НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

**КУРСОВА РОБОТА**

з дисципліни «Аналіз даних в інформаційних системах»

на тему: «Аналіз гоночної серії FORMULA 1»

Студентів 2 курсу групи ІП-21

Спеціальності: 121

«Інженерія програмного забезпечення»

Загребельний О.А.

Голованьов Г.О.

«ПРИЙНЯВ» з оцінкою

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

доц. Ліхоузова Т.А. / доц. Олійник Ю.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Підпис                    Дата

Київ - 2024 рік

Національний технічний університет України “КПІ ім. Ігоря Сікорського”

Кафедра інформатики та програмної інженерії

Дисципліна Аналіз даних в інформаційно-управляючих системах

Спеціальність 121 "Інженерія програмного забезпечення"

Курс 2 Група ІП-21 Семестр 4

**ЗАВДАННЯ**

**на курсову роботу студента**

|  |
| --- |
| Загребельного О.А., Голованьова Г.О. |

|  |  |
| --- | --- |
| 1.Тема роботи | Аналіз гоночної серії FORMULA 1 |
|  | |
|  | |

|  |  |
| --- | --- |
| 2.Строк здачі студентом закінченої роботи | 29.05.2024 |

|  |  |
| --- | --- |
| 3. Вхідні дані до роботи | методичні вказівки до курсової роботи, обрані дані з сайтів |
| https://github.com/f1db/f1db | |
| https://docs.fastf1.dev/ | |

4.Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

|  |
| --- |
| 1.Постановка задачі |
| 2.Аналіз предметної області |
| 3.Розробка сховища даних |
| 4.Інтелектуальний аналіз даних |

5.Перелік графічного матеріалу ( з точним зазначенням обов’язкових креслень )

|  |  |
| --- | --- |
|  | |
|  | |
|  | |
|  | |
| 6.Дата видачі завдання | 30.03.2024 |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи | 30.03.2024 |  |
| 2. | Визначення зовнішніх джерел даних | 06.04.2024 |  |
| 3. | Пошук та вивчення літератури з питань курсової роботи | 13.04.2024 |  |
| 4. | Розробка моделі сховища даних | 20.04.2024 |  |
| 4. | Розробка ETL процесів | 27.04.2024 |  |
| 5. | Обґрунтування методів інтелектуального аналізу даних | 01.05.2024 |  |
| 6. | Застосування та порівняння ефективності методів інтелектуального аналізу даних | 15.05.2024 |  |
| 7. | Підготовка пояснювальної записки | 22.05.2024 |  |
| 8. | Здача курсової роботи на перевірку | 29.05.2024 |  |
| 9. | Захист курсової роботи | 03.06.2024 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  | Загребельний О.А. |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |
| Студент | |  |  | Голованьов Г.О. |
|  | | (підпис) |  | (прізвище, ім’я, по батькові) |

|  |  |  |  |
| --- | --- | --- | --- |
| Керівник |  |  | доц. Ліхоузова Т.А |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |
| Керівник |  |  | доц. Олійник Ю.О. |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

"29" травня 2024 р.

**АНОТАЦІЯ**

Пояснювальна записка до курсової роботи: 129 сторінок, 92 рисунки, 22 таблиці, 8 посилань.

Об’єкт дослідження: інтелектуальний аналіз даних.

Предмет дослідження: створення програмного забезпечення, що проводить аналіз даних з подальшим прогнозуванням та графічним відображенням результатів.

Мета роботи: проектування та реалізація сховища даних та ETL процесів, а також реалізація програмного забезпечення для отримання даних зі сховища та їх подальшого аналізу та прогнозування.

Дана курсова робота включає в себе: опис проектування, створення та заповнення сховища даних за даною задачею за допомогою фізичної моделі бази даних, опис створення програмного забезпечення для інтелектуального аналізу даних, їх графічного відображення та прогнозування за допомогою різних моделей.

СХОВИЩЕ ДАННИХ, МОДЕЛЬ ПРОГНОЗУВАННЯ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ, ФІЗИЧНА МОДЕЛЬ БАЗИ ДАНИХ, ELT ПРОЦЕСИ, МОДЕЛЬ SARIMAX, ЧАСОВИЙ ПРОМІЖОК, ДОВІРЧІ ІНТЕРВАЛИ, МОДЕЛЬ ARIMA.

**Зміст**

[**Вступ** 5](#_Toc167910615)

[**1.Постановка задачі** 6](#_Toc167910616)

[**2.АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ** 8](#_Toc167910617)

[**3.РОЗРОБКА СХОВИЩА ДАНИХ** 10](#_Toc167910618)

[**4.ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ** 62](#_Toc167910619)

[**ВИСНОВКИ** 87](#_Toc167910620)

[**ПЕРЕЛІК ПОСИЛАНЬ** 88](#_Toc167910621)

[**Додаток А Тексти програмного коду** 89](#_Toc167910622)

**Вступ**

У сучасному світі автоспорт, зокрема серія перегонів Formula 1, є одним із найпопулярніших та найпрестижніших видів спорту, що привертає увагу мільйонів глядачів по всьому світу. Формула 1 не лише захоплює своїми динамічними гонками та високотехнологічними автомобілями, але й приносить величезні прибутки та стимулює розвиток інновацій у різних галузях. Щорічно серія перегонів Formula 1 привертає нових вболівальників та встановлює нові рекорди за відвідуваністю та переглядами.

У рамках даної курсової роботи розроблено сховище даних, яке містить інформацію про результати перегонів Formula 1, базуючись на фізичній моделі бази даних. Основу сховища було створено за допомогою SQL скриптів, а робота з базою даних реалізована через програмне забезпечення для імплементації ETL процесів та інтелектуального аналізу даних.

Як система керування сховищем даних для даної роботи використовується PostgreSQL, а мова програмування для реалізації застосунку - Python3. Використовуючи ці інструменти, було розроблено програмне забезпечення, яке дозволяє отримувати дані зі сховища, проводити їх аналіз, прогнозувати результати перегонів та графічно відображати отримані результати.

Ця робота спрямована на дослідження та аналіз даних перегонів Formula 1, що дозволить не лише зрозуміти поточні тенденції, але й зробити прогнози щодо майбутніх перегонів, використовуючи різні моделі прогнозування, такі як K-nearest neighbours та Random Forest.

**1.Постановка задачі**

Під час виконання курсової роботи необхідно виконати комплекс завдань, спрямованих на створення ефективного інструменту для аналізу та прогнозування результатів перегонів Formula 1. Насамперед, потрібно створити сховище даних типу «сніжинка», яке міститиме щонайменше шість таблиць вимірів та одну таблицю фактів.

Наступним кроком є створення ETL процесів для завантаження даних до сховища, їх отримання зі сховища за допомогою запитів, а також оновлення та додавання даних до таблиць.

Реалізація спроектованого сховища даних буде здійснюватися з використанням PostgreSQL, що забезпечить високу продуктивність та надійність системи. Для подальшого аналізу даних необхідно створити програмне забезпечення, яке дозволить отримувати вибірки даних зі створеного сховища, графічно відображати їх, проводити інтелектуальний аналіз та здійснювати передбачення за допомогою різних моделей прогнозування.

У роботі будуть використовуватись моделі прогнозування, такі як k-nearest neighbours, random forest, logistic regression та support vector machines для аналізу даних. Важливо застосувати найоптимальнішу модель прогнозування на згладжених даних та порівняти отримані результати з прогнозуванням на початкових даних.

Додатково, необхідно провести автокореляцію та часткову автокореляцію для виявлення повторюваних ділянок даних, незважаючи на наявність шумів. Аналіз отриманих результатів та порівняння різних методів прогнозування дозволить визначити найоптимальніший метод. Важливим аспектом є знаходження довірчих інтервалів з використанням найкращої моделі прогнозування.

Для реалізації застосунку буде використовуватись мова програмування Python 3. Весь проект повинен бути завершений до дедлайну, початку сесії, з дотриманням єдиного стилю написання коду (coding style), що забезпечить його читабельність та підтримуваність у майбутньому.

**2.АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

Сьогодні автоспорт, зокрема серія перегонів Formula 1, є одним із найпопулярніших видів спорту у світі, який постійно привертає увагу мільйонів глядачів. Formula 1, або "Королева автоспорту", не тільки захоплює своїми видовищними гонками та інноваційними технологіями, але й є значущою складовою світової економіки, генеруючи величезні доходи від трансляцій, спонсорства та продажу квитків. Гонки Formula 1 не лише пропонують захоплююче видовище, але й виступають платформою для впровадження новітніх технологій та інженерних рішень.

Аналіз результатів перегонів Formula 1 є складним завданням, яке включає багато факторів, таких як погодні умови, характеристики трас, технічний стан автомобілів та майстерність пілотів. Для команд та аналітиків важливо виявляти закономірності, що впливають на результати перегонів, аби оптимізувати стратегії, покращувати результати та передбачати майбутні успіхи. Вміння прогнозувати результати перегонів на основі історичних даних дозволяє командам краще готуватися до майбутніх змагань та приймати обґрунтовані рішення щодо розвитку техніки і тактики.

У рамках даної курсової роботи буде розроблено програмне забезпечення для аналізу та прогнозування результатів перегонів Formula 1. Функціональність цього програмного забезпечення включає:

* проектування сховища даних;
* створення ETL процесів для завантаження та оновлення даних;
* створення вибірки даних зі сховища;
* інтелектуальний аналіз даних;
* використання декількох моделей прогнозування даних;
* прогнозування результатів перегонів;
* визначення факторів, що впливають на результати перегонів;
* графічне відображення отриманих результатів та їх аналіз.

Сховище даних дозволить зберігати великі обсяги історичних даних про результати перегонів, що забезпечить основу для детального аналізу та прогнозування. ETL процеси забезпечать актуальність та повноту даних, необхідних для аналітичної роботи. Інтелектуальний аналіз даних, включаючи перевірку на сезонність, стаціонарність та шум, а також згладжування даних, дозволить виявити приховані тенденції та шаблони. Використання моделей прогнозування, таких як k-nearest neighbours, random forest, logistic regression та support vector machines дозволить робити точні передбачення та оцінювати вплив різних факторів на результати перегонів. Графічне відображення результатів допоможе візуалізувати дані та зрозуміти основні висновки аналізу.

# **3.РОЗРОБКА СХОВИЩА ДАНИХ**

3.1 Розробка моделі сховища даних

Для виконання курсової роботи було обрано 2 джерела відкритих даних на сайті <https://github.com> , що включають в себе:

* Загальна інформація про перегони:   
  <https://github.com/f1db/f1db>
* Інформація про час кожного гонщика на всіх гран-прі: <https://docs.fastf1.dev/>

На основі детального опису та проведеного аналізу предметної області було розроблено наступну модель сховища даних гоночної серії Формула-1 за типом „сніжинка” (рисунок 3.1).

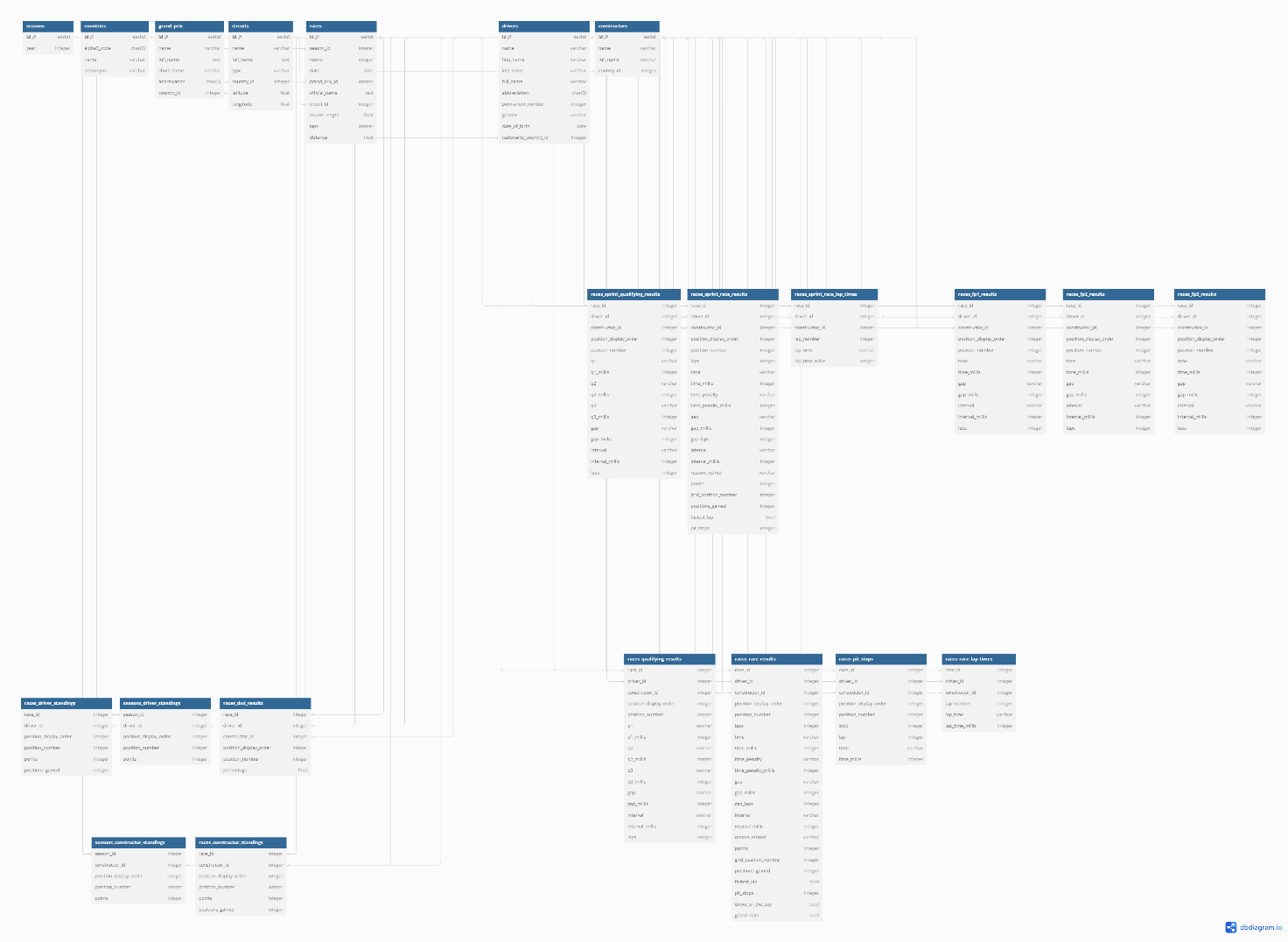


Рисунок 3.1 - Проектування моделі сховища за типом сніжинка

У моделі сховища за типом сніжинка спроектовано 15 таблиць фактів та 7 таблиць вимірів. Нижче наведено опис полів кожної таблиці сховища даних гоночної серії Формула-1 (таблиці 3.1-3.22).

