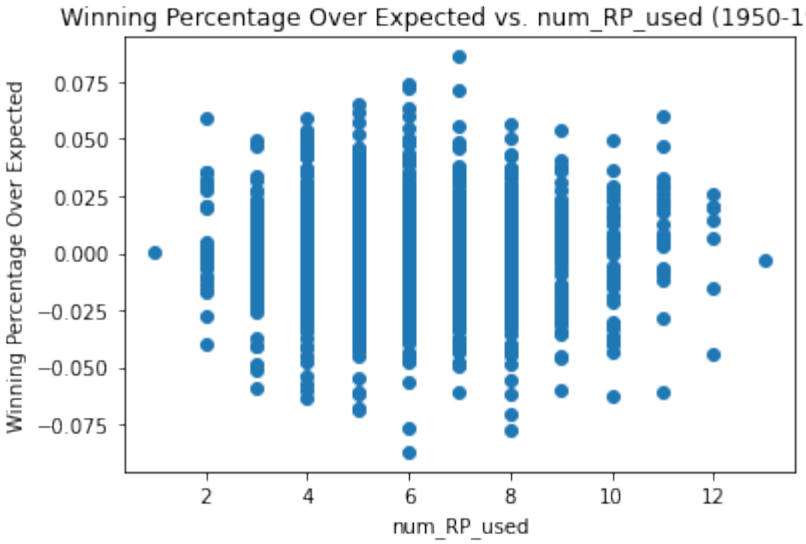
In [ ]:
```
plt.scatter(final_df['yearID'], final_df['total_RP_Apps'])
plt.title("Relief Pitcher Appearances for Each Team Over Time")
plt.xlabel("MLB Season")
plt.ylabel("Relief Pitcher Appearances")
plt.show()
```

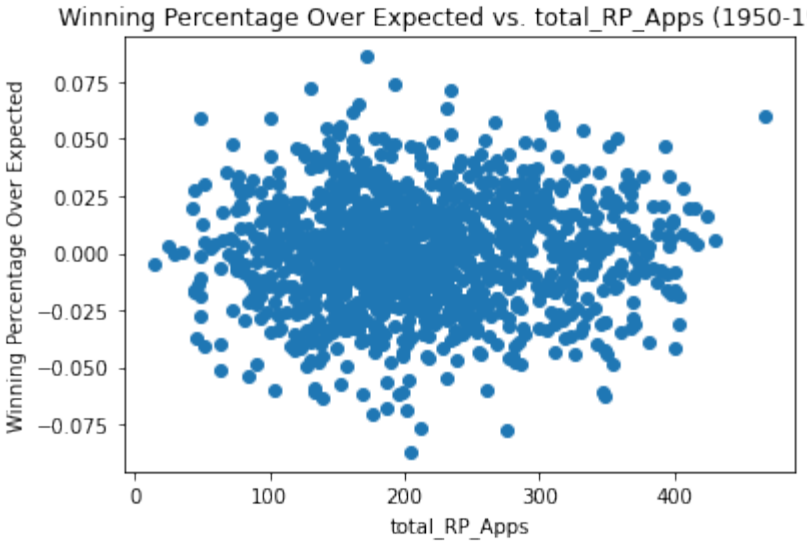


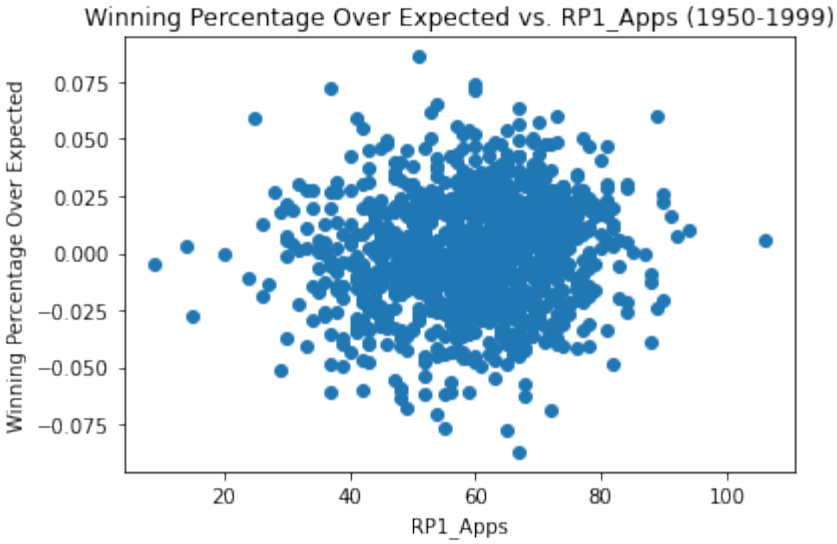
4. Plots of excess winning percentage vs predictors

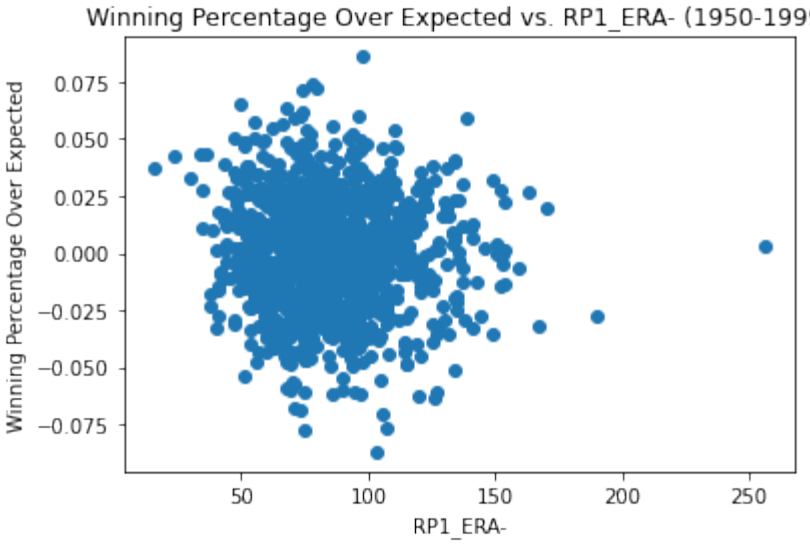
In [ ]:
```
predictors = ['num_RP_used', 'total_RP_Apps', 'RP1_Apps', 'RP1_ERA-', 'RP1_FIP-'] \
    + list(final_df.columns[30:])
response = ['ExcessWinP']
```

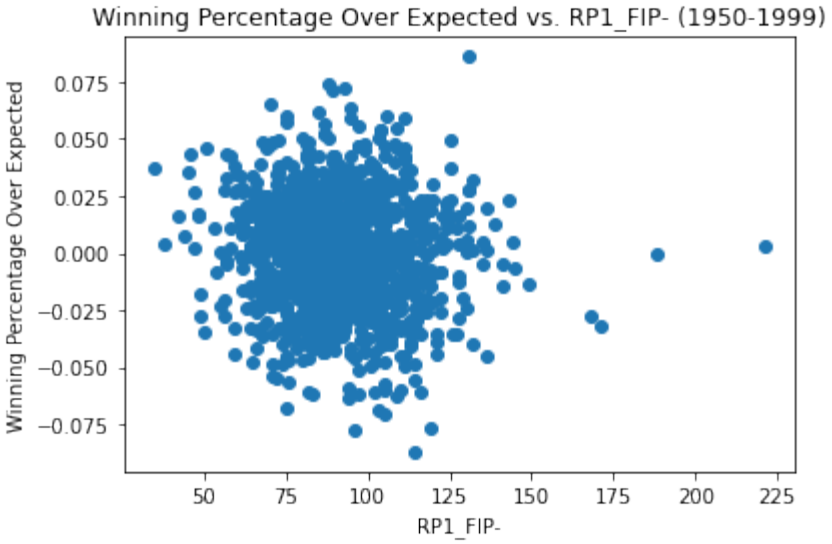
In [ ]:
```
early_df = final_df[(final_df['yearID'] >= 1950)&(final_df['yearID'] < 2000)]

for pred in predictors:
    plt.scatter(early_df[pred], early_df['ExcessWinP'])
    plt.title(f"Winning Percentage Over Expected vs. {pred} (1950-1999)")
    plt.xlabel(pred)
    plt.ylabel("Winning Percentage Over Expected")
    plt.show()
```

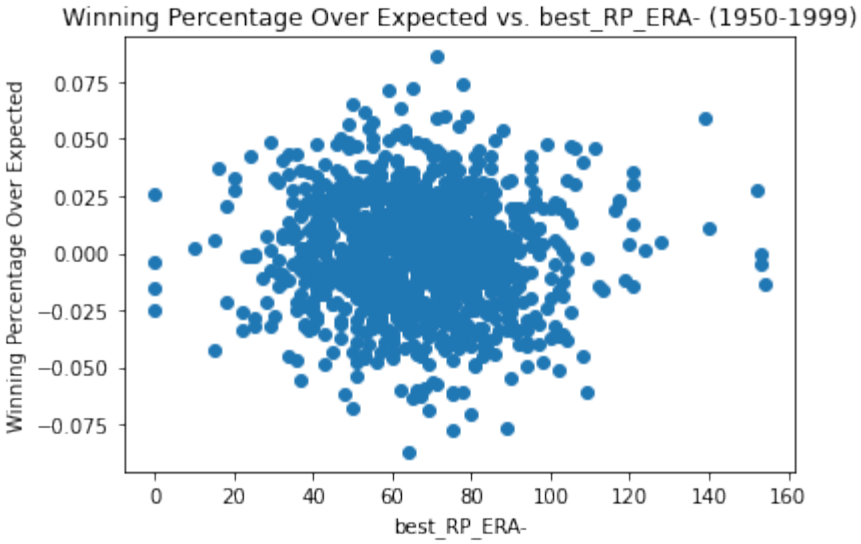

Winning Percentage Over Expected vs. num_RP_used (1950-1999)
Winning Percentage Over Expected
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
2
4
6
8
10
12
num_RP_used

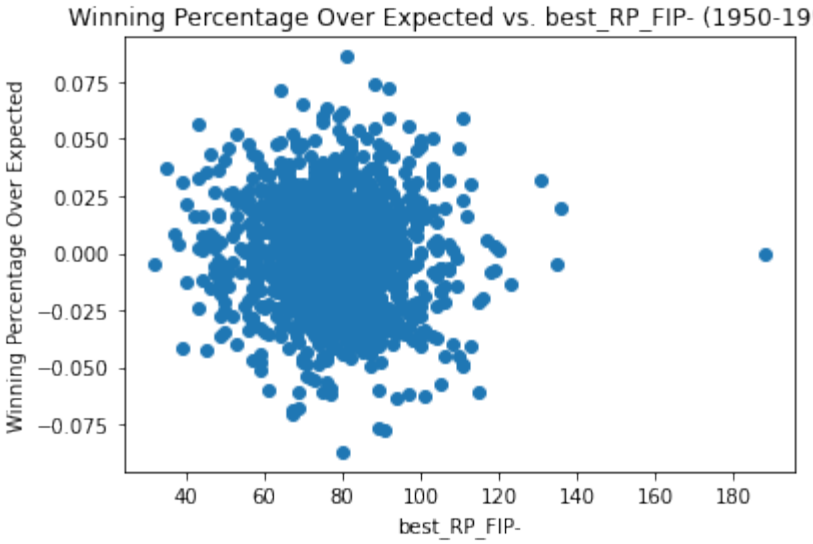

Winning Percentage Over Expected vs. total_RP_Apps (1950-1999)
Winning Percentage Over Expected
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
0
100
200
300
400
total_RP_Apps

