

Is artificial intelligence smarter than your meet handler?

Predicting the success of individual attempts in powerlifting competitions.



AUTHOR: MONTEL MOORE
SUPERVISOR: ANDREA CALI

PROJECT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE MSC IN DATA SCIENCE

BIRKBECK, UNIVERSITY OF LONDON
DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

This proposal is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text. I give my permission for it to be submitted to a plagiarism detection service.

The proposal may be freely copied and distributed provided the source is explicitly acknowledged.

1 ABSTRACT

There is a growing body of research surrounding the topic of sports outcome prediction. This project outlines a method and implementation of machine learning modelling to predict the likelihood of a powerlifting athlete passing or failing an attempt during competition, using past meet data from the openpowerlifting.org dataset. Extensive data cleaning, feature generation and exploratory data analysis techniques were performed in order to achieve optimal results. Two modelling algorithms, Multi-layer perceptron (MLP) and Histogram-Based Gradient boosted decision tree models (HBGBT) were trained with Bayesian and hyperband parameter optimisation in order to predict the outcomes of each of the nine attempts that lifters make during a competition. The data used was from USAPL competitions from 2014 onwards, as this data is “complete” in the openpowerlifting dataset. The predictions of these models were compared to a “Naïve classifier” which simply predicted the majority class. Both HBGBT and MLP models slightly outperformed the naïve classifier for attempts two (57% and 55% vs 54%) and three (58% and 54% vs 53%) of the Bench Press, and attempt three of the Deadlift (67% and 67% vs 66%). This project serves as an outline for attempt selection prediction and data cleaning of the openpowerlifting dataset, with optimism that more accurate models can be developed over time.

CONTENTS

1	Abstract.....	2
2	Introduction	5
3	Methodology.....	6
4	Data characteristics, cleaning and feature generation.....	7
4.1	Data source.....	7
4.2	Data characteristics and inconsistencies.....	7
4.3	Data cleaning	9
4.3.1	Dataset filtering.....	9
4.3.2	Removing repeated meets/truncating meets.....	9
4.3.3	Disambiguating ambiguous lifters.....	11
4.4	Feature Generation.....	15
4.4.1	Attempt Passes.....	15
4.4.2	Attempt Jumps	15
4.4.3	Attempt success and strength metrics	16
4.4.4	Competition experience metrics	18
4.4.5	Sex	19
4.4.6	Speculative features that cannot be generated.....	20
4.5	Post Feature engineering data cleaning.....	20
4.5.1	Competition filtration.....	20
4.5.2	Time sensitive data splitting	20
5	Exploratory data analysis	21
5.1	Feature correlations.....	21
5.2	Linear mixed models	23
6	Model Creation, tuning and evaluation.....	25
6.1	Data preparation	25
6.2	Neural network with hyperband hyperparameter tuning	25
6.2.1	Hyperband Hyperparameter Tuning	27
6.2.2	Trial and error epoch tuning	29
6.3	Histogram based gradient boosting Classifier	29
6.3.1	Bayesian optimisation model tuning.....	30
6.4	Model Evaluation	31
6.4.1	Results.....	31
6.4.2	Feature importance	32
6.5	Limitations	33

7	Conclusion.....	34
Appendix A	Dataset Filtering script.....	35
Appendix B	Repeat Data handing	36
Appendix C	Lifter Disambiguation	37
Appendix D	Lifter Disambiguation - Web Scraping function	38
Appendix E	Removing non-disambiguated lifters, and non-USAPL lifters from the dataset	39
Appendix F	Feature Generation Scripts	40
a.	Time since last meet	40
b.	Number of past meets	41
c.	Personal best squat, bench, deadlift, total and wilks score.....	42
d.	Squat Pass or fail:.....	43
i.	Missing values imputed	43
ii.	Missing values not imputed	44
Appendix G	Exploratory Data analysis script and results	45
a.	EDA Script.....	45
b.	Correlation Coefficients	46
c.	Linear mixed model output.....	47
Appendix H	Model Scripts	48
a.	Histogram Gradient Boosted Trees	48
b.	Multi-Layer perceptron.....	49
Appendix I	Model Results.....	50
8	Bibliography	51

2 INTRODUCTION

Preface

There is a growing body of research surrounding sporting prediction with machine learning models. The literature mainly surrounds mainstream sports such as basketball, soccer, and golf, which have well maintained datasets available. In some cases, this data can be incredibly granular.

From the 20th century, “gym culture” has been a prevailing trend, and competitive sports associated with lifting weights, such as bodybuilding and powerlifting are growing in popularity. The past decade has seen significant spike in raw powerlifting, as both an individual and a competitive hobby [1]. The ease of training, competing and the growth of social media has led to a large amount of people signing up to and competing in amateur powerlifting competitions, where lifters attempt to lift their maximum Squat, Bench and Deadlift.

Powerlifting Competitions

During a powerlifting competition, lifters compete in three rounds per lift, attempting weights chosen by themselves or their coach. The first attempt of each lift is the lightest weight the lifter can attempt – subsequent attempts must be heavier, or the same weight as previous attempts. A lifter passes an attempt if they manage to successfully lift the weight within the required range of motion, to the standard required, and in time with the judge’s commands. During “full power” meets, the squats, bench and deadlifts are contested. There are some meets that are “push-pull” only (bench and deadlift) or single lift only (most commonly bench press). The focus of this project will be on full power meets.

A lifter’s best squat, bench and deadlift in a meet contributes towards their “total”. Typically in a meet, lifters focus on being the top lifter within their weight, sex and age class. Some lifters in junior or senior weight classes will also have their total count towards the adult “open” weight class if they can compete with those in the open class, where the strongest lifters tend to be.

Openpowerlifting Dataset

The majority of powerlifting competitions are open to all, with a few being restricted to the most strongest lifters. There is great enthusiasm for data within the sport, much so that in 2014, a group of powerlifters developed a dataset containing data for powerlifting meets on openpowerlifting.org. The website can be filtered by lifters, meets, federations, countries, competition types and individuals. Great effort is made to maintain the dataset, however there is scope for the data to be more complete and cleaner.

Project Goals

The goals of this project are to:

- Identify areas for cleaning the openpowerlifting dataset, through manual data exploration.
- Perform data cleaning operations on the openpowerlifting dataset.
- Train and tune machine learning models to accurately predict the likelihood of a lifter passing an attempt.

3 METHODOLOGY

The methodology used during this project is similar to that outlined in the project proposal, with minor changes:

1. The data cleaning process took more time than expected. There were plans to develop a web scraping application to predict attempts in real time from liftingcast.com, and to develop a second application to continuously retrain the models using new data over time, however time constraints disallowed the development of these applications.
2. The scope for extending the model building and prediction workflows to cloud services was not explored due to time constraints.

The remaining methodology remained the same, and consisted of the steps outlined below. The steps were implemented sequentially:

1. Data and domain understanding: Reviewing the information available from openpowerlifting.org and the existing powerlifting/sporting literature.
2. Data preparation: Cleaning the data and producing additional speculative features which may positively impact on the model accuracies.
3. Exploratory data analysis: using descriptive statistics and feature correlations to determine data patterns and to form hypotheses.
4. Model building: Select a group of models and appropriate hyperparameter optimisation methods in order to ensure that the models are correctly tuned.
5. Model evaluation: Compare the results and training times of each model.

The technologies used to execute this project were:

1. Python Packages
 - a. Jupyter notebooks to execute data cleaning, analysis and modelling scripts
 - b. Numpy and pandas to manipulate, filter and clean the dataset
 - c. Statsmodels, matplotlib and numpy for descriptive statistics, modelling and data visualisation
 - d. Scikit-Learn, tensorflow and keras to run models on the data
2. Microsoft Excel to tabulate and format data for reporting
3. Microsoft Word for qualitative reporting

4 DATA CHARACTERISTICS, CLEANING AND FEATURE GENERATION

4.1 DATA SOURCE

The Openpowerlifting.org dataset is a nightly updated, open source CSV dataset containing the results of worldwide powerlifting competitions, from 1966 to present. The meet results are collected from 338 federations, each federation collecting reporting/collecting their own results in a variety of formats (PDF, HTML, JPEG images, databases magazines etc). In addition to this, most federations are indicated as having ‘Incomplete’ data, further compromising the integrity of the overall dataset. Attempt and age data are missing from many rows of the dataset.

4.2 DATA CHARACTERISTICS AND INCONSISTENCIES

Each row of the dataset represents the results of an individual lifter at a single meet; as of the time of writing (19th August 2021), the dataset contains 2513650 entries for 658059 lifters from 39136 meets [2].

The columns of the dataset can be divided into three categories, along with their inconsistencies, are identified in Table 1.

Feature category	Feature Name	Optional?	Data Type	Feature Description	Inconsistencies/Caveats
Lifter Characteristics	Name	No	String	Lifter's name. Lifters who share the same name are disambiguated with a '#' followed by a unique number.	Not all lifters that share the same name have been disambiguated. Additionally, it is not clear if the dataset has accounted for name changes e.g. surname changes after marriage.
	Age	Yes	Float	Age of the lifter at the time of the meet. If the birth year was the only information given, the age is expressed as current year - birth year - 0.5.	Age is not always correctly reported. This can be because the lifter has not yet been disambiguated.
	Sex	No	String	Sex of the lifter (M/F).	None identified
	Division	Yes	String	Division that the lifter has competed in.	Division names vary between federations.
	BodyweightKg	Yes	Float	The bodyweight of the lifter at the time of competition.	None identified
	WeightClassKg	Yes	Float	The weight class that the lifter competed in.	Weight Classes can vary between federations.
	Age Class	Yes	String	The age class interval the lifter is in on the day of the competition. Useful for when there is no other age information available.	Age Classes can vary between federations.
	Birth Year Class	Yes	String	The age interval that the lifter falls in after their birthday during the year in the "Date" column. Useful for when there is no other age information available.	None identified
	Equipment	No	String	The supportive equipment that the lifter performed the lifts in e.g. raw(unequipped), wraps, single-ply.	The 'Wraps' equipment only has bearing on the lifter's squat. Additionally, a lifter can compete with no equipment in a wrapped division.
	Tested	Yes	Boolean String	Whether the lifter entered a drug-tested category.	None identified
	Event	No	String	The competition that the lifter entered e.g. SBD for Squat/bench/deadlift, BD for Bench/Deadlift.	The WABDL federation (and potentially others) report single meets (e.g. BD meets) as two separate meets.
	Country / State	Yes	String	The home country and state of the lifter.	None identified
Lifter result Characteristics	(Squat Bench Deadlift) + (1 2 3) + Kg	Yes	Integer or float	First, second and third results of squat, bench and deadlift. A negative value represents a failed attempt. Highest of the three values contributes towards feature <i>Best3 + (Squat Bench Deadlift) + Kg</i>	Some federations only report the best attempt achieved in the 'Best 3' columns
	(Squat Bench Deadlift) + 4Kg	Yes	Integer or float	Rare feature, used if a lifter attempts to break a single-lift world record as a 4 th lift attempt. Does not count towards feature <i>Best3 + (Squat Bench Deadlift) + Kg</i>	None
	Best3 + (Squat Bench Deadlift) + Kg	Yes	Integer or float	Best squat, bench and deadlift achieved by the lifter, counting towards the feature <i>TotalKg</i>	Some federations report this as a negative value, to represent the heaviest attempt that a lifter failed.
	TotalKg	Yes	Integer or float	Sum of best squat, bench and deadlift achieved by the lifter. Empty if the lifter was disqualified.	None identified
	Place	No	Integer or string	Place of the lifter in the division they competed in. Also indicates guests/no shows/disqualifications	None identified
	Dots / Wilks / Glossbrenner / Goodlift	Yes	Float	Mathematical coefficient scores, used to compare lifters across age and weight classes.	None identified
Meet characteristics	Meet Name	No	String	Name of the powerlifting meet	None identified
	Federation	No	String	Federation of the meet	None identified
	Parent Federation	Yes	String	Parent federation, usually an international federation	None identified
	Meet Country	No	String	Country that the meet was hosted in.	None identified
	Meet State/Town	Yes	String	State/province that the meet was hosted in.	None identified
	Date	No	String	The date of the first day of the competition.	None identified

Table 1: The columns/attributes of the openpowerlifting dataset.

4.3 DATA CLEANING

The end goal of this project was to train a model on a reliable dataset. The original dataset had characteristics which prevented the training of machine learning models, therefore an extensive level of data filtering and cleaning was performed on the dataset.

In order to prepare the dataset for exploratory data analysis, the following tasks were required to be completed:

1. Filtering the dataset to only include lifters who have competed in at least one competition under the USAPL federation, unequipped (raw) and “full power” (*Event* == “SBD”).
2. Removing repeated meets
3. Disambiguating ambiguous lifters (lifters who share the same name, but have not been distinguished from each other).
4. Feature engineering

4.3.1 Dataset filtering

In order to reduce the size of the dataset, excess rows of data were removed prior to the data cleaning. **The machine learning models were to be trained on USAPL Raw lifters with “Complete” data.** As the USAPL meets are marked as “Complete from 2014”, all lifters who did not have at least one raw competition under the USAPL after 2013 were removed. The steps to do this were outlined as below:

1. An Boolean column was added to the dataset, marked “Competed raw in USAPL since 2014”, with the initial assignment of False.
2. The DataFrame was sorted by name, a list of names, sorted alphabetically, was created.
3. For each name in the list of names, the rows of the DataFrame (belonging to the name) were iterated through.
4. If a row entry met the conditions of: date after 2013, equipment == raw, federation == USAPL, then the Boolean column was marked as True for all rows belonging to that lifter.
5. All rows marked as True in the Boolean column were preserved.

This method not only kept the rows containing USAPL raw meet data from 2014 onwards, but it also preserved the rows belonging to the lifters that had not only competed under these conditions, but also competed in other federations, before 2014 and/or using additional equipment. This is because, though these other meets may contain incomplete data, we can generate additional features from them, such as **total number of meets to date** and **personal best squat/bench/deadlift/total/wilks**. The additional features could potentially provide a more accurate picture of the experience of the lifter. The filtering reduced the size of the dataset by approximately 91% (2492593 to 229372 rows).

Optimised Method

Further experimentation found that it was unnecessary to iterate through the rows of the DataFrame to isolate the desired subset of lifters. Instead, the pandas Boolean masking and ‘isin’ functions could be leveraged in order to generate the filtered dataset more efficiently. To iterate through the ~2.5 million rows of data, the cost of this optimised method was 5.59 seconds, compared to 687 seconds for the less efficient method.

4.3.2 Removing repeated meets/truncating meets

In competition, some lifters will register under two divisions (e.g. a junior lifter who is strong enough to compete in the “open” division, may register for junior and open in order to compete under both classes. In this case, their competition would be registered under two rows, with the only difference

between the two rows being the “Division” column (shown in Figure 1). This would interfere with the planned feature “**total number of meets to date**”, as both rows represent the same instance. In addition, some meets (particularly those under the WABDL federation) would separate the lifts into different rows (e.g. a bench and deadlift “BD” meet would be reported as one “B” meet and one “D” meet) or report the full meet as one row, *and* the separate lifts in their own rows (shown in Figure 2).

Place	Fed	Date	Location	Competition	Division	Age	Equip	Class	Weight	Squat	Bench	Deadlift	Total
1	USAPL	2019-10-16	USA-IL	Raw Nationals	MR-Jr	23	Raw	83	82.6	270 -282.5 -282.5	195 202.5 205	300 312.5 -325	787.5
2	USAPL	2019-10-16	USA-IL	Raw Nationals	MR-O	23	Raw	83	82.6	270 -282.5 -282.5	195 202.5 205	300 312.5 -325	787.5

Figure 1: This lifter had two entries in one competition as an MR-Jr (junior) and another as MR-O (open). However in reality, they only competed once.

Place	Fed	Date	Location	Competition	Division	Age	Equip	Class	Weight	Squat	Bench	Deadlift	Total
1	WABDL	2020-09-08	USA-CA	California State Bench Press and Deadlift Championships	Masters 40-46		Raw	110	110		255	260	255
1	WABDL	2020-09-08	USA-CA	California State Bench Press and Deadlift Championships	Masters 40-46		Raw	110	110		170		170
1	WABDL	2020-09-08	USA-CA	California State Bench Press and Deadlift	Masters 40-46		Raw	110	110		170	255	425

Figure 2: This lifter had three entries in one competition, (bench only, deadlift only, bench and deadlift), however in reality, they only competed once (bench and deadlift).

In the case of the identical meets, the duplicates were removed. In the case of the separated lifts, the results from each meet would be combined into one row.

In order to identify and remove duplicate meets, two columns were created: one for cases where the meets were repeated, and another for the cases where there were two meets in one day from the same lifter. The following steps were taken, similar to those outlined in 4.3.1:

1. Iterate through each unique lifter name in the dataset
2. For each lifter, seek the competitions that shared the same date
 - a. If a competition is found where the date is the same, check that the competition name is the same and that the bodyweight is the same for each row
 - i. Then if the equipment is the same for each meet, do the following:
 1. If the rows have the same event type (e.g. two SBD meets), label any row as “repeated” and remove that row
 2. Else if the rows have different event types but one contains more lifts, and they share lifts (e.g. BD and D or SBD and BD), then label the row with less lifts as ‘repeated’ and remove that row
 3. Else if the rows do not share any event types, combine the results from the separate events as one single row and recalculate the “Wilks” score .
 - ii. If the equipment is not the same for each meet, mark both rows as “two meets in one day”. Do not delete any of the rows, as it is likely that the rather a lifter has chosen to compete in two separate competitions (due to differences in equipment) on the same day.

- b. If the dates are the same and the bodyweight is not the same or the competition names are not the same, mark both rows as “two meets in one day”, as it is likely that they are two different lifters who share the same name and competed on the same date, or it is one lifter who competed in two separate competitions (and thus had to be weighed twice) on the same day.

The Jupyter notebook for the repeat data handing can be found in Appendix B.

4.3.3 Disambiguating ambiguous lifters

As mentioned in Table 1, some lifters who share the same name have not been disambiguated from each other. The original dataset disambiguates shared name lifters by adding a #x to the end of the name, where x is a number unique to each individual. A disambiguation function was developed by using the age progression of each lifter, which will be detailed in the following subsections.

4.3.3.1 *Date of birth interval creation*

Using the age information available, a *Date of birth interval* column was created. The value for each row was the range of possible birthdates that the lifter fell under, using the *Age* or *BirthYearClass* or *AgeClass* values. A pandas interval data type was used to store the interval value.

4.3.3.1.1 Age

If the age can be expressed as an integer (i.e. 30.0 can be expressed as 30, 30.5 cannot be expressed as an integer), this means that the earliest birthdate the lifter may have is meet date plus 1 day minus given age (lifter’s birthday is the day after the meet) and the latest birthday the lifter may have is meet date minus given age (lifter’s birthday is the day of the meet). The aforementioned date range was created using the dateutil package (to subtract years while taking account of leap years) [3] and pandas time interval datatype.

If the age is a .5 fraction, this means that the lifter turns the older age in that year (e.g. 30.5 means that the lifter turns 31 that year). This convention is used because some federations only provide the lifter’s birthyear. In this case, the oldest age that the lifter can be is first day of the year minus given age plus 0.5 years (lifter’s birthday is the first day of the year). The youngest age that the lifter can be is the last day of the year minus given age plus 0.5 years (lifter’s birthday is the last day of the year). For the “.5” ages, the date range was created with the above rules.

4.3.3.1.2 BirthYearClass and Age Class

In the case that there was a missing value in the *Age* column, then the *BirthYearClass* column was used to infer the date of birth intervals. If both columns had missing values, then the *AgeClass* column was used to infer the date of birth intervals. As the *BirthYearClass* and *AgeClass* columns are age ranges, the intervals were generated by using the oldest age the lifter could be and the youngest age that the lifter could be (e.g. if *BirthYearClass* was 40-49, the lifter could have turned 40 on the date of the meet, or could be turning 50 the day after the meet).

4.3.3.1.3 Missing Age, BirthYearClass and Age Class

In some rows, there were no values for *Age*, *BirthYearClass* and *AgeClass* columns. In many cases age ranges could be added to the *BirthYearClass* and *AgeClass* by inferring the range through the *Division* column. A function “`modify_unknown_division_ages()`” was written to allow the user to manually iterate through each value of *Division* which contained at least one row with no age-related data. An age range could be inferred through either looking at the actual division name (e.g. most divisions containing the substring “jr” indicate junior lifters at or between the ages of 20 and 23) or by viewing the value counts (named “head”) of the division. Figure 3 shows the output of the

function after choosing to view the value counts of division “M-C-M1”. In this case, it is clear that the lifters belong to the “Masters 1” division which is used through many federations; Masters 1 lifters are in their 40s. The user would be able to input the age intervals, and the *BirthYearClass* and *AgeClass* values corresponding to the division name would be replaced by the age range, and more importantly, the *Date of Birth Interval* would be generated relative to the meet date.

```
The division to be checked is M-C-M1. Press 1 to enter an interval, 2 if it is an open or ageless division,      3 if you
want to skip it for later, 4 if you want to see the head of the division name, and 5 to quit: 4
41.0    2
40.0    2
42.0    1
Name: Age, dtype: int64
40-44    5
Name: AgeClass, dtype: int64
40-49    29
Name: BirthYearClass, dtype: int64
```

Figure 3: Extract of “*modify_unknown_division_ages()*” function output.

In the case where an division was “open” or “ageless” (anyone of any age could enter), no *AgeClass*, *BirthYearClass*, or *DateofBirthInterval* would be assigned to the row.

While not fool proof (some lifters who share the same name may also be the same age, and some names should be joined as women who lose their maiden name may mistakenly be under two different IDs) this method was able to identify 6203 names which potentially required disambiguation.

4.3.3.2 *Disambiguation*

After the *DateOfBirthInterval* values were added to the dataset, the dataset was grouped by *Name*, and all possible pairwise comparisons were made between the *DateOfBirthInterval* values for each row. The **pandas.interval.overlap** function was used to make these comparisons; if at least one pairwise comparison produced a False value (intervals do not overlap), this indicated that either the name required disambiguation, or the wrong date of birth had been input for the lifter. An array was produced containing the set of names that potentially required disambiguating. The set of names was fed into the function which will be described below.

4.3.3.2.1 Disambiguation function

The disambiguation function would iterate through the set of lifter names. For each name, a pandas DataFrame would be displayed to the user, showing all of the rows that belonged to the lifter. The user could then disambiguate the lifter by entering indices that corresponded to rows relating to each other, while not relating to the other rows. The user could also choose to save the lifter for later. There were multiple rules for distinguishing lifters, mainly based on domain knowledge. Some of these rules are outlined below:

1. Age

A coherent age progression could be found within a subset of the rows, and this age progression conflicts with those found in the remaining rows e.g. lifter is 16 during a March 2020 competition, 19 during a July 2020 competition, and 17 during a September 2020 competition. It is likely that rows 2 belong to a different lifter than rows 1 and 3. Exceptions were made on a case-by-case basis, such as when very similar bodyweights, squat/bench/deadlift scores and locations were observed regardless of age difference. If this was to occur for the lifter described above, it is likely that the lifter’s age was input incorrectly in either row one & three or row two. The lifter’s meet results would then be manually looked up online to identify and modify the incorrect ages.

2. BodyweightKg

Unrealistic changes in *BodyweightKg* in short periods of time e.g. lifter is 100kg in a march 2020 competition, then 60kg in a June 2020 competition, and then 102kg in a September 2020 competition – rows 1 and 3 may belong to the same lifter, and row 2 may be a different lifter.

3. Lifter not yet disambiguated (USAPL federation only)

Lifter has been disambiguated on the USAPL lifter database [4], but not yet disambiguated on the openpowerlifting dataset. The USAPL lifter database was used to identify the separate individuals, and the meets that they did. An attempt to expand on this logic is with web scraping is detailed in 4.3.3.2.2.

4. Squat/Bench/Deadlift discrepancy

Unrealistic differences in Squat, Bench and/or deadlift results meet-to-meet (e.g. most results are within a given range, however one meet sees an anomaly-like jump or drop in the lifts when compared to the others). Exceptions were made if the equipment varied meet-to-meet – one would expect to see much larger squats or bench presses when competing in “Multi-ply”/“single-ply” when compared to “raw” competitions.

	Name	Federation	Event	Date	MeetCountry	Event	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearCla
14	Alexandra García	THSWPA	SBD	SBD	2015-01-31	USA	SBD	SBD	Girls	Single-ply	51.35	NaN
15	Alexandra García	THSWPA	SBD	SBD	2015-02-05	USA	SBD	SBD	Girls	Single-ply	60.15	NaN
16	Alexandra García	THSWPA	SBD	SBD	2015-02-21	USA	SBD	SBD	Girls	Single-ply	57.70	NaN
17	Alexandra García	THSWPA	SBD	SBD	2015-02-21	USA	SBD	SBD	Girls	Single-ply	51.44	NaN
18	Alexandra García	THSWPA	SBD	SBD	2015-03-07	USA	SBD	SBD	Div 2	Single-ply	51.17	NaN
19	Alexandra García	THSWPA	SBD	SBD	2015-03-19	USA	SBD	SBD	Girls	Single-ply	51.12	NaN
21	Alexandra García	USPA	SBD	SBD	2017-10-28	USA	SBD	SBD	Juniors 16-17	Raw	56.80	16.0
22	Alexandra García	USPA	SBD	SBD	2018-02-24	USA	SBD	SBD	Open	Raw	55.10	16.0
23	Alexandra García	USPA	SBD	SBD	2018-06-23	USA	SBD	SBD	Juniors 16-17	Raw	58.40	17.0
24	Alexandra García	USPA	SBD	SBD	2018-12-01	USA	SBD	SBD	Juniors 16-17	Raw	56.00	17.0
25	Alexandra García	USPA	SBD	SBD	2019-04-20	USA	SBD	SBD	Juniors 16-17	Raw	61.55	17.0

	Name	Federation	Event	Date	MeetCountry	Event	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearCla
21	Alexandra García	USPA	SBD	2017-10-28	USA	SBD	Juniors 16-17	Raw	56.80	16.0	16-17	14-
22	Alexandra García	USPA	SBD	2018-02-24	USA	SBD	Open	Raw	55.10	16.0	16-17	14-
23	Alexandra García	USPA	SBD	2018-06-23	USA	SBD	Juniors 16-17	Raw	58.40	17.0	16-17	14-
24	Alexandra García	USPA	SBD	2018-12-01	USA	SBD	Juniors 16-17	Raw	56.00	17.0	16-17	14-
25	Alexandra García	USPA	SBD	2019-04-20	USA	SBD	Juniors 16-17	Raw	61.55	17.0	16-17	14-

Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.
Type 'Q' to quit, or 'L' to save the lifter for later: y
Enter the row indicies for this disambiguation, separated by a space: 21 22 23 24 25
New lifter is as shown:

	Name	Federation	Event	Date	MeetCountry	Event	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearCla
21	Alexandra García	USPA	SBD	2017-10-28	USA	SBD	Juniors 16-17	Raw	56.80	16.0	16-17	14-
22	Alexandra García	USPA	SBD	2018-02-24	USA	SBD	Open	Raw	55.10	16.0	16-17	14-
23	Alexandra García	USPA	SBD	2018-06-23	USA	SBD	Juniors 16-17	Raw	58.40	17.0	16-17	14-
24	Alexandra García	USPA	SBD	2018-12-01	USA	SBD	Juniors 16-17	Raw	56.00	17.0	16-17	14-
25	Alexandra García	USPA	SBD	2019-04-20	USA	SBD	Juniors 16-17	Raw	61.55	17.0	16-17	14-

Are you happy with this? Enter Y if yes, or N if no: y

Figure 4: Example of lifter disambiguation.

The lifter disambiguation function can be found in Appendix C.

4.3.3.2.2 Web scraping integration

As there were cases where lifters would be disambiguated on the USAPL lifting dataset, but not the openpowerlifting dataset, it was deemed appropriate to modify the lifter disambiguation function to web scrape the USAPL lifting database to find the competitions that correspond to a given lifter, and find the matching rows in the openpowerlifting dataset. The web scraping was performed with the Python Selenium Package. The steps for this method are outlined below, and the script can be found in Appendix D:

1. A dictionary was constructed from scraping each page of the <https://usapl.liftingdatabase.com/lifters> index. The keys corresponded to the page number and the values corresponded to the list of lifters on that page.
 2. A second scraping function would pass a given lifter name to a function which would create a browser navigating to the lifter's individual page, and construct a DataFrame with their meet results.
 3. A modified version of the disambiguation function from 4.3.3.2.1 would compare the scraped DataFrame with the corresponding DataFrame from the openpowerlifting dataset, and select the rows that matched. If so, the user would then be prompted to add any additional rows to that DataFrame. If no rows matched, the function would behave as usual.

Figure 5 shows an example of this function operating on lifter “Brittany Turner”.

Figure 5: Modified lifter disambiguation function outputs. The first DataFrame is the intersection of the USAPL lifting database DataFrame with the corresponding Openpowerlifting DataFrame. The second DataFrame is the difference between the Openpowerlifting DataFrame and the USAPL database DataFrame. The user is prompted to select any more rows that can be added to the first DataFrame, representing a disambiguated lifter.

This function was discarded for the following reasons:

1. If the lifter was not disambiguated from the USAPL lifting dataset, the function had no way to determine this and would function as if the lifter was disambiguated, creating additional complexity for the user. Additional functionality could have been added to allow the user to reject the output USAPL DataFrame, however this increases operation time for the user due to the added element of complexity.

- The web scraping operations added seconds for every new lifter, as the pages from <https://usapl.liftingdatabase.com/> had to be loaded. This would unnecessarily increase running time, particularly for lifter names with a low number of rows (<5) that were easy to disambiguate regardless of web scraping.

4.3.3.2.3 Number of disambiguation/further filtering

Due to time constraints, 3000 out of the 6203 lifters were disambiguated. The remaining lifters that required disambiguation were removed from the dataset, and the remaining dataset was filtered again using the method described in 4.3.1. The filtering was repeated because after the disambiguation, there were more names that had not competed in the USAPL raw since 2013. For instance, “lifter A” may have had 5 single-ply competitions before 2000 and 5 raw competitions in the USAPL after 2014, however the disambiguation process determined that “lifter A” represented two individuals who were subsequently renamed to “lifter A #1” (competing in the meets before 2000) and “lifter A #2” (competing in the meets after 2013). This means that the filtering method would have only been able to remove “lifter A #1” after the disambiguation, as before the disambiguation, “lifter A” was presumed to be one individual who had competed in at least one USAPL raw meet after 2013. The script for the removal of these lifters can be found in Appendix E.

4.4 FEATURE GENERATION

The existing literature on powerlifting performance was consulted to provide a grounding for additional features which could serve to be useful in the analysis of the dataset. The following sections will underline the procedures used to generate meaningful features. The scripts for feature generation can be found in Appendix F.

4.4.1 Attempt Passes

To enable the data analysis, binary indicator features were created for each attempt. The original dataset represented “failed” attempts as the negation of the weight attempted (e.g. *Squat1Kg* = -100kg for lifter A failing to lift 100kg for their first attempt squat). For each attempt, a binary array was created displaying “0” for values at or below 0, and “1” for values above 0. The accompanied binary array was then given a name representing the attempt (i.e. *Squat1 Pass*, *Squat2 Pass*, *Deadlift3 Pass*). This representation is appropriate for the main goal of this study, which is to accurately predict failed and successful attempts.

```
openpowerlifting_USAPL_lifters.insert(11,"Squat1 Pass", openpowerlifting_USAPL_lifters["Squat1Kg"] > 0)
openpowerlifting_USAPL_lifters["Squat1 Pass"] = openpowerlifting_USAPL_lifters["Squat1 Pass"].astype(int)

openpowerlifting_USAPL_lifters.insert(13,"Squat2 Pass", openpowerlifting_USAPL_lifters["Squat2Kg"] > 0)
openpowerlifting_USAPL_lifters["Squat2 Pass"] = openpowerlifting_USAPL_lifters["Squat2 Pass"].astype(int)

openpowerlifting_USAPL_lifters.insert(15,"Squat3 Pass", openpowerlifting_USAPL_lifters["Squat3Kg"] > 0)
openpowerlifting_USAPL_lifters["Squat3 Pass"] = openpowerlifting_USAPL_lifters["Squat3 Pass"].astype(int)
```

Figure 6: Lift Pass binary variable generation (also performed for bench and deadlift)

4.4.2 Attempt Jumps

The jumps between lift attempts may be detrimental to probability of a lifter making an attempt. Howells et al [5] showed that those with higher absolute weight jumps were more likely to **win** powerlifting competitions, however this is to be expected because winners tend to have higher ceilings of strength. The impact of **relative weight jump** on **individual attempt success** has not yet been considered.

The process to create the percentage attempt jump was simple: The difference between attempts was calculated and divided by the preceding attempt, for each lift. The total jump (from attempt 1 to attempt 3) was also calculated.

$$\begin{aligned} \text{Squat2 Jump} &= \frac{\text{Squat2Kg} - \text{Squat1Kg}}{\text{Squat1Kg}} \\ \text{Squat3 Jump} &= \frac{\text{Squat3Kg} - \text{Squat2Kg}}{\text{Squat2Kg}} \\ \text{Total Squat Jump} &= \frac{\text{Squat3Kg} - \text{Squat1Kg}}{\text{Squat1Kg}} \end{aligned}$$

Equation 1: Attempt Jump examples for “Squat”

4.4.3 Attempt success and strength metrics

4.4.3.1 Past competition attempt success

Various sporting literature has shown the impacts of consecutive successes/failures on the psychology of athletes, and how success and failure can affect performance and the perception of “control” during competitions [6].

To quantify the success in past competitions, “success rates” were generated for each attempt. These “success rates” were the probability of a lifter passing their attempt, based on their past three meets. The success rates across each lift were also averaged to create another column, “overall success rate”. If the lifter had completed less than three meets in the past, then this was calculated based on every meet done to date. If it was the lifter’s first meet, the success rates were inferred with an iterative regressor (Bayesian ridge regression in this case). The Scikit-learn iterative imputer function was used to iteratively fill the missing success rate values. It was necessary to fill these missing values to allow machine modelling algorithms to work properly on the data.

To create the success rate columns, the subset of the data containing attempt-level data was grouped by lifter name (with pandas group by) and rolling means¹ (with window sizes of three) of the *Lift’X’Pass* columns were generated. For each lifter, the meets with no attempt data (best attempts only recorded) were filled with the preceding success rates. Additionally, the generated columns were shifted down by a row because the new values in each column were calculated from the same row prior to the shifting (e.g. in row 4, the average squat1 success rate is from rows 2,3 and 4, so there are hints given to how the lifter will perform if we were to use the non-shifted column in a machine learning model).

¹ The rows of the dataset were sorted by ascending date prior to any group by aggregations.

```

meets_where_attempts_recorded = pd.Series([False]*openpowerlifting_USAPL_lifters.shape[0])
### below for loop are indicies of lifters where all attempts are recorded
for name in ["Squat1", "Squat2", "Squat3",
             "Bench1", "Bench2", "Bench3",
             "Deadlift1", "Deadlift2", "Deadlift3"]:
    meets_where_attempts_recorded = meets_where_attempts_recorded | pd.notna(openpowerlifting_USAPL_lifters[f"{name}Kg"])

for name in ["Squat1", "Squat2", "Squat3",
             "Bench1", "Bench2", "Bench3",
             "Deadlift1", "Deadlift2", "Deadlift3"]:
    openpowerlifting_USAPL_lifters[f"{name} Past 3 meets success rate"] = np.nan
    openpowerlifting_USAPL_lifters.loc[
        (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
        & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
        & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
        & (openpowerlifting_USAPL_lifters["Date"] > pd.to_datetime("01/01/2014")) \
        & (pd.notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
        & (meets_where_attempts_recorded) \
        , [f"{name} Past 3 meets success rate"]]
= openpowerlifting_USAPL_lifters.loc[(openpowerlifting_USAPL_lifters["Event"] == "SBD") \
        & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
        & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
        & (openpowerlifting_USAPL_lifters["Date"] > pd.to_datetime("01/01/2014")) \
        & (pd.notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
        & (meets_where_attempts_recorded), :].groupby(
    "Name")[f"{name} Pass"].rolling(3, min_periods = 1).mean().reset_index(0)[f"{name} Pass"]

openpowerlifting_USAPL_lifters.loc[:, f"{name} Past 3 meets success rate"] = openpowerlifting_USAPL_lifters.groupby(
    "Name")[f"{name} Past 3 meets success rate"].shift()
openpowerlifting_USAPL_lifters.loc[:, f"{name} Past 3 meets success rate"] = openpowerlifting_USAPL_lifters.groupby(
    "Name")[f"{name} Past 3 meets success rate"].ffill()

```

Figure 7: “Attempt Success rate” feature generation

4.4.3.2 Past competition best squat/bench/deadlift/total/wilks

Vinh et al. [7] used features such as best squat, best deadlift and best bench to predict the total weight lifted by a powerlifter, therefore we will see if these features also work in combination with the weights lifted in a current competition to model the effects of strength potential, and relative fatigue i.e. if a lifter lifts too close to their personal best on their first attempt squat, it may impact the rest of their squat performance. The lifter’s personal best total and wilks scores were also added as features, as they are indicators of how advanced the lifter is with respect to those of a similar weight.

To calculate these features, the dataset was grouped by lifter name (with pandas group by function) and a cumulative max value was generated per for each group, using “*Best3(Squat/Bench/Deadlift)Kg*”, the total, and *Wilks* columns. Care was taken to ensure that squat/bench/deadlift results from assistive equipment meets (i.e. single-ply, multi-ply, wraps) were distinguished from raw squat/bench/deadlifts.

During the lifter’s first meet, they will have no “personal bests” yet. In this case, the personal best columns were filled with the value zero. In some cases where a lifter failed all of their attempts, the “*Best3(Squat/Bench/Deadlift)Kg*” columns had a negative value indicating the highest failed attempt that the lifter took. To prevent negative values from appearing in the new columns, these were given a value of zero.

```

openpowerlifting_USAPL_lifters["Best3SquatKg"] = openpowerlifting_USAPL_lifters[
    "Best3SquatKg"].where(openpowerlifting_USAPL_lifters["Best3SquatKg"] > 0, 0)
openpowerlifting_USAPL_lifters["Best3BenchKg"] = openpowerlifting_USAPL_lifters[
    "Best3BenchKg"].where(openpowerlifting_USAPL_lifters["Best3BenchKg"] > 0, 0)
openpowerlifting_USAPL_lifters["Best3DeadliftKg"] = openpowerlifting_USAPL_lifters[
    "Best3DeadliftKg"].where(openpowerlifting_USAPL_lifters["Best3DeadliftKg"] > 0, 0)

# Creating new columns
openpowerlifting_USAPL_lifters["Best Raw Squat to date"] = openpowerlifting_USAPL_lifters[
    openpowerlifting_USAPL_lifters["Equipment"] == "Raw"].groupby("Name")["Best3SquatKg"].cummax()

openpowerlifting_USAPL_lifters["Best Wrapped Squat to date"] = openpowerlifting_USAPL_lifters[
    openpowerlifting_USAPL_lifters["Equipment"] == "Wraps"].groupby("Name")["Best3SquatKg"].cummax()

openpowerlifting_USAPL_lifters["Best Raw Bench to date"] = openpowerlifting_USAPL_lifters[
    (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") | (
        openpowerlifting_USAPL_lifters["Equipment"] == "Wraps")].groupby("Name")["Best3BenchKg"].cummax()

openpowerlifting_USAPL_lifters["Best Raw Deadlift to date"] = openpowerlifting_USAPL_lifters[((
    openpowerlifting_USAPL_lifters["Equipment"] == "Raw") | (
        openpowerlifting_USAPL_lifters["Equipment"] == "Wraps"))].groupby("Name")["Best3DeadliftKg"].cummax()

openpowerlifting_USAPL_lifters["Best Raw Total to date"] = openpowerlifting_USAPL_lifters[
    openpowerlifting_USAPL_lifters["Equipment"] == "Raw"].groupby("Name")["TotalKg"].cummax()

openpowerlifting_USAPL_lifters["Best Raw Wilks to date"] = openpowerlifting_USAPL_lifters[
    openpowerlifting_USAPL_lifters["Equipment"] == "Raw"].groupby("Name")["Wilks"].cummax()

for column in openpowerlifting_USAPL_lifters.columns[-1:-24:-1]:
    openpowerlifting_USAPL_lifters[column] = openpowerlifting_USAPL_lifters.groupby("Name")[column].shift()
    openpowerlifting_USAPL_lifters[column] = openpowerlifting_USAPL_lifters.groupby("Name")[column].ffill()
    # due to cummax function leaving forward values empty when selecting a subset of data

```

Figure 8: “Personal best” feature generation

4.4.4 Competition experience metrics

Powerlifters tend to reach peak performance in the middle of their careers between the ages of 20 and 35 [7], and there is an optimal competition frequency for which the lifter is likely to improve on their personal bests [8]. In addition, we hypothesise that lifters who compete more frequently, and have done more lifetime competitions may understand their limits better and choose more sensible attempt weights, increasing their likelihood of success. Three features were created (*number of meets in past 12 months*, *number of past meets*, *time since last meet*) to model these factors.

4.4.4.1 Meet frequency

To produce the *Number of meets in the past 12 months* feature, the dataset was grouped by name, and a calculation was performed using the *date* columns of the grouped DataFrame, counting the length of the DataFrame produced by indexing the rows that were within a year behind the target row.

```

openpowerlifting_USAPL_lifters["Number of meets in past 12 months"] = openpowerlifting_USAPL_lifters.groupby(
    ["Name"])[[ "Date", "Name"]].apply(
    lambda x: pd.DataFrame({
        x[(x["Date"] < x.at[i, "Date"]) &
        (x["Date"] > x.at[i, "Date"] - relativedelta(years = 1))].shape[0] for i in x.index}, x.index))

```

Figure 9: “Number of meets in past 12 months” generation

4.4.4.2 Meet spacing

Producing the *time since last meet* feature was relatively simple; the dataset was grouped by name and the *diff()* function was used to calculate the difference between dates in adjacent rows.

For a lifter’s first meet, they have no “time since last meet”. In order to produce an appropriate input for some machine learning algorithms, this nan value had to be transformed to a numerical value; this value was the date of the meet minus the lifter’s earliest possible date of birth.

```

openpowerlifting_USAPL_lifters["Time Since Last Meet"] = np.nan
openpowerlifting_USAPL_lifters["Time Since Last Meet"] = openpowerlifting_USAPL_lifters.groupby("Name")["Date"].diff()
for i in openpowerlifting_USAPL_lifters.index:
    if type(openpowerlifting_USAPL_lifters.at[i,"Time Since Last Meet"]) == pd._libs.tslibs.nattype.NaTType:
        try:
            openpowerlifting_USAPL_lifters.at[
                i, "Time Since Last Meet"] = openpowerlifting_USAPL_lifters["Date"][i]\
                - openpowerlifting_USAPL_lifters["DateOfBirthInterval"][i].left
        except:
            for j in openpowerlifting_USAPL_lifters[
                openpowerlifting_USAPL_lifters["Name"] == openpowerlifting_USAPL_lifters.at[i,"Name"]].index:
                if (type(openpowerlifting_USAPL_lifters.at[j,"DateOfBirthInterval"])\
                    == pd.Interval) and (type(openpowerlifting_USAPL_lifters.at[j,"Age"]) == np.float64):
                    openpowerlifting_USAPL_lifters.at[i,"Time Since Last Meet"]\
                    = openpowerlifting_USAPL_lifters["Date"][j] - openpowerlifting_USAPL_lifters["DateOfBirthInterval"][j].left
                    break
            continue
openpowerlifting_USAPL_lifters["Time Since Last Meet"] = openpowerlifting_USAPL_lifters["Time Since Last Meet"].dt.days

```

Figure 10: “Time since last meet” feature generation

4.4.4.3 Number of past meets

Calculating the number of past meets was a relatively simple process: the dataset was grouped by name and a cumulative count was performed on each row.

There is a significant number of instances where a lifer will compete in two competitions on one day; we hypothesise that competing in two meets in one day may not have as much of a significant impact on expertise as a lifter competing two times three months apart, for instance. Therefore, another column was created *Number of past meets same day exclusive*, where these *same day meets* only count as one meet. To produce this feature, a Boolean indicator column was created, indicating rows where the *time since last meet* was not equal to zero. The dataset was once again grouped by name, and the cumulative sum of each group with respect to the Boolean column is the number of days in which the lifter competed (feature name *Number of Past Meets same day exclusive*).

```

openpowerlifting_USAPL_lifters["Number of Past Meets"] = np.nan
openpowerlifting_USAPL_lifters["Number of Past Meets"] = openpowerlifting_USAPL_lifters.groupby("Name")["Date"].cumcount()

openpowerlifting_USAPL_lifters["Number of Past Meets_Mod_Bool"] = (
    openpowerlifting_USAPL_lifters["Time Since Last Meet"] != pd.to_timedelta(0, unit='days'))

openpowerlifting_USAPL_lifters["Number of Past Meets_Mod_Bool"] = openpowerlifting_USAPL_lifters.groupby(
    ["Name"])["Number of Past Meets_Mod_Bool"].cumsum() - 1

openpowerlifting_USAPL_lifters.rename(
    columns = {"Number of Past Meets_Mod_Bool": "Number of Past Meets Same Day Exclusive"}, inplace = True)

```

Figure 11: “Number of Past Meets” and “Number of Past Meets same day exclusive” feature generation

4.4.5 Sex

The existing *Sex* feature was encoded as a string, with *M* representing males, *F* representing females, and one instance of “*Mx*” representing a trans-male. The sex feature was one-hot encoded into binary features *Male* and *Female*. The value of “1” means that the lifter belongs to the given sex, and the value “0” indicates that they do not. It was necessary to encode the sex feature this way in order for machine learning algorithms to work.

```

sex_dummies = pd.get_dummies(openpowerlifting_USAPL_lifters, columns = ["Sex"]).filter(["Sex_M", "Sex_F"], axis=1)
openpowerlifting_USAPL_lifters.insert(1, "Male", np.nan)
openpowerlifting_USAPL_lifters.insert(2, "Female", np.nan)
openpowerlifting_USAPL_lifters.drop(columns = "Sex", inplace = True)
openpowerlifting_USAPL_lifters["Male"] = sex_dummies["Sex_M"]
openpowerlifting_USAPL_lifters["Female"] = sex_dummies["Sex_F"]

```

Figure 12: “Male” and “Female” feature generation.

4.4.6 Speculative features that cannot be generated

Mahajan et al. [9] used physiological measures to predict weightlifting performance. This data was collected during training/prior to competition and many of the metrics were predictive of the strength of lifters. In addition, Jidovtseff et al. [10] predicted the “one rep max” load using bar velocity during lifts – weights closer to a lifter’s max showed consistent decreases in velocity. The technology is available to track bar velocity during competitions, and this may serve as a stronger indicator than the features collected and generated from the openpowerlifting dataset. We do not have these features, however they could be included in a future model pertaining to powerlifting, should the data become available.

4.5 POST FEATURE ENGINEERING DATA CLEANING

4.5.1 Competition filtration

To prepare the dataset for EDA and modelling, the dataset was further filtered to only include USAPL full-power raw meets from 2014 onwards. Though the **lifters** who had competed in these meets had already been isolated, many of them had competed in meets outside of these boundaries. The additional filtering was done **after** the exploratory data analysis and feature engineering because, though the analysis and engineering were done on some incomplete data (e.g. missing attempts, ages), there was still valuable information in the data that was used for the feature engineering, particularly for the features generated in 4.4.4: Attempt success and strength metrics.

4.5.2 Time sensitive data splitting

The nature of a powerlifting competition is sequenced, a lifter completes three squat attempts, then three bench attempts, then three deadlift attempts. Due to this, nine separate data sets needed to be created (one for each attempt) in order to avoid feeding a model future attempt outcomes (e.g. when predicting whether a lifter will pass their first squat, we should not know the outcome of their subsequent lifts). When predicting a given attempt, information about the prior attempts should be available for machines to make better predictions. This means that the feature space is larger for later lifts. Figure 13 displays the feature space for each attempt to be predicted.

```
squat1_features= ['Sex', 'WeightClassKg', 'Age', 'BodyweightKg', 'Time Since Last Meet',  
'Number of Past Meets', 'Number of Past Meets Same Day Exclusive', 'Number of meets in past 12 months',  
'Best Raw Squat to date', 'Best Wrapped Squat to date', 'Best Raw Total to date', 'Best Raw Wilks to date',  
'Squat1 Past 3 meets success rate', 'Squat2 Past 3 meets success rate', 'Squat3 Past 3 meets success rate',  
'Bench1 Past 3 meets success rate', 'Bench2 Past 3 meets success rate', 'Bench3 Past 3 meets success rate',  
'Deadlift1 Past 3 meets success rate', 'Deadlift2 Past 3 meets success rate', 'Deadlift3 Past 3 meets success rate',  
'Best Raw Bench to date', 'Best Raw Deadlift to date', 'Overall Attempt success Rate over past three meets',  
'Squat1Kg']  
  
squat2_features= squat1_features + ['Squat1 Pass', 'Squat2Kg', 'Squat2 Jump',]  
  
squat3_features= squat2_features + ['Squat2 Pass', 'Squat3Kg', 'Squat3 Jump', 'Total Squat Jump',]  
  
bench1_features= squat3_features + ['Squat3 Pass', 'Bench1Kg',]  
  
bench2_features= bench1_features + ['Bench1 Pass', 'Bench2Kg', 'Bench2 Jump',]  
  
bench3_features= bench2_features + ['Bench2 Pass', 'Bench3Kg', 'Bench3 Jump', 'Total Bench Jump',]  
  
deadlift1_features= bench3_features + ['Bench3 Pass', 'Deadlift1Kg',]  
  
deadlift2_features= deadlift1_features + ['Deadlift1 Pass', 'Deadlift2Kg', 'Deadlift2 Jump',]  
  
deadlift3_features= deadlift2_features + ['Deadlift2 Pass', 'Deadlift3Kg', 'Deadlift3 Jump', 'Total Deadlift Jump',]
```

Figure 13: The feature space for each attempt to be predicted.

5 EXPLORATORY DATA ANALYSIS

5.1 FEATURE CORRELATIONS

Prior to training the models, it was important to analyse and visualise the training data in order to determine which features were predictive of a lifter making a successful attempt, the features that shared high collinearity, and the between-lifter variation. To do this, the distributions of features were analysed, linear correlation tables were created, and average success rates/feature graphs were created. We will first analyse the Pearson's correlation values between the "attempt pass" variables and the main independent variables, and the correlations between the success rates of attempts following pass/failure of previous attempts. The correlation coefficient is a good measure of the linear correlation between two variables, however it is not a test of significance, so it is not guaranteed that features with high correlations are predictive of each other. In addition, the correlation coefficient is poor at assessing nonlinear relationships between features. Full plots and correlation tables can be found in 0.

Table 2 shows the top 5 linear correlations between each attempt success indicator and the independent variables. At the start of the competition, the lifter's personal bests and advancement correlate strongly with the probability of success of the first squat, indicating that more advanced lifters may tend to ensure that they pass their squats. As the meet progresses, the "in meet" features become the most correlative. Interestingly, larger relative weight "jumps" tend to have positive influences on the success of attempts. However, this finding may be due to a lifter choosing more conservative early attempts to minimise fatigue, allowing them to make higher percentage jumps to later attempts while remaining successful. Additionally, higher deadlift attempts correlate negatively with pass rates, which may be due to lifters attempting to push for a higher total at the end of a meet. The attempt jump findings are consistent with those found by Howells et al. [11].

Squat1 Pass			Squat2 Pass			Squat3 Pass		
Feature	Corr	Influence	Feature	Corr	Influence	Feature	Corr	Influence
Best Raw Wilks to date	0.0927	positive	Overall Attempt Success Rate over past three meets	0.0757	positive	Overall Attempt Success Rate over past three meets	0.0893	positive
Best Raw Squat to date	0.0890	positive	Squat3 Past 3 meets success rate	0.0493	positive	Squat3 Past 3 meets success rate	0.0706	positive
Best Raw Total to date	0.0876	positive	Age	0.0452	negative	Squat3 Jump	0.0610	positive
Best Raw Deadlift to date	0.0855	positive	Squat2 Past 3 meets success rate	0.0442	positive	Deadlift3 Past 3 meets success rate	0.0541	positive
Best Raw Bench to date	0.0802	positive	Male	0.0378	positive	BodyweightKg	0.0461	positive
Bench1 Pass			Bench2 Pass			Bench3 Pass		
Feature	Corr	Influence	Feature	Corr	Influence	Feature	Corr	Influence
Best Raw Wilks to date	0.0938	positive	Overall Attempt Success Rate over past three meets	0.0711	positive	Overall Attempt Success Rate over past three meets	0.0716	positive
Best Raw Squat to date	0.0811	positive	Squat3 Jump	0.0470	positive	Bench3 Past 3 meets success rate	0.0571	positive
Number of meets in past 12 months	0.0808	positive	Bench2 Past 3 meets success rate	0.0470	positive	Bench2 Past 3 meets success rate	0.0436	positive
Best Raw Total to date	0.0795	positive	Total Squat Jump	0.0460	positive	Squat2 Past 3 meets success rate	0.0381	positive
Best Raw Deadlift to date	0.0777	positive	Bench3 Past 3 meets success rate	0.0436	positive	Bench2 Jump	0.0371	negative
Deadlift1 Pass			Deadlift2 Pass			Deadlift3 Pass		
Feature	Corr	Influence	Feature	Corr	Influence	Feature	Corr	Influence
Deadlift1Kg	0.0543	negative	Deadlift2Kg	0.1052	negative	Deadlift2Kg	0.1866	negative
Total Bench Jump	0.0542	positive	Deadlift1Kg	0.0972	negative	Deadlift1Kg	0.1802	negative
Total Squat Jump	0.0540	positive	Overall Attempt Success Rate over past three meets	0.0896	positive	Deadlift3Kg	0.1715	negative
Squat1Kg	0.0529	negative	Squat2Kg	0.0879	negative	Best Raw Deadlift to date	0.1714	negative
Male	0.0513	negative	Squat1Kg	0.0859	negative	Best Raw Squat to date	0.1653	negative

Table 2: Top 5 linear relationships for each feature.

Of the three lifts, the bench press overall sees the lowest average success rate across all lifters. Lifters are more likely to pass their first attempt bench than their first attempt squat, however a large drop is seen from Bench 2 to Bench 3. The deadlift sees the highest average success rate.

Success rate per attempt								
Squat 1	Squat 2	Squat 3	Bench 1	Bench 2	Bench 3	Deadlift 1	Deadlift 2	Deadlift 3
88%	84%	68%	91%	46%	47%	95%	89%	66%

Table 3: Percentage successful lifts per attempt

Deadlift 2 and 3 success seem to be mostly impacted by Deadlift attempts 1 and 2 respectively and the same pattern is observed with the Squat attempts. Deadlift 2's impact on Deadlift 3 is shown in Figure 14. Interestingly, Squat attempt 3 seems to have a larger relative correlation to Deadlift 3. This may be due to the similarities between squats and deadlifts, which both engage the musculature of the lower body.

Bench press success seems to have a low impact on deadlift success, and squat success has a moderate impact on bench press success.

Largest between attempt correlation coefficients	Corr	Influence
Deadlift 2 -> Deadlift 3	0.2457	Positive
Deadlift 1 -> Deadlift 2	0.1629	Positive
Squat 3 -> Deadlift 3	0.1408	Positive
Squat 1 -> Squat 2	0.1405	Positive
Squat 2 -> Squat 3	0.1377	Positive

Lowest between attempt correlation coefficients	Corr	Influence
Bench 1 -> Bench 3	0.0183	Positive
Squat 1 -> Deadlift 3	0.0325	Positive
Bench 1 -> Deadlift 3	0.0417	Positive
Bench 3 -> Deadlift 1	0.0441	Positive
Squat 1 -> Squat 3	0.0481	Positive

Table 4 and Table 5: Top 5 (left) and bottom 5 (right) correlation coefficients between attempts.

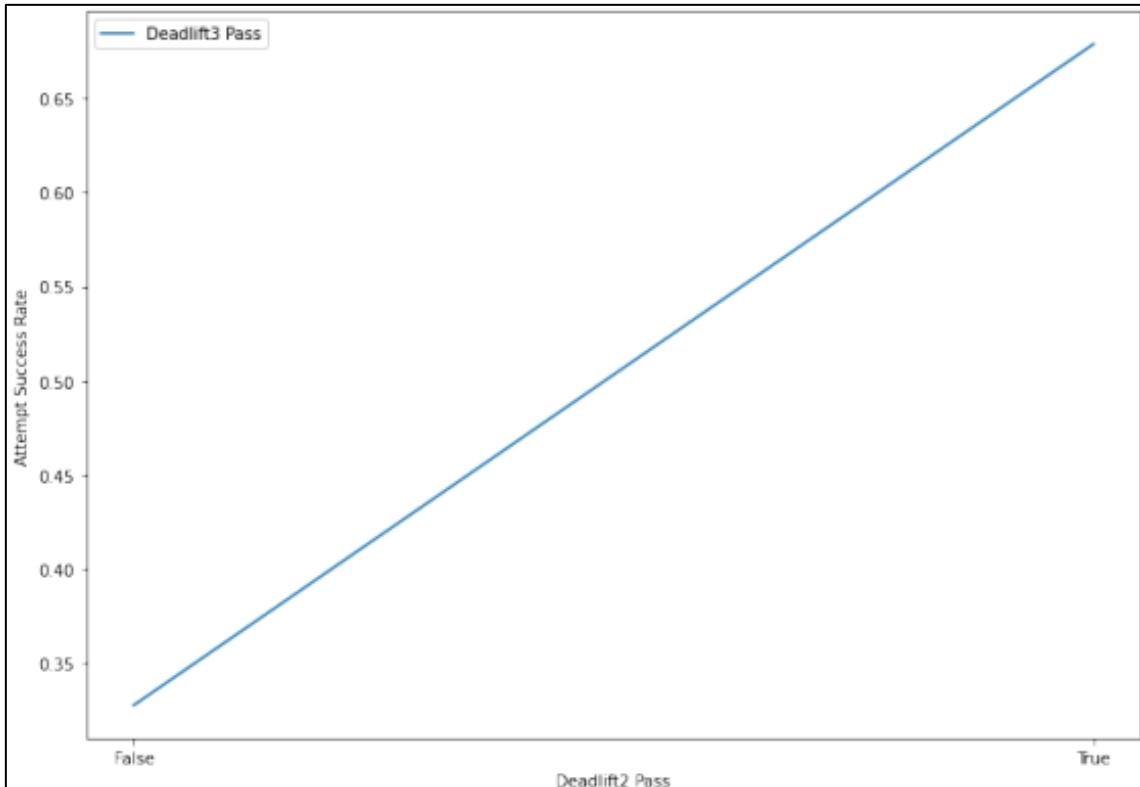


Figure 14: Deadlift 3's Average pass rate, depending on the success of Deadlift 2.

5.2 LINEAR MIXED MODELS

The python statsmodels module was used to create linear mixed effects models for each attempt. These models account for between-individual differences as well as general trends seen over the entire sample. The models were able to use the “Name” column in order to assess patterns seen in individuals. The applicability of these models to this project is unclear due to the low average number of meets per lifter (many lifters only had one or two records), however the models have been included for inference rather than prediction. In addition, linear mixed models are designed for regression rather than classification, which further discourages the use of them for prediction of attempt success/failure.

Each linear mixed model was fed with the entire feature space available for each attempt. The top five significant features for each attempt are shown below. The information should not be inferred from the magnitudes of the coefficients, as features were not scaled prior to input. Therefore features with higher magnitudes (e.g. *Squat1kg*) show smaller coefficients than those with lower magnitudes (e.g. *Total bench jump*). The features were sorted by z value rather than p value because the p values were often rounded down to zero.

The top five features for the mixed effects models are similarly patterned to the top 5 correlation coefficients for each attempt. The ‘in meet’ features become more significant as the meet progresses. This is consistent with the pattern observed in 5.1 Feature correlations.

Squat1 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.786	0.014	56
Age	-0	0	-11
Overall Attempt Success Rate over past three meets	0.125	0.014	9.3
Best Raw Wilks to date	0	0	7
BodyweightKg	0	0	-3
Squat1Kg	0	0	2.7
Group Var	0.004	0.002	

Squat2 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.519	0.016	31.613
Squat1 Pass	0.15	0.006	23.687
Overall Attempt Success Rate over past three meets	0.227	0.017	13.533
Age	-0	0	-8.126
BodyweightKg	0.001	0	7.178
Squat2Kg	0	0	-5.456
Group Var	0.001	0.002	

Squat3 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.159	0.022	7.107
Squat2 Pass	0.154	0.008	20.387
BodyweightKg	0.003	0	17.279
Squat3Kg	-0	0	-14.749
Overall Attempt Success Rate over past three meets	0.28	0.023	11.998
Squat1 Pass	0.054	0.008	6.36
Group Var	0.007	0.003	

Table 6, Table 7 and Table 8: Linear models for each squat attempt

Bench1 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.754	0.012	63
Squat2 Pass	0.038	0.004	11
Best Raw Wilks to date	0	0	10
Squat1 Pass	0.04	0.004	9.5
Squat3 Pass	0.025	0.003	9.2
Bench1Kg	0	0	-7.5
Group Var	0.001	0.002	

Bench2 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.421	0.02	21.491
Squat3 Pass	0.075	0.004	17.346
Squat2 Pass	0.059	0.006	10.138
Bench2Kg	-0	0	-9.417
Bench1 Pass	0.077	0.008	9.394
Overall Attempt Success Rate over past three meets	0.166	.019	8.846
Group Var	0.004	0.003	

Bench3 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	-0.02	0.025	-0.775
Bench2 Pass	0.173	0.008	21.668
Squat3 Pass	0.086	0.005	15.811
BodyweightKg	0.002	0	9.397
Total Bench Jump	-0.81	0.089	-9.177
Squat2 Pass	0.057	0.007	7.736
Group Var	0.006	0.003	

Table 9, Table 10 and Table 11: Linear models for each Bench attempt

Deadlift1 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.799	0.012	65
Bench1 Pass	0.046	0.004	11
Deadlift1Kg	0	0	-7.8
Squat1 Pass	0.026	0.004	7
Squat2 Pass	0.022	0.003	6.9
Bench2 Pass	0.019	0.003	6.6
Group Var	0.001	0.001	

Deadlift2 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.552	0.018	30.477
Deadlift2Kg	-0	0	-14.889
Deadlift1 Pass	0.113	0.008	13.521
Squat3 Pass	0.045	0.004	12.63
Squat2 Pass	0.05	0.005	10.383
Bench3 Pass	0.034	0.003	9.891
Group Var	0.002	0.002	

Deadlift3 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.243	0.027	9.13
Deadlift2 Pass	0.245	0.009	27.844
Deadlift3Kg	-0	0	-20.183
Squat3 Pass	0.076	0.005	14.472
Bench3 Pass	0.069	0.005	13.846
Bench2 Pass	0.063	0.007	9.313
Group Var	0.005	0.003	

Table 12, Table 13 and Table 14: Linear models for each deadlift attempt

The EDA scripts and full correlation coefficient tables can be found in Appendix G.

6 MODEL CREATION, TUNING AND EVALUATION

6.1 DATA PREPARATION

In order to train the neural network, the networks were fed “training” data. To keep in line with the goals of this project (predict future results based on past data), the training and testing data were split in a sequential nature. This training data consisted of the records between 01/01/2014 to 10/15/2019. The “testing set” i.e. the unseen data, consisted of the records after 10/15/2019 to 18/07/2021. Future data may give clues to the outcome of past data, therefore the model may be able to identify a lifter by interpolating between future and past values. The train: test ratio was 81:19. Of the training set, 40% of the data was held out for validation.

The models were trained and tuned on nine separate datasets, one for each attempt. The datasets were split with the same method used to prepare the data in 4.5.2.

The target classes were imbalanced, especially so for the first attempts of each lift ($\approx 90\%$ pass, 10% fail). Each dataset was balanced in favour of the minority class, as preliminary modelling on the imbalanced datasets swayed largely in favour of the majority classes.

WeightClassKg and *Sex*, both categorical features, were one-hot encoded prior to model training on the neural network, which led to 37 column additions to each dataset. The Histogram gradient boosted models had support for categorical features, therefore the features were simply ordinarily encoded and indicated as “categorical” prior to training.

6.2 NEURAL NETWORK WITH HYPERBAND HYPERPARAMETER TUNING

An artificial neural network is a directed graph of nodes, or “neurons”, which are connected via “activations”. They are hypothesised to mimic the way the human brain operates in order to make decisions.

A multi-layer perceptron (MLP) is a type of artificial neural network which consists of an input layer, one output layer, and at least one hidden layer. Each layer contains an array of neurons which are exhaustively connected to the neurons in the next layer. Each neuron contains a numeric value. In the non-input layers, each neuron value is calculated from the weighted sum of the neurons in the previous layer. In the case of this project, the input layer consists of the features of the data, and the output layer consists of the output i.e. the probability of a lifter passing or failing the attempt.

Everything between the input and output neurons (i.e. hidden layers) represent the “brain” which operates to solve the classification problem.

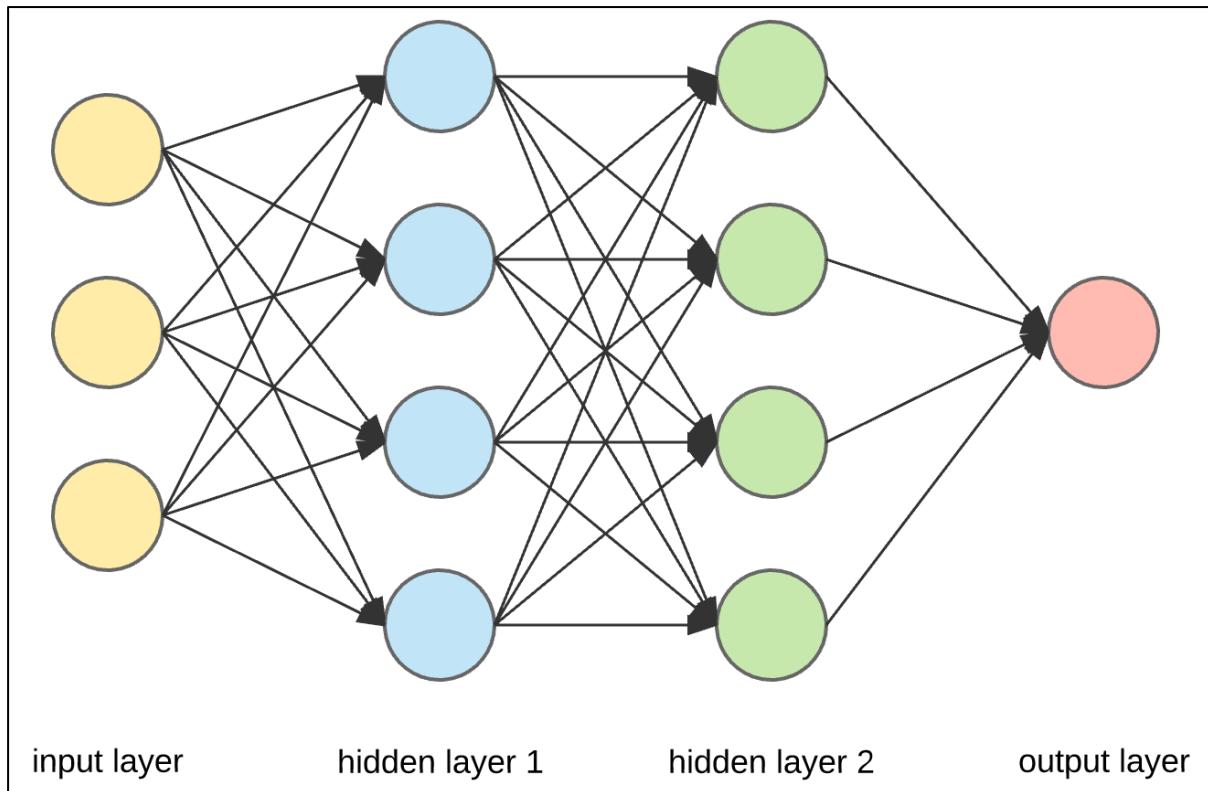


Figure 15: Typical multi-layer perceptron (MLP) neural network architecture [12]

The neural network is initialised with random weights, and undergoes several training cycles, or “epochs” where training data is fed through the network and the “weights” of each connection between neurons are adjusted in order to achieve the correct output. In our case, each epoch of training is judged on the “information loss function” between the predictions and the targets. In this case, the cost function was the **binary cross-entropy**. At the end of each epoch, an adjustment is made to the weights to cause the most rapid decrease to the loss function. The concept of minimizing the cost function is called “gradient descent”. This visualisation of gradient descent, for simple 2D and 3D functions are visualised in Figure 16 and Figure 17. In the case of the neural networks built for the openpowerlifting dataset, the cost function is highly dimensional, which increases the complexity of training.

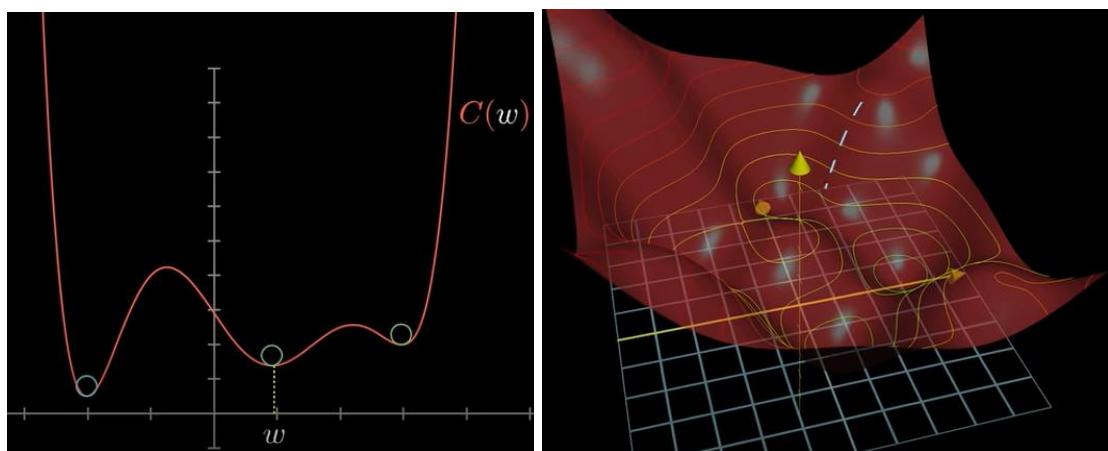


Figure 16 and Figure 17: Gradient descent visualised [13]

One obstacle to gradient descent, shown clearly above, is converging to a local minimum cost, rather than a global minimum. Imagine gradient descent as a ball rolling down a hill in a series of valleys; the ball can reach the bottom of one valley, however it may not be the lowest valley. Hyperparameter tuning can help address this problem.

6.2.1 Hyperband Hyperparameter Tuning

In order to optimise the set of hyperparameters for the neural network, the “Hyperband” optimisation algorithm [14] was implemented and coded with the Keras-Tuner python package.

In neural networks, hyperparameter configurations can experience some or all of the following during training [15]:

1. Immediate failure (divergence)
2. Low improvement

Fast improvement

Figure 18 is a demonstration of neural network behaviour. One set of hyperparameters may perform better than the other (lower loss) with less resource (epochs), however, with more resources (epochs), the lesser performing hyperparameter combination may perform better.

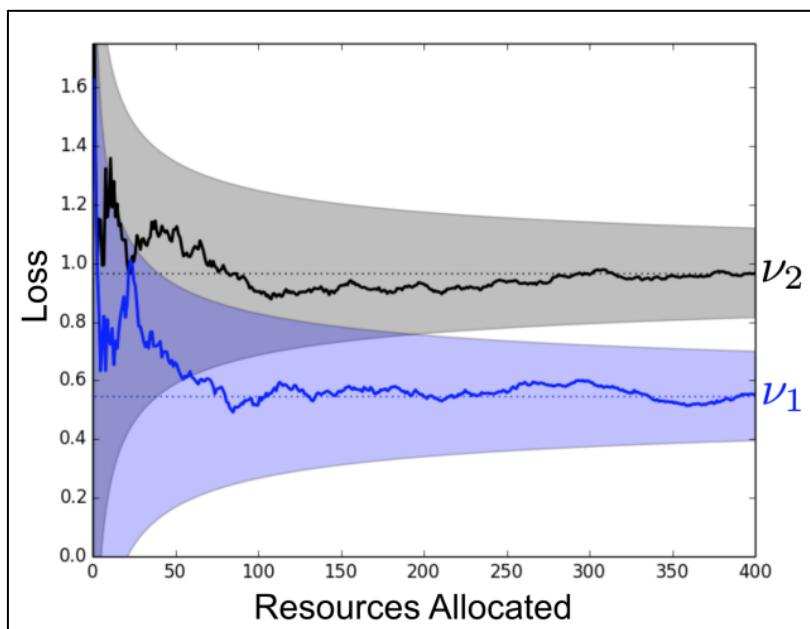


Figure 18: Performance of different hyperparameter combinations against total resources. [14]

A way to prevent resources being wasted on models that fail immediately, would be to implement a “successive halving” approach to the set of hyperparameters. A budget (e.g. number of training epochs) could be uniformly allocated to a random set of hyperparameter configurations. The worst 50% hyperparameter allocations will be removed, and the remaining configurations will be trained further. However, given a limited budget prior to halving (e.g. 10 epochs in the case of

Figure 18), some of the “worse” combinations may not get the chance to perform well. For example, if we had four clusters of hyperparameter combinations (each cluster has similar characteristics), successive halving would dispose of two clusters at epoch 10. It may, however, be the case that those disposed clusters tend to have superior performance if trained to epoch 100.

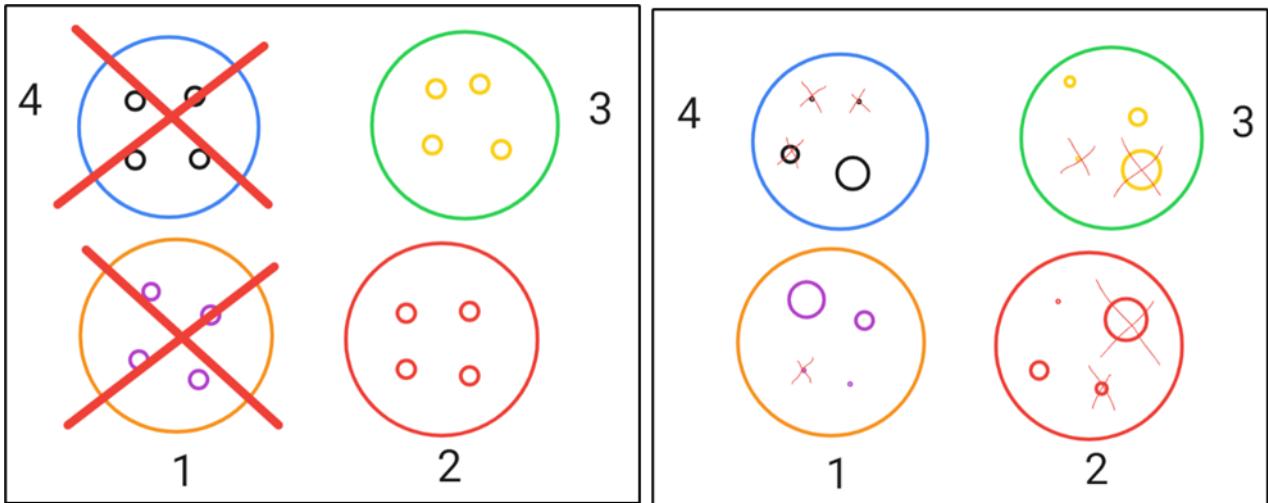


Figure 19 & Figure 20: Four clustered sets of four hyperparameter configurations. Left: each configuration is assigned $B/16$ epochs, and the worst 8 are discarded. Right: B epochs are randomly distributed amongst the 16 configurations, and the worst 8 performing configurations are discarded, providing a higher chance for “slow to train” estimators to display superior performance over the “fast to train” estimators.

The **hyperband** algorithm introduces another dimension of randomness, with a given budget. For example, if we were to have a set of 1000 total epochs, we would randomly distribute those epochs (with a minimum and maximum) to a set of hyperparameter combinations; combination 1 trains with 100 epochs, combination 2 is trained to 30 epochs, combination 3 is trained to 4 epochs. In the case of the four clusters of similar hyperparameters combinations, the hyperparameters that are “slow to train” have a higher likelihood to display superior performance. Hyperband essentially optimises successive halving, for a given number of resources.

The hyperparameter configurations were sampled from the search space defined in Figure 7. These search spaces were determined by reviewing the Keras API documentation.

Hyperparameter			Scale	Min	Max	Step	
Number of neurons in input layer			N/A	Number of features	Number of features	0	
Activation Function			N/A	Relu		N/A	
Output layer activation function			N/A	Sigmoid		N/A	
Number of hidden layers			N/A	5	5	0	
Number of neurons in each hidden layer			Linear	3	Number of features * 4	10	
Optimization algorithm	Adam	Learning Rate	Log	5e-5	2e-3	N/A	
	Adadelta	Learning rate	Log	0.8	5	N/A	
		Decay rate p	Linear	0.8	1.0	0.05	
Layer Activity regularization (l1)			Log	1e-6	1e-3	N/A	
Final Hidden Layer number of neurons			Linear	3	Number of features * 2	1	

Table 15: Hyperband hyperparameter search space for each neural network model.

The Relu activation function was chosen for the hidden layers, as this is shown to generally work well in neural networks. The sigmoid activation function was chosen in order to squeeze the output between 1 and 0, which would represent the probabilities of an attempt passing.

ADAM and Adadelta gradient optimisation algorithms were selected because ADAM has been shown to work well, and Adadelta was to be used as a failsafe.

L1 regularisation was applied to the output of each layer. This prevents the output from being dominated by features that encourage overfitting.

6.2.2 Trial and error epoch tuning

The model training parameters (batch size, number of epochs, early stopping, hyperband parameters) were determined through trial and error, by observing the validation accuracy fluctuations during model training, considering training time, and by reviewing user-based literature. The batch sizes, hyperband iterations, and hyperband factors were initially set to high values (5000, 6, and 40 respectively). This combination displayed a slow training time for the Squat1Pass model, a validation accuracy of ≈ 0.57 in the verbose output and a high number of epochs (>1000) to reach the that accuracy. The batch size was set to such a high value to speed up the training time of each epoch, and the belief that this would improve the generalisation accuracy. The batch size was lowered to 16 following a review of blog post *Effect of batch size on training dynamics* [16], and the training verbose output displayed validation accuracies of ≈ 0.59 with less epochs required (<100). It is not clear what causes the higher level of accuracy; there appears to be a trade-off between batch size and generalisation accuracy, and in this case, the smaller batch size gave a more optimal result.

A disadvantage of this trial and error approach is that the only verbose output monitored was for *Squat1 Pass*; the optimal combination of epoch parameters may have been different amongst the other target variables. The output monitoring could have been distributed amongst the other variables, however that would require the rewriting of various hyperparameter combinations amongst the various target variables, which is difficult for the average human to manage. With more time/resources, it would be sensible to automate the batch-size tuning process, rather than to rely on trial and error.

6.3 HISTOGRAM BASED GRADIENT BOOSTING CLASSIFIER

The second model trained on the data was Scikit-learn's experimental implementation of histogram-based gradient boosted decision trees (HGBDT). This model was chosen for its ability to handle missing values and categorical variables, and for its speed over conventional gradient boosting methods.

Gradient boosted models build an ensemble of decision trees. In each training iteration, the tree attempts to predict a value for each row. The residual errors are calculated from the predictions, and a new tree is then built to predict those residual errors. The new residual errors are multiplied by a learning rate to prevent overfitting. [17]

Histogram based gradient boosting brings significant improvements to the speed of training because continuous variables are placed into bins. This means that the variables are treated as integers during splitting, so continuous values do not have to be considered when making the splits. In addition, the binning reduces the number of splitting points to consider. [18]

Scikit-learn's implementation of histogram gradient boosting has support for missing values; when splitting trees, samples are assigned to the left or the right child based on the potential gain. This is advantageous for the *Time since last meet* and the set of *Attempt success rate* features, which were assigned values not specific to the lifter in order to be fed to the MLP model. The HGBDT models were trained on alternative datasets where the *Attempt success rate* values were not imputed (left

as NaN) and the *Time since last meet* features were not equal to the lifter's age (left as NaN) for "first meet lifters".

6.3.1 Bayesian optimisation model tuning

"Probability is orderly opinion and inference from data is nothing other than the revision of such opinion in the light of relevant new information."

Bayesian hyperparameter tuning [19] implements the concept of an optimal function that produces a model with the lowest loss $f(x)$, where x is a configuration of hyperparameters. During each iteration, the Bayesian Tuner attempts to select a hyperparameter combination that will provide the lowest loss, given the performance of prior iterations. If an iteration performs well, the tuner is adjusted to seek hyperparameter combinations similar to that combination. If a combination performs badly, the tuner is adjusted to seek hyper parameter combinations less similar to that combination.

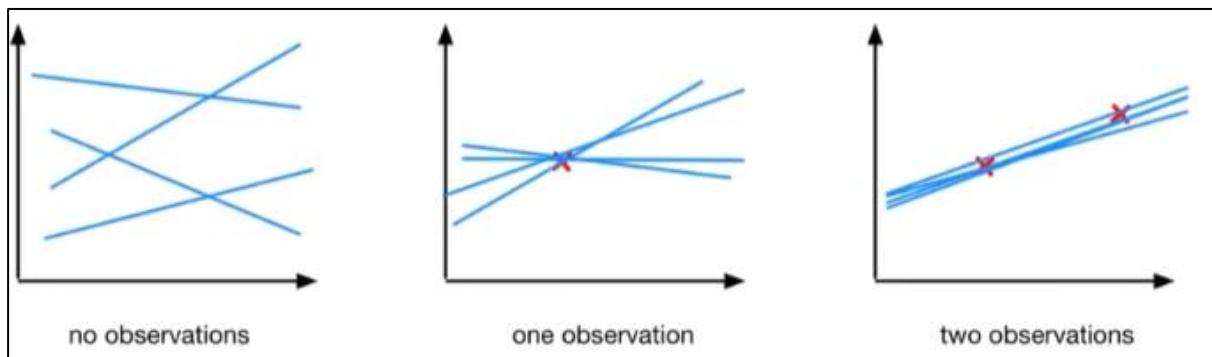


Figure 21: Bayesian optimisation of a linear function [20]

The tuner uses a "exploitation vs exploration" parameter β , which seeks to balance exploitation of the "current knowledge" of combinations that work, and exploration of "the unknown" – this is analogous to a data scientist selecting a model for a common data science problem; they could use the well-researched model that provides a predictable result if they exploit the well-researched methods (high exploitation, low β), or experiment with lesser researched methods, or even invent a new method (high exploration, high β). In this case, the Keras-tuner default value of 2.6 was used.

The second parameter to consider is the number of iterations of the tuner, or trials. The more trials, the closer one gets to the optimal number of hyperparameters. In case, 100 trials were used per dependent variable.

Hyperparameter	Scale	Min	Max	Step
Learning Rate	Log	0.01	1	N/A
Maximum Iterations	Log	1	1000	N/A
Maximum leaf nodes	Log	2	1000	N/A
Maximum Depth	Log	1	100	N/A
Minimum samples per leaf	Log	2	1000	N/A
L2 regularization	Log	0.0001	1.0	N/A
Maximum number of bins	Linear	40	255	20
Early stopping	Boolean	True	False	N/A

Table 16: Bayesian optimisation hyperparameter search space for each Histogram Gradient boosted tree model..

Due to the speed of the Bayesian optimisation algorithm, all possible hyperparameters were fed into the search space with large ranges.

6.4 MODEL EVALUATION

The primary method used for model evaluation was to compare the model test accuracy with the probability of the majority class occurring i.e. the average pass rate for each attempt, across the test data. This method places the models against a hypothetical person, the “Naïve guesser”, who simply classifies each attempt as belonging to the majority class. For example, for Deadlift1, the pass rate is 0.954, so for every 1000 lifts, 954 pass, making “pass” (Deadlift1 Pass = 1) the majority class. A person guessing that every lifter passes their deadlift would be right 95.4% of the time, therefore the model should have an accuracy > 95.4% to beat the person. The reason why this “Naïve guesser” was chosen is because preliminary model training was performed on the original imbalanced dataset, and the models were heavily biased towards the majority classes in all cases.

6.4.1 Results

The accuracies of each model vs the naïve guess for each attempt are shown below. All models were trained using a laptop running Windows 10 OS with a 1.3GHz CPU and 16GB RAM.

	HGBDT Model Accuracy	MLP Model Accuracy	Naive Guess	Naïve guess direction	HGBDT time to tune, train and test	MLP time to tune, train and test
Squat1 Pass	0.55	0.52	0.88	Pass rate	11m	1hr 6m
Squat2 Pass	0.62	0.51	0.84	Pass rate	8m	1hr 1m
Squat3 Pass	0.62	0.59	0.68	Pass rate	20m	2hr 1m
Bench1 Pass	0.61	0.46	0.91	Pass rate	22m	48m
Bench2 Pass	0.57	0.55	0.54	Fail rate	45m	2hr 0m
Bench3 Pass	0.58	0.54	0.53	Fail rate	41m	3hr 53m
Deadlift1 Pass	0.64	0.38	0.95	Pass rate	12m	37m
Deadlift2 Pass	0.71	0.68	0.89	Pass rate	20m	46m
Deadlift3 Pass	0.67	0.67	0.66	Pass rate	47m	2hr 7m

Table 17: Model accuracies vs the “naïve guess”

The machine learning models outperformed the naïve guesser (average pass or fail rate) for Bench2 Pass, Bench3 Pass and Deadlift3 Pass, by a small amount. The other models would be outperformed by an individual who simply assigns the majority class to the other attempts.

The Bench2 Pass, Bench3 Pass and Deadlift3 pass models being the most “well trained” may be due to the fact that they are the most balanced attempts in the dataset. Due to the under-sampling used to balance the classes, these features would have lost the lowest amount of rows, therefore losing the least amount of training data. This allows more accurate estimates to be made. In addition to this, these attempts are performed later on in the competition when there is more useful information pertaining to the lifter’s performance on the day. This may have consequently lead to stronger predictions.

The HGBDT models outperformed the MLP models for each attempt, and took less time to tune, train and test. The reasons for this are not simple to explain, however it is speculated that the following factors may have contributed to the higher performance:

1. The MLP models may require more detailed tuning.
2. The HGBDT models retained the NaN values, which served as a form of information. This information was lost when the dataset was trained on the MLP, as the MLP could not take NaN values.
3. The feature space was larger for the MLP because *WeightClassKg* was one-hot encoded.

Caution should be made when comparing the models to the naïve guess, as the naïve guess lacks sophistication. We have seen that there are large correlations between lifters failing attempts following the failure of a previous attempt. This is blatant when observing Deadlift3 (30% higher chance of success after passing Deadlift2, when all else is equal). The models only have a 1% difference over the naïve guess for Deadlift3, which shows that, while the models may have accounted for this “on the day” performance, they have also fit to the noise in the data, or do not have enough relevant data available to properly distinguish between the factors causing a lifter to fail or pass their attempts.

6.4.2 Feature importance

Table 18, Table 19 and Table 20 display the top five permutation-based feature importance’s for the strongest three models. These importance’s are calculated by randomly shuffling a feature and observing the decrease in model accuracy when making predictions on the shuffled data. This is an advantageous method to compare feature importance’s as it is model-agnostic. It is clear to see that the “on the day” features provide the most significance to the models, with the exception of the multi-layer perceptron Bench2 Pass model. A caveat to the permutation importance technique is that, if a more accurate predictor was permuted, the important features will not necessarily be the same. Features that are weak in a low-accuracy predictor may not be weak in a high accuracy feature. However, there is overlap between the permutation importance’s the correlation based importance’s and the linear mixed model significant predictors.

Bench2 Pass (Pass rate = 0.537)					
Histogram based gradient booster			Multi-layer perceptron		
Top 5 Features	Mean	Standard deviation	Top 5 Features	Mean	Standard deviation
<i>Bench2Kg</i>	0.0298	0.0014	<i>Squat3 Pass</i>	0.0144	0.0015
<i>Bench2 Jump</i>	0.0160	0.0012	<i>Number of Past Meets</i>	0.0133	0.0021
<i>Squat3 Pass</i>	0.0158	0.0034	<i>Time Since Last Meet</i>	0.0105	0.0021
<i>Best Raw Wilks to date</i>	0.0101	0.0022	<i>Bench3 Past 3 meets success rate</i>	0.0061	0.0013
<i>WeightClassKg</i>	0.0094	0.0024	<i>Squat2 Pass</i>	0.0055	0.0015

Table 18: Bench2 Pass Feature importance’s

Bench3 Pass (fail rate = 0.531)					
Histogram based gradient booster			Multi-layer perceptron		
Top 5 Features	Mean	Standard deviation	Top 5 Features	Mean	Standard deviation
<i>Bench3 Jump</i>	0.0388	0.0025	<i>Bench2 Pass</i>	0.0108	0.0013
<i>Bench3Kg</i>	0.0302	0.0017	<i>Bench3 Past 3 meets success rate</i>	0.0043	0.0021
<i>Squat3 Pass</i>	0.0112	0.0016	<i>Female</i>	0.0041	0.0018
<i>WeightClassKg</i>	0.0084	0.0011	<i>Squat3 Pass</i>	0.0036	0.0023
<i>Total Bench Jump</i>	0.0079	0.0015	<i>Squat2 Pass</i>	0.0031	0.0013

Table 19: Bench3 Pass Feature importance’s

Deadlift3 Pass (Pass rate = 0.660)					
Histogram based gradient booster			Multi-layer perceptron		
Top 5 Features	Mean	Standard deviation	Top 5 Features	Mean	Standard deviation
<i>Deadlift3Kg</i>	0.0389	0.0022	<i>Time Since Last Meet</i>	0.0308	0.0047
<i>Deadlift3Jump</i>	0.0278	0.0021	<i>Deadlift2 Pass</i>	0.0153	0.0012
<i>Deadlift2Kg</i>	0.0198	0.0024	<i>Female</i>	0.0095	0.0019
<i>Squat3 Pass</i>	0.0077	0.0021	<i>Bench3 Pass</i>	0.0081	0.0012
<i>WeightClassKg</i>	0.0052	0.0019	<i>Deadlift2Kg</i>	0.0078	0.0016

Table 20: Deadlift3 Pass Feature importance's

The full model results, including recall, precision f1 scores and ROC curves can be found in Appendix I.

6.5 LIMITATIONS

There are several factors that may have impacted the ability for the models to train optimally:

1. The dataset name disambiguation was partially complete due to time constraints. 3203 individuals identified as requiring disambiguation were not yet disambiguated, therefore the rows belonging to those individuals were not used for training and testing. Had this data been included, it is likely that the models would have reached a higher training and testing accuracy.
2. Of the data that had been included in the training, it is still possible that the data was not clean, or incomplete:
 - While the USAPL had reported all of their meets from 2014 onwards, it is possible that other federations had not report all their meets. This would have affected features such as *time since last meet* and *number of past meets*.
 - Of the data included in training and testing, it is possible that there were individuals who required disambiguation, but were not identified as such, such as lifters who shared the same name but were very close in age. However, this was assumed to be quite rare.
3. The parallelisation capabilities of the hyperband model were not leveraged, which limited the number of resources that could be used to tune the MLP hyperparameters. Consequently, the maximum number of training epochs was an arbitrary limit based on the resources available, rather than the actual number of epochs required for optimal convergence of the MLP models.

7 CONCLUSION

Powerlifting attempt selection and success is a complex human issue that is difficult to predict with a machine. However, this project has determined that there is scope to make predictions that are on the threshold of meaningful, particularly for lift attempts that occur later in powerlifting meets, when taking “on the day” data into account.

This project provides a format for more accurate models to be developed from the openpowerlifting dataset. The following additions are likely to strengthen these models:

1. The addition of the full, cleaned and disambiguated USAPL lifter set. These is also scope to add data from other powerlifting federations that are also marked as “complete” on openpowerlifting.org.
2. Features that are not yet available, particularly bar velocity, which is shown to decrease as attempts get closer to a lifter’s max.
3. A lifter’s pre-competition training data could be used as model inputs.

This project has also shown that there are partially automated algorithms to clean the openpowerlifting dataset which are recommended for implementation.

Should a more accurate model be trained, there is scope to create an interface which web scrapes live powerlifting meets from websites such as liftingcast.com, which provides live in-meet data including the bodyweight of the lifter, weight class and weights to be attempted.

Appendix A Dataset Filtering script

Appendix_A_Dataset Transformation

September 9, 2021

Importing files

```
[18]: import pandas as pd
import time

#import numpy as np

openpowerlifting_full_dataset = pd.read_csv("open_powerlifting_original_data.
→csv")

openpowerlifting_full_dataset.shape
```

[18]: (2492593, 41)

Displaying head of dataset

```
[7]: print(openpowerlifting_full_dataset.head())
```

	Name	Sex	Event	Equipment	Age	AgeClass	BirthYearClass	\
0	Dominik Gabriel	M	B	Raw	17.0	16-17	14-18	
1	Marek Herák	M	B	Multi-ply	19.0	18-19	19-23	
2	Miroslav Adamové	M	B	Multi-ply	18.0	18-19	14-18	
3	Kamil Lipiński	M	B	Multi-ply	19.0	18-19	19-23	
4	Gabriel Kováč	M	B	Multi-ply	20.0	20-23	19-23	

	Division	BodyweightKg	WeightClassKg	...	Tested	Country	State	\
0	T 16-17	102.5	110	...	NaN	Slovakia	NaN	
1	T 18-19	59.8	60	...	NaN	Slovakia	NaN	
2	T 18-19	87.7	90	...	NaN	Slovakia	NaN	
3	T 18-19	89.5	90	...	NaN	Poland	NaN	
4	Juniors	81.6	82.5	...	NaN	Slovakia	NaN	

	Federation	ParentFederation	Date	MeetCountry	MeetState	MeetTown	\
0	WUAP	WUAP	2015-06-16	Czechia	NaN	Praha	
1	WUAP	WUAP	2015-06-16	Czechia	NaN	Praha	
2	WUAP	WUAP	2015-06-16	Czechia	NaN	Praha	
3	WUAP	WUAP	2015-06-16	Czechia	NaN	Praha	
4	WUAP	WUAP	2015-06-16	Czechia	NaN	Praha	

```
MeetName
0 European Championships
1 European Championships
2 European Championships
3 European Championships
4 European Championships
```

[5 rows x 41 columns]

Reviewing types of each column:

```
[8]: openpowerlifting_full_dataset.dtypes
```

```
Name          object
Sex          object
Event         object
Equipment    object
Age           float64
AgeClass      object
BirthYearClass object
Division      object
BodyweightKg  float64
WeightClassKg object
Squat1Kg      float64
Squat2Kg      float64
Squat3Kg      float64
Squat4Kg      float64
Best3SquatKg  float64
Bench1Kg      float64
Bench2Kg      float64
Bench3Kg      float64
Bench4Kg      float64
Best3BenchKg  float64
Deadlift1Kg   float64
Deadlift2Kg   float64
Deadlift3Kg   float64
Deadlift4Kg   float64
Best3DeadliftKg float64
TotalKg       float64
Place         object
Dots          float64
Wilks          float64
Glossbrenner  float64
Goodlift       float64
Tested        object
Country        object
State          object
Federation     object
```

```

ParentFederation      object
Date                  object
MeetCountry          object
MeetState             object
MeetTown              object
MeetName              object
dtype: object

```

We need to identify lifters who have not competed in the USAPL, have not competed raw, and have not competed after 2013, then we will save a database containing the lifters who have competed in the USAPL, raw and after 2013.

```
[9]: start_time = time.time()
openpowerlifting_full_dataset = openpowerlifting_full_dataset.
    ↪sort_values(by=["Name"], ignore_index=True)
openpowerlifting_full_dataset.insert(loc = 41, column = "Competed raw in USAPL"
    ↪since 2014", value = False)
starting_index = 0 # linear search
finishing_index = 0
for name in openpowerlifting_full_dataset["Name"].drop_duplicates():
    competed_raw_in_USAPL_since_2014 = False # Assumed False
    while openpowerlifting_full_dataset.iloc[finishing_index]["Name"] == name:
        if openpowerlifting_full_dataset.iloc[finishing_index]["Federation"] == \
            ↪"USAPL" \
            and openpowerlifting_full_dataset.iloc[finishing_index]["Date"] >= \
            ↪"2014-01-01" \
            and openpowerlifting_full_dataset.iloc[finishing_index]["Equipment"] == \
            ↪"Raw": #
                competed_raw_in_USAPL_since_2014 = True
        finishing_index += 1
        if finishing_index == openpowerlifting_full_dataset.shape[0]:
            break #error handling
    if competed_raw_in_USAPL_since_2014 == True:
        openpowerlifting_full_dataset.iloc[starting_index:
            ↪finishing_index]["Competed raw in USAPL since 2014"] = True
        starting_index = finishing_index

print(f"{time.time() - start_time} seconds taken")
print(len(openpowerlifting_full_dataset[openpowerlifting_full_dataset["Competed"
    ↪raw in USAPL since 2014"] == True]))
```

```
<ipython-input-9-74b8b4beab82>:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
`openpowerlifting_full_dataset.iloc[starting_index:finishing_index]["Competed`

```
raw in USAPL since 2014"] = True  
687.0685620307922 seconds taken  
229372
```

```
[7]: filename = "open_powerlifting_marked_subsets.csv"  
compression_options = dict(method='zip', archive_name=f'{filename}.csv')  
openpowerlifting_full_dataset.to_csv(f'{filename}.zip',  
                                   compression=compression_options)  
openpowerlifting_full_dataset[openpowerlifting_full_dataset["Competed raw in"  
                                   "USAPL since 2014"] == True].drop(  
    labels = "Competed raw in USAPL since 2014", axis = 1).  
    to_csv("open_powerlifting_USAPL_Raw_active_lifters.csv")
```

Optimised version of the subsetting code - timed to assess the difference

```
[17]: openpowerlifting_full_dataset = pd.read_csv("open_powerlifting_original_data.  
                                                csv")  
start_time = time.time()  
openpowerlifting_full_dataset = openpowerlifting_full_dataset.  
    sort_values(by=["Name"], ignore_index=True)  
openpowerlifting_USAPL_lifters = openpowerlifting_full_dataset[  
    openpowerlifting_full_dataset["Name"].isin(  
        openpowerlifting_full_dataset[(openpowerlifting_full_dataset["Federation"]  
        == "USAPL") &  
        (openpowerlifting_full_dataset["Equipment"] == "Raw") &  
        (openpowerlifting_full_dataset["Date"] >= "2014-01-01")]["Name"].drop_duplicates())]  
print(f"{time.time() - start_time} seconds taken")  
print(len(openpowerlifting_USAPL_lifters))
```

```
5.593708038330078 seconds taken  
229372
```

```
[ ]:
```

Appendix B Repeat Data handling

Appendix_B_Data cleaning - Repeat data and age

September 16, 2021

Many of the statistics have missing values due to incomplete data, either from old meets or federations who do not enter their lifter data:

Age: Should not have any missing values. Approximate ages are given as integer + 0.5 i.e. 23.5 implies that lifter turns 24 that year.

Ageclass: Range of ages where lifter's age falls. Useful especially when age column is empty. This will be used to interpolate ages

BirthYearClass: similar to age class, but primarily used by IPF affiliates. This will be used to interpolate ages.

Division: Free-form text. Only 23 missing values. This is the lowest number of missing values, so will be investigated first.

BodyweightKg: This should not really have any missing values, as weight class is what determines who the lifter competes against. Early speculation is that this is old data, therefore it will be investigated further. and flagged.

WeightClassKg: Similar logic to BodyweightKG. This should not have missing values, as lifter compete in weight classes. Likely not IPF affiliate federations, or they are federations with old data.

Squat/Bench/Deadlift(1kg/2kg/3kg): First, second and third attempt information for each lift. Some federations only report the best attempt in Best3(Squat/Bench/Deadlift)kg. In addition, non full-power meets will not always contain information for every lift. For example, Bench only meets (B in "Event" column) will not definitely not contain squat and deadlift information.

Squat4kg/Bench4kg/Deadlift4kg: Forth attempt lifts, used for recording single lift records. not counting towards the total, and mostly empty.

Best3(Squat/Deadlift/Bench)kg: Max of first three successfull attempts for each lift. These can be empty if lifter failed all three attempts of a lift, or if competition was not full-power. Additionally, can be negative number to show the lowest weight a lifter attempted and failed. All three lifts can be completely empty for some meets, and only the total will be reported in TotalKG

TotalKg: Small amount (5%) missing.

DOTS/Wilks/Glossbrenner/Goodlift: relative scoring formulas. Wilks is the oldest formula. Newer competitions are using goodlift and DOTS.+. https://www.powerlifting.sport/fileadmin/ipf/data/ipf-formula/Models_Evaluation-I-2020.pdf

Tested: Whether the lifter competed in a drug-tested meet or not. Empty indicated that the category was not drug tested. As we are analysing USAPL competitions from 2014 onwards, these

ones should all be indicated as YES. This will need to be checked.

Country: Origin country of the lifter.

State: Optional. not likely to be relevant. Home state of the lifter

ParentFederation: Optional

MeetState: Optional

MeetTown: ?

Importing files

```
[1]: import pandas as pd
import datetime
from dateutil.relativedelta import *
import time

#import numpy as np

openpowerlifting_USAPL_lifters = pd.
→read_csv("open_powerlifting_USAPL_Raw_active_lifters.csv")
```

```
[8]: openpowerlifting_USAPL_lifters.drop(columns = "Unnamed: 0", inplace = True)
```

1 Checking for repeats (i.e. where name and date are the same)

Some entries are repeat entries, as a lifter can enter multiple divisions (if one is an open division, and the other is a non-open division). We will remove those repeat entries.

```
[16]:
```

```
[17]: #1 use the name from the drop duplicates table
#2 starting index = The first row that has the occurrence of that name
```

```
[ ]: start_time = time.time()
openpowerlifting_USAPL_lifters = openpowerlifting_USAPL_lifters.
→sort_values(by=["Name", "Date"], ignore_index=True)
openpowerlifting_USAPL_lifters.insert(loc = 41, column = "Repeated_meet", value=
→= False)
openpowerlifting_USAPL_lifters.insert(loc = 42, column =
→"Two_meets_on_one_day", value = False)
starting_index = 0
finishing_index = 0
for name in openpowerlifting_USAPL_lifters["Name"].drop_duplicates():
    repeated_meet = False
    while openpowerlifting_USAPL_lifters.iloc[finishing_index]["Name"] == name:
        finishing_index +=1
        if finishing_index == openpowerlifting_USAPL_lifters.shape[0]:
```

```

        break
for i in range(starting_index, finishing_index):
    meet_1 = openpowerlifting_USAPL_lifters.iloc[i]
    competition_1 = meet_1["MeetName"]
    date_1 = meet_1["Date"]
    equipment_1 = meet_1["Equipment"]
    name_1 = meet_1["Name"]
    event_1 = meet_1["Event"]
    bodyweight_1 = meet_1["BodyweightKg"]
    for j in range(i+1, finishing_index):
        meet_2 = openpowerlifting_USAPL_lifters.iloc[j]
        competition_2 = meet_2["MeetName"]
        date_2 = meet_2["Date"]
        equipment_2 = meet_2["Equipment"]
        name_2 = meet_2["Name"]
        event_2 = meet_2["Event"]
        bodyweight_2 = meet_2["BodyweightKg"]
        if date_1 == date_2 and name_1 == name_2 and competition_1 == competition_2 and bodyweight_1 == bodyweight_2:
            if event_1 == "S":
                if event_2 in ["SD", "SB", "SBD"] and equipment_1 == equipment_2:
                    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
            elif event_2 == event_1 and equipment_1 == equipment_2:
                openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
            else:
                openpowerlifting_USAPL_lifters.at[i, "Two_meets_on_one_day"], openpowerlifting_USAPL_lifters.at[j, "Two_meets_on_one_day"] = True, True

            elif event_1 == "B":
                if event_2 in ["BD", "SB", "SBD"] and equipment_1 == equipment_2:
                    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
            elif event_2 == event_1 and equipment_1 == equipment_2:
                openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
            else:
                openpowerlifting_USAPL_lifters.at[i, "Two_meets_on_one_day"], openpowerlifting_USAPL_lifters.at[j, "Two_meets_on_one_day"] = True, True

        elif event_1 == "D":

```

```

if event_2 in ["BD", "SD", "SBD"] and equipment_1 == equipment_2:
    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
elif event_2 == event_1 and equipment_1 == equipment_2:
    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
else:
    openpowerlifting_USAPL_lifters.at[i, "Two_meets_on_one_day"], openpowerlifting_USAPL_lifters.at[j, "Two_meets_on_one_day"] = True, True

elif event_1 == "SD":
    if event_2 in ["S", "D"] and equipment_1 == equipment_2:
        openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True
elif event_2 == "SBD" and equipment_1 == equipment_2:
    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
elif event_2 == event_1 and equipment_1 == equipment_2:
    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
else:
    openpowerlifting_USAPL_lifters.at[i, "Two_meets_on_one_day"], openpowerlifting_USAPL_lifters.at[j, "Two_meets_on_one_day"] = True, True

elif event_1 == "SB":
    if event_2 in ["S", "B"] and equipment_1 == equipment_2:
        openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True
elif event_2 == "SBD" and equipment_1 == equipment_2:
    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
elif event_2 == event_1 and equipment_1 == equipment_2:
    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
else:
    openpowerlifting_USAPL_lifters.at[i, "Two_meets_on_one_day"], openpowerlifting_USAPL_lifters.at[j, "Two_meets_on_one_day"] = True, True

elif event_1 == "BD":
    if event_2 in ["B", "D"] and equipment_1 == equipment_2:
        openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True

```

```

        elif event_2 == "SBD" and equipment_1 == equipment_2:
            openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
        elif event_2 == event_1 and equipment_1 == equipment_2:
            openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
        else:
            openpowerlifting_USAPL_lifters.at[i, "Two_meets_on_one_day"], openpowerlifting_USAPL_lifters.at[j, "Two_meets_on_one_day"] = True, True

        elif event_1 == "SBD":
            if event_2 in ["S", "B", "D", "SD", "SB", "BD"] and
equipment_1 == equipment_2:
                openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True
        elif event_2 == event_1 and equipment_1 == equipment_2:
            openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
        else:
            openpowerlifting_USAPL_lifters.at[i, "Two_meets_on_one_day"], openpowerlifting_USAPL_lifters.at[j, "Two_meets_on_one_day"] = True, True
            elif date_1 == date_2 and name_1 == name_2 and (competition_1 != competition_2 or bodyweight_1 != bodyweight_2):
                openpowerlifting_USAPL_lifters.at[i, "Two_meets_on_one_day"], openpowerlifting_USAPL_lifters.at[j, "Two_meets_on_one_day"] = True, True

    starting_index = finishing_index

print(f"{time.time() - start_time} seconds remaining")

```

[90]: openpowerlifting_USAPL_lifters.
`to_csv("open_powerlifting_USAPL_Raw_active_lifters_columns_added.csv", index=False)`

next step with the new columns: delete the repeats, and then investigate the values of “same day meets”. If lifters competes on the same day but their bodyweight is different, then it is likely that they should be distinguished from each other. If lifters compete on the same day but the meet name is different, it is likely that they should be distinguished from each other.

2 Age investigation part 1

We will derive “date of birth” intervals from the age column, ageclass column, and birthyearclass columns.

```
[93]: openpowerlifting_USAPL_lifters = pd.  
      →read_csv("open_powerlifting_USAPL_Raw_active_lifters_columns_added.csv")  
openpowerlifting_USAPL_lifters['Date'] = pd.  
      →to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
```

```
[105]: openpowerlifting_USAPL_lifters.at[0, "Age"]
```

```
[105]: 21.5
```

```
[94]: openpowerlifting_USAPL_lifters.insert(loc = 43, column = "DateOfBirthInterval",  
      →value = None)
```

```
[75]: #print(openpowerlifting_USAPL_lifters["BirthYearClass"].unique(), "\n")  
#print(openpowerlifting_USAPL_lifters["AgeClass"].unique())
```

```
['19-23' '24-39' '40-49' '14-18' nan '70-999' '50-59' '60-69']
```

```
['20-23' '24-34' '45-49' '16-17' '13-15' '18-19' nan '35-39' '40-44'  
 '5-12' '65-69' '70-74' '75-79' '50-54' '55-59' '60-64' '80-999']
```

```
[96]: print("The number of lifters where the 'age' column is not empty = " +  
      →str(openpowerlifting_USAPL_lifters[pd.  
      →notna(openpowerlifting_USAPL_lifters["Age"])].shape[0]))  
print("The number of lifters where the 'age' column is empty, but the  
      →'AgeClass' column has values = " + str(openpowerlifting_USAPL_lifters[pd.  
      →isna(openpowerlifting_USAPL_lifters["Age"]) & pd.  
      →notna(openpowerlifting_USAPL_lifters["AgeClass"])].shape[0]))  
print("The number of lifters where the 'age' column is empty, the 'AgeClass'  
      →column is empty, but the 'BirthYearClass' column has values' = " +  
      →str(openpowerlifting_USAPL_lifters[pd.  
      →isna(openpowerlifting_USAPL_lifters["Age"]) & pd.  
      →isna(openpowerlifting_USAPL_lifters["AgeClass"]) & pd.  
      →notna(openpowerlifting_USAPL_lifters["BirthYearClass"])].shape[0]))
```

The number of lifters where the 'age' column is not empty = 194356
The number of lifters where the 'age' column is empty, but the 'AgeClass' column has values = 3690
The number of lifters where the 'age' column is empty, the 'AgeClass' column is empty, but the 'BirthYearClass' column has values' = 1583

```
[117]: openpowerlifting_USAPL_lifters[pd.notna(openpowerlifting_USAPL_lifters["Age"])]
```

```
[117]:      Name  Sex Event   Equipment   Age  AgeClass BirthYearClass \
64        AJ  Dawes    M     SBD      Wraps  NaN      NaN       NaN
65        AJ  Dawes    M     SBD        Raw  NaN      NaN       NaN
66        AJ  Dawes    M     SBD        Raw  NaN      NaN       NaN
67        AJ  Dawes    M     SBD        Raw  NaN      NaN       NaN
98    AJ Rodriguez    M     SBD  Single-ply  NaN      NaN       NaN
```

...									
229271	Zoe Gonzales	F	SBD	Single-ply	NaN	NaN	NaN	NaN	NaN
229272	Zoe Gonzales	F	SBD	Single-ply	NaN	NaN	NaN	NaN	NaN
229273	Zoe Gonzales	F	SBD	Single-ply	NaN	NaN	NaN	NaN	NaN
229274	Zoe Gonzales	F	SBD	Single-ply	NaN	NaN	NaN	NaN	NaN
229279	Zoe Gonzales	F	SBD	Single-ply	NaN	NaN	NaN	NaN	NaN
	Division	BodyweightKg	WeightClassKg	...	Federation	\			
64	pure	104.33	110	...	NASA				
65	MR-T3	114.90	120	...	USAPL				
66	MR-T3	110.00	110	...	USAPL				
67	MR-O	105.00	105	...	USAPL				
98	Boys	50.94	51.7	...	THSPA				
...				
229271	Girls	52.34	56	...	THSWPA				
229272	Girls	51.80	51.9	...	THSWPA				
229273	Girls	51.35	51.9	...	THSWPA				
229274	Girls	51.48	51.9	...	THSWPA				
229279	Girls	66.86	67.3	...	THSWPA				
	ParentFederation	Date	MeetCountry	MeetState	MeetTown	\			
64		NaN 2013-04-13	USA	OH	Springfield				
65		IPF 2014-02-22	USA	MI	Kalamazoo				
66		IPF 2014-04-12	USA	OH	NaN				
67		IPF 2015-04-18	USA	OH	NaN				
98		NaN 2017-02-04	USA	TX	Mission Veterans				
...				
229271		NaN 2014-02-06	USA	TX	San Angelo				
229272		NaN 2014-02-15	USA	TX	McCamey				
229273		NaN 2014-02-28	USA	TX	Mertzon				
229274		NaN 2014-03-14	USA	TX	Corpus Christi				
229279		NaN 2020-02-06	USA	TX	Whitesboro				
	MeetName	Repeated_meet	\						
64	Ohio State Championships	False							
65	No Frills Meet	False							
66	Battle of the Great Lakes	False							
67	18th Annual Battle of the Great Lakes	False							
98	Mission Veterans Invitational	False							
...							
229271	Lake View Powerlifting Meet	False							
229272	McCamey Powerlifting Meet	False							
229273	Girls Region 1 Division 3	False							
229274	Girls State Meet	False							
229279	Whitesboro Girls Meet	False							
	Two_meets_on_one_day	DateOfBirthInterval							

```

64           False      None
65           False      None
66           False      None
67           False      None
98           False      None
...
229271        ...      ...
229272        False      None
229273        False      None
229274        False      None
229279        False      None

```

[33785 rows x 44 columns]

The below function will create intervals from the known ages/age ranges. The Age column is prioritised, then the AgeClass column, due to smaller ranges than the BirthYearClass column, so we can infer more about a lifter's age.

There are 5694 lifter names for which there are entries where no age exists, out of 48275 names.

```

[ ]: def create_known_age_intervals(dataset):
    known_age_subset = dataset[pd.notna(dataset["Age"])]
    known_AgeClass_subset = dataset[pd.isna(dataset["Age"]) & pd.
    ↪notna(dataset["AgeClass"])]
    # age year class is the primary age class because it contains smaller age
    ↪ranges
    known_BirthYearClass_subset = dataset[pd.isna(dataset["Age"]) & pd.
    ↪isna(dataset["AgeClass"]) & pd.notna(dataset["BirthYearClass"])]
```



```

    for i in known_age_subset.index:
        print(i)
        age = known_age_subset.at[i, "Age"]
        date_of_meet = known_age_subset.at[i, "Date"]
        if age.is_integer(): # lifter could turn a year older
            dataset.at[i, "DateOfBirthInterval"] = pd.Interval(
                left = date_of_meet + relativedelta(days = 1) -
            ↪relativedelta(years = age+1),
                right = date_of_meet - relativedelta(years = age), closed = "
            ↪"both")
        else: #lifter turns age that year if their age is integer divisible by .
        ↪5
            age = age+0.5
            year_born = int(known_age_subset.at[i, "Date"].year - age)
            dataset.at[i, "DateOfBirthInterval"] = pd.Interval(
                left = pd.Timestamp(year = year_born, month = 1, day = 1),
                right = pd.Timestamp(year = year_born, month = 12, day = 31),
                closed = "both")
```

```

for i in known_AgeClass_subset.index:
    print(i)
    interval = known_AgeClass_subset.at[i, "AgeClass"].split("-")
    youngest_age, oldest_age = int(interval[0]), int(interval[1])
    if oldest_age > 123:
        oldest_age = 123
    date_of_meet = known_AgeClass_subset.at[i, "Date"]
    dataset.at[i, "DateOfBirthInterval"] \
        = pd.Interval(date_of_meet - relativedelta(years = oldest_age),
                      date_of_meet - relativedelta(years =_
→youngest_age))

for i in known_BirthYearClass_subset.index:
    print(i)
    interval = known_BirthYearClass_subset.at[i, "BirthYearClass"] .
→split("-")
    youngest_age, oldest_age = int(interval[0]), int(interval[1])
    if oldest_age > 123: #handling age range that were imputted as *-999,_
→as this produces an error.
        oldest_age = 123
    date_of_meet = known_BirthYearClass_subset.at[i, "Date"]
    dataset.at[i, "DateOfBirthInterval"] \
        = pd.Interval(date_of_meet - relativedelta(years = oldest_age),
                      date_of_meet - relativedelta(years =_
→youngest_age))

create_known_age_intervals(openpowerlifting_USAPL_lifters)

```

[120]: openpowerlifting_USAPL_lifters.
→to_csv("open_powerlifting_USAPL_Raw_active_lifters_known_ages_added.csv",
→index = False)

3 Age investigation part 2

We will derive “date of birth” intervals from divisions where “age”, “ageclass” and “birthyearclass” columns are empty. We will find the divisions that are not “open”, and attach age intervals to them.

[1]: import pandas as pd
import datetime
from dateutil.relativedelta import *
import json

```

openpowerlifting_USAPL_lifters = pd.
    ↪read_csv("open_powerlifting_USAPL_Raw_active_lifters_known_ages_added.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    ↪to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')

```

[9]: # counting the number of divisions where there are entries where age, age class, and birth year class are not available.

```

division_counts = openpowerlifting_USAPL_lifters[pd.
    ↪isna(openpowerlifting_USAPL_lifters["Age"]) \
        & pd.
    ↪isna(openpowerlifting_USAPL_lifters["AgeClass"]) \
        & pd.
    ↪isna(openpowerlifting_USAPL_lifters["BirthYearClass"])]["Division"] .
    ↪value_counts()
print(division_counts)
divisions = division_counts.index

```

F_TEM_X_APF	1
MSR	1
Amateur Open 18+	1
Masters 35-55	1
NV	1
M-2	1
F-TJ	1
M-I	1
M-C-M1	1
FR-M4	1
M-E-m1	1
retronov	1
Submasters 1	1
M-O-SP	1
Freshman/Sophomore	1
FMXR	1
Women	1
middle	1
Classic Law/Fire/Military	1
M-M4a	1
Teen 13-15	1
M-M7	1
Intermediate	1
wm2	1
F-L	1
Masters 60-64	1
F-C-U23	1
Submasters 2	1
M-M3b	1

```

Elite Amateur Submasters 33-39      1
S 33-39                           1
Masters LW                         1
F_SR_ACPF                         1
MR-M7                            1
MOpe-Ra                          1
Teen LW                           1
wpn                               1
M_OES_APF                         1
MO-MP                            1
M 40+                             1
M-E-m3                           1
F-Jr                            1
F-M1                            1
Law/Fire                          1
retrowpure                        1
Name: Division, dtype: int64

```

We can see that there are 486 unique divisions that do not have ages attached to them in some cases. These divisions will be manually labelled.

