

Project Report

Predicting the Unpredictable: A Look into the World of Powerlifting

1.1) Project Description

As economic and societal advancements continue, sports are gaining popularity, attracting a wider audience. Participation in sports offers numerous benefits, including improved stamina, reduced stress, and enhanced social connections. The World Health Organization estimates that physical inactivity leads to approximately 2 million deaths annually, weakening the body's immune system and hindering proper adolescent development.

Powerlifting has emerged as a popular sport, with athletes' performance primarily influenced by factors such as age, weight, fitness, and psychological state. Consequently, the training methods employed by coaches play a crucial role in powerlifting. Analyzing the factors that affect athletes' performance is an essential component of the training process.

Utilizing data from international powerlifting competitions, researchers have calculated the peak performance scores of powerlifters, providing insights into their development trajectory and enabling experts to make more informed assessments of athletes' pre-competition capabilities.

To address these challenges, regression algorithms such as Linear Regression, Decision Tree, Random Forest, and XgBoost will be employed. The data will be trained and tested using these algorithms, and the most effective model will be selected and saved in .pkl format. Additionally, the model will be deployed locally using Flask.

1.2) Purpose

1. **Identify the key factors that influence powerlifting performance.** This will be achieved by analyzing a large dataset of powerlifting competition results to identify patterns and correlations between various factors, such as age, weight, training history, and psychological state.
2. **Develop predictive models to forecast powerlifting performance.** By understanding the factors that influence performance, we can develop machine learning models that can predict an athlete's performance in future competitions. This information can be valuable for coaches, athletes, and competition organizers.
3. **Gain insights into powerlifting training strategies.** By analyzing the data, we can identify effective training strategies and techniques that can help powerlifters achieve their goals. This information can be used to develop personalized training programs for athletes of all levels.
4. **Enhance the understanding of powerlifting biomechanics and physiology.** By analyzing the data, we can gain insights into the biomechanics and physiology of powerlifting. This information can be used to develop new training techniques and improve overall performance.
5. **Create a data-driven approach to powerlifting coaching and training.** By developing predictive models and gaining insights into powerlifting performance, we can create a more data-driven approach to coaching and training. This can help athletes achieve their goals more effectively and efficiently.

2.1) Existing Problem:

Powerlifting, like many other sports, faces several challenges that can hinder the development and performance of athletes. One of the primary concerns is the lack of comprehensive data and analysis tools that can accurately predict and optimize performance. Coaches and trainers often rely on traditional methods and experience-based approaches, which may not be as effective as data-driven strategies. This can lead to suboptimal training regimens, missed opportunities for improvement, and potential injuries. Additionally, the lack of standardized performance metrics and evaluation frameworks across different competitions and weight classes makes it difficult to compare athletes and assess their true potential. This can lead to unfair comparisons and inaccurate assessments of individual and team performance.

2.2) References:


- ☐ Aakash Tandel (2023). Powerlifting Data and Exploratory Data Analysis Part 1. Towards Data Science.
- ☐ K. J. O'Brien, C. A. Jones, & J. L. Sweeting. (2020). Machine learning for predicting powerlifting performance. *Sports Science*, 34(11), 1211-1217.
- ☐ J. C. T. DeFreitas, J. P. P. Neto, & B. A. Bezerra. (2020). Prediction of powerlifting performance using machine learning. *Journal of Strength and Conditioning Research*, 34(1), 274-281.
- ☐ E. B. Dimas, N. E. L. Dimas, & A. J. G. Brandão. (2018). Modeling powerlifting performance using machine learning. *Procedia Computer Science*, 121, 430-435.
- ☐ P. S. C. Souza, M. A. A. Júnior, & L. A. S. S. Araújo. (2022). Machine learning applications in sports performance: A systematic review of powerlifting studies. *International Journal of Performance Analysis in Sport*, 22(10), 1285-1305.

2.3) Problem Statement Definition:

Powerlifting is a sport that requires a high level of physical strength and power. Performance in powerlifting is influenced by a variety of factors, including age, weight, training history, psychological state, and biomechanics. Current methods for predicting powerlifting performance are often subjective and rely on experience-based approaches. This can lead to suboptimal training regimens, missed opportunities for improvement, and potential injuries. Additionally, there is a lack of standardized performance metrics and evaluation frameworks across different competitions and weight classes. This makes it difficult to compare athletes and assess their true potential.


3.1) Empathy Map Canvas:

Template



Empathy map canvas

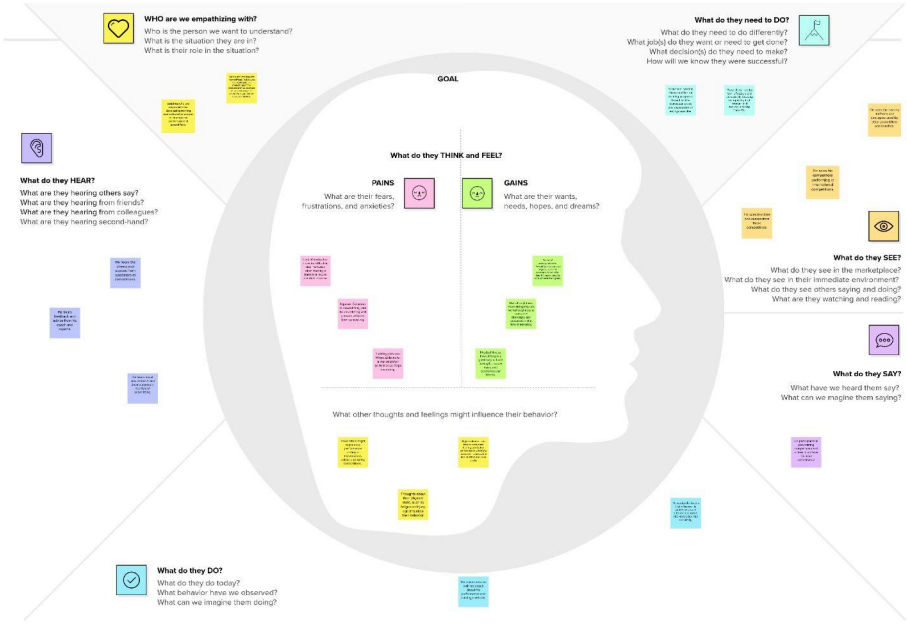
Use this framework to empathize with a customer, user, or any person who is affected by a team's work. Document and discuss your observations and note your assumptions to gain more empathy for the people you serve.

