Predictive Modeling with NFL Combine Data

Dylan Jorling & Dareh Aslanpour

Thursday 16th March, 2023

1 Background and Introduction

In 2021 the NFL generated over $17 billion of revenue, dwarfing every other domestic professional sports league despite playing considerably fewer games [1]. What often gets lost from a fan's perspective is that the league isn't just a form of entertainment, but also a massive corporation where billions of dollars are at stake. From a team owner's perspective, maximizing return on investment is the number one priority, while from a player's perspective, making as much money while their body still allows them to is a primary motivation. The league is competitive and cutthroat, and trying to find an edge over the competition is done on every level.

Winning on the field generates increased fan interest, leading to more ticket and merchandise sales, larger local tv deals, and more money in owners' pockets. Therefore, owners invest hundreds of millions of dollars each year on coaches to lead teams, scouts to properly assess talent, and players to perform on the field. Each NFL franchise has 52 roster spots and over $220 million (and rising) to spend every year on player salaries [5]. With 32 total franchises, this means that the 1,664 players who earn roster spots will split over $7 billion dollars in the upcoming season alone.

While the playoffs and super bowl are the highlight of the season for fans, there is no event that impacts more NFL personnel than the NFL draft. In the NFL draft, teams select from the best draft-eligible college football players in hopes of improving their current roster. Each franchise is allotted one pick per round over 7 total rounds, and the draft order is in reverse of the previous years' standings. The NFL draft is extremely important for every team involved, and for different reasons. Teams picking towards the top of the draft performed poorly in the prior season and are often devoid of talent. While they "earn" priority selection, enabling them to choose the most elite players, success in college football does not guarantee success in the NFL. Although top draft picks have the highest probability of being franchise-altering players, drafting a "bust"–a player who greatly underperforms his draft slot–is extremely costly and could set a team back years.

Teams picking towards the back of the draft typically have very little flexibility to improve, or even maintain, their rosters. Players on good teams often out-perform previous contracts, which makes it difficult for those teams to re-sign their players within the limit of the salary cap. These teams' clearest avenue to improvement is through the draft, which is made even more difficult by the fact they own lower-priority draft picks. For these teams to maintain success then, they must find "steals" in the draft who greatly outperform their draft slots.

From a player's perspective, the difference between getting drafted or not will likely affect their entire life's trajectory. It could be the difference between earning close to a million dollars in one year and working a minimum wage job. Getting drafted means an automatic guarantee of income. Even "Mr. Irrelevant"–the last overall pick, number 261–earns $85k just for being drafted, and will lock in $725k if he manages to make the final roster cut. A player drafted in the top 100 picks earns a minimum of $940k, while a top-32 first round pick earns a whopping minimum of $6.1M [5].

Given the draft stakes for both franchises and players, an intense pre-draft process occurs over many months prior to the actual draft. Teams are looking to identify undervalued talent while players are trying to stand out from their peers. One of the most important events in this process is the NFL combine. The NFL combine is a 4-day event where the top college football players are invited to partake in what is essentially a lengthy job interview [2]. Players are tested both physically, with measurements and drills, and mentally, with intelligence tests and long interviews. While college performance is typically the leading factor in evaluating a draft prospect, combine results are also thought to be an important part of the process.

In our paper we evaluate the predictive power of physical combine results in determining 1) whether a player is drafted or not and 2) what pick a player is likely to be drafted with. To answer the first question, we set up a binary classification problem and build a variety of predictive models. To answer the second question, we isolate only those players who are actually drafted and analyze a regression problem using draft pick as a response variable.

2 Data Cleaning and Overview

Initial combine data is sourced from nflcombineresults.com and contains 12 unique combine measurements over 13.544 players and 36 years [3]. One key feature missing from the initial data is our response variable, draft pick. We retrieve draft pick data from pro-football-reference.com over the same 36 year time period and append it to the initial dataset [4]. Due to inconsistencies with names of players, colleges and positions between the two datasets, matching names up between the two sets is non-trivial. Additionally, we do not know which players who were actually drafted did not attend the combine. We match the datasets first using name, and then by using a combination of college and year to identify cases where players have slightly different names in the two datasets. We assign draft picks to 8200 players out of the 9600 who were drafted over the 36 year period, and while we likely incorrectly assign some players, it is plausible that a majority of this discrepancy stems from drafted players who did not attend the combine.

Next, we trim positional categories from 25 down to 9, including completely dropping the specialist positions of kickers, punters and long snappers. We also remove the Wonderlic and 60 yard shuttle features since more than 2/3rds of the data is missing for each. Many of the remaining variables still contain significant amounts of missing data, and instead of trying to impute the data we decide to only keep the complete cases. The final dataset includes 5,843 observations with 10 numerical measurements each, plus the categorical position variable which we one-hot encode.

For the classification problem we assign a binary response variable based on whether a player was drafted or not. We analyze the regression problem on the 3,422 observations where players were actually drafted, and use raw overall draft pick as the response variable. It should be noted that a lower draft pick corresponds to an earlier pick and theoretically a better player. Many of the numerical variables are self-explanatory, such as height, weight, hand size and arm length. The remaining, less obvious variables are:

- 40-Yard Dash- electronically timed forty yard dash time, in seconds
- Bench- amount of consecutive repetitions of a 225-pound bench press completed
- Vertical Jump- standing vertical jump, in inches
- Broad Jump- the length, in inches of a player's standing broad jump
- Shuttle- time in a 20-yard back and forth "shuttle" run, in seconds
- 3cone- time in seconds taken to complete the 3-cone drill, another short-shuttle type drill involving running to various cones

3 Exploratory Data Analysis

3.1 Classification

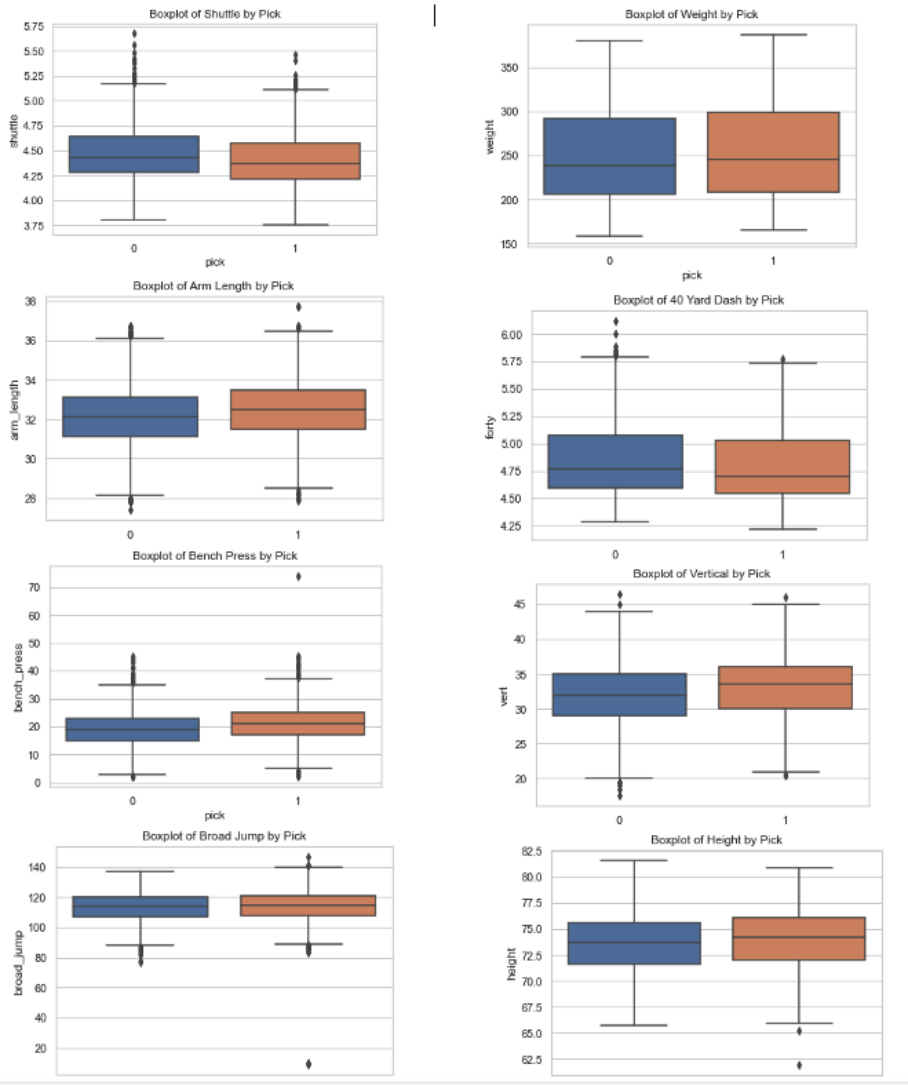


Figure 1: Predictor Boxplots: Drafted vs. Un-drafted

We convert the draft pick column to a binary variable where 0 indicates a player was un-drafted and 1 indicates the player was drafted after the combine. To get a general understanding of our data and a classification goal we observe drafted vs un-drafted statistics. From the pick column we see that about 58.6% of people who participated in the combine and 41.4% did not.

The boxplots in Figure 1 visualize the distributions of some of our numeric variables with respect to their draft standing (0 = un-drafted and 1 = drafted). From the boxplots we can see that variables such as Height and Broad Jump do not look very indicative of draft pick as the plots are very similar to one another. Other features such as 40-Yard Dash and Shuttle appear to have larger statistical differences between the two groups . From the 40-Yard Dash it is evident that players who are drafted tend to be quicker than those who are not, which is also supported by

	height	weight	hand_size	arm_length	forty	bench_press	vert	broad_jump	shuttle	cone
height	1.000000	0.749966	0.494092	0.723978	0.626795	0.357892	-0.438507	-0.424118	0.528578	0.508327
weight	0.749966	1.000000	0.496161	0.602941	0.883426	0.626132	-0.673852	-0.717905	0.748888	0.771973
hand_size	0.494092	0.496161	1.000000	0.504237	0.411424	0.294874	-0.275543	-0.274962	0.340291	0.346123
arm_length	0.723978	0.602941	0.504237	1.000000	0.468437	0.259697	-0.316546	-0.279479	0.461745	0.438580
forty	0.626795	0.883426	0.411424	0.468437	1.000000	0.459114	-0.754774	-0.795473	0.772295	0.809775
bench_press	0.357892	0.626132	0.294874	0.259697	0.459114	1.000000	-0.314744	-0.368405	0.374785	0.403321
vert	-0.438507	-0.673852	-0.275543	-0.316546	-0.754774	-0.314744	1.000000	0.786275	-0.691112	-0.668508
broad_jump	-0.424118	-0.717905	-0.274962	-0.279479	-0.795473	-0.368405	0.786275	1.000000	-0.670406	-0.702932
shuttle	0.528578	0.748888	0.340291	0.461745	0.772295	0.374785	-0.691112	-0.670406	1.000000	0.801462
cone	0.508327	0.771973	0.346123	0.438580	0.809775	0.403321	-0.668508	-0.702932	0.801462	1.000000

Figure 2: Numerical Predictor Correlation Matrix

the Shuttle plot. Additionally, there are significantly fewer "negative" outliers for drafted players in both features. The boxplots support the notion that players who are drafted tend to perform better in athleticism based tests in the NFL Combine, although the actual predictive power of these measurements is questionable.

We produce a feature correlation matrix shown in Figure 2 which unsurprisingly shows evidence of collinearity amongst various predictor pairs. Here shuttle, cone, forty, and weight are all highly correlated with each other while broad jump and vertical jump also show strong correlation. The first set of predictors are speed-related metrics, with the latter set both relating to jumping so these results are logical. Collinearity is an issue that both causes unreliable statistical inference and unnecessarily complicates models, and we therefore plan to address this issue when building out our models.

3.2 Regression

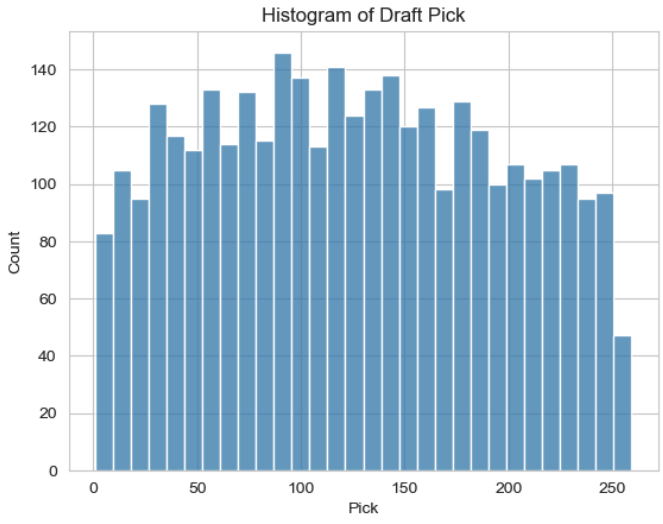


Figure 3: Draft Pick Distribution

The first thing we examine for the regression model is the distribution of the response variable displayed as a histogram in Figure 3. Although the distribution of actual draft picks is uniform, the histogram looks somewhat more normally distributed. This makes sense given the fact that many top draft picks do not want to hurt their prospects at the combine or at least not partake in certain drills, and players picked towards the end of the draft are sometimes not invited to the combine.

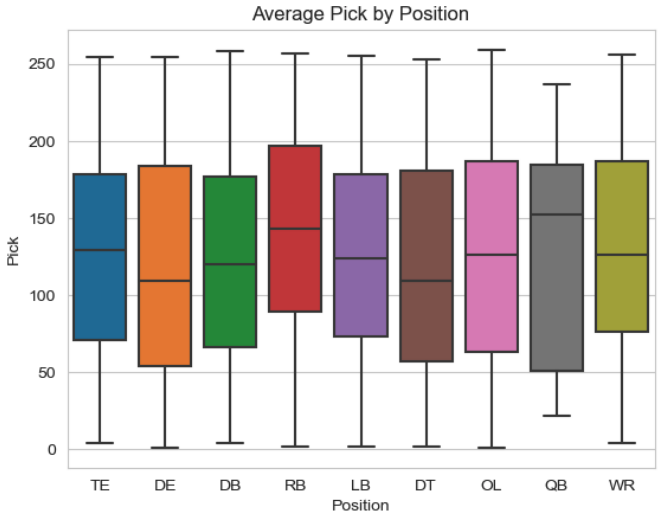


Figure 4: Average Pick by Position

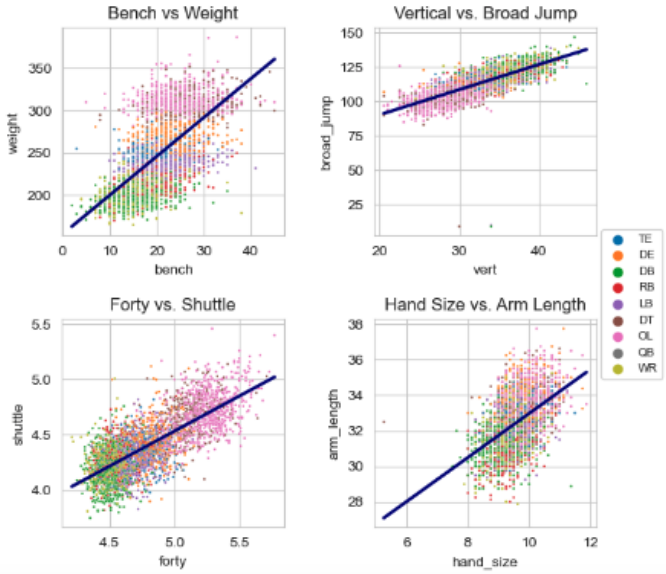


Figure 5: Comparing Predictor Variables

Figure 4 displays a boxplot of the categorical variable, position, with the response variable, pick. Overall, the positions are fairly similar to each other, with most positions having similar pick distributions. Quarterbacks have the most skewed distribution since a top QB is the most valuable position on the field while a middle-of-the-road quarterback holds very little value. Defensive Ends

and Tackles, who are often tasked with bringing down the quarterback on the opposing team, have the lowest average draft pick selection.

Similar to the full dataset, Figure 5 shows strong correlation amongst many of the predictors and therefore multi-collinearity needs to be addressed. Figure 6 shows the response variable plotted against four predictor variables that showed promise in the classification EDA. Surprisingly, none of the four variables appear to have a very strong relationship with pick. While these plots are somewhat discouraging, we still believe we can produce an interesting study even if our prior thoughts about combine results are disproven.

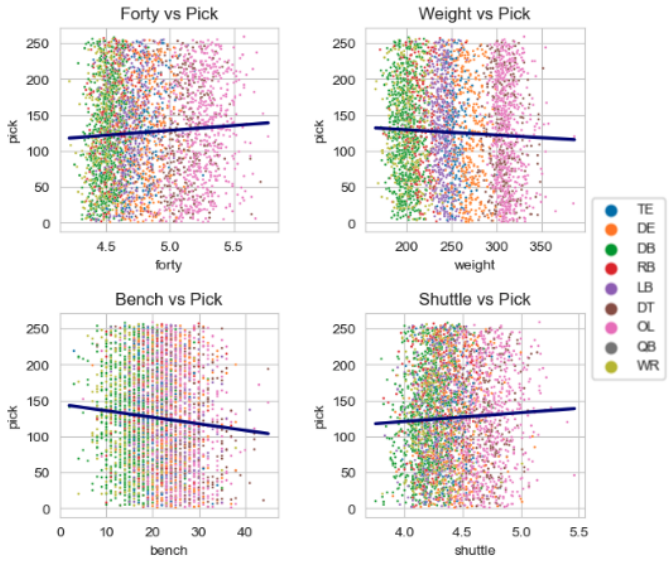


Figure 6: Comparing Predictors with Response

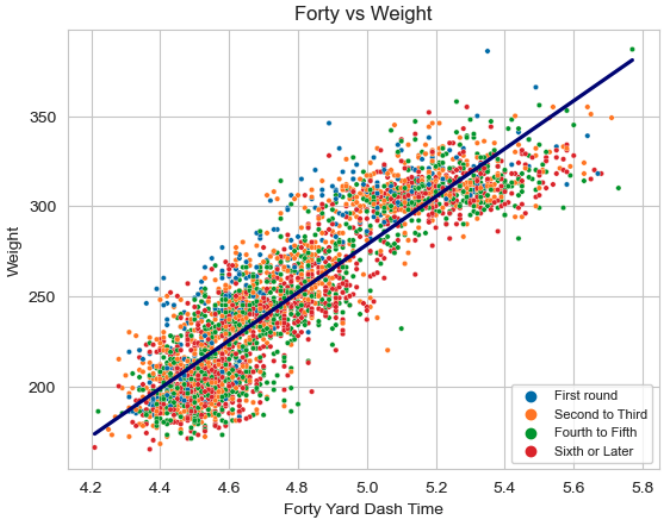


Figure 7: 40 time vs. Weight

Figure 7 displays the interaction between forty yard dash time and weight. The logic behind

including this plot is that players who are both big and fast would seemingly be picked towards the top of the draft. We encode the draft pick variable into categorical "draft rounds" for visual effect. Generally, the residuals confirm our intuition, particularly with first round picks. On average these top draft picks run faster (lower time) than other players of the same weight, and are bigger than players who run as fast.

