

Mini project


Data Science and Visualization, S2026


Fodha Alwan²

Jonas Smedegaard ¹

Krzysztof Dariusz Ignatiuk²

Phillip Andreas von der Heide Skimminge²

¹Roskilde University, Department of People and Technology 

²Roskilde University, Department of Science and Environment 

2026-05-15

Introduction

Prediction problem

How athletic is a given person?

Some people have had their athleticism thoroughly measured, while others have not.

We analyse, through supervised learning, a large set of measured people (called ‘athletes’), across a range of data points like age, height and distance from the measurement facility of the type of athletics (called ‘Olympic Games’), to identify which parameters correlate best with a person’s athletics.

Ideas

We have considered a number of ideas for features.

Idea 1: Athletic performance as it relates to distance from home

Expectation: The further away the host country is from the country of origin the more disadvantaged the athlete will be.

Idea 2: Athletic performance as it relates to altitude

Expectation: Athletes from higher elevations will have an advantage in destinations with higher altitudes.

Idea 3: Athletic performance as it relates to home economy

Expectation: The lower the GDP per capita of the country of origin the lower your chance of success (i.e. an inverse correlation with GDP per capita).

Idea 4: Athletic performance as it relates to height of participant

Expectation: Taller athletes will perform better in disciplines that involve large distances (football, running, long jump, etc.), while shorter athletes will perform better in sports that require precise movement (figure skating, curling, etc.).

Source data

Wikidata from SPARQL endpoint

We use data from [Wikidata](#) about athletes and cities of the Olympic Games in the period 2010–2020.

Although the dataset spans a period of time, it is used as cross-sectional data; the time of events is used only to derive the age of the athlete at the time of the event.

The data is fetched from a SPARQL endpoint: either the official [Wikidata](#) or the faster [QLever](#) endpoints.

At the time of writing, the official endpoint is rate-limited and fails to resolve a second query altogether, making [QLever](#) the preferred option.

```
1 | sparql_endpoint = "https://qllever.dev/api/wikidata"  
2 | sparql_delay = 0
```

```
3 |  
4 | #sparql_endpoint = "https://query.wikidata.org/sparql"  
5 | #sparql_delay = 60
```

The SPARQL endpoint is accessed using the [SPARQLWrapper](#) library, extended locally to transform JSON-serialized results to Pandas dataframes.

Methods

In this section we discuss the methods we used in the project to gather and normalise the data in such a way that it could be analysed using linear and tree-based regression.

Data gathering

The first stage is data gathering, where we will use SPARQL queries to collect data from Wikidata. As the performance of the data source varies wildly depending on the time of day, our plan is to gather the data once and cache it for local use. This allows us to work on the data without pulling from the API every time we want to change something. We have decided to limit the data collection to the Olympic Games between 2010 and 2020 to avoid some irregularities with the countries in the older games and some missing data for the later games. The data from the queries is then inserted into a dataframe for use in later parts of the project.

Data cleaning

After setting up the dataframe with all the data we require, we will move on to the second stage: data cleaning and normalisation. This step is crucial, as the collected data contains some missing values for some attributes that, if left in, could cause our models to fit incorrectly. In preparation for this step we have also noticed a number of problems with the data which we could fix in the query.

Partitioning

The last step in what could be considered preparation is partitioning the data into smaller datasets for training, validation and testing. We do this to check that the derived model works on data that are not part of the training data. The data points are randomly assigned to either the training, validation or testing datasets, which roughly correlate to a 50-25-25 split between the different groupings.

Modelling

Once the data has been prepared, we will start utilising it by mapping some linear regression plots showing the relation between an athletes rank and the specific attributes. For clarity, we would like to inform the reader that we do not expect the answer to be as simple as ‘the wealthier the country, the higher the rank’. If that, or any other single-attribute correlation, were indicative of Olympic performance, there would be no suspense regarding the outcome, and it would thus be a rather lacklustre spectator event. This is why we will also be conducting supervised data analysis in the form of linear and tree-based regression models.

Prepare system and source data

Prepare environment

Load external libraries

Most libraries are included as-is: the common packages pandas, matplotlib, plotnine and scikit-learn, as well as adding geopy, geopandas and shapely to compute distances between geopoints.

```
1 | # Importing all relevant libraries
2 |
3 | # Import the numpy library
4 | import numpy as np
5 |
6 | # Import the pandas library
7 | import pandas as pd
8 |
9 | # Import the Plotnine library
```

```

10 from plotnine import *
11
12 # Import the matplotlib library
13 import matplotlib.pyplot as plt
14
15 # Import the time library
16 import time
17
18 # Import IPython libraries for rendering math
19 from IPython.display import display, Math
20
21 # Import submodules from scikit-learn
22 from sklearn.linear_model import LinearRegression
23 from sklearn.neighbors import KNeighborsRegressor
24 from sklearn.preprocessing import StandardScaler
25 from sklearn.linear_model import LogisticRegression
26 from sklearn.metrics import roc_curve, roc_auc_score
27 from sklearn.preprocessing import PolynomialFeatures, StandardScaler
28 from sklearn.neighbors import KNeighborsClassifier
29
30 from sklearn.tree import DecisionTreeRegressor, plot_tree
31 from sklearn.ensemble import BaggingRegressor
32 from sklearn.ensemble import RandomForestRegressor
33 from sklearn.ensemble import GradientBoostingRegressor
34
35 from sklearn.tree import DecisionTreeClassifier, plot_tree
36 from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
37 from sklearn.ensemble import GradientBoostingClassifier
38 from sklearn.metrics import roc_curve, roc_auc_score
39 from sklearn.model_selection import train_test_split, GridSearchCV
40 from sklearn.inspection import permutation_importance
41
42 # Import submodules for distance calculation
43 import geopy.distance, geopandas as gpd
44 from shapely.geometry import Point

```

Load vendored library

The library SPARQLWrapper, used to access web-based SPARQL API endpoints, has been custom-loaded in order to monkey-patch support for caching. Concretely, to avoid risking being banned from the service when we need to reload many times and, more generally, to act as polite netizens by not burdening public services unnecessarily.

```
1 # SPDX-FileCopyrightText: 2026 Jonas Smedegaard <dr@jones.dk
2 # SPDX-License-Identifier: GPL-3.0-or-later
3 import urllib.request
4 import urllib.error
5 import requests_cache
6
7 cache_session = requests_cache.CachedSession(
8     'sparql2pandas',
9     backend='sqlite',
10    compression='gzip',
11    allowable_methods=('POST'),
12 )
13
14 def cached_urlopen(request, timeout=None):
15     response = cache_session.post(
16         request.get_full_url(),
17         data=request.data,
18         headers=dict(request.header_items()),
19         timeout=timeout or 30,
20     )
21     if response.status_code >= 400:
22         raise urllib.error.URLError(f"HTTP {response.status_code}")
23     return type('CachedResponse', (), {
24         'read': lambda self: response.content,
25         'geturl': lambda self: response.url,
26         'info': lambda self: response.headers,
27         '__enter__': lambda self: self,
28         '__exit__': lambda *args: None,
29     })()
30
31 urllib.request.urlopen = cached_urlopen
32
33 from SPARQLWrapper.SmartWrapper import SPARQLWrapper2, Bindings
```

```

34 import pandas as pd
35 import time
36
37 def _topandas(self) -> pd.DataFrame:
38     """Convert SPARQL Bindings to DataFrame."""
39     return pd.DataFrame(
40         [[binding.get(var).value if binding.get(var) else None
41          for var in self.variables]
42          for binding in self.bindings],
43         columns=self.variables)
44
45 Bindings.topandas = _topandas
46
47 def sparql2pandas(endpoint, rate_limit, query):
48     time.sleep(rate_limit)
49     sparql = SPARQLWrapper2(endpoint)
50     sparql.agent = "SPARQL2Pandas"
51     sparql.setMethod("POST")
52     sparql.setQuery(query)
53     return sparql.query().topandas()

```

Configure Notebook rendering

Configuration of pandas rendering has been tuned to limit table printouts, so as not to clutter this report too much with the many samples provided throughout.

```

1 pd.set_option('display.max_rows', 4)
2 pd.set_option('display.max_columns', 10)
3 pd.set_option('display.max_colwidth', 18)
4 pd.set_option('display.precision', 3)
5 pd.set_eng_float_format(accuracy=3, use_eng_prefix=True)

```

Data gathering

Data from Wikidata is gathered by use of three queries to avoid reaching timeout on the public web service. Two queries are about Olympic data: one focusing on qualities tied to the participating athletes and one on qualities tied to the venue hosting the games. A third query fetches data about countries.

The queries about Olympic data were initially structured from different ends: athlete data started at persons with ties to sports and winning medals, where those medals were granted at an Olympic event; and game data started at the organisation, resolving the events they have organised. It turned out, however, that the person end was far too heavy on the public services available, and the final SPARQL queries presented here have been consolidated to both start by limiting scope to Olympic events and then divert in person or venue directions.

Events

First, data about OL games is collected.

```
1 | events_query = ""
2 | PREFIX wd: <http://www.wikidata.org/entity/>
3 | PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 | PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 | PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
6 | PREFIX p: <http://www.wikidata.org/prop/>
7 | PREFIX psn: <http://www.wikidata.org/prop/statement/value-normalized/>
8 | PREFIX wikibase: <http://wikiba.se/ontology#>
9 |
10 | SELECT DISTINCT ?year ?season ?category
11 |   ?host_country ?host_country_pos
12 | # (GROUP_CONCAT(DISTINCT ?place; SEPARATOR="; ") AS ?places)
13 |   (MAX(?elevation) AS ?host_elevation)
14 | # ?games_id
15 | WHERE {
16 |   BIND("2010" AS ?games_since)
17 |   BIND("2020" AS ?games_before)
18 |   VALUES ?game_variant {
19 |     wd:Q135976384 # Summer Olympic Games
20 |     wd:Q137592217 # Winter Olympic Games
```

```

21     wd:Q3327913      # Summer Paralympic Games
22     wd:Q3317976      # Winter Paralympic Games
23     wd:Q138029040    # Winter Youth Olympic Games
24     wd:Q138028979    # Summer Youth Olympic Games
25 }
26 ?games_id wdt:P31 ?game_variant ;
27     wdt:P580 ?launch .
28 BIND(STR(YEAR(?launch)) AS ?year)
29 FILTER(?year > ?games_since && ?year <= ?games_before)
30
31 BIND(IF(?game_variant IN (wd:Q137592217, wd:Q3317976, wd:Q138029040),
32     "winter", "summer") AS ?season)
33 BIND(IF(?game_variant IN (wd:Q3327913, wd:Q3317976), "para",
34     IF(?game_variant IN (wd:Q138029040, wd:Q138028979), "youth",
35     "main")) AS ?category)
36
37 OPTIONAL {
38     ?games_id wdt:P17 ?host_country_id .
39     ?host_country_id rdfs:label ?host_country .
40     FILTER(LANG(?host_country) = "en")
41     OPTIONAL {
42         ?host_country_id wdt:P625 ?host_country_pos .
43     }
44 }
45
46 OPTIONAL {
47     VALUES ?place_property { wdt:P276 wdt:P131 } # geo or admin place
48     ?games_id ?place_property ?place_id .
49     OPTIONAL {
50         ?place_id p:P2044/psn:P2044 [
51             wikibase:quantityAmount ?elevation_amount ;
52             wikibase:quantityUnit ?elevation_unit ] .
53         BIND(
54             IF(?elevation_unit = wd:Q11573, ?elevation_amount,          # m
55             IF(?elevation_unit = wd:Q3710, ?elevation_amount * 0.3048, # foot
56             IF(?elevation_unit = wd:Q828224, ?elevation_amount * 1000, # km
57             ?elevation_amount))) AS ?elevation
58         )
59     }
60 }

