# Classifying cyclists based on their performances

*IronHack Data Analytics*

*Bulduk Eker*

*December 17th 2023*

# CONTENT OF PRESENTATION

**BUSINESS CASE & OBJECTIVES**

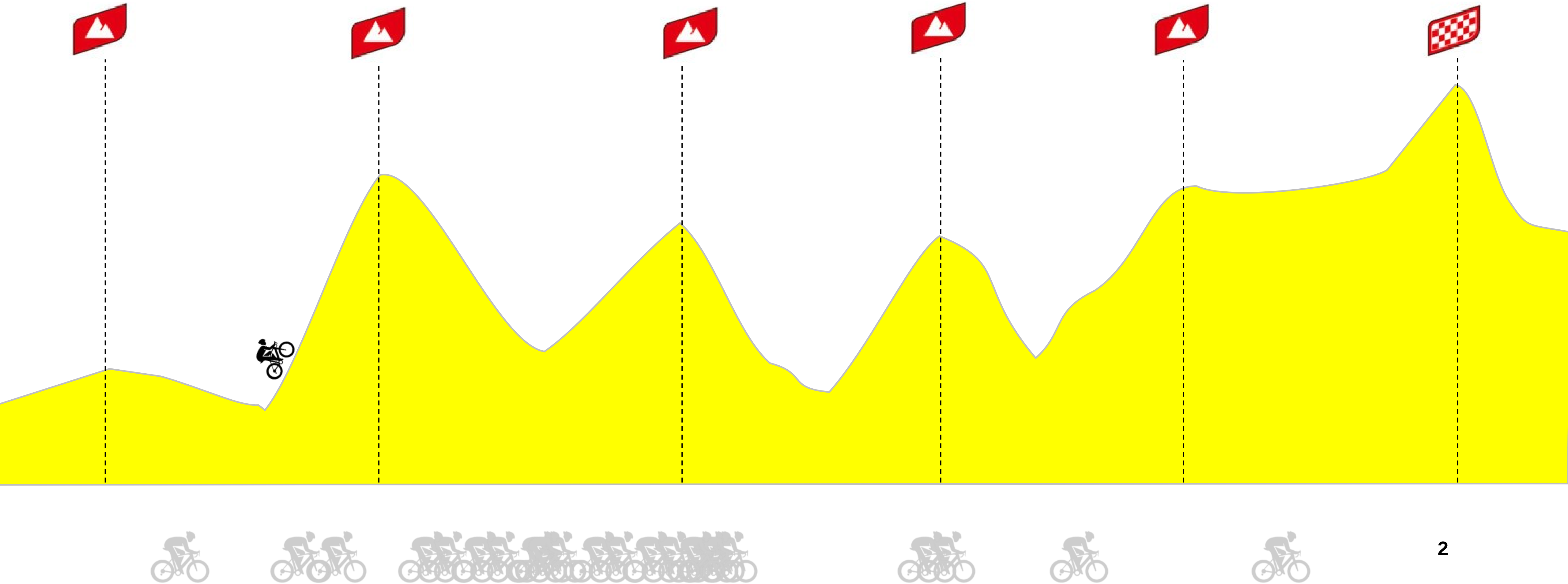
**DATA SOURCES, COLLECTION, CLEANING & PREPARATION**

**EXPLORATORY DATA ANALYSIS / DATA VISUALISATION**

**DATABASE, ERD & QUERIES**

**UNSUPERVISED MACHINE LEARNING MODELS**

**INSIGHT & RECOMMENDATION DEMO**



# CONTEXT & OBJECTIVES

**Personal interest in road cycling competition**

**Growing interest in looking at cycling in strategic perspective, in team manager point of view**

**How can we classify the riders in professional cycling?**

# PLANNING

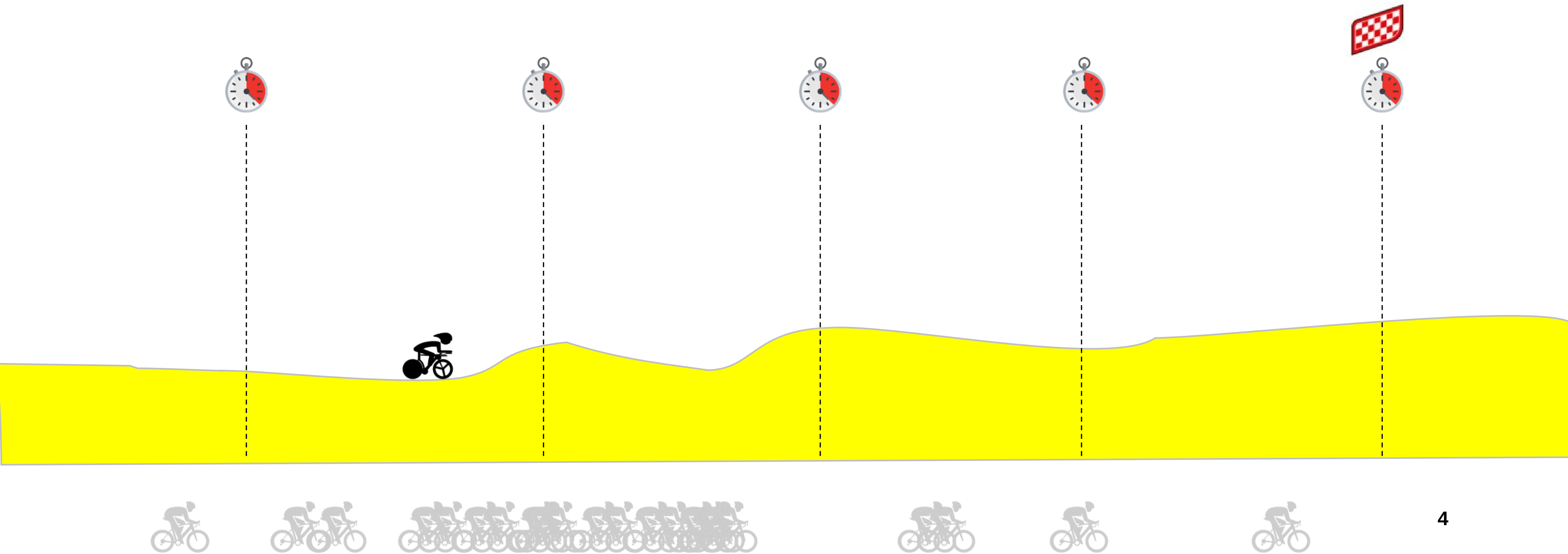
**Checkpoint 1:**
**Web scrapping**

**Checkpoint 2:**
**Data cleaning, preparation & EDA**

**Checkpoint 3:**
**Databases, Queries & ERD**

**Checkpoint 4:**
**Unsupervised ML**

**Checkpoint 5:**
**Building recommendation tools**



# DATA SOURCES & DATA COLLECTION

## Data sources

PRO CYCLING STATS

*<u>Information available:</u>*

- *Race information, race by race, detailed ranking*
- *Rider information, detailed statistics and past performances*
- *Team information, detailed statistics and past performances*
- *Injury records*

## Data collection method

**WEB SCRAPPING**

*<u>Data collected:</u>*

- *Race information: 839 races since 2019, 20 columns*
- *Race results: 117k individual race results since 2019, 13 columns*
- *Team information: 36 teams, 19 columns*
- *Team historical performances: 139 historical performances, 8 columns*
- *Riders information:  936 riders, 27 columns*

# WEB SCRAPPING THE DATA

**Stage 1:**
**Scrapping the URLs of races, teams and riders**

*Stage difficulty:*

**Stage 2:**
**Scrapping the teams informations, riders informations, race informations**

*Stage difficulty:*

**Stage 3:**
**Scrapping the race results**

*Stage difficulty:*

# EXAMPLE OF SCRAPPING: THE RACE RESULTS

## Scrapping the results based on 1 URL

```
def create_dataframe_from_url(url):
    response = requests.get(url)
    if response.status_code != 200:
        return None
    soup = BeautifulSoup(response.content, "html.parser")
    rows = soup.find_all('tr')
    columns = [th.text.strip() for th in rows[0].find_all('th')]
    data = []
    for row in rows:
        values = [td.text.strip() for td in row.find_all('td')]

        # if the row contains a rider (i.e., it's not a header or footer row), add it to the data list
        # following line does not apply to one day race => to put in comment when scrapping for one day races
        # and remove indentation on "data.append(values)"
        if (values and values[2].startswith('+')) or (values and values[0] == 'DNF') or (values and values[0] == 'DNS'):
            data.append(values)

    df = pd.DataFrame(data, columns=columns)
    df['race_code'] = url.replace('https://www.procyclingstats.com/race/','')

    return df
```