4 Methodology and Results

4.1 Classification

4.1.1 Logistic Regression

We begin our classification with a simple Logistic Regression model containing every variable. In this initial model we use the variables Height, Weight, Hand Size, Arm Length, Forty, Bench Press, Vertical, Broad Jump, Shuttle, Cone, and Position against our response variable of whether or not a player was drafted. This simple and efficient Logistic model yields a cross validated (CV = 10) accuracy of 64.3%. The initial model correctly predicts 1,127 of our 1,753 observations. The model identifies 81% of drafted players at the expense of only correctly classifying 43% of un-drafted players.

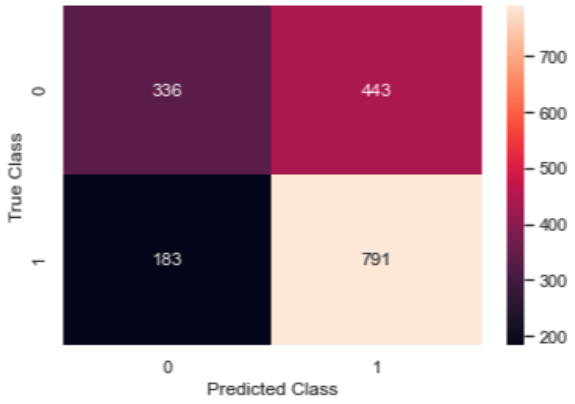


Figure 8: Full Model Logistic Regression Confusion Matrix

Using a feature importance plot from a subsequent Random Forest model we create a reduced model, eliminating features deemed statistically insignificant. The reduced model includes only forty, weight, and shuttle and attains a cross validated accuracy of 65.8%. The model increases accuracy by 1.5% while also significantly reducing model complexity. Seemingly then, combine results are a pretty good indicator of whether or not a player is drafted.

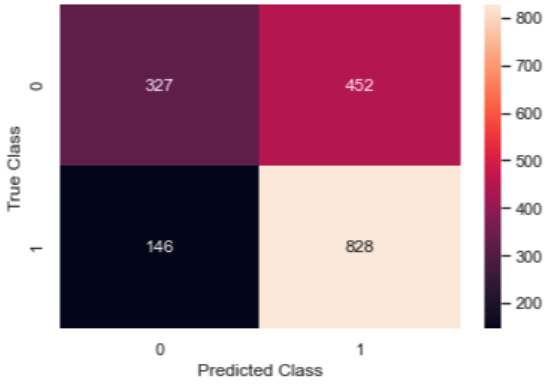


Figure 9: Reduced Model Logistic Regression Confusion Matrix

Figure 9 shows the cross-validated confusion matrix for the reduced logistic regression model while Figure 10 displays various other model metrcis. From our reduced model we get coefficients of 0.1765 for hand size, 0.1667 for arm length, 0.0508 in bench press, and 0.1268 in vertical. This means that a 1 inch increase in hand size is associated with an increase of 19.3% in the odds of being drafted, a 1 inch increase in arm length is associated with an increase of 18.1% in the odds of being drafted, a 1 rep increase in bench press is associated with an increase of 5.2% in the odds of being drafted, and a 1 inch increase in vertical is associated with an increased of 13.5% in the odds of being drafted.

	precision	recall	f1-score	support
0	0.69	0.42	0.52	779
1	0.65	0.85	0.73	974
accuracy			0.66	1753
macro avg	0.67	0.63	0.63	1753
weighted avg	0.67	0.66	0.64	1753

Figure 10: Reduced Model Logistic Regression Metrics

In addition to being more accurate than the full model, the reduced model increased precision from 64% to 67% and recall from 64% to 66%. Although the reduced model outperforms the full model in nearly every statistic, the reduced model actually produces a worse recall score amongst un-drafted players at just 42%.

4.1.2 Random Forest

We follow a similar protocol as the logistic regression model by starting with all variables to fit a random forest model that produces an accuracy of 64.2%, which falls short of our logistic regression results.

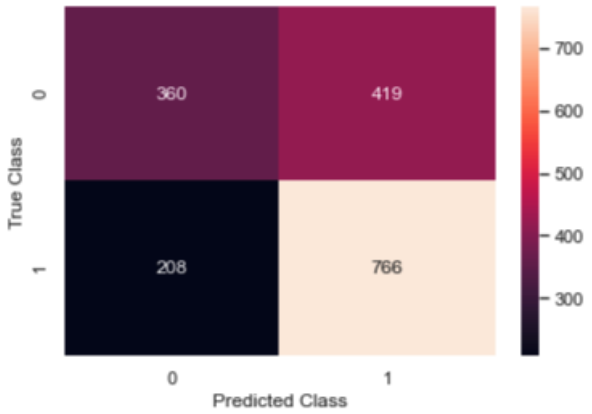


Figure 11: Random Forest Confusion Matrix

Figure 11 displays the cross-validated confusion matrix for the random forest model, while Figure 12 displays model metrics. Most of the results from the random forest fall slightly below our logistic models results. The one thing that stands out from the random forest model is the

	precision	recall	f1-score	support
0	0.63	0.46	0.53	779
1	0.65	0.79	0.71	974
accuracy			0.64	1753
macro avg	0.64	0.62	0.62	1753
weighted avg	0.64	0.64	0.63	1753

Figure 12: Random Forest Metrics

4% increase in recall amongst un-drafted players, meaning that it identifies about 4% more of the true un-drafted players. The trade-off here is precision, where the reduced logistic model is 6% more precise amongst un-drafted players.

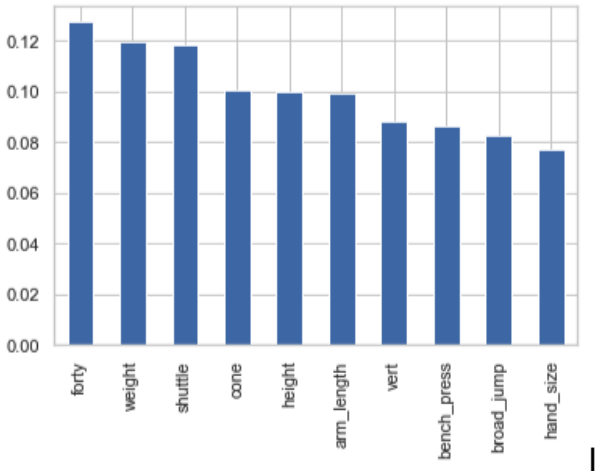


Figure 13: Random Forest Feature Importance

Figure 13 represents feature importance for the Random Forest Model with forty, weight, and shuttle being the three most important variables. We further attempt to optimize our model through variable reduction, and achieve a max accuracy of about 63.8% which falls just short of the full random forest model performance. Therefore we decide to keep the Random Forest Model as is and include each numerical predictor.

4.2 Regression

4.2.1 OLS

We begin our regression analysis with an OLS statistical model using draft pick as the response. In our first model we use all 10 numerical predictors along with the positional categorical variable. Full model results yield an R-squared of 0.088 and an adjusted R-squared of 0.084, which means that only 8.4% of the variance in draft pick is explained by this set of predictors. The lack of strong linear relationships in our EDA scatterplots makes this result unsurprising.

Next, we implement a backward selection process where we iteratively remove the max p-value variable until all remaining predictors are statistically significant. The final model contains 5 numerical variables and 3 of the 8 categorical dummy variables.

OLS Regression Results

Dep. Variable:	pick	R-squared:	0.084
Model:	OLS	Adj. R-squared:	0.082
Method:	Least Squares	F-statistic:	39.27
Date:	Fri, 03 Mar 2023	Prob (F-statistic):	2.81e-60
Time:	07:25:40	Log-Likelihood:	-19210.
No. Observations:	3418	AIC:	3.844e+04
Df Residuals:	3409	BIC:	3.849e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-222.0558	49.765	-4.462	0.000	-319.628	-124.484
weight	-0.6549	0.088	-7.462	0.000	-0.827	-0.483
arm_length	-6.3900	1.051	-6.079	0.000	-8.451	-4.329
forty	119.8296	9.628	12.445	0.000	100.952	138.708
bench	-0.5226	0.249	-2.102	0.036	-1.010	-0.035
shuttle	37.9831	7.230	5.253	0.000	23.807	52.159
DB	-15.0889	3.667	-4.115	0.000	-22.279	-7.899
DT	-18.5718	5.954	-3.119	0.002	-30.245	-6.898
OL	-28.7441	5.753	-4.996	0.000	-40.024	-17.465

Omnibus:	486.318	Durbin-Watson:	1.925
Prob(Omnibus):	0.000	Jarque-Bera (JB):	129.009
Skew:	0.147	Prob(JB):	9.68e-29
Kurtosis:	2.095	Cond. No.	1.13e+04

Figure 14: OLS Summary

The model is summarized by Figure 14. In the final model, each individual variable is highly significant with the response variable, except for bench, which is only significant at the alpha = 0.05 level. Not surprisingly, shuttle and forty, which relate to timed speed, are positively correlated with a later draft pick. This makes sense because lower forty and shuttle times indicate faster speed. Additionally, the other numerical predictors that relate to size are negatively correlated with draft pick which is also logical–bigger players, holding all else equal, are generally drafted earlier.

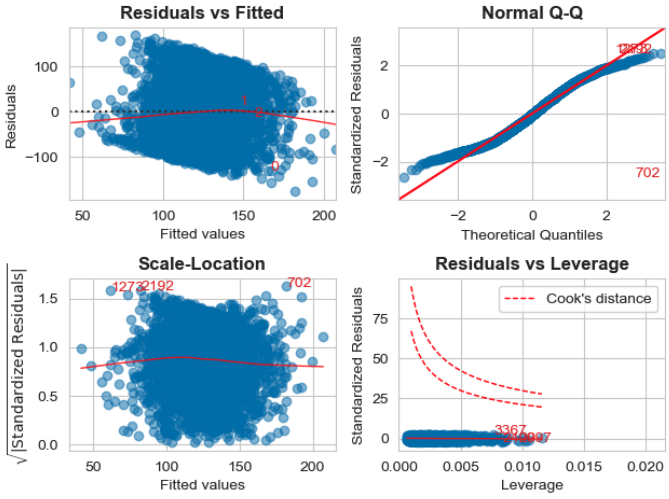


Figure 15: Diagnostic Plots

The diagnostic plots shown in Figure 15 bring into question the goodness-of-fit of a linear model. The errors do not appear to be normally distributed as shown by the residuals vs. fitted

and normal q-q plots. One positive note is that the data does not appear to contain any outliers, as shown by the scale-location and leverage plots. Unfortunately, re-specifying the model is not straightforward. Using a robust regression would not have any effect since the data does not contain outliers. Implementing polynomial regression or basis splines on the other hand, is complicated given the number of predictors utilized. The best approach is likely transforming the pick variable into a categorical variable with multiple rounds and then running a multinomial regression. However, since we already conduct a classification model in this study, we decide to leave the variable as is and proceed with a regression approach.

4.2.2 Predictive Linear Modeling

Next, we create a set of predictive models, cross-validate them and compare results in terms of RMSE. The first predictive model we build is based on the final OLS model. We conduct 3-fold cross validation and yield an RMSE of 67.02. The null model standard deviation is 69.8, meaning if you were to just predict the mean for each pick you would be off by 69.8 picks on average. Therefore the predictive power of the model is limited.

4.2.3 PCR

We implement PCA/PCR since our data consists of several features that are highly correlated with each other. Principal Component Analysis (PCA) is a method used to reduce predictor dimensions by eliminating collinearity within the dataset. The first principal component is computed by finding a linear combination of input variables with greatest variation in the data. Subsequent components are computed by finding the linear combination of variables with maximum variation in the data, that is also orthogonal to all prior components.

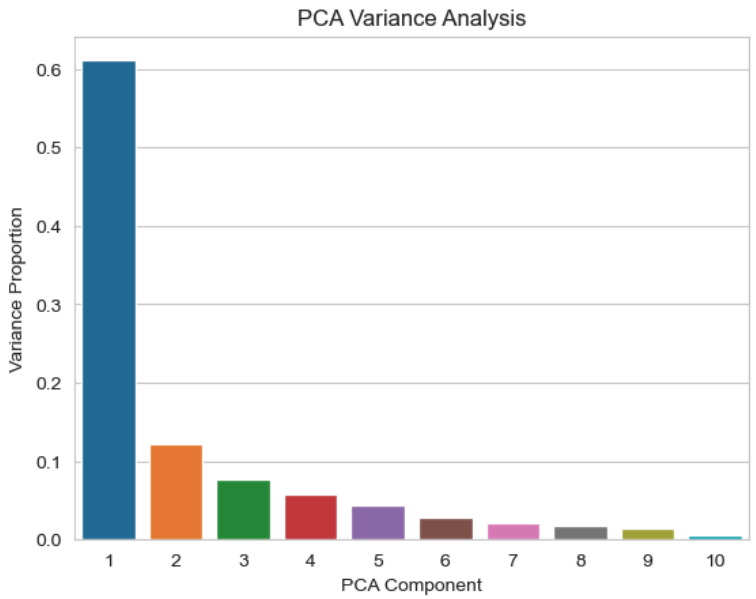


Figure 16: PCA Variance

Figure 16 shows PCA results for our 10 numerical predictors. The first principal component accounts for over 62% of the variation in the data and by the 5th principal component, 91.1% of variation in the data is accounted for.

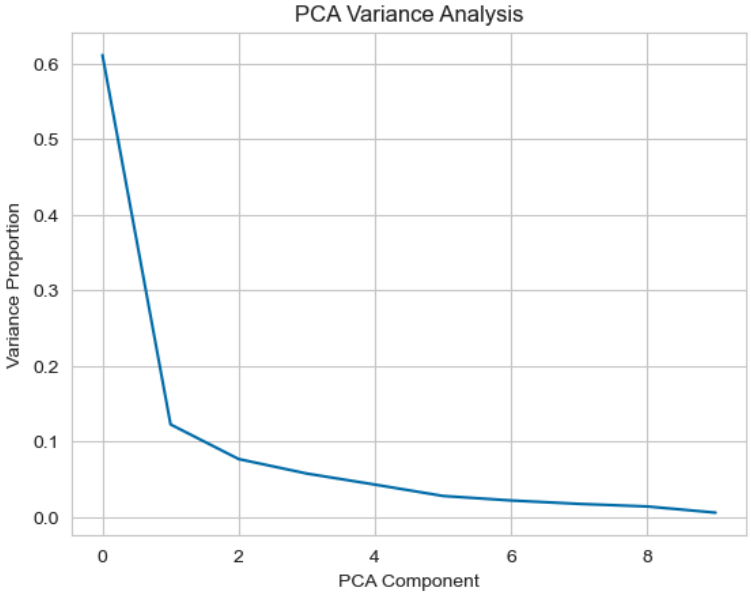


Figure 17: PCA Variance Analysis

Figure 17 displays total variance of each component. Prior to implementing PCR, we must determine how many principal components to use in the model. The rate change in variance proportion significantly flattens after both 2 and 5 principal components, and we therefore choose 3 Principle Components, which is between the two elbows.

	PC1	PC2	PC3
height	0.304736	-0.405880	-0.107401
weight	0.385545	-0.038562	0.142756
hand_size	0.228303	-0.461385	0.002039
arm_length	0.263038	-0.533021	-0.253794
forty	0.373551	0.149111	-0.022600
bench	0.232707	-0.048905	0.913638
vert	-0.319665	-0.333223	0.195778
broad_jump	-0.324417	-0.386599	0.068543
shuttle	0.342782	0.121846	-0.135350
3cone	0.344318	0.191882	-0.084137

Figure 18: PCA Loadings

Figure 18 shows PCA loadings for each of the three principal components we selected. Generally, we classify PC1 as a summary of a player's size. PC2 is more difficult to explain but appears inversely related to hand/arm size and jumping ability, while PC3 is almost purely strength related.

