Since I could not successfully install the necessary procedure-- *pip install pymc-bart* for BART model implementation in cursor. Finally, I used Random Forest to mimic BART. The model's name is *train_and_predict_bart.py.* However, in the end, I will introduce the logic of my BART model, even though I do not run it successfully.

The specific process is following:

Key Features of the Model:

1. **Ensemble Learning**: Uses 200 decision trees; Each tree is trained on a bootstrap sample of the data; Final prediction is an average of all tree predictions
2. **Uncertainty Estimation**: Provides mean and median predictions; Calculates 90% confidence intervals (5th to 95th percentile); Stores full distribution of predictions
3. **Feature Engineering**: Original feature names in prediction data are transformed into Standardized names used in model. Define a get_common_features to select the common features in the training dataset and prediction dataset. The selected features are:

Race-level features	Horse-level features
distance	speed_rating
post_position	odds
field_size	weight
purse	recent performance (last 3 races)
surface_code	jockey and trainer statistics
race_type	pedigree information

4. **Data Handling**: Handles missing values by filling with mean values; Ensures feature consistency between training and prediction data; Uses NetCDF format for efficient storage of large datasets

My results (The results are generate by *analyze_race_predictions.py*):

Race number	Race 4	Race 5	Race 6	Race 7
Top Pick	INTERMITTENT FAST	ROCKET NIGHT	TIFFANY TWIST	SATIN BLUE
Predicted finish	6.63	1.58	2.79	7.11
Win probability	18.6%	13.4%	15.6%	8.8%

Note	Strong speed rating of 67.55 and good post position (1)	Excellent speed rating of 107.28 **This race has three very strong contenders (ROCKET NIGHT, PAVED IN GOLD, and BAOBAB)**	Strong speed rating of 93.26 Good post position (9)	Speed rating: 61.29,Good post position (1)

I also want to mention the data preprocessing process.

The *1CleanPredictionData.py* script serves as a data preprocessing and standardization tool for horse racing prediction data. Its primary function is to transform raw racing data into a structured format that can be used by the prediction model. Specifically, it takes raw racing data from *CDX0515.csv* and processes it in two ways:

First, it creates a fully processed version (*CDX0515_processed.csv*) that includes all columns with standardized headers, and second, it generates a filtered version (*CDX0515_filtered.csv*) that contains only the columns that have been mapped to specific header names. The script uses a column mapping file (*column_mapping.csv*) to ensure consistent naming across datasets, which is crucial for the model's feature alignment process.

The *parse_race_data.py* and *parse_results_data.py* scripts form a crucial data processing pipeline for the horse racing prediction model. *parse_race_data.py* processes past performance data from XML files, extracting comprehensive pre-race information including race details (surface, distance, purse), horse information, and historical performance metrics, while also handling data standardization tasks like converting fractional odds to decimal and distance strings to furlongs.

In contrast, *parse_results_data.py* focuses on processing actual race results, capturing detailed outcome information such as finish positions, timing data, payoffs, and speed ratings. Together, these scripts transform raw XML data into structured NetCDF datasets (*processed_race_data.nc* and *processed_results_data.nc*), which serve as the foundation for model training and validation. The *parse_race_data.py* output provides the features needed for making predictions, while *parse_results_data.py* creates the validation dataset that helps assess the model's performance. Both scripts include robust error handling, data cleaning, and standardization procedures to ensure the quality and consistency of the training data.

BART model description

train_and_predict_true_bart

1. **BART Prior (μ)**:

- m=100: Uses 100 regression trees in the ensemble
- alpha=0.95: Controls the tree depth prior (higher values favor deeper trees)
- beta=2.0: Additional tree depth prior parameter
- k=2.0: Controls the variance of leaf values (higher values make predictions more conservative)
- split_prior=0.5: Prior probability of splitting at any node

2. **Observation Noise (σ)**:

- Uses a HalfNormal distribution with scale=5
- This represents the uncertainty in the observations
- The HalfNormal ensures the standard deviation is always positive

3. **Likelihood**:

- Uses a Normal distribution for the observed finish positions
- Mean is the BART prediction (μ)
- Standard deviation is the observation noise (σ)
- **Prediction Process:**
- Uses the trained model to generate posterior predictions
- Calculates mean and standard deviation of predictions
- Converts predictions to win probabilities using softmax (exponential normalization)