A Bayesian Approach to Fantasy Baseball

Adam Mandelson
ISYE6240
25-April-2021

Introduction

This project explores weekly matchup data from my fantasy baseball league for the 2016-2019 seasons.
Parts of this analysis are motivated by Peadar Coyle's Rugby Analysis case study, and PyMC3's GLM: Logistic Regression example.

The data was collected using the Yahoo Fantasy Baseball API and is available in the `full_data.csv` file.

The league consists of 14 teams and each season usually consists of 21 weekly matchups.
Scoring is based on 18 statistical categories. Weekly "scores" are based on the head-to-head matchup in each category. Category wins, losses and ties are compiled each week and throughout the season to determine the league standings.

The 2016-2019 seasons were chosen because the categories were consistent throughout those four seasons.
They consist of the following counting (aggregate) categories as identified in the data:

> **Batting**: `b_BB` : Walks; `b_HR` : Home Runs; `b_K` : Strikeouts; `b_NSB` : Net Stolen Bases; `b_R` : Runs; `b_RBI` : Runs Batted In.
> **Pitching**: `p_BB` : Walks; `p_IP` : Innings Pitched; `p_K` : Strikeouts; `p_L` : Losses; `p_NSVH` : Net Saves and Holds; `p_QS` : Quality Starts; `p_W` : Wins.

The rest are rate categories:

> **Batting**: `b_AVGr` : Batting Average; `b_OBPr` : On-Base Percentage; `b_SLGr` : Slugging Percentage.
> **Pitching**: `p_ERAr` : Earned-run Average; `p_WHIPr` : Walks plus Hits per Innings Pitched.

As noted above, the season standings are not influenced by the overall weekly winner, and consist of an aggregate score. For example, in a weekly matchup of two teams, the result is not purely a win or a loss. The "winning" team might win 10 categories, lose six and tie two, and the "score" of 10-6-2 is simply added to their standings.

Despite this, Yahoo's API includes a "weekly winning team" category, and for the sake of exploring the data and Bayesian approaches, the concept of a team winning the weekly matchup will be used to develop exploratory models.

Motivation

Ideally, this analysis could be used to infer a parameter or a threshold that would lead to more wins in each category. One thought was to identify key categories to focus on and develop a predictive model for weekly matchup wins based on individual categories. However, I found that the variances in the data and the large number of categories meant that each category has a relatively small individual impact on the overall weekly likelihood of a "win".

Likewise, as we'll see below, the data is highly-correlated in ways that make sense: a hit results in an increase in batting average, on-base percentage, and slugging percentage; a home run results in an increase in home runs, runs, runs batted in, slugging percentage, and on-base percentage.

As such, this analysis aims to explore the data and the Bayesian logistic regression toolset in the `PyMC3` library, and to make general observations about the data and how the models could be used or fine-tuned.

Lastly, I should note that the high-dimensionality in the data meant that attempts to elicit priors were ineffective, and sometimes caused errors in the computations. Based on experiences this semester in OpenBugs in which we defined uninformative priors, I felt it was a good idea to defer to PyMC3's defaults for generalized linear models, which are a Flat, uninformative prior distribution for the intercept, and a Normal distribution with mean=0 and precision=1.0E-6 for all regressors.

Data Import and Analysis

In [1]:
```
import pandas as pd
import os

import warnings
warnings.filterwarnings('ignore')

# Import data directories, or load file if in the same directory
# full_data.csv will be included with this notebook
try:
    from config import DATA_DIRECTORY, LEAGUE_DATA_DIRECTORY
    league_directory = LEAGUE_DATA_DIRECTORY
    data_directory = DATA_DIRECTORY
    df = pd.read_csv(os.path.join(data_directory, "full_data.csv"))
    df.info()
except FileNotFoundError:
    df = pd.read_csv("./full_data.csv")
    df.info()

%matplotlib inline
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22644 entries, 0 to 22643
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   season        22644 non-null  int64
 1   week          22644 non-null  int64
 2   n_days        22644 non-null  int64
 3   team_id       22644 non-null  int64
 4   category      22644 non-null  object
 5   value         22644 non-null  float64
 6   won_category  22644 non-null  float64
 7   won_week      22644 non-null  bool
dtypes: bool(1), float64(2), int64(4), object(1)
memory usage: 1.2+ MB
```

The data exported from Yahoo is long, 22644 lines, each representing a single category value for the season, week and team_id. The `won_category` field is an integer value of 1 or 0 depending on whether the team won (1), lost (0) or tied (0).