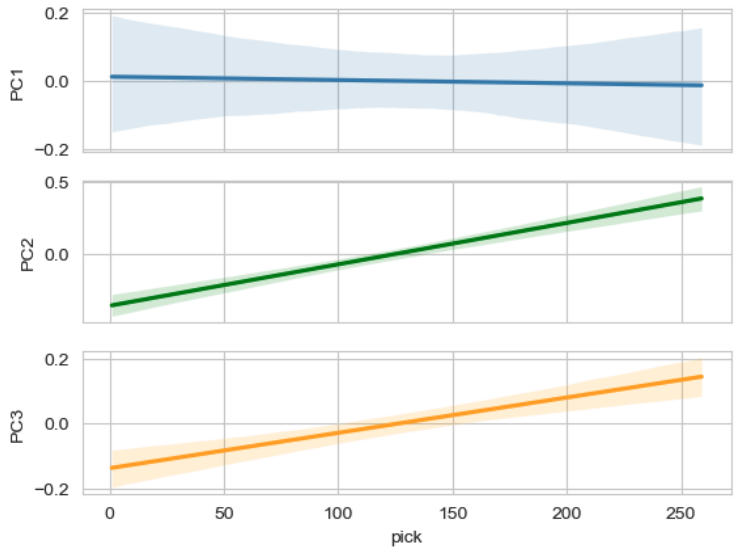


Figure 19: PCA vs Response

Figure 19 shows each principal component's relationship with the response variable. The first principal component shows 0 correlation with the response variable y. This is not surprising since PCA does not take into account the response variable at all. The other two principal components show a much larger positive correlation with draft pick however, meaning the larger each component is the later in the draft a player will be selected. This makes a lot of sense in regards to PC2 given its loadings, but since PC3 is mostly composed of strength, a positive correlation is surprising.

To ensure that our PCR model is fairly measured against our other models, we add the one-hot encoded variables for the three positions used in the linear model: DB, DT and OL. Using 3-fold cross validation, the PCR model produces an RMSE of 68, meaning that on average, the model will predict a draft pick position that is 68 picks away from the actual value. Given a null model standard deviation of 69.8 this proves again to not be very predictive. The model also significantly underperforms compared to the linear model.

4.2.4 XGBoost

The final predictive model we use for the regression problem is XGBoost. Boosting is an ensemble method where committee trees are sequentially added to the prior committee. The idea behind boosting is to treat the error from the previous ensemble as the response variable for the next tree grown. XGBoost is a form of boosting where both L1 and L2 regularization are implemented into the loss function, and is solved using iterated re-weighted least squares.

XGBoost includes various hyper-parameters that can be adjusted. To implement hyper-parameter tuning, we first split the data into a training set containing 70% of observations and a testing set containing 30% of observations. We implement a cross-validated grid search on the training set to determine the optimal combination of max_depth, eta, and n_estimators. Max depth refers to the maximum depth, or layers in each tree that is built, Eta is the learning rate, and n_estimators is the number of trees that are built.

Cross-validation yields the following optimized parameters: max_depth=1, eta=0,25, and n_estimators=100. Finally, we run another iteration of XGBoost using the tuned hyper-parameters on the unseen test set. The result yields an RMSE of 68.45. Surprisingly the XGBoost model which is known to be quite powerful, performs worse than PCR and substantially worse than the linear model.

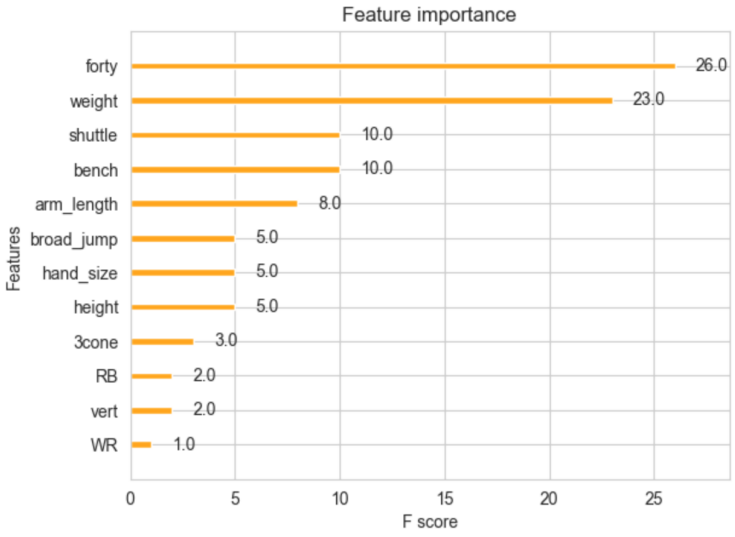


Figure 20: XGB Feature Importance

Figure 20 shows a plot of feature importance based on the F score of each variable in our XGBoost model. Forty time and weight are the two most predictive features in the dataset, with shuttle and bench belonging in the next tier. Speed, size and strength are generally important factors in playing football so these results seem logical. Not surprisingly these features also correspond with the variable importance in the OLS model.

5 Limitations, Further Research and Conclusion

5.1 Limitations

One limitation of the model is that we do not account for 1400 players who were drafted. The proportion of these players who did not attend the combine is unknown meaning we cannot rule out some mis-classifications. Additionally, we have significant amounts of missing data which limits the size of the dataset used. We also do not have any data on any interviews players partake in and cannot determine whether those interviews have any impact on draft position.

5.2 Further Research

One interesting thing that could be looked into is how players' combine stats compare to their performance once they actually reach the NFL. The question to be asked here would be: how predictive are combine stats in determining the quality of an NFL playing career? The previously mentioned missing data would be another important aspect to examine. Many players will only do certain drills at their school-specific Pro-Days, so using this data to impute missing combine data would be helpful in utilizing a larger dataset.

Another thing that could be examined is the small amount of data on the Wonderlic score, which is an "intelligence" test typically seen as being important for only certain positions. Finally, the data could be used to determine if a player is a candidate to play a different position in the NFL due to their combine numbers. For example, a player not projected to be drafted as a tight end due to lack of speed could have better odds to be drafted as an offensive lineman if they have enough size. It could be interesting for a player to explore potential positional transitions based on their physical metrics.

5.3 Conclusion

For the classification problem, the model with the the highest accuracy is the reduced-model Logistic Regression which produces an accuracy of 65.8% a metric we are satisfied with given the dataset. For the regression problem, the model that produces the lowest RMSE is the basic linear model using the subset of variables determined by the OLS model. Although the RMSE of 67.02 is not very predictive, the model does perform much better than both the PCR and XGBoost models.

Given the significantly better performance of the classification model compared to the regression model, we conclude that combine stats are more important in determining whether a player is drafted than where in the draft they are taken. Combine results are seemingly used more as a threshold a prospect must meet before other factors are used to determine where a player is actually selected. We also conclude that other factors are likely much more important in determining draft position such as college stats and level of playing competition.

Presentation Questions

Nicole: In the Regression model, what was your backward selection process?

Our backward selection process began with a full model using all 10 numerical variables and 8 dummy variables. We iteratively dropped the variable with the highest p-value until each variable had a p-value significant at the alpha = 0.05 level, leaving us with our final model.

Yulin: Can you explain why your data has missing data and what your approach to this data was?

The data contained a lot of missing data because players will often only choose to partake in only certain drills at the combine. In many situations, players will only participate in running drills at their "Pro Day" which is basically a mini-combine done for each individual college. For our purposes, after removing two columns that contained more than 2/3rds NAs every variable outside of height and weight contained at least 10% missing values. At this point we decided to keep only the complete cases which represented about half of the data. Although throwing out half the data is not ideal, we still had about 6,000 observations to work with which is why we felt comfortable just using these observations.

References

[1] Christina Gough. Total revenue of the national football league 2001-2021. *Statista*, January 2023.

[2] NFL.com. Nfl scouting combine.

[3] Nflcombineresults.com. Combine results, 2022.

[4] Pro Football Reference. Historical draft picks, 2022.

[5] Spotrac. Nfl draft salary per round, 2023.

dataclean

March 16, 2023

[5]:
```
import pandas as pd
import numpy as np
```

[9]:
```
# load data...source: nflcombineresults.com
path = '/Users/dylanjorling/UCLA/412proj/data/'
name = 'combine.csv'
combine = pd.read_csv(path+name)
combine.head()
```

[9]:

	Year	Name	College	POS	Height (in)	Weight (lbs)	\
0	1987	Mike Adams	Arizona State	CB	69.8	198	
1	1987	John Adickes	Baylor	C	74.8	266	
2	1987	Tommy Agee	Auburn	FB	71.8	217	
3	1987	David Alexander	Tulsa (OK)	C	75.0	279	
4	1987	Lyneal Alston	Southern Mississippi	WR	72.1	202	

	Hand Size (in)	Arm Length (in)	Wonderlic	40 Yard	Bench Press	\
0	8.50	30.50	NaN	4.42	13.0	
1	10.25	30.00	NaN	4.97	25.0	
2	9.00	30.75	NaN	NaN	15.0	
3	10.50	32.75	NaN	5.13	22.0	
4	10.00	33.00	NaN	4.64	7.0	

	Vert Leap (in)	Broad Jump (in)	Shuttle	3Cone	60Yd Shuttle
0	32.0	118.0	4.60	NaN	11.91
1	26.5	103.0	4.60	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN
3	27.5	105.0	4.33	NaN	NaN
4	32.0	114.0	4.52	NaN	11.85

[11]:
```
# clean draft data
draft_init = pd.read_csv(path+'1987.csv')
draft_init2 = draft_init.iloc[:, :6]
draft_init2['College'] = draft_init['College/Univ']
draft_ttl = pd.DataFrame(columns=draft_init2.columns)
for i in range(1987, 2023):
    file=path+str(i) + '.csv'
```

```
    draft = pd.read_csv(file)
    draft_int = draft.iloc[:, :6]
    draft_int['College'] = draft['College/Univ']
    draft_int['Year'] = np.repeat(i, draft_int.shape[0])
    draft_ttl = pd.concat([draft_ttl, draft_int], axis=0)
draft_ttl.reset_index(inplace=True)
draft_ttl.drop(columns="index", inplace=True)
draft_ttl['Player'] = [player[:-4] if "HOF" in player else player for player in␣
 ↪draft_ttl["Player"]] # clean out "HOF"
draft_ttl['Year'] = draft_ttl['Year'].astype("int")
draft_ttl.shape
```

[11]: (9558, 8)

```
[12]: # clean college
combine['College'].fillna(value="No College", inplace=True) # clean nas
clean_name = []
for name in combine['College']:
    if (name[-1:] == ')') & (name != 'Miami (OH)'):
        clean_name.append(name[:-5])
    else:
        clean_name.append(name)
combine['College'] = clean_name
clean_name = []
for name in combine['College']:
    if name[-5:] == 'State':
        clean_name.append(name[:-5] + 'St.')
    else:
        clean_name.append(name)
combine['College'] = clean_name
combine['College'].replace({'Southern California': 'USC',
                            'Louisiana St.': 'LSU',
                            'Boston College': 'Boston Col.',
                            'Miami': 'Miami (FL)',
                            'Texas Christian': 'TCU',
                            'Frenso St.': 'Fresno St.',
                            'Southern Methodist': 'SMU',
                            'Southern Mississippi': 'Southern Miss',
                            'Nevada Las Vegas':'UNLV',
                            'Missouri St.': 'Missouri State',
                            'Middle Tennessee St.': 'Middle Tenn. St.',
                            'Eastern Kentucky': 'East. Kentucky',
                            'Louisiana-Monroe': 'La-Monroe',
                            'Cal-State Fullerton': 'Cal State-Fullerton',
                            'Arkansas-Pine Bluff': 'Ark-Pine Bluff',
                            'Eastern Washington': 'East. Washington',
                            'Citadel': 'The Citadel',
```

```
                           'Cal Poly': 'Cal Poly-San Luis Obispo',
                           'Tennessee-Chattanooga': 'Chattanooga',
                           'Northwestern St.': 'Northwestern St. (LA)',
                           'Brigham Young': 'BYU',
                           'Louisiana-Lafayette': 'Louisiana',
                           'Stephen F. Austin': 'S.F. Austin',
                           'Eastern Michigan': 'East. Michigan',
                           'Northwest Missouri St.': 'NW Missouri St.',
                           'Southeast Missouri St.': 'SE Missouri St.',
                           'Alabama-Birmingham': 'Ala-Birmingham',
                           'Tennessee-Martin': 'UT Martin',
                           'Central St.': 'Central State (OH)',
                           'Central Missouri': 'Central Missouri St.',
                           'Wisconsin-Whitewater': 'Wisconsin-Whitewater',
                           'Wisconsin-Steven\'s Point': 'Wisconsin-Stevens␣
 ↪Point',
                           'UC-Davis': 'UC Davis',
                           'Mississippi Valley St.': 'Miss. Valley St.',
                           'North Carolina Charlotte': 'Charlotte',
                           'Eastern New Mexico': 'East. New Mexico',
                           'East. Illinois': 'Eastern Illinois',
                           'Sonoma': 'Sonoma St.',
                           'Missouri Western': 'Missouri Western St.',
                           'Kutztown': 'Kutztown (PA)',
                           'Albany St.': 'Albany State (GA)',
                           'Albany': 'Albany (NY)',
                           'Central Connecticut': 'Central Connecticut St.',
                           'East Central': 'East Central (OK)',
                           'Concordia - St Paul': 'Concordia-St.Paul (MN)',
                           'Charleston': 'Charleston (WV)',
                           'Augustana': 'Augustana (SD)',
                           'Angelo St.': 'Angelo State (TX)',
                           'Southwest Minnesota': 'SW Minnesota',
                           'Western Ontario (Ca': 'Western Ontario',
                           'Case Western': 'Case Western Reserve',
                           'Wayne St.': 'Wayne State (NE)'}, inplace=True)
```

[7]:
```
in_combine = []
for college in draft_ttl['College'].value_counts().index[:300]:
    if college in list(combine['College'].unique()):
        in_combine.append(college)
    else:
        print(college + " Not in set")
print(len(in_combine))
# couldnt find matches for these
```

```
Indiana (PA) Not in set
```

```
Oregon Tech Not in set
East. Illinois Not in set
Knoxville Not in set
NW Oklahoma St. Not in set
Wisconsin-LaCrosse Not in set
Boston Univ. Not in set
Robert Morris Not in set
California (PA) Not in set
Long Beach CC Not in set
290
```

[8]:
```
# now check how many drafted player have unique names
print(len(draft_ttl['Player']), len(draft_ttl['Player'].unique())) # 208 repeat␣
 ↪names. add position condition by might not match
unique_names = draft_ttl['Player'].unique()
dups = list(draft_ttl['Player'][draft_ttl['Player'].duplicated()])
drafted_list = []
for i, name in enumerate(combine['Name']):
    if (name not in dups) & (name in list(draft_ttl['Player'])):
        idx = draft_ttl['Player'][draft_ttl['Player'] == name].index[0]
        drafted_list.append(draft_ttl['Pick'][idx])
    elif name in list(draft_ttl['Player']):
        year = combine['Year'][i]
        college = combine['College'][i]
        idx = draft_ttl['Player'][(draft_ttl['Player'] == name) &
                                  (draft_ttl['Year'] == year) &
                                  (draft_ttl['College'] == college)].index
        if len(idx) > 0:
            drafted_list.append(draft_ttl['Pick'][idx[0]])
        else:
            drafted_list.append('Can\'t Find')
    else:
        drafted_list.append('Can\'t Find')
```

```
9558 9350
```

[9]:
```
combine['Pick'] = drafted_list
combine.head()
```

[9]:

	Year	Name	College	POS	Height (in)	Weight (lbs)	\
0	1987	Mike Adams	Arizona St.	CB	69.8	198	
1	1987	John Adickes	Baylor	C	74.8	266	
2	1987	Tommy Agee	Auburn	FB	71.8	217	
3	1987	David Alexander	Tulsa	C	75.0	279	
4	1987	Lyneal Alston	Southern Miss	WR	72.1	202	

	Hand Size (in)	Arm Length (in)	Wonderlic	40 Yard	Bench Press	\

0	8.50	30.50	NaN	4.42	13.0
1	10.25	30.00	NaN	4.97	25.0
2	9.00	30.75	NaN	NaN	15.0
3	10.50	32.75	NaN	5.13	22.0
4	10.00	33.00	NaN	4.64	7.0

	Vert Leap (in)	Broad Jump (in)	Shuttle	3Cone	60Yd Shuttle	Pick
0	32.0	118.0	4.60	NaN	11.91	Can't Find
1	26.5	103.0	4.60	NaN	NaN	154
2	NaN	NaN	NaN	NaN	NaN	Can't Find
3	27.5	105.0	4.33	NaN	NaN	121
4	32.0	114.0	4.52	NaN	11.85	Can't Find

[10]:
```
# count total players with a pick value and compare to 9558 total players in␣
↪drafted list
print(combine[combine['Pick'] != 'Can\'t Find'].shape[0])
# check for missing drafted players
missing_players = []
for name in draft_ttl['Player']:
    if name not in list(combine['Name']):
        missing_players.append(name)
len(missing_players)
```

```
7821
```

[10]:
```
1870
```

[11]:
```
# try to id cases where different names/spellings used
pot_matches = {}
for player in missing_players:
    idx = draft_ttl['Player'][draft_ttl['Player'] == player].index[0]
    col = draft_ttl['College'][idx]
    year = draft_ttl['Year'][idx]
    pot_matches[player] = list(combine[(combine['College'] == col) &␣
↪(combine['Year'] == year)]['Name'])
pot_matches = {k: v for k, v in pot_matches.items() if v}
# there are some lower/upper case matches and a bunch of nicknames....will␣
↪filter for last names too
```

[12]:
```
# find matches on lowercase last name match in addition to college and year
new_pot_matches = {}
for k, v in pot_matches.items():
    last_name = k.split()[-1].lower()
    match_list = []
    for i in v:
        last_name_match = i.split()[-1].lower()
        if last_name_match == last_name:
```

```
            match_list.append(i)
    new_pot_matches[k] = match_list
new_pot_matches = {k: v for k,v in new_pot_matches.items() if v}
```

[13]:
```
# write down mismatches: 'Emmitt Smith': ['Cedric Smith'], 'Al Johnson': ['Ben␣
↪Johnson']
del new_pot_matches['Emmitt Smith']
del new_pot_matches['Al Johnson']
new_pot_matches
for k, v in new_pot_matches.items():
    idx = draft_ttl[draft_ttl['Player'] == k].index[0]
    pick = draft_ttl['Pick'][idx]
    x = v[0]
    idxc = combine[combine['Name'] == x].index[0]
    combine['Pick'][idxc] = pick
```

```
/var/folders/7r/4ts1_nhj4lz2ccky2m4dl_6h0000gn/T/ipykernel_55078/649930420.py:10
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  combine['Pick'][idxc] = pick
```

