

Bryan Jamieson

BAMTESTING - <https://www.bamtesting.com/bamscore> (<https://www.bamtesting.com/bamscore>)

Data From Martin (Head Data Scientist) @ Bamtesting

Business Case:

- Can I predict BAMscore using NBA Combine Test Results or use it as a predictor to understand which of the 12 athletic tests actually can improve bamscore?
- How can it give insight to athletes + coaches?

Hypothesis

- Protocols (specifically 2 agility tests) are more important for predicting and or improving bamscore, athletes need to focus on improving those

Target Audience

= High School Basketball Athletes that are trying to plan and play at next level

DATA Explained

bamid

- Number randomly assigned to each athlete without knowing full name

bamscore

- Single Numerical Value that measures and benchmarks athletic performance
- "Athletic SAT Score"
- Standardized athletic assessment that gives coaches insight into where players need improvement

Protocols (Athletic Tests)

approach_vertical

- running start, jump as high as you can

vertical_jump

- stationary start, jump as high as you can

three_quarter_court_sprint

- 75 ft straight sprint

four_way_agility

- run around 4 points in box, run back through

reaction_shuttle

- agility test - start in middle of box, run left, right, run back and finish through left line

Anthros (Body Measurements)

wingspan

- horizontal distance from arms extended side to side

reach

- standing vertical reach

height

- how tall you are in inches

weight

- how much you weigh in pounds

body_comp

- body fat (varies in measurement methods)

hand_length

- vertical length of hand

hand_width

- horizontal length of hand

Flow

Notebook (1) Explore and Clean BAM Data

- Visualize the data to see trends, relationships, outliers
- Clean Outliers
- Make Observation/Hypothesis
- Save as "clean data"

Notebook (2) Modeling

- Model data in modeling notebook (2)
- Try different regressions and models
- Iterate to improve r2 score without over fitting

1) Import Libraries + Data

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as scs
import seaborn as sns
import os
import plotly.express as px
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import maxabs_scale
from sklearn.pipeline import Pipeline
```

```
In [2]: df = pd.read_excel('bam_data.xlsx')
        ## Name the dataset "df"
```

```
In [3]: df.info()
        # See if anything is funky with the data using .info() and the eye test
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1059 entries, 0 to 1058
Data columns (total 16 columns):
BAMid      1057 non-null object
Approach Vertical  986 non-null float64
Vertical Jump  1019 non-null float64
3/4 Court sprint  1026 non-null object
4-Way agility  1020 non-null object
Reaction Shuttle  1024 non-null float64
BAMScore    1056 non-null float64
Wingspan    1012 non-null float64
Reach       1012 non-null float64
Height      1012 non-null float64
Weight      1012 non-null float64
Body Comp   1012 non-null float64
Hand Length 1010 non-null float64
Hand Width  1009 non-null float64
Unnamed: 14  0 non-null float64
Unnamed: 15  1 non-null object
dtypes: float64(12), object(4)
memory usage: 132.5+ KB
```

```
In [4]: df
```

Out[4]:

	BAMid	Approach Vertical	Vertical Jump	3/4 Court sprint	4-Way agility	Reaction Shuttle	BAMScore	Wingspan	Reach	Height
0	1037	33.5	28.5	3.376	11.471	3.669	2003.0	72.75	94.0	70.00
1	656	30.5	21.5	3.486	12.114	3.355	1865.0	82.00	104.5	79.50
2	477	37.0	31.0	3.23	12.036	3.562	2005.0	81.50	99.0	74.00
3	1200	29.0	23.0	3.37	12.509	3.173	1902.0	79.50	101.0	77.50
4	1501	31.0	26.0	3.389	12.724	3.316	1903.0	77.00	101.5	78.00
...
1054	574	36.0	31.0	3.424	12.654	3.635	1917.0	72.00	88.0	68.25
1055	651	31.5	26.5	3.256	11.136	3.343	2029.0	74.00	91.5	68.00
1056	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1057		NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1058	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

1059 rows × 16 columns

Observation - This shows we have 1059 rows and 16 columns

I dont like the looks of Unnamed 14 + 15. Replace spaces with _ for punctuation purposes

```
In [5]: df.columns = [column.strip().lower().replace(" ", "_") for column in df.
columns]
df.head()
```

Out[5]:

	bamid	approach_vertical	vertical_jump	3/4_court_sprint	4- way_agility	reaction_shuttle	bamsco
0	1037	33.5	28.5	3.376	11.471	3.669	2003
1	656	30.5	21.5	3.486	12.114	3.355	1865
2	477	37.0	31.0	3.23	12.036	3.562	2005
3	1200	29.0	23.0	3.37	12.509	3.173	1902
4	1501	31.0	26.0	3.389	12.724	3.316	1903

Have to change spaces (strings) to numeric values to we can work with it

Fill spaces/na with 0 so we can fill with mean later

```
In [6]: column_names = ['bamid', '3/4_court_sprint', '4-way_agility', 'unnamed:_14', 'unnamed:_15']

for column in df.columns:
    if column in column_names:
        df[column].replace('', 0)
        df[column] = pd.to_numeric(df[column], errors='coerce')
        df[column].fillna(df[column].mean(), inplace=True)
        print(f'{column}: fixed!')
# Have to change spaces (strings) to numeric values so we can work with it
```

```
bamid: fixed!
approach_vertical: fixed!
vertical_jump: fixed!
3/4_court_sprint: fixed!
4-way_agility: fixed!
reaction_shuttle: fixed!
bamscore: fixed!
wingspan: fixed!
reach: fixed!
height: fixed!
weight: fixed!
body_comp: fixed!
hand_length: fixed!
hand_width: fixed!
unnamed:_14: fixed!
unnamed:_15: fixed!
```

```
In [7]: df.columns
```

```
Out[7]: Index(['bamid', 'approach_vertical', 'vertical_jump', '3/4_court_sprint',
              '4-way_agility', 'reaction_shuttle', 'bamscore', 'wingspan', 'reach',
              'height', 'weight', 'body_comp', 'hand_length', 'hand_width',
              'unnamed:_14', 'unnamed:_15'],
              dtype='object')
```

```
In [8]: df.rename(columns = {'3/4_court_sprint': 'three_quarter_court_sprint', '4-way_agility': 'four_way_agility'}
                  , inplace = True)
# rename columns so they all work later on in notebook
```

```
In [9]: df.drop(columns=['unnamed:_14', 'unnamed:_15', 'bamid'], inplace=True)
# Remove because only 1 value from our df.info check above
# Check df.info() again to make sure it worked
```

```
In [10]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1059 entries, 0 to 1058
Data columns (total 13 columns):
approach_vertical      1059 non-null float64
vertical_jump          1059 non-null float64
three_quarter_court_sprint  1059 non-null float64
four_way_agility       1059 non-null float64
reaction_shuttle       1059 non-null float64
bamscore               1059 non-null float64
wingspan               1059 non-null float64
reach                  1059 non-null float64
height                 1059 non-null float64
weight                 1059 non-null float64
body_comp              1059 non-null float64
hand_length            1059 non-null float64
hand_width             1059 non-null float64
dtypes: float64(13)
memory usage: 107.7 KB
```

