(2) Modeling

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 plotly.express as px
import statsmodels.api as sm
import statsmodels.formula.api as smf
import random
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn import tree
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.feature_selection import SelectKBest, mutual_info_regressio
n
```

In [2]:

```
df = pd.read_csv('bam_data.csv')
df.head()
```

Out[2]:

	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	

In [3]:
```
df.describe()
```

Out[3]:

	approach_vertical	vertical_jump	three_quarter_court_sprint	four_way_agility	reaction_shuttl
count	1059.000000	1059.000000	1059.000000	1059.000000	1059.00000
mean	31.889632	25.864994	3.431342	12.222711	3.48093
std	3.027257	2.806422	0.143779	0.601871	0.20879
min	24.000000	18.000000	3.053000	10.598000	2.95700
25%	30.000000	24.000000	3.344000	11.813000	3.34850
50%	31.829615	25.860157	3.424000	12.247000	3.49300
75%	34.000000	27.500000	3.511500	12.634000	3.60200
max	40.000000	34.000000	3.831000	13.904000	4.05100

1) Split Train/Test Data

2) - normalize/standardize data with outliers [0,1] - also use min max scalers

3) - feature importance

4) - random forrest

5) - iterate model

6) - Find best model

1) Split train/test data

In [4]:
```
y = df['bamscore']
X = df.drop('bamscore',axis=1) #drop bamscorerank too once I added so no
data leakage
# y is bamscore, x is everything else
# now xtrain/split will take those two variables and create two datasets
from it

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=0, stratif
y = None)
```

In [5]:
```python
train_test_split(y, shuffle=False)
```

Out[5]:
```
[0      2003.0
 1      1865.0
 2      2005.0
 3      1902.0
 4      1903.0
         ...
 789    1916.0
 790    1852.0
 791    1593.0
 792    1814.0
 793    2114.0
 Name: bamscore, Length: 794, dtype: float64, 794     1843.000000
 795     1968.000000
 796     1965.000000
 797     1950.000000
 798     1902.000000
            ...
 1054    1917.000000
 1055    2029.000000
 1056    1890.976326
 1057    1890.976326
 1058    1890.976326
 Name: bamscore, Length: 265, dtype: float64]
```

2) Normalize + Scale Data to adjust for outliers

- going to use min/max normalization to make all points between 0-1

- will help us understand our data better especially with big data - to avoid data leakage

- meaning, make data all within 0-1 range. 5.5 on 0-10 scale would be .55 on normalized scale

Why are we normalizing data?

to make data cleaner, sometimes gets messed up on backend and removes unstructured data

In [6]:
```python
X_train_ids_reg = X_train.copy()
X_test_ids_reg = X_test.copy()
y_train_ids_reg = y_train.copy()
y_test_ids_reg = y_test.copy()
```

In [7]:
```python
y_train=(y_train-y_train.min())/(y_train.max()-y_train.min())
y_test=(y_test-y_test.min())/(y_test.max()-y_test.min())
# we had to calculate the y train and y test manually because the functi
on would not accept pandas series
```

In [8]:
```
normalize(X_train, copy = False)
normalize(X_test, copy = False)
df_train = pd.DataFrame(data = np.concatenate((X_train.values, y_train.v
alues.reshape(-1,1)),axis = 1)
                        , columns = list(X_train.columns) + ["bamscore"
])
```

In [9]:
```
df_train
# df_train is same as x_train dataset with bamscores included as y
# statsmodels issue so we had to include bamscore (full data set)
```

Out[9]:

	approach_vertical	vertical_jump	three_quarter_court_sprint	four_way_agility	reaction_shuttle
0	0.127644	0.110904	0.015016	0.049245	0.015585
1	0.139731	0.113526	0.015712	0.052135	0.014799
2	0.173806	0.130354	0.016736	0.060029	0.016363
3	0.108274	0.098607	0.013093	0.046635	0.012846
4	0.159682	0.126732	0.018128	0.065885	0.017337
...	...	...	...	...	...
736	0.149692	0.128307	0.015715	0.054516	0.016851
737	0.181446	0.144698	0.015926	0.056106	0.015287
738	0.127032	0.098270	0.016855	0.060357	0.015843
739	0.143284	0.115483	0.013507	0.050077	0.013610
740	0.156546	0.115666	0.015418	0.051942	0.015319

741 rows × 13 columns

In [10]:
```
# https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_s
caling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py
```

3) Feature Importance + Feature Selection

Want to get rough idea of what features are most important in pred bamscore

SelectKBest

In [11]:
```
X_train
```

Out[11]:

	approach_vertical	vertical_jump	three_quarter_court_sprint	four_way_agility	reaction_shuttle
899	0.127644	0.110904	0.015016	0.049245	0.01558
635	0.139731	0.113526	0.015712	0.052135	0.01479
310	0.173806	0.130354	0.016736	0.060029	0.01636
961	0.108274	0.098607	0.013093	0.046635	0.01284
723	0.159682	0.126732	0.018128	0.065885	0.01733
...	...	...	...	...	..
1033	0.149692	0.128307	0.015715	0.054516	0.016851
763	0.181446	0.144698	0.015926	0.056106	0.01528
835	0.127032	0.098270	0.016855	0.060357	0.01584
559	0.143284	0.115483	0.013507	0.050077	0.01361
684	0.156546	0.115666	0.015418	0.051942	0.01531

741 rows × 12 columns

In [12]:
```
# want to fit and transform the model on training x and y data
```

Picking 7 best features

Picking 7 out of 12 best factors and going from there. Trying to focus on factors that are actually trainable

In [13]:
```
#skb = SelectKBest(f_regression, k=8)
skb = SelectKBest(score_func=mutual_info_regression, k=7)
fit = skb.fit(X_train, y_train)
features = fit.transform(X_train)

