**Part 1: Documentation**

**Objective: Play-by-Play Stat Extraction**

To extract play-by-play stats from unstructured PDF files using OCR and text mining for various sports and generate a structured XML format for the stats.

**Stage 0: Data Preparation**

1. **PDF pre-processing and OCR:**
   1. Evaluate PDF for degraded documents (Jaisys) based on availability and clarity of play context information.
   2. Identify and extract useful pages for stats extraction and create a new pdf with these pages: Play by play game record pages; Game rosters; Season schedule (Example page 9-13,25-28,38-41 from Purdue 1992 pdf for game files (games play by play pages, rosters and season schedule)
   3. Prepare the new PDF pages for AWS Textract using Frank’s preprocessing script to remove boarder, contrast, paper color, etc.
   4. Run OCR algorithm using APIs developed by Taheer on the new PDF to convert it into raw text files.
   5. Examine OCR quality. Jaisys validate the usability of the output text file and manually correct the text if the OCR output text is unusable due to too much character misrecognition.
2. **Jaisys text cleaning:**

After ensuring the OCR output text is usable and accurate compared to the PDF file, Jaisys prepares the text file for algorithm based on a list of requirements by Frank.

Requirements list:

* 1. Fix obvious broken lines by putting them into one line:

hands to whitman, runs right tackle, gain 14 (p. harris

mccoy)

* 1. Broken Context (context that are inconsistent in structure, not recognizable, and can't be corrected): restore the context structure

4-3 fitzkee punts to alabama 44 by ogilvie, 36 kards

==>

'X-00 4-3 fitzkee punts to alabama 44 by ogilvie, 36 kards'

* 1. Only possession and side can be letters. Others must be numbers. Fix common misrecognition

ms 4-l p-4 rush jones off left tackle no gain

==>

ms 4-1 p-4 rush jones off left tackle no gain

* 1. Fix typos and misrecognition in the sentence text besides the context:

(5,S); (0,'O'); (1, 'l', 'I')

* 1. Context and main text should be separated by at least 1 white space
  2. (A bit hard) State ball possession in front of every line:

14:56-alabama-rutledge

a-20 1-10 rutledge toss to nathan left, gain x 4 (millen)

a-24 2-6 rutledge hands to nathan to 27 (millen kubin)

==>

14:56

A a-20 1-10 rutledge toss to nathan left, gain x 4 (millen)

A a-24 2-6 rutledge hands to nathan to 27 (millen kubin)

* 1. Mark score reporting using word 'Score' (example 1) and game summary if not already existing. Ensure scores of two teams are in the same line. Ensure this score statement is in its own line (example 2). Use team abbreviations instead of whole school names.

Alabama-14

Penn State-7

==>

SCORE A:14, P:7

p 4-3 ms 3 field goal attempt by griese teter holds kick good p-3 ms 0

(score is enbedded in the sentence so move it into a new line)

==>

p 4-3 ms 3 field goal attempt by griese teter holds kick good

score: p:3, ms:0

* 1. Ensure quarters sentences are in separate lines from other sentences

louisiana superdome, new orleans, la. 2 quarter

==>

louisiana superdome, new orleans, la.

2nd quarter

* 1. Make sure ball spots are using team abbreviations instead of whole school names:

kick off by chatlos to the purdue 18 run back by kuzniewlski to the purdue 34

==>

kick off by chatlos to the p18 run back by kuzniewlski to the p34

* 1. Make sure all yards has 'yards' following the number

p 1-10 p 34 r minnear off right tackle gain 4

==>

p 1-10 p 34 r minnear off right tackle gain 4 yards

* 1. Put the kickoff team as ball possession after the coin toss and before the first kickoff of the game. Make sure the first kickoff is on a separate line (sometimes in the same line of coin toss)

michigan state won the toss purdue defended the south goal and received

kick off by chatlos to the purdue 18 run back by kuzniewlski to the purdue 34

==>

michigan state won the toss purdue defended the south goal and received

possession: ms

kick off by chatlos to the purdue 18 run back by kuzniewlski to the purdue 34

* 1. Make sure coin toss and the following kick off is in separate lines

i won the toss and detected to receive. p will defend the n goal ohl kicked off to willis at the i 1, slipped end fell at i 17

==>

i won the toss and detected to receive. p will defend the n goal

ohl kicked off to willis at the i 1, slipped end fell at i 17

* 1. For each team, the letters used to identify side and possession should be the same. i.e. The team abbreviations should be consistent throughout the file.
  2. Every time stamp is on a new line.