```
In [11]: df.columns.isna()
```

```
# Check and make sure nulls are gone - looking for all false.
```

```
Out[11]: array([False, False, False, False, False, False, False, False, False, False,
                False, False, False, False])
```

Data Visualization + EDA

What does our data even look like?

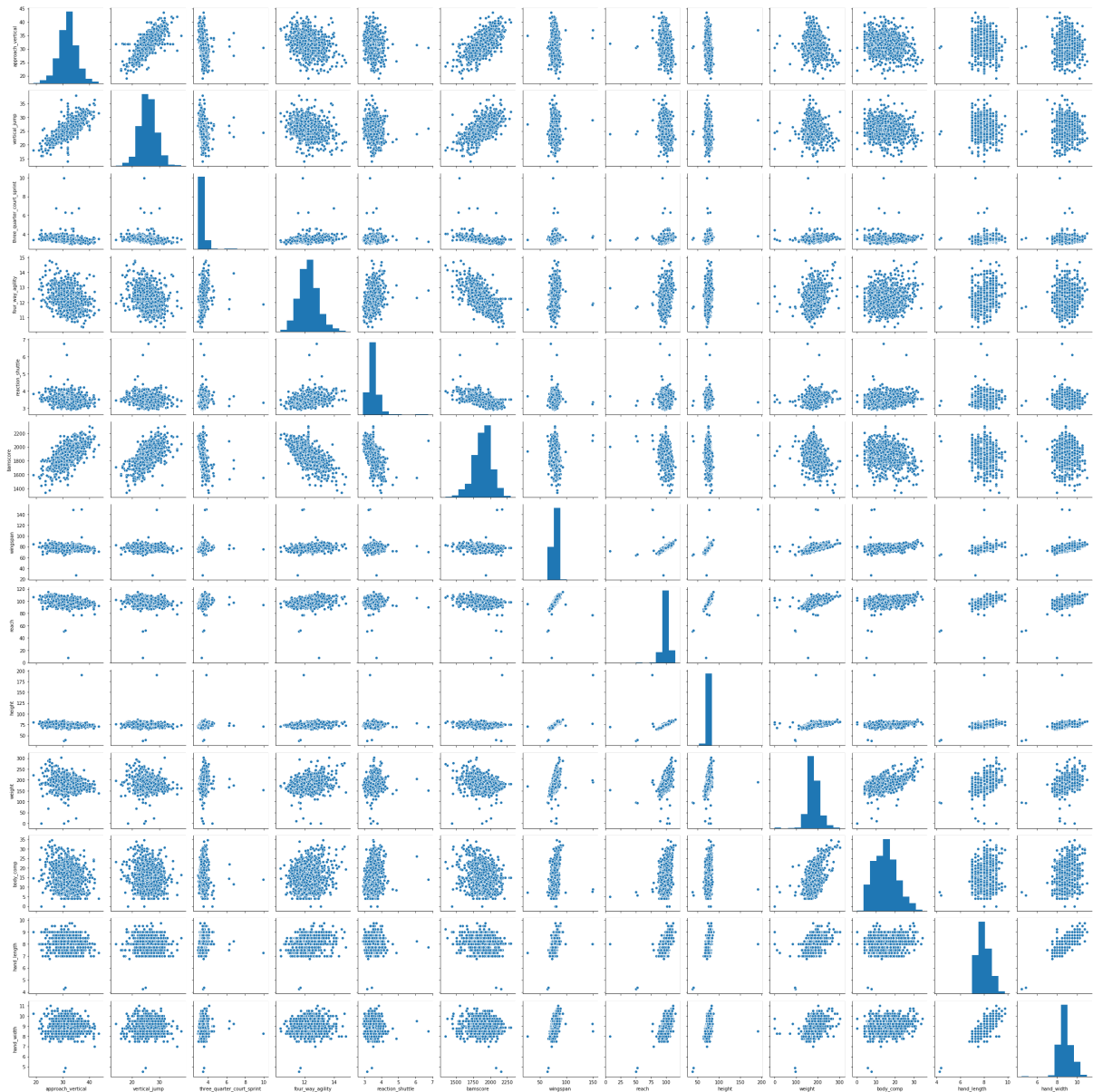
First lets create a simple pairplot to:

- *Use the eye test to look at relationships in our data*

- *Identify multicollinearity*

```
In [12]: # First lets use the eye test and look at relationships in our data + identify multicollinearity.
sns.pairplot(df)
```

```
Out[12]: <seaborn.axisgrid.PairGrid at 0x7fcc910d0160>
```



Observations:

Top right column where points look like they are lining up - this means it could be a categorical variable

Diagonals shows us baseline normal distribution

Can see on reaction shuttle and 3/4 court sprint outlier because it's skewed to left

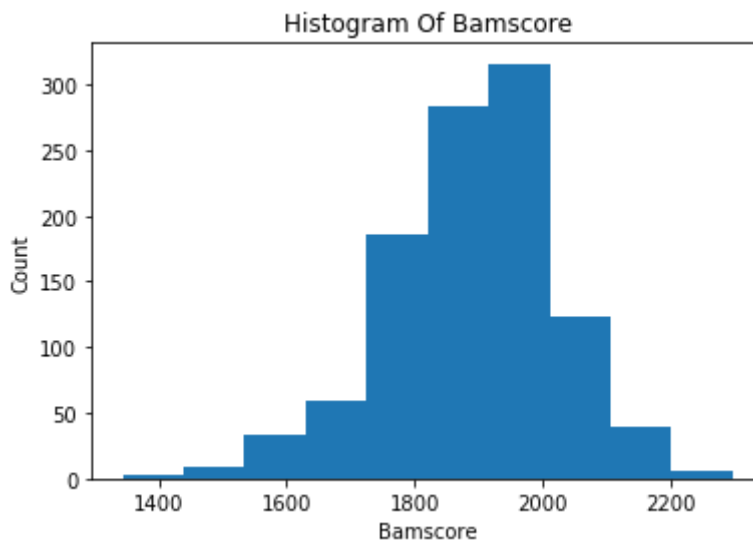
What does bamscore look like? What is it?

```
In [13]: df['bamscore'].describe(),
```

```
Out[13]: (count      1059.000000
         mean      1890.976326
         std       134.866028
         min       1343.000000
         25%       1811.000000
         50%       1899.000000
         75%       1981.000000
         max       2298.000000
         Name: bamscore, dtype: float64,)
```

```
In [14]: fig,ax = plt.subplots(1,1)
         df['bamscore'].plot.hist(bins=10)
         ax.set_title("Histogram Of Bamscore")
         ax.set_xlabel('Bamscore')
         ax.set_ylabel('Count')
```

```
Out[14]: Text(0, 0.5, 'Count')
```



Observations:

BAMscore Range: [1343-2298]

BAMscore Min: [1343]

BAMscore Max: [2298]

- More above avg/good bamscores than poor

Create BAM_score_rank

Split bamscore into 5 ranks (1-5) and add it as a new column

- For visualization purposes + EDA only

- !! Don't forget to drop before saving notebook because bam_score_rank is a classifier and we are doing a regression !!

- In the future, if we decide to change this to a classification, then we can keep bam_score_rank and ranks for other factors

```
In [15]: bam_mu = df['bamscore'].mean()
bam_std = df['bamscore'].std()

min95 = bam_mu-2*bam_std
max95 = bam_mu+2*bam_std

def get_rank_bam_score(bam):
    if bam > bam_mu+1*bam_std:
        return 5
    if bam > bam_mu:
        return 4
    if bam > bam_mu - bam_std:
        return 3
    if bam > bam_mu - 2*bam_std:
        return 2
    return 1
```

```
In [16]: df['bam_score_rank'] = df['bamscore'].apply(get_rank_bam_score)
df.head()
# I ended up dropping this, but if I want to do a classification instead
of regression, use bam_score_rank
# Instead of splitting data into 5 collections and then referencing a list,
I will use bamscorerank
```

Out[16]:

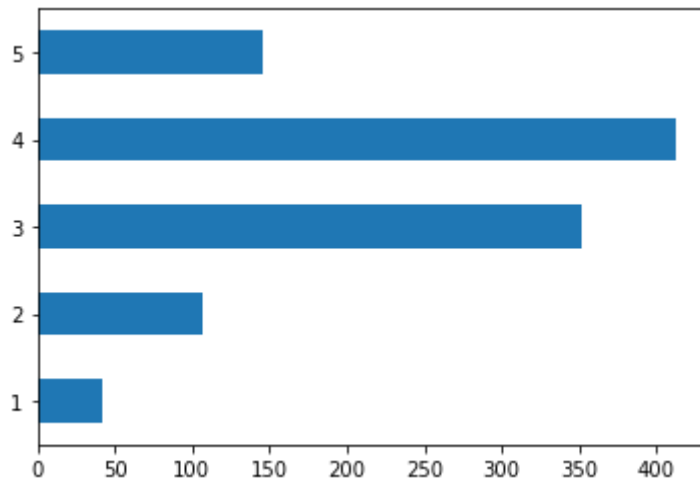
	approach_vertical	vertical_jump	three_quarter_court_sprint	four_way_agility	reaction_shuttle	t
0	33.5	28.5	3.376	11.471	3.669	
1	30.5	21.5	3.486	12.114	3.355	
2	37.0	31.0	3.230	12.036	3.562	
3	29.0	23.0	3.370	12.509	3.173	
4	31.0	26.0	3.389	12.724	3.316	

BAM_SCORE_RANK

1 = Worst Bam Scores | 5 = Best Bam Scores