# Two of the variables don't matter that much but need to keep in data.
 k=10 took them out
# mutual_info_regression is used instead of f_reg because we have Mutual
information for a continuous target.
# f_regression is for continuous to categorical, we want continuous to c
ontinuous
```

This is the data for the 7 most important features

Save them as new data frame to model called skb_7

In [14]:
```
mask = fit.get_support()
# get_support - get mask or integers index of selected features
# true bools = yes part of 7/12 best
# false bool = no , not part of 7 best
skb_7 = X_train[X_train.columns[mask]]
skb_7
```

Out[14]:

	approach_vertical	vertical_jump	three_quarter_court_sprint	four_way_agility	reaction_shuttle
899	0.127644	0.110904	0.015016	0.049245	0.01558
635	0.139731	0.113526	0.015712	0.052135	0.01479
310	0.173806	0.130354	0.016736	0.060029	0.01636
961	0.108274	0.098607	0.013093	0.046635	0.01284
723	0.159682	0.126732	0.018128	0.065885	0.01733
...	...	...	...	...	..
1033	0.149692	0.128307	0.015715	0.054516	0.016851
763	0.181446	0.144698	0.015926	0.056106	0.01528
835	0.127032	0.098270	0.016855	0.060357	0.01584
559	0.143284	0.115483	0.013507	0.050077	0.01361
684	0.156546	0.115666	0.015418	0.051942	0.01531

741 rows × 7 columns

So far, Our 7 best features are:

- approach vertical

- vertical jump

- 3/4 court sprint

- four way agility

- reaction shuttle

- hand length

- hand width

Now that we have a good sense of 7 important features, lets put this to the side and try an OLS Regression

I modeled all 12 factors to have a starting point and what the model is saying

4) OLS Regression (Ordinary Least Squares)

Use this because it's the most simple baseline model and creates a best fit line that has minimized distance between points and in best fit line

Model gives best approximate of true population regression line.

The principle of OLS is to minimize the square of errors

In [15]:
```
# Look at all columns and see what coeff is saying
## larger coef means it stronger contribution for that feature to bamsco
re
```

r2 tells us that 63.7% of the variance in bamscore is explained in the variance of those 12 factors

12 Factors

Ran for all 12 and focused on picking 5 with large coefficients to find trainable characteristics

In [16]:

```
formula = "bamscore ~ approach_vertical + vertical_jump + three_quarter_
court_sprint + four_way_agility + reaction_shuttle + wingspan + reach +
 height + weight + body_comp + hand_length + hand_width"
lm = smf.ols(formula = formula, data = df_train).fit()
print(lm.summary())
print("After selecting best 12 features:", features.shape)