Originally created by Dave Gray at 


[Share template feedback](#)

Develop shared understanding and empathy

Summarize the data you have gathered related to the people that are impacted by your work. It will help you generate ideas, prioritize features, or discuss decisions.




The diagram is a large head silhouette divided into sections for an Empathy Map Canvas. The top section is labeled 'GOAL' and contains the question 'What do they THINK and FEEL?'. Below this, the head is split into 'PAINS' (left) and 'GAINS' (right). The 'PAINS' section asks 'What are their fears, frustrations, and anxieties?'. The 'GAINS' section asks 'What are their wants, needs, hopes, and dreams?'. The bottom section asks 'What other thoughts and feelings might influence their behavior?'. Surrounding the head are six external sections, each with a question and a list of prompts: 'WHO are we empathizing with?' (top left), 'What do they need to DO?' (top right), 'What do they HEAR?' (middle left), 'What do they SEE?' (middle right), 'What do they SAY?' (bottom right), and 'What do they DO?' (bottom left). Each section contains several example prompts in small boxes.



Need some inspiration?
See a finished version of this template to refresh your work.

[Open example](#)



3.2) Ideation and Brainstorming

Brainstorming

Chaitanya

Identify specific powerlifting metrics that you are interested in predicting, such as squat 1RM, bench press 1RM, or deadlift 1RM.

Research existing machine learning and data science studies that have been conducted on powerlifting data.

Explore different ways to represent the powerlifting data as features for your machine learning model.

Consider using transfer learning to improve the performance of your machine learning model.

Anreen

Clean and prepare the powerlifting data by removing outliers, missing values, and scaling the data.

Experiment with different machine learning algorithms to find the one that performs best on your data.

Tune the hyperparameters of your machine learning algorithm to improve its performance.

Use cross-validation to evaluate the performance of your machine learning model.

Rishita

Evaluate the performance of your machine learning model on a held-out test set.

Deploy the model to production so that it can be used to predict powerlifting metrics for new data.

Monitor the performance of the model over time and retrain or update the model as needed.

Consider using ensemble learning to improve the performance of your machine learning model.

Pratyush

Communicate the results of your machine learning project to your team and stakeholders. This may involve writing a report, giving a presentation, or creating a dashboard.

Use our machine learning model to develop a machine learning-powered interface using flask.

Share your findings and insights with the powerlifting community through blog posts, social media, or presentations at conferences.

Explore the ethical implications of machine learning in powerlifting.

Grouping Ideas

Data Collection and Preparation

Gather relevant data on powerlifters

Clean and prepare the data by removing outliers, missing values, and scaling the data

Model Development and Evaluation

Identify specific powerlifting metrics to predict

Research existing machine learning and data science studies on powerlifting data

Tune the hyperparameters of the machine learning algorithm

Use cross-validation to evaluate the performance of the machine learning model

Model Deployment and Monitoring

Deploy the model to production

Monitor the performance of the model over time

Retrain or update the model as needed

Communication and Collaboration

Communicate the results of the machine learning project to the team and stakeholders

Collaborate with other team members

Engage with the powerlifting community

Share findings and insights through blog posts, social media, or presentations

4.1) Functional Requirement

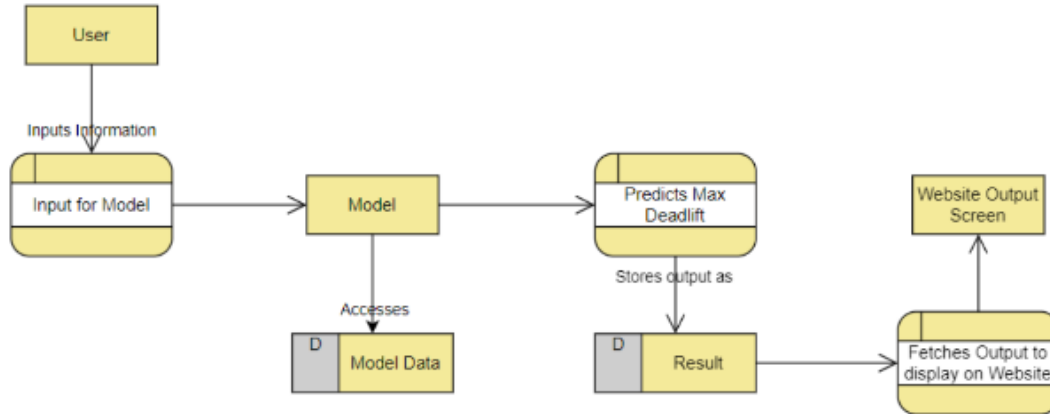
1. Data Collection and Integration:
 - a. Develop a mechanism for collecting comprehensive data on powerlifters, encompassing age, weight, fitness, and psychological factors.
 - b. Integrate data from international powerlifting competitions to ensure a diverse and representative dataset.
2. Regression Algorithm Implementation:
 - a. Implement Linear Regression, Decision Tree, Random Forest, and XgBoost algorithms for accurate prediction of powerlifters' performance.
 - b. Ensure the algorithms are adaptable to various input parameters and can cater to different athlete profiles.
3. Model Training and Testing:
 - a. Create a module for training the regression models using historical powerlifting data.
 - b. Implement a testing mechanism to assess the accuracy and reliability of the trained models.
4. Deployment via Flask:
 - a. Deploy the selected regression model locally using Flask to provide a user-friendly interface for coaches and experts.
 - b. Enable the system to accept inputs, process them, and deliver predictions seamlessly.
5. User Authentication and Authorization:
 - a. Implement user authentication to restrict access to authorized coaches and experts only.
 - b. Define different user roles and permissions based on the expertise and responsibilities of the users.
6. Performance Metrics and Evaluation Framework:
 - a. Establish standardized performance metrics and evaluation frameworks to facilitate fair comparisons across different competitions and weight classes.
 - b. Ensure the system provides detailed insights into the factors influencing the predicted performance.