```
In [17]: av_rank_bam_score = pd.value_counts(df['bam_score_rank'].values, sort=False)
av_rank_bam_score.plot.barh()
```

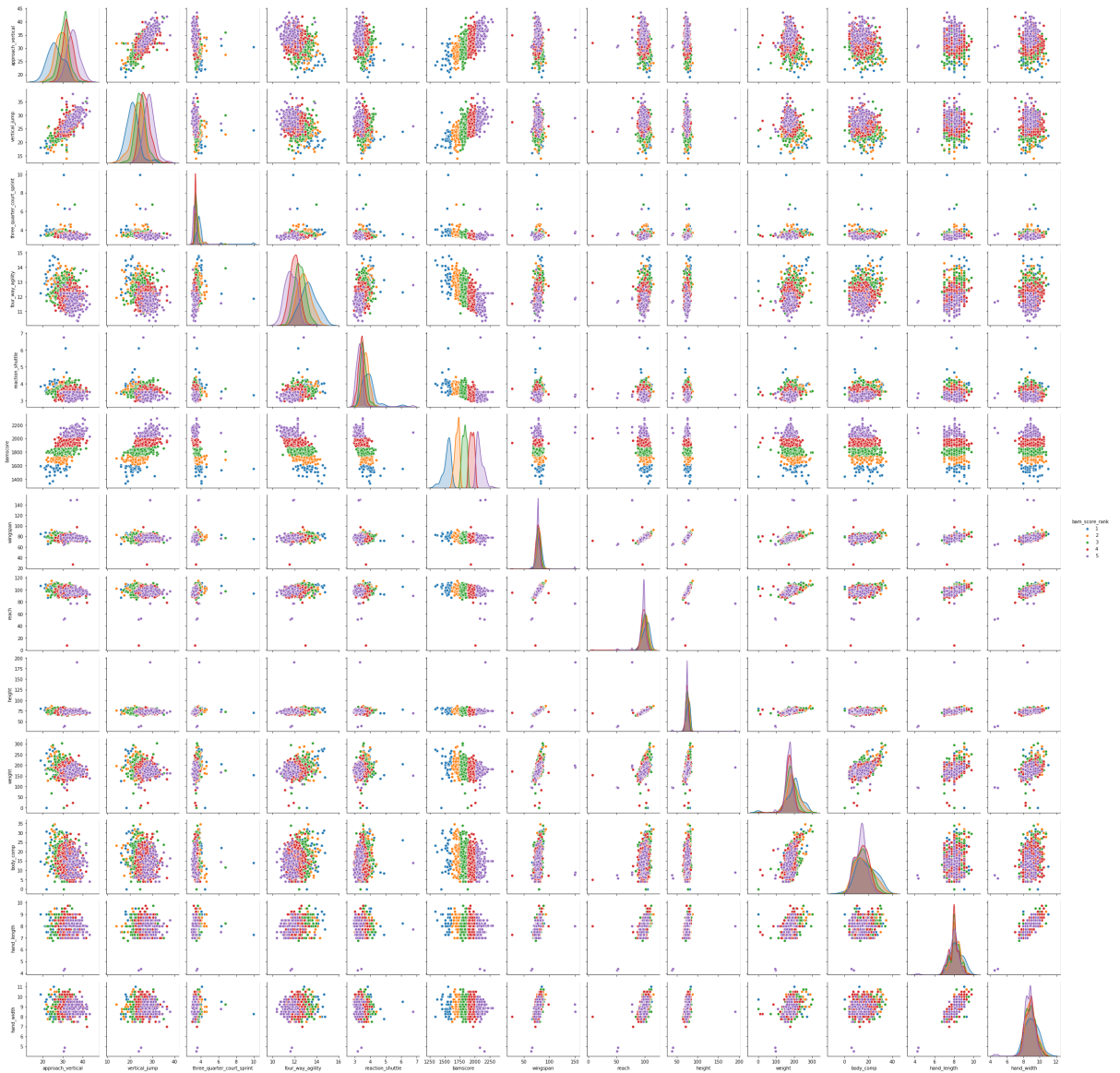
```
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7fccd0ccfd30>
```



I wonder what the pairplot looks like if I split bamscore into 5 groups

```
In [18]: sns.pairplot(df, hue='bam_score_rank')
# hard to see but just gives us more insight + verifies that data makes
# generally makes sense
# https://seaborn.pydata.org/generated/seaborn.pairplot.html
```

Out[18]: <seaborn.axisgrid.PairGrid at 0x7fcc98b06898>



Next let's look at the relationship of each factor with respect to bam score

I picked histograms + violin plots because:

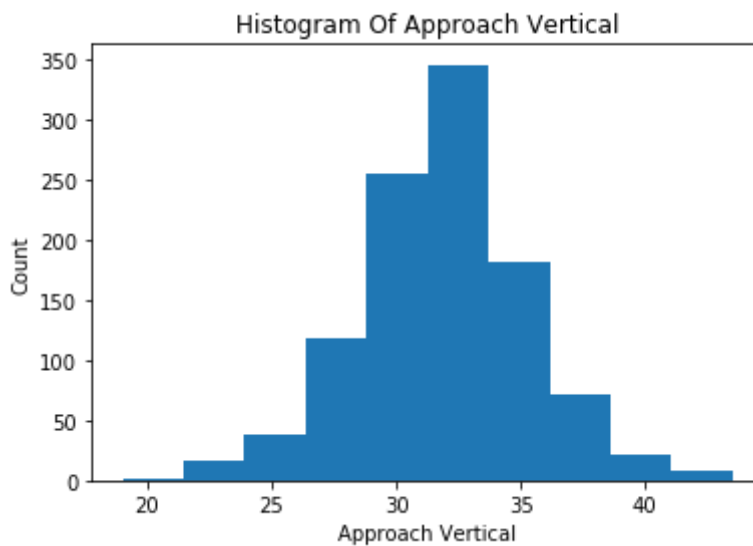
- Gives us quick information to help us understand the data
- Outliers tend to stick out like a sore thumb
- Easy to look at and visualize

Note my observations in comments

Approach Vertical

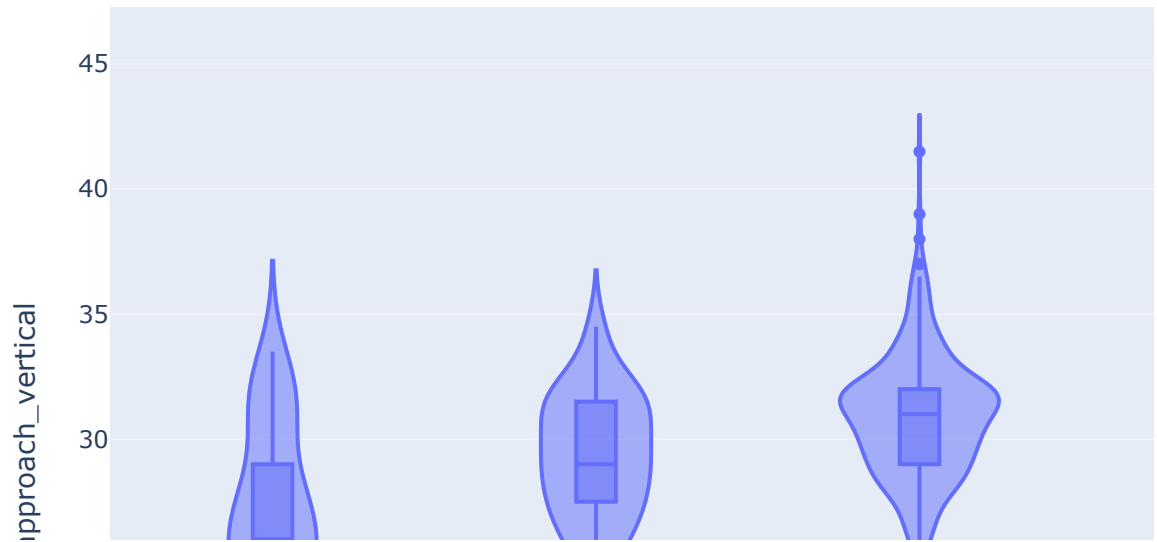
```
In [19]: fig, ax = plt.subplots(1,1)
df['approach_vertical'].plot.hist(bins=10)
ax.set_title("Histogram Of Approach Vertical")
ax.set_xlabel('Approach Vertical')
ax.set_ylabel('Count')
```

Out[19]: Text(0, 0.5, 'Count')



```
In [20]: fig = px.violin(df, y='approach_vertical', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='Approach Vertical  
1 with respect to Bam Score Rank')  
fig.show()
```

Approach Vertical with respect to Bam Score Rank

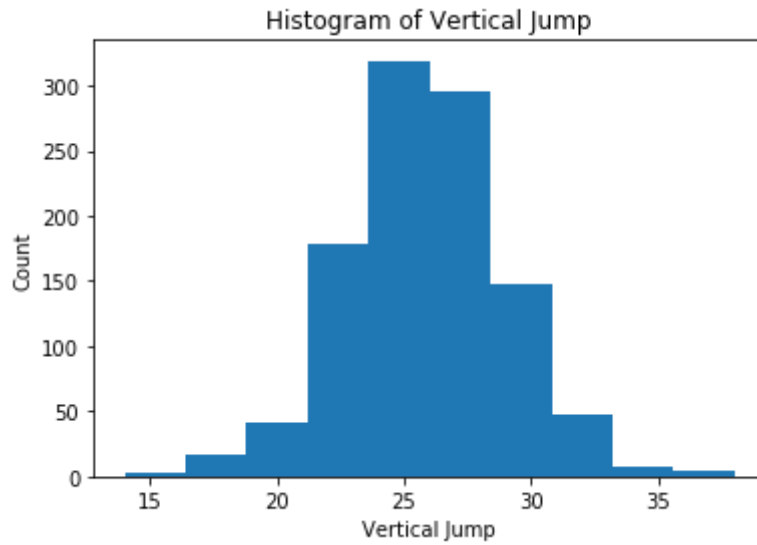


In []:

Vertical Jump

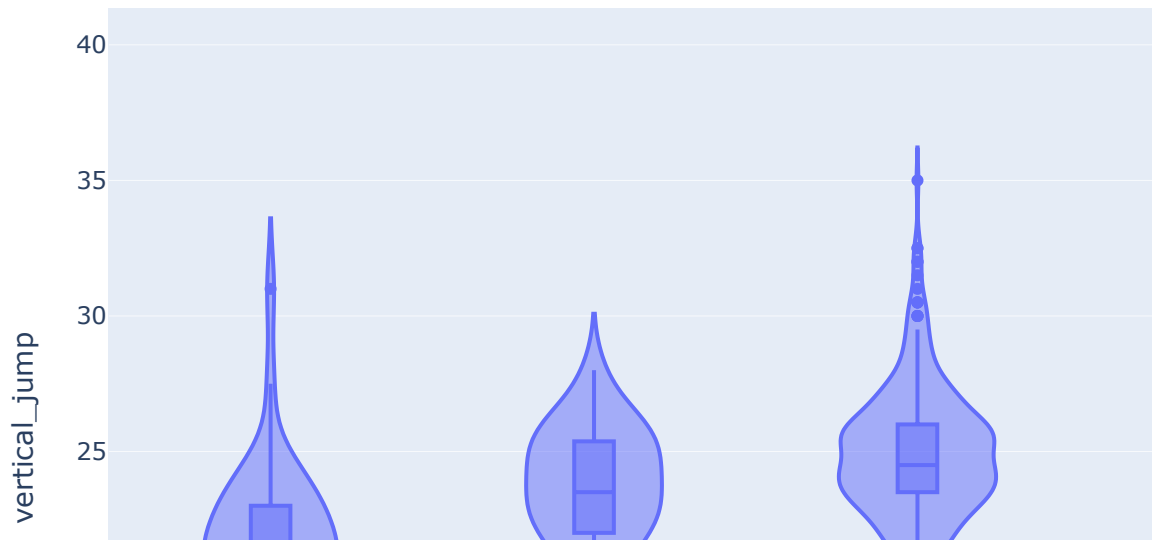
```
In [21]: fig,ax = plt.subplots(1,1)
df['vertical_jump'].plot.hist(bins=10)
ax.set_title("Histogram of Vertical Jump")
ax.set_xlabel('Vertical Jump')
ax.set_ylabel('Count')
```

Out[21]: Text(0, 0.5, 'Count')