```

[3]: # There are 486 divisions which contain lifters that have no age information.
# we will create an automated process which assigns intervals to classes for
# which an age can be found.
# drop from index: pd.Index.drop(labels,)
# we want to create list in the file containing the age intervals that an age
# class can or may contain, with some caveats
# we then want to insert those intervals into birth year class and age class.

def modify_unknown_division_ages(dataset):
    i = 0
    while True:
        division_counts = dataset[pd.isna(dataset["Age"]) \
                                    & pd.isna(dataset["AgeClass"]) \
                                    & pd.isna(dataset["BirthYearClass"])]["Division"]. \
            value_counts()
        divisions = division_counts.index
        if len(divisions) == 0:
            print("There are no more divisions to check")
            dataset.to_csv("open_powerlifting_USAPL_Raw_active_lifters.csv", \
                           index = False)
            break
        next_division = divisions[i]
        choice = int(input(f"The division to be checked is {next_division}. \
                           Press 1 to enter an interval, 2 if it is an open or ageless division, \
                           3 if you want to skip it for later, 4 if you want to see the head of \
                           the division name, and 5 to quit: "))

```

```

"""choice to create new interval beloning to a division, choice to drop
the division name,
choice to skip the division for now (i+1),
choice to visualise the head of the data containing that specific
division"""

if choice == 1:
    # first: enter the intervals
    lower_interval = int(input(f"Enter the minimum age for this
division({next_division}): or -1 to quit: "))
    higher_interval = int(input(f"Enter the maximum age for this
division({next_division}), or -1 to quit: "))

    if (lower_interval == -1) or (higher_interval == -1):
        dataset.

    to_csv("open_powerlifting_USAPL_Raw_active_lifters_known_ages_added.csv",_
index = False)
        break

    interval = f"{lower_interval}-{higher_interval}"
    youngest_age = lower_interval
    oldest_age = higher_interval + 1 # if the lifter is on the edge of
their category i.e. 36 about to turn 37 in the 30-36 category: the date of
birth interval will account for this.
    # second: identify the subset containing the division and empty age/
birth year classes, and populate the date of birth interval
    ##### dataset dob interval: first value = meet date - higher
interval in years.
    ##### second value = meet date - lower
interval in years.
    #####
    ##### create interval with pd.interval
subset = dataset.loc[pd.isna(dataset["Age"]) \
& pd.isna(dataset["AgeClass"])\
& pd.isna(dataset["BirthYearClass"])\
& (dataset["Division"] == next_division)]
num_rows_changed = subset.shape[0]
for number in subset.index:
    date_of_meet = subset.at[number, "Date"]

    dataset.at[number, "DateOfBirthInterval"] \
= pd.Interval(date_of_meet - relativedelta(years = oldest_age),
              date_of_meet - relativedelta(years =
youngest_age))
    # third: modify the birth and age class pre-existing columns to
contain these intervals

```

```

        dataset.at[number, "AgeClass"], dataset.at[number, "BirthYearClass"] = interval, interval

    try:
        #1: create a new file if it does not exist, then add the first
        ↪dictionary entry to it
        #2: if the file already exists, load the json dictionary from it
        #3:
        with open("division_age_dictionary.txt", "r+") as file:
            division_age_dict = json.loads(file.read())
            division_age_dict[next_division] = (interval, ↪
        ↪f"{{num_rows_changed}} rows")
            file.close()
        with open("division_age_dictionary.txt", "w") as file:
            json.dump(division_age_dict, file)
            file.close()
    except:
        with open("division_age_dictionary.txt", "w") as file:
            division_age_dict = {next_division: (interval, ↪
        ↪f"{{num_rows_changed}} rows")}
            json.dump(division_age_dict, file)
            file.close()

    if choice == 2:
        description = str(input("Enter 'Open' if the division is 'Open', or
        ↪a custom description for the age, or Q to quit: "))
        if description == "Q" or description == "q":
            dataset.
        ↪to_csv("open_powerlifting_USAPL_Raw_active_lifters_known_ages_added.csv",
        ↪index = False)
            break

        subset = dataset.loc[pd.isna(dataset["Age"]) \
            & pd.isna(dataset["AgeClass"])\
            & pd.isna(dataset["BirthYearClass"])\
            & (dataset["Division"] == next_division)]
        num_rows_changed = subset.shape[0]

        for number in subset.index:

            dataset.at[number, "DateOfBirthInterval"] \
            = description
            # third: modify the birth and age class pre-existing columns to
            ↪contain these intervals
            dataset.at[number, "AgeClass"], dataset.at[number, "BirthYearClass"] = description, description

```

```

try:
    #1: create a new file if it does not exist, then add the first
    ↪dictionary entry to it
    #2: if the file already exists, load the json dictionary from it
    #3:
    with open("division_age_dictionary.txt", "r+") as file:
        division_age_dict = json.loads(file.read())
        division_age_dict[next_division] = (description, ↪
    ↪f"{{num_rows_changed}} rows")
        file.close()
    with open("division_age_dictionary.txt", "w") as file:
        json.dump(division_age_dict, file)
        file.close()
except:
    with open("division_age_dictionary.txt", "w") as file:
        division_age_dict = {next_division: (description, ↪
    ↪f"{{num_rows_changed}} rows")}
        json.dump(division_age_dict, file)
        file.close()
if choice == 3:
    i += 1
if choice == 4:
    print(dataset[(pd.notna(dataset["Age"]) \ 
                  | pd.notna(dataset["AgeClass"]))\ 
                  | pd.notna(dataset["BirthYearClass"])) & 
    ↪(dataset["Division"] == next_division)]["Age"].value_counts())
    print(dataset[(pd.notna(dataset["Age"]) \ 
                  | pd.notna(dataset["AgeClass"]))\ 
                  | pd.notna(dataset["BirthYearClass"])) & 
    ↪(dataset["Division"] == next_division)]["AgeClass"].value_counts())
    print(dataset[(pd.notna(dataset["Age"]) \ 
                  | pd.notna(dataset["AgeClass"]))\ 
                  | pd.notna(dataset["BirthYearClass"])) & 
    ↪(dataset["Division"] == next_division)]["BirthYearClass"].value_counts())
    if choice == 5:
        dataset.
    ↪to_csv("open_powerlifting_USAPL_Raw_active_lifters_known_ages_added.csv", ↪
    ↪index = False)
        break

```

[11]: modify_unknown_division_ages(openpowerlifting_USAPL_lifters)

The division to be checked is M-I. Press 1 to enter an interval, 2 if it is an open or ageless division, 3 if you want to skip it for later, 4 if you want to see the head of the division name, and 5 to quit: 4

Series([], Name: Age, dtype: int64)
Series([], Name: AgeClass, dtype: int64)

```
Series([], Name: BirthYearClass, dtype: int64)
The division to be checked is M-I. Press 1 to enter an interval, 2 if it is an
open or ageless division,           3 if you want to skip it for later, 4 if you
want to see the head of the division name, and 5 to quit: 2
Enter 'Open' if the division is 'Open', or a custom description for the age, or
Q to quit: Open
The division to be checked is M-C-M1. Press 1 to enter an interval, 2 if it is
an open or ageless division,           3 if you want to skip it for later, 4 if
you want to see the head of the division name, and 5 to quit: 4
41.0      2
40.0      2
42.0      1
Name: Age, dtype: int64
40-44      5
Name: AgeClass, dtype: int64
40-49      29
Name: BirthYearClass, dtype: int64
The division to be checked is M-C-M1. Press 1 to enter an interval, 2 if it is
an open or ageless division,           3 if you want to skip it for later, 4 if
you want to see the head of the division name, and 5 to quit: 1
Enter the minimum age for this division(M-C-M1): or -1 to quit: 40
Enter the maximum age for this division(M-C-M1), or -1 to quit: 49
The division to be checked is FR-M4. Press 1 to enter an interval, 2 if it is an
open or ageless division,           3 if you want to skip it for later, 4 if you
want to see the head of the division name, and 5 to quit: 4
69.5      19
70.5      15
71.5      14
73.5      9
74.5      8
78.5      8
72.5      6
75.5      5
71.0      4
73.0      4
76.5      3
77.5      3
70.0      3
72.0      2
74.0      2
75.0      2
58.5      2
78.0      1
90.0      1
69.0      1
54.5      1
55.5      1
Name: Age, dtype: int64
```

```

70-74      67
75-79      22
65-69      20
55-59       3
80-999     1
50-54       1
Name: AgeClass, dtype: int64
70-999     110
50-59       4
Name: BirthYearClass, dtype: int64
The division to be checked is FR-M4. Press 1 to enter an interval, 2 if it is an
open or ageless division,           3 if you want to skip it for later, 4 if you
want to see the head of the division name, and 5 to quit: 1
Enter the minimum age for this division(FR-M4): or -1 to quit: 70
Enter the maximum age for this division(FR-M4), or -1 to quit: 74
The division to be checked is M-E-m1. Press 1 to enter an interval, 2 if it is
an open or ageless division,           3 if you want to skip it for later, 4 if
you want to see the head of the division name, and 5 to quit: 4
Series([], Name: Age, dtype: int64)
Series([], Name: AgeClass, dtype: int64)
Series([], Name: BirthYearClass, dtype: int64)
The division to be checked is M-E-m1. Press 1 to enter an interval, 2 if it is
an open or ageless division,           3 if you want to skip it for later, 4 if
you want to see the head of the division name, and 5 to quit: 2
Enter 'Open' if the division is 'Open', or a custom description for the age, or
Q to quit: Open
The division to be checked is Submasters 1. Press 1 to enter an interval, 2 if
it is an open or ageless division,           3 if you want to skip it for later, 4
if you want to see the head of the division name, and 5 to quit: 4
30.5      1
Name: Age, dtype: int64
24-34      1
Name: AgeClass, dtype: int64
24-39      1
Name: BirthYearClass, dtype: int64
The division to be checked is Submasters 1. Press 1 to enter an interval, 2 if
it is an open or ageless division,           3 if you want to skip it for later, 4
if you want to see the head of the division name, and 5 to quit: 1
Enter the minimum age for this division(Submasters 1): or -1 to quit: 30
Enter the maximum age for this division(Submasters 1), or -1 to quit: 39
The division to be checked is retronnov. Press 1 to enter an interval, 2 if it is
an open or ageless division,           3 if you want to skip it for later, 4 if
you want to see the head of the division name, and 5 to quit: 4
Series([], Name: Age, dtype: int64)
Series([], Name: AgeClass, dtype: int64)
Series([], Name: BirthYearClass, dtype: int64)
The division to be checked is retronnov. Press 1 to enter an interval, 2 if it is
an open or ageless division,           3 if you want to skip it for later, 4 if

```

```
you want to see the head of the division name, and 5 to quit: 2
Enter 'Open' if the division is 'Open', or a custom description for the age, or
Q to quit: Open
The division to be checked is Freshman/Sophomore. Press 1 to enter an interval,
2 if it is an open or ageless division,           3 if you want to skip it for
later, 4 if you want to see the head of the division name, and 5 to quit: 4
16.0      1
Name: Age, dtype: int64
16-17     1
Name: AgeClass, dtype: int64
14-18     1
Name: BirthYearClass, dtype: int64
The division to be checked is Freshman/Sophomore. Press 1 to enter an interval,
2 if it is an open or ageless division,           3 if you want to skip it for
later, 4 if you want to see the head of the division name, and 5 to quit: 1
Enter the minimum age for this division(Freshman/Sophomore): or -1 to quit: 14
Enter the maximum age for this division(Freshman/Sophomore), or -1 to quit: 16
The division to be checked is FMXR. Press 1 to enter an interval, 2 if it is an
open or ageless division,           3 if you want to skip it for later, 4 if you
want to see the head of the division name, and 5 to quit: 4
40.0      1
61.0      1
Name: Age, dtype: int64
60-64     1
40-44     1
Name: AgeClass, dtype: int64
60-69     1
40-49     1
Name: BirthYearClass, dtype: int64
The division to be checked is FMXR. Press 1 to enter an interval, 2 if it is an
open or ageless division,           3 if you want to skip it for later, 4 if you
want to see the head of the division name, and 5 to quit: 4
40.0      1
61.0      1
Name: Age, dtype: int64
60-64     1
40-44     1
Name: AgeClass, dtype: int64
60-69     1
40-49     1
Name: BirthYearClass, dtype: int64
The division to be checked is FMXR. Press 1 to enter an interval, 2 if it is an
open or ageless division,           3 if you want to skip it for later, 4 if you
want to see the head of the division name, and 5 to quit: 2
Enter 'Open' if the division is 'Open', or a custom description for the age, or
Q to quit: Open
The division to be checked is M 40+. Press 1 to enter an interval, 2 if it is an
open or ageless division,           3 if you want to skip it for later, 4 if you
```

want to see the head of the division name, and 5 to quit: 4
 Series([], Name: Age, dtype: int64)
 Series([], Name: AgeClass, dtype: int64)
 Series([], Name: BirthYearClass, dtype: int64)
 The division to be checked is M 40+. Press 1 to enter an interval, 2 if it is an open or ageless division, 3 if you want to skip it for later, 4 if you want to see the head of the division name, and 5 to quit: 1
 Enter the minimum age for this division(M 40+): or -1 to quit: 40
 Enter the maximum age for this division(M 40+), or -1 to quit: 123
 The division to be checked is Women. Press 1 to enter an interval, 2 if it is an open or ageless division, 3 if you want to skip it for later, 4 if you want to see the head of the division name, and 5 to quit: 4
 Series([], Name: Age, dtype: int64)
 Series([], Name: AgeClass, dtype: int64)
 Series([], Name: BirthYearClass, dtype: int64)
 The division to be checked is Women. Press 1 to enter an interval, 2 if it is an open or ageless division, 3 if you want to skip it for later, 4 if you want to see the head of the division name, and 5 to quit: 2
 Enter 'Open' if the division is 'Open', or a custom description for the age, or Q to quit: Open
 The division to be checked is MR-M7. Press 1 to enter an interval, 2 if it is an open or ageless division, 3 if you want to skip it for later, 4 if you want to see the head of the division name, and 5 to quit: 4
 Series([], Name: Age, dtype: int64)
 Series([], Name: AgeClass, dtype: int64)
 Series([], Name: BirthYearClass, dtype: int64)
 The division to be checked is MR-M7. Press 1 to enter an interval, 2 if it is an open or ageless division, 3 if you want to skip it for later, 4 if you want to see the head of the division name, and 5 to quit: 2
 Enter 'Open' if the division is 'Open', or a custom description for the age, or Q to quit: Open
 The division to be checked is Classic Law/Fire/Military. Press 1 to enter an interval, 2 if it is an open or ageless division, 3 if you want to skip it for later, 4 if you want to see the head of the division name, and 5 to quit: 2
 Enter 'Open' if the division is 'Open', or a custom description for the age, or Q to quit: Open
 The division to be checked is M-M4a. Press 1 to enter an interval, 2 if it is an open or ageless division, 3 if you want to skip it for later, 4 if you want to see the head of the division name, and 5 to quit: 4

71.0	13
72.0	10
70.5	9
70.0	9
71.5	7
73.0	7
69.5	6
73.5	4

```

72.5      4
74.0      3
74.5      1
Name: Age, dtype: int64
70-74     67
65-69     6
Name: AgeClass, dtype: int64
70-999    86
Name: BirthYearClass, dtype: int64
The division to be checked is M-M4a. Press 1 to enter an interval, 2 if it is an
open or ageless division,           3 if you want to skip it for later, 4 if you
want to see the head of the division name, and 5 to quit: 1
Enter the minimum age for this division(M-M4a): or -1 to quit: 70
Enter the maximum age for this division(M-M4a), or -1 to quit: 74
The division to be checked is retrowpure. Press 1 to enter an interval, 2 if it
is an open or ageless division,           3 if you want to skip it for later, 4 if
you want to see the head of the division name, and 5 to quit: 2
Enter 'Open' if the division is 'Open', or a custom description for the age, or
Q to quit: Open
The division to be checked is Teen 13-15. Press 1 to enter an interval, 2 if it
is an open or ageless division,           3 if you want to skip it for later, 4 if
you want to see the head of the division name, and 5 to quit: 1
Enter the minimum age for this division(Teen 13-15): or -1 to quit: 13
Enter the maximum age for this division(Teen 13-15), or -1 to quit: 15
The division to be checked is M-E-m3. Press 1 to enter an interval, 2 if it is
an open or ageless division,           3 if you want to skip it for later, 4 if
you want to see the head of the division name, and 5 to quit: 4
Series([], Name: Age, dtype: int64)
Series([], Name: AgeClass, dtype: int64)
Series([], Name: BirthYearClass, dtype: int64)
The division to be checked is M-E-m3. Press 1 to enter an interval, 2 if it is
an open or ageless division,           3 if you want to skip it for later, 4 if
you want to see the head of the division name, and 5 to quit: 2
Enter 'Open' if the division is 'Open', or a custom description for the age, or
Q to quit: Open
There are no more divisions to check

```

Algorithm to determine lifters that have large gaps between meets: create new column called “time passed since last meet”. for each lifter in dataframe: iterate through each row belonging to that lifter. if row before name != name of current row: fill in “time passed since last meet” with 0. subtract the date of current meet from the date of previous meet to get time passed since previous meet, and fill in the “time passed since last meet” with that time (in fractional years) ### first meet will have zero gap between meets

4 Deleting Repeats

We will now delete the lifters with repeated meets, and investigate those who have meets on the same day, but it does not look like the meets are repeated.

```
[20]: import pandas as pd

openpowerlifting_USAPL_lifters = pd.
    →read_csv("open_powerlifting_USAPL_Raw_active_lifters_known_ages_added.csv")
openpowerlifting_USAPL_lifters =_
    →openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Repeated_meet"]_]
    ↵== False]
openpowerlifting_USAPL_lifters.
    →to_csv("open_powerlifting_USAPL_Raw_active_lifters.csv", index = False)
```

```
[21]: import pandas as pd

pd.set_option("display.max_columns", None)

openpowerlifting_USAPL_lifters = pd.
    →read_csv("open_powerlifting_USAPL_Raw_active_lifters.csv")
openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Two_meets_on_one_day"]_]
    ↵== True]
```

	Name	Sex	Event	Equipment	Age	AgeClass	BirthYearClass	\
18	AJ Alvarez	M	SBD	Single-ply	16.0	16-17		14-18
19	AJ Alvarez	M	SBD	Single-ply	16.0	16-17		14-18
22	AJ Alvarez	M	SBD	Single-ply	16.0	16-17		14-18
23	AJ Alvarez	M	SBD	Single-ply	16.0	16-17		14-18
28	AJ Alvarez	M	SBD	Single-ply	17.0	16-17		14-18
...
203277	Zack Bartell	M	B	Raw	23.0	20-23		19-23
203507	Zane Johnson	M	SBD	Wraps	NaN	14-19		14-19
203508	Zane Johnson	M	BD	Raw	NaN	14-19		14-19
203790	Zoe Haynes	F	D	Raw	23.0	20-23		NaN
203791	Zoe Haynes	F	B	Raw	23.0	20-23		NaN
	Division	BodyweightKg	WeightClassKg	Squat1Kg	Squat2Kg	\		
18	Boys	80.38	82.1	NaN	NaN			
19	Boys	88.09	89.8	NaN	NaN			
22	Boys	89.36	89.8	NaN	NaN			
23	Boys	80.47	82.1	NaN	NaN			
28	Boys	88.36	89.8	NaN	NaN			
...			
203277	Juniors	20-23	99.40	100	NaN	NaN		
203507		hs	50.62	52	NaN	NaN		
203508		hs	50.62	52	NaN	NaN		
203790		F_JR_AWPC	59.30	60	NaN	NaN		
203791		F_JR_WPC	59.30	60	NaN	NaN		
	Squat3Kg	Squat4Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	\	
18	NaN	NaN	158.76	NaN	NaN	NaN		

19	NaN	NaN	206.38	NaN	NaN	NaN	NaN	
22	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
23	NaN	NaN	165.56	NaN	NaN	NaN	NaN	
28	NaN	NaN	208.65	NaN	NaN	NaN	NaN	
...	
203277	NaN	NaN	NaN	180.0	187.5	-192.5		
203507	NaN	NaN	102.50	NaN	NaN	NaN	NaN	
203508	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
203790	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
203791	NaN	NaN	NaN	60.0	-67.5	-67.5		
	Bench4Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	\		
18	NaN	92.99	NaN	NaN	NaN	NaN		
19	NaN	111.13	NaN	NaN	NaN	NaN		
22	NaN	NaN	NaN	NaN	NaN	NaN		
23	NaN	99.79	NaN	NaN	NaN	NaN		
28	NaN	117.93	NaN	NaN	NaN	NaN		
...		
203277	NaN	187.50	NaN	NaN	NaN	NaN		
203507	NaN	65.00	NaN	NaN	NaN	NaN		
203508	NaN	65.00	NaN	NaN	NaN	NaN		
203790	NaN	NaN	132.5	-142.5	-143.0			
203791	NaN	60.00	NaN	NaN	NaN	NaN		
	Deadlift4Kg	Best3DeadliftKg	TotalKg	Place	Dots	Wilks	\	
18	NaN	176.90	428.64	17	294.75	291.76		
19	NaN	208.65	526.17	2	343.97	339.67		
22	NaN	NaN	NaN	DQ	NaN	NaN		
23	NaN	176.90	442.25	15	303.90	300.82		
28	NaN	204.12	530.70	7	346.38	342.04		
...		
203277	NaN	NaN	187.50	1	115.71	114.39		
203507	NaN	142.50	310.00	2	304.68	312.99		
203508	NaN	142.50	207.50	2	203.94	209.50		
203790	NaN	132.50	132.50	1	147.98	149.07		
203791	NaN	NaN	60.00	1	67.01	67.51		
	Glossbrenner	Goodlift	Tested	Country	State	Federation	\	
18	281.06	50.45	Yes	USA	NaN	THSPA		
19	325.85	58.74	Yes	USA	NaN	THSPA		
22	NaN	NaN	Yes	USA	NaN	THSPA		
23	289.77	52.01	Yes	USA	NaN	THSPA		
28	328.08	59.15	Yes	USA	NaN	THSPA		
...		
203277	109.26	86.14	Yes	USA	CA	USPA		
203507	308.72	55.70	Yes	USA	VA	NASA		
203508	206.64	NaN	Yes	USA	VA	NASA		

203790	131.74	NaN	Yes	USA	NaN	WPC
203791	59.65	51.93	NaN	USA	NaN	WPC
	ParentFederation	Date	MeetCountry	MeetState		\
18	NaN	2011-02-12	USA	TX		
19	NaN	2011-02-12	USA	TX		
22	NaN	2011-02-26	USA	TX		
23	NaN	2011-02-26	USA	TX		
28	NaN	2012-02-04	USA	TX		
...		
203277	IPL	2019-09-28	USA	CA		
203507	NaN	2017-03-26	USA	NaN		
203508	NaN	2017-03-26	USA	NaN		
203790	WPC	2016-08-06	USA	MI		
203791	WPC	2016-08-06	USA	MI		
	MeetTown			MeetName		\
18	NaN	Mission Veterans Memorial Power Meet				
19	Gregory-Portland Texas	GP Wildcat Powerlifting Invitational				
22	Kingsville	3rd Annual Brahma Bash				
23	PSJA Southwest	Southwest Powerlifting				
28	Mission Vets	Mission Vets Invitational				
...		
203277	Long Beach	Drug Tested Southern California Open				
203507	NaN	Highschool Nationals				
203508	NaN	Highschool Nationals				
203790	Grand Rapids	Single Lift Championships				
203791	Grand Rapids	Single Lift Championships				
	Repeated_meet	Two_meets_on_one_day		DateOfBirthInterval		
18	False	True	[1994-02-13, 1995-02-12]			
19	False	True	[1994-02-13, 1995-02-12]			
22	False	True	[1994-02-27, 1995-02-26]			
23	False	True	[1994-02-27, 1995-02-26]			
28	False	True	[1994-02-05, 1995-02-04]			
...		
203277	False	True	[1995-09-29, 1996-09-28]			
203507	False	True	(1997-03-26, 2003-03-26]			
203508	False	True	(1997-03-26, 2003-03-26]			
203790	False	True	[1992-08-07, 1993-08-06]			
203791	False	True	[1992-08-07, 1993-08-06]			

[8045 rows x 44 columns]

If a lifter has completed twice in one day, it is very likely that it was the same meet if their best squat, best deadlift or best bench was exactly the same. Therefore we will delete the row that contains the “lesser” meet e.g. if one meet was “SB” and the other was “SBD” but the best squat and best bench were equal, the “SB” meet will be deleted as it does not contain the deadlift data.

```
[ ]: starting_index = 0
finishing_index = 0
for name in openpowerlifting_USAPL_lifters["Name"].drop_duplicates():
    print(name)
repeated_meet = False
while openpowerlifting_USAPL_lifters.iloc[finishing_index]["Name"] == name:
    finishing_index +=1
    if finishing_index == openpowerlifting_USAPL_lifters.shape[0]:
        break
for i in range(starting_index, finishing_index):
    meet_1 = openpowerlifting_USAPL_lifters.iloc[i]
    competition_1 = meet_1["MeetName"]
    date_1 = meet_1["Date"]
    equipment_1 = meet_1["Equipment"]
    name_1 = meet_1["Name"]
    event_1 = meet_1["Event"]
    bodyweight_1 = meet_1["BodyweightKg"]
    bestsquat_1 = meet_1["Best3SquatKg"]
    bestbench_1 = meet_1["Best3BenchKg"]
    bestdead_1 = meet_1["Best3DeadliftKg"]
    for j in range(i+1, finishing_index):
        meet_2 = openpowerlifting_USAPL_lifters.iloc[j]
        competition_2 = meet_2["MeetName"]
        date_2 = meet_2["Date"]
        equipment_2 = meet_2["Equipment"]
        name_2 = meet_2["Name"]
        event_2 = meet_2["Event"]
        bodyweight_2 = meet_2["BodyweightKg"]
        bestsquat_2 = meet_2["Best3SquatKg"]
        bestbench_2 = meet_2["Best3BenchKg"]
        bestdead_2 = meet_2["Best3DeadliftKg"]

        if date_1 == date_2 and name_1 == name_2 and ((bestsquat_1 == bestsquat_2) or (bestbench_1 == bestbench_2) or (bestdead_1 == bestdead_2)):
            if event_1 == "S":
                if event_2 in ["SD", "SB", "SBD"]:
                    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
            elif event_1 == "B":
                if event_2 in ["BD", "SB", "SBD"]:
                    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] = True
            elif event_1 == "D":
                if event_2 in ["BD", "SD", "SBD"]:
```

```

        openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] =_
↪True

        elif event_1 == "SD":
            if event_2 in ["S", "D"]:
                openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] =_
↪True
            elif event_2 == "SBD":
                openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] =_
↪True

            elif event_1 == "SB":
                if event_2 in ["S", "B"]:
                    openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] =_
↪True
                elif event_2 == "SBD":
                    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] =_
↪True

            elif event_1 == "BD":
                if event_2 in ["B", "D"]:
                    openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] =_
↪True
                elif event_2 == "SBD":
                    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] =_
↪True

            elif event_1 == "SBD":
                if event_2 in ["S", "B", "D", "SD", "SB", "BD"]:
                    openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] =_
↪True
                elif event_2 == event_1:
                    openpowerlifting_USAPL_lifters.at[i, "Repeated_meet"] =_
↪True

        starting_index = finishing_index

```

```
[23]: openpowerlifting_USAPL_lifters =_
↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Repeated_meet"] =_
↪== False]
openpowerlifting_USAPL_lifters.
↪to_csv("open_powerlifting_USAPL_Raw_active_lifters.csv", index = False)
```

Appendix C Lifter Disambiguation

Appendix_C_Data cleaning part 3 - truncating lifts and identifying incorrect intervals

September 16, 2021

- 1 Deleting duplicate meets where bench and deadlift have been separated from each other, and creating list of lifters where ages do not align in their meets, or bodyweights do not align on “Same-day” meets.

```
[6]: import pandas as pd
import datetime
from dateutil.relativedelta import *
import json

openpowerlifting_USAPL_lifters = pd.
    ↪read_csv("open_powerlifting_USAPL_Raw_active_lifters.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    ↪to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
```

```
[7]: pd.set_option("display.max_columns", None)
```

```
[8]: # function to truncate meets where single lifts have been split
def truncate_meets():
    def male_wilks_coefficients():
        return (-216.0475144, 16.2606339, -0.002388645, -0.00113732, 7.
    ↪01863*10**-6, -1.291*10**-8)
    def female_wilks_coefficients():
        return (594.31747775582, -27.23842536447, 0.82112226871, -0.
    ↪00930733913, 4.731582*10**-5, -9.054*10**-8)

    starting_index = 0
    finishing_index = 0
    for name in openpowerlifting_USAPL_lifters["Name"].drop_duplicates():
        print(name)
        repeated_meet = False
        while openpowerlifting_USAPL_lifters.iloc[finishing_index]["Name"] ==
    ↪name:
            finishing_index +=1
```

```

    if finishing_index == openpowerlifting_USAPL_lifters.shape[0]:
        break
    for i in range(starting_index, finishing_index):
        meet_1 = openpowerlifting_USAPL_lifters.iloc[i]
        competition_1 = meet_1["MeetName"]
        date_1 = meet_1["Date"]
        equipment_1 = meet_1["Equipment"]
        name_1 = meet_1["Name"]
        event_1 = meet_1["Event"]
        bodyweight_1 = meet_1["BodyweightKg"]
        bestsquat_1 = meet_1["Best3SquatKg"]
        bestbench_1 = meet_1["Best3BenchKg"]
        bestdead_1 = meet_1["Best3DeadliftKg"]
        for j in range(i+1, finishing_index):
            meet_2 = openpowerlifting_USAPL_lifters.iloc[j]
            competition_2 = meet_2["MeetName"]
            date_2 = meet_2["Date"]
            equipment_2 = meet_2["Equipment"]
            name_2 = meet_2["Name"]
            event_2 = meet_2["Event"]
            bodyweight_2 = meet_2["BodyweightKg"]
            bestsquat_2 = meet_2["Best3SquatKg"]
            bestbench_2 = meet_2["Best3BenchKg"]
            bestdead_2 = meet_2["Best3DeadliftKg"]

            if date_1 == date_2 and name_1 == name_2 and bodyweight_1 == bodyweight_2 and competition_1 == competition_2:
                if event_1 == "S":
                    if event_2 == "B":
                        openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True
                        openpowerlifting_USAPL_lifters.at[i, "Event"] = "SB"
                        for k in ["Bench1Kg", "Bench2Kg", "Bench3Kg", "Bench4Kg", "Best3BenchKg"]:
                            openpowerlifting_USAPL_lifters.at[i, k] = openpowerlifting_USAPL_lifters.at[j, k]
                            if pd.notna(openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]:
                                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]
                                if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                                    a, b, c, d, e, f = male_wilks_coefficients()
                                    bw = bodyweight_1

```

```

    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
    else:
        a, b, c, d, e, f = \
female_wilks_coefficients()
        bw = bodyweight_1
        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)

    elif event_2 == "D":
        openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True
        openpowerlifting_USAPL_lifters.at[i, "Event"] = "SD"
        for k in ["Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Deadlift4Kg", "Best3DeadliftKg"]:
            openpowerlifting_USAPL_lifters.at[i, k] = \
openpowerlifting_USAPL_lifters.at[j, k]
            if pd.notna(openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]:
                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]
                if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                    a, b, c, d, e, f = male_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                else:
                    a, b, c, d, e, f = \
female_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \

```

```

                                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)

                                elif event_2 == "S" and bestsquat_1 == bestsquat_2:
                                openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True

                                elif event_1 == "B":
                                if event_2 == "S":
                                openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True
                                openpowerlifting_USAPL_lifters.at[i, "Event"] = "SB"
                                for k in ["Squat1Kg", "Squat2Kg", "Squat3Kg", "Squat4Kg", "Best3SquatKg"]:
                                openpowerlifting_USAPL_lifters.at[i, k] = openpowerlifting_USAPL_lifters.at[j, k]
                                if pd.notna(openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]:
                                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]
                                if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                                a, b, c, d, e, f = male_wilks_coefficients()
                                bw = bodyweight_1
                                openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                                else:
                                a, b, c, d, e, f = female_wilks_coefficients()
                                bw = bodyweight_1
                                openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)

                                elif event_2 == "D":
                                openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True
                                openpowerlifting_USAPL_lifters.at[i, "Event"] = "BD"

```

```

        for k in ["Deadlift1Kg", "Deadlift2Kg", □
→"Deadlift3Kg", "Deadlift4Kg", "Best3DeadliftKg"]:
            openpowerlifting_USAPL_lifters.at[i, k] = □
→openpowerlifting_USAPL_lifters.at[j, k]
            if pd.notna(openpowerlifting_USAPL_lifters.at[i, □
→"Best3DeadliftKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, □
→"Best3DeadliftKg"]:
                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] □
→+= openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]
                if openpowerlifting_USAPL_lifters.at[i, "Sex"] □
→== "M":
                    a, b, c, d, e, f = male_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, □
→"Wilks"] = \
                        openpowerlifting_USAPL_lifters.at[i, □
→"TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + □
→f*bw**5)
                else:
                    a, b, c, d, e, f = □
→female_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, □
→"Wilks"] = \
                        openpowerlifting_USAPL_lifters.at[i, □
→"TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + □
→f*bw**5)

            elif event_2 == "B" and bestbench_1 == bestbench_2:
                openpowerlifting_USAPL_lifters.at[j, □
→"Repeated_meet"] = True

            elif event_1 == "D":
                if event_2 == "S":
                    openpowerlifting_USAPL_lifters.at[j, □
→"Repeated_meet"] = True
                    openpowerlifting_USAPL_lifters.at[i, "Event"] = "SD"
                    for k in ["Squat1Kg", "Squat2Kg", "Squat3Kg", □
→"Squat4Kg", "Best3SquatKg"]:
                        openpowerlifting_USAPL_lifters.at[i, k] = □
→openpowerlifting_USAPL_lifters.at[j, k]
                        if pd.notna(openpowerlifting_USAPL_lifters.at[i, □
→"Best3SquatKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, □
→"Best3SquatKg"]:

```

```

        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] =
←+= openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]
                if openpowerlifting_USAPL_lifters.at[i, "Sex"] =
←== "M":
                    a, b, c, d, e, f = male_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i,
←"Wilks"] = \
                        openpowerlifting_USAPL_lifters.at[i,
←"TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 +
←f*bw**5)
                    else:
                        a, b, c, d, e, f =
←female_wilks_coefficients()
                        bw = bodyweight_1
                        openpowerlifting_USAPL_lifters.at[i,
←"Wilks"] = \
                            openpowerlifting_USAPL_lifters.at[i,
←"TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 +
←f*bw**5)

            elif event_2 == "B":
                openpowerlifting_USAPL_lifters.at[j,
←"Repeated_meet"] = True
                openpowerlifting_USAPL_lifters.at[i, "Event"] = "BD"
                for k in ["Bench1Kg", "Bench2Kg", "Bench3Kg",
←"Bench4Kg", "Best3BenchKg"]:
                    openpowerlifting_USAPL_lifters.at[i, k] =
←openpowerlifting_USAPL_lifters.at[j, k]
                    if pd.notna(openpowerlifting_USAPL_lifters.at[i,
←"Best3BenchKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i,
←"Best3BenchKg"]:
                        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] =
←+= openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]
                        if openpowerlifting_USAPL_lifters.at[i, "Sex"] =
←== "M":
                            a, b, c, d, e, f = male_wilks_coefficients()
                            bw = bodyweight_1
                            openpowerlifting_USAPL_lifters.at[i,
←"Wilks"] = \
                                openpowerlifting_USAPL_lifters.at[i,
←"TotalKg"] * 500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                            else:
                                a, b, c, d, e, f =
←female_wilks_coefficients()

```

```

        bw = bodyweight_1
        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * 500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)

        elif event_2 == "D" and bestdead_1 == bestdead_2:
            openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True

starting_index = finishing_index

```

[]: truncate_meets()
openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Repeated_meet"] == True]

Removed 1349 rows

[97]: openpowerlifting_USAPL_lifters =
openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Repeated_meet"] == False]
openpowerlifting_USAPL_lifters.
to_csv("open_powerlifting_USAPL_Raw_active_lifters.csv", index = False)

[99]: openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == "Zoe Haynes"]

	Name	Sex	Event	Equipment	Age	AgeClass	BirthYearClass	\
202695	Zoe Haynes	F	BD	Raw	23.0	20-23	NaN	
202697	Zoe Haynes	F	SBD	Raw	46.5	45-49	40-49	
	Division	BodyweightKg	WeightClassKg	Squat1Kg	Squat2Kg	Squat3Kg	\	
202695	F_JR_AWPC	59.3	60	NaN	NaN	NaN		
202697	FR-0	61.4	63	115.0	120.0	-127.5		
	Squat4Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	Bench4Kg	\	
202695	NaN	NaN	60.0	-67.5	-67.5	NaN		
202697	NaN	120.0	62.5	65.0	-70.0	NaN		
	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Deadlift4Kg	\		
202695	60.0	132.5	-142.5	-143.0	NaN			
202697	65.0	142.5	150.0	157.5	NaN			

	Best3DeadliftKg	TotalKg	Place	Dots	Wilks	Glossbrenner	\
202695	132.5	192.5	1	147.98	216.579666	131.74	
202697	157.5	342.5	2	374.24	375.120000	331.34	

	Goodlift	Tested	Country	State	Federation	ParentFederation	Date	\
202695	NaN	Yes	USA	NaN	WPC	WPC	2016-08-06	
202697	76.21	Yes	USA	MI	USAPL	IPF	2016-11-05	

	MeetCountry	MeetState	MeetTown	\
202695	USA	MI	Grand Rapids	
202697	USA	MI	NaN	

	MeetName	Repeated_meet	\
202695	Single Lift Championships	False	
202697	Michigan Powerlifting and BP State Championships	False	

	Two_meets_on_one_day	DateOfBirthInterval
202695	True	[1992-08-07, 1993-08-06]
202697	False	[1969-01-01, 1969-12-31]

2 Identifying lifters who have intervals/ages that do not overlap, or have remaining meets that share the same date

```
[148]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json

openpowerlifting_USAPL_lifters = pd.
    ↪read_csv("open_powerlifting_USAPL_Raw_active_lifters.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    ↪to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
```

```
[149]: openpowerlifting_USAPL_lifters
```

	Name	Sex	Event	Equipment	Age	AgeClass	BirthYearClass	\
0	A'Dren Hye	M	SBD	Raw	21.5	20-23	19-23	
1	A'Dren Hye	M	SBD	Wraps	21.5	20-23	19-23	
2	A'Dren Hye	M	SBD	Raw	22.5	20-23	19-23	
3	A'Dren Hye	M	SBD	Raw	23.5	24-34	24-39	
4	A.C. Alexander	M	BD	Raw	48.0	45-49	40-49	
...	
201419	Zuleyka Weaver	F	SBD	Raw	29.5	24-34	24-39	
201420	Zwick Ivan	M	B	Raw	77.5	75-79	70-999	
201421	Zyler Greene	M	SBD	Raw	15.5	16-17	14-18	

201422	Zyler Greene	M	B	Raw	15.5	16-17		14-18
201423	Zyler Greene	M	SBD	Raw	16.5	16-17		14-18
0		Division	BodyweightKg	WeightClassKg	Squat1Kg	Squat2Kg	\	
1	Amateur Juniors	MR-Jr	92.00	93	NaN	NaN		
2		20-23	99.16	100	NaN	NaN		
3		MR-O	91.10	93	-240.0	260.0		
4		MR-O	104.10	105	-260.0	272.5		
...	Law-Fire	48-55	100.00	100	NaN	NaN		
201419		FR-O	51.20	52	77.5	82.5		
201420		MR-M4b	55.60	59	NaN	NaN		
201421		MR-T1	94.60	105	-165.0	165.0		
201422		MR-T1	98.85	105	NaN	NaN		
201423		MR-T2	107.20	120	167.5	180.0		
0	Squat3Kg	Squat4Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	\	
1	NaN	NaN	245.00	NaN	NaN	NaN		
2	272.5	NaN	274.42	NaN	NaN	NaN		
3	272.5	NaN	272.50	155.0	-165.0	-165.0		
4	287.5	NaN	287.50	177.5	185.0	195.0		
...		
201419	-87.5	NaN	82.50	45.0	50.0	-52.5		
201420	NaN	NaN	NaN	25.0	42.5	47.5		
201421	-175.0	NaN	165.00	102.5	-107.5	-107.5		
201422	NaN	NaN	NaN	-105.0	107.5	112.5		
201423	187.5	NaN	187.50	135.0	140.0	-145.0		
0	Bench4Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	\		
1	NaN	165.00	NaN	NaN	NaN			
2	NaN	185.97	NaN	NaN	NaN			
3	NaN	155.00	-280.0	280.0	-282.5			
4	NaN	195.00	260.0	270.0	275.0			
...			
201419	NaN	50.00	122.5	127.5	135.0			
201420	NaN	47.50	NaN	NaN	NaN			
201421	NaN	102.50	150.0	170.0	-182.5			
201422	NaN	112.50	NaN	NaN	NaN			
201423	NaN	140.00	170.0	182.5	190.0			
0	Deadlift4Kg	Best3DeadliftKg	TotalKg	Place	Dots	Wilks	\	
1	NaN	257.50	667.50	1	426.93	421.50		
2	NaN	288.03	748.43	1	462.37	457.04		
3	NaN	280.00	707.50	1	454.70	448.92		
4	NaN	275.00	757.50	1	458.44	454.03		

4	NaN	187.50	325.00	1	200.04	197.79
...
201419	NaN	135.00	267.50	3	329.61	337.48
201420	NaN	NaN	47.50	1	42.79	43.55
201421	NaN	170.00	437.50	1	276.11	272.66
201422	NaN	NaN	112.50	1	69.60	68.79
201423	NaN	190.00	517.50	1	309.54	307.06

	Glossbrenner	Goodlift	Tested	Country	State	Federation	\
0	403.59	87.79	Yes	USA	NaN	USAPL	
1	436.59	94.93	Yes	USA	NaN	RPS	
2	430.02	93.50	Yes	USA	NC	USAPL	
3	433.54	93.90	Yes	USA	NC	USAPL	
4	188.91	NaN	Yes	USA	WI	WABDL	
...	
201419	299.33	68.46	Yes	USA	AZ	USAPL	
201420	42.71	29.63	Yes	USA	IL	USAPL	
201421	260.80	56.76	Yes	USA	KY	USAPL	
201422	65.71	51.82	Yes	USA	KY	USAPL	
201423	293.29	63.28	Yes	USA	KY	USAPL	

	ParentFederation	Date	MeetCountry	MeetState	MeetTown	\
0	IPF	2015-03-21	USA	NC	NaN	
1	NaN	2015-12-12	USA	NC	Fayetteville	
2	IPF	2016-06-04	USA	NC	NaN	
3	IPF	2017-04-08	USA	SC	NaN	
4	NaN	2018-04-21	USA	WI	NaN	
...	
201419	IPF	2018-08-11	USA	AZ	NaN	
201420	IPF	2017-05-06	USA	IL	NaN	
201421	IPF	2020-07-18	USA	OH	NaN	
201422	IPF	2020-08-29	USA	OH	NaN	
201423	IPF	2021-04-17	USA	OH	NaN	

	MeetName	Repeated_meet	\
0	Battle on the Border IX	False	
1	2nd Annual Holiday Havok	False	
2	North Carolina State Championships	False	
3	Battle on the Border	False	
4	World Cup BP & DL	False	
...	
201419	Southwest Regional Championships	False	
201420	Illinois State Meet	False	
201421	PowerMania MMXX	False	
201422	Ohio Open	False	
201423	PowerMania MMXXI	False	

```

Two_meets_on_one_day      DateOfBirthInterval
0                         False   [1993-01-01, 1993-12-31]
1                         False   [1993-01-01, 1993-12-31]
2                         False   [1993-01-01, 1993-12-31]
3                         False   [1993-01-01, 1993-12-31]
4                         False   [1969-04-22, 1970-04-21]
...
201419                    ...
201420                    False   [1988-01-01, 1988-12-31]
201421                    False   [1939-01-01, 1939-12-31]
201422                    False   [2004-01-01, 2004-12-31]
201423                    False   [2004-01-01, 2004-12-31]

```

[201424 rows x 44 columns]

[]: #reconverting date of birth interval from list to interval

```

for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open" ↴
    ↪and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        print(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] [1:-1]. ↪
        ↪split(", "))
        list_date = openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] [1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd. ↪
        ↪Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed = ↪
        ↪"both")

```

[]: # function to store lifters names in a list, where the lifter has DOB intervals ↴
 ↪which do not align,
or contains an "Open" meet, or where DOB intervals are empty

```

starting_index = 0
finishing_index = 0
can_compare_ages = False
names = []
for name in openpowerlifting_USAPL_lifters["Name"].drop_duplicates():
    print(name)
    name_indices = ↪
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↪
    ↪name].index
    num_records = len(name_indices)
    for i in range(num_records):
        if len(names) > 0 and names[-1] == name:
            break

```

```

    DOB_1 = openpowerlifting_USAPL_lifters.at[name_indices[i], u
↪"DateOfBirthInterval"]
    age_1 = openpowerlifting_USAPL_lifters.at[name_indices[i], "Age"]
    date_1 = openpowerlifting_USAPL_lifters.at[name_indices[i], "Date"]
    if (type(age_1) == int) or (type(age_1) == float):
        age_1 = int(age_1)
        can_compare_ages = True
    else:
        can_compare_ages = False

    if type(DOB_1) != pd._libs.interval.Interval:
        names.append(name)
        print(1)
        break

    if (i+1) == num_records:
        break

    for j in range(i+1, num_records):
        DOB_2 = openpowerlifting_USAPL_lifters.at[name_indices[j], u
↪"DateOfBirthInterval"]
        age_2 = openpowerlifting_USAPL_lifters.at[name_indices[j], "Age"]
        date_2 = openpowerlifting_USAPL_lifters.at[name_indices[j], "Date"]

        if (type(age_2) == int) or (type(age_2) == float) and u
↪can_compare_ages == True:
            age_2 = int(age_2)
            if age_2 < age_1: # meets organised in date order, therefore u
↪age_2 should be equal to or more than age 1
                names.append(name)
                print(2)
                break

            if type(DOB_2) != pd._libs.interval.Interval:
                names.append(name)
                print(3)
                break

            if not DOB_2.overlaps(DOB_1): # check if date of birth intervals u
↪overlap or not
                names.append(name)
                print(4)
                break

        if date_1 == date_2:
            names.append(name)

```

```
    print(5)
    break
```

```
[152]: np.save("name disambiguation/duplicatenames", names)
[153]: np.save("name disambiguation/remainingnames", names)
[154]: np.save("name disambiguation/new_lifter_names", np.array([]))
[155]: openpowerlifting_USAPL_lifters.
        ↪to_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv", index_
        ↪= False)
```

3 Iterating through the name list to clean up/segment names

```
[156]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json
pd.set_option("display.max_columns", None)

openpowerlifting_USAPL_lifters = pd.
    ↪read_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    ↪to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')

#reconverting date of birth interval from list to interval
for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open"|
    ↪and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        list_date = openpowerlifting_USAPL_lifters.at[i,|
        ↪"DateOfBirthInterval"][1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd.
            ↪Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed =
            ↪"both")
```

```
[157]: original_dataset_names = {}
for i in pd.read_csv("open_powerlifting_original_data.csv")["Name"].
    ↪drop_duplicates():
    a = i.split(" #")
```

```

if len(a) == 3:
    print(a)
if len(a) == 0:
    print(a)
if len(a) == 2 and a[0] not in original_dataset_names:
    original_dataset_names[a[0]] = [int(a[1])]
elif len(a) == 2 and a[0] in original_dataset_names:
    if original_dataset_names[a[0]] == None:
        original_dataset_names[a[0]] = [0, int(a[1])]
    else:
        original_dataset_names[a[0]].append(int(a[1]))
elif len(a) == 1 and a[0] in original_dataset_names:
    if original_dataset_names[a[0]] == None:
        print("This should not be the case")
    else:
        original_dataset_names[a[0]].append(0)
elif len(a) == 1 and a[0] not in original_dataset_names:
    original_dataset_names[a[0]] = None

```

```
C:\Users\Montel\anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3146: DtypeWarning: Columns (33,35,38)
have mixed types.Specify dtype option on import or set low_memory=False.
has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

```
[158]: for i in original_dataset_names.keys():
    if type(original_dataset_names[i]) == list:
        original_dataset_names[i].sort()
```

Over 6000 lifters to check through

```
[161]: original_dataset_names_json = json.dumps(original_dataset_names)
jsonFile = open("name_disambiguation/original_dataset_names.json", "w")
jsonFile.write(original_dataset_names_json)
jsonFile.close()

new_dataset_names_json = json.dumps(original_dataset_names)
jsonFile = open("name_disambiguation/new_dataset_names.json", "w")
jsonFile.write(new_dataset_names_json)
jsonFile.close()
```

4 algorithm

for every name in the remaining names list 1. lookup the openpowerlifting dataset for each name and show the rows belonging to the name. 2. prompt the user to type in the row indicies that correspond to one person, separated by spaces. let the person type “q” to quit, or “f” to finish the person, or “l” to save them for later. 3. If a list is entered with commas separated, Then, the row indicies in the openpowerlifting table corresponding to the name will be updated (the name column)

with the “name” + “#”+str(i), where i corresponds to the ith disambiguation of the individual.
 4. The lifter’s name + number will then be added to “new lifter names”. The dataframe will then be segmented by the lifter name + number and this lifter segmentation will be saved to a csv file. The openpowerlifting dataset will also be resaved to csv with the updated names.

```
[1]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json
import copy
import random
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', 500)

openpowerlifting_USAPL_lifters = pd.
    ↪read_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    ↪to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')

#reconverting date of birth interval from list to interval
for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open" ↪
        ↪and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        list_date = openpowerlifting_USAPL_lifters.at[i, ↪
            ↪"DateOfBirthInterval"][1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd.
            ↪Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed = ↪
            ↪"both")
```

```
[85]: def lifter_name_disambiguation():
    name_list = np.load("name disambiguation/remainingnames.npy")
    name_list = np.sort(name_list)#[::-1]
    new_names = np.load("name disambiguation/new_lifter_names.npy")
    original_dataset_names = json.load(open("name disambiguation/
        ↪original_dataset_names.json"))
    new_dataset_names = json.load(open("name disambiguation/new_dataset_names.
        ↪json"))
    name_list_in_progress = copy.deepcopy(name_list)
    counter = 0
    end_loop = False
```

```

#     slice_list = ["Name", "Federation", "Event", "Date", "MeetCountry", "Event", "Division", "Equipment", "BodyweightKg", "Age", "AgeClass", "BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", "DateOfBirthInterval"]
#     for name in name_list:
#         if end_loop == True:
#             break
#         lifter_in_progress = True
#         print(f"{len(name_list_in_progress)} names remaining")
#         print(name)
#         if "#" in name:
#             ambiguous_name = name.split(" #")[0]
#             lifter_in_progress = True
#             number_disambiguation = 1
#             df = openpowerlifting_USAPL_lifters[[["Name", "Federation", "Event", "Date", "MeetCountry", "Event", "Division", "Equipment", "BodyweightKg", "Age", "AgeClass", "BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", "DateOfBirthInterval"]]]
#             original_index = df.index[0]
#             reset_df = df.set_axis(labels = [b - original_index for b in df.index])
#             print("original_index = "+str(original_index))
#             while lifter_in_progress == True:
#                 if len(reset_df.loc[(reset_df["Federation"] == "USAPL") & (pd.notna(reset_df["Age"])) & (reset_df["Equipment"] == "Raw") & (reset_df["Event"] == "SBD") & (reset_df["Date"] > pd.to_datetime("01/01/2014"))]) == 0:
#                     next_step = "N"
#                     elif reset_df["Age"].count() == 0:
#                         next_step = "N"
#
#                     if len(openpowerlifting_USAPL_lifters[[["Name", "Federation", "Event", "Date", "MeetCountry", "Event", "Division", "Equipment", "BodyweightKg", "Age", "AgeClass", "BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", "DateOfBirthInterval"]]]) > 120:

```

```

#display(reset_df[["Name", "Federation", "Event", "Date",
                   "MeetCountry", "Event", "Division", "Equipment", "BodyweightKg", "Age",
                   "AgeClass", "BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg",
                   "Best3SquatKg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg",
                   "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg",
                   "DateOfBirthInterval"]])

    #next_step = input("""Can this lifter be further
    ↵disambiguated? Press Y if they can, press N if they cannot.
    ↵Type 'Q' to quit, or 'L' to save the lifter for later: """)
    successful_loop = False
    next_step = "l"

    elif ↵
        ↵len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==
        ↵name) > 1:
            display(reset_df[["Name", "Federation", "Event", "Date",
                               "MeetCountry", "Event", "Division", "Equipment", "BodyweightKg", "Age",
                               "AgeClass", "BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg",
                               "Best3SquatKg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg",
                               "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg",
                               "DateOfBirthInterval"]])
            next_step = input("""Can this lifter be further
            ↵disambiguated? Press Y if they can, press N if they cannot.
            ↵Type 'Q' to quit, or 'L' to save the lifter for later: """)
            successful_loop = False

    else:
        next_step = "n"

try:
    if next_step.upper() == "Q":
        jsonFile = open("name_disambiguation/new_dataset_names.
        ↵json", "w")
        jsonFile.write(json.dumps(new_dataset_names))
        jsonFile.close()
        openpowerlifting_USAPL_lifters.
        ↵to_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv", index_
        ↵= False)
        np.save("name_disambiguation/remainingnames", ↵
        ↵name_list_in_progress)
        np.save("name_disambiguation/new_lifter_names", ↵
        ↵new_names)

```

```

        lifter_in_progress = False
        end_loop = True
        break
    elif next_step.upper() == "N":
        ↵
        ↵openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == name].to_csv(
            f"Name Disambiguation/{name}.csv", index = False)
        name_list_in_progress = []
        ↵name_list_in_progress[name_list_in_progress != name]
        np.save("name disambiguation/remainingnames", np.array(name_list_in_progress))
        lifter_in_progress = False
        break

    elif next_step.upper() == "L":
        name_list_in_progress = []
        ↵name_list_in_progress[name_list_in_progress != name]
        name_list_in_progress = np.array(
            np.append(name_list_in_progress, name))
        np.save("name disambiguation/remainingnames", np.array(name_list_in_progress))
        lifter_in_progress = False
        break

    elif next_step.upper() == "Y":
        while not successful_loop:
            if openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == name].shape[0] == 2:
                separation = [
                    openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == name].index[1]]
                while number_disambiguation in new_dataset_names[ambiguous_name]:
                    number_disambiguation += 1
                    for number in separation:
                        openpowerlifting_USAPL_lifters.at[int(number), "Name"] = f"{ambiguous_name} #{number_disambiguation}"
                openpowerlifting_USAPL_lifters.iloc[separation].to_csv(
                    f"Name Disambiguation/{ambiguous_name}#{number_disambiguation}.csv", index = False)
                    name_list_in_progress = []
                    ↵name_list_in_progress[name_list_in_progress != name]

```

```

                new_dataset_names[ambiguous_name] .
↪append[number_disambiguation]
                    new_names = np.append(new_names, ↪
↪f" {ambiguous_name} #{number_disambiguation}" )

                    number_disambiguation += 1
                    successful_loop = True
            else:
                separation = input("Enter the row indicies for ↪
↪this disambiguation, separated by a space: ").split()
                print("New lifter is as shown: ")
                display(
                    reset_df.loc[[int(number) for number in ↪
↪separation]])
                    while number_disambiguation in ↪
↪new_dataset_names[ambiguous_name]:
                        number_disambiguation += 1
                        while True:
                            decision = input("Are you happy with this? ↪
↪Enter Y if yes, or N if no: ")
                            if type(decision) == str and decision.
↪upper() == "Y":
                                reset_df = reset_df.drop(labels = ↪
↪[int(number) for number in separation], axis = 0)
                                separation = [int(original_index) + ↪
↪int(number) for number in separation]
                                for number in separation:
                                    openpowerlifting_USAPL_lifters.
↪at[number, "Name"] = f" {ambiguous_name} #{number_disambiguation}""

                                openpowerlifting_USAPL_lifters.
↪iloc[separation].to_csv(
                                f"Name Disambiguation/
↪{ambiguous_name} #{number_disambiguation}.csv", index = False)

                new_dataset_names[ambiguous_name] .
↪append[number_disambiguation]

                if ↪
↪len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↪
↪name]) <= 1:
                    name_list_in_progress = ↪
↪name_list_in_progress[name_list_in_progress != name]
                    np.save("name disambiguation/
↪remainingnames", name_list_in_progress)

```

```

        new_names = np.append(new_names, u
↳f"{{ambiguous_name}} #{{number_disambiguation}}")

        number_disambiguation += 1
        successful_loop = True
        break

    elif type(decision) == str and decision.
↳upper() == "N":
        break
    else:
        continue

except TypeError:
    continue

else:
    number_disambiguation = 1
    lifter_in_progress = True
    df = u
↳openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == u
↳name][["Name", "Federation", "Event", "Date", "MeetCountry", "Event", u
↳"Division", "Equipment", "BodyweightKg", "Age", "AgeClass", u
↳"BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", u
↳"Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", u
↳"Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", u
↳"DateOfBirthInterval"]]
    original_index = df.index[0]
    reset_df = df.set_axis(labels = [b - original_index for b in df.
↳index])
    print("original_index = "+str(original_index))
    while lifter_in_progress == True:
#
        if len(reset_df.loc[(reset_df["Federation"] == "USAPL") & \
#
#                                (pd.notna(reset_df["Age"])) & \
#                                (reset_df["Equipment"] == "Raw") & \
#                                (reset_df["Event"] == "SBD") & \
#                                (reset_df["Date"] > pd.to_datetime("01/01/
↳2014"))]) == 0:
#
            next_step = "N"

#
        elif reset_df["Age"].count() == 0:
            next_step = "N"

#
        if u
↳len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == u
↳name]) > 120:

```

```

#display(reset_df[["Name", "Federation", "Event", "Date",
→ "MeetCountry", "Event", "Division", "Equipment", "BodyweightKg", "Age",
→ "AgeClass", "BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg",
→ "Best3SquatKg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg",
→ "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg",
→ "DateOfBirthInterval"]])

    next_step = input("""Can this lifter be further
→ disambiguated? Press Y if they can, press N if they cannot.
        #Type 'Q' to quit, or 'L' to save the lifter for later: """)
    successful_loop = False
    next_step = "l"

    elif:
        → len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==
        → name) > 1:
            display(reset_df[["Name", "Federation", "Event", "Date",
→ "MeetCountry", "Event", "Division", "Equipment", "BodyweightKg", "Age",
→ "AgeClass", "BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg",
→ "Best3SquatKg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg",
→ "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg",
→ "DateOfBirthInterval"]])
            next_step = input("""Can this lifter be further
→ disambiguated? Press Y if they can, press N if they cannot.
        Type 'Q' to quit, or 'L' to save the lifter for later: """)
            successful_loop = False

    else:
        next_step = "n"

    try:
        if next_step.upper() == "Q":
            jsonFile = open("name_disambiguation/new_dataset_names.
→ json", "w")
            jsonFile.write(json.dumps(new_dataset_names))
            jsonFile.close()
            openpowerlifting_USAPL_lifters.
→ to_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv", index_
→ = False)
            np.save("name_disambiguation/remainingnames",
→ name_list_in_progress)
            np.save("name_disambiguation/new_lifter_names",
→ new_names)
            lifter_in_progress = False
            end_loop = True
            break

```

```

        elif next_step.upper() == "N":
            ↵openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↵
            ↵name].to_csv(
                f"Name Disambiguation/{name}.csv", index = False)
            name_list_in_progress = ↵
            ↵name_list_in_progress[name_list_in_progress != name]
            np.save("name disambiguation/remainingnames", ↵
            ↵name_list_in_progress)
            lifter_in_progress = False
            break

        elif next_step.upper() == "L":
            name_list_in_progress = ↵
            ↵name_list_in_progress[name_list_in_progress != name]
            name_list_in_progress = np.
            ↵append(name_list_in_progress, name)
            np.save("name disambiguation/remainingnames", name_list)

            lifter_in_progress = False
            break

        elif next_step.upper() == "Y":
            while not successful_loop:
                if ↵
                    ↵openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↵
                    ↵name].shape[0] == 2:
                        separation = ↵
                        ↵[openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↵
                        ↵name].index[1]]
                        while ↵
                            ↵len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↵
                            ↵\

                            ↵f"{name} #{number_disambiguation}") > 0:
                                number_disambiguation += 1
                                for number in separation:
                                    openpowerlifting_USAPL_lifters.
                                    ↵at[int(number), "Name"] = f"{name} #{number_disambiguation}"

                                openpowerlifting_USAPL_lifters.iloc[separation].
                                ↵to_csv(
                                    f"Name Disambiguation/{name}" ↵
                                    ↵#{number_disambiguation}.csv", index = False)

```

```

                name_list_in_progress =_
↪name_list_in_progress[name_list_in_progress != name]

                if type(new_dataset_names[name]) == list:
                    new_dataset_names[name] .

↪append[number_disambiguation]
                else:
                    new_dataset_names[name] = [1]

                new_names = np.append(new_names, (f"{name}"_
↪#{number_disambiguation}))
```

successful_loop = **True**
number_disambiguation += 1

```

                else:
                    separation = input("Enter the row indicies for_
↪this disambiguation, separated by a space: ").split()
                    print("New lifter is as shown: ")
                    display(
                        reset_df.loc[[int(number) for number in_
↪separation]])
                    while_
↪len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] ==_
↪\
↪f"{name} #{number_disambiguation}"]) > 0:
                        number_disambiguation += 1
```

□

```

                    while True:
                        decision = input("Are you happy with this?_
↪Enter Y if yes, or N if no: ")
                        if type(decision) == str and decision.
↪upper() == "Y":
                            reset_df = reset_df.drop(labels =_
↪[int(number) for number in separation], axis = 0)
                            separation = [int(original_index) +_
↪int(number) for number in separation]
                            for number in separation:
                                openpowerlifting_USAPL_lifters.
↪at[int(number), "Name"] = f"{name} #{number_disambiguation}"

                            openpowerlifting_USAPL_lifters.
↪iloc[separation].to_csv(
                                f"Name Disambiguation/{name}"_
↪#{number_disambiguation}.csv", index = False)
```

```

        if ↴
↳len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] == ↴
↳name]) <= 1:
                                name_list_in_progress = ↴
↳name_list_in_progress[name_list_in_progress != name]
                                np.save("name disambiguation/
↳remainingnames", name_list_in_progress)

                if type(new_dataset_names[name]) == ↴
↳list:
                    new_dataset_names[name] =
↳append[number_disambiguation]
                else:
                    new_dataset_names[name] = [1]

                new_names = np.append(new_names, ↴
↳(f"{name} #{number_disambiguation}))}

                successful_loop = True
                number_disambiguation += 1
                break

            elif type(decision) == str and decision.
↳upper() == "N":
                break
            else:
                continue

        except TypeError:
            continue
    openpowerlifting_USAPL_lifters.
↳to_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv", index_
↳= False)
    np.save("name disambiguation/remainingnames", name_list_in_progress)
    np.save("name disambiguation/new_lifter_names", new_names)

```

```
[36]: def lifter_name_disambiguation():
    name_list = np.load("name disambiguation/remainingnames.npy")
    new_names = np.load("name disambiguation/new_lifter_names.npy")
    original_dataset_names = json.load(open("name disambiguation/
↳original_dataset_names.json"))
    new_dataset_names = json.load(open("name disambiguation/new_dataset_names.
↳json"))
    #name_list = copy.deepcopy(name_list)
```

```

end_loop = False
while len(name_list) != 0:
    if end_loop == True:
        break
    lifter_in_progress = True
    print(f"{len(name_list)} names remaining")
    random.shuffle(name_list)
    name = random.choice(name_list)
    print(name)
    if "#" in name:
        ambiguous_name = name.split(" #")[0]
        lifter_in_progress = True
        number_disambiguation = 1
        df = openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == name][["Name", "Federation", "Date", "MeetCountry", "Event", "Division", "Equipment", "BodyweightKg", "Age", "AgeClass", "BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", "DateOfBirthInterval"]]
        original_index = df.index[0]
        reset_df = df.set_axis(labels = [b - original_index for b in df.index])
        print("original_index = "+str(original_index))
        while lifter_in_progress == True:
            if len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == name]) > 1:
                display(reset_df)
                next_step = input("""Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.
Type 'Q' to quit, or 'L' to save the lifter for later: """)
                successful_loop = False

            else:
                next_step = "N"

            try:
                if next_step.upper() == "Q":
                    jsonFile = open("name_disambiguation/new_dataset_names.json", "w")
                    jsonFile.write(json.dumps(new_dataset_names))
                    jsonFile.close()
                    openpowerlifting_USAPL_lifters.to_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv", index=False)

```

```

        np.save("name_disambiguation/remainingnames", name_list)
        np.save("name_disambiguation/new_lifter_names", ↴
new_names)

        lifter_in_progress = False
        end_loop = True
        break

    elif next_step.upper() == "N":
        ↴
→openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↴
→name].to_csv(
            f"Name_Disambiguation/{name}.csv", index = False)
        name_list = name_list[name_list != name]
        np.save("name_disambiguation/remainingnames", name_list)
        lifter_in_progress = False
        break

    elif next_step.upper() == "L":
        lifter_in_progress = False
        break

    elif next_step.upper() == "Y":
        while not successful_loop:
            if ↴
→openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↴
→name].shape[0] == 2:
                separation = ↴
→[openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↴
→name].index[1]]
                while number_disambiguation in ↴
→new_dataset_names[ambiguous_name]:
                    number_disambiguation += 1
                    for number in separation:
                        openpowerlifting_USAPL_lifters.
→at[int(number), "Name"] = f"{ambiguous_name} #{number_disambiguation}"

                    openpowerlifting_USAPL_lifters.iloc[separation].
→to_csv(
                    f"Name_Disambiguation/{ambiguous_name} ↴
→#{number_disambiguation}.csv", index = False)
                    name_list = name_list[name_list != name]
                    new_dataset_names[ambiguous_name].
→append[number_disambiguation]
                    new_names = np.append(new_names, ↴
→f"{ambiguous_name} #{number_disambiguation}")

                    number_disambiguation += 1

```

```

                successful_loop = True
            else:
                separation = input("Enter the row indicies for\u
→this disambiguation, separated by a space: ").split()
                print("New lifter is as shown: ")
                display(
                    reset_df.loc[[int(number) for number in\u
→separation]])
                while number_disambiguation in\u
→new_dataset_names[ambiguous_name]:
                    number_disambiguation += 1
                    while True:
                        decision = input("Are you happy with this?\u
→Enter Y if yes, or N if no: ")
                        if type(decision) == str and decision.
→upper() == "Y":
                            reset_df = reset_df.drop(labels =\u
→[int(number) for number in separation], axis = 0)
                            separation = [int(original_index) +\u
→int(number) for number in separation]
                            for number in separation:
                                openpowerlifting_USAPL_lifters.
→at[number, "Name"] = f"{ambiguous_name} #{number_disambiguation}"

                            openpowerlifting_USAPL_lifters.
→iloc[separation].to_csv(
                                f"Name Disambiguation/\u
→{ambiguous_name} #{number_disambiguation}.csv", index = False)

                            new_dataset_names[ambiguous_name].
→append(number_disambiguation)

                            if\u
→len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] ==\u
→name]) <= 1:
                                name_list = name_list[name_list !=\u
→name]
                                np.save("name disambiguation/\u
→remainingnames", name_list)

                                new_names = np.append(new_names,\u
→f"{ambiguous_name} #{number_disambiguation}")

                                number_disambiguation += 1
                                successful_loop = True
                                break

```

```

                elif type(decision) == str and decision.
upper() == "N":
                    break
                else:
                    continue

            except TypeError:
                continue

        else:
            number_disambiguation = 1
            lifter_in_progress = True
            df =
openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] ==
name][["Name", "Federation", "Date", "MeetCountry", "Event", "Division",
"Equipment", "BodyweightKg", "Age", "AgeClass", "BirthYearClass",
"Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", "Bench1Kg", "Bench2Kg",
"Bench3Kg", "Best3BenchKg", "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg",
"Best3DeadliftKg", "TotalKg", "DateOfBirthInterval"]]
            original_index = df.index[0]
            reset_df = df.set_axis(labels = [b - original_index for b in df.
index])
            print("original_index = "+str(original_index))
            while lifter_in_progress == True:

                if
len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] ==
name]) > 1:
                    display(reset_df)
                    next_step = input("""Can this lifter be further
disambiguated? Press Y if they can, press N if they cannot.
Type 'Q' to quit, or 'L' to save the lifter for later: """)
                    successful_loop = False

                else:
                    next_step = "N"

            try:
                if next_step.upper() == "Q":
                    jsonFile = open("name disambiguation/new_dataset_names.
json", "w")
                    jsonFile.write(json.dumps(new_dataset_names))
                    jsonFile.close()

```

```

        openpowerlifting_USAPL_lifters.

→to_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv", index=False)
→= False)

        np.save("name disambiguation/remainingnames", name_list)
        np.save("name disambiguation/new_lifter_names", ↵
new_names)

        lifter_in_progress = False
        end_loop = True
        break

    elif next_step.upper() == "N":
        ↵
→openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↵
→name].to_csv(
            f"Name Disambiguation/{name}.csv", index = False)
        name_list = name_list[name_list != name]
        np.save("name disambiguation/remainingnames", name_list)
        lifter_in_progress = False
        break

    elif next_step.upper() == "L":
        np.save("name disambiguation/remainingnames", name_list)

        lifter_in_progress = False
        break

    elif next_step.upper() == "Y":
        while not successful_loop:
            if ↵
→openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↵
→name].shape[0] == 2:
                separation = ↵
→[openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↵
→name].index[1]]
                while ↵
→len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↵
→\

→f"{name} #{number_disambiguation}"] > 0:
                    number_disambiguation += 1
                    for number in separation:
                        openpowerlifting_USAPL_lifters.
→at[int(number), "Name"] = f"{name} #{number_disambiguation}"

                    openpowerlifting_USAPL_lifters.iloc[separation].
→to_csv(

```

```

f"Name Disambiguation/{name}"]
↪#{"number_disambiguation}.csv", index = False)
name_list = name_list[name_list != name]

if type(new_dataset_names[name]) == list:
    new_dataset_names[name] =
↪append[number_disambiguation]
else:
    new_dataset_names[name] = [1]

new_names = np.append(new_names, (f" {name}"]
↪#{"number_disambiguation"}))

successful_loop = True
number_disambiguation += 1

else:
    separation = input("Enter the row indicies for"
↪this disambiguation, separated by a space: ").split()
    print("New lifter is as shown: ")
    display(
        reset_df.loc[[int(number) for number in
↪separation]])
    while
↪len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] ==
↪\
↪f" {name} {"number_disambiguation"}]) > 0:
        number_disambiguation += 1

    while True:
        decision = input("Are you happy with this? "
↪Enter Y if yes, or N if no: ")
        if type(decision) == str and decision.
↪upper() == "Y":
            reset_df = reset_df.drop(labels =
↪[int(number) for number in separation], axis = 0)
            separation = [int(original_index) +
↪int(number) for number in separation]
            for number in separation:
                openpowerlifting_USAPL_lifters.
↪at[int(number), "Name"] = f" {name} {"number_disambiguation"}"

            openpowerlifting_USAPL_lifters.
↪iloc[separation].to_csv(

```

```

f"Name Disambiguation/{name}"]
→#{number_disambiguation}.csv", index = False)

if len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == name]) <= 1:
    name_list = name_list[name_list != name]
    np.save("name disambiguation/
→remainingnames", name_list)

if type(new_dataset_names[name]) == list:
    new_dataset_names[name] =
→append[number_disambiguation]
else:
    new_dataset_names[name] = [1]

new_names = np.append(new_names, [
→(f"{name} #{number_disambiguation}")])

successful_loop = True
number_disambiguation += 1
break

elif type(decision) == str and decision.
→upper() == "N":
    break
else:
    continue

except TypeError:
    continue

```

[87]: lifter_name_disambiguation()

3010 names remaining
Alexandra García
original_index = 6453

	Name	Federation	Event	Event	Date	MeetCountry	Event	\
0	Alexandra García	THSWPA	SBD	SBD	2014-01-11	USA	SBD	
1	Alexandra García	THSWPA	SBD	SBD	2014-01-18	USA	SBD	
2	Alexandra García	THSWPA	SBD	SBD	2014-01-25	USA	SBD	
3	Alexandra García	THSWPA	SBD	SBD	2014-02-01	USA	SBD	
4	Alexandra García	THSWPA	SBD	SBD	2014-02-10	USA	SBD	
5	Alexandra García	THSWPA	SBD	SBD	2014-03-01	USA	SBD	

7	Alexandra García	THSWPA	SBD	SBD	2014-03-15	USA	SBD
9	Alexandra García	THSWPA	SBD	SBD	2015-01-10	USA	SBD
10	Alexandra García	THSWPA	SBD	SBD	2015-01-14	USA	SBD
11	Alexandra García	THSWPA	SBD	SBD	2015-01-17	USA	SBD
12	Alexandra García	THSWPA	SBD	SBD	2015-01-24	USA	SBD
13	Alexandra García	THSWPA	SBD	SBD	2015-01-24	USA	SBD
14	Alexandra García	THSWPA	SBD	SBD	2015-01-31	USA	SBD
15	Alexandra García	THSWPA	SBD	SBD	2015-02-05	USA	SBD
16	Alexandra García	THSWPA	SBD	SBD	2015-02-21	USA	SBD
17	Alexandra García	THSWPA	SBD	SBD	2015-02-21	USA	SBD
18	Alexandra García	THSWPA	SBD	SBD	2015-03-07	USA	SBD
19	Alexandra García	THSWPA	SBD	SBD	2015-03-19	USA	SBD
21	Alexandra García	USPA	SBD	SBD	2017-10-28	USA	SBD
22	Alexandra García	USPA	SBD	SBD	2018-02-24	USA	SBD
23	Alexandra García	USPA	SBD	SBD	2018-06-23	USA	SBD
24	Alexandra García	USPA	SBD	SBD	2018-12-01	USA	SBD
25	Alexandra García	USPA	SBD	SBD	2019-04-20	USA	SBD

	Event	Division	Equipment	BodyweightKg	Age	AgeClass	\
0	SBD	Girls	Single-ply	50.80	NaN	13-19	
1	SBD	Girls	Single-ply	50.53	NaN	13-19	
2	SBD	Girls	Single-ply	50.08	NaN	13-19	
3	SBD	Girls	Single-ply	49.90	NaN	13-19	
4	SBD	Girls	Single-ply	51.89	NaN	13-19	
5	SBD	Girls	Single-ply	50.08	NaN	13-19	
7	SBD	Girls	Single-ply	50.08	NaN	13-19	
9	SBD	Girls	Single-ply	51.62	NaN	13-19	
10	SBD	Girls	Single-ply	58.29	NaN	13-19	
11	SBD	Girls	Single-ply	51.35	NaN	13-19	
12	SBD	Girls	Single-ply	51.26	NaN	13-19	
13	SBD	Girls	Single-ply	57.70	NaN	13-19	
14	SBD	Girls	Single-ply	51.35	NaN	13-19	
15	SBD	Girls	Single-ply	60.15	NaN	13-19	
16	SBD	Girls	Single-ply	57.70	NaN	13-19	
17	SBD	Girls	Single-ply	51.44	NaN	13-19	
18	SBD	Div 2	Single-ply	51.17	NaN	16-19	
19	SBD	Girls	Single-ply	51.12	NaN	13-19	
21	SBD	Juniors	16-17	Raw	56.80	16.0	16-17
22	SBD		Open	Raw	55.10	16.0	16-17
23	SBD	Juniors	16-17	Raw	58.40	17.0	16-17
24	SBD	Juniors	16-17	Raw	56.00	17.0	16-17
25	SBD	Juniors	16-17	Raw	61.55	17.0	16-17

	BirthYearClass	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	\
0	13-19	NaN	NaN	NaN	108.86	NaN	
1	13-19	NaN	NaN	NaN	115.67	NaN	
2	13-19	NaN	NaN	NaN	120.20	NaN	
3	13-19	NaN	NaN	NaN	124.74	NaN	

4	13-19	NaN	NaN	NaN	122.47	NaN
5	13-19	NaN	NaN	NaN	113.40	NaN
7	13-19	NaN	NaN	NaN	115.67	NaN
9	13-19	NaN	NaN	NaN	111.13	NaN
10	13-19	NaN	NaN	NaN	72.57	NaN
11	13-19	NaN	NaN	NaN	120.20	NaN
12	13-19	NaN	NaN	NaN	120.20	NaN
13	13-19	NaN	NaN	NaN	79.38	NaN
14	13-19	NaN	NaN	NaN	124.74	NaN
15	13-19	NaN	NaN	NaN	74.84	NaN
16	13-19	NaN	NaN	NaN	83.91	NaN
17	13-19	NaN	NaN	NaN	129.27	NaN
18	16-19	NaN	NaN	NaN	127.01	NaN
19	13-19	NaN	NaN	NaN	129.27	NaN
21	14-18	NaN	NaN	NaN	85.00	NaN
22	14-18	NaN	NaN	NaN	87.50	NaN
23	14-18	NaN	NaN	NaN	93.00	NaN
24	14-18	NaN	NaN	NaN	85.00	NaN
25	14-18	NaN	NaN	NaN	102.50	NaN

	Bench2Kg	Bench3Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	\
0	NaN	NaN	58.97	NaN	NaN	NaN	
1	NaN	NaN	63.50	NaN	NaN	NaN	
2	NaN	NaN	61.23	NaN	NaN	NaN	
3	NaN	NaN	61.23	NaN	NaN	NaN	
4	NaN	NaN	65.77	NaN	NaN	NaN	
5	NaN	NaN	65.77	NaN	NaN	NaN	
7	NaN	NaN	61.23	NaN	NaN	NaN	
9	NaN	NaN	68.04	NaN	NaN	NaN	
10	NaN	NaN	NaN	NaN	NaN	NaN	
11	NaN	NaN	68.04	NaN	NaN	NaN	
12	NaN	NaN	NaN	NaN	NaN	NaN	
13	NaN	NaN	29.48	NaN	NaN	NaN	
14	NaN	NaN	63.50	NaN	NaN	NaN	
15	NaN	NaN	29.48	NaN	NaN	NaN	
16	NaN	NaN	29.48	NaN	NaN	NaN	
17	NaN	NaN	65.77	NaN	NaN	NaN	
18	NaN	NaN	68.04	NaN	NaN	NaN	
19	NaN	NaN	65.77	NaN	NaN	NaN	
21	NaN	NaN	37.50	NaN	NaN	NaN	
22	NaN	NaN	45.00	NaN	NaN	NaN	
23	NaN	NaN	45.00	NaN	NaN	NaN	
24	NaN	NaN	40.00	NaN	NaN	NaN	
25	NaN	NaN	52.50	NaN	NaN	NaN	

	Best3DeadliftKg	TotalKg	DateOfBirthInterval
0	120.20	288.03	[1994-01-11, 2001-01-11]
1	127.01	306.17	[1994-01-18, 2001-01-18]

2	129.27	310.71	[1994-01-25, 2001-01-25]
3	133.81	319.78	[1994-02-01, 2001-02-01]
4	138.35	326.59	[1994-02-10, 2001-02-10]
5	136.08	315.25	[1994-03-01, 2001-03-01]
7	142.88	319.78	[1994-03-15, 2001-03-15]
9	131.54	310.71	[1995-01-10, 2002-01-10]
10	NaN	NaN	[1995-01-14, 2002-01-14]
11	138.35	326.59	[1995-01-17, 2002-01-17]
12	NaN	NaN	[1995-01-24, 2002-01-24]
13	88.45	197.31	[1995-01-24, 2002-01-24]
14	136.08	324.32	[1995-01-31, 2002-01-31]
15	86.18	190.51	[1995-02-05, 2002-02-05]
16	83.91	197.31	[1995-02-21, 2002-02-21]
17	138.35	333.39	[1995-02-21, 2002-02-21]
18	142.88	337.93	[1995-03-07, 1999-03-07]
19	140.61	335.66	[1995-03-19, 2002-03-19]
21	115.00	237.50	[2000-10-29, 2001-10-28]
22	120.00	252.50	[2001-02-25, 2002-02-24]
23	110.00	248.00	[2000-06-24, 2001-06-23]
24	115.00	240.00	[2000-12-02, 2001-12-01]
25	125.00	280.00	[2001-04-21, 2002-04-20]

Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.

Type 'Q' to quit, or 'L' to save the lifter for later: 21 22
23 24 25

	Name	Federation	Event	Event	Date	Meet	Country	Event	\
0	Alexandra García	THSWPA	SBD	SBD	2014-01-11		USA	SBD	
1	Alexandra García	THSWPA	SBD	SBD	2014-01-18		USA	SBD	
2	Alexandra García	THSWPA	SBD	SBD	2014-01-25		USA	SBD	
3	Alexandra García	THSWPA	SBD	SBD	2014-02-01		USA	SBD	
4	Alexandra García	THSWPA	SBD	SBD	2014-02-10		USA	SBD	
5	Alexandra García	THSWPA	SBD	SBD	2014-03-01		USA	SBD	
7	Alexandra García	THSWPA	SBD	SBD	2014-03-15		USA	SBD	
9	Alexandra García	THSWPA	SBD	SBD	2015-01-10		USA	SBD	
10	Alexandra García	THSWPA	SBD	SBD	2015-01-14		USA	SBD	
11	Alexandra García	THSWPA	SBD	SBD	2015-01-17		USA	SBD	
12	Alexandra García	THSWPA	SBD	SBD	2015-01-24		USA	SBD	
13	Alexandra García	THSWPA	SBD	SBD	2015-01-24		USA	SBD	
14	Alexandra García	THSWPA	SBD	SBD	2015-01-31		USA	SBD	
15	Alexandra García	THSWPA	SBD	SBD	2015-02-05		USA	SBD	
16	Alexandra García	THSWPA	SBD	SBD	2015-02-21		USA	SBD	
17	Alexandra García	THSWPA	SBD	SBD	2015-02-21		USA	SBD	
18	Alexandra García	THSWPA	SBD	SBD	2015-03-07		USA	SBD	
19	Alexandra García	THSWPA	SBD	SBD	2015-03-19		USA	SBD	
21	Alexandra García	USPA	SBD	SBD	2017-10-28		USA	SBD	
22	Alexandra García	USPA	SBD	SBD	2018-02-24		USA	SBD	
23	Alexandra García	USPA	SBD	SBD	2018-06-23		USA	SBD	

24	Alexandra García	USPA	SBD	SBD	2018-12-01	USA	SBD
25	Alexandra García	USPA	SBD	SBD	2019-04-20	USA	SBD

	Event	Division	Equipment	BodyweightKg	Age	AgeClass	\
0	SBD	Girls	Single-ply	50.80	NaN	13-19	
1	SBD	Girls	Single-ply	50.53	NaN	13-19	
2	SBD	Girls	Single-ply	50.08	NaN	13-19	
3	SBD	Girls	Single-ply	49.90	NaN	13-19	
4	SBD	Girls	Single-ply	51.89	NaN	13-19	
5	SBD	Girls	Single-ply	50.08	NaN	13-19	
7	SBD	Girls	Single-ply	50.08	NaN	13-19	
9	SBD	Girls	Single-ply	51.62	NaN	13-19	
10	SBD	Girls	Single-ply	58.29	NaN	13-19	
11	SBD	Girls	Single-ply	51.35	NaN	13-19	
12	SBD	Girls	Single-ply	51.26	NaN	13-19	
13	SBD	Girls	Single-ply	57.70	NaN	13-19	
14	SBD	Girls	Single-ply	51.35	NaN	13-19	
15	SBD	Girls	Single-ply	60.15	NaN	13-19	
16	SBD	Girls	Single-ply	57.70	NaN	13-19	
17	SBD	Girls	Single-ply	51.44	NaN	13-19	
18	SBD	Div 2	Single-ply	51.17	NaN	16-19	
19	SBD	Girls	Single-ply	51.12	NaN	13-19	
21	SBD	Juniors	16-17	Raw	56.80	16.0	16-17
22	SBD		Open	Raw	55.10	16.0	16-17
23	SBD	Juniors	16-17	Raw	58.40	17.0	16-17
24	SBD	Juniors	16-17	Raw	56.00	17.0	16-17
25	SBD	Juniors	16-17	Raw	61.55	17.0	16-17

	BirthYearClass	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	\
0	13-19	NaN	NaN	NaN	108.86	NaN	
1	13-19	NaN	NaN	NaN	115.67	NaN	
2	13-19	NaN	NaN	NaN	120.20	NaN	
3	13-19	NaN	NaN	NaN	124.74	NaN	
4	13-19	NaN	NaN	NaN	122.47	NaN	
5	13-19	NaN	NaN	NaN	113.40	NaN	
7	13-19	NaN	NaN	NaN	115.67	NaN	
9	13-19	NaN	NaN	NaN	111.13	NaN	
10	13-19	NaN	NaN	NaN	72.57	NaN	
11	13-19	NaN	NaN	NaN	120.20	NaN	
12	13-19	NaN	NaN	NaN	120.20	NaN	
13	13-19	NaN	NaN	NaN	79.38	NaN	
14	13-19	NaN	NaN	NaN	124.74	NaN	
15	13-19	NaN	NaN	NaN	74.84	NaN	
16	13-19	NaN	NaN	NaN	83.91	NaN	
17	13-19	NaN	NaN	NaN	129.27	NaN	
18	16-19	NaN	NaN	NaN	127.01	NaN	
19	13-19	NaN	NaN	NaN	129.27	NaN	
21	14-18	NaN	NaN	NaN	85.00	NaN	

22	14-18	NaN	NaN	NaN	87.50	NaN
23	14-18	NaN	NaN	NaN	93.00	NaN
24	14-18	NaN	NaN	NaN	85.00	NaN
25	14-18	NaN	NaN	NaN	102.50	NaN

	Bench2Kg	Bench3Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	\
0	NaN	NaN	58.97	NaN	NaN	NaN	
1	NaN	NaN	63.50	NaN	NaN	NaN	
2	NaN	NaN	61.23	NaN	NaN	NaN	
3	NaN	NaN	61.23	NaN	NaN	NaN	
4	NaN	NaN	65.77	NaN	NaN	NaN	
5	NaN	NaN	65.77	NaN	NaN	NaN	
7	NaN	NaN	61.23	NaN	NaN	NaN	
9	NaN	NaN	68.04	NaN	NaN	NaN	
10	NaN	NaN	NaN	NaN	NaN	NaN	
11	NaN	NaN	68.04	NaN	NaN	NaN	
12	NaN	NaN	NaN	NaN	NaN	NaN	
13	NaN	NaN	29.48	NaN	NaN	NaN	
14	NaN	NaN	63.50	NaN	NaN	NaN	
15	NaN	NaN	29.48	NaN	NaN	NaN	
16	NaN	NaN	29.48	NaN	NaN	NaN	
17	NaN	NaN	65.77	NaN	NaN	NaN	
18	NaN	NaN	68.04	NaN	NaN	NaN	
19	NaN	NaN	65.77	NaN	NaN	NaN	
21	NaN	NaN	37.50	NaN	NaN	NaN	
22	NaN	NaN	45.00	NaN	NaN	NaN	
23	NaN	NaN	45.00	NaN	NaN	NaN	
24	NaN	NaN	40.00	NaN	NaN	NaN	
25	NaN	NaN	52.50	NaN	NaN	NaN	

	Best3DeadliftKg	TotalKg	DateOfBirthInterval
0	120.20	288.03	[1994-01-11, 2001-01-11]
1	127.01	306.17	[1994-01-18, 2001-01-18]
2	129.27	310.71	[1994-01-25, 2001-01-25]
3	133.81	319.78	[1994-02-01, 2001-02-01]
4	138.35	326.59	[1994-02-10, 2001-02-10]
5	136.08	315.25	[1994-03-01, 2001-03-01]
7	142.88	319.78	[1994-03-15, 2001-03-15]
9	131.54	310.71	[1995-01-10, 2002-01-10]
10	NaN	NaN	[1995-01-14, 2002-01-14]
11	138.35	326.59	[1995-01-17, 2002-01-17]
12	NaN	NaN	[1995-01-24, 2002-01-24]
13	88.45	197.31	[1995-01-24, 2002-01-24]
14	136.08	324.32	[1995-01-31, 2002-01-31]
15	86.18	190.51	[1995-02-05, 2002-02-05]
16	83.91	197.31	[1995-02-21, 2002-02-21]
17	138.35	333.39	[1995-02-21, 2002-02-21]
18	142.88	337.93	[1995-03-07, 1999-03-07]

19	140.61	335.66	[1995-03-19, 2002-03-19]
21	115.00	237.50	[2000-10-29, 2001-10-28]
22	120.00	252.50	[2001-02-25, 2002-02-24]
23	110.00	248.00	[2000-06-24, 2001-06-23]
24	115.00	240.00	[2000-12-02, 2001-12-01]
25	125.00	280.00	[2001-04-21, 2002-04-20]

Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.

Type 'Q' to quit, or 'L' to save the lifter for later: y
Enter the row indicies for this disambiguation, separated by a space: 21 22 23
24 25

New lifter is as shown:

	Name	Federation	Event	Date	MeetCountry	Event	\	
21	Alexandra García	USPA	SBD	2017-10-28	USA	SBD		
22	Alexandra García	USPA	SBD	2018-02-24	USA	SBD		
23	Alexandra García	USPA	SBD	2018-06-23	USA	SBD		
24	Alexandra García	USPA	SBD	2018-12-01	USA	SBD		
25	Alexandra García	USPA	SBD	2019-04-20	USA	SBD		
	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	\	
21	Juniors	16-17	Raw	56.80	16.0	16-17	14-18	
22		Open	Raw	55.10	16.0	16-17	14-18	
23	Juniors	16-17	Raw	58.40	17.0	16-17	14-18	
24	Juniors	16-17	Raw	56.00	17.0	16-17	14-18	
25	Juniors	16-17	Raw	61.55	17.0	16-17	14-18	
	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	\
21	NaN	NaN	NaN	85.0	NaN	NaN	NaN	
22	NaN	NaN	NaN	87.5	NaN	NaN	NaN	
23	NaN	NaN	NaN	93.0	NaN	NaN	NaN	
24	NaN	NaN	NaN	85.0	NaN	NaN	NaN	
25	NaN	NaN	NaN	102.5	NaN	NaN	NaN	
	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Best3DeadliftKg	\		
21	37.5	NaN	NaN	NaN	115.0			
22	45.0	NaN	NaN	NaN	120.0			
23	45.0	NaN	NaN	NaN	110.0			
24	40.0	NaN	NaN	NaN	115.0			
25	52.5	NaN	NaN	NaN	125.0			
	TotalKg	DateOfBirthInterval						
21	237.5	[2000-10-29, 2001-10-28]						
22	252.5	[2001-02-25, 2002-02-24]						
23	248.0	[2000-06-24, 2001-06-23]						
24	240.0	[2000-12-02, 2001-12-01]						
25	280.0	[2001-04-21, 2002-04-20]						

Are you happy with this? Enter Y if yes, or N if no: y

	Name	Federation	Event	Event	Date	MeetCountry	Event	\
0	Alexandra García	THSWPA	SBD	SBD	2014-01-11	USA	SBD	
1	Alexandra García	THSWPA	SBD	SBD	2014-01-18	USA	SBD	
2	Alexandra García	THSWPA	SBD	SBD	2014-01-25	USA	SBD	
3	Alexandra García	THSWPA	SBD	SBD	2014-02-01	USA	SBD	
4	Alexandra García	THSWPA	SBD	SBD	2014-02-10	USA	SBD	
5	Alexandra García	THSWPA	SBD	SBD	2014-03-01	USA	SBD	
7	Alexandra García	THSWPA	SBD	SBD	2014-03-15	USA	SBD	
9	Alexandra García	THSWPA	SBD	SBD	2015-01-10	USA	SBD	
10	Alexandra García	THSWPA	SBD	SBD	2015-01-14	USA	SBD	
11	Alexandra García	THSWPA	SBD	SBD	2015-01-17	USA	SBD	
12	Alexandra García	THSWPA	SBD	SBD	2015-01-24	USA	SBD	
13	Alexandra García	THSWPA	SBD	SBD	2015-01-24	USA	SBD	
14	Alexandra García	THSWPA	SBD	SBD	2015-01-31	USA	SBD	
15	Alexandra García	THSWPA	SBD	SBD	2015-02-05	USA	SBD	
16	Alexandra García	THSWPA	SBD	SBD	2015-02-21	USA	SBD	
17	Alexandra García	THSWPA	SBD	SBD	2015-02-21	USA	SBD	
18	Alexandra García	THSWPA	SBD	SBD	2015-03-07	USA	SBD	
19	Alexandra García	THSWPA	SBD	SBD	2015-03-19	USA	SBD	

	Event	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	\
0	SBD	Girls	Single-ply	50.80	NaN	13-19	13-19	
1	SBD	Girls	Single-ply	50.53	NaN	13-19	13-19	
2	SBD	Girls	Single-ply	50.08	NaN	13-19	13-19	
3	SBD	Girls	Single-ply	49.90	NaN	13-19	13-19	
4	SBD	Girls	Single-ply	51.89	NaN	13-19	13-19	
5	SBD	Girls	Single-ply	50.08	NaN	13-19	13-19	
7	SBD	Girls	Single-ply	50.08	NaN	13-19	13-19	
9	SBD	Girls	Single-ply	51.62	NaN	13-19	13-19	
10	SBD	Girls	Single-ply	58.29	NaN	13-19	13-19	
11	SBD	Girls	Single-ply	51.35	NaN	13-19	13-19	
12	SBD	Girls	Single-ply	51.26	NaN	13-19	13-19	
13	SBD	Girls	Single-ply	57.70	NaN	13-19	13-19	
14	SBD	Girls	Single-ply	51.35	NaN	13-19	13-19	
15	SBD	Girls	Single-ply	60.15	NaN	13-19	13-19	
16	SBD	Girls	Single-ply	57.70	NaN	13-19	13-19	
17	SBD	Girls	Single-ply	51.44	NaN	13-19	13-19	
18	SBD	Div 2	Single-ply	51.17	NaN	16-19	16-19	
19	SBD	Girls	Single-ply	51.12	NaN	13-19	13-19	

	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	\
0	NaN	NaN	NaN	108.86	NaN	NaN	NaN	
1	NaN	NaN	NaN	115.67	NaN	NaN	NaN	
2	NaN	NaN	NaN	120.20	NaN	NaN	NaN	
3	NaN	NaN	NaN	124.74	NaN	NaN	NaN	
4	NaN	NaN	NaN	122.47	NaN	NaN	NaN	

5	NaN	NaN	NaN	113.40	NaN	NaN	NaN
7	NaN	NaN	NaN	115.67	NaN	NaN	NaN
9	NaN	NaN	NaN	111.13	NaN	NaN	NaN
10	NaN	NaN	NaN	72.57	NaN	NaN	NaN
11	NaN	NaN	NaN	120.20	NaN	NaN	NaN
12	NaN	NaN	NaN	120.20	NaN	NaN	NaN
13	NaN	NaN	NaN	79.38	NaN	NaN	NaN
14	NaN	NaN	NaN	124.74	NaN	NaN	NaN
15	NaN	NaN	NaN	74.84	NaN	NaN	NaN
16	NaN	NaN	NaN	83.91	NaN	NaN	NaN
17	NaN	NaN	NaN	129.27	NaN	NaN	NaN
18	NaN	NaN	NaN	127.01	NaN	NaN	NaN
19	NaN	NaN	NaN	129.27	NaN	NaN	NaN

	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Best3DeadliftKg	\
0	58.97	NaN	NaN	NaN	120.20	
1	63.50	NaN	NaN	NaN	127.01	
2	61.23	NaN	NaN	NaN	129.27	
3	61.23	NaN	NaN	NaN	133.81	
4	65.77	NaN	NaN	NaN	138.35	
5	65.77	NaN	NaN	NaN	136.08	
7	61.23	NaN	NaN	NaN	142.88	
9	68.04	NaN	NaN	NaN	131.54	
10	NaN	NaN	NaN	NaN	NaN	
11	68.04	NaN	NaN	NaN	138.35	
12	NaN	NaN	NaN	NaN	NaN	
13	29.48	NaN	NaN	NaN	88.45	
14	63.50	NaN	NaN	NaN	136.08	
15	29.48	NaN	NaN	NaN	86.18	
16	29.48	NaN	NaN	NaN	83.91	
17	65.77	NaN	NaN	NaN	138.35	
18	68.04	NaN	NaN	NaN	142.88	
19	65.77	NaN	NaN	NaN	140.61	

	TotalKg	DateOfBirthInterval
0	288.03	[1994-01-11, 2001-01-11]
1	306.17	[1994-01-18, 2001-01-18]
2	310.71	[1994-01-25, 2001-01-25]
3	319.78	[1994-02-01, 2001-02-01]
4	326.59	[1994-02-10, 2001-02-10]
5	315.25	[1994-03-01, 2001-03-01]
7	319.78	[1994-03-15, 2001-03-15]
9	310.71	[1995-01-10, 2002-01-10]
10	NaN	[1995-01-14, 2002-01-14]
11	326.59	[1995-01-17, 2002-01-17]
12	NaN	[1995-01-24, 2002-01-24]
13	197.31	[1995-01-24, 2002-01-24]
14	324.32	[1995-01-31, 2002-01-31]

```

15 190.51 [1995-02-05, 2002-02-05]
16 197.31 [1995-02-21, 2002-02-21]
17 333.39 [1995-02-21, 2002-02-21]
18 337.93 [1995-03-07, 1999-03-07]
19 335.66 [1995-03-19, 2002-03-19]

```

Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.

Type 'Q' to quit, or 'L' to save the lifter for later: n

3009 names remaining

Amanda Garcia

original_index = 8901

	Name	Federation	Event	Event	Date	Meet	Country	Event	Event	\
0	Amanda Garcia	USAPL	SBD	SBD	2004-05-15		USA	SBD	SBD	
1	Amanda Garcia	USAPL	SBD	SBD	2005-04-01		USA	SBD	SBD	
2	Amanda Garcia	APF	SBD	SBD	2011-02-19		USA	SBD	SBD	
3	Amanda Garcia	APF	SBD	SBD	2011-04-01		USA	SBD	SBD	
4	Amanda Garcia	THSWPA	SBD	SBD	2015-01-14		USA	SBD	SBD	
5	Amanda Garcia	THSWPA	SBD	SBD	2015-01-17		USA	SBD	SBD	
6	Amanda Garcia	THSWPA	SBD	SBD	2015-01-24		USA	SBD	SBD	
7	Amanda Garcia	THSWPA	SBD	SBD	2015-01-30		USA	SBD	SBD	
8	Amanda Garcia	THSWPA	SBD	SBD	2015-02-02		USA	SBD	SBD	
9	Amanda Garcia	THSWPA	SBD	SBD	2015-02-05		USA	SBD	SBD	
10	Amanda Garcia	THSWPA	SBD	SBD	2015-02-14		USA	SBD	SBD	
11	Amanda Garcia	THSWPA	SBD	SBD	2015-02-21		USA	SBD	SBD	
12	Amanda Garcia	THSWPA	SBD	SBD	2015-02-21		USA	SBD	SBD	
13	Amanda Garcia	THSWPA	SBD	SBD	2016-01-09		USA	SBD	SBD	
14	Amanda Garcia	THSWPA	SBD	SBD	2016-01-30		USA	SBD	SBD	
15	Amanda Garcia	THSWPA	SBD	SBD	2016-02-11		USA	SBD	SBD	
16	Amanda Garcia	THSWPA	SBD	SBD	2016-03-05		USA	SBD	SBD	
17	Amanda Garcia	THSWPA	SBD	SBD	2016-03-08		USA	SBD	SBD	
18	Amanda Garcia	THSWPA	SBD	SBD	2017-01-14		USA	SBD	SBD	
19	Amanda Garcia	THSWPA	SBD	SBD	2017-01-28		USA	SBD	SBD	
20	Amanda Garcia	THSWPA	SBD	SBD	2017-02-04		USA	SBD	SBD	
21	Amanda Garcia	THSWPA	SBD	SBD	2017-03-04		USA	SBD	SBD	
22	Amanda Garcia	THSWPA	SBD	SBD	2017-03-18		USA	SBD	SBD	
23	Amanda Garcia	THSWPA	SBD	SBD	2018-01-03		USA	SBD	SBD	
24	Amanda Garcia	THSWPA	SBD	SBD	2018-02-03		USA	SBD	SBD	
25	Amanda Garcia	THSWPA	SBD	SBD	2018-03-03		USA	SBD	SBD	
26	Amanda Garcia	THSWPA	SBD	SBD	2018-03-16		USA	SBD	SBD	
27	Amanda Garcia	NASA	SBD	SBD	2019-10-19		USA	SBD	SBD	
28	Amanda Garcia	THSWPA	SBD	SBD	2020-01-18		USA	SBD	SBD	
29	Amanda Garcia	NASA	SBD	SBD	2020-02-22		USA	SBD	SBD	
30	Amanda Garcia	USPA	SBD	SBD	2020-09-19		USA	SBD	SBD	
31	Amanda Garcia	USAPL	SBD	SBD	2020-12-12		USA	SBD	SBD	
32	Amanda Garcia	THSWPA	SBD	SBD	2021-01-21		USA	SBD	SBD	
33	Amanda Garcia	THSWPA	SBD	SBD	2021-01-29		USA	SBD	SBD	
34	Amanda Garcia	WUAP-USA	SBD	SBD	2021-04-25		USA	SBD	SBD	

	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	\	
0	F-T1	Single-ply	NaN	NaN	13-15	14-18		
1	F-V	Single-ply	93.10	16.0	16-17	14-18		
2	F_OR_APF	Raw	59.87	27.0	24-34	24-39		
3	F_OR_AAPF	Raw	59.70	27.0	24-34	24-39		
4	Girls	Single-ply	104.55	NaN	13-19	13-19		
5	Girls	Single-ply	70.58	NaN	13-19	13-19		
6	Girls	Single-ply	105.23	NaN	13-19	13-19		
7	Girls	Single-ply	105.96	NaN	13-19	13-19		
8	Girls	Single-ply	70.76	NaN	13-19	13-19		
9	Girls	Single-ply	104.55	NaN	13-19	13-19		
10	Girls	Single-ply	106.87	NaN	13-19	13-19		
11	Girls	Single-ply	103.78	NaN	13-19	13-19		
12	Girls	Single-ply	74.84	NaN	13-19	13-19		
13	Girls	Single-ply	108.91	NaN	13-19	13-19		
14	Girls	Single-ply	108.50	NaN	13-19	13-19		
15	Girls	Single-ply	107.95	NaN	13-19	13-19		
16	Girls	Single-ply	110.04	NaN	13-19	13-19		
17	Girls	Single-ply	109.27	NaN	13-19	13-19		
18	Girls	Single-ply	99.70	NaN	13-19	13-19		
19	Girls	Single-ply	99.70	NaN	13-19	13-19		
20	Girls	Single-ply	99.61	NaN	13-19	13-19		
21	Girls	Single-ply	99.11	NaN	13-19	13-19		
22	Girls	Single-ply	97.70	NaN	13-19	13-19		
23	Girls	Single-ply	99.34	NaN	13-19	13-19		
24	Girls	Single-ply	99.20	NaN	13-19	13-19		
25	Girls	Single-ply	99.61	NaN	13-19	13-19		
26	Girls	Single-ply	98.84	NaN	13-19	13-19		
27	wjr	Wraps	59.19	NaN	20-23	NaN		
28	Girls	Single-ply	59.15	NaN	13-19	13-19		
29	wjr	Wraps	58.97	NaN	20-23	NaN		
30	Open	Raw	65.20	33.0	24-34	24-39		
31	FR-C	Raw	59.73	21.5	20-23	19-23		
32	Girls	Single-ply	55.47	NaN	13-19	13-19		
33	Girls	Single-ply	55.43	NaN	13-19	13-19		
34	Open	Wraps	66.50	32.0	24-34	24-39		
	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	\
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	-127.5	127.5	-137.5	127.50	62.5	67.5	70.0	
2	NaN	NaN	NaN	92.99	NaN	NaN	NaN	
3	85.0	90.0	95.0	95.00	65.0	67.5	-70.0	
4	NaN	NaN	NaN	88.45	NaN	NaN	NaN	
5	NaN	NaN	NaN	68.04	NaN	NaN	NaN	
6	NaN	NaN	NaN	97.52	NaN	NaN	NaN	
7	NaN	NaN	NaN	106.59	NaN	NaN	NaN	
8	NaN	NaN	NaN	63.50	NaN	NaN	NaN	

9	NaN	NaN	NaN	111.13	NaN	NaN	NaN
10	NaN	NaN	NaN	124.74	NaN	NaN	NaN
11	NaN	NaN	NaN	129.27	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	170.10	NaN	NaN	NaN
14	NaN	NaN	NaN	181.44	NaN	NaN	NaN
15	NaN	NaN	NaN	181.44	NaN	NaN	NaN
16	NaN	NaN	NaN	188.24	NaN	NaN	NaN
17	NaN	NaN	NaN	174.63	NaN	NaN	NaN
18	NaN	NaN	NaN	179.17	NaN	NaN	NaN
19	NaN	NaN	NaN	185.97	NaN	NaN	NaN
20	NaN	NaN	NaN	179.17	NaN	NaN	NaN
21	NaN	NaN	NaN	183.70	NaN	NaN	NaN
22	NaN	NaN	NaN	179.17	NaN	NaN	NaN
23	NaN	NaN	NaN	185.97	NaN	NaN	NaN
24	NaN	NaN	NaN	190.51	NaN	NaN	NaN
25	NaN	NaN	NaN	183.70	NaN	NaN	NaN
26	NaN	NaN	NaN	183.70	NaN	NaN	NaN
27	NaN	NaN	NaN	90.00	NaN	NaN	NaN
28	NaN	NaN	NaN	79.38	NaN	NaN	NaN
29	NaN	NaN	NaN	102.50	NaN	NaN	NaN
30	85.0	-92.5	-92.5	85.00	-50.0	50.0	52.5
31	92.5	97.5	102.5	102.50	52.5	57.5	62.5
32	NaN	NaN	NaN	70.31	NaN	NaN	NaN
33	NaN	NaN	NaN	70.31	NaN	NaN	NaN
34	127.5	132.5	137.5	137.50	67.5	70.0	-72.5

	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Best3DeadliftKg	\
0	NaN	NaN	NaN	NaN	NaN	
1	70.00	132.5	142.5	145.0	145.00	
2	61.23	NaN	NaN	NaN	113.40	
3	67.50	105.0	115.0	120.0	120.00	
4	45.36	NaN	NaN	NaN	106.59	
5	27.22	NaN	NaN	NaN	97.52	
6	45.36	NaN	NaN	NaN	113.40	
7	52.16	NaN	NaN	NaN	115.67	
8	31.75	NaN	NaN	NaN	102.06	
9	63.50	NaN	NaN	NaN	124.74	
10	56.70	NaN	NaN	NaN	113.40	
11	54.43	NaN	NaN	NaN	117.93	
12	NaN	NaN	NaN	NaN	NaN	
13	74.84	NaN	NaN	NaN	122.47	
14	79.38	NaN	NaN	NaN	142.88	
15	79.38	NaN	NaN	NaN	147.42	
16	83.91	NaN	NaN	NaN	147.42	
17	95.25	NaN	NaN	NaN	136.08	
18	79.38	NaN	NaN	NaN	151.95	
19	83.91	NaN	NaN	NaN	142.88	

20	88.45	NaN	NaN	NaN	161.03
21	106.59	NaN	NaN	NaN	156.49
22	108.86	NaN	NaN	NaN	147.42
23	115.67	NaN	NaN	NaN	156.49
24	117.93	NaN	NaN	NaN	147.42
25	124.74	NaN	NaN	NaN	165.56
26	115.67	NaN	NaN	NaN	151.95
27	52.50	NaN	NaN	NaN	122.50
28	38.56	NaN	NaN	NaN	83.91
29	57.50	NaN	NaN	NaN	145.00
30	52.50	112.5	120.0	122.5	122.50
31	62.50	132.5	137.5	142.5	142.50
32	36.29	NaN	NaN	NaN	88.45
33	43.09	NaN	NaN	NaN	97.52
34	70.00	150.0	157.5	162.5	162.50

	TotalKg	DateOfBirthInterval
0	NaN	[1989-05-15, 1991-05-15]
1	342.50	[1988-04-02, 1989-04-01]
2	267.62	[1983-02-20, 1984-02-19]
3	282.50	[1983-04-02, 1984-04-01]
4	240.40	[1995-01-14, 2002-01-14]
5	192.78	[1995-01-17, 2002-01-17]
6	256.28	[1995-01-24, 2002-01-24]
7	274.42	[1995-01-30, 2002-01-30]
8	197.31	[1995-02-02, 2002-02-02]
9	299.37	[1995-02-05, 2002-02-05]
10	294.84	[1995-02-14, 2002-02-14]
11	301.64	[1995-02-21, 2002-02-21]
12	NaN	[1995-02-21, 2002-02-21]
13	367.41	[1996-01-09, 2003-01-09]
14	403.70	[1996-01-30, 2003-01-30]
15	408.23	[1996-02-11, 2003-02-11]
16	419.57	[1996-03-05, 2003-03-05]
17	405.97	[1996-03-08, 2003-03-08]
18	410.50	[1997-01-14, 2004-01-14]
19	412.77	[1997-01-28, 2004-01-28]
20	428.64	[1997-02-04, 2004-02-04]
21	446.79	[1997-03-04, 2004-03-04]
22	435.45	[1997-03-18, 2004-03-18]
23	458.13	[1998-01-03, 2005-01-03]
24	455.86	[1998-02-03, 2005-02-03]
25	474.00	[1998-03-03, 2005-03-03]
26	451.32	[1998-03-16, 2005-03-16]
27	265.00	[1996-10-19, 1999-10-19]
28	201.85	[2000-01-18, 2007-01-18]
29	305.00	[1997-02-22, 2000-02-22]
30	260.00	[1986-09-20, 1987-09-19]

31 307.50 [1998-01-01, 1998-12-31]
 32 195.04 [2001-01-21, 2008-01-21]
 33 210.92 [2001-01-29, 2008-01-29]
 34 370.00 [1988-04-26, 1989-04-25]

Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.

Type 'Q' to quit, or 'L' to save the lifter for later: y
 Enter the row indicies for this disambiguation, separated by a space: 27 29 31
 New lifter is as shown:

	Name	Federation	Event	Date	MeetCountry	Event	Division	\
27	Amanda Garcia	NASA	SBD	2019-10-19	USA	SBD	wjr	
29	Amanda Garcia	NASA	SBD	2020-02-22	USA	SBD	wjr	
31	Amanda Garcia	USAPL	SBD	2020-12-12	USA	SBD	FR-C	
	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	Squat1Kg	Squat2Kg	\
27	Wraps	59.19	NaN	20-23		NaN	NaN	NaN
29	Wraps	58.97	NaN	20-23		NaN	NaN	NaN
31	Raw	59.73	21.5	20-23		19-23	92.5	97.5
	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	Best3BenchKg	\	
27	NaN	90.0	NaN	NaN	NaN		52.5	
29	NaN	102.5	NaN	NaN	NaN		57.5	
31	102.5	102.5	52.5	57.5	62.5		62.5	
	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Best3DeadliftKg	TotalKg	\		
27	NaN	NaN	NaN	122.5	265.0			
29	NaN	NaN	NaN	145.0	305.0			
31	132.5	137.5	142.5	142.5	307.5			
	DateOfBirthInterval							
27	[1996-10-19, 1999-10-19]							
29	[1997-02-22, 2000-02-22]							
31	[1998-01-01, 1998-12-31]							

Are you happy with this? Enter Y if yes, or N if no: y

	Name	Federation	Event	Event	Date	MeetCountry	Event	Event	\
0	Amanda Garcia	USAPL	SBD	SBD	2004-05-15	USA	SBD	SBD	
1	Amanda Garcia	USAPL	SBD	SBD	2005-04-01	USA	SBD	SBD	
2	Amanda Garcia	APF	SBD	SBD	2011-02-19	USA	SBD	SBD	
3	Amanda Garcia	APF	SBD	SBD	2011-04-01	USA	SBD	SBD	
4	Amanda Garcia	THSWPA	SBD	SBD	2015-01-14	USA	SBD	SBD	
5	Amanda Garcia	THSWPA	SBD	SBD	2015-01-17	USA	SBD	SBD	
6	Amanda Garcia	THSWPA	SBD	SBD	2015-01-24	USA	SBD	SBD	
7	Amanda Garcia	THSWPA	SBD	SBD	2015-01-30	USA	SBD	SBD	
8	Amanda Garcia	THSWPA	SBD	SBD	2015-02-02	USA	SBD	SBD	
9	Amanda Garcia	THSWPA	SBD	SBD	2015-02-05	USA	SBD	SBD	
10	Amanda Garcia	THSWPA	SBD	SBD	2015-02-14	USA	SBD	SBD	

11	Amanda Garcia	THSWPA	SBD	SBD	2015-02-21	USA	SBD	SBD
12	Amanda Garcia	THSWPA	SBD	SBD	2015-02-21	USA	SBD	SBD
13	Amanda Garcia	THSWPA	SBD	SBD	2016-01-09	USA	SBD	SBD
14	Amanda Garcia	THSWPA	SBD	SBD	2016-01-30	USA	SBD	SBD
15	Amanda Garcia	THSWPA	SBD	SBD	2016-02-11	USA	SBD	SBD
16	Amanda Garcia	THSWPA	SBD	SBD	2016-03-05	USA	SBD	SBD
17	Amanda Garcia	THSWPA	SBD	SBD	2016-03-08	USA	SBD	SBD
18	Amanda Garcia	THSWPA	SBD	SBD	2017-01-14	USA	SBD	SBD
19	Amanda Garcia	THSWPA	SBD	SBD	2017-01-28	USA	SBD	SBD
20	Amanda Garcia	THSWPA	SBD	SBD	2017-02-04	USA	SBD	SBD
21	Amanda Garcia	THSWPA	SBD	SBD	2017-03-04	USA	SBD	SBD
22	Amanda Garcia	THSWPA	SBD	SBD	2017-03-18	USA	SBD	SBD
23	Amanda Garcia	THSWPA	SBD	SBD	2018-01-03	USA	SBD	SBD
24	Amanda Garcia	THSWPA	SBD	SBD	2018-02-03	USA	SBD	SBD
25	Amanda Garcia	THSWPA	SBD	SBD	2018-03-03	USA	SBD	SBD
26	Amanda Garcia	THSWPA	SBD	SBD	2018-03-16	USA	SBD	SBD
28	Amanda Garcia	THSWPA	SBD	SBD	2020-01-18	USA	SBD	SBD
30	Amanda Garcia	USPA	SBD	SBD	2020-09-19	USA	SBD	SBD
32	Amanda Garcia	THSWPA	SBD	SBD	2021-01-21	USA	SBD	SBD
33	Amanda Garcia	THSWPA	SBD	SBD	2021-01-29	USA	SBD	SBD
34	Amanda Garcia	WUAP-USA	SBD	SBD	2021-04-25	USA	SBD	SBD

	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	\
0	F-T1	Single-ply	NaN	NaN	13-15	14-18	
1	F-V	Single-ply	93.10	16.0	16-17	14-18	
2	F_OR_APF	Raw	59.87	27.0	24-34	24-39	
3	F_OR_AAPF	Raw	59.70	27.0	24-34	24-39	
4	Girls	Single-ply	104.55	NaN	13-19	13-19	
5	Girls	Single-ply	70.58	NaN	13-19	13-19	
6	Girls	Single-ply	105.23	NaN	13-19	13-19	
7	Girls	Single-ply	105.96	NaN	13-19	13-19	
8	Girls	Single-ply	70.76	NaN	13-19	13-19	
9	Girls	Single-ply	104.55	NaN	13-19	13-19	
10	Girls	Single-ply	106.87	NaN	13-19	13-19	
11	Girls	Single-ply	103.78	NaN	13-19	13-19	
12	Girls	Single-ply	74.84	NaN	13-19	13-19	
13	Girls	Single-ply	108.91	NaN	13-19	13-19	
14	Girls	Single-ply	108.50	NaN	13-19	13-19	
15	Girls	Single-ply	107.95	NaN	13-19	13-19	
16	Girls	Single-ply	110.04	NaN	13-19	13-19	
17	Girls	Single-ply	109.27	NaN	13-19	13-19	
18	Girls	Single-ply	99.70	NaN	13-19	13-19	
19	Girls	Single-ply	99.70	NaN	13-19	13-19	
20	Girls	Single-ply	99.61	NaN	13-19	13-19	
21	Girls	Single-ply	99.11	NaN	13-19	13-19	
22	Girls	Single-ply	97.70	NaN	13-19	13-19	
23	Girls	Single-ply	99.34	NaN	13-19	13-19	
24	Girls	Single-ply	99.20	NaN	13-19	13-19	

25	Girls	Single-ply	99.61	NaN	13-19	13-19
26	Girls	Single-ply	98.84	NaN	13-19	13-19
28	Girls	Single-ply	59.15	NaN	13-19	13-19
30	Open	Raw	65.20	33.0	24-34	24-39
32	Girls	Single-ply	55.47	NaN	13-19	13-19
33	Girls	Single-ply	55.43	NaN	13-19	13-19
34	Open	Wraps	66.50	32.0	24-34	24-39

	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	\
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	-127.5	127.5	-137.5	127.50	62.5	67.5	70.0	
2	NaN	NaN	NaN	92.99	NaN	NaN	NaN	
3	85.0	90.0	95.0	95.00	65.0	67.5	-70.0	
4	NaN	NaN	NaN	88.45	NaN	NaN	NaN	
5	NaN	NaN	NaN	68.04	NaN	NaN	NaN	
6	NaN	NaN	NaN	97.52	NaN	NaN	NaN	
7	NaN	NaN	NaN	106.59	NaN	NaN	NaN	
8	NaN	NaN	NaN	63.50	NaN	NaN	NaN	
9	NaN	NaN	NaN	111.13	NaN	NaN	NaN	
10	NaN	NaN	NaN	124.74	NaN	NaN	NaN	
11	NaN	NaN	NaN	129.27	NaN	NaN	NaN	
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
13	NaN	NaN	NaN	170.10	NaN	NaN	NaN	
14	NaN	NaN	NaN	181.44	NaN	NaN	NaN	
15	NaN	NaN	NaN	181.44	NaN	NaN	NaN	
16	NaN	NaN	NaN	188.24	NaN	NaN	NaN	
17	NaN	NaN	NaN	174.63	NaN	NaN	NaN	
18	NaN	NaN	NaN	179.17	NaN	NaN	NaN	
19	NaN	NaN	NaN	185.97	NaN	NaN	NaN	
20	NaN	NaN	NaN	179.17	NaN	NaN	NaN	
21	NaN	NaN	NaN	183.70	NaN	NaN	NaN	
22	NaN	NaN	NaN	179.17	NaN	NaN	NaN	
23	NaN	NaN	NaN	185.97	NaN	NaN	NaN	
24	NaN	NaN	NaN	190.51	NaN	NaN	NaN	
25	NaN	NaN	NaN	183.70	NaN	NaN	NaN	
26	NaN	NaN	NaN	183.70	NaN	NaN	NaN	
28	NaN	NaN	NaN	79.38	NaN	NaN	NaN	
30	85.0	-92.5	-92.5	85.00	-50.0	50.0	52.5	
32	NaN	NaN	NaN	70.31	NaN	NaN	NaN	
33	NaN	NaN	NaN	70.31	NaN	NaN	NaN	
34	127.5	132.5	137.5	137.50	67.5	70.0	-72.5	

	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Best3DeadliftKg	\
0	NaN	NaN	NaN	NaN	NaN	
1	70.00	132.5	142.5	145.0	145.00	
2	61.23	NaN	NaN	NaN	113.40	
3	67.50	105.0	115.0	120.0	120.00	
4	45.36	NaN	NaN	NaN	106.59	

5	27.22	NaN	NaN	NaN	97.52
6	45.36	NaN	NaN	NaN	113.40
7	52.16	NaN	NaN	NaN	115.67
8	31.75	NaN	NaN	NaN	102.06
9	63.50	NaN	NaN	NaN	124.74
10	56.70	NaN	NaN	NaN	113.40
11	54.43	NaN	NaN	NaN	117.93
12	NaN	NaN	NaN	NaN	NaN
13	74.84	NaN	NaN	NaN	122.47
14	79.38	NaN	NaN	NaN	142.88
15	79.38	NaN	NaN	NaN	147.42
16	83.91	NaN	NaN	NaN	147.42
17	95.25	NaN	NaN	NaN	136.08
18	79.38	NaN	NaN	NaN	151.95
19	83.91	NaN	NaN	NaN	142.88
20	88.45	NaN	NaN	NaN	161.03
21	106.59	NaN	NaN	NaN	156.49
22	108.86	NaN	NaN	NaN	147.42
23	115.67	NaN	NaN	NaN	156.49
24	117.93	NaN	NaN	NaN	147.42
25	124.74	NaN	NaN	NaN	165.56
26	115.67	NaN	NaN	NaN	151.95
28	38.56	NaN	NaN	NaN	83.91
30	52.50	112.5	120.0	122.5	122.50
32	36.29	NaN	NaN	NaN	88.45
33	43.09	NaN	NaN	NaN	97.52
34	70.00	150.0	157.5	162.5	162.50

	TotalKg	DateOfBirthInterval
0	NaN	[1989-05-15, 1991-05-15]
1	342.50	[1988-04-02, 1989-04-01]
2	267.62	[1983-02-20, 1984-02-19]
3	282.50	[1983-04-02, 1984-04-01]
4	240.40	[1995-01-14, 2002-01-14]
5	192.78	[1995-01-17, 2002-01-17]
6	256.28	[1995-01-24, 2002-01-24]
7	274.42	[1995-01-30, 2002-01-30]
8	197.31	[1995-02-02, 2002-02-02]
9	299.37	[1995-02-05, 2002-02-05]
10	294.84	[1995-02-14, 2002-02-14]
11	301.64	[1995-02-21, 2002-02-21]
12	NaN	[1995-02-21, 2002-02-21]
13	367.41	[1996-01-09, 2003-01-09]
14	403.70	[1996-01-30, 2003-01-30]
15	408.23	[1996-02-11, 2003-02-11]
16	419.57	[1996-03-05, 2003-03-05]
17	405.97	[1996-03-08, 2003-03-08]
18	410.50	[1997-01-14, 2004-01-14]

```

19  412.77 [1997-01-28, 2004-01-28]
20  428.64 [1997-02-04, 2004-02-04]
21  446.79 [1997-03-04, 2004-03-04]
22  435.45 [1997-03-18, 2004-03-18]
23  458.13 [1998-01-03, 2005-01-03]
24  455.86 [1998-02-03, 2005-02-03]
25  474.00 [1998-03-03, 2005-03-03]
26  451.32 [1998-03-16, 2005-03-16]
28  201.85 [2000-01-18, 2007-01-18]
30  260.00 [1986-09-20, 1987-09-19]
32  195.04 [2001-01-21, 2008-01-21]
33  210.92 [2001-01-29, 2008-01-29]
34  370.00 [1988-04-26, 1989-04-25]

```

Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.

