



POLISH-JAPANESE ACADEMY OF INFORMATION TECHNOLOGY

Wydział Informatyki

Katedra metod programowania

Inżynieria oprogramowania, procesów biznesowych i baz danych (AM)

Marcin Wałachowski

s19541

Proces tworzenia aplikacji bukmacherskiej

Praca magisterska

Krzysztof Barteczko

Warszawa, Czerwiec, 2023r.

Streszczenie

W ramach mojej pracy dyplomowej utworzyłem prostą wersję aplikacji służącej do obstawiania przez użytkownika zakładów bukmacherskich. Ze względu na charakter naukowy pracy skupiłem się głównie na procesie tworzenia oprogramowania.

Z tego powodu wewnątrz mojej pracy zawarłem wiele porównań różnych potencjalnych technologii za pomocą których mogłem zrealizować poszczególne fragmenty projektu.

Z tego względu utworzone oprogramowanie jest dość uproszczone, przez co wymaga dodania kilka funkcjonalności przed wykorzystaniem go w celach biznesowych.

Pisemną część pracy podzieliłem na cztery główne rozdziały, z których każdy odpowiada jednemu z kroków tworzenia tego typu aplikacji.

W pierwszym rozdziale skupiłem się na procesie pozyskiwania niezbędnych danych dla mojej aplikacji, którymi oczywiście były informacje o meczach piłkarskich. W tej części pracy porównałem dwie popularne technologie Selenium oraz BeautifulSoup, między innymi za pomocą testów wydajnościowych.

W kolejnej części mojej pracy za pomocą uczenia maszynowego stworzyłem algorytm, którego celem jest wyliczenie kursów, na poszczególne rezultaty nadchodzących wydarzeń sportowych. Wewnątrz tego rozdziału zawarłem również część przygotowującą dane do nauki oraz przeprowadziłem analizę, dzięki której mogłem udoskonalić mój algorytm tak aby zwracane przez niego rezultaty były dla mnie zadowalające.

Trzeci rozdział pracy dotyczył natomiast części bazodanowej czyli sposobu przechowywania danych. Wewnątrz tego rozdziału porównałem bazy danych opierające się na języku SQL oraz technologie od niego odchodzące. W dalszej części przedstawiłem sposób w jaki utworzyłem bazę danych w wybranej technologii oraz opisałem komunikację pomiędzy nią a aplikacją.

Ostatni rozdział mojej pracy naukowej zawiera prezentację aplikacji mobilnej utworzonej w ramach mojego projektu. Poza opisem zawartych funkcjonalności, umieściłem w tym kilka informacji dotyczących implementacji wizualnej części projektu oraz krótko opisałem dlaczego zdecydowałem się na skorzystanie z Android Studio.

Słowa kluczowe:

Python, Aplikacja mobilna, Nauczanie maszynowe,
Zakłady bukmacherskie, Web Scraping

WSTĘP	3
Zakłady bukmacherskie	3
Cel projektu.....	3
1. POZYSKIWANIE DANYCH	3
1.1 Wybór technologii	3
1.2 Implementacja	6
2. WYLICZANIE KURSÓW	15
2.1 Przygotowanie danych do uczenia	16
2.2 Analiza danych	19
2.3 Uczenie maszynowe	37
3. BAZA DANYCH.....	56
3.1 Wybór technologii	56
3.2 Zapisanie nadchodzących meczów w bazie danych	58
3.3 Komunikacja aplikacji mobilnej z bazą danych	62
3.4 Rozliczanie zakładów	66
4. APLIKACJA MOBILNA	68
4.1 Wybór technologii	69
4.2 System Autentykacji.....	69
4.3 Widoki zalogowanego użytkownika	74
PODSUMOWANIE	81
BIBLIOGRAFIA	82

Wstęp

Zakłady bukmacherskie

W ramach mojego projektu utworzyłem aplikację, której funkcją jest danie możliwości obstawiania przez użytkownika zakładów bukmacherskich. Ten termin oznacza rodzaj zakładu, w którym osoba biorąca w nim udział stara się przewidzieć wynik spotkania sportowego. W praktyce użytkownik decyduje się obstawić wybraną sumę pieniędzy na wybrany rezultat meczu, taki jak zwycięstwo drużyny A. (Wikipedia, 2019)

Oczywiście więksi bukmacherzy w swojej ofercie pozwalają na obstawianie nie tylko, która z drużyn zwycięży w spotkaniu, lecz w przypadku mojego ze względu na uproszczenie problemu ograniczę się tylko do tego typu wydarzeń.

Cel projektu

Głównym celem projektu jest przeanalizowanie procesu tworzenia aplikacji bukmacherskiej. Podczas wspomnianej analizy rozważę kilka technologii, które można brać pod uwagę w kontekście poszczególnych części pisania tego typu oprogramowania.

Kolejnym celem, który wyznaczyłem dla mojego projektu jest utworzenie algorytmu korzystającego ze sztucznej inteligencji, który na podstawie pozyskanych danych będzie wyznaczał kursy na poszczególne wydarzenia w widowisku sportowym.

Ze względu na naukowy charakter pracy celem projektu nie jest utworzenie w pełni funkcjonalnej aplikacji, przez co napisana przeze mnie aplikacja będzie uproszczona i będzie służyła do wizualizacji.

1. Pozyskiwanie danych

1.1 Wybór technologii

Istnieje kilka sposobów do pozyskiwania danych potrzebnych do działania tego typu systemu. W mojej pracy inżynierskiej, w ramach której stworzyłem system pozwalający na wyszukiwanie wyników meczów e-sportowych do pozyskiwania danych użyłem zewnętrznego api o nazwie PandaScore. To rozwiązanie niestety okazało się nie najlepsze, ponieważ nakładało ono sporo ograniczeń na moją aplikację. Główną wadą tego typu rozwiązań jest to, że abonamenty do tego typu serwisów są dość drogie, a ich darmowe wersje są mocno okrojone. W wykorzystanym api w mojej poprzedniej pracy dyplomowej, miałem ograniczoną ilość zapytań na godzinę, przez co musiałem specjalnie rozplanować cykl aktualizowania danych w api, przez co nie możliwym było aktualizowanie wyników na żywo.

Drugim problemem tego typu rozwiązania było to, że darmowa wersja PandaScore nie dostarczała wszystkich potrzebnych danych dla mojej aplikacji, przez co w celach prezentacji część danych musiałem wygenerować losowo.

Z powodu problemów wygenerowanych przez poprzednie rozwiązanie w tym projekcie postanowiłem dostarczać dane za pomocą Web Scrapingu.

Web Scraping

Web Scraping to technika służąca do pozyskiwania danych, znajdujących się wewnątrz danej strony internetowej. Jest to obecnie dość popularna i przydatna technologia, a za jej pomocą można między innymi porównywać ceny produktów z różnych sklepów internetowych, śledzić działania konkurencji lub pobierać wyniki meczów, czyli dokładnie to czego potrzebuje w systemie tworzonym w ramach tej pracy dyplomowej.

Web Scraping polega na napisaniu programu, który pobiera kod źródłowy wybranej strony internetowej, z którego możemy pozyskać potrzebne dane, które znajdują się w odpowiednich tagach HTML. (Mamczur, 2020)

Mimo, że skrobanie sieci to dosyć popularna, oraz legalna technologia, to nie możemy jej stosować na każdej stronie i musimy zastosować się do kilku zasad.

Jedną z nich jest sprawdzenie pliku robots.txt, znajdującego się praktycznie na każdej stronie internetowej. Jego celem jest informowanie robotów, o czynnościach zabronionych na danej witrynie. Dotyczy to głównie programów działających automatycznie, które łączą się ze stroną, na której znajduje się ten plik. Głównymi powodami tworzenia takich plików są ograniczenie obciążenia strony oraz brak zezwolenia właściciela strony na pobieranie wybranych elementów. (Wikipedia, 2022)

Robots.txt

Sprawdzenie legalności pozyskiwania danych w wyżej opisanym pliku robots.txt, polega na sprawdzeniu dla których elementów witryny Web Scraping jest zabroniony.

Do pozyskiwania danych do mojego systemu użyłem witryny o nazwie fbref.com, która zawiera wyniki oraz rozbudowane statystyki dla największych lig piłki nożnej.

```

User-agent:*
# Disallow: /cbb/
# Disallow: /cfb/
# Disallow: /olympics/

# Disallow: /awards/
# Disallow: /blog/
# Disallow: /boxscores/
# Disallow: /coaches/
# Disallow: /draft/
# Disallow: /executives/
# Disallow: /friv/
# Disallow: /hof/
# Disallow: /leaders/
# Disallow: /play-index/
# Disallow: /players/
# Disallow: /route.cgi
# Disallow: /schools/
# Disallow: /search/
# Disallow: /stadiums/
# Disallow: /static/
# Disallow: /teams/
# Disallow: /years/

Disallow: /feedback/
Disallow: /linker/
Disallow: /my/

Disallow: /news/
Disallow: /en/news/
Disallow: /pt/news/
Disallow: /de/news/
Disallow: /fr/news/
Disallow: /es/news/
Disallow: /it/news/
Disallow: /news/

Disallow: /req/
Disallow: /short/
Disallow: /nocdn/

User-agent: AhrefsBot
Disallow: /

User-agent: AhrefsBot/5.0
Disallow: /

## sitemaps generated by copyit/sitemaps/build_sitemaps.pl
##
Sitemap: https://fbref.com/sitemaps/sitemap.xml

```

1 Zawartość pliku robots.txt dla strony fbref.com

Na powyższym rzucie ekranu przedstawiony jest tekst zawarty wewnątrz pliku robots.txt dla witryny, z której zdecydowałem się pobierać dane. Składa się z on z rekordów, które posiadają pola User-agent oraz Disallow. Gdzie Disallow oznaczają jakich fragmentów

strony nie można „skrobać”, a User-agent pokazuje jakich programów dotyczą ograniczenia znajdujące się w tej samej sekcji. Z powyższego pliku wynika, że dla programów AhrefsBot i AhrefsBot/5.0, Web Scraping jest całkowicie zabroniony, a pozostałe roboty mogą pobierać dane znajdujące się na innych podstronach niż tych zawartych w pierwszej sekcji.

Do pozyskiwania danych w moim programie użyłem podstron /squads/ oraz /comps/.

Jak wynika z powyższego zrzutu ekranu, te podstrony nie znajdują się w polach Disallow wewnątrz pliku robots.txt, co oznacza, że mogą legalnie pozyskiwać dane potrzebne do działania mojego systemu z tej strony internetowej.

Innymi warunkami pobierania danych ze strony za pomocą technologii są między innymi nie naruszanie praw autorskich strony, czyli nie pobieranie autorskich treści, takich jak całe artykuły, oraz unikanie mocnego obciążania witryny. (Urban, 2022)

Te warunki również zostały spełnione przez mój system.

1.2 Implementacja

Mechanizm pozyskiwania danych zdecydowałem się zaimplementować za pomocą języka Python. Zdecydowałem się na ten język, ponieważ posiada on wiele bibliotek do tego przeznaczonych oraz z powodu, że za jego pomocą będę również starał się wyliczyć kursy meczów za pomocą sztucznej inteligencji w dalszej części projektu.

Jak wcześniej wspomniałem istnieje kilka bibliotek, lecz ja wybrałem dwie z nich i w dalszej części pracy, postaram się je porównać i wybrać, która z nich zostanie wykorzystana w końcowej części systemu.

Selenium

Pierwsza z rozważanych bibliotek to Selenium. Jest to dosyć popularne narzędzie, do Web Scrapingu strony internetowej, wykorzystujące przeglądarkę do pozyskania pożądaných danych. Działanie tego narzędzia symuluje zachowanie użytkownika na stronie i podczas swojej pracy wykonuje ono typowe czynności wykonywane, przez człowieka na stronie takie jak zaakceptowanie ciasteczek, oraz przejście do kolejnych podstron za pomocą naciskania odpowiednich przycisków. Ta właściwość czyni tę bibliotekę dość intuicyjną, dzięki czemu jej implementacja jest dość prosta. (Kwapisz, 2020)

Implementacja

W implementacji to narzędzie do poprawnego działania nie potrzebuje importowania innych bibliotek, ale wymaga dodania do katalogu z projektem specjalnego sterownika.

Działanie tej biblioteki postanowiłem przetestować, poprzez pobieranie wyników meczy oraz statystyk minionych spotkań z obecnego sezonu Premier League ze strony fbref.com. Pierwszą rzeczą jaką wykonywał skrypt było pobranie adresów do statystyk wszystkich drużyn z tabeli ligowej.

```

from selenium import webdriver
from selenium.webdriver.common.by import By
import pandas as pd

premier_league_url = "https://fbref.com/en/comps/9/Premier-League-Stats"

driver = webdriver.Chrome()
driver.get(premier_league_url)

cookies_accept = driver.find_element(
    by=By.XPATH, value = '//*[@id="qc-cmp2-ui"]/div[2]/div/button[3]'
)
cookies_accept.click()

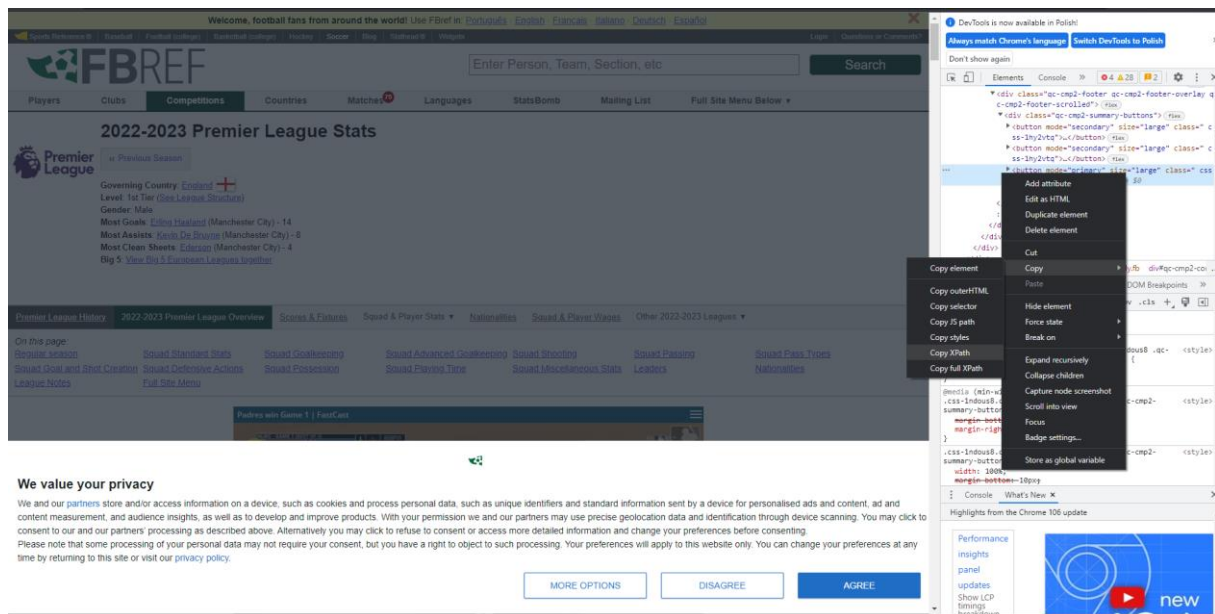
table = driver.find_element(
    by=By.XPATH, value='//*[@id="results2022-202391_overall"]'
)
teams = table.find_elements(by=By.TAG_NAME, value='tr')[1:]

team_urls = [
    team.find_element(by=By.CSS_SELECTOR, value = 'td[data-stat="team"]')
    .find_element(by=By.TAG_NAME, value='a')
    .get_attribute('href')
    for team in teams
]

```

2 Kod odpowiedzialny za pobieranie linków do statystyk wszystkich drużyn

Powyższy kod pokazuje przykładowe działanie biblioteki Selenium. Na początku załadowywany jest sterownik, znajdujący się w wcześniej utworzonym pliku. Następnie za pomocą metody `get()` wywołanej na naszym sterowniku, ładujemy naszą stronę, z której będą pobierane dane. Po załadowaniu strony za pomocą metody `find_element()`, wyszukujemy obiekt klasy `WebElement` odpowiedzialny za działanie przycisku akceptującego ciasteczka na stronie. Znajdujemy go za pomocą jego XPath, który możemy uzyskać przy pomocy trybu dewelopera na stronie internetowej, pokazanym na poniższym zrzucie ekranu.



3 Skopiowane XPath przycisku

Następnie za pomocą metody `click()`, wywołanej na poprzednio znalezionym obiekcie klikamy na przycisk. W następnej kolejności w analogiczny sposób, pobieram obiekt tabeli zawierającej statystyki, oraz wywołuję na nim metodę `find_elements()`, za pomocą której pobieram wszystkie elementy typu `<tr>`, które odpowiadają za rzędy tabeli. Na końcu pobieram adresy url do statystyk drużyn, poprzez pobranie atrybutów typu `href` z rekordów zawierających informacje o drużynach, znajdujących się we wcześniej pobranych wierszach. Pobrane adresy zapisuje w liście, z którą następnie będę iterował w celu pozyskania danych dla każdej z drużyn.

```

scores_df, upcoming_matches_df = pd.DataFrame(), pd.DataFrame()
for url in team_urls:
    team_scores_df, team_upcoming_matches_df = get_matches_of_team(url)
    scores_df = pd.concat([scores_df, team_scores_df])
    upcoming_matches_df = pd.concat(
        [upcoming_matches_df, team_upcoming_matches_df], ignore_index=True
    )

```

4 Kod pobierający statystyki każdej z drużyn na podstawie ich url

Jak pokazano na powyższym fragmencie kodu, w następnej kolejności za pomocą utworzonej wcześniej listy, pobrałem wszystkie potrzebne dane i umieściłem je w DataFrame, czyli strukturze danych dostarczonej przez bibliotekę Pandas, która jest odpowiednikiem tabeli.

Kod zawierający pobieranie danych zawarłem w metodzie `get_matches_of_team`, która na początku za pomocą odpowiednich przycisków oraz fragmentów html

dostaje się do odpowiednich tabel i analogicznie jak poprzednio opisana metoda, pobiera potrzebne dane wrzucając je do odpowiedniego DataFrame.

```
def get_matches_of_team(url):
    driver = webdriver.Chrome()
    driver.get(url)

    cookies_accept = driver.find_element(
        by=By.XPATH, value = '//*[@id="qc-cmp2-ui"]/div[2]/div/button[3]'
    )
    cookies_accept.click()

    shooting_href = driver.find_element(
        by=By.XPATH, value = '//*[@id="content"]/div[6]/div[2]/a'
    ).get_attribute('href')
    driver.get(shooting_href)

    table = driver.find_element(by=By.CSS_SELECTOR, value='table')
    team_scores = table.find_elements(by=By.CSS_SELECTOR, value='tr')[2:-1]

    team_scores_df = pd.DataFrame()

    for team_score in team_scores:
        row = get_result_of_match(team_score)
        team_scores_df = pd.concat([team_scores_df, row])

    element = driver.find_element(
        by=By.XPATH, value='//*[@id="all_matchlogs"]/div[3]/div[2]/a'
    )
    driver.execute_script("arguments[0].click();", element)

    table = driver.find_element(
        by=By.CSS_SELECTOR,
        value='div[class="table_container is_setup current"]'
    )
    opponent_scores = table.find_elements(
        by=By.CSS_SELECTOR, value='tr'
    )[2:-1]

    opponent_scores_df = pd.DataFrame()
```

```

for opponent_score in opponent_scores:
    row = get_result_of_match(opponent_score)
    opponent_scores_df = pd.concat([opponent_scores_df, row])

opponent_scores_df.drop(
    ['Result', 'Goals_for', 'Goals_against', 'Venue'], axis=1, inplace=True
)
opponent_scores_df.rename(columns={
    "Opponent": "Team",
    "Shots": "Opponent_shots",
    "Shots_on_target": "Opponent_shots_on_target",
    "Average_shot_distance": "Opponent_average_shot_distance",
    "Expected_goals": "Opponent_expected_goals"
}, inplace=True)

full_scores_df = team_scores_df.merge(
    opponent_scores_df,
    on=['Date', 'Time', 'Round', 'Day_of_week', 'Competition'],
    how='left'
)

scores_href = driver.find_element(
    by=By.XPATH, value='//*[@id="content"]/div[3]/div[1]/a'
).get_attribute('href')
driver.get(scores_href)

table = driver.find_element(by=By.CSS_SELECTOR, value='table')
matches = table.find_elements(by=By.CSS_SELECTOR, value='tr')

upcoming_matches_df = pd.DataFrame()

for match in matches:
    row = get_upcoming_match(match)
    if row.empty:
        continue
    upcoming_matches_df = pd.concat([upcoming_matches_df, row])

upcoming_matches_df['Team'] = full_scores_df['Team'][0]

return full_scores_df, upcoming_matches_df

```

5 Kod metody `get_matches_of_team()`

Na potrzeby działania, powyższego kodu utworzyłem również metody `get_result_of_match()` i `get_upcoming_match()`, które wybierają odpowiednie dane z podanego w argumencie wiersza tabeli, oraz zwracają je w postaci `DataFrame`.

```
result = match.find_element(  
    by=By.CSS_SELECTOR, value = 'td[data-stat="result"]'  
).text
```

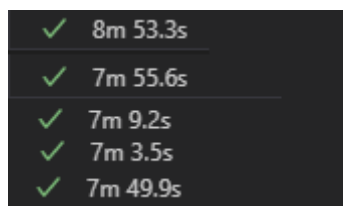
6 Fragment metody `get_result_of_match()`

Powyższy fragment kodu zawiera, fragment odpowiedzialny za pobieranie informacji o wyniku meczu znajdującej się w jednym z rekordów, podanego w argumencie wiersza (`match`). W tym przypadku pożądaną komórkę znajdujemy znajdując element zawierający atrybut „data-stat” o wartości „result”, a następnie pobieramy tekst, który zawiera będący szukaną przez nas wartością. Analogicznie pobierane są pozostałe dane.

Testy wydajnościowe

Działanie napisanego Web Scrapera przy pomocy narzędzia Selenium, przetestowałem za pomocą kompilatora Visual Studio Code, który jest jednym z bardziej popularnych kompilatorów języka Python. (Software Testing Help, 2022)

Podczas sprawdzania wydajności, uruchomiłem napisany skrypt pięć razy.



✓	8m 53.3s
✓	7m 55.6s
✓	7m 9.2s
✓	7m 3.5s
✓	7m 49.9s

7 Pomiary czasu kompilacji
scrapera używającego Selenium

Testowany skrypt, zwrócił wszystkie potrzebne dane, w oczekiwanym przeze mnie formacie, ale jego średni czas wykonania wynosił aż 7m 46.3s.

Beautiful Soup

Drugim narzędziem, które przetestowałem do celów swojego projektu, jest biblioteka Beautiful Soup. Została ona stworzona do ustrukturyzowanych danych HTML i XML, za pomocą selektorów CSS. W porównaniu do poprzedniej opcji nie ładuje ona całej strony internetowej, a pobiera jedynie jej kod źródłowy. Z tego powodu jest ona znacząco wydajniejsza, co potwierdzę w dalszej części tego podrozdziału. (Nair, 2014)

Implementacja

Implementacja tej biblioteki jest dość prosta i w porównaniu do Selenium, nie potrzebuje ona specjalnego sterownika. W tym przypadku jednak do jej poprawnego działania oprócz samego BeautifulSoup musimy zaimportować w projekcie również bibliotekę requests.

Działanie tej biblioteki tak samo jak dla poprzedniej przetestowałem pobierając statystyki drużyn grających w Premier League w sezonie 2022/23. Z tego powodu również musiałem zacząć od wydobycia adresów URL do stron zawierających dane meczów wszystkich drużyn biorących udział w rozgrywkach.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

premier_league_url = "https://fbref.com/en/comps/9/Premier-League-Stats"

data = requests.get(premier_league_url)

soup = BeautifulSoup(data.text)
standings_table = soup.select('table.stats_table')[0]
links = standings_table.find_all('a')
links = [l.get("href") for l in links]
links = [l for l in links if '/squads/' in l]

team_urls = [f"https://fbref.com{l}" for l in links]

upcoming_matches, match_stats = pd.DataFrame(), pd.DataFrame()

for url in team_urls:
    team_upcoming_matches, team_stats = get_matches_of_team(url)
    match_stats = pd.concat([match_stats, team_stats], ignore_index=True)
    upcoming_matches = pd.concat(
        [upcoming_matches, team_upcoming_matches], ignore_index=True
    )
```

8 Kod odpowiedzialny za pobieranie linków do statystyk drużyn

Jak widać na powyższym fragmencie kodu, działanie tego narzędzia jest dość podobne do poprzedniego, ale posiada jednak kilka kluczowych różnic. Pierwszą kluczową zmianą jest to, że BeautifulSoup nie wymaga, akceptacji ciasteczek, oraz innych akcji symulujących działanie użytkownika. Jest to spowodowane tym, że nie używa on przeglądarki podczas swojego działania.

Podobnie jak w przypadku Selenium, pierwszą rzeczą jaką zaimplementowałem było pobranie zawartości strony. Aby tego dokonać musiałem użyć wcześniej wspomnianej biblioteki requests. Następną akcją podobnie jak dla poprzedniego narzędzia było zapisanie zawartości pożądanej tabeli do obiektu. Kolejno zapisałem do listy wszystkie elementy HTML typu <a>, znajdujące się w wcześniej zapisanej tabeli, oraz wydzieliłem z nich wszystkie atrybuty typu href, czyli potrzebne w dalszej części adresy URL. Na końcu dla każdego linku wykonałem metodę get_matches_of_team, której działanie przedstawię poniżej.

