Team Outlier Solution

Solution of Problem 1:

1. Data Preprocessing Methodology

The initial phase of the analysis focused on preparing the raw data for further investigation and modeling. This involved several key steps:

- **Data Ingestion:**
  The analysis started by loading two main datasets—one containing match-level information (e.g., match ID, team names, dates) and another providing ball-by-ball delivery details. These datasets were imported from CSV files.

- **Data Cleaning:**

  - **Outlier and Missing Value Handling:**
    The code checked for potential outliers (e.g., runs per match and overs limits) and managed missing values appropriately. For instance, missing values in the 'winner' column were understood as representing matches with no result.

  - **Standardization of Categorical Data:**
    Inconsistencies in team names were resolved by mapping old names (e.g., "Delhi Daredevils") to their current equivalents (e.g., "Delhi Capitals"). Similarly, season values were standardized (e.g., converting "2007/08" to "2008") to ensure consistency.

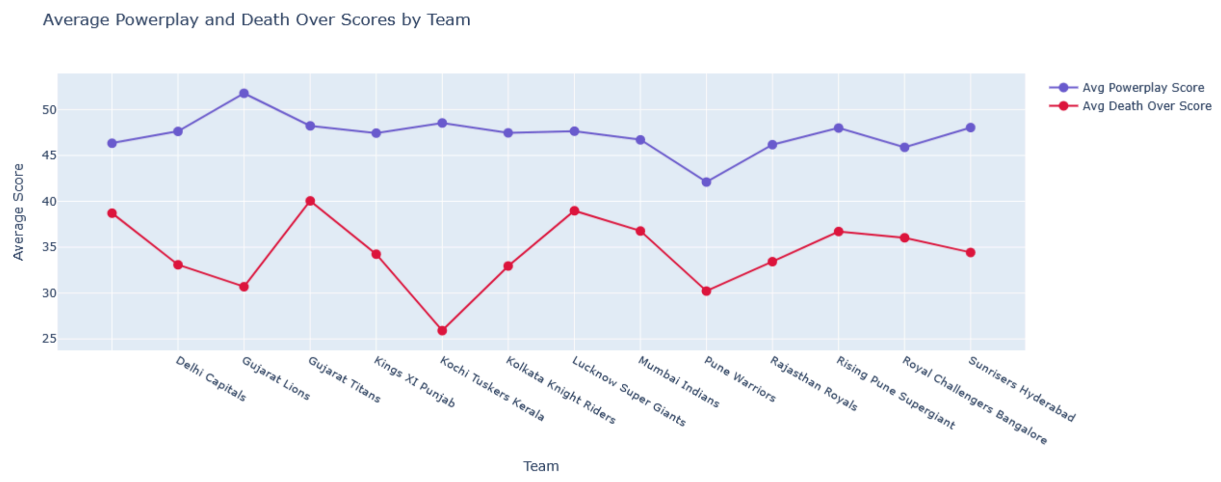
  - **Filtering of Irrelevant Data:**
    Records not relevant for the core analysis, such as Super Over innings (innings 3 and 4 in this case), were removed. This step ensured that subsequent analyses focused on standard gameplay data.