[14]:
```
# print out new missing
print(combine[combine['Pick'] != 'Can\'t Find'].shape[0])
# print missing pick data for 2000 + draft picks:
print(combine[(combine['Pick'] != 'Can\'t Find') & (combine['Year'] >= 2000)].
↪shape[0])
print(draft_ttl[draft_ttl['Year'] >= 2000].shape[0])
```

```
8166
5332
5871
```

[15]:
```
# at this point I am content with the amount of pick data we got...some picks do␣
↪not go to combine
# we probably missed a couple but would be hard to get many more I think can say␣
↪*most* of "Cant find" when undrafted
combine['Pick'].replace({'Can\'t Find': 'undrafted'}, inplace=True)
print(combine.isna().sum())
# looking at the nas, immediately we should drop the wonderlic and 60yd shuffle␣
↪columns
combine = combine.drop(columns=['Wonderlic', '60Yd Shuttle'])
print()
print()
print(combine.isna().sum())
```

```
Year                   0
Name                   0
College                0
POS                    0
Height (in)            0
Weight (lbs)           0
Hand Size (in)      1309
Arm Length (in)     1625
Wonderlic          13113
40 Yard             1424
Bench Press         3636
Vert Leap (in)      1885
Broad Jump (in)     1998
Shuttle             2841
3Cone               5506
60Yd Shuttle       10302
Pick                   0
dtype: int64
```

```
Year                 0
Name                 0
College              0
POS                  0
Height (in)          0
Weight (lbs)         0
Hand Size (in)    1309
Arm Length (in)   1625
40 Yard           1424
Bench Press       3636
Vert Leap (in)    1885
Broad Jump (in)   1998
Shuttle           2841
3Cone             5506
Pick                 0
dtype: int64
```

[16]:
```
# clean positions
combine['POS'].value_counts()
combine.replace({'NT': 'DT',
                 'CB': 'DB',
                 'DL': 'DT',
                 'OT': 'OL',
                 'OLB': 'LB',
                 'OG': 'OL',
                 'ILB': 'LB',
                 'FS': 'DB',
```

```
                'SS': 'DB',
                'C': 'OL',
                'FB': 'RB',
                'S': 'DB',
                'EDG': 'DE'},
                inplace=True)

# now drop special teams positions: P, K, LS
combine = combine[(combine['POS'] != 'P') & (combine['POS'] != 'K') &␣
 ↪(combine['POS'] != 'LS')]
combine['POS'].value_counts()
```

[16]:
```
DB    2423
OL    2286
WR    1787
LB    1637
RB    1509
DE    1051
DT    1008
TE     768
QB     741
Name: POS, dtype: int64
```

[ ]:
```
combine
```

[17]:
```
# lets coucheck nt complete cases
complete_cases = combine.dropna()
print(complete_cases.shape) #5855 complete cases
print()
print(complete_cases['Pick'].value_counts()) #2428 undrafted vs 3427 drafted
```

```
(5843, 15)

undrafted    2421
89             23
119            20
64             20
150            20
              ...
320             2
259             1
258             1
257             1
321             1
Name: Pick, Length: 262, dtype: int64
```

[ ]:
```
complete_cases.head()
```

[19]:
```
combine.columns
```

[19]:
```
Index(['Year', 'Name', 'College', 'POS', 'Height (in)', 'Weight (lbs)',
       'Hand Size (in)', 'Arm Length (in)', '40 Yard', 'Bench Press',
       'Vert Leap (in)', 'Broad Jump (in)', 'Shuttle', '3Cone', 'Pick'],
      dtype='object')
```

[25]:
```
# rename cols
combine.rename(columns={'Year':'year', 'Name':'name', 'College':'college',
                        'POS': 'pos', 'Height (in)': 'height', 'Weight (lbs)':␣
↪'weight',
                        'Hand Size (in)': 'hand_size', 'Arm Length (in)':␣
↪'arm_length',
                        '40 Yard':'forty', 'Bench Press': 'bench', 'Vert Leap␣
↪(in)': 'vert',
                        'Broad Jump (in)': 'broad'})
```

[25]:

	year	name	college	pos	Height (in)	Weight (lbs)	\
0	1987	Mike Adams	Arizona St.	DB	69.80	198	
1	1987	John Adickes	Baylor	OL	74.80	266	
2	1987	Tommy Agee	Auburn	RB	71.80	217	
3	1987	David Alexander	Tulsa	OL	75.00	279	
4	1987	Lyneal Alston	Southern Miss	WR	72.10	202	
...	...	...	...	..	...	...	
13539	2022	Mykael Wright	Oregon	DB	70.50	173	
13540	2022	Devonte Wyatt	Georgia	DT	74.88	304	
13541	2022	Jalen Wydermyer	Texas A&M	TE	75.88	255	
13542	2022	Nick Zakelj	Fordham	OL	78.13	316	
13543	2022	Bailey Zappe	Western Kentucky	QB	72.50	215	

	Hand Size (in)	Arm Length (in)	40 Yard	bench	Vert Leap (in)	\
0	8.50	30.50	4.42	13.0	32.0	
1	10.25	30.00	4.97	25.0	26.5	
2	9.00	30.75	NaN	15.0	NaN	
3	10.50	32.75	5.13	22.0	27.5	
4	10.00	33.00	4.64	7.0	32.0	
...	...	...	...	...	...	
13539	9.00	30.50	4.57	NaN	NaN	
13540	9.88	32.63	4.77	NaN	29.0	
13541	9.75	33.13	NaN	NaN	NaN	
13542	9.88	32.88	5.13	27.0	28.5	
13543	9.75	31.38	4.88	NaN	30.0	

	Broad Jump (in)	Shuttle	3Cone	Pick
0	118.0	4.60	NaN	undrafted
1	103.0	4.60	NaN	154
2	NaN	NaN	NaN	119

```
3               105.0     4.33    NaN        121
4               114.0     4.52    NaN  undrafted
...               ...      ...    ...        ...
13539             NaN      NaN    NaN  undrafted
13540           111.0      NaN    NaN         28
13541             NaN      NaN    NaN  undrafted
13542           110.0     4.71   7.75        187
13543           109.0     4.40   7.19        137

[13210 rows x 15 columns]
```

[32]:
```
combine.columns = ['year', 'name', 'college', 'pos', 'height', 'weight',␣
↪'hand_size',
                   'arm_length', 'forty', 'bench', 'vert', 'broad_jump',␣
↪'shuttle',
                   '3cone', 'pick']
combine.head()
```

[32]:
```
   year             name        college pos  height  weight  hand_size  \
0  1987       Mike Adams    Arizona St.  DB    69.8     198       8.50
1  1987     John Adickes         Baylor  OL    74.8     266      10.25
2  1987       Tommy Agee         Auburn  RB    71.8     217       9.00
3  1987  David Alexander          Tulsa  OL    75.0     279      10.50
4  1987    Lyneal Alston  Southern Miss  WR    72.1     202      10.00

   arm_length  forty  bench  vert  broad_jump  shuttle  3cone       pick
0       30.50   4.42   13.0  32.0       118.0     4.60    NaN  undrafted
1       30.00   4.97   25.0  26.5       103.0     4.60    NaN        154
2       30.75    NaN   15.0   NaN         NaN      NaN    NaN        119
3       32.75   5.13   22.0  27.5       105.0     4.33    NaN        121
4       33.00   4.64    7.0  32.0       114.0     4.52    NaN  undrafted
```

[33]:
```
# save full dataset
combine.to_csv('full_combine_data')
```

modeling_class

March 16, 2023

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import warnings
import xgboost as xgb
from statsmodels.stats.anova import anova_lm
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import os as os
import math

warnings.filterwarnings('ignore')
```

[4]:
```
import statsmodels
import statsmodels.formula.api as smf
import pandas as pd
import numpy as np
import seaborn as sns
from statsmodels.tools.tools import maybe_unwrap_results
from statsmodels.graphics.gofplots import ProbPlot
from statsmodels.stats.outliers_influence import variance_inflation_factor
import matplotlib.pyplot as plt
from typing import Type

style_talk = 'seaborn-talk'    #refer to plt.style.available
```

[5]:
```
# load data
path = '/Users/dylanjorling/UCLA/412proj/data/'
name = 'full_combine_data'
```

1

```
data = pd.read_csv(path + name, index_col=0)
data.head()
```

[5]:
```
   year             name        college pos  height  weight  hand_size  \
0  1987       Mike Adams    Arizona St.  DB    69.8     198       8.50
1  1987     John Adickes         Baylor  OL    74.8     266      10.25
2  1987       Tommy Agee         Auburn  RB    71.8     217       9.00
3  1987  David Alexander          Tulsa  OL    75.0     279      10.50
4  1987    Lyneal Alston  Southern Miss  WR    72.1     202      10.00

   arm_length  forty  bench  vert  broad_jump  shuttle  3cone       pick
0       30.50   4.42   13.0  32.0       118.0     4.60    NaN  undrafted
1       30.00   4.97   25.0  26.5       103.0     4.60    NaN        154
2       30.75    NaN   15.0   NaN         NaN      NaN    NaN        119
3       32.75   5.13   22.0  27.5       105.0     4.33    NaN        121
4       33.00   4.64    7.0  32.0       114.0     4.52    NaN  undrafted
```

[4]:
```
data.columns = [x.lower() for x in data.columns]
data.head()
```

[4]:
```
   year             name        college pos  height (in)  weight (lbs)  \
0  1987       Mike Adams    Arizona St.  DB         69.8           198
1  1987     John Adickes         Baylor  OL         74.8           266
2  1987       Tommy Agee         Auburn  RB         71.8           217
3  1987  David Alexander          Tulsa  OL         75.0           279
4  1987    Lyneal Alston  Southern Miss  WR         72.1           202

   hand size (in)  arm length (in)  40 yard  bench press  vert leap (in)  \
0            8.50            30.50     4.42         13.0            32.0
1           10.25            30.00     4.97         25.0            26.5
2            9.00            30.75      NaN         15.0             NaN
3           10.50            32.75     5.13         22.0            27.5
4           10.00            33.00     4.64          7.0            32.0

   broad jump (in)  shuttle  3cone       pick
0            118.0     4.60    NaN  undrafted
1            103.0     4.60    NaN        154
2              NaN      NaN    NaN        119
3            105.0     4.33    NaN        121
4            114.0     4.52    NaN  undrafted
```

[5]:
```
data.head()
```

[5]:
```
   year             name        college pos  height (in)  weight (lbs)  \
0  1987       Mike Adams    Arizona St.  DB         69.8           198
1  1987     John Adickes         Baylor  OL         74.8           266
2  1987       Tommy Agee         Auburn  RB         71.8           217
```

```
3  1987  David Alexander          Tulsa  OL         75.0           279
4  1987    Lyneal Alston  Southern Miss  WR         72.1           202

   hand size (in)  arm length (in)  40 yard  bench press  vert leap (in)  \
0            8.50            30.50     4.42         13.0            32.0
1           10.25            30.00     4.97         25.0            26.5
2            9.00            30.75      NaN         15.0             NaN
3           10.50            32.75     5.13         22.0            27.5
4           10.00            33.00     4.64          7.0            32.0

   broad jump (in)  shuttle  3cone       pick
0            118.0     4.60    NaN  undrafted
1            103.0     4.60    NaN        154
2              NaN      NaN    NaN        119
3            105.0     4.33    NaN        121
4            114.0     4.52    NaN  undrafted
```

[6]:
```
data.columns = [x.lower() for x in data.columns]
data.head()
```

[6]:
```
   year             name        college pos  height (in)  weight (lbs)  \
0  1987       Mike Adams    Arizona St.  DB         69.8           198
1  1987     John Adickes         Baylor  OL         74.8           266
2  1987       Tommy Agee         Auburn  RB         71.8           217
3  1987  David Alexander          Tulsa  OL         75.0           279
4  1987    Lyneal Alston  Southern Miss  WR         72.1           202

   hand size (in)  arm length (in)  40 yard  bench press  vert leap (in)  \
0            8.50            30.50     4.42         13.0            32.0
1           10.25            30.00     4.97         25.0            26.5
2            9.00            30.75      NaN         15.0             NaN
3           10.50            32.75     5.13         22.0            27.5
4           10.00            33.00     4.64          7.0            32.0

   broad jump (in)  shuttle  3cone       pick
0            118.0     4.60    NaN  undrafted
1            103.0     4.60    NaN        154
2              NaN      NaN    NaN        119
3            105.0     4.33    NaN        121
4            114.0     4.52    NaN  undrafted
```

[7]:
```
data = data.dropna()
data["pick"][data["pick"] == "undrafted"] = 0
data["pick"] = pd.to_numeric(data["pick"])
data["pick"][data["pick"] > 0] = 1
#drafted = 1
#undrafted = 0
```

```
y = data['pick']
X = data.iloc[:, 4:-1]
#X = data.iloc[:, 3:-1]
#X = pd.get_dummies(X)
print(X)
y
```

```
      height (in)  weight (lbs)  hand size (in)  arm length (in)  40 yard  \
3350        76.40           244           10.00            32.50     5.01
3351        74.60           239            9.00            32.75     4.92
3355        75.50           274            9.50            35.63     5.03
3357        70.60           190           10.25            32.00     4.80
3360        70.00           203            9.50            32.00     4.66
...           ...           ...             ...              ...      ...
13495       76.88           308           10.25            33.88     4.96
13496       71.50           194            9.00            31.13     4.40
13502       78.50           249            9.75            33.00     4.76
13507       78.13           315           10.50            33.88     5.27
13542       78.13           316            9.88            32.88     5.13

      bench press  vert leap (in)  broad jump (in)  shuttle  3cone
3350         15.0            32.0            112.0     4.32   7.45
3351         18.0            33.5            115.0     4.28   7.48
3355         19.0            32.5            110.0     4.83   8.80
3357         12.0            32.5            111.0     4.05   7.14
3360         18.0            36.0            117.0     4.41   7.85
...           ...             ...              ...      ...    ...
13495        21.0            22.5            107.0     4.65   7.40
13496        15.0            37.5            127.0     4.13   6.84
13502        17.0            27.0            120.0     4.41   7.06
13507        25.0            25.0            104.0     4.93   8.31
13542        27.0            28.5            110.0     4.71   7.75

[5843 rows x 10 columns]
```

[7]:
```
3350     1
3351     0
3355     1
3357     1
3360     1
        ..
13495    1
13496    0
13502    1
13507    1
13542    1
Name: pick, Length: 5843, dtype: int64
```

[11]:
```
data.columns = ["year", "name", "college", "pos", "height", "weight",␣
↪"hand_size", "arm_length", "forty", "bench_press", "vert", "broad_jump",␣
↪"shuttle", "cone", "pick"]
```

[13]:
```
data.head()
X.head()
```

[13]:

	height (in)	weight (lbs)	hand size (in)	arm length (in)	40 yard	\
3350	76.40	244	10.00	32.50	5.01	
3351	74.60	239	9.00	32.75	4.92	
3355	75.50	274	9.50	35.63	5.03	
3357	70.60	190	10.25	32.00	4.80	
3360	70.00	203	9.50	32.00	4.66	
...	...	...	...	...	...	
13495	76.88	308	10.25	33.88	4.96	
13496	71.50	194	9.00	31.13	4.40	
13502	78.50	249	9.75	33.00	4.76	
13507	78.13	315	10.50	33.88	5.27	
13542	78.13	316	9.88	32.88	5.13	

	bench press	vert leap (in)	broad jump (in)	shuttle	3cone
3350	15.0	32.0	112.0	4.32	7.45
3351	18.0	33.5	115.0	4.28	7.48
3355	19.0	32.5	110.0	4.83	8.80
3357	12.0	32.5	111.0	4.05	7.14
3360	18.0	36.0	117.0	4.41	7.85
...	...	...	...	...	...
13495	21.0	22.5	107.0	4.65	7.40
13496	15.0	37.5	127.0	4.13	6.84
13502	17.0	27.0	120.0	4.41	7.06
13507	25.0	25.0	104.0	4.93	8.31
13542	27.0	28.5	110.0	4.71	7.75

[5843 rows x 10 columns]

[14]:
```
X.columns = ["height", "weight", "hand_size", "arm_length", "forty",␣
↪"bench_press", "vert", "broad_jump", "shuttle", "cone"]
```

[15]:
```
X.corr()
```

[15]:

	height	weight	hand_size	arm_length	forty	bench_press	\
height	1.000000	0.749966	0.494092	0.723978	0.626795	0.357892	
weight	0.749966	1.000000	0.496161	0.602941	0.883426	0.626132	
hand_size	0.494092	0.496161	1.000000	0.504237	0.411424	0.294874	
arm_length	0.723978	0.602941	0.504237	1.000000	0.468437	0.259697	
forty	0.626795	0.883426	0.411424	0.468437	1.000000	0.459114	
bench_press	0.357892	0.626132	0.294874	0.259697	0.459114	1.000000	