```
def get_matches_of_team(url):
    data = requests.get(url)
    soup = BeautifulSoup(data.text)

    links = soup.find_all('a')
    links = [l.get("href") for l in links]
    links = [l for l in links if l and 'all_comps/shooting/' in l]

    data = requests.get(f"https://fbref.com{links[0]}")
    soup = BeautifulSoup(data.text)
    team_games_table = soup.select('table.stats_table')[0]

    team_stats = pd.read_html(str(team_games_table))[0]
    team_stats.columns = team_stats.columns.droplevel()
    team_stats.drop([
        'Gls', 'SoT%', 'G/Sh', 'G/SoT', 'FK', 'PK', 'PKatt', 'npxG', 'npxG/Sh',
        'G-xG', 'np:G-xG', 'Match Report'
    ], axis=1, inplace=True)

    team_stats.rename(columns={
        "Comp": "Competition", "Day": "Day_of_week", "GF": "Goals_for",
        "GA": "Goals_against", "Sh": "Shots", "SoT": "Shots_on_target",
        "Dist": "Average_shot_distance", "xG": "Expected_goals"
    }, inplace=True)
    team_stats.drop(team_stats.tail(1).index, inplace=True)
```

9 Fragment metody get_matches_of_team()

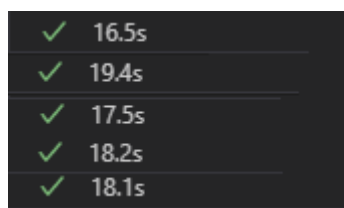
Na powyższym fragmencie kodu znajduje się fragment wcześniej wspomnianej klasy, który odpowiada za pobranie danych dotyczących statystyk danej drużyny w rozegranych meczach. Pierwsze linijki kodu odpowiadają za pobranie adresu do podstrony zawierającej pożądaną tabelę. Następnie podobnie jak w poprzednio zapisujemy daną tabelę do obiektu.

Kolejna rzecz jaką zrobiłem to zapisanie pobranej tabeli jako DataFrame za pomocą metody `read_html()`, dostarczonej przez bibliotekę Pandas. Jest to dosyć przydatna funkcjonalność, ponieważ nie wymaga ręcznego pobrania każdej wartości tabeli. Minusem tego rozwiązania jest to, że uzyskana struktura danych, nie zawsze pokrywa się z oczekiwaną, dlatego w dalszej części przerobiłem ją usuwając zbędny poziom zawierający nagłówki oraz usuwając zbędne kolumny. Na końcu zmieniłem ich nazwy na bardziej czytelne, a na końcu usunąłem ostatni wiersz, który zawierał podsumowania.

Pozostałymi tabelami, których wartości pobrałem, były nadchodzące mecze oraz statystyki przeciwników. Implementacja scrapingu tych tabel była praktycznie identyczna jak w przypadku poprzednio opisanej, więc nie umieściłem jej w pisemnej części pracy.

Testy wydajnościowe

Wydajność skryptu używającego BeautifulSoup, przetestowałem w takich samych warunkach co poprzedniego, również wykonując pięć pomiarów czasu wykonania.



✓	16.5s
✓	19.4s
✓	17.5s
✓	18.2s
✓	18.1s

10 Pomiarów czasu kompilacji scrapera wykorzystującego BeautifulSoup

Za pomocą tego skryptu również uzyskałem wszystkie potrzebne dane, lecz wykonywał się on znacząco szybciej od poprzednika i jego średni czas wynosił 17.74 sekund.

Końcowe porównanie

Jak wynika z poprzedniej części rozdziału, mimo że obie technologie w zasadzie są stworzone do tego samego posiadają jednak wiele różnic.

Elastyczność

W tej kategorii moim zdaniem wygrywa Selenium, ponieważ za jego pomocą można dość łatwo pozyskać dane z większości stron WWW. W przypadku BeautifulSoup niestety w przypadku gorzej zorganizowanych stron dostęp do danych jest znacznie gorszy.

Mimo tego pierwsze z narzędzi również ma swoje wady, gdyż w przypadku zmian na stronie istnieje większe prawdopodobieństwo, że to właśnie scraper używający tej biblioteki przestanie działać.

Łatwość użycia

Na podstawie mojego doświadczenia z tymi dwoma narzędziami uważam, że obie z nich są dość proste w użytku, natomiast według mnie praca z BeautifulSoup jest znacznie wygodniejsza, gdyż można szybko pozyskać pożądane dane i od razu mieć je w formie ramki danych. W przypadku Selenium pobieranie danych było dość uciążliwe,

gdyż każdy rekord w tabeli musiałem pozyskać oddzielnie. Mimo to wydaje mi się, że pierwsze z narzędzi jest łatwiejsze do zrozumienia, gdyż symuluje one działanie użytkownika, jednak kiedy znamy strukturę strony praca z Beutiful Soup jest znacznie wygodniejsza.

Wydajność

W tym przypadku, jak wynika z wcześniej przeprowadzonego testu biblioteka Beutiful Soup jest znacznie wydajniejsza i dzięki niej pozyskałem oczekiwany dane prawie 30 razy szybciej.

Podsumowanie

Obie biblioteki spełniają swoją funkcję, natomiast obie nadają się do innego rodzaju projektów. Pierwsza z nich idealnie sprawdza się do pozyskiwania danych z bardziej złożonych stron. Strona z której pozyskuje dane jednak posiada dość dobrze zorganizowaną strukturę i jest dość prosta, więc z powodu dużej różnicy w wydajności w moim projekcie znacznie lepiej sprawdzi się biblioteka Beautiful Soup.

2. Wyliczanie kursów

Kolejnym krokiem w projekcie będzie wyliczenie kursów nadchodzących meczów. Są one kluczową rzeczą w tego typu aplikacji, ponieważ wyznaczają one potencjalną wygraną po postawieniu na dane zdarzenie.

Czym są kursy?

Przykładowo na mecz drużyn A i B, kurs na zwycięstwo drużyny A może wynosić na przykład 2.5, oznacza to że w przypadku zwycięstwa tej drużyny, osoba obstawiająca jako nagrodę otrzyma iloczyn tego kursu oraz swojego wkładu, więc jeśli obstawi 10 zł to będzie miała szansę otrzymania 25 zł. Jest to oczywiście najprostszy przykład zakładu gdzie, gdzie istnieją trzy możliwe rezultaty meczu (wygrana, remis oraz przegrana). Poza tego typu zakładami, istnieje wiele innych takich jak zakłady podwójna szansa (np. zwycięstwo lub remis) oraz zakłady dotyczące wydarzeń w meczu (np. dotyczące ilości strzałów). Pomimo tego, że praktycznie wszystkie większe strony bukmacherskie oferują wiele różnorodnych zakładów, w celu uproszczenia projektu skupię się tylko na zakładach typu 1X2, których przykład podałem powyżej (mecz pomiędzy drużynami A i B). (Stachak, 2022)



11 przykładowy zrzut ekranu zawierający kursy na mecz Legii Warszawa z Wisłą Płock z aplikacji Betlic

Jak bukmacherzy wyznaczają kursy?

Najważniejszą rzeczą w wyznaczeniu kursu meczu jest obliczenie prawdopodobieństwa danego zdarzenia. Kurs jest po prostu wynikiem ilorazu 100% przez prawdopodobieństwo.

$$kurs = \frac{100\%}{prawdopodobieństwo}$$

Z tego powodu, tak naprawdę jeśli znamy prawdopodobieństwo danego zdarzenia, wyznaczenie kursu jest bardzo proste. Niestety wyznaczenie procentowej szansy nie jest aż tak proste. Oczywiście nie jest możliwe dokładne wyznaczenie tego prawdopodobieństwa, ponieważ wchodzi w nie zbyt wiele czynników, a dodatkowo w sportach biorą udział ludzie, którzy nie są idealnie, więc na rezultat wydarzenia sportowego mogą mieć wpływ rzeczy takie jak forma i nastrój zawodnika. Z tego powodu kursy na różnych stronach bukmacherskich nie są takie same i zależą one od pracy wyspecjalizowanych analityków, posiadających doświadczenie w danej dziedzinie. Naturalnie prawdopodobieństwo nie jest wyznaczane tylko na podstawie subiektywnej opinii pracowników, ale posiadają oni specjalne programy zawierające szczegółową analizę dotyczącą rozgrywek danej dyscypliny. (Redakcja Goal.pl, 2022)

W ramach mojego projektu planuję natomiast stworzyć automatyczny algorytm wyliczający kursy na dane spotkania, działający na podstawie danych, które pozyskałem w poprzednim rozdziale mojej pracy dyplomowej.

W tym rozdziale postaram się opisać działanie napisanego przeze mnie algorytmu, porównując jego wyniki z kursami najpopularniejszych bukmacherów.

2.1 Przygotowanie danych do uczenia

Pozyskane w poprzednim rozdziale dane nie są w stu procentach gotowe do wykorzystania przez mój program, dlatego w tym podrozdziale zajmę się ich modyfikacją.

Na tym etapie posiadam DataFrame dotyczący rezultatów zakończonych meczów, zawierający następujące kolumny:

- Date – Zawiera datę meczu w formacie YYYY-MM-DD
- Time – Zawiera godzinę rozpoczęcia meczu
- Competition – Definiuje nazwę rozgrywek, w ramach których odbył się mecz.
- Round – Zawiera nazwę rundy, w ramach której odbył się mecz
- Day_of_week – Zawiera dzień tygodnia, w którym odbył się mecz
- Venue – Definiuje, czy drużyna jest gospodarzem meczu czy gra na wyjeździe.
- Result – Opisuje rezultat meczu, możliwe wartości tej kolumny to
 - W – zwycięstwo
 - D – remis
 - L – porażka

- Team – Zawiera nazwę drużyny
- Opponent – Zawiera nazwę przeciwnika
- Goals_for – Zawiera liczbę bramek strzeloną przez drużynę
- Goals_against – Zawiera liczbę bramek, które drużyna straciła
- Shots – Zawiera łączną liczbę strzałów drużyny
- Opponent_Shots – Zawiera łączną liczbę strzałów przeciwnika
- Shots_on_target – Zawiera liczbę strzałów celnych drużyny
- Opponent_Shots_on_target – Zawiera liczbę strzałów celnych przeciwnika
- Average_shot_distance – Zawiera średni dystans strzału drużyny
- Opponent_Average_shot_distance – Zawiera średni dystans strzału przeciwnika
- Expected_goals – Zawiera liczbę oczekiwanych goli drużyny, wyznaczoną przez stronę z której pobierałem dane
- Opponent_Expected_goals – Zawiera liczbę oczekiwanych goli przeciwnika, wyznaczoną przez stronę z której pobierałem dane

Usunięcie rozgrywek z poza Premier League

W tym projekcie skupiłem się tylko na rozgrywkach ligi angielskiej, a jak wiadomo większość drużyn z tej ligi bierze również udział w innych rozgrywkach takich jak liga mistrzów. Z tego powodu pozyskane dane zawierają również informacje o meczach rozegranych w ramach innych turniejów.

```
match_stats.Competition.unique()
✓ 0.0s
array(['Premier League', 'Europa Lg', 'EFL Cup', 'FA Cup',
      'Community Shield', 'Champions Lg', 'Conf Lg'], dtype=object)
```

12 Wszystkie rozgrywki znajdujące się w pozyskanych danych

Aby rozwiązać ten problem, po prostu usunąłem wszystkie wiersze, których wartość kolumny odpowiadającej za rozgrywki, była różna od Premier League.

```
match_stats.drop(match_stats[match_stats.Competition != 'Premier League'].index, inplace=True)
✓ 0.0s
```

13 Kod odpowiedzialny za usunięcie innych rozgrywek z DataFrame

Ujednolicenie nazw klubów

Kolejnym problemem na który trafiłem w pobranych danych, był problem dotyczący nazw drużyn piłkarskich. Polegał on na tym, że nazwy tej samej drużyny czasami były różne dla innych wierszy w DataFrame.

```
pd.concat([match_stats.Opponent, match_stats.Team]).unique()
✓ 0.0s
array(['Crystal Palace', 'Leicester City', 'Bournemouth', 'Fulham',
      'Aston Villa', 'Manchester Utd', 'Brentford', 'Tottenham',
      'Liverpool', 'Leeds United', 'Southampton', 'Forest', 'Chelsea',
      'Wolves', 'West Ham', 'Brighton', 'Newcastle Utd', 'Everton',
      'Manchester City', 'Arsenal', 'Newcastle United',
      'Manchester United', 'Tottenham Hotspur', 'Brighton & Hove Albion',
      'Wolverhampton Wanderers', 'West Ham United', 'Nottingham Forest'],
      dtype=object)
```

14 Wszystkie nazwy drużyn znajdujące się w pobranych danych

Jak widać na powyższym zrzucie ekranu część drużyn się powtarza. Przykładowo drużyna Manchester United występuje dwukrotnie raz jako своя pełna nazwa, a drugi skrótowo Manchester Utd. Ten problem rozwiązałem ujednolicając nazwy drużyn, poprzez zastąpienie części nazw innymi.

```
match_stats.Team.replace(r'Leeds United', r'Leeds', inplace=True)
match_stats.Team.replace(r'Tottenham Hotspur', r'Tottenham', inplace=True)
match_stats.Team.replace(r'West Ham United', r'West Ham', inplace=True)
match_stats.Team.replace(r'Manchester United', r'Man Utd', inplace=True)
match_stats.Team.replace(r'Manchester Utd', r'Man Utd', inplace=True)
match_stats.Team.replace(r'Manchester City', r'Man City', inplace=True)
match_stats.Team.replace(r'Newcastle Utd', r'Newcastle', inplace=True)
match_stats.Team.replace(r'Newcastle United', r'Newcastle', inplace=True)
match_stats.Team.replace(r'Wolves', r'Wolverhampton', inplace=True)
match_stats.Team.replace(r'Wolverhampton Wanderers', r'Wolverhampton', inplace=True)
match_stats.Team.replace(r'Leicester City', r'Leicester', inplace=True)
match_stats.Team.replace(r'Manchester City', r'Man City', inplace=True)
match_stats.Team.replace(r'Brighton & Hove Albion', r'Brighton', inplace=True)
match_stats.Team.replace(r'Forest', r'Nottingham Forest', inplace=True)
```

15 Kod odpowiedzialny za zmianę nazw drużyn

W kodzie przedstawionym powyżej poprawiłem nazwy drużyn, której dotyczy dany wiersz. Analogicznie zamieniłem również nazwy ich przeciwników, stosując te funkcje również dla wartości kolumny „Opponent”.

Stworzenie dodatkowych kolumn

Goals_diff

Pierwszą z dodatkowych kolumn, które przydadzą mi się w dalszej części tworzenia algorytmu, jest kolumna „Goals_diff”, która jest różnicą strzelonych oraz straconych goli.

Wartości tej kolumny będą przydatne, ponieważ są one liczbową reprezentacją rezultatu spotkania, oraz zawierającą również jego skalę.

```
match_stats['Goals_diff'] = match_stats['Goals_for'] - match_stats['Goals_against']
```

16 Kod odpowiedzialny za stworzenie kolumny "Goals_diff"

2.2 Analiza danych

Następnym krokiem podczas tworzenia algorytmu wyznaczającego kursy meczów była analiza pozyskanych danych. Dzięki temu procesowi mogłem wyznaczyć, które atrybuty miały największy wpływ na rezultaty, co miało wpływ na tworzenie modelu uczenia w dalszej części tego rozdziału.

Wpływ atrybutów tekstowych na wynik meczu

Pierwszą grupą wartości potencjalnie mających wpływ na rezultat meczu są kolumny zawierające tekst. Tego typu kolumny zazwyczaj nie mają zbyt wiele unikalnych wartości, przez co mogą się okazać przydatne w wyznaczaniu prawdopodobieństw zdarzeń.

Dodatkowo rozważane kolumny w tej sekcji nie będą dotyczyły przebiegu meczu, lecz informacji, które są znane przed jego rozpoczęciem. Z tego powodu dostępne są one również w DataFrame zawierającym informacje o nadchodzących spotkaniach, dla których kursy postaram się wyznaczyć.

Venue

Pierwszą z rozważanych kolumn tej kategorii jest kolumna „Venue” zawierająca informację o tym czy drużyna jest gospodarzem czy gra na wyjeździe.

Aby zbadać wpływ tego czynnika na rezultat spotkania użyłem modułu `express` zawartym w bibliotece `plotly`, który pozwala między innymi na przedstawienie danych w formie wykresu.

```
import plotly.express as px

fig = px.histogram(match_stats, "Result", facet_col="Venue",
                    color="Result",
                    title="Counts of results per venue",
                    height=500,
                    facet_col_wrap=2,
                    facet_col_spacing=0.1)

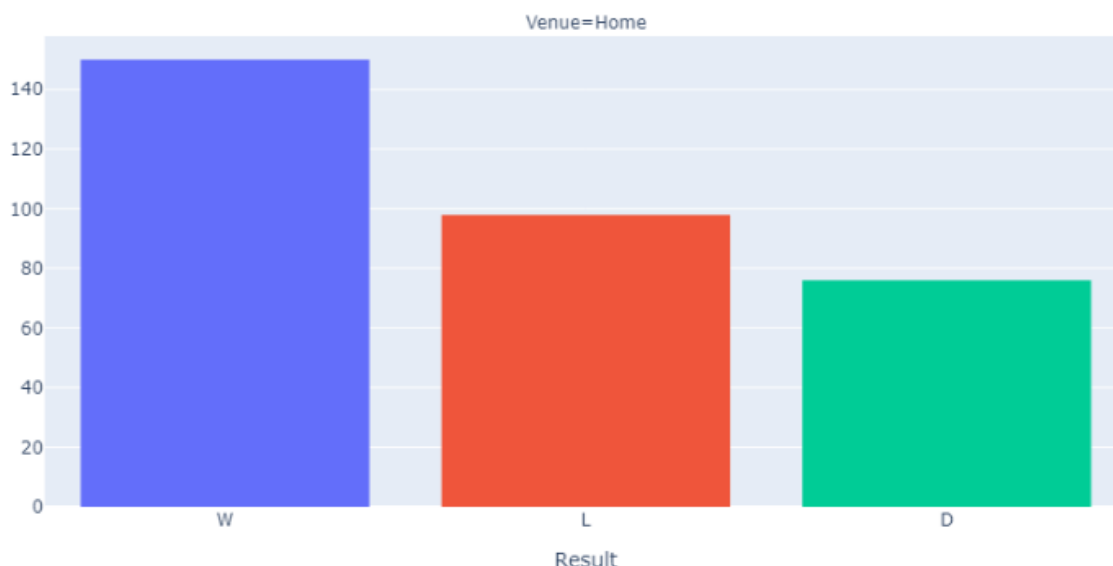
fig.update_layout(showlegend=False)
fig.update_yaxes(matches=None, showticklabels=True)
```

16 Kod odpowiedzialny za zestawienie wyników meczów w zależności od miejsca spotkania w formie wykresu

Po uruchomieniu powyższego skryptu otrzymałem następujące wykresy:



17 Wykres zawierający liczbę zwycięstw, remisów oraz porażek dla drużyn grających „na wyjeździe”



18 Wykres zawierający liczbę zwycięstw, remisów oraz porażek dla drużyn grających „u siebie”

Jak można się było spodziewać, miejsce spotkania ma dość znaczący wpływ na rezultat meczu. Na wykresach widać, że drużyny będące gospodarzem spotkania wygrywają więcej meczów niż zespoły przyjezdne. Spowodowane jest to tym, że gracze są bardziej przyzwyczajeni do gry na boisku, na którym grają oraz trenują regularnie, a dodatkowo innym czynnikiem faworyzującym lokalny zespół są kibice, których jest znacznie więcej po stronie zespołu grającego „u siebie”.

Day_of_week

Kolejną kolumną, której wartości mogą mieć wpływ na wynik spotkania jest kolumna zawierająca informacje o dniu tygodnia. Oczywiście w tym przypadku obie drużyny grają tego samego dnia, więc jej wpływ powinien być mniejszy niż poprzednio analizowanego parametru.

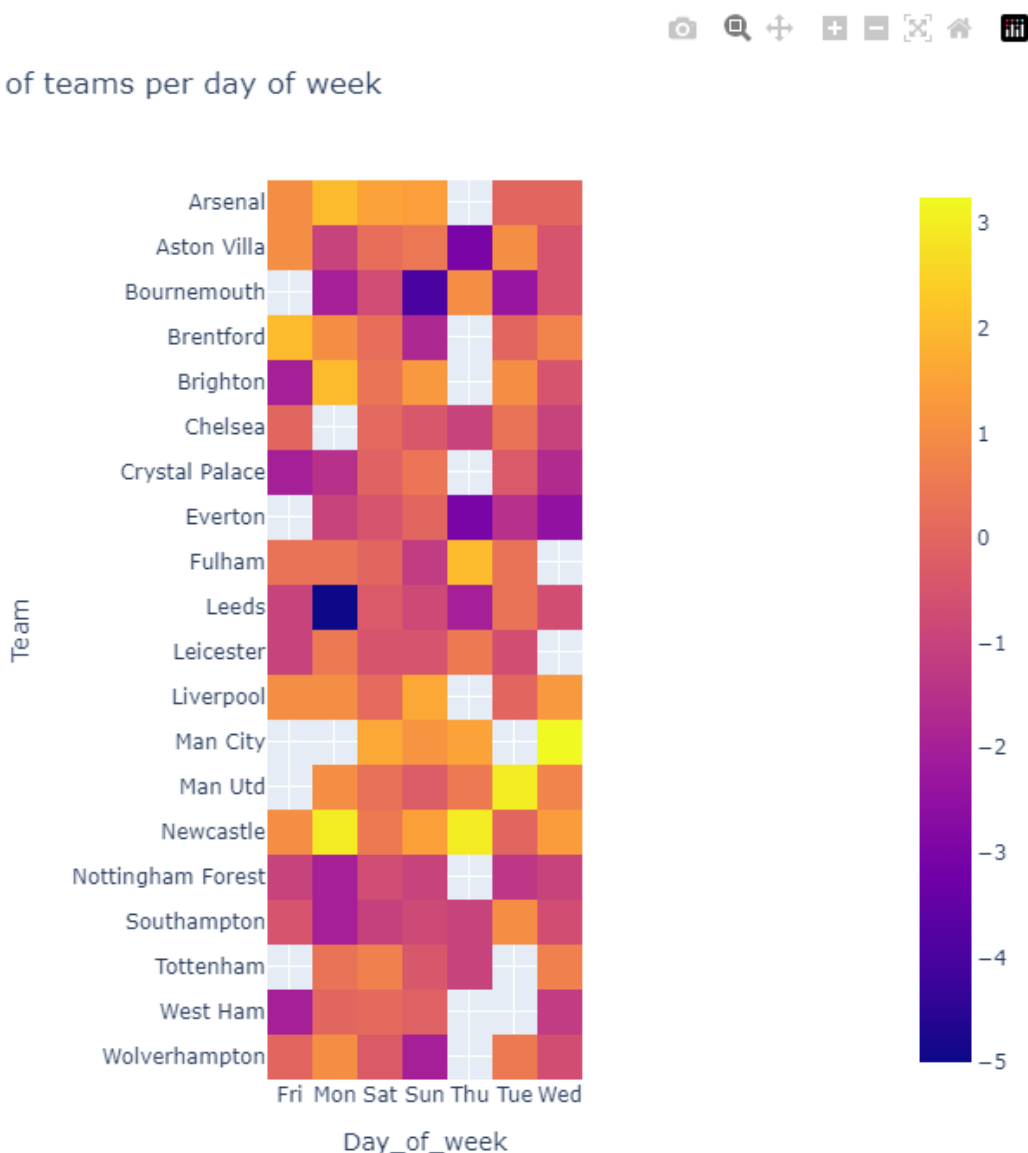
```
dayofweek_df = match_stats.pivot_table('Goals_diff', 'Team', 'Day_of_week')

px.imshow(dayofweek_df, title="Results of teams per day of week", width=700, height=700)
```

19 Kod odpowiedzialny za stworzenia zestawienia zależności rezultatu oraz dnia tygodnia

Jak widać na powyższym kodzie tym razem nie zastosowałem wykresu słupkowego, ponieważ zestawienie rezultatów z poszczególnymi dniami tygodnia nie miałoby sensu, gdyż w każdym meczu w przypadku braku remisu, zawsze jedna drużyna zwycięża, a druga przegrywa. Z tego powodu zdecydowałem się na sprawdzenie, czy drużyny lepiej radzą sobie w poszczególne dni tygodnia.

Drugą różnicą jest to, że zamiast kolumny „Result”, zawierającej sam rezultat w postaci tekstu, zastosowałem tutaj nowo utworzoną wcześniej kolumnę „Goals_diff”, która poza rezultatem definiuje również jego skalę.



20 Wygenerowany wykres, zawierający zależności rezultatów drużyn w zależności od dni tygodnia

Z powyższego wykresu wynika, że w dla niektórych drużyn dni tygodnia mogą wpływać na wynik spotkania. Problemem tego wykresu jest to, że zawiera on kilka pustych miejsc oraz dla niektórych dni tygodnia mogła się odbyć mała liczba meczów, przez co wartości mogą być dość zakłamane.

Przykładowo najciemniejszym punktem na wykresie jest średnia różnica bramek drużyny Leeds dla spotkań odbywających się w poniedziałek i wynosi ona aż -5. Może ona mylić osobę patrzącą na wykres, ponieważ sugeruje ona, że ta drużyna totalnie nie radzi sobie

w meczach odbywających się pierwszego dnia tygodnia i średnio tego dnia przegrywa mecz różnicą pięciu trafień. Tak naprawdę spowodowane jest to tym, że ta drużyna zagrała tego w poniedziałek tylko jeden mecz w tym sezonie, w którym uległa drużynie Liverpoolu aż 6:1. Oczywiście Liverpool jest jedną z mocniejszych drużyn w lidze, więc ta statystyka nie oznacza, że zespół Leeds gra znacznie gorzej w poniedziałki, a po prostu był słabszy od znacznie lepszego rywala.