```

```

61 }
62 GROUP BY ?year ?season ?category ?host_country ?host_country_pos
63         ?games_id
64 ORDER BY ?year ?season ?category
65
66 """
67
68 events = sparql2pandas(sparql_endpoint, sparql_delay, events_query)
69 print(events)

```

	year	season	category	host_country	host_country_pos	host_elevation
0	2012	summer	main	United Kingdom	POINT(-2.00000...	15.0
1	2012	summer	para	United Kingdom	POINT(-2.00000...	15.0
..
11	2018	winter	para	South Korea	POINT(128.0000...	600.0
12	2020	winter	youth	Switzerland	POINT(8.231973...	495.0

[13 rows x 6 columns]

Participants

Then data about participants at OL games is collected.

```

1 athletes_query = """
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
6 PREFIX p: <http://www.wikidata.org/prop/>
7 PREFIX ps: <http://www.wikidata.org/prop/statement/>
8 PREFIX psn: <http://www.wikidata.org/prop/statement/value-normalized/>
9 PREFIX pq: <http://www.wikidata.org/prop/qualifier/>
10 PREFIX wikibase: <http://wikiba.se/ontology#>
11
12 SELECT ?person ?rank ?age ?sex_or_gender ?height ?mass
13        ?country ?year ?season ?category ?sport
14 # ?person_id ?event_id
15 WHERE {
16     BIND("2010" AS ?games_since)
17     BIND("2020" AS ?games_before)

```

```

18 VALUES ?game_variant {
19     wd:Q135976384 # Summer Olympic Games
20     wd:Q137592217 # Winter Olympic Games
21     wd:Q3327913   # Summer Paralympic Games
22     wd:Q3317976   # Winter Paralympic Games
23     wd:Q138029040 # Winter Youth Olympic Games
24     wd:Q138028979 # Summer Youth Olympic Games
25 }
26 ?games_id wdt:P31 ?game_variant ;
27     wdt:P580 ?launch .
28 BIND(STR(YEAR(?launch)) AS ?year)
29 FILTER(?year > ?games_since && ?year <= ?games_before)
30
31 BIND(IF(?game_variant IN (wd:Q137592217, wd:Q3317976, wd:Q138029040),
32     "winter", "summer") AS ?season)
33 BIND(IF(?game_variant IN (wd:Q3327913, wd:Q3317976), "para",
34     IF(?game_variant IN (wd:Q138029040, wd:Q138028979), "youth",
35     "main")) AS ?category)
36
37 ?event_id wdt:P361+ ?games_id .
38
39 # Event must be a sports competition
40 ?event_id wdt:P641/rdfs:label ?sport .
41 FILTER(LANG(?sport) = "en")
42
43 ?event_id wdt:P31 ?event_type .
44 ?event_type rdfs:label ?event_type_raw .
45 FILTER(LANG(?event_type_raw) = "en")
46 FILTER(
47     CONTAINS(LCASE(?event_type_raw), "olympic") ||
48     CONTAINS(LCASE(?event_type_raw), "paralympic")
49 )
50 FILTER(CONTAINS(LCASE(?event_type_raw), "event") ||
51     CONTAINS(LCASE(?event_type_raw), "competition"))
52
53 # Find persons with country, and with either rank or medal in events
54 ?person_id wdt:P31 wd:Q5 ;
55     wdt:P27 ?country_id ;
56     p:P1344 ?participation .
57 ?participation ps:P1344 ?event_id .

```

```

58
59 # Must have either rank or medal
60 OPTIONAL {
61     ?participation pq:P1352 ?rank_raw .
62 }
63 OPTIONAL {
64     ?participation pq:P166 ?medal_id .
65     VALUES ?medal_id { wd:Q15243387 wd:Q15243388 wd:Q15243389 }
66 }
67 FILTER(BOUND(?rank_raw) || BOUND(?medal_id))
68
69 BIND(
70     IF(BOUND(?rank_raw), ?rank_raw,
71         IF(?medal_id = wd:Q15243387, 1,
72         IF(?medal_id = wd:Q15243388, 2,
73         IF(?medal_id = wd:Q15243389, 3,
74         ?rank_raw)))) AS ?rank
75 )
76
77 OPTIONAL {
78     ?person_id wdt:P569 ?birth_raw .
79 }
80 BIND(
81     IF(BOUND(?birth_raw),
82         YEAR(?launch) - YEAR(?birth_raw) -
83         IF(MONTH(?launch) < MONTH(?birth_raw) ||
84             (MONTH(?launch) = MONTH(?birth_raw) &&
85             DAY(?launch) < DAY(?birth_raw)),
86         1, 0),
87         ?birth_raw) AS ?age
88 )
89
90 OPTIONAL {
91     ?person_id wdt:P21 ?sex_or_gender_id .
92     ?sex_or_gender_id rdfs:label ?sex_or_gender .
93     FILTER(LANG(?sex_or_gender) = "en")
94 }
95
96 OPTIONAL {
97     ?person_id p:P2048/psn:P2048 [

```

```

98     wikibase:quantityAmount ?height_amount ;
99     wikibase:quantityUnit ?height_unit ] .
100 BIND(
101     IF(?height_unit = wd:Q11573, ?height_amount,          # meter
102     IF(?height_unit = wd:Q174728, ?height_amount * 0.01, # centimeter
103     IF(?height_unit = wd:Q218593, ?height_amount * 0.0254, # inch
104     IF(?height_unit = wd:Q3710, ?height_amount * 0.3048, # foot
105     ?height_amount * -1)))) AS ?height          # ambiguous
106 )
107 }
108
109 OPTIONAL {
110     ?person_id p:P2067/psn:P2067 [
111         wikibase:quantityAmount ?mass_amount ;
112         wikibase:quantityUnit ?mass_unit ] .
113     BIND(
114         IF(?mass_unit = wd:Q11570, ?mass_amount,          # kilogram
115         IF(?mass_unit = wd:Q41803, ?mass_amount * 0.001, # gram
116         IF(?mass_unit = wd:Q100995, ?mass_amount * 0.45359237, # pound
117         ?mass_amount * -1)))) AS ?mass          # ambiguous
118     )
119 }
120
121 # find newest citizenship
122 {
123     SELECT ?person_id (SAMPLE(?c_id) AS ?country_id)
124     WHERE {
125         {
126             SELECT ?person_id ?c_id (MAX(?c_start) AS ?max_start)
127             WHERE {
128                 ?person_id p:P27 ?citizenship .
129                 ?citizenship ps:P27 ?c_id .
130                 OPTIONAL { ?citizenship pq:P580 ?c_start . }
131             }
132             GROUP BY ?person_id ?c_id
133         }
134         BIND(COALESCE(?max_start, "0001-01-01"^^xsd:dateTime)
135             AS ?sort_date)
136     }
137     GROUP BY ?person_id

```

```

138     ORDER BY DESC(?sort_date)
139 }
140
141 ?country_id rdfs:label ?country .
142 FILTER(LANG(?country) = "en")
143
144 ?person_id rdfs:label ?person .
145 FILTER (LANG(?person) = "en")
146
147 # TODO: resolve citizenship at the time of event (not simply newest)
148 # FIXME: avoid double-counting athletes with multiple participations
149 }
150 ORDER BY ?year ?season ?category ?sport ?person
151
152 """
153
154 athletes = sparql2pandas(sparql_endpoint, sparql_delay, athletes_query)
155 print(athletes)

```

	person	rank	age	sex_or_gender	height	...	country	year	\
0	Aída Román	7.0	24	female	1.68	...	Mexico	2012	
1	Aída Román	2.0	24	female	1.68	...	Mexico	2012	
...	
11165	Valeriya Sorok...	14.0	16	female	NaN	...	Russia	2020	
11166	Yukino Yoshida	3.0	16	female	NaN	...	Japan	2020	

	season	category	sport
0	summer	main	archery
1	summer	main	archery
...
11165	winter	youth	speed skating
11166	winter	youth	speed skating

[11167 rows x 11 columns]

Countries

Finally, data about countries is collected.

```

1 country_query = ""
2 prefix wd: <http://www.wikidata.org/entity/>
3 prefix wdt: <http://www.wikidata.org/prop/direct/>
4 prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 prefix p: <http://www.wikidata.org/prop/>
6 prefix ps: <http://www.wikidata.org/prop/statement/>
7 prefix pq: <http://www.wikidata.org/prop/qualifier/>
8
9 select ?country ?gdp ?country_pos ?capital ?capital_elevation
10 # ?country_id
11 where {
12     ?country_id wdt:P31 wd:Q6256 . # sovereign state
13
14     ?country_id p:P2131 [
15         ps:P2131 ?gdp ;
16         pq:P585 ?gdp_time ;
17     ] ;
18     rdfs:label ?country .
19     filter(lang(?country) = "en")
20
21     optional {
22         ?country_id wdt:P625 ?country_pos .
23     }
24     optional {
25         ?country_id wdt:P36 ?capital_id .
26         ?capital_id rdfs:label ?capital .
27         filter(lang(?capital) = "en")
28     }
29     optional {
30         ?capital_id wdt:P2044 ?capital_elevation .
31     }
32
33     # subquery to find newest GDP
34     {
35         select ?country_id (max(?year) as ?max_year)
36         where {
37             ?country_id wdt:P31 wd:Q6256 ;
38             p:P2131 [
39                 ps:P2131 ?val ;
40                 pq:P585 ?pt ;

```

```

41     ] .
42     bind(year(?pt) as ?year)
43   }
44   group by ?country_id
45 }
46 filter(year(?gdp_time) = ?max_year)
47 }
48 order by ?country
49
50 """
51
52 country = sparql2pandas(sparql_endpoint, sparql_delay, country_query)
53 print(country)

```

	country	gdp	country_pos	capital	capital_elevation
0	Afghanistan	17233051620.0	POINT(66.00000...	Kabul	1790.0
1	Albania	23547179830.0	POINT(20.00000...	Tirana	110.0
..
203	Zimbabwe	20678055598.0	POINT(30.00000...	Harare	1494.0
204	Zimbabwe	20678055598.0	POINT(30.00000...	Harare	1492.0