```
vert        -0.438507 -0.673852  -0.275543   -0.316546 -0.754774    -0.314744
broad_jump  -0.424118 -0.717905  -0.274962   -0.279479 -0.795473    -0.368405
shuttle      0.528578  0.748888   0.340291    0.461745  0.772295     0.374785
cone         0.508327  0.771973   0.346123    0.438580  0.809775     0.403321

                vert  broad_jump   shuttle      cone
height      -0.438507   -0.424118  0.528578  0.508327
weight      -0.673852   -0.717905  0.748888  0.771973
hand_size   -0.275543   -0.274962  0.340291  0.346123
arm_length  -0.316546   -0.279479  0.461745  0.438580
forty       -0.754774   -0.795473  0.772295  0.809775
bench_press -0.314744   -0.368405  0.374785  0.403321
vert         1.000000    0.786275 -0.691112 -0.668508
broad_jump   0.786275    1.000000 -0.670406 -0.702932
shuttle     -0.691112   -0.670406  1.000000  0.801462
cone        -0.668508   -0.702932  0.801462  1.000000
```

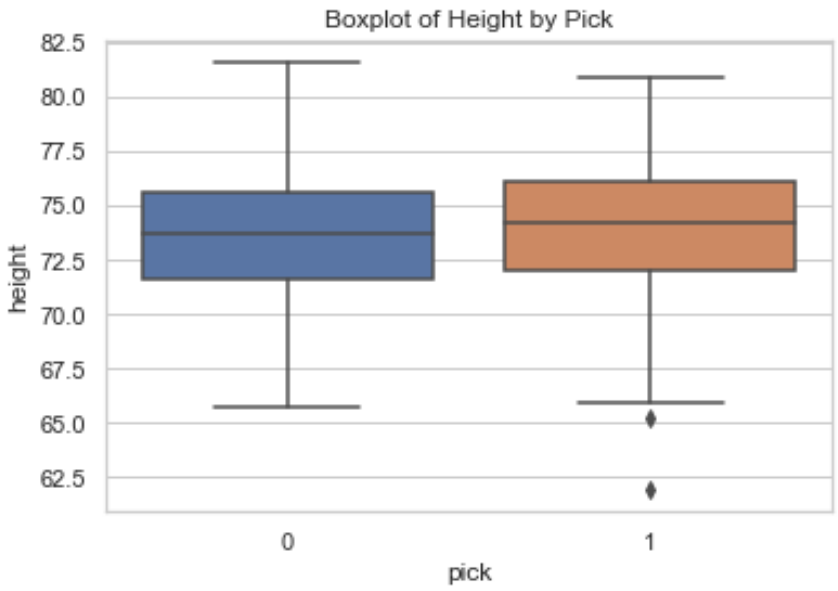
[16]:
```
np.mean(y)
```

[16]:
```
0.5856580523703577
```

[135]:
```
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,␣
↪recall_score, ConfusionMatrixDisplay
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint
from sklearn.metrics import classification_report
```

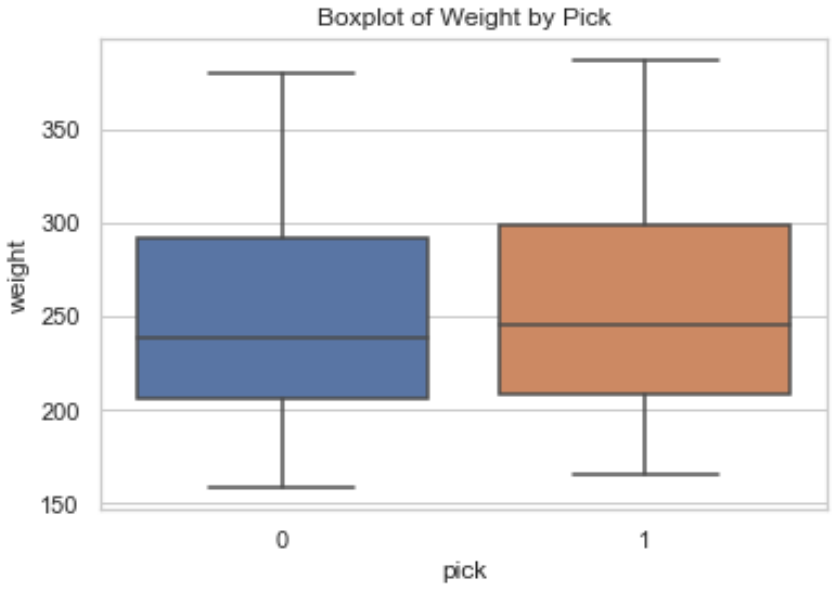
[70]:
```
sns.set(style='whitegrid')
sns.boxplot(x='pick', y='height', data=data).set(title = "Boxplot of Height by␣
↪Pick")
```

[70]:
```
[Text(0.5, 1.0, 'Boxplot of Height by Pick')]
```



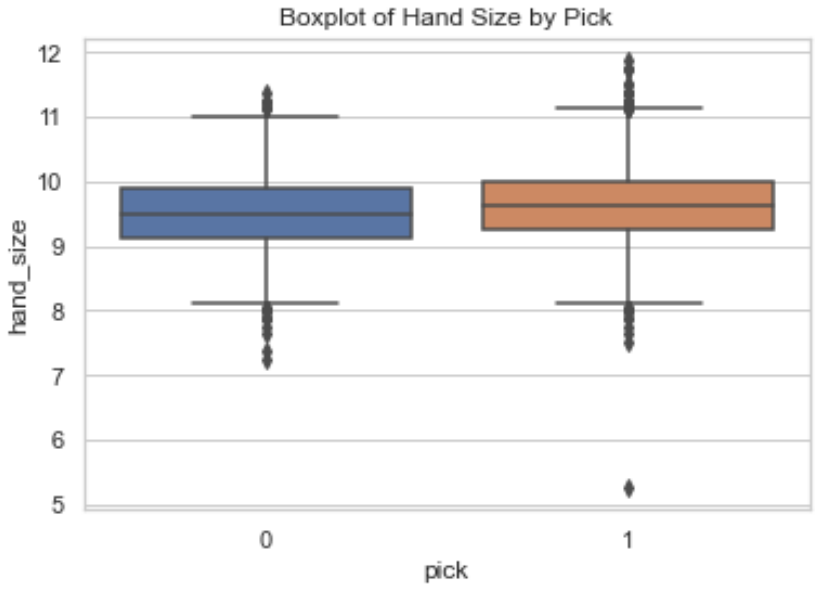
[18]:
```
sns.set(style='whitegrid')
sns.boxplot(x='pick', y='weight', data=data).set(title = "Boxplot of Weight by␣
↪Pick")
```

[18]:
```
[Text(0.5, 1.0, 'Boxplot of Weight by Pick')]
```



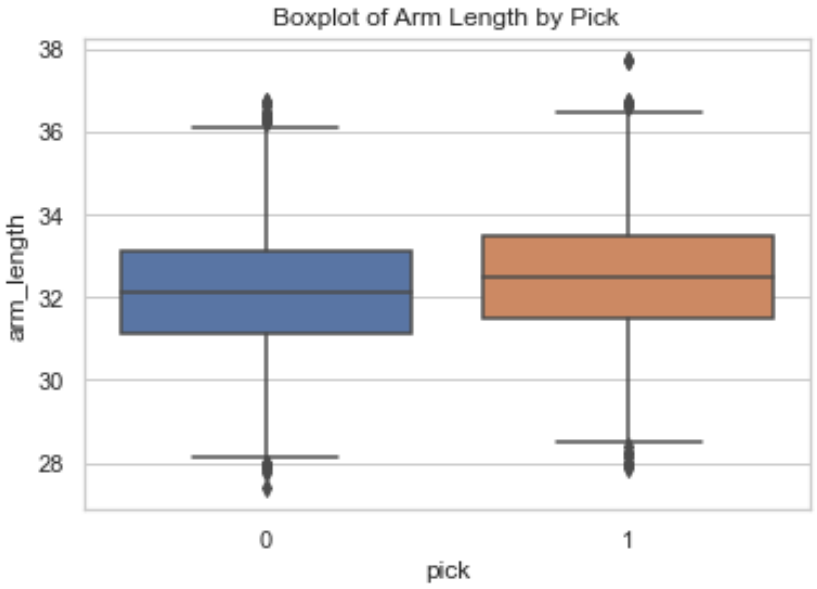
[37]:
```
sns.set(style='whitegrid')
sns.boxplot(x='pick', y='hand_size', data=data).set(title = "Boxplot of Hand␣
↪Size by Pick")
```

[37]:
```
[Text(0.5, 1.0, 'Boxplot of Hand Size by Pick')]
```



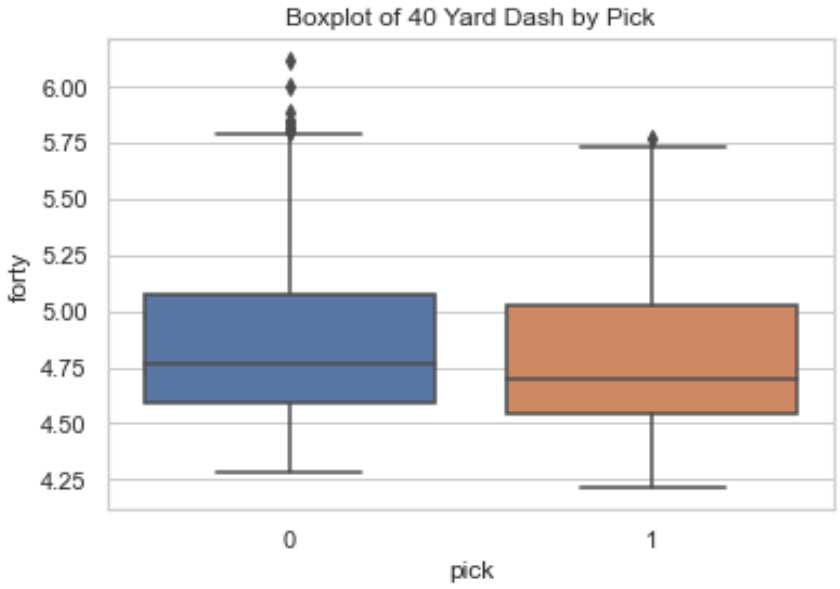
[38]:
```
sns.set(style='whitegrid')
sns.boxplot(x='pick', y='arm_length', data=data).set(title = "Boxplot of Arm␣
↪Length by Pick")
```

[38]:
```
[Text(0.5, 1.0, 'Boxplot of Arm Length by Pick')]
```



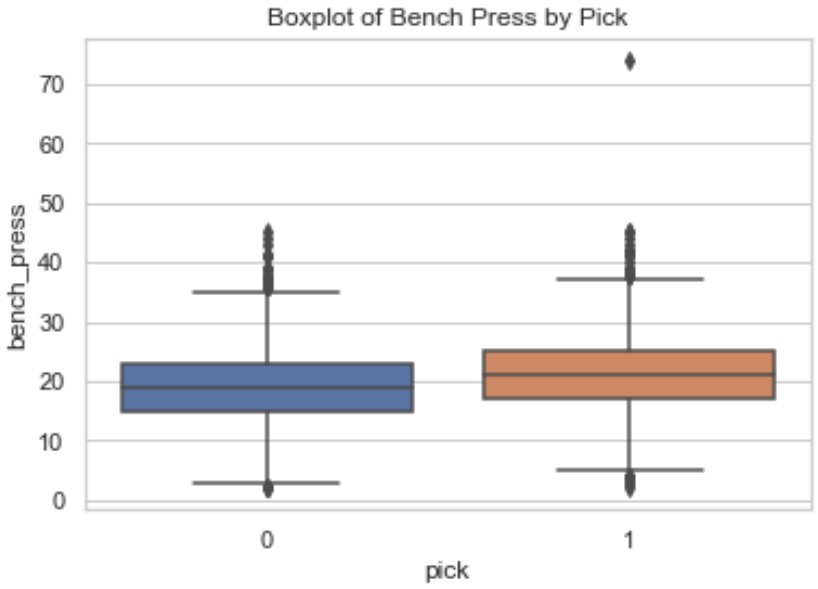
[39]:
```
sns.set(style='whitegrid')
sns.boxplot(x='pick', y='forty', data=data).set(title = "Boxplot of 40 Yard Dash␣
↪by Pick")
```

[39]:
```
[Text(0.5, 1.0, 'Boxplot of 40 Yard Dash by Pick')]
```



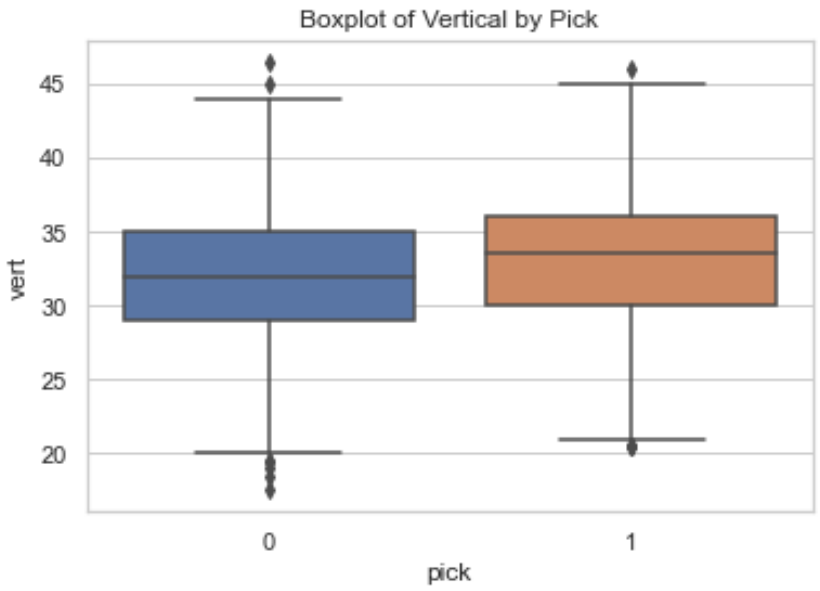
[55]:
```
sns.set(style='whitegrid')
sns.boxplot(x='pick', y='bench_press', data=data).set(title = "Boxplot of Bench␣
↪Press by Pick")
```

[55]:
```
[Text(0.5, 1.0, 'Boxplot of Bench Press by Pick')]
```



[40]:
```
sns.set(style='whitegrid')
sns.boxplot(x='pick', y='vert', data=data).set(title = "Boxplot of Vertical by␣
↪Pick")
```

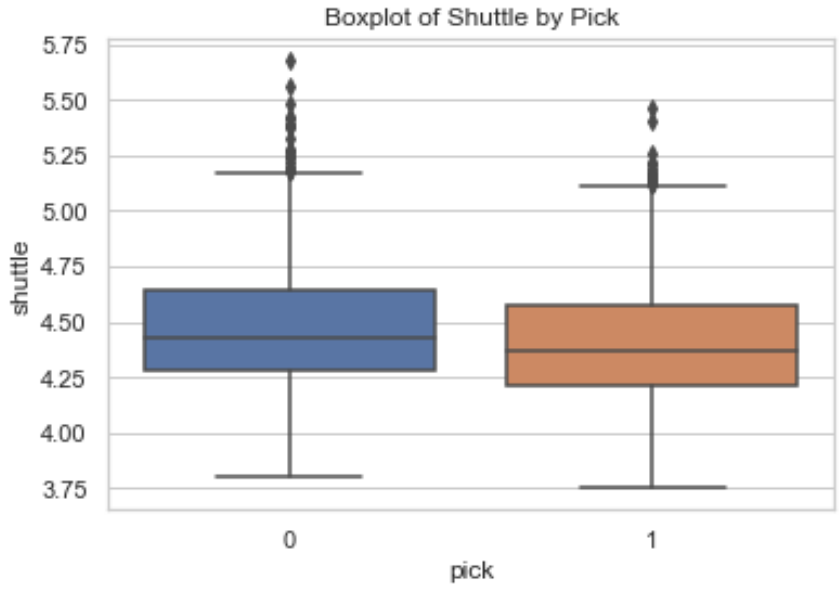
[40]:
```
[Text(0.5, 1.0, 'Boxplot of Vertical by Pick')]
```



[ ]:
```
sns.set(style='whitegrid')
sns.boxplot(x='pick', y='broad_jump', data=data).set(title = "Boxplot of Broad␣
↪Jump by Pick")
```

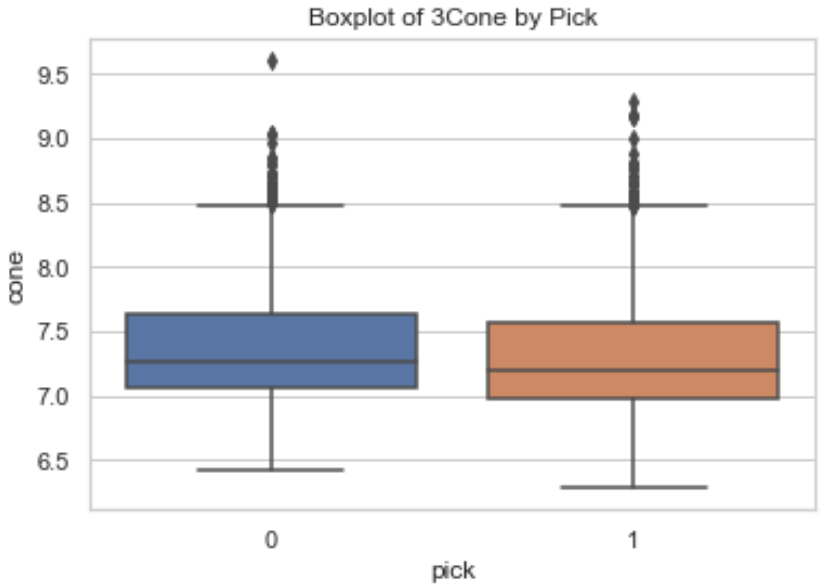
[42]:
```
sns.set(style='whitegrid')
sns.boxplot(x='pick', y='shuttle', data=data).set(title = "Boxplot of Shuttle by␣
↪Pick")
```

[42]:
```
[Text(0.5, 1.0, 'Boxplot of Shuttle by Pick')]
```



[43]:
```
sns.set(style='whitegrid')
sns.boxplot(x='pick', y='cone', data=data).set(title = "Boxplot of 3Cone by␣
↪Pick")
```

[43]:
```
[Text(0.5, 1.0, 'Boxplot of 3Cone by Pick')]
```



1 Condense Logistic Model

[205]:
```
new = X.loc[:, X.columns.isin(["forty", #56.8
                               "weight",#, 62.9
                               "shuttle"#,65.9
                               #"cone"#,66.05
                              #"height", #65.9
                              #"arm_length"#, 66.05
                              #"vert"#, 65.7
                              #"bench_press"#,65.2
                              #"broad_jump"#, 65.0
                              #"hand_size" 64.3
                              ])]


new_train, new_test, y_train, y_test = train_test_split(new, y, test_size=0.3,␣
 ↪random_state=42)
```

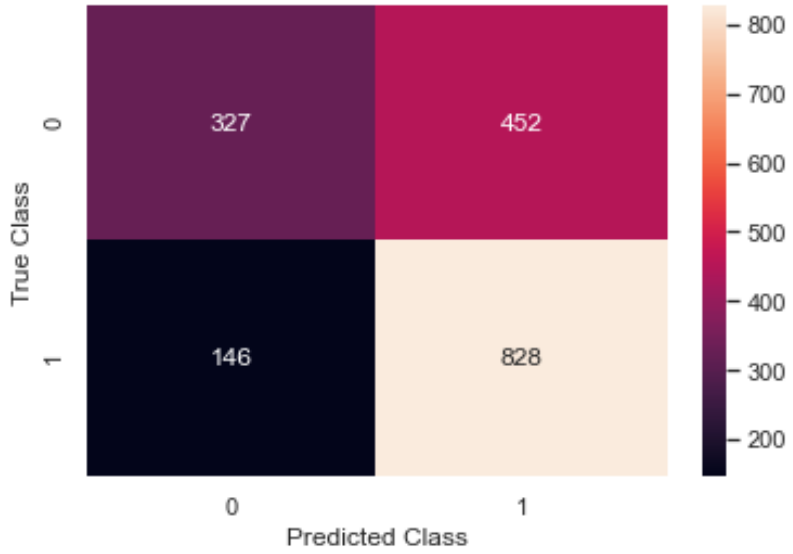
[206]:
```
# Logistic Regression with CV
new_model_cv = LogisticRegressionCV(cv=10, random_state=42)
new_model_cv.fit(new_train, y_train)
new_model_cv.score(new_test, y_test)
```

[206]: 0.6588705077010839

[207]:
```
y_pred = new_model_cv.predict(new_test)
cf_matrix = confusion_matrix(y_test, y_pred)
print(cf_matrix)
import seaborn as sns
conf = sns.heatmap(cf_matrix, annot=True, fmt=".0f")
conf.set(xlabel='Predicted Class', ylabel='True Class')
```