Type 'Q' to quit, or 'L' to save the lifter for later: n
3008 names remaining
Amy Thompson
original_index = 10127

	Name	Federation	Event	Event	Date	MeetCountry	Event	Event	\
0	Amy Thompson		PA	SBD	SBD	2005-06-25	Australia	SBD	SBD
1	Amy Thompson		PA	SBD	SBD	2005-07-30	Australia	SBD	SBD
2	Amy Thompson	GPC-AUS	BD	BD	2012-11-23	Australia	BD	BD	
3	Amy Thompson	USPA	SBD	SBD	2017-02-11	USA	SBD	SBD	
4	Amy Thompson	USAPL	SBD	SBD	2017-02-18	USA	SBD	SBD	
5	Amy Thompson	USAPL	SBD	SBD	2017-09-02	USA	SBD	SBD	
6	Amy Thompson	USPA	SBD	SBD	2017-12-09	USA	SBD	SBD	
7	Amy Thompson	USAPL	B	B	2018-01-20	USA	B	B	
8	Amy Thompson	USAPL	SBD	SBD	2018-07-14	USA	SBD	SBD	
9	Amy Thompson	USPA	SBD	SBD	2018-10-13	USA	SBD	SBD	
10	Amy Thompson	USAPL	SBD	SBD	2018-11-10	USA	SBD	SBD	
11	Amy Thompson	USPA	SBD	SBD	2018-12-15	USA	SBD	SBD	
12	Amy Thompson	USAPL	B	B	2019-01-19	USA	B	B	
13	Amy Thompson	USAPL	SBD	SBD	2019-02-16	USA	SBD	SBD	
14	Amy Thompson	USPA	SBD	SBD	2019-05-05	USA	SBD	SBD	
15	Amy Thompson	USAPL	SBD	SBD	2019-07-13	USA	SBD	SBD	
16	Amy Thompson	USAPL	B	B	2019-08-10	USA	B	B	
17	Amy Thompson	USPA	SBD	SBD	2019-10-13	USA	SBD	SBD	
18	Amy Thompson	USAPL	SBD	SBD	2019-11-09	USA	SBD	SBD	
19	Amy Thompson	USAPL	B	B	2020-02-29	USA	B	B	
20	Amy Thompson	USAPL	B	B	2020-02-29	USA	B	B	

	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	\
0	Open	Single-ply	66.15	NaN	Open	Open	
1	Open	Single-ply	68.30	NaN	Open	Open	
2	F-OR	Raw	55.90	28.0	24-34	24-39	
3	Submasters	35-39	Raw	75.00	39.0	35-39	NaN

4		FR-Jr	Raw	88.60	22.5	20-23	19-23
5		FR-Jr	Raw	96.77	22.5	20-23	19-23
6		Open	Raw	71.60	42.0	40-44	40-49
7		FR-Jr	Raw	101.20	23.5	24-34	24-39
8		FR-O	Raw	108.70	23.5	24-34	24-39
9		Open	Raw	68.50	43.0	40-44	40-49
10		FR-O	Raw	113.30	23.5	24-34	24-39
11	Masters	40-44	Raw	71.70	43.0	40-44	40-49
12		FR-O	Raw	114.80	24.5	24-34	24-39
13		FR-O	Raw	114.20	24.5	24-34	24-39
14	Masters	40-44	Raw	73.30	43.0	40-44	40-49
15		FR-O	Raw	108.95	24.5	24-34	24-39
16		FR-O	Raw	108.65	24.5	24-34	24-39
17	Masters	40-44	Raw	74.80	44.0	40-44	40-49
18		FR-O	Raw	107.30	24.5	24-34	24-39
19		FR-O	Raw	111.80	25.5	24-34	24-39
20		F-O	Single-ply	111.80	25.5	24-34	24-39

	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	\
0	110.0	-120.0	-125.0	110.0	60.0	65.0	-70.0	
1	120.0	-130.0	-130.0	120.0	-65.0	-65.0	65.0	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	97.5	NaN	NaN	NaN	
4	90.0	97.5	-102.5	97.5	60.0	65.0	-72.5	
5	-107.5	-107.5	-112.5	NaN	-80.0	-82.5	-82.5	
6	NaN	NaN	NaN	70.0	NaN	NaN	NaN	
7	NaN	NaN	NaN	NaN	67.5	70.0	77.5	
8	82.5	87.5	92.5	92.5	-77.5	80.0	85.0	
9	NaN	NaN	NaN	65.0	NaN	NaN	NaN	
10	102.5	107.5	115.0	115.0	-82.5	87.5	92.5	
11	NaN	NaN	NaN	72.5	NaN	NaN	NaN	
12	NaN	NaN	NaN	NaN	82.5	90.0	95.0	
13	-122.5	-122.5	-122.5	NaN	NaN	NaN	NaN	
14	NaN	NaN	NaN	75.0	NaN	NaN	NaN	
15	97.5	105.0	-115.0	105.0	-82.5	85.0	-97.5	
16	NaN	NaN	NaN	NaN	82.5	87.5	-92.5	
17	67.5	72.5	77.5	77.5	42.5	47.5	50.0	
18	100.0	-107.5	-107.5	100.0	82.5	85.0	87.5	
19	NaN	NaN	NaN	NaN	80.0	85.0	87.5	
20	NaN	NaN	NaN	NaN	80.0	82.5	85.0	

	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Best3DeadliftKg	\
0	65.0	130.0	-140.0	140.0	140.0	
1	65.0	135.0	142.5	-145.0	142.5	
2	65.0	NaN	NaN	NaN	150.0	
3	67.5	NaN	NaN	NaN	125.0	
4	65.0	112.5	122.5	127.5	127.5	
5	NaN	137.5	140.0	142.5	142.5	

6	40.0	NaN	NaN	NaN	87.5
7	77.5	NaN	NaN	NaN	NaN
8	85.0	130.0	-137.5	142.5	142.5
9	45.0	NaN	NaN	NaN	102.5
10	92.5	142.5	150.0	157.5	157.5
11	47.5	NaN	NaN	NaN	97.5
12	95.0	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	NaN	NaN
14	47.5	NaN	NaN	NaN	105.0
15	85.0	142.5	152.5	-167.5	152.5
16	87.5	NaN	NaN	NaN	NaN
17	50.0	92.5	102.5	107.5	107.5
18	87.5	150.0	155.0	162.5	162.5
19	87.5	NaN	NaN	NaN	NaN
20	85.0	NaN	NaN	NaN	NaN

	TotalKg	DateOfBirthInterval
0	315.0	Open
1	327.5	Open
2	215.0	[1983-11-24, 1984-11-23]
3	290.0	[1977-02-12, 1978-02-11]
4	290.0	[1994-01-01, 1994-12-31]
5	NaN	[1994-01-01, 1994-12-31]
6	197.5	[1974-12-10, 1975-12-09]
7	77.5	[1994-01-01, 1994-12-31]
8	320.0	[1994-01-01, 1994-12-31]
9	212.5	[1974-10-14, 1975-10-13]
10	365.0	[1994-01-01, 1994-12-31]
11	217.5	[1974-12-16, 1975-12-15]
12	95.0	[1994-01-01, 1994-12-31]
13	NaN	[1994-01-01, 1994-12-31]
14	227.5	[1975-05-06, 1976-05-05]
15	342.5	[1994-01-01, 1994-12-31]
16	87.5	[1994-01-01, 1994-12-31]
17	235.0	[1974-10-14, 1975-10-13]
18	350.0	[1994-01-01, 1994-12-31]
19	87.5	[1994-01-01, 1994-12-31]
20	85.0	[1994-01-01, 1994-12-31]

Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.

Type 'Q' to quit, or 'L' to save the lifter for later: q

Might be helpful to select Place column

Appendix D Lifter Disambiguation - Web Scraping function

Appendix_D_Data cleaning part 3 - truncating lifts and identifying incorrect intervals_webscrapingedition

September 16, 2021

- 1 Deleting duplicate meets where bench and deadlift have been separated from each other, and creating list of lifters where ages do not align in their meets, or bodyweights do not align on “Same-day” meets.

```
[6]: import pandas as pd
import datetime
from dateutil.relativedelta import *
import json

openpowerlifting_USAPL_lifters = pd.
    ↪read_csv("open_powerlifting_USAPL_Raw_active_lifters.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    ↪to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
```

```
[7]: pd.set_option("display.max_columns", None)
```

```
[8]: # function to truncate meets where single lifts have been split
def truncate_meets():
    def male_wilks_coefficients():
        return (-216.0475144, 16.2606339, -0.002388645, -0.00113732, 7.
    ↪01863*10**-6, -1.291*10**-8)
    def female_wilks_coefficients():
        return (594.31747775582, -27.23842536447, 0.82112226871, -0.
    ↪00930733913, 4.731582*10**-5, -9.054*10**-8)

    starting_index = 0
    finishing_index = 0
    for name in openpowerlifting_USAPL_lifters["Name"].drop_duplicates():
        print(name)
        repeated_meet = False
        while openpowerlifting_USAPL_lifters.iloc[finishing_index]["Name"] ==
    ↪name:
            finishing_index +=1
```

```

        if finishing_index == openpowerlifting_USAPL_lifters.shape[0]:
            break
        for i in range(starting_index, finishing_index):
            meet_1 = openpowerlifting_USAPL_lifters.iloc[i]
            competition_1 = meet_1["MeetName"]
            date_1 = meet_1["Date"]
            equipment_1 = meet_1["Equipment"]
            name_1 = meet_1["Name"]
            event_1 = meet_1["Event"]
            bodyweight_1 = meet_1["BodyweightKg"]
            bestsquat_1 = meet_1["Best3SquatKg"]
            bestbench_1 = meet_1["Best3BenchKg"]
            bestdead_1 = meet_1["Best3DeadliftKg"]
            for j in range(i+1, finishing_index):
                meet_2 = openpowerlifting_USAPL_lifters.iloc[j]
                competition_2 = meet_2["MeetName"]
                date_2 = meet_2["Date"]
                equipment_2 = meet_2["Equipment"]
                name_2 = meet_2["Name"]
                event_2 = meet_2["Event"]
                bodyweight_2 = meet_2["BodyweightKg"]
                bestsquat_2 = meet_2["Best3SquatKg"]
                bestbench_2 = meet_2["Best3BenchKg"]
                bestdead_2 = meet_2["Best3DeadliftKg"]

                if date_1 == date_2 and name_1 == name_2 and bodyweight_1 == bodyweight_2 and competition_1 == competition_2:
                    if event_1 == "S":
                        if event_2 == "B":
                            openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True
                            openpowerlifting_USAPL_lifters.at[i, "Event"] = "SB"
                            for k in ["Bench1Kg", "Bench2Kg", "Bench3Kg", "Bench4Kg", "Best3BenchKg"]:
                                openpowerlifting_USAPL_lifters.at[i, k] = openpowerlifting_USAPL_lifters.at[j, k]
                            if pd.notna(openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]:
                                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]
                                if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                                    a, b, c, d, e, f = male_wilks_coefficients()
                                    bw = bodyweight_1

```

```

        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * 500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
        else:
            a, b, c, d, e, f = \
                female_wilks_coefficients()
            bw = bodyweight_1
            openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * 500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)

    elif event_2 == "D":
        openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True
        openpowerlifting_USAPL_lifters.at[i, "Event"] = "SD"
        for k in ["Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Deadlift4Kg", "Best3DeadliftKg"]:
            openpowerlifting_USAPL_lifters.at[i, k] = \
                openpowerlifting_USAPL_lifters.at[j, k]
            if pd.notna(openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]:
                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]
                if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                    a, b, c, d, e, f = male_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                else:
                    a, b, c, d, e, f = \
                        female_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)

```

```

        elif event_2 == "S" and bestsquat_1 == bestsquat_2:
            openpowerlifting_USAPL_lifters.at[j, "Event"] = "SB"
            openpowerlifting_USAPL_lifters.at[j, "Best3SquatKg"] = True
            openpowerlifting_USAPL_lifters.at[j, "TotalKg"] = openpowerlifting_USAPL_lifters.at[j, "Best3SquatKg"] + openpowerlifting_USAPL_lifters.at[j, "Wilks"]

        elif event_1 == "B":
            if event_2 == "S":
                openpowerlifting_USAPL_lifters.at[j, "Event"] = "SB"
                for k in ["Squat1Kg", "Squat2Kg", "Squat3Kg", "Squat4Kg", "Best3SquatKg"]:
                    openpowerlifting_USAPL_lifters.at[i, k] = openpowerlifting_USAPL_lifters.at[j, k]
                if pd.notna(openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]:
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]
                    if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                        a, b, c, d, e, f = male_wilks_coefficients()
                        bw = bodyweight_1
                        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                    else:
                        a, b, c, d, e, f = female_wilks_coefficients()
                        bw = bodyweight_1
                        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)

            elif event_2 == "D":
                openpowerlifting_USAPL_lifters.at[j, "Event"] = "BD"
                openpowerlifting_USAPL_lifters.at[j, "Best3DeadliftKg"] = True
                openpowerlifting_USAPL_lifters.at[j, "TotalKg"] = openpowerlifting_USAPL_lifters.at[j, "Best3DeadliftKg"] + openpowerlifting_USAPL_lifters.at[j, "Wilks"]

```

```

                if pd.notna(openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]:
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]
                    if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                        a, b, c, d, e, f = male_wilks_coefficients()
                        bw = bodyweight_1
                        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                    else:
                        a, b, c, d, e, f = female_wilks_coefficients()
                        bw = bodyweight_1
                        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)

                elif event_2 == "B" and bestbench_1 == bestbench_2:
                    openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True

            elif event_1 == "D":
                if event_2 == "S":
                    openpowerlifting_USAPL_lifters.at[j, "Repeated_meet"] = True
                    openpowerlifting_USAPL_lifters.at[i, "Event"] = "SD"
                    for k in ["Squat1Kg", "Squat2Kg", "Squat3Kg", "Squat4Kg", "Best3SquatKg"]:
                        openpowerlifting_USAPL_lifters.at[i, k] = openpowerlifting_USAPL_lifters.at[j, k]
                        if pd.notna(openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]:
                            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]
                            if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                                a, b, c, d, e, f = male_wilks_coefficients()

```

```

        bw = bodyweight_1
        openpowerlifting_USAPL_lifters.at[i, □
→"Wilks"] = \
            openpowerlifting_USAPL_lifters.at[i, □
→"TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + □
→f*bw**5)
            else:
                a, b, c, d, e, f = □
→female_wilks_coefficients()
                bw = bodyweight_1
                openpowerlifting_USAPL_lifters.at[i, □
→"Wilks"] = \
                    openpowerlifting_USAPL_lifters.at[i, □
→"TotalKg"] * bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + □
→f*bw**5)

            elif event_2 == "B":
                openpowerlifting_USAPL_lifters.at[j, □
→"Repeated_meet"] = True
                    openpowerlifting_USAPL_lifters.at[i, "Event"] = "BD"
                    for k in ["Bench1Kg", "Bench2Kg", "Bench3Kg", □
→"Bench4Kg", "Best3BenchKg"]:
                        openpowerlifting_USAPL_lifters.at[i, k] = □
→openpowerlifting_USAPL_lifters.at[j, k]
                        if pd.notna(openpowerlifting_USAPL_lifters.at[i, □
→"Best3BenchKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, □
→"Best3BenchKg"]:
                            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] □
→+= openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]
                            if openpowerlifting_USAPL_lifters.at[i, "Sex"] □
→== "M":
                                a, b, c, d, e, f = male_wilks_coefficients()
                                bw = bodyweight_1
                                openpowerlifting_USAPL_lifters.at[i, □
→"Wilks"] = \
                                    openpowerlifting_USAPL_lifters.at[i, □
→"TotalKg"] * 500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                                    else:
                                        a, b, c, d, e, f = □
→female_wilks_coefficients()
                                        bw = bodyweight_1
                                        openpowerlifting_USAPL_lifters.at[i, □
→"Wilks"] = \
                                            openpowerlifting_USAPL_lifters.at[i, □
→"TotalKg"] * 500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)

```

```

        elif event_2 == "D" and bestdead_1 == bestdead_2:
            openpowerlifting_USAPL_lifters.at[j, u
    ↪"Repeated_meet"] = True

starting_index = finishing_index

```

[]: truncate_meets()
openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Repeated_meet"]
↪== True]

Removed 1349 rows

[97]: openpowerlifting_USAPL_lifters =
↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Repeated_meet"]
↪== False]
openpowerlifting_USAPL_lifters.
↪to_csv("open_powerlifting_USAPL_Raw_active_lifters.csv", index = False)

[99]: openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == "Zoe
↪Haynes"]

	Name	Sex	Event	Equipment	Age	AgeClass	BirthYearClass	\
202695	Zoe Haynes	F	BD	Raw	23.0	20-23	NaN	
202697	Zoe Haynes	F	SBD	Raw	46.5	45-49	40-49	
	Division	BodyweightKg	WeightClassKg	Squat1Kg	Squat2Kg	Squat3Kg	\	
202695	F_JR_AWPC	59.3	60	NaN	NaN	NaN		
202697	FR-O	61.4	63	115.0	120.0	-127.5		
	Squat4Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	Bench4Kg	\	
202695	NaN	NaN	60.0	-67.5	-67.5	NaN		
202697	NaN	120.0	62.5	65.0	-70.0	NaN		
	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Deadlift4Kg	\		
202695	60.0	132.5	-142.5	-143.0	NaN			
202697	65.0	142.5	150.0	157.5	NaN			
	Best3DeadliftKg	TotalKg	Place	Dots	Wilks	Glossbrenner	\	
202695	132.5	192.5	1	147.98	216.579666	131.74		
202697	157.5	342.5	2	374.24	375.120000	331.34		
	Goodlift	Tested	Country	State	Federation	ParentFederation	Date	\
202695	NaN	Yes	USA	NaN	WPC		WPC 2016-08-06	

```

202697      76.21    Yes     USA     MI      USAPL          IPF 2016-11-05

MeetCountry MeetState      MeetTown \
202695        USA         MI  Grand Rapids
202697        USA         MI           NaN

MeetName Repeated_meet \
202695      Single Lift Championships      False
202697 Michigan Powerlifting and BP State Championships      False

Two_meets_on_one_day DateOfBirthInterval
202695      True [1992-08-07, 1993-08-06]
202697      False [1969-01-01, 1969-12-31]

```

2 Identifying lifters who have intervals/ages that do not overlap, or have remaining meets that share the same date

```
[148]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json

openpowerlifting_USAPL_lifters = pd.
    →read_csv("open_powerlifting_USAPL_Raw_active_lifters.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    →to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
```

```
[149]: openpowerlifting_USAPL_lifters
```

	Name	Sex	Event	Equipment	Age	AgeClass	BirthYearClass	Division	BodyweightKg	WeightClassKg	Squat1Kg	Squat2Kg
0	A'Dren Hye	M	SBD	Raw	21.5	20-23	19-23					
1	A'Dren Hye	M	SBD	Wraps	21.5	20-23	19-23					
2	A'Dren Hye	M	SBD	Raw	22.5	20-23	19-23					
3	A'Dren Hye	M	SBD	Raw	23.5	24-34	24-39					
4	A.C. Alexander	M	BD	Raw	48.0	45-49	40-49					
...					
201419	Zuleyka Weaver	F	SBD	Raw	29.5	24-34	24-39					
201420	Zwick Ivan	M	B	Raw	77.5	75-79	70-999					
201421	Zyler Greene	M	SBD	Raw	15.5	16-17	14-18					
201422	Zyler Greene	M	B	Raw	15.5	16-17	14-18					
201423	Zyler Greene	M	SBD	Raw	16.5	16-17	14-18					
0	MR-Jr			92.00		93	NaN					
1	Amateur Juniors	20-23		99.16		100	NaN					

2		MR-0	91.10	93	-240.0	260.0
3		MR-0	104.10	105	-260.0	272.5
4	Law-Fire	48-55	100.00	100	NaN	NaN
...
201419		FR-0	51.20	52	77.5	82.5
201420		MR-M4b	55.60	59	NaN	NaN
201421		MR-T1	94.60	105	-165.0	165.0
201422		MR-T1	98.85	105	NaN	NaN
201423		MR-T2	107.20	120	167.5	180.0

	Squat3Kg	Squat4Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	\
0	NaN	NaN	245.00	NaN	NaN	NaN	
1	NaN	NaN	274.42	NaN	NaN	NaN	
2	272.5	NaN	272.50	155.0	-165.0	-165.0	
3	287.5	NaN	287.50	177.5	185.0	195.0	
4	NaN	NaN	NaN	NaN	NaN	NaN	
...	
201419	-87.5	NaN	82.50	45.0	50.0	-52.5	
201420	NaN	NaN	NaN	25.0	42.5	47.5	
201421	-175.0	NaN	165.00	102.5	-107.5	-107.5	
201422	NaN	NaN	NaN	-105.0	107.5	112.5	
201423	187.5	NaN	187.50	135.0	140.0	-145.0	

	Bench4Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	\
0	NaN	165.00	NaN	NaN	NaN	
1	NaN	185.97	NaN	NaN	NaN	
2	NaN	155.00	-280.0	280.0	-282.5	
3	NaN	195.00	260.0	270.0	275.0	
4	NaN	137.50	NaN	NaN	NaN	
...	
201419	NaN	50.00	122.5	127.5	135.0	
201420	NaN	47.50	NaN	NaN	NaN	
201421	NaN	102.50	150.0	170.0	-182.5	
201422	NaN	112.50	NaN	NaN	NaN	
201423	NaN	140.00	170.0	182.5	190.0	

	Deadlift4Kg	Best3DeadliftKg	TotalKg	Place	Dots	Wilks	\
0	NaN	257.50	667.50	1	426.93	421.50	
1	NaN	288.03	748.43	1	462.37	457.04	
2	NaN	280.00	707.50	1	454.70	448.92	
3	NaN	275.00	757.50	1	458.44	454.03	
4	NaN	187.50	325.00	1	200.04	197.79	
...	
201419	NaN	135.00	267.50	3	329.61	337.48	
201420	NaN	NaN	47.50	1	42.79	43.55	
201421	NaN	170.00	437.50	1	276.11	272.66	
201422	NaN	NaN	112.50	1	69.60	68.79	

201423	NaN	190.00	517.50	1	309.54	307.06
--------	-----	--------	--------	---	--------	--------

	Glossbrenner	Goodlift	Tested	Country	State	Federation	\
0	403.59	87.79	Yes	USA	NaN	USAPL	
1	436.59	94.93	Yes	USA	NaN	RPS	
2	430.02	93.50	Yes	USA	NC	USAPL	
3	433.54	93.90	Yes	USA	NC	USAPL	
4	188.91	NaN	Yes	USA	WI	WABDL	
...	
201419	299.33	68.46	Yes	USA	AZ	USAPL	
201420	42.71	29.63	Yes	USA	IL	USAPL	
201421	260.80	56.76	Yes	USA	KY	USAPL	
201422	65.71	51.82	Yes	USA	KY	USAPL	
201423	293.29	63.28	Yes	USA	KY	USAPL	

	ParentFederation	Date	MeetCountry	MeetState	MeetTown	\
0	IPF	2015-03-21	USA	NC	NaN	
1	NaN	2015-12-12	USA	NC	Fayetteville	
2	IPF	2016-06-04	USA	NC	NaN	
3	IPF	2017-04-08	USA	SC	NaN	
4	NaN	2018-04-21	USA	WI	NaN	
...	
201419	IPF	2018-08-11	USA	AZ	NaN	
201420	IPF	2017-05-06	USA	IL	NaN	
201421	IPF	2020-07-18	USA	OH	NaN	
201422	IPF	2020-08-29	USA	OH	NaN	
201423	IPF	2021-04-17	USA	OH	NaN	

	MeetName	Repeated_meet	\
0	Battle on the Border IX	False	
1	2nd Annual Holiday Havok	False	
2	North Carolina State Championships	False	
3	Battle on the Border	False	
4	World Cup BP & DL	False	
...	
201419	Southwest Regional Championships	False	
201420	Illinois State Meet	False	
201421	PowerMania MMXX	False	
201422	Ohio Open	False	
201423	PowerMania MMXXI	False	

	Two_meets_on_one_day	DateOfBirthInterval
0	False	[1993-01-01, 1993-12-31]
1	False	[1993-01-01, 1993-12-31]
2	False	[1993-01-01, 1993-12-31]
3	False	[1993-01-01, 1993-12-31]
4	False	[1969-04-22, 1970-04-21]

```

...
201419          ...          ...
201420          False   [1988-01-01, 1988-12-31]
201421          False   [1939-01-01, 1939-12-31]
201422          False   [2004-01-01, 2004-12-31]
201423          False   [2004-01-01, 2004-12-31]

```

[201424 rows x 44 columns]

[]: #reconverting date of birth interval from list to interval

```

for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open" ↴
    ↪and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        print(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] [1:-1].split(", "))
        list_date = openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] [1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd.
    ↪Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed =
    ↪"both")

```

[]: # function to store lifters names in a list, where the lifter has DOB intervals ↴ which do not align,
or contains an "Open" meet, or where DOB intervals are empty

```

starting_index = 0
finishing_index = 0
can_compare_ages = False
names = []
for name in openpowerlifting_USAPL_lifters["Name"].drop_duplicates():
    print(name)
    name_indices = ↪openpowerlifting_USAPL_lifters[name == name].index
    num_records = len(name_indices)
    for i in range(num_records):
        if len(names) > 0 and names[-1] == name:
            break

        DOB_1 = openpowerlifting_USAPL_lifters.at[name_indices[i], "DateOfBirthInterval"]
        age_1 = openpowerlifting_USAPL_lifters.at[name_indices[i], "Age"]
        date_1 = openpowerlifting_USAPL_lifters.at[name_indices[i], "Date"]
        if (type(age_1) == int) or (type(age_1) == float):
            age_1 = int(age_1)
            can_compare_ages = True

```

```

    else:
        can_compare_ages = False

    if type(DOB_1) != pd._libs.interval.Interval:
        names.append(name)
        print(1)
        break

    if (i+1) == num_records:
        break

    for j in range(i+1, num_records):
        DOB_2 = openpowerlifting_USAPL_lifters.at[name_indicies[j], "DateOfBirthInterval"]
        age_2 = openpowerlifting_USAPL_lifters.at[name_indicies[j], "Age"]
        date_2 = openpowerlifting_USAPL_lifters.at[name_indicies[j], "Date"]

        if (type(age_2) == int) or (type(age_2) == float) and can_compare_ages == True:
            age_2 = int(age_2)
            if age_2 < age_1: # meets organised in date order, therefore
                age_2 should be equal to or more than age 1
            names.append(name)
            print(2)
            break

        if type(DOB_2) != pd._libs.interval.Interval:
            names.append(name)
            print(3)
            break

        if not DOB_2.overlaps(DOB_1): # check if date of birth intervals
            overlap or not
            names.append(name)
            print(4)
            break

        if date_1 == date_2:
            names.append(name)
            print(5)
            break

```

```
[152]: np.save("name disambiguation/duplicatenames", names)
[153]: np.save("name disambiguation/remainingnames", names)
[154]: np.save("name disambiguation/new_lifter_names", np.array([]))
[155]: openpowerlifting_USAPL_lifters.
    →to_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv", index_
    →= False)
```

3 Iterating through the name list to clean up/segment names

```
[156]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json
pd.set_option("display.max_columns", None)

openpowerlifting_USAPL_lifters = pd.
    →read_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    →to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')

#reconverting date of birth interval from list to interval
for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open"_
    →and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        list_date = openpowerlifting_USAPL_lifters.at[i,_
        →"DateOfBirthInterval"][1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd.
            →Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed =
            →"both")
```

```
[157]: original_dataset_names = {}
for i in pd.read_csv("open_powerlifting_original_data.csv")["Name"].
    →drop_duplicates():
    a = i.split(" #")
    if len(a) == 3:
        print(a)
    if len(a) == 0:
        print(a)
    if len(a) == 2 and a[0] not in original_dataset_names:
        original_dataset_names[a[0]] = [int(a[1])]
    elif len(a) == 2 and a[0] in original_dataset_names:
```

```

if original_dataset_names[a[0]] == None:
    original_dataset_names[a[0]] = [0, int(a[1])]
else:
    original_dataset_names[a[0]].append(int(a[1]))
elif len(a) == 1 and a[0] in original_dataset_names:
    if original_dataset_names[a[0]] == None:
        print("This should not be the case")
    else:
        original_dataset_names[a[0]].append(0)
elif len(a) == 1 and a[0] not in original_dataset_names:
    original_dataset_names[a[0]] = None

```

```

C:\Users\Montel\anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3146: DtypeWarning: Columns (33,35,38)
have mixed types.Specify dtype option on import or set low_memory=False.
    has_raised = await self.run_ast_nodes(code_ast.body, cell_name,

```

```
[158]: for i in original_dataset_names.keys():
    if type(original_dataset_names[i]) == list:
        original_dataset_names[i].sort()
```

Over 6000 lifters to check through

```

[161]: original_dataset_names_json = json.dumps(original_dataset_names)
jsonFile = open("name disambiguation/original_dataset_names.json", "w")
jsonFile.write(original_dataset_names_json)
jsonFile.close()

new_dataset_names_json = json.dumps(original_dataset_names)
jsonFile = open("name disambiguation/new_dataset_names.json", "w")
jsonFile.write(new_dataset_names_json)
jsonFile.close()

```

4 algorithm

for every name in the remaining names list 1. lookup the openpowerlifting dataset for each name and show the rows belonging to the name. 2. prompt the user to type in the row indicies that correspond to one person, separated by spaces. let the person type “q” to quit, or “f” to finish the person, or “l” to save them for later. 3. If a list is entered with commas separated, Then, the row indicies in the openpowerlifting table corresponding to the name will be updated (the name column) with the “name” + “#”+str(i), where i corresponds to the ith disambiguation of the individual. 4. The lifter’s name + number will then be added to “new lifter names”. The dataframe will then be segmented by the lifter name + number and this lifter segmentation will be saved to a csv file. The openpowerlifting dataset will also be resaved to csv with the updated names.

```
[81]: import pandas as pd
import numpy as np
```

```

import datetime
from dateutil.relativedelta import *
import json
import copy
import requests
import selenium
from selenium import webdriver
from selenium.webdriver.chrome.options import Options

pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', 500)

openpowerlifting_USAPL_lifters = pd.
    ↪read_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    ↪to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')

#reconverting date of birth interval from list to interval
for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open" ↪
        ↪and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        list_date = openpowerlifting_USAPL_lifters.at[i, ↪
            ↪"DateOfBirthInterval"][1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd.
            ↪Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed = ↪
            ↪"both")

```

```

[48]: def usapl_db_dict_generator():

    page_counter = 1

    lifter_page_dict = dict()
    CHROME_DRIVER_PATH = 'C:/Users/montel/Downloads/chromedriver_win32/ ↪
        ↪chromedriver'

    options = Options()
    options.headless = True
    options.add_argument("--window-size=1920,1200")

    driver = webdriver.Chrome(options = options, executable_path = ↪
        ↪CHROME_DRIVER_PATH)
    driver = webdriver.Chrome(executable_path = CHROME_DRIVER_PATH)
    usapl_gen_page_string = "https://usapl.liftingdatabase.com/lifters-default? ↪
        ↪p="

```

```

while True:
    usapl_specific_page_string = usapl_gen_page_string + str(page_counter)
    driver.get(usapl_specific_page_string)
    page_content = driver.find_element_by_id('content')
    table_of_lifters = page_content.find_element_by_tag_name('table')
    lifter_list = table_of_lifters.text.split("\n")
    if len(lifter_list) < 2:
        break
    lifter_page_dict[page_counter] = lifter_list
    page_counter += 1

return lifter_page_dict

lifter_page_dict = usapl_db_dict_generator

```

```

[101]: def lifter_dataframe_scraper(ambiguous_name, lifter_page_dict):
    options = Options()
    options.headless = True
    options.add_argument("--window-size=1920,1200")
    CHROME_DRIVER_PATH = 'C:/Users/Montel/Google Drive/Data Science MSc/
    ↪Msc-Project---Powerlifting/Datasets/chromedriver'
    usapl_gen_page_string = "https://usapl.liftingdatabase.com/lifters-default?
    ↪p="
    lifter_dataframes = []
    driver = webdriver.Chrome(options = options, executable_path = ↪
    ↪CHROME_DRIVER_PATH)
    for i,j in lifter_page_dict.items():
        lifter_dataframe = None
        if ambiguous_name in j:
            usapl_specific_page_string = usapl_gen_page_string + str(i)
            driver.get(usapl_specific_page_string) #navigate to page containing ↪
    ↪key
            page_content = driver.find_element_by_id('content')
            table_of_lifters = page_content.find_element_by_tag_name('table')
            matching_names = table_of_lifters.
    ↪find_elements_by_link_text(ambiguous_name)
            for i in matching_names:
                #if page contains date of birth:
                #replace ages with .5
                lifter_dataframe_list = pd.read_html(i.get_attribute('href'))
                indicies_to_remove = []
                for i in range(len(lifter_dataframe_list)):
                    if not all(elem in lifter_dataframe_list[i].columns for ↪
    ↪elem in ["Competition",
                            "Competition.1",
                            "Weight",
                            "Squat",

```

```

        "Squat.1",
        "Squat.2",
        "Bench press",
        "Bench press.1",
        "Bench press.2",
        "Deadlift",
        "Deadlift.1",
        "Deadlift.2",]):
    indicies_to_remove.append(i)
for i in range(len(indicies_to_remove)-1,-1,-1):
    lifter_dataframe_list.pop(indicies_to_remove[i])

if len(lifter_dataframe_list) > 1:
    # join together dataframes, and sort by date
    # for loop iterating through lifter dataframe list

    lifter_dataframe = pd.concat(lifter_dataframe_list) ↴
    ↪#computationally efficient to concat all at once
    elif len(lifter_dataframe_list) == 1:
        lifter_dataframe = lifter_dataframe_list[0]
    if type(lifter_dataframe) != None:
        lifter_dataframe["Competition"] = pd.
    ↪to_datetime(lifter_dataframe["Competition"], format='%m/%d/%Y')
        lifter_dataframe.sort_values(by = ["Competition"], axis = 0, ↴
    ↪inplace = True)
        lifter_dataframe.rename(columns = {"Competition": "Date", ↴
    ↪"Weight": "BodyweightKg", "Squat": "Squat1Kg", "Squat.1": "Squat2Kg", "Squat.2": "Squat3Kg",
                                            "Bench press": "Bench1Kg", ↴
    ↪"Bench press.1": "Bench2Kg", "Bench press.2": "Bench3Kg",
                                            "Deadlift": "Deadlift1Kg", ↴
    ↪"Deadlift.1": "Deadlift2Kg", "Deadlift.2": "Deadlift3Kg"}, inplace = True)

    lifter_dataframes.append(lifter_dataframe)

return lifter_dataframes

```

[109]: `def lifter_name_disambiguation():`

```

name_list = np.load("name disambiguation/remainingnames.npy")
new_names = np.load("name disambiguation/new_lifter_names.npy")
original_dataset_names = json.load(open("name disambiguation/
↪original_dataset_names.json"))

```

```

    new_dataset_names = json.load(open("name disambiguation/new_dataset_names.
→json"))
    name_list_in_progress = copy.deepcopy(name_list)
    end_loop = False

    def divide_choice_or_save_for_later(name):
        □
→display(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] ↵
→== name][["Name", "Federation", "Event", "Date", "MeetCountry", "Event", ↵
→"Division", "Equipment", "BodyweightKg", "Age", "AgeClass", ↵
→"BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", ↵
→"Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", ↵
→"Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", ↵
→"DateOfBirthInterval"]])
        next_step = input("""Can this lifter be further disambiguated? Press Y ↵
→if they can, press N if they cannot.
        Type 'Q' to quit, or 'L' to save the lifter for later: """)
        □

    return next_step

def save_lifter(separation, name, ambiguous_name, number_disambiguation):
    nonlocal new_dataset_names
    nonlocal name_list_in_progress
    nonlocal new_names
    for number in separation:
        openpowerlifting_USAPL_lifters.at[int(number), "Name"] = ↵
→f"{ambiguous_name} #{number_disambiguation}"

        openpowerlifting_USAPL_lifters.iloc[separation].to_csv(
            f"Name Disambiguation/{ambiguous_name} #{number_disambiguation}.csv",
            index = False)
        name_list_in_progress = name_list_in_progress[name_list_in_progress != ↵
→name]

        if type(new_dataset_names[ambiguous_name]) == list:
            new_dataset_names[ambiguous_name].append(number_disambiguation)
        else:
            new_dataset_names[ambiguous_name] = [1]

        new_names = np.append(new_names, (f"{ambiguous_name}" ↵
→#{number_disambiguation}"))
        np.save("name disambiguation/new_lifter_names", new_names)
        np.save("name disambiguation/remainingnames", name_list_in_progress)
        jsonFile = open("name disambiguation/new_dataset_names.json", "w")
        jsonFile.write(json.dumps(new_dataset_names))
        jsonFile.close()

```

```

for name in name_list:
    print(f"{name_list_in_progress.shape[0]} names left")
    if end_loop == True:
        break
    lifter_in_progress = True
    if "#" in name:
        ambiguous_name = name.split(" #")[0]
        lifter_in_progress = True
        number_disambiguation = 1
        scraped = False
        while lifter_in_progress == True:
            #1. find matches in dataframe and find those indicies
            #2. those indicies will equal separation with decision = y
            #3. if there are indicies remaining, show the user the
→generated dataframe and ask if any of the indicies remaining
            # can go into the generated dataframe. Have an option for all
→of the indicies going into the df.
            # When the user inserts the indicies that can go into the
→dataframe, the resultant dataframe will be assigned a
            # number, and that will be classed as an individual.
            #5. repeat this for each dataframe generated, as long as there
→are individuals remaining

    if
→len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==
→name) < 2:
        next_step = "N"
        lifter_in_progress = False

    elif scraped == False:
        matching_dataframes = lifter_dataframe_scraper(name,
→lifter_page_dict)
        if len(matching_dataframes) >= 1:
            for i in matching_dataframes:
                separation =
→list(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==
→name].merge(i, how = "inner", on = ["Date", "BodyweightKg", "Squat1Kg",
→"Squat2Kg", "Squat3Kg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Deadlift1Kg",
→"Deadlift2Kg", "Deadlift3Kg"], right_index = True).index.unique())
                if len(separation) == 0:
                    continue

```

```

        elif len(separation) == 1
    ↵openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == 1
    ↵name].shape[0]:
            while number_disambiguation in 1
    ↵new_dataset_names[ambiguous_name]:
                    number_disambiguation += 1
                    save_lifter(separation, name, ambiguous_name, 1
    ↵number_disambiguation)
                    number_disambiguation += 1
                    lifter_in_progress = False
                    next_step = "N"
                    break
            elif len(separation) < 1
    ↵openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == 1
    ↵name].shape[0]:
                    display(openpowerlifting_USAPL_lifters.
    ↵iloc[separation][["Name", "Federation", "Event", "Date", "MeetCountry", 1
    ↵"Event", "Division", "Equipment", "BodyweightKg", "Age", "AgeClass", 1
    ↵"BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", 1
    ↵"Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", 1
    ↵"Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", 1
    ↵"DateOfBirthInterval"]
    ↵
    ↵        ])
    ↵
    ↵        ]
    ↵        display(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] 1
    ↵== name][["Name", "Federation", "Event", "Date", "MeetCountry", "Event", 1
    ↵"Division", "Equipment", "BodyweightKg", "Age", "AgeClass", 1
    ↵"BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", 1
    ↵"Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", 1
    ↵"Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", 1
    ↵"DateOfBirthInterval"]
    ↵
    ↵        ].drop(separation, axis = 0))
        choice = input("""Look at the last two 1
    ↵dataframes. Enter 'y' if some indicies from df2 can go into
                df1, 'x' if all indicies can go from df2 to 1
    ↵df1, 'n' if no indicies can go from df2 to df1,
                'l' to save for later, or 'q' to quit.""))
    ↵
        if choice.upper() == "Y":
            separation.extend([int(b) for b in 1
    ↵input("Enter the row indicies to add, separated by a space: ").split()])
            separation = list(set(separation))
            separation.sort()

```

```

                while number_disambiguation in u
↳new_dataset_names[ambiguous_name]:
                    number_disambiguation += 1
                    save_lifter(separation, name, u
↳ambiguous_name, number_disambiguation)
                    number_disambiguation += 1
                    if u
↳len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == u
↳name]) < 2:
                        next_step = "N"
                    else:
                        next_step = u
↳divide_choice_or_save_for_later(name)

                    break
                elif choice.upper() == "X":
                    separation.extend(list(
                        u
↳openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == u
↳name][["Name", "Federation", "Event", "Date", "MeetCountry", "Event", u
↳"Division", "Equipment", "BodyweightKg", "Age", "AgeClass", u
↳"BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", u
↳"Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", u
↳"Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", u
↳"DateOfBirthInterval"]
                        u
↳
                        ].drop(separation, axis = 0).index))
                    separation = list(set(separation))
                    separation.sort()
                    while number_disambiguation in u
↳new_dataset_names[ambiguous_name]:
                        number_disambiguation += 1
                        save_lifter(separation, name, u
↳ambiguous_name, number_disambiguation)
                        number_disambiguation += 1
                        next_step = "N"
                        break
                    elif choice.upper() == "N":
                        while number_disambiguation in u
↳new_dataset_names[ambiguous_name]:
                            number_disambiguation += 1
                            save_lifter(separation, name, u
↳ambiguous_name, number_disambiguation)
                            number_disambiguation += 1
                            next_step = u
↳divide_choice_or_save_for_later(name)

```

```

                break
            elif choice.upper() == "L":
                next_step = "L"
                break
            elif choice.upper() == "Q":
                next_step = "Q"
                break
            elif choice.upper() == "F":
                scraped = True
                next_step = □
→divide_choice_or_save_for_later(name)
                break

            next_step = divide_choice_or_save_for_later(name)

        else:
            next_step = divide_choice_or_save_for_later(name)

    else:
        next_step = divide_choice_or_save_for_later(name)

try:
    if next_step.upper() == "Q":
        jsonFile = open("name disambiguation/new_dataset_names.
→json", "w")
        jsonFile.write(json.dumps(new_dataset_names))
        jsonFile.close()
        openpowerlifting_USAPL_lifters.
→to_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv", index=□
→= False)
        np.save("name disambiguation/remainingnames", □
→name_list_in_progress)
        np.save("name disambiguation/new_lifter_names", □
→new_names)
        lifter_in_progress = False
        end_loop = True
        break
    elif next_step.upper() == "N":
        □
→openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == □
→name].to_csv(
            f"Name Disambiguation/{name}.csv", index = False)
        name_list_in_progress = □
→name_list_in_progress[name_list_in_progress != name]

```

```

        np.save("name_disambiguation/remainingnames", name_list_in_progress)

        lifter_in_progress = False
        break

    elif next_step.upper() == "L":
        name_list_in_progress = []
        if name_list_in_progress[name_list_in_progress != name]:
            name_list_in_progress = np.append(name_list_in_progress, name)
        np.save("name_disambiguation/remainingnames", name_list_in_progress)
        lifter_in_progress = False
        break

    elif next_step.upper() == "Y":
        if len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == name]) == 2:
            while number_disambiguation in new_dataset_names[ambiguous_name]:
                number_disambiguation += 1
                separation = [openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == name].index[1]]
                save_lifter(separation, name, ambiguous_name, number_disambiguation)
                number_disambiguation += 1
                successful_loop = True
                break
            else:
                successful_loop = False
                while not successful_loop:
                    separation = input("Enter the row indicies for this disambiguation, separated by a space: ").split()
                    display(
                        openpowerlifting_USAPL_lifters[["Name", "Federation", "Event", "Date", "MeetCountry", "Event", "Division", "Equipment", "BodyweightKg", "Age", "AgeClass", "BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", "DateOfBirthInterval"]].iloc[separation])
                    while number_disambiguation in new_dataset_names[ambiguous_name]:
                        number_disambiguation += 1

```

```

        while True:
            decision = input("Are you happy with this? ")
            ↪Enter Y if yes, or N if no: "
            if type(decision) == str and decision.
            ↪upper() == "Y":
                save_lifter(separation, name, ↪
            ↪ambiguous_name, number_disambiguation)
                number_disambiguation += 1
                successful_loop = True
                break

            elif type(decision) == str and decision.
            ↪upper() == "N":
                break
            else:
                continue

        except TypeError:
            continue

    else:
        number_disambiguation = 1
        lifter_in_progress = True
        scraped = False
        while lifter_in_progress == True:
            #1. find matches in dataframe and find those indicies
            #2. those indicies will equal separation with decision = y
            #3. if there are indicies remaining, show the user the ↪
            ↪generated dataframe and ask if any of the indicies remaining
            # can go into the generated dataframe. Have an option for all ↪
            ↪of the indicies going into the df.
            # When the user inserts the indicies that can go into the ↪
            ↪dataframe, the resultant dataframe will be assigned a
            # number, and that will be classed as an individual.
            #5. repeat this for each dataframe generated, as long as there ↪
            ↪are individuals remaining

            length_of_frame = ↪
            ↪len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] == ↪
            ↪name))

            if ↪
            ↪len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] == ↪
            ↪name)) < 2:
                next_step = "N"
                lifter_in_progress = False

```

```
        elif scraped == False and length_of_frame > 7:
            matching_dataframes = lifter_dataframe_scraper(name, u
↳lifter_page_dict)
                if len(matching_dataframes) >= 1:
                    for i in matching_dataframes:
                        separation = u
↳list(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] u
↳== name].merge(i, how = "inner", on = ["Date", "BodyweightKg", "Squat1Kg", u
↳"Squat2Kg", "Squat3Kg", "Bench1Kg", "Bench2Kg", "Bench3Kg", "Deadlift1Kg", u
↳"Deadlift2Kg", "Deadlift3Kg"], right_index = True).index.unique())
                        if len(separation) == 0:
                            continue
                        elif len(separation) == u
↳openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == u
↳name].shape[0]:
                            save_lifter(separation, name, name, u
↳number_disambiguation)
                            number_disambiguation += 1
                            lifter_in_progress = False
                            next_step = "N"
                            break
                        elif len(separation) < u
↳openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == u
↳name].shape[0]:
                            display(openpowerlifting_USAPL_lifters.
↳iloc[separation][["Name", "Federation", "Event", "Date", "MeetCountry", u
↳"Event", "Division", "Equipment", "BodyweightKg", "Age", "AgeClass", u
↳"BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", u
↳"Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", u
↳"Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", u
↳"DateOfBirthInterval"]
↳
↳        ])
↳
↳        display(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] u
↳== name][["Name", "Federation", "Event", "Date", "MeetCountry", "Event", u
↳"Division", "Equipment", "BodyweightKg", "Age", "AgeClass", u
↳"BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", u
↳"Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg", u
↳"Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg", u
↳"DateOfBirthInterval"]
↳
↳                ].drop(separation, axis = 0))
                choice = input("""Look at the last two u
↳dataframes. Enter 'y' if some indicies from df2 can go into
```

```

        df1, 'x' if all indicies can go from df2 to
→df1, 'n' if no indicies can go from df2 to df1,
        'l' to save for later, 'f' if incorrect, or 'q'
→to quit.""")

    if choice.upper() == "Y":
        separation.extend([int(b) for b in
→input("Enter the row indicies to add, separated by a space: ").split()])
        separation = list(set(separation))
        separation.sort()
        while
→len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==
→\
            f"{name}"]
→#{number_disambiguation}) > 0:
            number_disambiguation += 1
            save_lifter(separation, name, name,
→number_disambiguation)
            number_disambiguation += 1
            if
→len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==
→name]) < 2:
                next_step = "N"
            else:
                next_step =
→divide_choice_or_save_for_later(name)

            break
        elif choice.upper() == "X":
            separation.extend(list(
→openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==
→name][["Name", "Federation", "Event", "Date", "MeetCountry", "Event",
→"Division", "Equipment", "BodyweightKg", "Age", "AgeClass",
→"BirthYearClass", "Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg",
→"Bench1Kg", "Bench2Kg", "Bench3Kg", "Best3BenchKg", "Deadlift1Kg",
→"Deadlift2Kg", "Deadlift3Kg", "Best3DeadliftKg", "TotalKg",
→"DateOfBirthInterval"])

            ].drop(separation, axis = 0).index))
            separation = list(set(separation))
            separation.sort()
            while
→len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==
→\

```

```

f"{{name}}"

→#{number_disambiguation}"]) > 0:
    number_disambiguation += 1
    save_lifter(separation, name, name, ↴
→number_disambiguation)
    number_disambiguation += 1
    next_step = "N"
    break
elif choice.upper() == "N":
    while ↴
→len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↴
→\
    f"{{name}}"

→#{number_disambiguation}") > 0:
    number_disambiguation += 1
    display(separation, name, ↴
→number_disambiguation)
    save_lifter(separation, name, name, ↴
→number_disambiguation)
    number_disambiguation += 1
    next_step = ↴
→divide_choice_or_save_for_later(name)
    break
elif choice.upper() == "L":
    next_step = "L"
    break
elif choice.upper() == "Q":
    next_step = "Q"
    break
elif choice.upper() == "F":
    scraped = True
    next_step = ↴
→divide_choice_or_save_for_later(name)
    break

next_step = divide_choice_or_save_for_later(name)

else:
    next_step = divide_choice_or_save_for_later(name)

else:
    next_step = divide_choice_or_save_for_later(name)

try:

```

```

        if next_step.upper() == "Q":
            jsonFile = open("name_disambiguation/new_dataset_names.
↪json", "w")
                jsonFile.write(json.dumps(new_dataset_names))
                jsonFile.close()
                openpowerlifting_USAPL_lifters.
↪to_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv", index_
↪= False)
                    np.save("name_disambiguation/remainingnames", □
↪name_list_in_progress)
                    np.save("name_disambiguation/new_lifter_names", □
↪new_names)
                    lifter_in_progress = False
                    end_loop = True
                    break
            elif next_step.upper() == "N":
                □
↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == □
↪name].to_csv(
                    f"Name_Disambiguation/{name}.csv", index = False)
                    name_list_in_progress = □
↪name_list_in_progress[name_list_in_progress != name]
                    np.save("name_disambiguation/remainingnames", □
↪name_list_in_progress)

                    lifter_in_progress = False
                    break

            elif next_step.upper() == "L":

                name_list_in_progress = □
↪name_list_in_progress[name_list_in_progress != name]
                name_list_in_progress = np.
↪append(name_list_in_progress, name)
                np.save("name_disambiguation/remainingnames", name_list)

                lifter_in_progress = False
                break

            elif next_step.upper() == "Y":
                successful_loop = False
                if □
↪len(openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == □
↪name]) == 2:

```

```

        while\u
→len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==\u
→\
→f"\{name}\u
→#{number_disambiguation}"]) > 0:
            number_disambiguation += 1
            separation =\u
→[openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==\u
→name].index[1]
            save_lifter(separation, name, name,\u
→number_disambiguation)
            number_disambiguation += 1
            successful_loop = True
            break
        else:

            while not successful_loop:
                separation = input("Enter the row indicies for\u
→this disambiguation, separated by a space: ").split()
                display(
                    openpowerlifting_USAPL_lifters[["Name",\u
→"Federation", "Event", "Date", "MeetCountry", "Event", "Division",\u
→"Equipment", "BodyweightKg", "Age", "AgeClass", "BirthYearClass",\u
→"Squat1Kg", "Squat2Kg", "Squat3Kg", "Best3SquatKg", "Bench1Kg", "Bench2Kg",\u
→"Bench3Kg", "Best3BenchKg", "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg",\u
→"Best3DeadliftKg", "TotalKg", "DateOfBirthInterval"]].iloc[separation])
                while\u
→len(openpowerlifting_USAPL_lifters/openpowerlifting_USAPL_lifters["Name"] ==\u
→\
→f"\{name} #{number_disambiguation}") > 0:
                    number_disambiguation += 1
                    while True:
                        decision = input("Are you happy with this?\u
→Enter Y if yes, or N if no: ")
                        if type(decision) == str and decision.
→upper() == "Y":
                            save_lifter(separation, name, name,\u
→number_disambiguation)
                            successful_loop = True
                            number_disambiguation += 1
                            break

                        elif type(decision) == str and decision.
→upper() == "N":
                            break

```

```

        else:
            continue

    except TypeError:
        continue

```

[110]: lifter_name_disambiguation()

5466 names left

	Name	Federation	Event	Date	MeetCountry	Event	\
29796	Brittany Miller	USAPL	SBD	2019-09-21	USA	SBD	
29797	Brittany Miller	RPS	SBD	2021-05-22	USA	SBD	
	Division	Equipment	BodyweightKg	Age	AgeClass	\	
29796	FR-0	Raw	84.35	18.5	18-19		
29797	Amateur Submasters	33-39	Wraps	81.10	NaN	33-39	
	BirthYearClass	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	\
29796	19-23	87.5	92.5	102.5	102.50	50.0	
29797	33-39	NaN	NaN	NaN	136.08	NaN	
	Bench2Kg	Bench3Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	\	
29796	57.5	60.0	60.0	117.5	132.5		
29797	NaN	NaN	63.5	NaN	NaN		
	Deadlift3Kg	Best3DeadliftKg	TotalKg	DateOfBirthInterval			
29796	137.5	137.5	300.00	[2000-01-01, 2000-12-31]			
29797	NaN	170.1	369.68	[1981-05-22, 1988-05-22]			

Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.

Type 'Q' to quit, or 'L' to save the lifter for later: y

5465 names left

	Name	Federation	Event	Date	MeetCountry	Event	\
29842	Brittany Russell	AAU	SBD	1998-07-31	USA	SBD	
29843	Brittany Russell	AAU	SBD	1999-08-06	USA	SBD	
29844	Brittany Russell	AAU	SBD	2001-07-27	USA	SBD	
29845	Brittany Russell	USAPL	SBD	2020-11-07	USA	SBD	
	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	\
29842	Youth Under 11	Single-ply	44.0	7.0	5-12	NaN	
29843	RY 8-9	Raw	30.0	8.0	5-12	NaN	
29844	RY 10-11	Raw	40.0	10.0	5-12	NaN	
29845	FR-0	Raw	59.8	24.5	24-34	24-39	
	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	\
29842	NaN	NaN	NaN	20.41	NaN	NaN	

29843	NaN	NaN	NaN	31.75	NaN	NaN
29844	NaN	NaN	NaN	49.90	NaN	NaN
29845	77.5	82.5	87.5	87.50	40.0	42.5

	Bench3Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	\
29842	NaN	20.41	NaN	NaN	NaN	
29843	NaN	18.14	NaN	NaN	NaN	
29844	NaN	29.48	NaN	NaN	NaN	
29845	-45.0	42.50	87.5	92.5	102.5	

	Best3DeadliftKg	TotalKg	DateOfBirthInterval
29842	45.36	86.18	[1990-08-01, 1991-07-31]
29843	52.16	102.06	[1990-08-07, 1991-08-06]
29844	68.04	147.42	[1990-07-28, 1991-07-27]
29845	102.50	232.50	[1995-01-01, 1995-12-31]

Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.

Type 'Q' to quit, or 'L' to save the lifter for later: y

Enter the row indicies for this disambiguation, separated by a space: 29845

	Name	Federation	Event	Date	MeetCountry	Event	\
29845	Brittany Russell	USAPL	SBD	2020-11-07	USA	SBD	

	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	\
29845	FR-0	Raw	59.8	24.5	24-34	24-39	

	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	\
29845	77.5	82.5	87.5	87.5	40.0	42.5	

	Bench3Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	\
29845	-45.0	42.5	87.5	92.5	102.5	

	Best3DeadliftKg	TotalKg	DateOfBirthInterval
29845	102.5	232.5	[1995-01-01, 1995-12-31]

Are you happy with this? Enter Y if yes, or N if no: y

	Name	Federation	Event	Date	MeetCountry	Event	\
29842	Brittany Russell	AAU	SBD	1998-07-31	USA	SBD	
29843	Brittany Russell	AAU	SBD	1999-08-06	USA	SBD	
29844	Brittany Russell	AAU	SBD	2001-07-27	USA	SBD	

	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	\
29842	Youth	Under 11	Single-ply	44.0	7.0	5-12	NaN
29843		RY 8-9	Raw	30.0	8.0	5-12	NaN
29844		RY 10-11	Raw	40.0	10.0	5-12	NaN

	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	\
29842	NaN	NaN	NaN	20.41	NaN	NaN	

29843	NaN	NaN	NaN	31.75	NaN	NaN
29844	NaN	NaN	NaN	49.90	NaN	NaN

	Bench3Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	\
29842	NaN	20.41	NaN	NaN	NaN	
29843	NaN	18.14	NaN	NaN	NaN	
29844	NaN	29.48	NaN	NaN	NaN	

	Best3DeadliftKg	TotalKg	DateOfBirthInterval			
29842	45.36	86.18	[1990-08-01, 1991-07-31]			
29843	52.16	102.06	[1990-08-07, 1991-08-06]			
29844	68.04	147.42	[1990-07-28, 1991-07-27]			

Can this lifter be further disambiguated? Press Y if they can, press N if they cannot.

Type 'Q' to quit, or 'L' to save the lifter for later: n
5464 names left

```
C:\Users\Montel\anaconda3\lib\site-packages\pandas\core\reshape\merge.py:1113:
UserWarning: You are merging on int and float columns where the float values are
not equal to their int representation
    warnings.warn(
```

	Name	Federation	Event	Date	MeetCountry	Event	Division	\
29863	Brittany Smith	USAPL	SBD	2018-02-24	USA	SBD	FR-O	
29864	Brittany Smith	USAPL	SBD	2018-04-21	USA	SBD	FR-O	
29865	Brittany Smith	USAPL	SBD	2018-09-08	USA	SBD	FR-O	
29866	Brittany Smith	USAPL	SBD	2018-11-24	USA	SBD	FR-O	

	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	Squat1Kg	\
29863	Raw	88.9	29.5	24-34	24-39	100.0	
29864	Raw	78.6	37.5	35-39	24-39	115.0	
29865	Raw	87.1	29.5	24-34	24-39	105.0	
29866	Raw	90.5	29.5	24-34	24-39	-105.0	

	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	\
29863	105.0	110.0	110.0	52.5	57.5	-65.0	
29864	125.0	135.0	135.0	57.5	62.5	-65.0	
29865	112.5	120.0	120.0	57.5	-60.0	-60.0	
29866	107.5	122.5	122.5	50.0	55.0	60.0	

	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Best3DeadliftKg	\
29863	57.5	137.5	142.5	152.5	152.5	
29864	62.5	120.0	135.0	-142.5	135.0	
29865	57.5	147.5	155.0	162.5	162.5	
29866	60.0	147.5	160.0	170.0	170.0	

	TotalKg	DateOfBirthInterval				
29863	320.0	[1988-01-01, 1988-12-31]				

29864	332.5	[1980-01-01, 1980-12-31]
29865	340.0	[1988-01-01, 1988-12-31]
29866	352.5	[1988-01-01, 1988-12-31]

	Name	Federation	Event	Date	MeetCountry	Event	Division	\
29859	Brittany Smith	THSWPA	SBD	2014-01-18	USA	SBD	Girls	
29860	Brittany Smith	THSWPA	SBD	2014-01-25	USA	SBD	Girls	
29861	Brittany Smith	THSWPA	SBD	2014-02-08	USA	SBD	Girls	
29862	Brittany Smith	THSWPA	SBD	2014-02-15	USA	SBD	Girls	
29867	Brittany Smith	APA	SBD	2019-06-22	USA	SBD	Veterans	
29868	Brittany Smith	APA	SBD	2019-09-07	USA	SBD	Open	

	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	Squat1Kg	\
29859	Single-ply	86.23	NaN	13-19	13-19	NaN	
29860	Single-ply	85.64	NaN	13-19	13-19	NaN	
29861	Single-ply	85.96	NaN	13-19	13-19	NaN	
29862	Single-ply	85.00	NaN	13-19	13-19	NaN	
29867	Wraps	77.56	30.0	24-34	24-39	83.91	
29868	Raw	78.90	31.0	24-34	24-39	85.00	

	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	\
29859	NaN	NaN	136.08	NaN	NaN	NaN	
29860	NaN	NaN	158.76	NaN	NaN	NaN	
29861	NaN	NaN	158.76	NaN	NaN	NaN	
29862	NaN	NaN	163.29	NaN	NaN	NaN	
29867	92.99	97.52	97.52	52.16	56.7	61.23	
29868	92.50	100.00	100.00	57.50	62.5	-67.50	

	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Best3DeadliftKg	\
29859	54.43	NaN	NaN	NaN	131.54	
29860	56.70	NaN	NaN	NaN	124.74	
29861	56.70	NaN	NaN	NaN	127.01	
29862	61.23	NaN	NaN	NaN	122.47	
29867	61.23	90.72	99.79	108.86	108.86	
29868	62.50	95.00	102.50	110.00	110.00	

	TotalKg	DateOfBirthInterval
29859	322.05	[1994-01-18, 2001-01-18]
29860	340.19	[1994-01-25, 2001-01-25]
29861	342.46	[1994-02-08, 2001-02-08]
29862	347.00	[1994-02-15, 2001-02-15]
29867	267.62	[1988-06-23, 1989-06-22]
29868	272.50	[1987-09-08, 1988-09-07]

Look at the last two dataframes. Enter 'y' if some indicies from df2 can go into df1, 'x' if all indicies can go from df2 to df1, 'n' if no indicies can go from df2 to df1,

'l' to save for later, 'f' if incorrect, or 'q' to quit.l

5464 names left

C:\Users\Montel\anaconda3\lib\site-packages\pandas\core\reshape\merge.py:1113:
UserWarning: You are merging on int and float columns where the float values are
not equal to their int representation

```
warnings.warn(
```

	Name	Federation	Event	Date	MeetCountry	Event	Division	\
29889	Brittany Turner	USAPL	SBD	2019-10-19	USA	SBD	FR-0	
	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	Squat1Kg		\
29889	Raw	80.5	27.5	24-34	24-39	115.0		
	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg		\
29889	122.5	127.5	127.5	67.5	70.0	-75.0		
	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Best3DeadliftKg			\
29889	70.0	145.0	155.0	165.0	165.0			
	TotalKg		DateOfBirthInterval					
29889	362.5	[1991-01-01, 1991-12-31]						
	Name	Federation	Event	Date	MeetCountry	Event	Division	\
29883	Brittany Turner	RAW	SBD	2017-08-19	USA	SBD	Open	
29884	Brittany Turner	RAW	SBD	2018-03-17	USA	SBD	Open	
29885	Brittany Turner	USPA	SBD	2018-07-07	USA	SBD	Open	
29886	Brittany Turner	RAW	SBD	2018-10-11	USA	SBD	Open	
29887	Brittany Turner	USPA	SBD	2019-01-13	USA	SBD	Open	
29888	Brittany Turner	USPA	SBD	2019-07-11	USA	SBD	Open	
29890	Brittany Turner	USPA	SBD	2020-02-15	USA	SBD	Open	
	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	Squat1Kg		\
29883	Raw	81.19	26.0	24-34	24-39	NaN		
29884	Raw	73.98	68.0	65-69	60-69	NaN		
29885	Raw	75.25	27.0	24-34	24-39	NaN		
29886	Raw	79.10	27.0	24-34	24-39	NaN		
29887	Raw	80.00	27.0	24-34	24-39	NaN		
29888	Raw	78.70	28.0	24-34	24-39	NaN		
29890	Raw	80.30	28.0	24-34	24-39	115.0		
	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg		\
29883	NaN	NaN	92.5	NaN	NaN	NaN		
29884	NaN	NaN	105.0	NaN	NaN	NaN		
29885	NaN	NaN	112.5	NaN	NaN	NaN		
29886	NaN	NaN	115.0	NaN	NaN	NaN		
29887	NaN	NaN	117.5	NaN	NaN	NaN		
29888	NaN	NaN	125.0	NaN	NaN	NaN		
29890	125.0	-127.5	125.0	65.0	72.5	75.0		

	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Best3DeadliftKg	\
29883	57.5	NaN	NaN	NaN	132.5	
29884	60.0	NaN	NaN	NaN	132.5	
29885	62.5	NaN	NaN	NaN	130.0	
29886	65.0	NaN	NaN	NaN	137.5	
29887	67.5	NaN	NaN	NaN	142.5	
29888	72.5	NaN	NaN	NaN	150.0	
29890	75.0	150.0	160.0	-167.5	160.0	

	TotalKg	DateOfBirthInterval
29883	282.5	[1990-08-20, 1991-08-19]
29884	297.5	[1949-03-18, 1950-03-17]
29885	305.0	[1990-07-08, 1991-07-07]
29886	317.5	[1990-10-12, 1991-10-11]
29887	327.5	[1991-01-14, 1992-01-13]
29888	347.5	[1990-07-12, 1991-07-11]
29890	360.0	[1991-02-16, 1992-02-15]

Look at the last two dataframes. Enter 'y' if some indicies from df2 can go into
df1, 'x' if all indicies can go from df2 to df1,
'n' if no indicies can go from df2 to df1,
'l' to save for later, 'f' if incorrect, or 'q'
to quit.

5464 names left

	Name	Federation	Event	Date	MeetCountry	Event	\
29896	Brittany Williams	USAPL	SBD	2015-03-27	USA	SBD	
29897	Brittany Williams	USAPL	BD	2016-02-13	USA	BD	
29898	Brittany Williams	USAPL	SBD	2016-04-01	USA	SBD	
29899	Brittany Williams	USAPL	SBD	2017-03-29	USA	SBD	

	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	\
29896	F-TJ	Single-ply	83.1	15.5	16-17	14-18	
29897	FR-V	Raw	46.3	16.5	16-17	14-18	
29898	F-V	Single-ply	78.4	16.5	16-17	14-18	
29899	F-TJ	Single-ply	80.5	17.5	18-19	14-18	

	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	\
29896	135.0	147.5	-155.0	147.5	62.5	70.0	
29897	NaN	NaN	NaN	NaN	20.0	-25.0	
29898	177.5	180.0	182.5	182.5	82.5	95.0	
29899	165.0	182.5	200.0	200.0	82.5	95.0	

	Bench3Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	\
29896	75.0	75.0	137.5	-150.0	152.5	
29897	-25.0	20.0	60.0	65.0	72.5	
29898	-97.5	95.0	160.0	170.0	-185.0	
29899	-100.0	95.0	150.0	172.5	185.0	

	Best3DeadliftKg	TotalKg	DateOfBirthInterval					
29896	152.5	375.0	[1999-01-01, 1999-12-31]					
29897	72.5	92.5	[1999-01-01, 1999-12-31]					
29898	170.0	447.5	[1999-01-01, 1999-12-31]					
29899	185.0	480.0	[1999-01-01, 1999-12-31]					
	Name	Federation	Event	Date	MeetCountry	Event	\	
29895	Brittany Williams	USAPL	SBD	2015-02-13	USA	SBD		
29900	Brittany Williams	USPA	D	2017-12-03	USA	D		
29901	Brittany Williams	USPA	SBD	2018-07-29	USA	SBD		
29902	Brittany Williams	USPA	SBD	2018-12-08	USA	SBD		
29903	Brittany Williams	USPA	SBD	2019-12-07	USA	SBD		
29904	Brittany Williams	USPA	SBD	2020-09-19	USA	SBD		
29905	Brittany Williams	USPA	SBD	2021-05-08	USA	SBD		
	Division	Equipment	BodyweightKg	Age	AgeClass	BirthYearClass	\	
29895	F-TJ	Single-ply	82.0	15.5	16-17	14-18		
29900	Open	Raw	93.2	25.0	24-34	24-39		
29901	Open	Raw	89.2	25.0	24-34	24-39		
29902	Open	Raw	88.0	26.0	24-34	24-39		
29903	Open	Raw	82.4	27.0	24-34	24-39		
29904	Open	Raw	80.9	27.0	24-34	24-39		
29905	Open	Raw	82.1	28.0	24-34	24-39		
	Squat1Kg	Squat2Kg	Squat3Kg	Best3SquatKg	Bench1Kg	Bench2Kg	\	
29895	NaN	NaN	NaN	138.3	NaN	NaN		
29900	NaN	NaN	NaN	NaN	NaN	NaN		
29901	NaN	NaN	NaN	102.5	NaN	NaN		
29902	NaN	NaN	NaN	117.5	NaN	NaN		
29903	-105.0	105.0	117.5	117.5	55.0	-60.0		
29904	110.0	117.5	125.0	125.0	55.0	60.0		
29905	120.0	-130.0	130.0	130.0	60.0	65.0		
	Bench3Kg	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	\		
29895	NaN	65.8	NaN	NaN	NaN			
29900	NaN	NaN	NaN	NaN	NaN			
29901	NaN	47.5	NaN	NaN	NaN			
29902	NaN	50.0	NaN	NaN	NaN			
29903	-60.0	55.0	132.5	142.5	-145.0			
29904	65.0	65.0	137.5	147.5	-150.0			
29905	-70.0	65.0	147.5	157.5	-167.5			
	Best3DeadliftKg	TotalKg	DateOfBirthInterval					
29895	138.3	342.4	[1999-01-01, 1999-12-31]					
29900	127.5	127.5	[1991-12-04, 1992-12-03]					
29901	130.0	280.0	[1992-07-30, 1993-07-29]					
29902	140.0	307.5	[1991-12-09, 1992-12-08]					
29903	142.5	315.0	[1991-12-08, 1992-12-07]					

```
29904      147.5    337.5 [1992-09-20, 1993-09-19]  
29905      157.5    352.5 [1992-05-09, 1993-05-08]
```

```
Look at the last two dataframes. Enter 'y' if some indicies from df2 can go into  
df1, 'x' if all indicies can go from df2 to df1,  
'n' if no indicies can go from df2 to df1,  
'l' to save for later, 'f' if incorrect, or 'q'  
to quit.q  
5464 names left
```

```
[ ]:
```

Appendix E Removing non-disambiguated lifters, and non-USAPL lifters from the dataset

Appendix_E_Dataframe Reduction - Removing names that do not match criteria - use this every time i progress with cleaning data

September 9, 2021

```
[11]: import time

[8]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json
import copy
import random
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', 500)

openpowerlifting_USAPL_lifters = pd.
    ↪read_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    ↪to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')

#reconverting date of birth interval from list to interval
for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open" ↪
        ↪and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        list_date = openpowerlifting_USAPL_lifters.at[i, ↪
            ↪"DateOfBirthInterval"][1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd.
            ↪Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed = ↪
            ↪"both")

name_list = np.load("name disambiguation/remainingnames.npy")
name_list = np.sort(name_list)#[::-1]
new_names = np.load("name disambiguation/new_lifter_names.npy")
original_dataset_names = json.load(open("name disambiguation/
    ↪original_dataset_names.json"))
new_dataset_names = json.load(open("name disambiguation/new_dataset_names.
    ↪json"))
name_list_in_progress = copy.deepcopy(name_list)
```

```
[28]: print(f"length of current dataframe : {len(openpowerlifting_USAPL_lifters)}")
length =_
↪len(openpowerlifting_USAPL_lifters[~openpowerlifting_USAPL_lifters["Name"] .
↪isin(name_list)])
print(f"length of new dataframe : {length}")
openpowerlifting_USAPL_lifters =_
↪openpowerlifting_USAPL_lifters[~openpowerlifting_USAPL_lifters["Name"] .
↪isin(name_list)]
```

Timer starting
length of current dataframe : 180320
length of new dataframe : 142273

```
[ ]: all_lifter_names = openpowerlifting_USAPL_lifters["Name"].drop_duplicates()

openpowerlifting_USAPL_lifters = openpowerlifting_USAPL_lifters[
    openpowerlifting_USAPL_lifters["Name"].isin(all_lifter_names)]
```

```
[31]: openpowerlifting_USAPL_lifters = openpowerlifting_USAPL_lifters.
    ↪groupby(["Name"]).filter(lambda x: len(x[(x["Federation"] == "USAPL") \& (x["Event"] == "SBD") \& (x["Equipment"] == "Raw") \& (pd.notna(x["Age"])) \& (x["Date"] > pd.to_datetime("01/01/2014"))]) > 0) # at least one per name
    ↪that does not contain
```

```
[41]: openpowerlifting_USAPL_lifters.
    ↪to_csv("open_powerlifting_USAPL_Raw_active_lifters_disambiguated_ready_for_analysis.csv", index = False)
```

[]:

Appendix F Feature Generation Scripts

- a. Time since last meet

Appendix_F_Time since last meet

September 10, 2021

```
[102]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json
import copy
import random
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', 500)

openpowerlifting_USAPL_lifters = pd.read_csv("C:/Users/Montel/Google Drive/Data_"
→Science MSc/Msc-Project---Powerlifting/Datasets/
→open_powerlifting_USAPL_Raw_active_lifters_disambiguated_ready_for_analysis.
→csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
→to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')

#reconverting date of birth interval from list to interval
for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open" ↳
    →and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        list_date = openpowerlifting_USAPL_lifters.at[i, ↳
        →"DateOfBirthInterval"][1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd.
        →Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed = ↳
        →"both")
```

creating new column for time since last meet

```
[124]: openpowerlifting_USAPL_lifters["Time Since Last Meet"] = np.nan

[128]: openpowerlifting_USAPL_lifters["Time Since Last Meet"] = ↳
→openpowerlifting_USAPL_lifters.groupby("Name")["Date"].diff()

[145]: openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Time Since Last_"
→Meet"] == pd.to_timedelta(0, unit='days')].shape

[145]: (184, 45)
```

```
[146]: openpowerlifting_USAPL_lifters.to_csv("C:/Users/Montel/Google Drive/Data_U  
↳Science MSc/Msc-Project---Powerlifting/Datasets/  
↳open_powerlifting_USAPL_Raw_active_lifters_disambiguated_ready_for_analysis.  
↳csv", index = False)
```

```
[144]: import math

# function to truncate meets where single lifts have been split

def male_wilks_coefficients():
    return (-216.0475144, 16.2606339, -0.002388645, -0.00113732, 7.  
→01863*10**-6, -1.291*10**-8)
def female_wilks_coefficients():
    return (594.31747775582, -27.23842536447, 0.82112226871, -0.00930733913, 4.  
→731582*10**-5, -9.054*10**-8)

openpowerlifting_USAPL_lifters["Time Since Last Meet"] =  
→openpowerlifting_USAPL_lifters.groupby("Name")["Date"].diff()
pair_list = []
for i in openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Time_U  
→Since Last Meet"] == pd.to_timedelta(0, unit='days')].index:
    pair_list.append((i-1, i))

for i,j in pair_list:

    try:
        meet_1 = openpowerlifting_USAPL_lifters.loc[i]
        competition_1 = meet_1["MeetName"]
        date_1 = meet_1["Date"]
        equipment_1 = meet_1["Equipment"]
        name_1 = meet_1["Name"]
        event_1 = meet_1["Event"]
        bodyweight_1 = meet_1["BodyweightKg"]
        bestsquat_1 = meet_1["Best3SquatKg"]
        bestbench_1 = meet_1["Best3BenchKg"]
        bestdead_1 = meet_1["Best3DeadliftKg"]
        meet_2 = openpowerlifting_USAPL_lifters.loc[j]
        competition_2 = meet_2["MeetName"]
        date_2 = meet_2["Date"]
        equipment_2 = meet_2["Equipment"]
        name_2 = meet_2["Name"]
        event_2 = meet_2["Event"]
        bodyweight_2 = meet_2["BodyweightKg"]
        bestsquat_2 = meet_2["Best3SquatKg"]
        bestbench_2 = meet_2["Best3BenchKg"]
        bestdead_2 = meet_2["Best3DeadliftKg"]

    except:
```

```

        continue
    if math.isclose(bodyweight_1, bodyweight_2, rel_tol = 0.01) and date_1 ==_
→date_2:
    print(name_1)

    if date_1 == date_2 and name_1 == name_2 and math.isclose(bodyweight_1,_
→bodyweight_2, rel_tol = 0.01) and competition_1 == competition_2 and_
→equipment_1 == equipment_2:
        if event_1 == "S":
            if event_2 == "B":

                openpowerlifting_USAPL_lifters.at[i, "Event"] = "SB"
                for k in ["Bench1Kg", "Bench2Kg", "Bench3Kg", "Bench4Kg",_
→"Best3BenchKg"]:
                    openpowerlifting_USAPL_lifters.at[i, k] =_
→openpowerlifting_USAPL_lifters.at[j, k]
                    if pd.notna(openpowerlifting_USAPL_lifters.at[i,_
→"Best3BenchKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i,_
→"Best3BenchKg"]:
                        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] +=_
→openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]
                        if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                            a, b, c, d, e, f = male_wilks_coefficients()
                            bw = bodyweight_1
                            openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] *_
→bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                        else:
                            a, b, c, d, e, f = female_wilks_coefficients()
                            bw = bodyweight_1
                            openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] *_
→bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                            openpowerlifting_USAPL_lifters.drop(j, inplace = True)

        elif event_2 == "D":
            openpowerlifting_USAPL_lifters.at[i, "Event"] = "SD"
            for k in ["Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg",_
→"Deadlift4Kg", "Best3DeadliftKg"]:
                openpowerlifting_USAPL_lifters.at[i, k] =_
→openpowerlifting_USAPL_lifters.at[j, k]
                if pd.notna(openpowerlifting_USAPL_lifters.at[i,_
→"Best3DeadliftKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i,_
→"Best3DeadliftKg"]:
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] +=_
→openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]

```

```

        if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
            a, b, c, d, e, f = male_wilks_coefficients()
            bw = bodyweight_1
            openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * \
                bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
        else:
            a, b, c, d, e, f = female_wilks_coefficients()
            bw = bodyweight_1
            openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * \
                bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
            openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    elif event_2 == "S" and bestsquat_1 == bestsquat_2:
        openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    if event_1 == "S":
        if event_2 == "B":

            openpowerlifting_USAPL_lifters.at[i, "Event"] = "SB"
            for k in ["Bench1Kg", "Bench2Kg", "Bench3Kg", "Bench4Kg", \
            "Best3BenchKg"]:
                openpowerlifting_USAPL_lifters.at[i, k] = \
                openpowerlifting_USAPL_lifters.at[j, k]
                if pd.notna(openpowerlifting_USAPL_lifters.at[i, \
                "Best3BenchKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, \
                "Best3BenchKg"]:
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += \
                    openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]
                    if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                        a, b, c, d, e, f = male_wilks_coefficients()
                        bw = bodyweight_1
                        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * \
                            bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                    else:
                        a, b, c, d, e, f = female_wilks_coefficients()
                        bw = bodyweight_1
                        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * \
                            bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                    openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    elif event_2 == "D":
        openpowerlifting_USAPL_lifters.at[i, "Event"] = "SD"

```

```

        for k in ["Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", ↴
→"Deadlift4Kg", "Best3DeadliftKg"]:
            openpowerlifting_USAPL_lifters.at[i, k] = ↴
→openpowerlifting_USAPL_lifters.at[j, k]
            if pd.notna(openpowerlifting_USAPL_lifters.at[i, ↴
→"Best3DeadliftKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, ↴
→"Best3DeadliftKg"]:
                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += ↴
→openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]
                if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                    a, b, c, d, e, f = male_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * ↴
→bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                else:
                    a, b, c, d, e, f = female_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * ↴
→bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
            openpowerlifting_USAPL_lifters.drop(j, inplace = True)

        elif event_2 == "S" and bestsquat_1 == bestsquat_2:
            openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    elif event_1 == "B":
        if event_2 == "S":
            openpowerlifting_USAPL_lifters.at[i, "Event"] = "SB"
            for k in ["Squat1Kg", "Squat2Kg", "Squat3Kg", "Squat4Kg", ↴
→"Best3SquatKg"]:
                openpowerlifting_USAPL_lifters.at[i, k] = ↴
→openpowerlifting_USAPL_lifters.at[j, k]
                if pd.notna(openpowerlifting_USAPL_lifters.at[i, ↴
→"Best3SquatKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, ↴
→"Best3SquatKg"]:
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += ↴
→openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]
                    if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                        a, b, c, d, e, f = male_wilks_coefficients()
                        bw = bodyweight_1
                        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * ↴
→bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                    else:

```

```

        a, b, c, d, e, f = female_wilks_coefficients()
        bw = bodyweight_1
        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * \
        ↵bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
        openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    elif event_2 == "D":
        openpowerlifting_USAPL_lifters.at[i, "Event"] = "BD"
        for k in ["Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", \
        ↵"Deadlift4Kg", "Best3DeadliftKg"]:
            openpowerlifting_USAPL_lifters.at[i, k] = \
        ↵openpowerlifting_USAPL_lifters.at[j, k]
            if pd.notna(openpowerlifting_USAPL_lifters.at[i, \
        ↵"Best3DeadliftKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, \
        ↵"Best3DeadliftKg"]:
                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += \
        ↵openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]
                if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                    a, b, c, d, e, f = male_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * \
        ↵bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                else:
                    a, b, c, d, e, f = female_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * \
        ↵bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                    openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    elif event_2 == "B" and bestbench_1 == bestbench_2:
        openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    elif event_1 == "D":
        if event_2 == "S":
            openpowerlifting_USAPL_lifters.at[i, "Event"] = "SD"
            for k in ["Squat1Kg", "Squat2Kg", "Squat3Kg", "Squat4Kg", \
            ↵"Best3SquatKg"]:
                openpowerlifting_USAPL_lifters.at[i, k] = \
            ↵openpowerlifting_USAPL_lifters.at[j, k]
                if pd.notna(openpowerlifting_USAPL_lifters.at[i, \
            ↵"Best3SquatKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, \
            ↵"Best3SquatKg"]:

```

```

        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += ↵
↪openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]
        if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
            a, b, c, d, e, f = male_wilks_coefficients()
            bw = bodyweight_1
            openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * ↵
↪bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
        else:
            a, b, c, d, e, f = female_wilks_coefficients()
            bw = bodyweight_1
            openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * ↵
↪bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
            openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    elif event_2 == "B":
        openpowerlifting_USAPL_lifters.at[i, "Event"] = "BD"
        for k in ["Bench1Kg", "Bench2Kg", "Bench3Kg", "Bench4Kg", ↵
↪"Best3BenchKg"]:
            openpowerlifting_USAPL_lifters.at[i, k] = ↵
↪openpowerlifting_USAPL_lifters.at[j, k]
            if pd.notna(openpowerlifting_USAPL_lifters.at[i, ↵
↪"Best3BenchKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, ↵
↪"Best3BenchKg"]:
                openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += ↵
↪openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]
                if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                    a, b, c, d, e, f = male_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * 500/ ↵
↪(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                else:
                    a, b, c, d, e, f = female_wilks_coefficients()
                    bw = bodyweight_1
                    openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * 500/ ↵
↪(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    elif event_2 == "D" and bestdead_1 == bestdead_2:
        openpowerlifting_USAPL_lifters.drop(j, inplace = True)

```

Allison Hind
Alma Kimura
Alvin Roswell

Amy Poplata
Amy Suzan
Andrea Pettyjohn
Andrew Sellers
Anthony Calhoun
Antonio Grogan
April Waller
Ashley Contorno
Bebe Burns
Bebe Burns
Bebe Burns
Bebe Burns
Bebe Burns
Benjamin Rowe
Bill Duncan
Bill Helmich
Binglei Zhou
Blaine Sumner
Blaine Sumner
Blaine Sumner
Bonica Brown
Bonica Brown
Brandon Maddox #1
Brandon Rojo
Brenda Kuhns
Brett Kempski
Brian Miller
Brian Morrison
Broden Thompson
Bruce Barry
Bruce Barry
Bruce Barry
Bruce Barry
Bruce Barry
Bryan Green #1
Cardell Oliver
Casey Landry
Caycee Bregel
Charleen Balcer Rowekamp
Charleen Balcer Rowekamp
Charleen Balcer Rowekamp
Charleen Balcer Rowekamp
Chelsie Hustad
Chelsie Hustad
Cheryl Edelstein
Chris Anderson #1
Chuck Akers
Cindy Amatuzzo

Cindy Amatuzzo
Cindy Goodrich
Clint Poore
Clint Poore
Clint Poore
Clint Poore
Craig Rasmussen
Craig Rasmussen
Cynthia Line
Damian Fronzaglia
Damian Fronzaglia
Daniel Gonzalez
Daniel Thurman
Daniel Thurman
Danna Snow
Darren Gillett
David Coxson
David Muñoz
David Wolf
David Wolf
Dennis Cannataro
Dennis Cieri
Dennis Cieri
Dennis Cieri
Dennis Cieri
Dennis Cieri
Dennis Cornelius
Denny Lawrence
Derek LoGrande
Donnie Haddock
Donnie Haddock Sr
Donovan Thompson
Donovan Thompson
Donovan Thompson
Donovan Thompson
Donovan Thompson
Donovan Thompson
Donovan Thompson

```
Dora Gonzalez #1
Easton Schuster
Easton Schuster
Easton Schuster
Edward Taylor #5
Edwin Nash
Elizabeth Richardson
Elizabeth Richardson
Eric Kupperstein
John Caruso
Khemarintr Suwanchote
Mike Burns
Mike Burns
Mike Burns
Mike Burns
Palmer Griffin
Robert Mullener
Susan Elwyn
Susan Elwyn
Xaviar Clark
Yarnell Marks
```

```
[100]:
```

```
[100]: 57.5
```

```
[143]: import math
```

```
# function to truncate meets where single lifts have been split

def male_wilks_coefficients():
    return (-216.0475144, 16.2606339, -0.002388645, -0.00113732, 7.
    ↪01863*10**-6, -1.291*10**-8)
def female_wilks_coefficients():
    return (594.31747775582, -27.23842536447, 0.82112226871, -0.00930733913, 4.
    ↪731582*10**-5, -9.054*10**-8)

openpowerlifting_USAPL_lifters["Time Since Last Meet"] = ↪
    ↪openpowerlifting_USAPL_lifters.groupby("Name")["Date"].diff()
pair_list = []
for i in openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Time Since Last Meet"] == pd.to_timedelta(0, unit='days')].index:
    pair_list.append((i-1, i))

for i,j in pair_list:
    try:
```

```

meet_1 = openpowerlifting_USAPL_lifters.loc[i]
competition_1 = meet_1["MeetName"]
date_1 = meet_1["Date"]
equipment_1 = meet_1["Equipment"]
name_1 = meet_1["Name"]
event_1 = meet_1["Event"]
bodyweight_1 = meet_1["BodyweightKg"]
bestsquat_1 = meet_1["Best3SquatKg"]
bestbench_1 = meet_1["Best3BenchKg"]
bestdead_1 = meet_1["Best3DeadliftKg"]
meet_2 = openpowerlifting_USAPL_lifters.loc[j]
competition_2 = meet_2["MeetName"]
date_2 = meet_2["Date"]
equipment_2 = meet_2["Equipment"]
name_2 = meet_2["Name"]
event_2 = meet_2["Event"]
bodyweight_2 = meet_2["BodyweightKg"]
bestsquat_2 = meet_2["Best3SquatKg"]
bestbench_2 = meet_2["Best3BenchKg"]
bestdead_2 = meet_2["Best3DeadliftKg"]

except:
    continue
if math.isclose(bodyweight_1, bodyweight_2, rel_tol = 0.01) and date_1 == date_2:
    print(name_1)

    if date_1 == date_2 and name_1 == name_2 and math.isclose(bodyweight_1, bodyweight_2, rel_tol = 0.01) and competition_1 == competition_2 and equipment_1 == equipment_2:
        if event_1 == "SD":
            if event_2 == "B":

                openpowerlifting_USAPL_lifters.at[i, "Event"] = "SBD"
                for k in ["Bench1Kg", "Bench2Kg", "Bench3Kg", "Bench4Kg", "Best3BenchKg"]:
                    openpowerlifting_USAPL_lifters.at[i, k] = openpowerlifting_USAPL_lifters.at[j, k]
                    if pd.notna(openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]:
                        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += openpowerlifting_USAPL_lifters.at[i, "Best3BenchKg"]
                        if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                            a, b, c, d, e, f = male_wilks_coefficients()
                            bw = bodyweight_1
                            openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \

```

```

        openpowerlifting_USAPL_lifters.at[i, "TotalKg"] *=
→bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
    else:
        a, b, c, d, e, f = female_wilks_coefficients()
        bw = bodyweight_1
        openpowerlifting_USAPL_lifters.at[i, "Wilks"] =
openpowerlifting_USAPL_lifters.at[i, "TotalKg"] *=
→bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
        openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    if event_1 == "SB":
        if event_2 == "D":
            openpowerlifting_USAPL_lifters.at[i, "Event"] = "SBD"
            for k in ["Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", u
→"Deadlift4Kg", "Best3DeadliftKg"]:
                openpowerlifting_USAPL_lifters.at[i, k] =
→openpowerlifting_USAPL_lifters.at[j, k]
                if pd.notna(openpowerlifting_USAPL_lifters.at[i, u
→"Best3DeadliftKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, u
→"Best3DeadliftKg"]:
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] +==
→openpowerlifting_USAPL_lifters.at[i, "Best3DeadliftKg"]
                    if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                        a, b, c, d, e, f = male_wilks_coefficients()
                        bw = bodyweight_1
                        openpowerlifting_USAPL_lifters.at[i, "Wilks"] =
openpowerlifting_USAPL_lifters.at[i, "TotalKg"] *=
→bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                    else:
                        a, b, c, d, e, f = female_wilks_coefficients()
                        bw = bodyweight_1
                        openpowerlifting_USAPL_lifters.at[i, "Wilks"] =
openpowerlifting_USAPL_lifters.at[i, "TotalKg"] *=
→bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
                    openpowerlifting_USAPL_lifters.drop(j, inplace = True)

    elif event_1 == "BD":
        if event_2 == "S":
            openpowerlifting_USAPL_lifters.at[i, "Event"] = "SBD"
            for k in ["Squat1Kg", "Squat2Kg", "Squat3Kg", "Squat4Kg", u
→"Best3SquatKg"]:
                openpowerlifting_USAPL_lifters.at[i, k] =
→openpowerlifting_USAPL_lifters.at[j, k]

```

```

        if pd.notna(openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]) and 0 < openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]:
            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] += openpowerlifting_USAPL_lifters.at[i, "Best3SquatKg"]
            if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
                a, b, c, d, e, f = male_wilks_coefficients()
                bw = bodyweight_1
                openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * \
                    bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
            else:
                a, b, c, d, e, f = female_wilks_coefficients()
                bw = bodyweight_1
                openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
                    openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * \
                    bodyweight_1*500/(a + b*bw + c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
        openpowerlifting_USAPL_lifters.drop(j, inplace = True)
    
```

Allison Hind
 Alma Kimura
 Alvin Roswell
 Amy Poplata
 Amy Suzan
 Andrea Pettyjohn
 Andrew Sellers
 Anthony Calhoun
 Antonio Grogan
 April Waller
 Ashley Contorno
 Bebe Burns
 Bebe Burns
 Bebe Burns
 Bebe Burns
 Bebe Burns
 Benjamin Rowe
 Bill Duncan
 Bill Helmich
 Binglei Zhou
 Blaine Sumner
 Blaine Sumner
 Blaine Sumner
 Bonica Brown
 Bonica Brown
 Brandon Maddox #1
 Brandon Rojo

Brenda Kuhns
Brett Kempski
Brian Miller
Brian Morrison
Broden Thompson
Bruce Barry
Bruce Barry
Bruce Barry
Bruce Barry
Bruce Barry
Bruce Barry
Bryan Green #1
Cardell Oliver
Casey Landry
Caycee Bregel
Charleen Balcer Rowekamp
Charleen Balcer Rowekamp
Charleen Balcer Rowekamp
Charleen Balcer Rowekamp
Chelsie Hustad
Chelsie Hustad
Cheryl Edelstein
Chris Anderson #1
Chuck Akers
Cindy Amatuzzo
Cindy Amatuzzo
Cindy Goodrich
Clint Poore
Clint Poore
Clint Poore
Clint Poore
Craig Rasmussen
Craig Rasmussen
Cynthia Line
Damian Fronzaglia
Damian Fronzaglia
Daniel Gonzalez
Daniel Thurman
Daniel Thurman
Danna Snow
Darren Gillett
David Coxson
David Muñoz
David Wolf
David Wolf
Dennis Cannataro
Dennis Cannataro
Dennis Cannataro
Dennis Cannataro

Dennis Cannataro
Dennis Cannataro
Dennis Cannataro
Dennis Cannataro
Dennis Cannataro
Dennis Cannataro
Dennis Cannataro
Dennis Cannataro
Dennis Cannataro
Dennis Cieri
Dennis Cieri
Dennis Cieri
Dennis Cieri
Dennis Cieri
Dennis Cieri
Dennis Cornelius
Denny Lawrence
Derek LoGrande
Donnie Haddock
Donnie Haddock Sr
Donovan Thompson
Dora Gonzalez #1
Easton Schuster
Easton Schuster
Easton Schuster
Edward Taylor #5
Edwin Nash
Elizabeth Richardson
Elizabeth Richardson
Eric Kupperstein
John Caruso
Khemarintr Suwanchote
Mike Burns
Mike Burns
Mike Burns
Mike Burns
Palmer Griffin
Robert Mullener
Susan Elwyn
Susan Elwyn
Xaviar Clark
Yarnell Marks

```
[ ]: openpowerlifting_USAPL_lifters.to_csv("C:/Users/Montel/Google Drive/Data_U  
↳Science MSc/Msc-Project---Powerlifting/Datasets/  
↳open_powerlifting_USAPL_Raw_active_lifters_disambiguated_ready_for_analysis.  
↳csv", index = False)
```

b. Number of past meets

Appendix_F_Number of past meets

September 10, 2021

```
[1]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json
import copy
import random
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', 500)

openpowerlifting_USAPL_lifters = pd.read_csv("C:/Users/Montel/Google Drive/Data_\
→Science MSc/Msc-Project---Powerlifting/Datasets/\
→open_powerlifting_USAPL_Raw_active_lifters_disambiguated_ready_for_analysis.\
→csv")
openpowerlifting_USAPL_lifters['Date'] = pd.\
→to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
openpowerlifting_USAPL_lifters["Time Since Last Meet"] = pd.\
→to_timedelta(openpowerlifting_USAPL_lifters["Time Since Last Meet"])

#reconverting date of birth interval from list to interval
for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open" \
→and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        list_date = openpowerlifting_USAPL_lifters.at[i, \
→"DateOfBirthInterval"][1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd.\
→Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed = \
→"both")
```

```
[2]: openpowerlifting_USAPL_lifters["Number of Past Meets"] = np.nan
openpowerlifting_USAPL_lifters["Number of Past Meets"] = \
→openpowerlifting_USAPL_lifters.groupby("Name")["Date"].cumcount()

openpowerlifting_USAPL_lifters["Number of Past Meets_Mod_Bool"] = (
    openpowerlifting_USAPL_lifters["Time Since Last Meet"] != pd.\
→to_timedelta(0, unit='days'))
```

```
openpowerlifting_USAPL_lifters["Number of Past Meets_Mod_Bool"] =  
    openpowerlifting_USAPL_lifters.groupby(  
        ["Name"])["Number of Past Meets_Mod_Bool"].cumsum() -1  
  
openpowerlifting_USAPL_lifters.rename(  
    columns = {"Number of Past Meets_Mod_Bool": "Number of Past Meets Same Day",  
               "Exclusive"}, inplace = True)
```

```
[42]: openpowerlifting_USAPL_lifters.to_csv("C:/Users/Montel/Google Drive/Data/  
    Science MSc/Msc-Project---Powerlifting/Datasets/  
    open_powerlifting_USAPL_Raw_active_lifters_disambiguated_ready_for_analysis.  
    csv", index = False)
```

```
[ ]:
```

c. Personal best squat, bench, deadlift, total and wilks score

Appendix_F_Personal Best Squat, Bench, Deadlift and Total

September 10, 2021

```
[10]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json
import copy
import random
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', 500)

openpowerlifting_USAPL_lifters = pd.read_csv("C:/Users/Montel/Google Drive/Data_
↪Science MSc/Msc-Project---Powerlifting/Datasets/
↪open_powerlifting_USAPL_Raw_active_lifters_disambiguated_ready_for_analysis.
↪csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
↪to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
openpowerlifting_USAPL_lifters["Time Since Last Meet"] = pd.
↪to_timedelta(openpowerlifting_USAPL_lifters["Time Since Last Meet"])
```

Rules: Now we will operate on the subset of lifters who have competed raw in the USAPL, full power since 2014 We want to find the best raw squat so far, the best raw bench so far, the best raw deadlift so far and the best total so far We also want to find the best single ply squat so far, best single ply

```
[11]: openpowerlifting_USAPL_lifters["Equipment"].value_counts()
```

```
[11]: Raw          109866
Single-ply    21974
Wraps         5772
Multi-ply     666
Unlimited      9
Name: Equipment, dtype: int64
```

```
[12]: def male_wilks_coefficients():
        return (-216.0475144, 16.2606339, -0.002388645, -0.00113732, 7.
↪01863*10**-6, -1.291*10**-8)
def female_wilks_coefficients():
```

```

    return (594.31747775582, -27.23842536447, 0.82112226871, -0.00930733913, 4.
→731582*10**-5, -9.054*10**-8)
for i in range(0, len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "Sex"] == "M":
        a, b, c, d, e, f = male_wilks_coefficients()
        bw = openpowerlifting_USAPL_lifters.at[i, "BodyweightKg"]
        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * 500/(a + b*bw +_
→c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)
    else:
        a, b, c, d, e, f = female_wilks_coefficients()
        bw = openpowerlifting_USAPL_lifters.at[i, "BodyweightKg"]
        openpowerlifting_USAPL_lifters.at[i, "Wilks"] = \
            openpowerlifting_USAPL_lifters.at[i, "TotalKg"] * 500/(a + b*bw +_
→c*bw**2 + d*bw**3 + e*bw**4 + f*bw**5)

```

method: group by name, then calculate the best value this array needs to be shifted down one for each group then ffill should be done for each group.

```
[13]: # Creating new columns
openpowerlifting_USAPL_lifters["Best Raw Squat to date"] =_
→openpowerlifting_USAPL_lifters[
    openpowerlifting_USAPL_lifters["Equipment"] == "Raw"] .
→groupby("Name")["Best3SquatKg"].cummax()

openpowerlifting_USAPL_lifters["Best Wrapped Squat to date"] =_
→openpowerlifting_USAPL_lifters[
    openpowerlifting_USAPL_lifters["Equipment"] == "Wraps"] .
→groupby("Name")["Best3SquatKg"].cummax()

openpowerlifting_USAPL_lifters["Best Raw Bench to date"] =_
→openpowerlifting_USAPL_lifters[
    (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") | (
        openpowerlifting_USAPL_lifters["Equipment"] == "Wraps")].
→groupby("Name")["Best3BenchKg"].cummax()

openpowerlifting_USAPL_lifters["Best Raw Deadlift to date"] =_
→openpowerlifting_USAPL_lifters[
    (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") | (
        openpowerlifting_USAPL_lifters["Equipment"] == "Wraps")].
→groupby("Name")["Best3DeadliftKg"].cummax()

openpowerlifting_USAPL_lifters["Best Raw Total to date"] =_
→openpowerlifting_USAPL_lifters[
    openpowerlifting_USAPL_lifters["Equipment"] == "Raw"] .
→groupby("Name")["TotalKg"].cummax()
```

```

openpowerlifting_USAPL_lifters["Best Raw Wilks to date"] =_
    ↪openpowerlifting_USAPL_lifters[
        openpowerlifting_USAPL_lifters["Equipment"] == "Raw"] .
    ↪groupby("Name")["Wilks"].cummax()

openpowerlifting_USAPL_lifters["Best Wrapped Total to date"] =_
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"] ==
        "Wraps"].groupby("Name")["TotalKg"].cummax()

openpowerlifting_USAPL_lifters["Best Wrapped Wilks to date"] =_
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"] ==
        "Wraps"].groupby("Name")["Wilks"].cummax()

openpowerlifting_USAPL_lifters["Best S-Ply Squat to date"] =_
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"] ==
        "Single-ply"].groupby("Name")["Best3SquatKg"].cummax()

openpowerlifting_USAPL_lifters["Best S-Ply Bench to date"] =_
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"] ==
        "Single-ply"].groupby("Name")["Best3BenchKg"].cummax()

openpowerlifting_USAPL_lifters["Best S-Ply Deadlift to date"] =_
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"] ==
        "Single-ply"].groupby("Name")["Best3DeadliftKg"].cummax()

openpowerlifting_USAPL_lifters["Best S-Ply Total to date"] =_
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"] ==
        "Single-ply"].groupby("Name")["TotalKg"].cummax()

openpowerlifting_USAPL_lifters["Best S-Ply Wilks to date"] =_
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"] ==
        "Single-ply"].groupby("Name")["Wilks"].cummax()

openpowerlifting_USAPL_lifters["Best M-Ply Squat to date"] =_
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"] ==
        "Multi-ply"].groupby("Name")["Best3SquatKg"].cummax()

openpowerlifting_USAPL_lifters["Best M-Ply Bench to date"] =_
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"] ==
        "Multi-ply"].groupby("Name")["Best3BenchKg"].cummax()

openpowerlifting_USAPL_lifters["Best M-Ply Deadlift to date"] =_
    ↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"] ==
        "Multi-ply"].groupby("Name")["Best3DeadliftKg"].cummax()

```

```

openpowerlifting_USAPL_lifters["Best M-Ply Total to date"] =_
↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"]]=_
↪== "Multi-ply"].groupby("Name")["TotalKg"].cummax()

openpowerlifting_USAPL_lifters["Best M-Ply Wilks to date"] =_
↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"]]=_
↪== "Multi-ply"].groupby("Name")["Wilks"].cummax()

openpowerlifting_USAPL_lifters["Best Unlimited Squat to date"] =_
↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"]]=_
↪== "Unlimited"].groupby("Name")["Best3SquatKg"].cummax()

openpowerlifting_USAPL_lifters["Best Unlimited Bench to date"] =_
↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"]]=_
↪== "Unlimited"].groupby("Name")["Best3BenchKg"].cummax()

openpowerlifting_USAPL_lifters["Best Unlimited Deadlift to date"] =_
↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"]]=_
↪== "Unlimited"].groupby("Name")["Best3DeadliftKg"].cummax()

openpowerlifting_USAPL_lifters["Best Unlimited Total to date"] =_
↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"]]=_
↪== "Unlimited"].groupby("Name")["TotalKg"].cummax()

openpowerlifting_USAPL_lifters["Best Unlimited Wilks to date"] =_
↪openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Equipment"]]=_
↪== "Unlimited"].groupby("Name")["Wilks"].cummax()

for column in openpowerlifting_USAPL_lifters.columns[-1:-24:-1]:
    openpowerlifting_USAPL_lifters[column] = openpowerlifting_USAPL_lifters.
↪groupby("Name")[column].shift()
    openpowerlifting_USAPL_lifters[column] = openpowerlifting_USAPL_lifters.
↪groupby("Name")[column].ffill()
    # due to cummax function leaving forward values empty when selecting a_
↪subset of data

```

[14]: openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] ==_
↪"Austin Perkins"]

	Name	Sex	Event	Equipment	Age	AgeClass	BirthYearClass	\
14793	Austin Perkins	M	SBD	Single-ply	14.5	13-15		14-18
14794	Austin Perkins	M	SBD	Single-ply	14.5	13-15		14-18
14795	Austin Perkins	M	SBD	Single-ply	14.5	13-15		14-18
14796	Austin Perkins	M	SBD	Single-ply	14.5	13-15		14-18
14797	Austin Perkins	M	SBD	Single-ply	15.5	16-17		14-18
14798	Austin Perkins	M	SBD	Single-ply	15.5	16-17		14-18

14799	Austin Perkins	M	SBD	Single-ply	16.5	16-17	14-18
14800	Austin Perkins	M	SBD	Single-ply	16.5	16-17	14-18
14801	Austin Perkins	M	SBD	Single-ply	17.5	18-19	14-18
14802	Austin Perkins	M	SBD	Single-ply	17.5	18-19	14-18
14803	Austin Perkins	M	SBD	Single-ply	17.5	18-19	14-18
14804	Austin Perkins	M	SBD	Single-ply	18.5	18-19	19-23
14805	Austin Perkins	M	SBD	Raw	18.5	18-19	19-23
14806	Austin Perkins	M	SBD	Raw	19.5	20-23	19-23
14807	Austin Perkins	M	SBD	Single-ply	19.5	20-23	19-23
14808	Austin Perkins	M	SBD	Single-ply	19.5	20-23	19-23
14809	Austin Perkins	M	SBD	Raw	19.5	20-23	19-23
14810	Austin Perkins	M	SBD	Raw	19.5	20-23	19-23
14811	Austin Perkins	M	SBD	Raw	20.5	20-23	19-23
14812	Austin Perkins	M	SBD	Single-ply	21.5	20-23	19-23
14813	Austin Perkins	M	SBD	Single-ply	21.5	20-23	19-23
14814	Austin Perkins	M	SBD	Raw	21.5	20-23	19-23

	Division	BodyweightKg	WeightClassKg	Squat1Kg	Squat2Kg	Squat3Kg	\
14793	Boys	47.26	51.7	NaN	NaN	NaN	
14794	Boys	47.67	51.7	NaN	NaN	NaN	
14795	Boys	48.44	51.7	NaN	NaN	NaN	
14796	Boys	47.90	51.7	NaN	NaN	NaN	
14797	Boys	53.98	55.7	NaN	NaN	NaN	
14798	Boys	53.07	55.7	NaN	NaN	NaN	
14799	M-HS	65.80	66	215.0	227.5	-235.0	
14800	M-V	66.80	67.5	-227.5	230.0	-235.0	
14801	M-T2	70.50	74	227.5	232.5	245.0	
14802	M-TJ	69.95	75	232.5	250.0	-262.5	
14803	Sub-Juniors	70.32	74	240.0	252.5	257.5	
14804	M-C	71.50	74	272.5	290.0	-302.5	
14805	MR-O	71.50	74	230.0	242.5	247.5	
14806	MR-C	72.20	74	240.0	256.5	265.5	
14807	M-C	72.10	74	300.0	310.0	-317.5	
14808	M-Jr	72.55	74	300.0	315.0	320.0	
14809	Open	72.63	74	260.0	270.0	275.0	
14810	MR-O	73.11	74	260.0	275.0	285.0	
14811	MR-G	72.70	74	272.5	285.0	300.0	
14812	M-C	74.10	83	330.0	-350.0	-350.0	
14813	M-O	72.42	74	310.0	320.0	-328.0	
14814	MR-O	73.44	74	275.0	290.0	-295.0	

	Squat4Kg	Best3SquatKg	Bench1Kg	Bench2Kg	Bench3Kg	Bench4Kg	\
14793	NaN	70.31	NaN	NaN	NaN	NaN	
14794	NaN	90.72	NaN	NaN	NaN	NaN	
14795	NaN	97.52	NaN	NaN	NaN	NaN	
14796	NaN	104.33	NaN	NaN	NaN	NaN	
14797	NaN	117.93	NaN	NaN	NaN	NaN	

14798	NaN	111.13	NaN	NaN	NaN	NaN
14799	NaN	227.50	102.5	110.0	115.0	NaN
14800	NaN	230.00	110.0	120.0	132.5	NaN
14801	NaN	245.00	115.0	125.0	130.0	NaN
14802	NaN	250.00	120.0	-147.5	147.5	NaN
14803	NaN	257.50	145.0	155.0	165.0	NaN
14804	NaN	290.00	155.0	-162.5	162.5	NaN
14805	NaN	247.50	140.0	147.5	150.0	NaN
14806	NaN	265.50	140.0	145.0	-150.5	NaN
14807	NaN	310.00	175.0	182.5	187.5	NaN
14808	NaN	320.00	175.0	185.0	192.5	NaN
14809	NaN	275.00	145.0	155.0	-165.0	NaN
14810	NaN	285.00	152.5	162.5	167.5	NaN
14811	NaN	300.00	-162.5	170.0	180.0	NaN
14812	NaN	330.00	160.0	170.0	175.0	NaN
14813	NaN	320.00	-180.0	187.5	200.0	NaN
14814	NaN	290.00	165.0	172.5	180.0	NaN

	Best3BenchKg	Deadlift1Kg	Deadlift2Kg	Deadlift3Kg	Deadlift4Kg	\
14793	52.16	NaN	NaN	NaN	NaN	NaN
14794	NaN	NaN	NaN	NaN	NaN	NaN
14795	52.16	NaN	NaN	NaN	NaN	NaN
14796	52.16	NaN	NaN	NaN	NaN	NaN
14797	65.77	NaN	NaN	NaN	NaN	NaN
14798	70.31	NaN	NaN	NaN	NaN	NaN
14799	115.00	220.0	227.5	232.5	NaN	
14800	132.50	225.0	232.5	237.5	NaN	
14801	130.00	227.5	235.0	250.0	NaN	
14802	147.50	242.5	260.0	265.0	NaN	
14803	165.00	245.0	260.0	-277.5	NaN	
14804	162.50	255.0	265.0	-272.5	NaN	
14805	150.00	250.0	265.0	272.5	NaN	
14806	145.00	260.0	265.0	278.0	NaN	
14807	187.50	260.0	270.0	285.0	NaN	
14808	192.50	272.5	282.5	290.0	NaN	
14809	155.00	270.0	287.5	307.5	NaN	
14810	167.50	275.0	295.0	305.0	NaN	
14811	180.00	285.0	305.0	320.0	NaN	
14812	175.00	285.0	300.0	-310.5	NaN	
14813	200.00	275.0	-285.0	-285.0	NaN	
14814	180.00	285.0	295.0	300.0	NaN	

	Best3DeadliftKg	TotalKg	Place	Dots	Wilks	Glossbrenner	\
14793	90.72	213.19	2	224.53	232.226181	229.84	
14794	NaN	NaN	DQ	NaN	NaN	NaN	
14795	106.59	256.28	5	263.15	271.515646	268.43	
14796	106.59	263.08	4	273.24	282.241243	279.18	

14797	124.74	308.44	4	285.32	291.240547	286.18
14798	120.20	301.64	2	283.44	289.802613	285.06
14799	232.50	575.00	1	451.54	452.617372	439.88
14800	237.50	600.00	3	465.97	466.508564	453.05
14801	250.00	625.00	1	467.17	465.837276	451.31
14802	265.00	662.50	1	497.89	496.740898	481.41
14803	260.00	682.50	2	511.05	509.682051	493.84
14804	265.00	717.50	2	531.18	529.157288	512.34
14805	272.50	670.00	1	496.02	494.125969	478.42
14806	278.00	688.50	1	506.39	504.127956	487.90
14807	285.00	782.50	1	576.06	573.539596	555.11
14808	290.00	802.50	1	588.33	585.524932	566.56
14809	307.50	737.50	1	540.28	537.667309	520.23
14810	305.00	757.50	2	552.52	549.617761	531.64
14811	320.00	800.00	G	585.70	582.823679	563.90
14812	300.00	805.00	1	582.03	578.493472	559.25
14813	275.00	795.00	2	583.53	580.812735	562.04
14814	300.00	770.00	2	559.98	556.880504	538.56

	Goodlift	Tested	Country	State	Federation	ParentFederation	Date	\
14793	37.41	Yes	USA	NaN	THSPA		NaN	2014-01-18
14794	NaN	Yes	USA	NaN	THSPA		NaN	2014-01-28
14795	43.98	Yes	USA	NaN	THSPA		NaN	2014-02-13
14796	45.60	Yes	USA	NaN	THSPA		NaN	2014-03-08
14797	48.22	Yes	USA	NaN	THSPA		NaN	2015-01-17
14798	47.83	Yes	USA	NaN	THSPA		NaN	2015-02-06
14799	77.18	Yes	USA	MS	USAPL		IPF	2016-03-19
14800	79.68	Yes	USA	MS	USAPL		IPF	2016-04-01
14801	79.97	Yes	USA	MS	USAPL		IPF	2017-01-21
14802	85.22	Yes	USA	MS	USAPL		IPF	2017-03-29
14803	87.48	Yes	USA	NaN	IPF		IPF	2017-08-28
14804	90.95	Yes	USA	MS	USAPL		IPF	2018-04-19
14805	100.14	Yes	USA	MS	USAPL		IPF	2018-10-20
14806	102.38	Yes	USA	MS	USAPL		IPF	2019-02-02
14807	98.64	Yes	USA	MS	USAPL		IPF	2019-04-12
14808	100.74	Yes	USA	MS	USAPL		IPF	2019-05-10
14809	109.33	Yes	USA	NaN	IPF		IPF	2019-07-22
14810	111.91	Yes	USA	MS	USAPL		IPF	2019-10-16
14811	118.53	Yes	USA	MS	USAPL		IPF	2020-11-14
14812	99.67	Yes	USA	MS	USAPL		IPF	2021-04-08
14813	99.92	Yes	USA	MS	USAPL		IPF	2021-06-14
14814	113.49	Yes	USA	MS	USAPL		IPF	2021-06-14

	MeetCountry	MeetState		MeetTown	\
14793	USA	TX		Mann Middle School	
14794	USA	TX	Madison	Middle School Abilene	
14795	USA	TX		Sweetwater	

14796	USA	TX	Midland
14797	USA	TX	ActionZone Abilene
14798	USA	TX	Cooper
14799	USA	MS	Nan
14800	USA	FL	Orlando
14801	USA	MS	Nan
14802	USA	PA	Scranton
14803	USA	NaN	Orlando
14804	USA	TX	College Station
14805	USA	WI	Nan
14806	USA	NE	Nan
14807	USA	OH	Columbus
14808	USA	IL	Nan
14809	Estonia	NaN	Tartu
14810	USA	IL	Lombard
14811	USA	NE	Nan
14812	USA	LA	Baton Rouge
14813	USA	FL	Daytona Beach
14814	USA	FL	Daytona Beach

	MeetName	Repeated_meet	\
14793	Abilene High Invitational	False	
14794	Abilene Cooper PWL Meet	False	
14795	Mustang Power Classic	False	
14796	Boys Region 1 Division 1	False	
14797	Abilene High Invitational	False	
14798	Abilene Cooper Powerlifting Meet	False	
14799	North MS Championships	False	
14800	High School Nationals	False	
14801	North Mississippi Championships	False	
14802	High School Raw & Equipped Nationals	False	
14803	World Sub-Juniors & Juniors Powerlifting Champ...	False	
14804	Collegiate Nationals	False	
14805	Milwaukee Collegiate Cup	False	
14806	Nebraska Collegiate State Championships	False	
14807	Collegiate Nationals	False	
14808	Open Nationals	False	
14809	University Powerlifting Cup	False	
14810	Raw Nationals	False	
14811	Warrior Collegiate Open	False	
14812	Collegiate and Junior Nationals	False	
14813	Equipped Nationals	False	
14814	Raw Nationals	False	

	Two_meets_on_one_day	DateOfBirthInterval	Time Since Last Meet	\
14793	False	[1999-01-01, 1999-12-31]	NaT	
14794	False	[1999-01-01, 1999-12-31]	10 days	

14795	False	[1999-01-01, 1999-12-31]	16 days
14796	False	[1999-01-01, 1999-12-31]	23 days
14797	False	[1999-01-01, 1999-12-31]	315 days
14798	False	[1999-01-01, 1999-12-31]	20 days
14799	False	[1999-01-01, 1999-12-31]	407 days
14800	False	[1999-01-01, 1999-12-31]	13 days
14801	False	[1999-01-01, 1999-12-31]	295 days
14802	False	[1999-01-01, 1999-12-31]	67 days
14803	False	[1999-01-01, 1999-12-31]	152 days
14804	False	[1999-01-01, 1999-12-31]	234 days
14805	False	[1999-01-01, 1999-12-31]	184 days
14806	False	[1999-01-01, 1999-12-31]	105 days
14807	False	[1999-01-01, 1999-12-31]	69 days
14808	False	[1999-01-01, 1999-12-31]	28 days
14809	False	[1999-01-01, 1999-12-31]	73 days
14810	False	[1999-01-01, 1999-12-31]	86 days
14811	False	[1999-01-01, 1999-12-31]	395 days
14812	False	[1999-01-01, 1999-12-31]	145 days
14813	True	[1999-01-01, 1999-12-31]	67 days
14814	True	[1999-01-01, 1999-12-31]	0 days

	Number of Past Meets	Number of Past Meets Same Day Exclusive	\
14793	0	0	0
14794	1	1	1
14795	2	2	2
14796	3	3	3
14797	4	4	4
14798	5	5	5
14799	6	6	6
14800	7	7	7
14801	8	8	8
14802	9	9	9
14803	10	10	10
14804	11	11	11
14805	12	12	12
14806	13	13	13
14807	14	14	14
14808	15	15	15
14809	16	16	16
14810	17	17	17
14811	18	18	18
14812	19	19	19
14813	20	20	20
14814	21	20	20

	Best Raw Squat to date	Best Wrapped Squat to date	\
14793	NaN	NaN	

14794	NaN	NaN
14795	NaN	NaN
14796	NaN	NaN
14797	NaN	NaN
14798	NaN	NaN
14799	NaN	NaN
14800	NaN	NaN
14801	NaN	NaN
14802	NaN	NaN
14803	NaN	NaN
14804	NaN	NaN
14805	NaN	NaN
14806	247.5	NaN
14807	265.5	NaN
14808	265.5	NaN
14809	265.5	NaN
14810	275.0	NaN
14811	285.0	NaN
14812	300.0	NaN
14813	300.0	NaN
14814	300.0	NaN

	Best Raw Bench to date	Best Raw Deadlift to date	\
14793	NaN	NaN	
14794	NaN	NaN	
14795	NaN	NaN	
14796	NaN	NaN	
14797	NaN	NaN	
14798	NaN	NaN	
14799	NaN	NaN	
14800	NaN	NaN	
14801	NaN	NaN	
14802	NaN	NaN	
14803	NaN	NaN	
14804	NaN	NaN	
14805	NaN	NaN	
14806	150.0	272.5	
14807	150.0	278.0	
14808	150.0	278.0	
14809	150.0	278.0	
14810	155.0	307.5	
14811	167.5	307.5	
14812	180.0	320.0	
14813	180.0	320.0	
14814	180.0	320.0	

Best Raw Total to date Best Raw Wilks to date \

14793	NaN	NaN
14794	NaN	NaN
14795	NaN	NaN
14796	NaN	NaN
14797	NaN	NaN
14798	NaN	NaN
14799	NaN	NaN
14800	NaN	NaN
14801	NaN	NaN
14802	NaN	NaN
14803	NaN	NaN
14804	NaN	NaN
14805	NaN	NaN
14806	670.0	494.125969
14807	688.5	504.127956
14808	688.5	504.127956
14809	688.5	504.127956
14810	737.5	537.667309
14811	757.5	549.617761
14812	800.0	582.823679
14813	800.0	582.823679
14814	800.0	582.823679

	Best Wrapped Total to date	Best Wrapped Wilks to date \
14793	NaN	NaN
14794	NaN	NaN
14795	NaN	NaN
14796	NaN	NaN
14797	NaN	NaN
14798	NaN	NaN
14799	NaN	NaN
14800	NaN	NaN
14801	NaN	NaN
14802	NaN	NaN
14803	NaN	NaN
14804	NaN	NaN
14805	NaN	NaN
14806	NaN	NaN
14807	NaN	NaN
14808	NaN	NaN
14809	NaN	NaN
14810	NaN	NaN
14811	NaN	NaN
14812	NaN	NaN
14813	NaN	NaN
14814	NaN	NaN

	Best S-Ply Squat to date	Best S-Ply Bench to date	\
14793	NaN	NaN	
14794	70.31	52.16	
14795	90.72	52.16	
14796	97.52	52.16	
14797	104.33	52.16	
14798	117.93	65.77	
14799	117.93	70.31	
14800	227.50	115.00	
14801	230.00	132.50	
14802	245.00	132.50	
14803	250.00	147.50	
14804	257.50	165.00	
14805	290.00	165.00	
14806	290.00	165.00	
14807	290.00	165.00	
14808	310.00	187.50	
14809	320.00	192.50	
14810	320.00	192.50	
14811	320.00	192.50	
14812	320.00	192.50	
14813	330.00	192.50	
14814	330.00	200.00	

	Best S-Ply Deadlift to date	Best S-Ply Total to date	\
14793	NaN	NaN	
14794	90.72	213.19	
14795	90.72	213.19	
14796	106.59	256.28	
14797	106.59	263.08	
14798	124.74	308.44	
14799	124.74	308.44	
14800	232.50	575.00	
14801	237.50	600.00	
14802	250.00	625.00	
14803	265.00	662.50	
14804	265.00	682.50	
14805	265.00	717.50	
14806	265.00	717.50	
14807	265.00	717.50	
14808	285.00	782.50	
14809	290.00	802.50	
14810	290.00	802.50	
14811	290.00	802.50	
14812	290.00	802.50	
14813	300.00	805.00	
14814	300.00	805.00	

	Best S-Ply Wilks to date	Best M-Ply Squat to date	\
14793	NaN	NaN	
14794	232.226181	NaN	
14795	232.226181	NaN	
14796	271.515646	NaN	
14797	282.241243	NaN	
14798	291.240547	NaN	
14799	291.240547	NaN	
14800	452.617372	NaN	
14801	466.508564	NaN	
14802	466.508564	NaN	
14803	496.740898	NaN	
14804	509.682051	NaN	
14805	529.157288	NaN	
14806	529.157288	NaN	
14807	529.157288	NaN	
14808	573.539596	NaN	
14809	585.524932	NaN	
14810	585.524932	NaN	
14811	585.524932	NaN	
14812	585.524932	NaN	
14813	585.524932	NaN	
14814	585.524932	NaN	

	Best M-Ply Bench to date	Best M-Ply Deadlift to date	\
14793	NaN	NaN	
14794	NaN	NaN	
14795	NaN	NaN	
14796	NaN	NaN	
14797	NaN	NaN	
14798	NaN	NaN	
14799	NaN	NaN	
14800	NaN	NaN	
14801	NaN	NaN	
14802	NaN	NaN	
14803	NaN	NaN	
14804	NaN	NaN	
14805	NaN	NaN	
14806	NaN	NaN	
14807	NaN	NaN	
14808	NaN	NaN	
14809	NaN	NaN	
14810	NaN	NaN	
14811	NaN	NaN	
14812	NaN	NaN	
14813	NaN	NaN	

14814	NaN	NaN
-------	-----	-----

	Best M-Ply Total to date	Best M-Ply Wilks to date	\
14793	NaN	NaN	
14794	NaN	NaN	
14795	NaN	NaN	
14796	NaN	NaN	
14797	NaN	NaN	
14798	NaN	NaN	
14799	NaN	NaN	
14800	NaN	NaN	
14801	NaN	NaN	
14802	NaN	NaN	
14803	NaN	NaN	
14804	NaN	NaN	
14805	NaN	NaN	
14806	NaN	NaN	
14807	NaN	NaN	
14808	NaN	NaN	
14809	NaN	NaN	
14810	NaN	NaN	
14811	NaN	NaN	
14812	NaN	NaN	
14813	NaN	NaN	
14814	NaN	NaN	

	Best Unlimited Squat to date	Best Unlimited Bench to date	\
14793	NaN	NaN	
14794	NaN	NaN	
14795	NaN	NaN	
14796	NaN	NaN	
14797	NaN	NaN	
14798	NaN	NaN	
14799	NaN	NaN	
14800	NaN	NaN	
14801	NaN	NaN	
14802	NaN	NaN	
14803	NaN	NaN	
14804	NaN	NaN	
14805	NaN	NaN	
14806	NaN	NaN	
14807	NaN	NaN	
14808	NaN	NaN	
14809	NaN	NaN	
14810	NaN	NaN	
14811	NaN	NaN	
14812	NaN	NaN	

14813	NaN	NaN
14814	NaN	NaN

	Best Unlimited Deadlift to date	Best Unlimited Total to date \
14793	NaN	NaN
14794	NaN	NaN
14795	NaN	NaN
14796	NaN	NaN
14797	NaN	NaN
14798	NaN	NaN
14799	NaN	NaN
14800	NaN	NaN
14801	NaN	NaN
14802	NaN	NaN
14803	NaN	NaN
14804	NaN	NaN
14805	NaN	NaN
14806	NaN	NaN
14807	NaN	NaN
14808	NaN	NaN
14809	NaN	NaN
14810	NaN	NaN
14811	NaN	NaN
14812	NaN	NaN
14813	NaN	NaN
14814	NaN	NaN

Best Unlimited Wilks to date

14793	NaN
14794	NaN
14795	NaN
14796	NaN
14797	NaN
14798	NaN
14799	NaN
14800	NaN
14801	NaN
14802	NaN
14803	NaN
14804	NaN
14805	NaN
14806	NaN
14807	NaN
14808	NaN
14809	NaN
14810	NaN
14811	NaN

```
14812          NaN  
14813          NaN  
14814          NaN
```

```
[15]: openpowerlifting_USAPL_lifters.to_csv("C:/Users/Montel/Google Drive/Data\u2192Science MSc/Msc-Project---Powerlifting/Datasets/\u2192open_powerlifting_USAPL_Raw_active_lifters_disambiguated_ready_for_analysis.\u2192csv", index = False)
```

```
[9]:
```

```
[9]: Index(['Best Unlimited Wilks to date', 'Best Unlimited Total to date',  
          'Best Unlimited Deadlift to date', 'Best Unlimited Bench to date',  
          'Best Unlimited Squat to date', 'Best M-Ply Wilks to date',  
          'Best M-Ply Total to date', 'Best M-Ply Deadlift to date',  
          'Best M-Ply Bench to date', 'Best M-Ply Squat to date',  
          'Best S-Ply Wilks to date', 'Best S-Ply Total to date',  
          'Best S-Ply Deadlift to date', 'Best S-Ply Bench to date',  
          'Best S-Ply Squat to date', 'Best Wrapped Wilks to date',  
          'Best Wrapped Total to date', 'Best Raw Wilks to date',  
          'Best Raw Total to date', 'Best Raw Deadlift to date',  
          'Best Raw Bench to date', 'Best Wrapped Squat to date',  
          'Best Raw Squat to date'],  
         dtype='object')
```

```
[ ]:
```

- d. Squat Pass or fail:
 - i. Missing values imputed

Appendix_F_Pass or Fail, filling na, absolute columns and gender variable

September 10, 2021

```
[1]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json
import copy
import random
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', 500)

openpowerlifting_USAPL_lifters = pd.read_csv("C:/Users/Montel/Google Drive/Data_"
→Science MSc/Msc-Project---Powerlifting/Datasets/
→open_powerlifting_USAPL_Raw_active_lifters_disambiguated_ready_for_analysis.
→csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
→to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
openpowerlifting_USAPL_lifters["Time Since Last Meet"] = pd.
→to_timedelta(openpowerlifting_USAPL_lifters["Time Since Last Meet"])

#reconverting date of birth interval from list to interval
for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open"|
→and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        list_date = openpowerlifting_USAPL_lifters.at[i,|
→"DateOfBirthInterval"][1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd.
→Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed =
→"both")
```

Calculating the number of meets in the past year

```
[2]: def number_of_meets_past_year(x):
    number = []
    for i in x.index:
        number.append(x[(x["Date"] < x.at[i, "Date"]) & (x["Date"] > x.
→at[i, "Date"] - relativedelta(years = 1))].shape[0])
```

```

    return pd.Series(x[(x["Date"] < x.at[i, "Date"]) & (x["Date"] > x.
→at[i, "Date"] - relativedelta(years = 1))].shape[0] for i in x.index)

openpowerlifting_USAPL_lifters["Number of meets in past 12 months"] =_
→openpowerlifting_USAPL_lifters.groupby(
    "Name")[[ "Date", "Name"]].apply(
        lambda x: pd.DataFrame((
            x[(x["Date"] < x.at[i, "Date"])\ 
            & (x["Date"] > x.at[i, "Date"] - relativedelta(years = 1))].shape[0]_
→for i in x.index), x.index))

```

If a lifter failed all their attempts, the “Best3X’Kg” columns will be filled with a negative value representing the lifter’s highest failed attempt. This will be converted to zero to avoid anomalies when inserted into the model. Additionally, the “Best lifts to date” have this problem, therefore backfilling will occur for these too.