A bit of adjusting is necessary. The 'won_category' field for each line has NA values that will be replaced with zeroes.
The season, week and team_id fields will be changed to strings to create an `id` field that identifies all of the categories and values for a specific team during a specific week within a specific season. This will help pivot the data to look at the overall weekly winners.

In [2]:
```
# Fill in NA values.
df['won_category'] = df['won_category'].fillna(0) # There are ties

# Create a unique 'id' to turn long data for each category for each week for each team into weekly team data w
df['season'] = df['season'].astype(str)
df['week'] = df['week'].astype(str)
df['team_id'] = df['team_id'].astype(str)
df['id'] = df['season'].str.cat(df.loc[:, ['week','team_id']], sep='_') # Ex: 2019_5_14 for 18 categories

# Change the won_category field back to bool
df['won_category'] = df['won_category'].astype(bool)

# Example:
df.sample(5)
```

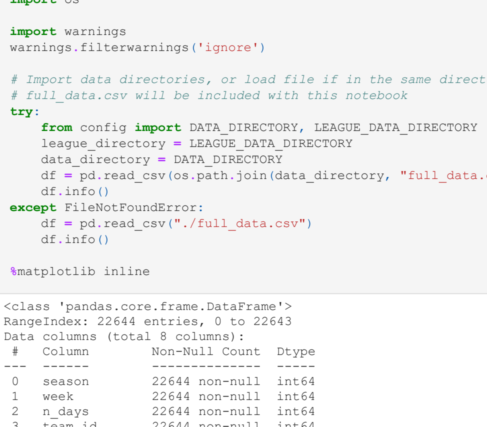
Out[2]:

	season	week	n_days	team_id	category	value	won_category	won_week	id
20592	2019	15	7	9	b_R	55.000	False	True	2019_15_9
12586	2018	6	7	10	b_K	59.000	False	False	2018_6_10
10375	2017	20	7	4	b_SLGr	0.383	False	False	2017_20_4
18553	2019	7	7	4	p_K	79.000	True	False	2019_7_4
18886	2019	8	7	12	b_K	61.000	False	False	2019_8_12

In [3]:
```
# Pivot the data
df_pivoted = df.pivot(index=['id'], columns='category', values = 'value')
df_pivoted['won_week'] = df.groupby('id')['won_week'].mean()
```

In [4]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# A correlation matrix
corr = df_pivoted.corr()