Таблиця 3.1 - Таблиця виміру сезони **seasons**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| id | Унікальний ідентифікатор сезону (автоматично згенерований) |
| year | Рік сезону |

Таблиця 3.2 - Таблиця виміру країни **countries**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| id | Унікальний ідентифікатор країни (автоматично згенерований) |
| alpha3\_code | Трилітерний код країни |
| name | Назва країни |
| demonym | Демонім (назва для жителів країни) |

Таблиця 3.3 - Таблиця виміру трас **circuits**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| id | Унікальний ідентифікатор траси (автоматично згенерований) |
| name | Назва траси |
| full\_name | Повна назва траси |
| type | Тип траси |
| country\_id | Зовнішній ключ, що посилається на id країни |
| latitude | Широта |
| longitude | Довгота |

Таблиця 3.4 - Таблиця виміру конструкторів **constructors**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| id | Унікальний ідентифікатор конструктора (автоматично згенерований) |
| name | Назва конструктора |
| full\_name | Повна назва конструктора |
| country\_id | Зовнішній ключ, що посилається на id країни |

Таблиця 3.5 - Таблиця виміру пілотів **drivers**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| id | Унікальний ідентифікатор гонщика (автоматично згенерований) |
| name | Прізвище гонщика |
| first\_name | Ім'я гонщика |
| last\_name | Прізвище гонщика |
| full\_name | Повне ім'я гонщика |
| abbreviation | Абревіатура |
| permanent\_number | Постійний номер гонщика |
| gender | Стать |
| date\_of\_birth | Дата народження |
| nationality\_country\_id | Зовнішній ключ, що посилається на id країни громадянства гонщика |

Таблиця 3.6 - Таблиця виміру гран-прі **grand\_prix**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| id | Унікальний ідентифікатор гран-прі (автоматично згенерований) |
| name | Назва гран-прі |
| full\_name | Повна назва гран-прі |
| short\_name | Скорочена назва гран-прі |
| abbreviation | Абревіатура |
| country\_id | Зовнішній ключ, що посилається на id країни |

Таблиця 3.7 - Таблиця виміру гонки **races**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| id | Унікальний ідентифікатор гонки (автоматично згенерований) |
| season\_id | Зовнішній ключ, що посилається на id сезону |
| round | Номер етапу |
| date | Дата гонки |
| grand\_prix\_id | Зовнішній ключ, що посилається на id гран-прі |
| official\_name | Офіційна назва |
| circuit\_id | Зовнішній ключ, що посилається на id траси |
| course\_length | Довжина траси |
| laps | Кількість кіл |
| distance | Дистанція |

Таблиця 3.8 - Таблиця виміру гонки в першій вільній практиці **races\_fp1\_results**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| time | Час гонки |
| time\_millis | Час у мілісекундах |
| gap | Різниця в часі до лідера |
| gap\_millis | Різниця в часі до лідера у мілісекундах |
| interval | Інтервал між гонками |
| interval\_millis | Інтервал між гонками у мілісекундах |
| laps | Кількість кіл, що пройдено |

Таблиця 3.9 - Таблиця виміру гонки в другій вільній практиці **races\_fp2\_results**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| time | Час гонки |
| time\_millis | Час у мілісекундах |
| gap | Різниця в часі до лідера |
| gap\_millis | Різниця в часі до лідера у мілісекундах |
| interval | Інтервал між гонками |
| interval\_millis | Інтервал між гонками у мілісекундах |
| laps | Кількість кіл, що пройдено |

Таблиця 3.10 - Таблиця виміру гонки в третій вільній практиці **races\_fp3\_results**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| time | Час гонки |
| time\_millis | Час у мілісекундах |
| gap | Різниця в часі до лідера |
| gap\_millis | Різниця в часі до лідера у мілісекундах |
| interval | Інтервал між гонками |
| interval\_millis | Інтервал між гонками у мілісекундах |
| laps | Кількість кіл, що пройдено |

Таблиця 3.11 - Таблиця виміру результати гоночної кваліфікації **races\_qualifying\_results**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| q1 | Час в кваліфікації 1 |
| q1\_millis | Час в кваліфікації 1 у мілісекундах |
| q2 | Час в кваліфікації 2 |
| q2\_millis | Час в кваліфікації 2 у мілісекундах |
| q3 | Час в кваліфікації 3 |
| q3\_millis | Час в кваліфікації 3 у мілісекундах |
| gap | Різниця в часі до лідера |
| gap\_millis | Різниця в часі до лідера у мілісекундах |
| interval | Інтервал між гонками |
| interval\_millis | Інтервал між гонками у мілісекундах |
| laps | Кількість кіл, що пройдено |

Таблиця 3.12 - Таблиця виміру результати спринтової кваліфікації **races\_sprint\_qualifying\_results**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| q1 | Час в кваліфікації 1 |
| q1\_millis | Час в кваліфікації 1 у мілісекундах |
| q2 | Час в кваліфікації 2 |
| q2\_millis | Час в кваліфікації 2 у мілісекундах |
| q3 | Час в кваліфікації 3 |
| q3\_millis | Час в кваліфікації 3 у мілісекундах |
| gap | Різниця в часі до лідера |
| gap\_millis | Різниця в часі до лідера у мілісекундах |
| interval | Інтервал між гонками |
| interval\_millis | Інтервал між гонками у мілісекундах |
| laps | Кількість кіл, що пройдено |

Таблиця 3.13 - Таблиця виміру результати гонки **races\_race\_results**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| laps | Кількість пройдених кол |
| time | Час гонки |
| time\_millis | Час гонки у мілісекундах |
| time\_penalty | Час штрафу |
| time\_penalty\_millis | Час штрафу у мілісекундах |
| gap | Різниця в часі до лідера |
| gap\_millis | Різниця в часі до лідера у мілісекундах |
| gap\_laps | Кількість кол відставання |
| interval | Інтервал до наступного гонщика |
| interval\_millis | Інтервал до наступного гонщика у мілісекундах |
| reason\_retired | Причина виходу з гонки |
| points | Очки |
| grid\_position\_number | Позиція на старті |
| positions\_gained | Кількість місць, що покращено за гонку |
| fastest\_lap | Прапорець, що позначає найшвидший круг |
| pit\_stops | Кількість зупинок на боксах |
| driver\_of\_the\_day | Прапорець, що позначає гонщика дня |
| grand\_slam | Прапорець, що позначає виграш "гранд-слему" |

Таблиця 3.14 - Таблиця виміру результати спринту **races\_sprint\_race\_results**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| laps | Кількість пройдених кол |
| time | Час гонки |
| time\_millis | Час гонки у мілісекундах |
| time\_penalty | Час штрафу |
| time\_penalty\_millis | Час штрафу у мілісекундах |
| gap | Різниця в часі до лідера |
| gap\_millis | Різниця в часі до лідера у мілісекундах |
| gap\_laps | Кількість кол відставання |
| interval | Інтервал до наступного гонщика |
| interval\_millis | Інтервал до наступного гонщика у мілісекундах |
| reason\_retired | Причина виходу з гонки |
| points | Очки |
| grid\_position\_number | Позиція на старті |
| positions\_gained | Кількість місць, що покращено за гонку |
| fastest\_lap | Прапорець, що позначає найшвидший круг |
| pit\_stops | Кількість зупинок на боксах |
| driver\_of\_the\_day | Прапорець, що позначає гонщика дня |
| grand\_slam | Прапорець, що позначає виграш "гранд-слему" |

Таблиця 3.15 - Таблиця виміру результати піт-стопів **races\_pit\_stops**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| stop | Номер зупинки |
| lap | Номер круга, на якому відбулась зупинка |
| time | Час зупинки |
| time\_millis | Час зупинки у мілісекундах |

Таблиця 3.16 - Таблиця виміру колові часи у спринті **races\_sprint\_race\_lap\_times**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| lap\_number | Номер круга |
| lap\_time | Час проходження круга |
| lap\_time\_millis | Час проходження круга у мілісекундах |

Таблиця 3.17 - Таблиця виміру колові часи у гонці **races\_race\_lap\_times**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| lap\_number | Номер круга |
| lap\_time | Час проходження круга |
| lap\_time\_millis | Час проходження круга у мілісекундах |

Таблиця 3.18 - Таблиця виміру позиції конструкторів під час певної гонки **races\_constructor\_standings**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| points | Кількість очок |
| positions\_gained | Кількість позицій, які були покращені |

Таблиця 3.19 - Таблиця виміру гонщик дня (Drive Of the Day) **races\_dod\_results**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| percentage | Відсоток голосів за гонщика дня |

Таблиця 3.20 - Таблиця виміру позиції гонщиків під час певної гонки **races\_driver\_standings**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| race\_id | Зовнішній ключ, що посилається на id гонки |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| points | Кількість очок |
| positions\_gained | Кількість позицій, які були покращені |

Таблиця 3.21 - Таблиця виміру позиції конструкторів після завершення сезону **seasons\_constructor\_standings**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| season\_id | Зовнішній ключ, що посилається на id сезону |
| constructor\_id | Зовнішній ключ, що посилається на id конструктора |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| points | Кількість очок |

Таблиця 3.22 - Таблиця виміру позиції гонщиків після завершення сезону **seasons\_driver\_standings**:

|  |  |
| --- | --- |
| **Поле** | **Пояснення** |
| season\_id | Зовнішній ключ, що посилається на id сезону |
| driver\_id | Зовнішній ключ, що посилається на id гонщика |
| position\_display\_order | Порядковий номер позиції |
| position\_number | Номер позиції |
| points | Кількість очок |

На основі спроектованої моделі було створено сховище даних гоночної серії Формула-1, що включає 7 таблиць вимірів та 15 таблиць фактів реалізованих за допомогою нижче наведених скриптів використовуючи PostgreSQL.

Створення таблиці виміру сезони в сховищі даних:

CREATE TABLE seasons (

    id SERIAL PRIMARY KEY,

    year INTEGER

);

Створення таблиці виміру країни в сховищі даних:

CREATE TABLE countries (

    id SERIAL PRIMARY KEY,

    alpha3\_code CHAR(3),

    name VARCHAR,

    demonym VARCHAR

);

Створення таблиці виміру кола в сховищі даних:

CREATE TABLE circuits (

    id SERIAL PRIMARY KEY,

    name VARCHAR,

    full\_name TEXT,

    type VARCHAR,

    country\_id INTEGER REFERENCES countries(id),

    latitude FLOAT,

    longitude FLOAT

);

Створення таблиці виміру конструктори в сховищі даних:

CREATE TABLE constructors (

    id SERIAL PRIMARY KEY,

    name VARCHAR,

    full\_name VARCHAR,

    country\_id INTEGER REFERENCES countries(id)

);

Створення таблиці виміру пілоти в сховищі даних:

CREATE TABLE drivers (

    id SERIAL PRIMARY KEY,

    name VARCHAR,

    first\_name VARCHAR,

    last\_name VARCHAR,

    full\_name VARCHAR,

    abbreviation CHAR(3),

    permanent\_number INTEGER,

    gender VARCHAR,

    date\_of\_birth DATE,

    nationality\_country\_id INTEGER REFERENCES countries(id)

);

Створення таблиці виміру гран-прі в сховищі даних:

CREATE TABLE grand\_prix (

    id SERIAL PRIMARY KEY,

    name VARCHAR,

    full\_name TEXT,

    short\_name VARCHAR,

    abbreviation CHAR(3),

    country\_id INTEGER REFERENCES countries(id)

);

Створення таблиці виміру гонки в сховищі даних:

CREATE TABLE races (

    id SERIAL PRIMARY KEY,

    season\_id INTEGER REFERENCES seasons(id),

    round INTEGER,

    date DATE,

    grand\_prix\_id INTEGER REFERENCES grand\_prix(id),

    official\_name TEXT,

    circuit\_id INTEGER REFERENCES circuits(id),

    course\_length FLOAT,

    laps INTEGER,

    distance FLOAT

);

Створення таблиці виміру результати першої вільної практики в сховищі даних:

CREATE TABLE races\_fp1\_results (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    time VARCHAR,

    time\_millis INTEGER,

    gap VARCHAR,

    gap\_millis INTEGER,

    interval VARCHAR,

    interval\_millis INTEGER,

    laps INTEGER

);

Створення таблиці виміру результати другої вільної практики в сховищі даних:

CREATE TABLE races\_fp2\_results (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    time VARCHAR,

    time\_millis INTEGER,

    gap VARCHAR,

    gap\_millis INTEGER,

    interval VARCHAR,

    interval\_millis INTEGER,

    laps INTEGER

);

Створення таблиці виміру результати третьої вільної практики в сховищі даних:

CREATE TABLE races\_fp3\_results (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    time VARCHAR,

    time\_millis INTEGER,

    gap VARCHAR,

    gap\_millis INTEGER,

    interval VARCHAR,

    interval\_millis INTEGER,

    laps INTEGER

);

Створення таблиці виміру результати кваліфікації в сховищі даних:

CREATE TABLE races\_qualifying\_results (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    q1 VARCHAR,

    q1\_millis INTEGER,

    q2 VARCHAR,

    q2\_millis INTEGER,

    q3 VARCHAR,

    q3\_millis INTEGER,

    gap VARCHAR,

    gap\_millis INTEGER,

    interval VARCHAR,

    interval\_millis INTEGER,

    laps INTEGER

);

Створення таблиці виміру результати гонки в сховищі даних:

CREATE TABLE races\_race\_results (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    laps INTEGER,

    time VARCHAR,

    time\_millis INTEGER,

    time\_penalty VARCHAR,

    time\_penalty\_millis INTEGER,

    gap VARCHAR,

    gap\_millis INTEGER,

    gap\_laps INTEGER,

    interval VARCHAR,

    interval\_millis INTEGER,

    reason\_retired VARCHAR,

    points INTEGER,

    grid\_position\_number INTEGER,

    positions\_gained INTEGER,

    fastest\_lap BOOLEAN,

    pit\_stops INTEGER,

    driver\_of\_the\_day BOOLEAN,

    grand\_slam BOOLEAN

);