[3]:

```

openpowerlifting_USAPL_lifters["Best3SquatKg"] = openpowerlifting_USAPL_lifters[
    "Best3SquatKg"].where(openpowerlifting_USAPL_lifters["Best3SquatKg"] > 0, 0)
openpowerlifting_USAPL_lifters["Best3BenchKg"] = openpowerlifting_USAPL_lifters[[
    "Best3BenchKg"].where(openpowerlifting_USAPL_lifters["Best3BenchKg"] > 0, 0)
openpowerlifting_USAPL_lifters["Best3DeadliftKg"] =_
→openpowerlifting_USAPL_lifters[
    "Best3DeadliftKg"].where(openpowerlifting_USAPL_lifters["Best3DeadliftKg"]_
→> 0, 0)

for column in openpowerlifting_USAPL_lifters.columns[-2:-25:-1]:
    openpowerlifting_USAPL_lifters[column] =_
→openpowerlifting_USAPL_lifters[column].
    where(openpowerlifting_USAPL_lifters[column] > 0, 0)

```

[4]:

```

openpowerlifting_USAPL_lifters.insert(11, "Squat1 Pass",_
→openpowerlifting_USAPL_lifters["Squat1Kg"] > 0)
openpowerlifting_USAPL_lifters["Squat1 Pass"] =_
→openpowerlifting_USAPL_lifters["Squat1 Pass"].astype(int)

openpowerlifting_USAPL_lifters.insert(13, "Squat2 Pass",_
→openpowerlifting_USAPL_lifters["Squat2Kg"] > 0)
openpowerlifting_USAPL_lifters["Squat2 Pass"] =_
→openpowerlifting_USAPL_lifters["Squat2 Pass"].astype(int)

openpowerlifting_USAPL_lifters.insert(15, "Squat3 Pass",_
→openpowerlifting_USAPL_lifters["Squat3Kg"] > 0)
openpowerlifting_USAPL_lifters["Squat3 Pass"] =_
→openpowerlifting_USAPL_lifters["Squat3 Pass"].astype(int)

```

```
openpowerlifting_USAPL_lifters.insert(19, "Bench1 Pass",  
    ↳ openpowerlifting_USAPL_lifters["Bench1Kg"] > 0)  
openpowerlifting_USAPL_lifters["Bench1 Pass"] =  
    ↳ openpowerlifting_USAPL_lifters["Bench1 Pass"].astype(int)
```

```
openpowerlifting_USAPL_lifters.insert(21, "Bench2 Pass",  
    ↳ openpowerlifting_USAPL_lifters["Bench2Kg"] > 0)  
openpowerlifting_USAPL_lifters["Bench2 Pass"] =  
    ↳ openpowerlifting_USAPL_lifters["Bench2 Pass"].astype(int)
```

```
openpowerlifting_USAPL_lifters.insert(23, "Bench3 Pass",  
    ↳ openpowerlifting_USAPL_lifters["Bench3Kg"] > 0)  
openpowerlifting_USAPL_lifters["Bench3 Pass"] =  
    ↳ openpowerlifting_USAPL_lifters["Bench3 Pass"].astype(int)
```

```
openpowerlifting_USAPL_lifters.insert(27, "Deadlift1 Pass",  
    ↳ openpowerlifting_USAPL_lifters["Deadlift1Kg"] > 0)  
openpowerlifting_USAPL_lifters["Deadlift1 Pass"] =  
    ↳ openpowerlifting_USAPL_lifters["Deadlift1 Pass"].astype(int)
```

```
openpowerlifting_USAPL_lifters.insert(29, "Deadlift2 Pass",  
    ↳ openpowerlifting_USAPL_lifters["Deadlift2Kg"] > 0)  
openpowerlifting_USAPL_lifters["Deadlift2 Pass"] =  
    ↳ openpowerlifting_USAPL_lifters["Deadlift2 Pass"].astype(int)
```

```
openpowerlifting_USAPL_lifters.insert(31, "Deadlift3 Pass",  
    ↳ openpowerlifting_USAPL_lifters["Deadlift3Kg"] > 0)  
openpowerlifting_USAPL_lifters["Deadlift3 Pass"] =  
    ↳ openpowerlifting_USAPL_lifters["Deadlift3 Pass"].astype(int)
```

Transforming nan values in the time since last meet column. Nan values are first meets, so we will insert the time between birth and the first meet. This time is large, so should be anomalous compared to the other times

```
[5]: for i in openpowerlifting_USAPL_lifters.index:  
    if type(openpowerlifting_USAPL_lifters.at[i, "Time Since Last Meet"]) == pd.  
        ↳ _libs.tslibs.nattype.NaTType:  
            try:  
                openpowerlifting_USAPL_lifters.at[  
                    i, "Time Since Last Meet"] =  
                    ↳ openpowerlifting_USAPL_lifters["Date"][i] \  
                        - openpowerlifting_USAPL_lifters["DateOfBirthInterval"][i].left  
            except:  
                for j in openpowerlifting_USAPL_lifters[
```

```

        openpowerlifting_USAPL_lifters["Name"] == u
→openpowerlifting_USAPL_lifters.at[i, "Name"].index:
    if (type(openpowerlifting_USAPL_lifters.
→at[j, "DateOfBirthInterval"])
        == pd._libs.interval.Interval) and u
→(type(openpowerlifting_USAPL_lifters.at[j, "Age"]) == np.float64):
    openpowerlifting_USAPL_lifters.at[i, "Time Since Last Meet"]\
        = openpowerlifting_USAPL_lifters["Date"][j] - u
→openpowerlifting_USAPL_lifters["DateOfBirthInterval"][j].left
    break
    continue

openpowerlifting_USAPL_lifters["Time Since Last Meet"] = u
→openpowerlifting_USAPL_lifters["Time Since Last Meet"].dt.days

```

[7]:

```

sex_dummies = pd.get_dummies(openpowerlifting_USAPL_lifters, columns = ["Sex"]).
→filter(["Sex_M", "Sex_F"], axis=1)
openpowerlifting_USAPL_lifters.insert(1, "Male", np.nan)
openpowerlifting_USAPL_lifters.insert(2, "Female", np.nan)
openpowerlifting_USAPL_lifters.drop(columns = "Sex", inplace = True)
openpowerlifting_USAPL_lifters["Male"] = sex_dummies["Sex_M"]
openpowerlifting_USAPL_lifters["Female"] = sex_dummies["Sex_F"]

```

creating % of squat/deadlift/bench attempts made in the past 3 meets

[8]:

```

meets_where_attempts_recorded = pd.
→Series([False]*openpowerlifting_USAPL_lifters.shape[0])
### below for loop are indicies of lifters where all attempts are recorded
for name in ["Squat1", "Squat2", "Squat3",
    "Bench1", "Bench2", "Bench3",
    "Deadlift1", "Deadlift2", "Deadlift3"]:
    meets_where_attempts_recorded = meets_where_attempts_recorded | pd.
→notna(openpowerlifting_USAPL_lifters[f"{name}Kg"])

for name in ["Squat1", "Squat2", "Squat3",
    "Bench1", "Bench2", "Bench3",
    "Deadlift1", "Deadlift2", "Deadlift3"]:
    openpowerlifting_USAPL_lifters[f"{name} Past 3 meets success rate"] = np.nan
    openpowerlifting_USAPL_lifters.loc[
        (openpowerlifting_USAPL_lifters["Event"] == u
→"SBD") \
            & (openpowerlifting_USAPL_lifters["Equipment"] u
→== "Raw") \
            & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \
            & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \

```

```

        & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \ 
            & (meets_where_attempts_recorded)
                , [f"{name} Past 3 meets success rate"]]\ \
= openpowerlifting_USAPL_lifters.
→loc[(openpowerlifting_USAPL_lifters["Event"] == "SBD")\ 
            & (openpowerlifting_USAPL_lifters["Equipment"]\ 
→== "Raw")\ 
            & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \ 
            & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \ 
            & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \ 
            & (meets_where_attempts_recorded), :].groupby(
    "Name") [f"{name} Pass"].rolling(3, min_periods = 1).mean().
→reset_index(0)[f"{name} Pass"]

    openpowerlifting_USAPL_lifters.loc[:, f"{name} Past 3 meets success rate"] = \
→openpowerlifting_USAPL_lifters.groupby(
    "Name") [f"{name} Past 3 meets success rate"].shift()
    openpowerlifting_USAPL_lifters.loc[:, f"{name} Past 3 meets success rate"] = \
→openpowerlifting_USAPL_lifters.groupby(
    "Name") [f"{name} Past 3 meets success rate"].ffill()

```

calculating overall attempt success rate and impute missing values

```
[9]: from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

imputer = IterativeImputer(random_state = 0, max_iter = 100, n_nearest_features\ 
→= 6)
imputer.fit(openpowerlifting_USAPL_lifters.loc[
            (openpowerlifting_USAPL_lifters["Event"] == "SBD")\ 
            & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw")\ 
            & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \ 
            & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \ 
            & (openpowerlifting_USAPL_lifters["Date"] < pd.
→to_datetime("07/01/2019")) \ 
            & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \ 
            & (meets_where_attempts_recorded)
```

```

        ,["Male", "Female", "Age", "BodyweightKg", \u2193
    "Number of meets in past 12 months", "Number of Past Meets Same Day\u2193
    Exclusive"] + list(map(lambda x: x + " Past 3 meets success rate", \u2193
    ["Squat1", "Squat2", "Squat3",
     "Bench1", "Bench2", "Bench3",
     "Deadlift1", "Deadlift2", "Deadlift3"])))]

openpowerlifting_USAPL_lifters.loc[
    (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
    & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
    & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
    & (openpowerlifting_USAPL_lifters["Date"] > pd.
    \u2193to_datetime("01/01/2014")) \
    & (pd.
    \u2193notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
    & (meets_where_attempts_recorded)
        ,["Male", "Female", "Age", "BodyweightKg", \u2193
    "Number of meets in past 12 months", "Number of Past Meets Same Day\u2193
    Exclusive"] + list(map(lambda x: x + " Past 3 meets success rate", \u2193
    ["Squat1", "Squat2", "Squat3",
     "Bench1", "Bench2", "Bench3",
     "Deadlift1", "Deadlift2", "Deadlift3"]))] =\

imputer.transform(openpowerlifting_USAPL_lifters.loc[
    (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
    & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
    & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
    & (openpowerlifting_USAPL_lifters["Date"] > pd.
    \u2193to_datetime("01/01/2014")) \
    & (pd.
    \u2193notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
    & (meets_where_attempts_recorded)
        ,["Male", "Female", "Age", "BodyweightKg", \u2193
    "Number of meets in past 12 months", "Number of Past Meets Same Day\u2193
    Exclusive"] + list(map(lambda x: x + " Past 3 meets success rate", \u2193
    ["Squat1", "Squat2", "Squat3",
     "Bench1", "Bench2", "Bench3",
     "Deadlift1", "Deadlift2", "Deadlift3"])))]

openpowerlifting_USAPL_lifters.loc[
    (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
    & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
    & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
    & (openpowerlifting_USAPL_lifters["Date"] > pd.
    \u2193to_datetime("01/01/2014")) \
    & (pd.
    \u2193notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \

```

```

        & (meets_where_attempts_recorded)
            ,["Male", "Female", "Age", "BodyweightKg", ↴
↳"Number of meets in past 12 months", "Number of Past Meets Same Day" ↴
↳Exclusive"] + list(map(lambda x: x + " Past 3 meets success rate", ↴
↳["Squat1", "Squat2", "Squat3",
    "Bench1", "Bench2", "Bench3",
    "Deadlift1", "Deadlift2", "Deadlift3"]))]

openpowerlifting_USAPL_lifters["Overall Attempt Success Rate over past three" ↴
↳meets"] = 0

for name in ["Squat1", "Squat2", "Squat3",
    "Bench1", "Bench2", "Bench3",
    "Deadlift1", "Deadlift2", "Deadlift3"]:
    openpowerlifting_USAPL_lifters["Overall Attempt Success Rate over past" ↴
↳three meets"] += openpowerlifting_USAPL_lifters[f"{name} Past 3 meets" ↴
↳success rate"]

openpowerlifting_USAPL_lifters["Overall Attempt Success Rate over past three" ↴
↳meets"] /= 9

```

C:\Users\Montel\anaconda3\lib\site-packages\sklearn\impute_iterative.py:685:
 ConvergenceWarning: [IterativeImputer] Early stopping criterion not reached.
 warnings.warn("[IterativeImputer] Early stopping criterion not"

[13]: openpowerlifting_USAPL_lifters.loc[:, "Squat1Kg": "Squat4Kg"] =
 ↳openpowerlifting_USAPL_lifters.loc[:, "Squat1Kg": "Squat4Kg"].abs()
 openpowerlifting_USAPL_lifters.loc[:, "Bench1Kg": "Bench4Kg"] =
 ↳openpowerlifting_USAPL_lifters.loc[:, "Bench1Kg": "Bench4Kg"].abs()
 openpowerlifting_USAPL_lifters.loc[:, "Deadlift1Kg": "Deadlift4Kg"] =
 ↳openpowerlifting_USAPL_lifters.loc[:, "Deadlift1Kg": "Deadlift4Kg"].abs()
 openpowerlifting_USAPL_lifters.loc[:, "Best Raw Squat to date": "Best Unlimited" ↴
 ↳Wilks to date"] = openpowerlifting_USAPL_lifters.loc[:, "Best Raw Squat to" ↴
 ↳date": "Best Unlimited Wilks to date"].fillna(0)

[52]: openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] ==
 ↳"USAPL") \\\n & (openpowerlifting_USAPL_lifters["Event"] ==
 ↳"SBD") \\\n & (openpowerlifting_USAPL_lifters["Equipment"] ↴
 ↳== "Raw") \\\n & (pd.
 ↳notna(openpowerlifting_USAPL_lifters["Age"])) \\\n & (openpowerlifting_USAPL_lifters["Date"] > pd.
 ↳to_datetime("01/01/2014")) \\\n & (pd.
 ↳notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \\\n

```

    & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat1Kg"])) \ 
        ].drop(columns = ["Event", "Equipment", "Glossbrenner", "Goodlift", "ParentFederation", "DateOfBirthInterval"]).
→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Squat1Kg.csv", index=False)

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \
    & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
    & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
    & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \
        & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \
            & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
                & (pd.
→notna(openpowerlifting_USAPL_lifters["Bench1Kg"])) \
                    & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat3Kg"])) \
                        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat2Kg"])) \
                            & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat1Kg"])) \
                                ].drop(columns = ["Event", "Equipment", "Glossbrenner", "Goodlift", "ParentFederation", "DateOfBirthInterval"]).
→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Bench1Kg.csv", index=False)

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \

```

```

        & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \\
        & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \\
        & (pd.
        ~notna(openpowerlifting_USAPL_lifters["Age"])) \\
        & (openpowerlifting_USAPL_lifters["Date"] > pd.
        ~to_datetime("01/01/2014")) \\
        & (pd.
        ~notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \\
        & (pd.
        ~notna(openpowerlifting_USAPL_lifters["Deadlift1Kg"])) \\
        & (pd.
        ~notna(openpowerlifting_USAPL_lifters["Bench3Kg"])) \\
        & (pd.
        ~notna(openpowerlifting_USAPL_lifters["Bench2Kg"])) \\
        & (pd.
        ~notna(openpowerlifting_USAPL_lifters["Bench1Kg"])) \\
        & (pd.
        ~notna(openpowerlifting_USAPL_lifters["Squat3Kg"])) \\
        & (pd.
        ~notna(openpowerlifting_USAPL_lifters["Squat2Kg"])) \\
        & (pd.
        ~notna(openpowerlifting_USAPL_lifters["Squat1Kg"])) \\
        ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Glossbrenner", "Goodlift", "Tested", "Country", "State", "Federation", "ParentFederation", "MeetName", "Repeated_meet", "Two_meets_on_one_day", "DateOfBirthInterval"]).
        ~to_csv("C:/Users/Montel/Google Drive/Data Science MSc/Msc-Project---Powerlifting/Datasets/attempt subsets/Deadlift1Kg.csv", index=False)
openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \\
        & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \\
        & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \\
        & (pd.
        ~notna(openpowerlifting_USAPL_lifters["Age"])) \\
        & (openpowerlifting_USAPL_lifters["Date"] > pd.
        ~to_datetime("01/01/2014"))

```

```

        & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat2Kg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat1Kg"])) \ 
            ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Tested", "Country", "State", "Federation", "MeetName", "Repeated_meet", "Two_meets_on_one_day", "Glossbrenner", "Goodlift", "ParentFederation", "DateOfBirthInterval"]).
→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/Msc-Project---Powerlifting/Datasets/attempt subsets/Squat2Kg.csv", index=False)
openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \
        & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
        & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \
        & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Bench2Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Bench1Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat3Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat2Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat1Kg"])) \
            ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Tested", "Country", "State", "Federation", "MeetName", "Repeated_meet", "Two_meets_on_one_day",

```

```

        "DateOfBirthInterval"]).

→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Bench2Kg.csv", index=_
→False)

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] ==_
→"USAPL") \
        & (openpowerlifting_USAPL_lifters["Event"] ==_
→"SBD") \
        & (openpowerlifting_USAPL_lifters["Equipment"] ==_
→"Raw") \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \
        & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Deadlift2Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Deadlift1Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Bench3Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Bench2Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Bench1Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat3Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat2Kg"])) \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat1Kg"])) \
        ].drop(columns = ["Event", "Equipment", _
→"AgeClass", "BirthYearClass", "Division",
                           "Glossbrenner", "Goodlift", _
→"Tested", "Country", "State", "Federation",
                           "ParentFederation", _
→"MeetName", "Repeated_meet", "Two_meets_on_one_day",
                           "DateOfBirthInterval"]).

→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Deadlift2Kg.csv", index=_
→False)

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] ==_
→"USAPL") \

```

```

        & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \\
        & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \\
        & (pd.
        →notna(openpowerlifting_USAPL_lifters["Age"])) \\
        & (openpowerlifting_USAPL_lifters["Date"] > pd.
        →to_datetime("01/01/2014")) \\
        & (pd.
        →notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \\
        & (pd.
        →notna(openpowerlifting_USAPL_lifters["Squat3Kg"])) \\
        & (pd.
        →notna(openpowerlifting_USAPL_lifters["Squat2Kg"])) \\
        & (pd.
        →notna(openpowerlifting_USAPL_lifters["Squat1Kg"])) \\
        ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Glossbrenner", "Goodlift", "Tested", "Country", "State", "Federation", "ParentFederation", "MeetName", "Repeated_meet", "Two_meets_on_one_day", "DateOfBirthInterval"]).
        →to_csv("C:/Users/Montel/Google Drive/Data Science MSc/Msc-Project---Powerlifting/Datasets/attempt subsets/Squat3Kg.csv", index=False)
openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \\
        & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \\
        & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \\
        & (pd.
        →notna(openpowerlifting_USAPL_lifters["Age"])) \\
        & (openpowerlifting_USAPL_lifters["Date"] > pd.
        →to_datetime("01/01/2014")) \\
        & (pd.
        →notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \\
        & (pd.
        →notna(openpowerlifting_USAPL_lifters["Bench3Kg"])) \\
        & (pd.
        →notna(openpowerlifting_USAPL_lifters["Bench2Kg"])) \\
        & (pd.
        →notna(openpowerlifting_USAPL_lifters["Bench1Kg"]))

```

```

        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat3Kg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat2Kg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat1Kg"])) \ 
            ].drop(columns = ["Event", "Equipment", 
→"AgeClass", "BirthYearClass", "Division", 
                "Glossbrenner", "Goodlift", 
→"Tested", "Country", "State", "Federation", 
                    "ParentFederation", 
→"MeetName", "Repeated_meet", "Two_meets_on_one_day", 
                        "DateOfBirthInterval"]).
→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Bench3Kg.csv", index=False)
openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == 
→"USAPL")\ 
        & (openpowerlifting_USAPL_lifters["Event"] == 
→"SBD")\ 
        & (openpowerlifting_USAPL_lifters["Equipment"] == 
→"Raw")\ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \ 
        & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Deadlift3Kg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Deadlift2Kg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Deadlift1Kg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Bench3Kg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Bench2Kg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Bench1Kg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat3Kg"])) \ 
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat2Kg"])) \

```

```

    & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat1Kg"])) \ 
        ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Glossbrenner", "Goodlift", "Tested", "Country", "State", "Federation", "ParentFederation", "MeetName", "Repeated_meet", "Two_meets_on_one_day", "DateOfBirthInterval"]).
→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Deadlift3Kg.csv", index=False)

```

[14]: openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \
 & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
 & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
 & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \
 & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \
 & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
 & (meets_where_attempts_recorded)
].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Glossbrenner", "Goodlift", "Tested", "Country", "State", "Federation", "ParentFederation", "MeetName", "Repeated_meet", "Two_meets_on_one_day", "DateOfBirthInterval"]).
→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Full Set.csv", index=False)

ii. Missing values not imputed

Appendix_F_Pass or Fail, filling na, absolute columns and gender variable - flow type - (na not filled for hist grad boost)

September 10, 2021

```
[1]: import pandas as pd
import numpy as np
import datetime
from dateutil.relativedelta import *
import json
import copy
import random
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', 500)

openpowerlifting_USAPL_lifters = pd.read_csv("C:/Users/Montel/Google Drive/Data_"
→Science MSc/Msc-Project---Powerlifting/Datasets/
→open_powerlifting_USAPL_Raw_active_lifters_disambiguated_ready_for_analysis.
→csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
→to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
openpowerlifting_USAPL_lifters["Time Since Last Meet"] = pd.
→to_timedelta(openpowerlifting_USAPL_lifters["Time Since Last Meet"])

#reconverting date of birth interval from list to interval
for i in range(len(openpowerlifting_USAPL_lifters)):
    if openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] != "Open" :
    →and pd.notna(openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]):
        list_date = openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"]
        →[1:-1].split(", ")
        openpowerlifting_USAPL_lifters.at[i, "DateOfBirthInterval"] = pd.
        →Interval(pd.Timestamp(list_date[0]), pd.Timestamp(list_date[1]), closed =
        →"both")
```

Calculating the number of meets in the past year

```
[2]: def number_of_meets_past_year(x):
    number = []
    for i in x.index:
        number.append(x[(x["Date"] < x.at[i, "Date"]) & (x["Date"] > x.
        →at[i, "Date"] - relativedelta(years = 1))].shape[0])
```

```

    return pd.Series(x[(x["Date"] < x.at[i, "Date"]) & (x["Date"] > x.
→at[i, "Date"] - relativedelta(years = 1))].shape[0] for i in x.index)

openpowerlifting_USAPL_lifters["Number of meets in past 12 months"] =_
→openpowerlifting_USAPL_lifters.groupby("Name")[["Date", "Name"]].apply(lambda_
→x: pd.DataFrame((x[(x["Date"] < x.at[i, "Date"]) & (x["Date"] > x.
→at[i, "Date"] - relativedelta(years = 1))].shape[0] for i in x.index), x.
→index))

```

If a lifter failed all their attempts, the “Best3’X’Kg” columns will be filled with a negative value representing the lifter’s highest failed attempt. This will be converted to zero to avoid anomalies when inserted into the model. Additionally, the “Best lifts to date” have this problem, therefore backfilling will occur for these too.

[3] :

```

openpowerlifting_USAPL_lifters["Best3SquatKg"] =_
→openpowerlifting_USAPL_lifters["Best3SquatKg"].
→where(openpowerlifting_USAPL_lifters["Best3SquatKg"] > 0, 0)
openpowerlifting_USAPL_lifters["Best3BenchKg"] =_
→openpowerlifting_USAPL_lifters["Best3BenchKg"].
→where(openpowerlifting_USAPL_lifters["Best3BenchKg"] > 0, 0)
openpowerlifting_USAPL_lifters["Best3DeadliftKg"] =_
→openpowerlifting_USAPL_lifters["Best3DeadliftKg"].
→where(openpowerlifting_USAPL_lifters["Best3DeadliftKg"] > 0, 0)

for column in openpowerlifting_USAPL_lifters.columns[-2:-25:-1]:
    openpowerlifting_USAPL_lifters[column] =_
→openpowerlifting_USAPL_lifters[column].
→where(openpowerlifting_USAPL_lifters[column] > 0, 0)

```

[4] :

```

openpowerlifting_USAPL_lifters.insert(11, "Squat1 Pass",_
→openpowerlifting_USAPL_lifters["Squat1Kg"] > 0)
openpowerlifting_USAPL_lifters["Squat1 Pass"] =_
→openpowerlifting_USAPL_lifters["Squat1 Pass"].astype(int)

openpowerlifting_USAPL_lifters.insert(13, "Squat2 Pass",_
→openpowerlifting_USAPL_lifters["Squat2Kg"] > 0)
openpowerlifting_USAPL_lifters["Squat2 Pass"] =_
→openpowerlifting_USAPL_lifters["Squat2 Pass"].astype(int)

openpowerlifting_USAPL_lifters.insert(15, "Squat3 Pass",_
→openpowerlifting_USAPL_lifters["Squat3Kg"] > 0)
openpowerlifting_USAPL_lifters["Squat3 Pass"] =_
→openpowerlifting_USAPL_lifters["Squat3 Pass"].astype(int)

```

```

openpowerlifting_USAPL_lifters.insert(19, "Bench1 Pass", ↴
    ↪openpowerlifting_USAPL_lifters["Bench1Kg"] > 0)
openpowerlifting_USAPL_lifters["Bench1 Pass"] = ↴
    ↪openpowerlifting_USAPL_lifters["Bench1 Pass"].astype(int)

```

```

openpowerlifting_USAPL_lifters.insert(21, "Bench2 Pass", ↴
    ↪openpowerlifting_USAPL_lifters["Bench2Kg"] > 0)
openpowerlifting_USAPL_lifters["Bench2 Pass"] = ↴
    ↪openpowerlifting_USAPL_lifters["Bench2 Pass"].astype(int)

```

```

openpowerlifting_USAPL_lifters.insert(23, "Bench3 Pass", ↴
    ↪openpowerlifting_USAPL_lifters["Bench3Kg"] > 0)
openpowerlifting_USAPL_lifters["Bench3 Pass"] = ↴
    ↪openpowerlifting_USAPL_lifters["Bench3 Pass"].astype(int)

```

```

openpowerlifting_USAPL_lifters.insert(27, "Deadlift1 Pass", ↴
    ↪openpowerlifting_USAPL_lifters["Deadlift1Kg"] > 0)
openpowerlifting_USAPL_lifters["Deadlift1 Pass"] = ↴
    ↪openpowerlifting_USAPL_lifters["Deadlift1 Pass"].astype(int)

```

```

openpowerlifting_USAPL_lifters.insert(29, "Deadlift2 Pass", ↴
    ↪openpowerlifting_USAPL_lifters["Deadlift2Kg"] > 0)
openpowerlifting_USAPL_lifters["Deadlift2 Pass"] = ↴
    ↪openpowerlifting_USAPL_lifters["Deadlift2 Pass"].astype(int)

```

```

openpowerlifting_USAPL_lifters.insert(31, "Deadlift3 Pass", ↴
    ↪openpowerlifting_USAPL_lifters["Deadlift3Kg"] > 0)
openpowerlifting_USAPL_lifters["Deadlift3 Pass"] = ↴
    ↪openpowerlifting_USAPL_lifters["Deadlift3 Pass"].astype(int)

```

Transforming nan values in the time since last meet column. Nan values are first meets, so we will insert the time between birth and the first meet. This time is large, so should be anomalous compared to the other times

```

[5]: # for i in openpowerlifting_USAPL_lifters.index:
#     if type(openpowerlifting_USAPL_lifters.at[i, "Time Since Last Meet"]) == ↴
#         pd._libs.tslibs.nattype.NaTType:
#             try:
#                 openpowerlifting_USAPL_lifters.at[i, "Time Since Last Meet"] = ↴
#                     openpowerlifting_USAPL_lifters["Date"][i] - ↴
#                     openpowerlifting_USAPL_lifters["DateOfBirthInterval"][i].left
#             except:
#                 for j in ↴
#                     openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] == ↴
#                         openpowerlifting_USAPL_lifters.at[i, "Name"]].index:

```

```

#             if (type(openpowerlifting_USAPL_lifters.
→at[j, "DateOfBirthInterval"]) == pd._libs.interval.Interval) and
→(type(openpowerlifting_USAPL_lifters.at[j, "Age"]) == np.float64):
#
#                 openpowerlifting_USAPL_lifters.at[i, "Time Since Last
→Meet"] = openpowerlifting_USAPL_lifters["Date"][j] -
→openpowerlifting_USAPL_lifters["DateOfBirthInterval"][j].left
#
#                 break
#
#             continue

# for i in openpowerlifting_USAPL_lifters.index:
#     if type(openpowerlifting_USAPL_lifters.at[i, "Time Since Last Meet"]) ==_
→pd._libs.tslibs.nattype.NaTType:
#         try:
#             openpowerlifting_USAPL_lifters.at[i, "Time Since Last Meet"] =
→openpowerlifting_USAPL_lifters["Date"][i] -
→openpowerlifting_USAPL_lifters["DateOfBirthInterval"][i].left
#
#         except:
#             for j in_
→openpowerlifting_USAPL_lifters[openpowerlifting_USAPL_lifters["Name"] ==_
→openpowerlifting_USAPL_lifters.at[i, "Name"]].index:
#                 if (type(openpowerlifting_USAPL_lifters.
→at[j, "DateOfBirthInterval"]) == pd._libs.interval.Interval) and
→(type(openpowerlifting_USAPL_lifters.at[j, "Age"]) == np.float64):
#
#                 openpowerlifting_USAPL_lifters.at[i, "Time Since Last
→Meet"] = openpowerlifting_USAPL_lifters["Date"][j] -
→openpowerlifting_USAPL_lifters["DateOfBirthInterval"][j].left
#
#                 break
#
#             continue

openpowerlifting_USAPL_lifters["Time Since Last Meet"] =
→openpowerlifting_USAPL_lifters["Time Since Last Meet"].dt.days

```

creating % of squat/deadlift/bench attempts made in the past 3 meets

```

[6]: meets_where_attempts_recorded = pd.
→Series([False]*openpowerlifting_USAPL_lifters.shape[0])
### below for loop are indicies of lifters where all attempts are recorded
for name in ["Squat1", "Squat2", "Squat3",
              "Bench1", "Bench2", "Bench3",
              "Deadlift1", "Deadlift2", "Deadlift3"]:
    meets_where_attempts_recorded = meets_where_attempts_recorded | pd.
→notna(openpowerlifting_USAPL_lifters[f"{'name'}Kg"])

for name in ["Squat1", "Squat2", "Squat3",
              "Bench1", "Bench2", "Bench3",
              "Deadlift1", "Deadlift2", "Deadlift3"]:
    openpowerlifting_USAPL_lifters[f"{'name'} Past 3 meets success rate"] = np.nan

```

```

openpowerlifting_USAPL_lifters.loc[
    (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
    & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
    & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
    & (openpowerlifting_USAPL_lifters["Date"] > pd.to_datetime("01/01/2014")) \
    & (pd.notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
    & (meets_where_attempts_recorded)
        , [f"{name} Past 3 meets success rate"]]
    = openpowerlifting_USAPL_lifters.
    loc[(openpowerlifting_USAPL_lifters["Event"] == "SBD") \
        & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
        & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
        & (openpowerlifting_USAPL_lifters["Date"] > pd.to_datetime("01/01/2014")) \
        & (pd.notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
        & (meets_where_attempts_recorded), :].
    groupby("Name")[f"{name} Pass"].rolling(3, min_periods = 1).mean().
    reset_index(0)[f"{name} Pass"]
        openpowerlifting_USAPL_lifters.loc[:, f"{name} Past 3 meets success rate"] =
    openpowerlifting_USAPL_lifters.groupby("Name")[f"{name} Past 3 meets success rate"].shift()
        openpowerlifting_USAPL_lifters.loc[:, f"{name} Past 3 meets success rate"] =
    openpowerlifting_USAPL_lifters.groupby("Name")[f"{name} Past 3 meets success rate"].ffill()

```

calculating overall attempt success rate and impute missing values

```

[7]: # from sklearn.experimental import enable_iterative_imputer
# from sklearn.impute import IterativeImputer

# imputer = IterativeImputer(random_state = 0, max_iter = 100)
# imputer.fit(openpowerlifting_USAPL_lifters.loc[
#     (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
#     & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
#     & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
#     & (openpowerlifting_USAPL_lifters["Date"] > pd.to_datetime("01/01/2014")) \

```

```

#                                     & (openpowerlifting_USAPL_lifters["Date"] < pd.
#   → to_datetime("07/01/2019")) \
#                                     & (pd.
#   → notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
#                                     & (meets_where_attempts_recorded)
#                                     , ["Male", "Female", "Age", "BodyweightKg", □
#   → "Number of meets in past 12 months", "Number of Past Meets Same Day" □
#   → Exclusive"] + list(map(lambda x: x + " Past 3 meets success rate", □
#   → ["Squat1", "Squat2", "Squat3",
#       "Bench1", "Bench2", "Bench3",
#       "Deadlift1", "Deadlift2", "Deadlift3"])))]

# openpowerlifting_USAPL_lifters.loc[
#                                     (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
#                                     & (openpowerlifting_USAPL_lifters["Equipment"] == □
#   → "Raw") \
#                                     & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
#                                     & (openpowerlifting_USAPL_lifters["Date"] > pd.
#   → to_datetime("01/01/2014")) \
#                                     & (pd.
#   → notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
#                                     & (meets_where_attempts_recorded)
#                                     , ["Male", "Female", "Age", "BodyweightKg", □
#   → "Number of meets in past 12 months", "Number of Past Meets Same Day" □
#   → Exclusive"] + list(map(lambda x: x + " Past 3 meets success rate", □
#   → ["Squat1", "Squat2", "Squat3",
#       "Bench1", "Bench2", "Bench3",
#       "Deadlift1", "Deadlift2", "Deadlift3"]))] = \
# imputer.transform(openpowerlifting_USAPL_lifters.loc[
#                                     (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
#                                     & (openpowerlifting_USAPL_lifters["Equipment"] == □
#   → "Raw") \
#                                     & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
#                                     & (openpowerlifting_USAPL_lifters["Date"] > pd.
#   → to_datetime("01/01/2014")) \
#                                     & (pd.
#   → notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
#                                     & (meets_where_attempts_recorded)
#                                     , ["Male", "Female", "Age", "BodyweightKg", □
#   → "Number of meets in past 12 months", "Number of Past Meets Same Day" □
#   → Exclusive"] + list(map(lambda x: x + " Past 3 meets success rate", □
#   → ["Squat1", "Squat2", "Squat3",
#       "Bench1", "Bench2", "Bench3",
#       "Deadlift1", "Deadlift2", "Deadlift3"])))

# openpowerlifting_USAPL_lifters.loc[
```

```

# (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
# & (openpowerlifting_USAPL_lifters["Equipment"] ==_
#   ↪ "Raw") \
# & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
# & (openpowerlifting_USAPL_lifters["Date"] > pd.
#   ↪ to_datetime("01/01/2014")) \
# & (pd.
#   ↪ notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
# & (meets_where_attempts_recorded)
# , ["Male", "Female", "Age", "BodyweightKg",_
#   ↪ "Number of meets in past 12 months", "Number of Past Meets Same Day"]
# ↪ Exclusive] + list(map(lambda x: x + " Past 3 meets success rate",_
#   ↪ ["Squat1", "Squat2", "Squat3",
#     "Bench1", "Bench2", "Bench3",
#     "Deadlift1", "Deadlift2", "Deadlift3"]))
# openpowerlifting_USAPL_lifters["Overall Attempt Success Rate over past three"
#   ↪ meets"] = 0

for name in ["Squat1", "Squat2", "Squat3",
    "Bench1", "Bench2", "Bench3",
    "Deadlift1", "Deadlift2", "Deadlift3"]:
    openpowerlifting_USAPL_lifters["Overall Attempt Success Rate over past"
    ↪ three meets"] += openpowerlifting_USAPL_lifters[f"{name} Past 3 meets"
    ↪ success rate"]

openpowerlifting_USAPL_lifters["Overall Attempt Success Rate over past three"
    ↪ meets"] /= 9

```

[8]:

```

openpowerlifting_USAPL_lifters.loc[:, "Squat1Kg": "Squat4Kg"] =_
    ↪ openpowerlifting_USAPL_lifters.loc[:, "Squat1Kg": "Squat4Kg"].abs()
openpowerlifting_USAPL_lifters.loc[:, "Bench1Kg": "Bench4Kg"] =_
    ↪ openpowerlifting_USAPL_lifters.loc[:, "Bench1Kg": "Bench4Kg"].abs()
openpowerlifting_USAPL_lifters.loc[:, "Deadlift1Kg": "Deadlift4Kg"] =_
    ↪ openpowerlifting_USAPL_lifters.loc[:, "Deadlift1Kg": "Deadlift4Kg"].abs()
#openpowerlifting_USAPL_lifters.loc[:, "Best Raw Squat to date": "Best Unlimited"
    ↪ Wilks to date"] = openpowerlifting_USAPL_lifters.loc[:, "Best Raw Squat to"
    ↪ date": "Best Unlimited Wilks to date"].fillna(0)

```

[]:

[9]:

```

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] ==_
    ↪ "USAPL") \
    & (openpowerlifting_USAPL_lifters["Event"] ==_
    ↪ "SBD")]

```

```

    & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
        & (pd.
    →notna(openpowerlifting_USAPL_lifters["Age"])) \
        & (openpowerlifting_USAPL_lifters["Date"] > pd.
    →to_datetime("01/01/2014")) \
        & (pd.
    →notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
        & (pd.
    →notna(openpowerlifting_USAPL_lifters["Squat1Kg"])) \
        ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Glossbrenner", "Goodlift", "Tested", "Country", "State", "Federation", "ParentFederation", "MeetName", "Repeated_meet", "Two_meets_on_one_day", "DateOfBirthInterval"]).to_csv("C:/Users/Montel/Google Drive/Data Science MSc/Msc-Project---Powerlifting/Datasets/attempt subsets/Squat1Kgnulls.csv", index= False)
openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \
    & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
    & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
        & (pd.
    →notna(openpowerlifting_USAPL_lifters["Age"])) \
        & (openpowerlifting_USAPL_lifters["Date"] > pd.
    →to_datetime("01/01/2014")) \
        & (pd.
    →notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
        & (pd.
    →notna(openpowerlifting_USAPL_lifters["Bench1Kg"])) \
        ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Glossbrenner", "Goodlift", "Tested", "Country", "State", "Federation", "ParentFederation", "MeetName", "Repeated_meet", "Two_meets_on_one_day", "DateOfBirthInterval"]).to_csv("C:/Users/Montel/Google Drive/Data Science MSc/Msc-Project---Powerlifting/Datasets/attempt subsets/Bench1Kgnulls.csv", index= False)

```

```

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \
                                & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
                                & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
                                & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
                                & (openpowerlifting_USAPL_lifters["Date"] > pd.to_datetime("01/01/2014")) \
                                & (pd.notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
                                & (pd.notna(openpowerlifting_USAPL_lifters["Deadlift1Kg"])) \
                                ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Glossbrenner", "Goodlift", "ParentFederation", "Tested", "Country", "State", "Federation", "MeetName", "Repeated_meet", "Two_meets_on_one_day", "DateOfBirthInterval"]).to_csv("C:/Users/Montel/Google Drive/Data Science MSc/Msc-Project---Powerlifting/Datasets/attempt subsets/Deadlift1Kgnulls.csv", index= False)

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \
                                & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
                                & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
                                & (pd.notna(openpowerlifting_USAPL_lifters["Age"])) \
                                & (openpowerlifting_USAPL_lifters["Date"] > pd.to_datetime("01/01/2014")) \
                                & (pd.notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
                                & (pd.notna(openpowerlifting_USAPL_lifters["Squat2Kg"])) \
                                ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Glossbrenner", "Goodlift", "ParentFederation", "Tested", "Country", "State", "Federation", "MeetName", "Repeated_meet", "Two_meets_on_one_day"],

```

```

        "DateOfBirthInterval"]).

→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Squat2Kgnulls.csv", □
→index= False)

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == □
→"USAPL")\

& (openpowerlifting_USAPL_lifters["Event"] == □
→"SBD")\

& (openpowerlifting_USAPL_lifters["Equipment"] □
→== "Raw")\

& (pd.

→notna(openpowerlifting_USAPL_lifters["Age"])) \

& (openpowerlifting_USAPL_lifters["Date"] > pd.

→to_datetime("01/01/2014")) \

& (pd.

→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \

& (pd.

→notna(openpowerlifting_USAPL_lifters["Bench2Kg"])) \

].drop(columns = ["Event", "Equipment", □
→"AgeClass", "BirthYearClass", "Division",
→"Glossbrenner", "Goodlift", □
→"Tested", "Country", "State", "Federation",
→"ParentFederation", □
→"MeetName", "Repeated_meet", "Two_meets_on_one_day",
→"DateOfBirthInterval"]).

→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Bench2Kgnulls.csv", □
→index= False)

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == □
→"USAPL")\

& (openpowerlifting_USAPL_lifters["Event"] == □
→"SBD")\

& (openpowerlifting_USAPL_lifters["Equipment"] □
→== "Raw")\

& (pd.

→notna(openpowerlifting_USAPL_lifters["Age"])) \

& (openpowerlifting_USAPL_lifters["Date"] > pd.

→to_datetime("01/01/2014")) \

& (pd.

→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \

& (pd.

→notna(openpowerlifting_USAPL_lifters["Deadlift2Kg"])) \

].drop(columns = ["Event", "Equipment", □
→"AgeClass", "BirthYearClass", "Division",
→"Glossbrenner", "Goodlift", □
→"Tested", "Country", "State", "Federation",
→"ParentFederation", □
→"MeetName", "Repeated_meet", "Two_meets_on_one_day",
→"DateOfBirthInterval"]).

```

```

"Glossbrenner", "Goodlift", □
→"Tested", "Country", "State", "Federation",
                                         "ParentFederation", □
→"MeetName", "Repeated_meet", "Two_meets_on_one_day",
                                         "DateOfBirthInterval"]).

→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Deadlift2Kgnulls.csv", □
→index= False)

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == □
→"USAPL") \
                                         & (openpowerlifting_USAPL_lifters["Event"] == □
→"SBD") \
                                         & (openpowerlifting_USAPL_lifters["Equipment"] □
→== "Raw") \
                                         & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \
                                         & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \
                                         & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
                                         & (pd.
→notna(openpowerlifting_USAPL_lifters["Squat3Kg"])) \
                                         ].drop(columns = ["Event", "Equipment", □
→"AgeClass", "BirthYearClass", "Division",
                                         "Glossbrenner", "Goodlift", □
→"Tested", "Country", "State", "Federation",
                                         "ParentFederation", □
→"MeetName", "Repeated_meet", "Two_meets_on_one_day",
                                         "DateOfBirthInterval"]).

→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Squat3Kgnulls.csv", □
→index= False)

openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == □
→"USAPL") \
                                         & (openpowerlifting_USAPL_lifters["Event"] == □
→"SBD") \
                                         & (openpowerlifting_USAPL_lifters["Equipment"] □
→== "Raw") \
                                         & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \
                                         & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \
                                         & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \

```

```

        & (pd.
→notna(openpowerlifting_USAPL_lifters["Bench3Kg"])) \ 
            ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Goodlift", "Glossbrenner", "ParentFederation", "MeetName", "Repeated_meet", "Two_meets_on_one_day", "DateOfBirthInterval"]).
→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Bench3Kgnulls.csv", index= False)
openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \
        & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
        & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
        & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \
            & (openpowerlifting_USAPL_lifters["Date"] > pd.
→to_datetime("01/01/2014")) \
                & (pd.
→notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \
                    & (pd.
→notna(openpowerlifting_USAPL_lifters["Deadlift3Kg"])) \
                        ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division", "Goodlift", "Glossbrenner", "ParentFederation", "MeetName", "Repeated_meet", "Two_meets_on_one_day", "DateOfBirthInterval"]).
→to_csv("C:/Users/Montel/Google Drive/Data Science MSc/
→Msc-Project---Powerlifting/Datasets/attempt subsets/Deadlift3Kgnulls.csv", index= False)

```

[10]: openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Federation"] == "USAPL") \
 & (openpowerlifting_USAPL_lifters["Event"] == "SBD") \
 & (openpowerlifting_USAPL_lifters["Equipment"] == "Raw") \
 & (pd.
→notna(openpowerlifting_USAPL_lifters["Age"])) \

```
& (openpowerlifting_USAPL_lifters["Date"] > pd.  
to_datetime("01/01/2014")) \\  
    & (pd.  
notna(openpowerlifting_USAPL_lifters["BodyweightKg"])) \\  
        ].drop(columns = ["Event", "Equipment", "AgeClass", "BirthYearClass", "Division",  
        "Glossbrenner", "Goodlift", "Tested", "Country", "State", "Federation",  
        "ParentFederation", "MeetName", "Repeated_meet", "Two_meets_on_one_day",  
        "DateOfBirthInterval"]).  
to_csv("C:/Users/Montel/Google Drive/Data Science MSc/  
Msc-Project---Powerlifting/Datasets/attempt subsets/Full Setnulls.csv",  
index= False)
```

[]:

Appendix G Exploratory Data analysis script and results

a. EDA Script

Appendix_F_All EDA

September 15, 2021

```
[1]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind
import statsmodels.api as sm
import matplotlib.ticker as plticker
import datetime
import locale
locale.setlocale(locale.LC_ALL, '')
from matplotlib import rcParams
rcParams['figure.figsize'] = 11.7,8.27
```

```
[2]: openpowerlifting_USAPL_lifters = pd.read_csv("C:/Users/Montel/Google Drive/Data
→Science MSc/Msc-Project---Powerlifting/Datasets/Attempt subsets/Full set.
→csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
→to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
openpowerlifting_USAPL_lifters["Squat2 Jump"] =_
→(openpowerlifting_USAPL_lifters["Squat2Kg"] -_
→openpowerlifting_USAPL_lifters["Squat1Kg"])/
→openpowerlifting_USAPL_lifters["Squat1Kg"]
openpowerlifting_USAPL_lifters["Squat3 Jump"] =_
→(openpowerlifting_USAPL_lifters["Squat3Kg"] -_
→openpowerlifting_USAPL_lifters["Squat2Kg"])/
→openpowerlifting_USAPL_lifters["Squat2Kg"]
openpowerlifting_USAPL_lifters["Total Squat Jump"] =_
→(openpowerlifting_USAPL_lifters["Squat3Kg"] -_
→openpowerlifting_USAPL_lifters["Squat1Kg"])/
→openpowerlifting_USAPL_lifters["Squat1Kg"]
openpowerlifting_USAPL_lifters["Bench2 Jump"] =_
→(openpowerlifting_USAPL_lifters["Bench2Kg"] -_
→openpowerlifting_USAPL_lifters["Bench1Kg"])/
→openpowerlifting_USAPL_lifters["Bench1Kg"]
```

```

openpowerlifting_USAPL_lifters["Bench3 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Bench3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Bench2Kg"])/
    ↳openpowerlifting_USAPL_lifters["Bench2Kg"]

openpowerlifting_USAPL_lifters["Total Bench Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Bench3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Bench1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Bench1Kg"]

openpowerlifting_USAPL_lifters["Deadlift2 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Deadlift2Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"]

openpowerlifting_USAPL_lifters["Deadlift3 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Deadlift3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Deadlift2Kg"])/
    ↳openpowerlifting_USAPL_lifters["Deadlift2Kg"]

openpowerlifting_USAPL_lifters["Total Deadlift Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Deadlift3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"]

openpowerlifting_USAPL_lifters =
    ↳openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters[["Squat1Kg", 'Squat2Kg', 'Squat3Kg'],
    ↳"Bench1Kg", 'Bench2Kg', 'Bench3Kg'],
    ↳"Deadlift1Kg", 'Deadlift2Kg', 'Deadlift3Kg']].notnull().sum(axis=1) > 3) &
    ↳(openpowerlifting_USAPL_lifters[["Squat1Kg", 'Squat2Kg', 'Squat3Kg',]].notnull().sum(axis=1) > 0) &
    ↳(openpowerlifting_USAPL_lifters[["Bench1Kg", 'Bench2Kg', 'Bench3Kg',]].notnull().sum(axis=1) > 0) &
    ↳(openpowerlifting_USAPL_lifters[["Deadlift1Kg", 'Deadlift2Kg', 'Deadlift3Kg',]].notnull().sum(axis=1) > 0)

```

```
[3]: predicting_columns = list(openpowerlifting_USAPL_lifters.columns.drop(['Name',_
    ↳'WeightClassKg',
    'Squat4Kg',
    'Bench4Kg',
    'Deadlift3 Pass',
    'Deadlift4Kg',
    'Best3DeadliftKg',
    'TotalKg',
    'Place',
```

```
'Dots',
'Wilks',
'Date',
'MeetCountry',
'MeetState',
'MeetTown',
'Best M-Ply Squat to date',
'Best M-Ply Bench to date',
'Best M-Ply Deadlift to date',
'Best M-Ply Total to date',
'Best M-Ply Wilks to date',
'Best Unlimited Squat to date',
'Best Unlimited Bench to date',
'Best Unlimited Deadlift to date',
'Best Unlimited Total to date',
'Best Unlimited Wilks to date',]))
```

[4]: predicting_columns

```
[4]: ['Male',
'Female',
'Age',
'BodyweightKg',
'Squat1Kg',
'Squat1 Pass',
'Squat2Kg',
'Squat2 Pass',
'Squat3Kg',
'Squat3 Pass',
'Best3SquatKg',
'Bench1Kg',
'Bench1 Pass',
'Bench2Kg',
'Bench2 Pass',
'Bench3Kg',
'Bench3 Pass',
'Best3BenchKg',
'Deadlift1Kg',
'Deadlift1 Pass',
'Deadlift2Kg',
'Deadlift2 Pass',
'Deadlift3Kg',
'Time Since Last Meet',
'Number of Past Meets',
'Number of Past Meets Same Day Exclusive',
'Best Raw Squat to date',
'Best Wrapped Squat to date',
```

```
'Best Raw Bench to date',
'Best Raw Deadlift to date',
'Best Raw Total to date',
'Best Raw Wilks to date',
'Best Wrapped Total to date',
'Best Wrapped Wilks to date',
'Best S-Ply Squat to date',
'Best S-Ply Bench to date',
'Best S-Ply Deadlift to date',
'Best S-Ply Total to date',
'Best S-Ply Wilks to date',
'Number of meets in past 12 months',
'Squat1 Past 3 meets success rate',
'Squat2 Past 3 meets success rate',
'Squat3 Past 3 meets success rate',
'Bench1 Past 3 meets success rate',
'Bench2 Past 3 meets success rate',
'Bench3 Past 3 meets success rate',
'Deadlift1 Past 3 meets success rate',
'Deadlift2 Past 3 meets success rate',
'Deadlift3 Past 3 meets success rate',
'Overall Attempt Success Rate over past three meets',
'Squat2 Jump',
'Squat3 Jump',
'Total Squat Jump',
'Bench2 Jump',
'Bench3 Jump',
'Total Bench Jump',
'Deadlift2 Jump',
'Deadlift3 Jump',
'Total Deadlift Jump']
```

```
[ ]:
```

```
[5]: train = openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] < pd.to_datetime("07/01/2019"))]
valid = openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] >= pd.to_datetime("07/01/2019")) & (openpowerlifting_USAPL_lifters["Date"] < pd.to_datetime("11/15/2019"))]
test = openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] >= pd.to_datetime("11/15/2019")) & (openpowerlifting_USAPL_lifters["Date"] < pd.to_datetime("04/01/2020"))]
train_and_valid = openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] < pd.to_datetime("11/15/2019"))]
print(train.shape[0], valid.shape[0], test.shape[0])
```

```

train_x, valid_x, train_y, valid_y = train[predicting_columns], □
↳valid[predicting_columns], train["Deadlift3 Pass"], valid["Deadlift3 Pass"]
train_and_valid_x, train_and_valid_y = train_and_valid[predicting_columns], □
↳train_and_valid["Deadlift3 Pass"]
test_x, test_y = test[predicting_columns], test["Deadlift3 Pass"]

```

60417 6436 6106

```

[55]: def graphing_function(metric):
    fig, ax = plt.subplots()
    bins = pd.qcut(train[pd.notna(train[metric])][metric], 5, duplicates = □
↳"drop")

    print(train[pd.notna(train[metric])].groupby(bins).size())

    for i in ["Squat1 Pass", "Squat2 Pass", "Squat3 Pass", "Bench1 Pass", □
↳"Bench2 Pass", "Bench3 Pass", "Deadlift1 Pass", "Deadlift2 Pass", "Deadlift3" □
↳"Pass",]:
        if (i in ["Squat1 Pass", "Squat2 Pass", "Squat3 Pass"]) and ((("Bench" □
↳in metric) or ("Deadlift" in metric)):
            continue
        elif (i in ["Bench1 Pass", "Bench2 Pass", "Bench3 Pass"]) and □
↳(("Deadlift" in metric)):
            continue
        elif (i in ["Squat1 Pass", "Squat2 Pass", "Squat3 Pass"]) and metric == □
↳"Squat3 Pass":
            continue
        elif (i in ["Squat1 Pass", "Squat2 Pass"]) and metric in ["Squat2" □
↳"Pass", "Squat3Kg", "Squat3 Jump", "Total Squat Jump"]:
            continue
        elif (i in ["Squat1 Pass"]) and metric in ["Squat1 Pass", "Squat2Kg", □
↳"Squat2 Jump", 'Squat2 Past 3 meets success rate', 'Squat3 Past 3 meets' □
↳'success rate']:
            continue
        elif (i in ["Squat2 Pass"]) and metric in ['Squat1 Past 3 meets success' □
↳'rate', 'Squat3 Past 3 meets success rate']:
            continue
        elif (i in ["Squat3 Pass"]) and metric in ['Squat1 Past 3 meets success' □
↳'rate', 'Squat2 Past 3 meets success rate']:
            continue
        elif (i in ["Bench1 Pass", "Bench2 Pass", "Bench3 Pass"]) and metric in □
↳(("Bench3 Pass", "Squat2 Jump", "Squat3 Jump")):
            continue
        elif (i in ["Bench1 Pass", "Bench2 Pass"]) and metric in ["Bench2" □
↳"Pass", "Bench3Kg", "Bench3 Jump", "Total Bench Jump"]:
            continue

```

```

        elif (i in ["Bench1 Pass"]) and metric in ["Bench1 Pass", "Bench2Kg", "Bench2 Jump", 'Bench2 Past 3 meets success rate', 'Bench3 Past 3 meets success rate', 'Squat1 Past 3 meets success rate', 'Squat2 Past 3 meets success rate', 'Squat3 Past 3 meets success rate', 'Best Raw Squat to date', ]:
            continue
        elif (i in ["Bench2 Pass"]) and metric in ['Bench1 Past 3 meets success rate', 'Bench3 Past 3 meets success rate', 'Squat1 Past 3 meets success rate', 'Squat2 Past 3 meets success rate', 'Squat3 Past 3 meets success rate']:
            continue
        elif (i in ["Bench3 Pass"]) and metric in ['Bench1 Past 3 meets success rate', 'Bench2 Past 3 meets success rate', 'Squat1 Past 3 meets success rate', 'Squat2 Past 3 meets success rate', 'Squat3 Past 3 meets success rate']:
            continue
        elif (i in ["Deadlift1 Pass", "Deadlift2 Pass", "Deadlift3 Pass"]) and metric in ("Deadlift3 Pass", "Squat2 Jump", "Squat3 Jump", "Bench2 Jump", "Bench3 Jump", 'Best Raw Squat to date', 'Best Raw Bench to date'):
            continue
        elif (i in ["Deadlift1 Pass", "Deadlift2 Pass"]) and metric in ["Deadlift2 Pass", "Deadlift3Kg", "Deadlift3 Jump", "Total"]:
            continue
        elif (i in ["Deadlift1 Pass"]) and metric in ["Deadlift1 Pass", "Deadlift2Kg", "Deadlift2 Jump", "Deadlift2 Past 3 meets success rate", 'Deadlift3 Past 3 meets success rate', 'Squat1 Past 3 meets success rate', 'Squat2 Past 3 meets success rate', 'Squat3 Past 3 meets success rate', 'Bench1 Past 3 meets success rate', 'Bench2 Past 3 meets success rate', 'Bench3 Past 3 meets success rate']:
            continue
        elif (i in ["Deadlift2 Pass"]) and metric in ['Deadlift1 Past 3 meets success rate', 'Deadlift3 Past 3 meets success rate', 'Squat1 Past 3 meets success rate', 'Squat2 Past 3 meets success rate', 'Squat3 Past 3 meets success rate', 'Bench1 Past 3 meets success rate', 'Bench2 Past 3 meets success rate', 'Bench3 Past 3 meets success rate']:
            continue
        elif (i in ["Deadlift3 Pass"]) and metric in ['Deadlift1 Past 3 meets success rate', 'Deadlift2 Past 3 meets success rate', 'Squat1 Past 3 meets success rate', 'Squat2 Past 3 meets success rate', 'Squat3 Past 3 meets success rate', 'Bench1 Past 3 meets success rate', 'Bench2 Past 3 meets success rate', 'Bench3 Past 3 meets success rate']:
            continue

```

```

        continue

    else:
        if len(train[pd.notna(train[metric])].groupby(bins).size()) == 1:
            sns.lineplot(data=(train.groupby(metric)[i].sum() / train.
→groupby(metric)[i].count()), label = i)
            ax.set_xticks([0,1])
            ax.set_xticklabels(["False", "True"])
        else:
            ax.plot([str(j) for j in (train[pd.notna(train[metric])].
→groupby(bins)[i].sum() / train[pd.notna(train[metric])].groupby(bins)[i].
→count()).index], (train[pd.notna(train[metric])].groupby(bins)[i].sum() /
→train[pd.notna(train[metric])].groupby(bins)[i].count()).values, label=i)

    ax.legend()
    ax.set_xlabel(metric)
    ax.set_ylabel("Attempt Success Rate")

plt.show()

#     plot = sns.lineplot(y = (train.groupby(bins)["Deadlift3 Pass"].sum() /
→train.groupby(bins)["Deadlift3 Pass"].count()).values, x = [str(i) for i in
→(train.groupby(bins)["Deadlift3 Pass"].sum() / train.
→groupby(bins)["Deadlift3 Pass"].count()).index])
#     plot.set(xlabel = metric, ylabel = "Average chance of hitting Deadlift3")
#     print(plot)
#     print(train.groupby(bins).size())

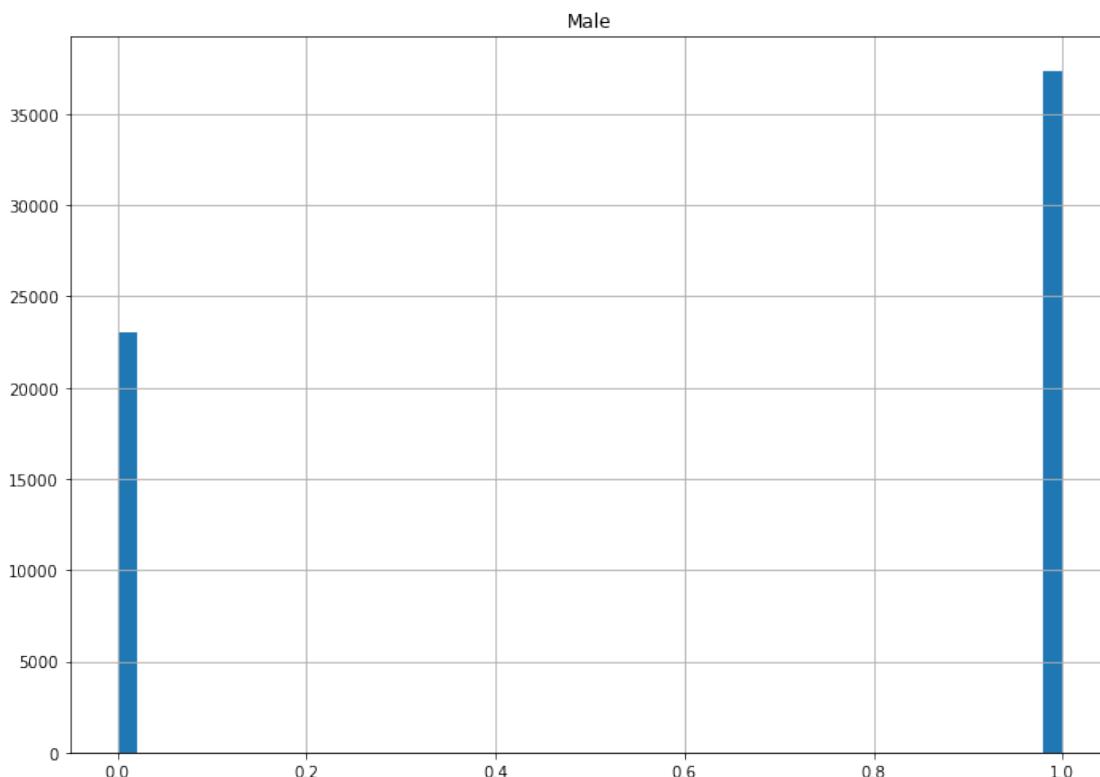
for i in ["Male", "BodyweightKg", "Age",
          "Time Since Last Meet", "Number of Past Meets", "Number of Past Meets",
→Same Day Exclusive",
          "Number of meets in past 12 months",
          "Squat1Kg", "Squat2Kg", "Squat3Kg", "Bench1Kg", "Bench2Kg",
→"Bench3Kg",
          "Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg",
          "Squat1 Pass", "Squat2 Pass", "Squat3 Pass", "Bench1 Pass", "Bench2
→Pass", "Bench3 Pass", "Deadlift1 Pass",
          "Deadlift2 Pass",
          "Squat2 Jump", "Squat3 Jump", "Total Squat Jump", "Bench2 Jump",
→"Bench3 Jump", "Total Bench Jump",
          "Deadlift2 Jump", "Deadlift3 Jump", "Total Deadlift Jump",
          'Best Raw Squat to date', 'Best Raw Bench to date',
          'Best Raw Deadlift to date', 'Best Raw Total to date', "Best Raw
→Wilks to date",
          'Squat1 Past 3 meets success rate', 'Squat2 Past 3 meets success
→rate', 'Squat3 Past 3 meets success rate',

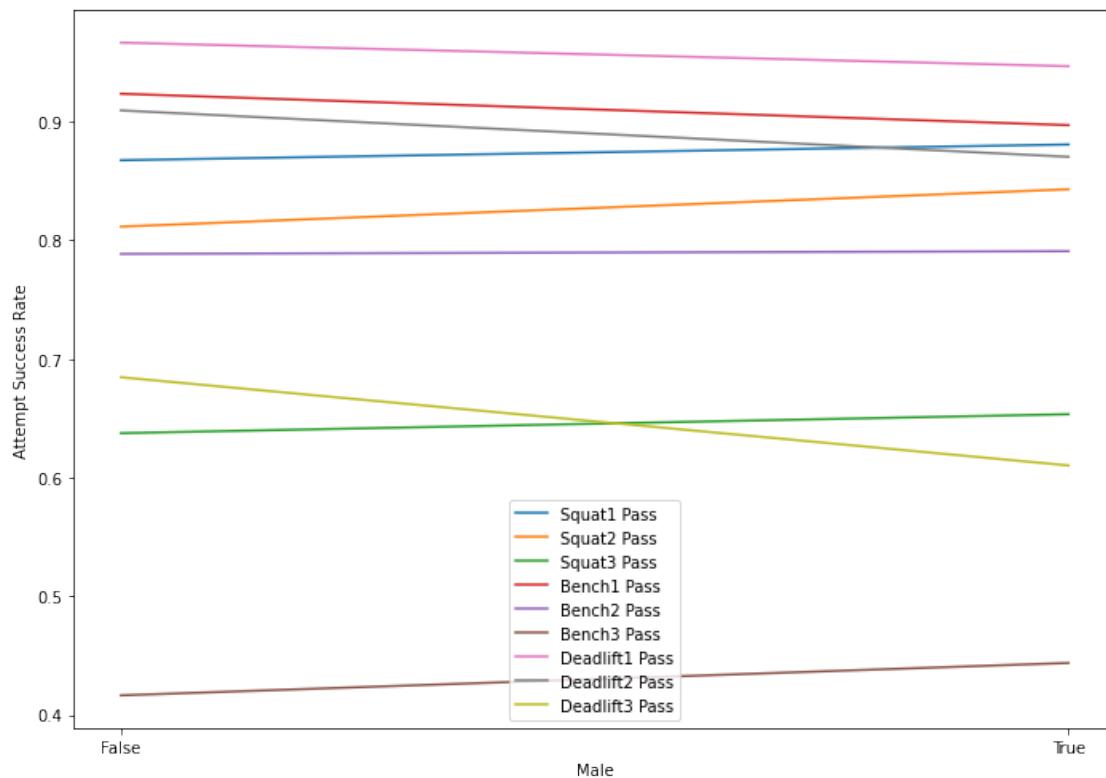
```

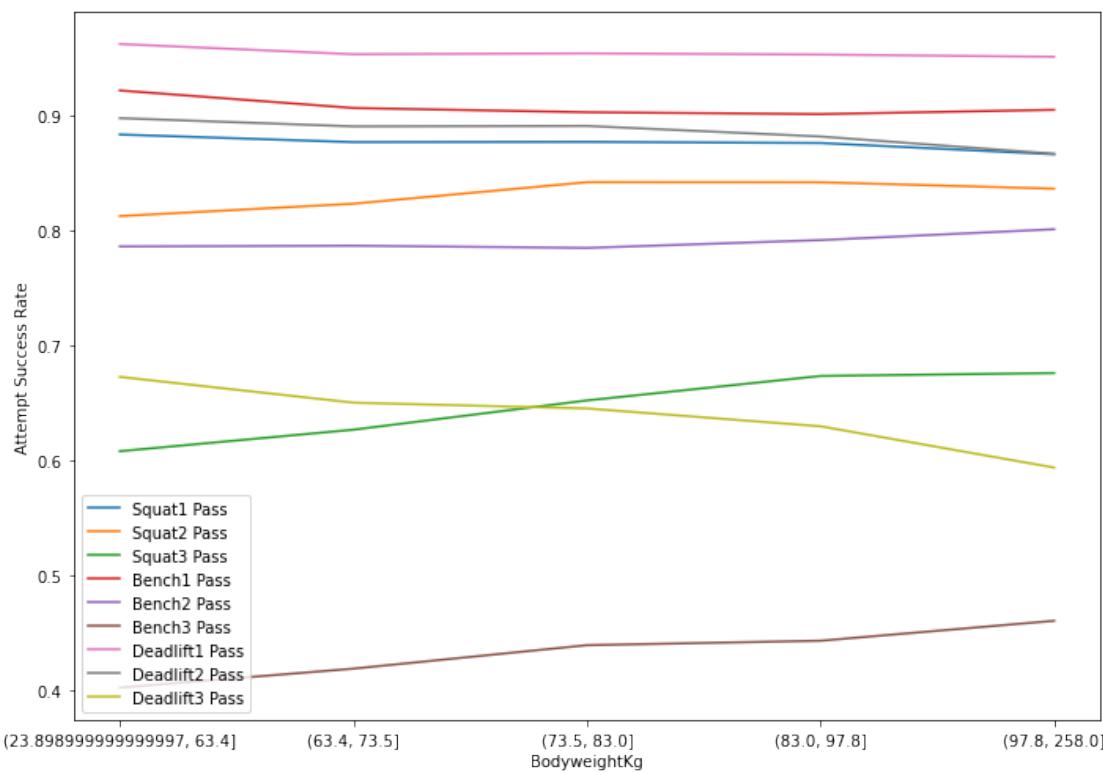
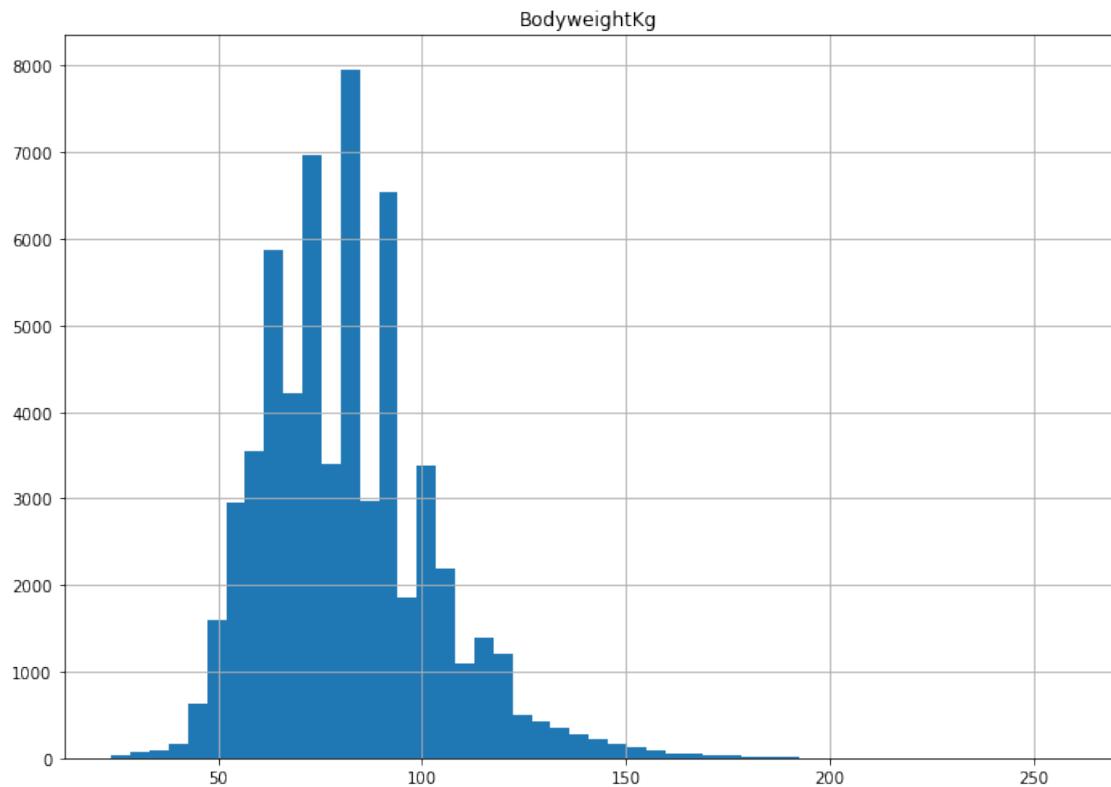
```

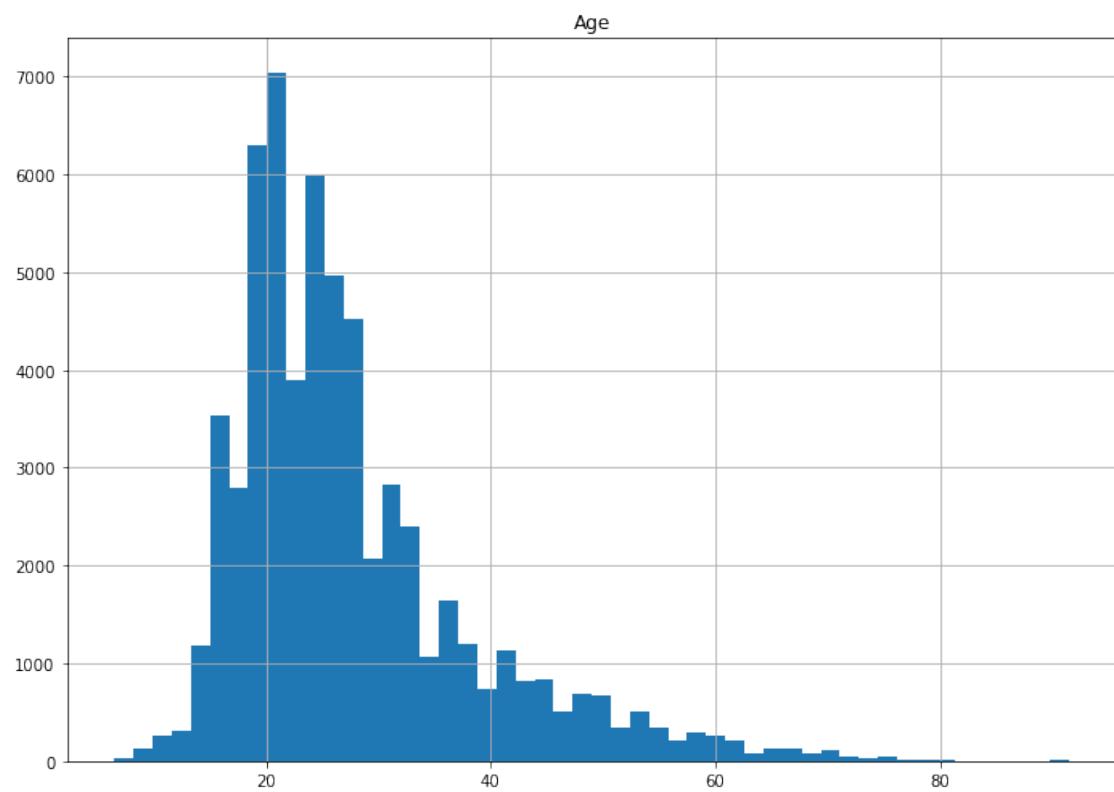
        'Bench1 Past 3 meets success rate', 'Bench2 Past 3 meets success_',
        ↪rate', 'Bench3 Past 3 meets success rate',
        'Deadlift1 Past 3 meets success rate','Deadlift2 Past 3 meets success_',
        ↪rate','Deadlift3 Past 3 meets success rate',
        "Overall Attempt Success Rate over past three meets",]:\n
if i == "Male":\n
    print("Lifter Characteristics")\n
if i == "Squat1Kg":\n
    print("Weights lifted per attempt")\n
if i == "Squat2 Jump":\n
    print("Weight Jumps")\n
if i == "Best Raw Squat to date":\n
    print("Personal Bests")\n
if i == 'Squat1 Past 3 meets success rate':\n
    print("Success Rates")\n
print(train.hist(i, bins = 50))\n
graphing_function(i)\n\n
# fig, ax = plt.subplots()\n# ax.plot(activity, dog, label="dog")\n# ax.plot(activity, cat, label="cat")\n# ax.legend()\n\n
# plt.show()

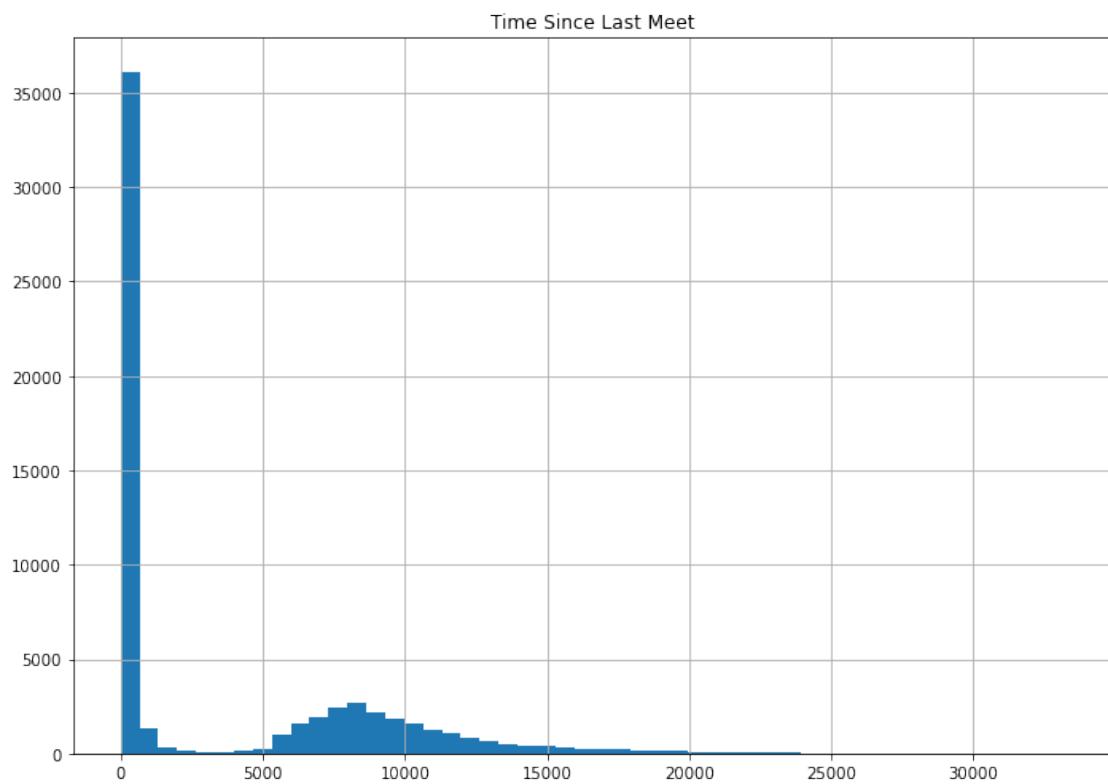
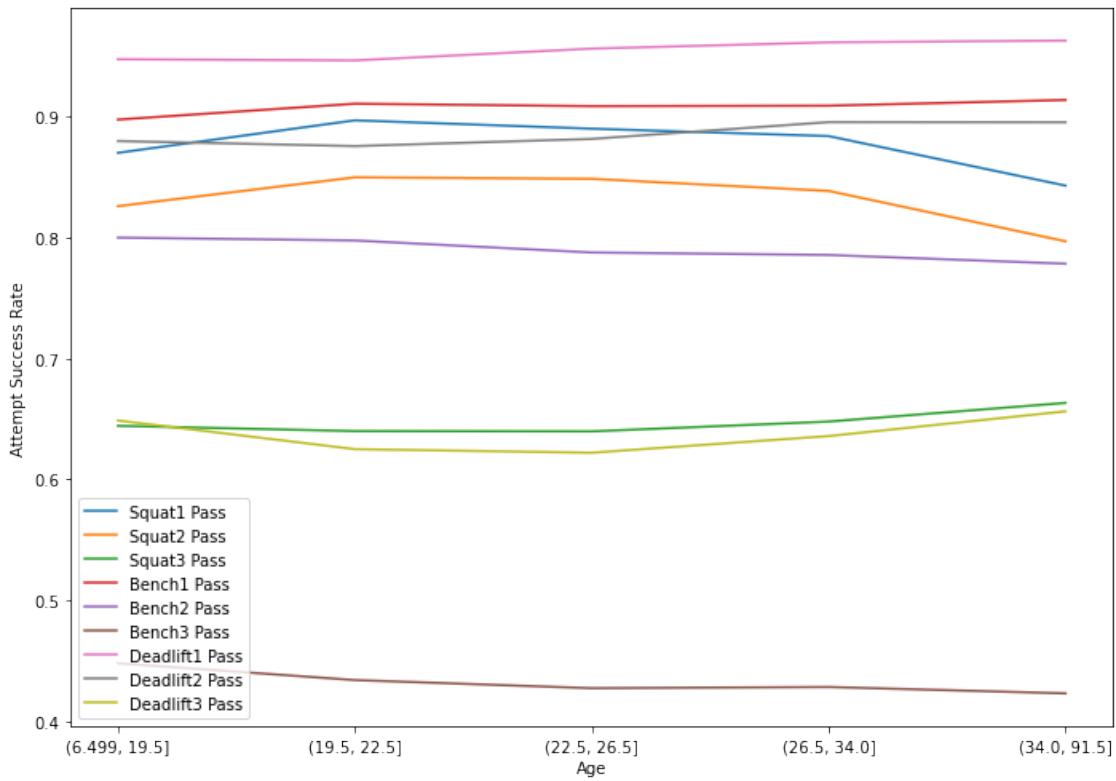
```

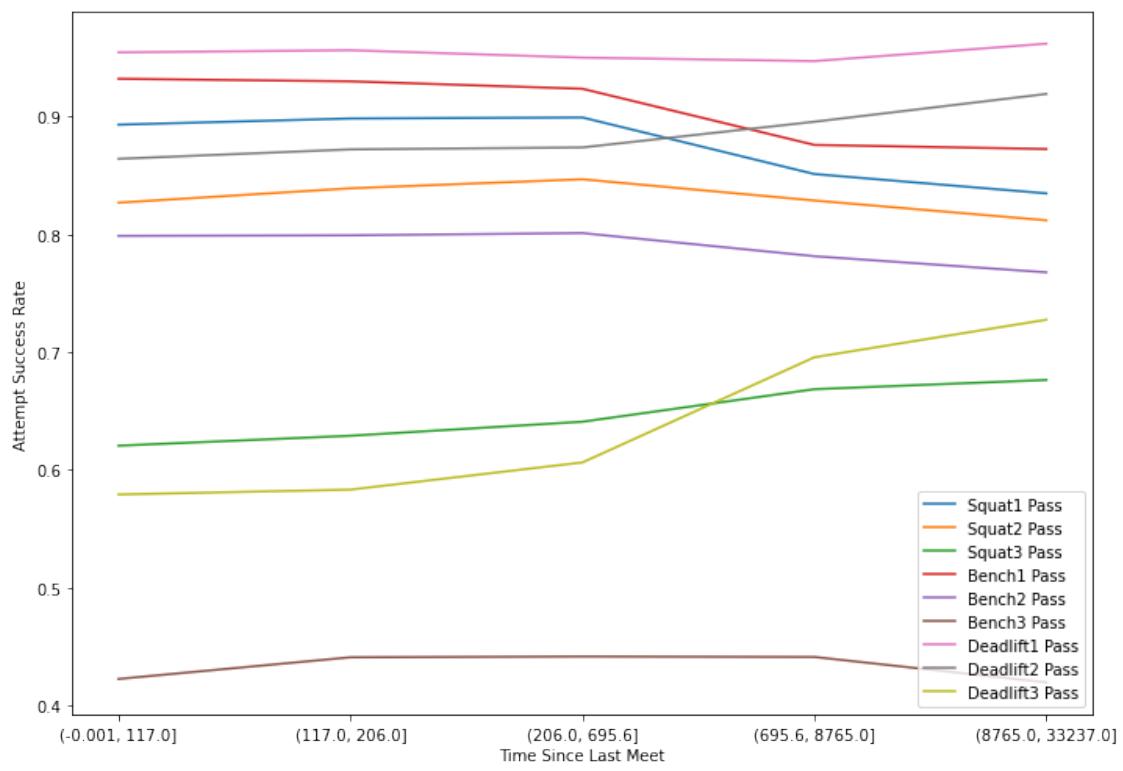


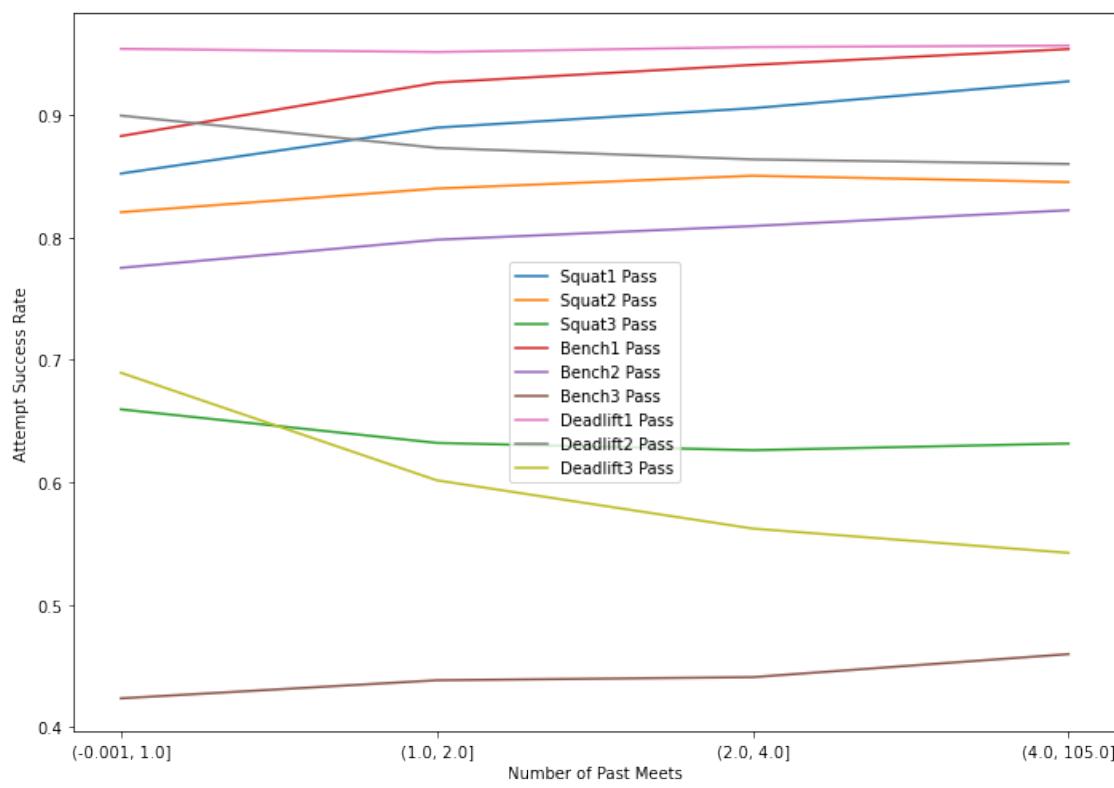
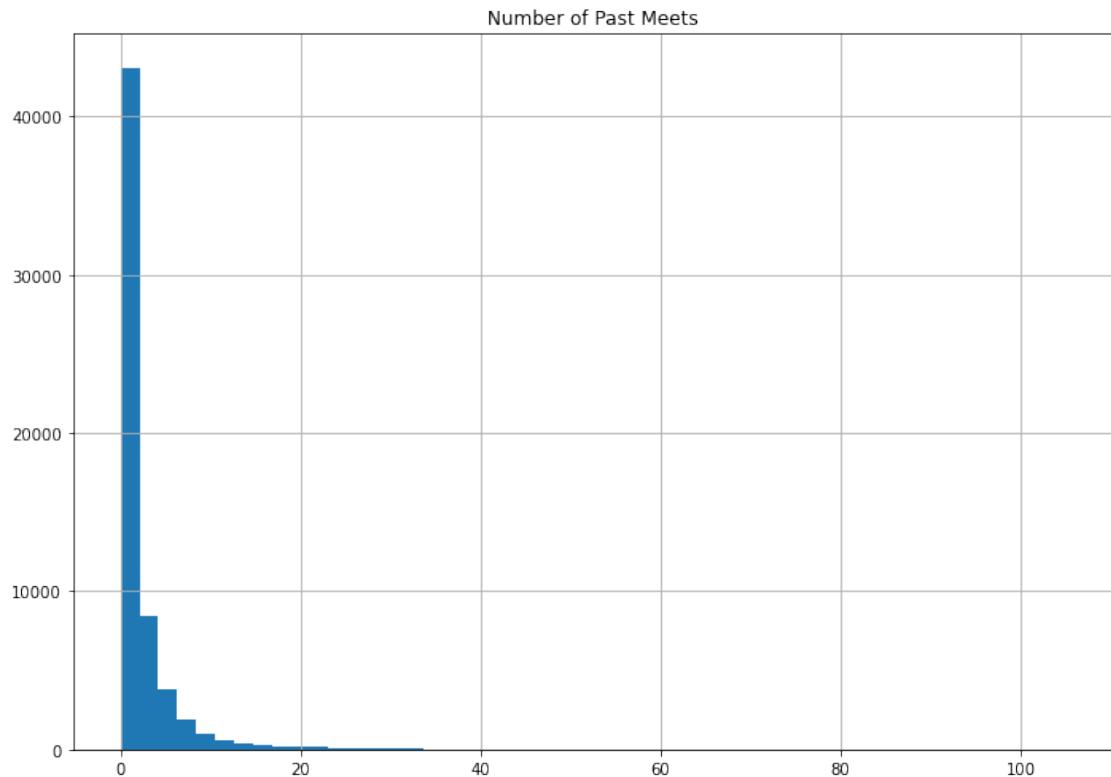




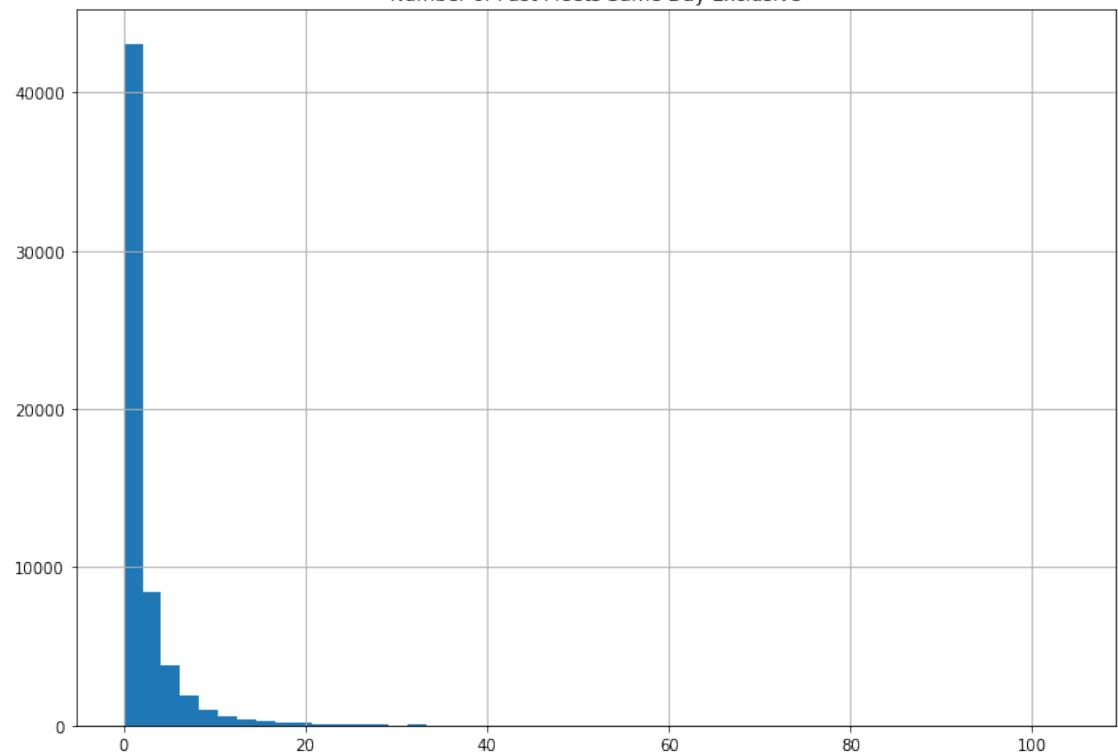


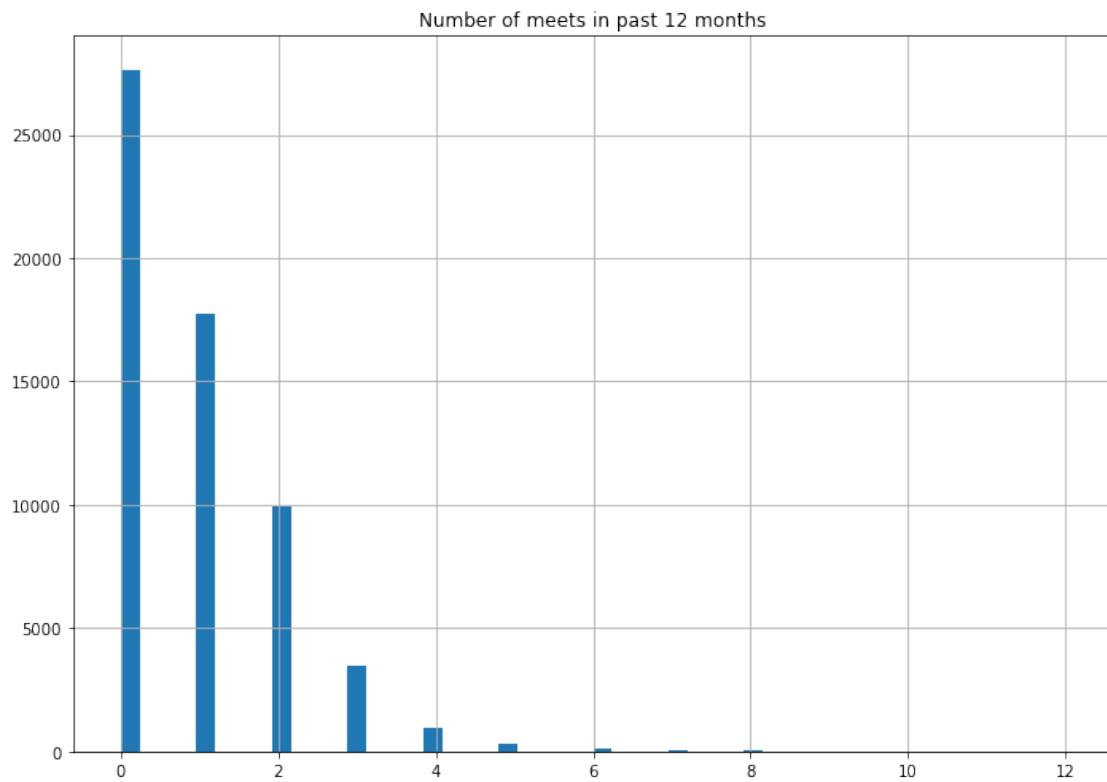
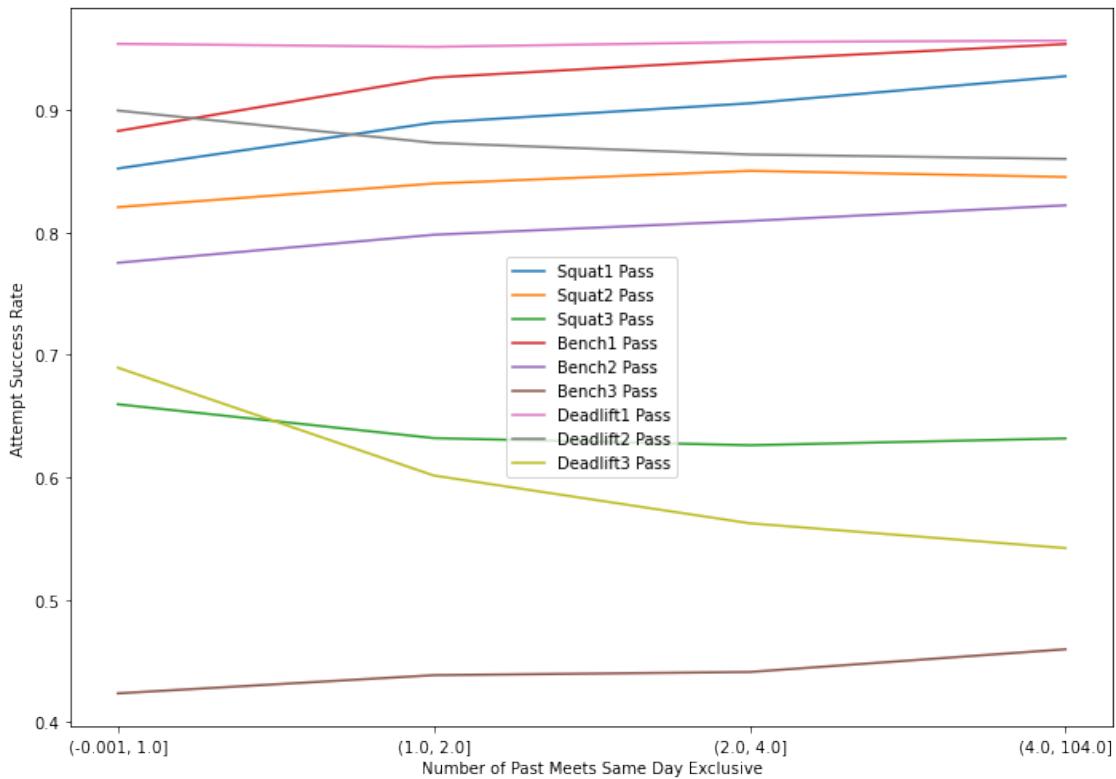


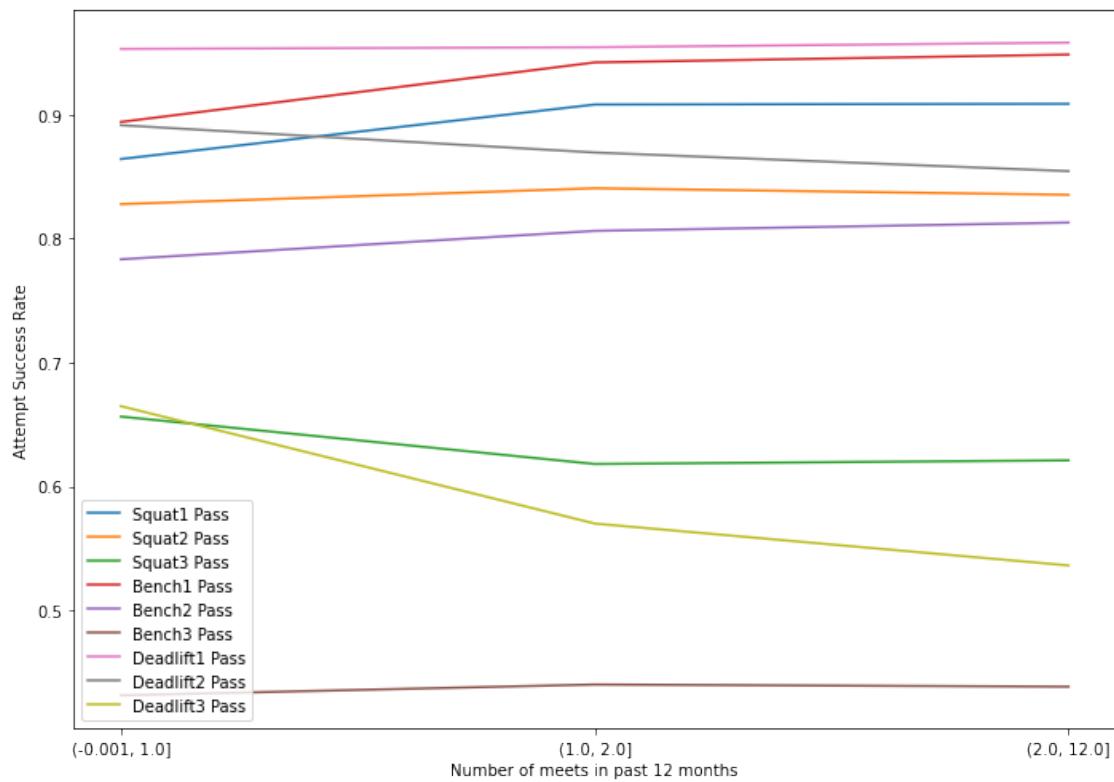


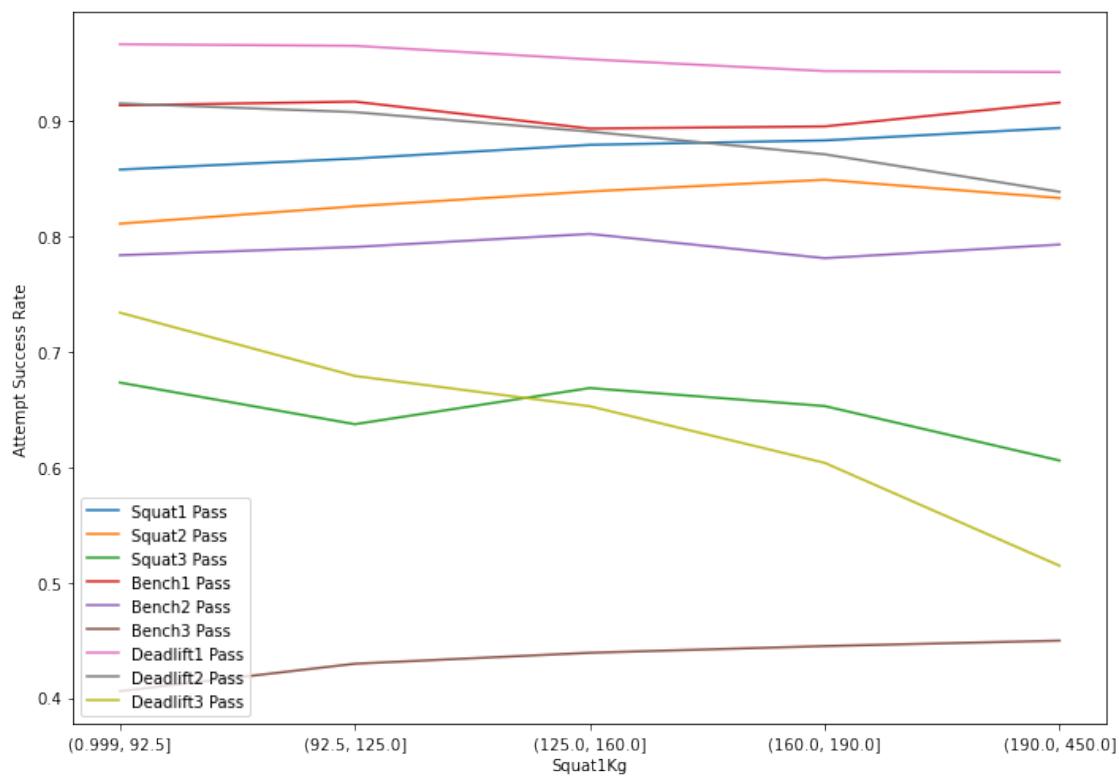
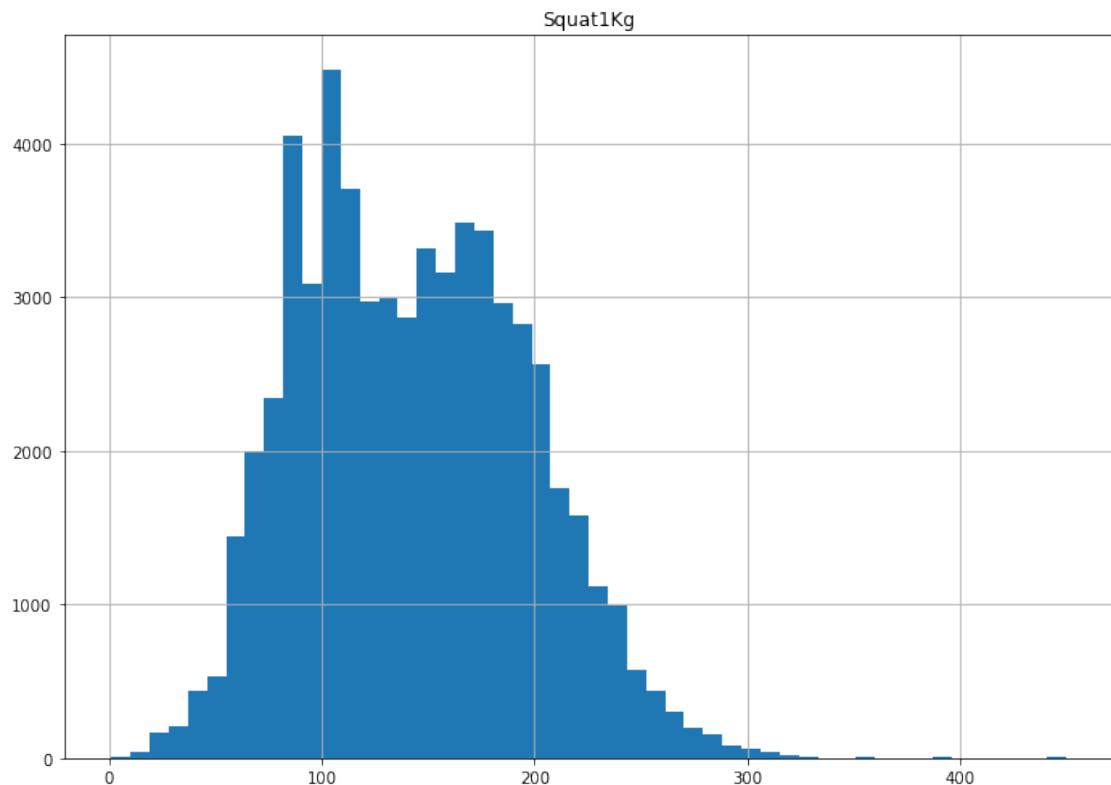


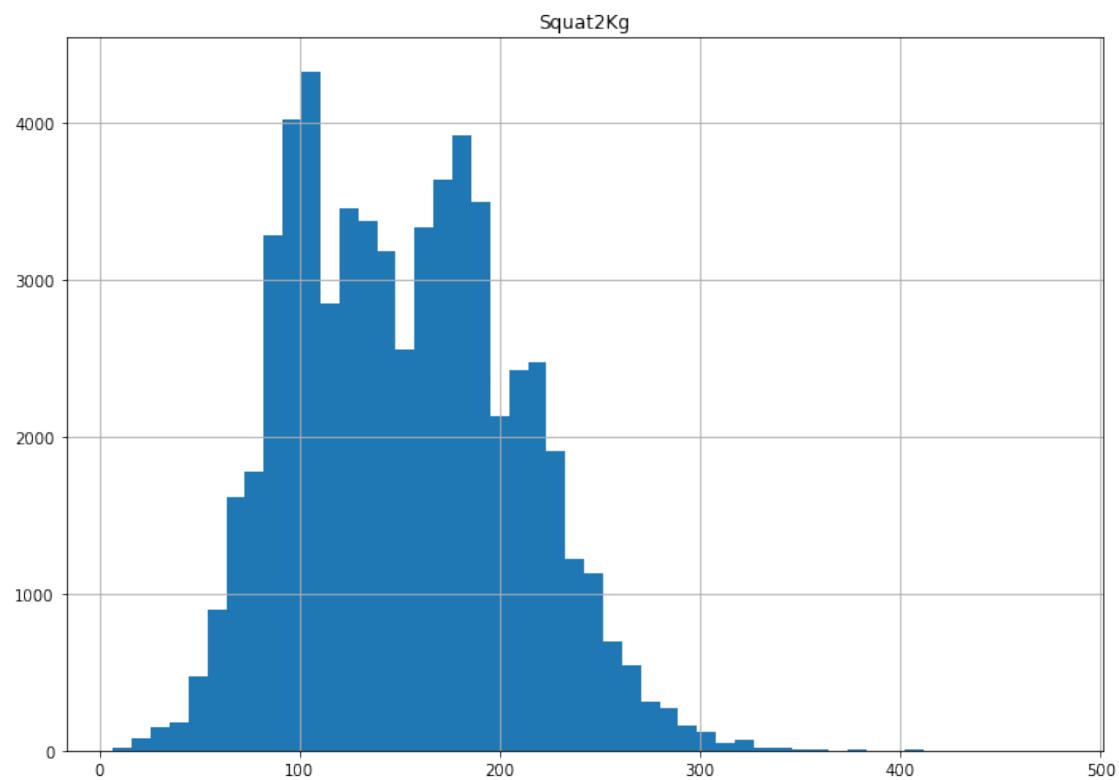
Number of Past Meets Same Day Exclusive

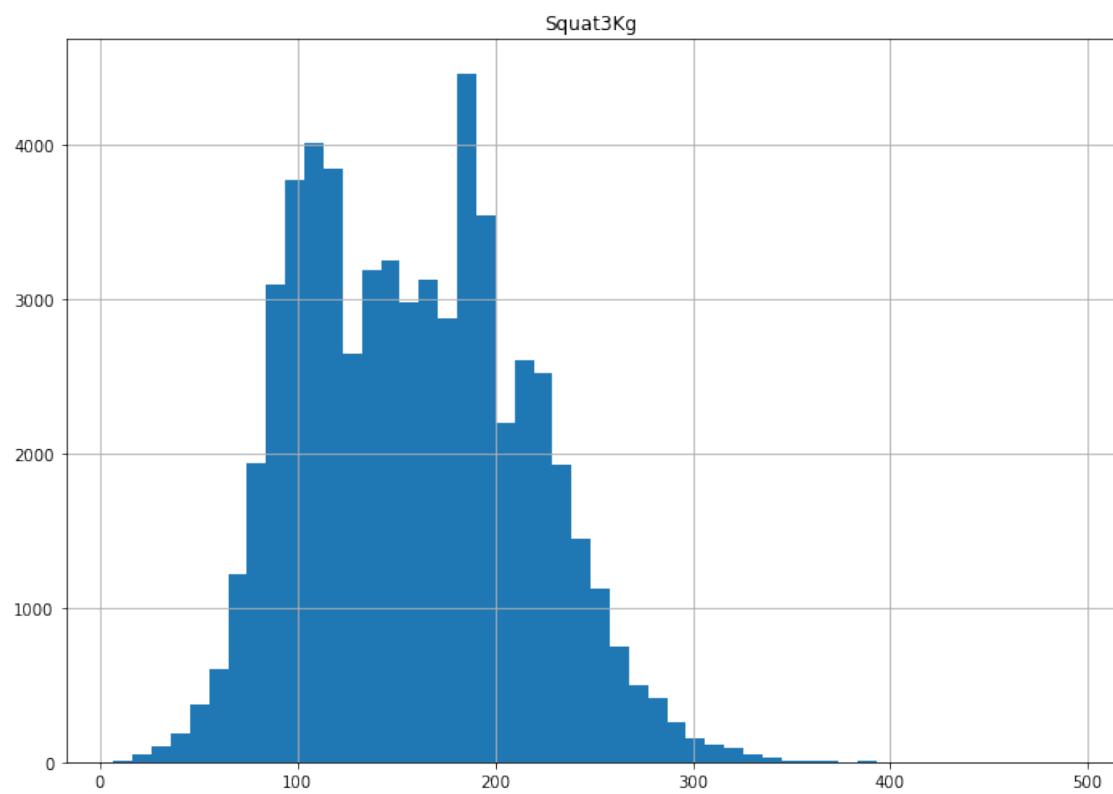
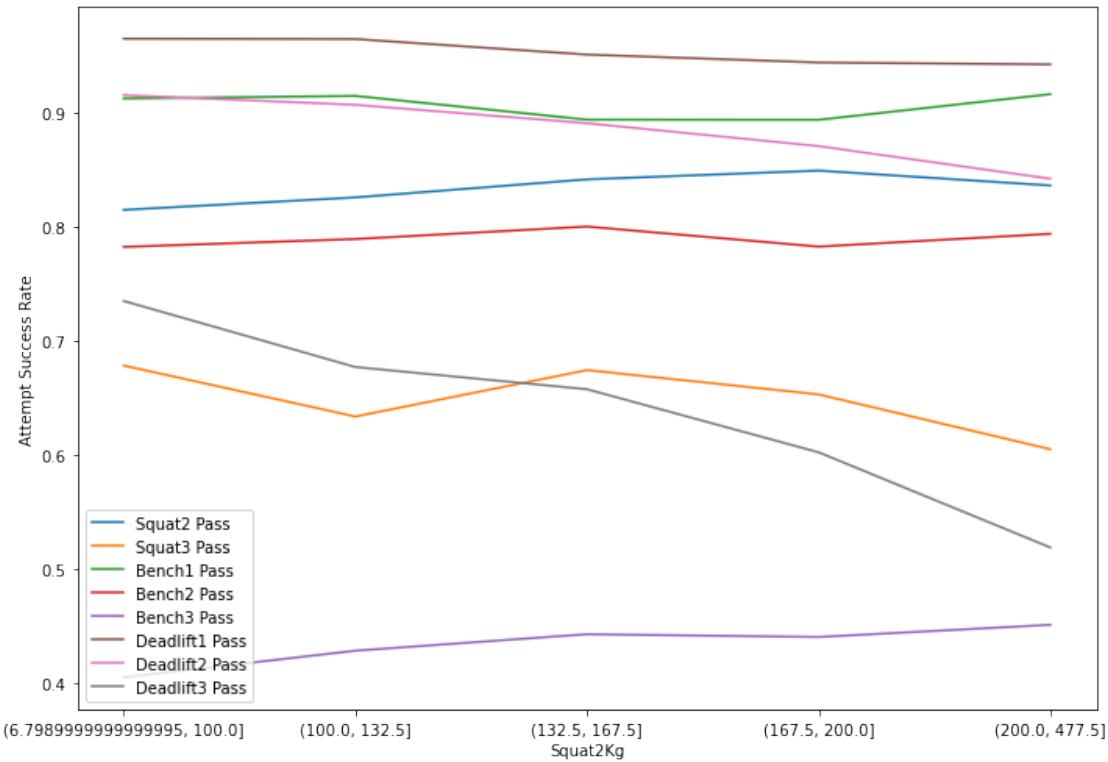