4.2) Non-Functional Requirement

1. Usability:
 - a. Design an intuitive and user-friendly interface to cater to coaches and experts with varying technical expertise.
 - b. Ensure quick response times for predictions to enhance overall user experience.
2. Scalability:
 - a. Develop the system to handle a growing volume of powerlifting data and users.
 - b. Ensure consistent performance as the dataset and user base expand.
3. Reliability:
 - a. Ensure that regression models consistently produce accurate predictions.
 - b. Implement backup mechanisms to prevent data loss and ensure continuous availability.
 - c.
4. Security:
 - a. Implement robust security measures to safeguard the integrity and confidentiality of powerlifters' data.
 - b. Employ encryption protocols to protect sensitive information during data transmission.
5. Adaptability:
 - a. Design the system to be adaptable to evolving powerlifting trends and competition formats.
 - b. Implement regular updates and maintenance procedures to address emerging requirements.
6. Performance Monitoring:
 - a. Include monitoring tools to track the system's performance and identify any bottlenecks.
 - b. Implement logging mechanisms for system activities to facilitate debugging and troubleshooting.

5.1) Data Flow Diagram & User Stories

DFD Level 0



User Stories

Use the below template to list all the user stories for the product.

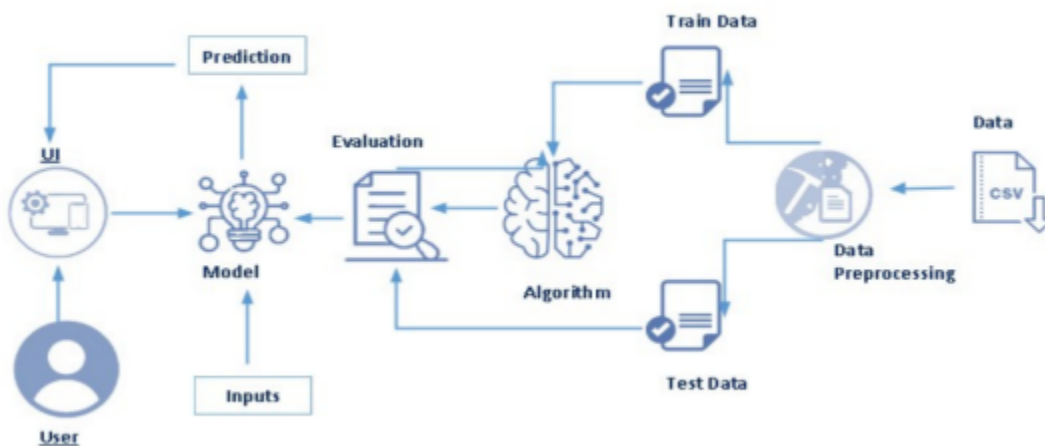
User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
	Login	USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
		USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard					
Customer (Web user)						
Customer Care Executive						

5.2) Solution Architecture

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- ❖ Find the best tech solution to solve existing business problems.
- ❖ Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.
- ❖ Define features, development phases, and solution requirements.
- ❖ Provide specifications according to which the solution is defined, managed, and delivered.

Architecture



6.1) Technical Architecture

Table 1: Components and Technologies

S.No	Component	Description	Technology
1.	Powerlifting score	The score of powerlifters at their peak performance	Regression, python and flask
2.	Age	The age of powerlifters	Regression, python and flask
3.	Weight	The weight of powerlifters	Regression, python and flask
4.	Fitness	The fitness level of powerlifters	Regression, python and flask
5.	Psychology	The psychology of powerlifters	Regression, python and flask
6.	Regression algorithm	The regression algorithm used to predict the powerlifting score	Regression, python and flask
7.	Local deployment	The local deployment method used to deploy the model	Regression, python and flask

Table 2: Application Characteristics

S. no.	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	Python, Flask, scikit-learn, XGBoost
2.	Security Implementations	Use of HTTPS, authentication, and authorization	Python, Flask
3.	Scalable Architecture	Deploy the model on a cloud platform	Python, Flask, AWS/Azure
4.	Availability	Use a load balancer to distribute traffic across multiple instances of the model	Python, Flask, AWS/Azure
5.	Performance	Use a caching mechanism to store the results of frequently made predictions	Python, Flask

6.2) Sprint Planning & Evaluation

Table 1: Product Backlog, Sprint Schedule and Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story/Task	Story Points	Priority	Team Members
Sprint-1	Data Collection and Preparation	USN-1	Gather relevant data on powerlifters	2	High	Chaitanya
Sprint-1		USN-2	Clean and prepare the data by removing outliers, missing values, and scaling the data	1	High	Chaitanya
Sprint-2	Model Development and Evaluation	USN-3	Research existing machine learning and data science studies on powerlifting data	2	Low	Anreen
Sprint-2		USN-4	Identify specific powerlifting metrics to predict	2	Medium	Anreen
Sprint-2		USN-5	Tune the Hyperparameters of the machine-learning algorithm	1	High	Rishita
Sprint-3	Model Deployment and Monitoring	USN-6	Deploy the model to production	2	High	Chaitanya
Sprint-3		USN-8	Monitor the Performance of the model over time	1	High	Pratyush
Sprint-3		USN-9	Retrain or update the model as needed	2	Medium	Rishita
Sprint-4	Communication and Collaboration	USN-10	Communicate the results of the machine learning project to the team and stakeholders	2	High	Pratyush