```
[[327 452]
 [146 828]]
```

[207]: [Text(0.5, 12.5, 'Predicted Class'), Text(30.5, 0.5, 'True Class')]



[208]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.69      0.42      0.52       779
           1       0.65      0.85      0.73       974

    accuracy                           0.66      1753
   macro avg       0.67      0.63      0.63      1753
weighted avg       0.67      0.66      0.64      1753
```

1.1 Odds

[162]:
```
odds = [math.exp(x) for x in new_model_cv.coef_[0]]
print(new_train.columns)
print(odds)
```

```
Index(['weight', 'forty', 'shuttle'], dtype='object')
[1.0374082566598295, 0.017438080181058697, 0.10777813172346859]
```

[163]:
```
new_model.coef_[0]
```

[163]:
```
array([ 0.03634102, -3.98757502, -2.24177384])
```

[164]:
```
new_train.columns
```

[164]:
```
Index(['weight', 'forty', 'shuttle'], dtype='object')
```

From our reduced model we get coefficients of 0.0363 for weight, -3.9875 for forty yard dash, and -2.2418 in shuttle. After converting these coefficients to odds we get that for every 1 pound increase in weight, the odds of being drafted to the NFL increase by 3.74%, for every 1 unit increase in Forty Yard Dash Time the odds of being drafted to the NFL decrease by 97.1%, and for every 1 unit increase in Shuttle Time the odds of being drafted to the NFL decrease by 89.6%

2 Full Logistic

[194]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

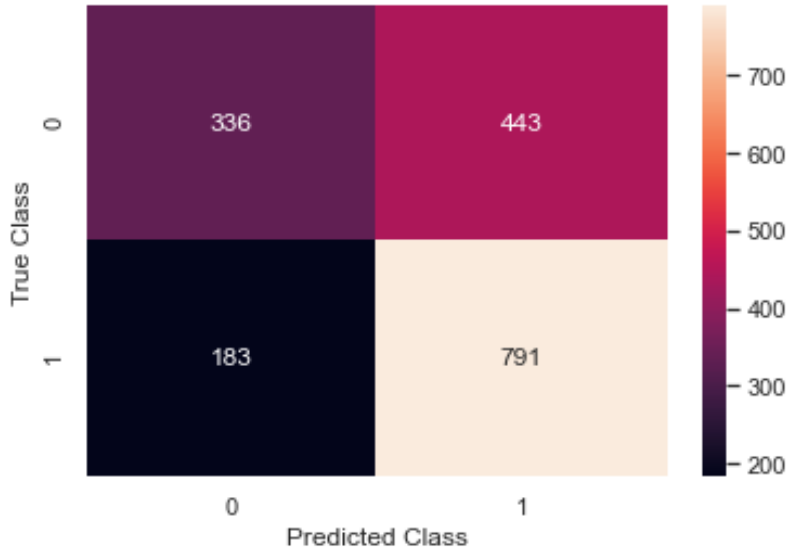
[195]:
```
# Logistic Regression with CV
model_cv = LogisticRegressionCV(cv=10, random_state=42)
model_cv.fit(X_train, y_train)
print(model_cv.score(X_test, y_test))
y_pred = model_cv.predict(X_test)
```

```
0.6428978893325727
```

[201]:
```
cf_matrix = confusion_matrix(y_test, y_pred)
print(cf_matrix)
import seaborn as sns
conf = sns.heatmap(cf_matrix, annot=True, fmt=".0f")
conf.set(xlabel='Predicted Class', ylabel='True Class')

print(classification_report(y_test, y_pred))
```

```
[[336 443]
 [183 791]]
              precision    recall  f1-score   support
```

0	0.65	0.43	0.52	779
1	0.64	0.81	0.72	974
accuracy			0.64	1753
macro avg	0.64	0.62	0.62	1753
weighted avg	0.64	0.64	0.63	1753



[204]:
```
791/(791+443) #of the predicted positive values it only identifies 64% correctly
791/(791+183) # of the drafted players it identifies 81% of them
```

[204]:
```
0.6410048622366289
```

2.1 RF

[209]:
```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

[209]:
```
RandomForestClassifier()
```

[210]:
```
y_pred = rf.predict(X_test)
```

[211]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6423274386765545
```

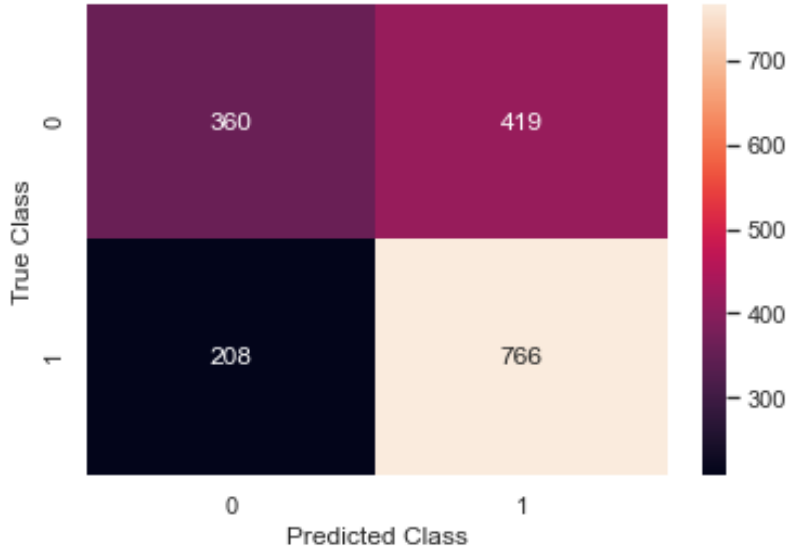
[212]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.63      0.46      0.53       779
           1       0.65      0.79      0.71       974

    accuracy                           0.64      1753
   macro avg       0.64      0.62      0.62      1753
weighted avg       0.64      0.64      0.63      1753
```

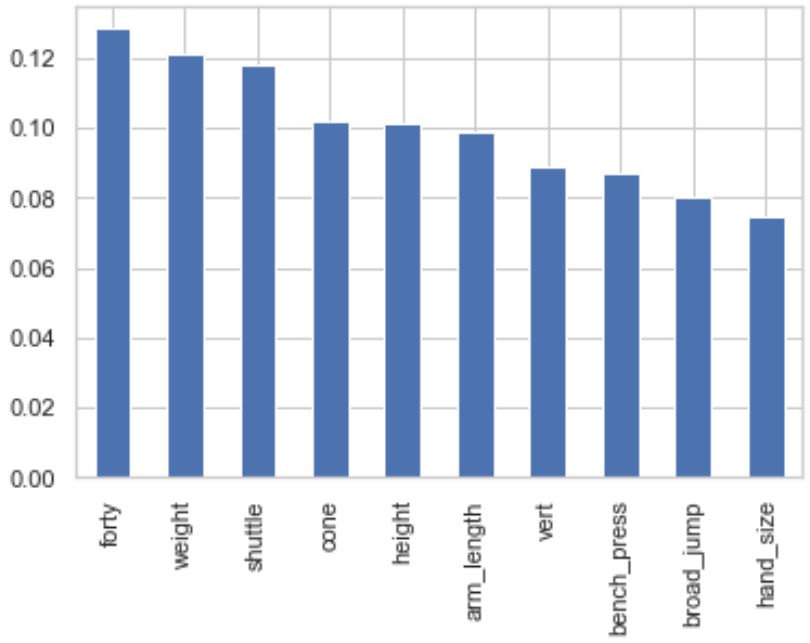
[213]:
```
cf_matrix1 = confusion_matrix(y_test, y_pred)
print(cf_matrix1)
import seaborn as sns
conf1 = sns.heatmap(cf_matrix1, annot=True, fmt=".0f")
conf1.set(xlabel='Predicted Class', ylabel='True Class')
```

```
[[360 419]
 [208 766]]
```

[213]:
```
[Text(0.5, 12.5, 'Predicted Class'), Text(30.5, 0.5, 'True Class')]
```



[182]:
```
feature_importances = pd.Series(rf.feature_importances_, index=X_train.columns).
↪sort_values(ascending=False)

# Plot a simple bar chart
feature_importances.plot.bar();
```



[214]:
```
fws = X[["forty", "weight", "shuttle"]]
new_train2, new_test2, y_train2, y_test2 = train_test_split(fws, y, test_size=0.
↪3, random_state=42)
```

[215]:
```
rf2 = RandomForestClassifier()
rf2.fit(new_train2, y_train2)
y_pred2 = rf2.predict(new_test2)
accuracy = accuracy_score(y_test2, y_pred2)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6383342840844267
```

[216]:
```
print(classification_report(y_test2,y_pred2))
```

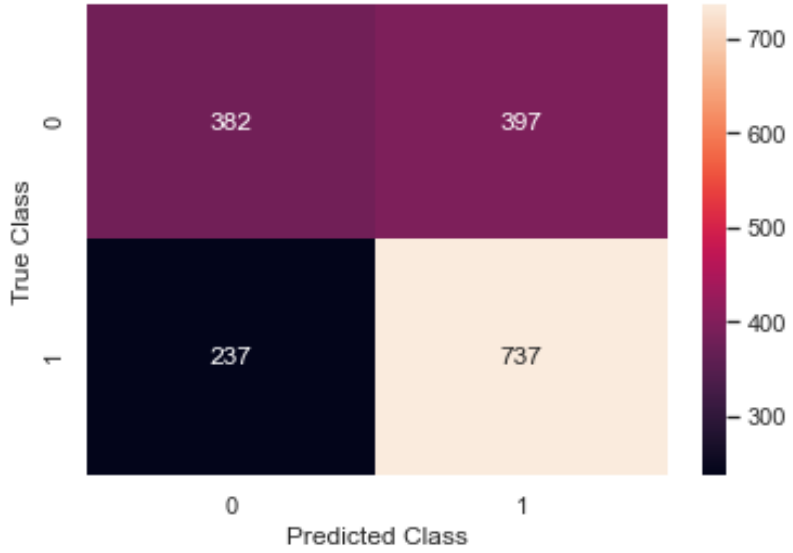
```
              precision    recall  f1-score   support
```

```
                0       0.62      0.49      0.55       779
                1       0.65      0.76      0.70       974

         accuracy                           0.64      1753
        macro avg       0.63      0.62      0.62      1753
     weighted avg       0.64      0.64      0.63      1753
```

[217]:
```
cf_matrix2 = confusion_matrix(y_test2, y_pred2)
print(cf_matrix2)
import seaborn as sns
conf2 = sns.heatmap(cf_matrix2, annot=True, fmt=".0f")
conf2.set(xlabel='Predicted Class', ylabel='True Class')
```

```
[[382 397]
 [237 737]]
```

[217]: `[Text(0.5, 12.5, 'Predicted Class'), Text(30.5, 0.5, 'True Class')]`



modeling_reg

March 16, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import warnings
import xgboost as xgb
from statsmodels.stats.anova import anova_lm
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from statsmodels.gam.api import GLMGam, BSplines


# import diagnostics from smlearn website
import SM_plots as SM

warnings.filterwarnings('ignore')
```

[2]:
```
# load data
path = '/Users/dylanjorling/UCLA/412proj/data/'
name = 'full_combine_data'
data = pd.read_csv(path + name, index_col=0)
data.shape
```

[2]:
```
(13210, 15)
```

[3]:
```
################## EDA Plots ##################
combine_complete = data.dropna()
print(combine_complete.shape)
combine_complete.head()
combine_complete = combine_complete[combine_complete['pick'] != 'undrafted']
```

```
#combine_complete = combine_complete[combine_complete['pos'] == 'LB'] #filter␣
 ↪for specific positions
#combine_complete = combine_complete[combine_complete['year'] >= 2010] #filter␣
 ↪for specific year...didn't alter much
combine_complete['pick'] = combine_complete['pick'].astype(int)
combine_complete = combine_complete[combine_complete['pick'] < 300]
combine_complete = combine_complete[combine_complete['bench'] < 74]
pos_one_hot = combine_complete['pos'].str.get_dummies()


y = combine_complete['pick']
X = combine_complete.iloc[:, 4:-1]
X = pd.concat([X, pos_one_hot], axis=1)
null_sd = y.std()
print(null_sd)

combine_complete.head()
```

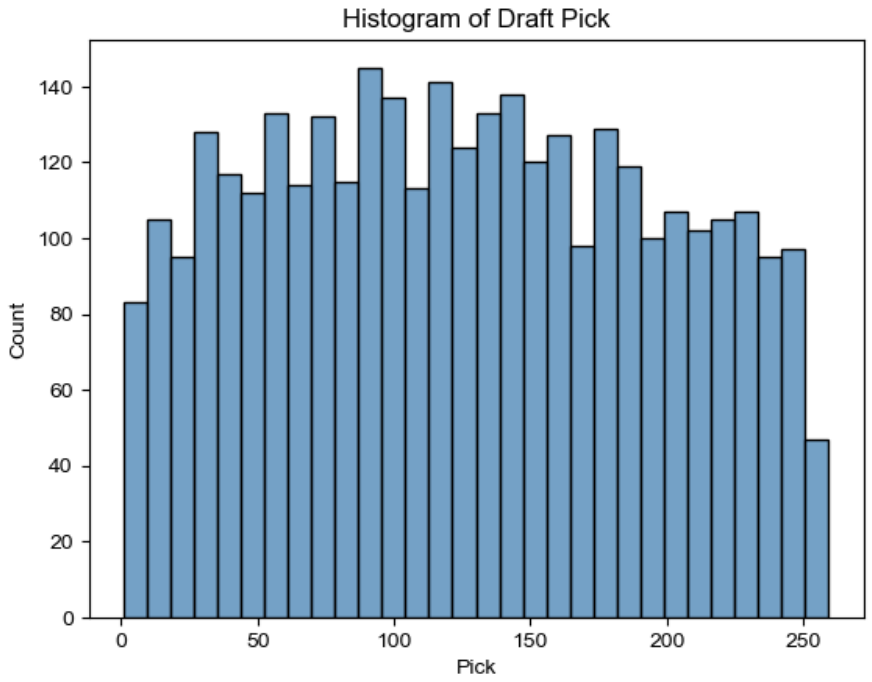
```
(5843, 15)
69.80294730041324
```

[3]:

	year	name	college	pos	height	weight	hand_size	\
3350	1997	John Allred	USC	TE	76.4	244	10.00	
3355	1997	Duane Ashman	Virginia	DE	75.5	274	9.50	
3357	1997	Raymond Austin	Tennessee	DB	70.6	190	10.25	
3360	1997	Antonio Banks	Virginia Tech	DB	70.0	203	9.50	
3361	1997	Ronde Barber	Virginia	DB	69.4	185	9.50	

	arm_length	forty	bench	vert	broad_jump	shuttle	3cone	pick
3350	32.50	5.01	15.0	32.0	112.0	4.32	7.45	38
3355	35.63	5.03	19.0	32.5	110.0	4.83	8.80	161
3357	32.00	4.80	12.0	32.5	111.0	4.05	7.14	145
3360	32.00	4.66	18.0	36.0	117.0	4.41	7.85	113
3361	31.63	4.68	14.0	34.5	118.0	4.46	7.22	66