# Ran for all 12 and focused on picking on large coefficients to find tr
ainable characteristics
# Chose my 5 features because - 1) coeff are large and 2) trainable char
acteristics
# What does it tell us?
# r2 - Most of the variance (64%) in bamscore can be predicted through t
he factors given
# coeff - largest coeff means it infact effects it
# std err - afte rmodel fit, how much error is left? Essentially measure
of how good each category fit the line
# Lower standard error is best (normally)
# For example, Reaction shuttle has a high standard error but high coeff
which tells us it's a good indication.
# not much variance
# t stat tells us "how different the means are"
# p value tells us - can't fit data well enough to for sure say 3/4 cour
t sprint, wingspan, and hand length can be good predictors
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:               bamscore   R-squared:
0.637
Model:                            OLS   Adj. R-squared:
0.631
Method:                 Least Squares   F-statistic:
106.4
Date:                Wed, 27 Apr 2022   Prob (F-statistic):          4.
44e-151
Time:                        13:24:47   Log-Likelihood:
563.70
No. Observations:                 741   AIC:
-1101.
Df Residuals:                     728   BIC:
-1041.
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
=======================
                                 coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
-----------------------
Intercept                     -6.6399      2.548     -2.606      0.009
-11.642      -1.638
approach_vertical              6.2192      0.580     10.729      0.000
5.081       7.357
vertical_jump                  4.8804      0.598      8.155      0.000
3.705       6.055
three_quarter_court_sprint     5.3725      6.835      0.786      0.432
-8.046      18.791
four_way_agility             -17.1495      1.846     -9.290      0.000
-20.774     -13.525
reaction_shuttle             -51.6624      5.078    -10.173      0.000
-61.632     -41.693
wingspan                       1.6944      1.106      1.532      0.126
-0.477       3.865
reach                          3.7804      1.371      2.758      0.006
1.089       6.471
height                         3.6030      1.101      3.272      0.001
1.442       5.765
weight                         5.1150      1.934      2.645      0.008
1.318       8.912
body_comp                      0.9108      0.257      3.544      0.000
0.406       1.415
hand_length                   -1.3277      3.046     -0.436      0.663
-7.307       4.652
hand_width                     4.9526      2.362      2.097      0.036
0.316       9.589
==============================================================================
=======
Omnibus:                       43.443   Durbin-Watson:
1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
158.504
```

```
Skew:                           0.044   Prob(JB):
3.81e-35
Kurtosis:                       5.264   Cond. No.
2.41e+03
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 2.41e+03. This might indicate that t
here are
strong multicollinearity or other numerical problems.
After selecting best 12 features: (741, 7)
```

In [17]:

```
# Reaction shuttle and four way agility are very important because magni
tude is a lot bigger than coef for others
# If Condition Number is too high I'll run into problems in matrix becau
se program may not catch small errors
```

In [18]:

```
#### Quick notes + Table Results Definitions

### df = samplesize - # of variables +1 --> number of independent observ
ations
### constant is intercept in regression line and tells us avg value of o
mmited variables and noise in model
### coeff term = slope aka rise over run, if x increases by 1 , and coef
f is .7, y increased by .7
### standard error = std
### Standard error shows the sampling variability of these parameters.σ^
2 is equal to residual sum squares
#### remember *σ2 in first param and its the numerator in second param

### H0  : B2  = 0       ( variable X has no influence on Y)
### Ha  : B2  ≠ 0      (X has significant impact on Y)
### b1  ~ N(B1, σb12) B1 is true mean of b1
### - b1 is param
### b2   ~ N(B2 , σb22) B2 is true mean of b2
### - b2 is param

### p value - reject null <0.05 or fail to reject if >0.05. if 0 t is pr
obably high

### R2 is the coefficient of determination that tells us that
### how much percentage variation independent variable can be explained
 by independent variable.