Possession: A 17:50

==>

Possession: A

17:50

* 1. In a context, possession, down, ytg, spot must be separated.

I1/10/I24 Wilson pass over middle for Strader broken up by Mckinnie

==>

I 1/10/I24 Wilson pass over middle for Strader broken up by Mckinnie

* 1. The format of context in the main body of the game must be consistent with the "CONTEXT Format" field in the game Header
  2. Put ball possession in front of every line if the line has context.
  3. Rename the cleaned file as: Team 1 Vs Team 2 YYYY.txt

e.g Purdue Vs Michigan State 1980.txt

* 1. Make sure the team abbreviations in the header are consistent in ball possession and ball spot side in all the plays. Teams with the same initial may cause confusion. When not clear ask Athlyte for clarification.

e.g. (In this game, the original PDF file used "A"(uppercase) for Alabama and "a" (lowercase) for Auburn

Team 1: Alabama

Team 2: Auburn

Team 1: Letter or text used to identify field side "UA"

Team 2 : Letter or text used to identify field side "A"

CONTEXT format [POSS-DOWN-YTG-SPOTSIDE-SPOT]

Original pdf:

A a49 1-10 Peoples slants inside, gains six yard (Cline, Boyd)

A A45 2-4 Brooks at right end, gains seven on the sideline and FIRST DOWN (Harris, Boyd)

Desired:

A 1-10 UA49 Peoples slants inside, gains six yard (Cline, Boyd)

A 2-4 A45 Brooks at right end, gains seven on the sideline and FIRST DOWN (Harris, Boyd)

1. **Jaisys adds Meta information as headers at the top of play-by-play records:**

e.g. Alabama Vs Kentucky 1972

Game title

GameType

Team1 Alabama

Team2 Kentucky

Team 1: Alabama

Team 2: Kentucky

Team 1: Letter or text used to identify field side "A"

Team 2 : Letter or text used to identify field side "K"

Team 1: Letter or text to identify POSSESSION format (abbreviation used) "A"

Team 2: Letter or text to identify POSSESSION format (abbreviation used) "K"

CONTEXT format [POSS-DOWN-YTG-SPOTSIDE-SPOT]

Context POSS: TRUE

POSSESSION start must have "POSSESSION" + Team Possession format": TRUE

Overview of the process (terminology)

Definitions for sentence, part of speech/grammar, identifiers

**Stage 1: Play-by-Play Sentence Parsing (Loop 1 algorithm)**

1. Get all necessary inputs:
   1. Active Roster
   2. Text file from stage 0
   3. Data Models

1.3.1 Context pattern collection in regex.

* + 1. Sentence-pattern model: For each sentence type, a collection of actions keywords are assigned. For each kind of actions/results, a collection of identifiers/synonyms is defined using regex. (create appendix for detailed layout/content of data models)
    2. Matching tuple dictionary: For each parent name what is the expected action-result pair.

1. Match and separate the **Context** (POSS-DOWN-YTG-SIDE-SPOT) from the main sentence text using pre-defined context template in regex.
2. Map every **Sentence** to the sentence type:
   1. Categorize into **Play** versus **Non-play** type of Sentences, match the regex of actions in the sentence. Infer play/non-play based on actions found.
   2. For **Play** sentences continue further processing into scrim/nonscrim.
   3. For **Non-Play** sentences categorize GameHeader and GameStatus.
      1. Perform a 'dictionary' approach (analysis) that also includes game header, TOSS, End of Quarter and score, scoring play summary, Play by play format descriptor, quarter start and end, drive start/possession, Overtime and other non-play information.

3.4. Matching the Context regex in the sentence also is useful to categorize the sentences.

E.g. If a ‘Rush’ pattern is found, the sentence is mapped to a scrim.

If a ‘kick off’ pattern is found, the sentence is mapped to a nonscrim.

Sometimes both a scrim pattern and a nonscrim pattern were found. For these ambiguous matching results, the following rules/assumptions are used:

* If a sentence has a valid context, it must be a scrim.
* If a sentence has touchdown, field goal, or punt patterns, it must be a scrim.
* Otherthan the above 2 situation, a sentence can never be a scrim.
* If both play and nonplay sentence types are matched, the play sentence type takes priority.