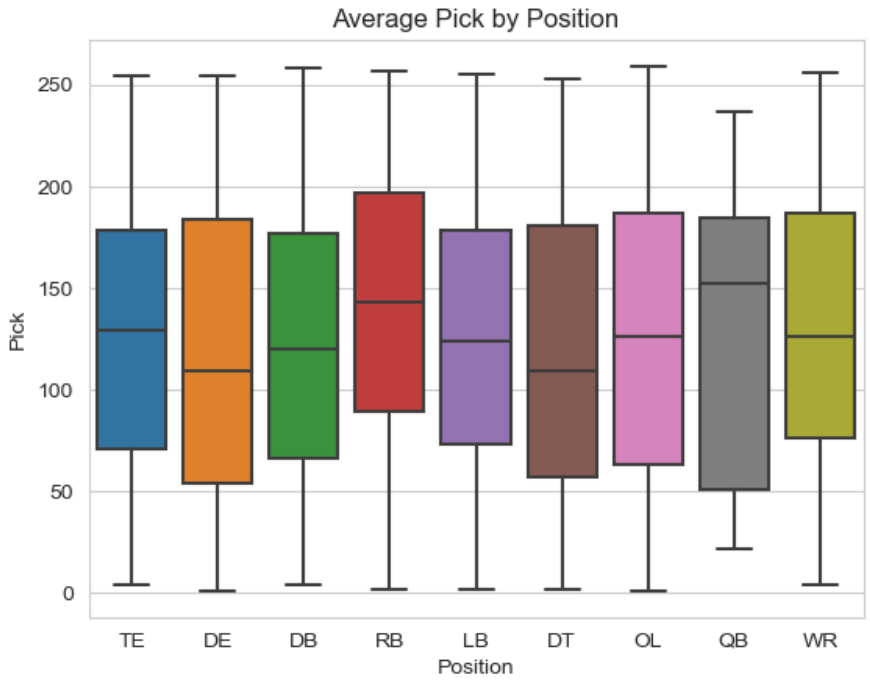
[5]:
```
ax = sns.histplot(combine_complete, x='pick', color="steelblue", bins=30)
ax.set(xlabel='Pick', ylabel='Count', title='Histogram of Draft Pick')
sns.set_style("whitegrid")
plt.show()
# underlying distribution is uniform
```



[6]:
```
pos_means = combine_complete.groupby('pos').mean()['pick']
ax = sns.boxplot(data=combine_completex=pos_means, y=pos_means.index, orient='h')
ax.set(xlabel='Pick', ylabel='Position', title='Average Pick by Position')
sns.set_style("whitegrid")
plt.show()
```

```
  File "/var/folders/z7/1q6khp0n3xq0p_97cq1hvpd40000gn/T/ipykernel_2929/292387673
↪py", line 2
    ax = sns.boxplot(data=combine_completex=pos_means, y=pos_means.index,␣
↪orient='h')
                                           ^
SyntaxError: invalid syntax
```

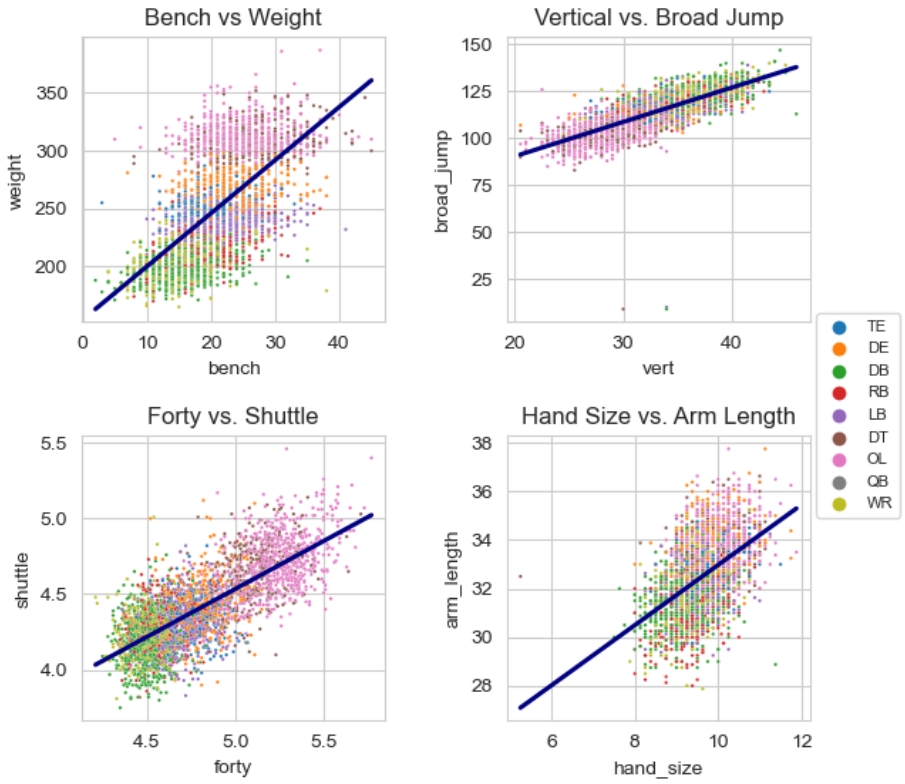
[7]:
```
pos_means = combine_complete.groupby('pos').mean()['pick']
ax = sns.boxplot(data=combine_complete, x='pos', y='pick')
ax.set(xlabel='Position', ylabel='Pick', title='Average Pick by Position')
sns.set_style("whitegrid")
plt.show()
```



[101]:
```
# This is a plot of some of the more predictive variables

fig, axs = plt.subplots(nrows=2, ncols=2)
sns.scatterplot(data=combine_complete,
                x='bench',
                y='weight',
                hue='pos',
                s=3,
                ax=axs[0,0],
               legend=False)
axs[0, 0].set(xlabel='Bench Press Reps', ylabel='Weight', title='Bench vs␣
 ↪Weight')
sns.regplot(data=combine_complete,
            x='bench',
            y='weight',
            scatter=False,
            ci=None,
            color="navy",
            ax=axs[0,0])
sns.set_style("whitegrid")
```

```
sns.scatterplot(data=combine_complete,
                x='vert',
                y='broad_jump',
                hue='pos',
                s=3,
                ax=axs[0,1],
               legend=False)
axs[0, 1].set(xlabel='Vert', ylabel='Broad Jump', title='Vertical vs. Broad␣
↪Jump')
sns.regplot(data=combine_complete,
            x='vert',
            y='broad_jump',
            scatter=False,
            ci=None,
            color="navy",
            ax=axs[0,1])
sns.set_style("whitegrid")

sns.scatterplot(data=combine_complete,
                x='forty',
                y='shuttle',
                hue='pos',
                s=3,
                ax=axs[1,0],
               legend=False)
axs[1, 0].set(xlabel='Forty', ylabel='Shuttle', title='Forty vs. Shuttle')
sns.regplot(data=combine_complete,
            x='forty',
            y='shuttle',
            scatter=False,
            ci=None,
            color="navy",
            ax=axs[1,0])
sns.set_style("whitegrid")

sns.scatterplot(data=combine_complete,
                x='hand_size',
                y='arm_length',
                hue='pos',
                s=3,
                ax=axs[1,1],
               legend=False)
axs[1, 1].set(xlabel='Hand Size', ylabel='Arm Length', title='Hand Size vs. Arm␣
↪Length')

sns.regplot(data=combine_complete,
```

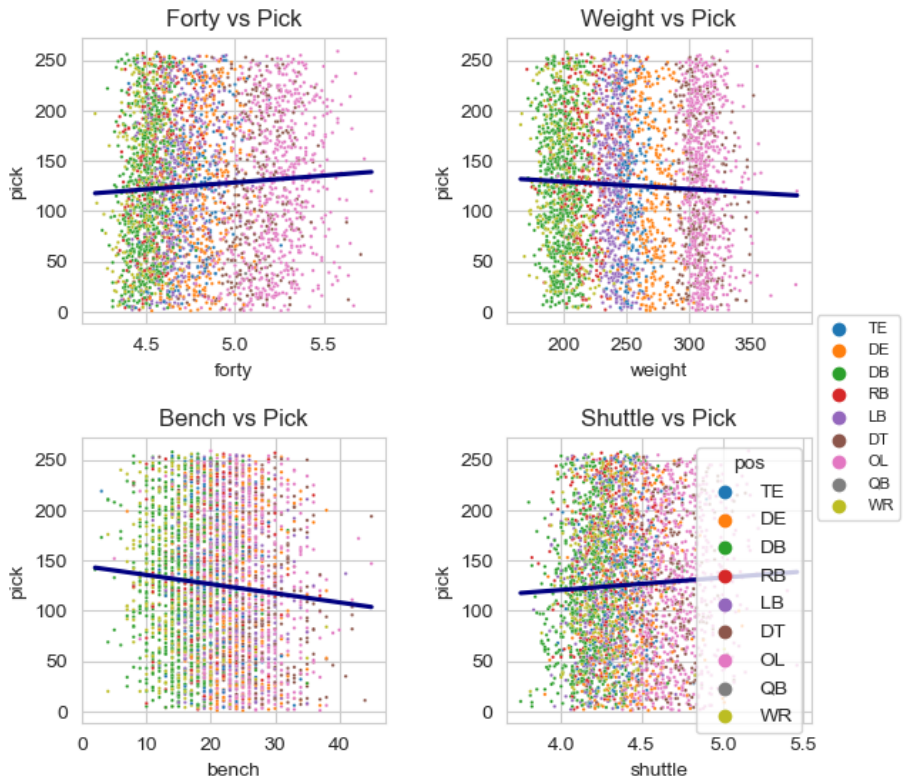
```
            x='hand_size',
            y='arm_length',
            scatter=False,
            ci=None,
            color="navy",
            ax=axs[1,1])
sns.set_style("whitegrid")

#handles, labels = axs[1,1].get_legend_handles_labels()
fig.legend(handles=handles, labels=labels, loc='center right', fontsize = 8)
plt.gca().get_legend_handles_labels()
plt.subplots_adjust(left=0.1,
                    bottom=0.5,
                    right=0.9,
                    top=1.5,
                    wspace=0.4,
                    hspace=0.4)
fig.show()
```



6

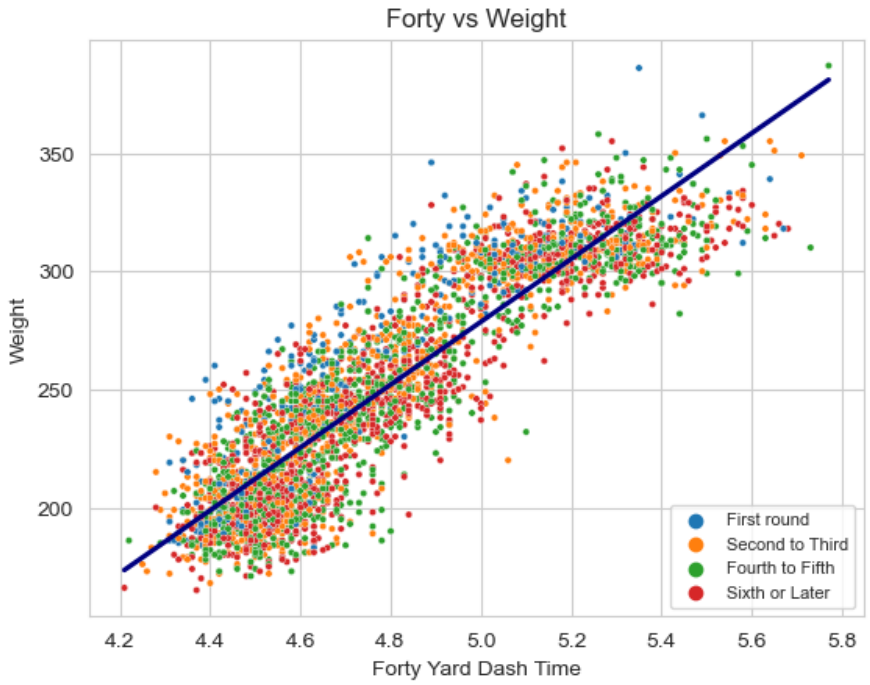
[94]:
```
# This is a plot of some of the more predictive variables

fig, axs = plt.subplots(nrows=2, ncols=2)
sns.scatterplot(data=combine_complete,
                x='forty',
                y='pick',
                hue='pos',
                s=3,
                ax=axs[0,0],
               legend=False)
axs[0, 0].set(xlabel='forty yard dash time', ylabel='Pick', title='Forty vs␣
 ↪Pick')
sns.regplot(data=combine_complete,
            x='forty',
            y='pick',
            scatter=False,
            ci=None,
            color="navy",
            ax=axs[0,0])
sns.set_style("whitegrid")

sns.scatterplot(data=combine_complete,
                x='weight',
                y='pick',
                hue='pos',
                s=3,
                ax=axs[0,1],
               legend=False)
axs[0, 1].set(xlabel='weight', ylabel='Pick', title='Weight vs Pick')
sns.regplot(data=combine_complete,
            x='weight',
            y='pick',
            scatter=False,
            ci=None,
            color="navy",
            ax=axs[0,1])
sns.set_style("whitegrid")

sns.scatterplot(data=combine_complete,
                x='bench',
                y='pick',
                hue='pos',
                s=3,
                ax=axs[1,0],
               legend=False)
```

```
axs[1, 0].set(xlabel='bench press reps', ylabel='Pick', title='Bench vs Pick')
sns.regplot(data=combine_complete,
            x='bench',
            y='pick',
            scatter=False,
            ci=None,
            color="navy",
            ax=axs[1,0])
sns.set_style("whitegrid")

sns.scatterplot(data=combine_complete,
                x='shuttle',
                y='pick',
                hue='pos',
                s=3,
                ax=axs[1,1],
               legend=True)
axs[1, 1].set(xlabel='shuttle time', ylabel='Pick', title='Shuttle vs Pick')

sns.regplot(data=combine_complete,
            x='shuttle',
            y='pick',
            scatter=False,
            ci=None,
            color="navy",
            ax=axs[1,1])
sns.set_style("whitegrid")

handles, labels = axs[1,1].get_legend_handles_labels()
fig.legend(handles=handles, labels=labels, loc='center right', fontsize = 8)
plt.gca().get_legend_handles_labels()
plt.subplots_adjust(left=0.1,
                    bottom=0.5,
                    right=0.9,
                    top=1.5,
                    wspace=0.4,
                    hspace=0.4)
fig.show()
```



[46]:
```
# This is weight v forty time binned by pick intervals
combine_round_est = combine_complete
combine_round_est['Estimated Round'] = pd.cut(
    combine_round_est['pick'],
    bins=[0, 32, 100, 175, 350],
    labels=['First round', 'Second to Third',
            'Fourth to Fifth', 'Sixth or Later'])

ax = sns.scatterplot(data=combine_round_est,
                     x='forty',
                     y='weight',
                     hue='Estimated Round',
                     s=10)
sns.regplot(data=combine_round_est,
            x='forty',
```

```
            y='weight',
            scatter=False,
            ci=None,
            color="navy",
            ax=ax)

ax.set(xlabel='Forty Yard Dash Time', ylabel='Weight', title='Forty vs Weight')
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels, loc='lower right', fontsize = 8)
sns.set_style("whitegrid")
plt.show()
```



[47]:
```
################## Linear Model w/ model selection ##################
# set up basic linear model with complete cases and to simplify further, filter␣
 ↪out undrafted players
model = sm.OLS(y, sm.add_constant(X))
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
```

Dep. Variable:	pick	R-squared:	0.088
Model:	OLS	Adj. R-squared:	0.084
Method:	Least Squares	F-statistic:	18.31
Date:	Thu, 02 Mar 2023	Prob (F-statistic):	3.50e-56
Time:	23:27:12	Log-Likelihood:	-19203.
No. Observations:	3418	AIC:	3.844e+04
Df Residuals:	3399	BIC:	3.856e+04
Df Model:	18		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-233.3898	74.944	-3.114	0.002	-380.330	-86.450
height	1.8195	0.908	2.003	0.045	0.038	3.600
weight	-0.7369	0.117	-6.279	0.000	-0.967	-0.507
hand_size	-3.9821	2.331	-1.708	0.088	-8.553	0.589
arm_length	-5.9567	1.235	-4.824	0.000	-8.378	-3.535
forty	115.4925	10.760	10.733	0.000	94.396	136.589
bench	-0.3985	0.258	-1.543	0.123	-0.905	0.108
vert	-0.2399	0.474	-0.507	0.612	-1.168	0.688
broad_jump	-0.2771	0.210	-1.320	0.187	-0.688	0.134
shuttle	35.6163	8.272	4.306	0.000	19.398	51.834
3cone	0.7358	5.257	0.140	0.889	-9.572	11.044
DB	-30.0442	8.843	-3.398	0.001	-47.382	-12.706
DE	-19.8457	10.113	-1.962	0.050	-39.674	-0.018
DT	-33.2663	11.393	-2.920	0.004	-55.604	-10.928
LB	-16.7721	8.838	-1.898	0.058	-34.101	0.557
OL	-45.8938	12.312	-3.728	0.000	-70.034	-21.754
QB	-45.7701	24.646	-1.857	0.063	-94.093	2.552
RB	-7.6332	8.077	-0.945	0.345	-23.469	8.203
TE	-15.4542	10.342	-1.494	0.135	-35.731	4.822
WR	-18.7102	9.329	-2.006	0.045	-37.002	-0.419

Omnibus:	448.685	Durbin-Watson:	1.922
Prob(Omnibus):	0.000	Jarque-Bera (JB):	125.679
Skew:	0.154	Prob(JB):	5.12e-28
Kurtosis:	2.113	Cond. No.	1.25e+18

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.87e-28. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