```
match_stats.Day_of_week.value_counts()
```

Sat	338
Sun	140
Wed	56
Tue	38
Mon	36
Fri	22
Thu	18
Name: Day_of_week, dtype: int64	

21 Ilość meczów w poszczególne dni tygodnia

Jak widać na powyższym zrzucie ekranu, ponad 70% meczów odbywa się w weekend. Z tego powodu dla mojego zbioru danych nie ma sensu umieszczania tej kolumny w algorytmie, ponieważ może ona negatywnie wpłynąć na jego działanie mocno zaniżając prawdopodobieństwo zwycięstwa drużyny podczas meczu w środku tygodnia, jeśli wcześniej odniosła tego dnia porażkę.

```
match_stats.drop('Day_of_week', axis=1, inplace=True)
```

22 Fragment kodu usuwający kolumnę Day_of_week z DataFrame

W wyniku powyższej analizy zdecydowałem się na usunięcie kolumny zawierającej informacje o dniach tygodniach, w których odbywały się mecze, w celu zwolnienia części pamięci, przechowującej mój DataFrame oraz uproszczeniu ramki danych przed dalszą analizą.

Time

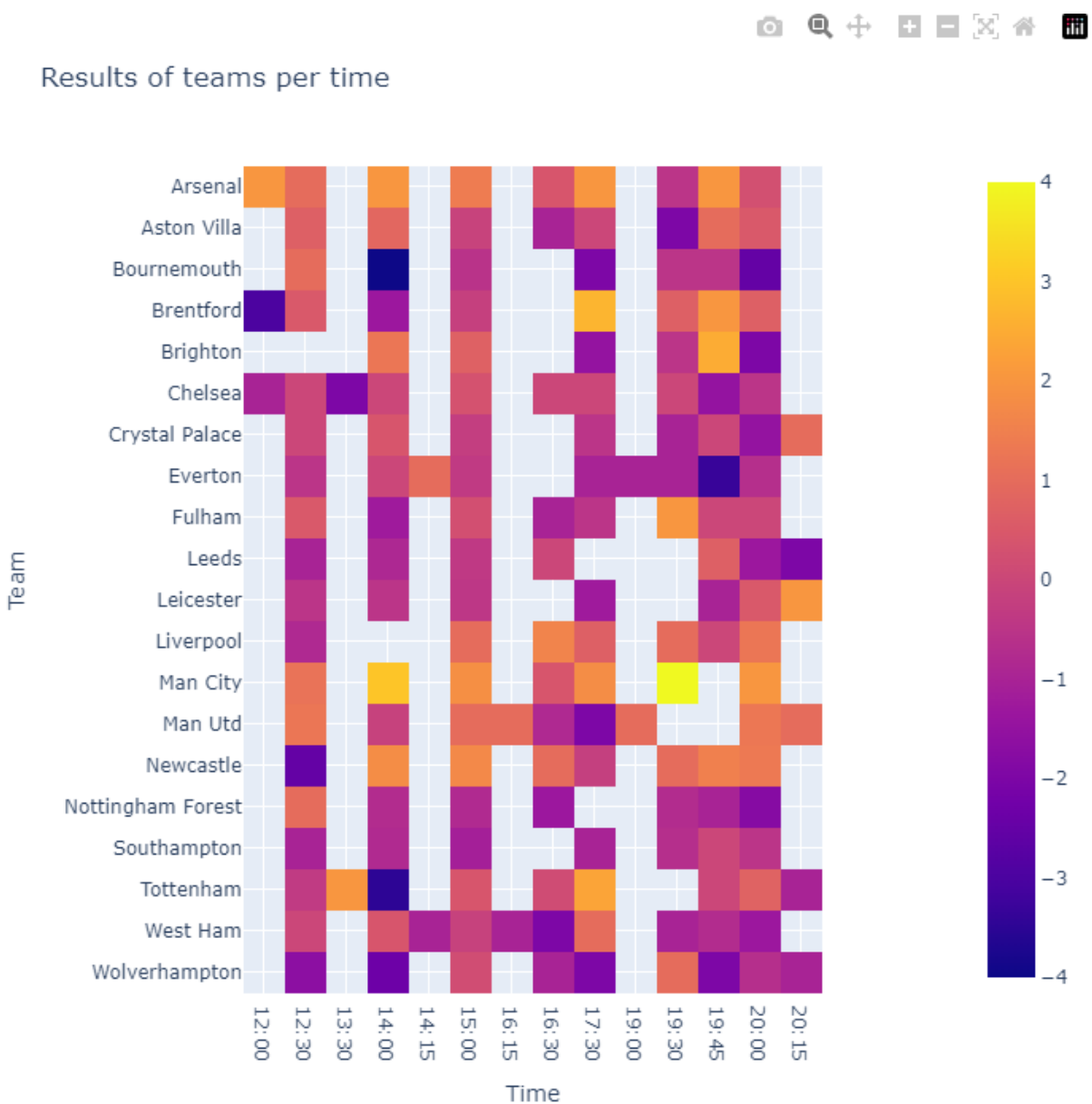
Kolejnym atrybutem meczu, który może mieć wpływ na jego rezultat jest „Time”, który zawiera informacje o czasie rozpoczęcia meczu. Jak można się domyślić część osób, woli grać rano, a część pewnie czuje się we wieczornych spotkaniach, dlatego uważam, że ta kolumna jest warta sprawdzenia.


```
time_df = match_stats.pivot_table('Goals_diff', 'Team', 'Time')

px.imshow(time_df, title="Results of teams per time", width=700, height=700)
```

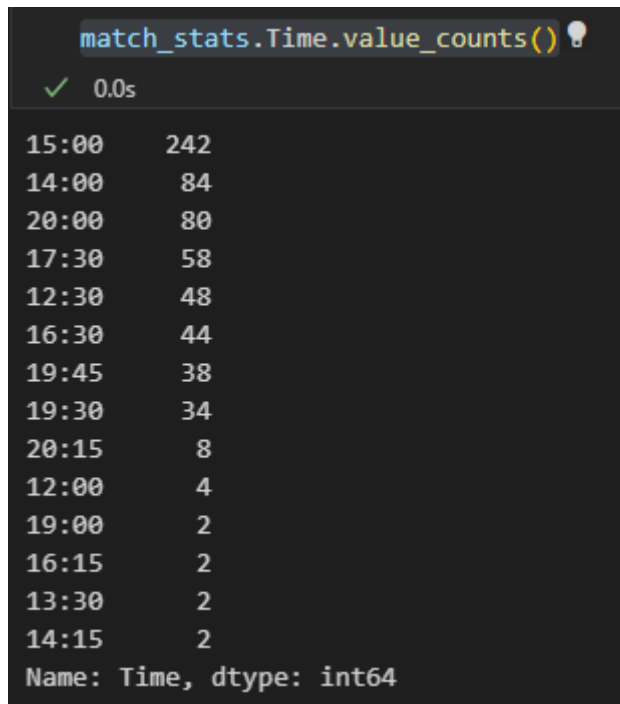
23 Kod odpowiedzialny za stworzenia zestawienia zależności rezultatu oraz dnia tygodnia

Jak widać na powyższym fragmencie kodu, to zestawienie utworzyłem praktycznie w ten sam sposób co poprzednie, zamieniając tylko wartość trzeciego argumentu funkcji `pivot_table` z „Day_of_week” na „Time”.



24 Wygenerowany wykres, zawierający zależności rezultatów drużyn w zależności od godziny rozpoczęcia

Analizując powyższy wykres, podobnie jak w przypadku dni tygodnia, zawiera on dużo godzin bez wartości, oraz wartości skrajnych takich jak 4 w przypadku drużyny Manchesteru City o godzinie 19:30.



25 Ilość meczów rozpoczętych o poszczególnych godzinach

Na powyższym rzucie ekranu, również można zauważyć nierównomierne rozłożenie wartości, gdyż zdecydowanie najwięcej meczów rozpoczyna się o godzinie 15:00.

Podsumowując, dla mojego zbioru danych ta kolumna również, nie nadaje się do użycia w analizie z tego samego powodu co poprzednio przeanalizowane dni tygodnia.

```
match_stats['DateTime'] = pd.to_datetime(
    match_stats['Date'] +
    ' ' +
    match_stats['Time']
)
match_stats.drop(['Date', 'Time'], axis=1, inplace=True)
```

26 Fragment kodu łączący kolumny Date oraz Time w jedną kolumnę

W rezultacie powyższej analizy, za pomocą powyższego kodu, połączyłem kolumny Date oraz Time, w jedną kolumnę zawierającą pełną datę rozpoczęcia meczu piłkarskiego.

Opponent

Ostatnią kolumnę, którą rozważę w tej kategorii jest kolumna „oponent”. W tym przypadku rezultat jest dość oczywisty, ponieważ wiadomo, że przeciwnik w meczu ma kluczowe

znaczenie dla jego przebiegu. Przykładowo dla drużyna ze środka tabeli ma znacznie większe szansę na triumf z drużyną z końcówki stawki niż z liderem ligi.

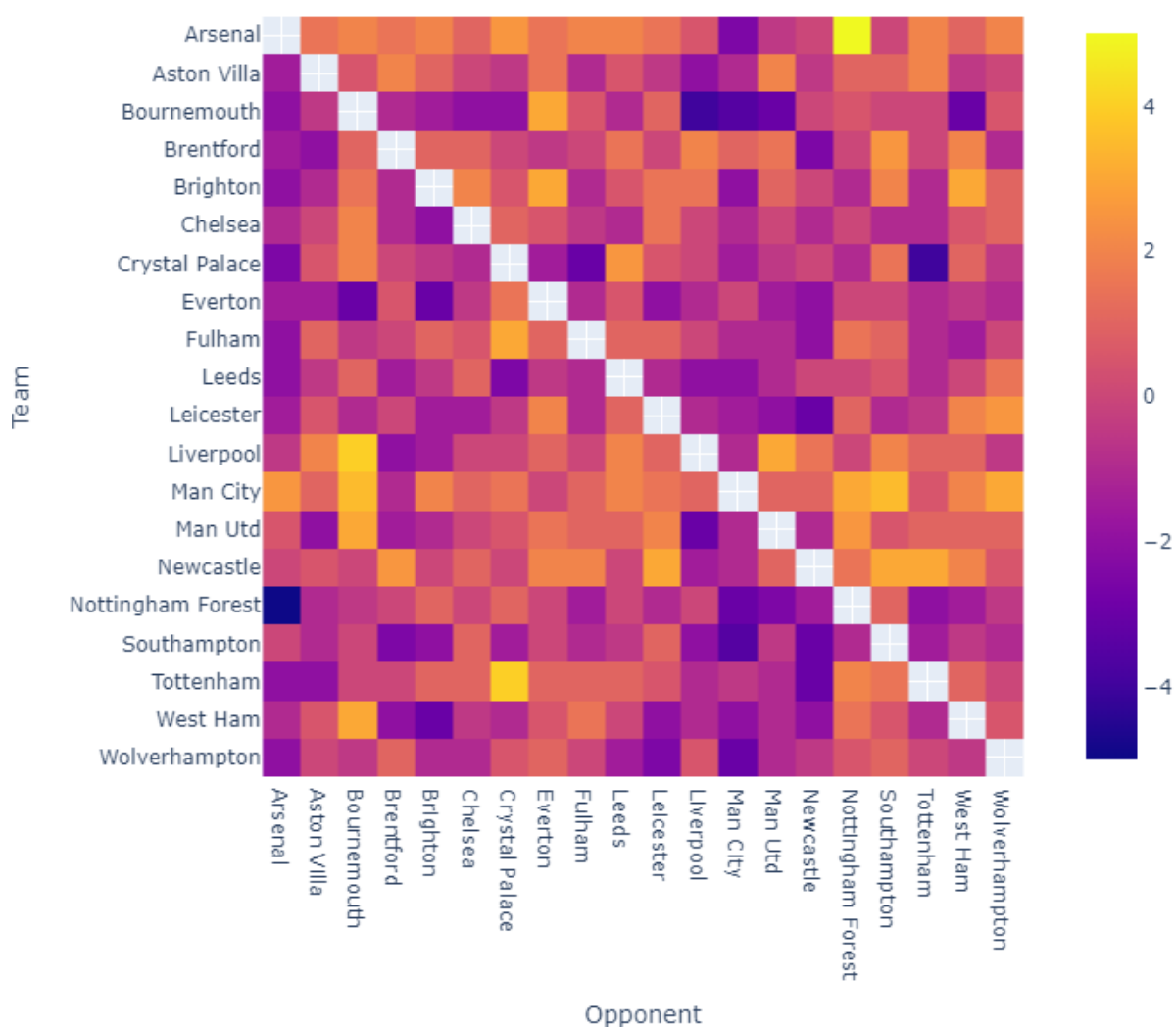
Mimo tego uważam, że wynik nie musi być aż tak zero jedynkowy, ponieważ oprócz samej formy drużyny, mogą liczyć się czynniki takie jak preferencje drużyny. Przykładowo drużyna A, może być najlepszą drużyną w lidze, natomiast od zawsze mieć problem w rywalizacji z klubem B, który jest przeciętną drużyną na tle innych. Tego typu sytuacja może, być spowodowana na przykład tym, że styl drużyny B jest niewygodny dla lepszej drużyny. Z tego powodu uważam, że warto sprawdzić tę zależność na wykresie.

```
direct_confrontations_df = match_stats.pivot_table('Goals_diff', 'Team', 'Opponent')  
  
px.imshow(direct_confrontations_df, title="Direct confrontations of teams",  
width=700, height=700)
```

27 Kod odpowiedzialny za stworzenia zestawienia zależności rezultatu meczu od przeciwnika

To zestawienie stworzyłem praktycznie w ten sam sposób jak dla poprzednich dwóch atrybutów, ustawiając wartość trzeciego argumentu funkcji `pivot_table` na „Opponent”.

Direct confrontations of teams



28 Wygenerowane zestawienie rezultatów pomiędzy drużynami w Premier League

Jak widać na powyższym zestawieniu, co jest dość oczywiste, przeciwnik ma kluczowe znaczenie w skali zwycięstwa drużyny. Przykładowo jak wcześniej wspominałem, w tym przypadku rolę nie tylko odgrywa ogólna forma zespołu w tym sezonie, lecz także same preferencje drużyn.

Przykładowo obecnie najmocniejsza drużyna w lidze Manchester City, najgorszy bilans bramkowy w tym sezonie ma przeciwko drużynie Brentford, która na dzień dzisiejszy (29.04.2023r.) plasuje się dopiero na 9 miejscu. Dodatkowo średnia liczba goli przeciwko wiceliderowi, którym jest Arsenal wynosi aż 2.5, co może oznaczać, że jest ona dla nich dość komfortowym przeciwnikiem.

Niestety powyższa analiza jest dość niepełna, ponieważ posiadam tylko dane z obecnego sezonu, przez co każda drużyna grała przeciwko innej od jednego do dwóch spotkań.

Mimo to uważam, że jest to najważniejsza statystyka podczas tej analizy, przez co wezmę ją pod uwagę podczas uczenia maszynowego, które zastosuje w dalszej części tego rozdziału.

Wpływ atrybutów liczbowych na wynik meczu

Drugą grupą kolumn, które mają wpływ na wynik meczu są wartości liczbowe.

W porównaniu do poprzednio przeanalizowanych wartości, tego typu statystyki dostępne są tylko po meczu, przez co nie mamy do nich dostępu w spotkaniu piłkarskim, którego wynik chcemy przewidzieć.

Mimo tego na podstawie historycznych wydarzeń za pomocą uczenia maszynowego jesteśmy w stanie, oszacować ich liczbę w nadchodzącym widowisku sportowym.

Z tego powodu w tej części podrozdziału przeanalizuję, które z nich mają największy wpływ na końcowy rezultat.

Rozkład wartości

Pierwszą właściwością badanych kolumn, którą przeanalizuje będzie rozkład ich wartości. Zdecydowałem się na sprawdzenie tej właściwości, ponieważ może być ona przydatna podczas uczenia oraz możemy dzięki niej lepiej poznać dane.

Rozkład wartości wygenerowałem za pomocą biblioteki Seaborn, która podobnie jak wcześniej użyte Plotly służy do wizualizacji danych.

Główną różnicą pomiędzy tymi dwoma narzędziami jest to, że wcześniej użyte Plotly pozwala na tworzenie interaktywnych wykresów, które użytkownik między innymi może przybliżać oraz przesuwać. Seaborn to natomiast biblioteka zbudowana na Matplotlib, posiadająca ulepszony interfejs oraz dodatkowe możliwości wizualizacji danych.

Główną jej zaletą jest to, że wymaga małej ilości kodu podczas tworzenia wizualizacji. (Sharma, 2023)

W przypadku mojego projektu obie biblioteki w stu procentach spełniają swoją funkcję.

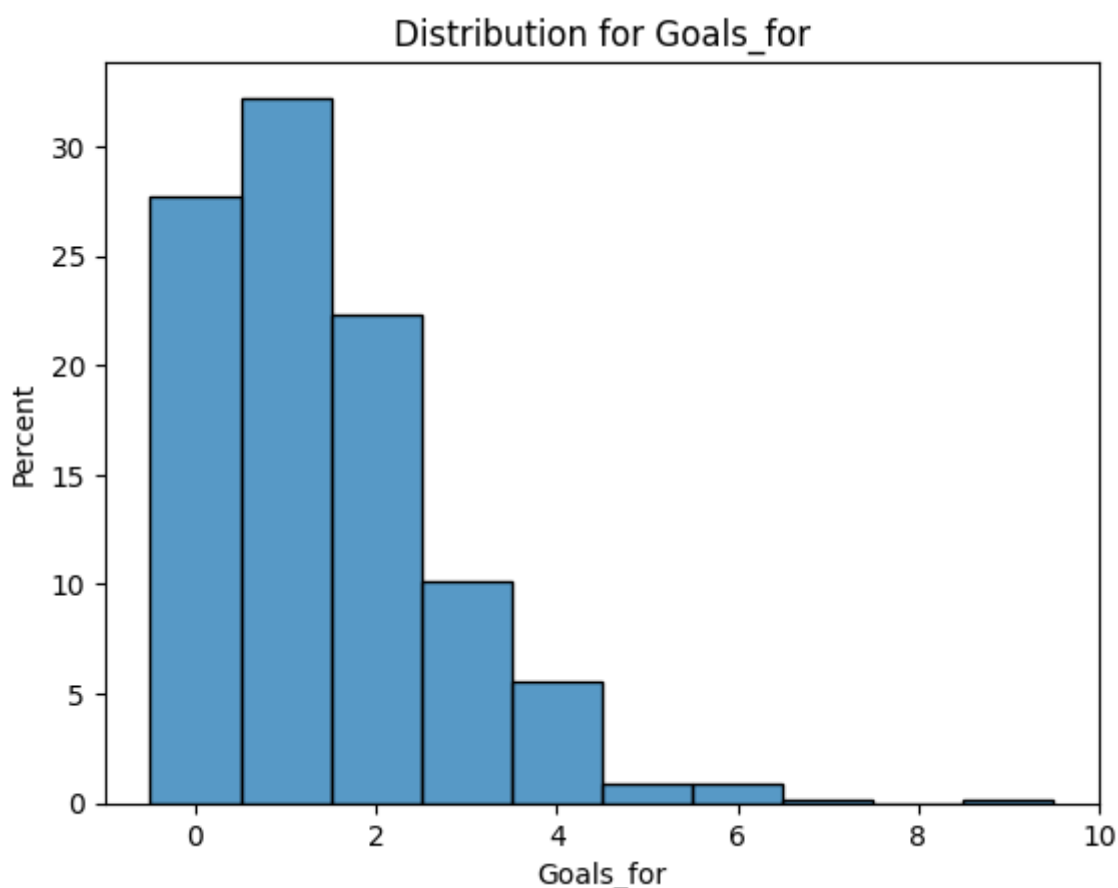
```
import matplotlib.pyplot as plt
import seaborn as sns

for col in match_stats.select_dtypes('number').columns:
    sns.histplot(match_stats[col], stat="percent", discrete=True)