## Scrapping the results based on list of URLs

```
def create_dataframe_from_urls(df, url_column):
    num_urls = len(df[url_column])
    dataframes=[]
    for i, url in enumerate(df[url_column]):
        print(f"\rProcessing URL {i+1}/{num_urls}: {url}", end="")
        dataframes.append(create_dataframe_from_url(url))
        # random delay to avoid getting banned
        timer = 0.5 + 0.5 * random.random()
        time.sleep(timer)
    # concatenate all the dataframes in the list into a single dataframe
    result_df = pd.concat(dataframes, ignore_index=True)
    return result_df
```

# DATA CLEANING AND PREPARATION

## Stage 1:
**Basing cleaning**
*(Dropping duplicates, changing to correct formats, concatenating all races results, information)*

*Example: changing date to datetime; speed & distance to float*

```
def clean_date_time_km(df):
    df['date'] = pd.to_datetime(df['date'], format='%d %B %Y')
    df['avg_speed_winner'] = df['avg_speed_winner'].str.replace(' km/h', '').asty
    df['distance'] = df['distance'].str.replace(' km', '').astype(float)
    return df
```

## Stage 2:
**Create new columns for race information** *(including type of race, time trial, identification of final stages)*

*Example: cleaning "won how" columns*

```
conditions = [df_all_race_info['won_how'].str.contains('sprint', case=False, regex=False),
    df_all_race_info['won_how'].str.contains('solo', case=False, regex=False),
    df_all_race_info['won_how'].str.contains('know', case=False, regex=False),
    df_all_race_info['won_how'].str.contains('time trial', case=False, regex=False),
    df_all_race_info['won_how'].str.contains('other', case=False, regex=False)]

values = ['sprint', 'solo', 'other', 'time trial', 'other']

df_all_race_info['type_win'] = np.select(conditions, values, default='')
```

## Stage 3:
**Identification of riders' code in race results and creating new columns with performance summary**

*Example: The function used for matching names with riders' code using fuzzywuzzy library*

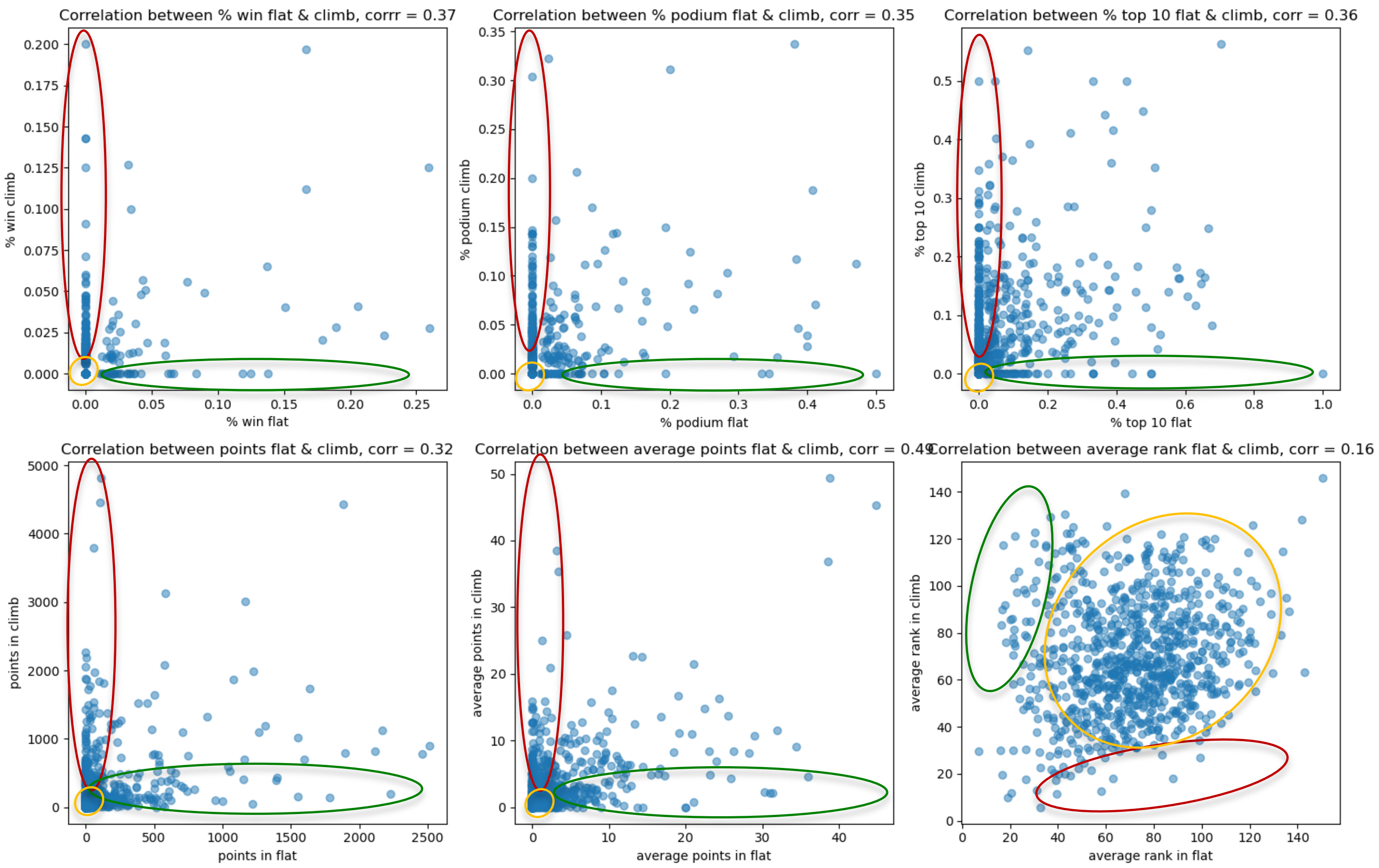
```
def get_rider_names(name, df_rider_check):
    choices = df_rider_check['fullname'].tolist()
    match = process.extractOne(name, choices,
                               scorer=fuzz.token_sort_ratio)
    if match[1] >= 80:
        index = choices.index(match[0])
        return df_rider_check.loc[index, 'rider_code']
    else:
        return None

df_all_results['rider_code'] = None

for i in tqdm(range(len(df_all_results))):
    name = df_all_results.loc[i, 'rider']
    code = get_rider_code(name, df_rider_check)
    df_all_results.loc[i, 'rider_code'] = code
```

# EXPLORATORY DATA ANALYSIS / VISUALISATION

## *Performance comparison between flat and climb races*




- Low correlation between flat stage and climb stage performance and suggests that there are indeed “specialists”
- Presence of lines in the 0 axis of x and y are the specialists
- Red circled cyclists can be considered as pure climber, while Green circled cyclists as pure sprinters
- Orange circled person
- Other points in the middle or top right are more likely to be “all-rounder”

Note: in 2022, 9% of the circuits’ riders have won a race, and 3% have cumulated more than 55% races win

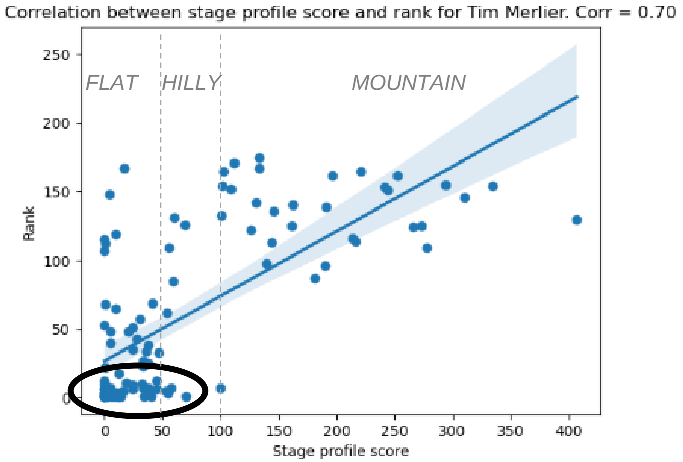
# EXPLORATORY DATA ANALYSIS / VISUALISATION