Створення таблиці виміру піт-стопи під час гонки в сховищі даних:

CREATE TABLE races\_pit\_stops (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    stop INTEGER,

    lap INTEGER,

    time VARCHAR,

    time\_millis INTEGER

);

Створення таблиці виміру результати кваліфікації на спринт в сховищі даних:

CREATE TABLE races\_sprint\_qualifying\_results (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    q1 VARCHAR,

    q1\_millis INTEGER,

    q2 VARCHAR,

    q2\_millis INTEGER,

    q3 VARCHAR,

    q3\_millis INTEGER,

    gap VARCHAR,

    gap\_millis INTEGER,

    interval VARCHAR,

    interval\_millis INTEGER,

    laps INTEGER

);

Створення таблиці виміру результат спринт-гонки в сховищі даних:

CREATE TABLE races\_sprint\_race\_results (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    laps INTEGER,

    time VARCHAR,

    time\_millis INTEGER,

    time\_penalty VARCHAR,

    time\_penalty\_millis INTEGER,

    gap VARCHAR,

    gap\_millis INTEGER,

    gap\_laps INTEGER,

    interval VARCHAR,

    interval\_millis INTEGER,

    reason\_retired VARCHAR,

    points INTEGER,

    grid\_position\_number INTEGER,

    positions\_gained INTEGER,

    fastest\_lap BOOLEAN,

    pit\_stops INTEGER

);

Створення таблиці виміру позиції конструкторів в таблиці очків під час гонки в сховищі даних:

CREATE TABLE races\_constructor\_standings (

    race\_id INTEGER REFERENCES races(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    points INTEGER,

    positions\_gained INTEGER

);

Створення таблиці виміру результати пілота дня в сховищі даних:

CREATE TABLE races\_dod\_results (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    percentage FLOAT

);

Створення таблиці виміру позиції пілотів в таблиці очків під час гонки в сховищі даних:

CREATE TABLE races\_driver\_standings (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    points INTEGER,

    positions\_gained INTEGER

);

Створення таблиці виміру позиції конструкторів після завершення сезону в сховищі даних:

CREATE TABLE seasons\_constructor\_standings (

    season\_id INTEGER REFERENCES seasons(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    points INTEGER

);

Створення таблиці виміру позиції пілотів після завершення сезону в сховищі даних:

CREATE TABLE seasons\_driver\_standings (

    season\_id INTEGER REFERENCES seasons(id),

    driver\_id INTEGER REFERENCES drivers(id),

    position\_display\_order INTEGER,

    position\_number INTEGER,

    points INTEGER

);

Створення таблиці виміру результати кіл під час спринт-гонки в сховищі даних:

CREATE TABLE races\_sprint\_race\_lap\_times (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    lap\_number INTEGER,

    lap\_time VARCHAR,

    lap\_time\_millis INTEGER

);

Створення таблиці виміру результати кіл під час гонки в сховищі даних:

CREATE TABLE races\_race\_lap\_times (

    race\_id INTEGER REFERENCES races(id),

    driver\_id INTEGER REFERENCES drivers(id),

    constructor\_id INTEGER REFERENCES constructors(id),

    lap\_number INTEGER,

    lap\_time VARCHAR,

    lap\_time\_millis INTEGER

);

3.2 Розробка ETL процесів

Для проектування сховища даних і подальшої роботи з ним необхідно враховувати набір методів, що реалізують процес перекладу вихідних даних в аналітичний формат, який підтримується сховищем даних і не порушує цілісність системи. Для цього ми обговоримо ключові особливості процесу ETL, включаючи:

**Процес завантаження.**

На цьому етапі ви перетягуєте всі дані про якість в ETL для подальшої обробки. Вхідні дані можуть бути джерелом різних типів і форматів, можуть створюватися в різних додатках і навіть використовувати різні кодування, тому в майбутньому для вирішення проблеми аналізу, запобігання впливу різних факторів, що ускладнюють коректний аналіз даних, дані будуть використовуватися в наступних цілях: зберігається в базі даних. Його необхідно перетворити в єдиний універсальний формат.

**Процес валідації даних.**

Основним завданням на цьому етапі є перевірка точності та повноти даних, після чого складається звіт про помилку для виправлення та узагальнення даних. Основне очищення даних у процесі ETL в основному є технічним, і основним завданням є підготовка даних до завантаження в сховище. Вторинна очистка в аналітичних системах визначається користувачем і призначена для підготовки даних для вирішення конкретних аналітичних задач. З цієї причини обидва етапи очищення однаково важливі і необхідні.

**Перетворення даних.**

Цей етап процесу ETL необхідний для підготовки даних до розміщення в сховищі даних і надання їм найбільш зручного формату для подальшого аналізу. У той же час слід враховувати вимоги до рівня якості даних. Тому в процесі перетворення можна використовувати найрізноманітніші інструменти, які можуть бути як найпростішими засобами ручного редагування даних, так і реалізацією досить складних методів обробки і очищення даних.

**Завантаження даних в сховище.**

Після того, як дані отримані з різних джерел і перетворені, агреговані і очищені, виконується останній етап ETL. Процес завантаження складається з перенесення даних з проміжної таблиці в структуру сховища даних. Таким чином, щоб пришвидшити завантаження даних, вам потрібно заздалегідь перевірити цілісність та точність даних, а потім розробити та вибрати оптимальний процес заповнення сховища. При завантаженні даних в сховище передається не вся інформація з джерела, а тільки інформація, яка змінилася за минулий з моменту попереднього завантаження час. В нашому випадку майже половина даних, які ми отримали з джерела, є непотрібні, тому під час завантаження їх до сховища були виконані роботи щодо цього.

3.3 Завантаження даних за допомогою ETL процесів

Для заповнення сховища даних відеоігор було реалізовано наступне:

1. Було зведено таблиці вимірів і заімпортовані до сховища, а саме:
   1. A screenshot of a computer

      Description automatically generatedCountries (країни)

Рисунок 3.2 – Вхідні дані таблиці **Countries**

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a black background with white text

Description automatically generated

Рисунок 3.3 – Вихідні дані

* 1. Circuits (кола)

A screenshot of a computer

Description automatically generated

Рисунок 3.4 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a computer

Description automatically generated

Рисунок 3.5 – Вихідні дані

* 1. Constructors (конструктори)

A screenshot of a computer

Description automatically generated

Рисунок 3.6 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a computer

Description automatically generated

Рисунок 3.7 – Вихідні дані

* 1. Drivers (пілоти)

A black screen with white text

Description automatically generated

Рисунок 3.8 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a computer

Description automatically generated

Рисунок 3.9 – Вихідні дані

* 1. Grand Prix (гран-прі)

A screenshot of a computer

Description automatically generated

Рисунок 3.10 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a computer

Description automatically generated

Рисунок 3.11 – Вихідні дані

* 1. Seasons (сезони)

A screenshot of a cell phone

Description automatically generated

Рисунок 3.12 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a cell phone

Description automatically generated

Рисунок 3.13 – Вихідні дані

* 1. Races (гонки)

A screenshot of a computer

Description automatically generated

Рисунок 3.14 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a computer

Description automatically generated

Рисунок 3.15 – Вихідні дані

б) Було зведено таблиці фактів і заімпортовані до сховища, а саме:

* 1. Races FP1 Results(Результати першої вільної практики)

A screenshot of a computer

Description automatically generated

Рисунок 3.16 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a computer

Description automatically generated

Рисунок 3.17 – Вхідні дані

* 1. Races FP2 Results(Результати другої вільної практики)

A screenshot of a computer

Description automatically generated

Рисунок 3.18 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a computer

Description automatically generated

Рисунок 3.19 – Вхідні дані

* 1. Races FP3 Results(Результати третьої вільної практики)

A screenshot of a computer

Description automatically generated

Рисунок 3.20 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a computer

Description automatically generated

Рисунок 3.21 – Вхідні дані

* 1. Races Qualifying Results(Результати кваліфікації)

A black screen with white text

Description automatically generated

Рисунок 3.22 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A black screen with white text

Description automatically generated

Рисунок 3.23 – Вхідні дані

* 1. Races Race Result(Результати гонки)

A screen shot of a computer

Description automatically generated

Рисунок 3.24 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screen shot of a black screen

Description automatically generated

Рисунок 3.25 – Вхідні дані

* 1. Races Pit Stops(Пітстопи під час гонки)

A black screen with white text

Description automatically generated

Рисунок 3.26 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a computer

Description automatically generated

Рисунок 3.27 – Вхідні дані

* 1. Races Sprint Qualifying Results(Результати кваліфікації на спринт-гонку)

A screenshot of a computer

Description automatically generated

Рисунок 3.28 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A black screen with white text

Description automatically generated

Рисунок 3.29 – Вхідні дані

* 1. Races Sprint Race Results(Результати спринт-гонки)

A screenshot of a computer

Description automatically generated

Рисунок 3.30 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A black screen with white text

Description automatically generated

Рисунок 3.31 – Вхідні дані

* 1. Races Constructor Standings(Позиції конструкторів під час сезону)

A screenshot of a computer

Description automatically generated

Рисунок 3.32 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a computer

Description automatically generated

Рисунок 3.33 – Вхідні дані

* 1. Races Driver Of the Day Results(Пілот дня)

A screenshot of a computer

Description automatically generated

Рисунок 3.34 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a black screen

Description automatically generated

Рисунок 3.35 – Вхідні дані

* 1. Races Driver Standings(Позиція пілотів під час сезону)

A screenshot of a computer

Description automatically generated

Рисунок 3.36 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a black screen

Description automatically generated

Рисунок 3.37 – Вхідні дані

* 1. Seasons Constructor Standing(Позиція конструкторів після завершення сезону)

A screenshot of a computer

Description automatically generated

Рисунок 3.38 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a black screen

Description automatically generated

Рисунок 3.39 – Вхідні дані

* 1. Seasons Driver Standings(Позиція пілотів після завершення сезону)

A screenshot of a black screen

Description automatically generated

Рисунок 3.40 – Вхідні дані

Приводимо дані у потрібний нам формат для сховища, видаляємо непотрібні поля та перейменуємо стовпці. Отже маємо такий результат:

A screenshot of a black screen

Description automatically generated

Рисунок 3.41 – Вхідні дані

в) Було створено функції завантаження даних у сховище даних. Тому вcі таблиці в сховищі тепер заповнені:

A computer code on a black background

Description automatically generated  
Рисунок 3.42 – Функція завантаження даних у сховище

1. За допомогою функції insert\_data() було завантажено circuits у створену таблицю виміру circuits (рисунок 3.43).

A screenshot of a computer

Description automatically generated

Рисунок 3.43 - Завантаження даних у таблицю circuits

1. A screenshot of a computer

   Description automatically generatedЗа допомогою функції insert\_data() було завантажено constructors у створену таблицю виміру constructors (рисунок 3.44)

Рисунок 3.44 - Завантаження даних у таблицю constructors

1. За допомогою функції insert\_data() було завантажено countries у створену таблицю виміру countries (рисунок 3.45).

A screenshot of a computer

Description automatically generated

Рисунок 3.45 - Завантаження даних у таблицю countries

1. За допомогою функції insert\_data() було завантажено drivers у створену таблицю виміру drivers (рисунок 3.46).

A screenshot of a computer

Description automatically generated

Рисунок 3.46 - Завантаження даних у таблицю drivers

1. За допомогою функції insert\_data() було завантажено grand\_prix у створену таблицю виміру grand\_prix (рисунок 3.47).

A screenshot of a computer

Description automatically generated

Рисунок 3.47 - Завантаження даних у таблицю grand\_prix

1. За допомогою функції insert\_data() було завантажено races у створену таблицю виміру races (рисунок 3.48).

A screenshot of a computer

Description automatically generated

Рисунок 3.48 - Завантаження даних у таблицю races

1. За допомогою функції insert\_data() було завантажено races\_constructor\_standings у створену таблицю виміру races\_constructor\_standings (рисунок 3.49).

A screenshot of a computer

Description automatically generated

Рисунок 3.49 - Завантаження даних у таблицю races\_constructor\_standings

1. За допомогою функції insert\_data() було завантажено races\_dod\_results у створену таблицю виміру races\_dod\_results (рисунок 3.50).

A screenshot of a computer

Description automatically generated

Рисунок 3.50 - Завантаження даних у таблицю races\_dod\_results

1. За допомогою функції insert\_data() було завантажено races\_driver\_standings у створену таблицю виміру races\_driver\_standings (рисунок 3.51).

A screenshot of a computer

Description automatically generated

Рисунок 3.51 - Завантаження даних у таблицю races\_driver\_standings

1. За допомогою функції insert\_data() було завантажено races\_fp1\_results у створену таблицю виміру races\_fp1\_results (рисунок 3.52).

A screenshot of a computer

Description automatically generated

Рисунок 3.52 - Завантаження даних у таблицю races\_fp1\_results

1. За допомогою функції insert\_data() було завантажено races\_fp2\_results у створену таблицю виміру races\_fp2\_results (рисунок 3.53).

A screenshot of a computer

Description automatically generated

Рисунок 3.53 - Завантаження даних у таблицю races\_fp2\_results

1. За допомогою функції insert\_data() було завантажено races\_fp3\_results у створену таблицю виміру races\_fp3\_results (рисунок 3.54).

A screenshot of a computer

Description automatically generated

Рисунок 3.54 - Завантаження даних у таблицю races\_fp3\_results

1. За допомогою функції insert\_data() було завантажено races\_pit\_stops у створену таблицю виміру races\_pit\_stops (рисунок 3.55).

A screenshot of a computer

Description automatically generated

Рисунок 3.55 - Завантаження даних у таблицю races\_pit\_stops

1. За допомогою функції insert\_data() було завантажено races\_qualifying\_results у створену таблицю виміру races\_qualifying\_results (рисунок 3.56).

A screenshot of a computer

Description automatically generated

Рисунок 3.56 - Завантаження даних у таблицю races\_qualifying\_results

1. Оскільки ми використвуємо дані з бібліотеки пайтон fastf1, то немає потреби імпортувати інформацію races\_race\_lap\_times в сховище, тому можливість продемонструвати є й у вигляді зчитування в датафрейм датасету (рисунок 3.57).