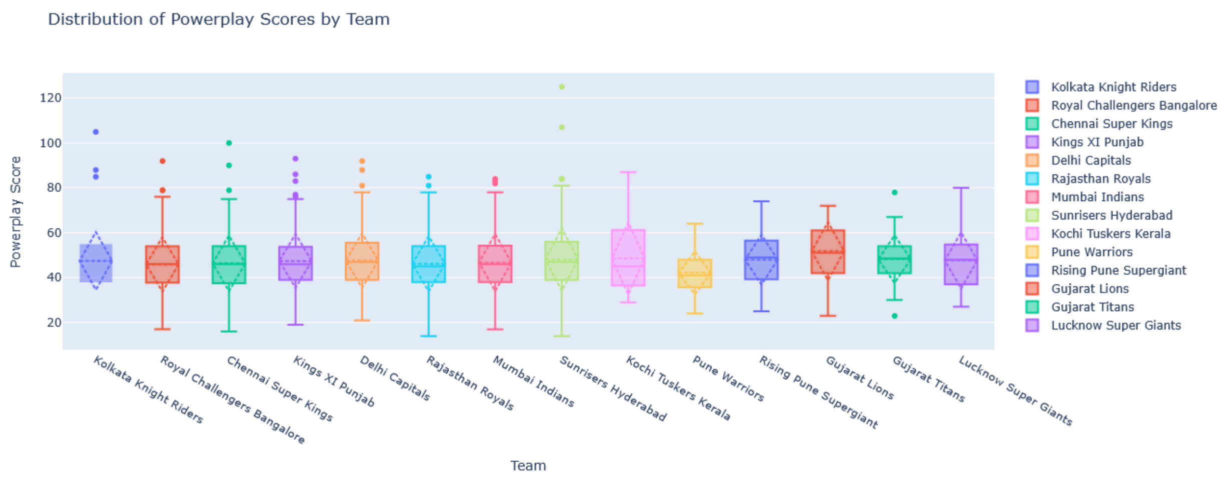
- **Encoding and Transformation (Pre-Modeling):**
  Later in the process, categorical features—particularly those related to teams—were encoded using label encoding. This transformation was crucial for feeding the data into machine learning models where numerical inputs are required.

This preprocessing step set a solid foundation by ensuring that the dataset was consistent, free of irrelevant information, and formatted appropriately for both exploratory analysis and model development.

2. Data Analysis Methodologies

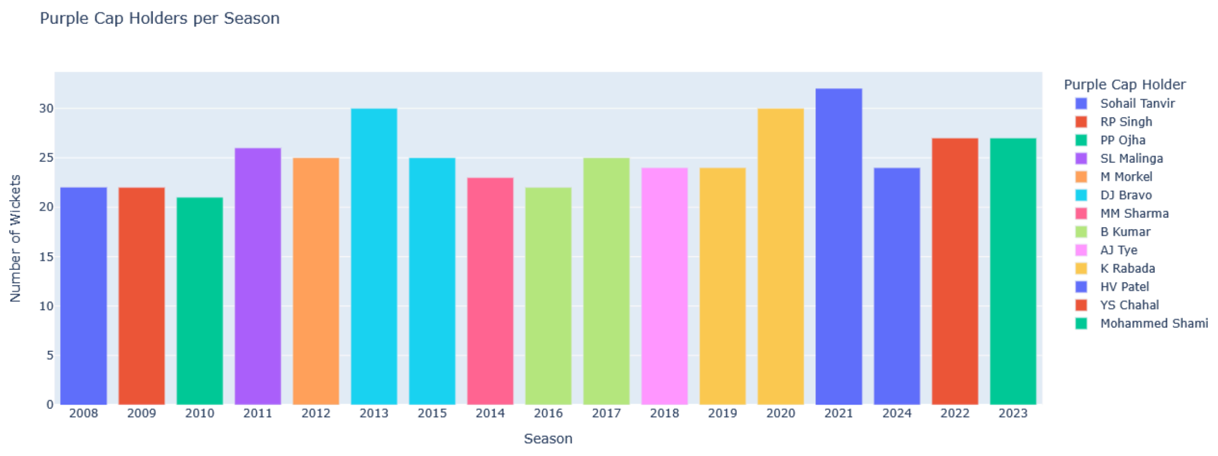
- **Aggregation and Feature Derivation:**
    - Computed team performance metrics (matches played, winning percentages) by grouping match records.
    - Calculated run rates (batting) and economy rates (bowling) from match-level summaries.
    - Derived additional features like total boundaries (4s and 6s) and over-specific scores (powerplay and death overs).



