(2) Modeling

In [1]:
```python
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
```

In [2]:
```python
df = pd.read_csv('bam_data.csv')
df.head()
```

Out[2]:

	approach_vertical	vertical_jump	three_quarter_court_sprint	four_way_agility	reaction_shuttle	k
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]:
```python
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	31.829615	25.860157	3.467047	12.247189	3.50524
std	3.423395	3.065653	0.335502	0.652726	0.27378
min	19.000000	14.000000	2.950000	10.359000	2.91400
25%	30.000000	24.000000	3.339500	11.806000	3.34800
50%	31.829615	25.860157	3.424000	12.244000	3.49200
75%	34.000000	28.000000	3.537500	12.658000	3.63400
max	43.500000	38.000000	9.954000	14.775000	6.75900

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

In [4]:
```
# https://scikit-learn.org/stable/modules/tree.html#classification
# https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_s
caling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py
# https://towardsdatascience.com/data-science-mistakes-to-avoid-data-lea
kage-e447f88aae1c
### - FIXED DATA LEAKAGE
```

1) Split train/test data

In [5]:
```
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

y = df['bamscore']
X = df.drop('bamscore',axis=1) #drop bamscorerank too once I added so no
data leakage
# y is bamscorerank, 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 [6]:
```
train_test_split(y, shuffle=False)
```

Out[6]:
```
[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

- 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 [7]:
```
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())
```

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
```
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	0.181453	0.144703	0.013625	0.056108	0.015288
738	0.127032	0.098270	0.016855	0.060357	0.015843
739	0.143284	0.115483	0.013507	0.050077	0.013610
740	0.155346	0.168662	0.015299	0.051544	0.015202

741 rows × 13 columns

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

3) Feature Importance + Feature Selection

In [11]:
```
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, f_regression
```

In [12]:
```
X_train
```

Out[12]:

	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	0.181453	0.144703	0.013625	0.056108	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	0.155346	0.168662	0.015299	0.051544	0.01520

741 rows × 12 columns

In [13]:
```
# want to fit and transform the model on training x and y data
```

In [14]:
```
#skb = SelectKBest(f_regression, k=8)
skb = SelectKBest(score_func=f_regression, k=5)
fit = skb.fit(X_train, y_train)
features = fit.transform(X_train)
#X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
```

In [15]:
```
mask = fit.get_support()
X_train[X_train.columns[mask]]
```

Out[15]:

	approach_vertical	vertical_jump	wingspan	weight	hand_width
899	0.127644	0.110904	0.321202	0.771723	0.037665
635	0.139731	0.113526	0.338029	0.726981	0.038412
310	0.173806	0.130354	0.370615	0.675797	0.039617
961	0.108274	0.098607	0.311287	0.781892	0.034802
723	0.159682	0.126732	0.372591	0.679282	0.045623
...	...	...	...	...	...
1033	0.149692	0.128307	0.332649	0.751786	0.039205
763	0.181453	0.144703	0.328452	0.739132	0.039047
835	0.127032	0.098270	0.342747	0.733430	0.043143
559	0.143284	0.115483	0.322924	0.761332	0.041702
684	0.155346	0.168662	0.343981	0.723470	0.037727

741 rows × 5 columns

In [16]:
```
print(mask)
print(features)
print(features.shape)
```

```
[ True  True False False False  True False False  True False False  Tru
e]
[[0.12764397 0.11090378 0.32120246 0.77172292 0.03766544]
 [0.13973142 0.11352561 0.33802858 0.72698095 0.03841234]
 [0.17380558 0.13035419 0.37061485 0.675797   0.03961745]
 ...
 [0.12703203 0.09827006 0.34274681 0.73343023 0.04314295]
 [0.14328437 0.11548293 0.32292448 0.76133189 0.04170217]
 [0.15534627 0.16866167 0.34398103 0.72346978 0.03772695]]
(741, 5)
```

In [17]:
```
### OLS Regression (Ordinary Least Squares)
### Use this because it's the most simple baseline model
### Model gives best approximate of true population regression line.
### The principle of OLS is to minimize the square of errors
```