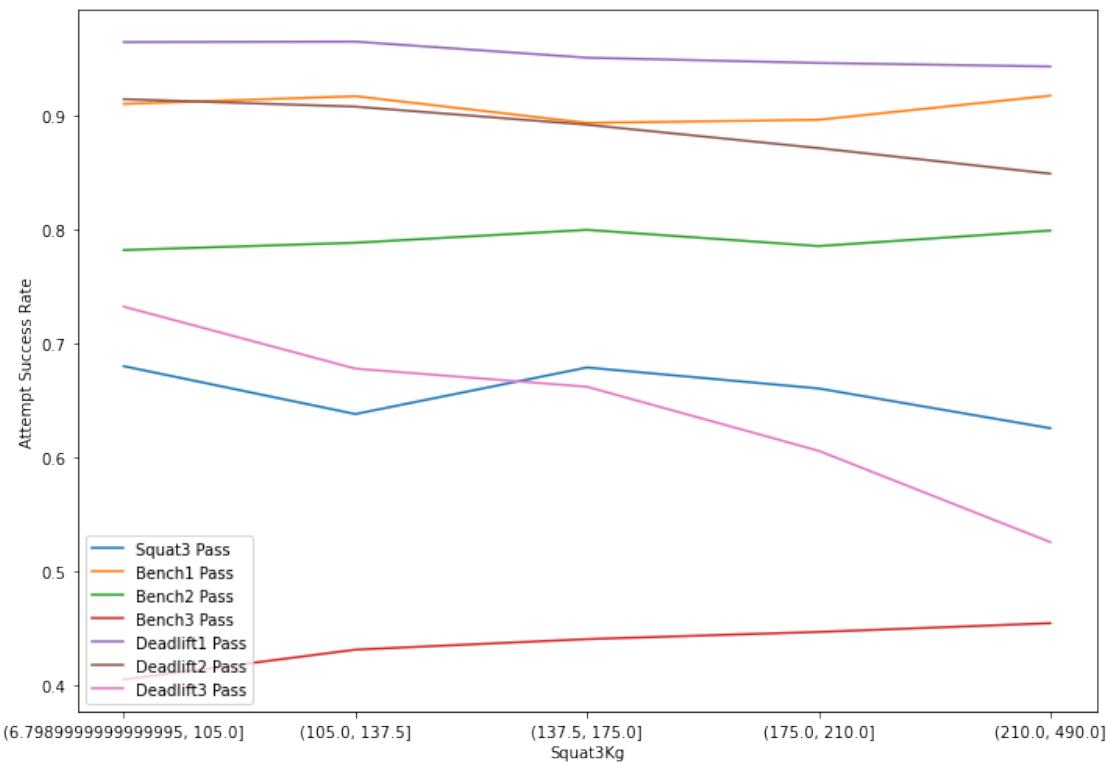


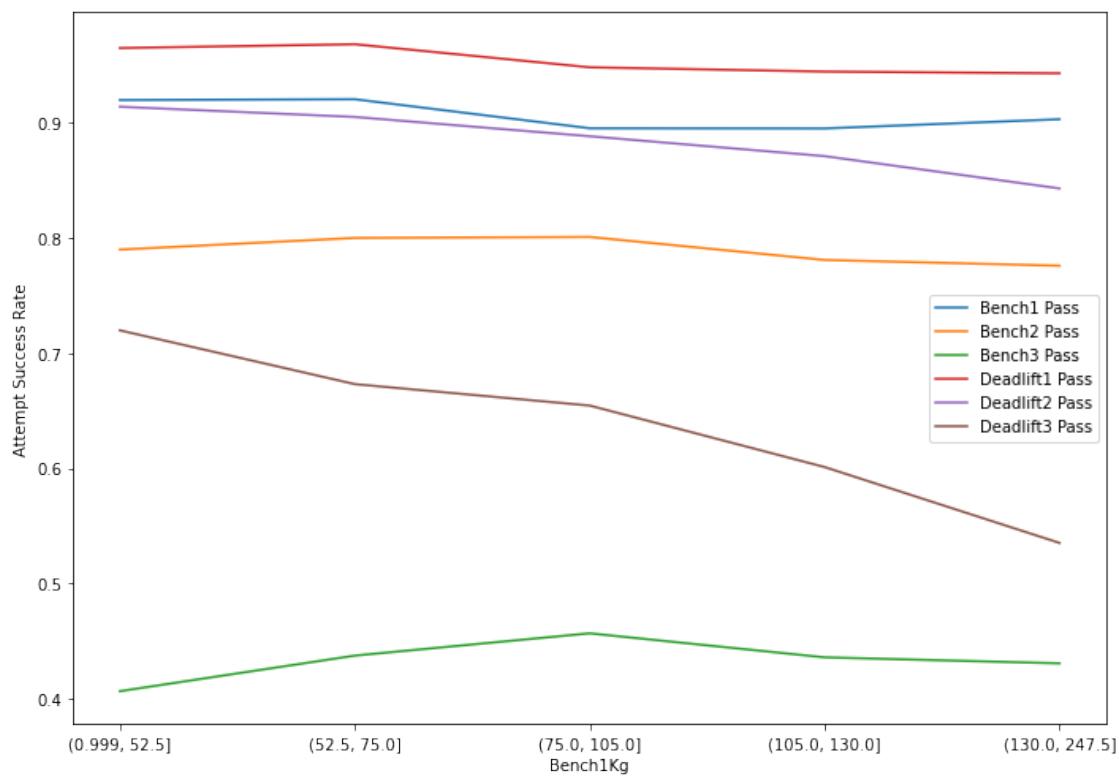
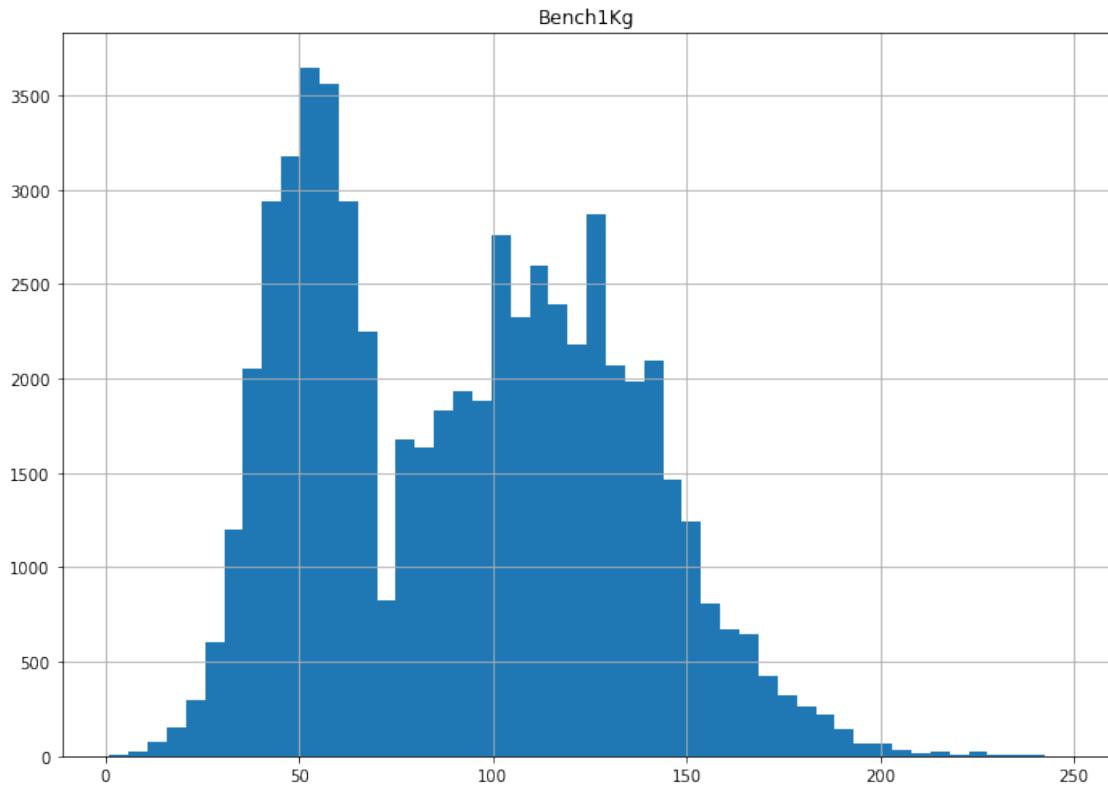


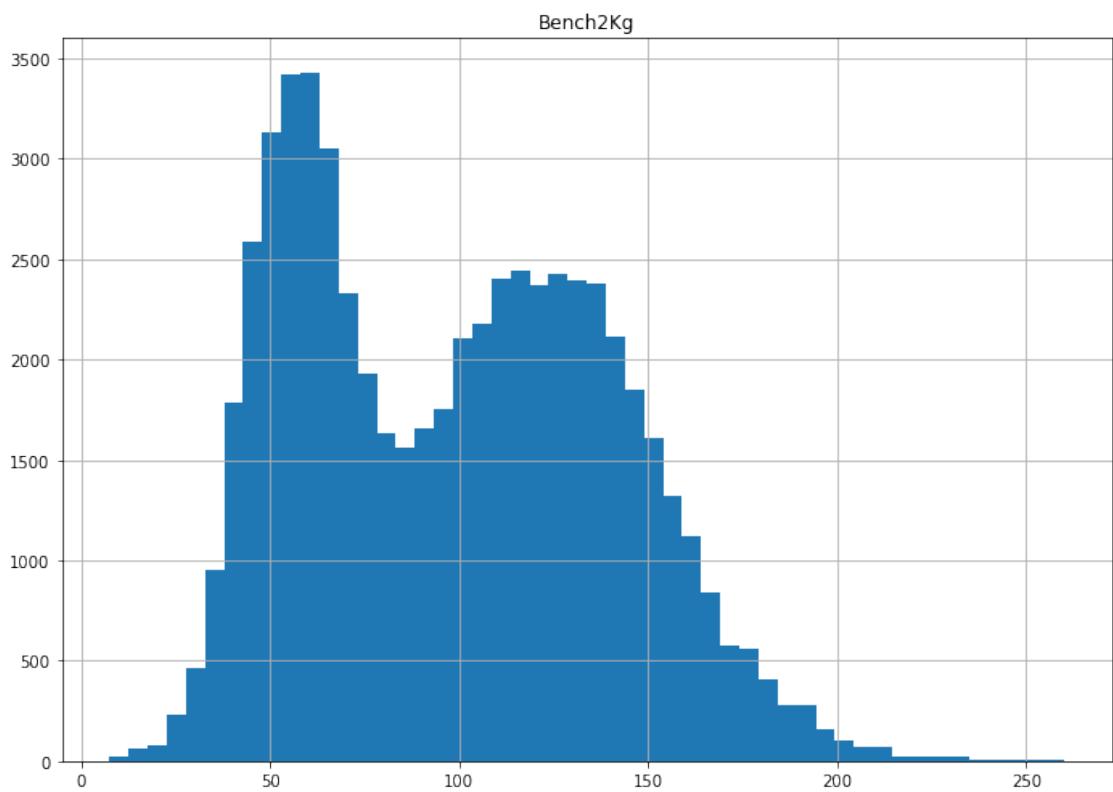


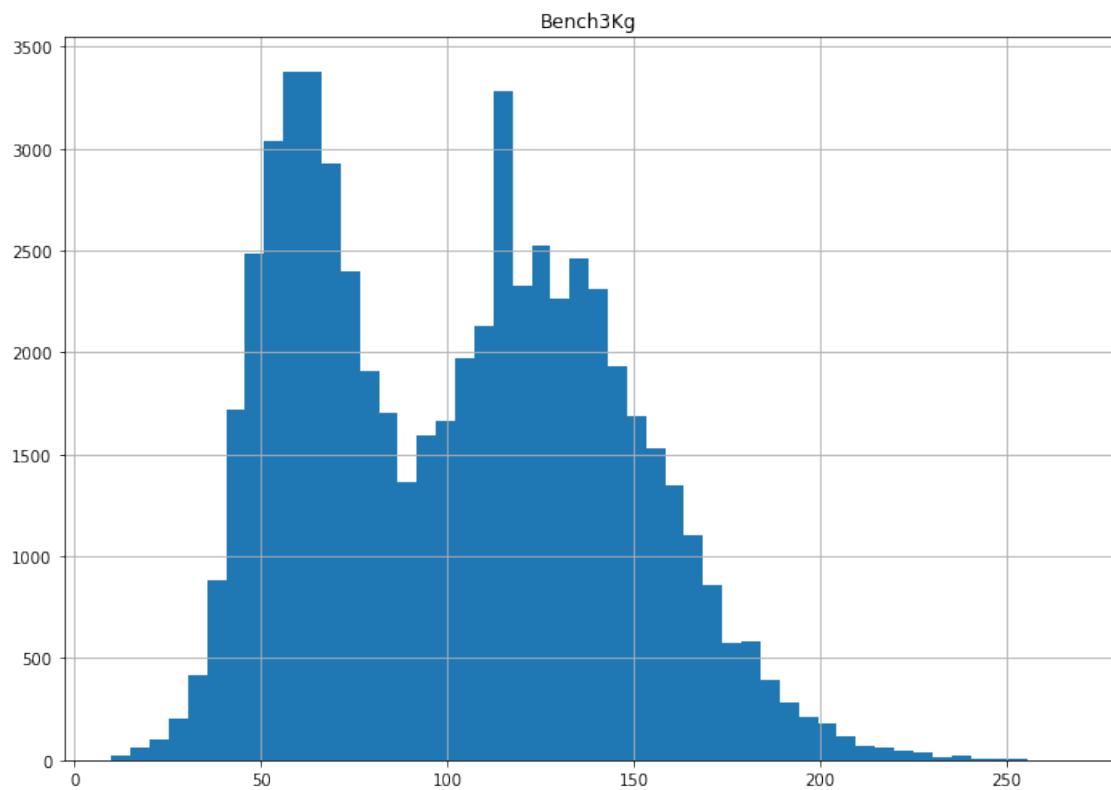
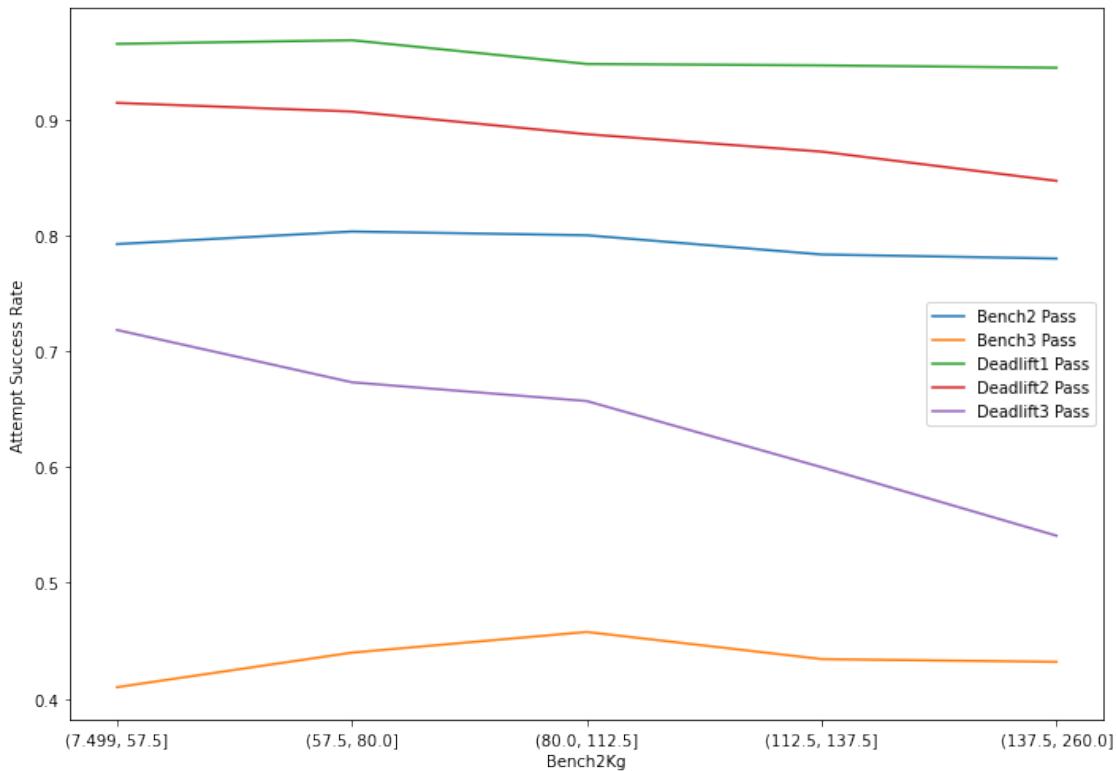


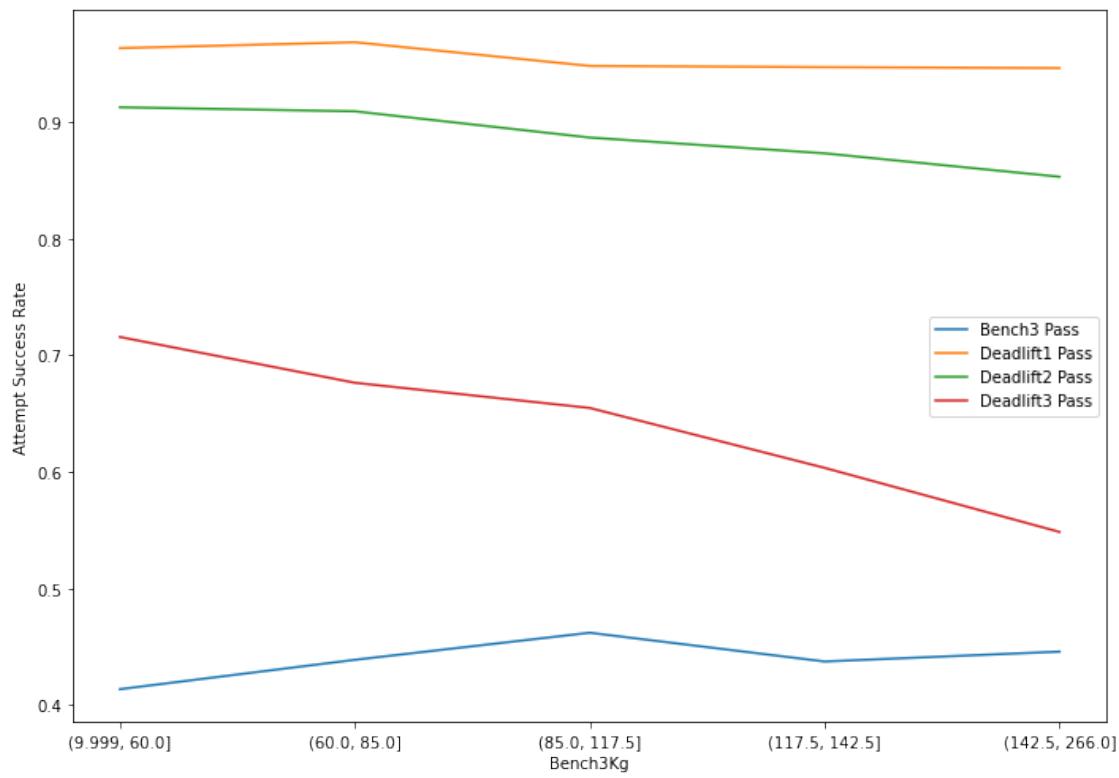


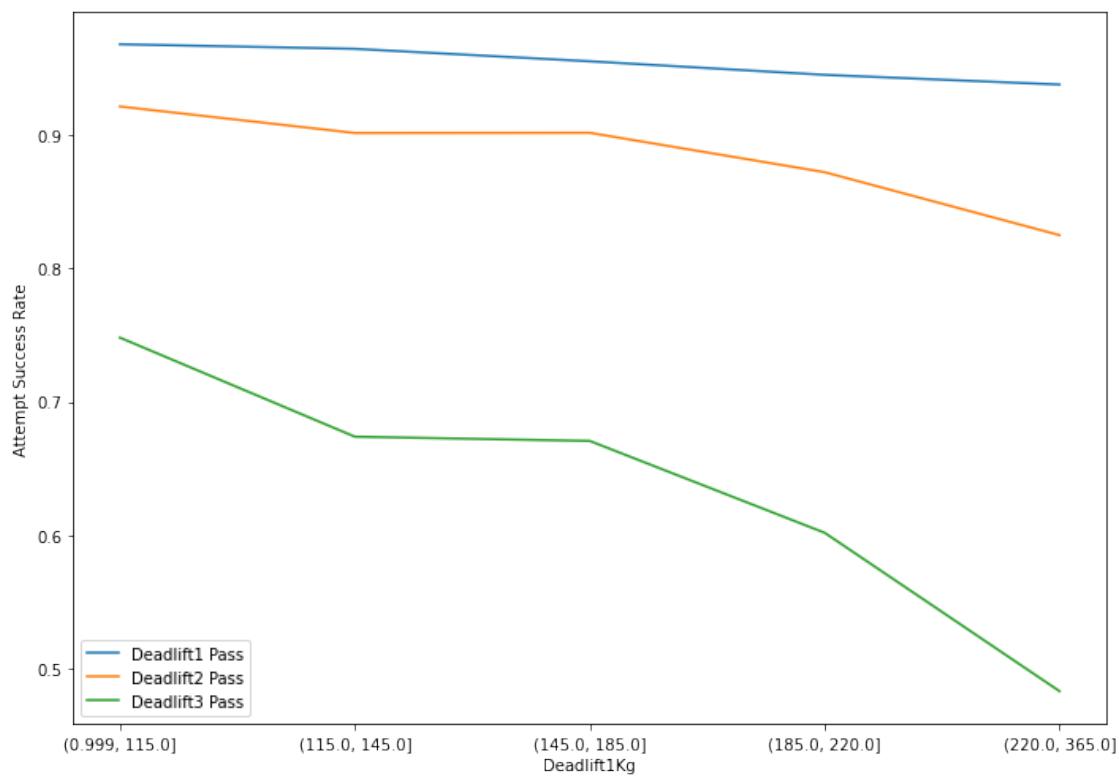
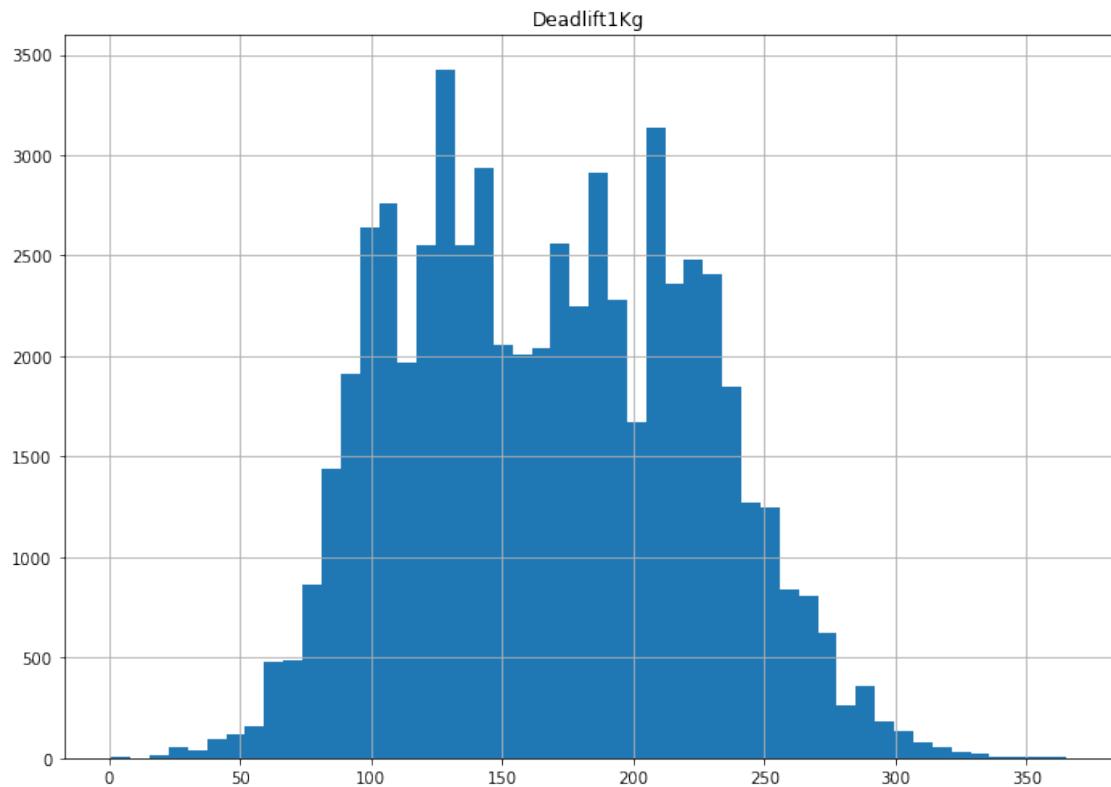


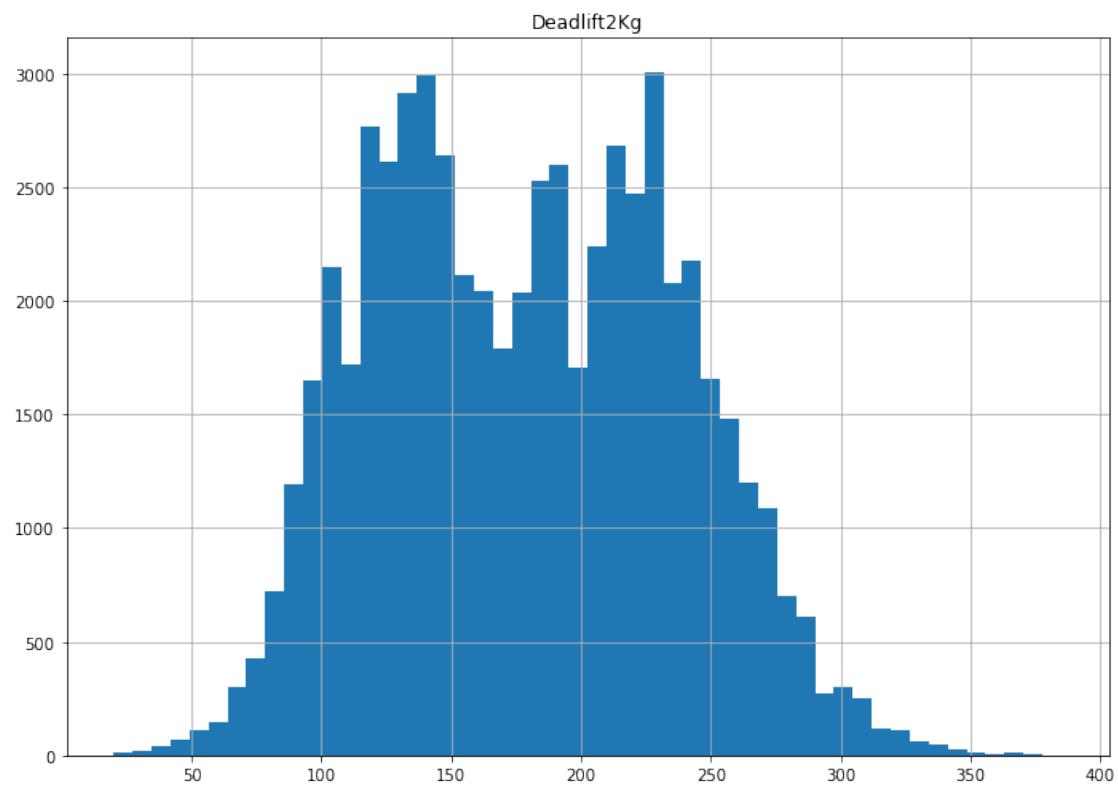


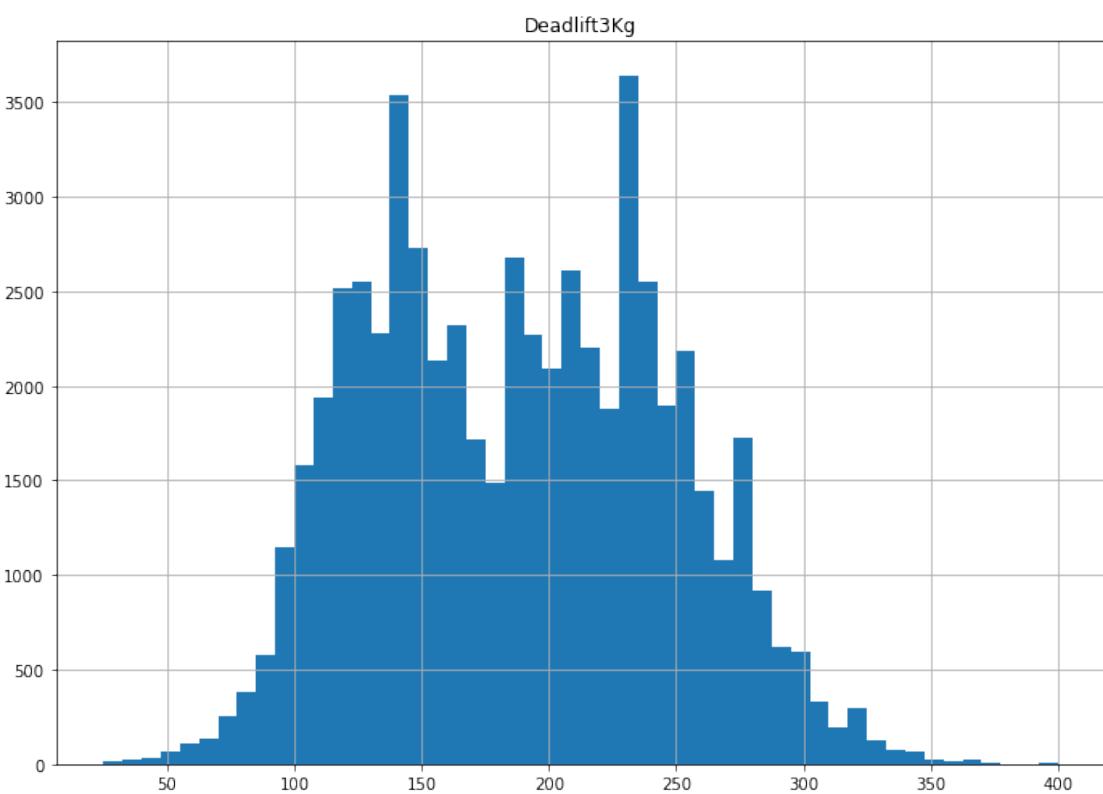
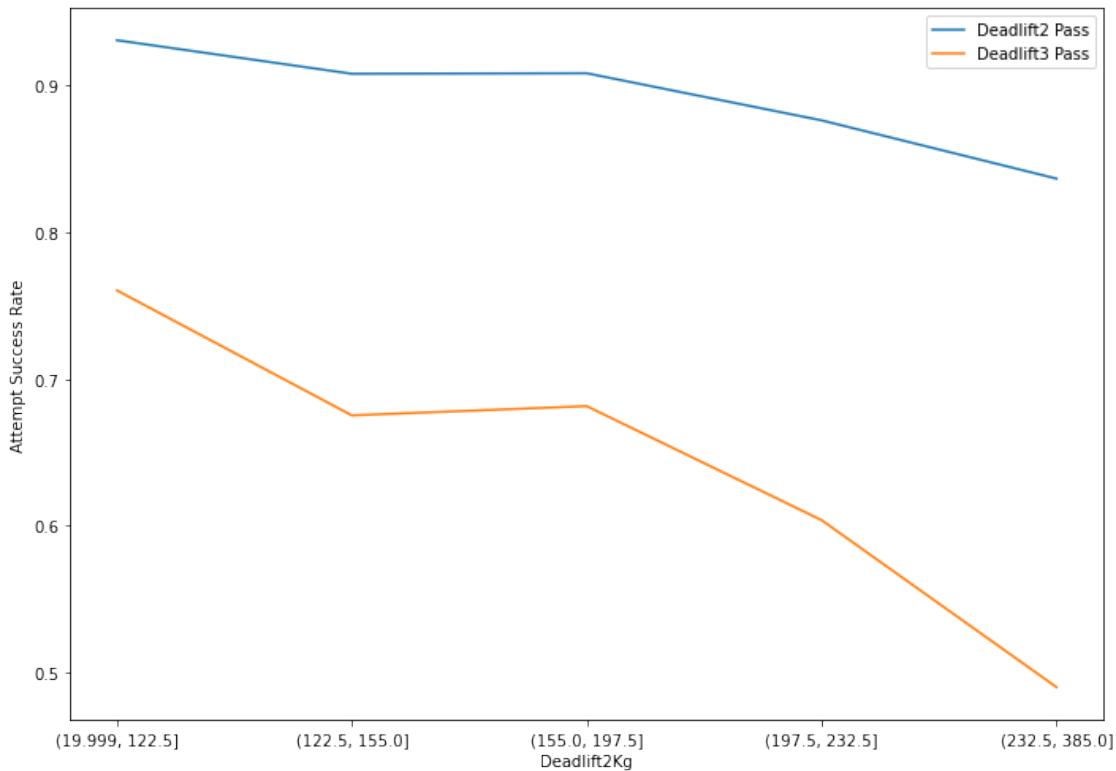


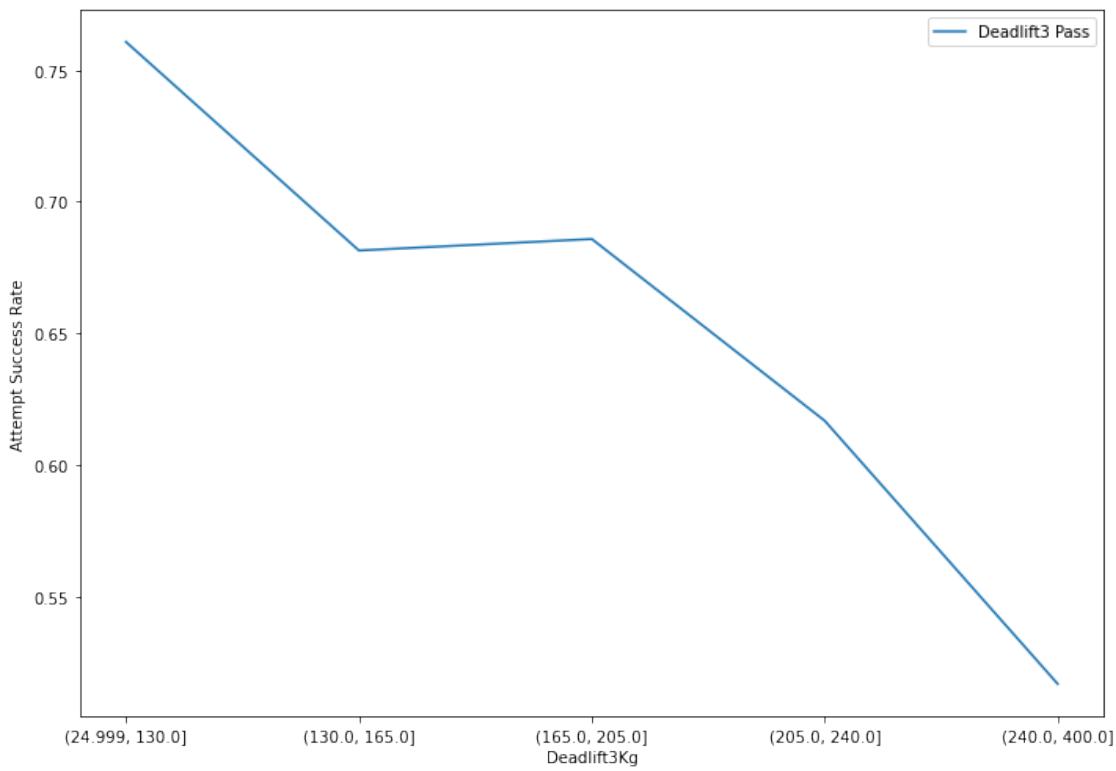


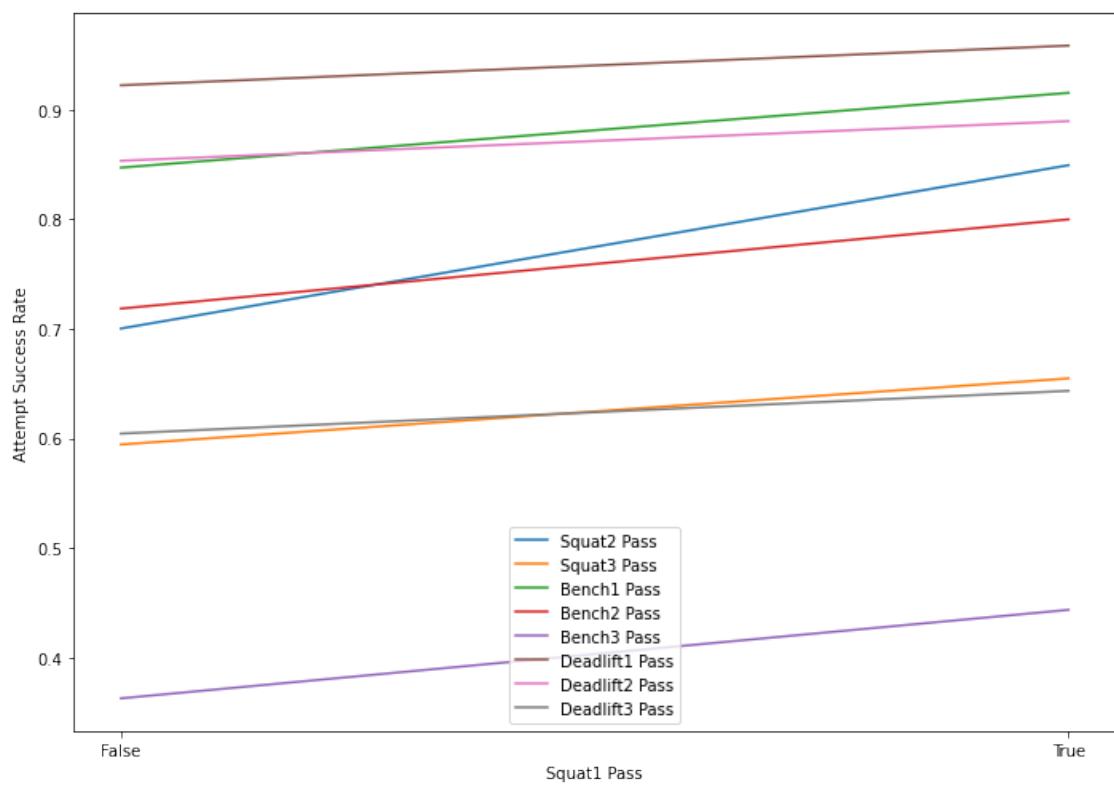
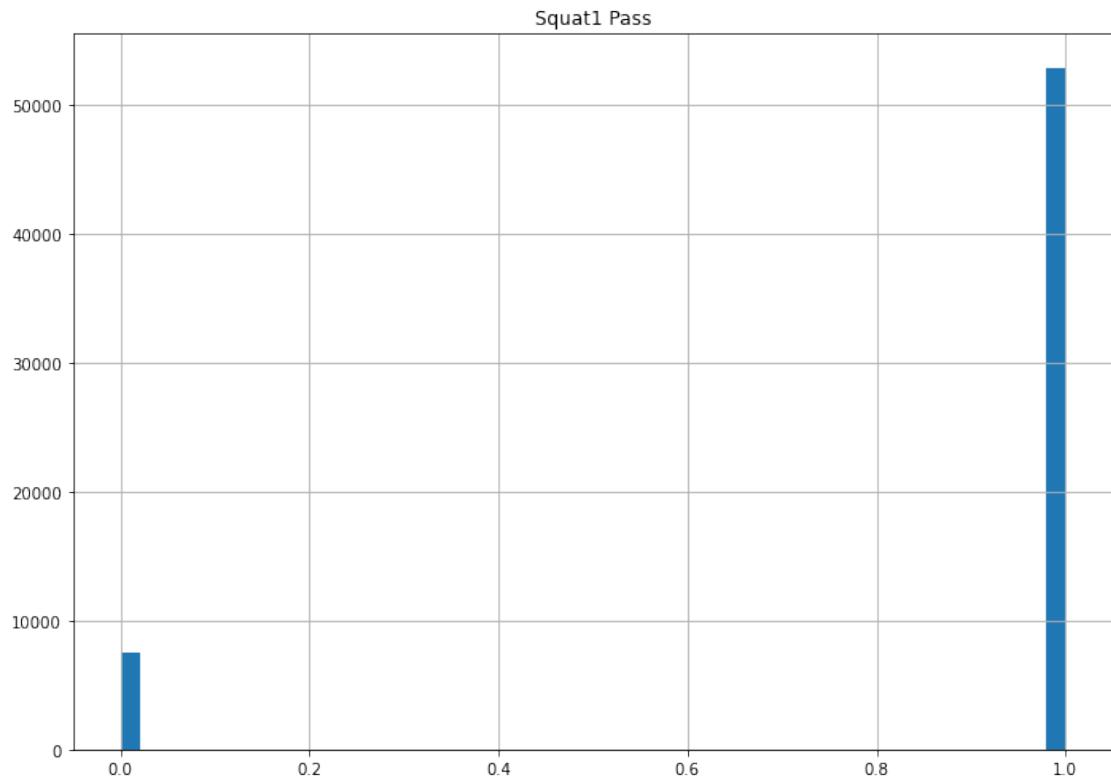


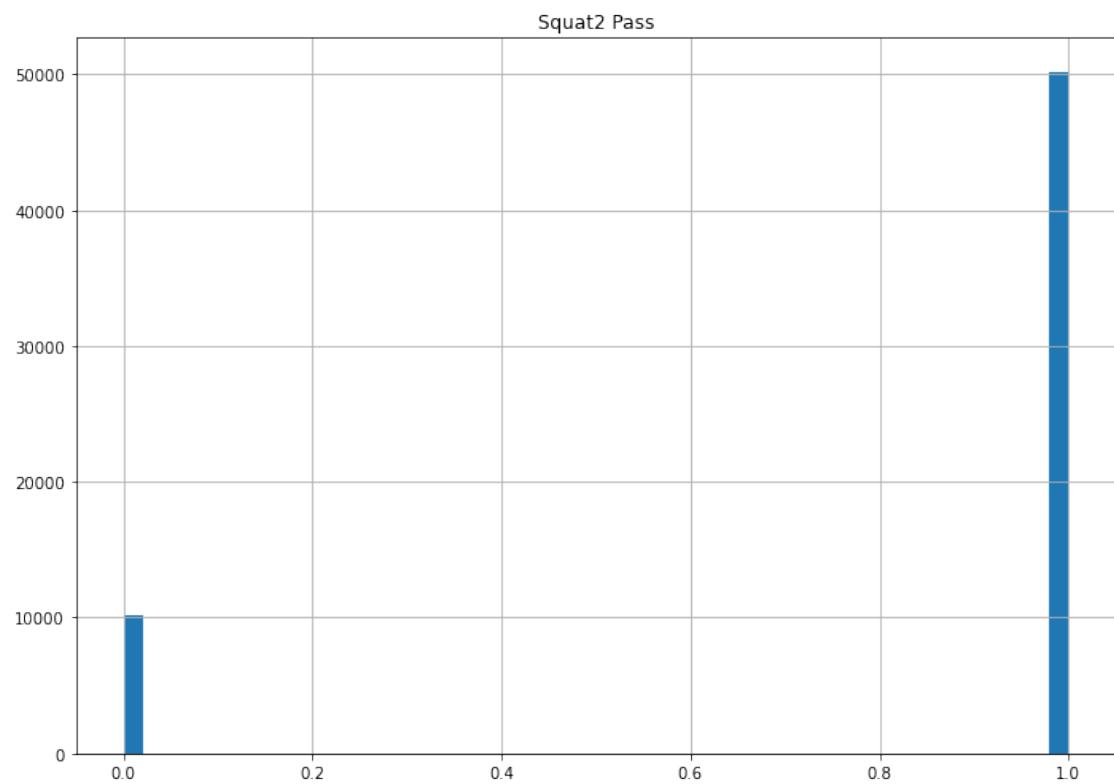


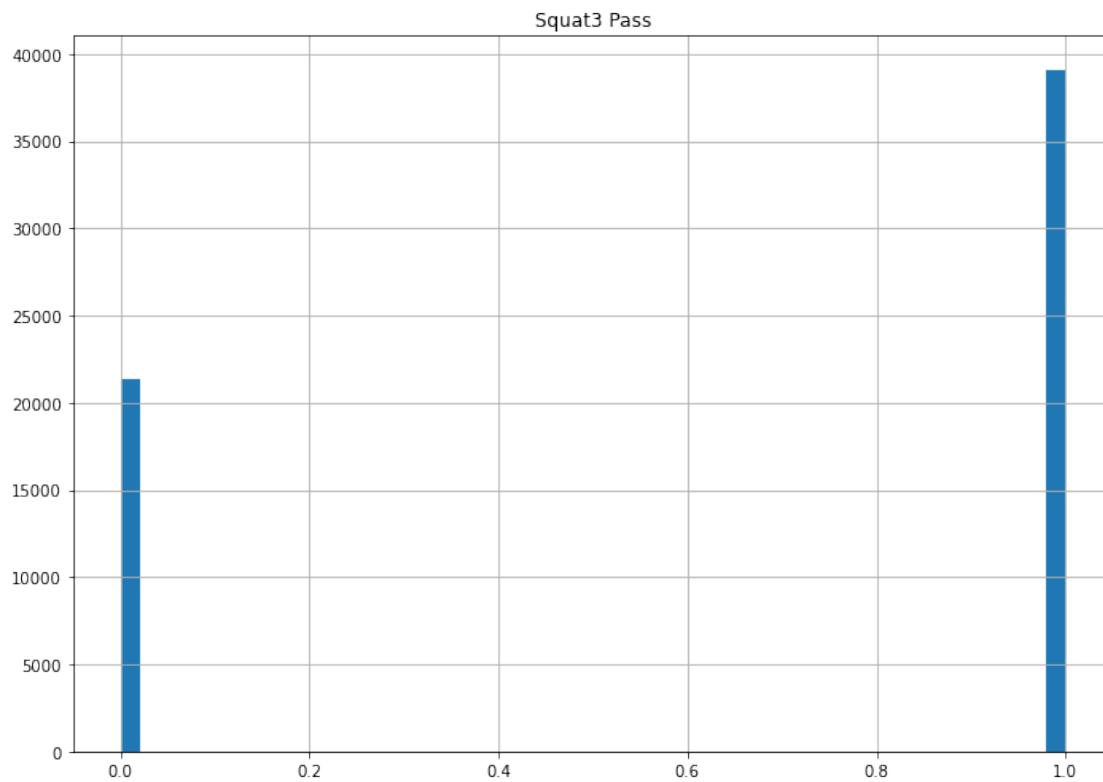
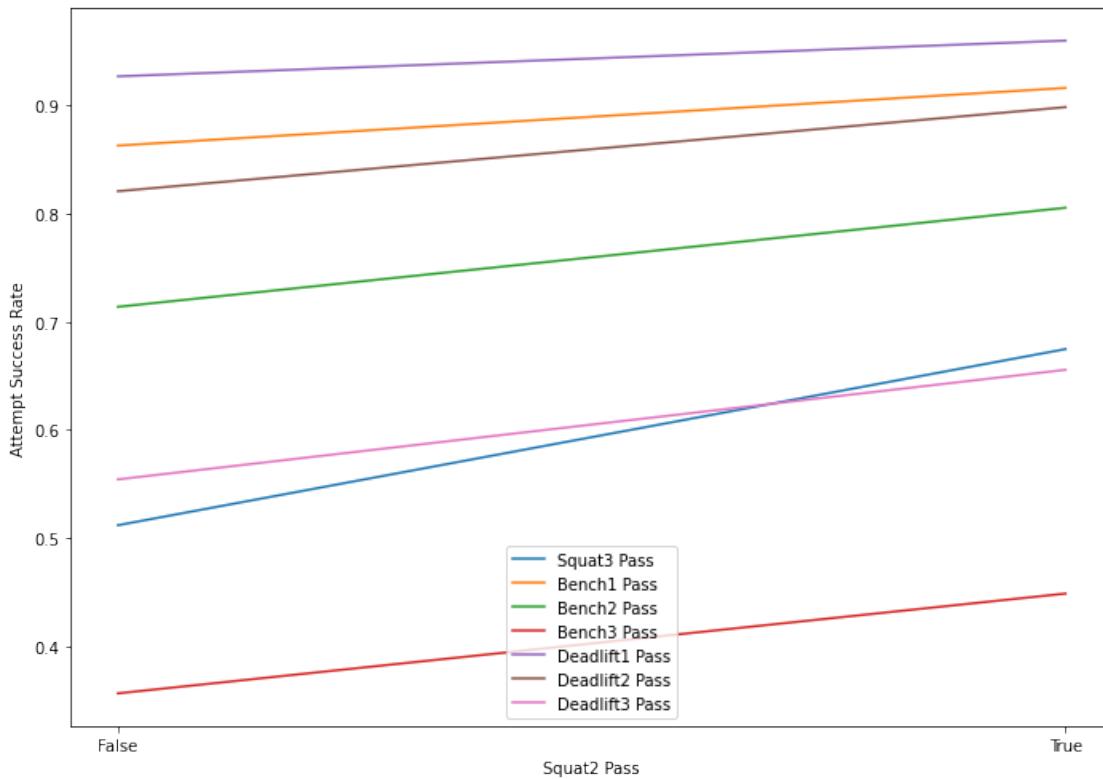


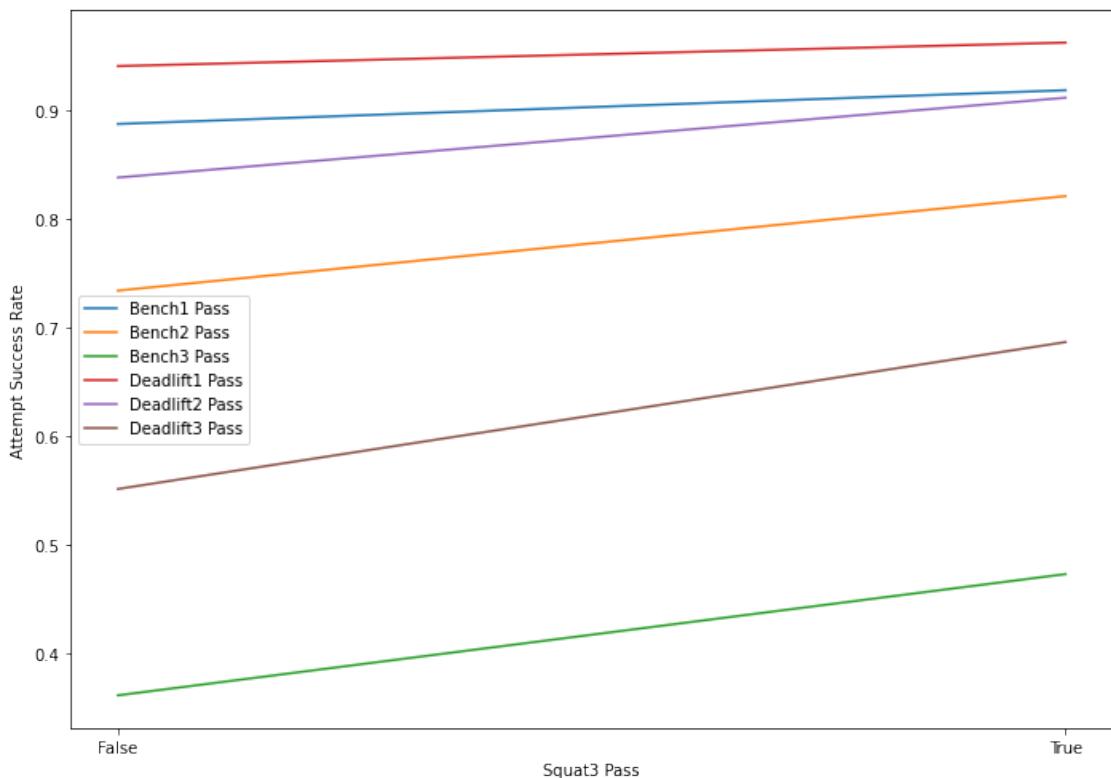


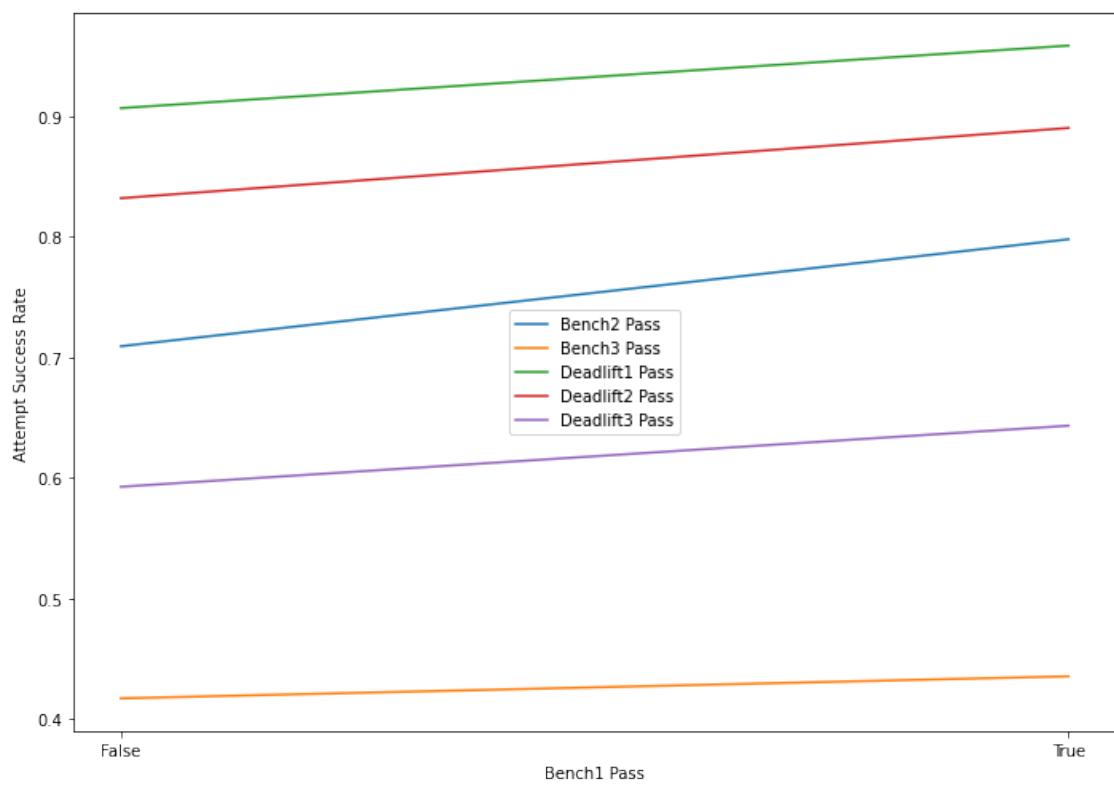
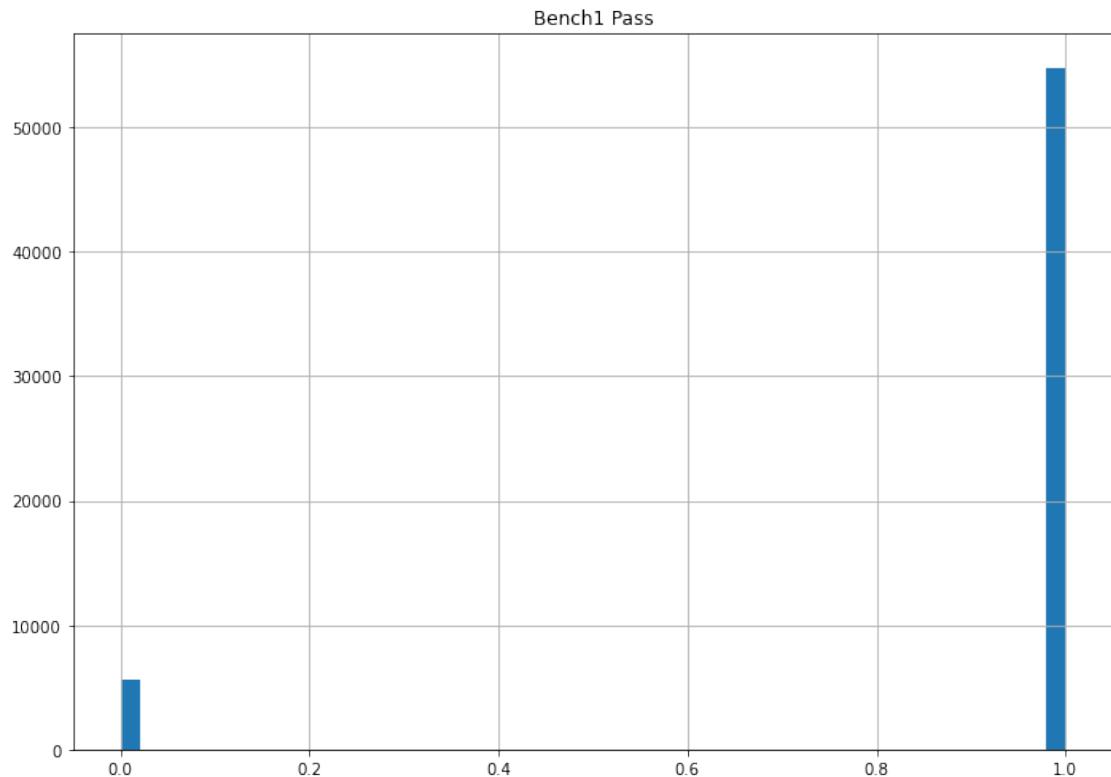


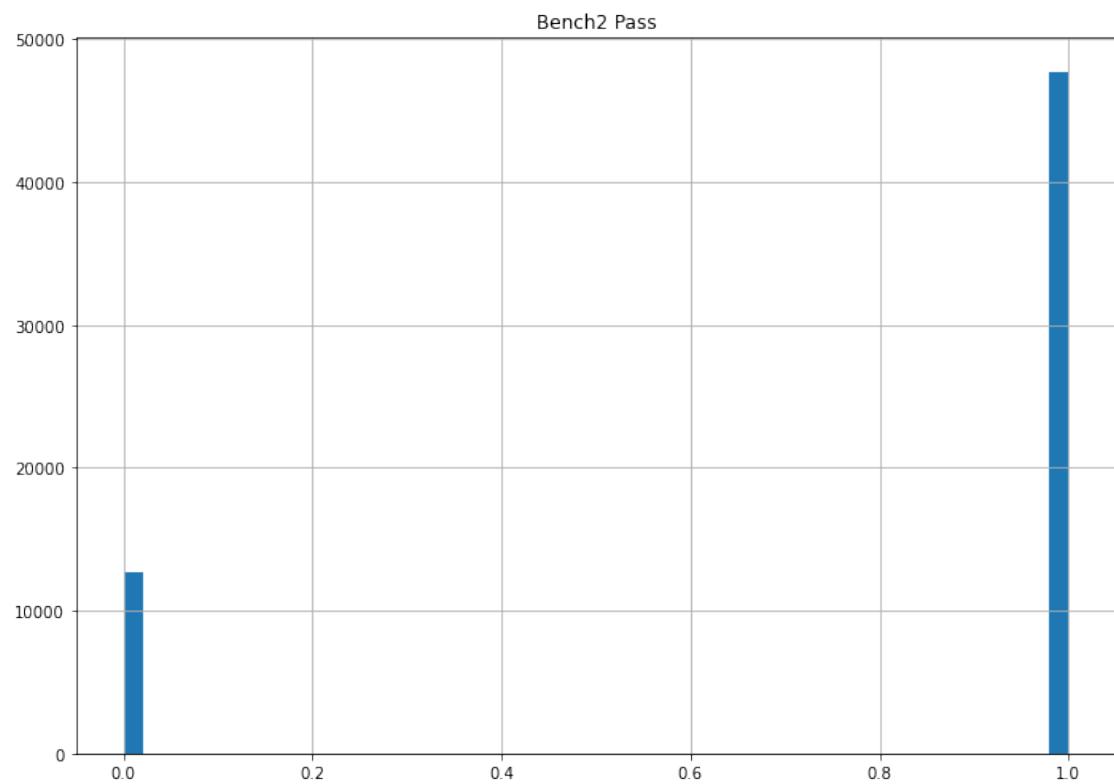


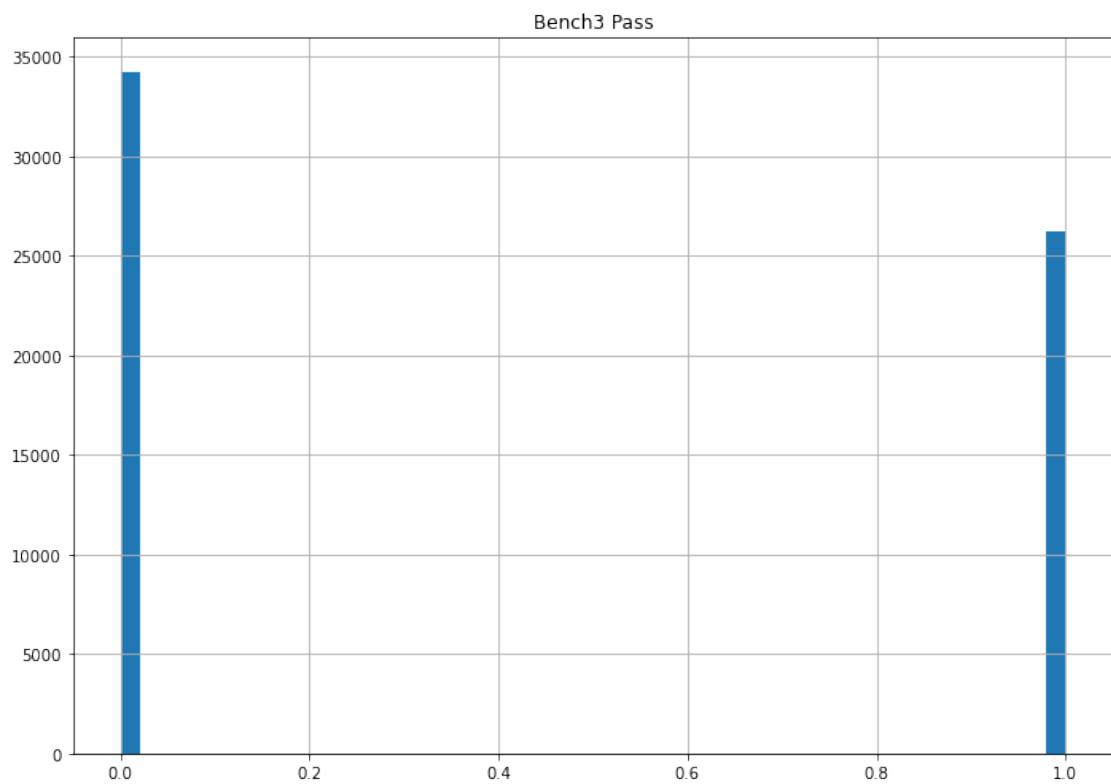
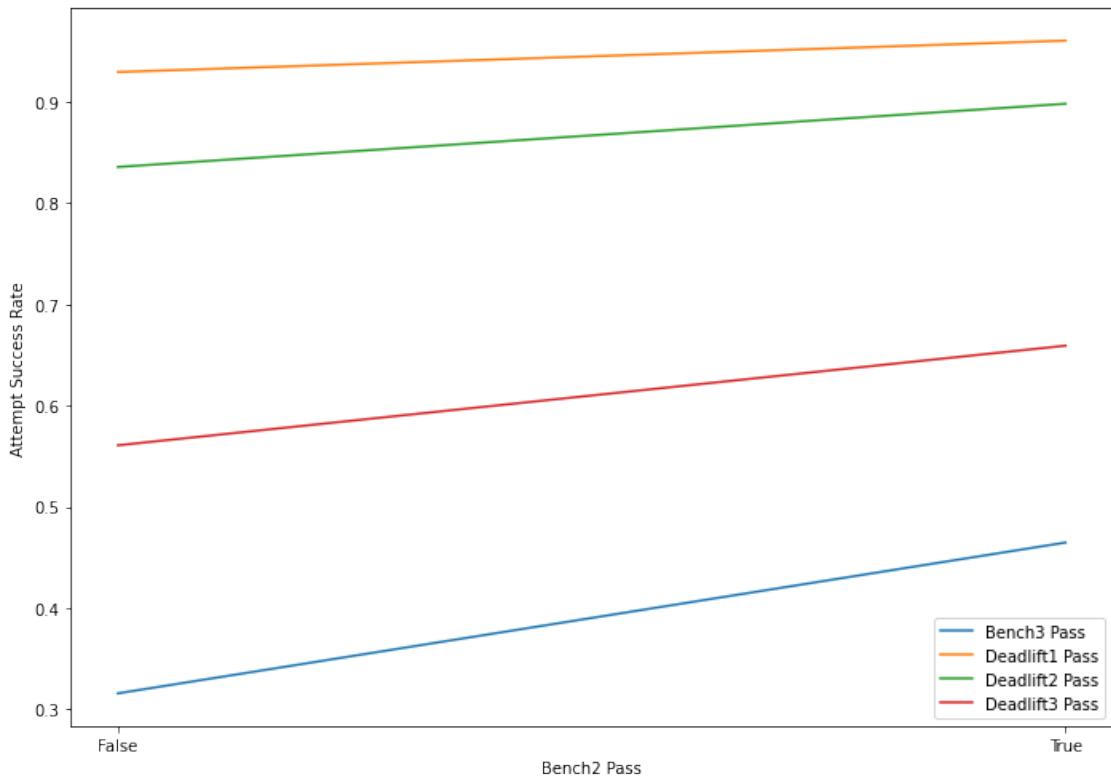


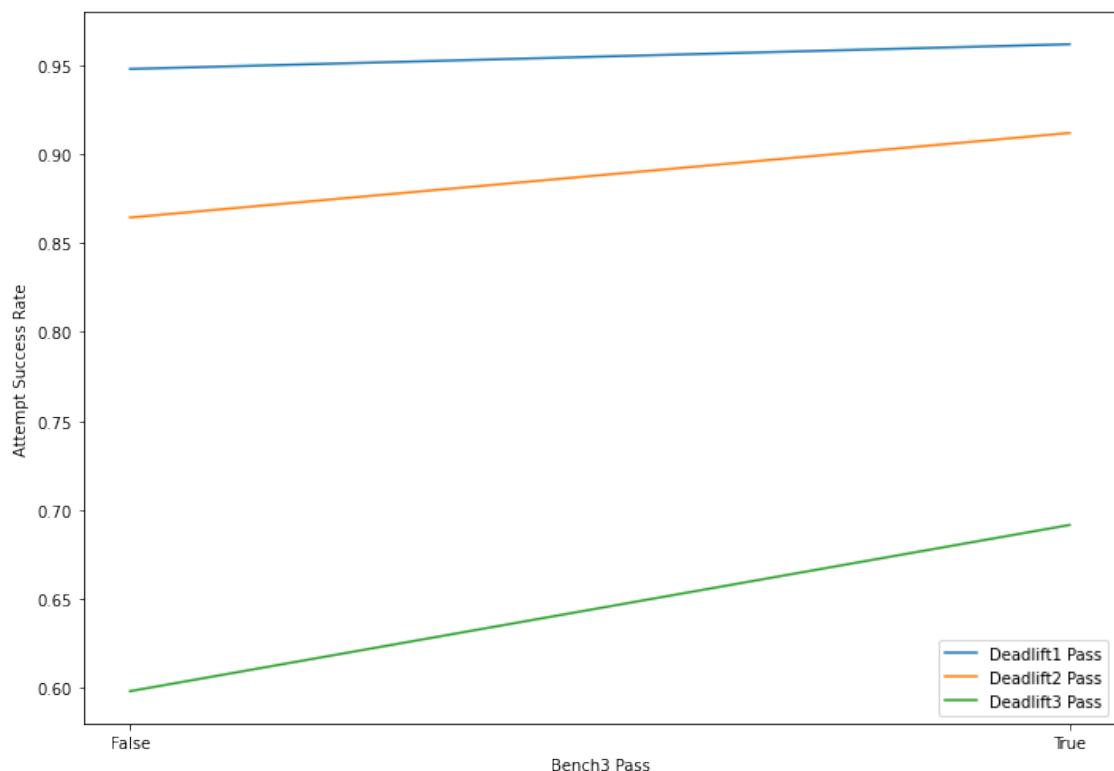


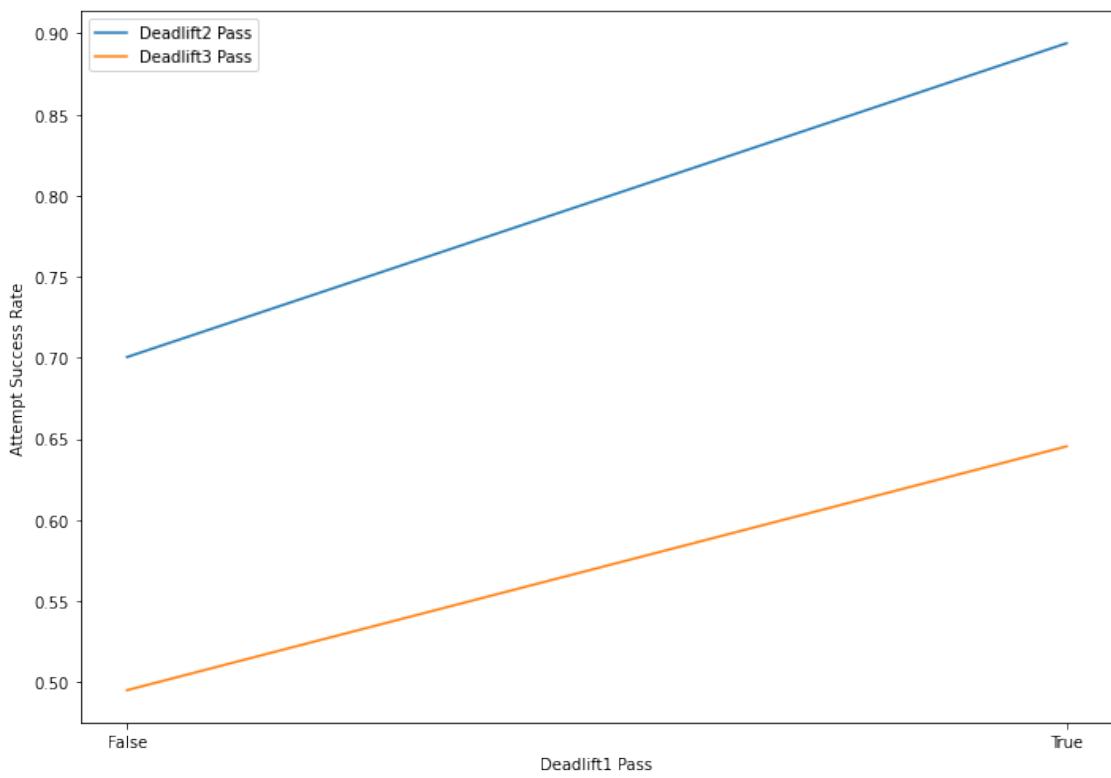
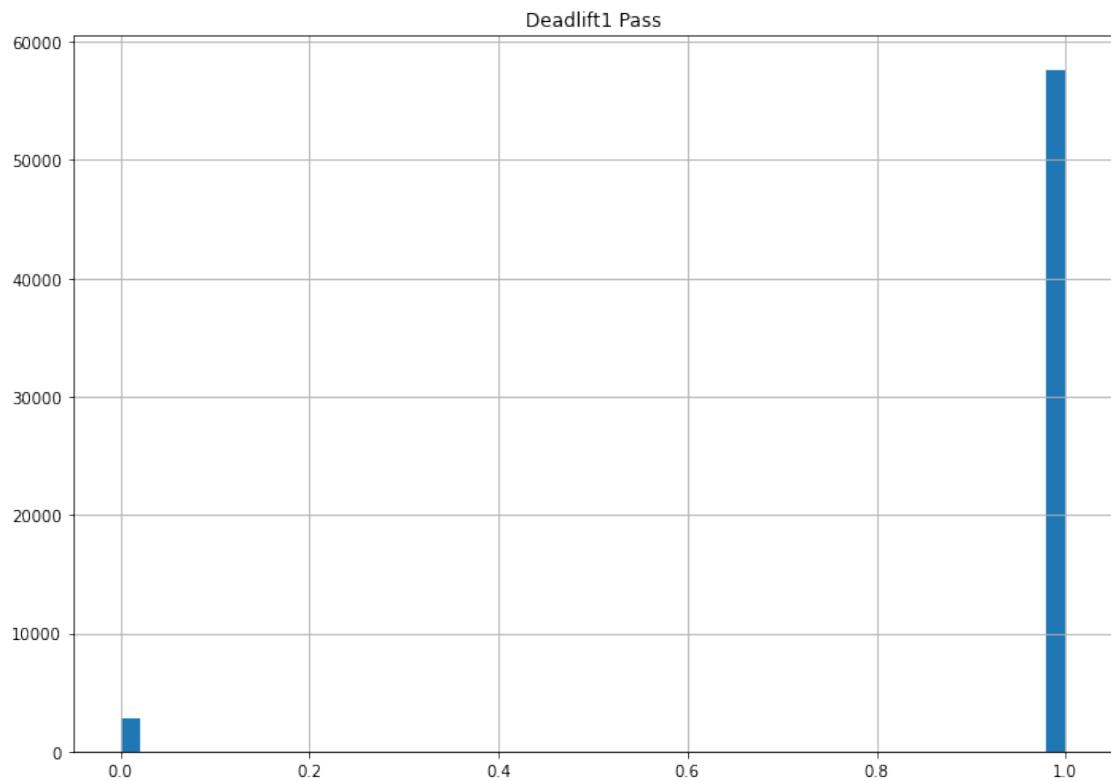


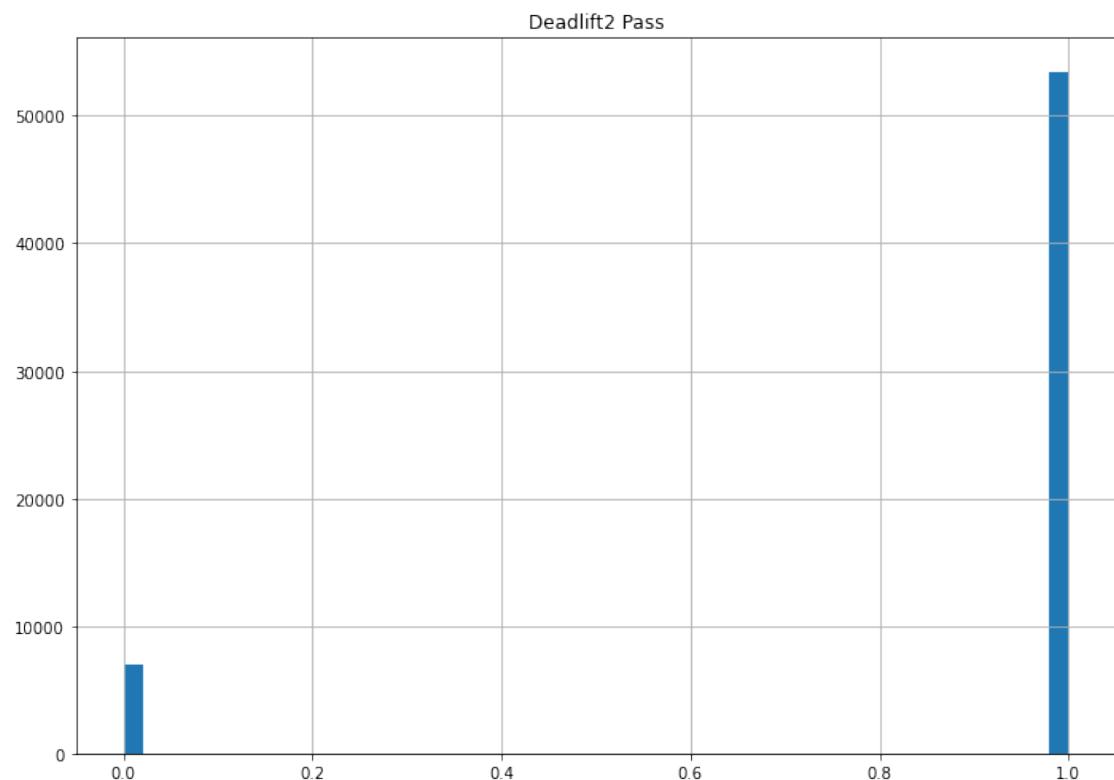


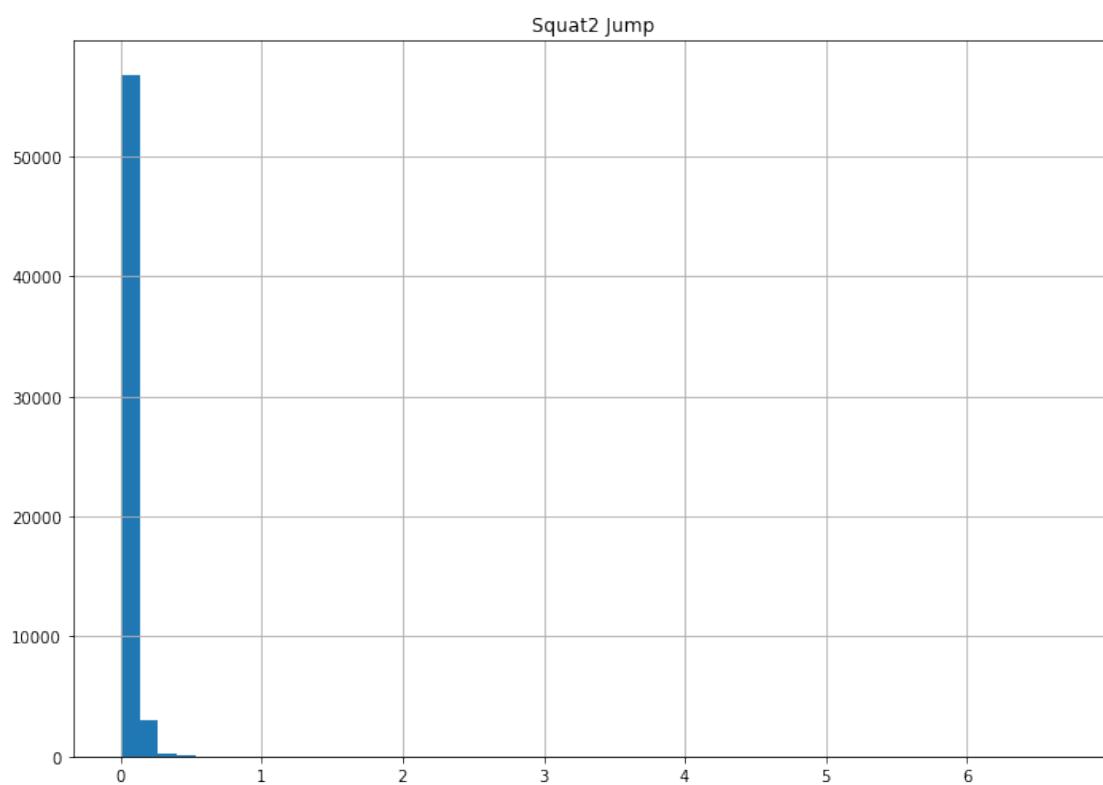
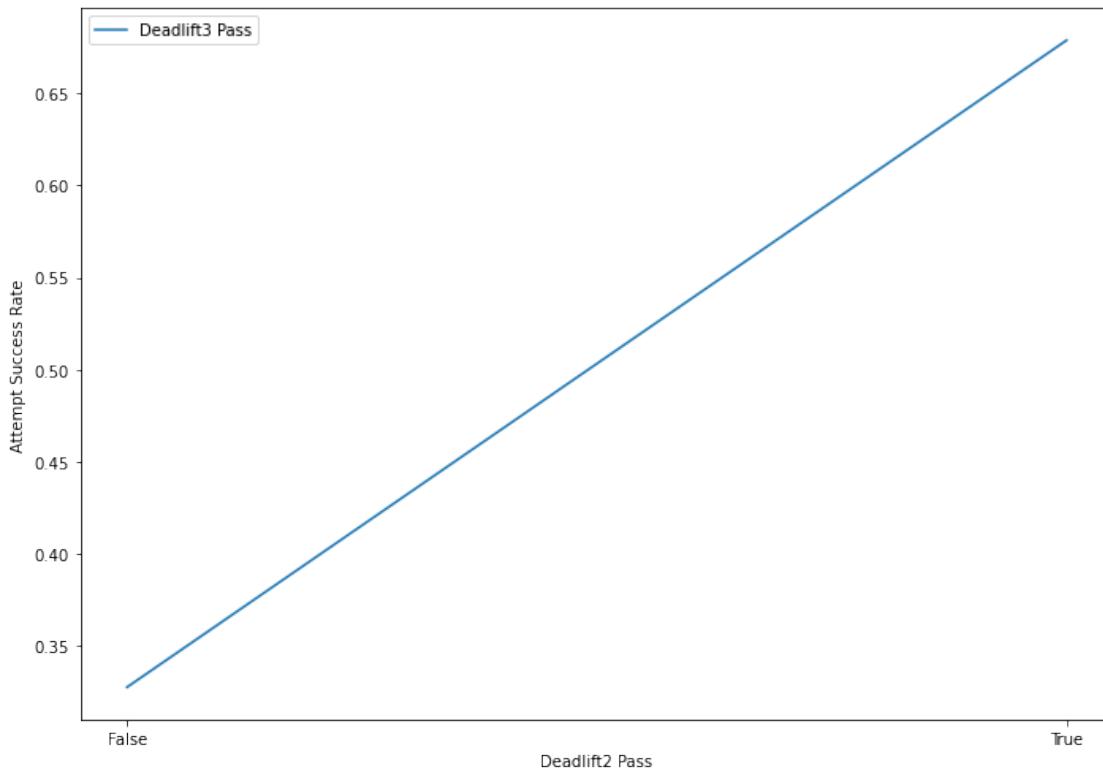


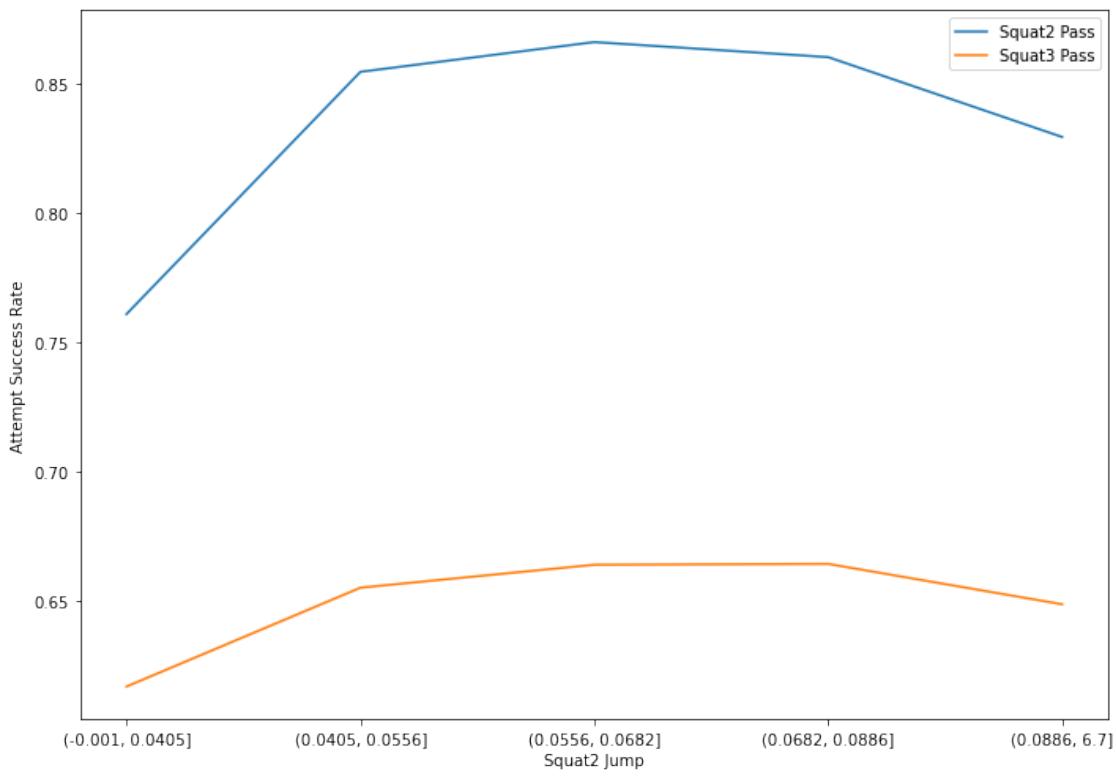


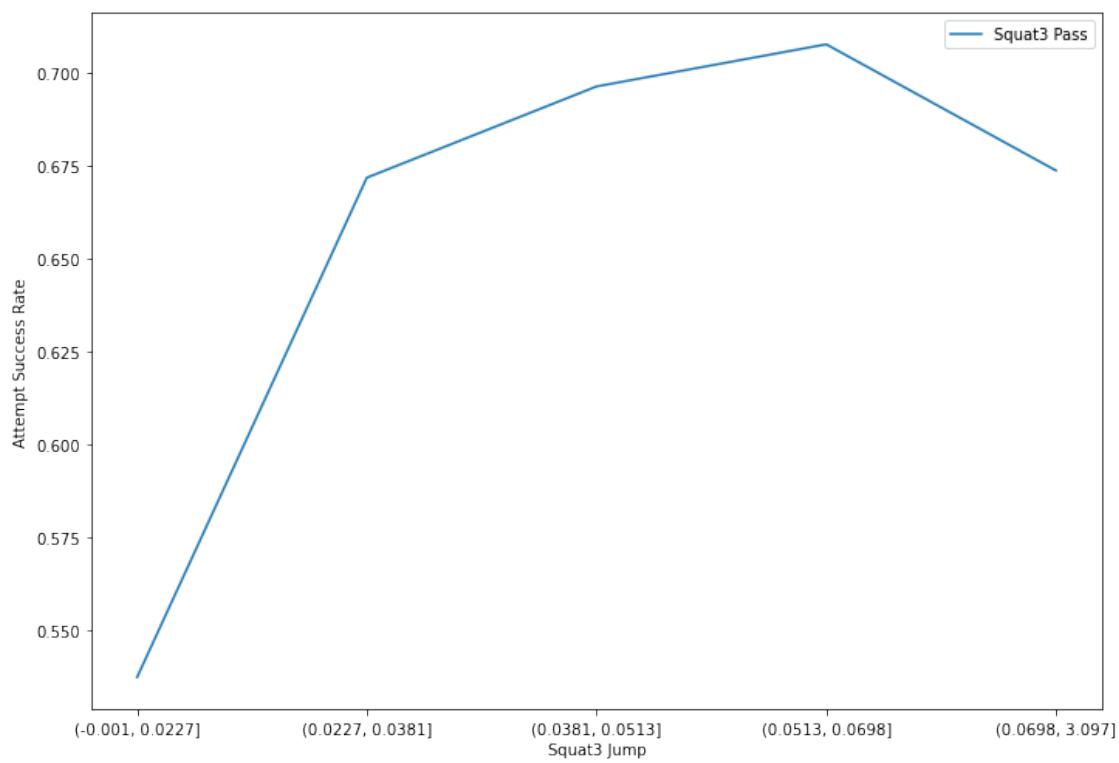
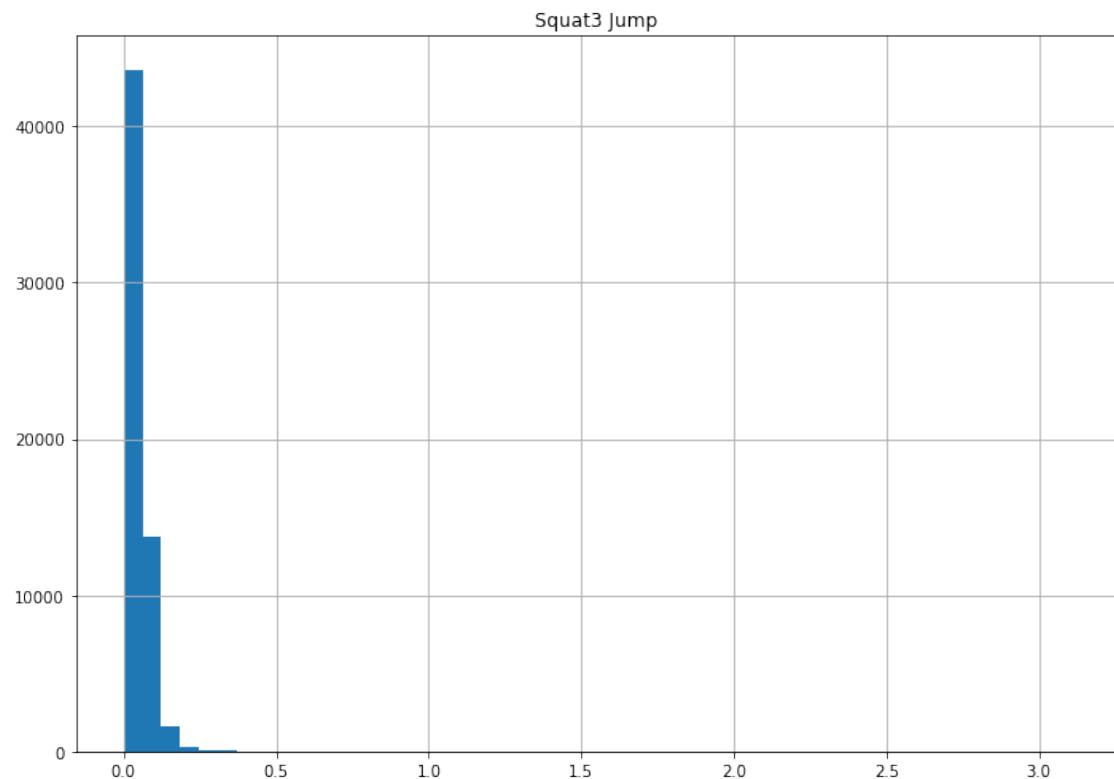


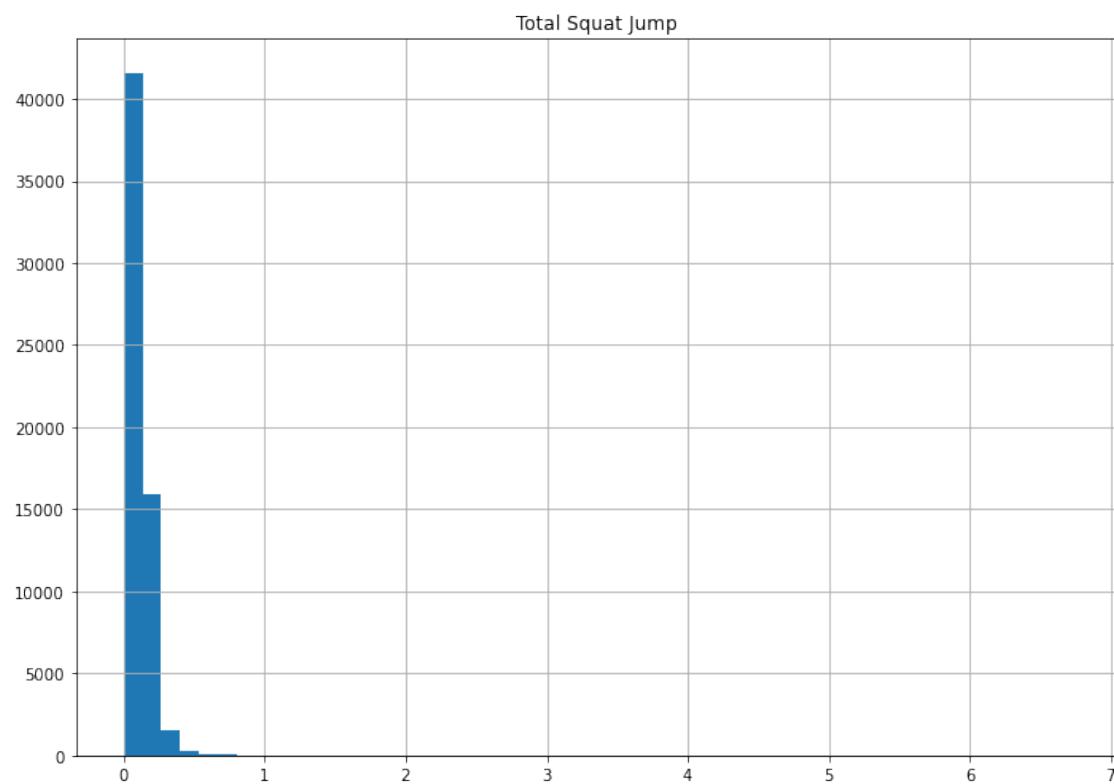


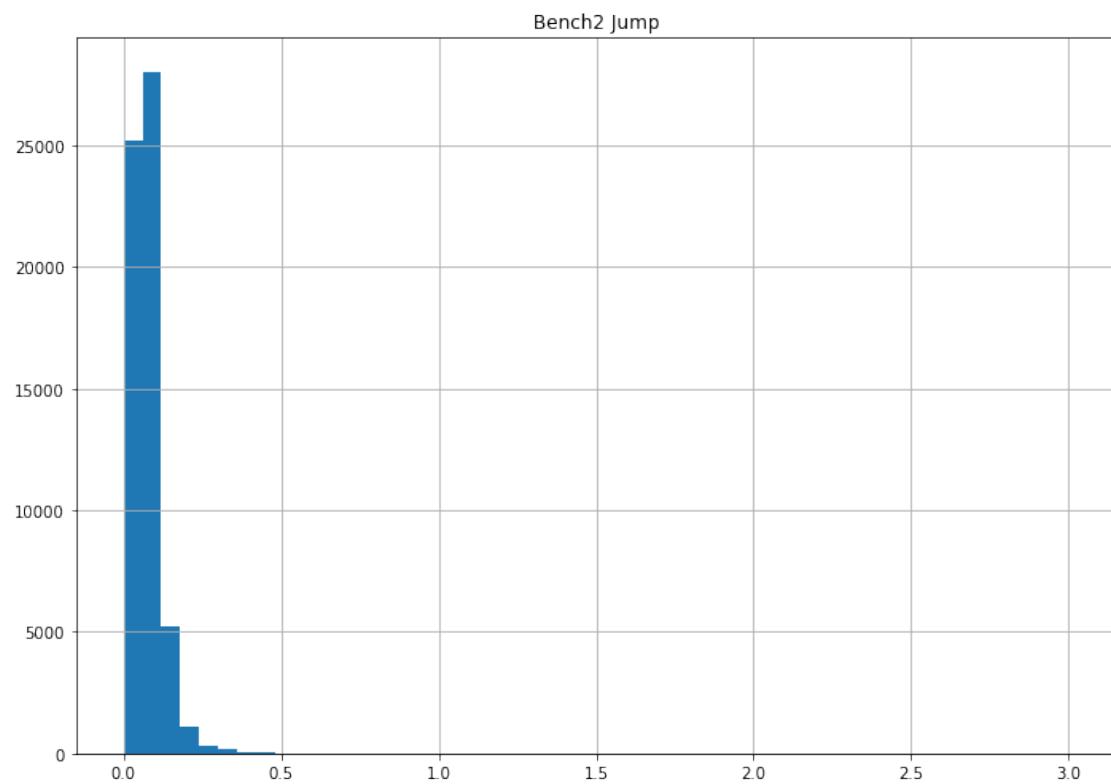
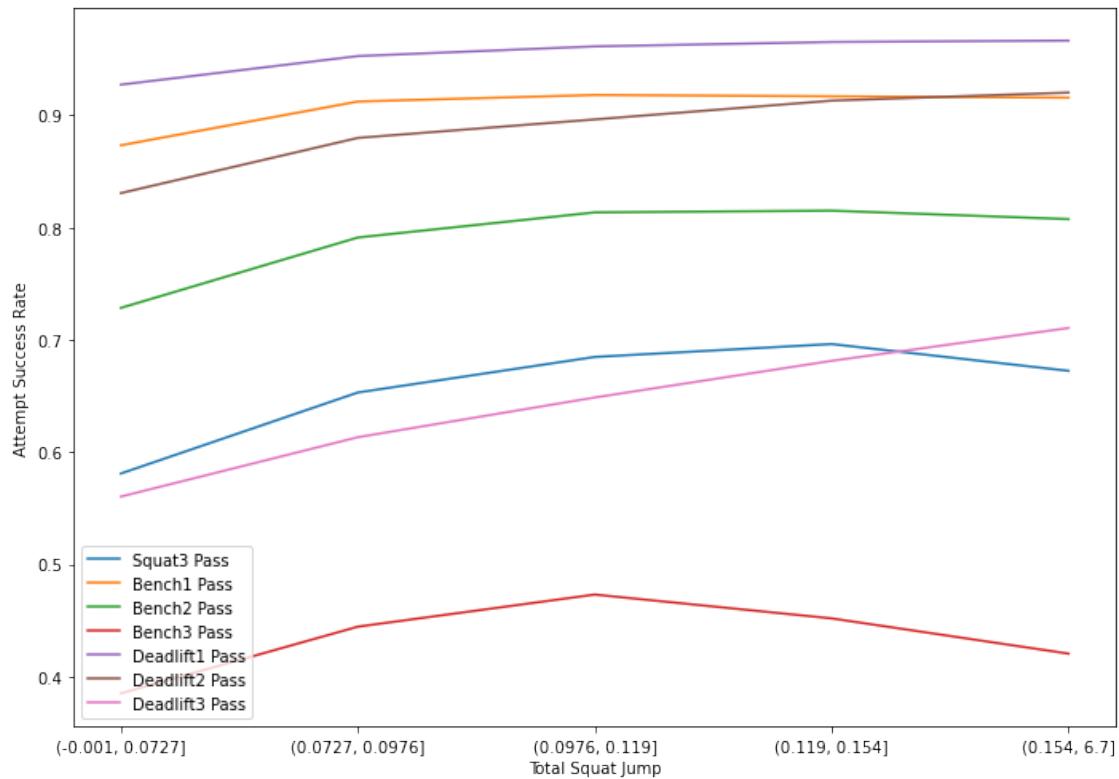


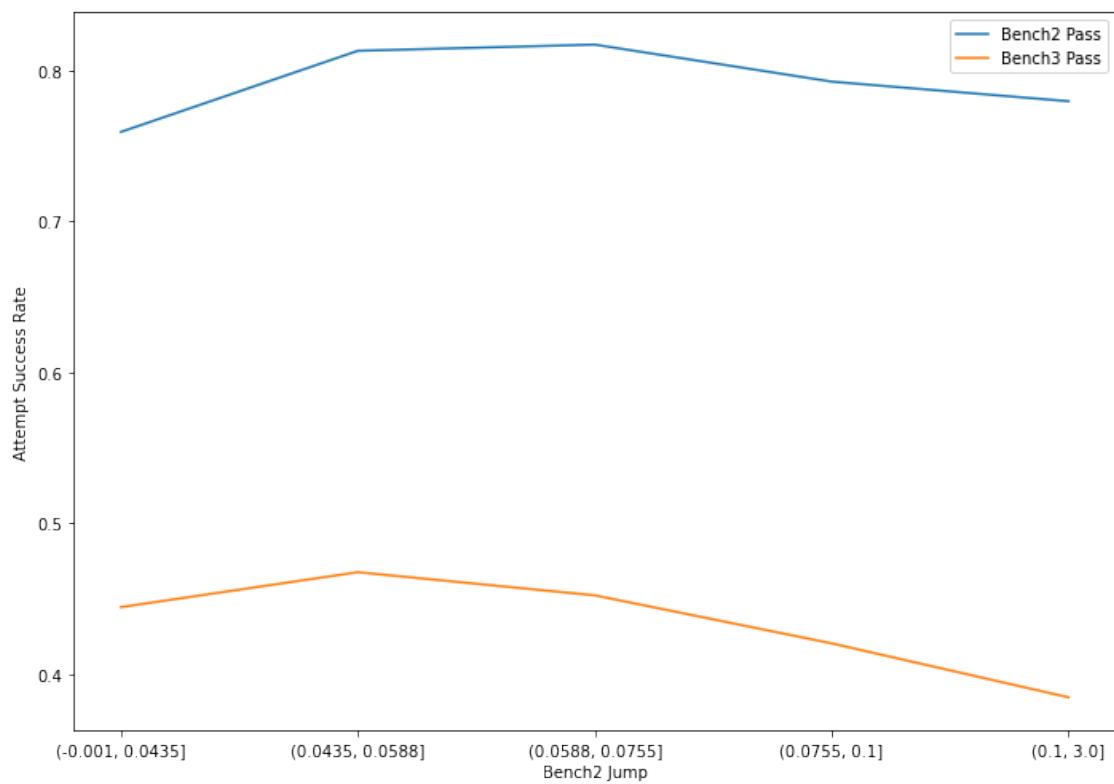


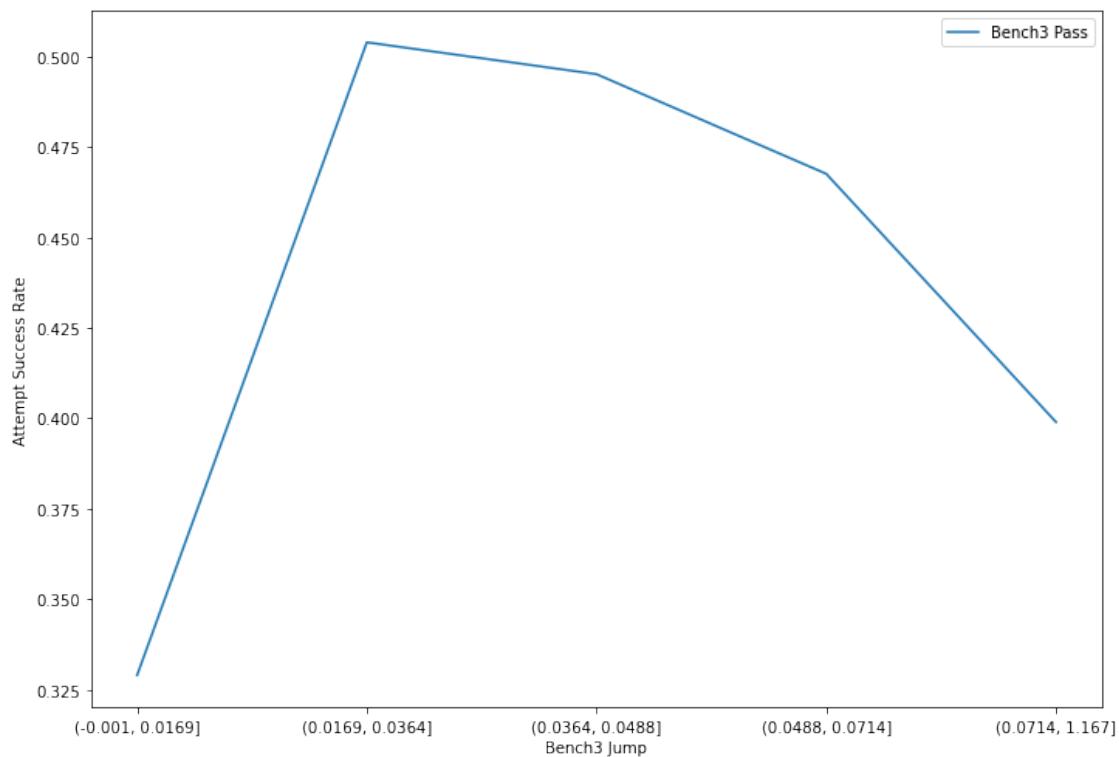
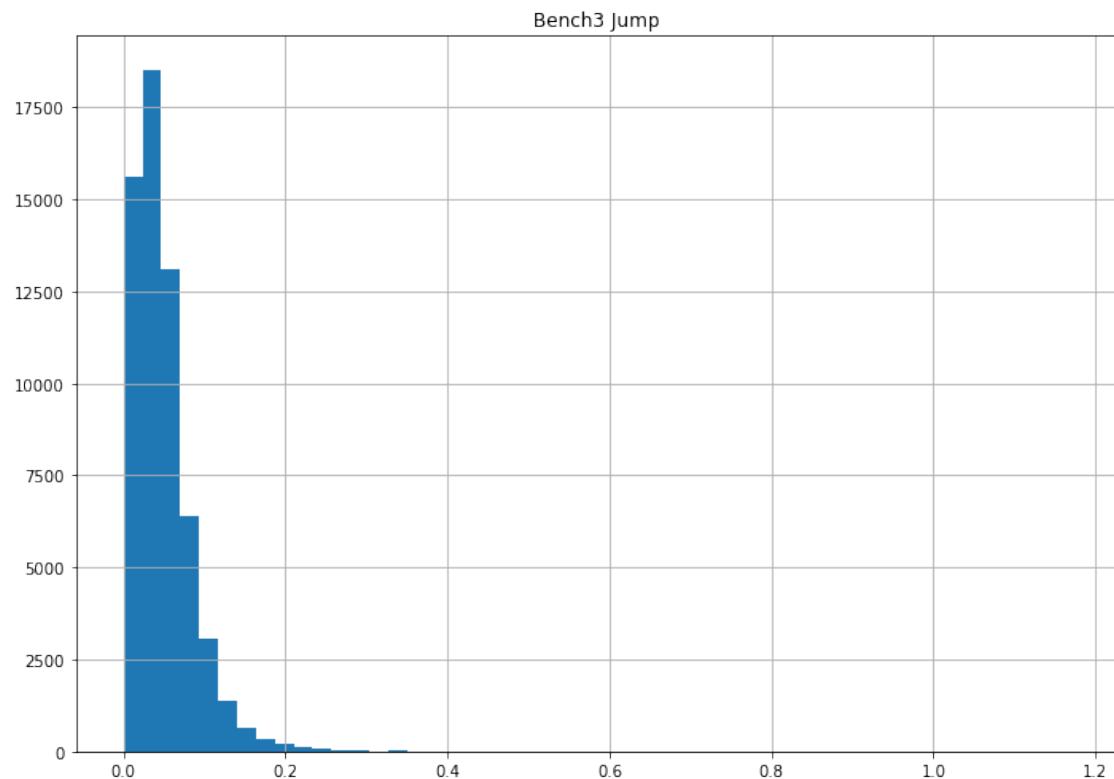


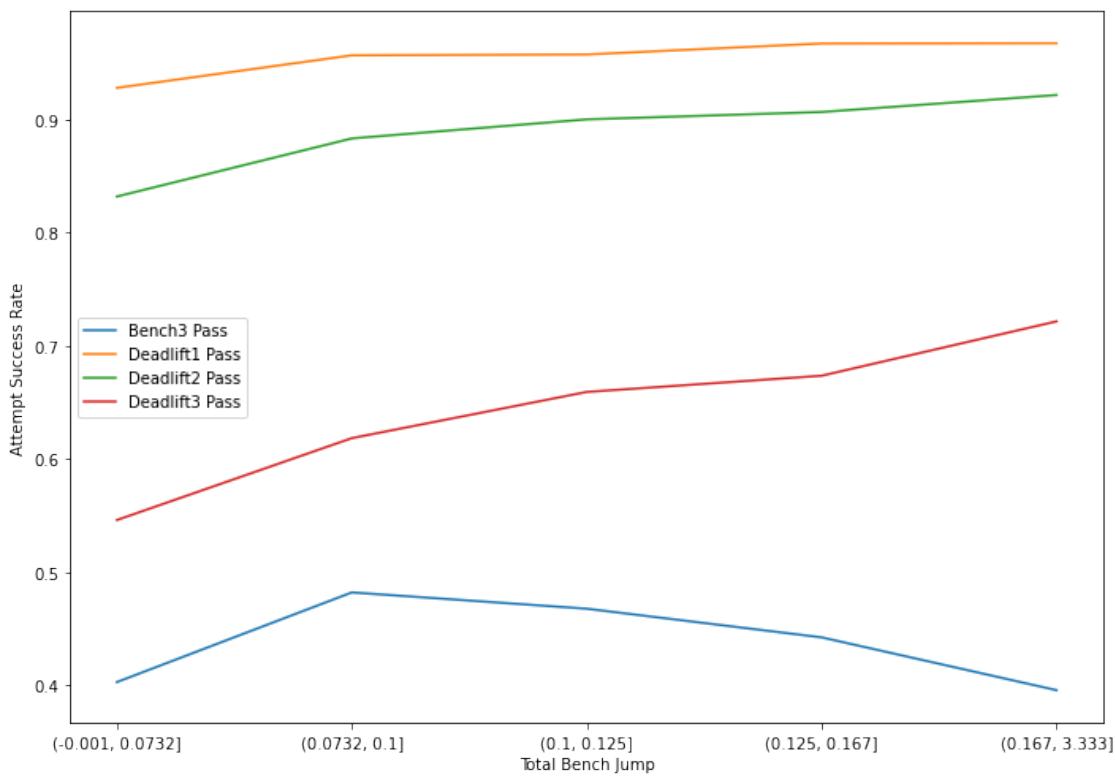
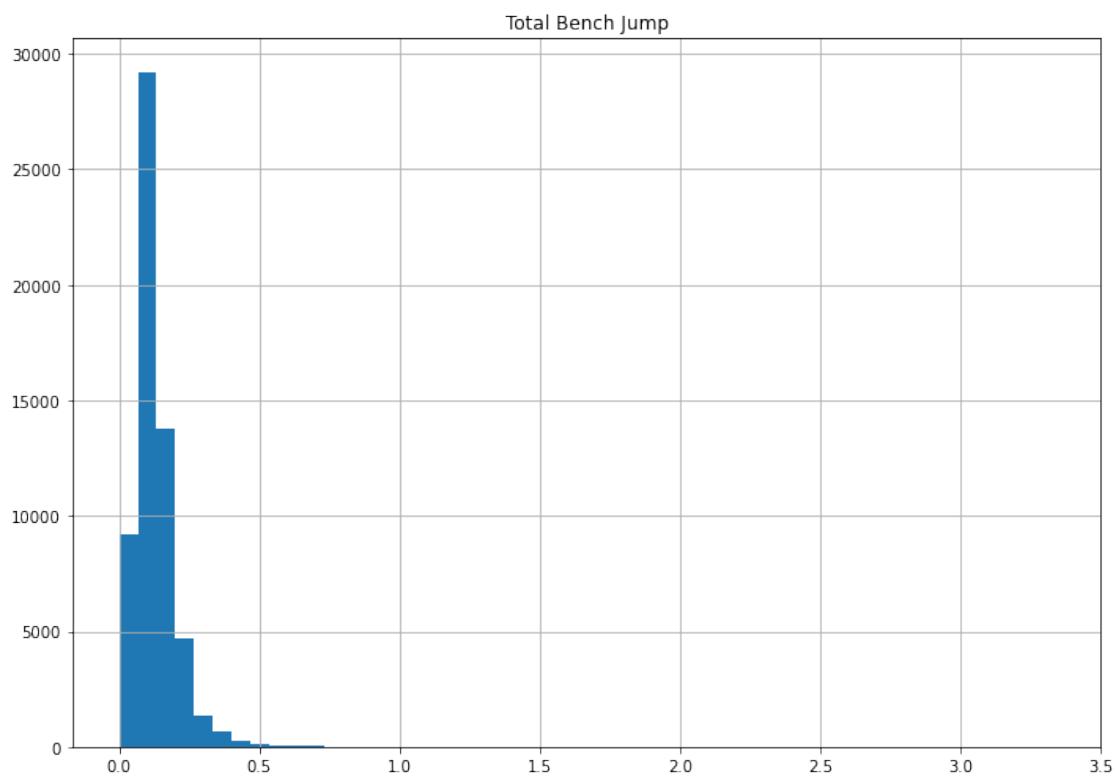


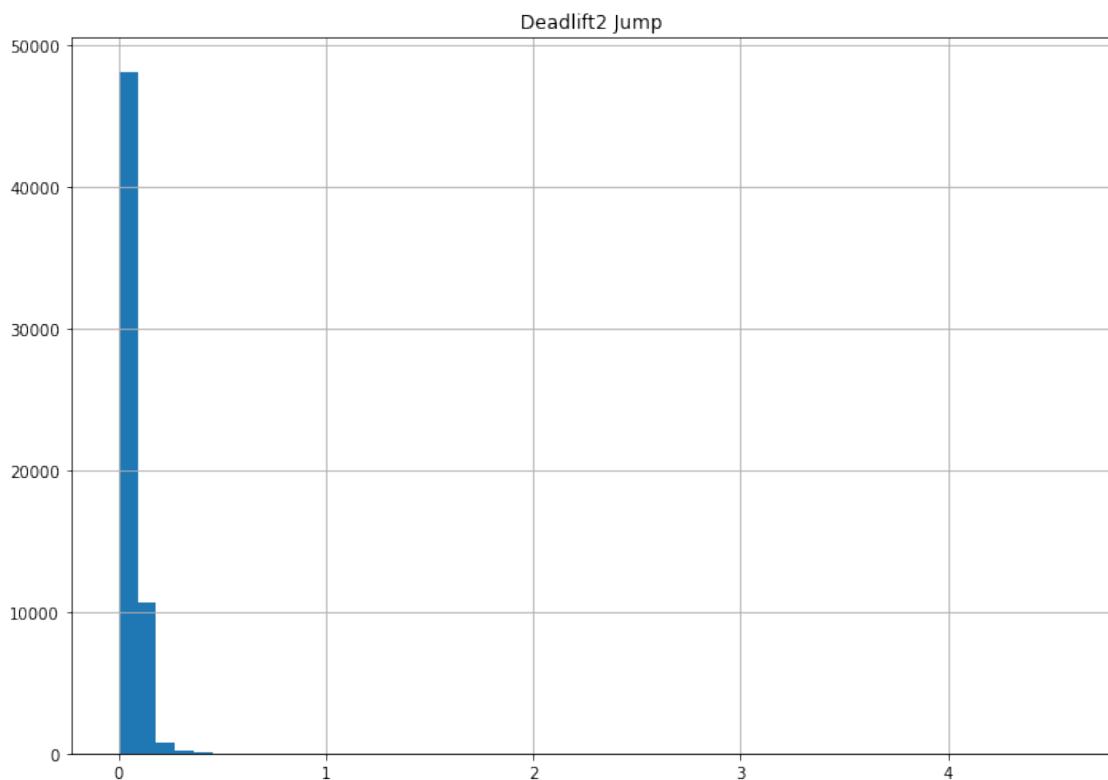


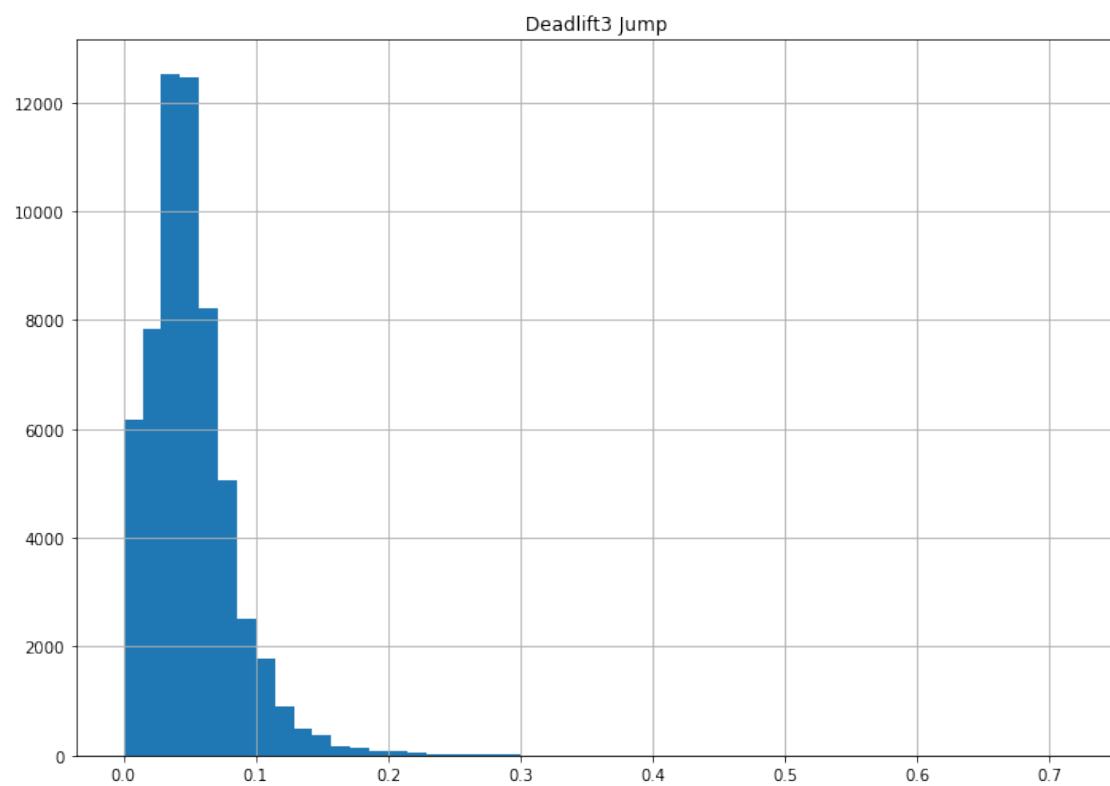
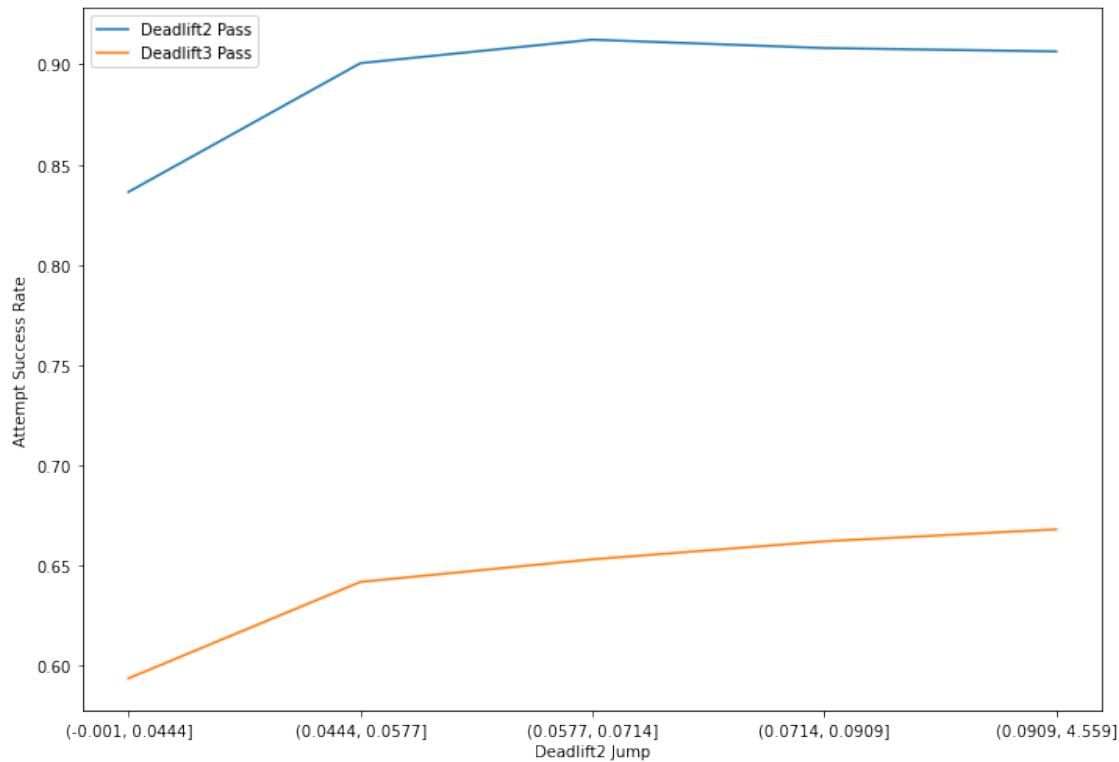