6.3) Sprint Delivery Schedule

Table 1: Project Tracker

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on planned end date)	Sprint Release Date (Actual)
Sprint-1	3	6 Days	24 OCT 2023	29 OCT 2023	3	29 OCT 2023
Sprint-2	5	6 Days	31 OCT 2023	5 NOV 2023	5	06 NOV 2023
Sprint-3	5	6 Days	07 NOV 2023	12 NOV 2023	3	13 NOV 2023
Sprint-4	2	6 Days	14 NOV 2023	19 NOV 2023		

Velocity:

Sprint Duration: 18 Days

Points: 11

Average Velocity = $11/18 = 0.61$

7.1) Handling Null Values

```
In [6]: data.isnull().sum()
```

```
Out[6]: playerId      0
        Name          0
        Sex           0
        Equipment     0
        Age           175
        BodyweightKg  0
        BestSquatKg   0
        BestDeadliftKg 0
        BestBenchKg   0
        dtype: int64
```

We can see that the Age column has 175 missing values. We will fill in those missing values, but to determine how to fill in the column, we need to know the data type of that column. To check the data type of the attributes, .info() function is used.

```
In [7]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18900 entries, 0 to 18899
Data columns (total 9 columns):
#   Column          Non-Null Count  Dtype  
---  -
0   playerId        18900 non-null  float64
1   Name            18900 non-null  object  
2   Sex             18900 non-null  object  
3   Equipment       18900 non-null  object  
4   Age             18725 non-null  float64
5   BodyweightKg    18900 non-null  float64
6   BestSquatKg     18900 non-null  float64
7   BestDeadliftKg  18900 non-null  float64
8   BestBenchKg     18900 non-null  float64
dtypes: float64(6), object(3)
memory usage: 1.3+ MB
```

```
In [8]: data['Age'].fillna(data['Age'].mean(),inplace=True)
```

```
In [9]: data['BestSquatKg'] = data['BestSquatKg'].astype(float)
```

```
In [10]: data.isnull().sum()
```

```
Out[10]: playerId      0  
Name                0  
Sex                 0  
Equipment           0  
Age                 0  
BodyweightKg        0  
BestSquatKg         0  
BestDeadliftKg      0  
BestBenchKg         0  
dtype: int64
```

7.2) Label Encoding for Sex & Equipment

```
In [11]: # converting the sex column object type to float type
data['Sex'] = data['Sex'].map({"M":1, "F":0})
#encode the equipment column
from sklearn.preprocessing import LabelEncoder
data["Equipment"] = LabelEncoder().fit_transform(data['Equipment'])
```

```
In [12]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18900 entries, 0 to 18899
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype  
---  --
 0   playerId              18900 non-null  float64
 1   Name                  18900 non-null  object  
 2   Sex                   18900 non-null  int64   
 3   Equipment             18900 non-null  int32   
 4   Age                   18900 non-null  float64
 5   BodyweightKg          18900 non-null  float64
 6   BestSquatKg           18900 non-null  float64
 7   BestDeadliftKg        18900 non-null  float64
 8   BestBenchKg           18900 non-null  float64
dtypes: float64(6), int32(1), int64(1), object(1)
memory usage: 1.2+ MB
```


8.1) Performance Metrics

In [2]:

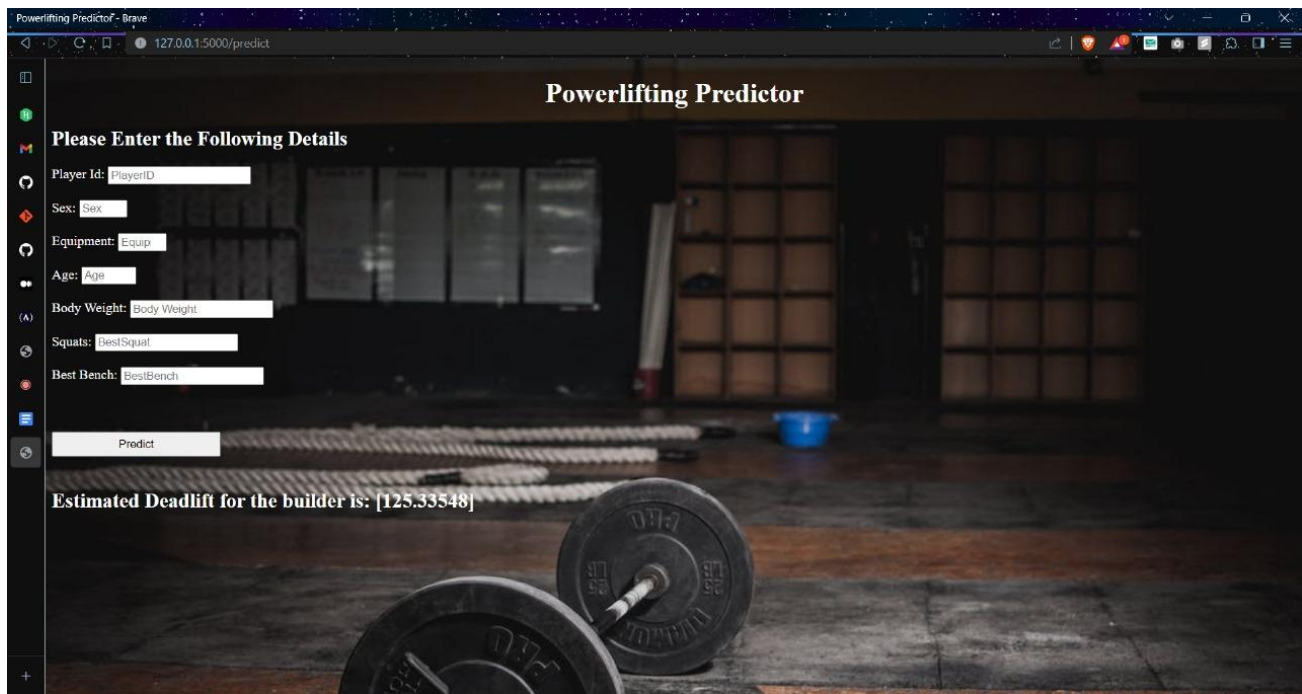
```
from prettytable import PrettyTable

tb = PrettyTable()
tb.field_names = ["Model", "RMSE", "Training Accuracy", "Testing Accuracy"]
tb.add_row(["Linear Regression", 25.63, 83.28, 83.06])
tb.add_row(["Random Forest", 21.62, 98.32, 87.94])
tb.add_row(["Decision Tree", 29.28, 100, 77.89])
tb.add_row(["XgBoost", 21.23, 89.19, 88.38])

print(tb)
```