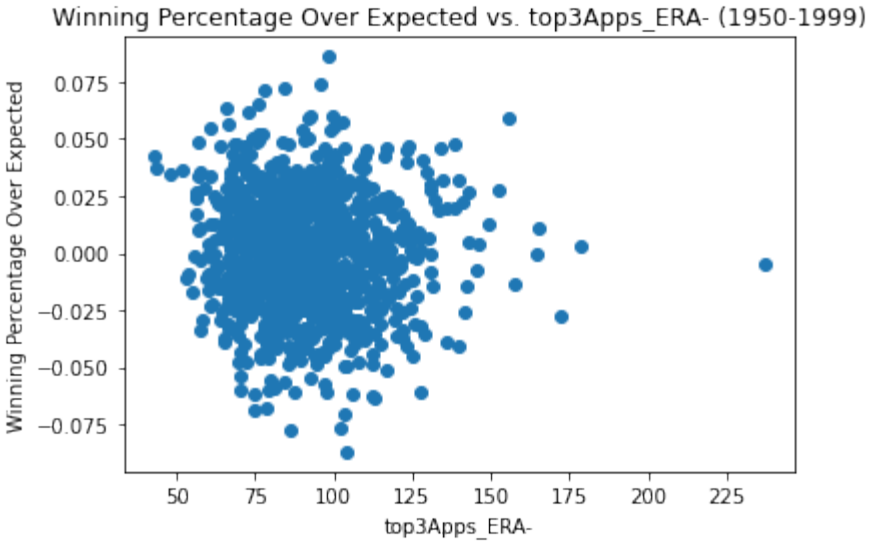

Winning Percentage Over Expected vs. RP1_Apps (1950-1999)
Winning Percentage Over Expected
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
20
40
60
80
100
RP1_Apps

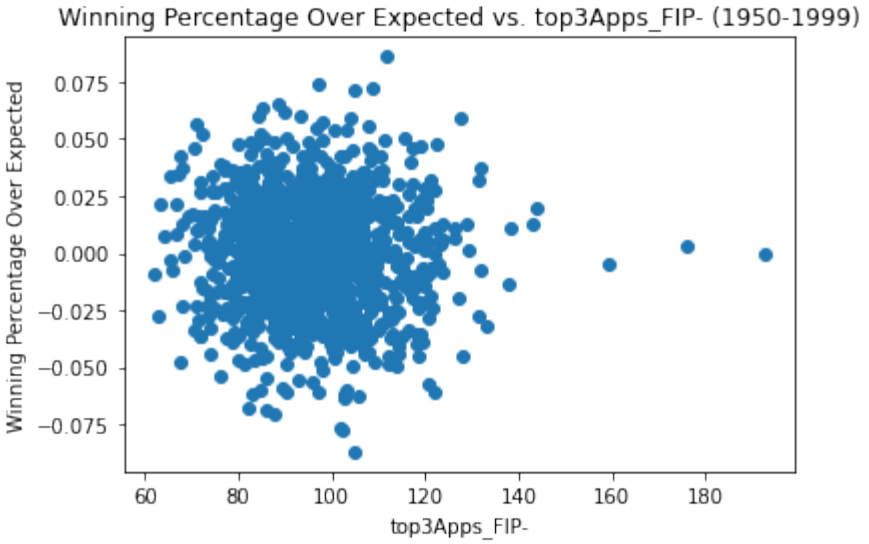

Winning Percentage Over Expected vs. RP1_ERA- (1950-1999)
Winning Percentage Over Expected
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
50
100
150
200
250
RP1_ERA-

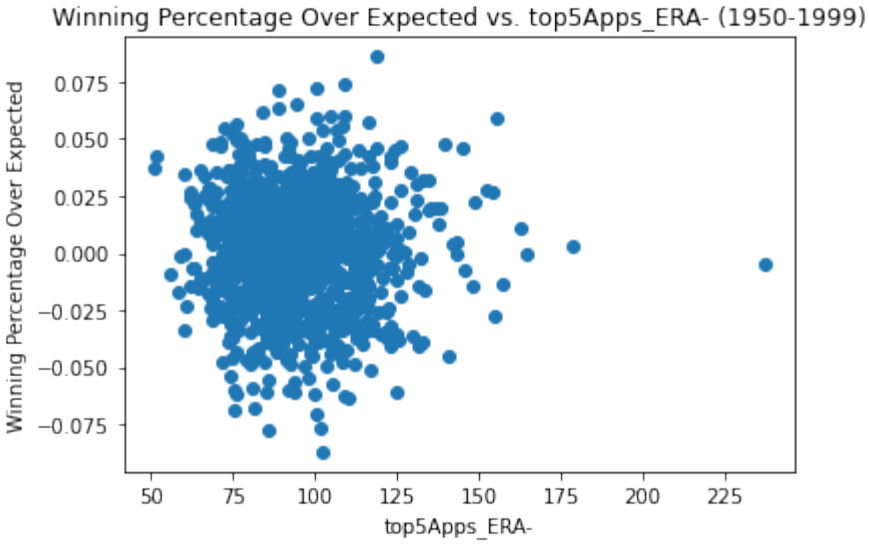

Winning Percentage Over Expected vs. RP1_FIP- (1950-1999)
Winning Percentage Over Expected
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
50
75
100
125
150
175
200
225
RP1_FIP-

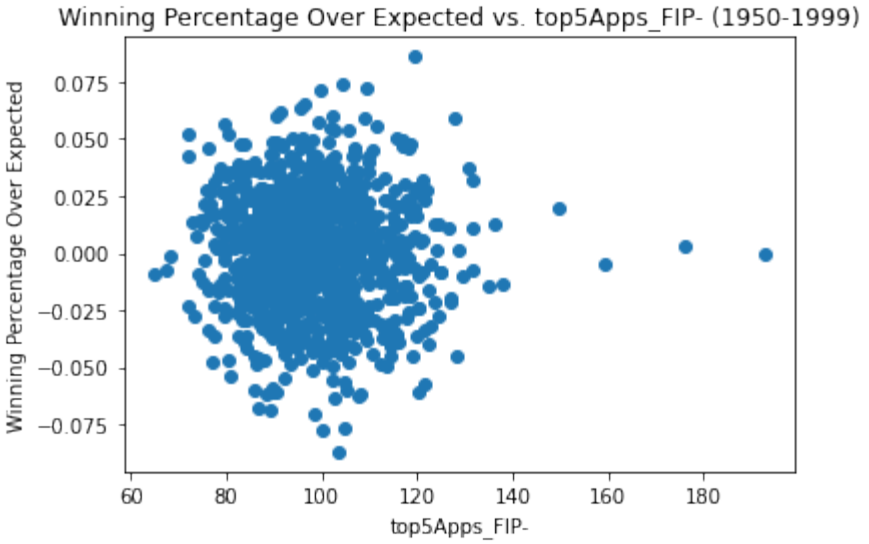

Winning Percentage Over Expected vs. best_RP_ERA- (1950-1999)
Winning Percentage Over Expected
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
0
20
40
60
80
100
120
140
160
best_RP_ERA-


Winning Percentage Over Expected vs. best_RP_FIP- (1950-1999)
Winning Percentage Over Expected
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
40
60
80
100
120
140
160
180
best_RP_FIP-

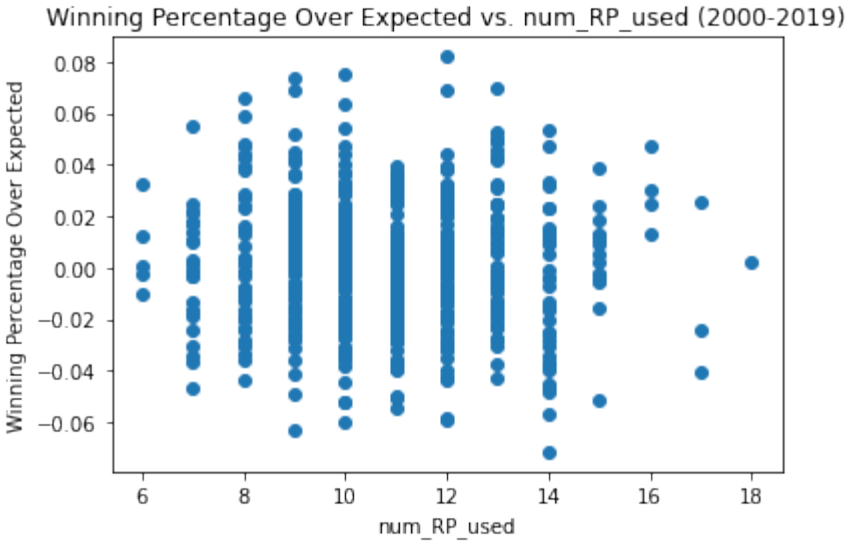

Winning Percentage Over Expected vs. top3Apps_ERA- (1950-1999)
Winning Percentage Over Expected
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
50
75
100
125
150
175
200
225
top3Apps_ERA-

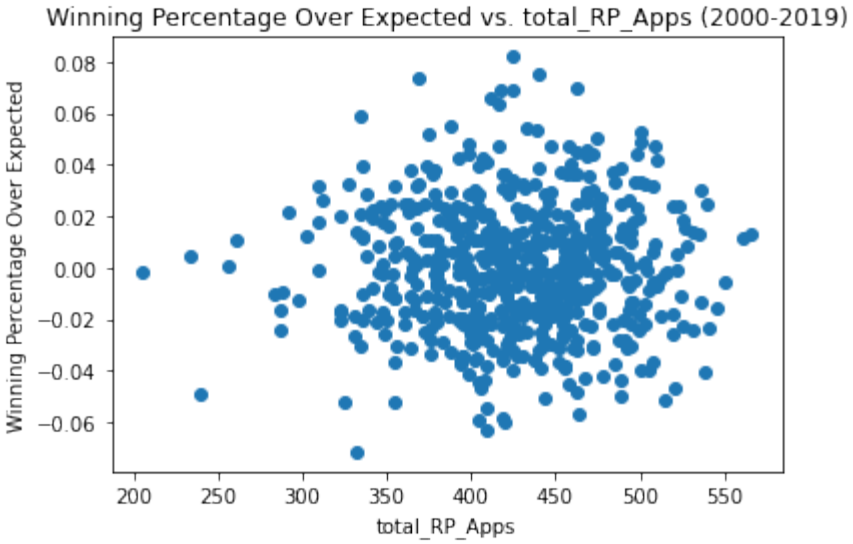

Winning Percentage Over Expected vs. top3Apps_FIP- (1950-1999)
Winning Percentage Over Expected
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
60
80
100
120
140
160
180
top3Apps_FIP-