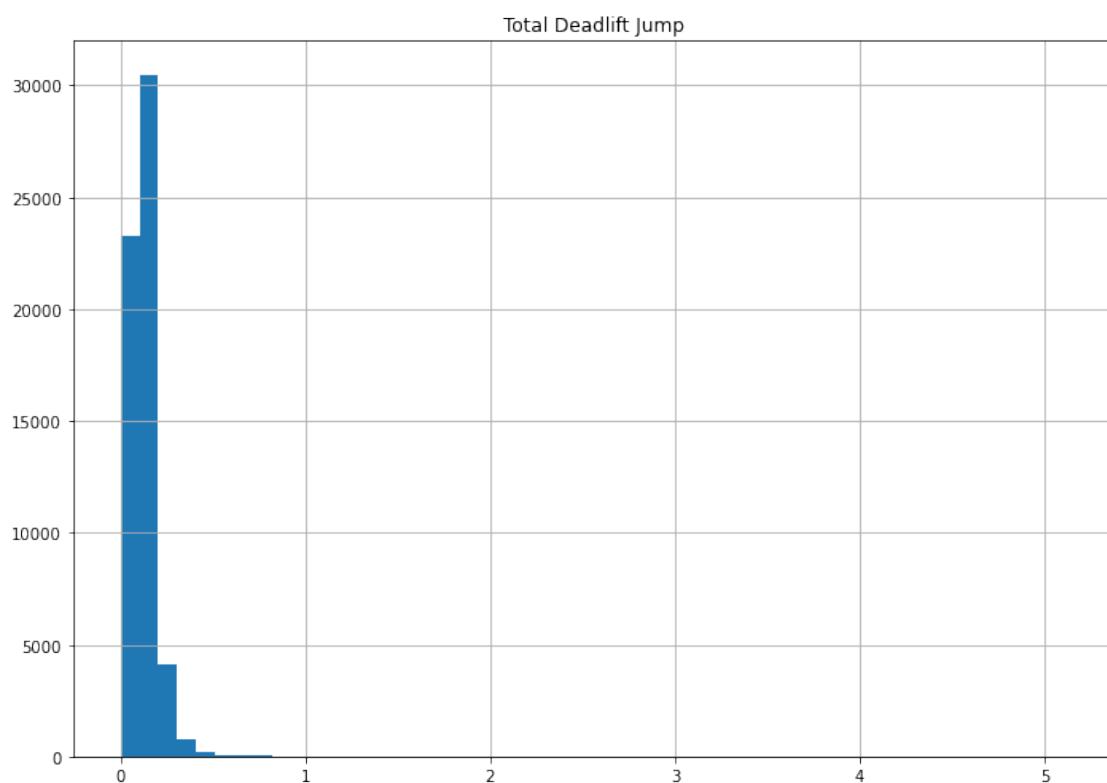
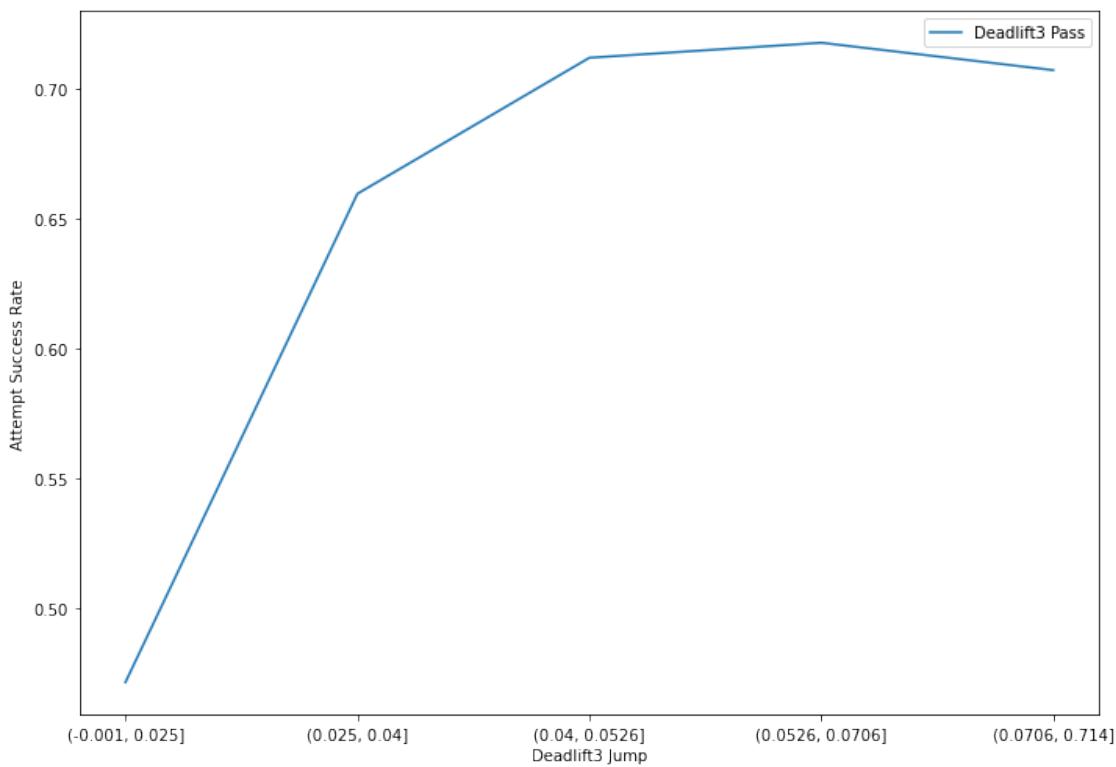


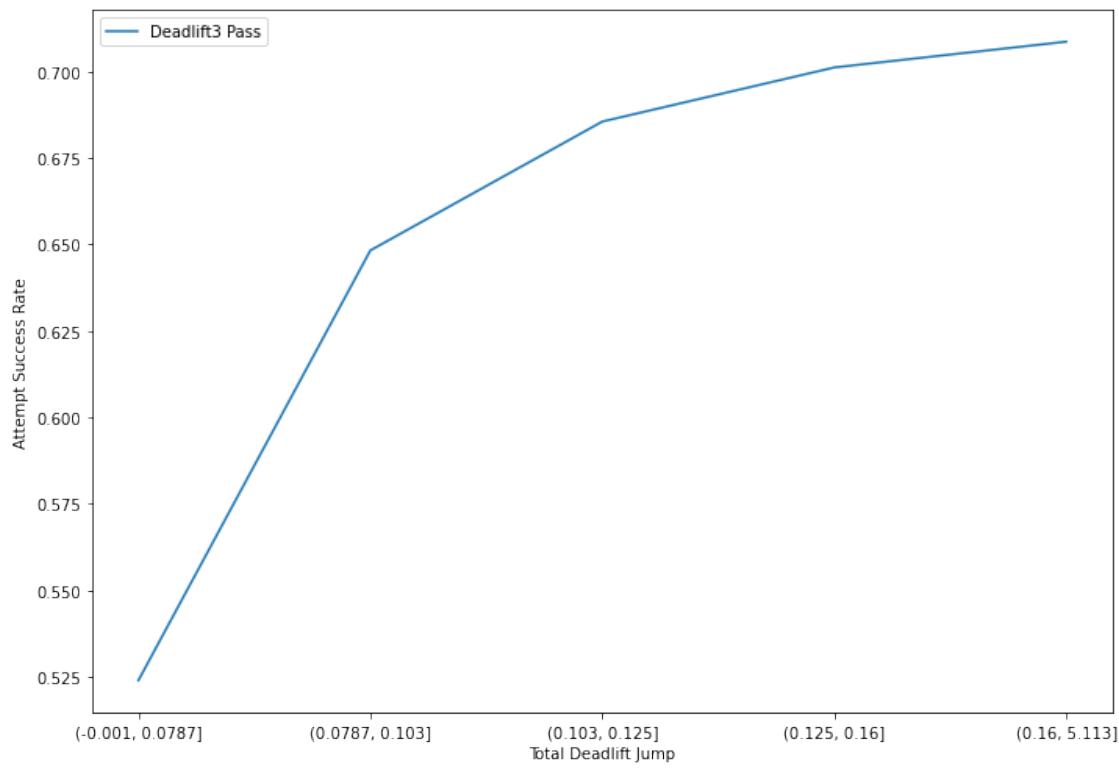


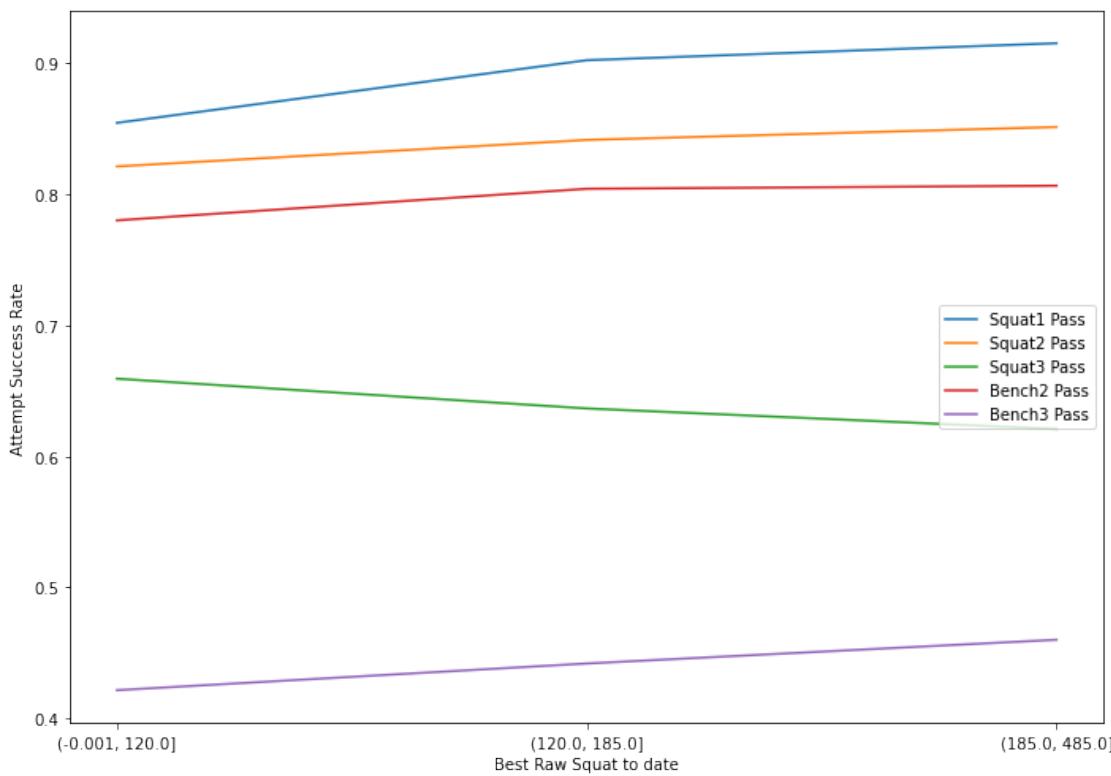
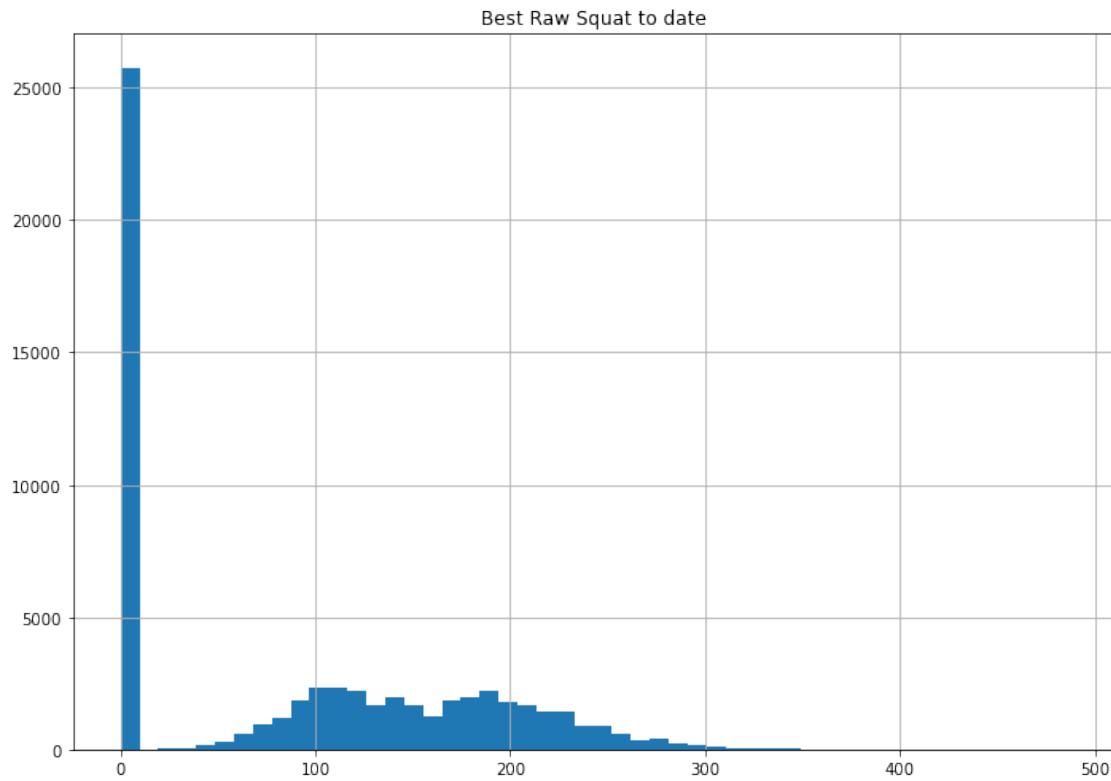




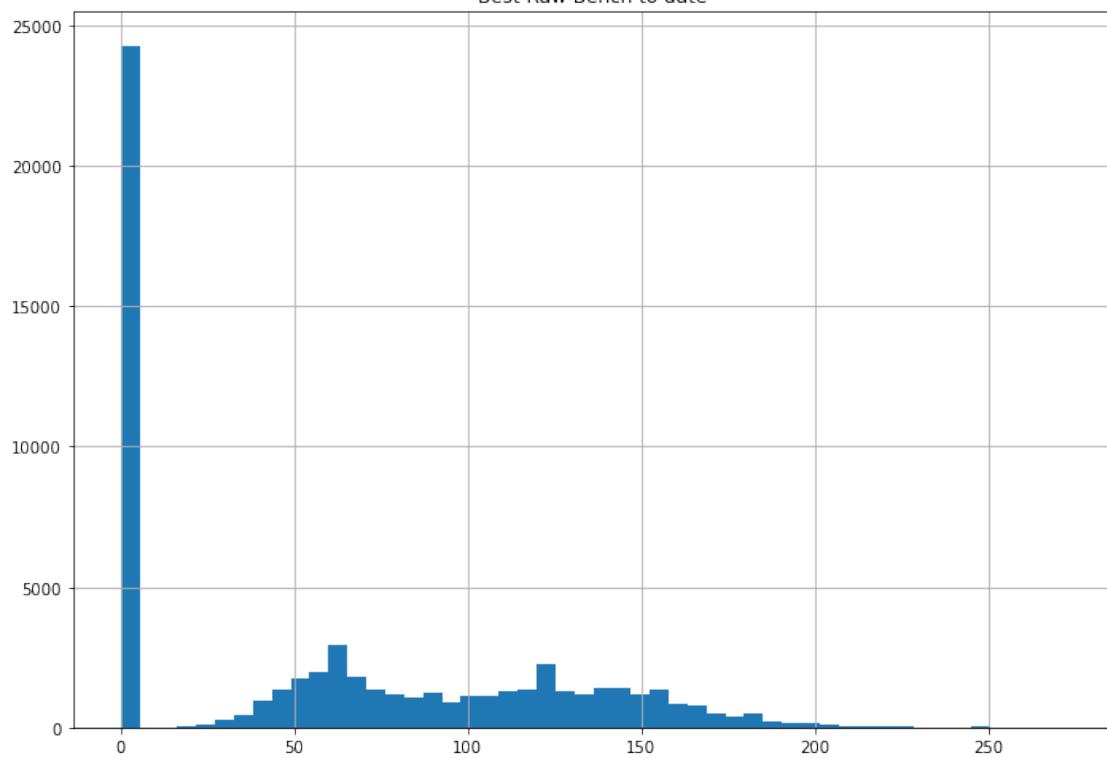


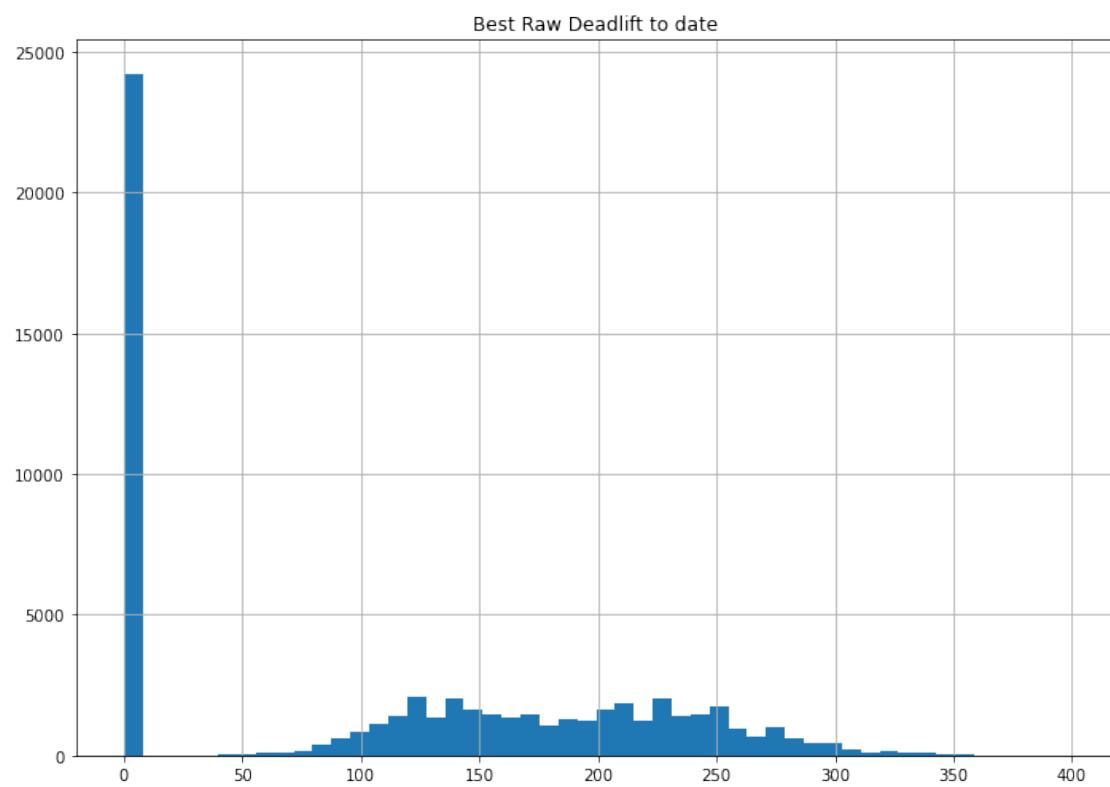
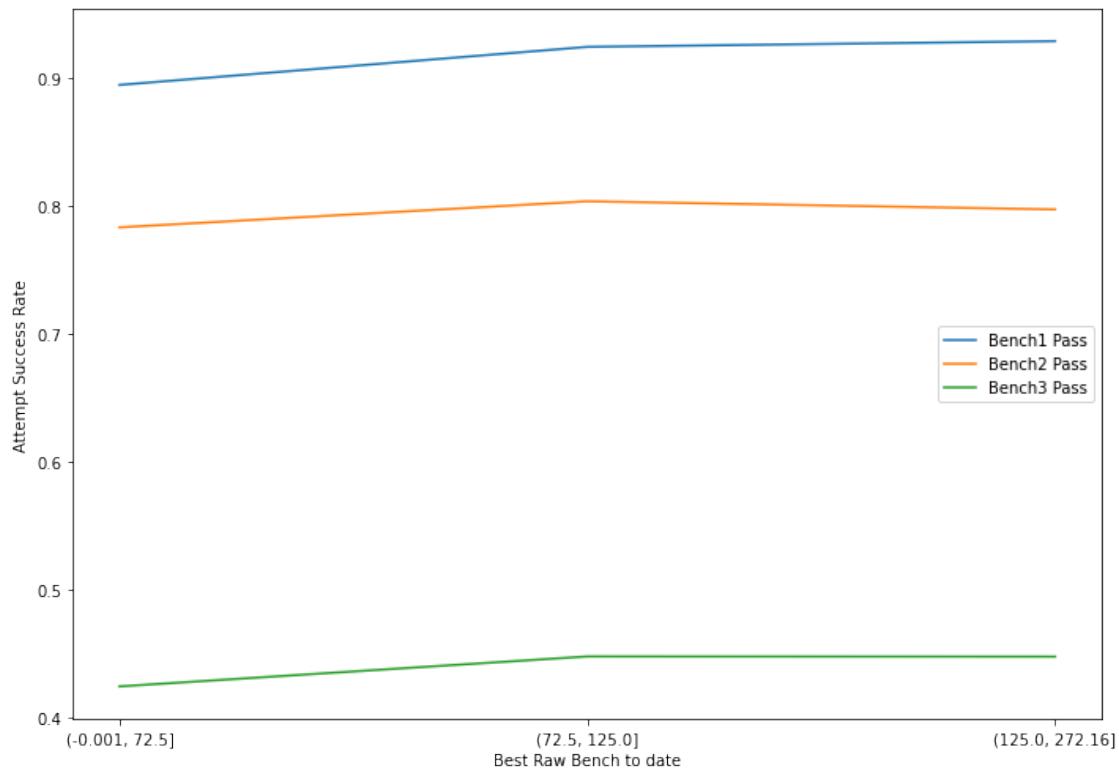


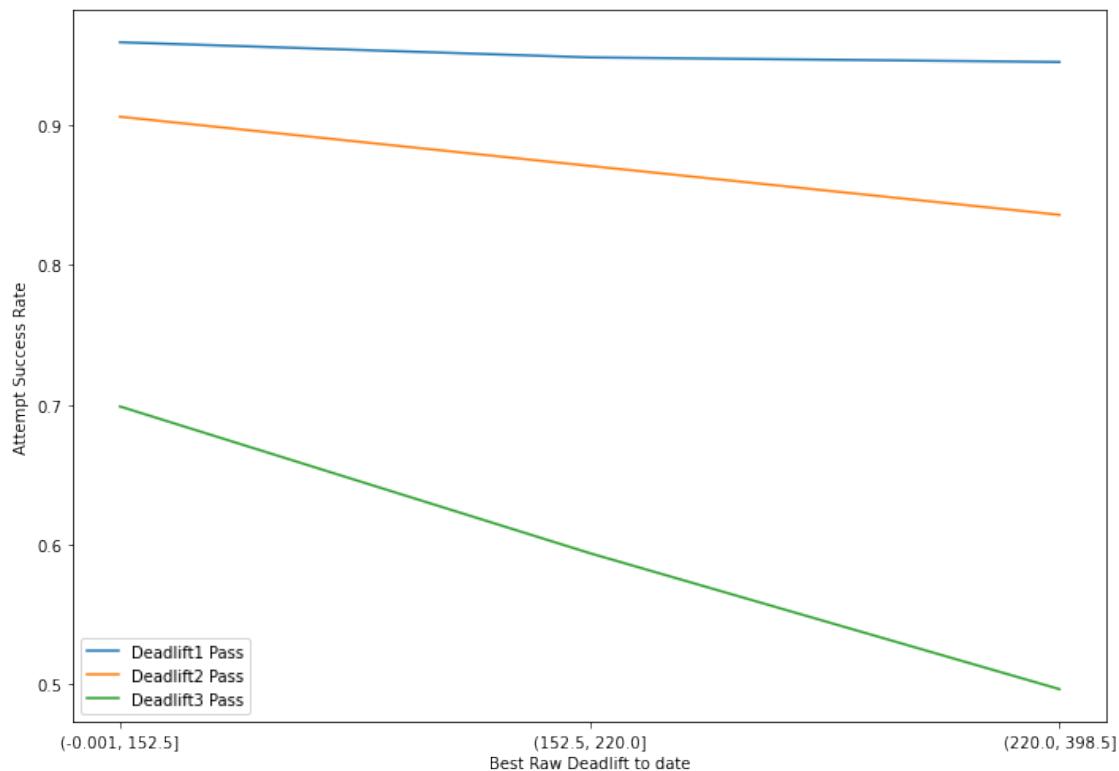


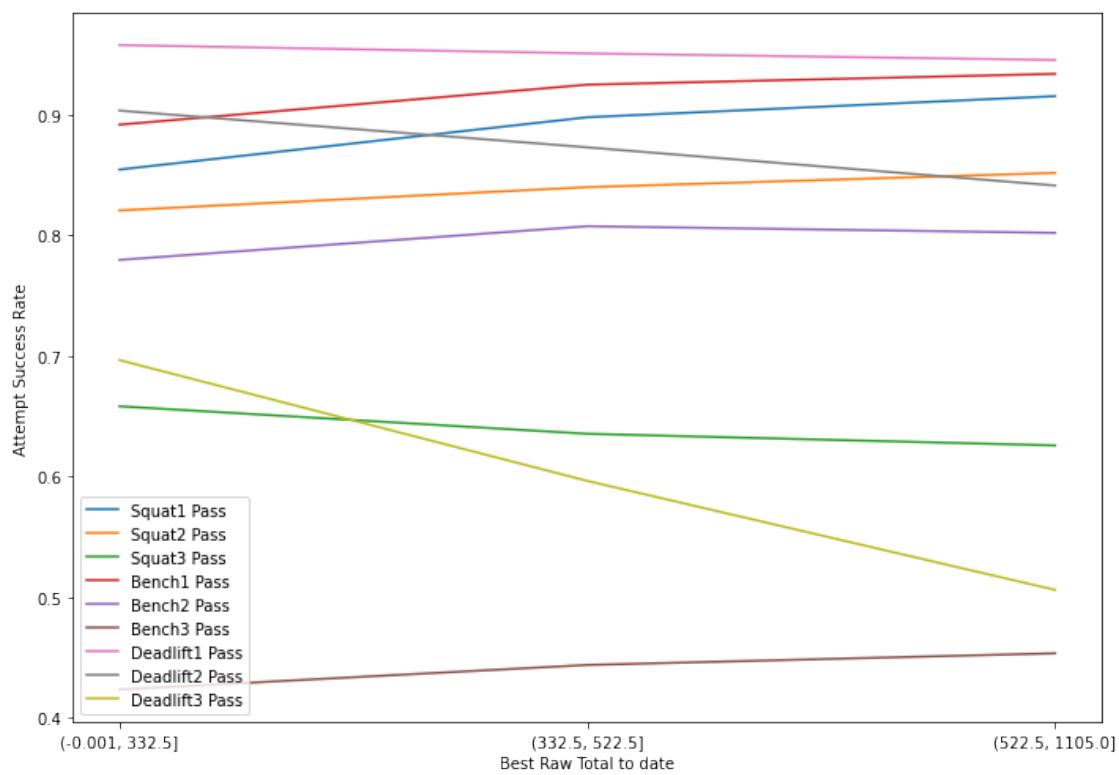
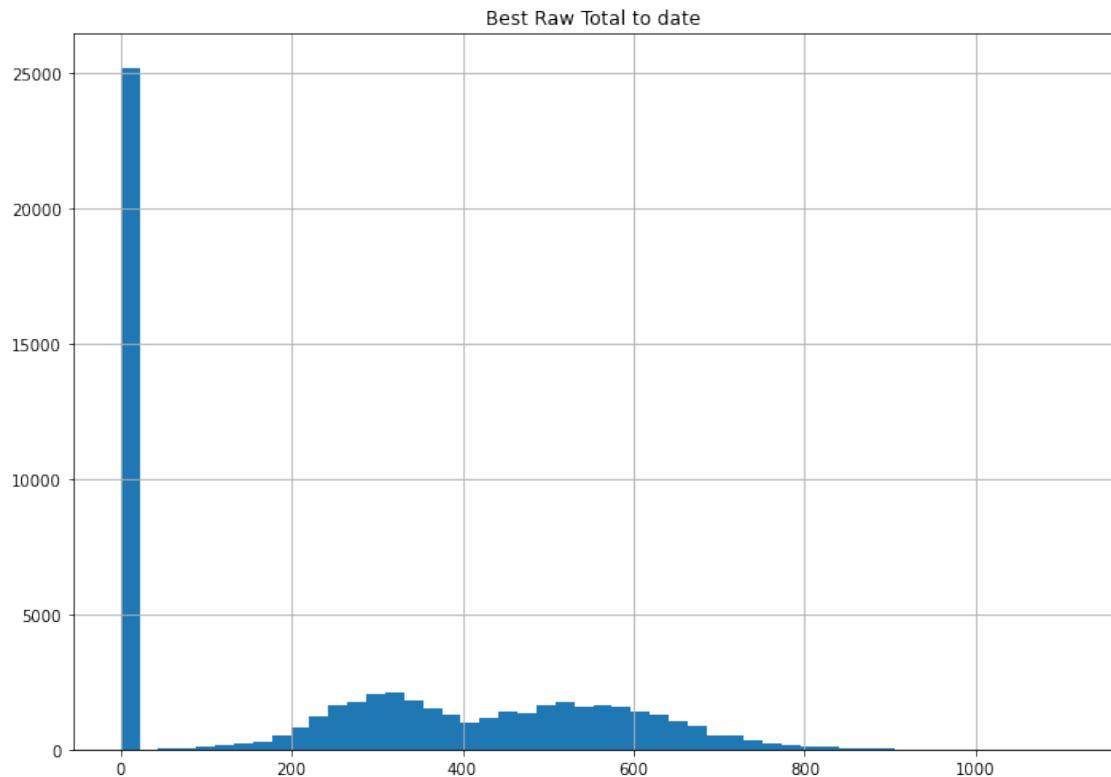


Best Raw Bench to date

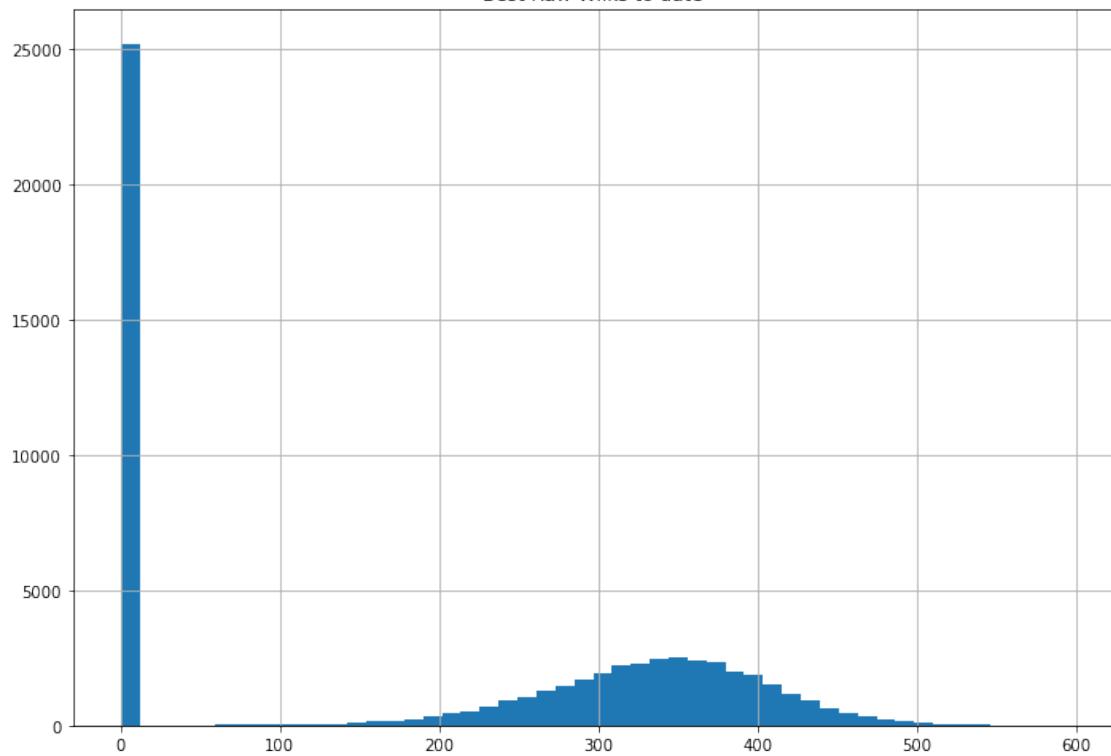


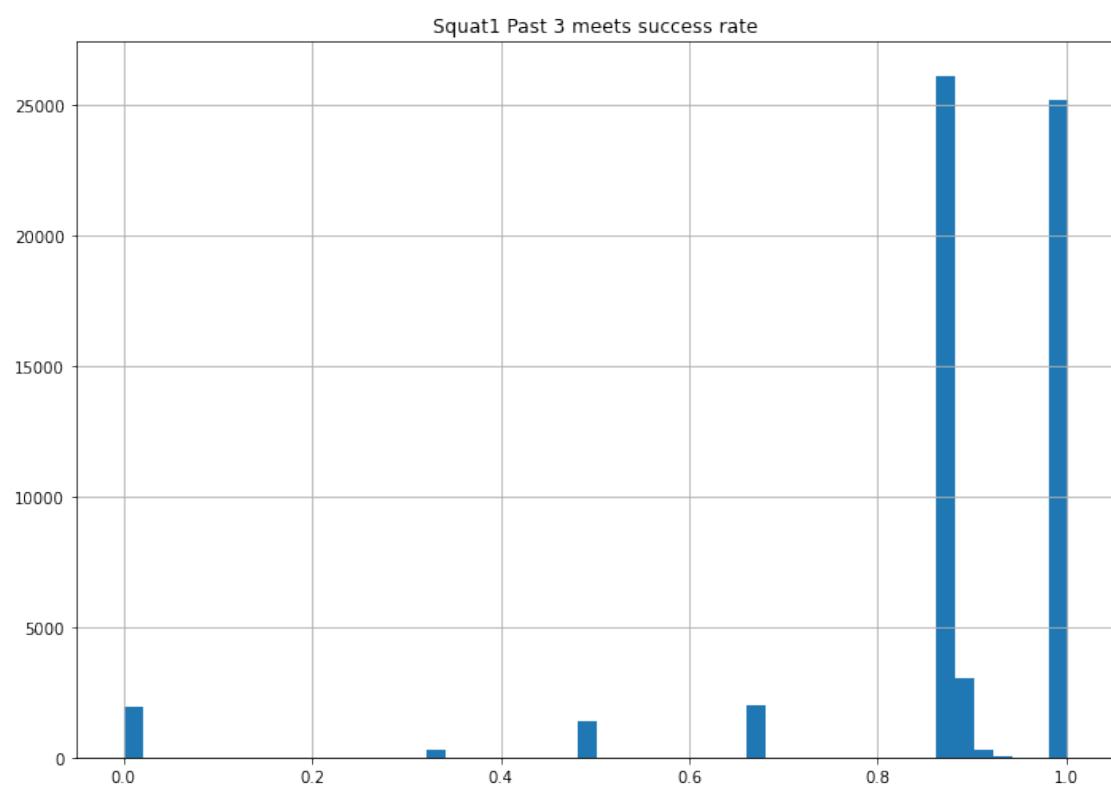
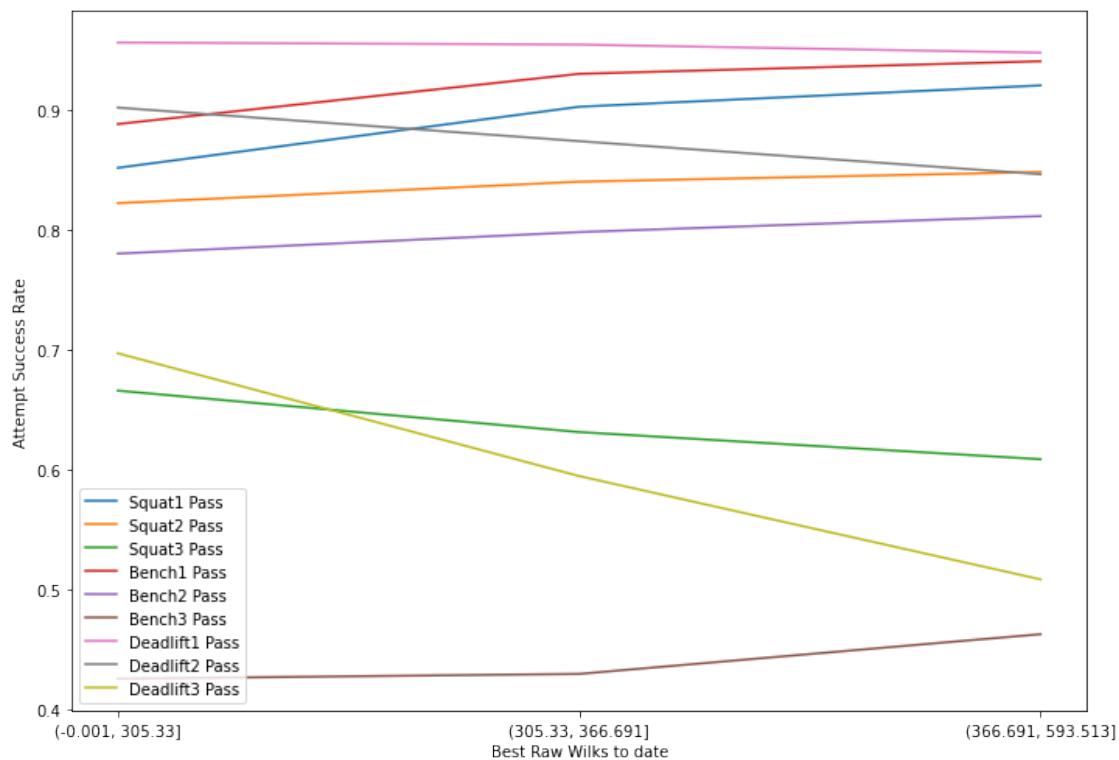


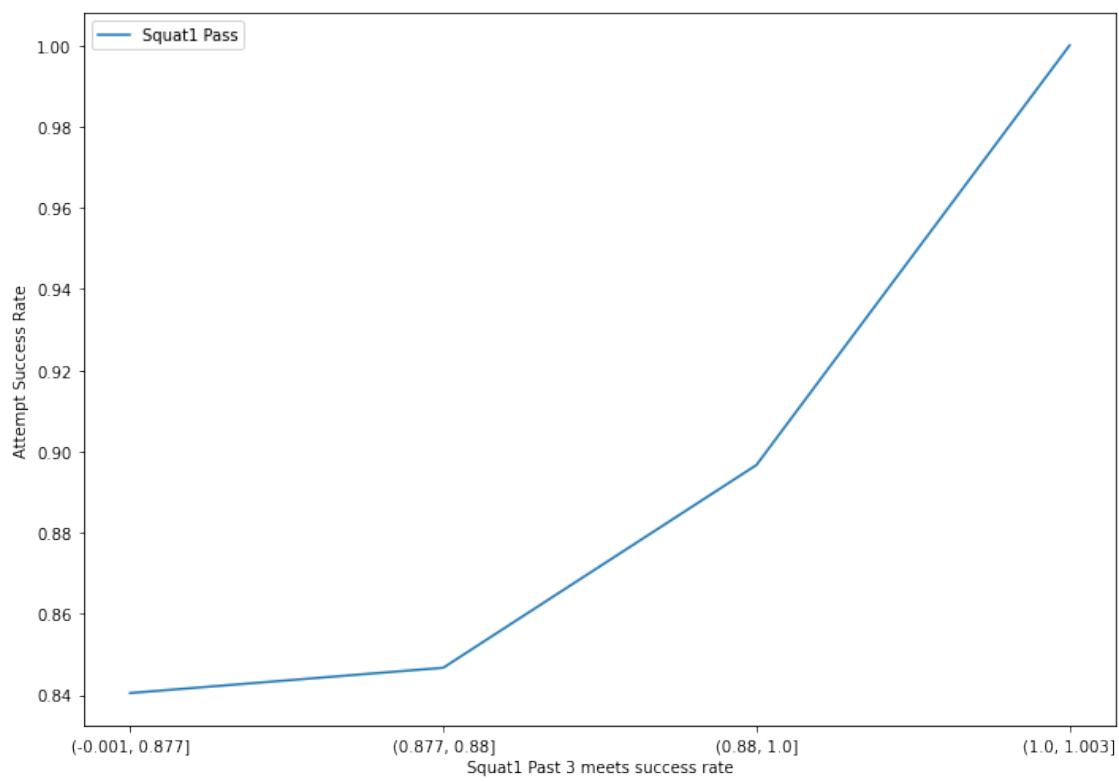




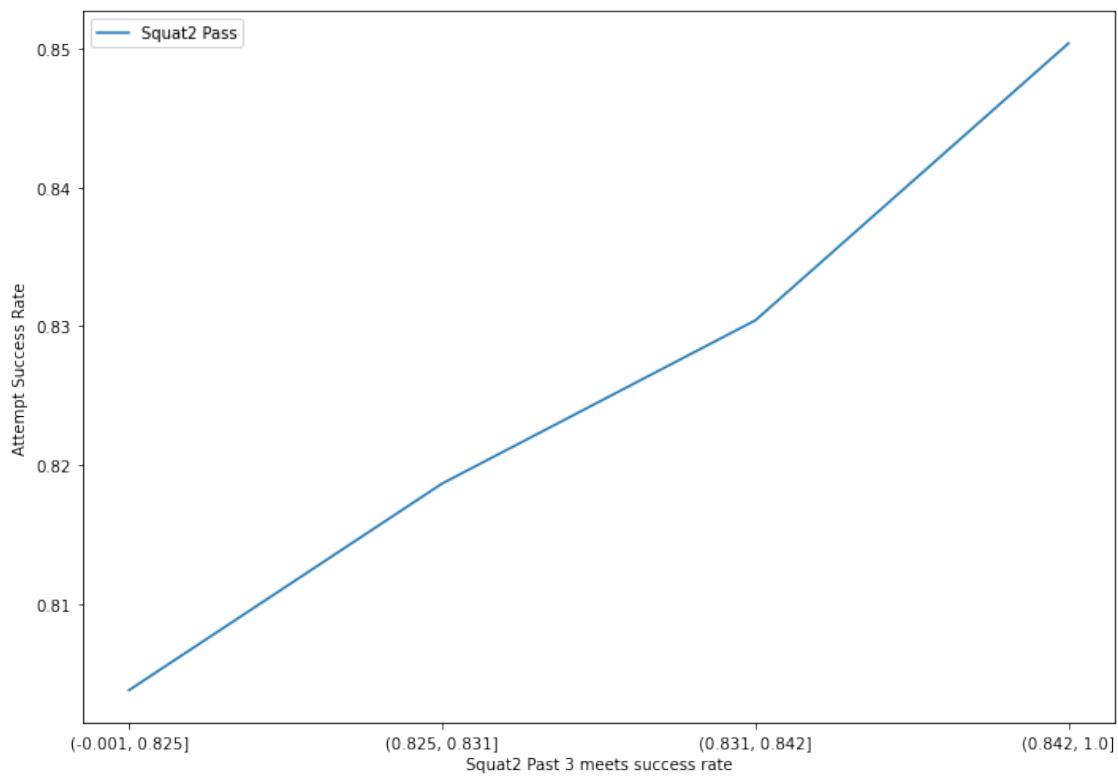
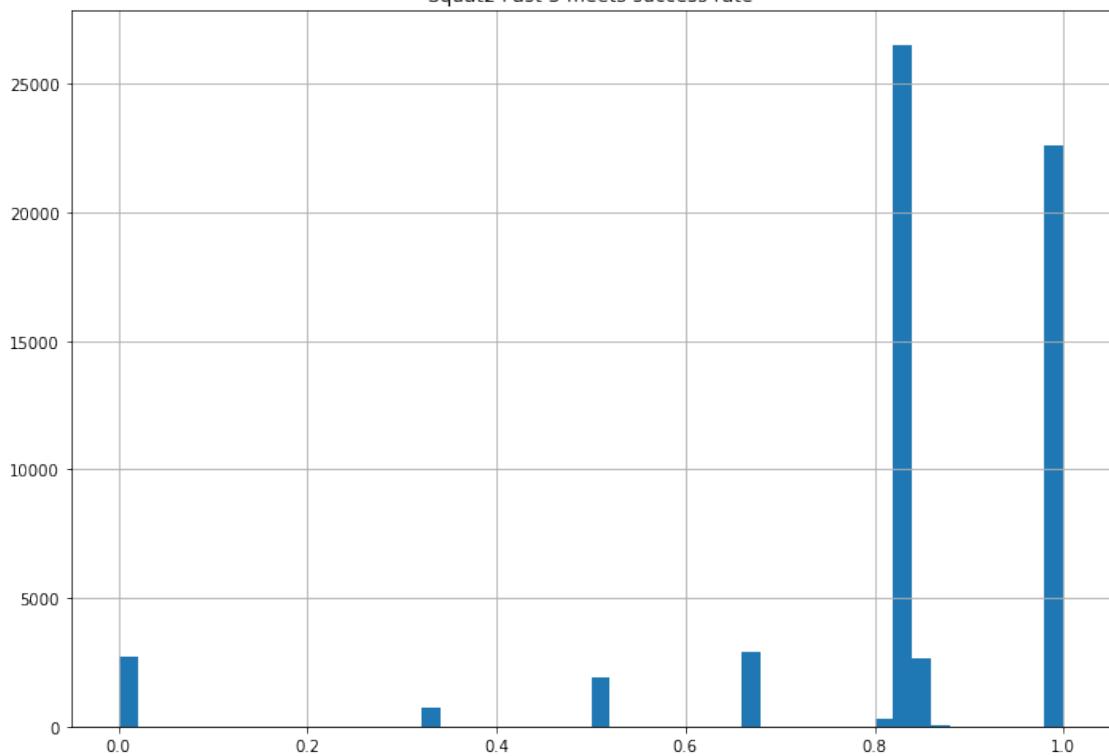
Best Raw Wilks to date



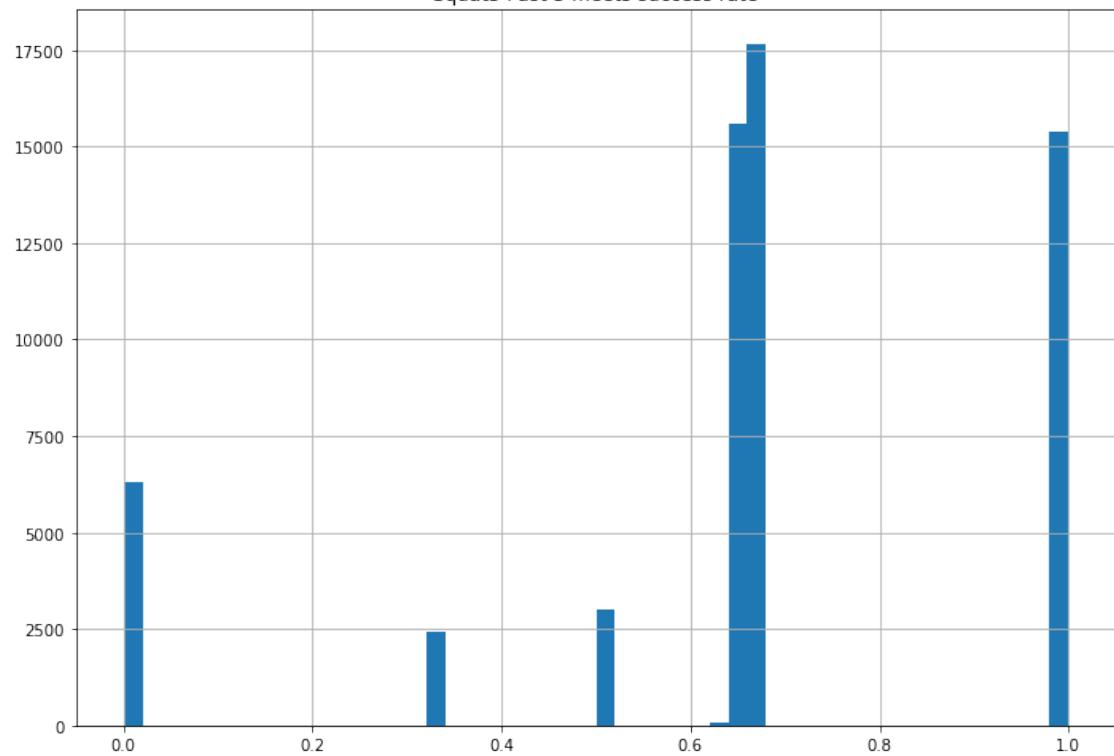


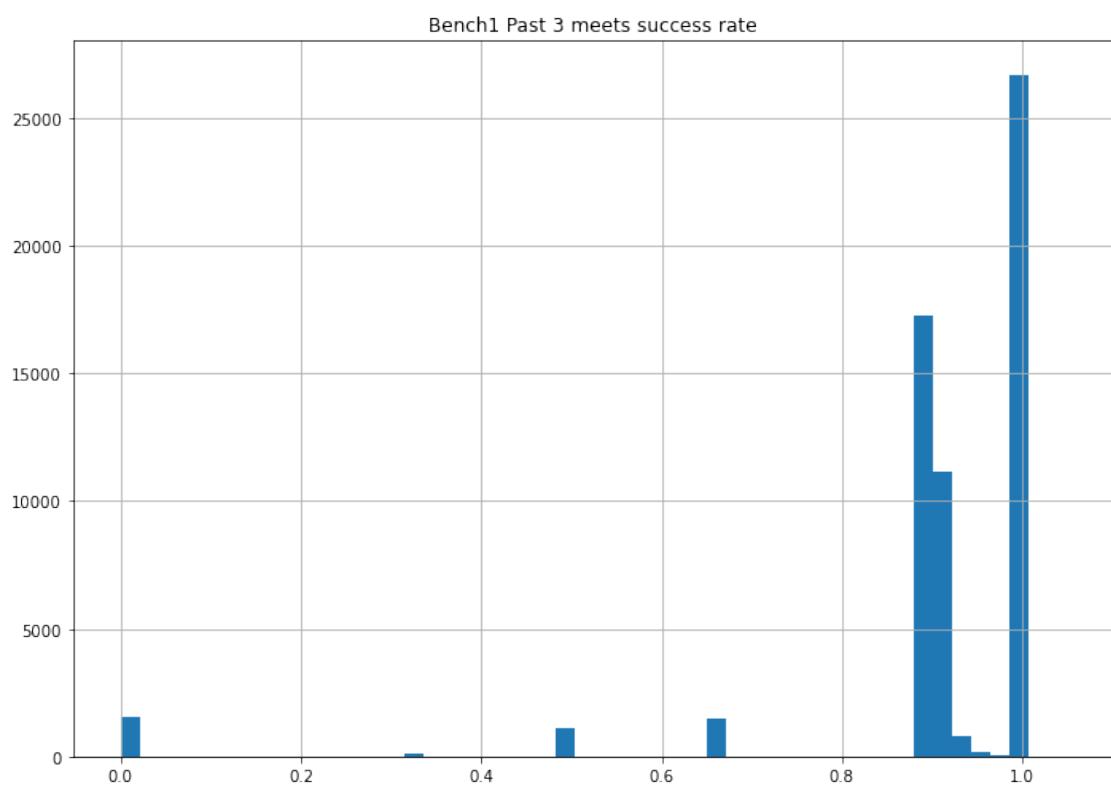
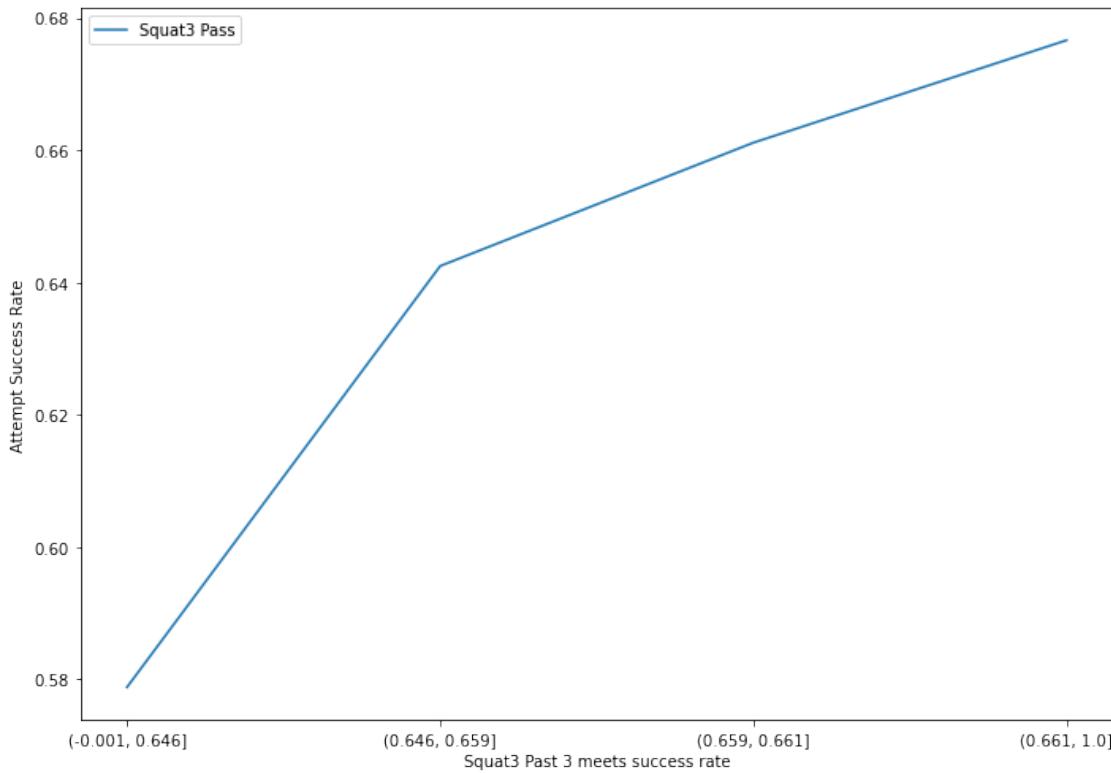


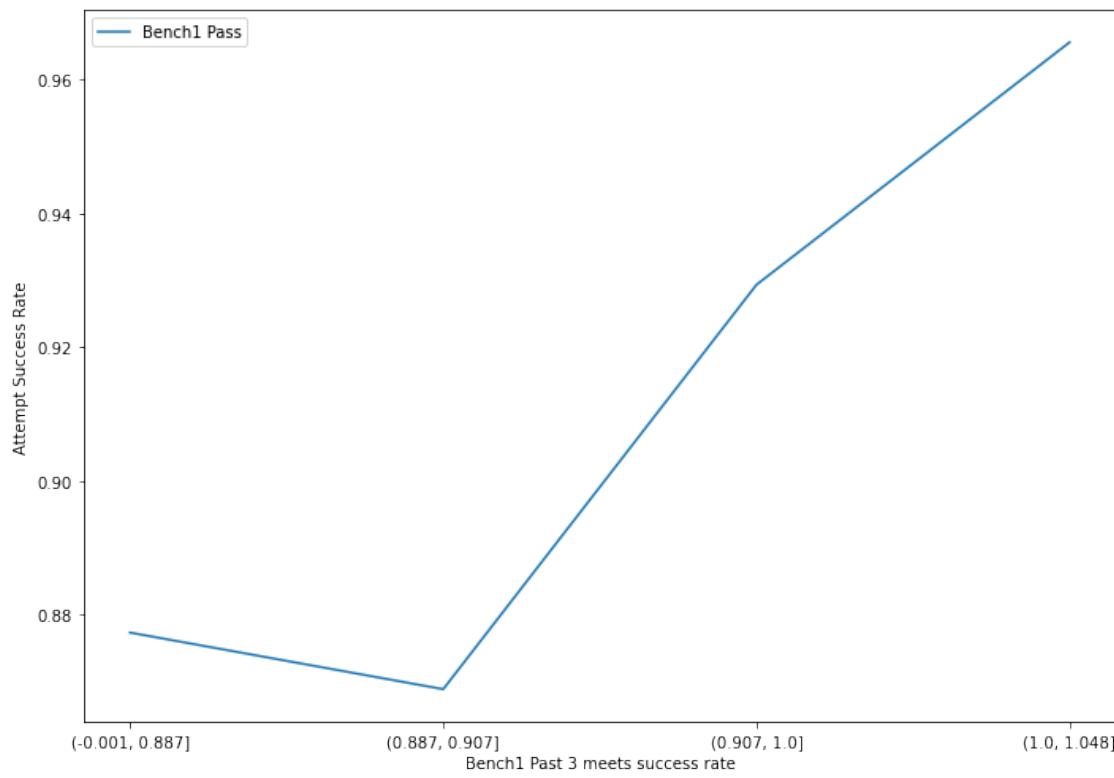
Squat2 Past 3 meets success rate

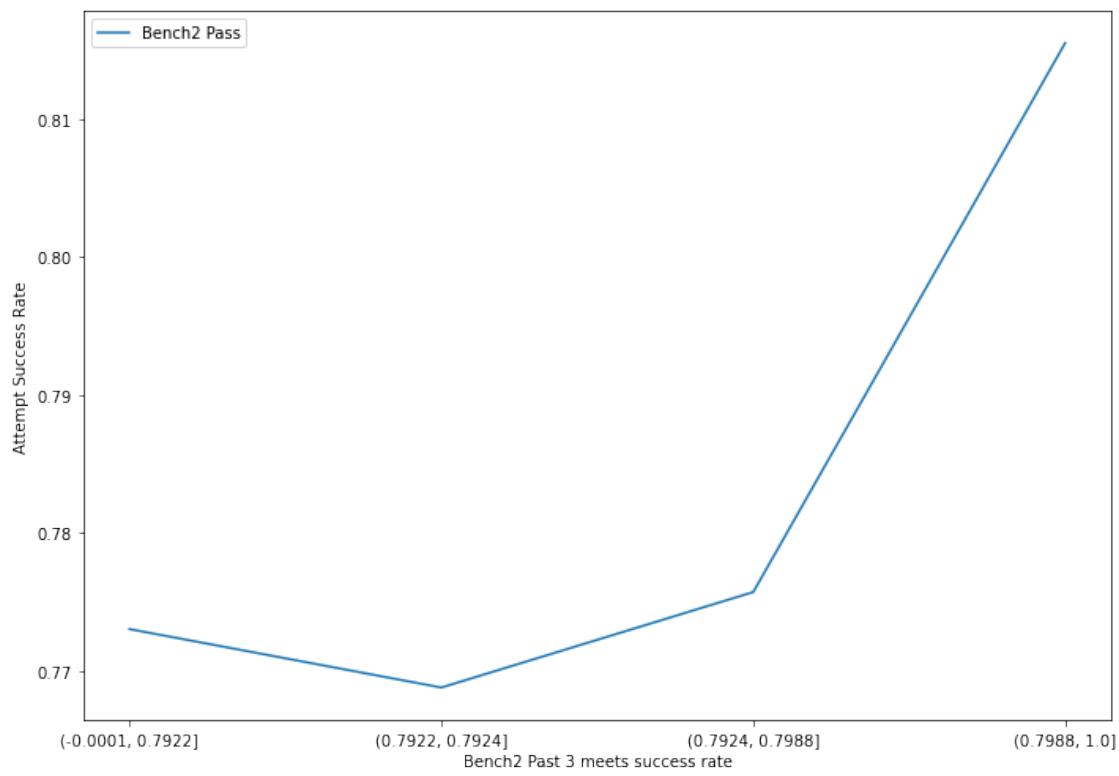
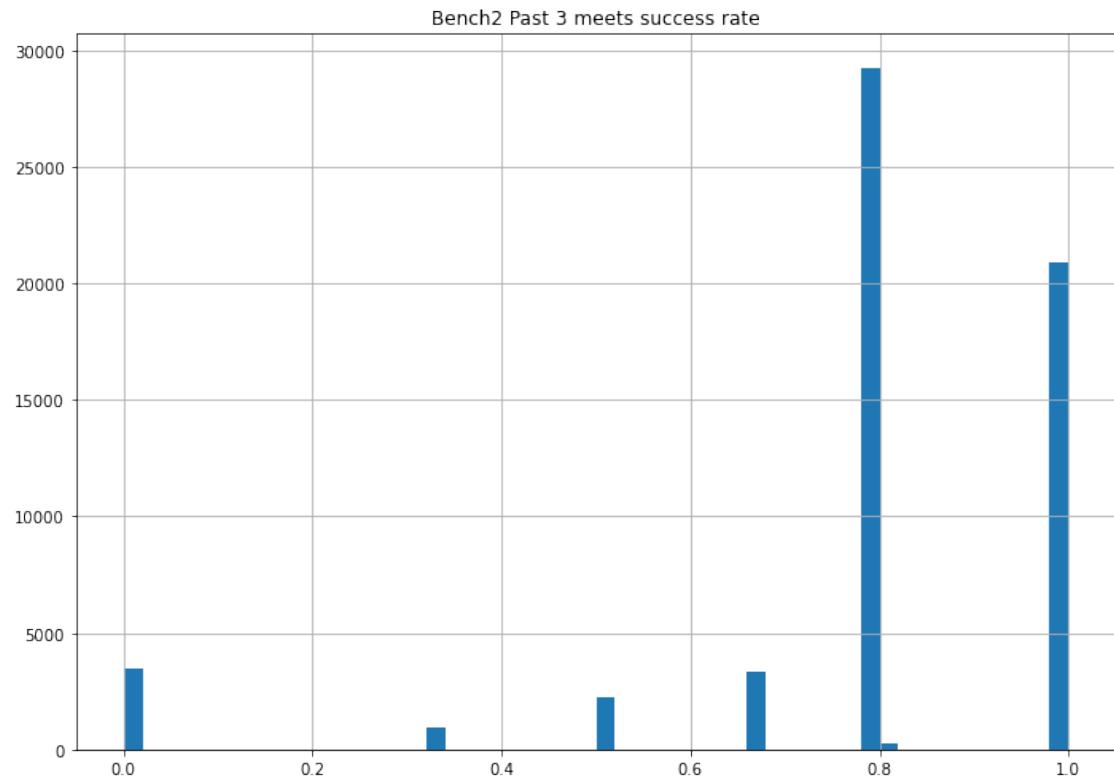


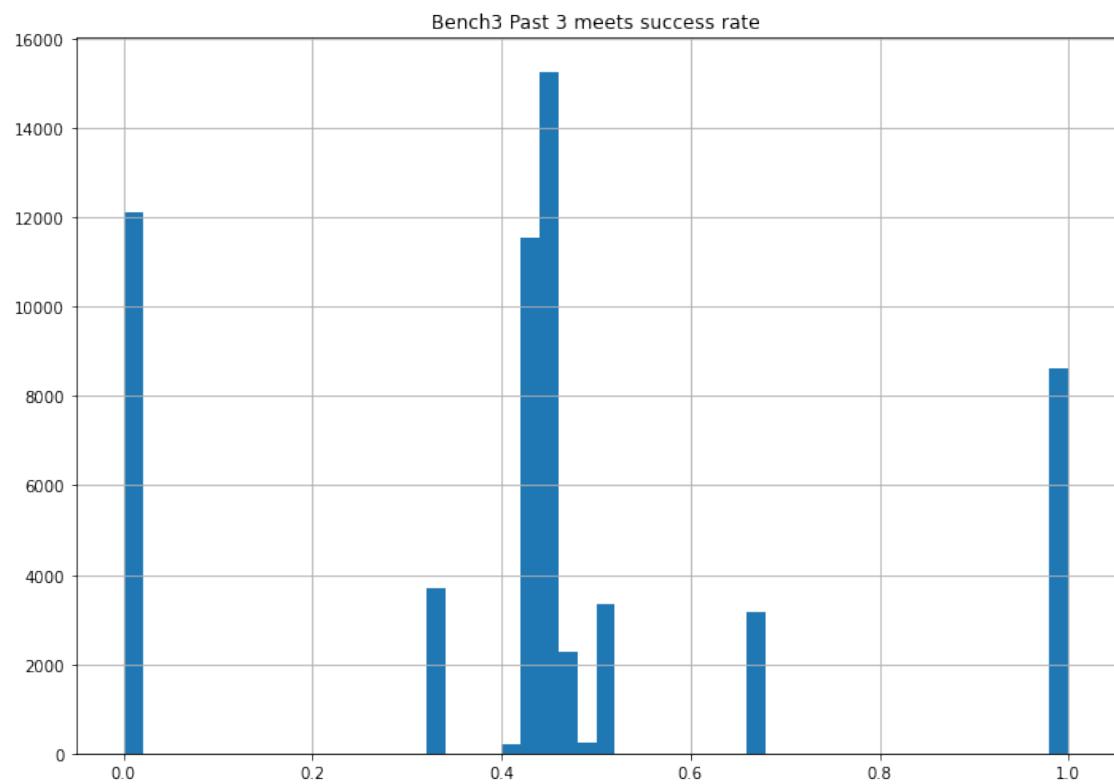
Squat3 Past 3 meets success rate

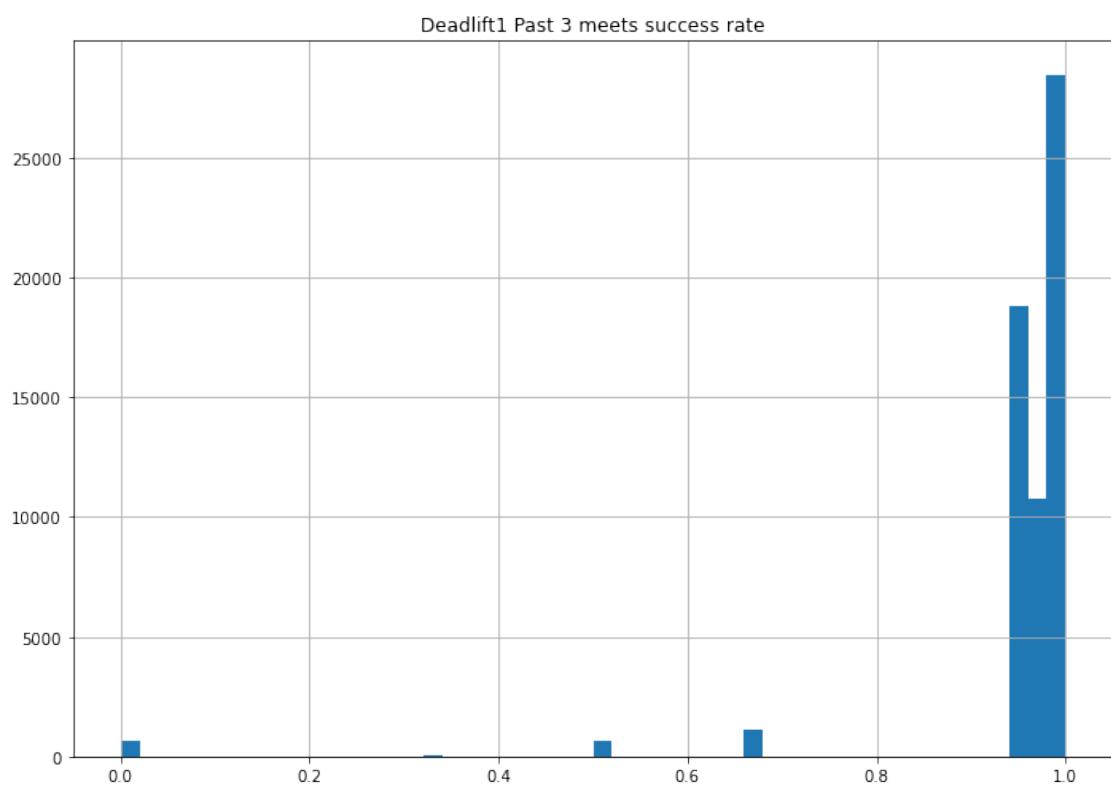
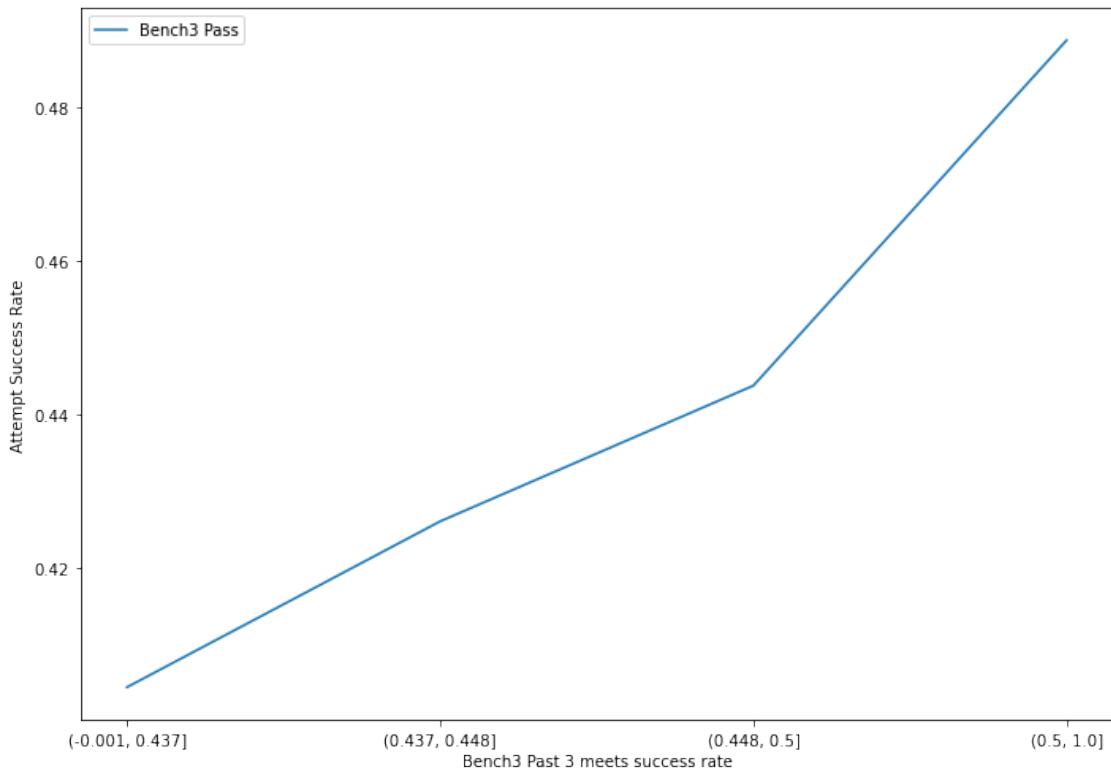


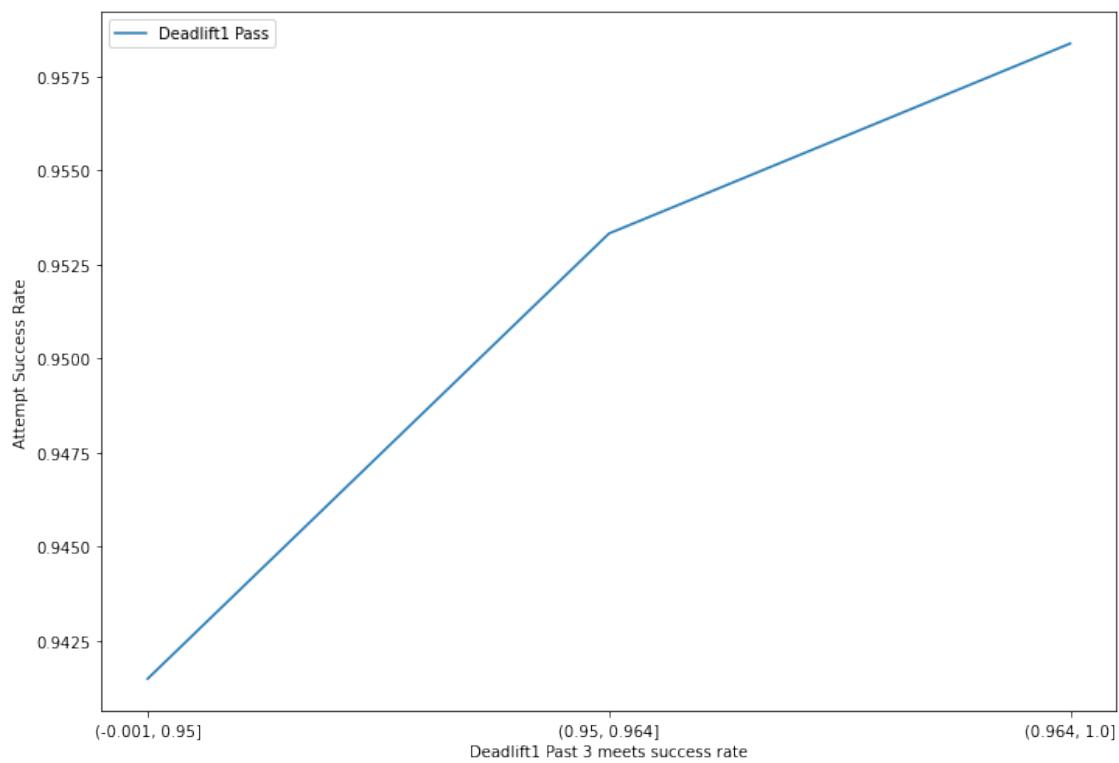


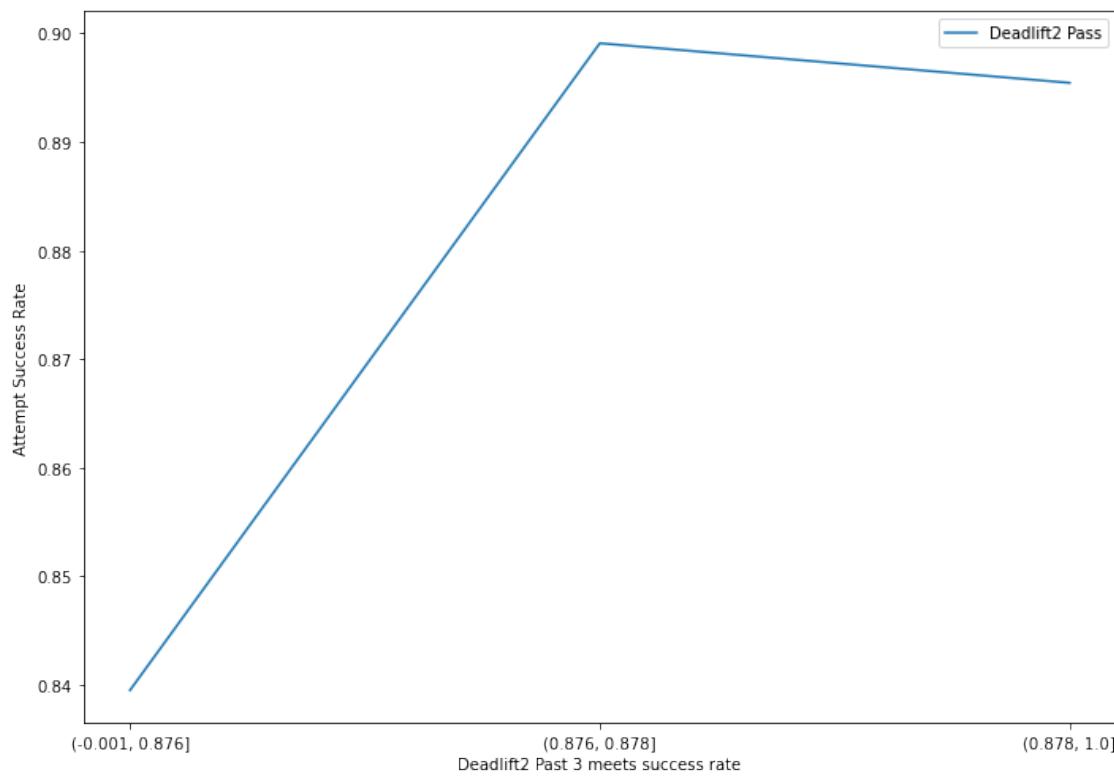
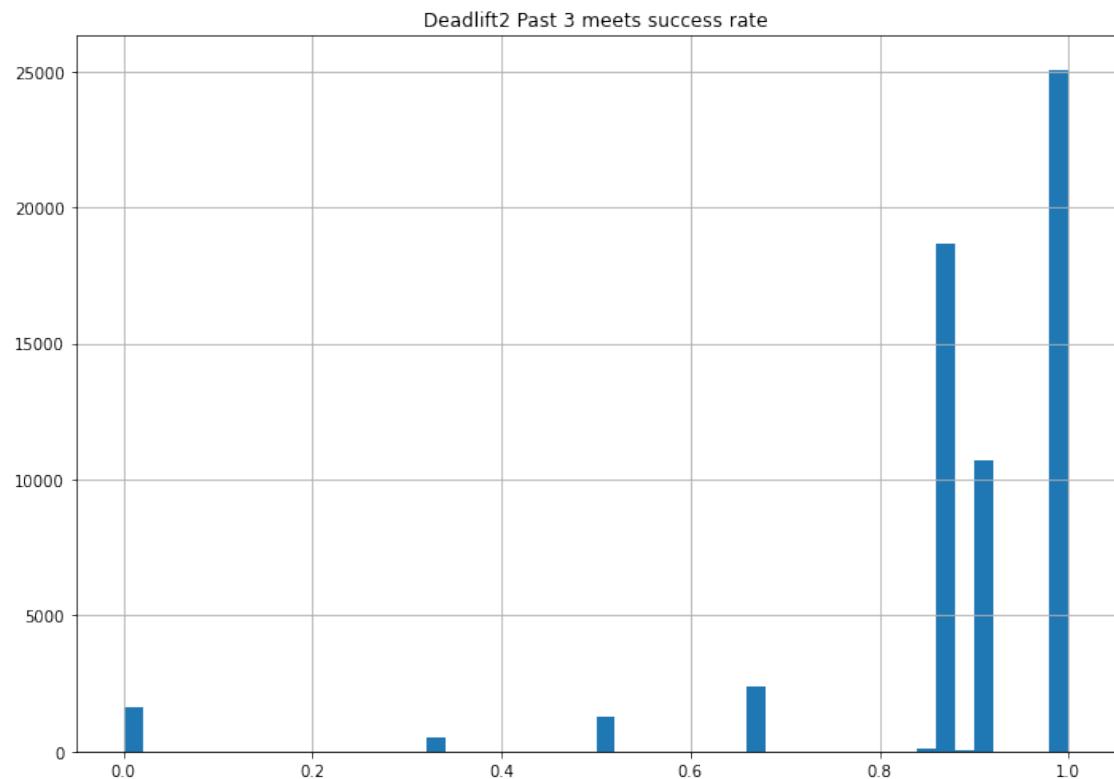




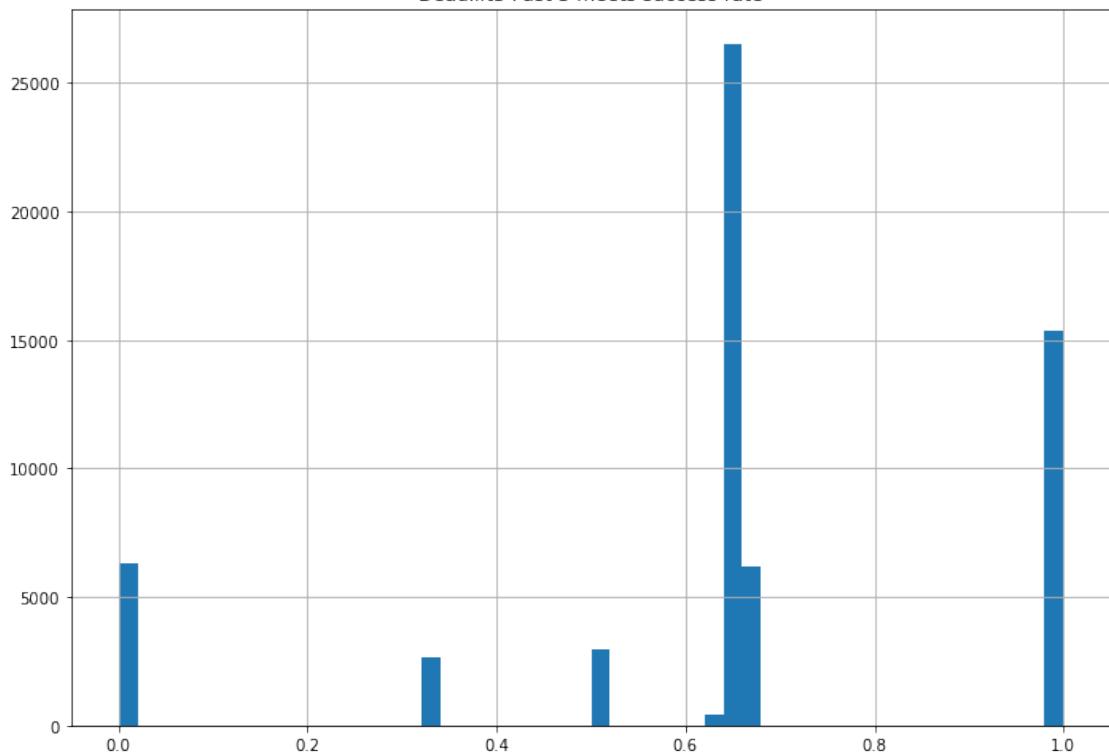


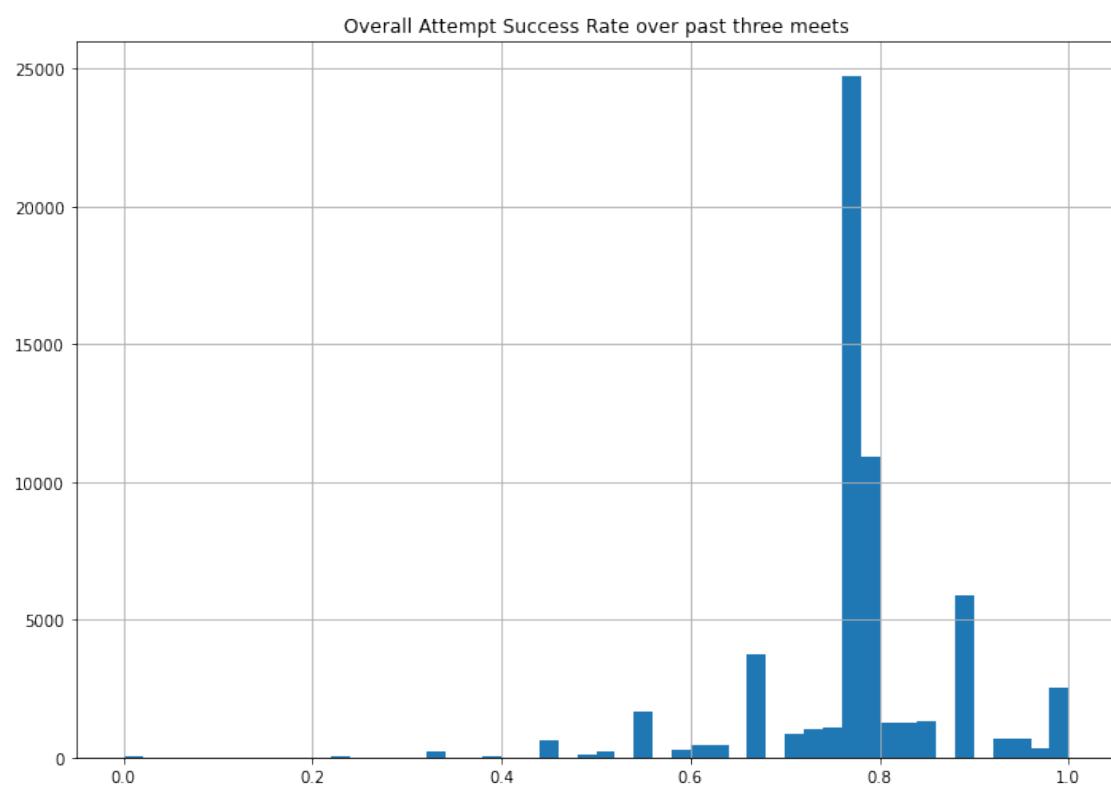
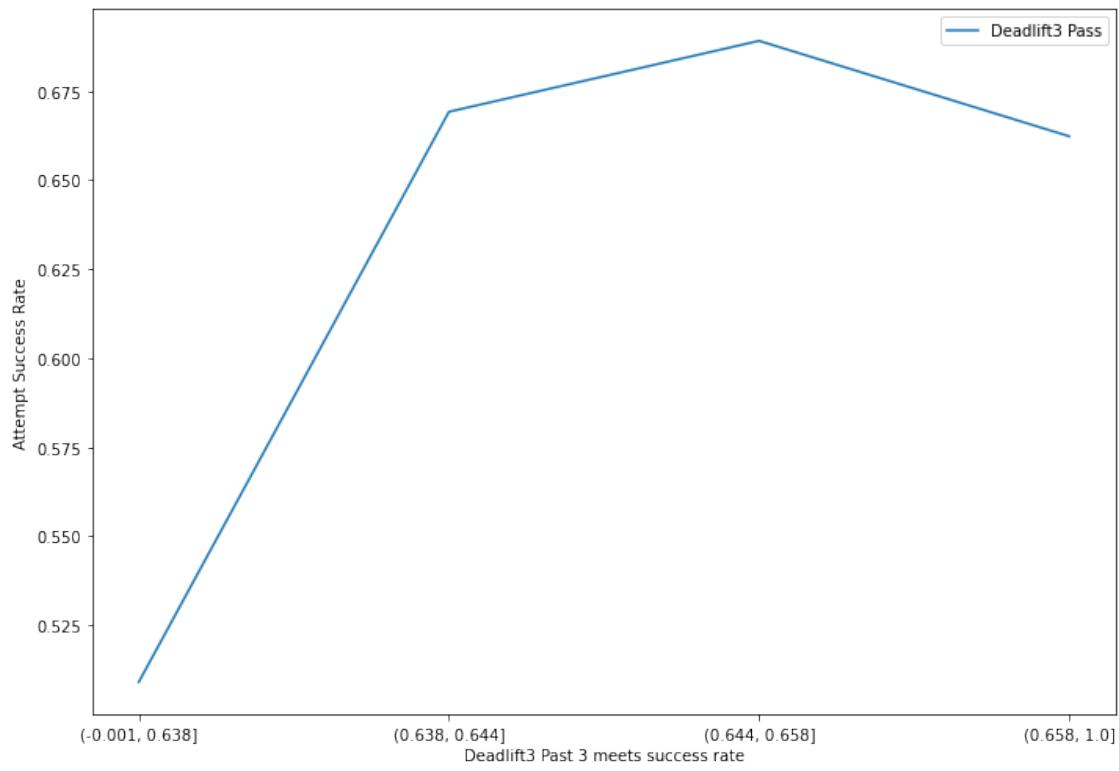


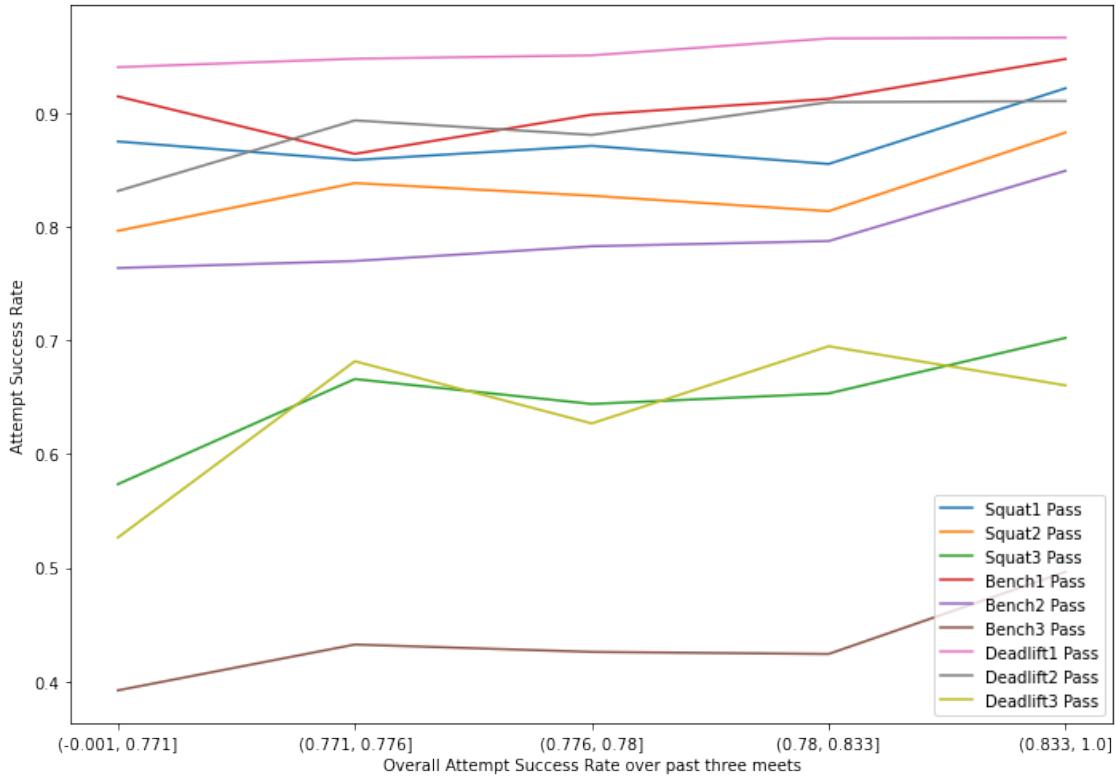




Deadlift3 Past 3 meets success rate







Test train valid splits. Eliminating “first meets” because there will be no past meet success rate data included.

```
[4]: train = openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] < pd.to_datetime("07/01/2019")) & (openpowerlifting_USAPL_lifters['Number of Past Meets Same Day Exclusive'] > 0) & (openpowerlifting_USAPL_lifters['Number of Past Meets'] > 0)]
valid = openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] >= pd.to_datetime("07/01/2019") & (openpowerlifting_USAPL_lifters["Date"] < pd.to_datetime("11/15/2019")) & (openpowerlifting_USAPL_lifters['Number of Past Meets Same Day Exclusive'] > 0) & (openpowerlifting_USAPL_lifters['Number of Past Meets'] > 0)]
test = openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] >= pd.to_datetime("11/15/2019") & (openpowerlifting_USAPL_lifters["Date"] < pd.to_datetime("04/01/2020")) & (openpowerlifting_USAPL_lifters['Number of Past Meets Same Day Exclusive'] > 0) & (openpowerlifting_USAPL_lifters['Number of Past Meets'] > 0)]
```

```

train_and_valid = openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] < pd.to_datetime("11/15/2019")) & (openpowerlifting_USAPL_lifters['Number of Past Meets Same Day Exclusive'] > 0) & (openpowerlifting_USAPL_lifters['Number of Past Meets'] > 0)]
print(train.shape[0], valid.shape[0], test.shape[0])

train_x, valid_x, train_y, valid_y = train[predicting_columns], valid[predicting_columns], train["Deadlift3 Pass"], valid["Deadlift3 Pass"]
train_and_valid_x, train_and_valid_y = train_and_valid[predicting_columns], train_and_valid["Deadlift3 Pass"]
test_x, test_y = test[predicting_columns], test["Deadlift3 Pass"]

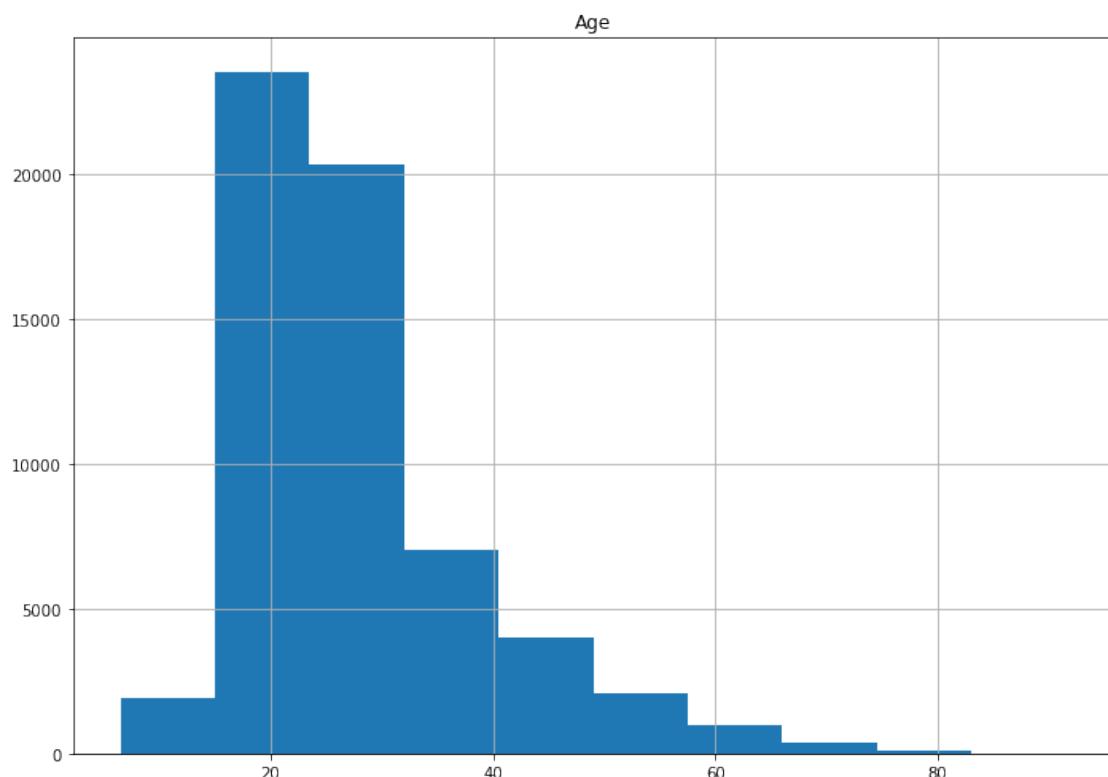
```

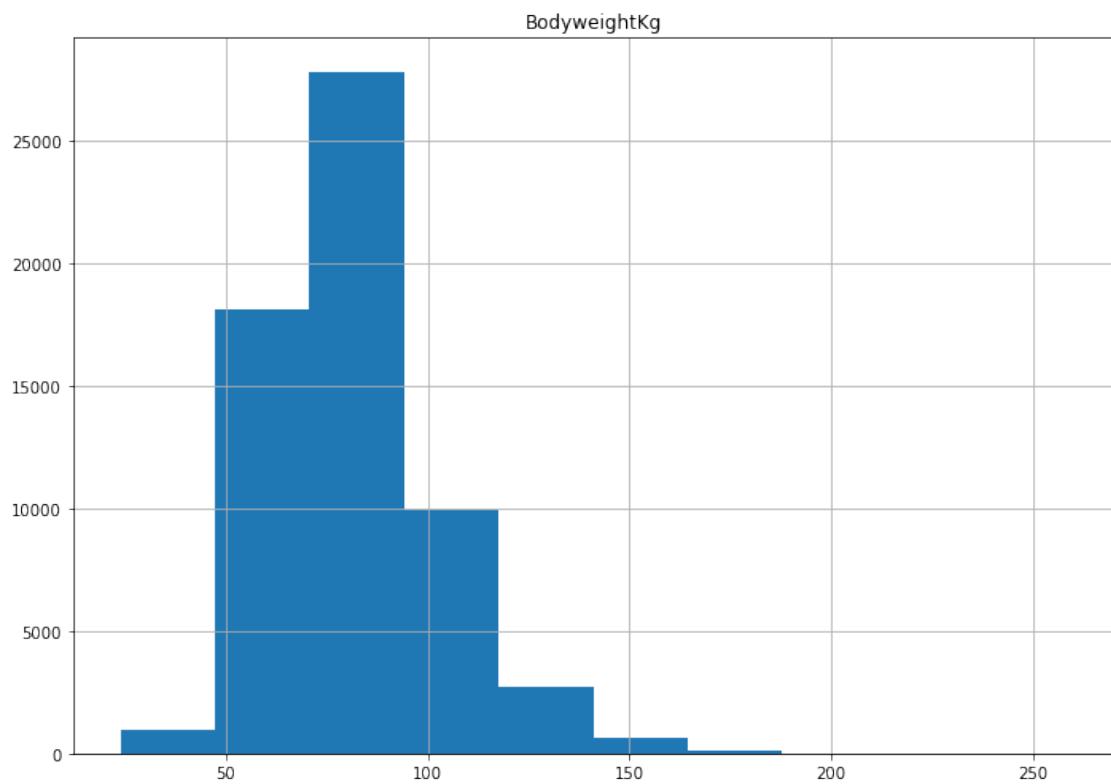
36000 4390 3466

[47]: len(train["Male"].unique())

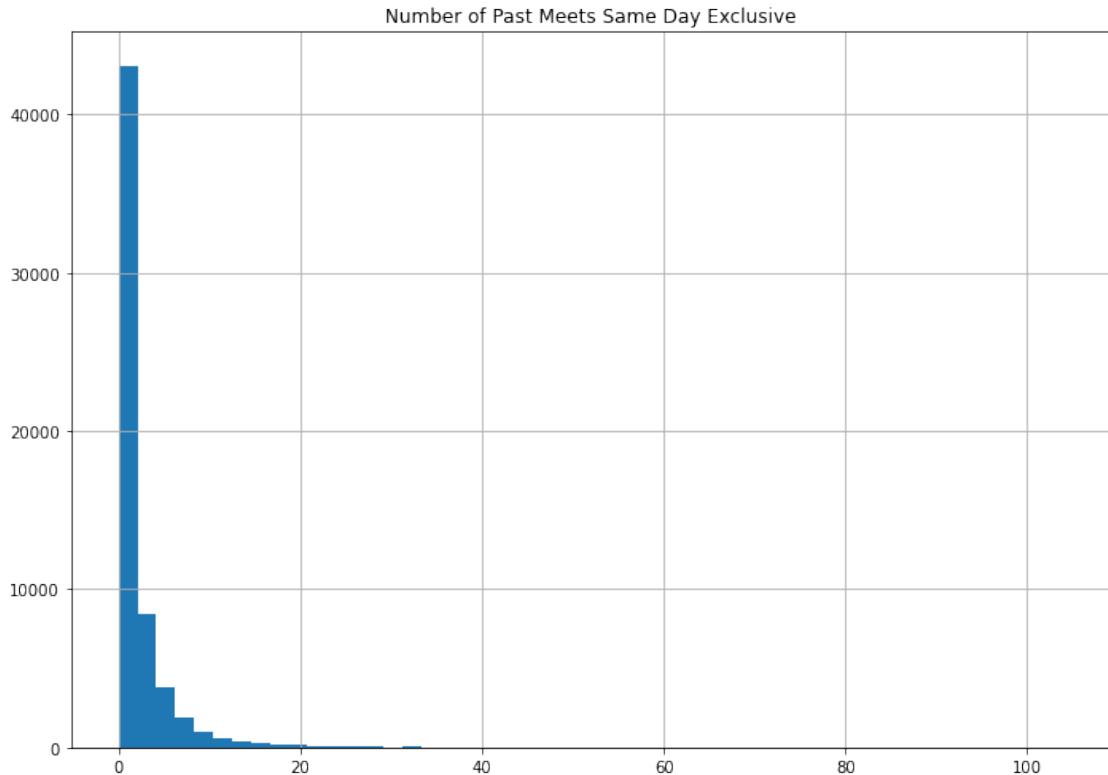
[47]: 2

[15]: train.hist(column = "Age", bins = 10)
plt.show()
train.hist(column = "BodyweightKg", bins = 10)
plt.show()
train.hist(column = "Number of Past Meets Same Day Exclusive", bins = 50)
plt.show()





```
[15]: array([[<AxesSubplot:title={'center':'Number of Past Meets Same Day Exclusive'}>]] ,  
           dtype=object)
```



```
[5]: def graphing_function(metric):
    plt.figure()
    bins = pd.cut(train[metric], [train[metric].sort_values().reset_index(drop=True)[i] for i in range(0, len(train[metric].sort_values().reset_index(drop=True)), len(train[metric].sort_values().reset_index(drop=True))//5)] + [train[metric].max()], duplicates = "drop")
    plot = sns.lineplot(y = (train.groupby(bins)["Deadlift3 Pass"].sum() / train.groupby(bins)["Deadlift3 Pass"].count()).values, x = [str(i) for i in (train.groupby(bins)["Deadlift3 Pass"].sum() / train.groupby(bins)["Deadlift3 Pass"].count()).index])
    plot.set(xlabel = metric, ylabel = "Average chance of hitting Deadlift3")
    print(plot)
    print(train.groupby(bins).size())

for i in ["Age", "BodyweightKg", "Time Since Last Meet", "Number of Past Meets", "Number of Past Meets Same Day Exclusive", "Total Squat Jump", 'Total Bench Jump', 'Deadlift3 Jump', 'Total Deadlift Jump', "Deadlift3Kg", "Deadlift3 Past 3 meets success rate", "Overall Attempt Success Rate over past three meets", "Best Raw Wilks to date", "Number of meets in past 12 months"]:
    graphing_function(i)
```

```

AxesSubplot(0.125,0.125;0.775x0.755)
Age
(7.5, 19.5]      7658
(19.5, 23.5]    8339
(23.5, 27.5]    6399
(27.5, 35.5]    6798
(35.5, 91.5]    6804
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
BodyweightKg
(25.2, 62.9]    7245
(62.9, 73.4]    7202
(73.4, 83.0]    7305
(83.0, 98.7]    7054
(98.7, 233.0]   7193
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Time Since Last Meet
(0, 83]          7230
(83, 133]        7263
(133, 191]       7148
(191, 309]       7161
(309, 13496]    7191
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Number of Past Meets
(1, 2]           7517
(2, 3]           4815
(3, 5]           5324
(5, 84]          6096
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Number of Past Meets Same Day Exclusive
(1.0, 2.0]       7516
(2.0, 3.0]       4817
(3.0, 5.0]       5325
(5.0, 80.0]      6091
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Total Squat Jump
(0.0, 0.0714]   6868
(0.0714, 0.0938] 7087
(0.0938, 0.114]  7239
(0.114, 0.143]   7086
(0.143, 6.7]     7179
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Total Bench Jump

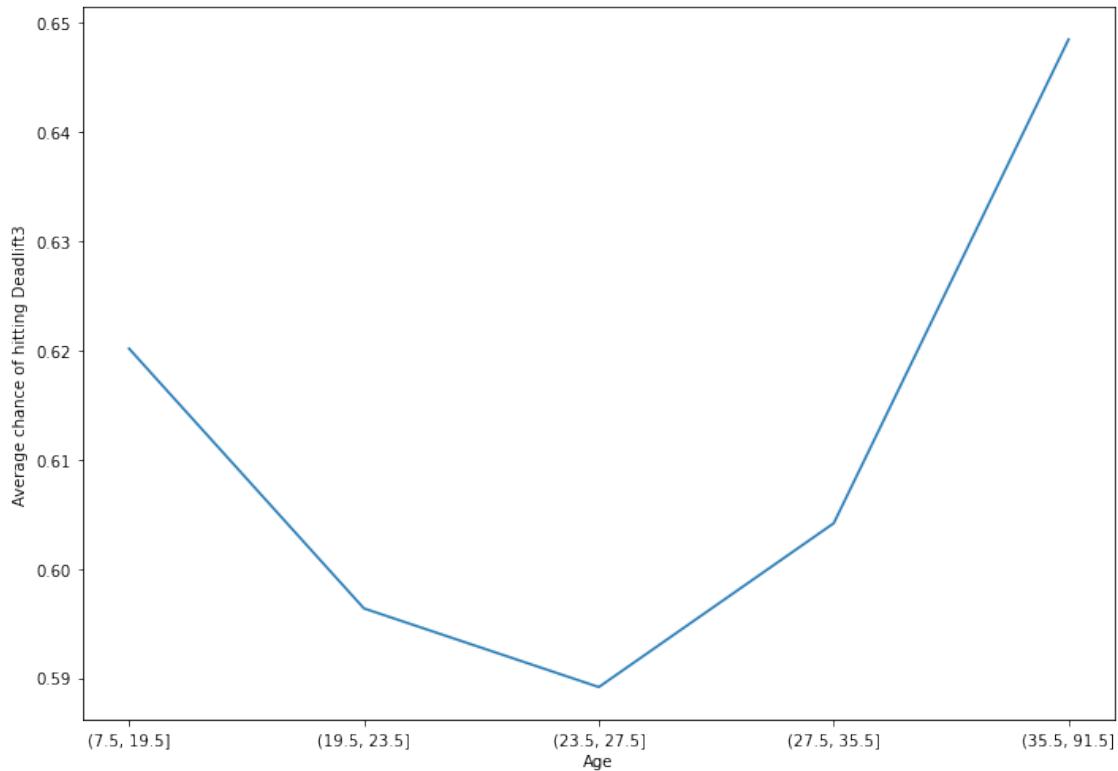
```

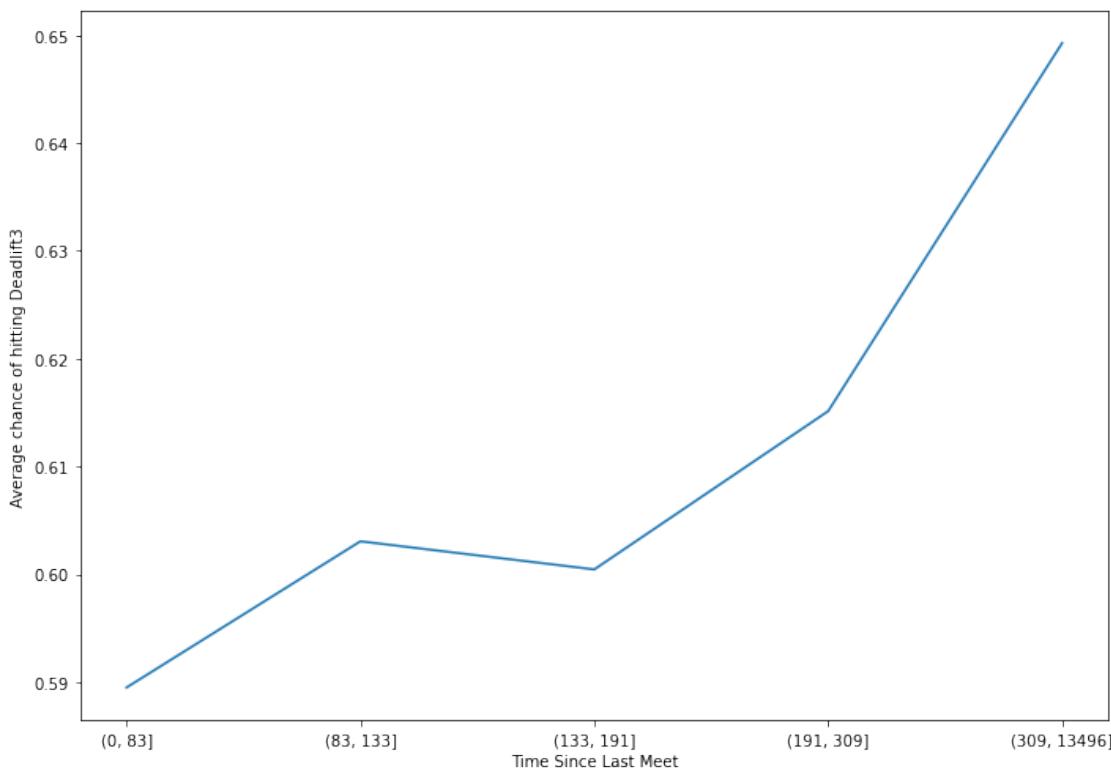
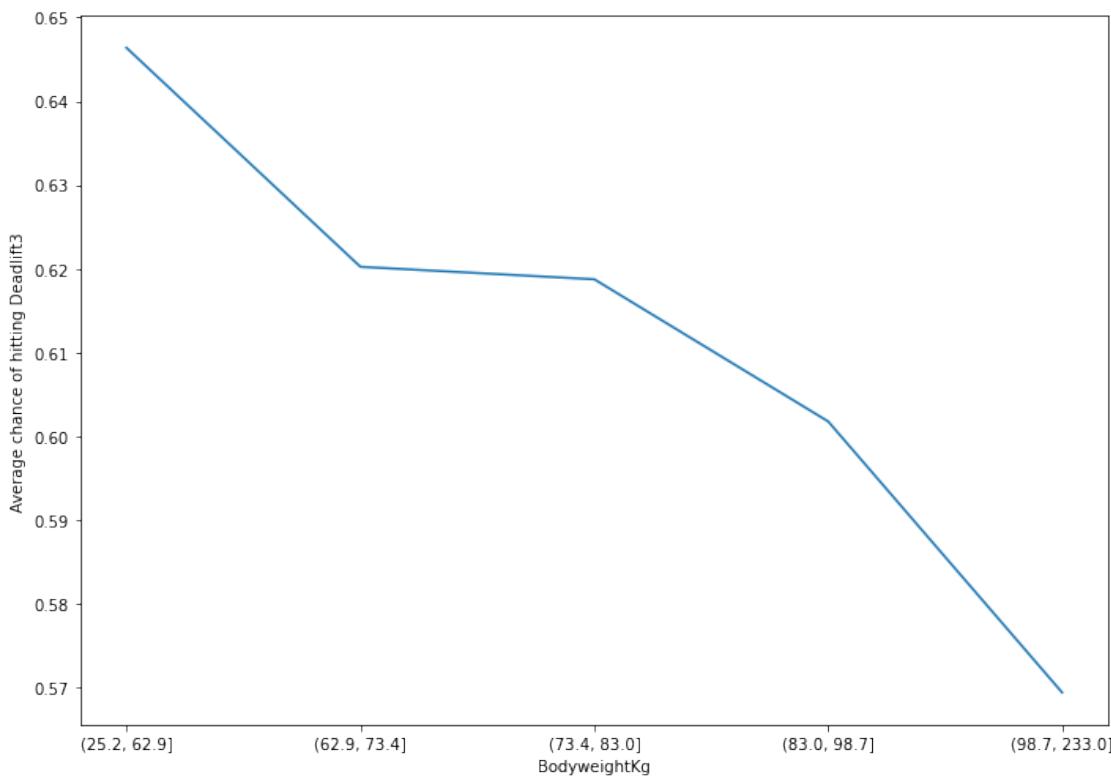
```

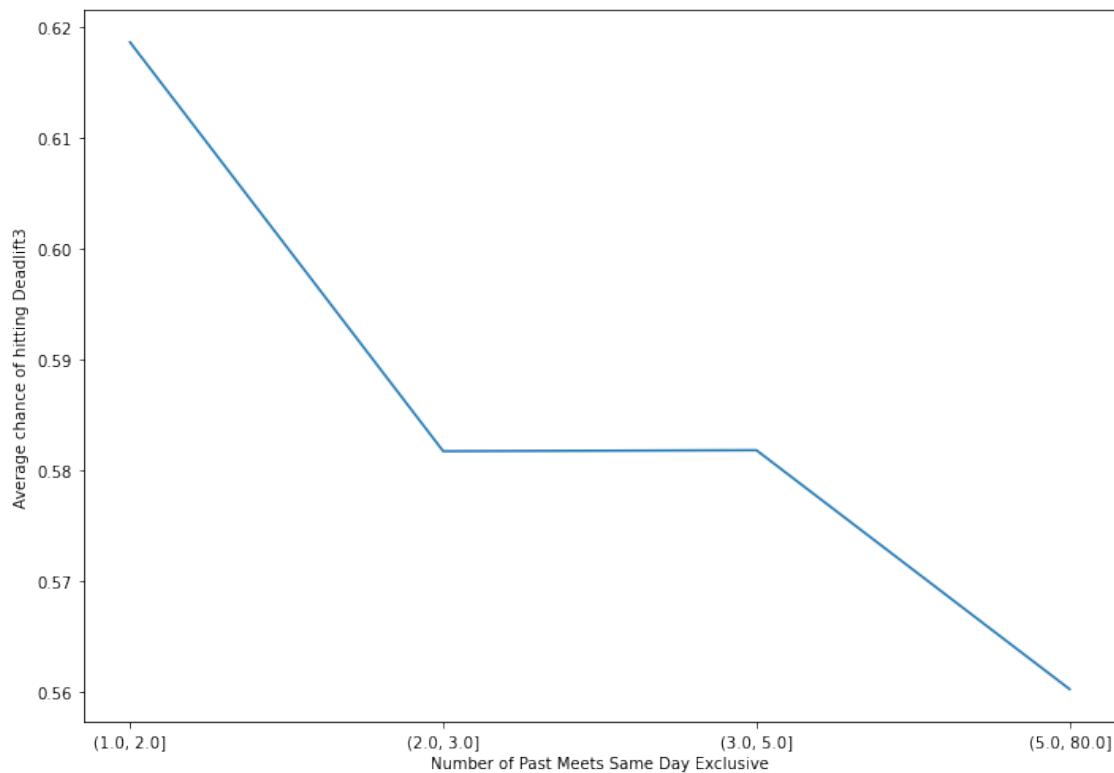
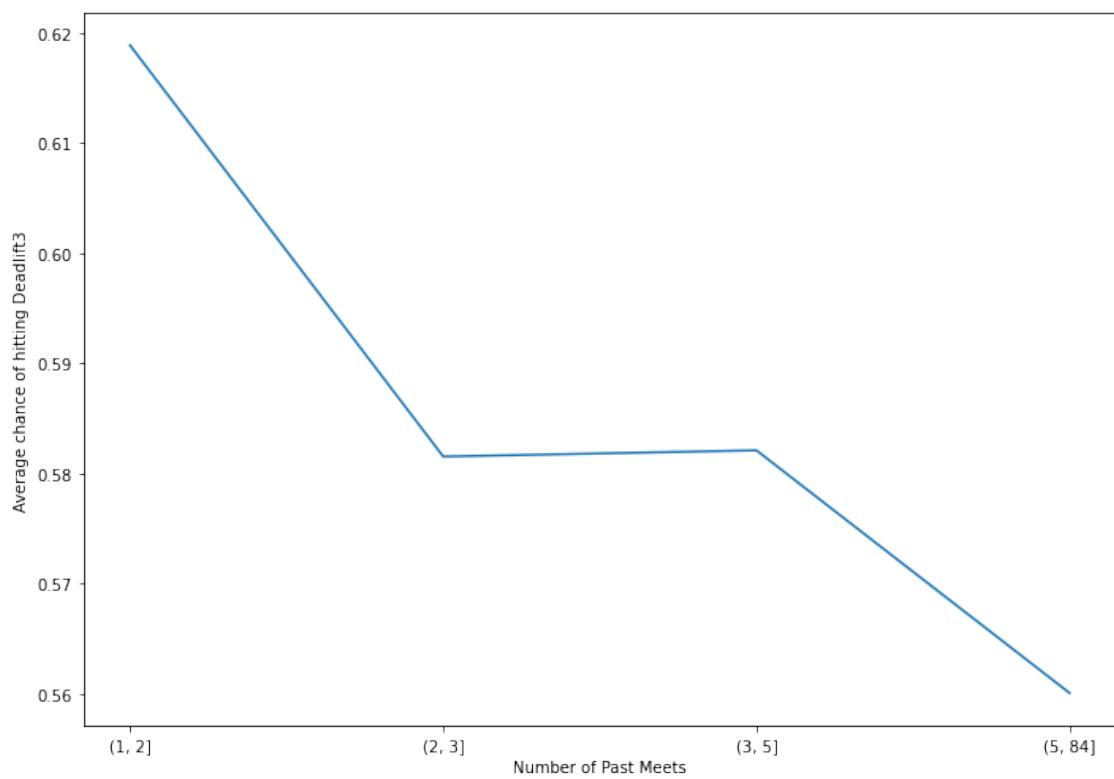
(0.0, 0.0714]      7142
(0.0714, 0.0943]  6890
(0.0943, 0.118]   7464
(0.118, 0.154]    7253
(0.154, 3.2]      6874
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Deadlift3 Jump
(0.0, 0.022]      3952
(0.022, 0.0364]   7204
(0.0364, 0.0484]  7252
(0.0484, 0.0649]  7176
(0.0649, 0.539]   7160
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Total Deadlift Jump
(0.0, 0.075]      7031
(0.075, 0.0978]   7196
(0.0978, 0.118]   7243
(0.118, 0.148]    7169
(0.148, 5.113]    7147
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Deadlift3Kg
(25.0, 137.5]     7806
(137.5, 170.0]    6785
(170.0, 215.0]    7262
(215.0, 250.0]    7383
(250.0, 400.0]    6761
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Deadlift3 Past 3 meets success rate
(0.0, 0.333]      2488
(0.333, 0.645]   6012
(0.645, 1.0]      21599
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Overall Attempt Success Rate over past three meets
(0.0, 0.667]      7336
(0.667, 0.778]   11844
(0.778, 0.784]   2399
(0.784, 0.889]   10344
(0.889, 1.0]     4055
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Best Raw Wilks to date
(0.0, 256.236]    4559
(256.236, 312.158] 7200

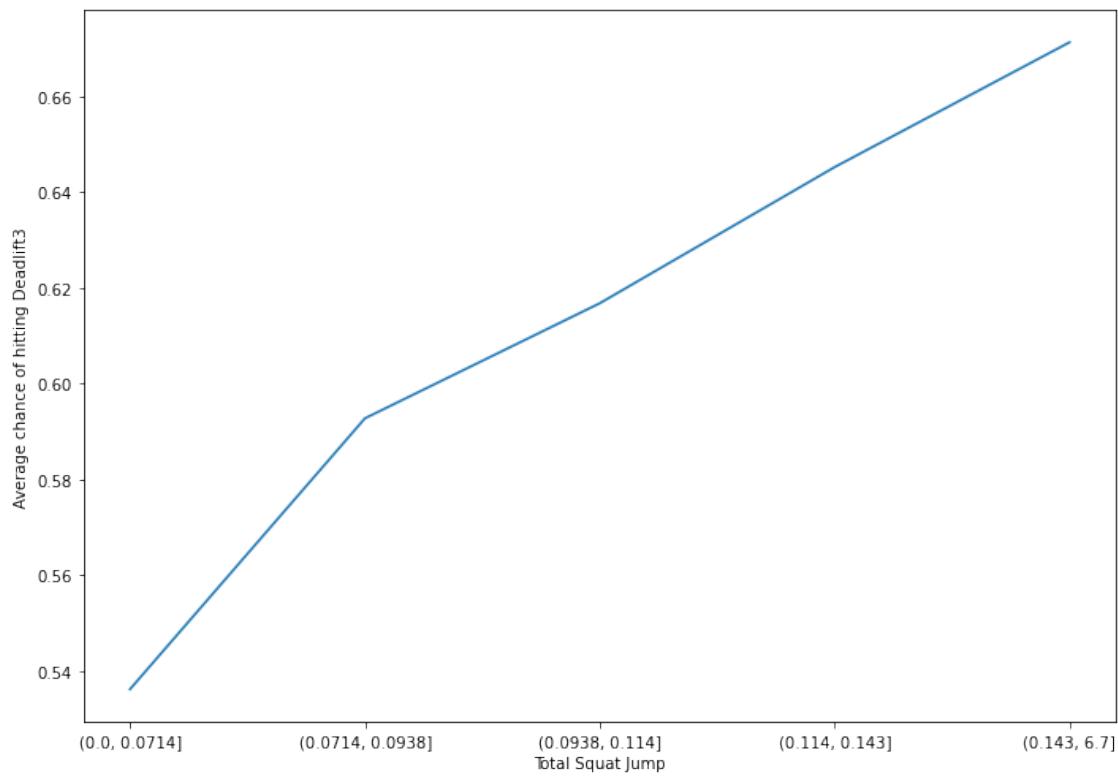
```