In [18]:

```
import statsmodels.formula.api as smf

formula = "bamscore ~ approach_vertical + vertical_jump + four_way_agili
ty"
lm = smf.ols(formula = formula, data = df_train).fit()
print(lm.summary())
print("After selecting best 3 features:", features.shape)
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:               bamscore   R-squared:
0.619
Model:                            OLS   Adj. R-squared:
0.617
Method:                 Least Squares   F-statistic:
398.5
Date:                Mon, 25 Apr 2022   Prob (F-statistic):          8.
99e-154
Time:                        12:02:45   Log-Likelihood:
714.13
No. Observations:                 741   AIC:
-1420.
Df Residuals:                     737   BIC:
-1402.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
==============
                        coef    std err          t      P>|t|      [0.0
25      0.975]
------------------------------------------------------------------------------
--------------
Intercept             0.5078      0.034     15.015      0.000       0.4
41       0.574
approach_vertical     3.9594      0.365     10.837      0.000       3.2
42       4.677
vertical_jump         3.3054      0.414      7.976      0.000       2.4
92       4.119
four_way_agility    -16.3254      0.752    -21.718      0.000     -17.8
01     -14.850
==============================================================================
=======
Omnibus:                       72.121   Durbin-Watson:
2.015
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
219.234
Skew:                          -0.459   Prob(JB):
2.48e-48
Kurtosis:                       5.502   Cond. No.
229.
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
After selecting best 3 features: (741, 5)
```

In [19]:
```
### 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.o^
2 is equal to residual sum squares
#### remember *o2 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/
```

Add features to try and improve r2 + find important features

In [20]:

```
import statsmodels.formula.api as smf
formula = "bamscore ~ approach_vertical + vertical_jump + reach + weight
+ body_comp + four_way_agility"
lm = smf.ols(formula = formula, data = df_train).fit()
print(lm.summary())
print(mask)
print(features)
print(features.shape)
#https://www.datatechnotes.com/2021/02/seleckbest-feature-selection-exam
ple-in-python.html
```

```
                            OLS Regression Results
=======================================================================
=======
Dep. Variable:              bamscore   R-squared:
0.623
Model:                           OLS   Adj. R-squared:
0.620
Method:                Least Squares   F-statistic:
202.5
Date:               Mon, 25 Apr 2022   Prob (F-statistic):          5.
79e-152
Time:                       12:02:45   Log-Likelihood:
718.80
No. Observations:                741   AIC:
-1424.
Df Residuals:                    734   BIC:
-1391.
Df Model:                          6
Covariance Type:           nonrobust
=======================================================================
==============
                       coef    std err          t      P>|t|      [0.0
25      0.975]
-----------------------------------------------------------------------
--------------
Intercept            0.9057      0.167      5.412      0.000       0.5
77       1.234
approach_vertical    3.8064      0.370     10.301      0.000       3.0
81       4.532
vertical_jump        3.1563      0.417      7.568      0.000       2.3
38       3.975
reach               -0.0871      0.168     -0.519      0.604      -0.4
16       0.242
weight              -0.3362      0.125     -2.699      0.007      -0.5
81      -0.092
body_comp           -0.0270      0.158     -0.171      0.864      -0.3
37       0.283
four_way_agility   -17.6547      0.904    -19.536      0.000     -19.4
29     -15.881
=======================================================================
=======
Omnibus:                      73.223   Durbin-Watson:
2.011
Prob(Omnibus):                 0.000   Jarque-Bera (JB):
227.170
Skew:                         -0.460   Prob(JB):
4.68e-50
Kurtosis:                      5.552   Cond. No.
360.
=======================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[ True  True False False False  True False False  True False False  Tru
e]
```

```
[[0.12764397 0.11090378 0.32120246 0.77172292 0.03766544]
 [0.13973142 0.11352561 0.33802858 0.72698095 0.03841234]
 [0.17380558 0.13035419 0.37061485 0.675797   0.03961745]
 ...
 [0.12703203 0.09827006 0.34274681 0.73343023 0.04314295]
 [0.14328437 0.11548293 0.32292448 0.76133189 0.04170217]
 [0.15534627 0.16866167 0.34398103 0.72346978 0.03772695]]
(741, 5)
```