[205 rows x 5 columns]

Data cleaning

```

1 df_ = pd.merge(athletes, events, on = ["year", "season", "category"])
2 df = pd.merge(df_, country, on = "country")
3
4 print(df)

```

	person	rank	age	sex_or_gender	height	...	host_elevation	\
0	Aída Román	7.0	24	female	1.68	...	15.0	
1	Aída Román	2.0	24	female	1.68	...	15.0	
...	
11240	Valeriya Sorok...	14.0	16	female	NaN	...	495.0	
11241	Yukino Yoshida	3.0	16	female	NaN	...	495.0	

	gdp	country_pos	capital	capital_elevation
0	1414187193992.0	POINT(-102.000...	Mexico City	2240.0

```

1      1414187193992.0 POINT(-102.000... Mexico City      2240.0
...
11240 2240422438363.0 POINT(94.25000... Moscow      156.0
11241 4231141201863.0 POINT(136.0000... Tokyo      6.0

```

[11242 rows x 18 columns]

Numeric values

Numeric fields are typecast as such.

```

1 df["rank"] = pd.to_numeric(df["rank"]).astype(int)
2
3 df["capital_elevation"] = pd.to_numeric(df["capital_elevation"])
4 df["host_elevation"] = pd.to_numeric(df["host_elevation"])
5
6 print(df)

```

```

           person  rank  age  sex_or_gender  height  ...  host_elevation  \
0      Aída Román    7   24      female    1.68  ...      15.000
1      Aída Román    2   24      female    1.68  ...      15.000
...
11240  Valeriya Sorok...  14  16      female    NaN  ...      495.000
11241   Yukino Yoshida    3  16      female    NaN  ...      495.000

```

```

           gdp      country_pos      capital  capital_elevation
0      1414187193992.0 POINT(-102.000... Mexico City      2.240k
1      1414187193992.0 POINT(-102.000... Mexico City      2.240k
...
11240 2240422438363.0 POINT(94.25000... Moscow      156.000
11241 4231141201863.0 POINT(136.0000... Tokyo      6.000

```

[11242 rows x 18 columns]

Rank and on podium

Rank is simplified as on podium or not – i.e. a top-three ranking.

```

1 df.loc[(df.loc[:, "rank"] == 1), "podium"] = "On podium"
2 df.loc[(df.loc[:, "rank"] == 2), "podium"] = "On podium"
3 df.loc[(df.loc[:, "rank"] == 3), "podium"] = "On podium"

```

```

4 df.loc[df.loc[:, "podium"].isna(), "podium"] = "Not on podium"
5
6 print(df)

```

	person	rank	age	sex_or_gender	height	...	gdp	\
0	Aída Román	7	24	female	1.68	...	1414187193992.0	
1	Aída Román	2	24	female	1.68	...	1414187193992.0	
...	
11240	Valeriya Sorok...	14	16	female	NaN	...	2240422438363.0	
11241	Yukino Yoshida	3	16	female	NaN	...	4231141201863.0	

	country_pos	capital	capital_elevation	podium
0	POINT(-102.000...	Mexico City	2.240k	Not on podium
1	POINT(-102.000...	Mexico City	2.240k	On podium
...
11240	POINT(94.25000...	Moscow	156.000	Not on podium
11241	POINT(136.0000...	Tokyo	6.000	On podium

[11242 rows x 19 columns]

Checking for and deleting NA values

```

1 # Specify a global theme to use in all plots
2 title_left = element_text(face = "bold", size = 10, ha = "left")
3 title_center = element_text(face = "bold", size = 10, ha = "center")
4 global_theme = (
5     theme_minimal() +
6     theme(
7         title = title_left,
8         axis_text = element_text(size = 10),
9         axis_title = title_center,
10        legend_position = "top",
11        legend_direction = "horizontal",
12        legend_justification = "left",
13        legend_text = element_text(size = 10),
14        legend_title = title_center,
15        legend_title_position = "bottom",
16        figure_size = (8, 6)
17    )

```

```

18 | )

1 | # Construct a DataFrame containing the data to be plotted
2 | # Compute percentages of missing and non-missing values on each column
3 | # Collect the values in a new DataFrame
4 | df_nan_plot = pd.DataFrame(
5 |     {
6 |         "col": df.columns,
7 |         "Missing": round(df.isna().sum() / df.shape[0] * 100, 1),
8 |         "Not missing": round(df.notna().sum() / df.shape[0] * 100, 1)
9 |     }
10 | )
11 | # Reshape the data to long format
12 | df_nan_plot = df_nan_plot.melt(
13 |     id_vars = "col",
14 |     value_vars = ["Missing", "Not missing"],
15 |     var_name = "nan_obs",
16 |     value_name = "pct"
17 | )

1 | # Reorder categorical variables and replace 0 with none
2 | # Reorder column names after percentage of nan values
3 | df_nan_plot["col"] = pd.Categorical(
4 |     df_nan_plot.loc[:, "col"],
5 |     categories = (df_nan_plot
6 |         .loc[df_nan_plot.loc[:, "nan_obs"] == "Missing", :]
7 |         .sort_values("pct")
8 |         .loc[:, "col"]
9 |         .tolist()
10 |     ),
11 |     ordered = True
12 | )
13 |
14 | # Reorder grouping column to make nan appear first in plot
15 | df_nan_plot["nan_obs"] = pd.Categorical(
16 |     df_nan_plot.loc[:, "nan_obs"],
17 |     categories = ["Not missing", "Missing"],
18 |     ordered = True
19 | )
20 |

```

```

21 | # Remove rows with percentage value 0
22 | df_nan_plot = df_nan_plot.loc[df_nan_plot.loc[:, "pct"] > 0, :]

1 | # Step 3: Create the plot
2 | nan_plot = (
3 |
4 |     # Specify data and mapping
5 |     ggplot(
6 |         data = df_nan_plot,
7 |         mapping = aes(x = "col", y = "pct", fill = "nan_obs")
8 |     ) +
9 |
10 |    # Select the geom for stacked bar plots
11 |    geom_col() +
12 |
13 |    # Assign labels to slices with geom_text()
14 |    geom_text(
15 |        mapping = aes(label = "pct"),
16 |        position = position_stack(vjust = 0.5),
17 |        size = 10,
18 |        fontweight = "bold",
19 |        color = "white"
20 |    ) +
21 |
22 |    # Display column names on the y axis
23 |    coord_flip() +
24 |
25 |    # Control colors use to represent categories
26 |    scale_fill_manual(
27 |        values = {
28 |            "Not missing": "dodgerblue",
29 |            "Missing": "firebrick"
30 |        }
31 |    ) +
32 |
33 |    # Edit axis labels and title
34 |    labs(
35 |        x = "",
36 |        y = "",
37 |        fill = "",

```

```

38     title = "Missing Values Across Columns in Athletics Data (%)"
39   ) +
40
41   # Use the global theme
42   global_theme +
43
44   # Reverse the order of the legend
45   guides(fill = guide_legend(reverse = True))
46 )

1 import warnings
2 warnings.filterwarnings("ignore")
3 nan_plot

```

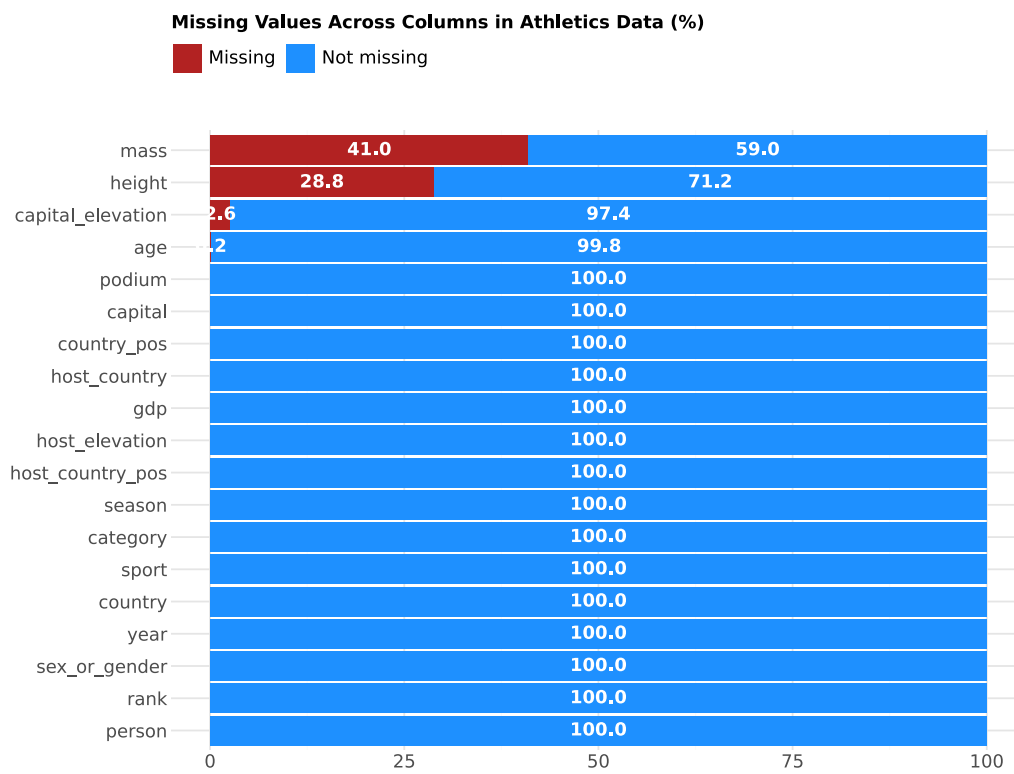


Figure 1: Stacked bar plot of missing values across columns

```

1 # Remove column missing for almost half of the data points
2 df = df.drop(columns = ["mass"])

```

```

1 | # Remove rows with missing values
2 | df = df.dropna().reset_index(drop = True)
3 |
4 | print(df)

```

	person	rank	age	sex_or_gender	height	...	gdp	\
0	Aída Román	7	24	female	1.68	...	1414187193992.0	
1	Aída Román	2	24	female	1.68	...	1414187193992.0	
...	
7801	Valentin Foubert	10	17	male	1.65	...	2782905325625.0	
7802	Yuna Kasai	6	15	female	1.65	...	4231141201863.0	

	country_pos	capital	capital_elevation	podium
0	POINT(-102.000...	Mexico City	2.240k	Not on podium
1	POINT(-102.000...	Mexico City	2.240k	On podium
...
7801	POINT(2.000000...	Paris	48.000	Not on podium
7802	POINT(136.0000...	Tokyo	6.000	Not on podium

[7803 rows x 18 columns]

Calculating distances

We use the geopy library to calculate the distances between athlete's home and the place of olympic games. Originally the data is stored as strings in the form of 'POINT(X,Y)' where we need them to be tuples of the pandas Point type. The following cells do just that.

```

1 | # Change the string "POINT(X,Y) into a tuple of strings ["X", "Y"]
2 |
3 | point_to_list = lambda x: x[6:-1].split()
4 |
5 | df["country_pos_num"] = (
6 |     df.loc[:, "country_pos"]
7 |     .astype("string")
8 |     .map(point_to_list)
9 | )
10 | df["host_country_pos_num"] = (
11 |     df.loc[:, "host_country_pos"]
12 |     .astype("string")