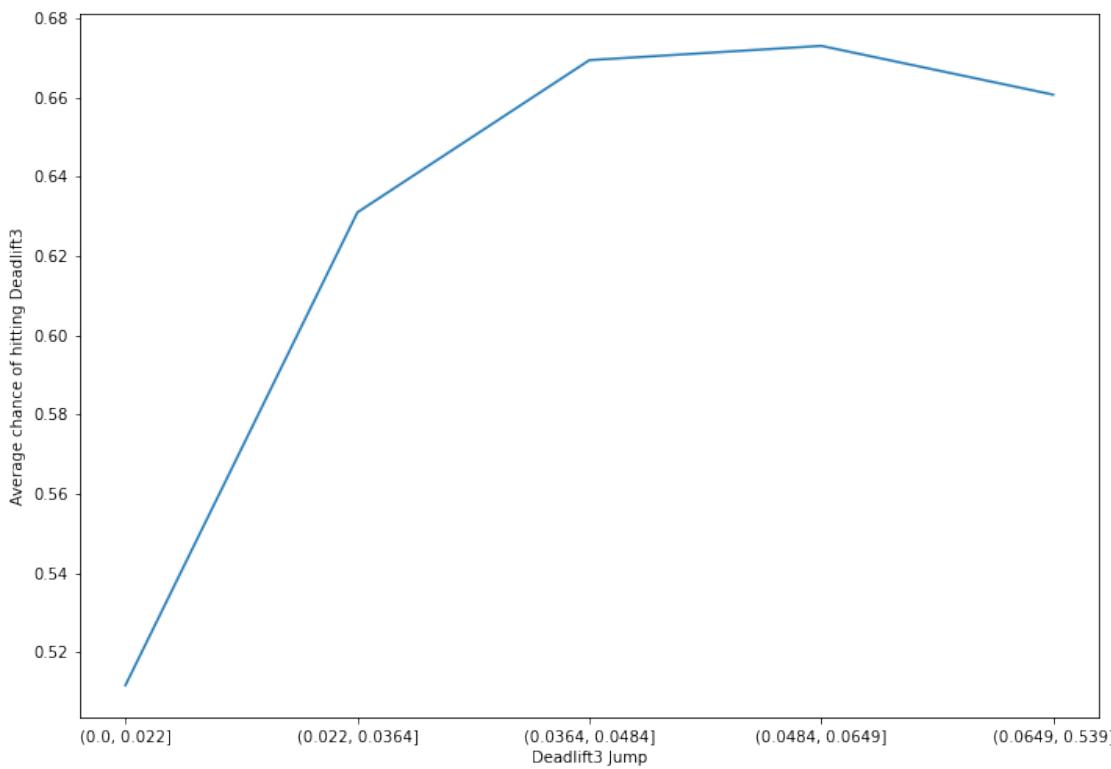
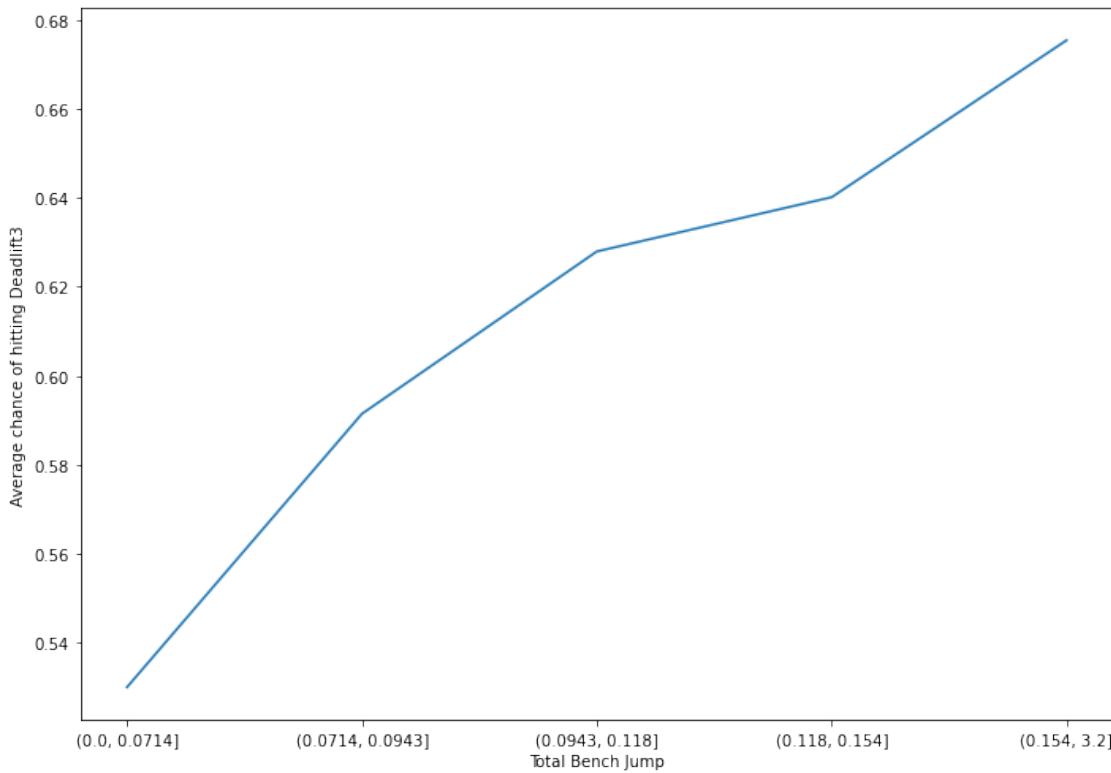
```
(312.158, 350.506]      7200
(350.506, 390.005]      7200
(390.005, 593.513]      7199
dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
Number of meets in past 12 months
(0.0, 1.0]      16921
(1.0, 2.0]      9414
(2.0, 12.0]     4762
dtype: int64
```

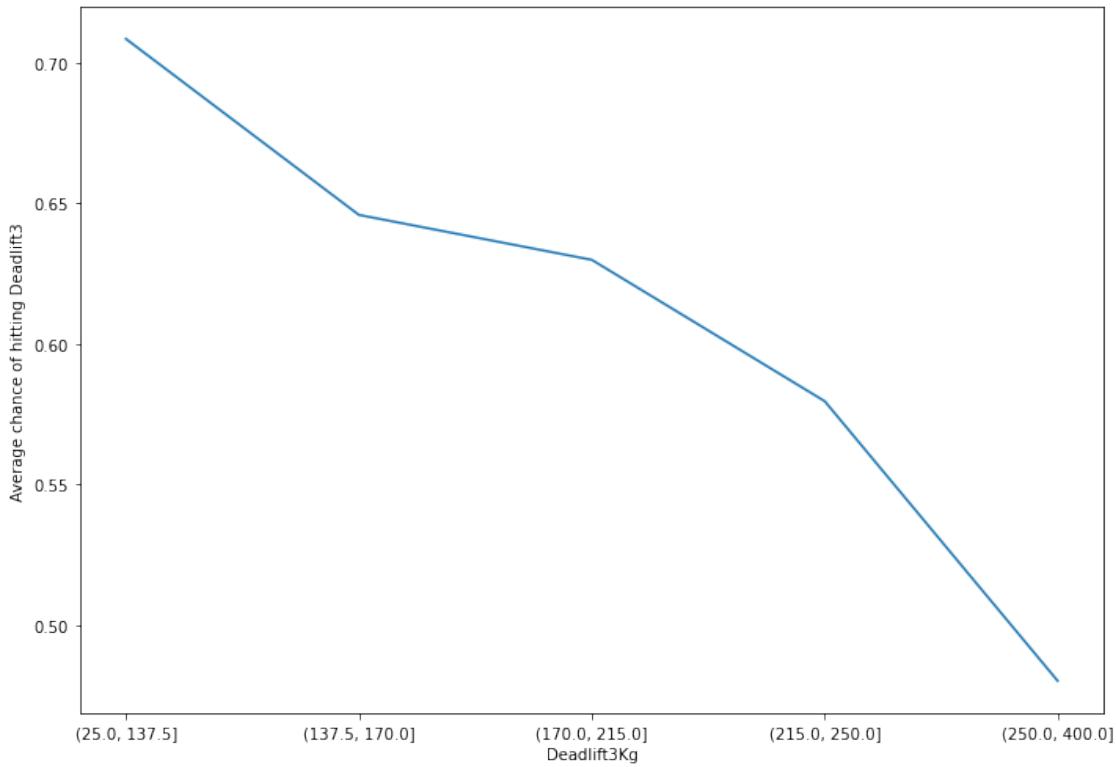
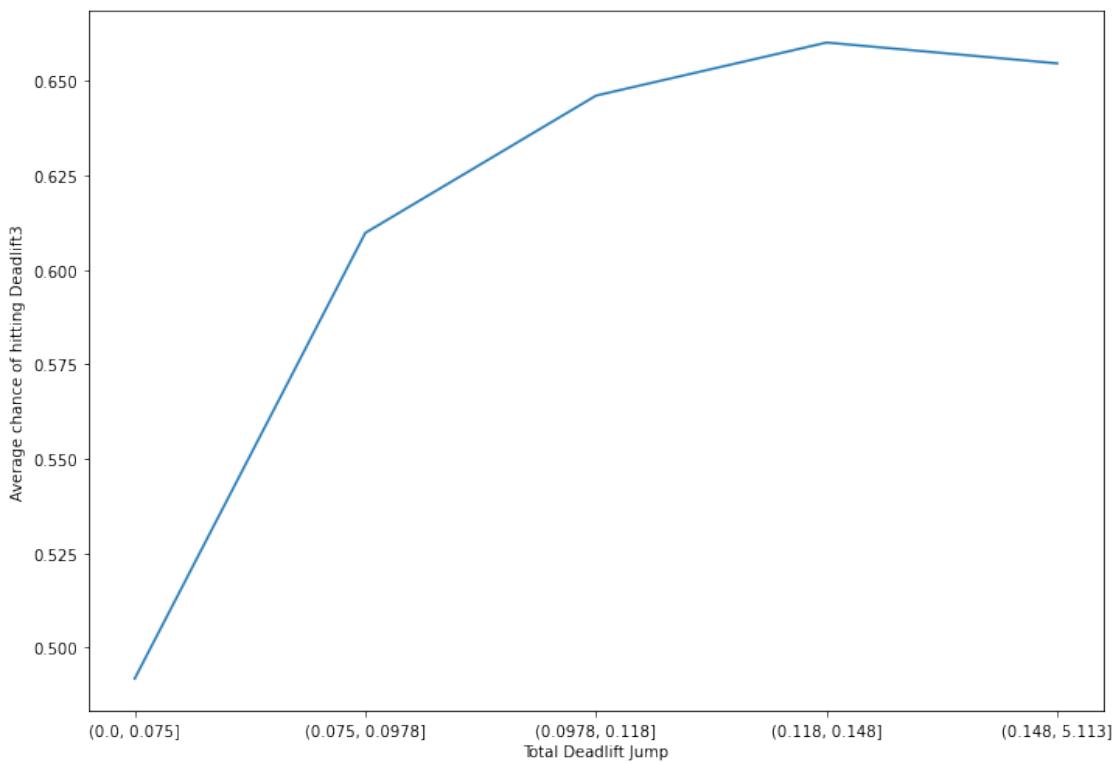


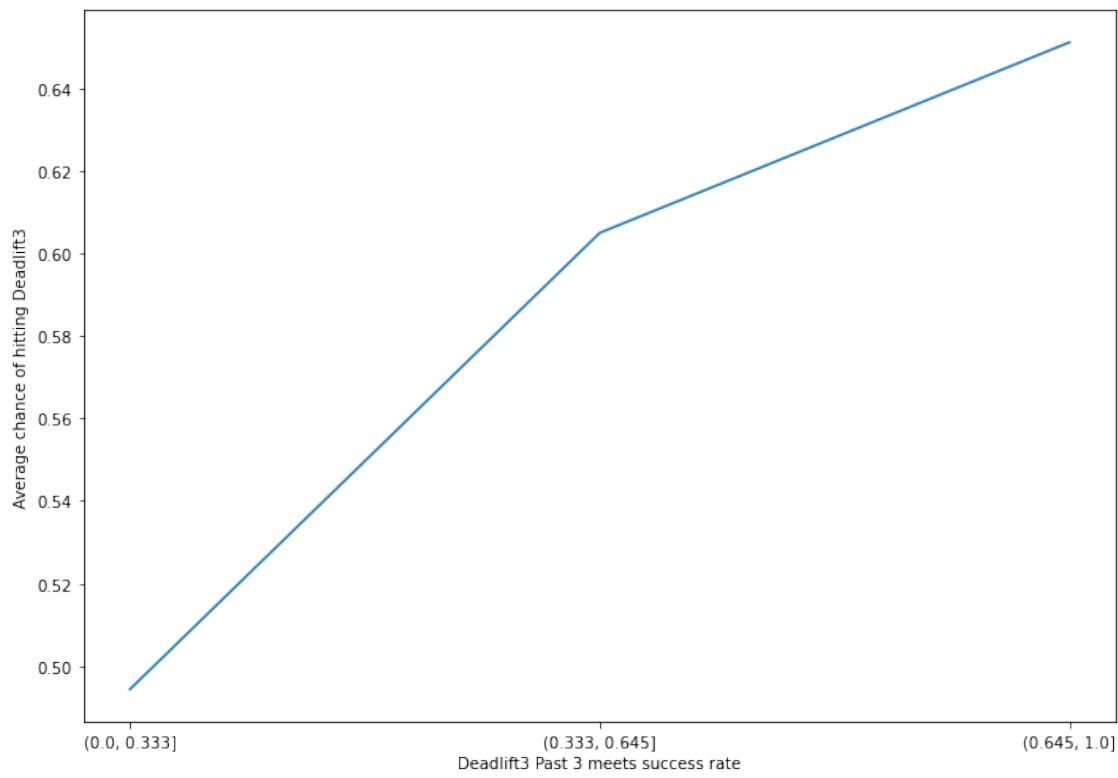


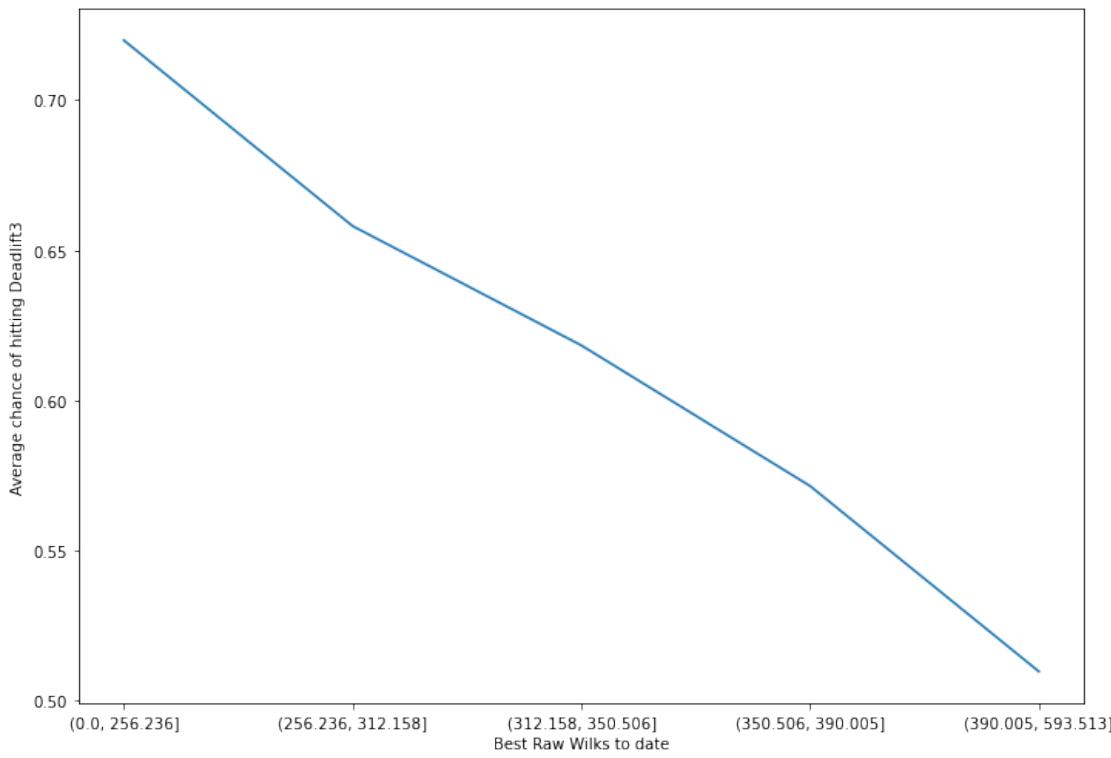
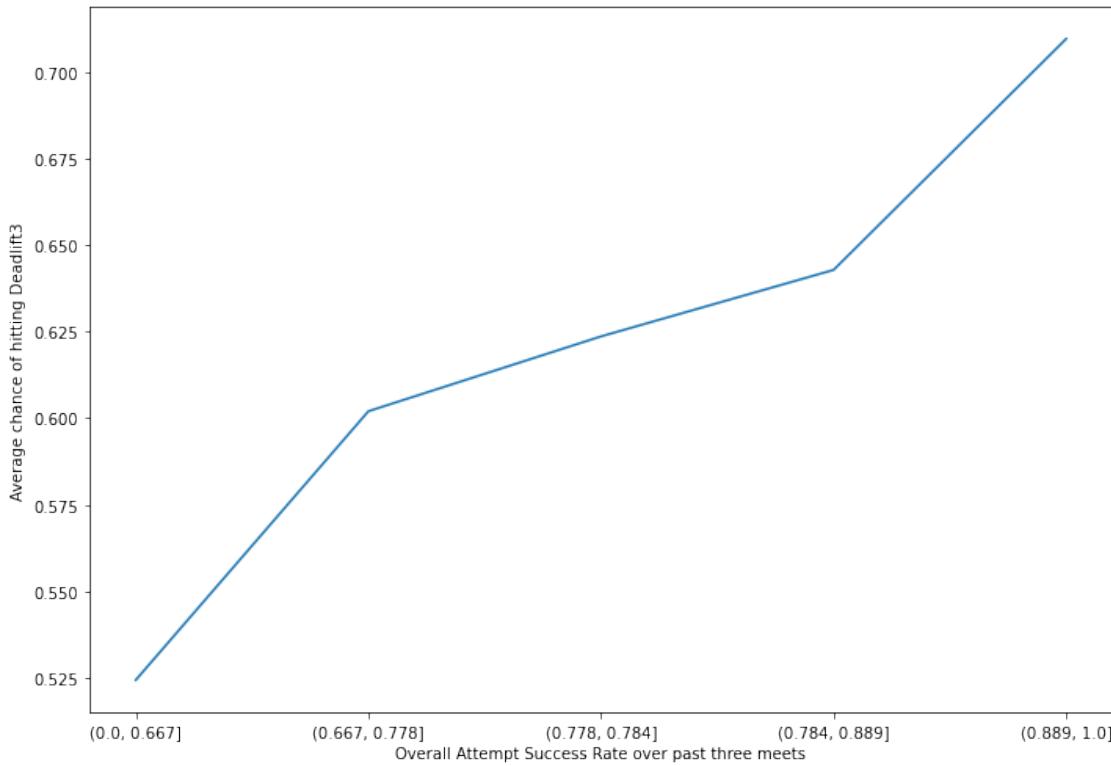


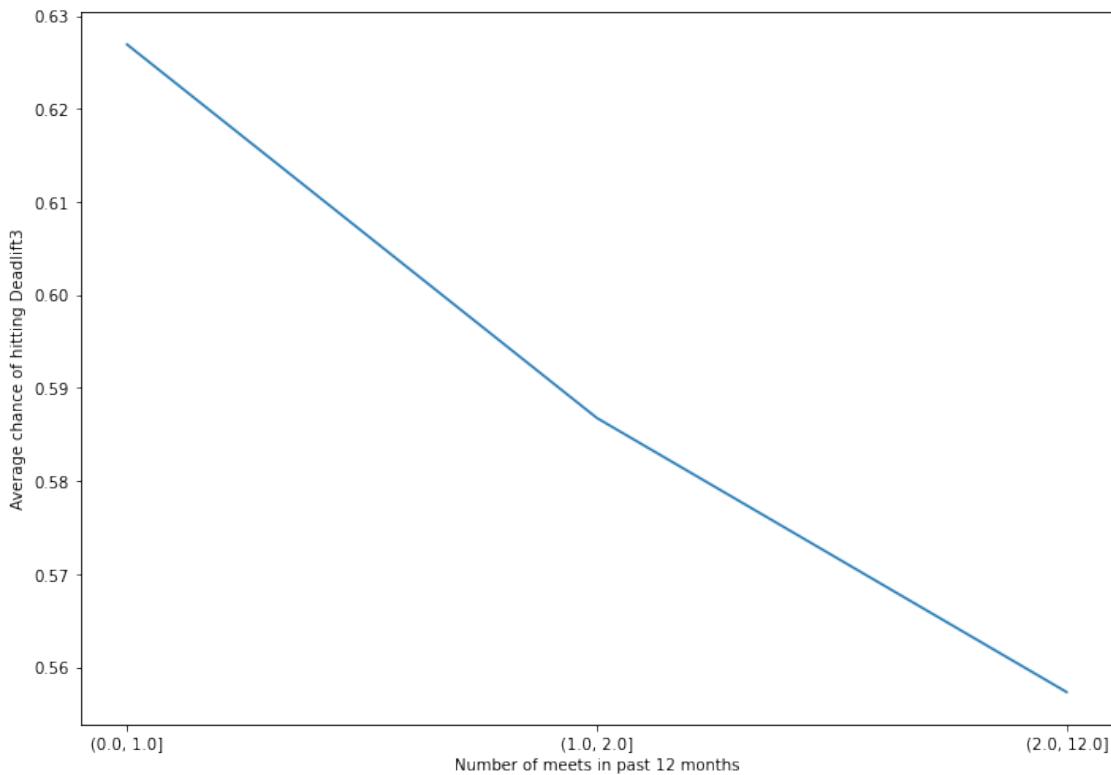












```
[6]: plt.figure()
print(sns.lineplot(data=(train.groupby("Male")["Deadlift3 Pass"].sum() / train.
                     groupby("Male")["Deadlift3 Pass"].count())))

plt.figure()
print(sns.lineplot(data=(train.groupby("Deadlift2 Pass")["Deadlift3 Pass"].
                     sum() / train.groupby("Deadlift2 Pass")["Deadlift3 Pass"].count())))

plt.figure()
print(sns.lineplot(data=(train.groupby("Deadlift1 Pass")["Deadlift3 Pass"].
                     sum() / train.groupby("Deadlift1 Pass")["Deadlift3 Pass"].count())))

plt.figure()
print(sns.lineplot(data=(train.groupby("Bench3 Pass")["Deadlift3 Pass"].sum() / train.
                     groupby("Bench3 Pass")["Deadlift3 Pass"].count())))

plt.figure()
print(sns.lineplot(data=(train.groupby("Bench2 Pass")["Deadlift3 Pass"].sum() / train.
                     groupby("Bench2 Pass")["Deadlift3 Pass"].count())))
```

```

plt.figure()
print(sns.lineplot(data=(train.groupby("Bench1 Pass")["Deadlift3 Pass"].sum() / 
    train.groupby("Bench1 Pass")["Deadlift3 Pass"].count())))

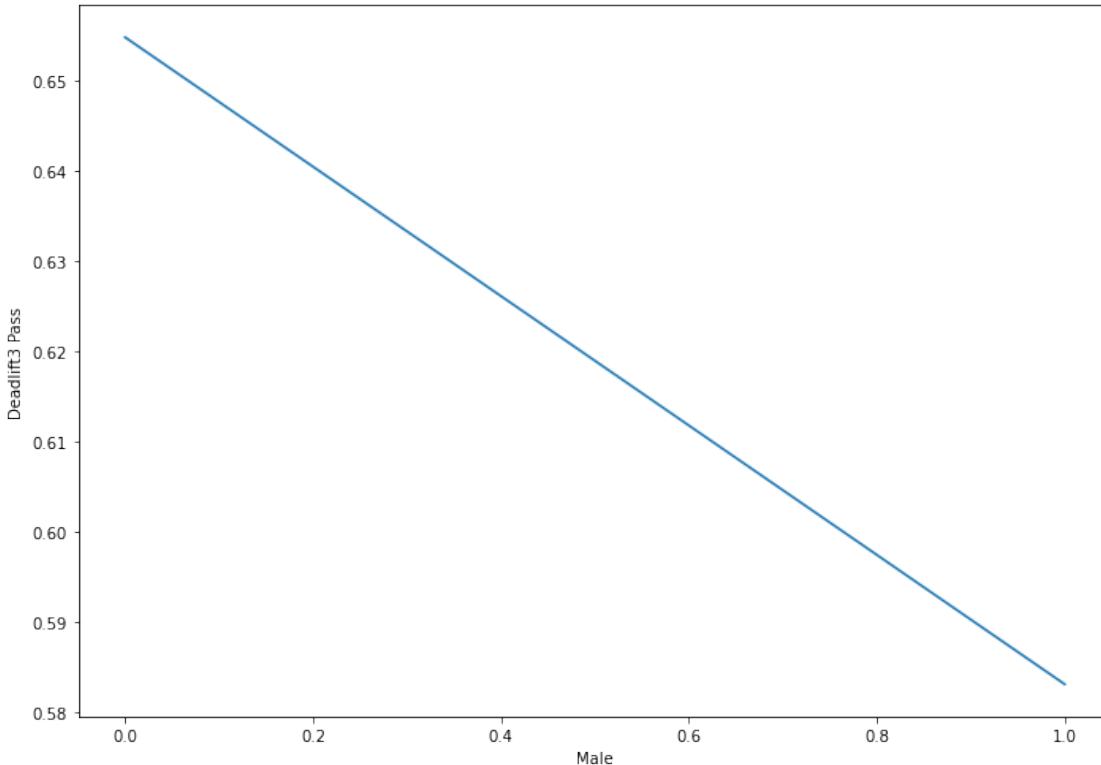
plt.figure()
print(sns.lineplot(data=(train.groupby("Squat1 Pass")["Deadlift3 Pass"].sum() / 
    train.groupby("Squat1 Pass")["Deadlift3 Pass"].count())))

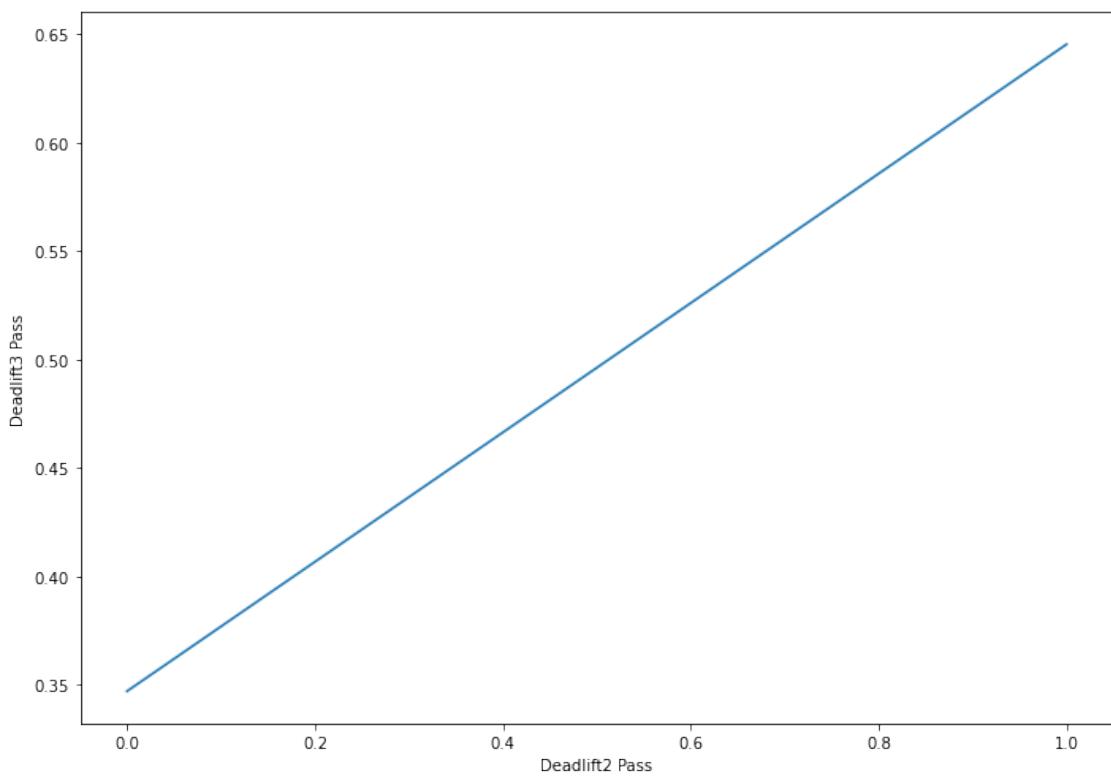
plt.figure()
print(sns.lineplot(data=(train.groupby("Squat2 Pass")["Deadlift3 Pass"].sum() / 
    train.groupby("Squat2 Pass")["Deadlift3 Pass"].count())))

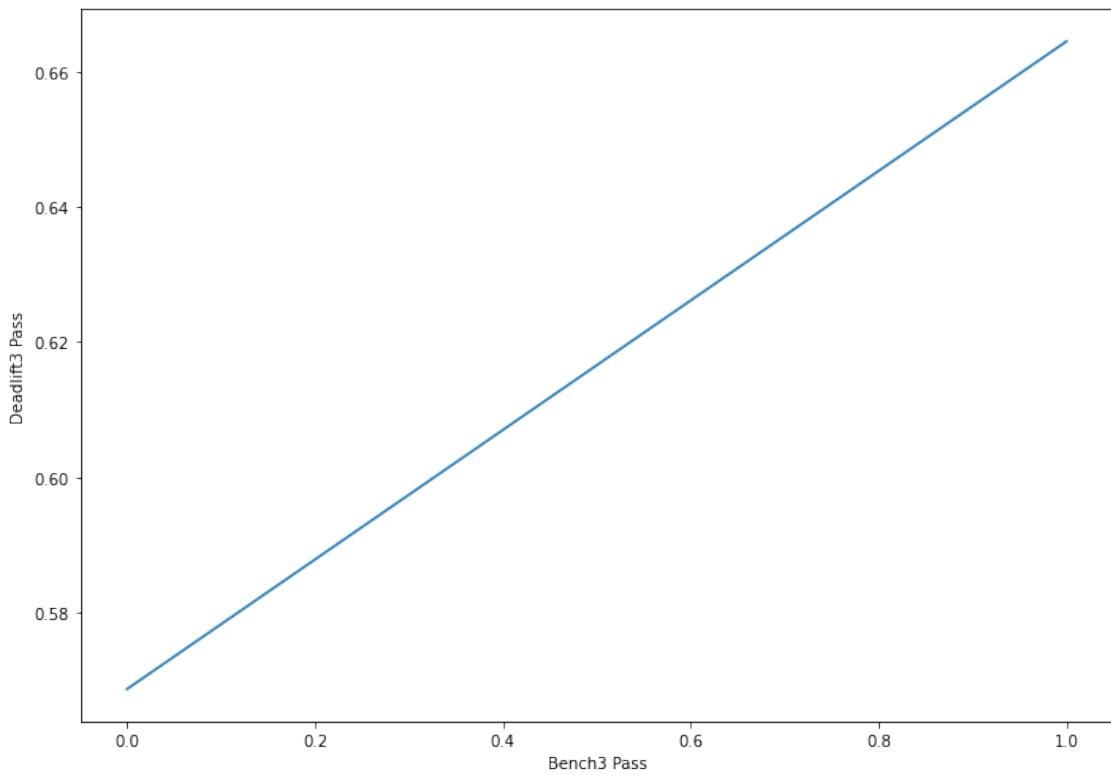
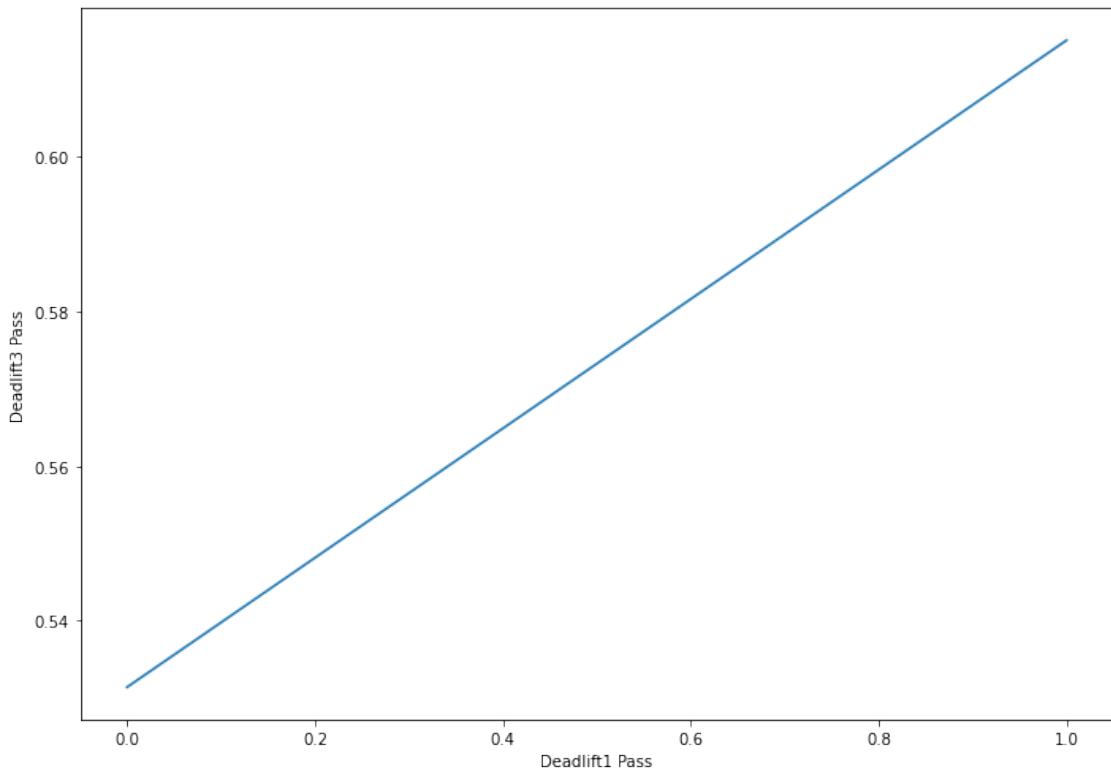
plt.figure()
print(sns.lineplot(data=(train.groupby("Squat3 Pass")["Deadlift3 Pass"].sum() / 
    train.groupby("Squat3 Pass")["Deadlift3 Pass"].count())))

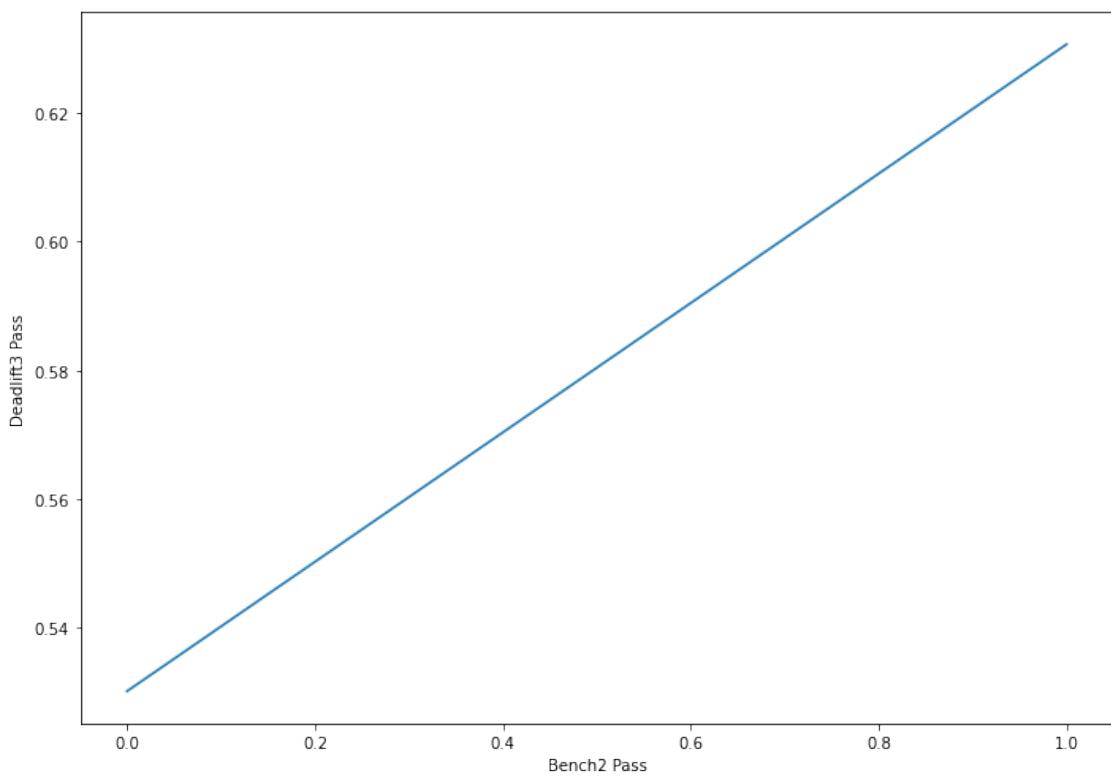
```

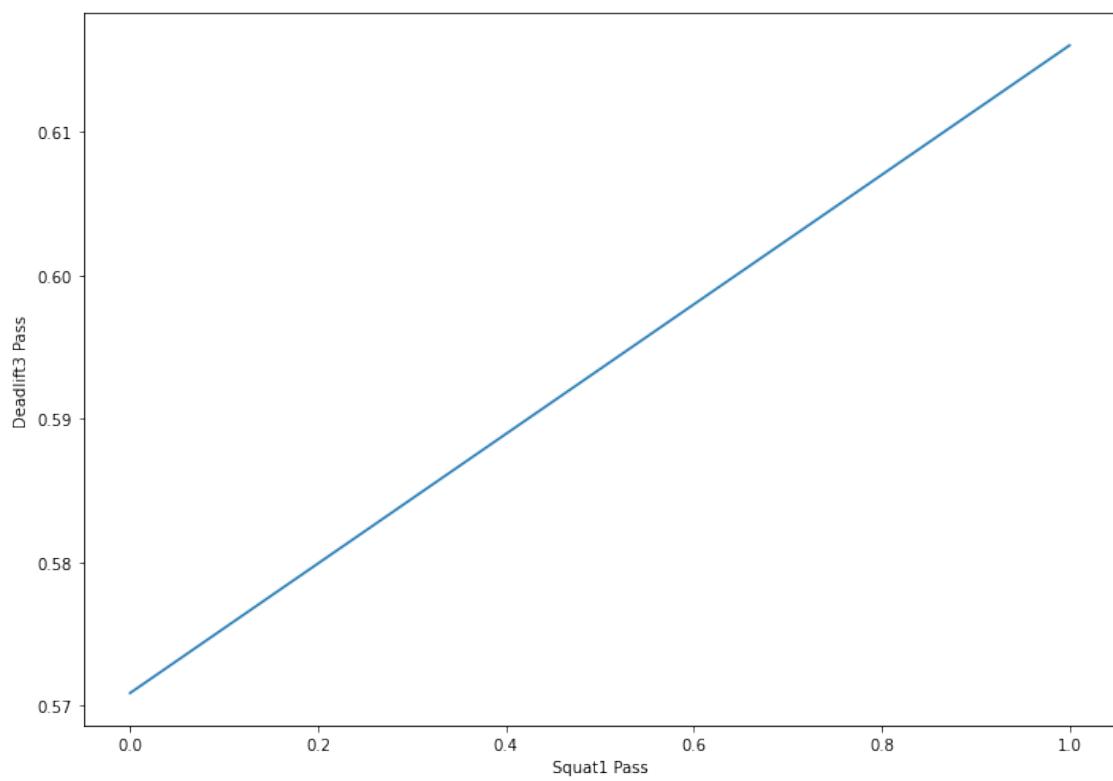
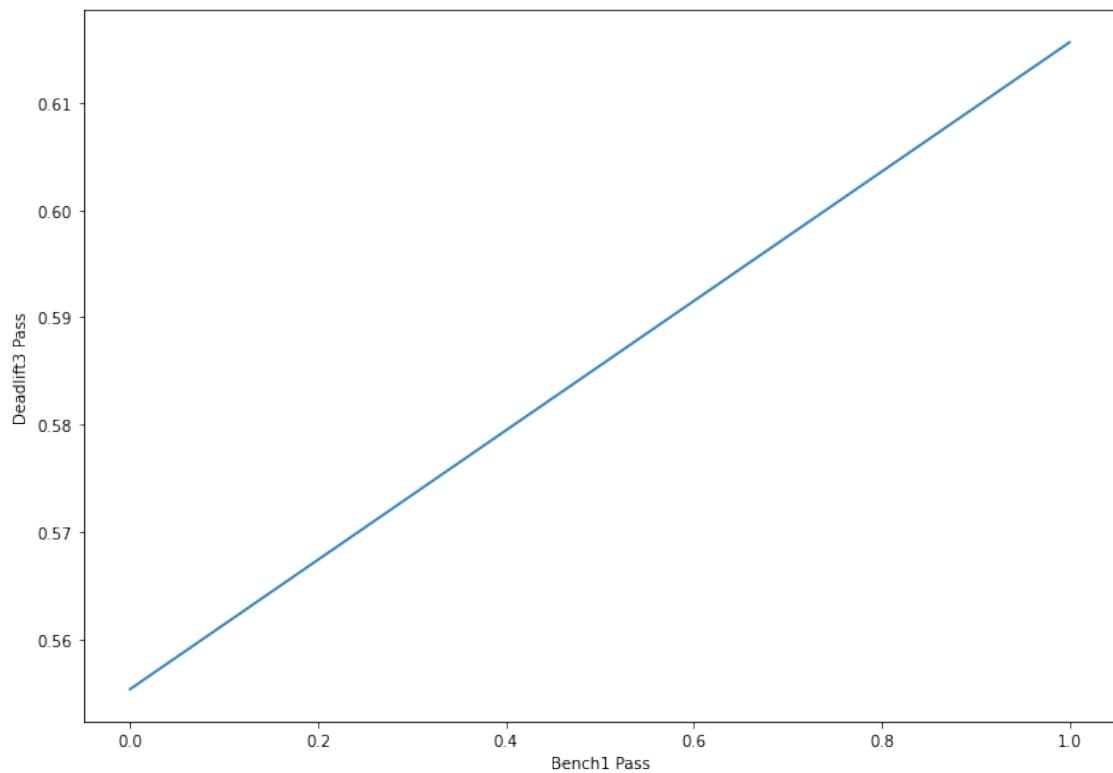
AxesSubplot(0.125,0.125;0.775x0.755)
AgesSubplot(0.125,0.125;0.775x0.755)
AgesSubplot(0.125,0.125;0.775x0.755)
AgesSubplot(0.125,0.125;0.775x0.755)
AgesSubplot(0.125,0.125;0.775x0.755)
AgesSubplot(0.125,0.125;0.775x0.755)
AgesSubplot(0.125,0.125;0.775x0.755)
AgesSubplot(0.125,0.125;0.775x0.755)
AgesSubplot(0.125,0.125;0.775x0.755)

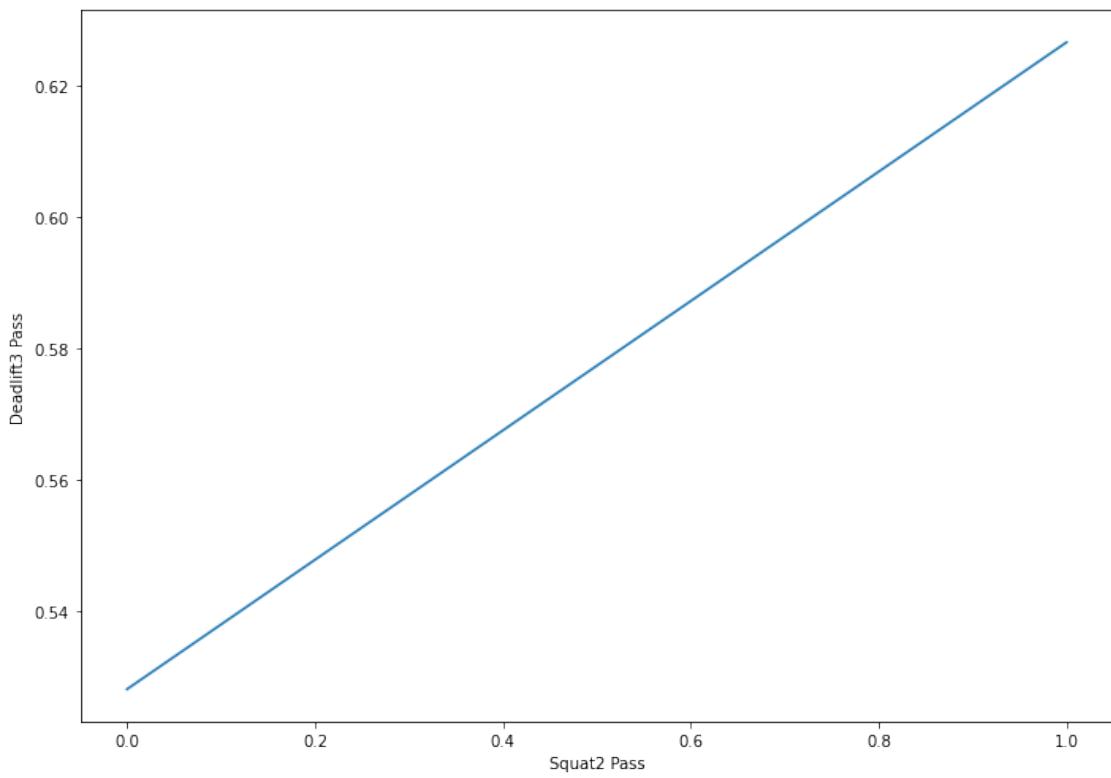


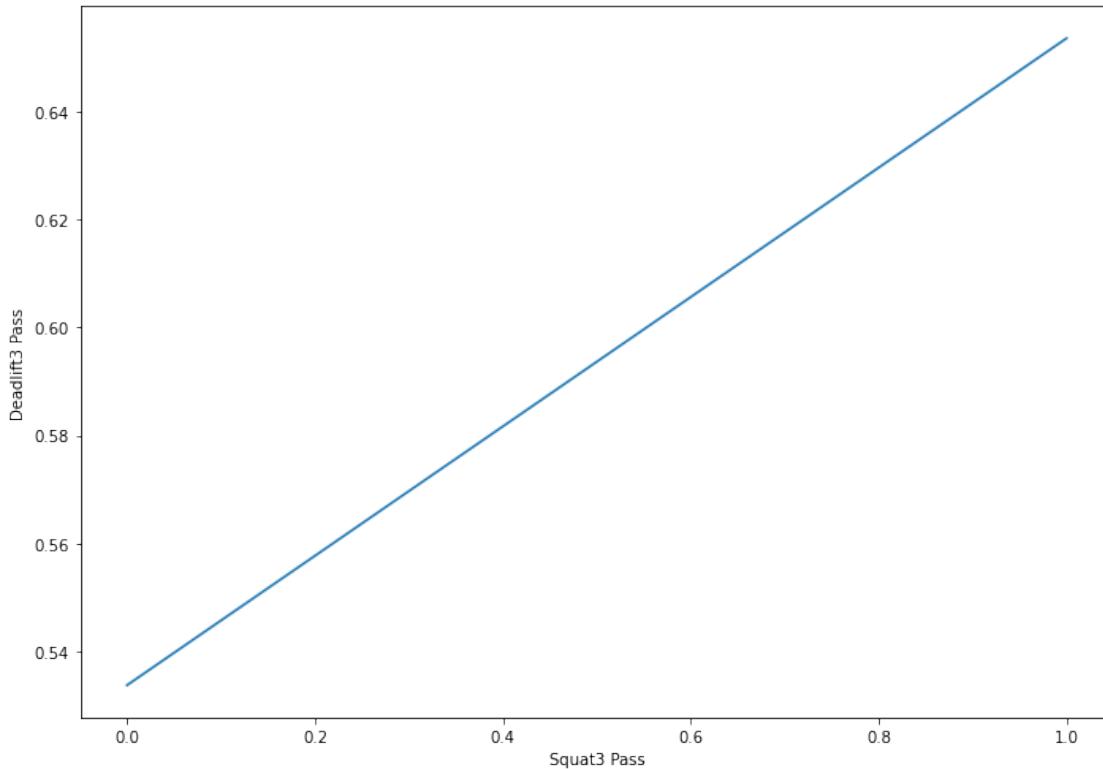












1 % higher chance of hitting squat1 in males

```
[11]: import statsmodels.formula.api as smf
md = smf.mixedlm(formula = "Q('Deadlift3 Pass') ~ Deadlift3Kg + Q('Squat1\u201d\u201dPass') + Q('Squat2 Pass') + Q('Squat3 Pass') + Q('Bench1 Pass') + Q('Bench2\u201d\u201dPass') + Q('Bench3 Pass')+ Q('Deadlift2 Pass') +Q('Number of meets in past\u201d\u201d12 months') + Q('Deadlift3 Past 3 meets success rate') + Q('Best Raw Wilks\u201d\u201dto date') + Q('Overall Attempt Success Rate over past three meets')", data =\u201d\u201dtrain, groups = train["Name"])
mdf = md.fit()
display(mdf.summary())
```

```
C:\Users\Montel\anaconda3\lib\site-
packages\statsmodels\regression\mixed_linear_model.py:2168: ConvergenceWarning:
The MLE may be on the boundary of the parameter space.
```

```
    warnings.warn(msg, ConvergenceWarning)

<class 'statsmodels.iolib.summary2.Summary'>
"""
                                                Mixed Linear Model Regression Results
=====
Model:                               MixedLM          Dependent Variable:   Q('Deadlift3 Pass')
```

No. Observations:	36000	Method:	REML
No. Groups:	17045	Scale:	0.2110
Min. group size:	1	Log-Likelihood:	-23503.3945
Max. group size:	16	Converged:	Yes
Mean group size:	2.1		
<hr/>			
		Coef.	Std.Err.
	P> z [0.025 0.975]		z
<hr/>			
Intercept		0.290	0.022 13.104
Deadlift3Kg		-0.001	0.000 -23.382
Q('Squat1 Pass')		0.024	0.008 2.989
Q('Squat2 Pass')		0.056	0.007 8.106
Q('Squat3 Pass')		0.077	0.005 14.761
Q('Bench1 Pass')		0.034	0.010 3.503
Q('Bench2 Pass')		0.060	0.006 9.451
Q('Bench3 Pass')		0.070	0.005 13.972
Q('Deadlift2 Pass')		0.241	0.008 30.737
Q('Number of meets in past 12 months')		-0.022	0.002 -9.610
Q('Deadlift3 Past 3 meets success rate')		0.039	0.010 4.057
Q('Best Raw Wilks to date')		-0.000	0.000 -8.055
Q('Overall Attempt Success Rate over past three meets')	0.208	0.024	8.744
Group Var	0.004	0.003	
<hr/>			

[8] :

```

for i in ['Age', 'BodyweightKg', 'Time Since Last Meet', 'Number of meets in past 12 months', 'Total Squat Jump', 'Squat1 Pass', 'Squat2 Pass', 'Squat3 Pass', 'Bench1 Pass', 'Bench2 Pass', 'Bench3 Pass', 'Total Bench Jump', 'Deadlift3Kg', 'Deadlift2 Jump', 'Deadlift1 Pass', 'Deadlift2 Pass', 'Total Deadlift Jump', 'Deadlift3 Past 3 meets success rate', 'Best Raw Wilks to date', 'Overall Attempt Success Rate over past three meets']:
    md = smf.mixedlm(formula = f"Q('Deadlift3 Pass') ~ Q('{i}')", data = train, groups = train["Name"])
    mdf = md.fit()
    display(mdf.summary())

```

```

<class 'statsmodels.iolib.summary2.Summary'>
"""
Mixed Linear Model Regression Results
=====
Model: MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000 Method: REML
No. Groups: 17045 Scale: 0.2154
Min. group size: 1 Log-Likelihood: -25046.5461
Max. group size: 16 Converged: Yes
Mean group size: 2.1
-----
      Coef.  Std.Err.      z   P>|z|  [0.025  0.975]
-----
Intercept      0.576      0.008  75.754  0.000    0.562    0.591
Q('Age')       0.001      0.000   5.321  0.000    0.001    0.002
Group Var     0.022      0.003
-----
"""

```

```

<class 'statsmodels.iolib.summary2.Summary'>
"""
Mixed Linear Model Regression Results
=====
Model: MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000 Method: REML
No. Groups: 17045 Scale: 0.2156
Min. group size: 1 Log-Likelihood: -25012.3251
Max. group size: 16 Converged: Yes
Mean group size: 2.1
-----
      Coef.  Std.Err.      z   P>|z|  [0.025  0.975]
-----
Intercept      0.721      0.011  64.822  0.000    0.699    0.743
Q('BodyweightKg') -0.001     0.000 -9.928  0.000   -0.002   -0.001
Group Var     0.021      0.003
-----

```

```

"""
<class 'statsmodels.iolib.summary2.Summary'>
"""

    Mixed Linear Model Regression Results
=====
Model:          MixedLM  Dependent Variable:  Q('Deadlift3 Pass')
No. Observations: 36000   Method:             REML
No. Groups:      17045   Scale:              0.2152
Min. group size: 1       Log-Likelihood:     -25046.7779
Max. group size: 16      Converged:         Yes
Mean group size: 2.1

Coef. Std.Err.      z   P>|z| [0.025 0.975]
-----
Intercept        0.605   0.003 190.973 0.000  0.599  0.611
Q('Time Since Last Meet') 0.000   0.000   5.951 0.000  0.000  0.000
Group Var        0.022   0.003
-----


"""
<class 'statsmodels.iolib.summary2.Summary'>
"""

    Mixed Linear Model Regression Results
=====
Model:          MixedLM  Dependent Variable:  Q('Deadlift3\u2192Pass')
No. Observations: 36000   Method:             REML
No. Groups:      17045   Scale:              0.2145
Min. group size: 1       Log-Likelihood:     -24997.7795
Max. group size: 16      Converged:         Yes
Mean group size: 2.1

Coef. Std.Err.      z   P>|z| [0.025 0.\u2192975]
-----
Intercept        0.652   0.004 146.963 0.000  0.643  0.\u2192661
Q('Number of meets in past 12 months') -0.027   0.002 -11.014 0.000 -0.032 -0.\u2192022
Group Var        0.022   0.003
-----


"""
<class 'statsmodels.iolib.summary2.Summary'>
"""

```

Mixed Linear Model Regression Results

```
=====
Model: MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000 Method: REML
No. Groups: 17045 Scale: 0.2153
Min. group size: 1 Log-Likelihood: -24978.6690
Max. group size: 16 Converged: Yes
Mean group size: 2.1
```

	Coef.	Std.Err.	z	P> z	[0.025 0.975]
Intercept	0.566	0.005	119.028	0.000	0.557 0.575
Q('Total Squat Jump')	0.420	0.034	12.439	0.000	0.354 0.486
Group Var	0.021	0.003			

'''

```
<class 'statsmodels.iolib.summary2.Summary'>
```

'''

Mixed Linear Model Regression Results

```
=====
Model: MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000 Method: REML
No. Groups: 17045 Scale: 0.2153
Min. group size: 1 Log-Likelihood: -25044.1350
Max. group size: 16 Converged: Yes
Mean group size: 2.1
```

	Coef.	Std.Err.	z	P> z	[0.025 0.975]
Intercept	0.575	0.008	71.180	0.000	0.560 0.591
Q('Squat1 Pass')	0.043	0.008	5.109	0.000	0.027 0.060
Group Var	0.022	0.003			

'''

```
<class 'statsmodels.iolib.summary2.Summary'>
```

'''

Mixed Linear Model Regression Results

```
=====
Model: MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000 Method: REML
No. Groups: 17045 Scale: 0.2150
Min. group size: 1 Log-Likelihood: -24971.1575
Max. group size: 16 Converged: Yes
Mean group size: 2.1
```

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
Intercept	0.535	0.007	81.149	0.000	0.523	0.548
Q('Squat2 Pass')	0.093	0.007	13.145	0.000	0.079	0.107
Group Var	0.021	0.003				

"""

```
<class 'statsmodels.iolib.summary2.Summary'>
"""

```

Mixed Linear Model Regression Results

Model:	MixedLM	Dependent Variable:	Q('Deadlift3 Pass')
No. Observations:	36000	Method:	REML
No. Groups:	17045	Scale:	0.2145
Min. group size:	1	Log-Likelihood:	-24837.8510
Max. group size:	16	Converged:	Yes
Mean group size:	2.1		

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
Intercept	0.541	0.004	121.649	0.000	0.532	0.550
Q('Squat3 Pass')	0.113	0.005	21.027	0.000	0.102	0.123
Group Var	0.020	0.003				

"""

```
<class 'statsmodels.iolib.summary2.Summary'>
"""

```

Mixed Linear Model Regression Results

Model:	MixedLM	Dependent Variable:	Q('Deadlift3 Pass')
No. Observations:	36000	Method:	REML
No. Groups:	17045	Scale:	0.2154
Min. group size:	1	Log-Likelihood:	-25041.5576
Max. group size:	16	Converged:	Yes
Mean group size:	2.1		

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
Intercept	0.562	0.010	57.595	0.000	0.543	0.581
Q('Bench1 Pass')	0.056	0.010	5.560	0.000	0.036	0.076
Group Var	0.022	0.003				

"""

```

<class 'statsmodels.iolib.summary2.Summary'>
"""
        Mixed Linear Model Regression Results
=====
Model:          MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000   Method:             REML
No. Groups:      17045    Scale:              0.2151
Min. group size: 1        Log-Likelihood:     -24954.4951
Max. group size: 16       Converged:         Yes
Mean group size: 2.1

Coef.  Std.Err.      z    P>|z|  [0.025  0.975]
-----
Intercept      0.539    0.006   90.643  0.000   0.527   0.550
Q('Bench2 Pass') 0.093    0.007   14.368  0.000   0.081   0.106
Group Var      0.021    0.003
"""

<class 'statsmodels.iolib.summary2.Summary'>
"""
        Mixed Linear Model Regression Results
=====
Model:          MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000   Method:             REML
No. Groups:      17045    Scale:              0.2146
Min. group size: 1        Log-Likelihood:     -24901.8118
Max. group size: 16       Converged:         Yes
Mean group size: 2.1

Coef.  Std.Err.      z    P>|z|  [0.025  0.975]
-----
Intercept      0.574    0.004   159.330  0.000   0.567   0.581
Q('Bench3 Pass') 0.091    0.005   17.692  0.000   0.081   0.101
Group Var      0.021    0.003
"""

<class 'statsmodels.iolib.summary2.Summary'>
"""
        Mixed Linear Model Regression Results
=====
Model:          MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000   Method:             REML
No. Groups:      17045    Scale:              0.2150
Min. group size: 1        Log-Likelihood:     -24936.1669
Max. group size: 16       Converged:         Yes

```

Mean group size: 2.1

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
Intercept	0.547	0.005	106.470	0.000	0.537	0.557
Q('Total Bench Jump')	0.568	0.037	15.503	0.000	0.496	0.640
Group Var	0.021	0.003				

====

<class 'statsmodels.iolib.summary2.Summary'>

====

Mixed Linear Model Regression Results

=====
Model: MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000 Method: REML
No. Groups: 17045 Scale: 0.2133
Min. group size: 1 Log-Likelihood: -24656.0412
Max. group size: 16 Converged: Yes
Mean group size: 2.1

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
Intercept	0.871	0.009	92.849	0.000	0.853	0.890
Q('Deadlift3Kg')	-0.001	0.000	-28.711	0.000	-0.001	-0.001
Group Var	0.018	0.003				

====

<class 'statsmodels.iolib.summary2.Summary'>

====

Mixed Linear Model Regression Results

=====
Model: MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000 Method: REML
No. Groups: 17045 Scale: 0.2155
Min. group size: 1 Log-Likelihood: -25048.3329
Max. group size: 16 Converged: Yes
Mean group size: 2.1

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
Intercept	0.601	0.004	137.669	0.000	0.593	0.610
Q('Deadlift2 Jump')	0.186	0.049	3.766	0.000	0.089	0.282
Group Var	0.022	0.003				

```

"""
<class 'statsmodels.iolib.summary2.Summary'>
"""

    Mixed Linear Model Regression Results
=====
Model:          MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000   Method:             REML
No. Groups:      17045   Scale:              0.2154
Min. group size: 1       Log-Likelihood:     -25039.3381
Max. group size: 16      Converged:         Yes
Mean group size: 2.1

Coef. Std.Err.      z      P>|z| [0.025 0.975]
-----
Intercept        0.542    0.012  43.538 0.000  0.518  0.567
Q('Deadlift1 Pass') 0.075    0.013  5.907  0.000  0.050  0.100
Group Var        0.022    0.003
=====

"""
<class 'statsmodels.iolib.summary2.Summary'>
"""

    Mixed Linear Model Regression Results
=====
Model:          MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000   Method:             REML
No. Groups:      17045   Scale:              0.2114
Min. group size: 1       Log-Likelihood:     -24421.0989
Max. group size: 16      Converged:         Yes
Mean group size: 2.1

Coef. Std.Err.      z      P>|z| [0.025 0.975]
-----
Intercept        0.359    0.008  47.369 0.000  0.344  0.374
Q('Deadlift2 Pass') 0.287    0.008  35.968 0.000  0.271  0.303
Group Var        0.017    0.003
=====

"""
<class 'statsmodels.iolib.summary2.Summary'>
"""

    Mixed Linear Model Regression Results
=====
Model:          MixedLM Dependent Variable: Q('Deadlift3 Pass')
No. Observations: 36000   Method:             REML
No. Groups:      17045   Scale:              0.2155

```

```

Min. group size:      1      Log-Likelihood:      -24968.7339
Max. group size:     16      Converged:          Yes
Mean group size:    2.1

=====
          Coef. Std.Err.      z      P>|z| [0.025 0.975]
-----
Intercept           0.558    0.005 110.082 0.000  0.548  0.568
Q('Total Deadlift Jump') 0.477    0.036 13.215 0.000  0.406  0.547
Group Var          0.021    0.003
=====

"""

C:\Users\Montel\anaconda3\lib\site-
packages\statsmodels\regression\mixed_linear_model.py:2168: ConvergenceWarning:
The MLE may be on the boundary of the parameter space.
    warnings.warn(msg, ConvergenceWarning)

<class 'statsmodels.iolib.summary2.Summary'>
"""

        Mixed Linear Model Regression Results
=====
Model:                 MixedLM   Dependent Variable:  Q('Deadlift3
    ↪Pass')
No. Observations:    36000    Method:             REML
No. Groups:          17045    Scale:              0.2260
Min. group size:     1        Log-Likelihood:      -24955.7271
Max. group size:     16       Converged:          Yes
Mean group size:    2.1

          Coef. Std.Err.      z      P>|z| [0.025 0.
    ↪975]
-----
Intercept           0.529    0.006 82.042 0.000  0.516  0.
    ↪541
Q('Deadlift3 Past 3 meets success rate') 0.128    0.009 14.504 0.000  0.110  0.
    ↪145
Group Var          0.008    0.003
=====

"""

<class 'statsmodels.iolib.summary2.Summary'>
"""

        Mixed Linear Model Regression Results
=====
Model:                 MixedLM   Dependent Variable:  Q('Deadlift3 Pass')
No. Observations:    36000    Method:             REML
No. Groups:          17045    Scale:              0.2146

```

```

Min. group size: 1 Log-Likelihood: -24875.2768
Max. group size: 16 Converged: Yes
Mean group size: 2.1
-----
          Coef. Std.Err. z P>|z| [0.025 0.975]
-----
Intercept 0.753 0.008 98.100 0.000 0.738 0.768
Q('Best Raw Wilks to date') -0.000 0.000 -19.458 0.000 -0.001 -0.000
Group Var 0.020 0.003
=====

```

"""

<class 'statsmodels.iolib.summary2.Summary'>

"""

Mixed Linear Model Regression Results

```

Model: MixedLM Dependent Variable: □
↳ Q('Deadlift3 Pass')
No. Observations: 36000 Method: □
↳ REML
No. Groups: 17045 Scale: 0.
↳ 2207
Min. group size: 1 Log-Likelihood: □
↳ -24929.0568
Max. group size: 16 Converged: Yes
Mean group size: 2.1
-----
```

	Coef.	Std.Err.	z	□
↳ P> z [0.025 0.975]				

```

Intercept 0.336 0.017 19.336 0.
↳ 000 0.302 0.370
Q('Overall Attempt Success Rate over past three meets') 0.354 0.022 16.160 0.
↳ 000 0.311 0.397
Group Var 0.014 0.003
-----
```

"""

```
[155]: for i in ["Squat1 Pass", "Squat2 Pass", "Squat3 Pass", "Bench1 Pass", "Bench2 Pass", "Bench3 Pass", "Deadlift1 Pass", "Deadlift2 Pass", "Deadlift3 Pass",]:
    print(train[i].mean())
```

```

0.8734504387336812
0.8270911874619298
0.6442717679403388
0.9040053997991538

```

```
0.7858189421003243  
0.43112786658545016  
0.950578667500782  
0.8804306669081211  
0.6352666150832195
```

```
[156]: pd.concat([train, pd.get_dummies(train["WeightClassKg"])], axis = 1).corr().  
       to_csv("Correlation Matrix2.csv")
```

```
[157]: pd.concat([train, pd.get_dummies(train["WeightClassKg"])], axis = 1).  
       drop(['Name', 'WeightClassKg', 'Squat2Kg',  
             'Squat2 Pass',  
             'Squat3Kg',  
             'Squat3 Pass',  
             'Squat4Kg',  
             'Best3SquatKg',  
             'Bench1Kg',  
             'Bench1 Pass',  
             'Bench2Kg',  
             'Bench2 Pass',  
             'Bench3Kg',  
             'Bench3 Pass',  
             'Bench4Kg',  
             'Best3BenchKg',  
             'Deadlift1Kg',  
             'Deadlift1 Pass',  
             'Deadlift2Kg',  
             'Deadlift2 Pass',  
             'Deadlift3Kg',  
             'Deadlift3 Pass',  
             'Deadlift4Kg',  
             'Best3DeadliftKg',  
             'TotalKg',  
             'Place',  
             'Dots',  
             'Wilks',  
             'Date',  
             'MeetCountry',  
             'MeetState',  
             'MeetTown',  
             'Best M-Ply Squat to date',  
             'Best M-Ply Bench to date',  
             'Best M-Ply Deadlift to date',  
             'Best M-Ply Total to date',  
             'Best M-Ply Wilks to date',  
             'Best Unlimited Squat to date',  
             'Best Unlimited Bench to date',
```

```
'Best Unlimited Deadlift to date',
'Best Unlimited Total to date',
'Best Unlimited Wilks to date'], axis = 1).corr()["Squat1 Pass"].
↪sort_values(ascending = False).to_csv("Squat1 Pass_corr.csv")
```

```
[158]: pd.concat([train, pd.get_dummies(train["WeightClassKg"])], axis = 1).
↪drop(['WeightClassKg',
'Squat3Kg',
'Squat3 Pass',
'Squat4Kg',
'Best3SquatKg',
'Bench1Kg',
'Bench1 Pass',
'Bench2Kg',
'Bench2 Pass',
'Bench3Kg',
'Bench3 Pass',
'Bench4Kg',
'Best3BenchKg',
'Deadlift1Kg',
'Deadlift1 Pass',
'Deadlift2Kg',
'Deadlift2 Pass',
'Deadlift3Kg',
'Deadlift3 Pass',
'Deadlift4Kg',
'Best3DeadliftKg',
'TotalKg',
'Place',
'Dots',
'Wilks',
'Date',
'MeetCountry',
'MeetState',
'MeetTown',
'Best M-Ply Squat to date',
'Best M-Ply Bench to date',
'Best M-Ply Deadlift to date',
'Best M-Ply Total to date',
'Best M-Ply Wilks to date',
'Best Unlimited Squat to date',
'Best Unlimited Bench to date',
'Best Unlimited Deadlift to date',
'Best Unlimited Total to date',
'Best Unlimited Wilks to date'], axis = 1).corr()["Squat2 Pass"].
↪sort_values(ascending = False).to_csv("Squat2 Pass_corr.csv")
```

```
[159]: pd.concat([train, pd.get_dummies(train["WeightClassKg"])], axis = 1).  
→drop(['WeightClassKg',  
       'Squat4Kg',  
       'Best3SquatKg',  
       'Bench1Kg',  
       'Bench1 Pass',  
       'Bench2Kg',  
       'Bench2 Pass',  
       'Bench3Kg',  
       'Bench3 Pass',  
       'Bench4Kg',  
       'Best3BenchKg',  
       'Deadlift1Kg',  
       'Deadlift1 Pass',  
       'Deadlift2Kg',  
       'Deadlift2 Pass',  
       'Deadlift3Kg',  
       'Deadlift3 Pass',  
       'Deadlift4Kg',  
       'Best3DeadliftKg',  
       'TotalKg',  
       'Place',  
       'Dots',  
       'Wilks',  
       'Date',  
       'MeetCountry',  
       'MeetState',  
       'MeetTown',  
       'Best M-Ply Squat to date',  
       'Best M-Ply Bench to date',  
       'Best M-Ply Deadlift to date',  
       'Best M-Ply Total to date',  
       'Best M-Ply Wilks to date',  
       'Best Unlimited Squat to date',  
       'Best Unlimited Bench to date',  
       'Best Unlimited Deadlift to date',  
       'Best Unlimited Total to date',  
       'Best Unlimited Wilks to date'], axis = 1).corr()["Squat3 Pass"].  
→sort_values(ascending = False).to_csv("Squat3 Pass_corr.csv")
```

```
[160]: pd.concat([train, pd.get_dummies(train["WeightClassKg"])], axis = 1).  
→drop(['Name', 'WeightClassKg',  
       'Bench2Kg',  
       'Bench2 Pass',  
       'Bench3Kg',  
       'Bench3 Pass',  
       'Bench4Kg',
```

```

'Best3BenchKg',
'Deadlift1Kg',
'Deadlift1 Pass',
'Deadlift2Kg',
'Deadlift2 Pass',
'Deadlift3Kg',
'Deadlift3 Pass',
'Deadlift4Kg',
'Best3DeadliftKg',
'TotalKg',
'Place',
'Dots',
'Wilks',
'Date',
'MeetCountry',
'MeetState',
'MeetTown',
'Best M-Ply Squat to date',
'Best M-Ply Bench to date',
'Best M-Ply Deadlift to date',
'Best M-Ply Total to date',
'Best M-Ply Wilks to date',
'Best Unlimited Squat to date',
'Best Unlimited Bench to date',
'Best Unlimited Deadlift to date',
'Best Unlimited Total to date',
'Best Unlimited Wilks to date'], axis = 1).corr()["Bench1 Pass"]].
↪sort_values(ascending = False).to_csv("Bench1 Pass_corr.csv")

```

```

[161]: pd.concat([train, pd.get_dummies(train["WeightClassKg"])], axis = 1).
↪drop(['Name', 'WeightClassKg',
'Bench3Kg',
'Bench3 Pass',
'Bench4Kg',
'Best3BenchKg',
'Deadlift1Kg',
'Deadlift1 Pass',
'Deadlift2Kg',
'Deadlift2 Pass',
'Deadlift3Kg',
'Deadlift3 Pass',
'Deadlift4Kg',
'Best3DeadliftKg',
'TotalKg',
'Place',
'Dots',
'Wilks',

```

```

'Date',
'MeetCountry',
'MeetState',
'MeetTown',
'Best M-Ply Squat to date',
'Best M-Ply Bench to date',
'Best M-Ply Deadlift to date',
'Best M-Ply Total to date',
'Best M-Ply Wilks to date',
'Best Unlimited Squat to date',
'Best Unlimited Bench to date',
'Best Unlimited Deadlift to date',
'Best Unlimited Total to date',
'Best Unlimited Wilks to date'], axis = 1).corr()["Bench1 Pass"].
↪sort_values(ascending = False).to_csv("Bench2 Pass_corr.csv")

```

```

[162]: pd.concat([train, pd.get_dummies(train["WeightClassKg"])], axis = 1).
↪drop(['Name', 'WeightClassKg',
'Best3BenchKg',
'Deadlift1Kg',
'Deadlift1 Pass',
'Deadlift2Kg',
'Deadlift2 Pass',
'Deadlift3Kg',
'Deadlift3 Pass',
'Deadlift4Kg',
'Best3DeadliftKg',
'TotalKg',
'Place',
'Dots',
'Wilks',
'Date',
'MeetCountry',
'MeetState',
'MeetTown',
'Best M-Ply Squat to date',
'Best M-Ply Bench to date',
'Best M-Ply Deadlift to date',
'Best M-Ply Total to date',
'Best M-Ply Wilks to date',
'Best Unlimited Squat to date',
'Best Unlimited Bench to date',
'Best Unlimited Deadlift to date',
'Best Unlimited Total to date',
'Best Unlimited Wilks to date'], axis = 1).corr()["Bench3 Pass"].
↪sort_values(ascending = False).to_csv("Bench3 Pass_corr.csv")

```

```
[163]: pd.concat([train, pd.get_dummies(train["WeightClassKg"])], axis = 1).  
    ↪drop(['Name', 'WeightClassKg',  
    'Deadlift2Kg',  
    'Deadlift2 Pass',  
    'Deadlift3Kg',  
    'Deadlift3 Pass',  
    'Deadlift4Kg',  
    'Best3DeadliftKg',  
    'TotalKg',  
    'Place',  
    'Dots',  
    'Wilks',  
    'Date',  
    'MeetCountry',  
    'MeetState',  
    'MeetTown',  
    'Best M-Ply Squat to date',  
    'Best M-Ply Bench to date',  
    'Best M-Ply Deadlift to date',  
    'Best M-Ply Total to date',  
    'Best M-Ply Wilks to date',  
    'Best Unlimited Squat to date',  
    'Best Unlimited Bench to date',  
    'Best Unlimited Deadlift to date',  
    'Best Unlimited Total to date',  
    'Best Unlimited Wilks to date'], axis = 1).corr()["Deadlift1 Pass"].  
    ↪sort_values(ascending = False).to_csv("Deadlift1 Pass_corr.csv")
```

```
[164]: pd.concat([train, pd.get_dummies(train["WeightClassKg"])], axis = 1).  
    ↪drop(['Name', 'WeightClassKg',  
    'Deadlift3Kg',  
    'Deadlift3 Pass',  
    'Deadlift4Kg',  
    'Best3DeadliftKg',  
    'TotalKg',  
    'Place',  
    'Dots',  
    'Wilks',  
    'Date',  
    'MeetCountry',  
    'MeetState',  
    'MeetTown',  
    'Best M-Ply Squat to date',  
    'Best M-Ply Bench to date',  
    'Best M-Ply Deadlift to date',  
    'Best M-Ply Total to date',  
    'Best M-Ply Wilks to date',
```

```
'Best Unlimited Squat to date',
'Best Unlimited Bench to date',
'Best Unlimited Deadlift to date',
'Best Unlimited Total to date',
'Best Unlimited Wilks to date'], axis = 1).corr()["Deadlift2 Pass"].
↪sort_values(ascending = False).to_csv("Deadlift2 Pass_corr.csv")
```

```
[165]: pd.concat([train, pd.get_dummies(train["WeightClassKg"])], axis = 1).
↪drop(['Name', 'WeightClassKg',
'Deadlift4Kg',
'Best3DeadliftKg',
'TotalKg',
'Place',
'Dots',
'Wilks',
'Date',
'MeetCountry',
'MeetState',
'MeetTown',
'Best M-Ply Squat to date',
'Best M-Ply Bench to date',
'Best M-Ply Deadlift to date',
'Best M-Ply Total to date',
'Best M-Ply Wilks to date',
'Best Unlimited Squat to date',
'Best Unlimited Bench to date',
'Best Unlimited Deadlift to date',
'Best Unlimited Total to date',
'Best Unlimited Wilks to date'], axis = 1).corr()["Deadlift3 Pass"].
↪sort_values(ascending = False).to_csv("Deadlift3 Pass_corr.csv")
```

```
[ ]:
```

b. Correlation Coefficients

Squat1 Pass		
Feature	Corr	Influence
Best Raw Wilks to date	0.0927	positive
Best Raw Squat to date	0.0890	positive
Best Raw Total to date	0.0876	positive
Best Raw Deadlift to date	0.0855	positive
Best Raw Bench to date	0.0802	positive
Number of meets in past 12 months	0.0636	positive
Number of Past Meets Same Day Exclusive	0.0602	positive
Number of Past Meets	0.0601	positive
Age	0.0513	negative
Overall Attempt Success Rate over past three meets	0.0511	positive
Squat1 Past 3 meets success rate	0.0491	positive
Squat2 Past 3 meets success rate	0.0394	positive
84+Kg Weight Class	0.0392	negative
Squat1Kg	0.0375	positive
Bench2 Past 3 meets success rate	0.0266	positive
Bench1 Past 3 meets success rate	0.0256	positive
Bench3 Past 3 meets success rate	0.0232	positive
BodyweightKg	0.0188	negative
Squat3 Past 3 meets success rate	0.0182	positive
Male	0.0174	positive
84Kg Weight Class	0.0171	negative
Deadlift1 Past 3 meets success rate	0.0141	positive
Deadlift2 Past 3 meets success rate	0.0109	positive
90+Kg Weight Class	0.0108	negative
93Kg Weight Class	0.0103	positive
74Kg Weight Class	0.0102	positive
83Kg Weight Class	0.0088	positive
63Kg Weight Class	0.0087	positive
67.5Kg Weight Class	0.0083	negative
43Kg Weight Class	0.0081	positive
47Kg Weight Class	0.0079	positive
100Kg Weight Class	0.0075	negative
56Kg Weight Class	0.0075	negative
125Kg Weight Class	0.0064	negative
120+Kg Weight Class	0.0061	negative
35Kg Weight Class	0.0060	positive
52Kg Weight Class	0.0054	positive
48Kg Weight Class	0.0053	positive
75Kg Weight Class	0.0052	negative
Deadlift3 Past 3 meets success rate	0.0047	positive
40Kg Weight Class	0.0046	positive
90Kg Weight Class	0.0041	negative
60Kg Weight Class	0.0039	positive
66Kg Weight Class	0.0039	negative

53Kg Weight Class	0.0035	positive
30Kg Weight Class	0.0027	positive
59Kg Weight Class	0.0026	positive
105Kg Weight Class	0.0024	positive
120Kg Weight Class	0.0023	positive
44Kg Weight Class	0.0023	positive
72Kg Weight Class	0.0017	positive
110Kg Weight Class	0.0014	positive
125+Kg Weight Class	0.0013	negative
82.5Kg Weight Class	0.0013	positive
57Kg Weight Class	0.0004	negative

Squat2 Pass		
Feature	Corr	Influence
Overall Attempt Success Rate over past three meets	0.0757	positive
Squat3 Past 3 meets success rate	0.0493	positive
Age	0.0452	negative
Squat2 Past 3 meets success rate	0.0442	positive
Male	0.0378	positive
Bench2 Past 3 meets success rate	0.0378	positive
Squat1 Past 3 meets success rate	0.0353	positive
Bench3 Past 3 meets success rate	0.0341	positive
Deadlift3 Past 3 meets success rate	0.0324	positive
Best Raw Total to date	0.0323	positive
Best Raw Deadlift to date	0.0319	positive
Best Raw Bench to date	0.0315	positive
Best Raw Squat to date	0.0311	positive
Best Raw Wilks to date	0.0283	positive
Deadlift2 Past 3 meets success rate	0.0241	positive
Squat2 Jump	0.0230	positive
Squat2Kg	0.0221	positive
Squat1Kg	0.0216	positive
83Kg Weight Class	0.0200	positive
Bench1 Past 3 meets success rate	0.0189	positive
105Kg Weight Class	0.0181	positive
82.5Kg Weight Class	0.0179	negative
67.5Kg Weight Class	0.0177	negative
56Kg Weight Class	0.0165	negative
63Kg Weight Class	0.0163	negative
BodyweightKg	0.0149	positive
93Kg Weight Class	0.0143	positive
Deadlift1 Past 3 meets success rate	0.0137	positive
57Kg Weight Class	0.0135	negative
Number of meets in past 12 months	0.0132	positive
125+Kg Weight Class	0.0128	negative
52Kg Weight Class	0.0127	negative
100Kg Weight Class	0.0121	negative
72Kg Weight Class	0.0118	negative
40Kg Weight Class	0.0111	positive
84+Kg Weight Class	0.0109	negative
Number of Past Meets	0.0108	positive
Number of Past Meets Same Day Exclusive	0.0107	positive
35Kg Weight Class	0.0075	positive
60Kg Weight Class	0.0073	negative
110Kg Weight Class	0.0068	negative
44Kg Weight Class	0.0055	positive
84Kg Weight Class	0.0052	negative
74Kg Weight Class	0.0049	positive

53Kg Weight Class	0.0045	positive
30Kg Weight Class	0.0045	negative
75Kg Weight Class	0.0040	negative
43Kg Weight Class	0.0040	negative
59Kg Weight Class	0.0039	negative
125Kg Weight Class	0.0033	negative
120Kg Weight Class	0.0028	positive
66Kg Weight Class	0.0025	positive
48Kg Weight Class	0.0016	negative
120+Kg Weight Class	0.0016	positive
90Kg Weight Class	0.0016	positive
47Kg Weight Class	0.0006	negative
90+Kg Weight Class	0.0001	negative

Squat3 Pass		
Feature	Corr	Influence
Overall Attempt Success Rate over past three meets	0.0893	positive
Squat3 Past 3 meets success rate	0.0706	positive
Squat3 Jump	0.0610	positive
Deadlift3 Past 3 meets success rate	0.0541	positive
BodyweightKg	0.0461	positive
Best Raw Wilks to date	0.0458	negative
Bench3 Past 3 meets success rate	0.0442	positive
Squat2Kg	0.0433	negative
Total Squat Jump	0.0424	positive
Best Raw Squat to date	0.0404	negative
Squat1Kg	0.0404	negative
Best Raw Deadlift to date	0.0392	negative
Deadlift2 Past 3 meets success rate	0.0392	positive
Number of meets in past 12 months	0.0384	negative
Bench2 Past 3 meets success rate	0.0380	positive
Squat2 Past 3 meets success rate	0.0378	positive
Best Raw Total to date	0.0369	negative
Best Raw Bench to date	0.0351	negative
84+Kg Weight Class	0.0320	positive
63Kg Weight Class	0.0290	negative
Squat3Kg	0.0288	negative
Number of Past Meets Same Day Exclusive	0.0267	negative
Number of Past Meets	0.0266	negative
57Kg Weight Class	0.0217	negative
74Kg Weight Class	0.0211	negative
Bench1 Past 3 meets success rate	0.0199	positive
120+Kg Weight Class	0.0185	positive
93Kg Weight Class	0.0173	positive
52Kg Weight Class	0.0168	negative
120Kg Weight Class	0.0152	positive
84Kg Weight Class	0.0147	positive
Male	0.0146	positive
Squat1 Past 3 meets success rate	0.0143	positive
60Kg Weight Class	0.0139	negative
90Kg Weight Class	0.0131	negative
59Kg Weight Class	0.0128	negative
105Kg Weight Class	0.0116	positive
Squat2 Jump	0.0112	positive
66Kg Weight Class	0.0109	negative
40Kg Weight Class	0.0104	positive
Age	0.0099	positive
82.5Kg Weight Class	0.0099	negative
Deadlift1 Past 3 meets success rate	0.0096	positive
47Kg Weight Class	0.0084	negative

35Kg Weight Class	0.0079	positive
125+Kg Weight Class	0.0079	negative
100Kg Weight Class	0.0077	negative
30Kg Weight Class	0.0076	positive
67.5Kg Weight Class	0.0076	negative
44Kg Weight Class	0.0069	positive
125Kg Weight Class	0.0069	negative
75Kg Weight Class	0.0060	negative
72Kg Weight Class	0.0060	negative
43Kg Weight Class	0.0056	positive
83Kg Weight Class	0.0053	positive
90+Kg Weight Class	0.0049	negative
53Kg Weight Class	0.0024	positive
110Kg Weight Class	0.0021	negative
48Kg Weight Class	0.0002	positive
56Kg Weight Class	0.0001	negative
Bench1Kg	0.0000	positive
Bench2Kg	0.0000	positive
Bench3Kg	0.0000	positive
Deadlift1Kg	0.0000	positive
Deadlift2Kg	0.0000	positive
Deadlift3Kg	0.0000	positive

Bench1 Pass		
Feature	Corr	Influence
Best Raw Wilks to date	0.0938	positive
Best Raw Squat to date	0.0811	positive
Number of meets in past 12 months	0.0808	positive
Best Raw Total to date	0.0795	positive
Best Raw Deadlift to date	0.0777	positive
Best Raw Bench to date	0.0686	positive
Number of Past Meets Same Day Exclusive	0.0672	positive
Number of Past Meets	0.0671	positive
Male	0.0475	negative
Bench1 Past 3 meets success rate	0.0402	positive
Overall Attempt Success Rate over past three meets	0.0365	positive
Total Squat Jump	0.0363	positive
Squat2 Jump	0.0349	positive
Bench1Kg	0.0325	negative
Squat3 Jump	0.0219	positive
52Kg Weight Class	0.0204	positive
Bench2 Past 3 meets success rate	0.0202	positive
84+Kg Weight Class	0.0202	positive
Squat1 Past 3 meets success rate	0.0197	positive
BodyweightKg	0.0191	negative
93Kg Weight Class	0.0165	negative
Bench3 Past 3 meets success rate	0.0160	positive
74Kg Weight Class	0.0154	negative
105Kg Weight Class	0.0153	negative
Squat2 Past 3 meets success rate	0.0150	positive
83Kg Weight Class	0.0145	negative
72Kg Weight Class	0.0141	positive
84Kg Weight Class	0.0132	positive
57Kg Weight Class	0.0132	positive
Squat3 Past 3 meets success rate	0.0129	positive
63Kg Weight Class	0.0116	positive
Deadlift1 Past 3 meets success rate	0.0115	positive
Deadlift2 Past 3 meets success rate	0.0101	positive
Squat1Kg	0.0100	negative
120Kg Weight Class	0.0095	negative
66Kg Weight Class	0.0093	negative
60Kg Weight Class	0.0090	positive
Age	0.0084	positive
Deadlift3 Past 3 meets success rate	0.0083	positive
67.5Kg Weight Class	0.0065	positive
110Kg Weight Class	0.0060	positive
30Kg Weight Class	0.0057	positive
40Kg Weight Class	0.0055	positive
44Kg Weight Class	0.0055	positive

120+Kg Weight Class	0.0053	negative
Squat2Kg	0.0051	negative
82.5Kg Weight Class	0.0048	positive
48Kg Weight Class	0.0042	positive
75Kg Weight Class	0.0042	positive
59Kg Weight Class	0.0039	positive
53Kg Weight Class	0.0036	positive
43Kg Weight Class	0.0036	positive
90+Kg Weight Class	0.0036	positive
90Kg Weight Class	0.0030	positive
35Kg Weight Class	0.0030	positive
125Kg Weight Class	0.0025	positive
Squat3Kg	0.0023	negative
47Kg Weight Class	0.0017	positive
56Kg Weight Class	0.0012	positive
100Kg Weight Class	0.0005	positive
125+Kg Weight Class	0.0001	positive

Bench2 Pass		
Feature	Corr	Influence
Overall Attempt Success Rate over past three meets	0.0711	positive
Squat3 Jump	0.0470	positive
Bench2 Past 3 meets success rate	0.0470	positive
Total Squat Jump	0.0460	positive
Bench3 Past 3 meets success rate	0.0436	positive
Best Raw Wilks to date	0.0353	positive
Squat3 Past 3 meets success rate	0.0351	positive
Best Raw Squat to date	0.0345	positive
Number of Past Meets Same Day Exclusive	0.0317	positive
Number of Past Meets	0.0316	positive
Best Raw Total to date	0.0313	positive
Bench1 Past 3 meets success rate	0.0307	positive
Squat2 Jump	0.0297	positive
Deadlift3 Past 3 meets success rate	0.0296	positive
Number of meets in past 12 months	0.0272	positive
Best Raw Deadlift to date	0.0262	positive
Squat2 Past 3 meets success rate	0.0261	positive
Deadlift2 Past 3 meets success rate	0.0254	positive
Squat1 Past 3 meets success rate	0.0225	positive
Bench2Kg	0.0212	negative
Best Raw Bench to date	0.0212	positive
Age	0.0209	negative
Bench1Kg	0.0206	negative
84+Kg Weight Class	0.0171	positive
Deadlift1 Past 3 meets success rate	0.0129	positive
Squat3Kg	0.0122	positive
57Kg Weight Class	0.0113	negative
53Kg Weight Class	0.0111	positive
BodyweightKg	0.0087	positive
40Kg Weight Class	0.0080	positive
47Kg Weight Class	0.0078	negative
Bench2 Jump	0.0071	positive
74Kg Weight Class	0.0066	negative
59Kg Weight Class	0.0064	positive
Squat2Kg	0.0061	positive
56Kg Weight Class	0.0059	positive
67.5Kg Weight Class	0.0058	positive
105Kg Weight Class	0.0056	positive
35Kg Weight Class	0.0052	positive
84Kg Weight Class	0.0051	negative
63Kg Weight Class	0.0050	negative
43Kg Weight Class	0.0044	positive
82.5Kg Weight Class	0.0044	positive
120+Kg Weight Class	0.0042	positive

Squat1Kg	0.0035	positive
72Kg Weight Class	0.0035	positive
90+Kg Weight Class	0.0031	positive
83Kg Weight Class	0.0031	negative
125+Kg Weight Class	0.0030	positive
93Kg Weight Class	0.0030	negative
30Kg Weight Class	0.0029	positive
100Kg Weight Class	0.0026	positive
66Kg Weight Class	0.0024	negative
110Kg Weight Class	0.0021	positive
60Kg Weight Class	0.0015	negative
120Kg Weight Class	0.0015	negative
75Kg Weight Class	0.0012	positive
52Kg Weight Class	0.0009	negative
90Kg Weight Class	0.0009	positive
48Kg Weight Class	0.0005	negative
44Kg Weight Class	0.0004	negative
Male	0.0000	negative
125Kg Weight Class	0.0000	negative

Bench3 Pass		
Feature	Corr	Influence
Overall Attempt Success Rate over past three meets	0.0716	positive
Bench3 Past 3 meets success rate	0.0571	positive
Bench2 Past 3 meets success rate	0.0436	positive
Squat2 Past 3 meets success rate	0.0381	positive
Bench2 Jump	0.0371	negative
Squat3 Past 3 meets success rate	0.0370	positive
BodyweightKg	0.0361	positive
Squat3Kg	0.0330	positive
Deadlift3 Past 3 meets success rate	0.0290	positive
Squat2Kg	0.0287	positive
Squat1Kg	0.0277	positive
Male	0.0253	positive
Best Raw Squat to date	0.0241	positive
Total Bench Jump	0.0212	negative
Best Raw Total to date	0.0209	positive
63Kg Weight Class	0.0198	negative
Deadlift2 Past 3 meets success rate	0.0196	positive
Squat1 Past 3 meets success rate	0.0186	positive
Number of Past Meets Same Day Exclusive	0.0184	positive
Number of Past Meets	0.0184	positive
Best Raw Wilks to date	0.0173	positive
Best Raw Deadlift to date	0.0171	positive
Best Raw Bench to date	0.0169	positive
57Kg Weight Class	0.0166	negative
Bench1 Past 3 meets success rate	0.0150	positive
52Kg Weight Class	0.0144	negative
84+Kg Weight Class	0.0144	positive
Age	0.0134	negative
Bench3Kg	0.0130	positive
72Kg Weight Class	0.0128	negative
35Kg Weight Class	0.0128	positive
Squat3 Jump	0.0126	positive
105Kg Weight Class	0.0125	positive
120+Kg Weight Class	0.0122	positive
Deadlift2 Jump	0.0114	positive
Total Squat Jump	0.0109	positive
Bench3 Jump	0.0109	positive
93Kg Weight Class	0.0107	positive
47Kg Weight Class	0.0104	negative
Bench1Kg	0.0091	positive
100Kg Weight Class	0.0087	positive
60Kg Weight Class	0.0086	negative
120Kg Weight Class	0.0079	positive
90+Kg Weight Class	0.0071	positive

Number of meets in past 12 months	0.0061	positive
Bench2Kg	0.0061	positive
82.5Kg Weight Class	0.0061	negative
48Kg Weight Class	0.0056	positive
125+Kg Weight Class	0.0049	positive
Squat2 Jump	0.0048	positive
59Kg Weight Class	0.0048	negative
74Kg Weight Class	0.0040	positive
125Kg Weight Class	0.0034	negative
66Kg Weight Class	0.0034	negative
43Kg Weight Class	0.0032	positive
56Kg Weight Class	0.0031	negative
90Kg Weight Class	0.0031	positive
44Kg Weight Class	0.0030	negative
75Kg Weight Class	0.0026	positive
Deadlift1 Past 3 meets success rate	0.0025	positive
83Kg Weight Class	0.0019	positive
30Kg Weight Class	0.0018	positive
40Kg Weight Class	0.0014	negative
53Kg Weight Class	0.0013	negative
67.5Kg Weight Class	0.0012	negative
110Kg Weight Class	0.0003	negative
84Kg Weight Class	0.0002	negative

Deadlift1 Pass		
Feature	Corr	Influence
Deadlift1Kg	0.0543	negative
Total Bench Jump	0.0542	positive
Total Squat Jump	0.0540	positive
Squat1Kg	0.0529	negative
Male	0.0513	negative
Bench1Kg	0.0507	negative
Squat2Kg	0.0483	negative
Overall Attempt Success Rate over past three meets	0.0480	positive
Bench2 Jump	0.0470	positive
Bench3 Jump	0.0465	positive
Squat3 Jump	0.0451	positive
Bench2Kg	0.0448	negative
Squat3Kg	0.0429	negative
Squat2 Jump	0.0423	positive
Bench3Kg	0.0403	negative
Deadlift2 Past 3 meets success rate	0.0371	positive
Deadlift3 Past 3 meets success rate	0.0367	positive
Deadlift1 Past 3 meets success rate	0.0291	positive
72Kg Weight Class	0.0262	positive
Best Raw Bench to date	0.0258	negative
Best Raw Deadlift to date	0.0240	negative
BodyweightKg	0.0229	negative
66Kg Weight Class	0.0229	negative
Best Raw Total to date	0.0225	negative
Best Raw Squat to date	0.0225	negative
84Kg Weight Class	0.0211	positive
Bench2 Past 3 meets success rate	0.0205	positive
Bench1 Past 3 meets success rate	0.0205	positive
Age	0.0205	positive
63Kg Weight Class	0.0195	positive
Squat3 Past 3 meets success rate	0.0184	positive
74Kg Weight Class	0.0171	negative
Bench3 Past 3 meets success rate	0.0144	positive
84+Kg Weight Class	0.0129	positive
120+Kg Weight Class	0.0125	negative
57Kg Weight Class	0.0124	positive
56Kg Weight Class	0.0124	negative
Squat2 Past 3 meets success rate	0.0122	positive
Best Raw Wilks to date	0.0120	negative
93Kg Weight Class	0.0118	negative
83Kg Weight Class	0.0106	negative
Squat1 Past 3 meets success rate	0.0100	positive
90+Kg Weight Class	0.0092	negative
125+Kg Weight Class	0.0084	negative

43Kg Weight Class	0.0074	positive
105Kg Weight Class	0.0072	negative
Number of Past Meets Same Day Exclusive	0.0072	positive
Number of Past Meets	0.0071	positive
82.5Kg Weight Class	0.0069	negative
30Kg Weight Class	0.0067	positive
120Kg Weight Class	0.0065	negative
47Kg Weight Class	0.0063	positive
75Kg Weight Class	0.0063	negative
110Kg Weight Class	0.0059	negative
125Kg Weight Class	0.0056	negative
48Kg Weight Class	0.0054	positive
40Kg Weight Class	0.0050	positive
100Kg Weight Class	0.0048	negative
59Kg Weight Class	0.0048	negative
35Kg Weight Class	0.0043	positive
Deadlift3 Jump	0.0043	negative
67.5Kg Weight Class	0.0039	positive
Number of meets in past 12 months	0.0038	positive
90Kg Weight Class	0.0031	negative
60Kg Weight Class	0.0029	positive
52Kg Weight Class	0.0018	positive
53Kg Weight Class	0.0014	negative
44Kg Weight Class	0.0012	positive

Deadlift2 Pass		
Feature	Corr	Influence
Deadlift2Kg	0.1052	negative
Deadlift1Kg	0.0972	negative
Overall Attempt Success Rate over past three meets	0.0896	positive
Squat2Kg	0.0879	negative
Squat1Kg	0.0859	negative
Best Raw Deadlift to date	0.0851	negative
Best Raw Bench to date	0.0833	negative
Deadlift3 Past 3 meets success rate	0.0819	positive
Bench1Kg	0.0807	negative
Squat3 Jump	0.0795	positive
Squat3Kg	0.0791	negative
Bench2Kg	0.0789	negative
Best Raw Total to date	0.0774	negative
Best Raw Squat to date	0.0770	negative
Total Bench Jump	0.0757	positive
Total Squat Jump	0.0746	positive
Bench3 Jump	0.0740	positive
Deadlift2 Past 3 meets success rate	0.0738	positive
Bench3Kg	0.0718	negative
Best Raw Wilks to date	0.0654	negative
Male	0.0629	negative
Bench2 Jump	0.0566	positive
Number of meets in past 12 months	0.0461	negative
Number of Past Meets Same Day Exclusive	0.0445	negative
Number of Past Meets	0.0444	negative
Squat3 Past 3 meets success rate	0.0436	positive
BodyweightKg	0.0425	negative
Squat2 Jump	0.0414	positive
Deadlift2 Jump	0.0387	positive
72Kg Weight Class	0.0361	positive
Bench2 Past 3 meets success rate	0.0348	positive
120+Kg Weight Class	0.0297	negative
Squat2 Past 3 meets success rate	0.0296	positive
Deadlift1 Past 3 meets success rate	0.0286	positive
84Kg Weight Class	0.0260	positive
63Kg Weight Class	0.0256	positive
Bench3 Past 3 meets success rate	0.0248	positive
74Kg Weight Class	0.0228	negative
Bench1 Past 3 meets success rate	0.0183	positive
84+Kg Weight Class	0.0177	positive
120Kg Weight Class	0.0174	negative
59Kg Weight Class	0.0164	negative
93Kg Weight Class	0.0157	negative
110Kg Weight Class	0.0145	negative

Squat1 Past 3 meets success rate	0.0142	positive
125+Kg Weight Class	0.0135	negative
66Kg Weight Class	0.0131	negative
Age	0.0115	positive
125Kg Weight Class	0.0105	negative
105Kg Weight Class	0.0099	negative
57Kg Weight Class	0.0084	positive
56Kg Weight Class	0.0069	negative
35Kg Weight Class	0.0068	positive
90Kg Weight Class	0.0063	negative
48Kg Weight Class	0.0063	positive
30Kg Weight Class	0.0056	positive
60Kg Weight Class	0.0051	negative
82.5Kg Weight Class	0.0040	negative
40Kg Weight Class	0.0039	positive
90+Kg Weight Class	0.0039	negative
53Kg Weight Class	0.0038	positive
47Kg Weight Class	0.0028	positive
100Kg Weight Class	0.0027	negative
43Kg Weight Class	0.0024	positive
75Kg Weight Class	0.0021	positive
52Kg Weight Class	0.0020	positive
44Kg Weight Class	0.0017	positive
67.5Kg Weight Class	0.0016	positive
83Kg Weight Class	0.0006	negative

Deadlift3 Pass		
Feature	Corr	Influence
Deadlift2Kg	0.1866	negative
Deadlift1Kg	0.1802	negative
Deadlift3Kg	0.1715	negative
Best Raw Deadlift to date	0.1714	negative
Best Raw Squat to date	0.1653	negative
Best Raw Total to date	0.1650	negative
Best Raw Bench to date	0.1614	negative
Squat2Kg	0.1600	negative
Squat1Kg	0.1591	negative
Best Raw Wilks to date	0.1526	negative
Squat3Kg	0.1514	negative
Deadlift3 Jump	0.1348	positive
Bench1Kg	0.1340	negative
Bench2Kg	0.1323	negative
Bench3Kg	0.1237	negative
Number of meets in past 12 months	0.1130	negative
Bench3 Jump	0.1060	positive
Total Bench Jump	0.1041	positive
Deadlift3 Past 3 meets success rate	0.1003	positive
Number of Past Meets Same Day Exclusive	0.0979	negative
Number of Past Meets	0.0977	negative
Squat3 Jump	0.0975	positive
Overall Attempt Success Rate over past three meets	0.0956	positive
Total Deadlift Jump	0.0878	positive
Total Squat Jump	0.0853	positive
Male	0.0768	negative
Bench2 Jump	0.0710	positive
BodyweightKg	0.0634	negative
Deadlift2 Past 3 meets success rate	0.0631	positive
Squat3 Past 3 meets success rate	0.0502	positive
Bench3 Past 3 meets success rate	0.0473	positive
Squat2 Jump	0.0424	positive
84Kg Weight Class	0.0390	positive
Bench2 Past 3 meets success rate	0.0389	positive
72Kg Weight Class	0.0366	positive
Deadlift2 Jump	0.0350	positive
120+Kg Weight Class	0.0349	negative
63Kg Weight Class	0.0260	positive
105Kg Weight Class	0.0242	negative
84+Kg Weight Class	0.0240	positive
120Kg Weight Class	0.0234	negative
93Kg Weight Class	0.0220	negative
Squat2 Past 3 meets success rate	0.0203	positive
74Kg Weight Class	0.0202	negative

125+Kg Weight Class	0.0191	negative
57Kg Weight Class	0.0146	positive
Deadlift1 Past 3 meets success rate	0.0146	positive
125Kg Weight Class	0.0119	negative
52Kg Weight Class	0.0116	positive
67.5Kg Weight Class	0.0111	negative
40Kg Weight Class	0.0110	positive
75Kg Weight Class	0.0108	negative
53Kg Weight Class	0.0107	positive
43Kg Weight Class	0.0105	positive
30Kg Weight Class	0.0105	positive
35Kg Weight Class	0.0104	positive
100Kg Weight Class	0.0099	negative
90Kg Weight Class	0.0098	negative
Age	0.0092	positive
90+Kg Weight Class	0.0085	negative
60Kg Weight Class	0.0082	negative
83Kg Weight Class	0.0082	negative
Squat1 Past 3 meets success rate	0.0075	positive
110Kg Weight Class	0.0061	negative
56Kg Weight Class	0.0055	negative
66Kg Weight Class	0.0051	negative
Bench1 Past 3 meets success rate	0.0050	positive
82.5Kg Weight Class	0.0046	negative
47Kg Weight Class	0.0046	positive
44Kg Weight Class	0.0039	positive
48Kg Weight Class	0.0022	negative
59Kg Weight Class	0.0020	negative

c. Linear mixed model output

Squat1 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.786	0.014	55.8
Age	-0.002	0	-11
Overall Attempt Success Rate over past three meets	0.125	0.014	9.28
Best Raw Wilks to date	0	0	7.04
BodyweightKg	0	0	-3
Squat1Kg	0	0	2.68
Group Var	0.004	0.002	

Squat2 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.519	0.016	31.613
Squat1 Pass	0.15	0.006	23.687
Overall Attempt Success Rate over past three meets	0.227	0.017	13.533
Age	-0.001	0	-8.126
BodyweightKg	0.001	0	7.178
Squat2Kg	0	0	-5.456
Group Var	0.001	0.002	

Squat3 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.159	0.022	7.107
Squat2 Pass	0.154	0.008	20.387
BodyweightKg	0.003	0	17.279
Squat3Kg	-0.001	0	-14.749
Overall Attempt Success Rate over past three meets	0.28	0.023	11.998
Squat1 Pass	0.054	0.008	6.36
Group Var	0.007	0.003	

Bench1 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.754	0.012	62.8
Squat2 Pass	0.038	0.004	10.7
Best Raw Wilks to date	0	0	10.4
Squat1 Pass	0.04	0.004	9.47
Squat3 Pass	0.025	0.003	9.22
Bench1Kg	0	0	-7.5
Group Var	0.001	0.002	

Bench2 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.421	0.02	21.491
Squat3 Pass	0.075	0.004	17.346
Squat2 Pass	0.059	0.006	10.138
Bench2Kg	-0.001	0	-9.417
Bench1 Pass	0.077	0.008	9.394
Overall Attempt Success Rate over past three meets	0.166	0.019	8.846
Group Var	0.004	0.003	

Bench3 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	-0.019	0.025	-0.775
Bench2 Pass	0.173	0.008	21.668
Squat3 Pass	0.086	0.005	15.811
BodyweightKg	0.002	0	9.397
Total Bench Jump	-0.812	0.089	-9.177
Squat2 Pass	0.057	0.007	7.736
Group Var	0.006	0.003	

Deadlift1 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.799	0.012	65.1
Bench1 Pass	0.046	0.004	10.8
Deadlift1Kg	0	0	-7.8
Squat1 Pass	0.026	0.004	7.04
Squat2 Pass	0.022	0.003	6.94
Bench2 Pass	0.019	0.003	6.6
Group Var	0.001	0.001	

Deadlift2 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.552	0.018	30.477
Deadlift2Kg	-0.001	0	-14.889
Deadlift1 Pass	0.113	0.008	13.521
Squat3 Pass	0.045	0.004	12.63
Squat2 Pass	0.05	0.005	10.383
Bench3 Pass	0.034	0.003	9.891
Group Var	0.002	0.002	

Deadlift3 Pass Linear Mixed Effects Model			
	Coef.	Std.Err.	z
Intercept	0.243	0.027	9.13
Deadlift2 Pass	0.245	0.009	27.844
Deadlift3Kg	-0.001	0	-20.183
Squat3 Pass	0.076	0.005	14.472
Bench3 Pass	0.069	0.005	13.846
Bench2 Pass	0.063	0.007	9.313
Group Var	0.005	0.003	

Appendix H Model Scripts

a. Histogram Gradient Boosted Trees

Appendix_G_I_Hist Gradient boosting model

September 16, 2021

```
[1]: import numpy as np
from numpy.random import seed
np.random.seed(1)
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import MinMaxScaler
import graphviz
import pydot
from imblearn.tensorflow import balanced_batch_generator
from imblearn.under_sampling import RandomUnderSampler
from sklearn.utils import shuffle
from keras.regularizers import l1, l1_l2
from sklearn.inspection import permutation_importance
from sklearn.metrics import classification_report
from sklearn.preprocessing import OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier
from contextlib import redirect_stdout
from sklearn.inspection import permutation_importance
from sklearn.metrics import classification_report, roc_curve, roc_auc_score, ↴RocCurveDisplay, balanced_accuracy_score, accuracy_score
import matplotlib.pyplot as plt

openpowerlifting_USAPL_lifters = pd.read_csv("C:/Users/Montel/Google Drive/Data\u2192Science MSc/Msc-Project---Powerlifting/Datasets/Attempt subsets/Full\u2192Setnulls.csv")
openpowerlifting_USAPL_lifters['Date'] = pd.    ↴to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')

[2]: openpowerlifting_USAPL_lifters["Squat2 Jump"] = ↴(openpowerlifting_USAPL_lifters["Squat2Kg"] - ↴openpowerlifting_USAPL_lifters["Squat1Kg"])/
    ↴openpowerlifting_USAPL_lifters["Squat1Kg"]
```

```

openpowerlifting_USAPL_lifters["Squat3 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Squat3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Squat2Kg"])/
    ↳openpowerlifting_USAPL_lifters["Squat2Kg"]

openpowerlifting_USAPL_lifters["Total Squat Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Squat3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Squat1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Squat1Kg"]

openpowerlifting_USAPL_lifters["Bench2 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Bench2Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Bench1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Bench1Kg"]

openpowerlifting_USAPL_lifters["Bench3 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Bench3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Bench2Kg"])/
    ↳openpowerlifting_USAPL_lifters["Bench2Kg"]

openpowerlifting_USAPL_lifters["Total Bench Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Bench3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Bench1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Bench1Kg"]

openpowerlifting_USAPL_lifters["Deadlift2 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Deadlift2Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"]

openpowerlifting_USAPL_lifters["Deadlift3 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Deadlift3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Deadlift2Kg"])/
    ↳openpowerlifting_USAPL_lifters["Deadlift2Kg"]

openpowerlifting_USAPL_lifters["Total Deadlift Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Deadlift3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"]

openpowerlifting_USAPL_lifters =_
    ↳openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters[["Squat1Kg", 'Squat2Kg', 'Squa
    ↳"Bench1Kg", 'Bench2Kg', 'Bench3Kg',
    ↳"Deadlift1Kg", 'Deadlift2Kg', 'Deadlift3Kg']].notnull().sum(axis=1) > 3) &
    ↳(openpowerlifting_USAPL_lifters[["Squat1Kg", 'Squat2Kg', 'Squat3Kg',]].notnull().sum(axis=1) > 0) &
    ↳(openpowerlifting_USAPL_lifters[["Bench1Kg", 'Bench2Kg', 'Bench3Kg',]].notnull().sum(axis=1) > 0) &

```

```

    ↵(openpowerlifting_USAPL_lifters[["Deadlift1Kg", 'Deadlift2Kg', 'Deadlift3Kg',]] .  

    ↵notnull().sum(axis=1) > 0)

column_trans = ColumnTransformer(  

    [(['ordinal', OrdinalEncoder(dtype='str'), ['Sex', 'WeightClassKg']] ),  

     ],  

    remainder='drop')

column_trans.fit(openpowerlifting_USAPL_lifters)

openpowerlifting_USAPL_lifters.loc[:, ["Sex", "WeightClassKg"]] = column_trans.  

    ↵transform(openpowerlifting_USAPL_lifters)

#for column in ["Squat1 Pass", "Squat2 Pass", "Squat3 Pass", "Bench1 Pass",  

    ↵"Bench2 Pass", "Bench3 Pass", "Deadlift1 Pass", "Deadlift2 Pass", "Deadlift3  

    ↵Pass",]:  

#    openpowerlifting_USAPL_lifters[column] =  

    ↵openpowerlifting_USAPL_lifters[column].astype('int64')
train_valid =  

    ↵openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] < pd.  

    ↵to_datetime("11/15/2019"))]
test = openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"]  

    ↵>= pd.to_datetime("11/15/2019"))]
print(train_valid.shape[0], test.shape[0])

```

66853 15582

[3]: openpowerlifting_USAPL_lifters

	Name	Sex	Age	BodyweightKg	WeightClassKg	Squat1Kg	\
1	A'Dren Hye	1	22.5	91.1	34	240.0	
2	A'Dren Hye	1	23.5	104.1	1	260.0	
3	A.C. Alexander	1	48.5	100.1	1	160.0	
4	A.J. Bailor	1	16.5	112.9	3	205.0	
5	A.J. Howard	1	12.5	72.6	25	95.0	
...	
87444	Zsombor Gal	1	17.5	72.7	25	132.5	
87445	Zuley Sosa-Guasch	0	21.0	63.4	24	102.5	
87446	Zuleyka Weaver	0	29.5	51.2	14	77.5	
87447	Zyler Greene	1	15.5	94.6	1	165.0	
87448	Zyler Greene	1	16.5	107.2	3	167.5	
	Squat1 Pass	Squat2Kg	Squat2 Pass	Squat3Kg	...	\	
1	0.0	260.0	1.0	272.5	...		
2	0.0	272.5	1.0	287.5	...		

3	1.0	170.0	1.0	185.0	...
4	1.0	220.0	1.0	235.0	...
5	1.0	107.5	1.0	112.5	...
...
87444	1.0	137.5	1.0	142.5	...
87445	1.0	117.5	1.0	127.5	...
87446	1.0	82.5	1.0	87.5	...
87447	0.0	165.0	1.0	175.0	...
87448	1.0	180.0	1.0	187.5	...