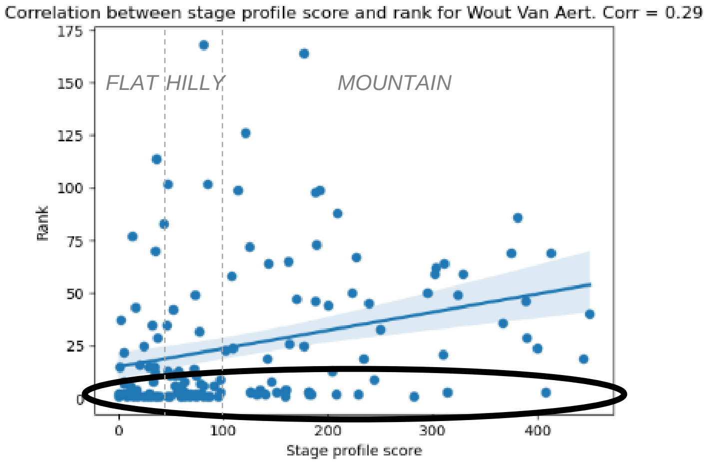
## *Correlation between profile score (stage difficulty) & achieved rank*

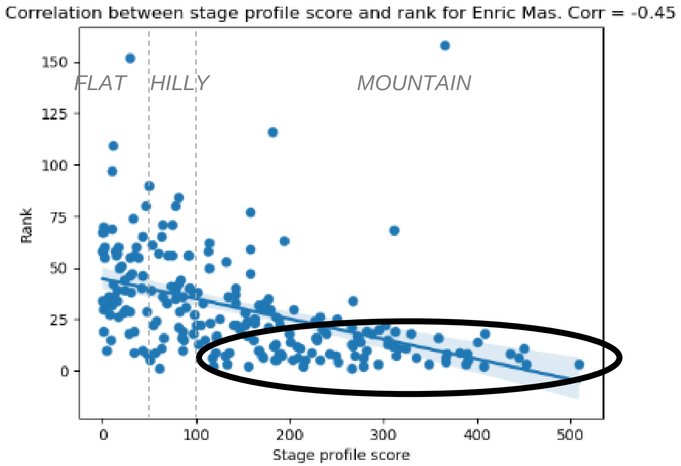
Name: **Tim Merlier**
Rider type: **Sprinter**
Weaknesses: **Climb**

Name: **Wout Van Aert**
Rider type: **All-rounder**
Weaknesses: **Too strong**

Name: **Enric Mas**
Rider type: **Climber**
Weaknesses: **Sprint**



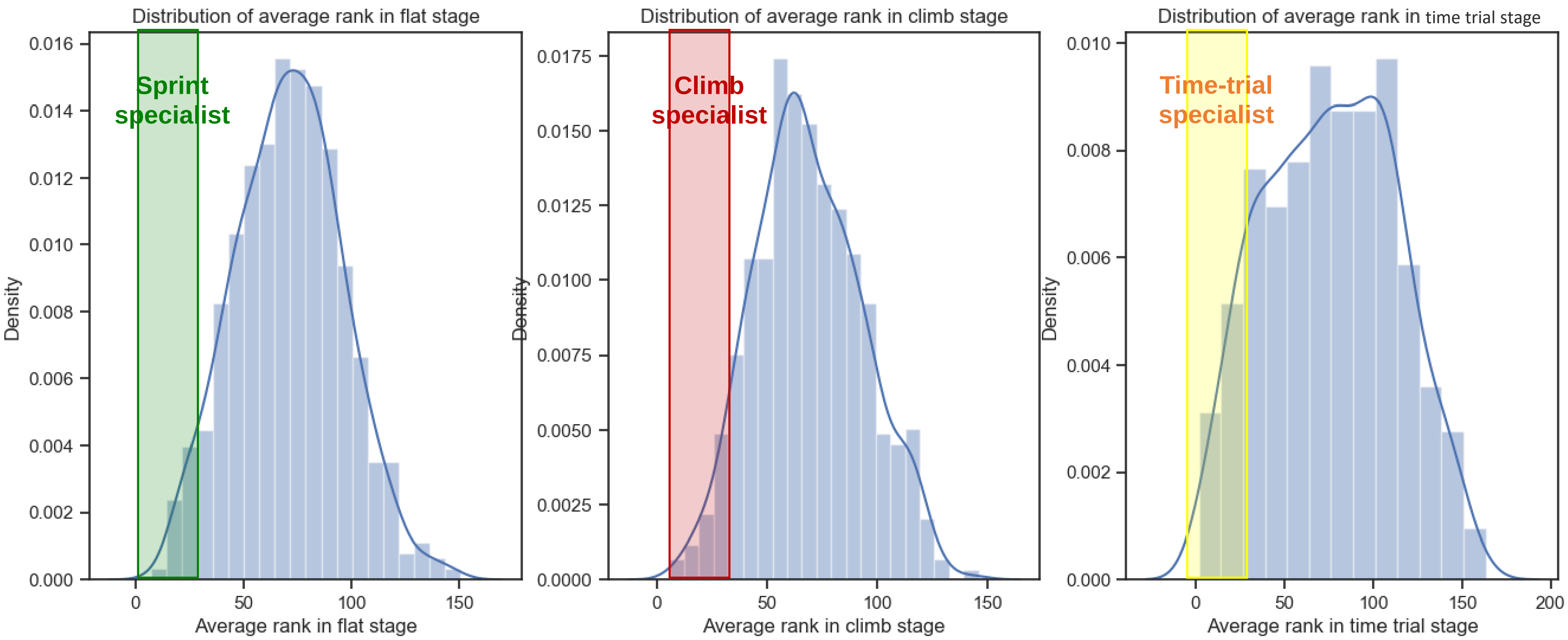

This sprinter gets in first position as long as it is a flat stage. He will drop out when the road rises.

Can get good ranking any types of stages, flat, hilly or mountain.

He will rarely get good positions in flat stage, but when the road rises, he is among the best.

# EXPLORATORY DATA ANALYSIS / VISUALISATION

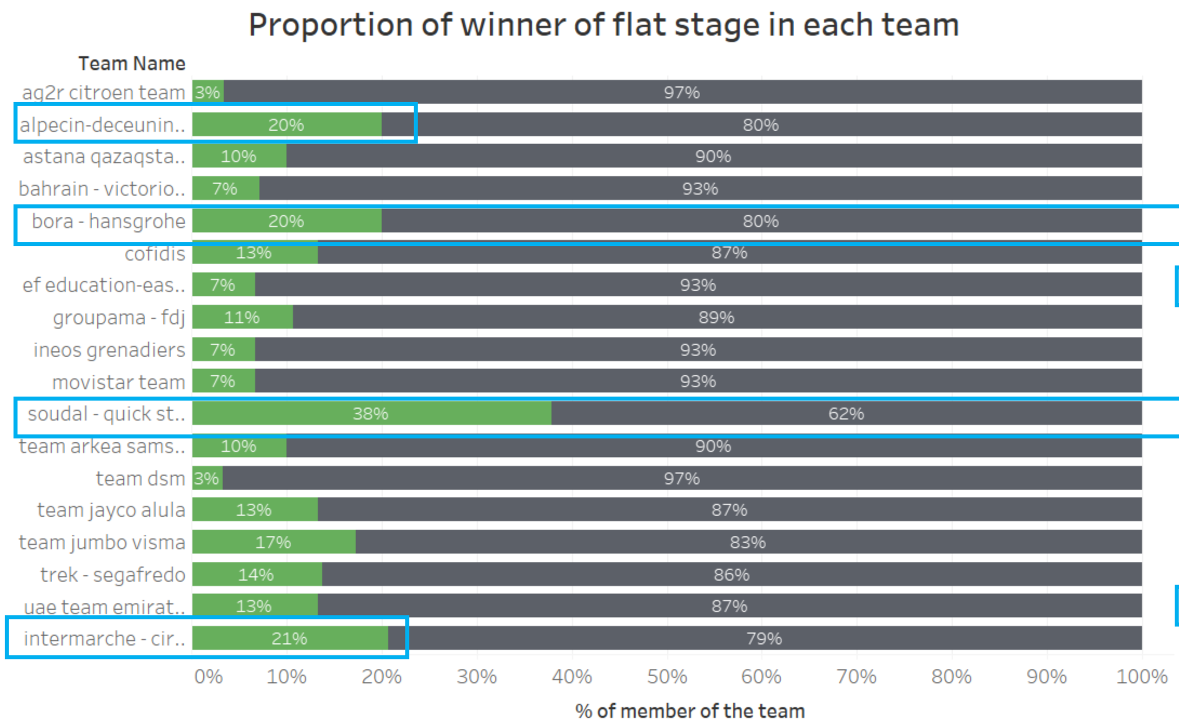
## *Distribution of average rank in flat, climb and time-trial*