```

```

13 |     .map(point_to_list)
14 | )

1 | print(df.loc[:,["country_pos_num", "host_country_pos_num"]])

      country_pos_num host_country_pos_num
0      [-102.000000, ... [-2.000000, 54...
1      [-102.000000, ... [-2.000000, 54...
...
7801 [2.000000, 47.... [8.231973, 46....
7802 [136.000000, 3... [8.231973, 46....

[7803 rows x 2 columns]

1 | # Turn the tuple of strings ["X", "Y"] into a tuple of floats [X, Y]
2 |
3 | final = lambda x: list(map(float, x))
4 |
5 | df["country_pos_num"] = df.loc[:, "country_pos_num"].map(final)
6 | df["host_country_pos_num"] = (
7 |     df.loc[:, "host_country_pos_num"]
8 |     .map(final)
9 | )
10 | print(df.loc[:,["country_pos_num", "host_country_pos_num"]])

      country_pos_num host_country_pos_num
0      [-102.0, 23.0]      [-2.0, 54.6]
1      [-102.0, 23.0]      [-2.0, 54.6]
...
7801      [2.0, 47.0] [8.231973, 46....
7802 [136.0, 35.0] [8.231973, 46....

[7803 rows x 2 columns]

1 | # Turn the tuples into geometry Points
2 |
3 | from shapely.geometry import Point
4 |
5 | points = lambda x: Point(x[0],x[1])
6 |
7 | df["country_position"] = df.loc[:, "country_pos_num"].map(points)

```

```

8 df["host_country_position"] = (
9     df.loc[:, "host_country_pos_num"]
10     .map(points)
11 )

```

```

1 df = df.drop(columns = [
2     "country_pos",
3     "host_country_pos",
4     "country_pos_num",
5     "host_country_pos_num",
6 ])

```

```
1 print(df)
```

	person	rank	age	sex_or_gender	height	...	capital	\
0	Aída Román	7	24	female	1.68	...	Mexico City	
1	Aída Román	2	24	female	1.68	...	Mexico City	
...	
7801	Valentin Foubert	10	17	male	1.65	...	Paris	
7802	Yuna Kasai	6	15	female	1.65	...	Tokyo	

	capital_elevation	podium	country_position	host_country_position
0	2.240k	Not on podium	POINT (-102 23)	POINT (-2 54.6)
1	2.240k	On podium	POINT (-102 23)	POINT (-2 54.6)
...
7801	48.000	Not on podium	POINT (2 47)	POINT (8.23197...
7802	6.000	Not on podium	POINT (136 35)	POINT (8.23197...

[7803 rows x 18 columns]

```

1 # Calculate the distances between home city and olympic city based on geometric points
2
3 def coordinate_distance_calculator(
4     input_coordinate_list=df.loc[:, "country_position"],
5     target_coordinate_list=df.loc[:, "host_country_position"]
6 ):
7     calculated_distance_df = []
8     for i in range(len(input_coordinate_list)):
9         input_coordinate = input_coordinate_list[i]
10        target_coordinate = target_coordinate_list[i]

```

```

11     coordinates_df = gpd.GeoDataFrame(
12         {'geometry': [target_coordinate, input_coordinate]},
13         crs='EPSG:4326',
14     )
15     coordinates_df = coordinates_df.to_crs('EPSG:5234')
16     distance = coordinates_df.geometry.iloc[0].distance(
17         coordinates_df.geometry.iloc[1])
18     calculated_distance_df.append(distance)
19     return calculated_distance_df
20
21 df['calculated_distance'] = coordinate_distance_calculator()
22 print(df)

```

	person	rank	age	sex_or_gender	height	...	capital_elevation	\
0	Aída Román	7	24	female	1.68	...	2.240k	
1	Aída Román	2	24	female	1.68	...	2.240k	
...	
7801	Valentin Foubert	10	17	male	1.65	...	48.000	
7802	Yuna Kasai	6	15	female	1.65	...	6.000	

	podium	country_position	host_country_position	calculated_distance
0	Not on podium	POINT (-102 23)	POINT (-2 54.6)	9.182M
1	On podium	POINT (-102 23)	POINT (-2 54.6)	9.182M
...
7801	Not on podium	POINT (2 47)	POINT (8.23197...	632.847k
7802	Not on podium	POINT (136 35)	POINT (8.23197...	10.523M

[7803 rows x 19 columns]

Create indicator (a.k.a. dummy) variables

We create dummy variables for the categorical variables, such as country, sport (discipline) and sex/gender of the athlete.

```

1 # Create dummies from the qualitative features listed
2 features = ["country", "sport", "sex_or_gender"]
3 prefixes = ["Country", "Sport", "Gender"]
4 df = pd.get_dummies(
5     df,
6     columns = features,

```

```

7 |     prefix =prefixes,
8 |     prefix_sep = ":",
9 |     drop_first = True,
10 | )
11 |
12 | print(df)

```

```

      person rank age height year ... Sport: wrestling \
0      Aída Román    7  24   1.68  2012 ...           False
1      Aída Román    2  24   1.68  2012 ...           False
...
7801 Valentin Foubert   10  17   1.65  2020 ...           False
7802      Yuna Kasai    6  15   1.65  2020 ...           False

```

```

      Sport: épée fencing Gender: intersex woman Gender: male \
0           False           False           False
1           False           False           False
...
7801           False           False           True
7802           False           False           False

```

```

      Gender: non-binary
0           False
1           False
...
7801           False
7802           False

```

```
[7803 rows x 231 columns]
```

Modelling

We have prepared 4 models to see which might best predict the results of olympics. The first two predict the rank of the athlete using simple linear regression or a tree-based regression. The second two focus on the prediction if the athlete will end on the podium or not, using simple classification or a tree-based classification.

Linear regression

Test: Multiple linear regression model only for height

Linear regression is computed for just one input as a check of working code.

```
1 # Fit a multiple linear regression model and output the coefficients
2 # Assign the rank column to the variable Y
3 Y = df.loc[:, "rank"]
4
5 # Assign the capital elevation column to the variable X
6 X = pd.DataFrame(df.loc[:, "capital_elevation"])
7
8 print(X)
9 print(Y)
```

	capital_elevation
0	2.240k
1	2.240k
...	...
7801	48.000
7802	6.000

```
[7803 rows x 1 columns]
0      7
1      2
..
7801  10
7802   6
Name: rank, Length: 7803, dtype: int64
```

```
1 # Fit a multiple linear regression model
2 mlr_1 = LinearRegression().fit(X, Y)
3
4 # Put the coefficients in a DataFrame and output it
5 print(pd.DataFrame(
6     {
7         "Term": ["Intercept", "Elevation"],
8         "Estimate": [mlr_1.intercept_.tolist()] + mlr_1.coef_.tolist()
9     }
10 ).round(4))
```

	Term	Estimate
0	Intercept	11.856
1	Elevation	2.300m

```

1 # Compute goodness of fit metrics
2 # Define a function to compute the RMSE and the R^2
3 def regression_gof(model, X, Y):
4
5     # Compute the predicted values
6     Y_hat = model.predict(X)
7
8     # Compute the residuals
9     e_hat = Y - Y_hat
10
11    # Compute the RSS
12    RSS = sum(e_hat ** 2)
13
14    # Compute the RMSE
15    RMSE = (RSS / X.shape[0]) ** (1/2)
16
17    # Compute the mean percentage deviation from the regression line
18    DEV = RMSE / Y.mean() * 100
19
20    # Compute R^2
21    R2 = Y.corr(pd.Series(Y_hat)) ** 2
22
23    # Return the RMSE and R^2
24    return (round(RMSE, 2), round(DEV, 2), round(R2, 3), Y_hat, e_hat)
25
26 # Compute and print the RMSE and the R^2
27 rmse, pct, r2, _, _ = regression_gof(mlr_1, X, Y)
28 print(f"RMSE = {rmse} ({pct}%)")
29 display(Math(f"R^2 = {r2}"))

```

RMSE = 14.48 (117.08%)

$R^2 = 0.004$

```

1 def residual_plot(Y_hat, e_hat, model_name):
2
3     # axis labels and title stub
4     x = "$\mathbf{\hat{f}(x)}$"

```

```

5   y = "$\mathbf{\hat{e}}$"
6   title_ = f"Predicted values [{x}] and residuals [{y}] for "
7
8   return (
9
10      # Specify data and aesthetics
11      ggplot(mapping = aes(x = Y_hat, y = e_hat)) +
12
13      # Select the geom for points
14      geom_point(
15          fill = "dodgerblue",
16          color = "white",
17          size = 3
18      ) +
19
20      # Add a LOESS curve to the plot
21      geom_smooth(
22          se = False,
23          size = 0.75,
24          method = "lowess",
25          span = 0.2
26      ) +
27
28      # Set axis labels and title
29      labs(x = x, y = y, title = title_ + model_name) +
30
31      # Use the global theme
32      global_theme +
33
34      # Adjust plot margin
35      theme(plot_margin = 0.025)
36  )
37
38  # Output the residual plot
39  residual_plot(
40      regression_gof(mlr_1, X, Y)[3],
41      regression_gof(mlr_1, X, Y)[4],
42      "Model 0: Relationship between height and rank"
43  )

```

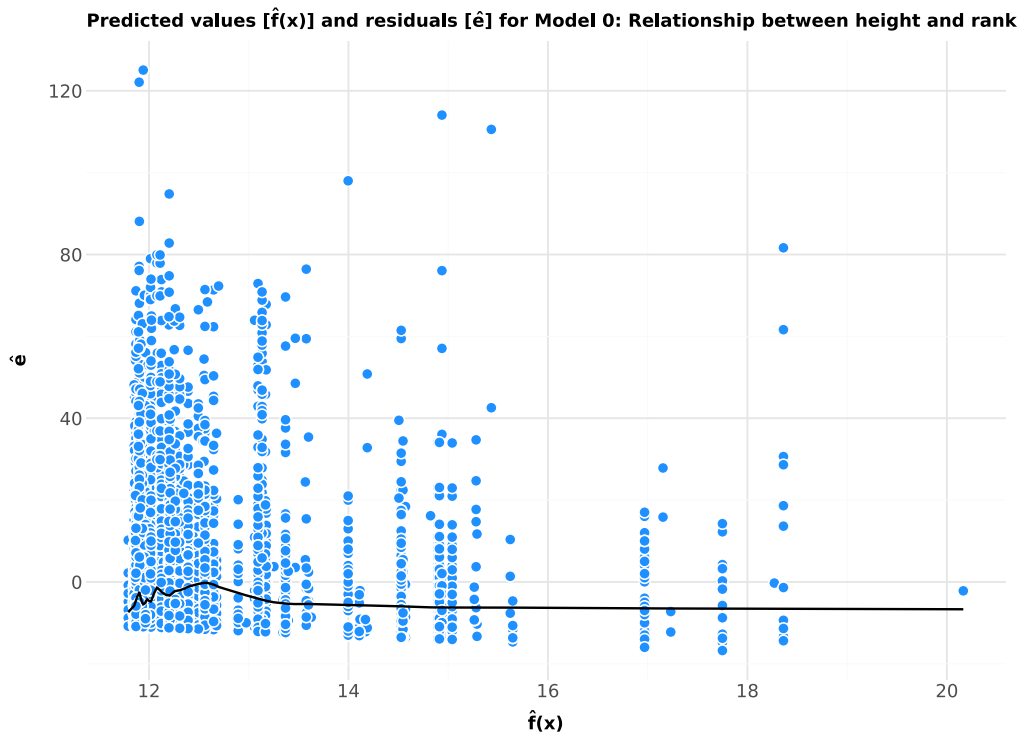


Figure 2: residual plot for height and rank

Multiple linear regression model

Moving onto the full model, we get rid of irrelevant variables and focus only on the ones of interest.

```

1 # Exclude irrelevant variables from the full model
2
3 features_to_exclude = [
4     "person",
5     "rank",
6     "podium",
7     "year",
8     "sex_or_gender",
9     "season",
10    "category",
11    "country",
12    "country_position",