mask = np.zeros_like(corr, dtype=np.bool_)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(8, 6))
cmap = sns.diverging_palette(10, 220, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=0.3, linewidths=0.5, cbar_kws={"shrink": 0.5}, ax=ax)
```

Out[4]: `<AxesSubplot:xlabel='category', ylabel='category'>`



As discussed earlier, there are some clear correlations here, though the highest is about 0.35 (`b_OBPr`). `b_AVGr` and `b_SLGr` are logically correlated, as well.

In [5]:
```
np.absolute(corr['won_week']).sort_values(ascending=False)
```

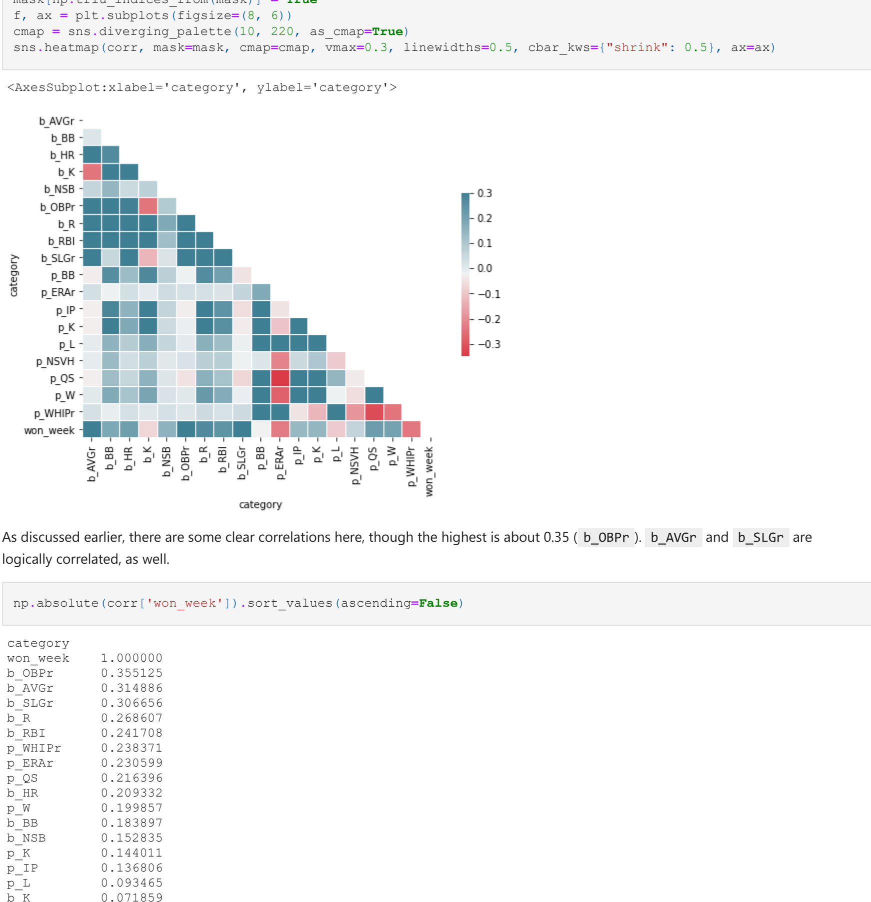
Out[5]:
```
category
won_week    1.000000
b_OBPr      0.355125
b_AVGr      0.314886
b_SLGr      0.306656
b_R         0.268607
b_RBI       0.241708
p_WHIPr     0.238371
p_ERAr      0.230599
p_QS        0.216396
b_HR        0.209332
p_W         0.199857
b_BB        0.183897
b_NSB       0.152835
p_K         0.144011
p_IP        0.136806
p_L         0.093465
b_K         0.071859
p_NSVH      0.058842
p_BB        0.024927
Name: won_week, dtype: float64
```

In [6]:
```
# Create some additional datasets for further exploration
df_batting = df_pivoted.loc[:, [c for c in df_pivoted.columns if 'b_' in c]]
df_pitching = df_pivoted.loc[:, [c for c in df_pivoted.columns if 'p_' in c]]

counts = df_pivoted.loc[:, [c for c in df_pivoted.columns if not 'r' in c]]
rates = df_pivoted.loc[:, [c for c in df_pivoted.columns if 'r' in c]]
batting_counts = counts.loc[:, [c for c in counts.columns if 'b_' in c]]
batting_rates = rates.loc[:, [c for c in rates.columns if 'b_' in c]]
pitching_counts = counts.loc[:, [c for c in counts.columns if 'p_' in c]]