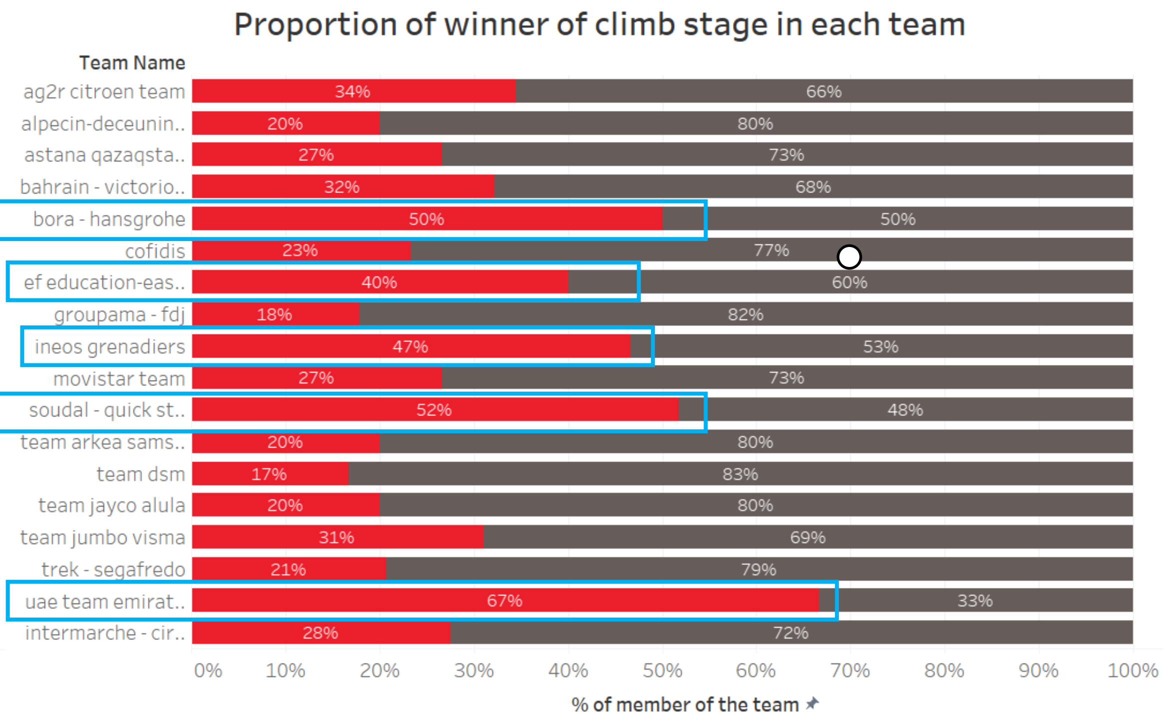



Cycling is a game where there are few winners, and others are just here for their leaders

# EXPLORATORY DATA ANALYSIS / VISUALISATION

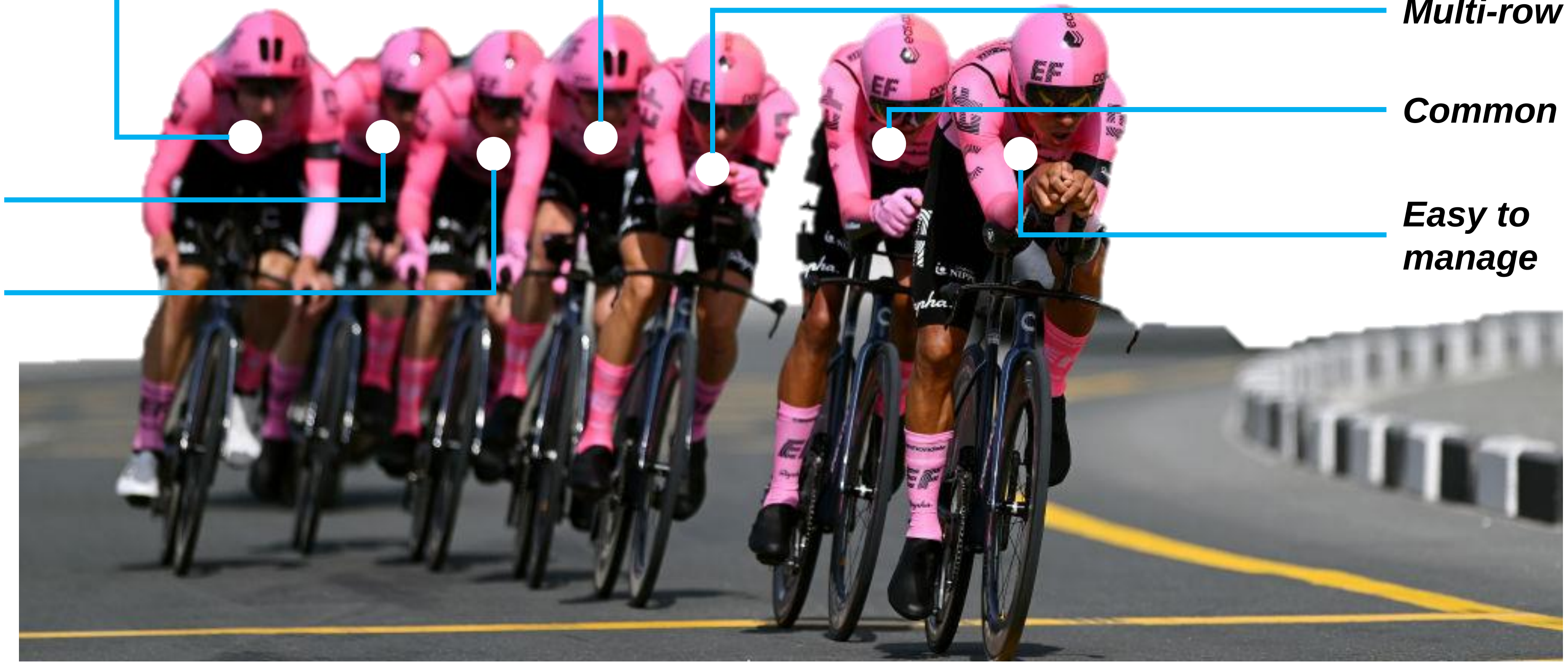
## *Proportion of winning riders in each team*