```

```

13     "capital",
14     "sport",
15     "person_id",
16     "event_id",
17     "host_country",
18     "host_country_position"
19 ]
20
21 features_to_include = [
22     column
23     for column in df.columns.tolist()
24     if column not in features_to_exclude
25 ]
26
27 X = df.loc[:, features_to_include]
28 mlr_2 = LinearRegression().fit(X,Y)
29
30 model = pd.DataFrame({
31     "Term": ["Intercept"] + features_to_include,
32     "Estimate": [mlr_2.intercept_.tolist()] + mlr_2.coef_.tolist()
33 }).round(4)
34
35 print(model)

```

	Term	Estimate
0	Intercept	8.825
1	age	0.000
..
220	Gender: male	-0.000
221	Gender: non-bi...	0.000

[222 rows x 2 columns]

```

1 # Compute and print the RMSE and the R^2
2 rmse, pct, r2, _, _ = regression_gof(mlr_2, X, Y)
3 print(f"RMSE = {rmse} ({pct}%)")
4 display(Math(f"R^2 = {r2}"))

```

RMSE = 13.88 (112.27%)

$R^2 = 0.084$

```

1 # Output the residual plot
2 residual_plot(
3     regression_gof(mlr_2, X, Y)[3],
4     regression_gof(mlr_2, X, Y)[4],
5     "Model 1: Linear regression"
6 )

```



Figure 3: Residual plot for all qualities

Tree-based regression

```

1 # Split the available data into a training set and a test set
2 X_train, X_test, Y_train, Y_test = train_test_split(
3     X,
4     Y,
5     test_size = 0.2,
6     random_state = 42,
7 )

```

```

1 # Fit and plot a classification tree
2 # Initialize classification tree estimator
3 reg_tree = DecisionTreeRegressor(max_depth = 3)
4
5 # Fit the classification tree in the training data
6 est_reg_tree = reg_tree.fit(X_train, Y_train)
7
8 # Create a plot of the classification tree
9 plot_reg_tree, axis = plt.subplots(figsize = (10, 6))
10 plot_tree(
11     decision_tree = est_reg_tree,
12     feature_names = X_train.columns.tolist(),
13     label = "root",
14     impurity = False,
15     precision = 2,
16     ax = axis,
17     fontsize = 10
18 )
19 plt.close()
20 plot_reg_tree

```

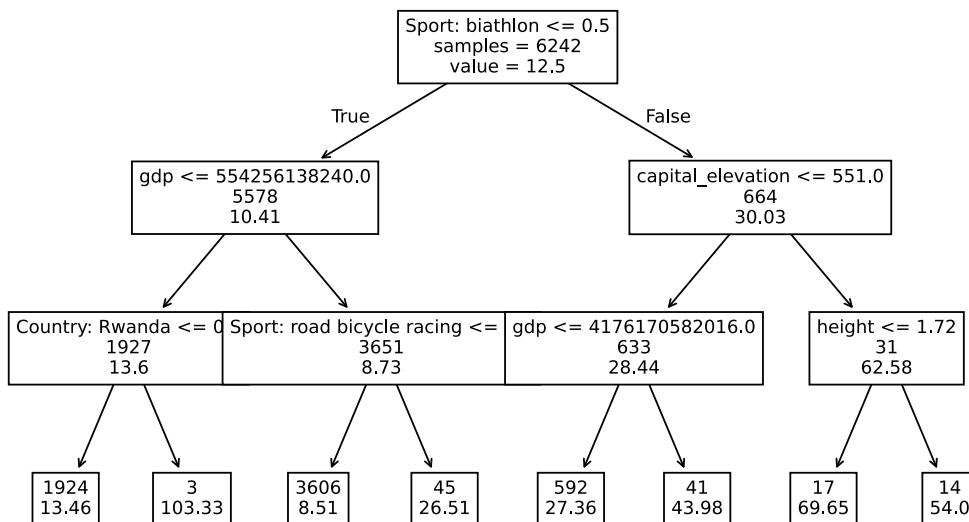


Figure 4: Classification tree plot for model 2

```

1 # Create a list of values of alpha
2 alphas = list(np.geomspace(0.01, 10, 100))
3
4 # Initialize 5-fold CV
5 CV_prune_reg_tree = GridSearchCV(
6     estimator = DecisionTreeRegressor(),
7     param_grid = {"ccp_alpha": alphas},
8     scoring = "neg_root_mean_squared_error"
9 )
10 est_cv_prune_reg_tree = CV_prune_reg_tree.fit(X_train, Y_train)

1 # Extract optimal hyperparameter value
2 best_alpha = est_cv_prune_reg_tree.best_params_["ccp_alpha"]
3
4 # Assign CV estimate of RMSE to new variable
5 RMSE_cv_prune_reg_tree = round(
6     est_cv_prune_reg_tree.best_score_ * (-1),
7     3,
8 )
9
10 # Compute percentage deviation from observed target
11 DEV_cv_prune_reg_tree = round(
12     RMSE_cv_prune_reg_tree / Y_train.mean() * 100,
13     2,
14 )
15
16 # Print optimal hyperparameter value, runtime, and CV estimate of RMSE
17 print(f"Optimal alpha = {round(best_alpha, 3)}")
18 print(f"CV RMSE = {RMSE_cv_prune_reg_tree} ({DEV_cv_prune_reg_tree}%)")

```

Optimal alpha = 0.305
CV RMSE = 12.519 (100.15%)

```

1 # Initialize regression tree estimator
2 prune_reg_tree = DecisionTreeRegressor(ccp_alpha = best_alpha)
3
4 # Fit the regression tree in the training data with optimal alpha
5 est_prune_reg_tree = prune_reg_tree.fit(X_train, Y_train)
6
7 # Create a plot of the regression tree

```

```
8 | plot_prune_reg_tree, axis = plt.subplots(figsize = (30,30))
9 | plot_tree(
10 |     decision_tree = est_prune_reg_tree,
11 |     feature_names = X_train.columns.tolist(),
12 |     label = "root",
13 |     impurity = False,
14 |     precision = 2,
15 |     ax = axis,
16 |     fontsize = 10
17 | )
18 | plt.close()
19 |
20 | # Output the regression tree plot
21 | plot_prune_reg_tree
```




Figure 6: Residual plot for regression in model 2

Comparison of regression models

As we can see, both models are quite unreliable, yielding high RMSE, as well as low R^2 values, which indicates that they are not a good fit. However, the tree-based model shows a considerably better result than the simple model, with R^2 values many times higher. In any way, simplifying the model into a classification problem might help to make it more reliable.

Linear classification

Test: Classification only of capital_elevation

```

1 # Construct a DataFrame containing the data to be plotted
2 df_density_plot = df.melt(
3     id_vars = "podium",
4     value_vars = [
5         "capital_elevation"
6     ],
7     var_name = "feature",
8     value_name = "value"
9 )
10
11 # Encode the target for plotting
12 encode_target = lambda x: (
13     "Not on the podium"
14     if x == "Not on podium"
15     else "On the Podium"
16 )
17 df_density_plot["podium"] = (
18     df_density_plot.loc[:, "podium"]
19     .astype("string")
20     .map(encode_target)
21 )

```



```

1 # Create the plot
2
3 # Title
4 title = "Distribution of Quantitative Features Across Values of Podium"
5
6 density_plot = (
7
8     # Specify data and aesthetics
9     ggplot(
10         data = df_density_plot,
11         mapping = aes(x = "value", fill = "podium")
12     ) +
13
14     # Select the geom for points
15     geom_density(alpha = 0.75) +
16
17     # Use facet_wrap() to create multiples
18     facet_wrap(

```

```

19     facets = "feature",
20     nrow = 4,
21     ncol = 1,
22     scales = "free"
23   ) +
24
25   # Control colors used to represent target values
26   scale_fill_manual(
27     values = {
28       "Not on the podium": "dodgerblue",
29       "On the Podium": "firebrick"
30     }
31   ) +
32
33   # Set axis labels and title
34   labs(x = "", y = "Density", fill = "", title = title) +
35
36   # Use the global theme
37   global_theme +
38
39   # Make the plot larger
40   # and control appearance of control titles of small multiples
41   theme(
42     figure_size = (8, 10),
43     strip_text = element_text(face = "bold", size = 10, ha = "left")
44   )
45 )
46
47 # Output the plot
48 density_plot