# F stat - prob f stat > f stat given = 0 means reject null

### https://www.geeksforgeeks.org/interpreting-the-results-of-linear-reg
ression-using-ols-summary/
```

Iteration:

- Going to choose what I think are the 5 most important features based on watching tests in real life

- Also picking the 5 features I think are most important from pairplot in EDA

Picking 5 Factors

2 of our 7 important factors are not trainable = hand_width and hand_length

Let's take them out

From our original 7, we picked these 5 because they had bigger coeff and are actually trainable

In [19]:

```
formula = "bamscore ~ approach_vertical + vertical_jump + four_way_agili
ty + reaction_shuttle + three_quarter_court_sprint"
lm = smf.ols(formula = formula, data = df_train).fit()
print(lm.summary())
#https://www.datatechnotes.com/2021/02/seleckbest-feature-selection-exam
ple-in-python.html
# Now that we have eliminated a few factors, the contribution from 3/4 c
ourt sprint is now statistically sig
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:               bamscore   R-squared:
0.611
Model:                            OLS   Adj. R-squared:
0.608
Method:                 Least Squares   F-statistic:
230.8
Date:                Wed, 27 Apr 2022   Prob (F-statistic):          5.
49e-148
Time:                        13:24:47   Log-Likelihood:
538.18
No. Observations:                 741   AIC:
-1064.
Df Residuals:                     735   BIC:
-1037.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
=======================
                                 coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
-----------------------
Intercept                      0.5175      0.054      9.585      0.000
0.411       0.623
approach_vertical              5.3676      0.484     11.090      0.000
4.417       6.318
vertical_jump                  4.6855      0.532      8.801      0.000
3.640       5.731
four_way_agility             -15.1359      1.778     -8.513      0.000
-18.627     -11.645
reaction_shuttle             -50.3648      5.054     -9.964      0.000
-60.288     -40.442
three_quarter_court_sprint    19.5406      6.144      3.181      0.002
7.479      31.602
==============================================================================
=======
Omnibus:                       37.151   Durbin-Watson:
1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
118.152
Skew:                           0.055   Prob(JB):
2.21e-26
Kurtosis:                       4.953   Cond. No.
1.55e+03
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.55e+03. This might indicate that t
here are
strong multicollinearity or other numerical problems.
```

Using just those 5, 61.1% of variance in bamscore is explained from those 5 factors

Very close to value when we used all 12

5) GLSAR

GLSAR is "experimental", may not make sense, but wanted to try it

In linear reg - typically y values come from normal distribution - means related to predictor

In GLSAR - typically y values come from exponential family distribution - means related to some function of the mean related to pred

Main difference is the method the coefficients (betas) are found

Linear use Least Squares or max likelihood in normal distribution

In generalized models, have to only use maximum likelihood

In [20]:
```
model = sm.GLSAR(y_train, X_train, rho=2)
for i in range(6):
    results = model.fit()
    print("AR coefficients: {0}".format(model.rho))
    rho, sigma = sm.regression.yule_walker(results.resid, order=model.or
der)
    model = sm.GLSAR(y_train, X_train, rho)

lm = model.fit()
print(lm.summary())
# (int) rho is order of autoregressive covariance
# covariance - extent to which 2 variables increase or decrease in tande
m to each other
### coeff in linear reg are linear
```

```
AR coefficients: [0. 0.]
AR coefficients: [0.00779657 0.01681878]
AR coefficients: [0.00797418 0.01742036]
AR coefficients: [0.00797655 0.0174426 ]
AR coefficients: [0.00797648 0.01744345]
AR coefficients: [0.00797647 0.01744349]
                                GLSAR Regression Results
========================================================================
================
Dep. Variable:                bamscore   R-squared (uncentered):
0.955
Model:                           GLSAR   Adj. R-squared (uncentered):
0.955
Method:                  Least Squares   F-statistic:
1296.
Date:                 Wed, 27 Apr 2022   Prob (F-statistic):
0.00
Time:                         13:24:47   Log-Likelihood:
560.08
No. Observations:                  739   AIC:
-1096.
Df Residuals:                      727   BIC:
-1041.
Df Model:                           12
Covariance Type:             nonrobust
========================================================================
=======================
                                coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------
-----------------------
approach_vertical             5.4270      0.486     11.168      0.000
4.473       6.381
vertical_jump                 4.1267      0.528      7.813      0.000
3.090       5.164
three_quarter_court_sprint    2.5040      6.762      0.370      0.711
-10.771      15.779
four_way_agility            -17.7259      1.840     -9.631      0.000
-21.339     -14.113
reaction_shuttle            -52.1421      5.093    -10.237      0.000
-62.141     -42.143
wingspan                     -0.5815      0.723     -0.804      0.422
-2.001       0.838
reach                         1.0247      0.811      1.264      0.207
-0.567       2.616
height                        1.6737      0.822      2.036      0.042
0.060       3.288
weight                        0.0769      0.066      1.164      0.245
-0.053       0.206
body_comp                     0.4737      0.200      2.363      0.018
0.080       0.867
hand_length                  -2.1260      3.068     -0.693      0.489
-8.150       3.898
hand_width                    5.2285      2.367      2.209      0.027
0.582       9.875
========================================================================
=======
```