A screenshot of a black screen

Description automatically generated

Рисунок 3.57 - Вміст races\_race\_lap\_times

1. За допомогою функції insert\_data() було завантажено races\_race\_results у створену таблицю виміру races\_race\_results (рисунок 3.57).

A screenshot of a computer

Description automatically generated

Рисунок 3.57 - Завантаження даних у таблицю races\_race\_results

1. За допомогою функції insert\_data() було завантажено races\_sprint\_qualifying\_results у створену таблицю виміру races\_sprint\_qualifying\_results (рисунок 3.58).

A screenshot of a computer

Description automatically generated

Рисунок 3.58 - Завантаження даних у таблицю races\_sprint\_qualifying\_results

1. Оскільки ми використвуємо дані з бібліотеки пайтон fastf1, то немає потреби імпортувати інформацію races\_sprint\_race\_lap\_times в сховище, тому можливість продемонструвати є й у вигляді зчитування в датафрейм датасету (рисунок 3.59).

A screenshot of a computer

Description automatically generated

Рисунок 3.59 – Вміст races\_sprint\_race\_lap\_times

1. За допомогою функції insert\_data() було завантажено races\_sprint\_race\_results у створену таблицю виміру races\_sprint\_race\_results (рисунок 3.60).

A screenshot of a computer

Description automatically generated

Рисунок 3.60 - Завантаження даних у таблицю races\_sprint\_race\_results

1. За допомогою функції insert\_data() було завантажено seasons у створену таблицю виміру seasons (рисунок 3.61).

A screenshot of a table

Description automatically generated

Рисунок 3.61 - Завантаження даних у таблицю seasons

1. За допомогою функції insert\_data() було завантажено seasons\_constructor\_standings у створену таблицю виміру seasons\_constructor\_standings (рисунок 3.62).

A screenshot of a computer

Description automatically generated

Рисунок 3.62 - Завантаження даних у таблицю seasons\_constructor\_standings

1. За допомогою функції insert\_data() було завантажено seasons\_driver\_standings у створену таблицю виміру seasons\_driver\_standings (рисунок 3.63).

A screenshot of a table

Description automatically generated

Рисунок 3.63 - Завантаження даних у таблицю seasons\_driver\_standings

# **4.ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ**

4.1 Обґрунтування вибору методів інтелектуального аналізу даних

В даній курсовій роботі для прогнозування результатів перегонів чемпіонату Формули 1 було обрано кілька методів машинного навчання: k-найближчих сусідів (K-Nearest Neighbors, KNN), випадковий ліс (Random Forest), логістична регресія (Logistic Regression) та метод опорних векторів (Support Vector Machine, SVM).

Вибір цих методів був здійснений на основі кількох ключових факторів, що полегшують проведення прогнозування та забезпечують точні результати:

1. **K-найближчих сусідів (KNN)**:
   * **Простота та ефективність**: KNN є інтуїтивно зрозумілим і легко реалізується методом. Він підходить для задач з невеликою кількістю змінних та високою кількістю зразків.
   * **Гнучкість**: Метод дозволяє легко адаптуватися до змін у даних і не вимагає складних налаштувань.
   * **Висока точність на невеликих наборах даних**: KNN показує добрі результати при прогнозуванні на невеликих вибірках, що важливо для аналізу окремих етапів перегонів.
2. **Випадковий ліс (Random Forest)**:
   * **Стабільність та точність**: Випадковий ліс об'єднує результати багатьох дерев рішень, що зменшує ризик переобучення та покращує точність прогнозів.
   * **Обробка великої кількості змінних**: Метод добре справляється з високорозмірними даними, що є типовим для аналізу перегонів, де враховується безліч факторів, таких як погодні умови, технічні характеристики автомобілів, показники пілотів тощо.
   * **Оцінка важливості змінних**: Випадковий ліс дозволяє визначати, які змінні найбільше впливають на результат перегонів, що корисно для подальшого аналізу.
3. **Логістична регресія (Logistic Regression)**:
   * **Інтерпретованість**: Логістична регресія надає зрозумілі коефіцієнти, які показують вплив кожної змінної на ймовірність події (наприклад, виграшу перегонів).
   * **Ефективність при двокласових задачах**: Цей метод ідеально підходить для задач, де необхідно передбачити ймовірність двох можливих результатів, таких як виграш чи програш.
   * **Низька вимогливість до ресурсів**: Логістична регресія є менш вимогливою до обчислювальних ресурсів порівняно з іншими методами, що дозволяє швидко отримувати прогнози.
4. **Метод опорних векторів (SVM)**:
   * **Висока точність**: SVM забезпечує високий рівень точності при роботі з великими наборами даних та складними структурами.
   * **Ефективність у багатовимірних просторах**: Метод добре справляється з даними високої розмірності, що актуально для прогнозування результатів перегонів з багатьма змінними.
   * **Робота з нелінійними даними**: Використання різних ядер (наприклад, поліноміальних або радіальних базисних функцій) дозволяє SVM обробляти нелінійні залежності в даних, що може бути важливим для моделювання складних взаємодій між змінними.

4.2 Математична модель прогнозування часових проміжків

Для дослідження часових рядів головними чинниками є опис та моделювання їх структури. Побудована модель може використовуватися для екстраполяції або прогнозування часового ряду, що є ключовим аспектом у визначенні результатів перегонів чемпіонату Формули 1. У цій курсовій роботі були використані різні математичні моделі для прогнозування результатів перегонів, а саме k-найближчих сусідів (KNN), випадковий ліс (Random Forest), логістична регресія (Logistic Regression) та метод опорних векторів (SVM).

**Модель k-найближчих сусідів (KNN)**

Модель k-найближчих сусідів (KNN) базується на знаходженні k найближчих зразків у просторі ознак та класифікації на основі їхніх значень. Формально, для нової точки x, прогнозується значення y на основі середнього (або найчастішого) значення серед найближчих сусідів:

Зображення, що містить Шрифт, Графіка, символ, дизайн

Автоматично згенерований опис

Рисунок 4.1 – Формула розрахунку для методу KNN

де y(i) – значення найближчих сусідів. Вибір параметру k здійснюється на основі крос-валідації, щоб забезпечити баланс між обчислювальною складністю та точністю моделі.

**Випадковий ліс (Random Forest)**

Випадковий ліс складається з множини дерев рішень, де кожне дерево рішень створюється на основі випадкової вибірки з навчальної множини. Прогноз для нового зразка робиться шляхом усереднення (для регресії) або голосування (для класифікації) результатів окремих дерев:

Зображення, що містить Шрифт, годинник, типографія, дизайн

Автоматично згенерований опис

Рисунок 4.2 – Формула розрахунку для методу Random Forest

де B – кількість дерев у лісі, а T\_b(x) – прогноз окремого дерева.

**Логістична регресія (Logistic Regression)**

Логістична регресія використовується для прогнозування ймовірності належності до певного класу. Модель має вигляд:

Зображення, що містить текст, Шрифт, знімок екрана

Автоматично згенерований опис

Рисунок 4.3 – Формула розрахунку для методу Logistic Regression

де beta\_0, beta\_1, ldots, beta\_p – параметри моделі, що оцінюються методом максимальної правдоподібності. Вона добре підходить для бінарних класифікаційних задач і надає інтерпретовані коефіцієнти.

**Метод опорних векторів (SVM)**

Метод опорних векторів (SVM) шукає гіперплощину, що максимально розділяє класи в багатовимірному просторі. Функція рішення має вигляд:

Зображення, що містить Шрифт, почерк, текст, типографія

Автоматично згенерований опис

Рисунок 4.4 – Формула розрахунку для методу SVM

де alpha\_i – вагові коефіцієнти, y\_i – класові мітки, K(x\_i, x) – ядро (наприклад, лінійне, поліноміальне, RBF), а b – зміщення. SVM ефективний для задач з високою розмірністю та нелінійними розділеннями.

Обрані моделі для прогнозування результатів перегонів чемпіонату Формули 1 дають змогу аналізувати складні залежності та взаємодії між різними змінними.

4.3 Практичне застосування моделей прогнозування часових проміжків

**Прогнозування результатів перегонів за допомогою методу K-nearest Neighbours**

Спочатку були завантажені необхідні дані для аналізу та прогнозування зі сховища за допомогою функції get\_data, у якій на вхід приймається номер сезону (рік), за даними якого виконуватимуться усі подальші дії.

def get\_data(*season*):

    query = f"""

        SELECT

            d.id AS driver\_id,

            r.round AS round,

            d.name AS driver\_name,

            fp1.position\_number AS fp1\_result,

            fp2.position\_number AS fp2\_result,

            fp3.position\_number AS fp3\_result,

            q.position\_number AS q\_result,

            rr.grid\_position\_number AS grid\_position,

            rr.position\_number AS finish\_position

        FROM

            drivers d

        JOIN

            races\_fp1\_results fp1 ON d.id = fp1.driver\_id

        JOIN

            races\_fp2\_results fp2 ON d.id = fp2.driver\_id

        JOIN

            races\_fp3\_results fp3 ON d.id = fp3.driver\_id

        JOIN

            races\_qualifying\_results q ON d.id = q.driver\_id

        JOIN

            races\_race\_results rr ON d.id = rr.driver\_id

        JOIN

            races r ON fp1.race\_id = r.id AND fp2.race\_id = r.id AND fp3.race\_id = r.id AND q.race\_id = r.id AND rr.race\_id = r.id

        JOIN

            seasons s ON s.id = r.season\_id

        WHERE

            s.year = :season

    """

*with* engine.connect() *as* connection:

        df = pd.read\_sql(text(query), connection, *params*={'season': *season*})

*return* df

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Рисунок 4.5 – Результат роботи функції get\_data

Дані були розділені на тренувальні та тестові, предикторами у прогнозах стали результати вільних практик та кваліфікацій, а також стартова позиція у гонці, а прогнозованим значенням була позиція на фініші.

Зображення, що містить знімок екрана, текст, Шрифт

Автоматично згенерований опис

Рисунок 4.6 – Розбиття даних на тренувальні та тестові

Масштабування ознак: Дані масштабуються для поліпшення результатів моделювання. Ознаки нормалізуються до середнього значення 0 та стандартного відхилення 1.

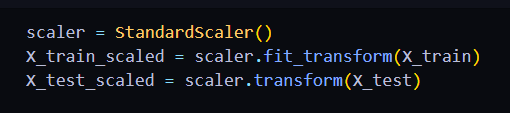


Рисунок 4.7 – Масштабування ознак

Підбір гіперпараметрів (k): Використовуються цикл та крос-валідація для вибору найкращого значення гіперпараметра k для моделі k-найближчих сусідів (KNN).

Зображення, що містить текст, знімок екрана, монітор, програмне забезпечення

Автоматично згенерований опис

Рисунок 4.8 – Підбір гіперпараметрів для KNN

Класифікація та оцінка моделі: Модель KNN навчається на тренувальних даних, після чого здійснюється передбачення на тестових даних. Оцінюється точність моделі за допомогою метрик accuracy\_score, classification\_report та confusion\_matrix.

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

Рисунок 4.9 – Класифікація та оцінка моделі

Точність моделі становить всього 18%, що є досить низьким значенням. Це означає, що модель не дуже ефективна в передбаченні позицій гонщиків у вказаному сезоні.

Проте слід враховувати, що точність рахується по тих значеннях, що точно співали, тобто коли модель спрогнозувала саме ту позицію, на якій гонщик насправді і фінішував.

Якщо модель спрогнозувала фінішну позицію на 1 вище або нижче ніж насправді цей прогноз уже вважається неточним.

Таким чином, враховуючи певні непередбачувані обставини в гонці, які неможливо спрогнозувати, модель непогано справляється з прогнозуванням приблизних очікуваних об'єктивних результатів.

Зображення, що містить текст, знімок екрана, програмне забезпечення

Автоматично згенерований опис

Рисунок 4.10 – Класифікаційний звіт

Оцінка моделі для кожного класу показує нерівномірність результатів. Наприклад, для позиції 2 модель має 100% точність, але для позицій 4, 5, 10, 11, тощо, вона має точність 0%. Це вказує на недостатню ефективність моделі в передбаченні певних класів позицій.

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Рисунок 4.11 – Матриця плутанини

За матрицею плутанини видно, що модель має тенденцію переважати деякі класи позицій (наприклад, позицію 1 та позицію 6), тоді як для інших класів вона здійснює менш точні передбачення.

Зображення, що містить текст, ряд, знімок екрана, Графік

Автоматично згенерований опис

Рисунок 4.12 – Графік розсіювання

Побудований графік розсіювання демонструє відповідність між фактичними та передбаченими позиціями гонщиків. Кожна точка на графіку відображає позицію у фактичному та передбаченому результатах. Червона пунктирна лінія відображає ідеальну відповідність, де фактична позиція рівна передбаченій.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

Рисунок 4.13 – Візуалізація таблиці

У даному випадку, виконується фільтрування та сортування даних для конкретного раунду гонки. Виводиться таблиця з даними про ім'я гонщика, номер раунду, фактичну та передбачену позиції.

Отже, в цілому, можна сказати, що модель потребує додаткового налаштування та покращень для досягнення кращих результатів у передбаченні позицій гонщиків.

**Прогнозування результатів перегонів за допомогою методу Random Forest**

Так само, як і раніше були використані дані зі сховища, розбиті на тренувальні та тестові, а також було проведено масштабування ознак.

Підбір гіперпараметрів (k): Використовуються цикл та крос-валідація для вибору найкращого значення гіперпараметра k для моделі Random Forest.

Зображення, що містить текст, знімок екрана, монітор, програмне забезпечення

Автоматично згенерований опис

Рисунок 4.14 – Підбір гіперпараметрів для Random Forest

Класифікація та оцінка моделі: Модель Random Forest навчається на тренувальних даних, після чого здійснюється передбачення на тестових даних. Оцінюється точність моделі за допомогою метрик accuracy\_score, classification\_report та confusion\_matrix.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Шрифт

Автоматично згенерований опис

Рисунок 4.15 – Класифікація та оцінка моделі

Точність моделі становить всього 17%, що є досить низьким значенням. Це означає, що модель не дуже ефективна в передбаченні позицій гонщиків у вказаному сезоні.