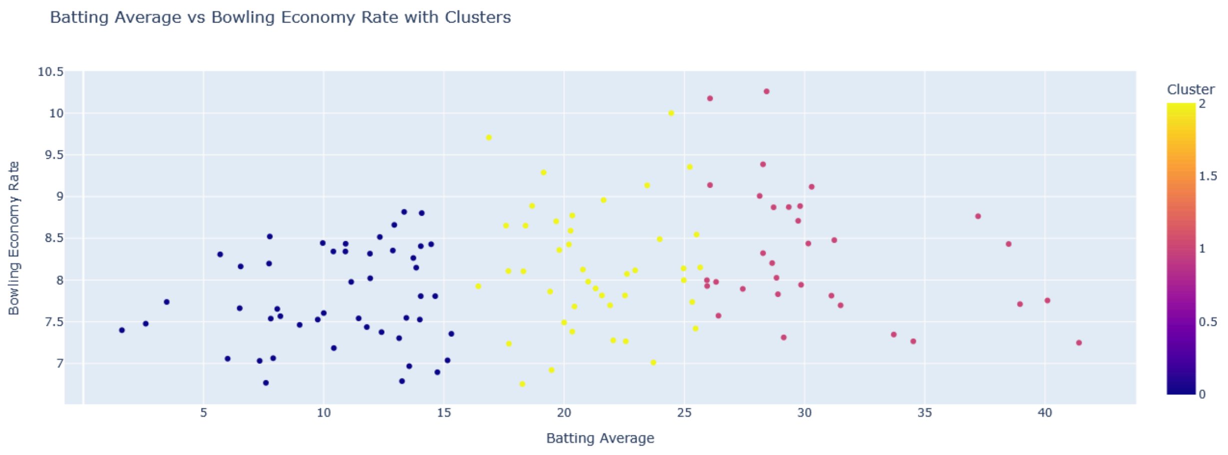


- **Visual Exploration:**
  - Used line and bar charts to show team metrics and performance distributions.
  - Employed scatter and box plots to examine relationships (e.g., batting average vs. strike rate) and score distributions.

    - Analyzed seasonal trends and head-to-head matchups for further insights.



- **Advanced Analysis Techniques:**
    - Applied K-Means clustering to group players based on batting and bowling performance for profiling.
    - Conducted comparative analysis to highlight top performers in various categories.



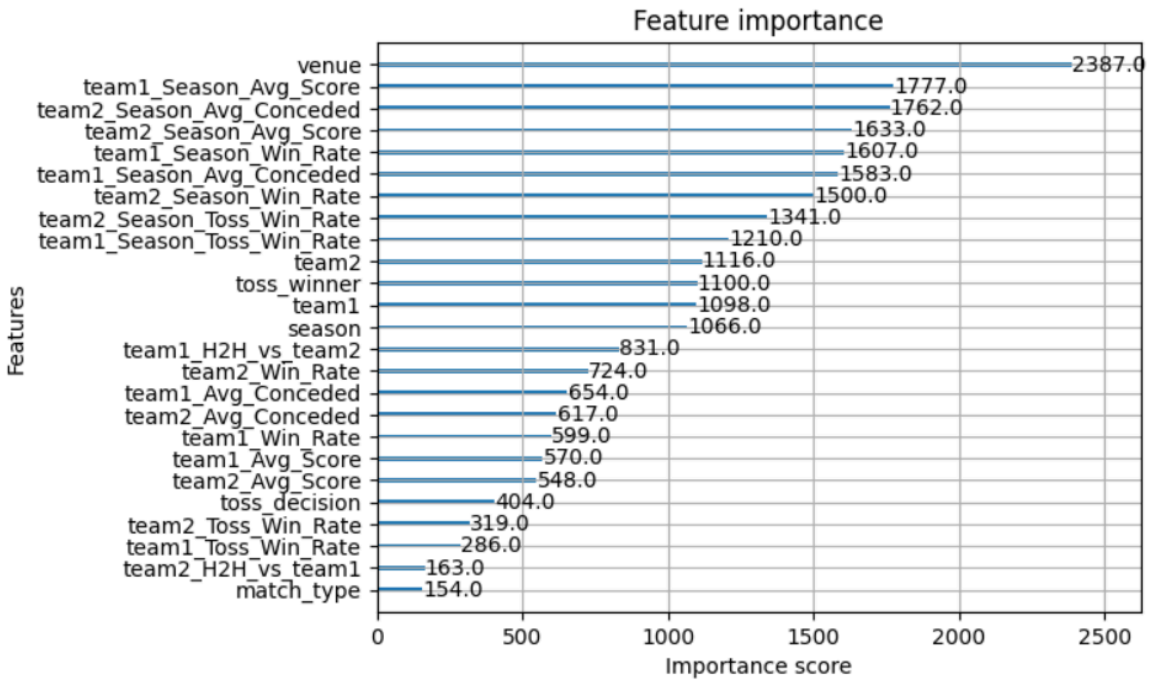
3. Feature Generation for Best Predictions

- **Team Base Features:**
  We calculated basic team stats like win rate, average score, and average runs conceded. We also measured how often winning the toss led to winning the match.