df_batting_counts = df_batting.loc[:, [c for c in df_batting.columns if not 'r' in c]]
df_batting_rates = df_batting.loc[:, [c for c in df_batting.columns if 'r' in c]]
df_pitching_counts = df_pitching.loc[:, [c for c in df_pitching.columns if not 'r' in c]]
df_pitching_rates = df_pitching.loc[:, [c for c in df_pitching.columns if 'r' in c]]
```

In [7]:
```
# sns.pairplot(df_batting) --> Too large for printing
```

In [8]:
```
# A few pitching categories for further exploration.
sns.pairplot(df_pitching.iloc[:, :5])
```

Out[8]: `<seaborn.axisgrid.PairGrid at 0x2030be4bb50>`

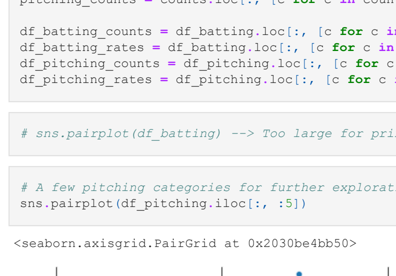


Not surprisingly, the rate categories broadly correlate with Innings Pitched.
The losses category has some interesting points in it, as well, as there is a clear ERA threshold as losses increase.
The strong correlation between innings pitched and strikeouts is logical, but what's notable is the lesser correlation with walks.
It's also worth noting that a few of these categories may be multimodal.

In [9]:
```
# sns.pairplot(rates)
```

In [10]:
```
# The strongest category-won_week correlation
sns.distplot(df_pivoted['b_OBPr'])
```

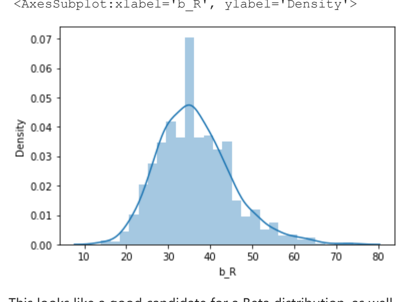
Out[10]: `<AxesSubplot:xlabel='b_OBPr', ylabel='Density'>`



The distribution appears mostly normal, but may be well-represented by a Beta distribution, as well.

In [11]:
```
# The strongest non-rate category-won_week correlation
sns.distplot(df_pivoted['b_R'])
```

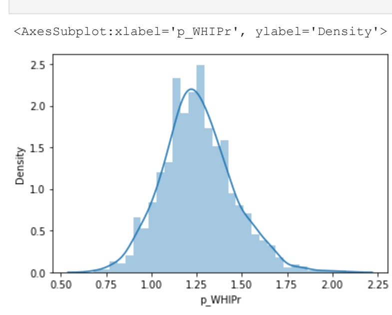
Out[11]: `<AxesSubplot:xlabel='b_R', ylabel='Density'>`



This looks like a good candidate for a Beta distribution, as well.

In [12]:
```
# The strongest pitching category-won_week correlation
sns.distplot(df_pivoted['p_WHIPr'])
```

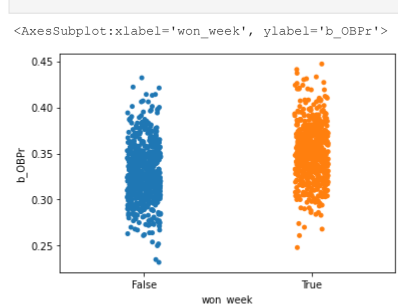
Out[12]: `<AxesSubplot:xlabel='p_WHIPr', ylabel='Density'>`



`p_WHIPr` appears to be normally distributed, but may be multimodal.

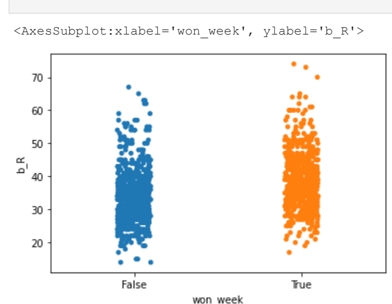
In [13]:
```
sns.stripplot(x="won_week", y="b_OBPr", data=df_pivoted, jitter=True)
```

Out[13]: `<AxesSubplot:xlabel='won_week', ylabel='b_OBPr'>`



In [14]:
```
sns.stripplot(x="won_week", y="b_R", data=df_pivoted, jitter=True)
```

Out[14]: `<AxesSubplot:xlabel='won_week', ylabel='b_R'>`



In [15]:
```
sns.stripplot(x="won_week", y="p_WHIPr", data=df_pivoted, jitter=True)
```

Out[15]: `<AxesSubplot:xlabel='won_week', ylabel='p_WHIPr'>`

Again, because of the large number of categories, we're seeing that there isn't a lot we can infer from these categories about how they contribute to weekly wins, though `p_WHIPr` shows that, like ERA, there appears to be a bit of a threshold where a WHIP above 1.75 or so rarely, if ever, results in a win. This may be a good category for our regression models.

The models

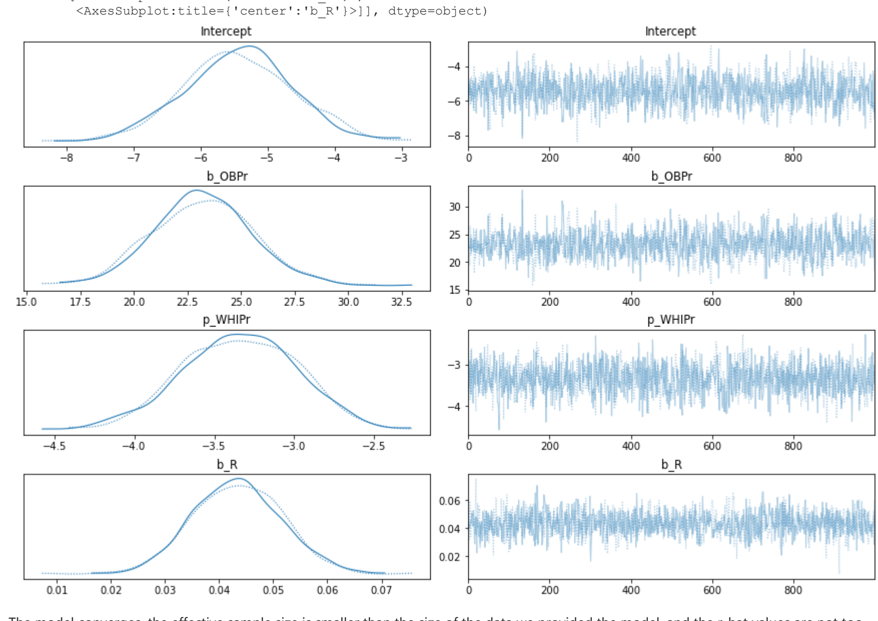
The first model will look at just the three categories we've just explored.

In [16]:
```
import pymc3 as pm
import arviz as az