```
In [22]: fig = px.violin(df, y='vertical_jump', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='Vertical Jump wi  
th respect to Bam Score Rank')  
fig.show()
```

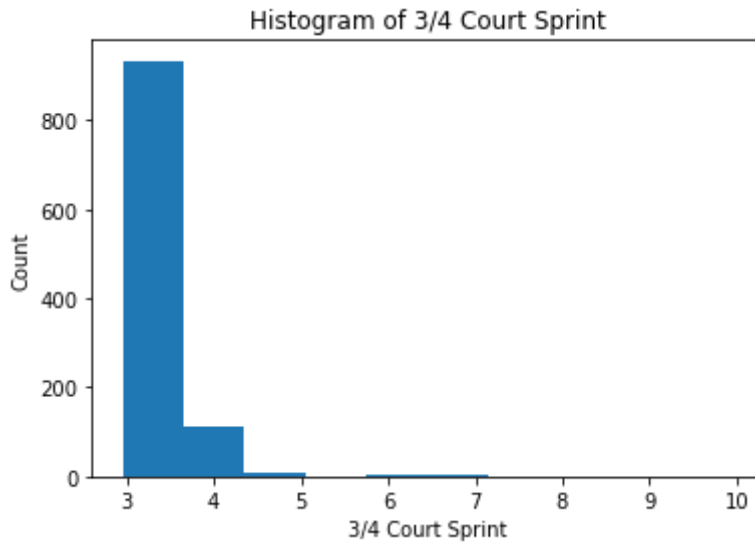
Vertical Jump with respect to Bam Score Rank



3/4 Court Sprint

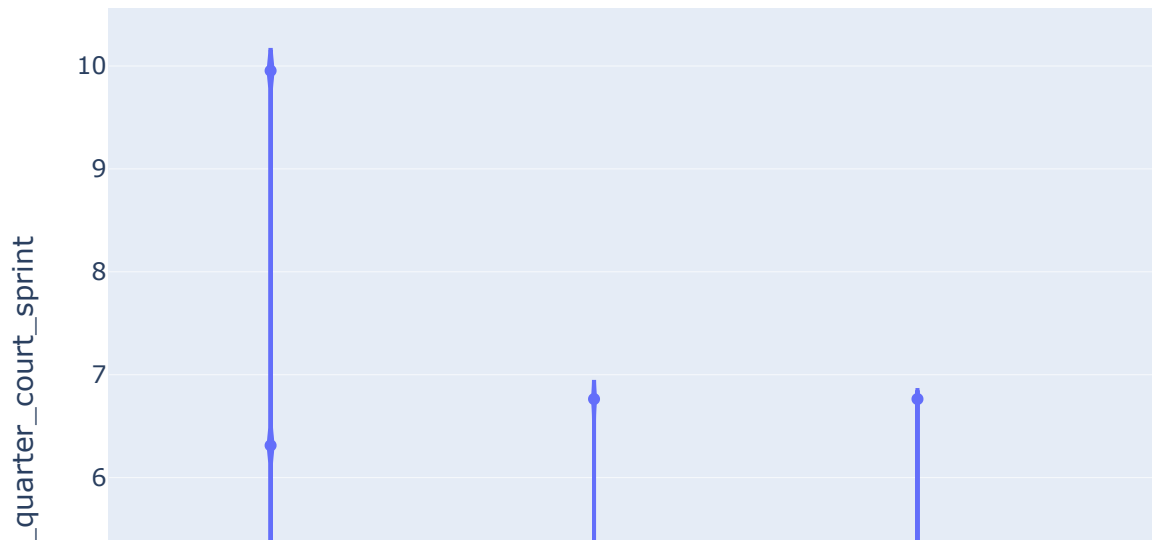

```
In [23]: fig,ax = plt.subplots(1,1)
df['three_quarter_court_sprint'].plot.hist(bins=10)
ax.set_title("Histogram of 3/4 Court Sprint")
ax.set_xlabel('3/4 Court Sprint')
ax.set_ylabel('Count')
```

Out[23]: Text(0, 0.5, 'Count')



```
In [24]: fig = px.violin(df, y='three_quarter_court_sprint', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='3/4 Court Sprint  
with respect to Bam Score Rank')  
fig.show()
```

3/4 Court Sprint with respect to Bam Score Rank

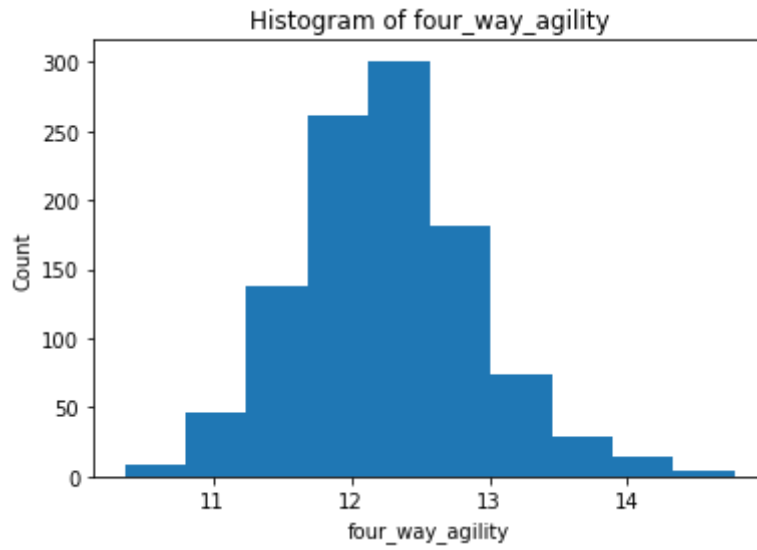


```
In [25]: # Outliers Present
```

4 Way Agility

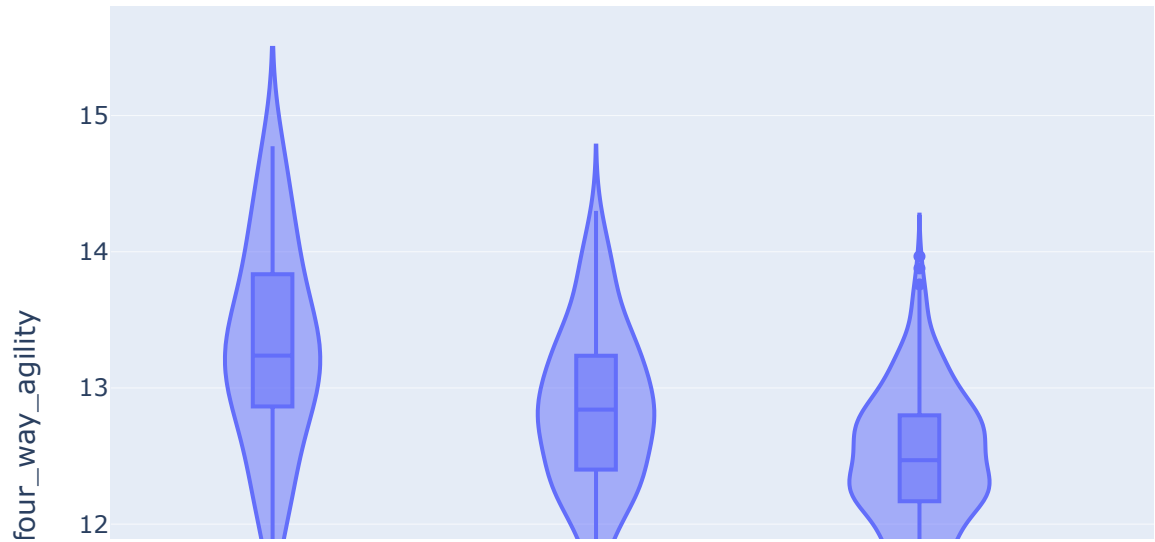
```
In [26]: fig,ax = plt.subplots(1,1)
df['four_way_agility'].plot.hist(bins=10)
ax.set_title("Histogram of four_way_agility")
ax.set_xlabel('four_way_agility')
ax.set_ylabel('Count')
```

Out[26]: Text(0, 0.5, 'Count')