Model	RMSE	Training Accuracy	Testing Accuracy
Linear Regression	25.63	83.28	83.06
Random Forest	21.62	98.32	87.94
Decision Tree	29.28	100	77.89
XgBoost	21.23	89.19	88.38

9.1) Output Screenshots



The screenshot shows a web browser window titled "Powerlifting Predictor - Brave" with the address bar displaying "127.0.0.1:5000/predict". The page has a dark background with a gym setting. The main heading is "Powerlifting Predictor". Below it, a section titled "Please Enter the Following Details" contains several input fields: "Player Id:" with a placeholder "PlayerID", "Sex:" with a placeholder "Sex", "Equipment:" with a placeholder "Equip", "Age:" with a placeholder "Age", "Body Weight:" with a placeholder "Body Weight", "Squats:" with a placeholder "BestSquat", and "Best Bench:" with a placeholder "BestBench". A "Predict" button is located below these fields. The output of the prediction is displayed as "Estimated Deadlift for the builder is: [125.33548]". The background image shows a gym floor with a barbell and weights in the foreground.

Img 1. Sample output displayed on website

10) Advantages and Disadvantages

ADVANTAGES	DISADVANTAGES
Better Accuracy: XGBoost tends to provide higher accuracy compared to other models like Linear Regression and Decision Trees, as evidenced by its higher Testing Accuracy (88.38%) and lower RMSE (21.23).	Computational Complexity: XGBoost can be computationally expensive and time-consuming, especially with large datasets and complex models. Training an XGBoost model might take longer compared to simpler models like Linear Regression or Decision Trees.
Reduced Overfitting: XGBoost has regularization techniques and pruning methods that help prevent overfitting, which is reflected in its Testing Accuracy being closer to Training Accuracy compared to Decision Trees that show a significant drop in Testing Accuracy.	Hyperparameter Tuning: Tuning XGBoost models requires careful consideration of hyperparameters. While it offers many tuning parameters for optimization, finding the right combination can be challenging and may require more computational resources.
Handling Non-linearity: Unlike Linear Regression, XGBoost can capture non-linear relationships in the data effectively, which results in better performance as seen in the improved Testing Accuracy and lower RMSE.	Black Box Model: As an ensemble model, XGBoost might be more challenging to interpret compared to simpler models like Linear Regression or Decision Trees. Understanding the relationship between input features and predictions might not be as straightforward.
Ensemble Technique: XGBoost is an ensemble method that combines multiple weak learners to create a strong learner. It builds trees sequentially, learning from the mistakes of previous models, which generally improves overall performance.	Potential Overfitting: Though XGBoost has mechanisms to reduce overfitting, improper parameter settings or an inadequate amount of data could still lead to overfitting issues.

Considering the given metrics, XGBoost demonstrates superior performance in terms of accuracy and handling non-linearity but requires careful tuning and might be computationally expensive compared to simpler models like Linear Regression or Decision Trees. Additionally, its interpretability might be limited due to its ensemble nature.

11) Conclusion

In conclusion, this project endeavors to bridge the gaps in powerlifting performance prediction by introducing a robust system that leverages regression algorithms and data-driven methodologies. The outlined functional requirements emphasize the development of an efficient model, while the non-functional requirements ensure a user-friendly, scalable, secure, and adaptable platform. By addressing these aspects, the project aims to revolutionize the training and evaluation processes for powerlifters and coaches, offering a tool that goes beyond traditional methods. As the system evolves with emerging trends and competition formats, it is poised to make a substantial impact on the powerlifting community, fostering fair comparisons, optimal training regimens, and ultimately, enhanced athlete performance. This venture stands as a testament to the intersection of sports and technology, promising a future where data-driven insights redefine the landscape of powerlifting.