# This models won_week as the likelihood product of bernoulli n trials
# By default the priors on the regressors are very vague at N(0, 1.0E-16)

with pm.Model() as base_model:
    pm.glm.GLM.from_formula(
        "won_week ~ b_OBPr + p_WHIPr + b_R", df_pivoted, family=pm.glm.families.Binomial()
    )
    base_trace = pm.sample(init='adapt_diag')
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [b_R, p_WHIPr, b_OBPr, Intercept]
```
100.00% [4000/4000 01:02<00:00 Sampling 2 chains, 0 divergences]
```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 72 seconds.
```

In [17]:
```
az.plot_trace(base_trace, compact=True)
```

Out[17]:
```
array([[<AxesSubplot:title={'center':'Intercept'}>,
        <AxesSubplot:title={'center':'Intercept'}>],
       [<AxesSubplot:title={'center':'b_OBPr'}>,
        <AxesSubplot:title={'center':'b_OBPr'}>],
       [<AxesSubplot:title={'center':'p_WHIPr'}>,
        <AxesSubplot:title={'center':'p_WHIPr'}>],
       [<AxesSubplot:title={'center':'b_R'}>,
        <AxesSubplot:title={'center':'b_R'}>]], dtype=object)
```



The model converges, the effective sample size is smaller than the size of the data we provided the model, and the r_hat values are not too high. Also of note, the credible intervals for each regressor do not include zero, which is an issue that other categories will present in upcoming models.

In [18]:
```
az.summary(base_trace)
```

Out[18]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	-5.438	0.809	-6.947	-3.916	0.026	0.018	995.0	1035.0	1.00
b_OBPr	23.214	2.340	19.041	27.880	0.076	0.054	950.0	1065.0	1.01
p_WHIPr	-3.341	0.354	-3.961	-2.655	0.010	0.007	1187.0	1104.0	1.00
b_R	0.043	0.008	0.028	0.058	0.000	0.000	1405.0	924.0	1.00

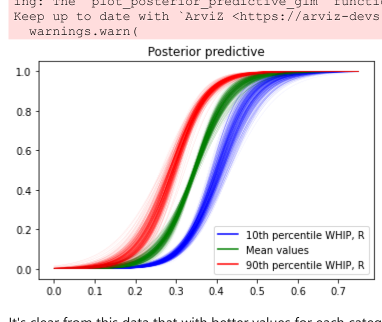
Next we'll look at how the probability of winning the week changes with On-base Percentage values. Since that category corresponds with other hitting metrics, and it showed a correlation with weekly wins, this will show us how the model predicts the probability of a win over changes `b_OBPr` in conjunction with different values of `p_WHIPr` and `b_R`.

In [19]:
```
def lm_full(trace, OBP, WHIP, R):
    shape = np.broadcast(OBP, WHIP, R).shape
    x_norm = np.asarray([np.broadcast_to(x, shape) for x in [OBP, WHIP, R]])
    return 1 / (
        1
        + np.exp(
            -(
                trace["Intercept"]
                + trace["b_OBPr"] * x_norm[0]
                + trace["p_WHIPr"] * x_norm[1]
                + trace["b_R"] * x_norm[2]
            )
        )
    )
```

In [20]:
```
# These values were inferred from the original dataset for the purpose of demonstrating the model's prediction
# values of the regressors