```
Omnibus:                       45.433   Durbin-Watson:
1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
169.727
Skew:                           0.078   Prob(JB):
1.39e-37
Kurtosis:                       5.343   Cond. No.
1.64e+03
=======================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.64e+03. This might indicate that t
here are
strong multicollinearity or other numerical problems.
```

In [21]:
```
ypred = lm.predict(X_test)
r2_score(y_test, ypred)
```

Out[21]:
```
0.5816926699396184
```

In [22]:
```
#https://www.statsmodels.org/dev/generated/statsmodels.regression.linear
_model.GLSAR.html
```

Insight:

GLSAR Model is very over fitted with r2 score of .9555

r2 on test data and overfitted

Tried less iterations on loop and higher rho and neither made a difference after 1 iteration

Going to move and try random forest model since GLSAR model is very overfitted

6) Random Forest

```
In [23]: # Random Forest
## Going to try random forrest model because GLSAR model and had bad ove
rfitting or the same results as OLS model
## Random forrest corrects for high variance/overfitting by using many d
ecision trees

#Steps:
# -create bootstrapped dataset with subset of variables
# -fit decision tree
# -repeat and tally predictions
```

```
In [24]: # create regressor object
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
cols = ["vertical_jump", "approach_vertical", "four_way_agility", "react
ion_shuttle", "three_quarter_court_sprint"]

# fit the regressor with x and y data
rf = regressor.fit(X_train[cols], y_train)
rf.score(X_test[cols], y_test)
# score if fit and test on full data frame
```

```
Out[24]: 0.6005390346228129
```

```
In [25]: # Messed around to see if anthros gave us a good r2 score and it did not
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
cols = ["wingspan", "reach", "height", "weight", "body_comp", "hand_leng
th", "hand_width"]
# fit the regressor with x and y data
rf = regressor.fit(X_train[cols], y_train)
rf.score(X_test[cols], y_test)
```

```
Out[25]: 0.19881330115476226
```

7) MLP Regressor = Multilayer perceptron

Wanted to try a new random model and see if I could get a better r2 score.

I wanted to try this because I had never dealt with hidden layers with different weights before

I put in one input-> it trains the weights within the multiple layers in the hidden data, goes through hidden data many times as it learns and spits out -> output - very complicated on backend

```
In [26]: regr = MLPRegressor(random_state=1, max_iter=500).fit(X_train, y_train)
regr.score(X_test, y_test)
```

```
Out[26]: -0.0022790688232969813
```

Add Learning Rate Init

In [27]:
```
# Mess around with hyper params
# add learning rate init - default =.001
regr = MLPRegressor(random_state=1, max_iter=500, learning_rate_init=.00
1).fit(X_train, y_train)
regr.score(X_test, y_test)
```

Out[27]: -0.0022790688232969813

In [28]:
```
# increase learning rate init to.005
regr = MLPRegressor(random_state=1, max_iter=500, learning_rate_init=.00
5).fit(X_train, y_train)
regr.score(X_test, y_test)
```

Out[28]: 0.39752644122382885

In [29]:
```
# increase learning rate init to.009 was the best after trying range .00
1-.01
regr = MLPRegressor(random_state=1, max_iter=500, learning_rate_init=.00
9).fit(X_train, y_train)
regr.score(X_test, y_test)
```

Out[29]: 0.4381936060903642

Add Solver

In [30]:
```
# Noticed "solver", default is adam and used for big data sets, so chang
e solver for a smaller data set we have
regr = MLPRegressor(random_state=1, max_iter=500, learning_rate_init=.00
9, solver='adam').fit(X_train, y_train)
regr.score(X_test, y_test)
```

Out[30]: 0.4381936060903642

In [31]:
```
regr = MLPRegressor(random_state=1, max_iter=500, learning_rate_init=.00
9, solver='lbfgs').fit(X_train, y_train)
regr.score(X_test, y_test)
```

Out[31]: 0.5506230429046819

Add maximum fun

In [32]:
```
##### Since we used solver = lbfgs, we can use maximum fun calls.
##### Default = 15000 calls
regr = MLPRegressor(random_state=1, max_iter=500, learning_rate_init=.00
9, solver='lbfgs', max_fun=15000).fit(X_train, y_train)
regr.score(X_test, y_test)
```

Out[32]: 0.5506230429046819