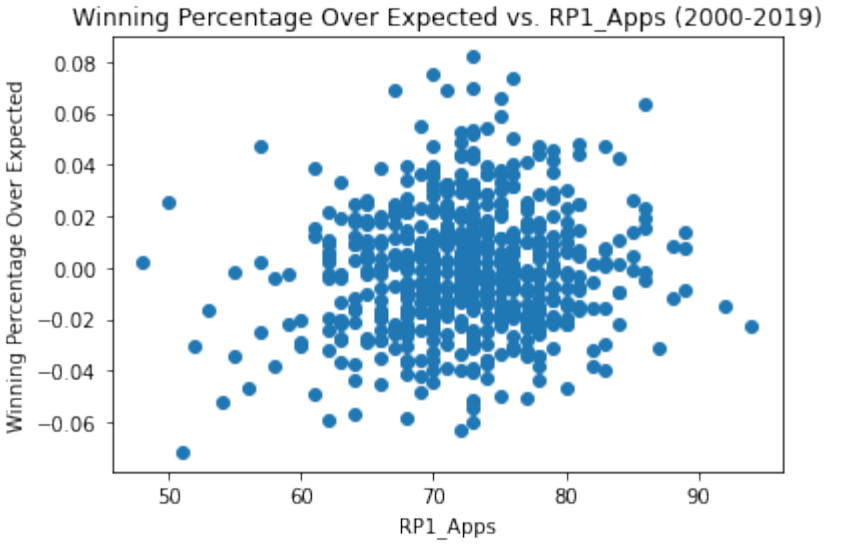


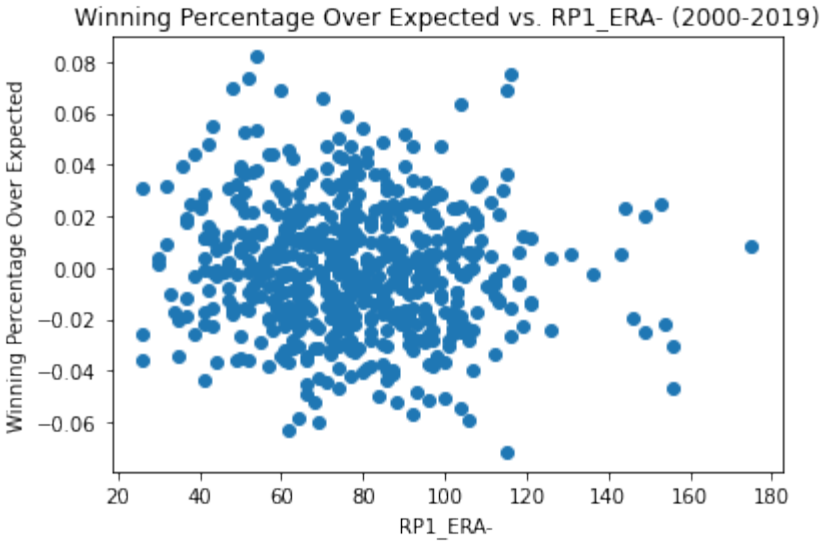


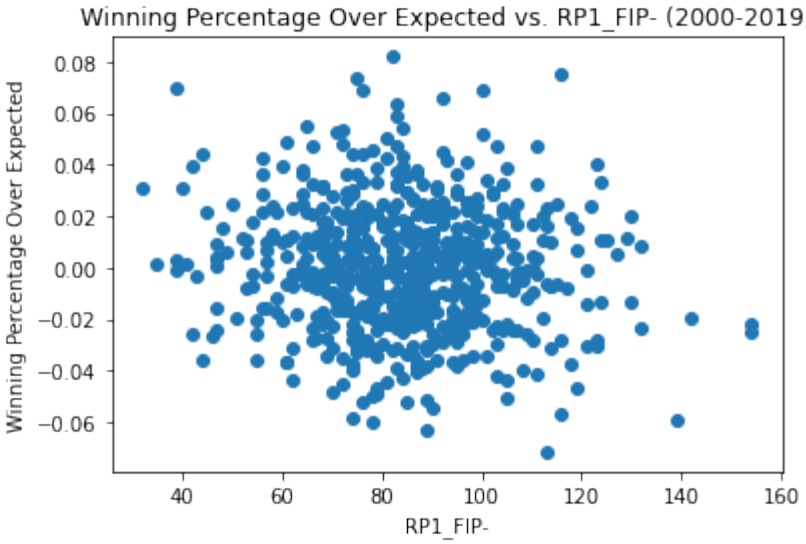
In [ ]:
```
modern_df = final_df[final_df['yearID'] >= 2000]

for pred in predictors:
    plt.scatter(modern_df[pred], modern_df['ExcessWinP'])
    plt.title(f"Winning Percentage Over Expected vs. {pred} (2000-2019)")
    plt.xlabel(pred)
    plt.ylabel("Winning Percentage Over Expected")
    plt.show()
```

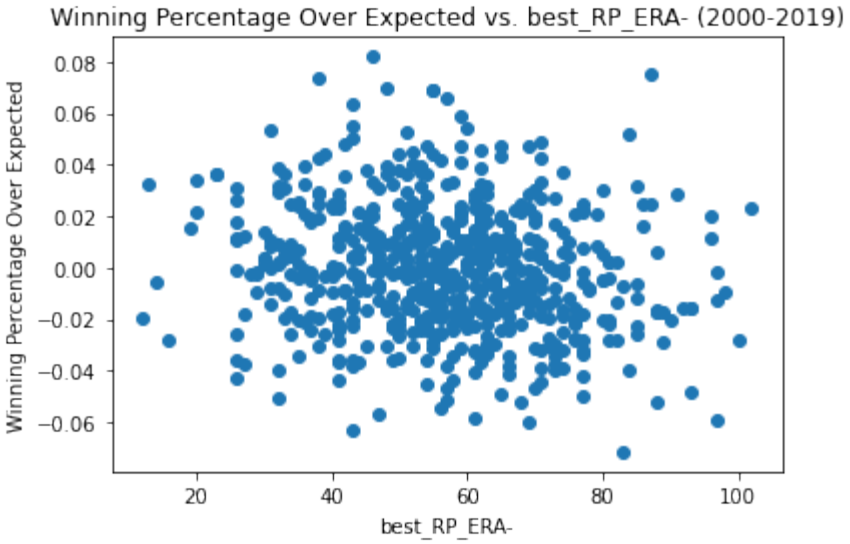

Winning Percentage Over Expected vs. num_RP_used (2000-2019)
Winning Percentage Over Expected
num_RP_used
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
6
8
10
12
14
16
18

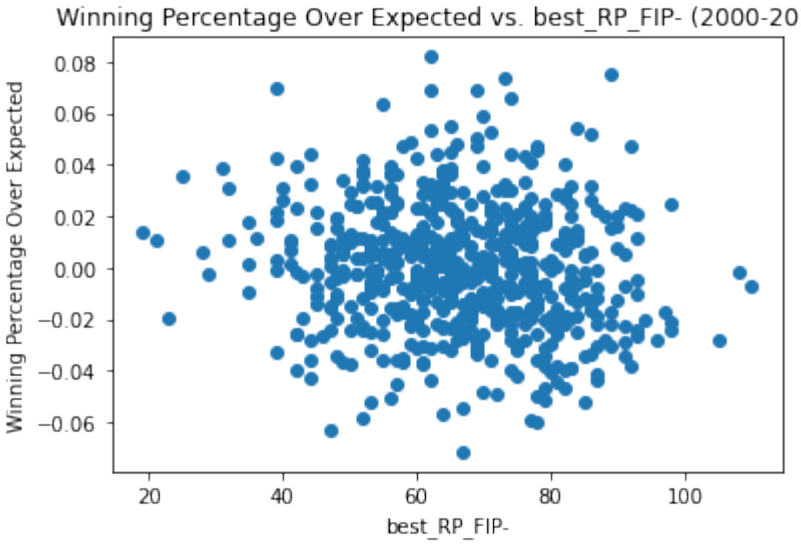

Winning Percentage Over Expected vs. total_RP_Apps (2000-2019)
Winning Percentage Over Expected
total_RP_Apps
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
200
250
300
350
400
450
500
550

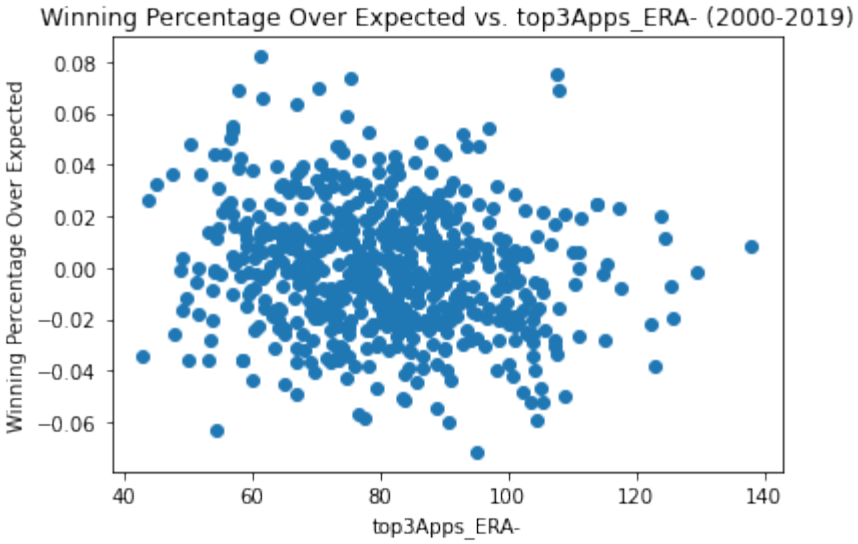

Winning Percentage Over Expected vs. RP1_Apps (2000-2019)
Winning Percentage Over Expected
RP1_Apps
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
50
60
70
80
90

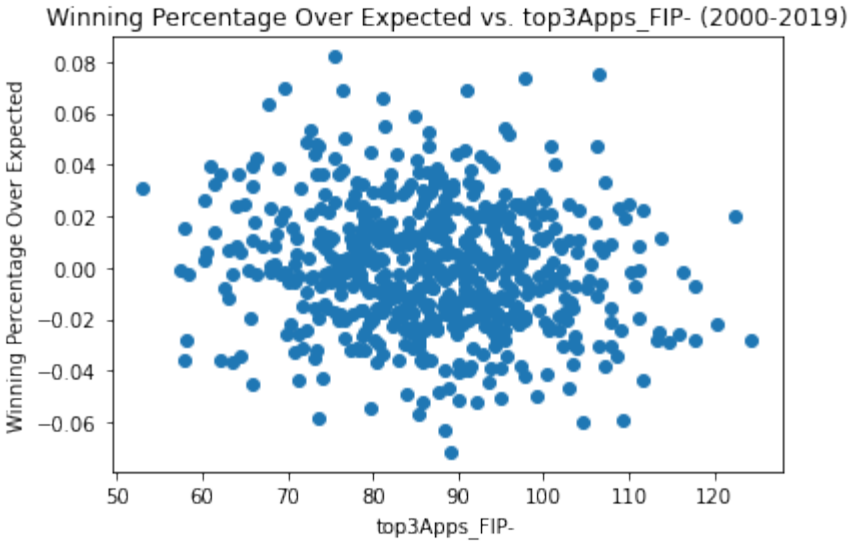

Winning Percentage Over Expected vs. RP1_ERA- (2000-2019)
Winning Percentage Over Expected
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
20
40
60
80
100
120
140
160
180
RP1_ERA-

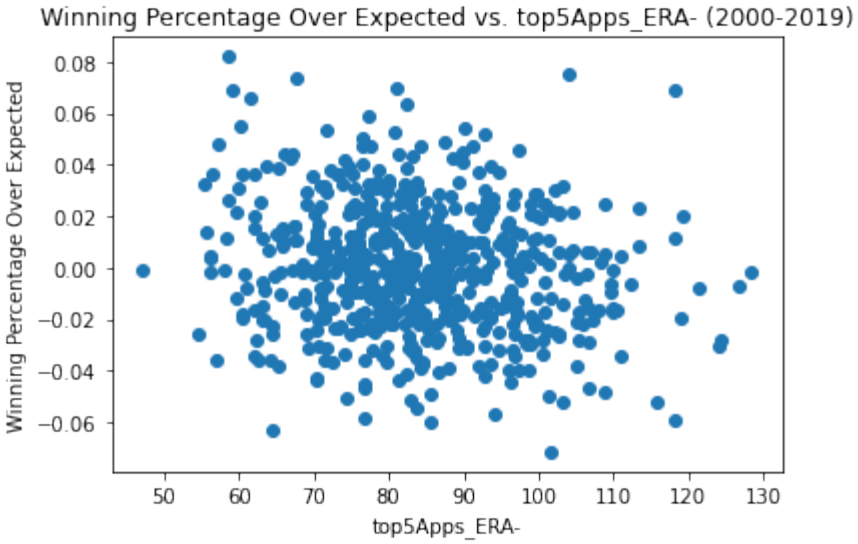

Winning Percentage Over Expected vs. RP1_FIP- (2000-2019)
Winning Percentage Over Expected
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
40
60
80
100
120
140
160
RP1_FIP-