lm = lambda x, samples: lm_full(samples, x, 1.52, 26)
lm2 = lambda x, samples: lm_full(samples, x, 1.24, 35.5)
lm3 = lambda x, samples: lm_full(samples, x, 1.02, 48)
```

In [21]:
```
pm.plot_posterior_predictive_glm(
    base_trace, eval=np.linspace(0, .75, 1000), lm=lm, samples=500, color="blue", alpha=0.15
)
pm.plot_posterior_predictive_glm(
    base_trace, eval=np.linspace(0, .75, 1000), lm=lm2, samples=500, color="green", alpha=0.15
)
pm.plot_posterior_predictive_glm(
    base_trace, eval=np.linspace(0, .75, 1000), lm=lm3, samples=500, color="red", alpha=0.15
)

import matplotlib.lines as mlines

blue_line = mlines.Line2D(["lm"], [], color="b", label="10th percentile WHIP, R")
green_line = mlines.Line2D(["lm2"], [], color="g", label="Mean values")
red_line = mlines.Line2D(["lm3"], [], color="r", label="90th percentile WHIP, R")
plt.legend(handles=[blue_line, green_line, red_line], loc="lower right")
plt.show()
```

```
C:\Users\mandelson\Miniconda3\envs\mypm3env2\lib\site-packages\pymc3\plots\posteriorplot.py:59: DeprecationWarning: The `plot_posterior_predictive_glm` function will migrate to Arviz in a future release.
Keep up to date with `ArviZ <https://arviz-devs.github.io/arviz/>`_ for future updates.
  warnings.warn(
C:\Users\mandelson\Miniconda3\envs\mypm3env2\lib\site-packages\pymc3\plots\posteriorplot.py:59: DeprecationWarning: The `plot_posterior_predictive_glm` function will migrate to Arviz in a future release.
Keep up to date with `ArviZ <https://arviz-devs.github.io/arviz/>`_ for future updates.
  warnings.warn(
C:\Users\mandelson\Miniconda3\envs\mypm3env2\lib\site-packages\pymc3\plots\posteriorplot.py:59: DeprecationWarning: The `plot_posterior_predictive_glm` function will migrate to Arviz in a future release.
Keep up to date with `ArviZ <https://arviz-devs.github.io/arviz/>`_ for future updates.
  warnings.warn(
```



It's clear from this data that with better values for each category, the model's posterior distribution predicts a higher probability of winning the week. It's also interesting to note that the better values of WHIP and R offset lower OBP values to the point that a weekly win is highly probable right around the mean value for OBP.

Other model runs were less informative, and are not worth including here. The same goes for attempts to model scaled data.

One model that I did want to compare was the full dataset in a logistic regression model:

In [22]:
```
with pm.Model() as full_model:

    pm.glm.GLM.from_formula(
        "won_week ~ b_OBPr + b_AVGr + b_SLGr + b_R + p_WHIPr + p_ERAr + b_RBI + b_BB + p_QS + p_W + b_HR + p_IP
    )
    full_trace = pm.sample(init='adapt_diag', cores=2)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [sd, p_BB, b_K, p_NSVH, p_L, b_NSB, p_K, p_IP, b_HR, p_W, p_QS, b_BB, b_RBI, p_ERAr, p_WHIPr, b_R, b_SLGr, b_AVGr, b_OBPr, Intercept]
```
100.00% [4000/4000 02:28<00:00 Sampling 2 chains, 0 divergences]
```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 160 seconds.
```

In [23]:
```
az.summary(full_trace)
```

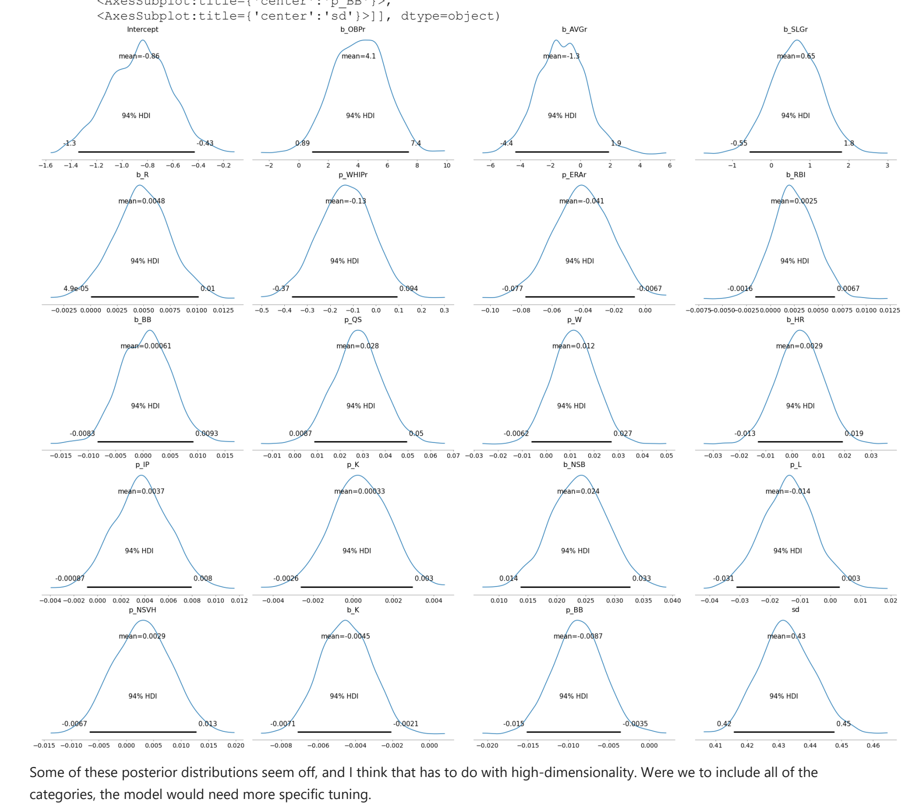
Out[23]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	-0.862	0.246	-1.345	-0.430	0.008	0.005	1043.0	1086.0	1.00
b_OBPr	4.074	1.820	0.886	7.443	0.067	0.047	747.0	1088.0	1.00
b_AVGr	-1.262	1.658	-4.350	1.920	0.055	0.039	907.0	1209.0	1.00
b_SLGr	0.649	0.643	-0.553	1.822	0.018	0.013	1309.0	1225.0	1.00
b_R	0.005	0.003	0.000	0.010	0.000	0.000	1535.0	1377.0	1.00
p_WHIPr	-0.130	0.122	-0.369	0.094	0.004	0.003	1109.0	1219.0	1.00
p_ERAr	-0.041	0.019	-0.077	-0.007	0.001	0.000	1322.0	1377.0	1.00
b_RBI	0.003	0.002	-0.002	0.007	0.000	0.000	1993.0	1659.0	1.00
b_BB	0.001	0.005	-0.008	0.009	0.000	0.000	759.0	1155.0	1.00
p_QS	0.028	0.011	0.009	0.050	0.000	0.000	1674.0	1625.0	1.00
p_W	0.012	0.009	-0.006	0.027	0.000	0.000	1635.0	1415.0	1.00
b_HR	0.003	0.009	-0.013	0.019	0.000	0.000	1385.0	1293.0	1.00
p_IP	0.004	0.002	-0.001	0.008	0.000	0.000	1100.0	1268.0	1.00
p_K	0.000	0.001	-0.003	0.003	0.000	0.000	1824.0	1665.0	1.00
b_NSB	0.024	0.005	0.014	0.033	0.000	0.000	2609.0	1268.0	1.01
p_L	-0.014	0.009	-0.031	0.003	0.000	0.000	1624.0	1414.0	1.00
p_NSVH	0.003	0.005	-0.007	0.013	0.000	0.000	2455.0	1385.0	1.00
b_K	-0.005	0.001	-0.007	-0.002	0.000	0.000	1612.0	1533.0	1.00
p_BB	-0.009	0.003	-0.015	-0.004	0.000	0.000	1417.0	1397.0	1.00
sd	0.433	0.009	0.416	0.448	0.000	0.000	2035.0	1382.0	1.00

Some of these coefficients have slightly higher r_hat values, or low effective sample size values. A number of them contain zero within their confidence interval, though I think with more distinct priors that could be avoided.

In [24]:
```
az.plot_posterior(full_trace)
```

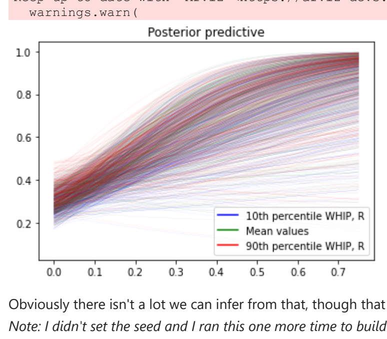
Out[24]:
```
array([[<AxesSubplot:title={'center':'Intercept'}>,
        <AxesSubplot:title={'center':'b_OBPr'}>,
        <AxesSubplot:title={'center':'b_AVGr'}>,
        <AxesSubplot:title={'center':'b_SLGr'}>],
       [<AxesSubplot:title={'center':'b_R'}>,
        <AxesSubplot:title={'center':'p_WHIPr'}>,
        <AxesSubplot:title={'center':'p_ERAr'}>,
        <AxesSubplot:title={'center':'b_RBI'}>],
       [<AxesSubplot:title={'center':'b_BB'}>,
        <AxesSubplot:title={'center':'p_QS'}>,
        <AxesSubplot:title={'center':'p_W'}>,
        <AxesSubplot:title={'center':'b_HR'}>],
       [<AxesSubplot:title={'center':'p_IP'}>,
        <AxesSubplot:title={'center':'p_K'}>,
        <AxesSubplot:title={'center':'b_NSB'}>,
        <AxesSubplot:title={'center':'p_L'}>],
       [<AxesSubplot:title={'center':'p_NSVH'}>,
        <AxesSubplot:title={'center':'b_K'}>,
        <AxesSubplot:title={'center':'p_BB'}>,
        <AxesSubplot:title={'center':'sd'}>]], dtype=object)
```



Some of these posterior distributions seem off, and I think that has to do with high-dimensionality. Were we to include all of the categories, the model would need more specific tuning.
For the purposes of demonstration, we'll also plot the same predictive plots we did above, but with the new model and sampled posterior distributions.

In [25]:
```
pm.plot_posterior_predictive_glm(
    full_trace, eval=np.linspace(0, .75, 1000), lm=lm, samples=500, color="blue", alpha=0.15
)
pm.plot_posterior_predictive_glm(
    full_trace, eval=np.linspace(0, .75, 1000), lm=lm2, samples=500, color="green", alpha=0.15
)
pm.plot_posterior_predictive_glm(
    full_trace, eval=np.linspace(0, .75, 1000), lm=lm3, samples=500, color="red", alpha=0.15
)