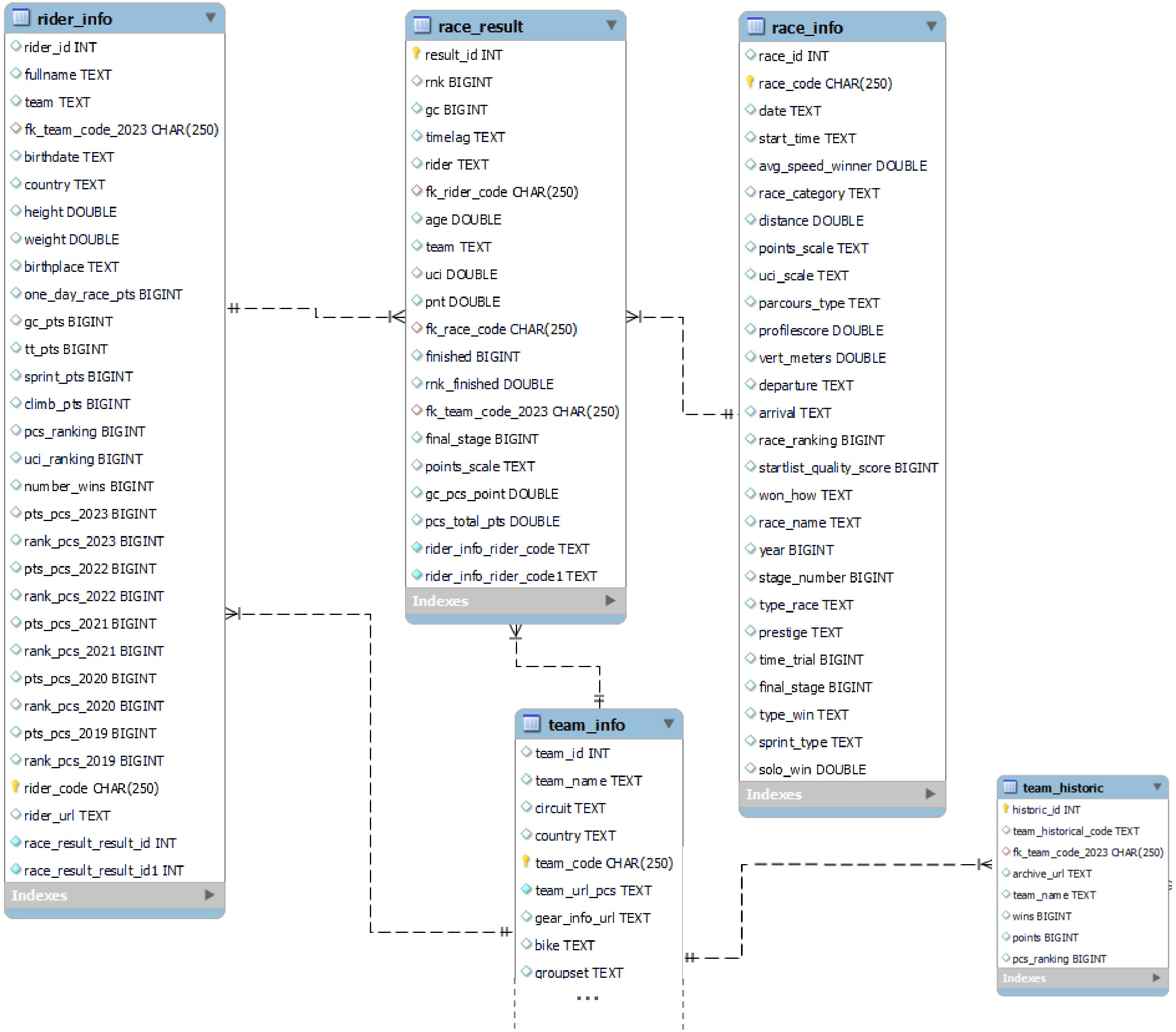




Sprinting seems to be an even more “specialised” exercise. Teams are more likely to get a chance to win in hilly and mountain stages: it is indeed easier to win breaking away from peloton in climbs than in flat stages where teams will work for their leaders to go on sprint.

# DATABASE SELECTION: SQL



# ENTITIY RELATIONSHIP DIAGRAM



# SQL: Creating the table thanks to sqlalchemy

```python
def convert_pd_df_tosql(fname,table_name,schema="cycling"):
    connection_string = 'mysql+pymysql://root:' + "lorem ipsum" + '@127.0.0.1:3306/'
    engine = create_engine(connection_string)
    engine.execute('CREATE SCHEMA IF NOT EXISTS cycling')
    df = pd.read_csv(fname)
    df.to_sql(table_name,engine,schema,index=False)
    return "Created table"

convert_pd_df_tosql('data_cleaning/final_cleaned_tables/race_info_cleaned.csv', "race_info")
convert_pd_df_tosql('data_cleaning/final_cleaned_tables/race_result_cleaned.csv', "race_result")
convert_pd_df_tosql('data_cleaning/final_cleaned_tables/rider_info_cleaned.csv', "rider_info")
convert_pd_df_tosql('data_cleaning/final_cleaned_tables/team_history_performance_cleaned.csv', "team_historic")
convert_pd_df_tosql('data_cleaning/final_cleaned_tables/team_info_and_material_cleaned.csv', "team_info")
```

# SQL Queries: Identifying most successful riders and teams

## RIDERS
*Most wins, excluding GC wins*

```sql
SELECT rider_info.fullname, race_result.fk_rider_code, COUNT(*) AS nb_wins
FROM race_result
JOIN rider_info ON rider_info.rider_code = race_result.fk_rider_code
WHERE race_result.rnk = 1
GROUP BY rider_info.fullname, race_result.fk_rider_code
ORDER BY nb_wins DESC
LIMIT 10;
```

| fullname | fk_rider_code | nb_wins |
|---|---|---|
| primož roglič | primoz-roglic | 29 |
| tadej pogačar | tadej-pogacar | 27 |
| sam bennett | sam-bennett | 23 |
| wout van aert | wout-van-aert | 23 |
| julian alaphilippe | julian-alaphilippe | 18 |
| caleb ewan | caleb-ewan | 18 |
| mathieu van der poel | mathieu-van-der-poel | 17 |
| tim merlier | tim-merlier | 16 |
| jasper philipsen | jasper-philipsen | 15 |

## TEAMS
*Most one-day-race wins*

```sql
SELECT team_info.team_name, race_result.fk_team_code_2023, COUNT(*) AS nb_wins
FROM race_result
JOIN team_info ON team_info.team_code = race_result.fk_team_code_2023
JOIN race_info ON race_info.race_code = race_result.fk_race_code
WHERE (race_result.rnk = 1) AND (race_info.type_race LIKE 'one%')
GROUP BY team_info.team_name, race_result.fk_team_code_2023
ORDER BY nb_wins DESC
LIMIT 10;
```

| team_name | fk_team_code_2023 | nb_wins |
|---|---|---|
| soudal - quick step | soudal-quick-step-2023 | 44 |
| alpecin-deceuninck | alpecin-deceuninck-2023 | 30 |
| uae team emirates | uae-team-emirates-2023 | 24 |
| team jumbo visma | team-jumbo-visma-2023 | 20 |
| lotto dstny | lotto-dstny-2023 | 17 |
| intermarche - circus - wanty | intermarche-circus-wanty-2023 | 16 |
| cofidis | cofidis-2023 | 15 |
| bora - hansgrohe | bora-hansgrohe-2023 | 13 |
| ef education-easypost | ef-education-easypost-2023 | 11 |

*Reading: Dots represents the classification of riders after performing the clustering presented later in the slides:*

*All-rounder* *Sprinter*
*Climber* *Other*

# SQL Queries: Identifying most successful riders and teams

## RIDERS

***Best average rank in climb stage***

```sql
SELECT rider_info.fullname, race_result.fk_rider_code,
avg(race_result.rnk) AS avg_position, count(*) AS nb_race
FROM race_result
JOIN rider_info ON rider_info.rider_code = race_result.fk_rider_code
JOIN race_info ON race_info.race_code = race_result.fk_race_code
WHERE (race_info.parcours_type = 'hill_uphill_finish'
OR race_info.parcours_type = 'mountain_flat_finish'
OR race_info.parcours_type = 'mountain_uphill_finish')
AND (race_result.finished = 1) AND (nb_race > 5)
GROUP BY rider_info.fullname, race_result.fk_rider_code
ORDER BY avg_position ASC
LIMIT 10;
```

| fullname | fk_rider_code | avg_position | nb_race |
|---|---|---|---|
| primož roglič | primoz-roglic | 11.376146788990825 | 109 |
| tadej pogačar | tadej-pogacar | 11.380952380952381 | 84 |
| juan ayuso | juan-ayuso-pesquera | 13.727272727272727 | 22 |
| egan bernal | egan-bernal | 15.316666666666666 | 60 |
| joão almeida | joao-almeida | 15.542857142857143 | 70 |
| adam yates | adam-yates | 16.46987951807229 | 83 |
| aleksandr vlasov | aleksandr-vlasov | 18.191176470588236 | 68 |
| richard carapaz | richard-carapaz | 19.326732673267326 | 101 |
| enric mas | enric-mas | 20.21551724137931 | 116 |
| guillaume martin | guillaume-martin | 21.410714285714285 | 112 |

*Reading: Dots represents the classification of riders after performing the clustering presented later in the slides:*