```

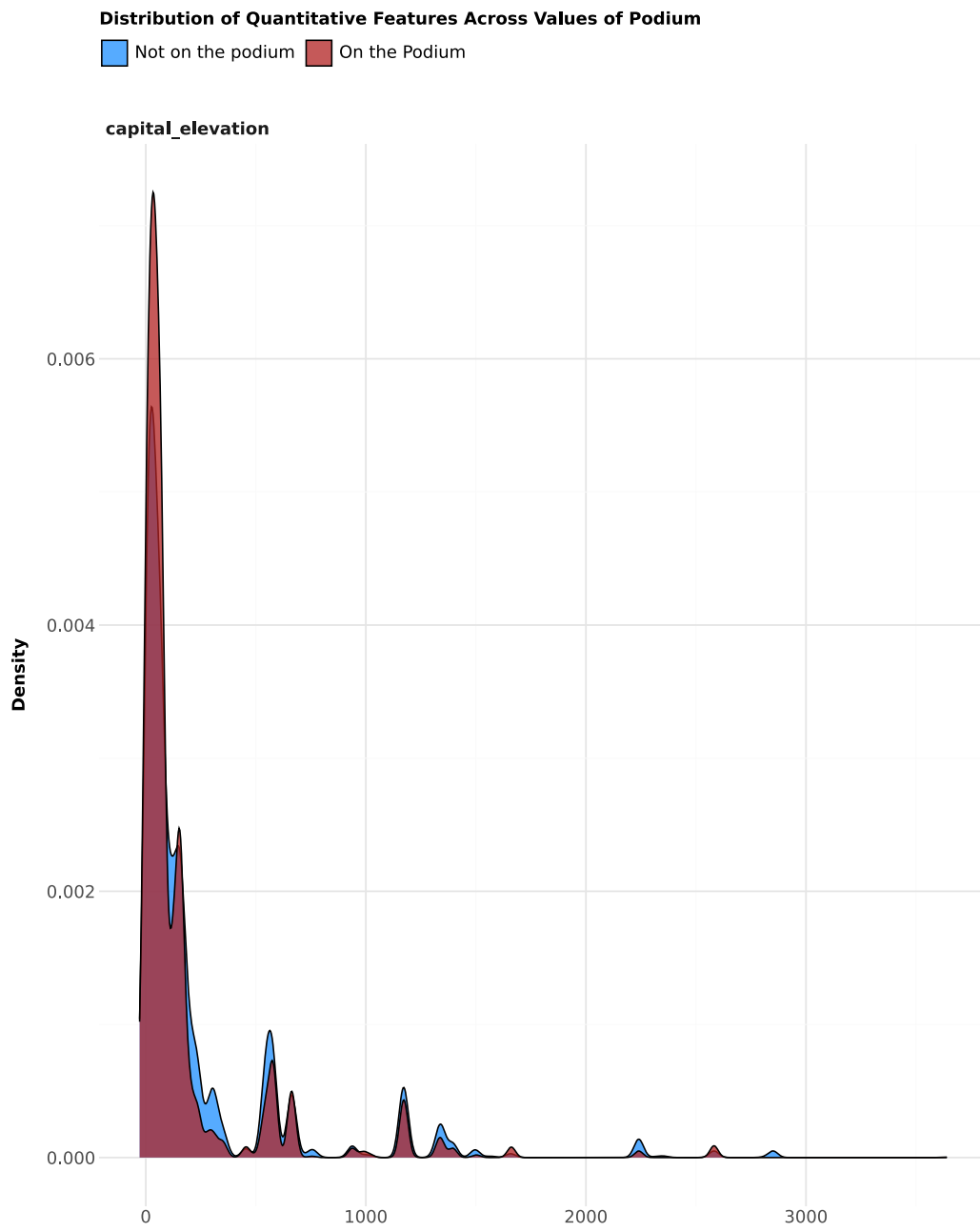


Figure 7: Density plot for distribution across Podium values

Logistic classification

```
1 # Fit a multiple linear regression model and output the coefficients
2 df["target"] = (df["podium"] == "On podium").astype(int)
3
4 Y = df.loc[:, "target"]
5
6 # Fit a simple logistic regression model
7 logit_1 = LogisticRegression(penalty = None, max_iter = 500).fit(X, Y)
8
9 # Put the coefficients in a DataFrame and output it
10
11 print(pd.DataFrame(
12     {
13         "Term": ["Intercept"] + features_to_include,
14         "Estimate": logit_1.intercept_.tolist() + logit_1.coef_[0].tolist()
15     }
16 ).round(4))
```

	Term	Estimate
0	Intercept	-0.000
1	age	-0.000
..
220	Gender: male	-0.000
221	Gender: non-bi...	0.000

[222 rows x 2 columns]

```
1 # Compute goodness of fit metrics and output them
2 # Define a function to plot the ROC curve
3
4 def plot_roc_curve(model, model_name, X, Y, AUC):
5
6     # Use a scikit_learn function to compute the ROC curve
7     roc = roc_curve(Y, model.predict_proba(X)[:, 1])
8
9     # Put the false and true positive rates in a DataFrame for plotting
10    tpr_fpr = pd.DataFrame({"FPR": roc[0], "TPR": roc[1]})
11
12    # Create the plot
13    return (
```

```

14
15 # Specify data and aesthetics
16 ggplot(
17   data = tpr_fpr,
18   mapping = aes(x = "FPR", y = "TPR")
19 ) +
20
21 # Use the geom for lines
22 geom_line(color = "dodgerblue") +
23
24 # Draw a 45-degree line through the origin
25 geom_abline(
26   intercept = 0,
27   slope = 1,
28   color = "firebrick",
29   linetype = "dashed"
30 ) +
31
32 # Display the AUC on the plot
33 annotate(
34   "text",
35   label = "AUC = " + str(round(AUC, 3)),
36   x = 0.875,
37   y = 0.1,
38   fontweight = "bold",
39   size = 10
40 ) +
41
42 # Control axis labels and title
43 labs(
44   x = "False positive rate",
45   y = "True positive rate",
46   title = "ROC Curve for " + model_name
47 ) +
48
49 #Use the global theme
50 global_theme
51 )
52
53 # Define a function to construct a confusion matrix

```

```

54 def confusion_matrix(Y, Y_hat):
55
56     # Construct a confusion matrix
57     CM = pd.crosstab(
58         index = Y,
59         columns = Y_hat,
60         rownames = ["Observed values"],
61         colnames = ["Predicted values"]
62     )
63
64     # Add a column of zeros if predictions are all zeroes
65     if CM.shape[1] == 1:
66         CM[1,0] = 0
67
68     return CM
69
70 # Define a function to compute GOF metrics for classification models
71 def classification_gof(model, X, Y, model_name = ""):
72
73     # Estimate response probabilities
74     p_hat = model.predict_proba(X)[:, 1]
75
76     # Predict classes
77     Y_hat = model.predict(X)
78
79     # Compute the error rate
80     err_bool = Y != Y_hat
81     err = pd.Series(err_bool).mean()
82
83     # Construct a confusion matrix
84     CM = confusion_matrix(Y, Y_hat)
85
86     # Compute precision, recall, and f1
87     P = CM.iloc[1, 1] / (CM.iloc[1, 1] + CM.iloc[0, 1])
88     R = CM.iloc[1, 1] / (CM.iloc[1, 1] + CM.iloc[1, 0])
89     F1 = 2 * P * R / (P + R)
90
91     # Compute the AUC
92     AUC = roc_auc_score(Y, model.predict_proba(X)[:, 1])
93

```

```

94     # Create a ROC curve plot
95     ROC = plot_roc_curve(model, model_name, X, Y, AUC)
96
97     return (
98         p_hat,
99         round(err, 3),
100        CM,
101        round(P, 3),
102        round(R, 3),
103        round(F1, 3),
104        ROC,
105        round(AUC, 3),
106    )
107
108    # Compute and print GOF metrics
109    model_name = "Model 3: Logistic Regression"
110    p_hat, err, CM, P, R, F1, ROC, AUC = classification_gof(
111        logit_1,
112        X,
113        Y,
114        model_name = model_name,
115    )
116    print(f"Error rate = {err}")
117    print(f"Precision = {P}")
118    print(f"Recall = {R}")
119    print(f"F1 = {F1}")
120
121    # Output the confusion matrix
122    print(CM)

```

Error rate = 0.299

Precision = 0.402

Recall = 0.16

F1 = 0.228

Predicted values 0 1

Observed values

0 5127 513

1 1818 345

```

1 | # Plot the ROC curve and compute the AUC
2 | ROC

```

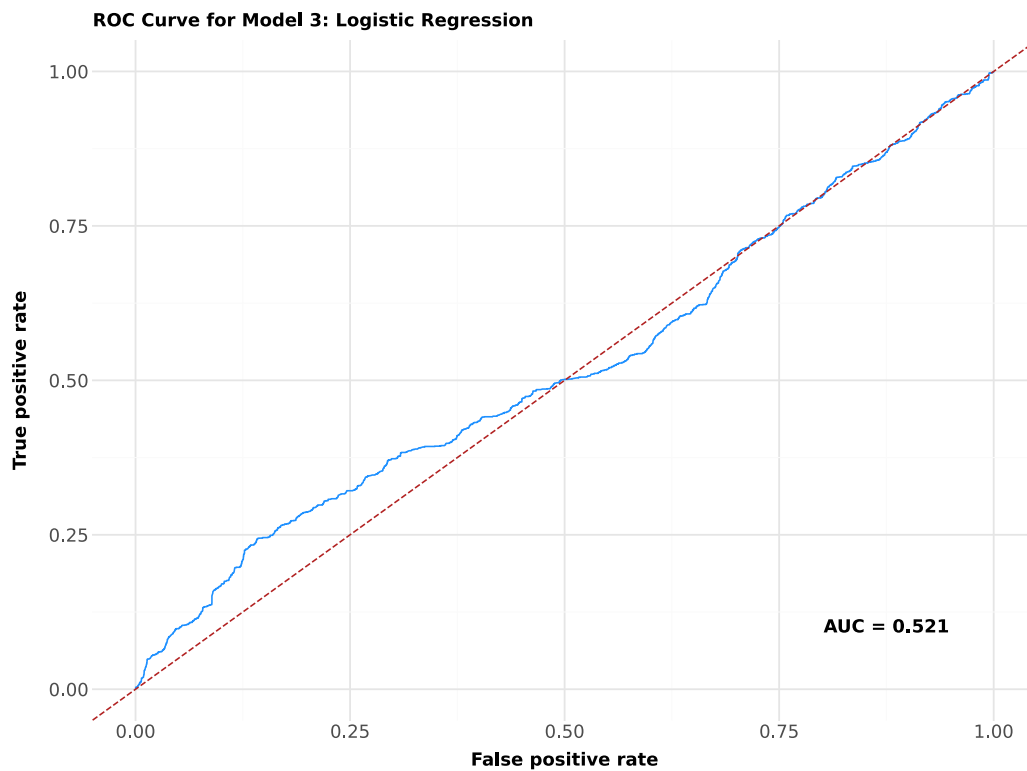


Figure 8: ROC Curve plot for model 3

```

1 | # Create a separation plot
2 | # Define a function to reuse the code later
3 |
4 | encode_target = lambda x: (
5 |     "Not on the podium" if x == "0" else "On the podium"
6 | )
7 |
8 | def separation_plot(Y, p_hat):
9 |
10 |     # Start by constructing a DataFrame
11 |     # with predicted and observed values of the target
12 |     df_separation_plot = pd.DataFrame(
13 |         {