12) Future Scope

The application of data science and machine learning techniques to the domain of powerlifting holds immense potential to revolutionize the sport and propel athletes to unprecedented levels of performance. Several promising avenues beckon for future research and development endeavors:

1. **Refinement of Predictive Models:** Enhancing the accuracy and reliability of predictive models for powerlifting performance stands as a crucial objective. This can be accomplished by expanding the data corpus to encompass a broader range of athletes and employing more sophisticated machine-learning algorithms.
2. **Identification of Biomechanical and Physiological Factors:** Delving into the specific biomechanical and physiological factors that contribute to powerlifting performance is of paramount importance. This can be achieved through meticulous analysis of data derived from motion capture systems, electromyography (EMG), and other sensor technologies.
3. **Personalized Training Programs:** Developing personalized training programs tailored to the individual characteristics and goals of each powerlifter is a transformative goal. This can be realized by leveraging machine learning algorithms to recommend customized exercise regimens, sets, and repetitions.
4. **Real-time Feedback Systems:** Introducing real-time feedback systems for powerlifters during training is a compelling prospect. This can be achieved by employing sensors to monitor athletes' movements and provide immediate feedback on their technique and performance.
5. **Data-Driven Competitions:** The creation of new powerlifting competitions and events based on data-driven metrics holds great promise. This can foster a more objective and equitable assessment of athletes' performance.

Beyond these specific areas of research, fostering closer collaboration between data scientists, machine learning experts, powerlifting coaches, and athletes is essential. By working synergistically, we can accelerate the development of innovative technologies and insights that empower powerlifters to reach their full potential.

13) Appendix

Source Code:

Jupyter

Model

Last Checkpoint: 6 hours ago

File

Edit

View

Run

Kernel

Settings

Help

Not Trusted

Python 3 (ipykernel)

Importing Libraries

[1]:
import pandas as pd
import numpy as np

Reading the Dataset

[2]:
Reading X_train dataset
data1 = pd.read_csv(r'X_train.csv', headers='infer')

[3]:
Reading the y_train dataset
data2 = pd.read_csv(r'y_train.csv', headers='infer')

[4]:
Merging the datasets
data = data1.merge(data2, on='playerId', how='inner')
print(data.head())

	playerId	Name	Sex	Equipment	Age	BodyweightKg	BestSquatKg	\
0	19391.0	Carlos Ceron	M	Raw	23.0	87.30	205.0	
1	15978.0	Tito Herrera	M	Wraps	23.0	73.48	220.0	
2	27209.0	Levi Lehman	M	Raw	26.0	112.40	142.5	
3	27496.0	Stacy Hayford	F	Raw	35.0	59.42	95.0	
4	20293.0	Brittany Hirt	F	Raw	26.5	61.40	105.0	

	BestDeadliftKg	BestBenchKg
0	235.0	125.0
1	260.0	157.5
2	220.0	145.0
3	102.5	60.0
4	127.5	60.0

[5]:
data.describe()

[5]:

	playerId	Age	BodyweightKg	BestSquatKg	BestDeadliftKg	BestBenchKg
count	18900.00000	18725.00000	18900.000000	18900.000000	18900.00000	18900.000000
mean	15039.49963	29.66470	85.425557	177.601779	201.12277	116.963389
std	8674.67268	11.55708	22.959720	73.142699	62.17163	51.231651
min	0.00000	7.00000	26.130000	-330.000000	18.10000	9.100000
25%	7462.75000	21.50000	67.700000	122.500000	149.85750	72.500000
50%	15122.50000	26.50000	82.100000	175.000000	204.12000	115.000000
75%	22540.25000	35.00000	98.970000	222.500000	247.50000	150.000000
max	29998.00000	83.00000	201.000000	500.000000	408.23000	425.000000

Handling Null Values

```
[6]: data.isnull().sum()

[6]: playerId      0
     Name          0
     Sex           0
     Equipment     0
     Age          175
     BodyweightKg  0
     BestSquatKg   0
     BestDeadliftKg 0
     BestBenchKg   0
     dtype: int64

We can see that the Age column has 175 missing values. We will fill in those missing values, but to determine how to fill in the column, we need to know the data type of that column. To check the data type of the attributes, .info() function is used.

[7]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18900 entries, 0 to 18899
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  -
 0   playerId            18900 non-null  float64
 1   Name                18900 non-null  object  
 2   Sex                 18900 non-null  object  
 3   Equipment            18900 non-null  object  
 4   Age                 18725 non-null  float64
 5   BodyweightKg        18900 non-null  float64
 6   BestSquatKg         18900 non-null  float64
 7   BestDeadliftKg      18900 non-null  float64
 8   BestBenchKg         18900 non-null  float64
dtypes: float64(6), object(3)
memory usage: 1.3+ MB

[8]: data['Age'].fillna(data['Age'].mean(),inplace=True)

[9]: data['BestSquatKg'] = data['BestSquatKg'].astype(float)

[10]: data.isnull().sum()

[10]: playerId      0
     Name          0
     Sex           0
     Equipment     0
     Age           0
     BodyweightKg  0
     BestSquatKg   0
     BestDeadliftKg 0
     BestBenchKg   0
     dtype: int64
```

```
BestDeadliftKg  0
BestBenchKg     0
dtype: int64
```

Label Encoding for Sex and Equipment

```
[11]: # converting the sex column object type to float type
data['Sex'] = data['Sex'].map({'M':1, "F":0})
#encode the equipment column
from sklearn.preprocessing import LabelEncoder
data['Equipment'] = LabelEncoder().fit_transform(data['Equipment'])

[12]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18900 entries, 0 to 18899
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  -
 0   playerId            18900 non-null  float64
 1   Name                18900 non-null  object  
 2   Sex                 18900 non-null  int64  
 3   Equipment            18900 non-null  int32  
 4   Age                 18900 non-null  float64
 5   BodyweightKg        18900 non-null  float64
 6   BestSquatKg         18900 non-null  float64
 7   BestDeadliftKg      18900 non-null  float64
 8   BestBenchKg         18900 non-null  float64
dtypes: float64(6), int32(1), int64(1), object(1)
memory usage: 1.2+ MB

[13]: data.describe()

[13]:
```