```
In [27]: fig = px.violin(df, y='four_way_agility', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='four_way_agility  
with respect to Bam Score Rank')  
fig.show()
```

four_way_agility with respect to Bam Score Rank

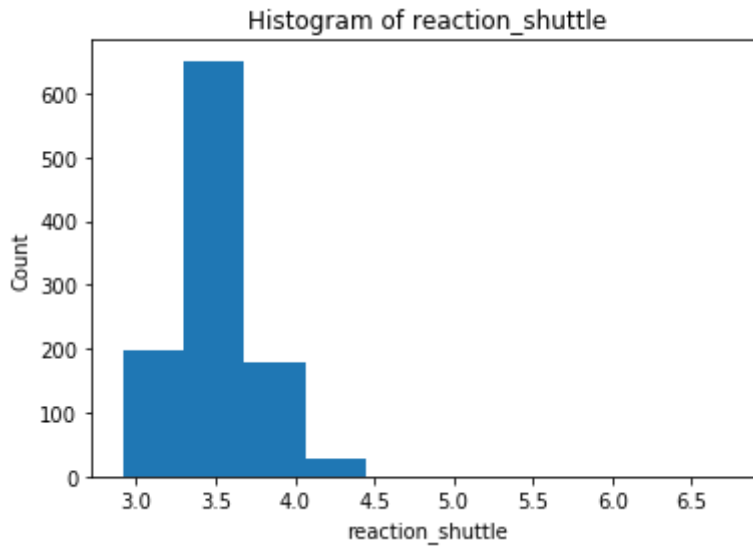


```
In [28]: # Outliers Present
```

Reaction Shuttle

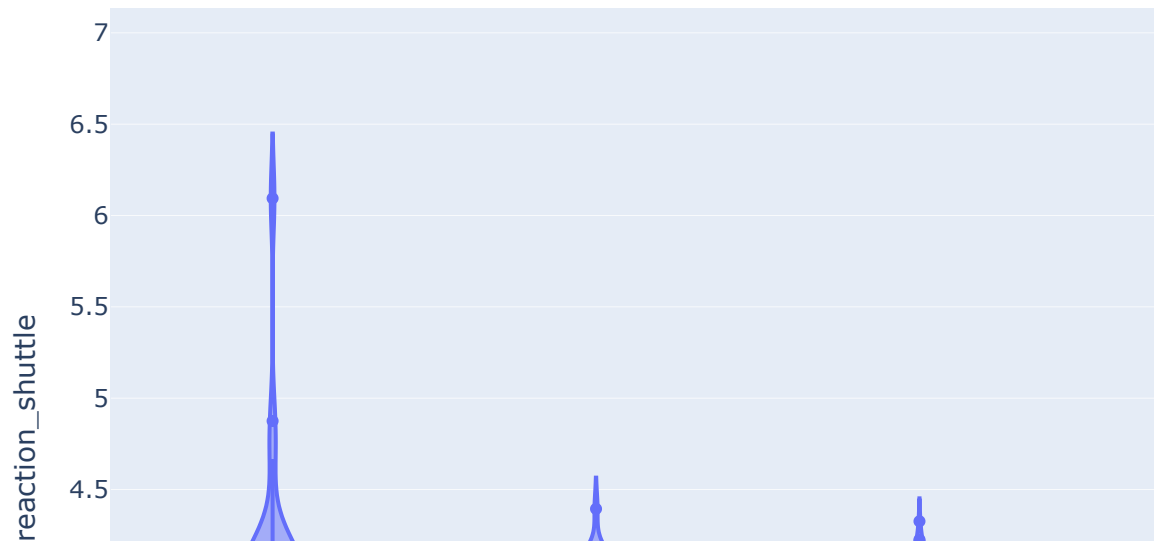
```
In [29]: fig,ax = plt.subplots(1,1)
df['reaction_shuttle'].plot.hist(bins=10)
ax.set_title("Histogram of reaction_shuttle")
ax.set_xlabel('reaction_shuttle')
ax.set_ylabel('Count')
```

Out[29]: Text(0, 0.5, 'Count')



```
In [30]: fig = px.violin(df, y='reaction_shuttle', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='Reaction Shuttle  
with respect to Bam Score Rank')  
fig.show()
```

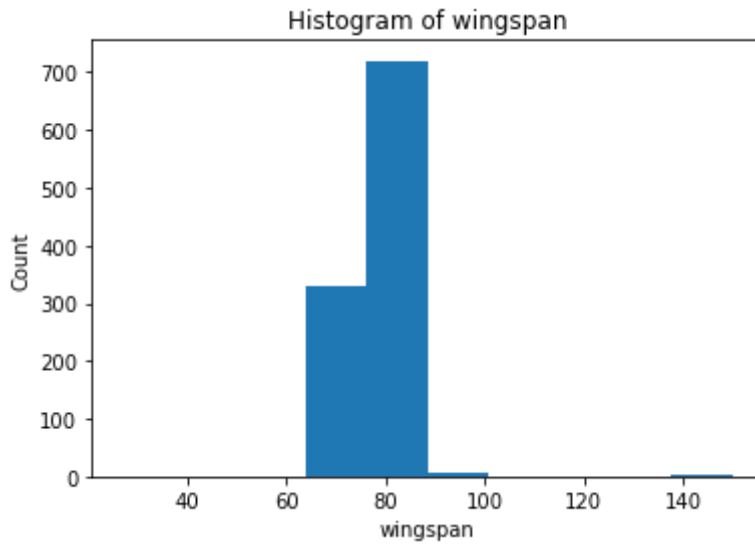
Reaction Shuttle with respect to Bam Score Rank



Wingspan

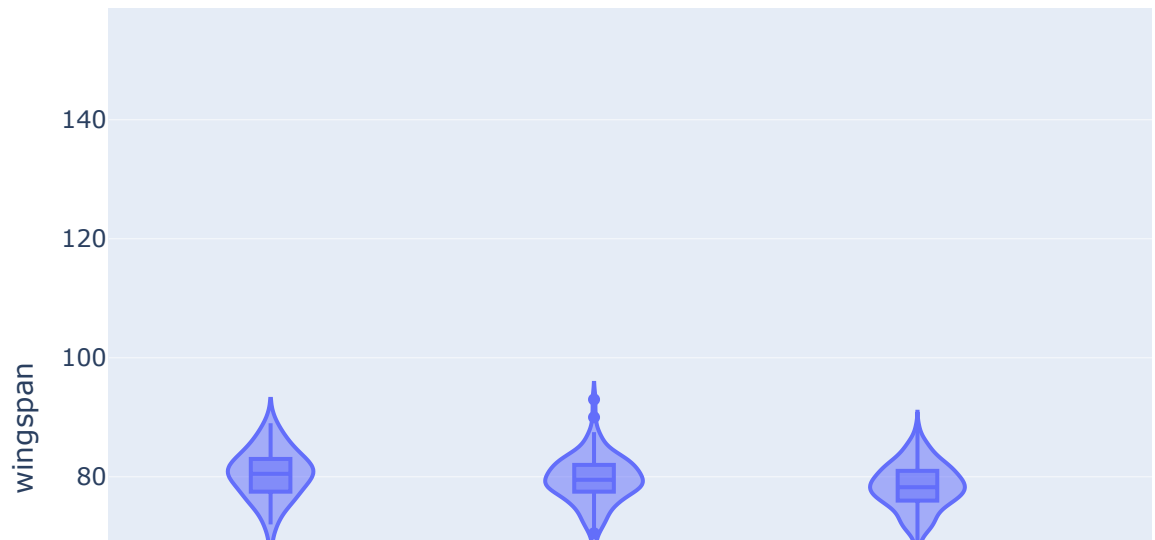
```
In [32]: fig,ax = plt.subplots(1,1)
df['wingspan'].plot.hist(bins=10)
ax.set_title("Histogram of wingspan")
ax.set_xlabel('wingspan')
ax.set_ylabel('Count')
```

Out[32]: Text(0, 0.5, 'Count')