Raise alerts for sentence with scrim keywords but no context.

All sentences unable to be matched is assigned as ‘other text’.

1. Within **Play** sentences categorize **scrim and nonscrim**:
   1. To identify Scrim/Nonscrim sentence match the sentence to a Parent Name
      1. By matching regex in the sentence, find **action** keywords and **result** keywords in the sentence.
      2. If the action-result pair exists in the matching tuple dictionary, assign the related parent name to the sentence.
      3. If multiple actions and multiple results are found, assign each action pairs with the nearest succeeding result. The first matched parent name is considered major parent name.

4.2 Output matching results to Excel Spreadsheet

**Stage 2: Stats Extraction and validation (Loop 2 algorithm):**

1. **Get all necessary inputs:**
   1. Active Roster
   2. Stage 1 results
   3. Data Models
      1. Parent name – role model: What offense/defense roles are expected for a parent name. e.g. RushSimple expects a rusher and a tackler, maybe more than one assist tacklers
      2. Role – stats model: What stats are expected for a role. E.g. for a rusher, yards, gain/loss, td etc are expected in the sentence, but it is not necessary that all expected stats are found.
      3. Stats – template model: What templates/patterns (regex) to extract for each stats. E.g. for td stats, templates to look for include ‘td’, ‘touchdown’, ‘run to goal line’, etc.
      4. Parent name – validation rule model: For a parent name, how to calculate the ending context using the beginning context and extracted stats.
2. Parse **Context**:

Extract information as **possession-down-ytg-side-ballposition** from the context.

Raise an alert “context structure error” if context doesn’t contain all the parts.

1. Find **Player Names** from sentence:
   1. A nonname stopword collection is formed using the regex from action/result model and common non-name words such as prepositions, the/an, numbers, etc. Remove these stopwords from the sentence.
   2. The remaining words are tokens that are possibly names. For each token, try to find a matched player name from the roster. If a good match can be found, the token is considered as a name. Else, if the first character of the token in the original sentence is in uppercase, than the token is considered a name.
   3. Assign roles to names found

Assumption: the sentence is always written in defender, assistant defender, offender order. Based on the parent name – model, assign the respective offending/defensing roles to the names found in the assumed order.

* 1. Match names to players in Active Roster

Use player name matcher (script provided by Taheer) to replace the name in the sentence by the name in the roster.

1. **Extract stats:**
   1. Now for each player found in the sentence, we have estimated the player’s role, team, and we also know what stats are expected for his role.
   2. According to stats-template model, search the sentence for the template of each stats. In addition, some stats such as ‘complete/incomplete’, ‘touchback’, ‘touchdown’ are directly indicated by the parent name.
2. **Context Validation:** 
   1. Use beginning context, extracted stats, and Parent name – validation rule model to calculate the ending context for scrim.
      1. Special treatment for Punt Return and Kickoff Return. The net change of yards is the result of Punt/Kickoff yards – Return yards.
   2. Take the beginning context of next scrim sentence as the supervision of the current sentence’s ending context. Compare the supervising ending context with the calculated ending context.
      1. If the same, the stats extraction is correct.
      2. If different, calculate the difference in ball spot between supervising ending context and current beginning context. If the difference can be found somewhere in the sentence, correct the extracted stats, mostly the yds, gain, loss, complete, incomplete stats.
      3. After correction, re-calculate the ending context. If still different from supervising ending context, raise alert “Reverse validation still have problem”. If agrees, put remark “reverse validation successful”.
      4. Special treatment for Punt Return and Kickoff Return. In total there are 4 values involved: start position, Punt/Kickoff yards, return yards, and ending position. If 3 are known, the last can be calculated. If less values are known, raise alert “no sufficient data for play with return”.
3. **Score Validation:**

Track the scores using extracted scoring stats, e.g td, fg good, pat. When processing a sentence with Parent Name ‘Score’, use the recorded score as supervising score. Calculate the difference between supervising score and the tracking score to make a guess on what scoring stats is missing in extraction.

* 1. If difference is 6, the possession team of last play had a unextracted td.
  2. If difference is 7, the possession team of last play had a unextracted td + PATKick.
  3. If difference is 8 and there is Rush/Pass regex in the last play, the possession team of last play had a unextracted td + PATRush/Pass.
  4. If difference is 3, the possession team of last play had a unextracted FGGood.