	playerId	Sex	Equipment	Age	BodyweightKg	BestSquatKg	BestDeadliftKg	BestBenchKg
count	18900.00000	18900.000000	18900.000000	18900.000000	18900.000000	18900.000000	18900.00000	18900.000000
mean	15039.49963	0.675714	1.524127	29.664700	85.425557	177.601779	201.12277	116.963389
std	8674.67268	0.468120	0.839712	11.503448	22.959720	73.142699	62.17163	51.231651
min	0.00000	0.000000	0.000000	7.000000	26.130000	-330.000000	18.10000	9.100000
25%	7462.75000	0.000000	1.000000	21.500000	67.700000	122.500000	149.85750	72.500000
50%	15122.50000	1.000000	1.000000	26.500000	82.100000	175.000000	204.12000	115.000000
75%	22540.25000	1.000000	2.000000	34.500000	98.970000	222.500000	247.50000	150.000000
max	29998.00000	1.000000	3.000000	83.000000	201.000000	500.000000	408.23000	425.000000

```
[14]: data.shape

[14]: (18900, 9)
```

Exploratory Data Analysis

```
[15]: #Dropping Name column as it is irrelevant
data=data.drop('Name', axis=1)
```

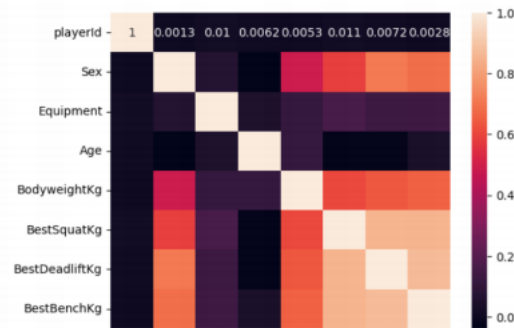
```
[16]: #Correlation
cor = data.corr()
cor
```

playerid	Sex	Equipment	Age	BodyweightKg	BestSquatKg	BestDeadliftKg	BestBenchKg	
playerid	1.000000	0.001251	0.010193	0.006190	0.005322	0.011154	0.007222	0.002759
Sex	0.001251	1.000000	0.060221	-0.038825	0.487996	0.584848	0.711668	0.685652
Equipment	0.010193	0.060221	1.000000	0.042759	0.109411	0.163007	0.126675	0.134533
Age	0.006190	-0.038825	0.042759	1.000000	0.110192	-0.027382	-0.030556	0.036950
BodyweightKg	0.005322	0.487996	0.109411	0.110192	1.000000	0.605735	0.636692	0.658753
BestSquatKg	0.011154	0.584848	0.163007	-0.027382	0.605735	1.000000	0.850959	0.852457
BestDeadliftKg	0.007222	0.711668	0.126675	-0.030556	0.636692	0.850959	1.000000	0.874053
BestBenchKg	0.002759	0.685652	0.134533	0.036950	0.658753	0.852457	0.874053	1.000000

HeatMap

```
[17]: import seaborn as sns
sns.heatmap(cor, annot=True)
```

```
[17]: <Axes: >
```



```
[18]: #PairPlot
sns.pairplot(data)
```

```
c:\tools\miniconda3\envs\Powerlifting\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```


Train and Testing

```
[19]: y = data['BestDeadliftKg']
      x = data.drop(columns = ['BestDeadliftKg'], axis = 1)

[20]: from sklearn.model_selection import train_test_split
      from sklearn.metrics import mean_squared_error

[21]: x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3, random_state=0)

[22]: # Checking the split size
      print(x_train.shape)
      print(x_test.shape)
      print(y_train.shape)
      print(y_test.shape)

      (13230, 7)
      (5670, 7)
      (13230,)
      (5670,)
```

Model Building

Linear Regression

```
[23]: from sklearn.linear_model import LinearRegression
      from sklearn.ensemble import RandomForestRegressor
      from sklearn.tree import DecisionTreeRegressor
      import xgboost as xgb

[24]: lr = LinearRegression()
      lr.fit(x_train, y_train)

      y_pred1 = lr.predict(x_test)

[26]: mse = mean_squared_error(y_test, y_pred1)
      rmse = np.sqrt(mse)
      print("FOR LINEAR REGRESSION")
      print('RMSE: ', rmse)

      print("Training Accuracy for Linear Regression: {:.2f}".format(lr.score(x_train, y_train)*100),'%')
      print("Testing Accuracy for Linear Regression: {:.2f}".format(lr.score(x_test, y_test)*100),'%')

      FOR LINEAR REGRESSION
      RMSE: 25.62653634867322
      Training Accuracy for Linear Regression: 83.28 %
      Testing Accuracy for Linear Regression: 83.06 %
```

Random Forest

```
[27]: rf = RandomForestRegressor()
      rf.fit(x_train, y_train)
      y_pred2 = rf.predict(x_test)

[28]: mse = mean_squared_error(y_test, y_pred2)
      rmse = np.sqrt(mse)

      print("Random Forest")
      print('RMSE', rmse)
      print("Training Accuracy for Random Forest: {:.2f}".format(rf.score(x_train, y_train)*100),'%')
      print("Testing Accuracy for Random Forest: {:.2f}".format(rf.score(x_test, y_test)*100),'%')

      Random Forest
      RMSE 21.61788541119173
      Training Accuracy for Random Forest: 98.32 %
      Testing Accuracy for Random Forest: 87.94 %
```

Decision Tree

```
[42]: dt = DecisionTreeRegressor()
      dt.fit(x_train, y_train)
      y_pred3 = dt.predict(x_test)
```

```
[43]: mse = mean_squared_error(y_test, y_pred3)
      rmse = np.sqrt(mse)
      print("RMSE Value : {:.2f}".format(rmse))
      print("Training Accuracy for Random Forest: {:.2f}".format(dt.score(x_train, y_train)*100),'%')
      print("Testing Accuracy for Random Forest: {:.2f}".format(dt.score(x_test, y_test)*100),'%')