```
In [33]: fig = px.violin(df, y='wingspan', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='wingspan with re  
spect to Bam Score Rank')  
fig.show()
```

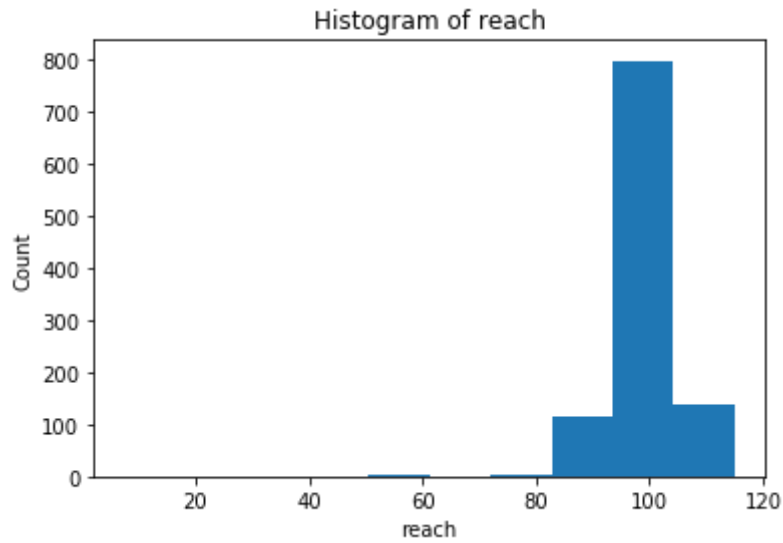
wingspan with respect to Bam Score Rank



Reach

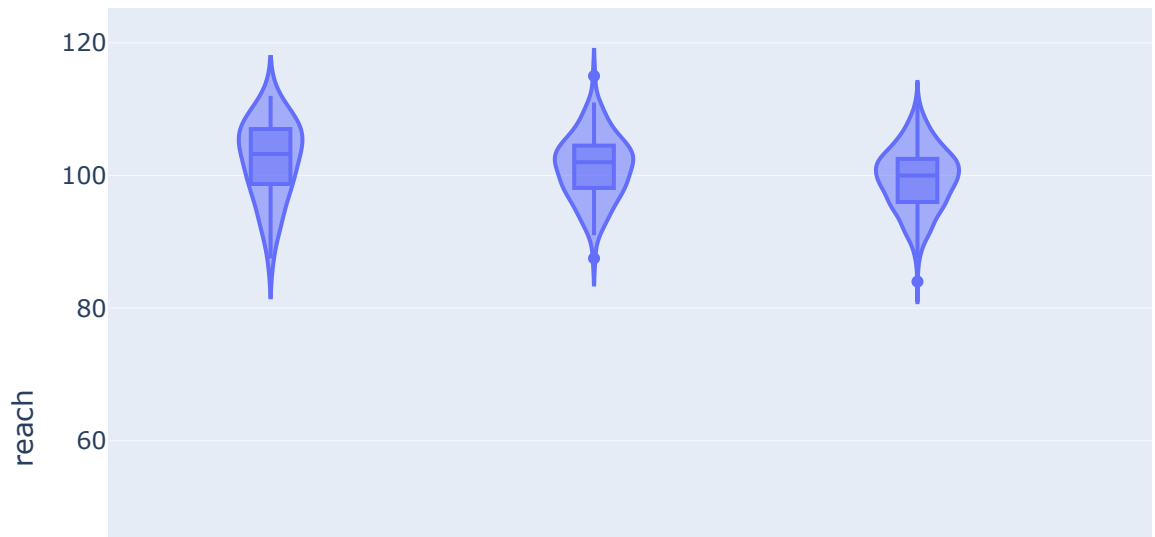

```
In [35]: fig,ax = plt.subplots(1,1)
df['reach'].plot.hist(bins=10)
ax.set_title("Histogram of reach")
ax.set_xlabel('reach')
ax.set_ylabel('Count')
```

Out[35]: Text(0, 0.5, 'Count')



```
In [36]: fig = px.violin(df, y='reach', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='reach with respe  
ct to Bam Score Rank')  
fig.show()
```

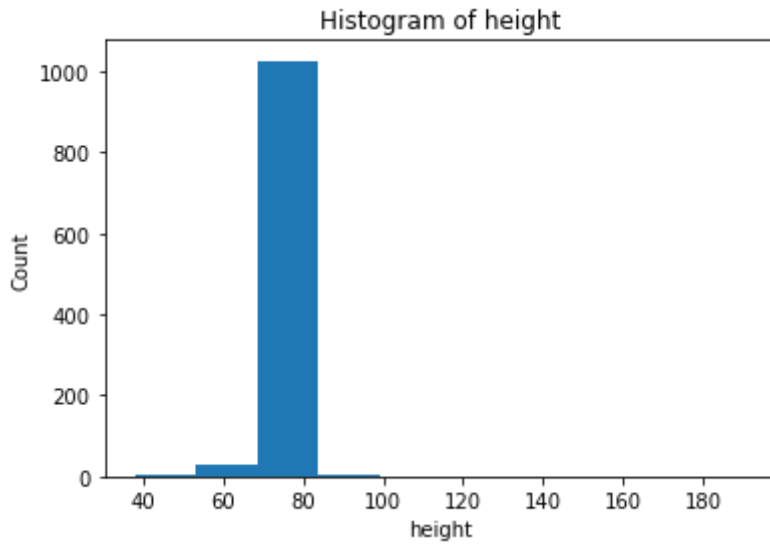
reach with respect to Bam Score Rank



Height

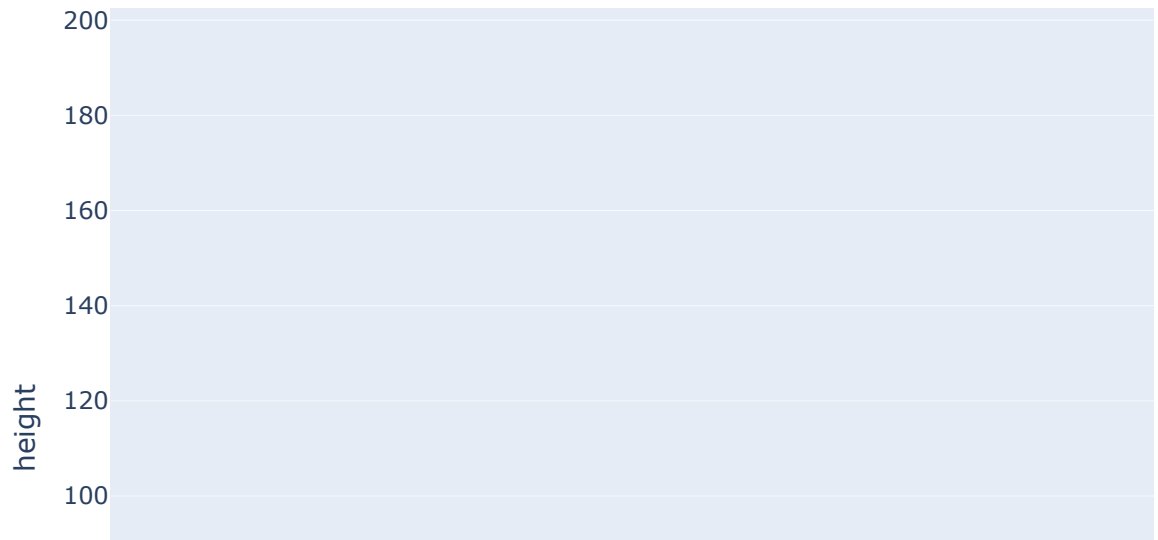
```
In [38]: fig,ax = plt.subplots(1,1)
df['height'].plot.hist(bins=10)
ax.set_title("Histogram of height")
ax.set_xlabel('height')
ax.set_ylabel('Count')
```

Out[38]: Text(0, 0.5, 'Count')