- **Seasonal Features:**
  For each season, we computed team win rates, average scores, and conceded runs. We also looked at head-to-head performance between teams.

- **Merging Data:**
  We combined the base and seasonal features with each match's data. This created separate features for both competing teams, such as team1's win rate vs. team2's.

- **Data Encoding:**
  Categorical data like team names were converted into numbers using label encoding so the model could process them.
- **As we can see in the below graph our top features were engineered by us.**



4. Time-Based Splitting and Modular Model Pipeline

- **Time-Based Data Splitting:**
  We split the data by year instead of randomly, ensuring that training, validation, and test sets follow a real-world time sequence. This time-bound split is our unique selling point.

- **Model Comparison:**
 We evaluated several models including Naive Bayes, LightGBM, and XGBoost. XGBoost outperformed the others.

- **Modular Code Design:**
 The code is organized into classes and functions. Users only need to provide the raw dataset—data cleaning, feature generation, and data splitting happen automatically.

- **Evaluation Metrics:**
 The models were assessed using metrics such as accuracy_score and f1_score, among others, to ensure robust performance evaluation.

Link of the notebook:-

BrainDead IPL Aalysis.ipynb

Solution of Problem 2:

Methodology

Our approach was designed to address the challenges of summarizing long research articles, particularly those in the biomedical domain. We adopted a hybrid pipeline that integrated extractive preprocessing with an abstractive summarization model (Pegasus) fine-tuning.

1. Data Preparation & Preprocessing:

- **Dataset:** We used the PubMed summarization dataset from the ccdv collection, which includes full-length research articles along with their manually annotated abstracts.

- **Preprocessing:**
  We developed a preprocessing function that tokenizes both the input articles and target abstracts using the Pegasus tokenizer. The function ensures that the text is truncated or padded to fit within the model's maximum input length (approximately 1024 tokens). To handle articles exceeding this limit, our approach involved splitting them into smaller chunks with overlapping tokens, generating individual summaries for each chunk, and subsequently merging these into a cohesive final summary.

2. Fine-Tuning Pegasus:

- **Model Choice:** We fine-tuned the `google/pegasus-large` model, an encoder-decoder architecture well-suited for abstractive summarization tasks.

- **Training Setup:**
  We employed Hugging Face's Trainer API with the `Seq2SeqTrainer` and configured training arguments such as learning rate, batch size, and gradient accumulation steps. To optimize memory usage, we enabled mixed precision training (fp16) and set up periodic checkpointing (using `save_steps` and `save_total_limit`) to resume training in case of interruptions.

- **Handling Memory Constraints:**
  Our approach included using techniques like data parallelism and gradient accumulation. However, despite these measures, we encountered GPU out-of-memory errors during training on the Kaggle environment. The errors were primarily due to the high memory demands of processing long input sequences and the computational overhead of our hybrid chunking strategy.

3. Experimental Challenges:

- **Memory Errors:**
  Our team observed that, on Kaggle kernels with limited GPU memory, the fine-tuning process for Pegasus was prone to memory errors. This issue arose particularly when processing large batches or extremely long texts. We attempted various strategies (e.g., gradient accumulation, dynamic padding, and checkpointing) to mitigate these issues, but the memory constraints on Kaggle ultimately led to kernel errors.

- **Impact on Results:**
  Due to these memory limitations, our full-scale experiments could not be completed on the Kaggle platform. The issues encountered were documented in our notebook, which can be accessed here.

4. Future Directions:

- **Scaling Up:**
  In future work, we plan to migrate our experiments to a higher-memory environment to fully exploit the potential of our hybrid summarization approach.

- **Model Optimization:**
  Further optimization in terms of model pruning, quantization, or employing more efficient training frameworks (like DeepSpeed or Fully Sharded Data Parallel) could help mitigate memory bottlenecks and enable scaling on resource-constrained platforms.

The Link of the Notebook:-https://www.kaggle.com/code/deepsutariya/outliers-brain-dead-2k25