*All-rounder* *Sprinter*

*Climber* *Other*

# SQL Queries: Identifying most points cumulated

**RIDERS**

***Most PCS points***

```sql
SELECT rider_info.fullname, race_result.fk_rider_code,
SUM(race_result.pcs_total_pts) AS pcs_points
FROM race_result
JOIN rider_info ON rider_info.rider_code = race_result.fk_rider_code
GROUP BY rider_info.fullname, race_result.fk_rider_code
ORDER BY pcs_points DESC
LIMIT 20;
```

| fullname | fk_rider_code | pcs_points |
|---|---|---|
| tadej pogačar | tadej-pogacar | 8650 |
| primož roglič | primoz-roglic | 8446 |
| wout van aert | wout-van-aert | 7825 |
| julian alaphilippe | julian-alaphilippe | 5215 |
| mathieu van der poel | mathieu-van-der-poel | 4350 |
| adam yates | adam-yates | 4046 |
| richard carapaz | richard-carapaz | 4038 |
| michael matthews | michael-matthews | 3884 |
| remco evenepoel | remco-evenepoel | 3754 |
| enric mas | enric-mas | 3669 |
| jasper philipsen | jasper-philipsen | 3506 |
| alexander kristoff | alexander-kristoff | 3476 |
| sam bennett | sam-bennett | 3419 |
| caleb ewan | caleb-ewan | 3397 |
| peter sagan | peter-sagan | 3350 |
| jakob fuglsang | jakob-fuglsang | 3331 |
| egan bernal | egan-bernal | 3292 |
| david gaudu | david-gaudu | 3281 |
| jonas vingegaard | jonas-vingegaard-ra... | 3090 |
| matteo trentin | matteo-trentin | 3066 |

*Reading: Dots represents the classification of riders after performing the clustering presented later in the slides:*

*All-rounder*
*Sprinter*
*Climber*
*Other*

# SQL Queries: Identifying the struggling riders

## RIDERS

*Most DNFs in peloton*

```sql
SELECT
    rider_info.fullname, race_result.fk_rider_code,
    COUNT( CASE WHEN rnk = 'dnf' THEN rider_code END) as dnf_count,
    COUNT(*) as nb_race
FROM race_result
JOIN rider_info ON rider_info.rider_code = race_result.fk_rider_code
GROUP BY rider_info.fullname, race_result.fk_rider_code
ORDER BY dnf_count DESC
LIMIT 10;
```

| fullname | fk_rider_code | dnf_count | nb_race |
|---|---|---|---|
| manuele boaro | manuele-boaro | 34 | 176 |
| tom bohli | tom-bohli | 31 | 174 |
| leonardo basso | leonardo-basso | 31 | 101 |
| tom devriendt | tom-devriendt | 31 | 96 |
| alexandr riabushenko | alexandr-riabushenko | 30 | 145 |
| maciej bodnar | maciej-bodnar | 29 | 183 |
| yevgeniy gidich | yevgeniy-gidich | 29 | 100 |
| ivo oliveira | ivo-emanuel-alves | 27 | 129 |
| guy sagiv | guy-sagiv | 26 | 82 |
| martin laas | martin-laas | 26 | 99 |

*Reading: Dots represents the classification of riders after performing the clustering presented later in the slides:*

*Peloton-guy* *Rookie*

# MACHINE LEARNING

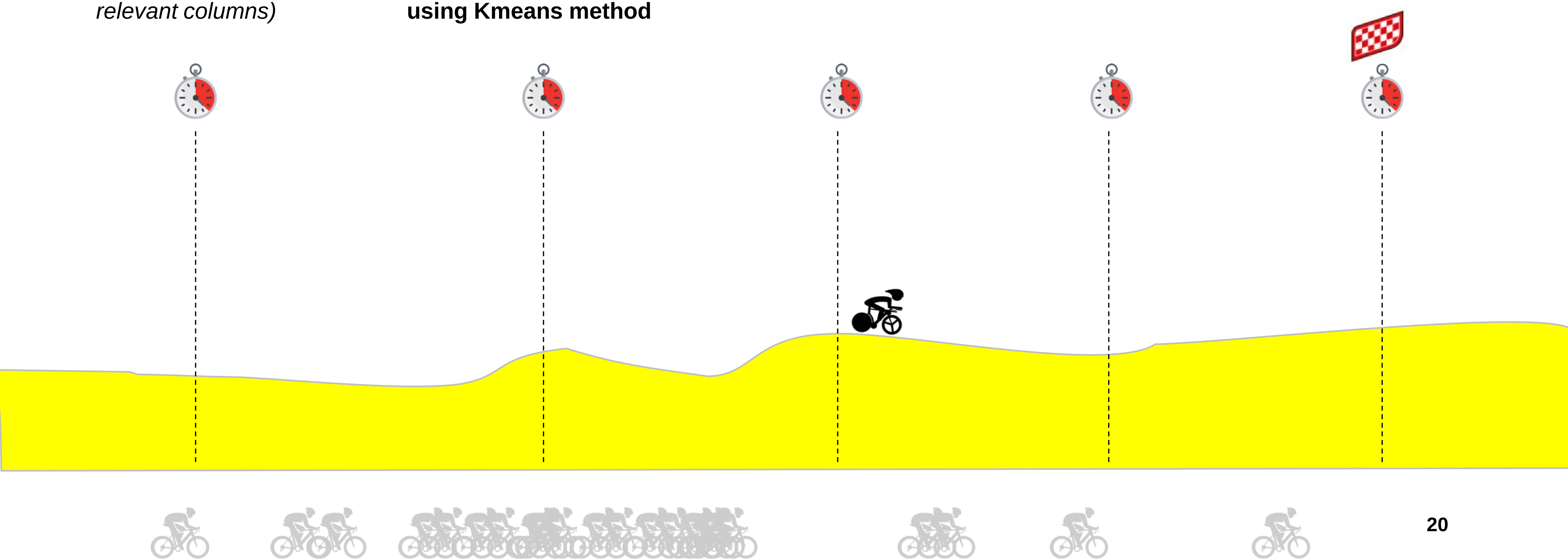
**Checkpoint 1:**
**Data preparation &**
*(handling none value, removing non consistent, relevant columns)*