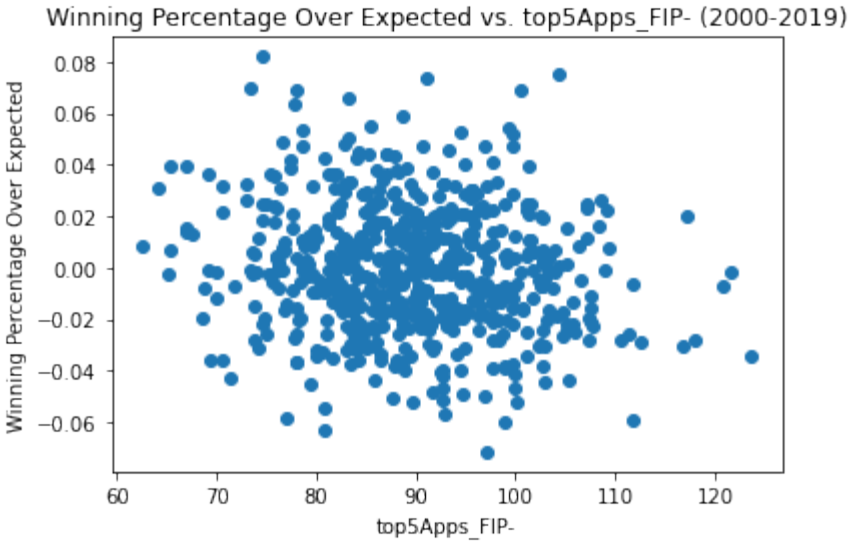

Winning Percentage Over Expected vs. best_RP_ERA- (2000-2019)
Winning Percentage Over Expected
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
20
40
60
80
100
best_RP_ERA-


Winning Percentage Over Expected vs. best_RP_FIP- (2000-2019)
Winning Percentage Over Expected
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
20
40
60
80
100
best_RP_FIP-


Winning Percentage Over Expected vs. top3Apps_ERA- (2000-2019)
Winning Percentage Over Expected
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
40
60
80
100
120
140
top3Apps_ERA-

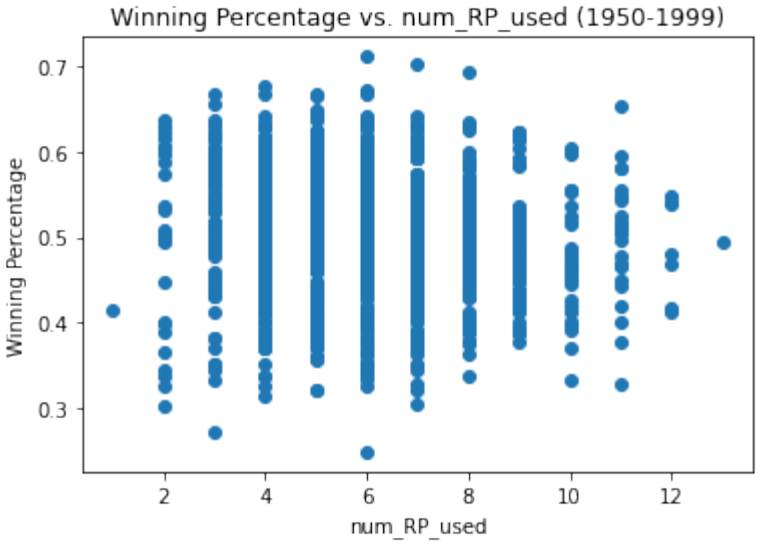

Winning Percentage Over Expected vs. top3Apps_FIP- (2000-2019)
Winning Percentage Over Expected
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
50
60
70
80
90
100
110
120
top3Apps_FIP-

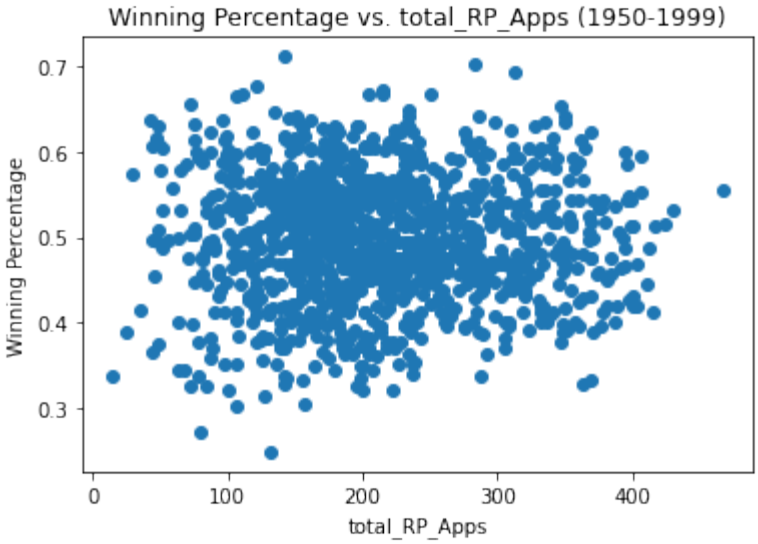


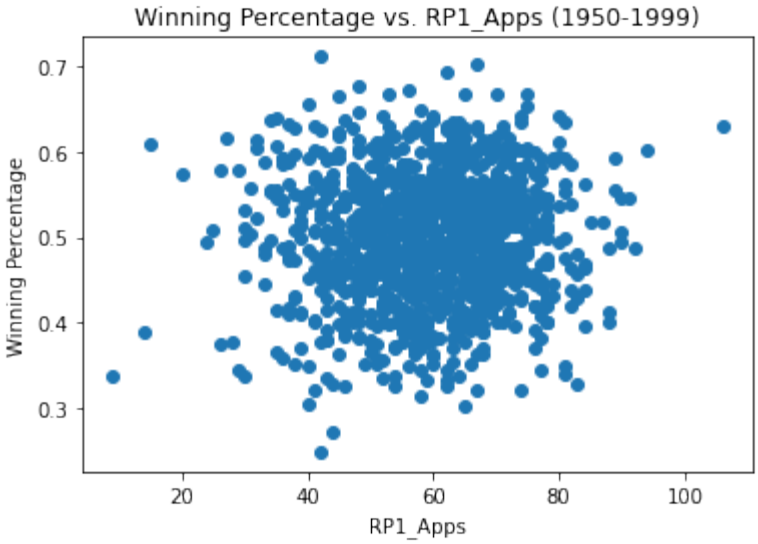


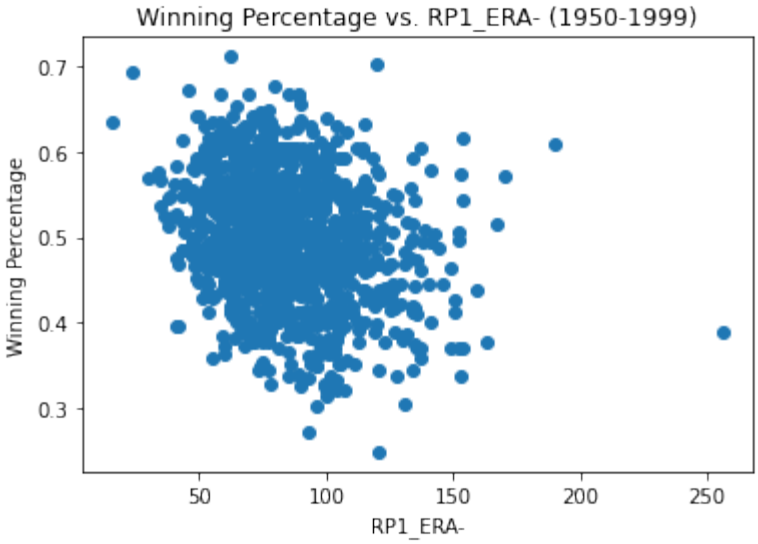
5. Plots of winning percentage vs metrics

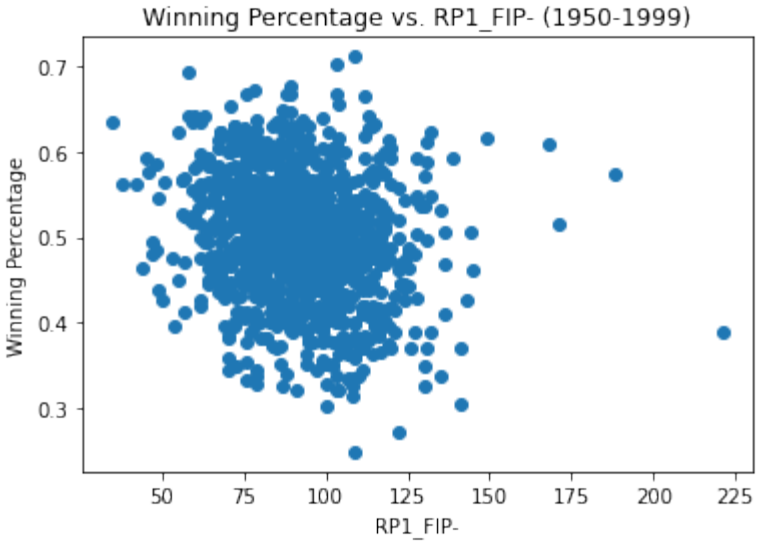
In [ ]:
```
early_df = final_df[(final_df['yearID'] >= 1950)&(final_df['yearID'] < 2000)]

for pred in predictors:
    plt.scatter(early_df[pred], early_df['WinP'])
    plt.title(f"Winning Percentage vs. {pred} (1950-1999)")
    plt.xlabel(pred)
    plt.ylabel("Winning Percentage")
    plt.show()
```

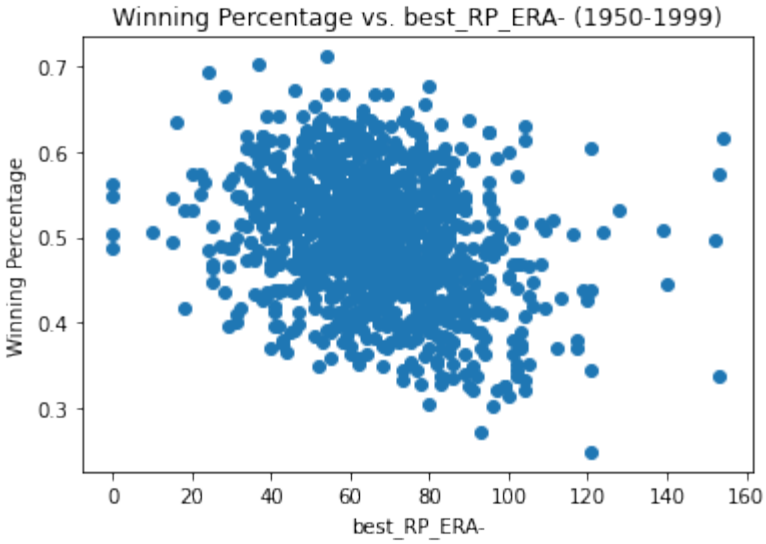

Winning Percentage vs. num_RP_used (1950-1999)
Winning Percentage
num_RP_used
0.7
0.6
0.5
0.4
0.3
2
4
6
8
10
12

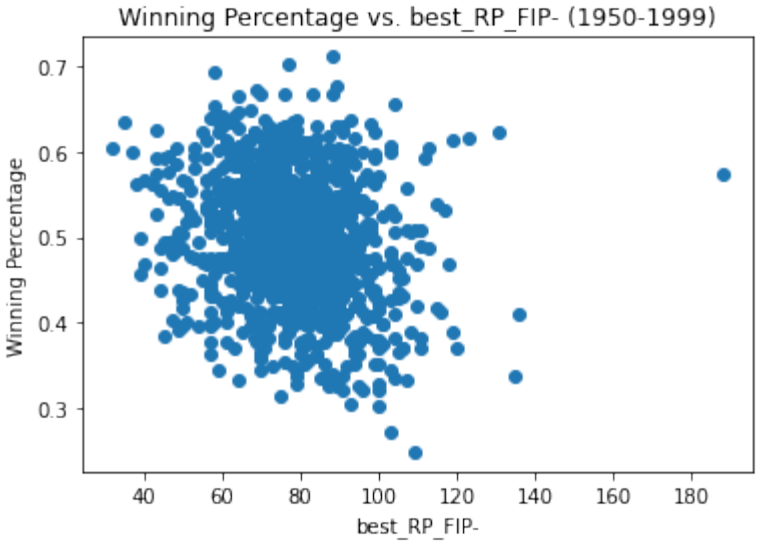

Winning Percentage vs. total_RP_Apps (1950-1999)
Winning Percentage
total_RP_Apps
0.7
0.6
0.5
0.4
0.3
0
100
200
300
400