Зображення, що містить текст, знімок екрана, меню

Автоматично згенерований опис

Рисунок 4.16 – Класифікаційний звіт

Зображення, що містить знімок екрана, чорно-білий, візерунок, текст

Автоматично згенерований опис

Рисунок 4.17 – Матриця плутанини

Зображення, що містить текст, знімок екрана, ряд, Графік

Автоматично згенерований опис

Рисунок 4.18 – Графік розсіювання

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Рисунок 4.19 – Візуалізація таблиці

За результатами моделі Random Forest ми бачимо, що точність моделі також складає всього 17%, що є низьким значенням. Класифікаційний звіт також показує низькі значення precision, recall та f1-score для більшості класів позицій гонщиків. Матриця плутанини демонструє розподіл передбачених та дійсних класів, підтверджуючи низьку точність моделі у передбаченні позицій. Отже, як і в попередньому випадку, модель Random Forest також потребує додаткового налаштування та покращень для досягнення кращих результатів.

**Прогнозування результатів перегонів за допомогою методу Logistic Regression**

Так само, як і раніше були використані дані зі сховища, розбиті на тренувальні та тестові, а також було проведено масштабування ознак.

Класифікація та оцінка моделі: Модель Logistic Regression навчається на тренувальних даних, після чого здійснюється передбачення на тестових даних. Оцінюється точність моделі за допомогою метрик accuracy\_score, classification\_report та confusion\_matrix.

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

Рисунок 4.20 – Класифікація та оцінка моделі

Точність моделі становить всього 8%, що є низьким значенням. Це означає, що модель не дуже ефективна в передбаченні позицій гонщиків у вказаному сезоні.

Зображення, що містить текст, меню, знімок екрана

Автоматично згенерований опис

Рисунок 4.21 – Класифікаційний звіт

Зображення, що містить знімок екрана, візерунок, чорно-білий, дизайн

Автоматично згенерований опис

Рисунок 4.22 – Матриця плутанини

Зображення, що містить текст, ряд, Графік, схема

Автоматично згенерований опис

Рисунок 4.23 – Графік розсіювання

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Рисунок 4.24 – Візуалізація таблиці

Отже, результати для логістичної регресії не виглядають дуже обнадійливо. Точність моделі дуже низька (0.08), що означає, що лише 8% передбачень були правильними. Крім того, показники precision, recall та f1-score для більшості класів також дуже низькі. Матриця плутанини показує, що модель в основному робить неправильні передбачення для більшості класів.

**Прогнозування результатів перегонів за допомогою методу SVM (Support Vector Machines)**

Так само, як і раніше були використані дані зі сховища, розбиті на тренувальні та тестові, а також було проведено масштабування ознак.

Класифікація та оцінка моделі: Модель SVM навчається на тренувальних даних, після чого здійснюється передбачення на тестових даних. Оцінюється точність моделі за допомогою метрик accuracy\_score, classification\_report та confusion\_matrix.

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

Рисунок 4.25 – Класифікація та оцінка моделі

Точність (Accuracy): 0.12, що вказує на те, що модель досягає правильної класифікації лише приблизно 12% випадків.

Зображення, що містить текст, знімок екрана, меню

Автоматично згенерований опис

Рисунок 4.26 – Класифікаційний звіт

Класифікаційний звіт (Classification Report): У класифікаційному звіті можна побачити, що модель показує низькі показники precision, recall та f1-score для більшості класів. Це означає, що модель має проблеми з правильною класифікацією більшості класів.

Зображення, що містить знімок екрана, візерунок, чорно-білий, текст

Автоматично згенерований опис

Рисунок 4.27 – Матриця плутанини

Матриця помилок (Confusion Matrix): Матриця помилок також відображає слабкість моделі, оскільки багато значень потрапили на діагональ, що вказує на неправильну класифікацію.

Зображення, що містить текст, ряд, Графік, схема

Автоматично згенерований опис

Рисунок 4.28 – Графік розсіювання

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Рисунок 4.29 – Візуалізація таблиці

Загалом, ці результати свідчать про те, що модель SVM, побудована на цих даних із застосуванням лінійного ядра, може бути недостатньою для ефективної класифікації позицій у гонці.

# **ВИСНОВКИ**

У процесі виконання курсової роботи було досягнуто декількох важливих результатів у сфері прогнозування та аналізу даних у контексті автоспорту, зокрема серії перегонів Формула 1. Основними досягненнями можна виділити наступні:

Розробка сховища даних: Було спроектовано модель сховища даних типу «сніжинка», що включає 15 таблиць фактів та 7 таблиць вимірів. Це дозволяє ефективно зберігати та обробляти великі обсяги історичних даних про результати перегонів.

Створення ETL процесів: Було розроблено ETL процеси для завантаження та оновлення даних у сховище. Це забезпечує актуальність і повноту даних, необхідних для проведення подальшого аналітичного дослідження.

Інтелектуальний аналіз даних: Проведено детальний аналіз даних із використанням методів інтелектуального аналізу, таких як перевірка на сезонність, стаціонарність та шум, а також згладжування даних. Це дозволило виявити приховані тенденції та шаблони у даних.

Використання моделей прогнозування: Було застосовано різні моделі прогнозування, включаючи k-nearest neighbours (KNN) та random forest, для точного передбачення результатів перегонів та оцінки впливу різних факторів. Це сприяло підвищенню точності прогнозів та дозволило краще зрозуміти взаємозв'язки між змінними.

Графічна візуалізація результатів: Результати аналізу було представлено у вигляді графіків та діаграм, що допомогло візуалізувати дані та зробити висновки більш наочними та зрозумілими.

Загалом, виконана робота продемонструвала ефективність використання сучасних методів обробки та аналізу даних для прогнозування результатів у спортивних змаганнях.

# **ПЕРЕЛІК ПОСИЛАНЬ**

Python Software Foundation. Python Documentation. URL: <https://docs.python.org/3/>

The pandas development team. pandas Documentation. URL: <https://pandas.pydata.org/docs/>

The fastf1 development team. fastf1 Documentation. URL: <https://docs.fastf1.dev/>

The SQLAlchemy Authors. SQLAlchemy Documentation. URL: <https://docs.sqlalchemy.org/en/20/>

The Matplotlib Development Team. Matplotlib Documentation. URL: <https://matplotlib.org/stable/index.html>

The scikit-learn developers. scikit-learn Documentation. URL: <https://scikit-learn.org/stable/>

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1), 21-27.

**Додаток А Тексти програмного коду**

*студентів групи ІП-21 ІІ курсу*

*Загребельного О.А.*

*Голованьова Г.О.*

(Обсяг програми (документа), арк., Кб)

*28 арк, 64 Кб*

(Вид носія даних)

*SSD*

(Найменування програми (документа))

*Тексти програмного коду* прогнозування результатів перегонів чемпіонату Формула-1

**Посилання на репозиторій GitHub**

<https://github.com/zagrebelnio/F1-Championship-Data-Analysis>

**Робота зі сховищем**

create\_storage.ipynb

*from* dotenv *import* dotenv\_values

*import* psycopg2

*from* sqlalchemy *import* create\_engine, text

config = dotenv\_values()

DB\_NAME = config.get('DB\_NAME')

DB\_USER = config.get('DB\_USER')

DB\_HOST = config.get('DB\_HOST')

DB\_PASSWORD = config.get('DB\_PASSWORD')

DB\_PORT = config.get('DB\_PORT')

DB\_URL\_NO\_DB = f"postgresql://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/"

*try*:

    con = psycopg2.connect(

*dbname*='postgres',

*user*=DB\_USER,

*password*=DB\_PASSWORD,

*host*=DB\_HOST,

*port*=DB\_PORT

    )

    con.autocommit = True

    cur = con.cursor()

    cur.execute(f"CREATE DATABASE {DB\_NAME}")

    cur.close()

    con.close()

    print(f"Database '{DB\_NAME}' created successfully.")

*except* psycopg2.errors.DuplicateDatabase:

    print(f"Database '{DB\_NAME}' already exists.")

*except* Exception *as* e:

    print(f"Error creating database '{DB\_NAME}': {e}")

DB\_URL = f"postgresql://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}"

engine = create\_engine(DB\_URL)

*with* open('create\_db.sql', 'r') *as* f:

    create\_sql = f.read()

sql\_commands = create\_sql.split(';')

sql\_commands.pop()

*with* engine.connect() *as* connection:

*try*:

        transaction = connection.begin()

*for* script *in* sql\_commands:

            connection.execute(text(script))

        transaction.commit()

        print("SQL script executed successfully.")

*except* Exception *as* e:

        transaction.rollback()

        print(f"Error executing SQL script: {e}")

dimension\_tables.ipynb

*import* pandas *as* pd

*from* dotenv *import* dotenv\_values

*from* sqlalchemy *import* create\_engine, text

*import* os

config = dotenv\_values()

DB\_NAME = config.get('DB\_NAME')

DB\_USER = config.get('DB\_USER')

DB\_HOST = config.get('DB\_HOST')

DB\_PASSWORD = config.get('DB\_PASSWORD')

DB\_PORT = config.get('DB\_PORT')

engine = create\_engine(f'postgresql+psycopg2://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}')

countries\_df = pd.read\_csv('../data/f1db-countries.csv')

countries\_df.head()

countries\_df = countries\_df[['alpha3Code', 'name', 'demonym', 'id']]

countries\_df.head()

countries\_df.columns = ['alpha3\_code', 'name', 'demonym', 'id']

countries\_df.head()

def insert\_data(*df*, *table*):

*with* engine.connect() *as* connection:

        result = connection.execute(text(f"SELECT COUNT(\*) FROM {*table*}"))

        count = result.scalar()

*if* count == 0:

*df*.to\_sql(*table*, engine, *if\_exists*='append', *index*=False)

        result = connection.execute(text(f"SELECT id FROM {*table*}"))

        generated\_ids = [row[0] *for* row *in* result]

*return* generated\_ids

countries\_df['dbId'] = insert\_data(countries\_df[['alpha3\_code', 'name', 'demonym']], 'countries')

countries\_df.head()

circuits\_df = pd.read\_csv('../data/f1db-circuits.csv')

circuits\_df.head()

circuits\_df['countryId'] = circuits\_df.merge(countries\_df[['id', 'dbId']], *left\_on*='countryId', *right\_on*='id', *how*='left')['dbId']

circuits\_df.head()

circuits\_df = circuits\_df[['name', 'fullName', 'type', 'countryId', 'latitude', 'longitude', 'id']]

circuits\_df.head()

circuits\_df.columns = ['name', 'full\_name', 'type', 'country\_id', 'latitude', 'longitude', 'id']

circuits\_df.head()

circuits\_df['dbId'] = insert\_data(circuits\_df[['name', 'full\_name', 'type', 'country\_id', 'latitude', 'longitude']], 'circuits')

circuits\_df.head()

constructors\_df = pd.read\_csv('../data/f1db-constructors.csv')

constructors\_df.head()

constructors\_df['countryId'] = constructors\_df.merge(countries\_df[['id', 'dbId']], *left\_on*='countryId', *right\_on*='id', *how*='left')['dbId']

constructors\_df.head()

constructors\_df = constructors\_df[['name', 'fullName', 'countryId', 'id']]

constructors\_df.head()

constructors\_df.columns = ['name', 'full\_name', 'country\_id', 'id']

constructors\_df.head()

constructors\_df['dbId'] = insert\_data(constructors\_df[['name', 'full\_name', 'country\_id']], 'constructors')

constructors\_df.head()

drivers\_df = pd.read\_csv('../data/f1db-drivers.csv')

drivers\_df.head()

drivers\_df['nationalityCountryId'] = drivers\_df.merge(countries\_df[['id', 'dbId']], *left\_on*='nationalityCountryId', *right\_on*='id', *how*='left')['dbId']

drivers\_df[['nationalityCountryId']].head()

drivers\_df = drivers\_df[['name', 'firstName', 'lastName', 'fullName', 'abbreviation', 'permanentNumber', 'gender', 'dateOfBirth', 'nationalityCountryId', 'id']]

drivers\_df.head()

drivers\_df.columns = ['name', 'first\_name', 'last\_name', 'full\_name', 'abbreviation', 'permanent\_number', 'gender', 'date\_of\_birth', 'nationality\_country\_id', 'id']

drivers\_df.head()

drivers\_df['dbId'] = insert\_data(drivers\_df[['name', 'first\_name', 'last\_name', 'full\_name', 'abbreviation', 'permanent\_number', 'gender', 'date\_of\_birth', 'nationality\_country\_id']], 'drivers')

drivers\_df.head()

grand\_prix\_df = pd.read\_csv('../data/f1db-grands-prix.csv')

grand\_prix\_df.head()

grand\_prix\_df['countryId'] = grand\_prix\_df.merge(countries\_df[['id', 'dbId']], *left\_on*='countryId', *right\_on*='id', *how*='left')['dbId'].astype('Int64')

grand\_prix\_df.head()

grand\_prix\_df = grand\_prix\_df[['name', 'fullName', 'shortName', 'abbreviation', 'countryId', 'id']]

grand\_prix\_df.head()

grand\_prix\_df.columns = ['name', 'full\_name', 'short\_name', 'abbreviation', 'country\_id', 'id']

grand\_prix\_df.head()

grand\_prix\_df['dbId'] = insert\_data(grand\_prix\_df[['name', 'full\_name', 'short\_name', 'abbreviation', 'country\_id']], 'grand\_prix')

grand\_prix\_df.head()

seasons\_df = pd.read\_csv('../data/f1db-seasons.csv')

seasons\_df.head()

seasons\_df['dbId'] = insert\_data(seasons\_df, 'seasons')

seasons\_df.head()

races\_df = pd.read\_csv('../data/f1db-races.csv')

races\_df.head()

races\_df['year'] = races\_df.merge(seasons\_df, *on*='year', *how*='left')['dbId'].astype('Int64')

races\_df.head()

races\_df['grandPrixId'] = races\_df.merge(grand\_prix\_df[['id', 'dbId']], *left\_on*='grandPrixId', *right\_on*='id', *how*='left')['dbId'].astype('Int64')

races\_df.head()

races\_df['circuitId'] = races\_df.merge(circuits\_df[['id', 'dbId']], *left\_on*='circuitId', *right\_on*='id', *how*='left')['dbId'].astype('Int64')

races\_df.head()

races\_df = races\_df[['id', 'year', 'round', 'date', 'grandPrixId', 'officialName', 'circuitId', 'courseLength', 'laps', 'distance']]

races\_df.head()

races\_df.columns = ['id', 'season\_id', 'round', 'date', 'grand\_prix\_id', 'official\_name', 'circuit\_id', 'course\_length', 'laps', 'distance']

races\_df.head()

insert\_data(races\_df, 'races')

races\_df.head()

columns = ['id', 'dbId']

seasons\_df.columns = columns

folder = '../adapted\_data/'

dfs\_to\_save = {

    'countries': countries\_df,

    'circuits': circuits\_df,

    'constructors': constructors\_df,

    'drivers': drivers\_df,

    'grand\_prix': grand\_prix\_df,

    'seasons': seasons\_df

}

*for* file\_name, df *in* dfs\_to\_save.items():

    file\_path = os.path.join(folder, f"{file\_name}Id.csv")

    df[columns].to\_csv(file\_path, *index*=False)

fact\_tables.ipynb

*import* pandas *as* pd

*from* dotenv *import* dotenv\_values

*from* sqlalchemy *import* create\_engine, text

*import* os

config = dotenv\_values()