```

```

14         "Y": Y.astype("string").map(encode_target),
15         "p_hat": p_hat
16     }
17 )
18
19 # Order the DataFrame by predicted values
20 # and construct an index variable
21 df_separation_plot.sort_values(by = "p_hat", inplace = True)
22 df_separation_plot.reset_index(drop = True, inplace = True)
23 df_separation_plot["idx"] = df_separation_plot.index
24
25 # Create the plot
26 return (
27     # Specify data and aesthetics
28     ggplot(
29         data = df_separation_plot,
30         mapping = aes(x = "idx", color = "Y")
31     ) +
32
33     # Use the geom for line segments
34     geom_segment(mapping = aes(xend = "idx", y = 0, yend = 1)) +
35
36     # Use the geom for lines
37     geom_line(
38         mapping = aes(y = "p_hat"),
39         color = "black"
40     ) +
41
42     # Control colors used to represent observed values
43     scale_color_manual(
44         values = {
45             "On the podium": "firebrick",
46             "Not on the podium": "dodgerblue"
47         }
48     ) +
49
50     # Control axis labels and title
51     labs(
52         x = "$\mathbf{i}$",
53         y = "$\mathbf{\hat{p}(x_i)}$",

```

```

54     color = "Observed Values",
55     title = "Separation plot for " + model_name
56   ) +
57
58   # Use the global theme
59   global_theme +
60
61   # Adjust plot margins
62   theme(plot_margin = 0.025)
63 )
64
65 # Output the separation plot
66 separation_plot(Y, p_hat)

```

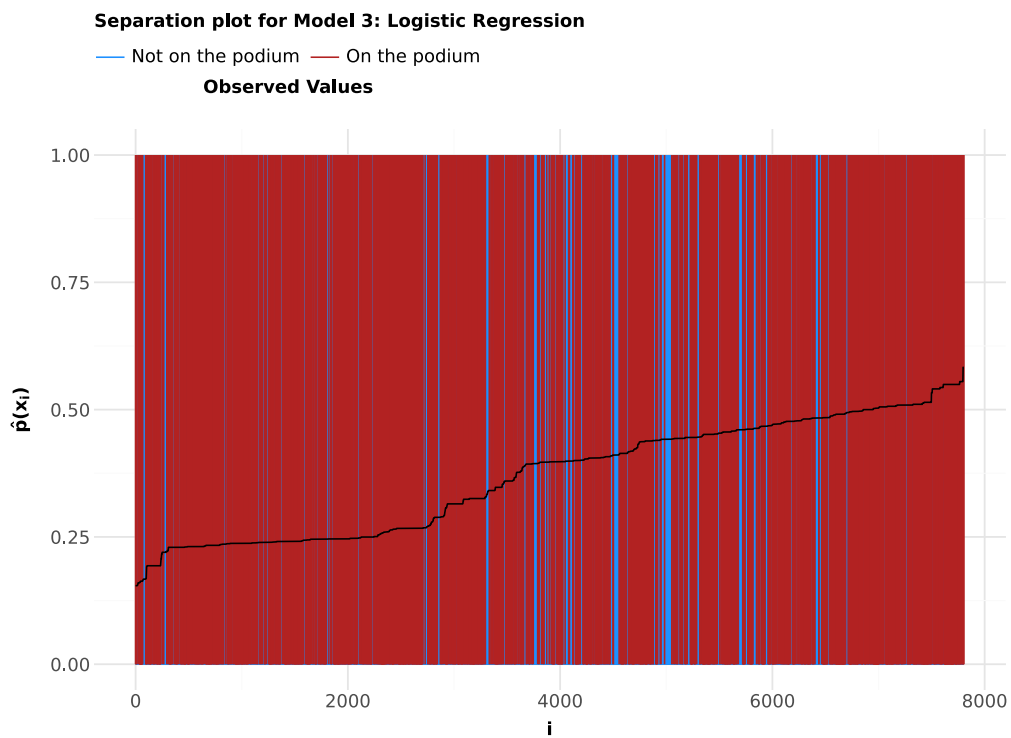


Figure 9: Separation plot for logistic regression

With the logistic regression model always predicting that the athlete is not on the podium, it seems it's worse than either one of the linear regression models.

Tree-based classification

```
1 # Split the available data into a training set and a test set
2 X_train, X_test, Y_train, Y_test = train_test_split(
3     X,
4     Y,
5     test_size = 0.2,
6     random_state = 42,
7 )

1 # Fit and plot a classification tree
2 # Initialize classification tree estimator
3 class_tree = DecisionTreeClassifier(max_depth = 3)
4
5 # Fit the classification tree in the training data with optimal alpha
6 est_class_tree = class_tree.fit(X_train, Y_train)
7
8 # Create a plot of the classification tree
9 plot_class_tree, axis = plt.subplots(figsize = (10, 6))
10 plot_tree(
11     decision_tree = est_class_tree,
12     feature_names = X_train.columns.tolist(),
13     label = "root",
14     impurity = False,
15     precision = 2,
16     ax = axis,
17     fontsize = 10
18 )
19 plt.close()
20 plot_class_tree
```

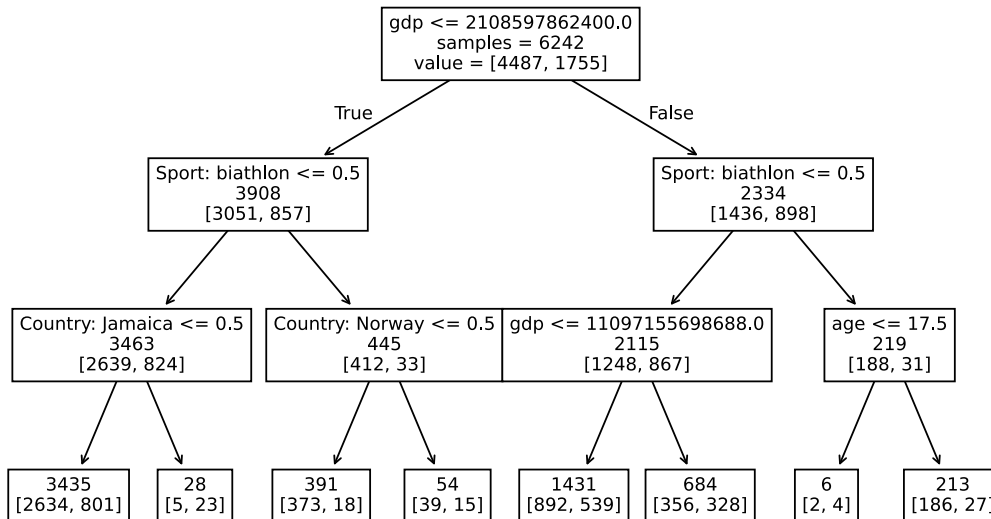


Figure 10: Classification tree plot for model 4

```

1 # Use 5-fold CV to tune a classification tree
2 # Create a list of values of alpha
3 alphas = list(np.geomspace(0.0005, 10, 100))
4
5 # Initialize 5-fold CV
6 CV_prune_class_tree = GridSearchCV(
7     estimator = DecisionTreeClassifier(),
8     param_grid = {"ccp_alpha": alphas},
9     scoring = "f1"
10 )
11
12 # Implement the 5-fold CV procedure
13 est_cv_prune_class_tree = CV_prune_class_tree.fit(X_train, Y_train)
14
15 # Print the results of the CV procedure
16 # Extract optimal hyperparameter value
17 best_alpha = est_cv_prune_class_tree.best_params_["ccp_alpha"]
18
19 # Assign CV estimate of f1 to new variable
20 F1_cv_prune_class_tree = round(est_cv_prune_class_tree.best_score_, 3)
21

```

```

8 | # Print optimal hyperparameter value, runtime,
9 | # and the CV estimate of the RMSE
10 | print(f"Optimal alpha = {round(best_alpha, 3)}")
11 | print(f"CV F1 = {F1_CV_prune_class_tree}")

Optimal alpha = 0.0
CV F1 = 0.48

1 | # Fit and plot a classification tree
2 | # Initialize classification tree estimator
3 | class_tree = DecisionTreeClassifier(ccp_alpha = best_alpha)
4 |
5 | # Fit the classification tree in the training data with optimal alpha
6 | est_class_tree = class_tree.fit(X_train, Y_train)
7 |
8 | # Create a plot of the classification tree
9 | plot_class_tree, axis = plt.subplots(figsize = (30, 20))
10 | plot_tree(
11 |     decision_tree = est_class_tree,
12 |     feature_names = X_train.columns.tolist(),
13 |     label = "root",
14 |     impurity = False,
15 |     precision = 2,
16 |     ax = axis,
17 |     fontsize = 10
18 | )
19 | plt.close()
20 | plot_class_tree

```

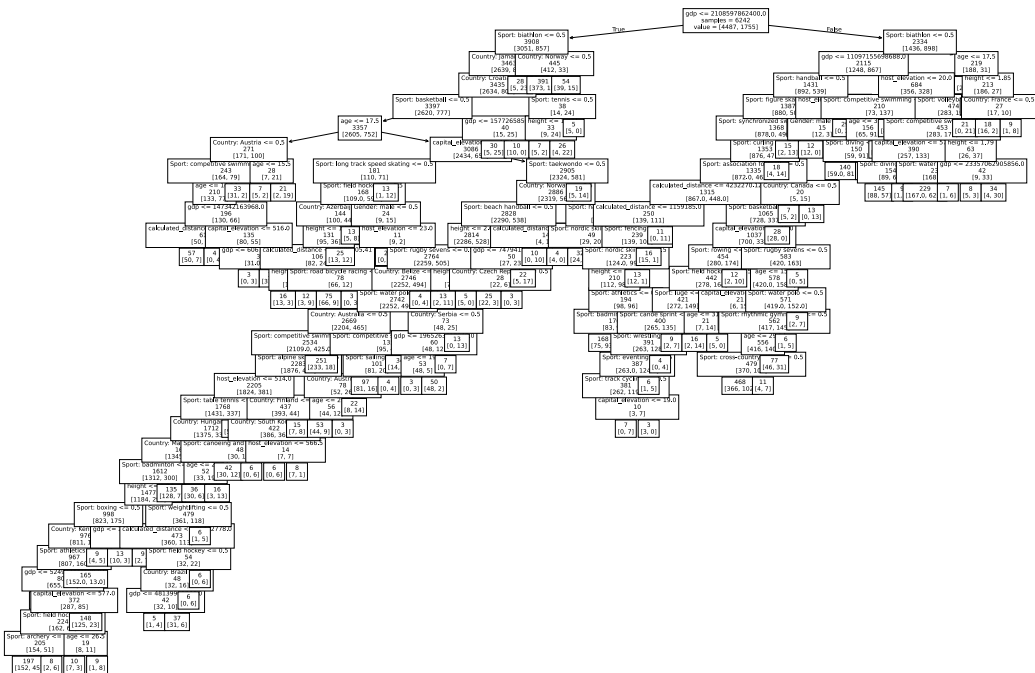


Figure 11: Classification tree plot

```

1 # Predict the target in the set and evaluate performance
2 # Compute predicted values in the test set
3 Y_hat = est_class_tree.predict(X_test)
4
5 # Estimate response probabilities
6 p_hat = est_class_tree.predict_proba(X_test)[: , 1]
7
8 # Compute the error rate
9 err_bool = Y_test != Y_hat
10 err = pd.Series(err_bool).mean()
11
12 # Construct a confusion matrix
13 CM = pd.crosstab(
14     index = Y_test,
15     columns = Y_hat,
16     rownames = ["Observed values"],
17     colnames = ["Predicted values"]
18 )
19