    plt.title(f"Distribution for {col}")
    plt.show()
```

29 Kod odpowiedzialny za stworzenie rozkładu wartości wszystkich kolumn typu liczbowego

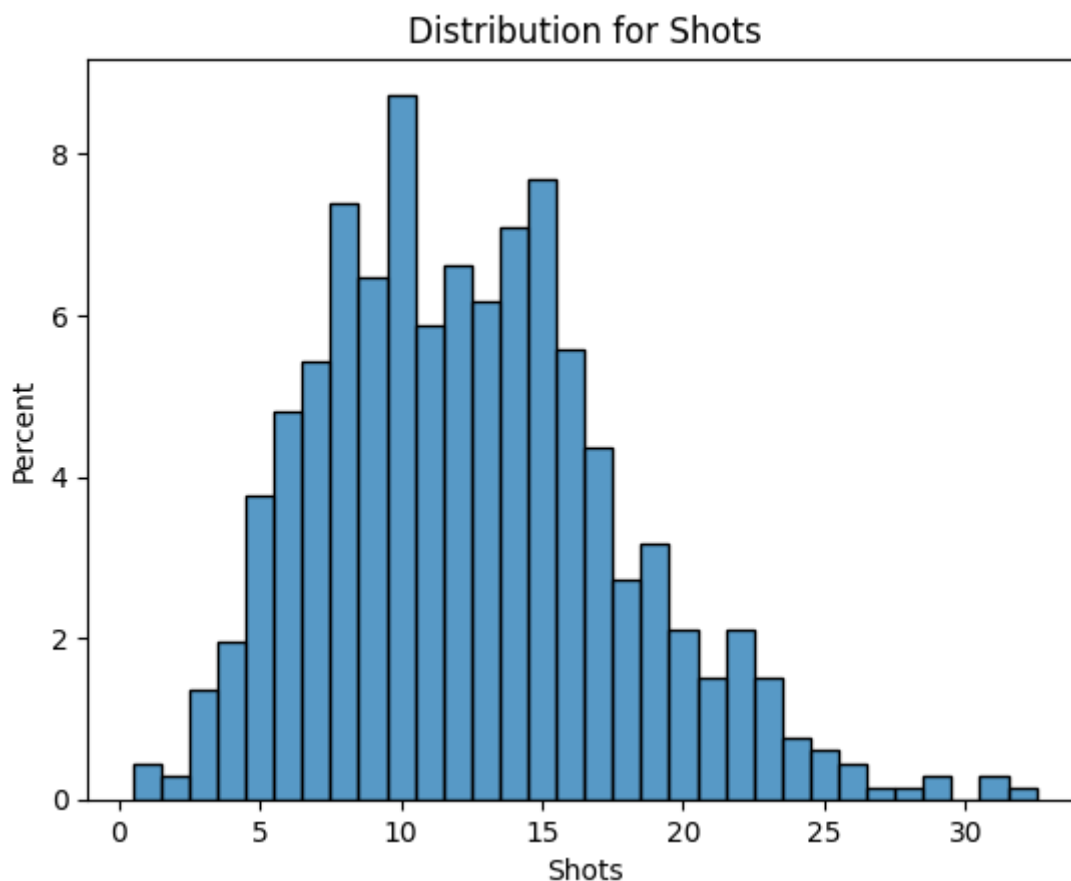
Jak widać na powyższym fragmencie kodu, tworzenie wykresów za pomocą powyżej opisanej biblioteki Seaborn jest dość proste i nie wymaga wielu linijek kodu. Dodatkowo można zauważyć, że pozwala ona również na korzystanie z funkcji oferowanych przez bibliotekę matplotlib, co wynika z tego, że Seaborn jest jej rozszerzeniem.



30 Wygenerowany za pomocą biblioteki Seaborn procentowy rozkład ilości strzelonych bramek

Jak widać na powyższym wykresie rozpiętość wartości strzelonych bramek nie jest zbyt szeroka i ich liczba wynosi głównie 0, 1 lub 2. Oczywiście taki wynik nie jest zaskoczeniem, ponieważ drużyny na najwyższym poziomie często grają bardziej defensywny football, przez co liczba strzelonych goli przez jedną z drużyn rzadko przekracza liczbę 2.

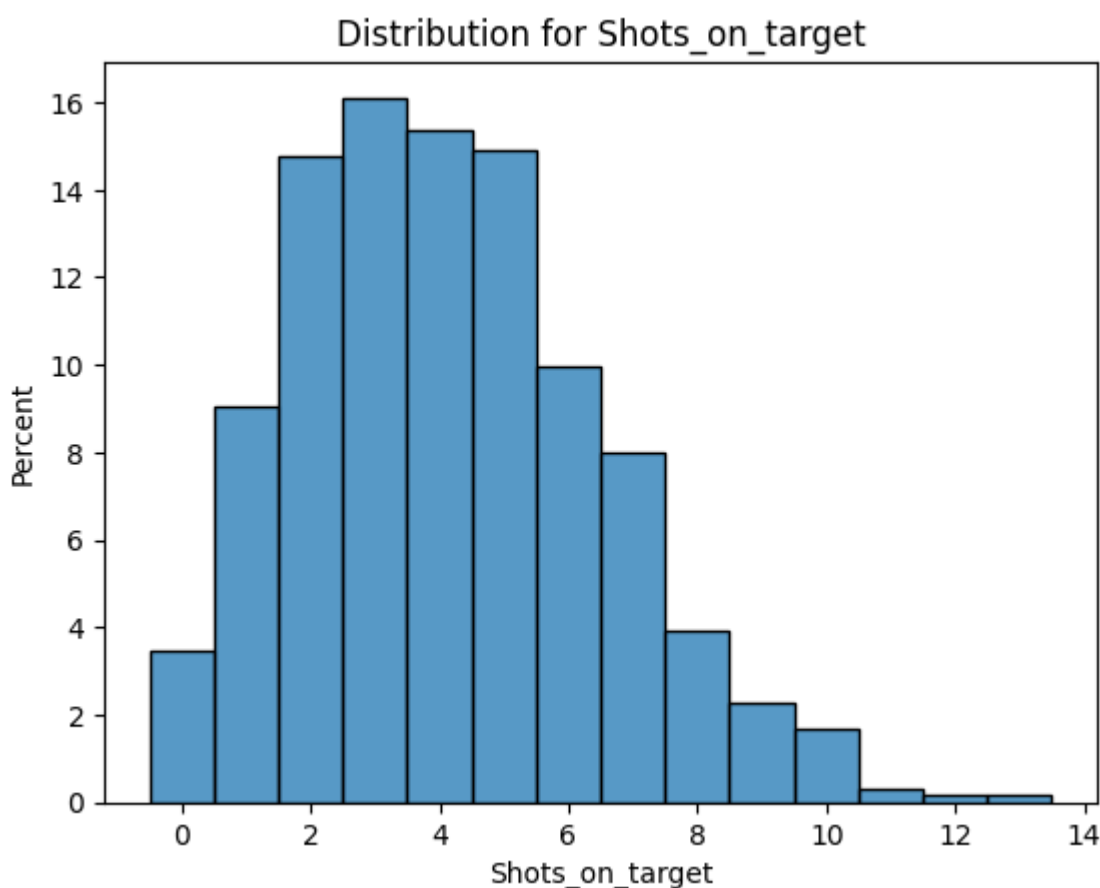
Z powyższego wniosku wynika, że ze względu na małą różnorodność wartości, lekka zmiana tego parametru może mieć duży wpływ na końcowy rezultat.



31 Wygenerowany za pomocą biblioteki Seaborn procentowy rozkład ilości oddanych strzałów

W tym przypadku wykres wygląda zupełnie inaczej niż poprzedni, gdyż wartość tej statystyki zawiera liczby ze znacznie większego przedziału. Większość z nich znajduje się pomiędzy liczbą 9 a 16, ale rekordowo ta statystyka wynosiła nawet 32 strzały, co jest dość imponującym wynikiem.

Na podstawie tego wykresu można stwierdzić, że nie wielka zmiana tej wartości prawdopodobnie nie będzie miała znaczącego wpływu na wynik spotkania.

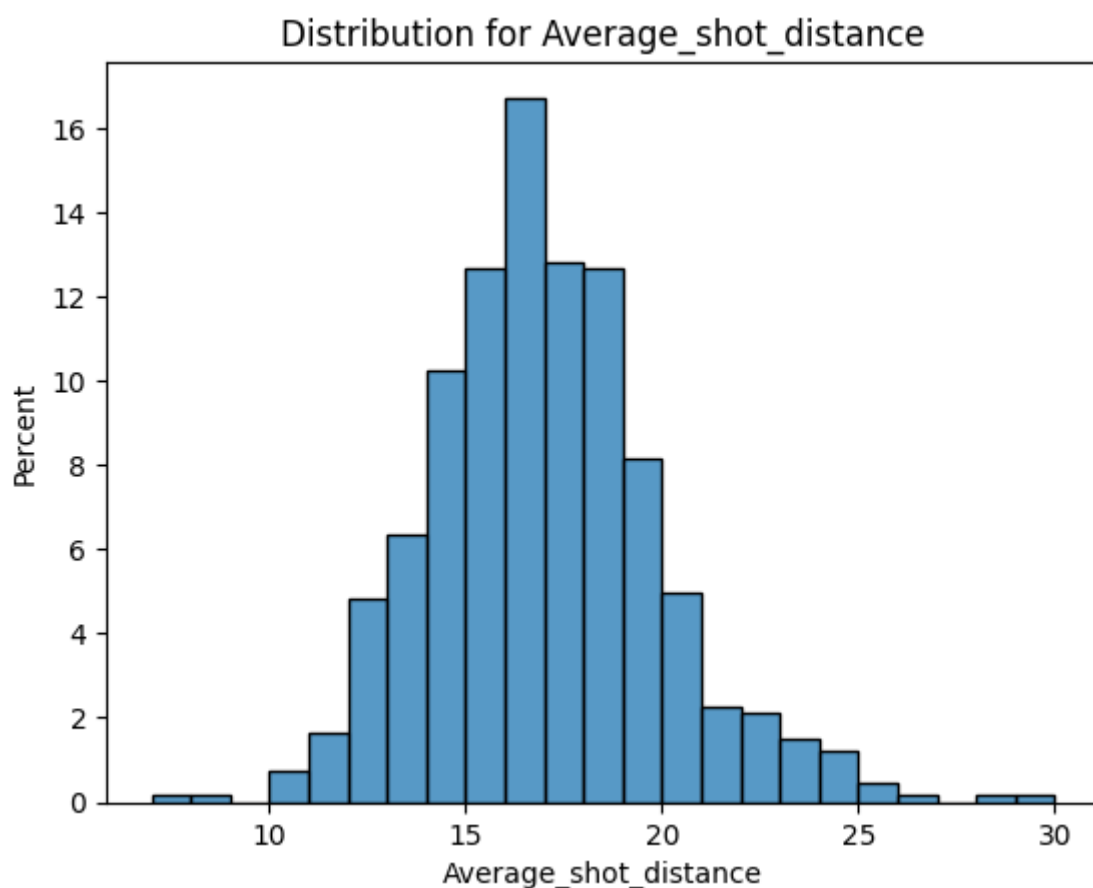


32 Wygenerowany za pomocą biblioteki Seaborn procentowy rozkład ilości oddanych strzałów celnych

Na powyższym wykresie można zauważyć, że wartości liczby oddanych strzałów są znacznie mniejsze od poprzednio przeanalizowanej właściwości. Oczywiście jest to dość logiczny wniosek, ponieważ liczba oddanych strzałów to po prostu suma strzałów celnych oraz niecelnych z czego wynika, że w najlepszym przypadku te wartości mogą być równe. Co ciekawe na powyższym zestawieniu widać, że większość wartości tej statystyki zawiera się w przedziale 2-5, z czego wynika, że w większości meczów jest znacznie więcej strzałów niecelnych.

Podsumowując analizę tego wykresu, można stwierdzić że statystyka oddanych strzałów celnych ma dość duży wpływ na końcowy rezultat meczu, ze względu na dość wąski rozkład jej wartości.

W porównaniu do poprzednich kolumn, w tym przypadku rezultat jest dla mnie dość zaskakujący, ponieważ nie spodziewałem się aż tak dużej różnicy pomiędzy strzałami celnymi oraz niecelnymi.



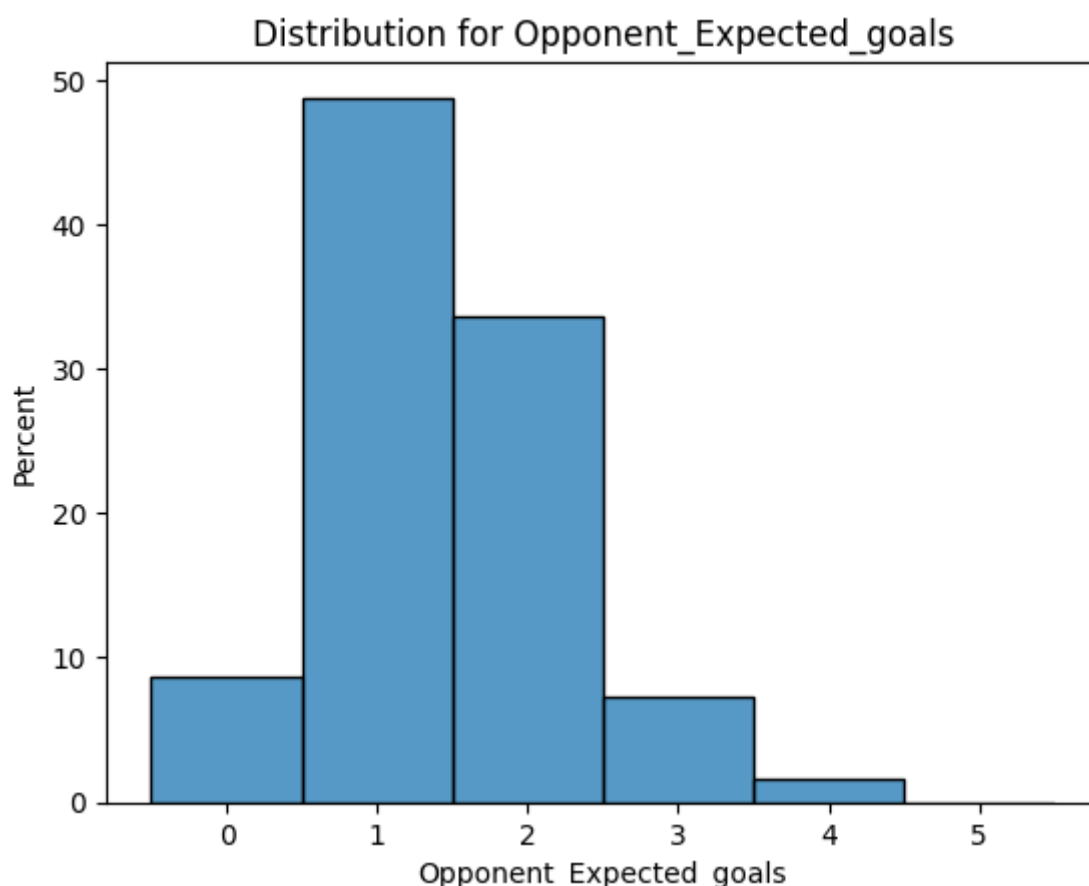
33 Wygenerowany za pomocą biblioteki Seaborn procentowy rozkład średniej odległości oddanych strzałów

Kolejną analizowaną statystyką jest średnia odległość oddanych strzałów przez drużynę. W tym przypadku w porównaniu do innych parametrów jest ona odwrotnie proporcjonalna do liczby strzelonych bramek, przez co jej niska wartość działa na korzyść zespołu.

Jak widać na wykresie większość jej wartości znajduje się w dość wąskim przedziale, ale zawiera ona również wartości mocno skrajne. Wynika to prawdopodobnie z tego, że w niektórych meczach jedna z drużyn oddała małą ilość strzałów, które akurat odbyły się w bardzo bliskiej bądź dalekiej odległości od bramki rywala.

Mimo że wartości tej statystyki znajdują się w tak samo szerokim przedziale co liczby oddanych strzałów, ich znacząca większość umiejscowiona jest w wąskim przedziale pomiędzy liczbą 15 a 20.

Z tego wynika, że we większości spotkaniach ta liczba będzie podobna, ale wystąpienie skrajnej wartości może mieć spory wpływ na wynik meczu.



34 Wygenerowany za pomocą biblioteki Seaborn procentowy rozkład oczekiwanej ilości strzelonych bramek

Jak widać na powyższym wykresie w tym przypadku rozkład tej statystyki jest największy ze wszystkich omówionych w tej części rozdziału dotyczącego uczenia maszynowego. Co ciekawe około 80% wartości omawianej kolumny znajduje się w przedziale od 1 do 2.

Taki rezultat wynika prawdopodobnie z tego, że wartości tej kategorii nie są po prostu pustymi statystykami z meczu, lecz są wyliczoną przez specjalny algorytm liczbą. Warto również podkreślić, że w porównaniu do liczby strzałów oraz bramek nie jest ona liczbą całkowitą i zawiera ona wartość ułamkową z dokładnością do jednego miejsca po przecinku.

Niestety nie znam dokładnego algorytmu wyliczającego tą statystykę, gdyż jej wartości tak samo jak wszystkie inne za pomocą Web Scrapingu, pozyskałem z zewnętrznej strony, ale z reguły polega ona na wyliczeniu prawdopodobieństwa strzelenia gola dla każdego strzału drużyny, a końcowym rezultatem jest suma prawdopodobieństw wszystkich strzałów.

Szanse na zdobycie punktu po oddaniu strzału są wyliczane na podstawie wielu parametrów, z których najważniejszymi są:

- Odległość od bramki
- Kąt oddania strzału
- Część ciała, z której piłkarz oddał strzał

Oczywiście istnieje wiele algorytmów wyliczających to prawdopodobieństwo, przez co wartości tej statystyki mogą się różnić na innych stronach. (StatsBomb)

Podsumowując, na podstawie bardzo niskiej rozbieżności wartości tego parametru można zauważyć, że jego zmiana prawdopodobnie ma ogromny wpływ na końcowy rezultat rozgrywek piłkarskich.

Wzajemna zależność wartości liczbowych

Kolejną rzeczą jaką sprawdzę w tym podrozdziale jest zależność wartości liczbowych od siebie. Dzięki tej statystyce będę mógł przeanalizować w jaki sposób różne wartości na siebie wpływają, co również pozwoli mi stwierdzić, które z nich mają największy wpływ na końcowy rezultat spotkania.

To zestawienie zrealizuje za pomocą wbudowanej do biblioteki Pandas funkcji `corr()`, która pozwoli mi w łatwy sposób wygenerować korelacje podanych w parametrze kolumn.

```
stat_cols = [
    'Shots', 'Shots_on_target', 'Average_shot_distance', 'Expected_goals',
    'Opponent_Shots', 'Opponent_Shots_on_target', 'Opponent_Average_shot_distance',
    'Opponent_Expected_goals', 'Goals_for', 'Goals_against', 'Goals_diff'
]

stat_correlations = match_stats[stat_cols].corr()
```

35 Kod odpowiedzialny za stworzenie korelacji wartości wybranych kolumn

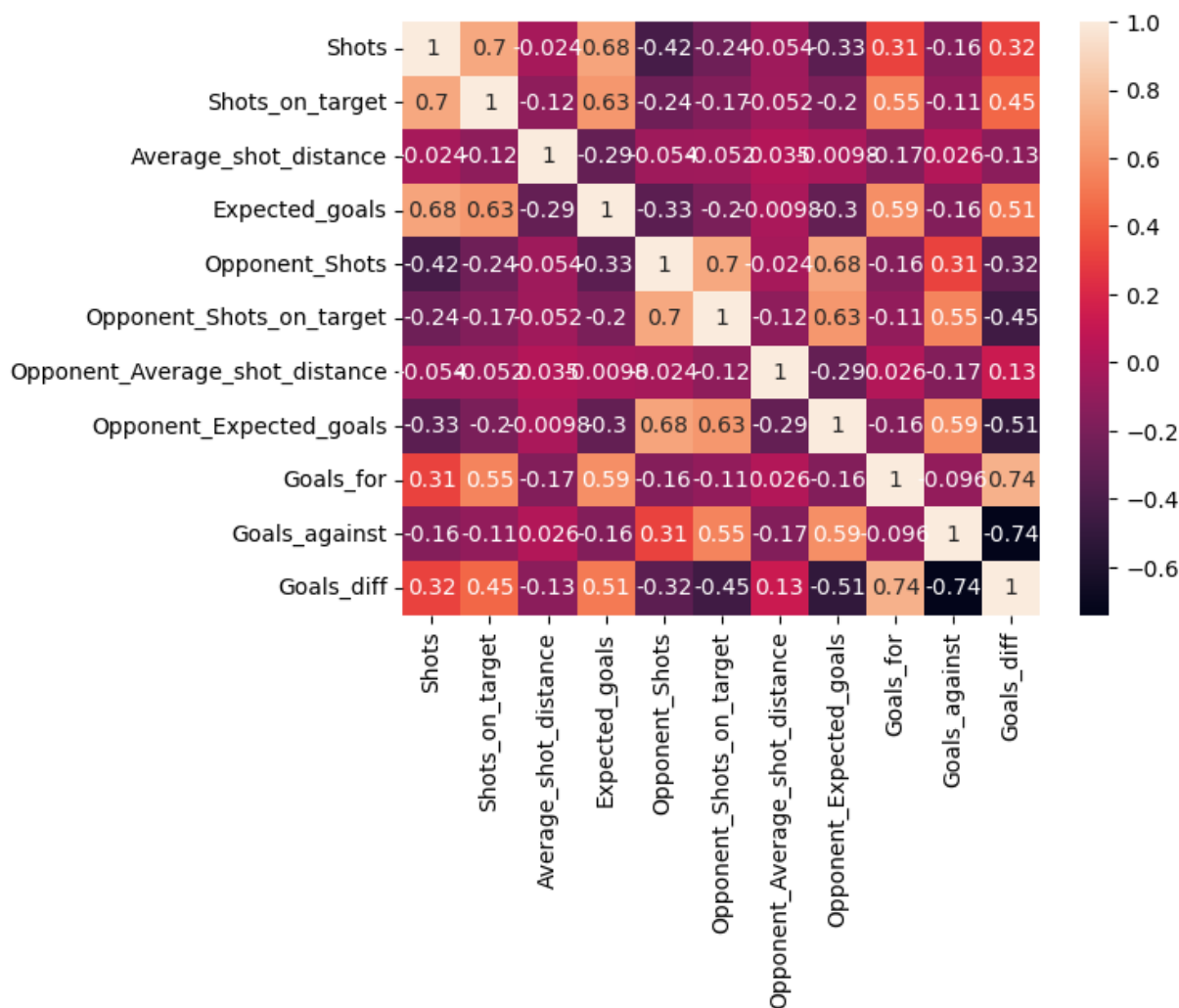
Jak widać na powyższym fragmencie kodu, dzięki wyżej wymienionej funkcji, stworzenie korelacji wartości jest bardzo proste i nie wymaga dużej ilości kodu.

Uzyskaną korelację danych w celu lepszej czytelności przedstawię w formie wykresu typu heatmap wygenerowanego za pomocą biblioteki Seaborn.

```
sns.heatmap(stat_correlations, annot=True)
```

36 Kod generujący heatmape zawierającą wzajemne zależności wartości liczbowych

Jak można zauważyć, podobnie jak w poprzednim przypadku biblioteka Seaborn pozwoliła na stworzenie oczekiwanego wykresu w prosty i szybki sposób.



37 Wygenerowany za pomocą biblioteki Seaborn wykres, zawierający wzajemne zależności wartości liczbowych

Na powyższym wykresie można zauważyć w jaki sposób dana wartość wpływa na inną. Wartości w każdej komórce oznaczają w jakim stopniu dane statystyki są powiązane. Wygenerowane wartości mieszczą się w przedziale od -1 do 1 i nim bliższe skrajnych wartości są tym bardziej wpływają na siebie. Dodatkowo dodatniość tej wartości oznacza, że parametry zwiększenie jednego powoduje również powiększenie drugiego, a w przypadku ujemnej wartości korelacji w przypadku zwiększenia jednego, drugiego wartość maleje.

Z wykresu możemy wyczytać kilka ciekawych własności takich jak wpływ ilości całkowitej liczby strzałów na liczbę strzałów na bramkę.

Dodatkowo możemy znaleźć bardziej nieoczywiste zależności takie jak wpływ ilości strzałów przeciwnika na strzały drużyny, z którego wynika, że nim więcej strzałów ma przeciwnik tym mniej ma drużyna. W porównaniu do poprzedniego zestawienia strzałów całkowitych oraz celnych, gdzie korelacja wynosiła aż 0.7, to tutaj jest ona znacznie bliższa 0 i wynosi -0.24. Ta wartość wynika prawdopodobnie z tego, że gdy przeciwnik oddaje dużo strzałów na bramkę jest prawdopodobnie bardziej wymagającym rywalem, więc trudniej się go atakuje,

a dodatkowo posiada piłkę przez więcej czasu, więc druga drużyna ma w rezultacie mniej sytuacji bramkowych.

Mimo tego, że wygenerowany wykres zawiera bardzo dużo interesujących informacji, dla mojego projektu zdecydowanie najistotniejsza jest ostatnia kolumna, która zawiera wpływ danej statystyki na różnice bramek, czyli tak naprawdę na końcowy rezultat meczu.

```
stat_correlations['Goals_diff'].sort_values()
✓ 0.0s
```

Goals_against	-0.740231
Opponent_Expected_goals	-0.511982
Opponent-Shots_on_target	-0.446206
Opponent-Shots	-0.319166
Average_shot_distance	-0.128965
Opponent_Average_shot_distance	0.128965
Shots	0.319166
Shots_on_target	0.446206
Expected_goals	0.511982
Goals_for	0.740231
Goals_diff	1.000000

Name: Goals_diff, dtype: float64

37 Wpływ poszczególnych statystyk na końcowy rezultat meczu

Jak można było się spodziewać największy wpływ na zwycięstwo ma ilość strzelonych bramek, ale to dość oczywisty wniosek wynikający ściśle z zasad tego sportu.

Dodatkową ciekawą własnością jest wpływ statystyki oczekiwanej liczby bramek na wynik, która wynosi około 0.52, co oznacza że również jest dość mocno powiązana z ostatecznym wynikiem spotkania. Za pomocą tego wyniku, można również zweryfikować skuteczność modelu generującego tę statystykę.

Kolejnymi statystykami są liczby oddanych strzałów, z których jak wiadomo liczba oddanych strzałów celnych jest dość znacząco bardziej istotna w wyniku niż łączna ilość strzałów, co również łatwo było zauważyć podczas badania rozkładu danych w poprzedniej części tego podrozdziału.

Ostatnim z badanych parametrów jest średnia odległość oddanych strzałów, która jak się okazało ma dość niski wpływ na wynik. Wynika to prawdopodobnie między innymi z właściwości, która zauważyłem wcześniej dotyczącą tego, że większość strzałów jest oddawana z podobnego dystansu.

Inną właściwością rzucającą się w oczy na powyższym zrzucie ekranu jest zależność statystyk drużyny oraz przeciwnika, gdzie wartości są do siebie przeciwne. Wynika to oczywiście z tego, że każda statystyka wpływa na wynik obu zespołów w taki sam sposób, tylko skierowany w przeciwnym kierunku.

2.3 Uczenie maszynowe

Jak łatwo można się domyślić w tym podrozdziale zajmę się wytrenowaniem modelu w celu jak najlepszego wyliczania kursów meczów. Do weryfikacji danych posłużę się kursami na mecze 35 kolejki ligi angielskiej oferowanymi przez STS, które jest obecnie największym bukmacherem w Polsce.

Utworzenie modeli

Pierwszym krokiem podczas tworzenia algorytmu, będzie wyuczenie modeli, które będą przewidywać wartość danego parametru liczbowego na podstawie posiadanych informacji o nadchodzącym spotkaniu.

Na podstawie wcześniejszej analizy doszedłem do wniosku, że najbardziej istotnymi parametrami dotyczącymi nadchodzącego meczu, które posiadamy są nazwy drużyn oraz informacja o lokalizacji wydarzenia. Z tego powodu w tej sekcji postaram się za ich pomocą przewidywać liczbę innych statystyk takich jak liczba strzelonych goli.

W tym celu utworzyłem pięć modeli, z których każdy jest odpowiedzialny za przewidywanie jednej z poniższych statystyk liczbowych:

- Liczba strzelonych goli
- Liczba oddanych strzałów
- Liczba strzałów celnych
- Oczekiwana liczba goli
- Średnia odległość oddanych strzałów przeciwnika

Wyżej wspomniane modele utworzyłem przy użyciu biblioteki o nazwie Statsmodels. Jest to mniej popularny od poprzednio wspomnianych modułów do Pythona, udostępniający szereg klas oraz funkcji służących do estymacji modeli statystycznych. (statsmodels, 2023)

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

goals_poisson_model = smf.glm(
    formula="Goals_for ~ Venue + Team + Opponent",
    data=match_stats,
    family=sm.families.Poisson()
).fit()
```

38 Kod tworzący model służący do przewidywania liczby strzelonych goli

Jak widać na powyższym fragmencie kodu, modele utworzyłem za pomocą metody `glm()`, oferowanej przez wcześniej wspomniane api.