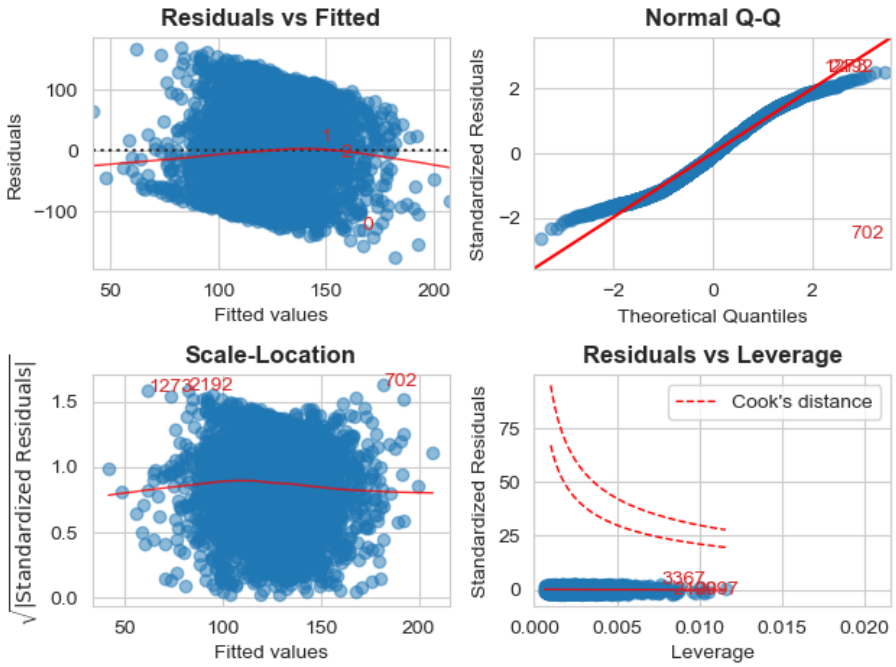
[95]:
```
# Backward Elimination to select best model
y = combine_complete['pick']
X = combine_complete.iloc[:, 4:-1]
X = pd.concat([X, pos_one_hot], axis=1)
cols = X.columns

for var in cols:
    model = sm.OLS(y, sm.add_constant(X))
    results = model.fit()
    p_values = pd.Series(results.pvalues, name='pvalues')
    max_p = p_values[p_values==p_values.max()]
    if max_p[0] > 0.05:
        drop_col = max_p.index[0]
        X = X.drop(columns=drop_col)
    else:
        print(results.summary())
        break
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   pick   R-squared:                       0.084
Model:                            OLS   Adj. R-squared:                  0.082
Method:                 Least Squares   F-statistic:                     39.27
Date:                Fri, 03 Mar 2023   Prob (F-statistic):           2.81e-60
Time:                        07:25:40   Log-Likelihood:                -19210.
No. Observations:                3418   AIC:                         3.844e+04
Df Residuals:                    3409   BIC:                         3.849e+04
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -222.0558     49.765     -4.462      0.000    -319.628    -124.484
weight        -0.6549      0.088     -7.462      0.000      -0.827      -0.483
arm_length    -6.3900      1.051     -6.079      0.000      -8.451      -4.329
forty        119.8296      9.628     12.445      0.000     100.952     138.708
bench         -0.5226      0.249     -2.102      0.036      -1.010      -0.035
shuttle       37.9831      7.230      5.253      0.000      23.807      52.159
DB           -15.0889      3.667     -4.115      0.000     -22.279      -7.899
DT           -18.5718      5.954     -3.119      0.002     -30.245      -6.898
OL           -28.7441      5.753     -4.996      0.000     -40.024     -17.465
==============================================================================
Omnibus:                      486.318   Durbin-Watson:                   1.925
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              129.009
Skew:                           0.147   Prob(JB):                     9.68e-29
Kurtosis:                       2.095   Cond. No.                     1.13e+04
```

```
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.13e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[143]:
```
ols = SM.Linear_Reg_Diagnostic(results)
```

[156]:
```
fig, axs = plt.subplots(nrows=2, ncols=2)

ols.residual_plot(ax=axs[0,0])
ols.qq_plot(ax=axs[0,1])
ols.scale_location_plot(ax=axs[1,0])
ols.leverage_plot(ax=axs[1,1])

fig.tight_layout(pad=1.0)
plt.show()
```

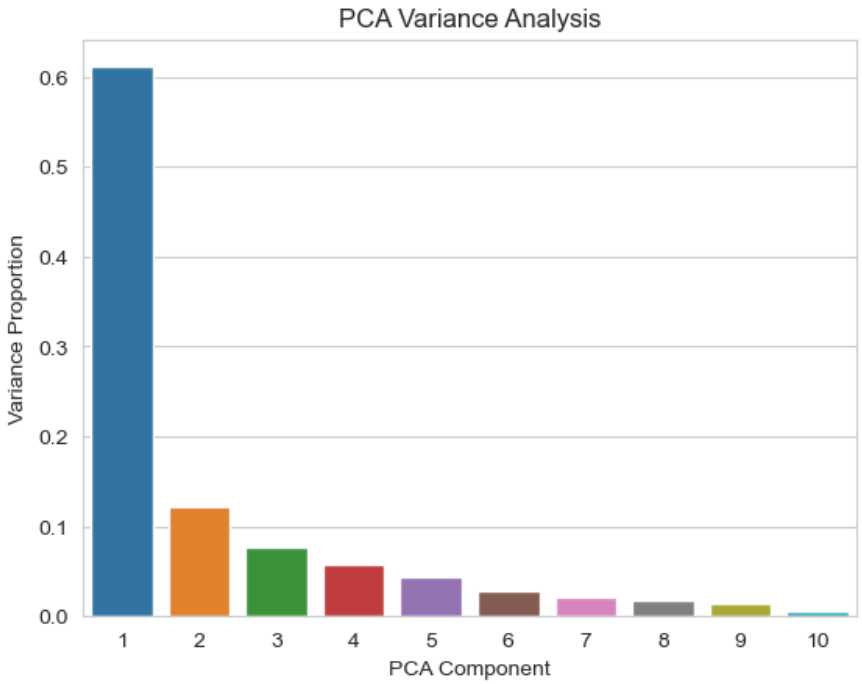


[ ]:
```
### Try robust ###
```

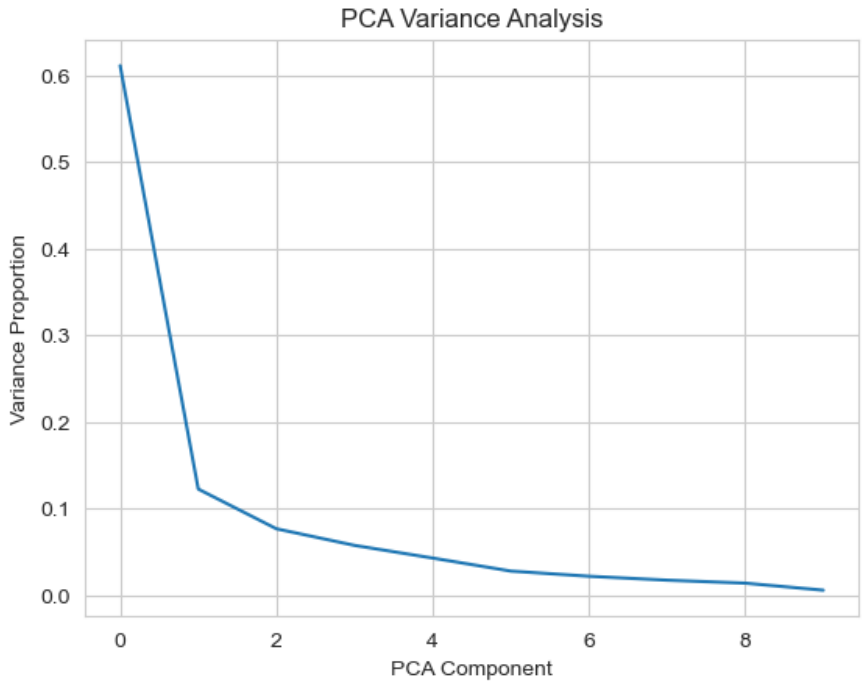
[96]:
```
################## Linear Model ##################
# will use the model generated from backward selection above
y = combine_complete['pick']
X = combine_complete.iloc[:, 4:-1]
X = pd.concat([X, pos_one_hot], axis=1)
X = X[['weight', 'arm_length','forty', 'bench', 'shuttle', 'DB', 'DT', 'OL']]

cv = KFold(n_splits=3, shuffle=True, random_state=100)
reg = LinearRegression()
scores = cross_val_score(reg, X, y, scoring='neg_mean_squared_error', cv=cv)
rmse = (scores.mean()*-1)**0.5

print(rmse)
```

```
67.01796200690795
```

[8]:
```
################## Predictive PCA/PCR ##################
y = combine_complete['pick']
X = combine_complete.iloc[:, 4:-1] # do not include posiiton for PCA
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state=100)

scaler = StandardScaler()
X_train_norm = scaler.fit_transform(X_train.values)
pcr = PCA()
pcr.fit(X_train_norm)
#transformed = model.transform(X)
features = range(pcr.n_components_)



ax = sns.barplot(x=pd.Series(features)+1, y=pd.Series(pcr.explained_variance_) /␣
 ↪pcr.explained_variance_.sum())
ax.set(xlabel='PCA Component', ylabel='Variance Proportion', title='PCA Variance␣
 ↪Analysis')
sns.set_style("whitegrid")
plt.show()
```



[130]:
```
# plot pca vs share of variance:
var_prop = pcr.explained_variance_ / pcr.explained_variance_.sum()
ax = sns.lineplot(x=pd.Series(features), y=pd.Series(var_prop))
ax.set(xlabel='PCA Component', ylabel='Variance Proportion', title='PCA Variance␣
↪Analysis')
sns.set_style("whitegrid")
plt.show()

# data shows we should use between 2 and 4 PCs so will use 3 here
```



[131]:
```
pca = PCA(n_components=3)
X_train_norm = scaler.fit_transform(X_train.values)
pca.fit_transform(X_train_norm)
linear = pd.DataFrame(pca.components_.T, columns=['PC1', 'PC2', 'PC3'], index=X.
 ↪columns)
linear

#PC1 is generally a size metric, PC2 relates to small hands and short arms, PC3␣
 ↪it almost purely strength
```

[131]:

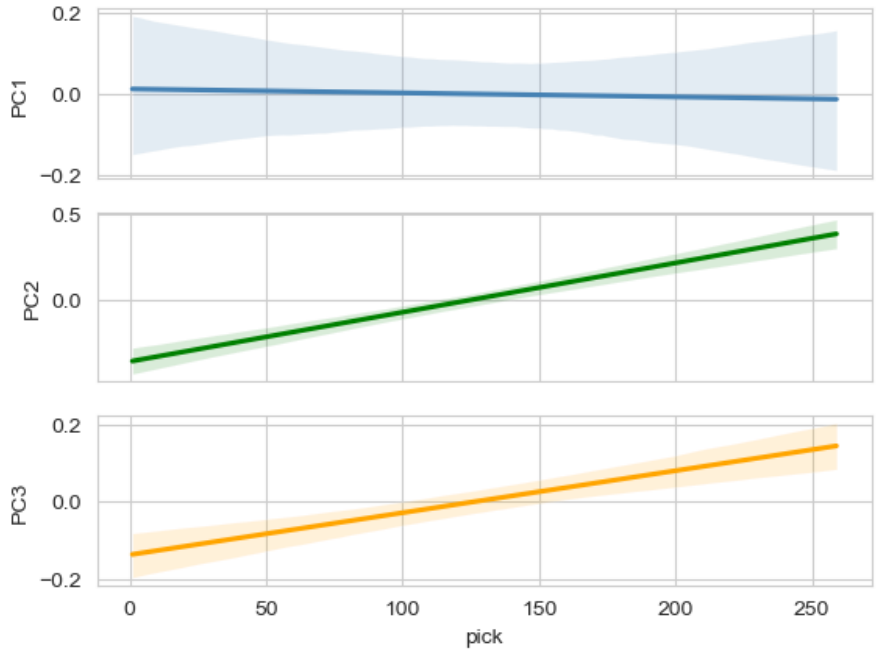
	PC1	PC2	PC3
height	0.304736	-0.405880	-0.107401
weight	0.385545	-0.038562	0.142756
hand_size	0.228303	-0.461385	0.002039
arm_length	0.263038	-0.533021	-0.253794
forty	0.373551	0.149111	-0.022600
bench	0.232707	-0.048905	0.913638
vert	-0.319665	-0.333223	0.195778
broad_jump	-0.324417	-0.386599	0.068543
shuttle	0.342782	0.121846	-0.135350

```
3cone       0.344318  0.191882 -0.084137
```

[133]:
```
# Cross-Validated PC Regression
# scale the test set
pca = PCA(n_components=3)
X_norm = scaler.fit_transform(X.values)

# dim reduce X
X_pca = pca.fit_transform(X_norm)
add_pos = pos_one_hot[['DB', 'DT', 'OL']].values
X_pca2 = np.concatenate([X_pca, add_pos], axis=1)
cv = KFold(n_splits=3, shuffle=True, random_state=100)
reg = LinearRegression()

scores = cross_val_score(reg, X_pca2, y, scoring='neg_mean_squared_error', cv=cv)
rmse = (scores.mean()*-1)**0.5

print(rmse)
```

```
68.05924969610729
```

[134]:
```
# PC scatterplot vs reponse
df_pca = np.concatenate([X_pca, y.values.reshape(-1,1)], axis=1)
df_pca = pd.DataFrame(df_pca, columns=['PC1', 'PC2', 'PC3', 'pick'])

fig, axs = plt.subplots(nrows=3, sharex=True)
sns.regplot(x='pick', y='PC1', data=df_pca, scatter=False, ax=axs[0],␣
 ↪color="steelblue")
axs[0].set_xlabel('')
sns.regplot(x='pick', y='PC2', data=df_pca,scatter=False, ax=axs[1],␣
 ↪color="green")
axs[1].set_xlabel('')
sns.regplot(x='pick', y='PC3', data=df_pca, scatter=False, ax=axs[2],␣
 ↪color="orange")
sns.set_style("whitegrid")
fig.show()

# note how pc1 has little relationship w y and therefore poor mse
```



[86]:
```
################## XG Boost ##################
y = combine_complete['pick']
X = combine_complete.iloc[:, 4:-1]
X = pd.concat([X, pos_one_hot], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
↪random_state=100)

scaler = StandardScaler()
X_norm = scaler.fit_transform(X)
X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.fit_transform(X_test)



n = 1000
max_depth = 1
xgb_reg = xgb.XGBRegressor(objective='reg:squarederror',
                           n_estimators=n,
                           max_depth=max_depth,
                           eta=0.01,
                           seed=100)
```

```
xgb_reg.fit(X_train_norm, y_train)
preds = xgb_reg.predict(X_test_norm)

rmse = mean_squared_error(preds, y_test, squared=False)
print(rmse)
```

```
69.51252126008916
```

[8]:
```
# Tune hyperparams w/ gridsearch
param_grid = {
    'max_depth': [1, 2, 3],
    'eta': [0.001, 0.01, 0.1, 0.25, 0.5],
    'n_estimators': [100, 500, 1000],
}
grid_mse = GridSearchCV(estimator=xgb_reg,
                        param_grid=param_grid,
                        scoring='neg_mean_squared_error',
                        cv=3,
                        verbose=1)
mse_grid = grid_mse.fit(X_norm, y)
```

```
Fitting 3 folds for each of 45 candidates, totalling 135 fits
```

[28]:
```
best_params = grid_mse.best_params_
rmse = np.abs(grid_mse.best_score_)**0.5

print(best_params)
print(rmse)
```

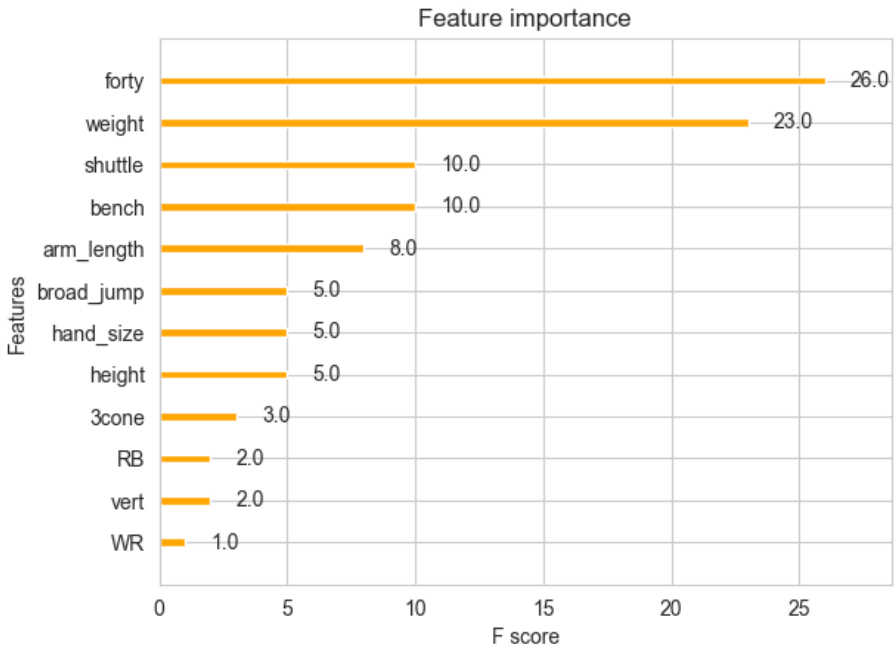
```
{'eta': 0.25, 'max_depth': 1, 'n_estimators': 100}
68.45412247241667
```

[29]:
```
xgb_final = xgb.XGBRegressor(objective='reg:squarederror',
                             n_estimators=100,
                             max_depth=1,
                             eta=0.25,
                             seed=100)
xgb_final.fit(X_norm, y)
```

[29]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eta=0.25, eval_metric=None,
             feature_types=None, gamma=None, gpu_id=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
```

```
max_cat_to_onehot=None, max_delta_step=None, max_depth=1,
max_leaves=None, min_child_weight=None, missing=nan,
monotone_constraints=None, n_estimators=100, n_jobs=None,
num_parallel_tree=None, predictor=None, ...)
```

[90]:
```
# plot feature importance
feature_names = X.columns

ax = xgb.plot_importance(xgb_final, color="orange")
ylabs = list(ax.get_yticklabels())
dict_features = dict(enumerate(feature_names))
ylabs_stripped = [ylabs[i].get_text().lstrip('f') for i in range(len(ylabs))]
ylabs_stripped = [dict_features[int(i)] for i in ylabs_stripped]

ax.set_yticklabels(ylabs_stripped)
plt.show()
```



[ ]:
```
################## Final Cross-Validated RMSE ##################
# Linear: 67.14958397888446
# PCR: 68.68564523905418
# XGBoost: 68.45412247241667
```