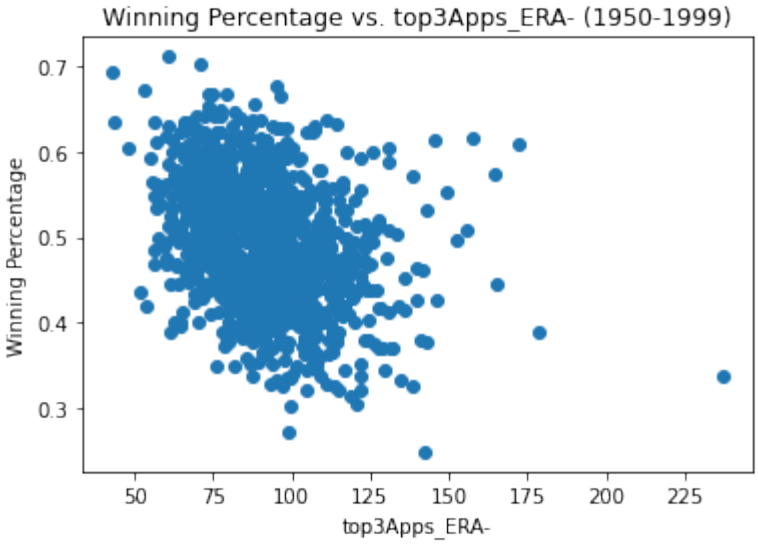

Winning Percentage vs. RP1_Apps (1950-1999)
Winning Percentage
RP1_Apps
0.7
0.6
0.5
0.4
0.3
20
40
60
80
100

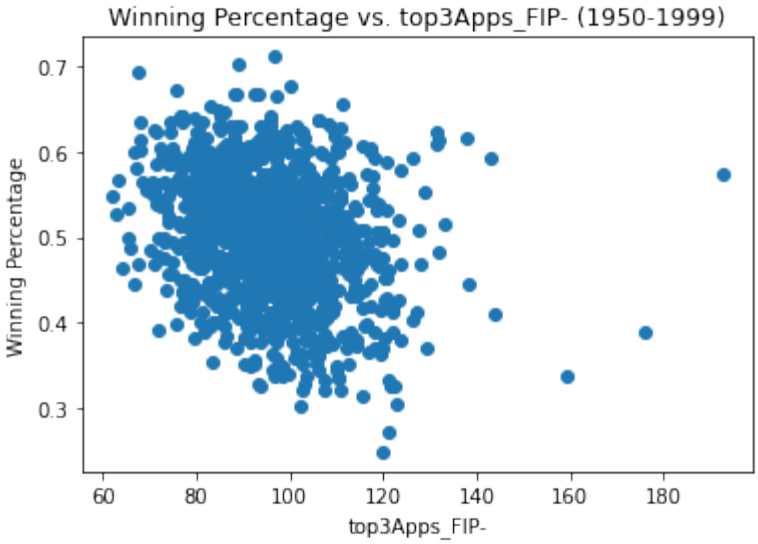

Winning Percentage vs. RP1_ERA- (1950-1999)
Winning Percentage
RP1_ERA-
0.7
0.6
0.5
0.4
0.3
50
100
150
200
250

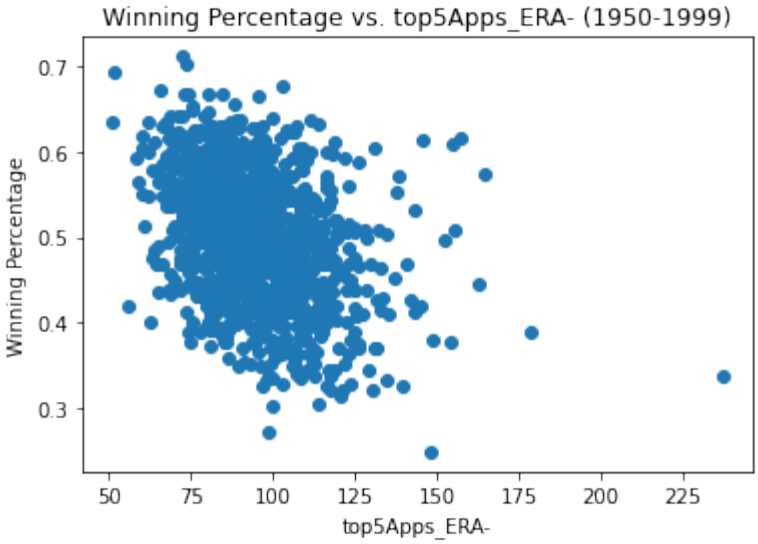

Winning Percentage vs. RP1_FIP- (1950-1999)
Winning Percentage
RP1_FIP-
0.7
0.6
0.5
0.4
0.3
50
75
100
125
150
175
200
225

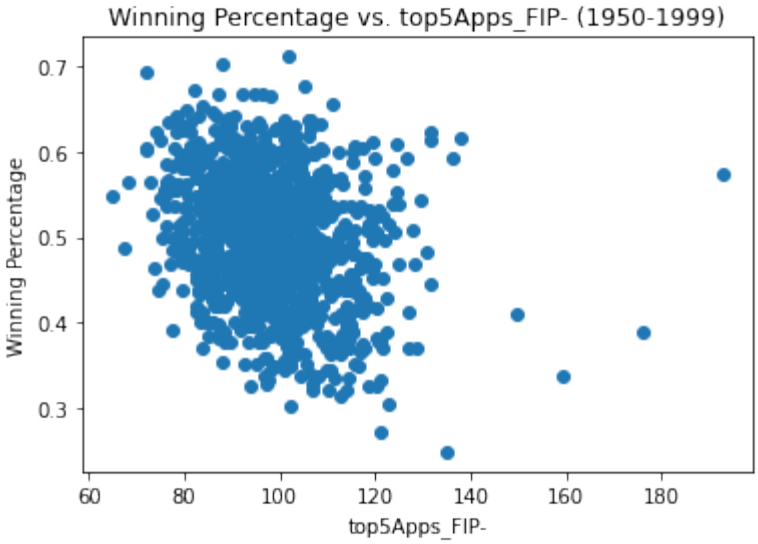

Winning Percentage vs. best_RP_ERA- (1950-1999)
Winning Percentage
best_RP_ERA-
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100
120
140
160


Winning Percentage vs. best_RP_FIP- (1950-1999)
Winning Percentage
0.7
0.6
0.5
0.4
0.3
40
60
80
100
120
140
160
180
best_RP_FIP-

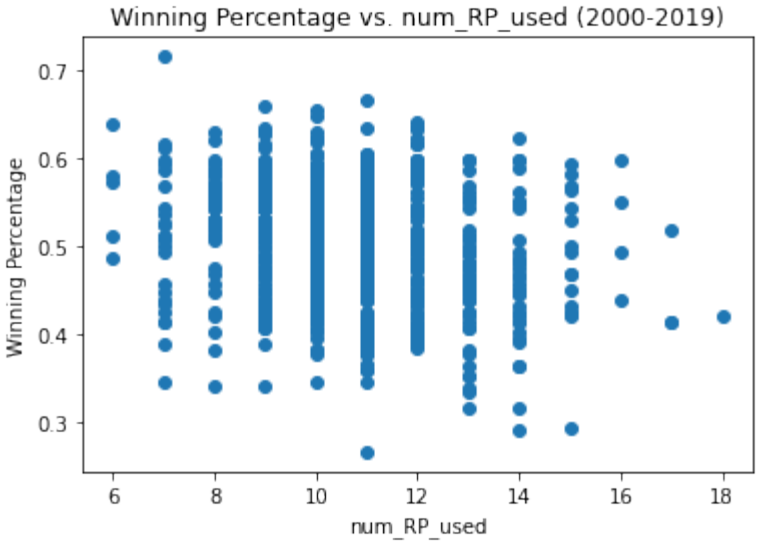

Winning Percentage vs. top3Apps_ERA- (1950-1999)
Winning Percentage
0.7
0.6
0.5
0.4
0.3
50
75
100
125
150
175
200
225
top3Apps_ERA-

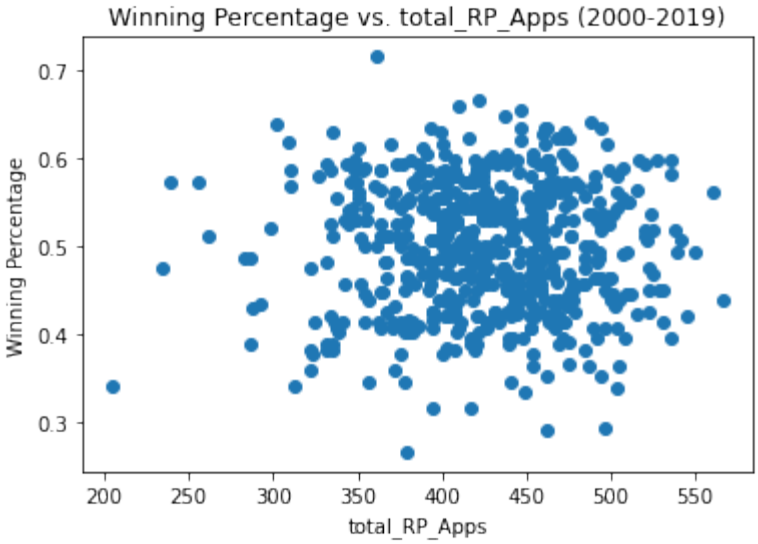

Winning Percentage vs. top3Apps_FIP- (1950-1999)
Winning Percentage
0.7
0.6
0.5
0.4
0.3
60
80
100
120
140
160
180
top3Apps_FIP-