Jako pierwszy parametr metody podałem formułę, która zawiera atrybut do wyliczenia oraz wartości, które ma brać pod uwagę. Kolejnym parametrem jest po prostu zestaw danych, które funkcja wykorzysta do uczenia. Jako ostatni podałem oferowany przez tę bibliotekę rozkład z jakiego ma korzystać podczas uczenia modelu. Jako wspomniany rozkład wybrałem regresję Poissona, która jest typem analizy regresji, używanym głównie do modelowania liczb.

Na opisanym fragmencie kodu utworzyłem model przewidujący liczbę strzelonych goli podczas spotkania. Analogicznie przeprowadziłem takie uczenie również dla wcześniej wymienionych statystyk.

```
goals_poisson_model.summary()
```

39 Kod generujący podsumowanie modelu

goals_poisson_model.summary() ?

✓ 0.1s

Generalized Linear Model Regression Results							
Dep. Variable:	Goals_for	No. Observations:	666				
Model:	GLM	Df Residuals:	626				
Model Family:	Poisson	Df Model:	39				
Link Function:	Log	Scale:	1.0000				
Method:	IRLS	Log-Likelihood:	-954.93				
Date:	Tue, 02 May 2023		Deviance:	717.82			
Time:	15:44:30		Pearson chi2:	615.			
No. Iterations:	5		Pseudo R-squ. (CS):	0.2381			
Covariance Type:	nonrobust						
		coef	std err	z	P> z	[0.025	0.975]
	Intercept	0.4902	0.205	2.394	0.017	0.089	0.892
	Venue[T.Home]	0.3038	0.066	4.616	0.000	0.175	0.433
	Team[T.Aston Villa]	-0.5437	0.186	-2.916	0.004	-0.909	-0.178
	Team[T.Bournemouth]	-0.7823	0.202	-3.869	0.000	-1.179	-0.386
	Team[T.Brentford]	-0.4277	0.180	-2.378	0.017	-0.780	-0.075
	Team[T.Brighton]	-0.1891	0.172	-1.102	0.270	-0.525	0.147
	Team[T.Chelsea]	-0.9446	0.215	-4.391	0.000	-1.366	-0.523
	Team[T.Crystal Palace]	-0.7974	0.204	-3.907	0.000	-1.198	-0.397
	Team[T.Everton]	-1.0712	0.224	-4.786	0.000	-1.510	-0.632
	Team[T.Fulham]	-0.5351	0.188	-2.845	0.004	-0.904	-0.166
	Team[T.Leeds]	-0.5967	0.191	-3.128	0.002	-0.971	-0.223
	Team[T.Leicester]	-0.5401	0.187	-2.892	0.004	-0.906	-0.174
	Team[T.Liverpool]	-0.1614	0.169	-0.955	0.340	-0.493	0.170
	Team[T.Man City]	0.1115	0.158	0.705	0.481	-0.198	0.421
	Team[T.Man Utd]	-0.4275	0.183	-2.337	0.019	-0.786	-0.069

40 fragment rezultatu komendy generującej podsumowanie modelu

Po utworzeniu podsumowania modelu dotyczącego przewidywania liczby strzelonych goli za pomocą umieszczonego wcześniej kodu, uzyskałem wynik przedstawiony na powyższym zrzucie ekranu. Pokazuje on między innymi wpływ danej wartości tekstowej wybranej kolumny na badaną liczbę. Wspomnianą zależność można wyczytać z kolumny coef,

której nim większa jest wartość tym więcej bramek pada w danym przypadku. Przykładowo w przypadku wartości „Home” kolumny „Venue”, coef wynosi aż 0.3 co oznacza, że bycie gospodarzem ma bardzo duży wpływ na bramkostrzelność. To podsumowanie modelu zawiera również wpływ grających drużyn na strzelone bramki, z czego ten współczynnik jest oczywiście najwyższy dla drużyny Manchester City, która strzeliła najwięcej bramek ze wszystkich zespołów w lidze. W dalszej części podsumowania (poza rzutem ekranu), znajdują się analogicznie informacje o wpływie przeciwnika na liczbę strzelonych bramek, dla których wartość coef jest największa dla klubów najczęściej tracących bramki.

Wyznaczanie prawdopodobieństwa danego rezultatu meczu

Po wygenerowaniu modelu kolejnym krokiem jest uzyskanie za pomocą tych wartości prawdopodobieństw poszczególnych rezultatów spotkania. W tym celu utworzyłem funkcję `simulate_match()`, która jako podstawowe argumenty przyjmuje wytrenowany wcześniej model oraz nazwy drużyn biorące udział w meczu.

```
import numpy as np

from scipy.stats import poisson

def simulate_match(foot_model, homeTeam, awayTeam, max_value=10):
    home_value_avg = foot_model.predict(pd.DataFrame(data={
        'Team': homeTeam,
        'Opponent': awayTeam,
        'Venue': 'Home'
    }, index=[1])).values[0]

    away_value_avg = foot_model.predict(pd.DataFrame(data={
        'Team': awayTeam,
        'Opponent': homeTeam,
        'Venue': "Away"
    }, index=[1])).values[0]

    home_team_pred = [poisson.pmf(i, home_value_avg) for i in range(0, max_value+1)]
    away_team_pred = [poisson.pmf(i, away_value_avg) for i in range(0, max_value+1)]

    return(np.outer(np.array(home_team_pred), np.array(away_team_pred)))
```

41 Funkcja generująca prawdopodobieństwa na poszczególne wyniki meczu

Powyższa funkcja na początku na podstawie wytrenowanego wcześniej za pomocą funkcji `predict()`, przewiduje liczbę strzelonych goli przez każdą z podanych w parametrach drużyn.

W następnym kroku za pomocą funkcji `pmf()` oferowanej przez bibliotekę `scipy`, generuje listę zawierającą prawdopodobieństwa na strzelenie danej liczby goli przez drużynę. Następnie po użyciu funkcji `outer()` z biblioteki `numpy`, łączymy te dwie listy w jedną poprzez wykonanie na nich iloczynu wektorowego. W jej rezultacie otrzymujemy zbiór prawdopodobieństw na wszystkie wyniki, których maksymalna liczba goli nie przekracza podanej w parametrze funkcji. Wynika to z tego, że przykładowo prawdopodobieństwo na strzelenie dokładnie jednego gola przez drużynę A wynosi 0.3, a przez drugi zespół 0.4 to szansa, że spotkanie zakończy się wynikiem 1:1 jest równe iloczynowi tych prawdopodobieństw czyli wynosi ono 12%.

Skoro mamy już listę prawdopodobieństw na uzyskanie poszczególnych wyników potrzebujemy tylko pozyskania na ich podstawie szans na zwycięstwo, remis oraz porażkę, których potrzebujemy do wyliczenia kursów. Jest to znacznie proste od poprzednio opisanego kroku, gdyż po prostu musimy zsumować prawdopodobieństwa na rezultaty, wliczające się do podanych kategorii. W tym celu utworzyłem trzy funkcje, które zwracają szanse na te trzy zdarzenia.

```
def get_prob_win(simulation):
    return np.sum(np.tril(simulation, -1))

def get_prob_draw(simulation):
    return np.sum(np.diag(simulation))

def get_prob_loss(simulation):
    return np.sum(np.triu(simulation, 1))
```

42 Funkcje służące do pozyskiwania poszczególnych prawdopodobieństw z listy

Jak widać na powyższym kodzie zrealizowałem funkcje dość prosto, korzystając z narzędzi oferowanych przez bibliotekę `Numpy`.

Obliczanie kursu

Po uzyskaniu prawdopodobieństw kolejnym krokiem było po prostu przerobienie ich na odpowiednie kursy. Dokonałem tego za pomocą wzoru, który opisałem we wstępie do tego rozdziału.

```
def probability_to_rate(probability):
    return 1/probability
```

43 Funkcja zmieniająca prawdopodobieństwo na kurs

Po wykonaniu powyższych kroków, dodałem możliwość uzyskiwania kursu na dany mecz. W kolejnej części rozdziału przeprowadzę uczenie dla większego zbioru danych oraz porównam i przetestuję kilka podejść do wyliczania kursów, porównując je z kursami oferowanymi przez STS.

Jak wynika z wcześniejszej części tego podrozdziału, algorytm wyliczający prawdopodobieństwo działa na podstawie tylko jednego modelu, a ja wytrenowałem aż 5. Oczywiście wynik tak naprawdę zależy tylko od liczby strzelonych bramek, ale inne statystyki też mają na niego wpływ. Z tego powodu stworzę różne warianty funkcji wyznaczającej wyniki meczów i porównam ich wyniki.

W celach testowych na początku stworzyłem osobny DataFrame zawierający tylko mecze z 35 kolejki Premier League i umieściłem w nim kursy na te mecze, oferowane przez STS dnia 02.05.2023r.

```
test_matches = upcoming_matches.drop(upcoming_matches[upcoming_matches.Round !=
'Matchweek 35'].index)
test_matches['sts_rate_win'], test_matches['sts_rate_draw'],
test_matches['sts_rate_loss'] = [
    [1.18, 2.45, 1.49, 1.82, 1.35, 2.65, 4, 2.95, 3.3, 2.04],
    [7.5, 3.45, 4.65, 3.75, 5.1, 3.3, 3.5, 3.15, 3.5, 3.4],
    [14.5, 2.7, 5.6, 4, 8.25, 2.6, 1.89, 2.43, 2.1, 3.5]
```

44 Kod tworzący DataFrame, który posłuży do testowania algorytmu

Po utworzeniu pomocniczej ramki danych, w dalszej części rozdziału stworzę kilka funkcji wyliczających dla niego kursy i porównam ich rezultaty.

Funkcja 1

Pierwsza z testowanych funkcji będzie zdecydowanie najprostsza i najbardziej podstawowa. Do wyliczenia kursu nadchodzącego meczu, użyję w niej jedynie modelu przewidującego liczbę strzelonych goli przez drużynę, która jest najbardziej wpływową statystyką z rozważanych danych.

```
def calculate_rates1(row):
    goals_simulation = simulate_match(
        goals_poisson_model,
        row['Team'],
        row['Opponent'],
        max_value=10
    )

    win_probability = get_prob_win(goals_simulation)
    draw_probability = get_prob_draw(goals_simulation)
    loss_probability = get_prob_loss(goals_simulation)

    return probability_to_rate(win_probability),
           probability_to_rate(draw_probability),
           probability_to_rate(loss_probability)
```

45 Funkcja wyliczająca kursy spotkań na podstawie liczby przewidywanych goli

Jak widać na powyższym kodzie do utworzenia wspomnianej funkcji użyłem tylko wcześniej utworzonych metod. Po jej utworzeniu zastosowałem ją dla wcześniej utworzonej ramki danych.

```
test_matches['rate_win1'], test_matches['rate_draw1'], test_matches['rate_loss1'] =
zip(*test_matches.apply(lambda row: calculate_rates1(row), axis=1))
```

46 Kod dodający stworzone przez powyższą funkcję kolumny do utworzonego DataFrame

Opponent	Team	sts_rate_win	sts_rate_draw	sts_rate_loss	rate_win1	rate_draw1	rate_loss1
Leeds	Man City	1.18	7.50	14.50	1.086910	19.429286	41.875883
Arsenal	Newcastle	2.45	3.45	2.70	1.960011	4.252881	3.926782
Brentford	Liverpool	1.49	4.65	5.60	1.655563	4.962689	5.142573
Crystal Palace	Tottenham	1.82	3.75	4.00	1.705176	4.639616	5.050301
Everton	Brighton	1.35	5.10	8.25	1.324089	6.244487	11.822200
Leicester	Fulham	2.65	3.30	2.60	1.761783	4.544764	4.709123
Chelsea	Bournemouth	4.00	3.50	1.89	2.877358	3.218538	2.926040
Aston Villa	Wolverhampton	2.95	3.15	2.43	3.722874	3.324086	2.322580
Man Utd	West Ham	3.30	3.50	2.10	3.250513	3.646993	2.391443
Southampton	Nottingham Forest	2.04	3.40	3.50	2.220999	3.446984	3.851437

47 Wybrane kolumny z testowej ramki danych po wykonaniu pierwszej funkcji wyliczającej kursy

Jak można zauważyć na powyższym fragmencie obecny algorytm działa dość dobrze, gdyż dla praktycznie wszystkich meczów dobrze wskazał faworyta oraz większość kursów wygląda sensownie.

Jedynym meczem, dla którego jako lepszą drużynę wskazał inną niż bukmacher jest spotkanie pomiędzy Chelsea oraz Bournemouth. W tym przypadku nie jest to jego błąd, lecz drużyna z Londynu (Chelsea) notuje obecnie fatalny sezon i dodatkowo przegrała ostatnie 4 mecze. STS wskazał ją na faworyta prawdopodobnie ze względu, że posiadają oni znacznie lepszych piłkarzy od zawodnika, a dodatkowo na pewno więcej ludzi na nich stawia ze względu, że są historycznie zdecydowanie lepszym klubem. Mój algorytm natomiast nie bierze tych czynników pod uwagę oraz został nauczony na danych z bieżącego sezonu, więc jego output jest jak najbardziej sensowny oraz poprawny.

W następnym w celach analitycznych za pomocą specjalnych funkcji, wyliczę procentową różnicę wartości pomiędzy moimi kursami, a oferowanymi przez bukmachera.

```
from statistics import mean

avg_win = mean(test_matches[['sts_rate_win', 'rate_win1']].pct_change(axis=1)['rate_win1'])
avg_draw = mean(test_matches[['sts_rate_draw', 'rate_draw1']].pct_change(axis=1)['rate_draw1'])
avg_loss = mean(test_matches[['sts_rate_loss', 'rate_loss1']].pct_change(axis=1)['rate_loss1'])

avg_total = (avg_win + avg_draw + avg_loss)/3

print(f'Average percentage difference of rate win: { avg_win }')
print(f'Average percentage difference of rate draw: { avg_draw }')
print(f'Average percentage difference of rate loss: { avg_loss }')
print(f'Average percentage difference of rate: { avg_total }')
```

48 Kod wyliczający procentowe różnice pomiędzy kursami dewelopera, a stworzonymi przez pierwszą funkcję

Jak wynika z powyższego fragmentu kodu, procentowe różnice utworzyłem za pomocą metody `pct.change()` oferowanej przez bibliotekę `pandas`, która wylicza po prostu procentową zmianę pomiędzy wartościami dwóch kolumn w ramce danych. Następnie zastosowałem funkcję `mean()` z biblioteki `statistics`, która wyliczyła średnią wartość z listy zawierającej procentowe różnice. Na końcu wypisałem uzyskane rezultaty oraz dodatkowo wyliczyłem ich średnią, która jest po prostu całkowitą różnicą.

```
Average percentage difference of rate win: -0.05184575054069255
Average percentage difference of rate draw: 0.27529533723048716
Average percentage difference of rate loss: 0.44978789184726464
Average percentage difference of rate: 0.22441249284568643
```

49 Średnie zmiany procentowe pomiędzy kursami bukmachera, a wygenerowanymi przez pierwszą funkcję

Jak widać różnica pomiędzy kursami wygranymi jest bardzo niska, więc prawdopodobnie zostały wyliczone dość dobrze. Problem pojawił się dla meczy zremisowanych oraz przegranych. Ponownie analizując uzyskaną ramkę z danymi można zauważyć, że wynik jest trochę zakłamanym przez kursy meczu Manchesteru City oraz Leeds. W tym spotkaniu mierzy się lider z jedną z najgorszych drużyn w lidze, przez co AI wyliczyła, że drużyna z Leeds, praktycznie nie ma w tej walce żadnych szans ustawiając ich kurs na wygraną prawie na 42.0.

STS natomiast ustawił kurs na to wydarzenie na znacznie mniejszy, ponieważ dalej jest to najwyższa klasa rozgrywkowa, przez co w teorii wszystko może się wydarzyć, a gdy kurs na jedną z drużyn jest tak niski to zniechęca graczy na obstawianie, a zbyt duży kurs na jeden z zespołów może narazić firmę na straty.

Aby sprawdzić wpływ tego meczu na resztę kursów usunąłem go z ramki danych i ponownie sprawdziłem procentowe różnice pomiędzy wynikami.

```
Average percentage difference of rate win: -0.050142192265931834
Average percentage difference of rate draw: 0.129941075523881
Average percentage difference of rate loss: 0.2914008013322805
Average percentage difference of rate: 0.12373322819674322
```

50 Średnie zmiany procentowe pomiędzy kursami bukmachera, a wygenerowanymi przez pierwszą funkcję po usunięciu meczu pomiędzy drużyną Leeds, a Manchesterem City

Jak można zauważyć wyniki wydają się teraz znacznie lepsze. Spowodowane jest to tym, że testuje dane dla małej próbki danych, więc lekka zmiana powoduje wyraźne różnice w statystyce.

Funkcja 2

W kolejnym podejściu zdecydowałem się zbadać rezultaty algorytmu korzystając z innych atrybutów podczas obliczania kursu, a następnie obliczać średnią arytmetyczną z prawdopodobieństw uzyskania lepszych rezultatów w danej statystyce.

```

def calculate_rates2(row):
    goals_simulation = simulate_match(
        goals_poisson_model,
        row['Team'],
        row['Opponent'],
        max_value=10
    )

    expected_goals_simulation = simulate_match(
        expected_goals_poisson_model,
        row['Team'],
        row['Opponent'],
        max_value=10
    )

    ...

    win_probability = (
        get_prob_win(goals_simulation) +
        get_prob_win(expected_goals_simulation) +
        get_prob_win(shots_simulation) +
        get_prob_win(shots_on_target_simulation) +
        get_prob_win(average_opponent_shot_distance_simulation)
    )/5

    ...

    return probability_to_rate(win_probability),
        probability_to_rate(draw_probability),
        probability_to_rate(loss_probability)

```

51 Fragment drugiej funkcji wyliczającej kursy

Jak widać na powyższym fragmencie kodu, umieściłem na nim tylko fragmenty tej funkcji, ponieważ symulacje dla innych atrybutów są przeprowadzane analogicznie do umieszczonych powyżej. Taka sama sytuacja dotyczy wyliczania prawdopodobieństw remisów oraz porażek.

Opponent	Team	sts_rate_win	sts_rate_draw	sts_rate_loss	rate_win2	rate_draw2	rate_loss2
Leeds	Man City	1.18	7.50	14.50	1.160928	18.870319	11.811531
Arsenal	Newcastle	2.45	3.45	2.70	1.980455	6.536954	2.923223
Brentford	Liverpool	1.49	4.65	5.60	1.501026	9.316096	4.416347
Crystal Palace	Tottenham	1.82	3.75	4.00	1.564920	7.111657	4.537745
Everton	Brighton	1.35	5.10	8.25	1.287970	11.644544	7.263301
Leicester	Fulham	2.65	3.30	2.60	1.916476	6.855925	3.008891
Chelsea	Bournemouth	4.00	3.50	1.89	2.829256	5.879937	2.098723
Aston Villa	Wolverhampton	2.95	3.15	2.43	2.446613	5.654622	2.412981
Man Utd	West Ham	3.30	3.50	2.10	2.534961	6.265923	2.242539
Southampton	Nottingham Forest	2.04	3.40	3.50	2.253291	5.623661	2.642814

52 Wybrane kolumny z testowej ramki danych po wykonaniu drugiej funkcji wyliczającej kursy

Z powyższego zrzutu ekranu wynika, że w tym przypadku kursy też są dość dobrze wyliczane, a dodatkowo nawet w meczu pomiędzy Chelsea, a Bournemouth faworyt został wyliczony zgodnie z bukmacherem. Jest to spowodowane tym, że obecny algorytm bierze pod uwagę wszystkie posiadane przeze mnie dane, w których Chelsea już nie prezentuje się aż tak źle.

Następnie ponownie przeprowadzę analizę procentowej różnicy pomiędzy kursami.

```
Average percentage difference of rate win: -0.12539194137110068
Average percentage difference of rate draw: 0.95908940031768
Average percentage difference of rate loss: -0.021560367638311772
Average percentage difference of rate: 0.2707123637694225
```

53 Średnie zmiany procentowe pomiędzy kursami bukmachera, a wygenerowanymi przez drugą funkcję

Na powyższym zrzucie ekranu rezultaty dotyczące kursów na zwycięstwa i porażki wydają się być bardzo dobre, ale problem pojawia się w przypadku rezultatów remisowych. Co więcej jest to wynik bez odrzucenia meczu zawierającego skrajny kurs.

Duża dysproporcja w jakości kursów remisowych jest spowodowana tym, że algorytm bierze pod uwagę inne parametry, dla których równa ilość występuje znacznie rzadziej. Przykładowo istnieje znacznie mniejsza szansa, że obie drużyny będą miały tyle samo oddanych strzałów niż równą liczbę strzelonych goli.

Z tego powodu to rozwiązanie nie wydaje się zbyt dobre.

Funkcja 3

Kolejne realizowane podejście będzie podobne do drugiego, lecz w tym przypadku zastosuję średnią ważoną, przez co liczba bramek będzie miała większy wpływ na rezultat

niż inne statystyki. Jako wagi zastosuję pozyskane podczas analizy wpływy poszczególnych parametrów na rezultat spotkania.

```
def calculate_rates3(row):
    goals_simulation = simulate_match(
        goals_poisson_model,
        row['Team'],
        row['Opponent'],
        max_value=10
    )

    expected_goals_simulation = simulate_match(
        expected_goals_poisson_model,
        row['Team'],
        row['Opponent'],
        max_value=10
    )

    ...

    goals_weight = 0.74
    expected_goals_weight = 0.51
    shots_on_target_weight = 0.45
    shots_weight = 0.32
    average_opponent_shot_distance_weight = 0.13

    win_probability = (
        get_prob_win(goals_simulation) * goals_weight +
        get_prob_win(expected_goals_simulation) * expected_goals_weight +
        get_prob_win(shots_simulation) * shots_weight +
        get_prob_win(shots_on_target_simulation) * shots_on_target_weight +
        get_prob_win(average_opponent_shot_distance_simulation
                     * average_opponent_shot_distance_weight)
    ) / (goals_weight + expected_goals_weight + shots_on_target_weight + shots_weight
         + average_opponent_shot_distance_weight)

    ...

    return probability_to_rate(win_probability),
           probability_to_rate(draw_probability),
           probability_to_rate(loss_probability)
```

54 Fragment trzeciej funkcji wyliczającej kursy

Jak widać na tym fragmencie kodu, podobnie jak w przypadku drugiej funkcji umieściłem tylko jej fragment, ponieważ jego druga część działa w analogiczny sposób.

Opponent	Team	sts_rate_win	sts_rate_draw	sts_rate_loss	rate_win3	rate_draw3	rate_loss3
Leeds	Man City	1.18	7.50	14.50	1.116008	17.831712	21.629646
Arsenal	Newcastle	2.45	3.45	2.70	1.992934	5.462454	3.173011
Brentford	Liverpool	1.49	4.65	5.60	1.455612	7.376585	5.636339
Crystal Palace	Tottenham	1.82	3.75	4.00	1.604486	5.859347	4.852552
Everton	Brighton	1.35	5.10	8.25	1.243841	9.512404	11.004055
Leicester	Fulham	2.65	3.30	2.60	1.889870	5.699111	3.385299
Chelsea	Bournemouth	4.00	3.50	1.89	3.015149	4.670992	2.201411
Aston Villa	Wolverhampton	2.95	3.15	2.43	2.745056	4.571822	2.398211
Man Utd	West Ham	3.30	3.50	2.10	2.848832	5.082111	2.211362
Southampton	Nottingham Forest	2.04	3.40	3.50	2.336391	4.585326	2.825638

55 Wybrane kolumny z testowej ramki danych po wykonaniu trzeciej funkcji wyliczającej kursy

W tym przypadku również algorytm poprawnie określił wszystkich faworytów spotkań, lecz jak można zauważyć kursy remisów ponownie są zawyżone.

```
Average percentage difference of rate win: -0.10548731419896486
Average percentage difference of rate draw: 0.6288515320188218
Average percentage difference of rate loss: 0.15344196217843617
Average percentage difference of rate: 0.22560205999943106
```

56 Średnie zmiany procentowe pomiędzy kursami bukmachera, a wygenerowanymi przez trzecią funkcję

Po uruchomieniu kodu wyliczającego procentowe różnice w kursach otrzymałem powyższy rezultat. Jak widać średnia różnica jest praktycznie taka sama jak w przypadku pierwszego algorytmu (bez usunięcia żadnego meczu). Mimo takiej samej średniej w tym przypadku kursy przegranych zostały wyliczone w znacznie lepszy sposób, lecz z podobnego powodu co w drugim przypadku kursy remisów znowu nie wydają się satysfakcjonujące.

