Powerlifter's 1 Rep Max Deadlift Analysis (by Andrew Dettor)

August 19, 2021

1 *Table of Contents*

2 *Introduction*

Going for a one rep max on a lift takes a lot of physical and mental preparation. Sometimes it can even cause injury if one chooses too high a weight. Is there a way to predict what someone's one rep max on the deadlift will be without actually doing it? With the information in this dataset, I think I can get pretty close.

This type of analysis can be applied to improve performance in any sport. It is especially pertinent to competitive powerlifting, obviously. What factors influence the main goal that needs to improve (i.e. points, speed, strength)? My goal is to understand the underlying processes of this phenomenon to give some reasoning behind why some people do better and some do worse. It can also be used to scope out the competition to guess how well they will

According to the dataset located here (Powerlifting Database), "this dataset is a snapshot of the OpenPowerlifting database as of April 2019. OpenPowerlifting is creating a public-domain archive of powerlifting history. Powerlifting is a sport in which competitors compete to lift the most weight for their class in three separate barbell lifts: the Squat, Bench, and Deadlift."

This project is written in Python. The output is from the Jupyter Notebook of my analysis.

My email: dettor.andrew@gmail.com

2.1 Goals:

- Deal with missing values present in dataset
- Explore distributions of features and their relationships with the target feature I chose, Best3DeadliftKg (best deadlift from 3 attempts)
- Preprocess/clean data in order to model it (look at outliers/skewed distributions/standardization)
- Test out different Regression models to predict Best3DeadliftKg and compare their performance
- Tune hyperparameters of the best model
- Explore which features were the most impactful
- Explore interactions between features

2.2 Basic Imports

[34]:
```
# Basic Important Imports
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
import matplotlib.pyplot as plt
%matplotlib inline
```

[35]:
```
import warnings
warnings.filterwarnings('ignore')
```

---

3 *Read in the Data*

[36]:
```
fname = "openpowerlifting.csv"
X = pd.read_csv(fname, parse_dates=True)
X.head()
```

[36]:
```
                Name Sex Event Equipment   Age AgeClass Division  BodyweightKg  \
0       Abbie Murphy   F   SBD     Wraps  29.0    24-34     F-OR          59.8
1        Abbie Tuong   F   SBD     Wraps  29.0    24-34     F-OR          58.5
2     Ainslee Hooper   F     B       Raw  40.0    40-44     F-OR          55.4
3   Amy Moldenhauer   F   SBD     Wraps  23.0    20-23     F-OR          60.0
4       Andrea Rowan   F   SBD     Wraps  45.0    45-49     F-OR         104.0

  WeightClassKg  Squat1Kg  …  McCulloch  Glossbrenner  IPFPoints  Tested  \
0            60      80.0  …     324.16        286.42     511.15     NaN
1            60     100.0  …     378.07        334.16     595.65     NaN
2            56       NaN  …      38.56         34.12     313.97     NaN
3            60    -105.0  …     345.61        305.37     547.04     NaN
4           110     120.0  …     338.91        274.56     550.08     NaN

   Country Federation        Date MeetCountry MeetState       MeetName
0      NaN    GPC-AUS  2018-10-27   Australia       VIC  Melbourne Cup
1      NaN    GPC-AUS  2018-10-27   Australia       VIC  Melbourne Cup
2      NaN    GPC-AUS  2018-10-27   Australia       VIC  Melbourne Cup
3      NaN    GPC-AUS  2018-10-27   Australia       VIC  Melbourne Cup
4      NaN    GPC-AUS  2018-10-27   Australia       VIC  Melbourne Cup

[5 rows x 37 columns]
```

[37]:
```
X.shape
```

[37]:
```
(1423354, 37)
```

There are 1423354 observations in 37 variables.

[38]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1423354 entries, 0 to 1423353
Data columns (total 37 columns):
 #   Column            Non-Null Count    Dtype
---  ------            --------------    -----
 0   Name              1423354 non-null  object
 1   Sex               1423354 non-null  object
 2   Event             1423354 non-null  object
 3   Equipment         1423354 non-null  object
 4   Age               757527 non-null   float64
 5   AgeClass          786800 non-null   object
 6   Division          1415176 non-null  object
```

```
 7   BodyweightKg     1406622 non-null  float64
 8   WeightClassKg    1410042 non-null  object
 9   Squat1Kg         337580 non-null   float64
 10  Squat2Kg         333349 non-null   float64
 11  Squat3Kg         323842 non-null   float64
 12  Squat4Kg         3696 non-null     float64
 13  Best3SquatKg     1031450 non-null  float64
 14  Bench1Kg         499779 non-null   float64
 15  Bench2Kg         493486 non-null   float64
 16  Bench3Kg         478485 non-null   float64
 17  Bench4Kg         9505 non-null     float64
 18  Best3BenchKg     1276181 non-null  float64
 19  Deadlift1Kg      363544 non-null   float64
 20  Deadlift2Kg      356023 non-null   float64
 21  Deadlift3Kg      339947 non-null   float64
 22  Deadlift4Kg      9246 non-null     float64
 23  Best3DeadliftKg  1081808 non-null  float64
 24  TotalKg          1313184 non-null  float64
 25  Place            1423354 non-null  object
 26  Wilks            1304407 non-null  float64
 27  McCulloch        1304254 non-null  float64
 28  Glossbrenner     1304407 non-null  float64
 29  IPFPoints        1273286 non-null  float64
 30  Tested           1093892 non-null  object
 31  Country          388884 non-null   object
 32  Federation       1423354 non-null  object
 33  Date             1423354 non-null  object
 34  MeetCountry      1423354 non-null  object
 35  MeetState        941545 non-null   object
 36  MeetName         1423354 non-null  object
dtypes: float64(22), object(15)
memory usage: 401.8+ MB
```

Anthing with a datatype of “object” is a string and anything with a datatype of “float64” is a number, obviously. However, some of the string datatypes could be numerical in nature. I need to do further investigation.

[39]:
```
X.describe().round(2)
```

[39]:
```
             Age  BodyweightKg   Squat1Kg   Squat2Kg   Squat3Kg  Squat4Kg  \
count  757527.00    1406622.00  337580.00  333349.00  323842.00   3696.00
mean       31.50         84.23     114.10      92.16      30.06     71.36
std        13.37         23.22     147.14     173.70     200.41    194.52
min         0.00         15.10    -555.00    -580.00    -600.50   -550.00
25%        21.00         66.70      90.00      68.00    -167.50   -107.84
50%        28.00         81.80     147.50     145.00     110.00    135.00
75%        40.00         99.15     200.00     205.00     192.50    205.00
max        97.00        258.00     555.00     566.99     560.00    505.50
```

```
       Best3SquatKg   Bench1Kg   Bench2Kg   Bench3Kg  …  Deadlift1Kg  \
count    1031450.00  499779.00  493486.00  478485.00  …    363544.00
mean         174.00      83.89      55.07     -18.52  …       162.70
std           69.24     105.20     130.30     144.23  …       108.68
min         -477.50    -480.00    -507.50    -575.00  …      -461.00
25%          122.47      57.50     -52.50    -140.00  …       125.00
50%          167.83     105.00      95.00     -60.00  …       180.00
75%          217.50     145.00     145.00     117.50  …       226.80
max          575.00     467.50     487.50     478.54  …       450.00

       Deadlift2Kg  Deadlift3Kg  Deadlift4Kg  Best3DeadliftKg     TotalKg  \
count    356023.00    339947.00      9246.00       1081808.00  1313184.00
mean        130.23        13.00        78.91           187.26      395.61
std         162.68       215.05       192.61            62.33      201.14
min        -470.00      -587.50      -461.00          -410.00        2.50
25%         115.00      -210.00      -110.00           138.35      232.50
50%         177.50       117.50       145.15           185.00      378.75
75%         230.00       205.00       210.00           230.00      540.00
max         460.40       457.50       418.00           585.00     1367.50

            Wilks   McCulloch  Glossbrenner   IPFPoints
count  1304407.00  1304254.00    1304407.00  1273286.00
mean       288.22      296.07        271.85      485.43
std        123.18      124.97        117.56      113.35
min          1.47        1.47          1.41        2.16
25%        197.90      204.82        182.81      402.86
50%        305.20      312.03        285.94      478.05
75%        374.56      383.76        355.28      559.70
max        779.38      804.40        742.96     1245.93

[8 rows x 22 columns]
```

All the lift Kg features have negative values, which indicate that they failed that lift with that weight. I'm assuming NaN means the lift wasn't even attempted.

3.1 Confusing Names for Features:

- Squat3Kg - Lifter's 3rd squat attempt weight. Negative means the attempt was failed. "Ignore NaN" according to the dataset creator, which doesn't make sense to me. How do I just 'ignore' NaN?
- Bench1Kg - Lifter's 1st bench attempt weight.
- Best3BenchKg - Lifter's highest weight benched out of 3 attempts.
- Wilks/McCulloch/Glossbrenner/IPFPoints - Metrics used to measure a lifter's performance against others. Take into account bodyweight and powerlifting total (which is equal to Best3BenchKg + Best3SquatKg + Best3DeadliftKg), among other things.

4 *Data Cleaning*

4.1 Missing Values

[40]:
```
# What percent of each feature is missing?

def percent_missing(X):
    lst = []
    for cname in X.columns.tolist():
        percentage = round((100*X[cname].isna().sum()/X[cname].isna().count()),␣
 ↪2)
        if percentage > 0.0:
            lst.append((cname, percentage))

    lst.sort(reverse=True, key = lambda x: x[1])

    for element in lst:
        print(element[0] + ": " + str(element[1]) + "%")

percent_missing(X)
```

```
Squat4Kg: 99.74%
Deadlift4Kg: 99.35%
Bench4Kg: 99.33%
Squat3Kg: 77.25%
Squat2Kg: 76.58%
Squat1Kg: 76.28%
Deadlift3Kg: 76.12%
Deadlift2Kg: 74.99%
Deadlift1Kg: 74.46%
Country: 72.68%
Bench3Kg: 66.38%
Bench2Kg: 65.33%
Bench1Kg: 64.89%
Age: 46.78%
AgeClass: 44.72%
MeetState: 33.85%
Best3SquatKg: 27.53%
Best3DeadliftKg: 24.0%
Tested: 23.15%
IPFPoints: 10.54%
Best3BenchKg: 10.34%
McCulloch: 8.37%
Wilks: 8.36%
Glossbrenner: 8.36%
TotalKg: 7.74%
BodyweightKg: 1.18%
WeightClassKg: 0.94%
```

```
Division: 0.57%
```

The target feature for this analysis is Best3DeadliftKg, which is the best deadlift from all attempts.

4.2 Dropping Columns

I want to remove any features that tell us about the lifter's overall performance or deadlift performance, to prevent target leakage.

[41]:
```
# Removing Deadlift1Kg, Deadlift2Kg, Deadlift3Kg, TotalKg and Place because␣
↪target leakage

X = X.drop(["Deadlift1Kg", "Deadlift2Kg", "Deadlift3Kg", "TotalKg", "Place"],␣
↪axis=1)
```

The following featues are metrics are used to compare lifters against each other, but they use TotalKg in their calculation, so they're another source of target leakage.

[42]:
```
# Removing the lifter metrics because they use Total in their calculation

X = X.drop(["Wilks", "Glossbrenner", "IPFPoints", "McCulloch"], axis=1)
```

Over 99% of the following features are missing, so there's not much use imputing them.

[43]:
```
# Remove Squat4Kg, Deadlift4Kg, and Bench4Kg because they're rarely used

X = X.drop(["Squat4Kg", "Deadlift4Kg", "Bench4Kg"], axis=1)
```

These next features are just ordinal encodings of the Age and BodyweightKg features, so remove them. These features provide more granularity.

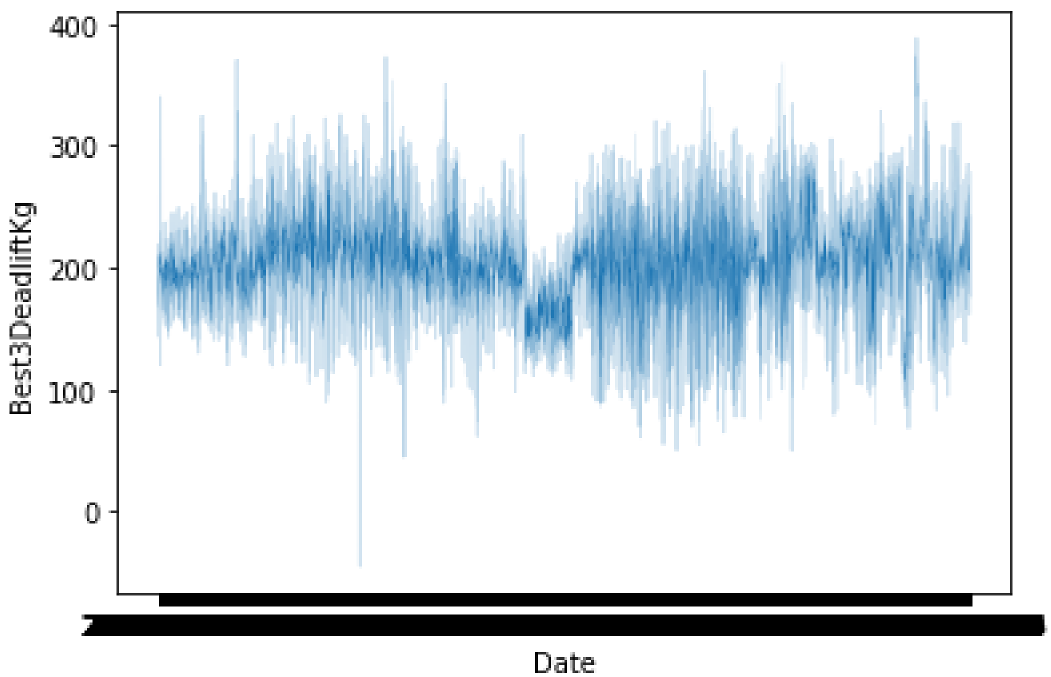
[44]:
```
# Remove AgeClass and WeightClass because they can be substituted by Age and␣
↪BodyWeightKg

X = X.drop(["AgeClass", "WeightClassKg"], axis=1)
```

The meet the event took place shouldn't matter.

[45]:
```
# Remove MeetName because I don't want it to affect predictions

X = X.drop(["MeetName"], axis=1)
```

[46]:
```
# Look at powerlifter's mean deadlifts over time
sns.lineplot(x=X["Date"], y=X["Best3DeadliftKg"], linewidth=.1,␣
↪estimator='mean')
```

[46]:
```
<AxesSubplot:xlabel='Date', ylabel='Best3DeadliftKg'>
```



There is not a noticeable trend. I would think people's deadlifts increase over time because of advances in exercise and nutrition science. Maybe that's too naive because there are always experienced and inexperienced people going to these events. But of course the graph is super noisy because within it are differences in sex/age/body weight/country/federation/division.

[47]:
```
# I don't want Date to be a factor in predicting deadlifts
# Also it doesn't seem to have any effect on Best Deadlift
# Drop Date

X = X.drop(["Date"], axis=1)
```

[48]:
```
X.columns
```

[48]:
```
Index(['Name', 'Sex', 'Event', 'Equipment', 'Age', 'Division', 'BodyweightKg',
       'Squat1Kg', 'Squat2Kg', 'Squat3Kg', 'Best3SquatKg', 'Bench1Kg',
       'Bench2Kg', 'Bench3Kg', 'Best3BenchKg', 'Best3DeadliftKg', 'Tested',
       'Country', 'Federation', 'MeetCountry', 'MeetState'],
      dtype='object')
```

4.3 Dropping Rows

After dropping entire columns, I want to drop entire rows that don't have the target feature or features that I intuitively think are important. I am also removing failed lifts.

```
[49]: # Want perfect data for the target feature
X = X.loc[(X["Best3DeadliftKg"].isna() == False) & (X["Best3DeadliftKg"] > 0)]
```

Want to limit the Event feature to having all three lifts. (SBD = Squat Bench Deadlift)

```
[50]: X["Event"].value_counts()
```

```
[50]: SBD    997487
D       55481
BD      26894
SD       1290
Name: Event, dtype: int64
```

```
[51]: X = X.loc[(X["Event"] == "SBD")]
```

```
[52]: # Can now drop Event
X = X.drop("Event", axis=1)
```

```
[53]: print(min(X["Best3SquatKg"]), max(X["Best3SquatKg"]))
print(min(X["Best3BenchKg"]), max(X["Best3BenchKg"]))
```

```
-445.0 575.0
-362.5 455.86
```

For best squat and bench, there are some people who didn't succeed once, and there are some people who have no recording of anything. I already limited the analysis to the SBD, but I want to limit if further to people who were successful in those other two lifts in that event.

```
[54]: X = X.loc[(X["Best3SquatKg"].isna() == False) & (X["Best3SquatKg"] > 0)]
X = X.loc[(X["Best3BenchKg"].isna() == False) & (X["Best3BenchKg"] > 0)]
```

```
[55]: print(min(X["BodyweightKg"]), max(X["BodyweightKg"]))
print(min(X["Age"]), max(X["Age"]))
```

```
17.69 250.05
0.0 95.5
```

Some people's Age and BodyweightKg were not recorded, so I can impute them in some way. For some reason, there are a few subjects who were 0 years old. Drop them.

```
[56]: # Drop those subjects who were 0 years old

X = X.loc[X["Age"] != 0.0]
```

```
[57]: percent_missing(X)
```

```
Country: 78.36%
Squat3Kg: 69.1%
Bench3Kg: 69.1%
Squat2Kg: 68.26%
Bench2Kg: 68.2%
```

```
Bench1Kg: 67.92%
Squat1Kg: 67.91%
Age: 52.28%
MeetState: 26.39%
Tested: 19.09%
Division: 0.55%
BodyweightKg: 0.49%
```

4.4 Imputation

4.4.1 Custom Imputation

[59]:
```
# Make the values in these columns easier to work with
# Removes NaNs in the process
# Categorizing these columns removes the possible target leakage from the␣
↪deadlift columns
# Creates a separate feature for if the lift attempt was failed/successful/
↪unknown outcome
import math
attemptCols = ["Squat1Kg", "Squat2Kg", "Squat3Kg", "Bench1Kg", "Bench2Kg",␣
↪"Bench3Kg"]

def attemptTransformer(datapoint):
    if math.isnan(datapoint):
        return "Unknown"
    elif datapoint <= 0:
        return "Fail"
    else:
        return "Success"

for col in attemptCols:
    X[col] = X[col].apply(lambda x: attemptTransformer(x))
```

4.4.2 Simple Imputer

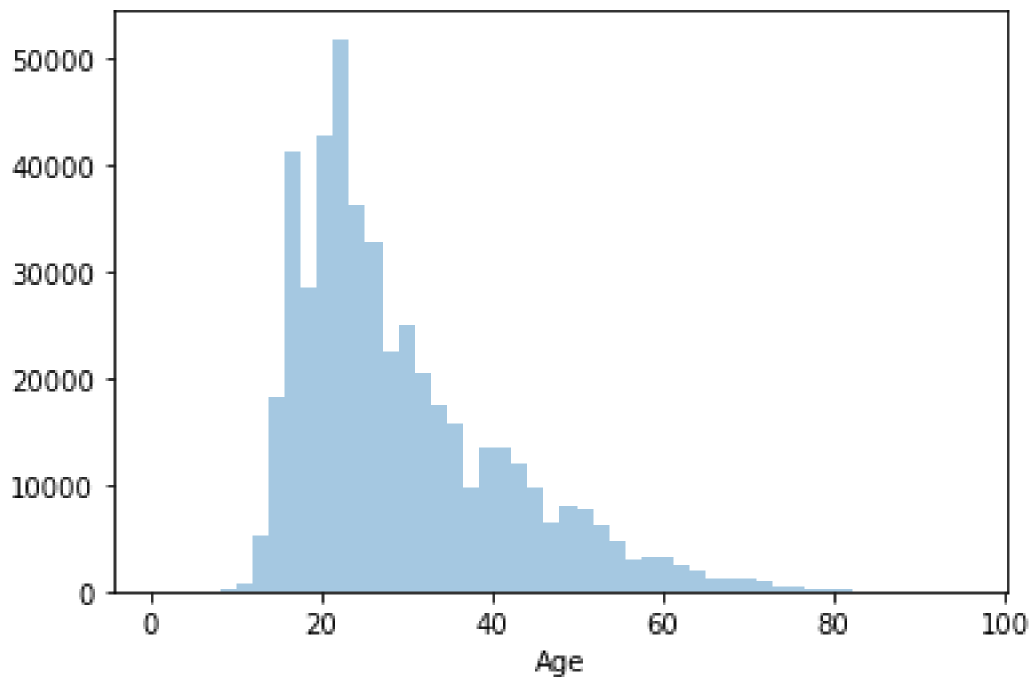
[60]:
```
# Use simple imputer to impute these categorical features
from sklearn.impute import SimpleImputer

sImputeCols = ["Country", "MeetState", "Division", "Tested"]

sImputer = SimpleImputer(strategy="constant", fill_value="Not Provided")
X[sImputeCols] = sImputer.fit_transform(X[sImputeCols])
```

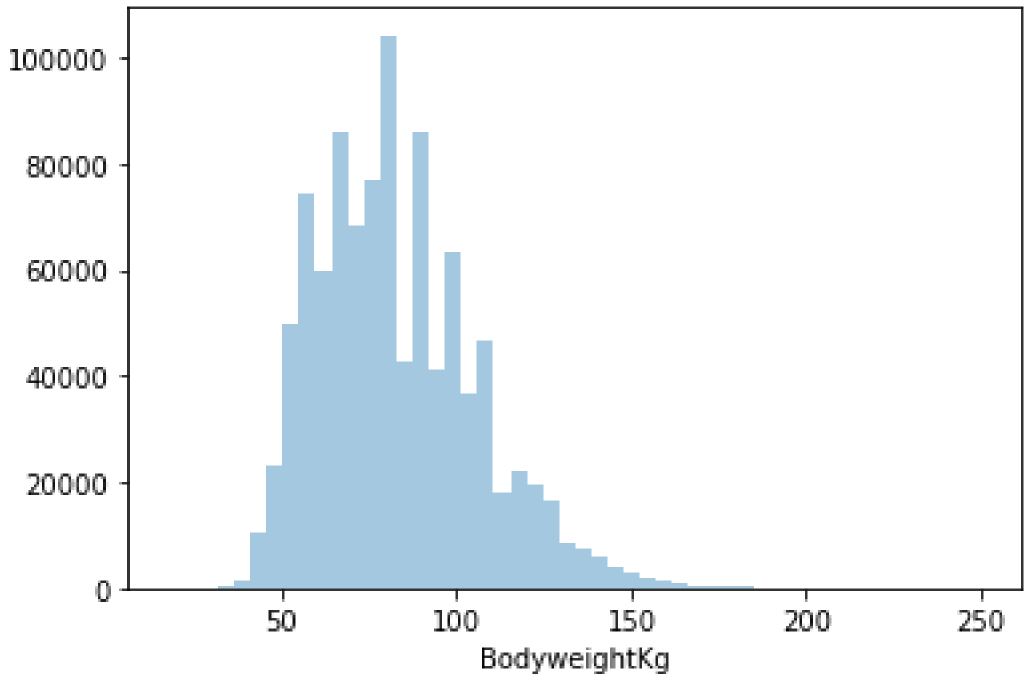
[61]:
```
percent_missing(X)
```