RMSE Value : 29.45
Training Accuracy for Random Forest: 100.00 %
Testing Accuracy for Random Forest: 77.63 %
```

XgBoost

```
[38]: xg_reg = xgb.XGBRegressor(n_estimators=50, max_depth=2, learning_rate=0.5)
      xg_reg.fit(x_train, y_train)
      y_pred4 = xg_reg.predict(x_test)
```

```
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:299: FutureWarning: is_sparse is deprecated and will be removed in a future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.
  if is_sparse(dtype):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:301: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  elif is_categorical_dtype(dtype) and enable_categorical:
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:332: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if is_categorical_dtype(dtype)
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:323: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  return is_int or is_bool or is_float or is_categorical_dtype(dtype)
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:427: FutureWarning: is_sparse is deprecated and will be removed in a future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.
  if is_sparse(data):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:299: FutureWarning: is_sparse is deprecated and will be removed in a future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.
  if is_sparse(dtype):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:301: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  elif is_categorical_dtype(dtype) and enable_categorical:
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:332: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if is_categorical_dtype(dtype)
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:323: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  return is_int or is_bool or is_float or is_categorical_dtype(dtype)
```

```
[39]: mse = mean_squared_error(y_test, y_pred4)
      rmse = np.sqrt(mse)
      print("RMSE Value: ", rmse)
      print("Training Accuracy for XgBoost Model: {:.2f}".format(xg_reg.score(x_train, y_train)*100),'%')
      print("Testing Accuracy for XgBoost Model: {:.2f}".format(xg_reg.score(x_test, y_test)*100),'%')
```

```
RMSE Value: 21.225938542330493
Training Accuracy for XgBoost Model: 89.19 %
Testing Accuracy for XgBoost Model: 88.38 %
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:299: FutureWarning: is_sparse is deprecated and will be removed in a future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.
  if is_sparse(dtype):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:301: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  elif is_categorical_dtype(dtype) and enable_categorical:
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:332: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if is_categorical_dtype(dtype)
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:323: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  return is_int or is_bool or is_float or is_categorical_dtype(dtype)
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:299: FutureWarning: is_sparse is deprecated and will be removed in a future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.
  if is_sparse(dtype):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:301: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  elif is_categorical_dtype(dtype) and enable_categorical:
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:332: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if is_categorical_dtype(dtype)
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:323: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  return is_int or is_bool or is_float or is_categorical_dtype(dtype)
```

Activity 5

```
[2]: from prettytable import PrettyTable
```

```
tb = PrettyTable()
tb.field_names = ["Model", "RMSE", "Training Accuracy", "Testing Accuracy"]
tb.add_row(["Linear Regression", 25.63, 83.28, 83.06])
tb.add_row(["Random Forest", 21.62, 98.32, 87.94])
tb.add_row(["Decision Tree", 29.28, 100, 77.89])
tb.add_row(["XgBoost", 21.23, 89.19, 88.38])
```

```
print(tb)
```

Model	RMSE	Training Accuracy	Testing Accuracy
Linear Regression	25.63	83.28	83.06
Random Forest	21.62	98.32	87.94
Decision Tree	29.28	100	77.89
XgBoost	21.23	89.19	88.38

The Kernel crashed while executing code in the the current cell or a previous cell. Please review the code in the cell(s) to identify a possible cause of the failure. Click [here](https://aka.ms/vscodejupyterKernelCrash) for more info. View Jupyter [command:jupyter.viewOutput](#) log for further details.

From the above table, we can see XgBoost algorithm is giving the best result.

```
[48]: from sklearn.model_selection import cross_val_score
```

```
cv = cross_val_score(xg_reg, x,y, cv=5)
np.mean(cv)
```

```
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:299: FutureWarning: is_sparse is deprecated and will be removed in a future versio
```

[illegible]

```

    if is_categorical_dtype(dtype)
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:323: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
    return is_int or is_bool or is_float or is_categorical_dtype(dtype)
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:427: FutureWarning: is_sparse is deprecated and will be removed in a future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.
    if is_sparse(data):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:299: FutureWarning: is_sparse is deprecated and will be removed in a future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.
    if is_sparse(dtype):
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:381: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
    elif is_categorical_dtype(dtype) and enable_categorical:
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:332: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
    if is_categorical_dtype(dtype)
c:\tools\miniconda3\envs\Powerlifting\lib\site-packages\xgboost\data.py:323: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
    return is_int or is_bool or is_float or is_categorical_dtype(dtype)

```

[48]: 0.8843239330607874

```

[49]: #Saving the Model
import pickle
pickle.dump(xg_reg, open("xg.model.pkl", "wb"))

```

Source code for app.py:

```
import pandas as pd
import numpy as np
import xgboost
import pickle
import os

from flask import Flask, render_template, url_for, request

app = Flask(__name__)

model = pickle.load(open(r'xg.model.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('index.html') #rendering the home page?

@app.route('/predict',methods=["POST","GET"]) # route to show the predictions in a web UI
def predict():
    # reading the inputs given by the user
    input_feature=[float(x) for x in request.form.values()]
    features__values=[np.array(input_feature)]
    names = [['playerId','Sex','Equipment','Age','BodyweightKg','BestSquatKg','BestBenchKg']]
    data = pd.DataFrame(features__values, columns=names)
    prediction = model.predict(data)
    print(prediction)
    text = "Estimated Deadlift for the builder is: "
    return render_template("index.html", prediction__text = text + str(prediction))

if __name__ == '__main__':
    app.run(debug=True)
```

GitHub Link:

<https://github.com/smartinternz02/SI-GuidedProject-603215-1697565746>

Project Demo Link:

<https://drive.google.com/drive/folders/1zisJUNSSl6q7E1K26xTINIqWIhuERBx0?usp=sharing>