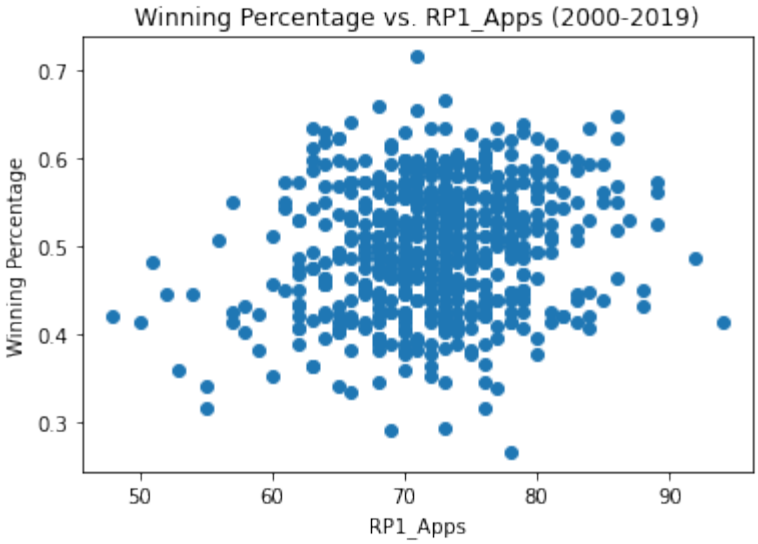


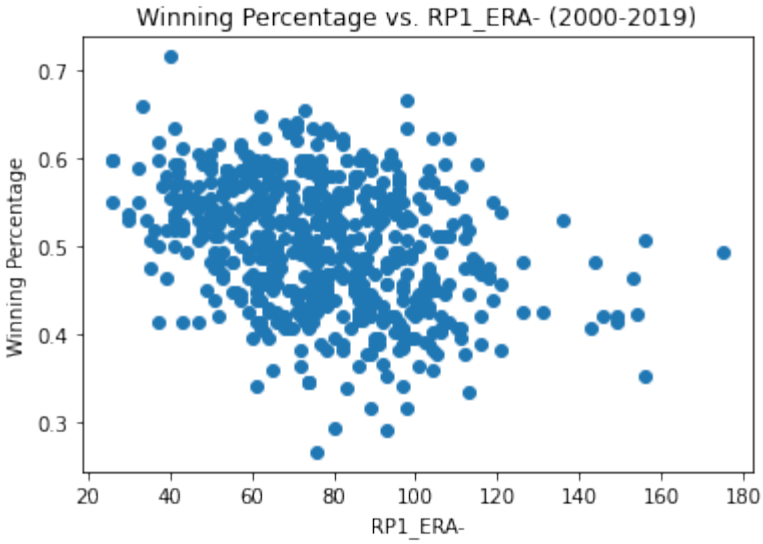


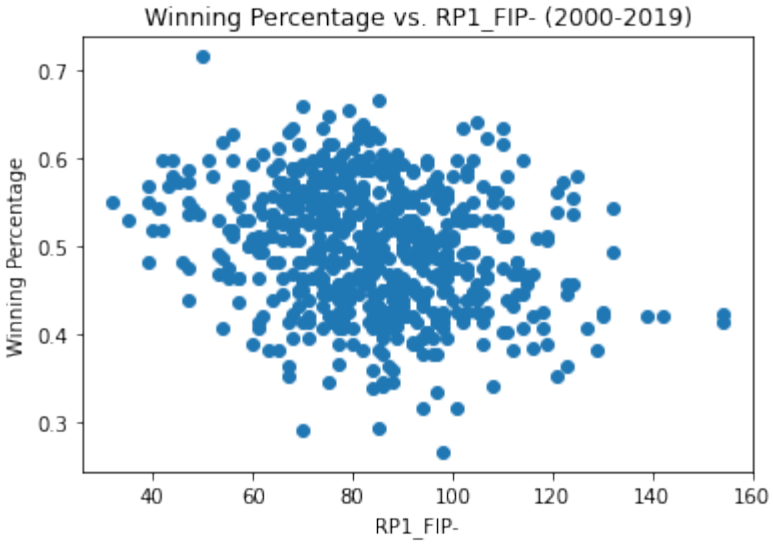
In [ ]:
```
modern_df = final_df[final_df['yearID'] >= 2000]

for pred in predictors:
    plt.scatter(modern_df[pred], modern_df['WinP'])
    plt.title(f"Winning Percentage vs. {pred} (2000-2019)")
    plt.xlabel(pred)
    plt.ylabel("Winning Percentage")
    plt.show()
```

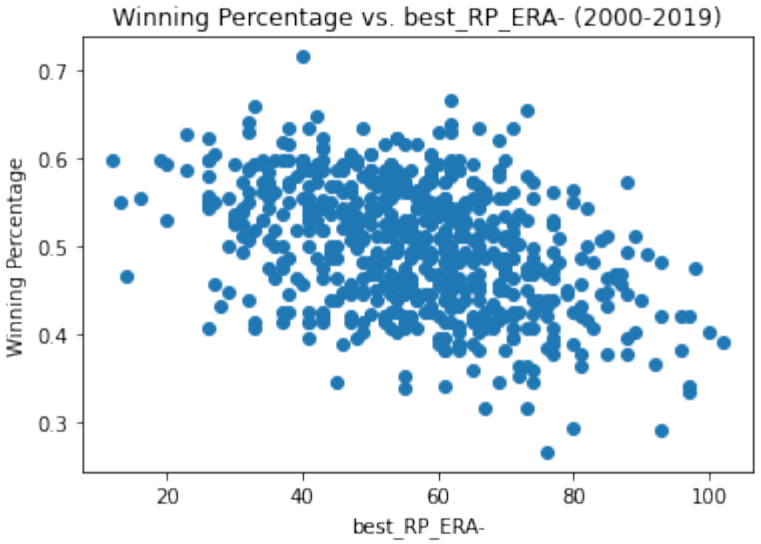

Winning Percentage vs. num_RP_used (2000-2019)
Winning Percentage
num_RP_used
0.7
0.6
0.5
0.4
0.3
6
8
10
12
14
16
18

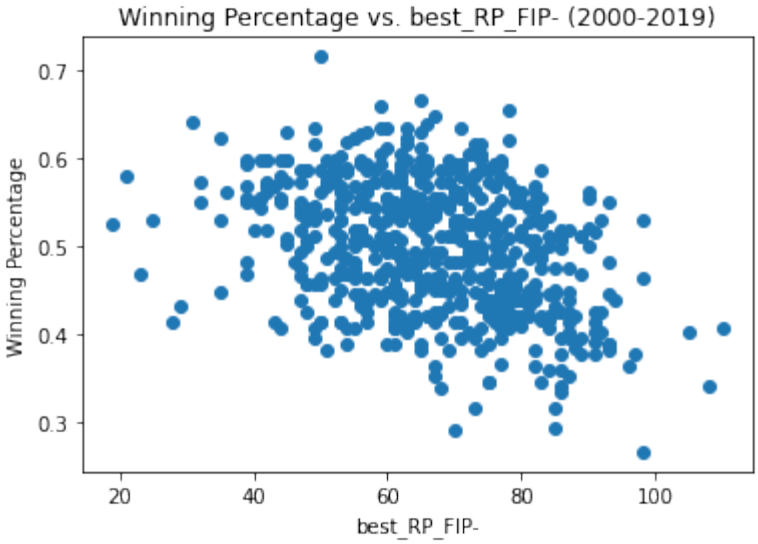

Winning Percentage vs. total_RP_Apps (2000-2019)
Winning Percentage
total_RP_Apps
0.7
0.6
0.5
0.4
0.3
200
250
300
350
400
450
500
550

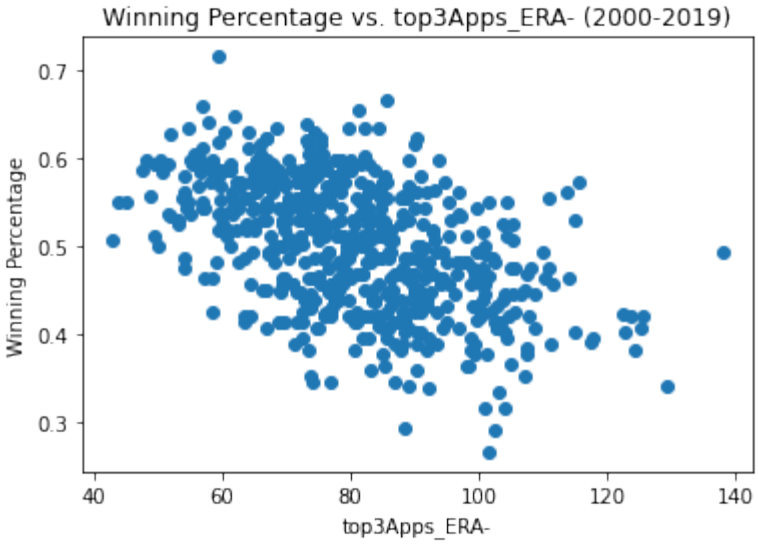

Winning Percentage vs. RP1_Apps (2000-2019)
Winning Percentage
RP1_Apps
0.7
0.6
0.5
0.4
0.3
50
60
70
80
90

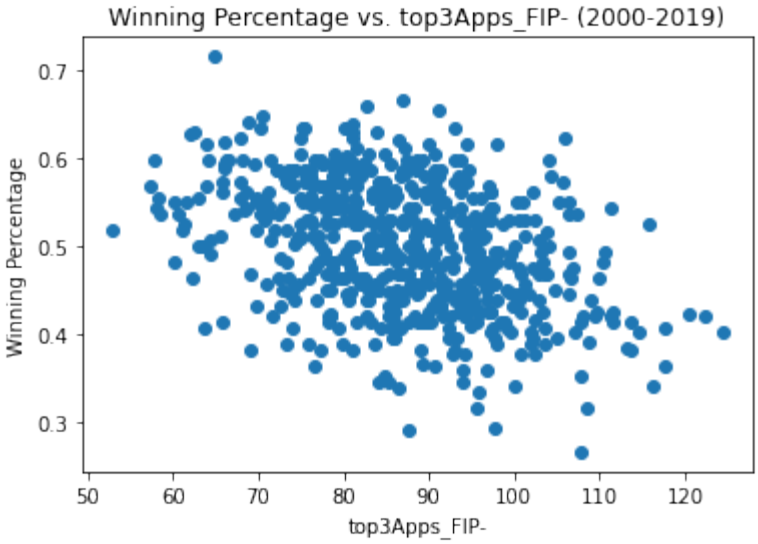

Winning Percentage vs. RP1_ERA- (2000-2019)
Winning Percentage
RP1_ERA-
0.7
0.6
0.5
0.4
0.3
20
40
60
80
100
120
140
160
180

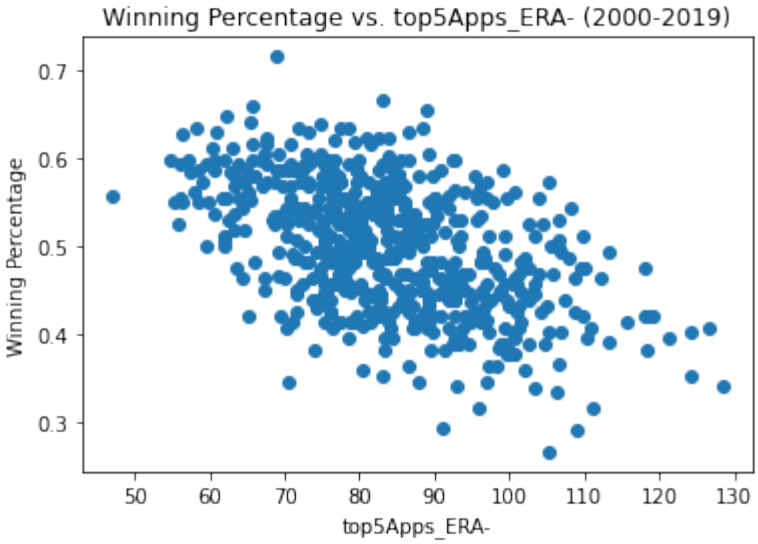

Winning Percentage vs. RP1_FIP- (2000-2019)
Winning Percentage
RP1_FIP-
0.7
0.6
0.5
0.4
0.3
40
60
80
100
120
140
160

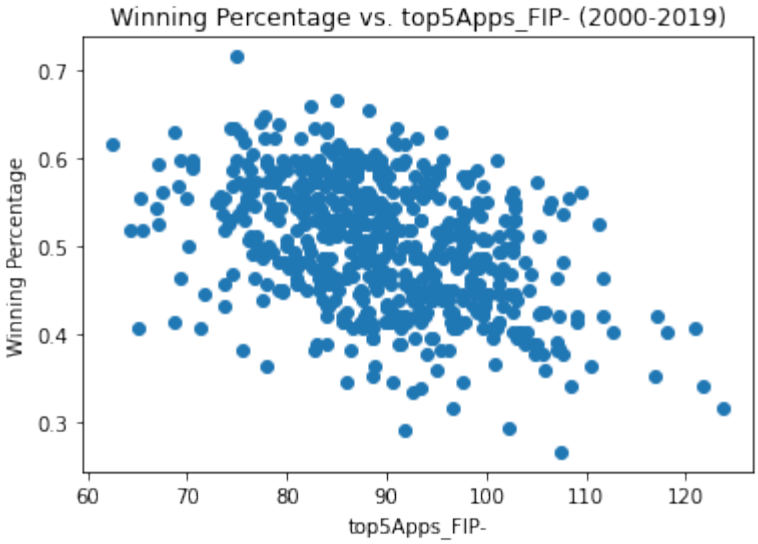

Winning Percentage vs. best_RP_ERA- (2000-2019)
Winning Percentage
best_RP_ERA-
0.7
0.6
0.5
0.4
0.3
20
40
60
80
100


Winning Percentage vs. best_RP_FIP- (2000-2019)
Winning Percentage
0.7
0.6
0.5
0.4
0.3
20
40
60
80
100
best_RP_FIP-