```
Age: 52.28%
BodyweightKg: 0.49%
```

I want to make a more educated guess about these 2 features because they are important, judging by what I know about weightlifting. Age would be hard to impute since so many values are missing, but the BodyweightKg should be okay. I would use a KNNImputer on everything but that would take too long because this dataset is so large. Instead, I'll do it based on a few other features. I would like to include Sex in this but it makes the KNN take too long. I hope th eeffect of Sex is present in body weight and the other lifts.

[62]:
```
# What do these look like before imputation?
sns.distplot(a=X["Age"], kde=False)
plt.show()
sns.distplot(a=X["BodyweightKg"], kde=False)
plt.show()
```





Age seems to be right skewed and has the most people in their twenties. Body weight has many people at the 10kg marks I assume; that's why there are so many tall bars spaced out as the are.

Going to impute Bodyweight and Age based on the Scikit-learn IterativeImputer. See the preprocessing pipeline later on in the notebook for that.

4.4.3 Iterative Imputer

[64]:
```
X[["Age", "BodyweightKg"]] = IterativeImputer(missing_values=np.nan,␣
↪max_iter=10).fit_transform(X[["Age", "BodyweightKg"]])
```

[65]:
```
# Check if there are any NaNs left

percent_missing(X)
```

[66]:
```
# See how many training examples and columns we're left with

X.shape
```

[66]:
```
(985418, 20)
```

[ ]:
```
# Reindex

X.index = np.arange(X.shape[0])
```

```
[ ]: # look to see if all columns are good

     X.columns
```

```
[ ]: Index(['Name', 'Sex', 'Equipment', 'Age', 'Division', 'BodyweightKg',
            'Squat1Kg', 'Squat2Kg', 'Squat3Kg', 'Best3SquatKg', 'Bench1Kg',
            'Bench2Kg', 'Bench3Kg', 'Best3BenchKg', 'Tested', 'Country',
            'Federation', 'MeetCountry', 'MeetState'],
           dtype='object')
```

```
[ ]: # Get X and y

     X = X.drop("Best3DeadliftKg", axis=1)
     y = X["Best3DeadliftKg"]
```

```
[ ]: # Sometimes this weird column gets added
     if "Unnamed: 0" in X.columns:
         X = X.drop("Unnamed: 0", axis=1)
```

```
[205]: # Make lists of the numerical and categorical columns
       categorical = [cname for cname in X.columns.tolist() if X[cname].dtype ==␣
        ↪"object"]
       numerical = [cname for cname in X.columns.tolist() if X[cname].dtype ==␣
        ↪"float64"]
```

---

5 *Exploratory Data Analysis*

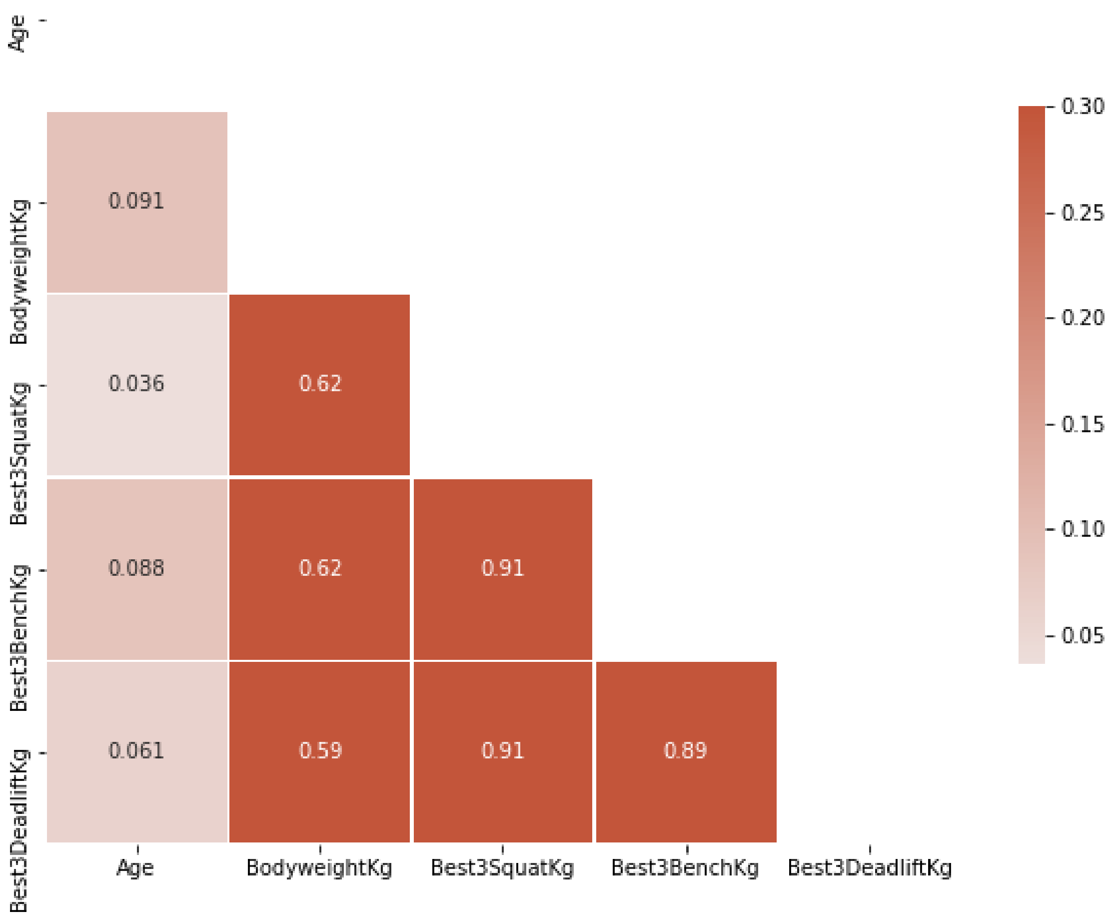
5.1 Numerical Features

5.1.1 Correlation Heatmap

```
[174]: # Create a correlation heatmap between the numerical values
       # Source: https://seaborn.pydata.org/examples/many_pairwise_correlations.html

       # Compute the correlation matrix
       corr = pd.concat([X[numerical], y], axis=1).corr()

       # Generate a mask for the upper triangle
       mask = np.triu(np.ones_like(corr, dtype=bool))

       # Set up the matplotlib figure
       f, ax = plt.subplots(figsize=(10,10))

       # Generate a custom diverging colormap
       cmap = sns.diverging_palette(230, 20, as_cmap=True)
```

```
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, annot=True, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

[174]: <AxesSubplot:>

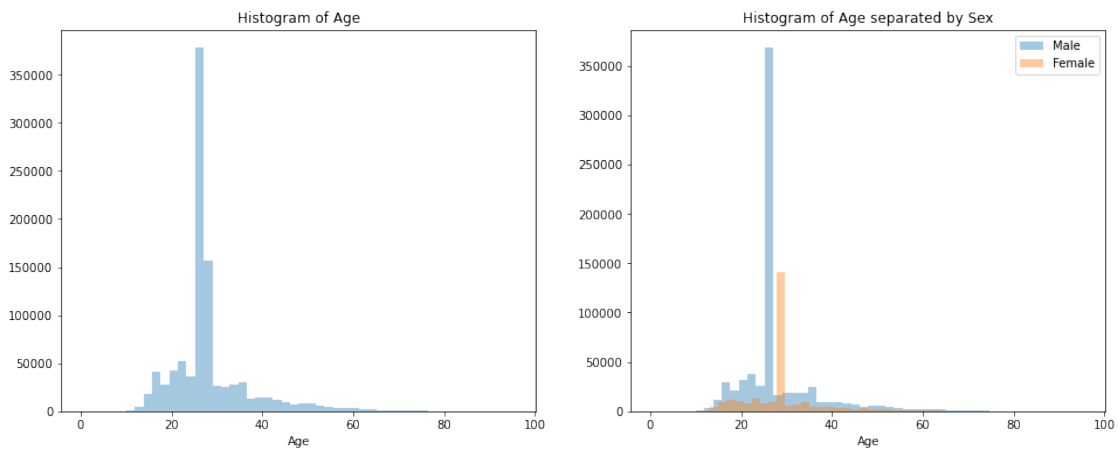


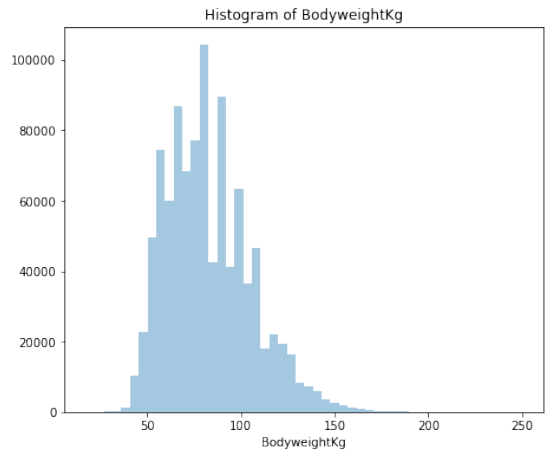
Very high correlation between best squat, bench, and deadlift. Bodyweight has some correlation with best squat, bench, and deadlift. Age has a little correlation.

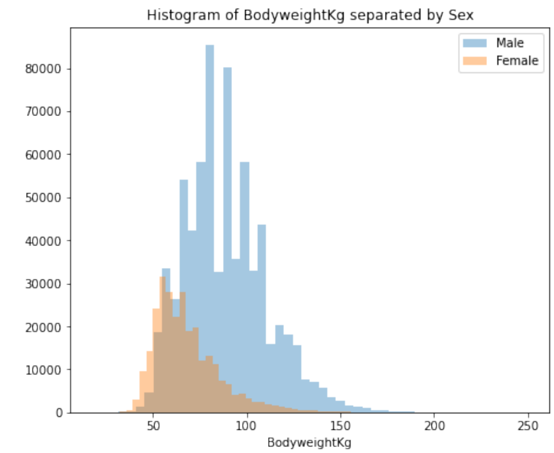
5.1.2 Histograms

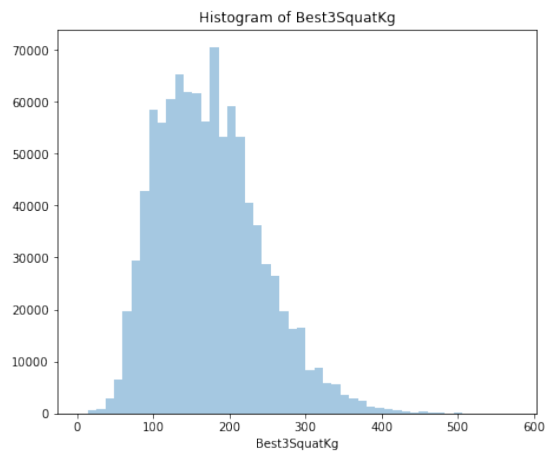
[188]:
```
# See distributions based on Sex
male = X.loc[X["Sex"] == "M"].index
female = X.loc[X["Sex"] == "F"].index

# print histograms for numerical features in X
```

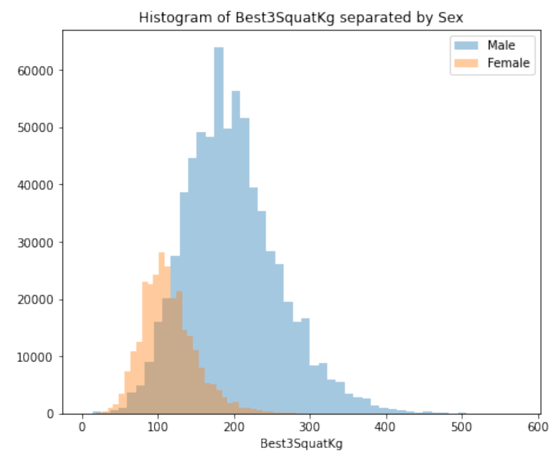
```python
for cname in numerical:
    fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=[16,6])

    sns.distplot(a=X[cname], kde=False, ax=ax1)
    ax1.set_title(f"Histogram of {cname}")

    sns.distplot(a=X[cname].iloc[male], kde=False, ax=ax2)
    ax2 = sns.distplot(a=X[cname].iloc[female], kde=False, ax=ax2)
    ax2.set_title(f"Histogram of {cname} separated by Sex")
    ax2.legend(labels=["Male", "Female"])

    plt.show()

# print histograms for y because it's also numerical
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=[16,6])

sns.distplot(a=y, kde=False, ax=ax1)
ax1.set_title("Histogram of Best3DeadliftKg")

sns.distplot(a=y.iloc[male], kde=False, ax=ax2)
ax2 = sns.distplot(a=y.iloc[female], kde=False, ax=ax2)
ax2.set_title("Histogram of Best3DeadliftKg separated by Sex")
ax2.legend(labels=["Male", "Female"])

plt.show()
```

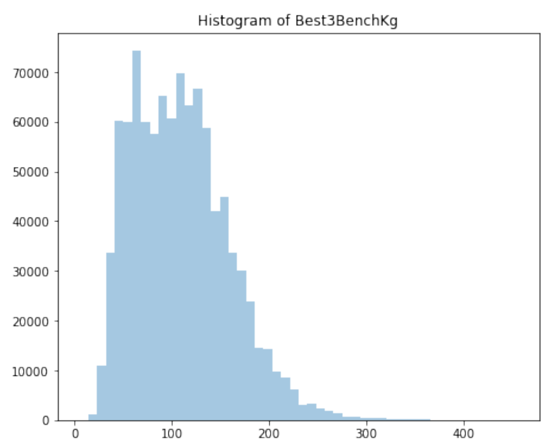


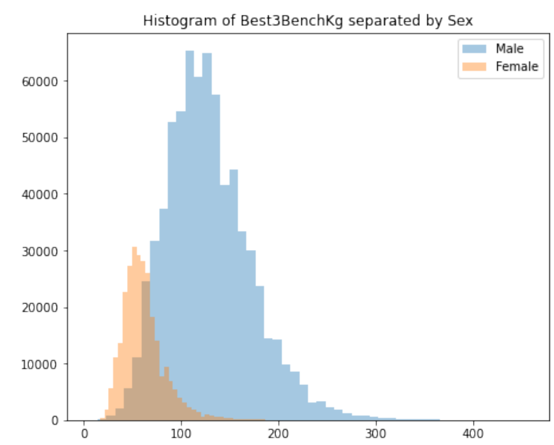

Histogram of BodyweightKg
BodyweightKg

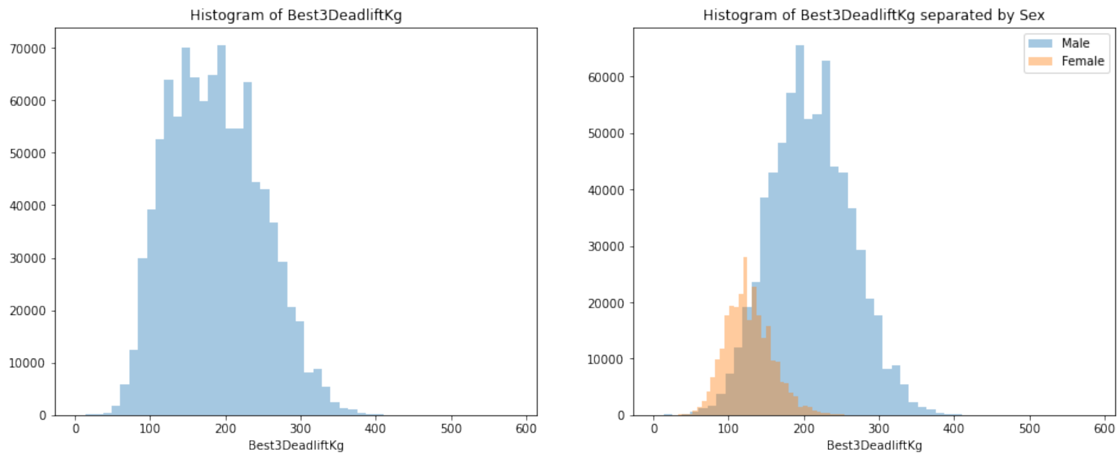

Histogram of BodyweightKg separated by Sex
Male
Female
BodyweightKg


Histogram of Best3SquatKg
Best3SquatKg


Histogram of Best3SquatKg separated by Sex
Male
Female
Best3SquatKg


Histogram of Best3BenchKg
Best3BenchKg


Histogram of Best3BenchKg separated by Sex
Male
Female
Best3BenchKg



Age: There were so many missing values for Age that the imputed values stick out like a sore thumb. I don't know if there was much I could do about that. Females seem to be older than males on average.

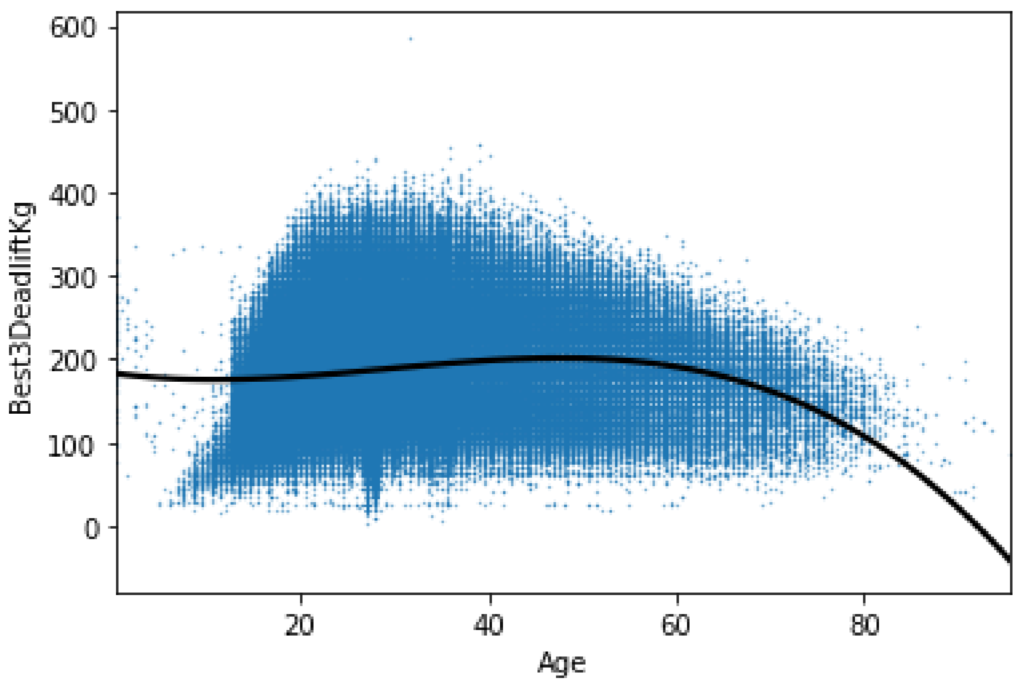
Most of these numerical features look normally distributed. Maybe a little right skew.

Distributions didn't seem to change much when accounting for Sex, except that the male Kgs on everything are shifted up by a lot and there are much more males in general.