**Checkpoint 2:**
**Determine the best numbers of clusters & Clustering the riders using Kmeans method**

**Checkpoint 3:**
**Selecting the clustering that have more consistent grouping**

**Checkpoint 4:**
**Producing additional insights**

**Checkpoint 5:**
**Building a recommendation tool**

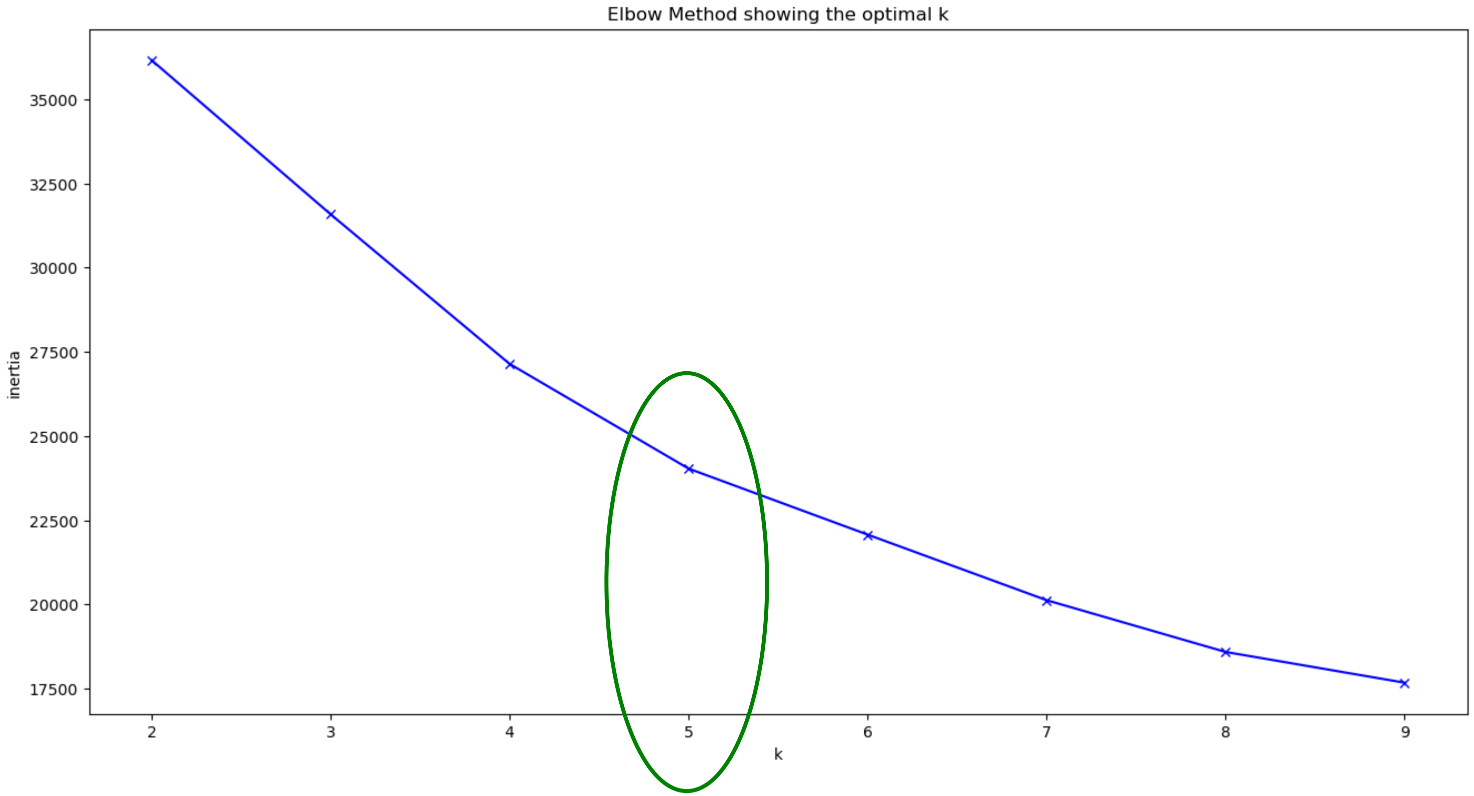


# MACHINE LEARNING

## *Best number of clusters*

```python
inertia = []
for k in range(2, 10):
    print("Training a K-Means model with {} clusters! ".format(k))
    print()
    kmeans = KMeans(n_clusters=k, random_state=1234)
    kmeans.fit(riders_scaled_df)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(16,8))
plt.plot(K, inertia, 'bx-')
plt.xlabel('k')
plt.ylabel('inertia')
plt.xticks(np.arange(min(K), max(K)+1, 1.0))
plt.title('Elbow Method showing the optimal k')
```




Best number of clusters in data analytical perspective is 5

# MACHINE LEARNING

```python
kmeans = KMeans(n_clusters=5, random_state=1234)
kmeans.fit(riders_scaled_df)
```

```python
clusters_kmeans = kmeans.predict(riders_scaled_df)
rider_df["cluster_kmeans_all"] = clusters_kmeans
```

**“Super Sprinters”**

**“Climbers”**

**“Super all-rounders”**

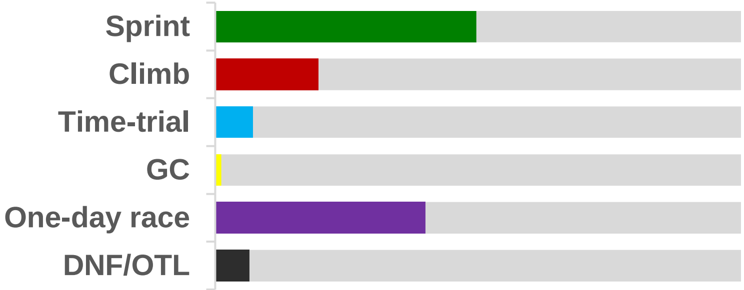
**“Peloton guys”**

**“Rookies”**

# MORE INSIGHTS FROM THE CLUSTERING

## *Sprinter*

**Profile**




*Based on % of Top 10, except DNF*

**Some names**

**Biniam Girmay (ERI)**

**Fabio Jakobsen (NED)**

**Arnaud De Lie (BEL)**

**Nacer Bouhanni (FRA)**

**Average number of win per rider 2019-23**

| | |
|---|---|
| **Flat** | **5.6** |
| **Climb** | **3.4** |
| **Time-trial** | **0.0** |
| **GC** | **0.0** |
| **One-day race** | **3.8** |

*Average races number: 214*

**Characteristics:** Pure sprinters, one-day race specialist, can sprint in large or small groups

**Weaknesses:** Too long climb, can't compete for GC, bad at time-trial

**Ideal environment:** Flat stage, hilly stage with short climb and flat finish, one-day races, in the peloton until last km

# MORE INSIGHTS FROM THE CLUSTERING

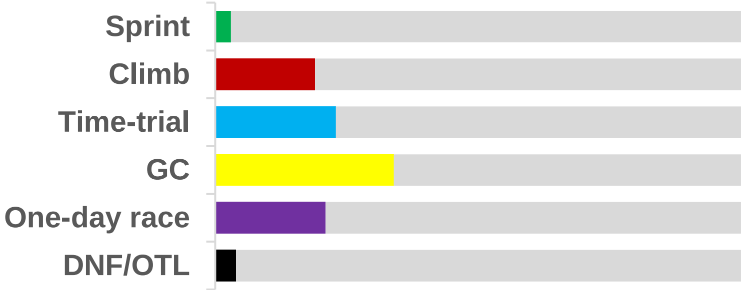
## *Climber*

**Characteristics:** Strong in long climbing, can perform in one-day races, fair level in time-trial

**Weaknesses:** Can't sprint

**Ideal environment:** Stage with uphill finish, in breakaway of mountain stages

### Profile




*Based on % of Top 10, except DNF*

**Average number of win per rider 2019-23**

| | |
|---|---|
| **Flat** | **0.1** |
| **Climb** | **2.2** |
| **Time-trial** | **0.4** |
| **GC** | **0.3** |
| **One-day race** | **0.8** |

*Average races number: 213*

**Some names**

**Richard Carapaz (ECU)**

**Bauke Molema (NED)**

**Valentin Madouas (FRA)**

**Kasper Asgreen (DEN)**

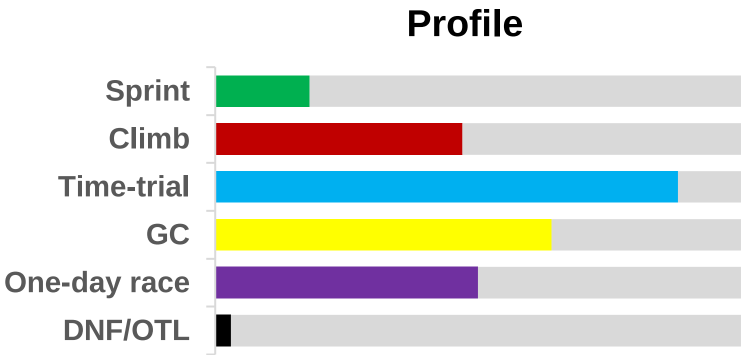
# MORE INSIGHTS FROM THE CLUSTERING

***All-rounder***


**Characteristics:** Can and will win everywhere, except large group sprint

**Weaknesses:** Too strong

**Ideal environment:** Mountain, time-trial, one-day races, stage races




*Based on % of Top 10, except DNF*

**Average number of win per rider 2019-23**

| | |
|---|---|
| **Flat** | **2.0** |
| **Climb** | **15.5** |
| **Time-trial** | **4.8** |
| **GC** | **4.5** |
| **One-day race** | **6.5** |

*Average races number: 203*

**Some names**

**Primož Roglič (SLO)**

**Wout Van Aert (BEL)**

**Remco Evenepoel (BEL)**

**Tadej Pogačar (SLO)**

# MORE INSIGHTS FROM THE CLUSTERING

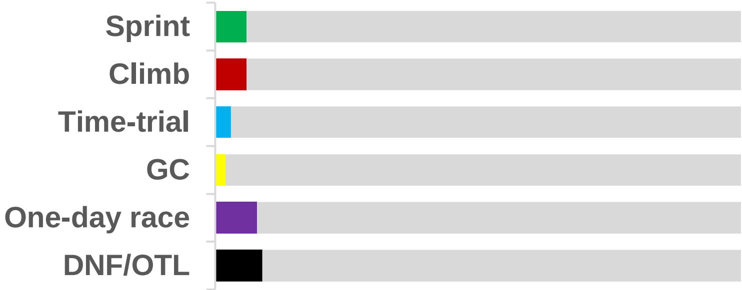
## *Peloton guy*


**Characteristics:** Are usually working for leaders but can get a chance to win once or twice in a year

**Weaknesses:** No clear speciality, can be beaten by any pure specialists or all-rounders

**Ideal environment:** In the peloton or in a breakaway

**Profile**




*Based on % of Top 10, except DNF*

**Average number of win per rider 2019-23**

| | |
|---|---|
| **Flat** | **0.1** |
| **Climb** | **0.4** |
| **Time-trial** | **0.0** |
| **GC** | **0.0** |
| **One-day race** | **0.2** |