DB\_NAME = config.get('DB\_NAME')

DB\_USER = config.get('DB\_USER')

DB\_HOST = config.get('DB\_HOST')

DB\_PASSWORD = config.get('DB\_PASSWORD')

DB\_PORT = config.get('DB\_PORT')

engine = create\_engine(f'postgresql+psycopg2://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}')

def insert\_data(*df*, *table*):

*with* engine.connect() *as* connection:

        result = connection.execute(text(f"SELECT COUNT(\*) FROM {*table*}"))

        count = result.scalar()

*if* count == 0:

*df*.to\_sql(*table*, engine, *if\_exists*='append', *index*=False)

circuits\_id\_df = pd.read\_csv('../adapted\_data/circuitsId.csv')

constructors\_id\_df = pd.read\_csv('../adapted\_data/constructorsId.csv')

countries\_id\_df = pd.read\_csv('../adapted\_data/countriesId.csv')

drivers\_id\_df = pd.read\_csv('../adapted\_data/driversId.csv')

grand\_prix\_id\_df = pd.read\_csv('../adapted\_data/grand\_prixId.csv')

seasons\_id\_df = pd.read\_csv('../adapted\_data/seasonsId.csv')

circuits\_id\_df.head()

races\_fp1\_results\_df = pd.read\_csv('../data/f1db-races-free-practice-1-results.csv')

races\_fp1\_results\_df.head()

races\_fp1\_results\_df['driver\_id'] = races\_fp1\_results\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

races\_fp1\_results\_df['constructor\_id'] = races\_fp1\_results\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

races\_fp1\_results\_df = races\_fp1\_results\_df[['raceId', 'driver\_id', 'constructor\_id', 'positionDisplayOrder', 'positionNumber', 'time', 'timeMillis', 'gap', 'gapMillis', 'interval', 'intervalMillis', 'laps']]

races\_fp1\_results\_df.head()

races\_fp1\_results\_df.columns = ['race\_id', 'driver\_id', 'constructor\_id', 'position\_display\_order', 'position\_number', 'time', 'time\_millis', 'gap', 'gap\_millis', 'interval', 'interval\_millis', 'laps']

insert\_data(races\_fp1\_results\_df, 'races\_fp1\_results')

races\_fp2\_results\_df = pd.read\_csv('../data/f1db-races-free-practice-2-results.csv')

races\_fp2\_results\_df.head()

races\_fp2\_results\_df['driver\_id'] = races\_fp2\_results\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

races\_fp2\_results\_df['constructor\_id'] = races\_fp2\_results\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

races\_fp2\_results\_df = races\_fp2\_results\_df[['raceId', 'driver\_id', 'constructor\_id', 'positionDisplayOrder', 'positionNumber', 'time', 'timeMillis', 'gap', 'gapMillis', 'interval', 'intervalMillis', 'laps']]

races\_fp2\_results\_df.head()

races\_fp2\_results\_df.columns = ['race\_id', 'driver\_id', 'constructor\_id', 'position\_display\_order', 'position\_number', 'time', 'time\_millis', 'gap', 'gap\_millis', 'interval', 'interval\_millis', 'laps']

insert\_data(races\_fp2\_results\_df, 'races\_fp2\_results')

races\_fp3\_results\_df = pd.read\_csv('../data/f1db-races-free-practice-3-results.csv')

races\_fp3\_results\_df.head()

races\_fp3\_results\_df['driver\_id'] = races\_fp3\_results\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

races\_fp3\_results\_df['constructor\_id'] = races\_fp3\_results\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

races\_fp3\_results\_df = races\_fp3\_results\_df[['raceId', 'driver\_id', 'constructor\_id', 'positionDisplayOrder', 'positionNumber', 'time', 'timeMillis', 'gap', 'gapMillis', 'interval', 'intervalMillis', 'laps']]

races\_fp3\_results\_df.head()

races\_fp3\_results\_df.columns = ['race\_id', 'driver\_id', 'constructor\_id', 'position\_display\_order', 'position\_number', 'time', 'time\_millis', 'gap', 'gap\_millis', 'interval', 'interval\_millis', 'laps']

insert\_data(races\_fp3\_results\_df, 'races\_fp3\_results')

races\_qualifying\_results\_df = pd.read\_csv('../data/f1db-races-qualifying-results.csv')

races\_qualifying\_results\_df.head()

races\_qualifying\_results\_df['driver\_id'] = races\_qualifying\_results\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

races\_qualifying\_results\_df['constructor\_id'] = races\_qualifying\_results\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

races\_qualifying\_results\_df = races\_qualifying\_results\_df[['raceId', 'driver\_id', 'constructor\_id', 'positionDisplayOrder', 'positionNumber', 'q1', 'q1Millis', 'q2', 'q2Millis', 'q3', 'q3Millis', 'gap', 'gapMillis', 'interval', 'intervalMillis', 'laps']]

races\_qualifying\_results\_df.head()

races\_qualifying\_results\_df.columns = ['race\_id', 'driver\_id', 'constructor\_id', 'position\_display\_order', 'position\_number', 'q1', 'q1\_millis', 'q2', 'q2\_millis', 'q3', 'q3\_millis', 'gap', 'gap\_millis', 'interval', 'interval\_millis', 'laps']

insert\_data(races\_qualifying\_results\_df, 'races\_qualifying\_results')

races\_race\_results\_df = pd.read\_csv('../data/f1db-races-race-results.csv')

races\_race\_results\_df.head()

races\_race\_results\_df.info()

races\_race\_results\_df['driver\_id'] = races\_race\_results\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

races\_race\_results\_df['constructor\_id'] = races\_race\_results\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

races\_race\_results\_df = races\_race\_results\_df[['raceId', 'driver\_id', 'constructor\_id', 'positionDisplayOrder', 'positionNumber', 'laps', 'time', 'timeMillis',

                                                'timePenalty', 'timePenaltyMillis', 'gap', 'gapMillis', 'gapLaps', 'interval','intervalMillis', 'reasonRetired','points','gridPositionNumber',

                                                'positionsGained','fastestLap','pitStops','driverOfTheDay','grandSlam']]

races\_race\_results\_df.head()

races\_race\_results\_df.columns = ['race\_id', 'driver\_id', 'constructor\_id', 'position\_display\_order', 'position\_number', 'laps', 'time', 'time\_millis',

                                       'time\_penalty', 'time\_penalty\_millis', 'gap', 'gap\_millis', 'gap\_laps', 'interval', 'interval\_millis', 'reason\_retired', 'points', 'grid\_position\_number',

                                       'positions\_gained', 'fastest\_lap', 'pit\_stops', 'driver\_of\_the\_day', 'grand\_slam']

insert\_data(races\_race\_results\_df, 'races\_race\_results')

races\_pit\_stops\_df = pd.read\_csv('../data/f1db-races-pit-stops.csv')

races\_pit\_stops\_df.head()

races\_pit\_stops\_df.info()

races\_pit\_stops\_df['driver\_id'] = races\_pit\_stops\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

races\_pit\_stops\_df['constructor\_id'] = races\_pit\_stops\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

races\_pit\_stops\_df = races\_pit\_stops\_df[['raceId', 'driver\_id', 'constructor\_id', 'positionDisplayOrder', 'positionNumber','stop', 'lap', 'time', 'timeMillis']]

races\_pit\_stops\_df.head()

races\_pit\_stops\_df.columns = ['race\_id', 'driver\_id', 'constructor\_id', 'position\_display\_order', 'position\_number','stop', 'lap', 'time', 'time\_millis']

insert\_data(races\_pit\_stops\_df, 'races\_pit\_stops')

races\_sprint\_qualifying\_results\_df = pd.read\_csv('../data/f1db-races-sprint-qualifying-results.csv')

races\_sprint\_qualifying\_results\_df.head()

*#races\_sprint\_qualifying\_results\_df.info(100)*

races\_sprint\_qualifying\_results\_df['driver\_id'] = races\_sprint\_qualifying\_results\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

races\_sprint\_qualifying\_results\_df['constructor\_id'] = races\_sprint\_qualifying\_results\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

races\_sprint\_qualifying\_results\_df = races\_sprint\_qualifying\_results\_df[['raceId','driver\_id', 'constructor\_id', 'positionDisplayOrder', 'positionNumber',

                                                            'q1', 'q1Millis', 'q2', 'q2Millis', 'q3', 'q3Millis', 'gap', 'gapMillis', 'interval',

                                                             'intervalMillis', 'laps']]

races\_sprint\_qualifying\_results\_df.head()

races\_sprint\_qualifying\_results\_df.head()

races\_sprint\_qualifying\_results\_df.columns = ['race\_id', 'driver\_id', 'constructor\_id', 'position\_display\_order', 'position\_number',

                            'q1', 'q1\_millis', 'q2', 'q2\_millis', 'q3', 'q3\_millis', 'gap', 'gap\_millis', 'interval',

                            'interval\_millis', 'laps']

insert\_data(races\_sprint\_qualifying\_results\_df, 'races\_sprint\_qualifying\_results')

races\_sprint\_race\_results\_df = pd.read\_csv('../data/f1db-races-sprint-race-results.csv')

races\_sprint\_race\_results\_df.head()

*#races\_sprint\_race\_results\_df.info(100)*

races\_sprint\_race\_results\_df['driver\_id'] = races\_sprint\_race\_results\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

races\_sprint\_race\_results\_df['constructor\_id'] = races\_sprint\_race\_results\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

races\_sprint\_race\_results\_df = races\_sprint\_race\_results\_df[['raceId', 'driver\_id', 'constructor\_id', 'positionDisplayOrder', 'positionNumber', 'laps',

                                                              'time', 'timeMillis', 'timePenalty', 'timePenaltyMillis', 'gap', 'gapMillis', 'gapLaps', 'interval',

                                                                'intervalMillis', 'reasonRetired', 'points', 'gridPositionNumber', 'positionsGained',

                                                                  'fastestLap', 'pitStops']]

races\_sprint\_race\_results\_df.head()

races\_sprint\_race\_results\_df.head()

races\_sprint\_race\_results\_df.columns = ['race\_id', 'driver\_id', 'constructor\_id', 'position\_display\_order', 'position\_number', 'laps',

                                         'time', 'time\_millis', 'time\_penalty', 'time\_penalty\_millis', 'gap', 'gap\_millis', 'gap\_laps', 'interval',

                                           'interval\_millis', 'reason\_retired', 'points', 'grid\_position\_number', 'positions\_gained', 'fastest\_lap', 'pit\_stops']

insert\_data(races\_sprint\_race\_results\_df, 'races\_sprint\_race\_results')

races\_constructor\_standings\_df = pd.read\_csv('../data/f1db-races-constructor-standings.csv')

races\_constructor\_standings\_df.head()

*#races\_constructor\_standings\_df.info()*

races\_constructor\_standings\_df['constructor\_id'] = races\_constructor\_standings\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

races\_constructor\_standings\_df = races\_constructor\_standings\_df[['raceId', 'constructor\_id', 'positionDisplayOrder', 'positionNumber', 'points', 'positionsGained']]

races\_constructor\_standings\_df['positionNumber'] = races\_constructor\_standings\_df['positionNumber'].astype('Int64')

races\_constructor\_standings\_df.head()

races\_constructor\_standings\_df.columns = ['race\_id', 'constructor\_id', 'position\_display\_order', 'position\_number', 'points', 'positions\_gained']

insert\_data(races\_constructor\_standings\_df, 'races\_constructor\_standings')

races\_dod\_results\_df = pd.read\_csv('../data/f1db-races-driver-of-the-day-results.csv')

races\_dod\_results\_df.head()

*#races\_dod\_results\_df.info(100)*

races\_dod\_results\_df['driver\_id'] = races\_dod\_results\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

races\_dod\_results\_df['constructor\_id'] = races\_dod\_results\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

races\_dod\_results\_df = races\_dod\_results\_df[['raceId', 'driver\_id', 'constructor\_id', 'positionDisplayOrder', 'positionNumber', 'percentage']]

races\_dod\_results\_df.head()

races\_dod\_results\_df.columns = ['race\_id', 'driver\_id', 'constructor\_id', 'position\_display\_order', 'position\_number', 'percentage']

insert\_data(races\_dod\_results\_df, 'races\_dod\_results')

races\_driver\_standings\_df = pd.read\_csv('../data/f1db-races-driver-standings.csv')

races\_driver\_standings\_df.head()

*#races\_driver\_standings\_df.info(100)*

races\_driver\_standings\_df['driver\_id'] = races\_driver\_standings\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

races\_driver\_standings\_df = races\_driver\_standings\_df[['raceId', 'driver\_id', 'positionDisplayOrder', 'positionNumber', 'points','positionsGained']]

races\_driver\_standings\_df.head()

races\_driver\_standings\_df.columns = ['race\_id', 'driver\_id', 'position\_display\_order', 'position\_number', 'points','positions\_gained']

insert\_data(races\_driver\_standings\_df, 'races\_driver\_standings')

seasons\_constructor\_standings\_df = pd.read\_csv('../data/f1db-seasons-constructor-standings.csv')

seasons\_constructor\_standings\_df.head()

seasons\_constructor\_standings\_df['constructor\_id'] = seasons\_constructor\_standings\_df.merge(constructors\_id\_df, *left\_on*='constructorId', *right\_on*='id', *how*='left')['dbId']

seasons\_constructor\_standings\_df['year'] = seasons\_constructor\_standings\_df.merge(seasons\_id\_df, *left\_on*='year', *right\_on*='id', *how*='left')['dbId']

seasons\_constructor\_standings\_df = seasons\_constructor\_standings\_df[['year', 'constructor\_id', 'positionDisplayOrder', 'positionNumber', 'points']]

seasons\_constructor\_standings\_df.columns = ['season\_id', 'constructor\_id', 'position\_display\_order', 'position\_number', 'points']

seasons\_constructor\_standings\_df.head()

insert\_data(seasons\_constructor\_standings\_df, 'seasons\_constructor\_standings')

seasons\_driver\_standings\_df = pd.read\_csv('../data/f1db-seasons-driver-standings.csv')

seasons\_driver\_standings\_df.head()