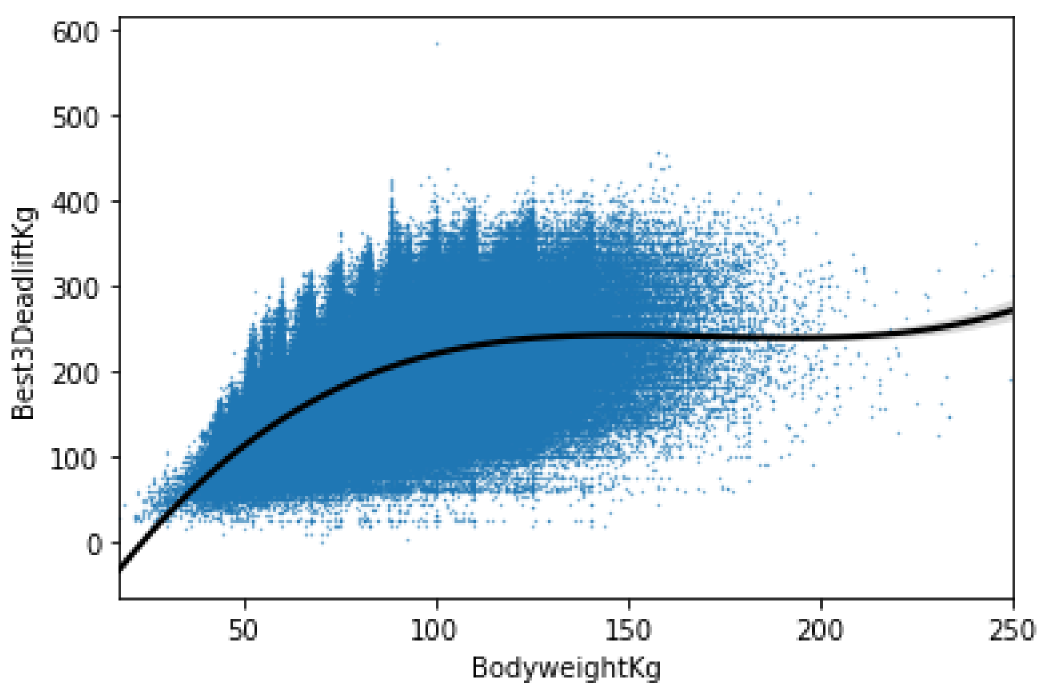
5.1.3 Line Plots

[176]:

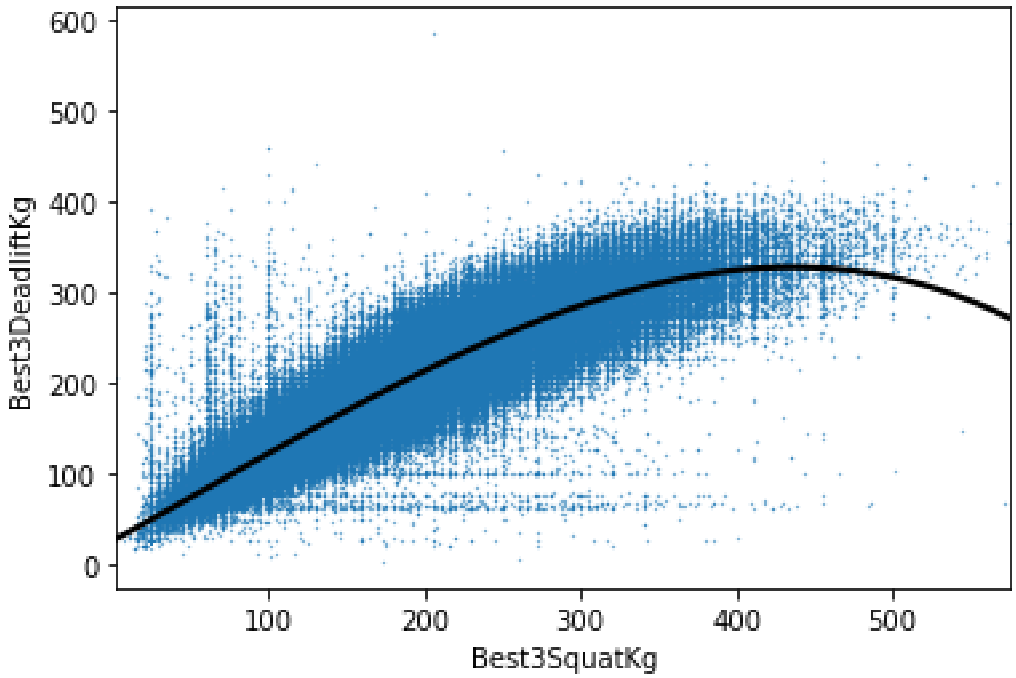
```
# Create scatter plots for each numerical value against the target

for cname in numerical:

    sns.regplot(x=X[cname], y=y, order=3, scatter_kws={'s': .1},␣
 ↪line_kws={'color': 'black'})

    plt.show()
```

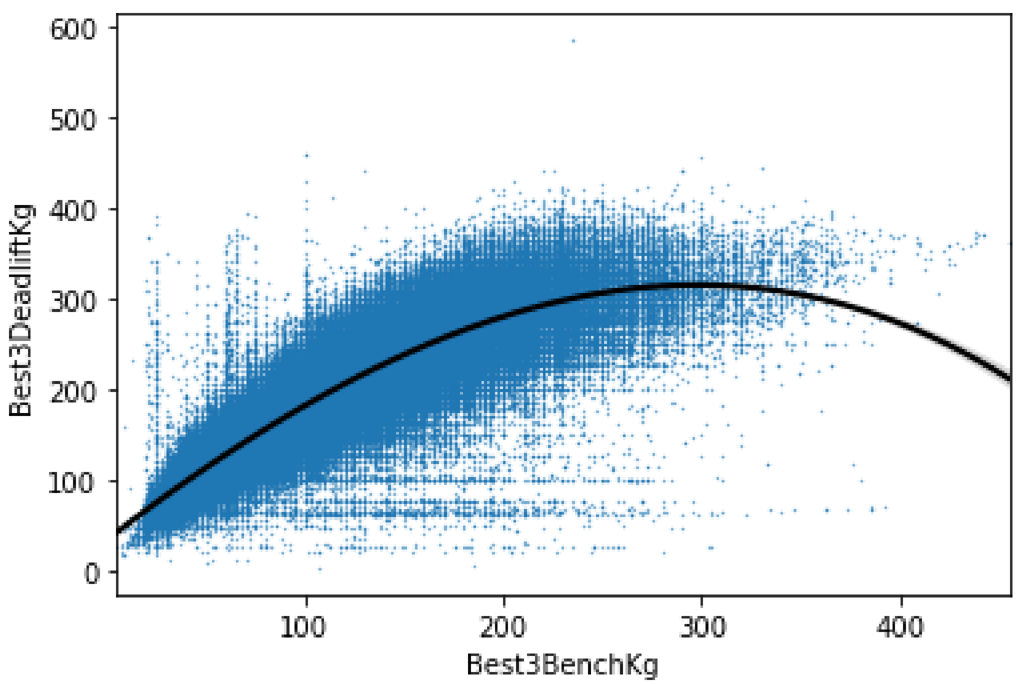


Best3DeadliftKg
Age



Best3DeadliftKg
BodyweightKg





Age: Doesn't seem to affect deadlift, unless they're 60+ years old.
Bodyweight: Deadlift increases logarithmically with body weight, but the very heaviest people seem to lift the most.

Squat: Deadlift increases linearly with squat, up until about 400kg squat.
Deadlift increases almost linearly with bench, up until about 300kg bench.

5.2 Categorical Features

[177]:
```
# How many unique values does each categorical feature have?

for cname in categorical:
    print(cname + ": " + str(X[cname].nunique()))
```

```
Name: 315731
Sex: 2
Equipment: 4
Division: 3685
Squat1Kg: 3
Squat2Kg: 3
Squat3Kg: 3
Bench1Kg: 3
Bench2Kg: 3
Bench3Kg: 3
Tested: 2
Country: 152
Federation: 200
MeetCountry: 93
MeetState: 110
```

Most of these have very very high cardinality.
One-Hot encode the ones with low cardinality.
Target encode the ones with high cardinality.

Take a look at Name quickly since it has so many unique values. Maybe some names are repeated?

[178]:
```
print(X["Name"].value_counts())
print()
print((X["Name"].value_counts() == 1).sum())
print((X["Name"].value_counts() != 1).sum())
print((X["Name"].value_counts() == 1).sum()/(X["Name"].value_counts() != 1).
 ↪sum())
```

```
Jose Hernandez        180
Jackie Blasbery       139
Karel Ruso            134
Jenny Hunter          127
Max Bristow           115
                     …
Hilde Selders           1
Austin Clark            1
B. Pipes                1
Trent Stilwell          1
```

```
Frederick Perry Jr      1
Name: Name, Length: 315731, dtype: int64

148126
167605
0.8837803168163241
```

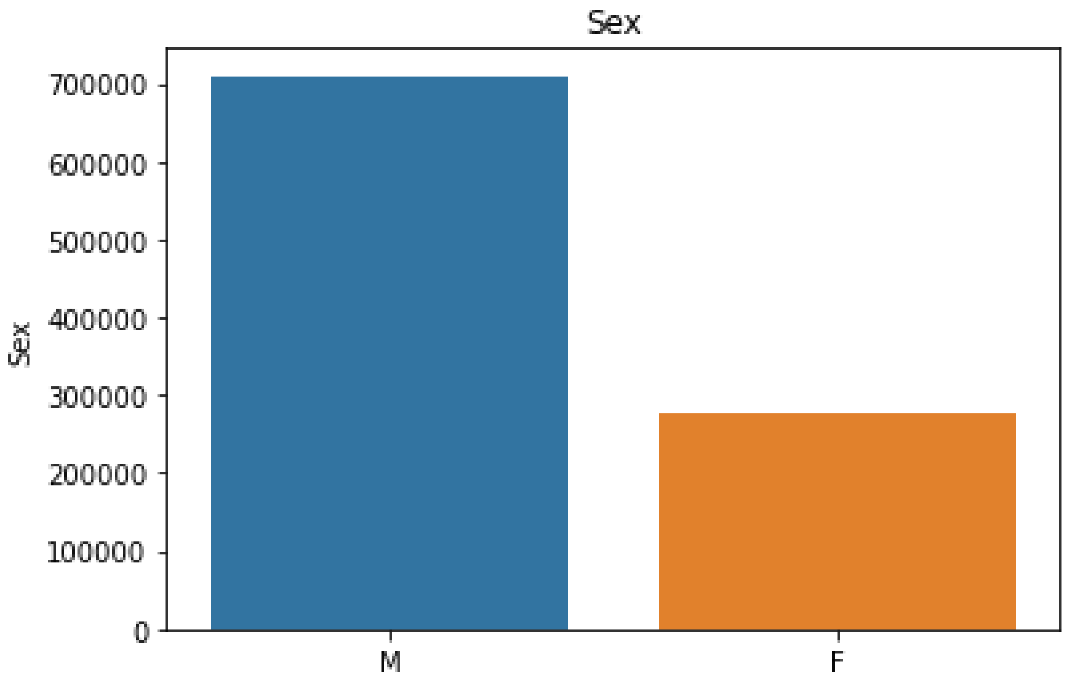
About 88% of people competed only once. In order to prevent target leakage, I should prevent any name from being in both the training and test sets.

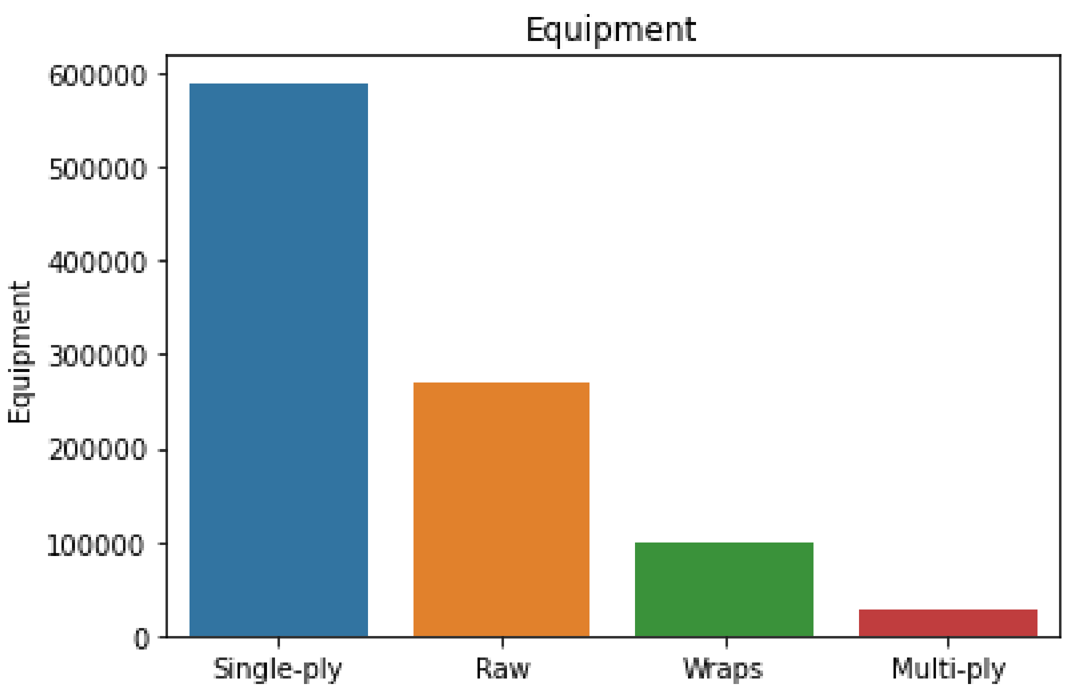
[179]:
```python
pd.set_option('display.max_columns', 50)
X.head()
```

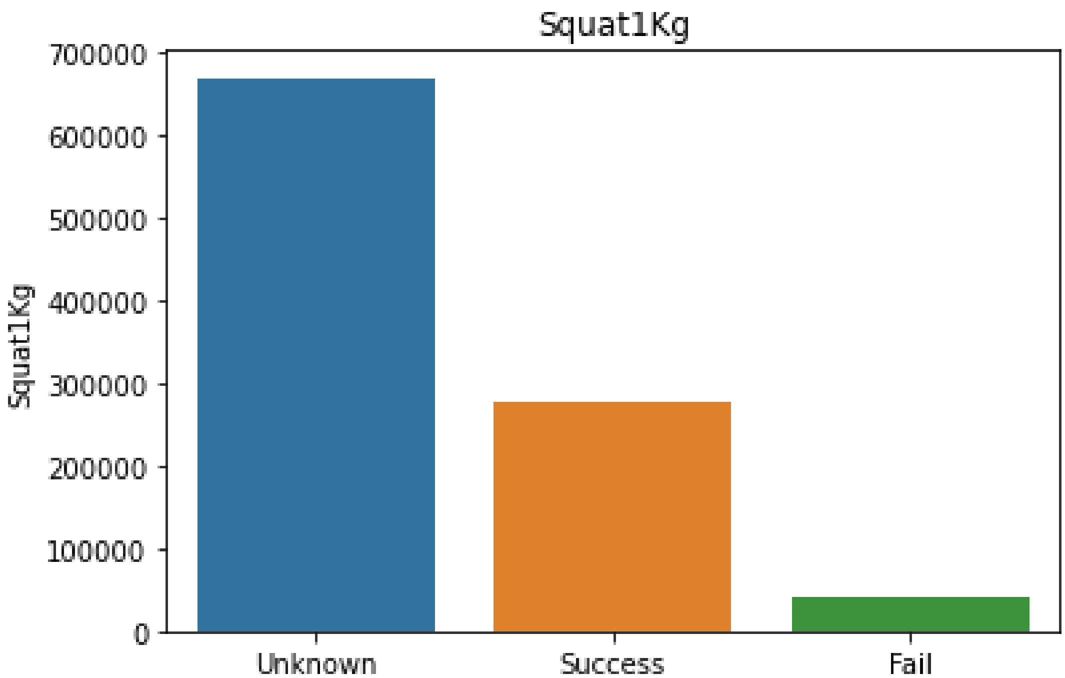
[179]:
```
   Unnamed: 0             Name Sex Equipment   Age Division  BodyweightKg  \
0           0     Abbie Murphy   F     Wraps  29.0     F-OR          59.8
1           1      Abbie Tuong   F     Wraps  29.0     F-OR          58.5
2           2  Amy Moldenhauer   F     Wraps  23.0     F-OR          60.0
3           3     Andrea Rowan   F     Wraps  45.0     F-OR         104.0
4           4    April Alvarez   F     Wraps  37.0     F-OR          74.0

  Squat1Kg Squat2Kg Squat3Kg  Best3SquatKg Bench1Kg Bench2Kg Bench3Kg  \
0  Success  Success  Success         105.0  Success  Success  Success
1  Success  Success  Success         120.0  Success  Success  Success
2     Fail     Fail  Success         105.0  Success  Success     Fail
3  Success  Success  Success         140.0  Success  Success  Success
4  Success  Success  Success         142.5  Success  Success  Success

   Best3BenchKg        Tested       Country Federation MeetCountry MeetState
0          55.0  Not Provided  Not Provided    GPC-AUS   Australia       VIC
1          67.5  Not Provided  Not Provided    GPC-AUS   Australia       VIC
2          72.5  Not Provided  Not Provided    GPC-AUS   Australia       VIC
3          80.0  Not Provided  Not Provided    GPC-AUS   Australia       VIC
4          82.5  Not Provided  Not Provided    GPC-AUS   Australia       VIC
```

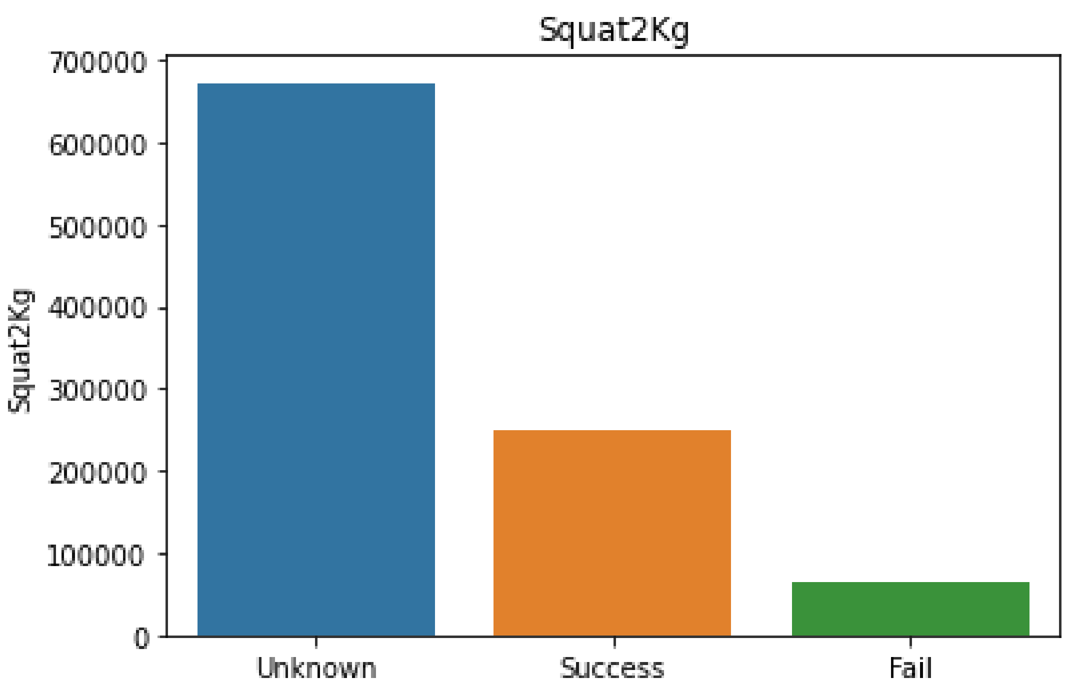
5.2.1 Barplots

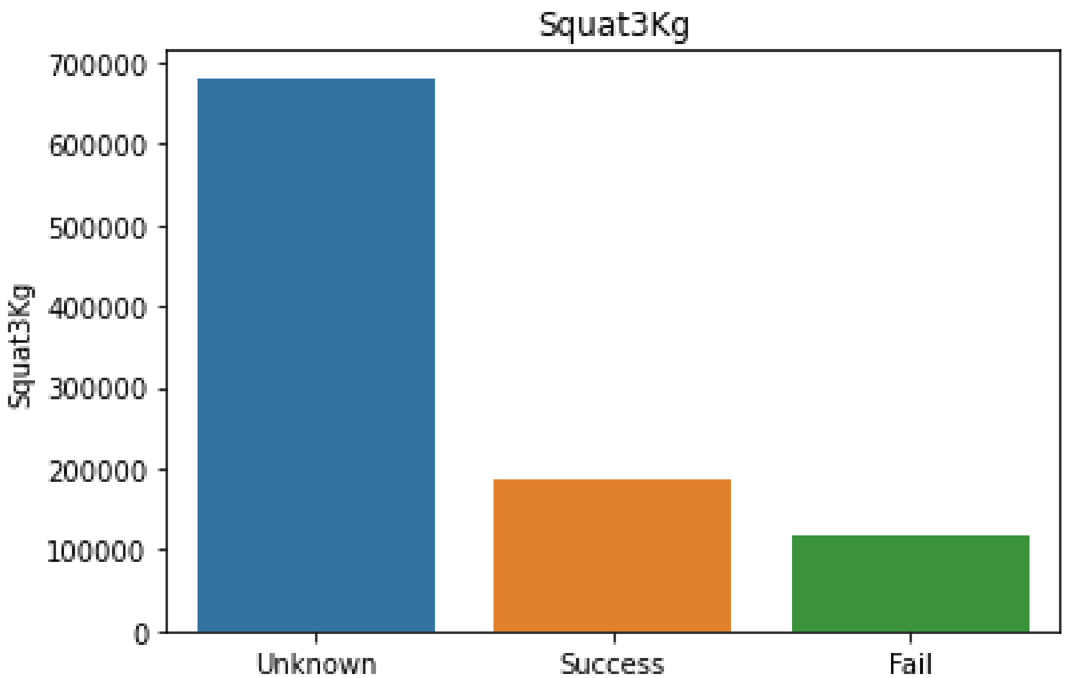
[180]:
```python
# Create barplots for each categorical feature to see their distributions

for cname in categorical:
    if X[cname].nunique() < 15:
        valueCounts = X[cname].value_counts()
        sns.barplot(valueCounts.index, valueCounts).set_title(cname)
        plt.show()
```