Overall Attempt Success Rate over past three meets					Squat2 Jump	\
1				NaN	0.083333	
2				0.444444	0.048077	
3				NaN	0.062500	
4				NaN	0.073171	
5				NaN	0.131579	
...				
87444				NaN	0.037736	
87445				NaN	0.146341	
87446				NaN	0.064516	
87447				NaN	0.000000	
87448				0.444444	0.074627	

Squat3 Jump Total Squat Jump Bench2 Jump Bench3 Jump \					
1	0.048077	0.135417	0.064516	0.000000	
2	0.055046	0.105769	0.042254	0.054054	
3	0.088235	0.156250	0.054545	0.000000	
4	0.068182	0.146341	0.041667	0.080000	
5	0.046512	0.184211	0.090909	0.083333	
...	
87444	0.036364	0.075472	0.057143	0.027027	
87445	0.085106	0.243902	0.153846	0.000000	
87446	0.060606	0.129032	0.111111	0.050000	
87447	0.060606	0.060606	0.048780	0.000000	
87448	0.041667	0.119403	0.037037	0.035714	

Total Bench Jump Deadlift2 Jump Deadlift3 Jump Total Deadlift Jump					
1	0.064516	0.000000	0.008929	0.008929	
2	0.098592	0.038462	0.018519	0.057692	
3	0.054545	0.027027	0.013158	0.040541	
4	0.125000	0.076923	0.071429	0.153846	
5	0.181818	0.160000	0.068966	0.240000	
...	
87444	0.085714	0.031250	0.030303	0.062500	
87445	0.153846	0.117647	0.052632	0.176471	
87446	0.166667	0.040816	0.058824	0.102041	
87447	0.048780	0.133333	0.073529	0.216667	

```
87448      0.074074      0.073529      0.041096      0.117647
```

```
[82435 rows x 83 columns]
```

```
[4]: openpowerlifting_USAPL_lifters.shape
```

```
[4]: (82435, 83)
```

```
[5]: for column in train_valid.columns:  
    if column in ["Squat1 Pass", "Squat2 Pass", "Squat3 Pass", "Bench1  
→Pass", "Bench2 Pass", "Bench3 Pass", "Deadlift1 Pass", "Deadlift2 Pass",  
→"Deadlift3 Pass",]:  
        train_valid[column], test[column] = train_valid[column].  
→astype('int64'), test[column].astype('int64')
```

```
<ipython-input-5-3520ea648613>:3: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column], test[column] = train_valid[column].astype('int64'),  
test[column].astype('int64')
```

```
[6]: squat1_features= ['Sex', 'WeightClassKg', 'Age', 'BodyweightKg', 'Time Since Last  
→Meet',  
'Number of Past Meets', 'Number of Past Meets Same Day Exclusive', 'Number of  
→meets in past 12 months',  
'Best Raw Squat to date', 'Best Wrapped Squat to date', 'Best Raw Total to  
→date', 'Best Raw Wilks to date',  
'Squat1 Past 3 meets success rate', 'Squat2 Past 3 meets success rate', 'Squat3  
→Past 3 meets success rate',  
'Bench1 Past 3 meets success rate', 'Bench2 Past 3 meets success rate', 'Bench3  
→Past 3 meets success rate',  
'Deadlift1 Past 3 meets success rate', 'Deadlift2 Past 3 meets success rate',  
→'Deadlift3 Past 3 meets success rate',  
'Best Raw Bench to date', 'Best Raw Deadlift to date', 'Overall Attempt Success  
→Rate over past three meets',  
'Squat1Kg']  
  
squat2_features= squat1_features + ['Squat1 Pass', 'Squat2Kg', 'Squat2 Jump',]  
  
squat3_features= squat2_features + ['Squat2 Pass', 'Squat3Kg', 'Squat3  
→Jump', 'Total Squat Jump',]  
  
bench1_features= squat3_features + ['Squat3 Pass', 'Bench1Kg',]
```

```

bench2_features= bench1_features + ['Bench1 Pass','Bench2Kg','Bench2 Jump',]

bench3_features= bench2_features + ['Bench2 Pass','Bench3Kg','Bench3
→Jump','Total Bench Jump',]

deadlift1_features= bench3_features + ['Bench3 Pass','Deadlift1Kg',]

deadlift2_features= deadlift1_features + ['Deadlift1_
→Pass','Deadlift2Kg','Deadlift2 Jump',]

deadlift3_features= deadlift2_features + ['Deadlift2_
→Pass','Deadlift3Kg','Deadlift3 Jump','Total Deadlift Jump',]

```

[7]: deadlift3_features

```

[7]: ['Sex',
'WeightClassKg',
'Age',
'BodyweightKg',
'Time Since Last Meet',
'Number of Past Meets',
'Number of Past Meets Same Day Exclusive',
'Number of meets in past 12 months',
'Best Raw Squat to date',
'Best Wrapped Squat to date',
'Best Raw Total to date',
'Best Raw Wilks to date',
'Squat1 Past 3 meets success rate',
'Squat2 Past 3 meets success rate',
'Squat3 Past 3 meets success rate',
'Bench1 Past 3 meets success rate',
'Bench2 Past 3 meets success rate',
'Bench3 Past 3 meets success rate',
'Deadlift1 Past 3 meets success rate',
'Deadlift2 Past 3 meets success rate',
'Deadlift3 Past 3 meets success rate',
'Best Raw Bench to date',
'Best Raw Deadlift to date',
'Overall Attempt Success Rate over past three meets',
'Squat1Kg',
'Squat1 Pass',
'Squat2Kg',
'Squat2 Jump',
'Squat2 Pass',
'Squat3Kg',
'Squat3 Jump',

```

```
'Total Squat Jump',
'Squat3 Pass',
'Bench1Kg',
'Bench1 Pass',
'Bench2Kg',
'Bench2 Jump',
'Bench2 Pass',
'Bench3Kg',
'Bench3 Jump',
'Total Bench Jump',
'Bench3 Pass',
'Deadlift1Kg',
'Deadlift1 Pass',
'Deadlift2Kg',
'Deadlift2 Jump',
'Deadlift2 Pass',
'Deadlift3Kg',
'Deadlift3 Jump',
'Total Deadlift Jump']
```

```
[ ]:
```

```
[8]: squat1pass_train_valid = train_valid["Squat1 Pass"]
squat2pass_train_valid = train_valid["Squat2 Pass"]
squat3pass_train_valid = train_valid["Squat3 Pass"]
bench1pass_train_valid = train_valid["Bench1 Pass"]
bench2pass_train_valid = train_valid["Bench3 Pass"]
bench3pass_train_valid = train_valid["Bench3 Pass"]
deadlift1pass_train_valid = train_valid["Deadlift1 Pass"]
deadlift2pass_train_valid = train_valid["Deadlift2 Pass"]
deadlift3pass_train_valid = train_valid["Deadlift3 Pass"]

squat1input_train_valid = train_valid[squat1_features]
squat2input_train_valid = train_valid[squat2_features]
squat3input_train_valid = train_valid[squat3_features]
bench1input_train_valid = train_valid[bench1_features]
bench2input_train_valid = train_valid[bench2_features]
bench3input_train_valid = train_valid[bench3_features]
deadlift1input_train_valid = train_valid[deadlift1_features]
deadlift2input_train_valid = train_valid[deadlift2_features]
deadlift3input_train_valid = train_valid[deadlift3_features]

train_valid_inputs = [squat1input_train_valid.to_numpy(), 
                     squat2input_train_valid.to_numpy(), squat3input_train_valid.to_numpy(),
                     bench1input_train_valid.to_numpy(), bench2input_train_valid.
                     to_numpy(), bench3input_train_valid.to_numpy(),
```

```

        deadlift1input_train_valid.to_numpy(), deadlift2input_train_valid.
    ↪to_numpy(), deadlift3input_train_valid.to_numpy()]

train_valid_outputs = [squat1pass_train_valid.to_numpy(), □
    ↪squat2pass_train_valid.to_numpy(), squat3pass_train_valid.to_numpy(),
    bench1pass_train_valid.to_numpy(), bench2pass_train_valid.to_numpy(), □
    ↪bench3pass_train_valid.to_numpy(),
        deadlift1pass_train_valid.to_numpy(), deadlift2pass_train_valid.
    ↪to_numpy(), deadlift3pass_train_valid.to_numpy()]

for i in range(len(train_valid_inputs)):
    train_valid_inputs[i], train_valid_outputs[i] = □
    ↪RandomUnderSampler(random_state=42).fit_resample(train_valid_inputs[i], □
    ↪train_valid_outputs[i])

```

[]:

[9]:

```

squat1pass_test = test[pd.notna(test["Squat1Kg"])]["Squat1 Pass"]
squat2pass_test = test[pd.notna(test["Squat2Kg"])]["Squat2 Pass"]
squat3pass_test = test[pd.notna(test["Squat3Kg"])]["Squat3 Pass"]
bench1pass_test = test[pd.notna(test["Bench1Kg"])]["Bench1 Pass"]
bench2pass_test = test[pd.notna(test["Bench2Kg"])]["Bench3 Pass"]
bench3pass_test = test[pd.notna(test["Bench3Kg"])]["Bench3 Pass"]
deadlift1pass_test = test[pd.notna(test["Deadlift1Kg"])]["Deadlift1 Pass"]
deadlift2pass_test = test[pd.notna(test["Deadlift2Kg"])]["Deadlift2 Pass"]
deadlift3pass_test = test[pd.notna(test["Deadlift3Kg"])]["Deadlift3 Pass"]

squat1input_test = test[pd.notna(test["Squat1Kg"])][squat1_features]
squat2input_test = test[pd.notna(test["Squat2Kg"])][squat2_features]
squat3input_test = test[pd.notna(test["Squat3Kg"])][squat3_features]
bench1input_test = test[pd.notna(test["Bench1Kg"])][bench1_features]
bench2input_test = test[pd.notna(test["Bench2Kg"])][bench2_features]
bench3input_test = test[pd.notna(test["Bench3Kg"])][bench3_features]
deadlift1input_test = test[pd.notna(test["Deadlift1Kg"])][deadlift1_features]
deadlift2input_test = test[pd.notna(test["Deadlift2Kg"])][deadlift2_features]
deadlift3input_test = test[pd.notna(test["Deadlift3Kg"])][deadlift3_features]

test_inputs = [squat1input_test.to_numpy(), squat2input_test.to_numpy(), □
    ↪squat3input_test.to_numpy(),
        bench1input_test.to_numpy(), bench2input_test.to_numpy(), □
    ↪bench3input_test.to_numpy(),
        deadlift1input_test.to_numpy(), deadlift2input_test.to_numpy(), □
    ↪deadlift3input_test.to_numpy()]

```

```

test_outputs = [squat1pass_test.to_numpy(), squat2pass_test.to_numpy(),
                ↳squat3pass_test.to_numpy(),
                bench1pass_test.to_numpy(), bench2pass_test.to_numpy(),
                ↳bench3pass_test.to_numpy(),
                deadlift1pass_test.to_numpy(), deadlift2pass_test.to_numpy(),
                ↳deadlift3pass_test.to_numpy()]

```

```

[15]: import keras_tuner as kt
import sklearn as sklearn
model_names = ["Squat1 Pass", "Squat2 Pass", "Squat3 Pass", "Bench1 Pass",
                ↳"Bench2 Pass", "Bench3 Pass", "Deadlift1 Pass", "Deadlift2 Pass", "Deadlift3
                ↳Pass",]

feature_indices = [squat1_features, squat2_features, squat3_features,
                ↳bench1_features, bench2_features, bench3_features,
                deadlift1_features, deadlift2_features, deadlift3_features]

tuner = None

number = 0
for i,j in zip(train_valid_inputs, train_valid_outputs):
    def model_builder(hp):
        model = HistGradientBoostingClassifier(learning_rate= hp.
            ↳Float("learning rate", 0.01, 1.0, sampling = "log"),
            max_iter = hp.Int("maximum
            ↳iterations", 1, 1000, sampling = "log"),
            random_state = 42,
            max_leaf_nodes = hp.Int("maximum
            ↳leaf nodes", 2, 1000, sampling = "log"),
            max_depth = hp.Int("max depth", ↳
            ↳1, 100, sampling = "log"),
            min_samples_leaf = hp.Int("min
            ↳samples per leaf", 2, 1000, sampling = "log"),
            l2_regularization= hp.
            ↳Float("l2", 0.01, 1.0, sampling = "log"),
            scoring = "accuracy",
            validation_fraction = 0.4,
            ↳categorical_features = [0,1], verbose = 3,
            max_bins = hp.Int("Max bins", 40,
            ↳255, 20),
            early_stopping = hp.
            ↳Boolean("early stopping"))

        return model

```

```

tuner = kt.tuners.SklearnTuner(hypermodel = model_builder, oracle = kt.
↪oracles.BayesianOptimization(objective = kt.Objective('score', 'max'), ↪
↪max_trials = 100),
                                scoring = sklearn.metrics.
↪make_scorer(sklearn.metrics.accuracy_score), cv = sklearn.model_selection.
↪StratifiedKFold(2),
                                directory = "hist grad boost " + ↪
↪model_names[number], project_name = "tests", overwrite = True)

tuner.search(i, j)

best_model = tuner.get_best_models(num_models=1)[0]
best_model.fit(i,j)
y_pred = best_model.predict(test_inputs[number])
y_scores= best_model.predict_proba(test_inputs[number])[:,1]
y_true = test_outputs[number]

fpr, tpr, thresholds = roc_curve(y_true, y_scores)
area_under_curve = roc_auc_score(y_true, y_scores)
display = RocCurveDisplay(fpr = fpr, tpr = tpr, roc_auc=area_under_curve)
display.plot(name = f"{model_names[number]} hist grad booster")
plt.savefig(f"{model_names[number]} hist grad booster ROC curve.pdf")

with open(f"{model_names[number]} hist grad booster classification_report.
↪txt", 'w') as f:
    with redirect_stdout(f):
        print(classification_report(y_true, y_pred))

with open(f"{model_names[number]} hist grad boost results.txt", 'w') as f:
    with redirect_stdout(f):
        tuner.results_summary()

r = permutation_importance(best_model, test_inputs[number], y_true,
                           n_repeats=5,
                           random_state=10, scoring = "balanced_accuracy")
feature_index = []
importance_means = []
importance_sds = []
for k in r.importances_mean.argsort()[:-1]:
    feature_index.append(feature_indices[number][k])
    importance_means.append(r.importances_mean[k])
    importance_sds.append(r.importances_std[k])

feature_importances = pd.DataFrame(data = {"mean_importance": ↪
↪importance_means, "standard deviation": importance_sds}, index = ↪
↪feature_index )

```

```
    feature_importances.to_csv(f"model_names[number]} hist grad booster"
    ↪feature importances.csv")
```

```
number += 1
```

```
Trial 38 Complete [00h 00m 00s]
score: 0.58584229390681
```

```
Best score So Far: 0.5859617682198328
Total elapsed time: 00h 00m 38s
```

```
Search: Running Trial #39
```

Hyperparameter	Value	Best Value So Far
learning rate	0.11517	0.29832
maximum iterations	30	28
maximum leaf nodes	293	2
max depth	1	13
min samples per... 28	2	
12	1	0.41035
Max bins	240	200
early stopping	True	False

```
Binning 0.001 GB of training data: 0.009 s
Binning 0.001 GB of validation data: 0.002 s
```

```
Fitting gradient boosted rounds:
```

```
[1/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57507, val score:
0.55944, in 0.003s
```

```
[2/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57507, val score:
0.55944, in 0.004s
```

```
[3/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57507, val score:
0.55944, in 0.004s
```

```
[4/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57507, val score:
0.55944, in 0.004s
```

```
[5/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57507, val score:
0.55944, in 0.004s
```

```
[6/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57507, val score:
0.55944, in 0.004s
```

```
[7/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57507, val score:
0.55944, in 0.005s
```

```
[8/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57746, val score:
0.57109, in 0.004s
```

```
[9/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57806, val score:
0.57168, in 0.005s
```

```
[10/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57865, val score:
0.57557, in 0.005s
```

```
[11/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57865, val score:
0.57557, in 0.005s
```

```
[12/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57965, val score:  
0.57616, in 0.005s  
[13/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57905, val score:  
0.57616, in 0.006s  
[14/30] 1 tree, 2 leaves, max depth = 1, train score: 0.57865, val score:  
0.57646, in 0.006s  
[15/30] 1 tree, 2 leaves, max depth = 1, train score: 0.58343, val score:  
0.58005, in 0.007s  
[16/30] 1 tree, 2 leaves, max depth = 1, train score: 0.58602, val score:  
0.57885, in 0.008s  
[17/30] 1 tree, 2 leaves, max depth = 1, train score: 0.58781, val score:  
0.58124, in 0.007s  
[18/30] 1 tree, 2 leaves, max depth = 1, train score: 0.58662, val score:  
0.58065, in 0.007s  
[19/30] 1 tree, 2 leaves, max depth = 1, train score: 0.58801, val score:  
0.58094, in 0.006s  
[20/30] 1 tree, 2 leaves, max depth = 1, train score: 0.58781, val score:  
0.58065, in 0.007s  
[21/30] 1 tree, 2 leaves, max depth = 1, train score: 0.58921, val score:  
0.58154, in 0.007s  
[22/30] 1 tree, 2 leaves, max depth = 1, train score: 0.59080, val score:  
0.57855, in 0.007s  
[23/30] 1 tree, 2 leaves, max depth = 1, train score: 0.59000, val score:  
0.57945, in 0.007s  
[24/30] 1 tree, 2 leaves, max depth = 1, train score: 0.59000, val score:  
0.58035, in 0.009s  
[25/30] 1 tree, 2 leaves, max depth = 1, train score: 0.59020, val score:  
0.58035, in 0.008s  
[26/30] 1 tree, 2 leaves, max depth = 1, train score: 0.59100, val score:  
0.58005, in 0.008s  
[27/30] 1 tree, 2 leaves, max depth = 1, train score: 0.59100, val score:  
0.58094, in 0.009s  
[28/30] 1 tree, 2 leaves, max depth = 1, train score: 0.59100, val score:  
0.58184, in 0.008s  
[29/30] 1 tree, 2 leaves, max depth = 1, train score: 0.59299, val score:  
0.58244, in 0.009s  
[30/30] 1 tree, 2 leaves, max depth = 1, train score: 0.59239, val score:  
0.58154, in 0.008s  
Fit 30 trees in 0.227 s, (60 total leaves)  
Time spent computing histograms: 0.000s  
Time spent finding best splits: 0.000s  
Time spent applying splits: 0.003s  
Time spent predicting: 0.002s
```

```
KeyboardInterrupt
```

```
Traceback (most recent call last)
```

```
<ipython-input-15-72c65ccc9774> in <module>
```

```

      30                     directory = "hist grad boost " +_
→model_names[number], project_name = "tests", overwrite = True)
      31
---> 32     tuner.search(i, j)
      33
      34     best_model = tuner.get_best_models(num_models=1)[0]

~\anaconda3\lib\site-packages\keras_tuner\tuners\sklearn_tuner.py in_
→search(self, X, y, sample_weight, groups)
 143         """
 144         # Only overridden for the docstring.
--> 145         return super().search(X, y, sample_weight=sample_weight,_
→groups=groups)
 146
 147     def run_trial(self, trial, X, y, sample_weight=None, groups=None):

~\anaconda3\lib\site-packages\keras_tuner\engine\base_tuner.py in search(self,_
→*fit_args, **fit_kwargs)
 174
 175         self.on_trial_begin(trial)
--> 176         self.run_trial(trial, *fit_args, **fit_kwargs)
 177         self.on_trial_end(trial)
 178         self.on_search_end()

~\anaconda3\lib\site-packages\keras_tuner\tuners\sklearn_tuner.py in_
→run_trial(self, trial, X, y, sample_weight, groups)
 173             score = model.score(X_test, y_test,_
→sample_weight=sample_weight_test)
 174         else:
--> 175             score = self.scoring(
 176                 model, X_test, y_test,_
→sample_weight=sample_weight_test
 177             )

~\anaconda3\lib\site-packages\sklearn\metrics\_scorer.py in __call__(self,_
→estimator, X, y_true, sample_weight)
 197             Score function applied to prediction of estimator on X.
 198             """
--> 199             return self._score(partial(_cached_call, None), estimator, X,_
→y_true,
 200                                         sample_weight=sample_weight)
 201

~\anaconda3\lib\site-packages\sklearn\metrics\_scorer.py in _score(self,_
→method_caller, estimator, X, y_true, sample_weight)
 234             """
 235

```

```

--> 236         y_pred = method_caller(estimator, "predict", X)
237         if sample_weight is not None:
238             return self._sign * self._score_func(y_true, y_pred,
239
~\anaconda3\lib\site-packages\sklearn\metrics\_scorer.py in _cached_call(cache,
→ estimator, method, *args, **kwargs)
51     """Call estimator with method and args and kwargs."""
52     if cache is None:
---> 53         return getattr(estimator, method)(*args, **kwargs)
54
55     try:
56
~\anaconda3\lib\site-packages\sklearn\ensemble\_hist_gradient_boosting\gradient_boosting.
→ py in predict(self, X)
1339         """
140         # TODO: This could be done in parallel
-> 141         encoded_classes = np.argmax(self.predict_proba(X), axis=1)
142         return self.classes_[encoded_classes]
143
144
~\anaconda3\lib\site-packages\sklearn\ensemble\_hist_gradient_boosting\gradient_boosting.
→ py in predict_proba(self, X)
1377             The class probabilities of the input samples.
1378             """
-> 1379             raw_predictions = self._raw_predict(X)
1380             return self._loss.predict_proba(raw_predictions)
1381
1382
~\anaconda3\lib\site-packages\sklearn\ensemble\_hist_gradient_boosting\gradient_boosting.
→ py in _raw_predict(self, X)
748             )
749             raw_predictions += self._baseline_prediction
--> 750             self._predict_iterations(
751                 X, self._predictors, raw_predictions, is_binned
752             )
753
754
~\anaconda3\lib\site-packages\sklearn\ensemble\_hist_gradient_boosting\gradient_boosting.
→ py in _predict_iterations(self, X, predictors, raw_predictions, is_binned)
771                     known_cat_bitsets=known_cat_bitsets,
772                     f_idx_map=f_idx_map)
--> 773             raw_predictions[k, :] += predict(X)
774
775     def _staged_raw_predict(self, X):
776
~\anaconda3\lib\site-packages\sklearn\ensemble\_hist_gradient_boosting\predictor.
→ py in predict(self, X, known_cat_bitsets, f_idx_map)
64             """

```

```

65         out = np.empty(X.shape[0], dtype=Y_DTYPE)
---> 66     _predict_from_raw_data(self.nodes, X, self.raw_left_cat_bitsets
67                         known_cat_bitsets, f_idx_map, out)
68     return out

```

KeyboardInterrupt:

```
[ ]: for i in test_outputs:
    print( i.mean())
```

1 Scrap code

```
[ ]: # from keras.regularizers import l1_l2

# squat1_input = keras.Input(shape = (len(squat1_features),), name = "Initial"
# Variables and Squat1")
# squat1_hidden = layers.Dense(len(squat1_features)//2, activation = "relu",)
# squat1_hidden_2 = layers.Dense(3, activation = "relu", name =
# "Squat1_hidden_2")(squat1_hidden)
# squat1_output = layers.Dense(1, activation = "sigmoid", name =
# "Squat1_result")(squat1_hidden_2)

# squat2_input = keras.Input(shape = (len(squat2_features),), name = "Squat2"
# Input")
# squat2_hidden = layers.Dense(squat2_input.shape[1]//2, activation = "relu",)
# squat2_hidden_2 = layers.Dense(3, activation = "relu", name =
# "Squat2_hidden_2")(squat2_hidden)
# squat2_output = layers.Dense(1, activation = "sigmoid", name =
# "Squat2_result")(squat2_hidden_2)

# squat3_input = keras.Input(shape = (len(squat3_features),), name = "Squat3"
# Input")
# squat3_hidden = layers.Dense(squat3_input.shape[1]//2, activation = "relu",)
# squat3_hidden_2 = layers.Dense(3, activation = "relu", name =
# "Squat3_hidden_2")(squat3_hidden)
# squat3_output = layers.Dense(1, activation = "sigmoid", name =
# "Squat3_result")(squat3_hidden_2)

# bench1_input = keras.Input(shape = (len(bench1_features),), name = "Bench1"
# Input")
```

```

# bench1_hidden = layers.Dense(bench1_input.shape[1]//2, activation = "relu",
#                               name = "Bench1_hidden")(bench1_input)
# bench1_hidden_2 = layers.Dense(3, activation = "relu", name =
#                                "bench1_hidden_2")(bench1_hidden)
# bench1_output = layers.Dense(1, activation = "sigmoid", name =
#                                "Bench1_result")(bench1_hidden_2)

# bench2_input = keras.Input(shape = (len(bench2_features),), name = "Bench2"
#                            Input")
# bench2_hidden = layers.Dense(bench2_input.shape[1]//2, activation = "relu",
#                               name = "Bench2_hidden")(bench2_input)
# bench2_hidden_2 = layers.Dense(3, activation = "relu", name =
#                                "bench2_hidden_2")(bench2_hidden)
# bench2_output = layers.Dense(1, activation = "sigmoid", name =
#                                "Bench2_result")(bench2_hidden_2)

# bench3_input = keras.Input(shape = (len(bench3_features),), name = "Bench3"
#                            Input")
# bench3_hidden = layers.Dense(bench3_input.shape[1]//2, activation = "relu",
#                               name = "Bench3_hidden")(bench3_input)
# bench3_hidden_2 = layers.Dense(3, activation = "relu", name =
#                                "bench3_hidden_2")(bench3_hidden)
# bench3_output = layers.Dense(1, activation = "sigmoid", name =
#                                "Bench3_result")(bench3_hidden_2)

# deadlift1_input = keras.Input(shape = (len(deadlift1_features),), name =
#                                "Deadlift1 Input")
# deadlift1_hidden = layers.Dense(deadlift1_input.shape[1]//2, activation =
#                                 "relu", name = "Deadlift1_hidden")(deadlift1_input)
# deadlift1_hidden_2 = layers.Dense(3, activation = "relu", name =
#                                "deadlift1_hidden_2")(deadlift1_hidden)
# deadlift1_output = layers.Dense(1, activation = "sigmoid", name =
#                                "Deadlift1_result")(deadlift1_hidden_2)

# deadlift2_input = keras.Input(shape = (len(deadlift2_features),), name =
#                                "Deadlift2 Input")
# deadlift2_hidden = layers.Dense(deadlift2_input.shape[1]//2, activation =
#                                 "relu", name = "Deadlift2_hidden")(deadlift2_input)
# deadlift2_hidden_2 = layers.Dense(3, activation = "relu", name =
#                                "deadlift2_hidden_2")(deadlift2_hidden)
# deadlift2_output = layers.Dense(1, activation = "sigmoid", name =
#                                "Deadlift2_result")(deadlift2_hidden_2)

# deadlift3_input = keras.Input(shape = (len(deadlift3_features),), name =
#                                "Deadlift3 Input")

```

```

# deadlift3_hidden = layers.Dense(deadlift3_input.shape[1]//2, activation = "relu", name = "Deadlift3_hidden")(deadlift3_input)
# deadlift3_hidden_2 = layers.Dense(3, activation = "relu", name = "deadlift3_hidden_2")(deadlift3_hidden)
# deadlift3_output = layers.Dense(1, activation = "sigmoid", name = "Deadlift3_result")(deadlift3_hidden_2)

# model = keras.Model(
#     inputs=[squat1_input, squat2_input, squat3_input,
#             bench1_input, bench2_input, bench3_input,
#             deadlift1_input, deadlift2_input, deadlift3_input,],
#     outputs=[squat1_output, squat2_output, squat3_output,
#             bench1_output, bench2_output, bench3_output,
#             deadlift1_output, deadlift2_output, deadlift3_output,],
# )
# model_inputs = dict(zip(model.input_names, train_valid_inputs))
# model_outputs = dict(zip(model.output_names, train_valid_outputs))

```

```

[ ]: # np.random.seed(100)
# model.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.0001), loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
# keras.metrics.FalsePositives(),])

# model.fit(model_inputs,
#           model_outputs, verbose = 1, epochs = 100, validation_split=0.2, batch_size = 40000)

```

```

[ ]: # print("squat1")

# model_squat1 = keras.Model(
#     inputs = squat1_input, outputs = squat1_output)

# model_squat1_inputs = dict(zip(model_squat1.input_names, [train_valid_inputs[0]]))
# model_squat1_outputs = dict(zip(model_squat1.output_names, [train_valid_outputs[0]]))

# model_squat1.compile(optimizer = 'adam', loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
# keras.metrics.FalsePositives(),])

# model_squat1.fit(model_squat1_inputs,
#                   model_squat1_outputs, verbose = 1, epochs = 100, validation_split=0.2, batch_size = 5000)

# print("squat2")

```

```

# model_squat2 = keras.Model(
#     inputs=[squat2_input],
#     outputs=[squat2_output],
# )
# model_squat2_inputs = dict(zip(model_squat2.input_names, [
#     [train_valid_inputs[1]]]))
# model_squat2_outputs = dict(zip(model_squat2.output_names, [
#     [train_valid_outputs[1]]]))

# model_squat2.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.001),
#     loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
#     keras.metrics.FalsePositives(),])

# model_squat2.fit(model_squat2_inputs,
#     model_squat2_outputs, verbose = 1, epochs = 100, validation_split=0.
#     2, batch_size = 5000)

# print("squat3")

# model_squat3 = keras.Model(
#     inputs=[squat3_input],
#     outputs=[squat3_output],
# )
# model_squat3_inputs = dict(zip(model_squat3.input_names, [
#     [train_valid_inputs[2]]]))
# model_squat3_outputs = dict(zip(model_squat3.output_names, [
#     [train_valid_outputs[2]]]))
# model_squat3.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.001),
#     loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
#     keras.metrics.FalsePositives(),])

# model_squat3.fit(model_squat3_inputs,
#     model_squat3_outputs, verbose = 1, epochs = 100, validation_split=0.
#     2, batch_size = 5000)

# print("bench1")

# model_bench1 = keras.Model(
#     inputs=[ bench1_input],
#     outputs=[
#         bench1_output],
# )
# model_bench1_inputs = dict(zip(model_bench1.input_names, [
#     [train_valid_inputs[3]])))

```

```

# model_bench1_outputs = dict(zip(model_bench1.output_names, [
#     [train_valid_outputs[3]]))
# model_bench1.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.001),
#     loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
#         keras.metrics.FalsePositives(),])
# model_bench1.fit(model_bench1_inputs,
#     model_bench1_outputs, verbose = 1, epochs = 100, validation_split=0.
#     2, batch_size = 5000)
# print("bench2")

# model_bench2 = keras.Model(
#     inputs=[bench2_input],
#     outputs=[bench2_output],
# )
# model_bench2_inputs = dict(zip(model_bench2.input_names, [
#     [train_valid_inputs[4]]))
# model_bench2_outputs = dict(zip(model_bench2.output_names, [
#     [train_valid_outputs[4]])))
# model_bench2.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.001),
#     loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
#         keras.metrics.FalsePositives(),])
# model_bench2.fit(model_bench2_inputs,
#     model_bench2_outputs, verbose = 1, epochs = 100, validation_split=0.
#     2, batch_size = 5000)
# print("bench3")

# model_bench3 = keras.Model(
#     inputs=[bench3_input],
#     outputs=[bench3_output],
# )
# model_bench3_inputs = dict(zip(model_bench3.input_names, [
#     [train_valid_inputs[5]]))
# model_bench3_outputs = dict(zip(model_bench3.output_names, [
#     [train_valid_outputs[5]])))
# model_bench3.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.001),
#     loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
#         keras.metrics.FalsePositives(),])
# model_bench3.fit(model_bench3_inputs,

```

```

#           model_bench3_outputs, verbose = 1, epochs = 100, validation_split=0.
#           ↵2, batch_size = 5000)

# print("deadlift1")

# model_deadlift1 = keras.Model(
#     inputs=[deadlift1_input],
#     outputs=[
#         deadlift1_output],
# )
# model_deadlift1_inputs = dict(zip(model_deadlift1.input_names, [
#     ↵[train_valid_inputs[6]])))
# model_deadlift1_outputs = dict(zip(model_deadlift1.output_names, [
#     ↵[train_valid_outputs[6]])))
# model_deadlift1.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.
#     ↵001), loss = "binary_crossentropy", metrics=[tf.keras.metrics.
#     ↵BinaryAccuracy(),
#           keras.metrics.FalsePositives(),])

# model_deadlift1.fit(model_deadlift1_inputs,
#     model_deadlift1_outputs, verbose = 1, epochs = 100, validation_split=0.2, batch_size = 5000)

# print("deadlift2")

# model_deadlift2 = keras.Model(
#     inputs=[deadlift2_input],
#     outputs=[deadlift2_output],
# )
# model_deadlift2_inputs = dict(zip(model_deadlift2.input_names, [
#     ↵[train_valid_inputs[7]])))
# model_deadlift2_outputs = dict(zip(model_deadlift2.output_names, [
#     ↵[train_valid_outputs[7]]]))
# model_deadlift2.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.
#     ↵001), loss = "binary_crossentropy", metrics=[tf.keras.metrics.
#     ↵BinaryAccuracy(),
#           keras.metrics.FalsePositives(),])

# model_deadlift2.fit(model_deadlift2_inputs,
#     model_deadlift2_outputs, verbose = 1, epochs = 100, validation_split=0.2, batch_size = 5000)

# print("deadlift3")

# model_deadlift3 = keras.Model(
#     inputs=[deadlift3_input],

```

```

#      outputs=[deadlift3_output],
# )

# model_deadlift3_inputs = dict(zip(model_deadlift3.input_names, □
# → [train_valid_inputs[8]]))
# model_deadlift3_outputs = dict(zip(model_deadlift3.output_names, □
# → [train_valid_outputs[8]]))
# model_deadlift3.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.
# → 001), loss = "binary_crossentropy", metrics=[tf.keras.metrics.
# → BinaryAccuracy(),
#           keras.metrics.FalsePositives(),])
# model_deadlift3.fit(model_deadlift3_inputs,
#                     model_deadlift3_outputs, verbose = 1, epochs = 100, □
# → validation_split=0.2, batch_size = 5000)

```

```
[ ]: #keras.utils.plot_model(model, "multi_input_and_output_model.png", rankdir = □
# → "LR", show_shapes=True)
```

```
[ ]: # model_inputs = dict(zip(model.input_names, train_valid_inputs))
# model_outputs = dict(zip(model.output_names, train_valid_outputs))
```

b. Multi-Layer perceptron

Appendix_G_HI_Deep Learning Model

September 16, 2021

```
[1]: import numpy as np
from numpy.random import seed
np.random.seed(1)
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import graphviz
import pydot
from imblearn.tensorflow import balanced_batch_generator
from imblearn.under_sampling import RandomUnderSampler
from sklearn.utils import shuffle
from keras.regularizers import l1, l1_l2
from contextlib import redirect_stdout
from sklearn.inspection import permutation_importance
from sklearn.metrics import classification_report, roc_curve, roc_auc_score,roc
    →CurveDisplay, balanced_accuracy_score, accuracy_score
from keras.wrappers.scikit_learn import KerasClassifier

openpowerlifting_USAPL_lifters = pd.read_csv("C:/Users/Montel/Google Drive/Data
    →Science MSc/Msc-Project---Powerlifting/Datasets/Attempt subsets/Full Set.
    →csv")
openpowerlifting_USAPL_lifters['Date'] = pd.
    →to_datetime(openpowerlifting_USAPL_lifters['Date'], format='%Y-%m-%d')
```

```
[2]: openpowerlifting_USAPL_lifters["Squat2 Jump"] =
    →(openpowerlifting_USAPL_lifters["Squat2Kg"] -
    →openpowerlifting_USAPL_lifters["Squat1Kg"])/
    →openpowerlifting_USAPL_lifters["Squat1Kg"]
openpowerlifting_USAPL_lifters["Squat3 Jump"] =
    →(openpowerlifting_USAPL_lifters["Squat3Kg"] -
    →openpowerlifting_USAPL_lifters["Squat2Kg"])/
    →openpowerlifting_USAPL_lifters["Squat2Kg"]
```

```

openpowerlifting_USAPL_lifters["Total Squat Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Squat3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Squat1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Squat1Kg"]

openpowerlifting_USAPL_lifters["Bench2 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Bench2Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Bench1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Bench1Kg"]

openpowerlifting_USAPL_lifters["Bench3 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Bench3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Bench2Kg"])/
    ↳openpowerlifting_USAPL_lifters["Bench2Kg"]

openpowerlifting_USAPL_lifters["Total Bench Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Bench3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Bench1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Bench1Kg"]

openpowerlifting_USAPL_lifters["Deadlift2 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Deadlift2Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"]

openpowerlifting_USAPL_lifters["Deadlift3 Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Deadlift3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Deadlift2Kg"])/
    ↳openpowerlifting_USAPL_lifters["Deadlift2Kg"]

openpowerlifting_USAPL_lifters["Total Deadlift Jump"] =_
    ↳(openpowerlifting_USAPL_lifters["Deadlift3Kg"] -_
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"])/
    ↳openpowerlifting_USAPL_lifters["Deadlift1Kg"]

weight_class_cols = pd.
    ↳get_dummies(openpowerlifting_USAPL_lifters["WeightClassKg"], prefix =_
    ↳"WeightClassKg").columns

openpowerlifting_USAPL_lifters =_
    ↳openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters[["Squat1Kg", 'Squat2Kg', 'Squa
    ↳"Bench1Kg", 'Bench2Kg', 'Bench3Kg',
    ↳"Deadlift1Kg", 'Deadlift2Kg', 'Deadlift3Kg']].notnull().sum(axis=1) > 3) &
    ↳(openpowerlifting_USAPL_lifters[["Squat1Kg", 'Squat2Kg', 'Squat3Kg',]].notnull().sum(axis=1) >
    ↳(openpowerlifting_USAPL_lifters[["Bench1Kg", 'Bench2Kg', 'Bench3Kg',]].notnull().sum(axis=1) >

```

```

→(openpowerlifting_USAPL_lifters[["Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg",]] .
→notnull().sum(axis=1) > 0)]
openpowerlifting_USAPL_lifters = pd.get_dummies(openpowerlifting_USAPL_lifters, u
→columns = ["WeightClassKg"])

for column in ["Squat1 Pass", "Squat2 Pass", "Squat3 Pass", "Bench1 Pass", u
→"Bench2 Pass", "Bench3 Pass", "Deadlift1 Pass", "Deadlift2 Pass", "Deadlift3 u
→Pass",]:
    openpowerlifting_USAPL_lifters[column] = u
    →openpowerlifting_USAPL_lifters[column].astype('int64')
train_valid = u
→openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] < pd.
→to_datetime("11/15/2019"))]
test = openpowerlifting_USAPL_lifters[(openpowerlifting_USAPL_lifters["Date"] u
→>= pd.to_datetime("11/15/2019"))]
print(train_valid.shape[0], test.shape[0])

```

66853 15582

```

[3]: for column in train_valid.columns:
    if train_valid[column].dtype == int or u
    →openpowerlifting_USAPL_lifters[column].dtype == float:
        scaler = MinMaxScaler(copy = True).fit(np.array(train_valid[column]) .
    →reshape(-1,1))
        train_valid[column] = scaler.transform(np.array(train_valid[column]) .
    →reshape(-1,1))
        test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
        if column in ["Squat1 Pass", "Squat2 Pass", "Squat3 Pass", "Bench1 u
    →Pass", "Bench2 Pass", "Bench3 Pass", "Deadlift1 Pass", "Deadlift2 Pass", u
    →"Deadlift3 Pass",]:
            train_valid[column], test[column] = train_valid[column].
    →astype('int64'), test[column].astype('int64')

```

<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```

train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
```

```
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
C:\Users\Montel\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py:400:  
RuntimeWarning: All-NaN slice encountered  
    data_min = np.nanmin(X, axis=0)
```

```
C:\Users\Montel\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py:401:  
RuntimeWarning: All-NaN slice encountered  
    data_max = np.nanmax(X, axis=0)  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
```

```
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
C:\Users\Montel\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py:400:  
RuntimeWarning: All-NaN slice encountered  
    data_min = np.nanmin(X, axis=0)  
C:\Users\Montel\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py:401:  
RuntimeWarning: All-NaN slice encountered  
    data_max = np.nanmax(X, axis=0)  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
```

```
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
C:\Users\Montel\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py:400:  
RuntimeWarning: All-NaN slice encountered  
    data_min = np.nanmin(X, axis=0)  
C:\Users\Montel\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py:401:  
RuntimeWarning: All-NaN slice encountered  
    data_max = np.nanmax(X, axis=0)  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
```

```
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
```

```
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    train_valid[column] =
scaler.transform(np.array(train_valid[column]).reshape(-1,1))
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
```

```
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train_valid[column] =  
scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    train_valid[column] =  
    scaler.transform(np.array(train_valid[column]).reshape(-1,1))  
<ipython-input-3-10047ff00971>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-  
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
    test[column] = scaler.transform(np.array(test[column]).reshape(-1,1))
```

[4]: weight_class_cols

[4]: Index(['WeightClassKg_100', 'WeightClassKg_105', 'WeightClassKg_110',
 'WeightClassKg_120', 'WeightClassKg_120+', 'WeightClassKg_125',
 'WeightClassKg_125+', 'WeightClassKg_30', 'WeightClassKg_35',
 'WeightClassKg_40', 'WeightClassKg_43', 'WeightClassKg_44',
 'WeightClassKg_47', 'WeightClassKg_48', 'WeightClassKg_52',
 'WeightClassKg_53', 'WeightClassKg_56', 'WeightClassKg_57',
 'WeightClassKg_59', 'WeightClassKg_60', 'WeightClassKg_63',
 'WeightClassKg_66', 'WeightClassKg_67.5', 'WeightClassKg_69',

```
'WeightClassKg_72', 'WeightClassKg_74', 'WeightClassKg_75',
'WeightClassKg_76', 'WeightClassKg_82.5', 'WeightClassKg_83',
'WeightClassKg_84', 'WeightClassKg_84+', 'WeightClassKg_90',
'WeightClassKg_90+', 'WeightClassKg_93'],
dtype='object')
```

```
[5]: squat1_features= ['Male',
'Female',
'Age',
'BodyweightKg',
'Time Since Last Meet',
'Number of Past Meets',
'Number of Past Meets Same Day Exclusive',
'Number of meets in past 12 months',
'Best Raw Squat to date',
'Best Wrapped Squat to date',
'Best Raw Total to date',
'Best Raw Wilks to date',
'Squat1 Past 3 meets success rate',
    'Squat2 Past 3 meets success rate',
    'Squat3 Past 3 meets success rate',
    'Bench1 Past 3 meets success rate',
    'Bench2 Past 3 meets success rate',
    'Bench3 Past 3 meets success rate',
    'Deadlift1 Past 3 meets success rate',
    'Deadlift2 Past 3 meets success rate',
    'Deadlift3 Past 3 meets success rate',
    'Best Raw Bench to date',
    'Best Raw Deadlift to date',
'Overall Attempt Success Rate over past three meets',
'Squat1Kg'
] + list(weight_class_cols)
squat2_features= list(set(squat1_features + [
    'Squat1 Pass',
    'Squat2Kg',
    'Squat2 Jump',
])))

squat3_features= list(set(squat2_features + ['Squat2 Pass',
'Squat3Kg',
'Squat3 Jump',
'Total Squat Jump',
])))

bench1_features= list(set(squat3_features + ['Squat3 Pass',
```

```

'Bench1Kg' ,  
  

]))  
  

bench2_features= list(set(bench1_features + ['Bench1 Pass' ,  

'Bench2Kg' ,  

'Bench2 Jump' ,  
  

]))  
  

bench3_features= list(set(bench2_features + ['Bench2 Pass' ,  

'Bench3Kg' ,  

'Bench3 Jump' ,  

'Total Bench Jump' ,  
  

]))  
  

deadlift1_features= list(set(bench3_features + ['Bench3 Pass' ,  

'Deadlift1Kg' ,  
  

]))  
  

deadlift2_features= list(set(deadlift1_features + ['Deadlift1 Pass' ,  

'Deadlift2Kg' ,  

'Deadlift2 Jump' ,  
  

]))  
  

deadlift3_features= list(set(deadlift2_features + ['Deadlift2 Pass' ,  

'Deadlift3Kg' ,  

'Deadlift3 Jump' ,  

'Total Deadlift Jump' ,  
  

]))
```

[6]: squat1pass_train_valid = train_valid[pd.notna(train_valid["Squat1Kg"])]["Squat1→Pass"]
squat2pass_train_valid = train_valid[pd.notna(train_valid["Squat2Kg"])]["Squat2→Pass"]
squat3pass_train_valid = train_valid[pd.notna(train_valid["Squat3Kg"])]["Squat3→Pass"]
bench1pass_train_valid = train_valid[pd.notna(train_valid["Bench1Kg"])]["Bench1→Pass"]
bench2pass_train_valid = train_valid[pd.notna(train_valid["Bench2Kg"])]["Bench3→Pass"]

```

bench3pass_train_valid = train_valid[pd.notna(train_valid["Bench3Kg"])]["Bench3\u2192Pass"]
deadlift1pass_train_valid = train_valid[pd.
    \u2192notna(train_valid["Deadlift1Kg"])]["Deadlift1 Pass"]
deadlift2pass_train_valid = train_valid[pd.
    \u2192notna(train_valid["Deadlift2Kg"])]["Deadlift2 Pass"]
deadlift3pass_train_valid = train_valid[pd.
    \u2192notna(train_valid["Deadlift3Kg"])]["Deadlift3 Pass"]

squat1input_train_valid = train_valid[pd.
    \u2192notna(train_valid["Squat1Kg"])] [squat1_features]
squat2input_train_valid = train_valid[pd.
    \u2192notna(train_valid["Squat2Kg"])] [squat2_features]
squat3input_train_valid = train_valid[pd.
    \u2192notna(train_valid["Squat3Kg"])] [squat3_features]
bench1input_train_valid = train_valid[pd.
    \u2192notna(train_valid["Bench1Kg"])] [bench1_features]
bench2input_train_valid = train_valid[pd.
    \u2192notna(train_valid["Bench2Kg"])] [bench2_features]
bench3input_train_valid = train_valid[pd.
    \u2192notna(train_valid["Bench3Kg"])] [bench3_features]
deadlift1input_train_valid = train_valid[pd.
    \u2192notna(train_valid["Deadlift1Kg"])] [deadlift1_features]
deadlift2input_train_valid = train_valid[pd.
    \u2192notna(train_valid["Deadlift2Kg"])] [deadlift2_features]
deadlift3input_train_valid = train_valid[pd.
    \u2192notna(train_valid["Deadlift3Kg"])] [deadlift3_features]

train_valid_inputs = [squat1input_train_valid.to_numpy(), \u2192
    \u2192squat2input_train_valid.to_numpy(), squat3input_train_valid.to_numpy(),
    \u2192bench1input_train_valid.to_numpy(), bench2input_train_valid.
    \u2192to_numpy(), bench3input_train_valid.to_numpy(),
    \u2192deadlift1input_train_valid.to_numpy(), deadlift2input_train_valid.
    \u2192to_numpy(), deadlift3input_train_valid.to_numpy()]

train_valid_outputs = [squat1pass_train_valid.to_numpy(), \u2192
    \u2192squat2pass_train_valid.to_numpy(), squat3pass_train_valid.to_numpy(),
    \u2192bench1pass_train_valid.to_numpy(), bench2pass_train_valid.to_numpy(), \u2192
    \u2192bench3pass_train_valid.to_numpy(),
    \u2192deadlift1pass_train_valid.to_numpy(), deadlift2pass_train_valid.
    \u2192to_numpy(), deadlift3pass_train_valid.to_numpy()]

for i in range(len(train_valid_inputs)):

```

```

    train_valid_inputs[i], train_valid_outputs[i] = 
    ↵RandomUnderSampler(random_state=42).fit_resample(train_valid_inputs[i], 
    ↵train_valid_outputs[i])
    train_valid_inputs[i], train_valid_outputs[i] = 
    ↵shuffle(train_valid_inputs[i], train_valid_outputs[i], random_state = 42)
    train_valid_inputs[i] = np.nan_to_num(train_valid_inputs[i])

```

[7]: squat1input_train_valid.to_numpy()

```

[7]: array([[1.        , 0.        , 0.1871345 , ..., 0.        , 0.        ,
   1.        ],
 [1.        , 0.        , 0.19883041, ..., 0.        , 0.        ,
   0.        ],
 [1.        , 0.        , 0.49122807, ..., 0.        , 0.        ,
   0.        ],
 ...,
 [1.        , 0.        , 0.12865497, ..., 0.        , 0.        ,
   0.        ],
 [0.        , 1.        , 0.16959064, ..., 0.        , 0.        ,
   0.        ],
 [0.        , 1.        , 0.26900585, ..., 0.        , 0.        ,
   0.        ]])

```

[8]: squat1pass_train_valid.to_numpy().reshape(-1,1)

```

[8]: array([[0],
 [0],
 [1],
 ...,
 [1],
 [1],
 [1]], dtype=int64)

```

```

[9]: squat1pass_test = test[pd.notna(test["Squat1Kg"])]["Squat1 Pass"]
squat2pass_test = test[pd.notna(test["Squat2Kg"])]["Squat2 Pass"]
squat3pass_test = test[pd.notna(test["Squat3Kg"])]["Squat3 Pass"]
bench1pass_test = test[pd.notna(test["Bench1Kg"])]["Bench1 Pass"]
bench2pass_test = test[pd.notna(test["Bench2Kg"])]["Bench3 Pass"]
bench3pass_test = test[pd.notna(test["Bench3Kg"])]["Bench3 Pass"]
deadlift1pass_test = test[pd.notna(test["Deadlift1Kg"])]["Deadlift1 Pass"]
deadlift2pass_test = test[pd.notna(test["Deadlift2Kg"])]["Deadlift2 Pass"]
deadlift3pass_test = test[pd.notna(test["Deadlift3Kg"])]["Deadlift3 Pass"]

squat1input_test = test[pd.notna(test["Squat1Kg"])][squat1_features]
squat2input_test = test[pd.notna(test["Squat2Kg"])][squat2_features]
squat3input_test = test[pd.notna(test["Squat3Kg"])][squat3_features]
bench1input_test = test[pd.notna(test["Bench1Kg"])][bench1_features]

```

```

bench2input_test = test[pd.notna(test["Bench2Kg"])][bench2_features]
bench3input_test = test[pd.notna(test["Bench3Kg"])][bench3_features]
deadlift1input_test = test[pd.notna(test["Deadlift1Kg"])][deadlift1_features]
deadlift2input_test = test[pd.notna(test["Deadlift2Kg"])][deadlift2_features]
deadlift3input_test = test[pd.notna(test["Deadlift3Kg"])][deadlift3_features]

test_inputs = [squat1input_test.to_numpy(), squat2input_test.to_numpy(), □
↳ squat3input_test.to_numpy(),
    bench1input_test.to_numpy(), bench2input_test.to_numpy(), □
↳ bench3input_test.to_numpy(),
    deadlift1input_test.to_numpy(), deadlift2input_test.to_numpy(), □
↳ deadlift3input_test.to_numpy()]

test_outputs = [squat1pass_test.to_numpy(), squat2pass_test.to_numpy(), □
↳ squat3pass_test.to_numpy(),
    bench1pass_test.to_numpy(), bench2pass_test.to_numpy(), □
↳ bench3pass_test.to_numpy(),
    deadlift1pass_test.to_numpy(), deadlift2pass_test.to_numpy(), □
↳ deadlift3pass_test.to_numpy()]

for i in range(len(test_inputs)):
    test_inputs[i] = np.nan_to_num(test_inputs[i])

```

```

[230]: import keras_tuner as kt
model_names = ["Squat1 Pass", "Squat2 Pass", "Squat3 Pass", "Bench1 Pass", □
↳ "Bench2 Pass", "Bench3 Pass", "Deadlift1 Pass", "Deadlift2 Pass", "Deadlift3 □
↳ Pass",]
feature_indices = [squat1_features, squat2_features, squat3_features, □
↳ bench1_features, bench2_features, bench3_features,
    deadlift1_features, deadlift2_features, deadlift3_features]

number = 0
for i,j in zip(train_valid_inputs, train_valid_outputs):

    callbacks = tf.keras.callbacks.EarlyStopping(
        monitor = "val_loss",
        min_delta=0.001,
        patience=5,
        verbose=0,
        mode="auto",
        baseline=None,
        restore_best_weights=True)

    def model_builder(hp):

```

```

global i
learning_rate_adam = hp.Float(f"learning rate value",
                                ←      5e-5,
                                ←      2e-3,
                                ←      sampling = "log",
                                ←      default = 0.01)
learning_rate_adadelta = hp.Float(f"learning rate adadelta",
                                    ←      0.8,
                                    ←      5.0,
                                    ←      sampling = "log",
                                    ←      )
optimizer = hp.Choice("optimizer", ["Adadelta", "Adam"])
feature_size = hp.Fixed("Feature_size", i.shape[1])
l1_reg = l1(hp.Float(f"activity reg ", 1e-6, 1e-3, sampling = "log",
                     ← default = 0.01))
hidden_size = hp.Int(f"intermediate_hidden_layer_sizes", 3,
                     ← feature_size*4, step = 10)
print(feature_size)
inputs = keras.Input(shape = (feature_size), name = "Input")
x = layers.Dense(hidden_size, activity_regularizer = l1_reg,
                  activation = "relu")(inputs)
x = layers.Dense(hidden_size, activity_regularizer = l1_reg,
                  activation = "relu")(x)
x = layers.Dense(hidden_size, activity_regularizer = l1_reg,
                  activation = "relu")(x)
x = layers.Dense(hidden_size, activity_regularizer = l1_reg,
                  activation = "relu")(x)
x = layers.Dense(hp.Int(f"final_hidden_layer_size", 3, feature_size*2,
                     ← step = 1), activity_regularizer = l1_reg,
                  activation = "relu", name = "final_hidden")(x)
output = layers.Dense(1, activation = "sigmoid", name = "Output")(x)

model = keras.Model(inputs, output)

if optimizer == "Adadelta":
    with hp.conditional_scope("optimizer", ["Adadelta"]):
        model.compile(loss = 'binary_crossentropy',

```

```

optimizer = keras.optimizers.Adadelta(learning_rate = learning_rate_adadelta,
→ rho = hp.Float(f" adadelta rho",
→         0.8,
→         1.0,
→         step = 0.05,
→         default = 0.01)),
→ metrics = ["binary_accuracy"])

elif optimizer == "Adam":
    with hp.conditional_scope("optimizer", ["Adam"]):
        model.compile(loss = 'binary_crossentropy',
                      optimizer = keras.optimizers.Adam(learning_rate = learning_rate_adam),
→ metrics = ["binary_accuracy"])

return model

tuner = kt.Hyperband(model_builder, objective = "val_binary_accuracy",
→ max_epochs = 40, hyperband_iterations= 5, factor = 4, seed = 42,
→ directory = model_names[number], project_name = "tests", overwrite=True)
print(tuner.search_space_summary())
print(model_names[number])
tuner.search(i,j.T, batch_size = 16, validation_split = 0.4,
→ shuffle = True, callbacks=[callbacks, keras.callbacks.TensorBoard(f"{model_names[number]} logs")]) #https://www.tensorflow.org/tutorials/keras/keras_tuner
with open(f"{model_names[number]} results.txt", 'w') as f:
    with redirect_stdout(f):
        tuner.results_summary()

final_model = tuner.hypermodel.build(tuner.get_best_hyperparameters(num_trials = 1)[0])
final_model.fit(i, j.T, epochs = 500, batch_size = 16, shuffle = True,
→ callbacks = [tf.keras.callbacks.EarlyStopping(
    monitor = "binary_accuracy",

```

```

    min_delta=0.001,
    patience=20,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=True)
])

y_scores = final_model.predict(test_inputs[number])
y_pred = (y_scores > 0.5).astype("int32")
y_true = test_outputs[number]

fpr, tpr, thresholds = roc_curve(y_true, y_scores)
area_under_curve = roc_auc_score(y_true, y_scores)
display = RocCurveDisplay(fpr = fpr, tpr = tpr, roc_auc=area_under_curve)
display.plot(name = f"{model_names[number]} Multi layer perceptron model")
plt.savefig(f"{model_names[number]} ROC curve.pdf")

with open(f"{model_names[number]} classification_report.txt", 'w') as f:
    with redirect_stdout(f):
        print(classification_report(y_true, y_pred))

permutation_form = KerasClassifier(build_fn = lambda: tuner.hypermodel.
build(tuner.get_best_hyperparameters(num_trials = 1)[0]))
permutation_form.fit(i, j.T, epochs = 500, batch_size = 16, shuffle = True, ▾
callbacks = [tf.keras.callbacks.EarlyStopping(
    monitor = "binary_accuracy",
    min_delta=0.001,
    patience=20,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=True)
])

r = permutation_importance(permutation_form, test_inputs[number], y_true,
                            n_repeats=5,
                            random_state=10, scoring = "balanced_accuracy")
feature_index = []
importance_means = []
importance_sds = []
for k in r.importances_mean.argsort()[:-1]:
    feature_index.append(feature_indices[number][k])
    importance_means.append(r.importances_mean[k])
    importance_sds.append(r.importances_std[k])

```

```

    feature_importances = pd.DataFrame(data = {"mean_importance":importance_means, "standard deviation": importance_sds}, index = feature_index )
    feature_importances.to_csv(f"{model_names[number]}_feature_importances.csv")

    number +=1

```

Trial 220 Complete [00h 01m 17s]
val_binary_accuracy: 0.613369345664978

Best val_binary_accuracy So Far: 0.627047598361969
Total elapsed time: 01h 54m 31s
INFO:tensorflow:Oracle triggered exit
85
Epoch 1/500
2833/2833 [=====] - 5s 2ms/step - loss: 1.7358 -
binary_accuracy: 0.5699
Epoch 2/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6758 -
binary_accuracy: 0.5969
Epoch 3/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6711 -
binary_accuracy: 0.6011
Epoch 4/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6715 -
binary_accuracy: 0.6017
Epoch 5/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6689 -
binary_accuracy: 0.6059
Epoch 6/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6684 -
binary_accuracy: 0.6031
Epoch 7/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6663 -
binary_accuracy: 0.6039
Epoch 8/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6666 -
binary_accuracy: 0.6064
Epoch 9/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6655 -
binary_accuracy: 0.6081
Epoch 10/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6652 -
binary_accuracy: 0.6044
Epoch 11/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6649 -
binary_accuracy: 0.6076
Epoch 12/500

```
2833/2833 [=====] - 5s 2ms/step - loss: 0.6657 -
binary_accuracy: 0.6083
Epoch 13/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6653 -
binary_accuracy: 0.6087
Epoch 14/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6648 -
binary_accuracy: 0.6087
Epoch 15/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6644 -
binary_accuracy: 0.6080
Epoch 16/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6634 -
binary_accuracy: 0.6097
Epoch 17/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6648 -
binary_accuracy: 0.6102
Epoch 18/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6625 -
binary_accuracy: 0.6123
Epoch 19/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6644 -
binary_accuracy: 0.6109
Epoch 20/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6649 -
binary_accuracy: 0.6113
Epoch 21/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6651 -
binary_accuracy: 0.6125
Epoch 22/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6654 -
binary_accuracy: 0.6114
Epoch 23/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6665 -
binary_accuracy: 0.6136
Epoch 24/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6654 -
binary_accuracy: 0.6122
Epoch 25/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6644 -
binary_accuracy: 0.6113
Epoch 26/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6632 -
binary_accuracy: 0.6134
Epoch 27/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6694 -
binary_accuracy: 0.6115
Epoch 28/500
```

```
2833/2833 [=====] - 5s 2ms/step - loss: 0.6651 -
binary_accuracy: 0.6134
Epoch 29/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6660 -
binary_accuracy: 0.6122
Epoch 30/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6682 -
binary_accuracy: 0.6119
Epoch 31/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6638 -
binary_accuracy: 0.6124
Epoch 32/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6631 -
binary_accuracy: 0.6131
Epoch 33/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6627 -
binary_accuracy: 0.6133
Epoch 34/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6654 -
binary_accuracy: 0.6118
Epoch 35/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6631 -
binary_accuracy: 0.6129
Epoch 36/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6643 -
binary_accuracy: 0.6124
Epoch 37/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6648 -
binary_accuracy: 0.6127
Epoch 38/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6642 -
binary_accuracy: 0.6152
Epoch 39/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6638 -
binary_accuracy: 0.6145
Epoch 40/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6617 -
binary_accuracy: 0.6127
Epoch 41/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6666 -
binary_accuracy: 0.6142
Epoch 42/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6628 -
binary_accuracy: 0.6144
Epoch 43/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6615 -
binary_accuracy: 0.6143
Epoch 44/500
```

```
2833/2833 [=====] - 5s 2ms/step - loss: 0.6655 -
binary_accuracy: 0.6142
Epoch 45/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6668 -
binary_accuracy: 0.6140
Epoch 46/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6663 -
binary_accuracy: 0.6137
Epoch 47/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6652 -
binary_accuracy: 0.6118
Epoch 48/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6663 -
binary_accuracy: 0.6138
Epoch 49/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6701 -
binary_accuracy: 0.6159
Epoch 50/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6658 -
binary_accuracy: 0.6139
Epoch 51/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6602 -
binary_accuracy: 0.6143
Epoch 52/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6636 -
binary_accuracy: 0.6143
Epoch 53/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6639 -
binary_accuracy: 0.6132
Epoch 54/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6626 -
binary_accuracy: 0.6152
Epoch 55/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6666 -
binary_accuracy: 0.6143
Epoch 56/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6664 -
binary_accuracy: 0.6150
Epoch 57/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6651 -
binary_accuracy: 0.6131
Epoch 58/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6641 -
binary_accuracy: 0.6143
85
Epoch 1/500
2833/2833 [=====] - 5s 2ms/step - loss: 1.9649 -
binary_accuracy: 0.5651
```

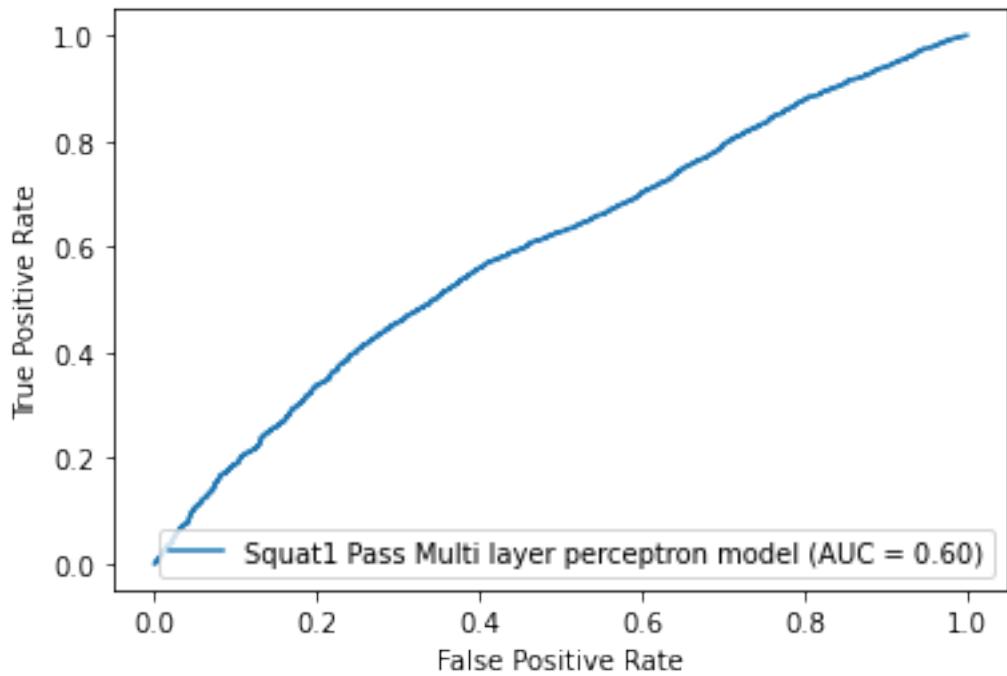
```
Epoch 2/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6750 -
binary_accuracy: 0.5930
Epoch 3/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6719 -
binary_accuracy: 0.6000
Epoch 4/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6710 -
binary_accuracy: 0.6021
Epoch 5/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6677 -
binary_accuracy: 0.6031
Epoch 6/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6675 -
binary_accuracy: 0.6044
Epoch 7/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6649 -
binary_accuracy: 0.6048
Epoch 8/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6665 -
binary_accuracy: 0.6059
Epoch 9/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6648 -
binary_accuracy: 0.6089
Epoch 10/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6655 -
binary_accuracy: 0.6076
Epoch 11/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6634 -
binary_accuracy: 0.6077
Epoch 12/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6645 -
binary_accuracy: 0.6100
Epoch 13/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6640 -
binary_accuracy: 0.6089
Epoch 14/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6644 -
binary_accuracy: 0.6097
Epoch 15/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6640 -
binary_accuracy: 0.6111
Epoch 16/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6628 -
binary_accuracy: 0.6093
Epoch 17/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6630 -
binary_accuracy: 0.6086
```

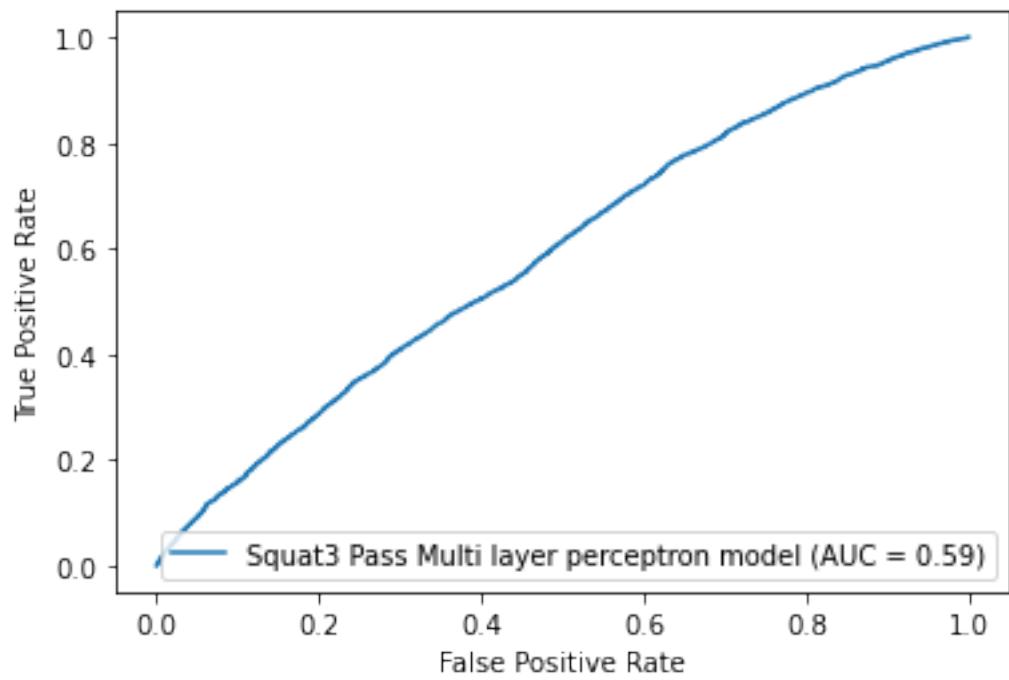
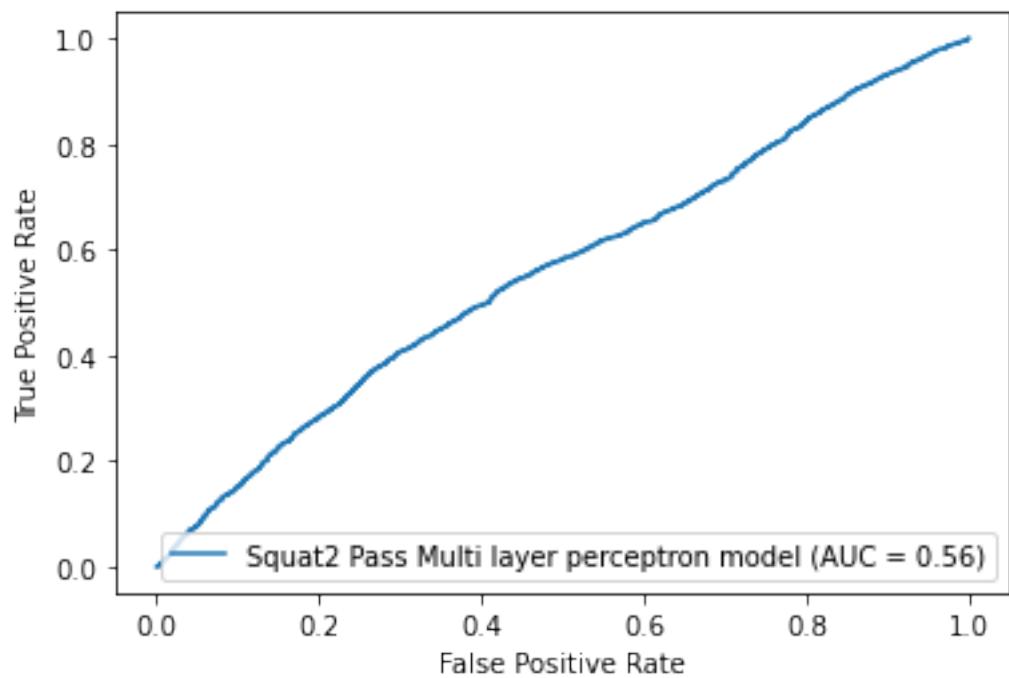
```
Epoch 18/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6620 -
binary_accuracy: 0.6099
Epoch 19/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6633 -
binary_accuracy: 0.6114
Epoch 20/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6640 -
binary_accuracy: 0.6113
Epoch 21/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6637 -
binary_accuracy: 0.6117
Epoch 22/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6642 -
binary_accuracy: 0.6113
Epoch 23/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6627 -
binary_accuracy: 0.6119
Epoch 24/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6619 -
binary_accuracy: 0.6131
Epoch 25/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6643 -
binary_accuracy: 0.6108
Epoch 26/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6633 -
binary_accuracy: 0.6132
Epoch 27/500
2833/2833 [=====] - 6s 2ms/step - loss: 0.6620 -
binary_accuracy: 0.6128
Epoch 28/500
2833/2833 [=====] - 6s 2ms/step - loss: 0.6642 -
binary_accuracy: 0.6124
Epoch 29/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6645 -
binary_accuracy: 0.6130
Epoch 30/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6627 -
binary_accuracy: 0.6124
Epoch 31/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6633 -
binary_accuracy: 0.6150
Epoch 32/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6633 -
binary_accuracy: 0.6145
Epoch 33/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6615 -
binary_accuracy: 0.6149
```

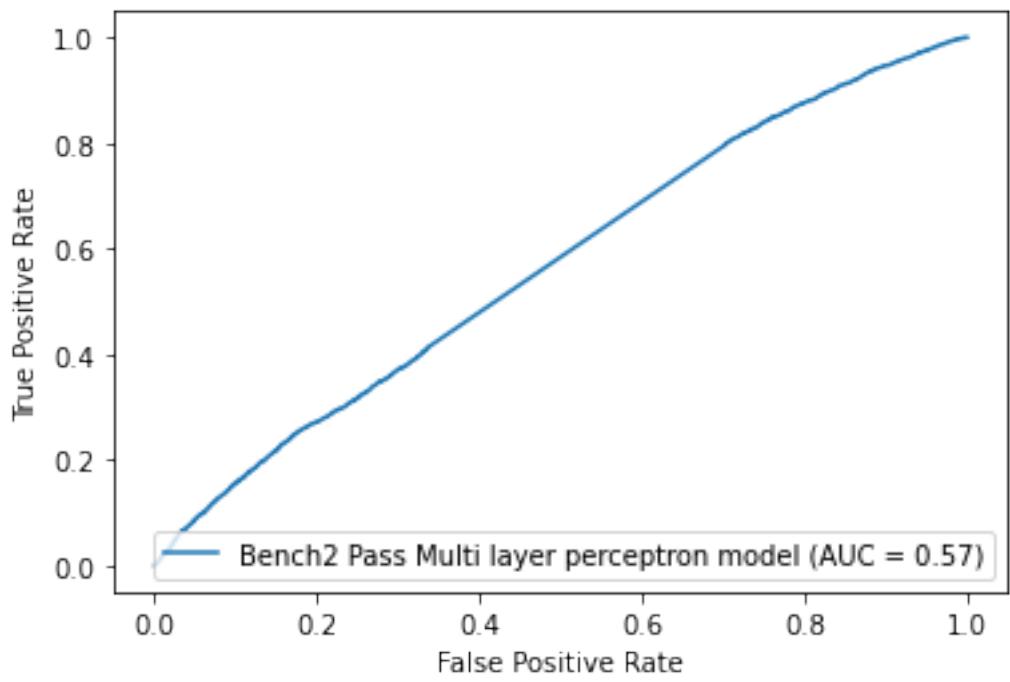
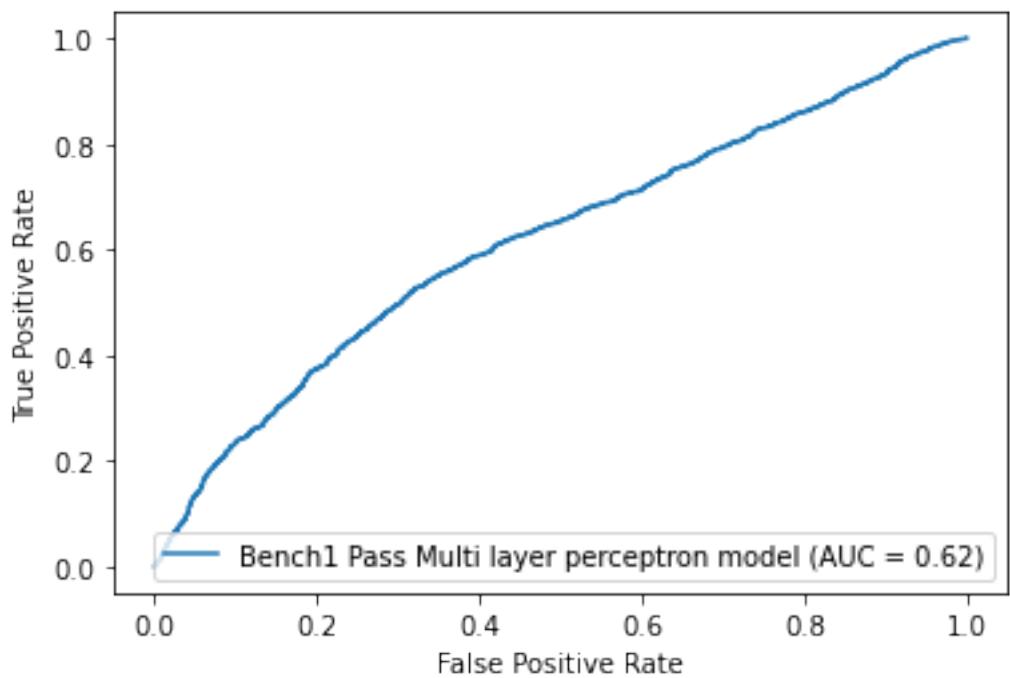
```
Epoch 34/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6625 -
binary_accuracy: 0.6136
Epoch 35/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6646 -
binary_accuracy: 0.6147
Epoch 36/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6630 -
binary_accuracy: 0.6135
Epoch 37/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6665 -
binary_accuracy: 0.6125
Epoch 38/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6628 -
binary_accuracy: 0.6144
Epoch 39/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6662 -
binary_accuracy: 0.6137
Epoch 40/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6649 -
binary_accuracy: 0.6143
Epoch 41/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6648 -
binary_accuracy: 0.6141
Epoch 42/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6660 -
binary_accuracy: 0.6131
Epoch 43/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6650 -
binary_accuracy: 0.6144
Epoch 44/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6658 -
binary_accuracy: 0.6145
Epoch 45/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6659 -
binary_accuracy: 0.6136
Epoch 46/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6641 -
binary_accuracy: 0.6142
Epoch 47/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6636 -
binary_accuracy: 0.6138
Epoch 48/500
2833/2833 [=====] - 6s 2ms/step - loss: 0.6652 -
binary_accuracy: 0.6155
Epoch 49/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6624 -
binary_accuracy: 0.6157
```

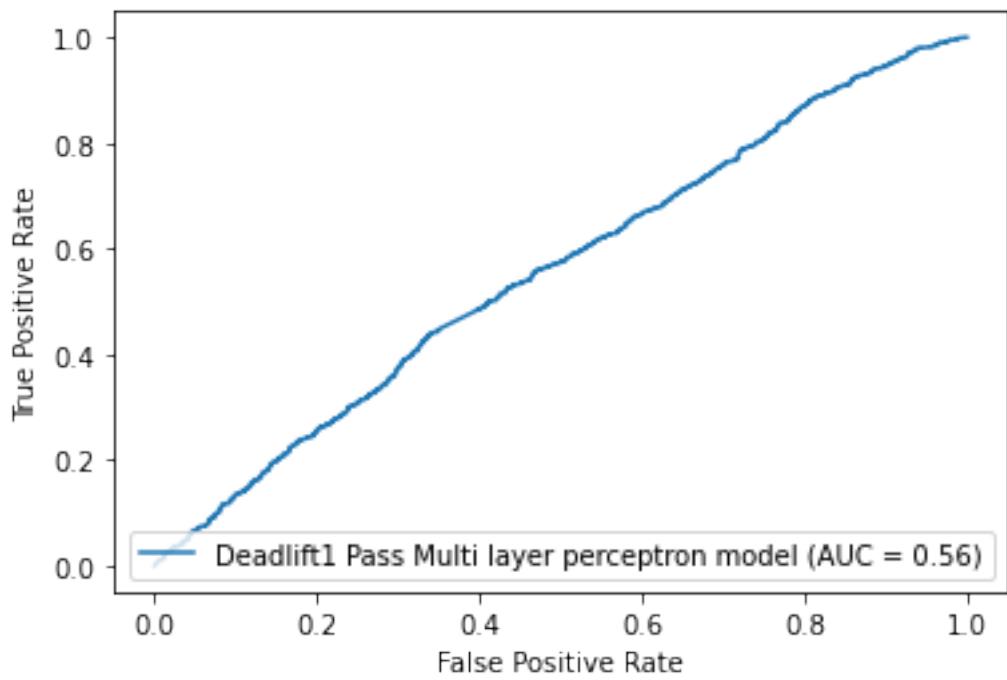
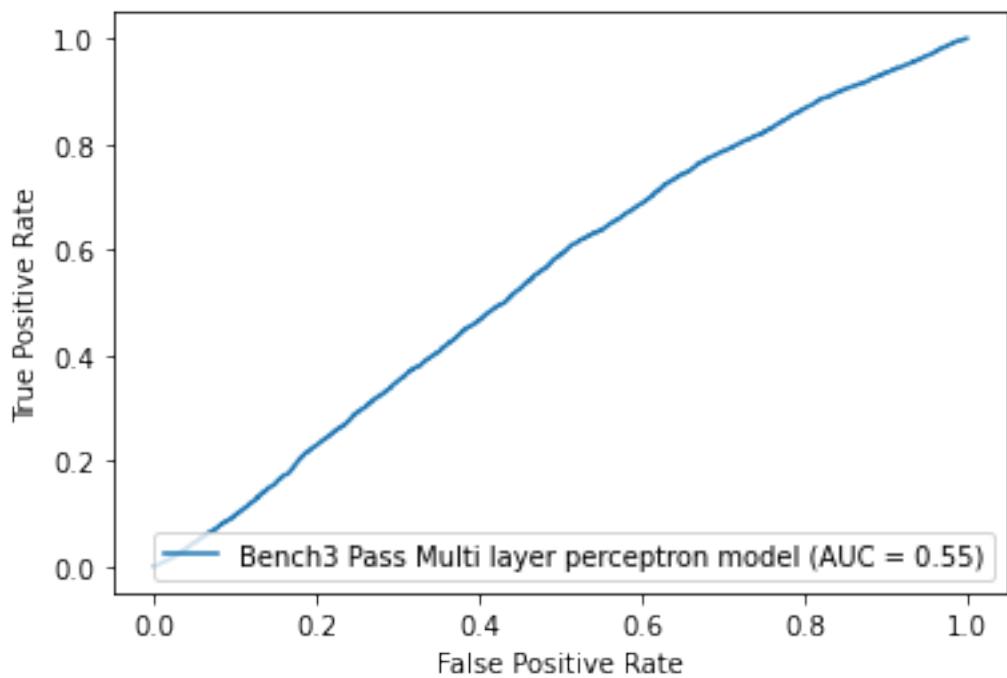
```
Epoch 50/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6624 -
binary_accuracy: 0.6168
Epoch 51/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6637 -
binary_accuracy: 0.6147
Epoch 52/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6652 -
binary_accuracy: 0.6155
Epoch 53/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6694 -
binary_accuracy: 0.6143
Epoch 54/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6634 -
binary_accuracy: 0.6150
Epoch 55/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6707 -
binary_accuracy: 0.6153
Epoch 56/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6652 -
binary_accuracy: 0.6144
Epoch 57/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6636 -
binary_accuracy: 0.6169
Epoch 58/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6648 -
binary_accuracy: 0.6156
Epoch 59/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6629 -
binary_accuracy: 0.6164
Epoch 60/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6656 -
binary_accuracy: 0.6154
Epoch 61/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6658 -
binary_accuracy: 0.6164
Epoch 62/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6685 -
binary_accuracy: 0.6167
Epoch 63/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6655 -
binary_accuracy: 0.6153
Epoch 64/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6655 -
binary_accuracy: 0.6162
Epoch 65/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6638 -
binary_accuracy: 0.6150
```

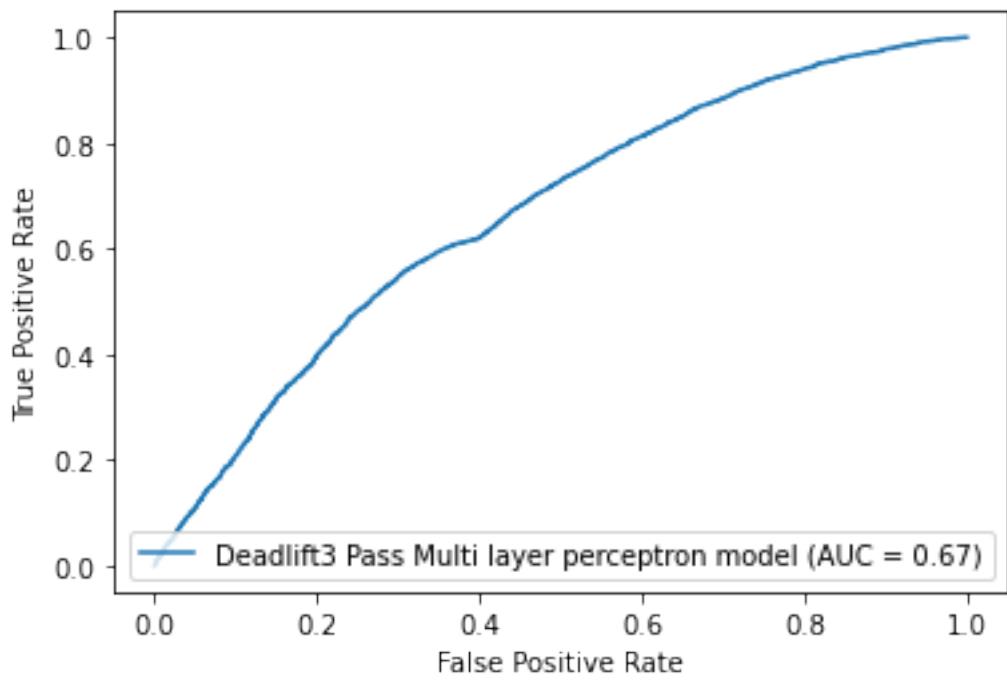
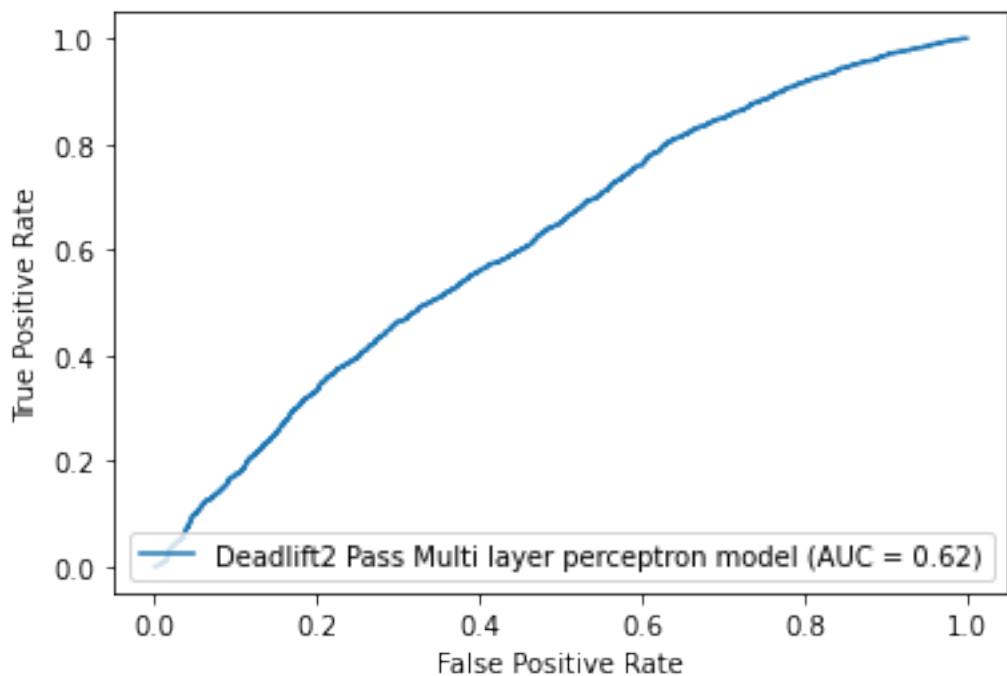
```
Epoch 66/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6641 -
binary_accuracy: 0.6176
Epoch 67/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6640 -
binary_accuracy: 0.6155
Epoch 68/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6683 -
binary_accuracy: 0.6149
Epoch 69/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6648 -
binary_accuracy: 0.6148
Epoch 70/500
2833/2833 [=====] - 5s 2ms/step - loss: 0.6633 -
binary_accuracy: 0.6162
```











```
[10]: for i in test_outputs:  
    print( i.mean())
```

```
0.8754251973557537
0.8439410515477186
0.6821967725143154
0.9084199717622898
0.4636749389224637
0.4696920372420079
0.9540465952121173
0.8913099702573387
0.6604654220352354
```

1 Scrap code

```
[ ]: # from keras.regularizers import l1_l2

# squat1_input = keras.Input(shape = (len(squat1_features),), name = "Initial\u202a
# Variables and Squat1",)
# squat1_hidden = layers.Dense(len(squat1_features)//2, activation = "relu",\u202a
# \u202a name = "Squat1_hidden",)(squat1_input)
# squat1_hidden_2 = layers.Dense(3, activation = "relu", name =\u202a
# \u202a "Squat1_hidden_2",)(squat1_hidden)
# squat1_output = layers.Dense(1, activation = "sigmoid", name =\u202a
# \u202a "Squat1_result")(squat1_hidden_2)

# squat2_input = keras.Input(shape = (len(squat2_features),), name = "Squat2\u202a
# \u202a Input")
# squat2_hidden = layers.Dense(squat2_input.shape[1]//2, activation = "relu",\u202a
# \u202a name = "Squat2_hidden",)(squat2_input)
# squat2_hidden_2 = layers.Dense(3, activation = "relu", name =\u202a
# \u202a "Squat2_hidden_2",)(squat2_hidden)
# squat2_output = layers.Dense(1, activation = "sigmoid", name =\u202a
# \u202a "Squat2_result")(squat2_hidden_2)

# squat3_input = keras.Input(shape = (len(squat3_features),), name = "Squat3\u202a
# \u202a Input")
# squat3_hidden = layers.Dense(squat3_input.shape[1]//2, activation = "relu",\u202a
# \u202a name = "Squat3_hidden",)(squat3_input)
# squat3_hidden_2 = layers.Dense(3, activation = "relu", name =\u202a
# \u202a "Squat3_hidden_2",)(squat3_hidden)
# squat3_output = layers.Dense(1, activation = "sigmoid", name =\u202a
# \u202a "Squat3_result")(squat3_hidden_2)

# bench1_input = keras.Input(shape = (len(bench1_features),), name = "Bench1\u202a
# \u202a Input")
# bench1_hidden = layers.Dense(bench1_input.shape[1]//2, activation = "relu",\u202a
# \u202a name = "Bench1_hidden",)(bench1_input)
```

```

# bench1_hidden_2 = layers.Dense(3, activation = "relu", name = "bench1_hidden_2")
# bench1_output = layers.Dense(1, activation = "sigmoid", name = "Bench1_result")(bench1_hidden_2)

# bench2_input = keras.Input(shape = (len(bench2_features),), name = "Bench2_Input")
# bench2_hidden = layers.Dense(bench2_input.shape[1]//2, activation = "relu", name = "Bench2_hidden")(bench2_input)
# bench2_hidden_2 = layers.Dense(3, activation = "relu", name = "bench2_hidden_2")
# bench2_output = layers.Dense(1, activation = "sigmoid", name = "Bench2_result")(bench2_hidden_2)

# bench3_input = keras.Input(shape = (len(bench3_features),), name = "Bench3_Input")
# bench3_hidden = layers.Dense(bench3_input.shape[1]//2, activation = "relu", name = "Bench3_hidden")(bench3_input)
# bench3_hidden_2 = layers.Dense(3, activation = "relu", name = "bench3_hidden_2")
# bench3_output = layers.Dense(1, activation = "sigmoid", name = "Bench3_result")(bench3_hidden_2)

# deadlift1_input = keras.Input(shape = (len(deadlift1_features),), name = "Deadlift1_Input")
# deadlift1_hidden = layers.Dense(deadlift1_input.shape[1]//2, activation = "relu", name = "Deadlift1_hidden")(deadlift1_input)
# deadlift1_hidden_2 = layers.Dense(3, activation = "relu", name = "deadlift1_hidden_2")
# deadlift1_output = layers.Dense(1, activation = "sigmoid", name = "Deadlift1_result")(deadlift1_hidden_2)

# deadlift2_input = keras.Input(shape = (len(deadlift2_features),), name = "Deadlift2_Input")
# deadlift2_hidden = layers.Dense(deadlift2_input.shape[1]//2, activation = "relu", name = "Deadlift2_hidden")(deadlift2_input)
# deadlift2_hidden_2 = layers.Dense(3, activation = "relu", name = "deadlift2_hidden_2")
# deadlift2_output = layers.Dense(1, activation = "sigmoid", name = "Deadlift2_result")(deadlift2_hidden_2)

# deadlift3_input = keras.Input(shape = (len(deadlift3_features),), name = "Deadlift3_Input")
# deadlift3_hidden = layers.Dense(deadlift3_input.shape[1]//2, activation = "relu", name = "Deadlift3_hidden")(deadlift3_input)

```

```

# deadlift3_hidden_2 = layers.Dense(3, activation = "relu", name =_
#   ↪"deadlift3_hidden_2")(deadlift3_hidden)
# deadlift3_output = layers.Dense(1, activation = "sigmoid", name =_
#   ↪"Deadlift3_result")(deadlift3_hidden_2)

# model = keras.Model(
#     inputs=[squat1_input, squat2_input, squat3_input,
#             bench1_input, bench2_input, bench3_input,
#             deadlift1_input, deadlift2_input, deadlift3_input,],
#     outputs=[squat1_output, squat2_output, squat3_output,
#             bench1_output, bench2_output, bench3_output,
#             deadlift1_output, deadlift2_output, deadlift3_output,],
# )
# model_inputs = dict(zip(model.input_names, train_valid_inputs))
# model_outputs = dict(zip(model.output_names, train_valid_outputs))

```

```

[ ]: # print("squat1")

# model_squat1 = keras.Model(
# inputs = squat1_input, outputs = squat1_output)

# model_squat1_inputs = dict(zip(model_squat1.input_names,_
#   ↪[train_valid_inputs[0]]))
# model_squat1_outputs = dict(zip(model_squat1.output_names,_
#   ↪[train_valid_outputs[0]]))

# model_squat1.compile(optimizer = 'adam', loss = "binary_crossentropy",_
#   ↪metrics=[tf.keras.metrics.BinaryAccuracy(),
#             keras.metrics.FalsePositives(),])

# model_squat1.fit(model_squat1_inputs,
#                   model_squat1_outputs, verbose = 1, epochs = 100, validation_split=0.
#   ↪2, batch_size = 5000)

# print("squat2")

# model_squat2 = keras.Model(
#     inputs=[squat2_input],
#     outputs=[squat2_output],
# )
# model_squat2_inputs = dict(zip(model_squat2.input_names,_
#   ↪[train_valid_inputs[1]]))
# model_squat2_outputs = dict(zip(model_squat2.output_names,_
#   ↪[train_valid_outputs[1]]))

```

```

# model_squat2.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.001),
#     loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
#     keras.metrics.FalsePositives(),])

# model_squat2.fit(model_squat2_inputs,
#     model_squat2_outputs, verbose = 1, epochs = 100, validation_split=0.
#     2, batch_size = 5000)

# print("squat3")

# model_squat3 = keras.Model(
#     inputs=[squat3_input],
#     outputs=[squat3_output],
# )

# model_squat3_inputs = dict(zip(model_squat3.input_names, [
#     train_valid_inputs[2]]))
# model_squat3_outputs = dict(zip(model_squat3.output_names, [
#     train_valid_outputs[2]]))
# model_squat3.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.001),
#     loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
#     keras.metrics.FalsePositives(),])

# model_squat3.fit(model_squat3_inputs,
#     model_squat3_outputs, verbose = 1, epochs = 100, validation_split=0.
#     2, batch_size = 5000)

# print("bench1")

# model_bench1 = keras.Model(
#     inputs=[ bench1_input],
#     outputs=[
#         bench1_output],
# )

# model_bench1_inputs = dict(zip(model_bench1.input_names, [
#     train_valid_inputs[3]]))
# model_bench1_outputs = dict(zip(model_bench1.output_names, [
#     train_valid_outputs[3]]))
# model_bench1.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.001),
#     loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
#     keras.metrics.FalsePositives(),])

# model_bench1.fit(model_bench1_inputs,
#     model_bench1_outputs, verbose = 1, epochs = 100, validation_split=0.
#     2, batch_size = 5000)

```

```

# print("bench2")

# model_bench2 = keras.Model(
#     inputs=[bench2_input],
#     outputs=[bench2_output],
# )
# model_bench2_inputs = dict(zip(model_bench2.input_names, [
#     train_valid_inputs[4]
# ]))
# model_bench2_outputs = dict(zip(model_bench2.output_names, [
#     train_valid_outputs[4]
# ]))
# model_bench2.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.001),
#     loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
#     keras.metrics.FalsePositives()])
# model_bench2.fit(model_bench2_inputs,
#     model_bench2_outputs, verbose = 1, epochs = 100, validation_split=0.
#     2, batch_size = 5000)

# print("bench3")

# model_bench3 = keras.Model(
#     inputs=[bench3_input],
#     outputs=[bench3_output],
# )
# model_bench3_inputs = dict(zip(model_bench3.input_names, [
#     train_valid_inputs[5]
# ]))
# model_bench3_outputs = dict(zip(model_bench3.output_names, [
#     train_valid_outputs[5]
# ]))
# model_bench3.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.001),
#     loss = "binary_crossentropy", metrics=[tf.keras.metrics.BinaryAccuracy(),
#     keras.metrics.FalsePositives()])
# model_bench3.fit(model_bench3_inputs,
#     model_bench3_outputs, verbose = 1, epochs = 100, validation_split=0.
#     2, batch_size = 5000)

# print("deadlift1")

# model_deadlift1 = keras.Model(
#     inputs=[deadlift1_input],
#     outputs=[
#         deadlift1_output],
# )

```

```

# model_deadlift1_inputs = dict(zip(model_deadlift1.input_names, □
#                                [train_valid_inputs[6]]))
# model_deadlift1_outputs = dict(zip(model_deadlift1.output_names, □
#                                    [train_valid_outputs[6]]))
# model_deadlift1.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.
#                                                        →001), loss = "binary_crossentropy", metrics=[tf.keras.metrics.
#                                                        →BinaryAccuracy(),
#                                                        keras.metrics.FalsePositives(),])
# print("deadlift1")

# model_deadlift2 = keras.Model(
#     inputs=[deadlift2_input],
#     outputs=[deadlift2_output],
# )
# model_deadlift2_inputs = dict(zip(model_deadlift2.input_names, □
#                                   [train_valid_inputs[7]]))
# model_deadlift2_outputs = dict(zip(model_deadlift2.output_names, □
#                                    [train_valid_outputs[7]]))
# model_deadlift2.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.
#                                                        →001), loss = "binary_crossentropy", metrics=[tf.keras.metrics.
#                                                        →BinaryAccuracy(),
#                                                        keras.metrics.FalsePositives(),])
# print("deadlift2")

# model_deadlift2.fit(model_deadlift2_inputs,
#                      model_deadlift2_outputs, verbose = 1, epochs = 100, □
#                      validation_split=0.2, batch_size = 5000)

# print("deadlift3")

# model_deadlift3 = keras.Model(
#     inputs=[deadlift3_input],
#     outputs=[deadlift3_output],
# )
# model_deadlift3_inputs = dict(zip(model_deadlift3.input_names, □
#                                   [train_valid_inputs[8]]))
# model_deadlift3_outputs = dict(zip(model_deadlift3.output_names, □
#                                    [train_valid_outputs[8]]))
# model_deadlift3.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.
#                                                        →001), loss = "binary_crossentropy", metrics=[tf.keras.metrics.
#                                                        →BinaryAccuracy(),

```

```
# keras.metrics.FalsePositives(),])
# model_deadlift3.fit(model_deadlift3_inputs,
#                      model_deadlift3_outputs, verbose = 1, epochs = 100, validation_split=0.2, batch_size = 5000)

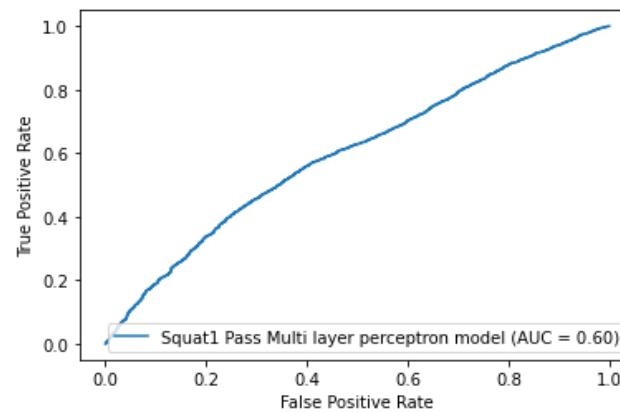
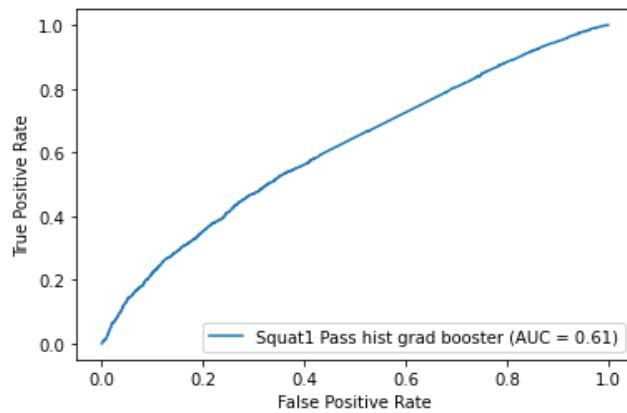
[ ]: #keras.utils.plot_model(model, "multi_input_and_output_model.png", rankdir = "LR", show_shapes=True)

[ ]: # model_inputs = dict(zip(model.input_names, train_valid_inputs))
# model_outputs = dict(zip(model.output_names, train_valid_outputs))
```

Appendix I Model Results

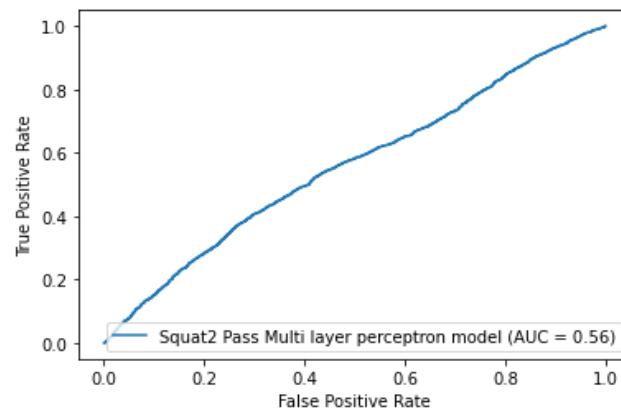
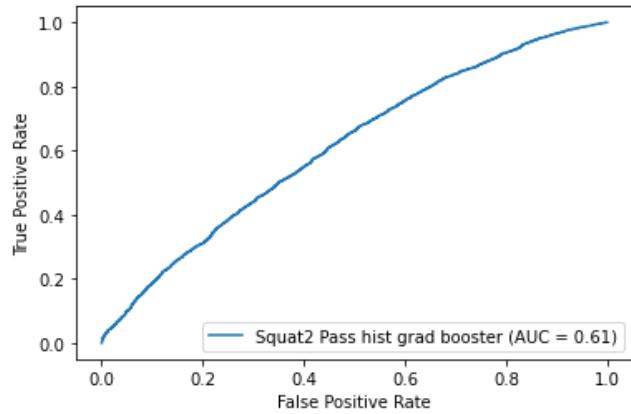
Squat1 Pass (Pass rate = 0.875)						
	Histogram based gradient booster			Multi-layer perceptron		
	Top 5 Features	Mean	Standard deviation	Top 5 Features	Mean	Standard deviation
1	Overall Attempt Success Rate over past three meets	0.0156	0.0047	Time Since Last Meet	0.0282	0.0045
2	Number of Past Meets	0.0152	0.0050	Number of Past Meets	0.0164	0.0034
3	Number of Past Meets Same Day Exclusive	0.0118	0.0030	Age	0.0096	0.0013
4	Age	0.0099	0.0024	WeightClassKg_84+	0.0033	0.0018
5	WeightClassKg	0.0061	0.0018	Squat2 Past 3 meets success rate	0.0030	0.0006

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.16	0.64	0.26	1941	0	0.16	0.66	0.25	1941
1	0.91	0.53	0.67	13640	1	0.91	0.50	0.64	13640
accuracy				0.55	accuracy				0.52
macro avg	0.54	0.59	0.47	15581	macro avg	0.53	0.58	0.45	15581
weighted avg	0.82	0.55	0.62	15581	weighted avg	0.82	0.52	0.59	15581



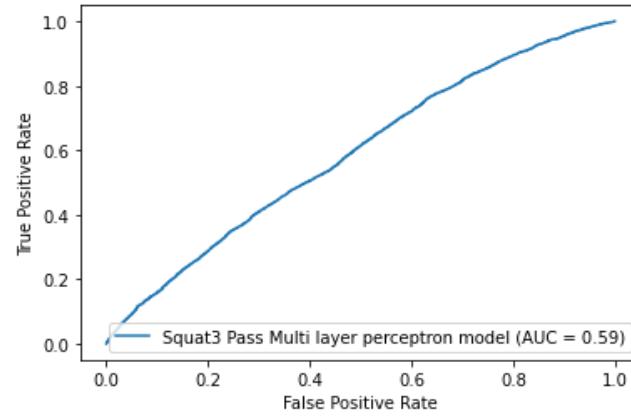
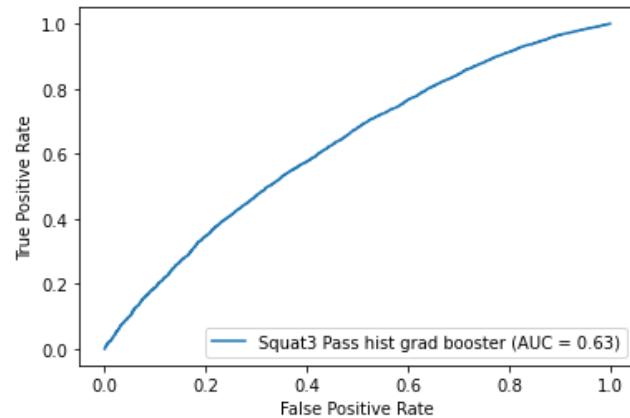
Squat2 Pass (Pass rate = 0.843)						
	Histogram based gradient booster			Multi-layer perceptron		
	Top 5 Features	Mean	Standard deviation	Top 5 Features	Mean	Standard deviation
1	WeightClassKg	0.0198	0.0046	Time Since Last Meet	0.0162	0.0084
2	Squat2 Jump	0.0154	0.0016	Squat1 Pass	0.0119	0.0014
3	Overall Attempt Success Rate over past three meets	0.0121	0.0023	Female	0.0046	0.0023
4	Squat1 Pass	0.0115	0.0018	Age	0.0021	0.0009
5	Squat2Kg	0.0087	0.0025	WeightClassKg_93	0.0017	0.0008

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.21	0.52	0.30	2425	0	0.18	0.61	0.28	2425
1	0.88	0.64	0.74	13114	1	0.87	0.49	0.63	13114
accuracy				0.62	accuracy				0.51
macro avg	0.54	0.58	0.52	15539	macro avg	0.53	0.55	0.45	15539
weighted avg	0.77	0.62	0.67	15539	weighted avg	0.76	0.51	0.57	15539



Squat3 Pass (Pass rate = 0.682)						
	Histogram based gradient booster			Multi-layer perceptron		
		Mean	Standard deviation	Top 5 Features	Mean	Standard deviation
1	WeightClassKg	0.0374	0.0039	Time Since Last Meet	0.0151	0.0046
2	Squat3Kg	0.0242	0.0029	Squat2 Pass	0.0148	0.0014
3	Squat3 Jump	0.0218	0.0025	Female	0.0094	0.0014
4	Squat2Kg	0.0114	0.0016	Squat3 Past 3 meets success rate	0.0038	0.0011
5	Overall Attempt Success Rate over past three meets	0.0109	0.0025	WeightClassKg_63	0.0037	0.0011

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.42	0.49	0.46	4884	0	0.38	0.47	0.42	4884
1	0.74	0.68	0.71	10484	1	0.72	0.65	0.68	10484
accuracy			0.62	15368	accuracy			0.59	15368
macro avg	0.58	0.59	0.58	15368	macro avg	0.55	0.56	0.55	15368
weighted avg	0.64	0.62	0.63	15368	weighted avg	0.62	0.59	0.60	15368



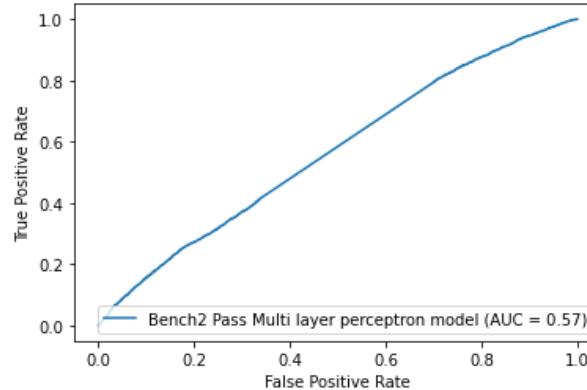
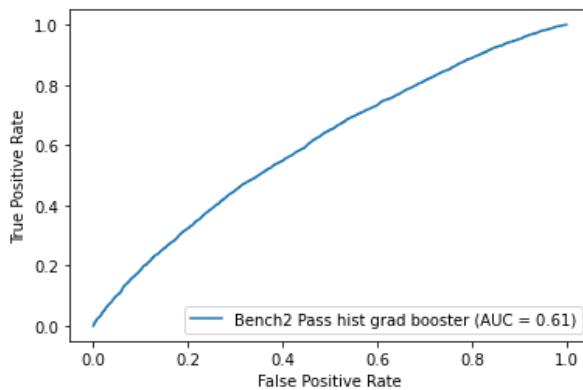
Bench1 Pass (Pass rate = 0.908)							
	Histogram based gradient booster			Multi-layer perceptron			
	Top 5 Features	Mean	Standard deviation	Top 5 Features	Mean	Standard deviation	
1	Bench1Kg	0.0201	0.0047	Time Since Last Meet	0.0373	0.0045	
2	Number of Past Meets	0.0180	0.0046	Number of Past Meets	0.0170	0.0059	
3	Squat2 Pass	0.0122	0.0032	Squat2 Pass	0.0072	0.0023	
4	Squat3 Pass	0.0103	0.0019	Squat1 Pass	0.0062	0.0012	
5	Squat2 Jump	0.0083	0.0012	Squat3 Pass	0.0059	0.0010	

Bench2 Pass (Pass rate = 0.463)						
	Histogram based gradient booster			Multi-layer perceptron		
	Top 5 Features	Mean	Standard deviation	Top 5 Features	Mean	Standard deviation
1	Bench2Kg	0.0298	0.0014	Squat3 Pass	0.0144	0.0015
2	Bench2 Jump	0.0160	0.0012	Number of Past Meets	0.0133	0.0021
3	Squat3 Pass	0.0158	0.0034	Time Since Last Meet	0.0105	0.0021
4	Best Raw Wilks to date	0.0101	0.0022	Bench3 Past 3 meets success rate	0.0061	0.0013
5	WeightClassKg	0.0094	0.0024	Squat2 Pass	0.0055	0.0015

	precision	recall	f1-score	support		precision	recall	f1-score	support
--	-----------	--------	----------	---------	--	-----------	--------	----------	---------

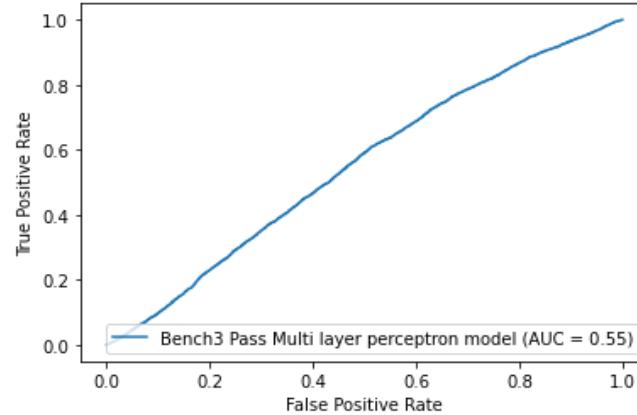
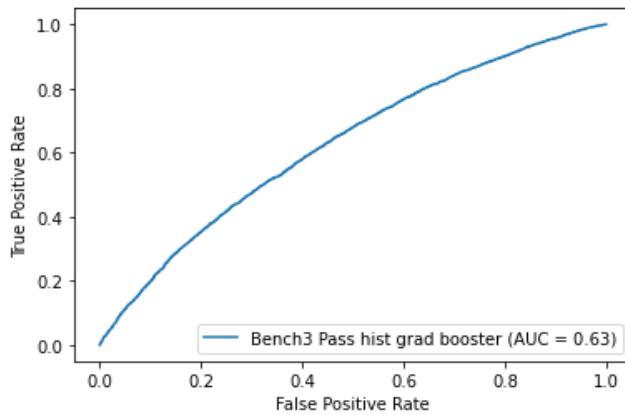
0	0.63	0.48	0.55	8342	0	0.56	0.68	0.62	8342
1	0.53	0.67	0.59	7212	1	0.51	0.39	0.44	7212

accuracy			0.57	15554	accuracy			0.55	15554
macro avg	0.58	0.58	0.57	15554	macro avg	0.54	0.54	0.53	15554
weighted avg	0.58	0.57	0.57	15554	weighted avg	0.54	0.55	0.54	15554



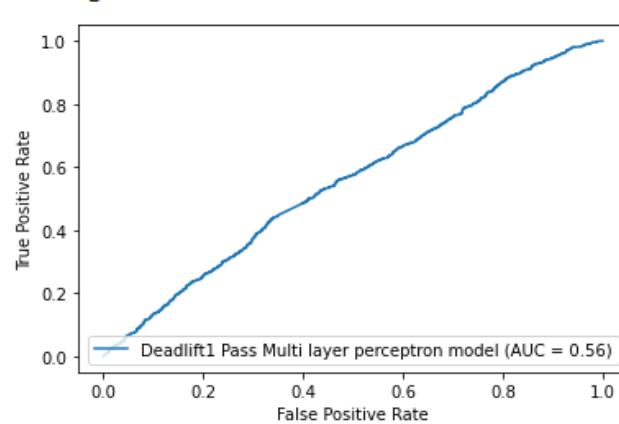
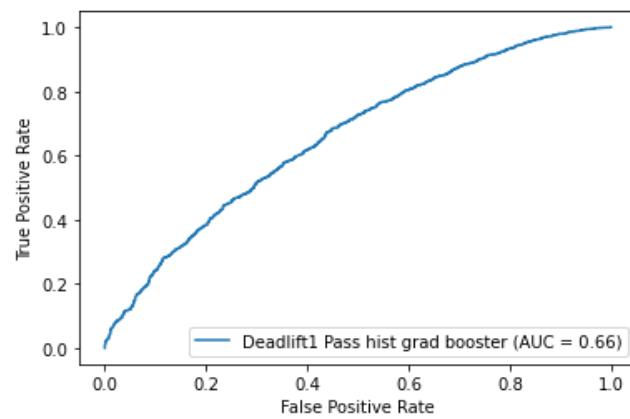
Bench3 Pass (pass rate = 0.469)						
	Histogram based gradient booster			Multi-layer perceptron		
	Top 5 Features	Mean	Standard deviation	Top 5 Features	Mean	Standard deviation
1	Bench3 Jump	0.0388	0.0025	Bench2 Pass	0.0108	0.0013
2	Bench3Kg	0.0302	0.0017	Bench3 Past 3 meets success rate	0.0043	0.0021
3	Squat3 Pass	0.0112	0.0016	Female	0.0041	0.0018
4	WeightClassKg	0.0084	0.0011	Squat3 Pass	0.0036	0.0023
5	Total Bench Jump	0.0079	0.0015	Squat2 Pass	0.0031	0.0013

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.64	0.48	0.55	8145	0	0.56	0.59	0.58	8145
1	0.54	0.69	0.61	7214	1	0.51	0.48	0.49	7214
accuracy			0.58	15359	accuracy			0.54	15359
macro avg	0.59	0.59	0.58	15359	macro avg	0.53	0.53	0.53	15359
weighted avg	0.60	0.58	0.58	15359	weighted avg	0.54	0.54	0.54	15359



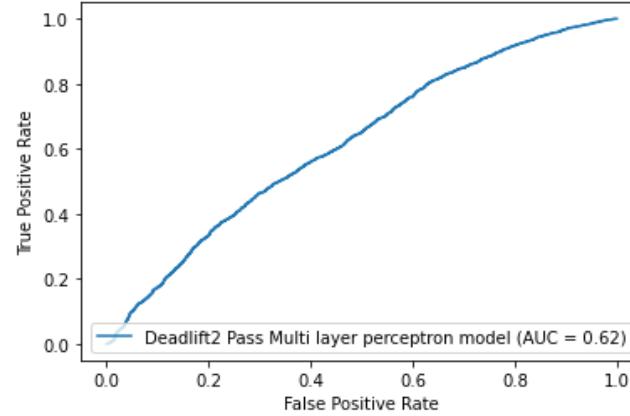
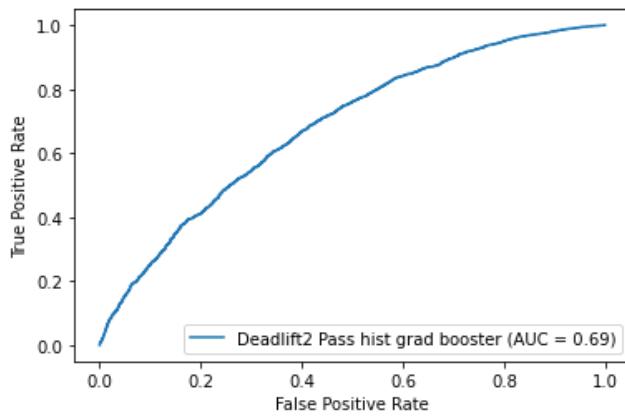
Deadlift1 Pass (Pass rate = 0.954)								
	Histogram based gradient booster			Multi-layer perceptron				
	Top 5 Features		Mean	deviation	Top 5 Features		Mean	Standard deviation
1	Deadlift1Kg		0.0167	0.0062	Time Since Last Meet		0.0125	0.0099
2	Total Squat Jump		0.0077	0.0033	Bench2 Pass		0.0101	0.0019
3	Squat3 Pass		0.0053	0.0026	Squat2 Pass		0.0077	0.0034
4	Age		0.0049	0.0033	Squat3 Pass		0.0072	0.0034
5	Squat2 Pass		0.0016	0.0030	Best Raw Bench to date		0.0037	0.0021

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.07	0.57	0.13	716	0	0.05	0.70	0.09	716
1	0.97	0.65	0.78	14865	1	0.96	0.36	0.53	14865
accuracy			0.64	15581	accuracy			0.38	15581
macro avg	0.52	0.61	0.45	15581	macro avg	0.51	0.53	0.31	15581
weighted avg	0.93	0.64	0.75	15581	weighted avg	0.92	0.38	0.51	15581



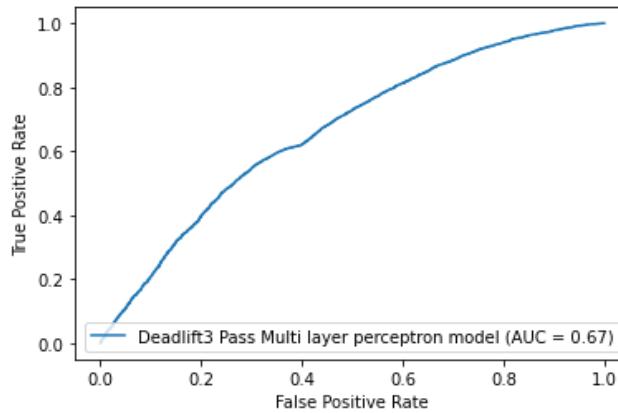
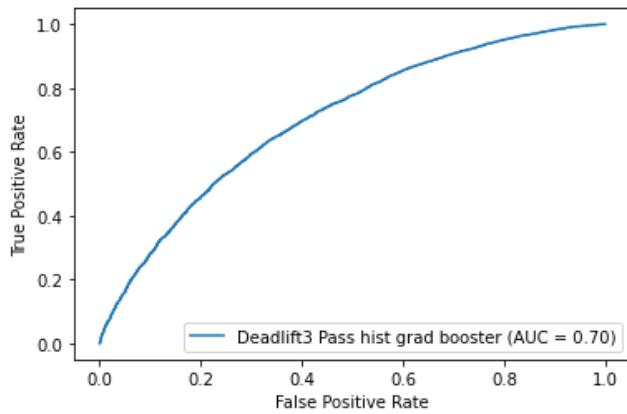
Deadlift2 Pass (Pass rate = 0.891)						
	Histogram based gradient booster			Multi-layer perceptron		
	Top 5 Features	Mean	Standard deviation	Top 5 Features	Mean	Standard deviation
1	Deadlift2Kg	0.0412	0.0039	Time Since Last Meet	0.0124	0.0035
2	Bench3 Pass	0.0114	0.0015	Bench3 Pass	0.0101	0.0021
3	Deadlift2 Jump	0.0112	0.0030	Squat3 Pass	0.0096	0.0022
4	Squat3 Pass	0.0085	0.0013	Bench2 Pass	0.0044	0.0021
5	Squat3 Jump	0.0067	0.0018	Deadlift3 Past 3 meets success rate	0.0039	0.0018

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.19	0.53	0.28	1681	0	0.16	0.44	0.23	1681
1	0.93	0.73	0.82	13785	1	0.91	0.71	0.80	13785
accuracy			0.71	15466	accuracy			0.68	15466
macro avg	0.56	0.63	0.55	15466	macro avg	0.54	0.58	0.52	15466
weighted avg	0.85	0.71	0.76	15466	weighted avg	0.83	0.68	0.74	15466



Deadlift3 Pass (Pass rate = 0.660)						
	Histogram based gradient booster			Multi-layer perceptron		
	Top 5 Features	Mean	Standard deviation	Top 5 Features	Mean	Standard deviation
1	Deadlift3Kg	0.0389	0.0022	Time Since Last Meet	0.0308	0.0047
2	Deadlift3 Jump	0.0278	0.0021	Deadlift2 Pass	0.0153	0.0012
3	Deadlift2Kg	0.0198	0.0024	Female	0.0095	0.0019
4	Squat3 Pass	0.0077	0.0021	Bench3 Pass	0.0081	0.0012
5	WeightClassKg	0.0052	0.0019	Deadlift2Kg	0.0078	0.0016

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.51	0.58	0.54	5165	0	0.51	0.43	0.47	5165
1	0.77	0.71	0.74	10047	1	0.73	0.79	0.76	10047
accuracy			0.67	15212	accuracy			0.67	15212
macro avg	0.64	0.65	0.64	15212	macro avg	0.62	0.61	0.61	15212
weighted avg	0.68	0.67	0.67	15212	weighted avg	0.66	0.67	0.66	15212



Squat1 Pass (Pass rate = 0.875)									
		Histogram based gradient booster				Multi-layer perceptron			
		Precision	Recall	f1 score	support	Precision	Recall	f1 score	support
Label	0	0.16	0.64	0.26	1941	0.16	0.66	0.25	1941
	1	0.82	0.53	0.67	13640	0.91	0.5	0.64	13640
AUROC		0.61				0.6			

Squat2 Pass (Pass rate = 0.843)									
		Histogram based gradient booster				Multi-layer perceptron			
		Precision	Recall	f1 score	support	Precision	Recall	f1 score	support
Label	0	0.21	0.52	0.30	2425	0.18	0.61	0.28	2425
	1	0.88	0.64	0.74	13314	0.87	0.49	0.63	13314
AUROC		0.61				0.56			

Squat3 Pass (Pass rate = 0.682)									
		Histogram based gradient booster				Multi-layer perceptron			
		Precision	Recall	f1 score	support	Precision	Recall	f1 score	support
Label	0	0.42	0.49	0.46	4484	0.38	0.47	0.42	4484
	1	0.74	0.68	0.71	10484	0.72	0.65	0.68	10484
AUROC		0.63				0.59			

Bench1 Pass (Pass rate = 0.908)									
		Histogram based gradient booster				Multi-layer perceptron			
		Precision	Recall	f1 score	support	Precision	Recall	f1 score	support
Label	0	0.14	0.62	0.22	1427	0.12	0.75	0.20	1427
	1	0.94	0.60	0.74	14155	0.95	0.44	0.60	14155
AUROC		0.66				0.62			

Bench2 Pass (Pass rate = 0.463)									
		Histogram based gradient booster				Multi-layer perceptron			
		Precision	Recall	f1 score	support	Precision	Recall	f1 score	support
Label	0	0.63	0.48	0.55	8342	0.56	0.68	0.62	8342
	1	0.53	0.67	0.59	7212	0.51	0.39	0.44	7212
AUROC		0.61				0.57			

Bench3 Pass (pass rate = 0.469)									
		Histogram based gradient booster				Multi-layer perceptron			
		Precision	Recall	f1 score	support	Precision	Recall	f1 score	support
Label	0	0.64	0.48	0.55	8145	0.56	0.59	0.58	8145
	1	0.54	0.69	0.51	7214	0.51	0.48	0.49	7214
AUROC		0.63				0.55			

Deadlift1 Pass (Pass rate = 0.954)									
		Histogram based gradient booster				Multi-layer perceptron			
		Precision	Recall	f1 score	support	Precision	Recall	f1 score	support
Label	0	0.07	0.57	0.13	716	0.05	0.70	0.09	716
	1	0.97	0.65	0.78	14865	0.96	0.36	0.53	14865
AUROC		0.66				0.56			

Deadlift2 Pass (Pass rate = 0.891)									
		Histogram based gradient booster				Multi-layer perceptron			
		Precision	Recall	f1 score	support	Precision	Recall	f1 score	support
Label	0	0.19	0.53	0.28	1681	0.16	0.44	0.23	1681
	1	0.93	0.73	0.82	13785	0.91	0.71	0.8	13785
AUROC		0.69				0.62			

Deadlift3 Pass (Pass rate = 0.660)									
		Histogram based gradient booster				Multi-layer perceptron			
		Precision	Recall	f1 score	support	Precision	Recall	f1 score	support
Label	0	0.51	0.58	0.54	5165	0.51	0.43	0.47	5165
	1	0.77	0.71	0.74	10047	0.73	0.79	0.76	10047
AUROC		0.7				0.67			

	HBGB Model Accuracy	MLP Accuracy	Naive Guess	
Squat1 Pass	0.55	0.52	0.88	Pass rate
Squat2 Pass	0.62	0.51	0.84	Pass rate
Squat3 Pass	0.62	0.59	0.68	Pass rate
Bench1 Pass	0.61	0.46	0.91	Pass rate
Bench2 Pass	0.57	0.55	0.54	Fail rate
Bench3 Pass	0.58	0.54	0.53	Fail rate
Deadlift1 Pass	0.64	0.38	0.95	Pass rate
Deadlift2 Pass	0.71	0.68	0.89	Pass rate
Deadlift3 Pass	0.67	0.67	0.66	Pass rate

8 BIBLIOGRAPHY

- [1] Q. J. Yap, "Raw Powerlifting is Coming at Full Force," Bruin Sports Analytics, 21 November 2018. [Online]. Available: <https://www.bruinsportsanalytics.com/post/powerlifting>. [Accessed 16 September 2021].
- [2] openpowerlifting.org, "Open Powerlifting Status," [Online]. Available: <https://www.openpowerlifting.org/status>. [Accessed 19 August 2021].
- [3] "dateutil - powerful extensions to datetime," [Online]. Available: <https://dateutil.readthedocs.io/en/stable/>. [Accessed 23 August 2021].
- [4] "USAPL Lifting Database," [Online]. Available: <https://usapl.liftingdatabase.com/>. [Accessed 25 August 2021].
- [5] R. J. Howells et al, "Impacts of squat attempt weight selection and success on powerlifting performance," *The Journal of Sports Medicine and Physical Fitness*, 15 March 2021.
- [6] J. Doron and P. Gaudreau, "A Point-by-Point Analysis of Performance in a Fencing Match: Psychological Processes Associated With Winning and Losing Streaks," *Journal of Sport and Exercise Psychology*, vol. 36, pp. 3-13, 2014.
- [7] V. Huy Chau, "Powerlifting score prediction using a machine learning method," *Mathematical Biosciences and Engineering*, vol. 18, no. 2, pp. 1040-1050, 2021.
- [8] J. Pearson, J. G. Spathis, D. J. van den Hoek, P. J. Owen, J. Weakley and C. Latella, "Effect of Competition Frequency on Strength Performance of Powerlifting Athletes," *Journal of Strength and Conditioning Research*, vol. 34, no. 5, pp. 1213-1219, 2020.
- [9] M. U. K. A. M. V. S. D. G. S., "Predicting Competitive Weightlifting Performance Using Regression and Tree-Based Algorithms," *Advanced Machine Learning Technologies and Applications, Advances in Intelligent Systems and Computing*, vol. 1141, 2021.
- [10] "Using the load-velocity relationship for 1RM prediction," *The Journal of Strength and Conditioning Research*, vol. 25, no. 1, pp. 267-70, 2011.
- [11] N. A. Coker, A. N. Varanoske, K. M. Baker, D. L. Hahs-Vaughn and A. J. Wells, "Predictors of competitive success of national-level powerlifters: a multilevel analysis," *International Journal of Performance Analysis in Sport*, vol. 18, no. 5, pp. 796-805, 2018.
- [12] C. Dabakoglu, "What is Convolutional Neural Network (CNN) ? — with Keras," 12 December 2018. [Online]. Available: <https://medium.com/@cdabakoglu/what-is-convolutional-neural-network-cnn-with-keras-cab447ad204c>. [Accessed 13 September 2021].
- [13] 3Blue1Brown, "Gradient descent, how neural networks learn | Chapter 2, Deep learning," 16 10 2017. [Online]. Available: <https://www.youtube.com/watch?v=IHZwWFHWa-w>. [Accessed 15 09 2021].

- [14] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh and A. Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1-52, 2018.
- [15] H. A. Labs, "AutoML with Hyperband," 1 July 2019. [Online]. Available: <https://www.youtube.com/watch?v=eqokKei1aEo>. [Accessed 12 09 2021].
- [16] K. Shen, "Effect of batch size on training dynamics," 19 June 2018. [Online]. Available: <https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e>. [Accessed 13 September 2021].
- [17] V. Aliyev, "Gradient Boosting Classification explained through Python," 5 September 2020. [Online]. Available: <https://towardsdatascience.com/gradient-boosting-classification-explained-through-python-60cc980eeb3d>. [Accessed 15 September 2021].
- [18] "1.11. Ensemble methods," scikit-learn, [Online]. Available: <https://scikit-learn.org/stable/modules/ensemble.html#histogram-based-gradient-boosting>. [Accessed 15 September 2021].
- [19] F. M. and H. F., "Hyperparameter Optimization," in *Automated Machine Learning*, Springer, Cham, 2019, pp. 3-33.
- [20] D. Gordon, "Bayesian Hyperparameter Optimisation," 18 August 2017. [Online]. Available: <https://www.youtube.com/watch?v=KhBB1J5JRzs>. [Accessed 13 09 2021].
- [21] J. Russell, "Mathematical Modelling of Attempt Selection," 2019. [Online]. Available: <https://startingstrength.com/article/mathematical-modeling-of-attempt-selection>. [Accessed 17 June 2021].