*#seasons\_driver\_standings\_df.info(100)*

seasons\_driver\_standings\_df['driver\_id'] = seasons\_driver\_standings\_df.merge(drivers\_id\_df, *left\_on*='driverId', *right\_on*='id', *how*='left')['dbId']

seasons\_driver\_standings\_df['year'] = seasons\_driver\_standings\_df.merge(seasons\_id\_df, *left\_on*='year', *right\_on*='id', *how*='left')['dbId']

seasons\_driver\_standings\_df = seasons\_driver\_standings\_df[['year', 'driver\_id', 'positionDisplayOrder', 'positionNumber', 'points']]

seasons\_driver\_standings\_df.head()

seasons\_driver\_standings\_df.columns = ['season\_id', 'driver\_id', 'position\_display\_order', 'position\_number', 'points']

seasons\_driver\_standings\_df.head()

insert\_data(seasons\_driver\_standings\_df, 'seasons\_driver\_standings')

lap\_times.ipynb

*import* fastf1

*import* pandas *as* pd

*from* dotenv *import* dotenv\_values

*from* sqlalchemy *import* create\_engine, text

config = dotenv\_values()

DB\_NAME = config.get('DB\_NAME')

DB\_USER = config.get('DB\_USER')

DB\_HOST = config.get('DB\_HOST')

DB\_PASSWORD = config.get('DB\_PASSWORD')

DB\_PORT = config.get('DB\_PORT')

engine = create\_engine(f'postgresql+psycopg2://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}')

def get\_races(*start\_year*):

*with* engine.connect() *as* connection:

        result = connection.execute(text(f"SELECT \* FROM races WHERE EXTRACT(YEAR FROM date) >= {*start\_year*} AND date <= NOW()"))

        data = result.fetchall()

        columns = result.keys()

*return* pd.DataFrame(data, *columns*=columns)

races\_df = get\_races(2018)

races\_df.head()

def is\_sprint\_weekend(*event*):

*return* *event*.Session3 == 'Sprint' or *event*.Session4 == 'Sprint'

def fetch\_data\_from\_session(*year*, *round*, *type*):

    session = fastf1.get\_session(*year*, *round*, *type*)

    session.load(

*laps* = True,

*telemetry* = False,

*weather* = False,

*messages* = False

    )

    drivers\_info = []

*for* driver *in* session.drivers:

        driver\_info = session.get\_driver(driver)

        drivers\_info.append({

            'DriverNumber': driver\_info['DriverNumber'],

            'FirstName': driver\_info['FirstName'],

            'Abbreviation': driver\_info['Abbreviation'],

        })

    driver\_info\_df = pd.DataFrame(drivers\_info)

    laps\_df = session.laps[['Driver', 'LapTime', 'LapNumber', 'Team', 'DriverNumber']]

    df = laps\_df.merge(driver\_info\_df, *on*='DriverNumber', *how*='left')

*return* df[['LapTime', 'LapNumber', 'Team', 'FirstName', 'Abbreviation']]

def get\_lap\_times(*races\_df*):

    sprint\_lap\_times\_df = pd.DataFrame()

    race\_lap\_times\_df = pd.DataFrame()

*for* index, row *in* *races\_df*.iterrows():

        date = row['date']

        round = row['round']

        year = date.year

        race\_id = row['id']

        event = fastf1.get\_event(*year*=year, *gp*=round)

*if* (is\_sprint\_weekend(event)):

            sprint\_lap\_times = fetch\_data\_from\_session(year, round, 'S')

            sprint\_lap\_times['race\_id'] = race\_id

            sprint\_lap\_times\_df = pd.concat([sprint\_lap\_times\_df, sprint\_lap\_times], *ignore\_index*=True)

        race\_lap\_times = fetch\_data\_from\_session(year, round, 'R')

        race\_lap\_times['race\_id'] = race\_id

        race\_lap\_times\_df = pd.concat([race\_lap\_times\_df, race\_lap\_times], *ignore\_index*=True)

*return* sprint\_lap\_times\_df, race\_lap\_times\_df

sprint\_lap\_times\_df, race\_lap\_times\_df = get\_lap\_times(races\_df)

sprint\_lap\_times\_df.head()

race\_lap\_times\_df.head()

race\_lap\_times\_df.info()

sprint\_lap\_times\_df.info()

race\_lap\_times\_df = race\_lap\_times\_df.dropna(*subset*=['LapTime'])

race\_lap\_times\_df.info()

sprint\_lap\_times\_df = sprint\_lap\_times\_df.dropna(*subset*=['LapTime'])

sprint\_lap\_times\_df.info()

race\_lap\_times\_df['lap\_time\_millis'] = race\_lap\_times\_df['LapTime'].dt.total\_seconds() \* 1000

race\_lap\_times\_df.head()

sprint\_lap\_times\_df['lap\_time\_millis'] = sprint\_lap\_times\_df['LapTime'].dt.total\_seconds() \* 1000

sprint\_lap\_times\_df.head()

race\_lap\_times\_df['LapNumber'] = race\_lap\_times\_df['LapNumber'].astype(int)

race\_lap\_times\_df['lap\_time\_millis'] = race\_lap\_times\_df['lap\_time\_millis'].astype(int)

race\_lap\_times\_df.head()

sprint\_lap\_times\_df['LapNumber'] = sprint\_lap\_times\_df['LapNumber'].astype(int)

sprint\_lap\_times\_df['lap\_time\_millis'] = sprint\_lap\_times\_df['lap\_time\_millis'].astype(int)

sprint\_lap\_times\_df.head()

def timedelta\_to\_string(*td*):

    minutes = *td*.seconds // 60

    seconds = *td*.seconds % 60

    milliseconds = *td*.microseconds // 1000

*return* '{:02d}:{:02d}.{:03d}'.format(minutes, seconds, milliseconds)

race\_lap\_times\_df['lap\_time'] = race\_lap\_times\_df['LapTime'].dt.total\_seconds().apply(lambda *x*: timedelta\_to\_string(pd.Timedelta(*seconds*=*x*)))

race\_lap\_times\_df.head()

sprint\_lap\_times\_df['lap\_time'] = sprint\_lap\_times\_df['LapTime'].dt.total\_seconds().apply(lambda *x*: timedelta\_to\_string(pd.Timedelta(*seconds*=*x*)))

sprint\_lap\_times\_df.head()

def get\_drivers():

*with* engine.connect() *as* connection:

        result = connection.execute(text(f"SELECT \* FROM drivers"))

        data = result.fetchall()

        columns = result.keys()

*return* pd.DataFrame(data, *columns*=columns)

drivers\_df = get\_drivers()

drivers\_df.head()

drivers\_df['combined\_key'] = drivers\_df['first\_name'] + '\_' + drivers\_df['abbreviation']

drivers\_dict = drivers\_df.set\_index('combined\_key')['id'].to\_dict()

race\_lap\_times\_df['combined\_key'] = race\_lap\_times\_df['FirstName'] + '\_' + race\_lap\_times\_df['Abbreviation']

race\_lap\_times\_df['driver\_id'] = pd.NA

race\_lap\_times\_df['driver\_id'] = race\_lap\_times\_df.apply(

    lambda *row*: drivers\_dict.get(*row*['combined\_key']) *if* pd.isnull(*row*['driver\_id']) *else* *row*['driver\_id'],

*axis*=1

)

race\_lap\_times\_df.info()

drivers\_df['combined\_key'] = drivers\_df['first\_name'] + '\_' + drivers\_df['abbreviation']

drivers\_dict = drivers\_df.set\_index('combined\_key')['id'].to\_dict()

sprint\_lap\_times\_df['combined\_key'] = sprint\_lap\_times\_df['FirstName'] + '\_' + sprint\_lap\_times\_df['Abbreviation']

sprint\_lap\_times\_df['driver\_id'] = pd.NA

sprint\_lap\_times\_df['driver\_id'] = sprint\_lap\_times\_df.apply(

    lambda *row*: drivers\_dict.get(*row*['combined\_key']) *if* pd.isnull(*row*['driver\_id']) *else* *row*['driver\_id'],

*axis*=1

)

sprint\_lap\_times\_df.info()

race\_lap\_times\_df.head()

def get\_constructors():

*with* engine.connect() *as* connection:

        result = connection.execute(text(f"SELECT \* FROM constructors"))

        data = result.fetchall()

        columns = result.keys()

*return* pd.DataFrame(data, *columns*=columns)

constructors\_df = get\_constructors()

constructors\_df.head()

constructors\_dict = constructors\_df.set\_index('full\_name')['id'].to\_dict()

race\_lap\_times\_df['constructor\_id'] = pd.NA

race\_lap\_times\_df['constructor\_id'] = race\_lap\_times\_df.apply(

    lambda *row*: constructors\_dict.get(*row*['Team']) *if* pd.isnull(*row*['constructor\_id']) *else* *row*['constructor\_id'],

*axis*=1

)

race\_lap\_times\_df.info()

constructors\_dict = constructors\_df.set\_index('name')['id'].to\_dict()

null\_constructor\_id\_df = race\_lap\_times\_df[race\_lap\_times\_df['constructor\_id'].isnull()]

null\_constructor\_id\_df['constructor\_id'] = null\_constructor\_id\_df.apply(

    lambda *row*: constructors\_dict.get(*row*['Team']) *if* pd.isnull(*row*['constructor\_id']) *else* *row*['constructor\_id'],

*axis*=1

)

null\_constructor\_id\_df.info()

race\_lap\_times\_df = pd.concat([race\_lap\_times\_df, null\_constructor\_id\_df])

race\_lap\_times\_df.info()

race\_lap\_times\_df = race\_lap\_times\_df.dropna(*subset*=['constructor\_id'])

race\_lap\_times\_df.info()

constructors\_dict = constructors\_df.set\_index('full\_name')['id'].to\_dict()

sprint\_lap\_times\_df['constructor\_id'] = pd.NA

sprint\_lap\_times\_df['constructor\_id'] = sprint\_lap\_times\_df.apply(

    lambda *row*: constructors\_dict.get(*row*['Team']) *if* pd.isnull(*row*['constructor\_id']) *else* *row*['constructor\_id'],

*axis*=1

)

constructors\_dict = constructors\_df.set\_index('name')['id'].to\_dict()

null\_constructor\_id\_df = sprint\_lap\_times\_df[sprint\_lap\_times\_df['constructor\_id'].isnull()]

null\_constructor\_id\_df['constructor\_id'] = null\_constructor\_id\_df.apply(

    lambda *row*: constructors\_dict.get(*row*['Team']) *if* pd.isnull(*row*['constructor\_id']) *else* *row*['constructor\_id'],

*axis*=1

)

sprint\_lap\_times\_df = pd.concat([sprint\_lap\_times\_df, null\_constructor\_id\_df])

sprint\_lap\_times\_df = sprint\_lap\_times\_df.dropna(*subset*=['constructor\_id'])

sprint\_lap\_times\_df.info()

race\_lap\_times\_df = race\_lap\_times\_df[['race\_id', 'driver\_id', 'constructor\_id', 'LapNumber', 'lap\_time', 'lap\_time\_millis']]

sprint\_lap\_times\_df = sprint\_lap\_times\_df[['race\_id', 'driver\_id', 'constructor\_id', 'LapNumber', 'lap\_time', 'lap\_time\_millis']]

race\_lap\_times\_df.rename(*columns*={'LapNumber': 'lap\_number'}, *inplace*=True)

sprint\_lap\_times\_df.rename(*columns*={'LapNumber': 'lap\_number'}, *inplace*=True)

race\_lap\_times\_df.head()

sprint\_lap\_times\_df.head()

race\_lap\_times\_df['constructor\_id'] = race\_lap\_times\_df['constructor\_id'].astype('Int64')

sprint\_lap\_times\_df['constructor\_id'] = sprint\_lap\_times\_df['constructor\_id'].astype('Int64')

race\_lap\_times\_df.head()

sprint\_lap\_times\_df.head()

def insert\_data(*df*, *table*):

*with* engine.connect() *as* connection:

        result = connection.execute(text(f"SELECT COUNT(\*) FROM {*table*}"))

        count = result.scalar()

*if* count != 0:

            connection.execute(text(f"DELETE FROM {*table*}"))

*df*.to\_sql(*table*, engine, *if\_exists*='append', *index*=False)

insert\_data(race\_lap\_times\_df, 'races\_race\_lap\_times')

insert\_data(sprint\_lap\_times\_df, 'races\_sprint\_race\_lap\_times')

**Робота з моделями**

k-nearest\_neighbours.ipynb

*import* pandas *as* pd

*from* dotenv *import* dotenv\_values

*from* sqlalchemy *import* create\_engine, text

config = dotenv\_values()

DB\_NAME = config.get('DB\_NAME')

DB\_USER = config.get('DB\_USER')

DB\_HOST = config.get('DB\_HOST')

DB\_PASSWORD = config.get('DB\_PASSWORD')

DB\_PORT = config.get('DB\_PORT')

engine = create\_engine(f'postgresql+psycopg2://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}')

def get\_data(*season*):

    query = f"""