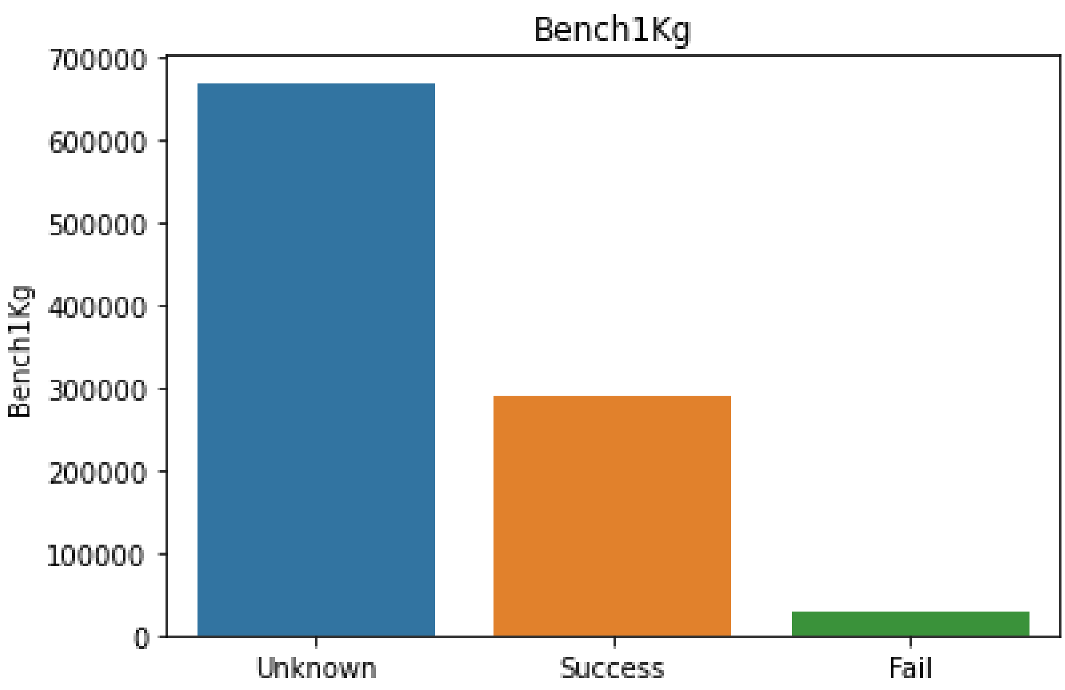

Sex
700000
600000
500000
400000
300000
200000
100000
0
Sex
M
F

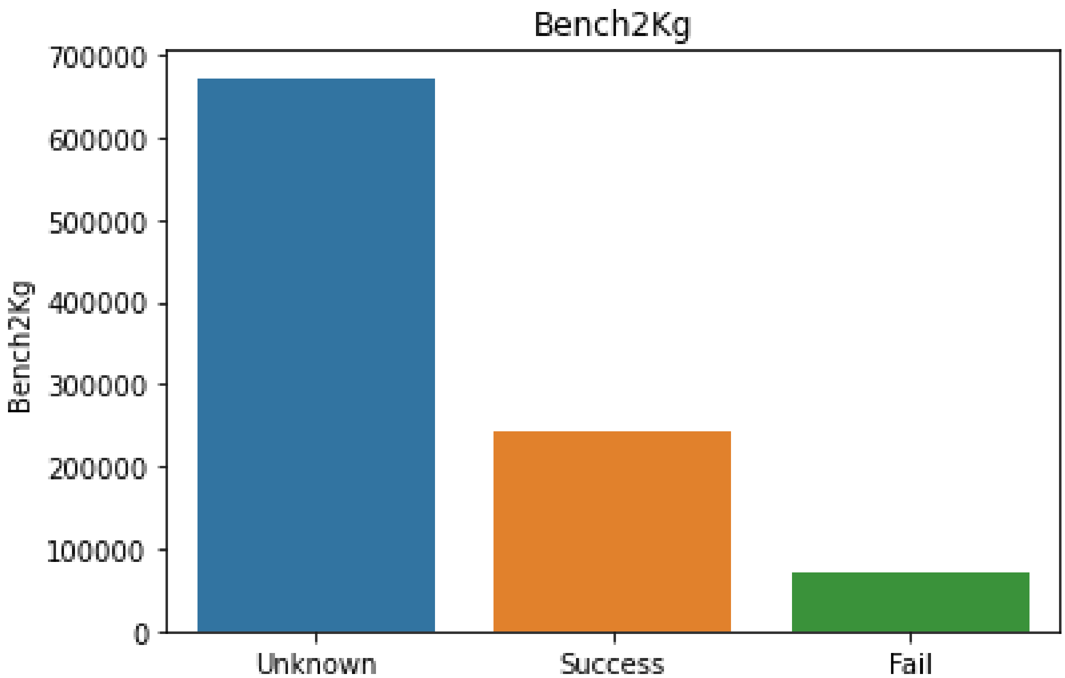

Equipment
600000
500000
400000
300000
200000
100000
0
Equipment
Single-ply
Raw
Wraps
Multi-ply

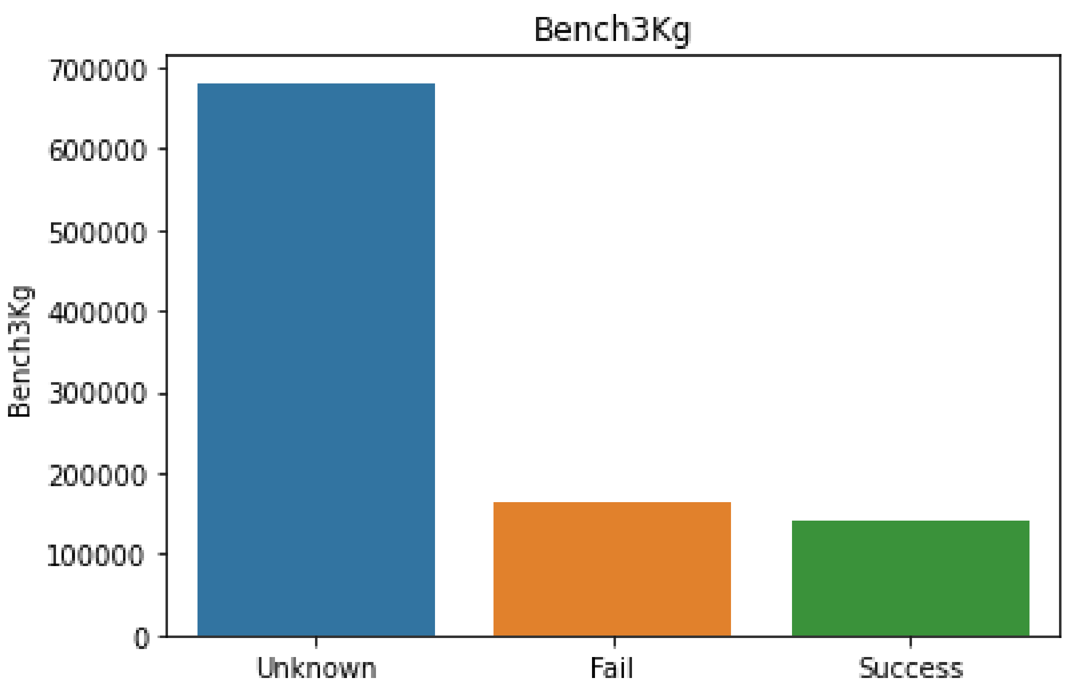

Squat1Kg
700000
600000
500000
400000
300000
200000
100000
0
Squat1Kg
Unknown
Success
Fail

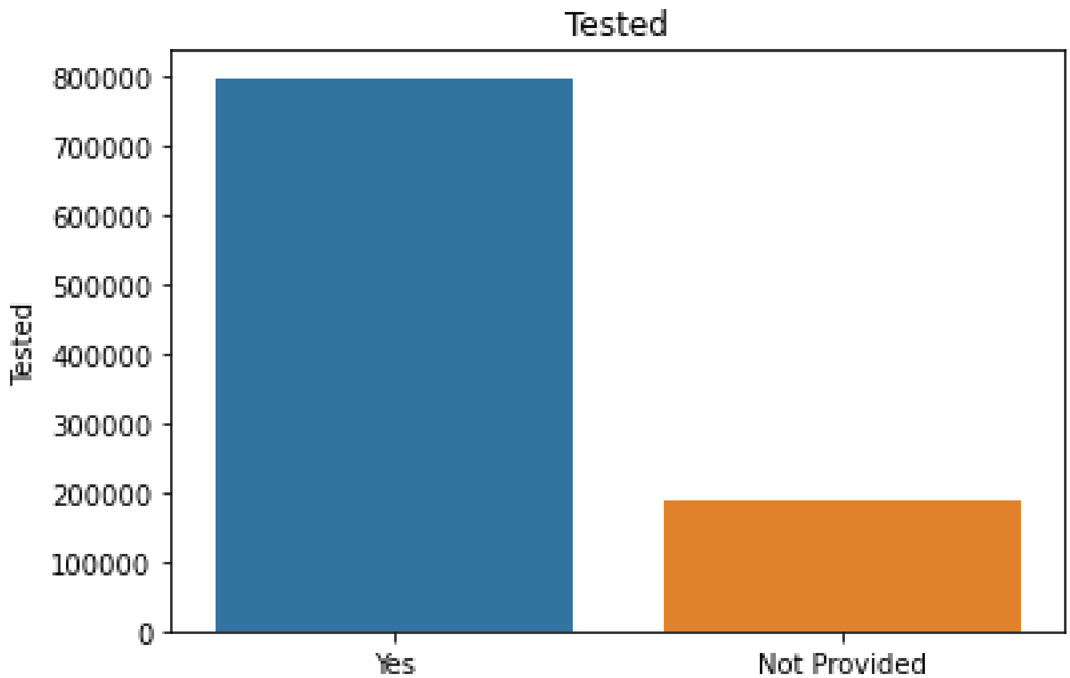

Squat2Kg
700000
600000
500000
400000
300000
200000
100000
0
Squat2Kg
Unknown
Success
Fail


Squat3Kg
Squat3Kg
700000
600000
500000
400000
300000
200000
100000
0
Unknown
Success
Fail


Bench1Kg
Bench1Kg
700000
600000
500000
400000
300000
200000
100000
0
Unknown
Success
Fail


Bench2Kg
700000
600000
500000
400000
300000
200000
100000
0
Bench2Kg
Unknown
Success
Fail

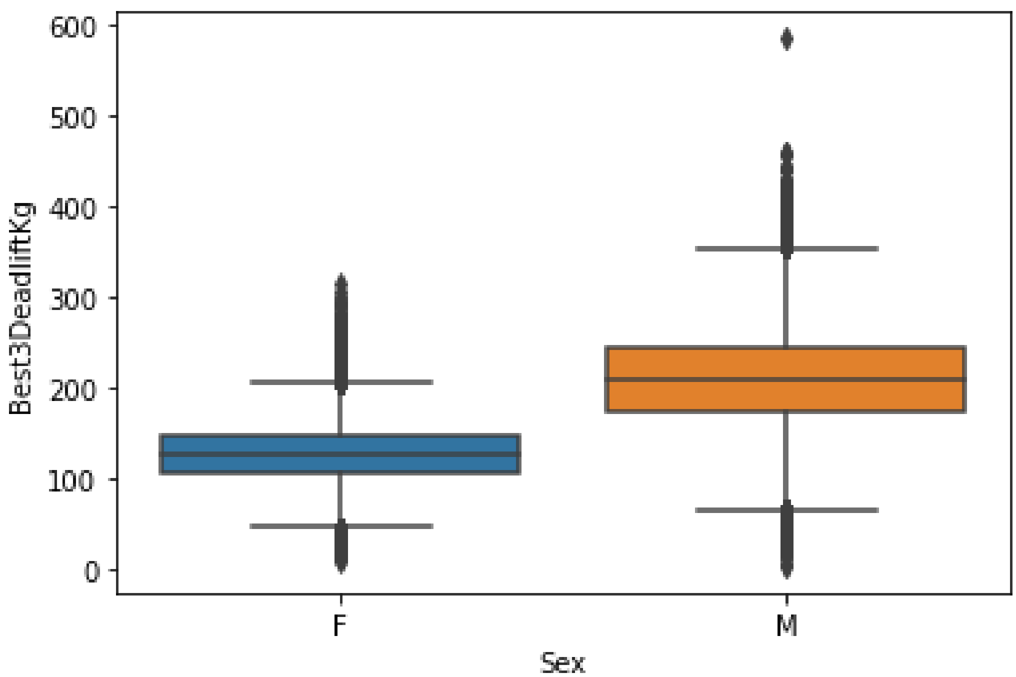

Bench3Kg
700000
600000
500000
400000
300000
200000
100000
0
Bench3Kg
Unknown
Fail
Success



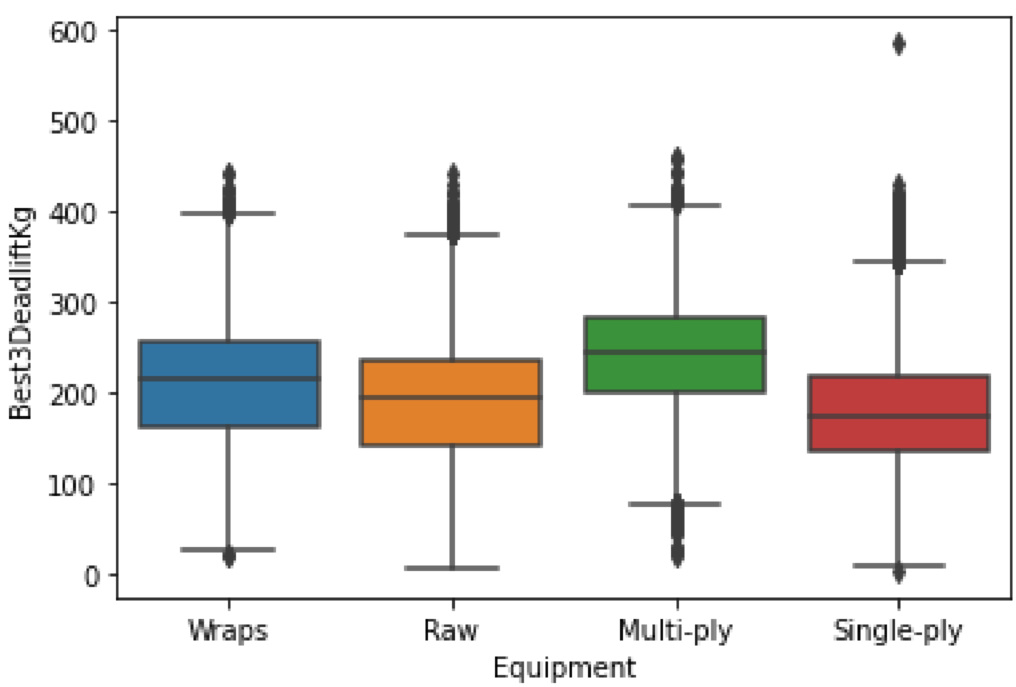
Sex: Males outnumber females by almost 3 to 1.
Equipment: Single ply is the most common, followed by Raw, then Wraps, then Multi-ply.
Attempt Features: Most attempts were not noted, but successful lifts happen more often than unsuccessful lifts.
Tested: Most people were tested.

5.2.2 Boxplots

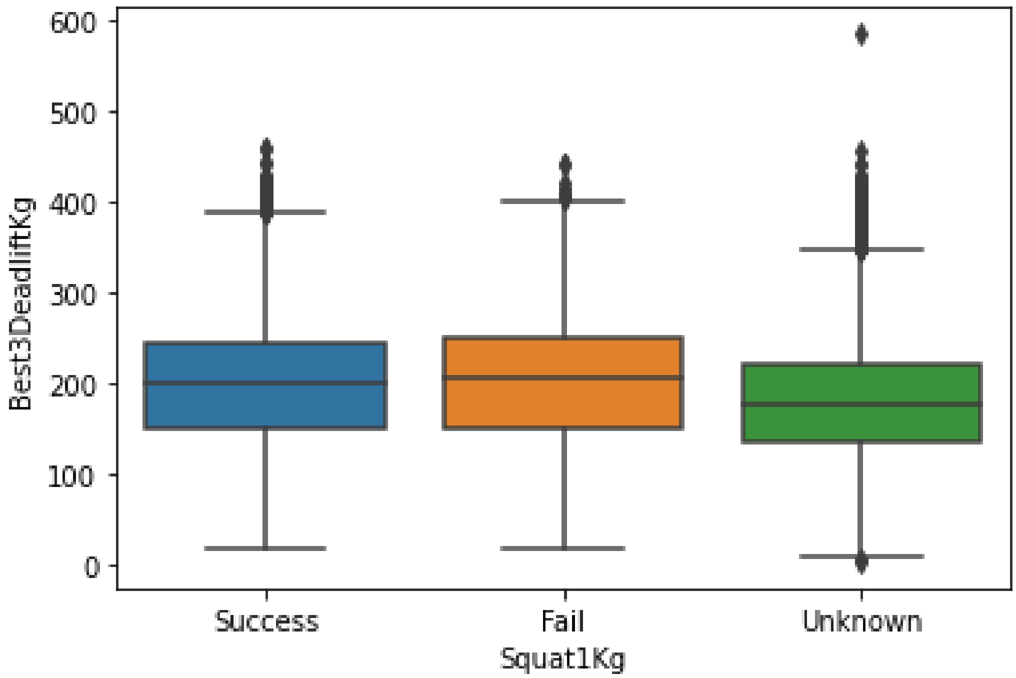
[181]:
```
# See the boxplots for categorical features against Best3DeadliftKg
# Don't make boxplots for high cardinality features because their plots are␣
↪hard to interpret

for cname in categorical:
    if X[cname].nunique() < 15:
        sns.boxplot(x=X[cname], y=y)
        plt.show()
```

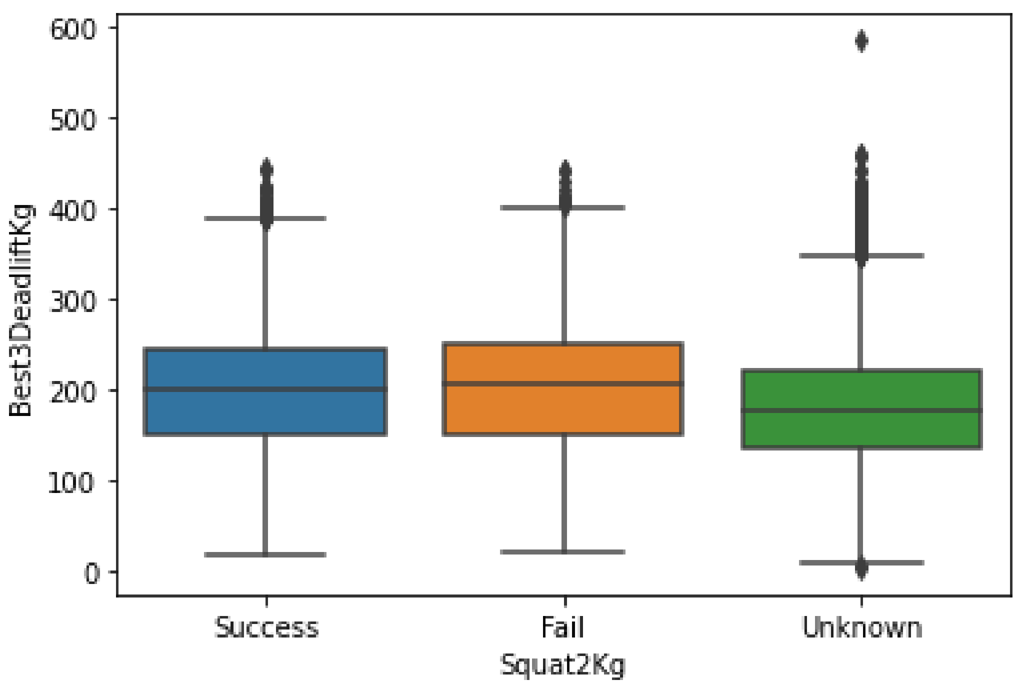


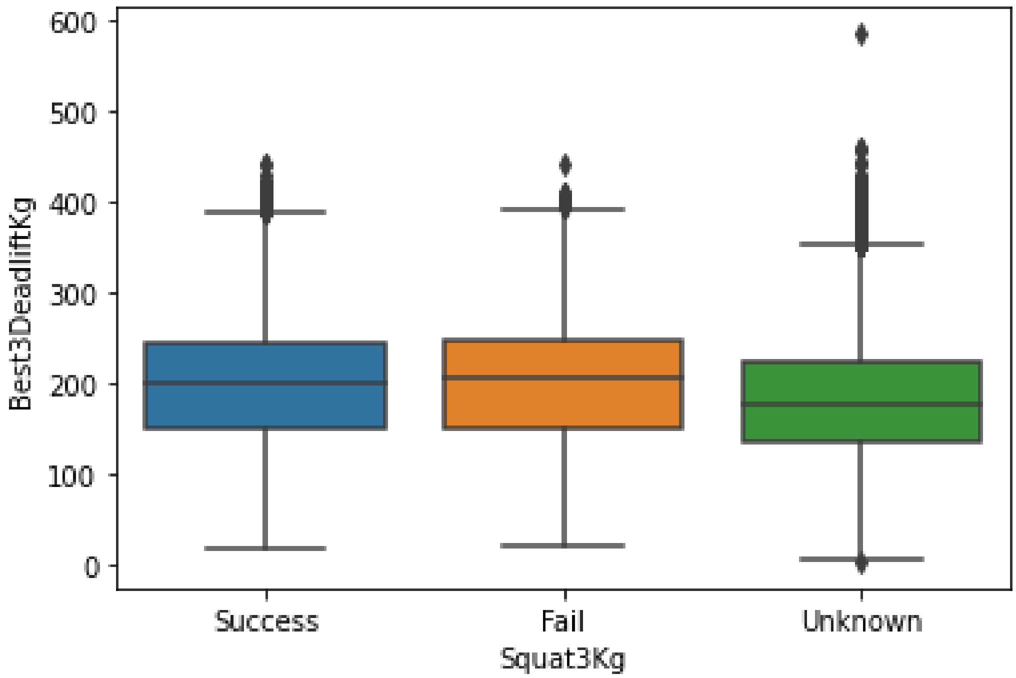
600
500
400
300
200
100
0
Best3DeadliftKg
F
M
Sex

