# Blog Post 3: Improving Data Collection with Match Analysis

## Objective

This week, we focused on expanding our data scraping process to gather more in-depth information about each match. We enriched our data to include not only basic match information, but also contextual data such as team lineups, player performances, and weather or referee statistics.

## Key Accomplishments

This week's work focused on increasing the analytics potential of the data by making it more granular. Here are the key milestones:

## 1. Season and Week Navigation

- Automated the process of collecting data for each week by switching between seasons.

- Designed a loop to cover each current season, ensuring no data was missing.

```python
 def navigate_and_scrape_seasons(driver):
 button_xpath = '//*[@id="__next"]/main/div/div[3]/div/div[1]/div[1]/div[1]/div[1]/div[2]/div[2]/div[1]/div/div[2]/div/div/div/div/button'
 button = WebDriverWait(driver, 10).until(
 EC.element_to_be_clickable((By.XPATH, button_xpath))
 )
 button.click()
 for season in range(5):
 season_xpath = f'({button_xpath})[{season + 1}]'
 WebDriverWait(driver, 10).until(
EC.element_to_be_clickable((By.XPATH, season_xpath))
).click()
```

` ` `

## 2. Detailed Match Information:

- Detailed data was collected as follows:
- Match date, home and away teams, and scores.
- Team lineups and performance ratings.
- Players' names and individual performance ratings.
- We kept the dataset consistent by skipping postponed matches.

## 3. Enhanced Interactivity

- We automated the transition to dedicated tabs where performance data appears.
- We used JavaScript-based solutions to extract performance data within pseudo-elements.

```python
def get_performance(driver, xpath):
script = (
"return window.getComputedStyle("
"document.evaluate(arguments[0], document, null, XPathResult.FIRST_ORDERED_NODE_TYPE, null).singleNodeValue, "
"'::after').getPropertyValue('content');"
)
performance = driver.execute_script(script, xpath).strip('"')
return performance
```

## 4. Creating Dynamic CSV Files

- We created CSV files dynamically to store data in an organized manner.
- We increased the accessibility of data by organizing the folder structure and defining file naming standards.

```python
def create_or_append_csv(base_path, file_name, headers, data):
file_path = os.path.join(base_path, file_name)
os.makedirs(base_path, exist_ok=True)
with open(file_path, mode='a', newline='', encoding='utf-8') as file:
writer = csv.writer(file)
if not os.path.isfile(file_path):
writer.writerow(headers)
writer.writerow(data)
```

## 5. Error Management and Robust Structure

- We added extensive error handling mechanisms to handle unexpected situations like missing elements, timeouts, or changes to site structure.

- We created a fully automated pipeline, minimizing manual intervention.

## Challenges Encountered

- Extracting Data from Pseudo-Elements: Extracting performance metrics from pseudo-elements required custom JavaScript integrations.

- Data Accuracy: Ensuring data accuracy across multiple pages and seasons has been a time-consuming process.

## Takes Away

This week allowed us to explore in depth how a more detailed data scraping process can be structured. This rich dataset is the foundation of our analytics models and will allow us to make more accurate predictions. This depth of data gained will be a huge advantage when building predictive models in the following stages.

## Next Steps

The next stage is to pre-process the collected data, handle missing values, standardize formats, and do feature engineering. With a strong data foundation, we will begin exploring machine learning models to understand the factors that affect football match results.