NBA Model Data Processing / Pipeline Method:

**Pulling the data**

The first step in this process is obviously acquiring the data. We used two databases to combine historical odds data as well as additional information such as whether a team was playing at home or away, how deep into their season that game was, where the game was played, and more. We accessed ‘The-Odds-API’ to find cheap, accurate historical sports betting odds. Each pull request contains the market data for all of the bookmakers specified under your ‘bookmakers’ parameter or in our case, when left blank, it retrieves all the bookmakers that are carried under your ‘region’ parameter. We used US, EU and UK bookmakers. In the metadata lies a ‘previous timestamp’ value which can be stored and used as the date for the next pull request, ensuring that we are getting the very next datapoint in the series. The data for the API spans from June 2020 through present date, however the scoring data we need to create our ‘target’ variable is not easily accessible for the current season, so our whole data set contains the 2020, 2021 and 2022 seasons. The extra information was pulled from the omprehensive baseball database website, “Retrosheet.org.” For the 2020-2022 seasons, we pulled the ‘game logs’ data sets which contain even more information than we used, and trimmed down to find the variables we thought would be useful, but just as important, be available to find in real time when the model is operational.

**Combining the two datasets**

This step was the most difficult to do primarily because of the logic involved. The goal was to create a new variable in each of the datasets that concatenated the names of two teams playing, the game date, and a value representing if that game was a double header or not. If we could create this variable in each of the sheets using the data belonging to each sheet, then we could easily map values from one to another. In addition, we could account for edge cases with the double header variable so that we don’t interpret two games from a double header as one.

To start, we had to get the team names in the same format between both datasets. The team names in the extra info sheet were abbreviations but full length in the odds data sheet, so we had to map the full-length team names from the odds sheet to the extra info sheet.

It became clear that the odds corresponding to the first team in the odds sheet for each data point and the odds corresponding to the second team in the odds sheet for each data point were not correlated with which team was home or away, but merely in ascending alphabetical order. So the ‘Arizona Cardinals’ data would always come before the ‘Texas Rangers’ data should they be playing each other, no matter who was home or away. Therefore, the ’my\_id’ variable in the odds sheet will always display the opposing teams in alphabetical order. In the extra info sheet, I merely created new columns that contain the concatenated team names in alphabetical order.

Next, the variables representing the start time of the game were different between the two sheets. In the extra info sheet, it was a value such as ‘20201020’ corresponding to October 20, 2020. In the odds sheet, the times were values such as "2021-10-10T15:55:00Z" corresponding to the commence time of the game in GMT. To change this into the same format as the extra sheet, I subtracted 7 hours from the time in GMT and used a python function to extract the date in the right YYYMMDD format. I subtracted 7 hours because it’s the time difference between GMT and MST for most of the year when MST is not in Daylight savings time. As long as the actual start *date* is correct and consistent, the hour doesn’t mean a whole lot.

Now for the hard part. Over the course of the 3 season, I had data on, there were 184 double header events. I didn’t want to throw out this potentially important data but didn’t have a way of identifying double headers in my odds sheet. I made a data structure in python that maps each built in ‘commence time’ metadata variable to the work-in-progress ‘my\_id’ variable (‘team\_1’, ‘team\_2’, ‘date’). It finds all of the unique commence\_times that correspond to each ‘my\_id’ value. If there are 2 values in the commence times for a single ‘my\_id’, it compares the times and makes a new entry such that the my\_id value adds ‘\_1’ to the key of the new key value pair corresponding to the early game and ‘\_2’ to the key of the new key value pair corresponding to the second game. If there was no double header, ‘\_0’ was added to each key for those key value pairs. We could now make a column in both the extra info sheet and the odds sheet that have ‘my\_id’ in the right format. After that, we simply mapped each extra variable from the extra sheet to the odds sheet, and the datasets were combined.

**Cleaning the data**

Cleaning the data was pretty easy. Of the 1.2M observations we had, there were only 39k bad datapoints (fully empty rows). Dropped those and I was ready to go create some more variables.

**Creating more variables**

I wanted to make a better time value that represents the ‘snapshot’ time variable. Because the API holds the most recent updates from each bookmaker before the specified ‘snapshot\_time,’ it’s not always the best indicator of the time the market was last updated. I wanted to create a variable that took the max value from the list of all the bookmakers last\_update time and used that as the snapshot time. I did this because when pulling live odds, the market is typically lightning fast and there is very small delta between the time you pull and the bookmakers last update times.

To do this, first I subtracted 7 hours from each time column except the ‘commence\_time’ because that was already in the right timezone. Then I simply ran a line code that finds the max value of all the bookmakers update times and made a new column with the corresponding variables.

Next, I didn’t know what to do about missing data. Some bookmakers we’rent carried until recently so they have tons and tons of missing data points, and sometimes the books don’t provide betting opportunities like, say, in the middle of an important at-bat. 41% of the dataset is empty for these reasons. For the purposes of this initial model, I filled all missing data points with 0’s. This is one major area for experimentation. What would happen if it was filled with market average instead? Or other method s of filling the data? Not sure….

The next variable I wanted to make was the ‘minutes\_since\_commence’ so that the model could have some sort of information on what point in the game the datapoint was taken. This variable was the commence time of the game subtracted from the snapshot time. I then went on to create a ‘hour\_of\_start’ variable which was merely the hour value of the commence\_time.

I filtered the dataset for data whose minutes\_since\_commence value is greater than or equal to -360: only games that are live or within 6 hours of start were included. This could be modified.

**Transforming the data for the model**

One of the last steps before feeding the data to the model for training and testing was to standardize the continuous numerical variables and encode the categorical variables as one-hot variables and storing the corresponding standardization and encoding objects such that they can be accessed later when the model is running live and needs to convert new data into the same format. This was easy, there are many python libraries that do this easily.

**Train test splits**

More on this later.

**Data set experiments:**

* Filter by magnitude of odds before training to have different strategies for different risk tolerances like we did at Ascend.
  + A strategy where you try to minimize the difference between TPR and FPR while maximizing EV
  + Do strategy 1 to see the EV and then try to develop the head-fake so that we can stay low
* Decode the categorical variables and keep track of how many games these bets are spread across.
* When we pick the maximizing thresholds, we should filter by only thresholds that bet on x number of games or more.
  + When we look at each row of the correct an incorrect predictions, unfilter the right set of columns, make the new my\_id varaiable, and then make a dict that stores a counter as the value for the game\_id. FOR EACH THRESHOLD.