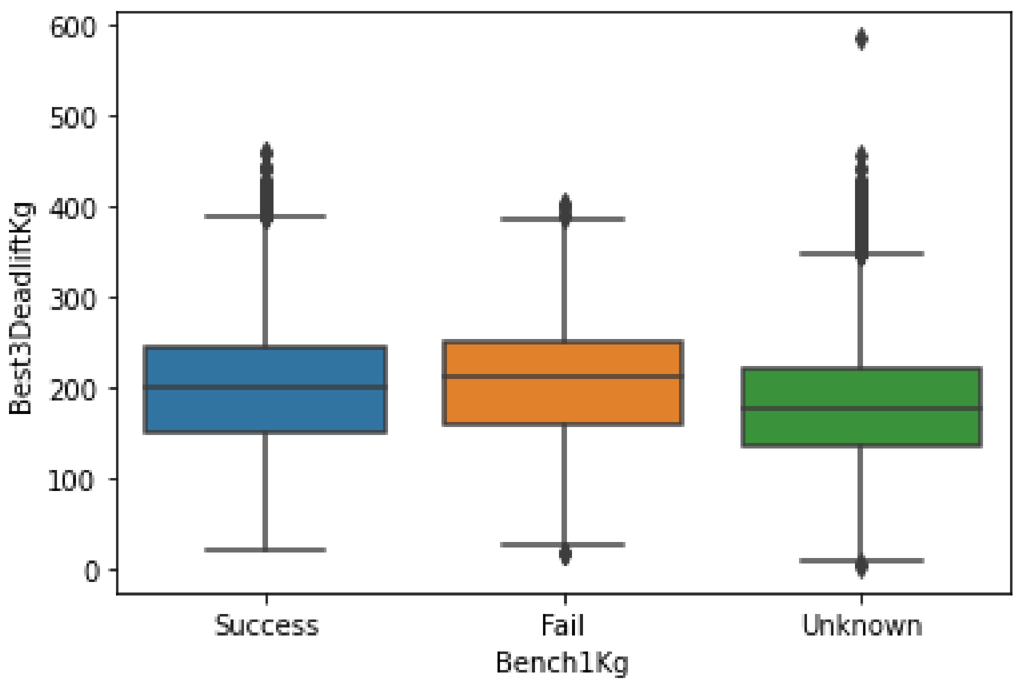


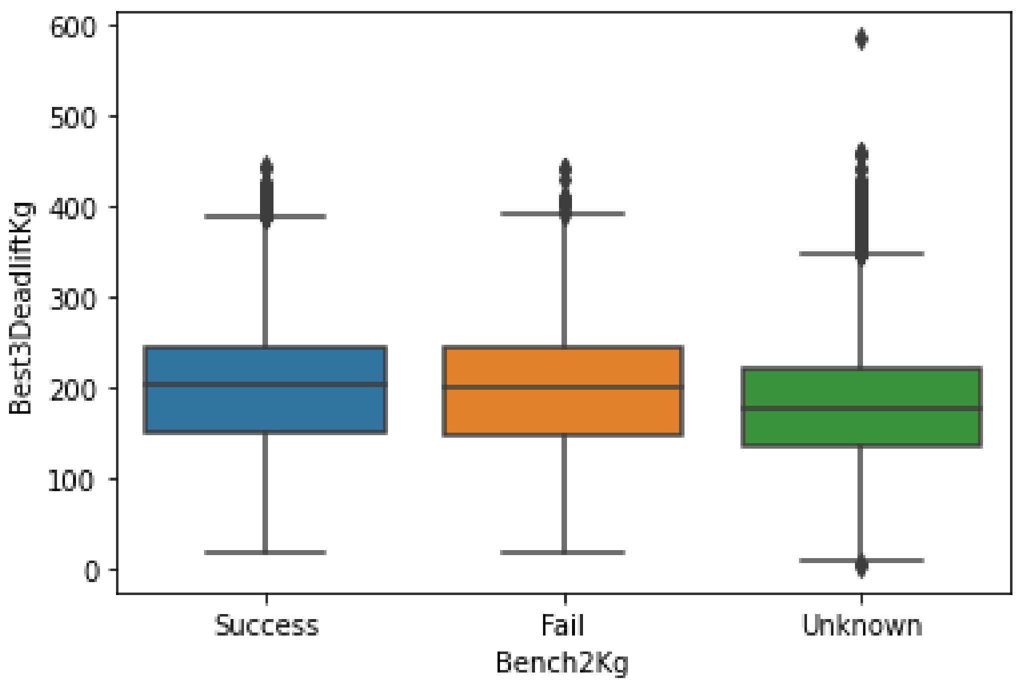
600
500
400
300
200
100
0
Best3DeadliftKg
Wraps
Raw
Multi-ply
Single-ply
Equipment

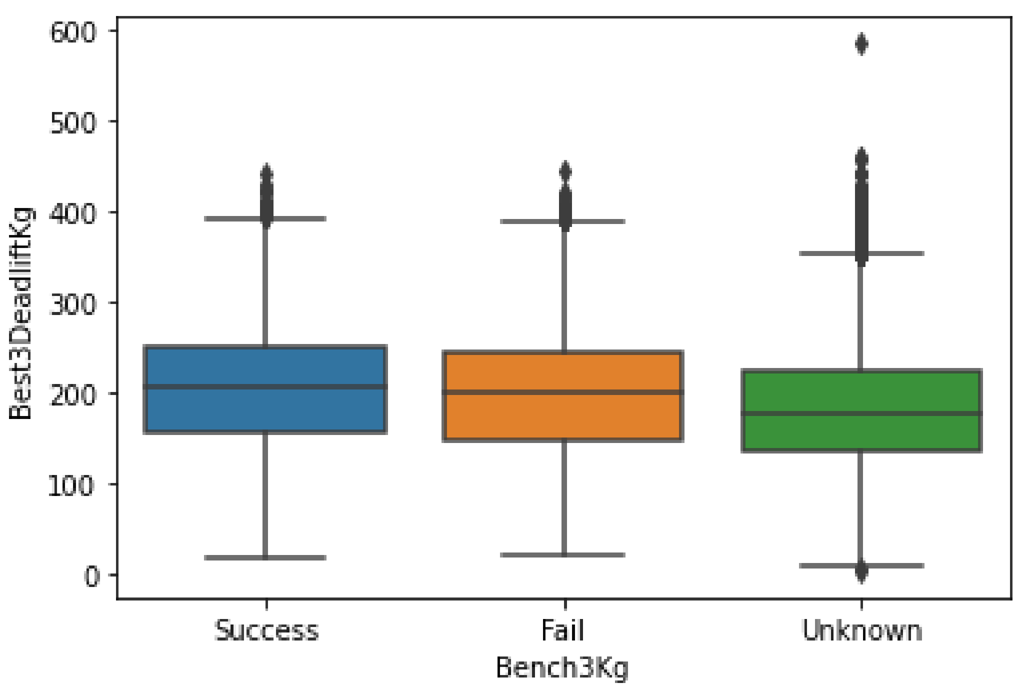

600
500
400
300
200
100
0
Best3DeadliftKg
Success
Fail
Unknown
Squat1Kg

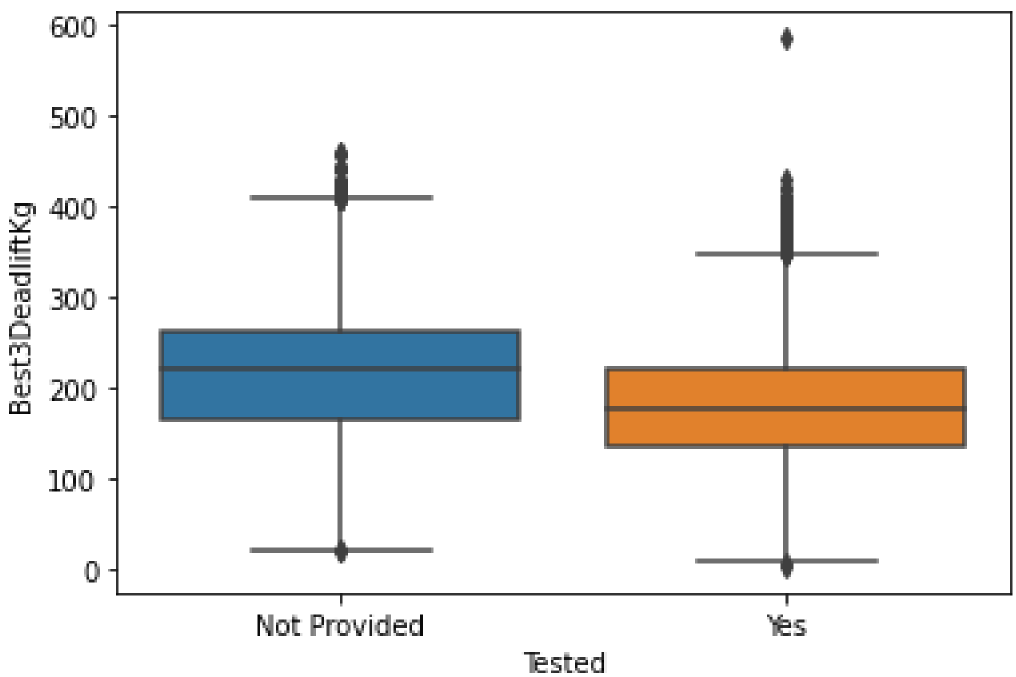

600
500
400
300
200
100
0
Best3DeadliftKg
Success
Fail
Unknown
Squat2Kg


600
500
400
300
200
100
0
Best3DeadliftKg
Success
Fail
Unknown
Squat3Kg


600
500
400
300
200
100
0
Best3DeadliftKg
Success
Fail
Unknown
Bench1Kg


600
500
400
300
200
100
0
Best3DeadliftKg
Success
Fail
Unknown
Bench2Kg


600
500
400
300
200
100
0
Best3DeadliftKg
Success
Fail
Unknown
Bench3Kg



Sex: Male deadlift median higher than female.
Equipment: Multi-ply deadlift median higher than other equipment, even though it's the least used.
Attempt Features: On average, people who fail their bench and squat end up lifting more for their deadlift. This probably means they push themself more?
Tested: Non-tested deadlift higher than tested.
All makes sense.

5.2.3 High Cardinality Features

[182]:
```
for cname in categorical:
    if X[cname].nunique() >= 15:
        print(X[cname].value_counts().head(), "\n")
```

```
Jose Hernandez     180
Jackie Blasbery    139
Karel Ruso         134
Jenny Hunter       127
Max Bristow        115
Name: Name, dtype: int64

Boys       241382
Open       220673
Girls      103546
Juniors     37948
```

```
MR-O        25335
Name: Division, dtype: int64

Not Provided    772149
USA              52596
Russia           17704
Australia        12565
Finland          11096
Name: Country, dtype: int64

THSPA     250067
THSWPA    104641
USAPL      97993
USPA       59061
CPU        28822
Name: Federation, dtype: int64

USA          653497
Canada        37831
Russia        34939
Australia     29916
Ukraine       22276
Name: MeetCountry, dtype: int64

TX              382614
Not Provided    260004
CA               31060
FL               20977
OH               18598
Name: MeetState, dtype: int64
```

Name: Jose Hernandez has meet to the most meets.
Division: Boys is the most populous division, followed by Open.
Country: The majority of lifters don't provide their country of origin.
Federation: THSPA and THSWPA are the most popular federations.
MeetCountry: The vast majority of meets happen in the US.
MeetState: Most meets happen in Texas, but a lot of the time it's not provided or it's in a country with no states.

5.2.4 Pivot Tables

[186]:
```
# Look at high cardinality categorical features vs. target feature
# See the top 5 and bottom 5 categorical values by median

for cname in categorical:
    if X[cname].nunique() >= 15:
```

```
        print(X.groupby([cname])["Best3DeadliftKg"].agg(func='median').
↪sort_values(ascending=False))
        print("\n")
```

```
Name
Andy Bolton          417.75
Ralph Atchinson      412.50
Chris Weist          410.00
Mikhail Koklyaev     408.75
John McMahon         400.00
                      …
Berkley Holmes        20.41
F. Walcakuwa          20.41
B. Hayashi            20.41
Chelsea Koceski       11.34
Bianca Roos           10.00
Name: Best3DeadliftKg, Length: 315731, dtype: float64


Division
Super Heavyweight     366.275
Open/Masters 40-44    365.000
Elite Pro Open        360.000
MM-2 RA               350.000
1974                  350.000
                        …
7-U                    40.820
FR-M6                  35.000
Ironman 8-9            34.020
Y 6-7                  30.000
7-8                    29.480
Name: Best3DeadliftKg, Length: 3685, dtype: float64


Country
Yugoslavia                  285.0
Bulgaria                    275.0
Ghana                       275.0
Swaziland                   271.0
Central African Republic    270.0
                            …
N.Ireland                   155.0
Syria                       152.5
Hong Kong                   142.5
Nepal                       140.0
Djibouti                    120.0
Name: Best3DeadliftKg, Length: 152, dtype: float64
```

```
Federation
SCT            360.00
SPSS           340.00
WPC-Germany    305.00
WPC-Iceland    295.00
XPC            274.42
                …
IDFPA          160.00
RAWU           158.76
USSports       156.49
MHSPLA         149.69
THSWPA         111.13
Name: Best3DeadliftKg, Length: 200, dtype: float64
```

```
MeetCountry
Algeria       242.50
Azerbaijan    240.00
Tahiti        237.50
Austria       232.50
Luxembourg    232.50
               …
Costa Rica    175.00
USA           172.37
Kyrgyzstan    170.00
Kazakhstan    170.00
Nicaragua     160.00
Name: Best3DeadliftKg, Length: 93, dtype: float64
```

```
MeetState
WB     270.00
TKI    232.50
MOW    230.00
OH     227.50
TN     226.80
        …
RP     168.75
WI     165.00
DL     158.75
TX     156.49
STL    152.50
Name: Best3DeadliftKg, Length: 110, dtype: float64
```

Name: Andy Bolton has the highest deadlift on median at about 420kg. Some girl got 10kg.

Division: Highest are Super Heavyweight and Open divisions. Lowest are Ages 7-8 and Youth 6-7.
Country: Highest lifters are from Syria, Yugoslavia, and Ghana. Lowest are from Hong Kong, Nepal, and Djibouti.
Federation: Highest are SCT and SPSS. Lowest are MHSPLA and THSWPA. (no clue what any of those are).
MeetCountry: Highest lifts are in Algeria, Azerbaijan, and Tahiti. Lowest are in Kazakhstan and Nicaragua.
MeetState: Highest lifts are from WB and TKI. Lowest are from TX and STL. (no clue what any of those are).

---

6 *Feature Engineering*

6.1 Creating New Features

[207]:
```
X.columns
```

[207]:
```
Index(['Name', 'Sex', 'Equipment', 'Age', 'Division', 'BodyweightKg',
       'Squat1Kg', 'Squat2Kg', 'Squat3Kg', 'Best3SquatKg', 'Bench1Kg',
       'Bench2Kg', 'Bench3Kg', 'Best3BenchKg', 'Tested', 'Country',
       'Federation', 'MeetCountry', 'MeetState'],
      dtype='object')
```

I don't have much of a good idea of how to create new good features here, but I do want to limit the number of features total.
Going to create a "score" feature in place of the attempt features to remove all 6 of them.

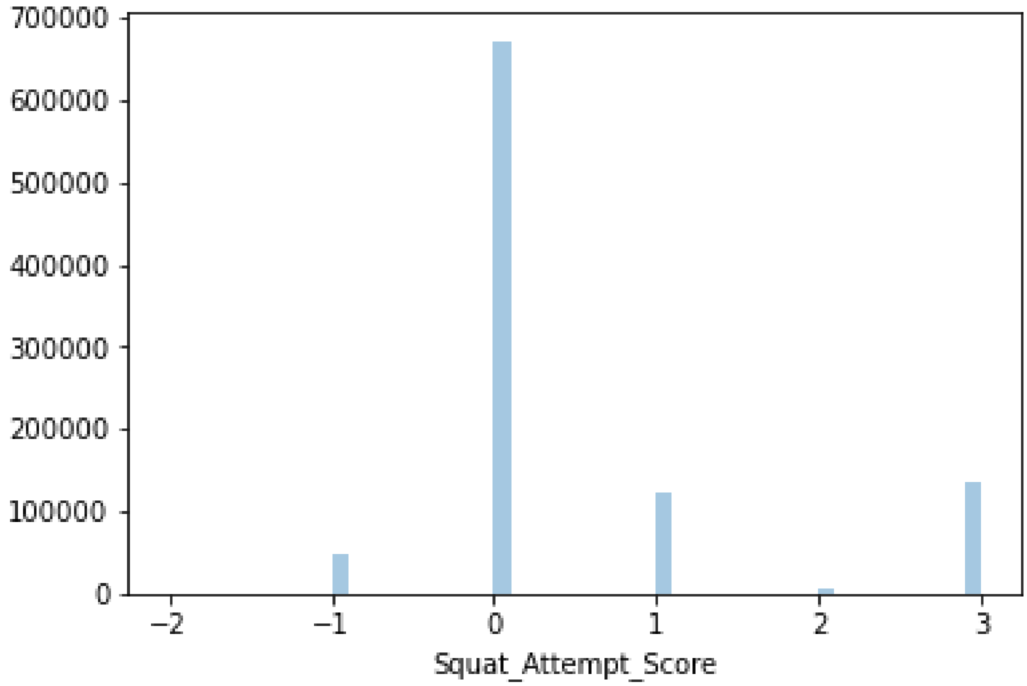
[226]:
```
# re do these features to turn them into numbers via a "score"
def attemptScoreTransformer(X):
    scoreDict = {"Unknown": 0, "Success": 1, "Fail": -1}

    for col in X.columns:
        X[col] = X[col].apply(lambda x: scoreDict[x])

    return X.sum(axis=1)


X["Squat_Attempt_Score"] = attemptScoreTransformer(X[['Squat1Kg', 'Squat2Kg',␣
↪'Squat3Kg']])
X["Bench_Attempt_Score"] = attemptScoreTransformer(X[['Bench1Kg', 'Bench2Kg',␣
↪'Bench3Kg']])
X["Total_Attempt_Score"] = X["Squat_Attempt_Score"] + X["Bench_Attempt_Score"]

print(X["Squat_Attempt_Score"].unique())
print(X["Bench_Attempt_Score"].unique())
print(X["Total_Attempt_Score"].unique())
```