Funkcja 4

Na podstawie wniosków z rezultatów poprzednich funkcji narzuca się rozwiązanie będące hybrydą pierwszego i trzeciego rozwiązania. Gdzie pierwsze z nich lepiej radzi sobie z remisami, a ostatnie działa dobrze dla zwycięstw oraz porażek.

Aby to zrobić zmodyfikowałem napisane wcześniej metody wyliczające prawdopodobieństwa z uzyskanych symulacji, aby przyjmowały dodatkowy parametr „shift”, który definiuje rozszerzenie danych uznawanych również za remis. Taka zmiana może rozwiązać obecny problem, ponieważ wcześniej na przykład w przypadku liczby strzałów,

tylko w przypadku ich równej wartości oznaczały remis, a rezultat w którym jeden zespół miał o tylko jeden strzał, więcej oznaczał już jego triumf.

Z tego powodu w ulepszonej wersji algorytmu, po ustawieniu parametru „Shift” na przykład na wartość 1, to w przypadku różnicy tylko jednego strzału algorytm pokaże dalej remis, co w niektórych przypadkach jest znacznie sensowniejszym rozwiązaniem.

```
def get_prob_win(simulation, shift = 0):  
    return np.sum(np.tril(simulation, -1 - shift))  
  
def get_prob_draw(simulation, shift = 0):  
    sum = 0  
    for x in range(-shift, shift + 1):  
        sum += np.sum(np.diag(simulation, x))  
    return sum  
  
def get_prob_loss(simulation, shift = 0):  
    return np.sum(np.triu(simulation, 1 + shift))
```

57 Zmodyfikowane metody odpowiedzialne za pozyskiwanie kursów zdarzeń z symulacji

Na powyższym rzucie, zmodyfikowanie tych metod było dosyć proste, ponieważ opierało się na pobieraniu dodatkowych wartości w odległości zdefiniowanego przesunięcia od przekątnej dla remisów, oraz analogicznie nie pobieranie ich dla wygranych oraz przegranych.

W dalszej kolejności utworzyłem nową funkcję, która jest trzecią funkcją wzbogaconą o dodanie przesunięć do funkcji wyliczających prawdopodobieństwa zdarzeń. Wartości tych przesunięć ustawiłem na podstawie wcześniej przeprowadzonej analizy rozkładu wartości statystyk. Oczywiście ustawione wartości nie są optymalne, ale nie będę się zajmował ich dokładnym ustawianiem, ponieważ jest to niemożliwe bez większego zbioru danych.

```

def calculate_rates2(row):
    goals_simulation = simulate_match(
        goals_poisson_model,
        row['Team'],
        row['Opponent'],
        max_value=10
    )

    expected_goals_simulation = simulate_match(
        expected_goals_poisson_model,
        row['Team'],
        row['Opponent'],
        max_value=10
    )

    ...

    goals_weight = 0.74
    expected_goals_weight = 0.51
    shots_on_target_weight = 0.45
    shots_weight = 0.32
    average_opponent_shot_distance_weight = 0.13

    win_probability = (
        get_prob_win(goals_simulation) * goals_weight +
        get_prob_win(expected_goals_simulation) * expected_goals_weight +
        get_prob_win(shots_simulation, 3) * shots_weight +
        get_prob_win(shots_on_target_simulation, 1) * shots_on_target_weight +
        get_prob_win(average_opponent_shot_distance_simulation, 1)
        * average_opponent_shot_distance_weight
    ) / (goals_weight + expected_goals_weight + shots_on_target_weight + shots_weight
        + average_opponent_shot_distance_weight)

    ...

    return probability_to_rate(win_probability),
        probability_to_rate(draw_probability),
        probability_to_rate(loss_probability)

```

58 Fragment czwartej funkcji wyliczającej kursy

Jak można zauważyć czwarta funkcja jest praktycznie identyczna co trzecia i jedyną różnicą są dodane w niektórych miejscach wcześniej wspomniane przesunięcia. W dalszej kolejności sprawdzę w jaki sposób ta zmiana wpłynęła na jakość algorytmu.

Opponent	Team	sts_rate_win	sts_rate_draw	sts_rate_loss	rate_win4	rate_draw4	rate_loss4
Leeds	Man City	1.18	7.50	14.50	1.138652	12.370409	25.448989
Arsenal	Newcastle	2.45	3.45	2.70	2.250160	3.427275	3.790620
Brentford	Liverpool	1.49	4.65	5.60	1.501432	5.959388	6.018759
Crystal Palace	Tottenham	1.82	3.75	4.00	1.748551	3.915314	5.790795
Everton	Brighton	1.35	5.10	8.25	1.275515	7.438207	12.266207
Leicester	Fulham	2.65	3.30	2.60	2.103600	3.662800	3.974454
Chelsea	Bournemouth	4.00	3.50	1.89	3.464970	3.224141	2.492292
Aston Villa	Wolverhampton	2.95	3.15	2.43	3.259232	3.001831	2.777398
Man Utd	West Ham	3.30	3.50	2.10	3.291162	3.349593	2.515015
Southampton	Nottingham Forest	2.04	3.40	3.50	2.693696	2.979360	3.411569

59 Wybrane kolumny z testowej ramki danych po wykonaniu czwartej funkcji wyliczającej kursy

Jak widać nowo stworzona funkcja działa całkiem nieźle i również dobrze wytypowała każdego z faworytów. Jedyną rzeczą, która nie wygląda najlepiej jest ponownie duża różnica pomiędzy kursami pierwszego meczu. Mimo takiej dysproporcji jak wspominałem podczas badania pierwszego algorytmu, nie musi to być błąd, lecz po prostu zaniżenie kursu przez bukmachera.

```
Average percentage difference of rate win: -0.012072749556229844
Average percentage difference of rate draw: 0.12443393134282946
Average percentage difference of rate loss: 0.3330957593925618
Average percentage difference of rate: 0.14848564705972048
```

60 Średnie zmiany procentowe pomiędzy kursami bukmachera, a wygenerowanymi przez czwartą funkcję

Na powyższym rzucie ekranu tym razem średnie różnice pomiędzy kursami wyglądają bardzo dobrze nawet bez usunięcia pierwszego meczu. Z tego powodu uważam, że algorytm w takiej postaci spełnia moje oczekiwania i użyję go w finalnej wersji projektu.

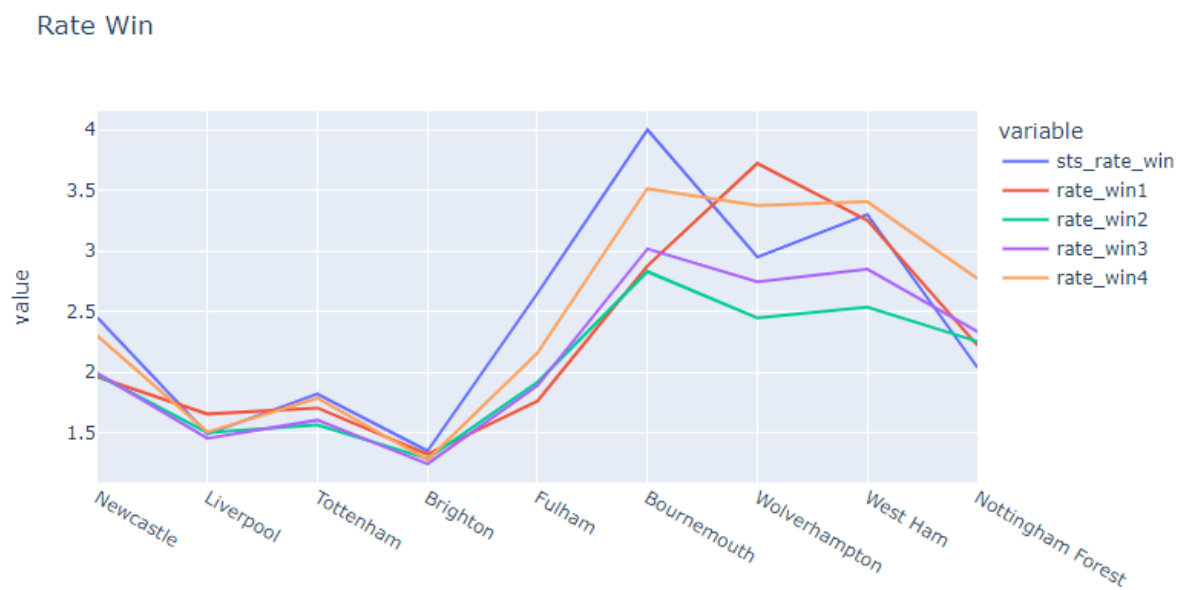
Na końcu w celu potwierdzenia tego wniosku zestawię wyniki wszystkich algorytmów za pomocą wykresów. Do ich utworzenia ponownie zastosuję bibliotekę Plotly.

```
fig = px.line(test_matches, x="Team", y=["sts_rate_win", "rate_win1", "rate_win2",
"rate_win3", "rate_win4"], title="Rate Win")
fig.show()

fig = px.line(test_matches, x="Team", y=["sts_rate_draw", "rate_draw1",
"rate_draw2", "rate_draw3", "rate_draw4"], title="Rate Draw")
fig.show()

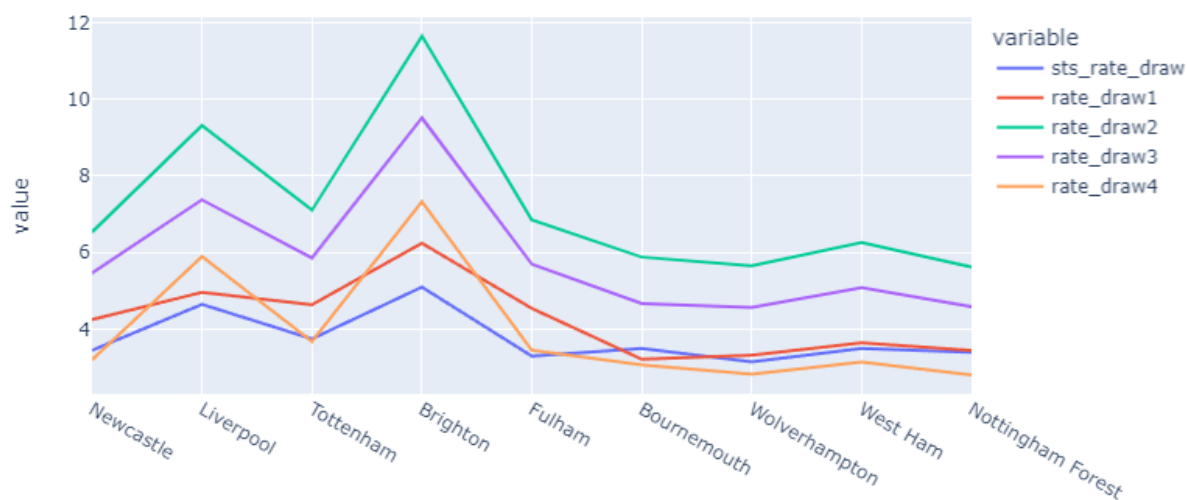
fig = px.line(test_matches, x="Team", y=["sts_rate_loss", "rate_loss1",
"rate_loss2", "rate_loss3", 'rate_loss4'], title="Rate Loss")
fig.show()
```

61 Kod odpowiedzialny za stworzenie wykresów zawierających porównania uzyskanych kursów



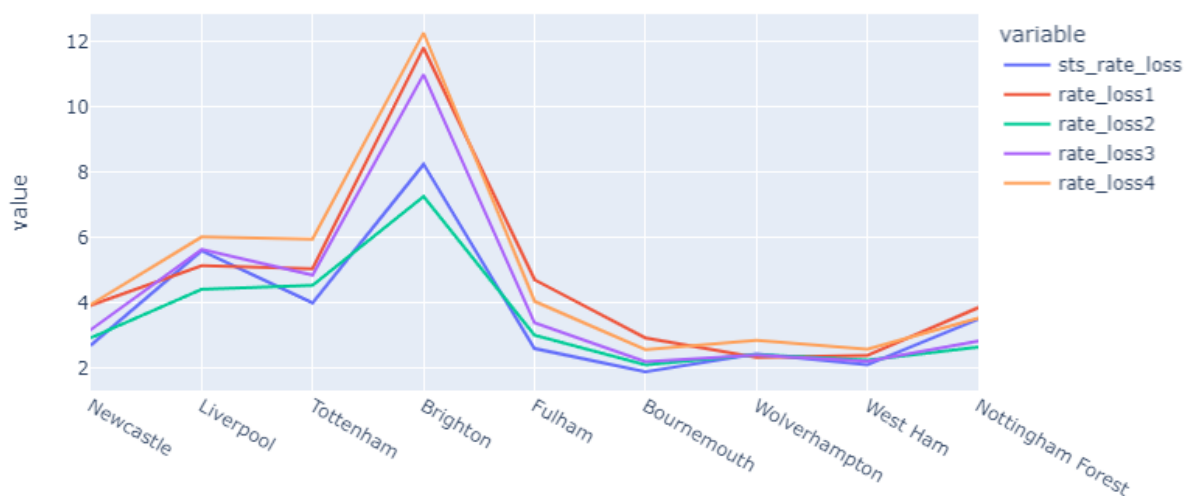
62 Wykres zawierający kursy zwycięstw

Rate Draw



63 Wykres zawierający kursy remisów

Rate Loss



64 Wykres zawierający kursy porażek

Za pomocą biblioteki Plotly uzyskałem powyższe wykresy, które zawierają porównanie wygenerowanych kursów. Na osi Y wykresu, umieszczone są drużyny gospodarzy z badanych meczy.

Dzięki tym wykresom można również potwierdzić, że ostatnia funkcja działa zdecydowanie najlepiej, gdyż wartości wygenerowanych przez nią kursów (kolor żółty), znajdują się najbliżej

niebieskiej linii, czyli kursów oferowanych przez bukmachera. Jedynym wyjątkiem jest wykres kursów przegranych, lecz na nim dla każdego podejścia wyniki są dość podobne.

Oczywiście ten algorytm można ulepszyć dostrajając wartości przesunięć oraz inne parametry, ale na obecna jego forma dobrze spełnia moje oczekiwania, dlatego zostanie zastosowana w dalszej części projektu.

Z tego powodu wyliczyłem wartości kursów dla wszystkich wartości ramki danych zawierających nadchodzące mecze i dodałem je do odpowiednich kolumn.

```
upcoming_matches['rate_win'], upcoming_matches['rate_draw'],  
upcoming_matches['rate_loss'] = zip(*upcoming_matches.apply(lambda row:  
    calculate_rates4(row), axis=1))
```

65 Kod dodający kolumny z kursami do DataFrame zawierającego nadchodzące mecze

Możliwości ulepszenia algorytmu

Oczywiście jak wcześniej wspominałem wyuczony przeze mnie model jest daleki od idealnego i istnieje wiele czynności mogących potencjalnie ulepszyć jego skuteczność.

Pierwszą z nich jest pobranie większej ilości historycznych meczy, dzięki czemu uczenie będzie pełniejsze. Obecna wersja zawiera tylko dane z sezonu 2022/2023, przez co pozyskanie danych na przykład z ostatnich pięciu sezonów mogło by w jakimś stopniu udoskonalić algorytm.

Kolejną rzeczą jest sprawdzenie innych rozkładów prawdopodobieństwa oraz innych bibliotek do nauczania maszynowego. Jak opisałem w tym rozdziale w moim projekcie zastosowałem rozkład Poissona zaimplementowany przez bibliotekę Statsmodels. Mimo że ten algorytm działa dość dobrze istnieje szansa, że dla mojego przypadku inny może działać lepiej. Z tego powodu potencjalna jego zmiana może polepszyć skuteczność algorytmu.

Innym działaniem, które prawdopodobnie ulepszyło by algorytm jest pobranie większej ilości rodzajów statystyk dotyczących meczów. Przykładowo dane takie jak posiadanie piłki oraz liczba czerwonych kartek, również mogły by ulepszyć mój program. Dodatkowo pobranie innych danych tekstowych takich jak sędzia prowadzący mecz, również byłoby korzystne dla algorytmu, gdyż analiza byłaby pełniejsza.

Dodatkowo bukmacherzy stosują jeszcze inne statystyki poza meczowe do ustawiania kursów takie jak zainteresowanie meczem, jego istotność oraz statystyki dotyczące osób obstawiających zakłady. Mają one również istotny wpływ na kurs meczu, lecz w moim przypadku ich zastosowanie było raczej trudne.

Podsumowując ten rozdział utworzony przeze mnie algorytm jest dosyć dobry i spełnia założone przeze mnie oczekiwania, lecz posiada jeszcze sporo miejsca na potencjalne udoskonalenia.

3. Baza danych

W tej części mojej pracy zajmę się utworzeniem bazy danych, w której będę przechowywać potrzebne informacje do działania aplikacji oraz przedstawię sposób w jaki komunikowałem się między nią, a moją aplikacją.

3.1 Wybór technologii

Do przechowywania danych w aplikacji rozważyłem dwa rozwiązania klasyczną bazę używającą SQL o nazwie MySQL oraz oferowaną przez firmę Google Firebase.

MySQL

Jest to najpopularniejszy na świecie system zarządzania bazą danych oferowany przez firmę Oracle. Dzięki niemu mógłbym bez problemu przechowywać wszystkie niezbędne przeze mnie dane, a dodatkowo jest on w stu procentach kompatybilny ze wszystkimi użytymi przeze mnie rozwiązaniami, więc jego implementacja nie sprawiła by mi dużego problemu. (vavatech)

Dodatkowo dużą zaletą tego narzędzia jest to, że jest ono całkowicie darmowe i nie nakłada ono żadnych dodatkowych ograniczeń takich jak limit zapytań.

Wdrożenie tego rozwiązania w moim przypadku opierało by się na utworzeniu kontenera na podstawie obrazu zawierającego MySQL za pomocą narzędzia Docker, a następnie dodaniu kodu odpowiedzialnego za komunikację pomiędzy moją aplikacją, a bazą danych.

Firebase

Jest to w porównaniu do wyżej opisanej baza NoSQL, która jak sama nazwa mówi nie jest oparta na tym języku programowania. Z tego powodu posiada również inny system przechowywania danych. W porównaniu do rozwiązań relacyjnych, nie przechowuje ona danych w postaci tabel, lecz korzysta ono z magazynów klucz-wartość.

Tak samo jak jej poprzednik jest ona dobrze ocenianym rozwiązaniem przez użytkowników i jego implementacja jest również bardzo wygodna w przypadku mojego projektu.

Dodatkowo w przypadku tego narzędzia, firma Google oferuje swoim użytkownikom wiele przydatnych dodatkowych funkcji, które byłyby przydatne w przypadku mojego projektu. Pierwszą z nich jest usługa pozwalająca na zrealizowanie w łatwy sposób systemu

autentykacji użytkownika. Dzięki tej funkcji wystarczy powiązać swoją aplikację z projektem utworzonym na stronie usługi oraz dodać kilka linii kodu aby stworzyć cały system użytkowników w systemie.

Innym atutem tego rozwiązania jest to, że oparte jest ono na chmurze, przez co nie musimy przechowywać danych w lokalnym środowisku ani kupować dodatkowego serwera. Nasze dane są po prostu przechowywane na serwerze udostępnionym przez wydawcę Firebase. Minusem tego podejścia jest niestety to, że w tym przypadku jesteśmy uzależnieni od firmy Google, a dodatkowo to narzędzie narzuca nam ograniczenia zapytań w darmowej wersji. (Google)

Porównanie rozważanych technologii

Jak wspomniałem we wcześniejszej części tego rozdziału oba rozwiązania spełniają moje wymagania, lecz posiadają wiele różnic pomiędzy sobą.

Pierwszą z nich jest to, że baza MySQL pozwala nam przechowywać bardziej złożone dane ze względu na jej relacyjność. W bazie typu klucz-wartość, raczej przechowuje się prostsze dane, lecz w moim przypadku jest ona jak najbardziej wystarczająca

Kolejną istotną różnicą jest to, że Firebase oferuje deweloperom znacznie więcej gotowych rozwiązań przez co implementacja bazy danych jest znacznie prostsza. Dodatkowo oferuje on dodatkowe funkcjonalności o których wspomniałem wcześniej, które również ułatwiają programiście prace. Mimo że w teorii jest to zaleta, to te właściwości czynią naszą bazę mniej elastyczną niż rozwiązanie firmy Oracle, które jest oferowane na zasadzie Open Source. Dodatkowo korzystając z bazy MySQL jesteśmy w stu procentach niezależni, dlatego rekomendowałbym to podejście dla większych projektów. (Dearmer, 2023)

Te dwa rozwiązania oczywiście posiadają jeszcze wiele różnic, które mają mniejszy wpływ dla mojego projektu.

Końcowy wybór technologii

Na podstawie powyższego porównania zdecydowałem się zastosować bazę Firebase. Uznałem, że to rozwiązanie będzie lepsze dla mojego projektu, ponieważ pozwala ono na znacznie łatwiejszą implementację potrzebnych mi funkcjonalności.

Dodatkowo w ramach mojej pracy dyplomowej nie tworzę aplikacji, która ma być w pełni funkcjonalna i wypuszczona jako produkt, przez co minusy bazy Firebase nie mają dla mnie większego znaczenia.

Ostatnim czynnikiem, który przeważał na korzyść rozwiązania oferowanego przez Google jest to, że wydaje się ono znacznie ciekawsze oraz innowacyjne, a o bazie MySQL słyszałem

prawdopodobnie każdy. Co więcej w mojej pracy inżynierskiej zastosowałem właśnie rozwiązania relacyjnego, więc tym razem poszedłem w inną stronę.

Stworzenie bazy danych

Aby stworzyć bazę danych najpierw zalogowałem się za pomocą swojego konta Google na stronie <https://firebase.google.com/>, a następnie utworzyłem tam mój projekt. W dalszej kolejności w zakładce Build, wybrałem opcje Realtime Database i utworzyłem bazę danych w moim projekcie.

Podczas tworzenia bazy w jej opcjach konfiguracyjnych jako lokalizację wybrałem Stany Zjednoczone, ponieważ wybranie tej opcji jest ze względów na stabilność, bardziej rekomendowane przez społeczność.

Po wykonaniu tych prostych kroków moja baza tak naprawdę była gotowa do przechowywania danych.

3.2 Zapisanie nadchodzących meczów w bazie danych

Na początku postanowiłem zapisać pozyskane dane w poprzednich rozdziałach w bazie danych, aby można było mieć do nich łatwy dostęp w aplikacji mobilnej, której tworzeniem zajmę się w ostatnim rozdziale mojej pracy dyplomowej.

Przygotowanie danych do zapisania

Aby zrealizować tę funkcjonalność musiałem w pierwszej kolejności zmienić format ramki danych tak aby były gotowe do zapisania ich w bazie Firebase.

```

upcoming_matches.drop(upcoming_matches[upcoming_matches.Venue != 'Home'].index,
inplace=True)

upcoming_matches.drop(['Venue', 'Competition'], axis=1, inplace=True)
upcoming_matches.rename(columns={"Round": "round", "Team": "team1",
                                "Opponent": "team2", "DateTime": "dateTime", "rate_win": "rateWin",
                                "rate_draw": "rateDraw", "rate_loss": "rateLoss"}, inplace=True)

upcoming_matches = upcoming_matches.round({'rate_win':2, 'rate_draw':2,
'rate_loss':2})

upcoming_matches['key'] = upcoming_matches['team1'] + '-' +
    upcoming_matches['dateTime'].astype(str)

upcoming_matches.set_index('key', inplace=True)

```

66 Kod odpowiedzialny za przygotowanie danych do zapisania w bazie danych

Jak widać na powyższym kodzie, przed zapisaniem danych do bazy lekko je zmodyfikowałem.

Na początku usunąłem wszystkie dane, w których opisywana drużyna nie była gospodarzem spotkania. W przypadku braku zastosowania tej zmiany, każdy mecz zostałby zapisany do bazy dwukrotnie, gdyż w poprzedniej wersji każde spotkanie było zawarte w DataFrame dwukrotnie, jako statystyki drużyny gospodarzy oraz gości.

W kolejnym kroku pozbyłem się zbędnych kolumn, których nie wykorzystam podczas tworzenia aplikacji. W moim przypadku nie potrzebowałem kolumn zawierających informację o rundzie oraz rozgrywkach, a dodatkowo kolumna, która określa gospodarza również okazała się zbędna gdyż po poprzedniej zmianie zawsze wartości „Team” określały gospodarza.

W dalszej kolejności zmieniłem nazwy użytych zmiennych aby były zgodne z dobrymi praktykami przechowywania danych w bazie Firebase oraz zaokrągliłem wartości kursów do dwóch miejsc po przecinku, ponieważ nie potrzebowałem przechowywać ich w aż tak dokładnej postaci.