```
In [33]: ##### Increasing max_fun = no change
         ##### Decreasing max_fun <150 decreased r2
```

8) Summary of 4 Models

This model gave us this, this ....

Best model

1) OLS = .612

2) Random Forest = .6

In [34]:

```
formula = "bamscore ~ approach_vertical + vertical_jump + four_way_agili
ty + reaction_shuttle + three_quarter_court_sprint"
lm = smf.ols(formula = formula, data = df_train).fit()
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:               bamscore   R-squared:
0.611
Model:                            OLS   Adj. R-squared:
0.608
Method:                 Least Squares   F-statistic:
230.8
Date:                Wed, 27 Apr 2022   Prob (F-statistic):          5.
49e-148
Time:                        13:24:49   Log-Likelihood:
538.18
No. Observations:                 741   AIC:
-1064.
Df Residuals:                     735   BIC:
-1037.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
=======================
                               coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
-----------------------
Intercept                    0.5175      0.054      9.585      0.000
0.411       0.623
approach_vertical            5.3676      0.484     11.090      0.000
4.417       6.318
vertical_jump                4.6855      0.532      8.801      0.000
3.640       5.731
four_way_agility           -15.1359      1.778     -8.513      0.000
-18.627     -11.645
reaction_shuttle           -50.3648      5.054     -9.964      0.000
-60.288     -40.442
three_quarter_court_sprint  19.5406      6.144      3.181      0.002
7.479      31.602
==============================================================================
=======
Omnibus:                       37.151   Durbin-Watson:
1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
118.152
Skew:                           0.055   Prob(JB):
2.21e-26
Kurtosis:                       4.953   Cond. No.
1.55e+03
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.55e+03. This might indicate that t
here are
strong multicollinearity or other numerical problems.
```

Based on our best model, if someone wants to best improve their bam score they should focus on improving:

1) reaction shuttle

2) 3/4 court sprint

3) four way agility

Also wanted to try and predict bamscore using our factors

Randomly select 5 people from our training data

In [35]:
```
idx = random.choices(range(X_train.shape[0]), k=5)
print(idx)
```

```
[436, 1, 526, 261, 654]
```

predict bamscore (normalized score)

In [36]:
```
y_pred = lm.predict(X_train.iloc[idx,:])
print(y_pred)
```

```
114    0.386732
635    0.571967
839    0.439271
475    0.404971
868    0.422371
dtype: float64
```

Unnormalize the predicted bamscore

In [37]:
```
X_in = X_train_ids_reg.iloc[idx,:]
```

In [38]:
```
y_pred_reg = y_pred*(y_train_ids_reg.max()-y_train_ids_reg.min()) + y_train_ids_reg.min()
y_pred_reg
```

Out[38]:
```
114    1814.016336
635    1936.641852
839    1848.797340
475    1826.090597
868    1837.609856
dtype: float64
```

Show table of test scores for our 5 players, bamscore above ^

In [39]:
```
X_in
```

Out[39]:

	approach_vertical	vertical_jump	three_quarter_court_sprint	four_way_agility	reaction_shuttle
114	29.000000	24.000000	3.464	12.104	3.631
635	31.829615	25.860157	3.579	11.876	3.371
839	29.500000	24.000000	3.620	11.751	3.616
475	30.000000	23.000000	3.664	12.486	3.471
868	30.500000	23.500000	3.403	11.839	3.607

We can now say improving at these 5 important factors will increase bamscore.

In the future I really wan't to be able to say "increasing you speed by .1 second in a test can increase bam score by a certain numerical amount"