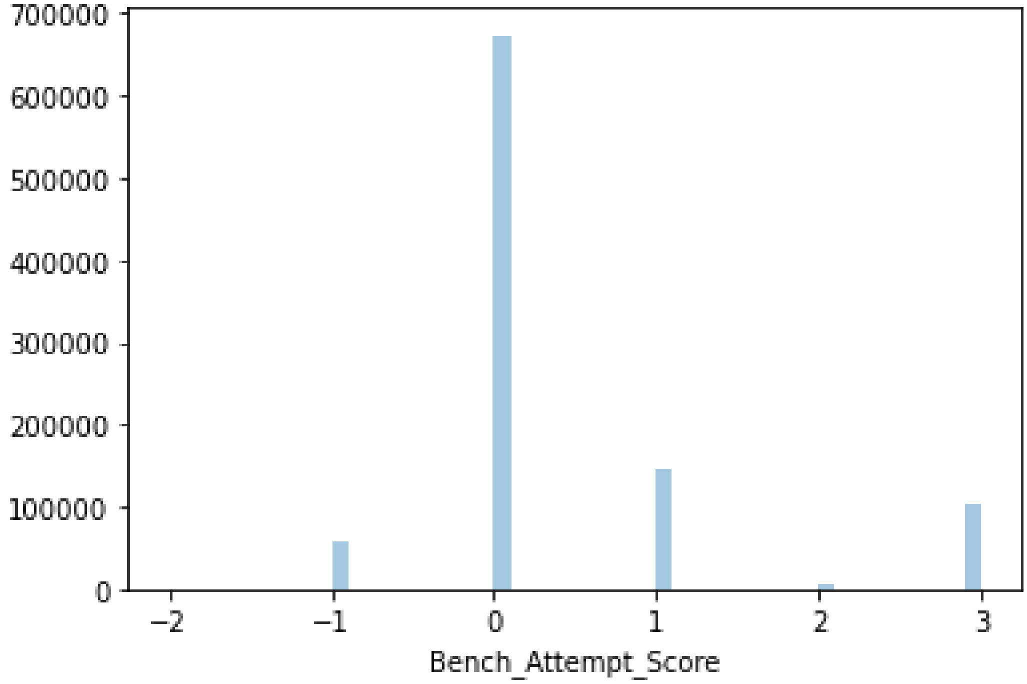
```
[ 3 -1  1  2  0 -2]
```

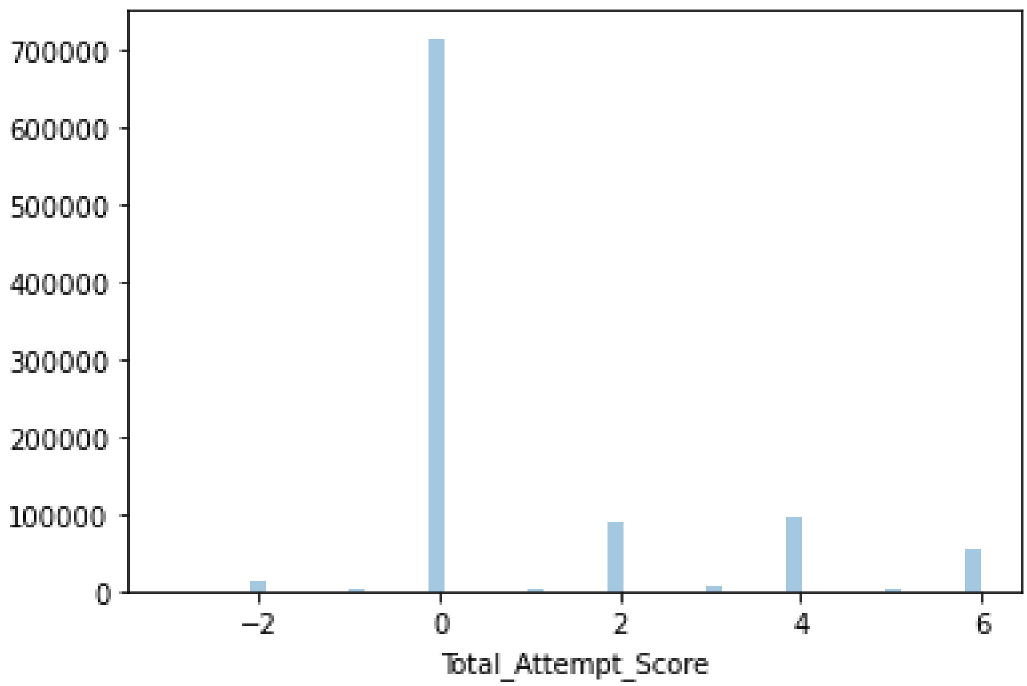
```
[ 3  1  2 -1  0 -2]
[ 6  0  2  5  4  3 -2 -1  1 -3]
```

It makes sense that there are no -3 values in the Squat and Bench Score. I removed all samples that failed all 3 lifts.

[235]:
```python
# see distributions
for col in ["Squat_Attempt_Score", "Bench_Attempt_Score",␣
 ↪"Total_Attempt_Score"]:
    sns.distplot(X[col], kde=False)
    plt.show()
```







Most of the time these features are 0, but that's because most events don't mark down the attempts and whether they're successful or not.

```
[ ]: # can now drop the attempt cols
X = X.drop(attemptCols, axis=1)
X.columns
```

I probably should include interaction features, but it will increase multicollinearity, so I won't.

```
[191]: # Creating Interaction Features

# X["AgePerWeight"] = X["Age"]/X["BodyweightKg"]
# X["BestSquatBench"] = X["Best3SquatKg"]*X["Best3BenchKg"]
# X["AgeWeightBenchSquat"] = X["AgePerWeight"]*X["BestSquatBench"]
```

```
[237]: # Re do these in case they've gotten messed up so far
categorical = [cname for cname in X.columns.tolist() if X[cname].dtype ==␣
 ↪"object"]
numerical = [cname for cname in X.columns.tolist() if X[cname].dtype ==␣
 ↪"float64"]
```

6.2 Categorical Encoding

```
[238]: X[categorical].head()
```

```
[238]:              Name Sex Equipment Division        Tested       Country  \
0     Abbie Murphy   F     Wraps     F-OR  Not Provided  Not Provided
1      Abbie Tuong   F     Wraps     F-OR  Not Provided  Not Provided
2  Amy Moldenhauer   F     Wraps     F-OR  Not Provided  Not Provided
3     Andrea Rowan   F     Wraps     F-OR  Not Provided  Not Provided
4    April Alvarez   F     Wraps     F-OR  Not Provided  Not Provided

  Federation MeetCountry MeetState
0    GPC-AUS   Australia       VIC
1    GPC-AUS   Australia       VIC
2    GPC-AUS   Australia       VIC
3    GPC-AUS   Australia       VIC
4    GPC-AUS   Australia       VIC
```

6.2.1 OneHot Encoding

```
[240]: # One-Hot Encode columns with low cardinality

toOHEnc = ["Sex", "Equipment", "Tested"]
from sklearn.preprocessing import OneHotEncoder
```

6.2.2 Target Encoding

[245]:
```
# Target Encode columns with high cardinality
# If the cols parameter isn't passed, every non-numeric column will be converted

toTargetEnc = ["Division", "Country", "Federation", "MeetCountry", "MeetState"]
from category_encoders import TargetEncoder
```

NOTE: Have to do these encodings on a specific training split to avoid target leakage.

---

7 *Data Cleaning Part 2*

7.1 Normalization and Outliers

The numerical columns all looked relatively normal so I won't mess with log transforms or boxcox transforms.
I didn't notice any huge outliers in any of the features.

7.2 Scaling

[249]:
```
# Scale the numerical data for the models that require it
# (like KNearestNeighbors)
from sklearn.preprocessing import StandardScaler
```

NOTE: Need to do this within the context of each cross validation fold. The parameters need to be accustomed to the training set and then applied to the test set.

---

8 *Data Preprocessing Pipeline*

8.1 Clean Rows

[9]:
```
def drop_rows_func(df):
    df = df.loc[(df["Best3DeadliftKg"].isna() == False) &
                (df["Best3DeadliftKg"] > 0) &
                (df["Best3SquatKg"].isna() == False) &
                (df["Best3SquatKg"] > 0) &
                (df["Best3BenchKg"].isna() == False) &
                (df["Best3BenchKg"] > 0) &
                (df["Event"] == "SBD") &
                (df["Age"] != 0.0)]
    # Reindex
    df.index = np.arange(df.shape[0])

    X, y = df.drop("Best3DeadliftKg", axis=1), df["Best3DeadliftKg"]
```

```
    return (X, y)
```

8.2 Custom Transformer

[11]:
```
from sklearn.base import TransformerMixin, BaseEstimator

class AttemptTransformer(TransformerMixin, BaseEstimator):
    # constructor
    def __init__(self):
        pass

    # helper function
    def _attemptScoreAdder(self, X):
        def _attemptImputer(x):
            import math
            if math.isnan(x):
                return 0
            elif x <= 0:
                return -1
            else:
                return 1

        for col in X.columns:
            X.loc[:,col] = X.loc[:,col].apply(lambda x: attemptImputer(x))

        return X.sum(axis=1)

    # fit
    def fit(self, X, y=None):
        return self

    # transform
    def transform(self, X, y=None):
        X["Squat_Attempt_Score"] = self._attemptScoreAdder(X[["Squat1Kg",␣
 ↪"Squat2Kg", "Squat3Kg"]])
        X["Bench_Attempt_Score"] = self._attemptScoreAdder(X[["Bench1Kg",␣
 ↪"Bench2Kg", "Bench3Kg"]])
        X["Total_Attempt_Score"] = X["Squat_Attempt_Score"] +␣
 ↪X["Bench_Attempt_Score"]

        return X.drop(["Squat1Kg", "Squat2Kg", "Squat3Kg", "Bench1Kg",␣
 ↪"Bench2Kg", "Bench3Kg"], axis=1)
```

[13]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler,␣
 ↪FunctionTransformer
```

```
from category_encoders import TargetEncoder
from sklearn.impute import SimpleImputer

 # explicitly require this experimental feature
from sklearn.experimental import enable_iterative_imputer  # noqa
# now you can import normally from sklearn.impute
from sklearn.impute import IterativeImputer
```

8.3 Column Transformer

[14]:
```
colTransformer = ColumnTransformer(
    transformers=[
        ("TargetEncoder", TargetEncoder(verbose=0,
                                        drop_invariant=True,
                                        handle_unknown='value',
                                        handle_missing='value'),
                                        ["Country", "Division", "MeetState",␣
 ↪"Federation", "MeetCountry"]), # returns 5 cols
        ("OneHotEncoder", OneHotEncoder(categories='auto',
                                        handle_unknown='ignore',
                                        sparse=False),
                                        ["Tested", "Sex", "Equipment"]), #␣
 ↪returns 8 cols
        ("IterativeImputer", IterativeImputer(missing_values=np.nan,
                                            max_iter=10),
                                            ["Age", "BodyweightKg"]), # returns␣
 ↪2 cols
        ("passthrough", "passthrough", ["Best3BenchKg", "Best3SquatKg"]) #␣
 ↪returns 2 cols
    ],
    remainder='drop'
)
```

[15]:
```
import pandas as pd
# Get fresh data
fresh_data = pd.read_csv("openpowerlifting.csv")
```

8.4 Cross Validation Splits

[16]:
```
from sklearn.model_selection import GroupShuffleSplit
# Source; https://www.kaggle.com/dansbecker/underfitting-and-overfitting
# We'll do a "grouped" split to keep all of an artist's songs in one
# split or the other. This is to help prevent signal leakage.
def group_split(X, y, group, train_size):
    splitter = GroupShuffleSplit(train_size=train_size)
    train_indices, test_indices = next(splitter.split(X, y, groups=group))
    return (train_indices, test_indices)
```

[17]:
```
X, y = drop_rows_func(fresh_data)
train_indices, test_indices = group_split(X, y, X["Name"], train_size=4/5)
Xtrain, ytrain, Xtest, ytest = X.iloc[train_indices], y.iloc[train_indices], X.
↪iloc[test_indices], y.iloc[test_indices]
```

[18]:
```
# test column transformer
output = colTransformer.fit_transform(Xtrain, ytrain)
np.isnan(np.sum(output))
```

[18]: False

[19]:
```
cv_indices_list = [group_split(Xtrain, ytrain, Xtrain["Name"], train_size=4/5)␣
↪for i in range(5)]
```

---

9 *Model Training*

IMPORTANT: Going to try base hyperparameters first (empty param grid). Grid search takes too long because I have so much data.

[20]:
```
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
```

9.1 Linear Regression

[496]:
```
from sklearn.linear_model import LinearRegression

linear_regression_pipeline = Pipeline(
    steps=[
        ('columns', colTransformer),
        ('scale', StandardScaler()), # scales every column
        ('model cv', GridSearchCV(estimator=LinearRegression(),
                                param_grid={},
                                scoring="neg_mean_absolute_error",
                                cv = cv_indices_list))
    ],
    verbose=True
)

linear_regression_pipeline.fit(Xtrain, ytrain)
```

```
[Pipeline] … (step 1 of 3) Processing columns, total=   7.0s
[Pipeline] … (step 2 of 3) Processing scale, total=   0.7s
[Pipeline] … (step 3 of 3) Processing model cv, total=   5.3s
```

[496]:
```
Pipeline(steps=[('columns',
                 ColumnTransformer(transformers=[('TargetEncoder',
TargetEncoder(drop_invariant=True),
                                                  ['Country', 'Division',
                                                   'MeetState', 'Federation',
                                                   'MeetCountry']),
                                                 ('OneHotEncoder',
OneHotEncoder(handle_unknown='ignore',
                                                               sparse=False),
                                                  ['Tested', 'Sex',
                                                   'Equipment']),
                                                 ('IterativeImputer',
                                                  IterativeImputer(),
                                                  ['Age', 'BodyweightKg']),
                                                 ('passthrough'…
                                   array([     2,      8,     13, …, 790116,
790126, 790127], dtype=int64)),
                                  (array([     0,      1,      2, …, 790123,
790124, 790125], dtype=int64),
                                   array([     4,      7,     12, …, 790121,
790126, 790127], dtype=int64)),
                                  (array([     0,      1,      2, …, 790125,
790126, 790127], dtype=int64),
                                   array([    12,     29,     30, …, 790100,
790105, 790123], dtype=int64))],
                              estimator=LinearRegression(), param_grid={},
                              scoring='neg_mean_absolute_error'))],
         verbose=True)
```

[497]:
```
# Mean cross-validated score of the best_estimator
linear_regression_pipeline['model cv'].best_score_
```

[497]: -15.162985514644623

9.2 Ridge Regression

[501]:
```
from sklearn.linear_model import Ridge

ridge_pipeline = Pipeline(
    steps=[
        ('columns', colTransformer),
        ('scale', StandardScaler()), # scales every column
        ('model cv', GridSearchCV(estimator=Ridge(),
                                  param_grid={},
                                  scoring="neg_mean_absolute_error",
                                  cv = cv_indices_list))
    ],
```

```
    verbose=True
)

ridge_pipeline.fit(Xtrain, ytrain)
```

```
[Pipeline] … (step 1 of 3) Processing columns, total=   6.6s
[Pipeline] … (step 2 of 3) Processing scale, total=   0.6s
[Pipeline] … (step 3 of 3) Processing model cv, total=   1.7s
```

[501]:
```
Pipeline(steps=[('columns',
                 ColumnTransformer(transformers=[('TargetEncoder',
TargetEncoder(drop_invariant=True),
                                                  ['Country', 'Division',
                                                   'MeetState', 'Federation',
                                                   'MeetCountry']),
                                                 ('OneHotEncoder',
OneHotEncoder(handle_unknown='ignore',
                                                               sparse=False),
                                                  ['Tested', 'Sex',
                                                   'Equipment']),
                                                 ('IterativeImputer',
                                                  IterativeImputer(),
                                                  ['Age', 'BodyweightKg']),
                                                 ('passthrough'…
                                    array([     2,      8,     13, …, 790116,
790126, 790127], dtype=int64)),
                                   (array([     0,      1,      2, …, 790123,
790124, 790125], dtype=int64),
                                    array([     4,      7,     12, …, 790121,
790126, 790127], dtype=int64)),
                                   (array([     0,      1,      2, …, 790125,
790126, 790127], dtype=int64),
                                    array([    12,     29,     30, …, 790100,
790105, 790123], dtype=int64))],
                              estimator=Ridge(), param_grid={},
                              scoring='neg_mean_absolute_error'))],
         verbose=True)
```

[502]:
```
# Mean cross-validated score of the best_estimator
ridge_pipeline['model cv'].best_score_
```

[502]: -15.162984629989595

9.3 Decision Tree Regressor

[503]:
```
from sklearn.tree import DecisionTreeRegressor

decision_tree_regressor_pipeline = Pipeline(
    steps=[
        ('columns', colTransformer),
        ('model cv', GridSearchCV(estimator=DecisionTreeRegressor(),
                                param_grid={},
                                scoring="neg_mean_absolute_error",
                                cv = cv_indices_list))
    ],
    verbose=True
)

decision_tree_regressor_pipeline.fit(Xtrain, ytrain)
```

```
[Pipeline] … (step 1 of 2) Processing columns, total=   6.4s
[Pipeline] … (step 2 of 2) Processing model cv, total= 1.2min
```

[503]:
```
Pipeline(steps=[('columns',
                 ColumnTransformer(transformers=[('TargetEncoder',
TargetEncoder(drop_invariant=True),
                                                  ['Country', 'Division',
                                                   'MeetState', 'Federation',
                                                   'MeetCountry']),
                                                 ('OneHotEncoder',
OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False),
                                                  ['Tested', 'Sex',
                                                   'Equipment']),
                                                 ('IterativeImputer',
                                                  IterativeImputer(),
                                                  ['Age', 'BodyweightKg']),
                                                 ('passthrough'…
                                     array([     2,      8,     13, …, 790116,
790126, 790127], dtype=int64)),
                                    (array([     0,      1,      2, …, 790123,
790124, 790125], dtype=int64),
                                     array([     4,      7,     12, …, 790121,
790126, 790127], dtype=int64)),
                                    (array([     0,      1,      2, …, 790125,
790126, 790127], dtype=int64),
                                     array([    12,     29,     30, …, 790100,
790105, 790123], dtype=int64))],
                              estimator=DecisionTreeRegressor(), param_grid={},
                              scoring='neg_mean_absolute_error'))],
         verbose=True)
```

[504]:
```python
# Mean cross-validated score of the best_estimator
decision_tree_regressor_pipeline['model cv'].best_score_
```

[504]: -20.16495723193819

9.4 K Nearest Neighbors Regressor

[505]:
```python
from sklearn.neighbors import KNeighborsRegressor

k_neighbors_regressor_pipeline = Pipeline(
    steps=[
        ('columns', colTransformer),
        ('scale', StandardScaler()), # scales every column
        ('model cv', GridSearchCV(estimator=KNeighborsRegressor(),
                                param_grid={},
                                scoring="neg_mean_absolute_error",
                                cv = cv_indices_list))
    ],
    verbose=True
)

k_neighbors_regressor_pipeline.fit(Xtrain, ytrain)
```

```
[Pipeline] … (step 1 of 3) Processing columns, total=   5.4s
[Pipeline] … (step 2 of 3) Processing scale, total=   0.4s
[Pipeline] … (step 3 of 3) Processing model cv, total=207.5min
```

[505]:
```
Pipeline(steps=[('columns',
                 ColumnTransformer(transformers=[('TargetEncoder',
TargetEncoder(drop_invariant=True),
                                                  ['Country', 'Division',
                                                   'MeetState', 'Federation',
                                                   'MeetCountry']),
                                                 ('OneHotEncoder',
OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False),
                                                  ['Tested', 'Sex',
                                                   'Equipment']),
                                                 ('IterativeImputer',
                                                  IterativeImputer(),
                                                  ['Age', 'BodyweightKg']),
                                                 ('passthrough'…
                                   array([     2,      8,     13, …, 790116,
790126, 790127], dtype=int64)),
                                  (array([     0,      1,      2, …, 790123,
790124, 790125], dtype=int64),
                                   array([     4,      7,     12, …, 790121,
790126, 790127], dtype=int64)),
```

```
                                    (array([     0,      1,      2, …, 790125,
790126, 790127], dtype=int64),
                                     array([    12,     29,     30, …, 790100,
790105, 790123], dtype=int64))],
                               estimator=KNeighborsRegressor(), param_grid={},
                               scoring='neg_mean_absolute_error'))],
         verbose=True)
```

[506]:
```
# Mean cross-validated score of the best_estimator
k_neighbors_regressor_pipeline['model cv'].best_score_
```

[506]:
```
-15.766715468856498
```

9.5 XGBoost Regressor

[507]:
```
from xgboost import XGBRegressor

xg_boost_regressor_pipeline = Pipeline(
    steps=[
        ('columns', colTransformer),
        ('model cv', GridSearchCV(estimator=XGBRegressor(),
                                  param_grid={},
                                  scoring="neg_mean_absolute_error",
                                  cv = cv_indices_list))
    ],
    verbose=True
)

xg_boost_regressor_pipeline.fit(Xtrain, ytrain)
```

```
[Pipeline] … (step 1 of 2) Processing columns, total=   6.6s
[Pipeline] … (step 2 of 2) Processing model cv, total= 3.8min
```

[507]:
```
Pipeline(steps=[('columns',
                 ColumnTransformer(transformers=[('TargetEncoder',
TargetEncoder(drop_invariant=True),
                                                  ['Country', 'Division',
                                                   'MeetState', 'Federation',
                                                   'MeetCountry']),
                                                 ('OneHotEncoder',
OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False),
                                                  ['Tested', 'Sex',
                                                   'Equipment']),
                                                 ('IterativeImputer',
                                                  IterativeImputer(),
                                                  ['Age', 'BodyweightKg']),
                                                 ('passthrough'…
```