Ostatnim krokiem było stworzenie zmiennej „key”, będącej kluczem definiującym dany mecz. Oczywiście w przypadku nie utworzenia tej zmiennej, baza sama by wygenerowała klucze, lecz wolałem stworzyć własne aby móc w łatwiejszy sposób manipulować danymi.

```
upcoming_matches.sample(5)
```

✓ 0.0s

	round	team2	team1	dateTime	rateWin	rateDraw	rateLoss
key							
Bournemouth-2023-05-20 15:00:00	Matchweek 37	Man Utd	Bournemouth	2023-05-20 15:00:00	5.08	3.94	1.82
Brentford-2023-05-28 16:30:00	Matchweek 38	Man City	Brentford	2023-05-28 16:30:00	6.76	5.48	1.49
Tottenham-2023-05-20 12:30:00	Matchweek 37	Brentford	Tottenham	2023-05-20 12:30:00	1.76	4.58	4.68
West Ham-2023-05-21 13:30:00	Matchweek 37	Leeds	West Ham	2023-05-21 13:30:00	2.00	3.43	4.77
Everton-2023-05-14 14:00:00	Matchweek 36	Man City	Everton	2023-05-14 14:00:00	13.60	6.30	1.30

67 Próba pięciu danych o nadchodzących spotkaniach gotowa do zapisania w bazie

Jak widać na powyższym zrzucie ekranu po wykonaniu wcześniejszych modyfikacji, dane w obecnej postaci zawierają tylko 6 kolumn, zawierających klucz, rundę rozgrywek, informacje o drużynach oraz kursy dotyczące opisywanego meczu.

Dane w takiej postaci są gotowe do zapisania ich w naszej bazie danych.

Zapisanie danych do Firebase

Do zapisania danych w bazie użyłem oferowanej przez producenta biblioteki `firebase_admin`, zawierające wszystkie niezbędne funkcję potrzebne do zrealizowania tej funkcjonalności.

```
import firebase_admin
from firebase_admin import credentials, db

cred = credentials.Certificate("betapp-7a0bb-firebase-adminsdk-k1jpq-2d03a6fdcc.json")
default_app = firebase_admin.initialize_app(cred)

ref = db.reference("/games", url = 'https://betapp-7a0bb-default-rtdb.firebaseio.com/')

data = upcoming_matches.to_dict(orient='index')
for rec in data:
    data[rec]['dateTime'] = str(data[rec]['dateTime'])

ref.set(data)
```

68 Kod zapisujący dane do Firebase

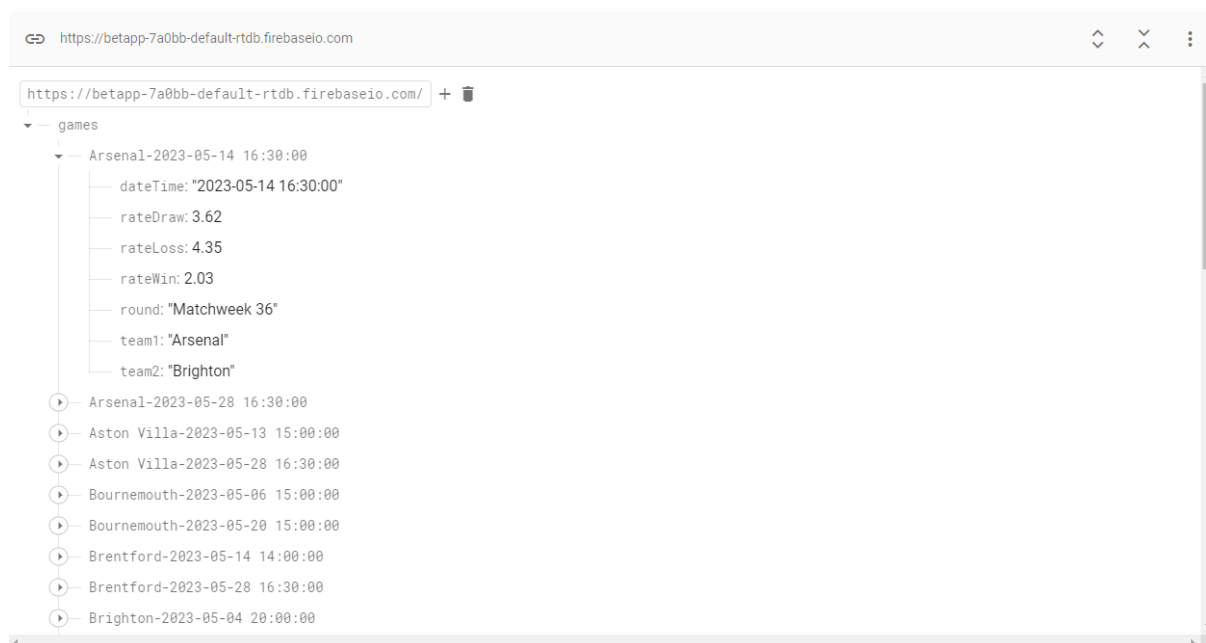
Jak wcześniej wspominałem dzięki oferowanym przez Google usługom, realizacja tej funkcjonalności nie wymagała zbyt wielu linii kodu.

W pierwszej części kodu połączyłem się z projektem utworzonym na stronie Firebase za pomocą wygenerowanego pliku z rozszerzeniem .json zawierającym wszystkie niezbędne do tego parametry.

W dalszej kolejności stworzyłem referencje do bazy, w której dane mają być zapisane. Aby to zrealizować użyłem metody `db.reference()`, podając jako jej parametry link do projektu oraz ścieżkę pod którą mają być przechowywane moje dane.

Następnie przekonwertowałem ramkę danych na typ Dictionary, ponieważ takiego typu oczekuje użyta przeze mnie biblioteka do zapisu danych. Dodatkowo musiałem zmienić typ danych o dotyczących daty meczu na tekstowy, ponieważ używany wcześniej typ nie był obsługiwany przez Firebase.

Na końcu za pomocą metody `set()` dodałem informacje o wszystkich meczach do bazy danych. W tym celu mogłem użyć również `push()`, lecz tamta metoda powoduje po prostu dodanie danych do bazy, a w przypadku metody `set()`, w przypadku gdy dane o tym samym id znajdują się już w bazie to są nadpisywane. W przeciwnym wypadku użyta przeze mnie metoda po prostu dodaje dane.



69 Zrzut ekranu ze strony Firebase, zawierający dane w mojej bazie

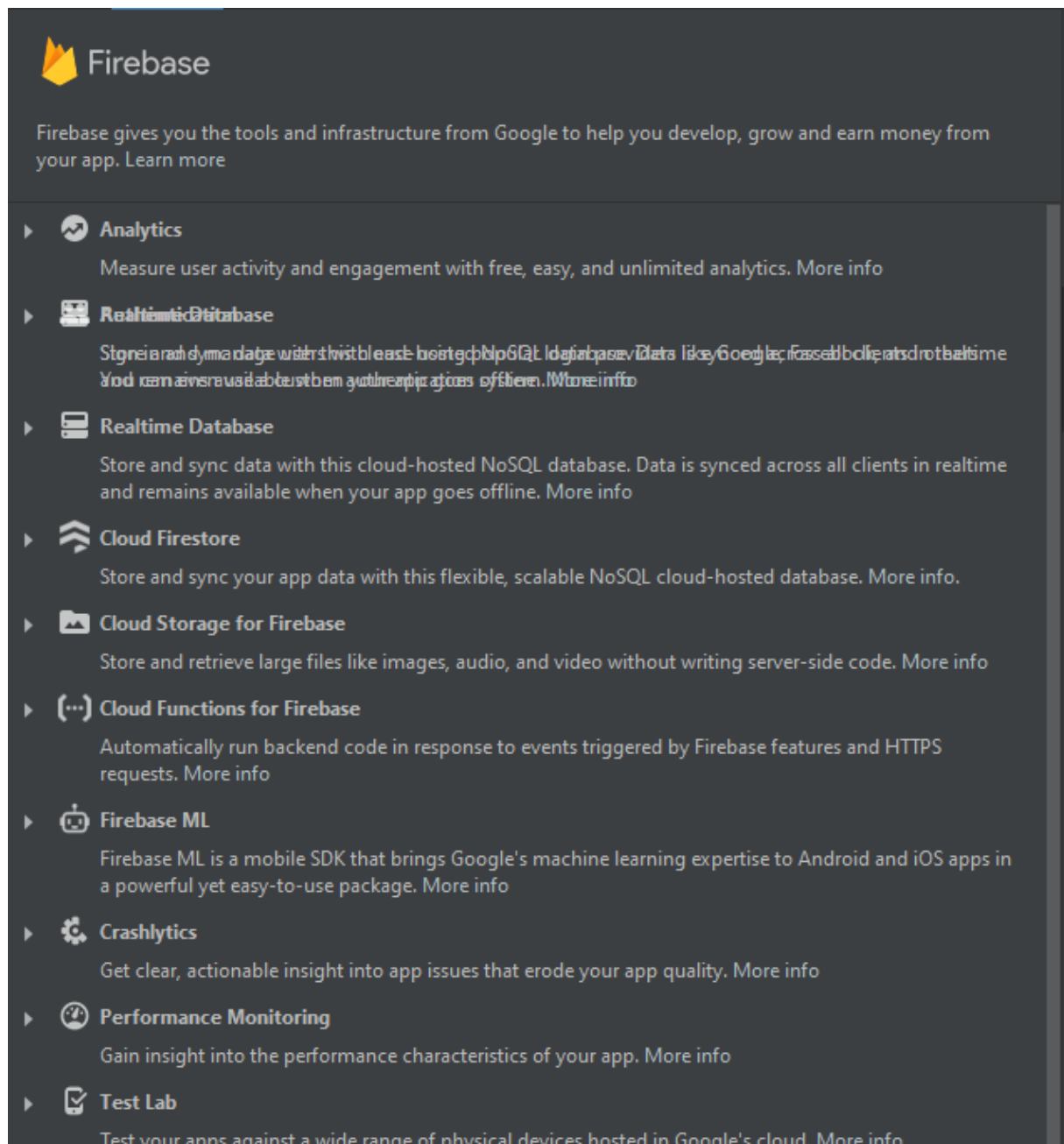
Na powyższym zrzucie ekranu znajduje się fragment panelu administracyjnego oferowanego przez Firebase, dzięki któremu w łatwy sposób mogłem zweryfikować czy dane zostały dodane we właściwy sposób.

Jak widać w bazie danych znajdują się wszystkie potrzebne informacje, więc mogę stwierdzić, że ich dodanie przebiegło pomyślnie.

3.3 Komunikacja aplikacji mobilnej z bazą danych

Skonfigurowanie bazy danych dla mojej aplikacji mobilnej było dosyć łatwe oraz szybkie, ze względu na to, że Android Studio posiada opcje, która praktycznie sama łączy aplikację z Firebase oraz prowadzi programistę krok po kroku w realizacji tej funkcjonalności.

Ta opcja jest dostępna w zakładce Tools/Firebase. Po wybraniu tej opcji użytkownik otrzymuje asystenta pomagającego mu w konfiguracji tego narzędzia.



70 Fragment asystenta Firebase oferowanego przez Android Studio

Po uruchomieniu wspomnianego powyżej asystenta, wybrałem opcję „Get started with Realtime Database”, znajdującą się w sekcji Realtime Database. Następnie zgodnie z instrukcją połączyłem moją aplikację z wcześniej utworzonym projektem oraz asystent automatycznie zaimportował wszystkie potrzebne zależności w moim projekcie.

Zapisywanie oraz wczytywanie zakładów w bazie danych

Komunikacje utworzonej aplikacji z bazą danych przedstawię na przykładzie zapisywanych zakładów. Na potrzeby mojego oprogramowania przechowywałem również dane użytkownika oraz informacje o odbytych meczach, ale ze względu że kod odpowiedzialny za tę funkcjonalność jest analogiczny do tego, który opiszę w tym podrozdziale, postanowiłem go pominąć w części pisemnej pracy.

Klasa Bet

W pierwszej kolejności utworzyłem klasę typu data w języku Kotlin, która zawiera wszystkie atrybuty, które zawierają zakłady.

```
data class Bet(  
    var id: String,  
    var gameId: String,  
    var rate: Double,  
    var result: Result,  
    var input: Double,  
    var settled: Boolean,  
    var output: Double  
)
```

71 Kod klasy Bet

Jak widać na powyższym fragmencie kodu każdy obiekt tego typu zawiera:

- id – unikalny identyfikator zakładu w bazie danych
- gameId – unikalny identyfikator meczu, którego dotyczy zakład
- rate – kurs, z którym użytkownik obstawił mecz
- result – rezultat, na który użytkownik obstawił mecz
- input – kwota wpłacona na zakład
- settled – atrybut typu Boolean, definiujący czy mecz został rozliczony
- output – liczba wygranych punktów w zakładzie (w przypadku przegranej wynosi 0)

Repozytorium

Kolejnym krokiem, który zrealizowałem było utworzenie repozytorium, będącym bezpośrednim połączeniem mojej aplikacji z użytą bazą danych.

Utworzona klasa jako argumenty przyjmuje instancje mojej bazy danych oraz obiekt definiujący, którego dotyczą przetwarzane dane. W ramach repozytorium utworzyłem również atrybut `allBets`, zawierający dane o wszystkich zakładach zalogowanego użytkownika. Ten atrybut opakowałem w klasę `MutableLiveData`, dzięki której będę mógł obserwować zmiany dokonane na tych obiektach podczas cyklu działania mojej aplikacji.

```
val allBets: MutableLiveData<HashMap<String, Bet>> =  
    MutableLiveData<HashMap<String, Bet>>().also{  
        it.value = HashMap<String, Bet>()  
    }
```

72 Utworzenie obiektu `allBets`

Następnie utworzyłem blok inicjalizacyjny, w którym odwołałem się do referencji do zakładów wybranego użytkownika, w ramach której utworzyłem listener, w którym przeciążyłem kilka metod definiujących akcje, w przypadku różnych operacji na zdefiniowanych obiektach.

Przykładowo w ramach przeciążenia metody wywoływanej po dodaniu nowego obiektu do bazy danych (`onChildAdded`), dodałem kod tworzący nowy obiekt typu `Bet` w mojej aplikacji oraz dodałem go do wcześniej zdefiniowanej listy.

```
override fun onChildAdded(snapshot: DataSnapshot, previousChildName: String?) {  
    val bet = Bet(  
        id = snapshot.ref.key as String,  
        gameId = snapshot.child("gameId").getValue(String::class.java)!!,  
        rate = snapshot.child("rate").getValue(Double::class.java)!!,  
        result = snapshot.child("result").getValue(Result::class.java)!!,  
        input = snapshot.child("input").getValue(Double::class.java)!!,  
        settled = snapshot.child("settled").getValue(Boolean::class.java)!!,  
        output = snapshot.child("output").getValue(Double::class.java)!!,  
    )  
    allBets.value?.put(bet.id, bet)  
    allBets.postValue(allBets.value)  
}
```

73 Przeciążona metoda `onChildAdded()`

W ramach tego repozytorium utworzyłem, również metodę `insert`, dzięki której będę mógł dodawać nowe zakłady do bazy danych, za pomocą odwołania się do odpowiedniej referencji. W tego typu projektach analogicznie można utworzyć również inne metody tego typu, takie jak `update`, lecz w moim przypadku podczas korzystania z aplikacji użytkownik

będzie mógł tylko dodawać oraz wyświetlać swoje zakłady, więc definiowanie innych funkcji było zbędne.

```
fun insert(bet: Bet) {
    fbdb.getReference("bets/" + user.uid).push().also{
        bet.id = it.ref.key.toString()
        it.setValue(bet)
    }
}
```

74 Funkcja insert, pozwalająca na dodanie nowego zakładu do bazy danych

ViewModel

Ostatnim typem klas służących do komunikacji z bazą jest model widoku. Jest on w skrócie połączeniem pomiędzy utworzonymi przeze mnie aktywnościami z opisanym powyżej repozytorium.

Zdecydowałem się na utworzenie tej klasy aby nie odwoływać się bezpośrednio do repozytorium z poziomu widoku, tylko do pośredniczącej klasy zawierającej tylko niezbędne dla mnie atrybuty oraz metody. Dodatkowo jest to po prostu dobra praktyka programistyczna.

```
class BetViewModel(application: Application) : AndroidViewModel(application) {
    private val repository: BetRepository
    private var firebaseDatabase: FirebaseDatabase =
        FirebaseDatabase.getInstance()
    private var user: FirebaseUser = FirebaseAuth.getInstance().currentUser!!
    val allBets: MutableLiveData<HashMap<String, Bet>>

    init{
        repository = BetRepository(firebaseDatabase, user)
        allBets = repository.allBets
    }

    suspend fun insert(bet: Bet) = repository.insert(bet)
}
```

75 Kod klasy BetViewModel

Jak widać na powyższym fragmencie kodu wewnątrz klasy tego typu, po prostu tworzę repozytorium oraz przypisuje jego zawartość do nowych zmiennych oraz funkcji, które będą mi potrzebne podczas tworzenia widoku.



76 Zrzut ekranu z bazy Firebase, zawierający utworzone zakłady

Po dodaniu nowego zakładu, poprzez utworzoną przeze mnie aplikację mobilną, dane zapisywane są w bazie Firebase w powyższej postaci. W tym przypadku w porównaniu do meczów, te wartości są przypisywane bezpośrednio do użytkowników, więc dane o zakładach nie są przypisane bezpośrednio do klucza bets, a zawierają pośrednio identyfikatory użytkowników, do których należą.

3.4 Rozliczanie zakładów

Ostatnią funkcjonalnością związaną z bazą danych było rozliczanie zakładów, których powiązane mecze dobiegły końca. Zrealizowałem to poprzez napisanie prostego skryptu w języku Python.

Do utworzenia tego algorytmu oczywiście potrzebowałem listy zakończonych meczów, zawierającą ich wyniki. Uzyskanie takiej listy byłoby identyczne jak w przypadku, zbioru danych który pobrałem na potrzeby uczenia maszynowego w pierwszym rozdziale mojej pracy. Z tego powodu wykorzystałem wcześniej utworzoną listę, do której dodałem tylko pole zawierające identyfikator spotkania w celu łatwiejszej identyfikacji danego wydarzenia.

```
match_stats['Key'] = match_stats['Team'] + '-' + match_stats['DateTime'].astype(str)
match_results = match_stats[['Key', 'Result']]
match_results
```

✓ 0.0s

	Key	Result
1	Man City-2022-08-07 16:30:00	W
2	Man City-2022-08-13 15:00:00	W
3	Man City-2022-08-21 16:30:00	D
4	Man City-2022-08-27 15:00:00	W
5	Man City-2022-08-31 19:30:00	W
...
941	Southampton-2023-04-30 14:00:00	L
942	Southampton-2023-05-08 20:00:00	L
943	Southampton-2023-05-13 15:00:00	L
944	Southampton-2023-05-21 14:00:00	L
945	Southampton-2023-05-28 16:30:00	D

760 rows × 2 columns

77 Kod tworzący listę zawierającą rezultaty meczów

Jak wynika z powyższego zrzutu ekranu, opisana zredukowałem liczbę kolumn w opisanej powyżej liście tak aby zawierała tylko niezbędne. Zdecydowałem się na takie rozwiązanie w celu zwiększenia wydajności algorytmu.

W kolejnym kroku napisałem algorytm, który pobiera dane z odpowiedniej referencji do mojej bazy danych, a następnie iteruje na początku po danych użytkowników, pozyskując informacje o ich zakładach, a następnie w przypadku gdy nie zostały one rozliczone oraz dany mecz znajduje się w zdefiniowanej wcześniej liście, realizuje je dodając odpowiednią ilość punktów użytkownikowi w przypadku wygranej oraz zmieniając ich status na rozliczony.

```

import pandas as pd

ref = db.reference("/bets", url = 'https://betapp-7a0bb-default-
rttdb.firebaseio.com/')

bets = ref.get()
for userKey, values in bets.items():
    for key, value in values.items():
        if(value['settled'] == False):
            match = match_results[match_results['Key'] == value['gameId']]
            if match.empty == False:
                if value['result'].startswith(match.iloc[0].Result):
                    output = value['input'] * value['rate']
                else:
                    output = 0

            ref.child(userKey).child(key).update({"output": round(output, 2),
            "settled": True })
            user_ref = db.reference("/userData/" + userKey,
            url = 'https://betapp-7a0bb-default-rttdb.firebaseio.com/')
            user_data = user_ref.get()

            user_ref.child(list(user_data.keys())[0]).update({"points":
            list(user_data.values())[0]['points'] + round(output,2)})

```

78 Skrypt rozliczający zakłady użytkowników

4. Aplikacja mobilna

W ramach tego projektu zamiast tworzyć klasyczną aplikację webową, postawiłem na aplikację mobilną. Powodem tej decyzji było głównie to, że oprogramowanie działające na przeglądarce, stworzyłem w ramach mojej pracy inżynierskiej, więc wolałem postawić na coś innego. Dodatkowo uważam, że do aplikacja na telefon jest znacznie wygodniejsza do obstawiania meczy, gdyż użytkownik może mieć do niej dostęp praktycznie wszędzie, bez logowania się do przeglądarki internetowej, co zwykle zabiera znacznie więcej czasu i jest zdecydowanie mniej praktyczną opcją.

Oprogramowanie, które utworzyłem w ramach tej części pracy ma charakter prezentacyjny i nie jest w stu procentach gotowe do wypuszczenia na rynek. Z tego powodu zawarłem w

nim tylko najistotniejsze funkcjonalności oraz posłużyłem się domyślnymi komponentami graficznymi oferowanymi przez Anroid Studio.

4.1 Wybór technologii

Do utworzenia tej części pracy rozważałem dwa potencjalne rozwiązania Flutter i Android Studio. Oba narzędzia służą do tworzenia aplikacji mobilnych, lecz oczywiście zawierają kilka różnic.

Android Studio to środowisko programistyczne służące do tworzenia aplikacji na Androida, stworzone na wzór IntelliJ IDEA, przez firmy JetBrains oraz Google. To narzędzie jest zarówno wszechstronne jak i łatwe w użyciu i dodatkowo oferuje on bardzo wygodną integrację z usługami Google, z których jedną jest użyty przeze mnie Firebase. (Google, 2023)

Flutter to natomiast zestaw do tworzenia oprogramowania zarówno na system Android jak i IOS. To narzędzie również pozwala na szybkie tworzenie aplikacji oraz daje nam wiele przydatnych funkcji takich jak przywrócenie wcześniejszego kodu javascript. Kolejną ciekawą funkcją Fluttera jest „hot reload”, czyli wizualizacja wprowadzonych zmian bez konieczności zresetowania aplikacji przez użytkownika oraz ponownej kompilacji kodu. (Flutter, 2021)

Główną różnicą między technologiami jest to, że w Android Studio stosuje się głównie Java lub Kotlin, a Flutter obsługuje mniej popularny język programowania o nazwie Dart. Mimo to oba narzędzia są bardzo popularne oraz lubiane przez użytkowników, a dodatkowo duże firmy często decydują się na ich zastosowanie podczas tworzenia swoich aplikacji. (InterviewBit, 2022)

W mojej pracy zdecydowałem zaprezentować Android Studio, ponieważ moja wiedza na jego temat jest zdecydowanie większa, a dzięki niemu mogłem bez problemu zaimplementować wszystkie potrzebne przeze mnie funkcjonalności. Mimo to polecam również zapoznać się z technologią Flutter, ponieważ wydają się ona bardziej przyszłościowa i istnieje spora szansa, że zdominuje ona rynek aplikacji mobilnych.

4.2 System Autentykacji

Pierwszą funkcjonalnością, którą utworzyłem w ramach tej aplikacji mobilnej jest system autentykacji użytkownika. Jest ona kluczowa dla działania tego typu aplikacji, ponieważ pozwala ona na posiadanie przez użytkownika własnego konta, na którym może on obstawiać wyniki meczów, oraz przypisana jest do niego liczba punktów, które posiada.

Jak wspominałem wcześniej system autoryzacji użytkownika utworzyłem przy pomocy Firebase. W tym celu dodałem zależność: `'com.google.firebase:firebase:auth:21.1.0'`

do sekcji dependencies w pliku build.gradle. Dzięki tej czynności uzyskałem dostęp do klas służących do realizacji autentykacji dostarczonych przez Firebase.

Rejestracja

Rejestrację zrealizowałem poprzez utworzenie nowej aktywności, czyli klasy odpowiedzialnej za interakcje z użytkownikiem, oraz tworzenie widoku naszej aplikacji.

Na początku w utworzonym automatycznie pliku z rozszerzeniem .xml, dodałem niezbędne komponenty graficzne, do których będę odwoływał się w kodzie klasy RegisterActivity.

Następnie wewnątrz tej aktywności utworzyłem obiekt auth, który reprezentuje instancję klasy FirebaseAuth. Za jego pomocą będę mógł komunikować się z systemem autentykacji w Firebase.

```
auth = FirebaseAuth.getInstance()
```

79 Utworzenie obiektu auth

Kolejnym krokiem było zdefiniowanie funkcjonalności utworzonego wcześniej przycisku do rejestracji. W pierwszej kolejności przypisałem wartości pól tekstowych, które użytkownik wypełnia podczas tej funkcjonalności do zmiennych, poprzez odwołanie się do atrybutu text należącego do tych komponentów.

```
var email = binding.editTextEmail.text.toString()
var password = binding.editTextPassword.text.toString()
var confirmPassword = binding.editTextConfirmPassword.text.toString()
```

80 Przypisanie wprowadzonego przez użytkownika tekstu do zmiennych

W dalszej kolejności sprawdziłem czy podane hasła są identyczne, a w przypadku ich niezgodności zwróciłem wyjątek informujący o tym użytkownika. Następnie za pomocą wspomnianej wcześniej instancji użyłem metody createUserWithEmailAndPassword, pozwalającej na utworzenie użytkownika, dla której jako argumenty podałem email oraz hasło wprowadzone przez użytkownika.