        SELECT

            d.id AS driver\_id,

            r.round AS round,

            d.name AS driver\_name,

            fp1.position\_number AS fp1\_result,

            fp2.position\_number AS fp2\_result,

            fp3.position\_number AS fp3\_result,

            q.position\_number AS q\_result,

            rr.grid\_position\_number AS grid\_position,

            rr.position\_number AS finish\_position

        FROM

            drivers d

        JOIN

            races\_fp1\_results fp1 ON d.id = fp1.driver\_id

        JOIN

            races\_fp2\_results fp2 ON d.id = fp2.driver\_id

        JOIN

            races\_fp3\_results fp3 ON d.id = fp3.driver\_id

        JOIN

            races\_qualifying\_results q ON d.id = q.driver\_id

        JOIN

            races\_race\_results rr ON d.id = rr.driver\_id

        JOIN

            races r ON fp1.race\_id = r.id AND fp2.race\_id = r.id AND fp3.race\_id = r.id AND q.race\_id = r.id AND rr.race\_id = r.id

        JOIN

            seasons s ON s.id = r.season\_id

        WHERE

            s.year = :season

    """

*with* engine.connect() *as* connection:

        df = pd.read\_sql(text(query), connection, *params*={'season': *season*})

*return* df

df = get\_data(2019)

df.head()

*import* pandas *as* pd

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* StandardScaler

*from* sklearn.neighbors *import* KNeighborsClassifier

*from* sklearn.metrics *import* accuracy\_score, classification\_report, confusion\_matrix

df = df.dropna()

features = ['fp1\_result', 'fp2\_result', 'fp3\_result', 'q\_result', 'grid\_position']

X = df[features]

y = df['finish\_position']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.2, *random\_state*=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

results = {'best\_n': 0, 'max\_accuracy': 0}

*for* k *in* range(1, 31):

    knn = KNeighborsClassifier(*n\_neighbors*=k)

    knn.fit(X\_train\_scaled, y\_train)

    y\_pred = knn.predict(X\_test\_scaled)

    accuracy = accuracy\_score(y\_test, y\_pred)

*if* results['max\_accuracy'] < accuracy:

        results = {'best\_n': k, 'max\_accuracy': accuracy}

n = results['best\_n']

print('Best n\_neighbours:', n)

knn = KNeighborsClassifier(*n\_neighbors*=n)

knn.fit(X\_train\_scaled, y\_train)

y\_pred = knn.predict(X\_test\_scaled)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

comparison\_df = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

comparison\_df.head(10)

*import* matplotlib.pyplot *as* plt

plt.figure(*figsize*=(10, 6))

plt.scatter(comparison\_df['Actual'], comparison\_df['Predicted'], *color*='blue', *alpha*=0.5)

plt.plot([min(comparison\_df['Actual']), max(comparison\_df['Actual'])],

         [min(comparison\_df['Actual']), max(comparison\_df['Actual'])],

*color*='red', *linestyle*='--')

plt.title('Actual vs Predicted Positions')

plt.xlabel('Actual Position')

plt.ylabel('Predicted Position')

plt.grid(True)

plt.show()

driver\_names = df.loc[X\_test.index]['driver\_name'].tolist()

rounds = df.loc[X\_test.index]['round'].tolist()

visualization\_df = pd.DataFrame({'Driver Name': driver\_names,

                                 'Round': rounds,

                                 'Actual Position': y\_test.values,

                                 'Predicted Position': y\_pred})

round = 1

filtered\_df = visualization\_df[visualization\_df['Round'] == round]

sorted\_df = filtered\_df.sort\_values(*by*='Actual Position')

sorted\_df.head(20)

random\_forest.ipynb

*import* pandas *as* pd

*from* dotenv *import* dotenv\_values

*from* sqlalchemy *import* create\_engine, text

config = dotenv\_values()

DB\_NAME = config.get('DB\_NAME')

DB\_USER = config.get('DB\_USER')

DB\_HOST = config.get('DB\_HOST')

DB\_PASSWORD = config.get('DB\_PASSWORD')

DB\_PORT = config.get('DB\_PORT')

engine = create\_engine(f'postgresql+psycopg2://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}')

def get\_data(*season*):

    query = f"""

        SELECT

            d.id AS driver\_id,

            r.round AS round,

            d.name AS driver\_name,

            fp1.position\_number AS fp1\_result,

            fp2.position\_number AS fp2\_result,

            fp3.position\_number AS fp3\_result,

            q.position\_number AS q\_result,

            rr.grid\_position\_number AS grid\_position,

            rr.position\_number AS finish\_position

        FROM

            drivers d

        JOIN

            races\_fp1\_results fp1 ON d.id = fp1.driver\_id

        JOIN

            races\_fp2\_results fp2 ON d.id = fp2.driver\_id

        JOIN

            races\_fp3\_results fp3 ON d.id = fp3.driver\_id

        JOIN

            races\_qualifying\_results q ON d.id = q.driver\_id

        JOIN

            races\_race\_results rr ON d.id = rr.driver\_id

        JOIN

            races r ON fp1.race\_id = r.id AND fp2.race\_id = r.id AND fp3.race\_id = r.id AND q.race\_id = r.id AND rr.race\_id = r.id

        JOIN

            seasons s ON s.id = r.season\_id

        WHERE

            s.year = :season

    """

*with* engine.connect() *as* connection:

        df = pd.read\_sql(text(query), connection, *params*={'season': *season*})

*return* df

df = get\_data(2019)

df.head()

*import* pandas *as* pd

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* StandardScaler

*from* sklearn.ensemble *import* RandomForestClassifier

*from* sklearn.metrics *import* accuracy\_score, classification\_report, confusion\_matrix

df = df.dropna()

features = ['fp1\_result', 'fp2\_result', 'fp3\_result', 'q\_result', 'grid\_position']

X = df[features]

y = df['finish\_position']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.2, *random\_state*=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

results = {'best\_n': 0, 'max\_accuracy': 0}

*for* k *in* range(1, 101):

    rf = RandomForestClassifier(*n\_estimators*=k, *random\_state*=42)

    rf.fit(X\_train\_scaled, y\_train)

    y\_pred = rf.predict(X\_test\_scaled)

    accuracy = accuracy\_score(y\_test, y\_pred)

*if* results['max\_accuracy'] < accuracy:

        results = {'best\_n': k, 'max\_accuracy': accuracy}

n = results['best\_n']

print('Best n\_estimators:', n)

rf = RandomForestClassifier(*n\_estimators*=n, *random\_state*=42)

rf.fit(X\_train\_scaled, y\_train)

y\_pred = rf.predict(X\_test\_scaled)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

comparison\_df = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

comparison\_df.head(10)

*import* matplotlib.pyplot *as* plt

plt.figure(*figsize*=(10, 6))

plt.scatter(comparison\_df['Actual'], comparison\_df['Predicted'], *color*='blue', *alpha*=0.5)

plt.plot([min(comparison\_df['Actual']), max(comparison\_df['Actual'])],

         [min(comparison\_df['Actual']), max(comparison\_df['Actual'])],

*color*='red', *linestyle*='--')

plt.title('Actual vs Predicted Positions')

plt.xlabel('Actual Position')

plt.ylabel('Predicted Position')

plt.grid(True)

plt.show()

driver\_names = df.loc[X\_test.index]['driver\_name'].tolist()

rounds = df.loc[X\_test.index]['round'].tolist()

visualization\_df = pd.DataFrame({'Driver Name': driver\_names,

                                 'Round': rounds,

                                 'Actual Position': y\_test.values,

                                 'Predicted Position': y\_pred})

round = 1

filtered\_df = visualization\_df[visualization\_df['Round'] == round]

sorted\_df = filtered\_df.sort\_values(*by*='Actual Position')

sorted\_df.head(20)

logistic-regression.ipynb

*import* pandas *as* pd

*from* dotenv *import* dotenv\_values

*from* sqlalchemy *import* create\_engine, text

config = dotenv\_values()

DB\_NAME = config.get('DB\_NAME')

DB\_USER = config.get('DB\_USER')

DB\_HOST = config.get('DB\_HOST')

DB\_PASSWORD = config.get('DB\_PASSWORD')

DB\_PORT = config.get('DB\_PORT')

engine = create\_engine(f'postgresql+psycopg2://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}')

def get\_data(*season*):

    query = f"""

        SELECT

            d.id AS driver\_id,

            r.round AS round,

            d.name AS driver\_name,

            fp1.position\_number AS fp1\_result,

            fp2.position\_number AS fp2\_result,

            fp3.position\_number AS fp3\_result,

            q.position\_number AS q\_result,

            rr.grid\_position\_number AS grid\_position,

            rr.position\_number AS finish\_position

        FROM

            drivers d

        JOIN

            races\_fp1\_results fp1 ON d.id = fp1.driver\_id

        JOIN

            races\_fp2\_results fp2 ON d.id = fp2.driver\_id

        JOIN

            races\_fp3\_results fp3 ON d.id = fp3.driver\_id

        JOIN

            races\_qualifying\_results q ON d.id = q.driver\_id

        JOIN

            races\_race\_results rr ON d.id = rr.driver\_id

        JOIN

            races r ON fp1.race\_id = r.id AND fp2.race\_id = r.id AND fp3.race\_id = r.id AND q.race\_id = r.id AND rr.race\_id = r.id

        JOIN

            seasons s ON s.id = r.season\_id

        WHERE

            s.year = :season

    """

*with* engine.connect() *as* connection:

        df = pd.read\_sql(text(query), connection, *params*={'season': *season*})

*return* df

df = get\_data(2019)

df.head()

*import* pandas *as* pd

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* StandardScaler

*from* sklearn.linear\_model *import* LogisticRegression

*from* sklearn.metrics *import* accuracy\_score, classification\_report, confusion\_matrix

df = df.dropna()

features = ['fp1\_result', 'fp2\_result', 'fp3\_result', 'q\_result', 'grid\_position']

X = df[features]

y = df['finish\_position']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.2, *random\_state*=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

lr = LogisticRegression(*random\_state*=42)

lr.fit(X\_train\_scaled, y\_train)

y\_pred = lr.predict(X\_test\_scaled)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

comparison\_df = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

comparison\_df.head(10)

*import* matplotlib.pyplot *as* plt

plt.figure(*figsize*=(10, 6))

plt.scatter(comparison\_df['Actual'], comparison\_df['Predicted'], *color*='blue', *alpha*=0.5)

plt.plot([min(comparison\_df['Actual']), max(comparison\_df['Actual'])],

         [min(comparison\_df['Actual']), max(comparison\_df['Actual'])],

*color*='red', *linestyle*='--')

plt.title('Actual vs Predicted Positions')

plt.xlabel('Actual Position')

plt.ylabel('Predicted Position')

plt.grid(True)

plt.show()

driver\_names = df.loc[X\_test.index]['driver\_name'].tolist()

rounds = df.loc[X\_test.index]['round'].tolist()

visualization\_df = pd.DataFrame({'Driver Name': driver\_names,

                                 'Round': rounds,

                                 'Actual Position': y\_test.values,

                                 'Predicted Position': y\_pred})

round = 1

filtered\_df = visualization\_df[visualization\_df['Round'] == round]

sorted\_df = filtered\_df.sort\_values(*by*='Actual Position')

sorted\_df.head(20)

support\_vector\_machines.ipynb

*import* pandas *as* pd

*from* dotenv *import* dotenv\_values

*from* sqlalchemy *import* create\_engine, text

config = dotenv\_values()

DB\_NAME = config.get('DB\_NAME')

DB\_USER = config.get('DB\_USER')

DB\_HOST = config.get('DB\_HOST')

DB\_PASSWORD = config.get('DB\_PASSWORD')

DB\_PORT = config.get('DB\_PORT')

engine = create\_engine(f'postgresql+psycopg2://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}')

def get\_data(*season*):

    query = f"""

        SELECT

            d.id AS driver\_id,

            r.round AS round,

            d.name AS driver\_name,

            fp1.position\_number AS fp1\_result,

            fp2.position\_number AS fp2\_result,

            fp3.position\_number AS fp3\_result,

            q.position\_number AS q\_result,

            rr.grid\_position\_number AS grid\_position,

            rr.position\_number AS finish\_position

        FROM

            drivers d

        JOIN

            races\_fp1\_results fp1 ON d.id = fp1.driver\_id

        JOIN

            races\_fp2\_results fp2 ON d.id = fp2.driver\_id

        JOIN

            races\_fp3\_results fp3 ON d.id = fp3.driver\_id

        JOIN

            races\_qualifying\_results q ON d.id = q.driver\_id

        JOIN

            races\_race\_results rr ON d.id = rr.driver\_id

        JOIN

            races r ON fp1.race\_id = r.id AND fp2.race\_id = r.id AND fp3.race\_id = r.id AND q.race\_id = r.id AND rr.race\_id = r.id

        JOIN

            seasons s ON s.id = r.season\_id

        WHERE

            s.year = :season

    """

*with* engine.connect() *as* connection:

        df = pd.read\_sql(text(query), connection, *params*={'season': *season*})

*return* df

df = get\_data(2019)

df.head()

*import* pandas *as* pd

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* StandardScaler

*from* sklearn.svm *import* SVC

*from* sklearn.metrics *import* accuracy\_score, classification\_report, confusion\_matrix

df = df.dropna()

features = ['fp1\_result', 'fp2\_result', 'fp3\_result', 'q\_result', 'grid\_position']

X = df[features]

y = df['finish\_position']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.2, *random\_state*=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

svm = SVC(*kernel*='linear', *random\_state*=42)

svm.fit(X\_train\_scaled, y\_train)

y\_pred = svm.predict(X\_test\_scaled)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

comparison\_df = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

comparison\_df.head(10)

*import* matplotlib.pyplot *as* plt

plt.figure(*figsize*=(10, 6))

plt.scatter(comparison\_df['Actual'], comparison\_df['Predicted'], *color*='blue', *alpha*=0.5)

plt.plot([min(comparison\_df['Actual']), max(comparison\_df['Actual'])],

         [min(comparison\_df['Actual']), max(comparison\_df['Actual'])],

*color*='red', *linestyle*='--')

plt.title('Actual vs Predicted Positions')

plt.xlabel('Actual Position')

plt.ylabel('Predicted Position')

plt.grid(True)

plt.show()

driver\_names = df.loc[X\_test.index]['driver\_name'].tolist()

rounds = df.loc[X\_test.index]['round'].tolist()

visualization\_df = pd.DataFrame({'Driver Name': driver\_names,

                                 'Round': rounds,

                                 'Actual Position': y\_test.values,

                                 'Predicted Position': y\_pred})

round = 1

filtered\_df = visualization\_df[visualization\_df['Round'] == round]

sorted\_df = filtered\_df.sort\_values(*by*='Actual Position')

sorted\_df.head(20)