```
                                                        max_delta_step=None,
                                                        max_depth=None,
                                                        min_child_weight=None,
                                                        missing=nan,
                                                        monotone_constraints=None,
                                                        n_estimators=100,
                                                        n_jobs=None,
                                                        num_parallel_tree=None,
                                                        random_state=None,
                                                        reg_alpha=None,
                                                        reg_lambda=None,
                                                        scale_pos_weight=None,
                                                        subsample=None,
                                                        tree_method=None,
                                                        validate_parameters=None,
                                                        verbosity=None),
                                       param_grid={},
                                       scoring='neg_mean_absolute_error'))],
         verbose=True)
```

[508]:
```
# Mean cross-validated score of the best_estimator
xg_boost_regressor_pipeline['model cv'].best_score_
```

[508]: -13.949171582232406

9.6 Random Forest Regressor

[509]:
```
from sklearn.ensemble import RandomForestRegressor

random_forest_regressor_pipeline = Pipeline(
    steps=[
        ('columns', colTransformer),
        ('model cv', GridSearchCV(estimator=RandomForestRegressor(),
                                  param_grid={},
                                  scoring="neg_mean_absolute_error",
                                  cv = cv_indices_list))
    ],
    verbose=True
)

random_forest_regressor_pipeline.fit(Xtrain, ytrain)
```

```
[Pipeline] … (step 1 of 2) Processing columns, total=   6.4s
[Pipeline] … (step 2 of 2) Processing model cv, total=55.6min
```

[509]:
```
Pipeline(steps=[('columns',
                 ColumnTransformer(transformers=[('TargetEncoder',
TargetEncoder(drop_invariant=True),
```

```
                                                              ['Country', 'Division',
                                                               'MeetState', 'Federation',
                                                               'MeetCountry']),
                                                             ('OneHotEncoder',
OneHotEncoder(handle_unknown='ignore',
                                                                            sparse=False),
                                                              ['Tested', 'Sex',
                                                               'Equipment']),
                                                             ('IterativeImputer',
                                                              IterativeImputer(),
                                                              ['Age', 'BodyweightKg']),
                                                             ('passthrough'…
                                                array([     2,       8,      13, …, 790116,
790126, 790127], dtype=int64)),
                                               (array([     0,       1,       2, …, 790123,
790124, 790125], dtype=int64),
                                                array([     4,       7,      12, …, 790121,
790126, 790127], dtype=int64)),
                                               (array([     0,       1,       2, …, 790125,
790126, 790127], dtype=int64),
                                                array([    12,      29,      30, …, 790100,
790105, 790123], dtype=int64))],
                                           estimator=RandomForestRegressor(), param_grid={},
                                           scoring='neg_mean_absolute_error'))],
         verbose=True)
```

[510]:
```
# Mean cross-validated score of the best_estimator
random_forest_regressor_pipeline['model cv'].best_score_
```

[510]: -14.777143747140917

9.7 Linear Support Vector Regressor

[511]:
```
from sklearn.svm import LinearSVR

# Not using base SVR() because of the quadratic fit time with respect to␣
 ↪observations

linear_svr_regressor_pipeline = Pipeline(
    steps=[
        ('columns', colTransformer),
        ('scale', StandardScaler()), # scales every column
        ('model cv', GridSearchCV(estimator=LinearSVR(),
                                  param_grid={},
                                  scoring="neg_mean_absolute_error",
                                  cv = cv_indices_list))
    ],
```

```
    verbose=True
)

linear_svr_regressor_pipeline.fit(Xtrain, ytrain)
```

```
[Pipeline] … (step 1 of 3) Processing columns, total=   5.2s
[Pipeline] … (step 2 of 3) Processing scale, total=   0.6s
[Pipeline] … (step 3 of 3) Processing model cv, total=  44.4s
```

[511]:

```
Pipeline(steps=[('columns',
                 ColumnTransformer(transformers=[('TargetEncoder',
TargetEncoder(drop_invariant=True),
                                                  ['Country', 'Division',
                                                   'MeetState', 'Federation',
                                                   'MeetCountry']),
                                                 ('OneHotEncoder',
OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False),
                                                  ['Tested', 'Sex',
                                                   'Equipment']),
                                                 ('IterativeImputer',
                                                  IterativeImputer(),
                                                  ['Age', 'BodyweightKg']),
                                                 ('passthrough'…
                                   array([     2,      8,     13, …, 790116,
790126, 790127], dtype=int64)),
                                  (array([     0,      1,      2, …, 790123,
790124, 790125], dtype=int64),
                                   array([     4,      7,     12, …, 790121,
790126, 790127], dtype=int64)),
                                  (array([     0,      1,      2, …, 790125,
790126, 790127], dtype=int64),
                                   array([    12,     29,     30, …, 790100,
790105, 790123], dtype=int64))],
                              estimator=LinearSVR(), param_grid={},
                              scoring='neg_mean_absolute_error'))],
         verbose=True)
```

[512]:

```
# Mean cross-validated score of the best_estimator
linear_svr_regressor_pipeline['model cv'].best_score_
```

[512]:

```
-15.124922488081978
```

9.8 Cross Validation Results

The following results are the averaged performance across 5 randomly selected subsets of the training set.

Which model to choose? (based on Mean Absolute Error and default hyperparameters)

- **Linear Regression**:
  15.16 MAE
  13.2 seconds to fit
- **Ridge Regression**:
  15.16 MAE
  9.6 seconds to fit
- **Decision Tree**:
  20.16 MAE
  75.9 seconds to fit
- **K Nearest Neighbors**:
  15.77 MAE
  12457.2 seconds to fit
- **XGBoost**:
  13.95 MAE
  324.4 seconds to fit
- **Random Forest**:
  14.78 MAE
  3345.3 seconds to fit
- **Linear Support Vector Machine**:
  15.12 MAE
  50.3 seconds to fit

XGBoost performed the best, with under 14kg MAE, and a good enough fit time. KNN and RandomForest are unusable in this project because if their inordinately long fit times. The linear models performed extremely well for how fast they fit, with only about a 1.2kg MAE difference between them and XGBoost.

If I cared about speed of fitting and inference, or if I had much more data, I would honestly just go with one of the linear models. Statistical inference is much more fruitful with these models because (at least with the statsmodels module) you can see each feature's coefficient, respective t-test p-values, ANOVA compared to a base model, and R squared value.

I'm going to continue with XGBoost because of its superior predictive performance. Statistical inference is still possible with its built in feature importance algorithm and by using Shapley Values.

---

10 *Hyperparameter Optimization*

[415]:
```
# View default hyperparameters
xg_boost_regressor_pipeline['model cv'].best_estimator_
```

[415]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
```

```
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

10.1 Grid Search CV

[498]:
```
# Find the best hyperparamters for the chosen model using RandomizedSearchCV or␣
 ↪GridSearchCV
# default 100 estimators, 6 max_depth, and .3 learning rate

xg_boost_regressor_pipeline_grid = Pipeline(
    steps=[
        ('columns', colTransformer),
        ('model grid search cv', GridSearchCV(estimator=XGBRegressor(),
                                param_grid={'n_estimators': [50, 100, 150, 200],
                                            'max_depth': [5, 6, 7, 8],
                                            'learning_rate': [.1, .2, .3, .4]},
                                scoring="neg_mean_absolute_error",
                                cv = cv_indices_list))
    ],
    verbose=True
)

xg_boost_regressor_pipeline_grid.fit(Xtrain, ytrain)
```

```
[Pipeline] … (step 1 of 2) Processing columns, total=   6.7s
[Pipeline]  (step 2 of 2) Processing model grid search cv, total=272.3min
```

[498]:
```
Pipeline(steps=[('columns',
                 ColumnTransformer(transformers=[('TargetEncoder',
TargetEncoder(drop_invariant=True),
                                                  ['Country', 'Division',
                                                   'MeetState', 'Federation',
                                                   'MeetCountry']),
                                                 ('OneHotEncoder',
OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False),
                                                  ['Tested', 'Sex',
                                                   'Equipment']),
                                                 ('IterativeImputer',
                                                  IterativeImputer(),
                                                  ['Age', 'BodyweightKg']),
                                                 ('passthrough'…
                                                     monotone_constraints=None,
                                                     n_estimators=100,
                                                     n_jobs=None,
```

```
                                            num_parallel_tree=None,
                                            random_state=None,
                                            reg_alpha=None,
                                            reg_lambda=None,
                                            scale_pos_weight=None,
                                            subsample=None,
                                            tree_method=None,
                                            validate_parameters=None,
                                            verbosity=None),
                                 param_grid={'learning_rate': [0.1, 0.2, 0.3, 0.4],
                                             'max_depth': [5, 6, 7, 8],
                                             'n_estimators': [50, 100, 150, 200]},
                                 scoring='neg_mean_absolute_error'))],
         verbose=True)
```

[499]:
```
# see what the best model was in the grid search
xg_boost_regressor_pipeline_grid['model grid search cv'].best_estimator_
```

[499]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=8,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=200, n_jobs=8, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

The best hyperparameters were max_depth of 8, n_estimators of 200, and a learning rate of .1.

[21]:
```
from xgboost import XGBRegressor
```

10.2 Final Pipeline

[22]:
```
# Fit the model on the test set using tuned hyperparameters

final_pipeline = Pipeline(
    steps=[
        ('columns', colTransformer),
        ('model', XGBRegressor(base_score=0.5, booster='gbtree',␣
 ↪colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=8,
             min_child_weight=1, missing=np.nan, monotone_constraints='()',
             n_estimators=200, n_jobs=8, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None))
    ],
```

```
    verbose=True
)

final_pipeline.fit(Xtrain, ytrain)
```

```
[Pipeline] … (step 1 of 2) Processing columns, total=   6.5s
[Pipeline] … (step 2 of 2) Processing model, total= 2.9min
```

[22]:
```
Pipeline(steps=[('columns',
                 ColumnTransformer(transformers=[('TargetEncoder',
TargetEncoder(drop_invariant=True),
                                                  ['Country', 'Division',
                                                   'MeetState', 'Federation',
                                                   'MeetCountry']),
                                                 ('OneHotEncoder',
OneHotEncoder(handle_unknown='ignore',
                                                               sparse=False),
                                                  ['Tested', 'Sex',
                                                   'Equipment']),
                                                 ('IterativeImputer',
                                                  IterativeImputer(),
                                                  ['Age', 'BodyweightKg']),
                                                 ('passthrough'…
                              colsample_bytree=1, gamma=0, gpu_id=-1,
                              importance_type='gain',
                              interaction_constraints='', learning_rate=0.1,
                              max_delta_step=0, max_depth=8, min_child_weight=1,
                              missing=nan, monotone_constraints='()',
                              n_estimators=200, n_jobs=8, num_parallel_tree=1,
                              random_state=0, reg_alpha=0, reg_lambda=1,
                              scale_pos_weight=1, subsample=1,
                              tree_method='exact', validate_parameters=1,
                              verbosity=None))],
         verbose=True)
```

10.3 Test Data Performance

[27]:
```
from sklearn.metrics import mean_absolute_error

mean_absolute_error(final_pipeline.predict(Xtest), ytest)
```

[27]:
```
13.872437270549145
```

Very similar MAE to the training cross validation; within .1 MAE.

---

11 *Feature Selection*

Why? - Reduces variance by having fewer features - Train and do inference faster with fewer features - Needing fewer features allows one to gather more data faster

11.1 Get Feature Names

Sci-kit Learn Pipelines return numpy arrays, so I need to manually put back the names of the features and make the pipeline's output a DataFrame.

[27]:
```
OneHotEncoder(categories='auto',
              handle_unknown='ignore',
              sparse=False).fit(Xtrain[["Tested", "Sex", "Equipment"]], ytrain).
 ↪get_feature_names(["Tested", "Sex", "Equipment"])
```

[27]:
```
array(['Tested_Yes', 'Tested_nan', 'Sex_F', 'Sex_M',
       'Equipment_Multi-ply', 'Equipment_Raw', 'Equipment_Single-ply',
       'Equipment_Wraps'], dtype=object)
```

[16]:
```
feature_names = ["Country", "Division", "MeetState", "Federation",␣
 ↪"MeetCountry",
                 'Tested_Yes', 'Tested_nan', 'Sex_F', 'Sex_M',
                 'Equipment_Multi-ply', 'Equipment_Raw', 'Equipment_Single-ply',
                 'Equipment_Wraps', "Age", "BodyweightKg", "Best3BenchKg",␣
 ↪"Best3SquatKg"]
```

[36]:
```
transformed_data = pd.DataFrame(output)
transformed_data.columns = feature_names
transformed_data.head()
```

[36]:
```
      Country    Division   MeetState  Federation  MeetCountry  Tested_Yes  \
0  177.942157  145.595666  203.632652  206.435468   201.522116         0.0
1  177.942157  145.595666  203.632652  206.435468   201.522116         0.0
2  177.942157  145.595666  203.632652  206.435468   201.522116         0.0
3  177.942157  145.595666  203.632652  206.435468   201.522116         0.0
4  177.942157  145.595666  203.632652  206.435468   201.522116         0.0

   Tested_nan  Sex_F  Sex_M  Equipment_Multi-ply  Equipment_Raw  \
0         1.0    1.0    0.0                  0.0            0.0
1         1.0    1.0    0.0                  0.0            0.0
2         1.0    1.0    0.0                  0.0            0.0
3         1.0    1.0    0.0                  0.0            0.0
4         1.0    1.0    0.0                  0.0            0.0

   Equipment_Single-ply  Equipment_Wraps   Age  BodyweightKg  Best3BenchKg  \
0                   0.0              1.0  29.0          59.8          55.0
1                   0.0              1.0  29.0          58.5          67.5
2                   0.0              1.0  23.0          60.0          72.5
```

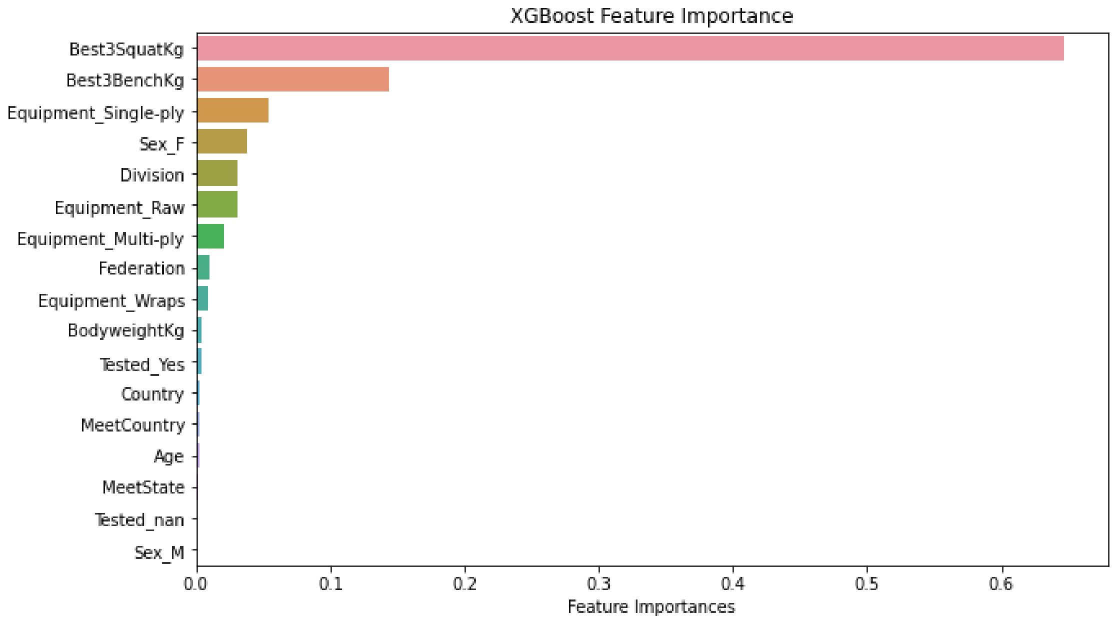
```
3                   0.0               1.0  45.0         104.0          80.0
4                   0.0               1.0  37.0          74.0          82.5

   Best3SquatKg
0         105.0
1         120.0
2         105.0
3         140.0
4         142.5
```

11.2 XGBoost Feature Importance

[35]:
```
importance = pd.Series(data=final_pipeline['model'].feature_importances_,␣
 ↪name="Feature Importances")
importance.index = feature_names
importance = importance.sort_values(ascending=False)
importance
```

[35]:
```
Best3SquatKg           0.647175
Best3BenchKg           0.144089
Equipment_Single-ply   0.053762
Sex_F                  0.037815
Division               0.030842
Equipment_Raw          0.030577
Equipment_Multi-ply    0.020431
Federation             0.009740
Equipment_Wraps        0.008967
BodyweightKg           0.003791
Tested_Yes             0.003382
Country                0.002622
MeetCountry            0.002522
Age                    0.002386
MeetState              0.001898
Tested_nan             0.000000
Sex_M                  0.000000
Name: Feature Importances, dtype: float32
```

[45]:
```
fig, ax = plt.subplots(figsize=[10,6])
ax = sns.barplot(x = importance, y = importance.index, orient = 'h')
ax.set_title("XGBoost Feature Importance")
plt.show()
```



It's not surprising that squat is an important way to determine one's deadlift, but what is surprising is how it seems to completely dominate the other features.

11.3 Lasso Regression Regularization

[47]:
```
# Use L1 Regularization to see the most important features
# The less important coefficients will be regularized to 0
# Make alpha higher to remove more features

from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel

# Lasso Relies on scaled data
scaled_Xtrain = StandardScaler().fit_transform(transformed_data)

lasso = Lasso(alpha=1).fit(scaled_Xtrain, ytrain)

# Select the nonzero coefficients
selector = SelectFromModel(lasso, prefit=True)

Xtrain_new = selector.transform(scaled_Xtrain)
```

[49]:
```
# Create a list of which features were selected

selected_X = pd.DataFrame(selector.inverse_transform(Xtrain_new),
                          index=np.arange(Xtrain_new.shape[0]),
                          columns=feature_names)
```

```
selected_features = selected_X.columns[selected_X.sum() != 0]

selected_features
```

[49]:
```
Index(['Division', 'MeetState', 'Sex_F', 'Equipment_Multi-ply',
       'Equipment_Raw', 'Equipment_Single-ply', 'BodyweightKg', 'Best3BenchKg',
       'Best3SquatKg'],
      dtype='object')
```

[51]:
```
# See which features should be removed according to this method

for col in transformed_data.columns.tolist():
    if col not in selected_features:
        print(col)
```

```
Country
Federation
MeetCountry
Tested_Yes
Tested_nan
Sex_M
Equipment_Wraps
Age
```

Age not being useful very much surprises me.
Also, how is being tested not useful? People taking PEDs should be able to lift more.
For some reason, MeetState is an important feature with this method, even though it has almost 0 importance according to XGBoost.
The rest of the unimportant features are in line with what XGBoost feature importance said.

11.4 Permutation Importance

[28]:
```
# save these variables for the following cells

model = final_pipeline['model']
transformed_Xtest = colTransformer.transform(Xtest)
```

[56]:
```
# Different aproach to see feature importance
# Permutation importance

import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(model).fit(transformed_Xtest, ytest)
eli5.show_weights(perm, top=100, feature_names = feature_names)
```

[56]:
```
<IPython.core.display.HTML object>
```

Permutation importance tells a similar story.
Best squat and bench are important, and so is bodyweight and Sex_F.
Federation and Division are important too.

11.5 Selected Features

Test the model's MAE score on only the selected features to see if it reduces variance.

[17]:
```
# decide which features to keep
# using results from XGBoost feature importance, Lasso Regression, and␣
↪Permutation Importance

removed_features = ["Sex_M", "Tested_Yes", "Tested_nan", "Equipment_Wraps",
                    "Age", "Country", "MeetCountry"]

best_features = set(feature_names) - set(removed_features)
best_features
```

[17]:
```
{'Best3BenchKg',
 'Best3SquatKg',
 'BodyweightKg',
 'Division',
 'Equipment_Multi-ply',
 'Equipment_Raw',
 'Equipment_Single-ply',
 'Federation',
 'MeetState',
 'Sex_F'}
```

[22]:
```
# create a pipeline for the feature subset
reduced_colTransformer = ColumnTransformer(
    transformers=[
        ("TargetEncoder", TargetEncoder(verbose=0,
                                        drop_invariant=True,
                                        handle_unknown='value',
                                        handle_missing='value'),
                                        ["Division", "MeetState",␣
↪"Federation"]), # returns 5 cols
        ("OneHotEncoder", OneHotEncoder(drop='first',
                                        categories='auto',
                                        handle_unknown='error',
                                        sparse=False),
                                        ["Sex", "Equipment"]), # returns 8 cols
        ("IterativeImputer", IterativeImputer(missing_values=np.nan,
                                            max_iter=10),
                                            ["BodyweightKg"]), # returns 2 cols
        ("passthrough", "passthrough", ["Best3BenchKg", "Best3SquatKg"]) #␣
↪returns 2 cols
```

```
    ],
    remainder='drop'
)
```

[23]:
```
# fit the pipeline
reduced_final_pipeline = Pipeline(
    steps=[
        ('columns', reduced_colTransformer),
        ('model', XGBRegressor(base_score=0.5, booster='gbtree',␣
↪colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=8,
             min_child_weight=1, missing=np.nan, monotone_constraints='()',
             n_estimators=200, n_jobs=8, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None))
    ],
    verbose=True
)

# fit the pipeline
reduced_final_pipeline.fit(Xtrain[['Best3BenchKg',
                                   'Best3SquatKg',
                                   'BodyweightKg',
                                   'Sex',
                                   'Equipment',
                                   'Division',
                                   'MeetState',
                                   'Federation']], ytrain)
```

```
[Pipeline] … (step 1 of 2) Processing columns, total=   3.9s
[Pipeline] … (step 2 of 2) Processing model, total= 2.2min
```

[23]:
```
Pipeline(steps=[('columns',
                 ColumnTransformer(transformers=[('TargetEncoder',
TargetEncoder(drop_invariant=True),
                                                  ['Division', 'MeetState',
                                                   'Federation']),
                                                 ('OneHotEncoder',
                                                  OneHotEncoder(drop='first',
                                                                sparse=False),
                                                  ['Sex', 'Equipment']),
                                                 ('IterativeImputer',
                                                  IterativeImputer(),
                                                  ['BodyweightKg']),
                                                 ('passthrough', 'passthrough',
```