```
In [39]: fig = px.violin(df, y='height', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='height with resp  
ect to Bam Score Rank')  
fig.show()
```

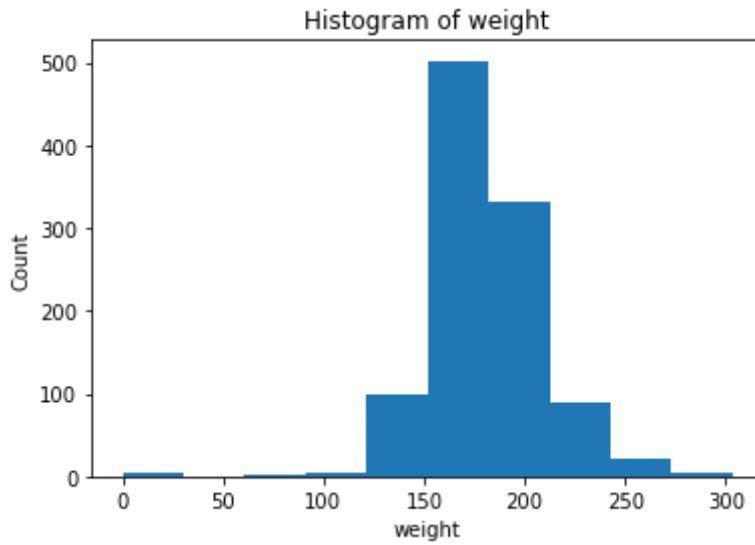
height with respect to Bam Score Rank



Weight

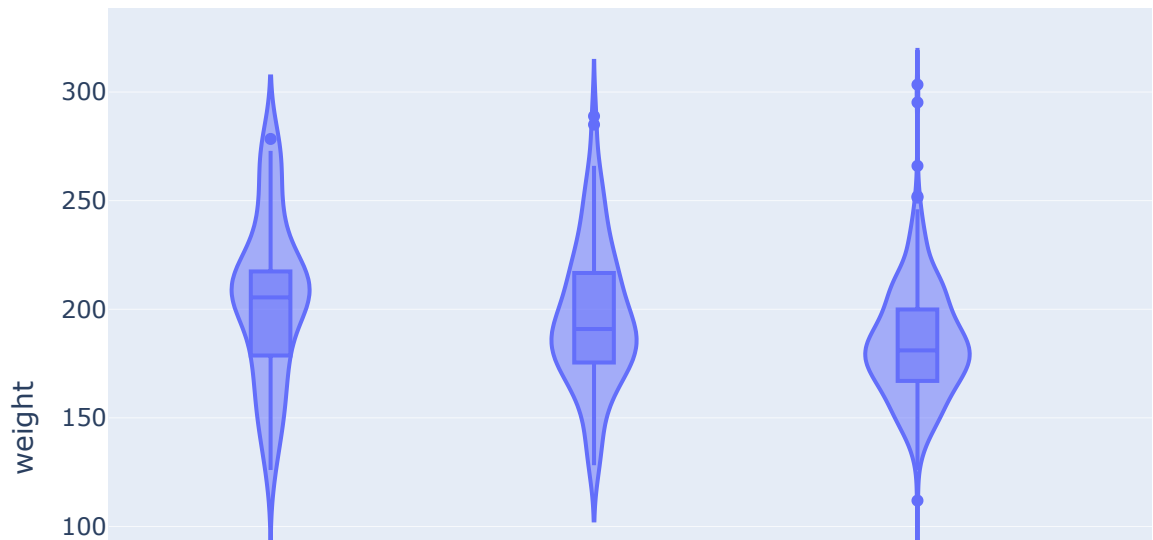
```
In [41]: fig,ax = plt.subplots(1,1)
df['weight'].plot.hist(bins=10)
ax.set_title("Histogram of weight")
ax.set_xlabel('weight')
ax.set_ylabel('Count')
```

```
Out[41]: Text(0, 0.5, 'Count')
```



```
In [42]: fig = px.violin(df, y='weight', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='weight with resp  
ect to Bam Score Rank')  
fig.show()
```

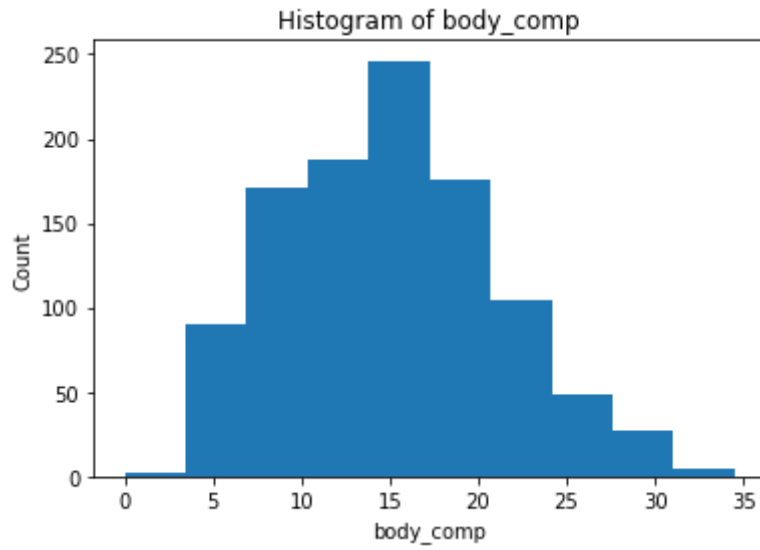
weight with respect to Bam Score Rank



Body Comp

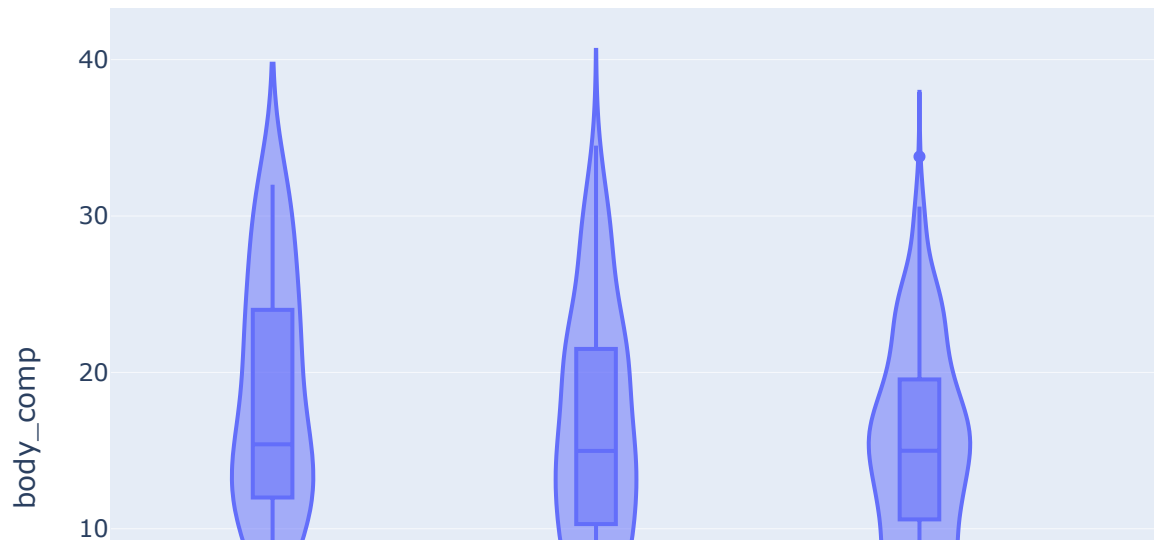
```
In [44]: fig,ax = plt.subplots(1,1)
df['body_comp'].plot.hist(bins=10)
ax.set_title("Histogram of body_comp")
ax.set_xlabel('body_comp')
ax.set_ylabel('Count')
```

Out[44]: Text(0, 0.5, 'Count')



```
In [45]: fig = px.violin(df, y='body_comp', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='body_comp with r  
respect to Bam Score Rank')  
fig.show()
```

body_comp with respect to Bam Score Rank

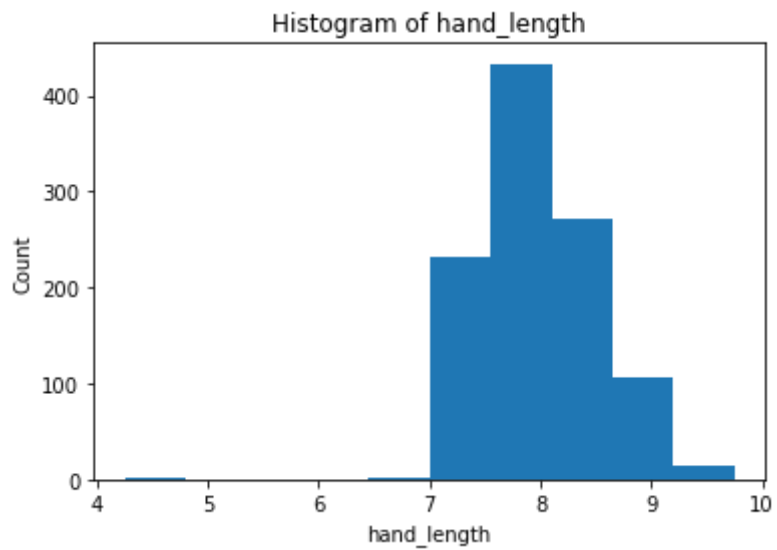


```
In [46]: # Outliers Present, has 0 values. Replace with mean after EDA
```

Hand Length

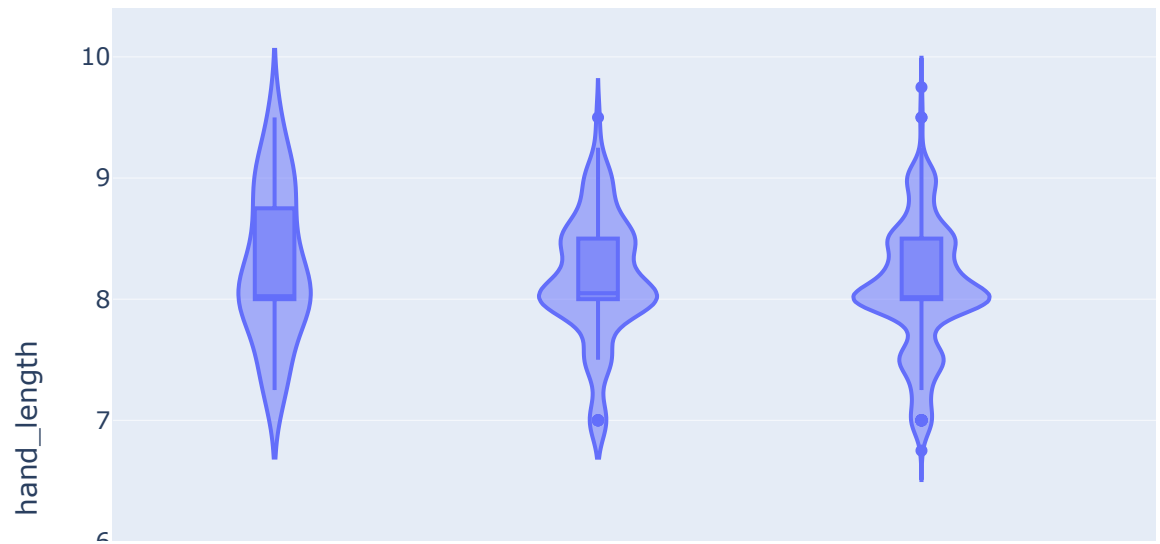