Winning Percentage vs. top3Apps_ERA- (2000-2019)
Winning Percentage
0.7
0.6
0.5
0.4
0.3
40
60
80
100
120
140
top3Apps_ERA-


Winning Percentage vs. top3Apps_FIP- (2000-2019)
Winning Percentage
0.7
0.6
0.5
0.4
0.3
50
60
70
80
90
100
110
120
top3Apps_FIP-





Using Relief Pitching Stats to Predict Excess Winning Percentage

1. Setup of Common Dataframes

In [ ]:
```
# Create 1950 - 1999 ("early") datasets
early_df = final_df[(final_df['yearID'] >= 1950)&(final_df['yearID'] < 2000)]
early_standard_df = early_df.copy()
for col in predictors:
    early_standard_df[col] = ((early_df[col] - early_df[col].mean()) \
                              / early_df[col].std())
predictors_1950_df = early_standard_df[predictors]

# Create post 2000 ("modern") datasets
final_df_post2000 = final_df[final_df['yearID'] >= 2000]
post2000_standard_df = final_df_post2000.copy()
for col in predictors:
    post2000_standard_df[col] = ((final_df_post2000[col] -
final_df_post2000[col].mean()) \
                                 / final_df_post2000[col].std())
predictors_2000_df = post2000_standard_df[predictors]
```

2. Correlations

In [ ]:
```
print('Correlations with Excess Winning Percentage in Early Dataset:\n')
    for pred in predictors:
        print(f"{pred}:     {early_df['ExcessWinP'].corr(early_df[pred])}")
```

Correlations with Excess Winning Percentage in Early Dataset:

num_RP_used: -0.01263672429288702
total_RP_Apps: 0.035470498018905124
RP1_Apps: 0.1135214207644494
RP1_ERA-: -0.08277308184983966
RP1_FIP-: -0.07229656929046015
best_RP_ERA-: -0.02152803542860749
best_RP_FIP-: -0.05193263106399975
top3Apps_ERA-: -0.05589592514578952
top3Apps_FIP-: -0.041286352893841145
top5Apps_ERA-: -0.004866908942188151
top5Apps_FIP-: -0.02618167866888331

In [ ]:
```
print('Correlations with Excess Winning Percentage in Modern Dataset:\n')
    for pred in predictors:
        print(f"{pred}:
    {final_df_post2000['ExcessWinP'].corr(final_df_post2000[pred])}")
```

Correlations with Excess Winning Percentage in Modern Dataset:

num_RP_used: -0.03993284757414983
total_RP_Apps: 0.016142040740656262
RP1_Apps: 0.1097489531708642
RP1_ERA-: -0.08395882594126938
RP1_FIP-: -0.09022669103560213
best_RP_ERA-: -0.14457764271081025
best_RP_FIP-: -0.12944725180364614
top3Apps_ERA-: -0.14703896807814046
top3Apps_FIP-: -0.13804865144329945
top5Apps_ERA-: -0.14329375620648166
top5Apps_FIP-: -0.14436197105405096

3. Direct Relationships

In [ ]:
```
df_dict = {}
    for pred in predictors:
        nona_df = early_df[early_df[pred].notna()]
        x = nona_df[pred]
        x = sm.add_constant(x)
        y = nona_df['ExcessWinP']
        temp_slr_model = sm.OLS(y, x).fit()
        df_dict[pred] = {'Intercept Coefficient': temp_slr_model.params[0],
                         'Intercept P-value': temp_slr_model.pvalues[0],
                         'Predictor Coefficient': temp_slr_model.params[1],
                         'Predictor P-value': temp_slr_model.pvalues[1],
                         'Adjusted R-Squared': temp_slr_model.rsquared_adj,
                         'F-stat': temp_slr_model.fvalue,
                         'F-test P-value': temp_slr_model.f_pvalue}

    xwp_early_models_df = pd.DataFrame.from_dict(df_dict, orient='index')
```

```
xwp_early_models_df
```

Out[ ]:

	Intercept Coefficient	Intercept P-value	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-stat	F-test P-value
num_RP_used	0.000923	0.686039	-0.000152	0.669679	-0.000717	0.182072	0.669679
total_RP_Apps	-0.002242	0.264377	0.000011	0.231020	0.000382	1.436105	0.231020
RP1_Apps	-0.013555	0.000166	0.000227	0.000121	0.012021	14.883109	0.000121
RP1_ERA-	0.007167	0.007090	-0.000085	0.005127	0.005980	7.864459	0.005127
RP1_FIP-	0.008928	0.016546	-0.000098	0.014538	0.004354	5.989853	0.014538
best_RP_ERA-	0.001785	0.485970	-0.000027	0.467352	-0.000413	0.528585	0.467352
best_RP_FIP-	0.006702	0.084849	-0.000086	0.079388	0.001822	3.082892	0.079388
top3Apps_ERA-	0.006677	0.064383	-0.000074	0.058982	0.002250	3.572927	0.058982
top3Apps_FIP-	0.007267	0.167326	-0.000076	0.163234	0.000829	1.946520	0.163234
top5Apps_ERA-	0.000656	0.871610	-0.000007	0.869502	-0.000853	0.027004	0.869502
top5Apps_FIP-	0.005319	0.380196	-0.000054	0.376722	-0.000191	0.781984	0.376722

In [ ]:
```
df_dict = {}
for pred in predictors:
    nona_df = final_df_post2000[final_df_post2000[pred].notna()]
    x = nona_df[pred]
    x = sm.add_constant(x)
    y = nona_df['ExcessWinP']
    temp_slr_model = sm.OLS(y, x).fit()
    df_dict[pred] = {'Intercept Coefficient': temp_slr_model.params[0],
                     'Intercept P-value': temp_slr_model.pvalues[0],
                     'Predictor Coefficient': temp_slr_model.params[1],
                     'Predictor P-value': temp_slr_model.pvalues[1],
                     'Adjusted R-Squared': temp_slr_model.rsquared_adj,
                     'F-stat': temp_slr_model.fvalue,
                     'F-test P-value': temp_slr_model.f_pvalue}

xwp_modern_models_df = pd.DataFrame.from_dict(df_dict, orient='index')
xwp_modern_models_df
```

Out[ ]:

	Intercept Coefficient	Intercept P-value	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-stat	F-test P-value
num_RP_used	0.005198	0.337232	-0.000484	0.328816	-0.000075	0.955113	0.328816
total_RP_Apps	-0.003095	0.695491	0.000007	0.693139	-0.001411	0.155859	0.693139
RP1_Apps	-0.030664	0.007349	0.000424	0.007128	0.010393	7.290624	0.007128
RP1_ERA-	0.007000	0.048527	-0.000091	0.039792	0.005389	4.245278	0.039792

RP1_FIP-	0.010045	0.030888	-0.000119	0.027106	0.006482	4.908189	0.027106
best_RP_ERA-	0.012513	0.000627	-0.000221	0.000381	0.019265	12.766669	0.000381
best_RP_FIP-	0.014665	0.001894	-0.000220	0.001485	0.015112	10.191211	0.001485
top3Apps_ERA-	0.018578	0.000388	-0.000231	0.000302	0.019984	13.214743	0.000302
top3Apps_FIP-	0.024320	0.000783	-0.000281	0.000697	0.017417	11.617748	0.000697
top5Apps_ERA-	0.022017	0.000506	-0.000262	0.000430	0.018895	12.536201	0.000430
top5Apps_FIP-	0.032237	0.000421	-0.000361	0.000389	0.019203	12.727799	0.000389

4. Relationships With Standardized Predictors

In [ ]:
```
df_dict = {}
for pred in predictors:
    nona_df = early_standard_df[early_standard_df[pred].notna()]
    x = nona_df[pred]
    x = sm.add_constant(x)
    y = nona_df['ExcessWinP']
    temp_slr_model = sm.OLS(y, x).fit()
    df_dict[pred] = {'Intercept Coefficient': temp_slr_model.params[0],
                     'Intercept P-value': temp_slr_model.pvalues[0],
                     'Predictor Coefficient': temp_slr_model.params[1],
                     'Predictor P-value': temp_slr_model.pvalues[1],
                     'Adjusted R-Squared': temp_slr_model.rsquared_adj,
                     'F-statistic': temp_slr_model.fvalue,
                     'F-test P-value': temp_slr_model.f_pvalue}

xwp_early_standard_models_df = pd.DataFrame.from_dict(df_dict, orient='index')
xwp_early_standard_models_df.to_csv('xwp_early_standard_models.csv')
xwp_early_standard_models_df
```

Out[ ]:

	Intercept Coefficient	Intercept P-value	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-statistic	F-test P-value
num_RP_used	2.560072e-17	1.0	-0.000311	0.669679	-0.000717	0.182072	0.669679
total_RP_Apps	2.560072e-17	1.0	0.000874	0.231020	0.000382	1.436105	0.231020
RP1_Apps	2.560072e-17	1.0	0.002796	0.000121	0.012021	14.883109	0.000121
RP1_ERA-	2.560072e-17	1.0	-0.002039	0.005127	0.005980	7.864459	0.005127
RP1_FIP-	2.560072e-17	1.0	-0.001781	0.014538	0.004354	5.989853	0.014538
best_RP_ERA-	2.560072e-17	1.0	-0.000530	0.467352	-0.000413	0.528585	0.467352

best_RP_FIP-	2.560072e-17	1.0	-0.001279	0.079388	0.001822	3.082892	0.079388
top3Apps_ERA-	2.560072e-17	1.0	-0.001377	0.058982	0.002250	3.572927	0.058982
top3Apps_FIP-	2.560072e-17	1.0	-0.001017	0.163234	0.000829	1.946520	0.163234
top5Apps_ERA-	2.560072e-17	1.0	-0.000120	0.869502	-0.000853	0.027004	0.869502
top5Apps_FIP-	2.560072e-17	1.0	-0.000645	0.376722	-0.000191	0.781984	0.376722

In [ ]:
```
df_dict = {}
for pred in predictors:
    nona_df = post2000_standard_df[post2000_standard_df[pred].notna()]
    x = nona_df[pred]
    x = sm.add_constant(x)
    y = nona_df['ExcessWinP']
    temp_slr_model = sm.OLS(y, x).fit()
    df_dict[pred] = {'Intercept Coefficient': temp_slr_model.params[0],
                     'Intercept P-value': temp_slr_model.pvalues[0],
                     'Predictor Coefficient': temp_slr_model.params[1],
                     'Predictor P-value': temp_slr_model.pvalues[1],
                     'Adjusted R-Squared': temp_slr_model.rsquared_adj,
                     'F-statistic': temp_slr_model.fvalue,
                     'F-test P-value': temp_slr_model.f_pvalue}

xwp_modern_standard_models_df = pd.DataFrame.from_dict(df_dict, orient='index')
xwp_modern_standard_models_df.to_csv('xwp_modern_standard_models.csv')
xwp_modern_standard_models_df
```

Out[ ]:

	Intercept Coefficient	Intercept P-value	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-statistic	F-test P-value
num_RP_used	1.144917e-16	1.0	-0.000980	0.328816	-0.000075	0.955113	0.328816
total_RP_Apps	1.144917e-16	1.0	0.000396	0.693139	-0.001411	0.155859	0.693139
RP1_Apps	1.144917e-16	1.0	0.002692	0.007128	0.010393	7.290624	0.007128
RP1_ERA-	1.144917e-16	1.0	-0.002059	0.039792	0.005389	4.245278	0.039792
RP1_FIP-	1.144917e-16	1.0	-0.002213	0.027106	0.006482	4.908189	0.027106
best_RP_ERA-	1.144917e-16	1.0	-0.003546	0.000381	0.019265	12.766669	0.000381
	1.144917e-						

best_RP_FIP-	16	1.0	-0.003175	0.001485	0.015112	10.191211	0.001485
top3Apps_ERA-	1.144917e-16	1.0	-0.003607	0.000302	0.019984	13.214743	0.000302
top3Apps_FIP-	1.144917e-16	1.0	-0.003386	0.000697	0.017417	11.617748	0.000697
top5Apps_ERA-	1.144917e-16	1.0	-0.003515	0.000430	0.018895	12.536201	0.000430
top5Apps_FIP-	1.144917e-16	1.0	-0.003541	0.000389	0.019203	12.727799	0.000389