In [21]:

```
# body comp and weight >0.05 - Ideally just drop them
```

Iteration - I wanted to take out because it is hard to measure already, now we know we can take it out because it's really throwing off the data and martin said it does not matter much, but model does not know that

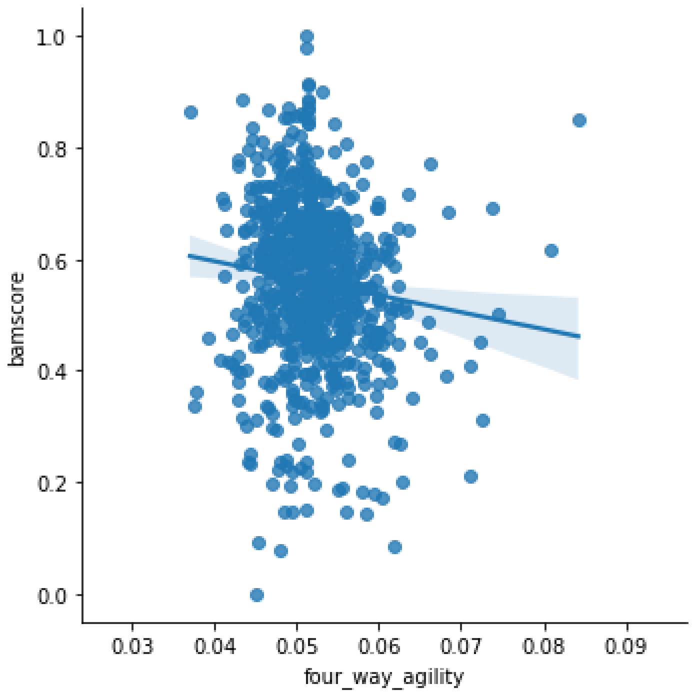
In [22]:

```
import statsmodels.formula.api as smf
formula = "bamscore ~ approach_vertical + vertical_jump + reach + four_w
ay_agility"
lm = smf.ols(formula = formula, data = df_train).fit()
print(lm.summary())
sns.lmplot(x="four_way_agility", y="bamscore" ,data=df_train)
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:               bamscore   R-squared:
0.620
Model:                            OLS   Adj. R-squared:
0.618
Method:                 Least Squares   F-statistic:
299.8
Date:                Mon, 25 Apr 2022   Prob (F-statistic):          7.
16e-153
Time:                        12:02:45   Log-Likelihood:
715.14
No. Observations:                 741   AIC:
-1420.
Df Residuals:                     736   BIC:
-1397.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
==============
                        coef    std err          t      P>|t|      [0.0
25      0.975]
------------------------------------------------------------------------------
--------------
Intercept             0.4708      0.043     11.028      0.000       0.3
87       0.555
approach_vertical     3.9304      0.366     10.749      0.000       3.2
13       4.648
vertical_jump         3.2492      0.416      7.810      0.000       2.4
32       4.066
reach                 0.1890      0.133      1.418      0.157      -0.0
73       0.451
four_way_agility    -16.9350      0.866    -19.566      0.000     -18.6
34     -15.236
==============================================================================
=======
Omnibus:                       69.439   Durbin-Watson:
2.018
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
228.944
Skew:                          -0.410   Prob(JB):
1.93e-50
Kurtosis:                       5.597   Cond. No.
283.
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

Out[22]: <seaborn.axisgrid.FacetGrid at 0x7ff491df7fd0>



Iteration - features I think are most important from pairplot in EDA

In [23]:

```
import statsmodels.formula.api as smf
formula = "bamscore ~ three_quarter_court_sprint + four_way_agility + re
action_shuttle + vertical_jump + approach_vertical"
lm = smf.ols(formula = formula, data = df_train).fit()
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:               bamscore   R-squared:
0.697
Model:                            OLS   Adj. R-squared:
0.695
Method:                 Least Squares   F-statistic:
337.9
Date:                Mon, 25 Apr 2022   Prob (F-statistic):          1.
01e-187
Time:                        12:02:46   Log-Likelihood:
799.16
No. Observations:                 741   AIC:
-1586.
Df Residuals:                     735   BIC:
-1559.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
=======================
                                coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
-----------------------
Intercept                     0.5582      0.030     18.313      0.000
0.498       0.618
three_quarter_court_sprint   -7.9533      1.923     -4.136      0.000
-11.728      -4.179
four_way_agility             -6.0437      1.019     -5.932      0.000
-8.044      -4.044
reaction_shuttle            -35.3052      2.773    -12.732      0.000
-40.749     -29.861
vertical_jump                 3.2517      0.370      8.788      0.000
2.525       3.978
approach_vertical             4.4234      0.328     13.489      0.000
3.780       5.067
==============================================================================
=======
Omnibus:                       76.718   Durbin-Watson:
2.017
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
457.485
Skew:                          -0.204   Prob(JB):                    4.
55e-100
Kurtosis:                       6.828   Cond. No.
955.
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

In [24]:
```
import statsmodels.formula.api as sm
model = sm.glsar(formula = formula, data = df_train)

lm = model.fit()
print(lm.summary())
```

```
                            GLSAR Regression Results
==============================================================================
=======
Dep. Variable:               bamscore   R-squared:
0.697
Model:                          GLSAR   Adj. R-squared:
0.695
Method:                 Least Squares   F-statistic:
338.1
Date:                Mon, 25 Apr 2022   Prob (F-statistic):           1.
10e-187
Time:                        12:02:46   Log-Likelihood:
798.14
No. Observations:                 740   AIC:
-1584.
Df Residuals:                     734   BIC:
-1557.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
=======================
                                coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
-----------------------
Intercept                     0.5579      0.030     18.301      0.000
0.498       0.618
three_quarter_court_sprint   -7.9910      1.923     -4.156      0.000
-11.766      -4.216
four_way_agility             -5.9933      1.020     -5.877      0.000
-7.995      -3.991
reaction_shuttle            -35.4437      2.776    -12.769      0.000
-40.893     -29.994
vertical_jump                 3.2400      0.370      8.753      0.000
2.513       3.967
approach_vertical             4.4347      0.328     13.518      0.000
3.791       5.079
==============================================================================
=======
Omnibus:                       76.581   Durbin-Watson:
2.013
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
458.040
Skew:                          -0.201   Prob(JB):                     3.
45e-100
Kurtosis:                       6.833   Cond. No.
955.
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

In [25]:

```
ypred = lm.predict(X_test)
```

In [26]:
```
r2_score(y_test, ypred)
```
Out[26]: 0.3491785846088724

In [27]:
```
# DO LM PLOT to see trend in data with respect to bamscore
```

In [28]:
```
# Random Forest
## Going to try random forrest because we had over fitting in models abo
ve
## 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 [29]:
```
from sklearn.ensemble import RandomForestRegressor

# create regressor object
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
cols = ["four_way_agility", "reaction_shuttle", "three_quarter_court_spr
int", "vertical_jump", "approach_vertical"]
# fit the regressor with x and y data
rf = regressor.fit(X_train, y_train)
rf.score(X_test, y_test)
# score if fit and test on full data frame
```
Out[29]: 0.5295593565201384

In [30]:
```
#x_train[cols]
#x_test[cols]

regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
cols = ["four_way_agility", "reaction_shuttle", "three_quarter_court_spr
int", "vertical_jump", "approach_vertical"]
# 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 just [cols]
```
Out[30]: 0.5382597305545691

In [31]:
```
# If Condition Number is too high I'll run into problems in matrix becau
se program may not catch small errors
```

In [32]:
```
# 5) Iterative modeling process
### What models are appropriate
### Compare Models
### Find which performance metrics to use and adjust to make the model b
etter.
```

In [33]:
```
### Assumptions to test in model:
#### - Combine Tests more important than physical measurments. Weigh com
bine tests as double.
#### - just tests, just measurments, 1:1 test/measurements, hypothesis -
2:1 test/measurment
```

In [34]:
```
#r2 = model isn't very accurate predicting bamscore with .453 bamscore
```

In [35]:
```
# Least Sum Squares to figure out best fit line and figure out most corr
elated features
```