```
                                        ['Best3BenchKg',
                                         'Best3SquatKg'])]))…
                                    colsample_bytree=1, gamma=0, gpu_id=-1,
                                    importance_type='gain',
                                    interaction_constraints='', learning_rate=0.1,
                                    max_delta_step=0, max_depth=8, min_child_weight=1,
                                    missing=nan, monotone_constraints='()',
                                    n_estimators=200, n_jobs=8, num_parallel_tree=1,
                                    random_state=0, reg_alpha=0, reg_lambda=1,
                                    scale_pos_weight=1, subsample=1,
                                    tree_method='exact', validate_parameters=1,
                                    verbosity=None))],
         verbose=True)
```

11.6 Selected Features Performance

[25]:
```python
# See the model's MAE score on only the selected features

from sklearn.metrics import mean_absolute_error

mean_absolute_error(reduced_final_pipeline.predict(Xtest[['Best3BenchKg',
                                    'Best3SquatKg',
                                    'BodyweightKg',
                                    'Sex',
                                    'Equipment',
                                    'Division',
                                    'MeetState',
                                    'Federation']]), ytest)
```

[25]: 13.969082189806572

After removing the 10 least important features (out of 17), the performance improved by .004 MAE, which could probably just be reduced to random chance. Removing these features did speed up fitting and inference, however.

Proof of Speed-Up:

With all features: - Preprocessing: 6.6s - Model Training: 2.7 minutes - Test set prediction: 1.4s

With most important features: - Preprocessing: 3.9s - Model Training: 2.2 minutes - Test set prediction: 1.3s

---

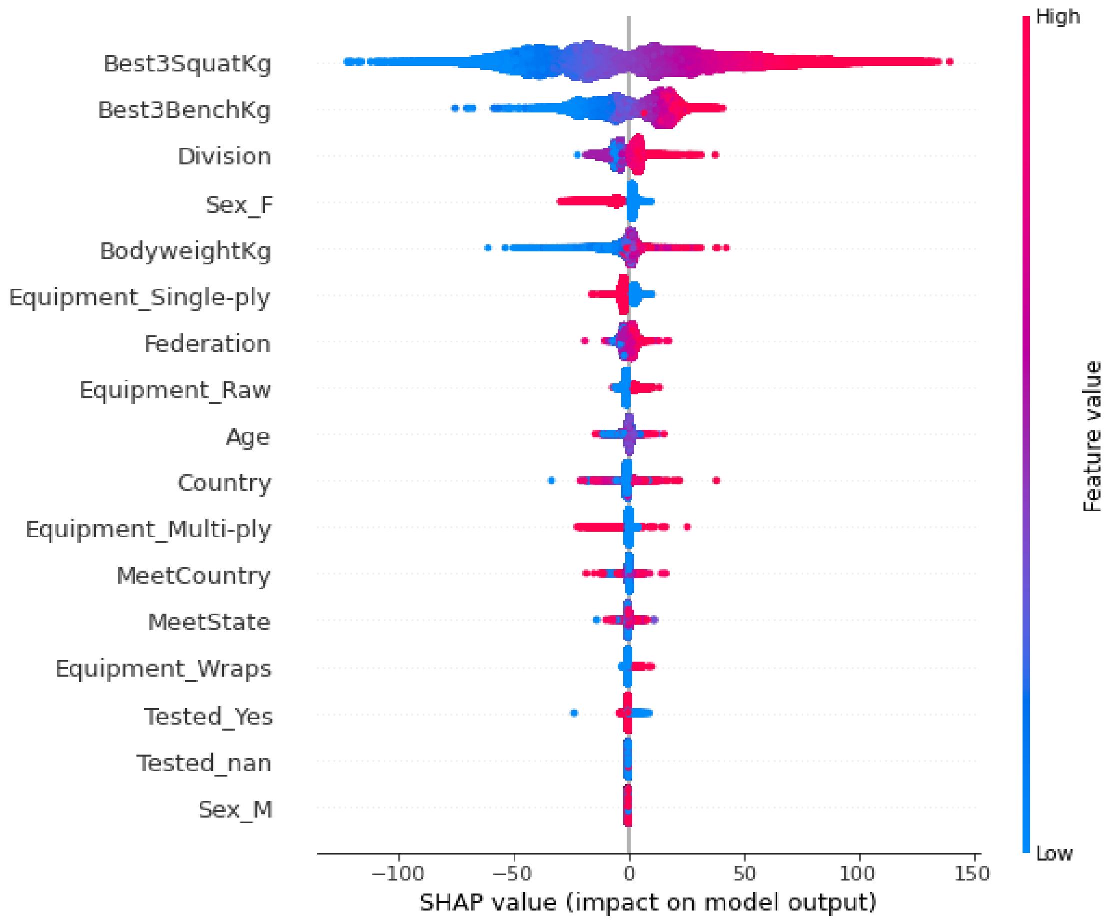
12 *Machine Learning Explainability*

Making sense of the model's predictions

[23]:
```
# save these variables for the following cells

model = final_pipeline['model']
transformed_Xtest = colTransformer.transform(Xtest)
feature_names = ["Country", "Division", "MeetState", "Federation",␣
 ↪"MeetCountry",
                 'Tested_Yes', 'Tested_nan', 'Sex_F', 'Sex_M',
                 'Equipment_Multi-ply', 'Equipment_Raw', 'Equipment_Single-ply',
                 'Equipment_Wraps', "Age", "BodyweightKg", "Best3BenchKg",␣
 ↪"Best3SquatKg"]
transformed_Xtest = pd.DataFrame(data=transformed_Xtest, columns=feature_names)
```

12.1 Shapley Values

[30]:
```
# Another way to see relationships between variables
# SHAP values and SHAP summary plots

import shap

explainer = shap.TreeExplainer(model)

shap_values = explainer.shap_values(transformed_Xtest)

shap.summary_plot(shap_values, transformed_Xtest)
```

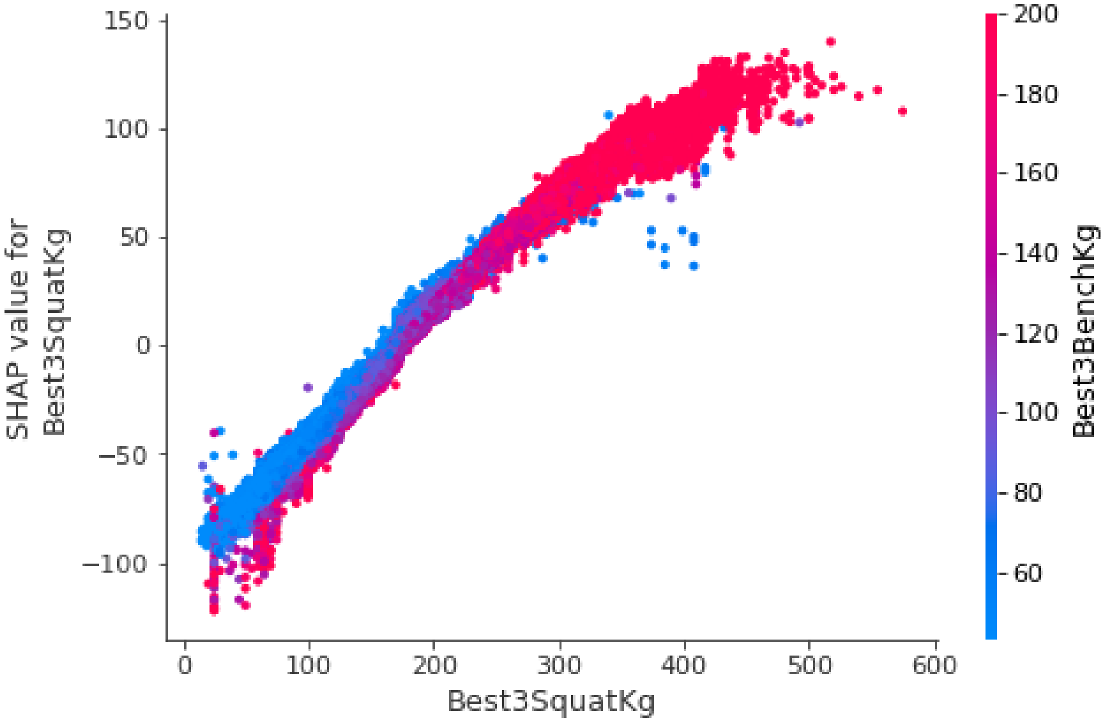


Clear indicators of a high 1 Rep Max Deadlift: * High 1RM Squat and Bench * Higher bodyweight * Male * In a Federation or Division that historically has high deadlifts

Not-so-clear indicators of a high 1RM Deadlift: * For some reason, lifting raw (without equipment) leads to a higher deadlift. I would think using wraps would help more. Maybe the original data is biased in some way. * I would think a lower age would help, but maybe not. * Country of origin and Multi-Ply Equipment can have a positive or negative effect on best Deadlift

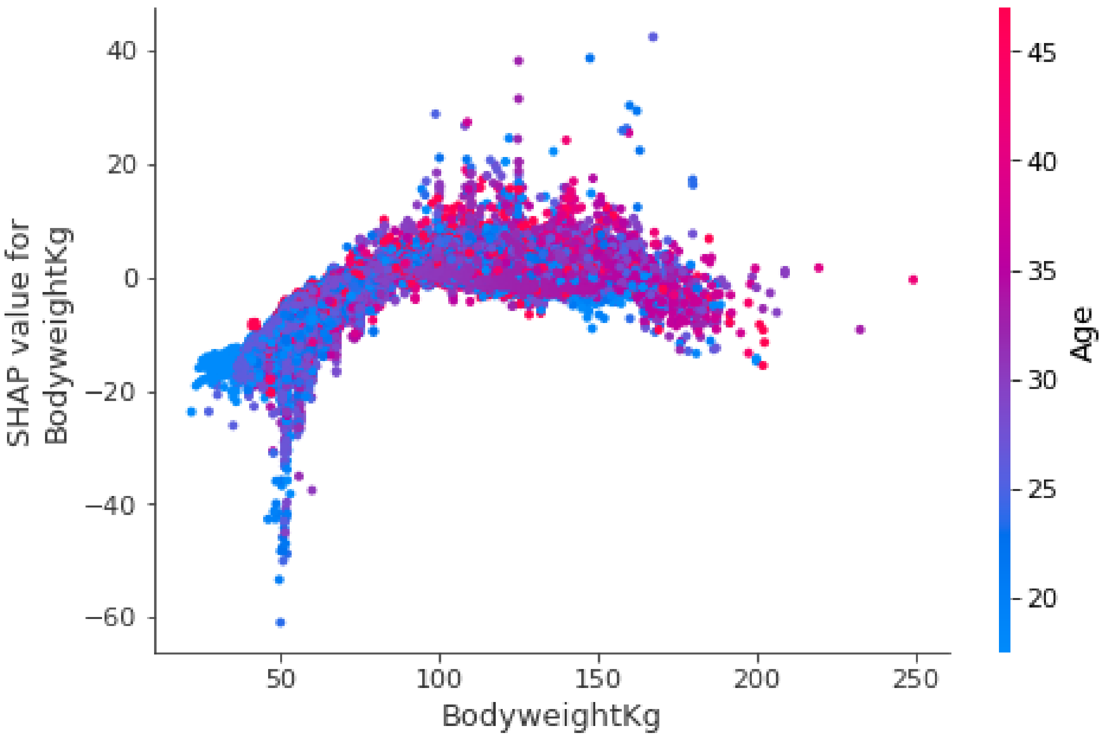
12.2 Partial Dependence Plots

[31]:
```
# Partial Dependence Plot, but enhanced with SHAP values
# Check out best squat and best bench

shap.dependence_plot("Best3SquatKg", shap_values, transformed_Xtest,␣
↪interaction_index="Best3BenchKg")
```



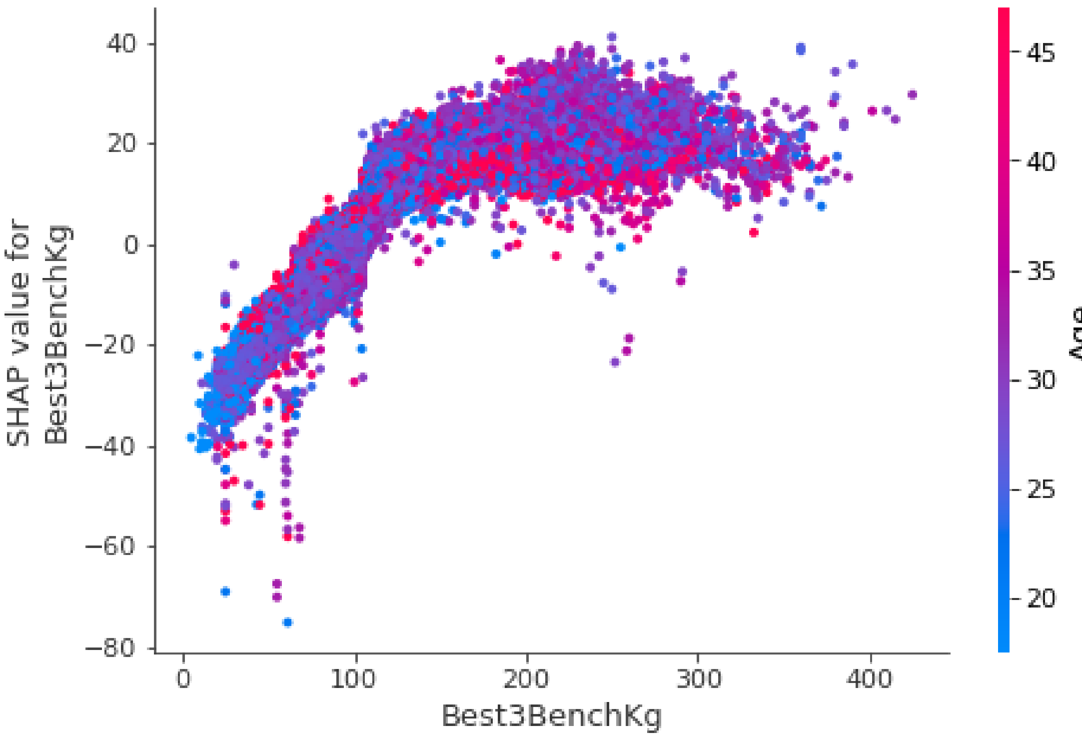
People with the highest Bench have the highest Squat. Go figure.

[32]:
```
# Partial Dependence Plot, but enhanced with SHAP values
# Check out Bodyweight and Age

shap.dependence_plot("BodyweightKg", shap_values, transformed_Xtest,␣
 ↪interaction_index="Age")
```



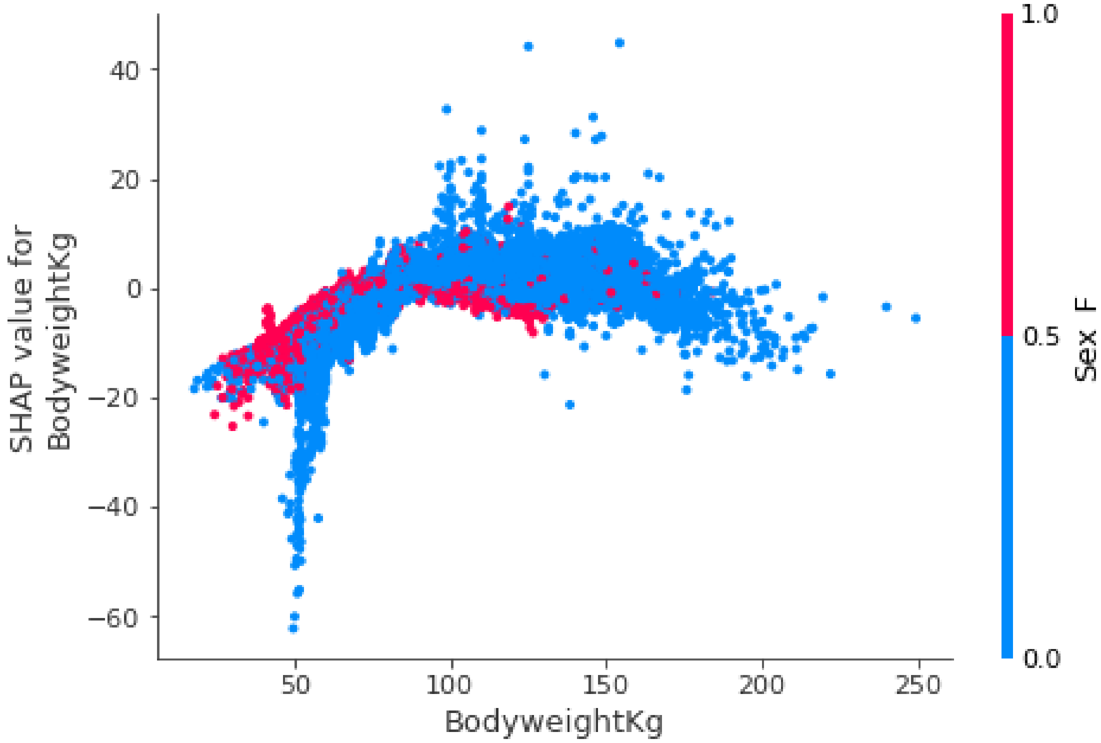
People over 100kg are all over the place in terms of age.

[33]:
```
# Partial Dependence Plot, but enhanced with SHAP values
# Check out Age and Bench

shap.dependence_plot("Best3BenchKg", shap_values, transformed_Xtest,␣
↪interaction_index="Age")
```



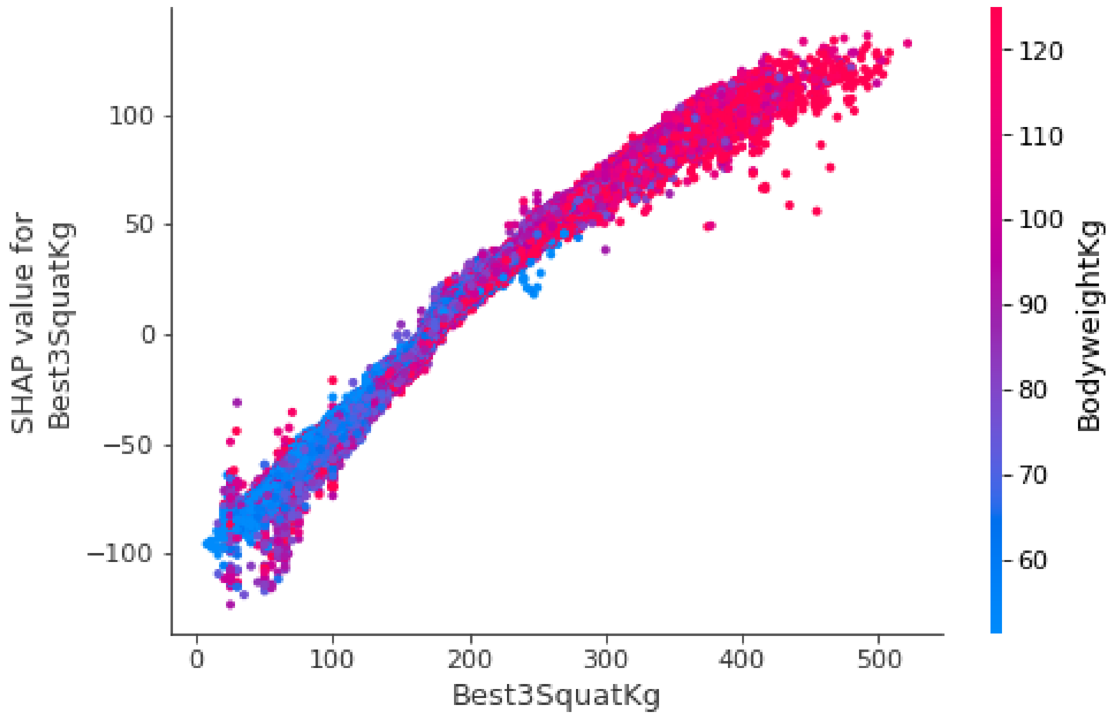
I don't see a clear pattern here between Bench and Age with the SHAP values from Best3DeadliftKg. As seen before, there is the correlation between bench and deadlift, but Age doesn't seem to be a significant covariate. The lowest of ages have the lowest bench and deadlift, but there are plenty of people in their late twenties that lift more than people in their late 40s.

[ ]:
```
# Partial Dependence Plot, but enhanced with SHAP values
# Check out the interaction between Squat and Sex_F

shap.dependence_plot("BodyweightKg", shap_values, transformed_Xtest,␣
 ↪interaction_index="Sex_F")
```



Clearly, there are more males than females in this dataset. Even though most women are of lower body weight, there are some around 50kg bodyweight who have a higher deadlift compared to men of the same bodyweight. Women above 100kg bodyweight have comparatively lower deadlifts than men of the same bodyweight.

[42]:
```
# Partial Dependence Plot, but enhanced with SHAP values
# Check out the interaction between Best3SquatKg and BodyweightKg

shap.dependence_plot("Best3SquatKg", shap_values, transformed_Xtest,␣
 ↪interaction_index="BodyweightKg")
```



Really only the heaviest people are able to squat 400kg+. And this correlates heavily with a higher deadlift because they're both lower body movements.