Następnie dodałem do tej metody OnCompleteListener, w którym zdefiniowałem akcje, które ma wykonać po jej ukończeniu. W ramach tego fragmentu kodu, najpierw sprawdzam czy akcja została wykonana pomyślnie, a w przypadku wystąpienia błędu wyświetlam

tą informacje użytkownikowi. Następnie za pomocą obiektu auth loguje użytkownika do aplikacji. W kolejnym kroku za pomocą utworzonego wcześniej ViewModelu, dodaję dane o użytkowniku do bazy danych, które zawierają startową liczbę punktów. Na końcu zmieniam scenę na główną.

```
auth.createUserWithEmailAndPassword(
    email,
    password
).addOnCompleteListener{
    if(it.isSuccessful){
        Toast.makeText(this, "Registered successfully",
            Toast.LENGTH_LONG).show()
        auth.signInWithEmailAndPassword(
            email,
            password,
        )
        val userDataViewModel = UserDataViewModel(application)

        CoroutineScope(Dispatchers.IO).launch {
            userDataViewModel.insert(
                UserData(
                    id = "",
                    points = 50.0,
                )
            )
        }
        startActivity(Intent(this, MainActivity::class.java))
        this.finish()
    }
    else{
        Toast.makeText(this, "Problem with registration: " +
            it.exception?.message, Toast.LENGTH_LONG).show()
    }
}
} catch (e: Exception){
    Toast.makeText(
        this,
        e?.message,
        Toast.LENGTH_LONG
    ).show()
}
}
```

81 Kod dodający użytkownika do bazy danych wraz z listenerem obsługującym to zdarzenie

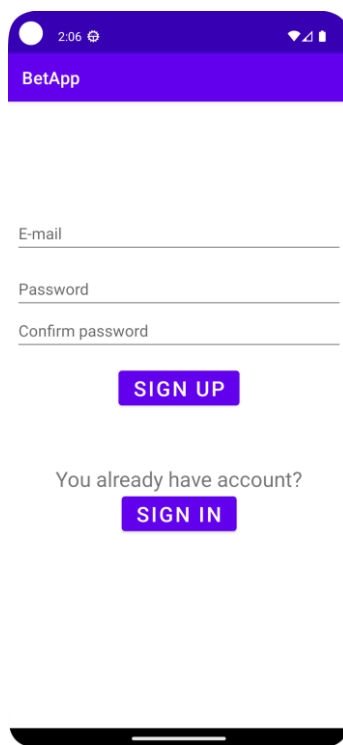
Ostatnią rzeczą, którą dodałem do tej aktywności było przeciążenie funkcji onResume(), w której sprawdzam czy użytkownik jest zalogowany i jeśli tak to kończę tą aktywność oraz zmieniam scenę na główną.


```

val user = auth.currentUser
if (user != null) {
    startActivity(Intent(this, MainActivity::class.java))
    finish()
}

```

82 Kod zawarty w funkcji onResume()



83 Ekran rejestracji użytkownika

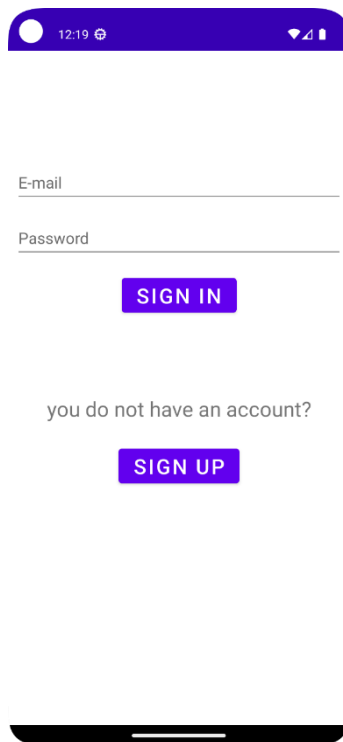
Na powyższym zrzucie ekranu znajduje się widok utworzonego widoku rejestracji użytkownika. Dodatkową funkcjonalnością, której nie opisałem w tym podrozdziale jest dodatkowy przycisk pozwalający na przejście do ekranu logowania, który opiszę w kolejnym podrozdziale.

Logowanie

Funkcjonalność logowania zrealizowałem, analogicznie do opisanej wcześniej rejestracji. W tym przypadku również utworzyłem formularz, w którym użytkownik podaje swój email oraz hasło, zawierający przycisk służący do zalogowania na konto o podanych danych.

W klasie LoginActivity, analogicznie jak w poprzednim przypadku zdefiniowałem funkcjonalność guzika logowania, korzystając z instancji klasy FirebaseAuth.

W tym przypadku również dodałem do użytej funkcji onCompleteListener, który sprawdza czy akcja przebiegła pomyślnie, a następnie wyświetla użytkownikowi odpowiedni komunikat oraz w przypadku sukcesu przekierowuje go do głównego widoku aplikacji.



84 Ekran logowania użytkownika

Na powyższym zrzucie ekranu, widok logowania wygląda bardzo podobnie do utworzonej poprzednio rejestracji. W tym przypadku osoba korzystająca z aplikacji również ma możliwość na szybkie przejście do funkcjonalności rejestracji nowego konta.

Przekierowanie do ekranu logowania

Ostatnią funkcjonalnością, którą dodałem w ramach systemu autentykacji, było automatycznie przekierowanie użytkownika do ekranu logowania z głównej aktywności w przypadku braku zalogowanego użytkownika.

Zrealizowałem to analogicznie jak w przypadku przekierowania do głównej aktywności z ekranu rejestracji lub logowania jeśli użytkownik jest zalogowany. Jedyną różnicą jest to, że w tym przypadku sprawdzam czy wartość atrybutu `currentUser` należącego do instancji klasy `FirebaseAuth` jest równa `null` i wtedy uruchamiam ekran logowania.

```
val user = auth.currentUser
if (user == null) {
    startActivity(Intent(this, LoginActivity::class.java))
    finish()
}
```

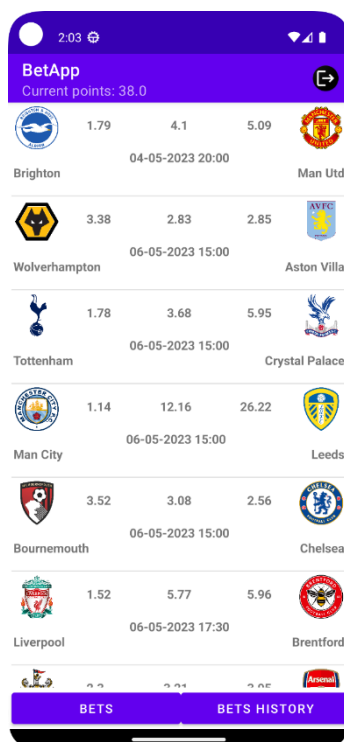
85 Kod odpowiedzialny za przekierowanie niezalogowanego użytkownika do ekranu logowania

4.3 Widoki zalogowanego użytkownika

W utworzonej w ramach mojej pracy dyplomowej aplikacji mobilnej, zdecydowałem się na utworzenie czterech głównych widoków, do których ma dostęp zalogowany użytkownik.

Widok główny

Najważniejszym ekranem, do którego ma dostęp zalogowany użytkownik, jest ekran startowy, zawierający listę wszystkich meczów, których wyniki można obstawić. Dodatkowo do tego ekranu dodałem również przyciski nawigacyjne, za pomocą których użytkownik będzie mógł przejść do innych sekcji, które opiszę w dalszej części tego rozdziału.



86 Widok główny w aplikacji mobilnej

Jak widać na powyższym rzucie ekranu pochodzącym z emulatora Android Studio, główną częścią ekranu startowego aplikacji jest lista zawierająca informacje o nadchodzących meczach. Każdy jej element zawiera dane o drużynach biorących udział w wydarzeniu, w których skład wchodzi ich nazwy oraz loga. Dodatkowo pomiędzy herbami zespołów umieściłem kursy na poszczególne rezultaty spotkania, z których kursy po bokach definiują szanse zwycięstwa ekipy, której logo znajduje się obok, a środkowy dotyczy remisu. Ostatnim elementem dotyczącym meczu jest data, która zawiera informacje o dniu, miesiącu, roku oraz dokładnej godzinie, o której odbywa się dane spotkanie.

Dodatkową funkcjonalnością zawartą w tym widoku jest możliwość przejścia do widoku obstawienia meczu, poprzez dłuższe kliknięcie na wybrany element listy.

Implementacja

Pierwszym krokiem podczas tworzenia tego widoku było utworzenie odpowiednich plików z rozszerzeniem .xml, odpowiedzialnych za ich wygląd. W pierwszej kolejności utworzyłem plik game_list_element.xml, w którym zdefiniowałem jak ma wyglądać pojedynczy element listy.

Następnie do pliku activity_main.xml dodałem RecyclerView, w którym umieszczę zdefiniowane powyżej elementy. Poza tym umieściłem w tym pliku również opisane wcześniej przyciski, za pomocą których będę nawigował pomiędzy ekranami.

Następnym krokiem było utworzenie adaptera, którego funkcją jest zarządzanie utworzoną listą. Wewnątrz tej klasy przeciążyłem kilka metod, z których najważniejsza była metoda onBindViewHolder(). Wspomniana funkcja jest odpowiedzialna za skonfigurowanie pojedynczego elementu w liście. Z tego powodu przypisałem odpowiednie atrybuty obiektu należącego do listy meczów do odpowiednich komponentów zdefiniowanych w pliku xml. Dodatkowo dodałem w do tej funkcji kod, który pozwoli użytkownikowi na przejście do widoku tworzenia nowego zakładu.

```
holder.binding.textViewTeam1.text = games[position].team1
holder.binding.textViewTeam2.text = games[position].team2

holder.binding.root.setOnLongClickListener {
    val intent = Intent(holder.binding.root.context, NewBetActivity::class.java)
    intent.putExtra("game", games[position])
    ContextCompat.startActivity(
        holder.binding.root.context,
        intent,
        Bundle()
    )
    true
}
```

87 Fragment funkcji onBindViewHolder(), odpowiedzialny za przypisanie nazw drużyn oraz zdefiniowanie akcji przenoszącej użytkownika do ekranu obstawiania wydarzenia

Ostatnim krokiem podczas implementacji listy, było utworzenie adaptera wewnątrz funkcji onCreate(), wewnątrz klasy MainActivity, a następnie przypisanie go do utworzonego wcześniej RecyclerView.

```
val adapter = GameAdapter()
binding.gamesRv.adapter = adapter
```

88 Przypisanie adaptera do RecyclerView

Poza tym wewnątrz tej klasy dodałem kilka funkcjonalności takich jak zdefiniowanie akcji przycisków nawigacyjnych oraz skonfigurowanie nagłówka aplikacji.

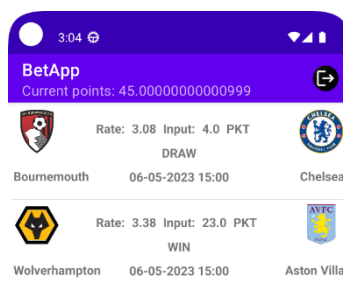
Ostatnią rzeczą podczas implementacji tej funkcjonalności było przypisanie listy obiektów pobranych z bazy danych do adaptera. Zrealizowałem tą funkcjonalność za pomocą metody `observer()`, która pozwala na kontrolowanie obiektów typu `LiveData`.

```
gameViewModel.allGames.observe(this, androidx.lifecycle.Observer {  
    it.let {  
        adapter.setGames(it.values.toList().sortedWith(compareBy { it2 ->  
it2.dateTime })))  
    }  
})
```

89 Przypisanie listy meczów do adaptera

Widok aktywnych zakładów użytkownika

Kolejnym widokiem, który utworzyłem była lista zawierająca wszystkie mecze, które obstawił zalogowany gracz, które nie zostały jeszcze rozstrzygnięte.



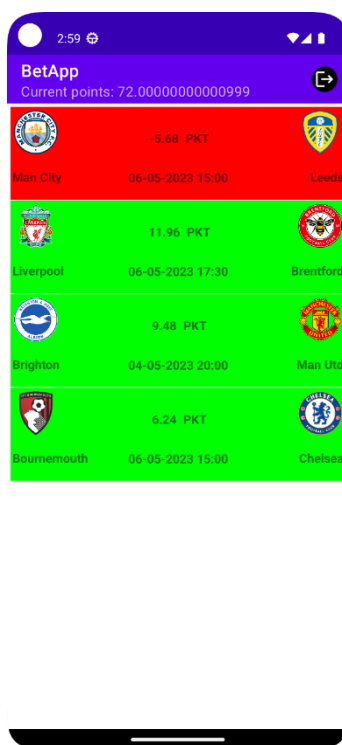
90 Widok aktywnych zakładów

Jak widać na powyższym zrzucie ekranu ten widok jest bardzo podobny do widoku zawierającego nadchodzące mecze i również zawiera informacje o drużynach oraz dacie rozpoczęcia spotkania. Główną różnicą pomiędzy nimi jest to, że zawiera on dodatkowo informacje o kursie, stawce oraz rezultacie na który gracz utworzył zakład.

Ze względu, że w tym przypadku implementacja była praktycznie identyczna jak w poprzednim przypadku, zdecydowałem się jej nie opisywać.

Widok historii zakładów

Ostatnim z głównych widoków zawierających listy, jest ekran historii zakładów. Zawiera on wszystkie zakłady, które gracz obstawił podczas korzystania z aplikacji, będące już rozliczone.



91 Ekran historii zakładów

Jak można zauważyć na zrzucie ekranu ta lista również zawiera informacje o drużynach oraz o dacie rozpoczęcia spotkania. Danymi unikalnymi dla tego widoku są natomiast bilans punktów zdobytych lub straconych w meczu oraz kolor elementu, który pokazuje czy zakład został wygrany czy przegrany.

Implementacja tego widoku była bardzo podobna do poprzednich główną różnicą było to, że w tym przypadku zmieniałem kolory elementów, za pomocą metody `setBackgroundColor()`.

```

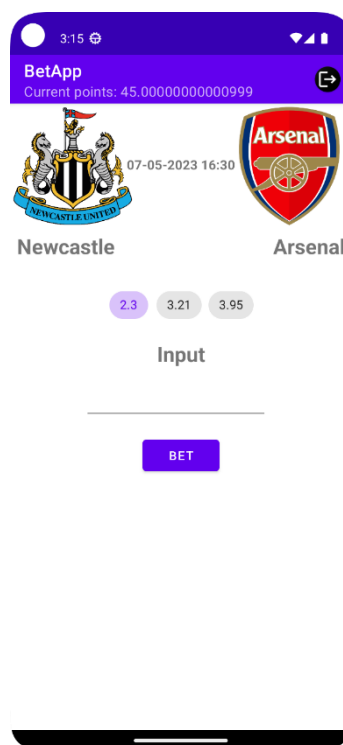
if(bets[position].output > 0)
    holder.binding.item.setBackgroundColor(Color.GREEN)
else
    holder.binding.item.setBackgroundColor(Color.RED)

```

92 Kod odpowiedzialny za zmianę koloru elementu listy w zależności od rezultatu spotkania

Widok dodawania nowego zakładu

Ostatnim widokiem, który umieściłem w mojej aplikacji był wcześniej wspomniany ekran pozwalający użytkownikowi utworzyć nowy zakład.



93 Ekran dodawania nowego zakładu

Jak widać na powyższym zrzucie ekranu ten widok zawiera podstawowe informacje o meczu, oraz kontrolki zawierający kursy na poszczególne rezultaty spotkania. Dodatkowo zawiera on formularz tekstowy, do którego użytkownik wprowadza liczbę punktów jaką chce obstawić na wybrane wydarzenie.

Implementacja

W tym przypadku implementacja oczywiście różni się od poprzednich widoków, gdyż tym razem nie jest to lista.

Pierwszym krokiem w tym przypadku oczywiście również było dodanie odpowiednich komponentów graficznych do utworzonego pliku o rozszerzeniu .xml.

W dalszej kolejności zdefiniowałem logikę tego ekranu, wewnątrz klasy NewBetActivity. W pierwszej kolejności przypisałem obiekt zawierający informacje do zmiennej, który uzyskałem poprzez wczytanie dodatku do obiektu typu Intent, który dodałem podczas przechodzenia do tej aktywności z widoku meczy.

```
val game = intent.extras?.get("game") as Game
```

94 Kod przypisujący dodatek do Intentu do zmiennej

W kolejnym kroku tak samo jak w poprzednich przypadkach przypisałem informacje pozyskane z obiektu typu Game, do odpowiednich komponentów graficznych.

Ostatnim krokiem w implementacji tej funkcjonalności było zdefiniowanie akcji utworzonego wewnątrz niej guzika. Na początku dodałem sprawdzanie czy użytkownik ustawił odpowiednią liczbę punktów i w przypadku gdy była ona nieprawidłowa, dodałem wyświetlanie odpowiedniego komunikatu. Następnie na podstawie wypełnionego formularza przez gracza, utworzyłem obiektu klasy Bet, który następnie dodałem do bazy Firebase, za pomocą utworzonej wcześniej metody insert(), należącej do klasy betViewModel, której działanie opisałem w trzecim rozdziale mojej pracy. Poza tym oczywiście dodałem również kod zmniejszający liczbę punktów użytkownika o obstawioną wartość w danym zakładzie.


```

binding.buttonBet.setOnClickListener{
    val input = binding.editTextInput.text.toString().toDoubleOrNull() ?: 0.0
    if(input > 0){
        if(input > userData.points){
            Toast.makeText(this, "You don't have enough points",
Toast.LENGTH_LONG).show()
        }
        else {
            val betViewModel = BetViewModel(application)
            val userDataViewModel = UserDataViewModel(application)
            var rate: Double
            var result: Result
            if (binding.chipWin.isChecked) {
                rate = binding.chipWin.text.toString().toDoubleOrNull() ?: 0.0
                result = Result.WIN
            } else if (binding.chipDraw.isChecked) {
                rate = binding.chipDraw.text.toString().toDoubleOrNull() ?: 0.0
                result = Result.DRAW
            } else {
                rate = binding.chipLoss.text.toString().toDoubleOrNull() ?: 0.0
                result = Result.LOSS
            }
            userData.points = userData.points - input

            CoroutineScope(Dispatchers.IO).launch {

                userDataViewModel.update(userData)

                betViewModel.insert(
                    Bet(
                        id = "",
                        gameId = game.id,
                        rate = rate,
                        result = result,
                        input = input,
                        settled = false,
                        output = 0.0
                    )
                )
                finish()
            }
        }
    }
    else {
        Toast.makeText(this, "You have to fill all required fields to bet",
Toast.LENGTH_LONG).show()
    }
}
}

```

95 Kod odpowiedzialny za dodanie nowego zakładu

Podsumowanie

W ramach mojej pracy magisterskiej przeszedłem przez proces tworzenia aplikacji mobilnej pozwalającej użytkownikowi na obstawianie meczów piłkarskich odbywających się w ramach Premier League, czyli ligi angielskiej.

Podczas opisu procesu tworzenia oprogramowania, w każdym kroku starałem się brać pod uwagę kilka różnych rozwiązań porównując je oraz wybierając jedno na potrzeby mojego projektu.

Na potrzeby mojej pracy dyplomowej utworzyłem algorytm, który na podstawie danych o zakończonych spotkaniach, wyznacza kursy nadchodzących meczów, wykorzystując przy tym sztuczną inteligencję. Skuteczność działania algorytmu zweryfikowałem porównując jego wyniki do kursów oferowanych, przez najpopularniejszego w Polsce bukmachera, a rezultaty okazały się zadowalające.

Kod który napisałem w ramach mojej pracy nie jest gotowy do wykorzystania w celach biznesowych, gdyż został on napisany aby zaprezentować teoretyczną część projektu. Dodatkowo celem tej pracy nie było utworzenie w pełni funkcjonalnej aplikacji, więc pisanie kodu w tym przypadku było drugoplanowe.

Oczywiście napisany przeze mnie kod nie jest bezwartościowy i po dodaniu kilku funkcjonalności może być on w pełni działającą aplikacją bukmacherską. Rzeczą która warto dodać w tym przypadku jest zmiana wyglądu użytych przeze mnie komponentów graficznych, gdyż użyłem domyślnych dla Android Studio, co nie jest najlepszym rozwiązaniem dla większej aplikacji. Ten problem można rozwiązać na przykład kupując gotowy zestaw komponentów. Dodatkowo wszystkie utworzone przeze mnie skrypty w Pythonie, znajdują się obecnie w formacie .ipynb, więc podczas wdrażania aplikacji należałoby umieścić je na wykupionym serwerze i uruchamiać cyklicznie w zdefiniowanym odstępie czasowym. Do zrealizowania tej funkcjonalności można użyć na przykład narzędzie Hangfire, które opisałem w mojej pracy inżynierskiej.

Bibliografia

- Dearmer, A. (2023, Marzec 14). *Firestore vs. MySQL: Battle of the Databases*. Pobrano z lokalizacji integrate.io: <https://www.integrate.io/blog/firebase-vs-mysql/>
- Flutter. (2021). *Flutter documentation*. Pobrano z lokalizacji Flutter: <https://docs.flutter.dev/>
- Google. (2023, Maj 22). *Meet Android Studio*. Pobrano z lokalizacji Developers Android: <https://developer.android.com/studio/intro>
- Google. (brak daty). *Learn the fundamentals*. Pobrano z lokalizacji Firebase: <https://firebase.google.com/docs?hl=en>
- InterviewBit. (2022, Lipiec 5). *Flutter Vs Android Studio: What's the Difference?* Pobrano z lokalizacji InterviewBit: <https://www.interviewbit.com/blog/flutter-vs-android-studio/>
- Kwapisz, K. (2020, Marzec 29). *Web scraping Selenium*. Pobrano z lokalizacji Kamil Kwapisz Blog o programowaniu, technologii i biznesie: <https://kamil.kwapisz.pl/web-scraping-selenium/>
- Mamczur, M. (2020, Listopad 14). *Web Scraping – co to i jakie są „dobre” praktyki?* Pobrano z lokalizacji mirosławmamczur: <https://mirosławmamczur.pl/web-scraping-co-to-i-jakie-sa-dobre-praktyki/>
- Nair, V. G. (2014). *Getting Started with BeautifulSoup*. Packt Publishing.
- Pornaras, G. (2021, Październik 7). *Selenium vs. BeautifulSoup: A Full Comparison*. Pobrano z lokalizacji blazemeter: <https://www.blazemeter.com/blog/selenium-vs-beautiful-soup-python>
- Redakcja Goal.pl. (2022, Grudzień 1). *Jak bukmacherzy ustalają kursy?* Pobrano z lokalizacji Goal.pl: <https://www.goal.pl/poradnik-typera/jak-bukmacherzy-ustalaja-kursy/>
- Sharma, H. (2023, Styczeń 26). *Comparing Python Libraries for Visualization*. Pobrano z lokalizacji medium: <https://medium.com/mlearning-ai/comparing-python-libraries-for-visualization-b2eb6c862542>
- Software Testing Help. (2022, Wrzesień 29). *Software Testing Help*. Pobrano z lokalizacji Top 13 Best Python Compiler For Python Developers [2022 Rankings]: <https://www.softwaretestinghelp.com/python-compiler/>
- Stachak, M. (2022, Czerwiec 9). *Rodzaje zakładów bukmacherskich*. Pobrano z lokalizacji TrustBet: <https://trustbet.pl/rodzaje-zakladow-bukmacherskich/>

StatsBomb. (brak daty). *What Are Expected Goals (xG)?* Pobrano z lokalizacji StatsBomb:
<https://statsbomb.com/soccer-metrics/expected-goals-xg-explained/>

statsmodels. (2023, Kwiecień 26). *Introduction*. Pobrano z lokalizacji statsmodels:
<https://www.statsmodels.org/stable/index.html>

Urban, O. (2022, Listopad 15). *Is web scraping legal?* Pobrano z lokalizacji APIFY Blog:
<https://blog.apify.com/is-web-scraping-legal/>

vavatech. (brak daty). *MySQL*. Pobrano z lokalizacji vavatech:
<https://vavatech.pl/technologie/bazy-danych/mysql>

Wikipedia. (2019, Czerwiec 19). *Zakłady bukmacherskie*. Pobrano z lokalizacji Wikipedia:
https://pl.wikipedia.org/wiki/Zak%C5%82ady_bukmacherskie

Wikipedia. (2022, Sierpień 26). *Robots Exclusion Protocol*. Pobrano z lokalizacji Wikipedia:
https://pl.wikipedia.org/wiki/Robots_Exclusion_Protocol