Using Bullpen Stats to Predict Winning Percentage Directly

1. Correlations

In [ ]:
```
print('Correlations with Winning Percentage in Early Dataset:\n')
    for pred in predictors:
        print(f"{pred}:     {early_df['WinP'].corr(early_df[pred])}")
```

Correlations with Winning Percentage in Early Dataset:

num_RP_used: -0.0940152874207297
total_RP_Apps: -0.012140260910676808
RP1_Apps: 0.06098882711149492
RP1_ERA-: -0.2696887805042949
RP1_FIP-: -0.2102063127506319
best_RP_ERA-: -0.2722622024743359
best_RP_FIP-: -0.19052324162361528
top3Apps_ERA-: -0.34017299184285427
top3Apps_FIP-: -0.2487598124917441
top5Apps_ERA-: -0.35519159658500604
top5Apps_FIP-: -0.264931229646232

In [ ]:
```
print('Correlations with Winning Percentage in Modern Dataset:\n')
    for pred in predictors:
        print(f"{pred}:     {final_df_post2000['WinP'].corr(final_df_post2000[pred])}")
```

Correlations with Winning Percentage in Modern Dataset:

num_RP_used: -0.20853955819303574
total_RP_Apps: -0.0013217733746189583
RP1_Apps: 0.18082935309026688
RP1_ERA-: -0.3209695891511317
RP1_FIP-: -0.23526186870761945
best_RP_ERA-: -0.4141269242258202
best_RP_FIP-: -0.3483907654873895
top3Apps_ERA-: -0.49288572184694557
top3Apps_FIP-: -0.402112166326959
top5Apps_ERA-: -0.5082340523142115
top5Apps_FIP-: -0.4273771826151209

2. SLR Models

In [ ]:
```
df_dict = {}
for pred in predictors:
    nona_df = early_df[early_df[pred].notna()]
    x = nona_df[pred]
    x = sm.add_constant(x)
    y = nona_df['WinP']
    temp_slr_model = sm.OLS(y, x).fit()
    df_dict[pred] = {'Intercept Coefficient': temp_slr_model.params[0],
                     'Intercept P-value': temp_slr_model.pvalues[0],
                     'Predictor Coefficient': temp_slr_model.params[1],
                     'Predictor P-value': temp_slr_model.pvalues[1],
                     'Adjusted R-Squared': temp_slr_model.rsquared_adj,
                     'F-statistic': temp_slr_model.fvalue,
                     'F-test P-value': temp_slr_model.f_pvalue}

winp_early_models_df = pd.DataFrame.from_dict(df_dict, orient='index')
winp_early_models_df
```

Out[ ]:

	Intercept Coefficient	Intercept P-value	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-statistic	F-test P-value
num_RP_used	0.519797	0.000000e+00	-0.003399	1.469336e-03	0.007969	10.166174	1.469336e-03
total_RP_Apps	0.501511	0.000000e+00	-0.000011	6.819324e-01	-0.000730	0.168045	6.819324e-01
RP1_Apps	0.477371	3.727946e-249	0.000366	3.933293e-02	0.002846	4.256218	3.933293e-02
RP1_ERA-	0.569239	0.000000e+00	-0.000830	1.757132e-20	0.071919	89.418083	1.757132e-20
RP1_FIP-	0.577050	2.284344e-308	-0.000853	7.170150e-13	0.043348	52.701538	7.170150e-13
best_RP_ERA-	0.566898	0.000000e+00	-0.001024	7.381858e-21	0.073315	91.269990	7.381858e-21
best_RP_FIP-	0.572943	8.879358e-290	-0.000946	8.530319e-11	0.035454	42.939651	8.530319e-11
top3Apps_ERA-	0.621062	0.000000e+00	-0.001349	2.486994e-32	0.114942	149.181016	2.486994e-32
top3Apps_FIP-	0.630509	2.886264e-228	-0.001373	1.441884e-17	0.061059	75.198221	1.441884e-17
top5Apps_ERA-	0.642686	0.000000e+00	-0.001520	2.730398e-35	0.125395	164.588250	2.730398e-35
top5Apps_FIP-	0.660605	1.439760e-202	-0.001644	8.520106e-20	0.069373	86.055033	8.520106e-20

In [ ]:
```
df_dict = {}
```

```
for pred in predictors:
    nona_df = final_df_post2000[final_df_post2000[pred].notna()]
    x = nona_df[pred]
    x = sm.add_constant(x)
    y = nona_df['WinP']
    temp_slr_model = sm.OLS(y, x).fit()
    df_dict[pred] = {'Intercept Coefficient': temp_slr_model.params[0],
                     'Intercept P-value': temp_slr_model.pvalues[0],
                     'Predictor Coefficient': temp_slr_model.params[1],
                     'Predictor P-value': temp_slr_model.pvalues[1],
                     'Adjusted R-Squared': temp_slr_model.rsquared_adj,
                     'F-statistic': temp_slr_model.fvalue,
                     'F-test P-value': temp_slr_model.f_pvalue}

winp_modern_models_df = pd.DataFrame.from_dict(df_dict, orient='index')
winp_modern_models_df
```

Out[ ]:

	Intercept Coefficient	Intercept P-value	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-statistic	F-test P-value
num_RP_used	0.580348	1.234697e-156	-0.007491	2.546115e-07	0.041889	27.188672	2.546115e-07
total_RP_Apps	0.500674	9.379348e-76	-0.000002	9.742255e-01	-0.001670	0.001045	9.742255e-01
RP1_Apps	0.350218	1.064335e-23	0.002072	8.313271e-06	0.031082	20.215176	8.313271e-06
RP1_ERA-	0.579216	4.969553e-248	-0.001028	7.632841e-16	0.101522	68.682629	7.632841e-16
RP1_FIP-	0.577527	3.234414e-185	-0.000921	5.451774e-09	0.053768	35.037450	5.451774e-09
best_RP_ERA-	0.606127	2.795227e-259	-0.001876	2.898229e-26	0.170116	123.787327	2.898229e-26
best_RP_FIP-	0.616872	2.463868e-202	-0.001757	1.456624e-18	0.119907	82.609778	1.456624e-18
top3Apps_ERA-	0.684449	1.288310e-217	-0.002292	4.775545e-38	0.241670	191.893938	4.775545e-38
top3Apps_FIP-	0.709825	1.067808e-151	-0.002422	1.006295e-24	0.160292	115.343503	1.006295e-24
top5Apps_ERA-	0.731305	2.490088e-194	-0.002756	1.007445e-40	0.257062	208.257912	1.007445e-40
top5Apps_FIP-	0.782706	1.093145e-130	-0.003164	4.889090e-28	0.181284	133.633840	4.889090e-28

3. SLR Models With Standardized Predictors

In [ ]:
```
df_dict = {}
for pred in predictors:
    nona_df = early_standard_df[early_standard_df[pred].notna()]
    x = nona_df[pred]
    x = sm.add_constant(x)
    y = nona_df['WinP']
    temp_slr_model = sm.OLS(y, x).fit()
    df_dict[pred] = {'Intercept Coefficient': temp_slr_model.params[0],
                     'Intercept P-value': temp_slr_model.pvalues[0],
                     'Predictor Coefficient': temp_slr_model.params[1],
                     'Predictor P-value': temp_slr_model.pvalues[1],
                     'Adjusted R-Squared': temp_slr_model.rsquared_adj,
                     'F-statistic': temp_slr_model.fvalue,
                     'F-test P-value': temp_slr_model.f_pvalue}

wp_early_standard_models_df = pd.DataFrame.from_dict(df_dict, orient='index')
wp_early_standard_models_df.to_csv('wp_early_standard_models.csv')
wp_early_standard_models_df
```

Out[ ]:

	Intercept Coefficient	Intercept P-value	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-statistic	F-test P-value
num_RP_used	0.49921	0.0	-0.006944	1.469336e-03	0.007969	10.166174	1.469336e-03
total_RP_Apps	0.49921	0.0	-0.000897	6.819324e-01	-0.000730	0.168045	6.819324e-01
RP1_Apps	0.49921	0.0	0.004504	3.933293e-02	0.002846	4.256218	3.933293e-02
RP1_ERA-	0.49921	0.0	-0.019918	1.757132e-20	0.071919	89.418083	1.757132e-20
RP1_FIP-	0.49921	0.0	-0.015525	7.170150e-13	0.043348	52.701538	7.170150e-13
best_RP_ERA-	0.49921	0.0	-0.020108	7.381858e-21	0.073315	91.269990	7.381858e-21
best_RP_FIP-	0.49921	0.0	-0.014071	8.530319e-11	0.035454	42.939651	8.530319e-11
top3Apps_ERA-	0.49921	0.0	-0.025124	2.486994e-32	0.114942	149.181016	2.486994e-32
top3Apps_FIP-	0.49921	0.0	-0.018372	1.441884e-17	0.061059	75.198221	1.441884e-17
top5Apps_ERA-	0.49921	0.0	-0.026233	2.730398e-35	0.125395	164.588250	2.730398e-35
top5Apps_FIP-	0.49921	0.0	-0.019567	8.520106e-20	0.069373	86.055033	8.520106e-20

In [ ]:
```
df_dict = {}
for pred in predictors:
    nona_df = post2000_standard_df[post2000_standard_df[pred].notna()]
    x = nona_df[pred]
    x = sm.add_constant(x)
    y = nona_df['WinP']
    temp_slr_model = sm.OLS(y, x).fit()
    df_dict[pred] = {'Intercept Coefficient': temp_slr_model.params[0],
                     'Intercept P-value': temp_slr_model.pvalues[0],
                     'Predictor Coefficient': temp_slr_model.params[1],
                     'Predictor P-value': temp_slr_model.pvalues[1],
                     'Adjusted R-Squared': temp_slr_model.rsquared_adj,
                     'F-statistic': temp_slr_model.fvalue,
                     'F-test P-value': temp_slr_model.f_pvalue}

wp_modern_standard_models_df = pd.DataFrame.from_dict(df_dict, orient='index')
wp_modern_standard_models_df.to_csv('wp_modern_standard_models.csv')
wp_modern_standard_models_df
```

Out[ ]:

	Intercept Coefficient	Intercept P-value	Predictor Coefficient	Predictor P-value	Adjusted R-Squared	F-statistic	F-test P-value
num_RP_used	0.499924	0.0	-0.015157	2.546115e-07	0.041889	27.188672	2.546115e-07
total_RP_Apps	0.499924	0.0	-0.000096	9.742255e-01	-0.001670	0.001045	9.742255e-01
RP1_Apps	0.499924	0.0	0.013143	8.313271e-06	0.031082	20.215176	8.313271e-06
RP1_ERA-	0.499924	0.0	-0.023328	7.632841e-16	0.101522	68.682629	7.632841e-16
RP1_FIP-	0.499924	0.0	-0.017099	5.451774e-09	0.053768	35.037450	5.451774e-09
best_RP_ERA-	0.499924	0.0	-0.030098	2.898229e-26	0.170116	123.787327	2.898229e-26
best_RP_FIP-	0.499924	0.0	-0.025321	1.456624e-18	0.119907	82.609778	1.456624e-18
top3Apps_ERA-	0.499924	0.0	-0.035823	4.775545e-38	0.241670	191.893938	4.775545e-38
top3Apps_FIP-	0.499924	0.0	-0.029225	1.006295e-24	0.160292	115.343503	1.006295e-24
top5Apps_ERA-	0.499924	0.0	-0.036938	1.007445e-40	0.257062	208.257912	1.007445e-40
top5Apps_FIP-	0.499924	0.0	-0.031062	4.889090e-28	0.181284	133.633840	4.889090e-28