1. **Stats aggregation:**

For each player, sum up all his stats. For each team, sum up the stats of all players.

Sort aggregated stats into offense stats, defense stats, and special teams’ stats.

1. **Existing XML validation:**

If the game has manually created XML available, parse the XML for player and team aggregated stats. The XML stats are printed next to extracted stats as reference in the BA template.

1. Output extracted stats according to **BA review template**:

The template contains 4 tabs: play-by-play stats, offense, defense, and special teams.

* 1. Play-by-play stats: extracted stats for each sentence, each player, each stat. BA reviews the extraction quality and
     1. Correct the mistakes
     2. Advise on algorithm update
     3. Propose new templates/patterns to be added to action/results regex.
  2. Offense, defense, and special teams.

**Stage 3: XML generation with all Stats.**

**Part 2: Code organization and descriptions:**

**A screenshot of a cell phone

Description automatically generated**

1. Data Models (folder)

The folder constitutes all the data model dictionaries used in the play-by-play stats extraction process as json files and name\_stopwords txt file consisting of an extensive list of football name stopwords.

1. player\_matching (folder)

The codes files within this folder provides the player name matching and team rosters related information used while extracting the stats.

1. Game\_stat\_extractor.py

The main driver file for instantiating the Play-by-Play Stats extraction modules for Step 1 and Step 2.

1. KickPuntAnalyser.py

Categorizes the play sentences into Kick or Punt.

1. PenaltyParser.py

All penalty rules and codes dictionaries useful for understanding the effects of Penalty in the game.

1. PlayerRoleMapper.py

Contains Action and Results dictionaries and action-role dictionaries useful in mapping the player roles.

1. step1\_data\_model.py

All data dictionaries used in Step 1 of the stats extraction process, mainly to create the json files stored in the Data Models folder.

1. step1\_objects.py

The main classes used in Step 1, OneGame: Game objects for loop 1 and OneLine: Line objects for loop 1. (loop 1 indicates Step1)

1. step2\_data\_model.py

Make use of all the data models from Data Models folder for extracting stats in Step 2.

1. step2\_objects.py

The main classes used in Step 2, TwoGame: Game Object for Loop 2 and TwoLine: Line Object for Loop 2. (Loop 2 indicates Step2)

1. xml\_stat\_validator.py

Generates the aggregated stats for teams and players and also performs the XML stats validation.

**Part 3: Input and Output files for Stats Extraction.**

**Input:**

1. Processed pfd’s of the Game by Jaisys, converted to txt format.
2. Rosters, in Athlyte DB

**Output:**

1. Aggregated stats of players and roles in XML form

**Part 4: Presentation on the process**

* Refer to the presentation in **Play-by-Play Stats\_Extraction\_Walk\_through.pptx**

**TODO List for the Stats Extraction process:**

1. Developing a **BA Utility**, to enable the Business Analyst to automatically perform validations on the play-by-play extracted stats. The output of the Stats extracted as the XML’s can be directly shared to the BA to review.

The Aim of the process is to perform 80% of the stats extraction effectively and get the remaining complex and ambiguous play sentences reviewed by the BA.

1. Automate the extraction and pre-processing step currently being handled by Jaisys, following the requirements mentioned in the documentation. Avoid dependency of Jaisys in the long run.
2. Utilize and Manage the Alerts in the Stats extraction process for Context validation, Player Name validation, Penalty validation and other.

Note: The Stats Extraction works well for short play sentences, but not for passages of text.

**Recommendations:**

1. Make use of Named Entity Recognition (NER) to identify the player names in the play-by-play sentences to reduce extensive list of name\_stopwords, and standardize the process throughout all other games.(Most of the NER techniques were already tried, still have the scope of trying out other alternatives).

Note: The player names have to extracted with a greater precision to get the whole stats extraction process correct.

1. Standardize the context part of the play-by-play sentences in Jaisys, to avoid multiple Regex Templates and ease the automation process as well. (Possibly use two-character team short names to facilitate future requirements).
2. Define player roles in a more standardized way, obtain all the player stats useful in real-time use cases.
3. Possibly create reusable dictionaries for the commonly used play categories, roles, etc.