```
In [47]: fig,ax = plt.subplots(1,1)
df['hand_length'].plot.hist(bins=10)
ax.set_title("Histogram of hand_length")
ax.set_xlabel('hand_length')
ax.set_ylabel('Count')
```

Out[47]: Text(0, 0.5, 'Count')



```
In [48]: fig = px.violin(df, y='hand_length', x='bam_score_rank',  
                        box=True, hover_data=df.columns, title='hand_length with  
respect to Bam Score Rank')  
fig.show()
```

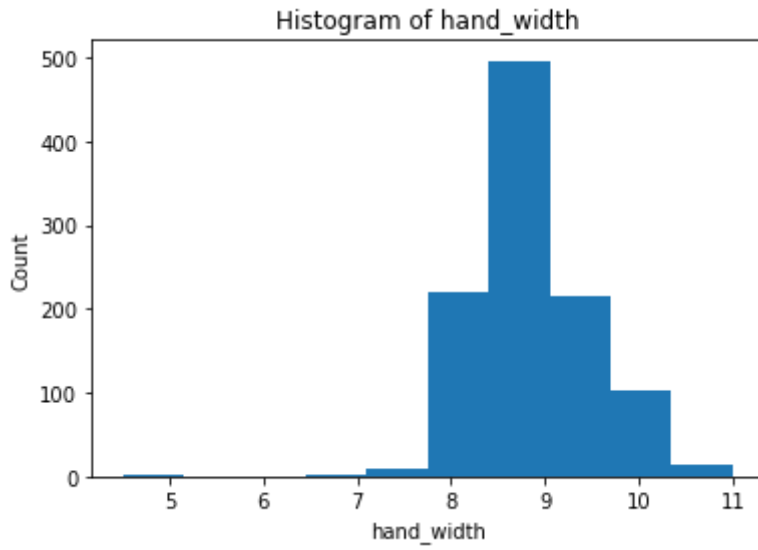
hand_length with respect to Bam Score Rank



Hand Width

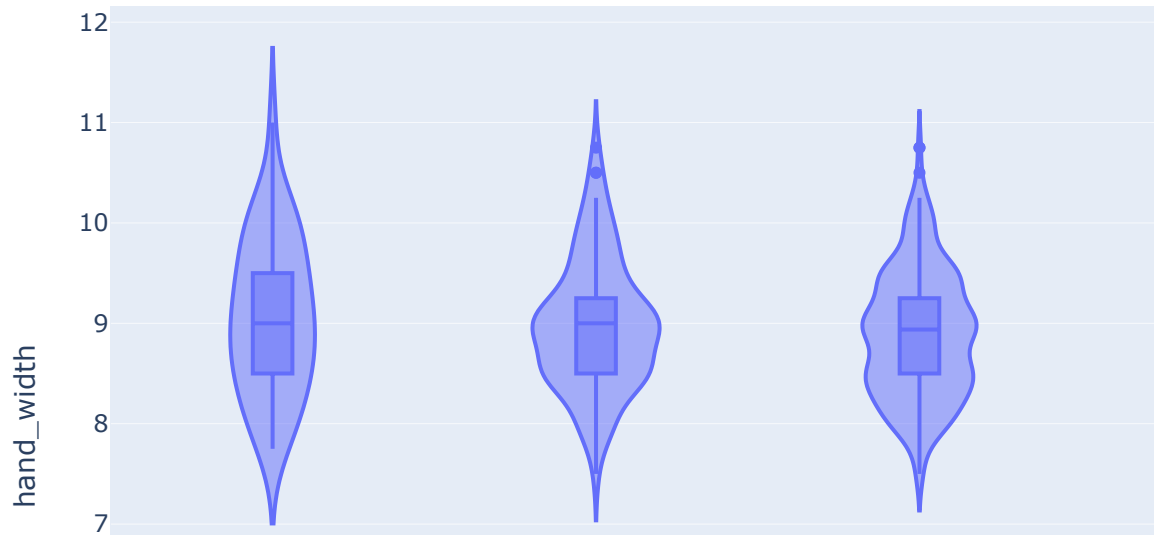
```
In [50]: fig,ax = plt.subplots(1,1)
df['hand_width'].plot.hist(bins=10)
ax.set_title("Histogram of hand_width")
ax.set_xlabel('hand_width')
ax.set_ylabel('Count')
```

```
Out[50]: Text(0, 0.5, 'Count')
```



```
In [51]: fig = px.violin(df, y='hand_width', x='bam_score_rank',
                        box=True, hover_data=df.columns, title='hand_width with
                        respect to Bam Score Rank')
fig.show()
```

hand_width with respect to Bam Score Rank



3) Find + Fix Outliers

```
In [53]: # Can redo in future this using axes example - I prefer to do my EDA step by step
## https://python.astrotech.io/matplotlib/multiple-figures.html
## https://matplotlib.org/stable/gallery/lines_bars_and_markers/simple_plot.html
```

```
In [54]: # I noticed weight and body comp had at least 1 value of 0. So I want to locate these and fix them (replace mean)
# Will do this after split
```

```
In [55]: df.describe()
```

Out[55]:

	approach_vertical	vertical_jump	three_quarter_court_sprint	four_way_agility	reaction_shuttle
count	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000
mean	31.829615	25.860157	3.467047	12.247189	3.505243
std	3.423395	3.065653	0.335502	0.652726	0.273788
min	19.000000	14.000000	2.950000	10.359000	2.914000
25%	30.000000	24.000000	3.339500	11.806000	3.348000
50%	31.829615	25.860157	3.424000	12.244000	3.492000
75%	34.000000	28.000000	3.537500	12.658000	3.634000
max	43.500000	38.000000	9.954000	14.775000	6.759000

```
In [56]: # df.loc[(df.weight == 0)]  
# df.loc[(df.body_comp == 0)]  
# to use in modeling notebook to prove why we need to replace 0 in weight and body comp
```

```
In [57]: # Replace these with mean now  
df.mean()
```

```
Out[57]: approach_vertical    31.829615  
vertical_jump                25.860157  
three_quarter_court_sprint   3.467047  
four_way_agility             12.247189  
reaction_shuttle             3.505243  
bamscore                    1890.976326  
wingspan                     78.089797  
reach                        98.714180  
height                       75.094195  
weight                       180.735820  
body_comp                    14.979496  
hand_length                  8.049381  
hand_width                   8.876189  
bam_score_rank               3.483475  
dtype: float64
```

```
In [58]: df
# Fix 0 outliers
```

Out[58]:

	approach_vertical	vertical_jump	three_quarter_court_sprint	four_way_agility	reaction_shuttle
0	33.500000	28.500000	3.376000	11.471000	3.669000
1	30.500000	21.500000	3.486000	12.114000	3.355000
2	37.000000	31.000000	3.230000	12.036000	3.562000
3	29.000000	23.000000	3.370000	12.509000	3.173000
4	31.000000	26.000000	3.389000	12.724000	3.316000
...
1054	36.000000	31.000000	3.424000	12.654000	3.635000
1055	31.500000	26.500000	3.256000	11.136000	3.343000
1056	31.829615	25.860157	3.467047	12.247189	3.505243
1057	31.829615	25.860157	3.467047	12.247189	3.505243
1058	31.829615	25.860157	3.467047	12.247189	3.505243

1059 rows × 6 columns

```
In [59]: df['body_comp'].replace(to_replace=0, value = df['body_comp'].mean(), inplace = True)
```

```
In [60]: df['weight'].replace(to_replace=0, value = df['weight'].mean(), inplace = True)
```

```
In [61]: df.drop(columns=['bam_score_rank'],inplace=True)
```

```
In [62]: df.describe()
```

Out[62]:

	approach_vertical	vertical_jump	three_quarter_court_sprint	four_way_agility	reaction_shuttle
count	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000
mean	31.829615	25.860157	3.467047	12.247189	3.505243
std	3.423395	3.065653	0.335502	0.652726	0.273781
min	19.000000	14.000000	2.950000	10.359000	2.914000
25%	30.000000	24.000000	3.339500	11.806000	3.348000
50%	31.829615	25.860157	3.424000	12.244000	3.492000
75%	34.000000	28.000000	3.537500	12.658000	3.634000
max	43.500000	38.000000	9.954000	14.775000	6.759000

4) Save Cleaned Data as csv

```
In [63]: df.to_csv('bam_data.csv', index = None)
```

```
In [64]: # Index = None - do this to not have blank index/unnamed column
```