*Average races number: 191*

**Some names**

**Lennard Kämna (GER)**

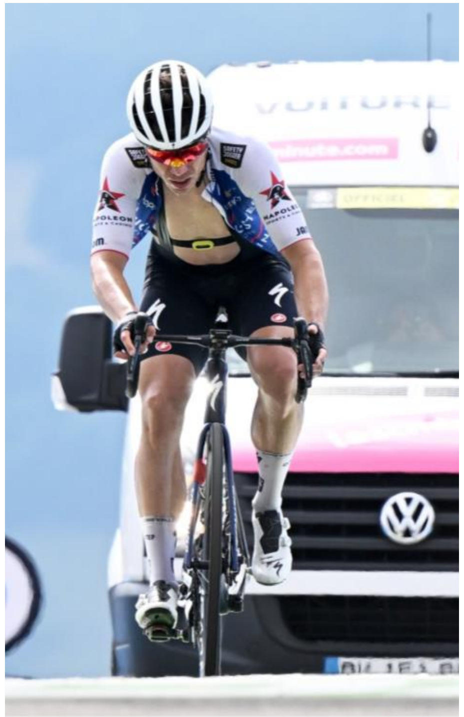
**Bryan Coquard (FRA)**

**Toms Skujiņš (DEN)**

**Quinn Simmons (USA)**

# MORE INSIGHTS FROM THE CLUSTERING

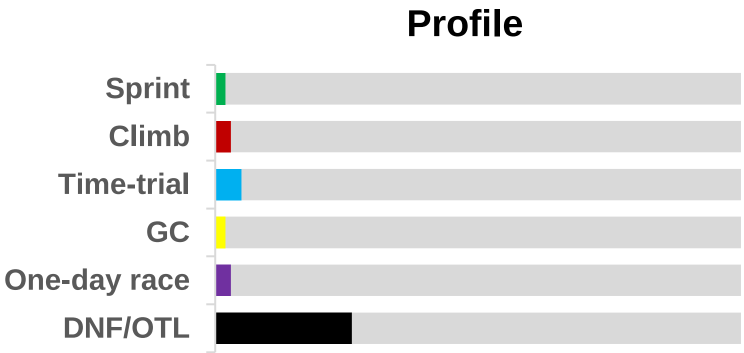
## *Rookie*



**Characteristics:** Young or inexperienced riders, as they do not have a lot of records. Some of them are promising talents

**Weaknesses:** Not sure to finish the race

**Ideal environment:** In a breakaway showing off their sponsor, in the grupetto, in front of the broom wagon




*Based on % of Top 10, except DNF*

**Average number of win per rider 2019-23**

| | |
|---|---|
| **Flat** | **0.0** |
| **Climb** | **0.0** |
| **Time-trial** | **0.0** |
| **GC** | **0.0** |
| **One-day race** | **0.0** |

*Average races number: 56*

**Some names**

**Lenny Martinez (FRA)**

**Marius Mayrhofer (GER)**

**Romain Grégoire (FRA)**

**Magnus Sheffield (USA)**

# RECOMMENDATIONS TOOLS

**Tool 1**

**Who to choose for a race ?**

**Tool 2**

**Who is the best replacement ?**

**Tool 3**

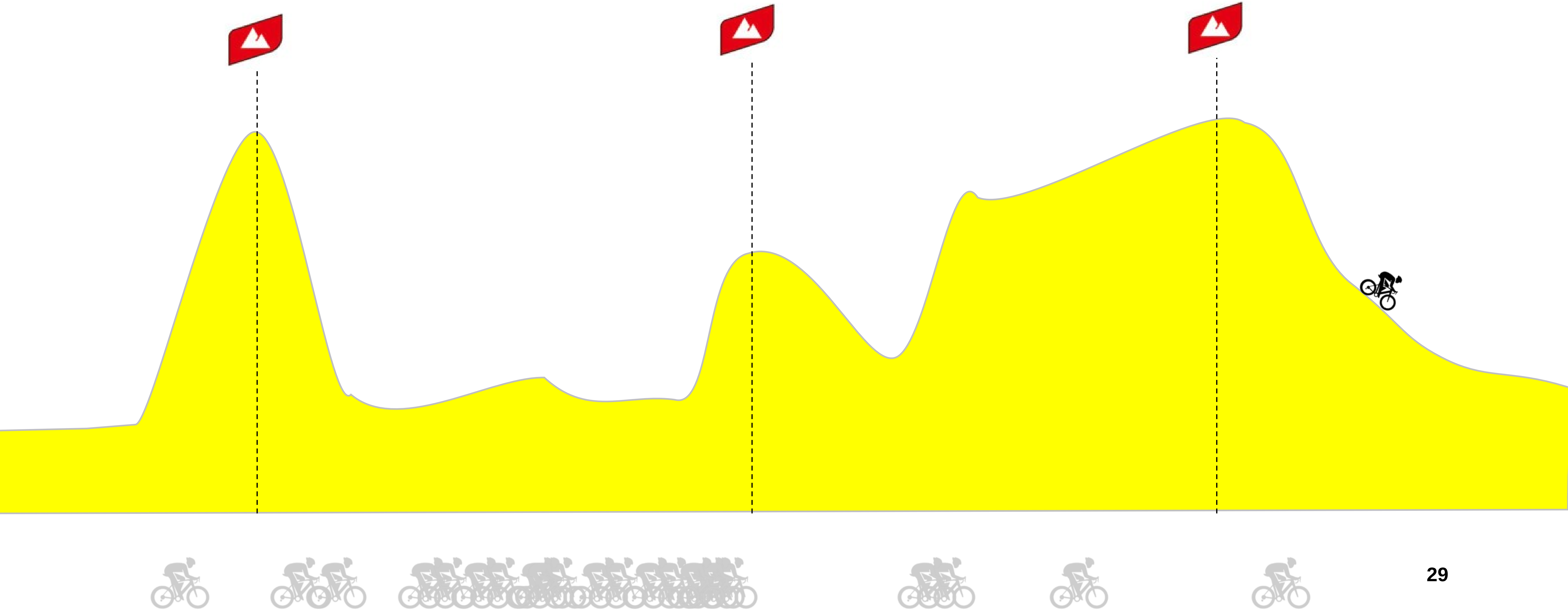
**Who can I recruit ?**

# CHALLENGES FACED

**Web Scrapping:**
Identifying the right part of the code, with the right ‘find’ or ‘findall’ function of scrapping. Many tests until having a function that works well

**Data cleaning:**
Challenging in term of time management, as it is time consuming

**Analysis:**
Choosing the right variable to analyse and/or to remove eventually



# CONCLUSION

- **We successfully identified 5 groups of riders, first through exploratory data analysis, and further investigated with a clustering:**
  - **Super all-rounders**
  - **Pure sprinters**
  - **Climbers / GC specialist**
  - **Peloton guy / Domestique / Once-in-a-year winners**
  - **Rookie / inexperienced**

- **We identified their characteristics, strengths and weaknesses**

- **We built a tool for best recommendations of riders, riders' replacement and recruitment**

# NEXT STEPS

**Digging further in the relation between performance and characteristics of race**

- Type of win (sprint/solo)
- Type of finish (uphill, flat)
- Other characteristics (Flandrian cobbled classics, Ardennes classics)
- Startlist quality
- Breakaway information
- Presence of teammates
- Time series analysis of performance

**Building a tool for recommendation for stage races (that have a mix of flat, mountain, time-trial stages)**

**Additional data sources, when available, such as**

- Strava API training and race informations
- Average power generated (watts) in race in specific segments (final sprint, steep climb, long climb)
- Teams’ collected data

# Thank you !

*IronHack Data Analytics*

*Bulduk Eker*

*bulduk.eker@gmail.com*