import matplotlib.lines as mlines

blue_line = mlines.Line2D(["lm"], [], color="b", label="10th percentile WHIP, R")
green_line = mlines.Line2D(["lm2"], [], color="g", label="Mean values")
red_line = mlines.Line2D(["lm3"], [], color="r", label="90th percentile WHIP, R")
plt.legend(handles=[blue_line, green_line, red_line], loc="lower right")
plt.show()
```

```
C:\Users\mandelson\Miniconda3\envs\mypm3env2\lib\site-packages\pymc3\plots\posteriorplot.py:59: DeprecationWarning: The `plot_posterior_predictive_glm` function will migrate to Arviz in a future release.
Keep up to date with `ArviZ <https://arviz-devs.github.io/arviz/>`_ for future updates.
  warnings.warn(
C:\Users\mandelson\Miniconda3\envs\mypm3env2\lib\site-packages\pymc3\plots\posteriorplot.py:59: DeprecationWarning: The `plot_posterior_predictive_glm` function will migrate to Arviz in a future release.
Keep up to date with `ArviZ <https://arviz-devs.github.io/arviz/>`_ for future updates.
  warnings.warn(
C:\Users\mandelson\Miniconda3\envs\mypm3env2\lib\site-packages\pymc3\plots\posteriorplot.py:59: DeprecationWarning: The `plot_posterior_predictive_glm` function will migrate to Arviz in a future release.
Keep up to date with `ArviZ <https://arviz-devs.github.io/arviz/>`_ for future updates.
  warnings.warn(
```



Obviously there isn't a lot we can infer from that, though that was to be expected.
Note: I didn't set the seed and I ran this one more time to build a pdf file, and some of the values on this last model run looked better, so some of my comments on this final model may not fully apply to the output here.

Ultimately, I think there is a lot of potential for Bayesian analysis to be useful for this data, but probably in the form of individual predictive models or hierarchical models that utilize prior distributions in a logical manner.