```

```

20 # Compute precision, recall, and f1
21 P = CM.iloc[1, 1] / (CM.iloc[1, 1] + CM.iloc[0, 1])
22 R = CM.iloc[1, 1] / (CM.iloc[1, 1] + CM.iloc[1, 0])
23 F1 = 2 * P * R / (P + R)
24
25 # Compute the AUC
26 AUC = roc_auc_score(Y_test, p_hat)

1 # Create a separation plot
2 # Start by constructing a DataFrame
3 # with predicted and observed values of the target
4 df_separation_plot = pd.DataFrame(
5     {
6         "Y": (
7             Y_test.astype("string")
8             .map(lambda x: (
9                 "Not on the podium"
10                if x == "0"
11                else "On the podium")
12            )
13        ),
14        "p_hat": p_hat
15    }
16 )
17
18 # Order the DataFrame by predicted values
19 # and construct an index variable
20 df_separation_plot.sort_values(by = "p_hat", inplace = True)
21 df_separation_plot.reset_index(drop = True, inplace = True)
22 df_separation_plot["idx"] = df_separation_plot.index
23
24 # Specify plot subtitle
25 alpha = est_CV_prune_class_tree.best_params_["ccp_alpha"]
26 subtitle = [
27     "Optimal alpha = " + str(alpha),
28     "Err = " + str(round(err * 100, 1)) + "%",
29     "F1 = " + str(round(F1, 3))
30 ]
31 subtitle = ", ".join(subtitle)
32

```

```

33 # Create the plot
34 separation_plot = (
35
36     # Specify data and aesthetics
37     ggplot(
38         data = df_separation_plot,
39         mapping = aes(x = "idx", color = "Y")
40     ) +
41
42     # Use the geom for line segments
43     geom_segment(
44         mapping = aes(xend = "idx", y = 0, yend = 1),
45         size = 1
46     ) +
47
48     # Use the geom for lines
49     geom_line(
50         mapping = aes(y = "p_hat"),
51         color = "black"
52     ) +
53
54     # Control colors used to represent observed values
55     scale_color_manual(
56         values = {
57             "On the podium": "firebrick",
58             "Not on the podium": "dodgerblue"
59         }
60     ) +
61     # Control axis labels and title
62     labs(
63         x = " $\mathbf{i}$ ",
64         y = " $\mathbf{\hat{p}(x_i)}$ ",
65         color = "Observed Values",
66         title = "Boosting Test Set Performance for Predicting Podium",
67         subtitle = subtitle
68     ) +
69
70     # Use the global theme
71     global_theme +
72

```

```

73 | # Adjust plot margins
74 | theme(plot_margin = 0.025)
75 | )

1 | # Output the separation plot
2 | separation_plot

```

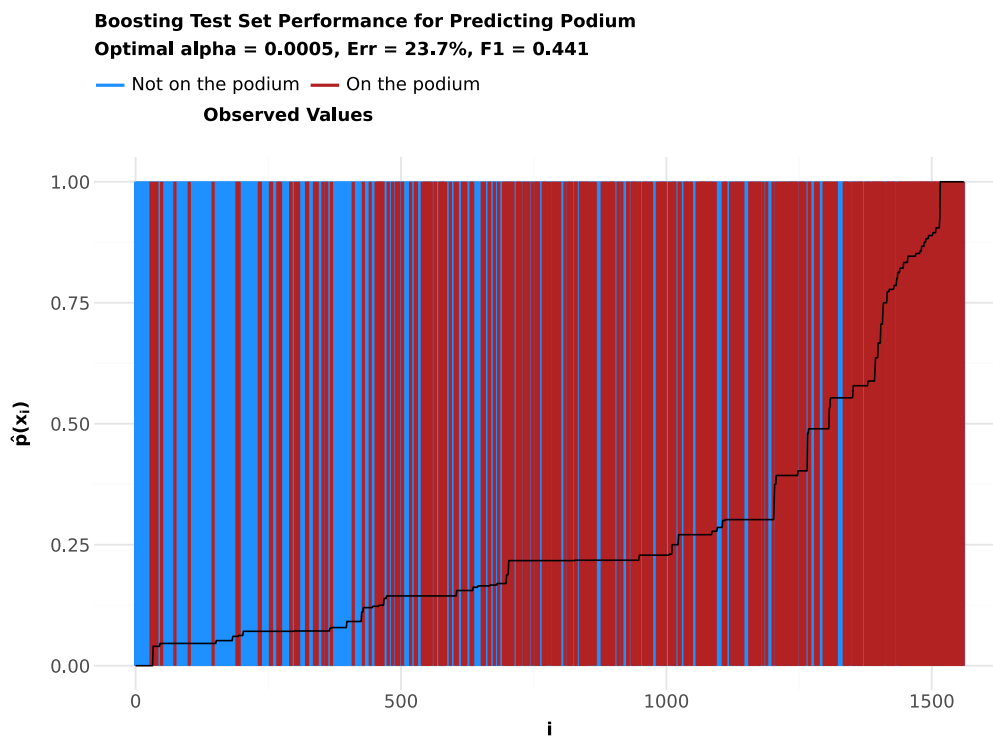


Figure 12: Separation plot

```

1 | # Output the test set confusion matrix
2 | print(CM)

```

Predicted values	0	1
Observed values		
0	1045	108
1	262	146

```

1 # Create a ROC curve
2 # Use a scikit_learn function to compute the ROC curve
3 title = "Test Set ROC Curve for Model 4: Tree-based logistic regression"
4 roc = roc_curve(Y_test, p_hat)
5
6 # Put the false and true positive rates in a DataFrame for plotting
7 tpr_fpr = pd.DataFrame({"FPR": roc[0], "TPR": roc[1]})
8
9 # Create the plot
10 roc_curve = (
11
12     # Specify data and aesthetics
13     ggplot(
14         data = tpr_fpr,
15         mapping = aes(x = "FPR", y = "TPR")
16     ) +
17
18     # Use the geom for lines
19     geom_line(color = "dodgerblue") +
20
21     # Draw a 45-degree line through the origin
22     geom_abline(
23         intercept = 0,
24         slope = 1,
25         color = "firebrick",
26         linetype = "dashed"
27     ) +
28
29     # Display the AUC on the plot
30     annotate(
31         "text",
32         label = "AUC = " + str(round(AUC, 3)),
33         x = 0.875,
34         y = 0.1,
35         fontweight = "bold",
36         size = 10
37     ) +
38
39     # Control axis labels and title
40     labs(

```

```

41     x = "False positive rate",
42     y = "True positive rate",
43     title = title
44 ) +
45
46 # Use the global theme
47 global_theme
48 )
49
50 # Output the ROC curve
51 roc_curve

```

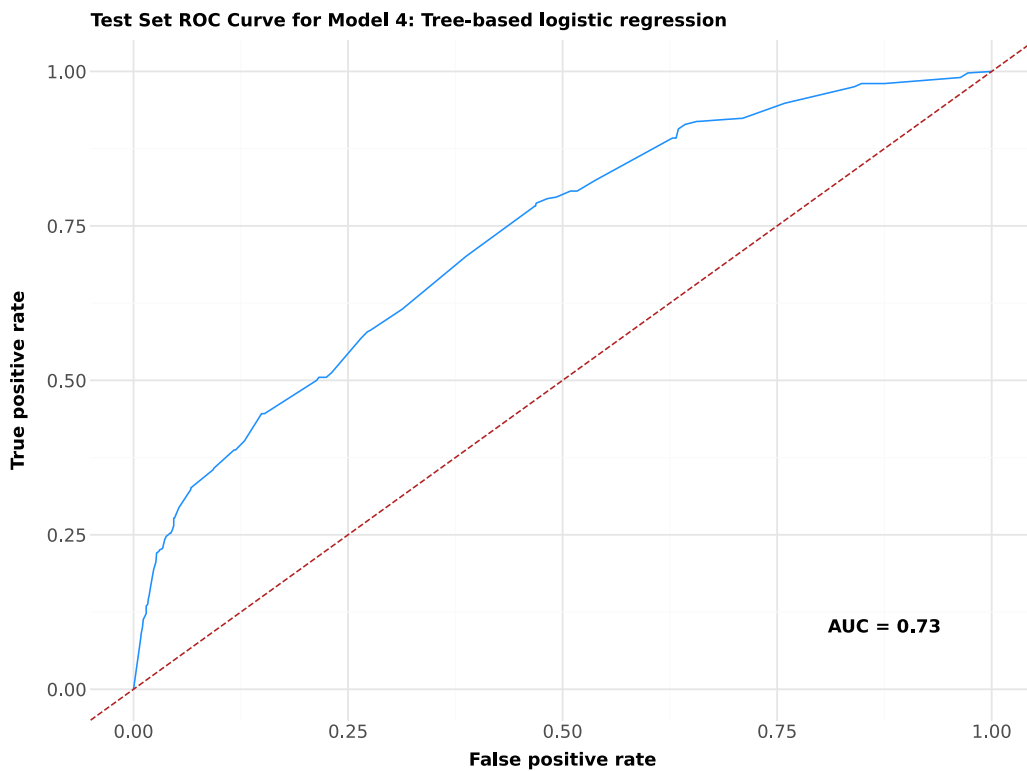


Figure 13: ROC Curve plot for Model 4: Tree-based logistic regression

Comparison of classification models

As we can see, the fourth model yields much better results than the previous ones, providing a somewhat reliable prediction of whether a given athlete will place on the podium.

Conclusion

In this project we tried to see if it was possible to predict athletic performance by using machine learning models and data from Wikidata. We looked at different features like GDP, height, elevation, distance between countries, age and different sports to see if they had some effect on athletic success.

The results showed that the linear regression models were not very good for predicting athletic performance. The R^2 values were very low, which means that the variables only explained a very small amount of the athlete rankings. This may show that athletic performance is much more complicated and not just based on simple linear relationships.

The tree-based regression model worked better than the linear regression model. The decision tree models could show more complicated patterns in the dataset, and variables like GDP, height, mass and elevation appeared many times in the important splits of the trees. This could maybe mean that these variables have some importance for athletic success, but they still cannot explain everything alone.

The logistic regression model also did not perform very strongly. The ROC-AUC score was close to 0.5, which means the model was only a little better than random guessing. The confusion matrix and F1 score also showed that it was difficult for the model to predict correctly which athletes would end on the podium. Finally, the tree-based classification model yielded more reliable results, with the ROC-AUC score closer to 0.7–0.8, which might indicate that it was a suitable way of predicting the results.

Another important part of the project was the data cleaning. There were many missing values and some inconsistent units in the dataset, especially for height and mass. Because of this, several cleaning and preprocessing steps had to be done before the models could be trained properly.

Overall, the project showed that machine learning can help find patterns in athletic data, but also that athletic success is very difficult to predict completely. There are many different factors that can affect athletic performance, and not all of them were included in the dataset.